Introducción al Método de Diferencias Finitas y su Implementación Computacional

Antonio Carrillo Ledesma,
Karla Ivonne González Rosas y Omar Mendoza Bernal
Facultad de Ciencias, UNAM
http://www.mmc.geofisica.unam.mx/acl/

Una copia de este trabajo se puede descargar de la página
http://www.mmc.geofisica.unam.mx/acl/Textos/

Primavera 2018, Versión 1.0β

Índice

1 Expansión en Series de Taylor

Sea $f(x)$ una función definida en (a, b) que tiene hasta la $k-$ésima derivada, entonces la expansión de $f(x)$ usando series de Taylor alrededor del punto x_i contenido en el intervalo (a, b) será

$$f(x) = f(x_i) + \frac{(x - x_i)}{1!} \left.\frac{df}{dx}\right|_{x_i} + \frac{(x - x_i)^2}{2!} \left.\frac{d^2 f}{dx^2}\right|_{x_i} + ... + \frac{(x - x_i)^k}{k!} \left.\frac{d^k f}{dx^k}\right|_{\varepsilon} \quad (1.1)$$

donde $\varepsilon = x_i + \theta(x - x_i)$ y $0 < \theta < 1$.

1.1 Aproximación de la Primera Derivada

Existen distintas formas de generar la aproximación a la primera derivada, nos interesa una que nos de la mejor precisión posible con el menor esfuerzo computacional.

1.1.1 Diferencias Progresivas

Considerando la Ec.(1.1) con $k = 2$ y $x = x_i + \Delta x$, tenemos

$$f(x_i + \Delta x) = f(x_i) + \Delta x \left.\frac{df}{dx}\right|_{x_i} + \frac{\Delta x^2}{2!} \left.\frac{d^2 f}{dx^2}\right|_{\varepsilon_p} \quad (1.2)$$

de esta ecuación obtenemos la siguiente expresión para la aproximación de la primera derivada

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x} - \frac{\Delta x}{2!} \left.\frac{d^2 f}{dx^2}\right|_{\varepsilon_p} \quad (1.3)$$

en este caso la aproximación de $f'(x)$ mediante diferencias progresivas es de primer orden, es decir $O(\Delta x)$. Siendo $O_p(\Delta x)$ el error local de truncamiento, definido como

$$O_p(\Delta x) = -\frac{\Delta x}{2!} \left.\frac{d^2 f}{dx^2}\right|_{\varepsilon_p}. \quad (1.4)$$

Es común escribir la expresión anterior como

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x} - O_p(\Delta x) \quad (1.5)$$

o

$$f'(x_i) = \frac{f_{i+1} - f_i}{\Delta x} \quad (1.6)$$

para simplificar la notación.

1.1.2 Diferencias Regresivas

Considerando la Ec.(1.1) con $k = 2$ y $x = x_i - \Delta x$, tenemos

$$f(x_i - \Delta x) = f(x_i) - \Delta x \left.\frac{df}{dx}\right|_{x_i} + \frac{\Delta x^2}{2!} \left.\frac{d^2 f}{dx^2}\right|_{\varepsilon_r} \tag{1.7}$$

de esta ecuación obtenemos la siguiente expresión para la aproximación de la primera derivada

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f(x_i) - f(x_i - \Delta x)}{\Delta x} - \frac{\Delta x}{2!} \left.\frac{d^2 f}{dx^2}\right|_{\varepsilon_r} \tag{1.8}$$

en este caso la aproximación de $f'(x)$ mediante diferencias regresivas es de primer orden, es decir $O(\Delta x)$. Siendo $O_r(\Delta x)$ el error local de truncamiento, definido como

$$O_r(\Delta x) = \frac{\Delta x}{2!} \left.\frac{d^2 f}{dx^2}\right|_{\varepsilon_r}. \tag{1.9}$$

Es común escribir la expresión anterior como

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f(x_i) - f(x_i - \Delta x)}{\Delta x} + O_r(\Delta x) \tag{1.10}$$

o

$$f'(x_i) = \frac{f_i - f_{i-1}}{\Delta x} \tag{1.11}$$

para simplificar la notación.

1.1.3 Diferencias Centradas

Considerando la Ec.(1.1) con $k = 3$ y escribiendo $f(x)$ en $x = x_i + \Delta x$ y $x = x_i - \Delta x$, tenemos

$$f(x_i + \Delta x) = f(x_i) + \Delta x \left.\frac{df}{dx}\right|_{x_i} + \frac{\Delta x^2}{2!} \left.\frac{d^2 f}{dx^2}\right|_{x_i} + \frac{\Delta x^3}{3!} \left.\frac{d^3 f}{dx^3}\right|_{\varepsilon_p} \tag{1.12}$$

y

$$f(x_i - \Delta x) = f(x_i) - \Delta x \left.\frac{df}{dx}\right|_{x_i} + \frac{\Delta x^2}{2!} \left.\frac{d^2 f}{dx^2}\right|_{x_i} - \frac{\Delta x^3}{3!} \left.\frac{d^3 f}{dx^3}\right|_{\varepsilon_r} \tag{1.13}$$

restando la Ec.(1.12) de la Ec.(1.13), se tiene

$$f(x_i + \Delta x) - f(x_i - \Delta x) = 2\Delta x \left.\frac{df}{dx}\right|_{x_i} + \frac{\Delta x^3}{3!} \left[\left.\frac{d^3 f}{dx^3}\right|_{\varepsilon_p} + \left.\frac{d^3 f}{dx^3}\right|_{\varepsilon_r} \right] \tag{1.14}$$

esta última expresión lleva a la siguiente aproximación de la primera derivada mediante diferencias centradas

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f(x_i + \Delta x) - f(x_i - \Delta x)}{2\Delta x} + O_c(\Delta x^2) \tag{1.15}$$

con un error local de truncamiento de segundo orden $O_c(\Delta x^2)$, es decir

$$O_c(\Delta x^2) = \frac{\Delta x^2}{3!}\left[\left.\frac{d^3 f}{dx^3}\right|_{\varepsilon_p} + \left.\frac{d^3 f}{dx^3}\right|_{\varepsilon_r}\right] \tag{1.16}$$

comparado el error local de truncamiento de la aproximación anterior $O_c(\Delta x^2)$, con los obtenidos previamente para diferencias progresivas $O_p(\Delta x)$ Ec.(1.5) y regresivas $O_r(\Delta x)$ Ec.(1.10), se tiene que

$$\lim_{\Delta x \to 0} O_c(\Delta x^2) < \lim_{\Delta x \to 0} O_p(\Delta x). \tag{1.17}$$

Es común encontrar expresada la derivada[1]

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f(x_i + \Delta x) - f(x_i - \Delta x)}{2\Delta x} \tag{1.18}$$

como

$$f'(x_i) = \frac{f_{i+1} - f_{i-1}}{2\Delta x} \tag{1.19}$$

para simplificar la notación.

Derivadas Usando Más Puntos Utilizando el valor de la función en más puntos se construyen fórmulas más precisas para las derivadas[2], algunos ejemplos son

$$\begin{aligned}
f'(x_i) &= \frac{-3f_i + 4f_{i+1} - f_{i+2}}{2\Delta x} + O(\Delta x^2) \qquad (1.20)\\
f'(x_i) &= \frac{3f_i - 4f_{i-1} + f_{i-2}}{2\Delta x} + O(\Delta x^2)\\
f'(x_i) &= \frac{2f_{i+1} + 3f_i - 6f_{i-1} + f_{i-2}}{6\Delta x} + O(\Delta x^3)\\
f'(x_i) &= \frac{f_{i-2} - 8f_{i-1} + 8f_{i+1} - f_{i+2}}{12\Delta x} + O(\Delta x^4)\\
f'(x_i) &= \frac{-25f_i + 48f_{i+1} - 36f_{i+2} + 16f_{i+3} - 3f_{i+4}}{12\Delta x} + O(\Delta x^4)
\end{aligned}$$

1.2 Derivadas de Ordenes Mayores

De forma análoga se construyen aproximaciones en diferencias finitas de orden mayor, aquí desarrollaremos la forma de calcular la derivada de orden dos en diferencias centradas.

[1] En el caso de que la derivada sea usada en una malla no homogénea, es necesario incluir en la derivada a que Δx se refiere, por ejemplo en cada punto i, tenemos la Δx_{i-} (por la izquierda) y la Δx_{i+} (por la derecha), i.e. $\left.\frac{df}{dx}\right|_{x_i} = \frac{f(x_i+\Delta x_{i-}) - f(x_i - \Delta x_{i+})}{(\Delta x_{i-}) + (\Delta x_{i+})}$.

[2] Al usar estas derivadas en el método de diferencias finitas mostrado en la sección (2) las matrices generadas no serán tridiagonales.

1.2.1 Derivada de Orden Dos

Partiendo del desarrollo de Taylor

$$f(x_i+\Delta x) = f(x_i)+\Delta x f'(x_i)+\frac{\Delta x^2}{2!}f''(x_i)+\frac{\Delta x^3}{3!}f'''(x_i)+\frac{\Delta x^4}{4!}f^{(4)}(\xi_p) \quad (1.21)$$

y

$$f(x_i-\Delta x) = f(x_i)-\Delta x f'(x_i)+\frac{\Delta x^2}{2!}f''(x_i)-\frac{\Delta x^3}{3!}f'''(x_i)+\frac{\Delta x^4}{4!}f^{(4)}(\xi_r) \quad (1.22)$$

eliminando las primeras derivadas, sumando las ecuaciones anteriores y despejando se encuentra que

$$f''(x_i) = \frac{f(x_i-\Delta x)-2f(x_i)+f(x_i+\Delta x)}{\Delta x^2} - \frac{\Delta x^2}{12}f^{(4)}(\xi_c) \quad (1.23)$$

así, la aproximación a la segunda derivada usando diferencias centradas con un error de truncamiento $O_c(\Delta x^2)$ es[3]

$$f''(x_i) = \frac{f(x_i-\Delta x)-2f(x_i)+f(x_i+\Delta x)}{\Delta x^2} \quad (1.24)$$

Es común escribir la expresión anterior como

$$f''(x_i) = \frac{f_{i-1}-2f_i+f_{i+1}}{\Delta x^2}$$

para simplificar la notación.

Derivadas Usando Más Puntos Utilizando el valor de la función en más puntos se construyen fórmulas más precisas para las derivadas, algunos ejemplos son

$$\begin{aligned} f''(x_i) &= \frac{f_{i+2}-2f_{i+1}+f_i}{\Delta x^2}+O(\Delta x) \quad (1.25)\\ f''(x_i) &= \frac{-f_{i+3}+4f_{i+2}-5f_{i+1}+2f_i}{\Delta x^2}+O(\Delta x^2)\\ f''(x_i) &= \frac{-f_{i+2}+16f_{i+1}-30f_i+16f_{i-1}-f_{i-2}}{12\Delta x^2}+O(\Delta x^4) \end{aligned}$$

[3] En el caso de que la derivada sea usada en una malla no homogénea, es necesario incluir en la derivada a que Δx se refiere, por ejemplo en cada punto i, tenemos la Δx_{i-} (por la izquierda) y la Δx_{i+} (por la derecha), i.e. $f''(x_i) = \frac{f(x_i-\Delta x_{i-})-2f(x_i)+f(x_i+\Delta x_{i+})}{(\Delta x_{i-})(\Delta x_{i+})}$

1.2.2 Derivadas de Orden Tres y Cuatro

De forma análoga se construyen derivadas de ordenes mayores utilizando el valor de la función en más puntos, algunos ejemplos para terceras derivadas son

$$\begin{aligned} f'''(x_i) &= \frac{f_{i+3} - 3f_{i+2} + 3f_{i+1} - f_i}{\Delta x^3} + O(\Delta x) \\ f'''(x_i) &= \frac{f_{i+2} - 2f_{i+1} + 2f_{i-1} - f_{i-2}}{2\Delta x^3} + O(\Delta x^2) \\ f'''(x_i) &= \frac{f_{i-3} - 8f_{i-2} + 13f_{i-1} - 13f_{i+1} + 8f_{i+2} - f_{i+3}}{8\Delta x^3} + O(\Delta x^4) \end{aligned} \tag{1.26}$$

Algunos ejemplos para cuartas derivadas son

$$\begin{aligned} f''''(x_i) &= \frac{f_{i+4} - 4f_{i+3} + 6f_{i+2} - 4f_{i+1} + f_i}{\Delta x^4} + O(\Delta x) \\ f''''(x_i) &= \frac{f_{i+2} - 4f_{i+1} + 6f_i - 4f_{i-1} + f_{i-2}}{\Delta x^4} + O(\Delta x^2) \\ f''''(x_i) &= \frac{-f_{i+3} + 12f_{i+2} - 39f_{i+1} + 56f_i - 39f_{i-1} + 12f_{i-2} - f_{i-3}}{6\Delta x^4} + O(\Delta x^4) \end{aligned} \tag{1.27}$$

2 Método de Diferencias Finitas en Una Dimensión

Consideremos la ecuación diferencial parcial

$$(p(x)u'(x))' + q(x)u'(x) - r(x)u(x) = f(x) \tag{2.1}$$

$$\text{en } a \leq x \leq b \qquad \text{donde: } u(a) = u_\alpha \text{ y } u(b) = u_\beta$$

con condiciones de frontera Dirichlet o cualquier otro tipo de condiciones de frontera. Para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, debemos de:

1. Generar una malla del dominio, i.e. una malla es un conjunto finito de puntos en los cuales buscaremos la solución aproximada a la ecuación diferencial parcial.
2. Sustituir las derivadas correspondientes con alguna de las formulas de diferencias finitas centradas (véase secciones 1.1 y 1.2), en cada punto donde la solución es desconocida para obtener un sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$.
3. Resolver el sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ (véase capítulo A), y así obtener la solución aproximada en cada punto de la malla.

2.1 Problema con Condiciones de Frontera Dirichlet

Consideremos un caso particular de la Ec.(2.1) definido por la ecuación

$$u''(x) = f(x), \qquad 0 \leq x \leq 1, \qquad u(0) = u_\alpha, \qquad u(1) = u_\beta \tag{2.2}$$

con condiciones de frontera Dirichlet. Para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, debemos de:

1. Generar una malla homogénea del dominio[4]

$$x_i = ih, \qquad i = 0, 1, ..., n, \qquad h = \frac{1}{n} = \Delta x \tag{2.3}$$

2. Sustituir la derivada con la Ec.(1.24) en cada punto donde la solución es desconocida para obtener un sistema algebraico de ecuaciones. Así, en cada punto x_i de la malla aproximamos la ecuación diferencial por[5]

$$u''(x_i) \approx \frac{u(x_i - h) - 2u(x) + u(x_i + h)}{h^2} \tag{2.4}$$

[4] En el caso de que la malla no sea homogénea, es necesario incluir en la derivada a que h se refiere, por ejemplo en cada punto i, tenemos la h_{i-} (por la izquierda) y la h_{i+} (por la derecha), i.e. $u''(x_i) \approx \frac{u(x_i - h_{i-}) - 2u(x) + u(x_i + h_{i+})}{(h_{i-})(h_{i+})}$.

[5] Notemos que en cada punto de la malla, la aproximación por diferencias finitas supone la solución de tres puntos de la malla x_{i-1}, x_i y x_{i+1}. El conjunto de estos tres puntos de la malla es comúnmente llamado el esténcil de diferencias finitas en una dimensión.

o en su forma simplificada

$$u''(x_i) \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} \tag{2.5}$$

definiendo la solución aproximada de $u(x)$ en x_i como u_i, entonces se genera el siguiente sistema lineal de ecuaciones

$$\begin{aligned}
\frac{u_\alpha - 2u_1 + u_2}{h^2} &= f(x_1) \\
\frac{u_1 - 2u_2 + u_3}{h^2} &= f(x_2) \\
&\vdots \\
\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} &= f(x_i) \\
&\vdots \\
\frac{u_{n-3} - 2u_{n-2} + u_{n-1}}{h^2} &= f(x_{n-2}) \\
\frac{u_{n-2} - 2u_{n-1} + u_\beta}{h^2} &= f(x_{n-1}).
\end{aligned} \tag{2.6}$$

Este sistema de ecuaciones se puede escribir como la matriz $\underline{\underline{A}}$ y los vectores $\underline{u}$ y $\underline{f}$ de la forma

$$\begin{bmatrix}
-\frac{2}{h^2} & \frac{1}{h^2} & & & & \\
\frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} & & & \\
& \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} & & \\
& & \ddots & \ddots & \ddots & \\
& & & \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} \\
& & & & \frac{1}{h^2} & -\frac{2}{h^2}
\end{bmatrix}
\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} =
\begin{bmatrix} f(x_1) - \frac{u_\alpha}{h^2} \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{n-2}) \\ f(x_{n-1}) - \frac{u_\beta}{h^2} \end{bmatrix}.$$

en este caso, podemos factorizar $1/h^2$ del sistema lineal $\underline{\underline{A}}\underline{u} = \underline{f}$, quedando como

$$\frac{1}{h^2}\begin{bmatrix}
-2 & 1 & & & & \\
1 & -2 & 1 & & & \\
& 1 & -2 & 1 & & \\
& & \ddots & \ddots & \ddots & \\
& & & 1 & -2 & 1 \\
& & & & 1 & -2
\end{bmatrix}
\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} =
\begin{bmatrix} f(x_1) - \frac{u_\alpha}{h^2} \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{n-2}) \\ f(x_{n-1}) - \frac{u_\beta}{h^2} \end{bmatrix}.$$

esta última forma de expresar el sistema lineal algebraico asociado es preferible para evitar problemas numéricos al momento de resolver el sistema lineal por métodos iterativos (véase capítulo A.2) principalmente cuando $h \to 0$.

3. Resolver el sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ (véase capítulo A), obtenemos la solución aproximada en cada punto interior de la malla. La solución completa al problema la obtenemos al formar el vector

$$\left[\begin{array}{ccccccccc} u_\alpha & u_1 & u_2 & u_3 & \cdots & u_{n-2} & u_{n-1} & u_\beta \end{array} \right].$$

Uno de los paquetes más conocidos y usados para el cálculo numérico es MatLab[6] el cual tiene un alto costo monetario para los usuarios. Por ello, una opción es usar alternativas desarrolladas usando licencia de código abierto, un par de estos paquetes son: GNU OCTAVE[7] y SCILAB[8].

Octave corre gran parte del código desarrollado para MatLab sin requerir cambio alguno, en cuanto a SCILAB es requerido hacer algunos ajustes en la codificación y ejecución. En los siguientes ejemplos[9] se mostrará como implementar la solución computacional. En la sección (G) se mostrará la implementación en diversos paquetes y lenguajes de programación.

Ejemplo 1 *Sea*

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

El programa queda implementado en OCTAVE (MatLab) como:

```
function [A,b,x] = fdm1d(xi,xf,vi,vf,n)
  N=n-2; % Nodos interiores
  h=(xf-xi)/(n-1); % Incremento en la malla
  %A = sparse(N,N); % Matriz SPARSE
  A=zeros(N,N); % Matriz A
  b=zeros(N,1); % Vector b
  R=1/(h^2);
  P=-2/(h^2);
  Q=1/(h^2);
  % Primer renglon de la matriz A y vector b
```

[6] MatLab es un programa comercial para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería [http://www.mathworks.com.products/matlab.html].

[7] GNU OCTAVE es un programa open source para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería [http://www.gnu.org/software/octave].

[8] SCILAB es un programa open source para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería [http://www.scilab.org].

[9] Los ejemplos que se muestran en el presente texto se pueden descargar de la página WEB:

http://mmc.geofisica.unam.mx/acl/MDF/

o desde GitHub (https://github.com/antoniocarrillo69/MDF) mediante

git clone git://github.com/antoniocarrillo69/MDF.git

```
  A(1,1)=P;
  A(1,2)=Q;
  b(1)=LadoDerecho(xi)-vi*R;
  % Renglones intermedios de la matriz A y vector b
  for i=2:N-1
    A(i,i-1)=R;
    A(i,i)=P;
    A(i,i+1)=Q;
    b(i)=LadoDerecho(xi+h*(i-1));
  end
  % Renglon final de la matriz A y vector b
  A(N,N-1)=R;
  A(N,N)=P;
  b(N)=LadoDerecho(xi+h*N)-vf*Q;
  % Resuleve el sistema lineal Ax=b
  x=inv(A)*b;
  % Prepara la graficacion
  xx=zeros(n,1);
  zz=zeros(n,1);
  for i=1:n
    xx(i)=xi+h*(i-1);
    zz(i)=SolucionAnalitica(xx(i));
  end
  yy=zeros(n,1);
  yy(1)=vi; % Condicion inicial
  for i=1:N
    yy(i+1)=x(i);
  end
  yy(n)=vf; % Condicion inicial
  % Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
  plot(xx,[yy,zz]);
endfunction
function y=LadoDerecho(x)
  y=-pi*pi*cos(pi*x);
endfunction
function y=SolucionAnalitica(x)
  y=cos(pi*x);
endfunction
```

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

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

donde es necesario indicar el inicio (-1) y fin (2) del dominio, el valor de la condición de frontera de inicio (-1) y fin (1), además de el número de nodos

(30) en la partición. La ejecución genera la gráfica de la solución calculada por el método de diferencias finitas y la solución analítica en los mismos puntos de la malla, además devuelve la matriz[10] y los vectores A, b, x generados por la función.

En OCTAVE (MatLab) es posible introducir funciones para que pasen cómo parámetros a otra función, de tal forma que se puede definir un programa genérico de diferencias finitas en una dimensión con condiciones de frontera Dirichlet en el cual se especifiquen los parámetros de la ecuación en la línea de comando de OCTAVE.

Usando este hecho, implementamos un programa para codificar el Método de Diferencias Finitas en una Dimensión para resolver el problema con condiciones Dirichlet

$$p(x)u'' + q(x)u' + r(x)u = f(x)$$

definido en el dominio $[xi, xf]$ y valores en la frontera de $u(xi) = vi$ y $u(xf) = vf$, además se le indica el tamaño de la partición n, si se graficará la solución indicando en $grf = 1$, con solución analítica $s(x)$ y si a ésta se proporciona entonces $sws = 1$. Regresando la matriz y los vectores $\underline{\underline{A}}\underline{u} = \underline{b}$ del sistema lineal generado así como los puntos del dominio y los valores de la solución en dichos puntos x, V, para cada problema que deseemos resolver.

Ejemplo 2 *El programa queda implementado en OCTAVE (MatLab) como:*

```
function [error,A,b,u,x,V] = fdm1d_DD(p,q,r,f,xi,xf,vi,vf,n,grf,s,sws)
    if n < 3
        return
    end
    % Numero de incognitas del problema
    tm = n -2;
    % Declaracion de la matriz y vectores de trabajo
    %A = sparse(tm,tm);
    A = zeros(tm,tm); % Matriz de carga
    b = zeros(tm,1); % Vector de carga
    u = zeros(tm,1); % Vector de solucion
    x = zeros(n,1); % Vector de coordenadas de la particion
    V = zeros(n,1) ; % Vector solucion
    h = (xf-xi)/(n-1);
    h1 = h*h;
    % Llenado de los puntos de la malla
    for i = 1: n,
        x(i) = xi + (i-1)*h;
```

[10] En Octave (MatLab) se define a una matriz mediante A = zeros(N,N), este tipo de matriz no es adecuada para nuestros fines (véase capítulo A). Para ahorrar espacio y acelerar los cálculos numéricos que se requieren para resolver el sistema lineal asociado usamos un tipo de matriz que no guarda valores innecesarios (ceros), esto se hace mediante la declaración de la matriz como A = sparse(N,N).

```
end
% Llenado de la matriz y vector
b(1) = f(xi) - p(xi)*(vi/h1);
A(1,2) = p(x(1))/h1 - q(x(1))/(2.0*h);
A(1,1) = ((-2.0 * p(x(1))) / h1) + r(x(1));
for i=2:tm-1,
    A(i,i-1) = p(x(i))/h1 - q(x(i))/(2.0*h);
    A(i,i) = ((-2.0 * p(x(i))) / h1) + r(x(i));
    A(i,i+1) = p(x(i))/h1 + q(x(i))/(2.0*h);
    b(i) = f(x(i));
end
A(tm,tm-1) = p(x(tm))/h1 - q(x(tm))/(2.0*h);
A(tm,tm) = ((-2.0 * p(x(tm))) / h1) + r(x(tm));
b(tm) = f(x(tm+1))-p(x(tm+1))*(vf/h1);
% Soluciona el sistema
u = inv(A)*b;
% Copia la solucion obtenida del sistema lineal al vector solucion
V(1) = vi;
for i=1:tm,
    V(i+1) = u(i);
end
V(n) = vf;
% Encuentra el error en norma infinita usando una particion par = 10
error = 0;
if sws == 1
    par = 10;
    for i = 1: n-1,
        inc = (x(i+1)-x(i))/par;
        for j = 1:par+1,
            px = x(i)+inc*(j-1);
            e = abs(s(px)-l(px,x(i),V(i),x(i+1),V(i+1)));
            if e > error
                error = e;
            end
        end
    end
end
% Revisa si se graficara
if grf == 1
    if sws == 1
        % Calcula la solucion analitica en la particion de calculo
        ua = zeros(n,1);
        for i = 1: n,
            ua(i) = s(x(i));
        end
    end
```

```
            % Graficar la solucion numerica
            plot(x,V,'o');
            hold
            % Graficar la solucion analitica en una particion de tamano xPart
            if sws == 1
                xPart = 1000;
                h = (xf-xi)/(xPart-1);
                xx = zeros(xPart,1);
                xa = zeros(xPart,1);
                for i = 1: xPart,
                    xx(i) = xi + (i-1)*h;
                    xa(i) = s(xx(i));
                end
                plot(xx,xa);
                % Grafica el error
                figure(2);
                plot(x,V-ua);
            end
        end
    end
    % Evalua el punto x en la recta dada por los puntos (x1,y1) y (x2,y2), se
usa para el calculo de la norma infinito
    function y = l(x,x1,y1,x2,y2)
        y = y1+((y2-y1)/(x2-x1))*(x-x1);
    end
```

De esta forma podemos implementar diversos problemas y ver su solución

Ejemplo 3 *Sea*

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

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:

```
p=@(x) 1;
q=@(x) 0;
r=@(x) 0;
f=@(x) -pi*pi*cos(pi*x);
s=@(x) cos(pi*x);
[error,A,b,u,x,V] = fdm1d_DD(p,q,r,f,-1,2,-1,1,11,1,s,1);
```

Otro ejemplo:

Ejemplo 4 *Sea*

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

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:

```
p=@(x) 1;
q=@(x) 0;
r=@(x) 0;
f=@(x) -pi*pi*cos(pi*x);
s=@(x) cos(pi*x);
[error,A,b,u,x,V] = fdm1d_DD(p,q,r,f,0,1,1,-1,40,1,s,1);
```

Ahora uno con un parámetro adicional (velocidad):

Ejemplo 5 *Sea*

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

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:

```
v=@(x) 1.0; % velocidad
p=@(x) -1.0;
q=@(x) v(x);
r=@(x) 0;
f=@(x) 0;
s=@(x) (exp(v(x)*x)-1.0)/(exp(v(x))-1.0);
[error,A,b,u,x,V] = fdm1d_DD(p,q,r,f,0,1,0,1,11,1,s,1);
```

En éste caso, la velocidad es $v = 1.0$ y nuestro programa lo puede resolver sin complicación alguna. Si aumentamos la velocidad a $v = 50.0$ y volvemos a correr el programa haciendo estos cambios, entonces:

Ejemplo 6

```
v=@(x) 50.0;
[error,A,b,u,x,V] = fdm1d_DN(p,q,r,f,0,1,-1,0,-1,1,11,1,s,1);
```

Aún el programa lo resuelve bien, pero si ahora subimos la velocidad a $v = 100.0$ y corremos nuevamente el programa, entonces:

Ejemplo 7

```
v=@(x) 100.0;
[error,A,b,u,x,V] = fdm1d_DN(p,q,r,f,0,1,-1,0,-1,1,11,1,s,1);
```

Entonces tendremos que la solución oscila en torno al cero. Para corregir esto, tenemos algunas opciones: aumentar el número de nodos en la malla de discretización, usar una malla no homogénea refinada adecuadamente para tener un mayor número de nodos en donde sea requerido, usar fórmulas más precisas para las derivadas o mejor aún, usar diferentes esquemas tipo Upwind, Scharfetter-Gummel y Difusión Artificial para implementar el Método de Diferencias Finitas, como se muestra a continuación.

Número de Péclet Para el problema general dado por la Ec.(2.1)

$$\left(p\left(x\right)u'\left(x\right)\right)' + q\left(x\right)u'\left(x\right) - r\left(x\right)u\left(x\right) = f\left(x\right) \tag{2.7}$$

$$\text{en } a \leq x \leq b \qquad \text{donde: } u\left(a\right) = u_\alpha \text{ y } u\left(b\right) = u_\beta$$

el término $q\left(x\right)u'\left(x\right)$ algunas veces es llamado el término advectivo[11] si u es la velocidad y puede ocasionar inestabilidad numérica en el Método de Dife-rencias Finitas como se mostró en el ejemplo anterior Ej.(7), para evitar dichas inestabilidades existen diversas técnicas de discretización mediante el análisis de la solución considerando el Número de Péclet (local y global) y su implementación en el Método de Diferencias Finitas (véase [35]) mediante el esquema Upwind, Scharfetter-Gummel y Difusión Artificial, entre otros, para ello consideremos:

- Si las funciones $p(x), q(x)$ y $r(x)$ son constantes, el esquema de diferencias finitas centradas para todas las derivadas, éste está dado por el esténcil

$$p_i\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} + q_i\frac{u_{i+1} - u_{i-1}}{2h} - r_iu_i = f_i \qquad i = 1, 2, ..., n. \tag{2.8}$$

  donde la ventaja de ésta discretización, es que el método es de segundo orden de exactitud. La desventaja es que los coeficientes de la matriz generada pueden no ser diagonal dominante si $r\left(x\right) > 0$ y $p\left(x\right) > 0$. Cuando la advección $\left|p\left(x\right)\right|$ es grande, la ecuación se comporta como la ecuación de onda.

- Tomando las funciones $p(x,t), q(x,t)$ y $r(x,t)$ más generales posibles, es necesario hacer una discretización que garantice que el método es de segundo orden de exactitud, ésto se logra mediante la siguiente discretización para $\left(p\left(x,t\right)u'\left(x,t\right)\right)'$ mediante

$$\begin{aligned}\frac{\partial}{\partial x}\left(p\frac{\partial u}{\partial x}\right)\left(x,t\right) \simeq & \left[p\left(x + \frac{\Delta x}{2}, t\right)\frac{u\left(x + \Delta x, t\right) - u\left(x,t\right)}{\Delta x}\right. \\ & \left. - p\left(x - \frac{\Delta x}{2}, t\right)\frac{u\left(x,t\right) - u\left(x - \Delta x, t\right)}{\Delta x}\right]/\Delta x\end{aligned} \tag{2.9}$$

  entonces se tiene que

$$\frac{p(u_{i+1},t)\frac{u_{i+1}+u_i}{h} - p\left(u_{i-1,t}\right)\frac{u_i - u_{i-1}}{h}}{h} + q_i\frac{u_{i+1} - u_{i-1}}{2h} - r_iu_i = f_i \tag{2.10}$$

  para $i = 1, 2, ..., n$, (véase [52] pág. 78 y 79).

- El esquema mixto, en donde se usa el esquema de diferencias finitas centradas para el término de difusión y el esquema *upwind* para el término de advección

$$\begin{aligned} p_i\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} + q_i\frac{u_{i+1} - u_{i-1}}{h} - r_iu_i &= f_i, \text{ si } q_i \geq 0 \\ p_i\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} + q_i\frac{u_i - u_{i-1}}{h} - r_iu_i &= f_i, \text{ si } q_i < 0 \end{aligned} \tag{2.11}$$

[11] Cuando la advección es fuerte, esto es cuando $\left|q\left(x\right)\right|$ es grande, la ecuación se comporta como si fuera una ecuación de onda.

el propósito es incrementar el dominio de la diagonal. Este esquema es de orden uno de exactitud y es altamente recomendable su uso si $|q(x)| \sim 1/h$, en caso de no usarse, se observará que la solución numérica oscila alrededor del cero.

2.2 Problema con Condiciones de Frontera Neumann

Consideremos el problema

$$u''(x) = f(x), \qquad 0 \leq x \leq 1 \tag{2.12}$$

con condiciones de frontera Neumann

$$\frac{du}{dx} = cte_1 \text{ en } u(0) \text{ y } \frac{du}{dx} = cte_2 \text{ en } u(1)$$

para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, primero debemos discretizar las condiciones de frontera, una manera sería usar para la primera condición de frontera una aproximación usando diferencias progresivas Ec.(1.5)

$$\left.\frac{du}{dx}\right|_{x_i} = \frac{u(x_i + h) - u(x_i)}{h}$$

quedando

$$\frac{u_1 - u_0}{h} = cte_1 \tag{2.13}$$

para la segunda condición de frontera una aproximación usando diferencias regresivas Ec.(1.10)

$$\left.\frac{du}{dx}\right|_{x_i} = \frac{u(x_i) - u(x_i - h)}{h}$$

quedando

$$\frac{u_n - u_{n-1}}{h} = cte_2 \tag{2.14}$$

pero el orden de aproximación no sería el adecuado pues estamos aproximando el dominio con diferencias centradas con un error local de truncamiento de segundo orden $O_c(\Delta x^2)$, en lugar de ello usaremos diferencias centradas Ec.(1.15) para tener todo el dominio con el mismo error local de truncamiento.

Para usar diferencias centradas Ec.(1.15)

$$\left.\frac{du}{dx}\right|_{x_i} = \frac{u(x_i + h) - u(x_i - h)}{2h}$$

en el primer nodo necesitamos introducir un punto de la malla ficticio $x_{-1} = (x_0 - \Delta x)$ con un valor asociado a u_{-1}, entonces

$$\frac{u_1 - u_{-1}}{2h} = cte_1 \tag{2.15}$$

así también, en el último nodo necesitamos introducir un punto de la malla ficticio $x_{n+1} = (x_n + \Delta x)$ con un valor asociado a u_{n+1}, obteniendo

$$\frac{u_{n+1} - u_{n-1}}{2h} = cte_2. \tag{2.16}$$

Éstos valores no tienen significado físico alguno, dado que esos puntos se encuentran fuera del dominio del problema. Entonces debemos de:

1. Generar una malla homogénea del dominio

$$x_i = ih, \qquad i = 0, 1, ..., n, \qquad h = \frac{1}{n} = \Delta x. \tag{2.17}$$

2. Sustituir la derivada con la Ec.(1.24)[12] en cada punto donde la solución es desconocida para obtener un sistema algebraico de ecuaciones. Así, en cada punto x_i de la malla aproximamos la ecuación diferencial por

$$u''(x_i) \approx \frac{u(x_i - h) - 2u(x) + u(x_i + h)}{h^2} \tag{2.18}$$

definiendo la solución aproximada de $u(x)$ en x_i como u_i como la solución del siguiente sistema lineal de ecuaciones

$$\begin{aligned}
\frac{u_1 - u_{-1}}{2h} &= cte_1 \\
\frac{u_0 - 2u_1 + u_2}{h^2} &= f(x_1) \\
&\vdots \\
\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} &= f(x_i) \\
&\vdots \\
\frac{u_{n-2} - 2u_{n-1} + u_n}{h^2} &= f(x_{n-1}) \\
\frac{u_{n+1} - u_{n-1}}{2h} &= cte_2.
\end{aligned} \tag{2.19}$$

3. Resolver el sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ (véase capítulo A), obtenemos la solución aproximada en cada punto de la malla.

2.3 Problema con Condiciones de Frontera Robin

El método de un punto de la malla ficticio es usado para el manejo de las condiciones de frontera mixta, también conocidas como condiciones de frontera Robin. Sin perdida de generalidad, supongamos que en $x = a$, tenemos

$$\alpha u'(a) + \beta u(b) = \gamma$$

[12] Para mantener la estabilidad es necesario tomar en cuenta las distintas formas de discretización en el Método de Diferencias Finitas mediante el esquema Upwind, Scharfetter-Gummel y Difusión Artificial, entre otros (véase sección: 2.1, Ecs. 2.8 a 2.11).

donde $\alpha \neq 0$. Entonces usando el punto de la malla ficticio, tenemos que

$$\alpha \frac{u_1 - u_{-1}}{2h} + \beta u_n = \gamma$$

o

$$u_{-1} = u_1 + \frac{2\beta}{\alpha} u_n - \frac{2h\gamma}{\alpha}$$

introduciendo esto en términos de diferencias finitas centradas, en $x = x_0$, entonces se tiene que

$$\left(-\frac{2}{h^2} + \frac{2\beta}{\alpha h} \right) u_n + \frac{2}{h^2} u_1 = f_0 + \frac{2\gamma}{\alpha h}$$

o

$$\left(-\frac{1}{h^2} + \frac{\beta}{\alpha h} \right) u_n + \frac{1}{h^2} u_1 = \frac{f_0}{2} + \frac{\gamma}{\alpha h}$$

lo que genera coeficientes simétricos en la matriz.

Consideremos el problema

$$u''(x) = f(x), \qquad 0 \leq x \leq 1 \tag{2.20}$$

con condiciones de frontera Dirichlet y Neumann

$$u(0) = u_\alpha \qquad \text{y} \qquad \frac{du}{dx} = cte_1 \text{ en } u(1).$$

respectivamente. Para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, primero debemos expresar la condición de frontera Neumann mediante diferencias centradas Ec.(1.15)

$$\left. \frac{du}{dx} \right|_{x_i} = \frac{u(x_i + h) - u(x_i - h)}{2h}$$

en el último nodo necesitamos introducir un punto de la malla ficticio $x_{n+1} = (x_n + \Delta x)$ con un valor asociado a u_{n+1} quedando

$$\frac{u_{n+1} - u_{n-1}}{2h} = cte_2. \tag{2.21}$$

Éste valor no tiene significado físico alguno, dado que este punto se encuentra fuera del dominio del problema.

Entonces debemos de:

1. Generar una malla homogénea del dominio

$$x_i = ih, \qquad i = 0, 1, ..., n, \qquad h = \frac{1}{n} = \Delta x. \tag{2.22}$$

2. Sustituir la derivada con la Ec.(1.24)[13] en cada punto donde la solución es desconocida para obtener un sistema algebraico de ecuaciones. Así, en cada punto x_i de la malla aproximamos la ecuación diferencial por

$$u''(x_i) \approx \frac{u(x_i - h) - 2u(x) + u(x_i + h)}{h^2} \tag{2.23}$$

definiendo la solución aproximada de $u(x)$ en x_i como u_i como la solución del siguiente sistema lineal de ecuaciones

$$\begin{aligned} \frac{u_\alpha - 2u_1 + u_2}{h^2} &= f(x_1) \\ \frac{u_1 - 2u_2 + u_3}{h^2} &= f(x_2) \\ &\vdots \\ \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} &= f(x_i) \\ &\vdots \\ \frac{u_{n-2} - 2u_{n-1} + u_n}{h^2} &= f(x_{n-1}) \\ \frac{u_{n+1} - u_{n-1}}{2h} &= cte_1. \end{aligned} \tag{2.24}$$

3. Resolver el sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ (véase capítulo A), obtenemos la solución aproximada en cada punto de la malla. La solución completa al problema la obtenemos al formar el vector

$$\left[\begin{array}{ccccccccc} u_\alpha & u_1 & u_2 & u_3 & \cdots & u_{n-2} & u_{n-1} & u_n \end{array} \right].$$

La implementación de un programa para codificar el Método de diferencias Finitas en una Dimensión para resolver el problema con condiciones Dirichlet o Neumann del problema

$$p(x)u'' + q(x)u' + r(x)u = f(x)$$

definido en el dominio $[xi, xf]$ y el tipo de frontera en xi —ti igual a -1 Dirichlet ($u(xi) = vi$) ó -2 Neumann ($u_x(xi) = vi$)— y el valor en la frontera xf —tf igual a -1 Dirichlet($u(xf) = vf$) o -2 Neumann ($u_x(xf) = vf$)—, además se le indica el tamaño de la partición n, si se graficará la solución indicando en $grf = 1$, con la solución analítica $s(x)$ y si ésta se proporciona $sws = 1$. Regresando la matriz y los vectores $\underline{\underline{A}}\underline{u} = \underline{b}$ del sistema lineal generado así como los puntos del dominio y los valores de la solución en dichos puntos x, V

[13] Para mantener la estabilidad es necesario tomar en cuenta las distintas formas de discretización en el Método de Diferencias Finitas mediante el esquema Upwind, Scharfetter-Gummel y Difusión Artificial, entre otros (véase sección: 2.1, Ecs. 2.8 a 2.11).

Ejemplo 8 *El programa queda implementado en OCTAVE (MatLab) como:*

```
function [error,A,b,u,x,V] = fdm1d(p,q,r,f,xi,xf,ti,vi,tf,vf,n,grf,s,sws)
    if n < 3
        return
    end
    % llenado de valores para el tipo de nodo y valor de frontera
    TN = zeros(n,1); % Vector para guardar el tipo de nodo
    V = zeros(n,1); % Vector para guardar los valores de la solucion y
frontera
    TN(1) = ti;
    TN(n) = tf;
    V(1) = vi;
    V(n) = vf;
    % Calcula el numero de incognitas del problema
    tm = 0;
    for i=1:n,
        if TN(i) == -1 % Descarta nodos frontera Dirichlet
            continue
        end
        tm = tm +1;
    end
    % Declaracion de la matriz y vectores de trabajo
    %A = sparse(tm,tm);
    A = zeros(tm,tm); % Matriz de carga
    b = zeros(tm,1); % Vector de carga
    u = zeros(tm,1); % Vector de solucion
    x = zeros(n,1); % Vector de coordenadas de la particion
    % Llenado de la matriz y vector
    h = (xf-xi)/(n-1);
    h1 = h*h;
    j = 1;
    for i=1:n,
        x(i) = xi + (i-1)*h;
        if TN(i) == -1 % Descarta nodos frontera Dirichlet
            continue;
        end
        % Diagonal central
        A(j,j) = ((-2.0 * p(x(i))) / h1) + r(x(i));
        if TN(i) == -2
            A(j,j) = -1/h1;
        end
        % Lado derecho
        b(j) = f(x(i));
        % Diagonal anterior
        if TN(i) == -2
            b(j) = V(i)/h;
```

```
        if i > 1
            A(j,j-1) = -1/h1;
        end
    elseif TN(i-1) == -1
        b(j) = f(x(i)) - p(x(i))*(V(i-1)/h1);
    else
        A(j,j-1) = p(x(i))/h1 - q(x(i))/(2.0*h);
    end
    % Diagonal superior
    if TN(i) == -2
        b(j) = V(i)/h;
        if i < n
            A(j,j+1) = -1/h1;
        end
    elseif TN(i+1) == -1
        b(j) = f(x(i)) - p(x(i))*(V(i+1)/h1);
    else
        A(j,j+1) = p(x(i))/h1 + q(x(i))/(2.0*h);
    end
    j = j + 1;
end
% Soluciona el sistema
u = A\b;
% Copia la solucion obtenida del sistema lineal al vector solucion
j = 1;
for i=1:n,
    if TN(i) == -1 % descarta nodos frontera Dirichlet
        continue
    end
    V(i) = u(j);
    j = j + 1;
end
% Encuentra el error en norma infinita usando una particion par = 10
error = 0;
if sws == 1
    par = 10;
    for i = 1: n-1,
        inc = (x(i+1)-x(i))/par;
        for j = 1:par+1,
            px = x(i)+inc*(j-1);
            e = abs(s(px)-l(px,x(i),V(i),x(i+1),V(i+1)));
            if e > error
                error = e;
            end
        end
    end
```

```
        end
        % Revisa si se graficara
        if grf == 1
            if sws == 1
                % Calcula la solucion analitica en la particion de calculo
                ua = zeros(n,1);
                for i = 1: n,
                    ua(i) = s(x(i));
                end
            end
            % Graficar la solucion numerica
            plot(x,V,'o');
            hold
            % Graficar la solucion analitica en una particion de tamano xPart
            if sws == 1
                xPart = 1000;
                h = (xf-xi)/(xPart-1);
                xx = zeros(xPart,1);
                xa = zeros(xPart,1);
                for i = 1: xPart,
                    xx(i) = xi + (i-1)*h;
                    xa(i) = s(xx(i));
                end
                plot(xx,xa);
                % Grafica el error
                figure(2);
                plot(x,V-ua);
            end
        end
    end
    % evalua el punto x en la recta dada por los puntos (x1,y1) y (x2,y2)
    function y = l(x,x1,y1,x2,y2)
        y = y1+((y2-y1)/(x2-x1))*(x-x1);
    end
```

Ejemplo 9 *Sea*

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

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:

```
v=@(x) 1.0; % velocidad, posibles valores 1,25,100, etc
p=@(x) -1.0;
q=@(x) v(x);
r=@(x) 0;
```

f=@(x) 0;
*s=@(x) (exp(v(x)*x)-1.0)/(exp(v(x))-1.0);*
[error,A,b,u,x,V] = fdm1d_ DN(p,q,r,f,0,1,-1,0,-1,1,11,1,s,1);

Ejemplo 10 *Sea*

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

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:
p=@(x) 1;
q=@(x) 0;
r=@(x) 0;
*f=@(x) -pi*pi*cos(pi*x);*
*s=@(x) cos(pi*x);*
[error,A,b,u,x,V] = fdm1d_ DN(p,q,r,f,0,1,-1,1,-1,-1,40,1,s,1);

Ejemplo 11 *Sea*

$$u''(x) = -\pi^2 cos(\pi x), \qquad 0 \leq x \leq 0.5, \qquad u(0) = 1, \qquad u_x(0.5) = -\pi$$

para ejecutar el programa anterior es necesario escribir en la consola de OCTAVE:
p=@(x) 1;
q=@(x) 0;
r=@(x) 0;
*f=@(x) -pi*pi*cos(pi*x);*
*s=@(x) cos(pi*x);*
[error,A,b,u,x,V] = fdm1d_ DN(p,q,r,f,0,0.5,-1,1,-2,-pi,40,1,s,1);

Pese a que con el programa anterior podríamos resolver la ecuación

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

esta ecuación se conoce como la ecuación de Helmholtz la cual es difícil de resolver si $r(x) \leq 0$ y $|r(x)|$ es grande, i.e. $r(x) \sim 1/h^2$. Para resolver correctamente dicha ecuación, usaremos otro programa con los esquemas de discretización de diferencias finitas y diferencias finitas exactas. Este último procedimiento fue desarrollado en: Exact Finite Difference Schemes for Solving Helmholtz equation at any wavenumber, Yau Shu Wong and Guangrui Lim International Journal of Numerical Analysis and Modeling, Volumen 2, Number 1, Pages 91-108, 2011. Y la codificación de prueba queda como:

Ejemplo 12 *Sea*

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

con $k = 150$, *entonces el programa en SCILAB queda implementado como:*

```
TEST = 1; // (0) Diferencias finitas, (1) Diferencias finitas exactas

function y=LadoDerecho(x)
 y=0.0;
endfunction

function y=SolucionAnalitica(x, k)
 //y=cos(k*x)+%i*sin(k*x);
 y=exp(%i*k*x);
endfunction

K = 150;
KK = K*K;
a=0; // Inicio dominio
c=1; // Fin dominio
M=300; // Particion
N=M-1; // Nodos interiores
h=(c-a)/(M-1); // Incremento en la malla
Y0=1; // Condicion Dirchlet inicial en el inicio del dominio
Y1=%i*K; // Condicion Neumann inicial en el fin del dominio

A=zeros(N,N); // Matriz A
b=zeros(N); // Vector b

if TEST = 0 then
R=-1/(h^2);
P=2/(h^2)-KK;
Q=-1/(h^2);
else
R=-1/(h^2);
P=(2*cos(K*h)+(K*h)^2)/(h^2) - KK;
Q=-1/(h^2);
end

// Primer renglon de la matriz A y vector b
A(1,1)=P;
A(1,2)=Q;
b(1)=LadoDerecho(a)-Y0*R; // Frontera Dirichlet
// Renglones intermedios de la matriz A y vector b
for i=2:N-1
 A(i,i-1)=R;
```

```
 A(i,i)=P;
 A(i,i+1)=Q;
 b(i)=LadoDerecho(a+h*(i-1));
end
// Renglon final de la matriz A y vector b
if TEST = 0 then
 A(N,N-1)=1/(h^2);
 A(N,N)=-1/(h^2)+ Y1/h;
 b(N)=LadoDerecho(c)/2;
else
 A(N,N-1)=1/(h^2);
 A(N,N)=-1/(h^2)+ %i*sin(K*h)/(h^2);
 b(N)=LadoDerecho(c)/2;
end

// Resuleve el sistema lineal Ax=b
x=inv(A)*b;

ESC = 5;
xxx=zeros(M*ESC,1);
zzz=zeros(M*ESC,1);
for i=1:M*ESC
 xxx(i)=a+h/ESC*(i-1);
 zzz(i)=SolucionAnalitica(xxx(i),K);
end
// Prepara la graficacion
xx=zeros(M,1);
zz=zeros(M,1);
for i=1:M
 xx(i)=a+h*(i-1);
 zz(i)=SolucionAnalitica(xx(i),K);
end
yy=zeros(M,1);
yy(1)=Y0; // Condicion inicial
for i=1:N
 yy(i+1)=x(i);
end
// Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
plot2d(xx,yy,15)
plot2d(xxx,zzz)
```

2.4 Discretización del Tiempo

Hasta ahora se ha visto como discretizar la parte espacial de las ecuaciones diferenciales parciales, lo cual nos permite encontrar la solución estática de los problemas del tipo elíptico. Sin embargo, para ecuaciones del tipo parabólico e

hiperbólico dependen del tiempo, se necesita introducir una discretización a las derivadas con respecto del tiempo. Al igual que con las discretizaciones espaciales, podemos utilizar algún esquema de diferencias finitas en la discretización del tiempo.

2.4.1 Ecuación con Primera Derivada Temporal

Para la solución de la ecuaciones diferenciales con derivada temporal (u_t), se emplean diferentes esquemas en diferencias finitas para la discretización del tiempo. Estos esquemas se conocen de manera general como esquemas $theta(\theta)$.

Definiendo la ecuación diferencial parcial general de segundo orden como

$$u_t = \mathcal{L}u \tag{2.25}$$

donde

$$\mathcal{L}u = \left(p\left(x\right)u'\left(x\right)\right)' + q\left(x\right)u'\left(x\right) - r\left(x\right)u\left(x\right) - f\left(x\right) \tag{2.26}$$

aquí, los coeficientes p, q y r pueden depender del espacio y del tiempo. Entonces el esquema $theta$ está dado por

$$u_t = \left(1-\theta\right)\left(\mathcal{L}u\right)^j + \theta\left(\mathcal{L}u\right)^{j+1}. \tag{2.27}$$

Existen diferentes casos del esquema $theta$ a saber:

- Para $\theta = 0$, se obtiene un esquema de diferencias finitas hacia adelante en el tiempo, conocido como esquema completamente explícito, ya que el paso $n+1$ se obtiene de los términos del paso anterior n. Es un esquema sencillo, el cual es condicionalmente estable cuando $\Delta t \leq \frac{\Delta x^2}{2}$.
- Para $\theta = 1$, se obtiene el esquema de diferencias finitas hacia atrás en el tiempo, conocido como esquema completamente implícito, el cual es incondicionalmente estable.
- Para $\theta = \frac{1}{2}$, se obtiene un esquema de diferencias finitas centradas en el tiempo, conocido como esquema Crank-Nicolson, este esquema es también incondicionalmente estable y es más usado por tal razón.

Para la implementación del esquema Crank-Nicolson se toma una diferencia progresiva para el tiempo y se promedian las diferencias progresivas y regresivas en el tiempo para las derivadas espaciales. Entonces, si tenemos la Ec. (2.26), las discretizaciones correspondientes son[14]:

$$u_t \simeq \frac{u_i^{j+1} - u_i^j}{\Delta t} \tag{2.28}$$

[14] Para mantener la estabilidad es necesario tomar en cuenta las distintas formas de discretización en el Método de Diferencias Finitas mediante el esquema Upwind, Scharfetter-Gummel y Difusión Artificial, entre otros (véase sección: 2.1, Ecs. 2.8 a 2.11).

$$(p(x)u'(x))' \simeq \frac{p}{2}\left[\frac{u_{i-1}^{j}-2u_i^j+u_{i+1}^j}{\Delta x^2}+\frac{u_{i-1}^{j+1}-2u_i^{j+1}+u_{i+1}^{j+1}}{\Delta x^2}\right] \tag{2.29}$$

$$q(x)u'(x) \simeq \frac{q}{2}\left[\frac{u_{i-1}^{j}+u_{i+1}^j}{2\Delta x}+\frac{u_{i-1}^{j+1}+u_{i+1}^{j+1}}{2\Delta x}\right] \tag{2.30}$$

además de $r(x), u_i^j$ y f_i^j.

Entonces, una vez que se sustituyen las derivadas por su forma en diferencias finitas, lo siguiente es formar el sistema:

$$Au^{j+1} = Bu^j + f^j \tag{2.31}$$

esto se logra, colocando del lado izquierdo la igualdad de los términos que contengan el paso del tiempo correspondiente a $j+1$ y del lado derecho a los correspondientes términos de j.

A continuación, veamos un ejemplo del esquema Crank-Nicolson desarrollados en SCILAB[15]

Ejemplo 13 *Sea*

$$u_t - a(x)u''(x) - b(x)u'(x) + c(x)u = f, \qquad l0 \le x \le l, \qquad 0 < t < T$$

entonces el programa queda implementado como:

```
// Crank-Nicolson
// Para una EDP de segundo orden
// u_t + a(x)u_xx + b(x)u_x +c(x)u = f
// Dominio
// l0<x<l
// 0<t<T
// Condiciones de frontera Dirichlet
// u(0,t) = u(l,t) = constante 0 < t < T cond de frontera
// Condicion inicial
// u(x,0) = g(x) l0 <= x <= l
// Datos de entrada
// intrevalo [l0, l]
// entero m>=3
// entero N>=1
// Salida
// aproximaciones w_ij a u(x_i,t_j)
// Funciones de los coeficientes
function y = a(x)
y = -1;
```

[15] Scilab es un programa open source para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería [http://www.scilab.org].

```
endfunction
function y = b(x)
y = -1;
endfunction
function y = c(x)
y = 1;
endfunction
function y = f(x)
y = 0;
endfunction
// funcion de la condicion inicial
function y = condicion_inicial(x)
y = sin(x * %pi);
endfunction
// Condiciones de frontera
// Solo Dirichlet
cond_izq = 0;
cond_der = 0;
// Datos de entrada
l0 = 0;l = 1; // intervalo [0,1]
m = 11; // Numero de nodos
M = m - 2; // Nodos interiores
// Division del espacio y del tiempo
h = (l - l0)/(m-1);
k = 0.025; // Paso del tiempo
N = 50; // Numero de iteraciones
// Aw^(j+1) = Bw^j + f^j
// creo el vector w donde se guardara la solucion para cada tiempo
// A matriz del lado izquierdo
// B matriz del lado derecho
// B_prima para guardar el resultado de Bw^j
// ff que es el vector de los valores de f en cada nodo
w = zeros(M,1);
ff = zeros(M,1)
A = zeros(M,M);
B = zeros(M,M);
//B_prima = zeros(M,1);
w_sol = zeros(m,m)
// Se crea el espacio de la solucion o malla
espacio = zeros(M,1)
for i = 1:m
xx = l0 + (i-1)*h;
espacio(i) = xx;
end
disp(espacio, "Espacio")
// Condicion inicial
```

```
for i=1:M
w(i) = condicion_inicial(espacio(i+1));
end
w_sol(1) = cond_izq;
for kk = 1:M
w_sol(kk + 1) = w(kk);
end
w_sol(m) = cond_izq;
plot(espacio, w_sol);
disp(w, "Condiciones iniciales")
// primer renglon de cada matriz
A(1,1) = 1/k - a(l0 + h)/(h*h);
A(1,2) = a(l0 + h)/(2*h*h) + b(l0 + h)/(4*h) ;
B(1,1) = 1/k + a(l0 + h)/(h*h) - c(l0 + h);
B(1,2) = - a(l0 + h)/(2*h*h) - b(l0 + h)/(4*h);
ff(1) = f(l0 + h) - cond_izq;
// se completa las matrices desde el renglon 2 hasta el m-2
for i = 2:M-1
A(i, i-1) = a(l0 + i*h)/(2*h*h) - b(l0 + i*h)/(4*h) ;
A(i,i) = 1/k - a(l0 + i*h)/(h*h);
A(i,i+1) = a(l0 + i*h)/(2*h*h) + b(l0 + i*h)/(4*h) ;
B(i, i-1) = - a(l0 + i*h)/(2*h*h) + b(l0 + i*h)/(4*h);
B(i,i) = 1/k + a(l0 + i*h)/(h*h) - c(l0 + i*h);
B(i,i+1) = - a(l0 + i*h)/(2*h*h) - b(l0 + i*h)/(4*h);
end
// Ultimo renglon de cada matriz
A(M,M-1) = a(l - h)/(2*h*h) - b(l-h)/(4*h) ;
A(M,M) = 1/k - a(l - h)/(h*h);
B(M,M-1) = - a(l-h)/(2*h*h) + b(l-h)/(4*h);
B(M,M) = 1/k + a(l-h)/(h*h) - c(l-h);
ff(M) = f(l - h) - cond_der;
// Resolvemos el sistema iterativamente
for j=1:21
t = j*k;
B_prima = B * w + ff;
w = inv(A) * B_prima;
disp(t, "tiempo")
disp(w, "Sol")
w_sol(1) = cond_izq;
for kk = 1:M
  w_sol(kk + 1) = w(kk);
end
w_sol(m) = cond_izq;
plot(espacio, w_sol);
end
```

2.4.2 Ecuación con Segunda Derivada Temporal

Para el caso de ecuaciones con segunda derivada temporal, esta se aproxima por diferencias finitas centradas en el tiempo, partiendo de la Ec. (2.26), las discretizaciones correspondientes son[16]

$$u_{tt} \simeq \frac{u_i^{j-1} - 2u_i^j + u_i^{j+1}}{\Delta t^2} \tag{2.32}$$

$$\left(p\left(x\right)u'\left(x\right)\right)' \simeq p\left[\frac{u_{i-1}^j - 2u_i^j + u_{i+1}^j}{\Delta x^2}\right] \tag{2.33}$$

$$q\left(x\right)u'\left(x\right) \simeq q\left[\frac{u_{i-1}^j + u_{i+1}^j}{2\Delta x}\right] \tag{2.34}$$

además de $r(x)$, u_i^j y f_i^j.

Entonces, una vez que se sustituyen las derivadas por su forma en diferencias finitas, lo siguiente es formar el sistema

$$u_i^{j+1} = 2u_i^j - u_i^{j-1} + \left(\Delta t\right)^2 Bu^j \tag{2.35}$$

esto se logra, colocando del lado izquierdo la igualdad de los términos que contengan el paso del tiempo correspondiente a $j+1$ y del lado derecho a los correspondientes términos de j y $j-1$. Para calcular u_i^{j+1} es necesario conocer u_{i-1}, u_i, u_{i+1} en los dos instantes inmediatos anteriores, i.e. t_j y t_{j-1}.

En particular para calcular u_i^1 es necesario conocer u_i^0 y u_i^{-1}, si consideramos

$$u_i^{j+1} = 2u_i^j - u_i^{j-1} + \left(\Delta t\right)^2 \mathcal{L}u^j \tag{2.36}$$

para $j = 0$, entonces

$$u_i^1 = 2u_i^0 - u_i^{-1} + \left(\Delta t\right)^2 \mathcal{L}u^0 \tag{2.37}$$

donde u_i^0 es la condición inicial y $\frac{u_i^1 - u_i^{-1}}{2\Delta t}$ es la primer derivada de la condición inicial. Así, para el primer tiempo tenemos

$$u_i^1 = u(x_i, 0) + \Delta t u'\left(x_i, 0\right) + \left(\Delta t\right)^2 \mathcal{L}u^0 \tag{2.38}$$

lo cual permite calcular u_i^1 a partir de las condiciones iniciales.

