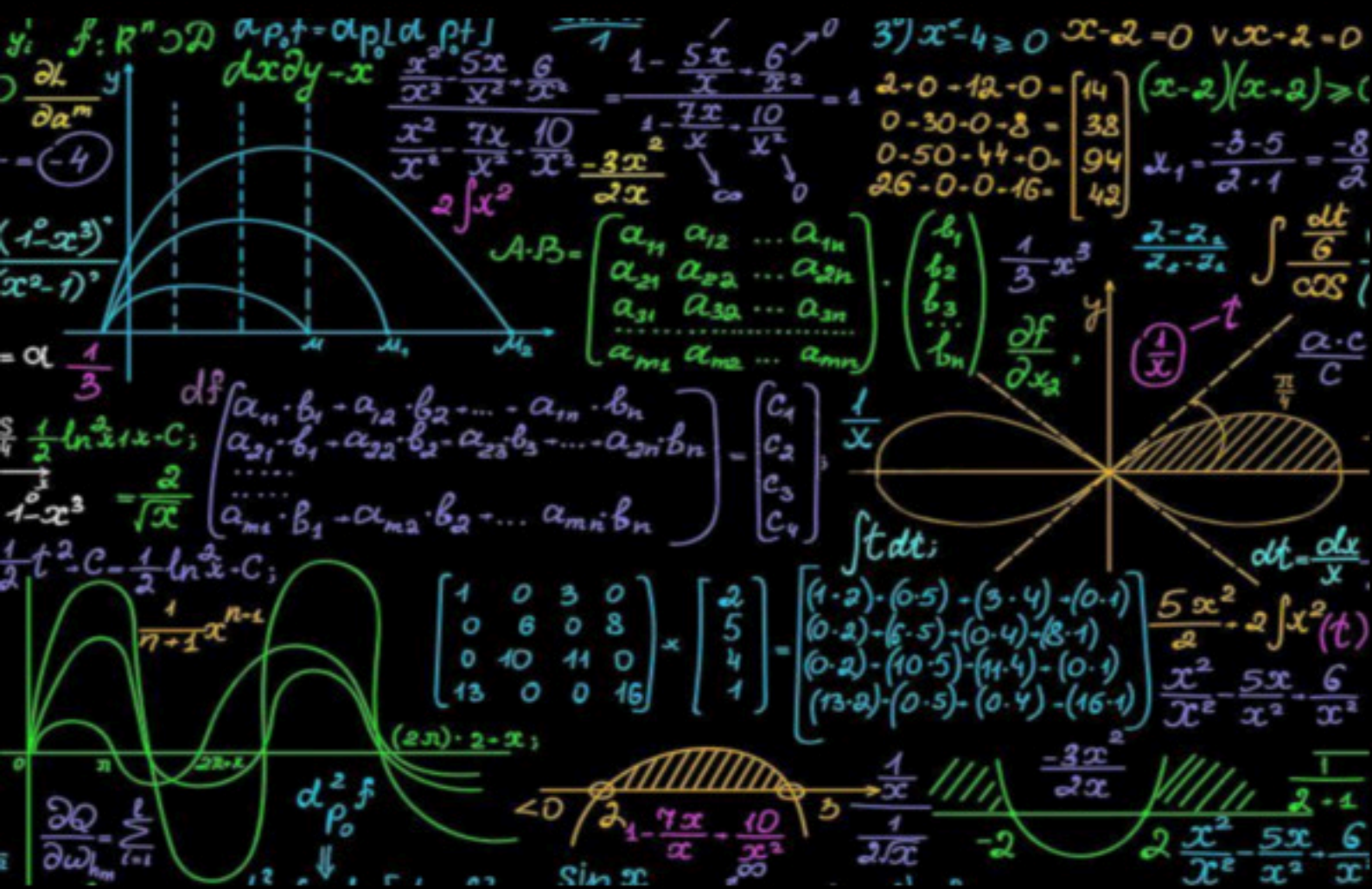
Solución de Grandes Sistemas de Ecuaciones Lineales



Antonio Carrillo Ledesma

Solución de Grandes Sistemas de Ecuaciones Lineales

Antonio Carrillo Ledesma
Facultad de Ciencias, UNAM
http://academicos.fciencias.unam.mx/antoniocarrillo

La última versión de este trabajo se puede descargar de la página:
https://sites.google.com/ciencias.unam.mx/acl/en-desarrollo
http://132.248.181.216/acl/EnDesarrollo.html

2025, Versión 1.0α[1]

Índice

1 Introducción

Para muchas personas en Ciencias e Ingenierías, las matemáticas aplicadas son principalmente física matemática. La física matemática es principalmente ecuaciones diferenciales. La solución numérica de ecuaciones diferenciales se reduce al álgebra lineal. Por lo tanto, el corazón de las matemáticas aplicadas es el álgebra lineal.

Si bien, hay mucho de verdad en el resumen anterior. El álgebra lineal es muy importante y gran parte de las matemáticas aplicadas dependen en última instancia de soluciones eficientes de grandes sistemas lineales. El eslabón más débil del argumento puede ser el primero: las matemáticas aplicadas son mucho más que la física matemática. La física matemática no ha decaído, pero otras áreas han crecido. Aun así, las áreas de las matemáticas aplicadas fuera de la física matemática y fuera de las ecuaciones diferenciales a menudo dependen críticamente del álgebra lineal.

Sin duda recomendaría que alguien interesado en las matemáticas aplicadas se familiarice con el álgebra lineal numérica. Si quieres ser un experto en ecuaciones diferenciales, optimización o en muchos otros campos, necesitas estar al menos familiarizado con el álgebra lineal numérica si vas a calcular algo. Como señala Stephen Boyd en su clase de optimización convexa, muchos de los avances en optimización de los últimos 20 años tienen en su núcleo avances en el álgebra lineal numérica. Los algoritmos mejorados han acelerado la solución de sistemas muy grandes más que la ley de Moore.

Puede parecer cuestionable decir que el álgebra lineal está en el corazón de las matemáticas aplicadas porque es lineal. ¿Qué pasa con las aplicaciones no lineales, como las ecuaciones diferenciales parciales no lineales? Las ecuaciones diferenciales no lineales conducen a ecuaciones algebraicas no lineales cuando se discretizan. Pero estos sistemas no lineales se resuelven mediante iteraciones de sistemas lineales, por lo que volvemos al álgebra lineal.

En el estudio de los sistemas continuos -sistemas físicos macroscópicos-, tales como los yacimientos petroleros, la atmósfera, los campos electromagnéticos, los océanos, el aparato circulatorio de los seres humanos, la corteza terrestre y muchos otros sistemas de interés en Ciencia y en Ingeniería, al modelarse contienen un gran número de grados de libertad[1].

Los sistemas lineales densos más grandes que se están resolviendo en la

[1] El número de grados de libertad en un sistema físico se refiere al número mínimo de números reales que es necesario especificar para determinar completamente el estado físico.

actualidad en un sólo equipo de cómputo son de orden $n = 10^9$ y los futuros sistemas informáticos a exaescala podrán abordar problemas aún mayores. El análisis del error de redondeo muestra que la solución calculada satisface un límite de error hacia atrás por componentes que, bajo supuestos favorables, es de orden $n\mu$, dónde μ es el redondeo unitario de la aritmética de punto flotante: $\mu = 10^{-16}$ para doble precisión y $\mu = 10^{-8}$ para precisión simple. Este límite de error hacia atrás no puede garantizar ninguna estabilidad para la solución de precisión simple de los problemas más grandes de hoy y sugiere una pérdida de la mitad de los dígitos en el error hacia atrás para precisión doble.

La aritmética de punto flotante de media precisión ahora está disponible en Hardware, tanto en el formato IEEE binary16 como en el formato bfloat16, y se utiliza cada vez más en el aprendizaje automático y en el cómputo científico en general. Para el cálculo del producto interno de dos n-vectores el límite de error hacia atrás es nuevamente de orden $n\mu$, y este límite excede 1 por $n > 864$ para ambos formatos de precisión media, lo que sugiere una pérdida potencialmente completa de estabilidad numérica. Sin embargo, productos internos con $n > 864$ se utilizan con éxito en cálculos de precisión media en la práctica.

Los límites de error a los que me he referido son límites superiores y, por lo tanto, limitan el peor de los casos sobre todos los posibles errores de redondeo. Su objetivo principal es revelar posibles inestabilidades en lugar de proporcionar estimaciones de error realistas. Sin embargo, necesitamos conocer los límites de lo que podemos calcular y, para las aplicaciones de misión crítica, debemos poder garantizar un cálculo exitoso.

¿Podemos entender el comportamiento de los algoritmos de álgebra lineal a escala extrema y en aritmética de punto flotante de baja precisión?

En gran medida, la respuesta es sí si aprovechamos tres características diferentes para obtener límites de error más pequeños.

Algoritmos Bloqueados muchos algoritmos se implementan en forma bloqueada. Por ejemplo, un producto interior x^T y de dos n-vectores x y y se puede calcular como

$$\begin{aligned} s_i &= x((i-1)b+1):ib)^T y((i-1)b+1:ib), i = 1:k, \\ s &= s_1 + s_2 + ... + s_k \end{aligned}$$

dónde $n = kb$ y $b \ll n$ es el tamaño del bloque. El producto interior se ha partido en k productos internos más pequeños de tamaño b, que se calculan de

forma independiente y luego se suman. Muchos algoritmos de álgebra lineal se bloquean de forma análoga, donde el bloqueo se realiza en submatrices con b filas o b columnas (o ambas). Un análisis cuidadoso del análisis de errores muestra que un algoritmo bloqueado tiene un límite de error alrededor de un factor de b más pequeño que el del algoritmo desbloqueado correspondiente. Los tamaños de bloque prácticos para algoritmos matriciales suelen ser 128 o 256, por lo que el bloqueo trae una reducción sustancial de los límites de error.

De hecho, uno puede hacerlo incluso mejor que un límite de error de orden $(n/b)\mu$. Calculando la suma $s = s_1 + s_2 + ... + s_k$ con un método de suma más preciso, la constante de error se reduce aún más a $b\mu + O(\mu^2)$ (este es el método FABsum de Blanchard et al. (2020)).

Características Arquitectónicas los procesadores Intel x86 admiten un formato de precisión extendido de 80 bits con un significado de 64 bits, que es compatible con el especificado en el estándar IEEE. Cuando un compilador usa este formato con registros de 80 bits para acumular sumas y productos internos, está trabajando efectivamente con un redondeo unitario de 2^{-64} en vez de 2^{-53} para precisión doble, dando límites de error más pequeños en un factor de hasta $2^{11} = 2048$.

Algunos procesadores Intel y AMD tienen una operación fusionada de multiplicación y suma (FMA), que calcula una multiplicación y una suma combinadas $x + yz$ con un error de redondeo en lugar de dos. Esto da como resultado una reducción en los límites de error por un factor 2.

Las operaciones FMA de bloques de precisión mixta $D = C + AB$, con matrices A, B, C y D de tamaño fijo, están disponibles en las unidades de procesamiento tensorial de Google, las GPU NVIDIA y en la arquitectura ARMv8-A. Para entradas de precisión media, estos dispositivos pueden producir resultados de calidad de precisión simple, lo que puede proporcionar un aumento significativo en la precisión cuando los bloques FMA se encadenan para formar un producto matricial de dimensión arbitraria.

Límites probabilísticos los límites del error de redondeo en el peor de los casos sufren el problema de que no se pueden alcanzar para la mayoría de los conjuntos de datos específicos y es poco probable que se alcancen casi por completo. Stewart (1990) señaló que:

> Para ser realistas, debemos eliminar lo improbable. Lo que

queda es necesariamente un enunciado probabilístico.

Theo Mary y Nick Higham han desarrollado recientemente un análisis de error de redondeo probabilístico, que hace suposiciones probabilísticas sobre los errores de redondeo y deriva límites que se mantienen con cierta probabilidad. La característica clave de los límites es que son proporcionales a $\sqrt{n\mu}$ cuando un límite correspondiente en el peor de los casos es proporcional a $n\mu$. En la forma más general del análisis (Connolly, Higham y Mary, 2021), se supone que los errores de redondeo son independientes de la media y de media cero, donde la independencia media es un supuesto más débil que la independencia.

Poniendo las Piezas Juntas las diferentes características que hemos descrito se pueden combinar para obtener límites de error significativamente más pequeños. Si usamos un algoritmo bloqueado con tamaño de bloque $b \ll n$ luego, en un producto interno, el límite de error estándar de orden $n\mu$ se reduce a un límite probabilístico de orden $\left(\sqrt{n/b}\right)\mu$, que es una reducción significativa. Los bloques FMA y los registros de precisión extendidos proporcionan reducciones adicionales.

Por ejemplo, para un sistema lineal de orden 10^7 resuelto en precisión simple con un tamaño de bloque de 256, el límite de error probabilístico es de orden 10^{-5} versus 1para el límite estándar del peor de los casos. Si se utiliza FABsum, el límite se reduce aún más.

Los autores concluyen que podemos resolver con éxito problemas de álgebra lineal de mayor tamaño y menor precisión de lo que sugiere el análisis de error de redondeo estándar. Sin embargo, los límites a priori siempre serán pesimistas. Se deben calcular los residuos a posteriori o los errores hacia atrás (según el problema) para evaluar la calidad de una solución numérica.

En el caso de los modelos matemáticos de los sistemas continuos, estos son sistemas de ecuaciones diferenciales, las cuales son parciales -con valores iniciales y condiciones de frontera- para casi todos los sistemas de mayor interés en la Ciencia y la Ingeniería. Salvo por los problemas más sencillos, no es posible obtener por métodos analíticos las soluciones de tales ecuaciones, que son las que permiten predecir el comportamiento de los sistemas continuos y realizar las simulaciones requeridas.

La capacidad para formular los modelos matemáticos de sistemas continuos complicados y de gran diversidad, es sin duda una contribución fun-

damental para el avance de la Ciencia y sus aplicaciones, tal contribución quedaría incompleta y, debido a ello, sería poco fecunda, si no se hubiera desarrollado simultáneamente su complemento esencial: los métodos numéricos y la computación electrónica.

Sin embargo, la solución numérica y las simulaciones computacionales de problemas concomitantes en Ciencias e Ingenierías han llevado al límite nuestra actual capacidad de predicción, por la gran cantidad de grados de libertad que necesitan nuestros modelos para tratar de representar a la realidad.

Con el desarrollo de nuevas herramientas numéricas y computacionales, la diversidad y complejidad de problemas que pueden ser tratados de forma satisfactoria y eficiente es impresionante. Pero hay que destacar, que todavía hay una gama de problemas que hasta la fecha no es posible resolver satisfactoriamente o con la precisión deseada -como la predicción climática a largo plazo o simulación de recuperación mejorada en yacimientos petroleros, entre otros-.

La solución numérica de ecuaciones diferenciales parciales por los esquemas tradicionales -tipo Diferencias Finitas (que trataremos brevemente en el apéndice), Volumen Finito y Elemento Finito- reducen el problema a la generación y solución de un -cada vez más grande- sistema algebraico de ecuaciones. La factorización directa de sistemas de gran escala $O(10^6)$ con toda su eficacia, no es, en general, una opción viable, y el uso de métodos iterativos básicos resultan en una convergencia bastante lenta con respecto a otras formas de solución.

El desarrollo de métodos numéricos para sistemas algebraicos grandes no densos, es central en el desarrollo de códigos eficientes para la solución de problemas en Ciencia e Ingeniería, actualmente cuando se necesita implementar una solución computacional, se dispone de bibliotecas optimizadas[2] para la solución de sistemas lineales que pueden correr en ambientes secuenciales y/o paralelos. Estas bibliotecas implementan métodos algebraicos robustos para muchos problemas prácticos, pero sus discretizaciones no pueden ser construidas por sólo técnicas algebraicas simples, tales como aproximaciones a la inversa o factorización incompleta.

En la actualidad, los sistemas computacionales paralelos son ubicuos. En ellos es posible encontrar más de una unidad de procesamiento, conocidas

[2]Como pueden ser las bibliotecas ATLAS (http://math-atlas.sourceforge.net) y HYPRE (https://computation.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods) entre muchas otras.

como Núcleo (Core). El número de Cores es creciente conforme avanza la tecnología, esto tiene una gran importancia en el desarrollo eficiente de algoritmos que resuelvan sistemas algebraicos en implementaciones paralelas. Actualmente la gran mayoría de los algoritmos desarrollados son algoritmos secuenciales y su implantación en equipos paralelos no es óptima, pero es una práctica común usar diversas técnicas de pseudoparalelización -a veces mediante la distribución de una gran matriz en la memoria de los múltiples Cores y otras mediante el uso de directivas de compilación-, pero la eficiencia resultante es pobre y no escalable a equipos masivamente paralelos por la gran cantidad de comunicación involucrada en la solución.

1.1 Computadoras Actuales

La computadora (también conocida como ordenador) actual es una máquina digital programable que ejecuta una serie de comandos para procesar los datos de entrada, obteniendo convenientemente información que posteriormente se envía a las unidades de salida. Una computadora está formada físicamente por numerosos circuitos integrados y varios componentes de apoyo, extensión y accesorios, que en conjunto pueden ejecutar tareas diversas con suma rapidez y bajo el control de un programa (Software).

La constituyen dos partes esenciales, el Hardware, que es su estructura física (circuitos electrónicos, cables, gabinete, teclado, etc.), y el Software, que es su parte intangible (programas, datos, información, documentación, etc.).

Con respecto al Hardware[3], se encuentra compuesto por una serie de dispositivos, clasificados según la función que estos desempeñen. Dicha clasificación se compone de:

[3]En Debian GNU/Linux podemos instalar la aplicación *lshw* para conocer los distintos componentes de la computadora, mediante:

```
# apt install lshw
```

Así, para ver de forma resumida los dispositivos que componen la computadora usamos:

```
# lshw -short
```

Si necesitamos más detalle usamos:

```
# lshw
```

- Los dispositivos de entrada son todos aquellos que permiten la entrada de datos a una computadora. Estos dispositivos (periféricos), son los que permiten al usuario interactuar con la computadora. Ejemplos: teclado, Mouse (ratón), micrófono, Webcam, Scanner, etc.
- Los dispositivos de salida, son todos aquellos que permiten mostrar la información procesada por la computadora. Ejemplos: monitor, impresora, auriculares, altavoces, etc.
- Los dispositivos de comunicación son aquellos que permiten la comunicación entre dos o más computadoras. Ejemplos: Modem, Router, placa de red, Bluetooth, etc.
- Los dispositivos de almacenamiento, son todos aquellos que permiten almacenar datos en el ordenador. Ejemplos: disco duro, Pendrive, Diskette, CD y DVD, etc.
- Los dispositivos de cómputo, son aquellos encargados de realizar las operaciones de control necesarias, sobre el resto de los dispositivos la computadora.

La unidad central de procesamiento (Central Processing Unit CPU) se comunica a través de un conjunto de circuitos o conexiones llamada Bus de datos o canal de datos. El bus conecta la CPU a los dispositivos de almacenamiento, los dispositivos de entrada y los de salida.

En la actualidad, una computadora puede tener una CPU de 32 o de 64[4] bits para describir anchura de registros, Bus de direcciones, Bus de datos o

[4] ¿Por qué se llama x86 a la de 32 bits y x64 a la de 64 bits?

Por pereza. El término adecuado para la arquitectura de 64 bits basada en las tecnologías desarrolladas por Intel y AMD desde 1976 es x86-64, que es como la llamó AMD cuando la desarrolló.

El término "x86" se acuñó porque Intel denominó 8086 a la primera CPU que utilizaba esta microarquitectura. Era una CPU de 16 bits con un conjunto de instrucciones similar al anterior Intel 8085 de 8 bits, por lo que Intel le dio un número más alto. A principios de los años 90, a los desarrollos posteriores de la 8086 se les asignaron números que terminaban en "86": 80186, 80286, 80386 y 80486. En la mayoría de los casos, se omitió la parte "80" y toda la familia de CPU se denominó "x86", por razones obvias. Con lo que habría sido el 80586, Intel eliminó los números porque no podían registrarlos como marca, así que nació la marca "Pentium", que primero significaba quinta generación, pero luego como una simple marca.

Cuando Intel pasó a los 32 bits con el conjunto de instrucciones x86 con el 386 en

instrucciones. Es decir que la cantidad de bits nos permite diferenciar el tipo de CPU que tiene nuestro equipo.

La CPU, el sistema operativo, el Software en general y los Drivers se basan en una misma arquitectura, pero que tenga más o menos bits puede ser significativo en la experiencia del usuario.

En la actualidad, la mayoría de ordenadores cuentan con CPU de 64[5] bits. Ahora bien, todavía hay equipos algo antiguos que cuentan con la arquitectura de 32 bits, por lo que sigue coexistiendo en el mercado y el sector hace desarrollos también pensando en sus capacidades.

Desde el punto de vista funcional es una máquina que posee, al menos,

1985, no cambiaron la marca de nada, pero cuando AMD extendió x86 a 64 bits, lo comercializaron como x86-64. Muchos desarrolladores de Software lo llamaron "AMD64" en su lugar, ya que AMD lo desarrolló. Intel no estaba dispuesto a hacer eso, así que lo llamaron "EMT64" cuando adoptaron las extensiones de 64 bits de AMD después del fracaso de Itanium. Al final, la mayoría de la gente simplemente lo llamó "x64", aunque esto no tiene ningún sentido.

[5] ¿Alguna empresa ha fabricado una CPU de 128 bits?

Si alguien quiere fabricar una CPU de 128 bits, puede utilizar el diseño RISC-V, que admite núcleos de 32, 64 y 128 bits. No hay problema, solo se necesita un diseño adecuado y listo. Muchos dicen que el beneficio de una CPU (núcleo) de 64 bits es la enorme memoria direccionable, 64 bits en comparación con los 32 bits de los núcleos de 32 bits. ¡Pero eso no es cierto!. Todos los núcleos de 64 bits (excepto los muy especiales) tienen un bus de direcciones de 52 o 56 bits. Windows y Linux utilizan internamente un espacio de direccionamiento de a lo más 48 bits. No hay necesidad de memoria física de 64 bits porque hoy en día no es posible tenerla (teóricamente hasta 16 Exabytes).

Algo similar ocurre con los núcleos de 128 bits. Como la memoria no es importante, lo único que queda es la matemática de 128 bits. La pregunta es: ¿quién necesita la matemática de 128 bits? En 1978, el 8086 tenía un coprocesador matemático, el 8087, que admitía tipos de punto flotante de 80 bits. Hoy en día, es similar, pero el tipo máximo admitido sigue siendo de 64 bits que tiene aproximadamente 16 dígitos decimales de precisión con un rango del orden de 1.7×10^{-308} a 1.7×10^{308}, el número de 32 bits que tiene 14 dígitos decimales de precisión con un rango del orden de 3.4×10^{-38} a 3.4×10^{38}. Además, existe el número de 80 bits que tiene 18 dígitos decimales de precisión con un rango del orden de 3.4×10^{-4096} a 1.1×10^{4096}.

En algunos compiladores existen 128 bits de datos, pero no existe matemática de Hardware, sino que se implementa en Software. Se necesitan tipos tan grandes en muy pocas circunstancias.

Lo que importa hoy en día es la paralelización, el procesamiento de múltiples tipos en paralelo. Durante mucho tiempo, x86 ha tenido AVX512, que es un motor matemático de 512 bits (reemplazó al 8087), pero es capaz de trabajar con tipos de 8 a 64 bits en paralelo, por ejemplo, 8 operaciones matemáticas en paralelo utilizando FP64 (punto flotante de 64 bits).

una unidad central de procesamiento, unidad de memoria (Random Access Memory RAM , Read Only Memory ROM y Caché) y dispositivos de entrada/salida (periféricos). Los periféricos de entrada permiten el ingreso de datos, la CPU se encarga de su procesamiento (operaciones aritmético-lógicas) y los dispositivos de salida los comunican a los medios externos. Es así, que la computadora recibe datos, los procesa y emite la información resultante, la que luego puede ser interpretada, almacenada, transmitida a otra máquina o dispositivo o sencillamente impresa; todo ello a criterio de un operador o usuario y bajo el control de un programa de computación.

CPU la Unidad Central de Proceso (Central Processing Unit CPU) es aquella parte del procesador que se encarga de ejecutar las diversas acciones que ordenemos al dispositivo que debe llevar a cabo. La CPU es el componente básico dentro de todo dispositivo inteligente, ya que prácticamente cualquier proceso que se ordene al sistema pasa por él. Con el paso del tiempo, además, su eficiencia y calidad ha alcanzado grandes cotas, aunque tecnologías al alza como las NPU han supuesto un mayor salto cualitativo que el de aumentar la potencia bruta de la CPU.

GPU, para el procesamiento gráfico la Unidad de Procesamiento Gráfico (Graphics Processing Unit GPU) es el apartado que se dedica a las acciones de mayor peso: las de componente gráfico. De este modo, acciones como la ejecución de videojuegos, o la edición y renderizado de vídeos, se llevan a cabo a través de la GPU del sistema. La calidad del teléfono, ordenador u otro dispositivo inteligente, suele ir supeditada a menudo a la calidad de su GPU, que dependerá de la banda de precio del aparato en cuestión. Eso sí, si los otros componentes no tienen la misma calidad, se puede producir el temido cuello de botella.

En los smartphones, la GPU ya va integrada en los procesadores, pero en los PC's, como AMD y NVIDIA se muestran como marcas especializadas en las GPUs. Cuentan con todo tipo de gamas y también poseen un amplio abanico de precios, un valor siempre directamente proporcional a la calidad y capacidad de sus gráficas, y a su vigencia en el mercado.

NPU, redes neuronales en tu dispositivo la Unidad de Procesamiento Neuronal (Neural Processing Unit NPU), a diferencia de la GPU, que cuenta con un funcionamiento paralelo al de la CPU, puede encargarse de

funciones similares a las de la CPU, pero lo hace de un modo mucho más eficiente. Impulsada por Inteligencia Artificial, una NPU[6] es capaz de priorizar procesos para ejecutarlos de un modo exponencialmente más veloz y con un consumo mucho menor. En móviles, se usa especialmente para mejorar el procesado de fotografías, aunque participa en muchos otros procesos.

Además, esta arquitectura todavía tiene años de progreso por delante, a diferencia de la CPU y la GPU, cuyas mejoras ya son de carácter más leve y basadas en aumentar la potencia bruta. La tecnología NPU, en cambio, lleva menos tiempo entre nosotros y todavía tiene mucho margen de mejora para ofrecer un rendimiento cada vez más poderoso.

Desde la llegada de los procesadores de más de un núcleo a PC, como consecuencia de la imposibilidad de hacerlos escalar en potencia solo por velocidad de reloj, la forma de entender los diferentes chips cambió. Han pasado ya dos décadas de dicho cambio y ante la inminente salida de los Chips disgregados o por Chiplets al mercado de masas, no está de más recordar la organización más común en el mundo del Hardware en todo este tiempo.

¿Qué es un SoC? las siglas SoC significan System on a Chip y hace referencia a todo chip que tiene la mayoría de componentes integrados en una misma pieza de silicio sin llegar a ser un microcontrolador. Se trata de la pieza de Hardware más usada por el hecho de que a día de hoy todo procesador para PC, teléfono móvil, consola de videojuegos, televisor o incluso servidores, es un SoC y pese a las diferencias entre ellos, todos tienen una organización común.

En realidad, todas las CPU actuales son SoC, ya que se trata de "varias CPU" diseñadas para funcionar alrededor de un elemento de intercomunicación central. Este se encarga de interconectar los diferentes elementos entre sí y de darles acceso a interfaces externas.

Por ejemplo, con la memoria RAM (u otros), a la que está asociado el controlador de memoria que comparten todos sus elementos o a los periféricos, y a los cuales se puede acceder directamente con una serie de interfaces específicas, o a través de un Chipset externo encargado de gestionar los diferentes periféricos y componentes.

[6]Para muestra, la compañía Cerebras con su procesador WSE-3 aglutina 4 billones de transistores, tiene una superficie de 46,225 mm^2, integra nada menos que 900,000 núcleos optimizados para IA y es capaz de entrenar hasta 24,000 millones de parámetros, lo que también equivaldría a un rendimiento máximo de IA de 125 petaflops.

En PC, debido a que la comunicación con los periféricos se hace a través del uso de direcciones de memoria de la RAM principal, los componentes relacionados con esta se encuentran subordinados al controlador de memoria. Por lo que son una pieza más conectada a la parte central.

¿Qué es una APU y en qué se diferencia de un SoC? las siglas APU significan Accelerated Processor Unit y fue usada por AMD cuando sus CPU e iGPU (o GPU integrada, la veremos a continuación) no traían consigo ningún sistema de gestión de E/S (entrada y salida). Sin embargo, la cosa empezó a cambiar ya con la arquitectura "Carrizo" que fue el nombre clave de los últimos SoC antes del lanzamiento de los AMD Ryzen, donde se incorporaron varias interfaces de periféricos directamente en la CPU.

A día de hoy todo es un SoC, lo que ocurre es que, en sobremesa, las torres tienen tanta conectividad y capacidad de expansión que se suele emplear un Chipset, y lo mismo ocurre en estaciones de trabajo y servidores, pero no en el resto. Y es que si hablamos de un Chip para un PC portátil, una consola o un móvil, entonces al no existir tantas interfaces para periféricos y otros componentes, entonces estás se pueden integrar en un mismo chip.

En realidad, el término APU es más bien comercial de AMD y al día de hoy se utiliza como sinónimo de SoC, pero se puede resumir en que una APU carece de cualquier gestión de periféricos y requiere de un Chip externo para ello. Mientras que un SoC tiene las especificaciones mínimas en ese aspecto, pese a ser también ampliables. Para simplificar la idea, un SoC es una APU más completa.

¿¡GPU qué es y cuáles son sus características concretas frente a una dGPU? las siglas iGPU corresponden a integrated GPU, y hace referencia a todo componente de este tipo que se encuentre integrado en una APU o un SoC. Por lo que se trata de procesadores gráficos de potencia limitada que se pueden ver lastrados en velocidad de reloj por el problema del ahogamiento termal (Thermal Throttling) que se produce cuando muchas partes comparten el mismo espacio físico.

Si bien, es posible llegar a ciertos niveles de rendimiento que son aceptables de cara a reproducir las escenas en 3D a tiempo real en los videojuegos, y otras tareas de carácter profesional donde se usa una tarjeta gráfica, estas se ven cuanto menos limitadas:

El hecho de tener que compartir espacio en el mismo Chip con los diferen-

tes núcleos de la CPU produce que esta no pueda alcanzar la misma velocidad de reloj que se alcanzaría siendo un Chip aparte e independiente.

En PC, debido a que como la memoria RAM usa DDR o LPDDR, el ancho de banda es pequeño y hemos de partir del hecho de que el rendimiento de todo Chip gráfico, incluido una iGPU, depende del ancho de banda que se le puede otorgar con dicha memoria RAM.

Los SoC con iGPU actuales tienen un tamaño fijo, definido este por la cantidad de pines soportados por la interfaz con la placa base. Esto limita el tamaño, no solo del Chip, sino también de la gráfica integrada en el mismo.

iGPU en consolas de videojuegos al contrario de lo que ocurre con los PC, las consolas de videojuegos no tienen que seguir una serie de normas respecto a sus componentes. Para empezar, no ven el tamaño de sus chips limitados por un estándar de placa base, dado que son productos únicos y exclusivos. Esto les permite tener el tamaño que quieran, incluso más que una CPU convencional para PC, lo que les permite tener una iGPU en su SoC mucho más avanzada.

El otro punto es la memoria utilizada, ya que la de las tarjetas gráficas también se usan como memoria principal. Esta da el ancho de banda necesario para que la iGPU alcance cierto nivel de rendimiento, pero su latencia es mucho más alta que la RAM convencional de PC que está más optimizada en ese aspecto, por lo que el rendimiento de su CPU suele ser más bajo que su equivalente para ordenador.

¿Qué es una dGPU en un PC o portátil? las dGPU, o GPU dedicadas, no son otra cosa que las GPU de toda la vida, pero la particularidad es que también son SoC, ya que tiene varios núcleos, especializados en tareas gráficas, alrededor de una interfaz central y compartiendo todos ellos un mismo acceso a memoria.

Sin embargo, carecen de núcleos de CPU en su interior, de ahí a que no se les llame APU o SoC, por el hecho de que de existir estos elementos pasarían a ser una iGPU. Por lo tanto, su principal particularidad de las dGPU es que tienen su propia memoria RAM (SDRAM para ser concretos) la cual suele rodear estos Chips, ya sea en forma de tarjeta gráfica o soldados en la placa de los ordenadores portátiles. A esta la llamamos VRAM y es de uso exclusivo de la dGPU o GPU.

Los equipos de cómputo los podemos clasificar[7] por:

- Equipos móviles: estos equipos buscan un equilibrio entre su capacidad de cómputo versus el rendimiento energético de sus baterías -para operar el mayor tiempo posible sin recargarse- y su peso, entre estos equipos destacan las Laptops, Notebook, Netbook, Ultrabook, tabletas, teléfonos inteligentes, etc.

- Equipos de escritorio: estos equipos al estar permanentemente conectados a la corriente eléctrica pueden tener un mayor número de componentes y disponen de una mejor capacidad disipación de calor por lo que pueden contener una mayor cantidad componentes, como discos, RAM o tarjetas de video y el tamaño, peso o consumo energético no es un inconveniente.

- Servidores: son equipos que suelen atender a múltiples usuarios simultáneamente y disponen de gran cantidad de Cores, RAM, disco y son interconectados por red de alta velocidad con otros servidores para atender las crecientes necesidades de los centros de datos los cuales deben estar permanentemente en operación. Generalmente los equipos son montados en Racks con otras decenas de equipos, por lo que su arquitectura se ve limitada a una moderada generación de calor por parte de sus componentes ya que su sistema de ventilación es por aire para todo el Rack.

- Estaciones de trabajo: son equipos individuales diseñados para atender cargas computacionales intensas, por lo que requieren Hardware más complejo y potente como puede ser múltiples tarjetas de video (con decenas de miles de Cores gráficos) , discos (con cientos de Terabytes), gran cantidad de RAM (pueden llegar a superar el Terabyte) y sistema de refrigeración por aire o líquido, etc.

- Cómputo intensivo: son equipos interconectados por red de alta velocidad con procesadores y tarjetas gráficas dedicadas para el cálculo numérico que soportan cargas intensas por largos periodos de tiempo,

[7] El ordenador del Apollo 11, el Block II, funcionaba a una velocidad de 2 MHz y tenía 2 KB de memoria RAM y 32 KB de memoria ROM. Para ponerlo en contexto, el chip de un cargador USB-C moderno es 563 veces más potente que la computadora que se usó en el Apollo 11, al menos en términos de potencia bruta.

los más comunes son los que forman parte de los Cluster que llegan a tener millones de cores.

FLOPS Una medida relativamente objetiva para analizar el rendimiento de un dispositivo suele ser medir sus operaciones de punto flotante por segundo o más conocidas como FLOPS[8]. Hay que tener en cuenta que la medición de FLOPS es muy compleja porque las diferentes operaciones en punto flotante llevan diferentes cantidades de tiempo para ejecutarse. Y no todo el mundo utiliza las mismas operaciones para establecer los cálculos.

Por ejemplo, una división simple como 1/5, toma significativamente menos tiempo que el cálculo del logaritmo de 5. Por eso, se estableció el algoritmo de Linpack como un estándar representativo con el que poder medir todos los sistemas bajo el mismo baremo de FLOPS-.

Es importante señalar que el algoritmo de Linpack utiliza el formato en punto flotante de doble precisión (64 bits[9] que tiene aproximadamente 16 dígitos decimales de precisión con un rango del orden de 1.7×10^{-308} a 1.7×10^{308}). Sin embargo, como veremos la mayoría de los valores que dan los fabricantes son con precisión simple (32 bits que tiene 14 dígitos decimales de precisión con un rango del orden de 3.4×10^{-38} a 3.4×10^{38}). Además,

[8] El Cray-1 fue puesto marcha en 1975 y utilizaba una CPU a 80 MHz y llevaba integrada una unidad SIMD de 64 bits de precisión de punto flotante, lo cual fue un salto de gigante que permitió un salto de los 3 MFLOPS de potencia del CDC 6600 a los 160 MFLOPS en el Cray-1.

[9] ¿Por qué no hay un procesador de 128 bits, si los hay de 64 y 32 bits?

En gran medida, esto se reduce a la cuestión de qué se entiende realmente por "procesador de 64 bits". Al inicio de las computadoras electrónicas, un "procesador de 8 bits" tenía registros de 8 bits, pero la mayoría tenía direccionamiento de 16 bits (y la Unidad de Aritmética y Lógica ALU a veces tenía 4 bits, por lo que cuando se añadían dos registros de 8 bits, la CPU a menudo lo hacía en dos pasos, añadiendo los 4 bits inferiores y luego añadiendo por separado los 4 bits superiores más el acarreo de los 4 inferiores).

Hoy en día, hemos invertido un poco esa situación: un procesador de "64 bits" (más o menos) tiene direccionamiento de 64 bits, pero tiene registros de 128 bits, 256 bits y, en algunos casos, incluso de 512 bits. Pero como los usos de números mayores a 64 bits son bastante raros, esos registros más grandes se usan normalmente para realizar operaciones en una cantidad de operandos más pequeños con una sola instrucción. Por lo tanto, en lugar de una ALU que tiene la mitad del tamaño de un registro y realiza una sola operación en múltiples ciclos de reloj, tenemos una ALU que tiene entre 2 y 4 veces el tamaño de un registro normal y realiza múltiples operaciones en un solo ciclo de reloj.

Dependiendo de cómo prefieras ver las cosas, podrías argumentar razonablemente que una CPU actual de "64 bits" es en realidad una CPU de 128 bits, 256 bits o incluso 512 bits.

los valores que dan los fabricantes suelen ser teóricos y en la práctica suelen ser inferiores debido a otros factores limitantes como la frecuencia de reloj o la velocidad de las memorias ROM y RAM.

Por tanto, aunque todos hemos acabado midiendo el rendimiento en FLOPS, no es una medida absoluta de la potencia del CPU ni de una GPU. Por ejemplo para algunos dispositivos tenemos:

Móviles El SoC Snapdragon 821 que monta una GPU Adreno 530 tiene una potencia de 519.2 gigaFLOPS (0.52 TFLOPS), y los Chips Apple A9X del iPad Pro alcanzan los 345.6 gigaFLOPS (0.35 TFLOPS), todos ellos medidos con precisión simple de 32-bits.

CPU

- Intel Xeon W-3245: 1.4 TFLOPS
- Intel Core i9-9900X: 1.2 TFLOPS
- AMD Ryzen 9 3950X: 1.1 TFLOPS

Los procesadores de gama media-alta rondan el medio TFLOPS:

- AMD Ryzen 7 3700X: 546.0 GFLOPS - 0.55 TFLOPS
- Intel Core i9-9900: 499.0 GFLOPS - 0.50 TFLOPS
- AMD Ryzen 5 3600X: 461.0 GFLOPS - 0.46 TFLOPS

Tarjetas Gráficas Ojo: La tabla está ordenada por los valores en precisión simple (32-bit) primer columna

GPU	FP32 TFLOPS	FP64 TFLOPS
TITAN V	13.8	6.9
Radeon RX Vega 64	12.7	0.8
GeForce GTX 1080 Ti	11.3	0.4
GeForce GTX 1080	8.9	0.3
Radeon R9 Fury X	8.6	0.5
Radeon HD 7990	7.8	1.9
GeForce GTX 1070	6.5	0.2
Radeon RX 480	5.8	0.4

GeForce GTX 690	5.6	0.2
Radeon R9 290X	5.6	0.7
GeForce GTX 780 Ti	5.3	0.2
Radeon HD 6990	5.1	1.3
GeForce GTX 980	4.9	0.15
Radeon RX 470	4.9	0.3
Radeon R9 290	4.8	0.6
GeForce GTX Titan	4.7	1.5
GeForce GTX 1060	4.4	0.14
Radeon HD 7970 GHz	4.3	1.1
GeForce GTX 780	4.1	0.17
Radeon R9 280X	4.0	1.0
Radeon R9 280	3.3	0.83
GeForce GTX 680	3.1	0.13
Radeon HD 7950	2.9	0.71

Como podemos ver, las tarjetas gráficas de Nvidia, normalmente, tienen una potencia muy alta en precisión simple, pero muy mala en precisión doble. La precisión simple es la que se usa en los juegos, pero la precisión doble es la que se utiliza en los cálculos complejos científicos y en el minado de muchas criptomonedas.

Consolas Todas ellos son en valores de precisión simple (32-bit)

- PlayStation 4: 1.3 TFLOPS
- Xbox One: 1.8 TFLOPS
- PlayStation 4 Pro: 4.2 TFLOPS
- Nintendo Switch: entre 0.4 y 0.5 TFLOPS
- PlayStation 5 promete una GPU con 10.28 TFLOPS
- La Xbox Series X promete una GPU de 12 TFLOPS

SuperCómputo Hace pocos se publicó la 64^a edición del ranking «Top 500» (noviembre del 2024) que clasifica los Clusters con mayor rendimiento del mundo. Esta lista se basa en la medida del rendimiento de los sistemas

utilizando la prueba de referencia LINPACK, que calcula la velocidad a la que un sistema puede resolver un conjunto de ecuaciones lineales.

Y en esta nueva edición, los Clusters que ocuparon los primeros tres lugares son:

- El sistema El Capitan del Laboratorio Nacional Lawrence Livermore, en California, EE.UU., es el nuevo sistema número 1 del TOP500. El sistema HPE Cray EX255a ha obtenido 1.742 exaflop/s en el benchmark HPL. El Capitan tiene 11,039,616 núcleos y está basado en procesadores AMD EPYC(TM) de 4^a generación con 24 núcleos a 1.8 GHz y aceleradores AMD Instinct(TM) MI300A. Utiliza la red Cray Slingshot 11 para la transferencia de datos y alcanza una eficiencia energética de 60.3 Gigaflops/vatio.

- Frontier es ahora el sistema número 2 del TOP500. Este sistema HPE Cray EX fue el primer sistema estadounidense con un rendimiento superior a un exaflop/s. Está instalado en el Laboratorio Nacional Oak Ridge (ORNL) en Tennessee, EE.UU., donde se utiliza para el Departamento de Energía (DOE). Actualmente ha alcanzado 1.353 Exaflop/s utilizando 8,699,904 núcleos. La arquitectura HPE Cray EX combina CPUs AMD EPYC(TM) de tercera generación optimizadas para HPC e IA, con aceleradores AMD Instinct(TM) 250X y una interconexión Slingshot-11.

- Aurora es actualmente el número3 con una puntuación HPL preliminar de 1.012 Exaflop/s. Está instalado en Argonne Leadership Computing Facility, Illinois, EE. UU., donde también se opera para el Departamento de Energía (DOE). Este nuevo sistema de Intel se basa en HPE Cray EX - Intel Exascale Compute Blades. Utiliza procesadores Intel Xeon CPU Max Series, aceleradores Intel Data Center GPU Max Series y una interconexión Slingshot-11.

Para poner en contexto los avances en este campo, en el año 2004 IBM era dueña y señora del mundo de la supercomputación, su espectacular BlueGene/L dominaba la lista TOP 500. Aquel monstruo contaba con 32,768 procesadores PowerPC 440 a 700 MHz y 16 TB de memoria. 20 años después una sola NVIDIA GeForce RTX 4090 con 24 GB de memoria GDDR6X es más potente que esa supercomputadora -lo es al menos en rendimiento

bruto-, BlueGene/L contaba en ese momento con un rendimiento de 70.72 TFLOPS, pero la propia NVIDIA dejaba claro en el lanzamiento de sus RTX 4090 que estas tarjetas gráficas contaban con una potencia de 83 TFLOPS.

Es más, cuatro RTX 4090 con soporte FP8 logran también rivalizar con la supercomputadora más potente de 2009. Y eso sin apretarle las tuercas a las RTX 4090: en noviembre de 2022 es precisamente lo que hicieron en Wccftech y lograron que la RTX 4090 se convirtiera en la primera tarjeta gráfica del mundo en alcanzar los 100 TFLOPS.

Esa comparación es como decimos real en esa potencia de cálculo en bruto, pero también es cierto que en esa y otras supercomputadoras se tenian mecanismos especiales de comunicación entre procesadores o de transferencia de datos, algo para lo que las GPUs actuales, aún siendo sobresalientes, no están tan optimizadas.

¿Cómo Trabaja una Computadora? Todas las computadoras sean de uno o más procesadores ejecutan los programas realizando los siguientes pasos:

1. Se lee una instrucción
2. Se decodifica la instrucción
3. Se encuentra cualquier dato asociado que sea necesario para procesar la instrucción
4. Se procesa la instrucción
5. Se escriben los resultados

Esta serie de pasos, simple en apariencia, se complican debido a la jerarquía de memoria RAM, en la que se incluye la memoria Caché, la memoria principal y el almacenamiento no volátil como pueden ser los discos duros o de estado sólido (donde se almacenan las instrucciones y los datos del programa), que son más lentos que el procesador en sí mismo. Con mucha frecuencia, el paso (3) origina un retardo muy largo (en términos de ciclos del procesador) mientras los datos llegan en el bus de la computadora.

Durante muchos años, una de las metas principales del diseño microinformático ha sido la de ejecutar el mayor número posible de instrucciones en paralelo, aumentando así la velocidad efectiva de ejecución de un programa.

No obstante, estas técnicas han podido implementarse en Chips semiconductores cada vez más pequeños a medida que la fabricación de estos fue progresando y avanzando, lo que ha abaratado notablemente su costo.

El procesador es el cerebro de un ordenador. No hay que olvidar otros componentes como la memoria, el almacenamiento o la tarjeta gráfica dedicada, desde luego, pero el procesador está un escalafón por encima en la jerarquía.

Piensa que, si cambiamos el procesador en dos equipos con la misma memoria, almacenamiento o tarjeta gráfica, el comportamiento puede variar notablemente. Sin embargo, para un mismo procesador, los cambios en el resto de componentes no impactan de forma tan directa en la experiencia de uso de un equipo.

¿Qué es una CPU? Antes de nada, vamos a definir exactamente lo que es una CPU o un procesador. Como bien indican sus siglas en inglés (Central Processing Unit) es la unidad de procesamiento -puede ser Intel, AMD, ARM, etc- encargada de interpretar las instrucciones de un Hardware haciendo uso de distintas operaciones aritméticas y matemáticas. Características principales de un procesador:

- Frecuencia de reloj. Este primer término hace referencia a la velocidad de reloj que hay dentro del propio procesador. Es un valor que se mide en Mhz o Ghz y es básicamente la cantidad de potencia que alberga la CPU. La mayoría de ellas cuentan con una frecuencia base -para tareas básicas- y otra turbo que se utiliza para procesos más exigentes -con un aumento en el consumo de energía y por ende un aumento en la temperatura del procesador, requiriendo sistemas de disipación de calor eficientes-.

- Consumo energético. Es normal que nos encontremos con CPU 's donde su consumo energético varía notablemente. Es un valor que se muestra en vatios (W) y como es obvio, aquellos procesadores de gama superior, serán más propensos a consumir más energía. Ante esto, es importante contar con un eficiente sistema de enfriamiento además de contar con una fuente de alimentación acorde a la potencia requerida por el procesador, la tarjeta gráfica y sus respectivos sistemas de enfriamiento.

- Número de núcleos. Con el avance de la tecnología, ya es posible encontrar tanto procesadores de Intel como de AMD que cuentan ya con

decenas de núcleos. Estos cores son los encargados de llevar a cabo multitud de tareas de manera simultánea.

- Número de hilos. Si un procesador tiene Hyperthreading en el caso de Intel o SMT (Simultaneous Multi-Threading) en el caso de AMD, significa que cada uno de los núcleos es capaz de realizar dos tareas de manera simultánea, lo que se conoce como hilos de proceso. Por lo tanto, un procesador de cuatro núcleos físicos con Hyperthreading tendría ocho hilos de proceso, y sería capaz de ejecutar ocho órdenes al mismo tiempo -los hilos no tienen las mismas capacidades de un core real y en muchos casos su uso merma el rendimiento del CPU, pero los sistemas operativos los reconocen como si fueran cores reales-[10].

- Memoria Caché. A la hora de "recordar" cualquier tarea, el propio ordenador hace uso de la memoria RAM. Sin embargo no es eficiente este proceso y por tanto es necesario que utilice la memoria Caché de la CPU para paliar esta deficiencia. El Caché se caracteriza porque se llega a ella de forma más rápida y puede ser tipo L1, L2 y L3.

- Zócalo. Es el tipo de conector con pines o Socket al que se conecta la placa base. Por ejemplo, las últimas de Intel suelen tener el Socket LGA 1200, mientras que las de AMD con Ryzen son AM4.

- Red. Si bien la red es un recurso indispensable en un equipo de cómputo, en el caso de equipos paralelos la velocidad de la red es el mayor cuello de botella en cuanto a rendimiento, por ello es necesario usar redes de alto desempeño como las de InfiniBand con un alto costo económico pero de alto desempeño que pueden llegar al orden de cientos de Gigabytes por segundo.

Nuevos Procesadores la creciente demanda de dispositivos de cómputo ha generado una gran variedad de procesadores, los podemos clasificar como:

- Procesador compuesto por múltiples núcleos de alta eficiencia -con un consumo energético reducido- que sacrifican potencia de procesamiento

[10] El AMD EPYC 9845 de 160 núcleos y 320 hilos a una frecuencia de 2,00 GHz basada en Zen 5c, este se acompaña de 640 MB de caché L3.

en aras de extender la carga útil de las baterías de los dispositivos móviles.

- Procesador compuesto por múltiples núcleos de alto rendimiento que pueden estar al tope de su capacidad sin generar excesivo calor y son especialmente usados en servidores y en cómputo intensivo.

- Procesadores compuestos por múltiples núcleos de alto rendimiento que pueden ajustar su velocidad de reloj de manera dinámica para tratar cargas de trabajo pesadas por un cierto tiempo -pues generan gran cantidad de calor-, por lo que requieren un sistema eficiente de enfriamiento, son ideales para estaciones de trabajo.

- Procesadores compuestos por múltiples núcleos híbridos que en lugar de tener un único tipo de núcleo multipropósito, estos Chips cuentan con dos grupos de núcleos. El primero de ellos, compuesto por múltiples núcleos de alta eficiencia, se encarga de procesar las tareas más livianas o en segundo plano que deba realizar un procesador, todo ello, con un consumo energético menor. El otro grupo, compuesto por múltiples núcleos de alto rendimiento, sigue una dinámica opuesta, su consumo ener-gético es superior, pero únicamente entran en funcionamiento cuando la tarea en cuestión requiere un extra de procesamiento.

Para gestionar esta división de núcleos híbridos, se ha integrado un "Thread Director", un elemento que se encarga de determinar qué núcleo procesa cada tarea. Las compañías, además, ha modificado cómo funciona la caché de sus procesadores:

- Cada núcleo de rendimiento tiene su propia caché L2.

- Cada cluster de núcleos de eficiencia tiene una "piscina" de memoria L2 común, de la que beben todos los núcleos que sean partícipes.

- Tanto los núcleos de rendimiento como los de eficiencia tienen acceso a una "piscina" de memoria L3 común para todos ellos.

Otros de los cambios que impactarán en el desempeño de las CPUs es el aumento de velocidad y una mayor cantidad de memoria Caché, compatibilidad con memorias DDR6 y con la interfaz PCIe 6.0.

PCIe PCI Express (Peripheral Component Interconnect Express), abreviado como PCIe, es una tecnología de conexión de Hardware utilizada para la comunicación de alta velocidad entre diferentes componentes de un equipo informático. Este estándar se ha convertido en la interfaz más habitual para la conexión de tarjetas de expansión, como tarjetas gráficas que sirven para correr juegos, tarjetas de sonido, tarjetas de red y dispositivos de almacenamiento de alta velocidad.

Una de las ventajas destacadas de PCI Express es su arquitectura de canales independientes, que permiten la transferencia simultánea de datos en ambos sentidos. Cada carril tiene una tasa de transferencia específica, medida en gigabits por segundo (Gbps), y la capacidad de un slot PCIe se expresa como el número de carriles que tiene. Esto se traduce en un ancho de banda total mayor, lo que facilita la conexión de dispositivos que requieren altas tasas de transferencia, como las tarjetas gráficas modernas o los dispositivos de almacenamiento de última generación.

Los diferentes tipos de ranuras de PCI Express según su tamaño PCIe X1 carriles 1, pines 18, PCIe x4 carriles 4, pines 32, PCIe x8 carriles 8, pines 49, PCIe x16 Carriles 16, pines 82.

Adicionalmente, también es interesante fijarse en las diferentes versiones que se han ido lanzando desde que PCIe se lanzó al mercado:

- PCIe 1.0 ancho banda 8 GB/s, velocidad de transferencia 2.5 GT/s
- PCIe 2.0 ancho banda 16 GB/s, velocidad de transferencia 5 GT/s
- PCIe 3.0 ancho banda 32 GB/s, velocidad de transferencia 8 GT/s
- PCIe 4.0 ancho banda 64 GB/s, velocidad de transferencia 16 GT/s
- PCIe 5.0 ancho banda 128 GB/s, velocidad de transferencia 32 GT/s
- PCIe 6.0 ancho banda 256 GB/s, velocidad de transferencia 64 GT/s

Características Arquitectónicas los procesadores Intel x86 admiten un formato de precisión extendido de 80 bits con un significado de 64 bits, que es compatible con el especificado en el estándar IEEE. Cuando un compilador usa este formato con registros de 80 bits para acumular sumas y productos internos, está trabajando efectivamente con un redondeo unitario de 2^{-64} en vez de 2^{-53} para precisión doble, dando límites de error más pequeños en un factor de hasta $2^{11} = 2048$.

Algunos procesadores Intel y AMD tienen una operación fusionada de multiplicación y suma (FMA), que calcula una multiplicación y una suma combinadas $x + yz$ con un error de redondeo en lugar de dos. Esto da como resultado una reducción en los límites de error por un factor 2.

Las operaciones FMA de bloques de precisión mixta $D = C + AB$, con matrices A, B, C y D de tamaño fijo, están disponibles en las unidades de procesamiento tensorial de Google, las GPU NVIDIA y en la arquitectura ARMv8-A. Para entradas de precisión media, estos dispositivos pueden producir resultados de calidad de precisión simple, lo que puede proporcionar un aumento significativo en la precisión cuando los bloques FMA se encadenan para formar un producto matricial de dimensión arbitraria.

Meltdown y Spectre El tres de enero del 2018 se dio a conocer al público, que 6 meses antes se habían detectado dos distintos fallos en los procesadores de los equipos de cómputo, comunicaciones y redes de internet que usamos. Esto para dar tiempo a los desarrolladores de procesadores y de sistemas operativos de implementar estrategias para mitigar el problema. Estos son problemas de diseño de los procesadores de Intel, AMD, IBM POWER y ARM, esto significa que procesos con privilegios bajos -aquellos que lanzan las aplicaciones de usuarios convencionales- podían acceder a la memoria del Kernel del sistema operativo[11].

Un ataque que explota dicho problema permitiría a un Software malicioso espiar lo que están haciendo otros procesos y también espiar los datos que están en esa memoria en el equipo de cómputo (o dispositivo móvil) atacado. En máquinas y servidores multiusuario, un proceso en una máquina virtual podría indagar en los datos de los procesos de otros procesos en ese servidor compartido.

Ese primer problema, es en realidad solo parte del desastre. Los datos actuales provienen especialmente de un grupo de investigadores de seguridad formados por expertos del llamado Project Zero[12] de Google. Ellos han publicado los detalles de dos ataques (no son los únicos[13]) basados en estos

[11] En GNU/Linux, el Kernel (si usamos una versión actualizada) nos indica las fallas del procesador a las que es vulnerable, usando:

```
$ cat /proc/cpuinfo
$ lscpu
```

[12] https://googleprojectzero.blogspot.com/

[13] Entre las distintas vulnerabilidades detectadas y sus variantes resaltan: Meltdown

fallos de diseño. Los nombres de esos ataques son Meltdown y Spectre. Y en un sitio Web dedicado a describir estas vulnerabilidades destacan que "aunque los programas normalmente no tienen permiso para leer datos de otros programas, un programa malicioso podría explotar Meltdown, Spectre y apropiarse de secretos almacenados en la memoria de otros programas". Como revelan en su estudio, la diferencia fundamental entre ambos es que Meltdown permite acceder a la memoria del sistema, mientras que Spectre permite acceder a la memoria de otras aplicaciones para robar esos datos.

Ya que Meltdown y Spectre son problemas de diseño en los procesadores, no es posible encontrar solución por Hardware para los procesadores existentes y dado que constantemente aparecen nuevas formas de explotar dichos fallos, la única manera de mantener el equipo de cómputo, comunicaciones y redes de internet a salvo es mediante Software que debe implementar las soluciones en los sistemas operativos. En particular en el Kernel de Linux se trabaja en parchar en cada versión del Kernel todos los fallos reportados, por esto y por otra gama de fallos e inseguridades es necesario mantener siempre el sistema operativo y sus aplicaciones actualizadas.

Como se había comentado anteriormente, estos problemas de diseño afectan a todos los procesadores Intel, AMD, IBM POWER y ARM. Eso incluye básicamente a todos los procesadores que están funcionando al día de hoy[14] en nuestros equipos, ya que estos procesadores llevan produciéndose desde 1995. Afecta a una amplia gama de sistemas.

En el momento de hacerse pública su existencia se incluían todos los dispositivos que no utilizasen una versión convenientemente parcheada de IOS, GNU/Linux, MacOS, Android, Windows y Android. Por lo tanto, muchos servidores y servicios en la nube se han visto impactados, así como potencialmente la mayoría de dispositivos inteligentes y sistemas embebidos que utilizan procesadores con arquitectura ARM (dispositivos móviles, televisores

(AC, DE, P, SM, SS, UD, GP, NM, RW, XD, BR, PK, BND), Spectre (PHT, BTB, RSB, STL, SSB, RSRE), PortSmash, Foreshadow, Spoiler, ZombieLoad (1 y 2), Kaiser, RIDL, Plundervolt, LVI, Take a Way, Collide+Probe, Load+Reload, LVI-LFB, MSD, CSME, RYZENFALL (1, 2, 3, 4), FALLOUT (1, 2, 3) , CHIMERA (FW, HW), MASTERKEY (1, 2, 3), SWAPGS, ITLB_Multihit, SRBDS, L1TF, etc. Más información en:

https://cve.mitre.org
https://meltdownattack.com/

[14]Solo en el año 2021 se detectaron 16 vulnerabilidades en procesadores INTEL y 31 en los procesadores AMD.

inteligentes y otros), incluyendo una amplia gama de equipo usado en redes. Se ha considerado que una solución basada únicamente en Software para estas fallas alenta los equipos de cómputo entre un 20 y un 40 por ciento dependiendo de la tarea que realizan y el procesador del equipo.

Memoria RAM La memoria RAM (Random Access Memory) o memoria de acceso aleatorio es un componente físico de nuestro ordenador, generalmente instalado sobre la misma placa base. La memoria RAM es extraíble y se puede ampliar mediante módulos de distintas capacidades.

La función de la memoria RAM es la de cargar los datos e instrucciones que se ejecutan en el procesador. Estas instrucciones y datos provienen del sistema operativo, dispositivos de entrada y salida, de discos duros y todo lo que está instalado en el equipo.

En la memoria RAM se almacenan todos los datos e instrucciones de los programas que se están ejecutando, estas son enviadas desde las unidades de almacenamiento antes de su ejecución. De esta forma podremos tener disponibles todos los programas que ejecutamos. Se llama memoria de acceso aleatorio porque se puede leer y escribir en cualquiera de sus posiciones de memoria sin necesidad de respetar un orden secuencial para su acceso.

De forma general existen o han existido dos tipos de memorias RAM. Las de tipo asíncrono, que no cuentan con un reloj para poder sincronizarse con el procesador. Y las de tipo síncrono que son capaces de mantener la sincronización con el procesador para ganar en eficacia y eficiencia en el acceso y almacenamiento de información en ellas. Veamos cuales existen de cada tipo.

Memorias de Tipo Asíncrono o DRAM las primeras memorias DRAM (Dinamic RAM) o RAM dinámica eran de tipo asíncrono. Se denomina DRAM por su característica de almacenamiento de información de forma aleatoria y dinámica. Su estructura de transistor y condensador hace que para que un dato quede almacenado dentro una celda de memoria, será necesario alimentar el condensador de forma periódica.

Estas memorias dinámicas eran de tipo asíncrono, por lo que no existía un elemento capaz de sincronizar la frecuencia del procesador con la frecuencia de la propia memoria. Esto provocaba que existiera menor eficiencia en la comunicación entre estos dos elementos.

Memorias de Tipo Síncrono o SDRAM a diferencia de las anteriores esta memoria RAM dinámica cuenta con un reloj interno capaz de sincronizar esta con el procesador. De esta forma se mejoran notablemente los tiempos de acceso y la eficiencia de comunicación entre ambos elementos. Actualmente todos nuestras computadoras cuentan con este tipo de memorias operando en ellos. Las principales tipos de memoria son: DDR, DDR2, DDR3, DDR4 y la nueva DDR5. Donde las tasas de transferencia (GB/s) son: DDR (2.1 - 3.2), DDR2 (4.2 - 6.4), DDR3 (8.5 - 14.9), DDR4 (17 - 25.6) y DDR5 (38.4 - 51.2).

Aparte las características propias de cada una de las diferentes memorias DDR, la característica más importante es que, por ejemplo, en la memoria DDR4 cuatro cores pueden acceder simultáneamente a ella y en la DDR5 serán cinco cores.

Caché L1, L2 y L3 La memoria Caché es otra de las especificaciones importantes de los procesadores, y sirve de manera esencial de la misma manera que la memoria RAM: como almacenamiento temporal de datos. No obstante, dado que la memoria Caché está en el procesador en sí, es mucho más rápida y el procesador puede acceder a ella de manera más eficiente, así que el tamaño de esta memoria puede tener un impacto bastante notable en el rendimiento, especialmente cuando se realizan tareas que demandan un uso intensivo del CPU como en el cómputo de alto desempeño o cómputo científico.

La Caché se divide en diferentes jerarquías de acceso:

- La Caché L1 es el primer sitio donde la CPU buscará información, pero también es la más pequeña y la más rápida, a veces para mayor eficiencia, la Caché L1 se subdivide en L1d (datos) y L1i (instrucciones), actualmente los procesadores modernos en cada core tiene su propio cache de datos e instrucciones.

- La Caché L2 suele ser más grande que la L1 pero es algo más lenta. Sin embargo, por norma general es la que mayor impacto tiene en el rendimiento, este también está incluido en cada core.

- La Caché L3 es mucho más grande que las anteriores, y generalmente se comparte entre todos los núcleos del procesador (a diferencia de las anteriores, que normalmente van ligadas a cada core). Este tercer nivel

es en el que buscará el procesador la información tras no encontrarla en la L1 y L2, por lo que su tiempo de acceso es todavía mayor.

Para poner en contexto la relevancia de la memoria Caché, supongamos que el acceso a los datos de la memoria Caché L1 por el procesador es de dos ciclos de reloj, el acceso a la memoria Caché L2 es de 6 ciclos de reloj, el acceso a la memoria Caché L3 es de 12 ciclos y el acceso a la RAM es de 32 ciclos de reloj.

Además supongamos que la operación suma y resta necesitan de 2 ciclos de reloj para completar la operación una vez que cuente con los datos involucrados en dicha operación, que la multiplicación requiere 4 ciclos de reloj para completar la operación, la división necesita 6 ciclos de reloj para completar la operación y estamos despreciando el tiempo necesario para poner los datos del Caché L1 a los registros del procesador para poder iniciar el cálculo, así también despreciamos el tiempo requerido para sacar el resultado de los registros del procesador al Caché L1.

Esto nos da una idea del número máximo teórico de operaciones básicas que un procesador puede realizar por segundo dependiendo de la velocidad de reloj de la CPU[15].

Si nosotros necesitamos hacer la multiplicación de una matriz $\underline{\underline{A}}$ es de tamaño $n \times n$ por un vector $\underline{u}$ de tamaño n y guardar el resultado en el vector $\underline{f}$ de tamaño n. Entonces algunos escenarios son posibles:

1. Si el código del programa cabe en el Caché L1 de instrucciones y la matriz $\underline{\underline{A}}$, los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en el Caché L1 de datos, entonces el procesador estará siendo utilizado de forma óptima al hacer los cálculos pues no tendrá tiempos muertos por espera de datos.

2. Si el código del programa cabe en el Caché L1 de instrucciones y los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en el Caché L1 de datos pero la matriz $\underline{\underline{A}}$ está dispersa entre los Cachés L1 y L2, entonces el procesador estará teniendo algunos tiempos muertos mientras carga la parte que necesita de la matriz del Caché L2 a L1 para hacer los cálculos y

[15] Por ejemplo en un procesador AMD Ryzen 9 3900X con 12 Cores (2 Threads por Core) por procesador emulando un total de 24 Cores, corre a una frecuencia base de 3,340 MHz, con una frecuencia mínima de 2,200 MHz y máxima de 4,917 Mhz, con Caché L1d de 384 KiB, L1i de 384 KiB, Caché L2 de 6 MiB y Caché L3 de 64 MiB.

utilizado de forma óptima el procesador mientras no salga del Caché L1.

3. Si el código del programa cabe en el Caché L1 de instrucciones y los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en el Caché L1 de datos pero la matriz $\underline{\underline{A}}$ está dispersa entre los Cachés L1, L2 y L3, entonces el procesador estará teniendo muchos tiempos muertos mientras carga la parte que necesita de la matriz del Caché L3 y L2 a L1 para hacer los cálculos resultando en mediana eficiencia en el uso del procesador.

4. Si el código del programa cabe en el Caché L1 de instrucciones y los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en los Cachés L3, L2 y L1 pero los datos de la matriz $\underline{\underline{A}}$ está dispersa entre la RAM y los Cachés L3, L2 y L1, entonces el procesador estará teniendo un exceso de tiempos muertos mientras carga la parte que necesita de la matriz de la RAM a los Cachés L3, L2 y L1 para hacer los cálculos resultando en una gran pérdida de eficiencia en el uso del procesador.

Además, debemos recordar que la computadora moderna nunca dedica el cien por ciento del CPU a un solo programa, ya que los equipos son multitarea[16] y multiusuario[17] por lo que la conmutación de procesos (que se realiza cada cierta cantidad de milisegundos) degrada aún más la eficiencia computacional de los procesos que demandan un uso intensivo de CPU[18].

[16] Cuentan con la capacidad para ejecutar varios procesos simultáneamente en uno o más procesadores, para ello necesitan hacer uso de la conmutación de tareas, es decir, cada cierto tiempo detiene el programa que está corriendo y guardan sus datos, para poder cargar en memoria otro programa y sus respectivos datos y así reiniciar su ejecución por un período determinado de tiempo, una vez concluido su tiempo de ejecución se reinicia la conmutación de tareas con otro proceso.

[17] Se refiere a todos aquellos sistemas operativos que permiten el empleo de sus procesamientos y servicios al mismo tiempo. Así, el sistema operativo cuenta con la capacidad de satisfacer las necesidades de varios usuarios al mismo tiempo, siendo capaz de gestionar y compartir sus recursos en función del número de usuarios que estén conectados a la vez.

[18] Actualmente existen una gran cantidad de distribuciones de GNU/Linux que vienen muy optimizadas intentando conseguir la mejor desenvoltura de su arquitectura y configuraciones de serie. En el caso de la configuración por omisión de Debian GNU/Linux y Ubuntu, están pensadas para que sean lo más robusta posible y que se use en todas las circunstancias imaginables, por ello están optimizadas de forma muy conservadora para tener un equilibrio entre eficiencia y consumo de energía. Pero es posible agregar uno o más Kernels GNU/Linux generados por terceros que contenga las optimizaciones

Last Level Cache se le llama Last Level Cache siempre al último nivel de Caché de una CPU, existen dos tipos:

- Last Level Cache Estándar.
- Victim Cache.

Una Victim Cache no actúa como la Caché de último nivel de una CPU, sino que en ese caso lo hace el penúltimo nivel y en la Victim Cache acaban los últimos datos descartados de la Caché y que han sido volcados en la RAM, los cuales son copiados en la Victim Cache para poder acceder a ellos más rápido.

Smart Cache la Smart Cache (o Caché) es esencialmente L3 pero optimizada por Intel para ser más eficiente a la hora de compartir la información en los núcleos de la CPU. A efectos prácticos, se comporta de igual manera que la Caché L3.

Disco Son dispositivos no volátiles (los hay del orden de hasta 32 TB y continuamente incrementan su capacidad[19]), lo que significa que retienen datos

necesarias para hacer más eficiente y competitivo en cuestiones de gestión y ahorro de recursos del sistema.

Hay varias opciones del Kernel GNU/Linux optimizado (Liquorix viene optimizado para multimedia y Juegos, por otro lado XanMod tiene uno para propósito general, otro aplicaciones críticas en tiempo real y otro más para cálculos intensivos) de las últimas versiones estable del Kernel.

[19]Durante años la tecnología más popular entre los fabricantes ha sido la PMR (Perpendicular Magnetic Recording), también conocida como CMR (Conventional Magnetic Recording). A esta tecnología luego se le sumó la variante SMR (Shingled Magnetic Recording), que lograba aumentar la densidad de grabación, pero lo hacía sacrificando velocidad de transferencia y fiabilidad de las operaciones.

Western Digital ha creado la tecnología ePMR (energy-assisted Perpendicular Magnetic Recording) que permite ofrecer mayores densidades de grabación y, según este fabricante, mejorar la fiabilidad de las escrituras y evitar así los sacrificios que había que hacer con SMR (en los últimos tiempos han aparecido unidades de 20 a 32 TB basadas en dicha tecnología).

Más interesante aún es el sistema de grabación MAMR (Microwave-Assisted Magnetic Recording) que hace uso de microondas para calentar el medio de almacenamiento y así lograr mejorar densidad de grabación y fiabilidad de lecturas y escrituras. Hace años ya prometían que gracias a esta tecnología contaríamos con unidades de 40 TB en 2025, pero parece que dicho logro aún tardará en llegar.

incluso cuando no tienen energía. La información almacenada permanece segura e intacta a menos que el disco duro sea destruido o interferido. La información se almacena o se recupera de manera aleatoria en lugar de acceso secuencial. Esto implica que se puede acceder a los bloques de datos en cualquier momento sin necesidad de pasar por otros bloques de datos.

Actualmente, podemos agrupar los discos duros disponibles en cuatro tipos:

- Parallel Advanced Technology Attachment (PATA)
- Serial ATA (SATA)
- Interfaz de sistema de computadora pequeña (SCSI)
- Adjunto de tecnología avanzada paralela
- Unidades de estado sólido (SSD)

En promedio, las velocidades máximas de los discos actuales son:

- Disco SATA3 de 5,400 RPM, Lectura: 102 MB/s, Escritura: 96 MB/s
- Disco SATA3 de 7,200 RPM, Lectura: 272 MB/S, Escritura: 200 MB/s
- Disco SSD SATA, Lectura 550 MB/s, Escritura 520 MB/s
- Disco SSD NVMe, Lectura 6,600 MB/s, Escritura 5,500 MB/s
- Disco SSD PCI 5.0, Lectura 13,000 MB/s, Escritura 12,000 MB/s
- Unidad Flash USB[20] 2.0, 35 MB/s

Esta otra opción es una alternativa a las microondas, pero en HAMR (Heat-Assisted Magnetic Recording) el proceso de calentar el medio de almacenamiento lo realiza un láser. Toshiba prometió lanzar unidades de más de 32 TB en 2024, mientras que Seagate también quería ofrecer esa capacidad de forma inminente para luego dar el salto a unidades de 40 TB e incluso a los 100 TB que plantean para 2030. Ahí es donde probablemente entre en acción la evolución de HAMR+, que tratará de exprimir aún más la densidad de grabación.

[20]Los colores en los puertos USB son: USB 1.X blanco (12 Mbps), USB 2.X Negro (480 Mbps), USB 3.0 Azul oscuro (5 Gbps), USB 3.1 Azul claro (10 Gbps), USB 3.2 Rojo (20 Gbps). En el caso del USB de color Amarillo, esté es un puerto de carga aún con el dispositivo apagado.

- Unidad Flash USB 3.0 o 3.1 gen 1, 5 Gbit/s
- Unidad Flash USB 3.0 o 3.1 gen 2, 10 Gbit/s
- Unidad Flash USB 3.2 gen 2x2, 20 Gbit/s

Disco de Estado Sólido SSD Estos son los últimos avances en tecnología de almacenamiento que tenemos en la industria de las computadoras. Son totalmente diferentes de las otras unidades en que no consisten en partes móviles. Tampoco almacenan datos utilizando magnetismo. En su lugar, hacen uso de la tecnología de memoria flash, circuitos integrados o dispositivos semiconductores para almacenar datos de forma permanente, al menos hasta que se borren. Estas son algunas de sus ventajas.

- Acceso a datos más rápido
- Menos susceptible a los golpes
- Menores tiempos de acceso y latencia
- Menos consumo de energía

Los SSD actuales están disponibles tanto en versiones SATA como en versiones M.2, U.2 y en formato de tarjeta PCI Express 4.0. Los tres últimos hacen uso del protocolo NVMe y la interfaz PCI Express 4.0 x4, lo que les permite superar los 6,600 MB/s de velocidades de lectura y escritura, frente a los 550 MB/s que suelen alcanzar como máximo las unidades SATA. La nueva versión PCI 5.0 ofrece un ancho de banda de 32 GT/s el doble de PCI 4.0, permitiendo discos SSD con 13,000 MS/s de velocidad de lectura secuencial y realizar hasta 2,500K operaciones por segundo de lectura aleatoria y tamaño máximo 15.36 TB a un precio exorbitante.

Tarjetas microSD Cuando compramos una tarjeta microSD para ampliar el almacenamiento de nuestro Smartphone, cámara, tableta o cualquier otro dispositivo electrónico la mayoría de la gente normalmente solo se fija en la capacidad de almacenamiento. Ahora bien, ¿qué significan todas esas etiquetas y nombres que llevan adscritas las microSD? ¿Cuál es la diferencia entre una microSDXC y una microSDHC? ¿Es mejor una UHS-I o una UHS-II? ¿Qué quiere decir que una tarjeta es A1 y V30? A continuación, intentamos aclarar toda esta nomenclatura.

Lo primero que tenemos que tener claro es que cuando analizamos una tarjeta micro SD existen multitud de factores que limitan su velocidad y capacidad de almacenamiento. Su rendimiento depende de factores como el tipo de tarjeta que estamos usando, su clase, y otros detalles como el tipo de bus o el número de operaciones que puede realizar por segundo.

Tipos de tarjetas microSD actualmente existen 4 generaciones distintas de tarjetas de memoria microSD. Cuanto más modernas sean, mayores velocidades y almacenamiento podrán ofrecer:

- Tarjetas micro SD (Secure Digital): Estas son las memorias de primera generación. Las desarrolló el fabricante SanDisk y fueron las primeras en utilizar el formato de 15 x 11 x 1 milímetros. Su capacidad máxima es de 32 GB.
- Tarjetas micro SDHC (Secure Digital High Capacity): Tarjetas de segunda generación. Cuentan con un bus de datos mejorado que permite alcanzar velocidades superiores, aunque su capacidad máxima sigue siendo de 32 GB.
- Tarjetas micro SDXC (Secure Digital Extended Capacity): Estas micro SD utilizan un sistema de archivos exFAT y su velocidad de transferencia puede llegar hasta los 312 MB/s. Su capacidad de almacenamiento puede llegar hasta los 2 TB y es el tipo de tarjeta más común utilizado a día de hoy.
- Tarjetas micro SDUC: Estas son las tarjetas de memoria más modernas y punteras. Utilizan el sistema de archivos exFAT y permiten almacenar entre 2 TB y 128 TB de datos.

Evidentemente con esto no es suficiente. Si queremos tener una idea aproximada de la velocidad de la micro SD tendremos que fijarnos en aspectos como la clase y tipo de bus que emplea.

La clase es una característica que nos indica la velocidad de transferencia de datos mínima de la tarjeta de memoria. Actualmente hay 4 tipos de clase diferentes:

- Clase 2: Velocidad mínima de 2 MB/s
- Clase 4: Velocidad mínima de 4 MB/s

- Clase 6: Velocidad mínima de 6 MB/s
- Clase 10: Velocidad mínima de 10MB/s

Hoy en día la mayoría de tarjetas micro SD son de clase 10, ya que son capaces de transferir más de 10 MB/s y superan esa cifra fácilmente.

El bus determina la velocidad de la interfaz de la tarjeta de memoria, y nos puede servir como indicativo para conocer la rapidez con la que se pueden leer y escribir los datos:

- Bus estándar: Su velocidad de transferencia alcanza hasta los 12.5 MB/s. Es el tipo de bus utilizado en tarjetas de clase 2, 4 y 6.
- Bus de alta velocidad (High Speed): Se utiliza en tarjetas de clase 10 y alcanza una velocidad de hasta 25 MB/s.
- Bus Ultra High Speed (UHS): Estos son los buses con la interfaz más rápida, y existen varios tipos:
  - UHS-I: Hay dos tipos de buses UHS-I. Por un lado, tenemos el UHS-I clase 1 (U1) que alcanza velocidades de 50 MB/s. Y luego tenemos el UHS-I clase 3 (U3) que llega hasta los 104 MB/s.
  - UHS-II: Alcanza velocidades de transferencia hasta 312 MB/s.
  - UHS-III: Velocidades de transferencia de datos que alcanzan hasta los 624 MB/s.
- SD-Express: Este es el tipo de bus más potente de todos, llegando hasta los 985 MB/s.

Como referencia, te interesará saber que actualmente la mayoría de tarjetas micro SD de gama media utilizan un bus UHS-I de clase 3 (U3) con velocidades de lectura de hasta 104 MB/s.

Otro factor importante es la velocidad de lectura y escritura aleatoria (IOPS) u operaciones por segundo que puede realizar una tarjeta. Este dato determina el rendimiento mínimo en la lectura y escritura aleatoria de la SD:

- Clase de rendimiento de aplicación A1: Las tarjetas A1 tienen una velocidad mínima de lectura aleatoria de 1,500 IOPS, y una velocidad mínima de escritura aleatoria de 500 IOPS.

- Clase de rendimiento de aplicación A2: Las tarjetas A2 ofrecen velocidades superiores, con 4,000 IOPS de lectura y 2,000 IOPS de escritura.

Normalmente con una tarjeta A1 es más que suficiente para tareas del día a día, aunque si necesitamos un rendimiento superior, por ejemplo, para ejecutar aplicaciones desde la SD o jugar a videojuegos, las tarjetas A2 ofrecen un mejor rendimiento.

La velocidad de escritura aleatoria (A1 y A2) es un dato relevante para los Smartphones y tabletas, pero si tenemos una cámara de grabación, una Action camera o un Dron, la característica en la que nos tenemos que fijar es en el Velocidad de escritura secuencial (sistema V) que utiliza (en inglés, Video Speed Class). O dicho de otra forma, en su velocidad de escritura secuencial.

Esta característica nos indica la cantidad de datos que se pueden grabar en la micro SD de forma constante sin bajar de una velocidad mínima. Esto resulta esencial cuando queremos grabar vídeos en alta y ultra-alta definición:

- V30 (Video Speed Class 30): Velocidad de escritura mínima de 30 MB/s
- V60 (Video Speed Class 60): Velocidad de escritura mínima de 60 MB/s
- V90 (Video Speed Class 90): Velocidad de escritura mínima de 90 MB/s

Por ejemplo, si vamos a grabar vídeo en resolución 4K directamente en la tarjeta micro SD, es necesario que la velocidad V sea lo máximo posible, especialmente si vamos a utilizar un amplio BitRate con bajos niveles de compresión, o calidades superiores como el 8K.

Cintas Magnéticas En el año 2010, se comunicó que todos los datos utilizados para el proyecto del satélite Nimbus se recuperaron de cintas que en ese momento tenían 46 años. A partir de dicho comunicado se extendió el uso de la cinta magnética para almacenamiento de datos en todo el mundo.

En un mundo altamente digital, la cinta magnética es una de las pocas tecnologías que utiliza señales analógicas para mover parte de los datos, en su esencia, la cinta se parece mucho a un HDD, utiliza materiales de base magnética, pero en este caso, la cinta es literalmente una base de material

generalmente nailon que tiene un revestimiento magnético. Y en lugar de un disco giratorio, la cinta entra, está enhebrada. Puede parecer una cinta VHS, pero es mucho más robusta.

La cinta se mueve linealmente hacia la unidad de cinta y hacia el cartucho de cinta. Para escribir, el cabezal de la cinta toma señales electrónicas y crea un mini campo magnético que puede cambiar la polaridad del material de la película para formar un patrón de ceros y unos. Una vez que los datos se escriben en la cinta, no se pueden cambiar (pero se pueden borrar y reescribir).

La inmutabilidad y las capacidades de encriptación de la cinta, así como la simplicidad de crear un espacio para almacenarla en una bóveda hacen de la cinta un arma clave para asegurar que los datos sobrevivan frente al Ransomware. Uno de los grandes productores de cintas en la actualidad es IBM, los cuales argumentan que esta función hace que la cinta sea el medio ideal para almacenar datos de archivo a los que no es necesario acceder con frecuencia.

La cinta también puede servir como una copia de seguridad y de versiones fuera de línea de archivos importantes o confidenciales que son resistentes a los ataques cibernéticos. Los tipos de datos que permanecen en la cinta abarcan registros financieros, registros médicos, información de identificación personal y documentos que forman parte de una retención legal de múltiples gobiernos.

Una sola cinta mide aproximadamente 3 pulgadas por 3 pulgadas y 3/4 de pulgada de grosor. Es más pequeño que una unidad de disco duro (HDD), pero pesa alrededor de 0.6 kilogramos. Un cartucho puede almacenar 18 Terabytes de datos sin comprimir y 45 Terabytes comprimidos. IBM está trabajando para duplicar esta capacidad en la próxima generación de la tecnología. En cuanto a la velocidad de recuperación, se obtiene un flujo de datos de una unidad de cinta de 1,000 Megabits por segundo, comprimidos.

Una biblioteca de cintas puede variar en tamaño desde algo que puede poner en su escritorio hasta algo que es del tamaño de un refrigerador pequeño (alrededor de 8 pies cuadrados). La pequeña biblioteca del tamaño de un refrigerador tiene capacidad para 1584 cartuchos. IBM promociona que su biblioteca Diamondback será la biblioteca de cintas más densa del mercado. Podrá contener 69 Petabytes de información mientras ocupa menos de 8 pies cuadrados de espacio.

La cinta magnética supera al disco duro y al flash en cuanto a longevidad, costo financiero y costo de huella de carbono, pero pierde en velocidad de

acceso. Las cintas no son recomendables para poner datos de producción en vivo o incluso copias de seguridad, pero son perfectas para cualquier información a la que se acceda con poca frecuencia y que deba conservarse durante mucho tiempo, como registros médicos o datos de archivo.

Es conocido que muchas empresas han usado y seguirán usando cinta magnética en sus operaciones, entre las que destacan las hiperescalas (empresas que han crecido tanto que ofrecen sus propias infraestructuras o tienen datos masivos como resultado de su infraestructura) siempre necesitan muchas formas diferentes de tecnología para manejar la variedad de datos que ingresan a sus sistemas para alimentar una gama de servicios. Entre otras destacan: Bancos, Gobiernos, Milicia y organizaciones como CERN, así como corporaciones como Amazon, Google, Meta, Baidu, Alibaba y Tencent.

Tarjeta Gráfica La tarjeta gráfica o tarjeta de vídeo es un componente que viene integrado en la placa base de la computadora o se instala aparte para ampliar sus capacidades. Concretamente, esta tarjeta está dedicada al procesamiento de datos relacionados con el vídeo y las imágenes que se están reproduciendo en la computadora.

Procesador Gráfico GPU el corazón de la tarjeta gráfica es la GPU o Unidad de procesamiento gráfico, un circuito muy complejo que integra varios miles de millones de transistores diminutos y puede tener desde uno a miles de núcleos (ya es común encontrar computadoras personales con tarjeta de 10496 cores y 24 GB de GRAM) que tienen capacidad de procesamiento independiente. De la cantidad y capacidad de estos núcleos dependerá la potencia.

Así como los procesadores centrales de las CPU, están diseñados con pocos núcleos pero altas frecuencias de reloj, las GPU tienden al concepto opuesto, contando con grandes cantidades de núcleos con frecuencias de reloj relativamente bajas. Luego tienes la memoria gráfica de acceso aleatorio o GRAM, que son Chips de memoria que almacenan y transportan información entre sí. Esta memoria no es algo que vaya a determinar de forma importante el rendimiento máximo de una tarjeta gráfica, aunque si no es suficiente puede acabar lastrando y limitando la potencia de la CPU.

La idea de usar esa potencia para otros menesteres se denomina GPGPU (General Purpose Computation on Graphics Processing Units) o GPU Computing. En el momento en el que las tarjetas gráficas permiten que se pro-

gramen funciones sobre su Hardware se empieza a hacer uso de GPGPU. Al principio era necesario utilizar los lenguajes enfocados a la visualización en pantalla (como OpenGL) para realizar otros cálculos no relacionados con los gráficos. Esto implicaba el uso de funciones muy poco flexibles, originalmente diseñadas para otros fines, lo que hacía que la labor de programar para tal fin fuese realmente tediosa y complicada.

Para facilitar el empleo de las tarjetas gráficas para cualquier uso no vinculado con los gráficos, NVidia desarrolló toda una tecnología alrededor de la tarjeta, que permitía usar la misma para cualquier tarea: CUDA. ATI, la principal (y actualmente casi única) competidora, un poco más tarde haría lo propio lanzando su propia tecnología: Stream. En un principio las tarjetas gráficas solo trabajaban con aritmética de 32 bits, pero en la actualidad ya se cuenta con aritmética de 64 bit (para lograr esto, muchas tarjetas usan dos de sus cores de 32 bits para emular uno de 64 bits, reduciendo su número de cores útiles a la mitad).

Procesador Gráfico Integrado muchos procesadores CPU incorporan una o más GPU en su interior, llamada gráfica integrada (iGPU Integrated Graphics Processing Units o APU Accelerated Processing Unit). Generalmente es muy poco potente, pero lo suficiente para realizar tareas básicas como navegar por Internet, ver vídeos, e incluso para algunos juegos básicos, especialmente en las últimas generaciones puesto que cada vez son más potentes. No obstante, en las últimas generaciones de procesadores cada vez se están introduciendo gráficos integrados más potentes, y ya son capaces de manejar varios monitores, resoluciones 4K e incluso son capaces de mover algunos juegos a una tasa digna de FPS.

Tarjeta Gráfica por ejemplo, la tarjeta gráfica de AMD Instinct MI200 cuenta con más de 200,000 cores y 128 GB de HBM2, NVidia GEFORCE RTX 3090 proporciona 10,496 cores y 24 GB de GDDR6x, NVidia A100 cuenta con 80 GB de memoria HBM2 6192 cores y 432 núcleos tensor[21], mientras que la tarjeta NVidia Titan RTX proporciona 130 Tensor TFLOP de rendimiento, 576 núcleos tensores y 24 GB de memoria GDDR6.

[21] Un Tensor core (o núcleos Tensor) calculan la operación de una matriz 4x4 completa, la cual se calcula por reloj. Estos núcleos pueden multiplicar dos matrices FP16 4x4 y sumar la matriz FP32 al acumulador.

