Un algoritmo paralelo de descomposición LU para matrices 'sparse'

Maruja Ortega F. & Alfonso Reinoza
Dpto. de Computación y Tecnología de la Información
Universidad Simón Bolívar
Caracas, Venezuela
emsca!usb!mof@sun.com

Resumen: Se describe un algoritmo paralelo de descomposición LU para matrices 'sparse' no simétricas utilizando una arquitectura MIMD con memoria distribuida. El objetivo es encontrar una factorización de la matriz que minimice el llenado y que a la vez sea estable. Para reducir el llenado se utiliza el criterio de Markowitz. Las matrices se distribuyen de manera envolvente ('wrap-mapping'); la matriz triangular superior U por columnas y la matriz triangular inferior L por filas . Esta distribución es aprovechada para reducir el tiempo de selección del pivote en cada iteración y para acelerar el proceso de reducción.

Palabras Claves: MIMD, descomposición LU, matrices sparse, wrap-mapping.

1 Introducción.

Son muchas las aplicaciones que requieren la resolución de sistemas de ecuaciones lineales de la forma Ax=b de una manera estable. Una de las formas más comunes de llevar esto a cabo es mediante la descomposición de la matriz A en una matriz triangular superior U y una matriz triangular inferior L, con lo cual el sistema Ax=b se resuelve mediante la solución del sistema triangular inferior Ly=b seguido de la solución del sistema triangular superior Ux=y. Una de tales aplicaciones es el algoritmo Simplex. Una versión de este algoritmo, planteada por Bartels y Golub (1969)[3], está basada en la descomposición LU de la matriz básica B asociada a cada iteración.

Este trabajo en particular se enmarca dentro de un proyecto mayor [12] para el desarrollo de una versión paralela del algoritmo de Bartels y Golub, hecho que debió ser tomado en cuenta para la selección de las estructuras de datos así como de las rutinas de comunicación utilizadas.

El algoritmo presentado resuelve el problema de encontrar la descomposición LU de una matriz no simétrica y 'sparse' utilizando una arquitectura MIMD con memoria distribuida.

El objetivo es encontrar una factorización de la matriz que minimice el llenado y que a la vez sea estable, por lo que se debe llegar a un compromiso entre

estos dos criterios. Para resolver el problema del llenado se utilizó el criterio de Markowitz, empleado con éxito por Reid [11] y Gill, Murray, Saunders y Wright [8], en sus algoritmos secuenciales. En este sentido tambien fueron considerados los métodos de ordenamiento P3 y P4 de Hellerman y Rarick, utilizados por Saunders [15], los cuales fueron descartados por estar basados en recorridos tipo DFS de la matriz, siendo este último uno de los algoritmos hasta la fecha catalogados como "dificilmente paralelizables" [14].

La distribución de los datos se ha hecho de manera envolvente ('wrap-mapping'); por columnas para la matriz original B y para la matriz triangular superior U, mientras que la matriz triangular inferior L se distribuye por filas. Por cuanto se supone que la descomposición LU de una matriz será en general utilizada para la posterior solución de sistemas de ecuaciones, y los métodos que hasta la fecha ofrecen mejores resulados requieren que las matrices estén distribuidas de manera envolvente, se desea que las matrices triangulares resultantes preserven esta distribución. Una consecuencia de esto es que durante el proceso de factorización es necesario hacer el pivoteo explicitamente.

2 Algoritmo de Descomposición.

El algoritmo que se plantea recibe como información de entrada una matriz A $m \times n$ distribuida por columnas en forma envolvente y un vector indB que contiene los indices de las m columnas de A que deben conformar la matriz B a la que se aplicará la descomposición LU.

Los pasos del algoritmo ejecutados por cada procesador son los siguientes:

1. Inicialización de estructuras.
2. Para i=1 hasta m hacer
   - 2.1 Seleccion del pivote usando criterio de Markowitz.
   - 2.2 Pivotear filas y columnas.
   - 2.3 Actualizar estructuras.
   - 2.4 Reducción.

El primer paso corresponde a la inicialización de las estructuras correspondientes a las matrices L y U, así como la información necesaria para la seleción de los pivotes aplicando el criterio de Markowitz. La matriz B corresponde a la matriz inicial U. La matriz L inicial es la identidad.

2.1 Estructuras de Datos.

La matriz A se almacena por columnas utilizando tres arreglos unidimensionales StartPos_A, Val_A e IndRow_A, los cuales contienen la

posición donde se se encuentra el primer elemento de cada columna, los valores de los elementos y la fila que ocupa cada uno. Los elementos de una misma columna i se encuentran en posiciones contiguas desde StartPos_A[i] hasta StarPos_A[i+1]-1.

