Práctica

1

Multiprocesadores

Utilización de librerías de paso de mensajes. MPI.

En esta práctica se trabajará con aspectos básicos de la programación MPI. Empezaremos con las comunicaciones punto a punto síncronas (bloqueantes).

Ejercicio 1. Uso del modo básico

Se te ha hablado en clase del orden y la equidad en la llegada de los mensajes.

A. Deberás hacer un programa en MPI de dos procesos, un emisor y un receptor. El emisor escribirá 10 mensajes de tipo `int`, el orden de envío será a través de la etiqueta del mensaje. El receptor recogerá esos mensajes y los pintará en pantalla. Indica cuál es el orden de llegada.

B. Copia y modifica (si es necesario) el programa anterior para que haya varios emisores y un único receptor. El receptor recogerá esos mensajes y los pintará en pantalla. Indica cuál es el orden de llegada.

C. Copia y modifica el programa anterior. El receptor en este caso escogerá dentro del buffer de recepción tanto el emisor como la etiqueta en un orden que no sea el natural y que has comprobado en el punto B, por ejemplo, en orden inverso.

D. Como has visto en teoría, en caso de que dos procesos hagan un ping-pong (send-rec) se pueden dar casos (dependiendo de la implementación de los búferes) de programas seguros, no seguros o erróneos (bloqueo). Escribe tres programas que creen dos procesos MPI haciendo un ping-pong en los tres casos que hemos visto. Básate en el ejemplo del Moodle. El mensaje será un string con un mensaje de saludo. En el caso del ejemplo 3 indica si funciona. Si funciona di porqué. Cambia el programa para que recoja de la línea de comando una longitud. Ejecuta varias veces el programa para distintos tamaños de mensaje hasta que se bloquee. Di por qué ocurre.

E. Copia y modifica el programa B para que el receptor además escriba por cada mensaje en número de bytes que ha recibido.

Ejercicio 2. Uso de otros modos de envío

Además del modo básico, hemos visto que existen otros modos de funcionamiento. En este apartado comprobaremos que el mundo teórico no es tan perfecto como el real y que muchas cosas dependen de la implementación del estándar. O que incluso podemos hacer las cosas mal sin que el sistema nos avise (el peor de los casos). Desarrollaremos esto en los siguientes apartados:

A. Copia el ejemplo del apartado 2.E y modifícalo para usar el envío bufereado. En este caso el mensaje será un array de N enteros, donde N será pasado desde la línea de comando (`mpirun`). Deberás crear tu propio búfer de la manera que quieras (memoria dinámica, arrays estáticos o arrays dinámicos). El tamaño será del total de los mensajes más el MPI_BSEND_OVERHEAD.

B. Copia el anterior y modifica el programa para que el búfer sea de 1 byte intencionadamente ¿qué ocurre y qué debiera ocurrir? Prueba diferentes valores de longitud de mensaje hasta que el sistema te avise. Indica cuál es esta longitud.

C. Usando como modelo el programa 2.C, copia y cambia el programa y haz que un proceso haga dos envíos y el otro dos recepciones con diferente orden, por ejemplo se envía 1 y 2 y se recibe 2 y 1. ¿Cómo debes poner el modo de los envíos para que no termine el programa? ¿Por qué? Compruébalo.

D. Lee el manual de los envíos *`ready`* y prueba si tu implementación lo cumple (puedes poner `sleep` para asegurarte y comprobar lo que devuelve la primitiva `MPI_Rsend`). El manual literalmente dice:

A send that uses the ready communication mode may be started only if the matching receive is already posted. Otherwise, the operation is erroneous and its outcome is undefined. On some systems, this allows the removal of a hand-shake operation that is otherwise required and results in improved performance. The completion of the send operation does not depend on the status of a matching receive, and merely indicates that the send buffer can be reused. A send operation that uses the ready mode has the same semantics as a standard send operation, or a synchronous send operation; it is merely that the sender provides additional information to the system (namely that a matching receive is already posted), that can save some overhead. In a correct program, therefore, a ready send could be replaced by a standard send with no effect on the behavior of the program other than performance.

Ejercicio 3. La memoria paralela y el paradigma master/slave

Cuando se piensa en un programa paralelo MPI, es un error frecuente replicar la memoria del programa secuencial en todos los procesos y acceder desde cada programa a sólo la parte correspondiente (paralelismo de datos). En muchos casos simplemente por comodidad a la hora de escribirlo.

En este ejercicio deberás pensar desde el principio en un problema que se realiza de forma paralela, usando algo tan sencillo como realizar la suma de los N primeros números naturales, donde N se te pasa desde la línea de comando el programa. El resultado debería ser:

$$\frac{n(n+1)}{2}$$

Harás varias versiones:

A. La primera será la errónea. Se tendrá un *array* de N posiciones (que estará en cada proceso ocupando memoria innecesariamente) inicializadas con el índice del *array*. Cada proceso esclavo accederá a una zona del *array*. El último proceso será el será el máster, el cero será el primer esclavo. Piensa en cómo deben inicializar los *array*. ¿Lo podrías hacer sólo en el máster? Después cada proceso hará la suma parcial del *subarray* que le toca y mandará el resultado al proceso máster que a su vez recogerá los mensajes, hará la suma total de lo recogido y pintará el resultado. El tamaño N del array para simplificar el programa será múltiplo de P-1, siendo P el número de procesos. Comprueba que el resultado es correcto.

B. La segunda, la correcta. Si cada esclavo sólo accede a N / (P-1) datos, sólo es necesario declarar un array de este tamaño en cada proceso esclavo (ver figura). Cada proceso esclavo dependiendo de su índice rellenará el array de la forma correcta, sumará los datos y se los enviará al máster. Comprueba su correcto funcionamiento.

C. La tercera sería la real. En este ejemplo tan sencillo piensa si realmente necesitas tener un *array* para almacenar esos datos. Modifica el B y compruébalo.