Universidad de Granada

Sistemas concurrenetes y distribuidos

Seminario 1: Cálculo concurrente de π

Autora:
Elena Merelo Molina

Septiembre de 2018

UNIVERSIDAD
DE GRANADA

1 Enunciado de la práctica

Como actividad se propone copiar la plantilla en ejemplo09.cpp y completar en este archivo la implementación del cálculo concurrente del número π . En la salida se presenta el valor exacto de π y el calculado de las tres formas: secuencial, concurrente distribuyendo el trabajo entre hebras de manera contigua, y distribuyéndolo entrelazadamente (sirve para verificar si el programa es correcto). Asimismo, el programa imprimirá la duración del cálculo concurrente contiguo, concurrente entrelazado, secuencial y el porcentaje de tiempo de estos dos últimos respecto del secuencial.

2 Explicación de mi solución

Para resolver el problema he usado cuatro funciones. Veámoslas una a una.

Cálculo de las sumas parciales de forma contigua

• `double funcion_hebra_contigua( long i )` Función que ejecuta cada hebra: recibe el índice de la hebra (i<n) y evalúa f desde el comienzo del intervalo, m/n (lo que viene siendo el tamaño de un chunk) multiplicado por la hebra que es, hasta el fin. De esta manera cada hebra calcula un bloque: hebra 0 el primer intervalo formado por cuatro subintervalos, hebra 1 los siguientes cuatro, y así.

```
1 double funcion_hebra_contigua( long i ) {
2   double suma_hebra= 0.0;
3   for( long j= m/n*i; j< m/n*(i+1); j++)
4     suma_hebra += f((j + double(0.5)) /m);
5
6   return suma_hebra/m;
7 }
```

Cálculo de las sumas parciales de forma entrelazada

• `double funcion_hebra_entrelazada( long i )` Función que ejecuta cada hebra: recibe i==índice de la hebra (i<n) y evalúa f desde el comienzo del subintervalo, i, hasta m, el final, dando saltos de n en n, el número de hebras. Consecuentemente, cada hebra calcula un subintervalo: hebra 1 el primero, hebra 2 el segundo,... hebra 1 el quinto, y así sucesivamente.

```
1 double funcion_hebra_entrelazada( long i ) {
2   double suma_hebra= 0.0;
3   for( long j= i; j< m ; j += n)
4     suma_hebra += f((j + double(0.5)) /m);;
5
6   return suma_hebra/m;
7 }
```

Cálculo concurrente de π de forma contigua

• `double calcular_integral_concurrente_contigua( )` Calcula la integral de forma concurrente. Una vez las hebras han calculado las sumas parciales de los valores de f, la hebra principal las recoge en una suma total, repartiendo los bloques entre las hebras de manera contigua.

```
double calcular_integral_concurrente_contigua( ) {
  double suma= 0.0;
  future<double> futuros[n];

  //Ponemos en marcha todas las hebras y obtenemos los futuros
  for( long i= 0; i< n; i++)
    futuros[i] = async( launch::async, funcion_hebra_contigua, i);
  //Esperamos a que cada hebra termine y vamos obteneniendo el resultado de la
   integral
  for(long i= 0; i<n; i++)
    suma += futuros[i].get();

  return suma;
}
```

Cálculo concurrente de π de forma entrelazada

• `double calcular_integral_concurrente_entrelazada( )` Calcula la integral de forma concurrente. Una vez las hebras han calculado las sumas parciales de los valores de f, la hebra principal las recoge en una suma total, repartiendo los bloques entre las hebras de manera entrelazada.

```
double calcular_integral_concurrente_entrelazada( ) {
  double suma= 0.0;
  future<double> futuros[n];

  //Ponemos en marcha todas las hebras y obtenemos los futuros
  for( long i= 0; i< n; i++)
    futuros[i] = async( launch::async, funcion_hebra_entrelazada, i);
  //Esperamos a que cada hebra termine y vamos obteneniendo el resultado de la
   integral
  for(long i= 0; i<n; i++)
    suma += futuros[i].get();

  return suma;
}
```