La derivación del método parte de

$$\begin{aligned} u_{tt} &= \mathcal{L}u^j \\ \frac{u_i^{j-1} - 2u_i^j + u_i^{j+1}}{\left(\Delta t\right)^2} &= \mathcal{L}u^j \\ u_i^{j+1} &= 2u_i^j - u_i^{j-1} + \left(\Delta t\right)^2 \mathcal{L}u^j \end{aligned} \tag{2.39}$$

[16] Para mantener la estabilidad es necesario tomar en cuenta las distintas formas de discretización en el Método de Diferencias Finitas mediante el esquema Upwind, Scharfetter-Gummel y Difusión Artificial, entre otros (véase sección: 2.1, Ecs. 2.8 a 2.11).

donde el error intrínseco a la discretización es de orden cuadrático, pues se ha usado diferencias centradas, tanto para el espacio como para el tiempo.

Ejemplo 14 *Sea*

$$u_{tt} - 4u''(x) = 0, \qquad 0 \leq x \leq l, \qquad 0 < t < T$$

sujeta a

$$u(0,t) = u(1,t) = 0, \qquad u(x,0) = \sin(\pi x), \qquad u_t(x,0) = 0$$

con solución analítica

$$u(x,t) = sen(\pi x) * cos(2\pi t)$$

entonces el programa queda implementado como:

```
// Dominio
a_ = 0
b_ = 1
// Particion en x
m = 101; // numero de nodos
h = (b_ - a_)/(m-1)
dt = 0.001 // salto del tiempo
// Para que sea estable se debe cumplir que
// sqrt(a) <= h/dt
// Coeficiente
function y = a(x)
y = -4;
endfunction
// Condicion inicial
function y = inicial(x)
y = sin(%pi * x)
endfunction
function y = u_t(x)
y = 0;
endfunction
// Solucion analitica
function y = analitica(x,t)
y = sin(%pi * x) * cos(2* %pi * t)
endfunction
////////////////////////////////////
// Aw^(j+1) = Bw^j
// creo el vector w donde se guradaria la solucion para cada tiempo
// A matriz del lado izquierdo
// B matriz del lado derecho
// B_prima para guardar el resultado de Bw^j
w_sol = zeros(m,1);
```

```
w_sol_temp = zeros(m,1);
w_temp = zeros(m,1);
w = zeros(m,1);
w_ = zeros(m,1);
A = eye(m,m);
B = zeros(m,m);
B_prima = zeros(m,1);
espacio = zeros(m,1)
sol = zeros(m,1);
// primer renglon de cada matriz
B(1,1) = 2*a(a_)*dt*dt/(h*h)
B(1,2) = -a(a_)*dt*dt/(h*h)
// se completa las matrices desde el renglon 2 hasta el m-1
for i = 2:m-1
B(i, i-1) = -a(i*h)*dt*dt/(h*h)
B(i,i) = 2*a(i*h)*dt*dt/(h*h)
B(i,i+1) = -a(i*h)*dt*dt/(h*h)
end
// Ultimo renglon de cada matriz
B(m,m-1) = -a(b_)*dt*dt/(h*h)
B(m,m) = 2*a(b_)*dt*dt/(h*h)
// muestro la matriz
//printf("Matriz B\n");
//disp(B);
for i=1:m
xx = (i-1)*h;
espacio(i) = a_ + xx;
w(i) = inicial(espacio(i)); // Condiciones iniciales
w_(i) = inicial(espacio(i)) + u_t(espacio(i)) * dt
end
//
//disp(espacio)
//disp(w)
////////////
//Para t = 0
B_prima = B * w;
for i = 1:m
w_sol(i) = w_(i) + B_prima(i);
end
////////////
printf("w para t = 0\n");
disp(w_sol);
for i = 1:m
sol(i) = analitica(espacio(i), 0)
end
printf("Solucion analitica para t = 0\n")
```

```
disp(sol)
plot(espacio,w_sol)
//plot(espacio,sol,'r')
w_sol_temp = w_sol;
w_temp = w
for i=1:500
t = i*dt
B_prima = B * w_sol_temp;
w_ = 2 * w_sol_temp - w_temp
w_sol = w_ + B_prima;
////
// for j = 1:m
// sol(j) = analitica(espacio(j), t)
// end
//
// printf("Sol analitica dt = %f", t)
// disp(sol)
// printf("Sol metodo dt = %f", t)
// disp(w_sol)
w_temp = w_sol_temp
w_sol_temp = w_sol
if i == 5 | i == 50 | i == 100 | i == 150 | i == 200 | i == 250 | i ==
300 | i== 350 | i == 400 | i == 450 | i == 500 then
plot(espacio,w_sol)
end
//plot(espacio,sol,'r')
end
```

2.5 Consistencia, Estabilidad, Convergencia y Error del Método de Diferencias Finitas

Cuando se usa algún método para la resolución de ecuaciones diferenciales, se necesita conocer cuan exacta es la aproximación obtenida en comparación con la solución analítica (en caso de existir).

Error Global Sea $U = [U_1, U_2, ..., U_n]^T$ el vector solución obtenido al utilizar el método de diferencias finitas y $u = [u(x_1), u(x_n), ..., u(x_n)]$ la solución exacta de los puntos de la malla. El vector de error global se define como $E = U - u$. lo que se desea es que el valor máximo sea pequeño. Usualmente se utilizan distintas normas para encontrar el error.

- La norma infinito, definida como $\|E\|_\infty = \max |e_i|$
- La norma-1, definida como $\|E\|_1 = \sum_{i=1}^{n} |e_i|$

- La norma-2, definida como $\|E\|_2 = \left(\sum_{i=1}^{n} e_i^2 \right)^{\frac{1}{2}}$

La norma infinito $\|E\|_\infty$ es en general, la más apropiada para calcular los errores relativos a la discretización.

Definición 1 *Si* $\|E\| \leq Ch^p, p > 0$, *decimos que el método de diferencias finitas es de orden-p de precisión.*

Definición 2 *Un método de diferencias finitas es llamado convergente si*

$$\lim_{h \to 0} \|E\| = 0. \tag{2.40}$$

Error de Truncamiento Local Sea el operador diferencia $\mathcal{L}u$, definimos el operador en diferencias finitas $\mathcal{L}_h$, por ejemplo para la ecuación de segundo orden $u''(x) = f(x)$, uno de los operadores de diferencias finitas puede ser

$$\mathcal{L}_h u(x) = \frac{u(x-h) - 2u(x) + u(x+h)}{h^2}. \tag{2.41}$$

Definición 3 *El error de truncamiento local es definido como*

$$T(x) = \mathcal{L}u - \mathcal{L}_h u. \tag{2.42}$$

Para la ecuación diferencial $u''(x) = f(x)$ y el esquema de diferencias centradas usando tres puntos $\frac{u(x-h)-2u(x)+u(x+h)}{h^2}$, el error de truncamiento local es

$$\begin{aligned} T(x) &= \mathcal{L}u - \mathcal{L}_h u = u''(x) - \frac{u(x-h) - 2u(x) + u(x+h)}{h^2} \\ &= f(x) - \frac{u(x-h) - 2u(x) + u(x+h)}{h^2}. \end{aligned} \tag{2.43}$$

Note que el error de truncamiento local sólo depende de la solución del esténcil en diferencias finitas (en el ejemplo es usando tres puntos) pero no depende de la solución global, es por ello que se denomina error de truncamiento local. Este es una medida de que tanto la discretización en diferencias finitas se aproxima a la ecuación diferencial.

Definición 4 *Un esquema de diferencias finitas es llamado consistente si*

$$\lim_{h \to 0} T(x) = \lim_{h \to 0} (\mathcal{L}u - \mathcal{L}_h u) = 0. \tag{2.44}$$

Si $T(x) = Ch^p, p > 0$, entonces se dice que la discretización es de orden$-p$ de precisión, donde C es una constante independiente de h pero puede depender de la solución de $u(x)$. Para conocer cuando un esquema de diferencias finitas es consistente o no, se usa la expansión de Taylor. Por ejemplo, para el esquema

de diferencias finitas centradas usando tres puntos para la ecuación diferencial $u''(x) = f(x)$, tenemos que

$$T(x) = u''(x) - \frac{u(x-h) - 2u(x) + u(x+h)}{h^2} = -\frac{h^2}{12}u^{(4)}(x) + ... = Ch^2 \quad (2.45)$$

donde $C = \frac{1}{12}u^{(4)}(x)$. Por lo tanto, este esquema de diferencias finitas es consistente y la discretización es de segundo orden de precisión.

La consistencia no puede garantizar que un esquema de diferencias finitas trabaje. Para ello, necesitamos determinar otra condición para ver si converge o no. Tal condición es llamada la estabilidad de un método de diferencias finitas. Para el problema modelo, tenemos que

$$Au = F + T, \qquad AU = F, \qquad A(u - U) = T = -AE \quad (2.46)$$

donde A son los coeficientes de la matriz de las ecuaciones en diferencias finitas, F son los términos modificados por la condición de frontera, y T es el vector local de truncamiento en los puntos de la malla.

Si la matriz A es no singular, entonces $\|E\| = \left\|A^{-1}T\right\| \leq \left\|A^{-1}\right\| \|T\|$. Para el esquema de diferencias finitas centradas, tenemos que $\|E\| = \left\|A^{-1}\right\| h^2$. Tal que el error global depende del error de truncamiento local y $\left\|A^{-1}\right\|$.

Definición 5 *Un método de diferencias finitas para la ecuación diferencial elíptica es estable si A es invertible y*

$$\left\|A^{-1}\right\| \leq C, \text{ para todo } h \leq h_0 \quad (2.47)$$

donde C y h_0 son constantes.

Teorema 6 *Si el método de diferencias finitas es estable y consistente, entonces es convergente.*

3 Método de Diferencias Finitas en Dos y Tres Dimensiones

Consideremos la ecuación diferencial parcial

$$\left(p\left(x\right)u'\left(x\right)\right)' + q\left(x\right)u'\left(x\right) - r\left(x\right)u\left(x\right) = f\left(x\right) \tag{3.1}$$

$$\text{en } a \leq x \leq b \qquad \text{donde: } u\left(a\right) = u_\alpha \text{ y } u\left(b\right) = u_\beta$$

con condiciones de frontera Dirichlet o cualquier otro tipo de condiciones de frontera. Para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, debemos de:

1. Generar una malla del dominio, i.e. una malla es un conjunto finito de puntos en los cuales buscaremos la solución aproximada a la ecuación diferencial parcial.
2. Sustituir las derivadas correspondientes con alguna de las formulas de diferencias finitas centradas (véase secciones 1.1 y 1.2), en cada punto donde la solución es desconocida para obtener un sistema algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$.
3. Resolver el sistema de ecuaciones (véase capítulo A), y así obtener la solución aproximada en cada punto de la malla.

3.1 Derivadas en Dos y Tres Dimensiones

De forma análoga se construyen aproximaciones en diferencias finitas de primer y segundo orden en dos y tres dimensiones.

3.1.1 Derivadas en Dos Dimensiones

Usando el teorema de Taylor para funciones en dos variables x y y, es posible escribir de forma exacta para el punto x_i y y_j

$$\begin{aligned} f(x_i + \Delta x, y_j) &= f(x_i, y_j) + \Delta x \frac{\partial f\left(x_i, y_j\right)}{\partial x} + \frac{\Delta x}{2}\frac{\partial^2 f\left(x_i + \theta_1 \Delta x, y_j\right)}{\partial x^2} \\ f(x_i, y_j + \Delta y) &= f(x_i, y_j) + \Delta y \frac{\partial f\left(x_i, y_j\right)}{\partial y} + \frac{\Delta y}{2}\frac{\partial^2 f\left(x_i, y_j + \theta_2 \Delta y\right)}{\partial y^2}. \end{aligned} \tag{3.2}$$

Así, la aproximación en diferencias hacia adelante de $\partial f/\partial x$ y $\partial f/\partial y$ es

$$\begin{aligned} \frac{\partial f\left(x_i, y_j\right)}{\partial x} &\simeq \frac{f\left(x_i + \Delta x, y_j\right) - f\left(x_i, y_j\right)}{\Delta x} \\ \frac{\partial f\left(x_i, y_j\right)}{\partial y} &\simeq \frac{f\left(x_i, y_j + \Delta y\right) - f\left(x_i, y_j\right)}{\Delta y} \end{aligned} \tag{3.3}$$

o en su forma simplificada (asociamos $\Delta x = h$ y $\Delta y = k$), entonces tenemos

$$\begin{aligned} \frac{\partial f(x_i, y_j)}{\partial x} &\simeq \frac{f_{i+1,j} - f_{i,j}}{h} \\ \frac{\partial f(x_i, y_j)}{\partial y} &\simeq \frac{f_{i,j+1} - f_{i,j}}{k}. \end{aligned} \tag{3.4}$$

La aproximación en diferencias hacia atrás de $\partial f/\partial x$ y $\partial f/\partial y$ es

$$\begin{aligned} \frac{\partial f(x_i, y_j)}{\partial x} &\simeq \frac{f(x_i, y_j) - f(x_i - \Delta x, y_j)}{\Delta x} \\ \frac{\partial f(x_i, y_j)}{\partial y} &\simeq \frac{f(x_i, y_j) - f(x_i, y_j - \Delta y)}{\Delta y} \end{aligned} \tag{3.5}$$

o en su forma simplificada tenemos

$$\begin{aligned} \frac{\partial f(x_i, y_j)}{\partial x} &\simeq \frac{f_{i,j} - f_{i-1,j}}{h} \\ \frac{\partial f(x_i, y_j)}{\partial y} &\simeq \frac{f_{i,j} - f_{i,j-1}}{k}. \end{aligned} \tag{3.6}$$

La aproximación en diferencias centradas de $\partial f/\partial x$ y $\partial f/\partial y$ es

$$\begin{aligned} \frac{\partial f(x_i, y_j)}{\partial x} &\simeq \frac{f(x_i + \Delta x, y_j) - f(x_i - \Delta x, y_j)}{2\Delta x} \\ \frac{\partial f(x_i, y_j)}{\partial y} &\simeq \frac{f(x_i, y_j + \Delta y) - f(x_i, y_j - \Delta y)}{2\Delta y} \end{aligned} \tag{3.7}$$

o en su forma simplificada tenemos

$$\begin{aligned} \frac{\partial f(x_i, y_j)}{\partial x} &\simeq \frac{f_{i+1,j} - f_{i-1,j}}{2h} \\ \frac{\partial f(x_i, y_j)}{\partial y} &\simeq \frac{f_{i,j+1} - f_{i,j-1}}{2k}. \end{aligned} \tag{3.8}$$

Por otro lado, la aproximación en diferencias centradas de $\partial^2 f/\partial x^2$ y $\partial^2 f/\partial y^2$ es

$$\begin{aligned} \frac{\partial^2 f(x_i, y_j)}{\partial x^2} &\simeq \frac{f(x_i - \Delta x, y_j) - 2f(x_i, y_j) + f(x_i + \Delta x, y_j)}{\Delta x^2} \\ \frac{\partial^2 f(x_i, y_j)}{\partial y^2} &\simeq \frac{f(x_i, y_j - \Delta y) - f(x_i, y_j) + f(x_i, y_j + \Delta y)}{\Delta y^2} \end{aligned} \tag{3.9}$$

o en su forma simplificada tenemos

$$\begin{aligned} \frac{\partial^2 f(x_i, y_j)}{\partial x^2} &\simeq \frac{f_{i-1,j} - 2f_{i,j} + f_{i+1,j}}{h^2} \\ \frac{\partial^2 f(x_i, y_j)}{\partial y^2} &\simeq \frac{f_{i,j-1} - 2f_{i,j} + f_{i,j+1}}{k^2}. \end{aligned} \tag{3.10}$$

3.1.2 Derivadas en Tres Dimensiones

Usando el teorema de Taylor para funciones de tres variables x, y y z, es posible escribir de forma exacta para el punto x_i, y_j y z_k

$$\begin{aligned} f(x_i+\Delta x, y_j, z_k) &= f(x_i, y_j, z_k) + \Delta x \frac{\partial f(x_i, y_j, z_k)}{\partial x} + \frac{\Delta x}{2} \frac{\partial^2 f(x_i+\theta_1 \Delta x, y_j, z_k)}{\partial x^2} \\ f(x_i, y_j+\Delta y, z_k) &= f(x_i, y_j, z_k) + \Delta y \frac{\partial f(x_i, y_j, z_k)}{\partial y} + \frac{\Delta y}{2} \frac{\partial^2 f(x_i, y_j+\theta_2 \Delta y, z_k)}{\partial y^2} \\ f(x_i, y_j, z_k+\Delta z) &= f(x_i, y_j, z_k) + \Delta z \frac{\partial f(x_i, y_j, z_k)}{\partial z} + \frac{\Delta z}{2} \frac{\partial^2 f(x_i, y_j, z_k+\theta_3 \Delta z)}{\partial z^2} \end{aligned} \tag{3.11}$$

Así, la aproximación en diferencias hacia adelante de $\partial f/\partial x, \partial f/\partial y$ y $\partial f/\partial z$ es

$$\begin{aligned} \frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f(x_i+\Delta x, y_j, z_k) - f(x_i, y_j, z_k)}{\Delta x} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f(x_i, y_j+\Delta y, z_k) - f(x_i, y_j, z_k)}{\Delta y} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f(x_i, y_j, z_k+\Delta z) - f(x_i, y_j, z_k)}{\Delta z} \end{aligned} \tag{3.12}$$

o en su forma simplificada (para simplificar la notación, asociamos $\Delta x = h, \Delta y = l$ y $\Delta z = m$), tenemos

$$\begin{aligned} \frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f_{i+1,j,k} - f_{i,j,k}}{h} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f_{i,j+1,k} - f_{i,j,k}}{l} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f_{i,j,k+1} - f_{i,j,k}}{m}. \end{aligned} \tag{3.13}$$

La aproximación en diferencias hacia atrás de $\partial f/\partial x, \partial f/\partial y$ y $\partial f/\partial z$ es

$$\begin{aligned} \frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f(x_i, y_j, z_k) - f(x_i-\Delta x, y_j, z_k)}{\Delta x} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f(x_i, y_j, z_k) - f(x_i, y_j-\Delta y, z_k)}{\Delta y} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f(x_i, y_j, z_k) - f(x_i, y_j, z_k-\Delta z)}{\Delta z} \end{aligned} \tag{3.14}$$

o en su forma simplificada tenemos

$$\begin{aligned} \frac{\partial f(x_i, y_j, z_k)}{\partial x} &\simeq \frac{f_{i,j,k} - f_{i-1,j,k}}{h} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial y} &\simeq \frac{f_{i,j,k} - f_{i,j-1,k}}{l} \\ \frac{\partial f(x_i, y_j, z_k)}{\partial z} &\simeq \frac{f_{i,j,k} - f_{i,j,k-1}}{m}. \end{aligned} \tag{3.15}$$

La aproximación en diferencias centradas de $\partial f/\partial x, \partial f/\partial y$ y $\partial f/\partial z$ es

$$\begin{aligned} \frac{\partial f(x_i,y_j,z_k)}{\partial x} &\simeq \frac{f(x_i+\Delta x,y_j,z_k)-f(x_i-\Delta x,y_j,z_k)}{2\Delta x} \\ \frac{\partial f(x_i,y_j,z_k)}{\partial y} &\simeq \frac{f(x_i,y_j+\Delta y,z_k)-f(x_i,y_j-\Delta y,z_k)}{2\Delta y} \\ \frac{\partial f(x_i,y_j,z_k)}{\partial z} &\simeq \frac{f(x_i,y_j,z_k+\Delta z)-f(x_i,y_j,z_k-\Delta z)}{2\Delta z} \end{aligned} \tag{3.16}$$

o en su forma simplificada tenemos

$$\begin{aligned} \frac{\partial f(x_i,y_j,z_k)}{\partial x} &\simeq \frac{f_{i+1,j,k}-f_{i-1,j,k}}{2h} \\ \frac{\partial f(x_i,y_j,z_k)}{\partial y} &\simeq \frac{f_{i,j+1,k}-f_{i,j-1,k}}{2l} \\ \frac{\partial f(x_i,y_j,z_k)}{\partial z} &\simeq \frac{f_{i,j,k+1}-f_{i,j,k-1}}{2m}. \end{aligned} \tag{3.17}$$

Por otro lado, la aproximación en diferencias centradas de $\partial^2 f/\partial x^2, \partial^2 f/\partial y^2$ y $\partial^2 f/\partial z^2$ es

$$\begin{aligned} \frac{\partial^2 f(x_i,y_j,z_k)}{\partial x^2} &\simeq \frac{f(x_i-\Delta x,y_j,z_k)-2f(x_i,y_j,z_k)+f(x_i+\Delta x,y_j,z_k)}{\Delta x^2} \\ \frac{\partial^2 f(x_i,y_j,z_k)}{\partial y^2} &\simeq \frac{f(x_i,y_j-\Delta y,z_k)-f(x_i,y_j,z_k)+f(x_i,y_j+\Delta y,z_k)}{\Delta y^2} \\ \frac{\partial^2 f(x_i,y_j,z_k)}{\partial z^2} &\simeq \frac{f(x_i,y_j,z_k-\Delta z)-f(x_i,y_j,z_k)+f(x_i,y_j,z_k+\Delta z)}{\Delta z^2} \end{aligned} \tag{3.18}$$

o en su forma simplificada tenemos

$$\begin{aligned} \frac{\partial^2 f(x_i,y_j,z_k)}{\partial x^2} &\simeq \frac{f_{i-1,j,k}-2f_{i,j,k}+f_{i+1,j,k}}{h^2} \\ \frac{\partial^2 f(x_i,y_j,z_k)}{\partial y^2} &\simeq \frac{f_{i,j-1,k}-2f_{i,j,k}+f_{i,j+1,k}}{l^2} \\ \frac{\partial^2 f(x_i,y_j,z_k)}{\partial z^2} &\simeq \frac{f_{i,j,k-1}-2f_{i,j,k}+f_{i,j,k+1}}{m^2}. \end{aligned} \tag{3.19}$$

4 Paralelización del Método de Diferencias Finitas

La solución numérica de ecuaciones diferenciales parciales por los esquemas tradicionales —tipo Diferencias Finitas, Volumen Finito y Elemento Finito— reducen el problema a la generación y solución de un —cada vez más grande— sistema algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{b}$(véase [3]). La factorización directa de sistemas de gran escala $O(10^6)$ con toda su eficacia, no es, en general una opción viable, y el uso de métodos iterativos básicos —tales como el método de gradiente conjugado o residual mínimo generalizado— resultan en una convergencia bastante lenta —al ser algoritmos secuenciales— con respecto a otras formas de discretización como son los métodos de descomposición de dominio (véase [6] y [8]).

El desarrollo de métodos numéricos para sistemas algebraicos grandes, es central en el desarrollo de códigos eficientes para la solución de problemas en Ciencia e Ingeniería, actualmente cuando se necesita implementar una solución computacional, se dispone de bibliotecas optimizadas[17] para la solución de sistemas lineales que pueden correr en ambientes secuenciales y/o paralelos. Estas bibliotecas implementan métodos algebraicos robustos para muchos problemas prácticos, pero sus discretizaciones no pueden ser construidas por sólo técnicas algebraicas simples, tales como aproximaciones a la inversa o factorización incompleta.

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

Para hacer eficiente la solución de sistemas de ecuaciones diferenciales parciales, se introdujeron los métodos de descomposición de dominio que toman en cuenta la ecuación diferencial parcial y su discretización, permitiendo una alta eficiencia computacional en diversas arquitecturas paralelas (véase [6] y [8]). La idea básica detrás de los métodos de descomposición de dominio es que en lugar de resolver un enorme problema sobre un dominio, puede ser conveniente —o necesario— resolver múltiples problemas de tamaño menor sobre un solo subdominio un cierto número de veces. Mucho del trabajo en la descomposición de dominio se relaciona con la selección de subproblemas que aseguren que la razón

[17] Como pueden ser las bibliotecas ATLAS —http://math-atlas.sourceforge.net— y HYPRE —http://acts.nersc.gov/hypre/— entre muchas otras.

de convergencia del nuevo método iterativo sea rápida. En otras palabras, los métodos de descomposición de dominio proveen precondicionadores a priori que puedan acelerarse por métodos en el espacio de Krylov (véase [19]).

La descomposición de dominio generalmente se refiere a la separación de una ecuación diferencial parcial o una aproximación de ella dentro de problemas acoplados sobre subdominios pequeños formando una partición del dominio ori-ginal. Esta descomposición puede hacerse a nivel continuo, donde diferentes modelos físicos, pueden ser usados en diferentes regiones, o a nivel discreto, donde puede ser conveniente el empleo de diferentes métodos de aproximación en diferentes regiones, o en la solución del sistema algebraico asociado a la aproximación de la ecuación diferencial parcial —estos tres aspectos están íntimamente interconectados en la práctica (véase [19])—.

Los método de descomposición de dominio —Domain Decomposition Methods (DDM)— se basan en la suposición de que dado un dominio $\Omega \subset \mathbb{R}^n$, se puede particionar en E subdominios $\Omega_i, i = 1, 2, ..., E$; tales que $\Omega = \left(\bigcup_{i=1}^{E} \Omega_i \right)$, entre los cuales puede o no existir traslape (véase [3] y [19]). Entonces, el problema es reformulado en términos de cada subdominio —mediante el uso de algún método de discretización— obteniendo una familia de subproblemas de tamaño reducido independientes entre sí, y que están acoplados a través de la solución en la interfase —que es desconocida— de los subdominios.

De esta manera, se puede clasificar de forma burda a los métodos de descomposición de dominio (véase [6]), 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 a los nodos de la interfase o frontera interior—.

Desde hace ya algún tiempo, la comunidad internacional[18] inicio el estudio intensivo de los métodos de descomposición de dominio, la atención se ha desplazado (véase [9]) de los métodos con traslape en el dominio (véase [50]) a los métodos sin traslape en el dominio (véase [43], [44], [45], [46] y [47]), ya que estos últimos son más efectivos para una gran variedad de problemas de la Ciencia y la Ingeniería.

Los métodos de descomposición de dominio sin traslape son un paradigma natural usado por la comunidad de modeladores (véase [1]). Los sistemas físicos son descompuestos en dos o más subdominios contiguos basados en consideraciones fenomenológicas o computacionales. Esta descomposición se refleja en la Ingeniería de Software del código correspondiente, además, el uso de la programación orientada a objetos, permite dividir en niveles la semántica de los

[18] Ello se refleja en las más de 19 conferencias internacionales de Métodos Descomposición de Dominio (véase [18]) y de las cuales se han publicado 14 libros que recopilan los trabajos más relevantes de cada conferencia. Además de varios mini simposios de descomposición de dominio en congresos mundiales como es el World Congress on Computational Mechanics o el Iberian Latin American Congress on Computational Methods in Engineering.

sistemas complejos, tratando así con las partes, más manejables que el todo, permitiendo una implementación, extensión y mantenimiento sencillo (véase [17]).

Tomando en cuenta que los métodos de descomposición de dominio sin traslape son fácilmente implementados para su uso en computadoras paralelas mediante técnicas de programación orientada a objetos —porqué el algoritmo del método es paralelo–, además, con los continuos avances en cómputo, en particular, en la computación en paralelo mediante equipos de cómputo de alto desempeño y/o Clusters parecen ser el mecanismo más efectivo para incrementar la capacidad y velocidad de resolución de varios tipos de problemas de interés en Ciencias e Ingenierías usando métodos de descomposición de dominio (véase [44], [45], [46], [48] y [49]).

La implementación de los métodos de descomposición de dominio permite utilizar de forma eficiente, las crecientes capacidades del cómputo en paralelo (véase [18] y [19]) —Grids[19] de decenas de Clusters, cada uno con cientos o 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 se necesitan resolver, los cuales superan la capacidad de cómputo actual, ya sea por el tiempo requerido para su solución, por el consumo excesivo de memoria o ambos.

Así, los métodos de descomposición de dominio que introducen desde la formulación matemática del problema una separación natural de las tareas a realizar y simplifican considerablemente la transmisión de información entre los subdominios (véase [19]), en conjunción con la programación orientada a objetos y el cómputo en paralelo forman una amalgama poderosa. La cual permite construir aplicaciones que coadyuven en la solución una gran gama de problemas concomitantes en Ciencias e Ingenierías que requieren hacer uso de una gran cantidad de grados de libertad.

Por otro lado, 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 (véase [18] y [19]), 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.

[19] Bajo el Macroproyecto: Tecnologías para la Universidad de la Información y la Computación de la UNAM, se interconectaron cuatro Cluster heterogéneos —dos en la Facultad de Ciencias, uno en el Instituto de Geofísica y otro en el Instituto de Matemáticas Aplicadas y Sistemas— con redes no dedicadas y en ellos se probo una versión de los códigos, en los cuales se vio que es factible el uso en Grids y si estos cuentan con una red dedicada —de alta velocidad— su eficiencia puede llegar a ser alta.

4.1 Métodos de Descomposición de Dominio

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 Diferencias Finitas o Elemento Finito) obteniendo una familia de subproblemas de tamaño reducido independientes en principio entre si, 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 [15] 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

$$\begin{aligned} \mathcal{L}u &= f \quad \text{en } \Omega \\ u &= g \quad \text{en } \partial\Omega \end{aligned} \tag{4.1}$$

donde

$$\mathcal{L}u = -\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu \tag{4.2}$$

como un caso particular del operador elíptico de orden dos.

4.2 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 como 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 [3].

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:

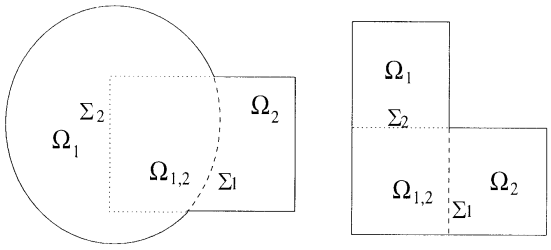

Figura 1: Dominio Ω subdividido en dos subdominios Ω_1 y Ω_2.

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

$$\begin{cases} \mathcal{L}u_1^{k+1} = f \quad \text{en } \Omega_1 \\ u_1^{k+1} = u_2^k \quad \text{en } \Sigma_1 \\ u_1^{k+1} = 0 \quad \text{en } \partial\Omega_1 \cap \partial\Omega \end{cases} \tag{4.3}$$

y

$$\begin{cases} \mathcal{L}u_2^{k+1} = f \quad \text{en } \Omega_2 \\ u_2^{k+1} = u_1^{k+1} \quad \text{en } \Sigma_2 \\ u_2^{k+1} = 0 \quad \text{en } \partial\Omega_2 \cap \partial\Omega \end{cases} \tag{4.4}$$

resolviendo los problemas secuencialmente en cada subdominio (por ejemplo con el método de Diferencias Finitas o Elemento Finito). Este método se conoce como Schwarz multiplicativo.

El método alternante de Schwarz dado por las Ecs. (4.3) y (4.4) converge a la solución u de (4.1) 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

$$\begin{aligned} \left\| u_{|\Omega_1} - u_1^{k+1} \right\|_{L^\infty(\Omega_1)} &\leq C_1^k C_2^k \left\| u - u^0 \right\|_{L^\infty(\Sigma_1)} \\ \left\| u_{|\Omega_2} - u_2^{k+1} \right\|_{L^\infty(\Omega_2)} &\leq C_1^{k+1} C_2^k \left\| u - u^0 \right\|_{L^\infty(\Sigma_2)} \end{aligned} \tag{4.5}$$

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 [2].

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

$$\begin{cases} \mathcal{L}u_1^{k+1} = f \quad \text{en } \Omega_1 \\ u_1^{k+1} = u_2^k \quad \text{en } \Sigma_1 \\ u_1^{k+1} = 0 \quad \text{en } \partial\Omega_1 \cap \partial\Omega \end{cases} \tag{4.6}$$

y

$$\begin{cases} \mathcal{L}u_2^{k+1} = f \quad \text{en } \Omega_2 \\ u_2^{k+1} = u_1^k \quad \text{en } \Sigma_2 \\ u_2^{k+1} = 0 \quad \text{en } \partial\Omega_2 \cap \partial\Omega \end{cases} \tag{4.7}$$

resolviendo los problemas en paralelo de cada subdominio (por ejemplo con el método de Diferencias Finitas o 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:

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

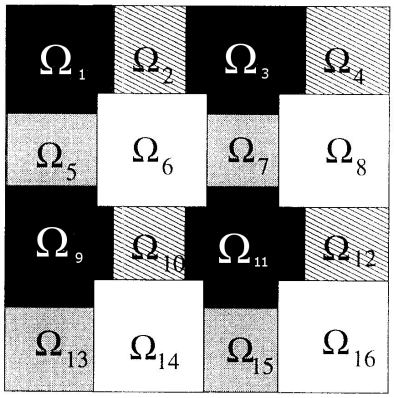

Figura 2: Descomposición de Ω en múltiples subdominios con traslape para el método de Schwarz.

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

$$\left\langle J_i^T F, v \right\rangle = \left\langle F, J_i v \right\rangle, \quad \forall F \in V^{'}, v \in V_i^*. \tag{4.8}$$

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

$$\rho_i^T v_i = \tilde{v}_i \text{ para toda } v_i \in V_i^0 \tag{4.9}$$

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

$$\langle \rho_i G, v_i \rangle = \left\langle G, \rho_i^T v_i \right\rangle, \text{para toda } G \in (V_i^*)', v_i \in V_i^0. \tag{4.10}$$

De lo anterior se tiene que

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

y

$$P_i = J_i P_i^* : V \to V \text{ para } i = 1, ..., E. \tag{4.11}$$

Entonces el método Multiplicativo de Schwarz queda como

$$u^{k+\frac{i}{E}} = (I - P_i) u^{k+\frac{i-1}{E}} + J_i \rho_i^T \mathcal{L}_i^{-1} \rho_i J_i^T f \tag{4.12}$$

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

$$u^{k+1} = \left(I - \sum_{i=1}^{E} P_i \right) u^k + \sum_{i=1}^{M} J_i \rho_i^T \mathcal{L}_i^{-1} \rho_i J_i^T f \tag{4.13}$$

y la correspondiente ecuación de error como

$$u - u^{k+1} = (I - P_m) ... (I - P_1)(u - u^k). \tag{4.14}$$

El el método Aditivo de Schwarz queda como

$$u - u^{k+1} = \left(I - \sum_{i=1}^{E} P_i \right) (u - u^k) \tag{4.15}$$

y la correspondiente ecuación de error como

$$u - u^{k+1} = (I - P_m) ... (I - P_1)(u - u^k). \tag{4.16}$$

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 [**?**], [6] y [8].
- 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 Diferencias Finitas o 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.

4.3 Método de Descomposición de Dominio de Subestructuración (DDM)

La solución numérica por los esquemas tradicionales de discretización tipo Diferencias Finitas y Elemento Finito 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 Diferencias Finitas o Elemento Finito) obteniendo una familia de subproblemas de tamaño reducido independientes en principio entre si, que están acoplados a través de la solución en la interfaz de los subdominios que es desconocida [4], [3], [8] y [10], 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 perdida de generalidad se considerarán problemas con valor en la frontera (VBVP) de la forma

$$\begin{aligned} \mathcal{L}u &= f \quad \text{en } \Omega \\ u &= g \quad \text{en } \partial\Omega \end{aligned} \tag{4.17}$$

donde

$$\mathcal{L}u = -\nabla \cdot \underline{\underline{a}} \cdot \nabla u + cu \tag{4.18}$$

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.

Consideremos el problema dado por la Ec. (4.17) 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

$$\Omega_i \cap \Omega_j = \varnothing \quad \forall i \neq j \quad \text{y} \quad \overline{\Omega} = \bigcup_{i=1}^{E} \overline{\Omega}_i, \tag{4.19}$$

y al conjunto

$$\Gamma = \bigcup_{i=1}^{E} \Gamma_i, \quad \text{si } \Gamma_i = \partial\Omega_i \backslash \partial\Omega \tag{4.20}$$

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:

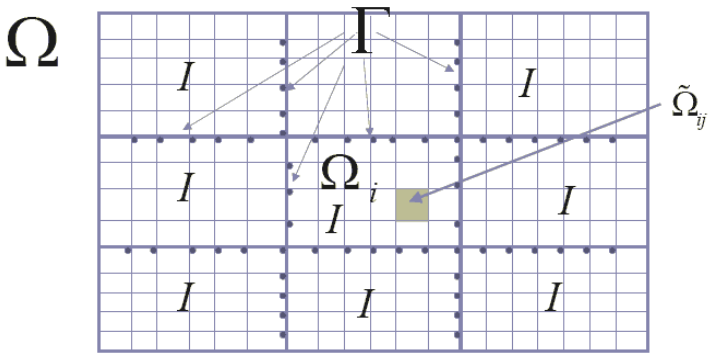

Figura 3: Dominio Ω descompuesto en una partición gruesa de 3×3 y cada subdominio Ω_i en una partición fina de 7×5.

Sin perdida de generalidad tomemos $g = 0$ en $\partial\Omega$, notemos que siempre es posible poner el problema de la Ec. (4.17) 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

$$u \cdot v = \langle u, v \rangle \tag{4.21}$$

Considerando la existencia de los subconjuntos linealmente independientes

$$\begin{gathered} \mathcal{B} \subset \tilde{D}, \mathcal{B}_I \subset \tilde{D}_I, \mathcal{B}_\Gamma \subset \tilde{D}_\Gamma \\ \mathcal{B}_\Gamma \subset \tilde{D}_\Gamma, \mathcal{B}_{\Gamma J} \subset \tilde{D}_{\Gamma 1}, \mathcal{B}_{\Gamma M} \subset \tilde{D}_{\Gamma 2} \end{gathered} \tag{4.22}$$

los cuales satisfacen

$$\mathcal{B} = \mathcal{B}_I \cup \mathcal{B}_\Gamma \text{ y } \bar{\mathcal{B}}_\Gamma = \mathcal{B}_{\Gamma J} \cup \mathcal{B}_{\Gamma M} \tag{4.23}$$

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

$$\mathcal{B}_I = \left\{w_I^1, ..., w_I^{\bar{N}_I}\right\}, \mathcal{B}_{\Gamma M} = \left\{w_M^1, ..., w_M^{\bar{N}_\Gamma}\right\} \text{ y } \mathcal{B}_{\Gamma J} = \left\{w_J^1, ..., w_J^{\bar{N}_\Gamma}\right\} \tag{4.24}$$

de las funcionales lineales ϕ_i en Ω.

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

$$\underline{\underline{A}}_{II}^{\delta} \equiv \left[\left\langle w_I^i, w_I^j \right\rangle\right] \tag{4.25}$$

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

$$\underline{\underline{A}}_{II} \equiv \begin{bmatrix} \underline{\underline{A}}_{II}^1 & & & \\ & \underline{\underline{A}}_{II}^2 & & \\ & & \ddots & \\ & & & \underline{\underline{A}}_{II}^E \end{bmatrix} \tag{4.26}$$

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

De forma similar definimos

$$\underline{\underline{A}}_{I\Gamma}^{\delta} \equiv \left[\left\langle w_I^i, w_\Gamma^\alpha \right\rangle\right], \quad \underline{\underline{A}}_{\Gamma I}^{\delta} \equiv \left[\left\langle w_\Gamma^\alpha, w_I^i \right\rangle\right] \tag{4.27}$$

y

$$\underline{\underline{A}}_{\Gamma\Gamma}^{\delta} \equiv \left[\langle w_\Gamma^\alpha, w_\Gamma^\alpha \rangle\right] \tag{4.28}$$

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

$$\underline{\underline{A}}_{\Gamma\Gamma}^{\delta} = \left[\langle w_\Gamma^\alpha, w_\Gamma^\alpha \rangle\right] = \left[\left\langle w_{\Gamma_J}^\alpha, w_{\Gamma_J}^\alpha \right\rangle\right] + \left[\left\langle w_{\Gamma_M}^\alpha, w_{\Gamma_M}^\alpha \right\rangle\right] \tag{4.29}$$

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

$$\underline{\underline{A}}_{I\Gamma} \equiv \begin{bmatrix} \underline{\underline{A}}_{I\Gamma}^1 \\ \underline{\underline{A}}_{I\Gamma}^2 \\ \vdots \\ \underline{\underline{A}}_{I\Gamma}^E \end{bmatrix} \tag{4.30}$$

$$\underline{\underline{A}}_{\Gamma I} \equiv \begin{bmatrix} \underline{\underline{A}}_{\Gamma I}^1 & \underline{\underline{A}}_{\Gamma I}^2 & \cdots & \underline{\underline{A}}_{\Gamma I}^E \end{bmatrix} \tag{4.31}$$

y

$$\underline{\underline{A}}_{\Gamma\Gamma} \equiv \left[\sum_{i=1}^{E} \underline{\underline{A}}^{i}_{\Gamma\Gamma}\right] \tag{4.32}$$

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

$$\begin{aligned} \underline{\underline{A}}_{II}\underline{u_I} + \underline{\underline{A}}_{I\Gamma}\underline{u_\Gamma} &= \underline{b_I} \\ \underline{\underline{A}}_{\Gamma I}\underline{u_I} + \underline{\underline{A}}_{\Gamma\Gamma}\underline{u_\Gamma} &= \underline{b_\Gamma} \end{aligned} \tag{4.33}$$

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. (4.33) 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. (4.33) entonces tenemos

$$\left(\underline{\underline{A}}_{\Gamma\Gamma} - \underline{\underline{A}}_{\Gamma I}\left(\underline{\underline{A}}_{II}\right)^{-1}\underline{\underline{A}}_{I\Gamma}\right)\underline{u_\Gamma} = \underline{b_\Gamma} - \underline{\underline{A}}_{\Gamma I}\left(\underline{\underline{A}}_{II}\right)^{-1}\underline{b_I} \tag{4.34}$$

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

$$\underline{\underline{S}} = \underline{\underline{A}}_{\Gamma\Gamma} - \underline{\underline{A}}_{\Gamma I}\left(\underline{\underline{A}}_{II}\right)^{-1}\underline{\underline{A}}_{I\Gamma}. \tag{4.35}$$

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

$$\underline{\underline{S}}_i = \underline{\underline{A}}^i_{\Gamma\Gamma} - \underline{\underline{A}}^i_{\Gamma I}\left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{\underline{A}}^i_{I\Gamma} \tag{4.36}$$

adicionalmente definimos

$$\underline{b}_i = \underline{b_\Gamma}_i - \underline{\underline{A}}^i_{\Gamma I}\left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{b_I}_i. \tag{4.37}$$

El sistema dado por la Ec. (4.34) lo escribimos como

$$\underline{\underline{S}}\,\underline{u_\Gamma} = \underline{b} \tag{4.38}$$

y queda definido de manera virtual a partir de

$$\left[\sum_{i=1}^{E}\underline{\underline{S}}_i\right]\underline{u_\Gamma} = \left[\sum_{i=1}^{E}\underline{b}_i\right] \tag{4.39}$$

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. (4.38) 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. (4.39) 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

$$\underline{u_I}_i = \left(\underline{\underline{A}}^i_{II}\right)^{-1}\left(\underline{b_I}_i - \underline{\underline{A}}^i_{I\Gamma}\underline{u_\Gamma}_i\right) \tag{4.40}$$

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*

$$\underline{\tilde{u}_\Gamma}^i = \left(\underline{\underline{A}}^i_{\Gamma\Gamma} - \underline{\underline{A}}^i_{\Gamma I}\left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{\underline{A}}^i_{I\Gamma}\right)\underline{y_\Gamma}_i. \tag{4.41}$$

Para evaluar eficientemente esta expresión, realizamos las siguientes operaciones equivalentes

$$\begin{aligned} \underline{x1} &= \underline{\underline{A}}^i_{\Gamma\Gamma}\underline{y_\Gamma}_i \\ \underline{x2} &= \left(\underline{\underline{A}}^i_{\Gamma I}\left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{\underline{A}}^i_{I\Gamma}\right)\underline{y_\Gamma}_i \\ \underline{\tilde{u}_\Gamma}^i &= \underline{x1} - \underline{x2} \end{aligned} \tag{4.42}$$

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

$$\underline{x3} = \underline{\underline{A}}^i_{I\Gamma}\underline{y_\Gamma}_i \tag{4.43}$$

con este resultado intermedio deberíamos calcular

$$\underline{x4} = \left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{x3} \tag{4.44}$$

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

$$\underline{\underline{A}}^i_{II}\underline{x4} = \underline{\underline{A}}^i_{II}\left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{x3} \tag{4.45}$$

al simplificar, tenemos

$$\underline{\underline{A}}^i_{II}\underline{x4} = \underline{x3}. \tag{4.46}$$

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*

$$\underline{x2} = \underline{\underline{A}}^i_{\Gamma I}\underline{x4} \tag{4.47}$$

así

$$\underline{\tilde{u}_\Gamma}^i = \underline{x1} - \underline{x2} \tag{4.48}$$

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

Observación 2 *En el caso del cálculo de*

$$\underline{b}_i = \underline{b_\Gamma}_i - \underline{\underline{A}}^i_{\Gamma I}\left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{b_I}_i \tag{4.49}$$

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*

$$\underline{y1} = \left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{b_I}_i \tag{4.50}$$

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

$$\underline{\underline{A}}^i_{II}\underline{y1} = \underline{\underline{A}}^i_{II}\left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{b_I}_i \tag{4.51}$$

simplificando, tenemos

$$\left(\underline{\underline{A}}^i_{II}\right)\underline{y1} = \underline{b_I}_i \tag{4.52}$$

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

$$\underline{y2} = \underline{\underline{A}}^i_{\Gamma I}\underline{y1} \tag{4.53}$$

y para finalizar el cálculo, calculamos

$$\underline{b}_i = \underline{b_\Gamma}_i - \underline{y2}. \tag{4.54}$$

Observación 3 *En la evaluación de*

$$\underline{u_I}_i = \left(\underline{\underline{A}}^i_{II}\right)^{-1}\left(\underline{b_I}_i - \underline{\underline{A}}^i_{I\Gamma}\underline{u_\Gamma}_i\right) \tag{4.55}$$

esta nuevamente 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*

$$\begin{aligned} \underline{x4} &= \underline{b_I}_i - \underline{\underline{A}}^i_{I\Gamma}\underline{u_\Gamma}_i \\ \underline{u_I}_i &= \left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{x4} \end{aligned} \tag{4.56}$$

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

$$\underline{\underline{A}}^i_{II}\underline{u_I}_i = \underline{\underline{A}}^i_{II}\left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{x4} \tag{4.57}$$

simplificando, tenemos

$$\underline{\underline{A}}^i_{II}\underline{u_I}_i = \underline{x4} \tag{4.58}$$

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

Como se indico en las últimas observaciones, para resolver el sistema $\underline{\underline{A}}^i_{II}\underline{x} = \underline{b}$ 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 del 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

$$\begin{aligned} \underline{\underline{L}}\underline{y} &= \underline{b} \\ \underline{\underline{U}}\underline{x} &= \underline{y} \end{aligned} \tag{4.59}$$

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

$$\left\|\underline{u_\Gamma}\right\|^2_\Gamma = \sum_{i=1}^{E} \left\|\underline{u_\Gamma}\right\|^2_{L^2(\partial\Omega_i)}. \tag{4.60}$$

Teorema 8 *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*

$$cH\left\|\underline{u_\Gamma}\right\|^2_\Gamma \leq s(\underline{u_\Gamma}, \underline{u_\Gamma}) \leq Ch^{-1}\left\|\underline{u_\Gamma}\right\|^2_\Gamma \tag{4.61}$$

de este modo

$$\kappa = cond(\underline{\underline{S}}) \leq \frac{C}{Hh}. \tag{4.62}$$

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

$$\begin{pmatrix} \underline{\underline{A}} & \underline{\underline{B}} \\ \underline{\underline{C}} & \underline{\underline{D}} \end{pmatrix} \begin{pmatrix} \underline{u} \\ \underline{v} \end{pmatrix} = \begin{pmatrix} \underline{a} \\ \underline{b} \end{pmatrix} \tag{4.63}$$

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

$$\left(\underline{\underline{D}} - \underline{\underline{C}}\underline{\underline{A}}^{-1}\underline{\underline{B}}\right)\underline{v} = \underline{b} - \underline{\underline{C}}\underline{\underline{A}}^{-1}\underline{a} \tag{4.64}$$

y donde

$$\underline{u} = \underline{\underline{A}}^{-1}\left(\underline{a} - \underline{\underline{B}}\underline{v}\right). \tag{4.65}$$

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

$$\underline{\underline{N}}\underline{v} = \underline{g} \tag{4.66}$$

pero con un menor número de grados de libertad, donde

$$\underline{\underline{N}} = \left(\underline{\underline{D}} - \underline{\underline{C}}\underline{\underline{A}}^{-1}\underline{\underline{B}}\right), \qquad \underline{g} = \underline{b} - \underline{\underline{C}}\underline{\underline{A}}^{-1}\underline{a}. \tag{4.67}$$

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.

4.4 Métodos del Espacio de Vectores Derivados (DVS)

El espacio de vectores derivados (DVS) es mostrado como una formulación numérica la cual esta íntimamente relacionada con la implementación computacional, tanto en la versión secuencial como paralela del código (véase [62] y [61]).

La implementación computacional dicta ciertas consideraciones sobre selección de los algoritmos numéricos, así como, la forma de implementación de estos. Con la única finalidad de aprovechar sus propiedades computacionales en la definición de una robusta jerarquía de clases que resuelva de forma eficiente el problema.

Para ello, se inicia considerando el operador de segundo orden dado por la ecuación

$$\begin{aligned} \mathcal{L}u &= f \quad \text{en } \Omega \\ u &= g \quad \text{sobre } \partial\Omega \end{aligned} \tag{4.68}$$

Consideremos el problema dado por la Ec. (4.68) 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

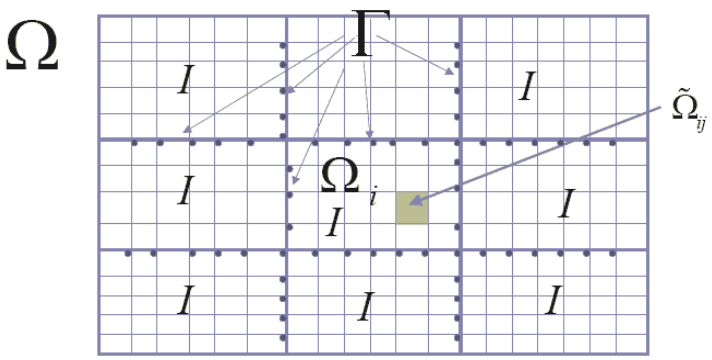

Figura 4: Dominio Ω descompuesto en una partición gruesa de 3×3 y cada subdominio Ω_i en una partición fina de 7×5.

$$\Omega_i \cap \Omega_j = \varnothing \quad \forall i \neq j \quad \text{y} \quad \overline{\Omega} = \bigcup_{i=1}^{E} \overline{\Omega}_i, \tag{4.69}$$

y al conjunto

$$\Gamma = \bigcup_{i=1}^{E} \Gamma_i, \quad \text{si } \Gamma_i = \partial\Omega_i \backslash \partial\Omega \tag{4.70}$$

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 anterior. Sin perdida de generalidad tomemos $g = 0$ en $\partial\Omega$, notemos que siempre es posible poner el problema de la Ec. (4.68) como uno con condiciones de frontera Dirichlet que se nulifiquen mediante la adecuada manipulación del término del lado derecho de la ecuación.

Para resolver dicho problema, se puede usar cualquiera de los ocho algoritmos que se han desarrollado, de cada uno de ellos se han dado formulaciones explícitas en términos de matrices.

Así, para generar la implementación de cualquiera de los algoritmos desarrollados se necesita generar las matrices locales

$$\underline{\underline{A}}^{\alpha}_{\Pi\Pi}, \underline{\underline{A}}^{\alpha}_{\Pi\Delta}, \underline{\underline{A}}^{\alpha}_{\Delta\Pi}, \underline{\underline{A}}^{\alpha}_{\Delta\Delta} \tag{4.71}$$

que están definidas de $W_r \rightarrow W_r$, o más explícitamente las matrices

$$\underline{\underline{A}}^{\alpha}_{II}, \underline{\underline{A}}^{\alpha}_{I\pi}, \underline{\underline{A}}^{\alpha}_{I\Delta}, \underline{\underline{A}}^{\alpha}_{\pi I}, \underline{\underline{A}}^{\alpha}_{\pi\pi}, \underline{\underline{A}}^{\alpha}_{\pi\Delta}, \underline{\underline{A}}^{\alpha}_{\Delta I}, \underline{\underline{A}}^{\alpha}_{\Delta\pi} \text{ y } \underline{\underline{A}}^{\alpha}_{\Delta\Delta} \tag{4.72}$$

ello se puede hacer de dos formas, a saber:

- A partir de la matriz que es obtenida después de que el problema se ha discretizado —a partir de una discretización por el método de Elemento Finito, Volumen Finito o Diferencias Finitas— y para su aplicación no se requiere ninguna información acerca de la ecuación diferencial parcial de la cual se originó.
- A partir la discretización de la ecuación diferencial parcial generada localmente en cada subdominio por algún método de descomposición de dominio sin traslapes, como el usado para FETI-DP o BDDC.

En la sección (B.8) se derivo la forma de obtener las matrices locales a partir de la matriz $\underline{\underline{A}}^{t} : W \rightarrow W$ que es obtenida después de que el problema se ha discretizado y la cual es puesta en el espacio de vectores derivados. En la siguiente sección se da la forma de derivar cada una de las matrices locales a partir la discretización de la ecuación diferencial parcial generada localmente en cada subdominio; para que en las siguientes secciones, mediante el uso de dichas matrices se pueda implementar cualquiera de los métodos desarrollados.

4.4.1 Discretización de los Métodos Partiendo de la Formulación Local

Los métodos de descomposición de dominio —como son FETI-DP y BDDC— y el esquema DVS puede partir de la discretización local mediante algún método

de discretización como Diferencias Finitas, Volumen Finito o Elemento Finito –este último es uno de los más usados en los DDM— para construir cada una de las matrices locales

$$\underline{\underline{A}}^{\alpha}_{II}, \underline{\underline{A}}^{\alpha}_{I\pi}, \underline{\underline{A}}^{\alpha}_{I\Delta}, \underline{\underline{A}}^{\alpha}_{\pi I}, \underline{\underline{A}}^{\alpha}_{\pi\pi}, \underline{\underline{A}}^{\alpha}_{\pi\Delta}, \underline{\underline{A}}^{\alpha}_{\Delta I}, \underline{\underline{A}}^{\alpha}_{\Delta\pi} \text{ y } \underline{\underline{A}}^{\alpha}_{\Delta\Delta} \tag{4.73}$$

para cada Ω_α con $\alpha = 1, ..., E$. Una vez construida las matrices locales, se procede a definir al complemento de Schur local por subdominio —más detalles véase sección (4.3)—

$$\underline{\underline{S}}^{\alpha}_{\pi} = \underline{\underline{A}}^{\alpha}_{\pi\pi} - \underline{\underline{A}}^{\alpha}_{\pi I} \left(\underline{\underline{A}}^{\alpha}_{II} \right)^{-1} \underline{\underline{A}}^{\alpha}_{I\pi} \tag{4.74}$$

con él se define

$$\underline{\underline{A}}^{\alpha}_{\Pi\Pi} = \begin{pmatrix} \underline{\underline{A}}^{\alpha}_{II} & \underline{\underline{A}}^{\alpha}_{I\pi} \\ \underline{\underline{A}}^{\alpha}_{\pi I} & \underline{\underline{A}}^{\alpha}_{\pi\pi} \end{pmatrix} \tag{4.75}$$

que a su vez define

$$\underline{\underline{S}}^{\alpha} = \underline{\underline{A}}^{\alpha}_{\Delta\Delta} - \underline{\underline{A}}^{\alpha}_{\Delta\Pi} \left(\underline{\underline{A}}^{\alpha}_{\Pi\Pi} \right)^{-1} \underline{\underline{A}}^{\alpha}_{\Pi\Delta} \tag{4.76}$$

recordando que

$$\begin{cases} \underline{\underline{A}}_{\Pi\Pi} = \sum_{\alpha=1}^{E} \underline{\underline{A}}^{\alpha}_{\Pi\Pi}, & \underline{\underline{A}}_{\Pi\Delta} = \sum_{\alpha=1}^{E} \underline{\underline{A}}^{\alpha}_{\Pi\Delta} \\ \underline{\underline{A}}_{\Delta\Pi} = \sum_{\alpha=1}^{E} \underline{\underline{A}}^{\alpha}_{\Delta\Pi}, & \underline{\underline{A}}_{\Delta\Delta} = \sum_{\alpha=1}^{E} \underline{\underline{A}}^{\alpha}_{\Delta\Delta} \end{cases} \tag{4.77}$$

$$\underline{\underline{S}} = \sum_{\alpha=1}^{E} \underline{\underline{S}}^{\alpha} \qquad \text{y} \qquad \left(\underline{\underline{S}} \right)^{-1} = \sum_{\alpha=1}^{E} \left(\underline{\underline{S}}^{\alpha} \right)^{-1} \tag{4.78}$$

entonces, finalmente se tienen definidas a $\underline{\underline{S}}$ y $\underline{\underline{S}}^{-1}$. Con estas definiciones, ahora ya es posible implementar cualesquiera de los métodos del esquema del espacio de vectores derivados.

Nótese que, por la forma de construcción de las matrices, se tienen las siguientes propiedades importantes

$$\left(\underline{\underline{A}}_{\Pi\Pi} \right)^{-1} = \sum_{\alpha=1}^{E} \left(\underline{\underline{A}}^{\alpha}_{\Pi\Pi} \right)^{-1} \qquad \text{y} \qquad \left(\underline{\underline{A}} \right)^{-1} = \sum_{\alpha=1}^{E} \left(\underline{\underline{A}}^{\alpha} \right)^{-1} \tag{4.79}$$

esto implica que el cálculo de la inversa de la matriz $\underline{\underline{A}}_{\Pi\Pi}$ es exclusivamente a partir de las inversas locales a cada subdominio.

4.4.2 Formulación Operacional de los Métodos DVS

Para la resolución de problemas de interés en Ciencias e Ingenierías donde es necesario resolver, por ejemplo el problema dado por la Ec.(B.96) —más detalles véase la sección (B.9)— que consiste en buscar una función $\underline{u}^{'} \in W_r$ que satisfaga

$$\underline{\underline{aA}}\underline{u}^{'} = \underline{f} \qquad \text{y} \qquad \underline{\underline{j}}\underline{u}^{'} = 0 \tag{4.80}$$

aquí, el vector $\overline{\underline{f}} \in W_{12}$ es un dato del problema y donde el vector $\underline{u}^{'}$ y $\overline{\underline{f}}$ esta formado por

$$\begin{pmatrix} \underline{u}_{\Pi} \\ \underline{u}_{\Delta} \end{pmatrix} = \underline{u}^{'} \qquad \text{y} \qquad \begin{pmatrix} \underline{f}_{\Pi} \\ \underline{f}_{\Delta} \end{pmatrix} = \overline{\underline{f}} \tag{4.81}$$

que satisfaga —véase sección (C.3) del operador Steklov-Poincaré Ecs.(C.62 y C.63)—

$$\left(\underline{\underline{L}} + \underline{\underline{aR}} - \underline{\underline{R}}^{T}\,\underline{\underline{j}}\right)\underline{u}^{'} = \overline{\underline{f}}. \tag{4.82}$$

Entonces es necesario transformar el problema Ec.(4.82) en uno que

$$\underline{f}_{\Pi} = 0 \tag{4.83}$$

para ello, se introduce un vector auxiliar

$$\underline{u}_{p} = \underline{\underline{A}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{f}_{\Pi} \tag{4.84}$$

en el cual $\left(\underline{u}_{p}\right)_{\Delta} = 0$, por lo tanto la Ec.(4.82) toma la forma

$$\left(\underline{\underline{aR}} - \underline{\underline{R}}^{T}\,\underline{\underline{j}}\right)\underline{u}_{\Delta} = \underline{f}_{\Delta} - \underline{u}_{p} \tag{4.85}$$

así, la solución $\underline{u}^{'}$ al problema será $\underline{u}^{'} = \underline{u}_{\Delta} - \underline{u}_{p}$.