Para almacenar la matriz U se utiliza una estructura de listas por columnas, similar a la utilizada por Reid [14], implementada mediante cuatro arreglos unidimensionales StartPos_U, Val_U, IndRow_U y ColLink, que contienen la posición del primer elemento de cada columna, los valores de los elementos, la fila que ocupan y la posición del próximo elemento de la misma columna. Esta estructura de listas facilita el crecimiento del número de elementos de la matriz ya que no requiere que los elementos de una misma columna se encuentren en posiciones contiguas.

La matriz triangular inferior L será almacenada por filas, utilizando una estructura similar a la anterior.

La distribución de las matrices L y U por filas y por columnas respectivamente permitirá en un futuro comparar el comportamiento de los algoritmos correspondientes para resolver sistemas de ecuaciones triangulares. La matriz B se construye a partir de las columnas de A y del vector indB durante el proceso de inicialización y se almacena en el espacio reservado para U.

Parala aplicación del criterio de Markowiz se requiere información acerca del tamaño o cantidad de elementos distintos de cero de las filas y columnas de B, la cual se almacena en sendos arreglos de tamaño m durante el proceso de inicialización. Igualmente se dispone de una lista ElemsxCol que enlaza las columnas con igual cantidad de elementos organizándolas en orden creciente. Cada procesador dispondrá sólo de la información referente a su conjunto de columnas, mientras que la información de las filas deberá estar replicada en todos.

2.2 Aplicación del Criterio de Markowitz.

El criterio de Markowitz para matrices no simétricas y 'sparse' consiste en seleccionar como elemento pivote en una iteración k, un elemento $B_{ij}^{(k-1)}$ de la matriz reducida $B^{(k-1)}$, obtenida luego de k-1 iteraciones. El elemento seleccionado debe minimizar el producto $M_{ij} = (f_i-1)(c_j-1)$, donde f_i es el tamaño de la fila i y c_j es el tamaño de la columna j. Para que la factorización resultante sea estable se agrega una prueba de estabilidad con un margen de tolerancia μ, $0 < \mu \leq 1$, de forma que una vez hechas las permutaciones se cumpla que $|B_{kk}^{(k-1)}| \geq \mu\ |B_{ik}^{(k-1)}|$ para $k<i\leq m$.

Una aplicación estricta del criterio de Markowitz implica una revisión completa de la matriz reducida.

Para recortar esta búsqueda, en el algoritmo paralelo propuesto, cada procesador selecciona su candidato a pivote buscando sólo en sus q columnas con menos elementos y aplicando como estrategia para resolver empates entre dos elementos, quedarse con el primero de ellos que fue examinado. Luego se comparan los candidatos de los p procesadores para entre ellos seleccionar el mejor. En esta decisión basta tomar en cuanta el criterio de Markowitz pues todos los candidatos ya han pasado la prueba de estabilidad. Los empates se resuelven escogiendo el candidato que se encuentre en la columna con menor índice. Este criterio es definitivo puesto que no puede haber dos columnas con el mismo índice y garantiza que todos los procesadores seleccionarán el mismo elemento. Esto es imprescindible para mantener la consistencia de la información y el buen funcionamiento del algoritmo.

Comparando el comportamiento del algoritmo paralelo propuesto con su equivalente secuencial, se observa que si el mismo valor q que se utiliza en el algoritmo secuencial es utilizado en paralelo por cada uno de los procesadores, entonces el tiempo que tarda cada uno en seleccioner su candidato es el mismo que tarda el algoritmo secuencial en encontrar el pivote definitivo, a lo que debe agregarse, en el caso paralelo, el proceso posterior de elección entre los candidatos de cada procesador. Sin embargo, es de notar que para un valor de q, el resultado que se obtiene en paralelo es equivalente al obtenido en secuencial con q'=q*p. En vista de lo anterior y tomando en cuenta que el valor utlizado para q suele ser pequeño, bastará con fijar q=1 en el algoritmo paralelo para obtener resultados equivalentes a los que se pueden obtener en secuencial utilizando q=p.

2.3 Pivoteo y Actualización de Listas.

Dado que se debe mantener el balance de la carga de trabajo y la distribución envolvente será necesario hacer explicitamente el intercambio de la fila k con la fila pivote y de la columna k con la columna pivote. Sin embargo, las estructuras elegidas para el almacenamiento de L y U, permiten que las permutaciones de filas en U y las de columnas en L puedan hacerse de forma implícita, incorporándolas a las matrices de permutación P y Q.

En el proceso paralelo de intercambio de filas y columnas podrán estar involucrados hasta tres procesadores, a saber, el procesador que tiene la fila y la columna k y los procesadores que tienen la fila pivote y la columna pivote respectivamente. Notese que estos procesadores no necesariamente serán todos

distintos. En caso de haber al menos dos procesadores distintos, se establecerá entre ellos comunicación para hacer el intercambio.

