Servicios de Computación de Altas Prestaciones y Disponibilidad
MPI

Daniel Aguado Araujo

10 de mayo de 2015

Índice

1. Enunciado

Conocida una función f(x) mediante una secuencia de N pares $(x_i, f(x_i))$, tal que $x_{i-1} < x_i$.

Se conoce como:

Suma de Riemann por la izquierda.

$S_{izquierda} = \sum_{i=1}^{N-1} f(x_{i-1}) * (x_i - x_{i-1})$

Suma de Riemann por la derecha.

$S_{derecha} = \sum_{i=1}^{N-1} f(x_i) * (x_i - x_{i-1})$

Suma de Riemann trapezoidal.

$S_{izquierda} = \sum_{i=1}^{N-1} \frac{f(x_{i-1})+f(x_i)}{2} * (x_i - x_{i-1})$

Desarrollar un programa en C con MPI que calcule de forma paralela, en un entorno con P procesadores, los valores $S_{izquierda}$, $S_{derecha}$ y $S_{trapezoidal}$ de una función definida por puntos almacenada en un fichero secuencial *data.dat* de N puntos, y muestre el tiempo que ha tardado en ejecutarse.

El fichero almacena los puntos por parejas $[x_0, f(x_0), x_1, f(x_1), ..., , x_{N-1}, f(x_{N-1})]$.

Todos los valores son de tipo *double*.

Nota

Realizar la implementación mediante un proceso inicial que lee el fichero y distribuye los datos entre los demás procesos, para finalmente mostrar el resultado.

El tiempo de ejecución a calcular, será el transcurrido a partir de recibir todos los procesos sus datos.

Entrega

- **Documento Teoría.pdf** En este documento se explicita los pasos necesarios para realizar el algoritmos paralelo: división, comunicaciones, agrupación y asignación. También se incluirá una breve descripción del algoritmo y su implementación.
- **Fichero fuente Riemann.c** Este fichero deberá de compilarse *mpicc -o nombre_ del_ ejecutable Riemann.c* sin errores para poder ejecutar sin error el siguiente comando: *mpirun -np numero_ procesadores nombre_ del_ ejecutable tamaño_ del_ problema Fichero.dat*. El resultado deberá ser:
  - Sizq = Valor
  - Sder = Valor
  - Strap = Valor
  - Tiempo máximo de ejecución = Valor segundos.

2. División

El tamaño del problema viene dado por parámetros del programa según el enunciado, por lo que se tomará dicho número para la división del problema.

En este caso el tamaño del problema se representa por el número de conjuntos de pares que tiene el problema, por lo que cada tarea tendrá los datos de un conjunto de pares.

Siendo más claro, el número de tareas en las que se divide el programa coincide con el tamaño del problema especificado.

3. Comunicaciones

Según la división establecida en el punto anterior y las dependencias de datos encontradas en las fórmulas de las sumas de Riemann, tenemos que cada tarea necesita saber los datos de la tarea anterior. Es decir, que para calcular las sumas de Riemann la tarea N necesita conocer los datos de la tarea N - 1.

Debido a que el tamaño del problema es muy pequeño y que se va a trabajar en una máquina local, las tareas conocerán los datos del problema completo. De esta forma se ahorra en condiciones de control y tiempo en distribuir los datos. En un problema más grande o para ejecutarse en un clúster real, habría que tener en cuenta la topología de la red para ahorrar el máximo de tiempo en las comunicaciones.

4. Agrupación y Asignaciones

La agrupación establecida será una tarea a cada proceso, siempre que haya suficientes. Si hay un menor número de procesos que de tareas, entonces la agrupación será tal que cada proceso ejecute *tamaño del problema / número de procesos* tareas.

La forma en la que se detalla la agrupación es la siguiente:

Supongamos un tamaño del problema 10 y un número de procesos de 3. Entonces el proceso 0 ejecutará las sumas con índices 1, 4, 7 y 10; el proceso 1 ejecutará las sumas con índices 2, 5 y 8; y el proceso 2 ejecutará las sumas con índices 3, 6 y 9. Es decir, cada proceso ejecutará las sumas con índices tal que índice = (identificador del proceso + 1) (más uno porque las sumas de Riemann empiezan en 1), sumándole al índice el valor del número de procesos mientras que dicho índice sea menor que el tamaño del problema. En el siguiente listado se ve con más claridad cómo queda el reparto en un bucle for.

```
1 for (i = id + 1; i < tamanyo_problema; i += numero_procesadores)
```

Debido a que cada procesador realiza una parte del sumatorio diferente al resto, queda agrupar los resultados parciales. Para ello se utilizan las funciones de reducción de MPI, que aplica la operación suma a todos los valores recibidos y devuelve el resultado total.

Como se ha visto las agrupaciones y asignaciones son dinámicas dependiendo del tamaño del problema y del número de procesos que se ejecuten, aunque hay ciertas tareas que están asignadas de manera fija al proceso con identificador igual a 0. Dichas tareas son las de lectura del fichero y

envío de datos, y la recopilación e impresión de los resultados obtenidos.

5. Resultado

Para obtener el resultado del problema se ha realizado primero una versión del mismo en secuencial y después una versión en paralelo, para poder comprobar la correcta realización de las operaciones y la temporización.

Para obtener la temporización en el problema secuencial se ha utilizado la función clock de C y para el problema paralelo se han utilizado las funciones clock de C y MPI_WTime de MPI, que arrojan resultados que difieren un poco, ya que la función de C cuenta el tiempo real de ejecución y la función de MPI el tiempo total de ejecución en el procesador.

Los tiempos de ejecución con un problema de tamaño 10 y 10 procesos ha sido de:

- Tiempo secuencial: (Tiempo C) 0.000039 segundos
- Tiempo paralelo: (Tiempo C) 0.000115 segundos (Tiempo MPI) 0.000567

Los tiempos de ejecución con un problema de tamaño 1000 y 10 procesos ha sido de:

- Tiempo secuencial: (Tiempo C) 0.000069 segundos
- Tiempo paralelo: (Tiempo C) 0.000299 segundos (Tiempo MPI) 0.000295

Los tiempos de ejecución con un problema de tamaño 10000 y 4 procesos ha sido de:

- Tiempo secuencial: (Tiempo C) 0.000153 segundos
- Tiempo paralelo: (Tiempo C) 0.000106 segundos (Tiempo MPI) 0.000104

6. Conclusiones

Se puede concluir que para un problema como este, donde las operaciones tienen un coste despreciable, el coste las comunicaciones es mayor que el coste computacional secuencial y no merece la pena paralelizarlo.

La única ocasión en la que merecería la pena paralelizarlo sería en tamaños de problema gigantescos donde el coste de las operaciones sea mayor que el coste de las comunicaciones, y eso parece que solo se da con tamaños de problema de 10000 y mayores y con no demasiada paralelización.