Dado que se necesita expresar el problema en términos del espacio W_{12} entonces

$$\underline{f}_{\Delta_2} = \underline{\underline{a}}\underline{f}_{\Delta}, \qquad \text{y} \qquad \underline{f}_{\Delta_1} = \underline{\underline{j}}\underline{f}_{\Delta} = 0 \tag{4.86}$$

$$\left(\underline{u}_{p}\right)_{\Delta} = \underline{\underline{aA}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{f}_{\Pi} \tag{4.87}$$

de lo anterior, la expresión dada por la Ec.(4.85), se puede reescribir como

$$\left(\underline{\underline{aS}} - \underline{\underline{S}}\underline{\underline{j}}\right)\underline{u}_{\Delta} = \underline{\underline{a}}\underline{f}_{\Delta} - \underline{\underline{aA}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{f}_{\Pi} \tag{4.88}$$

donde

$$\underline{\underline{S}} = \underline{\underline{A}}_{\Delta\Delta} - \underline{\underline{A}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{\underline{A}}_{\Pi\Delta} \tag{4.89}$$

o más compactamente se escribe como

$$\left(\underline{\underline{aS}} - \underline{\underline{S}}\underline{\underline{j}}\right)\underline{u}_{\Delta} = \underline{f}_{\Delta} - \left(\underline{u}_{p}\right)_{\Delta}. \tag{4.90}$$

Por todo lo anterior, los algoritmos desarrollados quedan escritos de manera explícita como:

1. **Formulacion del Algoritmo del Complemento de Schur** (PRIMAL#1) —véase ecuación (C.4) en la sección (C.1.1)—:

   "Encontrar a $\underline{u}_\Delta \in W_\Delta$ tal que

   $$\underline{\underline{aSu}}_\Delta = \underline{f}_\Delta - \left(\underline{u}_p\right)_\Delta \tag{4.91}$$

   sujeto a $\underline{\underline{j}}\underline{u}_\Delta = 0$". O en su forma desarrollada

   $$\underline{\underline{aSu}}_\Delta = \underline{f}_\Delta - \underline{\underline{aA}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{f}_\Pi. \tag{4.92}$$

2. **Formulación Dual del Problema Neumann-Neumann** (DUAL#1) —véase la ecuación (C.11) de la sección (C.1.2)—:

   "Dada $\underline{f}_\Delta \in W_\Delta$, buscar a $\underline{\lambda} \in W_\Delta$ tal que

   $$\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{\lambda}_\Delta = \underline{\underline{j}}\underline{\underline{S}}^{-1}\left(\underline{f}_\Delta - \left(\underline{u}_p\right)_\Delta\right) \tag{4.93}$$

   sujeto a $\underline{\underline{a}}\underline{\lambda}_\Delta = 0$". O en su forma desarrollada

   $$\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{\lambda}_\Delta = \underline{\underline{j}}\underline{\underline{S}}^{-1}\left(\underline{f}_\Delta - \underline{\underline{aA}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{f}_\Pi\right). \tag{4.94}$$

3. **Formulación Primal del Problema Neumann-Neumann** (PRIMAL#2) —véase ecuación (C.21) en la sección (C.1.3)—:

   "Dada $\underline{f}_\Delta \in W_\Delta$, encontrar $\underline{v}_\Delta \in W_\Delta$ tal que

   $$\underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{v}_\Delta = \underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{\underline{S}}^{-1}\left(\underline{f}_\Delta - \left(\underline{u}_p\right)_\Delta\right) \tag{4.95}$$

   sujeto a $\underline{\underline{aSv}}_\Delta = 0$". O en su forma desarrollada

   $$\underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{v}_\Delta = \underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{\underline{S}}^{-1}\left(\underline{f}_\Delta - \underline{\underline{aA}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{f}_\Pi\right). \tag{4.96}$$

4. **Formulación Dual del Problema Neumann-Neumann** (DUAL#2) —véase la ecuación (C.29) de la sección (C.1.4)—:

   "Dada $\underline{f}_\Delta \in W_\Delta$, sea $\underline{\mu}_\Delta \in W_\Delta$ tal que

   $$\underline{\underline{Sa\mu}}_\Delta = \underline{\underline{SaS}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\left(\underline{f}_\Delta - \left(\underline{u}_p\right)_\Delta\right) \tag{4.97}$$

   sujeto a $\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{\mu}_\Delta = 0$". O en su forma desarrollada

   $$\underline{\underline{Sa\mu}}_\Delta = \underline{\underline{SaS}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\left(\underline{f}_\Delta - \underline{\underline{aA}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{f}_\Pi\right). \tag{4.98}$$

5. **Formulación Precondicionada del Algoritmo BDDC** (PRIMAL#1) —véase la ecuación (C.30) en la sección (C.2.1)—:

   "Buscar a $\underline{u}_\Delta \in W_\Delta$ tal que

$$\underline{\underline{aS}}^{-1}\underline{\underline{aS}}\underline{u}_\Delta = \underline{\underline{aS}}^{-1}\left(\underline{f}_\Delta - \left(\underline{u}_p\right)_\Delta\right) \tag{4.99}$$

   sujeto a $\underline{\underline{j}}\underline{u}_\Delta = 0$". O en su forma desarrollada

$$\underline{\underline{aS}}^{-1}\underline{\underline{aS}}\underline{u}_\Delta = \underline{\underline{aS}}^{-1}\left(\underline{f}_\Delta - \underline{\underline{aA}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{f}_\Pi\right). \tag{4.100}$$

6. **Formulación Precondicionada del Algoritmo FETI-DP** (DUAL#1) —véase la ecuación (C.36) en la sección (C.2.2)—:

   "Dada $\underline{f}_\Delta \in W_\Delta$, sea $\underline{\lambda}_\Delta \in W_\Delta$ tal que

$$\underline{\underline{j}}\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{\lambda}_\Delta = \underline{\underline{j}}\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\left(\underline{f}_\Delta - \left(\underline{u}_p\right)_\Delta\right) \tag{4.101}$$

   sujeto a $\underline{\underline{a}}\underline{\lambda}_\Delta = 0$". O en su forma desarrollada

$$\underline{\underline{j}}\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{\lambda}_\Delta = \underline{\underline{j}}\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\left(\underline{f}_\Delta - \underline{\underline{aA}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{f}_\Pi\right) \tag{4.102}$$

   donde una vez calculada $\underline{\lambda}_\Delta$ entonces $\underline{u}_\Delta$ es obtenida mediante

$$\underline{u}_\Delta = \underline{\underline{aS}}^{-1}\left(\underline{f}_\Delta - \underline{\underline{j}}\underline{\lambda}_\Delta\right). \tag{4.103}$$

7. **Formulación Primal Precondicionada del Problema Neumann-Neumann DVS-PRIMAL** (PRIMAL#2) —véase la ecuación (C.43) en la sección (C.2.3)—:

   "Dada $\underline{f}_\Delta \in W_\Delta$, buscar $\underline{v} \in W_\Delta$ tal que

$$\underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{\underline{S}}\underline{\underline{j}}\underline{v}_\Delta = \underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\left(\underline{f}_\Delta - \left(\underline{u}_p\right)_\Delta\right) \tag{4.104}$$

   sujeto a $\underline{\underline{aS}}\underline{v}_\Delta = 0$". O en su forma desarrollada

$$\underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{\underline{S}}\underline{\underline{j}}\underline{v}_\Delta = \underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\left(\underline{f}_\Delta - \underline{\underline{aA}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{f}_\Pi\right) \tag{4.105}$$

   donde una vez calculada $\underline{v}_\Delta$ entonces $\underline{u}_\Delta$ es obtenida mediante

$$\underline{u}_\Delta = \underline{\underline{aS}}^{-1}\left(\underline{f}_\Delta - \underline{\underline{j}}\underline{\underline{S}}\underline{v}_\Delta\right). \tag{4.106}$$

8. **Formulación Dual Precondicionada del Problema Neumann-Neumann DVS-DUAL** (DUAL#2) —véase la ecuación (C.50) en la sección (C.2.4)—:

   "Dada $\underline{f}_{\Delta} \in W_{\Delta}$, sea $\underline{\mu}_{\Delta} \in W_{\Delta}$ tal que

$$\underline{\underline{SaS}}^{-1}\underline{\underline{a}}\underline{\mu}_{\Delta} = \underline{\underline{SaS}}^{-1}\underline{\underline{aS}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\left(\underline{f}_{\Delta} - \left(\underline{u}_{p}\right)_{\Delta}\right) \tag{4.107}$$

   sujeto a $\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{\mu}_{\Delta} = 0$". O en su forma desarrollada

$$\underline{\underline{SaS}}^{-1}\underline{\underline{a}}\underline{\mu}_{\Delta} = \underline{\underline{SaS}}^{-1}\underline{\underline{aS}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\left(\underline{f}_{\Delta} - \underline{\underline{aA}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{f}_{\Pi}\right) \tag{4.108}$$

   donde una vez calculada $\underline{\mu}_{\Delta}$ entonces $\underline{u}_{\Delta}$ es obtenida mediante

$$\underline{u}_{\Delta} = \underline{\underline{aS}}^{-1}\left(\underline{f}_{\Delta} + \underline{\mu}_{\Delta}\right). \tag{4.109}$$

4.4.3 Implementación Numérica de DVS

La implementación numérica de por ejemplo, el algoritmo DVS-BDDC (PRIMAL#1) puesto en su forma operacional, Ec.(4.100) es

$$\underline{\underline{aS}}^{-1}\underline{\underline{aS}}\underline{u}_{\Delta} = \underline{\underline{aS}}^{-1}\left(\underline{f}_{\Delta} - \underline{\underline{aA}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{f}_{\Pi}\right) \tag{4.110}$$

en la cual define el sistema lineal virtual a resolver

$$\underline{\underline{M}}\underline{u}_{\Delta} = \underline{b} \tag{4.111}$$

donde

$$\underline{\underline{M}} = \underline{\underline{aS}}^{-1}\underline{\underline{aS}} \quad \text{y} \quad \underline{b} = \underline{\underline{aS}}^{-1}\left(\underline{f}_{\Delta} - \underline{\underline{aA}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{f}_{\Pi}\right) \tag{4.112}$$

entonces el sistema lineal virtual puede ser implementado mediante el método de Gradiente Conjugado o alguna variante de GMRES, dependiendo del tipo de matriz que sea $\underline{\underline{S}}$. Si $\underline{\underline{S}}$ es simétrica y definida positiva, entonces $\underline{\underline{S}}^{-1}$ también será simétrica y definida positiva y por tanto se usaría el método de Gradiente Conjugado, en caso contrario se usaría el método de GMRES o algún otro.

Nótese que, en los métodos iterativos, la adecuada selección de $\underline{u}^0$ puede ser tan costosa —computacionalmente hablando— como encontrar la solución $\underline{u}$, pues tomar una $\underline{u}^0$ no adecuada, generalmente ocasiona realizar más iteraciones para converger en el método que tomar $\underline{u}^0 = 0$.

4.4.4 Implementación para Matrices Simétricas

Suponiendo que $\underline{\underline{S}}$ es simétrica y definida positiva, la formulación $\underline{\underline{M}}\underline{u}_{\Delta} = \underline{b}$ puede ser implementada como el método de Gradiente Conjugado —véase sección (A.2.1)— usando el algoritmo dado en la Ec.(A.14) en el cual se usa el producto interior según corresponda:

- $\underline{\underline{aS}}^{-1}\underline{\underline{aS}}$ será simétrico con respecto al producto interior definido por

$$\langle \underline{u}, \underline{w} \rangle = \underline{u} \cdot \underline{w} \tag{4.113}$$

- $\underline{\underline{Sj}}\underline{\underline{S}}^{-1}\underline{\underline{j}}$ será simétrico con respecto al producto interior definido por

$$\langle \underline{u}, \underline{w} \rangle = \underline{\underline{S}}\underline{u} \cdot \underline{w} \tag{4.114}$$

La implementación del algoritmo se inicia tomando $\underline{u}^0$, una elección común es tomar a $\underline{u}^0 = 0$, si este es el caso, entonces

$$\underline{r}^0 = \underline{\underline{aS}}^{-1}\left(\underline{f}_\Delta - \underline{\underline{aA}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{f}_\Pi\right), \qquad \underline{p}^0 = \underline{r}^0 \tag{4.115}$$

y la parte iterativa[20] queda como:

$$\begin{array}{l}
\text{Para } n = 1, 2, ...\{ \\
\quad \alpha^n = \frac{\langle \underline{p}^n, \underline{\underline{S}}\underline{p}^n \rangle}{\langle \underline{p}^n, \underline{\underline{SaS}}^{-1}\underline{\underline{aS}}\underline{p}^n \rangle} \\
\quad \underline{u}^{n+1} = \underline{u}^n + \alpha^n \underline{p}^n \\
\quad \underline{r}^{n+1} = \underline{r}^n - \alpha^n \underline{\underline{aS}}^{-1}\underline{\underline{aS}}\underline{p}^n \\
\quad < \text{Prueba de convergencia} > \\
\quad \beta^n = \frac{\langle \underline{r}^{n+1}, \underline{\underline{S}}\underline{r}^{n+1} \rangle}{\langle \underline{r}^n, \underline{\underline{S}}\underline{r}^n \rangle} \\
\quad \underline{p}^{n+1} = \underline{r}^{n+1} + \beta^n \underline{p}^n \\
\}
\end{array} \tag{4.116}$$

4.4.5 Implementación para Matrices no Simétricas e Indefinidas

Suponiendo que $\underline{\underline{S}}$ es no simétrica o indefinida, la formulación $\underline{\underline{M}}\underline{u}_\Delta = \underline{b}$ puede ser implementada como el método de GMRES —véase sección (A.2.2)— usando el algoritmo dado en la Ec.(4.100), en cual se inicia tomando $\underline{u}^0$, una elección común es tomar a $\underline{u}^0 = 0$, si este es el caso, entonces

$$\underline{r}^0 = \underline{\underline{aS}}^{-1}\left(\underline{f}_\Delta - \underline{\underline{aA}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{f}_\Pi\right), \beta^0 = \left\|r^0\right\|, \underline{v}^1 = \underline{r}^0/\beta^0 \tag{4.117}$$

y la parte iterativa queda como:

$$\begin{array}{l}
\text{Para } n = 1, 2, ..., \text{Mientras } \beta^n < \tau\beta^0 \ \{ \\
\quad \underline{w}_0^{n+1} = \underline{\underline{aS}}^{-1}\underline{\underline{aS}}\underline{v}^n \\
\quad \text{Para } l = 1 \text{ hasta } n \ \{ \\
\qquad h_{l,n} = \langle \underline{w}_l^{n+1}, \underline{v}^l \rangle \\
\qquad \underline{w}_{l+1}^{n+1} = \underline{w}_l^{n+1} - h_{l,n}\underline{v}^l \\
\quad \} \\
\quad h_{n+1,n} = \left\|\underline{w}_{n+1}^{n+1}\right\| \\
\quad \underline{v}^{n+1} = \underline{w}_{n+1}^{n+1}/h_{n+1,n} \\
\quad \text{Calcular } \underline{y}^n \text{ tal que } \beta n = \left\|\beta^0\underline{e}_1 - \underline{\underline{\hat{H}}}_n\underline{y}^y\right\| \text{ es mínima} \\
\}
\end{array} \tag{4.118}$$

[20] En este caso se usa el producto interior definido por $\langle \underline{u}, \underline{w} \rangle = \underline{\underline{S}}\underline{u} \cdot \underline{w}$.

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

$$\underline{r}^n = \underline{r}^0 - \underline{\underline{aS}}^{-1} \underline{\underline{aSV}}_k y^n = \underline{\underline{V}}_{n+1} \left(\beta^0 \underline{e}_1 - \underline{\underline{\hat{H}}}_n \underline{y}^n \right). \tag{4.119}$$

4.4.6 Evaluación de los Operadores Virtuales $\underline{\underline{S}}$ y $\underline{\underline{S}}^{-1}$

Para implementar cualesquiera de los ocho algoritmos desarrollados, sin importar si las matrices involucradas son simétricas o no simétricas; y como se mostró en las implementaciones de los algoritmos en la sección anterior, estos requieren por un lado, la evaluación de $\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1} \underline{f}_\Pi$ y por otro la evaluación de los operadores virtuales $\underline{\underline{S}}^{-1}\underline{v}$ y $\underline{\underline{S}}\underline{v}$, en los tres casos, ninguna de estas matrices es construida pues resultarían matrices densas con el consiguiente costo computacional de su manejo. Para mostrar como hacer su evaluación óptima, primero nótese que el sistema lineal virtual $\underline{\underline{A}}$ a partir del cual se deriva el esquema DVS, está definido como

$$\underline{\underline{A}} = \begin{pmatrix} \underline{\underline{A}}_{\Pi\Pi} & \underline{\underline{A}}_{\Pi\Delta} \\ \underline{\underline{A}}_{\Delta\Pi} & \underline{\underline{A}}_{\Delta\Delta} \end{pmatrix} = \begin{pmatrix} \underline{\underline{A}}_{II} & \underline{\underline{A}}_{I\pi} & \underline{\underline{A}}_{I\Delta} \\ \underline{\underline{A}}_{\pi I} & \underline{\underline{A}}_{\pi\pi} & \underline{\underline{A}}_{\pi\Delta} \\ \underline{\underline{A}}_{\Delta I} & \underline{\underline{A}}_{\Delta\pi} & \underline{\underline{A}}_{\Delta\Delta} \end{pmatrix} \tag{4.120}$$

donde

$$\begin{aligned} \underline{\underline{A}}_{\Pi\Pi} &= \begin{pmatrix} \underline{\underline{A}}_{II} & \underline{\underline{A}}_{I\pi} \\ \underline{\underline{A}}_{\pi I} & \underline{\underline{A}}_{\pi\pi} \end{pmatrix} \qquad \underline{\underline{A}}_{\Pi\Delta} = \begin{pmatrix} \underline{\underline{A}}_{I\Delta} \\ \underline{\underline{A}}_{\pi\Delta} \end{pmatrix} \\ \underline{\underline{A}}_{\Delta\Pi} &= \begin{pmatrix} \underline{\underline{A}}_{\Delta I} & \underline{\underline{A}}_{\Delta\pi} \end{pmatrix} \end{aligned} \tag{4.121}$$

entonces el operador $\underline{\underline{S}}$ de los nodos duales queda definido por

$$\underline{\underline{S}} = \underline{\underline{A}}_{\Delta\Delta} - \underline{\underline{A}}_{\Delta\Pi} \left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1} \underline{\underline{A}}_{\Pi\Delta} \tag{4.122}$$

y este es formado por $\underline{\underline{S}} = \sum_{\alpha=1}^{E} \underline{\underline{S}}^\alpha$, donde $\underline{\underline{S}}^\alpha$ esta formada por el complemento de Schur local

$$\underline{\underline{S}}^\alpha = \underline{\underline{A}}^\alpha_{\Delta\Delta} - \underline{\underline{A}}^\alpha_{\Delta\Pi} \left(\underline{\underline{A}}^\alpha_{\Pi\Pi}\right)^{-1} \underline{\underline{A}}^\alpha_{\Pi\Delta}. \tag{4.123}$$

En el apéndice A se detallan la forma de realizar todas las operaciones involucradas en los métodos DVS, aquí, bajo el supuesto de que se tienen construidas las matrices locales $\underline{\underline{A}}^i_{II}, \underline{\underline{A}}^i_{I\pi}, \underline{\underline{A}}^i_{I\Delta}, \underline{\underline{A}}^i_{\pi\pi}, \underline{\underline{A}}^i_{\pi\Delta}$ y $\underline{\underline{A}}^i_{\Delta\Delta}$ en cada uno de los subdominios de la partición. Entonces, para resolver $\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1} \underline{u}$, donde $\underline{u} \in W_\Pi$, se puede reescribir como

$$\begin{pmatrix} \underline{w}_I \\ \underline{w}_\pi \end{pmatrix} = \begin{pmatrix} \underline{\underline{A}}_{II} & \underline{\underline{A}}_{I\pi} \\ \underline{\underline{A}}_{\pi I} & \underline{\underline{A}}_{\pi\pi} \end{pmatrix}^{-1} \begin{pmatrix} \underline{u}_I \\ \underline{u}_\pi \end{pmatrix} \tag{4.124}$$

i.e. se necesita resolver $\underline{\underline{A}}_{\Pi\Pi}\underline{w} = \underline{u}$,la cual se expresa como

$$\begin{pmatrix} \underline{\underline{A}}_{II} & \underline{\underline{A}}_{I\pi} \\ \underline{\underline{A}}_{\pi I} & \underline{\underline{A}}_{\pi\pi} \end{pmatrix} \begin{pmatrix} \underline{w}_I \\ \underline{w}_\pi \end{pmatrix} = \begin{pmatrix} \underline{u}_I \\ \underline{u}_\pi \end{pmatrix} \tag{4.125}$$

entonces, $\underline{w}_\pi \in W_\Pi$ es solución de

$$\left(\underline{\underline{A}}_{\pi\pi} - \underline{\underline{A}}_{\pi I}\left(\underline{\underline{A}}_{II}\right)^{-1}\underline{\underline{A}}_{I\pi}\right)\underline{w}_\pi \equiv \underline{\underline{S}}_\pi \underline{w}_\pi = \underline{u}_\pi - \underline{\underline{A}}_{\pi I}\left(\underline{\underline{A}}_{II}\right)^{-1}\underline{u}_I \tag{4.126}$$

mientras

$$\underline{w}_I = \left(\underline{\underline{A}}_{II}\right)^{-1}\left(\underline{u}_I - \underline{\underline{A}}_{I\pi}\underline{w}_\pi\right) \tag{4.127}$$

donde

$$\left(\underline{\underline{A}}_{II}\right)^{-1} = \sum_{i=1}^{N}\left(\underline{\underline{A}}^i_{II}\right)^{-1}. \tag{4.128}$$

Por último, para evaluar $\underline{\underline{S}}\underline{v}$ se aplica el procedimiento similar al detallado en la sección (D.2) y para evaluar $\underline{\underline{S}}^{-1}\underline{v}$ se aplica el procedimiento detallado en la sección (D.3).

De las evaluaciones indicadas en esta sección, una gran parte de ellas es posible realizar en paralelo, y como se mostrará más tarde, la granularidad[21] paralela es gruesa, la cual es ideal para implementarse en equipos de cómputo paralelos como los Clusters, esto se demuestra mediante ejemplos en el capítulo de Análisis de Rendimiento (véase [59], [62] y [61]).

[21] La granularidad de un conjunto de tareas paralelas es la cantidad de trabajo que se puede hacer de forma independiente de otros cálculos.

5 Implementación Computacional Secuencial y Paralela de DDM

A partir de los modelos matemáticos (capítulo 2 y 3) y los modelos numéricos (capítulos 4, 5, y 6), 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 (capítulo 7).

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.

5.1 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:

$$\begin{aligned} -\nabla^2 u &= f_\Omega \text{ en } \Omega \\ u &= g_{\partial\Omega} \text{ en } \partial\Omega. \end{aligned} \tag{5.1}$$

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:

$$\Omega = [-1, 1] \times [0, 1] \tag{5.2}$$

donde

$$f_\Omega = 2n^2\pi^2 \sin(n\pi x) * \sin(n\pi y) \quad \text{y} \quad g_{\partial\Omega} = 0 \tag{5.3}$$

cuya solución es

$$u(x, y) = \sin(n\pi x) * \sin(n\pi y). \tag{5.4}$$

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:

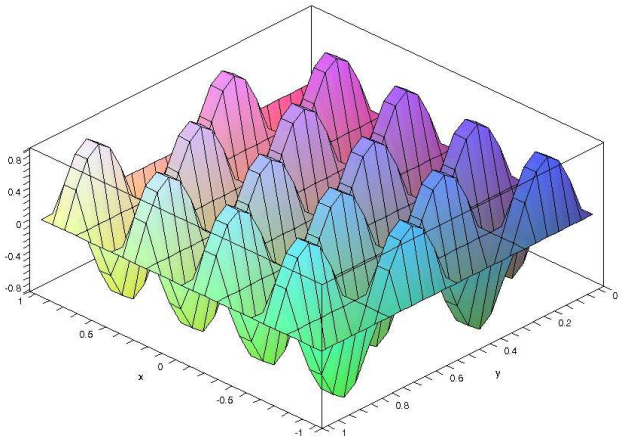

Figura 5: Solución a la ecuación de Poisson para n=4.

Hay que hacer notar que al implementar la solución numérica por el método del Diferencias Finitas, 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 Diferencias Finitas 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.

5.2 Método de Diferencias Finitas Secuencial

A partir de la formulación del método de Diferencias Finitas, la implementación computacional que se desarrolló tiene la jerarquía de clases siguiente:

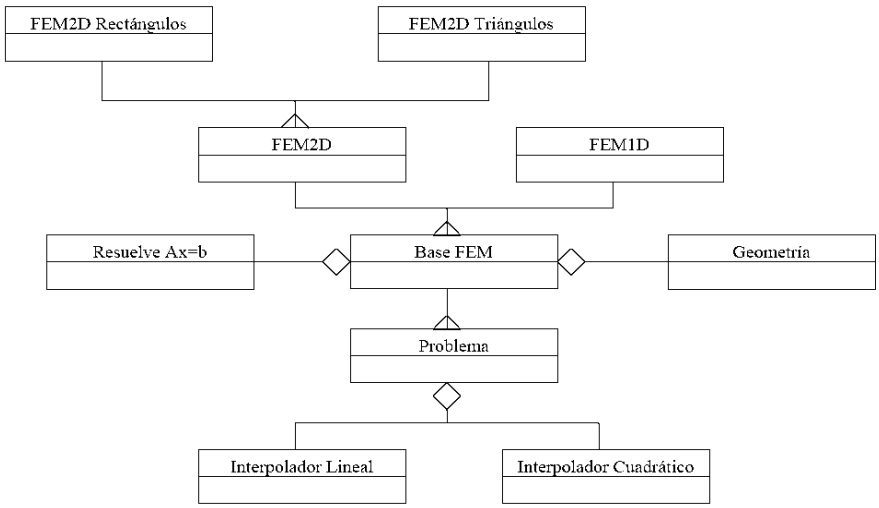

Figura 6: Jerarquía de clases para el método de elemento finito

Donde las clases participantes en *FEM2D Rectángulos* son:

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

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 menciono, el método de Diferencias Finitas o 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. (5.1) 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 Diferencias Finitas 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 Diferencias FInitas 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 Diferencias Finitas, pero no es costeable en rendimiento su implementación.

Finalmente, para el método de Diferencias Finitas 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 operaciones 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 Diferencias Finitas.

5.3 Método de Subestructuración Secuencial

A partir de la formulación del método de subestructuración visto en la sección (4.3) 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

$$\left[\sum_{i=1}^{E} \underline{\underline{S}}_i\right] \underline{u_\Sigma} = \left[\sum_{i=1}^{E} \underline{b}_i\right] \tag{5.5}$$

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

$$\underline{\underline{S}}\,\underline{u_\Sigma} = \underline{b} \tag{5.6}$$

es resuelto usando el método de gradiente conjugado visto en la sección (A.2.1), 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.

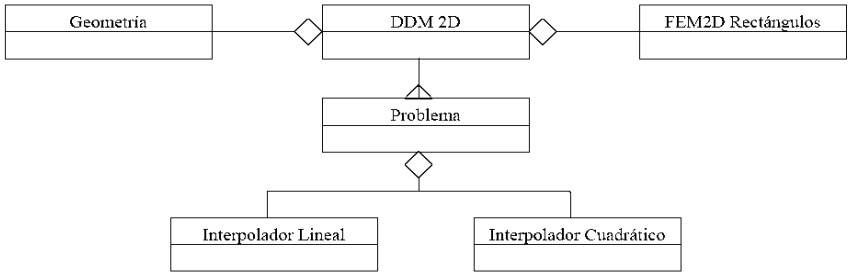

Figura 7: Jerarquía de clases para el método de subestructuración secuencial

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:

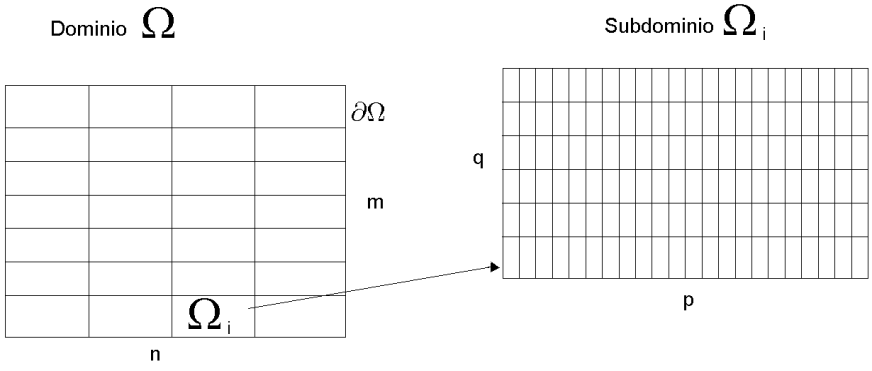

Figura 8: Descomposición del dominio Ω en $E = n \times m$ subdominios y cada subdominio Ω_i en $p \times q$ subdominios

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. (7.1), 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 Diferencias Finitas 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.

5.4 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 del 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 esta principalmente en las actividades 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.

Para resolver la Ec. (5.1), 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:

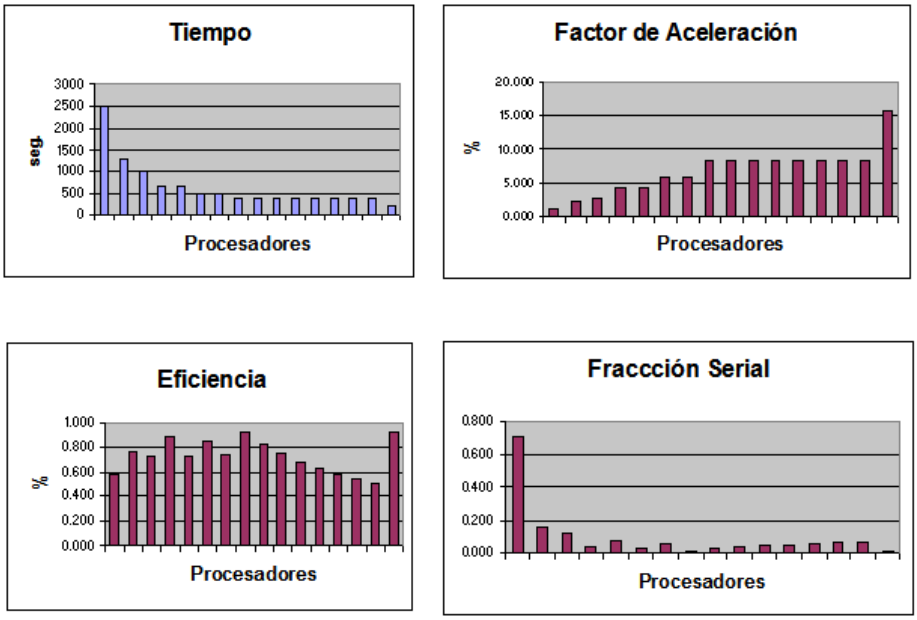

Figura 9: Métricas de desempeño de 2 a 17 procesadores

Primeramente notemos que existe mal balanceo 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:

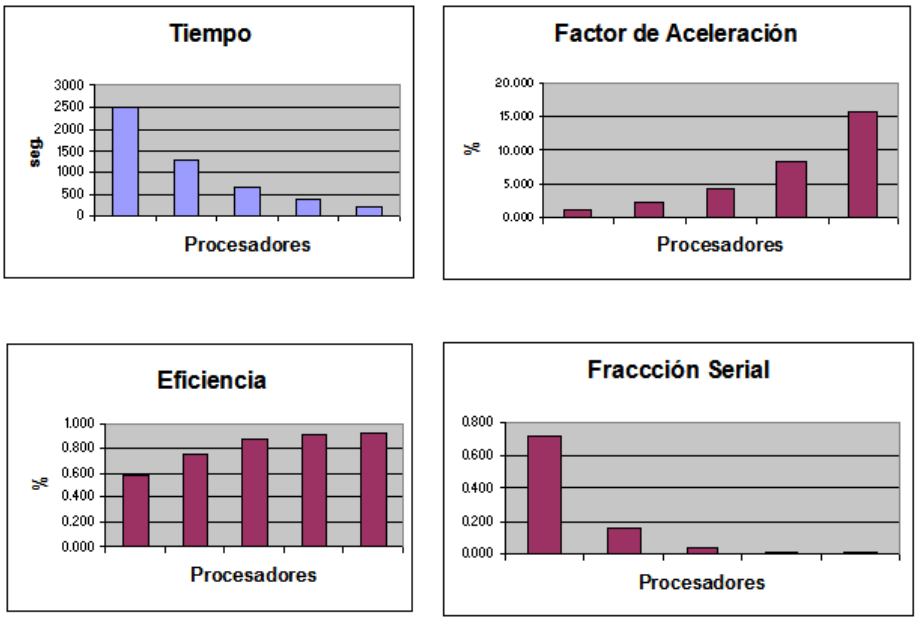

Figura 10: Métricas de desempeño mostrando sólo cuando las cargas están bien balanceadas (2, 3, 5, 9 y 17 procesadores).

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.

5.5 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 (A.2.1) 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 li-neal algebraico. Es por ello que usaremos los precondicionadores a priori. 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 vasta.

La implementación de los métodos a priori, requieren de más trabajo tanto en la face 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. (5.1) 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:

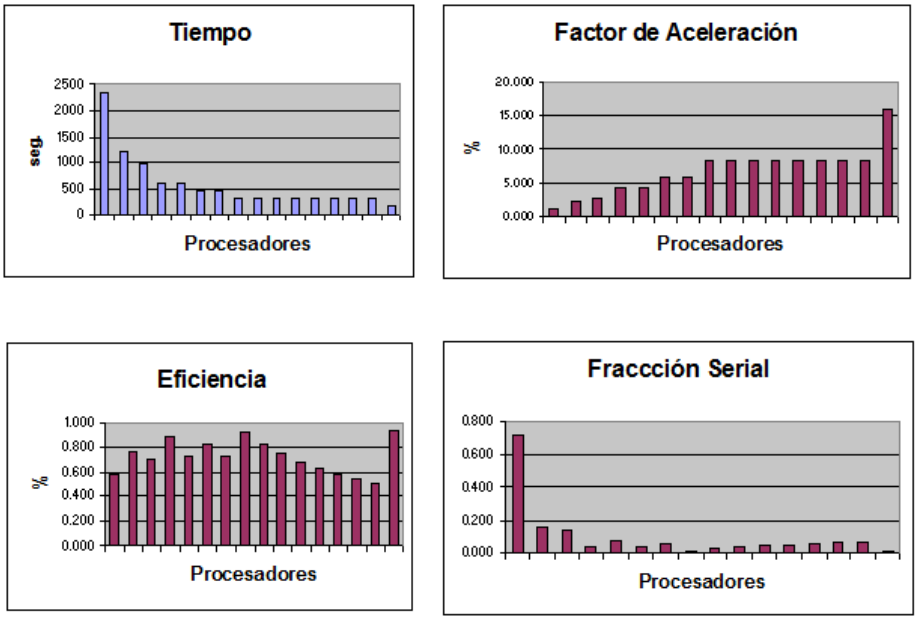

Figura 11: Métricas de desempeño de 2 a 17 procesadores

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 si 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:

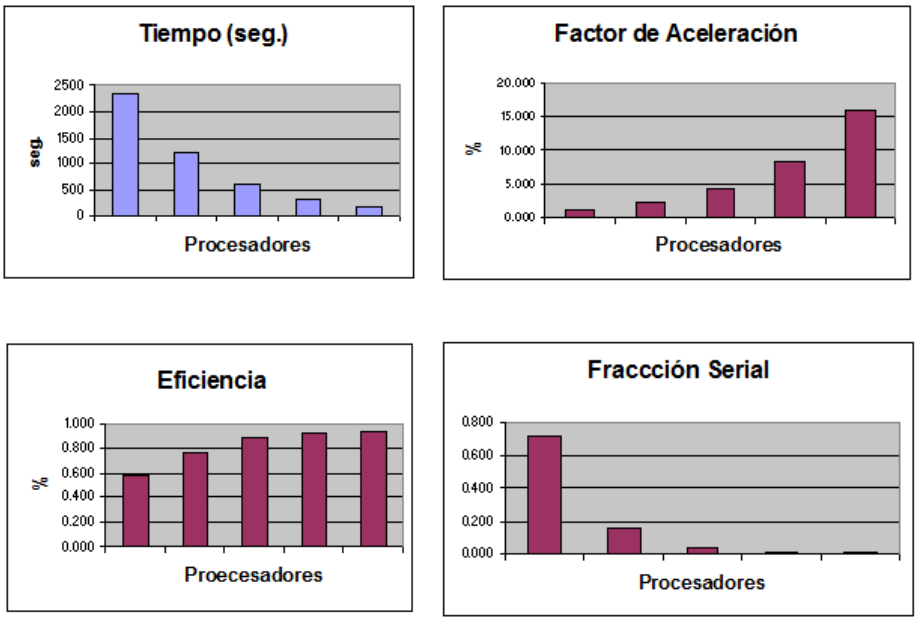

Figura 12: Métricas de desempeño mostrando sólo cuando las cargas están bien balanceadas (2, 3, 5, 9 y 17 procesadores).

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

6 Análisis de Rendimiento y Conclusiones para el método DDM

Uno de los grandes retos del área de cómputo científico es poder analizar a priori una serie 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 si, 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.

6.1 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 trasmite 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 (5.4), 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 trasmitidas 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 información.

F) A lo más $2 * p * q$ coordenadas son transmitidas a los objetos *FEM2D Rectángulos* en los nodos esclavos y estos 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.

6.2 Afectación del Rendimiento al Aumentar el Número de Subdominios en 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.

6.3 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. (5.1), 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 balanceo 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 balanceo 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 dependerán 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 si.

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 domino es que los subdominios son en principio independientes entre si 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 maestroesclavo.

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 domino 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.

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.

6.4 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 especificas para ciertos procesadores (Intel compiler para 32 y 64 bits) que pueden mejorar aun 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 si, 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 Diferencias Finitas y se pa-raleliza 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 aun 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 Diferencias Finitas 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 bibliotecas 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.

6.5 Conclusiones

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.

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.

7 Implementación Computacional Secuencial y Paralela de DVS

La implementación computacional de los métodos de descomposición de dominio sin traslape en general (véase [9], [4], [5] y [6]) y de los métodos de descomposición de dominio en el espacio de vectores derivados (DVS) en particular (véase [62] y [61]), en los cuales cada subdominio genera sus matrices locales y el método que resuelve el sistema global virtual —CGM o GMRES— es tal que necesita sólo una porción de la información que generan los subdominios, queda fácilmente estructurado mediante el esquema Maestro-Esclavo, tanto para la implementación del código secuencial como paralela.

El esquema Maestro-Esclavo parece ser un forma óptima[22] de dividir la carga computacional requerida para solucionar un problema de descomposición de dominio sin traslapes, en el cual, uno o más subdominios son asignados a un nodo esclavo, tanto en su implementación secuencial —donde cada nodo esclavo es un objeto— como en su implementación paralela —donde cada nodo esclavo esta asignado a un procesador—, en el cual el nodo maestro de forma síncrona controla las tareas que requiere el esquema DVS, las cuales son llevadas a cabo por los nodos esclavos, donde la comunicación sólo se da entre el nodo maestro y cada nodo esclavo —no existiendo comunicación entre los nodos esclavos—, optimizando así las comunicaciones.

7.1 Esquema Maestro-Esclavo como una Forma de Implementación

El esquema Maestro-Esclavo permite que en los nodos esclavos se definan uno o más subdominios —en los cuales se generen y manipulen las matrices locales de cada subdominio— y que el maestro controle las actividades necesarias para implementar cualquiera de los métodos desarrollados. En particular la implementación del método de resolución del sistema lineal virtual $\underline{\underline{M}u}_{\Delta} = \underline{b}$ esquematizados por los algoritmos descritos en la Ec.(4.116 ó 4.118), donde el nodo maestro controlará a cada uno de sus nodos esclavos mediante comunicaciones entre este y los esclavos, pero no entre esclavos, como se muestra en la figura.

Esta forma de descomponer el problema dentro del esquema Maestro-Esclavo, permite hacer la implementación del código tanto para la parte secuencial como su paralelización de manera fácil y eficientemente, donde tomando en cuenta la implementación en estrella del Cluster o equipo multiCore, el modelo de paralelismo de MPI y las necesidades propias de comunicación del programa, el nodo maestro tendrá comunicación sólo con cada nodo esclavo, esto reducirá las comunicaciones y optimizará el paso de mensajes (véase [17], [15] y [16]).

[22] El esquema Maestro-Esclavo esta intrínseco a la definición de los métodos de descomposición de dominio tipo subestructuración, ya que las tareas implementadas por los subdominios son pensados como procesos esclavos los cuales son controlados por el maestro que implementa la solución de los nodos de frontera interior (véase [62] y [61]).

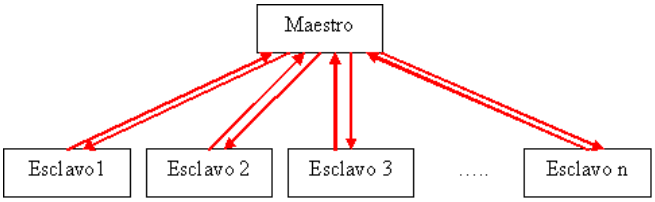

Figura 13: Esquema del Maestro-Esclavo

Además el esquema de paralelización Maestro-Esclavo permite sincronizar fácilmente por parte del nodo maestro las tareas que realizan en paralelo los nodos esclavos, éste modelo puede ser explotado de manera eficiente si existe poca comunicación entre el maestro y los esclavos; 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 siendo usados de forma eficiente y existirán pocos tiempos muertos, aumentando así la eficiencia global de la implementación de los métodos de descomposición de dominio en el espacio de vectores derivados bajo el esquema Maestro-Esclavo.

7.2 Análisis, Diseño y Programación Orientada a Objetos

Desde el inicio del proyecto para la implementación computacional de los métodos de descomposición de dominio en el espacio de vectores derivados se planteó la necesidad de que el código desarrollado fuera orientado a objetos, que su implementación computacional debería de correr en equipos secuenciales y paralelos para dominios en dos y tres dimensiones. Por ello se optó por usar el lenguaje de programación C++ y la paralelización se haría usando la biblioteca de paso de mensajes MPI.

Dentro de las consideraciones básicas en el análisis orientado a objetos es que el código debería de correr tanto en equipos secuenciales como en paralelos, con un mínimo de cambios y que la interdependencia de la parte paralela no debería afectar la parte secuencial. Para que cualquier cambio en el código de los métodos desarrollados no requiera grandes cambios en el código paralelo. Esto se logra mediante la programación orientada a objetos haciendo uso de clases abstractas[23] o contenedores.

Esto permitió desarrollar un código que fuera robusto y flexible, además de que la escritura, depuración y optimización se hace desde cualquier Notebook y su ejecución puede ser hecha en cualquier computadora personal o Clusters sin ningún cambio en el código.

[23] En general las clases abstractas que definen comportamientos virtuales pueden no ser eficientes si son llamadas una gran cantidad de veces durante la ejecución del programa. Para el caso del esquema DVS, en el cual se usa CGM o GMRES para resolver el sistema lineal virtual, este sólo se llama una sola vez; y el proceso de solución del sistema lineal asociado consume la mayoría del tiempo de ejecución, por eso se considera eficiente.

Por ejemplo, en el uso de los métodos numéricos tanto directos como iterativos para resolver sistemas lineales en los cuales la matriz es real —existe como tal— o es virtual —esta dispersa por los distintos subdominios— se creo una jerarquía de clases que implementa mediante herencia a la clase abstracta, la cual usan los algoritmos que requerían solucionar un sistema lineal, esta clase abstracta se llama Solvable —véase apéndice A—. La jerarquía[24] de clases mostrada en la figura (14) permite contener a cualquiera de los métodos numéricos de solución de sistemas lineales actuales y cualquier implementación futura y es independiente de si se usa para generar código secuencial o paralelo.

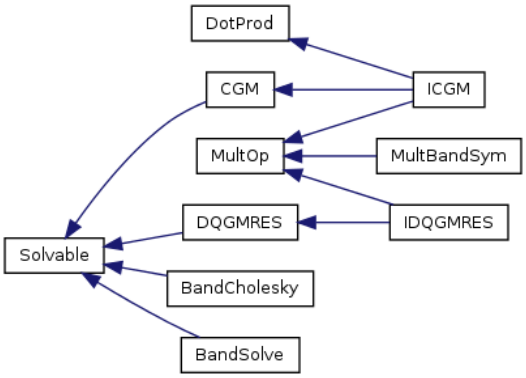

Figura 14: Jerarquía de clases para la implementación de los métodos de resolución de sistemas lineales.

Nótese que, en general, 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, además de hacer el código extensible y reutilizable. Esto tiene especial interés cuando se piensa en la cantidad de meses invertidos en la programación comparada con los segundos consumidos en la ejecución del mismo.

7.2.1 Implementación Secuencial en C++

Usando la filosofía del manejo de clases abstractas desde el análisis y durante el diseño de la implementación computacional de los ocho métodos de descomposición de dominio en el espacio de vectores derivados, se pensó en usar una jerarquía de clases que especializarían a una clase abstracta llamada DPMethod, la cual permite implementar uno o más de los métodos de descomposición de do-

[24] Las jerarquías de clases de herencia mostradas en las figuras fueron generadas usando Doxygen documentation (véase [66]) a partir del código fuente en C++.

minio desarrollados; y dada una ecuación o sistemas de ecuaciones diferenciales parciales, se usaría el método iterativo —Gradiente Conjugado o el método Residual Mínimo Generalizado o cualquier otro— dependiendo de que la matriz global virtual fueran simétrica o no simétrica; su jerarquía de clases se muestra en la figura (15).

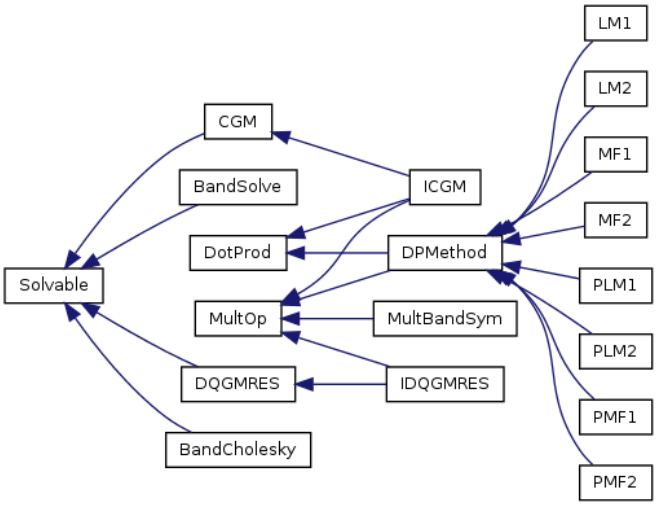

Figura 15: Jerarquía de clases para la implementación secuencial

De esta forma, es posible implementar uno o más de los algoritmos desarrollados de descomposición de dominio en el espacio de vectores derivados sin realizar cambios en la base del código, permitiendo especializar el código para alguna necesidad particular sin cargar con código no requerido en la resolución de un problema específico, pero en caso de evaluar el desempeño de cada uno de los métodos ante un problema determinado, se pueda realizar sin afectación del código.

Además de la flexibilidad anteriormente comentada, también se reutiliza la jerarquía de clases para la resolución de sistemas lineales, permitiendo que cualquier cambio o refinamiento a estas clases redunde en el desempeño global del sistema, permitiendo que en un futuros se agreguen y refinen métodos que manejen con eficiencia la solución de los sistemas lineales asociados al método de descomposición de dominio DVS.

7.2.2 Implementación Paralela en C++ Usando MPI

Para poder intercomunicar al nodo maestro con cada uno de los nodos esclavos se usa la interfaz de paso de mensajes —Message Passing Interface (MPI)—, una biblioteca de comunicación para procesamiento en paralelo. 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 la 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. 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 mediante MPI para equipos con uno o más Cores, 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.

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.

  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 etq de destino des asociado con la comunicación com.

- Para el que recibe, debe de tener la dirección de memoria donde se pondrán los datos recibidos, junto con la dirección del proceso del que los envío.

  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 etq de la fuente $fuent$ asociado con la comunicación com y el estado st.

El conjunto básico de directivas (en este caso sólo se usan estas) en C++ de MPI son:

