2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 1. Programación paralela I: Directivas OpenMP

Estudiante (nombre y apellidos): Raúl Castro Moreno
Grupo de prácticas y profesor de prácticas: B2
Fecha de entrega:
Fecha evaluación en clase:

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

Ejercicios basados en los ejemplos del seminario práctico

1. Usar la directiva `parallel` combinada con directivas de trabajo compartido en los ejemplos `bucle-for.c` y `sections.c` del seminario. Incorporar el código fuente resultante al cuaderno de prácticas.

RESPUESTA: Captura que muestre el código fuente `bucle-forModificado.c`

```c
//gcc -fopenmp -O2 bucle-for.c -o bucle-for

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main(int argc, char **argv) {

        int i, n = 9;

        if(argc < 2) {
                fprintf(stderr,"\n[ERROR] - Falta no iteraciones \n");
                exit(-1);
        }

        n = atoi(argv[1]);

        #pragma omp parallel for

        for (i=0; i<n; i++)
                printf("thread %d ejecuta la iteración %d del bucle\n", omp_get_thread_num(),i);

        return(0);
}
```

RESPUESTA: Captura que muestre el código fuente `sectionsModificado.c`

```c
//gcc -fopenmp -O2 sections.c -o sections

#include <stdio.h>
#include <omp.h>
void funcA() {
        printf("En funcA: esta sección la ejecuta el thread %d\n", omp_get_thread_num());
}

void funcB() {
        printf("En funcB: esta sección la ejecuta el thread %d\n", omp_get_thread_num());
}

int main() {
        #pragma omp parallel sections
        {
                #pragma omp section
                        (void) funcA();

                #pragma omp section
                        (void) funcB();
        }
        //return 0;
}
```

2. Imprimir los resultados del programa `single.c` usando una directiva `single` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `single` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `single`. Incorpore en su cuaderno de trabajo el código fuente y volcados de pantalla con los resultados de ejecución obtenidos.

RESPUESTA: Captura que muestre el código fuente `singleModificado.c`

CAPTURAS DE PANTALLA:

```c
//gcc -fopenmp -O2 single.c -o single

#include <stdio.h>
#include <omp.h>
int main() {

        int n = 9, i, a, b[n];

        for (i=0; i<n; i++)
                b[i] = -1;

        #pragma omp parallel
        {
                #pragma omp single
                {
                        printf("Introduce valor de inicialización a: ");
                        scanf("%d", &a );
                        printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
                }

                #pragma omp for
                        for (i=0; i<n; i++)
                                b[i] = a;
                #pragma omp single
                {
                        for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
                        printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
                }

        }
}
```

3. Imprimir los resultados del programa `single.c` usando una directiva `master` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `master` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `master`. Incorpore en su cuaderno el código fuente y volcados de pantalla con los resultados de ejecución obtenidos. ¿Qué diferencia observa con respecto a los resultados de ejecución del ejercicio anterior?

   RESPUESTA: Captura que muestre el código fuente `singleModificado2.c`

CAPTURAS DE PANTALLA:

```c
//gcc -fopenmp -O2 single.c -o single

#include <stdio.h>
#include <omp.h>
int main() {

        int n = 9, i, a, b[n];

        for (i=0; i<n; i++)
                b[i] = -1;

        #pragma omp parallel
        {
                #pragma omp single
                {
                        printf("Introduce valor de inicialización a: ");
                        scanf("%d", &a );
                        printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
                }

                #pragma omp for
                        for (i=0; i<n; i++)
                                b[i] = a;
                #pragma omp master
                {
                        for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
                        printf("Master ejecutada por el thread %d\n", omp_get_thread_num());
                }

        }
}
```

RESPUESTA A LA PREGUNTA:

Con la directiva single la thread que ejecuta la parte de imprimir los resultados del programa, puede ser cualquiera mientras que con la directiva master, siempre lo ejecuta la thread 0.

4. ¿Por qué si se elimina directiva `barrier` en el ejemplo `master.c` la suma que se calcula e imprime no siempre es correcta? Responda razonadamente.

