Universidad de Alcalá

DEPARTAMENTO DE AUTOMÁTICA
ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

Grado en Ingeniería de Computadores
COMPUTACIÓN DE ALTAS PRESTACIONES

Práctica 3

Multiplicación de matrices mediante el paradigma de memoria distribuida

OBJETIVO

En muchos problemas de la ingeniería, especialmente en el ámbito de la simulación, aparece la necesidad de resolver sistemas de ecuaciones lineales. Esto, en sí mismo, no tendría ninguna dificultad si no fuera porque, en general, el número de incógnitas es muy alto, a veces del orden de cientos de miles o millones.

Por ello, se han desarrollado unas técnicas que permiten la solución de tales sistemas distribuyendo el cómputo sobre sistemas multicomputadores, es decir, sistemas con capacidad de cómputo y almacenamiento propios, e interconectados mediante algún mecanismo, típicamente una red.

Uno de los cómputos más típicos es el producto de dos matrices. En esta práctica se trata de implementar un conjunto de algoritmos que, basados el paradigma de memoria distribuida, alcanzan un creciente rendimiento. En concreto, se propone implementar un algoritmo *simple*, para después continuar con el algoritmo de Fox y, por último, el algoritmo de Cannon.

En lo que sigue, suponemos una distribución de datos por bloques bidimensionales en que cada proceso alberga el bloque que le corresponde. Si nombramos los procesos como $P_{i,j}$, donde $0 \leq i < q$, $0 \leq j < q$, tendremos un total de $p = q^2$ procesos. Cada $P_{i,j}$ es "dueño", es decir, alberga los bloques $A_{i,j}$, $B_{i,j}$ y es responsable de calcular el resultado parcial $C_{i,j}$. Los bloques son, sencillamente, matrices de tamaño $(n/q) \times (n/q)$.

Algoritmo secuencial

Comencemos recordando cómo se multiplican dos matrices densas de modo secuencial. Llamamos matrices densas a aquellas en que el número de elementos distintos de cero es la inmensa mayoría. Supongamos las matrices cuadradas A y B de tamaños $n \times n$ y queremos realizar la operación $C = A \times B$. El código secuencial puede ser el siguiente:

```
int MatMat(double **A, double **B, double **C, int n)
{
int i, j, k;

   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
      {
         C[i][j] = 0.0;
         for (k = 0; k < n; k++)
            C[i][j] += A[i][k]*B[k][j];
      }
}
```

Obviamente, para realizar toda la multiplicación hay que recorrer todos los lazos. Por lo tanto, con este algoritmo el tiempo de ejecución es $\Theta(n^3)$.

Algoritmo simple

En este algoritmo, muy sencillo, hacemos que cada proceso sea responsable de calcular cada bloque $C_{i,j}$ de la matriz resultado. Para poder hacerlo, recordemos que ha de realizar esta operación:

$$C_{i,j} = \sum_{k=0}^{q} A_{i,k} \times B_{k,j}.$$

Por consiguiente, el proceso $P_{i,j}$ necesita los bloques $A_{i,k}$ y $B_{k,j}$, $0 \leq k < q$. Así pues, en primer lugar, es necesaria una difusión de todos-con-todos de los bloques de A entre los procesos de la misma "fila" y de los bloque de B entre los procesos de la misma "columna". Una vez realizada, basta que cada proceso realice la operación citada.

Algoritmo de Fox

El algoritmo de Fox sigue la misma técnica que hemos desarrollado en que cada proceso alberga un bloque, por lo que va a resultar más eficiente en términos de requisitos de memoria. En particular, en este algoritmo cada proceso guarda un bloque de A, una copia de un bloque de A enviada por un proceso vecino, y los bloques B y C: en total, 4 bloques.

Suponemos la situación estándar en que cada proceso $P_{i,j}$ es dueño de los bloques $A_{i,j}$, $B_{i,j}$ y $C_{i,j}$. El algoritmo se desarrolla según los siguientes pasos:

1. Inicializamos una variable $\ell = 0$.
2. Cada proceso $P_{i,i+\ell}$ difunde su bloque $A_{i,i+\ell}$ a todos los procesos $P_{i,k}$, $0 \leq k < q$, de su misma fila, quienes los almacenan en un buffer destinado al efecto.
3. Cada proceso multiplica el bloque recibido $A_{i,i+\ell}$ por $B_{i,j}$ y lo acumula en $C_{i,j}$.
4. Se incrementa la variable $\ell = \ell + 1$ y se vuelve al paso 2.

Una sucesión de q difusiones y multiplicaciones completan la operación.

Algoritmo de Cannon

El algoritmo de Cannon sigue la misma técnica que hemos desarrollado en que cada proceso alberga un bloque. Pero es más eficiente en términos de consumo de memoria porque, en cada momento, solo necesita albergar un bloque de cada matriz. Cada proceso, pues, solo necesita almacenar 3 bloques: uno de A, otro de B y otro de C.

Suponemos la situación estándar en que cada proceso $P_{i,j}$ es dueño de los bloques $A_{i,j}$, $B_{i,j}$ y $C_{i,j}$. El algoritmo se desarrolla según los siguientes pasos:

1. Se realiza una "alineación" inicial de modo que el bloque $A_{i,j}$ se mueve i lugares a la izquierda y el bloque $B_{i,j}$ se mueve j lugares hacia arriba, todo módulo q.
2. Cada proceso $P_{i,j}$ multiplica su bloque de $A_{i,k}$ por el de $B_{k,j}$ y lo acumula en $C_{i,j}$.
3. Todos los procesos mueven su bloque A al proceso de su izquierda, módulo q.
4. Todos los procesos mueven su bloque B al proceso de arriba, módulo q.

Una sucesión de q multiplicaciones y desplazamientos completan la operación.

Requisitos de los programas

Se deben desarrollar los tres algoritmos utilizando el paradigma de programación con paso de mensajes mediante la biblioteca MPI.

DESARROLLO DE LA PRÁCTICA

Presentarse en la máquina

Como en las demás prácticas, hemos de presentarnos en el nodo *frontend* de nombre `sesamo`.

Edición de un programa

Para editar un programa, hay que utilizar algún editor de texto de los ya utilizados en anteriores prácticas.

Compilación y ejecución

La compilación se puede realizar editando algún fichero `Makefile` de los ya conocidos, cambiando lo necesario de acuerdo al modelo ya existente.

Para esta práctica, hay que tener en cuenta que el compilador se llama con un nombre distinto y necesitamos "cargar" en nuestro proceso la ruta para alcanzarlo. Por ello, antes de invocar al compilador debemos usar el siguiente comando:

```
cap_prac1@sesamo:~$ module load mpe2
```

Tal comando nos da acceso al compilador que soporta la biblioteca MPI. Este compilador se invoca con el comando `mpicc`, que lleva los mismos argumentos que el compilador estándar `gcc`. Un ejemplo de compilación podría ser:

```
cap_prac1@sesamo:~$ mpicc -g a.c -o a -lm
```

que tiene el significado habitual de compilar el fichero fuente `a.c` con la opción `-g` para depurar y generando como archivo ejecutable el fichero `a`, enlazando con la biblioteca matemática `libm.a`.

Como se ha indicado, podemos utilizar esos comando dentro de un fichero `Makefile` con la única precaución de cargar previamente el módulo `mpe2` en la forma explicada más arriba.

Para ejecutar el programa en el entorno MPI es necesario tener presente lo que se explica en el documento sobre el gestor de carga SLURM en la sección titulada «Cómo ejecutar programas desarrollados para MPI».