Computación Paralela - Práctica 2

Uso de números aleatorios en OpenMP y cálculo del número Pi usando el método de Montecarlo e Hibridación MPI-OpenMP

Cálculo de Pi usando el método de Montecarlo

El cálculo de Pi usando el método de Montecarlo se basa en generar un punto de forma aleatoria en un cuadrado de lado 1. Si el punto está a una distancia inferior o igual a 1 del punto (0,0), estaría dentro de un cuadrante de una circunferencia de radio 1, como se muestra en la figura 1.

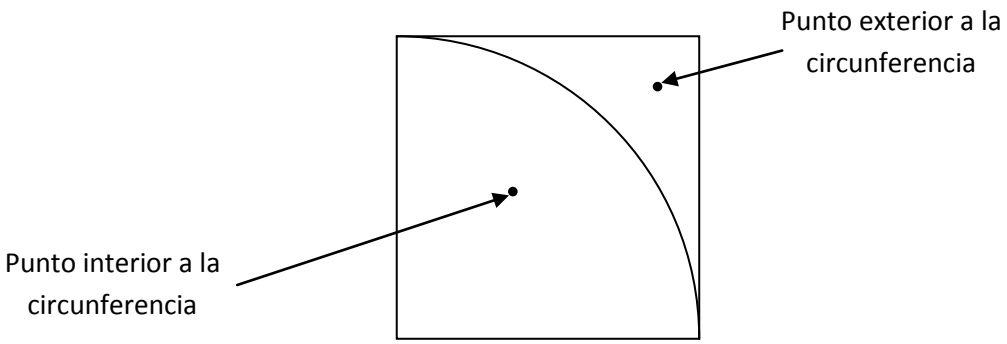

Figura 1. Cuadrante de una circunferencia de radio 1 en un cuadrado de lado 1.

Si tenemos en cuenta que el área de una circunferencia es $A_c=\pi r^2$, y dado que r=1, tenemos que $A_c=\pi$. Además, como estamos teniendo en cuenta únicamente un cuadrante de la circunferencia, $A_q=\pi/4$, por lo que $\pi=4A_q$. Debemos entonces estimar el área A_q, y para ello lo que vamos a hacer es generar puntos aleatorios dentro del cuadrado de 1x1, es decir, entre los puntos (0, 0) y (1, 1), de tal forma que, si la distancia *d* entre el punto generado (x, y) y el punto de origen (0, 0) es inferior o igual a 1, incrementamos el contador de números internos, siendo $d = \sqrt{x^2 + y^2}$. Una vez hayamos terminado de generar los puntos aleatorios, basta con dividir el valor del contador entre el número de iteraciones (puntos internos entre puntos totales), lo que nos da un valor estimado de A_q, y por tanto, un valor estimado de π. Cuando mayor sea el número de puntos, más nos aproximaremos a π. El algoritmo 1 muestra el pseudocódigo del método aquí descrito.

```
contador=0
para i desde 1 hasta NumIteraciones
  x=generar número aleatorio
  y=generar número aleatorio
  d=distancia_a_0_0(x, y)
  si d<=1 entonces contador=contador+1
fin para
pi=contador*4/NumIteraciones
```

Algoritmo 1. Cálculo de Pi usando el método de Montecarlo.

Uso de números aleatorios en OpenMP

Para estudiar la influencia de los números aleatorios en OpenMP vamos a utilizar tres librerías: la estándar de C++ (rand), la librería mramd y la librería MersenneTwister (ambas pueden descargarse del campus virtual). A continuación se muestra cómo generar números aleatorios con las tres librerías.

Estándar C++

```
srand(semilla); //Inicializa la semilla para que la secuencia sea siempre diferente.
x=rand();//Genera un número entero entre 0 y RAND_MAX.
        //Para generar entre 0 y 1 basta dividir x entre RAND_MAX.
```

mramd

```
sramd(semilla); //Inicializa la semilla para que la secuencia sea siempre diferente.
x=ramd();//Genera un número entero entre 0 y MRAND_MAX.
         //Para generar entre 0 y 1 basta dividir x entre MRAND_MAX, o
x=UniformRand(Max, Min); // Genera un número real entre Max y Min (ambos inclusive).
```