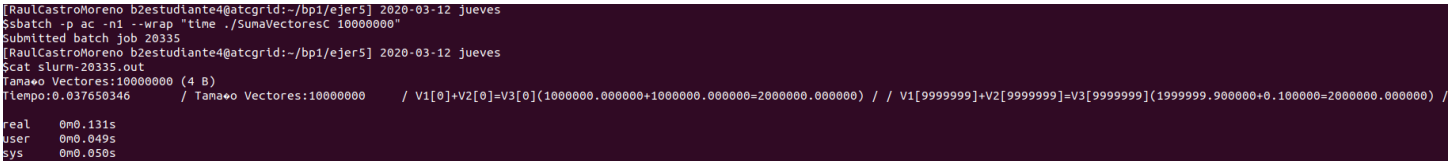
   RESPUESTA:

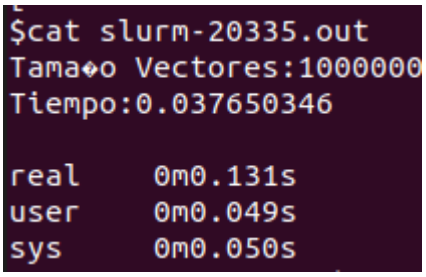
   Esto es debido a que las thread que terminan de realizar la suma , avanzan antes de que terminen todas de realizar su operacion, y en este caso, si la 0 (master) que es la que muestra el resultado final, avanza antes que las demás, imprime el resultado sin tener en cuenta las thread que no han sido mas rapidas que la 0 en realizar el proceso de la suma.

Resto de ejercicios

5. El programa secuencial C del Listado 1 calcula la suma de dos vectores (v3 = v1 + v2; v3(i) = v1(i) + v2(i), i=0,…N-1). Generar el ejecutable del programa del Listado 1 para **vectores globales**. Usar `time` (Lección 3/ Tema 1) en la línea de comandos para obtener, en atcgrid, el tiempo de ejecución (*elapsed time*) y el tiempo de CPU del usuario y del sistema generado. Obtenga los tiempos para vectores con 10000000 componentes. ¿La suma de los tiempos de CPU del usuario y del sistema es menor, mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

   CAPTURAS DE PANTALLA:





   49 + 50 = 99 s

   131s (real) > 99s (CPU time)

   La suma de los tiempos de CPU del usuario y del sistema es MENOR que el tiempo real. Esto es debido a que solo tenemos un flujo de control, es decir , el programa es secuencial.

6. Generar el código ensamblador a partir del programa secuencial C del Listado 1 para **vectores globales** (para generar el código ensamblador tiene que compilar usando `-S` en lugar de `-o`). Utilice el fichero con el código fuente ensamblador generado y el fichero ejecutable generado en el ejercicio 5 para obtener para atcgrid los MIPS (*Millions of Instructions Per Second*) y los MFLOPS (*Millions of FLOating-point Per Second*) del código que obtiene la suma de vectores (código entre las funciones `clock_gettime()`); el cálculo se debe hacer para 10 y 10000000 componentes en los vectores (consulte la Lección 3/Tema1 AC). Razonar cómo se han obtenido los valores que se necesitan para calcular los MIPS y MFLOPS. Incorpore **el código ensamblador de la parte de la suma de vectores** en el cuaderno.

CAPTURAS DE PANTALLA (que muestren la generación del código ensamblador y del código ejecutable, y la obtención de los tiempos de ejecución):

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp1/ejer6] 2020-03-21 sábado
$gcc -S SumaVectoresC.c
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp1/ejer6] 2020-03-21 sábado
$gcc -o SumaVectoresC SumaVectoresC.c
```