3 Medición de tiempos

De ello se ocupa el programa principal:

```
int main() {
```

```
  time_point<steady_clock> inicio_sec  = steady_clock::now() ;
  const double             result_sec  = calcular_integral_secuencial(  );
  time_point<steady_clock> fin_sec     = steady_clock::now() ;

  double x = sin(0.4567);
  time_point<steady_clock> inicio_conc_ent = steady_clock::now() ;
  const double             result_conc_ent =
    calcular_integral_concurrente_entrelazada(  );
  time_point<steady_clock> fin_conc_ent    = steady_clock::now() ;

  time_point<steady_clock> inicio_conc_cont = steady_clock::now() ;
  const double             result_conc_cont =
    calcular_integral_concurrente_contigua(  );
  time_point<steady_clock> fin_conc_cont   = steady_clock::now() ;

  duration<float,milli>    tiempo_sec  = fin_sec  - inicio_sec ,
                           tiempo_conc_ent = fin_conc_ent - inicio_conc_ent,
                           tiempo_conc_cont= fin_conc_cont - inicio_conc_cont;
  const float              porc_ent        = 100.0*tiempo_conc_ent.count()/
    tiempo_sec.count(),
                           porc_cont       = 100.0*tiempo_conc_cont.count()/
    tiempo_sec.count();


  constexpr double pi = 3.14159265358979323846l ;

  cout << "NÃºmero de muestras (m)   : " << m << endl
       << "NÃºmero de hebras (n)     : " << n << endl
       << setprecision(18)
       << "Valor de PI              : " << pi << endl
       << "Resultado secuencial     : " << result_sec  << endl
       << "Resultado concurrente entrelazada   : " << result_conc_ent << endl
       << "Resultado concurrente contigua   : " << result_conc_cont << endl
       << setprecision(5)
       << "Tiempo secuencial        : " << tiempo_sec.count()  << " milisegundos. "
    << endl
       << "Tiempo concurrente entrelazada      : " << tiempo_conc_ent.count() << "
    milisegundos. " << endl
       << "Tiempo concurrente contigua      : " << tiempo_conc_cont.count() << "
    milisegundos. " << endl
       << setprecision(4)
       << "Porcentaje t.conc_entrelazada/t.sec. : " << porc_ent << "%" << endl
       << "Porcentaje t.conc_contigua/t.sec. : " << porc_cont << "%" << endl;
}
```

Al ejecutarlo, resulta:

```
 2018-09-26 09:18:15  elena in ~/Escritorio/University stuff/3º/1ºCuatrimestre
/SCD/Prácticas/Seminario1/scd-s1-fuentes
○ → g++ -std=c++11 -pthread ejemplo09-plantilla.cpp -o ejemplo09

 2018-09-26 09:19:42  elena in ~/Escritorio/University stuff/3º/1ºCuatrimestre
/SCD/Prácticas/Seminario1/scd-s1-fuentes
○ → ./ejemplo09
Número de muestras (m)    : 1073741824
Número de hebras (n)      : 4
Valor de PI               : 3.14159265358979312
Resultado secuencial      : 3.14159265358998185
Resultado concurrente entrelazada    : 3.14159265358978601
Resultado concurrente contigua    : 3.14159265358982731
Tiempo secuencial         : 24636 milisegundos.
Tiempo concurrente entrelazada       : 7426.3 milisegundos.
Tiempo concurrente contigua       : 6608.2 milisegundos.
Porcentaje t.conc_entrelazada/t.sec. : 30.14%
Porcentaje t.conc_contigua/t.sec. : 26.82%
```

Figure 1: Resultado de la ejecución

Observamos cómo el cálculo concurrente entrelazado produce un número π más cercano al real, seguido del contiguo y por último el secuencial, que es más inexacto. No obstante, el cálculo contiguo tarda menos que el entrelazado, y como cabía esperar el secuencial es el más lento.

En la siguiente imagen vemos (en un sistema Ubuntu 16 con 4 CPUs) cómo van evolucionando los porcentajes de uso de cada CPU a lo largo de la ejecución del programa con 4 hebras (es una captura de pantalla del system monitor):

Como bien se puede observar en las imágenes, al principio las CPUs están desocupadas, salvo la correspondiente a la hebra principal, en color azul, que ejecuta la versión secuencial en primer lugar, ocupando una CPU al cien por cien. Posteriormente, las cuatro hebras creadas por la principal pasan a ocupar cada una una CPU al 100%, y la principal les espera, uniéndose una vez finalizado el cálculo.

