Índice

1 Implementación Computacional de DVS

La implementación computacional de los métodos de descomposición de dominio sin traslape en general (véase [8]) y de los métodos de descomposición de dominio en el espacio de vectores derivados (DVS) en particular, 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 es una forma óptima[1] de dividir la de la carga computacional requerida para solucionar un problema de descomposición de dominio sin traslapes, en el cual uno o más subdominio 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.

1.1 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 $\underline{\underline{Mu}}_{\Delta} = \underline{b}$ esquematizados por los algoritmos descritos en la Ec.(**??** ó **??**), 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.

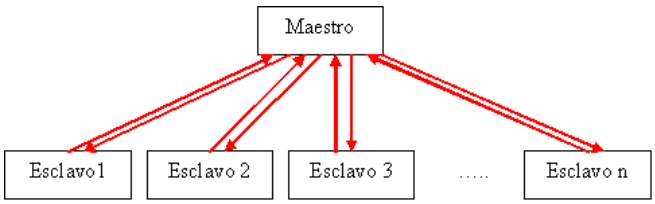

Figura 1: Esquema del Maestro-Esclavo

[1] 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.

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 [16], [14] y [15]).

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.

1.2 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[2] 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.

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

[2] 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.

una jerarquía de clases que implementa mediante herencia a la clase abstracta, el cual usan los algoritmos que requerían solucionar un sistema lineal, esta clase abstracta se llama Solvable —véase apéndice **??**—. La jerarquía[3] de clases mostrada en la figura (2) 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.

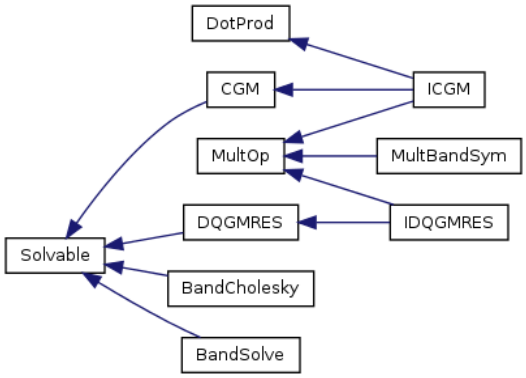

Figura 2: Jerarquía de clases para la implementación de los métodos de resolución de sistemas lineales.

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.

1.2.1 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 ocho métodos de descomposición de dominio desarrollados y dependiendo del la ecuación diferencial parcial a resolver, se usaría el método iterativo —Gradiente Conjugado o el método Residual Mínimo Generalizado o cualquier otro— dependiendo de que la matriz

[3]Las jerarquías de clases de herencia mostradas en las figuras fueron generadas usando Doxygen documentation (véase [58]) a partir del código fuente en C++.

global virtual fueran simétrica o no simétrica; su jerarquía de clases se muestra en la figura (3).

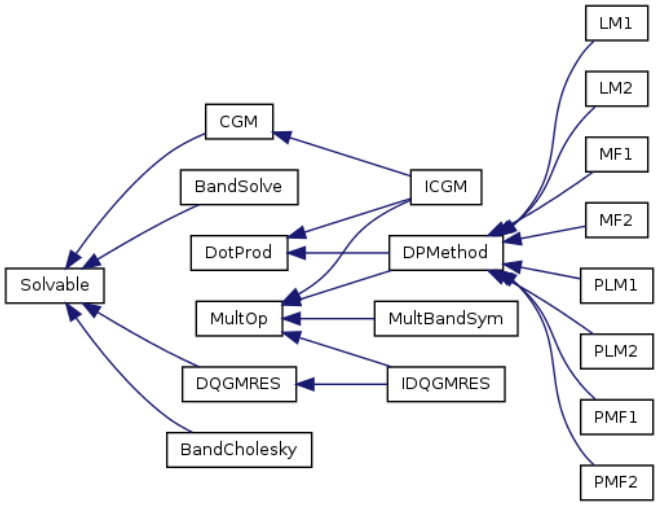

Figura 3: Jerarquía de clases para la implementación secuencial

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 necesario 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 del sistema lineal global virtual asociado al método de descomposición de dominio.

1.2.2 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.

