UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Computación en la Nube y de Altas Prestaciones

Práctica 3: El algoritmo SUMMA para el producto matricial distribuido.

Índice

1 Introducción

El objetivo de esta práctica es el de realizar el producto de matrices siguiente:

$$C = A \times B \,, \tag{1}$$

donde $A, B, C \in \mathbb{R}^{n \times n}$. Por simplicidad seguiremos asumiendo matrices cuadradas.

Para realizar el producto de matrices expuesto vamos a utilizar "paralelismo de datos", es decir, particionaremos y distribuiremos los datos entre los procesos de manera que cada proceso se encargará de realizar los cálculos sobre los datos que almacena.

Supóngase el siguiente particionado de la matriz C:

$$C = \begin{pmatrix} C_{00} & C_{01} & \dots & C_{0(N-1)} \\ C_{10} & C_{11} & \dots & C_{0(N-1)} \\ \vdots & \vdots & & \vdots \\ C_{(N-1)0} & C_{(N-1)1} & \dots & C_{(N-1)(N-1)} \end{pmatrix} , \tag{2}$$

siendo $N = n/b$, donde b es el tamaño de bloque, es decir, N es el número de bloques. Asumiremos también por simplicidad que n es un múltiplo de b. De esta manera, todos los bloques son cuadrados de orden b, esto es, $C_{ij} \in \mathbb{R}^{b \times b}$, $\forall i, j = 0, \dots, N-1$. Supongamos asimismo que el particionado de las matrices A y B es el mismo.

Existen diversas formas de realizar el producto de matrices (1). Nosotros vamos a utilizar la mostrada a continuación dado que nos ayudará a realizar una paralelización eficiente en memoria distribuida. Dicha forma es conocida como el algoritmo SUMMA por ser el acrónimo de *Scalable Universal Matrix Multiplication*

Algorithm. Para entenderla utilizamos un ejemplo en el que $N = 4$:

$$\begin{pmatrix} C_{00} & C_{01} & C_{02} & C_{03} \\ C_{10} & C_{11} & C_{12} & C_{13} \\ C_{20} & C_{21} & C_{22} & C_{23} \\ C_{30} & C_{31} & C_{32} & C_{33} \end{pmatrix} = \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix} \times \begin{pmatrix} B_{00} & B_{01} & B_{02} & B_{03} \\ B_{10} & B_{11} & B_{12} & B_{13} \\ B_{20} & B_{21} & B_{22} & B_{23} \\ B_{30} & B_{31} & B_{32} & B_{33} \end{pmatrix} =$$

$$\begin{pmatrix} A_{00} \\ A_{10} \\ A_{20} \\ A_{30} \end{pmatrix} \times \begin{pmatrix} B_{00} & B_{01} & B_{02} & B_{03} \end{pmatrix} + \begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \\ A_{31} \end{pmatrix} \times \begin{pmatrix} B_{10} & B_{11} & B_{12} & B_{13} \end{pmatrix} +$$

$$\begin{pmatrix} A_{02} \\ A_{12} \\ A_{22} \\ A_{32} \end{pmatrix} \times \begin{pmatrix} B_{20} & B_{21} & B_{22} & B_{23} \end{pmatrix} + \begin{pmatrix} A_{03} \\ A_{13} \\ A_{23} \\ A_{33} \end{pmatrix} \times \begin{pmatrix} B_{30} & B_{31} & B_{32} & B_{33} \end{pmatrix} =$$

