Algunas Ideas para la Paralelización de la Factorización Cholesky

Fernando G. Tinetti*
III-LIDI, Facultad de Informática, UNLP
50 y 120, 1900, La Plata
Argentina

Reporte Técnico PLA-001-2011**
Enero 2011

Resumen

En este reporte técnico, inicialmente se describe el método de factorización Cholesky para matrices simétricas definidas positivas. Luego se dan algunas ideas de rendimiento secuencial para código no optimizado y se lo compara con el de la multiplicación de matrices. Se dan dos alternativas de paralelización en función de las dependencias de datos. Se puede considerar que estas alternativas tienen conceptos generales útiles, pero son más orientadas a un curso universitario de procesamiento paralelo que a cómputo paralelo de alto rendimiento (*high performance parallel computing*). Como reporte técnico introductorio, solamente se describe el método de factorización Cholesky más algunas ideas de paralelismo, sin tener en cuenta las ideas más importantes desarrolladas en el contexto de rendimiento paralelo de alto rendimiento.

1. Introducción

La factorización Cholesky, se aplica a una matriz $A \in I\!R^{n \times n}$ simétrica y definida positiva, y es tal que:

$$A = L \times L^T$$

con

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \dots \\ a_{21} & a_{22} & a_{23} & a_{24} & \dots \\ a_{31} & a_{32} & a_{33} & a_{34} & \dots \\ a_{41} & a_{42} & a_{43} & a_{44} & \dots \\ \dots & \dots & \dots & \dots & \ddots \end{pmatrix}$$

y L triangular inferior, es decir,

$$L = \begin{pmatrix} l_{11} & 0 & \dots & \dots & \dots \\ l_{21} & l_{22} & 0 & \dots & \dots \\ l_{31} & l_{32} & l_{33} & 0 & \dots \\ l_{41} & l_{42} & l_{43} & l_{44} & \dots \\ \dots & \dots & \dots & \dots & \ddots \end{pmatrix}$$

cada a_{ij} se calcula como el producto escalar de la fila i de L y la columna j de L^T, es decir

$$a_{ij} = \sum_{k=1}^{n} l_{ik} \times l^T_{kj}$$

donde l^T_{kj} es el elemento kj de L^T. Por la propia definición de matriz traspuesta, $l^T_{kj} = l_{jk}$, y usando esto en la definición de a_{ij},

$$a_{ij} = \sum_{k=1}^{n} l_{ik} \times l_{jk}$$

*Investigador Comisión de Investigaciones Científicas de la Prov. de Bs. As.
**PLA: de la sigla de Parallel Linear Algebra

Es decir que cada a_{ij} se calcula con el producto escalar de la fila i con la columna j de la matriz L. La expresión en función de las matrices involucradas quedaría

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \dots \\ a_{21} & a_{22} & a_{23} & a_{24} & \dots \\ a_{31} & a_{32} & a_{33} & a_{34} & \dots \\ a_{41} & a_{42} & a_{43} & a_{44} & \dots \\ \dots & \dots & \dots & \dots & \ddots \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & \dots & \dots & \dots \\ l_{21} & l_{22} & 0 & \dots & \dots \\ l_{31} & l_{32} & l_{33} & 0 & \dots \\ l_{41} & l_{42} & l_{43} & l_{44} & \dots \\ \dots & \dots & \dots & \dots & \ddots \end{pmatrix} \times \begin{pmatrix} l_{11} & l_{21} & l_{31} & l_{41} & \dots \\ 0 & l_{22} & l_{32} & l_{42} & \dots \\ \dots & 0 & l_{33} & l_{43} & \dots \\ \dots & \dots & 0 & l_{44} & \dots \\ \dots & \dots & \dots & \dots & \ddots \end{pmatrix}$$

Dado que A es simétrica, solamente los elementos de la parte triangular inferior (incluye la diagonal principal) son significativos en cuanto a su cálculo, por lo tanto,

$$a_{ij} = \sum_{k=1}^{n} l_{ik} \times l_{jk}; \qquad j \leq i$$

Es decir que cada a_{ij} se calcula con el producto escalar entre la fila i y la fila j de la matriz L, siendo la fila j menor o igual que la fila i (o la fila i mayor o igual que la fila j). Teniendo en cuenta que L es triangular inferior, los elementos de la fila j sobre la diagonal principal son iguales a 0, es decir $l_{jk} = 0$ para $k > j$, por lo tanto el cálculo de a_{ij} será

$$a_{ij} = \sum_{k=1}^{j} l_{ik} \times l_{jk}; \qquad j \leq i$$

Ahora se puede tomar por separado el cálculo de los elementos de la diagonal principal de A por un lado, a_{ii}, y los elementos bajo la diagonal principal de A, a_{ij} con $j < i$, por el otro. Más específicamente

$$a_{ii} = \sum_{k=1}^{i} l_{ik} \times l_{ik}$$

y

$$a_{ij} = \sum_{k=1}^{j} l_{ik} \times l_{jk}; \qquad j < i$$

En el caso específico de los elementos de la diagonal principal,

$$a_{ii} = \sum_{k=1}^{i} l_{ik} \times l_{ik} = \sum_{k=1}^{i} l_{ik}^2$$

o, lo que es lo mismo,

$$a_{ii} = \sum_{k=1}^{i-1} l_{ik}^2 + l_{ii}^2$$

Por lo tanto, los elementos de la diagonal principal de L, l_{ii}, son tales que

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

Se puede notar, además, que para calcular el elemento de la diagonal principal l_{ii} son necesarios todos los elementos de la misma fila a la que pertenece el elemento de la diagonal principal, es decir los elementos de la i_ésima fila desde el 1 al $i - 1$.

Retomando el cálculo de los elementos que están por debajo de la diagonal principal de A, éstos se calculan con

$$a_{ij} = \sum_{k=1}^{j} l_{ik} \times l_{jk}; \qquad j < i$$

Es decir que son el resultado del producto escalar entre la fila i y la fila j de L, con $j < i$. Expresado de otra manera,

$$a_{ij} = \sum_{k=1}^{j-1} l_{ik} \times l_{jk} + l_{ij} \times l_{jj}; \qquad j < i$$

Y esto implica que los elementos de L debajo de la diagonal principal son tales que

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

Se nota, además, que para calcular estos elementos de L debajo de la diagonal principal, l_{ij} con $j < i$, se utilizan todos los elementos de la fila j (que es "previa" a la i, dado que $i < j$), y los elementos de la fila i desde el 1 hasta el $j - 1$.

Siguiendo el patrón de dependencias de datos para los cálculos de la matriz L, se puede ver por filas de la siguiente manera:

1. l_{11} que pertenece a la diagonal principal de L y es lo único que se puede calcular inicialmente, dado que depende solamente de a_{11}.
2. l_{21}, l_{22} en este orden, dado que teniendo l_{11} se puede calcular *inmediatamente* l_{21} y con l_{21} se puede calcular l_{22}, que pertenece a la diagonal principal de L.
3. A partir de la fila $i - 1$ se pueden calcular los elementos l_{ij}, con $j = 1, \ldots, i - 1$ en este orden, es decir desde el l_{i1} avanzando en la fila, dado que para cada l_{ij} son necesarios los elementos de la misma fila i desde la columna 1 hasta la columna $j - 1$ con los *correspondientes* elementos de la fila j desde la columna 1 hasta la columna $j - 1$. Una vez calculados todos los elementos l_{ij} con $j = 1, \ldots, i - 1$, es decir todos los elementos de la fila i excepto el de la diagonal principal, se puede calcular, justamente, el elemento de la diagonal principal, es decir el elemento l_{ii}.

