Practica de MPI: Multiplicación Paralela de Matrices en Máquinas de Memoria Distribuida

Como indica el título del proyecto, habrá que realizar una aplicación paralela que calcule el producto de dos matrices de números en coma flotante de grandes dimensiones. La idea es ver el beneficio que se puede obtener con una aplicación paralela frente a una secuencial, por tanto, habrá que tener un programa secuencial con el que comparar los resultados que obtengamos, para ver que nuestro programa funciona correctamente, así como para comparar los tiempos obtenidos.

A la hora de realizar esta aplicación, lo más sencillo es emplear un esquema Maestro –Esclavo. El maestro organiza el trabajo a realizar y lo va distribuyendo entre los N esclavos, quedando a la espera de que le soliciten más trabajo. Además, deberá gestionar los resultados.

Los esclavos, por su parte, van pidiendo el trabajo a realizar al maestro y, cuando acaben con dicho trabajo, vuelven a pedirle más trabajo, hasta que no haya más trabajo a realizar. Entonces avisan al maestro que han acabado, finalizando su ejecución.

Sobre este esquema podemos emplear dos tamaños distintos para las tareas. En el primero caso la unidad de trabajo será el elemento (i, j) y en el segundo será toda una fila de la matriz resultado.

El inconveniente de este esquema maestro-esclavo es que habrá un proceso encargado de repartir el trabajo y de agrupar los resultados, teniendo solo N-1 esclavos. Por tanto, habrá un proceso que no hará trabajo útil. La alternativa, como todas las tareas tienen un tamaño parecido, es hacer una distribución estática del trabajo y suprimir el maestro. Así, todos los nodos calcularán elementos de la matriz resultado.

La forma más sencilla de medir el tiempo empleado por la aplicación es usar la función MPI_Wtime, para obtener el tiempo inicial y el final, de forma que la diferencia entre ambos es el tiempo empleado.

Entonces, **habrá que realizar**:

- Un programa secuencial, que servirá de referencia a los paralelos.
- Uno distribuido con asignación dinámica de trabajo, cuya unidad de trabajo es el elemento (i, j) de la matriz resultante.

  MR[i][j] = M1[i][1] * M2[1][j] + ... + M1[i][N]*M2[N][j]
- Otro distribuido, también con asignación dinámica de trabajo, cuya unidad de trabajo sea la fila i-ésima de la matriz resultado:

  $M_{i1}, M_{i2}, ..., M_{iN}$
- Un cuarto caso con asignación estática del trabajo de forma que no preguntarán que trabajo les corresponde hacer. Habrá que agrupar los resultados igualmente, pero la asignación ya vendrá establecida de antemano. En este esquema todos los nodos harán trabajo útil.

Una vez hechos estos programas **habrá que medir el tiempo que emplea cada uno** de ellos en ciertos casos. Para simplificar al máximo aspectos no relativos a la práctica, como la gestión de memoria, se tomarán matrices estáticas, de forma que no habrá que asignarles memoria al inicio de la ejecución ni liberarlas al final de la aplicación.

Como las matrices son estáticas, los datos iniciales sobre los que trabajará cada esclavo ya estarán disponibles. El maestro indicará solo que trabajo debe hacer cada uno, pero no los datos sobre los que debe operar. Como una alternativa interesante se deja la posibilidad de distribuir los datos de la matriz usando Broadcast o Scatter (MPI_Bcast o MPI_Scatter), indicándolo claramente.

Los casos a considerar son los siguientes:

- 60x80000 * 80000x60
- 600x8000 * 8000x600
- 600x400 * 400x8000
- 6000x100 * 100x6000

La ejecución de estos cuatro programas sobre estos cuatro casos nos dará 16 tiempos. Cada medida deberá repetirse varias veces para evitar distorsiones provocadas por un momento de alta carga.

Con estas medidas se calculará el *speedup* conseguido en cada caso.

Por último, deberéis comentar y razonar por qué se dan estas medidas o resultados. A partir de las explicaciones que hayáis encontrado indicad vuestras conclusiones sobre los resultados obtenidos.

En la memoria se incluirá el código, así como las medidas obtenidas en cada caso. Además, se pondrán las explicaciones a dichas medidas, las conclusiones que se pueden obtener a partir de estas medidas y los comentarios personales sobre la práctica realizada.

Observación: Compilación y ejecución

Un pequeño comentario sobre las máquinas a utilizar. Serán las máquinas `triqui1` a `triqui4`. Estas máquinas tienen cada una dos procesadores, con cuatro núcleos cada uno. Además tienen tres tarjetas de red Gigabit cada una, que se han usado para dar acceso de red a máquinas virtuales que se han implementado sobre los nodos reales, de modo que se ven bastantes más nodos de los que realmente existen.

Hay dos compiladores instalados, el gcc 4.1.2 de GNU (por defecto) y el de Intel, el icc 11.0 (`/opt/intel/Compiler/11.0/074/bin/intel64`)

Usaremos MPICH (está en /usr/local/mpich2-1.4/bin) y gcc para la práctica de modo que los resultados sean fácilmente comparables. Si se quiere cambiar de compilador de C, basta con indicar el compilador a usar, gcc o icc, en la variable MPICH_CC.

A la hora de compilar se empleará “mpicc”:

mpicc –O3 -o ejecutable codigo.c

A la hora de ejecutar los distintos casos emplearemos 6 nodos. Por tanto, la ejecución será:

```
mpirun -np 6 ./ejecutable argumentos
```

o `mpiexec -np 6 ./ejecutable argumentos`

Todas las medidas se tomarán con ejecutable optimizados (-O3) para poder comparar resultados.

Fecha límite para la entrega: viernes 5 de diciembre 2014