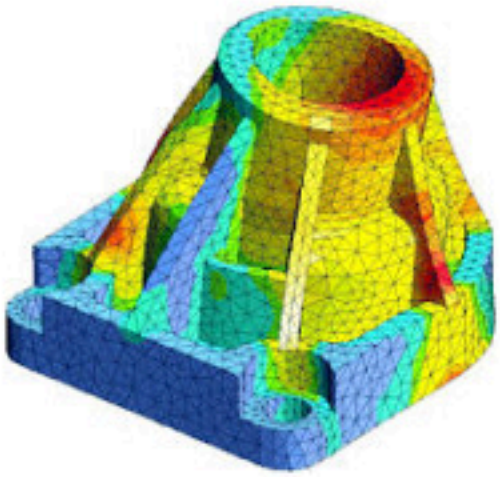
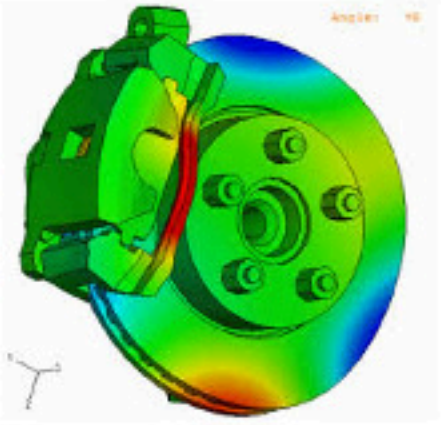
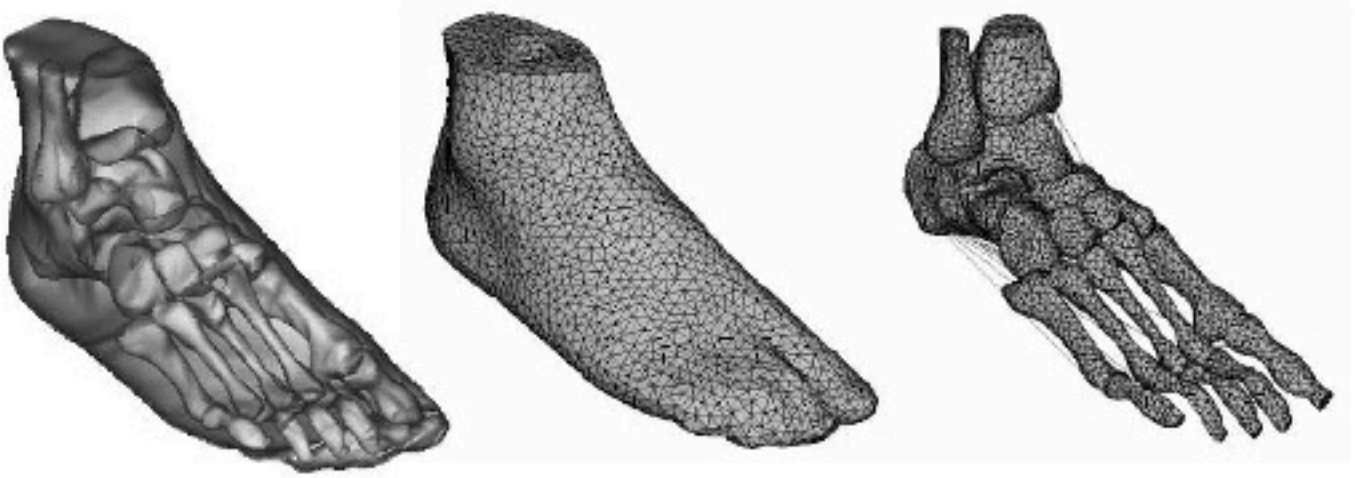
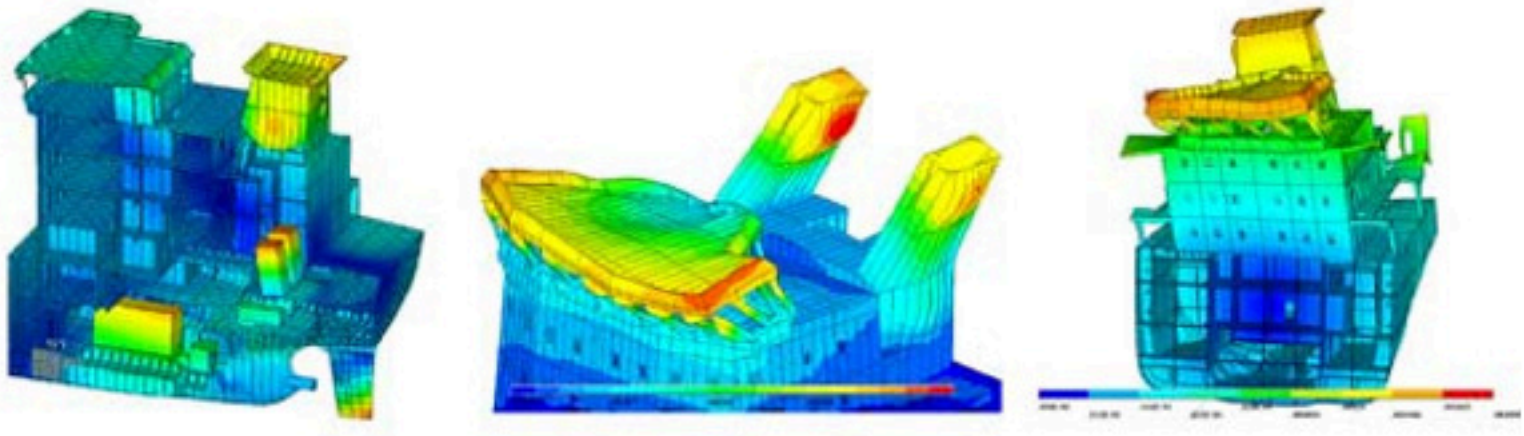
Introducción al Método de Descomposición de Dominio de Subestructuración









Antonio Carrillo Ledesma

Introducción al Método de Descomposición de Dominio de Subestructuración

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

La necesidad de entender su entorno y anticiparse a los acontecimientos tiene raíces muy profundas en el ser humano. Desde la prehistoria, el hombre trató de predecir a la naturaleza, pues de ella dependía su supervivencia, para lo cual inicialmente nuestros antepasados utilizaron a la brujería, así como el pensamiento mágico y el religioso. Sin embargo, el medio más efectivo para predecir el comportamiento de la naturaleza es el método científico (o su antecesor el método empírico) y es por eso que este anhelo humano ancestral, a través de la historia, ha sido motor de la ciencia.

La maduración y el progreso de la predicción científica es, sin duda, el resultado del avance general de la ciencia, pero además ha habido elementos catalizadores esenciales sin los cuales esto no hubiera sido posible. La predicción científica, además de ser científica, es matemática y computacional. En la actualidad, cuando deseamos predecir el comportamiento de un sistema, los conocimientos científicos y tecnológicos se integran en modelos matemáticos los cuales se convierten en programas de cómputo que son ejecutados por las computadoras tanto secuenciales como paralelas (entenderemos por una arquitectura paralela a un conjunto de procesadores interconectados capaces de cooperar en la solución de un problema).

Aunque el sólo hecho de poseer la capacidad de predicción científica nos llena de satisfacción y orgullo, sin embargo, aún más trascendente es el hecho de que ella también es la base de una gran parte del extraordinario progreso material que la humanidad ha experimentado en épocas recientes. En efecto, nuestra facultad para predecir es una herramienta muy poderosa de la ingeniería, de la tecnología y de la ciencia misma, la cual, entre otras muchas cosas, nos ha permitido ampliar la disponibilidad de los recursos naturales y utilizarlos con mayor eficiencia.

Para resolver, por ejemplo, algún problema del área de ciencias e ingeniería (el movimiento de un fluido libre o de un fluido en un medio poroso) lo primero que debemos de hacer es formular un modelo matemático del problema a tratar. Esto se logra a través de los fundamentos de la física macroscópica los cuales son proporcionados por la 'teoría de los medios continuos' [8]. Con base en ella se introduce una formulación clara, general y sencilla de los modelos matemáticos de los sistemas continuos.

La formulación es tan sencilla y tan general, que los modelos básicos de sistemas tan complicados y diversos como la atmósfera, los océanos, los yacimientos petroleros, o los geotérmicos, se derivan por medio de la apli-

cación repetida de una sola ecuación diferencial: 'la ecuación diferencial de balance' [34].

Dicha formulación también es muy clara, pues en el modelo general no hay ninguna ambigüedad; en particular, todas las variables y parámetros que intervienen en él, están definidos de manera unívoca. En realidad, este modelo general de los sistemas continuos constituye una realización extraordinaria de los paradigmas del pensamiento matemático. El descubrimiento del hecho de que los modelos matemáticos de los sistemas continuos, independientemente de su naturaleza y propiedades intrínsecas, pueden formularse por medio de balances, cuya idea básica no difiere mucho de los balances de la contabilidad financiera, fue el resultado de un largo proceso de perfeccionamiento en el que concurrieron una multitud de mentes brillantes.

Los modelos matemáticos de los sistemas continuos son 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, o sistemas de tales ecuaciones. Salvo para 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.

La capacidad para formular los modelos matemáticos de sistemas complicados y de gran diversidad, es sin duda una contribución fundamental 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 matemáticos y la computación electrónica. En cambio, la diversidad y complejidad de problemas que pueden ser tratados con métodos numéricos y computacionales es impresionante.

Los modelos de los sistemas continuos -es decir, sistemas físicos macroscópicos tales como los yacimientos petroleros, la atmósfera, los campos electromagnéticos, los océanos, los metalúrgicos, el aparato circulatorio de los seres humanos, la corteza terrestre, lo suelos y las cimentaciones, muchos sistemas ambientales, y muchos otros cuya enumeración ocuparía un espacio enormecontienen un gran número de grados libertad.

Por ello, la solución numérica por los esquemas tradicionales tipo diferencias finitas y elemento finito generan una discretización del problema, la cual es usada para generar sistemas de ecuaciones algebraicos [34]. Estos sistemas algebraicos en general son de gran tamaño para problemas reales, al ser estos algoritmos secuenciales su implantación suele hacerse en equipos secuenciales y por ello no es posible resolver muchos problemas que involucren el uso de una gran cantidad de memoria.

Actualmente para tratar de subsanar la limitante de procesar sólo en equipos secuenciales, se usan equipos paralelos para soportar algoritmos secuenciales mediante directivas de compilación, haciendo ineficiente su implantación en dichos equipos.

La computación en paralelo es una técnica que nos permite distribuir una gran carga computacional entre muchos procesadores. Y es bien sabido que una de las mayores dificultades del procesamiento en paralelo es la coordinación de las actividades de los diferentes procesadores y el intercambio de información entre los mismos [14]. Al usar métodos de descomposición de dominio conjuntamente con el cómputo en paralelo (supercomputadoras, clusters o grids) nos permite atacar una gran variedad de problemas que sin estas técnicas sería imposible hacerlo de manera eficiente.

1.1 Métodos de Descomposición de Dominio

Los métodos de descomposición de dominio introducen desde la formulación matemática del problema una separación natural de las tareas a realizar por el método y simplifican considerablemente la transmisión de información entre los subdominios [3]. En ellos, los sistemas físicos representados por su modelo son descompuestos en varios subdominios.

Los métodos de descomposición de dominio permiten tratar los problemas de tamaño considerable, empleando algoritmos paralelos en computadoras secuenciales y/o paralelas. Esto es posible ya que cualquier método de descomposición de dominio se basa en la suposición de que dado un dominio computacional Ω, este se puede particionar en subdominios $\Omega_i, i = 1, 2, ..., E$ entre los cuales puede o no existir traslape [1] y [3]. Entonces, el problema es reformulado en términos de cada subdominio (empleando algún método de discretización) obteniendo una familia de subproblemas de tamaño reducido independientes en principio entre sí, y que están acoplados a través de la solución en la interfaz de los subdominios que es desconocida.

De esta manera, podemos clasificar de forma burda a los métodos de descomposición de dominio, como aquellos en que: existe traslape entre los subdominios y en los que no existe traslape. A la primera clase pertenece el método de Schwarz (en el cual el tamaño del traslape es importante en la convergencia del método) y a los de la segunda clase pertenecen los métodos del tipo subestructuración (en el cual los subdominios sólo tienen en común los nodos de la frontera interior).

Estos métodos son un paradigma natural usado por la comunidad de mo-

deladores. Los sistemas físicos son descompuestos en dos o más subdominios contiguos basados en consideraciones fenomenológicas. Estas descomposiciones basadas en dominios físicos son reflejadas en la ingeniería de Software del código correspondiente.

Así, mediante los métodos de descomposición de dominio, la programación orientada a objetos (que nos permite dividir en niveles la semántica de los sistemas complejos tratando así con las partes, más manejables que el todo, permitiendo su extensión y un mantenimiento más sencillo) y esquemas de paralelización que usan el paso de mensajes, es posible construir aplicaciones que coadyuven a la solución de problemas concomitantes en ciencia e ingeniería.

Esta metodología permite utilizar todas las capacidades del cómputo en paralelo (grids de decenas de clusters, cada uno con miles de procesadores interconectados por red con un creciente poder de cómputo medible en Peta Flops), así como el uso de una amplia memoria (ya sea distribuida y/o compartida del orden de Tera Bytes), permitiendo atacar una gran variedad de problemas que sin estas técnicas es imposible hacerlo de manera flexible y eficiente.

Pero hay que notar que existe una amplia gama de problemas que nos interesan resolver que superan las capacidades de cómputo actuales, ya sea por el tiempo requerido para su solución, por el consumo excesivo de memoria o ambos.

La lista de los métodos de descomposición de dominio y el tipo de problemas que pueden ser atacados por estos, es grande y está en constante evolución [3], ya que se trata de encontrar un equilibrio entre la complejidad del método (aunada a la propia complejidad del modelo), la eficiencia en el consumo de los recursos computacionales y la precisión esperada en la solución encontrada por los diversos métodos y las arquitecturas paralelas en la que se implante.

1.2 Objetivos del Trabajo

Uno de los grandes retos del área de cómputo científico es poder analizar a priori una serie de consideraciones dictadas por factores externos al problema de interés que repercuten directamente en la forma de solucionar el problema, estas consideraciones influirán de manera decisiva en la implementación computacional de la solución numérica. Algunas de estas consideraciones son:

- Número de Procesadores Disponibles
- Tamaño y Tipo de Partición del Dominio
- Tiempo de Ejecución Predeterminado

Siendo común que ellas interactúan entre sí, de forma tal que normalmente el encargado de la implementación computacional de la solución numérica tiene además de las complicaciones técnicas propias de la solución, el conciliarlas con dichas consideraciones.

Esto deja al implementador de la solución numérica con pocos grados de libertad para hacer de la aplicación computacional una herramienta eficiente y flexible que cumpla con los lineamientos establecidos a priori y permita también que esta sea adaptable a futuros cambios de especificaciones -algo común en ciencia e ingeniería-.

Para tratar de tener una idea clara de cómo afectan a la aplicación computacional estos factores, a continuación detallamos algunas consideraciones acerca de cada uno de ellos.

Número de Procesadores Disponibles El número de procesadores disponibles es una barrera en constante evolución pero las principales limitantes son económicas y tecnológicas, es común ahora contar con miles de procesadores interactuando de manera conjunta en grids, pero todo ese poder de cómputo creciente es aún insuficiente para las necesidades del cómputo científico.

Por otro lado la gran mayoría de los proyectos científicos y tecnológicos no cuentan con recursos computacionales grandes, ya que este es un recurso costoso y por ende muy limitado y en la mayoría de los casos no es el adecuado a las necesidades propias del problema. Así, para poder atacar el problema de forma eficiente es necesario adaptarse al equipo de cómputo disponible y tratar de conocer de antemano los factores que mermaran el rendimiento de la implementación computacional para buscar alternativas que mejoren la eficiencia de la implementación.

Tamaño y Tipo de Partición del Dominio Normalmente cuando se plantea un problema de sistemas continuos la determinación del dominio y tipo de malla a usar en la solución del mismo está sujeta a restricciones fenomenológicas, por ello la malla deberá de adaptarse de la mejor forma posible para tratar de capturar los rasgos esenciales del fenómeno estudiado. Pero a

la hora de la implementación computacional es común hacer adecuaciones a esta, para que pueda ser soportada por el equipo de cómputo y su ejecución sea en un tiempo razonable. Esto puede ser un problema sobre todo al ser implementada la solución usando cómputo paralelo, ya que comúnmente una elección de una malla no homogénea ocasionará problemas de mal balance de carga de trabajo entre los procesadores utilizados en la implementación computacional.

Tiempo de Ejecución Predeterminado Otro factor determinante es el tiempo de solución esperado de la implementación computacional del problema, esto es crítico en problemas de control en tiempo real, comunes en la ciencia e ingeniería actual. Por ello en estos casos es permisible el aumento en el número y capacidades del equipo de cómputo necesario en la implementación con el fin de lograr un tiempo de ejecución por debajo del máximo permisible dictado por las especificaciones propias del problema.

Todos estos factores influirán en la versión final implementada en la solución computacional de un problema particular, debiendo ser todas ellas sopesadas antes de tomar una decisión en cuanto a las especificaciones que el programa de cómputo deberá satisfacer.

Así, el objetivo de este trabajo es desarrollar una metodología en la cual se implemente de forma paralela el método numérico de descomposición de dominio de subestructuración precondicionado, explicando detalladamente los fundamentos matemáticos de la modelación de fenómenos de sistemas continuos por medio de ecuaciones diferenciales parciales, los fundamentos del método de descomposición de dominio al aplicarse a ecuaciones diferenciales parciales elípticas y las ventajas sobre otros métodos de solución numérica.

Mostraremos cómo construir el modelo computacional, esto no sólo nos ayudará a demostrar que es factible la construcción del propio modelo computacional a partir del modelo matemático y numérico para la solución de problemas reales. Además, mostrará los alcances y limitaciones en el consumo de recursos computacionales y nos permitirá la evaluación de algunas de las variantes de los métodos numéricos con los que es posible implementar el modelo computacional y haremos algo de análisis de rendimiento sin llegar a ser exhaustivo esté.

También se muestra el diseño, análisis y programación de la aplicación computacional del método de descomposición de dominio en forma serial como paralela, basada en el paradigma de programación orientado a objetos

y cómo este paradigma ofrece una forma robusta y flexible para la representación de entidades abstractas y que permiten una modelación más versátil así como una forma más eficiente para la organización y reutilización de código.

1.2.1 Objetivos Generales

- Mostrar las bases de una metodología que se utiliza para aplicar el cómputo en paralelo a la modelación matemática y computacional de sistemas continuos, de forma flexible, escalable y eficiente.
- Mostrar los alcances y limitaciones de la metodología usando como herramientas de evaluación a los métodos de elemento finito secuencial, método de subestructuración secuencial y método de subestructuración paralelo.
- Mostrar los diversos esquemas de optimización de los recursos computacionales aplicables a un problema específico.

Para facilitar la comprensión de las ideas básicas, se ha tomado la ecuación de Poisson. Es un ejemplo sencillo, pero gobierna los modelos de muchos sistemas de la ingeniería y de la ciencia, entre ellos el flujo de agua subterránea a través de un acuífero isotrópico, homogéneo bajo condiciones de equilibrio y es muy usada en múltiples ramas de la física. Por ejemplo, gobierna la ecuación de la conducción de calor en un sólido bajo condiciones de equilibrio.

1.2.2 Objetivos Particulares

Como objetivos particulares de este trabajo tenemos:

- Mostrar cómo aplicar la metodología para manejar problemas de gran tamaño (descomposición de malla fina).
- Mostrar cómo descomponer un dominio en un conjunto de subdominios que den una partición en la que el tiempo de cálculo sea mínimo para una configuración de Hardware dada.
- Mostrar cuales son las posibles optimizaciones aplicables a una configuración de Hardware dada.

- Mostrar que es posible trabajar problemas con una malla muy fina en un equipo paralelo pequeño.

Para poder cumplir con los objetivos, primeramente empezaremos describiendo las bases de los sistemas continuos y sus modelos, para conocer algunas propiedades de las ecuaciones diferenciales parciales que son generadas en la elaboración del modelo matemático. Después veremos la forma de pasar del modelo matemático al modelo numérico (el cual no es muy cercano al modelo computacional) introduciendo los fundamentos del método Galerkin y la derivación a partir de este del método de elemento finito, para posteriormente describir dos métodos de descomposición de dominio (Schwarz y subestructuración).

Mostraremos cómo construir el modelo computacional que dependerá fuertemente de la arquitectura de cómputo disponible y del lenguaje de programación usado. Esto nos ayudará a demostrar que es factible la construcción del propio modelo computacional a partir del modelo matemático y numérico para la solución de problemas reales.

Además conoceremos los alcances y limitaciones en el consumo de recursos computacionales y nos permitirá la evaluación de algunas de las variantes de los métodos numéricos con los que es posible implementar el modelo computacional y haremos algo de análisis de rendimiento sin llegar a ser exhaustivo este.

Finalmente se exploran los alcances y limitaciones de cada uno de los métodos implementados (elemento finito secuencial, descomposición de dominio secuencial y paralelo) y se muestra cómo es posible optimizar los recursos computacionales con que se cuente.

1.3 Infraestructura Usada

El modelo computacional generado, está contenido en un programa de cómputo bajo el paradigma de orientación a objetos, programado en el lenguaje C++ en su forma secuencial y en su forma paralela en C++ en conjunto con la interfaz de paso de mensajes (MPI) bajo el esquema de maestro-esclavo. Una versión de estos programas está disponible en la página Web Subestructuración:

Herramientas
http://132.248.181.216/Herramientas/

Para desarrollar estos códigos, se realizó una jerarquía de clases para cada uno de los distintos componentes del sistema de elemento finito como de descomposición de dominio (rehusando todo el código de elemento finito en este), permitiendo usarlos tanto en forma secuencial como paralela.

Las pruebas de rendimiento de los distintos programas se realizaron en equipos secuenciales y paralelos (clusters) que están montados en el Instituto de Geofísica de la UNAM, en las pruebas de análisis de rendimiento se usaron para la parte secuencial el equipo:

- Computadora Pentium IV HT a 2.8 GHtz con 1 GB de RAM corriendo bajo el sistema operativo Linux Debian Stable con el compilador g++ de GNU.

Para la parte paralela se usaron los siguientes equipos:

- Cluster heterogéneo con el nodo maestro Pentium IV HT a 3.4 GHtz con 1 GB de RAM y 7 nodos esclavos Pentium IV HT a 2.8 GHtz con 0.5 GB de RAM por nodo, unidos mediante una red E-thernet de 100 Mb, corriendo bajo el sistema operativo Linux Debian Stable con el compilador mpiCC de MPI de GNU. A cargo del Dr. Ismael Herrera Revilla del departamento de Recursos Naturales.

Hay que notar que, el paradigma de programación orientada a objetos sacrifica algo de eficiencia computacional por requerir mayor manejo de recursos computacionales al momento de la ejecución. Pero en contraste, permite mayor flexibilidad a la hora adaptar los códigos a nuevas especificaciones. Adicionalmente, disminuye notoriamente el tiempo invertido en el mantenimiento y búsqueda de errores dentro del código. Esto tiene especial interés cuando se piensa en la cantidad de meses invertidos en la programación comparado con los segundos consumidos en la ejecución del mismo.

1.4 Agradecimientos

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

En la realización de este texto se han revisado -en la mayoría de los casos indicamos la referencia, pero pudimos omitir varias de ellas, por lo cual

pedimos una disculpa- múltiples páginas Web, artículos técnicos, libros, entre otros materiales bibliográficos, los más representativos y de libre acceso los ponemos a su disposición en la siguiente liga:

Herramientas
http://132.248.181.216/Herramientas/

Agradezco la revisión y aportaciones del Dr. Robert Yates (Alternativas en Computación, S.A de C.V), del Dr. Ismael Herrera Revilla (Instituto de Geofísica, UNAM) y del Dr. Martín Alberto Díaz Viera (Instituto Mexicano del Petróleo, IMP).

2 Métodos de Descomposición de Dominio

La solución numérica por los esquemas tradicionales de discretización tipo elemento finito y diferencias finitas generan una discretización del problema, la cual es usada para generar un sistema de ecuaciones algebraicos $\underline{\underline{A}}\underline{u} = \underline{b}$. Este sistema algebraico en general es de gran tamaño para problemas reales, al ser estos algoritmos secuenciales su implantación suele hacerse en equipos secuenciales y por ello no es posible resolver muchos problemas que involucren el uso de una gran cantidad de memoria, actualmente para tratar de subsanar dicha limitante, se usa equipo paralelo para soportar algoritmos secuenciales, haciendo ineficiente su implantación en dichos equipos.

Los métodos de descomposición de dominio son un paradigma natural usado por la comunidad de modeladores. Los sistemas físicos son descompuestos en dos o más subdominios contiguos basados en consideraciones fenomenológicas. Estas descomposiciones basadas en dominios físicos son reflejadas en la ingeniería de Software del código correspondiente.

Los métodos de descomposición de dominio permiten tratar los problemas de tamaño considerable, empleando algoritmos paralelos en computadoras secuenciales y/o paralelas. Esto es posible ya que cualquier método de descomposición de dominio se basa en la suposición de que dado un dominio computacional Ω, este se puede particionar en subdominios $\Omega_i, i = 1, 2, ..., E$ entre los cuales puede o no existir traslape. Entonces el problema es reformulado en términos de cada subdominio (empleando algún método del tipo elemento finito) obteniendo una familia de subproblemas de tamaño reducido independientes en principio entre sí, que están acoplados a través de la solución en la interfaz de los subdominios que es desconocida.

De esta manera, podemos clasificar de manera burda a los métodos de descomposición de dominio, como aquellos en que: existe traslape entre los subdominios y en los que no existe traslape. A la primera clase pertenece el método de Schwarz (en el cual el tamaño del traslape es importante en la convergencia del método) y a los de la segunda clase pertenecen los métodos del tipo subestructuración (en el cual los subdominios sólo tienen en común los nodos de la frontera interior).

La computación en paralelo es una técnica que nos permite distribuir una gran carga computacional entre muchos procesadores. Y es bien sabido que una de las mayores dificultades del procesamiento en paralelo es la coordinación de las actividades de los diferentes procesadores y el intercambio de información entre los mismos [14] mediante el paso de mensajes.

Así, mediante los métodos de descomposición de dominio, la programación orientada a objetos y esquemas de paralelización que usan el paso de mensajes, es posible construir aplicaciones que coadyuven a la solución de problemas concomitantes en ciencia e ingeniería, ya que permiten utilizar todas las capacidades del cómputo en paralelo (supercomputadoras, Clusters o Grids), de esta forma es posible atacar una gran variedad de problemas que sin estas técnicas es imposible hacerlo de manera flexible y eficiente.

Pero hay que notar que existen una amplia gama de problemas que nos interesan resolver que superan las capacidades de cómputo actuales, ya sea por el tiempo requerido para su solución, por el consumo excesivo de memoria o ambos.

La lista de los métodos de descomposición de dominio y el tipo de problemas que pueden ser atacados por estos, es grande y está en constante evolución, ya que se trata de encontrar un equilibrio entre la complejidad del método (aunada a la propia complejidad del modelo), la eficiencia en el consumo de los recursos computacionales y la precisión esperada en la solución encontrada por los diversos métodos y las arquitecturas paralelas en la que se implante.

A continuación describiremos algunos de estos métodos generales. En este capítulo se considerarán problemas con valor en la frontera (VBVP) de la forma

$$\begin{aligned} \mathcal{L}u &= f \quad \text{en } \Omega \\ u &= g \quad \text{en } \partial\Omega \end{aligned} \tag{2.1}$$

donde

$$\mathcal{L}u = -\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu \tag{2.2}$$

como un caso particular del operador elíptico definido por la Ec. (7.43) de orden dos.

2.1 Método de Schwarz

El método fue desarrollado por Hermann Amandus Schwarz en 1869 (no como un método de descomposición de dominio), ya que en esos tiempos los matemáticos podían resolver problemas con geometrías sencillas de manera analítica, pero no tenían una idea clara de cómo poder resolver problemas que involucraran el traslape de esas geometrías sencillas. Como se conocía la solución para las geometrías sencillas por separado, la idea de Schwarz

fue usar estas para conocer la solución en la geometría resultante al tener traslape, para más detalle ver [1].

Para describir el método, consideremos primero un dominio Ω que está formado de dos subdominios Ω_1 y Ω_2 traslapados, es decir $\Omega_1 \cap \Omega_2 \neq \varnothing$, entonces $\Omega = \Omega_1 \cup \Omega_2$ y denotemos a $\Sigma_1 = \partial\Omega_1 \cap \Omega_2$, $\Sigma_2 = \partial\Omega_2 \cap \Omega_1$ y $\Omega_{1,2} = \Omega_1 \cap \Omega_2$, como se muestra en la figura para dos dominios distintos:

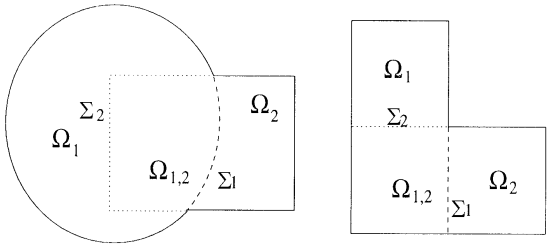


Figura 1: Dominio Ω subdividido en dos subdominios Ω_1 y Ω_2.

La forma original del método iterativo de Schwarz conocido como métodos alternantes de Schwarz, consiste en resolver sucesivamente los siguientes problemas.

Sea u^o una función de inicialización definida en Ω, que se nulifica en $\partial\Omega$, además hacemos $u_1^0 = u^0_{|\Omega_1}$ y $u_2^0 = u^0_{|\Omega_2}$. Para $k \geq 0$ definimos dos sucesiones u_1^{k+1} y u_2^{k+1} para resolver respectivamente

$$\begin{cases} \mathcal{L}u_1^{k+1} = f \quad \text{en } \Omega_1 \\ u_1^{k+1} = u_2^k \quad \text{en } \Sigma_1 \\ u_1^{k+1} = 0 \quad \text{en } \partial\Omega_1 \cap \partial\Omega \end{cases} \tag{2.3}$$

y

$$\begin{cases} \mathcal{L}u_2^{k+1} = f \quad \text{en } \Omega_2 \\ u_2^{k+1} = u_1^{k+1} \quad \text{en } \Sigma_2 \\ u_2^{k+1} = 0 \quad \text{en } \partial\Omega_2 \cap \partial\Omega \end{cases} \tag{2.4}$$

resolviendo los problemas secuencialmente en cada subdominio (por ejemplo con el método de elemento finito). Este método se conoce como Schwarz multiplicativo.

El método alternante de Schwarz dado por las Ecs. (2.3) y (2.4) converge a la solución u de (2.1) si suponemos alguna suavidad en los subdominios Ω_1

y Ω_2, ya que existen constantes C_1 y $C_2 \in (0,1)$ tal que para todo $k \geq 0$ se tiene

$$\begin{aligned} \left\|u_{|\Omega_1} - u_1^{k+1}\right\|_{L^\infty(\Omega_1)} &\leq C_1^k C_2^k \left\|u - u^0\right\|_{L^\infty(\Sigma_1)} \\ \left\|u_{|\Omega_2} - u_2^{k+1}\right\|_{L^\infty(\Omega_2)} &\leq C_1^{k+1} C_2^k \left\|u - u^0\right\|_{L^\infty(\Sigma_2)} \end{aligned} \tag{2.5}$$

las constantes C_1 y C_2 de reducción de error deben de estar bastante cerca de 1 si la región de traslape $\Omega_{1,2}$ es delgada, la prueba de esta estimación puede encontrarse en [45].

Por otro lado, teniendo el conjunto $u_1^0 = u^0_{|\Omega_1}$ y $u_2^0 = u^0_{|\Omega_2}$, podemos generar dos pasos independientes uno de otro

$$\begin{cases} \mathcal{L}u_1^{k+1} = f \quad \text{en } \Omega_1 \\ u_1^{k+1} = u_2^k \quad \text{en } \Sigma_1 \\ u_1^{k+1} = 0 \quad \text{en } \partial\Omega_1 \cap \partial\Omega \end{cases} \tag{2.6}$$

y

$$\begin{cases} \mathcal{L}u_2^{k+1} = f \quad \text{en } \Omega_2 \\ u_2^{k+1} = u_1^k \quad \text{en } \Sigma_2 \\ u_2^{k+1} = 0 \quad \text{en } \partial\Omega_2 \cap \partial\Omega \end{cases} \tag{2.7}$$

resolviendo los problemas en paralelo de cada subdominio (por ejemplo con el método de elemento finito). Este método se conoce como Schwarz aditivo.

La convergencia de este método en general requiere de algunas hipótesis adicionales, pero si converge, el número de iteraciones necesarias para converger será del doble que el método Schwarz multiplicativo.

La generalización del método de Schwarz en el caso en que Ω es particionada en $E > 2$ subdominios traslapados puede describirse como sigue:

Descomponiendo el dominio Ω en E subdominios Ω_e con traslape como por ejemplo, la descomposición siguiente

Entonces para resolver el problema por el método de Schwarz, primeramente es necesario definir los siguientes subespacios del espacio de Sobolev $H^1(\Omega)$, en ellos estarán definidas las funciones usadas en el desarrollo del método:

$$\begin{aligned} V_i &= \left\{v_i \in H^1(\Omega_i) \mid v_{|\partial\Omega\cap\partial\Omega_i} = 0\right\} \\ V_i^0 &= H_0^1(\Omega_i) \\ V_i^* &= \left\{v \in H_0^1(\Omega) \mid v = 0 \text{ en } \Omega\backslash\bar{\Omega}_i\right\}. \end{aligned}$$

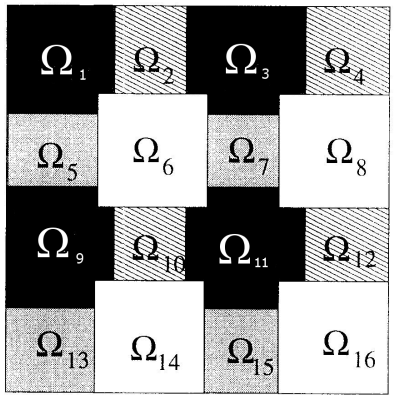


Figura 2: Descomposición de Ω en múltiples subdominios con traslape para el método de Schwarz.

Denotaremos por I al operador identidad, por $J_i, i = 1, ...E$ la inmersión de V_i^* sobre V (i.e. $J_i v = v$ para toda $v \in V_i^*$), y por $J_i^T : H_0^1(\Omega_i) \to V_i^*$ al transpuesto del operador J_i definido por

$$\left\langle J_i^T F, v \right\rangle = \left\langle F, J_i v \right\rangle, \quad \forall F \in V^{'}, v \in V_i^*. \tag{2.8}$$

y definimos

$$P_i = J_i P_i^* : H_0^1(\Omega_i) \to H_0^1(\Omega_i).$$

Sea $\mathcal{L}_i : V_i^0 \to \left(V_i^0\right)^{'}$ la restricción del operador $\mathcal{L}$ al subespacio V_i^0, definido como

$$\langle \mathcal{L}_i w_i, v_i \rangle = a\left(w_i, v_i\right) \text{ para toda } w_i, v_i \in V_i^0$$

y como un operador de extensión $\rho_i^T : V_i^0 \to V_i^*$ definido como

$$\rho_i^T v_i = \tilde{v}_i \text{ para toda } v_i \in V_i^0 \tag{2.9}$$

y el transpuesto del operador restricción $\rho_i : (V_i^*)' \to (V_i^0)'$ como

$$\langle \rho_i G, v_i \rangle = \left\langle G, \rho_i^T v_i \right\rangle, \text{para toda } G \in (V_i^*)', v_i \in V_i^0. \tag{2.10}$$

De lo anterior se tiene que

$$\mathcal{L}_i = \rho_i J_i^T \mathcal{L} J_i \rho_i^T$$

y

$$P_i = J_i P_i^* : V \to V \text{ para } i = 1, ..., E. \tag{2.11}$$

Entonces el método Multiplicativo de Schwarz queda como

$$u^{k+\frac{i}{E}} = (I - P_i)u^{k+\frac{i-1}{E}} + J_i \rho_i^T \mathcal{L}_i^{-1} \rho_i J_i^T f \tag{2.12}$$

para $i = 1, 2, ..., E$ y

$$u^{k+1} = \left(I - \sum_{i=1}^{E} P_i \right) u^k + \sum_{i=1}^{M} J_i \rho_i^T \mathcal{L}_i^{-1} \rho_i J_i^T f \tag{2.13}$$

y la correspondiente ecuación de error como

$$u - u^{k+1} = (I - P_m) \dots (I - P_1)(u - u^k). \tag{2.14}$$

El el método Aditivo de Schwarz queda como

$$u - u^{k+1} = \left(I - \sum_{i=1}^{E} P_i \right) \left(u - u^k \right) \tag{2.15}$$

y la correspondiente ecuación de error como

$$u - u^{k+1} = (I - P_m) \dots (I - P_1)(u - u^k). \tag{2.16}$$

Observaciones:

- La precisión del método depende fuertemente del número de iteraciones realizadas en el proceso iterativo y converge a la precisión usada en la solución de cada subdominio en el mejor de los casos.
- El método aditivo de Schwarz es secuencial, en el caso del método multiplicativo de Schwarz es paralelizable pero tiene una parte serial importante en el algoritmo y su convergencia no es la óptima en esta formulación, pero existen variantes del método que permiten remediar esto, para más detalles ver [3], [5] y [6].
- Hay que notar que por cada subdominio (supóngase n) y en cada iteración (supóngase I) se resuelve un problema para cada Ω_i, esto significa que si se usa el método de elemento finito para resolver el problema local donde se usen en promedio r iteraciones para

resolver el sistema lineal (no tomando en cuenta el costo invertido en generar las matrices), el total de iteraciones necesarias para resolver el problema en el dominio Ω será $r * n * I$, resultando muy costoso computacionalmente con respecto a otros métodos de descomposición de dominio.

2.2 Método de Subestructuración

La solución numérica por los esquemas tradicionales de discretización del tipo Elemento Finito y Diferencias Finitas generan una discretización del problema, la cual es usada para generar un sistema de ecuaciones algebraicos $\underline{\underline{A}}\underline{u} = \underline{b}$. Este sistema algebraico en general es de gran tamaño para problemas reales, al ser estos algoritmos secuenciales su implantación suele hacerse en equipos secuenciales y por ello no es posible resolver muchos problemas que involucren el uso de una gran cantidad de memoria, actualmente para tratar de subsanar dicha limitante, se usa equipo paralelo para soportar algoritmos secuenciales, haciendo ineficiente su implantación en dichos equipos.

Los métodos de descomposición de dominio son un paradigma natural usado por la comunidad de modeladores. Los sistemas físicos son descompuestos en dos o más subdominios contiguos basados en consideraciones fenomenológicas. Estas descomposiciones basadas en dominios físicos son reflejadas en la ingeniería de Software del código correspondiente.

Los métodos de descomposición de dominio permiten tratar los problemas de tamaño considerable, empleando algoritmos paralelos en computadoras secuenciales y/o paralelas. Esto es posible ya que cualquier método de descomposición de dominio se basa en la suposición de que dado un dominio computacional Ω, este se puede particionar -triangular- en E subdominios $\Omega_\alpha, \alpha = 1, 2, ..., E$ entre los cuales no existe traslape. Entonces el problema es reformulado en términos de cada subdominio (empleando por ejemplo algún método tipo elemento finito) obteniendo una familia de subproblemas de tamaño reducido independientes en principio entre sí, que están acoplados a través de la solución en la interfaz de los subdominios que es desconocida [2], [1], [6] y [9], uno de los primeros Métodos de Descomposición de Dominio sin traslape es el de Subestructuración -mejor conocido como Schur- y el cual es la base los métodos más comúnmente usados como son FETI-DP, BDDC y el propio DVS-DDM.

En esta sección y sin pérdida de generalidad se considerarán problemas

con valor en la frontera (VBVP) de la forma

$$\begin{aligned} \mathcal{L}u &= f \quad \text{en } \Omega \\ u &= g \quad \text{en } \partial\Omega \end{aligned} \tag{2.17}$$

donde

$$\mathcal{L}u = -\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu \tag{2.18}$$

con $\underline{\underline{a}}$ una matriz positiva definida, simétrica y $c \geq 0$, como un caso particular del operador elíptico de orden 2 y para ejemplificar tomaremos un dominio $\Omega \subset R^2$ con fronteras poligonales, es decir, Ω es un conjunto abierto acotado y conexo tal que su frontera $\partial\Omega$ es la unión de un número finito de polígonos.

Si multiplicamos a la ecuación $-\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu = f_\Omega$ por $v \in V = H_0^1(\Omega)$, obtenemos

$$-v\left(\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu\right) = v f_\Omega \tag{2.19}$$

aplicando el teorema de Green obtenemos

$$\int_\Omega \left(\nabla v \cdot \underline{\underline{a}} \cdot \nabla u + cuv\right) d\underline{x} = \int_\Omega v f_\Omega d\underline{x}. \tag{2.20}$$

Definiendo el operador bilineal

$$a(u,v) = \int_\Omega \left(\nabla v \cdot \underline{\underline{a}} \cdot \nabla u + cuv\right) d\underline{x} \tag{2.21}$$

y la funcional lineal

$$l(v) = \langle f, v\rangle = \int_\Omega v f_\Omega d\underline{x} \tag{2.22}$$

entonces podemos reescribir el problema en forma variacional, haciendo uso de la forma bilineal $a(\cdot,\cdot)$ y la funcional lineal $l(\cdot)$.

Donde las funciones lineales definidas por pedazos en Ω_e en nuestro caso serán polinomios de orden uno en cada variable separadamente y cuya restricción de ϕ_i a Ω_e es $\phi_i^{(e)}$. Para simplificar los cálculos en esta etapa, supondremos que la matriz $\underline{\underline{a}} = a\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, entonces se tiene que la integral del lado izquierdo de la Ec. (2.20) queda escrita como

$$\int_\Omega \left(a\nabla\phi_i \cdot \nabla\phi_j + c\phi_i\phi_j\right) dxdy = \int_\Omega f_\Omega \phi_j dxdy \tag{2.23}$$

donde

$$\begin{aligned} K_{ij} &= \int_{\Omega} \left(a\nabla\phi_i \cdot \nabla\phi_j + c\phi_i\phi_j \right) dxdy \\ &= \sum_{e=1}^{E} \int_{\Omega_e} \left(a\nabla\phi_i^{(e)} \cdot \nabla\phi_j^{(e)} + c\phi_i^{(e)}\phi_j^{(e)} \right) dxdy \\ &= \sum_{e=1}^{E} \int_{\Omega_e} \left(a \left[\frac{\partial\phi_i^{(e)}}{\partial x}\frac{\partial\phi_j^{(e)}}{\partial x} + \frac{\partial\phi_i^{(e)}}{\partial y}\frac{\partial\phi_j^{(e)}}{\partial y} \right] + c\phi_i^{(e)}\phi_j^{(e)} \right) dxdy \end{aligned} \tag{2.24}$$

y el lado derecho como

$$\begin{aligned} F_j &= \int_{\Omega} f_{\Omega}\phi_j dxdy \\ &= \sum_{e=1}^{E} \int_{\Omega_e} f_{\Omega}\phi_j^{(e)} dxdy. \end{aligned} \tag{2.25}$$

Consideremos el problema dado por la Ec. (2.17) en el dominio Ω, el cual es subdividido en E subdominios Ω_i, $i = 1, 2, ..., E$ sin traslape, también conocida como malla gruesa $\mathcal{T}_H$, es decir

$$\Omega_i \cap \Omega_j = \varnothing \quad \forall i \neq j \quad \text{y} \quad \overline{\Omega} = \bigcup_{i=1}^{E} \overline{\Omega}_i, \tag{2.26}$$

y al conjunto

$$\Gamma = \bigcup_{i=1}^{E} \Gamma_i, \quad \text{si } \Gamma_i = \partial\Omega_i \backslash \partial\Omega \tag{2.27}$$

lo llamaremos la frontera interior del dominio Ω, denotamos por H al diámetro $H_i = Diam(\Omega_i)$ de cada Ω_i que satisface $Diam(\Omega_i) \leq H$ para cada $i = 1, 2, ..., E$, además, cada subdominio Ω_i es descompuesto en una mallado fino $\mathcal{T}_h$ de K subdominios mediante una triangulación Ω_e de modo que esta sea conforme, denotamos por h al diámetro $h_i = Diam(\Omega_e)$ de cada Ω_e que satisface $Diam(\Omega_e) \leq h$ para cada $e = 1, 2, ..., K$ de cada $i = 1, 2, ..., E$.

Un ejemplo de un dominio Ω y su descomposición en subdominios Ω_i y cada Ω_i a su vez descompuesto en Ω_e subdominios se muestra en la figura:

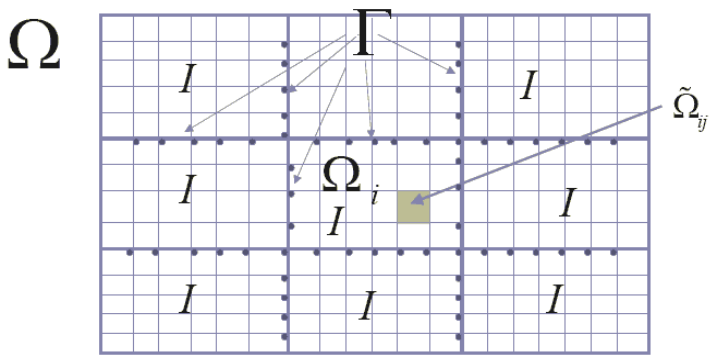


Figura 3: Dominio Ω descompuesto en una partición gruesa de 3×3 y cada subdominio Ω_i en una partición fina de 7×5.

Sin pérdida de generalidad tomemos $g = 0$ en $\partial\Omega$, notemos que siempre es posible poner el problema de la Ec. (2.17) como uno con condiciones de frontera Dirichlet que se nulifiquen mediante la adecuada manipulación del término del lado derecho de la ecuación.

Primeramente sea $D \subset H_0^1(\Omega)$ un espacio lineal de funciones de dimensión finita N, en el cual esté definido un producto interior denotado para cada $u, v \in D$ por

$$u \cdot v = \langle u, v \rangle \tag{2.28}$$

Considerando la existencia de los subconjuntos linealmente independientes

$$\begin{gathered} \mathcal{B} \subset \tilde{D}, \mathcal{B}_I \subset \tilde{D}_I, \mathcal{B}_\Gamma \subset \tilde{D}_\Gamma \\ \mathcal{B}_\Gamma \subset \tilde{D}_\Gamma, \mathcal{B}_{\Gamma J} \subset \tilde{D}_{\Gamma 1}, \mathcal{B}_{\Gamma M} \subset \tilde{D}_{\Gamma 2} \end{gathered} \tag{2.29}$$

los cuales satisfacen

$$\mathcal{B} = \mathcal{B}_I \cup \mathcal{B}_\Gamma \text{ y } \bar{\mathcal{B}}_\Gamma = \mathcal{B}_{\Gamma J} \cup \mathcal{B}_{\Gamma M} \tag{2.30}$$

el espacio generado por cada uno de los subconjuntos $\mathcal{B}_\Gamma$ y $\bar{\mathcal{B}}_\Gamma$ es $\tilde{D}_\Gamma$, sin embargo distinguimos la propiedad de que los miembros de $\mathcal{B}_\Gamma$ tienen soporte local.

Definimos las bases

$$\mathcal{B}_I = \left\{ w_I^1, ..., w_I^{\bar{N}_I} \right\}, \mathcal{B}_{\Gamma M} = \left\{ w_M^1, ..., w_M^{\bar{N}_\Gamma} \right\} \text{ y } \mathcal{B}_{\Gamma J} = \left\{ w_J^1, ..., w_J^{\bar{N}_\Gamma} \right\} \tag{2.31}$$

de las funcionales lineales ϕ_i en Ω.

Entonces definiendo para toda $\delta = 1, ..., K$, la matriz de $N_\delta \times N_\delta$

$$\underline{\underline{A}}^{\delta}_{II} \equiv \left[\left\langle w^i_I, w^j_I \right\rangle\right] \tag{2.32}$$

que sólo está definida en cada subespacio (subdominio Ω_δ). Entonces, la matriz virtual $\underline{\underline{A}}_{II}$ es dada por la matriz diagonal de la forma

$$\underline{\underline{A}}_{II} \equiv \begin{bmatrix} \underline{\underline{A}}^1_{II} & & & \\ & \underline{\underline{A}}^2_{II} & & \\ & & \ddots & \\ & & & \underline{\underline{A}}^E_{II} \end{bmatrix} \tag{2.33}$$

donde el resto de la matriz fuera de la diagonal en bloques es cero.

De forma similar definimos

$$\underline{\underline{A}}^{\delta}_{I\Gamma} \equiv \left[\left\langle w^i_I, w^\alpha_\Gamma \right\rangle\right], \quad \underline{\underline{A}}^{\delta}_{\Gamma I} \equiv \left[\left\langle w^\alpha_\Gamma, w^i_I \right\rangle\right] \tag{2.34}$$

y

$$\underline{\underline{A}}^{\delta}_{\Gamma\Gamma} \equiv \left[\langle w^\alpha_\Gamma, w^\alpha_\Gamma \rangle\right] \tag{2.35}$$

para toda $\delta = 1, ..., E$, obsérvese que como $\bar{\mathcal{B}}_\Gamma = \mathcal{B}_{\Gamma J} \cup \mathcal{B}_{\Gamma M}$ entonces

$$\underline{\underline{A}}^{\delta}_{\Gamma\Gamma} = \left[\langle w^\alpha_\Gamma, w^\alpha_\Gamma \rangle\right] = \left[\left\langle w^\alpha_{\Gamma_J}, w^\alpha_{\Gamma_J} \right\rangle\right] + \left[\left\langle w^\alpha_{\Gamma_M}, w^\alpha_{\Gamma_M} \right\rangle\right] \tag{2.36}$$

también que $\underline{\underline{A}}^{\delta}_{I\Gamma} = \left(\underline{\underline{A}}^{\delta}_{\Gamma I}\right)^T$. Entonces las matrices virtuales $\underline{\underline{A}}_{\Gamma I}, \underline{\underline{A}}_{I\Gamma}$ y $\underline{\underline{A}}_{\Gamma\Gamma}$ quedarán definidas como

$$\underline{\underline{A}}_{I\Gamma} \equiv \begin{bmatrix} \underline{\underline{A}}^1_{I\Gamma} \\ \underline{\underline{A}}^2_{I\Gamma} \\ \vdots \\ \underline{\underline{A}}^E_{I\Gamma} \end{bmatrix} \tag{2.37}$$

$$\underline{\underline{A}}_{\Gamma I} \equiv \begin{bmatrix} \underline{\underline{A}}^1_{\Gamma I} & \underline{\underline{A}}^2_{\Gamma I} & \cdots & \underline{\underline{A}}^E_{\Gamma I} \end{bmatrix} \tag{2.38}$$

y

$$\underline{\underline{A}}_{\Gamma\Gamma} \equiv \left[\sum_{i=1}^{E} \underline{\underline{A}}^i_{\Gamma\Gamma}\right] \tag{2.39}$$

donde $\left[\sum_{i=1}^{E} \underline{\underline{A}}^i_{\Gamma\Gamma}\right]$ es construida sumando las $\underline{\underline{A}}^i_{\Gamma\Gamma}$ según el orden de los nodos globales versus los nodos locales.

También consideremos al vector $\underline{u} \equiv (u_1, .., u_E)$ el cual puede ser escrito como $\underline{u} = (\underline{u_I}, \underline{u_\Gamma})$ donde $\underline{u_I} = (u_1, ..., _{N_I})$ y $\underline{u_\Gamma} = (u_1, ..., _{N_\Gamma})$.