2. Cantidad de Operaciones

Dado que ya se tiene definido el procesamiento para cada elemento de la matriz L resultado de la factorización de Cholesky sobre una matriz A y la cantidad de elementos de L que surge de la cantidad de elementos de A, se pueden definir los requerimientos de operaciones de punto flotante para la factorización de Cholesky. A partir de una matriz A simétrica definida positiva de $n \times n$ elementos, la cantidad de elementos de L que surge de la factorización de Cholesky sobre A es la cantidad de elementos de una matriz triangular inferior. Esto significa que la primera fila tendrá un elemento, la segunda dos, y así siguiendo. En general, la fila i-ésima tiene i elementos, por lo tanto, la cantidad de elementos de L, $|L|$, será

$$|L| = \sum_{i=1}^{n} i = \frac{n^2 + n}{2} \tag{1}$$

Por lo tanto, solamente quedaría definir la cantidad de operaciones para cada elemento y se podría calcular la cantidad total de operaciones. Sin embargo, dado que la cantidad de operaciones para cada elemento de L no es constante, conviene analizar la cantidad de operaciones por elemento primero y luego el total de acuerdo a la cantidad total de elementos.

La cantidad de operaciones para calcular cada elemento de L es relativamente constante para los elementos de una columna. Los elementos de la primera columna de L requieren una sola operación, que es la de división, dado que usando la definición de la sección anterior,

$$l_{i1} = a_{i1} / \, l_{11}; \qquad 2 \leq i \leq n$$

Los elementos de la segunda columna de L requieren, por su parte, tres operaciones, dado que

$$l_{i2} = \left(a_{i2} - \sum_{k=1}^{1} l_{ik} \times l_{2k} \right) / \, l_{22} = (a_{i2} - l_{i1} \times l_{21}) / \, l_{22}; \qquad 3 \leq i \leq n$$

Los elementos de la tercera columna de L requieren, por su parte, cinco operaciones, dado que

$$l_{i3} = \left(a_{i3} - \sum_{k=1}^{2} l_{ik} \times l_{3k} \right) / \, l_{33} = (a_{i3} - (l_{i1} \times l_{31} + l_{i2} \times l_{32})) / \, l_{33}; \qquad 4 \leq i \leq n$$

En realidad, teniendo en cuenta la definición de los elementos de L dada anteriormente,

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

en general, para la columna j de L con $j \geq 2$ se requieren:

- $j-1$ operaciones de multiplicación.
- $j-2$ operaciones de suma.
- 1 operación de resta.
- 1 operación de división.

Y esto implica que la cantidad de operaciones que se requieren para cualquier elemento de la columna j con $j \geq 2$, co_{ej}, está dada por

$$co_{ej} = j - 1 + j - 2 + 1 + 1 = 2j - 1$$

Y dado que la columna j tiene $n-j+1$ elementos, la cantidad total de operaciones que se requieren para calcular la columna j, con $j \geq 2$, co_j, está dada por

$$co_j = (n-j+1)(2j-1) = 2nj - n - 2j^2 + j + 2j - 1 = (2n+3)j - 2j^2 - n - 1, \qquad j \geq 2$$

Dado que los elementos de la primera columna de L se calculan con una sola operación y hay n elementos en la primera columna, la cantidad total de operaciones para los elementos de la primera columna está dada por

$$co_1 = n$$

Por lo tanto, en total, la cantidad de operaciones para calcular L, co_L, está dada por

$$co_L = n + \sum_{j=2}^{n}(2n+3)j - 2j^2 - n - 1 = n + \sum_{j=2}^{n}(2n+3)j - \sum_{j=2}^{n}2j^2 - \sum_{j=2}^{n}n + 1 \tag{2}$$

Ahora bien,

$$\sum_{j=2}^{n}(2n+3)j = \left(\sum_{j=1}^{n}(2n+3)j\right) - (2n+3) = -2n - 3 + (2n+3)\sum_{j=1}^{n}j$$

Y dado que

$$\sum_{j=1}^{n}j = \frac{n^2+n}{2}$$

Se llega a que

$$\begin{aligned}
\sum_{j=2}^{n}(2n+3)j &= -2n - 3 + (2n+3)\frac{n^2+n}{2} = -2n - 3 + n^3 + n^2 + \frac{3}{2}n^2 + \frac{3}{2}n \\
\sum_{j=2}^{n}(2n+3)j &= n^3 + \frac{5}{2}n^2 - \frac{1}{2}n - 3
\end{aligned} \tag{3}$$

Por otro lado,

$$\sum_{j=2}^{n}2j^2 = \left(\sum_{j=1}^{n}2j^2\right) - 2 = -2 + 2\sum_{j=1}^{n}j^2$$

Y dado que

$$\sum_{j=1}^{n}j^2 = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$$

Se llega a que

$$\begin{aligned}
\sum_{j=2}^{n}2j^2 &= -2 + 2\left(\frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}\right) \\
\sum_{j=2}^{n}2j^2 &= \frac{2}{3}n^3 + n^2 + \frac{1}{3}n - 2
\end{aligned} \tag{4}$$

Además se tiene que

$$\begin{aligned}\sum_{j=2}^{n} n+1 &= (n+1)(n-2+1) = n^2-n+n-1 \\ \sum_{j=2}^{n} n+1 &= n^2-1 \end{aligned} \tag{5}$$

Utilizando la Ec. (3), la Ec. (4) y la Ec. (5) en la Ec. (2) se tiene

$$\begin{aligned} co_L &= n+n^3+\frac{5}{2}n^2-\frac{1}{2}n-3-\left(\frac{2}{3}n^3+n^2+\frac{1}{3}n-2\right)-(n^2-1) \\ &= n^3-\frac{2}{3}n^3+\frac{5}{2}n^2-n^2-n^2+n-\frac{1}{2}n-\frac{1}{3}n-3+2+1 \\ co_L &= \frac{n^3}{3}+\frac{n^2}{2}+\frac{n}{6} \end{aligned} \tag{6}$$

2.1. Tiempo Secuencial con Optimización del Compilador

Dado que actualmente no se cuenta con una estimación del tiempo de ejecución de los experimentos secuenciales, se puede hacer una a partir del rendimiento de la multiplicación de matrices. Dado que el código actual de la factorización no está optimizado ni se usa ninguna biblioteca optimizada, se puede recurrir al rendimiento de la multiplicación de matrices con código fuente no optimizado y compilación optimizada (opción de optimización básica). La Tabla 1 muestra el rendimiento de una computadora en particular para multiplicar distintos tamaños de matrices, y la estimación de tiempo de ejecución para la factorización Cholesky de acuerdo con el rendimiento de la multiplicación de matrices. Se debe notar que

n	Mflop/s (SGEMM)	t(Cholesky)
1000	134.16	2.49
2000	120.82	22.09
3000	47.67	188.89
4000	20.31	1050.92

Tabla 1: Rendimiento y Tiempo Estimado.

la gran diferencia en el rendimiento de una misma máquina con distintos tamaños de problema se debe básicamente a la utilización de un programa sin optimizar a nivel de código fuente, es decir optimizado solamente por el compilador. Cuando se utiliza código totalmente optimizado el rendimiento es muy superior y además casi independiente del tamaño del problema. Recién para los tamaños de matrices de 7000×7000 elementos y 8000×8000 elementos el rendimiento se "estabiliza" en un valor, pero este valor es muy bajo y se supone directamente relacionado con la velocidad de la memoria principal. No se pueden llevar a cabo experimentos con matrices de mayor tama no dado que el requerimiento de almacenamiento de los datos excede el tamaño de la memoria principal.

