4to Congreso Internacional, 2do Congreso Nacional de Métodos Numéricos en Ingeniería y Ciencias Aplicadas
M.C. Súarez, S. Gallegos, F. Zárate, S. Botello, M. Moreles, J. Pérez, M. Rodríguez y F. Domínguez (Editores)

APLICACIÓN DEL CÓMPUTO PARALELO A LA MODELACIÓN DE SISTEMAS CONTINUOS EN CIENCIAS E INGENIERÍA

A. Carrillo Ledesma e I. Herrera Revilla
Instituto de Geofísica
Departamento de Recursos Naturales
Universidad Nacional Autónoma de México
Ciudad Universitaria, México, Distrito Federal, C.P. 04510
Email: antonio@mmc.igeofcu.unam.mx
web page: http://www.mmc.igeofcu.unam.mx

Resumen. Los modelos de los sistemas continuos contienen un gran número de grados de libertad y al ser discretizados generan sistemas de ecuaciones lineales que llegan a ser de gran tamaño, lo que da lugar a que el tratamiento de sistemas grandes requieran los recursos computacionales más avanzados, en particular, la computación en paralelo es un recurso imprescindible que debe ser aprovechado en este contexto. El presente trabajo muestra una metodología eficiente, flexible y escalable en la cual se implementa de forma paralela el método numérico de descomposición de dominio de subestructuración precondicionado para poder resolver ecuaciones diferenciales parciales elípticas en equipos paralelos con memoria distribuida y mostrar las ventajas con respecto a otros métodos de solución numérica.

Palabras clave: Cómputo en Paralelo, Métodos de Descomposición de Dominio, Método de Subestructuración Precondicionado, Ecuaciones Diferenciales Parciales.

1 INTRODUCCIÓN

Los modelos de los sistemas continuos en ciencias e ingeniería contienen un gran número de grados de libertad y al ser discretizados por medio de algún método numérico generan sistemas de ecuaciones lineales algebraicos de gran tamaño, lo que da lugar a que el tratamiento de sistemas grandes requieran los recursos computacionales más avanzados, en particular, el cómputo paralelo –cómputo de alto desempeño- es un recurso indispensable que debe ser aprovechado en este contexto.

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 implementar la solución del problema, algunas de estas consideraciones son:

- Número de procesadores disponibles
- Tamaño y tipo de partición del dominio
- Tiempo de ejecución predeterminado

Siendo común que estos factores limiten y condicionen el desarrollo de la aplicación computacional, de forma tal que el encargado de la implementación computacional de la solución numérica tiene además de las

complicaciones técnicas propias del modelo matemático, numérico y computacional, el conciliarlos con dichas consideraciones limitantes en la implementación numérica del problema.

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

Por otro lado, la computación en paralelo es una técnica que permite distribuir una gran carga computacional entre muchos procesadores. Y es bien sabido [3] 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. Los métodos de descomposición de dominio [5], [6], [7] y [8] introducen desde la formulación matemática misma del problema una separación natural de las tareas y simplifican considerablemente la transmisión de información.

Así, en el presente trabajo mostraremos la base de una metodología en la cual se implemente la solución numérica del método de subestructuración precondicionado [1], [4] y [6] de forma paralela usando memoria distribuida bajo el paradigma de programación orientada a objetos, en el lenguaje de programación C++ y la librería de paso de mensajes MPI y veremos la forma de construir el modelo computacional, mostrando los alcances y limitaciones en el consumo de recursos computacionales, así como la evaluación de algunas variantes de los métodos numéricos con los que es posible implementar la solución numérica [2].

Una versión de estos programas está disponible en la página Web *http://www.mmc.igeofcu.unam.mx/*, bajo el rubro *FEM y DDM*.

2 MÉTODO DE SUBESTRUCTURACIÓN PRECONDICIONADO

Por simplicidad de la exposición, trabajaremos en el dominio $\Omega \subset \Re^2$, con la ecuación diferencial parcial de Poisson

$$\begin{aligned} -\nabla^2 u &= f_\Omega \text{ en } \Omega \\ u &= g_{\partial\Omega} \text{ en } \partial\Omega \end{aligned} \tag{1.1}$$

en particular, tomaremos $\Omega = [-1,1]\times[0,1]$, $f_\Omega = 2n^2\pi^2 sen(n\pi x) * sen(n\pi y)$ y $g_{\partial\Omega} = 0$, con $n \in \mathbb{N}$. Cuya solución es $u = sen(n\pi x) * sen(n\pi y)$, si por ejemplo $n = 4$, la solución se ve como en la figura 1.