Así, el sistema virtual

$$\begin{aligned} \underline{\underline{A}}_{II}\underline{u_I} + \underline{\underline{A}}_{I\Gamma}\underline{u_\Gamma} &= \underline{b_I} \\ \underline{\underline{A}}_{\Gamma I}\underline{u_I} + \underline{\underline{A}}_{\Gamma\Gamma}\underline{u_\Gamma} &= \underline{b_\Gamma} \end{aligned} \tag{2.40}$$

quedando expresado como

$$\begin{bmatrix} \underline{\underline{A}}^1_{II} & & \\ & \ddots & \\ & & \underline{\underline{A}}^E_{II} \end{bmatrix} \begin{bmatrix} \underline{u_I}_1 \\ \vdots \\ \underline{u_I}_E \end{bmatrix} + \begin{bmatrix} \underline{\underline{A}}^1_{I\Gamma} \\ \vdots \\ \underline{\underline{A}}^E_{I\Gamma} \end{bmatrix} \begin{bmatrix} \underline{u_\Gamma}_1 \\ \vdots \\ \underline{u_\Gamma}_E \end{bmatrix} = \begin{bmatrix} \underline{b_I}_1 \\ \vdots \\ \underline{b_I}_E \end{bmatrix}$$

$$\begin{bmatrix} \underline{\underline{A}}^1_{\Gamma I} & \cdots & \underline{\underline{A}}^E_{\Gamma I} \end{bmatrix} \begin{bmatrix} \underline{u_I}_1 \\ \vdots \\ \underline{u_I}_E \end{bmatrix} + \begin{bmatrix} \underline{\underline{A}}^{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} \underline{u_\Gamma}_1 \\ \vdots \\ \underline{u_\Gamma}_E \end{bmatrix} = \begin{bmatrix} \underline{b_\Gamma}_1 \\ \vdots \\ \underline{b_\Gamma}_E \end{bmatrix}$$

o más compactamente como $\underline{\underline{A}}\underline{u} = \underline{b}$, notemos que las matrices $\underline{\underline{A}}^i_{\Gamma\Gamma}, \underline{\underline{A}}^i_{\Gamma I}, \underline{\underline{A}}^i_{I\Gamma}$ y $\underline{\underline{A}}^i_{II}$ son matrices bandadas.

Si ahora despejamos $\underline{u_I}$ de la primera ecuación del sistema dado por la Ec. (2.40) obtenemos

$$\underline{u_I} = \left(\underline{\underline{A}}_{II}\right)^{-1}\left(\underline{b_I} - \underline{\underline{A}}_{I\Gamma}\underline{u_\Gamma}\right)$$

si sustituimos $\underline{u_I}$ en la segunda ecuación del sistema dado por la Ec. (2.40) entonces tenemos

$$\left(\underline{\underline{A}}_{\Gamma\Gamma} - \underline{\underline{A}}_{\Gamma I}\left(\underline{\underline{A}}_{II}\right)^{-1}\underline{\underline{A}}_{I\Gamma}\right)\underline{u_\Gamma} = \underline{b_\Gamma} - \underline{\underline{A}}_{\Gamma I}\left(\underline{\underline{A}}_{II}\right)^{-1}\underline{b_I} \tag{2.41}$$

en la cual los nodos interiores no figuran en la ecuación y todo queda en función de los nodos de la frontera interior $\underline{u_\Gamma}$.

A la matriz formada por $\underline{\underline{A}}_{\Gamma\Gamma} - \underline{\underline{A}}_{\Gamma I}\left(\underline{\underline{A}}_{II}\right)^{-1}\underline{\underline{A}}_{I\Gamma}$ se le conoce como el complemento de Schur global y se le denota como

$$\underline{\underline{S}} = \underline{\underline{A}}_{\Gamma\Gamma} - \underline{\underline{A}}_{\Gamma I}\left(\underline{\underline{A}}_{II}\right)^{-1}\underline{\underline{A}}_{I\Gamma}. \tag{2.42}$$

En nuestro caso, como estamos planteando todo en términos de subdominios Ω_i, con $i = 1, ..., E$, entonces las matrices $\underline{\underline{A}}^i_{\Gamma\Gamma}, \underline{\underline{A}}^i_{\Gamma I}, \underline{\underline{A}}^i_{I\Gamma}$ y $\underline{\underline{A}}^i_{II}$ quedan definidas de manera local, así que procedemos a definir el complemento de Schur local como

$$\underline{\underline{S}}_i = \underline{\underline{A}}^i_{\Gamma\Gamma} - \underline{\underline{A}}^i_{\Gamma I} \left(\underline{\underline{A}}^i_{II}\right)^{-1} \underline{\underline{A}}^i_{I\Gamma} \tag{2.43}$$

adicionalmente definimos

$$\underline{b}_i = \underline{b}_{\Gamma_i} - \underline{\underline{A}}^i_{\Gamma I} \left(\underline{\underline{A}}^i_{II}\right)^{-1} \underline{b}_{I_i}. \tag{2.44}$$

El sistema dado por la Ec. (2.41) lo escribimos como

$$\underline{\underline{S}}\,\underline{u_\Gamma} = \underline{b} \tag{2.45}$$

y queda definido de manera virtual a partir de

$$\left[\sum_{i=1}^{E} \underline{\underline{S}}_i\right] \underline{u_\Gamma} = \left[\sum_{i=1}^{E} \underline{b}_i\right] \tag{2.46}$$

donde $\left[\sum_{i=1}^{E} \underline{\underline{S}}_i\right]$ y $\left[\sum_{i=1}^{E} \underline{b}_i\right]$ podrían ser construida sumando las S_i y b_i respectivamente según el orden de los nodos globales versus los nodos locales.

El sistema lineal virtual obtenido de la Ec. (2.45) se resuelve -dependiendo de si es o no simétrico- eficientemente usando el método de Gradiente Conjugado o alguna variante de GMRES, para ello no es necesario construir la matriz $\underline{\underline{S}}$ con las contribuciones de cada S_i correspondientes al subdominio i, lo que hacemos es pasar a cada subdominio el vector $\underline{u_\Gamma}_i$ correspondiente a la i-ésima iteración del método iterativo usado para que en cada subdominio se evalué $\underline{\tilde{u}_\Gamma}^i = \underline{\underline{S}}_i \underline{u_\Gamma}_i$ localmente y con el resultado se forma el vector $\underline{\tilde{u}_\Gamma} = \sum_{i=1}^{E} \underline{\tilde{u}_\Gamma}_i$ y se continué con los demás pasos del método. Esto es ideal para una implementación en paralelo del método de Gradiente Conjugado o variante de GMRES.

Una vez resuelto el sistema de la Ec. (2.46) en el que hemos encontrado la solución para los nodos de la frontera interior $\underline{u_\Gamma}$, entonces debemos resolver localmente los $\underline{u_{I_i}}$ correspondientes a los nodos interiores para cada subespacio Ω_i, para esto empleamos

$$\underline{u_{I_i}} = \left(\underline{\underline{A}}^i_{II}\right)^{-1} \left(\underline{b}_{I_i} - \underline{\underline{A}}^i_{I\Gamma} \underline{u_\Gamma}_i\right) \tag{2.47}$$

para cada $i = 1, 2, ..., E$, quedando así resuelto el problema $\underline{\underline{A}}\underline{u} = \underline{b}$ tanto en los nodos interiores $\underline{u_I}_i$ como en los de la frontera interior $\underline{u_\Gamma}_i$ correspondientes a cada subespacio Ω_i.

Observación 1 *Notemos que normalmente las matrices locales* $\underline{\underline{S}}_i$ *y* $\left(\underline{\underline{A}}^i_{II}\right)^{-1}$ *no se construyen, ya que estas serian matrices densas y su construcción es computacionalmente muy costosa, y como sólo nos interesa el producto* $\underline{\underline{S}}\underline{y_\Gamma}$, *o más precisamente* $\left[\sum_{i=1}^{E} \underline{\underline{S}}_i\right] \underline{y_\Gamma}$, *entonces si llamamos* $\underline{y_\Gamma}_i$ *al vector correspondiente al subdominio* i, *entonces tendremos*

$$\underline{\tilde{u}_\Gamma}^i = \left(\underline{\underline{A}}^i_{\Gamma\Gamma} - \underline{\underline{A}}^i_{\Gamma I}\left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{\underline{A}}^i_{I\Gamma}\right)\underline{y_\Gamma}_i. \tag{2.48}$$

Para evaluar eficientemente esta expresión, realizamos las siguientes operaciones equivalentes

$$\begin{aligned} \underline{x1} &= \underline{\underline{A}}^i_{\Gamma\Gamma}\underline{y_\Gamma}_i \\ \underline{x2} &= \left(\underline{\underline{A}}^i_{\Gamma I}\left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{\underline{A}}^i_{I\Gamma}\right)\underline{y_\Gamma}_i \\ \underline{\tilde{u}_\Gamma}^i &= \underline{x1} - \underline{x2} \end{aligned} \tag{2.49}$$

la primera y tercera expresión no tienen ningún problema en su evaluación, para la segunda expresión tendremos que hacer

$$\underline{x3} = \underline{\underline{A}}^i_{I\Gamma}\underline{y_\Gamma}_i \tag{2.50}$$

con este resultado intermedio deberíamos calcular

$$\underline{x4} = \left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{x3} \tag{2.51}$$

pero como no contamos con $\left(\underline{\underline{A}}^i_{II}\right)^{-1}$, *entonces multiplicamos la expresión por* $\underline{\underline{A}}^i_{II}$ *obteniendo*

$$\underline{\underline{A}}^i_{II}\underline{x4} = \underline{\underline{A}}^i_{II}\left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{x3} \tag{2.52}$$

al simplificar, tenemos

$$\underline{\underline{A}}^i_{II}\underline{x4} = \underline{x3}. \tag{2.53}$$

Esta última expresión puede ser resuelta usando Factorización LU, Gradiente Conjugado o alguna variante de GMRES (cada una de estas opciones tiene ventajas y desventajas computacionales que deben ser evaluadas al momento de implementar el código para un problema particular). Una vez obtenido $\underline{x4}$*, podremos calcular*

$$\underline{x2} = \underline{\underline{A}}^i_{\Gamma I}\underline{x4} \tag{2.54}$$

así

$$\underline{\tilde{u}_\Gamma}^i = \underline{x1} - \underline{x2} \tag{2.55}$$

completando la secuencia de operaciones necesaria para obtener $\underline{\underline{S}}_i\underline{y_\Gamma}_i$.

Observación 2 *En el caso del cálculo de*

$$\underline{b}_i = \underline{b_\Gamma}_i - \underline{\underline{A}}^i_{\Gamma I}\left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{b_I}_i \tag{2.56}$$

algo análogo al comentario anterior deberá de hacerse, ya que nuevamente está involucrado $\left(\underline{\underline{A}}^i_{II}\right)^{-1}$*, por ello deberemos de usar el siguiente procedimiento para evaluar eficientemente esta expresión, realizando las operaciones equivalentes*

$$\underline{y1} = \left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{b_I}_i \tag{2.57}$$

multiplicando por $\underline{\underline{A}}^i_{II}$ *a la última expresión, obtenemos*

$$\underline{\underline{A}}^i_{II}\underline{y1} = \underline{\underline{A}}^i_{II}\left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{b_I}_i \tag{2.58}$$

simplificando, tenemos

$$\left(\underline{\underline{A}}^i_{II}\right)\underline{y1} = \underline{b_I}_i \tag{2.59}$$

donde esta última expresión puede ser resuelta usando Factorización LU, Gradiente Conjugado o alguna variante de GMRES, luego hacemos

$$\underline{y2} = \underline{\underline{A}}^i_{\Gamma I}\underline{y1} \tag{2.60}$$

y para finalizar el cálculo, calculamos

$$\underline{b}_i = \underline{b_\Gamma}_i - \underline{y2}. \tag{2.61}$$

Una vez resuelto el sistema de la Ec. (2.46) en el que hemos encontrado la solución para los nodos de la frontera interior $\underline{u_\Gamma}$, entonces debemos resolver localmente los $\underline{u_{I_i}}$ correspondientes a los nodos interiores para cada subespacio Ω_i, para esto empleamos

$$\underline{u_{I_i}} = \left(\underline{\underline{A}}_i^{II}\right)^{-1}\left(\underline{b_{I_i}} - \underline{\underline{A}}_i^{I\Gamma}\underline{u_{\Gamma_i}}\right) \tag{2.62}$$

para cada $i = 1, 2, ..., E$, quedando así resuelto el problema $\underline{\underline{A}}\underline{u} = \underline{b}$ tanto en los nodos interiores $\underline{u_{I_i}}$ como en los de la frontera interior $\underline{u_{\Gamma_i}}$ correspondientes a cada subespacio Ω_i.

Observación 3 *En la evaluación de*

$$\underline{u_{I_i}} = \left(\underline{\underline{A}}_{II}^{i}\right)^{-1}\left(\underline{b_{I_i}} - \underline{\underline{A}}_{I\Gamma}^{i}\underline{u_{\Gamma_i}}\right) \tag{2.63}$$

esta nuevamente involucrado $\left(\underline{\underline{A}}_{II}^{i}\right)^{-1}$, *por ello deberemos de usar el siguiente procedimiento para evaluar eficientemente esta expresión, realizando las operaciones equivalentes*

$$\begin{aligned} \underline{x4} &= \underline{b_{I_i}} - \underline{\underline{A}}_{I\Gamma}^{i}\underline{u_{\Gamma_i}} \\ \underline{u_{I_i}} &= \left(\underline{\underline{A}}_{II}^{i}\right)^{-1}\underline{x4} \end{aligned} \tag{2.64}$$

multiplicando por $\underline{\underline{A}}_{II}^{i}$ *a la última expresión, obtenemos*

$$\underline{\underline{A}}_{II}^{i}\underline{u_{I_i}} = \underline{\underline{A}}_{II}^{i}\left(\underline{\underline{A}}_{II}^{i}\right)^{-1}\underline{x4} \tag{2.65}$$

simplificando, tenemos

$$\underline{\underline{A}}_{II}^{i}\underline{u_{I_i}} = \underline{x4} \tag{2.66}$$

esta última expresión puede ser resuelta usando Factorización LU o Gradiente Conjugado.

Como se indico en las últimas observaciones, para resolver el sistema $\underline{\underline{A}}_{II}^{i}\underline{x} = \underline{b}$ podemos usar Factorización LU, Gradiente Conjugado, alguna variante de GMRES o cualquier otro método para resolver sistemas lineales, pero deberá de usarse aquel que proporcione la mayor velocidad en el cálculo o que consuma la menor cantidad de memoria (ambas condicionantes son

mutuamente excluyentes), por ello la decisión de que método usar deberá de tomarse al momento de tener que resolver un problema particular en un equipo dado y básicamente el condicionante es el tamaño de la matriz $\underline{\underline{A}}^i_{II}$.

Para usar el método de Factorización LU, se deberá primeramente de factorizar la matriz bandada $\underline{\underline{A}}^i_{II}$ en una matriz $\underline{\underline{LU}}$, la cual es bandada pero incrementa el tamaño de la banda a más del doble, pero esta operación sólo se deberá de realizar una vez por cada subdominio, y para solucionar los diversos sistemas lineales $\underline{\underline{A}}^i_{II}\underline{x} = \underline{b}$ sólo será necesario evaluar los sistemas

$$\begin{aligned} \underline{\underline{L}}\underline{y} &= \underline{b} \\ \underline{\underline{U}}\underline{x} &= \underline{y} \end{aligned} \tag{2.67}$$

en donde $\underline{y}$ es un vector auxiliar. Esto proporciona una manera muy eficiente de evaluar el sistema lineal pero el consumo en memoria para un problema particular puede ser excesivo.

Por ello, si el problema involucra una gran cantidad de nodos interiores y el equipo en el que se implantará la ejecución del programa tiene una cantidad de memoria muy limitada, es recomendable usar el método de Gradiente Conjugado o alguna variante de GMRES, este consume una cantidad de memoria adicional muy pequeña y el tiempo de ejecución se optimiza versus la Factorización LU.

De esta forma, es posible adaptar el código para tomar en cuenta la implementación de este en un equipo de cómputo en particular y poder sacar el máximo provecho al método de Subestructuración en la resolución de problemas elípticos de gran envergadura.

En lo que resta del presente trabajo, se asume que el método empleado para resolver $\underline{\underline{A}}^i_{II}\underline{x} = \underline{b}$ en sus respectivas variantes necesarias para evitar el cálculo de $\left(\underline{\underline{A}}^i_{II}\right)^{-1}$ es el método de Gradiente Conjugado, logrando así el máximo desempeño en velocidad en tiempo de ejecución.

El número de condicionamiento del complemento de Schur sin precondicionamiento puede ser estimado, para ello:

Definición 1 *Introducimos una norma-L^2 equivalente sobre Γ mediante*

$$\left\|\underline{u_\Gamma}\right\|^2_\Gamma = \sum_{i=1}^{E} \left\|\underline{u_\Gamma}\right\|^2_{L^2(\partial\Omega_i)}. \tag{2.68}$$

Teorema 2 *Sea* $\underline{u_\Gamma}$ *la traza de funciones de elemento finito en* V^h *sobre* Γ, *asumimos que los coeficientes de la ecuación diferencial parcial* $\rho_i = 1$, $i = 1, 2, ..., E$, *y que la malla fina* $\mathcal{T}_h$ *y la malla gruesa* $\mathcal{T}_H$ *sea cuasi-uniforme. Entonces existen dos constantes positivas* c *y* C, *independientes de* h *y* H, *tal que*

$$cH \left\|\underline{u_\Gamma}\right\|_\Gamma^2 \leq s(\underline{u_\Gamma}, \underline{u_\Gamma}) \leq Ch^{-1} \left\|\underline{u_\Gamma}\right\|_\Gamma^2 \tag{2.69}$$

de este modo

$$\kappa = cond(\underline{\underline{S}}) \leq \frac{C}{Hh}. \tag{2.70}$$

Por analogía al método de subestructuración desarrollado anteriormente, dado un sistema lineal $\underline{\underline{M}}\underline{x} = \underline{f}$ que proviene de la discretización de algún método tipo Diferencias Finitas, Elemento Finito o Volumen Finito, siempre es posible reacomodarlo como

$$\begin{pmatrix} \underline{\underline{A}} & \underline{\underline{B}} \\ \underline{\underline{C}} & \underline{\underline{D}} \end{pmatrix} \begin{pmatrix} \underline{u} \\ \underline{v} \end{pmatrix} = \begin{pmatrix} \underline{a} \\ \underline{b} \end{pmatrix} \tag{2.71}$$

con $\underline{\underline{M}} = \begin{pmatrix} \underline{\underline{A}} & \underline{\underline{B}} \\ \underline{\underline{C}} & \underline{\underline{D}} \end{pmatrix}$, $\underline{x} = \begin{pmatrix} \underline{u} \\ \underline{v} \end{pmatrix}$ y $\underline{f} = \begin{pmatrix} \underline{a} \\ \underline{b} \end{pmatrix}$, en la cual la matriz $\underline{\underline{A}}$ sea invertible, entonces

$$\left(\underline{\underline{D}} - \underline{\underline{CA}}^{-1}\underline{\underline{B}}\right)\underline{v} = \underline{b} - \underline{\underline{CA}}^{-1}\underline{a} \tag{2.72}$$

y donde

$$\underline{u} = \underline{\underline{A}}^{-1}\left(\underline{a} - \underline{\underline{B}}\underline{v}\right). \tag{2.73}$$

Así, hemos transformado el sistema $\underline{\underline{M}}\underline{x} = \underline{f}$, en otro equivalente

$$\underline{\underline{N}}\underline{v} = \underline{g} \tag{2.74}$$

pero con un menor número de grados de libertad, donde

$$\underline{\underline{N}} = \left(\underline{\underline{D}} - \underline{\underline{CA}}^{-1}\underline{\underline{B}}\right), \qquad \underline{g} = \underline{b} - \underline{\underline{CA}}^{-1}\underline{a}. \tag{2.75}$$

a esta descomposición matricial se le conoce como la descomposición de Schur.

Así, para solucionar el sistema lineal $\underline{\underline{N}}\underline{v} = \underline{g}$, seleccionaremos el método adecuado acorde a las propiedades de la matriz $\underline{\underline{N}}$ como se vio en el presente capítulo, siendo los más comunes el método CGM -para matrices simétricas- y variantes del método GMRES -para matrices no simétricas-, entre otros.

2.2.1 Precondicionador Derivado de la Matriz de Rigidez

En esta sección detallaremos la construcción del precondicionador derivado de la matriz de rigidez para problemas elípticos usando el método de subestructuración. Para mayor información de estos y otros precondicionadores ver [46], [6], [5] y [3].

Para el caso en que Ω es subdividido en $E = 2$ subdominios $\bar{\Omega} = \bar{\Omega}_1 \cup \bar{\Omega}_2$, la matriz $\underline{\underline{A}}$ queda expresada como

$$\underline{\underline{A}} = \begin{pmatrix} \underline{\underline{A}}_1^{II} & 0 & \underline{\underline{A}}_1^{I\Sigma} \\ 0 & \underline{\underline{A}}_2^{II} & \underline{\underline{A}}_2^{I\Sigma} \\ \underline{\underline{A}}_1^{\Sigma I} & \underline{\underline{A}}_2^{\Sigma I} & \underline{\underline{A}}_1^{\Sigma\Sigma} + \underline{\underline{A}}_2^{\Sigma\Sigma} \end{pmatrix} \tag{2.76}$$

y puede expresarse de forma factorizada como

$$\underline{\underline{A}} = \underline{\underline{L D L^T}} \tag{2.77}$$

donde, denotando por $\underline{\underline{I}}_1, \underline{\underline{I}}_2$ y $\underline{\underline{I}}_\Sigma$ a las matrices identidad de dimensión N_1, N_2 y N_Σ respectivamente, y a

$$\underline{\underline{L}} = \begin{pmatrix} \underline{\underline{I}}_1 & 0 & 0 \\ 0 & \underline{\underline{I}}_2 & 0 \\ \underline{\underline{A}}_1^{\Sigma I} \left(\underline{\underline{A}}_1^{II}\right)^{-1} & \underline{\underline{A}}_2^{\Sigma I} \left(\underline{\underline{A}}_2^{II}\right)^{-1} & \underline{\underline{I}}_\Sigma \end{pmatrix}, \tag{2.78}$$

$$\underline{\underline{D}} = \begin{pmatrix} \underline{\underline{A}}_1^{II} & 0 & 0 \\ 0 & \underline{\underline{A}}_2^{II} & 0 \\ 0 & 0 & \underline{\underline{S}}_1 + \underline{\underline{S}}_2 \end{pmatrix} \tag{2.79}$$

y

$$S_i = \underline{\underline{A}}_i^{\Sigma\Sigma} - \underline{\underline{A}}_i^{\Sigma I} \left(\underline{\underline{A}}_i^{II}\right)^{-1} \underline{\underline{A}}_i^{I\Sigma}, \text{ con } i = 1, 2 \tag{2.80}$$

entonces, equivalentemente, obtendremos la siguiente descomposición por bloques $\underline{\underline{LU}}$ de la matriz $\underline{\underline{A}} = \underline{\underline{LU}}$ donde

$$\underline{\underline{U}} = \underline{\underline{D L^T}} = \begin{pmatrix} \underline{\underline{A}}_1^{II} & 0 & \underline{\underline{A}}_1^{I\Sigma} \\ 0 & \underline{\underline{A}}_2^{II} & \underline{\underline{A}}_2^{I\Sigma} \\ 0 & 0 & \underline{\underline{S}}_1 + \underline{\underline{S}}_2 \end{pmatrix}. \tag{2.81}$$

Asumiendo que el precondicionador $\underline{\underline{P}}_h$ conveniente está disponible para la matriz $\underline{\underline{S}} = \underline{\underline{S}}_1 + \underline{\underline{S}}_2$, entonces podemos construir el siguiente precondicionador $\underline{\underline{Q}}_h$ de $\underline{\underline{A}}$:

$$\underline{\underline{Q}}_h = \underline{\underline{L\tilde{U}}} \tag{2.82}$$

donde $\underline{\underline{L}}$ es dada como en (2.78) y $\underline{\underline{\tilde{U}}}$ es obtenida de $\underline{\underline{U}}$ de (2.81) por la aproximación de $\underline{\underline{S}}$ con $\underline{\underline{P}}_h$; es decir

$$\underline{\underline{\tilde{U}}} = \begin{pmatrix} \underline{\underline{A}}_1^{II} & 0 & \underline{\underline{A}}_1^{I\Sigma} \\ 0 & \underline{\underline{A}}_2^{II} & \underline{\underline{A}}_2^{I\Sigma} \\ 0 & 0 & \underline{\underline{P}}_h \end{pmatrix}. \tag{2.83}$$

Notemos que los bloques de $\underline{\underline{Q}}_h$ coinciden con los de $\underline{\underline{A}}$, excepto para el bloque $(3,3)$, el cual es

$$\left(\underline{\underline{Q}}_h\right)_{3,3} = \underline{\underline{A}}_1^{\Sigma I}\left(\underline{\underline{A}}_1^{II}\right)^{-1}\underline{\underline{A}}_1^{I\Sigma} + \underline{\underline{A}}_2^{\Sigma I}\left(\underline{\underline{A}}_2^{II}\right)^{-1}\underline{\underline{A}}_2^{I\Sigma} + \underline{\underline{P}}_h \tag{2.84}$$

pero tomando como precondicionador $\underline{\underline{S}}_2$ obtenemos

$$\left(\underline{\underline{Q}}_h\right)_{3,3} = \underline{\underline{A}}_1^{\Sigma I}\left(\underline{\underline{A}}_1^{II}\right)^{-1}\underline{\underline{A}}_1^{I\Sigma} + \underline{\underline{A}}_2^{\Sigma\Sigma}. \tag{2.85}$$

Sea λ un eigenvalor de $\underline{\underline{Q}}_h\underline{\underline{A}}$ y $\underline{w} \in \mathbb{R}^{N_\Sigma}$ su correspondiente eigenvector, escribimos $\underline{w} = (\underline{w}_1, \underline{w}_2, \underline{w}_\Sigma)$, por obvias razones de notación, tenemos que

$$\underline{\underline{A}}\underline{w} = \lambda \underline{\underline{Q}}_h \underline{w} \tag{2.86}$$

donde

$$\begin{cases} (1-\lambda)\left(\underline{\underline{A}}_1^{II}\underline{w}_1 + \underline{\underline{A}}_1^{II}\underline{w}_\Sigma\right) = 0 \\ (1-\lambda)\left(\underline{\underline{A}}_2^{II}\underline{w}_2 + \underline{\underline{A}}_2^{II}\underline{w}_\Sigma\right) = 0 \\ (1-\lambda)\left(\underline{\underline{A}}_1^{\Sigma I}\underline{w}_1 + \underline{\underline{A}}_2^{\Sigma I}\underline{w}_2\right) + \underline{\underline{A}}^{\Sigma\Sigma}\underline{w}_\Sigma \\ = \lambda\left(\underline{\underline{P}}_h\underline{w}_\Sigma + \underline{\underline{A}}_1^{\Sigma I}\left(\underline{\underline{A}}_1^{II}\right)^{-1}\underline{\underline{A}}_1^{I\Sigma}\underline{w}_\Sigma + \underline{\underline{A}}_2^{\Sigma I}\left(\underline{\underline{A}}_2^{II}\right)^{-1}\underline{\underline{A}}_2^{I\Sigma}\underline{w}_\Sigma\right). \end{cases} \tag{2.87}$$

Reescribiendo la última ecuación como

$$(1-\lambda)\left(\underline{\underline{A}}_1^{\Sigma I}\underline{w}_1 + \underline{\underline{A}}_2^{\Sigma I}\underline{w}_2 + \underline{\underline{A}}_1^{\Sigma I}\left(\underline{\underline{A}}_1^{II}\right)^{-1}\underline{\underline{A}}_1^{I\Sigma}\underline{w}_\Sigma + \right. \tag{2.88}$$

$$\left. \underline{\underline{A}}_2^{\Sigma I}\left(\underline{\underline{A}}_2^{II}\right)^{-1}\underline{\underline{A}}_2^{I\Sigma}\underline{w}_\Sigma\right) + \underline{\underline{S}}\underline{w}_\Sigma = \lambda \underline{\underline{P}}_h \underline{w}_\Sigma.$$

y si $\lambda \neq 1$, tenemos

$$\begin{cases} \underline{\underline{A}}_1^{II}\underline{w}_1 + \underline{\underline{A}}_1^{I\Sigma}\underline{w}_\Sigma = 0 \\ \underline{\underline{A}}_2^{II}\underline{w}_2 + \underline{\underline{A}}_2^{I\Sigma}\underline{w}_\Sigma = 0 \end{cases} \tag{2.89}$$

por lo tanto, $\underline{w}_\Sigma \neq 0$ y $\underline{\underline{S}}\underline{w}_\Sigma = \lambda \underline{\underline{P}}_h \underline{w}_\Sigma$.

De lo anterior podemos concluir que la matriz $\left(\underline{\underline{Q}}_h\right)^{-1}\underline{\underline{A}}$ tiene los mismos eigenvalores que $\left(\underline{\underline{P}}_h\right)^{-1}$, además de el eigenvalor 1, el cual es también un eigenvalor de $\left(\underline{\underline{P}}_h\right)^{-1}\underline{\underline{S}}$ siempre que el correspondiente eigenvector $\underline{w}$ satisfazca que $\underline{w}_\Sigma \neq 0$.

Si asumimos que $\underline{\underline{P}}_h$ es espectralmente equivalente a $\underline{\underline{S}}$, es decir, existen dos constantes K_1 y K_2 independientes de h, tal que

$$K_1\left[\underline{\underline{P}}_h\underline{\eta}, \underline{\eta}\right] \leq \left[\underline{\underline{S}}\underline{\eta}, \underline{\eta}\right] \leq K_2\left[\underline{\underline{P}}_h\underline{\eta}, \underline{\eta}\right] \tag{2.90}$$

para toda $\underline{\eta} \in \mathbb{R}^{N_\Sigma}$. Entonces se deriva de la caracterización de los eigenvalores de $\left(\underline{\underline{Q}}_h\right)^{-1}\underline{\underline{A}}$ que satisface

$$\kappa = \left(\left(\underline{\underline{Q}}_h\right)^{-1}\underline{\underline{A}}\right) \leq \frac{\tilde{K}_2}{\tilde{K}_1} \tag{2.91}$$

donde

$$\tilde{K}_1 = \min\{1, K_1\}, \tilde{K}_2 = \min\{1, K_2\}$$

por lo tanto podemos concluir que $\underline{\underline{Q}}_h$ es espectralmente equivalente a $\underline{\underline{A}}$.

En conclusión, el precondicionador $\underline{\underline{Q}}_h$ hereda todas las buenas propiedades que tiene $\underline{\underline{P}}_h$ en términos de buen paralelismo y equivalencia espectral.

En el caso de una partición con $E > 2$ subdominios se puede proceder de manera similar. En este caso Ω será particionado en E subdominios Ω_i que no se traslapen, de diámetro h_i, con Σ como frontera interior, es decir

$$\Sigma = \bigcup_{i=1}^{E} \Sigma_i, \quad \text{si } \Sigma_i = \partial\Omega_i \backslash \partial\Omega$$

y sea $\alpha = \bigcup_{i=1}^{E} N_i$ que denote los índices correspondientes a los nodos internos.

Entonces, para usar la misma notación podemos escribir el problema algebraico $\underline{\underline{A}}\underline{u} = \underline{b}$ en bloques como sigue

$$\begin{pmatrix} \underline{\underline{A}}_{\alpha}^{II} & \underline{\underline{A}}_{\alpha}^{I\Sigma} \\ \underline{\underline{A}}_{\alpha}^{\Sigma I} & \underline{\underline{A}}_{\alpha}^{\Sigma\Sigma} \end{pmatrix} \begin{pmatrix} \underline{u}_{\alpha} \\ \underline{u}_{\Sigma} \end{pmatrix} = \begin{pmatrix} \underline{b}_{\alpha} \\ \underline{b}_{\Sigma} \end{pmatrix} \tag{2.92}$$

donde $\underline{\underline{A}}_{\alpha}^{\Sigma I} = \left(\underline{\underline{A}}_{\alpha}^{I\Sigma}\right)^{T}$.

Ya que los nodos interiores de cada subdominio permanecen desacoplados de los nodos interiores de los demás subdominios, obtenemos

$$\underline{\underline{A}}^{II} \equiv \begin{bmatrix} \underline{\underline{A}}_{1}^{II} & 0 & \cdots & 0 \\ 0 & \underline{\underline{A}}_{2}^{II} & 0 & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & 0 & \underline{\underline{A}}_{E}^{II} \end{bmatrix} \tag{2.93}$$

el bloque i de $\underline{\underline{A}}_{i}^{II}$ es la submatriz principal de la matriz local de rigidez que está asociada al problema del subdominio Ω_i.

En efecto, la matriz

$$\underline{\underline{A}}_{i}^{II} \equiv \begin{pmatrix} \underline{\underline{A}}_{i}^{II} & \underline{\underline{A}}_{i}^{I\Sigma} \\ \underline{\underline{A}}_{i}^{\Sigma I} & \underline{\underline{A}}_{i}^{\Sigma\Sigma} \end{pmatrix} \tag{2.94}$$

es la matriz de elemento finito asociado con el problema de Poisson en Ω_i con frontera Σ_i de Neumann.

Similarmente, la matriz

$$\underline{\underline{D}}_{i} \equiv \begin{pmatrix} \underline{\underline{A}}_{i}^{II} & \underline{\underline{A}}_{i}^{I\Sigma} \\ 0 & \underline{\underline{I}}_{\Sigma} \end{pmatrix} \tag{2.95}$$

donde $\underline{\underline{I}}_{\Sigma}$ es la matriz identidad de orden N_Σ que como en el caso anterior es asociada con el problema de Poisson en Ω_i con frontera Σ_i de Dirichlet.

La matriz del complemento de Schur $\underline{\underline{S}}$ asociada con las variables de la frontera interior u_Σ de (2.92) es

$$\underline{\underline{S}} = \underline{\underline{A}}^{\Sigma\Sigma} - \underline{\underline{A}}^{\Sigma I}\left(\underline{\underline{A}}^{II}\right)^{-1}\underline{\underline{A}}^{I\Sigma} \tag{2.96}$$

pero $\underline{\underline{S}} = \sum_{i=1}^{E} \underline{\underline{S}}_{i}$ donde

$$\underline{\underline{S}}_{i} = \underline{\underline{A}}_{i}^{\Sigma\Sigma} - \underline{\underline{A}}_{i}^{\Sigma I}\left(\underline{\underline{A}}_{i}^{II}\right)^{-1}\underline{\underline{A}}_{i}^{I\Sigma}. \tag{2.97}$$

En cuanto al número de condicionamiento de $\underline{\underline{S}}$ satisface

$$\kappa = \left(\underline{\underline{S}}\right) \leq C\frac{H_{\max}}{hH_{\min}^2} \tag{2.98}$$

donde $H_{\min}$ y $H_{\max}$ denotan respectivamente el mínimo y máximo de los diámetros de los subdominios.

Continuando con la aproximación directa usada en dos subdominios, obtenemos que la matriz de rigidez $\underline{\underline{A}}$ puede ser factorizada como sigue

$$\underline{\underline{A}} = \underline{\underline{LDL}}^T \tag{2.99}$$

con

$$\underline{\underline{L}} = \begin{pmatrix} \underline{\underline{I}}_{\Omega\setminus\Sigma} & 0 \\ \underline{\underline{A}}_{\alpha}^{\Sigma I}\left(\underline{\underline{A}}_{\alpha}^{II}\right)^{-1} & \underline{\underline{I}}_{\Sigma} \end{pmatrix}, \tag{2.100}$$

$$\underline{\underline{D}} = \begin{pmatrix} \underline{\underline{A}}_{\alpha}^{II} & 0 \\ 0 & \underline{\underline{S}} \end{pmatrix} \tag{2.101}$$

o

$$\underline{\underline{A}} = \underline{\underline{LU}} \tag{2.102}$$

con

$$\underline{\underline{U}} = \begin{pmatrix} \underline{\underline{A}}_{\alpha}^{II} & \underline{\underline{A}}_{\alpha}^{I\Sigma} \\ 0 & \underline{\underline{S}} \end{pmatrix}. \tag{2.103}$$

$\underline{\underline{P}}_h$ será un precondicionador para el complemento de Schur $\underline{\underline{S}}$, entonces la matriz

$$\underline{\underline{Q}}_h = \underline{\underline{L\tilde{U}}} \tag{2.104}$$

con

$$\underline{\underline{\tilde{U}}} = \begin{pmatrix} \underline{\underline{A}}_{\alpha}^{II} & \underline{\underline{A}}_{\alpha}^{I\Sigma} \\ 0 & \underline{\underline{P}}_h \end{pmatrix} \tag{2.105}$$

es un precondicionador para la matriz de rigidez $\underline{\underline{A}}$. Los eigenvalores de

$$\left(\underline{\underline{Q}}_h\right)^{-1}\underline{\underline{A}} \tag{2.106}$$

son los mismos que para

$$\left(\underline{\underline{P}}_h\right)^{-1}\underline{\underline{S}} \tag{2.107}$$

además del eigenvalor 1.

3 Implementación del Método de Subestructuración

A partir de los modelos matemáticos y los modelos numéricos, en este capítulo se describe el modelo computacional contenido en un programa de cómputo orientado a objetos en el lenguaje de programación C++ en su forma secuencial y en su forma paralela en C++ usando la interfaz de paso de mensajes (MPI) bajo el esquema maestro-esclavo.

Esto no sólo nos ayudará a demostrar que es factible la construcción del propio modelo computacional a partir del modelo matemático y numérico para la solución de problemas reales. Además, se mostrará los alcances y limitaciones en el consumo de los recursos computacionales, evaluando algunas de las variantes de los métodos numéricos con los que es posible implementar el modelo computacional y haremos el análisis de rendimiento sin llegar a ser exhaustivo esté.

También exploraremos los alcances y limitaciones de cada uno de los métodos implementados (FEM, DDM secuencial y paralelo) y como es posible optimizar los recursos computacionales con los que se cuente.

Primeramente hay que destacar que el paradigma de programación orientada a objetos es un método de implementación de programas, organizados como colecciones cooperativas de objetos. Cada objeto representa una instancia de alguna clase y cada clase es miembro de una jerarquía de clases unidas mediante relaciones de herencia, contención, agregación o uso.

Esto nos permite dividir en niveles la semántica de los sistemas complejos tratando así con las partes, que son más manejables que el todo, permitiendo su extensión y un mantenimiento más sencillo. Así, mediante la herencia, contención, agregación o usó nos permite generar clases especializadas que manejan eficientemente la complejidad del problema. La programación orientada a objetos organiza un programa entorno a sus datos (atributos) y a un conjunto de interfases bien definidas para manipular estos datos (métodos dentro de clases reusables) esto en oposición a los demás paradigmas de programación.

El paradigma de programación orientada a objetos sin embargo sacrifica algo de eficiencia computacional por requerir mayor manejo de recursos computacionales al momento de la ejecución. Pero en contraste, permite mayor flexibilidad al adaptar los códigos a nuevas especificaciones. Adicionalmente, disminuye notoriamente el tiempo invertido en el mantenimiento y búsqueda

de errores dentro del código. Esto tiene especial interés cuando se piensa en la cantidad de meses invertidos en la programación comparado con los segundos consumidos en la ejecución del mismo.

Para empezar con la implementación computacional, primeramente definiremos el problema a trabajar. Este, pese a su sencillez, no pierde generalidad permitiendo que el modelo mostrado sea usado en muchos sistemas de la ingeniería y la ciencia.

3.1 El Operador de Laplace y la Ecuación de Poisson

Consideramos como modelo matemático el problema de valor en la frontera (BVP) asociado con el operador de Laplace en dos dimensiones, el cual en general es usualmente referido como la ecuación de Poisson, con condiciones de frontera Dirichlet, definido en Ω como:

$$\begin{aligned} -\nabla^2 u &= f_\Omega \text{ en } \Omega \\ u &= g_{\partial\Omega} \text{ en } \partial\Omega. \end{aligned} \tag{3.1}$$

Se toma está ecuación para facilitar la comprensión de las ideas básicas. Es un ejemplo muy sencillo, pero gobierna los modelos de muchos sistemas de la ingeniería y de la ciencia, entre ellos el flujo de agua subterránea a través de un acuífero isotrópico, homogéneo bajo condiciones de equilibrio y es muy usada en múltiples ramas de la física. Por ejemplo, gobierna la ecuación de la conducción de calor en un sólido bajo condiciones de equilibrio.

En particular consideramos el problema con Ω definido en:

$$\Omega = [-1, 1] \times [0, 1] \tag{3.2}$$

donde

$$f_\Omega = 2n^2\pi^2 \sin(n\pi x) * \sin(n\pi y) \quad \text{y} \quad g_{\partial\Omega} = 0 \tag{3.3}$$

cuya solución es

$$u(x, y) = \sin(n\pi x) * \sin(n\pi y). \tag{3.4}$$

Para las pruebas de rendimiento en las cuales se evalúa el desempeño de los programas realizados se usa $n = 10$, pero es posible hacerlo con $n \in \mathbb{N}$ grande. Por ejemplo para $n = 4$, la solución es $u(x, y) = \sin(4\pi x) * \sin(4\pi y)$, cuya gráfica se muestra a continuación:

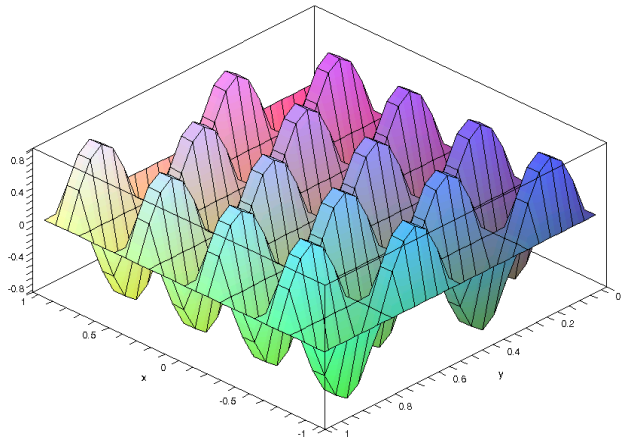


Figura 4: Solución a la ecuación de Poisson para n=4.

Hay que hacer notar que al implementar la solución numérica por el método del elemento finito y el método de subestructuración secuencial en un procesador, un factor limitante para su operación es la cantidad de memoria disponible en la computadora, ya que el sistema algebraico de ecuaciones asociado a esté problema crece muy rápido (del orden de n^2), donde n es el número de nodos en la partición. Es por ello que la elección de un buen manejador de matrices será determinante en la eficiencia alcanzada por las distintas implementaciones de los programas.

Actualmente existen múltiples bibliotecas que permiten manipular operaciones de matrices tanto en forma secuencial como en paralelo (hilos y Pipeline) para implementarlas tanto en procesadores con memoria compartida como distribuida. Pero no están presentes en todas las arquitecturas y sistemas operativos. Por ello en este trabajo se implementaron todas las operaciones necesarias usando clases que fueron desarrolladas sin usar ninguna biblioteca externa a las proporcionadas por el compilador de C++ de GNU, permitiendo la operación de los programas desarrollados en múltiples sistemas operativos del tipo Linux, Unix y Windows.

En cuanto a las pruebas de rendimiento que mostraremos en las siguientes secciones se usaron para la parte secuencial el equipo:

- Computadora Pentium IV HT a 2.8 GHtz con 1 GB de RAM corriendo bajo el sistema operativo Linux Debian Stable con el compilador g++ de GNU.

Para la parte paralela se usaron los equipos siguientes:

- Cluster homogéneo de 10 nodos duales Xeon a 2.8 GHtz con 1 GB de RAM por nodo, unidos mediante una red Ethernet de 1 Gb, corriendo bajo el sistema operativo Linux Debian Stable con el compilador mpiCC de MPI de GNU.
- Cluster heterogéneo con el nodo maestro Pentium IV HT a 3.4 GHtz con 1 GB de RAM y 7 nodos esclavos Pentium IV HT a 2.8 GHtz con 0.5 GB de RAM por nodo, unidos mediante una red Ethernet de 100 Mb, corriendo bajo el sistema operativo Linux Debian Stable con el compilador mpiCC de MPI de GNU.

A estos equipos nos referiremos en lo sucesivo como equipo secuencial, cluster homogéneo y cluster heterogéneo respectivamente.

El tiempo dado en los resultados de las distintas pruebas de rendimiento de los programas y mostrado en todas las tablas y gráficas fue tomado como un promedio entre por lo menos 5 corridas, redondeado el resultado a la unidad siguiente. En todos los cálculos de los métodos numéricos usados para resolver el sistema lineal algebraico asociado se usó una tolerancia mínima de 1×10^{-10}.

Ahora, veremos la implementación del método de elemento finito secuencial para después continuar con el método de descomposición de dominio tanto secuencial como paralelo y poder analizar en cada caso los requerimientos de cómputo, necesarios para correr eficientemente un problema en particular.

3.2 Método del Elemento Finito Secuencial

A partir de la formulación del método de elemento finito visto en la sección (8.3), la implementación computacional que se desarrolló tiene la jerarquía de clases siguiente:

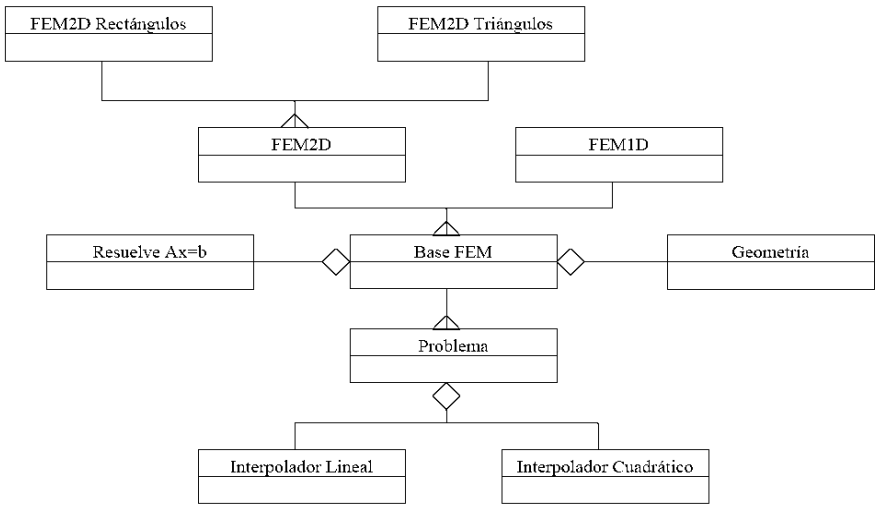


Figura 5: Jerarquía de clases para el método de elemento finito

Donde las clases participantes en *FEM2D Rectángulos* son:

La clase *Interpolador Lineal* define los interpoladores lineales usados por el método de elemento finito.

La clase *Problema* define el problema a tratar, es decir, la ecuación diferencial parcial, valores de frontera y dominio.

La clase *Base FEM* ayuda a definir los nodos al usar la clase *Geometría* y mantiene las matrices generadas por el método y a partir de la clase *Resuelve Ax=B* se dispone de diversas formas de resolver el sistema lineal asociado al método.

La clase *FEM2D* controla lo necesario para poder hacer uso de la geometría en 2D y conocer los nodos interiores y de frontera, con ellos poder montar la matriz de rigidez y ensamblar la solución.

La clase *FEM2D Rectángulos* permite calcular la matriz de rigidez para generar el sistema algebraico de ecuaciones asociado al método.

Notemos que esta misma jerarquía permite trabajar problemas en una y dos dimensiones, en el caso de dos dimensiones podemos discretizar usando rectángulos o triángulos, así como usar varias opciones para resolver el sistema lineal algebraico asociado a la solución de EDP.

Como ya se mencionó, el método de elemento finito es un algoritmo secuencial, por ello se implementa para que use un solo procesador y un factor limitante para su operación es la cantidad de memoria disponible en la computadora, por ejemplo:

Resolver la Ec. (3.1) con una partición rectangular de 513×513 nodos, genera 262144 elementos rectangulares con 263169 nodos en total, donde 261121 son desconocidos; así el sistema algebraico de ecuaciones asociado a este problema es de dimensión 261121×261121.

Usando el equipo secuencial, primeramente evaluaremos el desempeño del método de elemento finito con los distintos métodos para resolver el sistema algebraico de ecuaciones, encontrando los siguientes resultados:

Método Iterativo	Iteraciones	Tiempo Total
Jacobi	865037	115897 seg.
Gauss-Seidel	446932	63311 seg.
Gradiente Conjugado	761	6388 seg.

Como se observa el uso del método de gradiente conjugado es por mucho la mejor elección. En principio, podríamos quedarnos solamente con el método de gradiente conjugado sin hacer uso de precondicionadores por los buenos rendimientos encontrados hasta aquí, pero si se desea resolver un problema con un gran número de nodos, es conocido el aumento de eficiencia al hacer uso de precondicionadores.

Ahora, si tomamos ingenuamente el método de elemento finito conjuntamente con el método de gradiente conjugado con precondicionadores a posteriori (los más sencillos de construir) para resolver el sistema algebraico de ecuaciones, encontraremos los siguientes resultados:

Precondicionador	Iteraciones	Tiempo Total
Jacobi	760	6388 seg.
SSOR	758	6375 seg.
Factorización Incompleta	745	6373 seg.

Como es notorio el uso del método de gradiente conjugado precondicionado con precondicionadores a posteriori no ofrece una ventaja significativa que compense el esfuerzo computacional invertido al crear y usar un precondicionador en los cálculos por el mal condicionamiento del sistema algebraico. Existen también precondicionadores a priori para el método de elemento finito, pero no es costeable en rendimiento su implementación.

Finalmente, para el método de elemento finito las posibles mejoras de eficiencia para disminuir el tiempo de ejecución pueden ser:

- Al momento de compilar los códigos usar directivas de optimización (ofrece mejoras de rendimiento en ejecución de 30% aproximadamente en las pruebas realizadas).
- Usar la biblioteca Lapack++ de licencia GNU que optimiza las ope-raciones en el manejo de los elementos de la matriz usando punteros y hacen uso de matrices bandadas (obteniéndose una mejora del rendimiento de 15% aproximadamente en las pruebas realizadas).
- Combinando las opciones anteriores se obtiene una mejora sustancial de rendimiento en la ejecución (de 45% aproximadamente en las pruebas realizadas).

Adicionalmente si se cuenta con un equipo con más de un procesador con memoria compartida es posible usar bibliotecas para la manipulación de matrices y vectores que paralelizan o usan Pipeline como una forma de mejorar el rendimiento del programa. Este tipo de mejoras cuando es posible usarlas disminuyen sustancialmente el tiempo de ejecución, ya que en gran medida el consumo total de CPU está en la manipulación de matrices, pero esto no hace paralelo al método de elemento finito.

3.3 Método de Subestructuración Secuencial

A partir de la formulación del método de subestructuración visto en la sección (2.2) se generan las matrices locales $\underline{\underline{A}}_i^{II}, \underline{\underline{A}}_i^{I\Sigma}, \underline{\underline{A}}_i^{\Sigma I}$ y $\underline{\underline{A}}_i^{\Sigma\Sigma}$ y con ellas se construyen $\underline{\underline{S}}_i = \underline{\underline{A}}_i^{\Sigma\Sigma} - \underline{\underline{A}}_i^{\Sigma I}\left(\underline{\underline{A}}_i^{II}\right)^{-1}\underline{\underline{A}}_i^{I\Sigma}$ y $\underline{b}_i = \underline{\underline{A}}_i^{\Sigma I}\left(\underline{\underline{A}}_i^{II}\right)^{-1}\underline{b}_{I_i}$ que son problemas locales a cada subdominio Ω_i, con $i = 1, 2, ..., E$. Generando de manera virtual el sistema lineal $\underline{\underline{S}}\,\underline{u_\Sigma} = \underline{b}$ a partir de

$$\left[\sum_{i=1}^{E} \underline{\underline{S}}_i\right] \underline{u_\Sigma} = \left[\sum_{i=1}^{E} \underline{b}_i\right] \tag{3.5}$$

donde $\underline{\underline{S}} = \left[\sum_{i=1}^{E} \underline{\underline{S}}_i\right]$ y $\underline{b} = \left[\sum_{i=1}^{E} \underline{b}_i\right]$ podría ser construida sumando las $\underline{\underline{S}}_i$ y $\underline{b}_i$ respectivamente según el orden de los nodos globales versus los nodos locales a cada subdominio.