Se ha implementado la factorización Cholesky sin optimizaciones específicas a nivel de código fuente. Con este programa secuencial se llevaron a cabo los experimentos en función de los cuales se estimará el rendimiento secuencial para esta aplicación. La Tabla compilación optimizada (opción de optimización básica). La Tabla 2 muestra el rendimiento de la factorización Cholesky con distintos tamaños de matrices. Los resultados de rendimiento secuencial utilizando la factorización Cholesky tienen detalles significativa-

n	t(exp.)	Mflop/s(exp.)
1000	1.16	287.79
2000	8.31	321.14
3000	26.84	335.49
4000	61.76	345.55

Tabla 2: Experimentación: Tiempo y Rendimiento.

mente diferentes con respecto a los estimados a partir de la multiplicación de matrices. Uno de los detalles

más importantes son los valores relativamente constantes para diferentes tamaños de matrices. Para la multiplicación de matrices, el rendimiento se reduce significativamente desde el tamaño de matrices de 1000×1000 elementos a matrices de 4000×4000 elementos, más de 1/6 (Tabla 1). En el caso de la factorización Cholesky, el rendimiento no se reduce, sino que aumenta levemente, cerca de 20 % (Tabla 2). Una posible explicación se puede dar en función de la diferencia entre los patrones de cómputo de ambas operaciones. La multiplicación de matrices siempre accede a dos vectores para producir un resultado, y a medida que los tamaños son mayores, la posibilidad de reusar algún dato en memoria cache disminuye proporcionalmente. En el caso de la factorización Cholesky, los cálculos necesarios para cada columna, más que dependientes del tamaño de las matrices, son dependientes de la propia ubicación de las columnas. Los valores de la primera columna de la matriz resultado, por ejemplo, se calculan siempre con una operación, y siempre (re)utilizando el valor de l_{11}. Algo similar ocurre para la segunda columna, pero ahora son dos operaciones, y siempre se (re)utilizan los valores de la segunda fila: l_{21} y l_{22}. Esto significa que la reutilización de datos durante la factorización Cholesky es mayor que durante la multiplicación de matrices y que a mayor dimensión de las matrices la reutilización también es mayor. Por estas razones, se tiene mayor rendimiento para la factorización Cholesky que para la multiplicación de matrices. Además, el rendimiento para la factorización Cholesky aumenta levemente para tamaõs de matrices mayores, mientras que el rendimiento disminuye para la multiplicación de matrices.

Sin embargo, en ambas operaciones se tienen valores de rendimiento muy inferiores a los posibles en las computadoras utilizadas. Más específicamente, con optimización código fuente es posible llegar a rendimiento superior a 2 Gflop/s, mientras que, en el mejor de los casos que se muestran en la Tabla 1 y en la Tabla 2 se tiene 345 Mflop/s.

3. Dependencias de Datos en los Cálculos

La Figura 1 muestra la dependencia en los cálculos de la matriz L. En un primer paso, Paso 1, lo único que puede calcularse es el elemento l_{11} a partir de a_{11}. Una vez calculado l_{11} en el Paso 1, en el

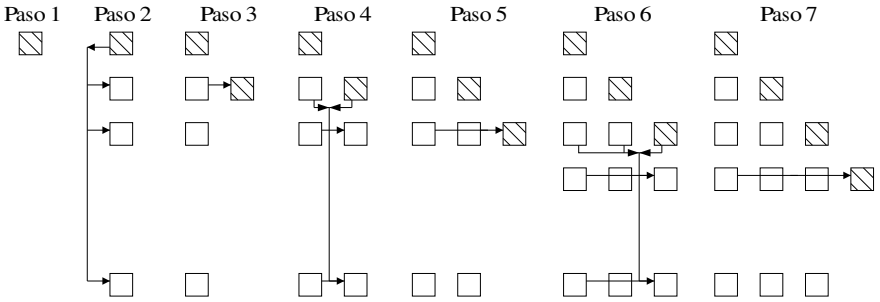

Figura 1: Dependencias de Cálculos.

siguiente paso, Paso 2, se pueden calcular todos los elementos de la primera columna de L debajo de la diagonal principal. Es decir que se pueden calcular los l_{i1} con $2 \leq i \leq n$ utilizando l_{11}. En el Paso 3, una vez que se tiene el elemento l_{21} se puede calcular el *segundo* elemento de la diagonal principal de L: l_{22}, que depende solamente de l_{21}. En el Paso 4 y con los elementos de la fila 2 se pueden calcular todos los elementos de la segunda columna debajo de la diagonal principal, es decir los l_{i2} con $3 \leq i \leq n$. La secuencia se puede continuar, es decir que se itera con el procesamiento:

1. Computar el elemento i-ésimo de la diagonal principal. Si $i = n$ termina.
2. Computar con la fila i-ésima completa todos los elementos de la columna i-ésima debajo de la diagonal principal, es decir los l_{ki} con $i + 1 \leq k \leq n$.

4. Paralelización por Bloques de Filas

Con la idea anterior de dependencias, se puede llegar a una paralelización relativamente sencilla, avanzando por filas de la matriz y considerando a los procesos interconectados en un arreglo unidimensional. Distribuyendo la matriz a factorizar por filas, todos los procesadores tendrán un conjunto de filas consecutivas de la matriz original (y a factorizar) y tendrán datos de todas las columnas de la matriz original

(y a factorizar). La Fig. 2 muestra un ejemplo con una matriz de 8×8 elementos distribuida entre cuatro computadoras, $P_1, \ldots, P_4$ (dos filas por procesador) interconectadas en un arreglo unidimensional de computadoras. Aunque la conectividad entre los procesadores puede ser bidireccional, para este caso en particular los datos (filas de L) se comunican en una sola dirección, tal como lo indican las flechas. El avance de los cálculos en los tres primeros pasos de cómputo y comunicación se muestra en Fig. 3.

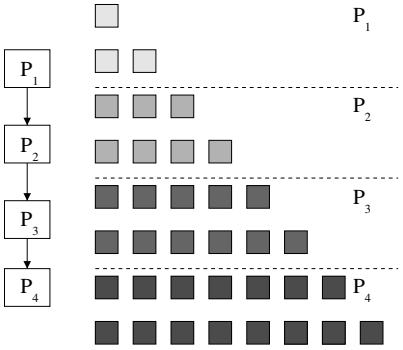

Figura 2: Cuatro Computadoras en un Arreglo Unidimensional.

Inicialmente, en el Paso 1, solamente la computadora P_1 puede hacer un cálculo, que es el de l_{11}. En el Paso 2, P_1 puede enviar l_{11} a P_2 mientras calcula el resto de la primera columna y una vez que computa la primera columna puede calcular l_{22}. En el inicio del Paso 3, P_1 tiene para enviar la segunda fila y P_2 tiene disponible la primera fila de L (l_{11}), para ser reenviada al procesador siguiente y para calcular sus datos de la primera columna de L. Por lo tanto, en este tercer paso se llevan a cabo: a) P_1 envía la segunda fila de L a P_2, b) P_2 reenvía la primera fila de L a P_3 mientras calcula los datos que tiene asignados de la primera columna. La Fig. 4 muestra los pasos 4 a 6 de procesamiento en las cuatro computadoras. En

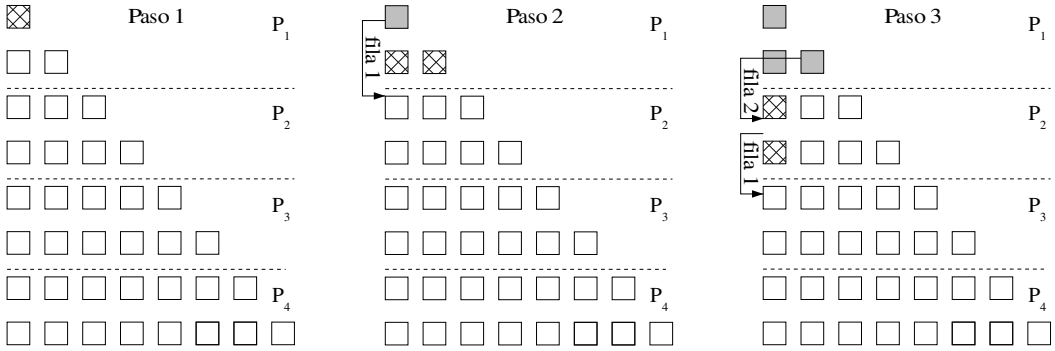

Figura 3: Pasos 1, 2 y 3 de Procesamiento.

el *inicio* del Paso 4, la *situación* en los procesadores es tal que:

- P_1 ya no tiene ninguna tarea para realizar dado que ya ha calculado todos los datos de L que tiene asignados y los ha enviado al procesador P_2.
- P_2 tiene toda la primera columna calculada y la segunda fila disponible para el cálculo de la segunda columna.
- P_3 tiene la primera fila disponible para el cálculo de la primera columna.

