Paralelización de métodos iterativos

Germán Larrazábal
Departamento de Computación
Facultad de Ciencias y Tecnología (FACYT), Universidad de Carabobo
Avenida Montes de Oca, No. 120-267, Edificio FACYT, Valencia, Estado Carabobo.
Teléfono/Fax: 0241-8678243
e-mail: glarraza@uc.edu.ve

Resumen

En este trabajo se realiza una paralelización de métodos iterativos. Dicha paralelización se basa en explotar el paralelismo grano grueso generado por una descomposición en dominios dentro del modelo paso de mensaje. Se estudia la influencia del secuenciamiento de las comunicaciones en el rendimiento global de la aplicación. Se presenta una aceleración cuasi-lineal cuando se trabaja con el sistema sin precondicionamiento y se obtiene una mejora de éste cuando se precondiciona el sistema.

Palabras clave: Sistemas lineales, métodos iterativos, paralelismo.

Parallelization of iterative methods

Abstract

In this work, we present a parallelization of iterative methods. This parallelism is based in exploiting the coarse grain parallelism generated by a domain decomposition under message passing model. We have studied the influence of communication scheduling on the global performance of the application. The results show a cuasi-linear speed-up when there is not preconditioner and we have obtained a significant speed-up when there is a preconditioner in the system.

Keywords: Linear systems, iterative methods, parallelism.

1. INTRODUCCIÓN

La paralelización de un método iterativo puede hacerse a diferentes niveles. Tradicionalmente, el paralelismo que más se ha explotado es el paralelismo grano fino a nivel de instrucciones en lenguaje de máquina o a nivel de bucles en un lenguaje de alto nivel [1], [2]. Normalmente, esta paralelización se realiza de forma automática por el compilador y es muy dependiente de la arquitectura concreta que se considere. Este tipo de paralelización ha dado muy buenos resultados en operaciones algebraicas sobre matrices densas. Sin embargo, la manipulación de matrices dispersas presenta una serie de inconvenientes adicionales que disminuyen considerablemente la eficiencia de este tipo de paralelización [3]. Estos inconvenientes son principalmente tres: los algoritmos están limitados más por el acceso a memoria que por el número de operaciones, los patrones de acceso a memoria no se conocen en tiempo de compilación, y por último la carga de trabajo por iteración en los bucles suele ser desconocida en tiempo de ejecución. Aunque existen muchas posibles distribuciones de datos para realizar una paralelización de grano grueso para los métodos iterativos [4], la descomposición en dominios se acomoda a la paralelización de aplicaciones completas, no únicamente a la resolución de sistemas lineales, y ha sido la que en los últimos años se ha venido investigando más intensamente [5-7]. Así pues, el modelo de distribución de datos en el que se centra el trabajo es la descomposición en dominios. El modelo de programación de paso de mensaje [8] es posiblemente uno de los más usados para explotar paralelismo de grano grueso. Una de sus principales ventajas es que

los programas desarrollados con este modelo son ejecutables tanto en arquitecturas de memoria compartida como de memoria distribuida. Uno de los inconvenientes atribuidos a este modelo de programación es que el desarrollo de programas es más complejo, al tener el programador que poner de forma explícita las comunicaciones entre procesos. Sin embargo, como se verá en este trabajo, en el área especifica de resolución de sistemas lineales mediante descomposición en dominios esto no es cierto, ya que se pueden desarrollar los programas con un modelo SPMD (Single Program Multiple Data) que da como resultado que todas las comunicaciones entre procesos se reduzcan a únicamente tres patrones.

2. DISTRIBUCIÓN DE DATOS EN UNA DESCOMPOSICIÓN EN DOMINIOS

La descomposición en dominios se realiza mediante algún algoritmo de partición de grafos [9-13]. En las aplicaciones de resolución de EDPs, la partición se aplica a la malla que discretiza el dominio de resolución. De forma abstracta, se puede imaginar que la partición se aplica al grafo de conectividad de la matriz del sistema lineal. El problema de generar una descomposición en dominios óptima es en general NP-completo [14]. El calificativo óptimo hace referencia al hecho de que la partición debe dejar la carga computacional por dominio perfectamente equilibrada y debe hacer que el costo de las comunicaciones sea el menor posible. Es decir, los dominios deben tener el número mínimo de vecinos (minimizar el número de mensajes) y el número de nodos en la frontera con cada vecino debe ser mínimo (minimizar la longitud de los mensajes). Sin embargo, existen heurísticas que logran soluciones considerablemente buenas. En cualquier caso, la partición del grafo se realiza de forma que se obtiene un conjunto de subgrafos, llamados dominios, cuya característica es que no tienen ninguna conexión directa entre éstos. Los dominios tienen únicamente conexiones con otro subgrafo llamado frontera.