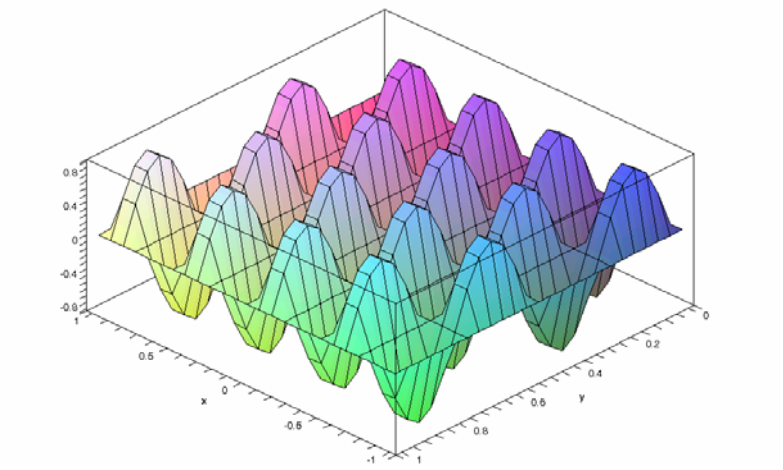

Figura 1 – Solución de la ecuación de Poisson con $n = 4$.

Si el dominio Ω es dividido en $E = n \times m$ subdominios Ω_δ, $\delta = 1, 2, ..., E$, sin traslape tal que

$$\Omega_i \cap \Omega_j = \varnothing, \quad \forall i \neq j \quad \text{y} \quad \overline{\Omega} = \bigcup_{i=1}^{E} \overline{\Omega_i} \tag{1.2}$$

y al conjunto

$$\Sigma = \bigcup_{i=1}^{E} \Sigma_i, \quad \text{si } \Sigma_i = \partial\Omega_i \setminus \partial\Omega \tag{1.3}$$

lo llamamos la frontera interior de los subdominios y además cada Ω_δ es descompuesto en $p \times q$ subdominios, obteniendo la descomposición fina del dominio. Un ejemplo de una descomposición $(3\times 3 \text{ y } 6\times 5)$ del dominio Ω, se muestra en la figura 2.

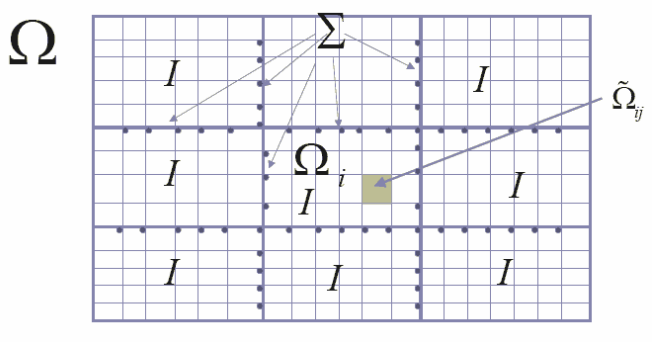

Figura 2 – Descomposición del dominio Ω *en una malla de* $(3\times 3 \text{ y } 6\times 5)$.

Eligiendo al conjunto de polinomios $P^h[k]$ de grado menor o igual a k, como el espacio de funciones lineales ϕ_i definidas por pedazos en cada Ω_δ, entonces definimos al espacio

$$V^h = Generado\left\{\phi_i \in P^h[k] \;\middle|\; \phi_i(x) = 0 \text{ en } \partial\Omega\right\} \tag{1.4}$$

así, la solución aproximada de la ecuación de Poisson usando el método Galerkin queda en términos de

$$\int_\Omega \nabla\phi_i \cdot \nabla\phi_j dxdy = \int_\Omega f_\Omega \phi_j dxdy. \tag{1.5}$$