Por otra parte, Intel ha desarrollado una aceleradora gráfica Artic Sound-M pensada para centro de datos (especialmente diseñada para juegos en la nube) que utiliza una GPU DG2 Xe-HPG que viene con una configuración de 512 unidades de ejecución lo que equivale a 4,096 Shaders, por ejemplo, esta aceleradora puede manejar hasta 8 Streamings simultáneos de video 4K o más de 30 si el vídeo es en 1080p y cuenta con más de 60 funciones virtualizadas.

En agosto del 2021, se anunció la construcción de la supercomputadora Polaris, acelerado por 2240 GPU NVIDIA A100 Tensor Core, el sistema puede alcanzar casi 1.4 exaflops de rendimiento teórico de IA y aproximadamente 44 petaflops de rendimiento máximo de doble precisión. Polaris, que será construido por Hewlett Packard Enterprise, combinará simulación y aprendizaje automático al abordar cargas de trabajo informáticas de alto rendimiento de inteligencia artificial y con uso intensivo de datos, impulsadas por 560 nodos en total, cada uno con cuatro GPU NVIDIA A100.

Tipos de Redes Según el Medio Físico Si bien, nuestros dispositivos de cómputo pueden funcionar sin conexión de red, estos se ven inmediatamente limitados. La red nos permite conectarnos a Internet que es el camino por el cual nos conectamos con el mundo. Las formas de conectar nuestros equipos a Internet en un principio fue exclusivamente por red alámbrica, desde ya hace unos años a la fecha se dispone de conexión a red alámbrica e inalámbrica, pero actualmente nuestros dispositivos cuentan casi exclusivamente con conexión inalámbrica.

¿Qué es el ancho de banda? Se trata de la capacidad máxima y la cantidad de datos que se pueden transmitir a través de una conexión (de internet, por ejemplo), en un momento determinado. Algo que debemos tener claro es que el ancho de banda de red es fundamental para la calidad y velocidad de la conexión.

El ancho de banda se mide en *bit/s* o en sus múltiplos *k/bits* o *m/bits* por segundo. Y para la mayoría de los casos, debemos asegurarnos siempre de tener el mayor ancho de banda que nos sea posible, porque de esta manera podremos tener una mejor y más rápida transferencia de datos.

¿Qué es entonces la velocidad de transmisión? Este término se puede definir como la velocidad a la que se transmite la información. Cuando un usuario adquiere un paquete con una empresa prestadora de servicios de

internet, recibe, por ejemplo, 10 mbps, 30 mbps, 100 mbps, etc. Y esto se refiere a la cantidad de datos que podemos descargar o subir a la red. Como recomendación, lo indicado es que para que la velocidad pueda existir será necesario tener un ancho de banda igual o superior a la velocidad contratada en el paquete de servicio.

Normalmente vemos en los anuncios de todos los operadores, que estos ofrecen una cantidad cualquiera de megas de navegación; este valor numérico corresponde a la velocidad de descarga únicamente. Para encontrar la velocidad de subida, es necesario acceder a un test que nos revele cuál es el resultado y si lo que nos prometen, es verdad o no.

¿Qué es la latencia? Es el tiempo total que transcurre desde que enviamos una información, hasta que la misma llega a un receptor. Su valor de medición se hace en milisegundos, también se conoce como *Ping* y está presente en actividades que realiza cotidianamente como jugar en línea o hacer videollamadas.

¿Altera la latencia la velocidad de la conexión? La velocidad de conexión influye en la latencia una vez que pasamos de un rango predeterminado. Poniéndolo en un ejemplo, si tenemos una conexión a Internet de 1 Mbps y la comparamos con una conexión de 100 Mbps, dependiendo del tamaño del paquete se notará una mejora grande en la velocidad. Otros factores que importan a la hora de hablar de latencia son el estar conectado a internet por Wi-fi o un cable, si tiene servicio de fibra óptica o qué tanta distancia hay entre su ordenador y un Router, etc.

La latencia en la conexión también es la suma de otros retardos:

- De procesamiento: se define en el tiempo que tardan los Routers en examinar la cabecera y a su vez, la respuesta en determinar a dónde hay que enviar cualquier paquete haciendo una previa comprobación de sus tablas de enrutamiento.
- De cola: tiempo de espera del paquete para poder ser transmitido a través de un enlace físico. Cabe anotar que no podemos saber previamente si va a haber un retardo de cola o no, ya que este cambia en tiempo real.
- De transmisión: es el tiempo que tarda el paquete en arribar hasta el siguiente nodo o destino final.

- De propagación: es el tiempo que tarda un bit en propagarse desde un punto cualquiera de origen hasta llegar a uno de destino. Su velocidad depende del medio físico por el que se transporte.

El retardo total es la suma de todos los retardos anteriormente enunciados.

Comúnmente se asocia la latencia con la banda de ancha, pero existe una diferencia sustancial entre ambas. Si bien ambas afectan la velocidad de la conexión, la banda ancha permite que se pueda transmitir una gran cantidad de datos, mientras la latencia determina a qué velocidad se transmite esa cantidad de datos.

Si bien, tener red nos permite estar conectados, tenemos una gran limitación por las velocidades de conexión a las que tendremos acceso según el tipo de medio físico que usemos para conectarnos, así como el número de dispositivos con los que compartamos la conexión. Las redes inalámbricas parecen ser omnipresentes, pero las velocidades de interconexión dejan mucho que desear como veremos a continuación.

Redes alámbricas se comunica a través de cables de datos (generalmente basada en Ethernet). Los cables de datos, conocidos como cables de red de Ethernet o cables con hilos conductores (CAT5), conectan computadoras y otros dispositivos que forman las redes. Las redes alámbricas son mejores cuando se necesita mover grandes cantidades de datos a altas velocidades, como medios multimedia de calidad profesional.

Ventajas

- Costos relativamente bajos
- Ofrece el máximo rendimiento posible
- Mayor velocidad - cable de Ethernet estándar hasta 1 Gbps[22]
- Mayor rendimiento de Voz sobre IP.
- Mejores estándares Ethernet en la industria.
- Mayor capacidad de ancho de banda por cables.

[22] El término bps se refiere a transmitir Bits por segundo (se requieren 8 Bits para formar un Byte). Por eso el término de 1,000 Mbps es equivalente a 125 MB/s.

- Aplicaciones que utilizan un ancho de banda continúo.

Desventajas

- El costo de instalación siempre ha sido un problema común en este tipo de tecnología, ya que el estudio de instalación, canaletas, conectores, cables y otros suman costos muy elevados en algunas ocasiones.
- El acceso físico es uno de los problemas más comunes dentro de las redes alámbricas. Ya que para llegar a ciertos lugares, es muy complicado el paso de los cables a través de las paredes de concreto u otros obstáculos.
- Dificultad y expectativas de expansión es otro de los problemas más comunes, ya que cuando pensamos tener un número definido de nodos en una oficina, la mayoría del tiempo hay necesidades de construir uno nuevo y ya no tenemos espacio en los Switches instalados.

Hoy en día se puede hacer la siguiente clasificación de las redes de protocolo Ethernet para cable y fibra óptica[23]:

- Ethernet, que alcanza no más de 10 Mbps de velocidad
- Fast Ethernet, que puede trabajar con hasta 100 Mbps
- Gigabit Ethernet, alcanza hasta 1 Gbps (1000 Mbps aprox.)
- 2.5 y 5 Gigabit Ethernet hasta 2.5 Gbps si usamos cableado Cat 5a y nada menos que 5 Gbps con cableado Cat 6

[23] El récord mundial en mayo del 2022 de transmisión en fibra óptica es de 1.02 petabits por segundo enviados a través de 51.7 kilómetros (que podría transmitir hasta 10 millones de canales por segundo de vídeo a resolución 8K), que rompe al récord anterior de 319 terabits por segundo sobre una distancia de 1,800 millas (con el estimado de descarga de 80,000 películas simultáneamente en un segundo).

En 2023 los investigadores de Electronics and Computer Engineering de Aston University en U.K. alcanzaron una velocidad de 301 terabits por segundo (Tbps), equivalente a transferir 1,800 películas 4K a través de Internet en un segundo, utilizando cables de fibra óptica existentes. En comparación, la velocidad media de banda ancha fija en los EE. UU. es de 242,38 megabits por segundo (Mbps), según Speed Test.

Los resultados de la prueba, que se realizaron utilizando el tipo de cables de fibra ya tendidos en el suelo, lograron esta velocidad vertiginosa enviando luz infrarroja a través de hilos tubulares de vidrio, que es como funciona generalmente la banda ancha de fibra óptica. Pero aprovecharon una banda espectral que nunca se ha utilizado en sistemas comerciales, llamada "banda E", utilizando dispositivos nuevos hechos a medida.

- 10 Gigabit Ethernet, que puede alcanzar hasta los 10 GMbps
- 1000GbE para alcanzar la Ethernet Terabit (125 Gbytes)

La categoría del cable o el tipo de fibra óptica determina la velocidad máxima soportada por cada tipo de cable, pero aunque haya cables con la misma velocidad hay otros factores que determinan su usabilidad, como el ancho de banda o la frecuencia. La frecuencia o ancho de banda determina la potencia de la red, a mayor frecuencia mayor ancho de banda y menor perdida de datos. Este factor es importante si vamos a conectar varios equipos al mismo cable de red o vamos a hacer una gran tirada de cable, ya que cuanto más largo sea el cable de red más potencia perderá. Siempre tendrá más velocidad un cable corto que un cable largo, pero si el ancho de banda es amplio tardará más metros en perder potencia y velocidad.

Ethernet es el estándar que domina la gran mayoría de mercado, presente de manera casi exclusiva tanto en mercado doméstico, como en pequeña y mediana empresa representa también un porcentaje muy significativo en grandes centros de datos. Pero existen otros protocolos que se pueden situar en el mismo nivel de calidad que Gigabit Ethernet como InfiniBand.

InfiniBand es un bus serie bidireccional de comunicaciones de alta velocidad en las que las rápidas comunicaciones entre servidores son críticas para el rendimiento, llegando a ofrecer velocidades de hasta 2.0 Gbps netos en cada dirección del enlace en un nodo simple, 4 Gbps netos en un nodo doble y hasta 8 Gbps netos en un nodo quadruple. Estos nodos a su vez se pueden agrupar en grupos de 4 ó 12 enlaces llegando a velocidades de hasta 96 Gbps netos en un grupo de 12 nodos cuádruples. El factor de velocidad neta viene relacionado con que Infiniband de cada 10 bits que transmite 8 de ellos son datos, basándose en la codificación 8B/10B.

Recientemente se han implementado sistemas en los que ya no se utiliza esta codificación 8B/10B sino la 64B/66B que permite mejorar el porcentaje de datos útiles por trama enviada y que ha permitido los nodos FDR-10 (Fourteen Data Rate-10 a 10 Gbps), FDR (Fourteen Data Rate a 13.64 Gbps) y EDR (Enhanced Data Rate a 25 Gbps). Este último en un grupo de 12 nodos proporciona hasta 300 Gbps. Los últimos desarrollos de Gigabit Ethernet, proporcionan hasta 100 Gbps por puerto.

Estas enormes velocidades de conexión hacen que Infiniband sea una conexión con una muy importante presencia en superordenadores y clústers, por ejemplo del top 500 de superordenadores en 2020, 226 están conectados

internamente con Infiniband, 188 lo están con Gigabig Ethernet y el resto con Myrinet, Cray, Fat Tree u otras interconexiones a medida.

Una de las principales ventajas de Infiniband sobre Ethernet es su bajísima latencia, por ejemplo y basándonos en los datos del estudio de Qlogic "Introduction to Ethernet Latency, an explanation to Latency and Latency measurement", la latencia en 10 Gpbs Ethernet se sitúa en 5 microsegundos mientras que la de Infiniband se sitúa por debajo de los 3 microsegundos.

Los sistemas de conmutadores inteligentes InfiniBand de NVIDIA Mellanox ofrecen el mayor rendimiento y densidad de puertos para computación de alto rendimiento (HPC), IA, Web 2.0, Big Data, nubes y centros de datos empresariales. La compatibilidad con configuraciones de 36 a 800 puertos a hasta 200 Gpbs por puerto permite que los clústeres de cómputo y los centros de datos convergentes funcionen a cualquier escala, lo que reduce los costos operativos y la complejidad de la infraestructura.

Redes inalámbricas: es una red en la que dos o más terminales (ordenadores, tabletas, teléfonos inteligentes, etc.) se pueden comunicar sin la necesidad de una conexión por cable. Se basan en un enlace que utiliza ondas electromagnéticas (radio e infrarrojo) en lugar de cableado estándar. Permiten que los dispositivos remotos se conecten sin dificultad, ya se encuentren a unos metros de distancia como a varios kilómetros.

Asimismo, la instalación de estas redes no requiere de ningún cambio significativo en la infraestructura existente como pasa con las redes cableadas. Tampoco hay necesidad de agujerear las paredes para pasar cables ni de instalar porta cables o conectores. Esto ha hecho que el uso de esta tecnología se extienda con rapidez.

Tipos de redes inalámbricas

- LAN Inalámbrica: Red de área local inalámbrica. También puede ser una red de área metropolitana inalámbrica.
- GSM (Global System for Mobile Communications): la red GSM es utilizada mayormente por teléfonos celulares.
- D-AMPS (Digital Advanced Mobile Phone Service): está siendo reemplazada por el sistema GSM.
- Fixed Wireless Data: Es un tipo de red inalámbrica de datos que puede ser usada para conectar dos o más edificios juntos para extender o

compartir el ancho de banda de una red sin que exista cableado físico entre los edificios.

- Wi-Fi[24]: es uno de los sistemas más utilizados para la creación de redes inalámbricas en computadoras, permitiendo acceso a recursos remotos como internet e impresoras. Utiliza ondas de radio.

Ventajas

- La instalación de redes inalámbricas suele ser más económica.
- Su instalación también es más sencilla.
- Permiten gran alcance; las redes hogareñas inalámbricas suelen tener hasta 100 metros desde la base transmisora.
- Permite la conexión de gran cantidad de dispositivos móviles. En las redes cableadas mientras más dispositivos haya, más complicado será el entramado de cables.

[24] Notemos que una conexión de WiFi tradicional sin obstáculos con una intensidad de banda de 2.4 GHZ puede tener un alcance máximo de 46 metros y para la banda de 5.0 GHz un alcance de 15 metros. Paro hay muchos objetos que interfieren con la señal del Router del WiFi y el sitio en el que colocamos nuestro dispositivo inalámbrico como computadora o teléfono inteligente:

- Superficies y/o objetos de metal o vidrio blindado
- Refrigeradores, lavadoras y radiadores
- Hornos de microondas, cámaras Web, monitores de bebés y teléfonos inalámbricos
- Paredes y muros
- Dispositivos arquitectónicos que funciones como una jaula de Faraday (es un contenedor recubierto por materiales conductores de electricidad como mallas metálicas, papel aluminio, cajas o cestos de basura de acero que funciona como un blindaje contra los efectos de un campo eléctrico proveniente del exterior).

En caso de que varios dispositivos se conecten a la red, se puede optar por colocar el Router en un punto medio, para que ninguna zona quede sin cobertura. Por otra parte, para una mejor conexión, también procura que el Router se encuentre en una zona elevada, pues esto mejorará el alcance de la señal inalámbrica. Una excelente opción para mejorar la calidad del internet inalámbrico cuando hay obstáculos es utilizar un repetidor de WiFi, este dispositivo es muy útil para amplificar la señal y llevarla a más sitios de nuestra red.

- Posibilidad de conectar nodos a grandes distancias sin cableado, en el caso de las redes inalámbricas corporativas.
- Permiten más libertad en el movimiento de los nodos conectados, algo que puede convertirse en un verdadero problema en las redes cableadas.
- Permite crear una red en áreas complicadas donde, por ejemplo, resulta dificultoso o muy caro conectar cables.

Desventajas

- Calidad de Servicio: La velocidad que posee la red inalámbrica no supera la cableada, ya que esta puede llegar a los 10 Mbps, frente a 100 Mbps que puede alcanzar la cableada. Hay que tomar en cuenta la tasa de error debida a las interferencias.
- Costo: En algunos casos, puede ser más barato cablear una casa/oficina que colocar un servicio de red inalámbrica.
- La señal inalámbrica puede verse afectada e incluso interrumpida por objetos, árboles, paredes, espejos, entre otros.

La velocidad máxima de transmisión inalámbrica de la tecnología 802.11b es de 11 Mbps. Pero la velocidad típica es solo la mitad: entre 1.5 y 5 Mbps dependiendo de si se transmiten muchos archivos pequeños o unos pocos archivos grandes. La velocidad máxima de la tecnología 802.11g es de 54 Mbps. Pero la velocidad típica de esta última tecnología es solo unas 3 veces más rápida que la de 802.11b: entre 5 y 15 Mbps. Resumiendo, las velocidades típicas de los diferentes tipos de red son:

Estándar	V.Máxima	V.Practica	Frecuencia	Ancho Banda	Alcance
802.11	2Mbit/s	1Mbit/s	2.4Ghz	22MHz	330 metros
802.11a(WiFi5)	54Mbit/s	22Mbit/s	5.4Ghz	20MHz	390 metros
802.11b	11Mbit/s	6Mbit/s	2.4Ghz	22MHz	460 metros
802.11g	54Mbit/s	22Mbit/s	2.4Ghz	20MHz	460 metros
802.11n	600Mbit/s	100Mbit/s	2.4Ghzy5.4Ghz	20 y 40MHz	820 metros
802.11ac	6.93Gbps	100Mbit/s	5.4Ghz	80 o hasta 160MHz	Poco alcance, pero sin interferencias
802.11ad	7.13Gbit/s	Hasta6Gbit/s	60Ghz	2MHz	300 metros
802.11ah	35.6Mbps	26.7Mbps	0.9Ghz	2MHz	1000 metros
802.11ax(WiFi6)	9,6Gbps	6,9Gbps	2.4Ghz y 5.4Ghz	20MHz	1000 metros

Como puedes ver, los principales factores que influyen en la calidad de una conexión WiFi, son la frecuencia, el ancho de banda y el alcance total. Considerando que, todo esto junto con la velocidad máxima y la velocidad práctica, se congregan en lo que es cada versión de este tipo de conexión a la red. Además, la conexión se degradará inexorablemente con la cantidad de dispositivos conectados y su consumo de datos.

Velocidad de los Proveedores cuando se contrata el servicio de internet, la velocidad de interconexión del mismo depende de cuánto sea el cobro, pero en la mayoría de los casos se tendrá una velocidad de descarga mayor a la velocidad de carga y nuestra red será una intranet (dirección de IP dinámica) compartida con otros miles de usuarios abonados al servicio de internet del proveedor. También es posible contratar una interconexión con velocidades homogéneas para carga y descarga dedicada, el costo del mismo se puede hasta triplicar con respecto a uno no homogéneo.

En caso de requerir una dirección de internet homologada o pública, el costo de contratar el servicio aumenta considerablemente, pero de esta forma nuestros equipos son visibles en el Internet (esto conlleva un aumento de riesgos al estar nuestros equipos más vulnerables a ataques informáticos).

El Tráfico Global en Internet El tráfico en Internet a nivel global sigue creciendo de manera vertiginosa. Sólo en 2024 aumentó un 17.2%, según los datos del informe de 2024 Cloudflare Radar Year in Review, publicado por la compañía por quinto año consecutivo, y en el que se repasa información sobre conectividad, seguridad, uso de dispositivos para acceso a la Red y frecuencia de los apagones, entre otras tendencias.

Según este informe, y para sorpresa de prácticamente nadie a estas alturas, los servicios de Internet más populares del mundo son Google, Facebook, Apple, TikTok y AWS. Chrome, con un 65.8% de los internautas como usuarios, es el navegador más utilizado en todo el mundo. WhatsApp sigue siendo la aplicación de mensajería más popular.

React, PHP y JQuery están entre las tecnologías para desarrollar webs más populares. HubSpot, Google y WordPress están entre los proveedores más populares de servicios y plataformas de soporte. Además, Go, ha superado a NodeJS como lenguaje más popular para hacer peticiones de API automatizadas.

Los rastreadores de IA son una gran fuente de tráfico, aunque despier-

tan cada vez más suspicacias por su actividad. Básicamente, se dedican a escanear la web para recopilar grandes cantidades de datos para entrenar modelos grandes de lenguaje. Preocupa bastante el hecho de que algunos recojan datos sin que tengan permiso para hacerlo, frente a los bots «buenos» y verificados, que suelen tener su origen en los motores de búsqueda y son transparentes sobre lo que son y lo que hacen. En este grupo entran GoogleBot, Qualys o BingBot.

Cloudflare, entre otras actividades, se dedica a rastrear el tráfico relacionado con la IA para determinar qué bots son los más agresivos, cuáles tienen el mayor volumen de peticiones y cuáles hacen rastreos de manera regular. Según los investigadores que han realizado el informe, el denominado facebookexternalhit es el que más tráfico de todos ha generado durante el año: un 27.16%. Le siguen Bytespider, de ByteDance, con un 23.35%; Amazonbot, con un 13.34%, ClaudeBot, de Anthropic, con un 8.06%; y GPTBot, con un 5.60%.

Al parecer, el tráfico generado por Bytespider, bot del propietario de TikTok, ha ido descendiendo de manera gradual durante 2024. En las semanas finales del año generaba entre un 80% y un 85% menos de tráfico que al principio. Mientras tanto el de ClaudeBot registró una subida muy pronunciada hacia mitad de 2024, y luego cayó otra vez de forma brusca.

La mayoría de las peticiones Web son todavía de HTTP2, que se lanzó en 2015, y cuenta aún con un 49.6% de las mismas. Un 29.9% todavía son de HTTP original, estandarizado en 1996, mientras que solo un 20.5% son de HTTP3, desplegado en 2022.

Cloudflare también registra, además del tráfico HTTP, las conexiones realizadas a través del protocolo de transmisión TCP, que asegura que hay una transferencia de datos fiable entre dispositivos de red. En 2024, un 20.7% de las conexiones TCP finalizaron de manera inesperada antes de que pudiesen terminar en un intercambio de datos útiles.

Estas anomalías en las conexiones TCP pueden deberse a varios motivos. Entre ellos a los ataques de denegación de servicio (DoS), al escaneado de redes, a las desconexiones de clientes, a la manipulación de las conexiones o a lo que Cloudflare ha llamado «comportamiento poco habitual del cliente».

La mayor parte de las interrupciones de una conexión TCP identificadas por Cloudflare se produjeron en 2024 después de que un servidor recibiese una petición de sincronización, pero antes de que recibiese un reconocimiento de la misma.

En cuanto a la seguridad, Cloudflare ha señalado que sólo un 4.3% de los

mensajes de correo electrónico enviados en 2024 eran maliciosos. En estos había sobre todo enlaces falsos (42.9%) o identidades falsas (35.1%). En el 70% de los casos se utilizaban los dos métodos.

Como datos curiosos en cuanto a seguridad, la compañía asegura que la vulnerabilidad Log4j todavía se usa como método de ataque, y se usa de manera mucho más activa que otras vulnerabilidades comunes. Además, prácticamente todos los mensajes de correo procesados por Cloudflare con dominios .bar, .rest y .uno eran o correos de Spam, o mensajes que, directamente, eran maliciosos.

En 2024, por otro lado, hubo 225 apagones de Internet de gran envergadura. La mayoría se dieron en África, Oriente Medio o India. Más de la mitad fueron apagones ordenados directamente por gobiernos, mientras que otros se debieron a otras causas.

Entre ellas, corte de cables, apagones eléctricos, problemas técnicos o de mantenimiento, episodios climáticos graves o ciberataques. Muchos duraron solo unas pocas horas, mientras que otros, como el experimentado en Bangladesh en julio, duró 10 días. Dentro de estos apagones está también el provocado por el incidente de CrowdStrike el pasado verano.

Otro de los aspectos tratados en el informe es la calidad de la conexión a Internet de los países, con base en su velocidad de subida, la de bajada y su latencia. España, con una velocidad de descarga media de 292.6 Mbps, y de 192.6 Mbps de subida, está a la cabeza en velocidad de conexión. Todos los países desarrollados presentan velocidades de descarga por encima de 200 Mbps de media.

Un 41.3% del tráfico de Internet a nivel mundial procede de dispositivos móviles, con el otro 58.7% producido en ordenadores portátiles y de sobremesa. Eso sí, en alrededor de un centenar de países del mundo, la mayoría del tráfico en 2024 procede de dispositivos móviles. Cuba y Siria tienen el mayor tráfico de dispositivos móviles, un 77%.

Otras áreas de alta demanda de tráfico de dispositivos de este tipo son Oriente Medio, África, Asia-Pacífico y América Central y del Sur. En este aspecto, las mediciones del tráfico son parecidas a las de 2023 y 2022.

Información de mi Computadora Si lo que deseamos es un listado detallado del Hardware de nuestro equipo de cómputo, entonces podemos usar cualquiera de estos comandos (que previamente deberemos instalar):

```
$ lscpu
```

```
# lshw
# dmidecode
# hwinfo
```

1.2 Sobre los Ejemplos de este Trabajo

La documentación y los diferentes ejemplos que se presentan en este trabajo, se encuentran disponibles en la siguiente liga:

Herramientas
http://132.248.181.216/Herramientas/

para que puedan ser copiados desde el navegador y ser usados en la terminal de línea de comandos. En aras de que el interesado pueda correr dichos ejemplos y afianzar sus conocimientos, además de que puedan ser usados en diferentes ámbitos a los presentados aquí.

1.3 Agradecimientos

Este texto es una recopilación de múltiples fuentes, mi aportación -si es que puedo llamarla así- es plasmarlo en este documento, en el que trato de dar coherencia a mi visión de los temas desarrollados.

En la realización de este texto se han revisado -en la mayoría de los casos indicó la referencia, pero pude omitir varias de ellas, por lo cual pido una disculpa- múltiples páginas Web, artículos técnicos, libros, entre otros materiales bibliográficos, los más representativos y de libre acceso los pongo a su disposición en la siguiente liga:

Herramientas
http://132.248.181.216/Herramientas/

Este proyecto fue posible gracias al apoyo recibido por la Facultad de Ciencias de la Universidad Nacional Autónoma de México (UNAM) y al tiempo robado a nuestras actividades académicas, principalmente durante el período de confinamiento de los años 2020 a 2022.

2 Estructura Óptima de las Matrices en su Implementación Computacional

Una parte fundamental de la implementación computacional de los métodos numéricos de resolución de sistemas algebraicos, es utilizar una forma óptima de almacenar, recuperar y operar las matrices, tal que, facilite los cálculos que involucra la resolución de grandes sistemas de ecuaciones lineales cuya implementación puede ser secuencial o paralela (véase [7]).

El sistema lineal puede ser expresado en la forma matricial $\underline{\underline{A}}\underline{u} = \underline{f}$, donde la matriz $\underline{\underline{A}}$ -que puede ser real o virtual- es de tamaño $n \times n$ con banda b, pero el número total de datos almacenados en ella es a los más $n * b$ números de doble precisión, en el caso de ser simétrica la matriz, el número de datos almacenados es menor a $(n * b)/2$. Además si el problema que la originó es de coeficientes constantes el número de valores almacenados se reduce drásticamente a sólo el tamaño de la banda b.

En el caso de que el método para la resolución del sistema lineal a usar sea del tipo Factorización LU o Cholesky, la estructura de la matriz cambia, ampliándose el tamaño de la banda de b a $2 * b + 1$ en la factorización, en el caso de usar métodos iterativos tipo CGM o GMRES la matriz se mantiene intacta con una banda b.

Para la resolución del sistema lineal virtual asociada a los métodos de descomposición de dominio, la operación básica que se realiza de manera reiterada, es la multiplicación de una matriz por un vector $\underline{v} = \underline{\underline{C}}\underline{u}$, la cual es necesario realizar de la forma más eficiente posible.

Un factor determinante en la implementación computacional, para que esta resulte eficiente, es la forma de almacenar, recuperar y realizar las operaciones que involucren matrices y vectores, de tal forma que la multiplicación se realice en la menor cantidad de operaciones y que los valores necesarios para realizar dichas operaciones queden en la medida de lo posible contiguos para ser almacenados en el Cache[25] del procesador.

[25] Nótese que la velocidad de acceso a la memoria principal (RAM) es relativamente lenta con respecto al Cache, este generalmente está dividido en sub-Caches L1 -de menor tamaño y el más rápido-, L2 y hasta L3 -el más lento y de mayor tamaño- los cuales son de tamaño muy reducido con respecto a la RAM.

Por ello, cada vez que las unidades funcionales de la Unidad de Aritmética y Lógica requieren un conjunto de datos para implementar una determinada operación en los registros, solicitan los datos primeramente a los Caches, estos consumen diversa cantidad de ciclos de reloj para entregar el dato si lo tienen -pero siempre el tiempo es menor que

2.1 Almacenamiento en la Memoria RAM

La memoria RAM (Random Access Memory) o memoria de acceso aleatorio es un componente físico de nuestro ordenador, generalmente instalado sobre la misma placa base. La memoria RAM es extraíble y se puede ampliar mediante módulos de distintas capacidades.

La función de la memoria RAM es la de cargar los datos e instrucciones que se ejecutan en el procesador. Estas instrucciones y datos provienen del sistema operativo, dispositivos de entrada y salida, de discos duros y todo lo que está instalado en el equipo.

En la memoria RAM se almacenan todos los datos e instrucciones de los programas que se están ejecutando, estas son enviadas desde las unidades de almacenamiento antes de su ejecución. De esta forma podremos tener disponibles todos los programas que ejecutamos. Se llama memoria de acceso aleatorio porque se puede leer y escribir en cualquiera de sus posiciones de memoria sin necesidad de respetar un orden secuencial para su acceso.

La RAM dinámica cuenta con un reloj interno capaz de sincronizar esta con el procesador. De esta forma se mejoran notablemente los tiempos de acceso y la eficiencia de comunicación entre ambos elementos. Actualmente todos nuestras computadoras cuentan con este tipo de memorias operando en ellos. Las principales tipos de memoria son: DDR, DDR2, DDR3, DDR4 y la nueva DDR5. Donde las tasas de transferencia (GB/s) son: DDR (2.1 - 3.2), DDR2 (4.2 - 6.4), DDR3 (8.5 - 14.9), DDR4 (17 - 25.6) y DDR5 (38.4 - 51.2). La característica más importante es que, por ejemplo, en la memoria DDR4 cuatro cores pueden acceder simultáneamente a ella y en la DDR5 serán cinco cores.

Caché L1, L2 y L3 La memoria Caché es otra de las especificaciones importantes de los procesadores, y sirve de manera esencial de la misma manera que la memoria RAM: como almacenamiento temporal de datos. No obstante, dado que la memoria Caché está en el procesador en sí, es mucho más rápida y el procesador puede acceder a ella de manera más eficiente, así que el tamaño de esta memoria puede tener un impacto bastante notable en el rendimiento, especialmente cuando se realizan tareas que demandan un uso intensivo del CPU como en el cómputo de alto desempeño o cómputo científico.

solicitarle el dato a la memoria principal-; en caso de no tenerlo, se solicitan a la RAM para ser cargados a los caches y poder implementar la operación solicitada.

La Caché se divide en diferentes jerarquías de acceso:

- La Caché L1 es el primer sitio donde la CPU buscará información, pero también es la más pequeña y la más rápida, a veces para mayor eficiencia, la Caché L1 se subdivide en L1d (datos) y L1i (instrucciones), actualmente los procesadores modernos en cada core tiene su propio cache de datos e instrucciones.

- La Caché L2 suele ser más grande que la L1 pero es algo más lenta. Sin embargo, por norma general es la que mayor impacto tiene en el rendimiento, este también está incluido en cada core.

- La Caché L3 es mucho más grande que las anteriores, y generalmente se comparte entre todos los núcleos del procesador (a diferencia de las anteriores, que normalmente van ligadas a cada core). Este tercer nivel es en el que buscará el procesador la información tras no encontrarla en la L1 y L2, por lo que su tiempo de acceso es todavía mayor.

Para poner en contexto la relevancia de la memoria Caché, supongamos que el acceso a los datos de la memoria Caché L1 por el procesador es de dos ciclos de reloj, el acceso a la memoria Caché L2 es de 6 ciclos de reloj, el acceso a la memoria Caché L3 es de 12 ciclos y el acceso a la RAM es de 32 ciclos de reloj.

Además supongamos que la operación suma y resta necesitan de 2 ciclos de reloj para completar la operación una vez que cuente con los datos involucrados en dicha operación, que la multiplicación requiere 4 ciclos de reloj para completar la operación, la división necesita 6 ciclos de reloj para completar la operación y estamos despreciando el tiempo necesario para poner los datos del Caché L1 a los registros del procesador para poder iniciar el cálculo, así también despreciamos el tiempo requerido para sacar el resultado de los registros del procesador al Caché L1.

Esto nos da una idea del número máximo teórico de operaciones básicas que un procesador puede realizar por segundo dependiendo de la velocidad de reloj de la CPU[26].

[26] Por ejemplo en un procesador AMD Ryzen 9 3900X con 12 Cores (2 Threads por Core) por procesador emulando un total de 24 Cores, corre a una frecuencia base de 3,340 MHz, con una frecuencia mínima de 2,200 MHz y máxima de 4,917 Mhz, con Caché L1d de 384 KiB, L1i de 384 KiB, Caché L2 de 6 MiB y Caché L3 de 64 MiB.

Si nosotros necesitamos hacer la multiplicación de una matriz $\underline{\underline{A}}$ es de tamaño $n \times n$ por un vector $\underline{u}$ de tamaño n y guardar el resultado en el vector $\underline{f}$ de tamaño n. Entonces algunos escenarios son posibles:

1. Si el código del programa cabe en el Caché L1 de instrucciones y la matriz $\underline{\underline{A}}$, los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en el Caché L1 de datos, entonces el procesador estará siendo utilizado de forma óptima al hacer los cálculos pues no tendrá tiempos muertos por espera de datos.

2. Si el código del programa cabe en el Caché L1 de instrucciones y los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en el Caché L1 de datos pero la matriz $\underline{\underline{A}}$ está dispersa entre los Cachés L1 y L2, entonces el procesador estará teniendo algunos tiempos muertos mientras carga la parte que necesita de la matriz del Caché L2 a L1 para hacer los cálculos y utilizado de forma óptima el procesador mientras no salga del Caché L1.

3. Si el código del programa cabe en el Caché L1 de instrucciones y los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en el Caché L1 de datos pero la matriz $\underline{\underline{A}}$ está dispersa entre los Cachés L1, L2 y L3, entonces el procesador estará teniendo muchos tiempos muertos mientras carga la parte que necesita de la matriz del Caché L3 y L2 a L1 para hacer los cálculos resultando en mediana eficiencia en el uso del procesador.

4. Si el código del programa cabe en el Caché L1 de instrucciones y los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en los Cachés L3, L2 y L1 pero los datos de la matriz $\underline{\underline{A}}$ está dispersa entre la RAM y los Cachés L3, L2 y L1, entonces el procesador estará teniendo un exceso de tiempos muertos mientras carga la parte que necesita de la matriz de la RAM a los Cachés L3, L2 y L1 para hacer los cálculos resultando en una gran pérdida de eficiencia en el uso del procesador.

Además, debemos recordar que la computadora moderna nunca dedica el cien por ciento del CPU a un solo programa, ya que los equipos son mul-

titarea[27] y multiusuario[28] por lo que la conmutación de procesos (que se realiza cada cierta cantidad de milisegundos) degrada aún más la eficiencia computacional de los procesos que demandan un uso intensivo de CPU[29].

Dado que la multiplicación de una matriz $\underline{\underline{C}}$ por un vector $\underline{u}$, dejando el resultado en $\underline{v}$ se realiza mediante el algoritmo

```
for (i=0; i<ren; i++)
{
    s = 0.0;
    for (j=0; j < col; j++)
    {
        s += C[i][j]*u[j];
    }
    v[i] = s;
}
```

[27] Cuentan con la capacidad para ejecutar varios procesos simultáneamente en uno o más procesadores, para ello necesitan hacer uso de la conmutación de tareas, es decir, cada cierto tiempo detiene el programa que está corriendo y guardan sus datos, para poder cargar en memoria otro programa y sus respectivos datos y así reiniciar su ejecución por un período determinado de tiempo, una vez concluido su tiempo de ejecución se reinicia la conmutación de tareas con otro proceso.

[28] Se refiere a todos aquellos sistemas operativos que permiten el empleo de sus procesamientos y servicios al mismo tiempo. Así, el sistema operativo cuenta con la capacidad de satisfacer las necesidades de varios usuarios al mismo tiempo, siendo capaz de gestionar y compartir sus recursos en función del número de usuarios que estén conectados a la vez.

[29] Actualmente existen una gran cantidad de distribuciones de GNU/Linux que vienen muy optimizadas intentando conseguir la mejor desenvoltura de su arquitectura y configuraciones de serie. En el caso de la configuración por omisión de Debian GNU/Linux y Ubuntu, están pensadas para que sean lo más robusta posible y que se use en todas las circunstancias imaginables, por ello están optimizadas de forma muy conservadora para tener un equilibrio entre eficiencia y consumo de energía. Pero es posible agregar uno o más Kernels GNU/Linux generados por terceros que contenga las optimizaciones necesarias para hacer más eficiente y competitivo en cuestiones de gestión y ahorro de recursos del sistema.

Hay varias opciones del Kernel GNU/Linux optimizado (Liquorix viene optimizado para multimedia y Juegos, por otro lado XanMod tiene uno para propósito general, otro aplicaciones críticas en tiempo real y otro más para cálculos intensivos) de las últimas versiones estable del Kernel.

Para lograr una eficiente implementación del algoritmo anterior, es necesario que el gran volumen de datos desplazados de la memoria al Cache y viceversa sea mínimo. Por ello, los datos se deben agrupar para que la operación más usada -en este caso multiplicación matriz por vector- se realice con la menor solicitud de datos a la memoria principal, si los datos usados -renglón de la matriz- se ponen contiguos minimizará los accesos a la memoria principal, pues es más probable que estos estarán contiguos en el Cache al momento de realizar la multiplicación.

Por ejemplo, en el caso de matrices bandadas de tamaño de banda b, el algoritmo anterior se simplifica a

```
for (i=0; i<ren; i++)
{
      s= 0.0;
      for (k=0; k < ban; k++)
      {
            if ((Ind[k] + i) >= 0 && (Ind[k]+i) < ren)
                  s += Dat[i][k]*u[Ind[k]+i];
      }
      v[i]=s;
}
```

Si, la solicitud de memoria para Dat[i] se hace de tal forma que los datos del renglón estén continuos -son b números de punto flotante-, esto minimizará los accesos a la memoria principal en cada una de las operaciones involucradas en el producto, como se explica en las siguientes secciones.

2.2 Matrices Bandadas

En el caso de las matrices bandadas de banda b -sin pérdida de generalidad y para propósitos de ejemplificación se supone pentadiagonal- típicamente

tiene la siguiente forma

$$\underline{\underline{A}} = \begin{bmatrix} a_1 & b_1 & & & c_1 & & & & \\ d_2 & a_2 & b_2 & & & c_2 & & & \\ & d_3 & a_3 & b_3 & & & c_3 & & \\ & & d_4 & a_4 & b_4 & & & c_4 & \\ e_5 & & & d_5 & a_5 & b_5 & & & c_5 \\ & e_6 & & & d_6 & a_6 & b_6 & & \\ & & e_7 & & & d_7 & a_7 & b_7 & \\ & & & e_8 & & & d_8 & a_8 & b_8 \\ & & & & e_9 & & & d_9 & a_9 \end{bmatrix} \tag{2.1}$$

la cual puede ser almacenada usando el algoritmo (véase [7]) Compressed Diagonal Storage (CDS), optimizado para ser usado en C++, para su almacenamiento y posterior recuperación. Para este ejemplo en particular, se hará uso de un vector de índices

$$\underline{Ind} = [-5, -1, 0, +1, +5] \tag{2.2}$$

y los datos serán almacenados usando la estructura

$$\underline{\underline{Dat}} = \begin{bmatrix} 0 & 0 & a_1 & b_1 & c_1 \\ 0 & d_2 & a_2 & b_2 & c_2 \\ 0 & d_3 & a_3 & b_3 & c_3 \\ 0 & d_4 & a_4 & b_4 & c_4 \\ e_5 & d_5 & a_5 & b_5 & c_5 \\ e_6 & d_6 & a_6 & b_6 & 0 \\ e_7 & d_7 & a_7 & b_7 & 0 \\ e_8 & d_8 & a_8 & b_8 & 0 \\ e_9 & d_9 & a_9 & 0 & 0 \end{bmatrix} \tag{2.3}$$

de tal forma que la matriz $\underline{\underline{A}}$ puede ser reconstruida de forma eficiente. Para obtener el valor $A_{i,j}$, cálculo $ind = j - i$, si el valor ind está en la lista de índices $\underline{Ind}$ -supóngase en la columna k-, entonces $A_{i,j} = Dat_{ik}$, en otro caso $A_{i,j} = 0$.

Casos Particulares de la Matriz Bandada $\underline{\underline{A}}$ Básicamente dos casos particulares surgen en el tratamiento de ecuaciones diferenciales parciales: El primer caso es cuando el operador diferencial parcial es simétrico y el otro, en el que los coeficientes del operador sean constantes.

Para el primer caso, al ser la matriz simétrica, sólo es necesario almacenar la parte con índices mayores o iguales a cero, de tal forma que se buscará el índice que satisfaga $ind = |j - i|$, reduciendo el tamaño de la banda a $b/2$ en la matriz $\underline{\underline{A}}$.

Para el segundo caso, al tener coeficientes constantes el operador diferencial, los valores de los renglones dentro de cada columna de la matriz son iguales, y sólo es necesario almacenarlos una sola vez, reduciendo drásticamente el tamaño de la matriz de datos.

Implementación de Matrices Bandadas Orientada a Objetos en C++ Una forma de implementar la matriz bandada $\underline{\underline{A}}$ de banda b en C++ es mediante:

```
// Creacion
double **Dat;
Dat = new double*[ren];
for (i = 0; i < ren; i++) Dat[i] = new double[b];
int *Ind;
Ind = new int[b];
// Inicializacion
for (i = 0; i < ren; i++)
      for (j = 0; j < b; j++) Dat[i][j] = 0.0;
for (i = 0; i < b; i++) Ind[i] = 0;
```

2.3 Matrices Dispersas

Las matrices dispersas de a lo más b valores distintos por renglón -sin pérdida de generalidad y para propósitos de ejemplificación se supone $b = 3$- que surgen en métodos de descomposición de dominio para almacenar algunas

matrices, típicamente tienen la siguiente forma

$$\underline{\underline{A}} = \begin{bmatrix} a_1 & & & & b_1 & & & c_1 \\ & a_2 & & b_2 & & & c_2 & \\ & & & a_3 & & & b_3 & c_3 \\ a_4 & & & b_4 & & & & \\ & & a_5 & & & b_5 & & c_5 \\ a_6 & b_6 & c_6 & & & & & \\ & & & & & a_7 & b_7 & c_7 \\ & & & a_8 & & b_8 & c_8 & \\ & & & & & a_9 & b_9 & \end{bmatrix} \tag{2.4}$$

la cual puede ser almacenada usando el algoritmo (véase [7]) Jagged Diagonal Storage (JDC), optimizado para ser usado en C++. Para este ejemplo en particular, se hará uso de una matriz de índices

$$\underline{\underline{Ind}} = \begin{bmatrix} 1 & 6 & 9 \\ 2 & 5 & 8 \\ 5 & 8 & 9 \\ 1 & 4 & 0 \\ 3 & 6 & 9 \\ 1 & 2 & 3 \\ 7 & 8 & 9 \\ 4 & 7 & 8 \\ 7 & 8 & 0 \end{bmatrix} \tag{2.5}$$

y los datos serán almacenados usando la estructura

$$\underline{\underline{Dat}} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \\ a_4 & b_4 & 0 \\ a_5 & b_5 & c_5 \\ a_6 & b_6 & c_6 \\ a_7 & b_7 & c_7 \\ a_8 & b_8 & c_8 \\ a_9 & b_9 & 0 \end{bmatrix} \tag{2.6}$$

de tal forma que la matriz $\underline{\underline{A}}$ puede ser reconstruida de forma eficiente. Para obtener el obtener el valor $A_{i,j}$, busco el valor j en la lista de índices $\underline{Ind}$ dentro del renglón i, si lo encuentro en la posición k, entonces $A_{i,j} = Dat_{ik}$, en otro caso $A_{i,j} = 0$.

Casos Particulares de la Matriz Dispersa $\underline{\underline{A}}$ Si la matriz $\underline{\underline{A}}$, que al ser almacenada, se observa que existen a lo más r diferentes renglones con valores distintos de los n con que cuenta la matriz y si $r << n$, entonces es posible sólo guardar los r renglones distintos y llevar un arreglo que contenga la referencia al renglón almacenado.

Implementación de Matrices Dispersas Orientada a Objetos en C++ Una forma de implementar la matriz bandada $\underline{\underline{A}}$ de banda b en C++ es mediante:

```
// Creación
double **Dat;
Dat = new double*[ren];
for (i = 0; i < ren; i++) Dat[i] = new double[b];
int **Ind;
Ind = new int*[ren];
for (i = 0; i < ren; i++) Ind[i] = new int[b];
// Inicialización
for (i = 0; i < ren; i++)
     for (j = 0; j < b; j++) Dat[i][j] = 0.0, Ind[i][j] = -1;
```

2.4 Multiplicación Matriz-Vector

Los métodos de descomposición de dominio requieren por un lado la resolución de al menos un sistema lineal y por el otro lado requieren realizar la operación de multiplicación de matriz por vector, i.e. $\underline{\underline{C}}\underline{u}$ de la forma más eficiente posible, por ello los datos se almacenan de tal forma que la multiplicación se realice en la menor cantidad de operaciones.

Dado que la multiplicación de una matriz $\underline{\underline{C}}$ por un vector $\underline{u}$, dejando el resultado en $\underline{v}$ se realiza mediante el algoritmo:

```
for (i=0; i<ren; i++)
{
     s = 0.0;
     for (j=0; j < col; j++)
     {
          s += C[i][j]*u[j];
     }
```

```
        v[i] = s;
}
```

En el caso de matrices bandadas, se simplifica a:

```
for (i=0; i<ren; i++)
{
        s= 0.0;
        for (k=0; k < ban; k++)
        {
                if ((Ind[k] + i) >= 0 && (Ind[k]+i) < ren)
                        s += Dat[i][k]*u[Ind[k]+i];
        }
        v[i]=s;
}
```

De forma similar, en el caso de matrices dispersas, se simplifica a:

```
for (i=0; i<ren; i++)
{
        s = 0.0, k = 0
        while (Ind[i][k] != -1)
        {
                s += Dat[i][k]*u[Ind[i][k]];
                k++;
                if (k >= b) break;
        }
        v[i] = s;
}
```

De esta forma, al tomar en cuenta la operación de multiplicación de una matriz por un vector, donde el renglón de la matriz involucrado en la multiplicación queda generalmente en una región contigua del Cache, se hace óptima la operación de multiplicación de matriz por vector.

2.5 Cálculo SVD y Pseudoinversa

Dada la descomposición singular de una matriz $\underline{\underline{A}}$,

$$\underline{\underline{A}} = \underline{\underline{U\Sigma V^*}}$$

la pseudoinversa de Moore-Penrose viene dada por

$$\underline{\underline{A}}^+ = \underline{\underline{V\Sigma}}^+\underline{\underline{U}}^*$$

Descomposición de Valores Singulares (SVD) la descomposición en valores singulares de una matriz es una especie de cambio de coordenadas que simplifica la matriz, una generalización de la diagonalización.

Diagonalización de matrices si una matriz cuadrada $\underline{\underline{A}}$ es diagonalizable, entonces existe una matriz $\underline{\underline{P}}$ tal que

$$\underline{\underline{A}} = \underline{\underline{PDP^{-1}}}$$

donde la matriz $\underline{\underline{D}}$ es diagonal. Podrías pensar en $\underline{\underline{P}}$ como un cambio de coordenadas que hace que la acción de $\underline{\underline{A}}$ sea lo más simple posible. Los elementos en la diagonal de $\underline{\underline{D}}$ son los valores propios de $\underline{\underline{A}}$ y las columnas de $\underline{\underline{P}}$ son los vectores propios correspondientes.

Lamentablemente no todas las matrices se pueden diagonalizar. La descomposición de valores singulares es una forma de hacer algo así como la diagonalización de cualquier matriz, incluso las que no son cuadradas.

Generalización a SVD la descomposición de valores singulares generaliza la diagonalización. La matriz $\underline{\underline{\Sigma}}$ en SVD es análoga a $\underline{\underline{D}}$ en diagonalización. $\underline{\underline{\Sigma}}$ es diagonal, aunque puede que no sea cuadrada. Las matrices a cada lado de $\underline{\underline{\Sigma}}$ son análogas a la matriz $\underline{\underline{P}}$ en diagonalización, aunque ahora hay dos matrices diferentes y no son necesariamente inversas entre sí. Las matrices $\underline{\underline{U}}$ y $\underline{\underline{V}}$ son cuadradas, pero no necesariamente de la misma dimensión.

Los elementos a lo largo de la diagonal de $\underline{\underline{\Sigma}}$ no son necesariamente valores propios sino valores singulares, que son una generalización de los valores propios. De manera similar, las columnas de $\underline{\underline{U}}$ y $\underline{\underline{V}}$ no son necesariamente vectores propios sino vectores singulares izquierdos y vectores singulares derechos, respectivamente.

El superíndice de estrella indica transposición conjugada. Si una matriz tiene todos los componentes reales, entonces la transpuesta conjugada es solo la transpuesta. Pero si la matriz tiene entradas complejas, se toma el conjugado y se transpone cada entrada.

Las matrices $\underline{\underline{U}}$ y $\underline{\underline{V}}$ son unitarias. Una matriz $\underline{\underline{M}}$ es unitaria si su inversa es su transpuesta conjugada, es decir,

$$\underline{\underline{M}}^*\underline{\underline{M}} = \underline{\underline{MM}}^* = \underline{\underline{I}}.$$

Pseudoinversa y SVD la pseudoinversa (Moore-Penrose) de una matriz generaliza la noción de inversa, algo así como la forma en que SVD generalizó la diagonalización. No todas las matrices tienen una inversa, pero todas las matrices tienen una pseudoinversa, incluso las matrices no cuadradas.

Calcular la pseudoinversa a partir del SVD es sencillo. Si

$$\underline{\underline{A}} = \underline{\underline{U\Sigma V}}^*$$

entonces

$$\underline{\underline{A}}^+ = \underline{\underline{V\Sigma}}^+\underline{\underline{U}}^*$$

donde $\underline{\underline{\Sigma}}^+$ se forma a partir de $\underline{\underline{\Sigma}}$ tomando el recíproco de todos los elementos distintos de cero, dejando todos los ceros solos y haciendo que la matriz tenga la forma correcta: si $\underline{\underline{\Sigma}}$ es una matriz de m por n, entonces $\underline{\underline{\Sigma}}^+$ debe ser una matriz de n por m.

Calcular SVD en Python Sea

$$\underline{\underline{A}} = \begin{bmatrix} 2 & -1 & 0 \\ 4 & 3 & -2 \end{bmatrix}$$

la descomposición en valores singulares de $\underline{\underline{A}}$ es

$$\begin{bmatrix} \frac{1}{\sqrt{26}} & -\frac{5}{\sqrt{26}} \\ \frac{5}{\sqrt{26}} & \frac{1}{\sqrt{26}} \end{bmatrix} \begin{bmatrix} \sqrt{30} & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix} \begin{bmatrix} \frac{11}{\sqrt{195}} & \frac{7}{\sqrt{195}} & -\sqrt{\frac{5}{39}} \\ -\frac{3}{\sqrt{26}} & 2\sqrt{\frac{2}{13}} & -\frac{1}{\sqrt{26}} \\ \frac{1}{\sqrt{30}} & \sqrt{\frac{2}{15}} & \sqrt{\frac{5}{6}} \end{bmatrix}$$

A continuación calculamos la descomposición del valor singular en Python (NumPy).

```
import numpy as np
A = np.matrix([[2, -1, 0],[4,3,-2]])
print(A)
```

```
[[ 2 -1 0]
[ 4 3 -2]]
```

```
u, s, vt = np.linalg.svd(A, full_matrices=True)
print(u)
print(s)
print(vt)
```

```
[[ 0.19611614 0.98058068]
[ 0.98058068 -0.19611614]]
[5.47722558 2. ]
[[ 0.78772636 0.50128041 -0.35805744]
[ 0.58834841 -0.78446454 0.19611614]
[ 0.18257419 0.36514837 0.91287093]]
```

Tengamos en cuenta que *np.linalg.svd* devuelve la transpuesta de $\underline{\underline{V}}$, no la $\underline{\underline{V}}$ en la definición de descomposición de valores singulares.

Además, el objeto s no es la matriz diagonal $\underline{\underline{\sum}}$ sino un vector que contiene sólo los elementos diagonales, es decir, sólo los valores singulares. Esto puede ahorrar mucho espacio si la matriz es grande. El método *NumPy svd* tiene otras opciones relacionadas con la eficiencia en las que no entraré aquí.

Podemos verificar que el SVD es correcto convirtiendo s nuevamente en una matriz

```
ss = np.matrix([[s[0], 0, 0], [0, s[1], 0]])
print(ss)
```

```
[[5.47722558 0. 0. ]
[0. 2. 0. ]]
```

y multiplicando los componentes

```
a = u*ss*vt
print(a)
```

```
[[ 2.00000000e+00 -1.00000000e+00 -4.05114794e-16]
[ 4.00000000e+00 3.00000000e+00 -2.00000000e+00]]
```

Esto devuelve la matriz $\underline{\underline{A}}$, con precisión de punto flotante. Dado que Python realiza cálculos de punto flotante, no cálculos simbólicos, el cero en $\underline{\underline{A}}$ se convierte en $-4.05114794e-16$.

Calcular Pseudoinverso en Python el pseudoinverso se puede calcular en NumPy con np.linalg.pinv

```
si = np.linalg.pinv(a)
print(si)
```

```
[[ 0.31666667 0.08333333]
[-0.36666667 0.16666667]
[ 0.08333333 -0.08333333]]
```

esto devuelve el resultado hasta precisión de punto flotante, para ello hacemos:

```
print(a*si)
```

```
[[ 1.00000000e+00 1.54033727e-16]
[-9.25185854e-18 1.00000000e+00]]
```

3 Solución de Grandes Sistemas de Ecuaciones Lineales

Diez Sorpresas del Álgebra Lineal Numérica aquí hay diez cosas sobre el álgebra lineal numérica que pueden resultarle sorprendentes si no está familiarizado con el campo:

- El álgebra lineal numérica aplica matemáticas muy avanzadas para resolver problemas que se pueden plantear con matemáticas de secundaria.
- Las aplicaciones prácticas a menudo requieren resolver enormes sistemas de ecuaciones, millones o incluso miles de millones de variables.
- El corazón de Google es un enorme problema de álgebra lineal. PageRank es esencialmente un problema de valores propios.
- La eficiencia de la resolución de sistemas de ecuaciones muy grandes se ha beneficiado al menos tanto de los avances en los algoritmos como de la ley de Moore.
- Muchos problemas prácticos (optimización, ecuaciones diferenciales, procesamiento de señales, etc.) se reducen a resolver sistemas lineales, incluso cuando los problemas originales no son lineales. El Software de elementos finitos, diferencias finitas y volumen finito por ejemplo, dedica casi todo su tiempo a resolver ecuaciones lineales.
- A veces, un sistema de un millón de ecuaciones se puede resolver en una PC común y corriente en menos de un milisegundo, dependiendo de la estructura de las ecuaciones.
- Los métodos iterativos, métodos que en teoría requieren un número infinito de pasos para resolver un problema, suelen ser más rápidos y precisos que los métodos directos, métodos que en teoría producen una respuesta exacta en un número finito de pasos.
- Hay muchos teoremas que limitan el error en soluciones producidas en computadoras reales. Es decir, los teoremas no sólo limitan el error de cálculos hipotéticos realizados en aritmética exacta, sino que limitan el error de la aritmética realizada en aritmética de punto flotante en Hardware de computadora.

- Casi nunca es necesario calcular la inversa de una matriz.
- Existe un Software notablemente maduro para álgebra lineal numérica. Personas brillantes han trabajado en este Software durante muchos años.

Los modelos matemáticos de muchos sistemas en Ciencia e Ingeniería requieren el procesamiento de sistemas algebraicos de gran escala. La solución de este sistema lineal puede ser expresado en la forma matricial siguiente

$$\underline{\underline{A}}\underline{u} = \underline{f} \tag{3.1}$$

donde la matriz $\underline{\underline{A}}$ es de tamaño $n \times n$, está puede ser densa, bandada (de banda es b), dispersa (de máximo número de columnas ocupadas es b) o rala.

Los métodos de resolución del sistema algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ se clasifican en dos grandes grupos (véase [4], [5], [6] y [8]):

- En los métodos directos la solución $\underline{u}$ se obtiene en un número fijo de pasos y sólo están sujetos a los errores de redondeo.
- En los métodos iterativos, se realizan iteraciones para aproximarse a la solución $\underline{u}$ aprovechando las características propias de la matriz $\underline{\underline{A}}$, tratando de usar un menor número de pasos que en un método directo.

Por lo general, es conveniente usar bibliotecas[30] para implementar de forma eficiente a los vectores, matrices -bandadas, dispersas o ralas- y resolver el sistemas lineal.

Los métodos iterativos rara vez se usan para resolver sistemas lineales de dimensión pequeña (el concepto de dimensión pequeña es muy relativo), ya que el tiempo necesario para conseguir una exactitud satisfactoria rebasa el que requieren los métodos directos. Sin embargo, en el caso de sistemas grandes con un alto porcentaje de elementos cero, son eficientes tanto en el almacenamiento en la computadora como en el tiempo que se invierte en

[30] Algunas de las bibliotecas más usadas para resolver sistemas lineales usando matrices bandadas y dispersar son PETCs, HYPRE, ATLAS, LAPACK++, LAPACK, EISPACK, LINPACK, BLAS, entre muchas otras alternativas, tanto para implementaciones secuenciales como paralelas y más recientemente para hacer uso de los procesadores CUDA en las GPU de nVidia.

su solución. Por ésta razón al resolver éstos sistemas algebraicos de ecuaciones es preferible aplicar métodos iterativos tal como Gradiente Conjugado -Conjugate Gradient Method (CGM)- o Residual Mínimo Generalizado -Generalized Minimum Residual Method (GMRES).

3.1 Métodos Directos

En los métodos directos (véase [4] y [7]), la solución $\underline{u}$ se obtiene en un número fijo de pasos y sólo están sujetos a errores de redondeo. Entre los métodos más importantes se puede considerar: Factorización LU -para matrices simétricas y no simétricas- y Factorización Cholesky -para matrices simétricas-. En todos los casos la matriz original $\underline{\underline{A}}$ es modificada y en caso de usar la Factorización LU el tamaño de la banda b crece a $2b+1$ si la factorización se realiza en la misma matriz. Los métodos aquí mencionados, se colocaron en orden descendente en cuanto al consumo de recursos computacionales y ascendente en cuanto al aumento en su eficiencia.

3.1.1 Eliminación Gausiana

Tal vez es el método más utilizado para encontrar la solución usando métodos directos. Este algoritmo sin embargo no es eficiente, ya que en general, un sistema de N ecuaciones requiere para su almacenaje en memoria de N^2 entradas para la matriz $\underline{\underline{A}}$, pero cerca de $N^3/3 + O(N^2)$ multiplicaciones y $N^3/3 + O(N^2)$ adiciones para encontrar la solución siendo muy costoso computacionalmente.

La eliminación Gausiana se basa en la aplicación de operaciones elementales a renglones o columnas de tal forma que es posible obtener matrices equivalentes.

Escribiendo el sistema de N ecuaciones lineales con N incógnitas como

$$\sum_{j=1}^{N} a_{ij}^{(0)} x_j = a_{i,n+1}^{(0)}, \quad i = 1, 2, ..., N \tag{3.2}$$

y si $a_{11}^{(0)} \neq 0$ y los pivotes $a_{ii}^{(i-1)}, i = 2, 3, ..., N$ de las demás filas, que se obtienen en el curso de los cálculos, son distintos de cero, entonces, el sistema lineal anterior se reduce a la forma triangular superior (eliminación hacia

adelante)

$$x_i + \sum_{j=i+1}^{N} a_{ij}^{(i)} x_j = a_{i,n+1}^{(i)}, \quad i = 1, 2, ..., N \tag{3.3}$$

donde

$$\begin{aligned} k &= 1, 2, ..., N; \{j = k+1, ..., N\{ \\ a_{kj}^{(k)} &= \frac{a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}}; \\ i &= k+1, ..., N+1\{ \\ a_{ij}^{(k)} &= a_{ij}^{(k-1)} - a_{kj}^{(k)} a_{ik}^{(k-1)}\}\}\} \end{aligned}$$

y las incógnitas se calculan por sustitución hacia atrás, usando las fórmulas

$$\begin{aligned} x_N &= a_{N,N+1}^{(N)}; \\ i &= N-1, N-2, ..., 1 \\ x_i &= a_{i,N+1}^{(i)} - \sum_{j=i+1}^{N} a_{ij}^{(i)} x_j. \end{aligned} \tag{3.4}$$

En algunos casos nos interesa conocer $\underline{\underline{A}}^{-1}$, por ello si la eliminación se aplica a la matriz aumentada $\underline{\underline{A}} \mid \underline{\underline{I}}$ entonces la matriz $\underline{\underline{A}}$ de la matriz aumentada se convertirá en la matriz $\underline{\underline{I}}$ y la matriz $\underline{\underline{I}}$ de la matriz aumentada será $\underline{\underline{A}}^{-1}$. Así, el sistema $\underline{\underline{A}}\underline{u} = \underline{b}$ se transformará en $\underline{u} = \underline{\underline{A}}^{-1}\underline{b}$ obteniendo la solución de $\underline{u}$.

3.1.2 Factorización LU

Sea $\underline{\underline{U}}$ una matriz triangular superior obtenida de $\underline{\underline{A}}$ por eliminación bandada. Entonces $\underline{\underline{U}} = \underline{\underline{L}}^{-1}\underline{\underline{A}}$, donde $\underline{\underline{L}}$ es una matriz triangular inferior con unos en la diagonal. Las entradas de $\underline{\underline{L}}^{-1}$ pueden obtenerse de los coeficientes $\underline{\underline{L}}_{ij}$ y pueden ser almacenados estrictamente en las entradas de la diagonal inferior de $\underline{\underline{A}}$ ya que estas ya fueron eliminadas. Esto proporciona una Factorización $\underline{\underline{LU}}$ de $\underline{\underline{A}}$ en la misma matriz $\underline{\underline{A}}$ ahorrando espacio de memoria, donde el ancho de banda cambia de b a $2b+1$.

En el algoritmo de Factorización LU, se toma como datos de entrada del sistema $\underline{\underline{A}}\underline{u} = \underline{f}$, a la matriz $\underline{\underline{A}}$, la cual será factorizada en la misma matriz,

está contendrá a las matrices $\underline{\underline{L}}$ y $\underline{\underline{U}}$ producto de la factorización, quedando el método numérico esquemáticamente como:

$$\begin{array}{l} A_{ii} = 1, \text{ para } i = 1, ..., N \\ \text{Para } J = 1, 2, ..., N \ \{ \\ \quad \text{Para } i = 1, 2, ..., j \\ \quad\quad A_{ij} = A_{ij} - \sum_{k=1}^{i-1} A_{ik} A_{kj} \\ \quad \text{Para } i = j+1, j+2, ..., N \\ \quad\quad A_{ij} = \frac{1}{A_{jj}} \left(A_{ij} - \sum_{k=1}^{j-1} A_{ik} A_{kj} \right) \\ \} \end{array} \tag{3.5}$$

El problema original $\underline{\underline{A}}\,\underline{u} = \underline{f}$ se escribe como $\underline{\underline{LU}}\,\underline{u} = \underline{f}$, donde la búsqueda de la solución $\underline{u}$ se reduce a la solución sucesiva de los sistemas lineales triangulares

$$\underline{\underline{L}}\,\underline{y} = \underline{f} \quad \text{y} \quad \underline{\underline{U}}\,\underline{u} = \underline{y}. \tag{3.6}$$

i,.e.

$$\underline{\underline{L}}\,\underline{y} = \underline{f} \Leftrightarrow \left\{ \begin{array}{l} y_1 = f_1 / l_{11} \\ y_i = \frac{1}{A_{ii}} \left(f_i - \sum_{j=1}^{i-1} A_{ii} y_i \right) \text{ para toda } i = 1, ..., n \end{array} \right. \tag{3.7}$$

y

$$\underline{\underline{U}}\,\underline{u} = \underline{y} \Leftrightarrow \left\{ \begin{array}{l} x_n = y_n / u_{nn} \\ x_i = \frac{1}{A_{ii}} \left(y_i - \sum_{j=i+1}^{n} A_{ij} x_j \right) \text{ para toda } i = n-1, ..., 1 \end{array} \right. \tag{3.8}$$

La descomposición $\underline{\underline{LU}}$ requiere $N^3/3$ operaciones aritméticas para la matriz llena, pero sólo Nb^2 operaciones aritméticas para la matriz con un ancho de banda de b siendo esto más económico computacionalmente.

3.1.3 Factorización Cholesky

Cuando la matriz es simétrica y definida positiva, se obtiene la descomposición $\underline{\underline{LU}}$ de la matriz $\underline{\underline{A}} = \underline{\underline{LDU}} = \underline{\underline{LDL}}^T$ donde $\underline{\underline{D}} = diag(\underline{\underline{U}})$ es la diagonal con entradas positivas.

En el algoritmo de Factorización Cholesky, se toma como datos de entrada del sistema $\underline{\underline{A}}\underline{u} = \underline{f}$, a la matriz $\underline{\underline{A}}$, la cual será factorizada en la misma matriz y contendrá a las matrices $\underline{\underline{L}}$ y $\underline{\underline{L}}^T$ producto de la factorización, quedando el método numérico esquemáticamente como:

$$\begin{aligned} &\text{para } i = 1, 2, ..., n \text{ y } j = i+1, ..., n \\ &A_{ii} = \sqrt{\left(A_{ii} - \sum_{k=1}^{i-1} A_{ik}^2 \right)} \\ &A_{ji} = \left(A_{ji} - \sum_{k=1}^{i-1} A_{jk} A_{ik} \right) / A_{ii} \end{aligned} \tag{3.9}$$

El problema original $\underline{\underline{A}}\underline{u} = \underline{f}$ se escribe como $\underline{\underline{L}}\underline{\underline{L}}^T\underline{u} = \underline{b}$, donde la búsqueda de la solución $\underline{u}$ se reduce a la solución sucesiva de los sistemas lineales triangulares

$$\underline{\underline{L}}\underline{y} = \underline{f} \quad \text{y} \quad \underline{\underline{L}}^T\underline{u} = \underline{y} \tag{3.10}$$

usando la formulación equivalente dada por las Ec.(3.7) y (3.8) para la descomposición LU.

La mayor ventaja de esta descomposición es que, en el caso en que es aplicable, el costo de cómputo es sustancialmente reducido, ya que requiere de $N^3/6$ multiplicaciones y $N^3/6$ sumas.

3.2 Métodos Iterativos

En los métodos iterativos, se realizan iteraciones para aproximarse a la solución $\underline{u}$ aprovechando las características propias de la matriz $\underline{\underline{A}}$, tratando de usar un menor número de pasos que en un método directo (véase [4] y [7]).

En los métodos iterativos tales como Jacobi, Gauss-Seidel y de Relajación Sucesiva (SOR) en el cual se resuelve el sistema lineal

$$\underline{\underline{A}}\underline{u} = \underline{f} \tag{3.11}$$

comienza con una aproximación inicial $\underline{u}^0$ a la solución $\underline{u}$ y genera una sucesión de vectores $\left\{u^k\right\}_{k=1}^{\infty}$ que converge a $\underline{u}$. Los métodos iterativos traen

consigo un proceso que convierte el sistema $\underline{\underline{A}}\underline{u} = \underline{f}$ en otro equivalente mediante la iteración de punto fijo de la forma $\underline{u} = \underline{\underline{T}}\underline{u} + \underline{c}$ para alguna matriz fija $\underline{\underline{T}}$ y un vector $\underline{c}$. Luego de seleccionar el vector inicial $\underline{u}^0$ la sucesión de los vectores de la solución aproximada se genera calculando

$$\underline{u}^k = \underline{\underline{T}}\underline{u}^{k-1} + \underline{c} \quad \forall k = 1, 2, 3, ... \tag{3.12}$$

La convergencia a la solución la garantiza el siguiente teorema (véase [8]).

Teorema 1 *Si* $\left\|\underline{\underline{T}}\right\| < 1$, *entonces el sistema lineal* $\underline{u} = \underline{\underline{T}}\underline{u} + \underline{c}$ *tiene una solución única* $\underline{u}^*$ *y las iteraciones* $\underline{u}^k$ *definidas por la fórmula* $\underline{u}^k = \underline{\underline{T}}\underline{u}^{k-1} + \underline{c} \quad \forall k = 1, 2, 3, ...$ *convergen hacia la solución exacta* $\underline{u}^*$ *para cualquier aproximación inicial* $\underline{u}^0$.

Nótese que, mientras menor sea la norma de la matriz $\underline{\underline{T}}$, más rápida es la convergencia, en el caso cuando $\left\|\underline{\underline{T}}\right\|$ es menor que uno, pero cercano a uno, la convergencia es lenta y el número de iteraciones necesario para disminuir el error depende significativamente del error inicial. En este caso, es deseable proponer al vector inicial $\underline{u}^0$ de forma tal que sea mínimo el error inicial. Sin embargo, la elección de dicho vector no tiene importancia si la $\left\|\underline{\underline{T}}\right\|$ es pequeña, ya que la convergencia es rápida.

Como es conocido, la velocidad de convergencia de los métodos iterativos dependen de las propiedades espectrales de la matriz de coeficientes del sistema de ecuaciones, cuando el operador diferencial $\mathcal{L}$ de la ecuación del problema a resolver es auto-adjunto se obtiene una matriz simétrica y positivo definida y el número de condicionamiento de la matriz $\underline{\underline{A}}$, es por definición

$$cond(\underline{\underline{A}}) = \frac{\lambda_{\max}}{\lambda_{\min}} \geq 1 \tag{3.13}$$

donde $\lambda_{\max}$ y $\lambda_{\min}$ es el máximo y mínimo de los eigen-valores de la matriz $\underline{\underline{A}}$. Si el número de condicionamiento es cercano a 1 los métodos numéricos al solucionar el problema convergerá en pocas iteraciones, en caso contrario se requerirán muchas iteraciones.

Frecuentemente al usar el método de Elemento Finito, Diferencias Finitas, entre otros, se tiene una velocidad de convergencia de $O\left(\frac{1}{h^2}\right)$ y en el caso de métodos de descomposición de dominio sin precondicionar se tiene una velocidad de convergencia de $O\left(\frac{1}{h}\right)$, donde h es la máxima distancia de separación entre nodos continuos de la partición, es decir, que poseen una pobre velocidad de convergencia cuando $h \to 0$ (véase [2], [3], [4] y [8]).

Los métodos aquí mencionados se colocaron en orden descendente en cuanto al consumo de recursos computacionales y ascendente en cuanto al aumento en la eficiencia en su desempeño, describiéndose a continuación:

3.2.1 Jacobi

Si todos los elementos de la diagonal principal de la matriz $\underline{\underline{A}}$ son diferentes de cero $a_{ii} \neq 0$ para $i = 1, 2, ...n$. Podemos dividir la $i-$ésima ecuación del sistema lineal (3.11) por a_{ii} para $i = 1, 2, ...n$, y después trasladamos todas las incógnitas, excepto x_i, a la derecha, se obtiene el sistema equivalente

$$\underline{u} = \underline{\underline{B}}\underline{u} + \underline{d} \tag{3.14}$$

donde

$$d_i = \frac{b_i}{a_{ii}} \quad \text{y} \quad B = \{b_{ij}\} = \begin{cases} -\frac{a_{ij}}{a_{ii}} & \text{si } j \neq i \\ 0 & \text{si } j = i \end{cases}.$$

Las iteraciones del método de Jacobi están definidas por la fórmula

$$x_i = \sum_{j=1}^{n} b_{ij} x_j^{(k-1)} + d_i \tag{3.15}$$

donde $x_i^{(0)}$ son arbitrarias $(i = 1, 2, ...n; k = 1, 2, ...)$.

También el método de Jacobi se puede expresar en términos de matrices. Supongamos por un momento que la matriz $\underline{\underline{A}}$ tiene la diagonal unitaria, esto es $diag(\underline{\underline{A}}) = \underline{\underline{I}}$. Si descomponemos $\underline{\underline{A}} = \underline{\underline{I}} - \underline{\underline{B}}$, entonces el sistema dado por la Ecs. (3.11) se puede reescribir como

$$\left(\underline{\underline{I}} - \underline{\underline{B}}\right) \underline{u} = \underline{b}. \tag{3.16}$$

Para la primera iteración asumimos que $\underline{k}=\underline{b}$; entonces la última ecuación se escribe como $\underline{u} = \underline{\underline{B}}\underline{u}+\underline{k}$. Tomando una aproximación inicial $\underline{u}^0$, podemos obtener una mejor aproximación reemplazando $\underline{u}$ por la más reciente aproximación de $\underline{u}^m$. Esta es la idea que subyace en el método Jacobi. El proceso iterativo queda como

$$\underline{u}^{m+1} = \underline{\underline{B}}\underline{u}^m + \underline{k}. \tag{3.17}$$

La aplicación del método a la ecuación de la forma $\underline{\underline{A}}\underline{u} = \underline{b}$, con la matriz $\underline{\underline{A}}$ no cero en los elementos diagonales, se obtiene multiplicando la Ec. (3.11) por $D^{-1} = \left[diag(\underline{\underline{A}})\right]^{-1}$ obteniendo

$$\underline{\underline{B}} = \underline{\underline{I}} - \underline{\underline{D}}^{-1}\underline{\underline{A}}, \quad \underline{k} = \underline{\underline{D}}^{-1}\underline{b}. \tag{3.18}$$

3.2.2 Gauss-Seidel

Este método es una modificación del método Jacobi, en el cual una vez obtenido algún valor de $\underline{u}^{m+1}$, este es usado para obtener el resto de los valores utilizando los valores más actualizados de $\underline{u}^{m+1}$. Así, la Ec. (3.17) puede ser escrita como

$$u_i^{m+1} = \sum_{j<i} b_{ij} u_j^{m+1} + \sum_{j>i} b_{ij} u_j^m + k_i. \tag{3.19}$$

Notemos que el método Gauss-Seidel requiere el mismo número de operaciones aritméticas por iteración que el método de Jacobi. Este método se escribe en forma matricial como

$$\underline{u}^{m+1} = \underline{\underline{E}}\,\underline{u}^{m+1} + \underline{\underline{F}}\,\underline{u}^m + \underline{k} \tag{3.20}$$

donde $\underline{\underline{E}}$ y $\underline{\underline{F}}$ son las matrices triangular superior e inferior respectivamente. Este método mejora la convergencia con respecto al método de Jacobi en un factor aproximado de 2.

3.2.3 Richardson

Escribiendo el método de Jacobi como

$$\underline{u}^{m+1} - \underline{u}^m = \underline{b} - \underline{\underline{A}}\,\underline{u}^m \tag{3.21}$$

entonces el método Richardson se genera al incorporar la estrategia de sobrerrelajación de la forma siguiente

$$\underline{u}^{m+1} = \underline{u}^m + \omega\left(\underline{b} - \underline{\underline{A}}\,\underline{u}^m\right). \tag{3.22}$$

El método de Richardson se define como

$$\underline{u}^{m+1} = \left(\underline{\underline{I}} - \omega\underline{\underline{A}}\right)\underline{u}^m + \omega\underline{b} \tag{3.23}$$

en la práctica encontrar el valor de ω puede resultar muy costoso computacionalmente y las diversas estrategias para encontrar ω dependen de las características propias del problema, pero este método con un valor ω óptimo resulta mejor que el método de Gauss-Seidel.

3.2.4 Relajación Sucesiva

Partiendo del método de Gauss-Seidel y sobrerrelajando este esquema, obtenemos

$$u_i^{m+1} = (1-\omega)\, u_i^m + \omega \left[\sum_{j=1}^{i-1} b_{ij} u_j^{m+1} + \sum_{j=i+1}^{N} b_{ij} u_j^m + k_i \right] \tag{3.24}$$

y cuando la matriz $\underline{\underline{A}}$ es simétrica con entradas en la diagonal positivas, éste método converge si y sólo si $\underline{\underline{A}}$ es definida positiva y $\omega \in (0,2)$. En la práctica encontrar el valor de ω puede resultar muy costoso computacionalmente y las diversas estrategias para encontrar ω dependen de las características propias del problema.

Espacio de Krylov Los métodos Jacobi, Gauss-Seidel y de Relajación Sucesiva (SOR) son usualmente menos eficientes que los métodos discutidos en el resto de esta sección basados en el espacio de Krylov (véase [12] y [7]). Estos métodos minimizan, en la k-ésima iteración alguna medida de error sobre el espacio afín $\underline{x_0} + \mathcal{K}_k$, donde $\underline{x_0}$ es la iteración inicial y $\mathcal{K}_k$ es el k-ésimo subespacio de Krylov

$$\mathcal{K}_k = Generado \left\{ \underline{r_0}, \underline{\underline{A}}\underline{r_0}, ..., \underline{\underline{A}}^{k-1}\underline{r_0} \right\} \text{ para } k \geq 1. \tag{3.25}$$

El residual es $\underline{r} = \underline{b} - \underline{\underline{A}}\underline{x}$, tal $\left\{ \underline{r_k} \right\}_{k \geq 0}$ denota la sucesión de residuales

$$\underline{r_k} = \underline{b} - \underline{\underline{A}}\underline{x_k}. \tag{3.26}$$

Entre los métodos más usados definidos en el espacio de Krylov para el tipo de problemas tratados en el presente trabajo se puede considerar: Método de Gradiente Conjugado -para matrices simétricas- y GMRES -para matrices no simétricas-.

El método del Gradiente Conjugado ha recibido mucha atención en su uso al resolver ecuaciones diferenciales parciales y ha sido ampliamente utilizado en años recientes por la notoria eficiencia al reducir considerablemente en número de iteraciones necesarias para resolver el sistema algebraico de ecuaciones. Aunque los pioneros de este método fueron Hestenes y Stiefel (1952), el interés actual arranca a partir de que Reid (1971) lo planteara como un método iterativo, que es la forma en que se le usa con mayor frecuencia en la actualidad, esta versión está basada en el desarrollo hecho en [10].

La idea básica en que descansa el método del Gradiente Conjugado consiste en construir una base de vectores ortogonales y utilizarla para realizar la búsqueda de la solución en forma más eficiente. Tal forma de proceder generalmente no sería aconsejable porqué la construcción de una base ortogonal utilizando el procedimiento de Gramm-Schmidt requiere, al seleccionar cada nuevo elemento de la base, asegurar su ortogonalidad con respecto a cada uno de los vectores construidos previamente. La gran ventaja del método de Gradiente Conjugado radica en que cuando se utiliza este procedimiento, basta con asegurar la ortogonalidad de un nuevo miembro con respecto al último que se ha construido, para que automáticamente esta condición se cumpla con respecto a todos los anteriores.

Definición 2 *Una matriz* $\underline{\underline{A}}$ *es llamada positiva definida si todos sus eigenvalores tienen parte real positiva o equivalentemente, si* $\underline{u}^T \underline{\underline{A}} \underline{u}$ *tiene parte real positiva para* $\underline{u} \in \mathbb{C} \backslash \{0\}$ *. Notemos en este caso que*

$$\underline{u}^T \underline{\underline{A}} \underline{u} = \underline{u}^T \frac{\underline{\underline{A}} + \underline{\underline{A}}^T}{2} \underline{u} > 0, \ \ con \ \underline{u} \in \mathbb{R}^n \backslash \{0\} .$$

3.2.5 Método de Gradiente Conjugado

Si la matriz generada por la discretización es simétrica -$\underline{\underline{A}} = \underline{\underline{A}}^T$- y definida positiva -$\underline{u}^T \underline{\underline{A}} \underline{u} > 0$ para todo $\underline{u} \neq 0$-, entonces es aplicable el método de Gradiente Conjugado -Conjugate Gradient Method (CGM)-. La idea básica en que descansa el método del Gradiente Conjugado consiste en construir una base de vectores ortogonales espacio de Krylov $\mathcal{K}_n \left(\underline{\underline{A}}, \underline{v}^n \right)$ y utilizarla para realizar la búsqueda de la solución en forma lo más eficiente posible.

Tal forma de proceder generalmente no sería aconsejable porqué la construcción de una base ortogonal utilizando el procedimiento de Gram-Schmidt requiere, al seleccionar cada nuevo elemento de la base, asegurar su ortogonalidad con respecto a cada uno de los vectores construidos previamente. La gran ventaja del método de Gradiente Conjugado radica en que cuando se utiliza este procedimiento, basta con asegurar la ortogonalidad de un nuevo miembro con respecto al último que se ha construido, para que automáticamente esta condición se cumpla con respecto a todos los anteriores.

En el algoritmo de Gradiente Conjugado, se toma a la matriz $\underline{\underline{A}}$ como simétrica y positiva definida, y como datos de entrada del sistema

$$\underline{\underline{A}} \underline{u} = \underline{f} \tag{3.27}$$

el vector de búsqueda inicial $\underline{u}^0$ y se calcula $\underline{r}^0 = \underline{f} - \underline{\underline{A}}\underline{u}^0$, $\underline{p}^0 = \underline{r}^0$, quedando el método numérico esquemáticamente como:

$$\begin{array}{l} \alpha^n = \frac{\langle \underline{p}^n, \underline{p}^n \rangle}{\langle \underline{p}^n, \underline{\underline{A}}\underline{p}^n \rangle} \\ \underline{u}^{n+1} = \underline{u}^n + \alpha^n \underline{p}^n \\ \underline{r}^{n+1} = \underline{r}^n - \alpha^n \underline{\underline{A}}\underline{p}^n \\ \text{Prueba de convergencia} \\ \beta^n = \frac{\langle \underline{r}^{n+1}, \underline{r}^{n+1} \rangle}{\langle \underline{r}^n, \underline{r}^n \rangle} \\ \underline{p}^{n+1} = \underline{r}^{n+1} + \beta^n \underline{p}^n \\ n = n+1 \end{array} \tag{3.28}$$

donde $\langle \cdot, \cdot \rangle = (\cdot, \cdot)$ será el producto interior adecuado al sistema lineal en particular, la solución aproximada será $\underline{u}^{n+1}$ y el vector residual será $\underline{r}^{n+1}$.

En la implementación numérica y computacional del método es necesario realizar la menor cantidad de operaciones posibles por iteración, en particular en $\underline{\underline{A}}\underline{p}^n$, una manera de hacerlo queda esquemáticamente como:

Dado el vector de búsqueda inicial $\underline{u}$, calcula $\underline{r} = \underline{f} - \underline{\underline{A}}\underline{u}$, $\underline{p} = \underline{r}$ y $\mu = \underline{r} \cdot \underline{r}$.

Para $n = 1, 2, ...,$Mientras $(\mu < \varepsilon)$ {
- $\underline{v} = \underline{\underline{A}}\underline{p}$
- $\alpha = \frac{\mu}{\underline{p} \cdot \underline{v}}$
- $\underline{u} = \underline{u} + \alpha \underline{p}$
- $\underline{r} = \underline{r} - \alpha \underline{v}$
- $\mu' = \underline{r} \cdot \underline{r}$
- $\beta = \frac{\mu'}{\mu}$
- $\underline{p} = \underline{r} + \beta \underline{p}$
- $\mu = \mu'$

}

La solución aproximada será $\underline{u}$ y el vector residual será $\underline{r}$.

Si se denota con $\{\lambda_i, V_i\}_{i=1}^N$ las eigen-soluciones de $\underline{\underline{A}}$, i.e. $\underline{\underline{A}}V_i = \lambda_i V_i$, $i = 0, 1, 2, ..., N$. Ya que la matriz $\underline{\underline{A}}$ es simétrica, los eigen-valores son reales y se pueden ordenar $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_N$. Se define el número de condición por $Cond(\underline{\underline{A}}) = \lambda_N / \lambda_1$ y la norma de la energía asociada a $\underline{\underline{A}}$ por $\|\underline{u}\|^2_{\underline{\underline{A}}} = \underline{u} \cdot \underline{\underline{A}}\underline{u}$ entonces

$$\left\| \underline{u} - \underline{u}^k \right\|_{\underline{\underline{A}}} \leq \left\| \underline{u} - \underline{u}^0 \right\|_{\underline{\underline{A}}} \left[\frac{1 - \sqrt{Cond(\underline{\underline{A}})}}{1 + \sqrt{Cond(\underline{\underline{A}})}} \right]^{2k}. \tag{3.29}$$

El siguiente teorema da idea del espectro de convergencia del sistema $\underline{\underline{A}}\underline{u} = \underline{b}$ para el método de Gradiente Conjugado.

Teorema 3 *Sea* $\kappa = cond(\underline{\underline{A}}) = \frac{\lambda_{\max}}{\lambda_{\min}} \geq 1$, *entonces el método de Gradiente Conjugado satisface la* $\underline{\underline{A}}-$*norma del error dado por*

$$\frac{\|e^n\|}{\|e^0\|} \leq \frac{2}{\left[\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^n + \left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^{-n}\right]} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^n \tag{3.30}$$

donde $\underline{e}^m = \underline{u} - \underline{u}^m$ *del sistema* $\underline{\underline{A}}\underline{u} = \underline{b}$.

Nótese que para κ grande se tiene que

$$\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \simeq 1 - \frac{2}{\sqrt{\kappa}} \tag{3.31}$$

tal que

$$\|\underline{e}^n\|_{\underline{\underline{A}}} \simeq \left\|\underline{e}^0\right\|_{\underline{\underline{A}}} \exp\left(-2\frac{n}{\sqrt{\kappa}}\right) \tag{3.32}$$

de lo anterior se puede esperar un espectro de convergencia del orden de $O(\sqrt{\kappa})$ iteraciones (véase [8] y [12]).

3.2.6 Gradiente Conjugado Precondicionado

Cuando la matriz $\underline{\underline{A}}$ es simétrica y definida positiva se puede escribir como

$$\lambda_1 \leq \frac{\underline{u}\underline{\underline{A}} \cdot \underline{u}}{\underline{u} \cdot \underline{u}} \leq \lambda_n \tag{3.33}$$

y tomando la matriz $\underline{\underline{C}}^{-1}$ como un precondicionador de $\underline{\underline{A}}$ con la condición de que

$$\lambda_1 \leq \frac{\underline{u}\underline{\underline{C}}^{-1}\underline{\underline{A}} \cdot \underline{u}}{\underline{u} \cdot \underline{u}} \leq \lambda_n \tag{3.34}$$

entonces la Ec. (3.27) se puede escribir como

$$\underline{\underline{C}}^{-1}\underline{\underline{A}}\underline{u} = \underline{\underline{C}}^{-1}b \tag{3.35}$$

donde $\underline{\underline{C}}^{-1}\underline{\underline{A}}$ es también simétrica y definida positiva en el producto interior $\langle \underline{u}, \underline{v} \rangle = \underline{u} \cdot \underline{\underline{C}}\underline{v}$, porque

$$\begin{aligned} \left\langle \underline{u}, \underline{\underline{C}}^{-1}\underline{\underline{A}}\underline{v} \right\rangle &= \underline{u} \cdot \underline{\underline{C}} \left(\underline{\underline{C}}^{-1}\underline{\underline{A}}\underline{v} \right) \\ &= \underline{u} \cdot \underline{\underline{A}}\underline{v} \end{aligned} \tag{3.36}$$

que por hipótesis es simétrica y definida positiva en ese producto interior.