La actualización de la información utilizada en la aplicación del criterio de Markowitz implica que cada procesador deberá disminuir en uno el tamaño de las columnas que tienen elementos en la fila del pivote, y tanto la fila como la columna del pivote deberán ser eliminadas de las listas de elegibles para futuras iteraciones.

Posteriormente, el cálculo de $B^{(k)}$ podrá generar nuevos elementos distintos de cero en algunas filas, en cuyo caso el tamaño de estas deberá ser nuevamente actualizado, al igual que el de las columnas correspondientes y, en consecuencia, la lista ElemsxCol.

2.4 Reducción de la matriz.

El proceso de reducción de la matriz $B^{(k-1)}$ para obtener $B^{(k)}$ comprende la eliminación de los elementos bajo la diagonal en la columna k mediante el cálculo de los multiplicadores, estos serán agregados a la fila correspondiente de L y utilizados para obtener $B^{(k)}$ sumando a cada fila i, $k<i\leq m$, el producto de la fila k por el multiplicador γ_{ik}.

Una alternativa para realizar este proceso es que el procesador que posee la columna k calcule uno a uno los multiplicadores correpondientes a las filas con elementos distintos de cero en esa columna. Cada multiplicador una vez que es calculado es inmediatamente enviado al resto de los procesadores para que actualicen sus columnas. El procesador que posee la fila del multiplicador, incluirá éste en L. El envío del multiplicador apenas es calculado permite que todos los procesadores comiencen a trabajar casi de inmediato.

Otra alternativa es que el procesador con la columna k calcule primero toda la columna de multiplicadores para luego enviarla al resto de los procesadores. Esta alternativa presenta la ventaja que sólo se establece comunicación una vez. La desventaja es que el cálculo del vector de multiplicadores se convierte un cuello de botella ya que durante este tiempo el resto de los procesadores estarían ociosos esperando. En algoritmos para matrices densas este inconveniente es posible reducirlo a un mínimo mediante una especie de encauzamiento en la aplicación del pivoteo parcial. En el caso 'sparse' esto no es posible pues en una iteración k no se sabe cual será columna pivote en la iteración k+1..

Entre estas alternativas la seleccionada fue la primera, pues el problema de serialización parece menor, a lo que se puede agregar que en teoría el tiempo de inicialización ('start-up') en los transputers es casi despreciable.

3 Resultados Computacionales.

El algoritmo presentado ha sido implementado en INMOS C paralelo para ser ejecutado sobre una red de p transputers conectados en anillo bidireccional. Se utilizó como plataforma de comunicaciones el subsistema correspondiente de SIMPAR[12], para el cual se require como mínimo p=3.

Se realizaron pruebas con p=1 y p=4 variando el tamaño y densidad de las matrices. Sin embargo, debido a la limitada capacidad de la configuración disponible no se pudo probar al comportamiento del algoritmo con matrices de grandes dimensiones. Las matrices utilizadas para las pruebas son bases optimas de problemas de progamación lineal.

En todos los casos, para la aplicación del criterio de Markowitz se fijo q=4 para el secuencial y q=1 para el paralelo, obteniendose siempre resultados similares o mejores en el paralelo.

Los resultados obtenidos muestran que para matrices de tamaño pequeño a medio, el balance entre tiempo de cómputo y tiempo de comunicaciones es desfavorable, de forma que con el algoritmo secuencial se obtienen resultados mejores que con el paralelo. Estos resultados se muestran en la tabla 1.

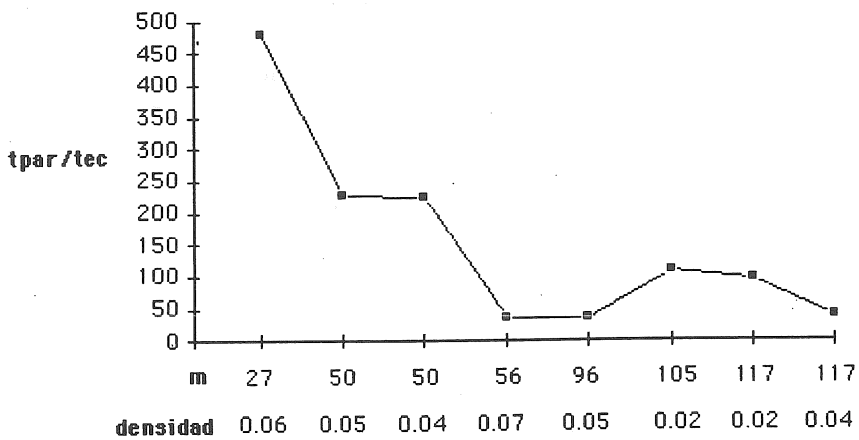

tabla 1.