Si definimos para cada subdominio Ω_δ, las bases ω_I^i y ω_Σ^α que corresponden a los nodos interiores y a los nodos de frontera de cada subdominio Ω_δ respectivamente y a partir de ellas definimos [4] para cada subdominio Ω_δ a las matrices

$$\begin{aligned} \underline{\underline{A_\delta^{II}}} &= \left[\left\langle \omega_I^i, \omega_I^j \right\rangle\right] \quad \underline{\underline{A_\delta^{\Sigma\Sigma}}} = \left[\left\langle \omega_\Sigma^\alpha, \omega_\Sigma^\alpha \right\rangle\right] \\ \underline{\underline{A_\delta^{I\Sigma}}} &= \left[\left\langle \omega_I^i, \omega_\Sigma^\alpha \right\rangle\right] \quad \underline{\underline{A_\delta^{\Sigma I}}} = \left[\left\langle \omega_\Sigma^\alpha, \omega_I^i \right\rangle\right] \end{aligned} \tag{1.6}$$

entonces, podríamos definir –no se construyen estas matrices globales- las matrices $\underline{\underline{A^{II}}}, \underline{\underline{A^{\Sigma I}}}, \underline{\underline{A^{I\Sigma}}}$ y $\underline{\underline{A^{\Sigma\Sigma}}}$ donde

$$\begin{aligned} \underline{\underline{A^{II}}} &= \begin{pmatrix} \underline{\underline{A_1^{II}}} & & 0 \\ & \ddots & \\ 0 & & \underline{\underline{A_E^{II}}} \end{pmatrix} \quad \underline{\underline{A^{I\Sigma}}} = \begin{bmatrix} \underline{\underline{A_1^{I\Sigma}}} \\ \vdots \\ \underline{\underline{A_E^{I\Sigma}}} \end{bmatrix} \\ \underline{\underline{A^{\Sigma I}}} &= \begin{bmatrix} \underline{\underline{A_1^{\Sigma I}}} & \cdots & \underline{\underline{A_E^{\Sigma I}}} \end{bmatrix} \quad \underline{\underline{A^{\Sigma}}} = \left[\sum_{i=1}^{E} \underline{\underline{A_i^{\Sigma\Sigma}}}\right] \end{aligned} \tag{1.7}$$

y definiendo $\underline{u} = \left(\underline{u_I}, \underline{u_\Sigma}\right)$ como $\underline{u_I} = \left(u_1, ..., u_{N_I}\right)$ y $\underline{u_\Sigma} = \left(u_1, ..., u_{N_\Sigma}\right)$. Entonces generaríamos el sistema

$$\begin{aligned} \underline{\underline{A^{II}}}\,\underline{u_I} + \underline{\underline{A^{I\Sigma}}}\,\underline{u_\Sigma} &= \underline{b_I} \\ \underline{\underline{A^{\Sigma I}}}\,\underline{u_I} + \underline{\underline{A^{\Sigma\Sigma}}}\,\underline{u_\Sigma} &= \underline{b_\Sigma} \end{aligned} \tag{1.8}$$

el cual es equivalente al sistema algebraico generado por el método de elemento finito. Pero si ahora despejamos $\underline{u_I}$ del sistema anterior, tenemos el sistema lineal

$$\left(\underline{\underline{A^{\Sigma\Sigma}}} - \underline{\underline{A^{\Sigma I}}}\left(\underline{\underline{A^{II}}}\right)^{-1}\underline{\underline{A^{I\Sigma}}}\right)\underline{u_\Sigma} = \underline{b_\Sigma} - \underline{\underline{A^{\Sigma I}}}\left(\underline{\underline{A^{II}}}\right)^{-1}\underline{b_I} \tag{1.9}$$

el cual queda en términos de $\underline{u_\Sigma}$. A

$$\underline{\underline{S}} \equiv \underline{\underline{A^{\Sigma\Sigma}}} - \underline{\underline{A^{\Sigma I}}}\left(\underline{\underline{A^{II}}}\right)^{-1}\underline{\underline{A^{I\Sigma}}} \tag{1.10}$$

comúnmente se le llama el complemento de Schur.

