Primeros resultados de la paralelización del algoritmo de factorización de Cholesky sobre *clusters* usando cómputos parciales

Gustavo Wolfmann *

Lab. de Computación - Universidad Nacional de Córdoba
Av. Velez Sarsfield 1611, 5000, Córdoba, Argentina, gwolfmann@gmail.com

Abstract. La paralelización de algoritmos con fuerte dependencia de datos no logra grandes mejoras de rendimiento en un entorno de memoria distribuida debido a las sincronizaciones, donde gran parte de los nodos quedan a la espera de datos procesados por otro/s nodo/s. Los nodos en espera pueden realizar cómputos parciales mientras esperan los datos que generan la dependencia siempre que dispongan de datos para poder realizarlos y que se preserven los resultados parciales hasta que sean necesarios. Se presenta los primeros resultados de aplicar esta técnica de paralelización sobre el algoritmo de factorización de Cholesky con mejora de hasta un tercio en el tiempo de ejecución.

1 Introducción

Los algoritmos de factorización de matrices pueden descomponerse en tareas de factorización propiamente dichas y de actualización, estas últimas originadas por la dependencia de datos. Paralelizar bajo un entorno de memoria distribuida impone una sincronización de los resultados que generalmente procesa un solo nodo, quedando los restantes nodos a la espera para poder seguir computando.

Una posibilidad de aprovechar los tiempos de espera es realizar cómputos parciales concebidos como el procesamiento en avanzada de una parte del cómputo a realizarse en etapas posteriores. La estrategia es factible cuando parte de los datos necesarios estén disponibles en el nodo en espera y bajo la condición de preservar dichos resultados hasta el momento en que sean necesarios.

Desde la perspectiva de los patrones de paralelismo[1], puede decirse que los cómputos parciales es una división de tareas de grano "ultra-fino". La división de tareas de un algoritmo para su paralelización, suele considerar a las fórmulas matemáticas como el punto de máxima división. Sin embargo, la división puede ser aun mayor si se despliegan las fórmulas donde intervengan secuencias de datos, como es el caso de aquellas donde participa una sumatoria ($\sum$).

La situación ideal para realizar cómputos parciales es que un nodo disponga de antemano parte de los datos necesarios para realizar cómputos posteriores para utilizar el tiempo de espera de sincronización. Si los datos deben enviarse

* Becario Universidad Nacional de Córdoba

desde un tercer nodo, los beneficios aun pueden existir, pero en menor rango. La mejora del rendimiento depende del algoritmo, del *cluster* donde se ejecute y del tamaño del problema. Este resumen presenta los primeros resultados de la aplicación de esta técnica sobre el algoritmo de factorización de Cholesky.

2 Paralelización del algoritmo de Cholesky

El algoritmo de Cholesky permite factorizar una matriz definida positiva A, de rango n, como $A = L \times L^T$, donde L es una matriz triangular. Los valores de L quedan definidos por:

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} \qquad 1 \leq i \leq n \tag{1}$$

$$l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} \times l_{jk} \right) / l_{jj} \qquad 1 \leq j < i \leq n \tag{2}$$

La primera fórmula determina los valores de la diagonal principal y la segunda los de la parte triangular inferior. Estas fórmulas imponen un orden de procesamiento desde la esquina superior izquierda hacia la inferior derecha para los elementos de la diagonal principal y para los restantes elementos el órden no es único, distinguiéndose una fuerte dependencia de datos.

Algunos intentos de paralelización de este algoritmo son:

•SCALAPACK (Scalable Linear Algebra package) [2] es la referencia de rutinas de álgebra lineal paralelas. Diseñada para correr bajo el modelo de memoria distribuida no tiene en cuenta la arquitectura de nodos *multicore* de la actualidad, por lo que genera un esquema de comunicaciones complejo.

