Práctica 3

Programación MPI con funciones de comunicación colectivas

Inversión de una matriz triangular inferior

Considera el problema del cálculo de la inversa de una matriz triangular inferior A de dimensión N. Escribe un programa MPI que realice el cálculo en paralelo, teniendo en cuenta que el proceso de rank 0 obtiene inicialmente la matriz. Realiza una distribución de A cíclica por bloques de tamaño C de columnas sobre todos los procesos. Al final el proceso de rank 0 debe obtener el resultado y lo imprime en la pantalla.

Algoritmo de Inversión

La base del algoritmo de inversión es la siguiente página: [Upper-Triangular Matrix Inverse Using Back-Substitution]. Es una pregunta, y además el código de ejemplo está mal, por lo que se corrigió en parte para que funcionase correctamente. Además, ese ejemplo es para una matriz triangular superior, por lo que se ha de modificar para poder aplicarse a la inferior.

Dicho esto, la resultante es la siguiente:

```
for (int j=0; j < COLS; j++) {
    M2[j][j] = 1. / M[j][j];

    for (int i=j+1; i < COLS; i++) {
        for (int k=j; k < i; k++) {
            M2[i][j] += M[i][k] * M2[k][j];
        }
        M2[i][j] /= -M[i][i];
    }
}
```

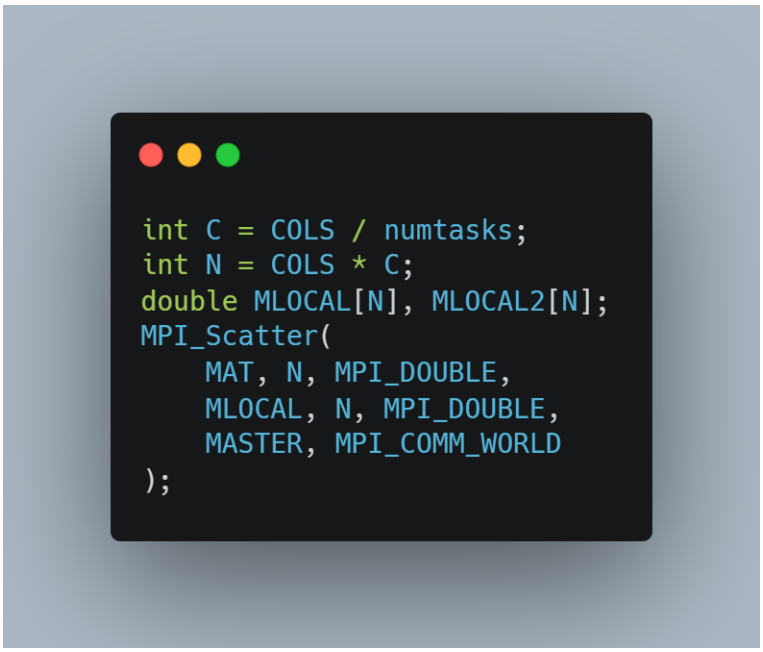
Este algoritmo invertirá la matriz triangular inferior M de tamaño $COLS\ x\ COLS$ guardando el resultado en la matriz $M2$ (previamente inicializada a 0's). Como se puede ver, la columna común es j por lo que se puede paralelizar (en parte).

Codificación del Algoritmo

La codificación se realizará de tal forma que se pueda paralelizar con MPI por columnas. De esta forma, se definen dos variables auxiliares: MT (matriz traspuesta) y MAT (matriz traspuesta como array). MPI permite enviar un array de datos y partirlo entre todas sus tareas, por lo que, si se convierte la matriz en un array con las columnas seguidas, se puede enviar ese mismo array. La definición de estas variables es la siguiente (sólo en la tarea 0, el padre):

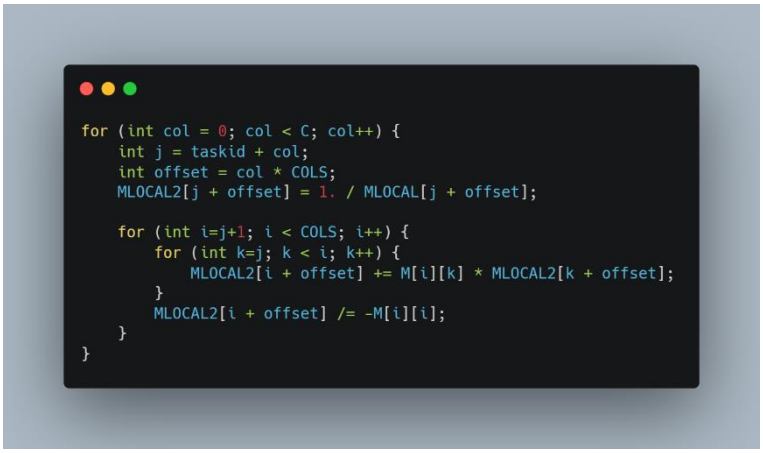
```
for (int i=0; i < COLS; i++) {
    for (int j=i; j < COLS; j++) {
        MT[i][j] = M[j][i];
        if (i != j) MT[j][i] = M[i][j];
    }
}

for (int i=0; i < COLS; i++) {
    for (int j=i; j < COLS; j++) {
        MAT[i * COLS + j] = MT[i][j];
        if (i != j) MAT[j * COLS + i] = MT[j][i];
    }
}
```

A continuación, se parte este array MAT en partes iguales dependiendo del número de procesos. En el siguiente bloque de código, C representa el número de columnas a mandar a cada proceso, y N el número de datos a recibir en cada tarea.