```
[RaulCastroMoreno b2estudiante4@atcgrid:~/bp1/ejer6] 2020-03-12 jueves
$sbatch -p ac -n1 --wrap "./SumaVectoresC 10"
Submitted batch job 20412
[RaulCastroMoreno b2estudiante4@atcgrid:~/bp1/ejer6] 2020-03-12 jueves
$sbatch -p ac -n1 --wrap "./SumaVectoresC 10000000"
Submitted batch job 20413
[RaulCastroMoreno b2estudiante4@atcgrid:~/bp1/ejer6] 2020-03-12 jueves
$cat slurm-20412.out
Tama◆o Vectores:10 (4 B)
Tiempo:0.000000197       / Tama◆o Vectores:10   / V1[0]+V2[0]=V3[0](1.00
[RaulCastroMoreno b2estudiante4@atcgrid:~/bp1/ejer6] 2020-03-12 jueves
$cat slurm-20413.out
Tama◆o Vectores:10000000 (4 B)
Tiempo:0.038437352       / Tama◆o Vectores:10000000     / V1[0]+V2[0]=V3
[RaulCastroMoreno b2estudiante4@atcgrid:~/bp1/ejer6] 2020-03-12 jueves
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

MIPS = Número de Instrucciones/ Tiempo de ejecución*10^6

Por el ensamblador obtenemos que se realizan en el bucle 6 instrucciones, y fuera de el, otras 3 instrucciones. Los tiempos de ejecución los tenemos en las capturas de encima, obtenidas en atcgrid.

Para 10 :

MIPS = (6*10 + 3) / (0.000000197*10^6) = 319.7969543

Para 10000000:

MIPS = (6*10000000 + 3) / (0.038437352*10^6) = 1560.981698

MFLOPS = Número de Operaciones en coma flotante / Tiempo de ejecución*10^6

Tambien obtenemos, que solo se realizan 3 instrucciones, y son dentro del bucle. Los tiempos de ejecución los obtenemos tambien de las capturas.

Para 10 :

MFLOPS = (3*10) / (0.000000197*10^6) = 152.284264

Para 10000000:

MFLOPS = (3*10000000) / (0.038437352*10^6) = 780.4908101

RESPUESTA: Captura que muesre el código ensamblador generado de la parte de la suma de vectores

```
        call    clock_gettime@PLT
        xorl    %eax, %eax
        .p2align 4,,10
        .p2align 3
.L9:
        movsd   0(%rbp,%rax), %xmm0
        addsd   (%r14,%rax), %xmm0
        movsd   %xmm0, (%r12,%rax)
        addq    $8, %rax
        cmpq    %r15, %rax
        jne     .L9
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
```

7. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores (v3 = v1 + v2; v3(i)=v1(i)+v2(i), i=0,…N-1) usando las directivas `parallel` y `for`. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Como en el código del Listado 1 se debe obtener el tiempo (*elapsed time*) que supone el cálculo de la suma. Para obtener este tiempo usar la función `omp_get_wtime()`, que proporciona el estándar OpenMP, en lugar de `clock_gettime()`. NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para varios tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de v1, v2 y v3 (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

RESPUESTA: Captura que muestre el código fuente implementado

```
  //Inicializar vectores
#pragma omp parallel for
  for(i=0; i<N; i++){
    v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
  }

 double t1 = omp_get_wtime();
  //Calcular suma de vectores
#pragma omp parallel for
  for(i=0; i<N; i++) {
    v3[i] = v1[i] + v2[i];
}

 double t2 = omp_get_wtime();
 double elapsed = t2-t1;
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA (compilación y ejecución para N=8 y N=11):

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp1/ejer7] 2020-03-21 sábado
$gcc -fopenmp -o SumaVecOmpFor SumaVecOmpFor.c
```