•PLASMA [3] es un desarrollo reciente de rutinas de álgebra lineal diseñada para los actuales equipos *multicore*. Genera un grafo de dependencia de tareas[4], las cuales se ejecutan bajo el patrón *master/worker* [1], logrando un balance de carga por la distribución dinámica de tareas productivo en memoria compartida.

•Kurzak y Dongarra aplicaron la técnica de *look ahead* como una alteración en el orden de las operaciones sobre computadoras *multicore*[5]. Los mejores resultados se alcanzan fijando el órden de ejecución dinámicamente.

•Un algoritmo utilizando *clusters* y MPI con comunicaciones colectivas fue experimentado por Tinetti et. al. [6] distribuyendo los datos en forma de bandas-fila entre los nodos. A pesar de la eficiencia lograda a partir de las comunicaciones colectivas, el balance de carga es pobre ya que la matriz procesada es simétrica.

Nuestros estudios están basados en pruebas de ejecución de 3 algoritmos:

1. La implementación distribuida del algoritmo de Cholesky de Scalapack.
2. El algoritmo de Tinetti utilizando *broadcast*. En un *cluster* con p nodos, los datos son divididos en p bandas filas y distribuidos entre estos, generando p iteraciones. En la iteracíon i, el nodo i computa los valores de la diagonal

principal (l_{ii}). Los restantes nodos esperan dicho resultado, que una vez calculado es replicado utilizando *broadcast*. Los nodos j, con $j > i$, calculan los valores de la columna i en la respectiva banda j, por lo que toda la columna i queda calculada. Su desbalance de carga se grafica en la fig. 1a).

3. Algoritmo con cómputos parciales: el procesamiento y la distribución de datos es igual al anterior. Aquí se anticipan algunas operaciones, lo cual se evidencia al desplegar las fórmulas expuestas. En la sumatoria de la fórm. 1 se procesan valores de una misma fila y la suma se puede desplegar como:

$$\sum_{k=1}^{i-1} l_{ik}^2 = \sum_{k=1}^{h} l_{ik}^2 + \sum_{k=h+1}^{i-1} l_{ik}^2 \qquad 1 \leq h \leq i-1 \quad (3)$$

En el algoritmo anterior se vió que finalizada la iteración h, los valores de dicha columna quedan calculados, por lo que puede realizarse la acumulación parcial de los datos de cada fila hasta dicha columna anticipadamente al momento en que se deba computar la diagonal principal.
La fórmula 2 puede desplegarse como:

$$l_{ij} = \left(a_{ij} - \sum_{k=1}^{h} l_{ik} \times l_{jk} - \sum_{k=h+1}^{j-1} l_{ik} \times l_{jk} \right) / l_{jj} \qquad 1 \leq h \leq j-1 \quad (4)$$

donde para el primer sumatorio estamos en un caso similar al anterior en cuanto a que una vez terminada la iteración h, puede realizarse el producto de $l_{ih} \times l_{jh}$. Aquí se diferencia en que hay un producto de valores de dos filas que pueden residir en diferentes nodos, por lo que en este caso para realizar cómputos parciales sería necesario una comunicación.

En los experimentos se usó precisión simple y programación híbrida OpenMP-MPI [7]. El cómputo intra-nodo se realizó utilizando rutinas de BLAS y Lapack optimizadas para computadoras *multicore* (sgemm, spotrf, ssyrk y strsm).

En tabla 1 se presentan los resultados de los *tests*. Se ejecutó Scalapack (1) y *broadcast* (2) en un *cluster* con resultados similares. En otro, *broadcast* (3) y cómputos parciales en dos variantes, con solo el cómputo parcial de los valores de la diagonal principal (4a), y agregándole el cómputo parcial de los valores de la última banda fila (4b), con mejoras en los tiempos de hasta un 35%. La fig. 1

