Programación de Sistemas Distribuidos

Curso 2015/2016

Práctica 2
Message Passing Interface (MPI)

1. Introducción

En los siguientes ejercicios, se pide utilizar el lenguaje C junto con una implementación del API MPI para crear distintos programas paralelos. En todos los programas paralelos debe suponer que un proceso actúa como coordinador; llamemos a los demás procesos trabajadores.

2. Multiplicación de matrices cuadrados

Se pide crear distintos programas que realicen la multiplicación de dos matrices. Supongamos que, al inicio, solo uno de los procesos (el coordinador) tiene los datos de las dos matrices que se quiere multiplicar. Para simplificar los ejercicios, la solución proporcionada para la multiplicación de matrices en C (ejercicio 0) incluye un pequeño programa para generar matrices de enteros.

En todos los ejercicios, debe usar la función `MPI_Wtime()` para cronometrar el tiempo de ejecución del programa (incluyendo la parte de comunicaciones, claro está). Debe probar lanzar su solución a cada ejercicio con distintos tamaños de matrices. Luego, debe comprobar cómo cambian los tiempos de ejecución con un tamaño fijo de matrices (de preferencia, bastante grande) y distintos valores del número de procesos, donde los valores de más interés son:

- todos los valores posibles más pequeños que el número de cores lógicos que tenga el ordenador. En Linux, se puede conocer el número de cores físicos que tiene el ordenador mediante el comando:
  ```
  grep "^core id" /proc/cpuinfo | sort -u | wc –l
  ```
  y el número de cores lógicos mediante el comando:
  ```
  grep -c ^processor /proc/cpuinfo
  ```
- algunos valores mayores que el número de cores lógicos.

Por supuesto, también es interesante mirar el resultado de variar el número de procesos en múltiples ordenadores con distinto número de cores. Finalmente, también debe comparar los tiempos de ejecución de las soluciones de los distintos ejercicios con un tamaño fijo de matrices (bastante grande) y un tamaño fijo de procesos (bastantes).

Siempre es de utilidad mostrar los resultados gráficamente. Por ejemplo, si se enviaran los resultados a un fichero (en modo `append`), digamos `ejercicioX.dat`, como sigue:

```
fprintf(fd,"%d\t%d\n", numeroDeProcesos, tiempoFin-tiempoIni)
```

para cada uno de los valores de número de procesos, luego se podría mostrar estos resultados gráficamente con el siguiente comando de gnuplot:

```
gnuplot -p -e "set terminal svg; set output 'Procesos.svg'; set style data linespoints; plot 'ejercicioX.dat'"
```

(o con otro programa, si prefiere). Si, en cada ejercicio, se haya podido generar un fichero usando el mismo tamaño de matriz y los mismos valores para el número de procesos, sería muy interesante mostrar en el mismo grafo los resultados de los ficheros de todos los ejercicios con el siguiente comando de gnuplot:

```
gnuplot -p -e "set terminal svg; set output 'Procesos.svg'; set style data linespoints; plot 'ejercicio1.dat', 'ejercicio2.dat',... 'ejercicioN.dat'"
```

(o con otro programa, si prefiere). Luego, otro grafo para otro número de cores.

A ser posible, debe ejecutar sus programas utilizando una versión *reciente* de OpenMPI para poder observar la asignación de procesos a cores mediante el parámetro del comando `mpiexec -report-bindings` (ver transparencias).

En todos los ejercicios siguientes, se debe partir una matriz en trozos para poder enviar un trozo distinto a cada uno de los distintos procesos. Los trozos constan de una o más columnas de la matriz en cuestión. No se pide paralelizar más; si hay más procesos que columnas de la matriz en cuestión, habrá procesos que no ejecutan ningún trabajo. Nótese que para facilitar el envío de trozos de la matriz, sería conveniente almacenarla en memoria como un array unidimensional, no como un array de dos dimensiones tal como se almacena en la solución proporcionada al ejercicio 0.