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp1/ejer7] 2020-03-21 sábado
$./SumaVecOmpFor 8
Tama◆o Vectores:8 (4 B)
Tiempo:0.001272138       / Tama◆o Vectores:8
/ V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) /
/ V1[1]+V2[1]=V3[1](0.900000+0.700000=1.600000) /
/ V1[2]+V2[2]=V3[2](1.000000+0.600000=1.600000) /
/ V1[3]+V2[3]=V3[3](1.100000+0.500000=1.600000) /
/ V1[4]+V2[4]=V3[4](1.200000+0.400000=1.600000) /
/ V1[5]+V2[5]=V3[5](1.300000+0.300000=1.600000) /
/ V1[6]+V2[6]=V3[6](1.400000+0.200000=1.600000) /
/ V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp1/ejer7] 2020-03-21 sábado
$./SumaVecOmpFor 11
Tama◆o Vectores:11 (4 B)
Tiempo:0.001848551       / Tama◆o Vectores:11   / V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) / / V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
```

8. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores usando las `parallel` y `sections/section` (se debe aprovechar el paralelismo de datos usando estas directivas en lugar de la directiva `for`); es decir, hay que repartir el trabajo (tareas) entre varios threads usando `sections/section`. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Para obtener este tiempo usar la función `omp_get_wtime()` en lugar de `clock_gettime()`. NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para tamaños pequeños de los vectores (por ejemplo, N = 8); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de v1, v2 y v3 (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

RESPUESTA: Captura que muestre el código fuente implementado

```
  //Inicializar vectores
#pragma omp parallel sections
{
  #pragma omp section
  for(i=0; i<N; i++){
    v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
   }
  #pragma omp section
  for(i=0; i<N; i++){
    v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
   }
  #pragma omp section
  for(i=0; i<N; i++){
    v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
   }
  #pragma omp section
  for(i=0; i<N; i++){
    v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
   }
}

 double t1 = omp_get_wtime();
  //Calcular suma de vectores
#pragma omp parallel sections
{
  #pragma omp section
  for(i=0; i<N; i++)
    v3[i] = v1[i] + v2[i];
  #pragma omp section
  for(i=0; i<N; i++)
    v3[i] = v1[i] + v2[i];
  #pragma omp section
  for(i=0; i<N; i++)
    v3[i] = v1[i] + v2[i];
  #pragma omp section
  for(i=0; i<N; i++)
    v3[i] = v1[i] + v2[i];
}

 double t2 = omp_get_wtime();
 double elapsed = t2-t1;
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA (compilación y ejecución para N=8 y N=11):

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp1/ejer8] 2020-03-21 sábado
$gcc -fopenmp -o SumaVecOmpSec SumaVecOmpSec.c
```