Pero como tenemos definidas en forma local en cada Ω_δ a las matrices $\underline{\underline{A_\delta^{II}}}, \underline{\underline{A_\delta^{\Sigma I}}}, \underline{\underline{A_\delta^{I\Sigma}}}$ y $\underline{\underline{A_\delta^{\Sigma\Sigma}}}$, entonces definimos el complemento de Schur local por $\underline{\underline{S_\delta}} = \underline{\underline{A_\delta^{\Sigma\Sigma}}} - \underline{\underline{A_\delta^{\Sigma I}}}\left(\underline{\underline{A_\delta^{II}}}\right)^{-1}\underline{\underline{A_\delta^{I\Sigma}}}$, generándose el sistema virtual cuyas incógnitas son los nodos de la frontera interior

$$\left[\sum_{\delta=1}^{E}\underline{\underline{S_\delta}}\right]\underline{u_\Sigma} = \left[\sum_{\delta=1}^{E}\underline{b_i}\right] \tag{1.11}$$

donde

$$\underline{b_i} = \underline{\underline{A_\delta^{\Sigma I}}}\left(\underline{\underline{A_\delta^{II}}}\right)^{-1}\underline{b_I}. \tag{1.12}$$

Usando el método de gradiente conjugado precondicionado, resolvemos del sistema virtual (1.10) obteniendo los nodos de la frontera interior $\underline{u_\Sigma}$, para ello es necesario hacer en cada iteración del método de gradiente conjugado una multiplicación de un vector de la dimensión de $\underline{u_\Sigma}$ con todos los $\underline{\underline{S_\delta}}$, esto se logra hacer pasando sólo las entradas correspondientes de los nodos pertenecientes al subdominio Ω_δ y así multiplicarlo por $\underline{\underline{S_\delta}}$.

Una vez conocido el vector $\underline{u_\Sigma}$ se calcula $\underline{u_I}$ en cada Ω_δ usando

$$\underline{u_{I_\delta}} = \left(\underline{\underline{A_\delta^{II}}}\right)^{-1}\left(\underline{b_{I_\delta}}\,\underline{\underline{A_\delta^{\Sigma I}}}\,\underline{u_{\Sigma_\delta}}\right). \tag{1.13}$$

El número de condicionamiento del la matriz resultante del sistema algebraico (1.19) del método de subestructuración es del orden de $1/h$ -donde h es el diámetro de la malla- en constaste al $1/h^2$ del método de elemento finito o diferencias finitas.

3 MODELO COMPUTACIONAL

La solución de los sistemas continuos usando ecuaciones diferenciales parciales en su discretización genera un sistema lineal algebraico asociado al problema, este requiere una gran cantidad de memoria e involucra un tiempo grande en su resolución; por ello nos interesa trabajar en computadoras que puedan satisfacer estas demandas.

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 el programa funciona en equipos paralelos y pueden usar toda la memoria compartida de dichos equipos, el algoritmo ejecutado continua siendo secuencial en la gran mayoría 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 nano segundos-,

pero el resto de los datos deberán de ser accesados mediante red -con acceso del orden de mili segundos-, siendo esto muy costoso en tiempo de procesamiento.

Por ello, si usamos métodos de descomposición de dominio es posible hacer que el sistema algebraico asociado pueda distribuirse en la memoria local de múltiples computadoras y que para encontrar la solución al problema se requiera poca comunicación entre los procesadores.

La alternativa más adecuada en costo y flexibilidad, es trabajar con computadoras de escritorio interconectadas por red que pueden usarse de manera cooperativa para resolver nuestro problema. Los gastos en la interconexión de los equipos son mínimos (sólo el switch y una tarjeta de red por equipo y cables para su interconexión). Por ello los clusters y los grids son en principio una buena opción para la resolución de sistemas de ecuaciones diferenciales parciales por los métodos de descomposición de dominio.