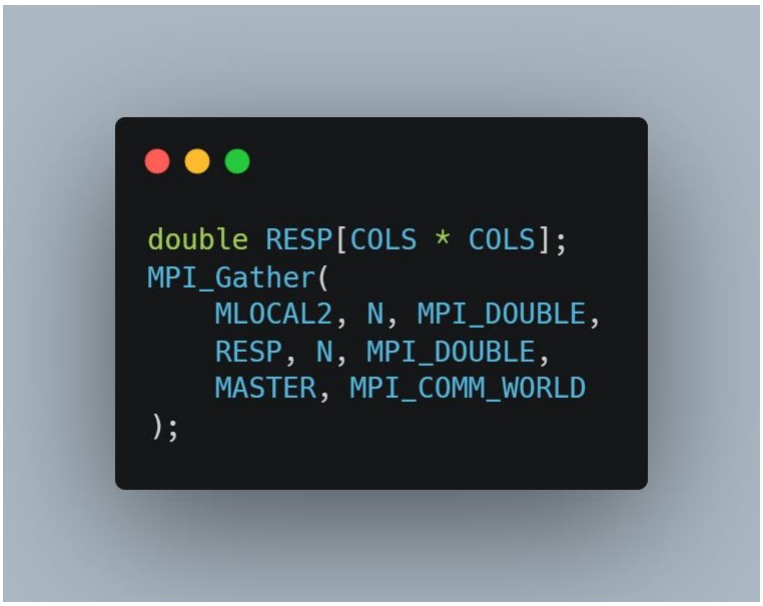
```
int C = COLS / numtasks;
int N = COLS * C;
double MLOCAL[N], MLOCAL2[N];
MPI_Scatter(
    MAT, N, MPI_DOUBLE,
    MLOCAL, N, MPI_DOUBLE,
    MASTER, MPI_COMM_WORLD
);
```

Ahora, toca aplicar el algoritmo, ya que este es código que se ejecutará en todas las tareas. Hay que tener en cuenta que puede que se envíen varias columnas a cada proceso, por lo que a pesar de que está paralelizado, todavía es necesario hacer una iteración por columnas. El resultado es el siguiente:



```
for (int col = 0; col < C; col++) {
    int j = taskid + col;
    int offset = col * COLS;
    MLOCAL2[j + offset] = 1. / MLOCAL[j + offset];

    for (int i=j+1; i < COLS; i++) {
        for (int k=j; k < i; k++) {
            MLOCAL2[i + offset] += M[i][k] * MLOCAL2[k + offset];
        }
        MLOCAL2[i + offset] /= -M[i][i];
    }
}
```

Finalmente, solo queda recolectar todos los datos y reconstruir la matriz final en el proceso padre. El código es el siguiente:



```
double RESP[COLS * COLS];
MPI_Gather(
    MLOCAL2, N, MPI_DOUBLE,
    RESP, N, MPI_DOUBLE,
    MASTER, MPI_COMM_WORLD
);
```

Y el array $RESP$ tendrá la matriz resultante por columnas. Se reconstruye, y se comprueba que efectivamente el resultado es el correcto.

Envío de Trabajos

Se ha diseñado el siguiente script para enviar los trabajos a la cola de SLURM:

```
#!/bin/bash

#SBATCH -n 16
#SBATCH -c 8
#SBATCH -t 00:00:01
#SBATCH -p thinnodes
#SBATCH -o ./out/slurm-%j.out

srun inverse
```

Medida de Tiempos

Para realizar el experimento, se toma como matriz de pruebas una matriz triangular inferior de tamaño 16x16. De esta forma, se pueden realizar pruebas con 1, 2, 4, 8 y 16 tareas (enviando 16, 8, 4, 2 y 1 columnas a cada tarea, respectivamente). Previamente se ha validado que los resultados son los correctos.

Los resultados en tiempo de computación, desde antes de realizar `MPI_Scatter` hasta después de realizar `MPI_Gather`, fueron (se han realizado 5 pruebas, se muestra la mediana):

# Tareas	Columnas / Tarea	Tiempo
1	16	0.000015
2	8	0.000109
4	4	0.000068
8	2	0.000046
16	1	0.000032

Como se puede ver, el mejor caso es en el que no se paraleliza, pero es muy probable debido al bajo tamaño de la muestra, ya que en tamaños muy pequeños no compensa paralelizar. Pero después, cuantos más procesos, se puede apreciar como lleva menos tiempo, aunque esta mejoría se va reduciendo cuantas más tareas (muy probable debido a que hay que hacer más envíos por la red y es costoso).