Arquitectura de Computadors



Víctor Colomé

Josep Sugrañes

En esta práctica se nos pide paralelizar un código iterativo que calcula el producto escalar de dos matrices y analizar el resultado de la ejecución en las distintos computadores, el gat y el cos. Cada una tiene sus características propias que seguidamente exponemos.

GAT

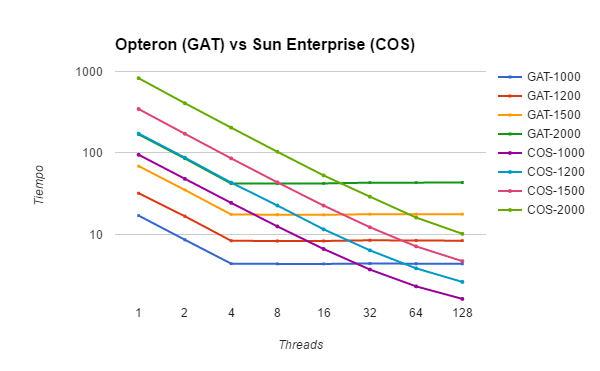
La máquina multiprocesador tiene 4 procesadores unicore que se comunican entre ellos mediante link directos y acceden a memoria de 4GB mediante un FSB. En concreto se tratan de un 4x Opteron 844 a 1.8Ghz.

COS

La máquina multithread tiene 2 procesadores con 8 cores y 8 threads cada uno. Así la máquina tiene una capacidad de cálculo simultánea de 128 threads. La memoria es de 4GB. Se trata de un Sun Enterprise T540 a 1.4 Ghz.

Para conocer el Speed-Up se utiliza la ley de Amdhal, la formula consiste en dividir el tiempo de ejecución del código secuencial que más ha tardado para cada dimensión de la matriz entre el paralelizado.

1. Se mostrará una gráfica con el tiempo de ejecución real para cada uno de los tamaños del problema. Donde en el eje de las Y estará el tiempo de ejecución con escala lineal o logarítmica (según se aprecien mejor los valores) y en el eje de las X las diferentes ejecuciones que se han hecho variando el número de thread

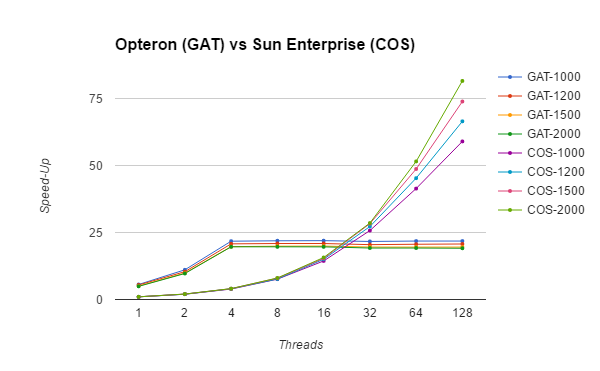


Con la gráfica podemos observar que el computador GAT al tener 4 núcleos de un thread, al principio es muy rápido y a partir de los 4 threads se mantiene constante (aunque todo y eso tarda o debería tardar un poco más ya que pierde tiempo en crear más threads y por el swap de estos).

En cambio, el computador COS con 2 CPUs y un total de 128 threads, al principio es lento ya que todo y tener más threads no es tan potente como el GAT que va a 1.8 GHz respecto los 1.4GHz del COS.

A medida que vamos creando más threads va tardando menos y al llegar a 128 es cuando alcanza su máximo. A partir de ahí se tendría que mantener constante o incluso tardar un poco más por crear threads de más y sus respectivos swappings.

1. Del mismo modo se mostrará otra gráfica con el speedup relativo a la versión secuencial más lenta. En este caso la versión secuencial del i7, donde en el eje de las Y estará el speedup y en el eje de las X las diferentes ejecuciones que se han hecho variando el número de threads.



Como se ha comentado anteriormente, el COS al ser menos potente tarda más en ejecutarse el código secuencial. Por tanto, estos son el speed-up de referencia (1) para cada dimensión de la matriz.