La elección del producto interior $\langle \cdot, \cdot \rangle$ quedará definido como

$$\langle \underline{u}, \underline{v} \rangle = \underline{u} \cdot \underline{\underline{C}}^{-1}\underline{\underline{A}}\underline{v} \tag{3.37}$$

por ello las Ecs. (3.28[1]) y (3.28[3]), se convierten en

$$\alpha^{k+1} = \frac{\underline{r}^k \cdot \underline{r}^k}{\underline{p}^{k+1} \cdot \underline{\underline{C}}^{-1}\underline{p}^{k+1}} \tag{3.38}$$

y

$$\beta^{k+1} = \frac{\underline{p}^k \cdot \underline{\underline{C}}^{-1}\underline{r}^k}{\underline{p}^k \cdot \underline{\underline{A}}\underline{p}^k} \tag{3.39}$$

generando el método de Gradiente Conjugado precondicionado con precondicionador $\underline{\underline{C}}^{-1}$. Es necesario hacer notar que los métodos Gradiente Conjugado y Gradiente Conjugado Precondicionado sólo difieren en la elección del producto interior.

Para el método de Gradiente Conjugado Precondicionado, los datos de entrada son un vector de búsqueda inicial $\underline{u}^0$ y el precondicionador $\underline{\underline{C}}^{-1}$. Calculandose $\underline{r}^0 = \underline{b} - \underline{\underline{A}}\underline{u}^0$, $\underline{p} = \underline{\underline{C}}^{-1}\underline{r}^0$, quedando el método esquemáticamente como:

$$\begin{aligned} \beta^{k+1} &= \frac{\underline{p}^k \cdot \underline{\underline{C}}^{-1}\underline{r}^k}{\underline{p}^k \cdot \underline{\underline{A}}\underline{p}^k} \\ \underline{p}^{k+1} &= \underline{r}^k - \beta^{k+1}\underline{p}^k \\ \alpha^{k+1} &= \frac{\underline{r}^k \cdot \underline{r}^k}{\underline{p}^{k+1} \cdot \underline{\underline{C}}^{-1}\underline{p}^{k+1}} \\ \underline{u}^{k+1} &= \underline{u}^k + \alpha^{k+1}\underline{p}^{k+1} \\ \underline{r}^{k+1} &= \underline{\underline{C}}^{-1}\underline{r}^k - \alpha^{k+1}\underline{\underline{A}}\underline{p}^{k+1}. \end{aligned} \tag{3.40}$$

Algoritmo Computacional del Método Dado el sistema $\underline{\underline{A}}\underline{u} = \underline{b}$, con la matriz $\underline{\underline{A}}$ simétrica y definida positiva de dimensión $n \times n$. La entrada al método será una elección de $\underline{u}^0$ como condición inicial, $\varepsilon > 0$ como la tolerancia del método, N como el número máximo de iteraciones y la matriz de precondicionamiento $\underline{\underline{C}}^{-1}$ de dimensión $n \times n$, el algoritmo del método de Gradiente Conjugado Precondicionado queda como:

$\underline{r} = \underline{b} - \underline{\underline{A}}\underline{u}$
$\underline{w} = \underline{\underline{C}}^{-1}\underline{r}$
$\underline{v} = (\underline{\underline{C}}^{-1})^T\underline{w}$
$\alpha = \sum_{j=1}^{n} w_j^2$
$k = 1$
Mientras que $k \leq N$

Si $\|\underline{v}\|_\infty < \varepsilon$ Salir
$\underline{x} = \underline{\underline{A}}\underline{v}$
$t = \dfrac{\alpha}{\sum_{j=1}^{n} v_j x_j}$
$\underline{u} = \underline{u} + t\underline{v}$
$\underline{r} = \underline{r} - t\underline{x}$
$\underline{w} = \underline{\underline{C}}^{-1}\underline{r}$
$\beta = \sum_{j=1}^{n} w_j^2$
Si $\|\underline{r}\|_\infty < \varepsilon$ Salir
$s = \dfrac{\beta}{\alpha}$
$\underline{v} = \left(\underline{\underline{C}}^{-1}\right)^T \underline{w} + s\underline{v}$
$\alpha = \beta$
$k = k + 1$

La salida del método será la solución aproximada $\underline{u} = (u_1, ..., u_n)$ y el residual $\underline{r} = (r_1, ..., r_n)$.

En el caso del método sin precondicionamiento, $\underline{\underline{C}}^{-1}$ es la matriz identidad, que para propósitos de optimización sólo es necesario hacer la asignación de vectores correspondiente en lugar del producto de la matriz por el vector. En el caso de que la matriz $\underline{\underline{A}}$ no sea simétrica, el método de Gradiente

Conjugado puede extenderse para soportarlas, para más información sobre pruebas de convergencia, resultados numéricos entre los distintos métodos de solución del sistema algebraico $\underline{\underline{A}}\underline{u} = \underline{b}$ generada por la discretización de un problema elíptico y cómo extender estos para matrices no simétricas ver [10] y [11].

Teorema 4 *Sean* $\underline{\underline{A}}, \underline{\underline{B}}$ *y* $\underline{\underline{C}}$ *tres matrices simétricas y positivas definidas entonces*

$$\kappa\left(\underline{\underline{C}}^{-1}\underline{\underline{A}}\right) \leq \kappa\left(\underline{\underline{C}}^{-1}\underline{\underline{B}}\right)\kappa\left(\underline{\underline{B}}^{-1}\underline{\underline{A}}\right).$$

3.2.7 Método Residual Mínimo Generalizado

Si la matriz generada por la discretización es no simétrica, entonces una opción, es el método Residual Mínimo Generalizado -Generalized Minimum Residual Method (GMRES)-, este representa una formulación iterativa común satisfaciendo una condición de optimización. La idea básica detrás del método se basa en construir una base ortonormal

$$\left\{\underline{v}^1, \underline{v}^2, ..., \underline{v}^n\right\} \tag{3.41}$$

para el espacio de Krylov $\mathcal{K}_n\left(\underline{\underline{A}}, \underline{v}^n\right)$. Para hacer $\underline{v}^{n+1}$ ortogonal a $\mathcal{K}_n\left(\underline{\underline{A}}, \underline{v}^n\right)$, es necesario usar todos los vectores previamente construidos $\{\underline{v}^{n+1j}\}_{j=1}^{n}$ -en la práctica sólo se guardan algunos vectores anteriores- en los cálculos. Y el algoritmo se basa en una modificación del método de Gram-Schmidt para la generación de una base ortonormal. Sea $\underline{\underline{V}}_n = [\underline{v}^1, \underline{v}^2, ..., \underline{v}^n]$ la cual denota la matriz conteniendo $\underline{v}^j$ en la j-ésima columna, para $j = 1, 2, ..., n$, y sea $\underline{\underline{H}}_n = [h_{i,j}], 1 \leq i, j \leq n$, donde las entradas de $\underline{\underline{H}}_n$ no especificadas en el algoritmo son cero. Entonces, $\underline{\underline{H}}_n$ es una matriz superior de Hessenberg. i.e. $h_{ij} = 0$ para $j < i - 1$, y

$$\begin{aligned} \underline{\underline{AV}}_n &= \underline{\underline{V}}_n\underline{\underline{H}}_n + h_{n+1,n}\left[0, ..., 0, \underline{v}^{n+1}\right] \\ \underline{\underline{H}}_n &= \underline{\underline{H}}_n^T\underline{\underline{AV}}_n. \end{aligned} \tag{3.42}$$

En el algoritmo del método Residual Mínimo Generalizado, la matriz $\underline{\underline{A}}$ es tomada como no simétrica, y como datos de entrada del sistema

$$\underline{\underline{A}}\underline{u} = \underline{f} \tag{3.43}$$

el vector de búsqueda inicial $\underline{u}^0$ y se calcula $\underline{r}^0 = \underline{f} - \underline{\underline{A}}\underline{u}^0$, $\beta^0 = \|r^0\|$, $\underline{v}^1 = \underline{r}^0/\beta^0$, quedando el método esquemáticamente como:

$$\begin{array}{l}
\text{Para } n = 1, 2, ..., \text{Mientras } \beta^n < \tau\beta^0 \ \{ \\
\quad \underline{w}_0^{n+1} = \underline{\underline{A}}\underline{v}^n \\
\quad \text{Para } l = 1 \text{ hasta } n \ \{ \\
\qquad h_{l,n} = \left\langle \underline{w}_l^{n+1}, \underline{v}^l \right\rangle \\
\qquad \underline{w}_{l+1}^{n+1} = \underline{w}_l^{n+1} - h_{l,n}\underline{v}^l \\
\quad \} \\
\quad h_{n+1,n} = \left\| \underline{w}_{n+1}^{n+1} \right\| \\
\quad \underline{v}^{n+1} = \underline{w}_{n+1}^{n+1}/h_{n+1,n} \\
\quad \text{Calcular } \underline{y}^n \text{ tal que } \beta^n = \left\| \beta^0 \underline{e}_1 - \underline{\underline{\hat{H}}}_n \underline{y}^n \right\| \text{ es mínima} \\
\}
\end{array} \tag{3.44}$$

donde $\underline{\underline{\hat{H}}}_n = [h_{ij}]_{1 \le i \le n+1, 1 \le j \le n}$, la solución aproximada será $\underline{u}^n = \underline{u}^0 + \underline{\underline{V}}_n \underline{y}^n$, y el vector residual será

$$\underline{r}^n = \underline{r}^0 - \underline{\underline{AV}}_n y^n = \underline{\underline{V}}_{n+1} \left(\beta^0 \underline{e}_1 - \underline{\underline{\hat{H}}}_n \underline{y}^n \right). \tag{3.45}$$

Teorema 5 *Sea $\underline{u}^k$ la iteración generada después de k iteraciones de GMRES, con residual $\underline{r}^k$. Si la matriz $\underline{\underline{A}}$ es diagonalizable, i.e. $\underline{\underline{A}} = \underline{\underline{V}}\Lambda\underline{\underline{V}}^{-1}$ donde Λ es una matriz diagonal de eigen-valores de $\underline{\underline{A}}$, y $\underline{\underline{V}}$ es la matriz cuyas columnas son los eigen-vectores, entonces*

$$\frac{\left\| r^k \right\|}{\left\| r^0 \right\|} \le \kappa(V) \min_{p_\kappa \in \Pi\kappa, p_k(0)=1} \max_{\lambda_j} |p_\kappa(\lambda_j)| \tag{3.46}$$

donde $\kappa(V) = \frac{\|\underline{\underline{V}}\|}{\|\underline{\underline{V}}^{-1}\|}$ es el número de condicionamiento de $\underline{\underline{V}}$.

3.3 Algunos Ejemplos

Sea el sistema lineal $\underline{\underline{A}}\underline{u} = \underline{f}$ dado por

$$\begin{bmatrix} 4 & 3 & 0 \\ 3 & 4 & 1 \\ 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 24 \\ 20 \\ -24 \end{bmatrix}$$

cuya solución es $\begin{bmatrix} 3 \\ 4 \\ -5 \end{bmatrix}$, si usamos métodos directos obtenemos los siguientes resultados:

- Solución usando el método Factorización LU:

$$\begin{bmatrix} +3.0000000000e+00 \\ +4.0000000000e+00 \\ -5.0000000000e+00 \end{bmatrix}$$

- Solución usando el método Tridiagonal:

$$\begin{bmatrix} +3.0000000000e+00 \\ +4.0000000000e+00 \\ -5.0000000000e+00 \end{bmatrix}$$

- Solución usando el método Choleski:

$$\begin{bmatrix} +3.0000000000e+00 \\ +4.0000000000e+00 \\ -5.0000000000e+00 \end{bmatrix}$$

- Solución usando el método de la Inversa:

$$\begin{bmatrix} +3.0000000000e+00 \\ +4.0000000000e+00 \\ -5.0000000000e+00 \end{bmatrix}$$

Si ahora usamos métodos iterativos (con tolerancia $1e-6$) obtenemos:

- Solución usando el método Jacobi, iteraciones necesarias para resolver el sistema lineal: 59

$$\begin{bmatrix} +3.0000036111e+00 \\ +4.0000042130e+00 \\ -5.0000012037e+00 \end{bmatrix}$$

- Solución usando el método Gauss-Seidel, iteraciones necesarias para resolver el sistema lineal: 25

$$\begin{bmatrix} +3.0000151461e+00 \\ +3.9999873782e+00 \\ -5.0000031554e+00 \end{bmatrix}$$

- Solución usando el método CGM, iteraciones necesarias para resolver el sistema lineal: 3

$$\begin{bmatrix} +3.0000000000e+00 \\ +4.0000000000e+00 \\ -5.0000000000e+00 \end{bmatrix}$$

Esto nos muestra que para ciertos sistemas lineales (en apariencia inofensivo) algunos métodos iterativos pueden requerir una gran cantidad de iteraciones para converger y en algunos casos, esto ni siquiera está garantizado, es por ello que debemos elegir el método más adecuado para el tipo de sistema lineal con el que se trabaje:

- En los métodos directos la solución $\underline{u}$ se obtiene en un número fijo de pasos y sólo están sujetos a los errores de redondeo.

- En los métodos iterativos, se realizan iteraciones para aproximarse a la solución $\underline{u}$ aprovechando las características propias de la matriz $\underline{\underline{A}}$, tratando de usar un menor número de pasos que en un método directo.

Por lo general, es conveniente usar bibliotecas[31] para implementar de forma eficiente a los vectores, matrices -bandadas, dispersas o ralas- y resolver el sistemas lineal.

[31] Algunas de las bibliotecas más usadas para resolver sistemas lineales usando matrices bandadas y dispersar son PETCs, HYPRE, ATLAS, LAPACK++, LAPACK, EISPACK, LINPACK, BLAS, entre muchas otras alternativas, tanto para implementaciones secuenciales como paralelas y más recientemente para hacer uso de los procesadores CUDA en las GPU de nVidia.

Los métodos iterativos rara vez se usan para resolver sistemas lineales de dimensión pequeña (el concepto de dimensión pequeña es muy relativo), ya que el tiempo necesario para conseguir una exactitud satisfactoria rebasa el que requieren los métodos directos. Sin embargo, en el caso de sistemas grandes con un alto porcentaje de elementos cero, son eficientes tanto en el almacenamiento en la computadora como en el tiempo que se invierte en su solución. Por ésta razón al resolver éstos sistemas algebraicos de ecuaciones es preferible aplicar métodos iterativos tal como Gradiente Conjugado -Conjugate Gradient Method (CGM)- o Residual Mínimo Generalizado -Generalized Minimum Residual Method (GMRES) según sea el tipo de matriz generada.

3.3.1 Usando Python

En Python es común usar *numpy* para definir un matrices y vectores además de sus operaciones relacionadas, para ello podemos usar:

```
import numpy as np
A = np.matrix([[4, 3, 0], [3, 4, 1], [0, -1, 4]])
b = np.matrix([[24], [20], [-24]])
```

Antes de resolver un sistema lineal, es conveniente validar que una matriz tenga inversa. Para que tenga inversa, la matriz no tiene que tener vectores linealmente independientes, es decir, no debe haber filas o columnas que puedan ser escritas como la combinación de otras filas o columnas. Para ello podemos usar:

```
import numpy as np
A = np.array(
[[0,1,0,0],
[0,0,1,0],
[0,1,1,0],
[1,0,0,1]]
)
lambdas, V = np.linalg.eig(A.T)
print(A[lambdas == 0, :])
```

visualizando las entradas que son linealmente independientes y cuáles no:

```
[[0 1 1 0]]
```

En Python hay diversas formas de resolver un sistema lineal, por ejemplo, si usamos el cálculo de la inversa $A**-1$:

```
import numpy as np
A = np.matrix([[4, 3, 0], [3, 4, 1], [0, -1, 4]])
b = np.matrix([[24], [20], [-24]])
if np.linalg.det(A) == 0:
    x = None
    print("No se puede resolver")
else:
    x = (A**-1)*b
    print(x)
    print(np.dot(A, x))
    print((np.dot(A, x) == b).all())
```

que nos entregará la siguiente salida:

```
[[3.]
[4.]
[-5.]]
[[24.]
[20.]
[-24]]
True
```

El primer vector es la solución del sistema lineal, la comprobación y la revisión de la igualdad entrada a entrada entre la solución y el vector b[32].

Otro ejemplo usando *linalg.solve*:

```
import numpy as np
A = np.array([[4, 3, 0], [3, 4, 1], [0, -1, 4]])
b = np.array([[24], [20], [-24]])
x = np.linalg.solve(A, b)
print(x)
```

[32]En este caso las soluciones son idénticas, pero en general pueden diferir en algunos dígitos por la pérdida de precisión, por lo que este ejemplo es solo es ilustrativo.

Otro ejemplo usando *linalg.inv*:

```
import numpy as np
A = np.array([[4, 3, 0], [3, 4, 1], [0, -1, 4]])
b = np.array([[24], [20], [-24]])
x = np.linalg.inv(A).dot(b)
print(x)
```

Otro ejemplo usando *linalg.pinv*:

```
import numpy as np
A = np.array([[4, 3, 0], [3, 4, 1], [0, -1, 4]])
b = np.array([[24], [20], [-24]])
x = np.linalg.pinv(A).dot(b)
print(x)
```

Otro ejemplo usando *linalg.qr*:

```
import numpy as np
A = np.array([[4, 3, 0], [3, 4, 1], [0, -1, 4]])
b = np.array([[24], [20], [-24]])
Q, R = np.linalg.qr(A)
y = np.dot(Q.T, b)
x = np.linalg.solve(R, y)
print(x)
```

Otro ejemplo usando sympy:

```
from sympy import symbols, Matrix, linsolve
x, y, z = symbols('x, y, z')
A = Matrix([[4, 3, 0], [3, 4, 1], [0, -1, 4]])
b = Matrix([[24], [20], [-24]])
xx = linsolve((A,b),[x, y, z])
print(xx)
```

3.3.2 Usando C++

GMM++[33] es una librería para C++ que permite definir diversos tipos de matrices y vectores además operaciones básicas de álgebra lineal. La facilidad de uso y la gran cantidad de opciones hacen que GMM++ sea una buena opción para trabajar con operaciones elementales de álgebra lineal.

Se instala en Debian Linux y/o Ubuntu como:

```
# apt install libgmm++-dev
```

Para compilar el ejemplo usar:

```
$ g++ ejemplito.cpp
```

Para ejecutar usar:

```
$ ./a.out
```

Ejemplo 1 *Un sencillo ejemplo de manejo de matrices y vectores en C++:*

```
#include <gmm/gmm.h>
#include <math.h>
int main(void)
{
int N = 100;
// Matriz densa
gmm::dense_ matrix<double> AA(N, N);
// Matriz dispersa
gmm::row_ matrix< gmm::rsvector<double> > A(N, N);
// Vectores
std::vector<double> x(N), b(N);
int i;
double P = -2 ;
double Q = 1;
double R = 1;
A(0, 0) = P; // Primer renglon de la matriz A y vector b
A(0, 1) = Q;
b[0] = P;
// Renglones intermedios de la matriz A y vector b
for(i = 1; i < N - 1; i++)
{
A(i, i - 1) = R;
A(i, i) = P;
```

[33] GMM++ [http://getfem.org/gmm.html]

```
A(i, i + 1) = Q;
b[i] = P);
}
A(N - 1, N - 2) = R; // Renglon final de la matriz A y vector b
A(N - 1, N - 1) = P;
b[N - 1] = P;
// Copia la matriz dispersa a la densa para usarla en LU
gmm::copy(A,AA);
// Visualiza la matriz y el vector
std::cout << "Matriz A"<< AA << gmm::endl;
std::cout << "Vector b"<< b << gmm::endl;
return 0;
}
```

3.4 Cómputo Paralelo

Los modelos matemáticos de muchos sistemas en Ciencia e Ingeniería requieren el procesamiento de sistemas algebraicos de gran escala. Pero cuando los tiempos de solución del sistema algebraico es excesivo o cuando un solo equipo de cómputo no puede albergar nuestro problema, entonces pensamos en el cómputo en paralelo.

La computación paralela es el uso de múltiples recursos computacionales para resolver un problema. Se distingue de la computación secuencial en que varias operaciones pueden ocurrir simultáneamente. Los sistemas de cómputo con procesamiento en paralelo surgen de la necesidad de resolver problemas complejos en un tiempo razonable, utilizando las ventajas de memoria, velocidad de los procesadores, formas de interconexión de estos y distribución de la tarea, a los que en su conjunto denominamos arquitectura en paralelo. Entenderemos por una arquitectura en paralelo a un conjunto de procesadores interconectados capaces de cooperar en la solución de un problema.

Así, para resolver un problema en particular, se usa una arquitectura o combinación de múltiples arquitecturas (topologías), ya que cada una ofrece ventajas y desventajas que tienen que ser sopesadas antes de implementar la solución del problema en una arquitectura en particular. También es necesario conocer los problemas a los que se enfrenta un desarrollador de programas que se desean correr en paralelo, como son: el partir eficientemente

un problema en múltiples tareas y cómo distribuir estas según la arquitectura en particular con que se trabaje.

El paralelismo clásico, o puesto de otra manera, el clásico uso del paralelismo, es el de diseño de programas eficientes en el ámbito científico. La simulación de problemas científicos es un área de gran importancia, los cuales requieren de una gran capacidad de procesamiento y de espacio de memoria, debido a las complejas operaciones que se deben realizar.

Tradicionalmente, los programas informáticos se han escrito para el cómputo en serie. Para resolver un problema, se construye un algoritmo y se implementa como un flujo en serie de instrucciones. Estas instrucciones se ejecutan en una unidad central de procesamiento en un ordenador. Sólo puede ejecutarse una instrucción a la vez y un tiempo después de que la instrucción ha terminado, se ejecuta la siguiente.

La computación en paralelo, por el contrario, utiliza simultáneamente múltiples elementos de procesamiento para resolver un problema. Esto se logra mediante la división del problema en partes independientes de modo que cada elemento de procesamiento pueda ejecutar su parte del algoritmo de manera simultánea con los otros. Los elementos de procesamiento son diversos e incluyen recursos tales como una computadora con múltiples procesadores, varios ordenadores en red, hardware especializado, o cualquier combinación de los anteriores.

Los programas informáticos paralelos son más difíciles de escribir que los secuenciales, porque la concurrencia introduce nuevos tipos de errores de software, siendo las condiciones de carrera las más comunes. La comunicación y sincronización entre diferentes subtareas son algunos de los mayores obstáculos para obtener un buen rendimiento del programa paralelo.

Equipo Paralelo de Memoria Compartida un multiprocesador puede verse como una computadora paralela compuesta por varios procesadores interconectados que comparten un mismo sistema de memoria (todos los equipos de cómputo actuales son de este tipo). Tienen un único espacio de direcciones para todos los procesadores. Para hacer uso de la memoria compartida (que puede ser de hasta Terabytes) por más de un procesador, se requiere hacer uso de técnicas de semáforos que mantienen la integridad de la memoria; esta arquitectura no puede crecer mucho en el número de procesadores interconectados por la saturación rápida del bus.

Equipo Paralelo de Memoria Distribuida los sistemas multicomputadoras se pueden ver como una computadora paralela en el cual cada procesador tiene su propia memoria local. En estos sistemas la memoria se encuentra distribuida y no compartida como en los sistemas multiprocesador. Los procesadores se comunican a través de paso de mensajes, ya que éstos sólo tienen acceso directo a su memoria local y no a las memorias del resto de los procesadores.

La transferencia de los datos se realiza a través de la red de interconexión que conecta un subconjunto de procesadores con otro subconjunto. La transferencia de unos procesadores a otros se realiza por múltiples transferencias entre procesadores conectados dependiendo del establecimiento de dicha red.

Dado que la memoria está distribuida entre los diferentes elementos de proceso, estos sistemas reciben el nombre de distribuidos. Por otra parte, estos sistemas son débilmente acoplados, ya que los módulos funcionan de forma casi independiente unos de otros. Este tipo de memoria distribuida es de acceso lento por ser peticiones a través de la red, pero es una forma muy efectiva de tener acceso a un gran volumen de memoria.

Equipo Paralelo de Memoria Compartida-Distribuida La tendencia actual en las máquinas paralelas es de aprovechar las facilidades de programación que ofrecen los ambientes de memoria compartida y la escalabilidad de los ambientes de memoria distribuida. En este modelo se conectan entre sí módulos de multiprocesadores, pero se mantiene la visión global de la memoria a pesar de que es distribuida. Ejemplo de este tipo de equipo son los Clusters. El desarrollo de sistemas operativos y compiladores del dominio público (Linux y Software GNU), estándares para interfaz de paso de mensajes (Message Passing Interface MPI), conexión universal a periféricos (Peripherial Component Interconnect PCI), entre otros, han hecho posible tomar ventaja de los recursos económicos computacionales de producción masiva (procesadores, discos, redes).

La principal desventaja que presenta a los proveedores de multicomputadoras es que deben satisfacer una amplia gama de usuarios, es decir, deben ser generales. Esto aumenta los costos de diseños y producción de equipos, así como los costos de desarrollo de Software que va con ellos: sistema operativo, compiladores y aplicaciones. Todos estos costos deben ser añadidos cuando se hace una venta. Por supuesto alguien que sólo necesita procesadores y un mecanismo de pase de mensajes no debería pagar por todos estos añadi-

dos que nunca usará. Estos usuarios son los que están impulsando el uso de Clusters principalmente de computadoras personales (PC)

¿Cómo Paralelizo mi Programa? El problema de todos los usuarios de métodos numéricos para solucionar sistemas lineales y su implementación computacional es: ¿cómo paralelizo mi programa?, esta pregunta no tiene una respuesta simple, depende de muchos factores, por ejemplo: en que lenguaje o paquete está desarrollado, el algoritmo usado para solucionar el problema, a que equipos paralelo tengo acceso, etc.

Algunas respuestas ingenuas son:

- Si uso lenguajes de programación compilables, puedo conseguir un compilador que permita usar directivas de compilación en equipos de memoria compartida sobre programas escritos de forma secuencial, con la esperanza que sean puestos por el compilador como programas paralelos haciendo uso de hilos, OpenMP y optimización del código generado.

- Si uso paquetes como MatLab, Julia o Python, es posible conseguir una versión del paquete que implemente bibliotecas que usen CUDAs, OpenMP o MPI para muchos de los algoritmos más usados en la programación.

- Si mi problema cabe en un sólo equipo, entonces puedo usar dos o más cores para resolver mi problema usando memoria compartida (puedo usar OpenMP, trabajar con hilos o usando paquetes como MatLab o lenguajes como Python, Fortran, C y C++, etc.), pero sólo puedo escalar para usar el máximo número de cores de un equipo (que en la actualidad puede ser del orden de 128 cores, hasta Terabytes de RAM y con almacenamiento de algunas decenas de Terabytes).

- Si mi problema cabe en la GRAM de una GPU, entonces el posible usar una tarjeta gráfica (usando paquetes como MatLab o lenguajes como Python, Fortran, C y C++, etc.), pero sólo puedo escalar a la capacidad de dichas tarjetas gráficas.

- Puedo usar un cluster que usa memoria distribuida-compartida en conjunto con tarjetas gráficas, este tipo de programación requiere una nueva expertes y generalmente implica el uso de paso de mensajes como MPI y rediseño de los algoritmos usados en nuestro programa.

Notemos primero que no todos los algoritmos son paralelizables. En cualquier caso se tienen que ver los pros y contras de la paralelización para cada caso particular. Pero es importante destacar que existen una gran cantidad de bibliotecas y paquetes que ya paralelizan la resolución de sistemas lineales[34] y no lineales.

[34] Algunas de las bibliotecas más usadas para resolver sistemas lineales usando matrices bandadas y dispersar son PETCs, HYPRE, ATLAS, LAPACK++, LAPACK, EISPACK, LINPACK, BLAS, entre muchas otras alternativas, tanto para implementaciones secuenciales como paralelas y más recientemente para hacer uso de los procesadores CUDA en las GPU de nVidia.

4 Precondicionadores

Una vía que permite mejorar la eficiencia de los métodos iterativos consiste en transformar al sistema de ecuaciones en otro equivalente, en el sentido de que posea la misma solución del sistema original pero que a su vez tenga mejores condiciones espectrales. Esta transformación se conoce como precondicionamiento y consiste en aplicar al sistema de ecuaciones una matriz conocida como precondicionador encargada de realizar el mejoramiento del número de condicionamiento.

Una amplia clase de precondicionadores han sido propuestos basados en las características algebraicas de la matriz del sistema de ecuaciones, mientras que por otro lado también existen precondicionadores desarrollados a partir de las características propias del problema que lo origina, un estudio más completo puede encontrarse en [1] y [9].

¿Qué es un Precondicionador? De una manera formal podemos decir que un precondicionador consiste en construir una matriz $\underline{\underline{C}}$, la cuál es una aproximación en algún sentido de la matriz $\underline{\underline{A}}$ del sistema $\underline{\underline{A}}\underline{u} = \underline{b}$, de manera tal que si multiplicamos ambos miembros del sistema de ecuaciones original por $\underline{\underline{C}}^{-1}$ obtenemos el siguiente sistema

$$\underline{\underline{C}}^{-1}\underline{\underline{A}}\underline{u} = \underline{\underline{C}}^{-1}\underline{b} \tag{4.1}$$

donde el número de condicionamiento de la matriz del sistema transformado $\underline{\underline{C}}^{-1}\underline{\underline{A}}$ debe ser menor que el del sistema original, es decir

$$Cond(\underline{\underline{C}}^{-1}\underline{\underline{A}}) < Cond(\underline{\underline{A}}), \tag{4.2}$$

dicho de otra forma un precondicionador es una inversa aproximada de la matriz original

$$\underline{\underline{C}}^{-1} \simeq \underline{\underline{A}}^{-1} \tag{4.3}$$

que en el caso ideal $\underline{\underline{C}}^{-1} = \underline{\underline{A}}^{-1}$ el sistema convergería en una sola iteración, pero el coste computacional del cálculo de $\underline{\underline{A}}^{-1}$ equivaldría a resolver el sistema por un método directo. Se sugiere que $\underline{\underline{C}}$ sea una matriz lo más próxima a $\underline{\underline{A}}$ sin que su determinación suponga un coste computacional elevado.

Dependiendo de la forma de plantear el producto de $\underline{\underline{C}}^{-1}$ por la matriz del sistema obtendremos distintas formas de precondicionamiento, estas son:

$\underline{\underline{C}}^{-1}\underline{\underline{A}}\underline{u} = \underline{\underline{C}}^{-1}\underline{b}$	Precondicionamiento por la izquierda
$\underline{\underline{A}}\underline{\underline{C}}^{-1}\underline{\underline{C}}\underline{u} = \underline{b}$	Precondicionamiento por la derecha
$\underline{\underline{C}}_1^{-1}\underline{\underline{A}}\underline{\underline{C}}_2^{-1}\underline{\underline{C}}_2\underline{u} = \underline{\underline{C}}_1^{-1}\underline{b}$	Precondicionamiento por ambos lados si $\underline{\underline{C}}$ puede factorizarse como $\underline{\underline{C}} = \underline{\underline{C}}_1\underline{\underline{C}}_2$.

El uso de un precondicionador en un método iterativo provoca que se incurra en un costo de cómputo extra debido a que inicialmente se construye y luego se debe aplicar en cada iteración. Teniéndose que encontrar un balance entre el costo de construcción y aplicación del precondicionador versus la ganancia en velocidad en convergencia del método.

Ciertos precondicionadores necesitan poca o ninguna fase de construcción, mientras que otros pueden requerir de un trabajo substancial en esta etapa. Por otra parte la mayoría de los precondicionadores requieren en su aplicación un monto de trabajo proporcional al número de variables; esto implica que se multiplica el trabajo por iteración en un factor constante.

De manera resumida un buen precondicionador debe reunir las siguientes características:

i) Al aplicar un precondicionador $\underline{\underline{C}}$ al sistema original de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{b}$, se debe reducir el número de iteraciones necesarias para que la solución aproximada tenga la convergencia a la solución exacta con una exactitud ε prefijada.

ii) La matriz $\underline{\underline{C}}$ debe ser fácil de calcular, es decir, el costo computacional de la construcción del precondicionador debe ser pequeño comparado con el costo total de resolver el sistema de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{b}$.

iii) El sistema $\underline{\underline{C}}\underline{z} = \underline{r}$ debe ser fácil de resolver. Esto debe interpretarse de dos maneras:

a) El monto de operaciones por iteración debido a la aplicación del precondicionador $\underline{\underline{C}}$ debe ser pequeño o del mismo orden que las que se requerirían sin precondicionamiento. Esto es importante si se trabaja en máquinas secuenciales.

b) El tiempo requerido por iteración debido a la aplicación del precondicionador debe ser pequeño.

En computadoras paralelas es importante que la aplicación del precondicionador sea paralelizable, lo cual eleva su eficiencia, pero debe de existir un

balance entre la eficacia de un precondicionador en el sentido clásico y su eficiencia en paralelo ya que la mayoría de los precondicionadores tradicionales tienen un componente secuencial grande.

Clasificación de los Precondicionadores En general se pueden clasificar en dos grandes grupos según su manera de construcción: los algebraicos o a posteriori y los a priori o directamente relacionados con el problema continuo que lo origina.

4.1 Precondicionador a Posteriori

Los precondicionadores algebraicos o a posteriori son los más generales, ya que sólo dependen de la estructura algebraica de la matriz $\underline{\underline{A}}$, esto quiere decir que no tienen en cuenta los detalles del proceso usado para construir el sistema de ecuaciones lineales $\underline{\underline{A}}\underline{u} = \underline{b}$. Entre estos podemos citar los métodos de precondicionamiento del tipo Jacobi, SSOR, factorización incompleta, inversa aproximada, diagonal óptimo y polinomial.

Precondicionador Jacobi El método precondicionador Jacobi es el precondicionador más simple que existe y consiste en tomar en calidad de precondicionador a los elementos de la diagonal de $\underline{\underline{A}}$

$$C_{ij} = \begin{cases} A_{ij} \quad si \quad i = j \\ \\ 0 \quad si \quad i \neq j. \end{cases} \tag{4.4}$$

Debido a que las operaciones de división son usualmente más costosas en tiempo de cómputo, en la práctica se almacenan los recíprocos de la diagonal de $\underline{\underline{A}}$.

> Ventajas: No necesita trabajo para su construcción y puede mejorar la convergencia.
>
> Desventajas: En problemas con número de condicionamiento muy grande, no es notoria la mejoría en el número de iteraciones.

Precondicionador SSOR Si la matriz original es simétrica, se puede descomponer como en el método de sobrerrelajamiento sucesivo simétrico (SSOR) de la siguiente manera

$$\underline{\underline{A}} = \underline{\underline{D}} + \underline{\underline{L}} + \underline{\underline{L}}^T \tag{4.5}$$

donde $\underline{\underline{D}}$ es la matriz de la diagonal principal y $\underline{\underline{L}}$ es la matriz triangular inferior.

La matriz en el método SSOR se define como

$$\underline{\underline{C}}(\omega) = \frac{1}{2-w}\left(\frac{1}{\omega}\underline{\underline{D}} + \underline{\underline{L}}\right)\left(\frac{1}{\omega}\underline{\underline{D}}\right)^{-1}\left(\frac{1}{\omega}\underline{\underline{D}} + \underline{\underline{L}}\right)^T \tag{4.6}$$

en la práctica la información espectral necesaria para hallar el valor óptimo de ω es demasiado costoso para ser calculado.

> Ventajas: No necesita trabajo para su construcción, puede mejorar la convergencia significativamente.
>
> Desventajas: Su paralelización depende fuertemente del ordenamiento de las variables.

Precondicionador de Factorización Incompleta Existen una amplia clase de precondicionadores basados en factorizaciones incompletas. La idea consiste en que durante el proceso de factorización se ignoran ciertos elementos diferentes de cero correspondientes a posiciones de la matriz original que son nulos. La matriz precondicionadora se expresa como $\underline{\underline{C}} = \underline{\underline{LU}}$, donde $\underline{\underline{L}}$ es la matriz triangular inferior y $\underline{\underline{U}}$ la superior. La eficacia del método depende de cuán buena sea la aproximación de $\underline{\underline{C}}^{-1}$ con respecto a $\underline{\underline{A}}^{-1}$.

El tipo más común de factorización incompleta se basa en seleccionar un subconjunto S de las posiciones de los elementos de la matriz y durante el proceso de factorización considerar a cualquier posición fuera de éste igual a cero. Usualmente se toma como S al conjunto de todas las posiciones (i, j) para las que $A_{ij} \neq 0$. Este tipo de factorización es conocido como factorización incompleta LU de nivel cero, ILU(0).

El proceso de factorización incompleta puede ser descrito formalmente como sigue:

Para cada k, si $i, j > k$:

$$S_{ij} = \begin{cases} A_{ij} - A_{ij}A_{ij}^{-1}A_{kj} & \text{Si} \quad (i,j) \in S \\ \\ A_{ij} & \text{Si} \quad (i,j) \notin S. \end{cases} \tag{4.7}$$

Una variante de la idea básica de las factorizaciones incompletas lo constituye la factorización incompleta modificada que consiste en que si el producto

$$A_{ij} - A_{ij}A_{ij}^{-1}A_{kj} \neq 0 \tag{4.8}$$

y el llenado no esta permitido en la posición (i, j), en lugar de simplemente descartarlo, esta cantidad se le substrae al elemento de la diagonal A_{ij}. Matemáticamente esto corresponde a forzar a la matriz precondicionadora a tener la misma suma por filas que la matriz original. Esta variante resulta de interés puesto que se ha probado que para ciertos casos la aplicación de la factorización incompleta modificada combinada con pequeñas perturbaciones hace que el número de condicionamiento espectral del sistema precondicionado sea de un orden inferior.

> Ventaja: Puede mejorar el condicionamiento y la convergencia significativamente.
>
> Desventaja: El proceso de factorización es costoso y difícil de paralelizar en general.

Precondicionador de Inversa Aproximada El uso del precondicionador de inversas aproximada se ha convertido en una buena alternativa para los precondicionadores implícitos debido a su naturaleza paralelizable. Aquí se construye una matriz inversa aproximada usando el producto escalar de Frobenius.

Sea $\mathcal{S} \subset C_n$, el subespacio de las matrices $\underline{\underline{C}}$ donde se busca una inversa aproximada explícita con un patrón de dispersión desconocido. La formulación del problema está dada como: Encontrar $\underline{\underline{C}}_0 \in \mathcal{S}$ tal que

$$\underline{\underline{C}}_0 = \arg mín_{\underline{\underline{C}} \in \mathcal{S}} \left\| \underline{\underline{AC}} - \underline{\underline{I}} \right\|. \tag{4.9}$$

Además, esta matriz inicial $\underline{\underline{C}}_0$ puede ser una inversa aproximada de $\underline{\underline{A}}$ en un sentido estricto, es decir,

$$\left\| \underline{\underline{AC}}_0 - \underline{\underline{I}} \right\| = \varepsilon < 1. \tag{4.10}$$

Existen dos razones para esto, primero, la ecuación (4.10) permite asegurar que $\underline{\underline{C}}_0$ no es singular (lema de Banach), y segundo, esta será la base para construir un algoritmo explícito para mejorar $\underline{\underline{C}}_0$ y resolver la ecuación $\underline{\underline{A}}\underline{u} = \underline{b}$.

La construcción de $\underline{\underline{C}}_0$ se realiza en paralelo, independizando el cálculo de cada columna. El algoritmo permite comenzar desde cualquier entrada de la columna k, se acepta comúnmente el uso de la diagonal como primera aproximación. Sea r_k el residuo correspondiente a la columna k-ésima, es decir

$$r_k = \underline{\underline{A}}\underline{C}_k - \underline{e}_k \tag{4.11}$$

y sea $\mathcal{I}_k$ el conjunto de índices de las entradas no nulas en r_k, es decir, $\mathcal{I}_k = \{i = \{1, 2, ..., n\} \mid r_{ik} \neq 0\}$. Si $\mathcal{L}_k = \{l = \{1, 2, ..., n\} \mid C_{lk} \neq 0\}$, entonces la nueva entrada se busca en el conjunto $\mathcal{J}_k = \{j \in \mathcal{L}_k^c \mid A_{ij} \neq 0, \forall i \in \mathcal{I}_k\}$. En realidad las únicas entradas consideradas en $\underline{C}_k$ son aquellas que afectan las entradas no nulas de r_k. En lo que sigue, asumimos que $\mathcal{L}_k \cup \{j\} = \left\{i_1^k, i_2^k, ..., i_{p_k}^k\right\}$ es no vacío, siendo p_k el número actual de entradas no nulas de $\underline{C}_k$ y que $i_{p_k}^k = j$, para todo $j \in \mathcal{J}_k$. Para cada j, calculamos

$$\left\|\underline{\underline{A}}\underline{C}_k - \underline{e}_k\right\|_2^2 = 1 - \sum_{l=1}^{p_k} \frac{\left[\det\left(\underline{\underline{D}}_l^k\right)\right]^2}{\det\left(\underline{\underline{G}}_{l-2}^k\right)\det\left(\underline{\underline{G}}_l^k\right)} \tag{4.12}$$

donde, para todo k, $\det\left(\underline{\underline{G}}_0^k\right) = 1$ y $\underline{\underline{G}}_l^k$ es la matriz de Gram de las columnas $i_1^k, i_2^k, ..., i_{p_k}^k$ de la matriz $\underline{\underline{A}}$ con respecto al producto escalar Euclideano; $\underline{\underline{D}}_l^k$ es la matriz que resulta de reemplazar la última fila de la matriz $\underline{\underline{G}}_l^k$ por $a_{ki_1^k}, a_{ki_2^k}, , ..., a_{ki_l^k}$, con $1 \leq l \leq p_k$. Se selecciona el índice j_k que minimiza el valor de $\left\|\underline{\underline{A}}\underline{C}_k - \underline{e}_k\right\|_2$.

Esta estrategia define el nuevo índice seleccionado j_k atendiendo solamente al conjunto $\mathcal{L}_k$, lo que nos lleva a un nuevo óptimo donde se actualizan todas las entradas correspondientes a los índices de $\mathcal{L}_k$. Esto mejora el criterio de (4.9) donde el nuevo índice se selecciona manteniendo las entradas correspondientes a los índices de $\mathcal{L}_k$. Así $\underline{\underline{C}}_k$ se busca en el conjunto

$$\mathcal{S}_k = \{\underline{C}_k \in \mathbb{R}^n \mid C_{ik} = 0, \forall i \in \mathcal{L}_k \cup \{j_k\}\},$$

$$\underline{m}_k = \sum_{l=1}^{p_k} \frac{\det\left(\underline{\underline{D}}_l^k\right)}{\det\left(\underline{\underline{G}}_{l-2}^k\right)\det\left(\underline{\underline{G}}_l^k\right)} \underline{\tilde{m}}_l \tag{4.13}$$

donde $\underline{\tilde{C}}_l$ es el vector con entradas no nulas i_h^k $(1 \leq h \leq l)$. Cada una de las entradas se obtiene evaluado el determinante correspondiente que resulta de reemplazar la última fila del $\det\left(\underline{\underline{G}}_l^k\right)$ por e_h^t, con $1 \leq l \leq p_k$.

Evidentemente, los cálculos de $\left\|\underline{\underline{A}}\underline{C}_k - \underline{e}_k\right\|_2^2$ y de $\underline{C}_k$ pueden actualizarse añadiendo la contribución de la última entrada $j \in \mathcal{J}_k$ a la suma previa de 1 a $p_k - 1$. En la práctica, $\det\left(\underline{\underline{G}}_l^k\right)$ se calcula usando la descomposición de Cholesky puesto que $\underline{\underline{G}}_l^k$ es una matriz simétrica y definida positiva. Esto sólo involucra la factorización de la última fila y columna si aprovechamos la descomposición de $\underline{\underline{G}}_{l-1}^k$. Por otra parte, $\det\left(\underline{\underline{D}}_l^k\right) / \det\left(\underline{\underline{G}}_l^k\right)$ es el valor de la última incógnita del sistema $\underline{\underline{G}}_l^k \underline{d}_l = \left(a_{ki_1^k}, a_{ki_2^k}, ..., a_{ki_l^k}\right)^T$ necesitándose solamente una sustitución por descenso. Finalmente, para obtener $\underline{\tilde{C}}_l$ debe resolverse el sistema $\underline{\underline{G}}_l^k \underline{v}_l = \underline{e}_l$, con $\underline{\tilde{C}}_{i_1^k l} = v_{hl}, (1 \leq h \leq l)$.

Ventaja: Puede mejorar el condicionamiento y la convergencia significativamente y es fácilmente paralelizable.

Desventaja: El proceso de construcción es algo laborioso.

4.2 Precondicionador a Priori

Los precondicionadores a priori son más particulares y dependen para su construcción del conocimiento del proceso de discretización de la ecuación diferencial parcial, dicho de otro modo dependen más del proceso de construcción de la matriz $\underline{\underline{A}}$ que de la estructura de la misma.

Estos precondicionadores usualmente requieren de más trabajo que los del tipo algebraico discutidos anteriormente, sin embargo permiten el desarrollo de métodos de solución especializados más rápidos que los primeros.

Veremos algunos de los métodos más usados relacionados con la solución de ecuaciones diferenciales parciales en general y luego nos concentraremos en el caso de los métodos relacionados directamente con descomposición de dominio.

En estos casos el precondicionador $\underline{\underline{C}}$ no necesariamente toma la forma simple de una matriz, sino que debe ser visto como un operador en general. De aquí que $\underline{\underline{C}}$ podría representar al operador correspondiente a una versión simplificada del problema con valores en la frontera que deseamos resolver.

Por ejemplo se podría emplear en calidad de precondicionador al operador original del problema con coeficientes variables tomado con coeficientes constantes. En el caso del operador de Laplace se podría tomar como precondicionador a su discretización en diferencias finitas centrales.

Por lo general estos métodos alcanzan una mayor eficiencia y una convergencia óptima, es decir, para ese problema en particular el precondicionador encontrado será el mejor precondicionador existente, llegando a disminuir el número de iteraciones hasta en un orden de magnitud. Donde muchos de ellos pueden ser paralelizados de forma efectiva.

El Uso de la Parte Simétrica como Precondicionador La aplicación del método del Gradiente Conjugado en sistemas no auto-adjuntos requiere del almacenamiento de los vectores previamente calculados. Si se usa como precondicionador la parte simétrica

$$(\underline{\underline{A}} + \underline{\underline{A}}^T)/2 \tag{4.14}$$

de la matriz de coeficientes $\underline{\underline{A}}$, entonces no se requiere de éste almacenamiento extra en algunos casos, resolver el sistema de la parte simétrica de la matriz $\underline{\underline{A}}$ puede resultar más complicado que resolver el sistema completo.

El Uso de Métodos Directos Rápidos como Precondicionadores En muchas aplicaciones la matriz de coeficientes $\underline{\underline{A}}$ es simétrica y positiva definida, debido a que proviene de un operador diferencial auto-adjunto y acotado. Esto implica que se cumple la siguiente relación para cualquier matriz $\underline{\underline{B}}$ obtenida de una ecuación diferencial similar

$$c_1 \leq \frac{\underline{x}^T \underline{\underline{A}} \underline{x}}{\underline{x}^T \underline{\underline{B}} \underline{x}} \leq c_2 \quad \forall \underline{x} \tag{4.15}$$

donde c_1 y c_2 no dependen del tamaño de la matriz. La importancia de esta propiedad es que del uso de $\underline{\underline{B}}$ como precondicionador resulta un método iterativo cuyo número de iteraciones no depende del tamaño de la matriz.

La elección más común para construir el precondicionador $\underline{\underline{B}}$ es a partir de la ecuación diferencial parcial separable. El sistema resultante con la matriz $\underline{\underline{B}}$ puede ser resuelto usando uno de los métodos directos de solución rápida, como pueden ser por ejemplo los basados en la transformada rápida de Fourier.

Como una ilustración simple del presente caso obtenemos que cualquier operador elíptico puede ser precondicionado con el operador de Poisson.

Construcción de Precondicionadores para Problemas Elípticos Empleando DDM Existen una amplia gama de este tipo de precondicionadores, pero son específicos al método de descomposición de dominio usado, para el método de subestructuración, los más importantes se derivan de la matriz de rigidez y por el método de proyecciones, en este trabajo se detallan en la implementación del método DVS.

Definición 6 *Un método iterativo para la solución de un sistema lineal es llamado óptimo, si la razón de convergencia a la solución exacta es independiente del tamaño del sistema lineal.*

Definición 7 *Un método para la solución del sistema lineal generado por métodos de descomposición de dominio es llamado escalable, si la razón de convergencia no se deteriora cuando el número de subdominios crece.*

La gran ventaja de los métodos de descomposición de dominio precondicionados entre los que destacan a FETI-DP y BDDC y por ende DVS es que estos métodos pueden ser óptimos y escalables según el tipo de precondicionador a priori que se use en ellos.

5 Algunos Ejemplos

Existen diferentes paquetes y lenguajes de programación en los cuales se puede implementar eficientemente la solución numérica de sistemas lineales. En lo que resta de esta sección mostraremos diversos ejemplos en los que surgen naturalmente los sistemas lineales. Por ejemplo, en la resolución de ecuaciones diferenciales parciales mediante el método de Diferencias Finitas, aquí describimos la implementación[35] en los paquetes de cómputo OCTAVE (MatLab), SciLab y en los lenguajes de programación C++, Python, Fortran y C.

Sobre los Ejemplos de este Trabajo La documentación y los diferentes ejemplos que se presentan en este trabajo, se encuentran disponibles en la página Web: Herramientas, para que puedan ser copiados desde el navegador y ser usados en la terminal de línea de comandos. En aras de que el interesado pueda correr dichos ejemplos y afianzar sus conocimientos, además de que puedan ser usados en diferentes ámbitos a los presentados aquí.

5.1 Implementación en C++ usando GMM++

GMM++[36] es una librería para C++ que permite definir diversos tipos de matrices y vectores además operaciones básicas de álgebra lineal. La facilidad de uso y la gran cantidad de opciones hacen que GMM++ sea una buena opción para trabajar con operaciones elementales de álgebra lineal.

Se instala en Debian Linux y/o Ubuntu como:

```
# apt install libgmm++-dev
```

Para compilar el ejemplo usar:

```
$ g++ ejemplito.cpp -O3
```

Para ejecutar usar:

[35] En los ejemplos cuando es posible se definen matrices que minimizan la memoria usada en el almacenamiento y maximizan la eficiencia de los métodos numéricos de solución del sistema lineal asociado. En el caso de Octave (MatLab) se define a una matriz mediante A = zeros(N,N), esta puede ser reemplazada por una matriz que no guarda valores innecesarios (ceros), esto se hace mediante la declaración de la matriz como A = sparse(N,N).

[36] GMM++ [http://download.gna.org/getfem/html/homepage/gmm/]

```
$ ./a.out
```

Ejemplo 2 *Sea*

$$-u''(x) + u(x) = 0, \qquad 0 \leq x \leq 1, \qquad u(0) = 0, \qquad u(1) = 1$$

entonces el programa queda implementado como:

```
#include <gmm/gmm.h>
#include <math.h>
const double pi = 3.141592653589793;
// Lado derecho
double LD(double x)
{
    return ( -pi * pi * cos(pi * x));
}
// Solucion analitica
double SA(double x)
{
    return (cos(pi * x));
}
int main(void)
{
    int M=11; // Particion
    int N=M-2; // Nodos interiores
    double a=0; // Inicio dominio
    double c=1; // Fin dominio
    double h=(c-a)/(M-1); // Incremento en la malla
    double Y0=1.0; // Condicion inicial en el inicio del dominio
    double Y1=-1.0; // Condicion inicial en el fin del dominio
    // Matriz densa
    gmm::dense_matrix<double> AA(N, N);
    // Matriz dispersa
    gmm::row_matrix< gmm::rsvector<double> > A(N, N);
    // Vectores
    std::vector<double> x(N), b(N);
    int i;
    double P = -2 / (h * h);
    double Q = 1 / (h * h);
```

```
double R = 1 / (h * h);
A(0, 0) = P; // Primer renglon de la matriz A y vector b
A(0, 1) = Q;
b[0] = LD(a + h) - (Y0 / (h * h));
// Renglones intermedios de la matriz A y vector b
for(i = 1; i < N - 1; i++) {
      A(i, i - 1) = R;
      A(i, i) = P;
      A(i, i + 1) = Q;
      b[i] = LD(a + (i + 1) * h);
}
A(N - 1, N - 2) = R; // Relglon final de la matriz A y vector b
A(N - 1, N - 1) = P;
b[N - 1] = LD(a + (i + 1) * h) - (Y1 / (h * h));
// Copia la matriz dispersa a la densa para usarla en LU
gmm::copy(A,AA);
// Visualiza la matriz y el vector
std::cout << "Matriz A"<< AA << gmm::endl;
std::cout << "Vector b"<< b << gmm::endl;
// LU para matrices densa
gmm::lu_solve(AA, x, b);
std::cout << "LU"<< x << gmm::endl;
gmm::identity_matrix PS; // Optional scalar product for cg
gmm::identity_matrix PR; // Optional preconditioner
gmm::iteration iter(10E-6);// Iteration object with the max residu
size_t restart = 50; // restart parameter for GMRES
// Conjugate gradient
gmm::cg(A, x, b, PS, PR, iter);
std::cout << "CGM"<< x << std::endl;
// BICGSTAB BiConjugate Gradient Stabilized
gmm::bicgstab(A, x, b, PR, iter);
std::cout << "BICGSTAB"<< x << std::endl;
// GMRES generalized minimum residual
gmm::gmres(A, x, b, PR, restart, iter);
std::cout << "GMRES"<< x << std::endl;
// Quasi-Minimal Residual method
gmm::qmr(A, x, b, PR, iter);
std::cout << "Quasi-Minimal"<< x << std::endl;
```

```
        // Visualiza la solucion numerica
        std::cout << "Solucion Numerica"<< std::endl;
        std::cout << a << " " << Y0 << gmm::endl;
        for(i = 0; i < N; i++)
            std::cout << (i + 1)*h << " " << x[i] << gmm::endl;
        std::cout << c << " " << Y1 << gmm::endl;
        // Visualiza la solucion analitica
        std::cout << "Solucion Analitica"<< std::endl;
        std::cout << a << " " << SA(a) << gmm::endl;
        for(i = 0; i < N; i++)
            std::cout << (i + 1)*h << " " << SA((a + (i + 1)*h)) <<
gmm::endl;
        std::cout << c << " " << SA(c) << gmm::endl;
        // Visualiza el error en valor absoluto en cada nodo
        std::cout << "Error en el calculo"<< std::endl;
        std::cout << a << "      " << abs(Y0 - SA(a)) << gmm::endl;
        for(i = 0; i < N; i++)
            std::cout << (i + 1)*h << "      " << abs(x[i] - SA((a + (i +
1)*h))) << gmm::endl;
        std::cout << c << "      " << abs(Y1 - SA(c)) << gmm::endl;
        return 0;
    }
```

5.2 Implementación en Python

Python[37] es un lenguaje de programación interpretado, usa tipado dinámico y es multiplataforma cuya filosofía hace hincapié en una sintaxis que favorezca el código legible y soporta programación multiparadigma -soporta orientación a objetos, programación imperativa y programación funcional-, además es desarrollado bajo licencia de código abierto.

Se instala en Debian Linux y/o Ubuntu como:

```
# apt install python3 python3-scipy python3-matplotlib python3-
sympy
```

Para ejecutar usar:

[37] Python [http://www.python.org]

```
$ python3 ejemplo.py
```

Ejemplo 3 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \qquad 0 \leq x \leq 1, \qquad u(0) = 1, \qquad u(1) = -1$$

entonces el programa usando matriz densa queda implementado como:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
""" Ejemplo del Metodo de Diferencias Finitas para resolver la ecuacion
diferencial parcial:
"""
# 2.X compatible
from __future__ import print_function
import sys
if sys.version[0] == '2': input = raw_input
import math
def LadoDerecho(x):
    "Lado derecho de la ecuacion diferencial parcial"
    return -math.pi * math.pi * math.cos(math.pi * x)
def Jacobi(A, b, N, iter):
    "Resuelve Ax=b usando el metodo Jacobi"
    sum = 0
    x = [0] * N
    xt = [0] * N
    for m in range(iter):
        for i in range(N):
            sum = 0.0
            for j in range(N):
                if i == j:
                    continue
                sum += A[i][j] * x[j]
            xt[i] = (1.0 / A[i][i]) * (b[i] - sum)
        for i in range(N):
            x[i] = xt[i];
    return x
def GaussSeidel(A, b, N, iter):
    "Resuelve Ax=b usando el metodo GaussSeidel"
```

```
        sum = 0
        x = [0] * N
         for m in range(iter):
             for i in range(N):
                 sum = 0.0
                 for j in range(N):
                     if i == j:
                         continue
                 sum += A[i][j] * x[j]
                 x[i] = (1.0 / A[i][i]) * (b[i] - sum)
        return x
    # MDF1DD
    if __name__ == '__main__':
        xi = 0.0 # Inicio del diminio
        xf = 1.0 # Fin del dominio
        vi = 1.0 # Valor en la frontera xi
        vf = -1.0 # Valor en la frontera xf
        n = 11 # Particion
        N = n-2 # Incognitas
        h = (xf - xi) / (n - 1); # Incremento en la malla
        # Declaracion de la matriz A y los vectores b y x
        A = [] # Matriz A
        for i in range(N):
            A.append([0]*N)
        b = [0] * N # Vector b
        x = [0] * N # Vector x
        # Stencil
        R = 1 / (h * h)
        P = -2 / (h * h)
        Q = 1 / (h * h)
        # Primer renglon de la matriz A y vector b
        A[0][0] = P
        A[0][1] = Q
        b[0] = LadoDerecho(xi) - vi * R
        print(b)
        # Renglones intermedios de la matriz A y vector b
        for i in range(1,N-1):
            A[i][i - 1] = R
```

```
        A[i][i] = P
        A[i][i + 1] = Q
        b[i] = LadoDerecho(xi + h * (i + 1))
    # Renglon final de la matriz A y vector b
    A[N - 1][N - 2] = R
    A[N - 1][N - 1] = P
    b[N - 1] = LadoDerecho(xi + h * N - 2) - vf * Q
    # Resuelve por el metodo Jacobi
    for i in range(N):
        for j in range(N):
            print(' ', A[i][j],end=")
        print("")
    print(b)
    x = Jacobi(A, b, N, N*14)
    print(x)
    # Solucion
    print(xi, vi)
    for i in range(N):
        print(xi + h * (i + 1), x[i])
        print(xf, vf)
    # Resuelve por el metodo Gauss-Seidel
     for i in range(N):
        for j in range(N):
            print(' ', A[i][j],end=")
            print("")
    print(b)
    x = GaussSeidel(A, b, N, N*7)
    print(x)
    # Solucion
    print(xi, vi)
    for i in range(N):
        print(xi + h * (i + 1), x[i])
    print(xf, vf)
```

Ejemplo 4 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \qquad 0 \leq x \leq 1, \qquad u(0) = 1, \qquad u(1) = -1$$

entonces el programa usando matriz bandada queda implementado como:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
""" Ejemplo del Metodo de Diferencias Finitas para resolver la ecuacion diferencia parcial:"""
# 2.X compatible
from __future__ import print_function
import sys
if sys.version[0] == '2': input = raw_input
import math
def LadoDerecho(x):
    "Lado derecho de la ecuacion diferencial parcial"
    return -math.pi * math.pi * math.cos(math.pi * x)
def Jacobi(A, b, N, iter):
    "Resuelve Ax=b usando el metodo Jacobi"
    sum = 0
    x = [0] * N
    xt = [0] * N
    for m in range(iter):
        sum = A[0][2] * x[1]
        xt[0] = (1.0 / A[0][1]) * (b[0] - sum)
        for i in range(1,N-1):
            sum = A[i][0] * x[i-1] + A[i][2] * x[i+1]
            xt[i] = (1.0 / A[i][1]) * (b[i] - sum)
        sum = A[N-1][0] * x[N-2]
        xt[N-1] = (1.0 / A[N-1][1]) * (b[N-1] - sum)
        for i in range(N):
            x[i] = xt[i];
    return x
def GaussSeidel(A, b, N, iter):
    "Resuelve Ax=b usando el metodo GaussSeidel"
    sum = 0
    x = [0] * N
    for m in range(iter):
        sum = A[0][2] * x[1]
        x[0] = (1.0 / A[0][1]) * (b[0] - sum)
        for i in range(1,N-1):
            sum = A[i][0] * x[i-1] +A[i][2] * x[i+1]
```

```
                x[i] = (1.0 / A[i][1]) * (bi] - sum)
            sum = A[N-1][0] * x[N-2]
            x[N-1] = (1.0 / A[N-1][1]) * (b[N-1] - sum)
        return x
    # MDF1DDBAND
    if __name__ == '__main__':
        xi = 0.0 # Inicio del diminio
        xf = 1.0 # Fin del dominio
        vi = 1.0 # Valor en la frontera xi
        vf = -1.0 # Valor en la frontera xf
        n = 11 # Particion
        N = n-2 # Incognitas
        h = (xf - xi) / (n - 1); # Incremento en la malla
        # Declaracion de la matriz A y los vectores b y x
        A = [] # Matriz A
        for i in range(N):
            A.append([0]*3)
        b = [0] * N # Vector b
        x = [0] * N # Vector x
        # Stencil
        R = 1 / (h * h)
        P = -2 / (h * h)
        Q = 1 / (h * h)
        # Primer renglon de la matriz A y vector b
        A[0][1] = P
        A[0][2] = Q
        b[0] = LadoDerecho(xi) - vi * R
        print(b)
        # Renglones intermedios de la matriz A y vector b
        for i in range(1,N-1):
            A[i][0] = R
            A[i][1] = P
            A[i][2] = Q
            b[i] = LadoDerecho(xi + h * (i + 1))
        # Renglon final de la matriz A y vector b
        A[N - 1][0] = R
        A[N - 1][1] = P
        b[N - 1] = LadoDerecho(xi + h * N - 2) - vf * Q
```

```
# Resuelve por el metodo Jacobi
for i in range(N):
      for j in range(3):
            print(' ', A[i][j],end=")
      print("")
print(b)
x = Jacobi(A, b, N, N*14)
print(x)
# Solucion
print(xi, vi)
for i in range(N):
      print(xi + h * (i + 1), x[i])
print(xf, vf)
# Resuelve por el metodo Gauss-Seidel
for i in range(N):
      for j in range(3):
            print(' ', A[i][j],end=")
      print("")
print(b)
x = GaussSeidel(A, b, N, N*7)
print(x)
# Solucion
print(xi, vi)
for i in range(N):
      print(xi + h * (i + 1), x[i])
print(xf, vf)
```

Ejemplo 5 *Sea*

$u_t + a(x)u'(x) = 0$, *si* $u(x,0) = f(x)$, *la solución analítica es* $u(x,t) = f(x-at)$

entonces el programa queda implementado como:

```
from math import exp
from scipy import sparse
import numpy as np
import matplotlib.pyplot as plt
# Ecuación
```

```
# u_t + 2 u_x = 0
coef_a = 2
def condicion_inicial (x):
"""
Condición inicial de la ecuación
"""
y = exp (-(x - 0.2)*(x - 0.02))
return y
def sol_analitica (x, t):
"""
Solución analítica de la ecuación
"""
y = exp(-(x-2*t)*(x-2*t))
return y
##########
# Dominio
##########
a = -2.0
b = 8.0
# Partición del dominio
nNodos = 401
h = (b - a)/(nNodos - 1)
# Intervalo de tiempo
dt = 0.012
# creo el vector w donde se guradarÃ¡ la soluciÃ³n para cada tiempo
# B matriz del lado derecho
w = np.zeros((nNodos,1))
B = np.zeros((nNodos, nNodos))
B = np.matrix(B)
espacio = np.zeros((nNodos,1))
for i in xrange(nNodos):
xx_ = a + i*h
espacio[i] = xx_
w[i] = condicion_inicial(xx_)
print "Espacio"
print espacio
print "Condición Inicial"
print w
```

```
mu = coef_ a * dt / h
if mu <= 1:
print "mu ", mu
print "Buena aproximaciÃ³n"
else:
print "mu ", mu
print "Mala aproximaciÃ³n"
if coef_ a >= 0:
B[0,0] = 1 - mu
for i in xrange (1, nNodos):
B[i,i-1] = mu
B[i,i] = 1 - mu
else:
B[0,0] = 1 + mu;
B[0,1] = -mu;
# se completa las matrices desde el renglon 2 hasta el m-2
for i in xrange(1, nNodos-1):
B[i,i] = 1 + mu;
B[i, i+1] = -mu;
B[nNodos-1,nNodos-1] = 1 + mu
# para guardar la soulción analitica
xx = np.zeros((nNodos,1));
iteraciones = 201
# Matriz sparse csr
Bs = sparse.csr_ matrix(B);
# Resolvemos el sistema iterativamente
for j in xrange (1, iteraciones):
t = j*dt;
w = Bs*w;
      # Imprimir cada 20 iteraciones
if j%20 == 0:
print "t", t
print "w", w
#for k_ in xrange (nNodos):
# xx[k_] = sol_ analitica(espacio[k_], t)
# print xx
plt.plot(espacio, w)
#plt.plot(espacio, xx)
```

```
# print "XX"
# print xx
# print "W"
# print w
```

5.3 Implementación en Octave (MatLab)

GNU OCTAVE[38] es un paquete de cómputo open source para el cálculo numérico -muy parecido a MatLab[39] pero sin costo alguno para el usuario- el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería.

Ejemplo 6 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \qquad 0 \leq x \leq 1, \qquad u(0) = 1, \qquad u(1) = -1$$

entonces el programa usando matriz densa queda implementado como:

```
function [A,b,x] = mdf1DDD(n)
  xi = 0.0; % Inicio de dominio
  xf = 1.0; % Fin de dominio
  vi = 1.0; % Valor en la forntera xi
  vf = -1.0; % Valor en la frontera xf
  N=n-2; % Nodos interiores
  h=(xf-xi)/(n-1); % Incremento en la malla
  A=zeros(N,N); % Matriz A
  b=zeros(N,1); % Vector b
  %Stencil
  R=1/(h^2);
  P=-2/(h^2);
  Q=1/(h^2);
  % Primer renglon de la matriz A y vector b
  A(1,1)=P;
  A(1,2)=Q;
  b(1)=LadoDerecho(xi)-vi*R;
  % Renglones intermedios de la matriz A y vector b
  for i=2:N-1
```

[38] GNU Octave [http://www.gnu.org/software/octave/]
[39] MatLab [http://www.mathworks.com/products/matlab/]

```
    A(i,i-1)=R;
    A(i,i)=P;
    A(i,i+1)=Q;
    b(i)=LadoDerecho(xi+h*(i));
end
% Relglon final de la matriz A y vector b
A(N,N-1)=R;
A(N,N)=P;
b(N)=LadoDerecho(xi+h*N)-vf*Q;
% Resuleve el sistema lineal Ax=b
x=Gauss(A,b,N,N*7);
% Prepara la graficacion
xx=zeros(n,1);
zz=zeros(n,1);
for i=1:n
    xx(i)=xi+h*(i-1);
    zz(i)=SolucionAnalitica(xx(i));
end
yy=zeros(n,1);
yy(1)=vi; % Condicion inicial
for i=1:N
    yy(i+1)=x(i);
end
yy(n)=vf; % Condicion inicial
% Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
plot(xx,[yy,zz]);
%plot(xx,zz);
% Resuleve el sistema lineal Ax=b
x=Jacobi(A,b,N,N*14);
% Prepara la graficacion
xx=zeros(n,1);
zz=zeros(n,1);
for i=1:n
    xx(i)=xi+h*(i-1);
    zz(i)=SolucionAnalitica(xx(i));
end
yy=zeros(n,1);
yy(1)=vi; % Condicion inicial
```

```
  for i=1:N
    yy(i+1)=x(i);
  end
   yy(n)=vf; % Condicion inicial
  % Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
  plot(xx,[yy,zz]);
endfunction
% Lado derecho de la ecuacion
function y=LadoDerecho(x)
  y=-pi*pi*cos(pi*x);
endfunction
% Solucion analitica a la ecuacion
function y=SolucionAnalitica(x)
  y=cos(pi*x);
endfunction
% Resuelve Ax=b usando el metodo Jacobi
function x=Jacobi(A,b,N, iter)
  x=zeros(N,1);
  xt=zeros(N,1);
  for it = 1: iter
    for i = 1: N
      sum = 0.0;
      for j = 1: N
        if i == j
          continue;
        end
        sum = sum + A(i,j) * x(j);
      end
      xt(i) = (b(i) - sum) / A(i,i);
    end
     for i = 1: N
      x(i)=xt(i);
    end
  end
endfunction
% Resuelve Ax=b usando el metodo Gauss-Seidel
function x=Gauss(A,b,N, iter)
  x=zeros(N,1);
```

```
    for it = 1: iter
      for i = 1: N
        sum = 0.0;
        for j = 1: N
          if i == j
            continue;
          end
          sum = sum + A(i,j) * x(j);
        end
        x(i) = (b(i) - sum) / A(i,i)
      end
    end
  endfunction
```

Si el programa lo grabamos en el directorio de trabajo con el nombre **fdm1DD.m**, entonces se puede ejecutar en la consola de OCTAVE (MatLab) mediante

$$[A, b, x] = fdm1DDD(11);$$

donde es necesario indicar el número de nodos (11) en la partición. La ejecución genera la gráfica de la solución calculada por el método de diferencias finitas y la solución analítica en los mismos puntos de la malla, además devuelve la matriz y los vectores A, b, x generados por la función.

Ejemplo 7 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \qquad 0 \leq x \leq 1, \qquad u(0) = 1, \qquad u(1) = -1$$

entonces el programa usando matriz bandada queda implementado como:

```
  function [A,b,x] = mdf1DDDBand(n)
    xi = 0.0; % Inicio de dominio
    xf = 1.0; % Fin de dominio
    vi = 1.0; % Valor en la forntera xi
    vf = -1.0; % Valor en la frontera xf
    N=n-2; % Nodos interiores
    h=(xf-xi)/(n-1); % Incremento en la malla
    A=zeros(N,3); % Matriz A
    b=zeros(N,1); % Vector b
    % Stencil
```

```
R=1/(h^2);
P=-2/(h^2);
Q=1/(h^2);
% Primer renglon de la matriz A y vector b
A(1,2)=P;
A(1,3)=Q;
b(1)=LadoDerecho(xi)-vi*R;
% Renglones intermedios de la matriz A y vector b
for i=2:N-1
   A(i,1)=R;
   A(i,2)=P;
   A(i,3)=Q;
   b(i)=LadoDerecho(xi+h*(i));
end
% Relglon final de la matriz A y vector b
A(N,1)=R;
A(N,2)=P;
b(N)=LadoDerecho(xi+h*N)-vf*Q;
% Resuleve el sistema lineal Ax=b
x=Gauss(A,b,N,200);
% Prepara la graficacion
xx=zeros(n,1);
zz=zeros(n,1);
for i=1:n
   xx(i)=xi+h*(i-1);
   zz(i)=SolucionAnalitica(xx(i));
end
yy=zeros(n,1);
yy(1)=vi; % Condicion inicial
for i=1:N
   yy(i+1)=x(i);
end
 yy(n)=vf; % Condicion inicial
% Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
plot(xx,[yy,zz]);
%plot(xx,zz);
% Resuleve el sistema lineal Ax=b
x=Jacobi(A,b,N,N*14);
```

```
    % Prepara la graficacion
    xx=zeros(n,1);
    zz=zeros(n,1);
    for i=1:n
      xx(i)=xi+h*(i-1);
      zz(i)=SolucionAnalitica(xx(i));
    end
    yy=zeros(n,1);
    yy(1)=vi; % Condicion inicial
    for i=1:N
      yy(i+1)=x(i);
    end
    yy(n)=vf; % Condicion inicial
    % Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
    plot(xx,[yy,zz]);
    %plot(xx,zz);
  endfunction
  % Lado derecho de la ecuacion
  function y=LadoDerecho(x)
    y=-pi*pi*cos(pi*x);
  endfunction
  % Solucion analitica a la ecuacion
  function y=SolucionAnalitica(x)
    y=cos(pi*x);
  endfunction
  % Resuelve Ax=b usando el metodo Jacobi
  function x=Jacobi(A,b,N,iter)
    x=zeros(N,1);
    xt=zeros(N,1);
    for it = 1: iter
      sum = A(1,3) * x(2);
      xt(1) = (1.0 / A(1,2)) * (b(1) - sum);
      for i = 2: N-1
        sum = A(i,1) * x(i-1) + A(i,3) * x(i+1);
        xt(i) = (1.0 / A(i,2)) * (b(i) - sum);
      end
      sum = A(N,1) * x(N-1);
      xt(N) = (1.0 / A(N,2)) * (b(N) - sum);
```

```
    for i = 1: N
      x(i)=xt(i);
    end
  end
endfunction
% Resuelve Ax=b usando el metodo Gauss-Seidel
function x=Gauss(A,b,N,iter)
  x=zeros(N,1);
  for it = 1: iter
    sum = A(1,3) * x(2);
    x(1) = (1.0 / A(1,2)) * (b(1) - sum);
    for i = 2: N-1
      sum = A(i,1) * x(i-1) + A(i,3) * x(i+1);
      x(i) = (1.0 / A(i,2)) * (b(i) - sum);
    end
    sum = A(N,1) * x(N-1);
    x(N) = (1.0 / A(N,2)) * (b(N) - sum);
  end
endfunction
```

Si el programa lo grabamos en el directorio de trabajo con el nombre **fdm1DDBand.m**, entonces se puede ejecutar en la consola de OCTAVE (MatLab) mediante

$$[A, b, x] = fdm1DDDBand(11);$$

donde es necesario indicar el número de nodos (11) en la partición. La ejecución genera la gráfica de la solución calculada por el método de diferencias finitas y la solución analítica en los mismos puntos de la malla, además devuelve la matriz y los vectores A, b, x generados por la función.

Dado que esta ecuación se puede extender a todo es espacio, entonces se puede reescribir así

Ejemplo 8 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \qquad xi \leq x \leq xf, \qquad u(xi) = vi, \qquad u(xf) = xf$$

entonces el programa queda implementado como:

```
function [A,b,x] = fdm1d(xi,xf,vi,vf,n)
  N=n-2; % Nodos interiores
  h=(xf-xi)/(n-1); % Incremento en la malla
  A=zeros(N,N); % Matriz A
  b=zeros(N,1); % Vector b
  R=1/(h^2);
  P=-2/(h^2);
  Q=1/(h^2);
  % Primer renglon de la matriz A y vector b
  A(1,1)=P;
  A(1,2)=Q;
  b(1)=LadoDerecho(xi)-vi*R;
  % Renglones intermedios de la matriz A y vector b
  for i=2:N-1
    A(i,i-1)=R;
    A(i,i)=P;
    A(i,i+1)=Q;
    b(i)=LadoDerecho(xi+h*(i-1));
  end
  % Renglon final de la matriz A y vector b
  A(N,N-1)=R;
  A(N,N)=P;
  b(N)=LadoDerecho(xi+h*N)-vf*Q;
  % Resuleve el sistema lineal Ax=b
  x=inv(A)*b;
  % Prepara la graficacion
  xx=zeros(n,1);
  zz=zeros(n,1);
  for i=1:n
    xx(i)=xi+h*(i-1);
    zz(i)=SolucionAnalitica(xx(i));
  end
  yy=zeros(n,1);
  yy(1)=vi; % Condicion inicial
  for i=1:N
    yy(i+1)=x(i);
  end
  yy(n)=vf; % Condicion inicial
```

```
    % Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
    plot(xx,[yy,zz]);
endfunction
function y=LadoDerecho(x)
    y=-pi*pi*cos(pi*x);
endfunction
function y=SolucionAnalitica(x)
    y=cos(pi*x);
endfunction
```

Si el programa lo grabamos en el directorio de trabajo con el nombre **fdm1d.m**, entonces se puede ejecutar en la consola de OCTAVE (MatLab) mediante

$$[A, b, x] = fdm1d(-1, 2, -1, 1, 30);$$

donde es necesario indicar el inicio (-1) y fin (2) del dominio, el valor de la condición de frontera de inicio (-1) y fin (1), además de el número de nodos (30) en la partición. La ejecución genera la gráfica de la solución calculada por el método de diferencias finitas y la solución analítica en los mismos puntos de la malla, además devuelve la matriz y los vectores A, b, x generados por la función.

5.4 Implementación en SciLab

Scilab[40] es un paquete de cómputo open source para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería.

Ejemplo 9 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \qquad 0 \leq x \leq 1, \qquad u(0) = 1, \qquad u(1) = -1$$

El programa usando matriz densa queda implementado como:

```
function [A,b,x] = mdf1DDD(n)
    xi = 0 // Inicio de dominio
    xf = 1; // Fin de dominio
    vi = 1; // Valor en la forntera xi
    vf = -1; // Valor en la frontera xf
```

[40] Scilab [http://www.scilab.org]

```
 N=n-2; // Nodos interiores
 h=(xf-xi)/(n-1); // Incremento en la malla
 A=zeros(N,N); // Matriz A
 b=zeros(N,1); // Vector b
// Stencil
R=1/(h^2);
P=-2/(h^2);
Q=1/(h^2);
 // Primer renglon de la matriz A y vector b
A(1,1)=P;
A(1,2)=Q;
b(1)=LadoDerecho(xi)-vi*R;
// Renglones intermedios de la matriz A y vector b
for i=2:N-1
   A(i,i-1)=R;
   A(i,i)=P;
   A(i,i+1)=Q;
   b(i)=LadoDerecho(xi+h*i);
end
// Renglon final de la matriz A y vector b
 A(N,N-1)=R;
 A(N,N)=P;
b(N)=LadoDerecho(xi+h*N)-vf*Q;
// Resuleve el sistema lineal Ax=b
x=Gauss(A,b,N,N*14);
 // Prepara la graficacion
xx=zeros(n,1);
zz=zeros(n,1);
for i=1:n
   xx(i)=xi+h*(i-1);
   zz(i)=SolucionAnalitica(xx(i));
end
yy=zeros(n,1);
yy(1)=vi; // Condicion inicial
for i=1:N
   yy(i+1)=x(i);
end
yy(n)=vf; // Condicion inicial
```

```
    // Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
    plot2d(xx,[yy,zz]);
     // Resuleve el sistema lineal Ax=b
    x=Jacobi(A,b,N,N*7);
    // Prepara la graficacion
    xx=zeros(n,1);
    zz=zeros(n,1);
    for i=1:n
    xx(i)=xi+h*(i-1);
    zz(i)=SolucionAnalitica(xx(i));
    end
     yy=zeros(n,1);
    yy(1)=vi; // Condicion inicial
    for i=1:N
       yy(i+1)=x(i);
    end
    yy(n)=vf; // Condicion inicial
     // Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
    plot2d(xx,[yy,zz]);
endfunction
// Lado derecho de la ecuacion
function y=LadoDerecho(x)
   y=-%pi*%pi*cos(%pi*x);
endfunction
// Solucion analitica a la ecuacion
function y=SolucionAnalitica(x)
   y=cos(%pi*x);
endfunction
// Resuelve Ax=b usando el metodo Jacobi
function x=Jacobi(A,b,N, iter)
   x=zeros(N,1);
   xt=zeros(N,1);
   for it = 1: iter
      for i = 1: N
         sum = 0.0;
         for j = 1: N
            if i == j then
               continue;
```

```
            end
            sum = sum + A(i,j) * x(j);
          end
          xt(i) = (b(i) - sum) / A(i,i);
        end
        sw = 0;
        for i = 1: N
          x(i)=xt(i);
        end
      end
    endfunction
    // Resuelve Ax=b usando el metodo Gauss-Seidel
    function x=Gauss(A,b,N, iter)
      x=zeros(N,1);
      for it = 1: iter
        for i = 1: N
          sum = 0.0;
          for j = 1: N
            if i == j then
              continue;
            end
            sum = sum + A(i,j) * x(j);
          end
          x(i) = (b(i) - sum) / A(i,i);
        end
      end
    endfunction
```

Si el programa lo grabamos en el directorio de trabajo con el nombre **fdm1DDD.sci**, entonces para poder ejecutar la función es necesario cargarla en la consola de SCILAB mediante

exec('./fdm1DDD.sci',-1)

para después ejecutar la función mediante:

$$[A, b, x] = fdm1DDD(11);$$

donde es necesario indicar el número de nodos (11) en la partición. La ejecución genera la gráfica de la solución calculada por el método de diferencias

finitas y la solución analítica en los mismos puntos de la malla, además devuelve la matriz y los vectores A, b, x generados por la función.

Ejemplo 10 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \qquad 0 \leq x \leq 1, \qquad u(0) = 1, \qquad u(1) = -1$$

El programa usando matriz bandada queda implementado como:

```
function [A,b,x] = mdf1DDDBand(n)
  xi = 0 // Inicio de dominio
  xf = 1; // Fin de dominio
  vi = 1; // Valor en la forntera xi
  vf = -1; // Valor en la frontera xf
  N=n-2; // Nodos interiores
  h=(xf-xi)/(n-1); // Incremento en la malla
  A=zeros(N,3); // Matriz A
  b=zeros(N,1); // Vector b
  // Stencil
  R=1/(h^2);
  P=-2/(h^2);
  Q=1/(h^2);
  // Primer renglon de la matriz A y vector b
  A(1,2)=P;
  A(1,3)=Q;
  b(1)=LadoDerecho(xi)-vi*R;
  // Renglones intermedios de la matriz A y vector b
  for i=2:N-1
    A(i,1)=R;
    A(i,2)=P;
    A(i,3)=Q;
    b(i)=LadoDerecho(xi+h*i);
  end
  // Renglon final de la matriz A y vector b
  A(N,1)=R;
  A(N,2)=P;
  b(N)=LadoDerecho(xi+h*N)-vf*Q;
  // Resuleve el sistema lineal Ax=b
  x=Gauss(A,b,N,N*7);
```

```
    // Prepara la graficacion
    xx=zeros(n,1);
    zz=zeros(n,1);
    for i=1:n
      xx(i)=xi+h*(i-1);
      zz(i)=SolucionAnalitica(xx(i));
    end
    yy=zeros(n,1);
    yy(1)=vi; // Condicion inicial
    for i=1:N
      yy(i+1)=x(i);
    end
    yy(n)=vf; // Condicion inicial
    // Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
    plot2d(xx,[yy,zz]);
    // Resuleve el sistema lineal Ax=b
    x=Jacobi(A,b,N,N*14);
    // Prepara la graficacion
    xx=zeros(n,1);
    zz=zeros(n,1);
    for i=1:n
      xx(i)=xi+h*(i-1);
      zz(i)=SolucionAnalitica(xx(i));
    end
    yy=zeros(n,1);
    yy(1)=vi; // Condicion inicial
    for i=1:N
      yy(i+1)=x(i);
    end
    yy(n)=vf; // Condicion inicial
    // Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
    plot2d(xx,[yy,zz]);
  endfunction
  // Lado derecho de la ecuacion
  function y=LadoDerecho(x)
    y=-%pi*%pi*cos(%pi*x);
  endfunction
  // Solucion analitica a la ecuacion
```

```
function y=SolucionAnalitica(x)
  y=cos(%pi*x);
endfunction
// Resuelve Ax=b usando el metodo Jacobi
function x=Jacobi(A,b,N, iter)
  x=zeros(N,1);
  xt=zeros(N,1);
  for it = 1: iter
    sum = A(1,3) * x(2);
    xt(1) = (1.0 / A(1,2)) * (b(1) - sum);
    for i = 2: N-1
      sum = A(i,1) * x(i-1) + A(i,3) * x(i+1);
      xt(i) = (1.0 / A(i,2)) * (b(i) - sum);
    end
    sum = A(N,1) * x(N-1);
    xt(N) = (1.0 / A(N,2)) * (b(N) - sum);
    for i = 1: N
      x(i)=xt(i);
    end
  end
endfunction
// Resuelve Ax=b usando el metodo Gauss-Seidel
function x=Gauss(A,b,N, iter)
  x=zeros(N,1);
  for it = 1: iter
    sum = A(1,3) * x(2);
    x(1) = (1.0 / A(1,2)) * (b(1) - sum);
    for i = 2: N-1
      sum = A(i,1) * x(i-1) + A(i,3) * x(i+1);
      x(i) = (1.0 / A(i,2)) * (b(i) - sum);
    end
    sum = A(N,1) * x(N-1);
    x(N) = (1.0 / A(N,2)) * (b(N) - sum);
  end
endfunction
```

Si el programa lo grabamos en el directorio de trabajo con el nombre **fdm1DDDBand.sci**, entonces para poder ejecutar la función es necesario

cargarla en la consola de SCILAB mediante

exec('./fdm1DDDBand.sci',-1)

para después ejecutar la función mediante:

$$[A, b, x] = fdm1DDDBand(11);$$

donde es necesario indicar el número de nodos (11) en la partición. La ejecución genera la gráfica de la solución calculada por el método de diferencias finitas y la solución analítica en los mismos puntos de la malla, además devuelve la matriz y los vectores A, b, x generados por la función.

Dado que esta ecuación se puede extender a todo es espacio, entonces se puede reescribir así

Ejemplo 11 *Sea*

$$-u''(x) = -\pi^2 \cos(\pi x), \qquad xi \leq x \leq xf, \qquad u(xi) = vi, \qquad u(xf) = vf$$

El programa queda implementado como:

```
function [A,b,x] = fdm1d(xi,xf,vi,vf,n)
  N=n-2; // Nodos interiores
  h=(xf-xi)/(n-1); // Incremento en la malla
  A=zeros(N,N); // Matriz A
  b=zeros(N,1); // Vector b
  R=1/(h^2);
  P=-2/(h^2);
  Q=1/(h^2);
  // Primer renglon de la matriz A y vector b
  A(1,1)=P;
  A(1,2)=Q;
  b(1)=LadoDerecho(xi)-vi*R;
  // Renglones intermedios de la matriz A y vector b
  for i=2:N-1
    A(i,i-1)=R;
    A(i,i)=P;
    A(i,i+1)=Q;
    b(i)=LadoDerecho(xi+h*(i-1));
  end
```

```
    // Renglon final de la matriz A y vector b
    A(N,N-1)=R;
    A(N,N)=P;
    b(N)=LadoDerecho(xi+h*N)-vf*Q;
    // Resuleve el sistema lineal Ax=b
    x=inv(A)*b;
    // Prepara la graficacion
    xx=zeros(n,1);
    zz=zeros(n,1);
    for i=1:n
       xx(i)=xi+h*(i-1);
       zz(i)=SolucionAnalitica(xx(i));
    end
    yy=zeros(n,1);
    yy(1)=vi; // Condicion inicial
    for i=1:N
       yy(i+1)=x(i);
    end
    yy(n)=vf; // Condicion inicial
    // Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
    plot2d(xx,[yy,zz]);
endfunction
function y=LadoDerecho(x)
    y=-%pi*%pi*cos(%pi*x);
endfunction
function y=SolucionAnalitica(x)
    y=cos(%pi*x);
endfunction
```

Si el programa lo grabamos en el directorio de trabajo con el nombre **fdm1d.sci**, entonces para poder ejecutar la función es necesario cargarla en la consola de SCILAB mediante

```
exec('./fdm1d.sci',-1)
```

para después ejecutar la función mediante:

$$[A, b, x] = fdm1d(-1, 2, -1, 1, 30);$$

donde es necesario indicar el inicio (-1) y fin (2) del dominio, el valor de la condición de frontera de inicio (-1) y fin (1), además de el número de nodos (30) en la partición. La ejecución genera la gráfica de la solución calculada por el método de diferencias finitas y la solución analítica en los mismos puntos de la malla, además devuelve la matriz y los vectores A, b, x generados por la función.