En el caso de métodos de descomposición de dominio del tipo subestructuración el mejor esquema de paralelización es el Maestro-Esclavo mostrado en la figura 3, aquí la implementación del método de descomposición de dominio se hará en el lenguaje de programación C++ bajo la interfaz de paso de mensajes MPI trabajando en un cluster Linux Debian. Donde tomando en cuenta la implementación en estrella del cluster, el modelo de paralelismo de MPI y las necesidades propias de comunicación del programa, el nodo maestro tendrá comunicación sólo con cada nodo esclavo y no existirá comunicación entre los nodos esclavos, esto reducirá las comunicaciones y optimizará el paso de mensajes.

El esquema de paralelización Maestro-Esclavo, permite sincronizar por parte del nodo maestro las tareas que se realizan en paralelo usando varios nodos esclavos, este modelo puede ser explotado de manera eficiente si existe poca comunicación entre el nodo maestro y los nodos esclavos, tratando que los tiempos consumidos en realizar las tareas asignadas sean 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.

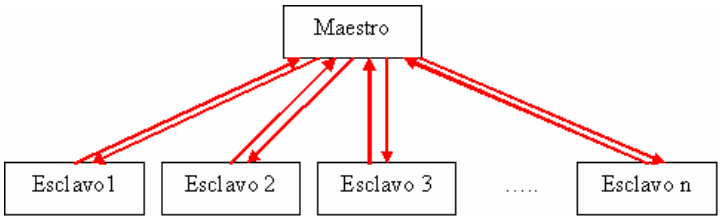

Figura 3 – Esquema de paralelización Maestro-Esclavo.

Un factor limitante en este esquema es que 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 el esquema Maestro-Esclavo en un cluster heterogéneo en donde el nodo maestro sea más poderoso computacionalmente que los nodos esclavos. 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 el esquema se verá degradado inexorablemente al aumentar el número de nodos esclavos.

4 IMPLEMENTACION COMPUTACIONAL

A partir del modelo matemático y el modelo numérico usados para hacer la discretización del problema, aquí describiremos el modelo computacional que estará 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 y se mostrarán 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.

Primeramente hay que destacar que el paradigma de programación orientada a objetos es un método de implementación de programas, organizándolos 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 sencillo. Así, mediante la herencia, contención, agregación o uso 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) 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.

A partir de la formulación del método de elemento finito FEM, la implementación computacional que se desarrolló [2] tiene la jerarquía de clases mostrada en la figura 4.

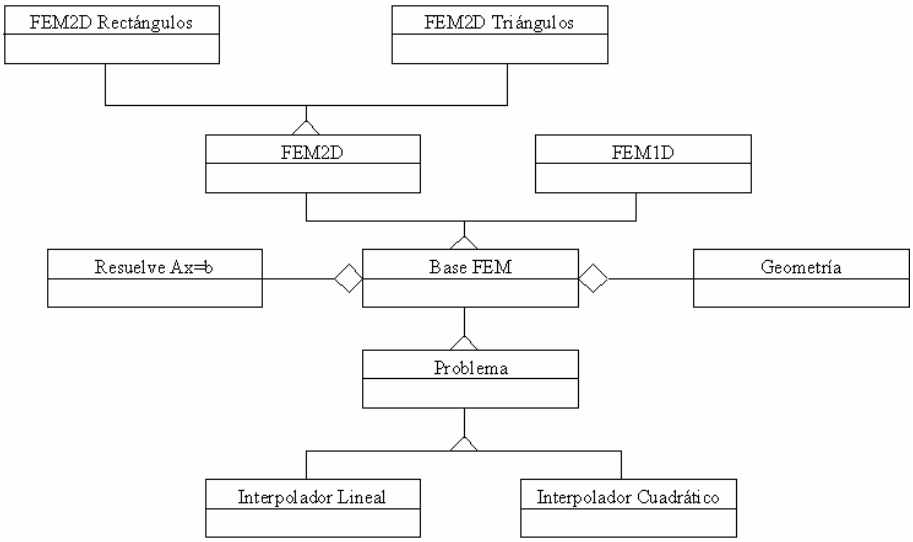

Figura 4 – Jerarquía de clases para el método de elemento finito.

Donde las clases participantes en el método de elemento finito en dos dimensiones usando una discretización del dominio por medio de rectángulos *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 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 discretización de la ecuación diferencial parcial.

