Nombre y apellido: Enric Gil Gallen

Nombre y apellido: Victor Granados Segara

Tiempo: 0:50

Tema 11. Comunicaciones Punto a Punto en MPI

Tema 12. Comunicaciones Colectivas en MPI

1 Se dispone de un código secuencial que calcula el número de primos contenidos en un vector de enteros, y se desea realizar una implementación paralela que permita aumentar las prestaciones.

Aprovecha el código secuencial para implementar una primera versión paralela en la que intervengan sólo dos procesos: el proceso 0 que dispone de todos los números a procesar, y el proceso 1 que realizará todo el procesamiento, ya que sólo éste ejecutará la función `esPrimo`. Aunque esta versión paralela no será nada eficiente, será muy útil como una primera aproximación.

En esta implementación, **el proceso 0 envía todos los números** que desea evaluar al proceso 1 de uno en uno, es decir, **cada mensaje solo incluye un número**. Una vez se haya enviado todos los números al proceso 1, **el proceso 0 debe enviar un mensaje especial de terminación (mensaje envenenado)** para indicarle al proceso 1 que debe terminar. Cuando el proceso 1 ha terminado de procesar todos los números, debe envíar al proceso 0 el número de primos encontrado, y éste lo imprimirá en pantalla.

Existen dos detalles que hay que considerar en la implementación de la versión paralela. En primer lugar, decir que el tipo base de los elementos del vector es `long long` (su correspondiente tipo en MPI es `MPI_LONG_LONG_INT`), es decir, un entero de 8 octetos. Respecto de la comunicación, los mensajes donde se incluyen números a procesar irán marcados con la etiqueta 88, mientras que el mensaje envenenado que avisa al trabajador (proceso 1) que debe terminar, irá marcado con la etiqueta 99.

No olvides comprobar que el número de primos obtenido por la versión paralela coincide con el obtenido por la versión secuencial.

Para facilitar la implementación del código, se debe trabajar con un vector corto con números no muy grandes. Por tanto, asegúrate de que al principio del código aparece la siguiente línea:

```
// Ejercicio 1
// El proceso 0 envia todos los numeros y un veneno
if (miId == 0) {
   for (i = 0; i < dimVectorNumeros; i++) {
      MPI_Send(&vectorNumeros[i], 1, MPI_LONG_LONG_INT, 1, 88, MPI_COMM_WORLD);
   }

   MPI_Send(&veneno, 1, MPI_LONG_LONG_INT, 1, 99, MPI_COMM_WORLD);

   MPI_Recv(&numPrimosPar, 1, MPI_INT, 1, 88, MPI_COMM_WORLD, &s);

}
```

```
// El proceso 1 recibe numeros hasta que recibe un veneno
// ...
// El proceso 1 envia cuantos primos habia al proceso 0, que lo recibe
// ...
if (miId == 1) {
   int cont = 0;
   while (1) {

      MPI_Recv(&actual, 1, MPI_LONG_LONG_INT, 0, MPI_ANY_TAG, MPI_COMM_WORLD, &s);

      if (s.MPI_TAG == 99) {
         MPI_Send(&cont, 1, MPI_INT, 0, 88, MPI_COMM_WORLD);
         break;
      }

      if (esPrimo(actual)) {
         cont++;
      }

   }

}


if (miId == 0) {
   t2 = MPI_Wtime();
   ttParEje1 = t2 - t1;
   printf("Implementacion paralea Ej1.  Tiempo (s): %lf\n", ttParEje1);
   printf("Numero de primos en el vector: %d\n", numPrimosPar);
   printf("\n");
}
```