```
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp1/ejer8] 2020-03-21 sábado
$./SumaVecOmpSec 8
Tama◆o Vectores:8 (4 B)
Tiempo:0.002013648       / Tama◆o Vectores:8
/ V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) /
/ V1[1]+V2[1]=V3[1](0.900000+0.700000=1.600000) /
/ V1[2]+V2[2]=V3[2](1.000000+0.600000=1.600000) /
/ V1[3]+V2[3]=V3[3](1.100000+0.500000=1.600000) /
/ V1[4]+V2[4]=V3[4](1.200000+0.400000=1.600000) /
/ V1[5]+V2[5]=V3[5](1.300000+0.300000=1.600000) /
/ V1[6]+V2[6]=V3[6](1.400000+0.200000=1.600000) /
/ V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
[RaulCastroMoreno raul@raul-OMEN-by-HP-Laptop-15-ce0xx:~/Escritorio/Home/AC/bp1/ejer8] 2020-03-21 sábado
$./SumaVecOmpSec 11
Tama◆o Vectores:11 (4 B)
Tiempo:0.001847545       / Tama◆o Vectores:11   / V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) / / V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
```

9. ¿Cuántos threads y cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 7? Razone su respuesta. ¿Cuántos threads y cuantos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta.

RESPUESTA:

En el ejercico 7, la versión puede usar tantos threads y cores como la máquina le permita, ya que no hemos especificado cuantos mediante OMP_NUM_THREADS.

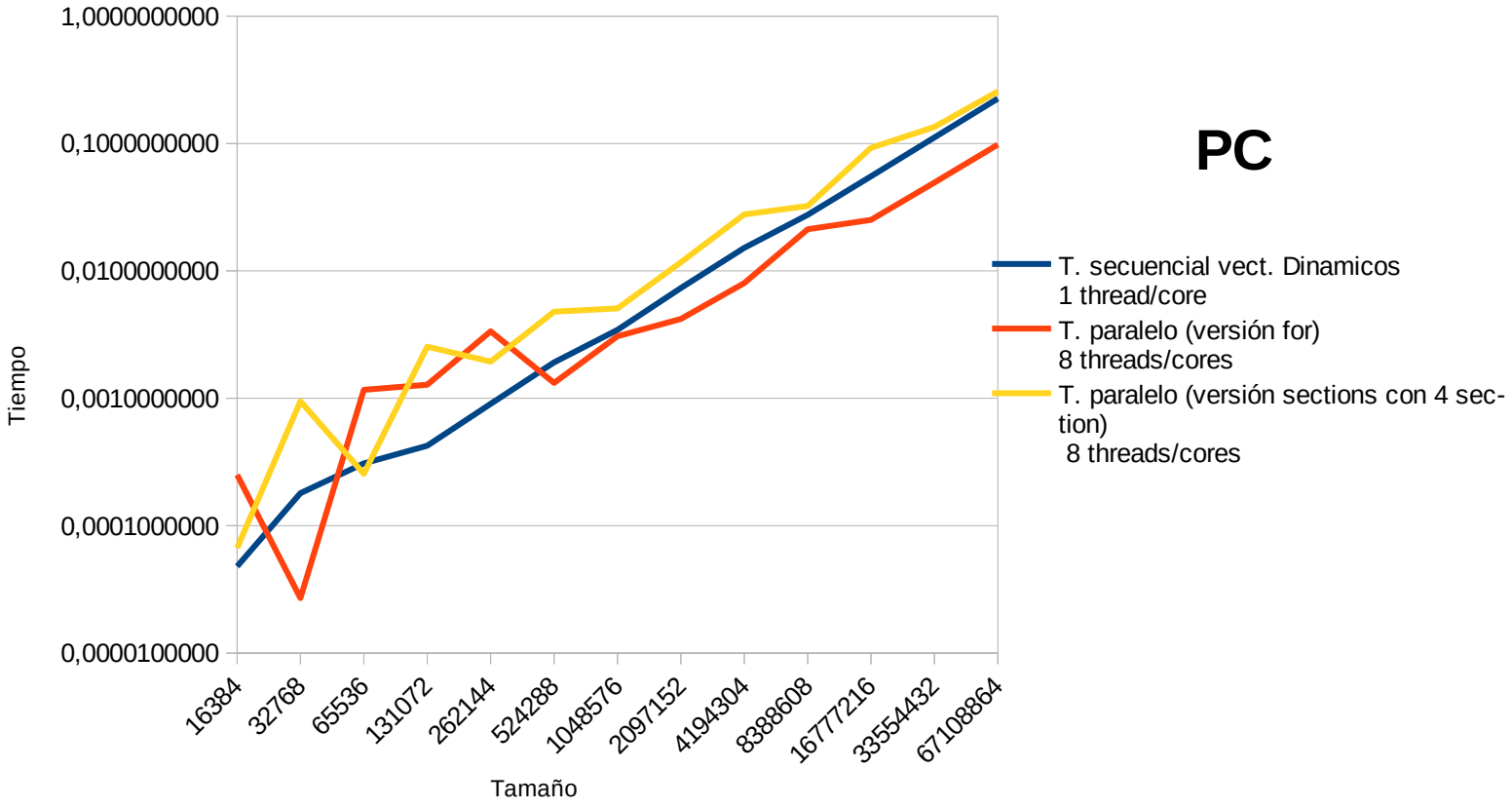
En el ejercicio 8 pasa de forma similar que en el 7 , pero al tener la directiva sections, si tenemos un menor numero de section que de threads y cores tenga la maquina, tendremos entonces threads y cores que no realizen nada mientras este la ejecución dentro de esa directiva.