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

La estructura básica del programa bajo el esquema Maestro-Esclavo codificada en C++ y usando MPI 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 Maestro
      } else {
          // Operaciones del esclavo con identificador ME_id
      }
      MPI::Finalize();
}
```

En este único programa se deberá de codificar todas las tareas necesarias para el nodo maestro y cada uno de los nodos esclavos, así como las formas de intercomunicación entre ellos usando como distintivo de los distintos procesos a la variable *ME_id* (véase [15] y [16]).

La jerarquía de clases del esquema Maestro-Esclavo en su implementación paralela permite repartir la carga de varias maneras en uno o más Cores. Reutilizando toda la jerarquía de clases de la implementación secuencial de los algoritmos DVS y sólo es necesario agregar clase que especializa algunos comportamientos que requieren hacer uso de las comunicaciones, mediante la biblioteca de paso de mensajes MPI. La jerarquía de clases es mostrada en la figura siguiente:

La reutilización de toda la jerarquía de clases generada para la implementación secuencial permite que el código paralelo soporte una gran cantidad de cambios sin afectación a la implementación paralela, teniendo así, un código robusto, flexible, modular y de fácil mantenimiento (véase [17]).

7.3 Alcances y Limitaciones del Esquema Maestro-Esclavo

El esquema Maestro-Esclavo es eficiente cuando se tiene una carga casi homogénea en cada nodo esclavo y se manejan una cantidad moderada de ellos. Un factor limitante en el esquema Maestro-Esclavo, es que el nodo maestro deberá de atender todas las peticiones hechas por todos y 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.

Una opción para optimizar el esquema Maestro-Esclavo es contar con un nodo maestro lo suficientemente poderoso para atender simultáneamente la mayor cantidad de las tareas síncronas del método de descomposición de dominio en el menor tiempo posible. Pero los factores limitantes del esquema Maestro-Esclavo son de tres tipos, a saber:

1. Los inherentes al método de descomposición de dominio.

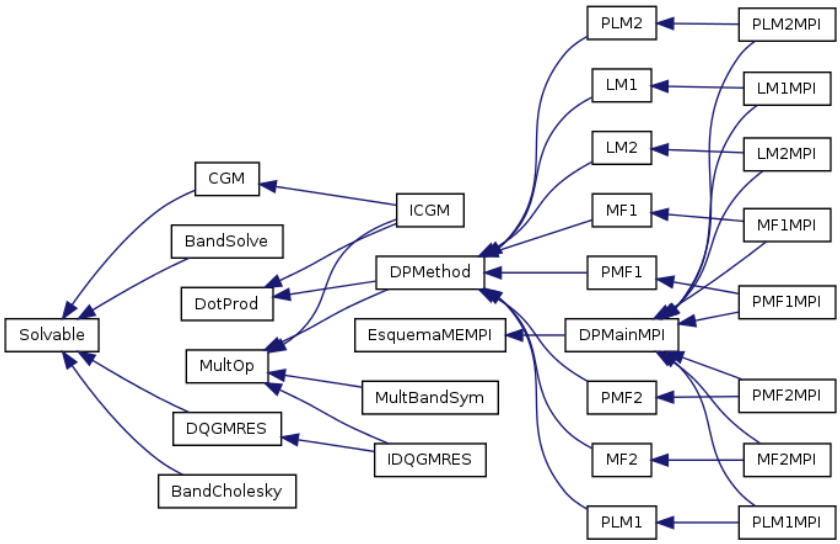

Figura 16: Jerarquía de clases para la implementación paralela rehusando toda la jerarquía de la implementación secuencial y de resolución de sistemas lineales

2. Los inherentes al propio esquema Maestro-Esclavo.
3. Los inherentes al equipo de cómputo en paralelo en el que se ejecute el programa.

En el primer caso, en cuanto a los inherentes al método de descomposición de dominio destacan:

- El método de descomposición de dominio es síncrono, es decir, si un nodo esclavo acaba la tarea asignada y avisa al nodo maestro, este no podrá asignarle otra tarea hasta que todos los nodos esclavos concluyan la suya, y se realicen las operaciones necesarias para asignar las nuevas tareas a los nodos esclavos.
- El nodo maestro sólo realiza tareas de control y sincronización pero no conoce o realiza cálculos relacionados con los sistemas lineales locales a cada uno de los subdominios que están asignados a los nodos esclavos.
- Por lo anterior, el esquema Maestro-Esclavo no es eficiente si sólo se usan dos procesos o Cores —uno para el nodo maestro y otro para el nodo esclavo—, por otro lado, cuando se realiza el análisis de rendimiento en P Cores, hay que tomar en cuenta que los únicos nodos que manipulan los sistemas lineales asociados al método de descomposición de dominio son los esclavos $(P-1)$ y el nodo maestro sólo realiza el control y sincronización de las tareas del los métodos DVS.

En el segundo caso, en cuanto a los inherentes al propio esquema Maestro-Esclavo destacan:

- El nodo maestro deberá distribuir las tareas a los nodos esclavos acorde al número de subdominios existentes en la descomposición y la malla fina de cada subdominio, de tal forma que cada nodo esclavo tenga una carga computacional equivalente a los demás nodos esclavos.

- En el caso de una carga homogénea en cada subdominio, si se usan P Cores en el equipo paralelo y la descomposición del dominio tiene E subdominios, tal que $(P-1) \nmid E$, esa descomposición de dominio no es adecuada para trabajar en dicha cantidad de Cores. En este caso, el número de procesadores P que se usen para tener buen balance de cargas es conocido a priori cuando el dominio Ω se descompone en $n \times m$ —$n \times m \times o$— subdominios homogéneos, entonces se generarán $E = n*m$ —$E = n*m*o$— subdominios Ω_α, teniendo un buen balanceo de cargas si $(P-1) \mid E$.

- Pese al buen balanceo de la carga en los nodos esclavos, es común que, un gran número de nodos esclavos envíen simultáneamente datos al nodo maestro saturando su canal de comunicación; y este en algún momento tendrá que tratar atender las múltiples comunicaciones, degradando su rendimiento al aumentar el número de nodos esclavos involucrados en la descomposición.

En el caso de generar desbalance de la carga en los nodos esclavos o una saturación de comunicaciones en el nodo maestro, se propicia a que algunos procesadores terminen antes que otros, generando tiempos muertos de ejecución en dichos Cores; propiciando una notoria degradación en la eficiencia global en el procesamiento, es por esto que, en algunos casos al aumentar el número de procesadores no se aprecia una disminución sustancial del tiempo de ejecución y en casos extremos puede ocasionar un aumento en el tiempo.

En el tercer caso, en cuanto a los inherentes al equipo de cómputo en paralelo en el que se ejecute el programa destacan:

- El programa se diseño para correr en cualquier cantidad de procesos o Cores y no hay límite establecido en cuanto al número de subdominios que soporta el programa, pero el equipo en el que se ejecute tiene un número predeterminado de Cores y cada uno de ellos tiene asignado una cantidad limitada de RAM, es por ello que, las dimensiones del problema que es posible correr en un equipo paralelo dado esta determinado por estas limitantes.

- En los equipos paralelos, el cuello de botella en cuanto a la eficiencia global de la ejecución, lo presentan las comunicaciones, entre más comunicaciones necesite el programa, es imperante el contar con una infraestructura que permita la mejor velocidad de comunicaciones entre el nodo maestro y los nodos esclavos; además de que esta cuente con la menor latencia posible

en las comunicaciones. Por otro lado, el acceso al disco duro es mínimo y no representa un costo significativo en las comunicaciones totales de la ejecución.

Para ejemplificar lo discutido anteriormente, se considera como modelo matemático el problema de valor en la frontera (BVP) asociado con la ecuación de Poisson, con condiciones de frontera Dirichlet, definido en Ω como:

$$\begin{aligned} -\nabla^2 u &= f_\Omega \text{ en } \Omega \\ u &= g_{\partial\Omega} \text{ en } \partial\Omega \end{aligned} \tag{7.1}$$

este ejemplo, 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 usado 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 se considera el problema con Ω definido en:

$$\Omega = [-1, -1] \times [1, 1] \tag{7.2}$$

donde

$$f_\Omega = 2n^2\pi^2 \sin(n\pi x) * \sin(n\pi y) \quad \text{y} \quad g_{\partial\Omega} = 0 \tag{7.3}$$

cuya solución es

$$u(x, y) = \sin(n\pi x) * \sin(n\pi y) \tag{7.4}$$

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:

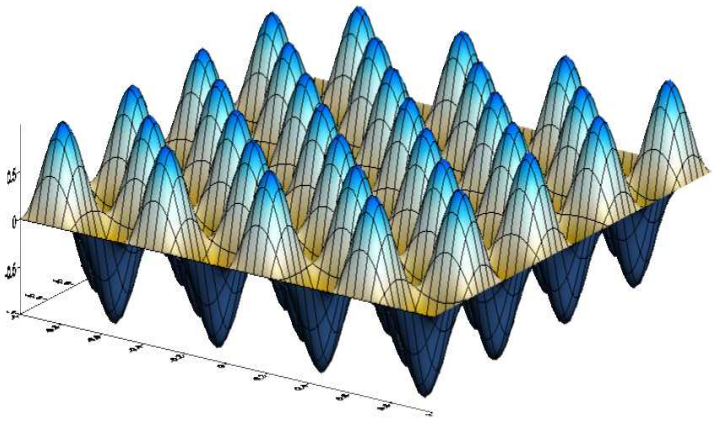

Figura 17: Solución a la ecuación de Poisson para n=4.

Para las pruebas de rendimiento[25] en las cuales se evalúa el desempeño de los programas realizados se usa $n = 100$, pero es posible hacerlo con $n \in \mathbb{N}$ grande.

Supóngase que se desea resolver el dominio Ω usando 1024×1024 nodos —$1,048,576$ grados de libertad— mediante el algoritmo precondicionado PRIMAL#2 , de manera inmediata surgen las siguientes preguntas: ¿cuáles son las posibles descomposiciones posibles? y ¿en cuántos procesadores se pueden resolver cada descomposición?. Para este ejemplo en particular, sin hacer la tabla exhaustiva, se tiene:

Partición	Subdominios	Procesadores
2x2 y 512x512	4	2,3,5
4x4 y 256x256	16	2,3,5,9,17
8x8 y 128x128	64	2,3,5,9,17,33,65
16x16 y 64x64	256	2,3,5,9,17,33,65,129,257
32x32 y 32x32	1024	2,3,5,9,17,33,65,129,..,1025
64x64 y 16x16	4096	2,3,5,9,17,33,65,129,...,4097
128x128 y 8x8	16384	2,3,5,9,17,33,65,129,...,16385
256x256 y 4x4	65536	2,3,5,9,17,33,65,129,...,65537
512x512 y 2x2	262144	2,3,5,9,17,33,65,129,...,262145

De esta tabla es posible seleccionar las descomposiciones que se adecuen a las necesidades particulares del equipo paralelo con que se cuente, para evaluar el tiempo de ejecución de este ejemplo se usó la PC Antipolis Intel Xeon a 2.33 GHtz de 64 bits con 8 Cores y 32 GB de RAM, obteniendo los siguientes resultados para una tolerancia de 10^{-6} usando norma infinita en todos los casos (tiempo en segundos):

	1 Core	2 Cores	3 Cores	4 Cores	5 Cores	6 Cores	7 Cores	8 Cores
Partición	Tiempo	Tiempo	Tiempo	Tiempo	Tiempo	Tiempo	Tiempo	Tiempo
2x2 y 512x512	16465	10659	7207	7105	4641			
4x4 y 256x256	2251	5063	2252	2103	1643	1233	1068	947
8x8 y 128x128	855	885	482	395	314	311	283	272
16x16 y 64x64	321	348	190	149	121	125	118	117
32x32 y 32x32	26	39	26	24	23	21	21	21
64x64 y 16x16	205	595	485	477	481	461	469	469
128x128 y 8x8	1026	5453	5352	5431	5633	5843	5843	5903
256x256 y 4x4	8544	26167	25892	25902	25939	25950	25969	26003
512x512 y 2x2	34845	64230	63293	63308	63389	63475	63502	63693

[25] En todos los ejemplos del presente trabajo no se realiza una análisis de comunicación de forma independiente al tiempo de cálculo, ya que los Clusters a los que se obtuvo acceso carecen de herramientas que permitan realizar dicho análisis, pero si se realizaron algunas pruebas con XMPI (véase [67]) y Vampir (véase [68]) para algunos ejemplos representativos en los cuales se muestra que se tiene granularidad gruesa (véase [62]).

De estos resultados, se desprende que:

1. Dependiendo del tamaño de la malla gruesa —número de subdominios a trabajar— y de la malla fina, es siempre posible encontrar una descomposición de dominio —32×32 y 32×32— en que el tiempo de cálculo sea mínimo:
   - Al usar un solo Core (programa secuencial 26 seg.).
   - Al usar múltiples Cores interconectados mediante MPI (6 Cores en 21 seg.).
2. Es notorio el efecto que genera el mal balanceo de carga[26], el cual se refleja en que no disminuye el tiempo de ejecución al aumentar el número de procesadores y en algunos casos el tiempo aumenta conforme se agregan más Cores.

En contraste con los 110 segundos en que se resolvió el mismo problema usando los métodos de Elemento Finito y Diferencias Finitas, usando en ambos casos Factorización Cholesky para resolver el sistema lineal asociado.

7.4 Afectación del Rendimiento al Refinar la Descomposición

Una parte fundamental al trabajar con problemas reales usando una descomposición fina es conocer a priori que factores afectan el rendimiento de la aplicación ante las posibles elecciones en la descomposición de dominio, la afectación se da por:

1. En el caso de contar con un gran número de subdominios que estén asignados a distintos nodos esclavos, la afectación se da por la saturación del nodo maestro con una gran cantidad de comunicaciones simultáneas por parte de los nodos esclavos que el nodo maestro deberá de atender y la velocidad de comunicación del canal usado para ello. Esto es especialmente importante en la implementación paralela en la cual la interconexión del equipo paralelo se hace mediante un canal de comunicación lento u ocupado por otros procesos.
2. En el caso de realizar una descomposición muy fina en cada subdominio, la afectación del rendimiento se da al aumentar el número de nodos involucrados en el complemento de Schur local $\underline{\underline{S}}^i$, ya que esto significa, por un lado generar matrices locales más grandes

$$\underline{\underline{A}}^{\alpha}_{II}, \underline{\underline{A}}^{\alpha}_{I\pi}, \underline{\underline{A}}^{\alpha}_{I\Delta}, \underline{\underline{A}}^{\alpha}_{\pi I}, \underline{\underline{A}}^{\alpha}_{\pi\pi}, \underline{\underline{A}}^{\alpha}_{\pi\Delta}, \underline{\underline{A}}^{\alpha}_{\Delta I}, \underline{\underline{A}}^{\alpha}_{\Delta\pi} \text{ y } \underline{\underline{A}}^{\alpha}_{\Delta\Delta} \tag{7.5}$$

[26] Por ejemplo, al usar una descomposición gruesa de $64 \times 64 = 4096$ subdominios repartidos en $3, 5, 6, 7$ nodos esclavos.

además de resolver el sistema $y = \left(\underline{\underline{A}}^i_{II}\right)^{-1} x$ de alguna forma. Si el número de nodos interiores en el subdominio es grande entonces solucionar el complemento de Schur local será costoso computacionalmente.

Para el primer caso, el uso de algunos cientos o miles de subdominios no afectan de manera considerable el desempeño del Esquema Maestro-Esclavo si la red es relativamente rápida (de un Gigabit por segundo o más), y como los avances en las comunicaciones son vertiginosos, en un corto tiempo se tendrá acceso a redes de mayor velocidad reduciendo el efecto de manipular un gran número de subdominios simultáneamente.

Para el segundo caso, al resolver el complemento de Schur local, se puede emplear diversos métodos de solución, la selección del método más adecuado al problema en particular depende por un lado de las capacidades computacionales del equipo donde se implemente la ejecución y las características propias de los sistemas lineales asociados al problema. Así, para solucionar el sistema $y = \left(\underline{\underline{A}}^i_{II}\right)^{-1} x$ correspondiente al complemento de Schur local $\underline{\underline{S}}^i$ se puede usar por ejemplo: Factorización LU, Factorización Cholesky, Gradiente Conjugado o alguna variante de GMRES, pero deberá de usarse aquel método 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 del la matriz $\underline{\underline{A}}^i_{II}$ versus el método numérico usado para resolver el sistema lineal asociado —todas esas opciones están implementadas en el código y pueden seleccionarse conforme sean requeridos en la ejecución, mediante directivas de compilación—

Por lo visto en el ejemplo anterior, si el problema involucra una gran cantidad de nodos interiores y el equipo —secuencial o paralelo— en el que se implantará la ejecución del programa tiene una cantidad de memoria reducida, es recomendable en los procesos locales a los subdominios usar métodos iterativos —Gradiente Conjugado o alguna variante de GMRES—, estos consume una cantidad de memoria pequeña comparada con los métodos directos —Factorización LU o Cholesky— pero requieren una gran cantidad de iteraciones para obtener la misma precisión que los directos.

Hay que tomar en cuenta que 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 y disminución del tamaño de las matrices a generar por subdominio; y esto se refleja en una pérdida de eficiencia en el tiempo de ejecución, esto es generado por la gran cantidad de subdominios que es necesario crear y manejar por el nodo maestro, incrementando sustancialmente las comunicaciones y por otro lado, cada subdominio manejará cada vez matrices más pequeñas con el consecuente aumento de los tiempos muertos, al invertir mucho más tiempo en comunicaciones que en cálculos.

Para mitigar los factores limitantes inherente al propio esquema Maestro-Esclavo, es posible implementar algunas operaciones del nodo maestro en paralelo, usando uno o más Cores distintos a los asignados a los nodos esclavos. Para la parte inherente al método de descomposición de dominio, la parte medular la da el balanceo de cargas. Es decir, cada nodo esclavo debe tener una carga de trabajo equivalente al resto de los nodos.

Tomando en cuenta lo discutido, para un problema particular y la descomposición del dominio Ω en la implementación paralela, hay que tomar en cuenta lo siguiente:

- Buscar que la descomposición de malla gruesa y su asociada malla fina, en la que cada nodo esclavo —asociado a un procesador— tenga una carga casi homogénea con respecto a los demás nodos esclavos, .i.e. buscar que en su conjunto, todos los subdominios Ω_α de la malla gruesa y su descomposición fina de cada uno de ellos, que estén asignados a cada nodo esclavo sean computacionalmente equivalentes.

- Elegir el método numérico local a cada subdominio para garantizar el uso de la menor cantidad de memoria posible y/o la mayor velocidad de ejecución versus la precisión global esperada del método de descomposición de dominio.

- Elegir de las distintas descomposiciones balanceadas del dominio Ω y las diferentes opciones de los métodos numéricos usados localmente en cada subdominio, aquella que presente el mejor rendimiento computacional acorde al equipo paralelo en el cual se implemente la solución del problema.

Nótese que el esquema Maestro-Esclavo paralelo lanza P procesos —uno para el nodo maestro y $P-1$ para los nodos esclavos—, 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 relativamente pequeñas —del orden de miles o millones de nodos— y cuando este listo para producción se puede mandar a cualquier equipo paralelo sin cambio alguno en el código.

7.5 Opciones para Soportar una Descomposición Fina del Dominio

Supóngase ahora que se necesita resolver el problema de una descomposición fina del dominio Ω, sin pérdida de generalidad, se puede suponer por ejemplo, que se usa una malla de 8192×8192 nodos, este tipo de problemas es común y 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. Si se supone que una descomposición de 100×100 nodos en un subdominio consume 1 GB de RAM y que 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 4096×4096	4096×4096	≈40.0 GB
257	16×16 y 512×512	512×512	≈5.0 GB
1025	32×32 y 256×256	256×256	≈2.5 GB
4097	64×64 y 128×128	128×128	≈1.2 GB

Nótese 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 fijo menor al sugerido y donde cada uno de ellos tiene solo memoria suficiente para soportar uno o más subdominios?, la respuesta es si.

Primero, nótese que al considerar una descomposición fina del tipo 64×64 y 128×128 se requiere aproximadamente 1.2 GB de RAM por Core, si además se supone que sólo se tienen unos cuantos procesadores con poca memoria —por ejemplo 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 domino es que los subdominios son en principio independientes entre si y que sólo están acoplados a través de la solución en la interfase de los subdominios que es desconocida.

Como sólo se requiere tener en memoria la información de la frontera interior, es posible bajar a disco duro todas las matrices y datos complementarios generados en cada subdominio —que consumen el 99% de la memoria del objeto *RectSub*—, 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 reducida 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_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 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 da una buena idea de las limitantes que existen en la resolución de problemas con dominios que tienen una descomposición fina y pone de manifiesto las características mínimas necesarias del equipo paralelo para soportar dicha implantación.

7.6 Otras Opciones de Paralelización

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 ya común contar con las cada vez más omnipresentes tarjetas NVIDIA, con los cada vez más numerosos Cores CUDA —que una sola tarjeta NVIDIA TESLA puede tener del orden de cientos 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 [64] y [65]), 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 se le encargan a los numerosos Cores CUDA de las tarjetas NVIDIA.

Los métodos de descomposición de dominio sin traslape y en particular el esquema DVS con sus ocho algoritmos, pueden hacer 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.

Para integrar esta forma de paralelismo en los códigos, es necesario hacer cambios mínimos[27] al mismo, ya que sólo es necesario reimplementar los comportamientos locales que requieren otro tipo de paralelismo, ya que la jerarquía de clases del código desarrollado permite especializar los comportamientos que implementan las comunicaciones, esto queda de manifiesto al reutilizar toda la jerarquía de clases de la implementación secuencial en la implementación paralela como se aprecia en la figura (16).

[27] Ya se tiene una versión operacional del código en los cuales se han realizado algunas pruebas de rendimiento, pero los resultados son limitados por la complejidad de la programación de las tarjetas NVIDIA y la falta de herramientas y bibliotecas de código abierto que optimicen y depuren las implementaciones.

Además, el esquema Maestro-Esclavo sólo requiere enviar un vector a cada subdominio en cada paso de la iteración del sistema lineal virtual —mediante el paso de mensajes usando MPI— el cual se coloca en la RAM de la memoria compartida, después este es copiado[28] a la RAM de la tarjeta NVIDIA según el subdominio que se este trabajando —se controla usando OpenMP—, aquí los múltiples Cores CUDA sin salir de su RAM local efectuarían las operaciones de multiplicación de matriz vector necesarias para regresar un único vector a la RAM de la memoria compartida y de ahí se enviaría por MPI al nodo maestro, concluyendo la iteración.

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 RAM de la tarjeta NVIDIA, véase figura (18).

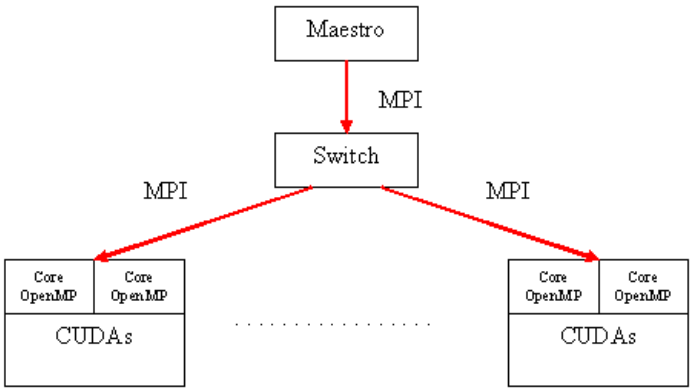

Figura 18: La intercomunicación de equipos con memoria compartida se realiza mediante MPI y la intercomunicación entre Cores que comparten la misma memoria se realiza con OpenMP, además las operaciones matriciales se le encargan a los numerosos Cores CUDA de las tarjetas NVIDIA.

De esta manera es posible adaptar el código para todos y cada uno de los métodos desarrollados, 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 podrán competir en un futuro con los grandes equipos de cómputo de alto desempeño.

[28] En tránsito de datos entre la RAM de la computadora y la RAM de los CUDAs, no es tan rápido como se requiere. Esto genera una baja de rendimiento considerable, que en ciertos problemas como los no lineales y con coeficientes variables es notorio.

8 Análisis de Rendimiento y Conclusiones para el método DVS

La uniformidad de las fórmulas presentadas en la sección (4.4.2) para los métodos de descomposición de dominio en el espacio de vectores derivados (DVS), nos han permitido implementar de forma eficiente dichos algoritmos en múltiples equipos secuenciales y paralelos, aprovechando el hecho que los algoritmos derivados son capaces de obtener *la solución global por la resolución de problemas locales exclusivamente.*

El grupo de ocho algoritmos desarrollados —de los cuales cuatro son precondicionados— a los que nos referiremos como los *algoritmos DVS* (véase [59], [61]), operan exclusivamente sobre los nodos primales y duales en la frontera interior y estos se implementan eficientemente mediante el uso de métodos ite-rativos — CGM para el caso simétrico o GMRES para el caso no simétrico— para resolver el sistema algebraico virtual asociado.

En esta sección, se mostrará —mediante los resultados de algunos experimentos numéricos conspicuos en Ciencias de la Tierra e Ingeniería— la eficiencia del esquema DVS (véase [62]), para ello; primero, se muestra el rendimiento en problemas simétrico y no simétricos —en dos y tres dimensiones—; segundo, se muestra el rendimiento en problemas indefinidos; tercero, se muestra el rendimiento en problemas de Advección-Difusión; después, se muestra el análisis de rendimiento en equipos paralelos hasta con 1024 Cores y por último se muestra lo que se consideran como criterios integrales para evaluar métodos de descomposición de dominio sin traslape y en especial al esquema DVS.

Para los experimentos numéricos reportados en esta sección, sólo se muestran los resultados de los cuatro métodos precondicionados del esquema DVS; en donde el dominio Ω fue discretizado usando una malla estructurada uniforme. Y en todos los casos, se tomaron como nodos primales a los vértices de los subdominios de la partición gruesa en dos dimensiones y a las aristas de los subdominios de la partición gruesa para tres dimensiones. Además, la tolerancia usada para concluir los métodos iterativos de resolución de sistemas lineal virtual asociado es en norma infinita.

8.1 Análisis de Rendimiento para Problemas Simétricos y no Simétricos

Para realizar el análisis de rendimiento, se usa la ecuación elíptica

$$-a\nabla^2 \underline{u} + \underline{b} \cdot \nabla \underline{u} + c\underline{u} = f \tag{8.1}$$

con condiciones de frontera Dirichlet cero, donde $a, c > 0$ son constantes, mientras que $\underline{b}$ es un vector constante de dimensión n. El dominio $\Omega \subset \mathbb{R}^n$ fue tomado con $n = 2, 3$ donde Ω es el cuadrado o cubo unitario según corresponda.

Las matrices generadas fueron obtenidas por discretización local en ambos casos —en dos y tres dimensiones— del problema con valores en la frontera descrito anteriormente, donde $a = 1$. La elección $\underline{b} = (1, 1)$ y $\underline{b} = (1, 1, 1)$ con

$c = 0$ generan matrices no simétricas, escogiendo $c = 1$ y $\underline{b} = 0$ se obtienen matrices simétricas las cuales son usadas con propósitos de comparación. En todos los ejemplos se usa una tolerancia de $1e - 6$.

Problemas en Dos Dimensiones En la primera tabla se muestra la descomposición usada, los grados de libertad asociados al sistema y el número de vértices primales usados:

Ejemplo	Partición	Subdominios	Grados Libertad	Primales
1	2×2 y 2×2	4	9	1
2	4×4 y 4×4	16	225	9
3	6×6 y 6×6	36	1225	25
4	8×8 y 8×8	64	3969	49
5	10×10 y 10×10	100	9801	81
6	12×12 y 12×12	144	20449	121
7	14×14 y 14×14	196	38025	169
8	16×16 y 16×16	256	65025	225
9	18×18 y 18×18	324	104329	289
10	20×20 y 20×20	400	159201	361
11	22×22 y 22×22	484	233289	441
12	24×24 y 24×24	576	330625	529
13	26×26 y 26×26	676	455625	625
14	28×29 y 28×28	784	613089	729
15	30×30 y 30×30	900	808201	841

En la segunda tabla se muestra el número de iteraciones requeridas para satisfacer la tolerancia solicitada al método CGM para el caso simétrico:

Ejemplo	PRIMAL#1	PRIMAL#1	DUAL#1	DUAL#2
1	2	1	2	1
2	7	7	6	5
3	9	9	7	6
4	10	10	9	7
5	11	11	10	8
6	12	11	13	9
7	12	12	13	12
8	13	12	14	12
9	13	13	15	13
10	13	13	15	14
11	13	14	15	16
12	14	14	15	15
13	14	14	15	15
14	14	14	15	15
15	15	14	15	15

En la tercer tabla se muestra el número de iteraciones requeridas para satisfacer la tolerancia solicitada al método GMRES para el caso no simétrico:

Ejemplo	PRIMAL#1	PRIMAL#2	DUAL#1	DUAL#2
1	2	1	2	1
2	8	6	6	6
3	10	8	8	8
4	12	10	9	9
5	13	12	9	10
6	14	12	10	10
7	15	13	11	11
8	15	14	11	11
9	16	14	11	12
10	16	15	12	12
11	17	16	12	12
12	17	16	12	13
13	17	16	13	13
14	18	17	13	13
15	18	17	13	13

Cuando la eficiencia de los algoritmos para matrices simétricas y no simétricas son comparadas, se observa que el número de iteraciones son del mismo orden y comparables con los resultados para este mismo tipo de problemas (véase [57]).

Problemas en Tres Dimensiones En la primera tabla se muestra la descomposición usada, los grados de libertad asociados al sistema y el número de vértices primales usados:

Ejemplo	Partición	Subdominios	Grados Libertad	Primales
1	$2 \times 2 \times 2$ y $2 \times 2 \times 2$	8	27	7
2	$3 \times 3 \times 3$ y $3 \times 3 \times 3$	27	512	80
3	$4 \times 4 \times 4$ y $4 \times 4 \times 4$	64	3375	351
4	$5 \times 5 \times 5$ y $5 \times 5 \times 5$	125	13824	1024
5	$6 \times 6 \times 6$ y $6 \times 6 \times 6$	216	42875	2375
6	$7 \times 7 \times 7$ y $7 \times 7 \times 7$	343	110592	4752
7	$8 \times 8 \times 8$ y $8 \times 8 \times 8$	512	250047	8575
8	$9 \times 9 \times 9$ y $9 \times 9 \times 9$	729	512000	14336
9	$10 \times 10 \times 10$ y $10 \times 10 \times 10$	1000	970299	22599

En la segunda tabla se muestra el número de iteraciones requeridas para satisfacer la tolerancia solicitada al método CGM para el caso simétrico:

Ejemplo	PRIMAL#1	PRIMAL#1	DUAL#1	DUAL#2
1	2	2	2	2
2	4	4	3	3
3	5	5	4	3
4	6	5	4	3
5	6	6	4	4
6	7	6	4	4
7	8	7	5	6
8	8	8	7	7
9	8	8	8	8

En la tercer tabla se muestra el número de iteraciones requeridas para satisfacer la tolerancia solicitada al método GMRES para el caso no simétrico:

Ejemplo	PRIMAL#1	PRIMAL#2	DUAL#1	DUAL#2
1	3	2	2	2
2	6	4	4	4
3	7	6	5	5
4	8	7	5	5
5	10	7	6	6
6	11	8	6	6
7	11	9	7	7
8	12	10	8	8
9	13	11	9	9

Cuando la eficiencia de los algoritmos para matrices simétricas y no simétricas son comparadas, se observa que el número de iteraciones son del mismo orden y comparables con los resultados para este mismo tipo de problemas (véase [57]).

De estos resultados, se puede concluir que los métodos de descomposición de dominio en el espacio de vectores derivados desarrollados tanto para tratar problemas simétricos y no simétricos en experimentos numéricos en dos y tres dimensiones presentan una eficiencia del mismo orden (véase [57] y [19]). Además, el desarrollo de los códigos se simplifica, al poder tratar con problemas simétricos y no simétricos en un mismo código.

8.2 Análisis de Rendimiento para Problemas Indefinidos

Para los problemas indefinidos, como es en el caso de la ecuación de Helmholtz, interesa encontrar una malla —lo más gruesa posible— en la cual el problema sea soluble sin obtener un error considerable para valores grandes de k, que normalmente generan inestabilidad numérica en los métodos de discretización, las cuales siempre se eliminan al refinar adecuadamente la malla, la ecuación utilizada es:

$$-\Delta u - k^2 u = f \tag{8.2}$$

en este caso, la discretización se realizo mediante el método de Diferencias Finitas centradas, los ejemplos se resolvieron mediante el método de GMRES con una tolerancia de 10^{-6}, con fines de ejemplificación, aquí mostramos los resultados para $k = 10$.

Para el primer ejemplo, la ecuación utilizada es en 2D; $(x, y) \in [-1, 1] \times [-1, 1]$, donde $u(x, y) = 0$ sobre $\partial\Omega$, los resultados obtenidos para las distintas descomposiciones de dominio, se muestran en la siguiente tabla:

Partición	Grados de Libertad	Primales	PRIMAL#1	PRIMAL#2	DUAL#1	DUAL#2
6×6 y 6×6	1225	25	8	8	8	7
10×10 y 10×10	9801	81	16	13	16	13
14×14 y 14×14	38025	169	18	15	18	15
18×18 y 18×18	104329	289	21	16	20	16
22×22 y 22×22	233289	441	20	17	21	16
26×26 y 26×26	455625	625	21	17	20	17
30×30 y 30×30	808201	841	26	18	21	17

Para el segundo ejemplo, la ecuación utilizada es en 3D; $(x, y, z) \in [-1, 1] \times [-1, 1] \times [-1, 1]$, donde $u(x, y, z) = 0$ sobre $\partial\Omega$, los resultados obtenidos para las distintas descomposiciones de dominio, se muestran en la siguiente tabla:

Partición	Grados de Libertad	Primales	PRIMAL#1	PRIMAL#2	DUAL#1	DUAL#2
$2\times2\times2$ y $2\times2\times2$	27	7	1	1	1	1
$3\times3\times3$ y $3\times3\times3$	512	80	4	4	4	3
$4\times4\times4$ y $4\times4\times4$	3375	351	5	4	4	3
$5\times5\times5$ y $5\times5\times5$	13824	1024	6	6	5	5
$6\times6\times6$ y $6\times6\times6$	42875	2375	7	7	6	5
$7\times7\times7$ y $7\times7\times7$	110592	4752	7	7	6	5
$8\times8\times8$ y $8\times8\times8$	250047	8575	8	8	6	5
$9\times9\times9$ y $9\times9\times9$	512000	14336	8	8	6	6
$10\times10\times10$ y $10\times10\times10$	970299	22599	9	6	6	6

De los resultados mostrados en esta sección, se puede concluir que el esquema de descomposición de dominio en el espacio de vectores derivados para problemas indefinidos presenta buenos resultados para distintos valores de k sin mostrar significativas inestabilidades numéricas en mallas burdas.

Además, haciendo los ajustes pertinentes al esquema de discretización de diferencias finitas —sin hacer cambio alguno al esquema DVS—, es posible resolver la ecuación de Helmholtz tal que no se introduzca error de truncamiento, consecuentemente se puede calcular la solución numérica exacta para la ecuación de Helmholtz para cualquier número de onda sin usar una malla fina (véase [51]).

8.3 Análisis de Rendimiento para Problemas de Advección-Difusión

En el caso de los problemas de Advección-Difusión interesa encontrar una malla —lo más gruesa posible— en la cual el problema sea soluble sin obtener un error considerable al usar valores de viscosidad pequeños que normalmente generan inestabilidad numérica en los métodos de discretización, las cuales siempre se eliminan al refinar adecuadamente la malla.

Para el primer ejemplo, la ecuación utilizada es:

$$-\nu\Delta u + \underline{b}\cdot\nabla u = 0 \tag{8.3}$$

en $(x, y) \in [0, 1] \times [0, 1]$, donde

$$u(x,y) = \begin{cases} 0, & (x,y) \in \xi_1 \\ 1, & (x,y) \in \xi_2 \end{cases} \tag{8.4}$$

y $\underline{b} = (1, 3)$, como se muestra en la figura:

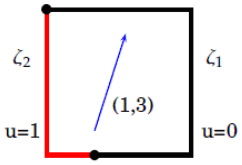

Figura 19: Dominio del problema

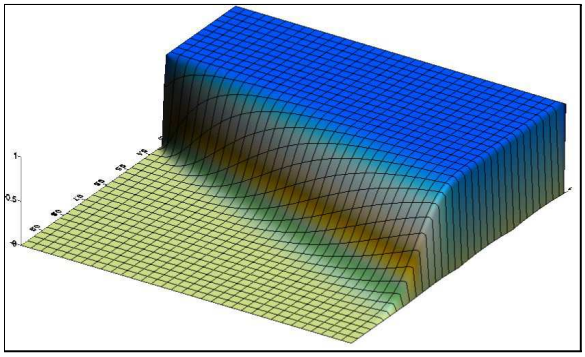

Figura 20: Solución del problema para $\nu = 0.01$

En este caso, la discretización se realizo mediante el método de Diferencias Finitas centradas y en la estabilización se usa el método de Difusión Artificial (véase [35]). Los ejemplos se resolvieron mediante el método de GMRES con

una tolerancia de 10^{-6}, en una malla global de 512×512 (261,121 grados de libertad), para distintos valores de la viscosidad ν (véase [29]). Los resultados obtenidos para las distintas descomposiciones de dominio usando el método BDDC[29] versus los algoritmos DVS, se muestran en la siguiente tabla:

Partición	ν	BDDC	PRIMAL#1	PRIMAL#2	DUAL#1	DUAL#2
8×8 y 64×64	0.01	12	12	11	11	11
8×8 y 64×64	0.001	9	8	8	8	7
8×8 y 64×64	0.0001	9	7	7	7	7
8×8 y 64×64	0.00001	9	7	7	7	7
16×16 y 32×32	0.01	20	19	17	17	18
16×16 y 32×32	0.001	17	14	13	14	13
16×16 y 32×32	0.0001	15	13	13	13	13
16×16 y 32×32	0.00001	16	13	13	13	13
32×32 y 16×16	0.01	33	33	29	29	31
32×32 y 16×16	0.001	30	26	25	25	25
32×32 y 16×16	0.0001	28	25	25	25	25
32×32 y 16×16	0.00001	29	25	25	25	26
64×64 y 8×8	0.01	52	53	53	52	59
64×64 y 8×8	0.001	53	46	46	46	47
64×64 y 8×8	0.0001	53	45	45	47	47
64×64 y 8×8	0.00001	54	45	45	47	48

Además se muestra el residual relativo de decaimiento para la malla gruesa 16×16 y varias mallas finas en las cuales se ve que la mejor convergencia se obtiene cuando la malla fina se incrementa y la convergencia es lenta cuando el subdominio tiene una pequeña cantidad de grados de libertad.

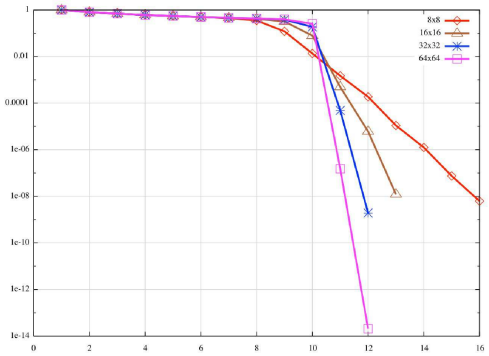

Figura 21: Residual relativo para la malla local de 16×16,en este caso $\underline{b} = (1, 3)$ y $\nu = 0.00001$ que corresponde a un valor de $Pe = 3.16e + 5$.

[29] Ejemplo realizado conjuntamente con Alberto Rosas Medina (véase [35]).

Para el segundo ejemplo, la ecuación a trabajar es

$$-\nu\Delta u + \underline{b}\cdot\nabla u + cu = 0 \tag{8.5}$$

en $(x,y)\in[-1,1]\times[0-1,1]$, donde

$$\begin{aligned} u(x,y) &= 1\begin{cases} y=-1, & 0<x\leq 1 \\ y=1, & 0<x\leq 1 \\ x=1, & -1\leq y\leq 1 \end{cases} \\ u(x,y) &= 0, \qquad \text{en cualquier otro caso} \end{aligned} \tag{8.6}$$

el coeficiente advectivo esta dado por $\underline{b}=(y,-x)$, el valor de $c=10^{-4}$.

En este caso, la discretización se realizo mediante el método de Diferencias Finitas centradas y en la estabilización se usa el método de Difusión Artificial (véase [35]). Los ejemplos se resolvieron mediante el método de GMRES con una tolerancia de 10^{-6}, en una malla global de 32×32 (961 grados de libertad), para distintos valores de la viscosidad ν (véase [41]), cuya solución es mostrada en la gráfica:

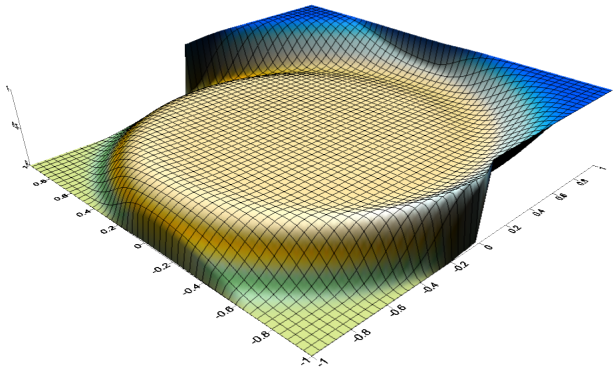

Figura 22: Solución del problema para $\nu = 0.01$

Los resultados obtenidos para las distintas descomposiciones de dominio usando el método FETI[30] versus los algoritmos DVS, se muestran en la siguiente tabla:

[30] Ejemplo realizado conjuntamente con Alberto Rosas Medina (véase [35]).

Partición	ν	FETI-DP	PRIMAL#1	DUAL#1	PRIMAL#2	DUAL#2
4×4 y 8×8	1	11	9	8	8	8
4×4 y 8×8	0.01	12	11	8	10	9
4×4 y 8×8	0.001	23	20	16	20	16
4×4 y 8×8	0.0001	45	24	19	24	18
4×4 y 8×8	0.00001	69	24	19	24	18
8×8 y 4×4	1	10	9	8	8	8
8×8 y 4×4	0.01	11	16	9	10	13
8×8 y 4×4	0.001	27	24	21	24	22
8×8 y 4×4	0.0001	68	32	25	30	26
8×8 y 4×4	0.00001	111	33	24	29	27
16×16 y 2×2	1	9	8	6	6	6
16×16 y 2×2	0.01	16	26	8	9	21
16×16 y 2×2	0.001	63	47	23	28	41
16×16 y 2×2	0.0001	176	48	29	34	42
16×16 y 2×2	0.00001	200	48	30	34	42

Para el tercer ejemplo, la ecuación a trabajar es

$$-\nu\Delta u + \underline{b}\cdot\nabla u + cu = 0 \tag{8.7}$$

en $(x,y)\in[-1,1]\times[-1,1]$, donde

$$\begin{aligned} u(x,y) &= 1\begin{cases} x=-1, & -1<y\leq 1 \\ y=1, & -1\leq x\leq 1 \end{cases} \\ u(x,y) &= 0, \qquad y=-1, -1\leq x\leq 1 \\ u(x,y) &= \frac{1+y}{2}, \qquad x=1, -1\leq y\leq 1 \end{aligned} \tag{8.8}$$

el coeficiente advectivo esta dado por $\underline{b}=\left(\frac{1+y}{2},0\right)$, el valor de $c=10^{-4}$.

En este caso, la discretización se realizo mediante el método de Diferencias Finitas centradas y en la estabilización se usa el método de Difusión Artificial (véase [35]), Los ejemplos se resolvieron mediante el método de GMRES con una to-lerancia de 10^{-6} en la norma infinita en una malla global de 32×32 (961 grados de libertad), para distintos valores de la viscosidad ν (véase [41]) , cuya solución es mostrada en la gráfica:

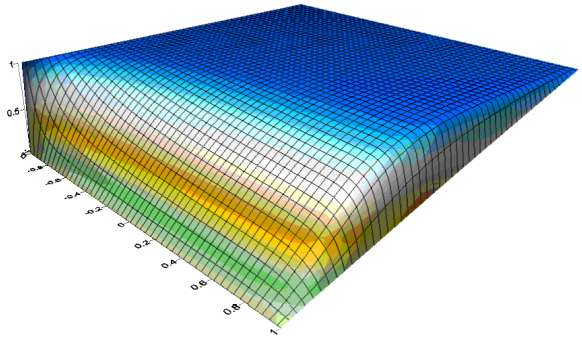

Figura 23: Solución del problema para $\nu = 0.01$

Los resultados obtenidos para las distintas descomposiciones de dominio usando el método FETI-DP[31] versus los algoritmos DVS, se muestran en la siguiente tabla:

Partición	ν	FETI-DP	PRIMAL#1	DUAL#1	PRIMAL#2	DUAL#2
4×4 y 8×8	1	13	10	10	8	8
4×4 y 8×8	0.01	13	11	10	8	7
4×4 y 8×8	0.001	9	8	9	6	6
4×4 y 8×8	0.0001	10	10	10	4	4
4×4 y 8×8	0.00001	11	9	10	3	4
4×4 y 8×8	0.000001	11	9	9	2	3
8×8 y 4×4	1	44	9	9	8	8
8×8 y 4×4	0.01	34	15	14	10	10
8×8 y 4×4	0.001	16	15	15	10	10
8×8 y 4×4	0.0001	16	27	28	9	9
8×8 y 4×4	0.00001	16	32	32	8	8
8×8 y 4×4	0.000001	16	25	25	6	5
16×16 y 2×2	1	159	8	8	6	5
16×16 y 2×2	0.01	98	22	21	9	8
16×16 y 2×2	0.001	38	37	37	18	18
16×16 y 2×2	0.0001	33	48	48	23	22
16×16 y 2×2	0.00001	46	42	41	20	20
16×16 y 2×2	0.000001	51	37	36	15	15

De los resultados mostrados en esta sección, se puede concluir que el esquema de descomposición de dominio en el espacio de vectores derivados para problemas de Advección-Difusión presenta una eficiencia del mismo orden y en algunos casos mejoran la mostrada por los métodos FETI y BDD (véase [29] y [41]).

[31] Ejemplo realizado conjuntamente con Alberto Rosas Medina (véase [35]).

8.4 Análisis de Rendimiento para Sistemas de Ecuaciones

En esta sección se muestra como usar el esquema DVS para resolver problemas con condiciones de frontera Dirichlet donde los desplazamientos son cero sobre la frontera del cuerpo elástico que ocupa el dominio Ω del espacio físico, donde sobre cada subdominio Ω_i que forma la partición gruesa del dominio Ω es resuelto el problema local usando el Método de Elemento Finito (FEM), usando funciones lineales como base.

En el caso de sistemas de ecuaciones, se resolvió[32] un sistema de ecuaciones diferenciales parciales en tres dimensiones que corresponde a la ecuación de elasticidad lineal

$$(\lambda + \mu)\,\nabla\nabla \cdot \underline{u} + \mu\Delta\underline{u} = \underline{f}_{\Omega},\ \text{en } \Omega \tag{8.9}$$

la cual es sujeta a las condiciones de frontera Dirichlet

$$\underline{u} = 0,\ \text{en } \partial\Omega \tag{8.10}$$

el dominio Ω para los experimentos numéricos es un cubo unitario homogéneo isotrópico lineal elástico. En todos nuestros experimentos los nodos primales fueron localizados en las aristas de los subdominios de la partición gruesa, lo cual es suficiente para que la matriz $\underline{\underline{A}}^t$ no sea singular.

Considerando λ y μ iguales a uno, La solución analítica de este problema se escribe como

$$\underline{u} = (\sin\pi x \sin\pi y \sin\pi z, \sin\pi x \sin\pi y \sin\pi z)\,. \tag{8.11}$$

En este caso el operador es simétrico y positivo definido, por ello se usa el método iterativo de Gradiente Conjugado para resolver el sistema lineal de ecuaciones que se genera en el esquema DVS, con una tolerancia de 10^{-7} (véase [63]). Los resultados obtenidos para las distintas descomposiciones de dominio usando el cluster Olintlali se muestran en la siguiente tabla:

Partición	Subdominios	DOF	PRIMAL#1	DUAL#1	PRIMAL#2	DUAL#2
5×5×5 y 5×5×5	125	41472	8	7	9	9
6×6×6 y 6×6×6	216	128625	8	8	10	10
7×7×7 y 7×7×7	343	331776	8	8	11	11
8×8×8 y 8×8×8	512	750141	8	8	12	12

Nótese que, el código desarrollado y usado para problemas escalares que originalmente se desarrollo para resolver una sola ecuación usando en la discretización al método de Diferencias Finitas, fue extendido para resolver problemas con el método de Elemento Finito para resolver sistemas de ecuaciones.

[32] Ejemplo realizado conjuntamente con Iván Contreras Trejo (véase [63]).

8.5 Análisis de Rendimiento en Equipos Paralelos

Para conocer el análisis de rendimiento en equipos paralelos de los métodos desarrollados de descomposición de dominio en el espacio de vectores derivados, se realizaron varias pruebas con la finalidad de conocer la eficiencia y escalabilidad de los códigos y por ende de los métodos en distintos equipos paralelos a los que se tuvo acceso, estos incluyen equipos con 8, 22, 104 y 1024 Cores.

Primeramente, es menester fundamental el encontrar la mejor descomposición de dominio para el problema a trabajar al usar la implementación secuencial y paralela acorde al equipo del que se disponga en aras de obtener la más alta eficiencia posible, después cuando el caso lo permite, se muestran las distintas métricas utilizables y sus limitaciones al aplicarlas en problemas de descomposiciones finas; por último se muestra la escalabilidad del esquema DVS usando hasta 1024 Cores.

8.5.1 Selección Óptima de una Descomposición del Dominio

Para comenzar con la selección óptima de la descomposición del dominio Ω, se toma el problema dado por la Ec.(7.1) como caso particular de la Ec.(8.1) en dos dimensiones con una descomposición fina de 1024×1024 nodos —$1,048,576$ grados de libertad— del dominio Ω, donde por ejemplo se toma sin pérdida de generalidad el algoritmo PRIMAL#1, calculado los tiempos de ejecución en los cuales se usa de uno a ocho Cores de la PC Antipolis y probando las diferentes descomposiciones[33] del dominio —que van desde 2×2 y 512×512 hasta 512×512 y 2×2— se muestran en la siguiente tabla:

	1 Core	2 Cores	3 Cores	4 Cores	5 Cores	6 Cores	7 Cores	8 Cores
Partición	Tiempo	Tiempo	Tiempo	Tiempo	Tiempo	Tiempo	Tiempo	Tiempo
2x2 y 512x512	16465	10659	7207	7105	4641			
4x4 y 256x256	2251	5063	2252	2103	1643	1233	1068	947
8x8 y 128x128	855	885	482	395	314	311	283	272
16x16 y 64x64	321	348	190	149	121	125	118	117
32x32 y 32x32	26	39	26	24	23	21	21	21
64x64 y 16x16	205	595	485	477	481	461	469	469
128x128 y 8x8	1026	5453	5352	5431	5633	5843	5843	5903
256x256 y 4x4	8544	26167	25892	25902	25939	25950	25969	26003
512x512 y 2x2	34845	64230	63293	63308	63389	63475	63502	63693

Por ejemplo, suponiendo que se quiere resolver una descomposición de 2×2 y 512×512 y usar la menor cantidad de Cores posible, entonces se tienen algunas

[33] Para las corridas secuenciales usando métodos de descomposición de dominio, el mejor tiempo de ejecución se obtuvo en una descomposición de 32×32 y 32×32 en 26 segundos —el tiempo de ejecución para el programa secuencial de elemento finito que resuelve el sistema lineal algebraico asociado mediante factorización Cholesky fue de 111 segundos—. Para las corridas en paralelo se obtuvo el mejor tiempo de ejecución en una descomposición de 32×32 y 32×32 con un tiempo de 21 segundos usando 6 Cores —uno para el maestro y 5 para los esclavos—.

opciones para mantener un buen balanceo de cargas —en este caso se tienen 4 subdominios— usando 3 ó 5 Cores:

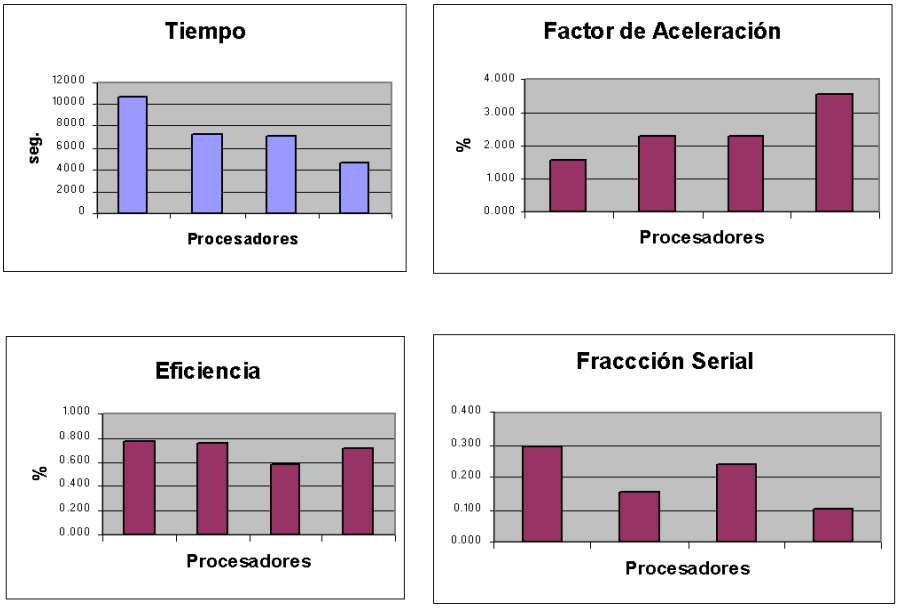

Figura 24: Métricas para la descomposición 2×2 y 512×512

- Si se desea usar 1 Core para el nodo maestro y 2 Cores para los nodos esclavos —dos subdominios por Core—, entonces el factor de aceleración[34] es

$$S(3) = T(1)/T(3) = 16465/7207 = 2.28,$$

la eficiencia es

$$E(3) = T(1)/\left(3 * T(3)\right) = 16465/(3 * 7207) = 0.761,$$

y la fracción serial es

$$F(3) = \frac{\frac{1}{S(3)} - \frac{1}{3}}{1 - \frac{1}{3}} = 0.158.$$

[34] El factor de aceleración S es tal que $1 \leq S(n) \leq n$, la eficiencia E es tal que $1/n \leq E(n) \leq 1$ y la fracción serial F es tal que $0 \leq F(n) \leq 1$.

Se considera que en el caso ideal, el factor de aceleración debería aumentar linealmente al aumentar el número de procesadores $S(p) \simeq p$; por su parte la eficiencia debería de ser cercana a la unidad cuando el hardware se está usando de forma eficiente y en caso contrario se desaprovecha este; por último la fracción serial debería tender a cero y cualquier aumento indica una sobrecarga en los procesos de comunicación.

- Si se desea usar 1 Core para el nodo maestro y 4 Cores para los nodos esclavos —un subdominio por Core—, entonces el factor de aceleración es

$$S(5) = T(1)/T(5) = 16465/4641 = 3.548,$$

la eficiencia es

$$E(5) = T(1)/\left(5 * T(5)\right) = 16465/(5 * 4641) = 0.709$$

y la fracción serial es

$$F(5) = \frac{\frac{1}{S(5)} - \frac{1}{5}}{1 - \frac{1}{5}} = 0.102.$$

En otro ejemplo, suponiendo que se quiere resolver una descomposición de 32×32 y 32×32 y usar la menor cantidad de Cores posible, entonces se tienen algunas opciones para mantener un buen balanceo de cargas —en este caso se tienen 1024 subdominios— usando 3 ó 5 Cores:

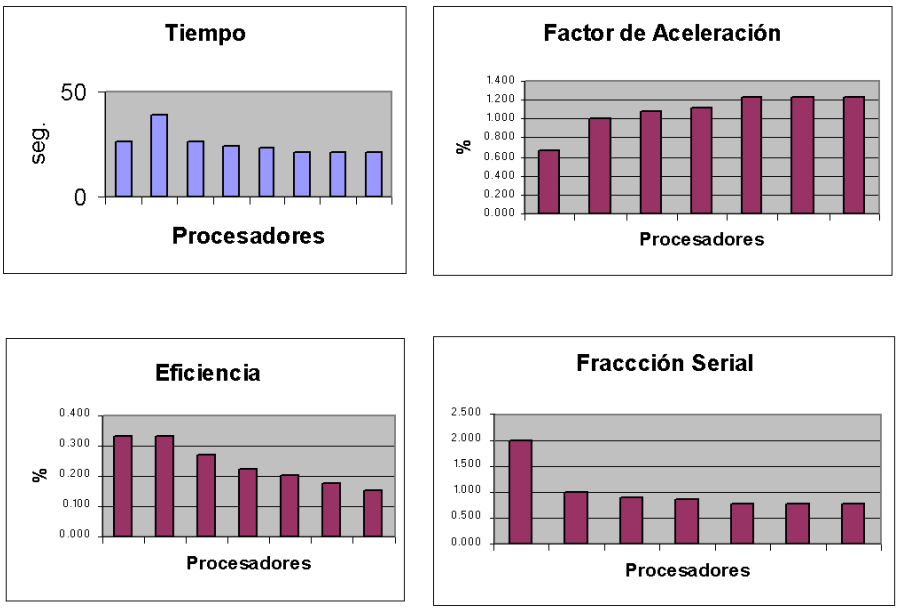

Figura 25: Métricas para la descomposición 32×32 y 32×32

- Si se desea usar 1 Core para el nodo maestro y 2 Cores para los nodos esclavos —512 subdominios por Core—, entonces el factor de aceleración es

$$S(3) = T(1)/T(3) = 26/26 = 1,$$

la eficiencia es

$$E(3) = T(1)/\left(3 * T(3)\right) = 26/(3 * 26) = 0.333$$

y la fracción serial es

$$F(3) = \frac{\frac{1}{S(3)} - \frac{1}{3}}{1 - \frac{1}{3}} = 1.$$

- Si se desea usar 1 Core para el nodo maestro y 4 Cores para los nodos esclavos —256 subdominios por Core—, entonces el factor de aceleración es

  $$S(5) = T(1)/T(5) = 26/23 = 1.130,$$

  la eficiencia es

  $$E(5) = T(1)/\left(5 * T(5)\right) = 26/(5 * 23) = 0.377,$$

  y la fracción serial es

  $$F(5) = \frac{\frac{1}{S(5)} - \frac{1}{5}}{1 - \frac{1}{5}} = 0.856.$$

Nótese que la descomposición usada en el primer ejemplo dista mucho de ser la óptima[35], ya que la descomposición de 32×32 y 32×32 usada en el segundo ejemplo genera el mejor tiempo de ejecución tanto en secuencial como en paralelo, pero las métricas no reflejan esta mejora, aunque el tiempo de ejecución es mínimo. Por ello es necesario siempre hacer corridas de prueba buscando la descomposición que presente el menor tiempo de ejecución posible para el equipo paralelo con el que se cuente. Estas pruebas dependen fuertemente de la capacidad computacional de cada Core y de la red usada para interconectar los Cores que forman parte del equipo paralelo.

Observación 4 *Nótese que esta forma de medir la eficiencia, el factor de aceleración y la fracción serial tiene un detalle fino, ya que el tiempo de ejecución tomado en un procesador —para este caso es de* 16465 *segundos en la descomposición* 2×2 *y* 512×512*— dista mucho de ser el mejor tiempo posible para el problema global de* 1024 *nodos —*26 *segundos—. Esto puede ser una limitante para obtener valores adecuados en las métricas; y medir la eficiencia al usar equipo paralelo cuando se trabaja con la resolución de dominios en los cuales se realiza una descomposición fina; particularmente cuando las descomposiciones son adecuadas para cientos de Cores, pues las pruebas en un Core o en pocos Cores no son posibles de realizar por el consumo excesivo de recursos computacionales y por que no son conmensurables con las corridas secuenciales.*

[35] En la descomposición de 2×2 y 512×512 el tiempo de ejecución secuencial es 16,465 seg., en paralelo usando 3 Cores es de 7,207 seg. y en 5 Cores es de 4,641 seg. Mientras que para la descomposición de 32×32 y 32×32, el tiempo de ejecución secuencial es de 26 seg., en paralelo usando 3 Cores es de 26 seg. y en 5 Cores es de 23 seg.

Además, de los datos de las corridas mostradas en la tabla, es notorio el efecto del mal balanceo de carga, nótese que:

- Para la malla de 32×32 y 32×32 el mejor tiempo de ejecución se obtiene en 6 Cores —21 segundos— y al aumentar el número de Cores en la corrida, no hay disminución del tiempo de cálculo.
- Para la malla de 512×512 y 2×2 el aumento en el número de procesadores sólo incide en un aumento en el tiempo de ejecución.
- En particular, para la malla de 2×2 y 512×512 al usar 3 Cores —sólo dos son usados realmente para el cálculo ya que el tercer Core se usa para la asignación de tareas y el control de los nodos esclavos— el factor de aceleración 2.28 es el esperado para el esquema Maestro-Esclavo.

8.5.2 Análisis de Rendimiento Usando Métricas

En esta sección se muestra mediante particiones relativamente pequeñas del dominio, el uso de las métricas —aceleración, eficiencias y fracción serial— en las cuales es posible obtener una alta eficiencia computacional cuando se proporciona una descomposición del dominio adecuada para el equipo paralelo con el que se cuente.

Haciendo uso del Cluster Pohualli de 104 Cores, a continuación se presentan varias tablas en las cuales se muestran las métricas para diferentes descomposiciones del dominio Ω.

1) Para una descomposición de 4×4 y 150×150 —$360,000$ grados de libertad— se obtiene

Cores	Tiempo	Aceleración	Eficiencia	Frac. Ser.
1	267			
3	146	1.82	0.60	0.32
5	85	3.14	0.62	0.14
9	56	4.76	0.52	0.11
17	33	8.09	0.47	0.06

2) Para una descomposición de 4×4 y 200×200 —$640,000$ grados de libertad— se obtiene

Cores	Tiempo	Aceleración	Eficiencia	Frac. Ser.
1	1082			
3	391	2.76	0.92	0.04
5	216	5.0	1.00	0.00
9	146	7.41	0.82	0.02
17	82	13.19	0.77	0.01

3) Para una descomposición de 4×4 y 250×250 —$1,000,000$ grados de libertad— se obtiene

Cores	Tiempo	Aceleración	Eficiencia	Frac. Ser.
1	2628			
3	946	2.77	0.92	0.039
5	539	4.87	0.97	0.006
9	329	7.98	0.88	0.015
17	184	14.20	0.83	0.012

4) Para una descomposición de 4×4 y 300×300 —$1,440,000$ grados de libertad— se obtiene

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

De estas tablas se desprende que seleccionando la descomposición adecuada se pueden tener excelentes resultados en la eficiencia como es el caso de la descomposición 4×4 y 200×200. Además se muestra una gama de otras eficiencias según el número de procesadores usado y la descomposición seleccionada. Nótese que en todos los casos la fracción serial disminuye sustancialmente con el aumento del número de procesadores.

Otros ejemplos interesantes en los cuales se muestra el efecto de mandar descomposiciones no adecuadas y que se reflejan en una baja eficiencia computacional sin importar el aumento del número de procesadores, pero en todos los casos el tiempo de ejecución siempre disminuye:

i) Para una descomposición de 8×8 y 250×250 —$4,000,000$ grados de libertad— en el Cluster Kanbalam se obtiene

Cores	Tiempo	Aceleración	Eficiencia	Frac. Ser.
1	11366			
3	5541	2.05	0.68	0.23
5	3011	3.77	0.75	0.08
9	1855	6.12	0.68	0.05
17	1031	11.02	0.64	0.03
33	595	19.10	0.57	0.02
65	375	30.30	0.46	0.01

ii) Para una descomposición de 10×9 y 250×250 —$5,625,000$ grados de libertad— en el Cluster Pohualli se obtiene

Cores	Tiempo	Aceleración	Eficiencia	Frac. Ser.
1	19387			
6	4777	4.05	0.67	0.09
11	2702	7.17	0.65	0.05
46	801	24.20	0.52	0.02
91	509	38.08	0.41	0.01

De todos estos ejemplos se desprende que buscando la adecuada descomposición es posible encontrar eficiencias altas para el esquema DVS, pero siempre se tiene que tener en cuenta el buscar el menor tiempo de ejecución —véase sección anterior— antes que una alta eficiencia con un mayor tiempo de ejecución.

Por otro lado, pese a que Kanbalam es más eficiente[36] que los otros Clusters a los que se tuvo acceso —en particular Pohualli—, es posible encontrar una descomposición del dominio que mejore el tiempo de ejecución, aún en equipos con recursos inferiores, para ello es necesario aprovechar las características propias del Hardware del Cluster haciendo una adecuada selección de la descomposición del dominio. Para mostrar esto, se toma una descomposición de 32×32 y 150×150 —$23,040,000$ grados de libertad— en ambos Clusters con los siguientes resultados:

Cores	Pohualli	Kanbalam
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 Pohualli usando como máximo 100 Cores obtiene un tiempo de cálculo inferior al que requiere Kanbalam usando a lo más los 128 Cores.

Haciendo uso de las métricas de aceleración y eficiencia relativa[37] se tiene que para el Cluster Kanbalam $S_{128}^{32} = 5937/2818 = 2.10$ donde lo esperado sería $S_{128}^{32} = 32/128 = 4.00$, para el caso de la eficiencia $E_{128}^{32} = (32/128) * (5937/2818) = 0.52$.

En el caso del Cluster Pohualli se tiene que $S_{100}^{16} = 9158/2661 = 3.44$ donde 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 Pohualli que para Kanbalam, se tiene que $S_{100}^{32} = 5178/2661 = 1.94$ donde 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 Pohualli obtiene valores de una aceleración y eficiencias relativas ligeramente mejores que el Cluster Kanbalam, pero esto no se refleja en la disminución de casi 6% del tiempo de ejecución y del uso de 28 Cores menos.

[36] El Cluster Kanbalam esta formado de procesadores AMD Opteron a 2.6 GHtz de 64 bits, cada 4 Cores con 8 GB de RAM interconectados con un switch de 10 Gbps de baja latencia.

El Cluster Pohualli esta formado de procesadores Intel Xeon a 2.33 GHtz de 64 bits, cada 8 Cores cuentan con 32 GB de RAM interconectados con un switch de 1 Gbps.

[37] 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 $E_p^{p'} = \frac{p'}{p} S_p^{p'} = \frac{p'}{p} \frac{T_{p'}}{T_p}$.

Además, el costo computacional[38] $C_p = P * T_p$, que para el caso del cluster Kanbalam es $C_{128} = 360,704$ y en Pohualli es $C_{100} = 266,100$ que representa una disminución de 27%; además de un factor muy importante, el Cluster Pohualli tuvo un costo monetario mucho menor con respecto del Cluster Kanbalam.

8.5.3 Escalabilidad del Esquema DVS

Por último, se realizaron pruebas[39] con el Cluster Kanbalam, mediante una petición especial para tener acceso a los 1024 Cores del Cluster, a la cual el Comité Técnico del mismo dio acceso después de concluir un reparación mayor del equipo que obligo a apagar el Cluster, este acceso sólo se dio por unas horas y de forma exclusiva, lo cual agradezco enormemente, ya que el Cluster tiene una gran demanda dentro y fuera de la UNAM.

Las pruebas realizadas se hicieron usando desde 32 hasta 1024 Cores, para las descomposiciones de 31×33 y 150×150 —$23,017,500$ grados de libertad—, 31×33 y 200×200 —$40,920,000$ grados de libertad— y 31×33 y 250×250 —$63,937,500$ grados de libertad— se obtienen los siguientes tiempos de ejecución.

	Cores					
Subdominio	32	64	128	256	512	1024
31×33 y 150×150	7315 s	4016 s	2619 s	1941 s	1541 s	1298 s
31×33 y 200×200	ND	16037 s	4916 s	3166 s	2688 s	2295 s
31×33 y 250×250	ND	ND	26587 s	8716 s	6388 s	ND

En donde si usamos las métricas de aceleración y eficiencia relativas obtenemos los siguientes resultados

Subdominio	Aceleración	Aceleración esperada	Eficiencia
31×33 y 150×150	$S_{512}^{32} = 4.7$	$S_{512}^{32} = 32$	$E_{512}^{32} = 0.2$
31×33 y 200×200	$S_{512}^{64} = 5.9$	$S_{512}^{32} = 8$	$E_{512}^{64} = 0.7$
31×33 y 250×250	$S_{512}^{128} = 4$	$S_{512}^{32} = 4$	$E_{512}^{128} = 1.0$

De esta última tabla —$E_{512}^{128} = 1.0$ para la descomposición 31×33 y 250×250—, se desprende que los algoritmos desarrollados son altamente escalables en equipos paralelos, ya que es posible obtener una alta eficiencia al encontrar descomposiciones adecuadas al Hardware. Y que pueden usarse para resolver problemas que involucren una gran cantidad de grados de libertad.

[38] El costo o trabajo de resolver un problema en paralelo es el producto del tiempo de cálculo en paralelo T_p por el número de procesadores usado P y se representa por $C_p = P * T_p$.

[39] No todas las pruebas que se plantearon fueron posibles de realizar, en algunos casos la limitante fue las características físicas de los equipos computacionales, en otros es el acceso limitado y de corta duración —para el uso de 256, 512 y 1024 Cores en el Cluster Kanbalam sólo se dispuso de unas cuantas horas de cómputo y en las cuales, varios Cores del Cluster presentaron fallas de hardware por lo que algunas corridas no se concluyeron de forma satisfactoria—.

8.6 Criterios Integrales para Evaluar el Esquema DVS

En el desarrollo e implementación numérica de los distintos métodos de descomposición de dominio, es necesario medir de alguna forma la eficiencia de los diversos métodos entre sí, algunos criterios comúnmente usados son:

1. Dado un dominio Ω y una descomposición fija, usar el número de iteraciones como criterio de eficiencia.
2. Dado un dominio Ω y haciendo refinamientos de la partición, usar el número de iteraciones como criterio de eficiencia.
3. Dado un dominio Ω y una descomposición del mismo, buscar aquella partición en la que el tiempo de ejecución sea mínimo al variar las particiones posibles.

En principio, estas formas de medir la eficiencia de los diversos métodos no deberían de ser excluyentes entre sí, por el contrario, juntas dan un criterio robusto de la eficiencia de un método de descomposición de dominio para un problema en particular implementado en un equipo de cómputo en las cuales ciertas descomposiciones son posibles —ya sea por limitaciones fenomenológicas o por cuestiones computacionales—.

Para mostrar las implicaciones de las distintas formas de medir la eficiencia, se hace un análisis de las diversas opciones para cada uno de los casos.

1.- Dado un dominio Ω y una descomposición fija, usar el número de iteraciones como criterio de eficiencia En este caso, se usa la Ec.(8.1) como simétrica, tomando una descomposición del dominio Ω en tres dimensiones en la cual se toma una malla gruesa$10 \times 10 \times 10$ que genera $10,000$ subdominios y en la que cada subdominio es descompuesto en $10 \times 10 \times 10$ elementos, los grados de libertad asociados al sistema son $970,299$, donde el número de vértices primales usados es de $22,599$. Obteniendo los siguientes resultados:

	PRIMAL#1	PRIMAL#2	DUAL#1	DUAL#2
Iteraciones:	8	8	8	8

Aquí, lo único que se observa, es que todos los métodos obtienen la misma eficiencia global en cuanto al número de iteraciones, pero nada dice de los tiempos involucrados en la ejecución. Si ahora se toma en cuenta los tiempos de ejecución en un procesador se obtiene:

	PRIMAL#1	PRIMAL#2	DUAL#1	DUAL#2
Tiempo:	1,380s	1,387s	1,490s	1,520s

Y si se usan varios procesadores de un Cluster —en este ejemplo se usó el Cluster Pohualli— se obtiene:

Cores	PRIMAL#1	PRIMAL#2	DUAL#1	DUAL#2
3	966s	965s	930s	953s
11	184s	186s	175s	181s
101	28s	29s	27s	27s

Esta forma integral de medir la eficiencia, da una idea más realista de la eficiencia de los métodos, pero nuevamente hay que tomar en cuenta que los tiempos de ejecución dependen directamente de la arquitectura de cómputo en la que se realicen las pruebas, en especial del balanceo de la carga de trabajo, de la infraestructura de red que interconecten los nodos del Cluster y si estos son Cores virtuales o reales.

2.- Dado un dominio Ω y haciendo refinamientos de la partición, usar el número de iteraciones como criterio de eficiencia En este caso, se usa la Ec.(8.1) como simétrica, se toma una descomposición del dominio Ω en dos dimensiones, en la primer tabla se muestra la descomposición usada, el número de subdominios, los grados de libertad asociados al sistema y el número de vértices primales usados:

Ejemplo	Partición	Subdominios	Grados Libertad	Primales
1	22×22 y 22×22	484	233,289	441
2	24×24 y 24×24	576	330,625	529
3	26×26 y 26×26	676	455,625	625
4	28×29 y 28×28	784	613,089	729
5	30×30 y 30×30	900	808,201	841

En la segunda tabla se muestra el número de iteraciones requeridas para alcanzar la tolerancia solicitada al método:

Ejemplo	PRIMAL#1	PRIMAL#2	DUAL#1	DUAL#2
1	13	14	15	16
2	14	14	15	15
3	14	14	15	15
4	14	14	15	15
5	15	14	15	15

En la siguiente tabla se muestra el tiempo de ejecución en un procesador para concluir las iteraciones:

Ejemplo	PRIMAL#1	PRIMAL#2	DUAL#1	DUAL#2
1	8s	7s	14s	27s
2	13s	13s	21s	40s
3	19s	19s	33s	61s
4	25s	27s	44s	85s
5	36s	38s	61s	116s

En la última tabla se muestra el tiempo de ejecución en 4 procesadores para concluir las iteraciones:

Ejemplo	PRIMAL#1	PRIMAL#2	DUAL#1	DUAL#2
1	2.9s	2.95s	2.93s	2.99s
2	4.80s	4.89s	4.81s	4.85s
3	7.2s	7.4s	7.3s	7.4s
4	8.92s	8.95s	8.91s	8.93s
5	13.02s	13.05s	13.02s	13.3s

Nuevamente, esta forma integral de medir la eficiencia, da una idea más realista de la eficiencia de los métodos, pero nuevamente hay que tomar en cuenta que los tiempos de ejecución dependen directamente de la arquitectura de cómputo en la que se realicen las pruebas, en especial del balanceo de la carga de trabajo, de la infraestructura de red que interconecten los nodos del Cluster y si estos son Cores virtuales o reales.

3.- Dado un dominio Ω y una descomposición del mismo, buscar aquella partición en la que el tiempo de ejecución sea mínimo al variar las particiones posibles Por último, supóngase que deseo resolver la Ec.(7.1) con un dominio Ω mediante una discretización de 1024×1024 nodos ($1,048,576$ grados de libertad) mediante el algoritmo NN-NP-PRIMAL#1 dado por la Ec.(C.21), de manera inmediata surgen las siguientes preguntas: ¿cuáles son las posibles descomposiciones validas? y ¿en cuántos procesadores se pueden resolver cada descomposición?. Para este ejemplo en particular, sin hacer la tabla exhaustiva, se tiene

Partición	Subdominios	Procesadores
2x2 y 512x512	4	2,3,5
4x4 y 256x256	16	2,3,5,9,17
8x8 y 128x128	64	2,3,5,9,17,33,65
16x16 y 64x64	256	2,3,5,9,17,33,65,129,257
32x32 y 32x32	1024	2,3,5,9,17,33,65,129,..,1025
64x64 y 16x16	4096	2,3,5,9,17,33,65,129,...,4097
128x128 y 8x8	16384	2,3,5,9,17,33,65,129,...,16385
256x256 y 4x4	65536	2,3,5,9,17,33,65,129,...,65537
512x512 y 2x2	262144	2,3,5,9,17,33,65,129,...,262145

De esta tabla es posible seleccionar las descomposiciones que se adecuen a las características del equipo paralelo con que se cuente, para evaluar el tiempo de ejecución de este ejemplo use la PC Antipolis, obteniendo resultados mostrados en la tabla de la sección (8.5.1).

De estos resultados, se desprende que, dependiendo del tamaño de la malla gruesa —número de subdominios a trabajar— y de la malla fina, es siempre posible encontrar una descomposición de dominio en que el tiempo de cálculo sea mínimo, tanto al usar un solo Core —programa secuencial 26 segundos—, como al usar múltiples Cores interconectados mediante la biblioteca de paso de mensajes MPI —el tiempo mínimo se obtuvo usando 6 Cores en 21 segundos—, pero es también notorio el efecto que genera el mal balanceo de carga, el cual

se refleja en que no disminuye el tiempo de ejecución al aumentar el número de procesadores y en algunos casos el tiempo aumenta conforme se agregan más Cores.

Nótese que conforme la partición en los subdominios se hace más fina, se incrementa notablemente el tiempo de cálculo necesario para resolver los sistemas lineales asociados a los subdominios, en particular en la resolución del sistema lineal asociado a $\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}$, si el número de nodos por subdominio es grande puede que exceda la cantidad de memoria que tiene a su disposición el Core y por el contrario, un número pequeño de nodos generarían una infrautilización del poder computacional de los nodos esclavos.

Por otro lado, el refinamiento de la malla gruesa, involucra un aumento considerable del número de objetos subdominio en los nodos esclavos, con los que el nodo maestro tendrá comunicación, incrementando la granularidad de las comunicaciones, con la consecuente degradación en la eficiencia.

De todo lo anterior se pueden hacer algunas observaciones importantes Para una evaluación objetiva e integra de la eficiencia de los diversos métodos de descomposición de dominio —en particular de los desarrollados— y su implementación computacional en una arquitectura de cómputo particular, es necesario tomar en cuanta los siguientes factores:

- Número de iteraciones para una descomposición dada.
- Número de iteraciones para diferentes particiones de una descomposición dada.
- Elección de la partición que genere el menor tiempo de ejecución para un problema en una arquitectura de cómputo específica.

Estas formas de medir la eficiencias en su conjunto, dan una idea realista de la eficiencia de los métodos, pero hay que tomar en cuenta que los tiempos de ejecución dependen directamente de la arquitectura de cómputo en la que se realicen las pruebas, en especial del balanceo de la carga de trabajo, de la infraestructura de red que interconecten los nodos del Cluster y si estos son Cores virtuales o reales.

A Consideraciones Sobre la Implementación de Métodos de Solución de Grandes Sistemas de Ecuaciones Lineales

Los modelos matemáticos de muchos sistemas en Ciencia e Ingeniería y en particular una gran cantidad de sistemas continuos geofísicos requieren el procesamiento de sistemas algebraicos de gran escala. En este trabajo se muestra como proceder, para transformar un problema de ecuaciones diferenciales parciales en un sistema algebraico de ecuaciones lineales; y así, poder hallar la solución a dicho problema al resolver el sistema lineal, estos sistemas lineales son expresados en la forma matricial siguiente

$$\underline{\underline{A}}\underline{u} = \underline{f} \tag{A.1}$$

donde la matriz $\underline{\underline{A}}$ es de tamaño $n \times n$ y generalmente bandada, cuyo tamaño de banda es b.

Los métodos de resolución del sistema algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ se clasifican en dos grandes grupos (véase [14]): los métodos directos y los métodos iterativos. 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 (véase [10], [11], [12] y [14]).

Por lo general, es conveniente usar librerías[40] para implementar de forma eficiente a los vectores, matrices —bandadas y dispersas— y resolver los sistemas lineales.

A.1 Métodos Directos

En los métodos directos (véase [10] y [13]), 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.

A.1.1 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

[40] Algunas de las librerías 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.

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

$$\begin{array}{l} \text{Para } i = 1, 2, ..., n \ \{ \\ \quad \text{Para } j = 1, 2, ..., n \ \{ \\ \quad\quad A_{ji} = A_{ji}/A_{ii} \\ \quad\quad \text{Para } k = i+1, ..., n \ \{ \\ \quad\quad\quad A_{jk} = A_{jk} - A_{ji}A_{ik} \\ \quad\quad \} \\ \quad \} \\ \} \end{array} \tag{A.2}$$

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

$$\underline{\underline{L}}\underline{y} = \underline{f} \quad \text{y} \quad \underline{\underline{U}}\underline{u} = \underline{y}. \tag{A.3}$$

Pero recordando que la matriz $\underline{\underline{A}}$ contiene a las matrices $\underline{\underline{L}}$ y $\underline{\underline{U}}$ entonces se tiene que

$$\underline{\underline{L}}\underline{y} = \underline{f} \Leftrightarrow \begin{cases} y_1 = f_1/A_{11} \\ y_i = \left(f_i - \displaystyle\sum_{j=1}^{i-1} A_{ij} y_j \right) \text{ para toda } i = 2, ..., n \end{cases} \tag{A.4}$$

y

$$\underline{\underline{U}}\underline{u} = \underline{y} \Leftrightarrow \begin{cases} x_n = y_n/A_{nn} \\ u_i = \frac{1}{A_{ii}} \left(y_i - \displaystyle\sum_{j=i+1}^{n} A_{ij} x_j \right) \text{ para toda } i = n-1, ..., 1 \end{cases} \tag{A.5}$$

La descomposición $\underline{\underline{LU}}$ requiere[41] $n^3/3 - n/3$ multiplicaciones/divisiones y $n^3/3 - n^2/2 + n/6$ sumas/restas —del orden $O(N^3/3)$ operaciones aritméticas

[41] Notemos que el método de eliminación gaussiana de una matriz $\underline{\underline{A}}$ de tamaño $n \times n$ requiere $n^3/3 + n^2 - n/3$ multiplicaciones/divisiones además de $n^3/3 + n^2/2 - 5n/6$ sumas/restas. El método de Gauss-Jordan requiere $n^3/2 + n^2 - n/2$ multiplicaciones/divisiones además de $n^3/2 + n/2$ sumas/restas.

para la matriz llena pero sólo del orden $O(Nb^2)$ operaciones aritméticas para la matriz con un ancho de banda de b—. La solución de los sistemas $\underline{\underline{L}}\underline{y} = \underline{f}$ y $\underline{\underline{U}}\underline{u} = \underline{y}$ requieren $n^2/2 - n/2$ operaciones aritméticas cada uno (véase [10] y [13]).

A.1.2 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 la matrz $\underline{\underline{L}}$, mientras $\underline{\underline{L}}^T$ no se calcula, quedando el método numérico esquemáticamente como:

$$
\begin{aligned}
&A_{ii} = \sqrt{A_{11}} \\
&\text{Para } j = 2, ..., n \text{ calcule } A_{j1} = A_{j1}/A_{11} \\
&\text{Para } i = 2, ..., n \ \{ \\
&\qquad A_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} A_{ik}^2} \\
&\qquad \text{Para } j = i+1, ..., n \\
&\qquad\qquad A_{ji} = \left(A_{ji} - \sum_{k=1}^{i-1} A_{jk} A_{ik} \right) / A_{ii} \\
&\} \\
&A_{nn} = \sqrt{A_{nn} - \sum_{k=1}^{n-1} \left(A_{nk}\right)^2}
\end{aligned}
\tag{A.6}
$$

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

$$
\underline{\underline{L}}\underline{y} = \underline{f} \quad \text{y} \quad \underline{\underline{L}}^T\underline{u} = \underline{y} \tag{A.7}
$$

usando la formulación equivalente dada por las Ec.(A.4) y (A.5) para la descomposición LU.

La descomposición $\underline{\underline{LDL}}^T$ requiere de $n^3/6 + n^2 - 7n/6$ multiplicaciones/divisiones además de $n^3/6 - n/6$ sumas/restas mientras que para la factorización Cholesky se requieren $n^3/6 + n^2/2 - 2n/3$ multiplicaciones/divisiones además de $n^3/6 - n/6$ sumas/restas, adicionalmente se requiere del cálculo de n raíces cuadradas (véase [10] y [13]).

A.1.3 Factorización LU para Matrices Tridiagonales

Como un caso particular de la Factorización LU, está el método de Crout o Thomas y este es aplicable cuando la matriz sólo tiene tres bandas —la diagonal

principal, una antes y una después de la diagonal principal— la Factorización LU se simplifica considerablemente, en este caso las matrices $\underline{\underline{L}}$ y $\underline{\underline{U}}$ también se dejan en la matriz $\underline{\underline{A}}$ y después de la factorización, se resuelven los sistemas $\underline{\underline{L}}\underline{y} = \underline{f}$ y $\underline{\underline{U}}\underline{u} = \underline{y}$, quedando el método numérico esquemáticamente como:

$$\begin{array}{l} A_{12} = A_{12}/A_{11} \\ y_1 = A_{1,n+1}/A_{11} \\ \text{Para } i = 2, ..., n-1 \text{ \{} \\ \quad A_{ii} = A_{ii} - A_{i,i-1}A_{i-1,i} \\ \quad A_{i,i+1} = A_{i,i+1}/A_{ii} \\ \quad y_i = \left(A_{i,n+1} - A_{i,i-1}y_{i-1}\right)/A_{ii} \\ \text{\}} \\ A_{nn} = A_{nn} - A_{n,n-1}A_{n-1,n} \\ y_n = \left(A_{n,n+1} - A_{n,n-1}y_{n-1}\right)/A_{nn} \\ u_n = y_n \\ \text{Para } i = n-1, ..., 1 \text{ tome } u_i = y_i - A_{i,i+1}u_{i+1} \end{array} \tag{A.8}$$

Esta factorización y la resolución del sistema $\underline{\underline{A}}\underline{u} = \underline{f}$ requiere sólo $(5n-4)$ multiplicaciones/divisiones y $(3n-3)$ sumas/restas, en contraste con la descomposición $\underline{\underline{L}}\underline{\underline{U}}$ que requiere $n^3/3 - n/3$ multiplicaciones/divisiones y $n^3/3 - n^2/2 + n/6$ sumas/restas.

A.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 [10] y [13]).

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

$$\underline{\underline{A}}\underline{u} = \underline{f} \tag{A.9}$$

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

$$\underline{u}^k = \underline{\underline{T}}\underline{u}^{k-1} + \underline{c} \quad \forall k = 1, 2, 3, ... \tag{A.10}$$

La convergencia a la solución la garantiza el siguiente teorema (véase [14]).

Teorema 9 *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

$$cond(\underline{\underline{A}}) = \frac{\lambda_{\max}}{\lambda_{\min}} \geq 1 \tag{A.11}$$

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 [8], [9], [10] y [14]).

Los métodos Jacobi y Gauss-Seidel son usualmente menos eficientes que los métodos discutidos en el resto de esta sección basados en el espacio de Krylov (véase [42] y [13]). Para un ejemplo de este hecho, tomemos el sistema lineal

$$\begin{bmatrix} 4 & 3 & 0 \\ 3 & 4 & -1 \\ 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 24 \\ 30 \\ -24 \end{bmatrix}$$

cuya solución es $x_1 = 3, x_2 = 4, x_3 = -5$; para el método Gauss-Seidel se requirieron 50 iteraciones, Jacobi requirieron 108 iteraciones y Gradiente Conjugado sólo 3 iteraciones.

Los métodos basados en el espacio de Krylov, 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

$$\mathcal{K}_k = Generado\left\{\underline{r_0}, \underline{\underline{A}}\underline{r_0}, ..., \underline{\underline{A}}^{k-1}\underline{r_0}\right\} \text{ para } k \geq 1. \tag{A.12}$$

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

$$\underline{r_k} = \underline{b} - \underline{\underline{A}}\underline{x_k}. \tag{A.13}$$

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—.

A.2.1 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 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.

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 $\underline{\underline{A}}\underline{u} = \underline{f}$, 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:

$$
\begin{array}{l}
\alpha^n = \frac{\left\langle \underline{p}^n, \underline{p}^n \right\rangle}{\left\langle \underline{p}^n, \underline{\underline{A}}\underline{p}^n \right\rangle} \\
\underline{u}^{n+1} = \underline{u}^n + \alpha^n \underline{p}^n \\
\underline{r}^{n+1} = \underline{r}^n - \alpha^n \underline{\underline{A}}\underline{p}^n \\
\text{Prueba de convergencia} \\
\beta^n = \frac{\left\langle \underline{r}^{n+1}, \underline{r}^{n+1} \right\rangle}{\left\langle \underline{r}^n, \underline{r}^n \right\rangle} \\
\underline{p}^{n+1} = \underline{r}^{n+1} + \beta^n \underline{p}^n \\
n = n + 1
\end{array}
\tag{A.14}
$$

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 $(\mu < \varepsilon)$ {
$v = \underline{\underline{A}}\underline{p}$
$\alpha = \frac{\mu}{\underline{p} \cdot \underline{v}}$
$\underline{u} = \underline{u} + \alpha \underline{p}$

$\underline{r} = \underline{r} - \alpha \underline{v}$
$\mu' = \underline{r} \cdot \underline{r}$
$\beta = \frac{\mu'}{\mu}$
$\underline{p} = \underline{r} + \beta \underline{p}$
$\mu = \mu'$
}

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

$$\left\| \underline{u} - \underline{u}^k \right\|_{\underline{\underline{A}}} \leq \left\| \underline{u} - \underline{u}^0 \right\|_{\underline{\underline{A}}} \left[\frac{1 - \sqrt{Cond(\underline{\underline{A}})}}{1 + \sqrt{Cond(\underline{\underline{A}})}} \right]^{2k} . \tag{A.15}$$

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 10 *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*

$$\frac{\|e^n\|}{\|e^0\|} \leq \frac{2}{\left[\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1} \right)^n + \left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1} \right)^{-n} \right]} \leq 2 \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \right)^n \tag{A.16}$$

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

$$\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \simeq 1 - \frac{2}{\sqrt{\kappa}} \tag{A.17}$$

tal que

$$\|\underline{e}^n\|_{\underline{\underline{A}}} \simeq \left\| \underline{e}^0 \right\|_{\underline{\underline{A}}} \exp\left(-2 \frac{n}{\sqrt{\kappa}} \right) \tag{A.18}$$

de lo anterior se puede esperar un espectro de convergencia del orden de $O(\sqrt{\kappa})$ iteraciones (véase [14] y [42]).

Definición 11 *Un método iterativo para la solución de un sistema lineal es llamado óptimo, si la razón de convergencia a la solución exacta es independiente del tamaño del sistema lineal.*

Definición 12 *Un método para la solución del sistema lineal generado por métodos de descomposición de dominio es llamado escalable, si la razón de convergencia no se deteriora cuando el número de subdominios crece.*

A.2.2 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

$$\left\{\underline{v}^1, \underline{v}^2, ..., \underline{v}^n\right\} \tag{A.19}$$

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 $\left\{\underline{v}^{n+1j}\right\}_{j=1}^{n}$ —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 $\underline{\underline{V}}_n = \left[\underline{v}^1, \underline{v}^2, ..., \underline{v}^n\right]$ la cual denota la matriz conteniendo $\underline{v}^j$ en la j-ésima columna, para $j = 1, 2, ..., n$, y sea $\underline{\underline{H}}_n = \left[h_{i,j}\right], 1 \leq i, j \leq n$, 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

$$\begin{aligned} \underline{\underline{AV}}_n &= \underline{\underline{V}}_n \underline{\underline{H}}_n + h_{n+1,n}\left[0, ..., 0, \underline{v}^{n+1}\right] \\ \underline{\underline{H}}_n &= \underline{\underline{H}}_n^T \underline{\underline{AV}}_n. \end{aligned} \tag{A.20}$$

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

$$\underline{\underline{A}}\underline{u} = \underline{f} \tag{A.21}$$

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

$$\begin{array}{l} \text{Para } n = 1, 2, ..., \text{Mientras } \beta^n < \tau\beta^0 \ \{ \\ \quad \underline{w}_0^{n+1} = \underline{\underline{A}}\underline{v}^n \\ \quad \text{Para } l = 1 \text{ hasta } n \ \{ \\ \qquad h_{l,n} = \left\langle \underline{w}_l^{n+1}, \underline{v}^l \right\rangle \\ \qquad \underline{w}_{l+1}^{n+1} = \underline{w}_l^{n+1} - h_{l,n}\underline{v}^l \\ \quad \} \\ \quad h_{n+1,n} = \left\|\underline{w}_{n+1}^{n+1}\right\| \\ \quad \underline{v}^{n+1} = \underline{w}_{n+1}^{n+1}/h_{n+1,n} \\ \quad \text{Calcular } \underline{y}^n \text{ tal que } \beta^n = \left\|\beta^0\underline{e}_1 - \underline{\underline{\hat{H}}}_n\underline{y}^n\right\| \text{ es mínima} \\ \} \end{array} \tag{A.22}$$

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

$$\underline{r}^n = \underline{r}^0 - \underline{\underline{AV}}_n y^n = \underline{\underline{V}}_{n+1}\left(\beta^0\underline{e}_1 - \underline{\underline{\hat{H}}}_n\underline{y}^n\right). \tag{A.23}$$

Teorema 13 *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. $\underline{\underline{A}} = \underline{\underline{V}}\underline{\underline{\Lambda}}\underline{\underline{V}}^{-1}$ 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*

$$\frac{\left\|r^k\right\|}{\left\|r^0\right\|} \leq \kappa\left(V\right) \min_{p_\kappa \in \Pi\kappa, p_k(0)=1} \max_{\lambda_j} \left|p_\kappa\left(\lambda_j\right)\right| \tag{A.24}$$

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

A.3 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[42], 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 [13]).

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 diferencias finitas, 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[43] del procesador.

[42] En el caso de los ejemplos de la sección (G) de MatLab y C++ se usan matrices que minimizan la memoria usada en el almacenamiento de las matrices y maximizan la eficiencia de los métodos numéricos de solución del sistema lineal asociado.

[43] 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

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;
}
```