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:

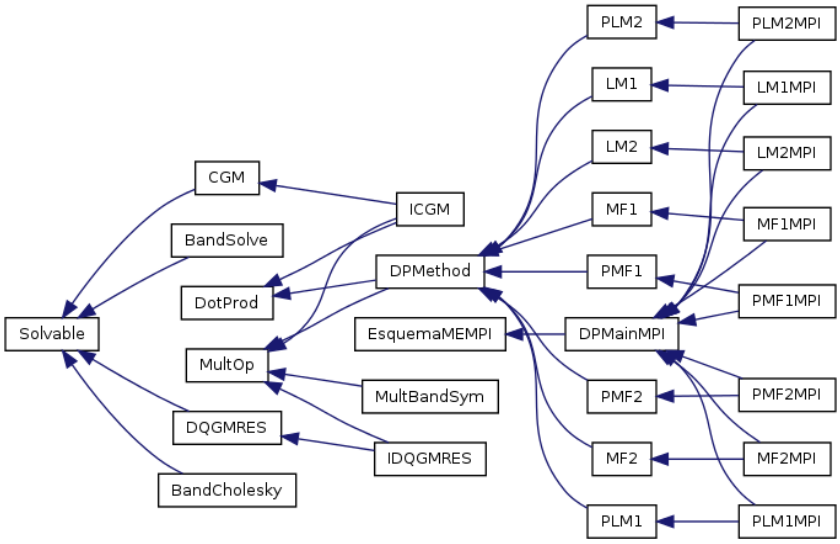

Figura 4: Jerarquía de clases para la implementación paralela rehusando toda la jerarquía de la implementación secuencial y de resolución de sistemas lineales

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 [16]).

1.3 Alcances y Limitaciones del Esquema Maestro-Esclavo

El esquema Maestro-Esclavo es eficiente cuando se tiene una carga 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
  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.

1.4 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:

- 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 al 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.

- 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

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

  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.

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, este cuello de botella 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 en comunicaciones que en el cálculo.

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 una 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 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.

Notemos que el esquema Maestro-Esclavo —secuencial o paralelo— lanza P objetos o 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 procesadore 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.

2 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] B. I. Wohlmuth; *Discretization Methods and Iterative Solvers Based on Domain Decomposition*. Springer, 2003.

[8] L. F. Pavarino, A. Toselli; *Recent Developments in Domain Decomposition Methods*. Springer, 2003.

[9] M.B. Allen III, I. Herrera & G. F. Pinder; *Numerical Modeling in Science And Engineering.* John Wiley & Sons, Inc . 1988.

[10] R. L. Burden y J. D. Faires; *Análisis Numérico*. Math Learning, 7 ed. 2004.

[11] S. Friedberg, A. Insel, and L. Spence; *Linear Algebra*, 4th Edition, Prentice Hall, Inc. 2003.

[12] Y. Saad; *Iterative Methods for Sparse Linear Systems*. SIAM, 2 ed. 2000.

[13] Y. Skiba; *Métodos y Esquemas Numéricos, un Análisis Computacional*. UNAM, 2005.

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

[15] I. Foster; *Designing and Building Parallel Programs*. Addison-Wesley Inc., Argonne National Laboratory, and the NSF, 2004.

[16] Jorge L. Ortega-Arjona, *Patterns for Parallel Software Design*, Wiley series in Software Design Patterns, 2010.

[17] DDM Organization, *Proceedings of International Conferences on Domain Decomposition Methods*, 1988-2012.

http://www.ddm.org and http://www.domain-decomposition.com

[18] Toselli, A., and Widlund O. *Domain decomposition methods- Algorithms and theory*, Springer Series in Computational Mathematics, Springer-Verlag, Berlin, 2005, 450p.

[19] Farhat, C. and Roux, F. X. *A Method of Finite Element Tearing and Interconnecting and its Parallel Solution Algorithm.* Int. J. Numer. Meth. Engng., 32:1205-1227, 1991.

[20] Mandel J. & Tezaur R. *Convergence of a Substructuring Method with Lagrange Multipliers*, Numer. Math. 73 (1996) 473-487.

[21] Farhat C., Lesoinne M. Le Tallec P., Pierson K. & Rixen D. *FETI-DP a Dual-Primal Unified FETI method*, *Part 1: A Faster Alternative to the two-level FETI Method*, Int. J. Numer. Methods Engrg. 50 (2001) 1523-1544.

[22] Farhat C., Lesoinne M., Pierson K. *A Scalable Dual-Primal Domain Decomposition Method*, Numer. Linear Algebra Appl. 7 (2000) 687-714.