Por lo tanto, en este Paso 4:

- P_2 reenvía la segunda fila al siguiente procesador, realiza los cálculos de la segunda columna y completa el cálculo de la tercera fila (l_{33}, en realidad).
- P_3 reenvía la primera fila al siguiente procesador y realiza el cálculo de la primera columna.

En el Paso 5 P_2 comienza a enviar sus propias filas a los procesadores siguientes. De alguna manera, el procesamiento ya está "en régimen" y avanza tal como lo ha hecho hasta ahora. A medida que más

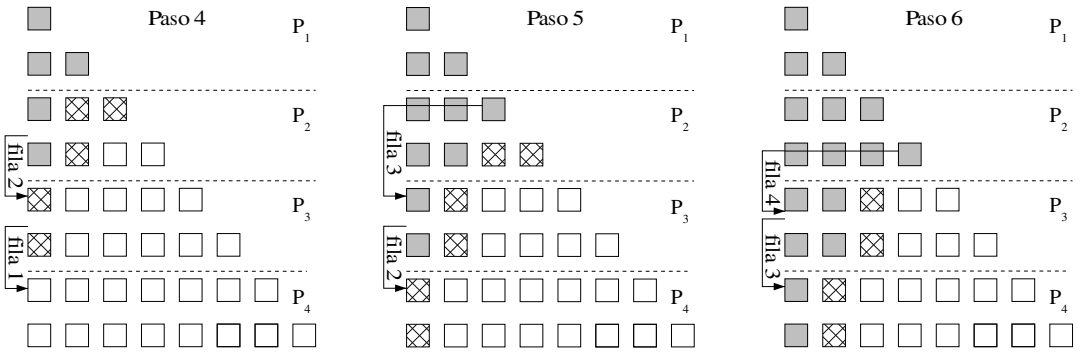

Figura 4: Pasos 4, 5 y 6 de Procesamiento.

pasos de cómputo y comunicación se llevan a cabo, más elementos de la matriz quedan definitivamente computados. De alguna manera, el procesamiento es *pipelined* a lo largo del arreglo unidimensional de máquinas por el que las filas de la matriz L se comunican entre procesadores *vecinos*.

La Fig. 5 muestra los últimos pasos de procesamiento en las cuatro computadoras. Lo que se muestra

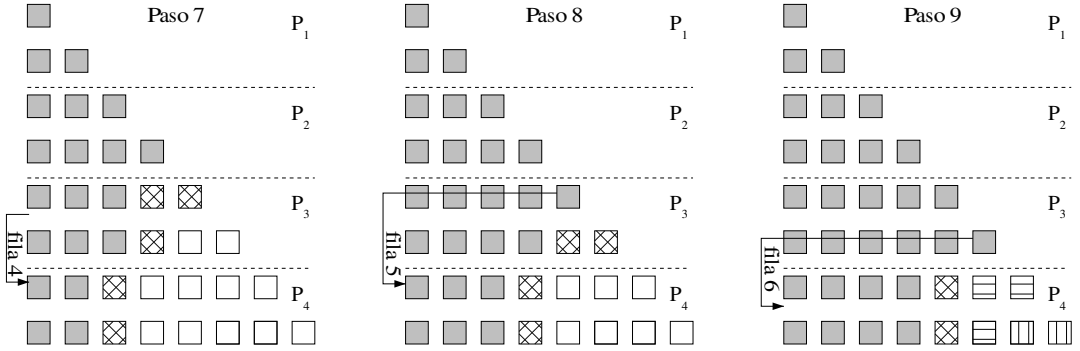

Figura 5: Ultimos Pasos de Procesamiento.

como "Paso 9" en realidad son varias *etapas* sucesivas de procesamiento:

- Mientras se reciben los datos de la Fila 6 se pueden utilizar los datos de la Fila 5 recibidos anteriormente y por lo tanto se calculan los datos de la columna 5. Estos cálculos se muestran con el relleno cruzado con diagonales en el Paso 9.
- Una vez recibidos los datos de la Fila 6 se pueden calcular los datos de la columna 6 y el elemento de la diagonal principal de la Fila 7. Estos cálculos se muestran con el relleno con líneas horizontales en el Paso 9.
- Una vez calculados todos los datos de la Fila 7 se pueden calcular los datos de la columna 7 y el elemento de la diagonal principal de la Fila 8. Estos cálculos se muestran con el relleno con líneas verticales en el Paso 9.

Se debería tener en cuenta este esquema de paralelización (mostrado en la Fig. 3 en la Fig. 4 y en la Fig. 5) para definir el procesamiento a realizar en cada máquina siguiendo el modelo SPMD (Single Program, Multiple Data). Inicialmente, la idea es que cada proceso en cada máquina siga una secuencia de recibir-enviar-procesar tal como lo muestra la Fig. 6. Sobre el pseudocódigo de la Fig. 6 se debe avanzar en varios sentidos para definir mejor el procesamiento. En principio, se debe establecer de manera precisa qué fila se recibe, qué fila se envía y qué columna se computa en cada iteración. También se debe tener en cuenta que la primera computadora no recibirá nunca una fila de una computadora o proceso *anterior* y la última no enviará nunca una fila a una computadora o proceso *siguiente*, tal como se ve en la Fig. 2. También es necesario avanzar en el nivel de detalle para diferenciar los casos en que las columnas se computan con datos locales de los casos en que se deben esperar datos de la computadora anterior para utilizarlos en los cálculos locales. Más específicamente, todos los datos de las filas locales se usan en los

```
for (todas las columnas a resolver)
{
  Recibir fila del proceso anterior
  Enviar fila al proceso siguiente
  Computar columna correspondiente
}
```

Figura 6: Iteración de Procesamiento en Cada Computadora.

cálculos de las columnas que les corresponden (fila i - columna i) y parte de estas columnas son locales (excepto en el caso de la última fila asignada a cada procesador). También es necesario definir exactamente el significado de la condición `todas las columnas a resolver`, que seguramente está relacionada con la partición/distribución de los datos de la matriz. Asumiendo como hasta ahora que se distribuyen los elementos de la matriz por bloques de filas, es claro que todas las computadoras tendrán elementos de las primeras columnas. Llamando `loccols` a la cantidad de columnas locales y numerando las computadoras desde 0, se puede llegar al pseudocódigo del proceso que se ejecuta en la computadora P_k de la Fig. 7. También se muestra en la Fig. 7 que la computadora P_0 de alguna manera comienza su procesamiento local de manera diferente que el resto de las computadoras.

```
/* El procesador 0 comienza de manera especial */
if (k == 0)
  Computar el primer elemento de la diagonal (0, 0)

/* Columnas locales de las filas correspondientes */
for (i = 0; i < loccols; i++)
{
  if (Fila i de un proceso anterior)
    Recibir fila i del proceso anterior

  if (Hay proceso siguiente)
    Enviar fila i al proceso siguiente

  Computar columna i con la fila i

  if (fila i+1 es local)
    Computar el elemento (i+1, i+1)
}
```

Figura 7: Cómputo y Comunicaciones en la computadora P_k.

