Aproximación de Pi a través del cálculo numérico de una integral

Salvador Romero Cortés

Introducción

En esta memoria se recoge la actividad del seminario 1. Este ejercicio sirve para practicar las técnicas de programación concurrente en C++11.

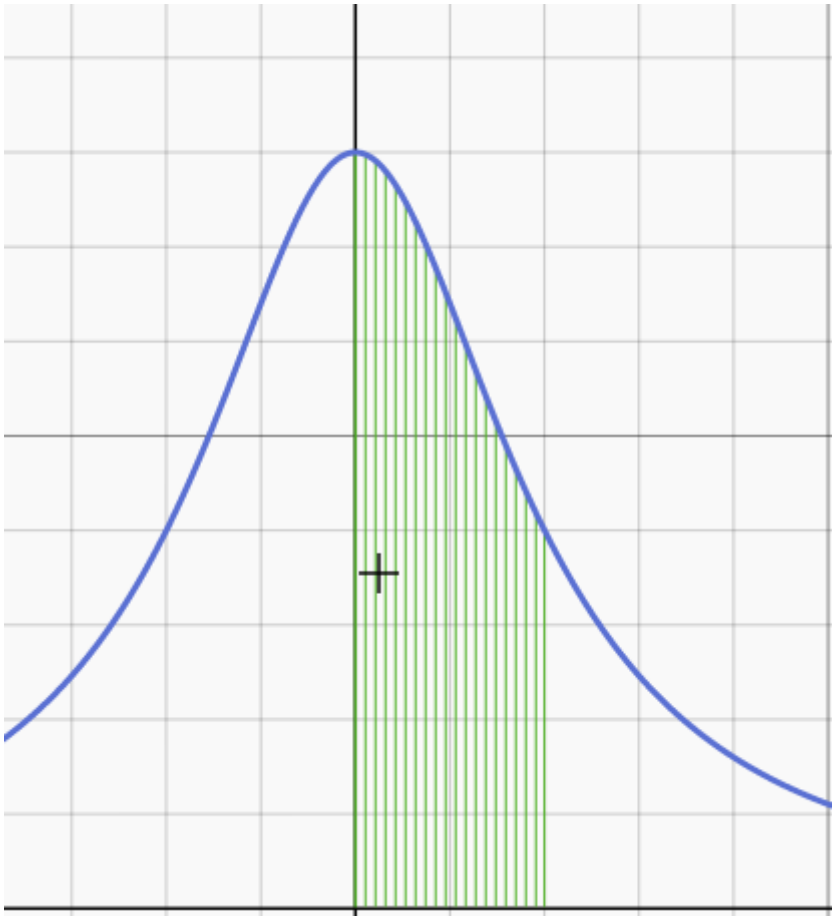
El ejercicio consiste en aproximar el valor del número π a través del cálculo número de la integral definida:

$$I = \int_0^1 \frac{4}{1+x^2}\, dx = \pi$$

Base matemática

El cálculo se puede hacer evaluando la función en m puntos del espacio en el intervalo $[0,1]$ y aproximando I como la media de todos esos valores. De esta manera tenemos que:

$$I \approx \frac{1}{m} \sum_{j=0}^{m-1} f(x_j) \quad : \quad x_j = fracj + 1/2m$$



Gráficamente podemos ver en la imagen el cálculo que vamos a realizar. Para hacer el cálculo concurrente dividiremos m entre el número de hebras que vayamos a usar. De esta manera cada hebra calcula sólo una parte del intervalo $[0,1]$.

Base informática

Vamos a trabajar en C++11, utilizando las utilidades incluidas en `<thread>`. Para obtener el resultado de cada hebra utilizamos un vector de tipo `future` con tantos elementos como `n` (número de hebras a utilizar). Este vector ejecuta una hebra cuando esta se inicializa, así conseguimos que la misma tarea se pueda repartir entre distintos hilos, reduciendo hasta en `n` veces el tiempo que se tardaría en conseguir el resultado, siendo `n` el número de hebras utilizadas. (Para que esto sea necesario, m debe ser varios órdenes de magnitud mayor a `n`).

Desarrollo

De esta manera inicializamos cada elemento del vector con la función:

```
double calcular_integral_secuencial();
```

De esta manera la inicialización queda como:

```
for (int i=0; i < num_hebras; i++)
    futuro[i] = async(launch::async, funcion_hebra, i);
```

La función que ejecuta cada hebra se encarga de realizar el intervalo que se le haya asignado a esa hebra. Este es: `[ih*muestras_por_hebra, ih*muestas_por_hebra+muestras_por_hebra]`. Una vez ajustado el intervalo, el resto del cálculo es trivial puesto que sigue la misma estructura que el cálculo secuencial. Podemos ver que ambas funciones son bastante similares.