La implementación computacional que se desarrolló para el método de subestructuración [2] 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 rehúsa todo el código desarrollado para *FEM2D Rectángulos*, la jerarquía queda como lo muestra la figura 5.

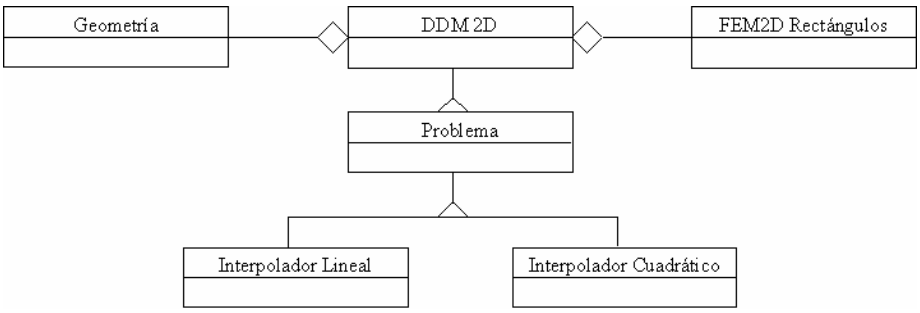

Figura 5 – Jerarquía de clases para el método de subestructuración.

La clase *DDM2D* realiza la partición gruesa del dominio Ω mediante la clase *Geometría* y controla la partición de cada subdominio mediante un objeto de la clase *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 precondicionado, necesaria para resolver los nodos internos de cada subdominio.

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 [2], es por ello que usaremos los precondicionadores a priori, en particular el derivado de la matriz de rigidez. Estos son más particulares y su construcción depende del proceso que origina el sistema lineal algebraico [1], [6] y [8].

La implementación de los precondicionadores a priori para el método de gradiente conjugado, 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 usando 81×81 nodos en la cual se toma una partición rectangular gruesa de 4×4 subdominios y cada subdominio se descompone en 20×20 subdominios -usando para el cálculo en un procesador el equipo secuencial y para la parte paralela el cluster homogéneo- resolviendo el sistema lineal algebraico asociado por el método de gradiente conjugado precondicionado, la solución se encontró en 47 iteraciones (una mejora en promedio cercana al 50 % con respecto a no usar precondicionador, 92 iteraciones) obteniendo cuando hay un buen balanceo de cargas -cuando se tienen 2, 3, 5, 9, 17 procesadores- los siguientes resultados [2], cuyas métricas de desempeño se muestran en la figura 6.

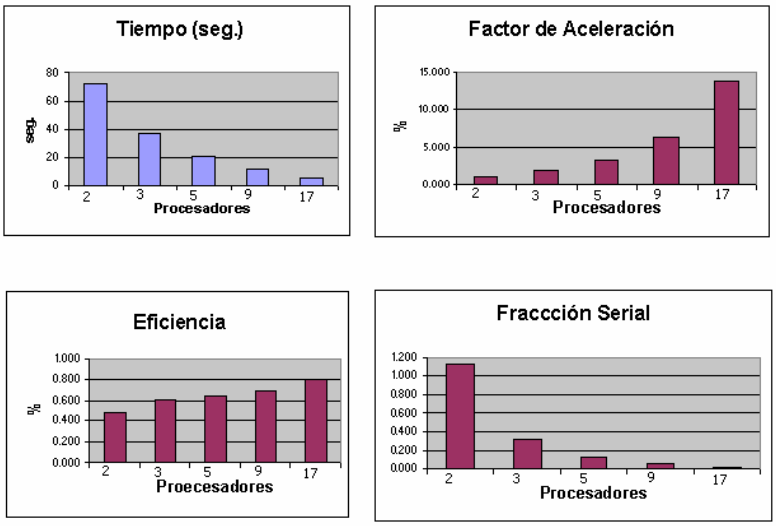

Figura 6 – Métricas de desempeño mostrando sólo cuando las cargas están bien balanceadas (2, 3, 5, 9 y 17 procesadores).

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, en nuestro caso no es lineal pero cumple bien este hecho. En la eficiencia, su valor deberá ser cercano a uno cuando el hardware es usado de manera eficiente, como es en nuestro caso. Y en la fracción serial, su valor debiera de tender a cero en el caso ideal, siendo este nuestro caso.