Aumentar el nivel de detalle sobre el pseudocódigo de la Fig. 7 necesariamente implica avanzar también sobre los detalles del particionamiento de los datos de la matriz, por ejemplo. De hecho, si la computadora P_k tiene asignadas las filas f_{ki} a f_{kf} numeradas desde f_{ki} hasta $f_{kf}-1$, la computadora P_k tendrá parte de los datos de las primeras f_{kf} columnas. Si se numeran las columnas desde 0 en adelante, la computadora P_k tendrá parte de los datos de las columnas 0 a $f_{kf}-1$, es decir que tendrá datos de las primeras f_{kf} columnas. Por lo tanto, teniendo en cuenta estas definiciones de f_{ki} y f_{kf} usadas en el pseudocódigo como variables `f_ki` y `f_kf` y considerando que `numtasks` es la cantidad total de computadoras, numeradas desde 0 hasta `numtasks-1`, se pueden especificar mejor las condiciones que aparecen en el pseudocódigo de la Fig. 7:

- `(Fila i de un proceso anterior)` $\Longrightarrow$ `(i < f_ki)`
- `(Hay proceso siguiente)` $\Longrightarrow$ `(k < (numtasks-1))`
- `(fila i+1 es local)` $\Longrightarrow$ `(((i+1) >= f_ki) && ((i+1) < f_kf))`

Y además usando estas definiciones se puede especificar con mayor nivel de detalle el procesamiento necesario para `Enviar fila i al proceso siguiente` como se muestra en la Fig. 8, dado que se puede identificar ahora si la fila a enviar es local o ha sido recibida desde la computadora *anterior*. De la misma manera, también se puede especificar con mayor nivel de detalle `Computar columna i con la fila i` como se muestra en la Fig. 9, dado que ahora se tienen los datos/valores necesarios para especificar el

```
if ((i >= f_ki) && (i < f_kf))
  Enviar fila i local;
else
  Enviar fila i recibida;
```

Figura 8: Envío de la Fila i desde la Computadora P_k.

```
if (i < f_ki)
  primf = 0;
else
  primf = i-f_ki;
for (f = primf; f < ((f_kf-f_ki)+1); f++)
  calcular elemento (f, i)
```

Figura 9: Cálculo de una columna en la Computadora P_k.

principio y el final de cada columna a partir de las variables `f_ki` y `f_kf`. En la Fig. 9 también se puede notar que cuando se distribuyen los datos entre las distintas computadoras, en particular los bloques de filas, cada computadora tendrá *solamente* los datos propios. La iteración del final de la Fig. 9 siempre comienza desde 0 si las columnas se procesan con datos recibidos desde la computadora anterior o desde la fila que corresponde localmente a la fila *global* de la matriz en la factorización Cholesky.

Finalmente, la Fig. 10 muestra el pseudocódigo del proceso que se ejecuta en la computadora P_k con la incorporación de los detalles previos. En la La Fig. 10 también se incluyen algunos comentarios, para aclarar a qué corresponden algunas secuencias de operaciones. Se puede notar que solamente la computadora P_0 comienza el procesamiento local de manera diferente del resto, dado que nunca recibe (*necesita*) ningún dato de otras computadoras y luego, cada computadora, para cada fila/columna sigue realizando de manera *independiente de las demás* la secuencia de pasos:

- Recibir fila del proceso anterior si debe hacerlo, de acuerdo a su *identificación* y la iteración (fila/columna) que se está resolviendo.
- Enviar fila al proceso siguiente, nuevamente dependiendo de su identificación y de la iteración que se está resolviendo.
- Computar columna correspondiente, con el agregado de que se incluye explícitamente también el cómputo del elemento de la columna *siguiente* en la computadora correspondiente.

Es interesante notar que el procesamiento de la factorización Cholesky definido por el pseudocódigo de la Fig. 10 tiene características similares a la mayoría de los algoritmos paralelos definidos en el área de álgebra lineal:

- Programación con pasaje de mensajes. Es el modelo de programación que se supone más adecuado (de mayor rendimiento, en realidad) en las arquitecturas paralelas de memoria distribuida.
- Procesamiento SPMD (Single Program, Multiple Data). Es el modelo de procesamiento más simple de especificar, dado que se define un único proceso que se ejecuta sobre todos los procesadores de la arquitectura paralela disponible. Se asume que es el que mejor escala, dado que *a priori* los procesos son asincrónicos y además no hay que definir nada *nuevo* o *en función de* la cantidad de computadoras disponibles.
- Todas las comunicaciones son punto-a-punto. De hecho, se define y se utiliza la interconexión de las computadoras en un arreglo unidimensional tal como el de la Fig. 2, donde cada computadora tiene una o dos computadoras directamente conectadas, dependiendo de su ubicación (identificación) en el arreglo.

4.1. Breve Análisis de Rendimiento de la Paralelización por Bloque de Filas

Si se analiza la paralelización por filas propuesta en la sección anterior a partir de la Fig. 3, se pueden identificar problemas de rendimiento casi inmediatamente. El más evidente es el que surge de todas las factorizaciones "right-looking" que transforman algún tipo de matriz densa en una o más matrices triangulares tales como LU, QR y Cholesky. En todas estas factorizaciones, las distribuciones

```
/* El procesador 0 comienza de manera especial */
if (k == 0)
  Computar el primer elemento de la diagonal (0, 0)

/* Columnas locales de las filas correspondientes */
for (i = 0; i < loccols; i++)
{
  if (i < f_ki)
    Recibir fila i del proceso anterior

  /* Enviar fila i al proceso sgte. */
  if (k < (numtasks-1))
    if ((i >= f_ki) && (i < f_kf))
      Enviar fila i local a k+1;
    else
      Enviar fila i recibida a k+1;

  /* Calcular la columna i */
  if (i < f_ki)
    primf = 0;
  else
    primf = i-f_ki;
  for (f = primf; f < ((f_kf-f_ki)+1); f++)
    calcular elemento (f, i);

  /* Calcular el elemento de la diagonal sgte. */
  if (((i+1) >= f_ki) && ((i+1) < f_kf))
    Computar el elemento (i+1, i+1);
}
```

Figura 10: Procesamiento en la computadora P_k.

por bloques de filas consecutivas producen inconvenientes claros de rendimiento: a medida que se avanza en el procesamiento, las primeras computadoras dejan de tener trabajo para realizar, comienzan a estar cada vez más ociosas y el desbalance de carga es immediato. En el caso de la factorización Cholesky, se agrega que, de hecho, las primeras computadoras tienen menor cantidad de datos para procesar que las últimas si la cantidad de filas por bloque es constante: $f_{kf} - f_{ki} = c$, $0 \leq k \leq numtasks - 1$.

5. Paralelización por Filas con Distribución Cíclica

La forma más usual de resolver el problema de desbalance de carga de las factorizaciones right-looking es la distribución cíclica de los datos. La idea básica es que todas las computadoras tengan datos de la matriz a procesar en la *mayoría* de las iteraciones. En el caso específico de la factorización Cholesky esto se logra de manera sencilla si todas las computadoras tienen filas que pertencen a distintos sectores de la matriz original, desde las primeras filas hasta las últimas. Y esto es justamente lo que se logra con la distribución cíclica de las filas entre las computadoras, que establece que la fila i de la matriz estará asignada a la computadora P_k si

$$k = i \bmod numtasks \qquad 0 \leq i \leq n-1 \tag{7}$$

donde se asume que las n filas se numeran desde 0 hasta $n-1$ y las $numtasks$ computadoras se numeran desde 0 hasta $numtasks - 1$. La Fig. 11 muestra esta distribución para una matriz de 8×8 elementos a distribuir entre cuatro computadoras. Siguiendo el ejemplo de la Fig. 11, y comparándolo con el algoritmo de la sección anterior, la computadora P_0 tendrá elementos de las filas 0 y 4 y por lo tanto tendrá elementos a procesar hasta la iteración 4 inclusive. Dado que todas las computadoras tendrán elementos de filas no consecutivas, la computadora P_k recibirá filas de todas las demás computadoras, no solamente de las *anteriores*. Esto hace que la interconexión entre las computadoras en anillo, tal como lo muestra la Fig. 12, sea más apropiada que la de un arreglo unidimensional simple. La idea ahora es seguir haciendo la factorización Cholesky de manera distribuida, pero con los datos asignados a las computadoras de manera

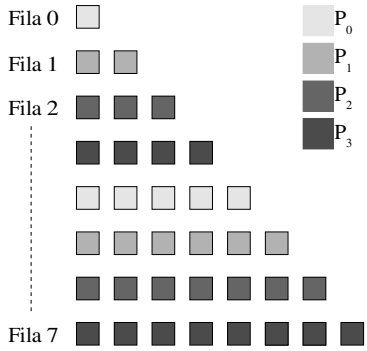

Figura 11: Distribución Cíclica de una Matriz.