El sistema lineal virtual resultante

$$\underline{\underline{S}}\,\underline{u_\Sigma} = \underline{b} \tag{3.6}$$

es resuelto usando el método de gradiente conjugado visto en la sección (11.2.5), para ello no es necesario construir la matriz $\underline{\underline{S}}$ con las contribuciones de cada S_i correspondientes al subdominio i. Lo que hacemos es pasar a cada subdominio el vector $\underline{u_\Sigma}^i$ correspondiente a la i-ésima iteración del método de gradiente conjugado para que en cada subdominio se evalué $\underline{\tilde{u}_\Sigma}^i = \underline{\underline{S}}_i \underline{u_\Sigma}^i$ localmente y con el resultado se forma el vector $\underline{\tilde{u}_\Sigma} = \sum_{i=1}^{E} \underline{\tilde{u}_\Sigma}^i$ y se continué con los demás pasos del método.

La implementación computacional que se desarrolló tiene una jerarquía de clases en la cual se agregan las clases *FEM2D Rectángulos* y *Geometría*, además de heredar a la clase *Problema*. De esta forma se rehusó todo el código desarrollado para *FEM2D Rectángulos*, la jerarquía queda como:

> La clase *DDM2D* realiza la partición gruesa del dominio mediante la clase *Geometría* y controla la partición de cada subdominio mediante un objeto de la clase de *FEM2D Rectángulos* generando la partición fina del dominio. La resolución de los nodos de la frontera interior se hace mediante el método de gradiente conjugado, necesaria para resolver los nodos internos de cada subdominio.

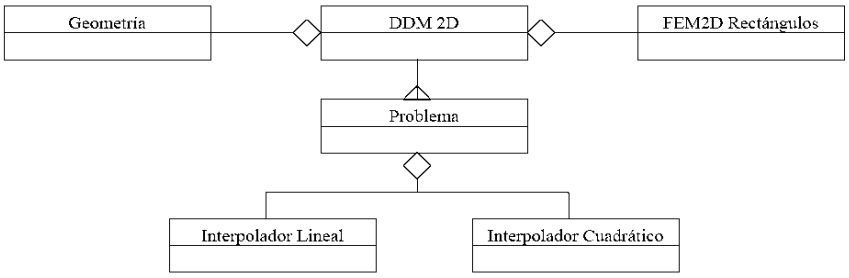


Figura 6: Jerarquía de clases para el método de subestructuración secuencial

Así, el dominio Ω es descompuesto en una descomposición gruesa de $n \times m$ subdominios y cada subdominio Ω_i se parte en $p \times q$ subdominios, generando la participación fina del dominio como se muestra en la figura:

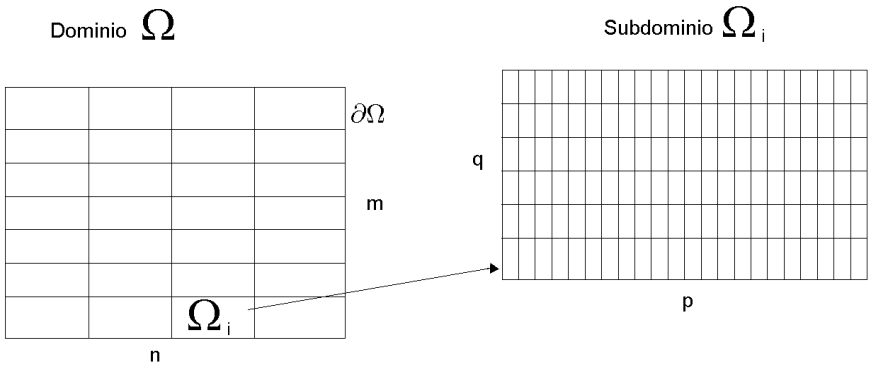


Figura 7: Descomposición del dominio Ω en $E = n \times m$ subdominios y cada subdominio Ω_i en $p \times q$ subdominios

El método de descomposición de dominio se implementó realizando las siguientes tareas:

A) La clase *DDM2D* genera la descomposición gruesa del dominio mediante la agregación de un objeto de la clase *Geometría* (supongamos particionado en $n \times m$ subdominios, generando $s = n * m$ subdominios Ω_i, $i = 1, 2, ..., E$).

B) Con esa geometría se construyen los objetos de *FEM2D Rectángulos* (uno por cada subdominio Ω_i), donde cada subdominio es particionado (supongamos en $p \times q$ subdominios) y regresando las coordenadas de los nodos de frontera del subdominio correspondiente a la clase DDM2D.

C) Con estas coordenadas, la clase *DDM2D* conoce a los nodos de la frontera interior (son estos los que resuelve el método de descomposición de dominio). Las coordenadas de los nodos de la frontera interior se dan a conocer a los objetos *FEM2D Rectángulos*, transmitiendo sólo aquellos que están en su subdominio.

D) Después de conocer los nodos de la frontera interior, cada objeto *FEM2D Rectángulos* calcula las matrices $\underline{\underline{A}}_i^{\Sigma\Sigma}, \underline{\underline{A}}_i^{\Sigma I}, \underline{\underline{A}}_i^{I\Sigma}$ y $\underline{\underline{A}}_i^{II}$ necesarias para construir el complemento de Schur local $\underline{\underline{S}}_i = \underline{\underline{A}}_i^{\Sigma\Sigma} - \underline{\underline{A}}_i^{\Sigma I} \left(\underline{\underline{A}}_i^{II}\right)^{-1} \underline{\underline{A}}_i^{I\Sigma}$ sin realizar comunicación alguna.

Al terminar de calcular las matrices se avisa a la clase *DDM2D* de la finalización de los cálculos.

E) Mediante la comunicación de vectores del tamaño del número de nodos de la frontera interior entre la clase *DDM2D* y los objetos *FEM2D Rectángulos*, se prepara todo lo necesario para empezar el método de gradiente conjugado y resolver el sistema lineal virtual $\left[\sum_{i=1}^{E} \underline{\underline{S}}_i\right] \underline{u_\Sigma} = \left[\sum_{i=1}^{E} \underline{b}_i\right]$.

F) Para usar el método de gradiente conjugado, se transmite un vector del tamaño del número de nodos de la frontera interior para que en cada objeto se realicen las operaciones pertinentes y resolver así el sistema algebraico asociado, esta comunicación se realiza de ida y vuelta entre la clase *DDM2D* y los objetos *FEM2D Rectángulos* tantas veces como iteraciones haga el método. Resolviendo con esto los nodos de la frontera interior $\underline{u_\Sigma}_i$.

G) Al término de las iteraciones se pasa la solución $\underline{u_\Sigma}_i$ de los nodos de la frontera interior que pertenecen a cada subdominio dentro de cada objeto *FEM2D Rectángulos* para que se resuelvan los nodos interiores $\underline{u_I}_i = \left(\underline{\underline{A}}_i^{II}\right)^{-1} \left(\underline{b_I}_i - \underline{\underline{A}}_i^{I\Sigma} \underline{u_\Sigma}_i\right)$, sin realizar comunicación alguna en el proceso, al concluir se avisa a la clase *DDM2D* de ello.

I) La clase *DDM2D* mediante un último mensaje avisa que se concluya el programa, terminado así el esquema maestro-esclavo secuencial.

Por ejemplo, para resolver la Ec. (3.1), usando 513×513 nodos (igual al ejemplo de *FEM2D Rectángulos* secuencial), podemos tomar alguna de las siguientes descomposiciones:

Descomposición	Nodos Interiores	Subdominios	Elementos Subdominio	Total Nodos Subdominio	Nodos Desconocidos Subdominio
2x2 y 256x256	260100	4	65536	66049	65025
4x4 y 128x128	258064	16	16384	16641	16129
8x8 y 64x64	254016	64	4096	4225	3969
16x16 y 32x32	246016	256	1024	1089	961
32x32 y 16x16	230400	1024	256	289	225

Cada una de las descomposiciones genera un problema distinto. Usando el equipo secuencial y evaluando el desempeño del método de subestructuración secuencial se obtuvieron los siguientes resultados:

Partición	Nodos Frontera Interior	Iteraciones	Tiempo Total
2x2 y 256x256	1021	139	5708 seg.
4x4 y 128x128	3057	159	2934 seg.
8x8 y 64x64	7105	204	1729 seg.
16x16 y 32x32	15105	264	1077 seg.
32x32 y 16x16	30721	325	1128 seg

Nótese que aún en un solo procesador es posible encontrar una descomposición que desminuya los tiempos de ejecución (la descomposición de 2x2 y 256x256 concluye en 5708 seg. versus los 6388 seg. en el caso de *FEM2D Rectángulos)*, ello es debido a que al descomponer el dominio en múltiples subdominios, la complejidad del problema es también disminuida y esto se ve reflejado en la disminución del tiempo de ejecución.

En la última descomposición, en lugar de disminuir el tiempo de ejecución este aumenta, esto se debe a que se construyen muchos objetos *FEM2D Rectángulos* (1024 en este caso), con los cuales hay que hacer comunicación resultando muy costoso computacionalmente.

Finalmente las posibles mejoras de eficiencia para el método de subestructuración secuencial para disminuir el tiempo de ejecución son las mismas que en el caso del método de elemento finito pero además se tienen que:

- Encontrar la descomposición pertinente entre las posibles descomposiciones que consuma el menor tiempo de cálculo.
- Adicionalmente si se cuenta con un equipo con más de un procesador con memoria compartida es posible usar bibliotecas para la manipulación de matrices y vectores que paralelizan o usan Pipeline como una forma de mejorar el rendimiento del programa.

Este tipo de mejoras cuando es posible usarlas disminuyen sustancialmente el tiempo de ejecución, ya que en gran medida el consumo total de CPU está en la manipulación de matrices, pero esto no hace paralelo al método de subestructuración secuencial.

3.4 Método de Subestructuración en Paralelo

La computación en paralelo es una técnica que nos permite distribuir una gran carga computacional entre muchos procesadores. Y es bien sabido que una de las mayores dificultades del procesamiento en paralelo es la coordinación de las actividades de los diferentes procesadores y el intercambio de información entre los mismos.

Para hacer una adecuada coordinación de actividades entre los diferentes procesadores, el programa que soporta el método de subestructuración paralelo, usa la misma jerarquía de clases que el método de subestructuración secuencial. Este se desarrolló para usar el esquema maestro-esclavo, de forma tal que el nodo maestro mediante la agregación de un objeto de la clase de *Geometría* genere la descomposición gruesa del dominio y los nodos esclavos creen un conjunto de objetos *FEM2D Rectángulos* para que en estos objetos se genere la participación fina y mediante el paso de mensajes (vía MPI) puedan comunicarse los nodos esclavos con el nodo maestro, realizando las siguientes tareas:

A) El nodo maestro genera la descomposición gruesa del dominio (supongamos particionado en $n \times m$ subdominios) mediante la agregación de un objeto de la clase *Geometría*, esta geometría es pasada a los nodos esclavos.

B) Con esa geometría se construyen los objetos *FEM2D Rectángulos* (uno por cada subdominio), donde cada subdominio es particionado (supongamos en $p \times q$ subdominios). Cada objeto de *FEM2D Rectángulos* genera la geometría solicitada, regresando las coordenadas de los nodos de frontera del subdominio correspondiente al nodo maestro.

C) Con estas coordenadas, el nodo maestro conoce a los nodos de la frontera interior (son estos los que resuelve el método de descomposición de dominio). Las coordenadas de los nodos de la frontera interior se dan a conocer a los objetos *FEM2D Rectángulos* en los nodos esclavos, transmitiendo sólo aquellos que están en su subdominio.

D) Después de conocer los nodos de la frontera interior, cada objeto *FEM2D Rectángulos* calcula las matrices $\underline{\underline{A}}_i^{\Sigma\Sigma}, \underline{\underline{A}}_i^{\Sigma I}, \underline{\underline{A}}_i^{I\Sigma}$ y $\underline{\underline{A}}_i^{II}$ necesarias para construir el complemento de Schur local

$\underline{\underline{S}}_i = \underline{\underline{A}}_i^{\Sigma\Sigma} - \underline{\underline{A}}_i^{\Sigma I}\left(\underline{\underline{A}}_i^{II}\right)^{-1}\underline{\underline{A}}_i^{I\Sigma}$ sin realizar comunicación alguna. Al terminar de calcular las matrices se avisa al nodo maestro de la finalización de los cálculos.

E) Mediante la comunicación de vectores del tamaño del número de nodos de la frontera interior entre el nodo maestro y los objetos *FEM2D Rectángulos*, se prepara todo lo necesario para empezar el método de gradiente conjugado y resolver el sistema lineal virtual $\left[\sum_{i=1}^{E}\underline{\underline{S}}_i\right]\underline{u_\Sigma} = \left[\sum_{i=1}^{E}\underline{b}_i\right]$.

F) Para usar el método de gradiente conjugado, se transmite un vector del tamaño del número de nodos de la frontera interior para que en cada objeto se realicen las operaciones pertinentes y resolver así el sistema algebraico asociado, esta comunicación se realiza de ida y vuelta entre el nodo maestro y los objetos *FEM2D Rectángulos* tantas veces como iteraciones haga el método. Resolviendo con esto los nodos de la frontera interior $\underline{u_{\Sigma_i}}$.

G) Al término de las iteraciones se pasa la solución $\underline{u_{\Sigma_i}}$ de los nodos de la frontera interior que pertenecen a cada subdominio dentro de cada objeto *FEM2D Rectángulos* para que se resuelvan los nodos interiores $\underline{u_{I_i}} = \left(\underline{\underline{A}}_i^{II}\right)^{-1}\left(\underline{b_{I_i}} - \underline{\underline{A}}_i^{I\Sigma}\underline{u_{\Sigma_i}}\right)$, sin realizar comunicación alguna en el proceso, al concluir se avisa al nodo maestro de ello.

I) El nodo maestro mediante un último mensaje avisa que se concluya el programa, terminado así el esquema maestro-esclavo.

Del algoritmo descrito anteriormente hay que destacar la sincronía entre el nodo maestro y los objetos *FEM2D Rectángulos* contenidos en los nodos esclavos, esto es patente en las actividades realizadas en los incisos A, B y C, estas consumen una parte no significativa del tiempo de cálculo.

Una parte importante del tiempo de cálculo es consumida en la generación de las matrices locales descritas en el inciso D que se realizan de forma independiente en cada nodo esclavo, esta es muy sensible a la discretización particular del dominio usado en el problema.

Los incisos E y F del algoritmo consumen la mayor parte del tiempo total de ejecución al resolver el sistema lineal que dará la solución a los nodos de la frontera interior. La resolución de los nodos interiores planteada en el inciso

G consume muy poco tiempo de ejecución, ya que sólo se realiza una serie de cálculos locales previa transmisión del vector que contiene la solución a los nodos de la frontera interior.

Este algoritmo es altamente paralelizable ya que los nodos esclavos están la mayor parte del tiempo ocupados y la fracción serial del algoritmo está principalmente en las actividades que realiza el nodo maestro, estas nunca podrán ser eliminadas del todo pero consumirá menos tiempo del algoritmo conforme se haga más fina la malla en la descomposición del dominio.

Para resolver la Ec. (3.1), usando 513×513 nodos (igual al ejemplo de *FEM2D Rectángulos* secuencial), en la cual se toma una partición rectangular gruesa de 4×4 subdominios y cada subdominio se descompone en 128×128 subdominios.

Usando para los cálculos en un procesador el equipo secuencial y para la parte paralela el cluster heterogéneo resolviendo por el método de gradiente conjugado sin precondicionador, la solución se encontró en 159 iteraciones obteniendo los siguientes valores:

Procesadores	Tiempo	Factor de Aceleración	Eficiencia	Fracción Serial
1	2943 seg.			
2	2505 seg.	1.17485	0.58742	0.70234
3	1295 seg.	2.27258	0.75752	0.16004
4	1007 seg.	2.92254	0.73063	0.12289
5	671 seg.	4.38599	0.87719	0.03499
6	671 seg.	4.38599	0.73099	0.07359
7	497seg.	5.92152	0.84593	0.03035
8	497 seg.	5.92152	0.74019	0.05014
9	359 seg.	8.19777	0.91086	0.01223
10	359 seg.	8.19777	0.81977	0.02442
11	359 seg.	8.19777	0.74525	0.03441
12	359 seg.	8.19777	0.68314	0.04216
13	359 seg.	8.19777	0.63059	0.04881
14	359 seg.	8.19777	0.58555	0.05444
15	359 seg.	8.19777	0.54651	0.05926
16	359 seg.	8.19777	0.51236	0.06344
17	188 seg	15.65425	0.92083	0.00537

Estos resultados pueden ser apreciados mejor de manera gráfica como se muestra a continuación:

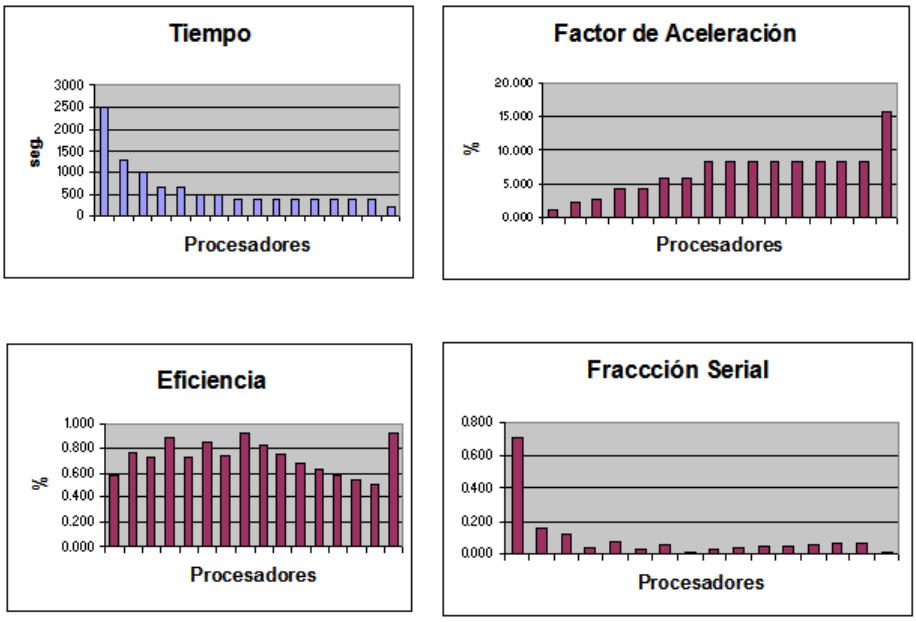


Figura 8: Métricas de desempeño de 2 a 17 procesadores

Primeramente notemos que existe mal balance de cargas. La descomposición adecuada del dominio para tener un buen balanceo de cargas se logra cuando se descompone en $n \times m$ nodos en la partición gruesa, generándose $n*m$ subdominios y si se trabaja con P procesadores (1 para el nodo maestro y $P-1$ para los nodos esclavos), entonces el balance de cargas adecuado será cuando $(P-1) \mid (n*m)$.

En nuestro caso se logra un buen balanceo de cargas cuando se tienen $2, 3, 5, 9, 17$ procesadores, cuyas métricas de desempeño se muestran a continuación:

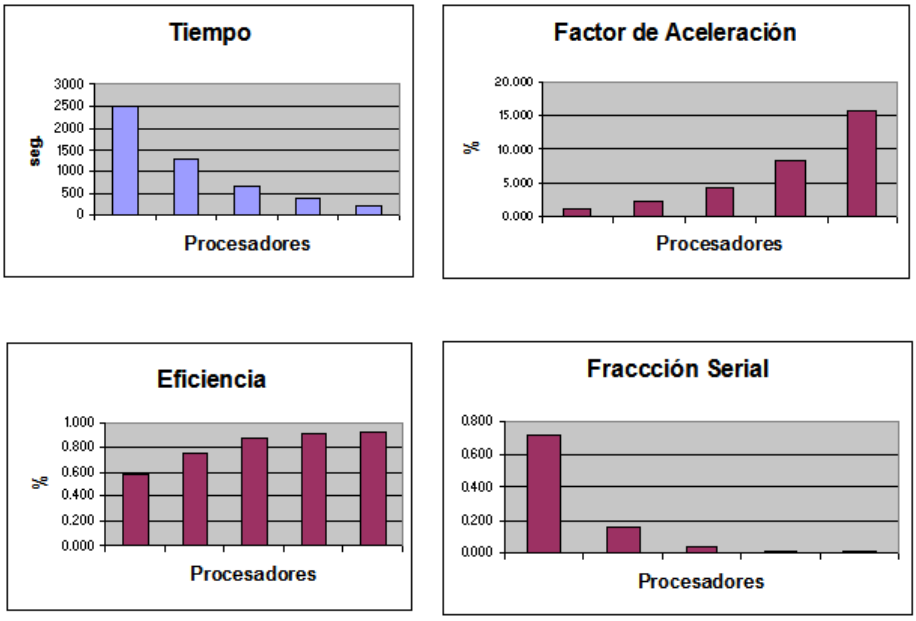


Figura 9: Métricas de desempeño mostrando sólo cuando las cargas están bien balanceadas (2, 3, 5, 9 y 17 procesadores).

En cuanto a las métricas de desempeño, obtenemos que el factor de aceleración en el caso ideal debería de aumentar de forma lineal al aumento del número de procesadores, que en nuestro caso no es lineal pero cumple bien este hecho si están balanceadas las cargas de trabajo.

El valor de la eficiencia deberá ser cercano a uno cuando el Hardware es usado de manera eficiente, como es en nuestro caso cuando se tiene un procesador por cada subdominio.

Y en la fracción serial su valor debiera de tender a cero en el caso ideal, siendo este nuestro caso si están balanceadas las cargas de trabajo, de aquí se puede concluir que la granularidad del problema es gruesa, es decir, no existe una sobrecarga en los procesos de comunicación siendo el cluster una buena herramienta de trabajo para este tipo de problemas.

Finalmente las posibles mejoras de eficiencia para el método de subestructuración en paralelo para disminuir el tiempo de ejecución pueden ser:

- Balanceo de cargas de trabajo homogéneo.
- Al compilar los códigos usar directivas de optimización.

- Usar bibliotecas que optimizan las operaciones en el manejo de los elementos de la matriz usando punteros en las matrices densas o bandadas.
- El cálculo de las matrices que participan en el complemento de Schur pueden ser obtenidas en paralelo.

3.5 Método de Subestructuración en Paralelo Precondicionado

En este método a diferencia del método de subestructuración paralelo, se agrega un precondicionador al método de gradiente conjugado precondicionado visto en la sección (11.2.6) a la hora de resolver el sistema algebraico asociado al método de descomposición de dominio.

En este caso por el mal condicionamiento de la matriz, los precondicionadores a posteriori no ofrecen una ventaja real a la hora de solucionar el sistema lineal algebraico. Es por ello que usaremos los precondicionadores a priori vistos en la sección (2.2.1). Estos son más particulares y su construcción depende del proceso que origina el sistema lineal algebraico.

Existe una amplia gama de este tipo de precondicionadores, pero son específicos al método de descomposición de dominio usado. Para el método de subestructuración usaremos el derivado de la matriz de rigidez, este no es el precondicionador óptimo para este problema, pero para fines demostrativos nos basta.

La implementación de los métodos a priori, requieren de más trabajo tanto en la fase de construcción como en la parte de su aplicación, la gran ventaja de este tipo de precondicionadores es que pueden ser óptimos, 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.

Por ejemplo, al resolver la Ec. (3.1) usando 513×513 nodos en la cual se toma una partición rectangular gruesa de 4×4 subdominios y cada subdominio se descompone en 128×128 subdominios.

Usando para el cálculo en un procesador el equipo secuencial y para la parte paralela el cluster heterogéneo resolviendo por el método de gradiente conjugado con precondicionador la solución se encontró en 79 iteraciones (una mejora en promedio cercana al 50 % con respecto a no usar precondicionador 159 iteraciones) obteniendo los siguientes valores:

Procesadores	Tiempo	Factor de Aceleración	Eficiencia	Fracción Serial
1	2740 seg.			
2	2340 seg.	1.17094	0.58547	0.70802
3	1204 seg.	2.27574	0.75858	0.15912
4	974 seg.	2.81314	0.70328	0.14063
5	626 seg.	4.37699	0.87539	0.03558
6	626 seg.	4.37699	0.72949	0.07416
7	475 seg.	5.76842	0.82406	0.03558
8	475 seg.	5.76842	0.72105	0.05526
9	334 seg.	8.20359	0.91151	0.01213
10	334 seg.	8.20359	0.82035	0.02433
11	334 seg.	8.20359	0.74578	0.03408
12	334 seg.	8.20359	0.68363	0.04207
13	334 seg.	8.20359	0.63104	0.04872
14	334 seg.	8.20359	0.58597	0.05435
15	334 seg.	8.20359	0.54690	0.05917
16	334 seg.	8.20359	0.51272	0.06335
17	173 seg.	15.83815	0.93165	0.00458

Estos resultados pueden ser apreciados mejor de manera gráfica como se muestra a continuación:

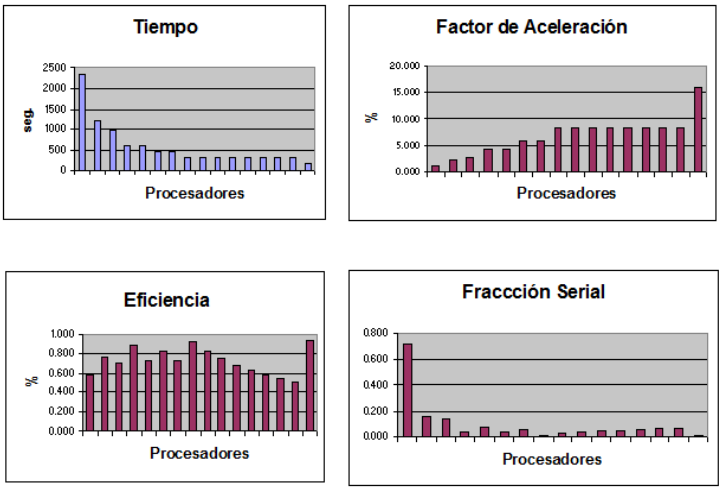


Figura 10: Métricas de desempeño de 2 a 17 procesadores

De las métricas de desempeño, se observa que el factor de aceleración, en el caso ideal debería de aumentar de forma lineal al aumentar el número de procesadores, que en nuestro caso no es lineal pero cumple bien este hecho sí está balanceada las cargas de trabajo.

En la eficiencia su valor deberá ser cercano a uno cuando el Hardware es usado de manera eficiente, como es en nuestro caso cuando se tiene un procesador por cada subdominio. Y en la fracción serial su valor debiera de tender a cero en el caso ideal, siendo este nuestro caso si están balanceadas las cargas de trabajo.

En este ejemplo, como en el caso sin precondicionador el mal balanceo de cargas está presente y es cualitativamente igual, para este ejemplo se logra un buen balanceo de cargas cuando se tienen $2, 3, 5, 9, 17$ procesadores, cuyas métricas de desempeño se muestran a continuación:

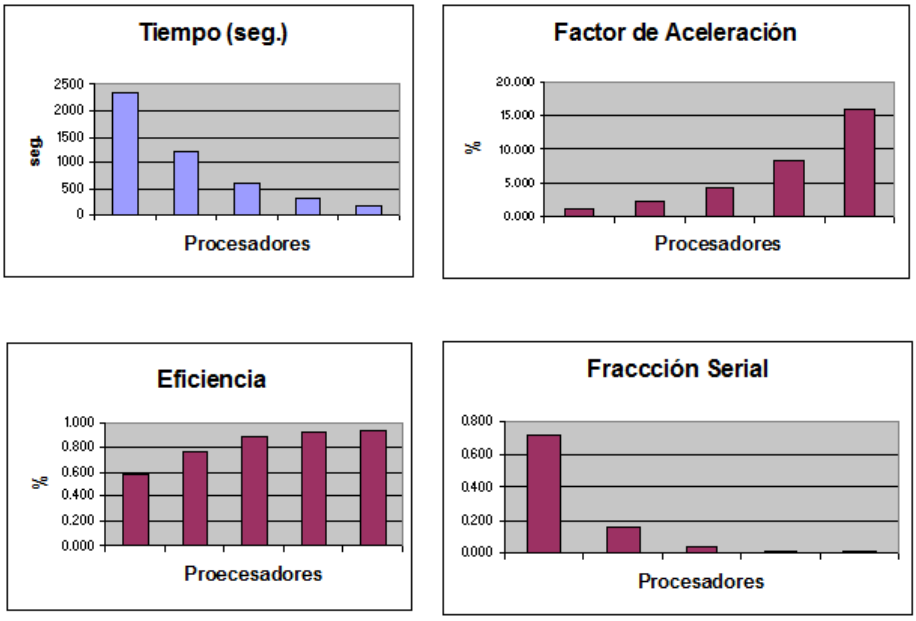


Figura 11: Métricas de desempeño mostrando sólo cuando las cargas están bien balanceadas (2, 3, 5, 9 y 17 procesadores).

Las mismas mejoras de eficiencia que para el método de subestructuración en paralelo son aplicables a este método, adicionalmente:

- Usar el mejor precondicionador a priori disponible para el pro-

blema en particular, ya que esto disminuye sustancialmente el número de iteraciones (hasta en un orden de magnitud cuando el precondicionador es óptimo).

3.5.1 Procedimiento General para Programar el Método

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 subestructuración que no es un problema trivial, debemos empezar seleccionado 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 para llenado de las matrices, ensamble en las matrices y solución del sistema lineal virtual $\left[\sum_{i=1}^{E} \underline{\underline{S}}_i\right] \underline{u_\Sigma} = \left[\sum_{i=1}^{E} \underline{b}_i\right]$ 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 *2D o 3D*, pero lo recomendable es iniciar en *2D* 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:

$$\begin{aligned} -\nabla^2 u &= f_\Omega \text{ en } \Omega \\ u &= g_{\partial\Omega} \text{ en } \partial\Omega. \end{aligned} \tag{3.7}$$

b) Generar la matriz de carga 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 de carga 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 *2D* una malla gruesa de *2x2* y una fina de *2x2* nodos y para *3D* una malla gruesa de *3x3x3* nodos y una fina de *3x3x3*- ahora podemos hacer el ensamble local $\underline{\underline{A}}_i^{\Sigma\Sigma}, \underline{\underline{A}}_i^{\Sigma I}, \underline{\underline{A}}_i^{I\Sigma}$ y $\underline{\underline{A}}_i^{II}$ y formar el global virtual $\left[\sum_{i=1}^{E} \underline{\underline{S}}_i\right] \underline{u_\Sigma} = \left[\sum_{i=1}^{E} \underline{b}_i\right]$ y compararla con la calculada por nosotros

e) Si es correcto el ensambla ahora podemos variar las dimensiones de nuestro dominio y la cantidad de nodos usados en la descretización -en *2D* ahora podemos usar una malla del tipo *2x2* y *3x2 ó 2x2 y 2x3*, 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* y *3x3* nodos, en *3D* será de *3x3x3* y *3x3x3* nodos-

f) Regresando a la malla mínima aplicable a la dimensión del problema, ahora solucionar el sistema virtual $\left[\sum_{i=1}^{E} \underline{\underline{S}}_i\right] \underline{u_\Sigma} = \left[\sum_{i=1}^{E} \underline{b}_i\right]$ 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 $\left[\sum_{i=1}^{E} \underline{\underline{S}}_i\right]$

g) Si contamos con la solución analítica de alguna ecuación, ahora podemos calcular la solución numérica por el método de subestructuración 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 de carga sea bien generada para tener certeza de que todo funcionará

4 Análisis de Rendimiento y Conclusiones

Uno de los grandes retos del área de cómputo científico es poder analizar a priori una serie de consideraciones dictadas por factores externos al problema de interés que repercuten directamente en la forma de solucionar el problema, estas consideraciones influirán de manera decisiva en la implementación computacional de la solución numérica. Algunas de estas consideraciones son:

- Número de Procesadores Disponibles
- Tamaño y Tipo de Partición del Dominio
- Tiempo de Ejecución Predeterminado

Siendo común que ellas interactúan entre sí, de forma tal que normalmente el encargado de la implementación computacional de la solución numérica tiene además de las complicaciones técnicas propias de la solución, el conciliarlas con dichas consideraciones.

Esto deja al implementador de la solución numérica con pocos grados de libertad para hacer de la aplicación computacional una herramienta eficiente y flexible que cumpla con los lineamientos establecidos a priori y permita también que esta sea adaptable a futuros cambios de especificaciones -algo común en ciencia e ingeniería-.

En este capítulo haremos el análisis de los factores que merman el rendimiento de la aplicación y veremos algunas formas de evitarlo, haremos una detallada descripción de las comunicaciones entre el nodo principal y los nodos esclavos, la afectación en el rendimiento al aumentar el número de subdominios en la descomposición y detallaremos algunas consideraciones generales para aumentar el rendimiento computacional. También daremos las conclusiones generales a este trabajo y veremos las diversas ramificaciones que se pueden hacer en trabajos futuros.

4.1 Análisis de Comunicaciones

Para hacer un análisis de las comunicaciones entre el nodo principal y los nodos esclavos en el método de subestructuración es necesario conocer qué se transmite y su tamaño, es por ello detallaremos en la medida de lo posible las comunicaciones existentes (hay que hacer mención que entre los nodos esclavos no hay comunicación alguna).

Tomando la descripción del algoritmo detallado en el método de subestructuración en paralelo visto en la sección (3.4), suponiendo una partición del dominio Ω en $n \times m$ y $p \times q$ subdominios, las comunicaciones correspondientes a cada inciso son:

A) El nodo maestro transmite 4 coordenadas (en dos dimensiones) correspondientes a la delimitación del subdominio.

B) $2{*}p{*}q$ coordenadas transmite cada objeto *FEM2D Rectángulos* al nodo maestro.

C) A lo más $n * m * 2 * p * q$ coordenadas son las de los nodos de la frontera interior, y sólo aquellas correspondientes a cada subdominio son transmitidas por el nodo maestro a los objetos *FEM2D Rectángulos* siendo estas a lo más $2 * p * q$ coordenadas.

D) Sólo se envía un aviso de la conclusión del cálculo de las matrices.

E) A lo más $2 * p * q$ coordenadas son transmitidas a los objetos *FEM2D Rectángulos* desde el nodo maestro y los nodos esclavos transmiten al nodo maestro esa misma cantidad de información.

F) A lo más $2 * p * q$ coordenadas son transmitidas a los objetos *FEM2D Rectángulos* en los nodos esclavos y éstos retornan un número igual al nodo maestro por iteración del método de gradiente conjugado.

G) A lo más $2 * p * q$ valores de la solución de la frontera interior son transmitidas a los objetos *FEM2D Rectángulos* desde el nodo maestro y cada objeto transmite un único aviso de terminación.

I) El nodo maestro manda un aviso a cada objeto *FEM2D Rectángulos* para concluir con el esquema.

La transmisión se realiza mediante paso de arreglos de enteros y números de punto flotante que varían de longitud pero siempre son cantidades pequeñas de estos y se transmiten en forma de bloque, por ello las comunicaciones son eficientes.

4.2 Afectación del Rendimiento al Refinar la Descomposición

Una parte fundamental a considerar es la afectación del rendimiento al aumentar el número de subdominios en descomposición, ya que el complemento de Schur local $\underline{\underline{S}}_i = \underline{\underline{A}}_i^{\Sigma\Sigma} - \underline{\underline{A}}_i^{\Sigma I}\left(\underline{\underline{A}}_i^{II}\right)^{-1}\underline{\underline{A}}_i^{I\Sigma}$ involucra el generar las matrices $\underline{\underline{A}}_i^{II}, \underline{\underline{A}}_i^{\Sigma\Sigma}, \underline{\underline{A}}_i^{\Sigma I}, \underline{\underline{A}}_i^{I\Sigma}$ y calcular de alguna forma $\left(\underline{\underline{A}}_i^{II}\right)^{-1}$.

Si el número de nodos interiores en el subdominio es grande entonces obtener la matriz anterior será muy costoso computacionalmente, como se ha mostrado en el transcurso de las últimas secciones del capítulo anterior.

Al aumentar el número de subdominios en una descomposición particular, se garantiza que las matrices a generar y calcular sean cada vez más pequeñas y fáciles de manejar.

Pero hay un límite al aumento del número de subdominio en cuanto a la eficiencia de ejecución, este cuello de botella es generado por el esquema maestro-esclavo y es reflejado por un aumento del tiempo de ejecución al aumentar el número de subdominios en una configuración de Hardware particular.

Esto se debe a que en el esquema maestro-esclavo, el nodo maestro deberá de atender todas las peticiones hechas por cada uno de los nodos esclavos, esto toma especial relevancia cuando todos o casi todos los nodos esclavos compiten por ser atendidos por el nodo maestro.

Por ello se recomienda implementar este esquema en un cluster heterogéneo en donde el nodo maestro sea más poderoso computacionalmente que los nodos esclavos. Si a éste 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 esclavos inexorablemente. Por ello hay que ser cuidadosos en cuanto al número de nodos esclavos que se usan en la implementación en tiempo de ejecución versus el rendimiento general del sistema al aumentar estos.

4.3 Descomposición Óptima para un Equipo Paralelo Dado.

Otra cosa por considerar es que normalmente se tiene a disposición un número fijo de procesadores, con los cuales hay que trabajar, así que es necesario

encontrar la descomposición adecuada para esta cantidad de procesadores. No es posible hacer un análisis exhaustivo, pero mediante pruebas podemos determinar cual es la mejor descomposición en base al tiempo de ejecución.

Para el análisis, consideremos pruebas con $3, 4, 5$ y 6 procesadores y veremos cual es la descomposición más adecuada para esta cantidad de procesadores tomando como referencia el resolver la Ec. (3.1), usando 513×513 nodos.

Usando para estos cálculos el cluster homogéneo, al resolver por el método de gradiente conjugado precondicionado para cada descomposición se obtuvieron los siguientes resultados:

Partición	Tiempo en 3 Procesadores	Tiempo en 4 Procesadores	Tiempo en 5 Procesadores	Tiempo en 6 Procesadores
2×2 y 256×256	2576 seg.	2084 seg.	1338 seg.	–
4×4 y 128×128	1324 seg.	1071 seg.	688 seg.	688 seg.
8×8 y 64×64	779 seg.	630 seg.	405 seg.	405 seg.
16×16 y 32×32	485 seg.	391 seg.	251 seg.	251 seg.

De estas pruebas se observa que el mal balance de cargas es reflejado en los tiempos de ejecución, pese a contar con más procesadores no hay una disminución del tiempo de ejecución.

Ahora para las mismas descomposiciones, usando el cluster heterogéneo para cada descomposición se obtuvieron los siguientes resultados:

Partición	Tiempo en 3 Procesadores	Tiempo en 4 Procesadores	Tiempo en 5 Procesadores	Tiempo en 6 Procesadores
2×2 y 256×256	2342 seg.	1895 seg.	1217 seg.	–
4×4 y 128×128	1204 seg.	974 seg.	626 seg.	626 seg.
8×8 y 64×64	709 seg.	573 seg.	369 seg.	369 seg.
16×16 y 32×32	441 seg.	356 seg.	229 seg.	229 seg.

Primeramente hay que destacar que los nodos esclavos de ambos clusters son comparables en poder de cómputo, pero aquí lo que hace la diferencia es que el nodo maestro del segundo ejemplo tiene mayor rendimiento. Es por ello que al disminuir la fracción serial del problema y atender mejor las comunicaciones que se generan en el esquema maestro-esclavo con todos los objetos *FEM2D Rectángulos* creados en cada uno de los nodos esclavos mejora sustancialmente el tiempo de ejecución.

En ambas pruebas el mal balanceo de cargas es un factor determinante del rendimiento, sin embargo el usó de un cluster en el que el nodo maestro sea más poderoso computacionalmente, hará que se tenga una mejora sustancial en el rendimiento.

Para evitar el mal balance de cargas se debe de asignar a cada nodo esclavo una cantidad de subdominios igual. La asignación mínima del número de nodos por subdominio queda sujeta a la velocidad de los procesadores involucrados para disminuir en lo posible los tiempos muertos, obteniendo así el máximo rendimiento.

La asignación máxima del número de nodos por subdominio a cada nodo esclavo, estará en función de la memoria que consuman las matrices que contienen cada uno de los objetos de *FEM2D Rectángulos*. La administración de ellos y las comunicaciones no son un factor limitante y por ello se pueden despreciar.

Descomposición Fina del Dominio Supongamos ahora que deseamos resolver el problema de una descomposición fina del dominio Ω en 65537×65537 nodos, este tipo de problemas surgen cotidianamente en la resolución de sistemas reales y las opciones para implantarlo en un equipo paralelo son viables, existen y son actualmente usadas. Aquí las opciones de partición del dominio son muchas y variadas, y la variante seleccionada dependerá fuertemente de las características del equipo de cómputo paralelo del que se disponga, es decir, si suponemos que una descomposición de 1000×1000 nodos en un subdominio consume 1 GB de RAM y el consumo de memoria crece linealmente con el número de nodos, entonces algunas posibles descomposiciones son:

Procesadores	Descomposición	Nodos Subdominio	RAM Mínimo
5	2×2 y 32768×32768	32768×32768	≈33.0 GB
257	16×16 y 4096×4096	4096×4096	≈4.0 GB
1025	32×32 y 2048×2048	2048×2048	≈2.0 GB
4097	64×64 y 1024×1024	1024×1024	≈1.0 GB

Notemos que para las primeras particiones, el consumo de RAM es excesivo y en las últimas particiones la cantidad de procesadores en paralelo necesarios es grande (pero ya de uso común en nuestros días). Como en general, contar con equipos paralelos de ese tamaño es en extremo difícil, ¿es posible resolver este tipo de problemas con una cantidad de procesadores

menor al número sugerido y donde cada uno de ellos tiene una memoria muy por debajo de lo sugerido?, la respuesta es sí.

Primero, notemos que al considerar una descomposición del tipo 64×64 y 1024×1024 subdominios requerimos aproximadamente 1.0 GB de RAM mínimo por nodo, si suponemos que sólo tenemos unos cuantos procesadores con memoria limitada (digamos 2 GB), entonces no es posible tener en memoria de manera conjunta a las matrices generadas por el método.

Una de las grandes ventajas de los métodos de descomposición de dominio es que los subdominios son en principio independientes entre sí y que sólo están acoplados a través de la solución en la interfaz de los subdominios que es desconocida.

Como sólo requerimos tener en memoria la información de la frontera interior, es posible bajar a disco duro todas las matrices y datos complementarios (que consumen el 99% de la memoria del objeto *FEM2D Rectángulos*) generados por cada subdominio que no se requieran en ese instante para la operación del esquema maestro-esclavo.

Recuperando del disco duro solamente los datos del subdominio a usarse en ese momento (ya que el proceso realizado por el nodo maestro es secuencial) y manteniéndolos en memoria por el tiempo mínimo necesario. Así, es posible resolver un problema de una descomposición fina, usando una cantidad de procesadores fija y con una cantidad de memoria muy limitada por procesador.

En un caso extremo, la implementación para resolver un dominio Ω descompuesto en un número de nodos grande es posible implementarla usando sólo dos procesos en un procesador, uno para el proceso maestro y otro para el proceso esclavo, en donde el proceso esclavo construiría las matrices necesarias por cada subdominio y las guardaría en disco duro, recuperándolas conforme el proceso del nodo maestro lo requiera. Nótese que la descomposición del dominio Ω estará sujeta a que cada subdominio Ω_i sea soportado en memoria conjuntamente con los procesos maestro y esclavo.

De esta forma es posible resolver un problema de gran envergadura usando recursos computacionales muy limitados, sacrificando velocidad de procesamiento en aras de poder resolver el problema. Está es una de las grandes ventajas de los métodos de descomposición de dominio con respecto a los otros métodos de discretización tipo diferencias finitas y elemento finito.

El ejemplo anterior nos da una buena idea de las limitantes que existen en la resolución de problemas con dominios que tienen una descomposición fina y nos pone de manifiesto las características mínimas necesarias del equipo

paralelo para soportar dicha implantación.

4.4 Consideraciones para Aumentar el Rendimiento

Algunas consideraciones generales para aumentar el rendimiento son:

a) Balanceo de cargas de trabajo homogéneo, si se descompone en $n \times m$ subdominios en la partición gruesa se y si se trabaja con P procesadores, entonces el balance de cargas adecuado será cuando $(P-1) \mid (n*m)$.), ya que de no hacerlo el rendimiento se ve degradado notoriamente.

b) Usar el mejor precondicionador a priori disponible para el problema en particular, ya que esto disminuye sustancialmente el número de iteraciones (hasta en un orden de magnitud cuando el precondicionador es óptimo).

c) Usar la biblioteca Lapack++ de licencia GNU que optimiza las operaciones en el manejo de los elementos de la matriz usando punteros en donde se usan matrices densas y bandadas.

d) Si se cuenta con un equipo en que cada nodo del Cluster tenga más de un procesador, usar bibliotecas (PLAPACK por ejemplo) que permitan paralelizar mediante el uso de procesos con memoria compartida, Pipeline o hilos, las operaciones que involucren a vectores y matrices; como una forma de mejorar el rendimiento del programa.

e) Siempre usar al momento de compilar los códigos, directivas de optimización (estas ofrecen mejoras de rendimiento en la ejecución de 30% aproximadamente en las pruebas realizadas), pero existen algunos compiladores con optimizaciones específicas para ciertos procesadores (Intel compiler para 32 y 64 bits) que pueden mejorar aún más este rendimiento (más de 50%).

Todas estas mejoras pueden ser mayores si se usa un nodo maestro del mayor poder computacional posible aunado a una red en el cluster de de 1 Gb o mayor y de ser posible de baja latencia, si bien las comunicaciones son pocas, estas pueden generar un cuello de botella sustancial.

Por otro lado, hay que hacer notar que es posible hacer usó de múltiples etapas de paralelización que pueden realizarse al momento de implantar el

método de descomposición de dominio, estas se pueden implementar conforme se necesite eficiencia adicional, pero implica una cuidadosa planeación al momento de hacer el análisis y diseño de la aplicación y una inversión cuantiosa en tiempo para implantarse en su totalidad, estas etapas se describen a continuación:

- El propio método de descomposición de dominio ofrece un tipo particular y eficiente de paralelización al permitir dividir el dominio en múltiples subdominios independientes entre sí, interconectados sólo por la frontera interior.
- A nivel subdominio otra paralelización es posible, específicamente en el llenado de las matrices involucradas en el método de descomposición de dominio, ya que varias de ellas son independientes.
- A nivel de los cálculos, entre matrices y vectores involucrados en el método también se pueden paralelizar de manera muy eficiente.
- A nivel del compilador es posible generar el ejecutable usando esquemas de paralelización automático y opciones para eficientizar la ejecución.

Por lo anterior es posible usar una serie de estrategias que permitan realizar estas etapas de paralelización de manera cooperativa y aumentar la eficiencia en una factor muy significativo, pero esto implica una programación particular para cada una de las etapas y poder distribuir las tareas paralelas de cada etapa en uno o más procesadores distintos a los de las otras etapas.

Notemos finalmente que si se toma el programa de elemento finito y se paraleliza usando sólo directivas de compilación, el aumento del rendimiento es notorio pero este se merma rápidamente al aumentar del número de nodos (esto es debido al aumento en las comunicaciones para mantener y acceder a la matriz del sistema algebraico asociado al método). Pero es aún más notorio cuando el método de descomposición de dominio serial usando las mismas directivas de compilación se paraleliza (sin existir merma al aumentar el número de nodos siempre y cuando las matrices generadas estén el la memoria local del procesador).

Esto se debe a que en el método de elemento finito la matriz estará distribuida por todos los nodos usando memoria distribuida, esto es muy costoso en tiempo de cómputo ya que su manipulación requiere de múltiples comunicaciones entre los procesadores, en cambio el el método de descomposición

de dominio ya están distribuidas las matrices en los nodos y las operaciones sólo involucran transmisión de un vector entre ellos, minimizando las comunicaciones entre procesadores.

Pero aún estos rendimientos son pobres con respecto a los obtenidos al usar el método de descomposición de dominio paralelizado conjuntamente con bi-bliotecas para manejo de matrices densas y dispersas en equipos con nodos que cuenten con más de un procesador, en donde mediante el uso de memoria compartida se pueden usar el resto de los procesadores dentro del nodo para efectuar en paralelo las operaciones en donde estén involucradas las matrices.

4.5 Conclusiones Generales

A lo largo del presente trabajo se ha mostrado que al aplicar métodos de descomposición de dominio conjuntamente con métodos de paralelización es posible resolver una gama más amplia de problemas de ciencias e ingeniería que mediante las técnicas tradicionales del tipo diferencias finitas y elemento finito.

La resolución del sistema algebraico asociado es más eficiente cuando se hace uso de precondicionadores a priori conjuntamente con el método de gradiente conjugado precondicionado al implantar la solución por el método de descomposición de dominio.

Y haciendo uso del análisis de rendimiento, es posible encontrar la manera de balancear las cargas de trabajo que son generadas por las múltiples discretizaciones que pueden obtenerse para la resolución de un problema particular, minimizando en la medida de lo posible el tiempo de ejecución y adaptándolo a la arquitectura paralela disponible, esto es especialmente útil cuando el sistema a trabajar es de tamaño considerable.

Adicionalmente se vieron los alcances y limitaciones de esta metodología, permitiendo tener cotas tanto para conocer las diversas descomposiciones que es posible generar para un número de procesadores fijo, como para conocer el número de procesadores necesarios en la resolución de un problema particular. También se vio una forma de usar los métodos de descomposición de dominio en casos extremos en donde una partición muy fina, genera un problema de gran envergadura y como resolver este usando recursos computacionales muy limitados, sacrificando velocidad de procesamiento en aras de poder resolver el problema.

Así, podemos afirmar de manera categórica que conjuntando los métodos

de descomposición de dominio, la programación orientada a objetos y esquemas de paralelización que usan el paso de mensajes, es posible construir aplicaciones que coadyuven a la solución de problemas en dos o más dimensiones concomitantes en ciencia e ingeniería, los cuales pueden ser de tamaño considerable.

Las aplicaciones desarrolladas bajo este paradigma serán eficientes, flexibles y escalables; a la vez que son abiertas a nuevas tecnologías y desarrollos computacionales y al ser implantados en clusters, permiten una codificación ordenada y robusta, dando con ello una alta eficiencia en la adaptación del código a nuevos requerimientos, como en la ejecución del mismo.

De forma tal que esta metodología permite tener a disposición de quien lo requiera, una gama de herramientas flexibles y escalables para coadyuvar de forma eficiente y adaptable a la solución de problemas en medios continuos de forma sistemática.

5 Apéndice A: Sistemas Continuos y sus Modelos

Los fundamentos de la física macroscópica nos proporciona la 'teoría de los medios continuos'. En este capítulo, en base a ella se introduce una formulación clara, general y sencilla de los modelos matemáticos de los sistemas continuos. Esta formulación es tan sencilla y tan general, que los modelos básicos de sistemas tan complicados y diversos como la atmósfera, los océanos, los yacimientos petroleros, o los geotérmicos, se derivan por medio de la aplicación repetida de una sola ecuación diferencial: 'la ecuación diferencial de balance'.

Dicha formulación también es muy clara, pues en el modelo general no hay ninguna ambigüedad; en particular, todas las variables y parámetros que intervienen en él, estan definidos de manera unívoca. En realidad, este modelo general de los sistemas continuos constituye una realización extraordinaria de los paradigmas del pensamiento matemático. El descubrimiento del hecho de que los modelos matemáticos de los sistemas continuos, independientemente de su naturaleza y propiedades intrínsecas, pueden formularse por medio de balances, cuya idea básica no difiere mucho de los balances de la contabilidad financiera, fue el resultado de un largo proceso de perfeccionamiento en el que concurrieron una multitud de mentes brillantes.