Esta gráfica muestra de forma clara cómo es la relación entre el índice de mejora y los threads. Para calcular el Speed-Up, tenemos que dividir el tiempo del código secuencial entre el tiempo del código paralelo, esto da como resultado el índice de mejora del Speed-Up.

Podemos observar como el computador GOS, al llegar a los 4 threads no mejora, se mantiene estable e incluso decrementa levemente. Este comportamiento es normal ya que, como hemos comentado anteriormente, al tener 4 nucleos no podrá ejecutar más de 4 threads a la vez. En cambio el COS al tener 128 threads puede hacer frente a todos los threads creados.

### Descripción de la tarea

Se nos daba un algoritmo secuencial que calculaba el producto escalar de dos matrices, y nos pedía paralelizarlo con threads con tal de aumentar su rendimiento distribuyendo el trabajo entre ellos.

Decidimos que cada thread calculase un número determinado de casillas de la matriz resultante (B), dependiendo de la dimensión de las matrices A y B y el número de threads; como podemos ver a continuación.

float numCeldasf = (float)(nn\*nn)/(float)numThreads;//celdas X Thread  
 int numCeldas = numCeldasf; //parte entera  
 float resto = numCeldasf - numCeldas; //parte decimal  
  
 if(resto != 0.00){  
 resto = resto\*numThreads;   
 for(i=0; i<resto; i++){  
 rangos[i] = rangos[i] + 1;  
 }  
 }  
  
 rangos[0] = rangos[0] + numCeldas;  
 for(i=1; i<numThreads; i++){  
 rangos[i] += rangos[i-1] + numCeldas;  
 }

Una vez calculada la cantidad que debe ejecutar cada thread, éste tendrá que saber cuál le corresponde mediante su índice de thread. Esto se puede ver a continuación.

void \* parallel\_code(void \* indexp){

int i, k, ini, fil, col;

int index = (uintptr\_t) indexp;

if(index == 0)

ini = 0;

else

ini = rangos[index-1];

for(i=ini; i<rangos[index]; i++){

fil = i/nn;

col = i%nn;

for(k=0; k<nn; k++)

C[fil][col] += A[fil][k] \* B[k][col];

}

pthread\_exit(0);

}

Los threads los creamos en un bucle así:

for(index = 0; index < numThreads; index++)  
 {  
 result\_code = pthread\_create(&threads[index], NULL, parallel\_code, (void \*) (uintptr\_t) index);  
 assert(!result\_code);  
 }

Y seguidamente esperamos la finalización de éstos con un bucle de pthread\_join’s.

for(index = 0; index < numThreads; index++)

{

// block until thread 'index' completes

result\_code = pthread\_join(threads[index], NULL );

assert(!result\_code);

}

5. Durante el proceso de paralelización estudiad mediante la función clock() el tiempo de ejecución de cada bucle. Aplicad la ley de Amdhal.

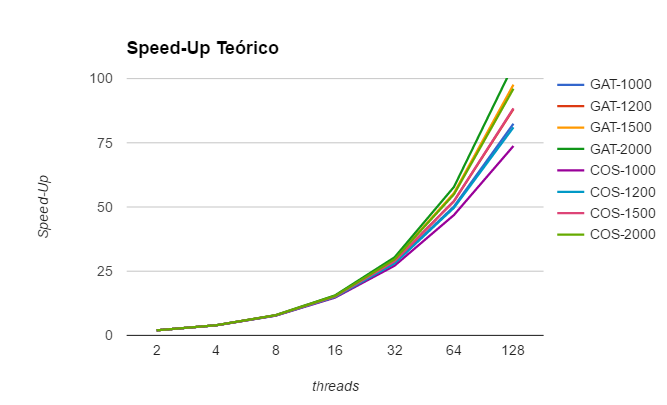
La Ley de Amdhal consiste en calcular el límite teórico del Speed-up sin tener en cuenta los costes de comunicación.

Sn *=*

Sn = Speed-Up según ‘n’ números de threads.

F = Tiempo Secuencial dividido entre Tiempo total.

n = número threads.



Como se ha comentado antes, no se tiene en cuenta el hardware y por tanto los valores teóricos tienen un valor superior que los reales. También hay que tener en cuenta que en los reales se comparan los valores con los secuenciales del COS y ahora se comparan individualmente con su configuración en su máquina.