Ejercicio 2

Escribe a continuación la parte de tu código que realiza la implementación paralela.

```
// Ejercicio 2
```

```
// Varialbe creadas por nosotros
char pedir = 0;
int procesosTrabajadores = numProcs - 1;
int cont = 0;
int envio = 0;
// -------------------------------------

MPI_Barrier(MPI_COMM_WORLD);
t1 = MPI_Wtime();
numPrimosPar = 0;

// El proceso 0 recibe peticiones y responde con numeros o venenos
if(miId == 0){
   for (i = 0; i < dimVectorNumeros; i++) {
      MPI_Recv(&pedir, 1, MPI_CHAR, MPI_ANY_SOURCE, 88, MPI_COMM_WORLD, &s);

      envio = s.MPI_SOURCE;


      MPI_Ssend(&vectorNumeros[i], 1, MPI_LONG_LONG_INT, envio, 88, MPI_COMM_WORLD);

   }

   for (i = 0; i < procesosTrabajadores; i++){
            MPI_Recv(&pedir, 1, MPI_CHAR, MPI_ANY_SOURCE, 88, MPI_COMM_WORLD, &s);

      envio = s.MPI_SOURCE;

      MPI_Ssend(&veneno, 1, MPI_LONG_LONG_INT, envio, 77, MPI_COMM_WORLD);
   }
}
else{
   // Los procesos trabajadores envian peticiones y reciben numeros o un veneno


   while (1) {
      MPI_Ssend(&pedir, 0, MPI_CHAR, 0, 88, MPI_COMM_WORLD);
            // El tapon tiene que ser 0
      MPI_Recv(&actual, 1, MPI_LONG_LONG_INT, 0, MPI_ANY_TAG, MPI_COMM_WORLD, &s);

      if (s.MPI_TAG == 77) {
         break;
      }

      if (esPrimo(actual)) {
         cont++;
      }

   }
}
// Todos los procesos colaboran para obtener el numero de primos
MPI_Reduce(&cont ,&numPrimosPar, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

MPI_Barrier(MPI_COMM_WORLD);
```

```
   t2 = MPI_Wtime();
   ttParEje2 = t2 - t1;


   if (miId == 0) {
      printf("Implementacion paralela.    Tiempo (s): %lf\n", ttParEje2);
      if (argc == 1)
         printf("Implementacion paralela.    Incremento: %lf\n", ttSec / ttParEje2);
      printf("Numero de primos en el vector: %d\n", numPrimosPar);
   }

// El proceso 0 destruye el vector de primos.
   if (miId == 0) {
      free(vectorNumeros);
   }

// Fin de programa.
   if (miId == 0) {
      printf("\n");
      fflush(stdout);
   }
   MPI_Barrier(MPI_COMM_WORLD);
   printf("Proc: %d  Fin de programa\n", miId);

   MPI_Finalize();

   return 0;
}
```

Ejercicio 3

3 Una vez funcione perfectamente el programa anterior, modifícalo para que emplee la rutina `MPI_Send` en lugar de la rutina `MPI_Ssend`. La primera es más eficiente pero ayuda menos en la detección de errores.

Comprueba que el programa continúa funcionando correctamente.

- Funciona correctamente

Ejercicio 4

4 Completa la siguiente tabla con la última versión paralela.

Desarrolla el programa en el ordenador del aula, mientras que la producción de tiempos e incrementos debes realizarlas en patan. Redondea los tiempos dejando sólo tres decimales y redondea los incrementos dejando dos decimales.

Para la obtención de los tiempos es conveniente que el programa procese un vector grande. Para ello, asegúrate de que al principio del código aparece la siguiente línea:

```
#undef VECTOR_CORTO
```

Una vez hayas comprobado el programa a fondo en tu ordenador local, debes sustituir las llamadas a la rutina `MPI_Ssend` de tu programa por llamadas a `MPI_Send`, para que así sea más eficiente su ejecución.

Ten en cuenta que en patan **el código secuencial sólo debe ser ejecutado en una única ejecución**. Es muy importante que el código secuencial no sea ejecutado para cada ejecución (para 2 procesos, para 4 procesos, etc.) para de esa forma agilizar las ejecuciones y no saturar la cola de trabajos. Para ello, simplemente debes añadir un parámetro cuando ejecutes el programa.

Implementación	Tiempo	Incremento
Secuencial		—
Paralela con 2 procesos		
Paralela con 4 procesos		
Paralela con 6 procesos		
Paralela con 8 procesos		

Justifica los resultados obtenidos.

Implementación	Tiempo	Incremento
Secuencial	19.916	-
Paralela con 2 procesos	19.847	1.00
Paralela con 4 procesos	6.634	2.97
Paralela con 6 procesos	4.149	4.74
Paralela con 8 procesos	3.059	6.42

Al tener una repartimiento distribuido (el trabajo se va repartiendo a medida que se hace) se observa un incremento el rendimiento a medida que aumentan los procesos

Ejercicio 5

5 ¿Cuál es el máximo incremento que se puede conseguir?

Pues diría que es el número de procesos menos uno ya que el proceso “0” es el que distribuye la faena.