Observación 1 *La diferencia entre los códigos de OCTAVE y SCILAb al menos para estos ejemplos estriba en la forma de indicar los comentarios y la manera de llamar para generar la gráfica, además de que en SCILAB es necesario cargar el archivo que contiene la función.*

Ejemplo 12 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \qquad 0 \leq x \leq 0.5, \qquad u(0) = 1, \qquad u'(0.5) = -\pi$$

entonces el programa queda implementado como:

```
function y=LadoDerecho(x)
 y=-%pi*%pi*cos(%pi*x);
endfunction

function y=SolucionAnalitica(x)
 y=cos(%pi*x);
endfunction

a=0; // Inicio dominio
c=0.5; // Fin dominio
M=40; // Partición
N=M-1; // Nodos interiores
h=(c-a)/(M-1); // Incremento en la malla
Y0=1; // Condicion Dirchlet inicial en el inicio del dominio
Y1=-%pi; // Condicion Neumann inicial en el fin del dominio
A=zeros(N,N); // Matriz A
b=zeros(N); // Vector b

R=1/(h^2);
P=-2/(h^2);
```

```
Q=1/(h^2);

// Primer renglon de la matriz A y vector b
A(1,1)=P;
A(1,2)=Q;
b(1)=LadoDerecho(a)-Y0*R; // Frontera Dirichlet
// Renglones intermedios de la matriz A y vector b
for i=2:N-1
 A(i,i-1)=R;
 A(i,i)=P;
A(i,i+1)=Q;
 b(i)=LadoDerecho(a+h*(i-1));
end
// Relglon final de la matriz A y vector b
A(N,N-1)=-1/(h^2);
A(N,N)=-1/(h^2);
b(N)=Y1/h; // Frontera Neumann

// Resuleve el sistema lineal Ax=b
x=inv(A)*b;

// Prepara la graficación
xx=zeros(M,1);
zz=zeros(M,1);
for i=1:M
 xx(i)=a+h*(i-1);
 zz(i)=SolucionAnalitica(xx(i));
end
yy=zeros(M,1);
yy(1)=Y0; // Condicion inicial
for i=1:N
 yy(i+1)=x(i);
end
// Grafica la solución de la Ecuacion Diferencial Parcial en 1D
plot2d(xx,[yy,zz])
```

Ejemplo 13 *Sea*

$$-u''(x) + u(x) = 0, \qquad 0 \leq x \leq 1, \qquad u(0) = 0, \qquad u(1) = 1$$

entonces el programa queda implementado como:

```
a=0; // Inicio dominio
c=1; // Fin dominio
M=50; // Partición
N=M-2; // Nodos interiores
h=(c-a)/(M-1); // Incremento en la malla
Y0=0; // Condicion inicial en el inicio del dominio
Y1=1; // Condicion inicial en el fin del dominio
A=zeros(N,N); // Matriz A
b=zeros(N); // Vector b


P=2/(h^2);
Q=-1/(h^2)+1/(2*h);
R=-1/(h^2)-1/(2*h);


// Primer renglon de la matriz A y vector b
A(1,1)=P;
A(1,2)=Q;
b(1)=-Y0*R;
// Renglones intermedios de la matriz A y vector b
for i=2:N-1
 A(i,i-1)=R;
 A(i,i)=P;
 A(i,i+1)=Q;
end
// Relglon final de la matriz A y vector b
A(N,N-1)=R;
A(N,N)=P;
b(N)=-Y1*Q;

// Resuleve el sistema lineal Ax=b
x=inv(A)*b;
```

```
// Prepara la graficación
xx=zeros(M,1);
for i=1:M
 xx(i)=a+h*(i-1);
end
yy=zeros(M,1);
yy(1)=Y0; // Condicion inicial
for i=1:N
 yy(i+1)=x(i);
end
yy(M)=Y1; // Condicion inicial
// Grafica la solución de la Ecuacion Diferencial Parcial en 1D
plot2d(xx,yy)
```

Ejemplo 14 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \qquad 0 \le x \le 1, \qquad u(0) = -1, \qquad u(1) = 1$$

entonces el programa queda implementado como:

```
function y=LadoDerecho(x)
 y=-%pi*%pi*cos(%pi*x);
endfunction

function y=SolucionAnalitica(x)
 y=cos(%pi*x);
endfunction

a=-1; // Inicio dominio
c=2; // Fin dominio
M=100; // Partición
N=M-2; // Nodos interiores
h=(c-a)/(M-1); // Incremento en la malla
Y0=-1; // Condicion inicial en el inicio del dominio
Y1=1; // Condicion inicial en el fin del dominio

A=zeros(N,N); // Matriz A
b=zeros(N); // Vector b
```

```
R=1/(h^2);
P=-2/(h^2);
Q=1/(h^2);
// Primer renglon de la matriz A y vector b
A(1,1)=P;
A(1,2)=Q;
b(1)=LadoDerecho(a)-Y0*R;
// Renglones intermedios de la matriz A y vector b
for i=2:N-1
 A(i,i-1)=R;
 A(i,i)=P;
 A(i,i+1)=Q;
 b(i)=LadoDerecho(a+h*(i-1));
end
// Relglon final de la matriz A y vector b
A(N,N-1)=R;
A(N,N)=P;
b(N)=LadoDerecho(a+h*N)-Y1*Q;

// Resuleve el sistema lineal Ax=b
x=inv(A)*b;

// Prepara la graficación
xx=zeros(M,1);
zz=zeros(M,1);
for i=1:M
 xx(i)=a+h*(i-1);
 zz(i)=SolucionAnalitica(xx(i));
end
yy=zeros(M,1);
yy(1)=Y0; // Condicion inicial
for i=1:N
 yy(i+1)=x(i);
end
yy(M)=Y1; // Condicion inicial
// Grafica la solución de la Ecuacion Diferencial Parcial en 1D
plot2d(xx,[yy,zz])
```

Ejemplo 15 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \qquad 0 \leq x \leq 0.5, \qquad u(0) = 1, \qquad u'(0.5) = -\pi$$

entonces el programa queda implementado como:

```
function y=LadoDerecho(x)
 y=-%pi*%pi*cos(%pi*x);
endfunction

function y=SolucionAnalitica(x)
 y=cos(%pi*x);
endfunction

a=0; // Inicio dominio
c=0.5; // Fin dominio
M=40; // Partición
N=M-1; // Nodos interiores
h=(c-a)/(M-1); // Incremento en la malla
Y0=1; // Condicion Dirchlet inicial en el inicio del dominio
Y1=-%pi; // Condicion Neumann inicial en el fin del dominio
A=zeros(N,N); // Matriz A
b=zeros(N); // Vector b

R=1/(h^2);
P=-2/(h^2);
Q=1/(h^2);

// Primer renglon de la matriz A y vector b
A(1,1)=P;
A(1,2)=Q;
b(1)=LadoDerecho(a)-Y0*R; // Frontera Dirichlet
// Renglones intermedios de la matriz A y vector b
for i=2:N-1
 A(i,i-1)=R;
 A(i,i)=P;
A(i,i+1)=Q;
 b(i)=LadoDerecho(a+h*(i-1));
end
// Relglon final de la matriz A y vector b
```

```
A(N,N-1)=-1/(h^2);
A(N,N)=-1/(h^2);
b(N)=Y1/h; // Frontera Neumann

// Resuleve el sistema lineal Ax=b
x=inv(A)*b;

// Prepara la graficación
xx=zeros(M,1);
zz=zeros(M,1);
for i=1:M
 xx(i)=a+h*(i-1);
 zz(i)=SolucionAnalitica(xx(i));
end
yy=zeros(M,1);
yy(1)=Y0; // Condicion inicial
for i=1:N
 yy(i+1)=x(i);
end
// Grafica la solución de la Ecuacion Diferencial Parcial en 1D
plot2d(xx,[yy,zz])
```

5.5 Implementación en Fortran

Fortran es un lenguaje de programación procedimental e imperativo que está adaptado al cálculo numérico y la computación científica, existen una gran variedad de subrutinas para manipulación de matrices, pero es fácil implementar lo necesario para nuestros fines.

Ejemplo 16 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \qquad 0 \leq x \leq 1, \qquad u(0) = 1, \qquad u(1) = -1$$

entonces el programa usando matriz densa queda implementado como:

```
program mdf1D
    implicit none
    integer i, N, nn
    real*16, allocatable :: A(:,:), b(:), x(:)
    real*16 xi, xf, vi, vf, h, R, P, Q, y
```

```
xi = 0.0 ! Inicio del diminio
xf = 1.0 ! Fin del dominio
vi = 1.0 ! Valor en la frontera xi
vf = -1.0 ! Valor en la frontera xf
nn = 11 ! Particion
N = nn - 2 ! Nodos interiores
h = (xf - xi) / (nn-1) ! Incremento en la malla
allocate (A(N,N), b(N), x(N))
! Stencil
R = 1. / (h * h)
P = -2. / (h * h)
Q = 1. / (h * h)
! Primer renglon de la matriz A y vector b
A(1,1)=P
A(2,1)=Q
call ladoDerecho(xi,y)
b(1)=y-vi*R
! Renglones intermedios de la matriz A y vector b
do i=2,N-1
      A(i-1,i)=R
      A(i,i)=P
      A(i+1,i)=Q
      call ladoDerecho(xi+h*(i),y)
      b(i)= y
end do
! Renglon final de la matriz A y vector b
A(N-1,N)=R
A(N,N)=P
call ladoDerecho(xi+h*N,y)
b(N)= y -vf*Q
call gaussSeidel(A, x, b, N, 1000)
 print*, "A: ", A
 print*, "b: ", b
print*, "x: ", x
 call jacobi(A, x, b, N, 1000)
 print*, "A: ", A
 print*, "b: ", b
 print*, "x: ", x
```

```
end program
subroutine ladoDerecho(x,y)
      real*16, intent(in) :: x
      real*16, intent(inout) :: y
      real*16 pi
      pi = 3.14159265358979323846264338327;
      y = -pi * pi * cos(pi * x);
end subroutine
subroutine gaussSeidel(a, x, b, nx, iter)
      implicit none
      integer, intent(in) :: nx, iter
      real*16, intent(in) :: a(nx,nx), b(nx)
      real*16, intent(inout) :: x(nx)
      integer i, j, m
      real*16 sum

      do m = 1, iter
            do i = 1, nx
                  sum = 0.0
                  do j = 1, nx
                        if (i .NE. j) then
                              sum = sum + a(i,j) * x(j)
                        end if
                  end do
                  x(i) = (1.0 / a(i,i)) * (b(i) - sum)
            end do
      end do
end subroutine
 subroutine jacobi(a, x, b, nx, iter)
      implicit none
       integer, intent(in) :: nx, iter
       real*16, intent(in) :: a(nx,nx), b(nx)
       real*16, intent(inout) :: x(nx)
       real*16, allocatable :: xt(:)
       integer i, j, m
       real*16 sum
       allocate (xt(nx))
       do m = 1, iter
```

```
            do i = 1, nx
                sum = 0.0
                do j = 1, nx
                    if (i .NE. j) then
                        sum = sum + a(i,j)*x(j)
                    end if
                end do
                xt(i) = (1.0 / a(i,i)) * (b(i) - sum)
            end do
            do i = 1, nx
                x(i) = xt(i)
            end do
        end do
    end subroutine
```

Ejemplo 17 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \qquad 0 \leq x \leq 1, \qquad u(0) = 1, \qquad u(1) = -1$$

entonces el programa usando matriz bandada queda implementado como:

```
program mdf1DDDBand
    implicit none
    integer i, N, nn
    real*16, allocatable :: A(:,:), b(:), x(:)
    real*16 xi, xf, vi, vf, h, R, P, Q, y
    xi = 0.0 ! Inicio del diminio
    xf = 1.0 ! Fin del dominio
    vi = 1.0 ! Valor en la frontera xi
    vf = -1.0 ! Valor en la frontera xf
    nn = 11 ! Particion
    N = nn - 2 ! Nodos interiores
    h = (xf - xi) / (nn-1) ! Incremento en la malla
    allocate (A(3,N), b(N), x(N))
    ! Stencil
    R = 1. / (h * h)
    P = -2. / (h * h)
    Q = 1. / (h * h)
    ! Primer renglon de la matriz A y vector b
```

```
        A(2,1)=P
        A(3,1)=Q
        call ladoDerecho(xi,y)
        b(1)=y-vi*R
        ! Renglones intermedios de la matriz A y vector b
        do i=2,N-1
            A(1,i)=R
            A(2,i)=P
            A(3,i)=Q
            call ladoDerecho(xi+h*(i),y)
            b(i)= y
        end do
        ! Renglon final de la matriz A y vector b
        A(1,N)=R
        A(2,N)=P
        call ladoDerecho(xi+h*N,y)
        b(N)= y -vf*Q
        call gaussSeidel(A, x, b, N, 1000)
        print*, "A: ", A
        print*, "b: ", b
        print*, "x: ", x
        call jacobi(A, x, b, N, 1000)
        print*, "A: ", A
        print*, "b: ", b
        print*, "x: ", x
    end program
    subroutine ladoDerecho(x,y)
        real*16, intent(in) :: x
        real*16, intent(inout) :: y
        real*16 pi
        pi = 3.14159265358979323846264338327;
        y = -pi * pi * cos(pi * x);
    end subroutine
    subroutine gaussSeidel(a, x, b, nx, iter)
        implicit none
        integer, intent(in) :: nx, iter
        real*16, intent(in) :: a(3,nx), b(nx)
        real*16, intent(inout) :: x(nx)
```

```
      integer i, j, m
      real*16 sum
      do m = 1, iter
            sum = a(3,1) * x(2)
            x(1) = (1.0 / a(2,1)) * (b(1) - sum)
            do i = 2, nx-1
                  sum = a(1,i) * x(i-1) + a(3,i) * x(i+1)
                 x(i) = (1.0 / a(2,i)) * (b(i) - sum)
           end do
            sum = a(1,i) * x(i-1)
            x(i) = (1.0 / a(2,i)) * (b(i) - sum)
      end do
end subroutine
subroutine jacobi(a, x, b, nx, iter)
      implicit none
      integer, intent(in) :: nx, iter
      real*16, intent(in) :: a(3,nx), b(nx)
      real*16, intent(inout) :: x(nx)
      real*16, allocatable :: xt(:)
      integer i, j, m
      real*16 sum
      allocate (xt(nx))
      do m = 1, iter
           sum = a(3,1) * x(2)
           xt(1) = (1.0 / a(2,1)) * (b(1) - sum)
            do i = 2, nx-1
                 sum = a(1,i) * x(i-1) + a(3,i) * x(i+1)
                  xt(i) = (1.0 / a(2,i)) * (b(i) - sum)
            end do
            sum = a(1,i) * x(i-1)
            xt(i) = (1.0 / a(2,i)) * (b(i) - sum)
            do i = 1, nx
                  x(i) = xt(i)
            end do
       end do
end subroutine
```

5.6 Implementación en C

C es un lenguaje de programación de tipos de datos estáticos, débilmente tipificado, de medio nivel, existen una gran variedad de subrutinas para manipulación de matrices, pero es fácil implementar lo necesario para nuestros fines.

Ejemplo 18 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \qquad 0 \leq x \leq 1, \qquad u(0) = 1, \qquad u(1) = -1$$

entonces el programa usando matriz densa queda implementado como:

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#define PARTICION 11 // Tamano de la particion
#define VISUALIZA 1 // (0) No visualiza la salida, otro valor la visualiza
// Lado derecho de la ecuacion diferencial parcial
double LadoDerecho(double x)
{
        double pi = 3.14159265358979323846264338327;
        return -pi * pi * cos(pi * x);
}
// Resuelve Ax=b usando el metodo Jacobi
void Jacobi(double **A, double *x, double *b, int n, int iter)
{
        int i, j, m;
        double sum;
        double *xt;
        xt = (double*)malloc(n*sizeof(double));
        for (m = 0; m < iter; m ++)
        {
             for (i = 0; i < n; i++)
             {
                    sum = 0.0;
                    for (j = 0; j < n; j ++)
                    {
                          if ( i == j) continue;
```

```
                    sum += A[i][j] * x[j];
                }
                xt[i] = (1.0 / A[i][i]) * (b[i] - sum);
            }
            for (i = 0; i < n; i++) x[i] = xt[i];
        }
        if (VISUALIZA)
        {
            printf("\nMatriz\n");
            for (i = 0; i < n; i++)
            {
                for (j = 0; j < n; j++)
                {
                    printf("%f ", A[i][j]);
                }
                printf("\n");
            }
            printf("\nb\n");
            for (i = 0; i < n; i++)
            {
                printf("%f ", b[i]);
            }
            printf("\nx\n");
            for (i = 0; i < n; i++)
            {
                printf("%f ", x[i]);
            }
            printf("\n");
        }
        free(xt);
    }
    // Resuelve Ax=b usando el metodo Gauss-Seidel
    void Gauss_Seidel(double **A, double *x, double *b, int n, int iter)
    {
        int i, j, m;
        double sum;
        for (m = 0; m < iter; m ++)
        {
```

```
            for (i = 0; i < n; i++)
            {
                sum = 0.0;
                for (j = 0; j < n; j ++)
                {
                    if ( i == j) continue;
                    sum += A[i][j] * x[j];
                }
                x[i] = (1.0 / A[i][i]) * (b[i] - sum);
            }
        }
        if (VISUALIZA)
        {
            printf("\nMatriz\n");
            for (i = 0; i < n; i++)
            {
                for (j = 0; j < n; j++)
                {
                    printf("%f ", A[i][j]);
                }
                printf("\n");
            }
            printf("\nb\n");
            for (i = 0; i < n; i++)
            {
                printf("%f ", b[i]);
            }
            printf("\nx\n");
            for (i = 0; i < n; i++)
            {
                printf("%f ", x[i]);
            }
            printf("\n");
        }
    }
    int main(int argc, const char* argv[])
    {
        double xi = 0.0; // Inicio del diminio
```

```
double xf = 1.0; // Fin del dominio
double vi = 1.0; // Valor en la frontera xi
double vf = -1.0; // Valor en la frontera xf
int n = PARTICION; // Particion
int N = n-2; // Incognitas
double h = (xf - xi) / (n - 1); // Incremento en la malla
int i;
// Declaracion de la matriz A y los vectores b y x
double **A; // Matriz A
double *b; // Vector b
double *x; // Vector x
// Solicitud de la memoria dinamica
b = (double*)malloc(N*sizeof(double));
x = (double*)malloc(N*sizeof(double));
A = (double**)malloc(N*sizeof(double*));
for (i= 0; i < N; i++) A[i] = (double*)malloc(N*sizeof(double));
// Stencil
double R = 1 / (h * h);
double P = -2 / (h * h);
double Q = 1 / (h * h);
// Primer renglon de la matriz A y vector b
A[0][0] = P;
A[0][1] = Q;
b[0] = LadoDerecho(xi) - vi * R;
// Renglones intermedios de la matriz A y vector b
for (i = 1; i < N - 1; i++)
{
    A[i][i - 1] = R;
    A[i][i] = P;
    A[i][i + 1] = Q;
    b[i] = LadoDerecho(xi + h * (i + 1));
}
// Renglon final de la matriz A y vector b
A[N - 1][N - 2] = R;
A[N - 1][N - 1] = P;
b[N - 1] = LadoDerecho(xi + h * N - 2) - vf * Q;
// Resuleve el sistema lineal Ax=b por Gauss-Seidel
for(i = 0; i < N; i++) x[i] = 0.0;
```

```
        Gauss_Seidel(A, x, b, N, N*7);
        printf("%f %f\n", xi, vi);
        for(i = 1; i < N+1; i++) printf("%f %f\n", xi + h*i, x[i-1]);
        printf("%f %f\n\n\n\n", xf, vf);
        // Resuleve el sistema lineal Ax=b por Jacobi
        for(i = 0; i < N; i++) x[i] = 0.0;
        Jacobi(A, x, b, N, N*14);
        printf("%f %f\n", xi, vi);
        for(i = 1; i < N+1; i++) printf("%f %f\n", xi + h*i, x[i-1]);
        printf("%f %f\n", xf, vf);
        // Liberacion de la memoria dinamica
        free(b);
        free(x);
        for (i= 0; i < N; i++) free(A[i]);
        free(A);
        return 0;
    }
```

Ejemplo 19 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \qquad 0 \leq x \leq 1, \qquad u(0) = 1, \qquad u(1) = -1$$

entonces el programa usando matriz bandada queda implementado como:

```
    #include <math.h>
    #include <stdio.h>
    #include <stdlib.h>
    #define PARTICION 11 // Tamano de la particion
    #define VISUALIZA 1 // (0) No visualiza la salida, otro valor la visualiza
    // Lado derecho de la ecuacion diferencial parcial
    double LadoDerecho(double x)
    {
        double pi = 3.14159265358979323846264338327;
        return -pi * pi * cos(pi * x);
    }
    // Resuelve Ax=b usando el metodo Jacobi
    void Jacobi(double **A, double *x, double *b, int n, int iter)
    {
```

```
int i, j, m;
double sum;
double *xt;
xt = (double*)malloc(n*sizeof(double));
for (i = 0; i < n; i++) xt[i] = 0.0;
for (m = 0; m < iter; m ++)
{
    sum = A[0][2] * x[1];
    xt[0] = (1.0 / A[0][1]) * (b[0] - sum);
    for (i = 1; i < n-1; i++) {
        sum = A[i][0] * x[i-1] + A[i][2] * x[i+1];
        xt[i] = (1.0 / A[i][1]) * (b[i] - sum);
    }
    sum = A[i][0] * x[i-1];
    xt[i] = (1.0 / A[i][1]) * (b[i] - sum);
    for (i = 0; i < n; i++) x[i] = xt[i];
}
if (VISUALIZA)
{
    printf("\nMatriz\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < 3; j++)
        {
            printf("%f ", A[i][j]);
        }
        printf("\n");
    }
    printf("\nb\n");
    for (i = 0; i < n; i++)
    {
        printf("%f ", b[i]);
    }
    printf("\nx\n");
    for (i = 0; i < n; i++)
    {
        printf("%f ", x[i]);
    }
```

```
            printf("\n");
        }
        free(xt);
    }
    // Resuelve Ax=b usando el metodo Gauss-Seidel
    void Gauss_Seidel(double **A, double *x, double *b, int n, int iter)
    {
        int i, j, m;
        double sum;
        for (m = 0; m < iter; m ++)
        {
            sum = A[0][2] * x[1];
            x[0] = (1.0 / A[0][1]) * (b[0] - sum);
            for (i = 1; i < n-1; i++)
            {
                sum = A[i][0] * x[i-1] + A[i][2] * x[i+1];
                x[i] = (1.0 / A[i][1]) * (b[i] - sum);
            }
            sum = A[i][0] * x[i-1];
            x[i] = (1.0 / A[i][1]) * (b[i] - sum);
        }
        if (VISUALIZA)
        {
            printf("\nMatriz\n");
            for (i = 0; i < n; i++)
            {
                for (j = 0; j < 3; j++)
                {
                    printf("%f ", A[i][j]);
                }
                printf("\n");
            }
            printf("\nb\n");
            for (i = 0; i < n; i++)
            {
                printf("%f ", b[i]);
            }
            printf("\nx\n");
```

```
            for (i = 0; i < n; i++)
            {
                printf("%f ", x[i]);
            }
            printf("\n");
        }
    }
    int main(int argc, const char* argv[])
    {
        double xi = 0.0; // Inicio del diminio
        double xf = 1.0; // Fin del dominio
        double vi = 1.0; // Valor en la frontera xi
        double vf = -1.0; // Valor en la frontera xf
        int n = PARTICION; // Particion
        int N = n-2; // Numero de incognitas
        double h = (xf - xi) / (n - 1); // Incremento en la malla
        int i;
        // Declaracion de la matriz A y los vectores b y x
        double **A; // Matriz A
        double *b; // Vector b
        double *x; // Vector x
        // Solicitud de la memoria dinamica
        b = (double*)malloc(N*sizeof(double));
        x = (double*)malloc(N*sizeof(double));
        A = (double**)malloc(N*sizeof(double*));
        for (i= 0; i < N; i++) A[i] = (double*)malloc(3*sizeof(double));
        // Stencil
        double R = 1 / (h * h);
        double P = -2 / (h * h);
        double Q = 1 / (h * h);
        // Primer renglon de la matriz A y vector b
        A[0][1] = P;
        A[0][2] = Q;
        b[0] = LadoDerecho(xi) - vi * R;
        // Renglones intermedios de la matriz A y vector b
        for (i = 1; i < N - 1; i++)
        {
            A[i][0] = R;
```

```
            A[i][1] = P;
            A[i][2] = Q;
            b[i] = LadoDerecho(xi + h * (i + 1));
        }
        // Renglon final de la matriz A y vector b
        A[N - 1][0] = R;
        A[N - 1][1] = P;
        b[N - 1] = LadoDerecho(xi + h * N - 2) - vf * Q;
        // Resuleve el sistema lineal Ax=b por Gauss-Seidel
        for(i = 0; i < N; i++) x[i] = 0.0;
        Gauss_Seidel(A, x, b, N, N*7);
        printf("%f %f\n", xi, vi);
        for(i = 1; i < N+1; i++) printf("%f %f\n", xi + h*i, x[i-1]);
        printf("%f %f\n\n\n", xf, vf);
        // Resuleve el sistema lineal Ax=b por Jacobi
        for(i = 0; i < N; i++) x[i] = 0.0;
        Jacobi(A, x, b, N, N*14);
        printf("%f %f\n", xi, vi);
        for(i = 1; i < N+1; i++) printf("%f %f\n", xi + h*i, x[i-1]);
        printf("%f %f\n", xf, vf);
        // Liberacion de la memoria dinamica
        free(b);
        free(x);
        for (i= 0; i < N; i++) free(A[i]);
        free(A);
        return 0;
}
```

6 Aritmética de Punto Flotante

La aritmética de punto flotante es considerada un tema esotérico para muchas personas. Esto es sorprendente porque el punto flotante es omnipresente en los sistemas informáticos. Casi todos los lenguajes de programación tienen un tipo de datos de punto flotante, i.e. los números no enteros como 1.2 ó $1e + 45$.

Los problemas de precisión del punto flotante son una preocupación crítica en la informática y la computación numérica, donde la representación y manipulación de números reales puede conducir a resultados inesperados y erróneos. Estos obstáculos surgen debido alas limitaciones inherentes de la aritmética de punto flotante, que aproxima números reales con una precisión finita.

Esto puede causar problemas importantes en diversas aplicaciones, desde simulaciones científicas hasta cálculos financieros, donde incluso las imprecisiones menores pueden propagarse y amplificarse, dando lugar a errores sustanciales. Comprender estos obstáculos es esencial para que los desarrolladores, ingenieros y científicos implementen algoritmos numéricos sólidos y confiables, garantizando que las matemáticas realizadas por las computadoras sigan siendo confiables y precisas.

Comprensión de los Errores de Redondeo en Aritmética de Punto Flotante El meollo de la cuestión es la representación de números de punto flotante. Las computadoras utilizan una cantidad finita de bits para almacenar estos números, y generalmente cumplen con el estándar IEEE 754[41]. Este estándar define cómo se almacenan los números en formato binario, con un número fijo de bits asignados para el signo, el exponente y la mantisa. Si bien esto permite representar una amplia gama de valores, también impone limitaciones a la precisión. No todos los números decimales se pueden representar exactamente en forma binaria, lo que genera pequeñas discrepancias conocidas como errores de redondeo.

Una de las primeras cosas que uno encuentra con sorpresa cuando hace cálculos en una computadora es que por ejemplo, si usamos números muy grandes y seguimos incrementando su valor eventualmente el resultado será

[41] El estándar IEEE 754-2008 define varios tamaños de números de punto flotante: media precisión (binary16), precisión simple (binary32), precisión doble (binary64), precisión cuádruple (binary128), etc., cada uno con su propia especificación.

negativo ... ¿qué pasó? Esto se llama desbordamiento aritmético al intentar crear un valor numérico que está fuera del rango que puede representarse con un número dado de dígitos, ya sea mayor que el máximo o menor que el mínimo valor representable. Algo similar pasa al restar al menor número representable en la máquina, el resultado será positivo y se denomina subdesbordamiento.

Por otro lado, tenemos errores de redondeo, estos ocurren porque el sistema binario no puede representar con precisión ciertas fracciones. Por ejemplo, el número decimal 0.1 no se puede representar exactamente en binario, lo que da como resultado una aproximación, por ejemplo al sumar 0.1 + 0.2 en una computadora usando por ejemplo el lenguaje Python obtenemos:

```
print(0.1+0.2)
0.30000000000000004
```

que no es exactamente lo que esperábamos[42]. Cuando se utilizan tales aproximaciones en los cálculos, los errores pueden acumularse. Este fenómeno es particularmente problemático en procesos iterativos, donde el mismo cálculo se realiza repetidamente y los errores se acumulan con el tiempo.

Estos errores de precisión se vuelven particularmente problemáticos cuando se realizan controles de igualdad. En un mundo ideal, comparar la igualdad de dos números de punto flotante sería sencillo. Sin embargo, debido a los pequeños errores introducidos durante las operaciones aritméticas, dos números que deberían ser iguales pueden no ser exactamente iguales en su representación binaria.

Por ejemplo, el resultado de sumar 0.1 y 0.2 puede no ser exactamente igual a 0.3 debido a la forma en que estos números se representan en binario, por ejemplo:

```
if 0.1 + 0.2 == 0.3:
    print("si")
else:
    print("no")
```

[42] Lo mismo obtenemos si usamos la multiplicación, por ejemplo:

```
print(0.1 * 3)
0.30000000000000004
```

la respuesta será "no". Esta discrepancia puede provocar que fallen las comprobaciones de igualdad, lo que provocará un comportamiento inesperado en los programas.

Para mitigar estos problemas, los desarrolladores suelen utilizar una técnica conocida como "comparación épsilon". En lugar de verificar la igualdad exacta, verifican si la diferencia absoluta entre dos números de punto flotante es menor que un valor pequeño predefinido, conocido como épsilon. Este enfoque reconoce la imprecisión inherente de la aritmética de punto flotante y proporciona una forma más sólida de comparar números. Sin embargo, elegir un valor épsilon apropiado puede resultar complicado, ya que depende del contexto específico y del rango de valores involucrados.

El problema se ve exacerbado por la precisión finita de los números de punto flotante. Al realizar operaciones aritméticas, el resultado a menudo debe redondearse para que se ajuste a los bits disponibles. Este redondeo puede introducir más imprecisiones. Por ejemplo, sumar dos números de punto flotante de magnitudes muy diferentes puede provocar una pérdida de precisión, ya que el número más pequeño puede ignorarse de hecho. Esto se conoce como cancelación catastrófica y puede afectar gravemente a la precisión de los cálculos.

Por ejemplo, ¿qué podría tener de interesante la humilde fórmula cuadrática?. Después de todo, es una fórmula. Simplemente le pones números. Bueno, hay un detalle interesante. Cuando el coeficiente lineal b es grande en relación con los otros coeficientes, la fórmula cuadrática puede dar resultados incorrectos cuando se implementa en aritmética de punto flotante. Eso es cierto, pero veamos qué sucede cuando tenemos $a = c = 1$ y $b = 10e8$ en

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ y } x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{6.1}$$

regresa

$$x_1 = -7.450580596923828e - 09 \text{ y } x_2 = -100000000.0$$

La primera raíz está equivocada en aproximadamente un 25%, aunque la segunda es correcta.

¿Qué pasó? La ecuación cuadrática violó la regla cardinal del análisis numérico: evitar restar números casi iguales. Cuanto más similares sean dos números, más precisión puedes perder al restarlos. En este caso $(b^2 - 4ac)$ es casi igual a b. Si evaluamos, obtenemos $1.49e - 8$ cuando sería la respuesta correcta $2.0e - 8$.

Si usamos la otra formula[43]

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \text{ y } x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}} \tag{6.2}$$

obtenemos

$$x_1 = -1e - 08 \text{ y } x_2 = -134217728.0$$

Entonces, ¿cuál formula cuadrática es mejor? Da la respuesta correcta para la primera raíz, exacta dentro de la precisión de la máquina. Pero ahora la segunda raíz está equivocada en un 34%. ¿Por qué la segunda raíz es incorrecta? La misma razón que antes: ¡restamos dos números casi iguales!

La versión familiar de la fórmula cuadrática calcula correctamente la raíz más grande y la otra versión calcula correctamente la raíz más pequeña. Ninguna versión es mejor en general. No estaríamos ni mejor ni peor usando siempre la nueva fórmula cuadrática que la anterior. Cada uno es mejor cuando evita restar números casi iguales. La solución es utilizar ambas fórmulas cuadráticas, utilizando la apropiada para la raíz que estás intentando calcular.

Otro error común es la suposición de que la aritmética de punto flotante es asociativa y distributiva, como lo es en matemáticas puras. En realidad, el orden de las operaciones puede afectar significativamente el resultado debido a errores de redondeo. Por ejemplo, la expresión (a+ b) + c puede producir un resultado diferente que a + (b + c) cuando se trata de números de punto flotante. Esta no asociatividad puede provocar errores sutiles, especialmente en algoritmos complejos que se basan en cálculos precisos.

Además, la aritmética de punto flotante puede introducir problemas en los algoritmos que requieren comparaciones exactas, como los que se utilizan en la clasificación o la búsqueda. Cuando los números de punto flotante se utilizan como claves en estructuras de datos como tablas hash o árboles de búsqueda binarios, los errores de precisión pueden provocar un comportamiento incorrecto, como no encontrar un elemento existente o insertar incorrectamente un duplicado. Para solucionar este problema, es posible que los desarrolladores necesiten implementar funciones de comparación personalizadas que tengan en cuenta la imprecisión del punto flotante.

Pasando a implicaciones prácticas, estos errores de redondeo pueden tener consecuencias de gran alcance. En las simulaciones científicas, pequeñas im-

[43] Para obtener la segunda fórmula, multiplique el numerador y denominador de x_1 por $-b - \sqrt{b^2 - 4ac}$ (y de manera similar para x_2).

precisiones pueden dar lugar a predicciones incorrectas, lo que podría socavar la validez de la investigación. En aplicaciones financieras, los errores de redondeo pueden dar lugar a importantes discrepancias monetarias, afectando todo, desde el análisis del mercado de valores hasta las transacciones bancarias. Incluso en aplicaciones cotidianas como la representación de gráficos, los errores de redondeo pueden causar artefactos visuales que restan valor a la experiencia del usuario.

Además, comprender las limitaciones de la aritmética de punto flotante puede ayudar a diseñar sistemas más robustos. Al anticipar dónde es probable que se produzcan errores de redondeo, los desarrolladores pueden implementar controles y equilibrios para detectar y corregir imprecisiones. Por ejemplo, la aritmética de intervalos puede proporcionar límites a los posibles valores de un cálculo, ofreciendo una manera de cuantificar la incertidumbre introducida por los errores de redondeo.

Estrategias para Mitigar los Problemas de Precisión del Punto Flotante en el Desarrollo de Software Los problemas de precisión del punto flotante son un desafío común en el desarrollo de Software y a menudo conducen a resultados inesperados y errores sutiles. Estos problemas surgen debido a las limitaciones inherentes a la representación de números reales en formato binario, lo que puede provocar errores de redondeo y pérdida de precisión. Para mitigar estos obstáculos, los desarrolladores deben emplear una variedad de estrategias que garanticen la precisión numérica y la confiabilidad en sus aplicaciones.

Una estrategia eficaz es utilizar tipos de datos de mayor precisión cuando sea necesario. Si bien los tipos de punto flotante estándar como "flotante" y "doble" son suficientes para muchas aplicaciones, es posible que no proporcionen la precisión requerida para cálculos más sensibles. En tales casos, el uso de tipos de precisión extendidos, como "long double" en C++ o bibliotecas de precisión arbitraria como GMP (Biblioteca aritmética de precisión múltiple GNU), puede reducir significativamente el riesgo de pérdida de precisión.

Sin embargo, es importante equilibrar la necesidad de precisión con consideraciones de rendimiento, ya que los tipos de mayor precisión pueden resultar costosos desde el punto de vista computacional. Otro enfoque consiste en implementar algoritmos numéricos que sean inherentemente más estables. Algunos algoritmos son más susceptibles a errores de redondeo que otros y elegir el algoritmo correcto puede marcar una diferencia significativa.

Por ejemplo, al resolver sistemas de ecuaciones lineales, utilizar métodos como la descomposición LU o la descomposición QR puede ser más estable que la eliminación gaussiana. Además, se pueden emplear técnicas de refinamiento iterativo para mejorar la precisión de la solución corrigiendo iterativamente los errores introducidos por la aritmética de punto flotante.

Los desarrolladores deben tener en cuenta el orden de las operaciones en sus cálculos. Las propiedades asociativas y distributivas de la aritmética no siempre se cumplen en la aritmética de punto flotante debido a errores de redondeo. Por lo tanto, reorganizar el orden de las operaciones a veces puede conducir a resultados más precisos. Por ejemplo, al sumar una gran cantidad de números de punto flotante, sumarlos en orden de magnitud ascendente puede minimizar la acumulación de errores de redondeo. Esta técnica, conocida como suma de Kahan, ayuda a preservar la precisión al compensar los pequeños errores que ocurren durante el proceso de suma.

Además de estas estrategias, es fundamental realizar pruebas y validaciones exhaustivas del Software numérico. Las pruebas unitarias deben diseñarse para cubrir una amplia gama de valores de entrada, incluidos casos extremos que probablemente expongan problemas de precisión. Comparar los resultados de los cálculos de punto flotante con soluciones analíticas conocidas o utilizar aritmética de mayor precisión como referencia puede ayudar a identificar discrepancias.

Además, se puede realizar un análisis de sensibilidad para evaluar cómo pequeños cambios en los valores de entrada afectan la salida, proporcionando información sobre la estabilidad y confiabilidad de los algoritmos numéricos.

Por último, la documentación y la comunicación desempeñan un papel fundamental a la hora de mitigar los problemas de precisión del punto flotante. Los desarrolladores deben documentar las limitaciones y suposiciones de sus algoritmos numéricos, así como cualquier fuente potencial de error. Esta información es invaluable para otros desarrolladores que necesiten mantener o ampliar el Software. Además, una comunicación clara con las partes interesadas sobre la precisión esperada del Software puede ayudar a gestionar las expectativas y evitar mal entendidos.

6.1 Aritmética de Punto Flotante IEEE

La aritmética de punto flotante es considerada un tema esotérico para muchas personas. Esto es sorprendente porque el punto flotante es omnipresente en los sistemas informáticos. Casi todos los lenguajes de programación tienen

un tipo de datos de punto flotante, i.e. los números no enteros como 1.2 ó $1e + 45$.

Un número de punto flotante de 64 bits (*double* en lenguaje C) tiene aproximadamente 16 dígitos decimales de precisión con un rango del orden de 1.7×10^{-308} a 1.7×10^{308} de acuerdo con el estándar 754 de IEEE -El estándar IEEE 754-2008 define varios tamaños de números de punto flotante: media precisión (binary16), precisión simple (binary32), precisión doble (binary64), precisión cuádruple (binary128), etc., cada uno con su propia especificación-, la implementación típica de punto flotante[44].

Una de las primeras cosas que uno encuentra con sorpresa cuando hace cálculos en una computadora es que por ejemplo, si usamos números muy grandes y seguimos incrementando su valor eventualmente el resultado será negativo ... ¿qué pasó? Esto se llama desbordamiento aritmético al intentar crear un valor numérico que está fuera del rango que puede representarse con un número dado de dígitos, ya sea mayor que el máximo o menor que el mínimo valor representable. Algo similar pasa al restar al menor número representable en la máquina, el resultado será positivo y se denomina subdesbordamiento.

Las computadoras, desde las PC hasta las supercomputadoras, tienen aceleradores de punto flotante; la mayoría de los compiladores deberán compilar algoritmos de punto flotante de vez en cuando y prácticamente todos los sistemas operativos deben responder a excepciones de punto flotante como el desbordamiento.

Desde ya hace mucho tiempo, los procesadores Intel x86 y todos los procesadores de las siguientes generaciones de todas las marcas admiten un formato de precisión extendido de 80 bits con un significado de 64 bits, que es compatible con el especificado en el estándar IEEE. Cuando un compilador usa este formato con registros de 80 bits para acumular sumas y productos internos, está trabajando efectivamente con un redondeo unitario de 2^{-64} en vez de 2^{-53} para precisión doble, dando límites de error más pequeños en un factor de hasta $2^{11} = 2048$.

¿Dieciséis lugares decimales es mucho? Casi ninguna cantidad medida se conoce con tanta precisión. Por ejemplo: en la constante en la ley de gravedad de Newton sólo se conoce con seis cifras significativas. En la carga

[44] Además, existe el número de 80 bits (*long double* en lenguaje C) que tiene 18 dígitos decimales de precisión con un rango del orden de 3.4×10^{-4096} a 1.1×10^{4096}. Y no podemos olvidar, el número de 32 bits (*float* en lenguaje C) que tiene 14 dígitos decimales de precisión con un rango del orden de 3.4×10^{-38} a 3.4×10^{38}.

de un electrón se conoce con 11 cifras significativas, mucha más precisión que la constante gravitacional de Newton, pero aún menos que un número de punto flotante[45].

Entonces, ¿cuándo no son suficientes 16 dígitos de precisión? Un área problemática es la resta. Las otras operaciones elementales (suma, multiplicación, división) son muy precisas. Siempre que no se presenten desbordamientos y subdesbordamientos, estas operaciones suelen producir resultados que son correctos hasta el último bit. Pero la resta puede ser desde exacta hasta completamente inexacta. Si dos números concuerdan con n cifras, puede perder hasta n cifras de precisión en su resta. Este problema puede aparecer inesperadamente en medio de otros cálculos.

¿Qué pasa con el desbordamiento o con el subdesbordamiento? ¿Cuándo se necesitan números mayores que 10^{308}? No tan a menudo, pero en los cálculos de probabilidad, por ejemplo, se usan todo el tiempo a menos que se haga un uso inteligente.

Es común en probabilidad calcular un número de tamaño mediano que es el producto de un número astronómicamente grande y un número infinitesimalmente pequeño. El resultado final encaja perfectamente en una computadora, pero es posible que los números intermedios no se deban a un desbordamiento o un subdesbordamiento. Por ejemplo, el número máximo de punto flotante en la mayoría de las computadoras está entre 170 factorial y 171 factorial. Estos grandes factoriales aparecen frecuentemente en aplicaciones, a menudo en proporciones con otros grandes factoriales.

[45] ¿Cuántos dígitos de π necesitamos?

- 3.1415 para diseñar los mejores motores.
- 3.1415926535 para obtener la circunferencia de la Tierra dentro de una fracción de pulgada.
- 3.141592653589793 para los cálculos de navegación interplanetaria de la NASA-JPL.
- 3.14159265358979323846264338327950288 42 para medir el radio del universo con una precisión igual al tamaño de un átomo de hidrógeno.

En el 2022 se calcularon los primero 100 billones de decimales del número π, para lograr este hito necesitó 157 días, 23 horas, 31 minutos y 7,651 segundos de cálculos, 515 Terabytes de almacenamiento y desplegar un abanico de tecnologías de computación en Compute Engine, un servicio de computación de Google Cloud.

Anatomía de un Número de Punto Flotante Un número de punto flotante de 64 bits codifica un número de la forma $\pm p \times 2^e$. El primer bit codifica el signo, 0 para números positivos y 1 para números negativos. Los siguientes 11 bits codifican el exponente e, y los últimos 52 bits codifican la precisión p.

El exponente se almacena con un sesgo de 1023. Es decir, los exponentes positivos y negativos se almacenan todos en un solo número positivo almacenando $e + 1023$ en lugar de almacenarlos directamente. Once bits pueden representar números enteros desde 0 hasta 2047. Restando el sesgo, esto corresponde a valores de e de -1023 a $+1024$. Definimos $e_{min} = -1022$ y $e_{max} = +1023$. Los valores $e_{min} - 1$ y $e_{max} + 1$ están reservados para usos especiales.

Los números de punto flotante se almacenan normalmente en forma normalizada. En base 10, un número está en notación científica normalizada si el significando es ≥ 1 y < 10. Por ejemplo, 3.14×10^2 está en forma norma-lizada, pero 0.314×10^3 y 31.4×10^2 no lo están.

En general, un número en base β está en forma normalizada si tiene la forma $p \times \beta^e$ donde $1 \leq p < \beta$. Esto dice que para expresar un número binario, es decir, $\beta = 2$, el primer bit del significado de un número normalizado es siempre 1. Dado que este bit nunca cambia, no es necesario almacenarlo. Por lo tanto, podemos expresar 53 bits de precisión en 52 bits de almacenamiento. En lugar de almacenar el significado directamente, almacenamos f, la parte fraccionaria, donde el significado es de la forma $1.f$.

El esquema anterior no explica cómo almacenar 0. Es imposible especificar valores de f y e de modo que $1.f \times 2^e = 0$. El formato de punto flotante hace una excepción a las reglas establecidas anteriormente. Cuando $e = e_{min} - 1$ y $f = 0$, los bits se interpretan como 0. Cuando $e = e_{min} - 1$ y $f \neq 0$, el resultado es un número desnormalizado. Los bits se interpretan como $0.f \times 2^{e_{min}}$. En resumen, el exponente especial reservado debajo de e_{min} se usa para representar 0 y números de punto flotante desnormalizados.

El exponente especial reservado arriba de e_{max} se usa para representar ∞ y NaN. Si $e = e_{max} + 1$ y $f = 0$, los bits se interpretan como ∞. Pero si $e = e_{max} + 1$ y $f \neq 0$, los bits se interpretan como un NaN o "no es un número" (Not a Number).

Dado que el exponente más grande es 1023 y el significativo más grande es $1.f$ donde f tiene 52 unidades, el número de punto flotante (en C y C++, esta

constante se define como DBL_MAX definido en <*float.h*>, en Python[46] en *sys.float_info)* más grande es $2^{1023}(2-2^{-52}) = 2^{1024} - 2^{971} \approx 2^{1024} \approx 1.8 \times 10^{308}$. Los números mayores que 2^{1024} i.e. $(2-2^{52})$ producen un desbordamiento conocido como Overflow.

Dado que el exponente más pequeño es -1022, el número normalizado positivo más pequeño es $1.0 \times 2^{-1022} \approx 2.2 \times 10^{-308}$ (en C y C++, esta constante se define como DBL_MIN definido en <*float.h*>, en Python en *sys.float_info)*. Sin embargo, no es el número positivo más pequeño representable como un número de punto flotante, solo el número de punto flotante normalizado más pequeño. Los números más pequeños se pueden expresar en forma desnormalizada, aunque con una pérdida de significado. El número positivo desnormalizado más pequeño ocurre con f que tiene 51 números 0 seguidos de un solo 1. Esto corresponde a $2^{-52} * 2^{-1022} = 2^{-1074} \approx 4.9 \times 10^{-324}$.

Los números que aparecen en los cálculos y tienen magnitud menor que 2^{-1023} i.e. $(1+2^{-52})$ producen un desbordamiento de la capacidad mínima o subdesbordamiento también conocido como Underflow y, por lo general se igualan a cero.

C y C++ da el nombre de $DBL\_EPSILON$ al número positivo más pequeño ϵ tal que $1+\epsilon \approx 1$, también llamado la precisión de la máquina. Dado que el significativo tiene 52 bits, está claro que $DBL\_EPSILON = 2^{-52} \approx 2.2 \times 10^{-16}$. Por eso decimos que un número de punto flotante tiene entre 15 y 16 cifras significativas (decimales).

Formatos de Punto Flotante Se han propuesto varias representaciones diferentes de números reales, pero por mucho la más utilizada es la representación de punto flotante. Las representaciones de punto flotante tienen una base (que siempre se asume que es par) y una precisión p. Si $\beta = 10$ y $p = 3$, entonces el número 0.1 se representa como 1.00×10^{-1}. Si $\beta = 2$ y

[46]

```
import sys
print(sys.float_info)
```

```
sys.float_info(max=1.7976931348623157e+308, max_exp=1024,
max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021,
min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-
16, radix=2, rounds=1)
```

$p = 24$, entonces el número decimal 0.1 no se puede representar exactamente, pero es aproximadamente $1.10011001100110011001101 \times 2^{-4}$.

En general, un número de punto flotante se representará como $\pm d.dd...d \times \beta^e$, donde $d.dd...d$ se llama mantisa y tiene p dígitos. De forma más precisa $\pm d_0.d_1d_2...d_{p-1} \times \beta^e$ representa el número

$$\pm \left(d_0 + d_1\beta^{-1} + ... + d_{p-1}\beta^{-(p-1)}\right)\beta^e, (0 \leq d_i < \beta). \tag{6.3}$$

El término número de punto flotante se utilizará para referirse a un número real que se puede representar exactamente en el formato en discusión. Otros dos parámetros asociados con las representaciones de punto flotante son los exponentes más grandes y más pequeños permitidos, e_{max} y e_{min}. Dado que hay β^p posibles significados, y $e_{max} - e_{min} + 1$ posibles exponentes, un número de punto flotante se puede codificar en

$$\left[\log_2 \left(e_{max} - e_{min} + 1\right)\right] + \left[\log_2 \left(\beta^p\right)\right] + 1$$

bits, donde el $+1$ final es para el bit de signo. La codificación precisa no es importante por ahora.

Hay dos razones por las que un número real podría no ser exactamente representable como un número de punto flotante. La situación más común se ilustra con el número decimal 0.1. Aunque tiene una representación decimal finita, en binario tiene una representación repetida infinita. Por lo tanto, cuando $\beta = 2$, el número 0.1 se encuentra estrictamente entre dos números de punto flotante y ninguno de ellos lo puede representar exactamente.

Una situación menos común es que un número real esté fuera de rango, es decir, su valor absoluto sea mayor que $\beta \times \beta^{e_{max}}$ o menor que $1.0 \times \beta^{e_{min}}$. La mayor parte de esta sección analiza cuestiones debidas a la primera razón.

Las representaciones de punto flotante no son necesariamente únicas. Por ejemplo, tanto 0.01×10^1 como 1.00×10^{-1} representan 0.1. Si el dígito inicial es distinto de cero ($d_0 \neq 0$ en la Ec. (6.3)), se dice que la representación está normalizada. El número de punto flotante 1.00×10^{-1} está normalizado, mientras que 0.01×10^1 no lo está.

¿Cuándo es Exacta la Conversión de Base de Punto Flotante de Ida y Vuelta? Suponga que almacenamos un número de punto flotante en la memoria, lo imprimimos en base 10 legible por humanos y lo regresamos a memoria. ¿Cuándo se puede recuperar exactamente el número original?

Suponga que comenzamos con la base β con p lugares de precisión y convertimos a la base γ con q lugares de precisión, redondeando al más cercano, luego volvemos a convertir a la base original β. El teorema de Matula dice que si no hay enteros positivos i y j tales que

$$\beta^i = \gamma^j$$

entonces una condición necesaria y suficiente para que la conversión de ida y vuelta sea exacta (suponiendo que no haya desbordamiento o subdesbordamiento) es que

$$\gamma^{q-1} > \beta^p.$$

En el caso de números de punto flotante (por ejemplo doble en C) tenemos $\beta = 2$ y $p = 53$. (ver Anatomía de un Número de Punto Flotante). Estamos imprimiendo a base $\gamma = 10$. Ninguna potencia positiva de 10 también es una potencia de 2, por lo que se mantiene la condición de Matula en las dos bases.

Si imprimimos $q = 17$ decimales, entonces

$$10^{16} > 2^{53}$$

por lo que la conversión de ida y vuelta será exacta si ambas conversiones se redondean al más cercano. Si q es menor, algunas conversiones de ida y vuelta no serán exactas.

También puede verificar que para un número de punto flotante de precisión simple (p = precisión de 24 bits) necesita $q = 9$ dígitos decimales, y para un número de precisión cuádruple (precisión de $p = 113$ bits) necesita $q = 36$ dígitos decimales[47].

Mirando hacia atrás en el teorema de Matula, claramente necesitamos

$$\gamma^q \geq \beta^p.$$

¿Por qué? Porque el lado derecho es el número de fracciones de base β y el lado izquierdo es el número de fracciones de base γ. No puede tener un mapa

[47] El número de bits asignados para la parte fraccionaria de un número de punto flotante es 1 menos que la precisión: la cifra inicial es siempre 1, por lo que los formatos IEEE ahorran un bit al no almacenar el bit inicial, dejándolo implícito. Entonces, por ejemplo, un doble en C tiene una precisión de 53 bits, pero 52 bits de los 64 bits en un doble se asignan para almacenar la fracción.

uno a uno de un espacio más grande a un espacio más pequeño. Entonces, la desigualdad anterior es necesaria, pero no suficiente. Sin embargo, es casi suficiente. Solo necesitamos una base γ con más cifras significativas, es decir, Matula nos dice

$$\gamma^{q-1} > \beta^p$$

es suficiente. En términos de base 2 y base 10, necesitamos al menos 16 decimales para representar 53 bits. Lo sorprendente es que un decimal más es suficiente para garantizar que las conversiones de ida y vuelta sean exactas. No es obvio a priori que cualquier número finito de decimales adicionales sea siempre suficiente, pero de hecho solo uno más es suficiente.

A continuación, se muestra un ejemplo para mostrar que el decimal adicional es necesario. Suponga que $p = 5$. Hay más números de 2 dígitos que números de 5 bits, pero si solo usamos dos dígitos, la conversión de base de ida y vuelta no siempre será exacta. Por ejemplo, el número 17/16 escrito en binario es 1.0001_{dos} y tiene cinco bits significativos. El equivalente decimal es 1.0625_{diez}, que redondeado a dos dígitos significativos es 1.1_{diez}. Pero el número binario más cercano a 1.1_{diez} con 5 bits significativos es $1.0010_{dos} = 1.125_{diez}$. En resumen, redondeando al más cercano da

$$1.0001_{dos} -> 1.1_{diez} -> 1.0010_{dos}$$

y así no volvemos al punto de partida.

Error de Representación se refiere al hecho de que la mayoría de las fracciones decimales no pueden representarse exactamente como fracciones binarias (en base 2). Esta es la razón principal de por qué muchos lenguajes de programación (Python, Perl, C, C++, Java, Fortran, y tantos otros) frecuentemente no mostrarán el número decimal exacto que esperas.

¿Por qué es eso? 1/10 no es representable exactamente como una fracción binaria. Casi todas las máquinas de hoy en día usan aritmética de punto flotante: IEEE-754, y casi todas las plataformas mapean los flotantes al «doble precisión» de IEEE-754. Estos «dobles» tienen 53 bits de precisión, por lo tanto en la entrada la computadora intenta convertir 0.1 a la fracción más cercana que puede de la forma $J/(2^N)$ donde J es un entero que contiene exactamente 53 bits. Reescribiendo

$$1/10 \approx J/2^N$$

como

$$J \approx 2^N/10$$

y recordando que J tiene exactamente 53 bits (es $\geq 2^{52}$ pero $< 2^{53}$), el mejor valor para N es 56. O sea, 56 es el único valor para N que deja J con exactamente 53 bits. El mejor valor posible para J es entonces el cociente redondeado, si usamos Python para los cálculos tenemos

```
>>> q, r = divmod(2**56, 10)
>>> r
6
```

Ya que el resto es más que la mitad de 10, la mejor aproximación se obtiene redondeándolo

```
>>> q+1
7205759403792794
```

Por lo tanto la mejor aproximación a 1/10 en doble precisión 754 es

$$7205759403792794/2**56$$

el dividir tanto el numerador como el denominador reduce la fracción a

$$3602879701896397/2**55$$

Notemos que como lo redondeamos, esto es un poquito más grande que 1/10; si no lo hubiéramos redondeado, el cociente hubiese sido un poquito menor que 1/10. ¡Pero no hay caso en que sea exactamente 1/10!

Entonces la computadora nunca «ve» 1/10: lo que ve es la fracción exacta de arriba, la mejor aproximación al flotante doble de 754 que puede obtener

```
>>> 0.1 * 2 ** 55
3602879701896397.0
```

Si multiplicamos esa fracción por 10^{55}, podemos ver el valor hasta los 55 dígitos decimales

```
>>> 3602879701896397 * 10 ** 55 // 2 ** 55
1000000000000000055511151231257827021181583404541015625
```

lo que significa que el valor exacto almacenado en la computadora es igual al valor decimal

$$0.1000000000000000055511151231257827021181583404541015625.$$

en lugar de mostrar el valor decimal completo, muchos lenguajes, redondean el resultado a 17 dígitos significativos.

Error de Redondeo Comprimir infinitos números reales en un número finito de bits requiere una representación aproximada. Aunque hay un número infinito de enteros, en la mayoría de los programas el resultado de los cálculos de números enteros se puede almacenar en 32 bits o 64 bits. Por el contrario, dado cualquier número fijo de bits, la mayoría de los cálculos con números reales producirán cantidades que no se pueden representar exactamente usando tantos bits. Por lo tanto, el resultado de un cálculo de punto flotante a menudo debe redondearse para volver a ajustarse a su representación finita. Este error de redondeo es el rasgo característico del cálculo de punto flotante.

Dado que la mayoría de los cálculos de punto flotante tienen errores de redondeo de todos modos, ¿importa si las operaciones aritméticas básicas introducen un poco más de error de redondeo de lo necesario? Esa pregunta es un tema principal a lo largo de esta sección. La sección Dígitos de Guarda analiza los dígitos de protección, un medio para reducir el error al restar dos números cercanos. IBM consideró que los dígitos de guarda eran lo suficientemente importantes que en 1968 añadió un dígito de guarda al formato de doble precisión en la arquitectura System / 360 (la precisión simple ya tenía un dígito de guarda) y modernizó todas las máquinas existentes en el campo.

El estándar IEEE va más allá de sólo requerir el uso de un dígito de protección. Proporciona un algoritmo para la suma, resta, multiplicación, división y raíz cuadrada y requiere que las implementaciones produzcan el mismo resultado que ese algoritmo. Por lo tanto, cuando un programa se mueve de una máquina a otra, los resultados de las operaciones básicas serán los mismos en todos los bits si ambas máquinas admiten el estándar IEEE. Esto simplifica enormemente la portabilidad de programas. Otros usos de esta especificación precisa se dan en operaciones exactamente redondeadas.

Error Relativo y Ulps Dado que el error de redondeo es inherente al cálculo de punto flotante, es importante tener una forma de medir este error. Considere el formato de punto flotante con $\beta = 10$ y $p = 3$, que se utilizará

en esta sección. Si el resultado de un cálculo de punto flotante es 3.12×10^{-2}, y la respuesta cuando se calcula con precisión infinita es 0.0314, está claro que tiene un error de 2 unidades en el último lugar. De manera similar, si el número real 0.0314159 se representa como 3.14×10^{-2}, entonces tiene un error de 0.159 unidades en el último lugar.

En general, si el número de punto flotante $dd..d \times \beta^e$ se usa para representar z, entonces tiene un error de $|d.d...d - (z/\beta^e)|^{\beta^{p-1}}$ unidades en el último lugar. El término ulps se utilizará como abreviatura de (units in the last place) "unidades en último lugar". Si el resultado de un cálculo es el número de punto flotante más cercano al resultado correcto, aún podría tener un error de hasta 0.5 ulp.

Otra forma de medir la diferencia entre un número de punto flotante y el número real al que se aproxima es el error relativo, que es simplemente la diferencia entre los dos números divididos por el número real. Por ejemplo, el error relativo cometido al aproximar 3.14159 por 3.14×10^0 es $0.00159/3.141590 \approx 0005$.

Para calcular el error relativo que corresponde a .5 ulp, observe que cuando un número real es aproximado por el número de punto flotante más cercano posible $\overbrace{d.dd...dd}^{p} \times \beta^e$, el error puede ser tan grande como $\overbrace{0.00...00}^{p}\beta' \times \beta^e$, donde β' es el dígito $\beta/2$, hay p unidades en el significado del número de punto flotante y p unidades de 0 en el significado del error. Este error es $((\beta/2)\beta^{-p}) \times \beta^e$. Dado que los números de la forma $d.dd...dd \times \beta^e$ tienen todos el mismo error absoluto, pero tienen valores que oscilan entre β^e y $\beta \times \beta^e$, el error relativo varía entre $((\beta/2)\beta^{-p}) \times \beta^e/\beta^e$ y $((\beta/2)\beta^{-p}) \times \beta^e/\beta^{e+1}$. Eso es,

$$\frac{1}{2}\beta^{-p} \leq \frac{1}{2}ulp \leq \frac{\beta}{2}\beta^{-p} \tag{6.4}$$

En particular, el error relativo correspondiente a 0.5 ulp puede variar en un factor de β. Este factor se llama bamboleo. Estableciendo $\epsilon = (\beta/2)\beta^{-p}$ en el mayor de los límites en Ec. (6.4) anterior, podemos decir que cuando un número real se redondea al número de punto flotante más cercano, el error relativo siempre está limitado por ϵ, que se conoce como épsilon de la máquina.

En el ejemplo anterior, el error relativo fue $0.00159/3.14159 \approx 0005$. Para evitar números tan pequeños, el error relativo normalmente se escribe como

factor multiplicado ϵ, que en este caso es $\epsilon = (\beta/2)\beta^{-p} = 5(10)^{-3} = 0.005$. Por lo tanto, el error relativo se expresaría como $(0.00159/3.14159)/0.005)\epsilon \approx 0.1\epsilon$.

Para ilustrar la diferencia entre ulps y error relativo, considere el número real $x = 12.35$. Se aproxima por $\widetilde{x} = 1.24 \times 10^1$. El error es 0.5 ulps, el error relativo es 0.8ϵ. A continuación, considere el cálculo $8\widetilde{x}$. El valor exacto es $8x = 98.8$, mientras que el valor calculado es $8\widetilde{x} = 9.92 \times 10^1$. El error ahora es 4.0 ulps, pero el error relativo sigue siendo 0.8ϵ. El error medido en ulps es 8 veces mayor, aunque el error relativo es el mismo.

En general, cuando la base es β, un error relativo fijo expresado en ulps puede oscilar en un factor de hasta β. Y a la inversa, como muestra la Ec. (6.4) anterior, un error fijo de 0.5 ulps da como resultado un error relativo que puede oscilar por β.

La forma más natural de medir el error de redondeo es en ulps. Por ejemplo, el redondeo al número de punto flotante más cercano corresponde a un error menor o igual a 0.5 ulp. Sin embargo, al analizar el error de redondeo causado por varias fórmulas, el error relativo es una mejor medida. Dado que se puede sobrestimar el efecto de redondear al número de punto flotante más cercano por el factor de oscilación de β, las estimaciones de error de las fórmulas serán más estrictas en máquinas con una pequeña β.

Cuando sólo interesa el orden de magnitud del error de redondeo, ulps y ϵ pueden usarse indistintamente, ya que difieren como máximo en un factor de β. Por ejemplo, cuando un número de punto flotante tiene un error de n ulps, eso significa que el número de dígitos contaminados es $\log_\beta n$. Si el error relativo en un cálculo es $n\epsilon$, entonces

$$\text{los dígitos contaminados} \approx \log_\beta n. \tag{6.5}$$

Dígito de Guarda Un método para calcular la diferencia entre dos números de punto flotante es calcular la diferencia exactamente y luego redondearla al número de punto flotante más cercano. Esto es muy caro si los operandos difieren mucho en tamaño. Suponiendo que $p = 3, 2.15 \times 10^{12} - 1.25 \times 10^{-5}$ se calcularía como

$x = 2.15 \times 10^{12}$

$y = 0.000000000000000125 \times 10^{12}$

$x - y = 2.149999999999999875 \times 10^{12}$

que se redondea a 2.15×10^{12}. En lugar de utilizar todos estos dígitos, el Hardware de punto flotante normalmente funciona con un número fijo

de dígitos. Suponga que el número de dígitos que se mantiene es p, y que cuando el operando más pequeño se desplaza hacia la derecha, los dígitos simplemente se descartan (en contraposición al redondeo). Entonces $2.15 \times 10^{12} - 1.25 \times 10^{-5}$ se convierte en

$x = 2.15 \times 10^{12}$
$y = 0.00 \times 10^{12}$
$x - y = 2.15 \times 10^{12}$

La respuesta es exactamente la misma que si la diferencia se hubiera calculado exactamente y luego se hubiera redondeado. Tome otro ejemplo: $10.1 - 9.93$. Esto se convierte en

$x = 1.01 \times 10^{1}$
$y = 0 - 99 \times 10^{1}$
$x - y = 0.02 \times 10^{1}$

La respuesta correcta es 0.17, por lo que la diferencia calculada está desviada en 30 ulps y es incorrecta en todos los dígitos. ¿Qué tan grave puede ser el error?.

Teorema 8 *Usando un formato de punto flotante con parámetros β y p, y calculando las diferencias usando p dígitos, el error relativo del resultado puede ser tan grande como $\beta - 1$.*

Cuando $\beta = 2$, el error relativo puede ser tan grande como el resultado, y cuando $\beta = 10$, puede ser 9 veces mayor. O para decirlo de otra manera, cuando $\beta = 2$, la Ec. (6.5) muestra que el número de dígitos contaminados es $log_2(1/\epsilon) = log_2(2^p) = p$. Es decir, ¡todos los dígitos p del resultado son incorrectos!. Suponga que se agrega un dígito adicional para protegerse contra esta situación (un dígito de guardia). Es decir, el número más pequeño se trunca a $p+1$ dígitos, y luego el resultado de la resta se redondea a p dígitos. Con un dígito de guarda, el ejemplo anterior se convierte en

$x = 1.010 \times 10^{1}$
$y = 0.993 \times 10^{1}$
$x - y = 0.017 \times 10^{1}$

y la respuesta es exacta. Con un solo dígito de guarda, el error relativo del resultado puede ser mayor que ϵ, como en $110 - 8.59$.

$x = 1.10 \times 10^{2}$
$y = 0.085 \times 10^{2}$
$x - y = 1.015 \times 10^{2}$

Esto se redondea a 102, en comparación con la respuesta correcta de 101.41, para un error relativo de 0.006, que es mayor que $\epsilon = 0.005$. En general, el error relativo del resultado puede ser solo un poco mayor que ϵ. De forma más precisa:

Teorema 9 *Si x y y son números de punto flotante en un formato con parámetros β y p, y si la resta se realiza con $p+1$ dígitos (es decir, un dígito de guarda), entonces el error de redondeo relativo en el resultado es menor que 2ϵ.*

Cancelación La última sección se puede resumir diciendo que sin un dígito de guarda, el error relativo cometido al restar dos cantidades cercanas puede ser muy grande. En otras palabras, la evaluación de cualquier expresión que contenga una resta (o una suma de cantidades con signos opuestos) podría resultar en un error relativo tan grande que todos los dígitos carecen de significado (Teorema (8)). Al restar cantidades cercanas, los dígitos más significativos de los operandos coinciden y se cancelan entre sí. Hay dos tipos de cancelación: catastrófica y benigna.

La cancelación catastrófica ocurre cuando los operandos están sujetos a errores de redondeo. Por ejemplo, en la fórmula cuadrática, aparece la expresión $b^2 - 4ac$. Las cantidades b^2 y $4ac$ están sujetas a errores de redondeo ya que son el resultado de multiplicaciones de punto flotante. Suponga que están redondeados al número de punto flotante más cercano y, por lo tanto, tienen una precisión de 0.5 ulp. Cuando se restan, la cancelación puede hacer que muchos de los dígitos precisos desaparezcan, dejando principalmente dígitos contaminados por errores de redondeo. Por tanto, la diferencia puede tener un error de muchos ulps. Por ejemplo, considere $b = 3.34, a = 1.22$ y $c = 2.28$. El valor exacto de $b^2 - 4ac$ es 0.0292. Pero b^2 se redondea a 11.2 y $4ac$ se redondea a 11.1, por lo que la respuesta final es 0.1, que es un error de 70 ulps, aunque $11.2 - 11.1$ es exactamente igual a 0.1. La resta no introdujo ningún error, sino que expuso el error introducido en las multiplicaciones anteriores.

La cancelación benigna ocurre al restar cantidades exactamente conocidas. Si x e y no tienen error de redondeo, entonces, según el Teorema (9), si la resta se realiza con un dígito de guarda, la diferencia $x - y$ tiene un error relativo muy pequeño (menos de 2ϵ).

Una fórmula que presenta una cancelación catastrófica a veces se puede reorganizar para eliminar el problema. Considere nuevamente la fórmula

cuadrática

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ y } x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{6.6}$$

Cuando $b^2 \gg 4ac$, entonces $b^2 - 4ac$ no implica una cancelación y

$$\sqrt{b^2 - 4ac} \approx |b| .$$

Pero la otra suma (resta) en una de las fórmulas tendrá una cancelación catastrófica. Para evitar esto, multiplique el numerador y denominador de x_1 por $-b - \sqrt{b^2 - 4ac}$ (y de manera similar para x_2) para obtener

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \text{ y } x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}} \tag{6.7}$$

Si $b^2 \gg ac$ y $b > 0$, entonces calcular x_1 usando la Ec. (6.6) implicará una cancelación. Por lo tanto, use la Ec. (6.7) para calcular x_1 y (6.6) para x_2. Por otro lado, si $b < 0$, use (6.6) para calcular x_1 y (6.7) para x_2.

La expresión $x^2 - y^2$ es otra fórmula que presenta una cancelación catastrófica. Es más exacto evaluarlo como

$$(x - y)(x + y)$$

A diferencia de la fórmula cuadrática, esta forma mejorada todavía tiene una resta, pero es una cancelación benigna de cantidades sin error de redondeo, no catastrófica. Según el Teorema (9), el error relativo en $x - y$ es como máximo 2ϵ. Lo mismo ocurre con $x + y$. Multiplicar dos cantidades con un pequeño error relativo da como resultado un producto con un pequeño error relativo.

Errores de Redondeo y de Aritmética La aritmética que realiza una computadora es distinta de la aritmética de nuestros cursos de álgebra o cálculo. En nuestro mundo matemático tradicional consideramos la existencia de números con una cantidad infinita de cifras, en la computadora cada número representable tienen sólo un número finito, fijo de cifras (véase **??**), los cuales en la mayoría de los casos es satisfactoria y se aprueba sin más, aunque a veces esta discrepancia puede generar problemas.

Un ejemplo de este hecho lo tenemos en el cálculo de raíces de:

$$ax^2 + bx + c = 0$$

cuando $a \neq 0$, donde las raíces se calculan comúnmente con el algoritmo Ec. (6.6) o de forma alternativa con el algoritmo que se obtiene mediante la racionalización del numerador Ec. (6.7).

Otro algoritmo que podemos implementar es el método de Newton-Raphson[48] para buscar raíces, que en su forma iterativa está dado por

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

en el cual se usa x_0 como una primera aproximación a la raíz buscada y x_n es la aproximación a la raíz después de n iteraciones (sí se converge a ella), donde $f(x) = ax^2 + bx + c$ y $f'(x_i) = 2ax + b$.

Salida del Cálculo de Raíces Para resolver el problema, usamos por ejemplo el siguiente código en Python usando programación procedimental:

```
import math
def f(x, a, b, c):
  """ Evalua la Funcion cuadratica """
  return (x * x * a + x * b + c);
def df(x, a, b):
  """ Evalua la derivada de la funcion cuadratica """
  return (2.0 * x * a + b);
def evalua(x, a, b, c):
  """ Evalua el valor X en la función cuadratica """
  print('Raiz (%1.16f), evaluacion raiz: %1.16e' % (x, f(x, a, b, c)))
def metodoNewtonRapson(x, ni, a, b, c):
  """ Metodo Newton-Raphson x = x - f(x)/f'(x) """
  for i in range(ni):
    x = x - (f(x, a, b, c) / df(x, a, b))
  return x
def raices(A, B, C):
  """ Calculo de raices """
```

[48] También podemos usar otros métodos, como el de Newton Raphson Modificado para acelerar la convergencia

$$x_{i+1} = x_i - \frac{f(x_i)f'(x_i)}{[f'(x_i)]^2 - f(x_i)f''(x_i)}$$

que involucra la función $f(x)$, la primera derivada $f'(x)$ y a la segunda derivada $f''(x)$.

```
    if A == 0.0:
      print("No es una ecuacion cuadratica")
      exit(1)
    # Calculo del discriminante
    d = B * B - 4.0 * A * C
    # Raices reales
    if d >= 0.0:
      print('\nPolinomio (%f) X^2 + (%f )X + (%f) = 0\n' % (A, B, C))
      print('\nChicharronera 1')
      X1 = (-B + math.sqrt(d)) / (2.0 * A)
      X2 = (-B - math.sqrt(d)) / (2.0 * A)
      evalua(X1, A, B, C)
      evalua(X2, A, B, C)
      print('\nChicharronera 2')
      X1 = (-2.0 * C) / (B + math.sqrt(d))
      X2 = (-2.0 * C) / (B - math.sqrt(d))
      evalua(X1, A, B, C)
      evalua(X2, A, B, C)
      # Metodo Newton-Raphson
      print("\n\nMetodo Newton-Raphson")
      x = X1 - 1.0;
      print("\nValor inicial aproximado de X1 = %1.16f" % x)
      x = metodoNewtonRapson(x, 6, A, B, C)
      evalua(x, A, B, C);
      x = X2 - 1.0;
      print("\nValor inicial aproximado de X2 = %1.16f" % x);
      x = metodoNewtonRapson(x, 6, A, B, C)
      evalua(x, A, B, C)
      print("\n")
    else:
      # Raices complejas
      print("Raices Complejas ...")
if __name__ == '__main__':
    raices(1.0, 4.0, 1.0)
```

y generan la siguiente salida:

```
Polinomio (1.000000) X^2 + (4.000000 )X + (1.000000) = 0
```

Chicharronera 1
Raiz (-0.2679491924311228), evaluacion raiz: -4.4408920985006262e-16
Raiz (-3.7320508075688772), evaluacion raiz: 0.0000000000000000e+00

Chicharronera 2
Raiz (-0.2679491924311227), evaluacion raiz: 0.0000000000000000e+00
Raiz (-3.7320508075688759), evaluacion raiz: -5.3290705182007514e-15

Metodo Newton-Raphson
Valor inicial aproximado de X1 = -1.2679491924311228
Raiz (-0.2679491924311227), evaluacion raiz: 0.0000000000000000e+00
Valor inicial aproximado de X2 = -4.7320508075688759
Raiz (-3.7320508075688772), evaluacion raiz: 0.0000000000000000e+00

En esta salida se muestra la raíz calculada y su evaluación en la ecuación cuadrática, la cual debería de ser cero al ser una raíz, pero esto no ocurre en general por los errores de redondeo. Además, nótese el impacto de seleccionar el algoritmo numérico adecuado a los objetivos que persigamos en la solución del problema planteado.

En cuanto a la implementación computacional, el paradigma de programación seleccionado depende la complejidad del algoritmo a implementar y si necesitamos reusar el código generado o no. Otras implementaciones computacionales se pueden consultar en las ligas, en las cuales se usan distintos lenguajes (C, C++, Java y Python) y diferentes paradigmas de programación (secuencial, procedimental y orientada a objetos).

Si lo que necesitamos implementar computacionalmente es una fórmula o conjunto de ellas que generen un código de decenas de líneas, la implementación secuencial es suficiente, si es menor a una centena de líneas puede ser mejor opción la implementación procedimental y si el proyecto es grande o complejo, seguramente se optará por la programación orientada a objetos o formulaciones híbridas de las anteriores.

En última instancia, lo que se persigue en la programación es generar un código: correcto, claro, eficiente, de fácil uso y mantenimiento, que sea flexible, reusable y en su caso portable.

6.2 Trabajando con Punto Flotante

Hay varias trampas en las que incluso los programadores muy experimentados caen cuando escriben código que depende de la aritmética de punto flotante. En esta sección explicamos algunas cosas a tener en cuenta al trabajar con números de punto flotante, es decir, tipos de datos: float (32 bits), double (64 bits) o long double (80 bits).

Problemas de Precisión Como ya vimos en las secciones precedentes, los números de punto flotante se representan en el Hardware de la computadora en fracciones en base 2 (binario). Por ejemplo, la fracción decimal

$$0.125$$

tiene el valor $1/10+2/100+5/1000$, y de la misma manera la fracción binaria

$$0.001$$

tiene el valor $0/2 + 0/4 + 1/8$. Estas dos fracciones tienen valores idénticos, la única diferencia real es que la primera está escrita en notación fraccional en base 10 y la segunda en base 2.

Desafortunadamente, la mayoría de las fracciones decimales no pueden representarse exactamente como fracciones binarias. Como consecuencia, en general los números de punto flotante decimal que usamos en la computadora son sólo aproximados por los números de punto flotante binario que realmente se guardan en la máquina.

El problema es más fácil de entender primero en base 10. Consideremos la fracción 1/3. Podemos aproximarla como una fracción de base 10 como: 0.3 o, mejor: 0.33 o, mejor: 0.333 y así. No importa cuántos dígitos desees escribir, el resultado nunca será exactamente 1/3, pero será una aproximación cada vez mejor de 1/3.

De la misma manera, no importa cuántos dígitos en base 2 quieras usar, el valor decimal 0.1 no puede representarse exactamente como una fracción en base 2. En base 2, 1/10 es la siguiente fracción que se repite infinitamente

$$0.0001100110011001100110011001100110011001100110011...$$

si nos detenemos en cualquier número finito de bits, y tendremos una aproximación. En la mayoría de las máquinas hoy en día, los double se aproximan

usando una fracción binaria con el numerador usando los primeros 53 bits con el bit más significativo y el denominador como una potencia de dos. En el caso de 1/10, la fracción binaria es

$$3602879701896397/2^{55}$$

que está cerca pero no es exactamente el valor verdadero de 1/10.

La mayoría de los usuarios no son conscientes de esta aproximación por la forma en que se muestran los valores. Varios lenguajes de programación como C, C++, Java y Python solamente muestra una aproximación decimal al valor verdadero decimal de la aproximación binaria almacenada por la máquina. En la mayoría de las máquinas, si fuéramos a imprimir el verdadero valor decimal de la aproximación binaria almacenada para 0.1, debería mostrar

$$0.1000000000000000055511151231257827021181583404541015625$$

esos son más dígitos que lo que la mayoría de la gente encuentra útil, por lo que los lenguajes de programación mantiene manejable la cantidad de dígitos al mostrar en su lugar un valor redondeado

$$1/10$$

como

$$0.1$$

sólo hay que recordar que, a pesar de que el valor mostrado resulta ser exactamente 1/10, el valor almacenado realmente es la fracción binaria más cercana posible. Esto queda de manifiesto cuando hacemos

$$0.1 + 0.1 + 0.1 \text{ ó } 0.1 * 3 \text{ ó } 0.1 + 0.2$$[49]

obtendremos

$$0.30000000000000004$$

[49] Para el caso de $0.1 + 0.2$ podemos hacer un programa que nos de 0.3, usando:

```
double x = 0.1;
double y = 0.1;
double z = (x*10.0 + y*10.0) / 10.0
```

que es distinto al 0.3 que esperábamos.

Es de hacer notar que hay varios números decimales que comparten la misma fracción binaria más aproximada. Por ejemplo, los números

$$0.1, 0.10000000000000001 \text{ y}$$

$$0.1000000000000000055511151231257827021181583404541015625$$

son todos aproximados por $3602879701896397/2^{55}$.

Notemos que esta es la verdadera naturaleza del punto flotante binario: no es un error del lenguaje de programación y tampoco es un error en tu código. Verás lo mismo en todos los lenguajes que soportan la aritmética de punto flotante de tu Hardware (a pesar de que en algunos lenguajes por omisión no muestren la diferencia, o no lo hagan en todos los modos de salida). Para una salida más elegante, quizás quieras usar el formateo de cadenas de texto para generar un número limitado de dígitos significativos.

Ejemplo, el número decimal 9.2 se puede expresar exactamente como una relación de dos enteros decimales 92/10, los cuales se pueden expresar exactamente en binario como $0b1011100/0b1010$. Sin embargo, la misma proporción almacenada como un número de punto flotante nunca es exactamente igual a 9.2:

usando 32 bits obtenemos: 9.1999999809265136
usando 64 bits obtenemos: 9.19999999999999928945

ya que se guarda como la fracción: $5179139571476070/2^{49}$.

Otro ejemplo, si tomamos el caso del número 0.02 y vemos su representación en el lenguaje de programación Python, tenemos que:

```
import decimal
print(decimal.Decimal(0.02))
```

y el resultado es:

0.020000000000000000416333634234433702658861875534057617187 5

Además, tenemos la no representabilidad de π (y $\pi/2$), esto significa que un intento de cálculo de $tan(\pi/2)$ no producirá un resultado de infinito, ni

siquiera se desbordará en los formatos habituales de punto flotante. Simplemente no es posible que el Hardware de punto flotante estándar intente calcular $tan(\pi/2)$, porque $\pi/2$ no se puede representar exactamente. Este cálculo en C:

```
doble pi =3.14159265358979323846264338327 95;
tan (pi / 2.0);
```

dará un resultado de $1.633123935319537e+16$. En precisión simple usando tanf, el resultado será -22877332.0.

De la misma manera, un intento de cálculo de $sin(\pi)$ no arrojará cero. El resultado será $1.2246467991473532e-16$ en precisión doble.

Si bien la suma y la multiplicación de punto flotante son conmutativas ($a+b=b+a$ y $a\times\ b=b\times\ a$), no son necesariamente asociativas . Es decir, $(a+b)+c$ no es necesariamente igual a $a+(b+c)$. Usando aritmética decimal significativa de 7 dígitos:

$$a=1234.567, b=45.67834, c=0.0004$$

$(a+b)+c$:

$1234.567+45.67834$ 1280.24534 se redondea a 1280.245
$1280.245+0.0004$ 1280.2454 se redondea a 1280.245

$a+(b+c)$:

$45.67834+0.0004$ 45.67874
$1234.567+45.6787$ 1280.24574 se redondea a 1280.246

Tampoco son necesariamente distributivos. Es decir $(a+b)\times c$ puede no ser lo mismo que $a\times c+b\times c$:

$1234.567*3.333333=4115.223$, $1.234567*3.333333=4.115223$,
$4115.223+4.115223=4119.338$

pero

$1234.567+1.234567=1235.802$, $1235.802*3.333333=4119.340$

Algunos Trucos Supongamos que necesitamos hacer el siguiente cálculo:

$$4.56 * 100$$

la respuesta es:

$$455.9999999999994$$

que no es la esperada. ¿Qué podemos hacer para obtener lo que esperamos?. Por ejemplo, con Python, podemos usar:

```
from sympy import *
print(nsimplify(4.56 * 100, tolerance=1e-1))
```

que nos dará el 456 esperado.

SymPy es un paquete matemático simbólico para Python, y nsimplify toma un número de punto flotante y trata de simplificarlo como una fracción con un denominador pequeño, la raíz cuadrada de un entero pequeño, una expresión que involucra constantes famosas, etc.

Por ejemplo, supongamos que algún cálculo arrojó 4.242640687119286 y sospechamos que hay algo especial en ese número. Así es como puede probar de dónde vino:

```
from sympy import *
print(nsimplify(4.242640687119286))
```

que nos arrojará:

$$3 * sqrt(2)$$

Tal vez hagamos un cálculo numérico y encontremos una expresión simple para el resultado y eso sugiera una solución analítica. Creo que una aplicación más común de nsimplify podría ser ayudarte a recordar fórmulas medio olvidadas. Por ejemplo, quizás estés oxidado con tus identidades trigonométricas, pero recuerdas que cos(p/6) es algo especial.

```
from sympy import *
print(nsimplify(cos(pi/6)))
```

que nos entregará:

$$sqrt(3)/2$$

O, para tomar un ejemplo más avanzado, supongamos que recordamos vagamente que la función gamma toma valores reconocibles en valores semienteros, pero no recordamos exactamente cómo. Tal vez algo relacionado con π o e. Puede sugerir que nsimplify incluya expresiones con π y e en su búsqueda.

```
from sympy import *
print(nsimplify(gamma(3.5), constants=[pi, E]))
```

obteniendo:

$$15 * sqrt(pi)/8$$

También podemos darle a nsimplify una tolerancia, pidiéndole que encuentre una representación simple dentro de una vecindad del número. Por ejemplo, aquí hay una manera de encontrar aproximaciones a π.

```
from sympy import *
print(nsimplify(pi, tolerance=1e-5))
```

obteniendo:

$$355/113$$

con una tolerancia más amplia, devolverá una aproximación más simple.

```
from sympy import *
print(nsimplify(pi, tolerance=1e-2))
```

obteneindo:

$$22/7$$

finalmente, aquí hay una aproximación de mayor precisión a π que no es exactamente simple:

```
from sympy import *
print(nsimplify(pi, tolerance=1e-7))
```

obteniendo:

$$exp(141/895 + sqrt(780631)/895)$$

Seno de un Googol ¿Cómo se evalúa el seno de un número grande en aritmética de punto flotante? ¿Qué significa el resultado?.

Seno de un billón Comencemos por encontrar el seno de un billón (10^{12}) usando aritmética de punto flotante. Hay un par de maneras de pensar en esto. El número de punto flotante $t = 1.0e12$ solo se puede representar con 15 o 16 cifras decimales significativas (para ser precisos, 53 bits[50]), por lo que podría considerarlo como un representante del intervalo de números que tienen todos la misma representación de punto flotante. Cualquier resultado que sea seno de un número en ese intervalo debe considerarse correcto.

Otra forma de ver esto es decir que t se puede representar exactamente -su representación binaria requiere 42 bits y tenemos 53 bits de precisión significativa disponibles-, por lo que esperamos $sin(t)$ devolver el seno de exactamente un billón, con precisión total.

Resulta que la aritmética del IEEE hace lo último, calculando $sin(1e12)$ correctamente hasta 16 dígitos.

Aquí está el resultado en Python.

```
sin(1.0e12)
-0.6112387023768895
```

y verificado por Mathematica calculando el valor con 20 decimales

```
In:= N[Sin[10^12], 20]
out= -0.61123870237688949819
```

Reducción de alcance tenga en cuenta que el resultado anterior no es el que obtendría si primero tomara el resto al dividir por 2π y luego tomará el seno.

```
sin(1.0e12 % (2*pi))
-0,6112078499756778
```

[50] Un doble IEEE 754 tiene 52 bits significativos, pero estos bits pueden representar 53 bits de precisión porque el primer bit de la parte fraccionaria es siempre 1, por lo que no es necesario representarlo.

Esto tiene sentido. El resultado de dividir un billón por la representación de punto flotante de 2π, 159154943091.89536, es correcto con precisión de punto flotante total. Pero la mayor parte de esa precisión está en la parte entera. La parte fraccionaria solo tiene cinco dígitos de precisión, por lo que debemos esperar que el resultado anterior sea correcto hasta un máximo de cinco dígitos. De hecho, tiene una precisión de cuatro dígitos.

Cuando su procesador calcula, $sin(1e12)$ no toma ingenuamente el resto por 2π y luego calcula el seno. Si así fuera, obtendríamos cuatro cifras significativas en lugar de 16.

Empezamos diciendo que había dos maneras de ver el problema y, según la primera, devolver sólo cuatro cifras significativas sería bastante razonable. Si pensamos en el valor t como una cantidad medida, medida con la precisión de la aritmética de punto flotante (aunque casi nada se puede medir con tanta precisión), entonces todo lo que deberíamos esperar sería cuatro cifras significativas. Pero las personas que diseñaron la implementación de la función seno en su procesador hicieron más de lo que se esperaba que hicieran, encontrando el seno de un billón con total precisión. Lo hacen utilizando un algoritmo de reducción de rango que conserva mucha más precisión que hacer una división ingenuamente.

¿Seno de un Googol? ¿Qué pasa si intentamos tomar el seno de un número ridículamente grande como un Googol (10^{100})? Esto no funcionará porque un Googol no se puede representar exactamente como un número de punto flotante (es decir, como un doble IEEE 754). No es demasiado grande; Los números de punto flotante pueden ser tan grandes como alrededor de 10^{308}. El problema es que un número tan grande no se puede representar con total precisión. Pero podemos representar 2^{333} exactamente, y un Googol está entre 2^{332} y 2^{333}. Y sorprendentemente, la función sinusoidal de Python (o más bien la función sinusoidal integrada en el procesador AMD de mi computadora) devuelve el resultado correcto con total precisión.

