UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Sistemas Distribuidos

Escuela Superior de Informática

Algoritmo de Cannon

David Villa Alises
Francisco Moya Fernández

ISRN ESI/SSDD--2014/15-1

versión 2014.11.04

Índice

Este documento constituye la especificación del primer entregable para la evaluación del laboratorio de la asignatura de Sistemas Distribuidos. El alumno debe construir una aplicación distribuida conforme a la siguiente especificación. La realización es individual y la entrega obligatoria.

Este ejercicio pretende exponer al alumno un problema realista que requiere cómputo intensivo. Por medio de un grid y utilizando el middleware Ice, el alumno deberá implementar una solución completa al problema que se detalla.

El producto de matrices es una operación matemática muy útil en muchos ámbitos de las matemáticas, el álgebra lineal y la ingeniería. Pero también es una operación costosa en términos de cómputo, concretamente $O(n^3)$ y de requerimientos de memoria, hasta el punto de que se utiliza habitualmente como benchmark. En este ejercicio se explica el algoritmo de Cannon y se propone una posible implementación.

1. Producto de matrices

Sean las matrices A y B:

$$A^{(m,r)} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,r} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,r} \end{pmatrix} \qquad B^{(r,n)} = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{r,1} & b_{r,2} & \cdots & b_{r,n} \end{pmatrix}$$

se define el producto de matrices $C = A \times B$ como:

$$c_{i,j} = \sum_{k=1}^{r} a_{ik} \times b_{kj} \tag{1}$$

siendo C:

$$C^{(m,n)} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m,1} & c_{m,2} & \cdots & c_{m,n} \end{pmatrix}$$

Para poder efectuar el producto de matrices $A \times B$ debe cumplirse que el número de columnas de A sea igual al número de filas de B (r en la formulación anterior). Por tanto, la matriz resultado (C) tendrá un tamaño de $m \times n$.

2. El algoritmo de Cannon

Cada elemento de la matriz resultado está compuesto por el sumatorio de r productos independientes. El algoritmo de Cannon [E.69, Tin03] aprovecha esta circunstancia para definir una forma de paralelizar el cómputo.

Sin embargo, realizar cada producto individual en un nodo de cómputo no es rentable. El tiempo necesario para llevar los datos al nodo y recuperar el resultado es normalmente mayor que la ventaja que supone la paralelización. Esto es relevante cuando se ejecuta el algoritmo sobre un multiprocesador,

pero tiene un impacto mucho mayor cuando se implementa sobre un grid. Recuerde que la red de comunicaciones (incluso una LAN) implica normalmente una sobrecarga en el tiempo de llamada hasta dos órdenes de magnitud mayor que una llamada local (en el mismo nodo).

Para paliar este efecto, el algoritmo de Cannon aplica el producto sobre bloques (submatrices) tomados de las matrices operando. A continuación aparece un ejemplo de multiplicación por bloques para matrices de 6×6 con un tamaño de bloque de 2×2.

Sea A:

$$A = \begin{pmatrix} 1 & 2 & -1 & 3 & -5 & 3 \\ 3 & 0 & 2 & 7 & 1 & 2 \\ 5 & 3 & 3 & 1 & 4 & 4 \\ 2 & 1 & 2 & 2 & 3 & 3 \\ -2 & 2 & 6 & 6 & 9 & 7 \\ -2 & 2 & -4 & 2 & 6 & 5 \end{pmatrix} = \begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}$$

Se definen las submatrices:

$$A_{1,1} = \begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix} \qquad A_{1,2} = \begin{pmatrix} -1 & 3 \\ 2 & 7 \end{pmatrix} \qquad A_{1,3} = \begin{pmatrix} -5 & 3 \\ 1 & 2 \end{pmatrix}$$

$$A_{2,1} = \begin{pmatrix} 5 & 3 \\ 2 & 1 \end{pmatrix} \qquad A_{2,2} = \begin{pmatrix} 3 & 1 \\ 2 & 2 \end{pmatrix} \qquad A_{2,3} = \begin{pmatrix} 4 & 4 \\ 3 & 3 \end{pmatrix}$$

$$A_{3,1} = \begin{pmatrix} -2 & 2 \\ -2 & 2 \end{pmatrix} \qquad A_{3,2} = \begin{pmatrix} 6 & 6 \\ -4 & 2 \end{pmatrix} \qquad A_{3,3} = \begin{pmatrix} 9 & 7 \\ 6 & 5 \end{pmatrix}$$