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.

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 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.

A.3.1 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

$$\underline{\underline{A}} = \begin{bmatrix} a_1 & b_1 & & & c_1 & & & & \\ d_2 & a_2 & b_2 & & & c_2 & & & \\ & d_3 & a_3 & b_3 & & & c_3 & & \\ & & d_4 & a_4 & b_4 & & & c_4 & \\ e_5 & & & d_5 & a_5 & b_5 & & & c_5 \\ & e_6 & & & d_6 & a_6 & b_6 & & \\ & & e_7 & & & d_7 & a_7 & b_7 & \\ & & & e_8 & & & d_8 & a_8 & b_8 \\ & & & & e_9 & & & d_9 & a_9 \end{bmatrix} \tag{A.25}$$

la cual puede ser almacenada usando el algoritmo (véase [13]) 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

$$\underline{Ind} = [-5, -1, 0, +1, +5] \tag{A.26}$$

y los datos serán almacenados usando la estructura

$$\underline{\underline{Dat}} = \begin{bmatrix} 0 & 0 & a_1 & b_1 & c_1 \\ 0 & d_2 & a_2 & b_2 & c_2 \\ 0 & d_3 & a_3 & b_3 & c_3 \\ 0 & d_4 & a_4 & b_4 & c_4 \\ e_5 & d_5 & a_5 & b_5 & c_5 \\ e_6 & d_6 & a_6 & b_6 & 0 \\ e_7 & d_7 & a_7 & b_7 & 0 \\ e_8 & d_8 & a_8 & b_8 & 0 \\ e_9 & d_9 & a_9 & 0 & 0 \end{bmatrix} \tag{A.27}$$

de tal forma que la matriz $\underline{\underline{A}}$ puede ser reconstruida de forma eficiente. Para obtener el valor $A_{i,j}$, calculo $ind = j - i$, si el valor ind esta 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 buscara 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.

A.3.2 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

$$\underline{\underline{A}} = \begin{bmatrix} a_1 & & & & & b_1 & & & c_1 \\ & a_2 & & & b_2 & & & c_2 & \\ & & & & a_3 & & & b_3 & c_3 \\ a_4 & & & b_4 & & & & & \\ & & a_5 & & & b_5 & & & c_5 \\ a_6 & b_6 & c_6 & & & & & & \\ & & & & & & a_7 & b_7 & c_7 \\ & & & a_8 & & & b_8 & c_8 & \\ & & & & & & a_9 & b_9 & \end{bmatrix} \tag{A.28}$$

la cual puede ser almacenada usando el algoritmo (véase [13]) Jagged Diagonal Storage (JDC), optimizado para ser usado en C++. Para este ejemplo en particular, se hará uso de una matriz de índices

$$\underline{\underline{Ind}} = \begin{bmatrix} 1 & 6 & 9 \\ 2 & 5 & 8 \\ 5 & 8 & 9 \\ 1 & 4 & 0 \\ 3 & 6 & 9 \\ 1 & 2 & 3 \\ 7 & 8 & 9 \\ 4 & 7 & 8 \\ 7 & 8 & 0 \end{bmatrix} \tag{A.29}$$

y los datos serán almacenados usando la estructura

$$\underline{\underline{Dat}} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \\ a_4 & b_4 & 0 \\ a_5 & b_5 & c_5 \\ a_6 & b_6 & c_6 \\ a_7 & b_7 & c_7 \\ a_8 & b_8 & c_8 \\ a_9 & b_9 & 0 \end{bmatrix} \tag{A.30}$$

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.

A.3.3 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.

B Marco Teórico del Espacio de Vectores Derivados DVS

En el marco de los métodos de descomposición de dominio sin traslape se distinguen dos categorías (véase [18], [33], [34], [44], [45] y [49]): Los esquemas duales —como es el caso del método Finite Element Tearing and Interconnect (FETI) y sus variantes— los cuales usan multiplicadores de Lagrange; y los esquemas primales —como es el caso del método Balancing Domain Decomposition (BDD) y sus variantes— que tratan el problema sin el recurso de los multiplicadores de Lagrange.

En este trabajo se ha derivado un sistema unificador (véase [59] y [61]), el esquema del espacio de vectores derivados (DVS), este es un esquema primal similar a la formulación BDD. Donde una significativa diferencia entre nuestro esquema y BDD es que en la formulación DVS el problema es transformado en otro definido en el espacio de vectores derivados, el cual es un espacio producto conteniendo funciones discontinuas, es en este espacio en el cual todo el trabajo del método es realizado.

Por otro lado, en la formulación BDD, el espacio original de funciones continuas nunca es abandonado completamente y constantemente se regresa a los grados de libertad asociados con el espacio de funciones continuas pertenecientes a las subestructuras, el cual en su formulación juega el rol del espacio producto.

Aunque el marco DVS es un esquema primal, las formulaciones duales pueden ser acomodadas en él; esta característica permite unificar en este terreno a ambos esquemas, duales y primales; en particular a BDDC y FETI-DP. También el espacio DVS constituye un espacio de Hilbert con respecto al adecuado producto interior —el producto interior Euclidiano— y mediante la utilización de la formulación DVS, se saca provecho de la estructura del espacio de Hilbert obteniendo de esta manera una gran simplicidad para el algoritmo definido en el espacio de funciones definidas por tramos y es usado para establecer una clara correspondencia entre los problemas a nivel continuo y aquellos obtenidos después de la discretización.

B.1 Formulaciones Dirichlet-Dirichlet y Neumann-Neumann a Nivel Continuo

Para poner la metodología desarrollada en una adecuada perspectiva, se inicia por revisar algunos conceptos elementales sobre las formulaciones mencionadas en el título de esta sección. También se toman algunas herramientas presentadas del trabajo "Theory of differential equations in discontinuous picewise-defined functions" (véase [54]).

El problema que se considera aquí[44] es: "Encontrar $u \in H^2(\Omega)$, tal que

$$\begin{cases} \mathcal{L}u = f_\Omega, \text{ en } \Omega \\ u = 0, \text{ sobre } \partial\Omega \end{cases} \text{„} \tag{B.1}$$

[44] La notación que se usa es la estándar para los espacios de Sobolev (véase [2]).

así, bajo las adecuadas condiciones (véase [6]), la existencia y la unicidad de la solución de este problema está garantizada. Este problema puede también formularse en espacio de funciones definidas por tramos (véase [54]).

Sin pérdida de generalidad, sea el dominio Ω descompuesto en dos subdominios Ω_1 y Ω_2, como se muestra en la figura:

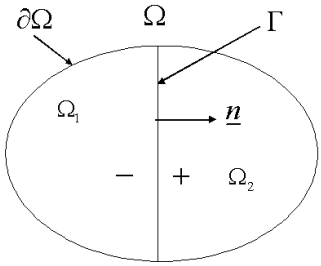

Figura 26: Partición del dominio Ω en dos subdominios Ω_1 y Ω_2.

Supóngase que el operador diferencial $\mathcal{L}$ es de segundo orden y se considera el espacio $H^2(\Omega_1) \oplus H^2(\Omega_2)$ de funciones discontinuas definidas por tramos (véase [54]). Una función en tal espacio es definida independientemente en cada uno de los dos subdominios y sus restricciones a Ω_1 y Ω_2 pertenecen a $H^2(\Omega_1)$ y $H^2(\Omega_2)$ respectivamente. Generalmente la discontinuidad a través de la interfase Γ tiene un salto no cero de la función misma y de sus derivadas normales. La notación para el 'salto' y el 'promedio' a través de Γ esta dado por

$$[\![u]\!] \equiv u_+ - u_- \qquad \text{y} \qquad \dot{\widehat{u}} = \frac{1}{2}(u_+ + u_-), \text{ sobre } \Gamma \tag{B.2}$$

respectivamente.

Nótese que, el espacio $H^2(\Omega)$ es un subespacio de $H^2(\Omega_1) \oplus H^2(\Omega_2)$. En efecto, sea $u \in H^2(\Omega_1) \oplus H^2(\Omega_2)$, entonces $u \in H^2(\Omega)$ si y sólo si

$$[\![u]\!] = \left[\!\left[\frac{\partial u}{\partial n}\right]\!\right] = 0, \text{ sobre } \Gamma. \tag{B.3}$$

Por lo tanto, una formulación del problema dado en la Ec.(B.1) es: "Buscar a $u \in H^2(\Omega_1) \oplus H^2(\Omega_2)$, tal que

$$\begin{cases} \mathcal{L}u = f_\Omega, \text{ en } \Omega \\ [\![u]\!] = 0 \text{ y } \left[\!\left[\frac{\partial u}{\partial n}\right]\!\right] = 0, \text{ sobre } \Gamma \quad \text{". } \\ u = 0, \text{ sobre } \partial\Omega \end{cases} \tag{B.4}$$

Se debe observar que, cuando el valor de la solución u es conocida sobre Γ, entonces u puede ser obtenida en cualquier lugar en Ω por la resolución de dos problemas Dirichlet con valor en la frontera, uno en cada uno de los subdominios

Ω_1 y Ω_2 —el título de Dirichlet-Dirichlet para el procedimiento es por el hecho de resolver este tipo de problemas—.

De manera similar, cuando los valores de la derivada normal $\frac{\partial u}{\partial n}$ son conocidos sobre Γ, entonces u puede ser obtenida en cualquier lugar en Ω por la resolución de dos problemas Neumann con valores en la frontera, uno para cada uno de los subdominios Ω_1 y Ω_2 —el título de Neumann-Neumann para el procedimiento es por el hecho de resolver este tipo de problemas—.

Estas observaciones son las bases de los dos enfoques de los métodos de descomposición de dominio que se consideran en esta sección: Los métodos Dirichlet-Dirichlet y Neumann-Neumann.

B.2 El Problema no Precondicionado Dirichlet-Dirichlet

Para el problema no precondicionado Dirichlet-Dirichlet, se toma u_Γ como la restricción a Γ de la solución u de la Ec.(B.4), entonces u es la única solución de los siguientes dos problemas Dirichlet

$$\begin{cases} \mathcal{L}u = f_\Omega, \text{ en } \Omega_\alpha, \alpha = 1, 2 \\ u = u_\Gamma, \text{ sobre } \Gamma \\ u = 0, \text{ sobre } \partial\Omega \end{cases} . \tag{B.5}$$

El enfoque Dirichlet-Dirichlet al método de descomposición de dominio consiste en buscar para la función u_Γ, esencialmente una elección de sucesiones de funciones de prueba: $u_\Gamma^0, u_\Gamma^1, ..., u_\Gamma^n, ...$, hasta que se encuentre la función buscada.

A este respecto, una primera pregunta es: ¿cómo se reconoce cuando la función de prueba es satisfactoria?. Se sabe que cuando u_Γ es la restricción a Γ de la solución del problema, la solución que es obtenida por la resolución de los dos problemas con valores en la frontera de la Ec.(B.5) satisfacen las condiciones de salto indicadas en la Ec.(B.4). Ahora, en el caso del enfoque Dirichlet-Dirichlet, el salto de la función se nulifica necesariamente, ya que

$$[\![u]\!] = u_+ - u_- = u_\Gamma - u_\Gamma = 0 \tag{B.6}$$

sin embargo, por lo general la condición

$$\left[\!\!\left[\frac{\partial u}{\partial n} \right]\!\!\right] = 0 \tag{B.7}$$

no es satisfecha. La selección de la función de prueba u_Γ será satisfactoria, si y sólo si, la Ec.(B.7) se satisface.

Si se escribe la solución del problema u de la Ec.(B.4) como

$$u = u_p + v \tag{B.8}$$

donde u_p satisface

$$\begin{cases} \mathcal{L}u_p = f_\Omega, \text{ en } \Omega_\alpha, \alpha = 1, 2 \\ u_p = 0, \text{ sobre } \Gamma \\ u_p = 0, \text{ sobre } \partial\Omega \end{cases} \tag{B.9}$$

y por lo tanto v satisface

$$\begin{cases} \mathcal{L}v = 0, \text{ en } \Omega_\alpha, \alpha = 1,2 \\ v = u_\Gamma, \text{ sobre } \Gamma \\ v = 0, \text{ sobre } \partial\Omega \end{cases} \tag{B.10}$$

entonces, la elección de la función de prueba u_Γ será satisfactoria, si y sólo si

$$\left[\!\left[\frac{\partial v}{\partial n}\right]\!\right] = -\left[\!\left[\frac{\partial u_p}{\partial n}\right]\!\right] \tag{B.11}$$

desde este punto de vista, la función $v \in H^2(\Omega_1) \oplus H^2(\Omega_2)$. Así, se busca una función tal que

$$\begin{cases} \mathcal{L}v = 0, \text{ en } \Omega_\alpha, \alpha = 1,2 \\ [\![v]\!] = 0, \left[\!\left[\frac{\partial v}{\partial n}\right]\!\right] = -\left[\!\left[\frac{\partial u_p}{\partial n}\right]\!\right], \text{ sobre } \Gamma \\ v = 0, \text{ sobre } \partial\Omega \end{cases}. \tag{B.12}$$

Esta condición puede expresarse teniendo en mente el operador de Steklov-Poincaré[45] τ, el cual para cualquier función u definida sobre Γ, se genera otra función definida sobre Γ, a saber (véase [3])

$$\tau(u_\Gamma) \equiv \left[\!\left[\frac{\partial v}{\partial n}\right]\!\right], \text{ sobre } \Gamma \tag{B.13}$$

donde v satisface la Ec.(B.10). Entonces la Ec.(B.12) es equivalente a

$$\tau(u_\Gamma) \equiv -\left[\!\left[\frac{\partial u_p}{\partial n}\right]\!\right], \text{ sobre } \Gamma. \tag{B.14}$$

B.3 El Problema no Precondicionado Neumann-Neumann

Para el caso del problema no precondicionado Neumann-Neumann, se toma a u como la solución de la Ec.(B.4) y sea $q_\Gamma \equiv \frac{\partial u}{\partial n}$ sobre Γ, entonces u es la única solución de los siguientes dos problemas Neumann

$$\begin{cases} \mathcal{L}u = f_\Omega, \text{ en } \Omega_\alpha, \alpha = 1,2 \\ \frac{\partial u}{\partial n} = q_\Gamma \text{ sobre } \Gamma \\ u = 0 \text{ sobre } \partial\Omega \end{cases}. \tag{B.15}$$

El título del enfoque Neumann-Neumann proviene de el hecho que la Ec.(B.15) implica resolver un problema Neumann en el subdominio Ω_1 y otro problema Neumann en Ω_2. Este enfoque consiste en buscar una función q_Γ —independientemente del valor de q_Γ—, ya que, cualquier solución de la Ec.(B.15) satisface

$$\left[\!\left[\frac{\partial u}{\partial n}\right]\!\right] = \left(\frac{\partial u}{\partial n}\right)_+ - \left(\frac{\partial u}{\partial n}\right)_- = q_\Gamma - q_\Gamma = 0 \tag{B.16}$$

[45] El operador de Steklov-Poincaré generalmente se define como la ecuación para la traza de la solución exacta u sobre Γ (véase [40]).

sin embargo, por lo general, la condición

$$[\![u]\!] = 0 \tag{B.17}$$

no es satisfecha. La selección de la función de prueba u_Γ será satisfactoria, si y sólo si, la Ec.(B.17) se satisface.

Si se toma nuevamente una solución $u = u_p + v$ como en la Ec.(B.8), donde u_p ahora satisface

$$\begin{cases} \mathcal{L}u_p = f_\Omega, \text{ en } \Omega_\alpha, \alpha = 1,2 \\ \frac{\partial u_p}{\partial n} = 0, \text{ sobre } \Gamma \\ u_p = 0, \text{ sobre } \partial\Omega \end{cases} \tag{B.18}$$

y por lo tanto v satisface

$$\begin{cases} \mathcal{L}v = 0, \text{ en } \Omega_\alpha, \alpha = 1,2 \\ \frac{\partial v}{\partial n} = q_\Gamma, \text{ sobre } \Gamma \\ v = 0, \text{ sobre } \partial\Omega \end{cases} \tag{B.19}$$

entonces, la función $v \in H^2(\Omega_1) \oplus H^2(\Omega_2)$ es caracterizada por

$$\begin{cases} \mathcal{L}v = 0, \text{ en } \Omega_\alpha, \alpha = 1,2 \\ [\![v]\!] = [\![u_p]\!], \left[\!\left[\frac{\partial v}{\partial n}\right]\!\right] = 0, \text{ sobre } \Gamma \\ v = 0, \text{ sobre } \partial\Omega \end{cases} . \tag{B.20}$$

Para la formulación Neumann-Neumann existe una contraparte —el operador μ— al operador de Steklov-Poincaré τ, dada cualquier función q_Γ, definida sobre Γ, se define $\mu(q_\Gamma)$, esta será una función definida sobre Γ dada por

$$\mu(q_\Gamma) \equiv [\![v]\!], \text{ sobre } \Gamma \tag{B.21}$$

aquí, v satisface la Ec.(B.19). Entonces, la Ec.(B.20) es equivalente a

$$\mu(q_\Gamma) \equiv -[\![u_p]\!], \text{ sobre } \Gamma. \tag{B.22}$$

B.4 Discretización del Dominio para los Métodos Duales y Primales

Dos de los enfoques más comúnmente usados (véase [18], [31], [32], [33], [34], [44], [45], [46], [48] y [49]) en los métodos de descomposición de dominio son FETI-DP y BDDC. Ambos, inicializan con la ecuación diferencial parcial y de ella, los grados de libertad son asociados con las funciones de base usadas. BDDC es un método directo, i.e. es un método que no hace uso de Multiplicadores de Lagrange, mientras que FETI-DP trabaja en el espacio producto mediante el uso de los Multiplicadores de Lagrange.

En los métodos FETI-DP y BDDC, el dominio Ω en el cual se define el problema, es subdividido en E subdominios 'sin traslape' Ω_α, $\alpha = 1, 2, ..., E$, es decir

$$\Omega_\alpha \cap \Omega_\beta = \varnothing \quad \forall \alpha \neq \beta \quad \text{y} \quad \overline{\Omega} = \bigcup_{\alpha=1}^{E} \overline{\Omega}_\alpha \tag{B.23}$$

y al conjunto

$$\Gamma = \bigcup_{\alpha=1}^{E} \Gamma_\alpha, \quad \text{si} \quad \Gamma_\alpha = \partial\Omega_\alpha \backslash \partial\Omega \tag{B.24}$$

se le llama la frontera interior del dominio Ω. La notación $\partial\Omega$ y $\partial\Omega_\alpha$, $\alpha = 1, ..., E$ es tomada de la frontera del dominio Ω y la frontera del subdominio Ω_i respectivamente. Claramente

$$\partial\Omega \subset \bigcup_{\alpha=1}^{E} \partial\Omega_\alpha \quad \text{y} \quad \Omega = \left(\bigcup_{\alpha=1}^{E} \Omega_\alpha \right) \bigcup \Gamma. \tag{B.25}$$

Se denota por H al máximo diámetro $H_\alpha = Diam(\Omega_\alpha)$ de cada Ω_α que satisface $Diam(\Omega_\alpha) \leq H$ para cada $\alpha = 1, 2, ..., E$, además, cada subdominio Ω_α es descompuesto en un mallado fino $\mathcal{T}_h$ de K subdominios mediante una triangulación Ω_j de modo que este sea conforme, se denota por h al máximo diámetro $h_j = Diam(\Omega_j)$ de cada Ω_α que satisface $Diam(\Omega_j) \leq h$ para cada $j = 1, 2, ..., K$ de cada $\alpha = 1, 2, ..., E$.

Para iniciar la discusión, se considera un ejemplo particular, de un conjunto de veinticinco nodos de una descomposición del dominio Ω 'sin traslape', se asume que la numeración de los nodos es de izquierda a derecha y de arriba hacía abajo, véase figura (27)

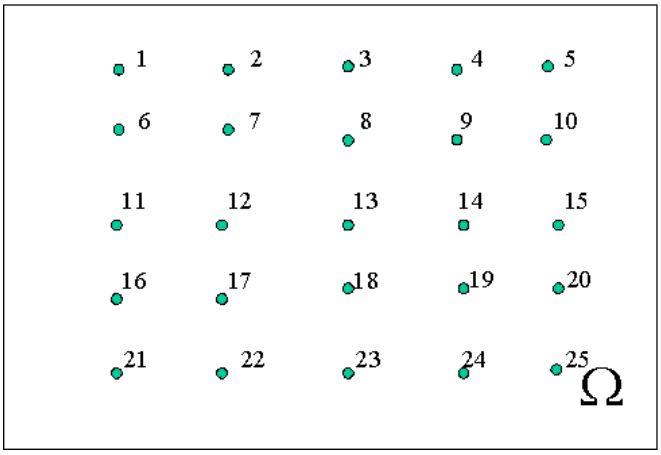

Figura 27: Conjunto de nodos originales

En este caso el conjunto de nodos originales, se particiona usando una malla gruesa de cuatro subdominios Ω_α $\alpha = 1, 2, 3, 4$, véase figura (28).

Entonces se tiene un conjunto de nodos y un conjunto de subdominios, los cuales se enumeran usando un conjunto de índices

$$\hat{N} \equiv \{1, 2, \ldots, 25\} \qquad \text{y} \qquad E \equiv \{1, 2, 3, 4\} \tag{B.26}$$

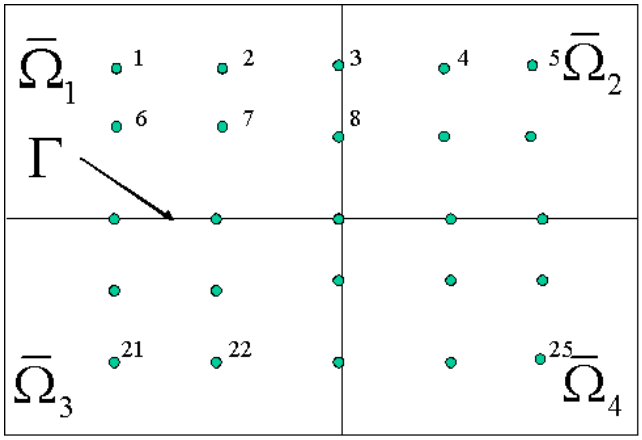

Figura 28: Nodos en una descomposición gruesa de cuatro subdominios

respectivamente. Entonces, el conjunto de 'nodos originales' correspondiente a tal descomposición de dominio 'sin traslape', en realidad tiene un 'traslape', esto es por que la familia de los cuatros subconjuntos

$$\begin{aligned} \hat{N}^1 &= \{1,2,3,6,7,8,11,12,13\} \\ \hat{N}^2 &= \{3,4,5,8,9,10,13,14,15\} \\ \hat{N}^3 &= \{11,12,13,16,17,18,21,22,23\} \\ \hat{N}^4 &= \{13,14,15,18,19,20,23,24,25\} \end{aligned} \tag{B.27}$$

no son disjuntos. En efecto, por ejemplo

$$\hat{N}^1 \cap \hat{N}^2 = \{3,8,13\}\,. \tag{B.28}$$

Con la idea de obtener una 'verdadera descomposición de dominio sin traslape', se reemplaza el conjunto de 'nodos originales' por otro conjunto, el conjunto de los 'nodos derivados' en los cuales se satisfaga la condición de que sea una verdadera descomposición de dominio sin traslapes.

B.5 Una Verdadera Descomposición de Dominio sin Traslapes

A la luz de lo visto en la sección anterior, es ventajoso trabajar con una descomposición de dominio sin traslape alguno del conjunto de nodos y, para este fin, se reemplaza el conjunto $\hat{N}$ de 'nodos originales' por otro conjunto de 'nodos derivados', los cuales son denotados por X^α, donde cada nodo en la frontera interior Γ se parte en un número que llamaremos la multiplicidad del nodo y es definida como el número de subdominios que comparten al nodo. Cada nodo derivado es definido por un par de números: (p,α), donde p corresponde a un nodo perteneciente a $\overline{\Omega}_p$, i.e. un 'nodo derivado' es el par de números (p,α) tal que $p \in \hat{N}^\alpha$, véase figura (29).

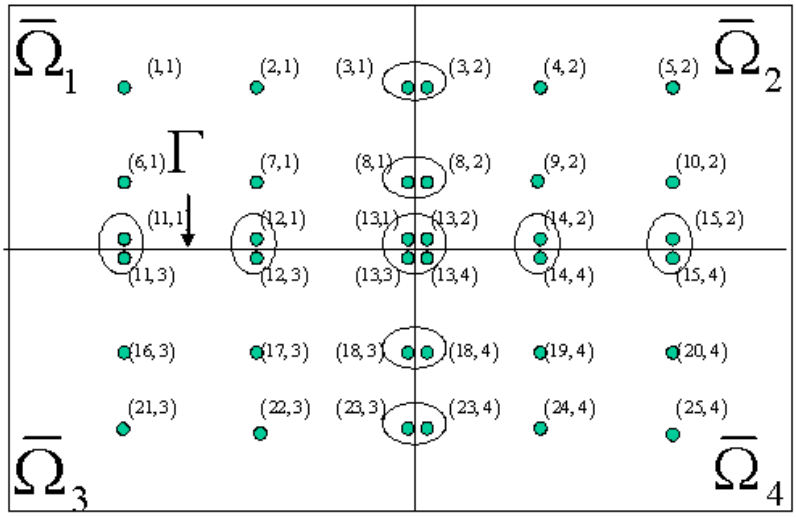

Figura 29: Mitosis de los nodos en la frontera interior

Se denota con X el conjunto total de nodos derivados; nótese que el total de nodos derivados para el ejemplo de la sección anterior es de 36, mientras el número de nodos originales es de 25.

Entonces, se define X^α como el conjunto de nodos derivados que puede ser escrito como

$$X^\alpha = \left\{ (p, \alpha) \mid \alpha \in \hat{E} \quad \text{y} \quad p \in \hat{N}^\alpha \right\}. \tag{B.29}$$

Para el ejemplo referido, tomando α sucesivamente como $1, 2, 3$ y 4, se tiene la familia de cuatro subconjuntos,

$$\left\{ X^1, X^2, X^3, X^4 \right\} \tag{B.30}$$

la cual es una descomposición de dominio sin traslape véase figura (30), donde el conjunto X satisface

$$X = \bigcup_{\alpha=1}^{4} X^\alpha \quad \text{y} \quad X^\alpha \cap X^\beta = \varnothing \quad \text{cuando} \quad \alpha \neq \beta. \tag{B.31}$$

Por supuesto, la cardinalidad de los subdominio —el número de nodos de cada uno de los subdominios es 36/4— es igual a 9.

Dada la discusión anterior, y para el desarrollo de una teoría general, sea el conjunto de nodos-índice y de subdominio-índice original definidos como

$$\hat{N} = \{1, ..., n\} \quad \text{y} \quad \hat{E} = \{1, ..., E\} \tag{B.32}$$

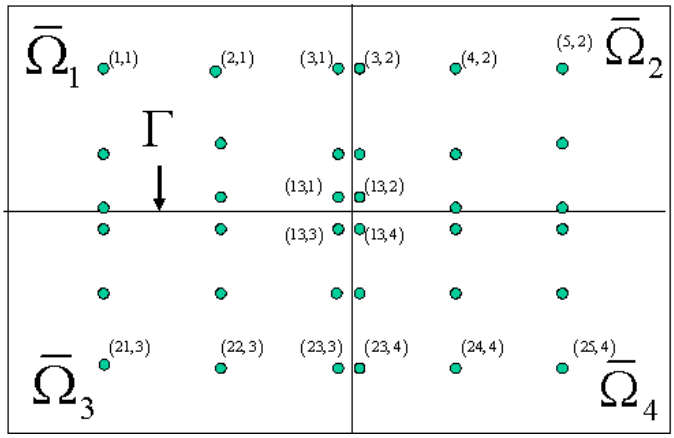

Figura 30: Conjunto de nodos derivados

respectivamente, donde n es el número total de nodos que se obtienen de la versión discretizada de la ecuación diferencial que se quiere resolver —generalmente, los nodos de frontera no son incluidos—.

Se define al conjunto de 'nodos derivados' por el par de números (p, α), tal que $p \in \hat{N}^{\alpha}$. Entonces, el conjunto de todos los nodos derivados, satisface

$$X^{\alpha} \equiv \left\{ (p, \alpha) \mid \alpha \in \hat{E} \quad \text{y} \quad p \in \hat{N}^{\alpha} \right\} \tag{B.33}$$

y para evitar repeticiones, ya que en lo sucesivo se hará uso extensivo de los nodos derivados, la notación (p, α) se reserva para el par tal que $(p, \alpha) \in X$.

Nótese que la cardinalidad de X es siempre mayor que la cardinalidad de $\hat{N}$, excepto en el caso trivial cuando $E = 1$. En lo que sigue los nodos derivados serán referidos, simplemente como nodos.

Por otro lado, dado cualquier nodo original, $p \in \hat{N}$, se define el conjunto $Z(p) \subset X$, como el conjunto de los nodos derivados de p que puede ser escrito como (p, α), para algún $1 \leq \alpha \leq E$. Además, dado cualquier nodo $(p, \alpha) \in X$, su multiplicidad es definida por la cardinalidad del conjunto $Z(p) \subset X$ y es denotada como $m(p)$.

Los nodos derivados son clasificados como 'nodos internos' o de 'frontera interna', dependiendo si su multiplicidad es uno o más grande que uno. Los subconjuntos $I \subset \overline{\Omega}$ y $\Gamma \subset \overline{\Omega}$ son el conjunto de nodos internos y de frontera interna respectivamente, que satisfacen la siguiente relación

$$X = I \cup \Gamma \quad \text{y} \quad I \cap \Gamma = \varnothing \tag{B.34}$$

nótese que para el ejemplo mostrado en la figura 3, la cardinalidad de Γ es 20.

Para cada $\alpha = 1, 2, \ldots, E$, se define

$$\Gamma_{\alpha} = \Gamma \cap X^{\alpha} \tag{B.35}$$

entonces, la familia de subconjuntos $\{\Gamma_1, \Gamma_2, \ldots, \Gamma_E\}$ es una partición de Γ, en el sentido de que

$$\Gamma \equiv \bigcup_{\alpha=1}^{E} \Gamma_\alpha \quad \text{mientras} \quad \Gamma_\alpha \cap \Gamma_\beta = \varnothing \quad \text{cuando} \quad \alpha \neq \beta. \tag{B.36}$$

En este desarrollo, Γ es descompuesto dentro de dos subconjuntos

$$\pi \subset \Gamma \subset X \quad \text{y} \quad \Delta \subset \Gamma \subset X \tag{B.37}$$

los nodos pertenecientes al primer conjunto π son llamados 'primales', mientras que si estos pertenecen al segundo conjunto Δ son llamados 'duales', donde

$$\Delta \equiv \Gamma - \pi$$

entonces

$$\Gamma = \pi \cup \Delta \quad \text{mientras} \quad \pi \cap \Delta = \varnothing. \tag{B.38}$$

Además, se define

$$\Pi \equiv I \cup \pi$$

en este caso, cada una de las familias de los subconjuntos $\{I, \pi, \Delta\}$ y $\{\Pi, \Delta\}$ son descomposiciones sin traslape de $\overline{\Omega}$, es decir,

$$X = I \cup \pi \cup \Delta \quad \text{mientras que} \quad I \cap \pi = \pi \cap \Delta = \Delta \cap I = \varnothing \tag{B.39}$$

y

$$X = \Pi \cup \Delta \quad \text{mientras que} \quad \Pi \cap \Delta = \varnothing. \tag{B.40}$$

Nótese que todos los conjuntos considerados hasta ahora son dimensionalmente finitos. Por lo tanto, cualquier función con valores en los reales[46] definida en cualquier conjunto define unívocamente un vector dimensionalmente finito.

Para el planteamiento de los métodos Dual-Primal, los nodos de la interfase son clasificados dentro de nodos Primales y Duales. Entonces se define:

- $\hat{N}_I \subset \hat{N}$ como el conjunto de nodos interiores.
- $\hat{N}_\Gamma \subset \hat{N}$ como el conjunto de nodos de la interfase.
- $\hat{N}_\pi \subset \hat{N}$ como el conjunto de nodos primales.
- $\hat{N}_\Delta \subset \hat{N}$ como el conjunto de nodos duales.

El conjunto $\hat{N}_\pi \subset \hat{N}_\Gamma$ es escogido arbitrariamente y entonces $\hat{N}_\Delta$ es definido como $\hat{N}_\Delta \equiv \hat{N}_\Gamma - \hat{N}_\pi$. Cada una de las siguientes dos familias de nodos son disjuntas:

$$\left\{\hat{N}_I, \hat{N}_\Gamma\right\} \quad \text{y} \quad \left\{\hat{N}_I, \hat{N}_\pi, \hat{N}_\Delta\right\}.$$

[46] Cuando se consideren sistemas de ecuaciones, como en los problemas de elasticidad, tales funciones son vectoriales.

Además, estos conjuntos de nodos satisfacen las siguientes relaciones:

$$\hat{N} = \hat{N}_I \cup \hat{N}_\Gamma = \hat{N}_I \cup \hat{N}_\pi \cup \hat{N}_\Delta \quad \text{y} \quad \hat{N}_\Gamma = \hat{N}_\pi \cup \hat{N}_\Delta \tag{B.41}$$

adicionalmente se define los siguientes conjuntos de X :

- $I \equiv \left\{(p,\alpha) \mid p \in \hat{N}_I\right\}.$
- $\Gamma \equiv \left\{(p,\alpha) \mid p \in \hat{N}_\Gamma\right\}.$
- $\pi \equiv \left\{(p,\alpha) \mid p \in \hat{N}_\pi\right\}.$
- $\Delta \equiv \left\{(p,\alpha) \mid p \in \hat{N}_\Delta\right\}.$

En vista de todo lo anterior, la familia de subconjuntos $\left\{X^1, ..., X^E\right\}$ es una descomposición de dominio del conjunto de nodos derivados en donde no se tiene ningún traslape (véase [59] y [61]), es decir

$$X = \bigcup_{\alpha=1}^{E} X^\alpha \quad \text{y} \quad X^\alpha \cap X^\beta = \varnothing \quad \text{cuando} \quad \alpha \neq \beta. \tag{B.42}$$

B.6 El Problema Original

El esquema DVS puede ser aplicado al sistema matricial que es obtenido después de la discretización, este procedimiento es independiente del método de discretización usado —puede ser el método de Elemento Finito, Diferencias Finitas o cualquier otro—. El procedimiento requiere de algunas suposiciones que deben de ser satisfechas, las cuales se explican a continuación (véase [59] y [61]). Tales suposiciones están dadas en términos del sistema matricial y dos conceptos adicionales: los nodos originales y una familia de subconjuntos de tales nodos, los cuales están asociados con la partición del dominio.

Para ilustrar como estos conceptos son introducidos, se considera la formulación variacional de la versión discretizada general de un problema de valores en la frontera, el cual consiste en encontrar $\hat{u} \in V$, tal que

$$a(\hat{u}, v) = (g, v), \quad \forall v \in V \tag{B.43}$$

aquí, V es un espacio lineal de dimensión finita de funciones real valuadas[47] definidas en cierto dominio espacial Ω, mientras $g \in V$ es una función dada.

Sea $\hat{N} = \{1, ..., n\}$ el conjunto de índices, el cual es el número de nodos usados en la discretización, y sea $\{\varphi_1, ..., \varphi_n\} \subset V$ una base de V, tal que para cada $i \in \hat{N}, \varphi_i = 1$ en el nodo i y cero en cualquier otro nodo. Entonces, ya que $\hat{u} \in V$, entonces

$$\hat{u} = \sum \widehat{u}_i \varphi_i \tag{B.44}$$

[47] La teoría aquí presentada, con ligeras modificaciones, trabaja también en el caso de que las funciones de V sean vectoriales.

aquí, $\widehat{u}_i$ es el valor de $\hat{u}$ en el nodo i. Sea $\underline{\widehat{u}}$ y $\underline{\widehat{f}}$ vectores[48] $\underline{\widehat{u}} \equiv \left(\widehat{u}_1, ..., \widehat{u}_n\right)$ y $\underline{\widehat{f}} \equiv \left(\widehat{f}_1, ..., \widehat{f}_n\right)$, donde

$$\widehat{f}_i \equiv (g, \varphi_i), \quad i \in \hat{N}. \tag{B.45}$$

La formulación variacional de la Ec.(B.43) es equivalente a

$$\underline{\underline{\widehat{A}}}\,\underline{\widehat{u}} = \underline{\widehat{f}} \tag{B.46}$$

donde la matriz $\underline{\underline{\widehat{A}}}$, será referida como la 'matriz original' y esta es definida mediante

$$\underline{\underline{\widehat{A}}} \equiv \left(\widehat{A}_{ij}\right) \tag{B.47}$$

con

$$\widehat{A}_{ij} \equiv \tilde{a}\left(\varphi_i, \varphi_j\right) \qquad i, j = 1, ..., n \tag{B.48}$$

después de que el problema ha sido discretizado —$\tilde{a}\left(\varphi_i, \varphi_j\right)$ son las funciones base usadas en la discretización—, donde se supone que el dominio Ω ha sido particionado mediante un conjunto de subdominios no traslapados $\{\Omega_1, ..., \Omega_E\}$; más precisamente, para cada $\alpha = 1, ..., E$; Ω_α es abierto y

$$\Omega_\alpha \cap \Omega_\beta = \varnothing \quad \forall \alpha \neq \beta \quad \text{y} \quad \Omega = \bigcup_{\alpha=1}^{E} \overline{\Omega}_\alpha \tag{B.49}$$

donde $\overline{\Omega}_\alpha$ es la clausura estándar del conjunto Ω_α.