10. Rellenar una tabla como la Tabla 2Error: no se encontró el origen de la referencia para atcgrid y otra para su PC con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución del trozo de código que realiza la suma en paralelo (este es el tiempo que deben imprimir los programas). Ponga en la tabla el número de threads/cores que usan los códigos (use el máximo número de cores físicos del computador que como máximo puede aprovechar el código, no use un número de threads superior al número de cores físicos). Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute código que imprima todos los componentes del resultado cuando este número sea elevado.

RESPUESTA: Mi PC tiene un Intel Core i7-7700HQ @ 2.80GHz*8

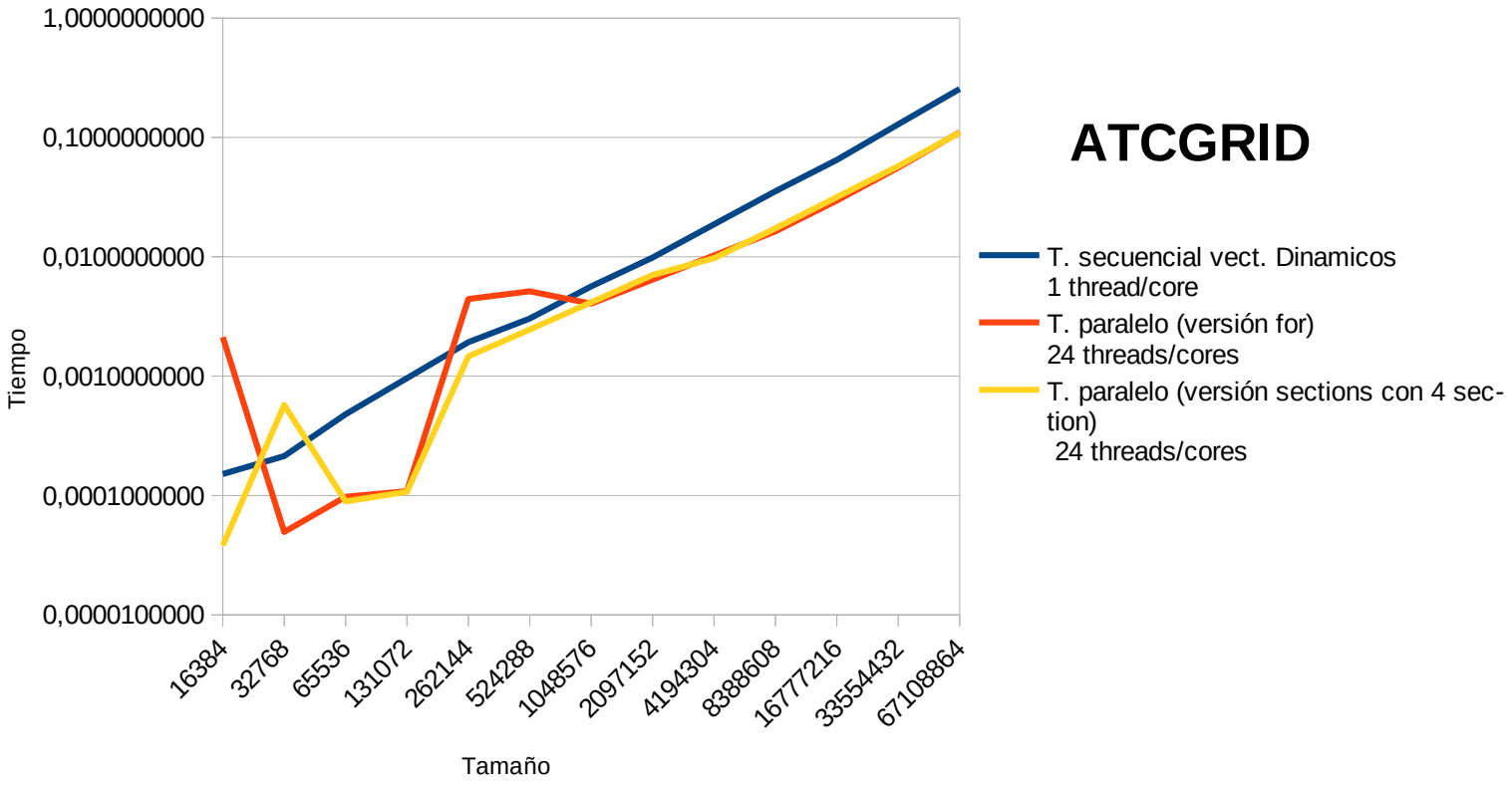
PC

Nº de Componentes	T. secuencial vect. Dinamicos 1 thread/core	T. paralelo (versión for) 8 threads/cores	T. paralelo (versión sections con 4 section) 8 threads/cores
16384	0,0000478740	0,0002503090	0,0000668290
32768	0,0001800690	0,0000270030	0,0009475910
65536	0,0003078450	0,0011653360	0,0002555650
131072	0,0004240900	0,0012796890	0,0025426220
262144	0,0009032060	0,0033577930	0,0019409490
524288	0,0019123890	0,0013196990	0,0047907750
1048576	0,0034425000	0,0030669740	0,0050681390
2097152	0,0073370570	0,0041952330	0,0116870830
4194304	0,0151813440	0,0080165050	0,0278042170
8388608	0,0275833410	0,0212225800	0,0323562960
16777216	0,0553102200	0,0251811750	0,0924719630
33554432	0,1115848050	0,0493671560	0,1351089990
67108864	0,2252885800	0,0979547720	0,2568085250



ATCGRID

Nº de Componentes	T. secuencial vect. Dinamicos 1 thread/core	T. paralelo (versión for) 24 threads/cores	T. paralelo (versión sections con 4 section) 24 threads/cores
16384	0,0001511980	0,0021203380	0,0000381100
32768	0,0002140800	0,0000494940	0,0005739280
65536	0,0004790800	0,0000976550	0,0000892690
131072	0,0009609540	0,0001085750	0,0001074060
262144	0,0019238290	0,0044247690	0,0014638080
524288	0,0030327080	0,0051432660	0,0024569690
1048576	0,0056485760	0,0040505790	0,0041534190
2097152	0,0098453280	0,0064382680	0,0070496690
4194304	0,0187167830	0,0102702320	0,0097135160