Y sea B:

$$B = \begin{pmatrix} 1 & 2 & 7 & 7 & -1 & 3 \\ 2 & 1 & 2 & 1 & -1 & 2 \\ 3 & 0 & 5 & 5 & 6 & 1 \\ 3 & 1 & 4 & 1 & 6 & 4 \\ 2 & 4 & 3 & 3 & 3 & 9 \\ 1 & 4 & 3 & 2 & 1 & 9 \end{pmatrix} = \begin{pmatrix} B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix}$$

Se definen sus submatrices como:

$$B_{1,1} = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \qquad B_{1,2} = \begin{pmatrix} 7 & 7 \\ 2 & 1 \end{pmatrix} \qquad B_{1,3} = \begin{pmatrix} -1 & 3 \\ -1 & 2 \end{pmatrix}$$

$$B_{2,1} = \begin{pmatrix} 3 & 0 \\ 3 & 1 \end{pmatrix} \qquad B_{2,2} = \begin{pmatrix} 5 & 5 \\ 4 & 1 \end{pmatrix} \qquad B_{2,3} = \begin{pmatrix} 6 & 1 \\ 6 & 4 \end{pmatrix}$$

$$B_{3,1} = \begin{pmatrix} 2 & 4 \\ 1 & 4 \end{pmatrix} \qquad B_{3,2} = \begin{pmatrix} 3 & 3 \\ 3 & 2 \end{pmatrix} \qquad B_{3,3} = \begin{pmatrix} 3 & 9 \\ 1 & 9 \end{pmatrix}$$

El resultado C será una matriz formada también por bloques 2×2:

$$C = \begin{pmatrix} C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix} \tag{2}$$

Dichas submatrices se obtienen como resultado de multiplicar los bloques correspondientes tal como indica la fórmula 1 para elementos individuales. Por tanto:

$$C_{1,1} = A_{1,1} \times B_{1,1} + A_{1,2} \times B_{2,1} + A_{1,3} \times B_{3,1}$$

$$C_{1,2} = A_{1,1} \times B_{1,2} + A_{1,2} \times B_{2,2} + A_{1,3} \times B_{3,2}$$

$$C_{1,3} = A_{1,1} \times B_{1,3} + A_{1,2} \times B_{2,3} + A_{1,3} \times B_{3,3}$$

$$C_{2,1} = A_{2,1} \times B_{1,1} + A_{2,2} \times B_{2,1} + A_{2,3} \times B_{3,1}$$

$$C_{2,2} = A_{2,1} \times B_{1,2} + A_{2,2} \times B_{2,2} + A_{2,3} \times B_{3,2}$$

$$C_{2,3} = A_{2,1} \times B_{1,3} + A_{2,2} \times B_{2,3} + A_{2,3} \times B_{3,3}$$

$$C_{3,1} = A_{3,1} \times B_{1,1} + A_{3,2} \times B_{2,1} + A_{3,3} \times B_{3,1}$$

$$C_{3,2} = A_{3,1} \times B_{1,2} + A_{3,2} \times B_{2,2} + A_{3,3} \times B_{3,2}$$

$$C_{3,3} = A_{3,1} \times B_{1,3} + A_{3,2} \times B_{2,3} + A_{3,3} \times B_{3,3}$$

Esto supone p^2 sumas y p^3 productos, siendo p el resultado de dividir el orden de las matrices operando entre el orden de los bloques.

El algoritmo de Cannon propone la utilización de una matriz de procesadores (multiplicadores simples) de orden p (fórmula 3). Cada procesador P_{ij} realizará el cómputo de la suma de productos correspondiente a un bloque de la matriz resultado C (fórmula 2). Por tanto, el orden de los operandos deberá ser **divisible** por p.

$$P = \begin{pmatrix} P_{1,1} & P_{1,2} & P_{1,3} \\ P_{2,1} & P_{2,2} & P_{2,3} \\ P_{3,1} & P_{3,2} & P_{3,3} \end{pmatrix} \tag{3}$$

Por ejemplo, para operandos de orden 243 y una matriz de procesadores de orden 3 (9 procesadores), las submatrices tendrán un orden $\frac{243}{3} = 81$.

Copiar la matriz completa a cada nodo implicaría una importante sobrecarga, tanto para las comunicaciones como para la memoria de los procesadores. Cannon proporciona a cada procesador únicamente la submatriz que necesita en cada momento (cada etapa), reduciendo a $\frac{1}{p}$ los requisitos de memoria y ancho de banda del procesador. A su vez, cada procesador envía a sus procesadores vecinos las submatrices del paso anterior. Así se evita centralizar el reparto de los datos en un único punto y se mejora el balanceo de carga.

