Calculo Número Pi

Yeray López Ramírez, C1

1 Introducción

En esta actividad implementamos el cálculo del número pi mediante integración númerica. También compararemos los tiempos de ejecucion de forma secuencial y concurrente:

2 Desarrollo

2.1 La base

El código cuenta esencialmente de 2 partes: La evaluación de la función a integrar que está implementada en la función de double **f** y una función **calcular_integral** que mediante un bucle llama a la función **f** m veces y devuelve la suma de sus valores.

2.2 Problema

Para una mayor precisión, el tamaño de m es absurdamente grande (del orden de mil millones) y eso conlleva un mayor coste computacional que se traduce en más tiempo de ejecución.

2.3 Solución

Para solucionar este problema, recurrimos a las hebras. Estas hebras significan que son capaces de ejecutar un código de forma simultánea en varios núcleos (físicos o virtuales). Para esta práctica usamos un total de 4 hebras que se dividirán m en bloques de m/n para reducir el tiempo de ejecución a un cuarto.

2.4 Implementación

El código implementado es el siguiente:

```
44 // -----------------------------------------------------------------------------
45 // función que ejecuta cada hebra: recibe $i$ ==indice de la hebra, ($0|leq i<n$)
46 double funcion_hebra( long i )
47 {
48    double sumap = 0.0;                        //incializar suma parcial
49    double b = m/n;                            //bloque a ejecutar por cada hebra
50    for(int j = i*b; j < (i+1)*b; j++){        //recorrer m en bloques para cada hebra
51        sumap += f((j+double(0.5))/m);         //calcular x_j
52    }
53    return sumap;                              //devuelver la suma parcial
54 }
55
56 // -----------------------------------------------------------------------------
57 // calculo de la integral de forma concurrente
58 double calcular_integral_concurrente( )
59 {
60    double suma = 0.0;                         //incializar suma total
61    future<double> futuros[n];                 //vector de futuros que contiene las hebras
62
63    for(int i = 0; i < n; i++)                 //lanzar las hebras
64       futuros[i] = async( launch::async, funcion_hebra, i);
65
66    for(int i = 0; i < n; i++)                 //Sumar los valores parciales
67       suma += futuros[i].get();
68
69  return suma/m;                               //devolver valor promedio de f
70 }
71 // -----------------------------------------------------------------------------
```

Figure 1: Podemos ver que el bucle en vez de calcular directamente la suma, lanza 4 hebras que a su vez llaman a f para completar la tarea en 4 veces menos tiempo.

Es importante destacar el uso del vector de futures que automatiza el proceso de join y facilita el valor obtenido a través del método get(). Además, el bucle for del método **funcion_hebra** responde a la formula: $inicio : ih * b \longrightarrow fin : (ih + 1) * b - 1$

2.5 Conclusión

El resultado final es que evidentemente los tiempos de ejecución se reducen como podemos apreciar en la siguiente imagen:

```
yerasito@yerasito-VivoBook-ASUSLaptop-X509DA-D509DA:/media/ye
Número de muestras (m)    : 1073741824
Número de hebras (n)      : 4
Valor de PI               : 3.14159265358979312
Resultado secuencial      : 3.14159265358998185
Resultado concurrente     : 3.14159265358982731
Tiempo secuencial         : 4311.5 milisegundos.
Tiempo concurrente        : 1345.4 milisegundos.
Porcentaje t.conc/t.sec. : 31.2%
yerasito@yerasito-VivoBook-ASUSLaptop-X509DA-D509DA:/media/ye
```

Los tiempos de ejecución son altamente variables ya que según el ordenador y su estado, pueden variar. Si desenchufamos el ordenador de la corriente...

```
yerasito@yerasito-VivoBook-ASUSLaptop-X509DA-D509DA:/media/yerasito
Número de muestras (m)    : 1073741824
Número de hebras (n)      : 4
Valor de PI               : 3.14159265358979312
Resultado secuencial      : 3.14159265358998185
Resultado concurrente     : 3.14159265358982731
Tiempo secuencial         : 6012.8 milisegundos.
Tiempo concurrente        : 1594.4 milisegundos.
Porcentaje t.conc/t.sec. : 26.52%
yerasito@yerasito-VivoBook-ASUSLaptop-X509DA-D509DA:/media/yerasito
```

Figure 2: Los tiempos de ejecución aumentan considerablemente, sobre todo el secuencial

2.6 Observaciones

Si aumentamos el número de hebras a 8, el resultado es una clara reducción del tiempo de ejecución concurrente: $1105ms$ frente los $1594ms$ de antes.
Sin embargo, si seguimos aumentando el número de hebras el tiempo de ejecución es más alto e incluso saltan errores:

```
yerasito@yerasito-VivoBook-ASUSLaptop-X509DA-D509DA:/media/yeras
Violación de segmento (`core' generado)
yerasito@yerasito-VivoBook-ASUSLaptop-X509DA-D509DA:/media/yeras
```

Figure 3: Se produce un "core" con 441324 hebras

Por otro, lado aumentar las muestras no suponen un cambio significativo para el coste computacional que supone.