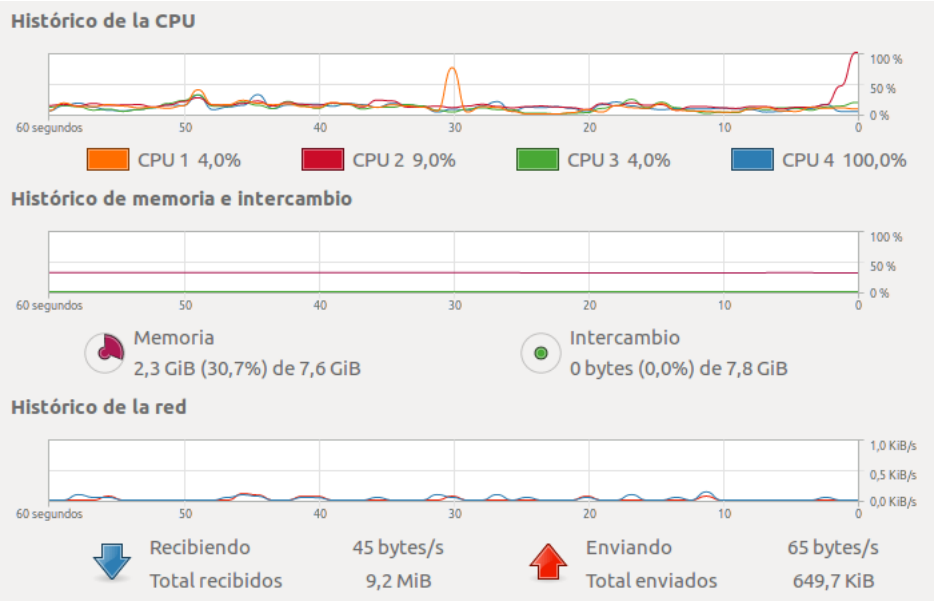

Figure 2: Inicio de la ejecución

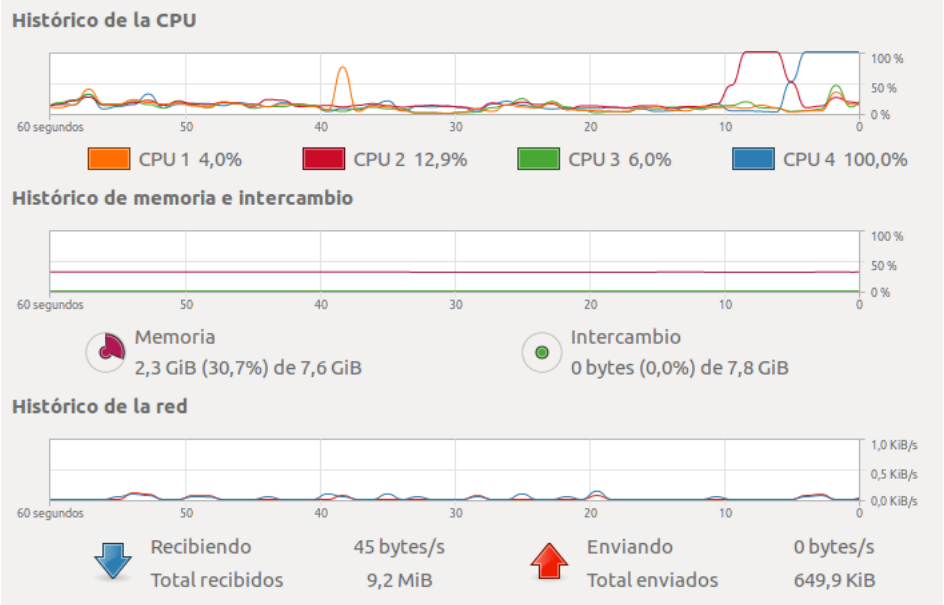

Figure 3: Comienza parte secuencial

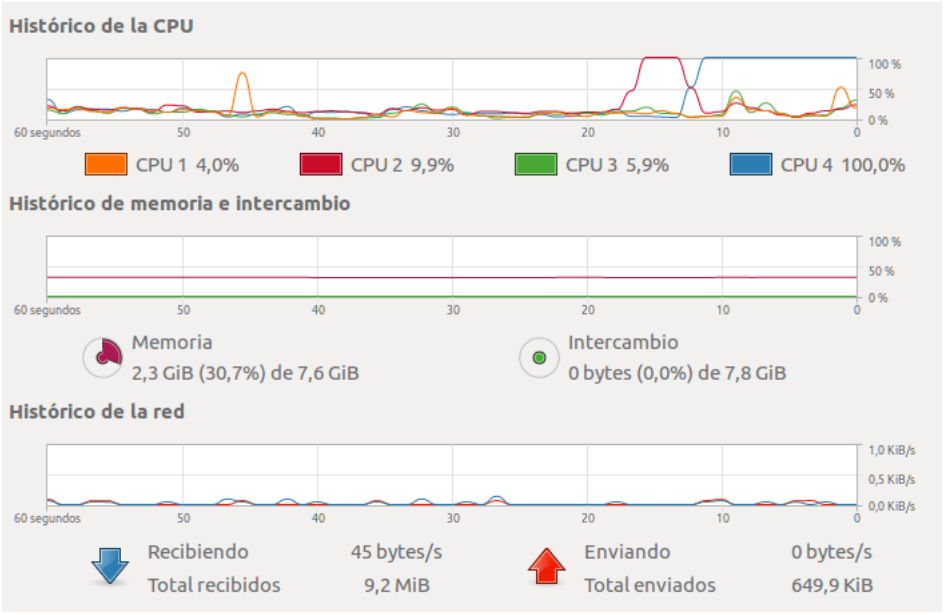