Un proceso externo debe realizar la carga inicial de las submatrices en todos procesadores. Para que los operandos obtenidos por cada procesador sean los adecuados es necesario desplazar las matrices originales. En la matriz A, la segunda fila se desplaza hacia la izquierda una columna, la tercera fila se desplaza dos columnas y así sucesivamente. En el caso de la matriz del ejemplo queda así:

$$A' = \begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} & \leftarrow \\ A_{2,2} & A_{2,3} & A_{2,1} & \leftarrow \\ A_{3,3} & A_{3,1} & A_{3,2} & \leftarrow \end{pmatrix} = \left(\begin{array}{cc|cc|cc} 1 & 2 & -1 & 3 & -5 & 3 \\ 3 & 0 & 2 & 7 & 1 & 2 \\ \hline 3 & 1 & 4 & 4 & 5 & 3 \\ 2 & 2 & 3 & 3 & 2 & 1 \\ \hline 9 & 7 & -2 & 2 & 6 & 6 \\ 6 & 5 & -2 & 2 & -4 & 2 \end{array}\right)$$

El operando B también debe ser desplazado, aunque en este caso verticalmente. La segunda columna de la matriz B se desplaza hacia arriba una fila, la tercera columna se desplaza dos filas y así sucesivamente. La matriz B del ejemplo queda así:

$$B' = \begin{pmatrix} B_{1,1} & B_{2,2} & B_{3,3} \\ B_{2,1} & B_{3,2} & B_{1,3} \\ B_{3,1} & B_{1,2} & B_{2,3} \\ & \uparrow & \uparrow\uparrow \end{pmatrix} = \left(\begin{array}{cc|cc|cc} 1 & 2 & 5 & 5 & 3 & 9 \\ 2 & 1 & 4 & 1 & 1 & 9 \\ \hline 3 & 0 & 3 & 3 & -1 & 3 \\ 3 & 1 & 3 & 2 & -1 & 2 \\ \hline 2 & 4 & 7 & 7 & 6 & 1 \\ 1 & 4 & 2 & 1 & 6 & 4 \end{array}\right)$$

Después se envían las submatrices a los procesadores, que realizarán los productos de dichos operandos:

$$\left(\begin{array}{ccc|ccc|ccc} A_{1,1} & \times & B_{1,1} & A_{1,2} & \times & B_{2,2} & A_{1,3} & \times & B_{3,3} \\ \hline A_{2,2} & \times & B_{2,1} & A_{2,3} & \times & B_{3,2} & A_{2,1} & \times & B_{1,3} \\ \hline A_{3,3} & \times & B_{3,1} & A_{3,1} & \times & B_{1,2} & A_{3,2} & \times & B_{2,3} \end{array}\right) \tag{4}$$

FÓRMULA 2.1: Operandos recibidos por cada procesador en la primera etapa (después del desplazamiento inicial)

Los procesadores almacenan localmente el resultado parcial $C_{i,j}$. A continuación envían la submatriz de A recibida al procesador de su izquierda y la submatriz de B al procesador superior, tal como aparece en la figura 1.

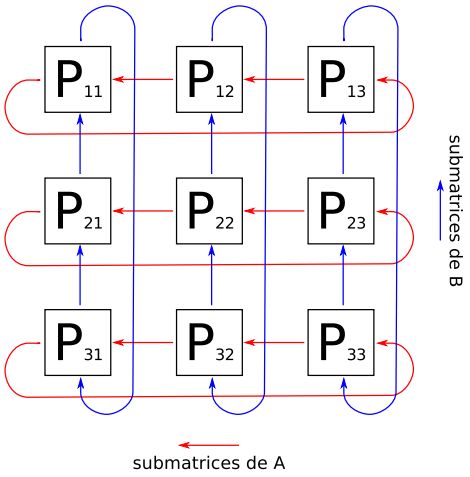

FIGURA 1: Matriz de procesadores

Del mismo modo, cada procesador recibirá los nuevos operandos de sus vecinos de abajo y derecha. Cuando disponga de ambos operandos empieza la segunda etapa, en la que realiza un nuevo producto con las submatrices recibidas. Después suma el resultado al valor local almacenado en el nodo (Ci, j) (realizado en la etapa anterior). Nótese que puede haber procesadores ejecutando etapas distintas. El único requisito para comenzar una nueva etapa es haber recibido los operandos correspondientes.