Ejercicio 0

En CV se encuentra una solución a la multiplicación de matrices de manera secuencial que escribe y lee matrices de fichero. Compile el programa con un compilador de C y ejecutelo. Luego modifíquelo, compilelo con un compilador de MPI y ejecútelo.

Ejercicio 1

Se pide paralelizar el programa del ejercicio 0 utilizando las siguientes funciones MPI:

- `MPI_Bcast` para enviar uno de los operandos del coordinador a los trabajadores.
- `MPI_Send`, que ejecutará el coordinador, y `MPI_Recv`, que ejecutarán los trabajadores, para enviar un trozo distinto del otro operando del coordinador a cada trabajador.
- `MPI_Send`, que ejecutarán los trabajadores, y `MPI_Recv`, que ejecutará el coordinador, para enviar el resultado calculado por cada trabajador al coordinador.

Ejercicio 2

Se pide modificar el programa del ejercicio 1 para que utilice las siguientes funciones:

- `MPI_Bcast` para enviar uno de los operandos del coordinador a los trabajadores;
- `MPI_Isend`, que ejecutará el coordinador, y `MPI_Recv`, que ejecutarán los trabajadores, para enviar un trozo distinto del otro operando del coordinador a cada trabajador,
- `MPI_Send`, que ejecutarán los trabajadores, y `MPI_Irecv`, que ejecutará el coordinador, para enviar el resultado calculado por cada trabajador al coordinador.

Con el fin de obligar al proceso coordinador a esperar hasta que terminen los `MPI_Isend` antes de emprender su parte del trabajo y hasta que terminen los Irecv antes de emprender la impresión de los resultados, se puede utilizar `MPI_Waitall()` (habiendo declarado y usado previamente un array de `MPI_Request`).

Ejercicio 3

Se pide modificar el programa del ejercicio 2 para que utilice las siguientes funciones:

- `MPI_Bcast` para enviar uno de los operandos del coordinador a los trabajadores,
- `MPI_Scatter` para enviar un trozo distinto del otro operando del coordinador a cada proceso,
- `MPI_Gather` para recoger en el coordinador el resultado calculado por cada proceso.

Observe que solo se debe usar `MPI_Scatter` si el tamaño de los datos (en este caso, la matriz que se divide en trozos) es divisible por el número de procesos. Por tanto, si no es el caso, el programa debería imprimir un mensaje adecuado y terminar antes de llamar a `MPI_Bcast`.

Ejercicio 4

Se pide modificar el programa del ejercicio 3 para que utilice las siguientes funciones:

- `MPI_Bcast` para enviar uno de los operandos del coordinador a los trabajadores,
- `MPI_Scatterv` para enviar un trozo distinto del otro operando del coordinador a cada proceso,
- `MPI_Gatherv` para recoger en el coordinador el resultado calculado por cado proceso.

Nótese que con `MPI_Scatterv`, el tamaño de las matrices no es necesariamente divisible por el número de procesos.

3. Entrega

El código de su solución a los distintos ejercicios se tiene que entregar por Campus Virtual en un solo archivo comprimido mediante zip. El archivo ha de incluir una breve memoria explicando las soluciones implementadas y lo que ha observado de los tiempos de ejecución en los casos pedidos, de preferencia, en forma gráfica.

Hay dos fechas de entrega posible tal como se explica a continuación:

- Si la práctica se entrega antes de las 12h del día 27 de enero de 2016, la corrección presencial podrá hacerse en el laboratorio que tiene lugar de 18h a 20h ese mismo día.
- Si no, la práctica se tiene que entregar antes de las 12h del día 18 de febrero de 2016 y la corrección presencial tiene que tener lugar en el despacho del profesor antes del 25 de febrero a una hora acordada con el profesor. Si la práctica se entrega antes, se puede acordar una hora para la corrección presencial antes también.

En los dos casos, en la práctica presencial deben estar presente los dos miembros del grupo de prácticas.