5.1 Los Modelos

Un modelo de un sistema es un sustituto de cuyo comportamiento es posible derivar el correspondiente al sistema original. Los modelos matemáticos, en la actualidad, son los utilizados con mayor frecuencia y también los más versátiles. En las aplicaciones específicas están constituidos por programas de cómputo cuya aplicación y adaptación a cambios de las propiedades de los sistemas es relativamente fácil. También, sus bases y las metodologías que utilizan son de gran generalidad, por lo que es posible construirlos para situaciones y sistemas muy diversos.

Los modelos matemáticos son entes en los que se integran los conocimientos científicos y tecnológicos, con los que se construyen programas de cómputo que se implementan con medios computacionales. En la actualidad, la simulación numérica permite estudiar sistemas complejos y fenómenos na-turales que sería muy costoso, peligroso o incluso imposible de estudiar

por experimentación directa. En esta perspectiva la significación de los modelos matemáticos en ciencias e ingeniería es clara, porqué la modelación matemática constituye el método más efectivo de predecir el comportamiento de los diversos sistemas de interés. En nuestro país, ellos son usados ampliamente en la industria petrolera, en las ciencias y la ingeniería del agua y en muchas otras.

5.1.1 Física Microscópica y Física Macroscópica

La materia, cuando se le observa en el ámbito ultramicroscópico, está formada por moléculas y átomos. Estos a su vez, por partículas aún más pequeñas como los protones, neutrones y electrones. La predicción del comportamiento de estas partículas es el objeto de estudio de la mecánica cuántica y la física nuclear. Sin embargo, cuando deseamos predecir el comportamiento de sistemas tan grandes como la atmósfera o un yacimiento petrolero, los cuales estan formados por un número extraordinariamente grande de moléculas y átomos, su estudio resulta inaccesible con esos métodos y en cambio el enfoque macroscópico es apropiado.

Por eso en lo que sigue distinguiremos dos enfoques para el estudio de la materia y su movimiento. El primero -el de las moléculas, los átomos y las partículas elementales- es el enfoque microscópico y el segundo es el enfoque macroscópico. Al estudio de la materia con el enfoque macroscópico, se le llama física macroscópica y sus bases teóricas las proporciona la mecánica de los medios continuos.

Cuando se estudia la materia con este último enfoque, se considera que los cuerpos llenan el espacio que ocupan, es decir que no tienen huecos, que es la forma en que los vemos sin el auxilio de un microscopio. Por ejemplo, el agua llena todo el espacio del recipiente donde está contenida. Este enfoque macroscópico está presente en la física clásica. La ciencia ha avanzado y ahora sabemos que la materia está llena de huecos, que nuestros sentidos no perciben y que la energía también está cuantizada. A pesar de que estos dos enfoques para el análisis de los sistemas físicos, el microscópico y el macroscópico, parecen a primera vista conceptualmente contradictorios, ambos son compatibles, y complementarios, y es posible establecer la relación entre ellos utilizando la mecánica estadística.

5.2 Cinemática de los Modelos de Sistemas Continuos

En la teoría de los sistemas continuos, los cuerpos llenan todo el espacio que ocupan. Y en cada punto del espacio físico hay una y solamente una partícula. Así, definimos como sistema continuo a un conjunto de partículas. Aún más, dicho conjunto es un subconjunto del espacio Euclidiano tridimensional. Un cuerpo es un subconjunto de partículas que en cualquier instante dado ocupa un dominio, en el sentido matemático, del espacio físico; es decir, del espacio Euclidiano tridimensional. Denotaremos por $B(t)$ a la región ocupada por el cuerpo $\mathcal{B}$, en el tiempo t, donde t puede ser cualquier número real.

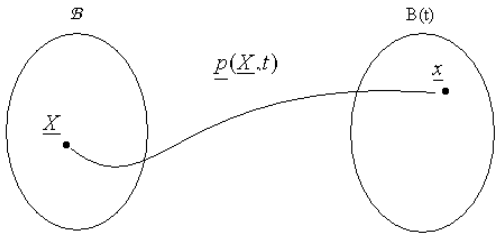


Figura 12: Representación del movimiento de partículas de un cuerpo $\mathcal{B}$, para un tiempo dado.

Frecuentemente, sin embargo, nuestro interés de estudio se limitará a un intervalo finito de tiempo. Dado un cuerpo $\mathcal{B}$, todo subdominio $\tilde{\mathcal{B}} \subset \mathcal{B}$, constituye a su vez otro cuerpo; en tal caso, se dice que $\tilde{\mathcal{B}} \subset \mathcal{B}$ es un subcuerpo de $\mathcal{B}$. De acuerdo con lo mencionado antes, una hipótesis básica de la teoría de los sistemas continuos es que en cualquier tiempo $t \in (-\infty, \infty)$ y en cada punto $x \in \mathcal{B}$ de la región ocupada por el cuerpo, hay una y sólo una partícula del cuerpo. Como en nuestra revisión se incluye no solamente la estática (es decir, los cuerpos en reposo), sino también la dinámica (es decir, los cuerpos en movimiento), un primer problema de la cinemática de los sistemas continuos consiste en establecer un procedimiento para identificar a las partículas cuando están en movimiento en el espacio físico.

Sea $\underline{X} \in \mathcal{B}$, una partícula y $p(\underline{X}, t)$ el vector de la posición que ocupa, en el espacio físico, dicha partícula en el instante t. Una forma, pero no la única, de identificar la partícula $\underline{X}$ es asociándole la posición que ocupa en un instante determinado. Tomaremos en particular el tiempo $t = 0$, en tal caso $\underline{p}(\underline{X}, 0) \equiv \underline{X}$.

A las coordenadas del vector $\underline{X} \equiv (x_1, x_2, x_3)$, se les llama las coordenadas materiales de la partícula. En este caso, las coordenadas materiales de una partícula son las coordenadas del punto del espacio físico que ocupaba la partícula en el tiempo inicial, $t = 0$. Desde luego, el tiempo inicial puede ser cualquier otro, si así se desea. Sea $\mathcal{B}$ el dominio ocupado por un cuerpo en el tiempo inicial, entonces $\underline{X} \in \mathcal{B}$ si y solamente si la partícula $\underline{X}$ es del cuerpo. Es decir, $\mathcal{B}$ caracteriza al cuerpo. Sin embargo, debido al movimiento, la región ocupada por el mismo cambia con el tiempo y será denotada por $\mathcal{B}(t)$.

Formalmente, para cualquier $t \in (-\infty, \infty)$, $\mathcal{B}(t)$ se define por

$$\mathcal{B}(t) \equiv \left\{ \underline{x} \in \mathbb{R}^3 \mid \exists \underline{X} \in \mathcal{B} \quad tal \quad que \quad \underline{x} = \underline{p}(\underline{X}, t) \right\} \tag{5.1}$$

el vector posición $\underline{p}(\underline{X}, t)$ es función del vector tridimensional $\underline{X}$ y del tiempo. Si fijamos el tiempo t, $\underline{p}(\underline{X}, t)$ define una transformación del espacio Euclidiano $\mathbb{R}^3$ en sí mismo y la Ec. (5.1) es equivalente a $\mathcal{B}(t) = \underline{p}(\mathcal{B}, t)$. Una notación utilizada para representar esta familia de funciones es $\underline{p}(\cdot, t)$. De acuerdo a la hipótesis de los sistemas continuos: En cualquier tiempo $t \in (-\infty, \infty)$ y en cada punto $\underline{x} \in \mathcal{B}$ de la región ocupada por el cuerpo hay una y sólo una partícula del cuerpo $\mathcal{B}$ para cada t fijo. Es decir, $\underline{p}(\cdot, t)$ es una función biunívoca, por lo que existe la función inversa $\underline{p}^{-1}(\cdot, t)$.

Si se fija la partícula $\underline{X}$ en la función $\underline{p}(\underline{X}, t)$ y se varía el tiempo t, se obtiene su trayectoria. Esto permite obtener la velocidad de cualquier partícula, la cual es un concepto central en la descripción del movimiento. Ella se define como la derivada con respecto al tiempo de la posición cuando la partícula se mantiene fija. Es decir, es la derivada parcial con respecto al tiempo de la función de posición $\underline{p}(\underline{X}, t)$. Por lo mismo, la velocidad como función de las coordenadas materiales de las partículas, está dada por

$$\underline{\mathrm{V}}(\underline{X}, t) \equiv \frac{\partial \underline{p}}{\partial t}(\underline{X}, t). \tag{5.2}$$

5.2.1 Propiedades Intensivas y sus Representaciones

En lo que sigue consideraremos funciones definidas para cada tiempo, en cada una de las partículas de un sistema continuo. A tales funciones se les llama 'propiedades intensivas'. Las propiedades intensivas pueden ser funciones escalares o funciones vectoriales. Por ejemplo, la velocidad, definida por la Ec. (5.2), es una función vectorial que depende de la partícula $\underline{X}$ y del tiempo t.

Una propiedad intensiva con valores vectoriales es equivalente a tres escalares, correspondientes a cada una de sus tres componentes. Hay dos formas de representar a las propiedades intensivas: la representación Euleriana y la representación Lagrangiana. Los nombres son en honor a los matemáticos Leonard Euler (1707-1783) y Joseph Louis Lagrange (1736-1813), respectivamente. Frecuentemente, el punto de vista Lagrangiano es utilizado en el estudio de los sólidos, mientras que el Euleriano se usa más en el estudio de los fluidos.

Considere una propiedad intensiva escalar, la cual en el tiempo t toma el valor $\phi(\underline{X}, t)$ en la partícula $\underline{X}$. Entonces, de esta manera se define una función $\phi : \mathcal{B} \to \mathbb{R}^1$, para cada $t \in (-\infty, \infty)$ a la que se denomina representación Lagrangiana de la propiedad intensiva considerada. Ahora, sea $\psi(\underline{x}, t)$ el valor que toma esa propiedad en la partícula que ocupa la posición $\underline{x}$, en el tiempo t. En este caso, para cada $t \in (-\infty, \infty)$ se define una función $\psi : \mathcal{B}(t) \to \mathbb{R}^1$ a la cual se denomina representación Euleriana de la función considerada. Estas dos representaciones de una misma propiedad están relacionadas por la siguiente identidad

$$\phi(\underline{X}, t) \equiv \psi(\underline{p}(\underline{X}, t), t). \tag{5.3}$$

Nótese que, aunque ambas representaciones satisfacen la Ec. (5.3), las funciones $\phi(\underline{X}, t)$ y $\psi(\underline{x}, t)$ no son idénticas. Sus argumentos $\underline{X}$ y $\underline{x}$ son vectores tridimensionales (es decir, puntos de $\mathbb{R}^3$); sin embargo, si tomamos $\underline{X} = \underline{x}$, en general

$$\phi(\underline{X}, t) \neq \psi(\underline{X}, t). \tag{5.4}$$

La expresión de la velocidad de una partícula dada por la Ec. (5.2), define a su representación Lagrangiana, por lo que utilizando la Ec. (5.3) es claro que

$$\frac{\partial p}{\partial t}(\underline{X}, t) = \underline{\mathrm{V}}(\underline{X}, t) \equiv \underline{\mathrm{v}}(\underline{p}(\underline{X}, t), t) \tag{5.5}$$

donde $\underline{\mathrm{v}}(\underline{x}, t)$ es la representación Euleriana de la velocidad. Por lo mismo

$$\underline{\mathrm{v}}(\underline{x}, t) \equiv \underline{\mathrm{V}}(\underline{p}^{-1}(\underline{x}, t), t). \tag{5.6}$$

Esta ecuación tiene la interpretación de que la velocidad en el punto $\underline{x}$ del espacio físico, es igual a la velocidad de la partícula que pasa por dicho punto en el instante t. La Ec. (5.6) es un caso particular de la relación

$$\psi(\underline{x}, t) \equiv \phi(\underline{p}^{-1}(\underline{x}, t), t)$$

de validez general, la cual es otra forma de expresar la relación de la Ec. (5.3) que existe entre las dos representaciones de una misma propiedad intensiva.

La derivada parcial con respecto al tiempo de la representación Lagrangiana $\phi(\underline{X}, t)$ de una propiedad intensiva, de acuerdo a la definición de la derivada parcial de una función, es la tasa de cambio con respecto al tiempo que ocurre en una partícula fija. Es decir, si nos montamos en una partícula y medimos a la propiedad intensiva y luego los valores así obtenidos los derivamos con respecto al tiempo, el resultado final es $\frac{\partial \phi(\underline{X},t)}{\partial t}$. En cambio, si $\psi(\underline{x}, t)$ es la representación Euleriana de esa misma propiedad, entonces $\frac{\partial \psi(\underline{x},t)}{\partial t}$ es simplemente la tasa de cambio con respecto al tiempo que ocurre en un punto fijo en el espacio. Tiene interés evaluar la tasa de cambio con respecto al tiempo que ocurre en una partícula fija, cuando se usa la representación Euleriana. Derivando con respecto al tiempo a la identidad de la Ec. (5.3) y la regla de la cadena, se obtiene

$$\frac{\partial \phi(\underline{X}, t)}{\partial t} = \frac{\partial \psi}{\partial t}(\underline{p}(\underline{X}, t), t) + \sum_{i=1}^{3} \frac{\partial \psi}{\partial x_i}(\underline{p}(\underline{X}, t), t) \frac{\partial p_i}{\partial t}(\underline{X}, t). \tag{5.7}$$

Se acostumbra definir el símbolo $\frac{D\psi}{Dt}$ por

$$\frac{D\psi}{Dt} = \frac{\partial \psi}{\partial t} + \sum_{i=1}^{3} v_i \frac{\partial \psi}{\partial x_i} \tag{5.8}$$

o, más brevemente,

$$\frac{D\psi}{Dt} = \frac{\partial \psi}{\partial t} + \underline{\mathrm{v}} \cdot \nabla \psi \tag{5.9}$$

utilizando esta notación, se puede escribir

$$\frac{\partial \phi(\underline{X}, t)}{\partial t} = \frac{D\psi}{Dt}(\underline{p}(\underline{X}, t) \equiv \left(\frac{\partial \psi}{\partial t} + \underline{\mathrm{v}} \cdot \nabla \psi \right) (\underline{p}(\underline{X}, t), t). \tag{5.10}$$

Por ejemplo, la aceleración de una partícula se define como la derivada de la velocidad cuando se mantiene a la partícula fija. Aplicando la Ec. (5.9) se tiene

$$\frac{D\underline{\mathrm{v}}}{Dt} = \frac{\partial \underline{\mathrm{v}}}{\partial t} + \underline{\mathrm{v}} \cdot \nabla \underline{\mathrm{v}} \tag{5.11}$$

una expresión más transparente se obtiene aplicando la Ec. (5.9) a cada una de las componentes de la velocidad. Así, se obtiene

$$\frac{D\mathrm{v}_i}{Dt} = \frac{\partial \mathrm{v}_i}{\partial t} + \underline{\mathrm{v}} \cdot \nabla \mathrm{v}_i. \tag{5.12}$$

Desde luego, la aceleración, en representación Lagrangiana es simplemente

$$\frac{\partial}{\partial t}\underline{\mathrm{V}}(\underline{X},t)=\frac{\partial^2}{\partial t^2}\underline{p}(\underline{X},t). \tag{5.13}$$

5.2.2 Propiedades Extensivas

En la sección anterior se consideraron funciones definidas en las partículas de un cuerpo, más precisamente, funciones que hacen corresponder a cada partícula y cada tiempo un número real, o un vector del espacio Euclidiano tridimensional $\mathbb{R}^3$. En esta, en cambio, empezaremos por considerar funciones que a cada cuerpo $\mathcal{B}$ de un sistema continuo, y a cada tiempo t le asocia un número real o un vector de $\mathbb{R}^3$. A una función de este tipo $\mathbb{E}(\mathcal{B},t)$ se le llama 'propiedad extensiva' cuando está dada por una integral

$$\mathbb{E}(\mathcal{B},t)\equiv\int_{\mathcal{B}(t)}\psi(\underline{x},t)d\underline{\mathrm{x}}. \tag{5.14}$$

Observe que, en tal caso, el integrando define una función $\psi(\underline{x},t)$ y por lo mismo, una propiedad intensiva. En particular, la función $\psi(\underline{x},t)$ es la representación Euleriana de esa propiedad intensiva. Además, la Ec. (5.14) establece una correspondencia biunívoca entre las propiedades extensivas y las intensivas, porqué dada la representación Euleriana $\psi(\underline{x},t)$ de cualquier propiedad intensiva, su integral sobre el dominio ocupado por cualquier cuerpo, define una propiedad extensiva. Finalmente, la notación empleada en la Ec. (5.14) es muy explícita, pues ahí se ha escrito $\mathbb{E}(\mathcal{B},t)$ para enfatizar que el valor de la propiedad extensiva corresponde al cuerpo $\mathcal{B}$. Sin embargo, en lo que sucesivo, se simplificará la notación omitiendo el símbolo $\mathcal{B}$ es decir, se escribirá $\mathbb{E}(t)$ en vez de $\mathbb{E}(\mathcal{B},t)$.

Hay diferentes formas de definir a las propiedades intensivas. Como aquí lo hemos hecho, es por unidad de volumen. Sin embargo, es frecuente que se le defina por unidad de masa véase [9]. Es fácil ver que la propiedad intensiva por unidad de volumen es igual a la propiedad intensiva por unidad de masa multiplicada por la densidad de masa (es decir, masa por unidad de volumen), por lo que es fácil pasar de un concepto al otro, utilizando la densidad de masa.

Sin embargo, una ventaja de utilizar a las propiedades intensivas por unidad de volumen, en lugar de las propiedades intensivas por unidad de masa, es que la correspondencia entre las propiedades extensivas y las intensivas es más directa: dada una propiedad extensiva, la propiedad intensiva

que le corresponde es la función que aparece como integrando, cuando aquélla se expresa como una integral de volumen. Además, del cálculo se sabe que

$$\psi(\underline{x},t) \equiv \lim_{Vol\to 0} \frac{\mathbb{E}(t)}{Vol} = \lim_{Vol\to 0} \frac{\int_{\mathcal{B}(t)} \psi(\underline{\xi},t) d\underline{\xi}}{Vol}. \tag{5.15}$$

La Ec. (5.15) proporciona un procedimiento efectivo para determinar las propiedades extensivas experimentalmente: se mide la propiedad extensiva en un volumen pequeño del sistema continuo de que se trate, se le divide entre el volumen y el cociente que se obtiene es una buena aproximación de la propiedad intensiva.

El uso que haremos del concepto de propiedad extensiva es, desde luego, lógicamente consistente. En particular, cualquier propiedad que satisface las condiciones de la definición de propiedad extensiva establecidas antes es, por ese hecho, una propiedad extensiva. Sin embargo, no todas las propiedades extensivas que se pueden obtener de esta manera son de interés en la mecánica de los medios continuos. Una razón básica por la que ellas son importantes es porqué el modelo general de los sistemas continuos se formula en términos de ecuaciones de balance de propiedades extensivas, como se verá más adelante.

5.2.3 Balance de Propiedades Extensivas e Intensivas

Los modelos matemáticos de los sistemas continuos están constituidos por balances de propiedades extensivas. Por ejemplo, los modelos de transporte de solutos (los contaminantes transportados por corrientes superficiales o subterráneas, son un caso particular de estos procesos de transporte) se construyen haciendo el balance de la masa de soluto que hay en cualquier dominio del espacio físico. Aquí, el término balance se usa, esencialmente, en un sentido contable. En la contabilidad que se realiza para fines financieros o fiscales, la diferencia de las entradas menos las salidas nos da el aumento, o cambio, de capital. En forma similar, en la mecánica de los medios continuos se realiza, en cada cuerpo del sistema continuo, un balance de las propiedades extensivas en que se basa el modelo.

Ecuación de Balance Global Para realizar tales balances es necesario, en primer lugar, identificar las causas por las que las propiedades extensivas pueden cambiar. Tomemos como ejemplo de propiedad extensiva a las existencias de maíz que hay en el país. La primera pregunta es: ¿qué causas

pueden motivar su variación, o cambio, de esas existencias?. Un análisis sencillo nos muestra que dicha variación puede ser debida a que se produzca o se consuma. También a que se importe o se exporte por los límites del país (fronteras o litorales). Y con esto se agotan las causas posibles; es decir, esta lista es exhaustiva. Producción y consumo son términos similares, pero sus efectos tienen signos opuestos, que fácilmente se engloban en uno solo de esos conceptos. De hecho, si convenimos en que la producción puede ser negativa, entonces el consumo es una producción negativa.

Una vez adoptada esta convención, ya no es necesario ocuparnos separadamente del consumo. En forma similar, la exportación es una importación negativa. Entonces, el incremento en las existencias $\Delta\mathbb{E}$ en un período Δt queda dado por la ecuación

$$\Delta\mathbb{E} = \mathbb{P} + \mathbb{I} \tag{5.16}$$

donde a la producción y a la importación, ambas con signo, se les ha representado por $\mathbb{P}$ y $\mathbb{I}$ respectivamente.

Similarmente, en la mecánica de los medios continuos, la lista exhaustiva de las causas por las que una propiedad extensiva de cualquier cuerpo puede cambiar, contiene solamente dos motivos:

i) Por producción en el interior del cuerpo; y

ii) Por importación (es decir, transporte) a través de la frontera.

Esto conduce a la siguiente ecuación de "balance global", de gran generalidad, para las propiedades extensivas

$$\frac{d\mathbb{E}}{dt}(t) = \int_{\mathcal{B}(t)} g(\underline{x},t)d\underline{x} + \int_{\partial\mathcal{B}(t)} q(\underline{x},t)d\underline{x} + \int_{\Sigma(t)} g_\Sigma(\underline{x},t)d\underline{x}. \tag{5.17}$$

Donde $g(\underline{x},t)$ es la generación en el interior del cuerpo, con signo, de la propiedad extensiva correspondiente, por unidad de volumen, por unidad de tiempo. Además, en la Ec. (5.17) se ha tomado en cuenta la posibilidad de que haya producción concentrada en la superficie $\Sigma(t)$, la cual está dada en esa ecuación por la última integral, donde $g_\Sigma(\underline{x},t)$ es la producción por unidad de área. Por otra parte $q(\underline{x},t)$ es lo que se importa o transporta hacia el interior del cuerpo a través de la frontera del cuerpo $\partial\mathcal{B}(t)$, en otras palabras, es el flujo de la propiedad extensiva a través de la frontera del cuerpo, por unidad de área, por unidad de tiempo. Puede demostrarse, con

base en hipótesis válidas en condiciones muy generales, que para cada tiempo t existe un campo vectorial $\tau(\underline{x},t)$ tal que

$$q(\underline{x},t) \equiv \tau(\underline{x},t)\cdot \underline{n}(\underline{x},t) \tag{5.18}$$

donde $\underline{n}(\underline{x},t)$ es normal exterior a $\partial\mathcal{B}(t)$. En vista de esta relación, la Ec. (5.17) de balance se puede escribir como

$$\frac{d\mathbb{E}}{dt}(t) = \int_{\mathcal{B}(t)} g(\underline{x},t)d\underline{x} + \int_{\partial\mathcal{B}(t)} \tau(\underline{x},t)\cdot\underline{n}(\underline{x},t)d\underline{x} + \int_{\Sigma(t)} g_{\Sigma}(\underline{x},t)d\underline{x}. \tag{5.19}$$

La relación (5.19) se le conoce con el nombre de "ecuación general de balance global" y es la ecuación básica de los balances de los sistemas continuos. A la función $g(\underline{x},t)$ se le denomina el generación interna y al campo vectorial $\tau(\underline{x},t)$ el campo de flujo.

Condiciones de Balance Local Los modelos de los sistemas continuos están constituidos por las ecuaciones de balance correspondientes a una colección de propiedades extensivas. Así, a cada sistema continuo le corresponde una familia de propiedades extensivas, tal que, el modelo matemático del sistema está constituido por las condiciones de balance de cada una de las propiedades extensivas de dicha familia.

Sin embargo, las propiedades extensivas mismas no se utilizan directamente en la formulación del modelo, en su lugar se usan las propiedades intensivas asociadas a cada una de ellas. Esto es posible porqué las ecuaciones de balance global son equivalentes a las llamadas condiciones de balance local, las cuales se expresan en términos de las propiedades intensivas correspondientes. Las condiciones de balance local son de dos clases: 'las ecuaciones diferenciales de balance local' y 'las condiciones de salto'.

Las primeras son ecuaciones diferenciales parciales, que se deben satisfacer en cada punto del espacio ocupado por el sistema continuo, y las segundas son ecuaciones algebraicas que las discontinuidades deben satisfacer donde ocurren; es decir, en cada punto de Σ. Cabe mencionar que las ecuaciones diferenciales de balance local son de uso mucho más amplio que las condiciones de salto, pues estas últimas solamente se aplican cuando y donde hay discontinuidades, mientras que las primeras en todo punto del espacio ocupado por el sistema continuo.

Una vez establecidas las ecuaciones diferenciales y de salto del balance local, e incorporada la información científica y tecnológica necesaria para

completar el modelo (la cual por cierto se introduce a través de las llamadas 'ecuaciones constitutivas'), el problema matemático de desarrollar el modelo y derivar sus predicciones se transforma en uno correspondiente a la teoría de la ecuaciones diferenciales, generalmente parciales, y sus métodos numéricos.

Las Ecuaciones de Balance Local En lo que sigue se supone que las propiedades intensivas pueden tener discontinuidades, de salto exclusivamente, a través de la superficie $\Sigma(t)$. Se entiende por 'discontinuidad de salto', una en que el límite por ambos lados de $\Sigma(t)$ existe, pero son diferentes.

Se utilizará en lo que sigue los resultados matemáticos que se dan a continuación, ver [34].

Teorema 3 *Para cada $t > 0$, sea $\mathcal{B}(t) \subset \mathbb{R}^3$ el dominio ocupado por un cuerpo. Suponga que la 'propiedad intensiva' $\psi(\underline{x}, t)$ es de clase C^1, excepto a través de la superficie $\Sigma(t)$. Además, sean las funciones $\underline{v}(\underline{x}, t)$ y $\underline{v}_\Sigma(\underline{x}, t)$ esta última definida para $\underline{x} \in \Sigma(t)$ solamente, las velocidades de las partículas y la de $\Sigma(t)$, respectivamente. Entonces*

$$\frac{d}{dt}\int_{\mathcal{B}(t)} \psi d\underline{x} \equiv \int_{\mathcal{B}(t)} \left\{ \frac{\partial \psi}{\partial t} + \nabla \cdot (\underline{v}\psi) \right\} d\underline{x} + \int_{\Sigma} \left[(\underline{v} - \underline{v}_\Sigma)\, \psi \right] \cdot \underline{n} d\underline{x}. \tag{5.20}$$

Teorema 4 *Considere un sistema continuo, entonces, la 'ecuación de balance global' (5.19) se satisface para todo cuerpo del sistema continuo si y solamente si se cumplen las condiciones siguientes:*

i) La ecuación diferencial

$$\frac{\partial \psi}{\partial t} + \nabla \cdot (\underline{v}\psi) = \nabla \cdot \underline{\tau} + g \tag{5.21}$$

vale en todo punto $\underline{x} \in \mathbb{R}^3$, de la región ocupada por el sistema.

ii) La ecuación

$$\left[\psi \left(\underline{v} - \underline{v}_\Sigma \right) - \underline{\tau} \right] \cdot \underline{n} = g_\Sigma \tag{5.22}$$

vale en todo punto $\underline{x} \in \Sigma$.

A las ecuaciones (5.21) y (5.22), se les llama 'ecuación diferencial de balance local' y 'condición de salto', respectivamente.

Desde luego, el caso más general que se estudiará se refiere a situaciones dinámicas; es decir, aquéllas en que las propiedades intensivas cambian con el tiempo. Sin embargo, los estados estacionarios de los sistemas continuos

son de sumo interés. Por estado estacionario se entiende uno en que las propiedades intensivas son independientes del tiempo. En los estados estacionarios, además, las superficies de discontinuidad $\Sigma(t)$ se mantienen fijas (no se mueven). En este caso $\frac{\partial \psi}{\partial t} = 0$ y $\underline{v}_\Sigma = 0$. Por lo mismo, para los estados estacionarios, la ecuación de balance local y la condición de salto se reducen a

$$\nabla \cdot (\underline{v}\psi) = \nabla \cdot \underline{\tau} + g \tag{5.23}$$

que vale en todo punto $\underline{x} \in \mathbb{R}^3$ y

$$[\psi \underline{v} - \underline{\tau}] \cdot \underline{n} = g_\Sigma \tag{5.24}$$

que se satisface en todo punto de la discontinuidad $\Sigma(t)$ respectivamente.

5.3 Ejemplos de Modelos

Una de las aplicaciones más sencillas de las condiciones de balance local es para formular restricciones en el movimiento. Aquí ilustramos este tipo de aplicaciones formulando condiciones que se deben cumplir localmente cuando un fluido es incompresible. La afirmación de que un fluido es incompresible significa que todo cuerpo conserva el volumen de fluido en su movimiento. Entonces, se considerarán dos casos: el de un 'fluido libre' y el de un 'fluido en un medio poroso'. En el primer caso, el fluido llena completamente el espacio físico que ocupa el cuerpo, por lo que el volumen del fluido es igual al volumen del dominio que ocupa el cuerpo, así

$$V_f(t) = \int_{\mathcal{B}(t)} d\underline{x} \tag{5.25}$$

aquí, $V_f(t)$ es el volumen del fluido y $\mathcal{B}(t)$ es el dominio del espacio físico (es decir, de $\mathbb{R}^3$) ocupado por el cuerpo. Observe que una forma más explícita de esta ecuación es

$$V_f(t) = \int_{\mathcal{B}(t)} 1 d\underline{x} \tag{5.26}$$

porqué en la integral que aparece en la Ec. (5.25) el integrando es la función idénticamente 1. Comparando esta ecuación con la Ec. (5.14), vemos que el volumen del fluido es una propiedad extensiva y que la propiedad intensiva que le corresponde es $\psi = 1$.

Además, la hipótesis de incompresibilidad implica

$$\frac{dV_f}{dt}(t) = 0 \tag{5.27}$$

está es el balance global de la Ec. (5.19), con $g = g_\Sigma = 0$ y $\tau = 0$, el cual a su vez es equivalente a las Ecs. (5.21) y (5.22). Tomando en cuenta además que $\psi = 1$, la Ec. (5.21) se reduce a

$$\nabla \cdot \underline{v} = 0. \tag{5.28}$$

Esta es la bien conocida condición de incompresibilidad para un fluido libre, además, aplicando la Ec. (5.22) donde haya discontinuidades, se obtiene $[\underline{v}] \cdot \underline{n} = 0$. Esto implica que si un fluido libre es incompresible, la velocidad de sus partículas es necesariamente continua.

El caso en que el fluido se encuentra en un 'medio poroso', es bastante diferente. Un medio poroso es un material sólido que tiene huecos distribuidos en toda su extensión, cuando los poros están llenos de un fluido, se dice que el medio poroso esta 'saturado'. Esta situación es la de mayor interés en la práctica y es también la más estudiada. En muchos de los casos que ocurren en las aplicaciones el fluido es agua o petróleo. A la fracción del volumen del sistema, constituido por la 'matriz sólida' y los huecos, se le llama 'porosidad' y se le representara por ϕ, así

$$\phi(x,t) = \lim_{V \to 0} \frac{\text{Volumen de huecos}}{\text{Volumen total}} \tag{5.29}$$

aquí hemos escrito $\phi(x,t)$ para enfatizar que la porosidad generalmente es función tanto de la posición como del tiempo. Las variaciones con la posición pueden ser debidas, por ejemplo, a la heterogeneidad del medio y los cambios con el tiempo a su elasticidad; es decir, los cambios de presión del fluido originan esfuerzos en los poros que los dilatan o los encogen.

Cuando el medio está saturado, el volumen del fluido V_f es igual al volumen de los huecos del dominio del espacio físico que ocupa, así

$$V_f(t) = \int_{\mathcal{B}(t)} \phi(x,t) d\underline{x}. \tag{5.30}$$

En vista de esta ecuación, la propiedad intensiva asociada al volumen de fluido es la porosidad $\phi(x,t)$ por lo que la condición de incompresibilidad del fluido contenido en un medio poroso, está dada por la ecuación diferencial

$$\frac{\partial \phi}{\partial t} + \nabla \cdot (\underline{v}\phi) = 0. \tag{5.31}$$

Que la divergencia de la velocidad sea igual a cero en la Ec. (5.28) como condición para que un fluido en su movimiento libre conserve su volumen, es ampliamente conocida. Sin embargo, este no es el caso de la Ec. (5.31), como condición para la conservación del volumen de los cuerpos de fluido contenidos en un medio poroso. Finalmente, debe observarse que cualquier fluido incompresible satisface la Ec. (5.28) cuando se mueve en el espacio libre y la Ec. (5.31) cuando se mueve en un medio poroso.

Cuando un fluido efectúa un movimiento en el que conserva su volumen, al movimiento se le llama 'isocorico'. Es oportuno mencionar que si bien es cierto que cuando un fluido tiene la propiedad de ser incompresible, todos sus movimientos son isocoricos, lo inverso no es cierto: un fluido compresible en ocasiones puede efectuar movimientos isocoricos.

Por otra parte, cuando un fluido conserva su volumen en su movimiento satisface las condiciones de salto de Ec. (5.22), las cuales para este caso son

$$[\phi(\underline{v} - \underline{v}_\Sigma)] \cdot \underline{n} = 0. \tag{5.32}$$

En aplicaciones a geohidrología y a ingeniería petrolera, las discontinuidades de la porosidad están asociadas a cambios en los estratos geológicos y por está razón estan fijas en el espacio; así, $\underline{v}_\Sigma = 0$ y la Ec. (5.32) se reduce a

$$[\phi \underline{v}] \cdot \underline{n} = 0 \tag{5.33}$$

o, de otra manera

$$\phi_+ v_{n_+} = \phi_- v_{n_-}. \tag{5.34}$$

Aquí, la componente normal de la velocidad es $v_n \equiv \underline{v} \cdot \underline{n}$ y los subíndices más y menos se utilizan para denotar los límites por los lado más y menos de Σ, respectivamente. Al producto de la porosidad por la velocidad se le conoce con el nombre de velocidad de Darcy $\underline{U}$, es decir

$$\underline{U} = \phi \underline{v} \tag{5.35}$$

utilizando, las Ecs. (5.33) y (5.34) obtenemos

$$[\underline{U}] \cdot \underline{n} = 0 \quad \text{y} \quad \underline{U}_{n_+} = \underline{U}_{n_-} \tag{5.36}$$

es decir, 1.

La Ec. (5.34) es ampliamente utilizada en el estudio del agua subterránea (geohidrología). Ahí, es frecuente que la porosidad ϕ sea discontinua en la

superficie de contacto entre dos estratos geológicos diferentes, pues generalmente los valores que toma esta propiedad dependen de cada estrato. En tal caso, $\phi_+ \neq \phi_-$ por lo que $v_{n_+} \neq v_{n_-}$ necesariamente.

Para más detalles de la forma y del desarrollo de algunos modelos usados en ciencias de la tierra, véase [34], [9], [47] y [8].

6 Apéndice B: Ecuaciones Diferenciales Parciales

Cada una de las ecuaciones de balance da lugar a una ecuación diferencial parcial u ordinaria (en el caso en que el modelo depende de una sola variable independiente), la cual se complementa con las condiciones de salto, en el caso de los modelos discontinuos. Por lo mismo, los modelos de los sistemas continuos estan constituidos por sistemas de ecuaciones diferenciales cuyo número es igual al número de propiedades intensivas que intervienen en la formulación del modelo básico.

Los sistemas de ecuaciones diferenciales se clasifican en elípticas, hiperbólicas y parabólicas. Es necesario aclarar que esta clasificación no es exhaustiva; es decir, existen sistemas de ecuaciones diferenciales que no pertenecen a ninguna de estas categorías. Sin embargo, casi todos los modelos de sistemas continuos, en particular los que han recibido mayor atención hasta ahora, si están incluidos en alguna de estas categorías.

6.1 Clasificación

Es importante clasificar a las ecuaciones diferenciales parciales y a los sistemas de tales ecuaciones, por qué muchas de sus propiedades son comunes a cada una de sus clases. Así, su clasificación es un instrumento para alcanzar el objetivo de unidad conceptual. La forma más general de abordar la clasificación de tales ecuaciones, es estudiando la clasificación de sistemas de ecuaciones. Sin embargo, aquí solamente abordaremos el caso de una ecuación diferencial de segundo orden, pero utilizando un método de análisis que es adecuado para extenderse a sistemas de ecuaciones.

La forma general de un operador diferencial cuasi-lineal de segundo orden definido en $\Omega \subset \mathbb{R}^2$ es

$$\mathcal{L}u \equiv a(x,y)\frac{\partial^2 u}{\partial x^2} + b(x,y)\frac{\partial^2 u}{\partial x \partial y} + c(x,y)\frac{\partial^2 u}{\partial y^2} = F(x,y,u,u_x,u_y) \qquad (6.1)$$

para una función u de variables independientes x e y. Nos restringimos al caso en que a, b y c son funciones sólo de x e y y no funciones de u.

Para la clasificación de las ecuaciones de segundo orden consideraremos una simplificación de la ecuación anterior en donde $F(x,y,u,u_x,u_y) = 0$ y

los coeficientes a, b y c son funciones constantes, es decir

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} = 0 \tag{6.2}$$

en la cual, examinaremos los diferentes tipos de solución que se pueden obtener para diferentes elecciones de a, b y c. Entonces iniciando con una solución de la forma

$$u(x, y) = f(mx + y) \tag{6.3}$$

para una función f de clase $\mathbb{C}^2$ y para una constante m, que deben ser determinadas según los requerimientos de la Ec. (6.2). Usando un apóstrofe para denotar la derivada de f con respecto de su argumento, las derivadas parciales requeridas de segundo orden de la Ec. (6.2) son

$$\frac{\partial^2 u}{\partial x^2} = m^2 f'', \quad \frac{\partial^2 u}{\partial x \partial y} = m f'', \quad \frac{\partial^2 u}{\partial y^2} = f'' \tag{6.4}$$

sustituyendo la ecuación anterior en la Ec. (6.2) obtenemos

$$\left(am^2 + bm + c\right) f'' = 0 \tag{6.5}$$

de la cual podemos concluir que $f'' = 0$ ó $am^2 + bm + c = 0$ ó ambas. En el caso de que $f'' = 0$ obtenemos la solución $f = f_0 + mx + y$, la cual es una función lineal de x e y y es expresada en términos de dos constantes arbitrarias, f_0 y m. En el otro caso obtenemos

$$am^2 + bm + c = 0 \tag{6.6}$$

resolviendo esta ecuación cuadrática para m obtenemos las dos soluciones

$$m_1 = \frac{\left(-b + \sqrt[2]{b^2 - 4ac}\right)}{2a}, \quad m_2 = \frac{\left(-b - \sqrt[2]{b^2 - 4ac}\right)}{2a} \tag{6.7}$$

de donde es evidente la importancia de los coeficientes de la Ec. (6.2), ya que el signo del discriminante $(b^2 - 4ac)$ es crucial para determinar el número y tipo de soluciones de la Ec. (6.6). Así, tenemos tres casos a considerar:

Caso I. $(b^2 - 4ac) > 0$, **Ecuación Hiperbólica**.

La Ec. (6.6) tiene dos soluciones reales distintas, m_1 y m_2. Así cualquier función de cualquiera de los dos argumentos $m_1 x + y$ ó $m_2 x + y$ resuelven a la Ec. (6.2). Por lo tanto la solución general de la Ec. (6.2) es

$$u(x, y) = \mathcal{F}\left(m_1 x + y\right) + \mathcal{G}(m_2 x + y) \tag{6.8}$$

donde $\mathcal{F}$ y $\mathcal{G}$ son cualquier función de clase $\mathbb{C}^2$. Un ejemplo de este tipo de ecuaciones es la ecuación de onda, cuya ecuación canónica es

$$\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial t^2} = 0. \tag{6.9}$$

Caso II. $(b^2 - 4ac) = 0$, **Ecuación Parabólica**.

Asumiendo que $b \neq 0$ y $a \neq 0$ (lo cual implica que $c \neq 0$). Entonces se tiene una sola raíz degenerada de la Ec. (6.6) con el valor de $m_1 = \frac{-b}{2a}$ que resuelve a la Ec. (6.2). Por lo tanto la solución general de la Ec. (6.2) es

$$u(x, y) = \mathcal{F}(m_1 x + y) + y\mathcal{G}(m_1 x + y) \tag{6.10}$$

donde $\mathcal{F}$ y $\mathcal{G}$ son cualquier función de clase $\mathbb{C}^2$. Si $b = 0$ y $a = 0$, entonces la solución general es

$$u(x, y) = \mathcal{F}(x) + y\mathcal{G}(x) \tag{6.11}$$

la cual es análoga si $b = 0$ y $c = 0$. Un ejemplo de este tipo de ecuaciones es la ecuación de difusión o calor, cuya ecuación canónica es

$$\frac{\partial^2 u}{\partial x^2} - \frac{\partial u}{\partial t} = 0. \tag{6.12}$$

Caso III. $(b^2 - 4ac) < 0$, **Ecuación Elíptica**.

La Ec. (6.6) tiene dos soluciones complejas m_1 y m_2 las cuales satisfacen que m_2 es el conjugado complejo de m_1, es decir, $m_2 = m_1$. La solución general puede ser escrita en la forma

$$u(x, y) = \mathcal{F}(m_1 x + y) + \mathcal{G}(m_2 x + y) \tag{6.13}$$

donde $\mathcal{F}$ y $\mathcal{G}$ son cualquier función de clase $\mathbb{C}^2$. Un ejemplo de este tipo de ecuaciones es la ecuación de Laplace, cuya ecuación canónica es

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0. \tag{6.14}$$

Consideremos ahora el caso de un operador diferencial lineal de segundo orden definido en $\Omega \subset \mathbb{R}^n$ cuya forma general es

$$\mathcal{L}u = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}\frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=1}^{n} b_i \frac{\partial u}{\partial x_i} + cu \tag{6.15}$$

y consideremos también la ecuación homogénea asociada a este operador

$$\mathcal{L}u = 0 \tag{6.16}$$

además, sea $\underline{x} \in \Omega$ un punto del espacio Euclidiano y $V(x)$ una vecindad de ese punto. Sea una función u definida en $V(\underline{x})$ con la propiedad de que exista una variedad Σ de dimensión $n-1$ cerrada y orientada, tal que la función u satisface la Ec. (6.16) en $V(\underline{x})\backslash\Sigma$. Se supone además que existe un vector unitario $\underline{n}$ que apunta en la dirección positiva (único) está definido en Σ. Además, la función u y sus derivadas de primer orden son continuas a través de Σ, mientras que los límites de las segundas derivadas de u existen por ambos lados de Σ. Sea $\underline{x} \in \Sigma$ tal que

$$\left[\frac{\partial^2 u}{\partial x_i \partial x_j}(\underline{x})\right] \neq 0 \tag{6.17}$$

para alguna pareja $i, j = 1, ..., n$. Entonces decimos que la función u es una solución débil de esta ecuación en $\underline{x}$.

Teorema 5 *Una condición necesaria para que existan soluciones débiles de la ecuación homogénea (6.16) en un punto $\underline{x} \in \Sigma$ es que*

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} n_i n_j = 0. \tag{6.18}$$

Así, si definimos a la matriz $\underline{\underline{A}} = (a_{ij})$ y observamos que

$$\underline{n} \cdot \underline{\underline{A}} \cdot \underline{n} = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} n_i n_j \tag{6.19}$$

entonces podemos decir que:

I) Cuando todos los eigenvalores de la matriz $\underline{\underline{A}}$ son distintos de cero y además del mismo signo, entonces se dice que el operador es **Elíptico**.

II) Cuando todos los eigenvalores de la matriz $\underline{\underline{A}}$ son distintos de cero y además $n-1$ de ellos tienen el mismo signo, entonces se dice que el operador es **Hiperbólico**.

III) Cuando uno y sólo uno de los eigenvalores de la matriz $\underline{\underline{A}}$ es igual a cero, entonces se dice que el operador es **Parabólico**.

Para el caso en que $n = 2$, esta forma de clasificación coincide con la dada anteriormente.

6.2 Condiciones Iniciales y de Frontera

Dado un problema concreto de ecuaciones en derivadas parciales sobre un dominio Ω, si la solución existe, esta no es única ya que generalmente este tiene un número infinito de soluciones. Para que el problema tenga una y sólo una solución es necesario imponer condiciones auxiliares apropiadas y estas son las condiciones iniciales y condiciones de frontera.

En esta sección sólo se enuncian de manera general las condiciones iniciales y de frontera que son esenciales para definir un problema de ecuaciones diferenciales:

A) **Condiciones Iniciales**

Las condiciones iniciales expresan el valor de la función al tiempo inicial $t = 0$ (t puede ser fijada en cualquier valor)

$$u(\underline{x}, \underline{y}, 0) = \gamma(\underline{x}, \underline{y}). \tag{6.20}$$

B) **Condiciones de Frontera**

Las condiciones de frontera especifican los valores que la función $u(\underline{x}, \underline{y}, t)$ o $\nabla u(\underline{x}, \underline{y}, t)$ tomarán en la frontera $\partial\Omega$, siendo de tres tipos posibles:

1) **Condiciones tipo Dirichlet**

Especifica los valores que la función $u(\underline{x}, \underline{y}, t)$ toma en la frontera $\partial\Omega$

$$u(\underline{x}, \underline{y}, t) = \gamma(\underline{x}, \underline{y}). \tag{6.21}$$

2) **Condiciones tipo Neumann**

Aquí se conoce el valor de la derivada de la función $u(\underline{x}, \underline{y}, t)$ con respecto a la normal $\underline{n}$ a lo largo de la frontera $\partial\Omega$

$$\nabla u(\underline{x}, \underline{y}, t) \cdot \underline{n} = \gamma(\underline{x}, \underline{y}). \tag{6.22}$$

3) **Condiciones tipo Robin**

esta condición es una combinación de las dos anteriores

$$\alpha(\underline{x}, \underline{y})u(\underline{x}, \underline{y}, t) + \beta(\underline{x}, \underline{y})\nabla u(\underline{x}, \underline{y}, t) \cdot \underline{n} = g_{\partial}(\underline{x}, \underline{y}) \tag{6.23}$$

$\forall\ \underline{x},\ \underline{y} \in \partial\Omega$.

En un problema dado se debe prescribir las condiciones iniciales al problema y debe existir alguno de los tipos de condiciones de frontera o combinación de ellas en $\partial\Omega$.

6.3 Modelos Completos

Los modelos de los sistemas continuos están constituidos por:

- Una colección de propiedades intensivas o lo que es lo mismo, extensivas.
- El conjunto de ecuaciones de balance local correspondientes (diferenciales y de salto).
- Suficientes relaciones que liguen a las propiedades intensivas entre sí y que definan a g, $\underline{\tau}$ y $\underline{v}$ en términos de estas, las cuales se conocen como leyes constitutivas.

Una vez que se han planteado las ecuaciones que gobiernan al problema, las condiciones iniciales, de frontera y mencionado los procesos que intervienen de manera directa en el fenómeno estudiado, necesitamos que nuestro modelo sea *completo.* Decimos que el modelo de un sistema es *completo* si define un problema *bien planteado.* Un problema de valores iniciales y condiciones de frontera es *bien planteado* si cumple que:

i) Existe una y sólo una solución y,

ii) La solución depende de manera continua de las condiciones iniciales y de frontera del problema.

Es decir, un modelo completo es aquél en el cual se incorporan condiciones iniciales y de frontera que definen conjuntamente con las ecuaciones diferenciales un problema bien planteado.

A las ecuaciones diferenciales definidas en $\Omega \subset \mathbb{R}^n$

$$\begin{aligned} \Delta u &= 0 \\ \frac{\partial^2 u}{\partial t^2} - \Delta u &= 0 \\ \frac{\partial u}{\partial t} - \Delta u &= 0 \end{aligned} \tag{6.24}$$

se les conoce con los nombres de ecuación de Laplace, ecuación de onda y ecuación del calor, respectivamente. Cuando se considera la primera de estas

ecuaciones, se entiende que u es una función del vector $x \equiv (x_1, ..., x_n)$, mientras que cuando se considera cualquiera de las otras dos, u es una función del vector $x \equiv (x_1, ..., x_n, t)$. Así, en estos últimos casos el número de variables independientes es $n+1$ y los conceptos relativos a la clasificación y las demás nociones discutidas con anterioridad deben aplicarse haciendo la sustitución $n \to n+1$ e identificando $x_{n+1} = t$.

Ecuación de Laplace Para la ecuación de Laplace consideraremos condiciones del tipo Robin. En particular, condiciones de Dirichlet y condiciones de Neumann. Sin embargo, en este último caso, la solución no es única pues cualquier función constante satisface la ecuación de Laplace y también $\frac{\partial u}{\partial \underline{n}} = g_\partial$ con $g_\partial = 0$.

Ecuación de Onda Un problema general importante consiste en obtener la solución de la ecuación de onda, en el dominio del espacio-tiempo $\Omega \times [0, t]$, que satisface para cada $t \in (0, t]$ una condición de frontera de Robin en $\partial\Omega$ y las condiciones iniciales

$$u(\underline{x}, 0) = u_0(\underline{x}) \quad \text{y} \quad \frac{\partial u}{\partial t}(\underline{x}, 0) = v_0(\underline{x}), \quad \forall \underline{x} \in \Omega \tag{6.25}$$

aquí $u_0(\underline{x})$ y $v_0(\underline{x})$ son dos funciones prescritas. El hecho de que para la ecuación de onda se prescriban los valores iniciales, de la función y su derivada con respecto al tiempo, es reminiscente de que en la mecánica de partículas se necesitan las posiciones y las velocidades iniciales para determinar el movimiento de un sistema de partículas.

Ecuación de Calor También para la ecuación del calor un problema general importante consiste en obtener la solución de la ecuación de onda, en el dominio del espacio-tiempo $\Omega \times [0, t]$, que satisface para cada $t \in (0, t]$ una condición de frontera de Robin y ciertas condiciones iniciales. Sin embargo, en este caso en ellas sólo se prescribe a la función

$$u(\underline{x}, 0) = u_0(\underline{x}), \quad \forall \underline{x} \in \Omega. \tag{6.26}$$

7 Apéndice C: Análisis Funcional y Problemas Variacionales

En este capítulo se detallan los conceptos básicos de análisis funcional y problemas variacionales con énfasis en problemas elípticos de orden par $2m$, para comenzar detallaremos lo que entendemos por un operador diferencial parcial elíptico de orden par $2m$ en n variables, para después definir a los espacios de Sobolev para poder tratar problemas variacionales con valor en la frontera.

En donde, restringiéndonos a problemas elípticos, contestaremos una cuestión central en la teoría de problemas elípticos con valores en la frontera, y está se relaciona con las condiciones bajo las cuales uno puede esperar que el problema tenga solución y esta es única, así como conocer la regularidad de la solución, para mayor referencia de estos resultados ver [45], [48], [49] y [53].

7.1 Operador Lineal Elíptico

Definición 6 *Entenderemos por un dominio al conjunto $\Omega \subset \mathbb{R}^n$ que sea abierto y conexo.*

Para poder expresar de forma compacta derivadas parciales de orden m o menor, usaremos la definición siguiente.

Definición 7 *Sea $\mathbb{Z}_+^n$ el conjunto de todas las n-duplas de enteros no negativos, un miembro de $\mathbb{Z}_+^n$ se denota usualmente por α ó β (por ejemplo $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$. Denotaremos por $|\alpha|$ la suma $|\alpha| = \alpha_1 + \alpha_2 + ... + \alpha_n$ y por $D^\alpha u$ la derivada parcial*

$$D^\alpha u = \frac{\partial^{|\alpha|} u}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2} ... \partial x_n^{\alpha_n}} \tag{7.1}$$

así, si $|\alpha| = m$, entonces $D^\alpha u$ denota la m-ésima derivada parcial de u.