5 ANÁLISIS DE RENDIMIENTO

Ahora haremos algo del análisis de rendimiento sin llegar a ser exhaustivo, veremos los factores más importantes que merman el rendimiento de la aplicación, así como, 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.

Para el análisis de comunicaciones entre el nodo principal y los nodos esclavos en el método de subestructuración paralelo precondicionado es necesario conocer qué se trasmite y su tamaño, 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 y suponiendo una partición del dominio Ω en una descomposición $(n \times m \text{ y } p \times q)$, el nodo maestro haciendo la agregación de un objeto de la clase *Geometría* genera la descomposición gruesa del dominio y los nodos esclavos crean un conjunto de objetos *FEM2D Rectángulos* para que en estos objetos se genere la participación fina de cada Ω_δ 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 mediante la agregación de un objeto de la clase *Geometría*, esta geometría es pasada a los nodos esclavos mediante la transmisión de 4 coordenadas en 2D correspondientes a la delimitación del dominio.

B) Con esa geometría se construyen los objetos *FEM2D Rectángulos* (uno por cada subdominio Ω_δ), donde cada subdominio es particionado. Cada objeto de *FEM2D Rectángulos* genera la geometría solicitada, regresando $2*p*q$ coordenadas de los nodos de frontera del subdominio Ω_δ, 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 -a lo más $n*m*2*p*q$ - 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 -a lo más $2*p*q$ -.

D) Después de conocer los nodos de la frontera interior, cada objeto *FEM2D Rectángulos* calcula las matrices $\underline{\underline{A_\delta^{II}}}, \underline{\underline{A_\delta^{\Sigma I}}}, \underline{\underline{A_\delta^{I\Sigma}}}$ y $\underline{\underline{A_\delta^{\Sigma\Sigma}}}$ necesarias para construir el complemento de Schur local $\underline{\underline{S_\delta}} = \underline{\underline{A_\delta^{\Sigma\Sigma}}} - \underline{\underline{A_\delta^{\Sigma I}}} \left(\underline{\underline{A_\delta^{II}}} \right)^{-1} \underline{\underline{A_\delta^{I\Sigma}}}$, sin realizar comunicación alguna. Al terminar de calcular las matrices se avisa al nodo maestro mediante un solo mensaje de la finalización de los cálculos.

E) Mediante la comunicación de vectores del tamaño del número de nodos de la frontera interior –a lo más $2*p*q$ coordenadas- 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 $\left[\sum_{\delta=1}^{E} \underline{\underline{S_\delta}} \right] \underline{u_\Sigma} = \left[\sum_{\delta=1}^{E} \underline{b_i} \right]$.

F) Para usar el método de gradiente conjugado precondicionado –recordando que los subdominios Ω_δ en general están en distintos procesadores-, se transmite un vector del tamaño del número de nodos de la frontera interior –a lo más $2*p*q$ coordenadas- 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 $\underline{u_\Sigma}$.

G) Al término de las iteraciones se pasa la solución $\underline{u_\Sigma}$ de los nodos de la frontera interior –a lo más $2*p*q$ coordenadas- que pertenecen a cada subdominio dentro de cada objeto *FEM2D Rectángulos* para que se resuelvan los nodos interiores $\underline{u_I}$, sin realizar comunicación alguna en el proceso mediante $\underline{u_{I_\delta}} = \left(\underline{\underline{A_\delta^{II}}}\right)^{-1}\left(\underline{b_{I_\delta}}\,\underline{\underline{A_\delta^{\Sigma I}}}\,\underline{u_{\Sigma_\delta}}\right)$, al concluir se avisa mediante un único mensaje al nodo maestro de ello.

I) Mediante un último mensaje el nodo maestro 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 todos los incisos.

Una parte no significativa del tiempo de ejecución 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, F* y *G* del algoritmo consumen la mayor parte del tiempo 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 de los nodos de la frontera interior.

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

La transmisión de información entre el nodo maestro y los nodos esclavos 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.