$$\begin{pmatrix} A_{00} \cdot B_{00} & A_{00} \cdot B_{01} & A_{00} \cdot B_{02} & A_{00} \cdot B_{03} \\ A_{10} \cdot B_{00} & A_{10} \cdot B_{01} & A_{10} \cdot B_{02} & A_{10} \cdot B_{03} \\ A_{20} \cdot B_{00} & A_{20} \cdot B_{01} & A_{20} \cdot B_{02} & A_{20} \cdot B_{03} \\ A_{30} \cdot B_{00} & A_{30} \cdot B_{01} & A_{30} \cdot B_{02} & A_{30} \cdot B_{03} \end{pmatrix} + \begin{pmatrix} A_{01} \cdot B_{10} & A_{01} \cdot B_{11} & A_{01} \cdot B_{12} & A_{01} \cdot B_{13} \\ A_{11} \cdot B_{10} & A_{11} \cdot B_{11} & A_{11} \cdot B_{12} & A_{11} \cdot B_{13} \\ A_{21} \cdot B_{10} & A_{21} \cdot B_{11} & A_{21} \cdot B_{12} & A_{21} \cdot B_{13} \\ A_{31} \cdot B_{10} & A_{31} \cdot B_{11} & A_{31} \cdot B_{12} & A_{31} \cdot B_{13} \end{pmatrix} +$$

$$\begin{pmatrix} A_{02} \cdot B_{20} & A_{02} \cdot B_{21} & A_{02} \cdot B_{22} & A_{02} \cdot B_{23} \\ A_{12} \cdot B_{20} & A_{12} \cdot B_{21} & A_{12} \cdot B_{22} & A_{12} \cdot B_{23} \\ A_{22} \cdot B_{20} & A_{22} \cdot B_{21} & A_{22} \cdot B_{22} & A_{22} \cdot B_{23} \\ A_{32} \cdot B_{20} & A_{32} \cdot B_{21} & A_{32} \cdot B_{22} & A_{32} \cdot B_{23} \end{pmatrix} + \begin{pmatrix} A_{03} \cdot B_{30} & A_{03} \cdot B_{31} & A_{03} \cdot B_{32} & A_{03} \cdot B_{33} \\ A_{13} \cdot B_{30} & A_{13} \cdot B_{31} & A_{13} \cdot B_{32} & A_{13} \cdot B_{33} \\ A_{23} \cdot B_{30} & A_{23} \cdot B_{31} & A_{23} \cdot B_{32} & A_{23} \cdot B_{33} \\ A_{33} \cdot B_{30} & A_{33} \cdot B_{31} & A_{33} \cdot B_{32} & A_{33} \cdot B_{33} \end{pmatrix} .$$

Es fácil observar que cada término es un producto de una columna de A por la correspondiente fila de B. Es lo que se conoce como suma de *productos exteriores* (o *outer product*). Un producto exterior es el resultado de multiplicar un vector columna por un vector fila y ese resultado es una matriz. La suma de estas matrices da lugar a la matriz final, C en este caso.

En una primera aproximación a la paralelización de este algoritmo, haríamos una asignación uno a uno entre bloque y proceso de manera que el proceso $P_{i,j}$ calculase el bloque C_{ij}. Este bloque se calcula como:

$$C_{ij} = \sum_{k=0}^{N-1} A_{ik} \times B_{kj} \ . \tag{3}$$

Por ejemplo, el proceso $P_{2,1}$ realizaría el siguiente cálculo:

$$C_{21} = A_{20} \times B_{01} + A_{21} \times B_{11} + A_{22} \times B_{21} + A_{23} \times B_{31} \ .$$

Para comenzar el algoritmo se particionan y reparten los datos. Los datos son las matrices A y B, que están particionadas al igual de C (2) y se repartirán de la misma manera, es decir, el proceso $P_{i,j}$ almacenará los bloques A_{ij} y B_{ij}. El procedimiento paralelo no resulta difícil de entender. Se trata de un procedimiento iterativo de N iteraciones. En cada una se calcula un término de la suma (3). Sin embargo, en cada iteración, solo uno de los procesos tiene todos los datos para realizar el cálculo, el resto necesita datos que están en los otros procesos.