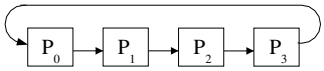

Figura 12: Cuatro Computadoras en un Anillo.

diferente a lo que se comentó previamente. La Fig. 13 muestra los primeros pasos de procesamiento en las cuatro computadoras con la distribución cíclica de filas. Inicialmente (Paso 1), sólo la computadora

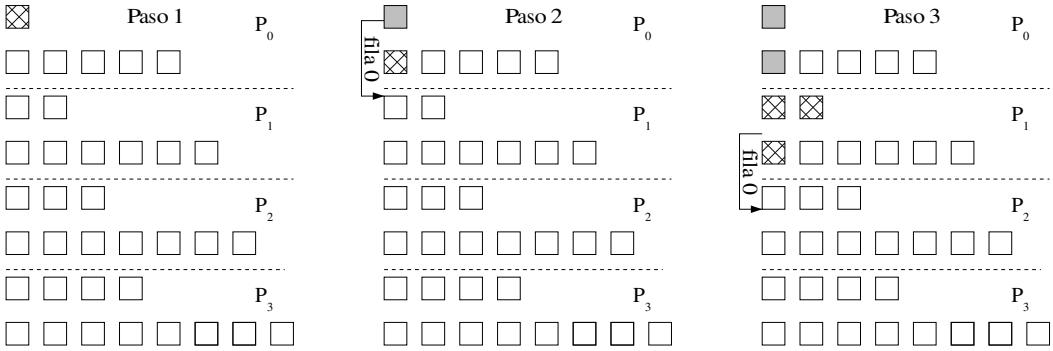

Figura 13: Primeros Tres Pasos de Procesamiento con Distribución Cíclica.

P_0 puede hacer un cálculo, que es el de l_{00}. Una vez que está calculado l_{00}, en el Paso 2 se envía este elemento a las demás computadoras vía P_1 mientras en P_0 se calculan los elementos de la columna 0. En el Paso 3, la computadora P_1 envía el elemento de la fila 0 a la computadora *siguiente*, calcula los elementos de la primera columna y luego calcula el elemento l_{11}, con el que se completan los elementos de la segunda fila, la Fila 1.

La Fig. 14 muestra el Paso 4, Paso 5 y Paso 6 de procesamiento en las cuatro computadoras con la distribución cíclica de filas. La *situación* en el inicio del Paso 4 es tal que:

- Todos los elementos de la Fila 1 están calculados en la computadora P_1 y *listos* para ser enviados a la computadora P_2.
- En la computadora P_2 se tienen los datos de la Fila 0, y pueden ser utilizados localmente *mientras* son enviados a la computadora P_3.

Por lo tanto, en el Paso 4 se lleva a cabo el cómputo de los elementos de la Columna 1 en P_1 mientras se envían los datos de la Fila 1 desde P_1 a P_2, y se lleva a cabo el cómputo de los elementos de la columna 0 en P_2 mientras se envían los datos de la Fila 0 desde P_2 a P_3. Tal como se muestra en la Fig. 14 en el Paso 5 de procesamiento todo lo que se puede hacer es:

- En P_2 se envía la Fila 1 a P_3 *mientras* se calculan los datos de la Columna 1 y luego se puede completar el cálculo del elemento de la diagonal principal de la Fila 2.

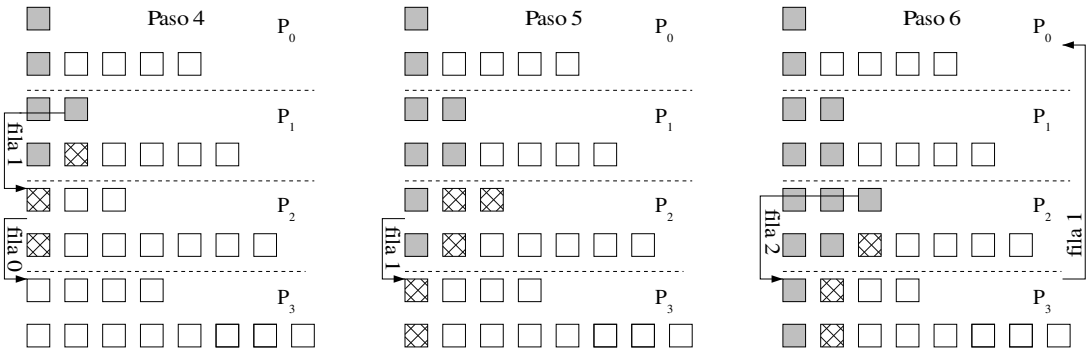

Figura 14: Paso 4, Paso 5 y Paso 6 de Procesamiento con Distribudión Cíclica.

- En P_2 se recibe la Fila 1 *mientras* se calculan los datos de la Columna 0. Se debe notar que no se envía ningún dato desde P_3 a P_0 dado que la única fila disponible (totalmente calculada) en P_3 es la Fila 0 y esa fila es local a P_0, por lo tanto no tiene sentido enviársela.

En el Paso 6 de procesamiento, la computadora P_2 envía la Fila 2 mientras computa los datos locales de la Columna 2. También en este Paso 6, la Fila 1 se envía desde P_3 a P_0 (que la utilizará para calcular sus datos en la Columna 1), mientras computa los datos locales de la Columna 1.

La Fig. 15 muestra el Paso 7, Paso 8 y Paso 9 de procesamiento en las cuatro computadoras con la

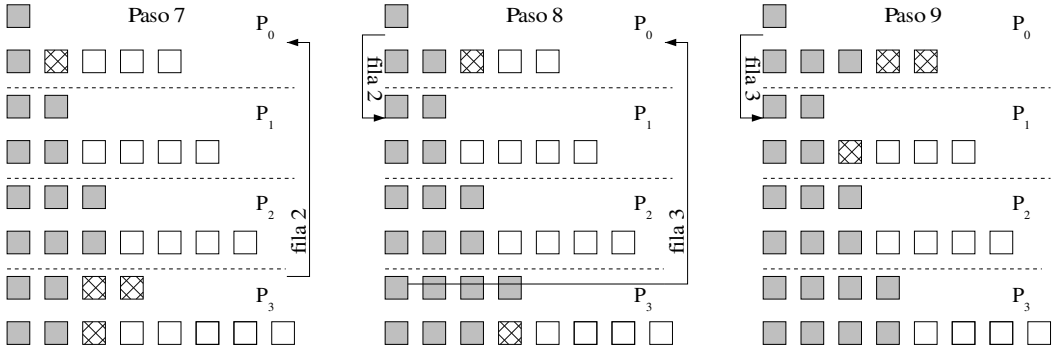

Figura 15: Paso 7, Paso 8 y Paso 9 de Procesamiento con Distribución Cíclica.

distribución cíclica de filas. La *situación* en el inicio del Paso 7 es tal que:

- Los elementos de la Fila 1 están disponibles en P_0 para ser utilizados en el cálculo de los elementos locales de la Columna 1.
- Todos los elementos de la Fila 2 están están disponibles en P_3 para ser utilizados en el cálculo de los elementos locales de la Columna 2.

Por lo tanto, en el Paso 7 se lleva a cabo el cómputo de los elementos de la columna 1 en P_0, se lleva a cabo el cómputo de los elementos de la columna 2 en P_3 mientras se envían los datos de la Fila 2 desde P_3 a P_0. Una vez calculados los elementos de la Columna 2 en P_3 se puede calcular el elemento de la diagonal principal l_{33} en P_3 con lo que se completa el cálculo de la Fila 3 en P_3.