Sea $\mathcal{L}$ un operador diferencial parcial de orden par $2m$ en n variables y de la forma

$$\mathcal{L}u = \sum_{|\alpha|,|\beta| \leq m} (-1)^{|\alpha|} D^\alpha \left(a_{\alpha\beta}(\underline{x}) D^\beta u \right), \quad \underline{x} \in \Omega \subset \mathbb{R}^n \tag{7.2}$$

donde Ω es un dominio en $\mathbb{R}^n$. Los coeficientes $a_{\alpha\beta}$ son funciones suaves real valuadas de $\underline{x}$.

El operador $\mathcal{L}$ es asumido que aparece dentro de una ecuación diferencial parcial con la forma

$$\mathcal{L}u = f, \tag{7.3}$$

donde f pertenece al rango del operador $\mathcal{L}$.

La clasificación del operador $\mathcal{L}$ depende sólo de los coeficientes de la derivada más alta, esto es, de la derivada de orden $2m$, y a los términos involucrados en esa derivada son llamados la parte principal del operador $\mathcal{L}$ denotado por $\mathcal{L}_0$ y para el operador (7.2) es de la forma

$$\mathcal{L}_0 = \sum_{|\alpha|,|\beta|\leq m} a_{\alpha\beta} D^{\alpha+\beta} u. \tag{7.4}$$

Teorema 8 *Sea ξ un vector en $\mathbb{R}^n$, y sea $\xi^\alpha = \xi_n^{\alpha_1} ... \xi_n^{\alpha_n}, \alpha \in \mathbb{Z}_n^+$. Entonces*
i) $\mathcal{L}$ es elíptico en $\underline{x_o} \in \Omega$, si

$$\sum_{|\alpha|,|\beta|=m} a_{\alpha\beta}\left(\underline{x_o}\right) \xi^{\alpha+\beta} \neq 0 \qquad \forall \xi \neq 0; \tag{7.5}$$

ii) $\mathcal{L}$ es elíptico, si es elíptico en todos los puntos de Ω;
iii) $\mathcal{L}$ es fuertemente elíptico, si existe un número $\mu > 0$ tal que

$$\left| \sum_{|\alpha|,|\beta|=m} a_{\alpha\beta}\left(\underline{x_o}\right) \xi^{\alpha+\beta} \right| \geq \mu \left|\xi\right|^{2m} \tag{7.6}$$

satisfaciéndose en todo punto $\underline{x_o} \in \Omega$, y para todo $\xi \in \mathbb{R}^n$. Aquí $|\xi| = \left(\xi_1^2 + ... + \xi_n^2\right)^{\frac{1}{2}}$.

Para el caso en el cual $\mathcal{L}$ es un operador de $2do$ orden ($m = 1$), la notación se simplifica, tomando la forma

$$\mathcal{L}u = -\sum_{i,j=1}^{n} \frac{\partial}{\partial x_i}\left(a_{ij}(\underline{x})\frac{\partial u}{\partial x_j}\right) + \sum_{j=1}^{n} a_j \frac{\partial u}{\partial x_j} + a_0 u = f \tag{7.7}$$

en Ω.

Para coeficientes adecuados a_{ij}, a_j y a_0 la condición para conocer si el operador es elíptico, es examinado por la condición

$$\sum_{i,j=1}^{n} a_{ij}(\underline{x_0})\xi_i\xi_j \neq 0 \qquad \forall \xi \neq 0 \tag{7.8}$$

y para conocer si el operador es fuertemente elíptico, es examinado por la condición

$$\sum_{i,j=1}^{n} a_{ij}(\underline{x_0})\xi_i\xi_j > \mu \left|\xi\right|^2 . \tag{7.9}$$

7.2 Espacios de Sobolev

En esta subsección detallaremos algunos resultados de los espacios de Sobolev sobre el conjunto de números reales, en estos espacios son sobre los cuales trabajaremos tanto para plantear el problema elíptico como para encontrar la solución al problema. Primeramente definiremos lo que entendemos por un espacio L^2.

Definición 9 *Una función medible $u(\underline{x})$ definida sobre $\Omega \subset \mathbb{R}^n$ se dice que pertenece al espacio $L^2(\Omega)$ si*

$$\int_{\Omega} |u(\underline{x})|^2 \, d\underline{x} < \infty \tag{7.10}$$

es decir, es integrable.

La definición de los espacios medibles, espacios L^p, distribuciones y derivadas de distribuciones están dados en el apéndice, estos resultados son la base para poder definir a los espacios de Sobolev.

Definición 10 *El espacio de Sobolev de orden m, denotado por $H^m(\Omega)$, es definido*

$$H^m(\Omega) = \left\{ u \mid D^\alpha u \in L^2(\Omega) \quad \forall \alpha \text{ tal que } |\alpha| \leq m \right\}. \tag{7.11}$$

El producto escalar $\langle \cdot, \cdot \rangle$ de dos elementos u y $v \in H^m(\Omega)$ está dado por

$$\langle u, v \rangle_{H^m} = \int_{\Omega} \sum_{|\alpha| \leq m} \left(D^\alpha u\right)\left(D^\alpha v\right) d\underline{x} \text{ para } u, v \in H^m\left(\Omega\right). \tag{7.12}$$

Nota: Es común que el espacio $L^2(\Omega)$ sea denotado por $H^0(\Omega)$.

Un espacio completo con producto interior es llamado un espacio de Hilbert, un espacio normado y completo es llamado espacio de Banach. Y como todo producto interior define una norma, entonces todo espacio de Hilbert es un espacio de Banach.

Definición 11 *La norma $\|\cdot\|_{H^m}$ inducida a partir del producto interior $\langle\cdot,\cdot\rangle_{H^m}$ queda definida por*

$$\|u\|_{H^m}^2 = \langle u,u\rangle_{H^m} = \int_\Omega \sum_{|\alpha|\leq m} \left(D^\alpha u\right)^2 d\underline{x}. \tag{7.13}$$

Ahora, con norma $\|\cdot\|_{H^m}$, el espacio $H^m(\Omega)$ es un espacio de Hilbert, esto queda plasmado en el siguiente resultado.

Teorema 12 *El espacio $H^m(\Omega)$ con la norma $\|\cdot\|_{H^m}$ es un espacio de Hilbert.*

Ya que algunas de las propiedades de los espacios de Sobolev sólo son válidas cuando la frontera del dominio es suficientemente suave. Para describir al conjunto donde los espacios de Sobolev están definidos, es común pedirle algunas propiedades y así definimos lo siguiente.

Definición 13 *Una función f definida sobre un conjunto $\Gamma \subset \mathbb{R}^n$ es llamada Lipschitz continua si existe una constante $L > 0$ tal que*

$$|f(x) - f(y)| \leq L\,|x-y| \quad \forall x,y \in \Gamma. \tag{7.14}$$

Notemos que una función Lipschitz continua es uniformemente continua.

Sea $\Omega \subset \mathbb{R}^n$ $(n \geq 2)$ un dominio con frontera $\partial\Omega$, sea $x_0 \in \partial\Omega$ y construyamos la bola abierta con centro en x_0 y radio ε, i.e. $B(x_0,\varepsilon)$, entonces definiremos el sistema coordenado $(\xi_1,...,\xi_n)$ tal que el segmento $\partial\Omega \cap B(x_0,\varepsilon)$ pueda expresarse como una función

$$\xi_n = f(\xi_1,...,\xi_{n-1}) \tag{7.15}$$

entonces definimos.

Definición 14 *La frontera $\partial\Omega$ del dominio Ω es llamada de Lipschitz si f definida como en la Ec. (7.15) es una función Lipschitz continua.*

El siguiente teorema resume las propiedades más importantes de los espacios de Sobolev $H^m(\Omega)$.

Teorema 15 *Sea $H^m(\Omega)$ el espacio de Sobolev de orden m y sea $\Omega \subset \mathbb{R}^n$ un dominio acotado con frontera Lipschitz. Entonces*

i) $H^r(\Omega) \subset H^m(\Omega)$ si $r \geq m$

ii) $H^m(\Omega)$ es un espacio de Hilbert con respecto a la norma $\|\cdot\|_{H^m}$

iii) $H^m(\Omega)$ es la cerradura con respecto a la norma $\|\cdot\|_{H^m}$ del espacio $C^\infty(\overline{\Omega})$.

De la parte *iii)* del teorema anterior, se puede hacer una importante interpretación: Para toda $u \in H^m(\Omega)$ es siempre posible encontrar una función infinitamente diferenciable f, tal que este arbitrariamente cerca de u en el sentido que

$$\|u - f\|_{H^m} < \varepsilon \tag{7.16}$$

para algún $\varepsilon > 0$ dado.

Cuando $m = 0$, se deduce la propiedad $H^0(\Omega) = L^2(\Omega)$ a partir del teorema anterior.

Corolario 16 *El espacio $L^2(\Omega)$ es la cerradura, con respecto a la norma L^2, del espacio $C^\infty(\overline{\Omega})$.*

Otra propiedad, se tiene al considerar a cualquier miembro de $u \in H^m(\Omega)$, este puede ser identificado con una función en $C^m(\overline{\Omega})$, después de que posiblemente sean cambiados algunos valores sobre un conjunto de medida cero, esto queda plasmado en los dos siguientes resultados.

Teorema 17 *Sean X y Y dos espacios de Banach, con $X \subset Y$. Sea $f : X \to Y$ tal que $f(u) = u$. Si el espacio X tiene definida la norma $\|\cdot\|_X$ y el espacio Y tiene definida la norma $\|\cdot\|_Y$, decimos que X está inmersa continuamente en Y si*

$$\|f(u)\|_Y = \|u\|_Y \leq K \|u\|_X \tag{7.17}$$

para alguna constante $K > 0$.

Teorema 18 *(Inmersión de Sobolev)*

Sea $\Omega \subset \mathbb{R}^n$ un dominio acotado con frontera $\partial\Omega$ de Lipschitz. Si $(m - k) > n/2$, entonces toda función en $H^m(\Omega)$ pertenece a $C^k(\overline{\Omega})$, es decir, hay un miembro que pertenece a $C^k(\overline{\Omega})$. Además, la inmersión

$$H^m(\Omega) \subset C^k(\overline{\Omega}) \tag{7.18}$$

es continua.

7.2.1 Trazas de una Función en $H^m(\Omega)$.

Una parte fundamental en los problemas con valores en la frontera definidos sobre el dominio Ω, es definir de forma única los valores que tomará la función sobre la frontera $\partial\Omega$, en este apartado veremos bajo qué condiciones es posible tener definidos de forma única los valores en la frontera $\partial\Omega$ tal que podamos definir un operador $tr(\cdot)$ continuo que actué en $\overline{\Omega}$ tal que $tr(u) = u_{|\partial\Omega}$.

El siguiente lema nos dice que el operador $tr(\cdot)$ es un operador lineal continuo de $C^1\left(\overline{\Omega}\right)$ a $C(\partial\Omega)$, con respecto a las normas $\|\cdot\|_{H^1(\Omega)}$ y $\|\cdot\|_{L^2(\partial\Omega)}$.

Lema 19 *Sea Ω un dominio con frontera $\partial\Omega$ de Lipschitz. La estimación*

$$\|tr(u)\|_{L^2(\partial\Omega)} \leq C\|u\|_{H^1(\Omega)} \tag{7.19}$$

se satisface para toda función $u \in C^1\left(\overline{\Omega}\right)$, para alguna constante $C > 0$.

Ahora, para el caso $tr(\cdot) : H^1(\Omega) \to L^2(\partial\Omega)$, se tiene el siguiente teorema.

Teorema 20 *Sea Ω un dominio acotado en $\mathbb{R}^n$ con frontera $\partial\Omega$ de Lipschitz. Entonces:*

i) Existe un único operador lineal acotado $tr(\cdot) : H^1(\Omega) \to L^2(\partial\Omega)$, tal que

$$\|tr(u)\|_{L^2(\partial\Omega)} \leq C\|u\|_{H^1(\Omega)}, \tag{7.20}$$

con la propiedad que si $u \in C^1\left(\overline{\Omega}\right)$, entonces $tr(u) = u|_{\partial\Omega}$.

ii) El rango de $tr(\cdot)$ es denso en $L^2(\partial\Omega)$.

El argumento anterior puede ser generalizado para los espacios $H^m(\Omega)$, de hecho, cuando $m > 1$, entonces para toda $u \in H^m(\Omega)$ tenemos que

$$D^\alpha u \in H^1(\Omega) \qquad \text{para } |\alpha| \leq m-1, \tag{7.21}$$

por el teorema anterior, el valor de $D^\alpha u$ sobre la frontera está bien definido y pertenece a $L^2(\Omega)$, es decir

$$tr(D^\alpha u) \in L^2(\Omega), \qquad |\alpha| \leq m-1. \tag{7.22}$$

Además, si u es m-veces continuamente diferenciable, entonces $D^\alpha u$ es al menos continuamente diferenciable para $|\alpha| \leq m-1$ y

$$tr(D^\alpha u) = (D^\alpha u)|_{\partial\Omega}. \tag{7.23}$$

7.2.2 Espacios $H_0^m(\Omega)$.

Los espacio $H_0^m(\Omega)$ surgen comúnmente al trabajar con problemas con valor en la frontera y serán aquellos espacios que se nulifiquen en la frontera del dominio, es decir

Definición 21 *Definimos a los espacios $H_0^m(\Omega)$ como la cerradura, en la norma de Sobolev $\|\cdot\|_{H^m}$, del espacio $C_0^m(\Omega)$ de funciones con derivadas continuas del orden menor que m, todas las cuales tienen soporte compacto en Ω, es decir $H_0^m(\Omega)$ es formado al tomar la unión de $C_0^m(\Omega)$ y de todos los límites de sucesiones de Cauchy en $C_0^m(\Omega)$ que no pertenecen a $C_0^m(\Omega)$.*

Las propiedades básicas de estos espacios están contenidas en el siguiente resultado.

Teorema 22 *Sea Ω un dominio acotado en $\mathbb{R}^n$ con frontera $\partial\Omega$ suficientemente suave y sea $H_0^m(\Omega)$ la cerradura de $C_0^\infty(\Omega)$ en la norma $\|\cdot\|_{H^m}$, entonces*

a) $H_0^m(\Omega)$ es la cerradura de $C_0^\infty(\Omega)$ en la norma $\|\cdot\|_{H^m}$;
b) $H_0^m(\Omega) \subset H^m(\Omega)$;
c) Si $u \in H^m(\Omega)$ pertenece a $H_0^m(\Omega)$, entonces

$$D^\alpha u = 0, \text{ sobre } \partial\Omega, |\alpha| \leq m-1. \tag{7.24}$$

Teorema 23 *(Desigualdad de Poincaré-Friedrichs)*
Sea Ω un dominio acotado en $\mathbb{R}^n$. Entonces existe una constante $C > 0$ tal que

$$\int_\Omega |u|^2 \, dx \leq C \int_\Omega |\nabla u|^2 \, dx \tag{7.25}$$

para toda $u \in H_0^1(\Omega)$.

Introduciendo ahora una familia de semi-normas sobre $H^m(\Omega)$ (una semi-norma $|\cdot|$ satisface casi todos los axiomas de una norma excepto el de positivo definido), de la siguiente forma:

Definición 24 *La semi-norma $|\cdot|_m$ sobre $H^m(\Omega)$, se define como*

$$|u|_m^2 = \sum_{|\alpha|=m} \int_\Omega |D^\alpha u|^2 \, dx. \tag{7.26}$$

Esta es una semi-norma, ya que $|u|_m = 0$ implica que $D^\alpha u = 0$ para $|\alpha| = m$, lo cual no implica que $u = 0$.

La relevancia de esta semi-norma está al aplicar la desigualdad de Poincaré-Friedrichs ya que es posible demostrar que $|\cdot|_1$ es de hecho una norma sobre $H_0^1(\Omega)$.

Corolario 25 *La semi-norma $|\cdot|_1$ es una norma sobre $H_0^1(\Omega)$, equivalente a la norma estándar $\|\cdot\|_{H^1}$.*

Es posible extender el teorema anterior y su corolario a los espacios $H_0^m(\Omega)$ para cualquier $m \geq 1$, de la siguiente forma:

Teorema 26 *Sea Ω un dominio acotado en $\mathbb{R}^n$. Entonces existe una constante $C > 0$ tal que*

$$\|u\|_{L^2}^2 \leq C\,|u|_m^2 \tag{7.27}$$

para toda $u \in H_0^m(\Omega)$, además, $|\cdot|_m$ es una norma sobre $H_0^m(\Omega)$ equivalente a la norma estándar $\|\cdot\|_{H^m}$.

Definición 27 *Sea Ω un dominio acotado en $\mathbb{R}^n$. Definimos por $H^{-m}(\Omega)$ al espacio de todas las funcionales lineales acotadas sobre $H_0^m(\Omega)$, es decir, $H^{-m}(\Omega)$ será el espacio dual del espacio $H_0^m(\Omega)$.*

Teorema 28 *q será una distribución de $H^{-m}(\Omega)$ si y sólo si q puede ser expresada en la forma*

$$q = \sum_{|\alpha|<m} D^\alpha q_\alpha \tag{7.28}$$

donde q_α son funcionales en $L^2(\Omega)$.

Algunos Comentarios y Precisiones Sea Ω un dominio tal que la frontera $\partial\Omega$ es suficientemente suave (considerando sólo frontera Lipschitz continua), entonces existe un operador $\gamma_0 : H^1(\Omega) \to L^2(\Omega)$ lineal y continuo, tal que $\gamma_0 v = tr(v)$ sobre $\partial\Omega$ para toda v suave (por ejemplo $v \in C^1\left(\overline{\Omega}\right)$), un análisis más profundo muestra que tomando las trazas de todas las funciones de $H^1(\Omega)$ uno no obtiene el espacio completo de $L^2(\Omega)$, sólo obtiene un subespacio de este. Tenemos entonces

$$H^1(\partial\Omega) \subset \gamma_0\left(H^1(\Omega)\right) \subset L^2(\partial\Omega) \equiv H^0(\partial\Omega) \tag{7.29}$$

donde cada inclusión es estricta.

Finalmente reconocemos que el espacio $\gamma_0\left(H^1(\Omega)\right)$ pertenece a la familia de espacios $H^s(\partial\Omega)$ y corresponde exactamente a el valor de $s = 1/2$. De tal forma que

$$H^{1/2}(\partial\Omega) = \gamma_0\left(H^1(\Omega)\right) \tag{7.30}$$

con

$$\|g\|_{H^{1/2}(\partial\Omega)} = \inf_{v\in H^1(\Omega) \text{ y } \gamma_0 v = g} \|v\|_{H^1(\Omega)} \tag{7.31}$$

de forma similar, se ve que las trazas de las funciones en $H^2(\Omega)$ pertenecen a el espacio $H^s(\partial\Omega)$ para $s = 3/2$, por lo tanto tenemos que

$$H^{3/2}(\partial\Omega) = \gamma_0\left(H^2(\Omega)\right) \tag{7.32}$$

$$\|g\|_{H^{3/2}(\partial\Omega)} = \inf_{v\in H^2(\Omega) \text{ y } \gamma_0 v = g} \|v\|_{H^2(\Omega)}. \tag{7.33}$$

Esto puede generalizarse a las trazas de derivadas de orden alto. Por ejemplo, si la frontera $\partial\Omega$ es suficientemente suave, podemos definir $\frac{\partial v}{\partial n}_{|\partial\Omega} \in H^{1/2}(\partial\Omega)$ para $v \in H^2(\Omega)$.

Por otro lado, para ejemplificar algunos casos de H_0^m notemos que

$$\begin{aligned} H_0^1(\Omega) &= \left\{v \mid v \in H^1(\Omega), v_{|\partial\Omega} = 0\right\} \\ H_0^2(\Omega) &= \left\{v \mid v \in H^2(\Omega), v_{|\partial\Omega} = 0 \text{ y } \frac{\partial v}{\partial n}_{|\partial\Omega} = 0\right\}. \end{aligned} \tag{7.34}$$

Además, en algunas ocasiones necesitamos considerar funciones que se nulifican en alguna parte de la frontera, supongamos que $\partial\Omega = D \cup N$, donde D es frontera tipo Dirichlet y N es frontera tipo Neumann y $D \cap N = \emptyset$, entonces podemos definir

$$H_{0,D}^1(\Omega) = \left\{v \mid v \in H^1(\Omega), v_{|_D} = 0\right\} \tag{7.35}$$

y donde tenemos que $H_0^1(\Omega) \subset H_{0,D}^1(\Omega) \subset H^1(\Omega)$.

7.2.3 Espacios $H(div, \Omega)$

Definición 29 *Sea Ω un dominio acotado en $\mathbb{R}^n$. Definimos a $\left(L^2(\Omega)\right)^n$ al espacio*

$$\left(L^2(\Omega)\right)^n = \left\{grad\ H^1(\Omega)\right\} \oplus \left\{rot\ H_0^1(\Omega)\right\}. \tag{7.36}$$

Definición 30 *Sea Ω un dominio acotado en $\mathbb{R}^n$. Definimos por $H(div,\Omega)$ al espacio*

$$H(div,\Omega)=\left\{\underline{q}\mid \underline{q}\in\left(L^2(\Omega)\right)^n,\ div\ \underline{q}\in L^2(\Omega)\right\}. \tag{7.37}$$

Definición 31 *La norma $\|\cdot\|^2_{H(div,\Omega)}$ de $H(div,\Omega)$, se define como*

$$\left\|\underline{q}\right\|^2_{H(div,\Omega)}=\left\|\underline{q}\right\|^2_{0,\Omega}+\left\|div\ \underline{q}\right\|^2_{0,\Omega}. \tag{7.38}$$

Cuando $H(div,\Omega)$ es equipada con la norma $\|\cdot\|^2_{H(div,\Omega)}$ el correspondiente producto interior se convierte en un espacio de Hilbert.

Notemos que, si Ω es un dominio acotado en $\mathbb{R}^n$, con frontera suave $\partial\Omega$, si $\underline{n}$ es un vector normal a $\partial\Omega$ y sea $\underline{q}\in H(div,\Omega)$, entonces los vectores de $H(div,\Omega)$ admiten una norma de la traza sobre $\partial\Omega$. Esta norma de la traza $\underline{q}\cdot\underline{n}$ pertenece a $H^{-1/2}(\partial\Omega)$ y esto se sigue de la fórmula de integración por partes

$$\int_\Omega \underline{q}\cdot\text{grad}\ v dx+\int_\Omega \text{div}\ \underline{q}v dx=\left\langle v,\underline{q}\cdot\underline{n}\right\rangle_{H^{1/2}(\partial\Omega)\times H^{-1/2}(\partial\Omega)} \tag{7.39}$$

para toda $\underline{q}\in H(div,\Omega)$ y cualquier $v\in H^1(\Omega)$. Pudiendo escribir formalmente $\int_{\partial\Omega} v\underline{q}\cdot\underline{n}ds$ en lugar del producto dual $\left\langle v,\underline{q}\cdot\underline{n}\right\rangle$.

Lema 32 *Sea $\underline{q}\in H(div,\Omega)$, podemos definir $\underline{q}\cdot\underline{n}_{|\partial\Omega}\in H^{-1/2}(\partial\Omega)$ y por la fórmula de Green*

$$\int_\Omega div\ \underline{q}v dx+\int_\Omega \underline{q}\cdot grad\ v dx=\left\langle v,\underline{q}\cdot\underline{n}\right\rangle \tag{7.40}$$

para toda $v\in H^1(\Omega)$.

Lema 33 *La traza del operador $\underline{q}\in H(div,\Omega)\rightarrow \underline{q}\cdot\underline{n}_{|\partial\Omega}\in H^{-1/2}(\partial\Omega)$ es suprayectiva.*

Sea Ω un dominio con frontera suave $\partial\Omega$, además supongamos que es frontera tipo Neumann $N=\partial\Omega$, entonces podemos definir

$$H_{0,N}(div,\Omega)=\left\{\underline{q}\mid \underline{q}\in H(div,\Omega),\left\langle v,\underline{q}\cdot\underline{n}\right\rangle=0,\forall v\in H^1_{0,D}(\Omega)\right\}. \tag{7.41}$$

este espacio contiene funciones del espacio $H(div,\Omega)$ cuyas trazas normales se nulifican en la frontera N.

Definición 34 *Un subespacio importante de* $H(div,\Omega)$ *es* $N^0(div,\Omega)$, *que se define como*

$$N^0(div,\Omega) = \left\{ \underline{q} \mid \underline{q} \in H(div,\Omega),\ div\ \underline{q} = 0 \right\}. \tag{7.42}$$

Lema 35 *El operador de traza normal* $\underline{q} \to \underline{q}\cdot\underline{n}_{|\partial\Omega}$ *es una forma suprayectiva* $N^0(div,\Omega)$ *sobre* $\left\{ \mu \mid \mu \in H^{-1/2}(\partial\Omega), \langle \mu, 1\rangle = 0 \right\}$.

7.3 Fórmulas de Green y Problemas Adjuntos

Una cuestión central en la teoría de problemas elípticos con valores en la frontera se relaciona con las condiciones bajo las cuales uno puede esperar una única solución a problemas de la forma

$$\begin{aligned} \mathcal{L}u &= f_\Omega \quad \text{en } \Omega \subset \mathbb{R}^n \\ &\left.\begin{aligned} B_o u &= g_0 \\ B_1 u &= g_1 \\ &\vdots \\ B_{m-1} u &= g_{m-1} \end{aligned}\right\} \quad \text{en } \partial\Omega \end{aligned} \tag{7.43}$$

donde $\mathcal{L}$ es un operador elíptico de orden $2m$, de forma

$$\mathcal{L}u = \sum_{|\alpha|\le m} (-1)^{|\alpha|} D^\alpha \left(\sum_{|\beta|\le m} a_{\alpha\beta}(\underline{x}) D^\beta u \right), \quad \underline{x} \in \Omega \subset \mathbb{R}^n \tag{7.44}$$

donde los coeficientes $a_{\alpha\beta}$ son funciones de $\underline{x}$ suaves y satisfacen las condiciones para que el operador sea elíptico, el conjunto $B_0, B_1, ..., B_{m-1}$ de operadores de frontera son de la forma

$$B_j u = \sum_{|\alpha|\le q_j} b_\alpha^{(j)} D^\alpha u = g_j \tag{7.45}$$

y constituyen un conjunto de condiciones de frontera que cubren a $\mathcal{L}$. Los coeficientes $b_\alpha^{(j)}$ son asumidos como funciones suaves.

En el caso de problemas de segundo orden la Ec. (7.45) puede expresarse como una sola condición de frontera

$$Bu = \sum_{j=1}^{n} b_j \frac{\partial u}{\partial x_j} + cu = g \quad \text{en } \partial\Omega. \tag{7.46}$$

Antes de poder ver las condiciones bajo las cuales se garantice la existencia y unicidad es necesario introducir el concepto de fórmula de Green asociada con el operador $\mathcal{L}^*$, para ello definimos:

Definición 36 *Con el operador dado como en la Ec. (7.44), denotaremos por $\mathcal{L}^*$ al operador definido por*

$$\mathcal{L}^* u = \sum_{|\alpha|\leq m} (-1)^{|\alpha|} D^\alpha \left(\sum_{|\beta|\leq m} a_{\beta\alpha}(\underline{x}) D^\beta u \right) \tag{7.47}$$

y nos referiremos a $\mathcal{L}^*$ como el adjunto formal del operador $\mathcal{L}$.

La importancia del adjunto formal es que si aplicamos el teorema de Green (89) a la integral

$$\int_\Omega v\mathcal{L}u d\underline{x} \tag{7.48}$$

obtenemos

$$\int_\Omega v\mathcal{L}u d\underline{x} = \int_\Omega u\mathcal{L}^* v d\underline{x} + \int_{\partial\Omega} F(u,v) d\underline{s} \tag{7.49}$$

en la cual $F(u,v)$ representa términos de frontera que se nulifican al aplicar el teorema ya que la función $v \in H_0^1(\Omega)$. Si $\mathcal{L} = \mathcal{L}^*$; i.e. $a_{\alpha\beta} = a_{\beta\alpha}$ el operador es llamado de manera formal el auto-adjunto.

En el caso de problemas de segundo orden, dos sucesivas aplicaciones del teorema de Green (89) y obtenemos, para i y j fijos

$$\begin{aligned} -\int_\Omega v \frac{\partial}{\partial x_i}\left(a_{ij}\frac{\partial u}{\partial x_j}\right) d\underline{x} &= -\int_{\partial\Omega} v a_{ij}\frac{\partial u}{\partial x_j} n_i d\underline{s} + \int_\Omega a_{ij}\frac{\partial u}{\partial x_j}\frac{\partial v}{\partial x_i} d\underline{x} \\ &= -\int_{\partial\Omega}\left[v a_{ij}\frac{\partial u}{\partial x_j} n_i - u a_{ij}\frac{\partial v}{\partial x_i} n_j\right] d\underline{s} \\ &\quad -\int_\Omega u\frac{\partial}{\partial x_j}\left(a_{ij}\frac{\partial v}{\partial x_i}\right) d\underline{x}. \end{aligned} \tag{7.50}$$

Pero sumando sobre i y j, obtenemos de la Ec. (7.49)

$$\mathcal{L}^* v = -\sum_{i,j=1}^{n} \frac{\partial}{\partial x_i}\left(a_{ji}(\underline{x})\frac{\partial v}{\partial x_j}\right) \tag{7.51}$$

y

$$F(u,v) = -\sum_{i,j=1}^{n} a_{ij}\left(v\frac{\partial u}{\partial x_j}n_i - u\frac{\partial v}{\partial x_i}n_j\right) \tag{7.52}$$

tal que $\mathcal{L}$ es formalmente el auto-ajunto si $a_{ji} = a_{ij}$.

Para hacer el tratamiento más simple, restringiremos nuestra atención al problema homogéneo, es decir, en el cual $g_0, g_1, ..., g_{m-1} = 0$ (esta no es una restricción real, ya que se puede demostrar que cualquier problema no-homogéneo con condiciones de frontera puede convertirse en uno con condiciones de frontera homogéneo de una manera sistemática), asumiremos también que Ω es suave y la frontera $\partial\Omega$ de Ω es de clase C^∞.

Así, en lo que resta de la sección, daremos los pasos necesarios para poder conocer bajo que condiciones el problema elíptico con valores en la frontera del tipo

$$\begin{aligned} \mathcal{L}u &= f_\Omega \quad \text{en } \Omega \subset \mathbb{R}^n \\ &\left.\begin{array}{c} B_o u = 0 \\ B_1 u = 0 \\ \vdots \\ B_{m-1} u = 0 \end{array}\right\} \quad \text{en } \partial\Omega \end{aligned} \tag{7.53}$$

donde el operador $\mathcal{L}$ y B_j están dados como en (7.44) y (7.45), con $s \geq 2m$ tiene solución y esta es única. Para ello, necesitamos adoptar el lenguaje de la teoría de operadores lineales, algunos resultados clave de álgebra lineal están detallados en el apéndice.

Primeramente denotemos $N(B_j)$ al espacio nulo del operador de frontera $B_j : H^s(\Omega) \to L^2(\Omega)$, entonces

$$N(B_j) = \{u \in H^s(\Omega) \mid B_j u = 0 \text{ en } \partial\Omega\} \tag{7.54}$$

para $j = 0, 1, 2, ..., m-1$.

Adicionalmente definimos al dominio del operador $\mathcal{L}$, como el espacio

$$\begin{aligned} D(\mathcal{L}) &= H^s(\Omega) \cap N(B_0) \cap ... \cap N(B_{m-1}) \\ &= \{u \in H^s(\Omega) \mid B_j u = 0 \text{ en } \partial\Omega, j = 0, 1, .., m-1\}. \end{aligned} \tag{7.55}$$

Entonces el problema elíptico con valores en la frontera de la Ec. (7.53) con $s \geq 2m$, puede reescribirse como, dado $\mathcal{L} : D(\mathcal{L}) \to H^{s-2m}(\Omega)$, hallar u que satisfaga

$$\mathcal{L}u = f_\Omega \quad \text{en } \Omega. \tag{7.56}$$

Lo primero que hay que determinar es el conjunto de funciones f_Ω en $H^{s-2m}(\Omega)$ para las cuales la ecuación anterior se satisface, i.e. debemos identificar el rango $R(\mathcal{L})$ del operador $\mathcal{L}$. Pero como nos interesa conocer bajo qué condiciones la solución u es única, entonces podemos definir el núcleo $N(\mathcal{L})$ del operador $\mathcal{L}$ como sigue

$$\begin{aligned} N(\mathcal{L}) &= \{u \in D(\mathcal{L}) \mid \mathcal{L}u = 0\} \\ &= \{u \in H^s(\Omega) \mid \mathcal{L}u = 0 \quad \text{en } \Omega, B_j u = 0 \text{ en } \partial\Omega, j = 0, 1, .., m-1\}. \end{aligned} \tag{7.57}$$

Si el $N(\mathcal{L}) \neq \{0\}$, entonces no hay una única solución, ya que si u_0 es una solución, entonces $u_0 + w$ también es solución para cualquier $w \in N(\mathcal{L})$, ya que

$$\mathcal{L}(u_0 + w) = \mathcal{L}u_0 + \mathcal{L}w = \mathcal{L}u_0 = f_\Omega. \tag{7.58}$$

Así, los elementos del núcleo $N(\mathcal{L})$ de $\mathcal{L}$ deberán ser excluidos del dominio $D(\mathcal{L})$ del operador $\mathcal{L}$, para poder asegurar la unicidad de la solución u.

Si ahora, introducimos el complemento ortogonal $N(\mathcal{L})^\perp$ del núcleo $N(\mathcal{L})$ del operador $\mathcal{L}$ con respecto al producto interior L^2, definiéndolo como

$$N(\mathcal{L})^\perp = \{v \in D(\mathcal{L}) \mid (v, w) = 0 \ \forall w \in N(\mathcal{L})\}. \tag{7.59}$$

De esta forma tenemos que

$$D(\mathcal{L}) = N(\mathcal{L}) \oplus N(\mathcal{L})^\perp \tag{7.60}$$

i.e. para toda $u \in D(\mathcal{L})$, u se escribe como $u = v + w$ donde $v \in N(\mathcal{L})^\perp$ y $w \in N(\mathcal{L})$. Además $N(\mathcal{L}) \cap N(\mathcal{L})^\perp = \{0\}$.

De forma similar, podemos definir los espacios anteriores para el problema adjunto

$$\begin{aligned} \mathcal{L}^* u &= f_\Omega \quad \text{en } \Omega \subset \mathbb{R}^n \\ &\left.\begin{array}{c} B_o^* u = 0 \\ B_1^* u = 0 \\ \vdots \\ B_{m-1}^* u = 0 \end{array}\right\} \quad \text{en } \partial\Omega \end{aligned} \tag{7.61}$$

y definimos

$$\begin{aligned} D(\mathcal{L}^*) &= H^s(\Omega) \cap N(B_0^*) \cap ... \cap N(B_{m-1}^*) \\ &= \left\{u \in H^s(\Omega) \mid B_j^* u = 0 \text{ en } \partial\Omega, j = 0, 1, .., m-1\right\}. \end{aligned} \tag{7.62}$$

Entonces el problema elíptico con valores en la frontera de la Ec. (7.53) con $s \geq 2m$, puede reescribirse como, dado $\mathcal{L}^* : D(\mathcal{L}^*) \to H^{s-2m}(\Omega)$, hallar u que satisfaga

$$\mathcal{L}^* u = f_\Omega \quad \text{en } \Omega. \tag{7.63}$$

Definiendo para el operador $\mathcal{L}^*$

$$\begin{aligned} N(\mathcal{L}^*) &= \{u \in D(\mathcal{L}^*) \mid \mathcal{L}^* u = 0\} \\ &= \left\{u \in H^s(\Omega) \mid \mathcal{L}^* u = 0 \quad \text{en } \Omega, B_j^* u = 0 \text{ en } \partial\Omega, j = 0, 1, .., m-1\right\}. \end{aligned} \tag{7.64}$$

y

$$N(\mathcal{L}^*)^\perp = \{v \in D(\mathcal{L}^*) \mid (v, w)_{L^2} = 0 \ \forall w \in N(\mathcal{L}^*)\}. \tag{7.65}$$

Así, con estas definiciones, es posible ver una cuestión fundamental, esta es, conocer bajo qué condiciones el problema elíptico con valores en la frontera de la Ec. (7.53) con $s \geq 2m$ tiene solución y esta es única, esto queda resuelto en el siguiente teorema cuya demostración puede verse en [49] y [45].

Teorema 37 *Considerando el problema elíptico con valores en la frontera de la Ec. (7.53) con $s \geq 2m$ definido sobre un dominio Ω acotado con frontera $\partial\Omega$ suave. Entonces*

i) Existe al menos una solución si y sólo si $f \in N(\mathcal{L}^*)^\perp$, esto es, si

$$(f, v)_{L^2(\Omega)} = 0 \quad \forall v \in N(\mathcal{L}^*). \tag{7.66}$$

ii) Asumiendo que la solución u existe, esta es única si $u \in N(\mathcal{L})^\perp$, esto es, si

$$(u, w)_{L^2(\Omega)} = 0 \quad \forall w \in N(\mathcal{L}). \tag{7.67}$$

iii) Si existe una única solución, entonces existe una única constante $C > 0$, independiente de u, tal que

$$\|u\|_{H^s} \leq C \|f\|_{H^{s-2m}}. \tag{7.68}$$

Observación 4 *i) El teorema afirma que el operador $\mathcal{L}$ es un operador suprayectivo de $D(\mathcal{L})$ sobre el subespacio de funciones en H^{s-2m} que satisface (7.67). Además el operador $\mathcal{L}$ es inyectivo si el dominio es restringido al espacio de funciones que satisfagan a (7.66).*

ii) La parte (iii) del teorema puede interpretarse como un resultado de regularidad, en el sentido en que se muestra

$$u \in H^{s-2m}(\Omega) \ \text{ si } f \in H^s(\Omega). \tag{7.69}$$

Así, formalmente podemos definir el adjunto formal de la siguiente manera

Definición 38 *Sea $\mathcal{L}$ un Operador Diferencial, decimos que un operador $\mathcal{L}^*$ es su adjunto formal si satisface la siguiente condición*

$$w\mathcal{L}u - u\mathcal{L}^*w = \nabla \cdot \underline{\mathfrak{D}}\,(u, w) \tag{3.1}$$

tal que las funciones u y w pertenecen a un espacio lineal. Aquí $\underline{\mathfrak{D}}(u, w)$ es una funcional bilineal que representa términos de frontera.

Ejemplos de Operadores Adjuntos Formales A continuación se muestra mediante ejemplos el uso de la definición de operadores adjuntos formales y la parte correspondiente a términos de frontera.

A) **Operador de la derivada de orden cero**

La derivada de orden cero de una función u es tal que

$$\frac{d^n u}{dx^n} = u \tag{7.70}$$

es decir, $n = 0$, sea el operador

$$\mathcal{L}u = u \tag{7.71}$$

de la definición de operador adjunto tenemos que

$$w\mathcal{L}u = u\mathcal{L}^*w + \nabla \cdot \underline{\mathfrak{D}}\,(u, w) \tag{7.72}$$

entonces el término izquierdo es

$$w\mathcal{L}u = wu \tag{7.73}$$

de aquí

$$u\mathcal{L}^*w = uw \tag{7.74}$$

por lo tanto el operador adjunto formal es

$$\mathcal{L}^*w = w \tag{7.75}$$

nótese que el operador es auto-adjunto.

B) **Operador de la derivada de primer orden**

La derivada de primer orden en términos del operador es

$$\mathcal{L}u = c\frac{du}{dx} \tag{7.76}$$

de la definición de operador adjunto tenemos

$$w\mathcal{L}u = u\mathcal{L}w + \nabla \cdot \underline{\mathfrak{D}}\,(u, w) \tag{7.77}$$

desarrollando el lado izquierdo

$$\begin{aligned} w\mathcal{L}u &= wc\frac{du}{dx} \\ &= \frac{d\,(wcu)}{dx} - u\frac{d\,(cw)}{dx} \\ &= \frac{d\,(wcu)}{dx} - uc\frac{dw}{dx} \end{aligned} \tag{7.78}$$

por lo tanto, el operador adjunto formal es

$$\mathcal{L}^* w = -c\frac{dw}{dx} \tag{7.79}$$

y los términos de frontera son

$$\mathfrak{D}\,(u, w) = wcu \tag{7.80}$$

C) **Operador Elíptico**

El operador elíptico más sencillo es el Laplaciano

$$\mathcal{L}u \equiv -\triangle u = -\frac{\partial}{\partial x_i}\left(\frac{\partial u}{\partial x_i}\right) \tag{7.81}$$

de la ecuación del operador adjunto formal tenemos

$$\begin{aligned} w\mathcal{L}u &= -w\frac{\partial}{\partial x_i}\left(\frac{\partial u}{\partial x_i}\right) \\ &= -\frac{\partial}{\partial x_i}\left(w\frac{\partial u}{\partial x_i}\right) + \frac{\partial u}{\partial x_i}\frac{\partial w}{\partial x_i} \\ &= -\frac{\partial}{\partial x_i}\left(w\frac{\partial u}{\partial x_i}\right) + \frac{\partial}{\partial x_i}\left(u\frac{\partial w}{\partial x_i}\right) - u\frac{\partial}{\partial x_i}\left(\frac{\partial w}{\partial x_i}\right) \\ &= \frac{\partial}{\partial x_i}\left(u\frac{\partial w}{\partial x_i} - w\frac{\partial u}{\partial x_i}\right) - u\frac{\partial}{\partial x_i}\left(\frac{\partial w}{\partial x_i}\right) \end{aligned} \tag{7.82}$$

entonces, el operador adjunto formal es

$$\mathcal{L}^* w = -u \frac{\partial}{\partial x_i} \left(\frac{\partial w}{\partial x_i} \right) \tag{7.83}$$

es decir, el operador es autoadjunto. Notemos que la función bilineal $\underline{\mathfrak{D}}(u, w)$ es

$$\underline{\mathfrak{D}}\,(u, w) = u \frac{\partial w}{\partial x_i} - w \frac{\partial u}{\partial x_i}. \tag{7.84}$$

D) Consideremos el operador diferencial elíptico más general de segundo orden

$$\mathcal{L}u = -\nabla \cdot \left(\underline{\underline{a}} \cdot \nabla u \right) + \nabla \cdot (\underline{b} u) + cu \tag{7.85}$$

de la definición de operador adjunto formal tenemos que

$$w\mathcal{L}u = u\mathcal{L}^* w + \nabla \cdot \underline{\mathfrak{D}}\,(u, w) \tag{7.86}$$

desarrollando el lado derecho de la ecuación anterior

$$\begin{aligned} w\mathcal{L}u &= w\left(-\nabla \cdot \left(\underline{\underline{a}} \cdot \nabla u \right) + \nabla \cdot (\underline{b} u) + cu\right) \\ &= -w\nabla \cdot \left(\underline{\underline{a}} \cdot \nabla u \right) + w\nabla \cdot (\underline{b} u) + wcu \end{aligned} \tag{7.87}$$

aplicando la igualdad de divergencia a los dos primeros sumandos se tiene que la ecuación anterior es

$$\begin{aligned} w\mathcal{L}u &= -\nabla \cdot \left(w\underline{\underline{a}} \cdot \nabla u \right) + \underline{\underline{a}} \cdot \nabla u \cdot \nabla w + \nabla \cdot (w\underline{b} u) \\ &\quad -\underline{b} u \cdot \nabla w + wcu \\ &= -\nabla \cdot \left(w\underline{\underline{a}} \cdot \nabla u \right) + \nabla \cdot \left(u\underline{\underline{a}} \nabla w \right) - u\nabla \cdot \left(\underline{\underline{a}} \cdot \nabla w \right) + \nabla \cdot (w\underline{b} u) \\ &\quad -\underline{b} u \cdot \nabla w + wcu \\ &= \nabla \cdot \left[\underline{\underline{a}} \left(u\nabla w - w\nabla u \right) \right] + \nabla \cdot (w\underline{b} u) - u\nabla \cdot \left(\underline{\underline{a}} \cdot \nabla w \right) \\ &\quad -\underline{b} u \cdot \nabla w + wcu \end{aligned} \tag{7.88}$$

reordenando términos se tiene

$$\begin{aligned} w\mathcal{L}u &= -u\nabla \cdot \left(\underline{\underline{a}} \cdot \nabla w \right) - u\underline{b} \cdot \nabla w + ucw + \\ &\quad \nabla \cdot \left[\underline{\underline{a}} \left(u\nabla w - w\nabla u \right) + (w\underline{b} u) \right] \end{aligned} \tag{7.89}$$

por lo tanto, el operador adjunto formal es

$$\mathcal{L}^* w = -\nabla \cdot \left(\underline{\underline{a}} \cdot \nabla w \right) - \underline{b} \cdot \nabla w + cw \tag{7.90}$$

y el término correspondiente a valores en la frontera es

$$\underline{\mathfrak{D}}\,(u,w) = \underline{\underline{a}} \cdot (u\nabla w - w\nabla u) + (w\underline{b}u)\,. \tag{7.91}$$

E) **La ecuación bi-armónica**

Consideremos el operador diferencial bi-armónico

$$\mathcal{L}u = \Delta^2 u \tag{7.92}$$

entonces se tiene que

$$w\mathcal{L}u = u\mathcal{L}^* w + \nabla \cdot \underline{\mathfrak{D}}\,(u,w) \tag{7.93}$$

desarrollemos el término del lado derecho

$$\begin{aligned} w\mathcal{L}u &= w\Delta^2 u \\ &= w\nabla\cdot(\nabla\Delta u) \end{aligned} \tag{7.94}$$

utilizando la igualdad de divergencia

$$\nabla\cdot(sV) = s\nabla\cdot V + V\cdot\nabla s \tag{7.95}$$

tal que s es función escalar y V vector, entonces sea $w = s$ y $\nabla\Delta u = V$, se tiene

$$\begin{aligned} & w\nabla\cdot(\nabla\Delta u) \\ = \;& \nabla\cdot(w\nabla\Delta u) - \nabla\Delta u\cdot\nabla w \end{aligned} \tag{7.96}$$

ahora sea $s = \Delta u$ y $V = \nabla w$, entonces

$$\begin{aligned} & \nabla\cdot(w\nabla\Delta u) - \nabla\Delta u\cdot\nabla w \\ = \;& \nabla\cdot(w\nabla\Delta u) + \Delta u\nabla\cdot\nabla w - \nabla\cdot(\Delta u\nabla w) \\ = \;& \Delta w\nabla\cdot\nabla u + \nabla\cdot(w\nabla\Delta u - \Delta u\nabla w) \end{aligned} \tag{7.97}$$

sea $s = \Delta w$ y $V = \nabla u$, entonces

$$\begin{aligned} & \Delta w\nabla\cdot\nabla u + \nabla\cdot(w\nabla\Delta u - \Delta u\nabla w) \\ = \;& \nabla\cdot(\Delta w\nabla u) - \nabla u\cdot\nabla(\Delta w) + \nabla\cdot(w\nabla\Delta u - \Delta u\nabla w) \\ = \;& -\nabla u\cdot\nabla(\Delta w) + \nabla\cdot(w\nabla\Delta u + \Delta w\nabla u - \Delta u\nabla w) \end{aligned} \tag{7.98}$$

por último sea $s = u$ y $V = \nabla(\Delta w)$ y obtenemos

$$\begin{aligned} & -\nabla u \cdot \nabla(\Delta w) + \nabla \cdot (w\nabla\Delta u + \Delta w \nabla u - \Delta u \nabla w) \qquad (7.99) \\ = \;& u\nabla \cdot (\nabla(\Delta w)) - \nabla \cdot (u\nabla(\Delta w)) + \nabla \cdot (w\nabla\Delta u + \Delta w\nabla u - \Delta u\nabla w) \end{aligned}$$

reordenando términos

$$w\mathcal{L}u = u\Delta^2 w + \nabla \cdot (w\nabla\Delta u + \Delta w\nabla u - \Delta u\nabla w - u\nabla\Delta w) \qquad (7.100)$$

entonces se tiene que el operador adjunto formal es

$$\mathcal{L}^* w = \Delta^2 w \qquad (7.101)$$

y los términos de frontera son

$$\underline{\mathfrak{D}}(u, w) = w\nabla\Delta u + \Delta w\nabla u - \Delta u\nabla w - u\nabla\Delta w. \qquad (7.102)$$

7.4 Adjuntos Formales para Sistemas de Ecuaciones

En esta sección trabajaremos con funciones vectoriales, para ello necesitamos plantear la definición de operadores adjuntos formales para este tipo de funciones.

Definición 39 *Sea $\underline{\underline{\mathcal{L}}}$ un operador diferencial, decimos que un operador $\underline{\underline{\mathcal{L}}}^*$ es su adjunto formal si satisface la siguiente condición*

$$\underline{w}\ \underline{\underline{\mathcal{L}}}\ \underline{u} - \underline{u}\ \underline{\underline{\mathcal{L}}}^*\underline{w} = \nabla \cdot \underline{\mathfrak{D}}(\underline{u}, \underline{w}) \qquad (7.103)$$

tal que las funciones $\underline{u}$ y $\underline{w}$ pertenecen a un espacio lineal. Aquí $\underline{\mathfrak{D}}(\underline{u}, \underline{w})$ representa términos de frontera.

Por lo tanto se puede trabajar con funciones vectoriales utilizando operadores matriciales.