Los operandos que recibe cada procesador en la segunda y tercera etapa se muestran en las figuras 2.2 y 2.3

$$\left(\begin{array}{ccc|ccc|ccc} A_{1,2} & \times & B_{2,1} & A_{1,3} & \times & B_{3,2} & A_{1,1} & \times & B_{1,3} \\ \hline A_{2,3} & \times & B_{3,1} & A_{2,1} & \times & B_{1,2} & A_{2,2} & \times & B_{2,3} \\ \hline A_{3,1} & \times & B_{1,1} & A_{3,2} & \times & B_{2,2} & A_{3,3} & \times & B_{3,3} \end{array}\right) \tag{5}$$

FÓRMULA 2.2: Operandos recibidos por cada procesador en la segunda etapa (después de 1 rotación)

$$\left(\begin{array}{ccc|ccc|ccc} A_{1,3} & \times & B_{3,1} & A_{1,1} & \times & B_{1,2} & A_{1,2} & \times & B_{2,3} \\ \hline A_{2,1} & \times & B_{1,1} & A_{2,2} & \times & B_{2,2} & A_{2,3} & \times & B_{3,3} \\ \hline A_{3,2} & \times & B_{2,1} & A_{3,3} & \times & B_{3,2} & A_{3,1} & \times & B_{1,3} \end{array}\right) \tag{6}$$

FÓRMULA 2.3: Operandos recibidos por cada procesador en la tercera etapa (después de 2 rotaciones)

Tras p etapas (siendo $p = 3$ en el ejemplo) cada procesador contiene la submatriz final de la matriz resultado. La formulación 2.4 muestra el desarrollo completo con el orden en el que se realizan las multiplicaciones de las submatrices según el algoritmo descrito.

3. Implementación

Para implementar el algoritmo descrito en un grid, se requieren los siguientes componentes:

- Un *frontend*. Recibe los operandos (matrices A y B) desde el cliente, realiza la operación de desplazamiento inicial sobre ambas, calcula las submatrices y las envía a los procesadores.
- p procesadores de submatrices. Cada procesador recibe los operandos de sus vecinos previos (o del frontend), los multiplica, suma el resultado con cálculo previo y después envía las submatrices recibidas a sus vecinos posteriores.
- Un *collector*. Recoge los resultados parciales de cada procesador cuando el algoritmo ha terminado y conforma la matriz resultado.

3.1. Consideraciones

En la implementación deben tenerse en cuenta las siguientes particularidades:

- Cuando un procesador disponga de las dos submatrices debe lanzar la multiplicación, pero **sin bloquear** al cliente que las proporciona. Si hubiera un bloqueo del cliente se eliminarían las posibilidades de paralelización del algoritmo y la solución no sería correcta. Es decir, la recogida de datos y el cómputo de los mismos deben estar desacoplados.
- Un procesador puede recibir operandos de la siguiente etapa cuando la multiplicación de la etapa actual está en curso o incluso antes de recibir los operandos de la etapa en curso. Por tanto, en esta situación el procesador puede necesitar **almacenar temporalmente** los operandos de las siguientes etapas.

$$C_{1,1} = A_{1,1} \times B_{1,1} + A_{1,2} \times B_{2,1} + A_{1,3} \times B_{3,1} =$$
$$\begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} + \begin{pmatrix} -1 & 3 \\ 2 & 7 \end{pmatrix} \times \begin{pmatrix} 3 & 0 \\ 3 & 1 \end{pmatrix} + \begin{pmatrix} -5 & 3 \\ 1 & 2 \end{pmatrix} \times \begin{pmatrix} 2 & 4 \\ 1 & 4 \end{pmatrix} = \begin{pmatrix} 4 & 1 \\ 34 & 25 \end{pmatrix}$$

$$C_{1,2} = A_{1,2} \times B_{2,2} + A_{1,3} \times B_{3,2} + A_{1,1} \times B_{1,2} =$$
$$\begin{pmatrix} -1 & 3 \\ 2 & 7 \end{pmatrix} \times \begin{pmatrix} 5 & 5 \\ 4 & 1 \end{pmatrix} + \begin{pmatrix} -5 & 3 \\ 1 & 2 \end{pmatrix} \times \begin{pmatrix} 3 & 2 \\ 3 & 2 \end{pmatrix} + \begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix} \times \begin{pmatrix} 7 & 7 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 12 & 2 \\ 68 & 45 \end{pmatrix}$$

$$C_{1,3} = A_{1,3} \times B_{3,3} + A_{1,1} \times B_{1,3} + A_{1,2} \times B_{2,3} =$$
$$\begin{pmatrix} -5 & 3 \\ 1 & 2 \end{pmatrix} \times \begin{pmatrix} 3 & 9 \\ 1 & 9 \end{pmatrix} + \begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix} \times \begin{pmatrix} -1 & 3 \\ -1 & 2 \end{pmatrix} + \begin{pmatrix} -1 & 3 \\ 2 & 7 \end{pmatrix} \times \begin{pmatrix} 6 & 1 \\ 6 & 4 \end{pmatrix} = \begin{pmatrix} -3 & 0 \\ 56 & 66 \end{pmatrix}$$