Tal como se muestra en la Fig. 15 en el Paso 8 de procesamiento todo lo que se puede hacer es:

- En P_0 se envía la Fila 2 a P_1 *mientras* se calculan los datos locales de la Columna 2 y se reciben los datos de la Fila 3 desde P_3.
- En P_1 se recibe la Fila 2.
- En P_3 se envían los datos de la Fila 3 a P_0 mientras se calculan los datos locales de la Columna 3.

En el Paso 9 de procesamiento, la computadora P_0 envía la Fila 3 a P_1 mientras computa los datos locales de la Columna 3 y luego puede calcular el elemento de la diagonal principal l_{44}. También en este Paso 9, P_1 recibe los datos de la Fila 3 mientras computa los elementos locales de la Columna 2.

La Fig. 16 muestra los últimos pasos de procesamiento de la factorización Cholesky correspondientes a la distribución cíclica de las filas de una matriz de 8×8 elementos en cuatro computadoras. Es interesante

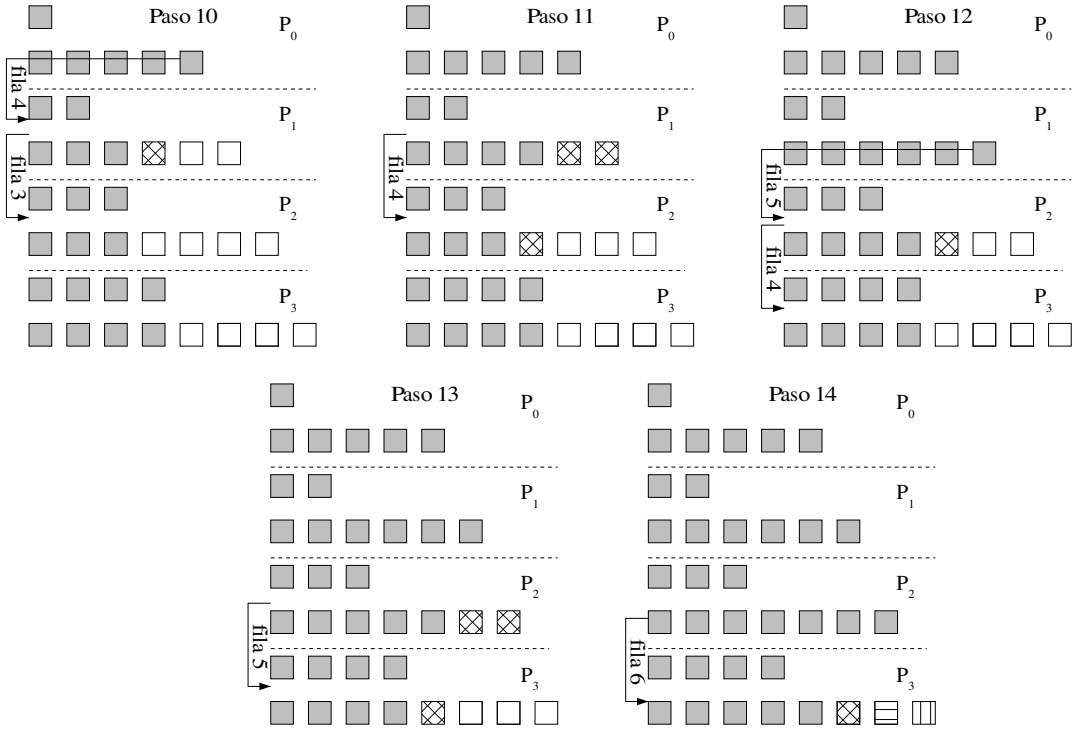

Figura 16: Ultimos Pasos de Procesamiento con Distribución Cíclica.

notar que si bien es cierto que se tiende a mantener el balance de carga por la distribución cíclica, también es cierto que las últimas iteraciones de la factorización ya no se pueden llevar a cabo en una única computadora. Por un lado, el beneficio de mantener al *máximo* (en realidad, hasta las últimas iteraciones) el balance de carga es evidente, pero por otro, las últimas iteraciones requieren comunicación de datos, algo que no se tiene cuando los datos se reparten por bloques de filas (Fig. 5, Paso 9).

Con la distribución cíclica de filas, el pseudocódigo de la Fig. 7 se mantiene en líneas generales, pero algunos de los detalles de implementación se deben cambiar de acuerdo con la *nueva* forma de distribución de los datos entre las computadoras. En lo referente a las condiciones del pseudocódigo de la Fig. 7:

- `(Fila i de un proceso anterior)`, ahora será `(Fila i de otro proceso)` que implica usar casi directamente la Eq. (7): `(Fila i de otro proceso)` $\Longrightarrow$ `(k != (i % numtasks))`
- `(Hay proceso siguiente)`, ahora será `(Fila i a proceso siguiente)`, y nuevamente usando la Eq. (7): `(Fila i a proceso siguiente)` $\Longrightarrow$ `(((k+1) mod numtasks) != (i mod numtasks))`.
- `(fila i+1 es local)` $\Longrightarrow$ `( ((i+1) < n) && (k == ((i+1) mod numtasks)) )`.

También se deben adecuar las especificaciones de `Enviar fila i al proceso siguiente` para tener en cuenta la distribución cíclica y la definición de `Computar columna i con la fila i`. En ambos casos se debe volver a utilizar la Ec. (7) dado que es la que define la *asignación* de filas a computadoras. La Fig. 17 muestra la especificación de `Enviar fila i al proceso siguiente`. De manera análoga, el

```
if (k == (i mod numtasks))
  Enviar fila i local;
else
  Enviar fila i recibida;
```

Figura 17: Envío de la Fila i desde la Computadora P_k con Distribución Cíclica.

pseudocódigo de la Fig. 18 muestra la definición de `Computar columna i con la fila i` donde se utiliza la variable `locrows` que proporciona la cantidad de filas locales en la computadora P_k. Esta cantidad es

```
primf = i / numtasks;
for (f = primf; f < locrows; f++)
  calcular elemento (f, i)
```

Figura 18: Cálculo de una columna en la Computadora P_k con Distribución Cíclica.

relativamente sencilla de obtener: será $locrows = n/numtasks$ si $k \geq n$ mód $numtasks$ (se da para todo k cuando n, la cantidad de filas y columnas de la matriz a factorizar, es múltiplo de $numtasks$, como en el ejemplo de la Fig. 11) o $locrows = n/numtasks+1$ en caso contrario, es decir si $k < n$ mód $numtasks$ (se da para las primeras n mód $numtasks$ computadoras cuando cuando n, la cantidad de filas y columnas de la matriz a factorizar, no es múltiplo de $numtasks$). Es este último caso, las primeras n mód $numtasks$ computadoras tienen una fila más que el resto de las computadoras.

Finalmente, la Fig. 19 muestra el pseudocódigo del proceso que se ejecuta en la computadora P_k con la incorporación de los detalles previos, definidos en función de la distribución cíclica de las filas de la matriz

```
/* Calculo de filas locales */
locrows = n / numtasks;
if ( k < (n % numtasks) )
  locrows = locrows + 1;

/* El procesador 0 comienza de manera especial */
if (k == 0)
  Computar el primer elemento de la diagonal (0, 0)

/* Todas las comp. tienen elementos de todas las cols. */
for (i = 0; i < n; i++)
{
  if (k != (i % numtasks))
    Recibir fila i del proceso (k-1) mod numtasks;

  /* Enviar fila i al proceso sgte. */
  if ( ((k+1) mod numtasks) != (i mod numtasks) )
    if (k == (i mod numtasks))
      Enviar fila i local a (k+1) mod numtasks;
    else
      Enviar fila i recibida a (k+1) mod numtasks;

  /* Calcular la columna i */
  primf = i / numtasks;
  for (f = primf; f < locrows; f++)
    calcular elemento (f, i)

  /* Calcular el elemento de la diagonal sgte. */
  if ( ((i+1) < n) && (k == ((i+1) mod numtasks)) )
    Computar el elemento (i+1, i+1);
}
```

Figura 19: Procesamiento en la computadora P_k con Distribución Cíclica.

a factorizar. En la La Fig. 19 también se incluyen algunos comentarios, para aclarar a qué corresponden algunas secuencias de operaciones. Y también en este caso, el procesamiento de la factorización Cholesky definido por el pseudocódigo de la Fig. 19 tiene características similares a la mayoría de los algoritmos paralelos definidos en el área de álgebra lineal, tal como se explicó previamente:

- Programación con pasaje de mensajes.
- Procesamiento SPMD (Single Program, Multiple Data).
- Todas las comunicaciones son punto-a-punto.