```
sin(2**333)
0.9731320373846353
```

¿Cómo sabemos que esto es correcto? Lo verifiqué en Mathematica:

```
In:= sin[2.0^333]
Out= 0.9731320373846353
```

¿Cómo sabemos que Mathematica tiene razón? Podemos hacer una reducción de rango ingenua usando precisión extendida, digamos 200 decimales.

```
In:= p = N[Pi, 200]
In:= theta = x - IntegerPart[x/ (2 p)] 2 p
Out= 1.80312861783346995593841966 89...
```

y verificar que el seno del valor reducido del rango sea correcto.

```
In:= Sin[theta]
Out= 0.97313203738463526...
```

Interpretación Aritmética de Intervalos debido a que los números de punto flotante tienen 53 bits de precisión, todos los números reales entre $2^{56} - 2^2$ y $2^{56} + 2^2$ se representan como el número de punto flotante 2^{56}. Este es un rango de ancho 8, más ancho que 2π, por lo que los senos de los números en este rango cubren los posibles valores de seno, es decir, $[-1, 1]$. Entonces, si piensa en un número de punto flotante como una muestra del conjunto de todos los números reales con la misma representación binaria, cada valor posible de seno es un valor de retorno válido para números mayores que 2^{56}. Pero si piensa que un número de punto flotante se representa exactamente a sí mismo, entonces tiene sentido preguntar por el seno de números como 2^{333} y mayores, hasta los límites de la representación de punto flotante[51].

No Pruebes por la Igualdad Cuando se usa Punto Flotante no es recomendable escribir código como el siguiente[52]:

```
double x;
double y;
...
if (x == y) {...}
```

[51] El mayor exponente de un doble IEEE es 1023, y el mayor significado es $2 - 2^{-53}$ (es decir, todos los unos), por lo que el mayor valor posible de un doble es $(2^{53} - 1)2^{1024-53}$ y de hecho la expresión de Python sin((2**53 - 1)*2**(1024-53)) devuelve el valor correcto con 16 cifras significativas.

[52] Sin pérdida de generalidad usamos algún lenguaje de programación en particular para mostrar los ejemplos, pero esto pasa en todos los lenguajes que usan operaciones de Punto Flotante.

La mayoría de las operaciones de punto flotante implican al menos una pequeña pérdida de precisión y, por lo tanto, incluso si dos números son iguales para todos los fines prácticos, es posible que no sean exactamente iguales hasta el último bit, por lo que es probable que la prueba de igualdad falle. Por ejemplo:

```
double x = 10;
double y = sqrt(x);
y *= y;
if (x == y)
cout << "La reaiz cuadrada es exacta\n";
else
cout << x-y << "\n";
```

el código imprime: $-1.778636e-015$, aunque en teoría, elevar al cuadrado debería deshacer una raíz cuadrada, la operación de ida y vuelta es ligeramente inexacta. En la mayoría de los casos, la prueba de igualdad anterior debe escribirse de la siguiente manera:

```
double tolerancia = ...
if (fabs(x - y) < tolerancia) {...}
```

Aquí la tolerancia es un umbral que define lo que está "lo suficientemente cerca" para la igualdad. Esto plantea la pregunta de qué tan cerca está lo suficientemente cerca. Esto no puede responderse en abstracto; tienes que saber algo sobre tu problema particular para saber qué tan cerca está lo suficientemente cerca en tu contexto.

Por ejemplo: ¿hay alguna garantía de que la raíz cuadrada de un cuadrado perfecto sea devuelta exactamente?, por ejemplo si hago $sqrt(81.0) == 9.0$, por lo visto anteriormente, la respuesta es no, pero podríamos cambiar la pregunta por $9.0 * 9.0 == 81.0$, esto funcionará siempre que el cuadrado esté dentro de los límites de la magnitud del punto flotante.

Por otro lado, es posible que las expectativas de las matemáticas no se cumplan en el campo del cálculo de punto flotante. Por ejemplo, se sabe que

$$(x + y)(x - y) = x^2 + y^2$$

y esta otra

$$\sin^2\theta + \cos^2\theta = 1$$

sin embargo, no se puede confiar en estos hechos cuando las cantidades involucradas son el resultado de un cálculo de punto flotante. Además, el error de redondeo puede afectar la convergencia y precisión de los procedimientos numéricos iterativos.

Aún más y sin pérdida de generalidad, si comparamos usando el lenguaje de programación Python:

```
print(0.1 + 0.2 == 0.3)
print(0.2 + 0.2 +0.2 == 0.6)
print(1.2 + 2.4 + 3.6 == 7.2)
print(0.1 + 0.2 <= 0.3)
```

en todos los casos dará un resultado de falso, además:

```
print(10.4 + 20.8 > 31.2)
print(0.8 - 0.1 > 0.7)
```

el resultado será verdadero. Por ello es siempre conveniente ver con que números trabajamos usando algo como:

```
print(format(0.1, ".17g"))
print(format(0.2, ".17g"))
print(format(0.3, ".17g"))
```

así cuando sumemos $0.1 + 0.2$, podemos ver el verdadero resultado:

```
print(print(format(0.1 + 0.2, ".17g"))
```

Teniendo esto en cuenta podemos comparar usando:

```
import math
print(math.isclose(0.1 + 0.2, 0.3)
```

nos dará la respuesta esperada. Podemos ajustar la tolerancia relativa usando:

```
import math
print(math.isclose(0.1 + 0.2, 0.3, rel_tol = 1e-20)
```

que en este caso nos dirá que es falso pues no son iguales en los 20 primeros dígitos solicitados.

Para las comparaciones >= o <=, por ejemplo para $a + b <= c$ se debe usar:

```
a, b , c = 0.1, 0.2, 0.3
print(math.isclose(a + b, c) or (a +c < c))
```

¿Cómo Compara Python los Flotantes y los Enteros? veamos los siguientes ejemplos:

```
>>> 9007199254740992 == 9007199254740992.0
True
>>> 9007199254740993 == 9007199254740993.0
False
>>> 9007199254740994 == 9007199254740994.0
True
```

¿Qué fue lo que salió mal?, para desmenuzar lo que pasó, veamos primero:

Representación IEEE-754 de 9007199254740992.0 comencemos con el valor del primer escenario: 9007199254740992.0, resulta que este número es en realidad es 2^{53}. Eso significa que en la representación binaria sería 1 seguido de 53 ceros:

$$10000000000000000000000000000000000000000000000000000.$$

la forma normalizada será $1.0 * 2^{53}$, que se obtendrá desplazando los bits 53 lugares a la derecha, lo que nos dará el exponente 53.

Además, recordemos que la mantisa tiene solo 52 bits de ancho y estamos desplazando nuestro valor 53 bits hacia la derecha, lo que significa que se perderá el bit menos significativo (LSB) de nuestro valor original. Sin embargo, como todos los bits aquí son 0, no hay ningún daño y aún podemos representar 9007199254740992.0 con precisión. Resumamos los componentes de 9007199254740992.0:

Bit de signo: 0

Exponente: $53 + 1023$ (bias) = 1076 (10000110100 en binario)

Mantisa: 0000000000000000000000000000000000000000000000000000

Representación IEEE-754:

0100001101000000000000000000000000000000000000000000000000000000

Representación IEEE-754 de 9007199254740993.0 el valor del segundo escenario de prueba es 9007199254740993.0 el que es uno mayor que el valor anterior. Su representación binaria se puede obtener simplemente sumando 1 a la representación binaria del valor anterior:

$$10000000000000000000000000000000000000000000000001.0$$

nuevamente, para convertir esto a la forma normalizada, tendremos que desplazarlo a la derecha 53 bits, lo que nos dará el exponente 53.

Sin embargo, la mantisa tiene sólo 52 bits de ancho. Cuando desplazamos nuestro valor a la derecha en 53 bits, el bit menos significativo (LSB) con el valor 1 se perderá, lo que dará como resultado que la mantisa sea todo ceros. Esto conduce a los siguientes componentes:

Bit de signo: 0

Exponente: $53 + 1023$ (bias) $= 1076$ (10000110100 en binario)

Mantisa: 0000000000000000000000000000000000000000000000000000

Representación IEEE-754:

0100001101000000000000000000000000000000000000000000000000000000

Observe que la representación IEEE-754 de 9007199254740993.0 es la misma que la de 9007199254740992.0. Esto significa que en su representación en memoria del número 9007199254740993.0 en realidad se representa como 9007199254740992.0.

Esto explica por qué Python da el resultado para

$$9007199254740993 == 9007199254740993.0$$

False, porque lo ve como una comparación entre

$$9007199254740993 \text{ y } 9007199254740992.0.$$

Representación IEEE-754 de 9007199254740994.0 El número en el tercer y último escenario de prueba es 9007199254740994.0, que es 1 más que el valor anterior. Su representación binaria se puede obtener sumando 1 a la representación binaria de 9007199254740993.0, dándonos:

$$10000000000000000000000000000000000000000000000010.0$$

esto también tiene 54 bits antes del punto binario y requiere un desplazamiento a la derecha de 53 bits para convertirlo a la forma normalizada, lo que nos da el exponente 53.

Observe que esta vez, el segundo bit menos significativo (LSB) tiene el valor 1. Por lo tanto, cuando desplazamos este número a la derecha 53 bits, se convierte en el LSB de la mantisa (que tiene 52 bits de ancho). Los componentes se ven así:

Bit de signo: 0

Exponente: $53 + 1023$ (bias) $= 1076$ (10000110100 en binario)

Mantisa: 0000000000000000000000000000000000000000000000000001

Representación IEEE-754:

0100001101000000000000000000000000000000000000000000000000000001

A diferencia de 9007199254740993.0, la representación IEEE-754 del número 9007199254740994.0 puede representarlo exactamente sin pérdida de precisión. Por tanto, el resultado de

$$9007199254740994 == 9007199254740994.0$$

es True en Python.

El estándar IEEE-754 y la aritmética de punto flotante son inherentemente complejos y comparar números de punto flotante no es sencillo. Lenguajes como C y Java implementan la promoción de tipos implícita, convirtiendo números enteros en dobles y comparándolos poco a poco. Sin embargo, esto aún puede dar lugar a resultados inesperados debido a la pérdida de precisión.

Python es único a este respecto. Tiene números enteros de precisión infinita, lo que hace que la promoción de tipos sea inviable en muchas situaciones. En consecuencia, Python utiliza su algoritmo especializado para comparar estos números, que a su vez tiene casos extremos.

Ya sea que esté utilizando C, Java, Python u otro lenguaje, es recomendable utilizar funciones de biblioteca para comparar valores de punto flotante en lugar de realizar comparaciones directas para evitar posibles errores.

Cuando Cómputo de Alto Rendimiento no es Alto Rendimiento
A todo el mundo le importa que los códigos se ejecuten rápidamente en sus computadoras. Las mejoras de Hardware de las últimas décadas lo han hecho posible. Pero, ¿qué tan bien estamos aprovechando las aceleraciones del Hardware?

Considere estos dos ejemplos de código C++. Supongamos aquí n = 10000000.

```
void sub(int* a, int* b) {
    for (int i=0; i<n; ++i)
        a[i] = i + 1;
    for (int i=0; i<n; ++i)
        b[i] = a[i];
}

void sub(int* a, int* b) {
    for (int i=0; i<n; ++i) {
        const int j = i + 1;
        a[i] = j;
        b[i] = j;
    }
}
```

¿Cuál corre más rápido? Ambos son simples y dan resultados idénticos (suponiendo que no haya alias). Sin embargo, en las arquitecturas modernas, dependiendo de la configuración de compilación, una generalmente se ejecutará significativamente más rápido que la otra.

En particular, se esperaría que el fragmento 2 se ejecutara más rápido que el fragmento 1. En el fragmento 1, los elementos de la matriz "a", que es demasiado grande para almacenarse en caché, deben recuperarse de la memoria después de escribirse, pero esto no es necesario para Fragmento 2. La tendencia durante más de dos décadas ha sido que la velocidad de computación de los sistemas recién entregados crezca mucho más rápido que la velocidad de la memoria, y la disparidad es extrema hoy en día. El rendimiento de estos núcleos depende casi por completo de la velocidad del ancho de banda de la memoria. Así, el fragmento 2, una versión de bucle fusionado del fragmento 1, mejora la velocidad al reducir el acceso a la memoria principal.

Es poco probable que bibliotecas como C++ STL ayuden, ya que esta operación es demasiado especializada para esperar que una biblioteca la admita (especialmente la versión de bucle fusionado). Además, el compilador no puede fusionar de forma segura los bucles automáticamente sin instrucciones específicas de que los punteros no tengan alias, y aun así no se garantiza que lo haga.

Afortunadamente, los lenguajes informáticos de alto nivel desde la década de 1950 han elevado el nivel de abstracción de la programación para todos nosotros. Naturalmente, a muchos de nosotros nos gustaría simplemente implementar la lógica empresarial requerida en nuestros códigos y dejar que el compilador y el Hardware hagan el resto. Pero, lamentablemente, no siempre se puede simplemente colocar el código en una computadora y esperar que se ejecute rápidamente. Cada vez más, a medida que el Hardware se vuelve más complejo, prestar atención a la arquitectura subyacente es fundamental para obtener un alto rendimiento.

Preocúpate más por la Suma y la Resta que por la Multiplicación y la División Los errores relativos en la multiplicación y división son siempre pequeños. La suma y la resta, por otro lado, pueden resultar en una pérdida completa de precisión. Realmente el problema es la resta; la suma sólo puede ser un problema cuando los dos números que se agregan tienen signos opuestos, por lo que puedes pensar en eso como una resta. Aún así, el código podría escribirse con un "+" que sea realmente una resta.

La resta es un problema cuando los dos números que se restan son casi iguales. Cuanto más casi iguales sean los números, mayor será el potencial de pérdida de precisión. Específicamente, si dos números están de acuerdo con n bits, se pueden perder n bits de precisión en la resta. Esto puede ser más fácil de ver en el extremo: si dos números no son iguales en teoría pero son iguales en su representación de máquina, su diferencia se calculará como cero, 100% de pérdida de precisión.

Aquí hay un ejemplo donde tal pérdida de precisión surge a menudo. La derivada de una función f en un punto x se define como el límite de

$$(f(x+h) - f(x))/h$$

cuando h llega a cero. Entonces, un enfoque natural para calcular la derivada de una función sería evaluar

$$(f(x+h) - f(x))/h$$

para alguna h pequeña. En teoría, cuanto menor es h, mejor se aproxima esta fracción a la derivada. En la práctica, la precisión mejora por un tiempo, pero más allá de cierto punto, valores más pequeños de h resultan en peores aproximaciones a la derivada. A medida que h disminuye, el error de aproximación se reduce pero el error numérico aumenta. Esto se debe a que la resta

$$f(x+h) - f(x)$$

se vuelve problemática. Si toma h lo suficientemente pequeño (después de todo, en teoría, más pequeño es mejor), entonces $f(x+h)$ será igual a $f(x)$ a la precisión de la máquina. Esto significa que todas las derivadas se calcularán como cero, sin importar la función, si solo toma h lo suficientemente pequeño. Aquí hay un ejemplo que calcula la derivada de $sin(x)$ en $x = 1$.

```
cout << std::setprecision(15);
for (int i = 1; i < 20; ++i)
{
double h = pow(10.0, -i);
cout << (sin(1.0+h) - sin(1.0))/h << "\n";
}
cout << "El verdadero resultado es: " << cos(1.0) << "\n";
```

Aquí está la salida del código anterior. Para que la salida sea más fácil de entender, los dígitos después del primer dígito incorrecto se han reemplazado por puntos.

```
0.4.........
0.53........
0.53........
0.5402......
0.5402......
0.540301....
0.5403022...
0.540302302.
0.54030235...
0.5403022...
0.540301....
0.54034.....
0.53........
```

0.544.....
0.55.....
0
0
0
0
El verdadero resultado es: 0.54030230586814

La precisión[53] mejora a medida que h se hace más pequeña hasta que $h = 10^{-8}$. Pasado ese punto, la precisión decae debido a la pérdida de precisión en la resta. Cuando $h = 10^{-16}$ o menor, la salida es exactamente cero porque $sin(1.0 + h)$ es igual a $sin(1.0)$ a la precisión de la máquina. (De hecho, 1 + h equivale a 1 a la precisión de la máquina. Más sobre eso a continuación).

¿Qué haces cuando tu problema requiere resta y va a causar una pérdida de precisión? A veces la pérdida de precisión no es un problema; los dobles comienzan con mucha precisión de sobra. Cuando la precisión es importante, a menudo es posible usar algún truco para cambiar el problema de modo que no requiera resta o no requiera la misma resta con la que comenzaste.

Los Números de Punto Flotante Tienen Rangos Finitos Todos saben que los números de punto flotante tienen rangos finitos, pero esta limitación puede aparecer de manera inesperada. Por ejemplo, puede encontrar sorprendente la salida de las siguientes líneas de código

```
float f = 16777216;
cout << f << " " << f+1 << "\n";
```

Este código imprime el valor 16777216 dos veces. ¿Que pasó? De acuerdo con la especificación IEEE para aritmética de punto flotante, un tipo flotante tiene 32 bits de ancho. Veinticuatro de estos bits están dedicados al significado (lo que solía llamarse la mantisa) y el resto al exponente. El número 16777216 es 2^{24} y, por lo tanto, a la variable flotante f no le queda precisión para representar $f + 1$. Ocurriría un fenómeno similar para 2^{53} si f fuera

[53] Los resultados anteriores se calcularon con Visual C ++ 2008. Cuando se compiló con gcc 4.2.3 en Linux, los resultados fueron los mismos, excepto los últimos cuatro números. Donde VC ++ produjo ceros, gcc produjo números negativos: -0.017 ..., - 0.17 ..., -1.7 ... y 17 ....

del tipo double porque un doble de 64 bits dedica 53 bits al significado. El siguiente código imprime 0 en lugar de 1.

```
x = 9007199254740992; // 2^53
cout << ((x+1) - x) << "\n";
```

También podemos quedarnos sin precisión al agregar números pequeños a números de tamaño moderado. Por ejemplo, el siguiente código imprime "¡Lo siento!" porque DBL_EPSILON (definido en float.h) es el número positivo más pequeño ϵ tal que $1 + \epsilon! = 1$ cuando se usan tipos dobles.

```
x = 1.0;
y = x + 0.5*DBL_EPSILON;
if (x == y)
cout << "¡Lo siento!\n";
```

De manera similar, la constante FLT_EPSILON es el número positivo más pequeño ϵ tal que $1 + \epsilon$ no es 1 cuando se usan tipos flotantes.

¿Por qué los Flotantes IEEE Tienen Dos Ceros: $+0$ y -0? Aquí hay un detalle extraño de la aritmética de punto flotante IEEE: las computadoras tienen dos versiones de 0: cero positivo y cero negativo. La mayoría de las veces, la distinción entre $+0$ y -0 no importa, pero de vez en cuando las versiones firmadas del cero son útiles.

Si una cantidad positiva llega a cero, se convierte en $+0$. Y si una cantidad negativa llega a cero, se convierte en -0. Podría pensar en $+0$ (respectivamente, -0) como el patrón de bits para un número positivo (negativo) demasiado pequeño para representarlo.

El estándar de punto flotante IEEE dice que $1/+0$ debería ser $+infinito$ y $1/-0$ debería ser $-infinito$. Esto tiene sentido si interpreta $+/-0$ como el fantasma de un número que se desbordó dejando solo su signo. El recíproco de un número positivo (negativo) demasiado pequeño para representarlo es un número positivo (negativo) demasiado grande para representarlo.

Para demostrar esto, ejecute el siguiente código C:

```
int main()
{
    double x = 1e-200;
```

```
    double y = 1e-200 * x;
    printf("Reciproco de +0: %g\n", 1/y);
    y = -1e-200*x;
    printf("Reciproco de -0: %g\n", 1/y);
}
```

En Linux con gcc, la salida es:

```
Reciproco de +0: inf
Reciproco de -0: -inf
```

Sin embargo, hay algo acerca de los ceros firmados y las excepciones que no tiene sentido. El informe acertadamente denominado "Lo que todo informático debería saber sobre la aritmética de punto flotante" tiene lo siguiente que decir sobre los ceros con signo.

> En aritmética IEEE, es natural definir $log0 = -\infty$ y $logx$ como un NaN cuando $x < 0$. Suponga que x representa un pequeño número negativo que se ha desbordado a cero. Gracias al cero con signo, x será negativo, por lo que log puede devolver un NaN. Sin embargo, si no hubiera un cero con signo, la función logarítmica no podría distinguir un número negativo subdesbordado de 0 y, por lo tanto, tendría que devolver $-\infty$.

Esto implica que $log(-0)$ debe ser NaN y $log(+0)$ debe ser $-\infty$. Eso tiene sentido, pero eso no es lo que sucede en la práctica. La función de log devuelve $-\infty$ para $+0$ y -0.

Ejecuté el siguiente código en C:

```
int main()
{
    double x = 1e-200;
    double y = 1e-200 * x;
    printf("Log de +0: %g\n", log(y));
    y = -1e-200*x;
    printf("Log de -0: %g\n", log(y));
}
```

En Linux, el código imprime:

```
Log de +0: -inf
Log de -0: -inf
```

Use Logaritmos para Evitar Desbordamiento y Subdesbordamiento Las limitaciones de los números de punto flotante descritos en la sección anterior provienen de tener un número limitado de bits en el significado. El desbordamiento y el subdesbordamiento resultan de tener también un número finito de bits en el exponente. Algunos números son demasiado grandes o demasiado pequeños para almacenarlos en un número de punto flotante.

Muchos problemas parecen requerir calcular un número de tamaño moderado como la razón de dos números enormes. El resultado final puede ser representable como un número de punto flotante aunque los resultados intermedios no lo sean. En este caso, los logaritmos proporcionan una salida. Si desea calcular M/N para grandes números M y N, calcule $log(M) - log(N)$ y aplique $exp()$ al resultado. Por ejemplo, las probabilidades a menudo implican proporciones de factoriales, y los factoriales se vuelven astronómicamente grandes rápidamente. Para $N > 170$, $N!$ es mayor que DBL_MAX, el número más grande que puede representarse por un doble (sin precisión extendida). Pero es posible evaluar expresiones como 200!/(190!10!) Sin desbordamiento de la siguiente manera:

```
x = exp( logFactorial(200) - logFactorial(190) - logFactorial(10) );
```

Una función logFactorial simple pero ineficiente podría escribirse de la siguiente manera:

```
double logFactorial(int n)
{
    double sum = 0.0;
    for (int i = 2; i <= n; ++i) sum += log((double)i);
    return sum;
}
```

Un mejor enfoque sería utilizar una función de registro gamma si hay una disponible. Consulte Cómo calcular las probabilidades binomiales para obtener más información.

Las Operaciones Numéricas no Siempre Devuelven Números Debido a que los números de punto flotante tienen sus limitaciones, a veces las ope-raciones de punto flotante devuelven "infinito" como una forma de decir "el resultado es más grande de lo que puedo manejar". Por ejemplo, el siguiente código imprime 1. # *INF* en Windows e inf en Linux.

```
x = DBL_MAX;
cout << 2*x << "\n";
```

A veces, la barrera para devolver un resultado significativo tiene que ver con la lógica en lugar de la precisión finita. Los tipos de datos de punto flotante representan números reales (a diferencia de los números complejos) y no hay un número real cuyo cuadrado sea -1. Eso significa que no hay un número significativo que devolver si el código solicita $sqrt(-2)$, incluso con una precisión infinita. En este caso, las operaciones de punto flotante devuelven *NaN*. Estos son valores de punto flotante que representan códigos de error en lugar de números. Los valores *NaN* se muestran como 1. # *IND* en Windows y *NAN* en Linux.

Una vez que una cadena de operaciones encuentra un *NaN*, todo es un *NaN* de ahí en adelante. Por ejemplo, suponga que tiene un código que equivale a algo como lo siguiente:

```
if (x - x == 0)
// hacer algo
```

¿Qué podría impedir que se ejecute el código que sigue a la instrucción if? Si x es un *NaN*, entonces también lo es $x - x$ y los *NaN* no equivalen a nada. De hecho, los *NaN* ni siquiera se igualan. Eso significa que la expresión $x == x$ se puede usar para probar si x es un número (posiblemente infinito). Para obtener más información sobre infinitos y *NaN*, consulte las excepciones de punto flotante IEEE en C ++.

Trabajando con Factoriales Los cálculos de probabilidad a menudo implican tomar la razón de números muy grandes para producir un número de tamaño moderado. El resultado final puede caber dentro de un doble con espacio de sobra, pero los resultados intermedios se desbordarían. Por ejemplo, suponga que necesita calcular el número de formas de seleccionar 10 objetos de un conjunto de 200. ¡Esto es 200!/(190!10!), Aproximadamente $2.2e16$. Pero 200! y 190! desbordaría el rango de un doble.

Hay dos formas de solucionar este problema. Ambos usan la siguiente regla: Use trucos algebraicos para evitar el desbordamiento.

El primer truco es reconocer que

$$200! = 200 * 199 * 198 * ... * 191 * 190!$$

y así

$$200!/(190!10!) = 200 * 199 * 198 * ... * 191/10!$$

esto ciertamente funciona, pero está limitado a factoriales.

Una técnica más general es usar logaritmos para evitar el desbordamiento: tome el logaritmo de la expresión que desea evaluar y luego exponga el resultado. En este ejemplo

$$log(200!/(190!10!)) = Log(200!) - log(190!) - log(10!)$$

si tiene un código que calcula el logaritmo de los factoriales directamente sin calcular primero los factoriales, puede usarlo para encontrar el logaritmo del resultado que desea y luego aplicar la función exp.

Calcular Inverso del Factorial Dado un número positivo x, ¿cómo se puede encontrar un número n tal que $n! = x$, o tal que $n! \approx x$?. El Software matemático tiende a trabajar con la función gamma en lugar de factoriales porque $\Gamma(x)$ extiende $x!$ en números reales, con la relación $\Gamma(x + 1) = x!$. El lado izquierdo a menudo se toma como una definición del lado derecho cuando x no es un número entero positivo.

No sólo preferimos trabajar con la función gamma, sino que es más fácil trabajar con el logaritmo de la función gamma para evitar el desbordamiento.

El siguiente código Python encuentra el inverso del logaritmo de la función gamma utilizando el método de bisección. Este método requiere un límite superior e inferior. Si solo pasamos valores positivos, 0 sirve como límite inferior. Podemos encontrar un límite superior probando potencias de 2 hasta obtener algo lo suficientemente grande.

```
from scipy.special import gammaln
from scipy.optimize import bisect
import math as mt
def inverse_log_gamma(logarg):
    assert(logarg > 0)
    a = 1
    b = 2
    while b < logarg:
        b = b*2
    return bisect(lambda x: gammaln(x) - logarg, a, b)
def inverse_factorial(logarg):
```

```
g = inverse_log_gamma(logarg)
return round(g)-1
```

Tenga en cuenta que la función inverse_factorial toma el logaritmo del valor cuyo inverso del factorial desea calcular. Por ejemplo:

```
print(inverse_factorial(mt.log(mt.factorial(42))))
```
regresa 42.

Trabajar en la escala logarítmica nos permite trabajar con números mucho mayores. El factorial de 171 es mayor que el mayor número de doble precisión de IEEE, por lo que inverse_factorial no podría devolver ningún valor mayor que 171 si pasáramos x en lugar de log x. Pero al tomar log x como argumento, podemos calcular factoriales inversos de números tan grandes que el factorial se desborda.

Por ejemplo, supongamos que queremos encontrar el valor de m tal que $m!$ es el factorial más cercano a $\sqrt{(2024!)}$. Podemos usar el código.

```
print(inverse_factorial(gammaln(2025)/2))
```

para encontrar $m = 1112$ aunque $1112!$ es del orden de 10^{2906}, mucho mayor que el mayor número de punto flotante representable, que es del orden de 10^{308}.

Cuando $n! = x$ no tiene una solución exacta, existe alguna opción sobre qué valor de n devolver como inverso del factorial de x. El código anterior minimiza la distancia desde $n!$ a x en una escala logarítmica. Es posible que desee modificar el código para minimizar la distancia en una escala lineal o encontrar la n más grande con $n! < x$, o la n más pequeña con $n! > x$ dependiendo de su aplicación.

Trabajando con Exponenciales y Logaritmos Supongamos que desea evaluar la función[54]

$$u(z) = \frac{e^z - 1 - z}{z^2}$$

para valores pequeños de z, digamos $z = 10^{-8}$.

El código Python

[54] Este ejemplo proviene de Lloyd N. Trefethen and J. A. C. Weideman. The Exponentially Convergent Trapezoid Rule. SIAM Review. Vol. 56, No. 3. pp. 385-458.

```
import numpy as np
def f(z): return (np.exp(z) - 1 - z)/z**2
print(f(1e-8))
```

imprime:

```
-0.607747099184471.
```

Ahora suponga que sospecha dificultades numéricas y calcula su resultado con 50 decimales usando *bc -l*[55] usando:

```
scale=50
z=10^-8
(e(z)-1-z)/z^2
```

imprime:

```
.50000000166666667083333334166666660000000000000000
```

Esto sugiere que el cálculo original era completamente erróneo. ¿Qué está sucediendo?

Para z pequeña,

$$e^z \approx 1 + z$$

y así perdemos precisión al evaluar directamente el numerador en la definición de u (en nuestro ejemplo, perdimos toda precisión).

El teorema del valor medio del análisis complejo dice que el valor de una función analítica en un punto es igual al promedio continuo de los valores sobre un círculo centrado en ese punto. Si aproximamos este promedio tomando el promedio de 16 valores en un círculo de radio 1 alrededor de nuestro punto, obtenemos una precisión total. El código Python

```
def g(z):
    N = 16
    ws = z + np.exp(2j*np.pi*np.arange(N)/N)
    return sum(f(ws))/N
print(g(1e-8))
```

[55] El comando bc (Basic Calculator) de la línea de comandos de Linux/Unix, es una calculadora con grandes posibilidades de uso.

imprime:

0.5000000016666668 + 8.673617379884035e-19j

que se aparta del resultado calculado con *bc* en el decimosexto decimal. En un nivel alto, evitamos dificultades numéricas al promediar puntos alejados de la región difícil.

Logit inverso A continuación, veamos el cálculo $f(x) = e^x/(1 + e^x)$. (Los estadísticos llaman a esto la función "logit inverso" porque es el inverso de la función que ellos llaman la función "logit".) El enfoque más directo sería calcular $exp(x)/(1 + exp(x))$. Veamos dónde se puede romper eso.

```
double x = 1000;
double t = exp(x);
double y = t/(1.0 + t);
```

Imprimir y da -1. # *IND*. Esto se debe a que el cálculo de t se desbordó, produciendo un número mayor que el que cabía en un doble. Pero podemos calcular el valor de y fácilmente. Si t es tan grande que no podemos almacenarlo, entonces $1 + t$ es esencialmente lo mismo que t y la relación es muy cercana a 1. Esto sugiere que descubramos qué valores de x son tan grandes que $f(x)$ será igual 1 a la precisión de la máquina, luego solo devuelva 1 para esos valores de x para evitar la posibilidad de desbordamiento. Esta es nuestra siguiente regla: No calcules un resultado que puedas predecir con precisión.

El archivo de encabezado float.h tiene un $DBL\_EPSILON$ constante, que es la precisión doble más pequeña que podemos agregar a 1 sin recuperar 1. Un poco de álgebra muestra que si x es más grande que $-log(DBL\_EPSILON)$, entonces $f(x)$ será igual a 1 a la precisión de la máquina. Así que aquí hay un fragmento de código para calcular $f(x)$ para valores grandes de x:

```
const double x_max = -log(DBL_EPSILON);
if (x > x_max) return 1.0;
```

El código provisto en esta sección calcula siete funciones que aparecen en las estadísticas. Cada uno evita problemas de desbordamiento, subdesbordamiento o pérdida de precisión que podrían ocurrir para grandes argumentos negativos, grandes argumentos positivos o argumentos cercanos a cero:

- LogOnePlusX calcúlese como $log(1 + x)$ como en ejemplo antes visto
- ExpMinusOne calcúlese como $e^x - 1$
- Logit calcúlese como $log(x/(1 - x))$
- LogitInverse calcúlese como $e^x/(1 + e^x)$ como se discutió en el ejemplo último
- LogLogitInverse calcúlese como $log(e^x/(1 + e^x))$
- LogitInverseDifference calcúlese como $LogitInverse(x) - LogitInverse(y)$
- LogOnePlusExpX calcúlese como $log(1 + exp(x))$
- ComplementaryLogLog calcúlese como $log(-log(1 - x))$
- ComplementaryLogLogInverse calcúlese como $1.0 - exp(-exp(x))$

Las soluciones presentadas aquí parecen innecesarias o incluso incorrectas al principio. Si este tipo de código no está bien comentado, alguien lo verá y lo "simplificará" incorrectamente. Estarán orgullosos de todo el desorden innecesario que eliminaron. Y si no prueban valores extremos, su nuevo código parecerá funcionar correctamente. Las respuestas incorrectas y los NaN solo aparecerán más tarde.

Log (1 + x) Ahora veamos el ejemplo del cálculo de $log(x+1)$. Considere el siguiente código:

```
double x = 1e-16;
double y = log(1 + x)/x;
```

En este código $y = 0$, aunque el valor correcto sea igual a 1 para la precisión de la máquina.

¿Qué salió mal? los números de doble precisión tienen una precisión de aproximadamente 15 decimales, por lo que $1 + x$ equivale a 1 para la precisión de la máquina. El registro de 1 es cero, por lo que y se establece en cero. Pero para valores pequeños de x, $log(1 + x)$ es aproximadamente x, por lo que $log(1 + x)/x$ es aproximadamente 1. Eso significa que el código anterior para calcular $log(1 + x)/x$ devuelve un resultado con 100% de error relativo. Si x no es tan pequeño que $1 + x$ es igual a 1 en la máquina, aún podemos tener problemas. Si x es moderadamente pequeño, los bits en x no se pierden totalmente al calcular $1 + x$, pero algunos sí. Cuanto más se acerca x a 0, más bits se pierden. Podemos usar la siguiente regla: Utilice aproximaciones analíticas para evitar la pérdida de precisión.

La forma favorita de aproximación de los analistas numéricos es la serie de potencia. ¡La serie de potencia para

$$log(1 + x) = x + x^2/2! + x^3/3! + ...$$

Para valores pequeños de x, simplemente devolver x para $log(1 + x)$ es una mejora. Esto funcionaría bien para los valores más pequeños de x, pero para algunos valores no tan pequeños, esto no será lo suficientemente preciso, pero tampoco lo hará directamente el cálculo del $log(1 + x)$.

La Precisión Arbitraria no es una Panacea Tener acceso a aritmética de precisión arbitraria no necesariamente hace que desaparezcan las dificultades de cálculo numérico. Aún necesitas saber lo que estás haciendo. Antes de ampliar la precisión, hay que saber hasta dónde ampliarla. Si no lo amplía lo suficiente, sus respuestas no serán lo suficientemente precisas. Si lo extiendes demasiado, desperdicias recursos. Tal vez esté bien desperdiciar recursos, por lo que extiende la precisión más de lo necesario. ¿Pero hasta qué punto es más de lo necesario?

Como ejemplo, considere el problema de encontrar valores de n tales que $tan(n) > n$, uno de los valores es:

$$k = 1428599129020608582548671.$$

Verifiquemos que $tan(k) > k$. Usaremos *bc* porque admite precisión arbitraria. Nuestra k es un número de 25 dígitos, así que digamos a *bc* que queremos trabajar con 30 decimales para tener un poco de espacio[56]. *bc* no

[56] *bc* automáticamente le otorga un poco más de espacio del que solicita, pero le pediremos aún más

tiene una función tangente, pero sí la tiene *s()* para seno y *c()* para coseno, por lo que calcularemos la tangente como seno sobre coseno.

```
$ bc -l
escala = 30
k = 14285991290206085825486 71
s(k)/c(k) - k
```

Esperamos que esto devuelva un valor positivo, pero en su lugar devuelve

-1428599362980017942210629.31...

Entonces, ¿está mal la hipótesis? ¿O es *bc* ignorar nuestra solicitud? Resulta que ninguna de las dos cosas es cierta.

No se puede calcular directamente la tangente de un número grande. Utiliza la reducción de rango para reducirlo al problema de calcular la tangente de un ángulo pequeño donde funcionarán sus algoritmos. Dado que la tangente tiene un período π, reducimos k mod π calculando $k - [k/\pi]\,\pi$. Es decir, restamos tantos múltiplos de π como podamos hasta obtener un número entre 0 y π. Volviendo a *bc*, calculamos

```
pi = 4*a(1)
k/pi
```

esto regresa

45473722616081240284265 6.50000050703322137044486 6152761

y entonces calculamos la tangente de

```
t = 0,50000050703322137044486 6152761*pi
= 1,57079791968674000258827 0178941
```

Como t es ligeramente mayor que $\pi/2$, la tangente será negativa. No podemos tener $tan(t)$ mayor que k porque $tan(t)$ ni siquiera es mayor que 0. Entonces, ¿dónde se estropearon las cosas?.

El cálculo de π tuvo una precisión de 30 cifras significativas y el cálculo de k/π tuvo una precisión de 30 cifras significativas, dado nuestro valor de π. Hasta ahora ha funcionado según lo prometido.

El valor calculado de k/π tiene una precisión de 29 cifras significativas, 23 antes del decimal y 6 después. Entonces, cuando tomamos la parte fraccionaria, solo tenemos seis cifras significativas y eso no es suficiente. Ahí es donde las cosas van mal. Obtenemos un valor $[k/\pi]$ que es mayor que 0.5 en el séptimo decimal, mientras que el valor exacto es menor que 0.5 en el vigésimo quinto decimal. Necesitábamos $25-6 = 19$ cifras más significativas.

Ésta es la principal dificultad del cálculo en punto flotante: restar números casi iguales pierde precisión. Si dos números coinciden en m decimales, puedes esperar perder m cifras significativas en la resta. El error en la resta será pequeño en relación con los datos de entrada, pero no pequeño en relación con el resultado.

Observe que calculamos k/π hasta 29 cifras significativas y, dado ese resultado, calculamos la parte fraccionaria exactamente . Calculamos con precisión $[k/\pi]\,\pi$, pero perdimos precisión cuando calculamos y restamos ese valor de k.

Nuestro error al calcular $k-[k/\pi]\,\pi$ fue pequeño en relación con k, pero no en relación con el resultado final. Nuestra k es del orden de 10^{25} y el error en nuestra resta es del orden de 10^{-7}, pero el resultado es del orden de 1. No hay ningún error en *bc*. Llevó a cabo todos los cálculos con la precisión anunciada, pero no tuvo suficiente precisión de trabajo para producir el resultado que necesitábamos.

6.3 Aritmética de Baja Precisión

La popularidad de la aritmética de baja precisión para el cómputo de alto rendimiento se ha disparado desde el lanzamiento en 2017 de la GPU Nvidia Volta. Los núcleos tensores de media precisión de Volta ofrecieron una enorme ganancia de rendimiento 16 veces mayor que la doble precisión para operaciones clave. Y el rendimiento del Hardware está mejorando aún más: el FP16 con núcleo tensor Nvidia H100 es 58 veces más rápido que el FP64 estándar.

Esta sorprendente aceleración ciertamente llama la atención. Sin embargo, en el cálculo científico, la aritmética de baja precisión suele considerarse insegura para los códigos de modelado y simulación. De hecho, a veces se puede aprovechar una precisión más baja, comúnmente en una configuración de "precisión mixta" en la que sólo partes del cálculo se realizan con baja precisión. Sin embargo, en general, cualquier precisión menor que el doble se considera inadecuada para modelar fenómenos físicos complejos

con fidelidad.

En respuesta, los desarrolladores han creado herramientas para medir la seguridad de la aritmética de precisión reducida en códigos de aplicación. Algunas herramientas pueden incluso identificar qué variables o matrices se pueden reducir de forma segura a una precisión menor sin perder precisión en el resultado final. Sin embargo, el uso de estas herramientas a ciegas, sin el respaldo de algún tipo de proceso de razonamiento, puede resultar peligroso.

Un ejemplo ilustrará esto:

> El método del gradiente conjugado para la resolución y optimización de sistemas lineales y el método de Lanczos, estrechamente relacionado, para la resolución de problemas de valores propios mostraron una gran promesa tras su invención a principios de los años cincuenta. Sin embargo, se consideraban inseguros debido a errores de redondeo catastróficos en la aritmética de punto flotante, que son aún más pronunciados a medida que se reduce la precisión del punto flotante.
>
> No obstante, Chris Paige demostró en su trabajo pionero en la década de 1970 que el error de redondeo, aunque sustancial, no excluye la utilidad de los métodos cuando se utilizan correctamente. El método del gradiente conjugado se ha convertido en un pilar del cómputo científico.

Teniendo en cuenta que ninguna herramienta podría llegar a este hallazgo sin un cuidadoso análisis matemático de los métodos. Una herramienta detectaría inexactitudes en el cálculo pero no podría certificar que estos errores no puedan perjudicar el resultado final.

Algunos podrían proponer en cambio un enfoque puramente basado en datos: simplemente pruebe con baja precisión en algunos casos de prueba; si funciona, utilice baja precisión en producción. Sin embargo, este enfoque está lleno de peligros: es posible que los casos de prueba no capturen todas las situaciones que podrían encontrarse en producción.

Por ejemplo, uno podría probar un código de aerodinámica sólo en regímenes de flujo suaves, pero las series de producción pueden encontrar flujos complejos con gradientes pronunciados, que la aritmética de baja precisión no puede modelar correctamente. Los artículos académicos que prueban métodos y herramientas de baja precisión deben evaluarse rigurosamente en escenarios desafiantes del mundo real como este.

Lamentablemente, los equipos de ciencia computacional frecuentemente no tienen tiempo para evaluar sus códigos para un uso potencial de aritmética de menor precisión. Las herramientas ciertamente podrían ayudar. Además, las bibliotecas que encapsulan métodos de precisión mixta pueden ofrecer beneficios a muchos usuarios. Una gran historia de éxito son los solucionadores lineales densos de precisión mixta, basados en el sólido trabajo teórico de Nick Highnam y sus colegas , que han llegado a bibliotecas como: Lu, Hao; Matheson, Michael; Wang, Feiyi; Joubert, Wayne; Ellis, Austin; Oles, Vladyslav. "OpenMxP-Opensource Mixed Precision Computing,".

Entonces la respuesta final es "depende". Cada nuevo caso debe examinarse cuidadosamente y tomar una decisión basada en alguna combinación de análisis y pruebas.

Sí bien, NVIDIA lidera el mercado de las GPU para inteligencia artificial (IA) con una cuota de mercado aproximada del 80%, pero no es en absoluto la única empresa que tiene en su porfolio chips de vanguardia para IA. La compañía californiana Cerebras posee, de hecho, los procesadores para este escenario de uso más complejos que existen. Su chip WSE-2, por ejemplo, aglutina nada menos que 2.6 billones de transistores contabilizados en la escala numérica larga y $850,000$ núcleos optimizados para IA.

Cerebras entrega a sus clientes estos procesadores integrados en una plataforma para IA conocida como CS-2, y precisamente uno de ellos es la compañía de Emiratos Árabes G42. Esta última está construyendo seis superordenadores para IA capaces de superar la barrera de la exaescala que aglutinan una gran cantidad de sistemas CS-2. Y según la CIA algunas de estas máquinas irán a parar a las grandes tecnológicas chinas. No obstante, esto no es todo. Y es que Cerebras dió a conocer en 2024 un procesador para IA aún más potente que su WSE-2.

El procesador WSE-3 Cerebras ya tiene listo su procesador WSE-3 (Wafer Scale Engine 3), un producto que, como podemos intuir, está llamado a suceder al también ambicioso WSE-2.

Ambos procesadores se fabrican a partir de una oblea completa de silicio, lo que permite a Cerebras integrar muchos más bloques funcionales y núcleos en la lógica que una GPU convencional como las que fabrican NVIDIA, AMD o Huawei.

Y es que aglutina 4 billones de transistores, tiene una superficie de $46,225$ mm^2, integra nada menos que $900,000$ núcleos optimizados para IA y tiene

una potencia de cálculo, según Cerebras, de 125 petaflops.

Según Cerebras su procesador WSE-3 es el doble de potente que el WSE-2. De hecho, de acuerdo con las especificaciones que ha publicado rinde como 62 GPU H100 de NVIDIA trabajando al unísono, y no debemos pasar por alto que este procesador de la compañía liderada por Jensen Huang es el más potente que tiene hasta que se produzca el lanzamiento de la GPU H200.

Sea como sea Cerebras entrega sus procesadores WSE-3 integrados en un superordenador conocido como CS-3 que es capaz de entrenar grandes modelos de IA con hasta 24 billones de parámetros. El mapa de memoria externa de este superordenador oscila entre 1.5 TB y 1.2 PB, un espacio de almacenamiento descomunal que permite almacenar modelos de lenguaje masivos en un único espacio lógico.

Según informan, el chip WSE-3 optimizado para la IA es capaz de entrenar hasta $24,000$ millones de parámetros, lo que también equivaldría a un rendimiento máximo de IA de 125 petaflops.

7 Apéndice A: Método de Diferencias Finitas

En la búsqueda de una descripción cualitativa de un determinado fenómeno, por lo general se plantea un sistema de ecuaciones diferenciales ordinarias o parciales, válidas para cierto dominio y donde se imponen sobre este, una serie de condiciones en la frontera y en su caso de condiciones iniciales. Después de esto, el modelo matemático se considera completo, y es aquí donde la implementación computacional entra a ayudar en la solución del problema, ya que sólo es posible resolver de forma exacta problemas simples y en fronteras geométricas triviales con los métodos matemáticos que disponemos.

En esta sección consideraremos como implementar la solución computacional del Método de Diferencias Finitas, este método es de carácter general que permite la resolución aproximada de ecuaciones diferenciales en derivadas parciales definidas en un dominio finito. Es de una gran sencillez conceptual y constituye un procedimiento muy adecuado para la resolución de una ecuación en una, dos o tres dimensiones.

El método consiste en una aproximación de las derivadas parciales por expresiones algebraicas con los valores de la variable dependiente en un número finito de puntos seleccionados en el dominio[57]. Como resultado de la aproximación, la ecuación diferencial parcial que describe el problema es reemplazada por un número finito de ecuaciones algebraicas, en términos de los valores de la variable dependiente en los puntos seleccionados. El valor de los puntos seleccionados se convierten en las incógnitas. La solución del sistema de ecuaciones algebraico permite obtener la solución aproximada en cada punto seleccionado de la malla.

7.1 Método de Diferencias Finitas en una Dimensión

Sin pérdida de generalidad, consideremos la ecuación diferencial parcial

$$\left(p\left(x\right)u'\left(x\right)\right)' + q\left(x\right)u'\left(x\right) - r\left(x\right)u\left(x\right) = f\left(x\right) \tag{7.1}$$

$$\text{en } a \leq x \leq b \qquad \text{donde: } u\left(a\right) = u_\alpha \text{ y } u\left(b\right) = u_\beta$$

[57] La técnica fundamental para los cálculos numéricos en diferencias finitas se basa en aproximar $f(x)$ mediante un polinomio cerca de $x = x_0$. Una de las aproximaciones clásicas es mediante la serie de Taylor, la cual también nos permite, aparte de la aproximación de las derivadas, el cálculo del error en la aproximación mediante su fórmula del residuo.

con condiciones de frontera Dirichlet, Neumann o Robin. Para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, debemos de:

1. Generar una malla del dominio, i.e. una malla es un conjunto finito de puntos en los cuales buscaremos la solución aproximada a la ecuación diferencial parcial.

2. Sustituir las derivadas correspondientes con alguna de las formulas de diferencias finitas centradas (véase 8.1 y 8.3), en cada punto donde la solución es desconocida para obtener un sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$.

3. Resolver el sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ (véase 3), y así obtener la solución aproximada en cada punto de la malla.

7.1.1 Problema con Condiciones de Frontera Dirichlet

Consideremos un caso particular de la Ec.(7.1) definido por la ecuación

$$u''(x) = f(x), \qquad 0 \leq x \leq 1, \qquad u(0) = u_\alpha, \qquad u(1) = u_\beta \tag{7.2}$$

con condiciones de frontera Dirichlet. Para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, debemos de:

1. Generar una malla homogénea del dominio[58]

$$x_i = ih, \qquad i = 0, 1, ..., n, \qquad h = \frac{1}{n} = \Delta x \tag{7.3}$$

2. Sustituir la derivada con la Ec.(8.24) en cada punto donde la solución es desconocida para obtener un sistema algebraico de ecuaciones. Así, en cada punto x_i de la malla aproximamos la ecuación diferencial por[59]

$$u''(x_i) \approx \frac{u(x_i - h) - 2u(x) + u(x_i + h)}{h^2} \tag{7.4}$$

[58] En el caso de que la malla no sea homogénea, es necesario incluir en la derivada a que h se refiere, por ejemplo en cada punto i, tenemos la h_{i-} (por la izquierda) y la h_{i+} (por la derecha), i.e. $u''(x_i) \approx \frac{u(x_i - h_{i-}) - 2u(x) + u(x_i + h_{i+})}{(h_{i-})(h_{i+})}$.

[59] Notemos que en cada punto de la malla, la aproximación por diferencias finitas supone la solución de tres puntos de la malla x_{i-1}, x_i y x_{i+1}. El conjunto de estos tres puntos de la malla es comúnmente llamado el esténcil de diferencias finitas en una dimensión.

o en su forma simplificada

$$u''(x_i) \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} \tag{7.5}$$

definiendo la solución aproximada de $u(x)$ en x_i como u_i, entonces se genera el siguiente sistema lineal de ecuaciones

$$\begin{aligned}
\frac{u_\alpha - 2u_1 + u_2}{h^2} &= f(x_1) \\
\frac{u_1 - 2u_2 + u_3}{h^2} &= f(x_2) \\
&\vdots \\
\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} &= f(x_i) \\
&\vdots \\
\frac{u_{n-3} - 2u_{n-2} + u_{n-1}}{h^2} &= f(x_{n-2}) \\
\frac{u_{n-2} - 2u_{n-1} + u_\beta}{h^2} &= f(x_{n-1}).
\end{aligned} \tag{7.6}$$

Este sistema de ecuaciones se puede escribir como la matriz $\underline{\underline{A}}$ y los vectores $\underline{u}$ y $\underline{f}$ de la forma

$$\begin{bmatrix}
-\frac{2}{h^2} & \frac{1}{h^2} & & & & \\
\frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} & & & \\
 & \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} & & \\
 & & \ddots & \ddots & \ddots & \\
 & & & \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} \\
 & & & & \frac{1}{h^2} & -\frac{2}{h^2}
\end{bmatrix}
\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} =
\begin{bmatrix} f(x_1) - \frac{u_\alpha}{h^2} \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{n-2}) \\ f(x_{n-1}) - \frac{u_\beta}{h^2} \end{bmatrix}.$$

en este caso, podemos factorizar $1/h^2$ del sistema lineal $\underline{\underline{A}}\underline{u} = \underline{f}$, quedando como

$$\frac{1}{h^2}\begin{bmatrix}
-2 & 1 & & & & \\
1 & -2 & 1 & & & \\
 & 1 & -2 & 1 & & \\
 & & \ddots & \ddots & \ddots & \\
 & & & 1 & -2 & 1 \\
 & & & & 1 & -2
\end{bmatrix}
\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} =
\begin{bmatrix} f(x_1) - \frac{u_\alpha}{h^2} \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{n-2}) \\ f(x_{n-1}) - \frac{u_\beta}{h^2} \end{bmatrix}.$$

esta última forma de expresar el sistema lineal algebraico asociado es preferible para evitar problemas numéricos al momento de resolver el sistema lineal por métodos iterativos (véase 3.2) principalmente cuando $h \longrightarrow 0$.

3. Resolver el sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ (véase 3), obtenemos la solución aproximada en cada punto interior de la malla. La solución completa al problema la obtenemos al formar el vector

$$\left[\begin{array}{ccccccccc} u_\alpha & u_1 & u_2 & u_3 & \cdots & u_{n-2} & u_{n-1} & u_\beta \end{array} \right].$$

Uno de los paquetes más conocidos y usados para el cálculo numérico es MatLab[60] el cual tiene un alto costo monetario para los usuarios. Por ello, una opción es usar alternativas desarrolladas usando licencia de código abierto, un par de estos paquetes son: GNU OCTAVE[61] y SCILAB[62].

Octave corre gran parte del código desarrollado para MatLab sin requerir cambio alguno, en cuanto a SCILAB es requerido hacer algunos ajustes en la codificación y ejecución. En los siguientes ejemplos[63] se mostrará cómo implementar la solución computacional. En la sección (5) se mostrará la implementación en diversos paquetes y lenguajes de programación.

[60] MatLab es un programa comercial para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería [https://www.mathworks.com/products/matlab-home.html].

[61] GNU OCTAVE es un programa open source para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería [http://www.gnu.org/software/octave].

[62] SCILAB es un programa open source para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería [http://www.scilab.org].

[63] Los ejemplos que se muestran en el presente texto se pueden descargar de la página WEB:

http://132.248.182.159/acl/MDF/

o desde GitHub (https://github.com/antoniocarrillo69/MDF) mediante

git clone git://github.com/antoniocarrillo69/MDF.git

Ejemplo 20 *Sea*

$$-u''(x) = -\pi^2 \cos(\pi x), \qquad xi \leq x \leq xf, \qquad u(xi) = vi, \qquad u(xf) = vf$$

El programa queda implementado en OCTAVE (MatLab) como:

```
function [A,b,x] = fdm1d(xi,xf,vi,vf,n)
  N=n-2; % Nodos interiores
  h=(xf-xi)/(n-1); % Incremento en la malla
  %A = sparse(N,N); % Matriz SPARSE
  A=zeros(N,N); % Matriz A
  b=zeros(N,1); % Vector b
  R=1/(h^2);
  P=-2/(h^2);
  Q=1/(h^2);
  % Primer renglon de la matriz A y vector b
  A(1,1)=P;
  A(1,2)=Q;
  b(1)=LadoDerecho(xi)-vi*R;
  % Renglones intermedios de la matriz A y vector b
  for i=2:N-1
    A(i,i-1)=R;
    A(i,i)=P;
    A(i,i+1)=Q;
    b(i)=LadoDerecho(xi+h*(i-1));
  end
  % Renglon final de la matriz A y vector b
  A(N,N-1)=R;
  A(N,N)=P;
  b(N)=LadoDerecho(xi+h*N)-vf*Q;
  % Resuleve el sistema lineal Ax=b
  x=inv(A)*b;
  % Prepara la graficacion
  xx=zeros(n,1);
  zz=zeros(n,1);
  for i=1:n
    xx(i)=xi+h*(i-1);
    zz(i)=SolucionAnalitica(xx(i));
  end
```

```
    yy=zeros(n,1);
    yy(1)=vi; % Condicion inicial
    for i=1:N
       yy(i+1)=x(i);
    end
    yy(n)=vf; % Condicion inicial
    % Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
    plot(xx,[yy,zz]);
  endfunction
  function y=LadoDerecho(x)
    y=-pi*pi*cos(pi*x);
  endfunction
  function y=SolucionAnalitica(x)
    y=cos(pi*x);
  endfunction
```

Si el programa lo grabamos en el directorio de trabajo con el nombre **fdm1d.m**, entonces se puede ejecutar en la consola de OCTAVE (MatLab) mediante

$$[A, b, x] = fdm1d(-1, 2, -1, 1, 30);$$

donde es necesario indicar el inicio (-1) y fin (2) del dominio, el valor de la condición de frontera de inicio (-1) y fin (1), además de el número de nodos (30) en la partición. La ejecución genera la gráfica de la solución calculada por el método de diferencias finitas y la solución analítica en los mismos puntos de la malla, además devuelve la matriz[64] y los vectores A, b, x generados por la función.

En OCTAVE (MatLab) es posible introducir funciones para que pasen cómo parámetros a otra función, de tal forma que se puede definir un programa genérico de diferencias finitas en una dimensión con condiciones de frontera Dirichlet en el cual se especifiquen los parámetros de la ecuación en la línea de comando de OCTAVE.

[64] En Octave (MatLab) se define a una matriz mediante A = zeros(N,N), este tipo de matriz no es adecuada para nuestros fines (véase capítulo 3). Para ahorrar espacio y acelerar los cálculos numéricos que se requieren para resolver el sistema lineal asociado usamos un tipo de matriz que no guarda valores innecesarios (ceros), esto se hace mediante la declaración de la matriz como A = sparse(N,N).

Usando este hecho, implementamos un programa para codificar el Método de Diferencias Finitas en una Dimensión para resolver el problema con condiciones Dirichlet

$$p(x)u'' + q(x)u' + r(x)u = f(x)$$

definido en el dominio $[xi, xf]$ y valores en la frontera de $u(xi) = vi$ y $u(xf) = vf$, además se le indica el tamaño de la partición n, si se graficara la solución indicando en $grf = 1$, con solución analítica $s(x)$ y si a esta se proporciona entonces $sws = 1$. Regresando la matriz y los vectores $\underline{\underline{A}}\underline{u} = \underline{b}$ del sistema lineal generado así como los puntos del dominio y los valores de la solución en dichos puntos x, V, para cada problema que deseemos resolver.

Ejemplo 21 *El programa queda implementado en OCTAVE (MatLab) como:*

```
function [error,A,b,u,x,V] = fdm1d_DD(p,q,r,f,xi,xf,vi,vf,n,grf,s,sws)
    if n < 3
        return
    end
    % Numero de incognitas del problema
    tm = n -2;
    % Declaracion de la matriz y vectores de trabajo
    %A = sparse(tm,tm);
    A = zeros(tm,tm); % Matriz de carga
    b = zeros(tm,1); % Vector de carga
    u = zeros(tm,1); % Vector de solucion
    x = zeros(n,1); % Vector de coordenadas de la particion
    V = zeros(n,1) ; % Vector solucion
    h = (xf-xi)/(n-1);
    h1 = h*h;
    % Llenado de los puntos de la malla
    for i = 1: n,
        x(i) = xi + (i-1)*h;
    end
    % Llenado de la matriz y vector
    b(1) = f(xi) - p(xi)*(vi/h1);
    A(1,2) = p(x(1))/h1 - q(x(1))/(2.0*h);
    A(1,1) = ((-2.0 * p(x(1))) / h1) + r(x(1));
    for i=2:tm-1,
        A(i,i-1) = p(x(i))/h1 - q(x(i))/(2.0*h);
```

```
    A(i,i) = ((-2.0 * p(x(i))) / h1) + r(x(i));
    A(i,i+1) = p(x(i))/h1 + q(x(i))/(2.0*h);
    b(i) = f(x(i));
end
A(tm,tm-1) = p(x(tm))/h1 - q(x(tm))/(2.0*h);
A(tm,tm) = ((-2.0 * p(x(tm))) / h1) + r(x(tm));
b(tm) = f(x(tm+1))-p(x(tm+1))*(vf/h1);
% Soluciona el sistema
u = inv(A)*b;
% Copia la solucion obtenida del sistema lineal al vector solucion
V(1) = vi;
for i=1:tm,
    V(i+1) = u(i);
end
V(n) = vf;
% Encuentra el error en norma infinita usando particion par=10
error = 0;
if sws == 1
    par = 10;
    for i = 1: n-1,
        inc = (x(i+1)-x(i))/par;
        for j = 1:par+1,
            px = x(i)+inc*(j-1);
            e = abs(s(px)-l(px,x(i),V(i),x(i+1),V(i+1)));
            if e > error
                error = e;
            end
        end
    end
end
% Revisa si se graficara
if grf == 1
    if sws == 1
        % Calcula la solucion analitica en la particion de calculo
        ua = zeros(n,1);
        for i = 1: n,
            ua(i) = s(x(i));
        end
```

```
            end
            % Graficar la solucion numerica
            plot(x,V,'o');
            hold
            % Graficar la solucion analitica en una particion tamano xPart
            if sws == 1
                xPart = 1000;
                h = (xf-xi)/(xPart-1);
                xx = zeros(xPart,1);
                xa = zeros(xPart,1);
                for i = 1: xPart,
                    xx(i) = xi + (i-1)*h;
                    xa(i) = s(xx(i));
                end
                plot(xx,xa);
                % Grafica el error
                figure(2);
                plot(x,V-ua);
            end
        end
    end
    % Evalua el punto x en la recta dada por los puntos (x1,y1) y (x2,y2), se
usa para el calculo de la norma infinito
    function y = l(x,x1,y1,x2,y2)
        y = y1+((y2-y1)/(x2-x1))*(x-x1);
    end
```

De esta forma podemos implementar diversos problemas y ver su solución

Ejemplo 22 *Sea*

$$-u''(x) = -\pi^2 cos(\pi x), \qquad -1 \leq x \leq 2, \qquad u(-1) = -1, \qquad u(2) = 1$$

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:

```
p=@(x) 1;
q=@(x) 0;
r=@(x) 0;
```

*f=@(x) -pi*pi*cos(pi*x);*
*s=@(x) cos(pi*x);*
[error,A,b,u,x,V] = fdm1d_DD(p,q,r,f,-1,2,-1,1,11,1,s,1);

Otro ejemplo:

Ejemplo 23 *Sea*

$$u''(x) + u = -\pi^2 cos(\pi x), \qquad 0 \leq x \leq 1, \qquad u(0) = 1, \qquad u(1) = -1$$

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:
p=@(x) 1;
q=@(x) 0;
r=@(x) 0;
*f=@(x) -pi*pi*cos(pi*x);*
*s=@(x) cos(pi*x);*
[error,A,b,u,x,V] = fdm1d_DD(p,q,r,f,0,1,1,-1,40,1,s,1);

Ahora veamos un ejemplo con un parámetro adicional (velocidad):

Ejemplo 24 *Sea*

$$-u''(x) + v(x)u = 0, \qquad 0 \leq x \leq 1, \qquad u(0) = 0, \qquad u(1) = 1$$

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:
v=@(x) 1.0; % velocidad
p=@(x) -1.0;
q=@(x) v(x);
r=@(x) 0;
f=@(x) 0;
*s=@(x) (exp(v(x)*x)-1.0)/(exp(v(x))-1.0);*
[error,A,b,u,x,V] = fdm1d_DD(p,q,r,f,0,1,0,1,11,1,s,1);

En éste caso, la velocidad es $v = 1.0$ y nuestro programa lo puede resolver sin complicación alguna. Si aumentamos la velocidad a $v = 50.0$ y volvemos a correr el programa haciendo estos cambios, entonces:

Ejemplo 25 *v=@(x) 50.0;*
[error,A,b,u,x,V] = fdm1d_DN(p,q,r,f,0,1,-1,0,-1,1,11,1,s,1);

Aún el programa lo resuelve bien, pero si ahora subimos la velocidad a $v = 100.0$ y corremos nuevamente el programa:

Ejemplo 26 *v=@(x) 100.0;*
[error,A,b,u,x,V] = fdm1d_DN(p,q,r,f,0,1,-1,0,-1,1,11,1,s,1);

Entonces tendremos que la solución oscila en torno al cero. Para corregir esto, algunas opciones son: aumentar el número de nodos en la malla de discretización, usar una malla no homogénea refinada adecuadamente para tener un mayor número de nodos en donde sea requerido, usar fórmulas más precisas para las derivadas o mejor aún, usar diferentes esquemas tipo Upwind, Scharfetter-Gummel y Difusión Artificial para implementar el Método de Diferencias Finitas, como se muestra a continuación.

Número de Péclet Para el problema general dado por la Ec.(7.1)

$$\left(p\left(x\right)u'\left(x\right)\right)' + q\left(x\right)u'\left(x\right) - r\left(x\right)u\left(x\right) = f\left(x\right) \tag{7.7}$$

$$\text{en } a \leq x \leq b \qquad \text{donde: } u\left(a\right) = u_\alpha \text{ y } u\left(b\right) = u_\beta$$

el término $q\left(x\right)u'\left(x\right)$ algunas veces es llamado el término advectivo[65] si u es la velocidad y puede ocasionar inestabilidad numérica en el Método de Diferencias Finitas como se mostró en el ejemplo anterior Ej.(26), para evitar dichas inestabilidades existen diversas técnicas de discretización mediante el análisis de la solución considerando el Número de Péclet (local y global) y su implementación en el Método de Diferencias Finitas (véase [**?**]) mediante el esquema Upwind, Scharfetter-Gummel y Difusión Artificial, entre otros, para ello consideremos:

- Si las funciones $p(x), q(x)$ y $r(x)$ son constantes, el esquema de diferencias finitas centradas para todas las derivadas, este esta dado por el estencil

$$p_i\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} + q_i\frac{u_{i+1} - u_{i-1}}{2h} - r_iu_i = f_i \qquad i = 1, 2, ..., n. \tag{7.8}$$

[65] Cuando la advección es fuerte, esto es cuando $|q\left(x\right)|$ es grande, la ecuación se comporta como si fuera una ecuación de onda.

donde la ventaja de esta discretización, es que el método es de segundo orden de exactitud. La desventaja es que los coeficientes de la matriz generada pueden no ser diagonal dominante si $r(x) > 0$ y $p(x) > 0$. Cuando la advección $|p(x)|$ es grande, la ecuación se comporta como la ecuación de onda.

- Tomando las funciones $p(x,t), q(x,t)$ y $r(x,t)$ más generales posibles, es necesario hacer una discretización que garantice que el método es de segundo orden de exactitud, esto se logra mediante la siguiente discretización para $(p(x,t)\,u'(x,t))'$ mediante

$$\begin{aligned}\frac{\partial}{\partial x}\left(p\frac{\partial u}{\partial x}\right)(x,t) \simeq & \left[p\left(x+\frac{\Delta x}{2},t\right)\frac{u(x+\Delta x,t)-u(x,t)}{\Delta x}\right. \\ & \left.-\,p\left(x-\frac{\Delta x}{2},t\right)\frac{u(x,t)-u(x-\Delta x,t)}{\Delta x}\right]/\Delta x\end{aligned} \tag{7.9}$$

entonces se tiene que

$$\frac{p(u_{i+1},t)\frac{u_{i+1}+u_i}{h}-p(u_{i-1,t})\frac{u_i-u_{i-1}}{h}}{h}+q_i\frac{u_{i+1}-u_{i-1}}{2h}-r_iu_i=f_i \tag{7.10}$$

para $i=1,2,...,n$, (véase [**?**] pág. 78 y 79).

- El esquema mixto, en donde se usa el esquema de diferencias finitas centradas para el término de difusión y el esquema *upwind* para el término de advección

$$\begin{aligned} p_i\frac{u_{i-1}-2u_i+u_{i+1}}{h^2}+q_i\frac{u_{i+1}-u_{i-1}}{h}-r_iu_i &= f_i, \text{ si } q_i\geq 0 \\ p_i\frac{u_{i-1}-2u_i+u_{i+1}}{h^2}+q_i\frac{u_i-u_{i-1}}{h}-r_iu_i &= f_i, \text{ si } q_i<0 \end{aligned} \tag{7.11}$$

el propósito es incrementar el dominio de la diagonal. Este esquema es de orden uno de exactitud y su uso es altamente recomendable si $|q(x)| \sim 1/h$, en caso de no usarse, se observará que la solución numérica oscila alrededor del cero.

7.1.2 Problema con Condiciones de Frontera Neumann

Consideremos el problema

$$u''(x)=f(x), \qquad 0\leq x\leq 1 \tag{7.12}$$

con condiciones de frontera Neumann

$$\frac{du}{dx} = cte_1 \text{ en } u(0) \text{ y } \frac{du}{dx} = cte_2 \text{ en } u(1)$$

para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, primero debemos discretizar las condiciones de frontera, una manera sería usar para la primera condición de frontera una aproximación usando diferencias progresivas Ec.(8.5)

$$\left.\frac{du}{dx}\right|_{x_i} = \frac{u(x_i + h) - u(x_i)}{h}$$

quedando

$$\frac{u_1 - u_0}{h} = cte_1 \tag{7.13}$$

para la segunda condición de frontera una aproximación usando diferencias regresivas Ec.(8.10)

$$\left.\frac{du}{dx}\right|_{x_i} = \frac{u(x_i) - u(x_i - h)}{h}$$

quedando

$$\frac{u_n - u_{n-1}}{h} = cte_2 \tag{7.14}$$

pero el orden de aproximación no sería el adecuado pues estamos aproximando el dominio con diferencias centradas con un error local de truncamiento de segundo orden $O_c(\Delta x^2)$, en lugar de ello usaremos diferencias centradas Ec.(8.15) para tener todo el dominio con el mismo error local de truncamiento.

Para usar diferencias centradas Ec.(8.15)

$$\left.\frac{du}{dx}\right|_{x_i} = \frac{u(x_i + h) - u(x_i - h)}{2h}$$

en el primer nodo necesitamos introducir un punto de la malla ficticio $x_{-1} = (x_0 - \Delta x)$ con un valor asociado a u_{-1}, entonces

$$\frac{u_1 - u_{-1}}{2h} = cte_1 \tag{7.15}$$

así también, en el último nodo necesitamos introducir un punto de la malla ficticio $x_{n+1} = (x_n + \Delta x)$ con un valor asociado a u_{n+1}, obteniendo

$$\frac{u_{n+1} - u_{n-1}}{2h} = cte_2. \tag{7.16}$$

Estos valores no tienen significado físico alguno, dado que esos puntos se encuentran fuera del dominio del problema. Entonces debemos de:

1. Generar una malla homogénea del dominio

$$x_i = ih, \qquad i = 0, 1, ..., n, \qquad h = \frac{1}{n} = \Delta x. \tag{7.17}$$

2. Sustituir la derivada con la Ec.(8.24)[66] en cada punto donde la solución es desconocida para obtener un sistema algebraico de ecuaciones. Así, en cada punto x_i de la malla aproximamos la ecuación diferencial por

$$u''(x_i) \approx \frac{u(x_i - h) - 2u(x) + u(x_i + h)}{h^2} \tag{7.18}$$

definiendo la solución aproximada de $u(x)$ en x_i como u_i como la solución del siguiente sistema lineal de ecuaciones

$$\begin{aligned}
\frac{u_1 - u_{-1}}{2h} &= cte_1 \\
\frac{u_0 - 2u_1 + u_2}{h^2} &= f(x_1) \\
&\vdots \\
\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} &= f(x_i) \\
&\vdots \\
\frac{u_{n-2} - 2u_{n-1} + u_n}{h^2} &= f(x_{n-1}) \\
\frac{u_{n+1} - u_{n-1}}{2h} &= cte_2.
\end{aligned} \tag{7.19}$$

3. Resolver el sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ (véase 3), obtenemos la solución aproximada en cada punto de la malla.

[66] Para mantener la estabilidad es necesario tomar en cuenta las distintas formas de discretización en el Método de Diferencias Finitas mediante el esquema Upwind, Scharfetter-Gummel y Difusión Artificial, entre otros (véase la sección: 7.1.1, Ecs. 7.8 a 7.11).

7.1.3 Problema con Condiciones de Frontera Robin

El método de un punto de la malla ficticio es usado para el manejo de las condiciones de frontera mixta, también conocidas como condiciones de frontera Robin. Sin pérdida de generalidad, supongamos que en $x = a$, tenemos

$$\alpha u'(a) + \beta u(b) = \gamma$$

donde $\alpha \neq 0$. Entonces usando el punto de la malla ficticio, tenemos que

$$\alpha \frac{u_1 - u_{-1}}{2h} + \beta u_n = \gamma$$

o

$$u_{-1} = u_1 + \frac{2\beta}{\alpha} u_n - \frac{2h\gamma}{\alpha}$$

introduciendo esto en términos de diferencias finitas centradas, en $x = x_0$, entonces se tiene que

$$\left(-\frac{2}{h^2} + \frac{2\beta}{\alpha h}\right) u_n + \frac{2}{h^2} u_1 = f_0 + \frac{2\gamma}{\alpha h}$$

o

$$\left(-\frac{1}{h^2} + \frac{\beta}{\alpha h}\right) u_n + \frac{1}{h^2} u_1 = \frac{f_0}{2} + \frac{\gamma}{\alpha h}$$

lo que genera coeficientes simétricos en la matriz.

Consideremos el problema

$$u''(x) = f(x), \qquad 0 \leq x \leq 1 \tag{7.20}$$

con condiciones de frontera Dirichlet y Neumann

$$u(0) = u_\alpha \qquad \text{y} \qquad \frac{du}{dx} = cte_1 \text{ en } u(1).$$

respectivamente. Para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, primero debemos expresar la condición de frontera Neumann mediante diferencias centradas Ec.(8.15)

$$\left.\frac{du}{dx}\right|_{x_i} = \frac{u(x_i + h) - u(x_i - h)}{2h}$$

en el último nodo necesitamos introducir un punto de la malla ficticio $x_{n+1} = (x_n + \Delta x)$ con un valor asociado a u_{n+1} quedando

$$\frac{u_{n+1} - u_{n-1}}{2h} = cte_2. \tag{7.21}$$

Este valor no tiene significado físico alguno, dado que este punto se encuentra fuera del dominio del problema.

Entonces debemos de:

1. Generar una malla homogénea del dominio

$$x_i = ih, \qquad i = 0, 1, ..., n, \qquad h = \frac{1}{n} = \Delta x. \tag{7.22}$$

2. Sustituir la derivada con la Ec.(8.24)[67] en cada punto donde la solución es desconocida para obtener un sistema algebraico de ecuaciones. Así, en cada punto x_i de la malla aproximamos la ecuación diferencial por

$$u''(x_i) \approx \frac{u(x_i - h) - 2u(x) + u(x_i + h)}{h^2} \tag{7.23}$$

definiendo la solución aproximada de $u(x)$ en x_i como u_i como la solución del siguiente sistema lineal de ecuaciones

$$\begin{aligned}
\frac{u_\alpha - 2u_1 + u_2}{h^2} &= f(x_1) \\
\frac{u_1 - 2u_2 + u_3}{h^2} &= f(x_2) \\
&\vdots \\
\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} &= f(x_i) \\
&\vdots \\
\frac{u_{n-2} - 2u_{n-1} + u_n}{h^2} &= f(x_{n-1}) \\
\frac{u_{n+1} - u_{n-1}}{2h} &= cte_1.
\end{aligned} \tag{7.24}$$

[67] Para mantener la estabilidad es necesario tomar en cuenta las distintas formas de discretización en el Método de Diferencias Finitas mediante el esquema Upwind, Scharfetter-Gummel y Difusión Artificial, entre otros (véase la sección: 7.1.1, Ecs. 7.8 a 7.11).

3. Resolver el sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ (véase 3), obtenemos la solución aproximada en cada punto de la malla. La solución completa al problema la obtenemos al formar el vector

$$\left[\begin{array}{ccccccccc} u_\alpha & u_1 & u_2 & u_3 & \cdots & u_{n-2} & u_{n-1} & u_n \end{array} \right].$$

La implementación de un programa para codificar el Método de Diferencias Finitas en una Dimensión para resolver el problema con condiciones Dirichlet o Neumann del problema

$$p(x)u'' + q(x)u' + r(x)u = f(x)$$

definido en el dominio $[xi, xf]$ y el tipo de frontera en xi -ti igual a -1 Dirichlet ($u(xi) = vi$) ó -2 Neumann ($u_x(xi) = vi$)- y el valor en la frontera xf -tf igual a -1 Dirichlet($u(xf) = vf$) o -2 Neumann ($u_x(xf) = vf$)-, además se le indica el tamaño de la partición n, si se graficara la solución indicando en $grf = 1$, con la solución analítica $s(x)$ y si esta se proporciona $sws = 1$. Regresando la matriz y los vectores $\underline{\underline{A}}\underline{u} = \underline{b}$ del sistema lineal generado así como los puntos del dominio y los valores de la solución en dichos puntos x, V

Ejemplo 27 *El programa queda implementado en OCTAVE (MatLab) como:*

```
function [error,A,b,u,x,V] = fdm1d(p,q,r,f,xi,xf,ti,vi,tf,vf,n,grf,s,sws)
    if n < 3
        return
    end
    % llenado de valores para el tipo de nodo y valor de frontera
    TN = zeros(n,1); % Vector para guardar el tipo de nodo
    V = zeros(n,1); % Vector para guardar la solucion y frontera
    TN(1) = ti;
    TN(n) = tf;
    V(1) = vi;
    V(n) = vf;
    % Calcula el numero de incognitas del problema
    tm = 0;
    for i=1:n,
        if TN(i) == -1 % Descarta nodos frontera Dirichlet
            continue
```

```
        end
        tm = tm +1;
    end
    % Declaracion de la matriz y vectores de trabajo
    %A = sparse(tm,tm);
    A = zeros(tm,tm); % Matriz de carga
    b = zeros(tm,1); % Vector de carga
    u = zeros(tm,1); % Vector de solucion
    x = zeros(n,1); % Vector de coordenadas de la particion
    % Llenado de la matriz y vector
    h = (xf-xi)/(n-1);
    h1 = h*h;
    j = 1;
    for i=1:n,
        x(i) = xi + (i-1)*h;
        if TN(i) == -1 % Descarta nodos frontera Dirichlet
            continue;
        end
        % Diagonal central
        A(j,j) = ((-2.0 * p(x(i))) / h1) + r(x(i));
        if TN(i) == -2
            A(j,j) = -1/h1;
        end
        % Lado derecho
        b(j) = f(x(i));
        % Diagonal anterior
        if TN(i) == -2
            b(j) = V(i)/h;
            if i > 1
                A(j,j-1) = -1/h1;
            end
        elseif TN(i-1) == -1
            b(j) = f(x(i)) - p(x(i))*(V(i-1)/h1);
        else
            A(j,j-1) = p(x(i))/h1 - q(x(i))/(2.0*h);
        end
        % Diagonal superior
        if TN(i) == -2
```

```
            b(j) = V(i)/h;
            if i < n
                  A(j,j+1) = -1/h1;
            end
      elseif TN(i+1) == -1
            b(j) = f(x(i)) - p(x(i))*(V(i+1)/h1);
      else
            A(j,j+1) = p(x(i))/h1 + q(x(i))/(2.0*h);
      end
      j = j + 1;
end
% Soluciona el sistema
u = A\b;
% Copia la solucion obtenida del sistema lineal al vector solucion
j = 1;
for i=1:n,
      if TN(i) == -1 % descarta nodos frontera Dirichlet
            continue
      end
      V(i) = u(j);
      j = j + 1;
end
% Encuentra error en norma infinita usando particion par = 10
error = 0;
if sws == 1
      par = 10;
      for i = 1: n-1,
            inc = (x(i+1)-x(i))/par;
            for j = 1:par+1,
                  px = x(i)+inc*(j-1);
                  e = abs(s(px)-l(px,x(i),V(i),x(i+1),V(i+1)));
                  if e > error
                        error = e;
                  end
            end
      end
end
% Revisa si se graficara
```

```
    if grf == 1
        if sws == 1
            % Calcula la solucion analitica en la particion de calculo
            ua = zeros(n,1);
            for i = 1: n,
                ua(i) = s(x(i));
            end
        end
        % Grafica la solucion numerica
        plot(x,V,'o');
        hold
        % Graficar solucion analitica en particion de tamano xPart
        if sws == 1
            xPart = 1000;
            h = (xf-xi)/(xPart-1);
            xx = zeros(xPart,1);
            xa = zeros(xPart,1);
            for i = 1: xPart,
                xx(i) = xi + (i-1)*h;
                xa(i) = s(xx(i));
            end
            plot(xx,xa);
            % Grafica el error
            figure(2);
            plot(x,V-ua);
        end
    end
end
% evalua el punto x en la recta dada por los puntos (x1,y1) y (x2,y2)
function y = l(x,x1,y1,x2,y2)
    y = y1+((y2-y1)/(x2-x1))*(x-x1);
end
```

Ejemplo 28 *Sea*

$$-u''(x) + u = 0, \qquad 0 \leq x \leq 1, \qquad u(0) = 0, \qquad u(1) = 1$$

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:

```
v=@(x) 1.0; % velocidad, posibles valores 1,25,100, etc.
p=@(x) -1.0;
q=@(x) v(x);
r=@(x) 0;
f=@(x) 0;
s=@(x) (exp(v(x)*x)-1.0)/(exp(v(x))-1.0);
[error,A,b,u,x,V] = fdm1d_DN(p,q,r,f,0,1,-1,0,-1,1,11,1,s,1);
```

Ejemplo 29 *Sea*

$$u''(x) + u = -\pi^2 cos(\pi x), \qquad 0 \leq x \leq 1, \qquad u(0) = 1, \qquad u(1) = -1$$

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:

```
p=@(x) 1;
q=@(x) 0;
r=@(x) 0;
f=@(x) -pi*pi*cos(pi*x);
s=@(x) cos(pi*x);
[error,A,b,u,x,V] = fdm1d_DN(p,q,r,f,0,1,-1,1,-1,-1,40,1,s,1);
```

Ejemplo 30 *Sea*

$$u''(x) = -\pi^2 cos(\pi x), \qquad 0 \leq x \leq 0.5, \qquad u(0) = 1, \qquad u_x(0.5) = -\pi$$

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:

```
p=@(x) 1;
q=@(x) 0;
r=@(x) 0;
f=@(x) -pi*pi*cos(pi*x);
s=@(x) cos(pi*x);
[error,A,b,u,x,V] = fdm1d_DN(p,q,r,f,0,0.5,-1,1,-2,-pi,40,1,s,1);
```

Pese a que con el programa anterior podríamos resolver la ecuación

$$-u''(x) - k^2 u(x) = 0, \qquad 0 \leq x \leq 1, \qquad u(0) = 1, \qquad u'(1) = iku(1)$$

esta ecuación se conoce como la ecuación de Helmholtz la cual es difícil de resolver si $r(x) \leq 0$ y $|r(x)|$ es grande, i.e. $r(x) \sim 1/h^2$. Para resolver correctamente dicha ecuación, usaremos otro programa con los esquemas de discretización de diferencias finitas y diferencias finitas exactas. Este último procedimiento fue desarrollado en: Exact Finite Difference Schemes for Solving Helmholtz Equation at any Wavenumber, Yau Shu Wong and Guangrui Lim International Journal of Numerical Analysis and Modeling, Volumen 2, Number 1, Pages 91-108, 2011. Y la codificación de prueba queda como:

Ejemplo 31 *Sea*

$$-u''(x) - k^2 u(x) = 0, \qquad 0 \leq x \leq 1, \qquad u(0) = 1, \qquad u'(1) = iku(1)$$

con $k = 150$, *entonces el programa en SCILAB queda implementado como:*

```
TEST = 1; // (0) Diferencias finitas, (1) Diferencias finitas exactas

function y=LadoDerecho(x)
 y=0.0;
endfunction

function y=SolucionAnalitica(x, k)
 //y=cos(k*x)+%i*sin(k*x);
 y=exp(%i*k*x);
endfunction

K = 150;
KK = K*K;
a=0; // Inicio dominio
c=1; // Fin dominio
M=300; // Particion
N=M-1; // Nodos interiores
h=(c-a)/(M-1); // Incremento en la malla
Y0=1; // Condicion Dirichlet inicial en el inicio del dominio
Y1=%i*K; // Condicion Neumann inicial en el fin del dominio
```

```
A=zeros(N,N); // Matriz A
b=zeros(N); // Vector b

if TEST = 0 then
R=-1/(h^2);
P=2/(h^2)-KK;
Q=-1/(h^2);
else
R=-1/(h^2);
P=(2*cos(K*h)+(K*h)^2)/(h^2) - KK;
Q=-1/(h^2);
end

// Primer renglon de la matriz A y vector b
A(1,1)=P;
A(1,2)=Q;
b(1)=LadoDerecho(a)-Y0*R; // Frontera Dirichlet
// Renglones intermedios de la matriz A y vector b
for i=2:N-1
 A(i,i-1)=R;
 A(i,i)=P;
 A(i,i+1)=Q;
 b(i)=LadoDerecho(a+h*(i-1));
end
// Renglon final de la matriz A y vector b
if TEST = 0 then
 A(N,N-1)=1/(h^2);
 A(N,N)=-1/(h^2)+ Y1/h;
 b(N)=LadoDerecho(c)/2;
else
 A(N,N-1)=1/(h^2);
 A(N,N)=-1/(h^2)+ %i*sin(K*h)/(h^2);
 b(N)=LadoDerecho(c)/2;
end

// Resuleve el sistema lineal Ax=b
x=inv(A)*b;

ESC = 5;
```

```
xxx=zeros(M*ESC,1);
zzz=zeros(M*ESC,1);
for i=1:M*ESC
 xxx(i)=a+h/ESC*(i-1);
 zzz(i)=SolucionAnalitica(xxx(i),K);
end
// Prepara la graficacion
xx=zeros(M,1);
zz=zeros(M,1);
for i=1:M
 xx(i)=a+h*(i-1);
 zz(i)=SolucionAnalitica(xx(i),K);
end
yy=zeros(M,1);
yy(1)=Y0; // Condicion inicial
for i=1:N
 yy(i+1)=x(i);
end
// Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
plot2d(xx,yy,15)
plot2d(xxx,zzz)
```

7.1.4 Ecuación con Primera Derivada Temporal

Hasta ahora se ha visto como discretizar la parte espacial de las ecuaciones diferenciales parciales, lo cual nos permite encontrar la solución estática de los problemas del tipo elíptico. Sin embargo, para ecuaciones del tipo parabólico e hiperbólico dependen del tiempo, se necesita introducir una discretización a las derivadas con respecto del tiempo. Al igual que con las discretizaciones espaciales, podemos utilizar algún esquema de diferencias finitas en la discretización del tiempo.

Para la solución de la ecuaciones diferenciales con derivada temporal (u_t), se emplean diferentes esquemas en diferencias finitas para la discretización del tiempo. Estos esquemas se conocen de manera general como esquemas $theta(\theta)$.

Definiendo la ecuación diferencial parcial general de segundo orden como

$$u_t = \mathcal{L}u \tag{7.25}$$

donde

$$\mathcal{L}u = \left(p\left(x\right)u'\left(x\right)\right)' + q\left(x\right)u'\left(x\right) - r\left(x\right)u\left(x\right) - f\left(x\right) \tag{7.26}$$

aquí, los coeficientes p, q y r pueden depender del espacio y del tiempo. Entonces el esquema *theta* está dado por

$$u_t = \left(1-\theta\right)\left(\mathcal{L}u\right)^j + \theta\left(\mathcal{L}u\right)^{j+1}. \tag{7.27}$$

Existen diferentes casos del esquema *theta* a saber:

- Para $\theta = 0$, se obtiene un esquema de diferencias finitas hacia adelante en el tiempo, conocido como esquema completamente explícito, ya que el paso $n+1$ se obtiene de los términos del paso anterior n. Es un esquema sencillo, el cual es condicionalmente estable cuando $\Delta t \leq \frac{\Delta x^2}{2}$.
- Para $\theta = 1$, se obtiene el esquema de diferencias finitas hacia atrás en el tiempo, conocido como esquema completamente implícito, el cual es incondicionalmente estable.
- Para $\theta = \frac{1}{2}$, se obtiene un esquema de diferencias finitas centradas en el tiempo, conocido como esquema Crank-Nicolson, este esquema es también incondicionalmente estable y es más usado por tal razón.

Para la implementación del esquema Crank-Nicolson se toma una diferencia progresiva para el tiempo y se promedian las diferencias progresivas y regresivas en el tiempo para las derivadas espaciales. Entonces, si tenemos la Ec. (7.26), las discretizaciones correspondientes son[68]:

$$u_t \simeq \frac{u_i^{j+1} - u_i^j}{\Delta t} \tag{7.28}$$

$$\left(p\left(x\right)u'\left(x\right)\right)' \simeq \frac{p}{2}\left[\frac{u_{i-1}^j - 2u_i^j + u_{i+1}^j}{\Delta x^2} + \frac{u_{i-1}^{j+1} - 2u_i^{j+1} + u_{i+1}^{j+1}}{\Delta x^2}\right] \tag{7.29}$$

[68] Para mantener la estabilidad es necesario tomar en cuenta las distintas formas de discretización en el Método de Diferencias Finitas mediante el esquema Upwind, Scharfetter-Gummel y Difusión Artificial, entre otros (véase sección: 7.1.1, Ecs. 7.8 a 7.11).

$$q\left(x\right)u'\left(x\right) \simeq \frac{q}{2}\left[\frac{u_{i-1}^{j}+u_{i+1}^{j}}{2\Delta x}+\frac{u_{i-1}^{j+1}+u_{i+1}^{j+1}}{2\Delta x}\right] \tag{7.30}$$

además de $r(x), u_i^j$ y f_i^j.

Entonces, una vez que se sustituyen las derivadas por su forma en diferencias finitas, lo siguiente es formar el sistema:

$$Au^{j+1} = Bu^j + f^j \tag{7.31}$$

esto se logra, colocando del lado izquierdo la igualdad de los términos que contengan el paso del tiempo correspondiente a $j+1$ y del lado derecho a los términos correspondientes de j.

A continuación, veamos un ejemplo del esquema Crank-Nicolson desarrollados en SCILAB[69]

Ejemplo 32 *Sea*

$$u_t - a(x)u''(x) - b(x)u'(x) + c(x)u = f, \qquad l0 \leq x \leq l, \qquad 0 < t < T$$

entonces el programa queda implementado como:

```
// Crank-Nicolson
// Para una EDP de segundo orden
// u_t + a(x)u_xx + b(x)u_x +c(x)u = f
// Dominio
// l0<x<l
// 0<t<T
// Condiciones de frontera Dirichlet
// u(0,t) = u(l,t) = constante 0 < t < T cond de frontera
// Condicion inicial
// u(x,0) = g(x) l0 <= x <= l
// Datos de entrada
// intrevalo [l0, l]
// entero m>=3
// entero N>=1
// Salida
```

[69] Scilab es un programa Open Source para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería [http://www.scilab.org].