$$C_{2,1} = A_{2,2} \times B_{2,1} + A_{2,3} \times B_{3,1} + A_{2,1} \times B_{1,1} =$$
$$\begin{pmatrix} 3 & 1 \\ 2 & 2 \end{pmatrix} \times \begin{pmatrix} 3 & 0 \\ 3 & 1 \end{pmatrix} + \begin{pmatrix} 4 & 4 \\ 3 & 3 \end{pmatrix} \times \begin{pmatrix} 2 & 4 \\ 1 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 3 \\ 2 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 35 & 46 \\ 25 & 31 \end{pmatrix}$$

$$C_{2,2} = A_{2,3} \times B_{3,2} + A_{2,1} \times B_{1,2} + A_{2,2} \times B_{2,2} =$$
$$\begin{pmatrix} 4 & 4 \\ 3 & 3 \end{pmatrix} \times \begin{pmatrix} 3 & 3 \\ 3 & 2 \end{pmatrix} + \begin{pmatrix} 5 & 3 \\ 2 & 1 \end{pmatrix} \times \begin{pmatrix} 7 & 7 \\ 2 & 1 \end{pmatrix} + \begin{pmatrix} 3 & 1 \\ 2 & 2 \end{pmatrix} \times \begin{pmatrix} 5 & 5 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 84 & 74 \\ 52 & 42 \end{pmatrix}$$

$$C_{2,3} = A_{2,1} \times B_{1,3} + A_{2,2} \times B_{2,3} + A_{2,3} \times B_{3,3} =$$
$$\begin{pmatrix} 5 & 3 \\ 2 & 1 \end{pmatrix} \times \begin{pmatrix} -1 & 3 \\ -1 & 2 \end{pmatrix} + \begin{pmatrix} 3 & 1 \\ 2 & 2 \end{pmatrix} \times \begin{pmatrix} 6 & 1 \\ 6 & 4 \end{pmatrix} + \begin{pmatrix} 4 & 4 \\ 3 & 3 \end{pmatrix} \times \begin{pmatrix} 3 & 9 \\ 1 & 9 \end{pmatrix} = \begin{pmatrix} 32 & 100 \\ 33 & 72 \end{pmatrix}$$

$$C_{3,1} = A_{3,3} \times B_{3,1} + A_{3,1} \times B_{1,1} + A_{3,2} \times B_{2,1} =$$
$$\begin{pmatrix} 9 & 7 \\ 6 & 5 \end{pmatrix} \times \begin{pmatrix} 2 & 4 \\ 1 & 4 \end{pmatrix} + \begin{pmatrix} -2 & 2 \\ -2 & 2 \end{pmatrix} \times \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} + \begin{pmatrix} 6 & 6 \\ -4 & 2 \end{pmatrix} \times \begin{pmatrix} 3 & 0 \\ 3 & 1 \end{pmatrix} = \begin{pmatrix} 63 & 68 \\ 13 & 44 \end{pmatrix}$$

$$C_{3,2} = A_{3,1} \times B_{1,2} + A_{3,2} \times B_{2,2} + A_{3,3} \times B_{3,2} =$$
$$\begin{pmatrix} -2 & 2 \\ -2 & 2 \end{pmatrix} \times \begin{pmatrix} 7 & 7 \\ 2 & 1 \end{pmatrix} + \begin{pmatrix} 6 & 6 \\ -4 & 2 \end{pmatrix} \times \begin{pmatrix} 5 & 5 \\ 4 & 1 \end{pmatrix} + \begin{pmatrix} 9 & 7 \\ 6 & 5 \end{pmatrix} \times \begin{pmatrix} 3 & 3 \\ 3 & 2 \end{pmatrix} = \begin{pmatrix} 92 & 65 \\ 11 & -2 \end{pmatrix}$$

$$C_{3,3} = A_{3,2} \times B_{2,3} + A_{3,3} \times B_{3,3} + A_{3,1} \times B_{1,3} =$$
$$\begin{pmatrix} 6 & 6 \\ -4 & 2 \end{pmatrix} \times \begin{pmatrix} 6 & 1 \\ 6 & 4 \end{pmatrix} + \begin{pmatrix} 9 & 7 \\ 6 & 5 \end{pmatrix} \times \begin{pmatrix} 3 & 9 \\ 1 & 9 \end{pmatrix} + \begin{pmatrix} -2 & 2 \\ -2 & 2 \end{pmatrix} \times \begin{pmatrix} -1 & 3 \\ -1 & 2 \end{pmatrix} = \begin{pmatrix} 106 & 172 \\ 11 & 101 \end{pmatrix}$$

FÓRMULA 2.4: Cálculo de las submatrices resultado en el mismo orden que lo realizan los procesadores

- Como suele ser habitual, se trata de una aplicación distribuida asíncrona, es decir, no puede haber ningún elemento que determine el momento en el que deben ocurrir las operaciones. Los procesadores realizan su función cuando disponen de los operandos adecuados.