La Figura 1, muestra un grafo partido en cinco dominios. A cada dominio se le asociará un proceso paralelo que almacenará los datos asociados al dominio.

La frontera se divide a su vez de alguna forma entre los procesos paralelos.

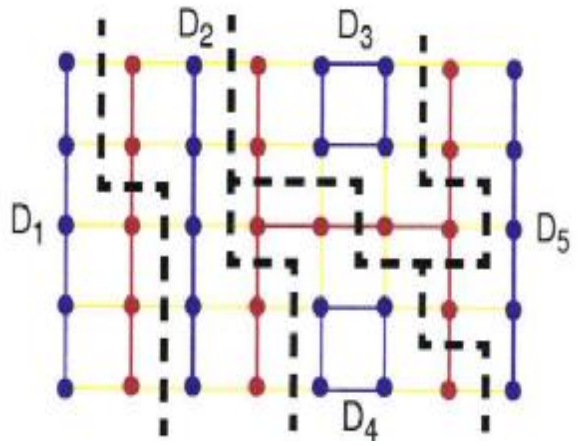

Figura 1. Partición de grafo.

Para hacer esta división se aprovecha el hecho de que un dominio no está conectado con toda la frontera, sino solamente con un subconjunto de la frontera. Se denomina frontera entre el dominio i-ésimo y el dominio j-ésimo al conjunto intersección de la frontera del dominio i-ésimo con la frontera del dominio j-ésimo. Entonces para dividir la frontera entre dominios, simplemente se dividen las fronteras entre parejas de dominios, asignando la mitad de dicha frontera a cada uno de los dos dominios vecinos. La Figura 1 muestra la partición de la frontera. Dado que cada nodo del grafo que se ha partido está asociado a una o varias incógnitas, numerando primeramente las incógnitas de los dominios y después las de la frontera, se obtiene un sistema lineal, donde la matriz tiene una estructura a bloques 2x2 como la siguiente:

$$\begin{pmatrix} A_{II} & A_{IB} \\ A_{BI} & A_{BB} \end{pmatrix}$$

Donde A_{II} es una matriz diagonal a bloques, y cada A_{ii} representa las conexiones entre nodos internos del dominio k-ésimo; las matrices A_{IB} y A_{BI} representan las conexiones entre nodos internos y nodos frontera, en ambos sentidos; y la matriz A_{BB} representa las conexiones entre nodos frontera. La Figura 2, muestra la estructura a bloques de la matriz.

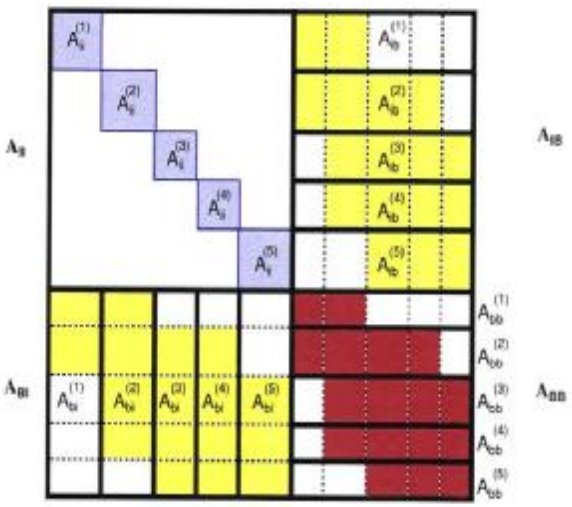

Figura 2. Estructura a bloques de la matriz A.