Veamos un ejemplo. Para calcular el primer término de la suma en el ejercicio anterior:

$$\begin{pmatrix} A_{00} \\ A_{10} \\ A_{20} \\ A_{30} \end{pmatrix} \times \begin{pmatrix} B_{00} & B_{01} & B_{02} & B_{03} \end{pmatrix} ,$$

solo el proceso $P_{0,0}$ tiene los datos necesarios para calcular su término correspondiente:

$$C_{00} \Leftarrow C_{00} + A_{00} \times B_{01} \ .$$

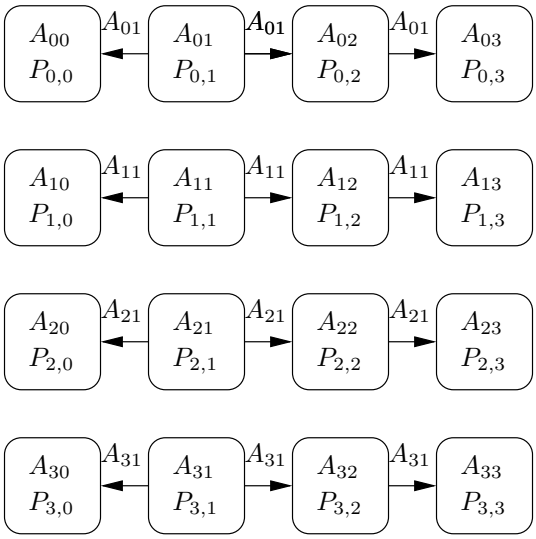

Figura 1: Difusión de datos de la columna en la segunda iteración del algoritmo.

Cada uno de los restantes 15 procesos necesitan datos que no tienen. Por ejemplo, el proceso $P_{3,1}$, realiza el cálculo

$$C_{31} \Leftarrow C_{31} + A_{30} \times B_{01} \ .$$

y necesita los bloques A_{30} y B_{01}. Estos bloques los tienen los procesos $P_{3,0}$ y $P_{0,1}$, respectivamente. Haciendo un análisis conveniente de los datos que necesita cada proceso veríamos que, para calcular todos los bloques del primer término, los procesos de la primera columna $P_{i,0}$ tendrían que difundir su bloque respectivo A_{i0} al resto de procesos de su fila i, mientras que los procesos de la primera fila $P_{0,j}$ tendrían que difundir su bloque respectivo B_{0j} al resto de procesos de su columna j. De esta manera, cada proceso $P_{i,j}$ recibe dos bloques, el bloque A_{i0} del proceso $P_{i,0}$ (de su misma fila) y el bloque B_{0j} del proceso $P_{0,j}$ (de su misma columna).

La Figura 1 trata de explicar este procedimiento según el ejemplo de 4×4 bloques para el caso de la segunda iteración donde, en primer lugar, se difunden los bloques de la matriz A de la segunda columna al resto de procesos de la fila correspondiente. En segundo lugar, los bloques de la matriz B correspondientes a la segunda fila se difunden al resto de procesos de la columna correspondiente (Figura 2). Una vez realizadas estas dos difusiones, todos los procesos tiene los datos necesarios para realizar el cálculo correspondiente a esa iteración, que sería

$$C_{ij} \Leftarrow C_{ij} + A_{i1} \times B_{1j} \ ,$$

para el proceso $P_{i,j}$.

2 Implementación

La implementación de este algoritmo requiere la utilización de herramientas potentes de MPI particulares para el tratamiento de problemas paralelos con distribución de datos regular, como es el caso. Entre estas herramientas cabe destacar las siguientes:

1. Utilización de tipos de datos MPI mediante funciones como `MPI_Type_vector`, `MPI_Type_commit`, etc.
2. Utilización de topología cartesiana, para lo cuál es conveniente manejar funciones del tipo `MPI_Cart_*`.
3. Manejo de comunicadores. Mediante la gestión de comunicadores es posible particionar los procesos en subconjuntos disjuntos con objeto de realizar comunicaciones de manera más sencilla y concurrente entre comunicadores. Cuando se trata de topologías cartesianas, entonces se puede utilizar `MPI_Cart_sub`.