El conjunto de subdominios-índices es denotado por $\hat{E} = \{1, ..., E\}$. Por otro lado $\hat{N}^\alpha$, $\alpha = 1, ..., E$, denota a los nodos originales que corresponden a los nodos pertenecientes a $\overline{\Omega}_\alpha$. Como es usual, los nodos son clasificados en 'interiores' y 'nodos de interfase'; un nodo es interior, si pertenece sólo a la clausura de una subpartición del dominio, y es un nodo de la interfase cuando pertenece a más de uno.

Las funciones real valuadas definidas en $\hat{N} = \{1, ..., n\}$ constituyen un espacio lineal el cual será denotado por $\widehat{W}$ y referido como el 'espacio vectorial original'. Los vectores $\underline{\widehat{u}} \in \widehat{W}$ se escriben como $\underline{\widehat{u}} = \left(\widehat{u}_1, ..., \widehat{u}_n\right)$, donde $\widehat{u}_i$ para $i = 1, ..., n$, son las componentes de un vector $\underline{\widehat{u}}$.

Entonces, el 'problema original' consiste en "Dada $\underline{\widehat{f}} \in \widehat{W}$, encontrar a $\underline{\widehat{u}} \in \widehat{W}$ tal que la Ec.(B.46) se satisfaga".

A lo largo de todo el desarrollo de esta metodología, la matriz original $\underline{\underline{\widehat{A}}}$ se asume como no singular —esto define una biyección de $\widehat{W}$ sobre sí misma—, para definir las condiciones sobre las cuales el esquema DVS es aplicable para matrices

[48] Hablando estrictamente estos deberían ser vectores columna, sin embargo, cuando se incorporan en el texto, se escriben como vectores renglón para ahorrar espacio de escritura.

indefinidas y/o no simétricas (véase [57]), se asume lo siguiente, (axioma): "Sean los índices $i \in \hat{N}^\alpha$ y $j \in \hat{N}^\beta$ de nodos interiores originales, entonces

$$\widehat{A}_{ij} = 0 \quad \text{siempre y cuando} \quad \alpha \neq \beta". \tag{B.50}$$

Así, para cada $\alpha = 1, ..., E$ se define el subespacio de vectores $\widehat{W}^\alpha \subset \widehat{W}$, el cual es constituido por los vectores que tienen la propiedad que para cada $i \notin \hat{N}^\alpha$, esta i-ésima componente se nulifica. Usando esta notación se define el espacio producto W por

$$W \equiv \prod_{\alpha=1}^{E} \widehat{W}^\alpha = \widehat{W}^1 \times ... \times \widehat{W}^E. \tag{B.51}$$

B.7 El Espacio de Vectores Derivado

Usando la notación de la sección anterior, en la cual se definió el 'problema original' dado por la Ec.(B.46), este es un problema formulado en el espacio vectorial original $\widehat{W}$, en el desarrollo que sigue se transforma en un problema que será reformulado en el espacio W, el cual es un espacio definido sobre las funciones discontinuas.

Para ello, se entiende por un 'vector derivado' una función real-valuada definida en el conjunto X de nodos derivados[49]. El conjunto de vectores derivados constituyen un espacio lineal, el cual será referido como el 'espacio de vectores derivados'. De manera análoga para cada subconjunto local de nodos derivados en el subespacio X^α existe un 'subespacio local de vectores derivados' W^α, el cual es definido con la condición de que los vectores de W^α se nulifiquen en cada nodo derivado que no pertenezca a X^α. Una manera formal de iniciar esta definición es

$$\underline{u} \in W^\alpha \subset W, \quad \text{si y sólo si} \quad \underline{u}\,(p, \beta) = 0 \quad \text{cuando} \quad \beta \neq \alpha. \tag{B.52}$$

Una importante diferencia entre los subespacios W^α y $\widehat{W}^\alpha$ puede se notada al observar que $W^\alpha \subset W$, mientras que $\widehat{W}^\alpha \subsetneq W$. En particular

$$W \equiv \prod_{\alpha=1}^{E} \widehat{W}^\alpha = W^1 \oplus ... \oplus W^E \tag{B.53}$$

en palabras: El espacio W es el producto de subespacios de la familia

$$\left\{\widehat{W}^1, ..., \widehat{W}^E\right\} \tag{B.54}$$

pero al mismo tiempo es la suma directa de la familia

$$\left\{W^1, ..., W^E\right\}. \tag{B.55}$$

[49] Para el tratamiento de sistemas de ecuaciones, tales como las involucradas en el tratamiento de la elasticidad lineal, tales funciones serán vectoriales.

En vista de la Ec.(B.53) es sencillo establecer una biyección —de hecho, un isomorfismo— entre el espacio de vectores derivados y el espacio producto. Por lo tanto, en lo que sigue, se identifica a ambos como uno solo.

Para cada par de vectores[50] $\underline{u} \in W$ y $\underline{w} \in W$, el 'producto interior Euclidiano' es definido por

$$\underline{u} \cdot \underline{w} = \sum_{(p,\alpha)\in X} \underline{u}\,(p,\alpha)\,\underline{w}\,(p,\alpha)\,. \tag{B.56}$$

Una importante propiedad es que el espacio de vectores derivados W, constituye un espacio de Hilbert dimensionalmente finito con respecto al producto interior Euclidiano. Nótese que el producto interior Euclidiano es independiente de la forma de la matriz original $\widehat{\underline{\underline{A}}}$; en particular esta puede ser simétrica, no simétrica o indefinida.

La inyección natural $R : \widehat{W} \to W$, de $\widehat{W}$ sobre W, es definida por la condición de que, para todo $\hat{\underline{u}} \in \widehat{W}$, se obtenga

$$\left(R\hat{\underline{u}}\right)(p,\alpha) = \hat{\underline{u}}\,(p)\,, \quad \forall\,(p,\alpha) \in X. \tag{B.57}$$

La 'multiplicidad', $m\,(p)$ de cualquier nodo original $p \in \hat{N}$ es caracterizada por la propiedad (véase [58] y [57]),

$$\sum_{\alpha=1}^{E} \left(R\hat{\underline{u}}\right)(p,\alpha) = m(p)\hat{\underline{u}}\,(p)\,. \tag{B.58}$$

Además, el espacio W puede ser descompuesto en dos subespacios ortogonales complementarios $W_{11} \subset W$ y $W_{12} \subset W$, tal que

$$W = W_{11} + W_{12} \quad \text{y} \quad \{0\} = W_{11} \cap W_{12} \tag{B.59}$$

donde, el subespacio $W_{12} \subset W$ es la inyección natural de $\widehat{W}$ dentro de W; i.e.

$$W_{12} \equiv R\widehat{W} \subset W \tag{B.60}$$

y $W_{11} \subset W$ es el complemento ortogonal con respecto al producto interior Euclidiano. Además se define la inversa de $R : \widehat{W} \to W$, cuando ésta es restringida a $W_{12} \subset W$ la cual siempre existe y es denotada por $R^{-1} : W_{12} \to \widehat{W}$.

Es costumbre el uso de la notación de suma directa

$$W = W_{12} \oplus W_{12} \tag{B.61}$$

cuando el par de condiciones de la Ec.(B.59) son satisfechas.

[50] En el caso vectorial —que surge en aplicaciones como en la teoría de elasticidad— cuando se trabajan sistemas de ecuaciones, i.e. cuando $\underline{u}\,(p,\alpha)$ es un vector, la Ec(B.56) es reemplazada por

$$\underline{u} \cdot \underline{w} = \sum_{(p,\alpha)\in X} \underline{u}\,(p,\alpha) \odot \underline{w}\,(p,\alpha)$$

donde, $\underline{u}\,(p,\alpha) \odot \underline{w}\,(p,\alpha)$ se identifica con el producto interior de vectores involucrados.

El 'subespacio de vectores continuos' es definido como el subespacio

$$W_{12} \subset W \tag{B.62}$$

mientras que el 'subespacio de vectores de promedio cero' es definido como el subespacio

$$W_{11} \subset W. \tag{B.63}$$

Dos matrices $\underline{\underline{a}} : W \to W$ y $\underline{\underline{j}} : W \to W$ son ahora introducidas; ellas son los operadores proyección con respecto al producto interior Euclidiano sobre W_{12} y W_{11} respectivamente. El primero será referido como el 'operador promedio' y el segundo como el 'operador salto' respectivamente.

En vista de la Ec.(B.59), cada vector $\underline{u} \in W$, puede ser escrito de forma única como la suma de un vector de promedio cero más un vector continuo (vector de salto cero), es decir

$$\underline{u} = \underline{u}_{11} + \underline{u}_{12} \text{ con } \begin{cases} \underline{u}_{11} \equiv \underline{\underline{j}}\underline{u} \in W_{11} \\ \underline{u}_{12} \equiv \underline{\underline{a}}\underline{u} \in W_{12} \end{cases} \tag{B.64}$$

los vectores $\underline{\underline{j}}\underline{u}$ y $\underline{\underline{a}}\underline{u}$ son llamados el 'salto' y el 'promedio' de $\underline{u}$ respectivamente.

Los subespacios lineales que se definen a continuación son elegidos conforme a la nomenclatura estándar. En particular W_I, W_Γ, W_π y W_Δ son definidos para imponer restricciones sobre sus miembros. Los vectores de:

- W_I se nulifica en cada nodo derivado que no es un nodo interno.
- W_Γ se nulifica en cada nodo derivado que no es un nodo de interfase.
- W_π se nulifica en cada nodo derivado que no es un nodo primal.
- W_Δ se nulifica en cada nodo derivado que no es un nodo dual.

Además

- $W_r \equiv W_I + \underline{\underline{a}}W_\pi + W_\Delta$.
- $W_\Pi \equiv W_I + \underline{\underline{a}}W_\pi$.

Nótese que, para cada una de las siguientes familias de subespacios

$$\{W_I, W_\Gamma\}, \{W_I, W_\pi, W_\Delta\}, \{W_\Pi, W_\Delta\} \tag{B.65}$$

son linealmente independientes. Y también satisfacen

$$W = W_I + W_\Gamma = W_I + W_\pi + W_\Delta \text{ y } W_r = W_\Pi + W_\Delta \tag{B.66}$$

la anterior definición de W_r es apropiada cuando se considera las formulaciones Dual-Primal. En el presente trabajo sólo se considera la restricción impuesta por el promedio en los nodos primales —otro ejemplo de restricción es tomar el salto sobre los nodos primales—. Otros tipos de restricciones requieren cambiar el término $\underline{\underline{a}}W_\pi$ por $\underline{\underline{a}}^r W_\pi$ donde $\underline{\underline{a}}^r$ es la proyección sobre el espacio restringido, el tipo de restricciones que pueden ser definidas están en función del problema particular a resolver (véase [29]).

B.8 Discretización Partiendo de la Construcción de la Matriz $\underline{\underline{A}}^t$

Una característica notable del enfoque DVS es que este inicia con la matriz que es obtenida después de que el problema se ha discretizado —a partir de una discretización por algún método como por ejemplo Elemento Finito o Diferencias Finitas— y para su aplicación no se requiere ninguna información acerca de la ecuación diferencial parcial de la cual se originó. Por supuesto, tal matriz no es definida en el espacio de vectores derivados y la teoría provee una fórmula para derivar la matriz en el espacio de vectores derivados (véase [59]); aquí se da un procedimiento para definir la matriz $\underline{\underline{A}}^t : W \to W$, la cual es usada para formular el problema en el espacio de vectores derivados.

Sea la matriz $\widehat{\underline{\underline{A}}} : \widehat{W} \to \widehat{W}$ dada por la Ec.(B.46) la cual puede ser escrita como

$$\widehat{\underline{\underline{A}}} \equiv \left(\widehat{A}_{pq}\right) \tag{B.67}$$

para cada par (p, q) tal que $p \in \hat{N}$ y $q \in \hat{N}$, y se define

$$\delta_{pq}^{\alpha} \equiv \begin{cases} 1, & \text{si } p, q \in \hat{N}^{\alpha} \\ 0, & \text{si } p \notin \hat{N}^{\alpha} \text{ ó } q \notin \hat{N}^{\alpha} \end{cases}, \alpha = 1, ..., E \tag{B.68}$$

junto con

$$m(p, q) \equiv \sum_{\alpha=1}^{N} \delta_{pq}^{\alpha} \tag{B.69}$$

y

$$s(p, q) \equiv \begin{cases} 1, & \text{cuando } m(p, q) = 0 \\ m(p, q), & \text{cuando } m(p, q) \neq 0 \end{cases} \tag{B.70}$$

la función $m(p, q)$ es llamada la 'multiplicidad' del par (p, q). Esto puede verse de la suposición básica dada por la Ec.(B.50), teniendo que

$$m(p, q) = 0 \Rightarrow \widehat{A}_{pq} = 0. \tag{B.71}$$

Definiendo ahora

$$\widehat{\underline{\underline{A}}}^{\alpha} \equiv \left(\widehat{A}_{pq}^{\alpha}\right) \text{ con } \widehat{A}_{pq}^{\alpha} = \frac{\widehat{A}_{pq}\delta_{pq}^{\alpha}}{s(p, q)} \tag{B.72}$$

se observa la identidad

$$\widehat{\underline{\underline{A}}} = \sum_{\gamma=1}^{E} \widehat{\underline{\underline{A}}}^{\gamma} \tag{B.73}$$

ya que se ha usado una verdadera descomposición del dominio sin traslape. Además, para cada $\gamma = 1, ..., E$, la matriz $\underline{\underline{A}}^{\gamma} : W \to W$ es definida por

$$\underline{\underline{A}}^{\gamma} \equiv \left(A_{(p,\alpha)(q,\beta)}^{\gamma}\right) \tag{B.74}$$

con

$$A^{\gamma}_{(p,\alpha)(q,\beta)} \equiv \delta^{\gamma}_{\alpha\gamma}\widehat{A}^{\gamma}_{pq}\delta_{\beta\gamma} \tag{B.75}$$

entonces la matriz $\underline{\underline{A}}^t : W \to W$ (t de total, no de transpuesta) es dada por

$$\underline{\underline{A}}^t \equiv \sum_{\gamma=1}^{E} \underline{\underline{A}}^{\gamma} \tag{B.76}$$

una propiedad fundamental de $\underline{\underline{A}}^t$ es que

$$\left(\underline{\underline{A}}^t\right)^{-1} = \sum_{\gamma=1}^{E} \left(\underline{\underline{A}}^{\gamma}\right)^{-1} \tag{B.77}$$

ya que las matrices $\underline{\underline{A}}^{\gamma}$, con $\gamma = 1, 2, ..., E$ son independientes entre si —al ser una verdadera descomposición del dominio, ver sección (B.5)—. Además, otra propiedad es que

$$R^{-1}\underline{\underline{aA}}R = R^{-1}\underline{\underline{aAa}}R = \widehat{A}. \tag{B.78}$$

Nótese que la matriz $\underline{\underline{A}}^t$ puede ser expresada en más de una manera. Algunas opciones útiles que se usan son (véase [57] y [58]):

$$\begin{aligned}
\underline{\underline{A}}^t &= \begin{pmatrix} \underline{\underline{A}}^t_{\Pi\Pi} & \underline{\underline{A}}^t_{\Pi\Delta} \\ \underline{\underline{A}}^t_{\Delta\Pi} & \underline{\underline{A}}^t_{\Delta\Delta} \end{pmatrix} = \begin{pmatrix} \underline{\underline{A}}^t_{II} & \underline{\underline{A}}^t_{I\pi} & \underline{\underline{A}}^t_{I\Delta} \\ \underline{\underline{A}}^t_{\pi I} & \underline{\underline{A}}^t_{\pi\pi} & \underline{\underline{A}}^t_{\pi\Delta} \\ \underline{\underline{A}}^t_{\Delta I} & \underline{\underline{A}}^t_{\Delta\pi} & \underline{\underline{A}}^t_{\Delta\Delta} \end{pmatrix} \\
&= \begin{pmatrix} \underline{\underline{A}}^1 & \underline{\underline{0}} & \cdots & \underline{\underline{0}} \\ \underline{\underline{0}} & \underline{\underline{A}}^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \underline{\underline{0}} \\ \underline{\underline{0}} & \cdots & \underline{\underline{0}} & \underline{\underline{A}}^N \end{pmatrix}.
\end{aligned} \tag{B.79}$$

A la luz de la Ec.(B.77), implica que, si el complemento de Schur local

$$\underline{\underline{S}}^{\alpha} = \underline{\underline{A}}^{\alpha}_{\Gamma\Gamma} - \underline{\underline{A}}^{\alpha}_{\Gamma I}\left(\underline{\underline{A}}^{\alpha}_{II}\right)^{-1}\underline{\underline{A}}^{\alpha}_{I\Gamma} \tag{B.80}$$

de cada $\underline{\underline{A}}^{\alpha}$ existe y es invertible —véase sección (4.3) para la definición y su implementación del complemento de Schur en general—, entonces, la matriz 'total del complemento de Schur', $\underline{\underline{S}}^t : W_r \to W_r$, satisface

$$\underline{\underline{S}}^t = \sum_{\alpha=1}^{E} \underline{\underline{S}}^{\alpha} \tag{B.81}$$

y

$$\left(\underline{\underline{S}}^t\right)^{-1} = \sum_{\alpha=1}^{E} \left(\underline{\underline{S}}^{\alpha}\right)^{-1} \tag{B.82}$$

al ser una verdadera descomposición del dominio —ver sección (B.5)— y porque se satisfacen las Ecs.(B.76 y B.77).

Por otro lado, si se define $\underline{u}^{'} \equiv R\widehat{\underline{u}}$, usando la Ec.(B.78) entonces el problema original Ec.(B.46), se transforma en uno equivalente a

$$\underline{\underline{a}}\underline{\underline{A}}^{t}\underline{u}^{'} = \overline{\underline{f}} \quad \text{con} \quad \underline{\underline{j}}\underline{u}^{'} = 0 \tag{B.83}$$

una vez que $\underline{u}^{'} \in W$ es obtenida, se puede recuperar $\widehat{\underline{u}} \in \widehat{W}$ usando

$$\widehat{\underline{u}} = R^{-1}\underline{u}^{'} \tag{B.84}$$

esta última es correcta cuando $\underline{u}^{'}$ es evaluada exactamente, pero en las aplicaciones numéricas, $\underline{u}^{'}$ sólo es una evaluación aproximada, por ello puede generar "inestabilidades", en el sentido de que una aproximación a $\underline{u}^{'}$, independientemente de cuan pequeño sea el error, puede no ser continua en cuyo caso $\widehat{u} = R^{-1}\underline{u}^{'}$ no esta definida. Por lo tanto es conveniente reemplazar la Ec.(B.84) por la siguiente versión estable

$$\widehat{\underline{u}} = R^{-1}\left(\underline{\underline{a}}\underline{u}^{'}\right). \tag{B.85}$$

Sea $\underline{\underline{a}}^{r} : W \to W_r$ el operador de proyección ortogonal de W a W_r y nótese que $\underline{\underline{a}} = \underline{\underline{a}}\underline{\underline{a}}^{r}$, entonces, se define

$$\underline{\underline{A}} \equiv \underline{\underline{a}}^{r}\underline{\underline{A}}^{t}\underline{\underline{a}}^{r}. \tag{B.86}$$

Por otro lado, se tiene que

$$\begin{aligned} \underline{\underline{A}} &\equiv \underline{\underline{a}}^{r}\underline{\underline{A}}^{t}\underline{\underline{a}}^{r} \equiv \begin{pmatrix} \underline{\underline{A}}_{\Pi\Pi} & \underline{\underline{A}}_{\Pi\Delta} \\ \underline{\underline{A}}_{\Delta\Pi} & \underline{\underline{A}}_{\Delta\Delta} \end{pmatrix} \\ &= \begin{pmatrix} \underline{\underline{A}}^{t}_{II} & \underline{\underline{A}}^{t}_{I\pi}\underline{\underline{a}}^{\pi} & \underline{\underline{A}}^{t}_{I\Delta} \\ \underline{\underline{a}}^{\pi}\underline{\underline{A}}^{t}_{\pi I} & \underline{\underline{a}}^{\pi}\underline{\underline{A}}^{t}_{\pi\pi}\underline{\underline{a}}^{\pi} & \underline{\underline{a}}^{\pi}\underline{\underline{A}}^{t}_{\pi\Delta} \\ \underline{\underline{A}}^{t}_{\Delta I} & \underline{\underline{A}}^{t}_{\Delta\pi}\underline{\underline{a}}^{\pi} & \underline{\underline{A}}^{t}_{\Delta\Delta} \end{pmatrix} \end{aligned} \tag{B.87}$$

la notación usada es tal que

$$\begin{cases} \underline{\underline{A}}_{\Pi\Pi} : W_\Pi \to W_\Pi, & \underline{\underline{A}}_{\Pi\Delta} : W_\Delta \to W_\Pi \\ \underline{\underline{A}}_{\Delta\Pi} : W_\Pi \to W_\Delta, & \underline{\underline{A}}_{\Delta\Delta} : W_\Delta \to W_\Delta \end{cases} \tag{B.88}$$

donde

$$\begin{cases} \underline{\underline{A}}_{\Pi\Pi}\underline{u} = \left(\underline{\underline{A}}\underline{u}_\Pi\right)_\Pi, & \underline{\underline{A}}_{\Delta\Pi}\underline{u} = \left(\underline{\underline{A}}\underline{u}_\Pi\right)_\Delta \\ \underline{\underline{A}}_{\Pi\Delta}\underline{u} = \left(\underline{\underline{A}}\underline{u}_\Delta\right)_\Pi, & \underline{\underline{A}}_{\Delta\Delta}\underline{u} = \left(\underline{\underline{A}}\underline{u}_\Delta\right)_\Delta \end{cases} \tag{B.89}$$

por otro lado, se tiene la inmersión natural en W_r, i.e.

$$\begin{aligned}
\underline{\underline{A}}_{\Pi\Pi} &\equiv \begin{pmatrix} \underline{\underline{A}}_{\Pi\Pi} & 0 \\ 0 & 0 \end{pmatrix} \\
\underline{\underline{A}}_{\Pi\Delta} &\equiv \begin{pmatrix} 0 & \underline{\underline{A}}_{\Pi\Delta} \\ 0 & 0 \end{pmatrix} \\
\underline{\underline{A}}_{\Delta\Pi} &\equiv \begin{pmatrix} 0 & 0 \\ \underline{\underline{A}}_{\Delta\Pi} & 0 \end{pmatrix} \\
\underline{\underline{A}}_{\Delta\Delta} &\equiv \begin{pmatrix} 0 & 0 \\ 0 & \underline{\underline{A}}_{\Delta\Delta} \end{pmatrix}.
\end{aligned} \tag{B.90}$$

Así, el 'Complemento de Schur Dual-Primal' está definido por

$$\underline{\underline{S}} = \underline{\underline{A}}_{\Delta\Delta} - \underline{\underline{A}}_{\Delta\Pi}\underline{\underline{A}}^{-1}_{\Pi\Pi}\underline{\underline{A}}_{\Pi\Delta} \tag{B.91}$$

que define al sistema virtual

$$\underline{\underline{S}}\underline{u}_{\Gamma} = \underline{f}_{\Gamma}. \tag{B.92}$$

Nótese que

$$\underline{\underline{A}}_{\Pi\Pi} = \begin{pmatrix} \underline{\underline{A}}_{II} & \underline{\underline{A}}_{I\pi} \\ \underline{\underline{A}}_{\pi I} & \underline{\underline{A}}_{\pi\pi} \end{pmatrix} \tag{B.93}$$

así, se definen a las matrices

$$\underline{\underline{A}}^{\alpha}_{II}, \underline{\underline{A}}^{\alpha}_{I\pi}, \underline{\underline{A}}^{\alpha}_{I\Delta}, \underline{\underline{A}}^{\alpha}_{\pi I}, \underline{\underline{A}}^{\alpha}_{\pi\pi}, \underline{\underline{A}}^{\alpha}_{\pi\Delta}, \underline{\underline{A}}^{\alpha}_{\Delta I}, \underline{\underline{A}}^{\alpha}_{\Delta\pi} \text{ y } \underline{\underline{A}}^{\alpha}_{\Delta\Delta} \tag{B.94}$$

las cuales son locales a cada subdominio. Usando esta metodología se puede construir cada componente de $\overset{\frown}{A}^{\alpha}_{pq}$ tomando los nodos p y q según correspondan.

Por ejemplo para construir $\underline{\underline{A}}^{\alpha}_{\pi I}$, en el subdominio $\overline{\Omega}_{\alpha}$ se construye

$$\underline{\underline{A}}^{\alpha}_{\pi I} = \left(\overset{\frown}{A}^{\alpha}_{pq}\right) \tag{B.95}$$

donde los nodos $p \in \pi$ y $q \in I$ y ambos pertenecen al $\alpha-$ésimo subdominio de Ω.

B.9 El Problema General con Restricciones

Recordando lo visto en la sección (B.6) en lo referente a la definición del problema original Ec.(B.46). Entonces "Un vector $\underline{\hat{u}} \in \widehat{W}$ es solución del problema original, si y sólo si, $\underline{u}' = R\underline{\hat{u}} \in W_r \subset W$ satisface la expresión

$$\underline{\underline{aA}}\underline{u}' = \overline{\underline{f}} \quad \text{y} \quad \underline{\underline{j}}\underline{u}' = 0 \tag{B.96}$$

el vector

$$\overline{\underline{f}} \equiv \left(R\underline{\hat{f}}\right) \in W_{12} \subset W_r \tag{B.97}$$

puede ser escrito como

$$\overline{\underline{f}} \equiv \overline{\underline{f}}_{\Pi} + \overline{\underline{f}}_{\Delta} \tag{B.98}$$

con $\overline{\underline{f}}_{\Pi} \in W_{\Pi}$ y $\overline{\underline{f}}_{\Delta} \in W_{\Delta}$".

Este es el 'problema Dual-Primal formulado en el espacio de vectores derivados'; o simplemente, el problema Dual-Primal-DVS. Recordando, que este problema esta formulado en el espacio W_r del espacio de vectores derivados W, en el cual las restricciones ya han sido incorporadas. Por lo tanto todos los algoritmos que se desarrollan en las siguientes secciones incluyen tales restricciones; en particular, aquellos impuestos por el promedio de los nodos primales.

Un vector $\underline{u}^{'} \in W$ satisface la Ec.(B.96) si y sólo si, satisface la Ec.(B.83). Para derivar este resultado se usa la propiedad de que cuando $\underline{u}^{'} \in W$ y $\underline{\underline{j}}\underline{u}^{'} = 0$, la siguiente relación se satisface

$$\underline{u}^{'} \in W_r \quad \text{y} \quad \underline{\underline{a}} \left(\underline{\underline{a}}^r \underline{\underline{A}}^t \underline{\underline{a}}^r \right) \underline{u}^{'} = \underline{\underline{\underline{a}}} \underline{\underline{A}}^t \underline{u}^{'}. \tag{B.99}$$

En las discusiones posteriores, la matriz $\underline{\underline{A}} : W_r \rightarrow W_r$ se asume invertible, en la mayoría de los casos, este hecho se puede garantizar cuando se toman un número suficientemente grande de nodos primales colocados adecuadamente.

Sea $\underline{u}^{'} \in W_r$ una solución de la Ec.(B.96), entonces $\underline{u}^{'} \in W_{12} \subset W$ necesariamente, ya que $\underline{\underline{j}}\underline{u}^{'} = 0$ y se puede aplicar la inversa de la proyección natural obteniendo

$$\underline{\hat{u}} = \underline{\underline{R}} \underline{u}^{'} \tag{B.100}$$

ya que este problema es formulado en el espacio de vectores derivados; en los algoritmos que se presentan en este trabajo (véase [59]), todas las operaciones son realizadas en dicho espacio; en particular, para realizar los cálculos, nunca se regresa al espacio vectorial original $\widehat{W}$, excepto al final cuando se aplica la Ec.(B.100).

C Formulaciones Dirichlet-Dirichlet y Neumann-Neumann en el Marco del Espacio de Vectores Derivados DVS

A nivel continuo, los algoritmos más estudiados son el Neumann-Neumann y el Dirichlet-Dirichlet. Durante el desarrollo del esquema del espacio de vectores derivados (DVS), se muestra una clara correspondencia entre el procedimiento a nivel continuo, y el procedimiento a nivel discreto (véase [39]). Usando tal correspondencia el resultado desarrollado puede ser resumido de una forma breve y efectiva, como sigue:

1. Algoritmos no Precondicionados

   - El Algoritmo del Complemento de Schur (la formulación Primal del problema Dirichlet-Dirichlet), sección (C.1.1).
   - La formulación Dual del Problema Neumann-Neumann, sección (C.1.2).
   - La formulación Primal del Problema Neumann-Neumann, sección (C.1.3).
   - La segunda formulación Dual del Problema Neumann-Neumann, sección (C.1.4).

2. Algoritmos Precondicionados

   - La Versión DVS del Algoritmo BDDC (la formulación Dirichlet-Dirichlet precondicionada), sección (C.2.1).
   - La Versión DVS del Algoritmo FETI-DP (la formulación Dual precondicionada del problema Neumann-Neumann), sección (C.2.2).
   - La formulación Primal Precondicionada del problema Neumann-Neumann, sección (C.2.3).
   - La segunda formulación Dual Precondicionada del problema Neumann-Neumann, sección (C.2.4).

Todos estos algoritmos están formulados en el espacio vectorial sujeto a restricciones, tal que todos estos son algoritmos con restricciones.

Para la discusión de todo el material de este capítulo se usa la notación de la sección (B.7), en especial la definición de $\underline{\underline{A}}$ en la Ec.(B.87)

$$\underline{\underline{A}} \equiv \begin{pmatrix} \underline{\underline{A}}_{\Pi\Pi} & \underline{\underline{A}}_{\Pi\Delta} \\ \underline{\underline{A}}_{\Delta\Pi} & \underline{\underline{A}}_{\Delta\Delta} \end{pmatrix} \tag{C.1}$$

y la dada por la Ec.(B.91) para la matriz del complemento de Schur

$$\underline{\underline{S}} = \underline{\underline{A}}_{\Delta\Delta} - \underline{\underline{A}}_{\Delta\Pi}\underline{\underline{A}}^{-1}_{\Pi\Pi}\underline{\underline{A}}_{\Pi\Delta}. \tag{C.2}$$

C.1 Algoritmos no Precondicionados

Pese a que los algoritmos no precondicionados carecen de utilidad práctica por su pobre desempeño computacional con respecto a los precondicionados, tienen una gran utilidad teórica y son el camino más sencillo para generar los algoritmos computacionales de los precondicionados, ya que estos permiten probar el marco computacional con algoritmos sencillos y luego sólo se precondicionan para tener el algoritmo final de interés. Se desarrollaron cuatro algoritmos no precondicionados (véase [59]), los cuales se detallan a continuación.

C.1.1 Algoritmo del Complemento de Schur

Sea $\underline{u} = \underline{u}^{'} - \underline{\underline{A}}_{\Pi\Pi}^{-1}\overline{\underline{f}}_{\Pi}$, entonces la Ec.(B.96) del problema general con restricciones —véase sección (B.9)—

$$\underline{\underline{a}}\underline{\underline{A}}\underline{u}^{'} = \overline{\underline{f}} \quad \text{y} \quad \underline{\underline{j}}\underline{u}^{'} = 0 \tag{C.3}$$

es equivalente a: "Dada $\underline{f} = \underline{f}_{\Delta} \in \underline{\underline{a}}W_{\Delta}$, encontrar una $\underline{u}_{\Delta} \in W_{\Delta}$ tal que

$$\underline{\underline{a}}\underline{\underline{S}}\underline{u}_{\Delta} = \underline{f}_{\Delta} \quad \text{y} \quad \underline{\underline{j}}\underline{u}_{\Delta} = 0" \tag{C.4}$$

aquí, $\underline{u} = \underline{u}_{\Pi} + \underline{u}_{\Delta}$, $\underline{f} \equiv \overline{\underline{f}}_{\Delta} - \underline{\underline{A}}_{\Delta\Pi}\underline{\underline{A}}_{\Pi\Pi}^{-1}\overline{\underline{f}}_{\Pi}$ y $\underline{u}_{\Pi} \equiv \underline{\underline{A}}_{\Pi\Pi}^{-1}\underline{\underline{A}}_{\Pi\Delta}\underline{u}_{\Delta}$.

Además, nótese que la matriz del complemento de Schur $\underline{\underline{S}} : W_{\Delta} \to W_{\Delta}$ es invertible cuando $\underline{\underline{A}} : W_r \to W_r$ (véase [58] y [57]).

En el capítulo anterior se mostró la versión continua del problema Dirichlet-Dirichlet no precondicionado, de la cual la Ec.(C.4) es su versión discreta. Por lo tanto este algoritmo pudiera ser llamado 'el algoritmo Dirichlet-Dirichlet no precondicionado'. Sin embargo, en lo que sigue, el algoritmo correspondiente a la Ec.(C.4) será referido como el 'algoritmo del complemento de Schur' ya que es la variante de una de las más simples formas del método de subestructuración la cual se detalla en la sección (4.3).

C.1.2 Formulación Dual del Problema Neumann-Neumann

Para iniciar este desarrollo, se toma la identidad

$$\underline{\underline{a}}\underline{\underline{S}} + \underline{\underline{j}}\underline{\underline{S}} = \underline{\underline{S}} \tag{C.5}$$

esta última ecuación es clara ya que

$$\underline{\underline{a}} + \underline{\underline{j}} = \underline{\underline{I}}. \tag{C.6}$$

Usando las Ecs.(C.4 y C.5) juntas, implican que

$$\underline{\underline{S}}\underline{u}_{\Delta} = \underline{\underline{a}}\underline{\underline{S}}\underline{u}_{\Delta} + \underline{\underline{j}}\underline{\underline{S}}\underline{u}_{\Delta} = \underline{f}_{\Delta} - \underline{\lambda}_{\Delta} \tag{C.7}$$

donde el vector $\underline{\lambda}_{\Delta}$ es definido por

$$\underline{\lambda}_{\Delta} = -\underline{\underline{j}}\underline{\underline{S}}\underline{u}_{\Delta}. \tag{C.8}$$

Por lo tanto, $\underline{\lambda}_\Delta \in \underline{\underline{j}} W_\Delta$. Entonces, el problema de encontrar $\underline{u}_\Delta$ ha sido transformado en uno, en que se debe encontrar 'el multiplicador de Lagrange $\underline{\lambda}_\Delta$', una vez encontrado $\underline{\lambda}_\Delta$ se puede aplicar $\underline{\underline{S}}^{-1}$ a la Ec.(C.7) para obtener

$$\underline{u}_\Delta = \underline{\underline{S}}^{-1}\left(\underline{f}_\Delta - \underline{\lambda}_\Delta\right). \tag{C.9}$$

Además, en la Ec.(C.9), $\underline{u}_\Delta \in \underline{\underline{a}} W_\Delta$, tal que

$$\underline{\underline{j}}\,\underline{\underline{S}}^{-1}\left(\underline{f}_\Delta - \underline{\lambda}_\Delta\right) = 0 \tag{C.10}$$

entonces, $\underline{\lambda}_\Delta \in W_\Delta$ satisface

$$\underline{\underline{j}}\,\underline{\underline{S}}^{-1}\underline{\lambda}_\Delta = \underline{\underline{j}}\,\underline{\underline{S}}^{-1}\underline{f}_\Delta \quad \text{junto con} \quad \underline{\underline{a}}\,\underline{\lambda}_\Delta = 0 \tag{C.11}$$

por lo anterior, $\underline{\underline{j}}\,\underline{\underline{S}}\underline{u}_\Delta$ es la versión discreta de el promedio de la derivada normal (véase [58] y [57]).

Formulación usando Multiplicadores de Lagrange Para obtener la formulación de los Multiplicadores de Lagrange, se escribe

$$\left.\begin{array}{c} J(u) = \frac{1}{2}\underline{u}\cdot\underline{\underline{S}}\underline{u} - \underline{u}\cdot\underline{f} \to \min \\ \underline{\underline{j}}\,\underline{u} = 0 \end{array}\right\} \tag{C.12}$$

tomando la variación en la Ec.(C.12), se obtiene

$$\underline{\underline{S}}\underline{u} + \underline{\underline{j}}\,\underline{\lambda} = \underline{f} \quad \text{junto con} \quad \underline{\underline{j}}\,\underline{u} = 0 \tag{C.13}$$

para asegurar la unicidad de $\underline{\lambda}$, se impone la condición de que $\underline{\underline{a}}\,\underline{u} = 0$, tal que $\underline{\underline{j}}\,\underline{\lambda} = \underline{\lambda}$. Entonces la Ec.(C.13) implica

$$\underline{\underline{a}}\,\underline{\underline{S}}\underline{u} + \underline{\underline{j}}\,\underline{\underline{S}}\underline{u} + \underline{\lambda} = \underline{f} \tag{C.14}$$

multiplicando por $\underline{\underline{j}}$ y $\underline{\underline{a}}$ respectivamente, se obtiene

$$\underline{\lambda} = -\underline{\underline{j}}\,\underline{\underline{S}}\underline{u} \quad \text{y} \quad \underline{\underline{a}}\,\underline{\underline{S}}\underline{u} = \underline{f} \tag{C.15}$$

entonces la Ec.(C.8) es clara.

C.1.3 Formulación Primal del Problema Neumann-Neumann

Para iniciar este desarrollo, se toma la Ec.(C.4) del algoritmo del complemento de Schur —véase sección (C.1.1)— y multiplicando la primera igualdad por $\underline{\underline{S}}^{-1}$, y observando que $\underline{\underline{a}}\,\underline{f}_\Delta = \underline{f}_\Delta$ se obtiene

$$\underline{\underline{S}}^{-1}\underline{\underline{a}}\,\underline{\underline{S}}\underline{u}_\Delta = \underline{\underline{S}}^{-1}\underline{f}_\Delta = \underline{\underline{S}}^{-1}\underline{\underline{a}}\,\underline{f}_\Delta = \underline{\underline{S}}^{-1}\underline{\underline{a}}\,\underline{\underline{S}}\left(\underline{\underline{S}}^{-1}\underline{f}_\Delta\right) \tag{C.16}$$

de este modo, la Ec.(C.4) puede ser transformada en

$$\underline{\underline{S}}^{-1}\underline{\underline{aS}}\left(\underline{u}_\Delta - \underline{\underline{S}}^{-1}\underline{f}_\Delta\right) = 0 \quad \text{y} \quad \underline{\underline{j}}\underline{u}_\Delta = 0 \tag{C.17}$$

o

$$\underline{\underline{aS}}\left(\underline{u}_\Delta - \underline{\underline{S}}^{-1}\underline{f}_\Delta\right) = 0 \quad \text{y} \quad \underline{\underline{j}}\underline{u}_\Delta = 0. \tag{C.18}$$

Si se define

$$\underline{v}_\Delta = \underline{\underline{S}}^{-1}\underline{f}_\Delta - \underline{u}_\Delta \tag{C.19}$$

entonces la Ec.(C.18) es transformada en

$$\underline{\underline{j}}\underline{v}_\Delta = \underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{f}_\Delta \quad \text{y} \quad \underline{\underline{aS}}\underline{v}_\Delta = 0 \tag{C.20}$$

la forma iterativa[51] de este algoritmo se obtiene al multiplicarlo por $\underline{\underline{S}}^{-1}$

$$\underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{v}_\Delta = \underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{f}_\Delta \quad \text{y} \quad \underline{\underline{aS}}\underline{v}_\Delta = 0. \tag{C.21}$$

Si la solución de la Ec.(C.4) es conocida, entonces $\underline{v}_\Delta \in W_\Delta$, definida por la Ec.(C.19), satisface la Ec.(C.21); a la inversa, si $\underline{v}_\Delta \in W_\Delta$ satisface la Ec.(C.21) entonces

$$\underline{u}_\Delta \equiv \underline{\underline{S}}^{-1}\underline{f}_\Delta - \underline{v}_\Delta \tag{C.22}$$

es solución de la Ec.(C.4).

En lo sucesivo, el algoritmo iterativo definido por la Ec.(C.21) será referido como la 'formulación libre de multiplicadores de Lagrange del problema no precondicionado Neumann-Neumann'.

C.1.4 Segunda Formulación Dual del Problema Neumann-Neumann

En este caso, se toma como inicio la Ec.(C.11) —véase sección (C.1.2)—. Nótese que la siguiente identidad se satisface

$$\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\left(\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\right) = \underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1} \tag{C.23}$$

entonces, se multiplica la primera igualdad en la Ec.(C.11) por $\underline{\underline{S}}$, obteniéndose

$$\begin{aligned} \underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\left(\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\right)\underline{\lambda}_\Delta &= \underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{\lambda}_\Delta = \underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\left(\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\right)\underline{f}_\Delta \\ \text{junto con} \quad \underline{\underline{a}}\underline{\lambda}_\Delta &= 0 \end{aligned} \tag{C.24}$$

o

$$\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\left(\left(\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\right)\underline{f}_\Delta - \underline{\lambda}_\Delta\right) \quad \text{junto con} \quad \underline{\underline{a}}\underline{\lambda}_\Delta = 0. \tag{C.25}$$

[51] Para que el algoritmo sea iterativo, se necesita que el dominio y contradominio sean el mismo espacio, que en este caso debe de ser W_Δ.

Si se multiplica la primera de estas igualdades por $\underline{\underline{S}}^{-1}$ y se define

$$\underline{\mu}_\Delta \equiv \underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{f}_\Delta - \underline{\lambda}_\Delta \tag{C.26}$$

la Ec.(C.25) es transformada en

$$\underline{\underline{a}}\underline{\mu}_\Delta = \underline{\underline{a}}\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{f}_\Delta \quad \text{y} \quad \underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{\mu}_\Delta = 0. \tag{C.27}$$

Nótese que esta última ecuación es equivalente a la Ec.(C.25) ya que $\underline{\underline{S}}^{-1}$ es no singular. Si la solución de la Ec.(C.11) es conocida, entonces $\underline{\mu}_\Delta \in W_\Delta$ definida por la Ec.(C.26) satisface la Ec.(C.27). A la inversa, si $\underline{\mu}_\Delta \in W_\Delta$ satisface la Ec.(C.27), entonces

$$\underline{\lambda}_\Delta \equiv \underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{f}_\Delta - \underline{\mu}_\Delta \tag{C.28}$$

es solución de la Ec.(C.11).

Por otro lado, la Ec.(C.27) no define un algoritmo iterativo. Sin embargo, si se multiplica la Ec.(C.27) por $\underline{\underline{S}}$ se obtiene el siguiente algoritmo iterativo

$$\underline{\underline{\underline{S}}\underline{\underline{a}}}\underline{\mu}_\Delta = \underline{\underline{\underline{S}}\underline{\underline{a}}\underline{\underline{S}}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{f}_\Delta \quad \text{y} \quad \underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{\mu}_\Delta = 0 \tag{C.29}$$

esta última ecuación provee una manera iterativa de aplicar la formulación con Multiplicadores de Lagrange. La igualdad $\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{\mu}_\Delta = 0$ puede ser interpretada como una restricción; y en efecto, se puede ver que es equivalente a $\underline{\mu}_\Delta \in \underline{\underline{\underline{S}}\underline{\underline{a}}}W_\Delta$.

C.2 Algoritmos Precondicionados

Los algoritmos precondicionados son los que se implementan para resolver los problemas de interés en Ciencias e Ingenierías, ya que su ejecución en equipos paralelos —como Clusters— es eficiente y con buenas características de convergencia (véase [62]). Se desarrollaron cuatro algoritmos precondicionados (véase [59]), los cuales se detallan a continuación.

C.2.1 Versión DVS del Algoritmo BDDC

La versión DVS del algoritmo BDDC se obtiene cuando el algoritmo de complemento de Schur es precondicionado por medio de la matriz $\underline{\underline{a}}\underline{\underline{S}}^{-1}$. Entonces: "Dada $\underline{f}_\Delta \in \underline{\underline{a}}W_\Delta$, encontrar $\underline{u}_\Delta \in W_\Delta$ tal que

$$\underline{\underline{a}}\underline{\underline{S}}^{-1}\underline{\underline{a}}\underline{\underline{S}}\underline{u}_\Delta = \underline{\underline{a}}\underline{\underline{S}}^{-1}\underline{f}_\Delta \quad \text{y} \quad \underline{\underline{j}}\underline{u}_\Delta = 0\text{".} \tag{C.30}$$

Para este algoritmo, las siguientes propiedades llaman la atención:

1. Este es un algoritmo iterativo.
2. La matriz iterada es $\underline{\underline{a}}\underline{\underline{S}}^{-1}\underline{\underline{a}}\underline{\underline{S}}$.
3. La iteración es realizada dentro del subespacio $\underline{\underline{a}}W_\Delta \subset W_\Delta$.

4. Este algoritmo es aplicable siempre y cuando la matriz del complemento de Schur $\underline{\underline{S}}$ es tal que la implicación lógica

$$\underline{\underline{aS}}^{-1}\underline{w} = 0 \quad \text{y} \quad \underline{\underline{j}}\underline{w} = 0 \Rightarrow \underline{w} = 0 \tag{C.31}$$

es satisfecha, para cualquier $\underline{w} \in W_\Delta$.

5. En particular, es aplicable cuando $\underline{\underline{S}}$ es definida.

Las propiedades 1) a 3) están interrelacionadas. La condición $\underline{\underline{j}}\underline{u}_\Delta = 0$ es equivalente a $\underline{u}_\Delta \in \underline{\underline{a}}W_\Delta$; de este modo, la búsqueda se realiza en el espacio $\underline{\underline{a}}W_\Delta$. Cuando la matriz $\underline{\underline{aS}}^{-1}\underline{\underline{aS}}$ es aplicada repetidamente, siempre se termina en $\underline{\underline{a}}W_\Delta$, ya que para cada $\underline{w} \in W_\Delta$, se obtiene $\underline{\underline{j}}\left(\underline{\underline{aS}}^{-1}\underline{\underline{aS}}\underline{w}\right) = 0$.

Para la propiedad 4), cuando ésta se satisface, la Ec.(C.30) implica la Ec.(C.4). Pare ver esto, nótese que

$$\underline{\underline{j}}\left(\underline{\underline{aS}}\underline{u}_\Delta - \underline{f}_\Delta\right) = 0 \tag{C.32}$$

y también que la Ec.(C.30) implica

$$\underline{\underline{aS}}^{-1}\left(\underline{\underline{aS}}\underline{u}_\Delta - \underline{f}_\Delta\right) = 0 \tag{C.33}$$

cuando la implicación de la Ec.(C.31) se satisface, las Ecs.(C.32 y C.33) juntas implican que

$$\underline{\underline{aS}}\underline{u}_\Delta - \underline{f}_\Delta = 0 \tag{C.34}$$

como se quería. Esto muestra que la Ec.(C.30) implica la Ec.(C.4), cuando la Ec.(C.31) se satisface.

Para la propiedad 5), nótese que la condición de la Ec.(C.31) es débil y requiere que la matriz del complemento de Schur $\underline{\underline{S}}$ sea definida, ya que la implicación de la Ec.(C.31) es siempre satisfecha cuando $\underline{\underline{S}}$ es definida. Asumiendo que $\underline{\underline{S}}$ es definida, entonces para cualquier vector $\underline{w} \in W_\Delta$ tal que $\underline{\underline{aS}}^{-1}\underline{w} = 0$ y $\underline{\underline{j}}\underline{w} = 0$, se obtiene

$$\underline{w}\cdot\underline{\underline{S}}^{-1}\underline{w} = \underline{w}\cdot\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{w} = \left(\underline{\underline{j}}\underline{w}\right)\cdot\underline{\underline{S}}^{-1}\underline{w} = 0 \tag{C.35}$$

esto implica que $\underline{w} = 0$, ya que $\underline{\underline{S}}^{-1}$ es definida cuando $\underline{\underline{S}}$ lo es. Por lo tanto la propiedad 5) es clara.

C.2.2 Versión DVS del Algoritmo FETI-DP

La versión DVS del algoritmo FETI-DP se obtiene cuando la formulación con Multiplicadores de Lagrange del problema no precondicionado Neumann-Neumann

de la Ec.(C.11) —véase sección (C.1.2)— es precondicionada por medio de la matriz $\underline{\underline{jS}}$. Entonces: "Dada $\underline{f}_\Delta \in \underline{\underline{a}}W_\Delta$, encontrar $\underline{\lambda}_\Delta \in W_\Delta$ tal que

$$\underline{\underline{jS}}\,\underline{\underline{jS}}^{-1}\underline{\lambda}_\Delta = \underline{\underline{jS}}\,\underline{\underline{jS}}^{-1}\underline{f}_\Delta \quad \text{y} \quad \underline{\underline{a}}\underline{\lambda}_\Delta = 0" \tag{C.36}$$

donde

$$\underline{u}_\Delta = \underline{\underline{aS}}^{-1}\left(\underline{f}_\Delta - \underline{\underline{j}}\underline{\lambda}_\Delta\right). \tag{C.37}$$

Para este algoritmo, las siguientes propiedades llaman la atención:

1. Este es un algoritmo iterativo.
2. La matriz iterada es $\underline{\underline{jS}}\,\underline{\underline{jS}}^{-1}$.
3. La iteración es realizada dentro del subespacio $\underline{\underline{j}}W_\Delta \subset W_\Delta$.
4. Este algoritmo es aplicable siempre y cuando la matriz del complemento de Schur $\underline{\underline{S}}$ es tal que la implicación lógica

$$\underline{\underline{jS}}\,\underline{w} = 0 \quad \text{y} \quad \underline{\underline{a}}\underline{w} = 0 \Rightarrow \underline{w} = 0 \tag{C.38}$$

es satisfecha, para cualquier $\underline{w} \in W_\Delta$.

5. En particular, es aplicable cuando $\underline{\underline{S}}$ es positiva definida.

Las propiedades 1) a 3) están interrelacionadas. La condición $\underline{\underline{a}}\underline{\lambda}_\Delta = 0$ es equivalente a $\underline{\lambda}_\Delta \in \underline{\underline{j}}W_\Delta$; de este modo, la búsqueda se realiza en el espacio $\underline{\underline{j}}W_\Delta$. Cuando la matriz $\underline{\underline{jS}}\,\underline{\underline{jS}}^{-1}$ es aplicada repetidamente, siempre se termina en $\underline{\underline{j}}W_\Delta$, ya que para cada $\underline{\mu} \in W_\Delta$, se obtiene $\underline{\underline{a}}\left(\underline{\underline{jS}}\,\underline{\underline{jS}}^{-1}\underline{\mu}\right) = 0$.

Para la propiedad 4), cuando esta se satisface, la Ec.(C.36) implica la Ec.(C.11). Pare ver esto, se asume la Ec.(C.36) y nótese que

$$\underline{\underline{a}}\left(\underline{\underline{jS}}^{-1}\underline{\lambda}_\Delta - \underline{\underline{jS}}^{-1}\underline{f}_\Delta\right) = 0 \tag{C.39}$$

y también que la Ec.(C.36) implica

$$\underline{\underline{jS}}\left(\underline{\underline{jS}}^{-1}\underline{\lambda}_\Delta - \underline{\underline{jS}}^{-1}\underline{f}_\Delta\right) = 0 \tag{C.40}$$

cuando la implicación de la Ec.(C.38) se satisface, las Ecs.(C.39 y C.40) juntas implican que

$$\underline{\underline{jS}}^{-1}\underline{\lambda}_\Delta - \underline{\underline{jS}}^{-1}\underline{f}_\Delta = 0 \tag{C.41}$$

como se quería. Esto muestra que la Ec.(C.36) implica la Ec.(C.11), cuando la Ec.(C.38) se satisface.

Para la propiedad 5), nótese que la condición de la Ec.(C.38) es débil y requiere que la matriz del complemento de Schur $\underline{\underline{S}}$ sea definida, ya que la implicación de la Ec.(C.38) es siempre satisfecha cuando $\underline{\underline{S}}$ es definida. Asumiendo

que $\underline{\underline{S}}$ es definida, entonces para cualquier vector $\underline{\mu} \in W_\Delta$ tal que $\underline{\underline{jS}}\underline{\mu} = 0$ y $\underline{\underline{a}}\underline{\mu} = 0$, se obtiene

$$\underline{\mu} \cdot \underline{\underline{S}}\underline{\mu} = \underline{\mu} \cdot \underline{\underline{aS}}\underline{\mu} = \left(\underline{\underline{a}}\underline{\mu}\right) \cdot \underline{\underline{S}}\underline{u} = 0 \tag{C.42}$$

esto implica que $\underline{\mu} = 0$, ya que $\underline{\underline{S}}$ es definida. Por lo tanto la propiedad 5) es clara.

C.2.3 Formulación Primal Precondicionada del Problema Neumann-Neumann

Este algoritmo es la versión precondicionada de la formulación libre de multiplicadores de Lagrange del problema no precondicionado Neumann-Neumann. Este puede derivarse multiplicando la Ec.(C.20) —véase sección (C.1.3)— por el precondicionador $\underline{\underline{S}}^{-1}\underline{\underline{jS}}$. "Entonces el algoritmo consiste en buscar $\underline{v}_\Delta \in W_\Delta$, tal que

$$\underline{\underline{S}}^{-1}\underline{\underline{jS}}\underline{\underline{j}}\underline{v}_\Delta = \underline{\underline{S}}^{-1}\underline{\underline{jS}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{f}_\Delta \quad \text{y} \quad \underline{\underline{aS}}\underline{v}_\Delta = 0" \tag{C.43}$$

donde

$$\underline{u}_\Delta = \underline{\underline{aS}}^{-1}\left(\underline{f}_\Delta - \underline{\underline{jS}}\underline{v}_\Delta\right). \tag{C.44}$$

Para este algoritmo, las siguientes propiedades llaman la atención:

1. Este es un algoritmo iterativo.
2. La matriz iterada es $\underline{\underline{S}}^{-1}\underline{\underline{jS}}\underline{\underline{j}}$.
3. La iteración es realizada dentro del subespacio $\underline{\underline{S}}^{-1}\underline{\underline{j}}W_\Delta \subset W_\Delta$.
4. Este algoritmo es aplicable siempre y cuando la matriz del complemento de Schur $\underline{\underline{S}}$ es tal que la implicación lógica

$$\underline{\underline{jS}}\underline{w} = 0 \quad \text{y} \quad \underline{\underline{a}}\underline{w} = 0 \Rightarrow \underline{w} = 0 \tag{C.45}$$

es satisfecha, para cualquier $\underline{w} \in W_\Delta$.

5. En particular, es aplicable cuando $\underline{\underline{S}}$ es positiva definida.

Las propiedades 1) a 3) están interrelacionadas. La condición $\underline{\underline{aS}}\underline{v}_\Delta = 0$ es equivalente a $\underline{v}_\Delta \in \underline{\underline{jS}}^{-1}W_\Delta$; de este modo, la búsqueda se realiza en el espacio $\underline{\underline{jS}}^{-1}W_\Delta$. Cuando la matriz $\underline{\underline{S}}^{-1}\underline{\underline{jS}}\underline{\underline{j}}$ es aplicada repetidamente, siempre se termina en $\underline{\underline{jS}}^{-1}W_\Delta$, ya que para cada $\underline{v}_\Delta \in W_\Delta$, se obtiene $\underline{\underline{Sa}}\left(\underline{\underline{S}}^{-1}\underline{\underline{jS}}\underline{\underline{j}}\underline{v}_\Delta\right) = 0$.

Para la propiedad 4), cuando esta se satisface, la Ec.(C.43) implica la Ec.(C.20). Pare ver esto, se asume la Ec.(C.43) y se define

$$\underline{w} = \underline{\underline{j}}\underline{v}_\Delta - \underline{\underline{jS}}^{-1}\underline{f}_\Delta \text{ tal que } \underline{\underline{a}}\underline{w} = 0 \tag{C.46}$$

además, en vista de la Ec.(C.43) implica

$$\underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{\underline{S}}\underline{w} = 0 \text{ y por lo tanto } \underline{\underline{j}}\underline{\underline{S}}\underline{w} = 0 \tag{C.47}$$

usando la Ec.(C.45) se ve que las Ecs.(C.46 y C.47) juntas implican que

$$\underline{\underline{j}}\underline{v}_{\Delta} - \underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{f}_{\Delta} = \underline{w} = 0 \tag{C.48}$$

ahora, la Ec.(C.20) es clara y la prueba se completa.

Para la propiedad 5), nótese que la condición de la Ec.(C.45) es débil y requiere que la matriz del complemento de Schur $\underline{\underline{S}}$ sea definida, ya que la implicación de la Ec.(C.45) es siempre satisfecha cuando $\underline{\underline{S}}$ es definida. Asumiendo que $\underline{\underline{S}}$ es definida, entonces para cualquier vector $\underline{w} \in W_{\Delta}$ tal que $\underline{\underline{j}}\underline{\underline{S}}\underline{w} = 0$ y $\underline{\underline{a}}\underline{w} = 0$, se obtiene

$$\underline{w}\cdot\underline{\underline{S}}\underline{w} = \underline{w}\cdot\underline{\underline{a}}\underline{\underline{S}}\underline{w} = \left(\underline{\underline{a}}\underline{w}\right)\cdot\underline{\underline{S}}\underline{w} = 0 \tag{C.49}$$

esto implica que $\underline{w} = 0$, ya que $\underline{\underline{S}}$ es definida. Por lo tanto la propiedad 5) es clara.

C.2.4 Segunda Formulación Dual Precondicionada del Problema Neumann-Neumann

Este algoritmo es la versión precondicionada de la segunda formulación con multiplicadores de Lagrange del problema no precondicionado Neumann-Neumann. Este puede derivarse multiplicando la Ec.(C.27) —véase sección (C.1.4)— por el precondicionador $\underline{\underline{S}}^{-1}\underline{\underline{a}}\underline{\underline{S}}$. "Entonces el algoritmo consiste en buscar $\underline{\mu}_{\Delta} \in W_{\Delta}$, tal que

$$\underline{\underline{S}}\underline{\underline{a}}\underline{\underline{S}}^{-1}\underline{\underline{a}}\underline{\mu}_{\Delta} = \underline{\underline{S}}\underline{\underline{a}}\underline{\underline{S}}^{-1}\underline{\underline{a}}\underline{\underline{S}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{f}_{\Delta} \quad \text{y} \quad \underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{\mu}_{\Delta} = 0" \tag{C.50}$$

donde

$$\underline{u}_{\Delta} = \underline{\underline{a}}\underline{\underline{S}}^{-1}\left(\underline{f}_{\Delta} + \underline{\mu}_{\Delta}\right) \tag{C.51}$$

este algoritmo es similar a FETI-DP.

Para este algoritmo, las siguientes propiedades llaman la atención:

1. Este es un algoritmo iterativo.
2. La matriz iterada es $\underline{\underline{S}}\underline{\underline{a}}\underline{\underline{S}}^{-1}\underline{\underline{a}}$.
3. La iteración es realizada dentro del subespacio $\underline{\underline{S}}\underline{\underline{a}}W_{\Delta} \subset W_{\Delta}$.
4. Este algoritmo es aplicable siempre y cuando la matriz del complemento de Schur $\underline{\underline{S}}$ es tal que la implicación lógica

   $$\underline{\underline{a}}\underline{\underline{S}}^{-1}\underline{w} = 0 \quad \text{y} \quad \underline{\underline{j}}\underline{w} = 0 \Rightarrow \underline{w} = 0 \tag{C.52}$$

   es satisfecha, para cualquier $\underline{w} \in W_{\Delta}$.
5. En particular, es aplicable cuando $\underline{\underline{S}}$ es positiva definida.