```
// aproximaciones w_ij a u(x_i,t_j)
// Funciones de los coeficientes
function y = a(x)
y = -1;
endfunction
function y = b(x)
y = -1;
endfunction
function y = c(x)
y = 1;
endfunction
function y = f(x)
y = 0;
endfunction
// funcion de la condicion inicial
function y = condicion_inicial(x)
y = sin(x * %pi);
endfunction
// Condiciones de frontera
// Solo Dirichlet
cond_izq = 0;
cond_der = 0;
// Datos de entrada
l0 = 0;l = 1; // intervalo [0,1]
m = 11; // Numero de nodos
M = m - 2; // Nodos interiores
// Division del espacio y del tiempo
h = (l - l0)/(m-1);
k = 0.025; // Paso del tiempo
N = 50; // Numero de iteraciones
// Aw^(j+1) = Bw^j + f^j
// creo el vector w donde se guardara la solucion para cada tiempo
// A matriz del lado izquierdo
// B matriz del lado derecho
// B_prima para guardar el resultado de Bw^j
// ff que es el vector de los valores de f en cada nodo
w = zeros(M,1);
ff = zeros(M,1)
```

```
A = zeros(M,M);
B = zeros(M,M);
//B_prima = zeros(M,1);
w_sol = zeros(m,m)
// Se crea el espacio de la solucion o malla
espacio = zeros(M,1)
for i = 1:m
xx = l0 + (i-1)*h;
espacio(i) = xx;
end
disp(espacio, "Espacio")
// Condicion inicial
for i=1:M
w(i) = condicion_inicial(espacio(i+1));
end
w_sol(1) = cond_izq;
for kk = 1:M
w_sol(kk + 1) = w(kk);
end
w_sol(m) = cond_izq;
plot(espacio, w_sol);
disp(w, "Condiciones iniciales")
// primer renglon de cada matriz
A(1,1) = 1/k - a(l0 + h)/(h*h);
A(1,2) = a(l0 + h)/(2*h*h) + b(l0 + h)/(4*h) ;
B(1,1) = 1/k + a(l0 + h)/(h*h) - c(l0 + h);
B(1,2) = - a(l0 + h)/(2*h*h) - b(l0 + h)/(4*h);
ff(1) = f(l0 + h) - cond_izq;
// se completa las matrices desde el renglon 2 hasta el m-2
for i = 2:M-1
A(i, i-1) = a(l0 + i*h)/(2*h*h) - b(l0 + i*h)/(4*h) ;
A(i,i) = 1/k - a(l0 + i*h)/(h*h);
A(i,i+1) = a(l0 + i*h)/(2*h*h) + b(l0 + i*h)/(4*h) ;
B(i, i-1) = - a(l0 + i*h)/(2*h*h) + b(l0 + i*h)/(4*h);
B(i,i) = 1/k + a(l0 + i*h)/(h*h) - c(l0 + i*h);
B(i,i+1) = - a(l0 + i*h)/(2*h*h) - b(l0 + i*h)/(4*h);
end
// Ultimo renglon de cada matriz
```

```
A(M,M-1) = a(l - h)/(2*h*h) - b(l-h)/(4*h) ;
A(M,M) = 1/k - a(l - h)/(h*h);
B(M,M-1) = - a(l-h)/(2*h*h) + b(l-h)/(4*h);
B(M,M) = 1/k + a(l-h)/(h*h) - c(l-h);
ff(M) = f(l - h) - cond_der;
// Resolvemos el sistema iterativamente
for j=1:21
t = j*k;
B_prima = B * w + ff;
w = inv(A) * B_prima;
disp(t, "tiempo")
disp(w, "Sol")
w_sol(1) = cond_izq;
for kk = 1:M
  w_sol(kk + 1) = w(kk);
end
w_sol(m) = cond_izq;
plot(espacio, w_sol);
end
```

7.1.5 Ecuación con Segunda Derivada Temporal

Para el caso de ecuaciones con segunda derivada temporal, esta se aproxima por diferencias finitas centradas en el tiempo, partiendo de la Ec. (7.26), las discretizaciones correspondientes son[70]

$$u_{tt} \simeq \frac{u_i^{j-1} - 2u_i^j + u_i^{j+1}}{\Delta t^2} \tag{7.32}$$

$$\left(p\left(x\right)u'\left(x\right)\right)' \simeq p\left[\frac{u_{i-1}^j - 2u_i^j + u_{i+1}^j}{\Delta x^2}\right] \tag{7.33}$$

$$q\left(x\right)u'\left(x\right) \simeq q\left[\frac{u_{i-1}^j + u_{i+1}^j}{2\Delta x}\right] \tag{7.34}$$

[70] Para mantener la estabilidad es necesario tomar en cuenta las distintas formas de discretización en el Método de Diferencias Finitas mediante el esquema Upwind, Scharfetter-Gummel y Difusión Artificial, entre otros (véase sección: 7.1.1, Ecs. 7.8 a 7.11).

además de $r(x)$, u_i^j y f_i^j.

Entonces, una vez que se sustituyen las derivadas por su forma en diferencias finitas, lo siguiente es formar el sistema

$$u_i^{j+1} = 2u_i^j - u_i^{j-1} + (\Delta t)^2 \, Bu^j \tag{7.35}$$

esto se logra, colocando del lado izquierdo la igualdad de los términos que contengan el paso del tiempo correspondiente a $j+1$ y del lado derecho a los correspondientes términos de j y $j-1$. Para calcular u_i^{j+1} es necesario conocer u_{i-1}, u_i, u_{i+1} en los dos instantes inmediatos anteriores, i.e. t_j y t_{j-1}.

En particular para calcular u_i^1 es necesario conocer u_i^0 y u_i^{-1}, si consideramos

$$u_i^{j+1} = 2u_i^j - u_i^{j-1} + (\Delta t)^2 \, \mathcal{L}u^j \tag{7.36}$$

para $j = 0$, entonces

$$u_i^1 = 2u_i^0 - u_i^{-1} + (\Delta t)^2 \, \mathcal{L}u^0 \tag{7.37}$$

donde u_i^0 es la condición inicial y $\frac{u_i^1 - u_i^{-1}}{2\Delta t}$ es la primer derivada de la condición inicial. Así, para el primer tiempo tenemos

$$u_i^1 = u(x_i, 0) + \Delta t u^{'}(x_i, 0) + (\Delta t)^2 \, \mathcal{L}u^0 \tag{7.38}$$

lo cual permite calcular u_i^1 a partir de las condiciones iniciales.

La derivación del método parte de

$$\begin{aligned} u_{tt} &= \mathcal{L}u^j \\ \frac{u_i^{j-1} - 2u_i^j + u_i^{j+1}}{(\Delta t)^2} &= \mathcal{L}u^j \\ u_i^{j+1} &= 2u_i^j - u_i^{j-1} + (\Delta t)^2 \, \mathcal{L}u^j \end{aligned} \tag{7.39}$$

donde el error intrínseco a la discretización es de orden cuadrático, pues se ha usado diferencias centradas, tanto para el espacio como para el tiempo.

Ejemplo 33 *Sea*

$$u_{tt} - 4u''(x) = 0, \qquad 0 \le x \le l, \qquad 0 < t < T$$

sujeta a

$$u(0,t) = u(1,t) = 0, \qquad u(x,0) = \sin(\pi x), \qquad u_t(x,0) = 0$$

con solución analítica

$$u(x,t) = sen(\pi x) * cos(2\pi t)$$

entonces el programa queda implementado como:

```
// Dominio
a_ = 0
b_ = 1
// Particion en x
m = 101; // numero de nodos
h = (b_ - a_)/(m-1)
dt = 0.001 // salto del tiempo
// Para que sea estable se debe cumplir que
// sqrt(a) <= h/dt
// Coeficiente
function y = a(x)
y = -4;
endfunction
// Condicion inicial
function y = inicial(x)
y = sin(%pi * x)
endfunction
function y = u_t(x)
y = 0;
endfunction
// Solucion analitica
function y = analitica(x,t)
y = sin(%pi * x) * cos(2* %pi * t)
endfunction
////////////////////////////////////
// Aw^(j+1) = Bw^j
// creo el vector w donde se guradaria la solucion para cada tiempo
// A matriz del lado izquierdo
// B matriz del lado derecho
// B_prima para guardar el resultado de Bw^j
```

```
w_sol = zeros(m,1);
w_sol_temp = zeros(m,1);
w_temp = zeros(m,1);
w = zeros(m,1);
w_ = zeros(m,1);
A = eye(m,m);
B = zeros(m,m);
B_prima = zeros(m,1);
espacio = zeros(m,1)
sol = zeros(m,1);
// primer renglon de cada matriz
B(1,1) = 2*a(a_)*dt*dt/(h*h)
B(1,2) = -a(a_)*dt*dt/(h*h)
// se completa las matrices desde el renglon 2 hasta el m-1
for i = 2:m-1
B(i, i-1) = -a(i*h)*dt*dt/(h*h)
B(i,i) = 2*a(i*h)*dt*dt/(h*h)
B(i,i+1) = -a(i*h)*dt*dt/(h*h)
end
// Ultimo renglon de cada matriz
B(m,m-1) = -a(b_)*dt*dt/(h*h)
B(m,m) = 2*a(b_)*dt*dt/(h*h)
// muestro la matriz
//printf("Matriz B\n");
//disp(B);
for i=1:m
xx = (i-1)*h;
espacio(i) = a_ + xx;
w(i) = inicial(espacio(i)); // Condiciones iniciales
w_(i) = inicial(espacio(i)) + u_t(espacio(i)) * dt
end
//
//disp(espacio)
//disp(w)
////////////
//Para t = 0
B_prima = B * w;
for i = 1:m
```

```
w_sol(i) = w_(i) + B_prima(i);
end
////////////
printf("w para t = 0\n");
disp(w_sol);
for i = 1:m
sol(i) = analitica(espacio(i), 0)
end
printf("Solucion analitica para t = 0\n")
disp(sol)
plot(espacio,w_sol)
//plot(espacio,sol,'r')
w_sol_temp = w_sol;
w_temp = w
for i=1:500
t = i*dt
B_prima = B * w_sol_temp;
w_ = 2 * w_sol_temp - w_temp
w_sol = w_ + B_prima;
////
// for j = 1:m
// sol(j) = analitica(espacio(j), t)
// end
//
// printf("Sol analitica dt = %f", t)
// disp(sol)
// printf("Sol metodo dt = %f", t)
// disp(w_sol)
w_temp = w_sol_temp
w_sol_temp = w_sol
if i == 5 | i == 50 | i == 100 | i == 150 | i == 200 | i == 250 | i ==
300 | i== 350 | i == 400 | i == 450 | i == 500 then
plot(espacio,w_sol)
end
//plot(espacio,sol,'r')
end
```

7.2 Consistencia, Estabilidad, Convergencia y Error del Método de Diferencias Finitas

Cuando se usa algún método para la resolución de ecuaciones diferenciales, se necesita conocer cuán exacta es la aproximación obtenida en comparación con la solución analítica (en caso de existir).

Error Global Sea $U = [U_1, U_2, ..., U_n]^T$ el vector solución obtenido al utilizar el método de diferencias finitas y $u = [u(x_1), u(x_n), ..., u(x_n)]$ la solución exacta de los puntos de la malla. El vector de error global se define como $E = U - u$, lo que se desea es que el valor máximo sea pequeño. Usualmente se utilizan distintas normas para encontrar el error.

- La norma infinito, definida como $\|E\|_\infty = \max |e_i|$
- La norma-1, definida como $\|E\|_1 = \sum_{i=1}^{n} |e_i|$
- La norma-2, definida como $\|E\|_2 = \left(\sum_{i=1}^{n} e_i^2 \right)^{\frac{1}{2}}$

La norma infinito $\|E\|_\infty$ es en general, la más apropiada para calcular los errores relativos a la discretización.

Definición 10 *Si* $\|E\| \leq Ch^p, p > 0$, *decimos que el método de diferencias finitas es de orden-p de precisión.*

Definición 11 *Un método de diferencias finitas es llamado convergente si*

$$\lim_{h \to 0} \|E\| = 0. \tag{7.40}$$

Error de Truncamiento Local Sea el operador diferencia $\mathcal{L}u$, definimos el operador en diferencias finitas $\mathcal{L}_h$, por ejemplo para la ecuación de segundo orden $u''(x) = f(x)$, uno de los operadores de diferencias finitas puede ser

$$\mathcal{L}_h u(x) = \frac{u(x-h) - 2u(x) + u(x+h)}{h^2}. \tag{7.41}$$

Definición 12 *El error de truncamiento local es definido como*

$$T(x) = \mathcal{L}u - \mathcal{L}_h u. \tag{7.42}$$

Para la ecuación diferencial $u''(x) = f(x)$ y el esquema de diferencias centradas usando tres puntos $\frac{u(x-h)-2u(x)+u(x+h)}{h^2}$, el error de truncamiento local es

$$\begin{aligned} T(x) &= \mathcal{L}u - \mathcal{L}_h u = u''(x) - \frac{u(x-h)-2u(x)+u(x+h)}{h^2} \\ &= f(x) - \frac{u(x-h)-2u(x)+u(x+h)}{h^2}. \end{aligned} \tag{7.43}$$

Note que el error de truncamiento local sólo depende de la solución del estencil en diferencias finitas (en el ejemplo es usando tres puntos) pero no depende de la solución global, es por ello que se denomina error de truncamiento local. Este es una medida de que tanto la discretización en diferencias finitas se aproxima a la ecuación diferencial.

Definición 13 *Un esquema de diferencias finitas es llamado consistente si*

$$\lim_{h\to 0} T(x) = \lim_{h\to 0} (\mathcal{L}u - \mathcal{L}_h u) = 0. \tag{7.44}$$

Si $T(x) = Ch^p, p > 0$, entonces se dice que la discretización es de orden$-p$ de precisión, donde C es una constante independiente de h pero puede depender de la solución de $u(x)$. Para conocer cuándo un esquema de diferencias finitas es consistente o no, se usa la expansión de Taylor. Por ejemplo, para el esquema de diferencias finitas centradas usando tres puntos para la ecuación diferencial $u''(x) = f(x)$, tenemos que

$$T(x) = u''(x) - \frac{u(x-h)-2u(x)+u(x+h)}{h^2} = -\frac{h^2}{12}u^{(4)}(x) + ... = Ch^2 \tag{7.45}$$

donde $C = \frac{1}{12}u^{(4)}(x)$. Por lo tanto, este esquema de diferencias finitas es consistente y la discretización es de segundo orden de precisión.

La consistencia no puede garantizar que un esquema de diferencias finitas trabaje. Para ello, necesitamos determinar otra condición para ver si converge o no. Tal condición es llamada la estabilidad de un método de diferencias finitas. Para el problema modelo, tenemos que

$$Au = F + T, \qquad AU = F, \qquad A(u - U) = T = -AE \tag{7.46}$$

donde A son los coeficientes de la matriz de las ecuaciones en diferencias finitas, F son los términos modificados por la condición de frontera, y T es el vector local de truncamiento en los puntos de la malla.

Si la matriz A es no singular, entonces $\|E\| = \|A^{-1}T\| \leq \|A^{-1}\| \|T\|$. Para el esquema de diferencias finitas centradas, tenemos que $\|E\| = \|A^{-1}\| h^2$. Tal que el error global depende del error de truncamiento local y $\|A^{-1}\|$.

Definición 14 *Un método de diferencias finitas para la ecuación diferencial elíptica es estable si A es invertible y*

$$\left\|A^{-1}\right\| \leq C, \text{ para todo } h \leq h_0 \tag{7.47}$$

donde C y h_0 son constantes.

Teorema 15 *Si el método de diferencias finitas es estable y consistente, entonces es convergente.*

7.3 Método de Diferencias Finitas en Dos y Tres Dimensiones

Sin pérdida de generalidad y a modo de ejemplificar, consideremos la ecuación diferencial parcial en dos dimensiones

$$\left(p\left(x,y\right)u'\left(x,y\right)\right)' + q\left(x,y\right)u'\left(x,y\right) - r\left(x,y\right)u\left(x,y\right) = f\left(x,y\right) \tag{7.48}$$

$$\text{en } a \leq x \leq b \text{ y } c \leq y \leq d \qquad \text{donde: } u\left(x,y\right) = u_{xy}$$

esta definida en la frontera con condiciones de frontera Dirichlet, Neumann o Robin. Para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, debemos:

1. Generar una malla del dominio, i.e. una malla es un conjunto finito de puntos en los cuales buscaremos la solución aproximada a la ecuación diferencial parcial.

2. Sustituir las derivadas correspondientes con alguna de las fórmulas de diferencias finitas centradas (véase 8.33 y 8.35 para dos dimensiones, 8.42 y 8.44 para tres dimensiones), en cada punto donde la solución es desconocida para obtener un sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$.

3. Resolver el sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ (véase 3), y así obtener la solución aproximada en cada punto de la malla.

Problema con Condiciones de Frontera Dirichlet Consideremos un caso particular de la Ec.(7.48) definido por la ecuación

$$u_{xx} + u_{yy} = 0, \qquad 0 \leq x \leq 1 \text{ y } 0 \leq y \leq 1, \qquad u(x,y) = u_{xy} \tag{7.49}$$

con condiciones de frontera Dirichlet. Para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, debemos:

1. Generar una malla homogénea del dominio, sin pérdida de generalidad lo supondremos un rectángulo[71]

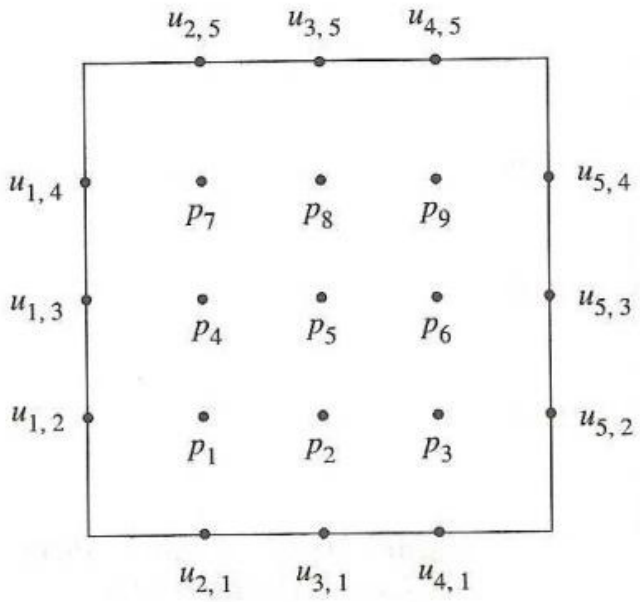


Figura 1: Si suponemos una malla discretizada en $5x5$ nodos, donde 9 serían interiores y 16 de frontera para tener un total de 25 nodos, donde la evaluación de los 9 nodos interiores los denotamos con $p_1, ..., p_9$.

2. Sustituir la derivada con la Ec.(8.35) en cada punto donde la solución es desconocida para obtener un sistema algebraico de ecuaciones. Así, en

[71] En el caso de que la malla no sea homogénea, es necesario incluir en la derivada a que h_x (h_{x+} y h_{x-}) y k_y (k_{y+} y k_{y-}) se refiere (como se hizo en una dimensión).

cada punto x, y de la malla aproximamos la ecuación diferencial por[72]

$$\begin{aligned} u_{xx} &\simeq \frac{u(x_i - \Delta x, y_j) - 2u(x_i, y_j) + u(x_i + \Delta x, y_j)}{\Delta x^2} \\ u_{yy} &\simeq \frac{u(x_i, y_j - \Delta y) - 2u(x_i, y_j) + u(x_i, y_j + \Delta y)}{\Delta y^2} \end{aligned} \tag{7.50}$$

o en su forma simplificada

$$\begin{aligned} u_{xx} &\simeq \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} \\ u_{yy} &\simeq \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{k^2} \end{aligned} \tag{7.51}$$

definiendo la solución aproximada de $u(x, y)$ en x_i, y_j como u_{ij}, y si suponemos que $h = k$, entonces la fórmula para la aproximación es

$$\frac{u_{i-1,j} + u_{i+1,j} - 4u_{i,j} + u_{i,j-1} + u_{i,j+1}}{h^2} = 0$$

si suponemos una malla discretizada en $5x5$ nodos, donde 9 serían interiores y 16 de frontera para tener un total de 25 nodos, donde la evaluación de los 9 nodos interiores los denotamos con $p_1, ..., p_9$ tendríamos algo como:

$$\begin{bmatrix} -4p_1 & +p_2 & & +p_4 & & & & & \\ p_1 & -4p_2 & +p_3 & & +p_5 & & & & \\ & p_2 & -4p_3 & & & +p_6 & & & \\ p_1 & & & -4p_4 & +p_5 & & p_7 & & \\ & p_2 & & +p4 & -4p_5 & p_6 & & p_8 & \\ & & p_3 & & +p_5 & -4p_6 & & & p_9 \\ & & & p_4 & & & -4p_7 & p_8 & \\ & & & & p_5 & & +p_7 & -4p_8 & p_9 \\ & & & & & p_6 & & p_8 & -4p_9 \end{bmatrix} = \begin{bmatrix} -u_{2,1} - u_{1,2} \\ -u_{3,1} \\ -u_{4,1} - u_{5,2} \\ -u_{1,3} \\ 0 \\ -u_{5,3} \\ -u_{2,5} - u_{1,4} \\ -u_{3,5} \\ -u_{4,5} - u_{5,4} \end{bmatrix}$$

3 Al resolver el sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{p} = \underline{f}$ (véase apéndice 3), obtenemos la solución aproximada en cada punto interior de la malla. La solución completa al problema la obtenemos al agregar los valores de la frontera que eran conocidos.

[72] Notemos que en cada punto de la malla, la aproximación por diferencias finitas supone la solución de cinco puntos de la malla. El conjunto de estos puntos de la malla es comúnmente llamado el esténcil de diferencias finitas en dos dimensiones.

7.3.1 Procedimiento General para Programar MDF

Como para casi todo problema no trivial que se desee programar, es necesario tener claro el procedimiento matemático del problema en cuestión. A partir de los modelos matemáticos y los modelos numéricos se plasmará el modelo computacional en algún lenguaje de programación usando algún paradigma de programación soportado por el lenguaje seleccionado.

En particular, para programar el método de Diferencias Finitas que no es un problema trivial, debemos empezar seleccionando la ecuación diferencial parcial más sencilla, con una malla lo más pequeña posible pero adecuada a la dimensión de nuestro problema, esto nos facilitará hacer los cálculos entendiendo a cabalidad el problema. Además de esta forma podemos encontrar errores lógicos, conceptuales, de cálculo en las derivadas, ensamble en las matrices y solución del sistema lineal $Ax = b$ asociado, de otra forma se está complicando innecesariamente el proceso de programación y depuración. Además tratar de rastrear errores sin tener primero claro los cálculos a los que debe llegar el programa es una complicación innecesaria.

Este procedimiento sirve tanto para *1D, 2D o 3D*, pero lo recomendable es iniciar en *1D* para adquirir la pericia necesaria para las dimensiones mayores, mediante el siguiente procedimiento:

a) Supondremos la ecuación diferencial parcial más sencilla, la ecuación de Poisson, con condiciones de frontera Dirichlet, definido en Ω como:

$$\begin{aligned} -\nabla^2 u &= f_\Omega \text{ en } \Omega \\ u &= g_{\partial\Omega} \text{ en } \partial\Omega. \end{aligned} \tag{7.52}$$

b) Generar la matriz correspondiente y compararla con la calculada por nosotros

c) Si la matriz es correcta, ahora usar una ecuación no homogénea constante para calcular el vector asociado y compararlo con lo calculado por nosotros, de ser necesario ahora podemos cambiar a una función del lado derecho cualquiera

d) En la malla homogénea más sencilla aplicable a la dimensión del problema -para *1D* una malla de 3 nodos, para *2D* una malla de *3x3* nodos y para *3D* una malla de *3x3x3* nodos- ahora podemos hacer el ensamble de la matriz global A y el vector b y compararla con la calculada por nosotros

e) Si es correcto el ensamble ahora podemos variar las dimensiones de nuestro dominio y la cantidad de nodos usados en la discretización -en *2D*

ahora podemos usar una malla del tipo *2x3* y *3x2*, algo similar en *3D* se puede hacer- para validar el ensamble correcto de la matriz, superado este punto ahora podemos ampliar la malla y ver que el ensamble se mantiene bien hecho -en *2D* la malla mínima adecuada es *3x3* nodos, en *3D* será de *3x3x3* nodos-

f) Regresando a la malla mínima aplicable a la dimensión del problema, ahora solucionar el sistema $Ax = b$ por inversa y revisar que concuerde con la solución calculada por nosotros. Si es correcta, ahora podemos usar algún método del tipo *CGM* o *GMRES* según sea simétrica o no simétrica la matriz A

g) Si contamos con la solución analítica de alguna ecuación, ahora podemos calcular la solución numérica por MDF para distintas mallas y compararla con la analítica y poder ver si la convergencia es adecuada al aumentar la malla

h) llegando a este punto, podemos ahora ampliar los términos de la ecuación diferencial parcial y bastará con revisar que la matriz y vector asociada sea bien generada para tener certeza de que todo funcionará.

7.4 Las Ecuaciones de Navier-Stokes

Las ecuaciones de Navier-Stokes reciben su nombre de los físicos Claude Louis Navier y George Gabriel Stokes (Francés e Irlandés respectivamente), quienes aplicaron los principios de la mecánica y termodinámica, resultando las ecuaciones en derivadas parciales no lineales que logran describir el comportamiento de un fluido. Estas ecuaciones no se concentran en una posición sino en un campo de velocidades, más específicamente en el flujo de dicho campo, lo cual es la descripción de la velocidad del fluido en un punto dado en el tiempo y en el espacio.

Cuando se estudia la dinámica de fluidos se consideran los siguientes componentes:

- $\overrightarrow{\mu}$, este término se usa para definir la velocidad del fluido
- ρ, este término se relaciona con la densidad del fluido
- p, este término hace referencia con la presión o fuerza por unidad de superficie que el fluido ejerce

- g, este término se relaciona con la aceleración de la gravedad, la cuál está aplicada a todo el cuerpo, en determinados casos la gravedad no es la única fuerza ejercida sobre el cuerpo, por tal razón se toma como la sumatoria de fuerzas externas
- v, este término hace referencia a la viscosidad cinemática

La ecuación de Navier-Stokes general es

$$\rho\left(\frac{\partial v}{\partial t} + v \cdot \nabla v\right) = -\nabla p + \nabla \cdot \mathbb{T} + f \tag{7.53}$$

el término $\mathbb{T}$, expresa el tensor de estrés para los fluidos y condensa información acerca de las fuerzas internas del fluido. Supondremos al fluido incompresible cuyo campo de velocidades $\overrightarrow{\mu}$ satisface la condición $\nabla \cdot \overrightarrow{\mu} = 0$, por tal razón el tensor de estrés se reduce a $\nabla^2 \overrightarrow{\mu}$, dando como resultado la siguiente ecuación

$$\frac{\partial \overrightarrow{\mu}}{\partial t} + \overrightarrow{\mu} \cdot \nabla \overrightarrow{\mu} + \frac{1}{\rho}\nabla p = g + v\nabla^2 \overrightarrow{\mu} \tag{7.54}$$

bajo la condición

$$\nabla \overrightarrow{\mu} = 0 \tag{7.55}$$

que garantiza la incompresibilidad del campo de velocidad sí la densidad permanece constante en el tiempo.

Así, las ecuaciones simplificadas de Navier-Stokes quedan como

$$\frac{\partial \overrightarrow{\mu}}{\partial t} = -\left(\overrightarrow{\mu}\nabla\right)\overrightarrow{\mu} + v\nabla^2 \overrightarrow{\mu} + f \tag{7.56}$$

$$\frac{\partial \rho}{\partial t} = -\left(\overrightarrow{\mu}\nabla\right)\rho + \kappa\nabla^2\rho + S \tag{7.57}$$

donde la primera ecuación describe la velocidad y la segunda hace referencia a la densidad a través de un campo vectorial. Estas ecuaciones son no lineales y no se conoce solución analítica a dicho problema.

La Solución Numérica de las Ecuaciones de Navier-Stokes La solución numérica de las ecuaciones (7.56, 7.57) partiendo del estado inicial $\overrightarrow{\mu}_0$ del campo de velocidad y el primer instante de tiempo $t = 0$, se busca estudiar su comportamiento para el tiempo $t > 0$. Si $\overrightarrow{\omega}_0$ es la estimación del campo vectorial en el instante t y $\overrightarrow{\omega}_4$ denota el campo vectorial de velocidades en el instante de tiempo $t + \Delta t$. Jos Stam (véase [**?**]) propone una solución numérica a partir de la descomposición de la ecuación de distintos términos y solucionando cada uno individualmente. Así, la solución de cada paso se construye a partir del paso anterior. La solución en el tiempo $t + \Delta t$, está dada por el campo de velocidades $u(x, t + \Delta t) = \overrightarrow{\omega}_4$, la solución se obtiene iterando estos cuatro pasos:

1. Sumatoria de Fuerzas
2. Paso de Advección
3. Paso de Difusión
4. Paso de Proyección

i.e. $\overrightarrow{\omega}_0 \overset{1}{\rightarrow} \overrightarrow{\omega}_1 \overset{2}{\rightarrow} \overrightarrow{\omega}_2 \overset{3}{\rightarrow} \overrightarrow{\omega}_3 \overset{4}{\rightarrow} \overrightarrow{\omega}_4$

Sumatoria de Fuerzas La manera más práctica para incorporar la sumatoria de fuerzas externas f, se obtiene asumiendo que la fuerza no varía de manera considerable en un paso de tiempo. Bajo este supuesto, utilizando el método de Euler progresivo para resolver ecuaciones diferenciales ordinarias (teniendo en cuenta que la velocidad y la fuerza estan relacionadas mediante la segunda ley de Newton), tenemos que

$$\overrightarrow{\omega}_1(x) = \overrightarrow{\omega}_0(x) + \Delta t f(x, t). \tag{7.58}$$

Paso de Advección Una perturbación en cualquier parte del fluido se propaga de acuerdo a la expresión

$$-(\overrightarrow{\mu} \cdot \nabla)\overrightarrow{\mu} \tag{7.59}$$

este término hace que la ecuación de Navier-Stokes sea no lineal, de aplicarse el método de Diferencias Finitas, éste es estable sólo cuando Δt sea suficientemente pequeño tal que $\Delta t < \Delta h / |u|$, donde Δh es el menor incremento

de la malla de discretización, para prevenir esto, se usa el método de las características. Este dice que una ecuación de la forma

$$\frac{\partial \alpha(x,t)}{\partial t} = -v(x)\nabla\alpha(x,t) \tag{7.60}$$

y

$$\alpha(x,t) = \alpha_0(x) \tag{7.61}$$

como las características del vector del campo v, que fluye a través del punto x_0 en $t = 0$, i.e.

$$\frac{d}{dt}p(x_0,t) = v(p(x_0,t)) \tag{7.62}$$

donde

$$p(x_0,0) = x_0. \tag{7.63}$$

De modo tal que $\alpha(x_0,t) = \alpha\left(p(x_0,t),t\right)$, es el valor del campo que pasa por el punto x_0 en $t = 0$. Para calcular la variación de esta cantidad en el tiempo se usa la regla de la cadena

$$\frac{d\alpha}{dt} = \frac{\partial\alpha}{\partial t} + v\nabla\alpha = 0 \tag{7.64}$$

que muestra que el valor de α no varía a lo largo de las líneas del flujo. En particular, se tiene que $\alpha(x_0,t) = \alpha(x_0,0) = \alpha_0(x_0)$; por lo tanto en el campo inicial de las líneas de flujo, en el sentido inverso se puede estimar el siguiente valor de α en x_0, esto es $\alpha\left(x_0, t+\Delta t\right)$.

Este método muestra en cada paso del tiempo, como las partículas del fluido se mueven por la velocidad del propio fluido. Por lo tanto, para obtener la velocidad en un punto x en el tiempo $t + \Delta t$, se regresa al punto x por el campo de velocidades $\overrightarrow{\omega}_1$ en el paso del tiempo Δt. Esta define una ruta $p(x,s)$ correspondiente a la trayectoria parcial del campo de velocidades. La nueva velocidad en el punto x, es entonces la velocidad de la partícula ahora en x que tenía su ubicación anterior en el tiempo Δt. De esta manera, se puede hallar el valor de la velocidad luego de la advección $\overrightarrow{\omega}_2$ resolviendo el problema

$$\overrightarrow{\omega}_2(x) = \overrightarrow{\omega}_1(p(x,-\Delta t)) \tag{7.65}$$

la ventaja que se obtiene interpolando linealmente entre valores adyacentes de este método es su estabilidad numérica. Ya conocida la velocidad en el instante t, la viscosidad del fluido y su naturaleza, obligan un proceso difusivo: la velocidad se propaga dentro del fluido, o bien en las partículas de este fluyen.

Paso de Difusión Este paso es para resolver el efecto de la viscosidad y es equivalente a la ecuación de difusión

$$\frac{\partial \overrightarrow{\omega}_2(x)}{\partial t} = \nu \nabla^2 \overrightarrow{\omega}_2(x) \tag{7.66}$$

esta es una ecuación para la cual se han desarrollado varios procedimientos numéricos. La forma más sencilla para resolver esta ecuación es discretizar el operador difusión ∇^2 y usar una forma explícita en el incremento del tiempo, sin embargo este método es inestable cuando la viscosidad es grande. Por ello una mejor opción es usar un método implícito para la ecuación

$$\left(\mathcal{I} - \nu \Delta t \nabla^2\right) \overrightarrow{\omega}_3(x) = \overrightarrow{\omega}_2(x) \tag{7.67}$$

donde $\mathcal{I}$ es el operador identidad.

Paso de Proyección Este último paso conlleva la proyección, que hace que resulte un campo libre de divergencia. Esto se logra mediante la solución del problema definido por

$$\nabla^2 q = \nabla \cdot \overrightarrow{\omega}_3(x) \qquad \text{y} \qquad \overrightarrow{\omega}_4(x) = \overrightarrow{\omega} 3(x) - \nabla q \tag{7.68}$$

es necesario utilizar, según sea el caso, aquel método numérico que proporcione una solución correcta y eficiente a la ecuación de Poisson, esto es especialmente importante cuando hay vórtices en el fluido.

8 Apéndice B: Expansión en Series de Taylor

Sea $f(x)$ una función definida en (a,b) que tiene hasta la $k-$ésima derivada, entonces la expansión de $f(x)$ usando series de Taylor alrededor del punto x_i contenido en el intervalo (a,b) será

$$f(x) = f(x_i) + \frac{(x-x_i)}{1!}\left.\frac{df}{dx}\right|_{x_i} + \frac{(x-x_i)^2}{2!}\left.\frac{d^2f}{dx^2}\right|_{x_i} + ... + \frac{(x-x_i)^k}{k!}\left.\frac{d^kf}{dx^k}\right|_{\varepsilon} \quad (8.1)$$

donde $\varepsilon = x_i + \theta(x-x_i)$ y $0 < \theta < 1$.

8.1 Aproximación de la Primera Derivada

Existen distintas formas de generar la aproximación a la primera derivada, nos interesa una que nos de la mejor precisión posible con el menor esfuerzo computacional.

8.1.1 Diferencias Progresivas

Considerando la Ec.(8.1) con $k = 2$ y $x = x_i + \Delta x$, tenemos

$$f(x_i + \Delta x) = f(x_i) + \Delta x \left.\frac{df}{dx}\right|_{x_i} + \frac{\Delta x^2}{2!}\left.\frac{d^2f}{dx^2}\right|_{\varepsilon_p} \quad (8.2)$$

de esta ecuación obtenemos la siguiente expresión para la aproximación de la primera derivada

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f(x_i+\Delta x) - f(x_i)}{\Delta x} - \frac{\Delta x}{2!}\left.\frac{d^2f}{dx^2}\right|_{\varepsilon_p} \quad (8.3)$$

en este caso la aproximación de $f'(x)$ mediante diferencias progresivas es de primer orden, es decir $O(\Delta x)$. Siendo $O_p(\Delta x)$ el error local de truncamiento, definido como

$$O_p(\Delta x) = -\frac{\Delta x}{2!}\left.\frac{d^2f}{dx^2}\right|_{\varepsilon_p}. \quad (8.4)$$

Es común escribir la expresión anterior como

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f(x_i+\Delta x) - f(x_i)}{\Delta x} - O_p(\Delta x) \quad (8.5)$$

o

$$f'(x_i) = \frac{f_{i+1} - f_i}{\Delta x} \quad (8.6)$$

para simplificar la notación.

8.1.2 Diferencias Regresivas

Considerando la Ec.(8.1) con $k = 2$ y $x = x_i - \Delta x$, tenemos

$$f(x_i - \Delta x) = f(x_i) - \Delta x \left.\frac{df}{dx}\right|_{x_i} + \frac{\Delta x^2}{2!} \left.\frac{d^2 f}{dx^2}\right|_{\varepsilon_r} \tag{8.7}$$

de esta ecuación obtenemos la siguiente expresión para la aproximación de la primera derivada

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f(x_i) - f(x_i - \Delta x)}{\Delta x} - \frac{\Delta x}{2!} \left.\frac{d^2 f}{dx^2}\right|_{\varepsilon_r} \tag{8.8}$$

en este caso la aproximación de $f'(x)$ mediante diferencias regresivas es de primer orden, es decir $O(\Delta x)$. Siendo $O_r(\Delta x)$ el error local de truncamiento, definido como

$$O_r(\Delta x) = \frac{\Delta x}{2!} \left.\frac{d^2 f}{dx^2}\right|_{\varepsilon_r}. \tag{8.9}$$

Es común escribir la expresión anterior como

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f(x_i) - f(x_i - \Delta x)}{\Delta x} + O_r(\Delta x) \tag{8.10}$$

o

$$f'(x_i) = \frac{f_i - f_{i-1}}{\Delta x} \tag{8.11}$$

para simplificar la notación.

8.1.3 Diferencias Centradas

Considerando la Ec.(8.1) con $k = 3$ y escribiendo $f(x)$ en $x = x_i + \Delta x$ y $x = x_i - \Delta x$, tenemos

$$f(x_i + \Delta x) = f(x_i) + \Delta x \left.\frac{df}{dx}\right|_{x_i} + \frac{\Delta x^2}{2!} \left.\frac{d^2 f}{dx^2}\right|_{x_i} + \frac{\Delta x^3}{3!} \left.\frac{d^3 f}{dx^3}\right|_{\varepsilon_p} \tag{8.12}$$

y

$$f(x_i - \Delta x) = f(x_i) - \Delta x \left.\frac{df}{dx}\right|_{x_i} + \frac{\Delta x^2}{2!} \left.\frac{d^2 f}{dx^2}\right|_{x_i} - \frac{\Delta x^3}{3!} \left.\frac{d^3 f}{dx^3}\right|_{\varepsilon_r} \tag{8.13}$$

restando la Ec.(8.12) de la Ec.(8.13), se tiene

$$f(x_i+\Delta x)-f(x_i-\Delta x)=2\Delta x\left.\frac{df}{dx}\right|_{x_i}+\frac{\Delta x^3}{3!}\left[\left.\frac{d^3f}{dx^3}\right|_{\varepsilon_p}+\left.\frac{d^3f}{dx^3}\right|_{\varepsilon_r}\right] \tag{8.14}$$

esta última expresión lleva a la siguiente aproximación de la primera derivada mediante diferencias centradas

$$\left.\frac{df}{dx}\right|_{x_i}=\frac{f(x_i+\Delta x)-f(x_i-\Delta x)}{2\Delta x}+O_c(\Delta x^2) \tag{8.15}$$

con un error local de truncamiento de segundo orden $O_c(\Delta x^2)$, es decir

$$O_c(\Delta x^2)=\frac{\Delta x^2}{3!}\left[\left.\frac{d^3f}{dx^3}\right|_{\varepsilon_p}+\left.\frac{d^3f}{dx^3}\right|_{\varepsilon_r}\right] \tag{8.16}$$

comparado el error local de truncamiento de la aproximación anterior $O_c(\Delta x^2)$, con los obtenidos previamente para diferencias progresivas $O_p(\Delta x)$ Ec.(8.5) y regresivas $O_r(\Delta x)$ Ec.(8.10), se tiene que

$$\lim_{\Delta x\to 0}O_c(\Delta x^2)<\lim_{\Delta x\to 0}O_p(\Delta x). \tag{8.17}$$

Es común encontrar expresada la derivada[73]

$$\left.\frac{df}{dx}\right|_{x_i}=\frac{f(x_i+\Delta x)-f(x_i-\Delta x)}{2\Delta x} \tag{8.18}$$

como

$$f'(x_i)=\frac{f_{i+1}-f_{i-1}}{2\Delta x} \tag{8.19}$$

para simplificar la notación.

8.2 Análisis de Convergencia

En el siguiente ejemplo se muestra la comparación de errores de truncamiento para los esquemas de aproximación de diferencias finitas progresiva, regresiva y centrada. Además gráfica el error y el orden estimado de convergencia, para la función Seno y su derivada exacta (Coseno) en el punto 1:

[73] En el caso de que la derivada sea usada en una malla no homogénea, es necesario incluir en la derivada a que Δx se refiere, por ejemplo en cada punto i, tenemos la Δx_{i-} (por la izquierda) y la Δx_{i+} (por la derecha), i.e. $\left.\frac{df}{dx}\right|_{x_i}=\frac{f(x_i+\Delta x_{i-})-f(x_i-\Delta x_{i+})}{(\Delta x_{i-})+(\Delta x_{i+})}$.

Ejemplo 34 *% u(x) = sin(x) en x=1. Derivada exacta u'(1) = cos(1)*

```
clear; close all
h=1.0;
for k = 1:33,
a(k,1) = h;
a(k,2) = (sin(1 + h) - sin(1)) / h - cos(1);
a(k,3) = (sin(1) - sin(1 - h)) / h - cos(1);
a(k,4) = (sin(1 + h) - sin(1 - h)) / (2 * h) - cos(1);
h = h / 2;
end

format short e
a % Visualiza el resultado

a = abs(a); % Toma el valor absoluto
h1 = a(:,1); % Extrae la primer columna que es la de h
e1 = a(:,2); e2 = a(:,3); e3 = a(:,4);

loglog(h1, e1, h1, e2, h1, e3)
axis('square')
axis([1e-6 1e1 1e-6 1e1])
gtext('Esperada de adelante y atras =1')
gtext('Esperada de central = 2')
```

Generado la siguiente salida (La primera columna es h, las demás son los errores para los esquemas de aproximación de diferencias finitas progresiva, regresiva y centrada respectivamente):