Figure 4: Continúa parte secuencial

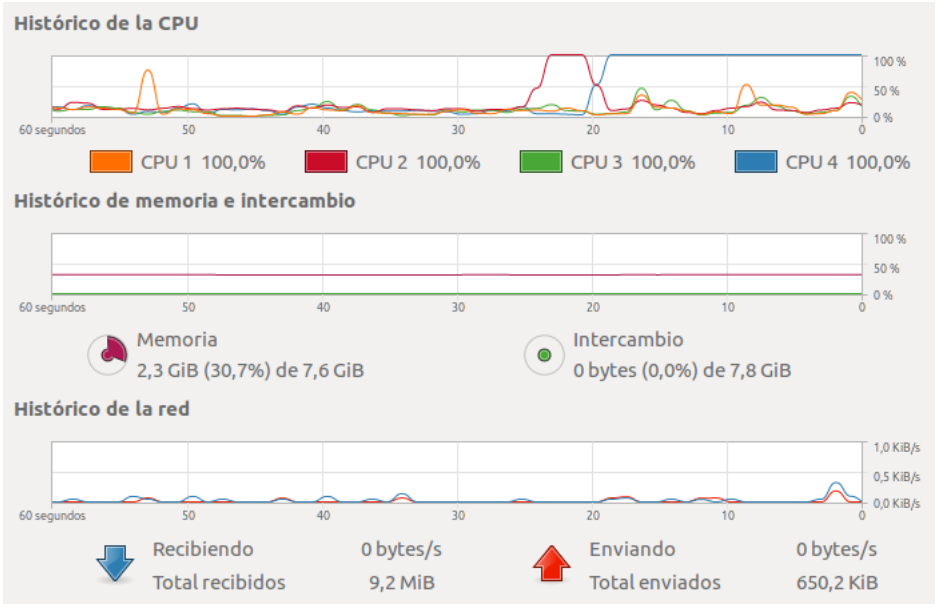

Figure 5: Inicio parte concurrente

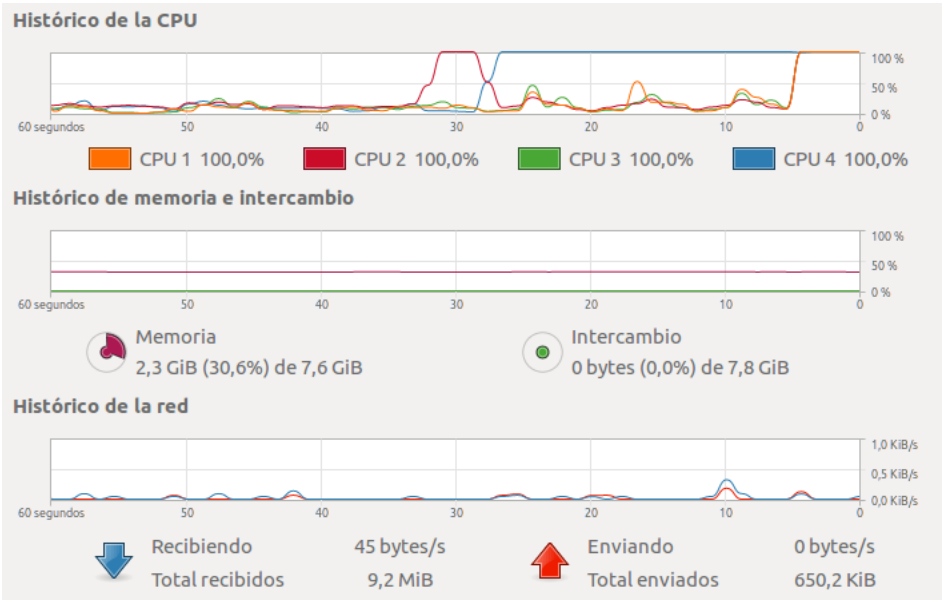