Comparando el pseudocódigo del procesamiento con la distribución con bloques de filas consecutivas de la Fig. 10 con el del procesamiento con la distribución cíclica de filas de la Fig. 19 la diferencia más importante está, justamente, en la iteración *principal*. Cuando la matriz se distribuye por bloques de filas consecutivas, cada proceso tiene datos de un subconjunto de columnas, más específicamente, las *primeras* f_{kf} columnas (según la notación/terminología de la sección 4). Cuando la matriz se distribuye cíclicamente por filas, todas las *numtasks* computadoras tienen datos de todas las columnas excepto, a lo sumo, las últimas *numtasks* filas, es decir que todas las computadoras tendrán datos para procesar en todas las iteraciones de la factorización Cholesky por la propia distribución de los datos. Esto necesariamente implica que las primeras computadoras en el caso de la distribución por bloques de filas consecutivas tienen mucha menos carga de procesamiento que en el caso de distribuir las filas de manera cíclica, en cuyo caso todas las computadoras tienen que procesar datos de *casi* todas las columnas. Otra de las diferencias que se ha puntualizado previamente es la de la necesidad de comunicaciones para resolver todas las filas en todas las computadoras. En el caso de los bloques de filas consecutivas, una vez que se comienzan a procesar los datos con filas *locales*, no se tiene necesidad de recibir datos de ninguna otra computadora. En el caso de la distribución cíclica de filas, para concluir el procesamiento de una fila en particular en una computadora, se necesita, por ejemplo, la fila inmediatamente anterior que justamente está en la computadora *inmediatamente anterior* de acuerdo con la numeración determinada para una cantidad de *numtasks* computadoras: P_0, ..., $P_{numtasks-1}$.

6. Experimentación

Se realizaron experimentos con los algoritmos paralelos que se describen en la sección anterior, con el objetivo de evaluar su rendimiento. Los experimentos se llevaron a cabo en un cluster de PCs homogéneo, las computadoras son tal como la que se utilizó para evaluar el rendimiento secuencial en una de las secciones anteriores. Todas las computadoras están interconectadas con una red Ethernet de 100 Mb/s con un único switch Ethernet (es decir que se tiene *switching* completo). El rendimiento se evalúa en función del speedup obtenido, que es una métrica muy conocida y útil en el contexto de los clusters homogéneos.

La Fig. 20 muestra los resultados preliminares de rendimiento al utilizar el algoritmo con distribución por bloques de filas. Los tamaños de matrices utilizados varían entre 1000 × 1000 elementos y 4000 × 4000

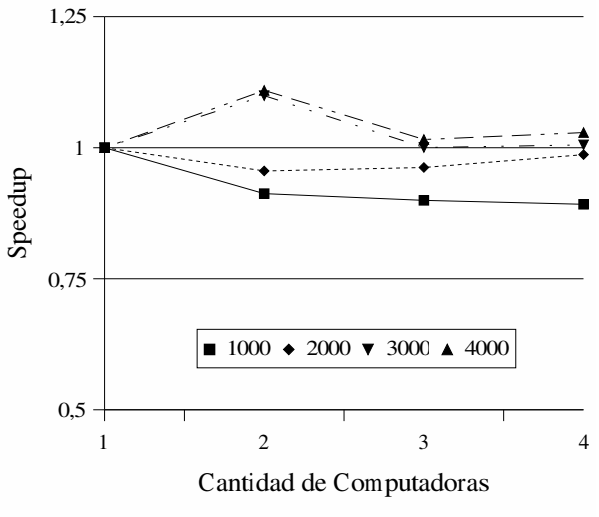

Figura 20: Resultados: Bloques de Filas.

elementos para tener un rango relativamente amplio de valores. La cantidad de computadoras utilizadas varía entre 1 y 4. No se realizaron experimentos con más computadoras porque con estas cantidades queda claro que el rendimiento obtenido es muy bajo, mucho menor que los valores de rendimiento máximo. Además, tienden a disminuir cuando la cantidad de computadoras es mayor. Al analizar el perfil de tiempos de ejecución, se nota rápidamente lo que se puntualiza antes respecto al desbalance de carga: las computadoras con los primeros bloques de filas terminan su tarea de procesamiento mucho antes que las computadoras con los últimos bloques de filas.

La Fig. 21 muestra los resultados obtenidos con el algoritmo con distribución cíclica de filas. Si bien los tamaños de matrices se mantienen entre 1000 × 1000 elementos y 4000 × 4000 elementos, las cantidades de computadoras utilizadas varían entre 1 y 14. Si bien en todos los casos se tienen valores de speedup

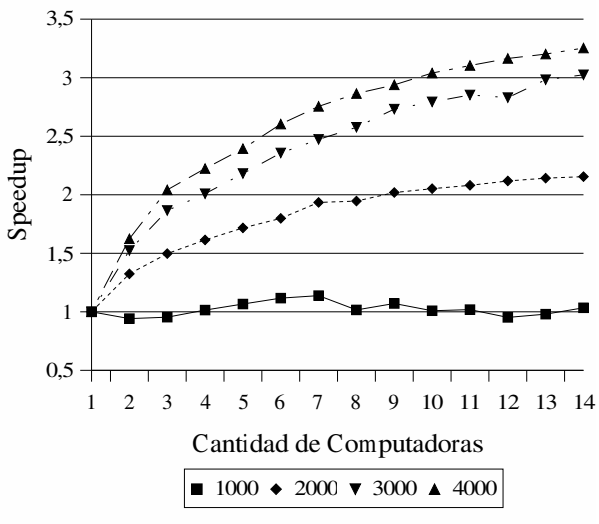

Figura 21: Resultados: Filas Cíclicas.

que no son cercanos a los óptimos, se tienen algunas características de rendimiento ventajosas respecto del algoritmo anterior. Con la excepción de las matrices con menor cantidad de elementos, 1000 × 1000, en todos los casos se tiene mejor rendimiento cuando se agregan computadoras, es decir que el speedup es creciente cuando la cantidad de computadoras es mayor. También se da otra característica usual en el contexto de álgebra lineal en paralelo: la capacidad de cómputo disponible se aprovecha mejor cuando la cantidad de datos a procesar es mayor. Es por esta razón que el rendimiento obtenido con las matrices de 4000 × 4000 elementos es el mayor de todos.

7. Conclusiones y Trabajo Futuro

En este reporte técnico se han presentado algunas ideas interesantes sobre la paralelización de la factorización de matrices Cholesky. Aunque no se han aplicado las ideas y resultados más importantes de cómputo paralelo de alto rendimiento se puede comprobar la dependencia del rendimiento con respecto a la distribución de datos del algoritmo paralelo. El impacto de las factorizaciones *right-looking* (entre las que se puede incluir la factorización Cholesky), sobre el balance de carga en cómputo paralelo es fundamental, aún cuando los valores de referencia de capacidad de cómputo sean mucho menores a los reales.

El paso inmediato siguiente incluye la aplicación de los principios de cómputo de alto rendimiento a la factorización de matrices. También se debe adaptar el algoritmo paralelo, ahora incluyendo en la medida de lo posible las optimizaciones propias para la arquitectura paralela de un cluster homogéneo.