3.2. Definición de interfaces

Las matrices son estructuras que contienen los datos de la matriz, el número de columnas y un identificador universal opcional. El número de filas se puede obtener simplemente dividiendo el tamaño de la secuencia entre el número de columnas. Se muestra a continuación la declaración en lenguaje Slice:

```
1  struct Matrix {
2    int ncols;
3    Ice::DoubleSeq data;
4    string UUID;
5  };
```

Los elementos de la matriz se almacenan en una secuencia plana de flotantes. Los primeros $ncols$ elementos correspondientes a la primera fila, los siguientes a la segunda fila y así sucesivamente.

La interfaz del procesador (multiplicador) es la siguiente:

```
1  interface Processor {
2    void init(int index, int order,
3              Processor* above, Processor* left, Collector* target);
4    void injectA(Matrix a, int step);
5    void injectB(Matrix b, int step);
6  };
```

El método `init()` prepara el procesador para una nueva multiplicación, el parámetro `index` le indica al procesador su posición en la matriz de procesadores (siendo 0 el primero). El parámetro `order` es el orden de la matriz de procesadores, que corresponde con el número de etapas y determina cuando ha terminado el algoritmo. Los parámetros `above` y `left` son las referencias a sus vecinos superior e izquierdo. Esto es necesario puesto que cada procesador es responsable de propagar a sus vecinos las submatrices que va recibiendo en cada etapa. El parámetro `target` es el *recolector*, al que el procesador debe enviar el resultado cuando haya terminado.

Los métodos `injectA()` e `injectB()` le dan al procesador las matrices operando. Son invocados por sus vecinos derecho e inferior respectivamente, o bien por el frontend al comienzo del algoritmo. El parámetro `step` le indica a qué etapa corresponde el operando. Las etapas se numeran desde 0 hasta $p - 1$.

La interfaz del *recolector* es sencilla:

```
1  interface Collector {
2    void inject(int index, Matrix m);
3  };
```

El *recolector* conforma la matriz final a partir de las submatrices que va recibiendo. Por último, la interfaz pública del servicio, la que será invocada por los usuarios:

```
1  interface Frontend {
2    Matrix multiply(Matrix a, Matrix b);
3  };
```

Esta interfaz debe ser implementada por el servidor y define el modo en el que recibe los operandos. Su sirviente debe realizar las funciones, ya indicadas, para desplazar las matrices, dividirlas en submatrices,

enviarlas a los procesadores y esperar el resultado creando un objeto *Collector*. Éste debe recomponer la matriz resultado cuando disponga de todos los fragmentos y por último devolver el resultado como valor de retorno del método `matrixMultiply()`.

La figura 2 muestra un diagrama de secuencia simplificado.

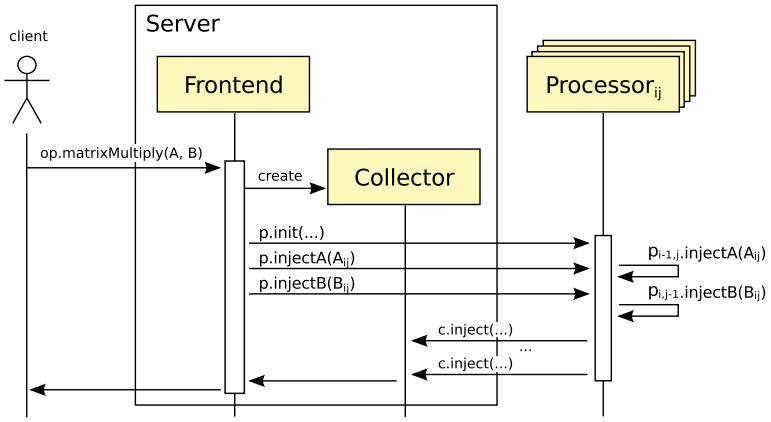

FIGURA 2: Diagrama de secuencia de una ejecución del algoritmo

3.3. Arquitectura

La implementación del algoritmo pasa por disponer de varios nodos de cómputo y el despliegue de cierta cantidad de procesadores repartidos en dichos nodos. En nuestro caso el objetivo final es desplegar 25 procesadores en 3 nodos. Además, el programa que ofrece el servicio también estará desplegado en uno de los nodos. La figura 3 muestra una posible distribución:

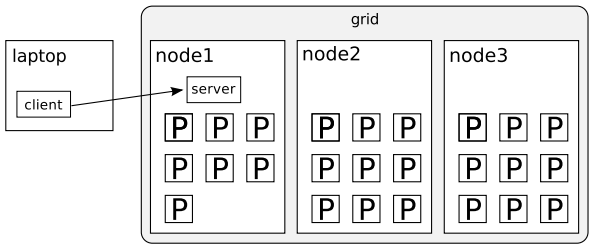

FIGURA 3: Propuesta de despliegue de los distintos elementos en un grid de 3 nodos

El frontend necesita averiguar cuáles son los procesadores disponibles (sus *proxies*) y debe asignarles su posición en la matriz de procesadores (por medio del método `init()`). El alumno debe implementar un mecanismo para hacer posible dicha tarea. Se recomienda utilizar un servicio sencillo que almacene los proxies de los procesadores.

3.4. Estrategia de implementación

A continuación se propone al alumno una estrategia de implementación incremental para facilitar el desarrollo de la práctica:

1. Implementación de una función local de multiplicación de matrices de flotantes de orden cualquiera.
2. Implementación de un *Collector* sencillo. Una primera implementación (*fake*) puede simplemente imprimir en consola las matrices que recibe. Nos servirá para probar el procesador.

3. Implementación del procesador. El procesador es un objeto independiente que puede ser desarrollado y probado de forma autónoma. Es posible invocar un procesador aislado pasando como argumentos las matrices completas e indicando `orden=1`. En ese caso, el procesador debe utilizar la función creada en el punto 1, y enviar el resultado al *Collector* creado en el punto 2.
   En este punto se deben realizar una serie de pruebas unitarias automáticas para verificar el correcto funcionamiento del procesador con matrices de orden diferente. Se recomienda substituir el *Collector* por un doble de prueba para escribir dichas pruebas.
4. Crear varios procesadores (4 es el mínimo) e implementar pruebas automáticas para comprobar que la *rotación* de submatrices es correcta. En este punto las submatrices iniciales pueden estar calculadas a mano.
5. Implementar una versión inicial del frontend (interfaz *Frontend*) que realice el desplazamiento y troceado de las matrices operando. Debe verificar mediante pruebas automáticas que es correcto utilizando dobles de prueba para los procesadores a fin de aislar esta tarea.
6. Conectar el frontend con una matriz de procesadores pequeña (orden 4) y comprobar que la operación se realiza correctamente. De nuevo, se debe usar un doble de prueba para el *Collector*.
7. Implementar un *Collector* más realista capaz de ensamblar la matriz final y devolverla al usuario. Es importante destacar que el cliente invocó el método `Frontend::matrixMultiply()` y, sin embargo, el resultado se obtiene a partir de las submatrices recibidas a través de `Collector::injectSubmatrix()`. Debe implementar un mecanismo que mantenga al cliente bloqueado hasta el momento en que el resultado está disponible. El servidor está recibiendo invocaciones concurrentes a ambos métodos durante la ejecución.
8. Definir una aplicación distribuida con IceGrid con una distribución similar a la de la figura 3. El objeto *Cannon::frontend* debería ser un objeto *bien conocido* con identidad (“Cannon”).

4. Evaluación y condiciones

La elaboración de la aplicación se compone de tres entregables. Todos ellos se realizarán de forma individual, se entregarán mediante una tarea moodle y serán defendidos por el alumno en una sesión de laboratorio.

1. Implementación del servicio *Processor* capaz de multiplicar matrices cuadradas de cualquier orden en un solo paso (ver sección 3.4, punto 3). Incluye una versión simplificada del *Collector* que permita comprobar que el resultado del procesador es correcto.
   El plazo límite de entrega y la defensa será la sesión 4 de laboratorio (4 u 11 de noviembre).
2. Implementación de un servicio *Frontend* básico capaz de desplazar y trocear los operandos, y enviarlos a una matriz de 4 procesadores (ver sección 3.4, punto 6).
   El plazo límite de entrega y la defensa será la sesión 5 de laboratorio (18 y 25 de noviembre).
3. Implementación del servicio integrado. Debe incluir una versión funcional de todos los elementos: Frontend, Collector y Processor, y superar las pruebas automáticas correspondientes. Mediante el cliente que se facilita debe poderse lanzar una multiplicación de dos matrices de al menos orden 400, con una matriz de procesadores de orden 5 (ver sección 3.4, punto 7).
   El plazo límite de entrega y la defensa será la sesión 6 de laboratorio (2 y 9 de diciembre).

En todos los casos las entregas se realizarán por medio de un archivo `.zip` o `.tgz` que incluya todos los ficheros necesarios para la compilación de los distintos componentes, especificación de la aplicación (fichero XML), y otros scripts y ficheros `Makefile` necesarios para el despliegue, ejecución y prueba de la aplicación.