La distribución de datos de la matriz A que induce una partición en dominios es evidente en lo que respecta a las matrices A_{ii}, A_{ib} y A_{bi}, siendo el proceso paralelo k-ésimo el que almacena dichas matrices. Sin embargo, con la matriz A_{BB} hay diferentes posibilidades. Dado que el algoritmo de partición habrá asignado cierta parte de la frontera al proceso k-ésimo, numerando las incógnitas de esa parte de la frontera consecutivamente, se podría asignar al proceso k-ésimo el grupo de filas o columnas asociadas con dichas incógnitas en la matriz A_{BB}. En nuestro caso, se ha optado por distribuir por grupos de filas, ya que todos los formatos de almacenamiento que se usan son orientados a filas. Se denotará como A_{bb} al grupo de filas de la matriz A_{BB} almacenadas por el proceso k-ésimo. Así pues, se ve que la matriz A se almacena sin ninguna replicación de datos entre procesos. La distribución de datos para los vectores se puede ver en la Figura 3.

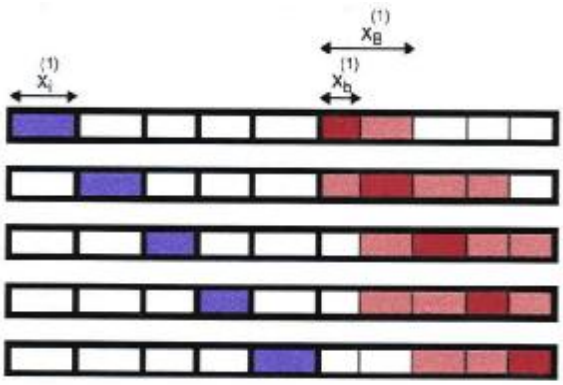

Figura 3. Estructura de los vectores.

El proceso k-ésimo almacena las incógnitas asociadas a los nodos internos del dominio k-ésimo. Se denomina como X_i al vector que almacena dichas incógnitas. El proceso k-ésimo también almacena las incógnitas asociadas a su parte de frontera, se denomina X_b al vector que almacena dichas incógnitas de la frontera. Sin embargo, habitualmente los nodos del dominio k-ésimo no tienen conexiones únicamente con la parte de la frontera asignada al proceso k-ésimo, es decir, las matrices A_{ib} y A_{bi} no son diagonales a bloques. Por ello, el proceso k-ésimo necesita tener replicados aquellos vectores X_b necesarios para realizar la operación matriz por vector. Se denomina X_B al vector formado por todos los X_b replicados en el proceso k-ésimo.

3. PARALELIZACIÓN DE OPERACIONES BÁSICAS

Una vez realizada una descomposición en dominios y obtenida una distribución de datos como la descrita en la sección anterior, se puede aplicar un método iterativo directamente al sistema lineal. Es decir, simplemente se usa la descomposición en dominios como una distribución de datos para la paralelización. Las operaciones básicas de un método iterativo son: operaciones tipo AXPY, productos escalares, producto matriz por vector y resolución de sistemas triangulares. Las tres primeras operaciones, son comunes en todos los métodos iterativos. La última, resolución de sistemas triangulares, aparece como consecuencia de uso de precondicionadores (factorizaciones incompletas). Sin embargo, si se usan precondicionadores polinomiales o los SPAI esta operación no se ejecuta. Algunos métodos, como por ejemplo el GMRES(m), tienen alguna operación adicional. Estas operaciones adicionales suelen estar asociadas al cálculo del algún parámetro del algoritmo mediante un pequeño ajuste de mínimos cuadrados. Seguidamente, se analizarán las tres operaciones básicas de los métodos iterativos.

3.1. Operaciones tipo AXPY: y = y + ax

Con la distribución de datos propuesta, estas operaciones se ejecutan totalmente en paralelo si el escalar "a" está replicado entre todos los procesos.

3.2. Productos escalares: a = <x, y>

Cada proceso debe calcular el producto escalar

parcial de sus componentes internas y de sus componentes propias de la frontera, es decir, la suma $<X_i,Y_i> + <X_b,Y_b>$. Luego, se debe hacer una fase de comunicación global que acumule todos los productos parciales en un único proceso. Finalmente, dicho proceso debe replicar entre todos los procesos el valor total del producto escalar. El algoritmo óptimo de comunicación para la reducción y la replicación global es bien conocido, es un algoritmo en árbol binario. El número de pasos hasta completar todo el proceso de comunicación es log2P, donde P es el número de procesos paralelos. Lógicamente, aunque es posible programar dichas comunicaciones mediante un algoritmo en árbol binario, la posibilidad real de hacer o no simultáneamente varias comunicaciones la dará la arquitectura específica del computador. Estas comunicaciones serán un cuello de botella si P es grande. Sin embargo, en nuestro caso, el interés es una paralelización de grano grueso y se considera que el número de procesos no será elevado (P < 100). Por lo tanto, estas comunicaciones no supondrán un cuello de botella real si se comparan con el caso del sistema triangular.