A) **Operador diferencial vector-valuado con elasticidad estática**

Sea

$$\underline{\underline{\mathcal{L}}}\ \underline{u} = -\nabla \cdot \underline{\underline{\underline{C}}} : \nabla\underline{u} \qquad (7.104)$$

de la definición de operador adjunto formal tenemos que

$$\underline{w}\ \underline{\underline{\mathcal{L}}}\ \underline{u} = \underline{u}\ \underline{\underline{\mathcal{L}}}^*\underline{w} + \nabla \cdot \underline{\mathfrak{D}}(\underline{u}, \underline{w}) \qquad (7.105)$$

para hacer el desarrollo del término del lado derecho se utilizará notación indicial, es decir, este vector $\underline{w}\underline{\underline{\mathcal{L}}}\underline{u}$ tiene los siguientes componentes

$$-w_i\left(\frac{\partial}{\partial x_j}\left(C_{ijpq}\frac{\partial u_p}{\partial x_q}\right)\right);\ i=1,2,3 \tag{7.106}$$

utilizando la igualdad de divergencia tenemos

$$\begin{aligned} & -w_i\left(\frac{\partial}{\partial x_j}\left(C_{ijpq}\frac{\partial u_p}{\partial x_q}\right)\right) \qquad (7.107)\\ = \ & C_{ijpq}\frac{\partial u_p}{\partial x_q}\frac{\partial w_i}{\partial x_j}-\frac{\partial}{\partial x_j}\left(w_iC_{ijpq}\frac{\partial u_p}{\partial x_q}\right)\\ = \ & \frac{\partial}{\partial x_j}\left(u_iC_{ijpq}\frac{\partial w_i}{\partial x_j}\right)-u_i\frac{\partial}{\partial x_j}\left(C_{ijpq}\frac{\partial w_i}{\partial x_j}\right)-\frac{\partial}{\partial x_j}\left(w_iC_{ijpq}\frac{\partial u_p}{\partial x_q}\right) \end{aligned}$$

reordenado términos tenemos que la ecuación anterior es

$$\frac{\partial}{\partial x_j}\left(u_iC_{ijpq}\frac{\partial w_i}{\partial x_j}-w_iC_{ijpq}\frac{\partial u_p}{\partial x_q}\right)-u_i\frac{\partial}{\partial x_j}\left(C_{ijpq}\frac{\partial w_i}{\partial x_j}\right) \tag{7.108}$$

en notación simbólica tenemos que

$$\underline{w}\ \underline{\underline{\underline{\underline{\mathcal{L}}}}}\ \underline{u}=-\underline{u}\nabla\cdot\left(\underline{\underline{\underline{\underline{C}}}}:\nabla\underline{w}\right)+\nabla\cdot\left(\underline{u}\cdot\underline{\underline{\underline{\underline{C}}}}:\nabla\underline{w}-\underline{w}\cdot\underline{\underline{\underline{\underline{C}}}}:\nabla\underline{u}\right) \tag{7.109}$$

por lo tanto el operador adjunto formal es

$$\underline{\underline{\underline{\underline{\mathcal{L}}}}}^*\underline{w}=-\nabla\cdot\left(\underline{\underline{\underline{\underline{C}}}}:\nabla\underline{w}\right) \tag{7.110}$$

y los términos de frontera son

$$\underline{\mathfrak{D}}\left(\underline{u},\underline{w}\right)=\underline{u}\cdot\underline{\underline{\underline{\underline{C}}}}:\nabla\underline{w}-\underline{w}\cdot\underline{\underline{\underline{\underline{C}}}}:\nabla\underline{u} \tag{7.111}$$

El operador de elasticidad es **auto-adjunto formal.**

B) **Métodos Mixtos a la Ecuación de Laplace**

Operador Laplaciano

$$\underline{\underline{\mathcal{L}}}\ \underline{u}=\Delta\underline{u}=\underline{f} \tag{7.112}$$

escrito en un sistema de ecuaciones se obtiene

$$\underline{\underline{\mathcal{L}}}\,\underline{u} = \begin{bmatrix} 1 & -\nabla \\ \nabla\cdot & 0 \end{bmatrix} \begin{bmatrix} \underline{p} \\ u \end{bmatrix} = \begin{bmatrix} 0 \\ f \end{bmatrix} \tag{7.113}$$

consideraremos campos vectoriales de 4 dimensiones, estos son denotados por :

$$\underline{u} \equiv \left\{\underline{p}, u\right\} \text{ y } \underline{w} = \left\{\underline{q}, w\right\} \tag{7.114}$$

ahora el operador diferencial vector-valuado es el siguiente

$$\begin{aligned} \underline{\underline{\mathcal{L}}}\underline{u} &= \begin{bmatrix} 1 & -\nabla \\ \nabla\cdot & 0 \end{bmatrix} \cdot \begin{bmatrix} \underline{p} \\ u \end{bmatrix} \\ &= \begin{bmatrix} \underline{p} - \nabla u \\ \nabla \cdot \underline{p} \end{bmatrix} \end{aligned} \tag{7.115}$$

entonces

$$\underline{w}\,\underline{\underline{\mathcal{L}}}\,\underline{u} = \begin{bmatrix} \underline{q} \\ w \end{bmatrix} \cdot \begin{bmatrix} 1 & -\nabla \\ \nabla\cdot & 0 \end{bmatrix} \cdot \begin{bmatrix} \underline{p} \\ u \end{bmatrix} \tag{7.116}$$

utilizando la definición de operador adjunto

$$\underline{w}\underline{\underline{\mathcal{L}}}\underline{u} = \underline{u}\underline{\underline{\mathcal{L}}}\underline{w} + \nabla \cdot \underline{\mathfrak{D}}\left(\underline{u}, \underline{w}\right) \tag{7.117}$$

haciendo el desarrollo del término izquierdo se tiene que

$$\begin{aligned} \underline{w}\underline{\underline{\mathcal{L}}}\underline{u} &= \begin{bmatrix} \underline{q} \\ w \end{bmatrix} \cdot \begin{bmatrix} 1 & -\nabla \\ \nabla\cdot & 0 \end{bmatrix} \cdot \begin{bmatrix} \underline{p} \\ u \end{bmatrix} \\ &= \begin{bmatrix} \underline{q} \\ w \end{bmatrix} \cdot \begin{bmatrix} \underline{p} - \nabla u \\ \nabla \cdot \underline{p} \end{bmatrix} \\ &= \underline{q}\underline{p} - \underline{q}\nabla \cdot u + w\nabla \cdot \underline{p} \end{aligned} \tag{7.118}$$

aquí se utiliza la igualdad de divergencia en los dos términos del lado derecho y obtenemos

$$\begin{aligned} & \underline{q}\underline{p} - \underline{q}\nabla \cdot u + w\nabla \cdot \underline{p} \\ = \;& \underline{q}\underline{p} + u\nabla \cdot \underline{q} - \nabla \cdot \left(\underline{q}u\right) - \underline{p} \cdot \nabla w + \nabla \cdot \left(w\underline{p}\right) \\ = \;& \underline{p}\left(\underline{q} - \nabla w\right) + u\nabla \cdot \underline{q} + \nabla \cdot \left(w\underline{p} - u\underline{q}\right) \end{aligned} \tag{7.119}$$

si se agrupa los dos primeros términos en forma matricial, se tiene

$$\begin{aligned} & \underline{p}\left(\underline{q}-\nabla w\right)+u\nabla\cdot\underline{q}+\nabla\cdot\left(w\underline{p}-u\underline{q}\right) \\ = & \left[\begin{array}{c}\underline{p}\\ u\end{array}\right]\left[\begin{array}{c}\underline{q}-\nabla w\\ w\end{array}\right]+\nabla\cdot\left(w\underline{p}-u\underline{q}\right) \\ = & \left[\begin{array}{c}\underline{p}\\ u\end{array}\right]\cdot\left[\begin{array}{cc}1 & -\nabla\\ \nabla\cdot & 0\end{array}\right]\cdot\left[\begin{array}{c}\underline{q}\\ w\end{array}\right]+\nabla\cdot\left(w\underline{p}-u\underline{q}\right) \end{aligned} \tag{7.120}$$

por lo tanto, el operador adjunto formal es

$$\begin{aligned} \underline{\underline{\mathcal{L}}}^{*}\underline{w} &= \left[\begin{array}{cc}1 & -\nabla\\ \nabla\cdot & 0\end{array}\right]\cdot\left[\begin{array}{c}\underline{q}\\ w\end{array}\right] \\ &= \left[\begin{array}{c}\underline{q}-\nabla w\\ \nabla\cdot\underline{q}\end{array}\right] \end{aligned} \tag{7.121}$$

y el término correspondiente a valores en la frontera es

$$\underline{\mathfrak{D}}\left(\underline{u},\underline{w}\right)=w\underline{p}-u\underline{q}. \tag{7.122}$$

C) **Problema de Stokes**

El problema de Stokes es derivado de la ecuación de Navier-Stokes, la cual es utilizada en dinámica de fluidos viscosos. En este caso estamos suponiendo que el fluido es estacionario, la fuerza gravitacional es nula y el fluido incompresible**.** Entonces el sistema de ecuaciones a ser considerado es

$$\begin{aligned} -\Delta\underline{u}+\nabla p &= f \\ -\nabla\cdot\underline{u} &= 0 \end{aligned} \tag{7.123}$$

se considerará un campo vectorial de 4 dimensiones. Ellos serán denotados por

$$\underline{U}=\{\underline{u},p\}\ \text{ y }\ \underline{W}=\{\underline{w},q\} \tag{7.124}$$

ahora el operador diferencial vector-valuado es el siguiente

$$\underline{\underline{\mathcal{L}U}}=\left[\begin{array}{cc}-\Delta & \nabla\\ -\nabla\cdot & 0\end{array}\right]\cdot\left[\begin{array}{c}\underline{u}\\ p\end{array}\right] \tag{7.125}$$

el desarrollo se hará en notación indicial, entonces tenemos que

$$\underline{W}\underline{\underline{\mathcal{L}U}}=\left\{\begin{array}{c}-\underline{w}\Delta\underline{u}+\underline{w}\nabla p\\ -q\nabla\cdot\underline{u}\end{array}\right. \tag{7.126}$$

usando notación indicial se obtiene

$$\begin{aligned} w_i\left(-\sum_j \frac{\partial^2 u_i}{\partial x_j^2}+\frac{\partial p}{\partial x_i}\right) &= -\sum_j w_i\frac{\partial^2 u_i}{\partial x_j^2}+w_i\frac{\partial p}{\partial x_i} \\ &= \sum_j \frac{\partial w_i}{\partial x_j}\frac{\partial u_i}{\partial x_j^2}-\sum_j \frac{\partial}{\partial x_j}\left(w_i\frac{\partial u_i}{\partial x_j}\right)- \\ & \quad p\frac{\partial w_i}{\partial x_i}+\frac{\partial}{\partial x_i}\left(w_i p\right) \end{aligned} \tag{7.127}$$

desarrollando la primera suma como la derivada de dos funciones se tiene

$$\begin{aligned} &-\sum_j u_i\frac{\partial^2 w_i}{\partial x_j^2}+\sum_j \frac{\partial}{\partial x_j}\left(u_i\frac{\partial w_i}{\partial x_j}\right) \\ &-\sum_j \frac{\partial}{\partial x_j}\left(w_i\frac{\partial u_i}{\partial x_j}\right)-p\frac{\partial w_i}{\partial x_i}+\frac{\partial}{\partial x_i}\left(w_i p\right) \end{aligned} \tag{7.128}$$

reordenando términos tenemos

$$\begin{aligned} &-\sum_j u_i\frac{\partial^2 w_i}{\partial x_j^2}-p\frac{\partial w_i}{\partial x_i}+ \\ &\sum_j \frac{\partial}{\partial x_j}\left(u_i\frac{\partial w_i}{\partial x_j}-w_i\frac{\partial u_i}{\partial x_j}\right)+\frac{\partial}{\partial x_i}\left(w_i p\right) \end{aligned} \tag{7.129}$$

Ahora consideremos la ecuación 2 en Ec. (7.126), tenemos

$$-q\nabla\cdot\underline{u} \tag{7.130}$$

en notación índicial se tiene

$$\begin{aligned} -q\sum_i \frac{\partial u_i}{\partial x_i} &= -\sum_i q\frac{\partial u_i}{\partial x_i} \\ &= \sum_i u_i\frac{\partial q}{\partial x_i}-\sum_i \frac{\partial}{\partial x_i}\left(q u_i\right) \end{aligned} \tag{7.131}$$

en la ecuación anterior se utilizó la igualdad de divergencia, entonces

agrupando las ecuaciones Ec. (7.129) y Ec. (7.131) se tiene

$$\begin{aligned} & w_i\left(-\sum_j \frac{\partial^2 u_i}{\partial x_j^2}+\frac{\partial p}{\partial x_i}\right)-q\sum_i \frac{\partial u_i}{\partial x_i} \qquad (7.132)\\ = & -\sum_j u_i\frac{\partial^2 w_i}{\partial x_j^2}-p\frac{\partial w_i}{\partial x_i}+\\ & \sum_j \frac{\partial}{\partial x_j}\left(u_i\frac{\partial w_i}{\partial x_j}-w_i\frac{\partial u_i}{\partial x_j}\right)+\frac{\partial}{\partial x_i}\left(w_i p\right)+\\ & \sum_i u_i\frac{\partial q}{\partial x_i}-\sum_i \frac{\partial}{\partial x_i}\left(q u_i\right) \end{aligned}$$

ordenando los términos tenemos

$$\begin{aligned} & -\sum_j u_i\frac{\partial^2 w_i}{\partial x_j^2}+\sum_i u_i\frac{\partial q}{\partial x_i}-p\frac{\partial w_i}{\partial x_i} \qquad (7.133)\\ & +\sum_j \frac{\partial}{\partial x_j}\left(u_i\frac{\partial w_i}{\partial x_j}-w_i\frac{\partial u_i}{\partial x_j}\right)+\frac{\partial}{\partial x_i}\left(w_i p\right)-\sum_i \frac{\partial}{\partial x_i}\left(q u_i\right) \end{aligned}$$

escribiendo la ecuación anterior en notación simbólica, se obtiene

$$-\underline{u}\Delta\underline{w}+\underline{u}\nabla q-p\nabla\cdot\underline{w}+\nabla\cdot(\underline{u}\nabla\underline{w}-\underline{w}\nabla\underline{u}+\underline{w}p-\underline{u}q) \qquad (7.134)$$

por lo tanto, el operador adjunto formal es

$$\underline{\underline{\mathcal{L}}}^{*}\underline{W}=\begin{bmatrix} -\Delta & \nabla \\ -\nabla\cdot & 0 \end{bmatrix}\cdot\begin{bmatrix} \underline{w} \\ q \end{bmatrix} \qquad (7.135)$$

y el término de valores de frontera es

$$\underline{\mathfrak{D}}\left(\underline{u},\underline{w}\right)=\underline{u}\nabla\underline{w}-\underline{w}\nabla\underline{u}+\underline{w}p-\underline{u}q. \qquad (7.136)$$

7.5 Problemas Variacionales con Valor en la Frontera

Restringiéndonos ahora en problemas elípticos de orden 2 (problemas de orden mayor pueden ser tratados de forma similar), reescribiremos este en su forma variacional. La formulación variacional es más débil que la formulación

convencional ya que esta demanda menor suavidad de la solución u, sin embargo cualquier problema variacional con valores en la frontera corresponde a un problema con valor en la frontera y viceversa.

Además, la formulación variacional facilita el tratamiento de los problemas al usar métodos numéricos de ecuaciones diferenciales parciales, en esta sección veremos algunos resultados clave como es la existencia y unicidad de la solución de este tipo de problemas, para mayores detalles, ver [49] y [45].

Si el operador $\mathcal{L}$ está definido por

$$\mathcal{L}u = -\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu \tag{7.137}$$

con $\underline{\underline{a}}$ una matriz positiva definida, simétrica y $c \geq 0$, el problema queda escrito como

$$\begin{aligned} -\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu &= f_\Omega \quad \text{en } \Omega \\ u &= g \quad \text{en } \partial\Omega. \end{aligned} \tag{7.138}$$

Si multiplicamos a la ecuación $-\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu = f_\Omega$ por $v \in V = H_0^1(\Omega)$, obtenemos

$$-v\left(\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu\right) = v f_\Omega \tag{7.139}$$

aplicando el teorema de Green (89) obtenemos la Ec. (7.50), que podemos re-escribir como

$$\int_\Omega \left(\nabla v \cdot \underline{\underline{a}} \cdot \nabla u + cuv\right) d\underline{x} = \int_\Omega v f_\Omega d\underline{x}. \tag{7.140}$$

Definiendo el operador bilineal

$$a(u,v) = \int_\Omega \left(\nabla v \cdot \underline{\underline{a}} \cdot \nabla u + cuv\right) d\underline{x} \tag{7.141}$$

y la funcional lineal

$$l(v) = \langle f, v\rangle = \int_\Omega v f_\Omega d\underline{x} \tag{7.142}$$

podemos reescribir el problema dado por la Ec. (7.43) de orden 2, haciendo uso de la forma bilineal $a(\cdot,\cdot)$ y la funcional lineal $l(\cdot)$.

Entonces entenderemos en el presente contexto un problema variacional con valores de frontera (VBVP) por uno de la forma: hallar una función u que pertenezca a un espacio de Hilbert $V = H_0^1(\Omega)$ y que satisfaga la ecuación

$$a(u,v) = \langle f, v\rangle \tag{7.143}$$

para toda función $v \in V$ donde $a(\cdot,\cdot)$ es una forma bilineal y $l(\cdot)$ es una funcional lineal.

Definición 40 *Sea V un espacio de Hilbert y sea $\|\cdot\|_V$ la norma asociada a dicho espacio, decimos que una forma bilineal $a(\cdot,\cdot)$ es continua si existe una constante $M > 0$ tal que*

$$|a(u,v)| \leq M \|u\|_V \|v\|_V \quad \forall u, v \in V \tag{7.144}$$

y es V-elíptico si existe una constante $\alpha > 0$ tal que

$$a(v,v) \geq \alpha \|v\|_V^2 \quad \forall v \in V \tag{7.145}$$

donde $\|\cdot\|_V$ es la norma asociada al espacio V.

Esto significa que una forma $V-$ elíptico es una que siempre es no negativa y toma el valor de 0 sólo en el caso de que $v = 0$, i.e. es positiva definida.

Notemos que el problema (7.138) definido en $V = H_0^1(\Omega)$ reescrito como el problema (7.143) genera una forma bilineal V-elíptico cuyo producto interior sobre V es simétrico y positivo definido ya que

$$a(v,v) \geq \alpha \|v\|_V^2 > 0, \quad \forall v \in V, v \neq 0 \tag{7.146}$$

reescribiéndose el problema (7.143), en el cual debemos encontrar $u \in V$ tal que

$$a(u,v) = \langle f, v\rangle - a(u_0, v) \tag{7.147}$$

donde $u_0 = g$ en $\partial\Omega$, para toda $v \in V$.

Entonces, la cuestión fundamental, es conocer bajo qué condiciones el pro-blema anterior tiene solución y esta es única, el teorema de Lax-Milgram nos da las condiciones bajo las cuales el problema (7.138) reescrito como el problema (7.143) tiene solución y esta es única, esto queda plasmado en el siguiente resultado.

Teorema 41 *(Lax-Milgram)*

Sea V un espacio de Hilbert y sea $a(\cdot,\cdot) : V \times V \to \mathbb{R}$ una forma bilineal continua V-elíptico sobre V. Además, sea $l(\cdot) : V \to \mathbb{R}$ una funcional lineal continua sobre V. Entonces

i) El VBVP de encontrar $u \in V$ que satisfaga

$$a(u,v) = \langle f, v \rangle, \forall v \in V \tag{7.148}$$

tiene una y sólo una solución;

ii) La solución depende continuamente de los datos, en el sentido de que

$$\|u\|_V \leq \frac{1}{\alpha} \|l\|_{V^*} \tag{7.149}$$

donde $\|\cdot\|_{V^*}$ es la norma en el espacio dual V^* de V y α es la constante de la definición de V-elíptico.

Más específicamente, considerando ahora V un subespacio cerrado de $H^m(\Omega)$ las condiciones para la existencia, unicidad y la dependencia continua de los datos queda de manifiesto en el siguiente resultado.

Teorema 42 *Sea V un subespacio cerrado de $H^m(\Omega)$, sea $a(\cdot,\cdot) : V \times V \to \mathbb{R}$ una forma bilineal continua V-elíptico sobre V y sea $l(\cdot) : V \to \mathbb{R}$ una funcional lineal continua sobre V. Sea P un subespacio cerrado de V tal que*

$$a(u+p, v+\overline{p}) = a(u,v) \quad \forall u, v \in V \text{ y } p, \overline{p} \in P. \tag{7.150}$$

También denotando por Q el subespacio de V consistente de las funciones ortogonales a P en la norma L^2; tal que

$$Q = \left\{ v \in V \mid \int_\Omega upd\underline{x} = 0 \quad \forall p \in P \right\}, \tag{7.151}$$

y asumiendo que $a(\cdot,\cdot)$ es Q-elíptico: existe una constante $\alpha > 0$ tal que

$$a(q,q) \geq \alpha \|q\|_Q^2 \quad \text{para } q \in Q, \tag{7.152}$$

la norma sobre Q será la misma que sobre V. Entonces

i) Existe una única solución al problema de encontrar $u \in Q$ tal que

$$a(u,v) = \langle l, v \rangle, \quad \forall v \in V \tag{7.153}$$

si y sólo si las condiciones de compatibilidad

$$\langle l, p \rangle = 0 \quad \text{para } p \in P \tag{7.154}$$

se satisfacen.

ii) La solución u satisface

$$\|u\|_Q \leq \alpha^{-1} \|l\|_{Q^*} \tag{7.155}$$

(dependencia continua de los datos).

Otro aspecto importante es la regularidad de la solución, si la solución u al VBVP de orden $2m$ con $f \in H^{s-2m}(\Omega)$ donde $s \geq 2m$, entonces u pertenecerá a $H^s(\Omega)$ y esto queda de manifiesto en el siguiente resultado.

Teorema 43 *Sea $\Omega \subset \mathbb{R}^n$ un dominio suave y sea $u \in V$ la solución al VBVP*

$$a(u, v) = \langle f, v \rangle, \quad v \in V \tag{7.156}$$

donde $V \subset H^m(\Omega)$. Si $f \in H^{s-2m}(\Omega)$ con $s \geq 2m$, entonces $u \in H^s(\Omega)$ y la estimación

$$\|u\|_{H^s} \leq C \|f\|_{H^{s-2m}} \tag{7.157}$$

se satisface.

8 Apéndice D: Métodos de Solución Aproximada para EDP

En el presente capítulo se prestará atención a varios aspectos necesarios para encontrar la solución aproximada de problemas variacionales con valor en la frontera (VBVP). Ya que en general encontrar la solución a problemas con geometría diversa es difícil y en algunos casos imposible usando métodos analíticos.

En este capítulo se considera el VBVP de la forma

$$\begin{aligned} \mathcal{L}u &= f_\Omega \quad \text{en } \Omega \\ u &= g \quad \text{en } \partial\Omega \end{aligned} \tag{8.1}$$

donde

$$\mathcal{L}u = -\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu \tag{8.2}$$

con $\underline{\underline{a}}$ una matriz positiva definida, simétrica y $c \geq 0$, como un caso particular del operador elíptico definido por la Ec. (7.43) de orden 2, con $\Omega \subset R^2$ un dominio poligonal, es decir, Ω es un conjunto abierto acotado y conexo tal que su frontera $\partial\Omega$ es la unión de un número finito de polígonos.

La sencillez del operador $\mathcal{L}$ nos permite facilitar la comprensión de muchas de las ideas básicas que se expondrán a continuación, pero tengamos en mente que esta es una ecuación que gobierna los modelos de muchos sistemas de la ciencia y la ingeniería, por ello es muy importante su solución.

Si multiplicamos a la ecuación $-\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu = f_\Omega$ por $v \in V = H_0^1(\Omega)$, obtenemos

$$-v\left(\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu\right) = v f_\Omega \tag{8.3}$$

aplicando el teorema de Green (89) obtenemos la Ec. (7.50), que podemos re-escribir como

$$\int_\Omega \left(\nabla v \cdot \underline{\underline{a}} \cdot \nabla u + cuv\right) d\underline{x} = \int_\Omega v f_\Omega d\underline{x}. \tag{8.4}$$

Definiendo el operador bilineal

$$a(u,v) = \int_\Omega \left(\nabla v \cdot \underline{\underline{a}} \cdot \nabla u + cuv\right) d\underline{x} \tag{8.5}$$

y la funcional lineal

$$l(v) = \langle f, v \rangle = \int_{\Omega} v f_{\Omega} d\underline{x} \tag{8.6}$$

podemos reescribir el problema dado por la Ec. (8.1) de orden 2 en forma variacional, haciendo uso de la forma bilineal $a(\cdot,\cdot)$ y la funcional lineal $l(\cdot)$.

8.1 Método Galerkin

La idea básica detrás del método Galerkin es, considerando el VBVP, encontrar $u \in V = H_0^1(\Omega)$ que satisfaga

$$a(u, v) = \langle f, v \rangle \quad \forall v \in V \tag{8.7}$$

donde V es un subespacio de un espacio de Hilbert H (por conveniencia nos restringiremos a espacios definidos sobre los números reales).

El problema al tratar de resolver la Ec. (8.7) está en el hecho de que el espacio V es de dimensión infinita, por lo que resulta que en general no es posible encontrar el conjunto solución. En lugar de tener el problema en el espacio V, se supone que se tienen funciones linealmente independientes $\phi_1, \phi_2, ..., \phi_N$ en V y definimos el espacio V^h a partir del subespacio dimensionalmente finito de V generado por las funciones ϕ_i, es decir,

$$V^h = Generado\{\phi_i\}_{i=1}^{N}, \quad V^h \subset V. \tag{8.8}$$

El índice $h = 1/N$ es un parámetro que estará entre 0 y 1, cuya magnitud da alguna indicación de cuan cerca V^h está de V, h se relaciona con la dimensión de V^h. Y como el número N de las funciones base se escoge de manera que sea grande y haga que h sea pequeño, en el límite, cuando $N \to \infty$, $h \to 0$.

Después de definir el espacio V^h, es posible trabajar con V^h en lugar de V y encontrar una función u_h que satisfaga

$$a(u_h, v_h) = \langle f, v_h \rangle \quad \forall v_h \in V^h. \tag{8.9}$$

Esta es la esencia del método Galerkin, notemos que u_h y v_h son sólo combinaciones lineales de las funciones base de V^h, tales que

$$u_h = \sum_{i=1}^{N} c_i \phi_i \quad \text{y} \quad v_h = \sum_{j=1}^{N} d_j \phi_j \tag{8.10}$$

donde v_h es arbitraria, como los coeficientes de d_j y sin pérdida de generalidad podemos hacer $v_h = \phi_j$. Así, para encontrar la solución u_h sustituimos las Ecs. (8.10) en la Ec. (8.9) y usando el hecho que $a(\cdot,\cdot)$ es una forma bilineal y $l(\cdot)$ es una funcional lineal se obtiene la ecuación

$$\sum_{i=1}^{N} a\left(\phi_i, \phi_j\right) c_i = \left\langle f, \phi_j \right\rangle \tag{8.11}$$

o más concisamente, como

$$\sum_{i=1}^{N} K_{ij} c_i - F_j = 0 \quad j = 1, 2, ...N \tag{8.12}$$

en la cual

$$K_{ij} = a\left(\phi_i, \phi_j\right) \quad \text{y} \quad F_j = \left\langle f, \phi_j \right\rangle \tag{8.13}$$

notemos que tanto K_{ij} y F_j pueden ser evaluados, ya que ϕ_i, $a(\cdot,\cdot)$ y $l(\cdot)$ son conocidas.

Entonces el problema se reduce a resolver el sistema de ecuaciones lineales

$$\sum_{i=1}^{N} K_{ij} c_i - F_j, \quad j = 1, 2, ...N \tag{8.14}$$

o más compactamente

$$\underline{\underline{\mathbb{K}}} u = \underline{F} \tag{8.15}$$

en la cual $\underline{\underline{\mathbb{K}}}$ y $\underline{F}$ son la matriz y el vector cuyas entradas son K_{ij} y F_j. Una vez que el sistema es resuelto, la solución aproximada u_h es encontrada.

Notemos que la forma bilineal $a(\cdot,\cdot)$ define un producto interior sobre V, si $a(\cdot,\cdot)$ es simétrica y V -elíptica, entonces las propiedades de linealidad y simétria son obvias, mientras que la propiedad de V- elíticidad de $a(\cdot,\cdot)$ es por

$$a(v,v) \geq \alpha \|v\|^2 > 0 \qquad \forall v \neq 0, \tag{8.16}$$

además, si $a(\cdot,\cdot)$ es continua, entonces la norma $\|v\|_a \equiv a(v,v)$ generada por este producto interior es equivalente a la norma estándar sobre V, tal que si V es completa con respecto a la norma estándar, esta también es completa con respecto a la norma $\|v\|_a$.

Por otro lado, si el conjunto de funciones base $\{\phi_i\}_{i=1}^N$ se eligen de tal forma que sean sean ortogonales entre sí, entonces el sistema (8.12) se simplifica considerablemente, ya que

$$K_{ij} = a\left(\phi_i, \phi_j\right) = 0 \qquad \text{si } i \neq j \tag{8.17}$$

y

$$K_{ii}c_i = F_i \qquad ó \qquad c_i = F_i/K_{ii}. \tag{8.18}$$

Así, el problema (8.1) definido en $V^h = H_0^1(\Omega)$ reescrito como el problema (8.7) genera una forma bilineal V^h-elíptica cuyo producto interior sobre V^h es simétrico y positivo definido ya que

$$a(v_h, v_h) \geq \alpha \|v_h\|_{V^h}^2 > 0, \quad \forall v_h \in V^h, v_h \neq 0 \tag{8.19}$$

reescribiéndose el problema (8.9) como el problema aproximado en el cual debemos encontrar $u_h \in V^h \subset V$ tal que

$$a(u_h, v_h) = \langle f, v_h \rangle - a(u_0, v_h) \tag{8.20}$$

donde $u_0 = g = 0$ en $\partial\Omega$, para toda $v_h \in V^h$, es decir

$$\int_\Omega \left(\nabla v_h \cdot \underline{\underline{a}} \cdot \nabla u_h + c u_h v_h\right) dxdy = \int_\Omega f_\Omega v_h dxdy \tag{8.21}$$

para todo $v_h \in V^h$.

Entonces, el problema (8.1) al aplicarle el método Galerkin obtenemos (8.4), el cual podemos reescribirlo como (8.21). Aplicando el teorema de Lax-Milgram (41) a este caso particular, tenemos que este tiene solución única y esta depende continuamente de los datos.

Como un caso particular del teorema de Lax-Milgram (41) tenemos el sigui-ente resultado

Teorema 44 *Sea V^h un subespacio de dimensión finita de un espacio de Hilbert V, sea $a(\cdot,\cdot) : V^h \times V^h \to \mathbb{R}$ una forma bilineal continua y V-elíptica, y $l(\cdot) : V^h \to \mathbb{R}$ una funcional lineal acotada. Entonces existe una única función $u_h \in V^h$ tal que satisface*

$$a(u_h, v_h) = \langle l, v_h \rangle \qquad \forall v_h \in V^h. \tag{8.22}$$

Además, si $l(\cdot)$ *es de la forma*

$$\langle l, v_h \rangle = \int_\Omega f_\Omega v_h d\underline{x} \tag{8.23}$$

con $f \in L^2(\Omega)$, *entonces*

$$\|u_h\|_V \leq \frac{1}{\alpha} \|f\|_{L^2}, \tag{8.24}$$

donde α *es la constante en (8.16).*

El siguiente resultado nos da una condición suficiente para que la aproximación u_h del método Galerkin converja a la solución u del problema dado por la Ec. (8.7), para más detalle véase [45] y [49].

Teorema 45 *Sea* V *un subespacio cerrado de un espacio de Hilbert, y sea la forma bilineal* $a(\cdot,\cdot) : V^h \times V^h \to \mathbb{R}$ *continua* V*-elíptica y sea* $l(\cdot)$ *una funcional lineal acotada. Entonces existe una constante* C*, independiente de* h*, tal que*

$$\|u - u_h\|_V \leq C \inf_{v_h \in V^h} \|u - v_h\|_V \tag{8.25}$$

donde u *es solución de (8.7) y* u_h *es solución de (8.20), consecuentemente, una condición suficiente para que la aproximación* u_h *del método Galerkin converge a la solución* u *del problema dado por la Ec. (8.7) es que exista una familia* $\{V^h\}$ *de subespacios con la propiedad de que*

$$\inf_{v_h \in V^h} \|u - v_h\|_V \to 0 \quad \textit{cuando} \quad h \to 0. \tag{8.26}$$

8.2 El Método de Residuos Pesados

Este método se basa en el método Galerkin, y se escogen subespacios U^h y V^h de tal manera que la dimensión $\dim U^h = \dim V^h = N$, eligiendo las bases como

$$\{\phi_i\}_{i=1}^N \text{ para } U^h \text{ y } \{\psi_j\}_{j=1}^N \text{ para } V^h \tag{8.27}$$

entonces

$$u_h = \sum_{i=1}^N c_i \phi_i \text{ y } v_h = \sum_{j=1}^N b_j \psi_j \tag{8.28}$$

donde los coeficientes b_j son arbitrarios ya que v_h es arbitraria.

Sustituyendo esta última expresión Ec. (8.28) en

$$(\mathcal{L}u_h - f, v_h) = 0 \tag{8.29}$$

se obtienen N ecuaciones simultáneas

$$\sum_{i=1}^{N} K_{ij}c_i = F_j \text{ con } j = 1, ..., N$$

en la cual en la cual $\underline{\underline{\mathbb{K}}}$ y $\underline{F}$ son la matriz y el vector cuyas entradas son

$$K_{ij} = \left(\mathcal{L}\phi_i, \psi_j\right) \text{ y } F_j = \left(f, \psi_j\right)$$

donde $(\cdot, \cdot)$ representa el producto interior asociado a L^2. A la expresión

$$\tau(u_h) \equiv \mathcal{L}u_h - f \tag{8.30}$$

se le llama el residuo; si u_h es la solución exacta, entonces por supuesto el residuo se nulifica.

8.3 Método de Elementos Finitos

El método Finite Elements Method (FEM) provee una manera sistemática y simple de generar las funciones base en un dominio con geometría Ω poligonal. Lo que hace al método de elemento finito especialmente atractivo sobre otros métodos, es el hecho de que las funciones base son polinomios definidos por pedazos (elementos Ω_i) que son no cero sólo en una pequeña parte de Ω, proporcionando a la vez una gran ventaja computacional al método ya que las matrices generadas resultan bandadas ahorrando memoria al implantarlas en una computadora.

Así, partiendo del problema aproximado (8.21), se elegirá una familia de espacios $V^h (h \in (0, 1))$ definido por el procedimiento de elementos finitos (descritos en las subsecciones siguientes en el caso de interpoladores lineales, para otros tipos de interpoladores, ver [9]), teniendo la propiedad de que V^h se aproxima a V cuando h se aproxima a cero en un sentido apropiado, esto es, por supuesto una propiedad indispensable para la convergencia del método Galerkin.

Mallado del dominio El Mallado o triangulación $\mathcal{T}_h$ del dominio Ω es el primer aspecto básico, y ciertamente el más característico, el dominio $\Omega \subset \mathbb{R}^2$ es subdividido en E subdominios o elementos Ω_e llamados elementos finitos, tal que

$$\overline{\Omega} = \bigcup_{e=1}^{E} \overline{\Omega}_e$$

donde:

- Cada $\Omega_e \in \mathcal{T}_h$ es un polígono (rectángulo o triángulo) con interior no vacío ($\mathring{\Omega}_e \neq \emptyset$) y conexo.
- Cada $\Omega_e \in \mathcal{T}_h$ tiene frontera $\partial\Omega_e$ Lipschitz continua.
- Para cada $\Omega_i, \Omega_j \in \mathcal{T}_h$ distintos, $\mathring{\Omega}_i \cap \mathring{\Omega}_j = \emptyset$.
- El diámetro $h_i = Diam(\Omega_e)$ de cada Ω_e satisface $Diam(\Omega_e) \leq h$ para cada $e = 1, 2, ..., E$.
- Cualquier cara de cualquier elemento $\Omega_i \in \mathcal{T}_h$ en la triangulación es también un subconjunto de la frontera $\partial\Omega$ del dominio Ω o una cara de cualquier otro elemento $\Omega_j \in \mathcal{T}_h$ de la triangulación, en este último caso Ω_i y Ω_j son llamados adyacentes.
- Los vértices de cada Ω_e son llamados nodos, teniendo N de ellos por cada elemento Ω_e.

Una vez que la triangulación $\mathcal{T}_h$ del dominio Ω es establecida, se procede a definir el espacio de elementos finitos $\mathbb{P}^h[k]$ a través del proceso descrito a continuación.

Funciones Base A continuación describiremos la manera de construir las funciones base usadas por el método de elemento finito. En este procedimiento debemos tener en cuenta que las funciones base están definidas en un subespacio de $V = H^1(\Omega)$ para problemas de segundo orden que satisfacen las condiciones de frontera.

Las funciones base deberán satisfacer las siguientes propiedades:

- Las funciones base ϕ_i son acotadas y continuas, i.e $\phi_i \in C(\Omega_e)$.
- Existen ℓ funciones base por cada nodo del polígono Ω_e, y cada función ϕ_i es no cero solo en los elementos contiguos conectados por el nodo i.

- $\phi_i = 1$ en cada i nodo del polígono Ω_e y cero en los otros nodos.
- La restricción ϕ_i a Ω_e es un polinomio, i.e. $\phi_i \in \mathbb{P}_k[\Omega_e]$ para alguna $k \geq 1$ donde $\mathbb{P}_k[\Omega_e]$ es el espacio de polinomios de grado a lo más k sobre Ω_e.

Decimos que $\phi_i \in \mathbb{P}_k[\Omega_e]$ es una base de funciones y por su construcción es evidente que estas pertenecen a $H^1(\Omega)$. Al conjunto formado por todas las funciones base definidas para todo Ω_e de Ω será el espacio $\mathbb{P}^h[k]$ de funciones base, i.e.

$$\mathbb{P}^h[k] = \bigcup_{e=1}^{E} \mathbb{P}_k[\Omega_e]$$

estas formarán las funciones base globales.

Solución aproximada Para encontrar la solución aproximada elegimos el espacio $\mathbb{P}^h[k]$ de funciones base, como el espacio de funciones lineales ϕ_i definidas por pedazos de grado menor o igual a k (en nuestro caso $k = 1$), entonces el espacio a trabajar es

$$V^h = Generado\left\{\phi_i \in \mathbb{P}^h[k] \;\middle|\; \phi_i(x) = 0 \text{ en } \partial\Omega\right\}. \tag{8.31}$$

La solución aproximada de la Ec. (8.21) al problema dado por la Ec. (8.1) queda en términos de

$$\int_\Omega \left(\nabla\phi_i \cdot \underline{\underline{a}} \cdot \nabla\phi_j - c\phi_i\phi_j\right) dxdy = \int_\Omega f_\Omega \phi_j dxdy \tag{8.32}$$

si definimos el operador bilineal

$$K_{ij} \equiv a\left(\phi_i, \phi_j\right) = \int_\Omega \left(\nabla\phi_i \cdot a_{ij} \cdot \nabla\phi_j - c\phi_i\phi_j\right) dxdy \tag{8.33}$$

y la funcional lineal

$$F_j \equiv \left\langle f, \phi_j \right\rangle = \int_\Omega f_\Omega \phi_j dxdy \tag{8.34}$$

entonces la matriz $\underline{\underline{K}} \equiv [K_{ij}]$, los vectores $\underline{u} \equiv (u_1, ..., u_N)$ y $\underline{F} \equiv (F_1, ..., F_N)$ definen el sistema lineal (que es positivo definido)

$$\underline{\underline{K}}\,\underline{u} = \underline{F} \tag{8.35}$$

donde $\underline{u}$ será el vector solución a la Ec. (8.35) cuyos valores serán la solución al problema dado por la Ec. (8.21) que es la solución aproximada a la Ec. (8.1) en los nodos interiores de Ω.

Un Caso más General Sea el operador elíptico (caso simétrico) en el dominio Ω, y el operador definido por

$$\begin{aligned} \mathcal{L}u &= f_\Omega \quad \text{en } \Omega\backslash\Sigma \\ u &= g \quad \text{en } \partial\Omega \\ [u]_\Sigma &= J_0 \\ [a_n \cdot \nabla u]_\Sigma &= J_1 \end{aligned} \tag{8.36}$$

donde

$$\mathcal{L}u = -\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu \tag{8.37}$$

conjuntamente con una partición $\prod = \{\Omega_1, ..., \Omega_E\}$ de Ω. Multiplicando por la función w obtenemos

$$w\mathcal{L}u = -w\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cwu = wf_\Omega \tag{8.38}$$

entonces si $w(x)$ es tal que $[w] = 0$ (es decir w es continua) y definimos

$$a(u,w) = \sum_{i=1}^{E} \int_{\Omega_i} \left(\nabla u \cdot \underline{\underline{a}} \cdot \nabla w + cwu\right) d\underline{x} \tag{8.39}$$

tal que $a(u,w)$ define un producto interior sobre

$$H^1(\Omega) = H^1(\Omega_1) \oplus H^1(\Omega_2) \oplus .... \oplus H^1(\Omega_E).$$

Entonces, reescribimos la Ec. (8.38) como

$$\begin{aligned} a(u,w) &= \int_\Omega wf d\underline{x} + \sum_{i=1}^{E} \int_{\partial\Omega} wa_n \cdot \nabla u d\underline{s} \\ &= \int_\Omega wf_\Omega d\underline{x} + \int_{\partial\Omega} wa_n \cdot \nabla u d\underline{s} - \int_\Sigma w\left[a_n \cdot \nabla u\right]_\Sigma d\underline{s}. \end{aligned} \tag{8.40}$$

Sea $u_0(x)$ una función que satisface las condiciones de frontera y J_0 una función que satisface las condiciones de salto, tal que

i) $u_0(x) = g(x)$ en $\partial\Omega$

ii) $[u_0(x)]_\Sigma = J_0$

y sea $u(x) = u_0(x) + v(x)$. Entonces $u(x)$ satisface la Ec. (8.39) si y sólo si $v(x)$ satisface

$$a(u, w) = \int_\Omega w f_\Omega d\underline{x} - \langle u_0, w \rangle - \int_\Sigma J_1 w d\underline{s} \tag{8.41}$$

para toda w tal que $w(x) = 0$ en $\partial\Omega$. Sea $\{\phi_i\}$ una base de un subespacio de dimensión finita V^h definido como

$$V^h = \left\{ \phi_i \mid \phi_i \in C^1(\Omega_i), \forall i, \phi_i = 0 \text{ en } \partial\Omega \text{ y } \phi_i \in C^0(\Omega) \right\}. \tag{8.42}$$

La solución por elementos finitos de (8.41) se obtiene al resolver el sistema lineal

$$\underline{\underline{K}}\,\underline{u} = \underline{F} \tag{8.43}$$

donde

$$K_{ij} = a\left(\phi_i, \phi_j\right) \tag{8.44}$$

y

$$F_j = \int_\Omega \phi_j f_\Omega d\underline{x} - a\left(u_0, \phi_j\right) - \int_\Sigma J_1 \phi_j d\underline{s} \tag{8.45}$$

esta solución será la solución en los nodos interiores de Ω.En el siguiente capítulo se prestará atención a varios aspectos necesarios para encontrar la solución aproximada de problemas variacionales con valor en la frontera (VBVP). Ya que en general encontrar la solución a problemas con geometría diversa es difícil y en algunos casos imposible usando métodos analíticos.

9 Apéndice E: Método de Elementos Finitos

En este capítulo se considera el VBVP de la forma

$$\begin{aligned} \mathcal{L}u &= f_\Omega \quad \text{en } \Omega \\ u &= g \quad \text{en } \partial\Omega \end{aligned} \tag{9.1}$$

donde

$$\mathcal{L}u = -\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu \tag{9.2}$$

con $\underline{\underline{a}}$ una matriz positiva definida, simétrica y $c \geq 0$, como un caso particular del operador elíptico definido por la Ec. (7.43) de orden 2, con $\Omega \subset R^2$ un dominio poligonal, es decir, Ω es un conjunto abierto acotado y conexo tal que su frontera $\partial\Omega$ es la unión de un número finito de polígonos.

La sencillez del operador $\mathcal{L}$ nos permite facilitar la comprensión de muchas de las ideas básicas que se expondrán a continuación, pero tengamos en mente que esta es una ecuación que gobierna los modelos de muchos sistemas de la ciencia y la ingeniería, por ello es muy importante su solución.

Si multiplicamos a la ecuación $-\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu = f_\Omega$ por $v \in V = H_0^1(\Omega)$, obtenemos

$$-v\left(\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu\right) = v f_\Omega \tag{9.3}$$

aplicando el teorema de Green (89) obtenemos la Ec. (7.50), que podemos re-escribir como

$$\int_\Omega \left(\nabla v \cdot \underline{\underline{a}} \cdot \nabla u + cuv\right) d\underline{x} = \int_\Omega v f_\Omega d\underline{x}. \tag{9.4}$$

Definiendo el operador bilineal

$$a(u,v) = \int_\Omega \left(\nabla v \cdot \underline{\underline{a}} \cdot \nabla u + cuv\right) d\underline{x} \tag{9.5}$$

y la funcional lineal

$$l(v) = \langle f, v\rangle = \int_\Omega v f_\Omega d\underline{x} \tag{9.6}$$

podemos reescribir el problema dado por la Ec. (9.1) de orden 2 en forma variacional, haciendo uso de la forma bilineal $a(\cdot,\cdot)$ y la funcional lineal $l(\cdot)$.

9.1 Triangulación

El Mallado o triangulación $\mathcal{T}_h$ del dominio Ω es el primer aspecto básico, y ciertamente el más característico, el dominio $\Omega \subset \mathbb{R}^2$ es subdividido en E subdominios o elementos Ω_e llamados elementos finitos, tal que

$$\overline{\Omega} = \bigcup_{e=1}^{E} \overline{\Omega}_e$$

donde:

- Cada $\Omega_e \in \mathcal{T}_h$ es un polígono (rectángulo o triángulo) con interior no vacío ($\mathring{\Omega}_e \neq \emptyset$) y conexo.
- Cada $\Omega_e \in \mathcal{T}_h$ tiene frontera $\partial\Omega_e$ Lipschitz continua.
- Para cada $\Omega_i, \Omega_j \in \mathcal{T}_h$ distintos, $\mathring{\Omega}_i \cap \mathring{\Omega}_j = \emptyset$.
- El diámetro $h_i = Diam(\Omega_e)$ de cada Ω_e satisface $Diam(\Omega_e) \leq h$ para cada $e = 1, 2, ..., E$.
- Los vértices de cada Ω_e son llamados nodos, teniendo N de ellos por cada elemento Ω_e.

Definición 46 *Una familia de triangulaciones $\mathcal{T}_h$ es llamada de forma-regular si existe una constante independiente de h, tal que*

$$h_K \leq C\rho_K, \ \ con\ K \in \mathcal{T}_h,$$

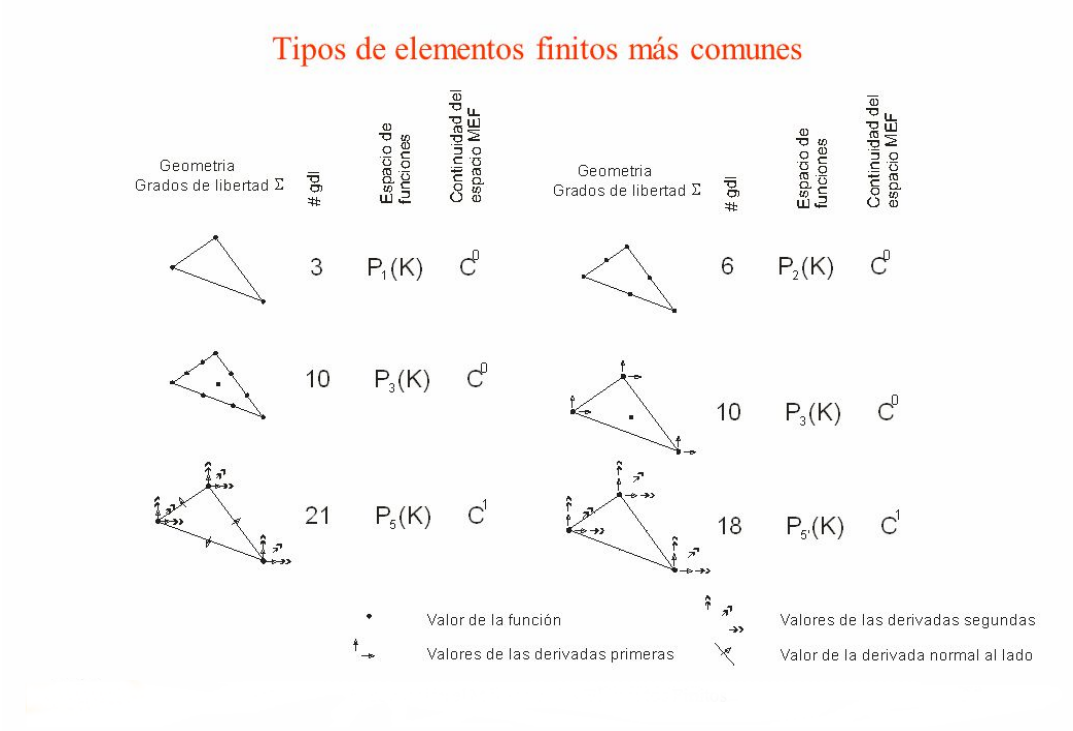
donde ρ_K es el radio del círculo más grande contenido en K. El radio h_K/ρ_K es llamado el aspect ratio de K.

Definición 47 *Una familia de triangulaciones $\mathcal{T}_h$ es llamada cuasi-uniforme si esta es de forma-regular y si existe una constante independiente de h, tal que*

$$h_K \leq Ch, \ \ con\ K \in \mathcal{T}_h.$$

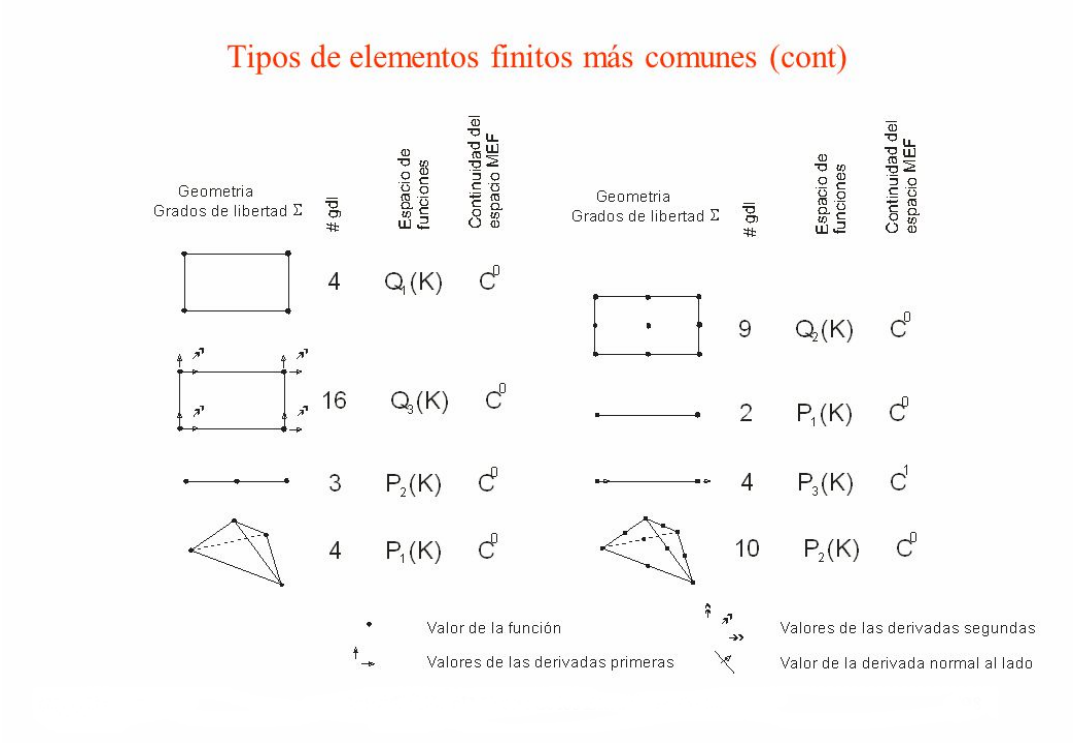
9.2 Generar Mallas con GMSH

GMSH es un generador para mallar tanto sus superficies como sus volúmenes para Elementos Finitos de código abierto con un motor integrado tipo CAD y un post-procesador. Su diseño tiene como fin proveer una herramienta rápida, ligera y amigable con un editor paramétrico y una gran capacidad de visualización avanzada. GMSH está organizado en cuatro módulos: geometría, malla, motor de cálculo, y post-proceso. La introducción de datos e instrucciones a cualquiera de los módulos puede hacerse ya sea de manera interactiva usando la interfaz gráfica, mediante archivos de texto ASCII empleando el lenguaje propio de GMSH ó a través de la interfaz de programación de aplicaciones C, C++, Python, Julia, Fortran, etc.

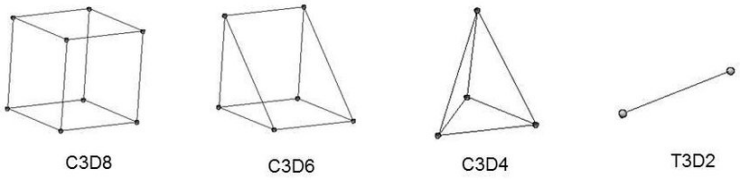


Mallado. Se define como mallado la discretización de una línea, superficie o volúmen en porciones de tamaño finito. Las porciones, además de tener un tamaño característico varias veces menor que el espacio discretizado,

serán entidades geométricas elementales como los triángulos o cuadriláteros en dos dimensiones o los tetraedros o prismas en tres dimensiones. Esta discretización en estructuras más elementales es esencial para la resolución de ecuaciones en derivadas parciales en dominios arbitrarios por el método de volúmenes finitos (FVM) o el método de elementos finitos (FEM).



El proceso de mallado más habitual es el de discretizar sucesivamente entidades de mayor dimensión como si de un problema de contorno se tratara. Por ejemplo, si queremos mallar un cubo primero definiremos los puntos de control en cada una de sus doce aristas. Luego discretizaremos en porciones elementales cada uno de sus seis lados y finalmente discretizaremos el volúmen. Para entender mejor este proceso, y con anterioridad a entender la sintaxis de los archivos GMSH presentamos este ejemplo de mallado de un cubo con tetraedros. El primer paso es definir los puntos básicos de la geometría.