Las propiedades 1) a 3) están interrelacionadas. La condición $\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{\mu}_{\Delta} = 0$ es equivalente a $\underline{\mu}_{\Delta} \in \underline{\underline{Sa}}W_{\Delta}$; de este modo, la búsqueda se realiza en el espacio $\underline{\underline{Sa}}W_{\Delta}$. Cuando la matriz $\underline{\underline{SaS}}^{-1}\underline{\underline{a}}$ es aplicada repetidamente, siempre se termina en $\underline{\underline{Sa}}W_{\Delta}$, ya que para cada $\underline{\mu}_{\Delta} \in W_{\Delta}$, se obtiene $\underline{\underline{aS}}\left(\underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{\underline{S}}\underline{\underline{j}}\underline{\mu}_{\Delta}\right) = 0$.

Para la propiedad 4), cuando esta se satisface, la Ec.(C.50) implica la Ec.(C.27). Para ver esto, se asume la Ec.(C.50) y se define

$$\underline{\eta} = \underline{\underline{a}}\underline{\mu}_{\Delta} - \underline{\underline{aS}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{f}_{\Delta} \text{ tal que } \underline{\underline{j}}\underline{\eta} = 0 \tag{C.53}$$

además, en vista de la Ec.(C.50) se obtiene

$$\underline{\underline{SaS}}^{-1}\underline{\eta} = 0 \tag{C.54}$$

usando las Ecs.(C.53 y C.54), juntas implican que

$$\underline{\underline{a}}\underline{\eta} - \underline{\underline{aS}}\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{f}_{\Delta} = \underline{\eta} = 0 \tag{C.55}$$

ahora, la Ec.(C.27) es clara, como se quería probar, además se ve que la Ec.(C.50) implica la Ec.(C.27), cuando la condición de la Ec.(C.52) se satisface.

Para la propiedad 5), nótese que la condición de la Ec.(C.52) es débil y requiere que la matriz del complemento de Schur $\underline{\underline{S}}$ sea definida, ya que la implicación de la Ec.(C.52) es siempre satisfecha cuando $\underline{\underline{S}}$ es definida. Asumiendo que $\underline{\underline{S}}$ es definida, entonces para cualquier vector $\underline{w} \in W_{\Delta}$ tal que $\underline{\underline{aS}}^{-1}\underline{w} = 0$ y $\underline{\underline{j}}\underline{w} = 0$, se obtiene

$$\underline{w} \cdot \underline{\underline{S}}^{-1}\underline{w} = \underline{w} \cdot \underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{w} = \left(\underline{\underline{j}}\underline{w}\right) \cdot \underline{\underline{S}}^{-1}\underline{w} = 0 \tag{C.56}$$

esto implica que $\underline{w} = 0$, ya que $\underline{\underline{S}}^{-1}$ es definida cuando $\underline{\underline{S}}$ lo es. Por lo tanto la propiedad 5) es clara.

C.3 El Operador de Steklov-Poincaré

El operador de Steklov-Poincaré generalmente se define como la ecuación para la traza de la solución exacta u sobre Γ del problema dado por la Ec.(B.4), donde el complemento de Schur —definido en la Ec.(B.92)— es una aproximación discreta del operador de Steklov-Poincaré (véase [40]). La discusión de este operador juega un papel importante en el desarrollo de la teoría del espacio de vectores derivados (véase [39]), para iniciar la discusión, se usa la siguiente definición.

Definición 14 *Sea $\underline{\underline{A}} : W_r \to W_r$ una matriz simétrica y positiva definida. El 'producto interior de energía' es definido por*

$$(\underline{u}, \underline{w}) \equiv \underline{u} \cdot \underline{\underline{A}}\underline{w} \quad \forall \underline{u}, \underline{w} \in W_r. \tag{C.57}$$

El espacio lineal, W_r, es un espacio de Hilbert (dimensionalmente finito) cuando este es dotado con el producto interior de energía. Usando la notación de la sección (B.7) y recordando que la matriz $\underline{\underline{A}}$ se escribe como

$$\underline{\underline{A}} \equiv \begin{pmatrix} \underline{\underline{A}}_{\Pi\Pi} & \underline{\underline{A}}_{\Pi\Delta} \\ \underline{\underline{A}}_{\Delta\Pi} & \underline{\underline{A}}_{\Delta\Delta} \end{pmatrix} \tag{C.58}$$

donde la notación usada es tal que

$$\begin{cases} \underline{\underline{A}}_{\Pi\Pi} : W_\Pi \to W_\Pi, & \underline{\underline{A}}_{\Pi\Delta} : W_\Delta \to W_\Pi \\ \underline{\underline{A}}_{\Delta\Pi} : W_\Pi \to W_\Delta, & \underline{\underline{A}}_{\Delta\Delta} : W_\Delta \to W_\Delta \end{cases} \tag{C.59}$$

además

$$\begin{cases} \underline{\underline{A}}_{\Pi\Pi}\underline{u} = \left(\underline{\underline{A}}\underline{u}_\Pi\right)_\Pi, & \underline{\underline{A}}_{\Delta\Pi}\underline{u} = \left(\underline{\underline{A}}\underline{u}_\Pi\right)_\Delta \\ \underline{\underline{A}}_{\Pi\Delta}\underline{u} = \left(\underline{\underline{A}}\underline{u}_\Delta\right)_\Pi, & \underline{\underline{A}}_{\Delta\Delta}\underline{u} = \left(\underline{\underline{A}}\underline{u}_\Delta\right)_\Delta \end{cases} \tag{C.60}$$

y nótese que se tiene la inmersión natural en W_r, i.e.

$$\begin{aligned} \underline{\underline{A}}_{\Pi\Pi} &\equiv \begin{pmatrix} \underline{\underline{A}}_{\Pi\Pi} & 0 \\ 0 & 0 \end{pmatrix} \\ \underline{\underline{A}}_{\Pi\Delta} &\equiv \begin{pmatrix} 0 & \underline{\underline{A}}_{\Pi\Delta} \\ 0 & 0 \end{pmatrix} \\ \underline{\underline{A}}_{\Delta\Pi} &\equiv \begin{pmatrix} 0 & 0 \\ \underline{\underline{A}}_{\Delta\Pi} & 0 \end{pmatrix} \\ \underline{\underline{A}}_{\Delta\Delta} &\equiv \begin{pmatrix} 0 & 0 \\ 0 & \underline{\underline{A}}_{\Delta\Delta} \end{pmatrix}. \end{aligned} \tag{C.61}$$

Ahora, se introducen las siguientes definiciones:

Definición 15 *Sea la matriz $\underline{\underline{L}}$ definida como*

$$\underline{\underline{L}} \equiv \begin{pmatrix} \underline{\underline{A}}_{\Pi\Pi} & \underline{\underline{A}}_{\Pi\Delta} \\ 0 & 0 \end{pmatrix} \tag{C.62}$$

y la matriz $\underline{\underline{R}}$ definida como

$$\underline{\underline{R}} \equiv \begin{pmatrix} 0 & 0 \\ \underline{\underline{A}}_{\Delta\Pi} & \underline{\underline{A}}_{\Delta\Delta} \end{pmatrix}. \tag{C.63}$$

Además, nótese la siguiente identidad

$$\underline{\underline{R}} = \underline{\underline{aR}} + \underline{\underline{jR}} \tag{C.64}$$

implica también que $\underline{\underline{A}} = \underline{\underline{A}}^T$, así

$$\underline{\underline{L}} + \underline{\underline{aR}} + \underline{\underline{jR}} = \underline{\underline{L}}^T + \underline{\underline{R}}^T\underline{\underline{a}} + \underline{\underline{R}}^T\underline{\underline{j}}. \tag{C.65}$$

Definición 16 *A la identidad*

$$\underline{\underline{L}} + \underline{\underline{aR}} - \underline{\underline{R}}^T \underline{\underline{j}} = \underline{\underline{L}}^T + \underline{\underline{R}}^T \underline{\underline{a}} - \underline{\underline{jR}} \tag{C.66}$$

la cual se deriva de la Ec.(C.65), la cual será referida como la fórmula Green-Herrera para matrices.

Nótese que los rangos de $\underline{\underline{L}}$ y $\underline{\underline{R}}$ son W_{Π} y W_{Δ}, respectivamente, mientras que los rangos de $\underline{\underline{aR}}$ y $\underline{\underline{jR}}$ están contenidos en $W_{12}(\Delta)$ y $W_{11}(\Delta)$ respectivamente. Inclusive más, estos últimos dos rangos son linealmente independientes. Además, para cualquier función $\underline{v} \in W_r$ se obtiene

$$\left(\underline{\underline{L}} + \underline{\underline{aR}} - \underline{\underline{R}}^T \underline{\underline{j}}\right) \underline{v} = 0 \tag{C.67}$$

si y sólo si

$$\underline{\underline{L}}\underline{v} = 0, \underline{\underline{aR}}\underline{v} = 0 \quad \text{y} \quad \underline{\underline{j}}\underline{v} = 0. \tag{C.68}$$

Esto establece la equivalencia entre las Ec.(C.67) y Ec.(C.68) y se puede usar el hecho de que los rangos de $\underline{\underline{L}}$ y $\underline{\underline{aR}}$ son linealmente independientes, junto con la ecuación

$$\left(\underline{\underline{j}}\underline{v}\right) \cdot \underline{\underline{R}}^T \underline{\underline{j}}\underline{v} = \left(\underline{\underline{j}}\underline{v}\right) \cdot \underline{\underline{A}}_{\Delta\Delta} \underline{\underline{j}}\underline{v} = \left(\underline{\underline{j}}\underline{v}\right) \cdot \underline{\underline{A}}\underline{\underline{j}}\underline{v} = 0 \tag{C.69}$$

la cual implica que $\underline{\underline{j}}\underline{v} = 0$. Esto es porque $\underline{\underline{A}}$ es positiva definida sobre W_r.

En lo que sigue, se usa la siguiente notación, para cada $\underline{u} \in W_r$, se escribe

$$\dot{\widehat{\underline{u}}} \equiv \underline{\underline{a}}\underline{u} \quad y \quad [\![\underline{u}]\!] \equiv \underline{\underline{j}}\underline{u} \tag{C.70}$$

entonces $\dot{\widehat{\underline{u}}} \in W_r$, mientras $[\![\underline{u}]\!]$ pertenecen a $W_{11}(\Gamma) \subset W_r$. La fórmula de Green-Herrera de la Ec.(C.66) es equivalente a

$$\underline{w} \cdot \underline{\underline{L}}\underline{u} + \dot{\widehat{\underline{w}}} \cdot \underline{\underline{aR}}\underline{u} - [\![\underline{u}]\!] \underline{\underline{jR}}\underline{w} = \underline{u} \cdot \underline{\underline{L}}\underline{w} + \dot{\widehat{\underline{u}}} \cdot \underline{\underline{aR}}\underline{w} - [\![\underline{w}]\!] \underline{\underline{jR}}\underline{u} \tag{C.71}$$

$\forall \underline{u}, \underline{w} \in W_r$. Ahora, las fórmulas de Green-Herrera que originalmente fueron introducidas para operadores diferenciales parciales actuando sobre funciones discontinuas, pueden ser aplicadas a cualquier operador cuando este es lineal. La Ec.(C.66) por otro lado, es vista como una extensión de este tipo de fórmulas actuando sobre vectores discontinuos y se tiene interés de comparar la Ec.(C.71) con las fórmulas de Green-Herrera para operadores diferenciales parciales. Para hacer esto, se introduce la siguiente notación

$$[\![\underline{\underline{R}}]\!] = -\underline{\underline{aR}} \quad \text{y} \quad \dot{\widehat{\underline{\underline{R}}}} \equiv -\underline{\underline{jR}} \tag{C.72}$$

haciendo uso de esta notación, la Ec.(C.71) se reescribe como

$$\underline{w} \cdot \underline{\underline{L}}\underline{u} + [\![\underline{u}]\!] \cdot \dot{\widehat{\underline{\underline{R}}}}\, \underline{w} - \dot{\widehat{\underline{w}}} \cdot [\![\underline{\underline{R}}]\!]\, \underline{u} = \underline{u} \cdot \underline{\underline{L}}\underline{w} + [\![\underline{w}]\!] \cdot \dot{\widehat{\underline{\underline{R}}}}\, \underline{u} - \dot{\widehat{\underline{u}}} \cdot [\![\underline{\underline{R}}]\!]\, \underline{w} \tag{C.73}$$

$\forall \underline{u}, \underline{w} \in W_r$.

Para el operador diferencial de Laplace actuando sobre funciones discontinuas definidas por tramos que satisfacen condiciones de frontera homogéneas, la fórmula Green-Herrera queda como

$$\int_\Omega w\mathcal{L}u dx + \int_\Gamma \left\{ [\![u]\!] \dot{\widehat{\frac{\partial w}{\partial n}}} - \dot{\widehat{w}} \left[\!\left[\frac{\partial u}{\partial n} \right]\!\right] \right\} dx =$$

$$\int_\Omega u\mathcal{L}w dx + \int_\Gamma \left\{ [\![w]\!] \dot{\widehat{\frac{\partial u}{\partial n}}} - \dot{\widehat{u}} \left[\!\left[\frac{\partial w}{\partial n} \right]\!\right] \right\} dx \tag{C.74}$$

la siguiente correspondencia entre las funcionales bilineales involucradas en ambas ecuaciones, comparando con las Ecs. (C.73 y C.74) se obtiene

$$\begin{array}{c} \int_\Omega w\mathcal{L}u dx \leftrightarrow \underline{w} \cdot \underline{\underline{L}}\underline{u} \\ \int_\Gamma [\![u]\!] \dot{\widehat{\frac{\partial w}{\partial n}}} dx \leftrightarrow [\![\underline{u}]\!] \cdot \dot{\underline{\underline{\widehat{R}}}}\, \underline{w} \\ \int_\Gamma \dot{\widehat{w}} \left[\!\left[\frac{\partial u}{\partial n} \right]\!\right] dx \leftrightarrow \dot{\underline{\widehat{w}}} \cdot [\![\underline{\underline{R}}]\!]\, \underline{u}. \end{array} \tag{C.75}$$

Para operadores diferenciales, en particular para el operador de Laplace, el operador de Steklov-Poincaré asociado con el salto de la derivada normal y de la funcional bilineal es

$$\int_\Gamma \dot{\widehat{w}} \left[\!\left[\frac{\partial u}{\partial n} \right]\!\right] dx \tag{C.76}$$

a nivel matricial, el operador de Steklov-Poincaré es asociado con la forma bilineal

$$\dot{\underline{w}} \cdot [\![\underline{\underline{R}}]\!]\, \underline{u} = \underline{w} \cdot \underline{\underline{aR}}\underline{u}, \forall \underline{u}, \underline{w} \in W_r \tag{C.77}$$

o más simplemente, con la matriz $\underline{\underline{aR}}$.

Definición 17 *Se define al operador de Steklov-Poincaré como la matriz*

$$\underline{\underline{aR}}. \tag{C.78}$$

En el esquema de vectores derivados (DVS), la versión discreta del operador de Steklov-Poincaré es $\underline{\underline{aS}}$, por lo tanto, la versión discreta de la Ec.(B.14) es

$$\underline{\underline{aS}}\underline{v} = \underline{\underline{aS}}\underline{u_p} \qquad \text{junto con } \underline{\underline{j}}\underline{v} = 0 \tag{C.79}$$

la cual puede ponerse en correspondencia con la Ec.(C.4) reemplazando

$$-\left[\!\left[\frac{\partial u_p}{\partial n} \right]\!\right] \leftrightarrow \underline{f}_\Delta \qquad \text{y} \qquad \underline{v} \leftrightarrow \underline{u}_\Delta. \tag{C.80}$$

Otra formulación al operador de Steklov-Poincaré, se deriva de la fórmula de Green-Herrera para el operador elíptico general, simétrico y de segundo orden

$$\int_{\Omega} w\mathcal{L}u dx + \int_{\Gamma} \left\{ [\![u]\!] \, \underline{a}_n \cdot \widehat{\nabla w} - \dot{\widehat{w}} \, [\![\underline{a}_n \cdot \nabla u]\!] \right\} dx =$$

$$\int_{\Omega} u\mathcal{L}w dx + \int_{\Gamma} \left\{ [\![w]\!] \, \underline{a}_n \cdot \widehat{\nabla u} - \dot{\widehat{u}} \, [\![\underline{a}_n \cdot \nabla w]\!] \right\} dx. \tag{C.81}$$

La correspondencia de la ecuación (C.75) todavía permanece para este caso, excepto que

$$\begin{cases} [\![\underline{a}_n \cdot \nabla u]\!] \leftrightarrow - \, [\![\underline{\underline{R}}]\!] \, \underline{u} \\ \dot{\underline{a}}_n \cdot \widehat{\nabla w} \leftrightarrow - \dot{\underline{\underline{\widehat{R}}}} \underline{u} \end{cases} . \tag{C.82}$$

Correspondencias similares a las dadas por las Ecs. (C.75 y C.82) pueden ser establecidas en general —su aplicación incluye a los sistemas gobernantes de ecuaciones de elasticidad lineal y muchos problemas más—. Nótese que las Ecs. (C.75 y C.82) implican una nueva fórmula para el operador *Steklov-Poincaré,* i.e., el salto de la derivada normal en el nivel discreto, el cual es diferente a las interpretaciones estándar que han sido presentadas por muchos autores. La fórmula (véase [57]) para el operador *Steklov-Poincaré* es

$$- \, [\![\underline{\underline{R}}]\!] \, \underline{u} \equiv -\underline{j}\underline{\underline{R}} \tag{C.83}$$

en particular, esta no contiene el lado derecho de la ecuación para ser resuelta; ganando con ello, en consistencia teórica. Nótese que la fórmula es aplicable para cualquier vector (función) independientemente de si está es solución del problema bajo consideración o no.

Aplicando la fórmula de Green-Herrera al problema transformado dado al inicio de este capítulo, se tiene el siguiente resultado:

Teorema 18 *Sea* $\overline{\underline{f}} = \begin{pmatrix} \overline{\underline{f}} \\ \underline{\underline{f}}_{\Pi} \\ \underline{f}_{\Delta} \end{pmatrix} \in W_r = W_{12}\left(\overline{\Omega}\right)$, *entonces una* $\underline{v} \in W_r$ *satisface*

$$\left(\underline{\underline{L}} + \underline{\underline{aR}} - \underline{\underline{R}}^T \underline{j} \right) \underline{v} = \overline{\underline{f}} \tag{C.84}$$

si y sólo si, $\underline{v}$ *es solución del el problema transformado.*

Demostración. Sea $\underline{u} \in W_r$ una solución del problema transformado y asumiendo que $\underline{v} \in W_r$ satisface la Ec.(C.84) entonces

$$\left(\underline{\underline{L}} + \underline{\underline{aR}} - \underline{\underline{R}}^T \underline{j} \right) \underline{u} = \left(\underline{\underline{L}} + \underline{\underline{aR}} \right) \underline{u} = \underline{\underline{aAu}} = \overline{\underline{f}} \tag{C.85}$$

para probar el inverso, se define $\underline{w} = \underline{v} - \underline{u}$, así se obtiene

$$\left(\underline{\underline{L}} + \underline{\underline{aR}} - \underline{\underline{R}}^T \underline{j} \right) \underline{w} = 0 \tag{C.86}$$

y usando las Ec.(C.67) y Ec.(C.68), se tiene que $\underline{v} = \underline{u} + \underline{w}$ satisface

$$\underline{\underline{A}}\underline{v} = \left(\underline{\underline{L}} + \underline{\underline{aR}}\right)\underline{v} = \left(\underline{\underline{L}} + \underline{\underline{aR}}\right)\underline{u} = \underline{\underline{aA}}\underline{u} = \overline{\underline{f}} \tag{C.87}$$

y

$$\underline{\underline{j}}\underline{v} = \underline{\underline{j}}\underline{u} = 0. \tag{C.88}$$

■

Por otro lado, para obtener la versión discreta del operador μ definido en la Ec.(B.21), primero se escribe la versión discreta de la Ec.(B.19), esta es

$$\underline{\underline{j}}\underline{\underline{S}}\underline{v} = \underline{q}_{\Gamma} \qquad \text{junto con} \qquad \underline{\underline{aS}}\underline{v} = 0 \tag{C.89}$$

por lo tanto, se tiene que

$$\underline{\underline{aq}}_{\Gamma} = 0 \text{ y } \underline{\underline{j}}\underline{v} = \underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{\underline{S}}\underline{v} = \underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{\underline{S}}\underline{v} = \underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{q}_{\Gamma} \tag{C.90}$$

esto establece la correspondencia de

$$\mu \leftrightarrow \underline{\underline{j}}\underline{\underline{S}}^{-1} \tag{C.91}$$

la versión discreta de la Ec.(B.22) es

$$\underline{\underline{j}}\underline{\underline{S}}^{-1}\underline{q}_{\Gamma} = -\underline{\underline{j}}\underline{u}_{p} \qquad \text{junto con} \qquad \underline{\underline{aq}}_{\Gamma} = 0. \tag{C.92}$$

Otra opción de abordar la versión discreta de la Ec.(B.20) sin hacer uso de la contraparte del operador de Steklov-Poincaré μ, en el cual, el correspondiente problema es

$$\underline{\underline{j}}\underline{v} = -\underline{\underline{j}}\underline{u}_{p} \qquad \text{y} \qquad \underline{\underline{aS}}\underline{v} = 0 \tag{C.93}$$

sin embargo, esta última ecuación no define un algoritmo iterativo, ya que

$$\underline{\underline{aS}}\underline{\underline{j}} \neq 0. \tag{C.94}$$

Una ecuación equivalente a la Ec.(C.93), la cual puede ser aplicada en un algoritmo iterativo es

$$\underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{v} = -\underline{\underline{S}}^{-1}\underline{\underline{j}}\underline{u}_{p} \qquad \text{y} \qquad \underline{\underline{aS}}\underline{v} = 0. \tag{C.95}$$

D Consideraciones Sobre la Formulación Numérica y su Implementación Computacional de DVS

Para realizar la implementación de cada uno de los métodos desarrollados de descomposición de dominio en el espacio de vectores derivados (DVS) —véase sección (4.4.2)—, es necesario trabajar con los operadores $\underline{\underline{a}}, \underline{\underline{j}}, \underline{\underline{S}}$ y $\underline{\underline{S}}^{-1}$, así como realizar las operaciones involucradas entre ellos.

Normalmente los operadores $\underline{\underline{S}}$ y $\underline{\underline{S}}^{-1}$ no se construyen —son operadores virtuales— porque su implementación generaría matrices densas, las cuales consumen mucha memoria y hacen ineficiente su implementación computacional. En vez de eso, sólo se realizan las operaciones que involucra la definición del operador, por ejemplo $\underline{\underline{S}}\underline{u_\Gamma}$

$$\underline{\underline{S}}\underline{u_\Gamma} = \left(\underline{\underline{A}}_{\Delta\Delta} - \underline{\underline{A}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{\underline{A}}_{\Pi\Delta}\right)\underline{u_\Gamma} \tag{D.1}$$

en donde, para su evaluación sólo involucra la operación de multiplicación matriz-vector y resta de vectores, haciendo la evaluación computacional eficiente.

En el presente apéndice se desarrollan las operaciones que se requieren para implementar a cada uno los operadores involucrados en los métodos desarrollados y que matrices en cada caso son necesarias de construir, siempre teniendo en cuenta el hacer lo más eficiente posible su implementación y evaluación en equipos de cómputo de alto desempeño.

D.1 Matrices Virtuales y Susceptibles de Construir

La gran mayoría de las matrices usadas en los métodos de descomposición de dominio son operadores virtuales, por ejemplo el operador $\underline{\underline{S}}$, que es definido como

$$\underline{\underline{S}} = \underline{\underline{A}}_{\Delta\Delta} - \underline{\underline{A}}_{\Delta\Pi}\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{\underline{A}}_{\Pi\Delta} \tag{D.2}$$

y es formado por

$$\underline{\underline{S}} = \sum_{\alpha=1}^{E}\underline{\underline{S}}^{\alpha} \tag{D.3}$$

donde $\underline{\underline{S}}^{\alpha}$ a su vez está constituida por el complemento de Schur local al subdominio Ω_α

$$\underline{\underline{S}}^{\alpha} = \underline{\underline{A}}^{\alpha}_{\Delta\Delta} - \underline{\underline{A}}^{\alpha}_{\Delta\Pi}\left(\underline{\underline{A}}^{\alpha}_{\Pi\Pi}\right)^{-1}\underline{\underline{A}}^{\alpha}_{\Pi\Delta} \tag{D.4}$$

pero, de nuevo, las matrices locales $\underline{\underline{S}}^{\alpha}$ y $\left(\underline{\underline{A}}^{\alpha}_{\Pi\Pi}\right)^{-1}$ no se construyen ya que $\underline{\underline{A}}^{\alpha}_{\Pi\Pi}\underline{u_\Pi} = \underline{\underline{S}}^{\alpha}_{\pi}\underline{u_\Pi}$ donde $\underline{\underline{S}}^{\alpha}_{\pi}$ es definido como

$$\underline{\underline{S}}^{\alpha}_{\pi} = \underline{\underline{A}}^{\alpha}_{\pi\pi} - \underline{\underline{A}}^{\alpha}_{\pi I}\left(\underline{\underline{A}}^{\alpha}_{II}\right)^{-1}\underline{\underline{A}}^{\alpha}_{I\pi}. \tag{D.5}$$

Entonces, las matrices susceptibles de ser construidas en cada subdominio Ω_α son

$$\underline{\underline{A}}^\alpha_{II}, \underline{\underline{A}}^\alpha_{I\pi}, \underline{\underline{A}}^\alpha_{I\Delta}, \underline{\underline{A}}^\alpha_{\pi I}, \underline{\underline{A}}^\alpha_{\pi\pi}, \underline{\underline{A}}^\alpha_{\pi\Delta}, \underline{\underline{A}}^\alpha_{\Delta I}, \underline{\underline{A}}^\alpha_{\Delta\pi} \text{ y } \underline{\underline{A}}^\alpha_{\Delta\Delta} \tag{D.6}$$

pero $\underline{\underline{A}}^\alpha_{I\pi} = \left(\underline{\underline{A}}^\alpha_{\pi I}\right)^T$, $\underline{\underline{A}}^\alpha_{I\Delta} = \left(\underline{\underline{A}}^\alpha_{\Delta I}\right)^T$ y $\underline{\underline{A}}^\alpha_{\pi\Delta} = \left(\underline{\underline{A}}^\alpha_{\Delta\pi}\right)^T$, así, las únicas matrices susceptibles de construir son

$$\underline{\underline{A}}^\alpha_{II}, \underline{\underline{A}}^\alpha_{I\pi}, \underline{\underline{A}}^\alpha_{I\Delta}, \underline{\underline{A}}^\alpha_{\pi\pi}, \underline{\underline{A}}^\alpha_{\pi\Delta} \text{ y } \underline{\underline{A}}^\alpha_{\Delta\Delta} \tag{D.7}$$

donde todas ellas son locales a cada subdominio Ω_α, con una estructura acorde al tipo de discretización discutida en las secciones (B.8 y 4.4.1) y en este apéndice.

Por lo anterior, nótese que el operador $\underline{\underline{S}}^{-1}$ tampoco se construye, para evaluar $\underline{\underline{S}}^{-1}\underline{u_\Gamma}$ se usa el procedimiento indicado en la sección (D.3).

Otros operadores como son las matrices $\underline{\underline{a}}$ y $\underline{\underline{j}}$ pueden ser o no construidos, pero es necesario recordar que $\underline{\underline{j}} = \underline{\underline{I}} - \underline{\underline{a}}$, así que, en principio $\underline{\underline{j}}$ no sería necesario construir, por otro lado los operadores $\underline{\underline{a}}$ y $\underline{\underline{j}}$ sólo existen en el nodo maestro —donde se controla a los nodos duales y primales y no en los subdominios—; y estas matrices en caso de ser construidas serán dispersas, con pocos valores por renglón —según el número de subdominios que compartan a cada uno de los nodos de la frontera interior— y están sujetas al tipo de triangulación usada en la descomposición de la malla gruesa del dominio.

D.2 Evaluación de la Matriz $\underline{\underline{S}}$ con Nodos Primales Definidos

En todos los casos donde aparece el operador $\underline{\underline{S}}$, ya que, sólo interesa la evaluación de $\underline{\underline{S}}\underline{y_\Gamma}$, i.e. $\underline{v_\Gamma} = \underline{\underline{S}}\underline{y_\Gamma}$, entonces se reescribe al operador $\underline{\underline{S}}$ en términos de sus componentes por subdominio

$$\underline{u_\Gamma} = \left[\sum_{\alpha=1}^{E} \underline{\underline{S}}^\alpha\right] \underline{y_\Gamma} \tag{D.8}$$

para evaluar, el vector $\underline{y_\Gamma}$ se descompone en los subvectores $\underline{y_\Gamma}^\alpha$ correspondientes a cada subdominio $\Omega\alpha$. Así, para evaluar $\underline{\tilde{u}_\Gamma}^\alpha = \underline{\underline{S}}^\alpha \underline{y_\Gamma}^\alpha$ se usa el hecho de que

$$\underline{\underline{S}}^\alpha = \underline{\underline{A}}^\alpha_{\Delta\Delta} - \underline{\underline{A}}^\alpha_{\Delta\Pi}\left(\underline{\underline{A}}^\alpha_{\Pi\Pi}\right)^{-1}\underline{\underline{A}}^\alpha_{\Pi\Delta} \tag{D.9}$$

en donde, se tiene que evaluar

$$\underline{\tilde{u}_\Gamma}^\alpha = \left(\underline{\underline{A}}^\alpha_{\Delta\Delta} - \underline{\underline{A}}^\alpha_{\Delta\Pi}\left(\underline{\underline{A}}^\alpha_{\Pi\Pi}\right)^{-1}\underline{\underline{A}}^\alpha_{\Pi\Delta}\right)\underline{y_\Gamma}^\alpha \tag{D.10}$$

estas evaluaciones en cada subdominio Ω_α pueden realizarse en paralelo.

Para evaluar de forma eficiente esta expresión, se realizan las siguientes operaciones equivalentes

$$\begin{aligned} \underline{x1} &= \underline{\underline{A}}^{\alpha}_{\Delta\Delta}\underline{y_{\Gamma}}^{\alpha} \\ \underline{x2} &= \left(\underline{\underline{A}}^{\alpha}_{\Delta\Pi}\left(\underline{\underline{A}}^{\alpha}_{\Pi\Pi}\right)^{-1}\underline{\underline{A}}^{\alpha}_{\Pi\Delta}\right)\underline{y_{\Gamma}}^{\alpha} \\ \underline{\tilde{u}_{\Gamma}}^{\alpha} &= \underline{x1} - \underline{x2} \end{aligned} \tag{D.11}$$

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

$$\underline{x3} = \underline{\underline{A}}^{\alpha}_{\Pi\Delta}\underline{y_{\Gamma}}^{\alpha} \tag{D.12}$$

con este resultado intermedio se debe calcular

$$\underline{x4} = \left(\underline{\underline{A}}^{\alpha}_{\Pi\Pi}\right)^{-1}\underline{x3} \tag{D.13}$$

pero como no se cuenta con $\left(\underline{\underline{A}}^{\alpha}_{\Pi\Pi}\right)^{-1}$ ya que seria una matriz densa, entonces se multiplica la expresión por $\underline{\underline{A}}^{\alpha}_{\Pi\Pi}$ obteniendo

$$\underline{\underline{A}}^{\alpha}_{\Pi\Pi}\underline{x4} = \underline{\underline{A}}^{\alpha}_{\Pi\Pi}\left(\underline{\underline{A}}^{\alpha}_{\Pi\Pi}\right)^{-1}\underline{x3} \tag{D.14}$$

al simplificar, se obtiene

$$\underline{\underline{A}}^{\alpha}_{\Pi\Pi}\underline{x4} = \underline{x3}. \tag{D.15}$$

Esta última expresión puede ser resuelta usando Gradiente Conjugado o alguna variante de GMRES. Una vez obtenido $\underline{x4}$, se puede calcular

$$\underline{x2} = \underline{\underline{A}}^{\alpha}_{\Delta\Pi}\underline{x4} \tag{D.16}$$

así

$$\underline{\tilde{u}_{\Gamma}}^{\alpha} = \underline{x1} - \underline{x2} \tag{D.17}$$

completando la secuencia de operaciones necesaria para obtener $\underline{\tilde{u}_{\Gamma}}^{\alpha} = \underline{\underline{S}}^{\alpha}\underline{y_{\Gamma}}^{\alpha}$.

Observación 5 *En el caso de la expresión dada por la Ec.(D.15) al aplicar un método iterativo, sólo interesará realizar el producto* $\underline{\underline{A}}^{\alpha}_{\Pi\Pi}\underline{x_{\Pi}}$, *o más precisamente* $\underline{\underline{S}}^{i}_{\pi}\underline{x_{\pi}}$, *donde*

$$\underline{\underline{S}}^{i}_{\pi} = \left(\underline{\underline{A}}^{i}_{\pi\pi} - \underline{\underline{A}}^{i}_{\pi I}\left(\underline{\underline{A}}^{i}_{II}\right)^{-1}\underline{\underline{A}}^{i}_{I\pi}\right) \tag{D.18}$$

entonces si se llama $\underline{x_{\pi}}^{i}$ *al vector correspondiente al subdominio* i, *se tiene*

$$\underline{\tilde{u}_{\pi}}^{i} = \left(\underline{\underline{A}}^{i}_{\pi\pi} - \underline{\underline{A}}^{i}_{\pi I}\left(\underline{\underline{A}}^{i}_{II}\right)^{-1}\underline{\underline{A}}^{i}_{I\pi}\right)\underline{x_{\pi}}^{i} \tag{D.19}$$

para evaluar de forma eficiente esta expresión, se realizan las siguientes operaciones equivalentes

$$\begin{aligned} \underline{u1} &= \underline{\underline{A}}^i_{\pi\pi}\underline{x}_i \\ \underline{u2} &= \left(\underline{\underline{A}}^i_{\pi I}\left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{\underline{A}}^i_{I\pi}\right)\underline{x}_i \\ \underline{\tilde{u}_\Gamma}^i &= \underline{u1} - \underline{u2} \end{aligned} \tag{D.20}$$

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

$$\underline{u3} = \underline{\underline{A}}^i_{I\pi}\underline{x}_i \tag{D.21}$$

con este resultado intermedio se debe calcular

$$\underline{u3} = \left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{u4} \tag{D.22}$$

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

$$\underline{\underline{A}}^i_{II}\underline{u3} = \underline{\underline{A}}^i_{II}\left(\underline{\underline{A}}^i_{II}\right)^{-1}\underline{u4} \tag{D.23}$$

al simplificar, se obtiene

$$\underline{\underline{A}}^i_{II}\underline{u4} = \underline{u3}. \tag{D.24}$$

Esta última expresión puede ser resuelta usando métodos directos —Factorización LU o Cholesky— o iterativos —Gradiente Conjugado o alguna variante de GMRES— cada una de estas opciones tiene ventajas y desventajas desde el punto de vista computacional —como el consumo de memoria adicional o el aumento de tiempo de ejecución por las operaciones involucradas en cada caso— que deben ser evaluadas al momento de implementar el código para un problema particular. Una vez obtenido $\underline{u4}$*, se puede calcular*

$$\underline{u2} = \underline{\underline{A}}^i_{\pi I}\underline{u4} \tag{D.25}$$

así

$$\underline{\tilde{u}_\Gamma}^i = \underline{u1} - \underline{u2} \tag{D.26}$$

completando la secuencia de operaciones necesaria para obtener $\underline{\underline{S}}^i_\pi\underline{x}_i$*.*

D.3 Evaluación de la Matriz $\underline{\underline{S}}^{-1}$ con Nodos Primales Definidos

En los algoritmos desarrollados interviene el cálculo de $\underline{\underline{S}}^{-1}$, dado que la matriz $\underline{\underline{S}}$ no se construye, entonces la matriz $\underline{\underline{S}}^{-1}$ tampoco se construye. En lugar de ello, se procede de la siguiente manera: Se asume que en las operaciones anteriores al producto de $\underline{\underline{S}}^{-1}$, se ha obtenido un vector. Supóngase que es $\underline{v_\Gamma}$, entonces para hacer

$$\underline{u_\Gamma} = \underline{\underline{S}}^{-1}\underline{v_\Gamma} \tag{D.27}$$

se procede a multiplicar por $\underline{\underline{S}}$ a la ecuación anterior

$$\underline{\underline{S}}\underline{u_\Gamma} = \underline{\underline{S}}\underline{\underline{S}}^{-1}\underline{v_\Gamma} \tag{D.28}$$

obteniendo

$$\underline{\underline{S}}\underline{u_\Gamma} = \underline{v_\Gamma} \tag{D.29}$$

es decir, usando algún proceso iterativo —como CGM, GMRES— se resuelve el sistema anterior, de tal forma que en cada iteración de $\underline{u_\Gamma}^i$ se procede como se indicó en la sección del cálculo de $\underline{\underline{S}}$, resolviendo $\underline{\underline{S}}\underline{u_\Gamma} = \underline{v_\Gamma}$ mediante iteraciones de $\underline{u_\Gamma}^{i+1} = \underline{\underline{S}}\underline{u_\Gamma}^i$.

D.4 Cálculo de los Nodos Interiores

La evaluación de

$$\underline{u_\Pi} = -\left(\underline{\underline{A}}_{\Pi\Pi}\right)^{-1}\underline{\underline{A}}_{\Pi\Delta}\underline{u_\Delta} \tag{D.30}$$

involucra de nuevo cálculos locales de la expresión

$$\underline{u_I}^\alpha = -\left(\underline{\underline{A}}^\alpha_{\Pi\Pi}\right)^{-1}\underline{\underline{A}}^\alpha_{\Pi\Delta}\underline{u_\Gamma}^\alpha \tag{D.31}$$

aquí otra vez está involucrado $\left(\underline{\underline{A}}^\alpha_{\Pi\Pi}\right)^{-1}$, por ello se debe usar el siguiente procedimiento para evaluar de forma eficiente esta expresión, realizando las operaciones equivalentes

$$\begin{aligned} \underline{x4} &= \underline{\underline{A}}^\alpha_{\Pi\Delta}\underline{u_\Gamma}^\alpha \\ \underline{u_I}^\alpha &= \left(\underline{\underline{A}}^\alpha_{\Pi\Pi}\right)^{-1}\underline{x4} \end{aligned} \tag{D.32}$$

multiplicando por $\underline{\underline{A}}^\alpha_{\Pi\Pi}$ la última expresión, se obtiene

$$\underline{\underline{A}}^\alpha_{\Pi\Pi}\underline{u_I}^\alpha = \underline{\underline{A}}^\alpha_{\Pi\Pi}\left(\underline{\underline{A}}^\alpha_{\Pi\Pi}\right)^{-1}\underline{x4} \tag{D.33}$$

simplificando, se obtiene

$$\underline{\underline{A}}^\alpha_{\Pi\Pi}\underline{u_I}^\alpha = \underline{x4} \tag{D.34}$$

esta última expresión puede ser resuelta usando métodos directos —como Factorización LU o Cholesky— o mediante métodos iterativos —como Gradiente Conjugado o alguna variante de GMRES— como se indica en la observación (5).

D.5 Descomposición de Schur sin Nodos Primales

En el caso de que en la descomposición no se usen nodos primales, la matriz virtual global $\underline{\underline{A}}$ queda como

$$\underline{\underline{A}} = \begin{pmatrix} \underline{\underline{A}}^1_{II} & \underline{\underline{0}} & \cdots & \underline{\underline{0}} & \underline{\underline{A}}^1_{I\Delta} \\ \underline{\underline{0}} & \underline{\underline{A}}^2_{II} & \cdots & \underline{\underline{0}} & \underline{\underline{A}}^2_{I\Delta} \\ \underline{\underline{0}} & \underline{\underline{0}} & \ddots & \underline{\underline{0}} & \vdots \\ \underline{\underline{0}} & \underline{\underline{0}} & \cdots & \underline{\underline{A}}^E_{II} & \underline{\underline{A}}^E_{I\Delta} \\ \underline{\underline{A}}^1_{\Delta I} & \underline{\underline{A}}^2_{\Delta I} & \cdots & \underline{\underline{A}}^E_{\Delta I} & \underline{\underline{A}}_{\Delta\Delta} \end{pmatrix} \tag{D.35}$$

de donde $\underline{\underline{A}}\underline{x} = \underline{b}$ se implementa como

$$\underline{\underline{A}} \begin{pmatrix} \underline{x}_I \\ \underline{x}_\Delta \end{pmatrix} = \begin{pmatrix} \underline{b}_I \\ \underline{b}_\Delta \end{pmatrix} \tag{D.36}$$

i.e.

$$\left(\sum_{\alpha=1}^{E} \left(\underline{\underline{A}}^\alpha_{\Delta\Delta} - \underline{\underline{A}}^\alpha_{\Delta I} \left(\underline{\underline{A}}^\alpha_{II} \right)^{-1} \underline{\underline{A}}^\alpha_{I\Delta} \right) \right) \underline{x}_\Delta = \underline{b}_\Delta - \sum_{\alpha=1}^{E} \left(\underline{\underline{A}}^\alpha_{\Delta I} \left(\underline{\underline{A}}^\alpha_{II} \right)^{-1} \underline{b}^\alpha_I \right) \tag{D.37}$$

una vez encontrado $\underline{x}_\Delta$, se encuentra $\underline{x}_I$ mediante

$$\underline{x}^\alpha_I = \left(\underline{\underline{A}}^\alpha_{II} \right)^{-1} \left(\underline{b}^\alpha_I - \underline{\underline{A}}^\alpha_{I\Delta} \underline{x}^\alpha_\Delta \right). \tag{D.38}$$

D.6 DVS para Ecuaciones Escalares y Vectoriales

Con el fin de ejemplificación, se supone una ecuación escalar en dos dimensiones definida en un dominio Ω, mediante una descomposición en subdominios usando una malla estructurada cartesiana como se muestra en la figura:

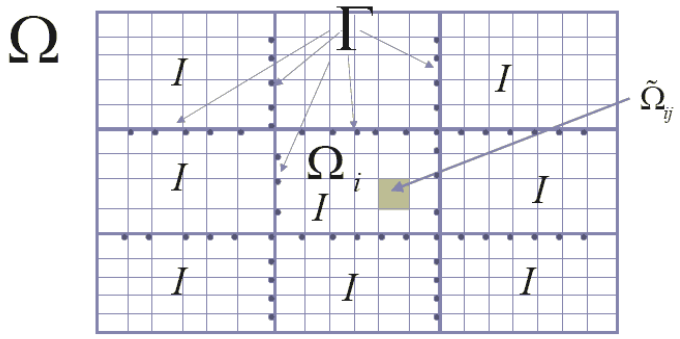

Figura 31: Dominio Ω descompuesto en una partición gruesa de 3×3 y cada subdominio Ω_i en una partición fina de 6×5.

En este caso, sería una descomposición del dominio Ω en 3×3 y 6×5 , la malla gruesa seria descompuesta en $3 \times 3 = 9$ subdominios Ω_α y donde cada

uno de ellos tiene una partición fina de 6×5 i.e. $42 = (6+1)*(5+1)$ grados de libertad por subdominio. En el caso vectorial, en el cual cada grado de libertad contiene C componentes, el número de grados de libertad total será igual a $((6+1)*(5+1)*C)$. Por ejemplo, en el caso de $C = 3$ se tienen 126 grados de libertad por subdominio (véase [63]).

EDP Vectoriales en Dos y Tres Dimensiones En el caso de una ecuación vectorial en dos —tres— dimensiones, si el dominio Ω es descompuesto en una malla estructurada cartesiana, donde la partición gruesa de $n \times m$ —ó $n \times m \times o$— subdominios Ω_α y donde cada uno es descompuesto en una partición fina de $r \times s$ —ó $r \times s \times t$—, en el cual cada grado de libertad contiene C componentes, el número de grados de libertad por subdominio es $(r+1)*(s+1)*C$ —ó $(r+1)*(s+1)*(t+1)*C$—.

A partir de la formulación de los métodos desarrollados, se generan las matrices locales en cada subdominio, de esta forma se obtiene la descomposición fina del dominio, generándose de manera virtual el sistema lineal definido por el método DVS que se este implementando.

La implementación computacional que se desarrolló tiene una jerarquía de clases en donde la clase *DPMethod* realiza la partición gruesa del dominio usando la clase *Interchange* y controla la partición de cada subdominio mediante un objeto de la clase de *RectSub* generando la partición fina del dominio. La resolución de los nodos de la frontera interior se hace mediante el método de CGM o alguna variante de GMRES.

El método de descomposición de dominio en el espacio de vectores derivados se implementó realizando las siguientes tareas:

A) La clase *DPMethod* genera la descomposición gruesa del dominio mediante la agregación de un objeto de la clase *Interchange,* se supone que se tiene particionado en $n \times m$ —ó $n \times m \times o$— subdominios.

B) Con esa geometría se construyen los objetos de *RectSub* —uno por cada subdominio Ω_α—, donde cada subdominio es particionado en $r \times s$ —ó $r \times s \times t$— subdominios y se regresan las coordenadas de los nodos de frontera del subdominio correspondiente a la clase *DPMethod*.

C) Con estas coordenadas, la clase *DPMethod* 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 *RectSub*, transmitiendo sólo aquellos que están en su subdominio.

D) Después de conocer los nodos de la frontera interior, cada objeto *RectSub* calcula las matrices locales sin realizar comunicación alguna. Al terminar de calcular las matrices se avisa a la clase *DPMethod* 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 *DPMethod* y los objetos *RectSub*, se prepara todo lo necesario para empezar el método de CGM o GMRES y resolver el sistema lineal virtual.

F) Para aplicar el método de CGM o GMRES, en cada iteración 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 *DPMethod* y los objetos *RectSub* tantas veces como iteraciones haga el método. Resolviendo con esto los nodos de la frontera interior $\underline{u_\Gamma}_i$.

G) Al término de las iteraciones, se pasa la solución $\underline{u_\Gamma}_i$ de los nodos de la frontera interior que pertenecen a cada subdominio dentro de cada objeto *RectSub* para que se resuelvan los nodos locales al subdominio $\underline{u_\pi}^\alpha = -\left(\underline{\underline{A}}^\alpha_{\Pi\Pi}\right)^{-1} \underline{\underline{A}}^\alpha_{\Pi\Delta} \underline{u_\Gamma}^\alpha$, sin realizar comunicación alguna en el proceso, al concluir se avisa a la clase *DDM* de ello.

I) La clase *DPMethod* mediante un último mensaje avisa que se concluya el programa, terminado así el esquema Maestro-Esclavo.

Análisis de Comunicaciones Para hacer un análisis de las comunicaciones en una ecuación vectorial con C componentes, entre el nodo principal y los nodos esclavos en el método DVS es necesario conocer qué se trasmite y su tamaño, es por ello que en la medida de lo posible se detallan las comunicaciones existentes —hay que hacer mención que entre los nodos esclavos no hay comunicación alguna—.

Tomando la descripción del algoritmo detallado anteriormente, en donde se supuso una partición del dominio Ω con una malla estructurada cartesiana en $n \times m$ —ó $n \times m \times o$ en tres dimensiones— para la malla gruesa y $r \times s$ —ó $r \times s \times t$— para la malla fina, las comunicaciones correspondientes a cada inciso son:

A) El nodo maestro transmite 2 coordenadas en dos —en tres– dimensiones correspondientes a la delimitación del subdominio.

B) $2 * (r + 1) * (s + 1) * C$ —ó $2 * (r + 1) * (s + 1) * (t + 1) * C$— coordenadas transmite cada subdominio al nodo maestro.

C) A lo más $n*m*2*(r+1)*(s+1)*C$ —ó $n*m*o*2*(r+1)*(s+1)*(t+1)*C$— coordenadas son las de los nodos de la frontera interior, y sólo aquellas correspondientes a cada subdominio son trasmitidas por el nodo maestro a los subdominios en los esclavos siendo estas a lo más $2 * (n * m) * (r * s) * C$ —ó $2 * (n * m * o) * (r * s * t) * C$— coordenadas.

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

E) A lo más $2*(r+1)*(s+1)*C$ —ó $2*(r+1)*(s+1)*(t+1)*C$— coordenadas son transmitidas a los subdominios en los nodos esclavos desde el nodo maestro y los nodos esclavos transmiten al nodo maestro esa misma cantidad información.

F) A lo más $2*(r+1)*(s+1)*C$ —ó $2*(r+1)*(s+1)*(t+1)*C$— coordenadas son transmitidas a los subdominios en los nodos esclavos y estos retornan un número igual al nodo maestro por iteración del método de CGM o alguna variante de GMRES.

G) A lo más $2*(r+1)*(s+1)*C$ —ó $2*(r+1)*(s+1)*(t+1)*C$— valores de la solución de la frontera interior son transmitidas a los subdominios en los nodos esclavos desde el nodo maestro y cada objeto transmite un único aviso de terminación.

I) El nodo maestro manda un aviso a cada subdominio en los nodos esclavos para concluir con el esquema.

En todos los casos, 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.

EDP Escalares como un Caso Particular de EDP Vectoriales En el caso de trabajar con ecuaciones escalares en dos o tres dimensiones con una malla estructurada cartesiana, en vez de ecuaciones vectoriales, todo el análisis anterior continua siendo válido y sólo es necesario tomar el número de componentes C igual a uno, manteniéndose la estructura del código intacta.

Esto le da robustez al código, pues permite trabajar con problemas escalares y vectoriales con un pequeño cambio en la estructura del programa, permitiendo resolver una gama más grande de problemas.

Tamaño de las Matrices Locales. Ahora, si se supone que el dominio $\Omega \subset \mathbb{R}^3$ es descompuesto con una malla estructurada cartesiana en una partición gruesa de $n \times m \times o$ subdominios Ω_α y cada subdominio Ω_α es descompuesto en una partición fina de $r \times s \times t$, con número de componentes igual a C, entonces el número de grados de libertad por subdominio es $(r+1)*(s+1)*(t+1)*C$. En la cual se usa un ordenamiento estándar en la numeración de los nodos dentro de cada subdominio

El número de nodos interiores es

$$N^I = (r-1)*(s-1)*(t-1) \tag{D.39}$$

el número de nodos primales, suponiendo que se usa restricción en los vértices es

$$N^\pi = (n-1)*(m-1)*(o-1) \tag{D.40}$$

y el número de nodos duales es a lo más

$$N^\Delta = 2*[((r-1)+(s-1))*(t-1)] \tag{D.41}$$

entonces se tiene que el tamaño y la estructura de cada una de las matrices locales

$$\underline{\underline{A}}^i_{II}, \underline{\underline{A}}^i_{I\pi}, \underline{\underline{A}}^i_{I\Delta}, \underline{\underline{A}}^i_{\pi I}, \underline{\underline{A}}^i_{\pi\pi}, \underline{\underline{A}}^i_{\pi\Delta}, \underline{\underline{A}}^i_{\Delta I}, \underline{\underline{A}}^i_{\Delta\pi} \text{ y } \underline{\underline{A}}^i_{\Delta\Delta}$$

generadas por la descomposición fina del subdominio será:

- $\underline{\underline{A}}^i_{II}$ es una matriz cuadrada de $N^I \times N^I$ nodos, con una estructura bandada
- $\underline{\underline{A}}^i_{I\pi}$ es una matriz rectangular de $N^I \times N^\pi$ nodos, con una estructura dispersa
- $\underline{\underline{A}}^i_{\pi I}$ es una matriz rectangular de $N^\pi \times N^I$ nodos, con una estructura dispersa y transpuesta de $\underline{\underline{A}}^i_{I\pi}$
- $\underline{\underline{A}}^i_{I\Delta}$ es una matriz rectangular de $N^I \times N^\Delta$ nodos, con una estructura dispersa
- $\underline{\underline{A}}^i_{\Delta I}$ es una matriz rectangular de $N^\Delta \times N^I$ nodos, con una estructura dispersa y transpuesta de $\underline{\underline{A}}^i_{I\Delta}$
- $\underline{\underline{A}}^i_{\pi\Delta}$ es una matriz rectangular de $N^\pi \times N^\Delta$ nodos, con una estructura dispersa
- $\underline{\underline{A}}^i_{\Delta\pi}$ es una matriz rectangular de $N^\pi \times N^\Delta$ nodos, con una estructura dispersa y transpuesta de $\underline{\underline{A}}^i_{\pi\Delta}$
- $\underline{\underline{A}}^i_{\pi\pi}$ es una matriz cuadrada de $N^\pi \times N^\pi$ nodos, con una estructura bandada
- $\underline{\underline{A}}^i_{\Delta\Delta}$ es una matriz cuadrada de $N^\Delta \times N^\Delta$ nodos, con una estructura bandada

Para información sobre la estructura de las matrices bandadas véase la sección (A.3.1) y para matrices dispersas véase la sección (A.3.2).

E El Cómputo en Paralelo

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 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 como distribuir estas según la arquitectura en particular con que se trabaje.

E.1 Arquitecturas de Software y Hardware

En esta sección se explican en detalle las dos clasificaciones de computadoras más conocidas en la actualidad. La primera clasificación, es la clasificación clásica de Flynn en dónde se tienen en cuenta sistemas con uno o varios procesadores, la segunda clasificación es moderna en la que sólo tienen en cuenta los sistemas con más de un procesador.

El objetivo de esta sección 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.

E.1.1 Clasificación de Flynn

Clasificación clásica de arquitecturas de computadoras que hace alusión a sistemas con uno o varios procesadores, Flynn 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 de este tipo se caracterizan por tener un único flujo de instrucciones sobre un único flujo de datos, es decir, se ejecuta una instrucción detrás de otra. Este es el concepto de arquitectura serie de Von Neumann donde, en cualquier

momento, sólo se ejecuta una única instrucción, un ejemplo de estos sistemas son las máquinas secuenciales convencionales.

Single Instruction stream, Multiple Data stream (SIMD) Estos sistemas 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.

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 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.

Multiple Instruction stream, Multiple Data stream (MIMD) Sistemas con un flujo de múltiples instrucciones 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).

E.1.2 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).

Multiprocesadores o 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.

Dado que los multiprocesadores comparten diferentes módulos de memoria, pudiendo acceder a un mismo módulo varios procesadores, a los multiprocesadores también se les llama sistemas de memoria compartida.

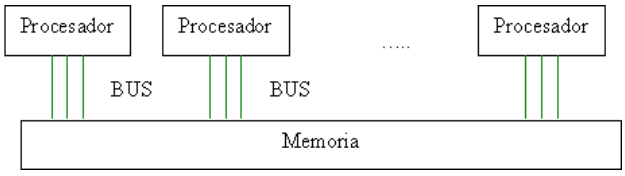

Figura 32: Arquitectura de una computadora paralela con memoria compartida

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.

Uniform Memory Access (UMA) Sistema multiprocesador con acceso uniforme a memoria. La memoria física es uniformemente compartida por todos los procesadores, esto quiere decir que todos los procesadores tienen el mismo tiempo de acceso a todas las palabras de la memoria. Cada procesador tiene su propia caché privada y también se comparten los periféricos.

Los multiprocesadores son sistemas fuertemente acoplados (tightly-coupled), dado el alto grado de compartición de los recursos (hardware o software) y el alto nivel de interacción entre procesadores, lo que hace que un procesador dependa 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 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

  Los sistemas multiprocesador asimétrico, son sistemas con procesadores maestros y procesadores esclavos, en donde sólo los primeros pueden ejecutar aplicaciones y dónde en tiempo de acceso para diferentes procesadores no es el mismo. Los procesadores esclavos (attached) ejecutan código usuario bajo la supervisión del maestro, por lo tanto cuando una aplicación es ejecutada en un procesador maestro dispondrá de una cierta prioridad.

Non Uniform Memory Access (NUMA) Un sistema multiprocesador NUMA es un sistema de memoria compartida donde el tiempo de acceso varía según donde se encuentre localizado el acceso.

El acceso a memoria, por tanto, no es uniforme para diferentes procesadores, existen memorias locales asociadas a cada procesador y estos pueden acceder a datos de su memoria local de una manera más rápida que a las memorias de otros procesadores, debido a que primero debe aceptarse dicho acceso por el procesador del que depende el módulo de memoria local.

Todas las memorias locales conforman la memoria global compartida y físicamente distribuida y accesible por todos los procesadores.

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.

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.

Multicomputadoras o 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.

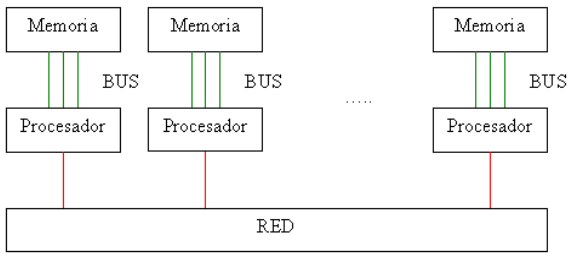

Figura 33: Arquitectura de una computadora paralela con memoria distribuida

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.

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 las ambientes de memoria distribuida. En este modelo se conectan entre si 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 el pase de mensajes (MPI), conexión universal a periféricos (PCI), etc. han hecho posible tomar ventaja de los económicos recursos computacionales de producción masiva (CPU, 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:

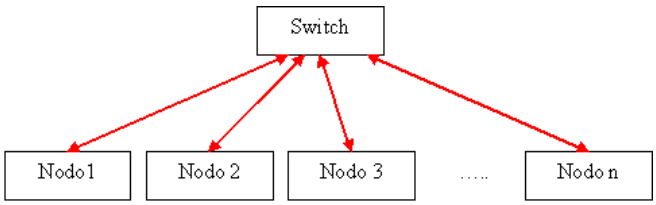

Figura 34: Arquitectura de un cluster

Los cluster se pueden clasificar en dos tipos según sus características físicas:

- Cluster homogéneo si todos los procesadores y/o nodos participantes en el equipo paralelo son iguales en capacidad de cómputo (en la cual 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.

Los cluster 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 cluster 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.
- Remplazar 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 esta 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.
- Alto-rendimiento (HPC o High Performance Computing): este tipo de cluster está 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 esta 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 conjuntamente con el balance de cargas y otra en la parte de hardware que tiene que ver con las características de cada arquitectura que conforman al cúmulo.

E.2 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 y 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

$$s = \frac{T(1)}{T(p)} \tag{E.1}$$

en ambos casos 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

$$e = \frac{T(1)}{pT(p)} = \frac{s}{p}. \tag{E.2}$$

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 el completar el cómputo de la tarea usando un solo procesador

$$f = \frac{T_s}{T(1)} \tag{E.3}$$

pero usando la ley de Amdahl

$$T(p) = T_s + \frac{T_p}{p}$$

y rescribiéndola en términos de factor de aceleración, obtenemos la forma operativa del cálculo de la fracción serial que adquiere la forma siguiente

$$f = \frac{\frac{1}{s} - \frac{1}{p}}{1 - \frac{1}{p}}. \tag{E.4}$$

Esta métrica permite ver las inconsistencias en el balance de cargas, ya que su valor debiera 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.

E.3 Cómputo Paralelo para Sistemas Continuos

Como se mostró en los capítulos anteriores, la solución de los sistemas conti-nuos usando ecuaciones diferenciales parciales genera un alto consumo de memoria e involucran un amplio tiempo de procesamiento; por ello nos interesa trabajar en computadoras que nos 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 corren 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.

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.

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 conexión). Por ello los clusters y los grids son en principio una buena opción para la resolución de este tipo de problemas.