Figure 6: Parte concurrente

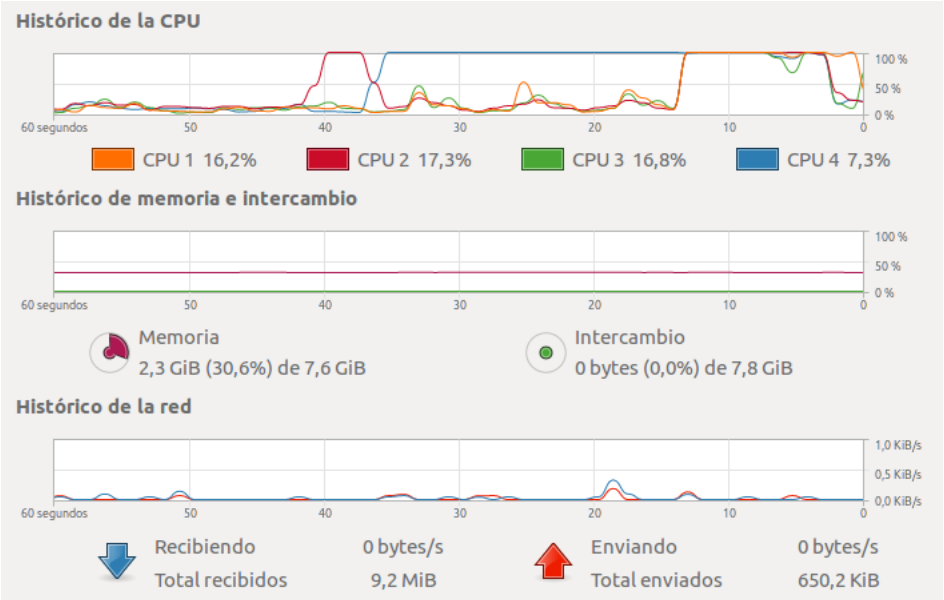

Figure 7: Finalización

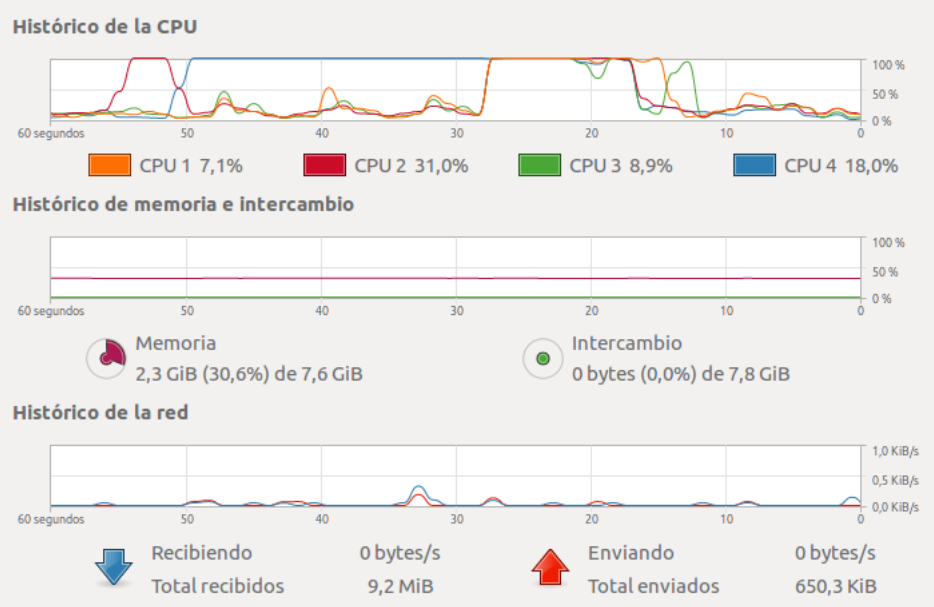

Figure 8: Histórico de lo que han hecho todas las hebras