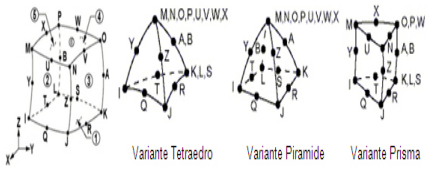


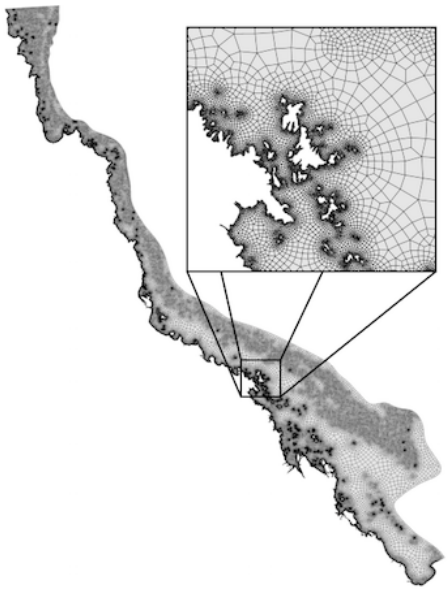
Fig. 3. Geometría del elemento finito sólido estructural

Mallas estructuradas Una malla estructurada es una malla que puede ser transformada, mediante una transformación biyectiva, a una cuadrícula. Esto significa que el problema podría ser resuelto en una malla uniforme y rectangular y devuelto a la geometría original sólo conociendo el jacobiano de la transformación porque este jacobiano se puede invertir.

Mallas no estructuradas Las mallas no estructuradas parten de distribuciones de puntos que cumplen una serie de propiedades dadas, la mayoría de ellas relacionadas con una distribución lo más uniforme posible dentro del contorno. Son adecuadas para geometrías irregulares donde es muy difícil generar una malla estructurada.

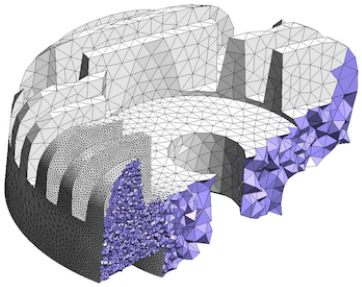
GMSH utiliza un algoritmo de creación de malla basado en la triangulación Delaunay. Dado un contorno se demuestra que sólo existe una triangulación óptima en la que, para cada triángulo formado por tres puntos de la malla, no exista ningún punto dentro de la circunferencia que pasa por los tres puntos.

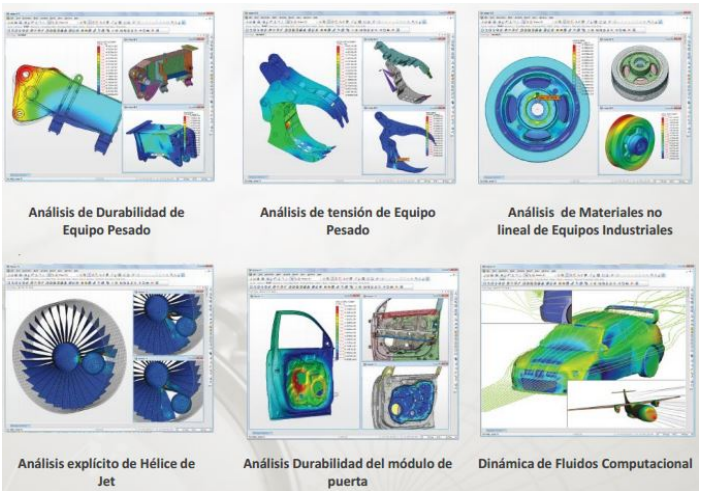
Si bien esta propiedad cierra el problema de la triangulación dados los puntos de la malla, en la mayoría de los casos sólo dispondremos del contorno. GMSH es capaz de encontrar una distribución no estructurada de puntos cuya triangulación resultante tiene suficiente calidad.



Discretización y calidad de malla. Una de las diferencias esenciales entre las mallas estructuradas y no estructuradas es el proceso de refinado. En algunos problemas la malla utilizada no produce suficiente precisión y es necesario rehacerla para aumentarla. En las mallas estructuradas el aumento de precisión no suele suponer un problema más allá de generar demasiados grados de libertad. Sin embargo en las mallas no estructuradas algunos algoritmos de refinado automático pueden funcionar mal.

Uno de los algoritmos más utilizados es el de partir cada uno de los triángulos o tetraedros en dos o más elementos. Este algoritmo recursivo puede refinar puntos arbitrarios en el espacio produciendo una pérdida de





precisión al introducir gradientes espurios debidos a una mala definición de la geometría.

GMSH se puede usar en conjunto con FreeFem++ que es un Software libre que sirve para la resolución numérica de problemas 2D y 3D utilizando el método de elementos finitos. ¿Qué problemas puedes resolver? De mecánica de fluidos, difusión de calor, elasticidad, electromagnetismo, etc. Todos aquellos que estén modelizados matemáticamente a través de una ecuación en derivadas parciales (EDP), por ejemplo:

https://www.um.es/freefem/ff++/pmwiki.php?n=Main.Elasticidad3D

Existe una gran cantidad herramientas para generar mallas en 2D, 3D que se pueden usar desde lenguajes de programación, entre las utilidades destacan:

- https://www.salome-platform.org/
- https://www.paraview.org/
- https://www.meshlab.net/
- https://wias-berlin.de/Software/index.jsp?id=TetGen&lang=1
- http://alice.loria.fr/Software/geogram/doc/html/index.html

Algo más de Ecuaciones Diferenciales Parciales En la red existen múltiples sitios especializados y una amplia bibliografía para conocer sobre Ecuaciones Diferenciales Parciales, nosotros hemos seleccionado diversos textos que ponemos a su disposición en:

EDP

Además existen una gran cantidad herramientas para para resolver ecuaciones diferenciales parciales que se pueden usar, entre ellas destacan:

- https://freefem.org/
- https://www.opengeosys.org/
- https://fenicsproject.org/
- https://www.openfoam.com/
- https://www.dealii.org/
- https://www.dune-project.org/
- https://www.aivc.org/resource/domus-20-whole-building-hygrothermal-simulation-program
- https://www.ctcms.nist.gov/fipy/
- http://onelab.info/
- https://www.csc.fi/web/elmer
- https://github.com/KratosMultiphysics/Kratos

- https://www.gomafem.com/
- http://www.juliafem.org/
- https://getfem.org/
- http://mofem.eng.gla.ac.uk/mofem/html/
- http://mech.fsv.cvut.cz/~sifel/
- https://dedalus-project.org/
- http://www.oofem.org/doku.php?id=en:oofem
- http://www.hpfem.org/hermes/
- https://cimec.org.ar/foswiki/Main/Cimec/PETScFEM
- https://pypi.org/project/HydPy/
- http://sfepy.org/doc-devel/index.html
- https://www.firedrakeproject.org/
- https://github.com/romeric/florence
- https://gfs.sourceforge.net/wiki/index.php/Main_Page
- https://www.comsol.com/
- https://www.featool.com/

Una vez que la triangulación $\mathcal{T}_h$ del dominio Ω es establecida, se procede a definir el espacio de elementos finitos $\mathbb{P}^h[k]$ a través del proceso descrito a continuación.

9.3 Interpolación para el Método de Elementos Finitos

Funciones Base A continuación describiremos la manera de construir las funciones base usadas por el método de elemento finito. En este procedimiento debemos tener en cuenta que las funciones base están definidas en un subespacio de $V = H^1(\Omega)$ para problemas de segundo orden que satisfacen las condiciones de frontera.

Las funciones base deberán satisfacer las siguientes propiedades:

i) Las funciones base ϕ_i son acotadas y continuas, i.e $\phi_i \in C(\Omega_e)$.

ii) Existen ℓ funciones base por cada nodo del polígono Ω_e, y cada función ϕ_i es no cero solo en los elementos contiguos conectados por el nodo i.

iii) $\phi_i = 1$ en cada i nodo del polígono Ω_e y cero en los otros nodos.

iv) La restricción ϕ_i a Ω_e es un polinomio, i.e. $\phi_i \in \mathbb{P}_k[\Omega_e]$ para alguna $k \geq 1$ donde $\mathbb{P}_k[\Omega_e]$ es el espacio de polinomios de grado a lo más k sobre Ω_e.

Decimos que $\phi_i \in \mathbb{P}_k[\Omega_e]$ es una base de funciones y por su construcción es evidente que estas pertenecen a $H^1(\Omega)$. Al conjunto formado por todas las funciones base definidas para todo Ω_e de Ω será el espacio $\mathbb{P}^h[k]$ de funciones base, i.e.

$$\mathbb{P}^h[k] = \bigcup_{e=1}^{E} \mathbb{P}_k[\Omega_e]$$

estas formarán las funciones base globales.

9.4 Discretización en 2D Usando Rectángulos

Para resolver la Ec. (9.1), usando una discretización con rectángulos, primero dividimos el dominio $\Omega \subset \mathbb{R}^2$ en N_x nodos horizontales por N_y nodos verticales, teniendo $E = (N_x - 1)(N_y - 1)$ subdominios o elementos rectangulares Ω_e tales que $\overline{\Omega} = \cup_{e=1}^{E} \overline{\Omega_e}$ y $\overline{\Omega_i} \cap \overline{\Omega_j} \neq \varnothing$ si son adyacentes, con un total de $N = N_x N_y$ nodos.

Donde las funciones lineales definidas por pedazos en Ω_e en nuestro caso serán polinomios de orden uno en cada variable separadamente y cuya restricción de ϕ_i a Ω_e es $\phi_i^{(e)}$. Para simplificar los cálculos en esta etapa, supondremos que la matriz $\underline{\underline{a}} = a \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, entonces se tiene que la integral del lado izquierdo de la Ec. (2.20) queda escrita como

$$\int_\Omega \left(a \nabla \phi_i \cdot \nabla \phi_j + c \phi_i \phi_j \right) dxdy = \int_\Omega f_\Omega \phi_j dxdy \tag{9.7}$$

donde

$$\begin{aligned} K_{ij} &= \int_\Omega \left(a \nabla \phi_i \cdot \nabla \phi_j + c \phi_i \phi_j \right) dxdy \\ &= \sum_{e=1}^{E} \int_{\Omega_e} \left(a \nabla \phi_i^{(e)} \cdot \nabla \phi_j^{(e)} + c \phi_i^{(e)} \phi_j^{(e)} \right) dxdy \\ &= \sum_{e=1}^{E} \int_{\Omega_e} \left(a \left[\frac{\partial \phi_i^{(e)}}{\partial x} \frac{\partial \phi_j^{(e)}}{\partial x} + \frac{\partial \phi_i^{(e)}}{\partial y} \frac{\partial \phi_j^{(e)}}{\partial y} \right] + c \phi_i^{(e)} \phi_j^{(e)} \right) dxdy \end{aligned} \tag{9.8}$$

y el lado derecho como

$$\begin{aligned} F_j &= \int_\Omega f_\Omega \phi_j dxdy \\ &= \sum_{e=1}^{E} \int_{\Omega_e} f_\Omega \phi_j^{(e)} dxdy. \end{aligned} \tag{9.9}$$

Para cada Ω_e de Ω, la submatriz de integrales (matriz de carga local)

$$K_{ij} = \int_{\Omega_e} \left(a \left[\frac{\partial \phi_i^{(e)}}{\partial x} \frac{\partial \phi_j^{(e)}}{\partial x} + \frac{\partial \phi_i^{(e)}}{\partial y} \frac{\partial \phi_j^{(e)}}{\partial y} \right] + c \phi_i^{(e)} \phi_j^{(e)} \right) dxdy \tag{9.10}$$

tiene la estructura

$$\begin{bmatrix} K_{1,1}^{(e)} & K_{1,2}^{(e)} & K_{1,3}^{(e)} & K_{1,4}^{(e)} \\ K_{2,1}^{(e)} & K_{2,2}^{(e)} & K_{2,3}^{(e)} & K_{2,4}^{(e)} \\ K_{3,1}^{(e)} & K_{3,2}^{(e)} & K_{3,3}^{(e)} & K_{3,4}^{(e)} \\ K_{4,1}^{(e)} & K_{4,2}^{(e)} & K_{4,3}^{(e)} & K_{4,4}^{(e)} \end{bmatrix}$$

la cual deberá ser ensamblada en la matriz de carga global que corresponda a la numeración de nodos locales del elemento Ω_e con respecto a la numeración global de los elementos en Ω.

De manera parecida, para cada Ω_e de Ω se genera el vector de integrales (vector de carga local)

$$F_j = \int_{\Omega_e} f_\Omega \phi_j^{(e)} dxdy \tag{9.11}$$

con la estructura

$$\begin{bmatrix} F_1^{(e)} \\ F_2^{(e)} \\ F_3^{(e)} \\ F_4^{(e)} \end{bmatrix}$$

el cual también deberá ser ensamblado en el vector de carga global que corresponda a la numeración de nodos locales al elemento Ω_e con respecto a la numeración global de los elementos de Ω.

Montando los $K_{ij}^{(e)}$ en la matriz $\underline{\underline{\mathbb{K}}}$ y los $F_j^{(e)}$ en el vector $\underline{\mathbb{F}}$ según la numeración de nodos global, se genera el sistema $\underline{\underline{\mathbb{K}}}\underline{u_h}=\underline{\mathbb{F}}$ donde $\underline{u_h}$ será el vector cuyos valores serán la solución aproximada a la Ec. (9.1) en los nodos interiores de Ω. La matriz $\underline{\underline{\mathbb{K}}}$ generada de esta forma, tiene una propiedad muy importante, es bandada y el ancho de banda es de 9 elementos, esto es muy útil al momento de soportar la matriz en memoria.

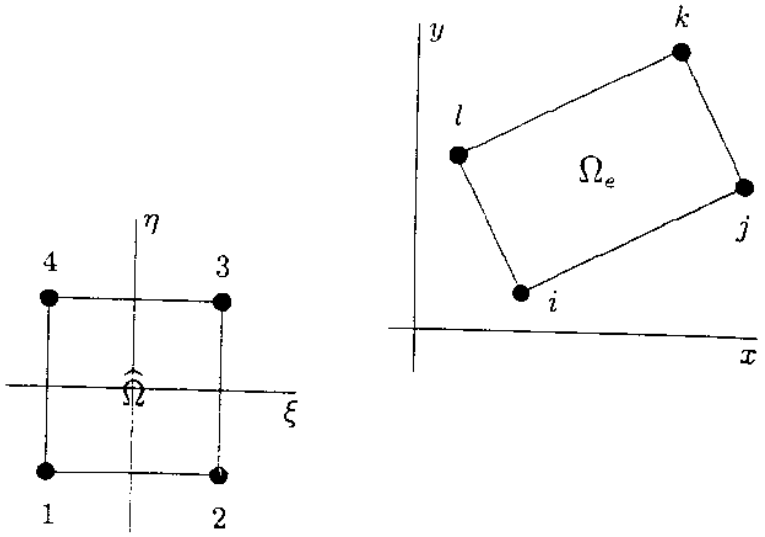
Para implementar numéricamente en cada Ω_e las integrales

$$\int_{\Omega_e} \left(a \left[\frac{\partial \phi_i^{(e)}}{\partial x} \frac{\partial \phi_j^{(e)}}{\partial x} + \frac{\partial \phi_i^{(e)}}{\partial y} \frac{\partial \phi_j^{(e)}}{\partial y} \right] + c\phi_i^{(e)} \phi_j^{(e)} \right) dxdy \tag{9.12}$$

y

$$\int_{\Omega_e} f_\Omega \phi_j^{(e)} dxdy, \tag{9.13}$$

teniendo en mente el simplificar los cálculos computacionales, se considera un elemento de referencia $\hat{\Omega}$ en los ejes coordenados (ε, η) cuyos vértices están el $(-1,-1), (1,-1), (1,1)$ y $(-1,1)$ respectivamente, en el cual mediante una función afín será proyectado cualquier elemento rectangular Ω_e cuyos vértices $(x_1^{(e)}, y_1^{(e)}), (x_2^{(e)}, y_2^{(e)}), (x_3^{(e)}, y_3^{(e)})$ y $(x_4^{(e)}, y_4^{(e)})$ están tomados en sentido contrario al movimiento de las manecillas del reloj como se muestra en la figura



mediante la transformación $f(x,y) = \underline{\underline{T}}(\varepsilon,\eta)+\underline{b}$, quedando dicha transformación como

$$\begin{aligned} x &= \frac{x_2^{(e)} - x_1^{(e)}}{2}\varepsilon + \frac{y_2^{(e)} - y_1^{(e)}}{2}\eta \\ y &= \frac{x_4^{(e)} - x_1^{(e)}}{2}\varepsilon + \frac{y_4^{(e)} - y_1^{(e)}}{2}\eta \end{aligned} \tag{9.14}$$

en la cual la matriz $\underline{\underline{T}}$ está dada por

$$\underline{\underline{T}} = \begin{pmatrix} \frac{x_2^{(e)} - x_1^{(e)}}{2} & \frac{y_2^{(e)} - y_1^{(e)}}{2} \\ \frac{x_4^{(e)} - x_1^{(e)}}{2} & \frac{y_4^{(e)} - y_1^{(e)}}{2} \end{pmatrix} \tag{9.15}$$

y el vector $\underline{b}= (b_1, b_2)$ es la posición del vector centroide del rectángulo Ω_e,

también se tiene que la transformación inversa es

$$\begin{aligned}\varepsilon &= \frac{x - b_1 - \frac{y_2^{(e)} - y_1^{(e)}}{2}\left[\frac{y - b_2}{\left(\frac{x_4^{(e)} - x_1^{(e)}}{2}\right)\left(\frac{x - b_1 - \frac{y_2^{(e)} - y_1^{(e)}}{2}}{\frac{x_2^{(e)} - x_1^{(e)}}{2}}\right)}\right]}{\frac{x_2^{(e)} - x_1^{(e)}}{2}} \\ \eta &= \frac{y - b_2}{\left(\frac{x_4^{(e)} - x_1^{(e)}}{2}\right)\left(\frac{x - b_1 - \frac{y_2^{(e)} - y_1^{(e)}}{2}}{\frac{x_2^{(e)} - x_1^{(e)}}{2}}\right) + \frac{y_4^{(e)} - y_1^{(e)}}{2}}.\end{aligned} \tag{9.16}$$

Entonces las $\phi_i^{(e)}$ quedan definidas en términos de $\hat{\phi}_i$ cómo

$$\begin{aligned}\hat{\phi}_1(\varepsilon,\eta) &= \frac{1}{4}(1-\varepsilon)(1-\eta) \\ \hat{\phi}_2(\varepsilon,\eta) &= \frac{1}{4}(1+\varepsilon)(1-\eta) \\ \hat{\phi}_3(\varepsilon,\eta) &= \frac{1}{4}(1+\varepsilon)(1+\eta) \\ \hat{\phi}_4(\varepsilon,\eta) &= \frac{1}{4}(1-\varepsilon)(1+\eta)\end{aligned} \tag{9.17}$$

y las funciones $\phi_i^{(e)}$ son obtenidas por el conjunto $\phi_i^{(e)}(x,y) = \hat{\phi}_i(\varepsilon,\eta)$ con (x,y) y (ε,η) relacionadas por la Ec. (9.14), entonces se tendrían las siguientes integrales

$$\begin{aligned}K_{ij}^{(e)} &= \int_{\Omega_e}\left(a\left[\frac{\partial\phi_i^{(e)}}{\partial x}\frac{\partial\phi_j^{(e)}}{\partial x} + \frac{\partial\phi_i^{(e)}}{\partial y}\frac{\partial\phi_j^{(e)}}{\partial y}\right] + c\phi_i^{(e)}\phi_j^{(e)}\right)dxdy \\ &= \int_{\hat{\Omega}}\left(\left[a\left(\frac{\partial\hat{\phi}_i}{\partial\varepsilon}\frac{\partial\varepsilon}{\partial x} + \frac{\partial\hat{\phi}_i}{\partial\eta}\frac{\partial\eta}{\partial x}\right)\left(\frac{\partial\hat{\phi}_j}{\partial\varepsilon}\frac{\partial\varepsilon}{\partial x} + \frac{\partial\hat{\phi}_j}{\partial\eta}\frac{\partial\eta}{\partial x}\right) + \right.\right. \\ &\quad \left.\left.\left(\frac{\partial\hat{\phi}_i}{\partial\varepsilon}\frac{\partial\varepsilon}{\partial y} + \frac{\partial\hat{\phi}_i}{\partial\eta}\frac{\partial\eta}{\partial y}\right)\left(\frac{\partial\hat{\phi}_j}{\partial\varepsilon}\frac{\partial\varepsilon}{\partial y} + \frac{\partial\hat{\phi}_j}{\partial\eta}\frac{\partial\eta}{\partial y}\right)\right] + c\hat{\phi}_i\hat{\phi}_j\right)|J|\,d\varepsilon d\eta\end{aligned} \tag{9.18}$$

donde el índice i y j varía de 1 a 4. En está última usamos la regla de la cadena y $dxdy = |J|\,d\varepsilon d\eta$ para el cambio de variable en las integrales,

aquí $|J| = \det T$, donde T está dado como en la Ec. (9.15). Para resolver $\int_{\Omega_e} f_\Omega \phi_j^{(e)} dxdy$ en cada Ω_e se genera las integrales

$$\begin{aligned} F_j^{(e)} &= \int_{\Omega_e} f_\Omega \phi_j^{(e)} dxdy \\ &= \int_{\hat{\Omega}} f_\Omega \hat{\phi}_j \, |J| \, d\varepsilon d\eta \end{aligned} \tag{9.19}$$

donde el índice i y j varía de 1 a 4.

Para realizar el cálculo numérico de las integrales en el rectángulo de refe-rencia $\hat{\Omega} = [-1, 1] \times [-1, 1]$, debemos conocer $\frac{\partial \phi i}{\partial \varepsilon}$, $\frac{\partial \phi i}{\partial \eta}$, $\frac{\partial \varepsilon}{\partial x}$, $\frac{\partial \varepsilon}{\partial y}$, $\frac{\partial \eta}{\partial x}$ y $\frac{\partial \eta}{\partial y}$, entonces realizando las operaciones necesarias a la Ec. (9.17) obtenemos

$$\begin{array}{ll} \frac{\partial \phi_1}{\partial \varepsilon} = -\frac{1}{4}(1-\eta) & \frac{\partial \phi_1}{\partial \eta} = -\frac{1}{4}(1-\varepsilon) \\ \frac{\partial \phi_2}{\partial \varepsilon} = \frac{1}{4}(1-\eta) & \frac{\partial \phi_2}{\partial \eta} = -\frac{1}{4}(1+\varepsilon) \\ \frac{\partial \phi_3}{\partial \varepsilon} = \frac{1}{4}(1+\eta) & \frac{\partial \phi_3}{\partial \eta} = \frac{1}{4}(1+\varepsilon) \\ \frac{\partial \phi_4}{\partial \varepsilon} = -\frac{1}{4}(1+\eta) & \frac{\partial \phi_4}{\partial \eta} = \frac{1}{4}(1-\varepsilon) \end{array} \tag{9.20}$$

y también

$$\begin{array}{ll} \frac{\partial \varepsilon}{\partial x} = \begin{pmatrix} \frac{y_4^{(e)} - y_1^{(e)}}{2 \det T} \end{pmatrix} & \frac{\partial \varepsilon}{\partial y} = \begin{pmatrix} \frac{x_4^{(e)} - x_1^{(e)}}{2 \det T} \end{pmatrix} \\ \frac{\partial \eta}{\partial x} = \begin{pmatrix} \frac{y_2^{(e)} - y_1^{(e)}}{2 \det T} \end{pmatrix} & \frac{\partial \eta}{\partial y} = \begin{pmatrix} \frac{x_2^{(e)} - x_1^{(e)}}{2 \det T} \end{pmatrix} \end{array} \tag{9.21}$$

las cuales deberán de ser sustituidas en cada $\underline{\underline{K_{ij}^{(e)}}}$ y $\underline{F_j^{(e)}}$ para calcular las integrales en el elemento Ω_e. Estas integrales se harán en el programa usando cuadratura Gaussiana, permitiendo reducir el número de cálculos al mínimo pero manteniendo el balance entre precisión y número bajo de operaciones necesarias para realizar las integraciones.

Suponiendo que Ω fue dividido en E elementos, estos elementos generan N nodos en total, de los cuales N_d son nodos desconocidos y N_c son nodos conocidos con valor γ_j, entonces el algoritmo de ensamble de la matriz $\underline{\underline{K}}$ y el vector $\underline{F}$ se puede esquematizar como:

$K_{i,j} = (\phi_i, \phi_j) \quad \forall i = 1, 2, ..., E, j = 1, 2, ..., E$

$F_j = (f_\Omega, \phi_j) \quad \forall j = 1, 2, ..., E$

$\forall j = 1, 2, ..., N_d$:

$$b_j = b_j - \gamma_i K_{i,j} \quad \forall i = 1, 2, ..., E$$

Así, se construye una matriz global en la cual están representados los nodos conocidos y los desconocidos, tomando sólo los nodos desconocidos de la matriz $\underline{\underline{K}}$ formaremos una matriz $\underline{\underline{A}}$, haciendo lo mismo al vector $\underline{F}$ formamos el vector $\underline{b}$, entonces la solución al problema será la resolución del sistema de ecuaciones lineales $\underline{\underline{A}}\underline{x}=\underline{b}$, este sistema puede resolverse usando por ejemplo el método de Gradiente Conjugado. El vector $\underline{x}$ contendrá la solución buscada en los nodos desconocidos N_d.

9.5 Discretización en 2D Usando Triángulos

Para resolver la Ec. (8.1), usando una discretización con triángulos, primero dividimos el dominio $\Omega \subset \mathbb{R}^2$ en N_x nodos horizontales por N_y nodos verticales, teniendo $E = 2(N_x - 1)(N_y - 1)$ subdominios o elementos triángulares Ω_e tales que $\overline{\Omega} = \cup_{e=1}^{E} \overline{\Omega_e}$ y $\overline{\Omega_i} \cap \overline{\Omega_j} \neq \varnothing$ si son adyacentes, con un total de $N = N_x N_y$ nodos.

Donde las funciones lineales definidas por pedazos en Ω_e en nuestro caso serán polinomios de orden uno en cada variable separadamente y cuya restricción de ϕ_i a Ω_e es $\phi_i^{(e)}$. Para simplificar los cálculos en esta etapa, supondremos que la matriz $\underline{\underline{a}} = a \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, entonces se tiene que la integral del lado izquierdo de la Ec. (2.20) queda escrita como

$$\int_\Omega \left(a \nabla \phi_i \cdot \nabla \phi_j + c \phi_i \phi_j \right) dxdy = \int_\Omega f_\Omega \phi_j dxdy \tag{9.22}$$

donde

$$\begin{aligned} K_{ij} &= \int_\Omega \left(a \nabla \phi_i \cdot \nabla \phi_j + c \phi_i \phi_j \right) dxdy \qquad (9.23) \\ &= \sum_{e=1}^{E} \int_{\Omega_e} \left(a \nabla \phi_i^{(e)} \cdot \nabla \phi_j^{(e)} + c \phi_i^{(e)} \phi_j^{(e)} \right) dxdy \\ &= \sum_{e=1}^{E} \int_{\Omega_e} \left(a \left[\frac{\partial \phi_i^{(e)}}{\partial x} \frac{\partial \phi_j^{(e)}}{\partial x} + \frac{\partial \phi_i^{(e)}}{\partial y} \frac{\partial \phi_j^{(e)}}{\partial y} \right] + c \phi_i^{(e)} \phi_j^{(e)} \right) dxdy \end{aligned}$$

y el lado derecho como

$$\begin{aligned} \underline{F_j} &= \int_\Omega f_\Omega \phi_j dxdy \\ &= \sum_{e=1}^{E} \int_{\Omega_e} f_\Omega \phi_j^{(e)} dxdy. \end{aligned} \tag{9.24}$$

Para cada Ω_e de Ω la submatriz de integrales (matriz de carga local)

$$\underline{\underline{K_{ij}}} = \int_{\Omega_e} \left(a \left[\frac{\partial \phi_i^{(e)}}{\partial x} \frac{\partial \phi_j^{(e)}}{\partial x} + \frac{\partial \phi_i^{(e)}}{\partial y} \frac{\partial \phi_j^{(e)}}{\partial y} \right] + c\phi_i^{(e)} \phi_j^{(e)} \right) dxdy \tag{9.25}$$

tiene la estructura

$$\begin{bmatrix} k_{1,1}^{(e)} & k_{1,2}^{(e)} & k_{1,3}^{(e)} \\ k_{2,1}^{(e)} & k_{2,2}^{(e)} & k_{2,3}^{(e)} \\ k_{3,1}^{(e)} & k_{3,2}^{(e)} & k_{3,3}^{(e)} \end{bmatrix}$$

la cual deberá ser ensamblada en la matriz de carga global que corresponda a la numeración de nodos locales del elemento Ω_e con respecto a la numeración global de los elementos en Ω.

De manera parecida, para cada Ω_e de Ω se genera el vector de integrales (vector de carla gocal)

$$F_j = \int_{\Omega_e} f_\Omega \phi_j^{(e)} dxdy \tag{9.26}$$

con la estructura

$$\begin{bmatrix} F_1^{(e)} \\ F_2^{(e)} \\ F_3^{(e)} \end{bmatrix}$$

el cual también deberá ser ensamblado en el vector de carga global que corresponda a la numeración de nodos locales al elemento Ω_e con respecto a la numeración global de los elementos de Ω.

Montando los $K_{ij}^{(e)}$ en la matriz $\underline{\underline{\mathbb{K}}}$ y los $\underline{F_j^{(e)}}$ en el vector $\underline{\mathbb{F}}$ según la numeración de nodos global, se genera el sistema $\underline{\underline{\mathbb{K}}}\underline{u_h}=\underline{\mathbb{F}}$ donde $\underline{u_h}$ será el vector cuyos valores serán la solución aproximada a la Ec. (8.1) en los nodos interiores de Ω. La matriz $\underline{\underline{\mathbb{K}}}$ generada de esta forma, tiene una propiedad muy importante, es bandada y el ancho de banda es de 7 elementos, esto es muy útil al momento de soportar la matriz en memoria.

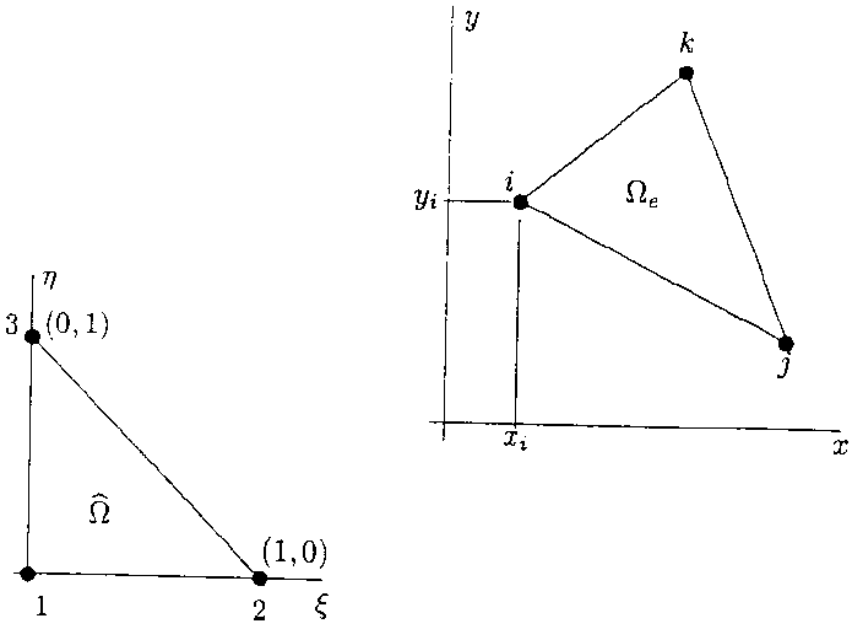
Para implementar numéricamente en cada Ω_e las integrales

$$\int_{\Omega_e} \left(a \left[\frac{\partial \phi_i^{(e)}}{\partial x} \frac{\partial \phi_j^{(e)}}{\partial x} + \frac{\partial \phi_i^{(e)}}{\partial y} \frac{\partial \phi_j^{(e)}}{\partial y} \right] + c \phi_i^{(e)} \phi_j^{(e)} \right) dxdy \tag{9.27}$$

y

$$\int_{\Omega_e} f_\Omega \phi_j^{(e)} dxdy$$

teniendo en mente el simplificar los cálculos computacionales se considera a un elemento de referencia $\hat{\Omega}$ en los ejes coordenados (ε, η) cuyos vertices estan en $(0,0), (1,0)$ y $(0,1)$ y en el cual mediante un mapeo afín será proyectado cualquier elemento triangular Ω_e cuyos vértices $(x_1^{(e)}, y_1^{(e)}), (x_2^{(e)}, y_2^{(e)}), (x_3^{(e)}, y_3^{(e)})$ están tomados en el sentido contrario al movimiento de las manecillas del reloj como se muestra en la figura



mediante la transformación $f(\varepsilon, \eta) = \underline{\underline{T}}(\varepsilon, \eta) + \underline{b}$, quedando dicha transformación como

$$\begin{aligned} x &= x_1^{(e)}(1 - \varepsilon - \eta) + x_2^{(e)}\varepsilon + x_3^{(e)}\eta \\ y &= y_1^{(e)}(1 - \varepsilon - \eta) + y_2^{(e)}\varepsilon + y_3^{(e)}\eta \end{aligned} \tag{9.28}$$

y en la cual la matriz $\underline{\underline{T}}$ está dada por

$$\underline{\underline{T}} = \begin{pmatrix} x_2^{(e)} - x_1^{(e)} & x_3^{(e)} - x_1^{(e)} \\ y_2^{(e)} - y_1^{(e)} & y_3^{(e)} - y_1^{(e)} \end{pmatrix} \tag{9.29}$$

donde $\underline{b}$ es un vector constante

$$\underline{b} = \begin{pmatrix} x_1^{(e)} \\ y_1^{(e)} \end{pmatrix} \tag{9.30}$$

también se tiene que la transformación inversa es

$$\begin{aligned} \varepsilon &= \frac{1}{2A_{\Omega_e}} \left[\left(y_3^{(e)} - y_1^{(e)} \right) \left(x - x_1^{(e)} \right) - \left(x_3^{(e)} - x_1^{(e)} \right) \left(y - y_1^{(e)} \right) \right] \quad (9.31) \\ \eta &= \frac{1}{2A_{\Omega_e}} \left[- \left(y_2^{(e)} - y_1^{(e)} \right) \left(x - x_1^{(e)} \right) - \left(x_2^{(e)} - x_1^{(e)} \right) \left(y - y_1^{(e)} \right) \right] \end{aligned}$$

donde

$$A_{\Omega_e} = \left| \det \begin{bmatrix} 1 & x_1^{(e)} & y_1^{(e)} \\ 1 & x_2^{(e)} & y_2^{(e)} \\ 1 & x_3^{(e)} & y_3^{(e)} \end{bmatrix} \right| . \tag{9.32}$$

Entoces las $\phi_i^{(e)}$ quedan definidas en términos de $\hat{\phi}_i$ cómo

$$\begin{aligned} \hat{\phi}_1(\varepsilon, \eta) &= 1 - \varepsilon - \eta \quad (9.33) \\ \hat{\phi}_2(\varepsilon, \eta) &= \varepsilon \\ \hat{\phi}_3(\varepsilon, \eta) &= \eta \end{aligned}$$

entonces las funciones $\phi_i^{(e)}$ son obtenidas por el conjunto $\phi_i^{(e)}(x, y) = \hat{\phi}_i(\varepsilon, \eta)$ con (x, y) y (ε,η) relacionadas por la Ec. (9.28), entonces se tendrían las siguientes integrales

$$\begin{aligned} k_{ij}^{(e)} &= \int_{\Omega_e} \left(a \left[\frac{\partial \phi_i^{(e)}}{\partial x} \frac{\partial \phi_j^{(e)}}{\partial x} + \frac{\partial \phi_i^{(e)}}{\partial y} \frac{\partial \phi_j^{(e)}}{\partial y} \right] + c \phi_i^{(e)} \phi_j^{(e)} \right) dx dy \quad (9.34) \\ &= \int_{\hat{\Omega}} \left(\left[a \left(\frac{\partial \hat{\phi}_i}{\partial \varepsilon} \frac{\partial \varepsilon}{\partial x} + \frac{\partial \hat{\phi}_i}{\partial \eta} \frac{\partial \eta}{\partial x} \right) \left(\frac{\partial \hat{\phi}_j}{\partial \varepsilon} \frac{\partial \varepsilon}{\partial x} + \frac{\partial \hat{\phi}_j}{\partial \eta} \frac{\partial \eta}{\partial x} \right) + \right. \right. \\ &\quad \left. \left. \left(\frac{\partial \hat{\phi}_i}{\partial \varepsilon} \frac{\partial \varepsilon}{\partial y} + \frac{\partial \hat{\phi}_i}{\partial \eta} \frac{\partial \eta}{\partial y} \right) \left(\frac{\partial \hat{\phi}_j}{\partial \varepsilon} \frac{\partial \varepsilon}{\partial y} + \frac{\partial \hat{\phi}_j}{\partial \eta} \frac{\partial \eta}{\partial y} \right) \right] + c \hat{\phi}_i \hat{\phi}_j \right) |J| \, d\varepsilon d\eta \end{aligned}$$

donde el índice i y j varía de 1 a 3. En está última usamos la regla de la cadena y $dxdy = |J|\, d\varepsilon d\eta$ para el cambio de variable en las integrales, aquí $|J| = \det T$, donde T está dado como en la Ec. (9.29). Para resolver $\int_{\Omega_e} f_\Omega \phi_j^{(e)} dxdy$ en cada Ω_e se genera las integrales

$$\begin{aligned} F_j^{(e)} &= \int_{\Omega_e} f_\Omega \phi_j^{(e)} dxdy \\ &= \int_{\hat{\Omega}} f_\Omega \hat{\phi}_j \, |J| \, d\varepsilon d\eta \end{aligned} \tag{9.35}$$

donde el índice i y j varía 1 a 3.

Para realizar el cálculo numérico de las integrales en el triángulo de referencia $\hat{\Omega}$, debemos conocer $\frac{\partial \phi i}{\partial \varepsilon}$, $\frac{\partial \phi i}{\partial \eta}$, $\frac{\partial \varepsilon}{\partial x}$, $\frac{\partial \varepsilon}{\partial y}$, $\frac{\partial \eta}{\partial x}$ y $\frac{\partial \eta}{\partial y}$, entonces realizando las operaciones necesarias a las Ec. (9.33) obtenemos

$$\begin{array}{ll} \frac{\partial \phi_1}{\partial \varepsilon} = -1 & \frac{\partial \phi_1}{\partial \eta} = -1 \\ \frac{\partial \phi_2}{\partial \varepsilon} = 1 & \frac{\partial \phi_2}{\partial \eta} = 0 \\ \frac{\partial \phi_3}{\partial \varepsilon} = 0 & \frac{\partial \phi_3}{\partial \eta} = 1 \end{array} \tag{9.36}$$

y también

$$\begin{array}{ll} \frac{\partial \varepsilon}{\partial x} = \frac{\left(y_3^{(e)} - y_1^{(e)}\right)}{2A_{\Omega_e}} & \frac{\partial \varepsilon}{\partial y} = -\frac{\left(x_3^{(e)} - x_1^{(e)}\right)}{2A_{\Omega_e}} \\ \frac{\partial \eta}{\partial x} = -\frac{\left(y_2^{(e)} - y_1^{(e)}\right)}{2A_{\Omega_e}} & \frac{\partial \eta}{\partial y} = \frac{\left(x_2^{(e)} - x_1^{(e)}\right)}{2A_{\Omega_e}} \end{array} \tag{9.37}$$

las cuales deberán de ser sustituidas en cada $\underline{\underline{K_{ij}^{(e)}}}$ y $\underline{F_j^{(e)}}$ para calcular las integrales en el elemento Ω_e.

Suponiendo que Ω fue dividido en E elementos, estos elementos generan N nodos en total, de los cuales N_d son nodos desconocidos y N_c son nodos conocidos con valor γ_j, entonces el algoritmo de ensamble de la matriz $\underline{\underline{K}}$ y el vector $\underline{F}$ se puede esquematizar como:

$K_{i,j} = (\phi_i, \phi_j) \quad \forall i = 1, 2, ..., E, j = 1, 2, ..., E$

$F_j = (f_\Omega, \phi_j) \quad \forall j = 1, 2, ..., E$

$\forall j = 1, 2, ..., N_d$:

$$b_j = b_j - \gamma_i K_{i,j} \quad \forall i = 1, 2, ..., E$$

Así, se construye una matriz global en la cual están representados los nodos conocidos y los desconocidos, tomando sólo los nodos desconocidos

de la matriz $\underline{\underline{K}}$ formaremos una matriz $\underline{\underline{A}}$, haciendo lo mismo al vector $\underline{F}$ formamos el vector $\underline{b}$, entonces la solución al problema será la resolución del sistema de ecuaciones lineales $\underline{\underline{A}}\underline{x}= \underline{b}$, este sistema puede resolverse usando por ejemplo el método de gradiente conjugado. El vector $\underline{x}$ contendrá la solución buscada en los nodos desconocidos N_d.

10 Apéndice F: 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 [12]).

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[1] del procesador.

[1] 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

10.1 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[2].

[2]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[3] y multiusuario[4] 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[5].

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;
}
```

[3] 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.

[4] 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.

[5] 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.

10.2 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

$$\underline{\underline{A}} = \begin{bmatrix} a_1 & b_1 & & & c_1 & & & & \\ d_2 & a_2 & b_2 & & & c_2 & & & \\ & d_3 & a_3 & b_3 & & & c_3 & & \\ & & d_4 & a_4 & b_4 & & & c_4 & \\ e_5 & & & d_5 & a_5 & b_5 & & & c_5 \\ & e_6 & & & d_6 & a_6 & b_6 & & \\ & & e_7 & & & d_7 & a_7 & b_7 & \\ & & & e_8 & & & d_8 & a_8 & b_8 \\ & & & & e_9 & & & d_9 & a_9 \end{bmatrix} \tag{10.1}$$

la cual puede ser almacenada usando el algoritmo (véase [12]) 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

$$\underline{Ind} = [-5, -1, 0, +1, +5] \tag{10.2}$$

y los datos serán almacenados usando la estructura

$$\underline{\underline{Dat}} = \begin{bmatrix} 0 & 0 & a_1 & b_1 & c_1 \\ 0 & d_2 & a_2 & b_2 & c_2 \\ 0 & d_3 & a_3 & b_3 & c_3 \\ 0 & d_4 & a_4 & b_4 & c_4 \\ e_5 & d_5 & a_5 & b_5 & c_5 \\ e_6 & d_6 & a_6 & b_6 & 0 \\ e_7 & d_7 & a_7 & b_7 & 0 \\ e_8 & d_8 & a_8 & b_8 & 0 \\ e_9 & d_9 & a_9 & 0 & 0 \end{bmatrix} \tag{10.3}$$

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;
```

10.3 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

$$\underline{\underline{A}} = \begin{bmatrix} a_1 & & & & & b_1 & & & c_1 \\ & a_2 & & & b_2 & & & c_2 & \\ & & & & a_3 & & & b_3 & c_3 \\ a_4 & & & b_4 & & & & & \\ & & a_5 & & & b_5 & & & c_5 \\ a_6 & b_6 & c_6 & & & & & & \\ & & & & & & a_7 & b_7 & c_7 \\ & & & a_8 & & & b_8 & c_8 & \\ & & & & & & a_9 & b_9 & \end{bmatrix} \tag{10.4}$$

la cual puede ser almacenada usando el algoritmo (véase [12]) Jagged Diagonal Storage (JDC), optimizado para ser usado en C++. Para este ejemplo en particular, se hará uso de una matriz de índices

$$\underline{\underline{Ind}} = \begin{bmatrix} 1 & 6 & 9 \\ 2 & 5 & 8 \\ 5 & 8 & 9 \\ 1 & 4 & 0 \\ 3 & 6 & 9 \\ 1 & 2 & 3 \\ 7 & 8 & 9 \\ 4 & 7 & 8 \\ 7 & 8 & 0 \end{bmatrix} \tag{10.5}$$

y los datos serán almacenados usando la estructura

$$\underline{\underline{Dat}} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \\ a_4 & b_4 & 0 \\ a_5 & b_5 & c_5 \\ a_6 & b_6 & c_6 \\ a_7 & b_7 & c_7 \\ a_8 & b_8 & c_8 \\ a_9 & b_9 & 0 \end{bmatrix} \tag{10.6}$$

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;
```

10.4 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.

11 Apéndice G: Solución de Grandes Sistemas de Ecuaciones Lineales

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

$$\underline{\underline{A}}\underline{u} = \underline{f} \tag{11.1}$$

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 [9], [10], [11] y [13]):

- 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[6] 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 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).

[6] 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.

11.1 Métodos Directos

En los métodos directos (véase [9] y [12]), 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.

11.1.1 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

$$\sum_{j=1}^{N} a_{ij}^{(0)} x_j = a_{i,n+1}^{(0)}, \quad i = 1, 2, ..., N \tag{11.2}$$

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)

$$x_i + \sum_{j=i+1}^{N} a_{ij}^{(i)} x_j = a_{i,n+1}^{(i)}, \quad i = 1, 2, ..., N \tag{11.3}$$

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

$$\begin{aligned} x_N &= a_{N,N+1}^{(N)}; \\ i &= N-1, N-2, ..., 1 \\ x_i &= a_{i,N+1}^{(i)} - \sum_{j=i+1}^{N} a_{ij}^{(i)} x_j. \end{aligned} \tag{11.4}$$

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 $\underline{u}$.

11.1.2 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:

$$\begin{array}{l} A_{ii} = 1, \text{ para } i = 1, ..., N \\ \text{Para } J = 1, 2, ..., N \ \{ \\ \quad \text{Para } i = 1, 2, ..., j \\ \quad\quad A_{ij} = A_{ij} - \sum_{k=1}^{i-1} A_{ik} A_{kj} \\ \quad \text{Para } i = j+1, j+2, ..., N \\ \quad\quad A_{ij} = \frac{1}{A_{jj}} \left(A_{ij} - \sum_{k=1}^{j-1} A_{ik} A_{kj} \right) \\ \} \end{array} \tag{11.5}$$

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

$$\underline{\underline{L}}\,\underline{y} = \underline{f} \quad \text{y} \quad \underline{\underline{U}}\,\underline{u} = \underline{y}. \tag{11.6}$$

i,.e.

$$\underline{\underline{L}}\,\underline{y} = \underline{f} \Leftrightarrow \left\{ \begin{array}{r} y_1 = f_1/l_{11} \\ y_i = \frac{1}{A_{ii}} \left(f_i - \sum_{j=1}^{i-1} A_{ii} y_i \right) \text{ para toda } i = 1, ..., n \end{array} \right. \tag{11.7}$$

y

$$\underline{\underline{U}}\,\underline{u} = \underline{y} \Leftrightarrow \left\{ \begin{array}{r} x_n = y_n/u_{nn} \\ x_i = \frac{1}{A_{ii}} \left(y_i - \sum_{j=i+1}^{n} A_{ij} x_j \right) \text{ para toda } i = n-1, ..., 1 \end{array} \right. \tag{11.8}$$

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.

11.1.3 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:

$$\begin{aligned} &\text{para } i = 1, 2, ..., n \text{ y } j = i+1, ..., n \\ &A_{ii} = \sqrt{\left(A_{ii} - \sum_{k=1}^{i-1} A_{ik}^2 \right)} \\ &A_{ji} = \left(A_{ji} - \sum_{k=1}^{i-1} A_{jk} A_{ik} \right) / A_{ii} \end{aligned} \tag{11.9}$$

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

$$\underline{\underline{L}}\underline{y} = \underline{f} \quad \text{y} \quad \underline{\underline{L}}^T\underline{u} = \underline{y} \tag{11.10}$$

usando la formulación equivalente dada por las Ec.(11.7) y (11.8) para la des-composición 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.

11.2 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 [9] y [12]).

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

$$\underline{\underline{A}}\underline{u} = \underline{f} \tag{11.11}$$

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

$$\underline{u}^k = \underline{\underline{T}}\underline{u}^{k-1} + \underline{c} \quad \forall k = 1, 2, 3, ... \tag{11.12}$$

La convergencia a la solución la garantiza el siguiente teorema (véase [13]).

Teorema 48 *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

$$cond(\underline{\underline{A}}) = \frac{\lambda_{\max}}{\lambda_{\min}} \geq 1 \tag{11.13}$$

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 [6], [7], [9] y [13]).

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:

11.2.1 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 (11.11) 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

$$\underline{u} = \underline{\underline{B}}\underline{u} + \underline{d} \tag{11.14}$$

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

$$x_i = \sum_{j=1}^{n} b_{ij} x_j^{(k-1)} + d_i \tag{11.15}$$

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. (11.11) se puede reescribir como

$$\left(\underline{\underline{I}} - \underline{\underline{B}}\right) \underline{u} = \underline{b}. \tag{11.16}$$

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

$$\underline{u}^{m+1} = \underline{\underline{B}}\underline{u}^m + \underline{k}. \tag{11.17}$$

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. (11.11) por $D^{-1} = \left[diag(\underline{\underline{A}})\right]^{-1}$ obteniendo

$$\underline{\underline{B}} = \underline{\underline{I}} - \underline{\underline{D}}^{-1}\underline{\underline{A}}, \quad \underline{k} = \underline{\underline{D}}^{-1}\underline{b}. \tag{11.18}$$

11.2.2 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. (11.17) puede ser escrita como

$$u_i^{m+1} = \sum_{j<i} b_{ij} u_j^{m+1} + \sum_{j>i} b_{ij} u_j^m + k_i. \tag{11.19}$$

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

$$\underline{u}^{m+1} = \underline{\underline{E}}\,\underline{u}^{m+1} + \underline{\underline{F}}\,\underline{u}^m + \underline{k} \tag{11.20}$$

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.

11.2.3 Richardson

Escribiendo el método de Jacobi como

$$\underline{u}^{m+1} - \underline{u}^m = \underline{b} - \underline{\underline{A}}\,\underline{u}^m \tag{11.21}$$

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

$$\underline{u}^{m+1} = \underline{u}^m + \omega \left(\underline{b} - \underline{\underline{A}}\,\underline{u}^m\right). \tag{11.22}$$

El método de Richardson se define como

$$\underline{u}^{m+1} = \left(\underline{\underline{I}} - \omega \underline{\underline{A}}\right) \underline{u}^m + \omega \underline{b} \tag{11.23}$$

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.

11.2.4 Relajación Sucesiva

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

$$u_i^{m+1} = (1-\omega)\, u_i^m + \omega \left[\sum_{j=1}^{i-1} b_{ij} u_j^{m+1} + \sum_{j=i+1}^{N} b_{ij} u_j^m + k_i \right] \tag{11.24}$$

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 [44] y [12]). 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

$$\mathcal{K}_k = Generado \left\{ \underline{r_0}, \underline{\underline{A}}\underline{r_0}, ..., \underline{\underline{A}}^{k-1}\underline{r_0} \right\} \text{ para } k \geq 1. \tag{11.25}$$

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

$$\underline{r_k} = \underline{b} - \underline{\underline{A}}\underline{x_k}. \tag{11.26}$$

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 [35].

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 49 *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*

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

11.2.5 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

$$\underline{\underline{A}} \underline{u} = \underline{f} \tag{11.27}$$

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:

$$\begin{array}{l} \alpha^n = \frac{\langle \underline{p}^n, \underline{p}^n \rangle}{\langle \underline{p}^n, \underline{\underline{A}}\underline{p}^n \rangle} \\ \underline{u}^{n+1} = \underline{u}^n + \alpha^n \underline{p}^n \\ \underline{r}^{n+1} = \underline{r}^n - \alpha^n \underline{\underline{A}}\underline{p}^n \\ \text{Prueba de convergencia} \\ \beta^n = \frac{\langle \underline{r}^{n+1}, \underline{r}^{n+1} \rangle}{\langle \underline{r}^n, \underline{r}^n \rangle} \\ \underline{p}^{n+1} = \underline{r}^{n+1} + \beta^n \underline{p}^n \\ n = n+1 \end{array} \tag{11.28}$$

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

$$\left\| \underline{u} - \underline{u}^k \right\|_{\underline{\underline{A}}} \leq \left\| \underline{u} - \underline{u}^0 \right\|_{\underline{\underline{A}}} \left[\frac{1 - \sqrt{Cond(\underline{\underline{A}})}}{1 + \sqrt{Cond(\underline{\underline{A}})}} \right]^{2k}. \tag{11.29}$$

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 50 *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*

$$\frac{\|e^n\|}{\|e^0\|} \leq \frac{2}{\left[\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^n + \left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^{-n}\right]} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^n \tag{11.30}$$

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

$$\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \simeq 1 - \frac{2}{\sqrt{\kappa}} \tag{11.31}$$

tal que

$$\|\underline{e}^n\|_{\underline{\underline{A}}} \simeq \left\|\underline{e}^0\right\|_{\underline{\underline{A}}} \exp\left(-2\frac{n}{\sqrt{\kappa}}\right) \tag{11.32}$$

de lo anterior se puede esperar un espectro de convergencia del orden de $O(\sqrt{\kappa})$ iteraciones (véase [13] y [44]).

11.2.6 Gradiente Conjugado Precondicionado

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

$$\lambda_1 \leq \frac{\underline{u}\underline{\underline{A}} \cdot \underline{u}}{\underline{u} \cdot \underline{u}} \leq \lambda_n \tag{11.33}$$

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

$$\lambda_1 \leq \frac{\underline{u}\underline{\underline{C}}^{-1}\underline{\underline{A}} \cdot \underline{u}}{\underline{u} \cdot \underline{u}} \leq \lambda_n \tag{11.34}$$

entonces la Ec. (11.27) se puede escribir como

$$\underline{\underline{C}}^{-1}\underline{\underline{A}}\underline{u} = \underline{\underline{C}}^{-1}b \tag{11.35}$$

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

$$\begin{aligned} \left\langle \underline{u}, \underline{\underline{C}}^{-1}\underline{\underline{A}}\underline{v} \right\rangle &= \underline{u} \cdot \underline{\underline{C}} \left(\underline{\underline{C}}^{-1}\underline{\underline{A}}\underline{v} \right) \\ &= \underline{u} \cdot \underline{\underline{A}}\underline{v} \end{aligned} \tag{11.36}$$

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

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

$$\langle \underline{u}, \underline{v} \rangle = \underline{u} \cdot \underline{\underline{C}}^{-1}\underline{\underline{A}}\underline{v} \tag{11.37}$$

por ello las Ecs. (11.28[1]) y (11.28[3]), se convierten en

$$\alpha^{k+1} = \frac{\underline{r}^k \cdot \underline{r}^k}{\underline{p}^{k+1} \cdot \underline{\underline{C}}^{-1}\underline{p}^{k+1}} \tag{11.38}$$

y

$$\beta^{k+1} = \frac{\underline{p}^k \cdot \underline{\underline{C}}^{-1}\underline{r}^k}{\underline{p}^k \cdot \underline{\underline{A}}\underline{p}^k} \tag{11.39}$$

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:

$$\begin{aligned} \beta^{k+1} &= \frac{\underline{p}^k \cdot \underline{\underline{C}}^{-1}\underline{r}^k}{\underline{p}^k \cdot \underline{\underline{A}}\underline{p}^k} \\ \underline{p}^{k+1} &= \underline{r}^k - \beta^{k+1}\underline{p}^k \\ \alpha^{k+1} &= \frac{\underline{r}^k \cdot \underline{r}^k}{\underline{p}^{k+1} \cdot \underline{\underline{C}}^{-1}\underline{p}^{k+1}} \\ \underline{u}^{k+1} &= \underline{u}^k + \alpha^{k+1}\underline{p}^{k+1} \\ \underline{r}^{k+1} &= \underline{\underline{C}}^{-1}\underline{r}^k - \alpha^{k+1}\underline{\underline{A}}\underline{p}^{k+1}. \end{aligned} \tag{11.40}$$

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 [35] y [50].

Teorema 51 *Sean $\underline{\underline{A}}, \underline{\underline{B}}$ y $\underline{\underline{C}}$ tres matrices simétricas y positivas definidas entonces*

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

11.2.7 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

$$\left\{\underline{v}^1, \underline{v}^2, ..., \underline{v}^n\right\} \tag{11.41}$$

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

$$\begin{aligned} \underline{\underline{AV}}_n &= \underline{\underline{V}}_n\underline{\underline{H}}_n + h_{n+1,n}\left[0, ..., 0, \underline{v}^{n+1}\right] \\ \underline{\underline{H}}_n &= \underline{\underline{H}}_n^T\underline{\underline{AV}}_n. \end{aligned} \tag{11.42}$$

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

$$\underline{\underline{A}}\underline{u} = \underline{f} \tag{11.43}$$

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:

$$\begin{array}{l}
\text{Para } n = 1, 2, ..., \text{Mientras } \beta^n < \tau\beta^0 \ \{ \\
\quad \underline{w}_0^{n+1} = \underline{\underline{A}}\underline{v}^n \\
\quad \text{Para } l = 1 \text{ hasta } n \ \{ \\
\qquad h_{l,n} = \left\langle \underline{w}_l^{n+1}, \underline{v}^l \right\rangle \\
\qquad \underline{w}_{l+1}^{n+1} = \underline{w}_l^{n+1} - h_{l,n}\underline{v}^l \\
\quad \} \\
\quad h_{n+1,n} = \left\| \underline{w}_{n+1}^{n+1} \right\| \\
\quad \underline{v}^{n+1} = \underline{w}_{n+1}^{n+1}/h_{n+1,n} \\
\quad \text{Calcular } \underline{y}^n \text{ tal que } \beta^n = \left\| \beta^0 \underline{e}_1 - \underline{\underline{\hat{H}}}_n \underline{y}^n \right\| \text{ es mínima} \\
\}
\end{array} \tag{11.44}$$

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

$$\underline{r}^n = \underline{r}^0 - \underline{\underline{AV}}_n y^n = \underline{\underline{V}}_{n+1} \left(\beta^0 \underline{e}_1 - \underline{\underline{\hat{H}}}_n \underline{y}^n \right). \tag{11.45}$$

Teorema 52 *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*

$$\frac{\left\| r^k \right\|}{\left\| r^0 \right\|} \le \kappa(V) \min_{p_\kappa \in \Pi\kappa, p_k(0)=1} \max_{\lambda_j} |p_\kappa(\lambda_j)| \tag{11.46}$$

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

11.3 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[7] para implementar de forma eficiente a los vectores, matrices -bandadas, dispersas o ralas- y resolver el sistemas lineal.

[7] 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.

11.3.1 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[8].

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)
```