Esquema de Paralelización Maestro-Esclavo La implementación de los métodos de descomposición de dominio que se trabajarán será mediante el esquema Maestro-Esclavo (Farmer) 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, éste modelo puede ser explotado de manera eficiente si existe poca comunicación entre el maestro y el esclavo 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.

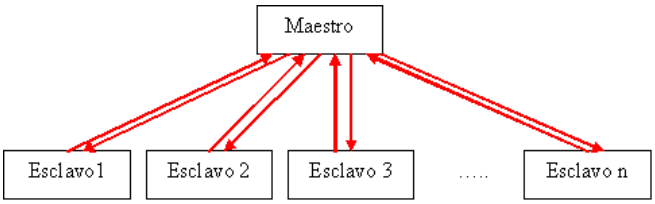

Figura 35: Esquema del maestro-esclavo

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.

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.

Pero 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, algunas observaciones posibles son:

- El esquema maestro-esclavo programado en C++ y usando MPI lanza P procesos (uno para el nodo maestro y $P-1$ para los nodos esclavos), 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 maquina 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 maestro-esclavo no es eficiente si sólo se usan dos procesadores (uno para el nodo maestro y otro para el nodo esclavo), ya que el nodo maestro en general no realiza los cálculos pesados y su principal función será la de distribuir tareas; los cálculos serán delegados al nodo esclavo. En el caso que nos interesa implementar, el método de descomposición de dominio adolece de este problema.

Paso de Mensajes Usando MPI Para poder intercomunicar al nodo maestro con cada uno de los nodos esclavos se usa la interfaz de paso de mensajes (MPI), una biblioteca de comunicación para procesamiento en paralelo. 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 la 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.

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 de tener la dirección de memoria donde se pondrán los datos recibidos, junto con la dirección del proceso del 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 etq 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 etq de la fuente $fuent$ asociado con la comunicación com y el estado st.

El conjunto básico de directivas (en nuestro caso sólo se usan estas) en C++ de MPI son:

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 Maestro-Esclavo La estructura del programa se realizo para que el nodo maestro mande trabajos de manera síncrona a los nodos esclavos. Cuando los nodos esclavos terminan la tarea asignada, avisan al nodo maestro para que se le 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 esclavo y que estos regresen también paquetes al nodo maestro, a manera de reducir las comunicaciones al mínimo y tratar de mantener siempre ocupados a los nodos esclavos para evitar los tiempos muertos, logrando con ello una granularidad gruesa, ideal para trabajar con clusters.

La estructura básica del programa bajo el esquema maestro-esclavo codificada en C++ y usando MPI 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 Maestro
    } else {
        // Operaciones del esclavo con identificador ME_id
    }
    MPI::Finalize();
}
```

En este único programa se deberá de codificar todas las tareas necesarias para el nodo maestro y cada uno de los nodos esclavos, 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 ver [15] y [16].

Los factores limitantes para el esquema maestro-esclavo pueden ser de dos tipos, los inherentes al propio esquema maestro-esclavo y al método de descomposición de dominio:

- El esquema de paralelización maestro-esclavo presupone contar con un nodo maestro lo suficientemente poderoso para atender simultáneamente las tareas síncronas del método de descomposición de dominio, ya que este distribuye tareas acorde al número de subdominios, estas si son balanceadas ocasionaran que todos los procesadores esclavos terminen al mismo tiempo y el nodo maestro tendrá que atender múltiples comunicaciones simultáneamente, degradando su rendimiento al aumentar el número de subdominios.
- Al ser síncrono el método de descomposición de dominio, si un nodo esclavo acaba la tarea asignada y avisa al nodo maestro, este no podrá asignarle otra tarea hasta que todos los nodos esclavos concluyan la suya.

Para los factores limitantes inherente al propio esquema maestro-esclavo, es posible implementar algunas operaciones del nodo maestro en paralelo, ya sea usando equipos multiprocesador o en más de un nodo distintos a los nodos esclavos.

Para la parte inherente al método de descomposición de dominio, la parte medular la da el balanceo de cargas. Es decir que cada nodo esclavo tenga una carga de trabajo igual al resto de los nodos. Este balanceo de cargas puede no ser homogéneo por dos razones:

- Al tener P procesadores en el equipo paralelo, la descomposición del dominio no sea la adecuada.
- Si se tiene una descomposición particular, esta se implemente en un número de procesadores inadecuado.

Cualquiera de las dos razones generarán desbalanceo de la carga en los nodos esclavos, ocasionando una perdida de eficiencia en el procesamiento de un problema bajo una descomposición particular en una configuración del equipo paralelo especifica, es por esto que en algunos casos al aumentar el número de procesadores que resuelvan la tarea no se aprecia una disminución del de tiempo de procesamiento.

El número de procesadores P que se usen para resolver un dominio Ω y tener buen balance de cargas puede ser conocido si aplicamos el siguiente procedimiento: Si el dominio Ω se descompone en $n \times m$ subdominios (la partición gruesa), entonces se generarán $s = n * m$ subdominios Ω_i, en este caso, se tiene un buen balanceo de cargas si $(P-1) \mid s$. La partición fina se obtiene al descomponer a cada subdominio Ω_i en $p \times q$ subdominios.

Como ejemplo, supongamos que deseamos resolver el dominio Ω usando 81×81 nodos ($nodos = n*p+1$ y $nodos = m*q+1$), de manera inmediata nos surgen las siguientes preguntas: ¿cuales son las posibles descomposiciones validas? y ¿en cuantos procesadores se pueden resolver cada descomposición?. Para este ejemplo, sin hacer la tabla exhaustiva obtenemos:

Partición	Subdominios	Procesadores
1x2 y 80x40	2	2,3
1x4 y 80x20	5	2,5
1x5 y 80x16	6	2,6
2x1 y 40x80	2	2,3
2x2 y 40x40	4	2,3,5
2x4 y 40x20	8	2,3,5,9
2x5 y 40x16	10	2,3,6,11
2x8 y 40x10	16	2,3,5,9,17
4x1 y 20x80	4	2,3,5
4x2 y 20x40	8	2,3,5,9
4x4 y 20x20	16	2,3,5,9,17
4x5 y 20x16	20	2,3,5,6,11,21
5x1 y 16x80	5	2,6
5x2 y 16x40	10	2,3,6,11
5x4 y 16x20	20	2,3,5,6,11,21
5x5 y 16x16	25	2,6,26

De esta tabla es posible seleccionar (para este ejemplo en particular), las descomposiciones que se adecuen a las necesidades particulares del equipo con que se cuente. Sin embargo hay que tomar en cuenta siempre el número de nodos por subdominio de la partición fina, ya que un número de nodos muy grande puede que exceda la cantidad de memoria que tiene el nodo esclavo y un número pequeño estaría infrautilizando el poder computacional de los nodos esclavos. De las particiones seleccionadas se pueden hacer corridas de prueba para evaluar su rendimiento, hasta encontrar la que menor tiempo de ejecución consuma, maximizando así la eficiencia del equipo paralelo.

Programación Paralela 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 proceso 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 todo 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 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.

E.4 Infraestructura Computacional 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++ y la interfaz de paso de mensajes (MPI) bajo el esquema Maestro-Esclavo.

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 de 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 comparada con los segundos consumidos en la ejecución.

Para desarrollar estos códigos, se realizó una jerarquía de clases para cada uno de los distintos componentes del sistema de descomposición de dominio en base a clases abstractas, las cuales reducen la complejidad del esquema DVS, permitiendo usarlo tanto en forma secuencial como paralela redefiniendo sólo algunos comportamientos.

La programación, depuración y puesta a punto de los códigos fue hecha en el siguiente equipo:

- Notebook Intel dual Core a 1.7 GHtz con 2 GB de RAM en Linux Debian, haciendo uso del compilador C++ GNU y MPICH para el paso de mensajes.
- PC Intel Quad Core a 2.4 GHtz con 3 GB de RAM en Linux Debian, haciendo uso del compilador C++ GNU y MPICH para el paso de mensajes.

Las pruebas de rendimiento de los distintos programas se realizaron en equipos multiCore y Clusters a los que se tuvo acceso y que están montados

en la Universidad Nacional Autónoma de México, en las pruebas de análisis de rendimiento se usó el siguiente equipo:

- Cluster homogéneo Kanbalam de 1024 Cores AMD Opteron a 2.6 GHtz de 64 bits, cada 4 Cores con 8 GB de RAM interconectados con un switch de 10 Gbps ubicado en el Departamento de Supercómputo de la D.G.C.T.I.C de la UNAM.
- Cluster homogéneo Olintlali de 108 Cores emulando 216 hilos, Intel Xeon a 2.67 GHtz, 9 nodos con 6 Cores y 8 hilos de ejecución con 48 GB RAM interconectados con GIGE 10/100/1000 Gb/s, a cargo del Dr. Ismael Herrera Revilla del Departamento de Recursos Naturales del Instituto de Geofísica de la UNAM.
- Cluster homogéneo Pohualli de 104 Cores Intel Xeon a 2.33 GHtz de 64 bits, cada 8 Cores cuentan con 32 GB de RAM interconectados con un switch de 1 Gbps, a cargo del Dr. Víctor Cruz Atienza del Departamento de Sismología del Instituto de Geofísica de la UNAM.
- Cluster homogéneo IO de 22 Cores Intel Xeon a 2.8 GHtz de 32 bits, cada 2 Cores con 1 GB de RAM interconectados con un switch de 100 Mbps, a cargo de la Dra. Alejandra Arciniega Ceballos del Departamento de Vulcanología del Instituto de Geofísica de la UNAM.
- PC de alto rendimiento Antipolis de 8 Cores Intel Xeon a 2.33 GHtz de 64 bits y 32 GB de RAM, a cargo del Dr. Víctor Cruz Atienza del Departamento de Sismología del Instituto de Geofísica de la UNAM.

F Las Ecuaciones de Navier-Stokes

Las ecuaciones de Navier-Stokes reciben su nombre de los físicos Claude Louis Navier y George Gabriel Stokes (Francés e Irlandés respectivamente), quienes aplicaron los principios de la mecánica y termodinámica, resultando las ecuaciones en derivadas parciales no lineales que logran describir el comportamiento de un fluido. Estas ecuaciones no se concentran en una posición sino en un campo de velocidades, más específicamente en el flujo de dicho campo, lo cual es la descripción de la velocidad del fluido en un punto dado en el tiempo y en el espacio.

Cuando se estudia la dinámica de fluidos se consideran los siguientes componentes:

- $\overrightarrow{\mu}$, este término se usa para definir la velocidad del fluido
- ρ, este término se relaciona con la densidad del fluido
- p, este término hace referencia con la presión o fuerza por unidad de superficie que el fluido ejerce
- g, este término se relaciona con la aceleración de la gravedad, la cuál esta aplicada a todo el cuerpo, en determinados casos la gravedad no es la única fuerza ejercida sobre el cuerpo, por tal razón de toma como la sumatoria de fuerzas externas
- v, este término hace referencia a la viscosidad cinemática

La ecuación de Navier-Stokes general es

$$\rho\left(\frac{\partial v}{\partial t} + v \cdot \nabla v\right) = -\nabla p + \nabla \cdot \mathbb{T} + f \tag{F.1}$$

el término $\mathbb{T}$, expresa el tensor de estrés para los fluidos y condensa información acerca de las fuerzas internas del fluido. Supondremos al fluido incompresible cuyo campo de velocidades $\overrightarrow{\mu}$ satisface la condición $\nabla \cdot \overrightarrow{\mu} = 0$, por tal razón el tensor de estrés se reduce a $\nabla^2 \overrightarrow{\mu}$, dando como resultado la siguiente ecuación:

$$\frac{\partial \overrightarrow{\mu}}{\partial t} + \overrightarrow{\mu} \cdot \nabla \overrightarrow{\mu} + \frac{1}{\rho}\nabla p = g + v\nabla^2 \overrightarrow{\mu} \tag{F.2}$$

bajo la condición

$$\nabla \overrightarrow{\mu} = 0 \tag{F.3}$$

que garantiza la incompresibilidad del campo de velocidad sí la densidad permanece constante en el tiempo.

Así, las ecuaciones simplificadas de Navier-Stokes quedan como

$$\frac{\partial \overrightarrow{\mu}}{\partial t} = -\left(\overrightarrow{\mu}\nabla\right)\overrightarrow{\mu} + v\nabla^2 \overrightarrow{\mu} + f \tag{F.4}$$

$$\frac{\partial \rho}{\partial t} = -\left(\overrightarrow{\mu}\nabla\right)\rho + \kappa\nabla^2\rho + S \tag{F.5}$$

donde la primera ecuación describe la velocidad y la segunda hace referencia a la densidad a través de un campo vectorial. Estas ecuaciones son no lineales y no se conoce solución analítica a dicho problema.

La Solución Numérica de las Ecuaciones de Navier-Stokes La solución numérica de las ecuaciones (F.4, F.5) partiendo del estado inicial $\overrightarrow{\mu}_0$ del campo de velocidad y el primer instante de tiempo $t = 0$, se busca estudiar su comportamiento para el tiempo $t > 0$. Si $\overrightarrow{\omega}_0$ es la estimación del campo vectorial en el instante t y $\overrightarrow{\omega}_4$ denota el campo vectorial de velocidades en el instante de tiempo $t + \Delta t$. Jos Stam (véase [53]) propone una solución numérica a partir de la descomposición de la ecuación de distintos términos y solucionando cada uno individualmente. Así, la solución de cada paso se construye a partir del paso anterior. La solución en el tiempo $t + \Delta t$, está dada por el campo de velocidades $u\left(x, t + \Delta t\right) = \overrightarrow{\omega}_4$, la solución se obtiene iterando estos cuatro pasos:

1. Sumatoria de Fuerzas
2. Paso de Advección
3. Paso de Difusión
4. Paso de Proyección

i.e. $\overrightarrow{\omega}_0 \overset{1}{\overrightarrow{\ }} \overrightarrow{\omega}_1 \overset{2}{\overrightarrow{\ }} \overrightarrow{\omega}_2 \overset{3}{\overrightarrow{\ }} \overrightarrow{\omega}_3 \overset{4}{\overrightarrow{\ }} \overrightarrow{\omega}_4$

Sumatoria de Fuerzas La manera más práctica para incorporar la sumatoria de fuerzas externas f, se obtiene asumiendo que la fuerza no varía de manera considerable en un paso de tiempo. Bajo este supuesto, utilizando el método de Euler progresivo para resolver ecuaciones diferenciales ordinarias (teniendo en cuenta que la velocidad y la fuerza están relacionadas mediante la segunda ley de Newton, tenemos que

$$\overrightarrow{\omega}_1(x) = \overrightarrow{\omega}_0(x) + \Delta t f(x,t). \tag{F.6}$$

Paso de Advección Una perturbación en cualquier parte del fluido se propaga de acuerdo a la expresión

$$-\left(\overrightarrow{\mu}\cdot\nabla\right)\overrightarrow{\mu} \tag{F.7}$$

este término hace que la ecuación de Navier-Stokes sea no lineal, de aplicarse el método de Diferencias Finitas, éste es estable sólo cuando Δt sea suficientemente pequeño tal que $\Delta t < \Delta h / |u|$, donde Δh es el menor incremento de la malla de discretización, para prevenir esto, se usa el método de Características. Este dice que una ecuación de la forma

$$\frac{\partial \alpha(x,t)}{\partial t} = -v(x)\nabla\alpha(x,t) \tag{F.8}$$

y

$$\alpha(x,t) = \alpha_0(x) \tag{F.9}$$

como las características del vector del campo v, que fluye a través del punto x_0 en $t = 0$, i.e.

$$\frac{d}{dt}p(x_0,t) = v(p(x_0,t)) \tag{F.10}$$

donde

$$p(x_0,0) = x_0. \tag{F.11}$$

De modo tal que $\alpha(x_0,t) = \alpha\left(p(x_0,t),t\right)$, es el valor del campo que pasa por el punto x_0 en $t = 0$. Para calcular la variación de esta cantidad en el tiempo se usa la regla de la cadena

$$\frac{d\alpha}{dt} = \frac{\partial\alpha}{\partial t} + v\nabla\alpha = 0 \tag{F.12}$$

que muestra que el valor de α no varía a lo largo de las líneas del flujo. En particular, se tiene que $\alpha(x_0,t) = \alpha(x_0,0) = \alpha_0(x_0)$; por lo tanto el campo inicial de las líneas de flujo, en el sentido inverso se pueden estimar el siguiente valor de α en x_0, esto es $\alpha\left(x_0, t+\Delta t\right)$.

Este método muestra en cada paso del tiempo, como las partículas del fluido se mueven por la velocidad del propio fluido. Por lo tanto, para obtener la velocidad en un punto x en el tiempo $t + \Delta t$, se devuelve al pinto x por el campo de velocidades $\overrightarrow{\omega}_1$ en el paso del tiempo Δt. Ésta define una ruta $p(x,s)$ correspondiente a la trayectoria parcial del campo de velocidades. La nueva velocidad en el punto x, es entonces la velocidad de la partícula ahora en x que tenía su anterior ubicación en el tiempo Δt. De esta manera, se puede hallar el valor de la velocidad luego de la advección $\overrightarrow{\omega}_2$ resolviendo el problema

$$\overrightarrow{\omega}_2(x) = \overrightarrow{\omega}_1(p(x,-\Delta t)) \tag{F.13}$$

la ventaja que se obtiene interpolando linealmente entre valores adyacentes de este método es su estabilidad numérica. Ya conocida la velocidad en el instante t, la viscosidad del fluido y su naturaleza obligan un proceso difusivo: La velocidad se propaga dentro del fluido, o bien en las partículas de este fluyen.

Paso de Difusión En este paso para resolver el efecto de la viscosidad y es equivalente a la ecuación de difusión

$$\frac{\partial\overrightarrow{\omega}_2(x)}{\partial t} = \nu\nabla^2\overrightarrow{\omega}_2(x) \tag{F.14}$$

esta es una ecuación para la cual se han desarrollado varios procedimientos numéricos. La forma más sencilla para resolver esta ecuación es discretizar el operador difusión ∇^2 y usar una forma explicita en el incremento del tiempo, sin embargo este método es inestable cuando la viscosidad es grande. Por ello una mejor opción es usar un método implícito para la ecuación

$$\left(\mathcal{I} - \nu\Delta t\nabla^2\right)\overrightarrow{\omega}_3(x) = \overrightarrow{\omega}_2(x) \tag{F.15}$$

donde $\mathcal{I}$ es el operador identidad.

Paso de Proyección Este último paso conlleva la proyección, que hace que resulte un campo libre de divergencia. Esto se logra mediante la solución del problema definido por

$$\nabla^2 q = \nabla \cdot \vec{\omega}_3(x) \qquad \text{y} \qquad \vec{\omega}_4(x) = \vec{\omega} 3(x) - \nabla q \tag{F.16}$$

es necesario utilizar, según sea el caso, aquel método numérico que proporcione una solución correcta y eficiente a la ecuación de Poisson, esto es especialmente importante cuando hay vórtices en el fluido.

G Implementación Computacional del Método de Diferencias Finitas para la Resolución de Ecuaciones Diferenciales Parciales

Existen diferentes paquetes y lenguajes de programación en los cuales se puede implementar eficientemente la solución numérica de ecuaciones diferenciales parciales mediante el método de Diferencias Finitas, en este capítulo se describe la implementación[52] en los paquetes de cómputo OCTAVE (MatLab), SciLab y en los lenguajesde programación C++, Python, Java, Mono (C#), Fortran y C, estos ejemplos y el presente texto se pueden descargar de la página WEB:

http://mmc.geofisica.unam.mx/acl/MDF/

o desde GitHub[53] (https://github.com/antoniocarrillo69/MDF) mediante

git clone git://github.com/antoniocarrillo69/MDF.git

G.1 Implementación en Octave (MatLab)

GNU OCTAVE[54] es un paquete de cómputo open source para el cálculo numérico —muy parecido a MatLab[55] pero sin costo alguno para el usuario— el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería.

Ejemplo 15 *Sea*

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

entonces el programa queda implementado como:

```
function [A,b,x] = fdm1d(n)
    xi = -1; % Inicio de dominio
    xf = 2; % Fin de dominio
    vi = -1; % Valor en la forntera xi
    vf = 1; % Valor en la frontera xf
    N=n-2; % Nodos interiores
    h=(xf-xi)/(n-1); % Incremento en la malla
    A=zeros(N,N); % Matriz A
    b=zeros(N,1); % Vector b
```

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

[53] GitHub es un plataforma de desarrollo colaborativo para alojar proyectos usando el sistema de control de versiones GIT.

[54] GNU Octave [http://www.gnu.org/software/octave/]

[55] MatLab [http://www.mathworks.com/products/matlab/]

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

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

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

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

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

Ejemplo 16 *Sea*

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

entonces el programa queda implementado como:

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

```
    xx(i)=xi+h*(i-1);
    zz(i)=SolucionAnalitica(xx(i));
  end
  yy=zeros(n,1);
  yy(1)=vi; % Condicion inicial
  for i=1:N
    yy(i+1)=x(i);
  end
  yy(n)=vf; % Condicion inicial
  % Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
  plot(xx,[yy,zz]);
endfunction
function y=LadoDerecho(x)
  y=-pi*pi*cos(pi*x);
endfunction
function y=SolucionAnalitica(x)
  y=cos(pi*x);
endfunction
```

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

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

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

G.2 Implementación en SciLab

Scilab[56] es un paquete de cómputo open source para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería.

Ejemplo 17 *Sea*

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

El programa queda implementado como:

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

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

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

Si el programa lo grabamos en el directorio de trabajo con el nombre **fdm1d.sci**, entonces para poder ejecutar la función es necesario cargarla en la consola de SCILAB mediante

exec('./fdm1d.sci',-1)

para después ejecutar la función mediante:

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

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

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

Ejemplo 18 *Sea*

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

El programa queda implementado como:

```
function [A,b,x] = fdm1d(xi,xf,vi,vf,n)
   N=n-2; // Nodos interiores
   h=(xf-xi)/(n-1); // Incremento en la malla
   A=zeros(N,N); // Matriz A
   b=zeros(N,1); // Vector b
```

```
  R=1/(h^2);
  P=-2/(h^2);
  Q=1/(h^2);
  // Primer renglon de la matriz A y vector b
  A(1,1)=P;
  A(1,2)=Q;
  b(1)=LadoDerecho(xi)-vi*R;
  // Renglones intermedios de la matriz A y vector b
  for i=2:N-1
    A(i,i-1)=R;
    A(i,i)=P;
    A(i,i+1)=Q;
    b(i)=LadoDerecho(xi+h*(i-1));
  end
  // Renglon final de la matriz A y vector b
  A(N,N-1)=R;
  A(N,N)=P;
  b(N)=LadoDerecho(xi+h*N)-vf*Q;
  // Resuleve el sistema lineal Ax=b
  x=inv(A)*b;
  // Prepara la graficacion
  xx=zeros(n,1);
  zz=zeros(n,1);
  for i=1:n
    xx(i)=xi+h*(i-1);
    zz(i)=SolucionAnalitica(xx(i));
  end
  yy=zeros(n,1);
  yy(1)=vi; // Condicion inicial
  for i=1:N
    yy(i+1)=x(i);
  end
  yy(n)=vf; // Condicion inicial
  // Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
  plot2d(xx,[yy,zz]);
endfunction
function y=LadoDerecho(x)
  y=-%pi*%pi*cos(%pi*x);
endfunction
function y=SolucionAnalitica(x)
  y=cos(%pi*x);
endfunction
```

Si el programa lo grabamos en el directorio de trabajo con el nombre **fdm1d.sci**, entonces para poder ejecutar la función es necesario cargarla en la consola de SCILAB mediante

exec('./fdm1d.sci',-1)

para después ejecutar la función mediante:

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

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

Observación 6 *La diferencia entre los códigos de OCTAVE y SCILAb al menos para estos ejemplos estriba en la forma de indicar los comentarios y la manera de llamar para generar la gráfica, además de que en SCILAB es necesario cargar el archivo que contiene la función.*

Ejemplo 19 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \qquad 0 \leq x \leq 0.5, \qquad u(0) = 1, \qquad u'(0.5) = -\pi$$

entonces el programa queda implementado como:

```
function y=LadoDerecho(x)
 y=-%pi*%pi*cos(%pi*x);
endfunction

function y=SolucionAnalitica(x)
 y=cos(%pi*x);
endfunction

a=0; // Inicio dominio
c=0.5; // Fin dominio
M=40; // Partición
N=M-1; // Nodos interiores
h=(c-a)/(M-1); // Incremento en la malla
Y0=1; // Condicion Dirchlet inicial en el inicio del dominio
Y1=-%pi; // Condicion Neumann inicial en el fin del dominio
A=zeros(N,N); // Matriz A
b=zeros(N); // Vector b

R=1/(h^2);
P=-2/(h^2);
Q=1/(h^2);

// Primer renglon de la matriz A y vector b
```

```
A(1,1)=P;
A(1,2)=Q;
b(1)=LadoDerecho(a)-Y0*R; // Frontera Dirichlet
// Renglones intermedios de la matriz A y vector b
for i=2:N-1
 A(i,i-1)=R;
 A(i,i)=P;
A(i,i+1)=Q;
 b(i)=LadoDerecho(a+h*(i-1));
end
// Relglon final de la matriz A y vector b
A(N,N-1)=-1/(h^2);
A(N,N)=-1/(h^2);
b(N)=Y1/h; // Frontera Neumann

// Resuleve el sistema lineal Ax=b
x=inv(A)*b;

// Prepara la graficación
xx=zeros(M,1);
zz=zeros(M,1);
for i=1:M
 xx(i)=a+h*(i-1);
 zz(i)=SolucionAnalitica(xx(i));
end
yy=zeros(M,1);
yy(1)=Y0; // Condicion inicial
for i=1:N
 yy(i+1)=x(i);
end
// Grafica la solución de la Ecuacion Diferencial Parcial en 1D
plot2d(xx,[yy,zz])
```

Ejemplo 20 *Sea*

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

entonces el programa queda implementado como:

```
a=0; // Inicio dominio
c=1; // Fin dominio
M=50; // Partición
N=M-2; // Nodos interiores
h=(c-a)/(M-1); // Incremento en la malla
Y0=0; // Condicion inicial en el inicio del dominio
Y1=1; // Condicion inicial en el fin del dominio
```

```
A=zeros(N,N); // Matriz A
b=zeros(N); // Vector b


P=2/(h^2);
Q=-1/(h^2)+1/(2*h);
R=-1/(h^2)-1/(2*h);


// Primer renglon de la matriz A y vector b
A(1,1)=P;
A(1,2)=Q;
b(1)=-Y0*R;
// Renglones intermedios de la matriz A y vector b
for i=2:N-1
 A(i,i-1)=R;
 A(i,i)=P;
 A(i,i+1)=Q;
end
// Relglon final de la matriz A y vector b
A(N,N-1)=R;
A(N,N)=P;
b(N)=-Y1*Q;

// Resuleve el sistema lineal Ax=b
x=inv(A)*b;

// Prepara la graficación
xx=zeros(M,1);
for i=1:M
 xx(i)=a+h*(i-1);
end
yy=zeros(M,1);
yy(1)=Y0; // Condicion inicial
for i=1:N
 yy(i+1)=x(i);
end
yy(M)=Y1; // Condicion inicial
// Grafica la solución de la Ecuacion Diferencial Parcial en 1D
plot2d(xx,yy)
```

Ejemplo 21 *Sea*

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

entonces el programa queda implementado como:

```
function y=LadoDerecho(x)
 y=-%pi*%pi*cos(%pi*x);
endfunction

function y=SolucionAnalitica(x)
 y=cos(%pi*x);
endfunction

a=-1; // Inicio dominio
c=2; // Fin dominio
M=100; // Partición
N=M-2; // Nodos interiores
h=(c-a)/(M-1); // Incremento en la malla
Y0=-1; // Condicion inicial en el inicio del dominio
Y1=1; // Condicion inicial en el fin del dominio

A=zeros(N,N); // Matriz A
b=zeros(N); // Vector b
R=1/(h^2);
P=-2/(h^2);
Q=1/(h^2);
// Primer renglon de la matriz A y vector b
A(1,1)=P;
A(1,2)=Q;
b(1)=LadoDerecho(a)-Y0*R;
// Renglones intermedios de la matriz A y vector b
for i=2:N-1
 A(i,i-1)=R;
 A(i,i)=P;
 A(i,i+1)=Q;
 b(i)=LadoDerecho(a+h*(i-1));
end
// Relglon final de la matriz A y vector b
A(N,N-1)=R;
A(N,N)=P;
b(N)=LadoDerecho(a+h*N)-Y1*Q;

// Resuleve el sistema lineal Ax=b
x=inv(A)*b;

// Prepara la graficación
xx=zeros(M,1);
zz=zeros(M,1);
for i=1:M
 xx(i)=a+h*(i-1);
```

```
 zz(i)=SolucionAnalitica(xx(i));
end
yy=zeros(M,1);
yy(1)=Y0; // Condicion inicial
for i=1:N
 yy(i+1)=x(i);
end
yy(M)=Y1; // Condicion inicial
// Grafica la solución de la Ecuacion Diferencial Parcial en 1D
plot2d(xx,[yy,zz])
```

Ejemplo 22 *Sea*

$$u''(x) = -\pi^2 \cos(\pi x), \qquad 0 \leq x \leq 0.5, \qquad u(0) = 1, \qquad u'(0.5) = -\pi$$

entonces el programa queda implementado como:

```
function y=LadoDerecho(x)
 y=-%pi*%pi*cos(%pi*x);
endfunction

function y=SolucionAnalitica(x)
 y=cos(%pi*x);
endfunction

a=0; // Inicio dominio
c=0.5; // Fin dominio
M=40; // Partición
N=M-1; // Nodos interiores
h=(c-a)/(M-1); // Incremento en la malla
Y0=1; // Condicion Dirchlet inicial en el inicio del dominio
Y1=-%pi; // Condicion Neumann inicial en el fin del dominio
A=zeros(N,N); // Matriz A
b=zeros(N); // Vector b

R=1/(h^2);
P=-2/(h^2);
Q=1/(h^2);

// Primer renglon de la matriz A y vector b
A(1,1)=P;
A(1,2)=Q;
b(1)=LadoDerecho(a)-Y0*R; // Frontera Dirichlet
// Renglones intermedios de la matriz A y vector b
for i=2:N-1
 A(i,i-1)=R;
```

```
 A(i,i)=P;
A(i,i+1)=Q;
 b(i)=LadoDerecho(a+h*(i-1));
end
// Relglon final de la matriz A y vector b
A(N,N-1)=-1/(h^2);
A(N,N)=-1/(h^2);
b(N)=Y1/h; // Frontera Neumann

// Resuleve el sistema lineal Ax=b
x=inv(A)*b;

// Prepara la graficación
xx=zeros(M,1);
zz=zeros(M,1);
for i=1:M
 xx(i)=a+h*(i-1);
 zz(i)=SolucionAnalitica(xx(i));
end
yy=zeros(M,1);
yy(1)=Y0; // Condicion inicial
for i=1:N
 yy(i+1)=x(i);
end
// Grafica la solución de la Ecuacion Diferencial Parcial en 1D
plot2d(xx,[yy,zz])
```

G.3 Implementación en C++

GMM++[57] es una librería para C++ que permite definir diversos tipos de matrices y vectores además operaciones básicas de algebra lineal. La facilidad de uso y la gran cantidad de opciones hacen que GMM++ sea una buena opción para trabajar con operaciones elementales de algebra lineal.

Se instala en Debian Linux y/o Ubuntu como:

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

Para compilar el ejemplo usar:

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

Para ejecutar usar:

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

Ejemplo 23 *Sea*

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

entonces el programa queda implementado como:

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

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

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

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

G.4 Implementación en Python

Python[58] es un lenguaje de programación interpretado, usa tipado dinámico y es multiplataforma cuya filosofía hace hincapié en una sintaxis que favorezca el código legible y soporta programación multiparadigma —soporta orientación a objetos, programación imperativa y programación funcional—, además es desarrollado bajo licencia de código abierto.

Ejemplo 24 *Sea*

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

entonces el programa queda implementado como:

```
from math import exp
from scipy import sparse
import numpy as np
import matplotlib.pyplot as plt
# Ecuación
# u_ t + 2 u_ x = 0
coef_ a = 2
```

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

```
def condicion_inicial (x):
"""
Condición inicial de la ecuación
"""
y = exp (-(x - 0.2)*(x - 0.02))
return y
def sol_analitica (x, t):
"""
Solución analítica de la ecuación
"""
y = exp(-(x-2*t)*(x-2*t))
return y
############
# Dominio
############
a = -2.0
b = 8.0
# Partición del dominio
nNodos = 401
h = (b - a)/(nNodos - 1)
# Intervalo de tiempo
dt = 0.012
# creo el vector w donde se guradarÃ¡ la soluciÃ³n para cada tiempo
# B matriz del lado derecho
w = np.zeros((nNodos,1))
B = np.zeros((nNodos, nNodos))
B = np.matrix(B)
espacio = np.zeros((nNodos,1))
for i in xrange(nNodos):
xx_ = a + i*h
espacio[i] = xx_
w[i] = condicion_inicial(xx_)
print "Espacio"
print espacio
print "Condición Inicial"
print w
mu = coef_a * dt / h
if mu <= 1:
print "mu ", mu
print "Buena aproximaciÃ³n"
else:
print "mu ", mu
print "Mala aproximaciÃ³n"
if coef_a >= 0:
B[0,0] = 1 - mu
for i in xrange (1, nNodos):
```

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

G.5 Implementación en Java

Java es un lenguaje de programación orientado a objetos de propósito general. concurrente y multiplataforma desarrollado bajo licencia de código abierto. El lenguaje no está diseñado especificamente para cómputo científico, pero es facil implementar lo necesario para nuestros fines.

Sea

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

Ejemplo 25 *El programa queda implementado en JAVA como:*

```
public class fdm1d {
      private int Visualiza;
```

```
// Constructor
public fdm1d() {
      Visualiza = 1;
}
// Resuelve Ax=b usando el metodo Jacobi
public void Jacobi(double[][] A, double[] x, double[] b, int n, int iter) {
      int i, j, m;
      double sum;
      double[] xt = new double [n];
     for (m = 0; m < iter; m ++) {
           for (i = 0; i < n; i++) {
                 sum = 0.0;
                 for (j = 0; j < n; j ++) {
                       if ( i == j) continue;
                             sum += A[i][j] * x[j];
                 }
                 if (A[i][i] == 0.0) return;
                 xt[i] = (1.0 / A[i][i]) * (b[i] - sum);
           }
           for (i = 0; i < n; i++) x[i] = xt[i];
      }
      if (Visualiza != 0) {
           System.out.println(" ");
           System.out.println("Matriz");
           for (i = 0; i < n; i++) {
                 for (j = 0; j < n; j++) {
                       System.out.print(A[i][j] + " ");
                 }
                 System.out.println(" ")
           }
           System.out.println(" ");
           System.out.println("b");
           for (i = 0; i < n; i++) {
                 System.out.print(b[i] + " ");
           }
           System.out.println(" ");
           System.out.println("x");
           for (i = 0; i < n; i++) {
                 System.out.print(x[i] + " ");
           }
           System.out.println(" ");
      }
}
// Resuelve Ax=b usando el metodo Gauus-Siedel
public void Gauss_Siedel(double[][] A, double[] x, double[] b, int n, int
iter) {
```

```
        int i, j, m;
        double sum;
       for (m = 0; m < iter; m ++) {
            for (i = 0; i < n; i++) {
                sum = 0.0;
                for (j = 0; j < n; j ++) {
                    if ( i == j) continue;
                    sum += A[i][j] * x[j];
                }
                if (A[i][i] == 0.0) return;
               x[i] = (1.0 / A[i][i]) * (b[i] - sum);
           }
        }
        if (Visualiza != 0) {
            System.out.println(" ");
            System.out.println("Matriz");
            for (i = 0; i < n; i++) {
                for (j = 0; j < n; j++) {
                    System.out.print(A[i][j] + " ");
                }
                System.out.println(" ");
            }
            System.out.println(" ");
            System.out.println("b");
            for (i = 0; i < n; i++) {
                System.out.print(b[i] + " ");
            }
            System.out.println(" ");
            System.out.println("x");
            for (i = 0; i < n; i++) {
                System.out.print(x[i] + " ");
            }
            System.out.println(" ");
        }
    }
    // Lado derecho de la ecuacion diferencial parcial
    public static double LadoDerecho(double x) {
        double pi = 3.14159265358979323846264338332;
        return -pi * pi * java.lang.Math.cos(pi * x);
    }
    // Funcion Principal ....
    public static void main(String[] args) {
        double xi = -1.0; // Inicio del diminio
        double xf = 2.0; // Fin del dominio
        double vi = -1.0; // Valor en la frontera xi
        double vf = 1.0; // Valor en la frontera xf
```

```
            int n = 11; // Particion
            int N = n - 2; // Nodos interiores
            double h = (xf - xi) / (n - 1); // Incremento en la malla
            double[][] A = new double[N][N]; // Matriz A
            double[] b = new double[N]; // Vector b
            double[] x = new double[N]; // Vector x
            double R = 1 / (h * h);
            double P = -2 / (h * h);
            double Q = 1 / (h * h);
            // Primer renglon de la matriz A y vector b
            A[0][0] = P;
            A[0][1] = Q;
            b[0] = LadoDerecho(xi) - vi * R;
            // Renglones intermedios de la matriz A y vector b
            for (int i = 1; i < N - 1; i++) {
                  A[i][i - 1] = R;
                  A[i][i] = P;
                  A[i][i + 1] = Q;
                  b[i] = LadoDerecho(xi + h * i);
            }
            // Renglon final de la matriz A y vector b
            A[N - 1][N - 2] = R;
            A[N - 1][N - 1] = P;
            b[N - 1] = LadoDerecho(xi + h * N - 2) - vf * Q;
            // Resuleve el sistema lineal Ax=b
            fdm1d ejem = new fdm1d();
            ejem.Gauss_Siedel(A, x, b, N, 1000);
            ejem.Jacobi(A, x, b, N, 1000);
      }
}
```

G.6 Implementación en MONO (C#)

Mono (C#) es un lenguaje de programación orientado a bojetos creado para desarrollar un grupo de herramientas libres basadas en GNU/Linux y compatibles con .NET (C#), no está especialmente desarrollado para cómputo científico, pero es facil implementar lo necesario para nuestros fines.

Sea

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

Ejemplo 26 *El programa queda implementado en MONO como:*

```
using System;
using System.IO;
using System.Text;
namespace FDM1D
```

```
{
public class fdm1d {
        private int Visualiza;
        // Constructor
        public fdm1d() {
              Visualiza = 1;
        }
              // Resuelve Ax=b usando el metodo Jacobi
             public void Jacobi(double[,] A, double[] x, double[] b, int n, int iter)
{
              int i, j, m;
              double sum;
                   double[] xt = new double [n];
                   for (m = 0; m < iter; m ++) {
                         for (i = 0; i < n; i++) {
                          sum = 0.0;
                          for (j = 0; j < n; j ++) {
                                if ( i == j) continue;
                                sum += A[i,j] * x[j];
                          }
                          if (A[i,i] == 0.0) return;
                          xt[i] = (1.0 / A[i,i]) * (b[i] - sum);
                    }
                    for (i = 0; i < n; i++) x[i] = xt[i];
              }
              if (Visualiza != 0) {
                    Console.WriteLine(" ");
                    Console.WriteLine("Matriz");
                    for (i = 0; i < n; i++) {
                          for (j = 0; j < n; j++) {
                                System.Console.Write(A[i,j] + " ");
                          }
                          System.Console.WriteLine(" ");
                    }
                    System.Console.WriteLine(" ");
                    System.Console.WriteLine("b");
                    for (i = 0; i < n; i++) {
                          System.Console.Write(b[i] + " ");
                    }
                    System.Console.WriteLine(" ");
                    System.Console.WriteLine("x");
                    for (i = 0; i < n; i++) {
                          System.Console.Write(x[i]+ " ");
                    }
                    System.Console.WriteLine(" ");
              }
```

```
        }
        // Resuelve Ax=b usando el metodo Gauus-Siedel
        public void Gauss_ Siedel(double[,] A, double[] x, double[] b, int n, int
iter) {
            int i, j, m;
            double sum;
            for (m = 0; m < iter; m ++) {
                for (i = 0; i < n; i++) {
                    sum = 0.0;
                    for (j = 0; j < n; j ++) {
                        if ( i == j) continue;
                        sum += A[i,j] * x[j];
                    }
                    if (A[i,i] == 0.0) return;
                    x[i] = (1.0 / A[i,i]) * (b[i] - sum);
                }
            }
            if (Visualiza != 0) {
                Console.WriteLine(" ");
                Console.WriteLine("Matriz");
                for (i = 0; i < n; i++) {
                    for (j = 0; j < n; j++) {
                        System.Console.Write(A[i,j] + " ");
                    }
                    System.Console.WriteLine(" ");
                }
                System.Console.WriteLine(" ");
                System.Console.WriteLine("b");
                for (i = 0; i < n; i++) {
                    System.Console.Write(b[i] + " ");
                }
                System.Console.WriteLine(" ");
                System.Console.WriteLine("x");
                for (i = 0; i < n; i++) {
                    System.Console.Write(x[i] + " ");
                }
                System.Console.WriteLine(" ");
            }
        }
        // Lado derecho de la ecuacion diferencial parcial
        public static double LadoDerecho(double x) {
            double pi = 3.14159265358979323846264338 32;
            return -pi * pi * Math.Cos(pi * x);
        }
        // Funcion Principal ....
        public static void Main(String[] args) {
```

```
        double xi = -1.0; // Inicio del diminio
        double xf = 2.0; // Fin del dominio
        double vi = -1.0; // Valor en la frontera xi
        double vf = 1.0; // Valor en la frontera xf
        int n = 11; // Particion
        int N = n - 2; // Nodos interiores
        double h = (xf - xi) / (n - 1); // Incremento en la malla
        double[,] A = new double[N,N]; // Matriz A
        double[] b = new double[N]; // Vector b
        double[] x = new double[N]; // Vector x
        double R = 1 / (h * h);
        double P = -2 / (h * h);
        double Q = 1 / (h * h);
        // Primer renglon de la matriz A y vector b
        A[0,0] = P;
        A[0,1] = Q;
        b[0] = LadoDerecho(xi) - vi * R;
        // Renglones intermedios de la matriz A y vector b
        for (int i = 1; i < N - 1; i++) {
            A[i,i - 1] = R;
            A[i,i] = P;
            A[i,i + 1] = Q;
            b[i] = LadoDerecho(xi + h * i);
        }
        // Renglon final de la matriz A y vector b
        A[N - 1,N - 2] = R;
        A[N - 1,N - 1] = P;
        b[N - 1] = LadoDerecho(xi + h * N - 2) - vf * Q;
        // Resuleve el sistema lineal Ax=b
        fdm1d ejem = new fdm1d();
        ejem.Gauss_Siedel(A, x, b, N, 1000);
        ejem.Jacobi(A, x, b, N, 1000);
    }
}
}
```

G.7 Implementación en Fortran

Fortran es un lenguaje de programación procedimental e imperativo que está adaptado al cálculo numérico y la computación científica, existen una gran variedad de subrutinas para manipulación de matrices, pero es facil implementar lo necesario para nuestros fines.

Sea

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

Ejemplo 27 *El programa queda implementado en FORTRAN como:*

```
program fdm1d
    implicit none
    integer i, N, nn
    real*8, allocatable :: A(:,:), b(:), x(:)
    real*8 xi, xf, vi, vf, h, R, P, Q, y
    xi = -1.0 ! Inicio del diminio
    xf = 2.0 ! Fin del dominio
    vi = -1.0 ! Valor en la frontera xi
    vf = 1.0 ! Valor en la frontera xf
    nn = 11 ! Particion
    N = nn - 2 ! Nodos interiores
    h = (xf - xi) / (nn-1) ! Incremento en la malla
    print*,"h:",h
    allocate (A(N,N), b(N), x(N))
    R = 1. / (h * h)
    P = -2. / (h * h)
    Q = 1. / (h * h)
    ! Primer renglon de la matriz A y vector b
    A(1,1)=P
    A(2,1)=Q
    call ladoDerecho(xi,y)
    b(1)=y-vi*R
    ! Renglones intermedios de la matriz A y vector b
    do i=2,N-1
        A(i-1,i)=R
        A(i,i)=P
        A(i+1,i)=Q
        call ladoDerecho(xi+h*(i-1),y)
        b(i)= y
    end do
    ! Renglon final de la matriz A y vector b
    A(N-1,N)=R
    A(N,N)=P
    call ladoDerecho(xi+h*N,y)
    b(N)= y -vf*Q
    call gaussSiedel(A, x, b, N, 1000)
     print*, "A: ", A
     print*, "b: ", b
    print*, "x: ", x
end program
subroutine ladoDerecho(x,y)
    real*8, intent(in) :: x
    real*8, intent(inout) :: y
    real*8 pi
    pi = 3.14159265358979323846264338 32;
```

```
        y = -pi * pi * cos(pi * x);
    end subroutine
    subroutine gaussSiedel(a, x, b, nx, iter)
        implicit none
        integer, intent(in) :: nx, iter
        real*8, intent(in) :: a(nx,nx), b(nx)
        real*8, intent(inout) :: x(nx)
        integer i, j, m
        real*8 sum

        do m = 1, iter
            do i = 1, nx
                sum = 0.0
                do j = 1, nx
                    if (i .NE. j) then
                        sum = sum + a(i,j) * x(j)
                    end if
                end do
                x(i) = (1.0 / a(i,i)) * (b(i) - sum)
            end do
        end do
    end subroutine
```

G.8 Implementación en C

C es un lenguaje de programación de tipos de datos estáticos, débilmente tipificado, de medio nivel, existen una gran variedad de subrutinas para manipulación de matrices, pero es facil implementar lo necesario para nuestros fines.

Sea

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

Ejemplo 28 *El programa queda implementado en C como:*

```
#include <math.h>
#include <stdio.h>
#define PARTICION 11 // Tamano de la particion
#define N 9 // Numero de incognitas
#define VISUALIZA 1 // (0) No visualiza la salida, otro valor la visualiza
// Lado derecho de la ecuacion diferencial parcial
double LadoDerecho(double x)
{
    double pi = 3.14159265358979323846264338832;
    return -pi * pi * cos(pi * x);
}
// Resuelve Ax=b usando el metodo Jacobi
void Jacobi(double A[N][N], double x[], double b[], int n, int iter)
```

```
{
    int i, j, m;
    double sum;
    double xt[N];
    for (m = 0; m < iter; m ++)
    {
        for (i = 0; i < n; i++)
        {
            sum = 0.0;
            for (j = 0; j < n; j ++)
            {
                if ( i == j) continue;
                sum += A[i][j] * x[j];
            }
            if (A[i][i] == 0.0) return;
            xt[i] = (1.0 / A[i][i]) * (b[i] - sum);
        }
        for (i = 0; i < n; i++) x[i] = xt[i];
    }
    if (VISUALIZA)
    {
        printf("\nMatriz\n");
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < n; j++)
            {
                printf("%f ", A[i][j]);
            }
            printf("\n");
        }
        printf("\nb\n");
        for (i = 0; i < n; i++)
        {
            printf("%f ", b[i]);
        }
        printf("\nx\n");
        for (i = 0; i < n; i++)
        {
            printf("%f ", x[i]);
        }
        printf("\n");
    }
}
// Resuelve Ax=b usando el metodo Gauus-Siedel
void Gauss_ Siedel(double A[N][N], double x[], double b[], int n, int iter)
{
```

```
        int i, j, m;
        double sum;
        for (m = 0; m < iter; m ++)
        {
            for (i = 0; i < n; i++)
            {
                sum = 0.0;
                for (j = 0; j < n; j ++)
                {
                    if ( i == j) continue;
                    sum += A[i][j] * x[j];
                }
                if (A[i][i] == 0.0) return;
                x[i] = (1.0 / A[i][i]) * (b[i] - sum);
            }
        }
        if (VISUALIZA)
        {
            printf("\nMatriz\n");
            for (i = 0; i < n; i++)
            {
                for (j = 0; j < n; j++)
                {
                    printf("%f ", A[i][j]);
                }
                printf("\n");
            }
            printf("\nb\n");
            for (i = 0; i < n; i++)
            {
                printf("%f ", b[i]);
            }
            printf("\nx\n");
            for (i = 0; i < n; i++)
            {
                printf("%f ", x[i]);
            }
            printf("\n");
        }
    }
    int main()
    {
        double xi = -1.0; // Inicio del diminio
        double xf = 2.0; // Fin del dominio
        double vi = -1.0; // Valor en la frontera xi
        double vf = 1.0; // Valor en la frontera xf
```

```
    int n = PARTICION; // Particion
    double h = (xf - xi) / (n - 1); // Incremento en la malla
    int i;
    double A[N][N]; // Matriz A
    double b[N]; // Vector b
    double x[N]; // Vector x
    double R = 1 / (h * h);
    double P = -2 / (h * h);
    double Q = 1 / (h * h);
    // Primer renglon de la matriz A y vector b
    A[0][0] = P;
    A[0][1] = Q;
    b[0] = LadoDerecho(xi) - vi * R;
    // Renglones intermedios de la matriz A y vector b
    for (i = 1; i < N - 1; i++)
    {
        A[i][i - 1] = R;
        A[i][i] = P;
        A[i][i + 1] = Q;
        b[i] = LadoDerecho(xi + h * i);
    }
    // Renglon final de la matriz A y vector b
    A[N - 1][N - 2] = R;
    A[N - 1][N - 1] = P;
    b[N - 1] = LadoDerecho(xi + h * N - 2) - vf * Q;
    // Resuleve el sistema lineal Ax=b
    Gauss_Siedel(A, x, b, N, 1000);
    Jacobi(A, x, b, N, 1000);
    return 0;
}
```

H Bibliografía

Referencias

[1] K. Hutter y K Jöhnk, *Continuum Methods of Physical Modeling*, Springer-Verlag Berlin Heidelberg New York, 2004.

[2] J. L. Lions y E. Magenes, *Non-Homogeneous Bounduary Value Problems and Applicarions* Vol. I, Springer-Verlag Berlin Heidelberg New York, 1972.

[3] A. Quarteroni y A. Valli, *Domain Decomposition Methods for Partial Differential Equations*. Clarendon Press Oxford, 1999.

[4] A. Quarteroni y A. Valli; *Numerical Approximation of Partial Diferential Equations*. Springer, 1994.

[5] B. Dietrich, *Finite Elements: Theory, Fast Solvers*, and Applications in Solid Mechanics, Cambridge University, 2001.

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

[7] Fuzhen Zhang, *The Schur Complement and its Aplications*, Springer, Numerical Methods and Algorithms, Vol. 4, 2005.

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

[9] L. F. Pavarino, A. Toselli; *Recent Developments in Domain Decomposition Methods*. Springer, 2003.

[10] M.B. Allen III, I. Herrera & G. F. Pinder; *Numerical Modeling in Science And Engineering.* John Wiley & Sons, Inc . 1988.

[11] R. L. Burden y J. D. Faires; *Análisis Numérico*. Math Learning, 7 ed. 2004.

[12] S. Friedberg, A. Insel, and L. Spence; *Linear Algebra*, 4th Edition, Prentice Hall, Inc. 2003.

[13] Y. Saad; *Iterative Methods for Sparse Linear Systems*. SIAM, 2 ed. 2000.

[14] Y. Skiba; *Métodos y Esquemas Numéricos, un Análisis Computacional*. UNAM, 2005.

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

[16] I. Foster; *Designing and Building Parallel Programs*. Addison-Wesley Inc., Argonne National Laboratory, and the NSF, 2004.

[17] Jorge L. Ortega-Arjona, *Patterns for Parallel Software Design*, Wiley series in Software Design Patterns, 2010.

[18] DDM Organization, *Proceedings of International Conferences on Domain Decomposition Methods*, 1988-2012.

http://www.ddm.org and http://www.domain-decomposition.com

[19] Toselli, A., and Widlund O. *Domain decomposition methods- Algorithms and theory*, Springer Series in Computational Mathematics, Springer-Verlag, Berlin, 2005, 450p.

[20] 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.

[21] Mandel J. & Tezaur R. *Convergence of a Substructuring Method with Lagrange Multipliers*, Numer. Math. 73 (1996) 473-487.

[22] 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.

[23] Farhat C., Lesoinne M., Pierson K. *A Scalable Dual-Primal Domain Decomposition Method*, Numer. Linear Algebra Appl. 7 (2000) 687-714.

[24] Mandel J. & Tezaur R. *On the Convergence of a Dual-Primal Substructuring Method*, Numer. Math. 88(2001), pp. 5443-558.

[25] Mandel, J. *Balancing Domain Decomposition*. Comm. Numer. Meth. Engrg., 9:233-241, 1993.

[26] Mandel J., & Brezina M., *Balancing Domain Decomposition for Problems with Large Jumps in Coefficients,* Math. Comput. 65 (1996) 1387-1401.

[27] Dohrmann C., *A Preconditioner for Substructuring Based on Constrained Energy Minimization*. SIAM J. Sci. Comput. 25 (2003) 246-258.

[28] Mandel J. & Dohrmann C., *Convergence of a Balancing Domain Decomposition by Constraints and Energy Minimization*. Numer. Linear Algebra Appl. 10 (2003) 639-659.

[29] 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.

[30] J. Li and O. Widlund, *FETI-DP, BDDC and block Cholesky Methods*, Int. J. Numer. Methods Engrg. 66, 250-271, 2005.

[31] 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.

[32] 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.

[33] J. Mandel, C. R. Dohrmann, and R. Tezaur, *An Algebraic Theory for Primal and Dual Substructuring Methods by Constraints*, Appl. Numer. Math., 54 (2005), pp. 167-193.

[34] A. Klawonn, O. B. Widlund, and M. Dryja, *Dual-primal FETI Methods for Three-Dimensional Elliptic Problems with Heterogeneous Coefficients*, SIAM J. Numer. Anal., 40 (2002), pp. 159-179.

[35] Agustín Alberto Rosas Medina, *El Número de Péclet y su Significación en la Modelación de Transporte Difusivo de Contaminantes,* Tesis para obtener el titulo de Matemático, UNAM, 2005.

[36] Omar Jonathan Mendoza Bernal, Resolución de Ecuaciones Diferenciales Parciales Mediante el Método de Diferencias Finitas y su Paralelización, Tesis para obtener el titulo de Matemático, UNAM, 2016.

[37] 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.

[38] Tezaur R., *Analysis of Lagrange Multipliers Based Domain Decomposition*. P.H. D. Thesis, University of Colorado, Denver, 1998.

[39] 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.

[40] Valeri I. Agoshkov, *Poincaré-Steklov Operators and Domain Decomposition Methods in Finite Dimensional Spaces*. First International Symposium on Domain Decomposition Methods for Partial Differential Equations, pages 73-112, Philadelphia, PA, 1988. SIAM. Paris, France, Junuary 7-9, 1987.

[41] Toselli, A., *FETI Domain Decomposition Methods for Escalar Advection-Diffusion Problems*. Compututational Methods Appl. Mech. Engrg. 190. (2001), 5759-5776.

[42] C.T. Keller, *Iterative Methods for Linear and Nonlinear Equations*, Societe for Industrial and Applied Mathematics, 1995.

[43] Manoj Bhardwaj, David Day, Charbel Farhat, Michel Lesoinne, Kendall Pierson, and Daniel Rixen. *Application of the PETI Method to ASCI Problems: Scalability Results on One Thousand Processors and Discussion of Highly Heterogeneous Problems*. Intemat. J. Numer. Methods Engrg., 47:513-535, 2000.

[44] Zdengk Dostdl and David Hordk. *Scalability and FETI Based Algorithm for Large Discretized Variational Inequalities*. Math. Comput. Simulation, 61(3-6): 347-357, 2003. MODELLING 2001 (Pilsen).

[45] Charbel Farhat, Michel Lesoinne, and Kendall Pierson. *A Scalable Dual-Primal Domain Decomposition Method*. Numer. Linear Algebra Appl., 7(7-8):687-714, 2000.

[46] Yannis Fragakis and Manolis Papadrakakis, *The Mosaic of High Performance Domain Decomposition Methods for Structural Mechanics: Formulation, Interrelation and Numerical Efficiency of Primal and Dual Methods*. Comput. Methods Appl. Mech. Engrg, 192(35-36):3799-3830, 2003.

[47] Kendall H. Pierson, *A family of Domain Decomposition Methods for the Massively Parallel Solution of Computational Mechanics Problems*. PhD thesis, University of Colorado at Boulder, Aerospace Engineering, 2000.

[48] Manoj Bhardwaj, Kendall H. Pierson, Garth Reese, Tim Walsh, David Day, Ken Alvin, James Peery, Charbel Farhat, and Michel Lesoinne. Salinas, *A Scalable Software for High Performance Structural and Mechanics Simulation*. In ACM/IEEE Proceedings of SC02: High Performance Networking and Computing. Gordon Bell Award, pages 1-19, 2002.

[49] Klawonn, A.; Rheinbach, O., *Highly Scalable Parallel Domain Decomposition Methods with an Application to Biomechanics*, Journal of Applied Mathematics and Mechanics 90 (1): 5-32, doi:10.1002/zamm.200900329.

[50] Petter E. BjØrstad and Morten Skogen. *Domain Decomposition Algorithms of Schwarz Type, Designed for Massively Parallel Computers*. In David E. Keyes, Tony F. Chan, Gerard A. Meurant, Jeffrey S. Scroggs, and Robert G. Voigt, editors. Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations, pages 362-375, Philadelphia, PA, 1992. SIAM. Norfolk, Virgina, May 6-8, 1991.

[51] Yau Shu Wong and Guangrui Li. *Exact Finite Difference Schemes for Solving Helmholtz Equation at any Wavenumber*. International Journal of Numerical Analysis and Modeling, Series B, Volume 2, Number 1, Pages 91-108, 2011.

[52] Zhangxin Chen, Guanren Huan and Yuanle Ma. *Computational Methods for Multiphase Flow in Porous Media*, SIAM, 2006.

[53] Jos Stam. SIGGRAPH 99, Proceedings of the 26th annual conference on Computer Graphics and Interactive Techniques, Pages 121-128, ACM, 1999.

[54] 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.

[55] 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.

[56] 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.

[57] 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)

[58] 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).

[59] 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.

[60] Herrera, I. and Pinder, G. F., *Mathematical Modelling in Science and Engineering: An Axiomatic Approach*, Wiley, 243p., 2012.

[61] Ismael Herrera and Alberto A. Rosas-Medina, *The Derived-Vector Space Framerwork and Four General purposes massively parallel DDM Algorithms*, Engineering Analysis with Boundary Elements, 20013, in press.

[62] Antonio Carrillo-Ledesma, Herrera, I, Luis M. de la Cruz, *Parallel Algorithms for Computational Models of Geophysical Systems*, Geofísica Internacional, en prensa, 2013.

[63] Iván Germán Contreras Trejo, *Métodos de Precondicionamiento para Sistemas de Ecuaciones Diferenciales Parciales*, Trabajo de Tesis Doctoral en Proceso, Postgrado en Ciencias e Ingeniería de la Computación, UNAM, 2012. 192(35-36):3799-3830, 2003.

[64] Holger Brunst, Bernd Mohr. *Performance Analysis of Large-Scale OpenMP and Hybrid MPI/OpenMP Applications with Vampir NG*. IWOMP 2005: 5-14.

[65] S.J. Pennycook, S.D. Hammond, S.A. Jarvis and G.R. Mudalige, *Performance Analysis of a Hybrid MPI/CUDA Implementation of the NAS-LU Benchmark*. ACM SIGMETRICS Perform. Eval. Rev. 38 (4). ISSN 0163-5999, (2011).

[66] Doxygen, *Gererate Documentation from Source Code.*
http://www.stack.nl/∼dimitri/doxygen/

[67] XMPI, *A Run/Debug GUI for MPI.*
http://www.lam-mpi.org/software/xmpi/

[68] VAMPIR, *Performance Optimization.*
http://www.vampir.eu/