Cálculo secuencial

```
double calcular_integral_secuencial(  )
{
   double suma = 0.0 ;                          // inicializar suma
   for( long j = 0 ; j < m ; j++ )              // para cada $j$ entre $0$ y $m-1$:
   {  const double xj = double(j+0.5)/m ;       //      calcular $x_j$
      suma += f( xj );                          //      añadir $f(x_j)$ a la suma
actual
   }
   return suma/m ;                              // devolver valor promedio de $f$
}
```

Cálculo concurrente

```
double funcion_hebra( long ih )
{
    long muestras_por_hebra = m / n;  //calculamos cuantas piezas del intervalo
le corresponden a cada hebra
    double suma = 0.0;                //inicializamos la suma
    for (int i=ih*muestras_por_hebra; i < ih*muestras_por_hebra +
muestras_por_hebra; i++){       //calculamos la suma parcial en el intervalo de
la hebra seleccionada
        double xi = (i+0.5)/m;
        suma += f(xi);
    }
    return suma/m;                    //devolvemos la media de ese intervalo
}
```

Resultados

A continuación se muestran los resultados al ejecutar el programa con 4 hebras y con 6 hebras.

- 4 hebras

```
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        : 3120.9 milisegundos.
Tiempo concurrente       : 890.12 milisegundos.
Porcentaje t.conc/t.sec. : 28.52%
```

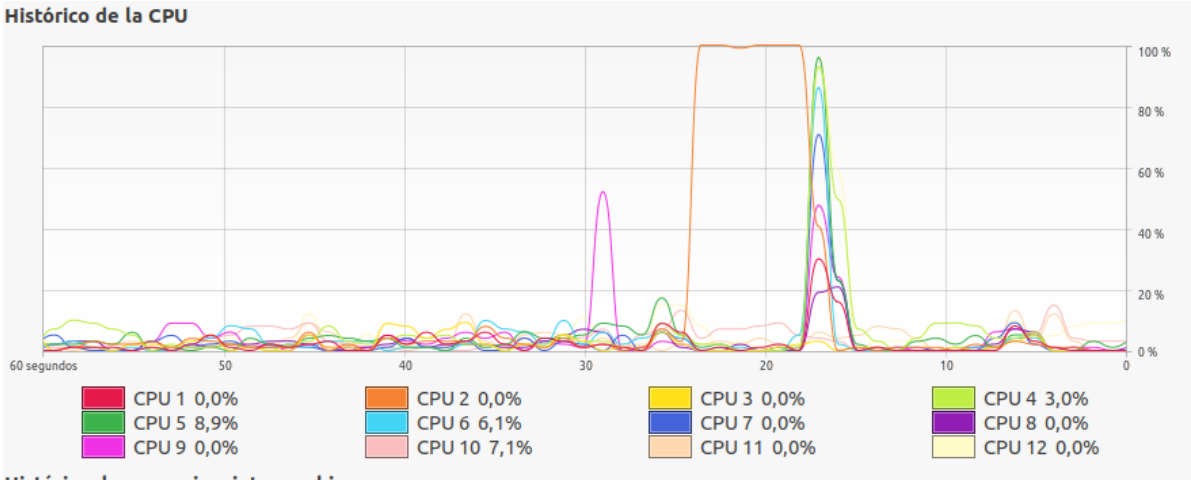
- 6 hebras

```
Número de muestras (m)   : 2147483646
Número de hebras (n)     : 6
Valor de PI              : 3.14159265358979312
Resultado secuencial     : 3.14159265358963546
Resultado concurrente    : 3.14159265358981798
Tiempo secuencial        : 6084.4 milisegundos.
Tiempo concurrente       : 1274.1 milisegundos.
Porcentaje t.conc/t.sec. : 20.94%
```

(En el caso de 6 hebras también se ha aumentado el número de muestras).

Como podemos observar la mejora es bastante notable, tanto en la eficiencia como en la precisión del resultado obtenido. Vemos que se acerca a la hipótesis teórica de una velocidad `n` más rápida. Además también se aprecia que se obtienen unos decimales algo más precisos que el secuencial.

Aquí podemos observar gráficamente el uso de las hebras del procesador cuando ejecutamos el programa con 6 hebras.



Vemos como al principio (cálculo secuencial) hay una única hebra que se encarga de todo el cómputo (CPU 2).

Luego vemos como 6 núcleos trabajan al mismo tiempo y tardan una fracción del tiempo.

Finalmente, observamos que el cálculo concurrente realmente ayuda a la velocidad de cálculos complejos numéricos.