Por otro lado, tenemos la afectación del rendimiento al aumentar el número de subdominios en la descomposición, ya que el complemento de Schur $\underline{\underline{S_\delta}} = \underline{\underline{A_\delta^{\Sigma\Sigma}}} - \underline{\underline{A_\delta^{\Sigma I}}}\left(\underline{\underline{A_\delta^{II}}}\right)^{-1}\underline{\underline{A_\delta^{I\Sigma}}}$ involucra el generar las matrices $\underline{\underline{A_\delta^{II}}}, \underline{\underline{A_\delta^{\Sigma I}}}, \underline{\underline{A_\delta^{I\Sigma}}}$ y $\underline{\underline{A_\delta^{\Sigma\Sigma}}}$ y calcular de alguna forma $\left(\underline{\underline{A_\delta^{II}}}\right)^{-1}$. Si el número de nodos interiores en el subdominio es grande entonces obtener la matriz $\left(\underline{\underline{A_\delta^{II}}}\right)^{-1}$ y hacer los productos involucrados en el complemento de Schur serán muy costosos computacionalmente. 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 inexorablemente al aumentar el número de nodos esclavos. Por ello hay que ser cuidadosos en cuanto al número de nodos esclavos que se usan en la implementación, es decir, hay que tomar en cuenta el tiempo de ejecución versus el rendimiento general del sistema, al aumentar el número de nodos esclavos.

Por último, una de las grandes ventajas de los métodos de descomposición de domino 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, 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 domino Ω estará sujeta a que cada subdominio Ω_δ 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. Esta 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, éstas y otras pruebas de análisis de rendimiento están detalladas en [2].

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 implementación.

6 CONCLUSIONES Y TRABAJO FUTURO

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.

Al hacer el 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, en el análisis de rendimiento se vieron los alcances y limitaciones de esta metodología, permitiendo tener cotas para conocer las diversas descomposiciones que son posibles generar para un número de procesadores fijo, como para conocer el número de procesadores necesarios en la resolución de un problema particular. También se vió 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 cómo resolverlo 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.

Pese a que en este trabajo sólo se muestra un método de descomposición de dominio, es posible adaptar la metodología usada a muchos otros métodos de descomposición de dominio (como Trefftz-Herrera, FETI, Multigrid, entre otros) donde cada uno de ellos acepta interpoladores de distintos grados. Permitiendo tener un grupo de herramientas que pueden ser usadas en múltiples problemas escogiendo la que ofrezca mayores ventajas computacionales, pero las ventajas y desventajas deberán de ser sopesadas al implementarse en una arquitectura computacional particular.

Además, a cada método de descomposición de dominio se pueden construir diversos precondicionadores a priori, con el objetivo de obtener un balance entre la complejidad de los diversos precondicionadores (aunado al del método de descomposición de dominio) y el aumento del rendimiento, tomando en cuenta que el precondicionador óptimo para un problema particular puede involucrar mucho trabajo de programación.

Finalmente, estos métodos no están constreñidos a problemas elípticos, pueden adaptarse a problemas parabólicos e hiperbólicos, tanto lineales como no lineales, permitiendo así atacar una gran gama de problemas en medios continuos, para más detalles ver [1], [5], [7] y [8].

7 REFERENCIAS

[1] 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).

[2] A. Carrillo; Aplicaciones del Cómputo en Paralelo a la Modelación de Sistemas Terrestres (Tesis de Maestría). Instituto de Geofísica, UNAM, (2006).

[3] W. Gropp, E. Lusk, A. Skjellem, Using MPI, Portable Parallel Programming Whit the Message Passing Interface. Scientific and Engineering Computation Series, 2ed, (1999).

[4] I. Herrera; Método de Subestructuración (Notas de Curso en Preparación). Instituto de Geofísica, (UNAM).

[5] A. Quarteroni, A. Valli; *Domain Decomposition Methods for Partial Differential Equations*. Clarendon Press Oxford (1999).

[6] B. F. Smith, P. E. Bj∅rstad, W. D. Gropp; *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, (1996).

[7] A. Toselli, O. Widlund; *Domain Decomposition Methods - Algorithms and Theory*. Springer, (2005).

[8] B. I. Wohlmuth; *Discretization Methods and Iterative Solvers Based on Domain Decomposition*. Springer, (2003).