A pesar de lo anterior, se observa que para un número constante de procesadores, la relación entre los tiempos de ejecución en paralelo y secuencial, tpar/tsec, se hace menor a medida que aumenta el tamaño de la matriz o la

densidad. Este resultado parece indicar que para matrices lo suficientemente grandes, la relación tpar/tsec debe invertirse haciendose más rápido el algoritmo paralelo. Esto último tiene sentido puesto que mientras más grande sea la matriz, mayor es el volumen de cálculo y en un determinado momento este volumen debe superar el over-head de comunicaciones necesario para poder repartir el trabajo. Sin embargo, deberán hacerse más pruebas para estudiar la veracidad de estas observaciones.

Por otro lado, debe notarse que al aumentar el número de procesadores, también debe ser mayor el tamaño de matriz para que el algoritmo paralelo pueda ser eficiente, debido al incremento de la distancia máxima entre procesadores lo cual influye en el tiempo de comunicaciones.

4 Conclusiones.

El algoritmo paralelo presentado resuelve el problema de encontrar la descomposición LU para matrices 'sparse' no simétricas aplicando el criterio de Markowitz para reducir el llenado. Las matrices triangulares resultantes quedan distribuidas de manera envolvente ('wrap-mapping') , lo que facilita la posterior aplicacion de algoritmos de resolución de sistemas de ecuaciones lineales. Las estructuras de almacenamiento elegidas para las matrices L y U se adaptan a requerimientos futuros, como lo es que esta descomposición pueda ser facilmente actualizada durante la aplicacion de la version de Bartels y Golub del algoritmo Simplex.

Para poder establecer el desempeño real del algoritmo se deberán hacer más pruebas aumenando el tamaño de las matrices.

Queda planteado estudiar el comportamiento del algoritmo si se agrega información adicional sobre la distribución de los elementos en las filas de B, o lo que es lo mismo, de U. Igualmente, dado el peso de las comunicaciones en al algoritmo, conviene volver a estudiar la alternativa de enviar en un solo mensaje toda la columna de multiplicadores.

Referencias:

[1] Gita Alaghband, H.F. Jordan. "Sparse Gaussian Elimination with Controlled Fill-in on a Shared Memory Multiprocessor." IEEE Trans. Comp. Vol. 38 Nº11, Nov 1989.

[2] R.H. Bartels. "A Stabilization of the Simplex Method." Numer. Math. 16, 1971.

[3] R.H. Bartels & G.H. Golub. "The Simplex Method of Linear Programming using LU Decomposition." Commun. ACM. 12, 1969.

[4] T.A. Davis & P. Yew. "A Nondeterministic Parallel Algorithm for General Unsymmetric Sparse LU Factorization". SIAM J. Matrix Anal. Appl. Vol11, Nº3. July 1990.

[5] I.S. Duff, A.M. Erisman & J.K. Reid. "Direct Methods for Sparse Matrices." Clarendon Press. Oxford. 1986.

[6] J.J.H. Forrest & J.A. Tomlin. "Updating Triangular Factors of the Basis to Maintain Sparsity in the Product Form Simplex Method. " Math. Programming 2, 1972.

[7] G.A. Geist, C.A. Romine. "LU Factorization Algorithms on Distributed Memory Multiprocessor Architectures." SIAM J. Sci. Stat. Comput. Vol 9. Nº4. July 1988.

[8] P.E. Gill, W. Murray, M.A. Saunders & M.H. Wright. "Maintaining LU Factors of a General Sparse Matrix." Linear Algebra Appl. 88/89, 1987.

[9] M.T. Heath, C.H. Romine. "Parallel Solution of Triangular Systems on Distributed-memory Multiprocessors" Tech. Report ORNL/TM-10384, Oak Ridge National Laboratory, Oak Ridge, TN. 1987.

[10] Guangye Li, T.F. Coleman "A Parallel Triangular Solver for a Distributed Memory Multiprocesor." SIAM J. Sci. Stat. Comp. Vol. 9, Nº 3, Mayo 1988.

[11] J.K. Reid. "A Sparsity-exploiting Variant of tha Bartels-Golub Decomposition for Linear Programming Bases." Math. Programming 24, 1982.

[12] A. Reinoza, M. Lentini, A. Teruel, L. Da Silva, A. Guillen, T. Lozada, O. Naím. "Simplex Paralelo (SIMPAR). Fase 1." Proceedings PANEL'91, XVII Conferencia Latinoamericana de Informática, Vol II. 991-1008.

[13] A. Reinoza et al. "Simplx Paralelo (SIMPAR). Fase 2. Reporte Interno. Dpto. de Computación. Universidad Simón Bolívar. Feb 1992.

[14] W. Rytter & A. Gibbons. "Efficient Parallel Algorithms". Cambridge Univ. Press. 1988.

[15] M.A. Saunders. "A Fast, Stable Implementation of the Simplex Method using Bartels-Golub Updating." Sparse Matrix Computations. Rose & Bunch, Eds. 1976.