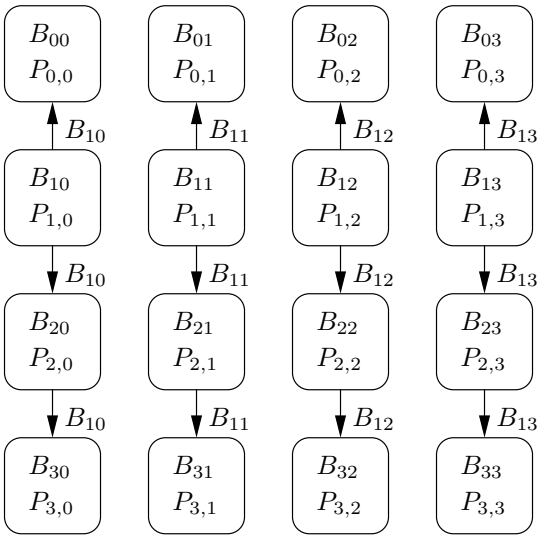

Figura 2: Difusión de datos de la fila en la segunda iteración del algoritmo.

4. Funciones típicas de comunicación entre procesos, especialmente colectivas.

Uno de los problemas con el que nos enfrentamos es la distribución de datos inicial. Inicialmente, solo el proceso 0 tiene las matrices A y B. Este proceso debe distribuir los bloques entre los procesos organizados ya en una malla 2D. Existen diversas formas de hacerlo. La que se sugiere aquí consiste en dos pasos:

1. El proceso 0 ($P_{0,0}$ en la malla cartesiana) distribuye bloques de columnas a cada proceso de la primera fila de procesos, es decir, a los procesos $P_{0,i}$, $i = 0, \ldots, N-1$, siendo N la dimensión de la malla. Por tanto, cada proceso recibirá $b = n/N$ columnas de la matriz A (o B). Para realizar este paso se pueden utilizar tipos de datos, aunque aquí no sería estrictamente necesario.

2. Cada proceso de la primera fila de procesos distribuye los bloques cuadrados de la columna de bloques que acaba de recibir de $P_{0,0}$ entre los procesos de su misma columna. En este caso sí es muy conveniente crear un tipo de dato derivado para acceder a los bloques cuadrados en la memoria del $P_{0,0}$ dado que las matrices están almacenadas por columnas.

El resto de la implementación consiste en aplicar el algoritmo explicado anteriormente, que consiste en un proceso iterativo de N pasos, donde cada iteración consiste en dos difusiones, una la para la A y otra para la B, y el producto matricial de los bloques.

3 Tarea a realizar

La tarea a realizar consiste en implementar el algoritmo paralelo en memoria distribuida utilizando las herramientas de MPI adecuadas para ello, es decir, las mencionadas en el apartado anterior. Una vez implementado y funcionando se ha de probar en el cluster `kahan`. El objetivo es comprobar la mejora que puede obtenerse con respecto al producto matricial en secuencial y con respecto al algoritmo paralelo que se obtuvo en la práctica anterior. Las multiplicaciones en secuencial se realizarán de igual manera que en la práctica anterior, utilizando con OpenMP los cores de los que dispone el proceso MPI, es decir, si hay un proceso MPI en el nodo, se utilizarán 32 hilos, si hay 4 procesos MPI, cada proceso utilizará 8 hilos.

Dado que las mallas que vamos a utilizar son necesariamente cuadradas, probaremos estas dos configuraciones:

- Una malla de 2×2 procesos MPI.
- Una malla de 4×4 procesos MPI.

En el segundo caso tendremos, obviamente, más procesos que nodos, pero al menos tendremos el mismo número de procesos en cada nodo. La idea es llevar a cabo un conjunto de ejecuciones variando el tamaño del problema. Estas ejecuciones se realizarán en un nodo y en varios utilizando las dos mallas cartesianas sugeridas. Recordad que tenemos restricciones: las matrices han de ser cuadradas y el tamaño de la matriz ha de ser múltiplo del tamaño de bloque y del número de procesos en cada dimensión.

Esta alternativa para realizar el producto de matrices en memoria distribuida no sustituye a la de la práctica anterior. Se supone que es un paso más en la optimización del producto matricial pero conservando los pasos anteriores para poder comparar. En la memoria final del proyecto se pedirá la comparación en cifras de las diferentes versiones.