```
1.0000e+00 -4.7248e-01 3.0117e-01 -8.5654e-02
5.0000e-01 -2.2825e-01 1.8379e-01 -2.2233e-02
2.5000e-01 -1.1025e-01 9.9027e-02 -5.6106e-03
1.2500e-01 -5.3929e-02 5.1118e-02 -1.4059e-03
6.2500e-02 -2.6639e-02 2.5936e-02 -3.5169e-04
3.1250e-02 -1.3235e-02 1.3059e-02 -8.7936e-05
1.5625e-02 -6.5958e-03 6.5519e-03 -2.1985e-05
7.8125e-03 -3.2925e-03 3.2815e-03 -5.4962e-06
3.9062e-03 -1.6449e-03 1.6421e-03 -1.3741e-06
1.9531e-03 -8.2209e-04 8.2141e-04 -3.4351e-07
```

9.7656e-04 -4.1096e-04 4.1079e-04 -8.5879e-08
4.8828e-04 -2.0546e-04 2.0542e-04 -2.1470e-08
2.4414e-04 -1.0272e-04 1.0271e-04 -5.3673e-09
1.2207e-04 -5.1361e-05 5.1358e-05 -1.3417e-09
6.1035e-05 -2.5680e-05 2.5679e-05 -3.3579e-10
3.0518e-05 -1.2840e-05 1.2840e-05 -8.3860e-11
1.5259e-05 -6.4199e-06 6.4199e-06 -2.2014e-11
7.6294e-06 -3.2100e-06 3.2100e-06 -1.8607e-13
3.8147e-06 -1.6050e-06 1.6050e-06 7.0899e-12
1.9073e-06 -8.0249e-07 8.0247e-07 -7.4620e-12
9.5367e-07 -4.0120e-07 4.0125e-07 2.1642e-11
4.7684e-07 -2.0062e-07 2.0055e-07 -3.6566e-11
2.3842e-07 -1.0050e-07 1.0020e-07 -1.5298e-10
1.1921e-07 -4.9746e-08 4.9906e-08 7.9849e-11
5.9605e-08 -2.5532e-08 2.4760e-08 -3.8581e-10
2.9802e-08 -1.0630e-08 1.1721e-08 5.4551e-10
1.4901e-08 -6.9051e-09 7.9961e-09 5.4551e-10
7.4506e-09 5.4551e-10 5.4551e-10 5.4551e-10
3.7253e-09 5.4551e-10 5.4551e-10 5.4551e-10
1.8626e-09 -2.9257e-08 -2.9257e-08 -2.9257e-08
9.3132e-10 -2.9257e-08 -2.9257e-08 -2.9257e-08
4.6566e-10 -2.9257e-08 -2.9257e-08 -2.9257e-08
2.3283e-10 -2.9257e-08 -2.9257e-08 -2.9257e-08

Cómo se nota en la anterior salida, la convergencia mejora conforme h se hace pequeña, hasta llegar a cierto valor en que por errores de truncamiento, el error en los esquemas de aproximación en la derivada comienza a incrementarse. Esto nos da una idea de los valores de h que podemos usar para los esquemas de aproximación de diferencias finitas progresiva, regresiva y centrada respectivamente.

Derivadas Usando Más Puntos Utilizando el valor de la función en más puntos se construyen fórmulas más precisas para las derivadas[74], algunos

[74] Al usar estas derivadas en el método de diferencias finitas mostrado en la sección (7.1) las matrices generadas no serán tridiagonales.

ejemplos son

$$\begin{aligned} f'(x_i) &= \frac{-3f_i + 4f_{i+1} - f_{i+2}}{2\Delta x} + O(\Delta x^2) \qquad (8.20)\\ f'(x_i) &= \frac{3f_i - 4f_{i-1} + f_{i-2}}{2\Delta x} + O(\Delta x^2)\\ f'(x_i) &= \frac{2f_{i+1} + 3f_i - 6f_{i-1} + f_{i-2}}{6\Delta x} + O(\Delta x^3)\\ f'(x_i) &= \frac{f_{i-2} - 8f_{i-1} + 8f_{i+1} - f_{i+2}}{12\Delta x} + O(\Delta x^4)\\ f'(x_i) &= \frac{-25f_i + 48f_{i+1} - 36f_{i+2} + 16f_{i+3} - 3f_{i+4}}{12\Delta x} + O(\Delta x^4) \end{aligned}$$

8.3 Derivadas de Ordenes Mayores

De forma análoga se construyen aproximaciones en diferencias finitas de orden mayor, aquí desarrollaremos la forma de calcular la derivada de orden dos en diferencias centradas.

8.3.1 Derivada de Orden Dos

Partiendo del desarrollo de Taylor

$$f(x_i + \Delta x) = f(x_i) + \Delta x f'(x_i) + \frac{\Delta x^2}{2!} f''(x_i) + \frac{\Delta x^3}{3!} f'''(x_i) + \frac{\Delta x^4}{4!} f^{(4)}(\xi_p) \qquad (8.21)$$

y

$$f(x_i - \Delta x) = f(x_i) - \Delta x f'(x_i) + \frac{\Delta x^2}{2!} f''(x_i) - \frac{\Delta x^3}{3!} f'''(x_i) + \frac{\Delta x^4}{4!} f^{(4)}(\xi_r) \qquad (8.22)$$

eliminando las primeras derivadas, sumando las ecuaciones anteriores y despejando se encuentra que

$$f''(x_i) = \frac{f(x_i - \Delta x) - 2f(x_i) + f(x_i + \Delta x)}{\Delta x^2} - \frac{\Delta x^2}{12} f^{(4)}(\xi_c) \qquad (8.23)$$

así, la aproximación a la segunda derivada usando diferencias centradas con un error de truncamiento $O_c(\Delta x^2)$ es[75]

$$f''(x_i) = \frac{f(x_i - \Delta x) - 2f(x_i) + f(x_i + \Delta x)}{\Delta x^2} \tag{8.24}$$

Es común escribir la expresión anterior como

$$f''(x_i) = \frac{f_{i-1} - 2f_i + f_{i+1}}{\Delta x^2}$$

para simplificar la notación.

Derivadas Usando Más Puntos Utilizando el valor de la función en más puntos se construyen fórmulas más precisas para las derivadas, algunos ejemplos son

$$\begin{aligned} f''(x_i) &= \frac{f_{i+2} - 2f_{i+1} + f_i}{\Delta x^2} + O(\Delta x) \\ f''(x_i) &= \frac{-f_{i+3} + 4f_{i+2} - 5f_{i+1} + 2f_i}{\Delta x^2} + O(\Delta x^2) \\ f''(x_i) &= \frac{-f_{i+2} + 16f_{i+1} - 30f_i + 16f_{i-1} - f_{i-2}}{12\Delta x^2} + O(\Delta x^4) \end{aligned} \tag{8.25}$$

8.3.2 Derivadas de Orden Tres y Cuatro

De forma análoga se construyen derivadas de órdenes mayores utilizando el valor de la función en más puntos, algunos ejemplos para terceras derivadas son

$$\begin{aligned} f'''(x_i) &= \frac{f_{i+3} - 3f_{i+2} + 3f_{i+1} - f_i}{\Delta x^3} + O(\Delta x) \\ f'''(x_i) &= \frac{f_{i+2} - 2f_{i+1} + 2f_{i-1} - f_{i-2}}{2\Delta x^3} + O(\Delta x^2) \\ f'''(x_i) &= \frac{f_{i-3} - 8f_{i-2} + 13f_{i-1} - 13f_{i+1} + 8f_{i+2} - f_{i+3}}{8\Delta x^3} + O(\Delta x^4) \end{aligned} \tag{8.26}$$

[75] En el caso de que la derivada sea usada en una malla no homogénea, es necesario incluir en la derivada a que Δx se refiere, por ejemplo en cada punto i, tenemos la Δx_{i-} (por la izquierda) y la Δx_{i+} (por la derecha), i.e. $f''(x_i) = \frac{f(x_i - \Delta x_{i-}) - 2f(x_i) + f(x_i + \Delta x_{i+})}{(\Delta x_{i-})(\Delta x_{i+})}$

Algunos ejemplos para cuartas derivadas son

$$
\begin{aligned}
f''''(x_i) &= \frac{f_{i+4} - 4f_{i+3} + 6f_{i+2} - 4f_{i+1} + f_i}{\Delta x^4} + O(\Delta x) \qquad (8.27)\\
f''''(x_i) &= \frac{f_{i+2} - 4f_{i+1} + 6f_i - 4f_{i-1} + f_{i-2}}{\Delta x^4} + O(\Delta x^2)\\
f''''(x_i) &= \frac{-f_{i+3} + 12f_{i+2} - 39f_{i+1} + 56f_i - 39f_{i-1} + 12f_{i-2} - f_{i-3}}{6\Delta x^4} + O(\Delta x^4)
\end{aligned}
$$

8.4 Derivadas en Dos y Tres Dimensiones

De forma análoga, se construyen aproximaciones en diferencias finitas de primer y segundo orden en dos y tres dimensiones.

8.4.1 Derivadas en Dos Dimensiones

Usando el teorema de Taylor para funciones en dos variables x y y, es posible escribir de forma exacta para el punto x_i y y_j

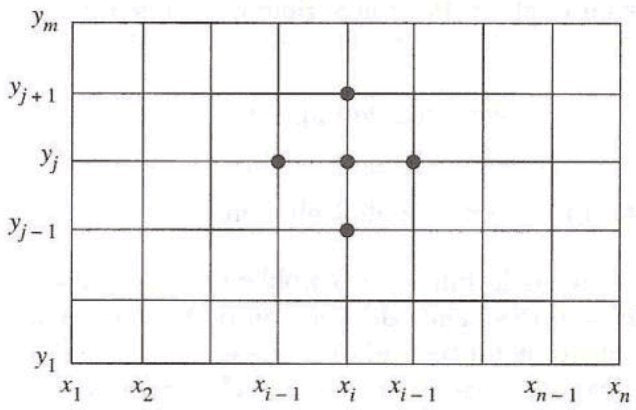


Figura 2: Discretización en un rectángulo.

$$
\begin{aligned}
f(x_i + \Delta x, y_j) &= f(x_i, y_j) + \Delta x \frac{\partial f\,(x_i, y_j)}{\partial x} + \frac{\Delta x}{2} \frac{\partial^2 f\,(x_i + \theta_1 \Delta x, y_j)}{\partial x^2} \qquad (8.28)\\
f(x_i, y_j + \Delta y) &= f(x_i, y_j) + \Delta y \frac{\partial f\,(x_i, y_j)}{\partial y} + \frac{\Delta y}{2} \frac{\partial^2 f\,(x_i, y_j + \theta_2 \Delta y)}{\partial y^2}.
\end{aligned}
$$

Así, la aproximación en diferencias hacia adelante de $\partial f/\partial x$ y $\partial f/\partial y$ es

$$\begin{array}{rcl} \dfrac{\partial f\left(x_i,y_j\right)}{\partial x} & \simeq & \dfrac{f\left(x_i+\Delta x,y_j\right)-f\left(x_i,y_j\right)}{\Delta x} \\ \dfrac{\partial f\left(x_i,y_j\right)}{\partial y} & \simeq & \dfrac{f\left(x_i,y_j+\Delta y\right)-f\left(x_i,y_j\right)}{\Delta y} \end{array} \tag{8.29}$$

o en su forma simplificada (asociamos $\Delta x = h$ y $\Delta y = k$), entonces tenemos

$$\begin{array}{rcl} \dfrac{\partial f\left(x_i,y_j\right)}{\partial x} & \simeq & \dfrac{f_{i+1,j}-f_{i,j}}{h} \\ \dfrac{\partial f\left(x_i,y_j\right)}{\partial y} & \simeq & \dfrac{f_{i,j+1}-f_{i,j}}{k}. \end{array} \tag{8.30}$$

La aproximación en diferencias hacia atrás de $\partial f/\partial x$ y $\partial f/\partial y$ es

$$\begin{array}{rcl} \dfrac{\partial f\left(x_i,y_j\right)}{\partial x} & \simeq & \dfrac{f\left(x_i,y_j\right)-f\left(x_i-\Delta x,y_j\right)}{\Delta x} \\ \dfrac{\partial f\left(x_i,y_j\right)}{\partial y} & \simeq & \dfrac{f\left(x_i,y_j\right)-f\left(x_i,y_j-\Delta y\right)}{\Delta y} \end{array} \tag{8.31}$$

o en su forma simplificada tenemos

$$\begin{array}{rcl} \dfrac{\partial f\left(x_i,y_j\right)}{\partial x} & \simeq & \dfrac{f_{i,j}-f_{i-1,j}}{h} \\ \dfrac{\partial f\left(x_i,y_j\right)}{\partial y} & \simeq & \dfrac{f_{i,j}-f_{i,j-1}}{k}. \end{array} \tag{8.32}$$

La aproximación en diferencias centradas de $\partial f/\partial x$ y $\partial f/\partial y$ es

$$\begin{array}{rcl} \dfrac{\partial f\left(x_i,y_j\right)}{\partial x} & \simeq & \dfrac{f\left(x_i+\Delta x,y_j\right)-f\left(x_i-\Delta x,y_j\right)}{2\Delta x} \\ \dfrac{\partial f\left(x_i,y_j\right)}{\partial y} & \simeq & \dfrac{f\left(x_i,y_j+\Delta y\right)-f\left(x_i,y_j-\Delta y\right)}{2\Delta y} \end{array} \tag{8.33}$$

o en su forma simplificada tenemos

$$\begin{array}{rcl} \dfrac{\partial f\left(x_i,y_j\right)}{\partial x} & \simeq & \dfrac{f_{i+1,j}-f_{i-1,j}}{2h} \\ \dfrac{\partial f\left(x_i,y_j\right)}{\partial y} & \simeq & \dfrac{f_{i,j+1}-f_{i,j-1}}{2k}. \end{array} \tag{8.34}$$

Por otro lado, la aproximación en diferencias centradas de $\partial^2 f/\partial x^2$ y $\partial^2 f/\partial y^2$ es

$$\begin{aligned} \frac{\partial^2 f(x_i, y_j)}{\partial x^2} &\simeq \frac{f(x_i - \Delta x, y_j) - 2f(x_i, y_j) + f(x_i + \Delta x, y_j)}{\Delta x^2} \\ \frac{\partial^2 f(x_i, y_j)}{\partial y^2} &\simeq \frac{f(x_i, y_j - \Delta y) - 2f(x_i, y_j) + f(x_i, y_j + \Delta y)}{\Delta y^2} \end{aligned} \tag{8.35}$$

o en su forma simplificada tenemos

$$\begin{aligned} \frac{\partial^2 f(x_i, y_j)}{\partial x^2} &\simeq \frac{f_{i-1,j} - 2f_{i,j} + f_{i+1,j}}{h^2} \\ \frac{\partial^2 f(x_i, y_j)}{\partial y^2} &\simeq \frac{f_{i,j-1} - 2f_{i,j} + f_{i,j+1}}{k^2}. \end{aligned} \tag{8.36}$$

8.4.2 Derivadas en Tres Dimensiones

Usando el teorema de Taylor para funciones de tres variables x, y y z, es posible escribir de forma exacta para el punto x_i, y_j y z_k

$$f(x_i + \Delta x, y_j, z_k) = f(x_i, y_j, z_k) + \Delta x \frac{\partial f(x_i, y_j, z_k)}{\partial x} + \frac{\Delta x}{2} \frac{\partial^2 f(x_i + \theta_1 \Delta x, y_j, z_k)}{\partial x^2} \tag{8.37}$$

$$f(x_i, y_j + \Delta y, z_k) = f(x_i, y_j, z_k) + \Delta y \frac{\partial f(x_i, y_j, z_k)}{\partial y} + \frac{\Delta y}{2} \frac{\partial^2 f(x_i, y_j + \theta_2 \Delta y, z_k)}{\partial y^2}$$

$$f(x_i, y_j, z_k + \Delta z) = f(x_i, y_j, z_k) + \Delta z \frac{\partial f(x_i, y_j, z_k)}{\partial z} + \frac{\Delta z}{2} \frac{\partial^2 f(x_i, y_j, z_k + \theta_3 \Delta z)}{\partial z^2}$$

Así, la aproximación en diferencias hacia adelante de $\partial f/\partial x, \partial f/\partial y$ y $\partial f/\partial z$ es

$$\begin{aligned} \frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f(x_i + \Delta x, y_j, z_k) - f(x_i, y_j, z_k)}{\Delta x} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f(x_i, y_j + \Delta y, z_k) - f(x_i, y_j, z_k)}{\Delta y} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f(x_i, y_j, z_k + \Delta z) - f(x_i, y_j, z_k)}{\Delta z} \end{aligned} \tag{8.38}$$

o en su forma simplificada (para simplificar la notación, asociamos $\Delta x =$

$h, \Delta y = l$ y $\Delta z = m$), tenemos

$$\begin{aligned} \frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f_{i+1,j,k} - f_{i,j,k}}{h} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f_{i,j+1,k} - f_{i,j,k}}{l} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f_{i,j,k+1} - f_{i,j,k}}{m}. \end{aligned} \tag{8.39}$$

La aproximación en diferencias hacia atrás de $\partial f/\partial x, \partial f/\partial y$ y $\partial f/\partial z$ es

$$\begin{aligned} \frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f(x_i, y_j, z_k) - f(x_i - \Delta x, y_j, z_k)}{\Delta x} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f(x_i, y_j, z_k) - f(x_i, y_j - \Delta y, z_k)}{\Delta y} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f(x_i, y_j, z_k) - f(x_i, y_j, z_k - \Delta z)}{\Delta z} \end{aligned} \tag{8.40}$$

o en su forma simplificada tenemos

$$\begin{aligned} \frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f_{i,j,k} - f_{i-1,j,k}}{h} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f_{i,j,k} - f_{i,j-1,k}}{l} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f_{i,j,k} - f_{i,j,k-1}}{m}. \end{aligned} \tag{8.41}$$

La aproximación en diferencias centradas de $\partial f/\partial x, \partial f/\partial y$ y $\partial f/\partial z$ es

$$\begin{aligned} \frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f(x_i + \Delta x, y_j, z_k) - f(x_i - \Delta x, y_j, z_k)}{2\Delta x} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f(x_i, y_j + \Delta y, z_k) - f(x_i, y_j - \Delta y, z_k)}{2\Delta y} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f(x_i, y_j, z_k + \Delta z) - f(x_i, y_j, z_k - \Delta z)}{2\Delta z} \end{aligned} \tag{8.42}$$

o en su forma simplificada tenemos

$$\begin{aligned} \frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f_{i+1,j,k} - f_{i-1,j,k}}{2h} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f_{i,j+1,k} - f_{i,j-1,k}}{2l} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f_{i,j,k+1} - f_{i,j,k-1}}{2m}. \end{aligned} \tag{8.43}$$

Por otro lado, la aproximación en diferencias centradas de $\partial^2 f/\partial x^2, \partial^2 f/\partial y^2$ y $\partial^2 f/\partial z^2$ es

$$\begin{aligned}
\frac{\partial^2 f(x_i, y_j, z_k)}{\partial x^2} &\simeq \frac{f(x_i - \Delta x, y_j, z_k) - 2f(x_i, y_j, z_k) + f(x_i + \Delta x, y_j, z_k)}{\Delta x^2} \\
\frac{\partial^2 f(x_i, y_j, z_k)}{\partial y^2} &\simeq \frac{f(x_i, y_j - \Delta y, z_k) - f(x_i, y_j, z_k) + f(x_i, y_j + \Delta y, z_k)}{\Delta y^2} \\
\frac{\partial^2 f(x_i, y_j, z_k)}{\partial z^2} &\simeq \frac{f(x_i, y_j, z_k - \Delta z) - f(x_i, y_j, z_k) + f(x_i, y_j, z_k + \Delta z)}{\Delta z^2}
\end{aligned} \tag{8.44}$$

o en su forma simplificada tenemos

$$\begin{aligned}
\frac{\partial^2 f(x_i, y_j, z_k)}{\partial x^2} &\simeq \frac{f_{i-1,j,k} - 2f_{i,j,k} + f_{i+1,j,k}}{h^2} \\
\frac{\partial^2 f(x_i, y_j, z_k)}{\partial y^2} &\simeq \frac{f_{i,j-1,k} - 2f_{i,j,k} + f_{i,j+1,k}}{l^2} \\
\frac{\partial^2 f(x_i, y_j, z_k)}{\partial z^2} &\simeq \frac{f_{i,j,k-1} - 2f_{i,j,k} + f_{i,j,k+1}}{m^2}.
\end{aligned} \tag{8.45}$$

9 Apéndice C: El Cómputo en Paralelo

La computación paralela es el uso de múltiples recursos computacionales para resolver un problema. Se distingue de la computación secuencial en que varias operaciones pueden ocurrir simultáneamente. Los sistemas de cómputo con procesamiento en paralelo surgen de la necesidad de resolver problemas complejos en un tiempo razonable, utilizando las ventajas de memoria, velocidad de los procesadores, formas de interconexión de estos y distribución de la tarea, a los que en su conjunto denominamos arquitectura en paralelo. Entenderemos por una arquitectura en paralelo a un conjunto de procesadores interconectados capaces de cooperar en la solución de un problema.

Así, para resolver un problema en particular, se usa una arquitectura o combinación de múltiples arquitecturas (topologías), ya que cada una ofrece ventajas y desventajas que tienen que ser sopesadas antes de implementar la solución del problema en una arquitectura en particular. También es necesario conocer los problemas a los que se enfrenta un programador que desea que su programa corra en paralelo, como son: el partir eficientemente[76] un problema en múltiples subtareas y cómo distribuir eficazmente[77] estas según la arquitectura en particular con que se trabaje. La eficacia difiere de la eficiencia en el sentido de que la eficiencia hace referencia en la mejor utilización de los recursos computacionales (procesadores), en tanto que la eficacia hace referencia en la capacidad para alcanzar un objetivo, aunque en el proceso no se haya hecho el mejor uso de los recursos computacionales.

El cómputo de alto rendimiento (HPC por High-Performance Computing) está formado por un conjunto de computadoras unidas entre sí en forma de Cluster[78], para aumentar su potencia de trabajo y rendimiento. En 2019 la supercomputadora *SUMMIT* de IBM funcionaban a más de 148 peta-Flops[79] (cada uno de ellos equivale a la realización de más de 1000 billones

[76] Es un indicador de la utilización efectiva de los diferentes recursos computacionales (principalmente del uso de los procesadores) en relación al algoritmo dado. Se entiende que la eficacia se da cuando se utilizan menos recursos para lograr un mismo objetivo. O al contrario, cuando se logran más objetivos con los mismos o menos recursos.

[77] La podemos definir como el nivel de consecución de metas y objetivos. La eficacia hace referencia a nuestra capacidad de lograr lo que no proponemos.

[78] Existe el Ranking de las 500 supercomputadoras más poderosas del mundo (esta se actualiza cada seis meses en junio y noviembre) y puede ser consultada en:
https://top500.org

[79] "FLOPS" operaciones de punto flotante por segundo. Describe una velocidad de procesamiento teórica: para hacer posible esa velocidad es necesario enviar datos a los

de operaciones por segundo de las pruebas de rendimiento del HPC-AI), mientras que en junio de 2020 *FUGAKU* de Japón superaba los 415.5 peta-Flops y en noviembre del mismo año alcanzó los 2.0 exa-Flops, siendo este el primer equipo en alcanzar valores por arriba de un exa-Flops para cualquier precisión sobre cualquier tipo de Hardware, manteniéndose en el primer lugar del top 500 durante el 2021

El clásico uso del paralelismo, es el de diseño de programas eficientes en el ámbito científico. La simulación de problemas científicos es un área de gran importancia, los cuales requieren de una gran capacidad de procesamiento y de espacio de memoria, debido a las complejas operaciones que se deben realizar.

Otro uso clásico es el de las gráficas y vídeos de simulaciones generadas por computadora. La generación de fotogramas y videos requiere de una gran cantidad de cálculos matemáticos. Esto supone una tarea muy compleja para un solo procesador, luego es necesario que haya algún tipo de paralelismo, para distribuir la tarea para que esta sea realizada eficiente y eficazmente.

Tradicionalmente, los programas informáticos se han escrito para el cómputo en secuencial. Para resolver un problema, se construye un algoritmo y se implementa como un flujo en serie de instrucciones. Estas instrucciones se ejecutan en una unidad central de procesamiento en un ordenador. Sólo puede ejecutarse una instrucción a la vez y un tiempo después de que la instrucción ha terminado, se ejecuta la siguiente.

Actualmente, es una práctica común usar directivas de compilación en equipos paralelos sobre programas escritos de forma secuencial, con la esperanza que sean puestos por el compilador como programas paralelos. Esto en la gran mayoría de los casos genera códigos poco eficientes, pese a que corren en equipos paralelos y pueden usar toda la memoria compartida de dichos equipos, el algoritmo ejecutado continúa siendo secuencial en una gran parte del código.

La computación en paralelo, por el contrario, utiliza simultáneamente múltiples elementos de procesamiento para resolver un problema. Esto se logra mediante la división del problema en partes independientes de modo que cada elemento de procesamiento pueda ejecutar su parte del algoritmo de manera simultánea con los otros. Los elementos de procesamiento son di-

procesadores de forma continua. Por lo tanto, el procesamiento de los datos se debe tener en cuenta en el diseño del sistema. La memoria del sistema, junto con las interconexiones que unen los nodos de procesamiento entre sí, impactan en la rapidez con la que los datos llegan a los procesadores.

versos e incluyen recursos tales como una computadora con múltiples procesadores, varios ordenadores en red, Hardware especializado, o cualquier combinación de los anteriores.

Los programas informáticos paralelos son más difíciles de escribir que los secuenciales, porque la concurrencia introduce nuevos tipos de errores de Software, siendo las condiciones de sincronización las más comunes. La comunicación y sincronización entre diferentes subtareas son algunos de los mayores obstáculos para obtener un buen rendimiento del programa paralelo.

¿Cómo Paralelizo mi programa? El problema de todos los usuarios de Ciencias e Ingeniería que hacen uso de métodos numéricos y su implementación computacional es: ¿cómo paralelizo mi programa?, esta pregunta no tiene una respuesta simple, depende de muchos factores, por ejemplo: en que lenguaje o paquete está desarrollado el programa, el algoritmo usado para solucionar el problema, a que equipos paralelo tengo acceso y un largo etc. Algunas respuestas ingenuas podrían ser:

- Si uso lenguajes de programación compilables, puedo conseguir un compilador que permita usar directivas de compilación en equipos de memoria compartida sobre programas escritos de forma secuencial, con la esperanza que sean puestos por el compilador como programas paralelos haciendo uso de hilos, OpenMP y optimización del código generado.

- Si uso paquetes como MatLab, Julia o Python, es posible conseguir una versión del paquete que implemente bibliotecas que usen CUDAs, OpenMP o MPI para muchos de los algoritmos más usados en la programación.

- Si mi problema cabe en un sólo equipo, entonces puedo usar dos o más cores para resolver mi problema usando memoria compartida (puedo usar OpenMP, trabajar con hilos o usando paquetes como MatLab o lenguajes como Python, Julia, Fortran, C y C++, etc.), pero sólo puedo escalar para usar el máximo número de cores de un equipo (que en la actualidad puede ser del orden de 128 cores, hasta Terabytes de RAM y con almacenamiento de algunas decenas de Terabytes).

- Si mi problema cabe en la GRAM de una GPU, entonces el posible usar una tarjeta gráfica (usando paquetes como MatLab o lenguajes como

Python, Julia, Fortran, C y C++, etc.), pero sólo puedo escalar a la capacidad de dichas tarjetas gráficas.

- Puedo usar un cluster que usa memoria distribuida-compartida en conjunto con tarjetas gráficas, este tipo de programación requiere una nueva expertes y generalmente implica el uso de paso de mensajes como MPI y rediseño de los algoritmos usados en nuestro programa.

Notemos primero, que no todos los algoritmos son paralelizables. En cualquier caso se tienen que ver los pros y contras de la paralelización para cada caso particular. Pero es importante destacar que existen una gran cantidad de bibliotecas y paquetes que ya paralelizan ciertos algoritmos que son ampliamente usados como la solución de sistemas lineales[80] y no lineales. Además, el tiempo de programación necesario para desarrollar una aplicación paralela eficiente y eficaz para la gran mayoría de los programadores puede ser de semanas o meses en el mejor de los casos. Por ello, es necesario hacer un balance entre las diferentes opciones de paralelización para no invertir un tiempo precioso que puede no justificar dicha inversión económica, de recursos computacionales y sobre todo de tiempo.

Antes de iniciar con los tópicos del cómputo en paralelo, es necesario conocer cómo está constituida nuestra herramienta de trabajo: la computadora. Los conocimientos básicos de arquitectura de las computadoras nos permite identificar los componentes de Hardware que limitarán nuestros esfuerzos por realizar una paralelización óptima de nuestro programa y en muchos casos descubriremos de la peor manera que los algoritmos usados en nuestro programa no son los más adecuados para paralelizar; y descubriremos con horror que en algunos casos los tiempos de ejecución no mejoran sin importar cuantos equipos adicionales usemos, en el peor de los casos, el tiempo de ejecución se incrementa al aumentar equipos en vez de disminuir.

[80] Algunas de las bibliotecas más usadas para resolver sistemas lineales usando matrices bandadas y dispersar son PETCs, HYPRE, ATLAS, LAPACK++, LAPACK, EISPACK, LINPACK, BLAS, entre muchas otras alternativas, tanto para implementaciones secuenciales como paralelas y más recientemente para hacer uso de los procesadores CUDA en las GPU de nVidia.

9.1 ¿Que es Computación Paralela?

En el sentido más simple, la computación paralela es el uso simultáneo de múltiples recursos computacionales para resolver un problema computacional:

- Un problema se divide en partes discretas que se pueden resolver simultáneamente
- Cada parte se descompone en una serie de instrucciones
- Las instrucciones de cada parte se ejecutan simultáneamente en diferentes procesadores
- Se emplea un mecanismo global de control/coordinación

¿Por qué se hace programación paralela? El hecho de que la programación paralela sea un paradigma da cuenta de que existe una razón por la cual no ha dejado de ser necesaria o no ha sido totalmente automatizable, igualmente hay otras razones interesantes detrás para entender la existencia, actualidad y contemporaneidad de la programación paralela:

- Ley de Moore: Esta ley propuesta por Gordon E. Moore en 1965 dice resumidamente que el número de transistores en un Chip determinado se doblaría cada dos años. Esto quiere decir un aumento del rendimiento en los procesadores del alrededor del 50%, esto se traduce en escalar la velocidad de reloj de los procesadores, pero esta ley no es fidedigna desde el 2002 dónde solo ha habido un 20%, lo cuál sigue siendo un aumento considerable, sin embargo, no sería suficiente para que todos los avances en computación que se han logrado hasta el día y las necesidades de procesamiento en crecimiento exponencial puedan satisfacerse totalmente.
- Overclocking infinito: El Overclokcing tiene un límite a pesar de que existiera una refrigeración perpetúa y adecuada del procesador. Esto es debido a las corrientes parásitas que impiden una velocidad teóricamente infinita a la cual los circuitos pueden cambiar entre estados, o de hecho sus transistores.

- Automatización del paralelismo: Se dice en este paradigma que el éxito es inversamente proporcional al número de cores precisamente porque existen complejidades en el corazón del paralelismo que implican cosas que todavía no se pueden predecir ni con inteligencia artificial, se menciona cuales son las posibles estrategias para atacar un problema de forma paralela, esto da cuenta de que existe una forma prácticamente determinada de abordarlos pero no de automatizarlos, a pesar de que sí existan algunas partes que son automatizables en el proceso.
- Solución en el Hardware: Un diseño adecuado del Hardware permitiría que la paralelización siempre estuviera presente con respecto a los procesadores que se están usando de tal modo que alguno los problemas que son inherentes al paradigma pudieran evitarse. Esto ha resultado imposible hasta la fecha, de hecho, solo diseñar solamente algo tan efectivo y tradicional como se ha hecho en programación secuencial es algo que no existe hasta ahora. Existen algunas aproximaciones como OpenMP y programación por hilos de las que hablaremos más adelante.

Ventajas

- Resuelve problemas que no se podrían realizar en una sola CPU
- Resuelve problemas que no se pueden resolver en un tiempo razonable
- Permite ejecutar problemas de un orden y complejidad mayor
- Permite ejecutar código de manera más rápida (aceleración)
- Permite ejecutar en general más problemas
- Obtención de resultados en menos tiempo
- Permite la ejecución de varias instrucciones en simultáneo
- Permite dividir una tarea en partes independientes

Desventajas

- Mayor consumo de energía
- Mayor dificultad a la hora de escribir programas

- Dificultad para lograr una buena sincronización y comunicación entre las tareas
- Retardos ocasionados por comunicación entre tareas
- Número de componentes usados es directamente proporcional a los fallos potenciales
- Condiciones de carrera
  - Múltiples procesos se encuentran en condición de carrera si el resultado de los mismos depende del orden de su llegada
  - Si los procesos que están en condición de carrera no son correctamente sincronizados, puede producirse una corrupción de datos

Paralelismo de Grano Fino, Grano Grueso y Paralelismo Vergonzoso las aplicaciones a menudo se clasifican según la frecuencia con que sus subtareas se sincronizan o comunican entre sí. Una aplicación muestra un paralelismo de grano fino si sus subtareas deben comunicarse muchas veces por segundo, se considera paralelismo de grano grueso si no se comunican muchas veces por segundo, y es vergonzosamente paralelo si nunca o casi nunca se tienen que comunicar. Aplicaciones vergonzosamente paralelas son consideradas las más fáciles de paralelizar.

El tipo de problemas complejos que aborda el cómputo de alto rendimiento (HPC por High-Performance Computing) no se pueden resolver con una computadora de escritorio y tienen una escala determinada: toma mucho tiempo resolverlos, se necesita una gran cantidad de memoria (RAM), se tienen que realizar muchísimos experimentos parecidos y hay restricciones concretas de tiempo para encontrar resultados. La mayoría de estas temáticas están relacionadas con la innovación en la industria y su aplicación en áreas como energía, medio ambiente, biotecnología, medicina, ingeniería, etc.

A su vez, la evolución del procesador la cantidad de transistores sigue creciendo pero la velocidad de los núcleos individuales (core) casi no aumenta. Podríamos comprar un procesador de 128 núcleos pero el desafío de aprovechar la potencia de un procesador es cómo distribuimos el tiempo de ejecución de una misma tarea computacional.

De este modo, cada tarea computacional se divide en dos partes: serial y paralelizable. Como todas las aplicaciones tienen una parte secuencial, el

tiempo de cómputo de la aplicación paralela está acotado por esa sección serial (Ley de Ahmdahl). Esto implica dos cuestiones fundamentales:

1. Es necesario reducir lo más posible el tiempo de ejecución serial
2. En una tarea computacional existe un máximo de procesadores que conviene poner a trabajar en forma paralela, más allá de ese punto no se justifica seguir paralelizando la tarea.

¿Qué debe tener en cuenta un centro de HPC para aprovechar al máximo los recursos de cómputo? Cada centro de HPC tiene que configurar, optimizar aplicaciones y sistemas operativos. Allí el monitoreo es clave, con la escala de las infraestructuras también surgen aplicaciones con problemas de memoria (cuello de botella) y la necesidad de utilizar mejores redes (PARAMNet-3, Infiniband, GigE). Tener poder de cómputo no es barato y solamente con el Hardware no alcanza. El equipo de operaciones de HPC necesita un alto grado de conocimiento y de disciplina para encarar la tarea.

Con el uso del cómputo de alto desempeño es posible por ejemplo reducir el tiempo que nos lleva descubrir nuevos medicamentos de años a meses. Una computadora de alto rendimiento es capaz de resolver este y otros tipos de problemas científicos avanzados mediante simulaciones, modelos y análisis. Estos sistemas abren las puertas de la "Cuarta Revolución Industrial", ya que ayudan a resolver muchas de las problemáticas más importantes del mundo. Los sistemas HPC ya se utilizan para las siguientes tareas:

- Descubrir nuevos componentes de drogas y probar los conocidos para combatir diferentes tipos de cáncer y otras enfermedades
- Simular dinámicas moleculares para crear nuevos materiales, como tejidos balísticos
- Pronosticar cambios climáticos considerables para mejorar la preparación de las comunidades afectadas

Las supercomputadoras representan lo último en sistemas HPC. La definición de supercomputadora depende de diferentes estándares que van cambiando a medida que las capacidades evolucionan. Un solo clúster de supercomputadoras puede incluir decenas de miles de procesadores, y los sistemas más caros y potentes del mundo pueden costar más de US$ 100 millones de dólares.

Casos de Uso Emergentes a medida que las tecnologías mejoraron, la computación de alto rendimiento comenzó a implementarse en una gama de capacidades más amplia. Ahora contamos con más potencia de procesamiento y memoria que nunca para solucionar problemas más complejos.

- Aprendizaje automático: como subconjunto de la inteligencia artificial (IA), el aprendizaje automático (AA) se refiere a un sistema que tiene la capacidad de aprender de forma activa por sí mismo, a diferencia de recibir, de forma pasiva, instrucciones para ejecutar. Los sistemas HPC pueden utilizarse en aplicaciones de AA altamente avanzadas donde se analizan grandes cantidades de datos, por ejemplo, investigación del cáncer para detectar el melanoma en imágenes.

- Análisis de grandes conjuntos de datos: se recurre a la comparación rápida y a la correlación de grandes conjuntos de datos para complementar investigaciones y resolver problemas académicos, científicos, financieros, comerciales, gubernamentales, de salud y de seguridad cibernética. Este trabajo requiere un rendimiento masivo y capacidades de cómputo de una potencia enorme. Con un estimado de 50 petabytes de datos al año generados por las misiones, la NASA se apoya en la supercomputación para analizar observaciones y realizar simulaciones con grandes conjuntos de información.

- Modelado avanzado y simulación: al no tener que realizar un montaje físico en las primeras etapas del proceso, el modelado avanzado y la simulación permiten que las empresas ahorren tiempo, materiales y costos de contratación de personal para lanzar sus productos al mercado con mayor rapidez. El modelado y la simulación en HPC se aplican en el descubrimiento y la prueba de fármacos, diseños automotrices y aeroespaciales, pronóstico de sistemas climáticos o meteorológicos, y aplicaciones energéticas.

Cómo Funciona la Computación de Alto Rendimiento Existen dos métodos principales para procesar la información en HPC:

- Procesamiento en serie: es el que realizan las unidades de procesamiento central (CPU). Cada núcleo de CPU, por lo general, realiza solo una tarea a la vez. Las CPU son fundamentales para ejecutar diferentes

funciones, como sistemas operativos y aplicaciones básicas (por ej., procesamiento de textos, productividad en la oficina).

- Procesamiento en paralelo: es el que se puede realizar mediante varias CPU o unidades de procesamiento de gráficos (GPU). Las GPU, diseñadas originalmente para gráficos independientes, son capaces de realizar diferentes operaciones aritméticas por medio de una matriz de datos (como píxeles de pantalla) de forma simultánea. La capacidad para trabajar en varios planos de datos al mismo tiempo hace que las GPU sean la elección natural para el procesamiento en paralelo en tareas de aplicaciones de aprendizaje automático (AA), como el reconocimiento de objetos en videos.

Para superar los límites de la supercomputación, se necesitan diferentes arquitecturas de sistemas. Para que el procesamiento en paralelo pueda llevarse a cabo, la mayoría de los sistemas HPC integran varios procesadores y módulos de memoria a través de interconexiones con un ancho de banda enorme. Ciertos sistemas HPC combinan varias CPU y GPU, lo que se conoce como computación heterogénea.

La potencia de procesamiento de las computadoras se mide en unidades llamadas "FLOPS"[81] (operaciones de punto flotante por segundo). A principios de 2019, la supercomputadora más potente que existe alcanzó los 143,5 peta-Flops (143×10^{15}). Este tipo de supercomputadora se llama equipo de petaescala y puede realizar más de mil billones de FLOPS. Por su parte, una computadora de escritorio para juegos de alta gama es más de un millón de veces más lenta y llega apenas a los 200 giga-FLOPS (1×10^{9}).

Gracias a los avances tanto en procesamiento como rendimiento, en el año 2020 fuimos testigos de un nuevo salto en la era de la supercomputación: la exaescala, que es casi 1000 veces más rápida que la petaescala. Esto significa que un sistema de exaescala realiza 10^{18} (o mil millones por mil millones) operaciones por segundo. Necesitaríamos 5 millones de computadoras de escritorio para alcanzar el nivel de rendimiento de procesamiento en supercomputación de 1 exa-Flops, suponiendo que cada computadora de escritorio es capaz de realizar 200 giga-FLOPS.

[81] "FLOPS" describe una velocidad de procesamiento teórica: para hacer posible esa velocidad es necesario enviar datos a los procesadores de forma continua. Por lo tanto, el procesamiento de los datos se debe tener en cuenta en el diseño del sistema. La memoria del sistema, junto con las interconexiones que unen los nodos de procesamiento entre sí, impactan en la rapidez con la que los datos llegan a los procesadores.

Arquitecturas de Software y Hardware En las dos siguientes secciones se explican en detalle las dos clasificaciones de computadoras más conocidas en la actualidad. La primera clasificación, es la clasificación clásica de Flynn en dónde se tienen en cuenta sistemas con uno o varios procesadores, la segunda clasificación es moderna en la que sólo se tienen en cuenta los sistemas con más de un procesador.

El objetivo es presentar de una forma clara los tipos de clasificación que existen en la actualidad desde el punto de vista de distintos autores, así como cuáles son las ventajas e inconvenientes que cada uno ostenta, ya que es común que al resolver un problema particular se usen una o más arquitecturas de Hardware interconectadas generalmente por red.

9.2 Clasificación Clásica de Flynn

La clasificación clásica de arquitecturas de computadoras que hace alusión a sistemas con uno o varios procesadores fue realizada por Michael J. Flynn y la publicó por primera vez en 1966 y por segunda vez en 1970.

Esta taxonomía se basa en el flujo que siguen los datos dentro de la máquina y de las instrucciones sobre esos datos. Se define como flujo de instrucciones al conjunto de instrucciones secuenciales que son ejecutadas por un único procesador y como flujo de datos al flujo secuencial de datos requeridos por el flujo de instrucciones. Con estas consideraciones, Flynn clasifica los sistemas en cuatro categorías:

Single Instruction stream, Single Data stream (SISD) Los sistemas Monoprocesador de este tipo se caracterizan por tener un único flujo de instrucciones sobre un único flujo de datos, es decir, se ejecuta una instrucción detrás de otra. Este es el concepto de arquitectura serie de Von Neumann donde, en cualquier momento, sólo se ejecuta una única instrucción, un ejemplo de estos sistemas son las máquinas secuenciales convencionales.

Single Instruction stream, Multiple Data stream (SIMD) Estos sistemas de procesador Matricial tienen un único flujo de instrucciones que operan sobre múltiples flujos de datos. Ejemplos de estos sistemas los tenemos en las máquinas vectoriales con Hardware escalar y vectorial.

El procesamiento es síncrono, la ejecución de las instrucciones sigue siendo secuencial como en el caso anterior, todos los elementos realizan una misma

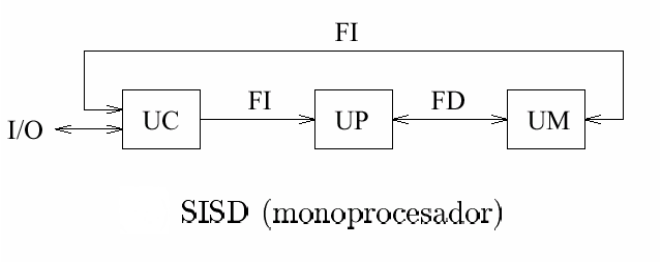


Figura 3: Ejemplo de máquina SISD

instrucción pero sobre una gran cantidad de datos. Por este motivo existirá concurrencia de operación, es decir, esta clasificación es el origen de la máquina paralela.

El funcionamiento de este tipo de sistemas es el siguiente. La unidad de control manda una misma instrucción a todas las unidades de proceso (ALUs). Las unidades de proceso operan sobre datos diferentes pero con la misma instrucción recibida.

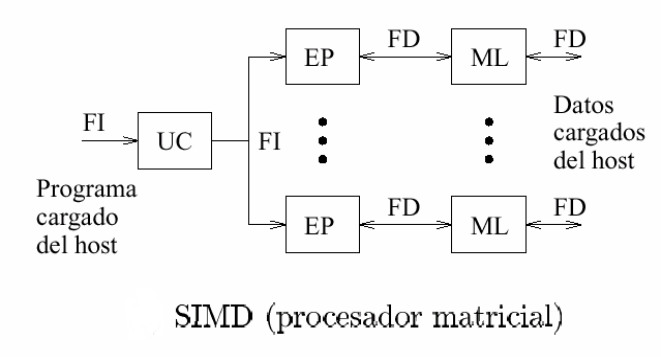


Figura 4: Ejemplo de máquina SIMD

Existen dos alternativas distintas que aparecen después de realizarse esta clasificación:

- Arquitectura Vectorial con segmentación, una CPU única particionada en unidades funcionales independientes trabajando sobre flujos de datos concretos.

- Arquitectura Matricial (matriz de procesadores), varias ALUs idénticas a las que el procesador da instrucciones, asigna una única instrucción pero trabajando sobre diferentes partes del programa.

Multiple Instruction stream, Single Data stream (MISD) Sistemas con múltiples instrucciones Array Sistólico que operan sobre un único flujo de datos. Este tipo de sistemas no ha tenido implementación hasta hace poco tiempo. Los sistemas MISD se contemplan de dos maneras distintas:

- Varias instrucciones operando simultáneamente sobre un único dato.
- Varias instrucciones operando sobre un dato que se va convirtiendo en un resultado que será la entrada para la siguiente etapa. Se trabaja de forma segmentada, todas las unidades de proceso pueden trabajar de forma concurrente.

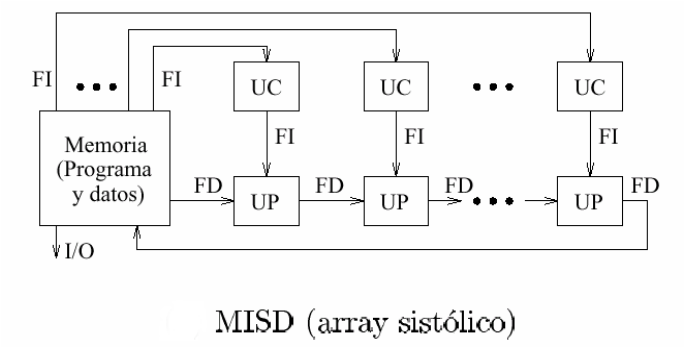


Figura 5: Ejemplo de máquina MISD

Multiple Instruction stream, Multiple Data stream (MIMD) Sistemas con un flujo de múltiples instrucciones Multiprocesador que operan sobre múltiples datos. Estos sistemas empezaron a utilizarse antes de la década de los 80s. Son sistemas con memoria compartida que permiten ejecutar varios procesos simultáneamente (sistema multiprocesador).

Cuando las unidades de proceso reciben datos de una memoria no compartida estos sistemas reciben el nombre de MULTIPLE SISD (MSISD). En arquitecturas con varias unidades de control (MISD Y MIMD), existe otro

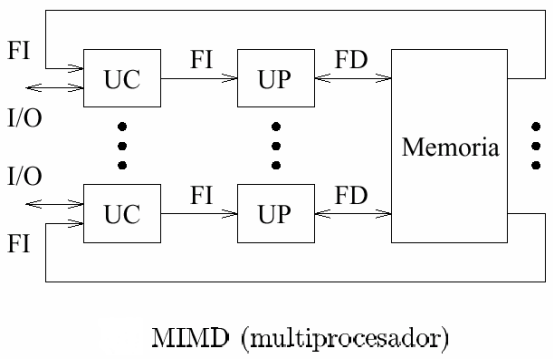


Figura 6: Ejemplo de máquina MIMD

nivel superior con una unidad de control que se encarga de controlar todas las unidades de control del sistema -ejemplo de estos sistemas son las máquinas paralelas actuales-.

9.3 Categorías de Computadoras Paralelas

Clasificación moderna que hace alusión única y exclusivamente a los sistemas que tienen más de un procesador (i.e máquinas paralelas). Existen dos tipos de sistemas teniendo en cuenta su acoplamiento:

- Los sistemas fuertemente acoplados son aquellos en los que los procesadores dependen unos de otros.
- Los sistemas débilmente acoplados son aquellos en los que existe poca interacción entre los diferentes procesadores que forman el sistema.

Atendiendo a esta y a otras características, la clasificación moderna divide a los sistemas en dos tipos: Sistemas multiprocesador (fuertemente acoplados) y sistemas multicomputadoras (débilmente acoplados).

9.3.1 Equipo Paralelo de Memoria Compartida

Un multiprocesador puede verse como una computadora paralela compuesta por varios procesadores interconectados que comparten un mismo sistema de memoria.

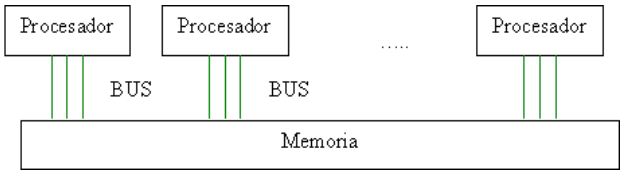


Figura 7: Arquitectura de una computadora paralela con memoria compartida

Los sistemas multiprocesadores son arquitecturas MIMD con memoria compartida. Tienen un único espacio de direcciones para todos los procesadores y los mecanismos de comunicación se basan en el paso de mensajes desde el punto de vista del programador.

Dado que los multiprocesadores comparten diferentes módulos de memoria, pueden acceder a un mismo módulo varios procesadores, a los multiprocesadores también se les llama sistemas de memoria compartida.

Para hacer uso de la memoria compartida por más de un procesador, se requiere hacer uso de técnicas de semáforos que mantienen la integridad de la memoria; esta arquitectura no puede crecer mucho en el número de procesadores interconectados por la saturación rápida del bus o del medio de interconexión.

Dependiendo de la forma en que los procesadores comparten la memoria, se clasifican en sistemas multiprocesador UMA, NUMA, COMA y Pipeline, que explicamos a continuación:

Uniform Memory Access (UMA) Sistema multiprocesador con acceso uniforme a memoria. La memoria física es uniformemente compartida por todos los procesadores, esto quiere decir que todos los procesadores tienen el mismo tiempo de acceso a todas las palabras de la memoria. Cada procesador tiene su propia Caché privada y también se comparten los periféricos.

Los multiprocesadores son sistemas fuertemente acoplados (tightly-coupled), dado el alto grado de compartición de los recursos (Hardware o Software) y el alto nivel de interacción entre procesadores, lo que hace que un procesador depende de lo que hace otro. El sistema de interconexión debe ser rápido y puede ser de uno de los siguientes tipos: bus común, red Crossbar[82] y red

[82]Red reconfigurable que permite la conexión de cada entrada con cualquiera de las salidas, es decir, permite cualquier permutación.

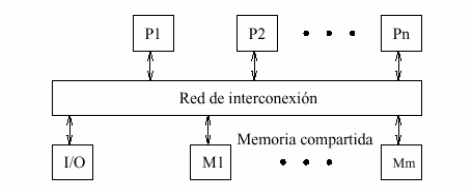


Figura 8: Acceso Uniforme a la memoria UMA

Multietapa. Este modelo es conveniente para aplicaciones de propósito general y de tiempo compartido por varios usuarios, existen dos categorías de sistemas UMA.

- Sistema Simétrico: cuando todos los procesadores tienen el mismo tiempo de acceso a todos los componentes del sistema (incluidos los periféricos), reciben el nombre de sistemas multiprocesador simétrico. Los procesadores tienen el mismo dominio (prioridad) sobre los periféricos y cada procesador tiene la misma capacidad para procesar.

- Sistema Asimétrico: son sistemas con procesador principal y procesadores subordinados, en donde sólo el primero puede ejecutar aplicaciones y dónde el tiempo de acceso para diferentes procesadores no es el mismo. Los procesadores subordinados (Attached) ejecutan código usuario bajo la supervisión del principal, por lo tanto cuando una aplicación es ejecutada en un procesador principal dispondrá de una cierta prioridad.

Non Uniform Memory Access (NUMA) Un sistema multiprocesador NUMA es un sistema de memoria compartida donde el tiempo de acceso varía según donde se encuentre localizado el acceso.

El acceso a memoria, por tanto, no es uniforme para diferentes procesadores, existen memorias locales asociadas a cada procesador y estos pueden acceder a datos de su memoria local de una manera más rápida que a las memorias de otros procesadores, debido a que primero debe aceptarse dicho acceso por el procesador del que depende el módulo de memoria local.

Todas las memorias locales conforman la memoria global compartida y físicamente distribuida y accesible por todos los procesadores.

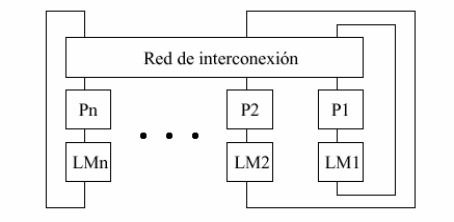


Figura 9: Acceso no uniforme a la memoria NUMA

Cache Only Memory Access (COMA) Los sistemas COMA son un caso especial de los sistemas NUMA. Este tipo de sistemas no ha tenido mucha trascendencia, al igual que los sistemas SIMD.

Las memorias distribuidas son memorias Cachés, por este motivo es un sistema muy restringido en cuanto a la capacidad de memoria global. No hay jerarquía de memoria en cada módulo procesador. Todas las Cachés forman un mismo espacio global de direcciones. El acceso a las Cachés remotas se realiza a través de los directorios distribuidos de las Cachés.

Dependiendo de la red de interconexión utilizada, se pueden utilizar jerarquías en los directorios para ayudar a la localización de copias de bloques de Caché.

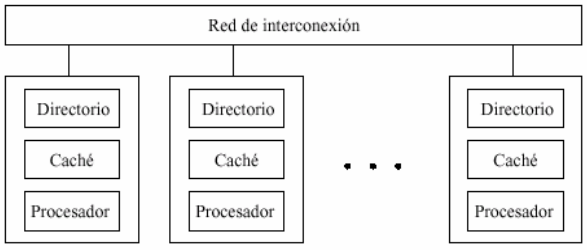


Figura 10: Ejemplo de COMA

Procesador Vectorial Pipeline En la actualidad es común encontrar en un solo procesador los denominados Pipeline o Procesador Vectorial Pipeline del tipo MISD. En estos procesadores los vectores fluyen a través de las unidades aritméticas Pipeline.

Las unidades constan de una cascada de etapas de procesamiento compuestas de circuitos que efectúan operaciones aritméticas o lógicas sobre el flujo de datos que pasan a través de ellas, las etapas están separadas por registros de alta velocidad usados para guardar resultados intermedios. Así la información que fluye entre las etapas adyacentes está bajo el control de un reloj que se aplica a todos los registros simultáneamente.

9.3.2 Equipo Paralelo de Memoria Distribuida

Los sistemas multicomputadoras (los más comunes son los Clusters) se pueden ver como una computadora paralela en el cual cada procesador tiene su propia memoria local. En estos sistemas la memoria se encuentra distribuida y no compartida como en los sistemas multiprocesador. Los procesadores se comunican a través de paso de mensajes, ya que éstos sólo tienen acceso directo a su memoria local y no a las memorias del resto de los procesadores.

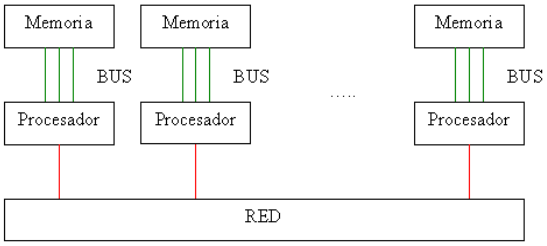


Figura 11: Arquitectura de una computadora paralela con memoria distribuida

La transferencia de los datos se realiza a través de la red de interconexión que conecta un subconjunto de procesadores con otro subconjunto. La transferencia de unos procesadores a otros se realiza por múltiples transferencias entre procesadores conectados dependiendo del establecimiento de dicha red.

Dado que la memoria está distribuida entre los diferentes elementos de proceso, estos sistemas reciben el nombre de distribuidos. Por otra parte, estos sistemas son débilmente acoplados, ya que los módulos funcionan de forma casi independiente unos de otros. Este tipo de memoria distribuida es de acceso lento por ser peticiones a través de la red, pero es una forma muy efectiva de tener acceso a un gran volumen de memoria.

9.3.3 Equipo Paralelo de Memoria Compartida-Distribuida

La tendencia actual en las máquinas paralelas es de aprovechar las facilidades de programación que ofrecen los ambientes de memoria compartida y la escalabilidad de los ambientes de memoria distribuida. En este modelo se conectan entre sí módulos de multiprocesadores, pero se mantiene la visión global de la memoria a pesar de que es distribuida.

Clusters El desarrollo de sistemas operativos y compiladores del dominio público (Linux y Software GNU), estándares para interfaz de paso de mensajes (Message Passing Interface MPI), conexión universal a periféricos (Peripheral Component Interconnect PCI), entre otros, han hecho posible tomar ventaja de los recursos económicos computacionales de producción masiva (procesadores, discos, redes).

La principal desventaja que presenta a los proveedores de multicomputadoras es que deben satisfacer una amplia gama de usuarios, es decir, deben ser generales. Esto aumenta los costos de diseños y producción de equipos, así como los costos de desarrollo de Software que va con ellos: sistema operativo, compiladores y aplicaciones. Todos estos costos deben ser añadidos cuando se hace una venta. Por supuesto alguien que sólo necesita procesadores y un mecanismo de pase de mensajes no debería pagar por todos estos añadidos que nunca usará. Estos usuarios son los que están impulsando el uso de Clusters principalmente de computadoras personales (PC), cuya arquitectura se muestra a continuación:

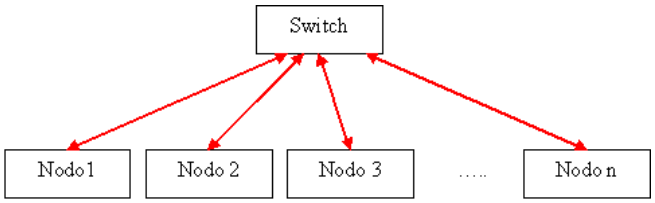


Figura 12: Arquitectura de un cluster

Los Cluster[83] se pueden clasificar en dos tipos según sus características físicas:

[83] Existe el Ranking de las 500 supercomputadoras más poderosas del mundo (esta se actualiza cada seis meses en junio y noviembre) y puede ser consultada en:
https://top500.org

- Cluster homogéneo: si todos los procesadores y/o nodos participantes en el equipo paralelo son iguales en capacidad de cómputo -es permitido variar la cantidad de memoria o disco duro en cada procesador-

- Cluster heterogéneo: es aquel en que al menos uno de los procesadores y/o nodos participantes en el equipo paralelo son de distinta capacidad de cómputo

El Clustering no presenta dependencias a nivel de Hardware (no todos los equipos necesitan el mismo Hardware) ni a nivel de Software (no necesitan el mismo sistema operativo). Este tipo de sistemas disponen de una interfaz que permite dirigir el comportamiento de los Clusters. Dicha interfaz es la encargada de la interacción con usuarios y procesos, realizando la división de la carga entre los diversos servidores que compongan el Cluster.

Los Clusters pueden formarse de diversos equipos; los más comunes son los de computadoras personales, pero es creciente el uso de computadoras multiprocesador de más de un procesador de memoria compartida interconectados por red con los demás nodos del mismo tipo, incluso el uso de computadoras multiprocesador de procesadores vectoriales Pipeline. Los Clusters armados con la configuración anterior tienen grandes ventajas para procesamiento paralelo:

- La reciente explosión en redes implica que la mayoría de los componentes necesarios para construir un Cluster son vendidos en altos volúmenes y por lo tanto son económicos. Ahorros adicionales se pueden obtener debido a que sólo se necesitará una tarjeta de vídeo, un monitor y un teclado por Cluster. El mercado de los multiprocesadores es más reducido y más costoso

- El reemplazar un componente defectuoso en un Cluster es relativamente trivial comparado con hacerlo en un multiprocesador, permitiendo una mayor disponibilidad de Clusters cuidadosamente diseñados

Desventajas del uso de Clusters de computadoras personales para procesamiento paralelo:

- Con raras excepciones, los equipos de redes generales producidos masivamente no están diseñados para procesamiento paralelo

y típicamente su latencia es alta y los anchos de banda pequeños comparados con multiprocesadores. Dado que los Clusters explotan tecnología que sea económica, los enlaces en el sistema no son veloces implicando que la comunicación entre componentes debe pasar por un proceso de protocolos de negociación lentos, incrementando seriamente la latencia. En muchos y en el mejor de los casos (debido a costos) se recurre a una red tipo Fast Ethernet restringimiento la escalabilidad del Cluster

- Hay poco soporte de Software para manejar un Cluster como un sistema integrado

- Los procesadores no son tan eficientes como los procesadores usados en los multiprocesadores para manejar múltiples usuarios y/o procesos. Esto hace que el rendimiento de los Clusters se degrade con relativamente pocos usuarios y/o procesos

- Muchas aplicaciones importantes disponibles en multiprocesadores y optimizadas para ciertas arquitecturas, no lo están en Clusters

Sin lugar a duda los Clusters presentan una alternativa importante para varios problemas particulares, no sólo por su economía, si no también porque pueden ser diseñados y ajustados para ciertas aplicaciones. Las aplicaciones que pueden sacar provecho de Clusters son en donde el grado de comunicación entre procesos es de bajo a medio.

Tipos de Cluster Básicamente existen tres tipo de Clusters, cada uno de ellos ofrece ventajas y desventajas, el tipo más adecuado para el cómputo científico es del de alto-rendimiento, pero existen aplicaciones científicas que pueden usar más de un tipo al mismo tiempo.

- Alta disponibilidad (Fail-over o High-Availability): este tipo de Cluster está diseñado para mantener uno o varios servicios disponibles incluso a costa de rendimiento, ya que su función principal es que el servicio jamás tenga interrupciones como por ejemplo un servicio de bases de datos de transacciones bancarias

- Alto rendimiento (HPC o High Performance Computing): este tipo de Cluster esta diseñado para obtener el máximo rendimiento de la aplicación utilizada incluso a costa de la disponibilidad del

sistema, es decir el Cluster puede sufrir caídas, este tipo de configuración está orientada a procesos que requieran mucha capacidad de cálculo.

• Balanceo de Carga (Load-Balancing): este tipo de Cluster está diseñado para balancear la carga de trabajo entre varios servidores, lo que permite tener, por ejemplo, un servicio de cálculo intensivo multiusuarios que detecte tiempos muertos del proceso de un usuario para ejecutar en dichos tiempos procesos de otros usuarios.

Grids Son cúmulos (grupo de Clusters) de arquitecturas en paralelo interconectados por red, los cuales distribuyen tareas entre los Clusters que lo forman, estos pueden ser homogéneos o heterogéneos en cuanto a los nodos componentes del cúmulo. Este tipo de arquitecturas trata de distribuir cargas de trabajo acorde a las características internas de cada Cluster y las necesidades propias de cada problema, esto se hace a dos niveles, una en la parte de programación en conjunto con el balance de cargas y otra en la parte de Hardware que tiene que ver con las características de cada arquitectura que conforman el cúmulo.

Balance de Carga A la hora de diseñar un sistema paralelo con compartición de recursos, es necesario considerar cómo balancear la carga de trabajo. Se entiende este concepto, como la técnica usada para dividir el trabajo a compartir entre varios procesos, equipos de cómputo, u otros recursos. Está muy relacionada con los sistemas multiproceso, que trabajan o pueden trabajar con más de una unidad para llevar a cabo su funcionalidad.

Para evitar los cuellos de botella, el balance de la carga de trabajo se reparte de forma equitativa a través de un algoritmo que estudia las peticiones del sistema y las redirecciona a la mejor opción.

Balance de Carga por Hardware presenta las siguientes características:

- A partir de un algoritmo de planificación de procesos -Round Robin, LRU-, examina las peticiones entrantes y selecciona el más apropiado entre los distintos elementos del sistema

- La selección del siguiente libre del sistema está basada en el algoritmo de sustitución y es aleatoria
- La sesión debe de ser mantenida por el desarrollador
- Al ser un proceso de Hardware, es muy rápido

Balance de Carga por Software presenta las siguientes características:

- Examinan la solicitud para garantizar que se puede satisfacer la demanda de los usuarios
- Distintas peticiones del mismo usuario son servidas simultáneamente o secuencialmente según se definan
- Más lentos que los balanceadores de Hardware
- Normalmente son soluciones baratas

Escalabilidad Se entiende por escalabilidad a la capacidad de adaptación y respuesta de un sistema con respecto al rendimiento del mismo a medida que aumentan de forma significativa la carga computacional del mismo. Aunque parezca un concepto claro, la escalabilidad de un sistema es un aspecto complejo e importante del diseño de sistemas paralelos.

La escalabilidad está íntimamente ligada al diseño de sistemas paralelos, influye en el rendimiento de forma significativa. Si una aplicación está bien diseñada, la escalabilidad no constituye un problema. Analizando la escalabilidad, se deduce de la implementación y del diseño general del sistema. No es atributo del sistema configurable.

La escalabilidad supone un factor crítico en el crecimiento de un sistema paralelo. Si un sistema tiene como objetivo crecer la carga computacional -en el número de usuarios o procesos- manteniendo su rendimiento actual, tiene que evaluar dos posibles opciones:

- Con un Hardware de mayor potencia o
- Con una mejor combinación de Hardware y Software

Se pueden distinguir dos tipos de escalabilidad, vertical y horizontal:

- El escalar verticalmente o escalar hacia arriba, significa el añadir más recursos a un solo nodo en particular dentro de un sistema, tal como el añadir memoria o un disco duro más rápido a una computadora.
- La escalabilidad horizontal, significa agregar más nodos a un sistema, tal como añadir una computadora nueva a un programa de aplicación para espejo.

Escalabilidad Vertical El escalar hacia arriba de un sistema viene a significar una migración de todo el sistema a un nuevo Hardware que es más potente y eficaz que el actual. Una vez se ha configurado el sistema futuro, se realizan una serie de validaciones y copias de seguridad y se pone en funcionamiento. Las aplicaciones que estén funcionando bajo la arquitectura Hardware antigua no sufren con la migración, el impacto en el código es mínimo.

Este modelo de escalabilidad vertical tiene un aspecto negativo. Al aumentar la potencia en base a ampliaciones de Hardware, llegará un momento que existirá algún tipo de limitación de Hardware. Además a medida que se invierte en Hardware de muy altas prestaciones, los costos se disparan tanto de forma temporal -ya que si se ha llegado al umbral máximo, hay componentes de Hardware que tardan mucho tiempo en ampliar su potencia de forma significativa- como económicos. Sin embargo a nivel estructural no supone ninguna modificación reseñable, lo que la convierte en una buena opción si los costos anteriores son asumibles.

Escalabilidad Horizontal La escalabilidad horizontal consiste en potenciar el rendimiento del sistema paralelo desde un aspecto de mejora global, a diferencia de aumentar la potencia de una única parte del mismo. Este tipo de escalabilidad se basa en la modularidad de su funcionalidad, por ello suele estar conformado por una agrupación de equipos que dan soporte a la funcionalidad completa. Normalmente, en una escalabilidad horizontal se añaden equipos para dar más potencia a la red de trabajo.

Con un entorno de este tipo, es lógico pensar que la potencia de procesamiento es directamente proporcional al número de equipos en la red. El total de la potencia de procesamiento es la suma de la velocidad física de cada equipo transferida por la partición de aplicaciones y datos extendida a través de los nodos.

Si se aplica un modelo de escalabilidad basado en la horizontalidad, no existen limitaciones de crecimiento a priori. Como principal defecto, este modelo de escalabilidad supone una gran modificación en el diseño, lo que conlleva a una gran trabajo de diseño y reimplantación. Si la lógica se ha concebido para un único servidor, es probable que se tenga que estructurar el modelo arquitectónico para soportar este modelo de escalabilidad.

El encargado de cómo realizar el modelo de partición de datos en los diferentes equipos es el desarrollador. Existen dependencias en el acceso a la aplicación. Es conveniente, realizar una análisis de actividad de los usuarios para ir ajustando el funcionamiento del sistema. Con este modelo de escalabilidad, se dispone de un sistema al que se pueden agregar recursos de manera casi infinita y adaptable al crecimiento de cargas de trabajo y nuevos usuarios.

La escalabilidad cuenta como factor crítico en el crecimiento de usuarios. Es mucho más sencillo diseñar un sistema con un número constante de usuarios -por muy alto que sea este- que diseñar un sistema con un número creciente y variable de usuarios. El crecimiento relativo de los números es mucho más importante que los números absolutos.

9.3.4 Cómputo Paralelo en Multihilos

En una computadora, sea secuencial o paralela, para aprovechar las capacidades crecientes del procesador, el sistema operativo divide su tiempo de procesamiento entre los distintos procesos, de forma tal que para poder ejecutar a un proceso, el Kernel les asigna a cada uno una prioridad y con ello una fracción del tiempo total de procesamiento, de forma tal que se pueda atender a todos y cada uno de los procesos de manera eficiente.

En particular, en la programación en paralelo usando MPI, cada proceso -que eventualmente puede estar en distinto procesador- se lanza como una copia del programa con datos privados y un identificador del proceso único, de tal forma que cada proceso sólo puede compartir datos con otro proceso mediante paso de mensajes.

Esta forma de lanzar procesos por cada tarea que se desee hacer en paralelo es costosa, por llevar cada una de ellas toda una gama de subprocesos para poderle asignar recursos por parte del sistema operativo. Una forma más eficiente de hacerlo es que un proceso pueda generar bloques de subprocesos que puedan ser ejecutados como parte del proceso (como subtareas), así en el tiempo asignado se pueden atender a más de un subproceso

de manera más eficiente, esto es conocido como programación multihilos.

Los hilos realizarán las distintas tareas necesarias en un proceso. Para hacer que los procesos funcionen de esta manera, se utilizan distintas técnicas que le indican al Kernel cuales son las partes del proceso que pueden ejecutarse simultáneamente y el procesador asignará una fracción de tiempo exclusivo al hilo del tiempo total asignado al proceso.

Los datos pertenecientes al proceso pasan a ser compartidos por los subprocesos lanzados en cada hilo y mediante una técnica de semáforos el Kernel mantiene la integridad de estos. Esta técnica de programación puede ser muy eficiente si no se abusa de este recurso, permitiendo un nivel más de paralelización en cada procesador. Esta forma de paralelización no es exclusiva de equipos multiprocesadores o multicomputadoras, ya que pueden ser implementados a nivel de sistema operativo.

9.3.5 Cómputo Paralelo en CUDA

Son las siglas de arquitectura unificada de dispositivos de cómputo (Compute Unified Device Architecture CUDA) que hace referencia a una plataforma de computación en paralelo (que incluye su propia RAM conocida como GRAM para hacer óptima la solicitud de dicho recurso por sus múltiples cores) incluyendo un compilador y un conjunto de herramientas de desarrollo que permiten a los programadores usar una variación del lenguaje de programación C -Por medio de Wrappers se puede usar MatLab, Python, Julia, Fortran y Java en vez de C/C++- para codificar algoritmos en las unidades de procesamiento de gráficos (Graphics Processing Unit GPU).

CUDA intenta explotar las ventajas de las GPU (de decenas a centenas de cores[84]) frente a las CPU (decenas de cores[85]) de propósito general utilizando el paralelismo que ofrecen sus múltiples cores o núcleos, que permiten el lan-

[84]La GRAM de las tarjetas CUDA es relativamente pequeña (a lo más de algunas decenas de Gigabytes), pero está diseñada para que pueda ser compartida por sus cores de forma óptima. Pero si es necesario que múltiples cores de la GPU soliciten datos a la RAM del equipo de cómputo, pueden ocasionar degradación en la eficiencia de la GPU porque la RAM tiene una arquitectura DDR4 o DDR5 (lo que significa que solo puede atender peticiones de datos de sólo 4 o 5 cores simultáneamente).

[85]Los actuales CPUs pueden tener decenas de cores y comparten hasta Terabyates de RAM. Pero la arquitectura actual de la RAM es de tipo DDR4 o DDR5, esto significa, que sólo hay 4 o 5 canales para satisfacer las solicitudes de petición de datos de/hacia los cores, si más cores necesita acceso a la RAM tendrán que esperar que los otros cores concluyan su uso, mermando la eficiencia de la CPU.

zamiento de un altísimo número de hilos simultáneos. Si una aplicación está bien diseñada utilizando numerosos hilos que realizan tareas independientes usando la memoria compartida de la tarjeta (que es lo que hacen las GPU al procesar gráficos, su tarea natural) hará un uso eficiente de este dispositivo.

Por ejemplo, la tarjeta gráfica de NVidia GEFORCE RTX 3090 proporciona 10,496 cores y 24 GB de GDDR6x, mientras que la tarjeta NVidia Titan RTX proporciona 130 Tensor[86] TFLOP de rendimiento, 576 núcleos tensores y 24 GB de memoria GDDR6.

Ahora, miles de desarrolladores, científicos e investigadores están encontrando innumerables aplicaciones prácticas para esta tecnología en campos como el procesamiento de vídeo e imágenes, la biología y la química computacional, la simulación de la dinámica de fluidos, la reconstrucción de imágenes de Tomografía Axial Computarizada TAC, el análisis sísmico o el trazado de rayos, entre otros.

Procesamiento paralelo con CUDA Los sistemas informáticos están pasando de realizar el «procesamiento central» en la CPU a realizar «coprocesamiento» repartido entre la CPU y la GPU. Para posibilitar este nuevo paradigma computacional, NVIDIA ha inventado la arquitectura de cálculo paralelo CUDA, que ahora se incluye en las GPUs GeForce, ION Quadro y Tesla GPUs, lo cual representa una base instalada considerable para los desarrolladores de aplicaciones.

CUDA ha sido recibida con entusiasmo por la comunidad científica. Por ejemplo, se está utilizando para acelerar AMBER, un simulador de dinámica molecular empleado por más de 60,000 investigadores del ámbito académico y farmacéutico de todo el mundo para acelerar el descubrimiento de nuevos medicamentos. En el mercado financiero, Numerix y CompatibL introdujeron soporte de CUDA para una nueva aplicación de cálculo de riesgo de contraparte y, como resultado, se ha multiplicado por 18 la velocidad de la aplicación. Cerca de 400 instituciones financieras utilizan Numerix en la actualidad.

Un buen indicador de la excelente acogida de CUDA es la rápida adopción de la GPU Tesla para aplicaciones de GPU Computing. En la actualidad hay más de 700 Clusters de GPUs instalados en compañías publicadas en Fortune

[86] Un Tensor core (o núcleos Tensor) calculan la operación de una matriz 4x4 completa, la cual se calcula por reloj. Estos núcleos pueden multiplicar dos matrices FP16 4x4 y sumar la matriz FP32 al acumulador.

500 de todo el mundo, lo que incluye empresas como Schlumberger y Chevron en el sector energético o BNP Pariba en el sector bancario. Y en el 2021 se puso en funcionamiento el cluster Perlmutter que esta formado por 6,159 NVIDIA A100 Tensor Core GPUs para ciencias astrofísicas y atmosféricas.

Por otra parte, la reciente llegada de los últimos sistemas operativos de Microsoft y Apple está convirtiendo el GPU Computing en una tecnología de uso masivo. En estos nuevos sistemas, la GPU no actúa únicamente como procesador gráfico, sino como procesador paralelo de propósito general accesible para cualquier aplicación.

Plataforma de Cálculo Paralelo CUDA proporciona unas cuantas extensiones de MatLab, Fortran, Python, Julia, C, C++, etc. Que permiten implementar el paralelismo en el procesamiento de tareas y datos con diferentes niveles de granularidad. El programador puede expresar ese paralelismo mediante diferentes lenguajes de alto nivel como Fortran, Python, Julia, C y C++ o mediante estándares abiertos como las directivas de OpenACC -que es un estándar de programación para el cómputo en paralelo desarrollado por Cray, CAPS, Nvidia y PGI diseñado para simplificar la programación paralela de sistemas heterogéneos de CPU/GPU-. En la actualidad, la plataforma CUDA se utiliza en miles de aplicaciones aceleradas en la GPU y en miles de artículos de investigación publicados, en las áreas de:

- Bioinformática
- Cálculo financiero
- Dinámica de fluidos computacional (CFD)
- Ciencia de los datos, analítica y bases de datos
- Defensa e Inteligencia
- Procesamiento de imágenes y visión computarizadas
- EDA (diseño automatizado)
- Aprendizaje automático, Inteligencia artificial
- Ciencia de los materiales
- Medios audiovisuales y entretenimiento

- Imágenes médicas
- Dinámica molecular
- Análisis numérico
- Química cuántica
- Exploración sísmica
- Mecánica estructural computacional
- Visualización e interacción de proteínas
- Modelos meteorológicos y climáticos

Algunas bibliotecas aceleradas en la GPU:

- Thrust C++ Template
- cuBLAS
- cuSPARSE
- NPP
- cuFFT

Algunos lenguajes de programación:

- CUDA C/C++, Fortran, Python, Julia
- .NET, Java

Algunos compiladores disponibles:

- OpenACC, paraleliza automáticamente los bucles de código Fortran o C utilizando directivas
- Compilador de autoparalelización de C y Fortran (de PGI) para CUDA C
- Compilador de autoparalelización HMPP de CAPS para CUDA C basado en C y Fortran

- Fortran, Python, Julia, Java, C++ y C
- Compilador de Fortran para CUDA de PGI
- Traductor de Fortran a C para CUDA
- FLAGON: librería de Fortran 95 para cálculo en la GPU
- Interfaz (Wrapper) de Python para CUDA: PyCUDA
- Wrapper de Java
- jCUDA: Java para CUDA
- Vínculos para las librerías BLAS y FFT de CUDA
- JaCUDA
- Integración de .NET para CUDA
- Thrust: librería de plantillas de C++ para CUDA
- CuPP : infraestructura de C++ para CUDA
- Libra: capa de abstracción de C/C++ para CUDA
- F# para CUDA
- Librería ArrayFire para acelerar el desarrollo de aplicaciones de GPU Computing en C, C++, Fortran y Python

Soporte de MATLAB, Mathematica, R, LabView:

- MATLAB, Mathematica, R, LabView
- MathWorks: Librerías MATLAB procesadas con GPUs NVIDIA
- Plugin Jacket basado en CUDA para MATLAB
- GPULib: librería de funciones matemáticas con vínculos para IDL y MATLAB
- Programación con CUDA en Mathematica de Wolfram

- Plugin de Mathematica para CUDA
- Habilitación del GPU Computing en el entorno estadístico de R
- Librería CUDA para LabVIEW de National Instruments
- Formas de usar CUDA desde LabVIEW CVI

Además existen herramientas de productividad y Cluster:

- Soporte de Eclipse para CUDA
- CUDA Occupancy Calculator
- Administrador de Clusters de cálculo para GPUs Tesla
- PBS Works para GPUs Tesla
- Scyld de Penguin Computing

9.4 Métricas de Desempeño

Las métricas de desempeño del procesamiento de alguna tarea en paralelo es un factor importante para medir la eficiencia y consumo de recursos al resolver una tarea con un número determinado de procesadores y recursos relacionados de la interconexión de éstos.

Entre las métricas para medir desempeño en las cuales como premisa se mantiene fijo el tamaño del problema, destacan las siguientes:

- Factor de aceleración
- Eficiencia
- Fracción serial

Cada una de ellas mide algo en particular y sólo la combinación de estas dan un panorama general del desempeño del procesamiento en paralelo de un problema en particular en una arquitectura determinada al ser comparada con otras.

Factor de Aceleración (o Speed-Up) Se define como el cociente del tiempo que se tarda en completar el cómputo de la tarea usando un sólo procesador entre el tiempo que necesita para realizarlo en p procesadores trabajando en paralelo:

$$s = \frac{T(1)}{T(p)} \tag{9.1}$$

se asume que se usará el mejor algoritmo tanto para un solo procesador como para p procesadores.

Esta métrica en el caso ideal debería de aumentar de forma lineal al aumento del número de procesadores.

Eficiencia Se define como el cociente del tiempo que se tarda en completar el cómputo de la tarea usando un solo procesador entre el número de procesadores multiplicado por el tiempo que necesita para realizarlo en p procesadores trabajando en paralelo:

$$e = \frac{T(1)}{pT(p)} = \frac{s}{p}. \tag{9.2}$$

Este valor será cercano a la unidad cuando el Hardware se esté usando de manera eficiente, en caso contrario el Hardware será desaprovechado.

Fracción serial Se define como el cociente del tiempo que se tarda en completar el cómputo de la parte secuencial de una tarea entre el tiempo que se tarda en completar el cómputo de la tarea usando un solo procesador:

$$f = \frac{T_s}{T(1)} \tag{9.3}$$

pero usando la ley de Amdahl:

$$T(p) = T_s + \frac{T_p}{p}$$

y reescribiéndola en términos de factor de aceleración, obtenemos la forma operativa del cálculo de la fracción serial que adquiere la forma siguiente:

$$f = \frac{\frac{1}{s} - \frac{1}{p}}{1 - \frac{1}{p}}. \tag{9.4}$$

Esta métrica permite ver las inconsistencias en el balance de cargas, ya que su valor debierá de tender a cero en el caso ideal, por ello un incremento en el valor de f es un aviso de granularidad fina con la correspondiente sobrecarga en los procesos de comunicación.

Costo o Trabajo Se define el costo o trabajo de resolver un problema en paralelo como el producto del tiempo de cálculo en paralelo T_p por el número de procesadores usando p y se representa por:

$$C_p = p * T_p.$$

Aceleración y Eficiencia Relativa Cuando se trabaja en más de un equipo paralelo -supongamos con p y p' procesadores con $p \geq p'$- es común comparar el desempeño entre dichos equipos. Para ello se define la aceleración relativa como:

$$S_p^{p'} = \frac{T_{p'}}{T_p}$$

para $p \geq p'$, en la cual se espera que:

$$S_p^{p'} \simeq \frac{p}{p'}$$

y eficiencia relativa como:

$$E_p^{p'} = \frac{p'}{p} S_p^{p'} = \frac{p'}{p} \frac{T_{p'}}{T_p}.$$

Análisis de Rendimiento Usando Métricas Suponiendo un Cluster con 17 Cores o núcleos[87], se muestra una ejemplificación de las métricas para un problema de ejemplo:

[87] A cada procesador encapsulado se le llama Core o núcleo, logrando que la comunicación entre ellos se realiza de una forma más rápida a través de un bus interno integrado en la propia pastilla de silicio sin tener que recurrir por tanto al bus del sistema mucho más lento. Al contrario del caso de la tecnología HyperThreading que el sistema operativo los reconoce como un core en el equipo, pero estos cores virtuales distan mucho del desempeño de uno real.

Cores	Tiempo	Aceleración	Eficiencia	Frac. Ser.
1	5295			
3	2538	2.08	0.69	0.218
5	1391	3.80	0.76	0.078
9	804	6.58	0.73	0.045
17	441	12.00	0.70	0.025

Nótese que en todos los casos la fracción serial disminuye sustancialmente con el aumento del número de procesadores, pero la aceleración está por debajo del valor esperado.

Suponiendo un Cluster A con 100 Cores y un Cluster B con 128 Cores, se muestra una ejemplificación de las métricas para un problema de ejemplo en ambos Clusters con los siguientes resultados:

Cores	Cluster A	Cluster B
16	9158 seg	ND
32	5178 seg	5937 seg
64	3647 seg	4326 seg
100	2661 seg	
128		2818 seg

Como se muestra en la tabla, en todos los casos el Cluster A usando como máximo 100 Cores obtiene un tiempo de cálculo inferior al que requiere Cluster B usando a lo más los 128 Cores.

Haciendo uso de las métricas de aceleración y eficiencia relativa[88] se tiene que para el Cluster B:

$$S_{128}^{32} = 5937/2818 = 2.10$$

donde lo esperado sería:

$$S_{128}^{32} = 32/128 = 4.00,$$

para el caso de la eficiencia:

$$E_{128}^{32} = (32/128) * (5937/2818) = 0.52.$$

[88] Aceleración relativa es $S_p^{p'} = \frac{T_{p'}}{T_p}$ para $p \geq p'$, en la cual se espera que $S_p^{p'} \simeq \frac{p}{p'}$ y eficiencia relativa es $E_p^{p'} = \frac{p'}{p} S_p^{p'} = \frac{p'}{p} \frac{T_{p'}}{T_p}$.

En el caso del Cluster A se tiene que:

$$S_{100}^{16} = 9158/2661 = 3.44$$

dónde lo esperado sería:

$$S_{100}^{16} = 16/100 = 6.35,$$

para el caso de la eficiencia:

$$E_{100}^{16} = (16/100) * (9158/2661) = 0.55.$$

Haciendo uso del mismo número de Cores base para el Cluster A que para Cluster B, se tiene que:

$$S_{100}^{32} = 5178/2661 = 1.94$$

dónde lo esperado sería:

$$S_{100}^{16} = 32/100 = 3.12,$$

para el caso de la eficiencia:

$$E_{100}^{16} = (32/100) * (5178/2661) = 0.62.$$

De todo lo anterior, se desprende que el Cluster A obtiene valores de una aceleración y eficiencias relativas ligeramente mejores que el Cluster B, pero esto no se refleja en la disminución de casi 6% del tiempo de ejecución y del uso de 28 Cores menos.

Además, el costo computacional:

$$C_p = P * T_p,$$

que para el caso del Cluster B es:

$$C_{128} = 360,704$$

y en Cluster A es:

$$C_{100} = 266,100$$

que representa una disminución de 27%; además de un factor muy importante, el Cluster A tuvo un costo monetario mucho menor con respecto del Cluster B.

9.5 Programación de Cómputo de Alto Rendimiento

Hay muchas aplicaciones a las herramientas computacionales, pero nos interesan aquellas que permitan resolver problemas concomitantes en Ciencia e Ingeniería. Muchas de estas aplicaciones caen en lo que comúnmente se llama cómputo científico. La computación científica es el campo de estudio relacionado con la construcción de modelos matemáticos, técnicas numéricas para resolver problemas científicos y de ingeniería; y su respectiva implementación computacional. La solución de estos problemas genera un alto consumo de memoria, espacio de almacenamiento o tiempo de cómputo; por ello nos interesa trabajar en computadoras que nos puedan satisfacer estas demandas.

La computación de alto rendimiento (véase 9) -High performance Computing o HPC en inglés- es la agregación de potencia de cálculo para resolver problemas complejos en Ciencia e Ingeniería o Gestión. Para lograr este objetivo, la computación de alto rendimiento se apoya en tecnologías computacionales como los Clusters, las supercomputadoras o la computación paralela. La mayoría de las ideas actuales de la computación distribuida se han basado en la computación de alto rendimiento.

La computación paralela o de alto rendimiento es una forma de cómputo en la que muchas instrucciones se ejecutan simultáneamente, operando sobre el principio de que problemas grandes, a menudo se pueden dividir en unos más pequeños, que luego son resueltos simultáneamente (en paralelo). Hay varias formas diferentes de computación paralela: paralelismo a nivel de bits, paralelismo a nivel de instrucción, paralelismo de datos y paralelismo de tareas.

El paralelismo se ha empleado durante muchos años, sobre todo en la computación de altas prestaciones, pero el interés en ella ha crecido últimamente debido a las limitaciones físicas que impiden el aumento de la frecuencia de los actuales procesadores comerciales. Como el consumo de energía –y por consiguiente la generación de calor– de las computadoras constituye una preocupación en los últimos años, la computación en paralelo se ha convertido en el paradigma dominante en la arquitectura de computadoras, principalmente en forma de procesadores multinúcleo.

Las computadoras paralelas pueden clasificarse según el nivel de paralelismo que admite su Hardware:

- Equipos con procesadores multinúcleo

- Equipos con multiprocesador (múltiples procesadores multinúcleo en una sola máquina)
- Procesadores masivamente paralelos
- Equipos con una o más tarjetas CUDA (Compute Unified Device Architecture)
- Cluster de equipos multinúcleo, multiprocesador y/o CUDAS
- Cúmulos de Clusters (Grids)

Muchas veces, para acelerar tareas específicas, se utilizan arquitecturas especializadas de computación en paralelo junto a procesadores tradicionales.

Existen múltiples vertientes en el cómputo en paralelo, algunas de ellas son:

- Cómputo en memoria compartida usando OpenMP
- Cómputo en memoria distribuida usando MPI

Como es de esperarse, los programas informáticos paralelos son más complejos y difíciles de escribir que los secuenciales, porque la concurrencia introduce nuevos tipos de errores de Software -por ello existe una creciente gama de herramientas que coadyuvan a mejorar la escritura, depuración y desempeño de los programas en paralelo-, pero la comunicación y sincronización entre diferentes subtareas son algunos de los mayores obstáculos para obtener un buen rendimiento del programa paralelo.

Además, el tiempo de programación necesario para desarrollar una aplicación paralela eficiente y eficaz para la gran mayoría de los programadores puede ser de semanas o meses en el mejor de los casos. Por ello, es necesario hacer un balance entre las diferentes opciones de paralelización para no invertir un tiempo precioso que puede no justificar dicha inversión económica, de recursos computacionales y sobre todo de tiempo.

Actualmente, en muchos centros de cómputo es una práctica común usar directivas de compilación en equipos paralelos sobre programas escritos de forma secuencial, con la esperanza que sean puestos por el compilador como programas paralelos. Esto en la gran mayoría de los casos genera códigos poco eficientes, pese a que corren en equipos paralelos y pueden usar toda

la memoria compartida de dichos equipos, el algoritmo ejecutado continúa siendo secuencial en una gran parte del código.

Si la arquitectura paralela donde se implemente el programa es UMA de acceso simétrico, los datos serán accesados a una velocidad de memoria constante. En caso contrario, al acceder a un conjunto de datos es común que una parte de estos sean locales a un procesador (con un acceso del orden de nanosegundos), pero el resto de los datos deberán ser accesados mediante red (con acceso del orden de milisegundos), siendo esto muy costoso en tiempo de procesamiento.

Por lo anterior, si se cuenta con computadoras con memoria compartida o que tengan interconexión por bus, salvo en casos particulares no será posible explotar éstas características eficientemente. Pero en la medida en que se adecuen los programas para usar bibliotecas y compiladores acordes a las características del equipo disponible -algunos de ellos sólo existen de manera comercial- la eficiencia aumentará de manera importante.

9.5.1 Programando con OpenMP para Memoria Compartida

OpenMP es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en múltiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++, Julia, Python y Fortran sobre la base del modelo de ejecución Fork-join. Está disponible en muchas arquitecturas, incluidas las plataformas de Unix, Linux y de Microsoft Windows. Se compone de un conjunto de directivas de compilador, rutinas de biblioteca, y variables de entorno que influyen en el comportamiento en tiempo de ejecución.

Definido conjuntamente por proveedores de Hardware y de Software, OpenMP es un modelo de programación portable y escalable que proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas, para plataformas que van desde las computadoras de escritorio hasta supercomputadoras. Una aplicación construida con un modelo de programación paralela híbrido se puede ejecutar en un Cluster de computadoras utilizando OpenMP y MPI, o a través de las extensiones de OpenMP para los sistemas de memoria distribuida.

OpenMP se basa en el modelo *Fork-join*, paradigma que proviene de los sistemas Unix, donde una tarea muy pesada se divide en K hilos (Fork) con menor peso, para luego «recolectar» sus resultados al final y unirlos en un solo resultado (Join).

Cuando se incluye una directiva de compilador OpenMP esto implica que se incluye una sincronización obligatoria en todo el bloque. Es decir, el bloque de código se marcará como paralelo y se lanzarán hilos según las características que nos dé la directiva, y al final de ella habrá una barrera para la sincronización de los diferentes hilos (salvo que implícitamente se indique lo contrario con la directiva Nowait). Este tipo de ejecución se denomina *Fork-join*.

OpenMP también soporta el modelo de paralelismo de tareas. El equipo de hilos del bloque de código paralelo ejecuta las tareas especificadas dentro de dicho bloque. Las tareas permiten un paralelismo asíncrono. Desde la versión 4.0 lanzada en 2013 admite la especificación de dependencias entre tareas, relegando a la biblioteca de tiempo de ejecución de OpenMP el trabajo de planificar las tareas y ponerlas en ejecución. Los hilos de ejecución irán ejecutando las tareas a medida que estas estén disponibles (sus dependencias ya estén satisfechas). El uso de tareas da lugar a sincronización con una granularidad más fina. El uso de barreras no es estrictamente necesario, de manera que los hilos pueden continuar ejecutando tareas disponibles sin necesidad de esperar a que todo el equipo de hilos acabe un bloque paralelo. El uso de tareas con dependencias crea un grafo, pudiéndose aplicar propiedades de grafos a la hora de escoger tareas para su ejecución.

Salvo el uso de implementaciones de Hardware de la biblioteca de tiempo de ejecución OpenMP (p.ej. en una matriz de puertas programables FPGAs), los sobrecostes de las tareas es mayor, este sobrecoste ha de ser amortizado mediante el potencial paralelismo adicional que las tareas exponen.

Estructura del Programa en C++ Ejemplo de cálculo de Pi usando OpenMP:

```
#include <stdio.h>
// Indica si se carga lo referente a OpenMP
#ifdef _OPENMP
#include <omp.h>
int threads=omp_get_num_threads();
#else
int threads=0;
#endif
#define STEPCOUNTER 1000000000
int main (void)
```

```
{
  long i;
  double pi=0;
  printf("threads %d", threads);
#pragma omp parallel for reduction(+:pi)
  for (i=0; i < STEPCOUNTER; i++)
  {
    pi += 1.0/(i*4.0 +1.0);
    pi -= 1.0/(i*4.0 +3.0);
  }
  pi = pi*4.0;
  printf("PI = %2.16lf ",pi);
  return 0;
}
```

El compilador de OpenMP es el mismo que para los lenguajes C, C++ o Fortran respectivamente. Por ello, para usarlo en C++ en línea de comandos, instalamos el compilador g++, mediante:

```
# apt install g++
```

Así, para compilar con g++[89], sin usar OpenMP, usamos:

```
$ g++ pi.cpp -o pi
```

Ejecutar midiendo el tiempo[90]:

```
$ time ./pi
```

[89] Compilar fuentes en C++ solicitando que el ejecutable tenga el nombre *ejemp*:

```
$ g++ *.cpp -o ejemp
```

en este caso no se usa ninguna directiva para optimizar el ejecutable generado. Para compilar usando diversas optimizaciones (O1, -O2 o -O3) usar por ejemplo:

```
$ g++ -O1 *.cpp
```

[90] Los sistemas operativos tipo Linux/Unix cuentan con un comando básico que mide el tiempo de ejecución de cualquier comando, usando:

```
$ time ls
```

Ahora, usando el compilar para OpenMP usamos:

$ g++ -o pi -fopenmp pi.cpp

Indicar el número de hilos, por ejemplo 2:

$ export OMP_NUM_THREADS=2

Ejecutar:

$ time ./pi

9.5.2 Programando con MPI para Memoria Distribuida

Para poder intercomunicar dos o más Cores en una o en múltiples computadoras se usa la «interfaz de paso de mensajes (Message Passing Interface MPI)» (véase [**?**], [**?**], [**?**] y [**?**]), una biblioteca de comunicación para procesamiento en paralelo que puede ser usada desde lenguajes de programación como: C, C++, Pytho, Julia, Fortran. MPI ha sido desarrollado como un estándar para el paso de mensajes y operaciones relacionadas.

Este enfoque es adoptado por usuarios e implementadores de bibliotecas, en el cual se proveen a los programas de procesamiento en paralelo de portabilidad y herramientas necesarias para desarrollar aplicaciones que puedan usar el cómputo paralelo de alto desempeño.

El modelo de paso de mensajes posibilita a un conjunto de procesos que tienen solo memoria local, la comunicación con otros procesos (usando Bus o red) mediante el envío y recepción de mensajes. Por definición el paso de mensajes posibilita transferir datos de la memoria local de un proceso a la memoria local de cualquier otro proceso que lo requiera.

En el modelo de paso de mensajes para equipos paralelos, los procesos se ejecutan en paralelo, teniendo direcciones de memoria separada para cada proceso, la comunicación ocurre cuando una porción de la dirección de memoria de un proceso es copiada mediante el envío de un mensaje dentro de otro proceso en la memoria local mediante la recepción del mismo.

---

pero podemos instalar el paquete que además de medir el tiempo de ejecución nos diga que recursos se usaron el la ejecución del comando indicado, para instalarlo usamos:

apt install time

el uso es el mismo del comando interno time.

Las operaciones de envío y recepción de mensajes es cooperativa y ocurre sólo cuando el primer proceso ejecuta una operación de envío y el segundo proceso ejecuta una operación de recepción, los argumentos base de estas funciones son:

- Para el que envía, la dirección de los datos a transmitir y el proceso destino al cual los datos se enviarán.
- Para el que recibe, debe tener la dirección de memoria donde se pondrán los datos recibidos, junto con la dirección del proceso que los envío.

Es decir:
Send(dir, lg, td, dest, etiq, com)

$\{dir, lg, td\}$ describe cuántas ocurrencias lg de elementos del tipo de dato td se transmitirán empezando en la dirección de memoria dir.

$\{des, etiq, com\}$ describe el identificador $etiq$ de destino des asociado con la comunicación com.

Recv(dir, mlg, td, fuent, etiq, com, st)

$\{dir, lg, td\}$ describe cuántas ocurrencias lg de elementos del tipo de dato td se transmitirán empezando en la dirección de memoria dir.

$\{fuent, etiq, com, est\}$ describe el identificador $etiq$ de la fuente $fuent$ asociado con la comunicación com y el estado est.

El conjunto básico de directivas (en nuestro caso sólo se usan estas) en C++ de MPI son (véase [**?**] y [**?**]):

MPI::Init	Inicializa al MPI
MPI::COMM_WORLD.Get_size	Busca el número de procesos existentes
MPI::COMM_WORLD.Get_rank	Busca el identificador del proceso
MPI::COMM_WORLD.Send	Envía un mensaje
MPI::COMM_WORLD.Recv	Recibe un mensaje
MPI::Finalize	Termina al MPI

Estructura del Programa en C++ Ejemplo de Hola_Mundo en MPI:

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
int rank, size;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
printf("Hola! Soy el %d de %d\n", rank, size);
MPI_Finalize();
return 0;
}
```

Otro ejemplo, para realizar una suma en MPI:

```
#include <iostream>
#include <iomanip>
#include <mpi.h>
using namespace std;
int main(int argc, char ** argv){
int mynode, totalnodes;
int sum = 0,startval,endval,accum;
MPI_Status status;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &totalnodes);
MPI_Comm_rank(MPI_COMM_WORLD, &mynode);
startval = 1000*mynode/totalnodes+1;
endval =1000*(mynode+1)/totalnodes;
for(int i=startval;i<=endval;i=i+1) sum = sum + i;
if(mynode!=0)
MPI_Send(&sum, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
else
for(int j=1;j<totalnodes;j=j+1){
MPI_Recv(&accum, 1, MPI_INT, j, 1,
MPI_COMM_WORLD, &status);
sum = sum + accum;
```

```
}
if(mynode == 0)
cout << "The sum from 1 to 1000 is: "<< sum << endl;
MPI_Finalize();
}
```

Existe una gran variedad de compiladores de MPI en línea de comandos, algunos disponibles en Debian GNU/Linux son instalados mediante:

```
# apt install lam-runtime libmpich-dev mpi-default-dev mpich\
mpi-default-bin openmpi-bin valgrind-mpi xmpi
```

Para compilar y ejecutar es posible usar alguna de estas opciones:

mpic++, mpic++.openmpi, mpiexec.mpich, mpif90.openmpi, mpirun.lam, mpicc, mpicxx, mpiexec.openmpi, mpifort, mpirun.mpich, mpiCC, mpicxx.mpich, mpif77, mpifort.mpich, mpirun.openmpi, mpicc.mpich, mpicxx.openmpi, mpif77.mpich, mpifort.openmpi, mpitask, mpicc.openmpi, mpiexec, mpif77.openmpi, mpimsg, mpivars, mpiCC.openmpi, mpiexec.hydra, mpif90, mpipython, mpichversion, mpipython, mpiexec.lam, mpif90.mpich, mpirun

Por ejemplo, para compilar *ejemp.cpp* en *mpic++* solicitando que el ejecutable tenga el nombre *ejemp*, usamos:

```
$ mpic++ ejemp.cpp -o ejemp
```

en este caso no se uso ninguna opción de optimización en tiempo de compilación, se puede hacer uso de ellas (-O1, -O2 o -O3), mediante:

```
$ mpic++ -O3 ejemp.cpp -o ejemp
```

para ejecutar el programa ya compilado en 4 procesos y medir el tiempo de ejecución, usamos:

```
$ time mpirun -np 4 ejemp
```

También podemos compilar *ejemp.c* en *mpicc* solicitando que el ejecutable tenga el nombre *ejemp*:

```
$ mpicc ejemp.cpp -o ejemp
```

en este caso no se uso ninguna opción de optimización en tiempo de compilación, se puede hacer uso de ellas (-O1, -O2 o -O3), mediante:

```
$ mpicc -O3 ejemp.c -o ejemp
```

para ejecutar el programa ya compilado en 4 procesos, usamos:

```
$ mpirun -np 4 ejemp
```

Un último ejemplo, en el caso de usar *mpiCC.mpic* y *lamboot*, entoces es necesario compilar usando:

```
$ mpiCC.mpich -O2 ejemp.cpp -o ejemp -lm
```

iniciar ambiente de ejecución paralelo:

```
$ lamboot -v
```

correr usando 8 procesos por ejemplo:

```
$ mpirun.mpich -np 8 ejemp
```

correr usando 5 procesos segun lista *machines.lolb* por ejemplo:

```
$ mpirun.mpich -machinefile machines.lolb -np 5 ejemp
```

terminar ambiente de ejecución paralelo:

```
$ lamhalt -v
```

Observación 2 *Para que en la ejecución de MPI no pida la clave de usuario:*
$ ssh-keygen -t rsa
En cada pregunta responder con ENTER, para después copiar usando:
$ ssh-copy-id usuario@servidor
Ojo: Si continúa pidiendo clave, es que esta instalado rsh o lsh.

9.5.3 Esquema de Paralelización Principal-Subordinados

El esquema de paralelización Principal-Subordinados -también conocido como Maestro-Esclavo-, permite sincronizar por parte del nodo principal las tareas que se realizan en paralelo usando varios nodos subordinados, éste modelo puede ser explotado de manera eficiente si existe poca comunicación entre el principal y el subordinado -entre los subordinados no debe de existir comunicación- y los tiempos consumidos en realizar las tareas asignadas son mayores que los períodos involucrados en las comunicaciones para la asignación de dichas tareas. De esta manera se garantiza que la mayoría de los procesadores estarán trabajando de manera continua y existirán pocos tiempos muertos.

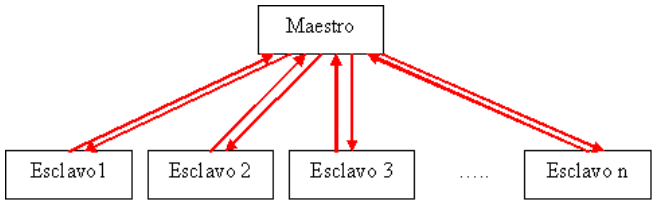


Figura 13: Esquema del Maestro-Esclavo

Donde, tomando en cuenta la implementación en estrella del Cluster, el modelo de paralelismo de MPI (véase 9.5.2) y las necesidades propias de comunicación del programa, el nodo principal tendrá comunicación sólo con cada nodo subordinado y no existirá comunicación entre los nodos subordinados, esto reducirá las comunicaciones y optimizará el paso de mensajes, algunos ejemplos de este esquema se pueden consultar en:

Herramientas
http://132.248.181.216/Herramientas/

Un factor limitante en este esquema es que el nodo principal deberá de atender todas las peticiones hechas por cada uno de los nodos subordinados, esto toma especial relevancia cuando todos o casi todos los nodos subordinados compiten por ser atendidos por el nodo principal.

Se recomienda implementar este esquema en un Cluster heterogéneo en donde el nodo principal sea más poderoso computacionalmente que los nodos subordinados. Si a este esquema se le agrega una red de alta velocidad y de

baja latencia, se le permitirá operar al Cluster en las mejores condiciones posibles, pero este esquema se verá degradado al aumentar el número de nodos subordinados inexorablemente.

Pero hay que ser cuidadosos en cuanto al número de nodos subordinados que se usan en la implementación en tiempo de ejecución versus el rendimiento general del sistema al aumentar estos, algunas observaciones posibles son:

- El esquema Principal-Subordinados programado usando MPI lanza P procesos (uno para el nodo principal y $P-1$ para los nodos subordinados), estos en principio corren en un solo procesador pero pueden ser lanzados en múltiples procesadores usando una directiva de ejecución, de esta manera es posible que en una sola máquina se programe, depure y sea puesto a punto el código usando mallas pequeñas (del orden de cientos de nodos) y cuando este listo puede mandarse a producción en un Cluster.

- El esquema Principal-Subordinados no es eficiente si sólo se usan dos procesadores (uno para el nodo principal y otro para el nodo subordinado), ya que el nodo principal en general no realiza los cálculos pesados y su principal función será la de distribuir tareas; los cálculos serán delegados al nodo subordinado.

Estructura del Programa Principal-Subordinados La estructura del programa se realizo para que el nodo principal mande trabajos de manera síncrona a los nodos subordinados. Cuando los nodos subordinados terminan la tarea asignada, avisan al nodo principal para que se les asigne otra tarea (estas tareas son acordes a la etapa correspondiente del método de descomposición de dominio ejecutándose en un instante dado). En la medida de lo posible se trata de mandar paquetes de datos a cada nodo subordinado y que estos regresen también paquetes al nodo principal, a manera de reducir las comunicaciones al mínimo y tratar de mantener siempre ocupados a los nodos subordinados para evitar los tiempos muertos, logrando con ello una granularidad gruesa, ideal para trabajar con Clusters.

La estructura básica del programa bajo el esquema Principal-Subordinados codificada en C++ y usando MPI (véase 9.5.2) es:

```
main(int argc, char *argv[])
```

```
{
   MPI::Init(argc,argv);
   ME_id = MPI::COMM_WORLD.Get_rank();
   MP_np = MPI::COMM_WORLD.Get_size();
   if (ME_id == 0) {
     // Operaciones del Principal
   } else {
     // Operaciones del Subordinado con identificador ME_id
    }
   MPI::Finalize();
}
```

En este único programa se deberá de codificar todas las tareas necesarias para el nodo principal y cada uno de los nodos subordinados, así como las formas de intercomunicación entre ellos usando como distintivo de los distintos procesos a la variable *ME_id*. Para más detalles de esta forma de programación y otras funciones de MPI (véase 9.5.2, [**?**] y [**?**]).

El principal factor limitante para el esquema Principal-Subordinados es que se presupone contar con un nodo principal lo suficientemente poderoso para atender simultáneamente las tareas síncronas del método, ya que este distribuye tareas acorde al número de nodos subordinados, estas si son balanceadas, ocasionaran que muchos de los procesadores subordinados terminen aproximadamente al mismo tiempo y el nodo principal tendrá que atender múltiples comunicaciones simultáneamente, degradando su rendimiento al aumentar el número de nodos subordinados que atender.

Para los factores limitantes inherente al propio esquema Principal-Subordinados, es posible implementar algunas operaciones del nodo principal en paralelo, ya sea usando equipos multiprocesador o en más de un nodo distinto a los nodos subordinados.

9.5.4 Opciones de Paralelización Híbridas

En la actualidad, casi todos los equipos de cómputo usados en estaciones de trabajo y Clusters cuentan con dos o más Cores, en ellos siempre es posible usar MPI para intercambiar mensajes entre procesos corriendo en

el mismo equipo de cómputo, pero no es un proceso tan eficiente como se puede querer. En estas arquitecturas llamadas de memoria compartida es mejor usar OpenMP o cualquiera de sus variantes para trabajar en paralelo. Por otro lado es común contar con las cada vez más omnipresentes tarjetas NVIDIA, y con los cada vez más numerosos Cores CUDA -que una sola tarjeta NVIDIA TESLA puede tener del orden de miles de ellos- y que en un futuro serán cada vez más numerosos.

Para lograr obtener la mayor eficiencia posible de estos tres niveles de paralelización, se están implementando procesos híbridos (véase [**?**] y [**?**]), en donde la intercomunicación de equipos con memoria compartida se realiza mediante MPI y la intercomunicación entre Cores que comparten la misma memoria se realiza con OpenMP, además las operaciones matriciales, vectoriales, etc. se le encargan a los numerosos Cores CUDA de las tarjetas NVIDIA.

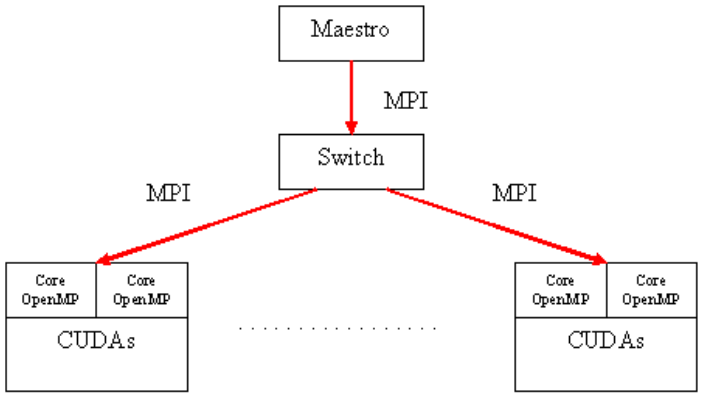


Figura 14: Paralelización Híbrida

Se han desarrollado múltiples programas que sus algoritmos resuelven ecuaciones diferenciales parciales concomitantes en Ciencias e Ingenierías que hacen uso de esta forma integradora de paralelismo. Para ello, la interconexión de equipos de memoria compartida se realizaría mediante MPI y en cada equipo de memoria compartida se manipularían uno o más subdominios mediante OpenMP -ya que cada subdominio es independiente de los demás- y la manipulación de matrices y operaciones entre matrices y vectores que requiere cada subdominio se realizarían en las tarjetas NVIDIA mediante los numerosos Cores CUDA sin salir a la RAM de la computadora.

Permitiendo así, tener una creciente eficiencia de paralelización que optimizan en gran medida los recursos computacionales, ya que todas las matrices y vectores se generarían en la GRAM de la tarjeta NVIDIA. De forma tal que sea reutilizable y que pueda usarse en problemas en los que el número de grados de libertad sea grande, permitiendo hacer uso de equipos de cómputo cada vez más asequibles y de menor costo, pero con una creciente eficiencia computacional que compiten con los grandes equipos de cómputo de alto desempeño.

9.6 Programando Desde la Nube

Existen diferentes servicios Web[91] que permiten editar, compilar y ejecutar código de diversos lenguajes y paquetes desde el navegador, esto en aras de que los estudiantes y profesores que cuenten con algún sistema de acceso a red y un navegador puedan programar en los más diversos lenguajes, IDEs y terminales sin hacer instalación alguna en su equipo de cómputo, tableta o teléfono celular.

Google Colaboratory Integrante de la G Suite for Education de Google permite a los usuarios que pertenezcan a esta Suite (como gran parte de los estudiantes de la UNAM) tener acceso desde el navegador para escribir y ejecutar código de Python (Jupyter), es posible elegir correr nuestro Notebook en una CPU, GPU o en una TPU de forma gratuita. Tiene algunas restricciones, como por ejemplo que una sesión dura 12 hrs, pasado ese tiempo se limpia nuestro ambiente y perdemos las variables y archivos que tengamos almacenados allí.

Es conveniente para principiantes que requieran experimentar con Machne Learning y Deep Learning pero sin recurrir en costos de procesamiento Cloud. Además el ambiente de trabajo ya viene con muchas librerías instaladas y listas para utilizar (como por ejemplo *Tensorflow, Sclkit-learn, Pytorch, Keras* y *OpenCV*), ahorrándonos el trabajo de configurar nuestro ambiente

[91] Cuando se trabaja desde la Web es recomendable usar el modo Privado o Incógnito para no guardar el historial de navegación, información introducida en los formularios y borrar al cerrar el navegador los datos de los sitios visitados. Pero recuerda que los sitios Web que visitamos sí guardan información de nuestra visita, nuestro proveedor de Internet también guarda constancia de nuestra visita y si descargamos algo, esto no se borra al igual que el historial de descargas, además de las marcas de páginas o favoritos se conservarán al cerrar el navegador.

de trabajo. Podemos importar nuestros archivos y datos desde *Google Drive, GitHub*, etc.

Más información sobre Google Colaboratory en:

https://colab.research.google.com/notebooks/intro.ipynb

10 Apéndice D: Software Libre y Propietario

Con el constante aumento de la comercialización de equipos de cómputo y/o comunicación (teléfonos inteligentes, tabletas, computadoras portátiles y de escritorio, etc.) y su relativo bajo costo, estos equipos se han convertido en objetos omnipresentes en nuestra vida diaria, ya que estos permiten realizar un creciente número de actividades cotidianas de miles de millones de usuarios.

Dichos equipos de cómputo y/o comunicación por sí solos tienen poca utilidad, pero su uso en conjunción con el Software adecuado forman un dúo que nos ha permitido tener los avances de los que actualmente disfrutamos. El Software -sistema operativo y los programas de aplicaciones- son los que realmente generan las soluciones al interactuar uno o más paquetes informáticos con los datos del usuario. También, es común que al comprar un equipo de cómputo y/o comunicación, en el costo total, se integre el del sistema operativo, aplicaciones ofimáticas y de antivirus, sean estos usados por el usuario o no y en la mayoría de los casos no es posible solicitar que no sean incluidos en el costo del equipo.

Por otro lado, el Software comercial suele quedar obsoleto muy rápido, ya que constantemente se le agregan nuevas funcionalidades al mismo y estas en general son vendidas como versiones independientes de la adquirida originalmente. Esto obliga al usuario -si quiere hacer uso de ellas- a comprar las nuevas versiones del Software para satisfacer sus crecientes necesidades informáticas y la obsolescencia programada.

Por lo anterior y dada la creciente complejidad de los paquetes de cómputo y el alto costo de desarrollo de aplicaciones innovadoras, en muchos casos, el costo total del Software que comúnmente los usuarios instalan -y que no necesariamente usan las capacidades avanzadas del programa, por las cuales el Software tiene un alto costo comercial- en sus equipos, suele ser más caro que el propio equipo en el que se ejecutan.

Hoy en día los usuarios disponemos de dos grandes opciones para adquirir el Software necesario para que nuestros equipos funcionen, a saber:

- Por un lado, podemos emplear programas comerciales (Software propietario), de los cuales no somos dueños del Software, sólo concesionarios al adquirir una licencia de uso del Software y nos proporcionan un instalable del programa adquirido. La licencia respectiva es en la gran mayoría de los casos muy restrictiva, ya que restringe su uso a un solo

equipo y/o usuario simultáneamente.

- Por otro lado, existe el Software libre[92], desarrollado por usuarios y para usuarios que, entre otras cosas, comparten los códigos fuente, el programa ejecutable y dan libertades para estudiar, adaptar y redistribuir a quien así lo requiera el programa y todos sus derivados.

Sobre la Obsolescencia Programada Es un conjunto de estrategias deliberadas destinadas a asegurarse que la versión actual de un determinado producto quedará desfasada o inservible en un plazo de tiempo predeterminado. De esta manera, los fabricantes se aseguran que los consumidores se verán obligados a reemplazarlo aunque funcione adecuadamente.

La obsolescencia puede lograrse mediante la introducción de un modelo con características superiores o diseñando intencionadamente un producto para que deje de funcionar correctamente en un plazo determinado. En cualquiera de los dos casos, se espera que los consumidores opten por el nuevo producto de la misma marca. Muchas veces la obsolescencia no es sobre el propio producto sino aplicando restricciones al producto de un competidor con la ayuda de una tercera empresa.

Tipos de Obsolescencia Programada Podemos dividir la obsolescencia programada en 4 tipos:

1- Establecimiento artificial del plazo de duración: Los productos se fabrican con piezas cuya duración tienen una vida útil limitada cuando, si se usaran otras de calidad superior ese plazo se extendería.

2- Actualizaciones de Software: Los desarrolladores de Software sacan nuevas versiones de sus aplicaciones que en un momento determinado dejan de ser compatibles con dispositivos antiguos. En muchos casos se ha podido comprobar que esa incompatibilidad es absolutamente artificial ya que al «engañar» al Software este funcionaba sin problemas.

[92] A veces también se han usado términos como FOSS y FLOSS. Ambas cosas son similares, ya que FOSS (Free and Open Source Software) traducido como "Software de código abierto" y FLOSS (Free/Libre and Open Source Software) "Software libre y de código abierto". Según quienes adoptan estos términos, lo hacen por tener una imparcialidad entre la carga filosófica del Software libre y el aspecto técnico y/o las ventajas que brinda este modelo de desarrollo. Richard Stallman nos invita a no usarlas y no se trata de un ad hóminem. Stallman y el proyecto GNU nos aconsejan que hablemos siempre de Software libre y aquí no cabe imparcialidad.

3- Obsolescencia percibida: Esta es una táctica psicológica, se trata de convencer al consumidor mediante publicidad y el uso de influenciadores de que el producto que se tiene actualmente está viejo y que se necesita uno nuevo. Como por ejemplo: ¿cuantos megapíxeles necesitas en tu teléfono para sacar una buena foto de tu mascota?

4- Trabas a la reparación: En el caso de los teléfonos por ejemplo, lo de impedir sacar la batería (con la excusa de hacer los teléfonos más delgados) es una forma de obligar a los consumidores a recurrir a los servicios oficiales y a disuadirlos de reemplazarlas por sustitutos más económicos. Otras tácticas son la utilización de piezas no estándar o que necesitan herramientas específicas para la reparación. Muchas veces se suele restringir el acceso a estas piezas o hacer una reducida producción de las mismas para aumentar artificialmente el costo.

Ejemplos de Obsolescencia Programada

- iPhone cada vez más lentos: La Justicia francesa comprobó que actualizaciones de Software hacían cada vez más lento el rendimiento de los modelos más viejos. La empresa le echó la culpa a las baterías, pero pagó una compensación de decenas de millones de dólares. Además rebajó los precios de sus baterías de repuesto para que los teléfonos fueran más rápidos con el nuevo Software y se comprometió a hacer más en el futuro para garantizar que los teléfonos no volvieran a ser más lentos. Con la salida de un nuevo modelo de teléfono cada año, seguro que hay algo de obsolescencia planificada en alguna parte.

- Impresoras: Esto es algo que todos conocemos. Muchas veces nos encontramos con impresoras a precio rebajado, pero al momento de tener que comprar un cartucho de tinta nos encontramos con que este tiene un precio igual o superior a comprar una nueva. Además, se ponen restricciones a la recarga o al uso de cartuchos alternativos. Hubo denuncias de que algunos modelos dejaban de funcionar a partir de cierta cantidad de páginas impresas o cierto tiempo desde la primera impresión.

- Certificados de seguridad: Por ejemplo, el pasado 30 de septiembre de 2021 caduco otro certificado de autentificación (CA de DST Root CA X3 de Let's Encrypt) que ayudaba a validar la conexión en internet a

los dispositivos que no fueron actualizados a otro certificado más actual -en la mayoría de los casos por no ser del interés económico de sus creadores-. Esto ocasionó que millones de dispositivos (teléfonos inteligentes, Smart TV, tabletas, computadoras portátiles y de escritorio, etc.) con algunos años de ser creados y perfectamente funcionales dejarán de conectarse a internet de un día para otro, forzando a sus dueños a desechar el dispositivo por carecer del servicio de internet en las aplicaciones instaladas.

- Cambio de la versión del sistema operativo: En el caso del sistema operativo Windows 10 a 11, la solicitud de requisitos mínimos de Hardware es para muchos equipos excesivo, ya que se estima que dejará fuera en su actualización a casi todos los equipos con más de 4 años de antigüedad por no contar por ejemplo con el Chip TPM 2.0 o GPU compatible con DirectX 12, siendo perfectamente funcionales con la versión actual del sistema operativo. Si bien Windows 10 seguirá con soporte hasta 2025, los usuarios que deseen tener las nuevas características del sistema operativo tendrán que cambiar de equipo.

10.1 Software Propietario

No existe consenso sobre el término a utilizar para referirse al opuesto del Software libre. La expresión «Software propietario (Proprietary Software)» (véase [**?**]), en la lengua anglosajona, "Proprietary" significa «poseído o controlado privadamente (Privately Owned and Controlled)», que destaca la manutención de la reserva de derechos sobre el uso, modificación o redistribución del Software. Inicialmente utilizado, pero con el inconveniente de que la acepción proviene de una traducción literal del inglés, no correspondiendo su uso como adjetivo en el español, de manera que puede ser considerado como un barbarismo.

El término "propietario" en español resultaría inadecuado, pues significa que «tiene derecho de propiedad sobre una cosa», por lo que no podría calificarse de "propietario" al Software, porque éste no tiene propiedad sobre nada (es decir, no es dueño de nada) y además, no podría serlo (porque es una cosa y no una persona). Así mismo, la expresión "Software propietario" podría ser interpretada como: "Software sujeto a propiedad" (derechos o titularidad) y su opuesto, el Software libre, también está sujeto al derecho de autor. Otra interpretación es que contrariamente al uso popular del término, se puede

afirmar que "todo Software es propietario", por lo que la forma correcta de referirse al Software con restricciones de uso, estudio, copia o mejora es la de Software privativo, según esta interpretación el término "propietario" podría aplicarse tanto para Software libre como Software privativo, ya que la diferencia entre uno y otro está en que el dueño del Software privativo lo licencia como propiedad privada y el de Software libre como propiedad social.

Con la intención de corregir el defecto de la expresión "Software propietario" aparece el llamado "Software con propietario", sin embargo se argumenta contra el término "con propietario" y justamente su similitud con Proprietary en inglés, que sólo haría referencia a un aspecto del Software que no es libre, manteniendo una de las principales críticas a éste (de "Software sujeto a derechos" o "propiedad"). Adicionalmente, si "propietario" se refiere al titular de los derechos de autor -y está claro que no se puede referir al usuario, en tanto éste es simplemente un cesionario-, no resuelve la contradicción: todo el Software libre tiene también titulares de derechos de autor.

La expresión Software no libre (en inglés Non-Free Software) es usado por la FSF para agrupar todo el Software que no es libre, es decir, incluye al llamado en inglés "Semi-Free Software" (Software semilibre) y al "Propietary Software". Asimismo, es frecuentemente utilizado para referirse al Software que no cumple con las Directrices de Software libre de Debian GNU/Linux, las cuales siguen la misma idea básica de libertad en el Software, propugnada por la FSF y sobre las cuales está basada la definición de código abierto de la Open Source Initiative.

Adicionalmente el Software de código cerrado nace como antónimo de Software de código abierto y por lo tanto se centra más en el aspecto de ausencia de acceso al código que en los derechos sobre el mismo, éste se refiere sólo a la ausencia de una sola libertad por lo que su uso debe enfocarse sólo a este tipo de Software y aunque siempre signifique que es un Software que no es libre, no tiene que ser Software de código cerrado.

La expresión Software privado es usada por la relación entre los conceptos de tener y ser privado. Este término sería inadecuado debido a que, en una de sus acepciones, la palabra "privado" se entiende como antónimo de "público", es decir, que «no es de propiedad pública o estatal, sino que pertenece a particulares», provocando que esta categoría se interpretará como no referente al Estado, lo que produciría la exclusión del Software no libre generado por el aparato estatal. Además, el "Software público" se asocia generalmente con Software de dominio público.

10.2 Software Libre

La definición de Software libre (véase [?], [?], [?], [?], [?] y [?]) estipula los criterios que se tienen que cumplir para que un programa sea considerado libre. De vez en cuando se modifica esta definición para clarificarla o para resolver problemas sobre cuestiones delicadas. «Software libre» significa que el Software respeta la libertad de los usuarios y la comunidad. En términos generales, los usuarios tienen la libertad de copiar, distribuir, estudiar, modificar y mejorar el Software. Con estas libertades, los usuarios -tanto individualmente como en forma colectiva- controlan el programa y lo que hace.

Cuando los usuarios no controlan el programa, el programa controla a los usuarios. Los programadores controlan el programa y a través del programa, controlan a los usuarios. Un programa que no es libre, llamado «privativo o propietario», es considerado por muchos como un instrumento de poder injusto.

El Software libre es la denominación del Software que respeta la libertad de todos los usuarios que adquirieron el producto y por tanto, una vez obtenido el mismo puede ser usado, copiado, estudiado, modificado y redistribuido libremente de varias formas. Según la Free Software Foundation (véase [?]), el Software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir y estudiar el mismo, e incluso modificar el Software y distribuirlo modificado.

Un programa es Software libre si los usuarios tienen las cuatro libertades esenciales:

0. La libertad de usar el programa, con cualquier propósito.

1. La libertad de estudiar cómo funciona el programa y modificarlo, adaptándolo a tus necesidades.

2. La libertad de distribuir copias del programa, con lo cual puedes ayudar a tu prójimo.

3. La libertad de mejorar el programa y hacer públicas esas mejoras a los demás, de modo que toda la comunidad se beneficie.

Un programa es Software libre si los usuarios tienen todas esas libertades. Por tanto, el usuario debe ser libre de redistribuir copias, tanto con o sin modificaciones, ya sea gratuitamente o cobrando una tarifa por la distribución,

a cualquiera en cualquier parte. El ser libre de hacer estas cosas significa, entre otras cosas, que no tiene que pedir ni pagar el permiso.

También debe tener la libertad de hacer modificaciones y usarlas en privado para su propio trabajo o pasatiempo, sin siquiera mencionar que existen. Si publica sus cambios, no debe estar obligado a notificarlo a nadie en particular, ni de ninguna manera.

La libertad de ejecutar el programa significa que cualquier tipo de persona u organización es libre de usarlo en cualquier tipo de sistema de computación, para cualquier tipo de trabajo y finalidad, sin que exista obligación alguna de comunicarlo al programador ni a ninguna otra entidad específica. En esta libertad, lo que importa es el propósito de los usuarios, no el de los programadores. El usuario es libre de ejecutar el programa para alcanzar sus propósitos y si lo distribuye a otra persona, también esa persona será libre de ejecutarlo para lo que necesite; nadie tiene derecho a imponer sus propios objetivos.

La libertad de redistribuir copias debe incluir las formas binarias o ejecutables del programa, así como el código fuente, tanto para las versiones modificadas como para las que no lo estén. Distribuir programas en forma de ejecutables es necesario para que los sistemas operativos libres se puedan instalar fácilmente. Resulta aceptable si no existe un modo de producir un formato binario o ejecutable para un programa específico, dado que algunos lenguajes no incorporan esa característica, pero debe tener la libertad de redistribuir dichos formatos si encontrara o programara una forma de hacerlo.

Para que se de la libertad que se menciona en los puntos 1 y 3 de realizar cambios y publicar las versiones modificadas tenga sentido, el usuario debe tener acceso al código fuente del programa. Por consiguiente, el acceso al código fuente es una condición necesaria para el Software libre. El «código fuente» compilado no es código fuente real y no cuenta como código fuente.

La libertad 1 incluye la libertad de usar su versión modificada en lugar de la original. Si el programa se entrega con un producto diseñado para ejecutar versiones modificadas de terceros, pero rechaza ejecutar las suyas, una práctica conocida como «tivoización» o «arranque seguro» [«Lockdown»] la libertad 1 se convierte más en una ficción teórica que en una libertad práctica, esto no es suficiente, en otras palabras, estos binarios no son Software libre, incluso si se compilaron desde un código fuente que es libre.

Una manera importante de modificar el programa es agregándole subrutinas y módulos libres ya disponibles. Si la licencia del programa especifica que no se pueden añadir módulos que ya existen y que están bajo una licencia

apropiada, por ejemplo si requiere que usted sea el titular de los derechos de autor del código que desea añadir, entonces se trata de una licencia demasiado restrictiva como para considerarla libre.

La libertad 3 incluye la libertad de publicar sus versiones modificadas como Software libre. Una licencia libre también puede permitir otras formas de publicarlas; en otras palabras, no tiene que ser una licencia de Copyleft. No obstante, una licencia que requiera que las versiones modificadas no sean libres, no se puede considerar libre.

«Software libre» no significa que «no es comercial». Un programa libre debe estar disponible para el uso comercial, la programación comercial y la distribución comercial. La programación comercial de Software libre ya no es inusual; el Software libre comercial es muy importante, ejemplo de ello es la empresa RedHat (ahora propiedad de IBM). Puede haber pagado dinero para obtener copias de Software libre, o puede haber obtenido copias sin costo. Pero sin tener en cuenta cómo obtuvo sus copias, siempre tiene la libertad de copiar y modificar el Software, incluso de vender copias.

El término Software no libre se emplea para referirse al Software distribuido bajo una licencia de Software más restrictiva que no garantiza estas cuatro libertades. Las leyes de la propiedad intelectual reservan la mayoría de los derechos de modificación, duplicación y redistribución para el dueño del Copyright; el Software dispuesto bajo una licencia de Software libre rescinde específicamente la mayoría de estos derechos reservados.

Los manuales de Software deben ser libres por las mismas razones que el Software debe ser libre y porque de hecho los manuales son parte del Software. También tiene sentido aplicar los mismos argumentos a otros tipos de obras de uso práctico, es decir, obras que incorporen conocimiento útil, tal como publicaciones educativas y de referencia. Wikipedia es el ejemplo más conocido.

La lista de proyectos de este tipo es realmente impresionante, algunos han conseguido un uso y alta calidad, por ejemplo el compilador GCC, el Kernel de Linux y el sistema operativo Debian GNU/Linux y Android. Mientras que otros proyectos han caído en el olvido, pero de la gran mayoría se tiene copia del código fuente que permitiría a quienes estén interesados en dicho proyecto poder reusarlo y en su caso ampliarlo.

La característica más importante que aparece típicamente en un proyecto de este tipo, es que un conjunto de personas separadas a gran distancia, sean capaces, a través de la Web, de los E-mail y de foros de aunar sus esfuerzos para crear, mejorar y distribuir un producto, de forma que todos

ellos se benefician unos de otros. Evidentemente, la mayor parte del peso recae en los desarrolladores, pero también es necesaria una difusión para que los usuarios documenten, encuentren errores, hagan foros de discusión, etc.