MersenneTwister

La librería MersenneTwister se basa en el uso de clases, por lo que para usarla hay que definir instancias de clase (cada hilo deberá tener su propia instancia, por lo que, o se crea un vector de tamaño NumHilos y cada hilo accede a una posición diferente usando su identificador, o se debe definir r como privada dentro de la región paralela):

```
MTRand r(semilla); //Se crea la instancia de clase r. Si se desea se puede dar una semilla.
                   //Si no se da una semilla, se usa un valor aleatorio del S. O.
r.rand();//Genera un número real entre 0 y 1 (ambos inclusive).
```

Ejercicios

1. Implementar el cálculo de Pi usando el método de Montecarlo de forma secuencial usando la librería de generación de números aleatorios estándar de C++. El programa recibirá un parámetro indicando el número de iteraciones y medirá el tiempo de ejecución (usar la función de OpenMP para medir tiempos).
2. Modificar el ejercicio 1 para que se ejecute en paralelo usando OpenMP. Este segundo ejercicio recibirá un segundo parámetro con el número de hilos que se lanzarán.
3. Modificar el ejercicio 2 para variar la librería de números aleatorios utilizada. Su usará un tercer parámetro para indicar la librería a usar (1 - rand; 2 - mramd; 3 - MersenneTwister). Estudiar qué ocurre.
4. Modificar el ejercicio 3 para implementar el problema de forma híbrida MPI-OpenMP (los experimentos en este caso se realizarán usando la librería de generación de números aleatorios con mejor resultado del ejercicio 3).

Notas

Para compilar los códigos se usarán los compiladores g++ o mpic++ ya que, tanto la librería mramd como la MersenneTwister, son código c++.
El comando que se ejecutará será el siguiente:

compilador codigo.cpp [mramd.cpp] -o ejecutable [-fopenmp]

donde mramd.cpp es necesario si usamos esa librería y -fopenmp es necesario si usamos openMP.

Para ejecutar el código mpi se usará el siguiente comando:

mpiexec -n NúmeroDeProcesosALanzar -f FicheroConDirecciones ./ejecutable NumIter NumHilos LibRand

donde *FicheroConDirecciones* es un fichero de texto que contendrá las direcciones de los nodos del cluster (direcciones 192.168.0.1 a 192.168.0.4), una en cada línea.

Para que la medida de tiempos sea justa, todas las pruebas deben hacerse con el mismo número de iteraciones. Por ejemplo, si en secuencial se usan 100.000.000 iteraciones, si usamos 4 hilos, cada hilo deberá procesar 25.000.000 iteraciones y, si usamos 4 procesos con 4 hilos cada uno, cada proceso procesará 25.000.000 iteraciones de las cuales cada hilo procesará 6.250.000.

Para hacer las pruebas finales se debe utilizar número alto de iteraciones, por ejemplo 100.000.000 (para comprobar si un ejercicio funciona, basta con probar con unos pocos cientos o miles de iteraciones).

Dado el uso intensivo que se va a realizar del cluster de docencia, hay que tratar, en la medida de lo posible, de coordinarse con el resto de compañeros para que no haya varias ejecuciones a la vez en el mismo nodo. Para ello, durante las ejecuciones secuenciales y las OpenMP estrictas, cada alumno usará un nodo diferente del cluster (usuarios cp01-cp03 al nodo 0, cp04-cp05 al nodo 2, cp06-cp07 al nodo 3 y cp08-cp09 al nodo 4), reduciendo los solapamientos. Para conectarse a un nodo diferente del 0 (el nodo al que nos conectamos remotamente), basta con hacer *ssh node00X*, donde x es un valor entre 2 y 4 (no debe pedirnos contraseña, si lo hace, avisar al profesor).

Para sumar los resultados parciales de todos los hilos/procesos, deben usarse las operaciones de reducción tanto de MPI como de OpenMP.

Las pruebas se harán variando el número de hilos/procesos tal como aparece en el excel de la práctica. Se deben rellenar los campos *Tiempos* y *Número de cores* de cada experimento así como el campo *Conclusiones*, con las conclusiones alcanzadas en función de los resultados obtenidos. Para calcular cuántos cores físicos se están usando, tendremos en cuenta el número de hilos/procesos así como la arquitectura del cluster. Recordemos que el cluster de docencia tiene 4 nodos, cada nodo 2 procesadores, cada procesador 4 cores y cada core tiene Hyperthreading.

Tanto el excel como las librerías de generación de números aleatorios están disponibles en el campus virtual (archivo p2.zip).

Deberán entregarse todos los ficheros de código (uno por cada ejercicio) junto con el excel relleno en un único archivo comprimido a través de la tarea abierta a tal efecto en el Campus Virtual.