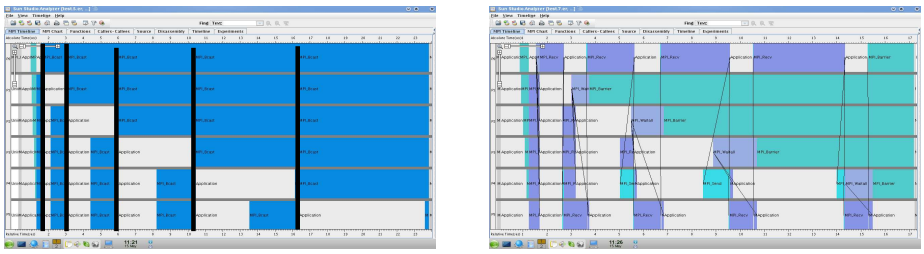
Rango Matriz	**Scalapack nb=64 (1)**	**Broadcast (2)**
12000	13.47 (1.00)	13.53 (1.004)
18000	40.52 (1.00)	38.08 (0.940)
24000	92.40 (1.00)	82.75 (0.896)

(a) *Cluster* con 8 nodos × 4 *threads*, Infiniband, y 32 procs. MPI p/Scalapack.

Broadcast (3)	**Cómput. parcial(4a)**	**Cómput. parcial(4b)**
04.77 (1.00)	04.34 (.901)	03.43 (.719)
13.19 (1.00)	10.61 (.804)	08.55 (.648)
27.20 (1.00)	21.83 (.802)	17.65 (.649)

(b) Ejecución sobre un cluster de 8 nodos con 8 *threads* por nodo e Infiniband.

Table 1: Tiempos de ejecución de los algoritmos usados en segundos



(a) Traza del algoritmo de *broadcast* (b) Traza del algor. de cómp. parciales

Fig. 1: Ejemplos de trazas de la ejecución de algoritmos en un cluster de 6 nodos

muestra las trazas de ejecución de los algoritmos de *broadcast* y de la segunda variante de cómputo parcial. El área blanca corresponde a cómputo y la azul a esperas de sincronización. Se evidencia las menores esperas en el segundo caso.

3 Conclusiones y futuros trabajos

Los primeros resultados para la técnica de "Cómputos Parciales" sobre algoritmos con altas dependencias de datos han sido promisorios con ganancias de hasta un 35% sobre algoritmos tradicionales, gracias a una mejora en el balance de carga al realizar una división de tareas de grano ultra-fino. Sin embargo, obtener dichos resultados es complejo y depende de la formulación matemática del problema, la distribución de los datos y las características del *cluster* donde se corran los programas. En el futuro se planea estudiar la optimización de la aplicación a este algoritmo, extender la técnica a otros algoritmos y modelizarla a los fines de poder aplicarla en forma generalizada.

References

[1] Mattson, T., Sanders, B., Massingill, B.: Patterns for parallel programming. Addison-Wesley Professional (2004)

[2] the Univ. of Tennessee: Scalapack project. `http://www.netlib.org/scalapack/`

[3] the Univ. of Tennessee: The plasma project. parallel linear algebra for scalable multi-core architectures. `http://icl.cs.utk.edu/plasma/`

[4] Agullo, E., Demmel, J., Dongarra, J., Hadri, B., Kurzak, J., Langou, J., Ltaief, H., Luszczek, P., Tomov, S.: Numerical linear algebra on emerging architectures: The plasma and magma projects. Journ. of Physics: Conf. Series **Vol. 180** (2009)

[5] Kurzak, J., Dongarra, J.: Implementing linear algebra routines on multi-core processors with pipelining and a look ahead. In: Applied Parallel Computing. PARA 2006, Umeå, Sweden, Revised Selected Papers. (2006) 147–156

[6] Tinetti, F.G., Romero, F.: Factorización de matrices cholesky: Paralelización y balance de carga. In: XI Congr. Argentino de Cs. de la Computación (CACIC), Concordia, Entre Ríos (2005)

[7] L.A.Smith: Mixed mode mpi / openmp programming. Technical report, Edimburgh Parallel Computing Centre (2000)