[8]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)
```

11.3.2 Usando C++

GMM++[9] 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;
```

[9] 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;
}
```

11.4 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[10] y no lineales.

[10] 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.

12 Apéndice H: 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[11] un problema en múltiples subtareas y cómo distribuir eficazmente[12] 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[13], para aumentar su potencia de trabajo y rendimiento. En 2019 la supercomputadora *SUMMIT* de IBM funcionaban a más de 148 peta-Flops[14] (cada uno de ellos equivale a la realización de más de 1000 billones

[11] 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.

[12] 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.

[13] 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

[14] "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[15] 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.

[15] 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.

12.1 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[16], se encuentra compuesto por una serie de dispositivos, clasificados según la función que estos desempeñen. Dicha clasificación se compone de:

- 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.

[16] 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 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.

Desde el punto de vista funcional es una máquina que posee, al menos, 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 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[17], 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

[17] 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.

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.

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 diferentes 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[18] 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

[18] 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.

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, 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. 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-bit). Sin embargo, como veremos la mayoría de los valores que dan los fabricantes son con precisión simple (32-bit). Además, 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[19], no es una medida absoluta de la potencia del CPU ni de una

[19] 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.

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 Se dio a conocer en noviembre del 2023 la publicación de la 62^a edición del ranking de las 500 computadoras de mayor rendimiento del mundo:

- El Frontier, ubicado en el Laboratorio Nacional Oak Ridge del Departamento de Energía de Estados Unidos, sigue posicionado en el primer lugar (manteniendo desde mediados del año pasado). El clúster tiene 8.7 millones de núcleos de procesador y proporciona un rendimiento de 1.194 exaflops, el doble que el clúster, que ocupa el segundo lugar (con un menor consumo de energía).

- El nuevo clúster Aurora, desplegado en el Laboratorio Nacional Argonne del Departamento de Energía de Estados Unidos. El clúster tiene casi 4.8 millones de núcleos de procesador y proporciona un rendimiento de 585 petaflops, que es 143 petaflops más que el clúster que anteriormente ocupaba el segundo lugar.

- El clúster Eagle, lanzado este año por Microsoft para la nube Azure. El clúster contiene 1.12 millones de núcleos de procesador (CPU Xeon Platinum 8480C 48C 2GHz) y demuestra un rendimiento máximo de 561 petaflops. El Software del clúster está basado en Ubuntu 22.04.

El umbral mínimo de rendimiento para ingresar al Top 500 durante 6 meses fue de 2.02 petaflops (hace seis meses, 1.87 petaflops). Hace cinco años, sólo 272 clusters mostraban un rendimiento de más de un petaflop, hace seis años 138 y hace siete años - 94). Para el Top 100, el umbral de entrada aumentó de 6.3 a 7.89 petaflops, y para el Top 10, de 61.44 a 94.64 petaflops.

El rendimiento total de todos los sistemas en la clasificación durante 6 meses aumentó de 5.2 a 7 exaflops (hace cuatro años era 1.650 exaflops y hace seis años, 749 petaflops). El sistema que cierra el ranking actual ocupaba en la última edición el puesto 454.

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-[20].

[20] El AMD EPYC 9845 de 160 núcleos y 320 hilos a una frecuencia de 2,00 GHz basada

- 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 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

en Zen 5c, este se acompaña de 640 MB de caché L3.

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[21].

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[22] de Google. Ellos han publicado los detalles de dos ataques (no son los únicos[23]) basados en estos 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

[21] 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
```

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

[23] Entre las distintas vulnerabilidades detectadas y sus variantes resaltan: Meltdown (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/

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[24] 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 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

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

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[25].

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

[25] 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.

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[26] y multiusuario[27] 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[28].

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.

[26] 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.

[27] 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.

[28] 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.

Una Victim Cache no actúia 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 24 TB y continuamente incrementan su capacidad[29]), lo que significa que retienen datos 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.

[29] 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 y 24 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.

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 30 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.

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[30] 2.0, 35 MB/s
- 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

[30]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.

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 programen 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[31], mientras que la tarjeta NVidia Titan RTX proporciona 130 Tensor TFLOP de rendimiento, 576 núcleos tensores y 24 GB de memoria GDDR6.

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

[31] 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.

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[32]
- Mayor rendimiento de Voz sobre IP.
- Mejores estándares Ethernet en la industria.
- Mayor capacidad de ancho de banda por cables.

[32] 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[33]:

- 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

[33]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[34]: 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.

[34]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).

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
```

12.2 ¿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 rea-

lizar 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"[35] (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.

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

[35] "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.

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.

12.3 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.

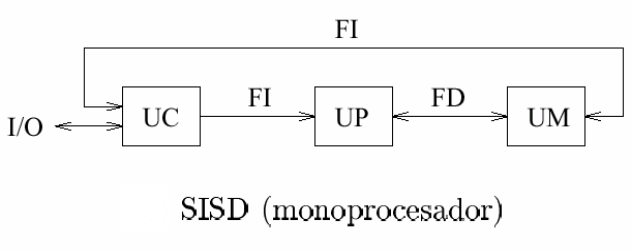


Figura 13: Ejemplo de máquina SISD

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 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.

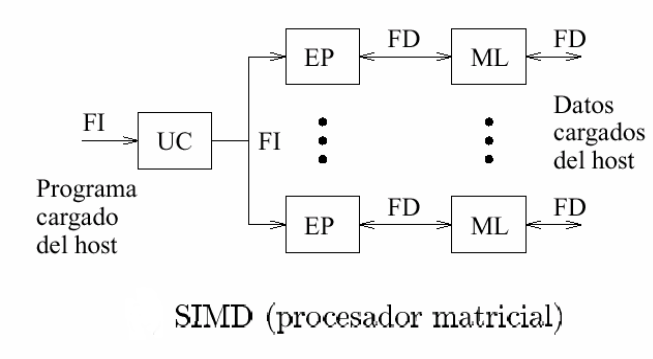


Figura 14: Ejemplo de máquina SIMD

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.

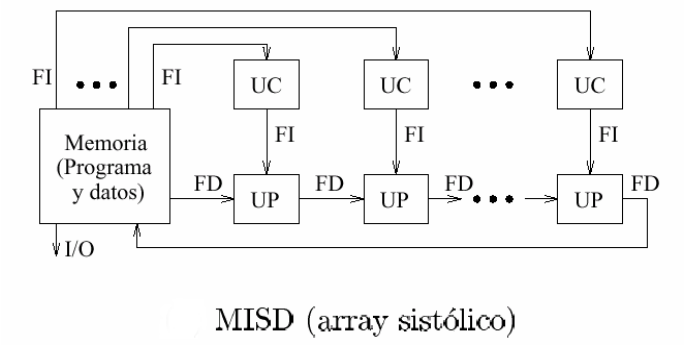


Figura 15: Ejemplo de máquina MISD

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 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-.

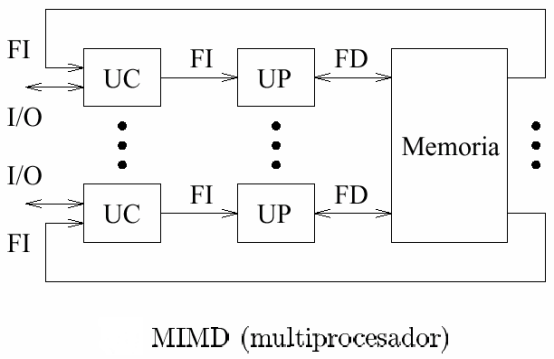


Figura 16: Ejemplo de máquina MIMD

12.4 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).

12.4.1 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.

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.

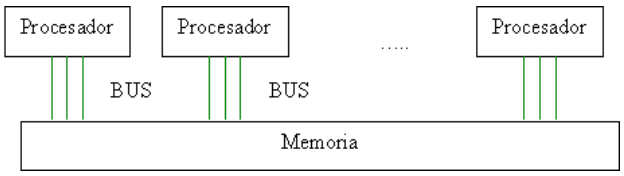


Figura 17: Arquitectura de una computadora paralela con memoria compartida

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.

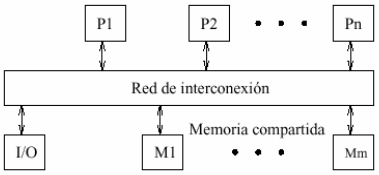


Figura 18: Acceso Uniforme a la memoria UMA

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[36] y red 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.

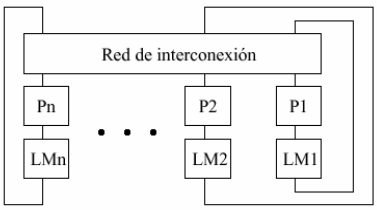


Figura 19: Acceso no uniforme a la memoria NUMA

El acceso a memoria, por tanto, no es uniforme para diferentes procesadores, existen memorias locales asociadas a cada procesador y estos pueden

[36] Red reconfigurable que permite la conexión de cada entrada con cualquiera de las salidas, es decir, permite cualquier permutación.

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.

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.

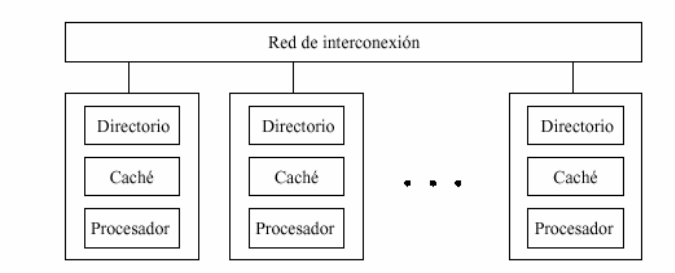


Figura 20: Ejemplo de COMA

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.

12.4.2 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.

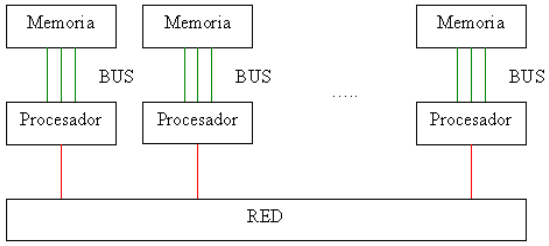


Figura 21: Arquitectura de una computadora paralela con memoria distribuida

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.

12.4.3 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:

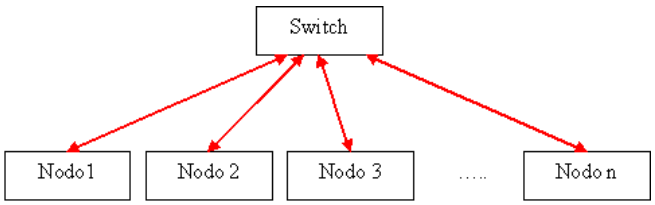


Figura 22: Arquitectura de un cluster

Los Cluster[37] se pueden clasificar en dos tipos según sus características físicas:

[37] 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.

12.4.4 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.

12.4.5 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[38]) frente a las CPU (decenas de cores[39]) de propósito general utilizando el paralelismo que ofrecen sus múltiples cores o núcleos, que permiten el lan-

[38]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).

[39]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[40] 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

[40] 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

12.5 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:

$$s = \frac{T(1)}{T(p)} \tag{12.1}$$

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:

$$e = \frac{T(1)}{pT(p)} = \frac{s}{p}. \tag{12.2}$$

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:

$$f = \frac{T_s}{T(1)} \tag{12.3}$$

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:

$$f = \frac{\frac{1}{s} - \frac{1}{p}}{1 - \frac{1}{p}}. \tag{12.4}$$

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[41], se muestra una ejemplificación de las métricas para un problema de ejemplo:

[41] 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[42] 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:

$$E_{128}^{32} = (32/128) * (5937/2818) = 0.52.$$

[42] 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.

12.6 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 -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.

12.6.1 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++[43], sin usar OpenMP, usamos:

```
$ g++ pi.cpp -o pi
```

Ejecutar midiendo el tiempo[44]:

```
$ time ./pi
```

[43] 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
```

[44] 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

12.6.2 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 [14], [15], [?] y [16]), 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 [14] y [15]):

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 5 *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.

12.6.3 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.

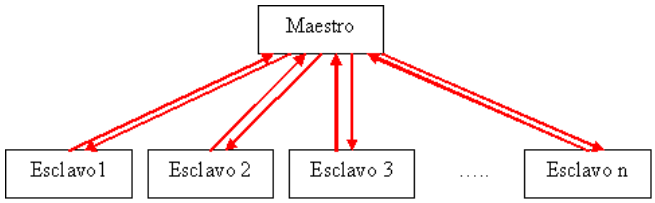


Figura 23: Esquema del Maestro-Esclavo

Donde, tomando en cuenta la implementación en estrella del Cluster, el modelo de paralelismo de MPI (véase 12.6.2) 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 12.6.2) 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 12.6.2, [14] y [15]).

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.

12.6.4 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.

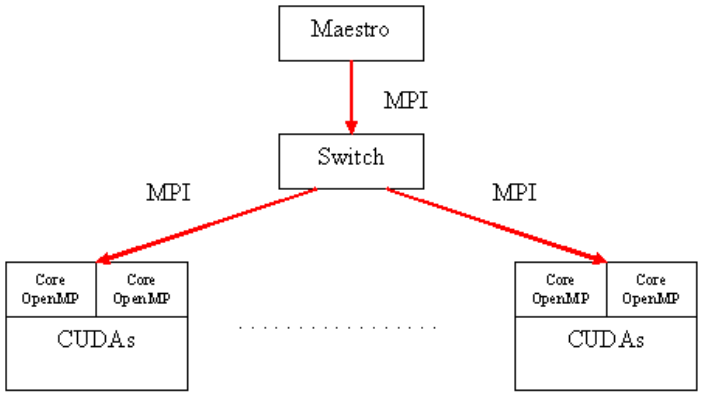


Figura 24: Paralelización Híbrida

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.

12.7 Programando Desde la Nube

Existen diferentes servicios Web[45] 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

[45] 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

13 Apéndice I: Algunos Conceptos Básicos

En este apéndice se darán algunas definiciones que se usan a lo largo del presente trabajo, así como se detallan algunos resultados generales de álgebra lineal y análisis funcional (en espacios reales) que se enuncian sin demostración pero se indica en cada caso la bibliografía correspondiente donde se encuentran estas y el desarrollo en detalle de cada resultado.

13.1 Nociones de Álgebra Lineal

A continuación detallaremos algunos resultados de álgebra lineal, las demostraciones de los siguientes resultados puede ser consultada en [11].

Definición 53 *Sea V un espacio vectorial y sea $f(\cdot) : V \to \mathbb{R}$, f es llamada funcional lineal si satisface la condición*

$$f(\alpha v + \beta w) = \alpha f(v) + \beta f(w) \quad \forall v, w \in V \quad y \quad \alpha, \beta \in \mathbb{R}. \tag{13.1}$$

Definición 54 *Si V es un espacio vectorial, entonces el conjunto V^* de todas las funcionales lineales definidas sobre V es un espacio vectorial llamado espacio dual de V.*

Teorema 55 *Si $\{v_1, ..., v_n\}$ es una base para el espacio vectorial V, entonces existe una única base $\{v_1^*, ..., v_n^*\}$ del espacio vectorial dual V^* llamado la base dual de $\{v_1, ..., v_n\}$ con la propiedad de que $V_i^* = \delta_{ij}$. Por lo tanto V es isomorfo a V^*.*

Definición 56 *Sea $D \subset V$ un subconjunto del espacio vectorial V. El nulo de D es el conjunto $N(D)$ de todas las funcionales en V^* tal que se nulifican en todo el subconjunto D, es decir*

$$N(D) = \{f \in V^* \mid f(v) = 0 \quad \forall v \in D\}. \tag{13.2}$$

Teorema 57 *Sea V un espacio vectorial y V^* el espacio dual de V, entonces*
a) $N(D)$ es un subespacio de V^*
b) Si $M \subset V$ es un subespacio de dimensión m, V tiene dimensión n, entonces $N(M)$ tiene dimensión $n - m$ en V^*.

Corolario 58 *Si $V = L \oplus M$ (suma directa) entonces $V^* = N(L) \oplus N(M)$.*

Teorema 59 *Sean V y W espacios lineales, si $T(\cdot) : V \to W$ es lineal, entonces el adjunto T^* de T es un operador lineal $T^* : W^* \to V^*$ definido por*

$$T^*(w^*)(u) = w^*(Tu). \tag{13.3}$$

Teorema 60 *Si H es un espacio completo con producto interior, entonces $H^* = H$.*

Definición 61 *Si V es un espacio vectorial con producto interior y $T(\cdot) : V \to V$ es una transformación lineal, entonces existe una transformación asociada a T llamada la transformación auto-adjunta T^* definida como*

$$\langle Tu, v\rangle = \langle u, T^*v\rangle . \tag{13.4}$$

Definición 62 *Sea V un espacio vectorial sobre los reales. Se dice que una función $\tau(\cdot,\cdot) : V \times V \to \mathbb{R}$ es una forma bilineal sobre V, si para toda $x, y, z \in V$ y $\alpha, \beta \in \mathbb{R}$ se tiene*

$$\begin{aligned} \tau(\alpha x + \beta y, z) &= \alpha\tau(x, z) + \beta\tau(y, z) \\ \tau(x, \alpha y + \beta z) &= \alpha\tau(x, y) + \beta\tau(x, z). \end{aligned} \tag{13.5}$$

Definición 63 *Si $\tau(\cdot,\cdot)$ es una forma bilineal sobre V, entonces la función $q_\tau(\cdot) : V \to \mathbb{R}$ definida por*

$$q_\tau(x) = \tau(x, x) \quad \forall x \in V \tag{13.6}$$

se le llama la forma cuadrática asociada a τ.

Notemos que para una forma cuadrática $q_\tau(\cdot)$ se tiene que $q_\tau(\alpha x) = |\alpha|^2 q_\tau(x)$ $\forall x \in V$ y $\alpha \in \mathbb{R}$.

Definición 64 *Sea $V \subset \mathbb{R}^n$ un subespacio, $P \in \mathbb{R}^n \times \mathbb{R}^n$*

13.2 σ-Álgebra y Espacios Medibles

A continuación detallaremos algunos resultados conjuntos de espacios σ-álgebra, conjuntos de medida cero y funciones medibles, las demostraciones de los siguientes resultados puede ser consultada en [52] y [49].

Definición 65 *Una σ-álgebra sobre un conjunto Ω es una familia ξ de subconjuntos de Ω que satisface*

- $\varnothing \in \xi$
- *Si* $\psi_n \in \xi$ *entonces* $\bigcup_{n=1}^{\infty} \psi_n \in \xi$
- *Si* $\psi \in \xi$ *entonces* $\psi^c \in \xi$.

Definición 66 *Si Ω es un espacio topológico, la familia de Borel es el conjunto σ-álgebra más pequeño que contiene a los abiertos del conjunto Ω.*

Definición 67 *Una medida μ sobre Ω es una función no negativa real valuada cuyo dominio es una σ-algebra ξ sobre Ω que satisface*

- $\mu(\varnothing) = 0$ *y*
- *Si* $\{\psi_n\}$ *es una sucesión de conjuntos ajenos de* ξ *entonces*

$$\mu\left(\bigcup_{n=1}^{\infty} \psi_n\right) = \sum_{n=1}^{\infty} \mu(\psi_n). \tag{13.7}$$

Teorema 68 *Existe una función de medida μ sobre el conjunto de Borel de $\mathbb{R}$ llamada la medida de Lebesgue que satisface $\mu([a,b]) = b - a$.*

Definición 69 *Una función $f : \Omega \to \mathbb{R}$ es llamada medible si $f^{-1}(U)$ es un conjunto medible para todo abierto U de Ω.*

Definición 70 *Sea $E \subset \Omega$ un conjunto, se dice que el conjunto E tiene medida cero si $\mu(E) = 0$.*

Teorema 71 *Si α es una medida sobre el espacio X y β es una medida sobre el espacio Y, podemos definir una medida μ sobre $X \times Y$ con la propiedad de que $\mu(A \times B) = \alpha(A)\beta(B)$ para todo conjunto medible $A \in X$ y $B \in Y$.*

Teorema 72 *(Fubini)*
Si $f(x,y)$ es medible en $X \times Y$ entonces

$$\int_{X\times Y} f(x,y)\, d\mu = \int_X \int_Y f(x,y)\, d\beta d\alpha = \int_Y \int_X f(x,y)\, d\alpha d\beta \tag{13.8}$$

en el sentido de que cualquiera de las integrales existe y son iguales.

Teorema 73 *Una función f es integrable en el sentido de Riemann en Ω si y sólo si el conjunto de puntos donde $f(\underline{x})$ es no continua tiene medida cero.*

Observación 6 *Sean f y g dos funciones definidas en Ω, decimos que f y g son iguales salvo en un conjunto de medida cero si $f(x) \neq g(x)$ sólo en un conjunto de medida cero.*

Definición 74 *Una propiedad P se dice que se satisface en casi todos lados, si existe un conjunto E con $\mu(E) = 0$ tal que la propiedad se satisface en todo punto de E^c.*

13.3 Espacios L^p

Las definiciones y material adicional puede ser consultada en [45], [48] y [49].

Definición 75 *Una función medible $f(\cdot)$ (en el sentido de Lebesgue) es llamada integrable sobre un conjunto medible $\Omega \subset \mathbb{R}^n$ si*

$$\int_{\Omega} |f| \, d\underline{x} < \infty. \tag{13.9}$$

Definición 76 *Sea p un número real con $p \geq 1$. Una función $u(\cdot)$ definida sobre $\Omega \subset \mathbb{R}^n$ se dice que pertenece al espacio $L^p(\Omega)$ si*

$$\int_{\Omega} |u(\underline{x})|^p \, d\underline{x} \tag{13.10}$$

es integrable.

Al espacio $L^2(\Omega)$ se le llama cuadrado integrable.

Definición 77 *La norma $L^2(\Omega)$ se define como*

$$\|u\|_{L^2(\Omega)} = \left(\int_{\Omega} |u(\underline{x})|^2 \, d\underline{x} \right)^{\frac{1}{2}} < \infty \tag{13.11}$$

y el producto interior en la norma $L^2(\Omega)$ como

$$\langle u, v \rangle_{L^2(\Omega)} = \int_{\Omega} u(\underline{x}) v(\underline{x}) d\underline{x}. \tag{13.12}$$

Definición 78 *Si $p \to \infty$, entonces definimos al espacio $L^\infty(\Omega)$ como el espacio de todas las funciones medibles sobre $\Omega \subset \mathbb{R}^n$ que sean acotadas en casi todo Ω (excepto posiblemente sobre un conjunto de medida cero), es decir,*

$$L^\infty(\Omega) = \{u \mid |u(x)| \leq k\} \tag{13.13}$$

definida en casi todo Ω, para algún $k \in \mathbb{R}$.

13.4 Distribuciones

La teoría de distribuciones es la base para definir a los espacios de Sobolev, ya que permiten definir las derivadas parciales de funciones no continuas, pero esta es coincidente con las derivadas parciales clásica si las funciones son continuas, para mayor referencia de estos resultados ver [45], [48] y [49]

Definición 79 *Sea $\Omega \subset \mathbb{R}^n$ un dominio, al conjunto de todas las funciones continuas definidas en Ω se denotará por $C^0(\Omega)$, o simplemente $C(\Omega)$.*

Definición 80 *Sea u una función definida sobre un dominio Ω la cual es no cero sólo en los puntos pertenecientes a un subconjunto propio $K \subset \Omega$. Sea $\overline{K}$ la clausura de K. Entonces $\overline{K}$ es llamado el soporte de u. Decimos que u tiene soporte compacto sobre Ω si su soporte $\overline{K}$ es compacto. Al conjunto de funciones continuas con soporte compacto se denota por $C_0(\Omega)$.*

Definición 81 *Sea $\mathbb{Z}_+^n$ el conjunto de todas las n-duplas de enteros no negativos, un miembro de $\mathbb{Z}_+^n$ se denota usualmente por α ó β (por ejemplo $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$. Denotaremos por $|\alpha|$ la suma $|\alpha| = \alpha_1 + \alpha_2 + ... + \alpha_n$ y por $D^\alpha u$ la derivada parcial*

$$D^\alpha u = \frac{\partial^{|\alpha|} u}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2} ... \partial x_n^{\alpha_n}} \tag{13.14}$$

así, si $|\alpha| = m$, entonces $D^\alpha u$ denota la m-ésima derivada parcial de u.

Definición 82 *Sea $C^m(\Omega)$ el conjunto de todas las funciones $D^\alpha u$ tales que sean funciones continuas con $|\alpha| = m$. Y $C^\infty(\Omega)$ como el espacio de funciones en el cual todas las derivadas existen y sean continuas en Ω.*

Definición 83 *El espacio $\mathcal{D}(\Omega)$ será el subconjunto de funciones infinitamente diferenciables con soporte compacto, algunas veces se denota también como $C_0^\infty(\Omega)$.*

Definición 84 *Una distribución sobre un dominio $\Omega \subset \mathbb{R}^n$ es toda funcional lineal continua sobre $\mathcal{D}(\Omega)$.*

Definición 85 *El espacio de distribuciones es el espacio de todas las funcionales lineales continuas definidas en $\mathcal{D}(\Omega)$, denotado como $\mathcal{D}^*(\Omega)$, es decir el espacio dual de $\mathcal{D}(\Omega)$.*

Definición 86 *Un función $f(\cdot)$ es llamada localmente integrable, si para todo subconjunto compacto $K \subset \Omega$ se tiene*

$$\int_K |f(x)|\, dx < \infty. \tag{13.15}$$

Ejemplo de una distribución es cualquier función $f(\cdot)$ localmente integrable en Ω. La distribución F asociada a f se puede definir de manera natural como $F : \mathcal{D}(\Omega) \to \mathbb{R}$ como

$$\langle F, \phi \rangle = \int_\Omega f\phi dx \tag{13.16}$$

con $\phi \in \mathcal{D}(\Omega)$.

Si el soporte de ϕ es $K \subset \Omega$, entonces

$$|\langle F, \phi \rangle| = \left| \int_\Omega f\phi dx \right| = \left| \int_K f\phi dx \right| \leq \sup_{x \in K} |\phi| \int_\Omega |f(x)|\, dx \tag{13.17}$$

la integral es finita y $\langle F, \phi \rangle$ tiene sentido. Bajo estas circunstancias F es llamada una distribución generada por f.

Otro ejemplo de distribuciones es el generado por todas las funciones continuas acotadas, ya que estas son localmente integrables y por lo tanto generan una distribución.

Definición 87 *Si una distribución es generada por funciones localmente integrables es llamada una distribución regular. Si una distribución no es generada por una función localmente integrable, es llamada distribución singular (ejemplo de esta es la delta de Dirac).*

Es posible definir de manera natural el producto de una función y una distribución. Específicamente, si $\Omega \subset \mathbb{R}^n$, u pertenece a $C^\infty(\Omega)$, y si $f(\cdot)$ es

una distribución sobre Ω, entonces entenderemos uf por la distribución que satisface

$$\langle (uf), \phi \rangle = \langle f, u\phi \rangle \tag{13.18}$$

para toda $\phi \in \mathcal{D}(\Omega)$. Notemos que la anterior ecuación es una generalización de la identidad

$$\int_\Omega [u(x) f(x)] \phi(x) dx = \int_\Omega f(x) [u(x) \phi(x)] dx \tag{13.19}$$

la cual se satisface si f es localmente integrable.

Derivadas de Distribuciones Funciones como la delta de Dirac y la Heaviside no tienen derivada en el sentido ordinario, sin embargo, si estas funciones son tratadas como distribuciones es posible extender el concepto de derivada de tal forma que abarque a dichas funciones, para ello recordemos que:

Teorema 88 *La versión clásica del teorema de Green es dada por la identidad*

$$\int_\Omega u \frac{\partial v}{\partial x_i} d\underline{x} = \int_{\partial\Omega} uvn_i d\underline{s} - \int_\Omega v \frac{\partial u}{\partial x_i} d\underline{x} \tag{13.20}$$

que se satisface para todas las funciones u, v en $C^1(\overline{\Omega})$, donde n_i es la i-ésima componente de la derivada normal del vector n en la frontera $\partial\Omega$ de un dominio Ω.

Una versión de la Ec. (13.20) en una dimensión se obtiene usando la fórmula de integración por partes, quedando como

$$\int_a^b uv'dx = [uv]\,|_a^b - \int_a^b vu'dx, \quad u, v \in C^1[a, b] \tag{13.21}$$

como un caso particular de la Ec. (13.20).

Este resultado es fácilmente generalizable a un resultado usando derivadas parciales de orden m de funciones $u, v \in C^m(\overline{\Omega})$ pero reemplazamos u por $D^\alpha u$ en la Ec. (13.20) y con $|\alpha| = m$, entonces se puede mostrar que:

Teorema 89 *Otra versión del teorema de Green es dado por*

$$\int_\Omega (D^\alpha u) v d\underline{x} = (-1)^{|\alpha|} \int_\Omega u D^\alpha v d\underline{x} + \int_{\partial\Omega} h(u, v) d\underline{s} \tag{13.22}$$

donde $h(u, v)$ es una expresión que contiene la suma de productos de derivadas de u y v de orden menor que m.

Ahora reemplazando v en la Ec. (13.22) por ϕ perteneciente a $\mathcal{D}(\Omega)$ y como $\phi = 0$ en la frontera $\partial\Omega$ tenemos

$$\int_{\Omega} (D^{\alpha}u)\,\phi d\underline{x} = (-1)^{|\alpha|} \int_{\Omega} uD^{\alpha}\phi d\underline{x} \tag{13.23}$$

ya que u es m-veces continuamente diferenciable, esta genera una distribución denotada por u, tal que

$$\langle u, \phi \rangle = \int_{\Omega} u\phi d\underline{x} \tag{13.24}$$

o, como $D^{\alpha}\phi$ también pertenece a $\mathcal{D}(\Omega)$, entonces

$$\langle u, D^{\alpha}\phi \rangle = \int_{\Omega} uD^{\alpha}\phi d\underline{x} \tag{13.25}$$

además, $D^{\alpha}u$ es continua, así que es posible generar una distribución regular denotada por $D^{\alpha}u$ satisfaciendo

$$\langle D^{\alpha}u, \phi \rangle = \int_{\Omega} (D^{\alpha}u)\,\phi d\underline{x} \tag{13.26}$$

entonces la Ec. (13.23) puede reescribirse como

$$\langle D^{\alpha}u, \phi \rangle = (-1)^{|\alpha|} \langle u, D^{\alpha}\phi \rangle\,, \quad \forall \phi \in D(\Omega). \tag{13.27}$$

Definición 90 *La derivada de cualquier distribución $f(\cdot)$ se define como: La $\alpha-$ésima derivada parcial distribucional o derivada generalizada de una distribución f es definida por una distribución denotada por $D^{\alpha}f$, que satisface*

$$\langle D^{\alpha}f, \phi \rangle = (-1)^{|\alpha|} \langle f, D^{\alpha}\phi \rangle\,, \quad \forall \phi \in \mathcal{D}(\Omega).$$

Nótese que si f pertenece a $C^{m}\left(\bar{\Omega}\right)$, entonces la derivada parcial distribucional coincide con la derivada parcial $\alpha-$ésima para $|\alpha| \leq m$.

Derivadas Débiles Supóngase que una función $u(\cdot)$ es localmente integrable que genere una distribución, también denotada por u, que satisface

$$\langle u, \phi \rangle = \int_{\Omega} u\phi dx \tag{13.28}$$

para toda $\phi \in \mathcal{D}(\Omega)$.

Además la distribución u posee derivada distribucional de todos los órdenes, en particular la derivada $D^{\alpha}u$ es definida por

$$\langle D^{\alpha}u, \phi\rangle = (-1)^{|\alpha|}\langle u, D^{\alpha}\phi\rangle, \quad \forall \phi \in D(\Omega). \tag{13.29}$$

por supuesto $D^{\alpha}u$ puede o no ser una distribución regular. Si es una distribución regular, entonces es generada por una función localmente integrable tal que

$$\langle D^{\alpha}u, \phi\rangle = \int_{\Omega} D^{\alpha}u(x)\,\phi(x)\,d\underline{x} \tag{13.30}$$

y se sigue que la función u y $D^{\alpha}u$ están relacionadas por

$$\int_{\Omega} D^{\alpha}u(x)\,\phi(x)\,d\underline{x} = (-1)^{|m|}\int_{\Omega} u(x)\,D^{\alpha}\phi(x)\,d\underline{x} \tag{13.31}$$

para $|\alpha| \leq m$.

Definición 91 *Llamamos a la función (o más precisamente, a la equivalencia de clases de funciones) $D^{\alpha}u$ obtenida en la Ec. (13.31), la $\alpha-$ésima derivada débil de la función u.*

Notemos que si u pertenece a $C^{m}\left(\bar{\Omega}\right)$, entonces la derivada $D^{\alpha}u$ coincide con la derivada clásica para $|\alpha| \leq m$.

14 Apéndice J: 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 $1e + 45$.

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, la implementación típica de punto flotante[46].

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

[46] 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}.

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 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[47].

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.

[47] ¿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 en *sys.float_info)* más grande es $2^{1023}(2 - 2^{-52}) = 2^{1024} - 2^{971} \approx 2^{1024} \approx$

1.8×10^{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 $p = 24$, 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

$$\pm (d_0 + d_1\beta^{-1} + ... + d_{p-1}\beta^{-(p-1)})\beta^e, (0 \leq d_i < \beta). \tag{14.1}$$

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

$$[\log_2 (e_{max} - e_{min} + 1)] + [\log_2 (\beta^p)] + 1$$

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. (14.1)), 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[48].

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 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.

[48] 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.

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,

$$\frac{1}{2}\beta^{-p} \leq \frac{1}{2}ulp \leq \frac{\beta}{2}\beta^{-p} \tag{14.2}$$

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. (14.2) 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. (14.2) 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

$$\text{los dígitos contaminados} \approx \log_\beta n. \tag{14.3}$$

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 92 *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. (14.3) 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 93 *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 (92)). 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 (93), 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

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ y } x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{14.4}$$

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

$$x_1 = \frac{-2c}{b+\sqrt{b^2-4ac}} \text{ y } x_2 = \frac{-2c}{b-\sqrt{b^2-4ac}} \tag{14.5}$$

Si $b^2 \gg ac$ y $b > 0$, entonces calcular x_1 usando la Ec. (14.4) implicará una cancelación. Por lo tanto, use la Ec. (14.5) para calcular x_1 y (14.4) para x_2. Por otro lado, si $b < 0$, use (14.4) para calcular x_1 y (14.5) 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 (93), 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.

14.1 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, 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. (14.4) o de forma alternativa con el algoritmo que se obtiene mediante la racionalización del numerador Ec. (14.5).

Otro algoritmo que podemos implementar es el método de Newton-Raphson[49] 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 """
  if A == 0.0:
    print("No es una ecuacion cuadratica")
    exit(1)
```

[49] 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 $f''(x)$.

```
    # 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.

14.2 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

$$0.1 + 0.1 + 0.1 \text{ ó } 0.1 * 3 \text{ ó } 0.1 + 0.2^{50}$$

obtendremos

$$0.30000000000000004$$

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

$$0.1, 0.10000000000000001 \text{ y}$$

[50] 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
```

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:

$$exp(141/895 + sqrt(780631)/895)$$

No Pruebes por la Igualdad Cuando se usa Punto Flotante no es recomendable escribir código como el siguiente[51]:

```
double x;
double y;
...
if (x == y) {...}
```

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.

[51] 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.

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))
```

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[52] 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";
```

[52]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 ....

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 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 operaciones 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 ++.

Relaciones de 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*.

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 $log(1+x)$.

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.

14.3 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.

15 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. Quarteroni y A. Valli; *Domain Decomposition Methods for Partial Differential Equations*. Clarendon Press Oxford, 1999. 7, 17, 21

[2] A. Quarteroni y A. Valli; Numerical Approximation of Partial Diferential Equations. Springer, 1994. 21

[3] A. Toselli, O. Widlund; *Domain Decomposition Methods - Algorithms and Theory*. Springer, 2005. 7, 8, 20, 33

[4] B. Dietrich, Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics, Cambridge University, 2001.

[5] B. F. Smith, P. E. BjØrstad, W. D. Gropp; *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996. 20, 33

[6] B. I. Wohlmuth; *Discretization Methods and Iterative Solvers Based on Domain Decomposition*. Springer, 2003. 20, 21, 33, 167

[7] L. F. Pavarino, A. Toselli; *Recent Developments in Domain Decomposition Methods*. Springer, 2003. 167

[8] K. Hutter & K. Jöhnk; *Continuum Methods of Physical Modeling*. Springer-Verlag Berlin Heidelber New York 2004. 5, 83

[9] M.B. Allen III, I. Herrera & G. F. Pinder; *Numerical Modeling in Science And Engineering*. John Wiley & Sons, Inc . 1988. 21, 75, 83, 125, 162, 163, 166, 167

[10] R. L. Burden y J. D. Faires; *Análisis Numérico*. Math Learning, 7 ed. 2004. 162

[11] S. Friedberg, A. Insel, and L. Spence; *Linear Algebra*, 4th Edition, Prentice Hall, Inc. 2003. 162, 279

[12] Y. Saad; *Iterative Methods for Sparse Linear Systems*. SIAM, 2 ed. 2000. 151, 157, 159, 163, 166, 170

[13] Y. Skiba; *Métodos y Esquemas Numéricos, un Análisis Computacional*. UNAM, 2005. 162, 167, 173

[14] W. Gropp, E. Lusk, A. Skjellem, *Using MPI, Portable Parallel Programming Whit the Message Passing Interface*. Scientific and Engineering Computation Series, 2ed, 1999. 7, 15, 268, 269, 275

[15] I. Foster; *Designing and Building Parallel Programs*. Addison-Wesley Inc., Argonne National Laboratory, and the NSF, 2004. 268, 269, 275

[16] Jorge L. Ortega-Arjona, Patterns for Parallel Software Design, Wiley series in Software Design Patterns, 2010. 268

[17] DDM Organization, Proceedings of International Conferences on Domain Decomposition Methods. www.ddm.org, 1988-2011.

[18] Toselli, A., and Widlund O. Domain decomposition methods- Algorithms and theory, Springer Series in Computational Mathematics, Springer-Verlag, Berlin, 2005, 450p.

[19] Farhat, C. and Roux, F. X. A method of finite element tearing and interconnecting and its parallel solution algorithm. Int. J. Numer. Meth. Engng., 32:1205-1227, 1991.

[20] Mandel J. & Tezaur R. Convergence of a substructuring method with Lagrange multipliers, Numer. Math. 73 (1996) 473-487.

[21] Farhat C., Lesoinne M. Le Tallec P., Pierson K. & Rixen D. FETI-DP a dual-primal unified FETI method, Part 1: A faster alternative to the two-level FETI method, Int. J. Numer. Methods Engrg. 50 (2001) 1523-1544.

[22] Farhat C., Lesoinne M., Pierson K. A scalable dual-primal domain decomposition method, Numer. Linear Algebra Appl. 7 (2000) 687-714.

[23] Mandel J. & Tezaur R. On the convergence of a dual-primal substructuring method, Numer. Math. 88(2001), pp. 5443-558.

[24] Mandel, J. Balancing domain decomposition. Comm. Numer. Meth. Engrg., 9:233-241, 1993.

[25] Mandel J., & Brezina M., Balancing domain decomposition for problems with large jumps in coefficients, Math. Comput. 65 (1996) 1387-1401.

[26] Dohrmann C., A preconditioner for substructuring based on constrained energy minimization. SIAM J. Sci. Comput. 25 (2003) 246-258.

[27] Mandel J. & Dohrmann C., Convergence of a balancing domain decomposition by constraints and energy minimization. Numer. Linear Algebra Appl. 10 (2003) 639-659.

[28] Da Conceição, D. T. Jr., Balancing domain decomposition preconditioners for non-symetric problems, Instituto Nacional de Matemática pura e Aplicada, Agencia Nacional do Petróleo PRH-32, Rio de Janeiro, May. 9, 2006.

[29] Herrera, I. "Theory of differential equations in discontinuous piecewise-defined-functions", NUMER METH PART D.E., 23(3): 597-639, 2007 OI 10.1002/num.20182.

[30] Herrera, I. New formulation of iterative substructuring methods without lagrange multipliers: Neumann-Neumann and FETI", NUMER METH PART D.E. 24(3) pp 845-878, May 2008 DOI 10.1002/num.20293.

[31] Herrera I. and R. Yates "Unified multipliers-free theory of dual primal domain decomposition methods". NUMER. METH. PART D. E. 25(3):552-581, May 2009, (Published on line May 13, 2008) DOI 10.1002/num.20359.

[32] Herrera, I. & Yates R. A. "The multipliers-free domain decomposition methods", NUMER. METH. PART D. E., 26(4): pp 874-905, July 2010. (Published on line: 23 April 2009, DOI 10.1002/num.20462)

[33] Herrera, I., Yates R. A. "The multipliers-free dual primal domain decomposition methods for nonsymmetric matrices". NUMER. METH. PART D. E. DOI 10.1002/num.20581 (Published on line April 28, 2010).

[34] I. Herrera, M. Díaz; *Modelación Matemática de Sistemas Terrestres* (Notas de Curso en Preparación). Instituto de Geofísica, (UNAM). 6, 79, 83

[35] I. Herrera;*Un Análisis del Método de Gradiente Conjugado*. Comunicaciones Técnicas del Instituto de Geofísica, UNAM; Serie Investigación, No. 7, 1988. 170, 176

[36] Herrera, I., Carrillo-Ledesma A. and Alberto Rosas-Medina, "A Brief Overview of Non-Overlapping Domain Decomposition Methods", Geofisica Internacional, Vol, 50, 4, October-December, 2011.

[37] Farhat Ch., Lesoinne M., Le Tallec P., Pierson K. and Rixen D. FETI-DP: A dual-primal unified FETI method-Part I: A faster alternative to the two level FETI method. Internal. J. Numer. Methods Engrg., 50:1523-1544, 2001.

[38] Rixen, D. and Farhat Ch. "A simple and efficient extension of a class of substructure based preconditioners to heterogeneous structural mechanics problems". Internal. J. Numer. Methods Engrg., 44:489-516, 1999.

[39] Alberto Rosas Medina, "Métodos de estabilización para problemas de Advección-Difusión", Trabajo de investigación para presentar el examen de candidaura de doctorado en Ciencias de la Tierra, 2011.

[40] Klawonn A. and Widlund O.B. FETI and Neumann-Neumann iterative substructuring methods: connections and new results. Comm. Pure and Appl. Math. 54(1): 57-90, 2001.

[41] Tezaur R., Analysis of Lagrange multipliers based domain decomposition. P.H. D. Thesis, University of Colorado, Denver, 1998.

[42] Herrera I. & Rubio E., "Unified theory of Differential Operators Acting on Discontinuos Functions and of Matrices Acting on Discontinuous Vectors", 19th International Conference on Domain Decomposition Methods, Zhangjiajie, China 2009. (Oral presentation). Internal report #5, GMMC-UNAM, 2011.

[43] Toselli, A., FETI domain Decomposition methods for Escalar Advection-Diffusion Problems. Compututational Methods Appl. Mech. Engrg. 190. (2001), 5759-5776.

[44] C.T. Keller, Iterative Methods for Linear and Nonlinear Equations, Societe for Industrial and Applied Mathematics, 1995. 170, 173

[45] J. L. Lions & E. Magenes; *Non-Homogeneous Bonduary Value Problems and Applications Vol. I*, Springer-Verlag Berlin Heidelber New York 1972. 18, 91, 105, 116, 124, 282, 283

[46] J. II. Bramble, J. E. Pasciak and A. II Schatz. *The Construction of Preconditioners for Elliptic Problems by Substructuring.* I. Math. Comput., 47, 103-134,1986. 33

[47] X. O. Olivella, C. A. de Sacribar; *Mecánica de Medios Continuos para Ingenieros.* Ediciones UPC, 2000. 83

[48] P.G. Ciarlet, J. L. Lions; *Handbook of Numerical Analysis, Vol. II.* North-Holland, 1991. 91, 282, 283

[49] B. D. Reddy; *Introductory Functional Analysis - With Applications to Boundary Value Problems and Finite Elements.* Springer 1991. 91, 105, 116, 124, 280, 282, 283

[50] G. Herrera; Análisis de Alternativas al Método de Gradiente Conjugado para Matrices no Simétricas. Tesis de Licenciatura, Facultad de Ciencias, UNAM, 1989. 176

[51] 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.

[52] W. Rudin; *Principles of Mathematical Analysis*. McGraw-Hill International Editions, 1976. 280

[53] F. Brezzi y M. Fortin; Mixed and Hibrid Finite Element Methods, Springer, 1991. 91

[54] Handbook of Floating-Point Arithmetic 2010th Edition, Jean-Michel Muller, Nicolas Brisebarre, Et alii, Birkhäuser, 2010.

[55] What Every Computer Scientist Should Know About Floating-Point Arithmetic, David Goldbergm, ACM Computing Surveys, Vol 23, No 1, March 1991

[56] 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.)

[57] 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)

[58] IEEE, https://www.ieee.org/

[59] Intel, https://www.intel.la

[60] AMD, https://www.amd.com

[61] ARM, https://www.arm.com/

[62] 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