Si bien, el Software libre no es más seguro (en el sentido de invulnerable) que el propietario, la diferencia estriba en que el código fuente en el Software libre está disponible para todos y cualquiera puede aportar una solución, y por lo general al poco tiempo de detectarse una vulnerabilidad (a veces en cuestión de horas) se puede disponer de una solución para la misma. Además, al tener acceso al código fuente se puede detectar fácilmente si alguien introdujo código malicioso a una determinada aplicación.

¿Por qué se Interesan los Autores, Alumnos y Profesores Universitarios en el Software Libre? La ventaja principal es porque bajo el Software libre subyace la idea de compartir conocimiento y favorecer la existencia de nuevas ideas[93]; y ¿qué es investigar y enseñar?, sino crear conocimiento y procurar que los alumnos aprendan e incluso vayan más allá de lo aprendido. Se comparte la idea, que el espíritu del Software libre es similar al que debería reinar en las instituciones educativas:

- Porque así no se condiciona a los estudiantes a usar siempre lo mismo.
- No se fomenta la piratería en los estudiantes y se evita pagar licencias que no son necesarias al existir alternativas gratuitas.
- Es mucho más seguro ya que el Software libre es público y se puede ver qué hace exactamente sin recelos.
- Se ofrece libertad de elección a los estudiantes y profesores al no limitarlos a usar una solución determinada, ampliando sus opciones y permitiendo un mayor aprendizaje.

Concretando estas ideas, profesores e investigadores necesitan herramientas para la investigación y docencia y estas deben tener una calidad mínima y ser fácilmente distribuibles entre los alumnos. En muchos casos las compañías desarrolladoras y distribuidoras de programas de cómputo no han

[93] ¿Por qué el Software creado con dinero de los impuestos no se publica como Software Libre?

¡El código pagado por los ciudadanos debería estar disponible para los ciudadanos y el mismo gobierno!

sabido ofrecer sus productos con la flexibilidad adecuada para las labores docentes o, en otros casos, los productos desarrollados no tienen la calidad esperada.

El Software libre es aún joven, pese a las decenas de miles de proyectos actuales -en los que se trabaja constantemente en mejorar la parte computacional de los algoritmos involucrados en el proyecto, haciendo y puliendo interfaces gráficas, generando ayuda en línea así como la documentación necesaria para que usuarios noveles y avanzados usen la mayor cantidad de opciones programadas- existen muchas otras necesidades profesionales y de investigación que requieren el desarrollo innovador de programas de cómputo para automatizarlas y hacerlas eficientes. Esto queda plasmado en las decenas de proyectos que a diario son registrados en las páginas especializadas en busca de difusión y apoyo para su proyecto.

En los últimos años, muchos proyectos han pasado de ser simples programas en línea de comandos a complejas aplicaciones multiplataforma -se ejecutan en distintos sistemas operativos como son Windows, Linux, Unix, Mac OS, Android- con ambientes gráficos multimedia que en muchos casos han superado a sus contrapartes comerciales -por ejemplo los navegadores Web-. Para muestra de este maravilloso avance, tomemos el proyecto del sistema operativo Android, que actualmente se ejecuta en millones de equipos -como celulares, tabletas, electrodomésticos, etc.- y en los cuales se pueden descargar miles de aplicaciones y está soportado por una gran cantidad de usuarios y empresas comerciales como Google, IBM y últimamente Microsoft -que años atrás era acérrima enemiga del Software libre-.

El Software libre ha logrado desplazar a muchos de sus competidores por sus múltiples bondades y bajo costo de desarrollo -es el caso de Windows Phone que fue reemplazado por Android de Google-, al reusar miles de aplicaciones ya existentes que usan Software libre y permitir desarrollar otro tanto de aplicaciones bajo una plataforma que se ejecuta en los más diversos procesadores. Además, el uso de Software libre y su posibilidad de ampliarlo y/o especializarlo según sea necesario, ha permitido crear de forma cada vez más rápida y confiable; para poner a disposición de un gran público programas de uso común, así como especializado que satisfagan las nuevas necesidades de los usuarios.

Software Libre en Ciencia y Educación Algunos puntos y reflexiones sobre porqué se considera que es interesante el Software libre en Ciencia y

Educación son:

- Accesible a todo el mundo aunque no sea rentable su desarrollo: El Software libre entre sus libertades permite que se pueda ejecutar por terceros, copiarlo, distribuirlo y estudiarlo/modificarlo. Eso hace que si en ciencia se usa Software libre el acceso a esos programas no suponga una barrera (se puede distribuir y ejecutar).

- Muchos de los desarrollos en el campo de la accesibilidad se realizan en universidades y son distribuidos como Software libre: Aunque no sea rentable muchas veces el desarrollo de herramientas de accesibilidad (no sea algo monetizable) en la universidad se consigue escapar a esa la lógica capitalista de solamente invertir en lo que pueda ofrecer beneficio económico.

- Transparencia: En ciencia es importante ver las costuras para comprobar si es verdad lo que se afirma, tener acceso al código fuente del Software empleado permite poder estudiarlo por si realiza algún cálculo mal.

- Propicia el espíritu crítico: Si no tienes acceso a las revistas o el acceso es privativo para los bolsillos de mucha gente no puedes comprobar la información. Se nos pide que seamos críticos con la información que se nos da del mundo científico pero no podemos entrenar el espíritu crítico sin acceso al conocimiento libre.

- Caramelos con droga en la puerta del colegio: Muchas empresas buscan introducir en los colegios, institutos y universidad su Software. Ofrecer un programa que permita trabajar y genere una dependencia quedando los datos muchas veces en las nubes (ordenadores de otras personas). Un ejemplo es Microsoft con Office 365. Dando cuentas gratuitas durante un tiempo para que se use su Software. Otro ejemplo podría ser Unity3D en vez de por ejemplo Godot.

- Especificaciones de protocolos abiertas VS cerradas: Gracias a que los protocolos TCP/IP, HTTP, POP, SNMP, DHCP, etc. son abiertos es posible construir herramientas por cualquier con conocimientos de programación. Con protocolos cerrados solamente quienes tuvieran acceso a las especificaciones podrían desarrollar y conocer cómo funcionan.

- Uso de estándares: Existiendo un estándar para documentos ofimáticos (procesador de textos, hoja de cálculo, presentaciones, etc.) algunas empresas como Microsoft se empeñan en ir con su propio formato y estándar en vez de sumarse a que sea más sencillo ir a una y que el usuario pueda optar por que herramientas usar para editar o trabajar con documentos ofimáticos.

- Software libre para la Ciencia ciudadana: Un ejemplo en el que es importante la colaboración ciudadana es el cambio climático y la defensa medioambiental del territorio. Desde imvec.tech usan herramientas de Software libre para medición y monitorización de contaminación.

Software Libre: Beneficios Más Allá de la Informática El uso de las tecnologías de código abierto supone cultivar el conocimiento y la puesta en valor de la libertad individual, lo personal y lo privado, todo ello sin menospreciar lo público y la construcción de una sociedad. El movimiento del Software libre, con Linux a la cabeza, ha capitaneado durante décadas planteamientos para un cambio en el modo de producción: el abaratamiento de costes empresariales para grandes y pequeños, el trabajo en línea, el desarrollo de Software y Hardware a pequeña escala, el replanteamiento del negocio informático, el sistema de normas éticas que rigen los grupos, la documentación abierta, etc.

Todo ello derivado de una simple idea: la libertad. Ha sido la clave que ha llevado a todo este movimiento hacia una autonomía y motivación que pocas veces se ve en otros sectores. El Open Source está lleno de alternativas con la libertad como pilar y consecuencia filosófica, de hecho muchos Forks (derivaciones) de proyectos aparecen cuando la disputa sobre la misma toma relevancia. Esta manera de hacer las cosas debería trasladarse directamente a la sociedad promoviendo esos valores para evitar caer en un mundo despótico que toma fuerza a pasos agigantados.

No estaría mal promover la comprensión de las licencias libres. Leer y entender una licencia GPL, MIT o BSD es infinitamente más sencillo y rápido que hacerlo con otras, siendo unas normas fáciles de cumplir porque encajan con un modelo ético y práctico comprensible por muchos.

Podríamos decir que GPL, BSD y MIT son las Constituciones que vertebran todo el movimiento del Software Libre, cosechando derechos de uso y logros como el ahorro o la legítima copia privada. Lo mismo podría ocurrir con las licencias Creative Commons para cierto contenido, un arma poderosa

en el sector divulgativo que está poco extendida por desconocimiento y el Status Quo de la propiedad intelectual.

La cooperación libre y voluntaria supuso un éxito hasta ahora pero está siendo amenazada por la dinámica actual de la Web, donde la centralización de las principales plataformas supone estar bajo el yugo de normas que rastrean con lupa y cambian sus términos con demasiada frecuencia. Ya ni digamos cuando eso se junta con el ansia de monetización: si quieres dinero más te vale no cabrear a los anunciantes o no tener un Strike por uso de contenido que podría ser reclamado.

Los claroscuros de este sistema hace que los creadores cada vez hagan menos de forma libre y altruista, y ello podría verse potenciado por las búsquedas con inteligencia artificial, lo que apunta a un empobrecimiento del contenido cultural fresco y renovador. Las polémicas con GitHub Copilot o ChatGPT sobre el entrenamiento de las IAs hace sobrevolar una vez más la cuestión del Copyright y el uso legítimo de los resultados brindados por éstas, lo que también condiciona la creación.

La libertad de expresión, de uso, de creación, de modificación ... esas cuestiones llevan irremediablemente a una libertad de pensamiento y acción, a una adaptación creativa que puede ser la motivación para romper moldes en todos los aspectos. El código abierto y sus licencias son, por tanto, un beneficio personal y social mucho más grande que el simple hecho de usar Linux o Software libre. El contenido despreocupado, que no prioriza el dinero y el posicionamiento/visualizaciones, se vuelve esencial para el inconformismo.

10.3 Seguridad del Software

Si bien, el Software Libre no es más seguro (en el sentido de invulnerable) que el propietario, la diferencia puede estribar en que el código fuente en el Software libre está disponible para todos y cualquiera puede aportar una solución y por lo general al poco tiempo de detectarse una vulnerabilidad (a veces en cuestión de horas) se puede disponer de una solución para la misma. Además, al tener acceso al código fuente se puede detectar si alguien introdujo código malicioso a una determinada aplicación.

Pero de todos es sabido, que los usuarios de Software de código abierto, como por ejemplo los que de manera habitual trabajan con equipos comandados por sistemas Linux, por regla general se sienten orgullosos de la seguridad que estos programas aportan con respecto a los sistemas cerrados propios de otras firmas, dígase Microsoft Windows o Mac de Apple.

¿Es Seguro el Software Libre? En primer lugar definiremos el concepto de "seguridad" como salvaguarda de las propiedades básicas de la información. Entre las características que debe cumplir para ser seguro, encontramos la integridad, es decir, que sólo los usuarios autorizados pueden crear y modificar los componentes del sistema, la confidencialidad, sólo estos usuarios pueden acceder a esos componentes, la disponibilidad, que todos los componentes estén a disposición de los usuarios siempre que lo deseen y el "no repudio", o lo que es lo mismo, la aceptación de un protocolo de comunicación entre el servidor y un cliente, por ejemplo, mediante certificados digitales.

Entre las diferencias de seguridad entre un Software Libre y el Software Propietario, podemos destacar:

- Seguridad en el Software Propietario: En el caso de tener "agujeros de seguridad", puede que no nos demos cuenta y que no podamos repararlos. Existe una dependencia del fabricante, retrasándose así cualquier reparación y la falsa creencia de que es más seguro por ser oscuro (la seguridad por oscuridad determina los fallos de seguridad no parcheados en cada producto).

- Seguridad en el Software Libre: Por su carácter público y su crecimiento progresivo, se van añadiendo funciones y se nos permite detectar más fácilmente los agujeros de seguridad para poder corregirlos. Los problemas tardan mucho menos en ser resueltos por el apoyo que tiene de los Hackers y una gran comunidad de desarrolladores y al ser un Software de código libre, cualquier empresa puede aportar soporte técnico.

Sin embargo esta es una pregunta sobre la que los expertos al día de hoy, tras muchos años de discusiones, siguen sin ponerse de acuerdo. ¿Es más seguro el Software de código abierto que los programas cerrados, o viceversa? Lo cierto es que, en términos generales, ambos bandos tienen sus razones con las que defender sus argumentos. Por un lado, los usuarios de las aplicaciones y sistemas de código abierto, defienden que, al estar el código fuente disponible a los ojos de todo el mundo, es mucho más fácil localizar posibles agujeros de seguridad y vulnerabilidades que pongan en peligro los datos de los usuarios.

Por otro lado, aquellos que consideran que los sistemas cerrados son más seguros en este sentido, afirman que al tener acceso tan solo los expertos al código fuente de sus aplicaciones, es más complicado que se produzcan

filtraciones o inserciones de Software malicioso en este tipo de sistemas. Hay que tener en cuenta que, por ejemplo, Google premia a las personas que descubren fallos de seguridad en su Software como Chrome, aunque no es el único gigante de la tecnología en utilizar estas tácticas.

De hecho muchas empresas están gastando miles de millones de dólares y/o euros en hacer que sus propuestas sean lo más seguras posible, argumentando que la seguridad de sus proyectos es una de sus prioridades, todo con el fin de intentar frenar que los atacantes vulneren sus sistemas. Por otro lado, otros aseguran que cuando el código fuente es público, más ojos están disponibles para detectar posibles vulnerabilidades o errores en dicho código, por lo que siempre será más rápido y sencillo poner soluciones con el fin de ganar en seguridad.

Sea como sea, en cualquiera de los dos casos, lo que ha quedado más que demostrado es que la seguridad no está garantizada en ningún momento, ya sean propuestas de código abierto, o no. Pero también es cierto que lo que se procura es que los riesgos de ser atacados se reduzcan en medida de lo posible. Los sistemas Linux son considerados desde hace mucho tiempo como un sistema operativo seguro, en buena parte debido a las ventajas que ofrece su diseño. Dado que su código está abierto, son muchas las personas que incorporan mejoras de las que el resto de usuarios de Linux se benefician, a diferencia de las propuestas de Windows o MacOS, donde estas correcciones generalmente se limitan a las que detectan Microsoft y Apple.

No obstante, en nuestra defensa del Software libre, diremos que su código abierto permite que los errores sean encontrados y solucionados con mayor rapidez, por lo que determinamos que es el Software más recomendable.

En general, puede afirmarse que el Software libre es más seguro, ya que debido a su carácter abierto y distribuido, un gran número de programadores y personas expertas pueden estar atentas al código fuente -especialmente en los grandes proyectos-, lo cual permite hacer auditorías con objeto de detectar errores y puertas traseras (Backdoor, en inglés) que pongan en riesgo nuestros datos.

Así, los grandes programas y proyectos de Software libre, con una extensa comunidad de desarrollo y usuarios que lo respalden, presentan niveles muy altos de seguridad, un alto grado de protección y una rápida respuesta a posibles vulnerabilidades.

10.4 Tipos de Licencias

Tanto la Open Source Initiative como la Free Software Foundation mantienen en sus páginas Web (véase [**?**], [**?**], y [**?**]) listados oficiales de las licencias de Software libre que aprueban.

Una licencia es aquella autorización formal con carácter contractual que un autor de un Software da a un interesado para ejercer "actos de explotación legales". Pueden existir tantas licencias como acuerdos concretos se den entre el autor y el licenciatario. Desde el punto de vista del Software libre, existen distintas variantes del concepto o grupos de licencias:

Licencias GPL Una de las más utilizadas es la Licencia Pública General de GNU (GNU GPL). El autor conserva los derechos de autoría (Copyright), y permite la redistribución y modificación bajo términos diseñados para asegurarse de que todas las versiones modificadas del Software permanecen bajo los términos más restrictivos de la propia GNU GPL. Esto hace que sea imposible crear un producto con partes no licenciadas GPL, el conjunto tiene que ser GPL.

En la práctica, esto hace que las licencias de Software libre se dividan en dos grandes grupos, aquellas que pueden ser mezcladas con código licenciado bajo GNU GPL (y que inevitablemente desaparecerán en el proceso, al ser el código resultante licenciado bajo GNU GPL) y las que no lo permiten al incluir mayores u otros requisitos que no contemplan ni admiten la GNU GPL y que por lo tanto no pueden ser enlazadas ni mezcladas con código gobernado por la licencia GNU GPL.

GPL Versión 1 la versión 1 de GNU GPL, fue presentada el 25 de febrero de 1989, impidió lo que eran las dos principales formas con las que los distribuidores de Software restringían las libertades definidas por el Software libre. El primer problema fue que los distribuidores publicaban únicamente los archivos binarios, funcionales y ejecutables, pero no entendibles o modificables por humanos. Para prevenir esto, la GPLv1 estableció que cualquier proveedor de Software libre además de distribuir el archivo binario debía liberar a su vez código fuente entendible y que pudiera ser modificado por el ser humano bajo la misma licencia (secciones 3a y 3b de la licencia).

El segundo problema era que los distribuidores podían añadir restricciones adicionales, añadiendo restricciones a la licencia o mediante la combinación del Software con otro que tuviera otras restricciones en su distribución. Si

esto se hacía, entonces la unión de los dos conjuntos de restricciones sería aplicada al trabajo combinado, entonces podrían añadirse restricciones inaceptables. Para prevenir esto, GPLv1 obligaba a que las versiones modificadas en su conjunto, tuvieran que ser distribuidas bajo los términos GPLv1 (secciones 2b y 4 de la licencia). Por lo tanto, el Software distribuido bajo GPLv1 puede ser combinado con Software bajo términos más permisivos y no con Software con licencias más restrictivas, lo que entraría en conflicto con el requisito de que todo Software tiene que ser distribuido bajo los términos de la GPLv1.

GPL Versión 2 según Richard Stallman, el mayor cambio en GPLv2 fue la cláusula "Liberty or Death" («libertad o muerte»). Esta sección dice que si alguien impone restricciones que le prohíben distribuir código GPL de tal forma que influya en las libertades de los usuarios (por ejemplo, si una ley impone que esa persona únicamente pueda distribuir el Software en binario), esa persona no puede distribuir Software GPL. La esperanza es que esto hará que sea menos tentador para las empresas el recurrir a las amenazas de patentes para exigir una remuneración de los desarrolladores de Software libre.

En 1991 se hizo evidente que una licencia menos restrictiva sería estratégicamente útil para la biblioteca C y para las bibliotecas de Software que esencialmente hacían el trabajo que llevaban a cabo otras bibliotecas comerciales ya existentes. Cuando la versión 2 de GPL fue liberada en junio de 1991, una segunda licencia Library General Public License fue introducida al mismo tiempo y numerada con la versión 2 para denotar que ambas son complementarias. Los números de versiones divergieron en 1999 cuando la versión 2.1 de LGPL fue liberada, esta fue renombrada como GNU Lesser General Public License para reflejar su lugar en esta filosofía.

GPL Versión 3 A finales de 2005, la Free Software Foundation (FSF) anunció estar trabajando en la versión 3 de la GPL (GPLv3). El 16 de enero de 2006, el primer borrador de GPLv3 fue publicado y se inició la consulta pública. La consulta pública se planeó originalmente para durar de nueve a quince meses, pero finalmente se extendió a dieciocho meses, durante los cuales se publicaron cuatro borradores. La GPLv3 oficial fue liberada por la FSF el 29 de junio de 2007.

Según Stallman los cambios más importantes se produjeron en el campo

de las patentes de Software, la compatibilidad de licencias de Software libre, la definición de código fuente y restricciones a las modificaciones de Hardware. Otros cambios están relacionados con la internacionalización, cómo son manejadas las violaciones de licencias y cómo los permisos adicionales pueden ser concedidos por el titular de los derechos de autor. También añade disposiciones para quitar al DRM su valor legal, por lo que es posible romper el DRM (Digital Rights Management) en el Software de GPL sin romper leyes como la DMCA (Digital Millennium Copyright Act).

GPLv2 vs GPL v3 GPLv3 contiene la intención básica de GPLv2 y es una licencia de código abierto con un Copyleft estricto. Sin embargo, el idioma del texto de la licencia fue fuertemente modificado y es mucho más completo en respuesta a los cambios técnicos, legales y al intercambio internacional de licencias.

La nueva versión de la licencia contiene una serie de cláusulas que abordan preguntas que no fueron o fueron cubiertas de manera insuficiente en la versión 2 de la GPL. Las nuevas regulaciones más importantes son las siguientes:

- GPLv3 contiene normas de compatibilidad que hacen que sea más fácil combinar el código GPL con el código que se publicó bajo diferentes licencias. Esto se refiere en particular al código bajo la licencia de Apache v. 2.0.

- Se insertaron normas sobre gestión de derechos digitales para evitar que el Software GPL se modifique a voluntad, ya que los usuarios recurrieron a las disposiciones legales para protegerse mediante medidas técnicas de protección (como la DMCA o la directiva sobre derechos de autor).

- La licencia GPLv3 contiene una licencia de patente explícita, según la cual las personas que licencian un programa bajo licencia GPL otorgan derechos de autor y patentes, en la medida en que esto sea necesario para utilizar el código que ellos otorgan. Por lo tanto, no se concede una licencia de patente completa. Además, la nueva cláusula de patente intenta proteger al usuario de las consecuencias de los acuerdos entre los titulares de patentes y los licenciatarios de la licencia pública general que solo benefician a algunos de los licenciatarios (correspondientes al

acuerdo Microsoft / Novell). Los licenciatarios deben garantizar que todos los usuarios disfrutan de tales ventajas (licencia de patente o liberación de reclamos) o que nadie puede beneficiarse de ellos.

- A diferencia de la GPLv2, la GPLv3 establece claramente que no es necesario divulgar el código fuente en un uso ASP (Application Service Provider) de los programas GPL, siempre que no se envíe una copia del Software al cliente. Si el efecto Copyleft debe extenderse al uso de ASP, debe aplicarse la Licencia pública general de Affero, versión 3 (AGPL) que solo difiere de la GPLv3 en esta consideración.

Licencias Estilo BSD Llamadas así porque se utilizan en gran cantidad de Software distribuido junto a los sistemas operativos BSD. El autor, bajo tales licencias, mantiene la protección de Copyright únicamente para la renuncia de garantía y para requerir la adecuada atribución de la autoría en trabajos derivados, pero permite la libre redistribución y modificación, incluso si dichos trabajos tienen propietario. Son muy permisivas, tanto que son fácilmente absorbidas al ser mezcladas con la licencia GNU GPL con quienes son compatibles. Puede argumentarse que esta licencia asegura "verdadero" Software libre, en el sentido que el usuario tiene libertad ilimitada con respecto al Software y que puede decidir incluso redistribuirlo como no libre.

Licencia Copyleft Hay que hacer constar que el titular de los derechos de autor (Copyright) de un Software bajo licencia Copyleft puede también realizar una versión modificada bajo su Copyright original y venderla bajo cualquier licencia que desee, además de distribuir la versión original como Software libre. Esta técnica ha sido usada como un modelo de negocio por una serie de empresas que realizan Software libre (por ejemplo MySQL); esta práctica no restringe ninguno de los derechos otorgados a los usuarios de la versión Copyleft.

Licencia estilo MIT Las licencias MIT son de las más permisivas, casi se consideran Software de dominio público. Lo único que requieren es incluir la licencia MIT para indicar que el Software incluye código con licencia MIT.

Licencia Apache License La licencia Apache trata de preservar los derechos de autor, incluir la licencia en el Software distribuido y una lista de

los cambios realizados. En modificaciones extensivas del Software original permite licenciar el Software bajo otra licencia sin incluir esas modificaciones en el código fuente.

Licencia Mozilla Public License MPL Esta licencia requiere que los archivos al ser distribuidos conserven la misma licencia original pero pueden ser usados junto con archivos con otra licencia, al contrario de la licencia GPL que requiere que todo el código usado junto con código GPL sea licenciado como código GPL. También en caso de hacer modificaciones extensivas permite distribuirlas bajo diferentes términos y sin incluir el código fuente en las modificaciones.

Licencia Código de Dominio Público Es un código que no está sujeto a derechos de autor que puede utilizarse sin restricciones.

Licencia Creative Commons Las licencias de Creative Commons son más utilizadas para cualquier creación digital que para el Software, entendiendo como creación digital desde fotos, artículos en blogs, música, vídeos, este trabajo, etc. Hay varios tipos de licencias de Creative Commons diferenciando entre permitir modificaciones a la obra original, solicitando crédito de la creación o permitiendo un uso comercial de la obra.

Licencias de Código Abierto Las licencias de código abierto son un intermedio entre las licencias privativas y las licencias de Software libre. Las licencias de código abierto permiten el acceso al código fuente pero no todas se consideran licencias de Software libre al no otorgar otros derechos que se requieren para considerar un Software como Software libre como el derecho al uso o con cualquier propósito, modificación y distribución.

Dado el éxito del Software libre como modelo de desarrollo de Software algunas empresas cuyo Software era privativo pueden decidir hacerlo de código abierto con la intención de suplir algunas carencias de Software privativo pero sin perder ciertos derechos que son la fuente de sus ingresos como la venta de licencias.

Las expresiones «Software libre» y «código abierto» se refieren casi al mismo conjunto de programas. No obstante, dicen cosas muy diferentes acerca de dichos programas, basándose en valores diferentes. El movimiento del Software libre defiende la libertad de los usuarios de ordenadores, en un

movimiento en pro de la libertad y la justicia. Por contra, la idea del código abierto valora principalmente las ventajas prácticas y no defiende principios. Esta es la razón por la que gran parte de la comunidad de Software libre está en desacuerdo con el movimiento del código abierto y nosotros no empleamos esta expresión en este texto.

Licencia Microsoft Public License La Microsoft Public License es una licencia de código abierto que permite la distribución del Software bajo la misma licencia y la modificación para un uso privado. Tiene restricciones en cuanto a las marcas registradas.

En caso de distribuir el Software de forma compilada o en forma de objeto binario no se exige proporcionar los derechos de acceso al código fuente del Software compilado o en forma de objeto binario. En este caso esta licencia no otorga más derechos de los que se reciben, pero si permite otorgar menos derechos al distribuir el Software (compilado o en forma de objeto binario).

Modelo de Desarrollo de Software Bazar y Catedral El tipo de licencia no determina qué Software es mejor o peor, si el privativo o el Software libre, la diferencia entre las licencias está en sus características éticas y legales. Aunque el modelo de desarrollo con una licencia de código abierto a la larga suele tener un mejor desarrollo y éxito que el Software privativo, más aún con un medio como internet que permite colaborar a cualquier persona independiente de donde esté ubicada en el mundo.

Comparación con el Software de Código Abierto Aunque en la práctica el Software de código abierto y el Software libre comparten muchas de sus licencias, la Free Software Foundation opina que el movimiento del Software de código abierto es filosóficamente diferente del movimiento del Software libre. Los defensores del término «código abierto (Open Source)», afirman que éste evita la ambigüedad del término en ese idioma que es «Free» en «Free Software».

Mucha gente reconoce el beneficio cualitativo del proceso de desarrollo de Software cuando los desarrolladores pueden usar, modificar y redistribuir el código fuente de un programa. El movimiento del Software libre hace especial énfasis en los aspectos morales o éticos del Software, viendo la excelencia técnica como un producto secundario de su estándar ético. El movimiento de código abierto ve la excelencia técnica como el objetivo prioritario, siendo

el compartir el código fuente un medio para dicho fin. Por dicho motivo, la FSF se distancia tanto del movimiento de código abierto como del término «código abierto (Open Source)».

Puesto que la «iniciativa de Software libre Open Source Initiative (OSI)» sólo aprueba las licencias que se ajustan a la «definición de código abierto (Open Source Definition)», la mayoría de la gente lo interpreta como un esquema de distribución, e intercambia libremente "código abierto" con "Software libre". Aún cuando existen importantes diferencias filosóficas entre ambos términos, especialmente en términos de las motivaciones para el desarrollo y el uso de tal Software, raramente suelen tener impacto en el proceso de colaboración.

Aunque el término "código abierto" elimina la ambigüedad de libertad frente a precio (en el caso del inglés), introduce una nueva: entre los programas que se ajustan a la definición de código abierto, que dan a los usuarios la libertad de mejorarlos y los programas que simplemente tienen el código fuente disponible, posiblemente con fuertes restricciones sobre el uso de dicho código fuente. Mucha gente cree que cualquier Software que tenga el código fuente disponible es de código abierto, puesto que lo pueden manipular, sin embargo, mucho de este Software no da a sus usuarios la libertad de distribuir sus modificaciones, restringe el uso comercial, o en general restringe los derechos de los usuarios.

10.4.1 Licencias Creative Commons

Las Licencias[94] Creative Commons (CC) de forma general no tienen una definición oficial, sin embargo, entre las muchas definiciones aceptadas están la de la UNESCO, la cual expresa la siguiente descripción:

Las Licencias Creative Commons (CC) son modelos de contratos que sirven para otorgar públicamente el derecho de utilizar una publicación protegida por los derechos de autor. Entre menos restricciones implique una licencia, mayores serán las posibilidades de utilizar y distribuir un contenido. Las Licencias CC permiten a cualquier usuario descargar, copiar, distribuir, traducir, reutilizar, adaptar y desarrollar su contenido sin costo alguno.

Sin embargo, en la Web oficial de la Organización Creative Commons se nos dice sobre las mismas lo siguiente:

[94] Las licencias de Creative Commons son más utilizadas para cualquier creación digital que para el Software, entendiendo como creación digital desde fotos, artículos en blogs, música, vídeos, este trabajo, etc.

Las Licencias Creative Commons (CC) brindan a todos, desde creadores individuales hasta grandes instituciones, una forma estandarizada de otorgar permiso al público para usar su trabajo creativo bajo la ley de derechos de autor. Desde la perspectiva del reutilizador, la presencia de una licencia Creative Commons sobre una obra protegida por derechos de autor responde a la pregunta: ¿Qué puedo hacer con esta obra?.

Las «Licencias Creative Commons» que hoy en día pertenecen a la organización mundial Creative Commons[95], y buscan regularizar y mantener, de forma equilibrada y satisfactoria, todo lo relacionado con el derecho de utilizar una publicación protegida por los derechos de autor a nivel mundial, han logrado un buen trabajo, sin duda alguna. Y seguramente en el tiempo, se irán adaptando a las nuevas realidades sociales y tecnológicas para poder seguir manteniendo de forma armónica las posibilidades de utilizar y distribuir cualquier contenido libre y abierto sobre la Internet, y más allá.

¿Cuáles son y cómo funcionan o para qué se usan? Las 7 distintas Licencias Creative Commons son las siguientes:

CC BY Esta licencia permite a los reutilizadores distribuir, remezclar, adaptar y desarrollar el material en cualquier medio o formato, siempre que se otorgue la atribución al creador. La licencia permite el uso comercial.

CC BY-SA Esta licencia permite a los reutilizadores distribuir, remezclar, adaptar y desarrollar el material en cualquier medio o formato, siempre que se otorgue la atribución al creador. La licencia permite el uso comercial. Si remezcla, adapta o construye sobre el material, debe licenciar el material modificado bajo términos idénticos.

CC BY-NC Esta licencia permite a los reutilizadores distribuir, remezclar, adaptar y desarrollar el material en cualquier medio o formato, únicamente con fines no comerciales y siempre que se otorgue la atribución al creador.

[95] Una organización mundial sin ánimo de lucro que permite compartir y reutilizar la creatividad y el conocimiento mediante el suministro de herramientas legales gratuitas. Y cuyas herramientas legales (licencias) ayudan a quienes quieren fomentar la reutilización de sus obras ofreciéndolas para su uso bajo términos generosos y estandarizados; a quienes quieren hacer usos creativos de las obras; y a quienes quieren beneficiarse de esta simbiosis.

CC BY-NC-SA Esta licencia permite a los reutilizadores distribuir, remezclar, adaptar y desarrollar el material en cualquier medio o formato, únicamente con fines no comerciales y siempre que se otorgue la atribución al creador. Si remezcla, adapta o construye sobre el material, debe licenciar el material modificado bajo términos idénticos.

CC BY-ND Esta licencia permite a los reutilizadores copiar y distribuir el material en cualquier medio o formato únicamente en forma no adaptada, y siempre y cuando se otorgue la atribución al creador. La licencia permite el uso comercial.

CC BY-NC-ND Esta licencia permite a los reutilizadores copiar y distribuir el material en cualquier medio o formato únicamente en forma no adaptada, únicamente con fines no comerciales y siempre que se otorgue la atribución al creador.

CC0 (CC Cero) Esta licencia es una herramienta de dedicación pública que permite a los creadores renunciar a sus derechos de autor y poner sus obras en el dominio público mundial. CC0 permite a los reutilizadores distribuir, remezclar, adaptar y desarrollar el material en cualquier medio o formato, sin condiciones.

10.4.2 Nuevas Licencias para Responder a Nuevas Necesidades

El mundo de la tecnología avanza mucho más rápido que las leyes y estas tienen que esforzarse para alcanzarlo. En el caso del Software libre y de código abierto, tanto la Free Software Foundation como la Open Source Initiative (los organismos encargados de regular las diferentes licencias) enfrentan periódicamente el problema de cómo mantener sus principios y al mismo tiempo evitar que alguien se aproveche indebidamente.

En el último tiempo, la Open Source Initiative le dio el sello de aprobación a otras nuevas licencias para propósitos específicos.

Nuevas Licencias de Código Abierto

- Cryptographic Autonomy License version 1.0 (CAL-1.0)

Fue creada en el 2019 por el equipo del proyecto de código abierto Holochain, esta licencia fue desarrollada para ser utilizada con aplicaciones criptográficas distribuidas. El inconveniente con las licencias tradicionales es que no obligaba a compartir los datos. Esto podría perjudicar el funcionamiento de toda la red. Por eso la CAL también incluye la obligación de proporcionar a terceros los permisos y materiales necesarios para utilizar y modificar el Software de forma independiente sin que ese tercero tenga una pérdida de datos o capacidad.

- Open Hardware Licence (OHL)

De la mano de la Organización Europea para la Investigación Nuclear (CERN) llegó esta licencia con tres variantes enfocadas en la posibilidad de compartir libremente tanto Hardware como Software.

Hay que hacer una aclaración. La OSI fue creada en principio pensando en el Software por lo que no tiene mecanismos para la aprobación de licencias de Hardware. Pero, como la propuesta del CERN se refiere a ambos rubros, esto posibilitó la aprobación:

- CERN-OHL-S es una licencia fuertemente recíproca: El que utilice un diseño bajo esta licencia deberá poner a disposición las fuentes de sus modificaciones y agregados bajo la misma licencia.
- CERN-OHL-W es una licencia débilmente recíproca: Sólo obliga a distribuir las fuentes de la parte del diseño que fue puesta originalmente bajo ella. No así los agregados y modificaciones.
- CERN-OHL-P es una licencia permisiva: Permite a la gente tomar un proyecto, relicenciarlo y utilizarlo sin ninguna obligación de distribuir las fuentes.

Hay que decir que la gente del CERN parece haber encontrado la solución a un problema que viene afectando a algunos proyectos de código abierto. Una gran empresa utiliza ese proyecto para comercializar servicios y no solo hace ningún aporte al proyecto original (ya sea con código o dando apoyo financiero) si no que también compite en el mismo mercado.

Post Open Zero-Cost en el año 2024 se desarrolla la licencia «Post Open Zero-Cost» con la cual busca abordar los desafíos surgidos en la interacción entre desarrolladores de código abierto y empresas comerciales, especialmente en lo que respecta a la compensación justa por el uso comercial del código.

La característica distintiva de la licencia «Post-Open» en comparación con las licencias abiertas existentes, como la GPL, es la introducción de un componente contractual que puede ser rescindido en caso de violación de los términos. Esta licencia ofrece dos tipos de acuerdos contractuales: gratuitos y de pago. El acuerdo de pago permite negociar derechos adicionales para la distribución comercial de productos o modificaciones sin requerir su divulgación pública.

Las situaciones que podrían llevar a la terminación del acuerdo contractual incluyen: violación de los términos de la licencia; reclamaciones por infracción de patentes; imposición de condiciones adicionales (como sanciones en contratos con clientes por divulgación de información sensible); cambios sujetos a leyes de control de exportaciones; ocultamiento de información sobre vulnerabilidades; y uso del código para entrenamiento de modelos de aprendizaje automático bajo términos no permitidos. Las relaciones contractuales no se rescinden de inmediato, sino que se notifica la infracción y se otorgan 60 días para corregirla antes de la rescisión efectiva del acuerdo.

Uno de los problemas que la nueva licencia busca abordar está relacionado con las limitaciones de la GPL, la cual se centra en otorgar derechos sin la capacidad de revocarlos, lo que permite a las empresas encontrar formas de eludir sus requisitos, especialmente en lo que respecta al acceso al código fuente. Estas lagunas son utilizadas para restringir la disponibilidad del código subyacente en productos comerciales mediante la imposición de términos contractuales adicionales con los usuarios finales.

Un claro ejemplo es el de RHEL, la cual los clientes firman un acuerdo con Red Hat que limita la redistribución del código fuente al imponer condiciones sobre la coincidencia de las copias instaladas y compradas de RHEL. Esto coloca a los usuarios en la disyuntiva entre su libertad para disponer del software y mantener su estatus de cliente de Red Hat. Aunque la GPL permite la distribución de parches que solucionan vulnerabilidades en el código de RHEL, esto podría interpretarse como una violación del acuerdo con Red Hat y podría resultar en la terminación de los servicios por parte de la empresa.

10.5 Implicaciones Económico-Políticas del Software Libre

Una vez que un producto de Software libre ha empezado a circular, rápidamente está disponible a un costo muy bajo. Al mismo tiempo, su utilidad no decrece. El Software, en general, podría ser considerado un bien de uso inagotable, tomando en cuenta que su costo marginal es pequeñísimo y que no es un bien sujeto a rivalidad -la posesión del bien por un agente económico no impide que otro lo posea-.

10.5.1 Software Libre y la Piratería

Puesto que el Software libre permite el libre uso, modificación y redistribución, a menudo encuentra un hogar entre usuarios para los cuales el coste del Software no libre es a veces prohibitivo, o como alternativa a la piratería. También es sencillo modificarlo localmente, lo que permite que sean posibles los esfuerzos de traducción a idiomas que no son necesariamente rentables comercialmente, además:

- Porque así no se condiciona a los usuarios a usar siempre lo mismo.
- Porque así no se fomenta la piratería en los usuarios al no pagar licencias.
- Porque así no se obliga a usar una solución concreta y se ofrece libertad de elección a los usuarios.
- Porque es mucho más seguro ya que el Software libre es público y se puede ver qué hace exactamente sin recelos.

La mayoría del Software libre se produce por equipos internacionales que cooperan a través de la libre asociación. Los equipos están típicamente compuestos por individuos con una amplia variedad de motivaciones y pueden provenir tanto del sector privado, del sector voluntario o del sector público. En los últimos años se ha visto un incremento notable de grandes corporativos (como IBM, Microsoft, Intel, Google, Samsung, Red Hat, etc.) que han dedicado una creciente cantidad de recursos humanos y computacionales para desarrollar Software libre, ya que esto apoya a sus propios negocios.

10.5.2 ¿Cuánto Cuesta el Software Libre?

En esta sección intentaremos dar una idea de cuál es el costo del desarrollo del Software Libre, por supuesto que no se tratará más que de una conjetura aproximada basada en las cifras proporcionadas por desarrolladores de Software comercial (al año 2020).

Gratis no Significa Gratuito Supongamos que todos los recursos humanos participantes en el desarrollo de un proyecto de Software libre lo hagan de forma voluntaria. De todas formas tenemos lo que los contables llaman «Costo de oportunidad» esto es, los ingresos que podrían haber generado esas personas si hubieran dedicado el tiempo y los conocimientos invertidos en el proyecto a uno en el que les pagaran. Así, el calcular el costo promedio por hora que cobra un programador, por la cantidad de horas invertidas al proyecto, nos da un razonable costo mínimo. Lo mismo puede hacerse con los voluntarios dedicados a la difusión en las redes. El costo de una campaña de marketing digital puede estimarse fácilmente.

Muchos proyectos de código abierto como una distribución Linux, son construidos a partir de la integración de otros proyectos, por los que sus costos de desarrollo también deberían sumarse.

Por otra parte, necesitamos recursos físicos. Aún cuando los voluntarios trabajen desde su casa, siguen teniendo que comprar y mantener sus equipos, además de pagar la electricidad que los hace funcionar.

Bases para el Cálculo Hay muchos factores que determinan el costo de desarrollar una pieza de Software. En un extremo tenemos una aplicación simple que requiere muy poca interacción del usuario o procesamiento del lado del servidor. Tal es el caso de un cliente de escritorio para redes sociales. Por el otro sistemas operativos que deben operar en múltiples plataformas realizando múltiples tareas (por ejemplo Debían que aspira a ser el sistema operativo universal). Sin embargo, el costo de una aplicación simple puede elevarse debido a que tiene múltiples pantallas diferentes. Por ejemplo un juego desarrollado con HTML5 y Javascript.

Los dos aspectos claves son la cantidad de horas de trabajo necesarias y las tecnologías involucradas. Para una aplicación de escritorio como un procesador de textos con las prestaciones habituales, optimizado para un determinado escritorio Linux, se estima que se tendría que contar con al menos el equivalente a 42,000 euros en trabajo voluntario. Un gestor de contenidos

para comercio electrónico con seguimiento de pedidos e integración con las principales plataformas de pago implicaría desembolsar unos 210,000 euros o su equivalente en trabajo voluntario.

Tomando en cuenta que este cálculo incluye lo que costó el desarrollo de las bibliotecas y otros proyectos libres y de código abierto incluidos, pero no los gastos que efectivamente deben desembolsarse en efectivo como la compra de equipos, Software de seguridad y desarrollo y el pago de electricidad e internet.

Por otro lado, el proceso de medición de costes del Software es un factor realmente importante en el análisis de un proyecto. Hay distintos métodos de estimación de costes de desarrollo de Software (también conocido como métrica del Software). La gran mayoría de estos métodos se basan en la medición del número de Líneas de Código que contiene el desarrollo (se excluyen comentarios y líneas en blanco de los fuentes).

Desarrollo de Fedora 9 La Linux Foundation ha calculado que costaría desarrollar el código de la distribución Fedora 9 que fue puesta a disposición del público el 13 de mayo de 2008, en el informe citado "Estimating the Total Development Cost of a Linux Distribution" se calcula que Fedora 9 tiene un valor de 10.8 mil millones de dólares y que el coste únicamente del Kernel (2.6.25 con 8,396,250 líneas de código) tendría un valor de 1.4 mil millones de dólares.

Esta distribución tiene unas 205 millones de líneas de código, el proyecto debería ser desarrollado por 1000 - 5000 desarrolladores (el trabajo invertido por una única persona desarrollándolo se alargaría durante unos 60.000 años) y esa estimación no va muy desencaminada ya que en los 2 últimos años del desarrollo de esa versión contribuyeron unos 3,200 desarrolladores aunque el número de trabajadores en la historia de la distribución es mucho mayor.

¿Qué pasa con GNU/Linux? en el año 2015 (las estadísticas más actuales que conseguimos) la Linux Foundation analizó el costo de desarrollo del núcleo. Combinando el aporte de los recursos humanos (voluntarios y de pago) y los desembolsos necesarios, la cuenta sumó 476,767,860,000.13 euros.

Todos sabemos que el hecho de tener desarrolladores asalariados no garantiza necesariamente Software de calidad. Pero, tener desarrolladores que pueden dedicar toda su atención a un proyecto en lugar de hacerlo en sus horas libres si lo hace. Lamentablemente, por el momento el único modo de

lograr eso es obtener el apoyo de corporaciones (Intel, Google, IBM, AMD, Sun Microsystems, Dell, Lenovo, Asus, HP, SGI, Oracle, RedHat, etc.) que solo lo hacen con los proyectos que son de su interés como el Kernel de Linux, hay que notar que para el Kernel de Linux un porcentaje importante (más del 10 %) lo hacen programadores independientes.

Costes Recordemos que la segunda de las cuatro libertades de un programa para ser Software libre es:

- Libre redistribución

y esta puede ser a través de un pago o sin costo. Es por ello que existen distintas empresas, organizaciones y usuarios que pueden apoyar a los usuarios finales en el desarrollo y soporte de algún programa de Software libre o una distribución personalizada de Linux por un costo determinado.

Mucha gente, en especial ejecutivos de empresas, se acercan a Linux bajo la promesa de que es una solución de bajo costo -muchos piensan que incluso es gratis-. Pero la realidad es que detrás de Linux y los programas de Software libre (y aquí la traducción correcta de la palabra inglesa, Free es libre, no gratis) pueden llevar una serie de costos ocultos que deben ser considerados al momento de decidir si se implementa una solución propietaria o una libre.

Los costos ocultos aparecen cuando se intenta instalar y capacitar en el uso de algún Software y se necesita la ayuda de un informático, al que se tiene que pagar, o alguna empresa quiere personalizar la interfaz de un programa y necesita la ayuda de un programador, que también tienen un coste, por lo que finalmente el comentario suele ser "el Software libre no es barato".

El primer punto a considerar al evaluar ambas alternativas es el costo de la licencia. Los productos de Software libre no suelen tener un costo de licencia asociado, que sí existe en los programas propietarios. De hecho es allí en donde los fabricantes de Software recargan sus costos de investigación y desarrollo, de producción e incluso sus ganancias. En este primer punto el ganador claro es el Software libre y es lo que los adeptos de este esquema publicitan: "su compañía puede ahorrar miles de dólares al año usando Software libre".

El segundo punto a considerar es el costo de instalación, configuración y capacitación. Dependiendo de su complejidad, algunos productos comerciales no contemplan costos extra por este concepto y otros -como Windows, por ejemplo- son tan populares que se puede encontrar numerosas opciones

de instalación -a través de empresas o profesionales- donde escoger en el mercado. A veces en el Software libre la configuración puede implicar recompilar el producto con algunas opciones particulares, algo que sólo pueden realizar técnicos con un nivel adecuado de conocimientos y que puede que no sean tan fáciles de encontrar. Aquí por lo general la ventaja va hacia el Software comercial.

Una vez instalado el Software, toca realizar actualizaciones de mantenimiento. Si bien es cierto lo que algunos de los fanáticos del Software libre dicen, que nadie lo obliga a mejorar el Software con que cuenta, la realidad -especialmente en lo que a seguridad se refiere- obliga a las empresas a mantener su Software actualizado. Aún así, los costos de actualizar Software libre suelen ser significativamente más bajos que los de productos comerciales y suelen ser menos exigentes con el Hardware necesario para ejecutarlos. La mayoría de las veces la ventaja es para el Software libre, pero hay que evaluar ya que varía dependiendo de cada caso.

Por último hay algunas casas que desarrollan productos de Software libre -dan el código y permiten que cualquiera lo modifique o reutilice- pero fijan contratos con cargos mensuales o anuales para mantenimiento del mismo, algo que se parece mucho a un cobro por licencia, por lo que hay que estar seguro de conocer todas las condiciones cuando una empresa nos ofrece una solución propia y la califica como Software libre.

Además de estas consideraciones hay una que debe sumarse eventualmente a esta evaluación: el costo de migrar a otra solución. En Software libre suelen usarse estándares abiertos para almacenar los datos, lo que facilita las migraciones. En cambio muchas soluciones propietarias suelen tener formatos propietarios que pueden dejar "amarrados" los datos de la empresa a una aplicación específica.

Sólo después de evaluar estos aspectos del Software, que pueden tener implicaciones importantes en el presupuesto, es que un CIO (Chief Information Officer) puede decir si una solución de Software libre le conviene más a una empresa o no, algo que va más allá de que la aplicación sea gratis o no.

10.5.3 La Nube y el Código Abierto

Desde hace años se han creado nuevos desafíos para el código abierto que plantea la nube, un término que para el usuario promedio puede significar cosas diferentes, pero que para la empresa se resume en servicios. Y es que los beneficios económicos que genera el mero Software de código abierto no

son comparables a los que se obtienen cuando se ofrece ese mismo Software a través de servicios, más allá -pero incluyendo- del soporte.

Este hecho diferencial lleva tiempo provocando fricciones entre desarrolladores y proveedores y hay quien adelantó incluso el fin del modelo de desarrollo del código abierto tal y como lo conocemos. ¿Quién tiene la razón?, ¿es para tanto la situación?.

En sus exposiciones, representantes de compañías y proyectos de código abierto muy populares en el ámbito empresarial, explican el supuesto perjuicio que les ocasiona el uso que los grandes proveedores de servicios en la nube hacen del Software que ellos desarrollan y cómo algunos han considerado y aplicado un enfoque más cerrado para sus productos con el fin de evitar lo que denominan como expolio. Hay declaraciones que merecen ser rescatadas para dotar de contexto a la discusión:

- El papel que juega el código abierto en la creación de oportunidades comerciales ha cambiado, durante muchos años les permitimos que las empresas de servicios tomaran lo que se ha desarrollado y ganasen dinero con ello.

- Empresas como Amazon Web Services, Azure de Microsoft, etc. Han ganado cientos de millones de dólares ofreciendo a sus clientes servicios basados en Software libre sin contribuir tanto a la comunidad de código abierto que construye y mantiene ese proyecto. Es imposible saber exactamente de cuánto dinero estamos hablando, pero es cierto que los proveedores de la nube se benefician del trabajo de los desarrolladores de código abierto que no emplean.

- Hay un mito ampliamente instalado en el mundo de código abierto que dice que los proyectos son impulsados por una comunidad de contribuyentes, pero en realidad, los desarrolladores pagados contribuyen con la mayor parte del código en la mayoría de los proyectos de código abierto modernos.

En resumen, todas estas voces se quejan de dos cosas: los grandes beneficios que obtienen los proveedores de servicios en la nube con su Software sin retribuirles en consecuencia, y la falta de colaboración manteniendo los productos con los que lucran. Sin embargo, no nos engañemos, el quid de la cuestión está principalmente en el dinero: la opinión generalizada de la

comunidad es que el Software de código abierto nunca fue pensado para que las empresas de servicios en la nube lo tomasen y lo vendieran.

Por otro lado, si es posible bifurcar un proyecto libre que se cierra, ¿no hubiese sido mejor colaborar con él antes y haber evitado el cierre? Si no se invierte y se mantiene con salud aquello que da beneficios, puede terminar por desaparecer. A medio camino entre el depredador y el parásito: así es como ven muchas desarrolladoras de código abierto a los proveedores de servicios en la nube.

Vale la pena retomar ahora la frase «el Software de código abierto nunca fue pensado para que las empresas de servicios en la nube lo tomasen y lo vendieran». ¿Para qué fue pensado el código abierto entonces? No hay ninguna licencia de código abierto o Software libre reconocida por la Free Software Foundation o la Open Source Initiative que prohíba hacer negocio con el Software. Lo que prohíben es la discriminación en la capacidad y alcance de su uso en función de la parte, se trate de un individuo o de la mayor multinacional imaginable.

¿Cuál es la solución a un embrollo de tamaña envergadura? Lo único claro es que no es una cuestión de blancos y negros y las consideraciones son demasiadas como para seguir ahondando: empresas que cotizan en bolsa quieren más dinero de otras compañías -que también cotizan en bolsa y por mucho más-, que además del Software ponen la infraestructura sobre la que distribuyen sus ofertas y que tienen la capacidad de clonar tu producto en un abrir y cerrar de ojos, no solo porque tienen el capital, sino porque tienen la experiencia necesaria tras contribuir técnica, pero también en muchos casos, económicamente, durante largo tiempo.

Pese a ello, esta situación está alterando el paradigma actual, en el que el modelo de desarrollo del código abierto se ha impuesto como impulsor de la innovación en el sector empresarial y ya hay quien habla de que nos acercamos al fin, o al principio del fin de la Era del Open Source, cuya preponderancia estaría sentenciada por la revolución de la nube, a la postre el mayor estímulo que haya tenido el código abierto hasta la fecha.

El futuro, pues, pasaría por el Shared Source Software, bajo el cual diferentes compañías con intereses alineados colaborarán en el desarrollo de proyectos concretos, pero limitando su explotación comercial a sí mismas. Todavía no estamos ahí, no obstante, y no parece tampoco que el relevo se vaya a dar en breve. De suceder, será muy llamativo: la muerte del código abierto por un éxito mal entendido.

10.5.4 El Código Abierto como Base de la Competitividad

En septiembre del 2021, se publicó un amplio y detallado informe llevado a cabo por el Fraunhofer ISI y por OpenForum Europe para la Comisión Europea, «The impact of open source software and hardware on technological independence, competitiveness and innovation in the EU economy», cuantifica la importancia económica del código abierto aplicado tanto al Software como al Hardware, su efecto en la contribución al producto interior bruto generado, la reducción en aspectos como el coste total de propiedad, dependencia del proveedor y autonomía digital; lanza una serie de recomendaciones específicas de políticas públicas destinadas a lograr un sector público digitalmente autónomo, una investigación y desarrollo abierto que fomente el crecimiento europeo y una industria más digitalizada y competitiva.

En las estimaciones del informe se apunta que las empresas europeas invirtieron alrededor de mil millones de euros en Software de código abierto en 2018, lo que resultó en un impacto en la economía europea de entre 65,000 y 95,000 millones de euros. El análisis estima una relación costo-beneficio superior a 1:4 y predice que un aumento del 10% de las contribuciones de a repositorios de código abierto sería susceptible de generar anualmente entre un 0.4% y un 0.6% adicional en el PIB, así como más de seiscientas nuevas empresas tecnológicas en la Unión Europea.

El análisis de las contribuciones a repositorios de Software de código abierto en la Unión Europea revela que el ecosistema tiene una naturaleza diferente frente al norteamericano, con un volumen de contribuciones que provienen sobre todo de empleados de compañías pequeñas o muy pequeñas, frente a un escenario en los Estados Unidos en el que predominan grandes compañías tecnológicas que se benefician en sus modelos de negocio de la gran cantidad y de la rápida mejora del Software disponible. En Europa, los contribuyentes individuales ascendieron a más de 260,000, lo que representa el 8% de los casi 3.1 millones de empleados de la UE en el sector del desarrollo de Software en 2018. En total, los más de 30 millones de desarrollos consolidados en repositorios en los estados miembros de la Unión Europea representan una inversión de personal equivalente a casi mil millones de euros, que han pasado a estar disponibles en el dominio público y que, por lo tanto, no tienen que ser desarrollados por otros actores.

Según el análisis, cuanto más pequeña es la empresa, mayor es la inversión relativa en Software de código abierto (las empresas con 50 empleados o menos asumieron casi la mitad de los desarrollos en la muestra de las com-

pañías más activas). Aunque más del 50% de los contribuyentes pertenecen a la industria tecnológica (el 8% del total de sus empleados participaron en estos desarrollos), también hubo participación significativa de empresas de consultoría, científicas, técnicas y, en menor medida, de distribuidores, minoristas y empresas del ámbito financiero.

¿Puede una filosofía de desarrollo como el código abierto, disponible para todo el mundo, llegar a convertirse en una fuente de ventajas diferenciales para el resto de los países, que se ha visto tradicionalmente muy superado en su relevancia en el entorno tecnológico por los gigantes tecnológicos de Estados Unidos o de China? El informe afirma que su uso puede llegar a incidir en gran medida en el desarrollo de una independencia tecnológica superior, de una mayor competitividad y de más innovación. Veremos si llegamos a ver en el resto del mundo políticas que incentiven el uso del código abierto como una variable estratégica clave para ello. La idea, capitalizar la tecnología de una forma más orientada al procomún y al desarrollo colaborativo, suena sin duda atractiva e interesante.

10.5.5 Software Libre en Empresas y Corporaciones

En esta sección exploraremos algunas de las claves por las cuales el Software libre está hoy en el punto de mira de todo tipo de empresas y grandes corporaciones (algunas de las cuales ayer eran sus acérrimos enemigos). Pero hay que empezar destacando que corporativos como Google, Amazon Web Services, Azure de Microsoft, Microsoft, IBM, entre otras, en los últimos años han ganado miles de millones de dólares ofreciendo a sus clientes servicios y/o productos basados en Software libre y con una queja recurrente por su pobre o nula contribución a la comunidad de código abierto que construyó y mantiene esos proyectos.

Si bien es imposible saber exactamente de cuánto dinero estamos hablando, pero es cierto que empresas y corporaciones se benefician diariamente del trabajo de los desarrolladores de código abierto que no emplean.

Grandes Equipos de Programadores GNU/Linux ha demostrado que los equipos de desarrolladores grandes, distribuidos, aunque desorganizados pueden crear Software viable.

Antes de la llegada de GNU/Linux, la mayoría del Software era desarrollado por pequeños equipos de programadores que trabajaban en estrecha coordinación entre sí. Ese era el enfoque recomendado por informáticos de hace

unos decenios, que advertían que añadir más programadores a un proyecto tendía a disminuir su eficiencia. Y estaban muy equivocados.

Desde el principio, el Kernel de Linux se desarrolló con un enfoque diferente, en el que programadores de todo el mundo, que en la mayoría de los casos no se conocían, escribieron e integraron el código de forma rápida y poco organizada. Gracias a la publicación temprana y frecuente, consiguieron que funcionara y hoy día es el Kernel más usado en informática en supercomputadoras y dispositivos móviles.

Pero actualmente hay un mito ampliamente instalado en el mundo de código abierto, que dice que los proyectos son impulsados por una comunidad de contribuyentes gratuitos, pero en realidad, los desarrolladores pagados contribuyen con la mayor parte del código en la mayoría de los proyectos de código abierto modernos de los cuales las corporaciones pueden sacar provecho. Claro ejemplo es el propio Kernel de Linux, en el cual una gran cantidad de desarrolladores actuales pertenecen o son subvencionados por empresas, fundaciones o corporaciones (actualmente cientos de ellas), como en el caso de Linus Torvalds que trabaja bajo los auspicios de la Fundación de Linux.

Reutilización de Software Parte de la razón por la que Linux se hizo muy popular entre los ingenieros de Software con relativa rapidez fue que Linux -y el Software libre en general- facilita la reutilización del código escrito por otras personas.

Hoy en día, la reutilización de Software de terceros es habitual, incluso entre los equipos de desarrollo cuyos productos no son de código libre. Es difícil imaginar la construcción de una aplicación hoy en día sin hacer uso de las bibliotecas de Software de origen, las API de terceros u otros recursos externos a su propio proyecto.

Es cierto que proyectos como GNU, que precedió al Kernel de Linux en siete años, promovían la reutilización de código antes de que apareciera el núcleo. Pero, podría decirse que Linux fue el proyecto que trajo las prácticas de codificación libre a la corriente que tanto parece interesar a ciertas grandes empresas, ayudando a crear el modelo de ingeniería de Software de componentes de Software modulares y reutilizables.

Gestión Actual del Código Fuente Linus Torvalds, que creó el núcleo de Linux cuando era estudiante en Helsinki, es el más famoso por ese trabajo.

Pero un hecho a menudo olvidado es que Torvalds es también el padre de Git, el masivamente popular gestor de código fuente libre.

Torvalds creó Git para ayudar a gestionar el código fuente de Linux. Si Linux no existiera, tampoco existiría Git. Tampoco existiría GitHub, ni GitLab, ni GitOps. Y, lo que es más importante, sin la idea de Software libre y colaborativo de Richard Stallman, tampoco existiría la cultura de intercambio y colaboración abierta que sostienen estas tecnologías.

Estrategias de Despliegue de Software "App Store" Apple puede atribuirse el mérito de haber lanzado la primera App Store, un lugar donde los desarrolladores pueden compartir aplicaciones y los usuarios pueden instalarlas fácilmente, utilizando un catálogo Online centralizado.

Pero al igual que con muchas cosas que ha hecho Apple, el concepto de App Store (que ahora es una estrategia de despliegue de Software apilado como servicio, especialmente pero no sólo en el ecosistema móvil) se parece mucho a lo que los desarrolladores de GNU/Linux estaban haciendo a través de los repositorios de Software mucho antes de que las tiendas de aplicaciones se convirtieran en algo común en el mundo del Software propietario, como también lo es la Tienda de Windows.

Los repositorios de Software en GNU/Linux hacen más o menos lo mismo que las tiendas de aplicaciones: Permiten a los usuarios seleccionar las aplicaciones que quieren de una lista centralizada y en línea, y luego instalarlas con unos pocos clics o bien órdenes de terminal (como el famoso *apt* de Debian).

Es cierto que empresas como Apple parecen tener el mérito de crear tiendas de aplicaciones muy fáciles de usar de hacer clic e ir, pero no es un invento de ellos. Y la historia del concepto de tienda de aplicaciones en general implica a más actores que sólo la comunidad de Apple. Aún así, creo que se podría argumentar con fuerza que, sin GNU/Linux y los repositorios de Software de GNU/Linux, las tiendas de aplicaciones tal y como las conocemos hoy no existirían.

Formatos Abiertos para Intercambio de Información Hay una gran variedad de tecnologías disponibles para producir y almacenar datos. Como son: hojas de cálculo, bases de datos, Software estadístico más específico y más. Esto genera una enorme diversidad de formatos, a veces esto es por decir lo menos caótico.

La ventaja de los archivos de formatos abiertos, es que permiten a los de-

sarrolladores producir varios paquetes de Software y servicios utilizando esos formatos. Esto entonces reduce al mínimo los obstáculos para la reutilización de la información que contienen.

El advenimiento del Software libre ha generado algunos de los formatos abiertos más usados para el intercambio de información, pero los entes generadores de información no siempre se adecuan a los niveles de apertura deseados, algunos de estos formatos abiertos son: XML, JSON, YAML, RDF, REBOL, PDF, CSV, ODF, OOXML, TXT, HTML, HDF.

Pero, incluso si la información se proporciona en formato electrónico, formato legible por máquina y en detalle, puede existir problemas relacionados con el formato del archivo en sí (principalmente el generado por los diversos sistemas operativos). Los formatos en los cuales la información es publicada -en otras palabras, la base digital en la cual la información es almacenada- puede ser "abierta" o "cerrada".

Un formato abierto es aquel donde las especificaciones del Software están disponibles para cualquier persona, de forma gratuita, así cualquiera puede usar dichas especificaciones en su propio Software sin ninguna limitación en su reutilización que fuere impuesta por derechos de propiedad intelectual.

Si el formato del archivo es "cerrado", esto puede ser debido a que el formato es propietario y sus especificaciones no están disponibles públicamente, o porque el formato es propietario y aunque las especificaciones se han hecho públicas, su reutilización es limitada. Si la información es liberada en un formato de archivo cerrado, esto puede causar grandes obstáculos para reutilizar la información codificada en él, forzando a aquellos que deseen usar la información a comprar Software innecesario.

El uso de formatos de archivo con propiedad, para el que la especificación no está disponible públicamente, puede crear dependencia de Software de terceros o de los titulares de licencias de los formatos de archivos. En el peor de los casos, esto puede significar que la información sólo se puede leer con cierto Software específico, que puede ser caro, o que puede quedar obsoleto.

La preferencia del término Gobierno de Datos Abiertos, es que la información se publicará en formatos de archivo abiertos, los cuales son de lectura mecánica y esto es una aportación más del Software libre.

Ciencia Abierta La ciencia abierta (Open Science) es el movimiento creciente para hacer que la ciencia sea abierta. La ciencia en sí misma se utilizó como un ejemplo principal de la eficacia del movimiento de código abierto, ci-

tando prácticas como la difusión abierta de información, métodos y revisión por pares de la literatura científica. Podría decirse que la ciencia abierta comenzó en el siglo XVII con el advenimiento de la revista científica y la práctica de repetir los experimentos presentados en los artículos académicos. Estas revistas se imprimirían y distribuirían en todo el mundo, a menudo supervisadas por sociedades científicas como la Royal Society.

¿Qué impulsó la necesidad de un movimiento de ciencia abierta? La Royal Society tenía el famoso lema "Nullius in verba", traducido de forma aproximada como "no tome la palabra de nadie". Esto encarnaba un principio general en la ciencia de que todas las teorías están abiertas a ser cuestionadas y los resultados declarados deben ser repetibles. De hecho, es una práctica generalizada que fue realizada por la sociedad en esos primeros años. En los últimos años esta práctica no ha sido tan común, con más y más ciencia confiando en elementos cerrados, lo que en última instancia conduce a errores que son más difíciles de detectar sin un intercambio completo de información: datos, métodos y publicaciones.

El movimiento de ciencia abierta afirma en términos generales que la ciencia debe realizarse de manera abierta y reproducible donde todos los componentes de la investigación estén abiertos. Muchas revistas permanecen estancadas en un formato en el que se imprimían físicamente, a pesar de que en la actualidad se distribuyen en gran medida en línea. A menudo, todavía utilizan archivos PDF como una forma de "papel electrónico" con publicaciones fijas, procesos cerrados de revisión por pares y poco o ningún acceso a los datos. Este fue sin duda el modo más eficiente de difundir el conocimiento científico antes de los albores de internet, pero ahora un número cada vez mayor lo considera lejos de ser el óptimo.

La ciencia abierta encarna una serie de aspectos, en el núcleo esto incluye acceso abierto, datos abiertos, código abierto y estándares abiertos que ofrecen una diseminación sin restricciones del discurso científico. Estas cosas permiten una ciencia reproducible al brindar acceso completo a los componentes principales de la investigación científica. Hay una serie de componentes adicionales que también se están explorando, como la revisión por pares abierta, donde los revisores de publicaciones científicas publican revisiones abiertamente con su nombre adjunto y la ciencia de libreta abierta donde las libretas (tradicionalmente cerradas) se publican abiertamente en línea a medida que se realiza la investigación.

¿Por qué la ciencia abierta es tan importante en la era digital? También existe una creciente comprensión de que, dado que la investigación científica

depende cada vez más del código informático para simulaciones, cálculos, análisis, visualización y procesamiento de datos en general, es importante tener acceso a este código tal como tradicionalmente ha sido importante mostrar (y derivar) cualquier nueva técnica matemática introducida para el análisis. Hay revistas como PLOS ONE y F1000 que exploran el significado de las publicaciones, ya sea que se deben congelar en el tiempo o se pueden actualizar. Los repositorios de datos también están ganando importancia a medida que las agencias de financiación requieren la publicación y preservación de los datos generados por la investigación financiada.

En esencia, la ciencia abierta se trata de volver a esos valores fundamentales inculcados por algunos de los primeros científicos de que no debemos confiar en la palabra de nadie, que es esencial que todos los elementos pertinentes a un descubrimiento pretendido se publiquen para que los resultados puedan repetirse y validarse. El movimiento de la ciencia abierta varía en el grado en que lo requiere, pero están surgiendo patrones. Se están estableciendo recomendaciones sobre licencias, como CC0 para datos, CC-BY para publicaciones, licencias compatibles con OSI para código fuente y formatos abiertos para datos. En última instancia, se trata de empoderar a todos para que participen en la ciencia, con internet como vehículo principal para la amplia difusión de este conocimiento.

Este movimiento está cambiando la forma en que se hace la ciencia, está recibiendo el respaldo de muchas agencias de financiamiento, ya que requieren planes de gestión de datos, planes de distribución de código fuente y una mayor validación de los resultados a través del acceso abierto a estos resultados para todos. Esto también mejora la transferencia de conocimientos de la academia a la industria, ya que se brinda acceso completo en el momento de la publicación o después de un período de embargo. El movimiento de la ciencia abierta se limita en gran medida a la investigación que está financiada por las agencias de financiación nacionales de todo el mundo y exige que todos los que financian la investigación tengan acceso total e igualitario a ella.

Open Hardware El concepto de Software libre también se permeó al Hardware. El término Open Hardware u Open Source Hardware, se refiere al Hardware cuyo diseño se hace públicamente disponible para que cualquiera pueda estudiarlo, modificarlo y distribuirlo, además de poder producir y vender Hardware basado en ese diseño. Tanto el Hardware como el Software

que lo habilita, siguen la filosofía del Software libre. Hoy en día, el término "hágalo usted mismo" (DIY por sus siglas en inglés) se está popularizando en el Hardware gracias a proyectos como Arduino que es una fuente abierta de prototipos electrónicos, una plataforma basada en Hardware flexible y fácil de utilizar que nació en Italia en el año 2005.

El movimiento de Hardware abierto o libre, busca crear una gran librería accesible para todo el mundo, lo que ayudaría a las compañías a reducir en millones de dólares en trabajos de diseño redundantes. Ya que es más fácil tener una lluvia de ideas propuesta por miles o millones de personas, que por solo una compañía propietaria del Hardware, tratando así de que la gente interesada entienda cómo funciona un dispositivo electrónico, pueda fabricarlo, programarlo y poner en práctica esas ideas en alianza con las empresas fa-bricantes, además se reduciría considerablemente la obsolescencia programada y en consecuencia evitaríamos tanta basura electrónica que contamina el medio ambiente. Al hablar de Open Hardware hay que especificar de qué tipo de Hardware se está hablando, ya que está clasificado en dos tipos:

- Hardware estático. Se refiere al conjunto de elementos materiales de los sistemas electrónicos (tarjetas de circuito impreso, resistencias, capacitores, LEDs, sensores, etcétera).

- Hardware reconfigurable. Es aquél que es descrito mediante un HDL (Hardware Description Language). Se desarrolla de manera similar a como se hace Software. Los diseños son archivos de texto que contienen el código fuente.

Para tener Hardware reconfigurable debemos usar algún lenguaje de programación con licencia GPL (General Public License). La licencia GPL, al ser un documento que cede ciertos derechos al usuario, asume la forma de un contrato, por lo que usualmente se le denomina contrato de licencia o acuerdo de licencia. La Organización Europea para la investigación Nuclear (CERN) publicó el 8 de julio de 2011 la versión 1.1 de la Licencia de Hardware Abierto.

Existen programas para diseñar circuitos electrónicos y aprender de la electrónica como EDA (Electronic Design Automation) y GEDA (GPL Electronic Design Automation), son aplicaciones de Software libre que permiten poner en práctica las ideas basadas en electrónica.

Es posible realizar el ciclo completo de diseño de Hardware reconfigurable desde una máquina con GNU/Linux, realizándose la compilación, simulación,

síntesis y descarga en una FPGA (Field Programmable Gate Arrays). Para la compilación y simulación se puede usar GHDL (https://ghdl.free.fr) junto con GTKWave (https://gtkwave.sourceforge.net) y para la síntesis el entorno ISE de Xilinx. Este último es Software comercial pero existe una versión gratuita con algunas restricciones.

Sabemos que tanto en el caso del Software como el Hardware, libre no es lo mismo que gratis. Específicamente, en el caso del Hardware, como estamos hablando de componentes físicos que son fabricados, la adquisición de componentes electrónicos puede ser costosa. Aun así, es un campo que no solo es apasionante sino que también tiene mucho futuro y representa grandes oportunidades.

Entusiasmo de la Comunidad Por último, pero no menos importante, probablemente el mayor impacto duradero de GNU/Linux en el modelo de ingeniería de Software se reduce a lo que podría llamarse entusiasmo de la comunidad. Me refiero a la forma en que GNU/Linux en particular, y el Software libre en general, ha animado a los desarrolladores de todo tipo a considerar las contribuciones a la comunidad como uno de sus objetivos finales y esto ahora es notorio no solo en Software, sino en Hardware abierto, obras literarias, escritos técnicos (como este trabajo), imágenes, vídeo, música y un largo etc.

En un mundo de código libre en el que las contribuciones a los proyectos de código pueden ser aceleradores de carrera y el código de licencia libre se reutiliza ampliamente, los desarrolladores entienden que hay un valor real en la construcción de Software que puede beneficiar a tantos usuarios como sea posible.

Tal vez los desarrolladores valorarían a la comunidad en su conjunto si GNU/Linux y el código libre nunca hubieran aparecido. Pero me cuesta imaginar un mundo en el que corporaciones como Microsoft y Google trabajarán juntos en la construcción de Software para GNU/Linux si GNU/Linux no hubiera popularizado el concepto de proyectos de Software impulsados por la comunidad que nadie posee realmente, pero que todos pueden utilizar.

Si bien, es innegable que todo lo anterior puso en la mira de las empresas de todos los tamaños y de las grandes corporaciones el Software libre, la principal razón es el poder utilizar una gran cantidad Software funcional, depurado y ampliamente usado para ofrecer servicios y/o productos basados en Software libre y así beneficiarse económicamente de ello.

Por otro lado, se ha visto a través de múltiples estudios, el impacto y la cuantificación de la importancia económica del código abierto aplicado tanto al Software como al Hardware, su efecto en la contribución al producto interior bruto generado, la reducción en aspectos como el coste total de propiedad, dependencia del proveedor y autonomía digital. Además de generar políticas públicas destinadas a lograr un sector público digitalmente autónomo, una investigación y desarrollo abierto que fomente el crecimiento de los países y una industria más digitalizada y competitiva.

Retomando la frase «el Software de código abierto nunca fue pensado para que las empresas de servicios lo tomasen y lo vendieran». ¿Para qué fue pensado el código abierto, entonces? No hay ninguna licencia de código abierto o Software libre reconocida por la Free Software Foundation o la Open Source Initiative que prohíba hacer negocio con el Software. Lo que prohíben es la discriminación en la capacidad y alcance de su uso en función de la parte, se trate de un individuo o de la mayor multinacional imaginable.

Pese a ello, esta situación está alterando el paradigma actual, en el que el modelo de desarrollo del código abierto se ha impuesto como impulsor de la innovación en el sector empresarial, a la postre el mayor estímulo que haya tenido el código abierto hasta la fecha.

¿Puede una filosofía de desarrollo como el código abierto, disponible para todo el mundo, llegar a convertirse en una fuente de ventajas diferenciales para el resto de los países, que se ha visto tradicionalmente muy superado en su relevancia en el entorno tecnológico por los gigantes tecnológicos de Estados Unidos o de China? Se afirma que su uso puede llegar a incidir en gran medida en el desarrollo de una independencia tecnológica superior, de una mayor competitividad y de más innovación. La idea, capitalizar la tecnología de una forma más orientada al procomún y al desarrollo colaborativo, suena sin duda atractiva e interesante pero no libre de inconvenientes para algunos sectores de desarrolladores de Software libre.

10.6 Código Abierto y las Organizaciones Internacionales

Aunque la Organización de las Naciones Unidas (ONU) ha hablado previamente bien del desarrollo del código abierto, varios eventos recientes muestran que la ONU está tomando medidas definitivas para presentar al mundo entero el camino del código abierto. En julio del 2021, el Consejo Económico y Social de la ONU (ECOSOC) adoptó un proyecto de resolución presentado por el representante de Pakistán titulado: Tecnologías de fuente abierta para

el desarrollo sostenible.

10.6.1 Las Naciones Unidas y el Código Abierto

El ECOSOC destacó la disponibilidad de tecnologías de código abierto que pueden contribuir a los Objetivos de Desarrollo Sostenible (ODS). El consejo invitó al Secretario General a "desarrollar propuestas específicas sobre formas de aprovechar mejor las tecnologías de código abierto para el desarrollo sostenible basadas en las aportaciones de los Estados Miembros interesados y otras partes interesadas".

El desarrollo de tecnología de código abierto puede ser una herramienta rápida y eficaz para la innovación. Aplicarlo a tecnologías apropiadas para ayudar a alcanzar los ODS es extremadamente prometedor. Las "tecnologías apropiadas" abarcan opciones y aplicaciones tecnológicas que son a pequeña escala, económicamente asequibles, descentralizadas, energéticamente eficientes, ambientalmente racionales y fácilmente utilizadas por las comunidades locales para satisfacer sus necesidades.

Existe un caso particularmente fuerte para las tecnologías apropiadas de código abierto OSAT (Outsourced Semiconductor Assembly and Test). OSAT podría ayudar a todos a salir de la pobreza y alcanzar un estado sostenible aprovechando el mismo tipo de desarrollo que hace que el Software de código abierto sea un éxito rotundo.

La Declaración Ministerial del Foro político de alto nivel sobre desarrollo sostenible también destacó la importancia de "tecnologías no patentadas que pueden contribuir a los Objetivos de Desarrollo Sostenible, a través de diversas fuentes de acceso abierto". Pidió "el desarrollo y la puesta en funcionamiento de una plataforma en línea en el marco del Mecanismo de facilitación de la tecnología para establecer un mapeo integral y servir como puerta de entrada a la información sobre iniciativas, mecanismos y programas de ciencia, tecnología e innovación existentes, dentro y fuera de las Naciones Unidas".

Es un pequeño paso, pero muy emocionante, porque las Naciones Unidas no se demoran una vez que ven formas de ayudar a sus Estados Miembros y a las personas que los integran. Ahora, el Departamento de Asuntos Económicos y Sociales de las Naciones Unidas (DESA) está trabajando para que esto suceda. DESA está utilizando una Nota sobre una base de datos centralizada propuesta de las Naciones Unidas de tecnologías apropiadas de código abierto publicada por la Conferencia de las Naciones Unidas sobre Comercio

y Desarrollo (UNCTAD) para hacerlo.

La Nota de la UNCTAD aboga por una base de datos centralizada de OSAT para acelerar el descubrimiento y la innovación en todos los sectores asociados con los ODS al tiempo que se minimizan los obstáculos legales o financieros. Esto es importante para la difusión del acervo mundial de conocimientos, especialmente en los países en desarrollo.

Actualmente, no existe un repositorio completo o una base de datos central de OSAT y Appropedia.org, quizás sea el mejor ejemplo. Sin embargo, la Nota de la UNCTAD dice: "Muchas organizaciones, organizaciones sin fines de lucro y empresas con fines de lucro están desarrollando OSAT y manteniendo bases de datos existentes a pequeña escala. Si bien hay muchos OSAT disponibles, se encuentran dispersos en varias bases de datos para tecnologías particulares. Mientras tanto, sigue existiendo una clara necesidad de aumentar la tasa de uso de OSAT.

Por lo tanto, existe una necesidad urgente de una base de datos de código abierto centralizada global (COSD) confiable. Al tener un alcance global, un repositorio de COSD proporcionaría una ventanilla única a la que todos pueden acceder para resolver los desafíos locales".

Concluye: "La ONU está bien posicionada para liderar el establecimiento de un COSD dado su papel bien establecido en la promoción de la tecnología de código abierto a través de varios foros y publicaciones intergubernamentales. En particular, 2030 Connect es una plataforma tecnológica en línea de la ONU que se desarrolló como parte del trabajo del Equipo de Trabajo Interinstitucional de la ONU. El COSD podría mejorarlo".

Con el liderazgo de la ONU, quizás no estemos demasiado lejos de cuándo, sí tiene un problema local (sin importar en qué parte del mundo se encuentre), pueda descargar una solución de código abierto examinada y probada. Quizás, esta es la potencia de fuego que necesitamos para alcanzar los ambiciosos Objetivos de Desarrollo Sostenible.

10.6.2 La Comisión Europea se Compromete a Liberar Todo el Software que Pueda Beneficiar a la Sociedad

A finales del 2021, la Unión Europea (UE) y su órgano legislativo, la Comisión Europea siguen avanzando en su estrategia digital con el Software de código abierto como uno de los pilares fundamentales. En esta ocasión ha sido esta última la que anuncia novedades para con la distribución del Software desarrollado para cubrir necesidades internas de la organización.

De acuerdo a la información publicada, la Comisión Europea ha aprobado una nueva regulación que favorece el libre acceso al Software que producen siempre y cuando existan beneficios potenciales para «los ciudadanos, las empresas u otros servicios públicos», lo que de la teoría a la práctica bien puede abarcar todo lo que se desarrolle bajo su tejado.

Esta nueva disposición se apoya a su vez en un reciente estudio realizado también por la Comisión sobre el impacto del Software de código abierto en áreas como la independencia tecnológica, la competitividad y la innovación en la economía de la Unión Europea. El objetivo, hallar evidencias sólidas con las que conformar las políticas europeas de código abierto para los próximos años.

En términos económicos, de hecho, los cálculos son de lo más optimistas y apuntan un impacto económico contundente, de miles de millones de euros de ahorro al año -a modo de ejemplo, se estimó entre 65 y 95 mil millones de euros solo en 2018- y con un incremento mínimo en la apuesta, se podría dar un crecimiento del PIB de la UE de en torno a los 100,000 millones de euros.

Con semejante escenario, no es de extrañar que la misma Comisión Europea esté interesada en promover las soluciones de código abierto dentro y fuera de las instituciones y no solo se basan en el beneficio económico directo: son muchas otras las ventajas del modelo también recogidas en el informe, tal y como se ha mencionado: independencia, competitividad, innovación... y en el caso de las administraciones públicas, colaboración, reutilización y transparencia.

En palabras de Johannes Hahn, comisario de Presupuesto y Administración: «El código abierto ofrece grandes ventajas en un ámbito en el que la UE puede desempeñar un papel de liderazgo. Las nuevas normas aumentarán la transparencia y ayudarán a la Comisión, así como a los ciudadanos, las empresas y los servicios públicos de toda Europa, a beneficiarse del desarrollo de Software de código abierto. Poner en común los esfuerzos para mejorar el Software y la creación conjunta de nuevas funciones reduce los costes para la sociedad, ya que también nos beneficiamos de las mejoras realizadas por otros desarrolladores. Esto también puede mejorar la seguridad, ya que especialistas externos e independientes comprueban los fallos y las deficiencias de seguridad de los programas informáticos».

La comisaría de Innovación, Investigación, Cultura, Educación y Juventud, Mariya Gabriel, ha declarado: «La Comisión pretende, con su ejemplo, estar al frente de la transición digital en Europa. Con las nuevas normas, la

Comisión aportará un valor significativo a las empresas, también las emergentes, a los innovadores, a los ciudadanos y las administraciones públicas, poniendo a su disposición el código abierto de sus soluciones informáticas. Esta decisión también ayudará a estimular la innovación, gracias al código de la Comisión disponible públicamente».

Como muestra del Software desarrollado bajo el amparo de la Comisión Europea que va a ser liberado se incluyen proyectos como eSignature, «un conjunto de normas, herramientas y servicios gratuitos que ayudan a las administraciones públicas y a las empresas a acelerar la creación y verificación de firmas electrónicas jurídicamente válidas en todos los Estados miembros de la UE»; o LEOS (Legislation Editing Open Software), «el Software utilizado en toda la Comisión para elaborar textos jurídicos. LEOS, escrito originalmente para la Comisión, se está desarrollando en estrecha colaboración con Alemania, España y Grecia».

Esta nueva iniciativa de la Comisión Europa contempla asimismo la creación de un repositorio centralizado para facilitar el descubrimiento, el acceso y la reutilización del Software incluido, el cual se sumará a todos los proyectos realizados por las diferentes administraciones públicas comunitarias en base al mismo modelo de desarrollo. Y viene de lejos este impulso, aun cuando comienza a unificarse ahora.

Sin ir más lejos, hace años que la propia Comisión Europea puso en marcha el programa Interoperable Delivery of European eGovernment Services to Public Administrations, Businesses and Citizens que dio origen al observatorio JoinUp (https://joinup.ec.europa.eu/), en cuyas páginas se recogen casi 3,000 soluciones de Software abierto, 133 colecciones de recursos y cuantiosa información relacionada.

Más tarde, de 2014 a 2017 se inició la «primera fase» en la estrategia de código abierto de la Unión Europea, especialmente dentro de la propia Comisión, estableciendo determinados requisitos en materia de Software de código abierto; actualmente se está desarrollando la nueva «estrategia de código abierto 2020-2023», con la que la Comisión Europea pretende ampliar y afianzar los objetivos de la estrategia digital y la contribución al programa Europa Digital.

11 Bibliografía

Este texto es una recopilación de múltiples fuentes, mi aportación -si es que puedo llamarla así- es plasmarlo en este documento, en el que trato de dar coherencia a mi visión de los temas desarrollados.

En la realización de este texto se han revisado -en la mayoría de los casos indico la referencia, pero pude omitir varias de ellas, por lo cual pido una disculpa- múltiples páginas Web, artículos técnicos, libros, entre otros materiales bibliográficos, los más representativos y de libre acceso los pongo a su disposición en la siguiente liga:

Herramientas
http://132.248.181.216/Herramientas/

Referencias

[1] A. Toselli, O. Widlund, *Domain Decomposition Methods - Algorithms and Theory*. Springer, 2005. 96

[2] B. I. Wohlmuth, *Discretization Methods and Iterative Solvers Based on Domain Decomposition*. Springer, 2003. 74

[3] L. F. Pavarino and A. Toselli, *Recent Developments in Domain Decomposition Methods*. Springer, 2003. 74

[4] M.B. Allen III, I. Herrera & G. F. Pinder, *Numerical Modeling in Science And Engineering.* John Wiley & Sons, Inc . 1988. 69, 70, 73, 74

[5] R. L. Burden and J. D. Faires, *Análisis Numérico*. Math Learning, 7 ed. 2004. 69

[6] S. Friedberg, A. Insel, and L. Spence, *Linear Algebra*, 4th Edition, Prentice Hall, Inc. 2003. 69

[7] Y. Saad, *Iterative Methods for Sparse Linear Systems*. SIAM, 2 ed. 2000. 53, 59, 61, 70, 73, 77

[8] Y. Skiba, *Métodos y Esquemas Numéricos, un Análisis Computacional*. UNAM, 2005. 69, 74, 80

[9] M. Diaz, I. Herrera, *Desarrollo de Precondicionadores para los Procedimientos de Descomposición de Dominio*. Unidad Teórica C, Posgrado de Ciencias de la Tierra, 22 pags, 1997. 96

[10] I. Herrera, *Un Análisis del Método de Gradiente Conjugado*. Comunicaciones Técnicas del Instituto de Geofísica, UNAM, Serie Investigación, No. 7, 1988. 77, 83

[11] G. Herrera, Análisis de Alternativas al Método de Gradiente Conjugado para Matrices no Simétricas. Tesis de Licenciatura, Facultad de Ciencias, UNAM, 1989. 83

[12] C.T. Keller, *Iterative Methods for Linear and Nonlinear Equations*, Societe for Industrial and Applied Mathematics, 1995. 77, 80

[13] Handbook of Floating-Point Arithmetic 2010th Edition, Jean-Michel Muller, Nicolas Brisebarre, Et alii, Birkhäuser, 2010.

[14] What Every Computer Scientist Should Know About Floating-Point Arithmetic, David Goldbergm, ACM Computing Surveys, Vol 23, No 1, March 1991

[15] 754-2008 - IEEE Standard for Floating-Point Arithmetic. IEEE. 29 de agosto de 2008. ISBN 978-0-7381-5752-8. doi:10.1109/IEEESTD.2008.4610935. (NB. Superseded by IEEE Std 754-2019, a revision of IEEE 754-2008.)

[16] Stochastic Rounding and Its Probabilistic Backward Error Analysis, Michael P. Connolly, Nicholas J. Higham , Methods and Algorithms for Scientific Computing, SIAM Journal on Scientific Computing Vol. 43, Iss. 1 (2021)

[17] IEEE, https://www.ieee.org/

[18] Intel, https://www.intel.la

[19] AMD, https://www.amd.com

[20] ARM, https://www.arm.com/

[21] Cerebras, https://www.cerebras.net/

Declaro terminado este trabajo sufrido, ideado y llevado a cabo entre los años 2007 al 2025, aún y a pesar de impedimentos tales como: la mala suerte, la desventura, el infortunio, la incomprensión, la gripe, el Covid-19, las horas de frío y de calor, la tristeza, la desesperanza, el cansancio, el presente, el pasado y mi futuro, el que dirán, la vergüenza, mis propias incapacidades y limitaciones, mis aversiones, mis temores, mis dudas y en fin, todo aquello que pudiera ser tomado por mi, o por cualquiera, como obstáculo en este tiempo de mentiras, verdades, de incredulidad e ignorancia o negación de la existencia real y física de la mala fe.

Atentamente

Antonio Carrillo Ledesma