La evaluación se realizará en un sistema operativo GNU/Linux, de modo que el alumno debe comprobar que funciona correctamente en dicho entorno. El alumno deberá realizar una defensa presencial e individual. La ejecución se efectuará en el computador del alumno.

La calificación obtenida por el alumno dependerá del nivel de complejidad que alcance dentro de los siguientes:

4.1. Nivel básico

Corresponde con la realización satisfactoria de los tres entregables indicados en la sección anterior. Se aplican las siguientes simplificaciones:

- Los operandos son siempre matrices cuadradas con orden múltiplo de 5.
- Todos los elementos de la aplicación se ejecutarán en el portátil del alumno. No se requiere el uso del IceGrid.

Este nivel dará lugar a la obtención de la calificación mínima para superar la asignatura[1], con una puntuación en el rango [8–13) sobre los 25 puntos de la actividad «Realización de prácticas de laboratorio». Además podrá obtener hasta 5 puntos adicionales en la actividad «Presentación oral de temas» por la defensa presencial (únicamente en la convocatoria ordinaria). Los alumnos que opten por este nivel deberán hacer su defensa en la última sesión de laboratorio.

4.2. Nivel medio

Para lograr este nivel, además de la realización correcta de los entregables, se aplican los siguientes condicionantes:

- Debe crear una aplicación distribuida gestionada con IceGrid con un despliegue similar al de la figura 3.
- Para ejecutar los tres nodos, el alumno puede usar máquinas virtuales que se ejecutan en su propio portátil.
- Los operandos son siempre matrices cuadradas con orden múltiplo de 2, 3, 4 o 5.

Se valorará:

- Cualquier mecanismo que automatice las tareas de compilación, despliegue, ejecución y prueba. Se recomienda especialmente el uso de `icegridadmin` (evitando `icegrid-gui`) para lograrlo. La necesidad de ejecución manual de comandos o acciones superfluas de cualquier tipo penaliza la puntuación.
- Utilización de **nombres significativos** para clases, tipos, métodos y variables. La presencia de comentarios evidencia código poco expresivo. Elimina los comentarios y elige buenos nombres para tus abstracciones.
- Implementación de plantillas para la creación de los servidores en la aplicación IceGrid.
- Implementación de pruebas automáticas unitarias y de integración.
- Cualquier esfuerzo por mejorar el rendimiento de la aplicación, por ejemplo, utilizando AMI/AMD para mejorar la paralelización de las invocaciones.

Este nivel dará lugar a la obtención de una calificación de notable, con una puntuación en el rango [13–18) sobre los 25 puntos de la actividad «Realización de prácticas de laboratorio». Además podrá obtener hasta 7 puntos adicionales en la actividad «Presentación oral de temas» por la defensa presencial (únicamente en la convocatoria ordinaria). Los alumnos que opten por este nivel harán su defensa en una fecha que se anunciará convenientemente una vez acabado el período de clases.

[1] teniendo en consideración los 2 puntos adicionales correspondientes a la defensa del ejercicio de la sesión 3

4.3. Nivel avanzado

En este nivel, además de los requisitos y criterios de evaluación del nivel medio, se aplican los siguientes:

- El alumno debe poder crear un grid de 9 nodos utilizando 3 computadores, es decir, el suyo y los de otros dos compañeros. Por tanto, este nivel requiere formar grupos de tres alumnos, pero con el único objetivo de compartir la infraestructura. La elaboración sigue siendo individual, y cada uno de los miembros del grupo tendrá que defender *su propia* aplicación.
- Los operandos pueden ser matrices con un orden superior a 1000.

Se valorará:

- Creación de los procesadores mediante un objeto factoría, en lugar de servidores en la aplicación IceGrid.
- Uso de más de un lenguaje de programación.
- Matrices no cuadradas y de distinto orden.

Este nivel dará lugar a la obtención de una calificación de sobresaliente, con una puntuación en el rango [19–23] correspondientes a la actividad «Realización de prácticas de laboratorio». Además podrá obtener hasta 10 puntos adicionales en la actividad «Presentación oral de temas» por la defensa presencial (únicamente en la convocatoria ordinaria). Los alumnos que opten por este nivel harán su defensa en una fecha que se anunciará convenientemente una vez acabado el período de clases.

Referencias

[E.69] Cannon L. E. *A Cellular Computer to Implement the Kalman Filter Algorithm*. PhD thesis, Montana State University, Bozman, Montana, 1969.

[Tin03] G. Tinetti. *Cómputo Paralelo en Redes Locales de Computadoras*. PhD thesis, Universitat Autònoma de Barcelona, Dic 2003.