[23] Mandel J. & Tezaur R. *On the Convergence of a Dual-Primal Substructuring Method*, Numer. Math. 88(2001), pp. 5443-558.

[24] Mandel, J. *Balancing Domain Decomposition.* Comm. Numer. Meth. Engrg., 9:233-241, 1993.

[25] Mandel J., & Brezina M., *Balancing Domain Decomposition for Problems with Large Jumps in Coefficients,* Math. Comput. 65 (1996) 1387-1401.

[26] Dohrmann C., *A Preconditioner for Substructuring Based on Constrained Energy Minimization.* SIAM J. Sci. Comput. 25 (2003) 246-258.

[27] Mandel J. & Dohrmann C., *Convergence of a Balancing Domain Decomposition by Constraints and Energy Minimization.* Numer. Linear Algebra Appl. 10 (2003) 639-659.

[28] Da Conceição, D. T. Jr., *Balancing Domain Decomposition Preconditioners for Non-symetric Problems*, Instituto Nacional de Matemática pura e Aplicada, Agencia Nacional do Petróleo PRH-32, Rio de Janeiro, May. 9, 2006.

[29] Herrera, I., *Theory of Differential Equations in Discontinuous Piecewise-Defined-Functions*, NUMER METH PART D.E., 23(3): 597-639, 2007 OI 10.1002/num.20182.

[30] Herrera, I. *New Formulation of Iterative Substructuring Methods Without Lagrange Multipliers: Neumann-Neumann and FETI*, NUMER METH PART D.E. 24(3) pp 845-878, May 2008 DOI 10.1002/num.20293.

[31] Herrera I. and R. Yates, *Unified Multipliers-Free Theory of Dual Primal Domain Decomposition Methods*. NUMER. METH. PART D. E. 25(3): 552-581, May 2009, (Published on line May 13, 2008) DOI 10.1002/num.20359.

[32] Herrera, I. & Yates R. A., *The Multipliers-Free Domain Decomposition Methods*, NUMER. METH. PART D. E., 26(4): pp 874-905, July 2010. (Published on line: 23 April 2009, DOI 10.1002/num.20462)

[33] Herrera, I., Yates R. A., *The multipliers-Free Dual Primal Domain Decomposition Methods for Nonsymmetric Matrices*. NUMER. METH. PART D. E. DOI 10.1002/num.20581 (Published on line April 28, 2010).

[34] 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.

[35] Antonio Carrillo-Ledesma, Herrera, I, Luis M. de la Cruz, Parallel Algorithms for Computational Models of Geophysical Systems, Geofísica Internacional, Vol, XXX.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] Alberto Rosas Medina, *Métodos de Estabilización para Problemas de Advección-Difusión*, Trabajo de Investigación para Sustentar el Examen de Candidaura al Doctorado, Postgrado en Ciencias de la Tierra, UNAM, 2011.

[41] 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.

[42] 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.

[43] Tezaur R., *Analysis of Lagrange Multipliers Based Domain Decomposition*. P.H. D. Thesis, University of Colorado, Denver, 1998.

[44] 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.

[45] 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.

[46] Toselli, A., *FETI Domain Decomposition Methods for Escalar Advection-Diffusion Problems*. Compututational Methods Appl. Mech. Engrg. 190. (2001), 5759-5776.

[47] C.T. Keller, *Iterative Methods for Linear and Nonlinear Equations*, Societe for Industrial and Applied Mathematics, 1995.

[48] 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.

[49] 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).

[50] Charbel Farhat, Michel Lesoinne, and Kendall Pierson. *A Scalable Dual-Primal Domain Decomposition Method*. Numer. Linear Algebra Appl., 7(7-8):687-714, 2000.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] Petter E. Bj∅rstad 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.

[56] Holger Brunst, Bernd Mohr, *Performance Analysis of Large-Scale OpenMP and Hybrid MPI/OpenMP Applications with Vampir NG*. IWOMP 2005: 5-14.

[57] 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).

[58] Doxygen, *Gererate Documentation from Source Code*.
http://www.stack.nl/∼dimitri/doxygen/

[59] XMPI A Run/Debug GUI for MPI.
http://www.lam-mpi.org/software/xmpi/

[60] VAMPIR Performance Optimization.
http://www.vampir.eu/