3.3. Producto matriz por vector: y = Ax

El algoritmo de multiplicación matriz por vector tiene comunicaciones entre parejas de dominios vecinos de forma que muchas de ellas pueden hacerse simultáneamente. El secuenciamiento de estas comunicaciones puede hacerse en el pre-procesado de la aplicación de forma que quede un secuenciamiento con el mínimo número de pasos y libre de bloqueos. Lógicamente, la bondad de este secuenciamiento depende del grafo de interconexión entre dominios. Siempre es posible generar una descomposición en dominios tan mala que todos los dominios sean vecinos de todos, pero se parte de la base que una descomposición en dominios sólo es aceptable si el número de vecinos de un determinado dominio es relativamente pequeño. En este trabajo, se han usado mallas reales descompuestas por un paquete de partición de grafos llamado METIS.

3.4. Sistemas triangulares: Ly = ® Uz = y

Los sistemas triangulares deben ser resueltos cuando se aplica el precondicionador en cada iteración del método. El patrón básico de comunicaciones de un algoritmo de sustitución hacia delante y hacia atrás es: Recibir de vecinos anteriores, calcular y enviar a vecinos posteriores; donde vecinos anteriores hacen referencia a aquellos vecinos de cuyos datos depende el cálculo del proceso en cuestión, y vecinos posteriores hace referencia a aquellos vecinos cuyos cálculos dependen de los datos generados por el proceso en cuestión. A este patrón de comunicaciones se le denomina comunicaciones secuencializadas. Estos algoritmos son un conocido cuello de botella secuencial debido a las dependencias de datos [15], [16]. Para obtener la máxima cantidad de paralelismo se debe realizar un coloreado de los dominios, de forma que dominios agrupados en el mismo color no tengan dependencias de datos entre estos, y en consecuencia puedan hacer la fase de cálculo en paralelo. El efecto cuello de botella que ejercen los sistemas triangulares se muestra claramente en las Figuras 4 y 5. La matriz usada para obtener ambas figuras tiene aproximadamente unas 50.000 filas. Se usó METIS para partir el grafo. En la Figura 6, se puede observar el peso porcentual de cada una de las operaciones básicas en una iteración del método iterativo BiCGstab conforme se aumenta el número de procesos. Se observa que el tiempo está claramente dominado por la resolución de los sistemas triangulares para un número de procesos relativamente grande.

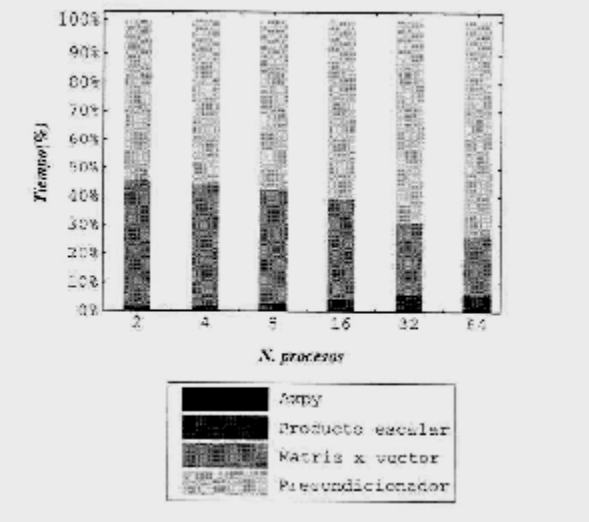

Figura 6. Peso en tiempo de cada una de las operaciones.

También, se puede observar como, aunque el producto escalar presenta comunicaciones que pueden ser un cuello de botella, para un número moderado de procesos su influencia no es significativa. En la Figura 7, se puede ver la aceleración (speed-up) de BiCGstab para el mismo problema con y sin precondicionador, podría ser rentable usar el método iterativo sin precondicionador. Desafortunadamente, en muchas

aplicaciones si no se usa un precondicionador el método iterativo no converge.

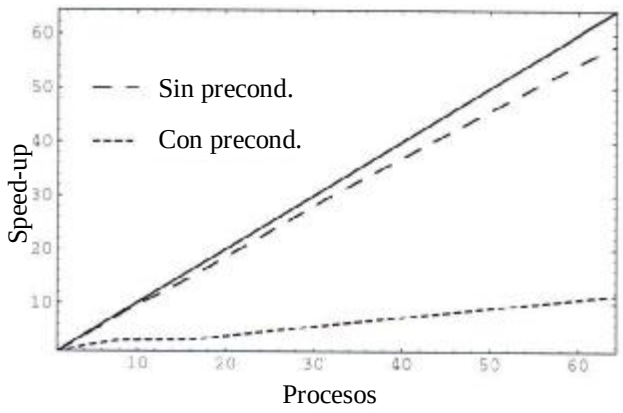

Figura 7. Aceleración obtenida con el BiCGstab.

4. CONCLUSIONES

Se ha presentado un esquema de paralelización de métodos iterativos para matrices provenientes de la discretización de EDPs. El desarrollo de programas paralelos para la resolución de sistemas lineales dispersos, usando el modelo de programación de paso de mensaje y la distribución de datos inducida por una descomposición en dominios requiere únicamente tres patrones de comunicación que hemos denominado: comunicaciones globales, intercambio entre vecinos y comunicaciones secuencializadas.

5. REFERENCIAS

[1] Dongarra, J.; Duff, I.; Sorensen, D.; Van der Vost, H. "Solving linear system on vector and shared memory computers, SIAM, ISBN 0-89871-270-X", 1991.

[2] Ortega, J. "Introduction to parallel and vector solution of linear systems, Plenum Press", ISBN 0-306-42862-8, 1988.

[3] Ortega, J.; Voig, R. "Solution of partial difierential equations on vector and parallel computers", SIAM, ISBN 0-89871-055-3, 1985.

[4] Romero, L.; Zapata, E. "Data distributions for sparse matrix vector multiplication", Tech. Report, Dept. Arquitectura de computadores, Universidad de Malaga, España, 1993.

[5] Brochard, L. "Communication and control cost of domain decomposition on loosely coupled multiprocessors", Proceeding of Supercomputing 1st International Conference, Springer-Verlag Lecture Notes in Computer Science, 1987.

[6] Gropp, W., Keyes, D. "Complexity of parallel implementation of domain decomposition technique for elliptic partial differential equations", SIAM Journal on Sci. and Stat. Comput., Vol 9, No. 2, pp. 312-326, 1988.

[7] Carey, G.; Schmidt, J.; Singh, V.; Yelton, D. "A scalable object oriented finite element solver for partial diferential equations on multiprocessors", ICS 92, ACM 0-89791-4856/92/0007/0387, pp. 387-396, 1992.

[8] Foster, C. "Designing and building parallel programs, Concepts and tools for parallel software engineering", Addison-Wesley Publishing Company, ISBN 0-201-57594-9, 1995.

[9] Berge, M.; Bokhari, S. "A partitioning strategy for nouniform problems on multiprocessors", IEEE Trans. on Computer, Vol. C-36, No. 5, 1987.

[10] Farhat, C. "A simple and efficient automatic FEM domain decomposer", Computer & Structures, Vol. 28, No. 5, pp. 579-602, 1988.

[11] Malone, G. "Automatic mesh decomposition and concurrent finite element analysis for hypercube multiprocessor computer", Comp. Meth. In Applied Mechanical Eng., Vol 70, pp. 27-58, 1988.

[12] Fox, G. "Numerical algorithms for modern parallel computers", Springer-Verlag, 1988.

[13] Diekmann, R.; Meyer, D.; Monien, B. "Parallel decomposition of unstructured FEM-Meshes", tech. Report, Dep. Mathematics and Computer Science, University of Padervorn, 1995. Also in Proceeding onf 2nd International workshop IRREGULAR 95, Springer-Verlag, 1995.

[14] Garey M.; Johnson D., "Computers and intractability", A guide to theory of NP-Completeness, San Francisco, CA, 1979.

[15] Anderson, E. "Parallel Implementation of Preconditioned Conjugate Gradient Methods for Solving Sparse systems of Linear Equations", Tech. Report CSR-805, CRSD, University of Illinois at Urbana-Champaign, 1998.

[16] Heath, M.; Esmond, NG; Peyton, B. "Parallel algorithms for sparse linear systems", Tech. Report ORNL-6889, Mathematical sciences section, Oak Ridge National Laboratory, 1989.