8388608	0,0353903470	0,0163482040	0,0172846410
16777216	0,0648195480	0,0295527450	0,0316392630
33554432	0,1288447200	0,0558574670	0,0575420280
67108864	0,2551396410	0,1105023730	0,1099452840



11. Rellenar una tabla como la Tabla 3 para atcgrid con el tiempo de ejecución, tiempo de CPU del usuario y tiempo CPU del sistema obtenidos con `time` para el ejecutable del ejercicio 7 y para el programa secuencial del Listado 1. Ponga en la tabla el número de threads/cores que usan los códigos. ¿El tiempo de CPU que se obtiene es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.
RESPUESTA:

En el secuencial, el real siempre es mayor o igual que la suma de los 2 tiempos de CPU(user + sys), mientras que en el paralelo, encontramos que el tiempo real es menor que los de CPU, esto se debe a que en el secuencial tiene un procesador(core) que lo realiza y solo cuenta ese tiempo, mientras que en el paralelo, el tiempo de CPU es la suma de los tiempos que tardan cada core en realizar el proceso.

Nº de Componentes	Tiempo secuencial vect. Globales 1 thread/core			Tiempo paralelo/versión for 24 Threads/cores		
	Elapsed	CPU-user	CPU- sys	Elapsed	CPU-user	CPU- sys
65536	real0m0.030s user0m0.001s Sys0m0.001s			real 0m0.026s user 0m0.001s Sys 0m0.004s		
131072	real 0m0.081s user 0m0.001s Sys 0m0.001s			real 0m0.023s User 0m0.000s sys 0m0.006s		
262144	real 0m0.038s user 0m0.002s Sys 0m0.006s			real 0m0.224s user 0m0.005s sys 0m0.006s		
524288	real 0m0.036s user 0m0.003s Sys 0m0.007s			real 0m0.040s user 0m0.014s Sys 0m0.007s		
1048576	real 0m0.032s user 0m0.006s Sys 0m0.011s			real 0m0.034s user 0m0.013s sys 0m0.013s		
2097152	real 0m0.050s user 0m0.013s Sys 0m0.015s			real 0m0.075s user 0m0.029s sys 0m0.022s		
4194304	real 0m0.095s user 0m0.026s Sys 0m0.025s			real 0m0.085s user 0m0.052s Sys 0m0.034s		
8388608	real 0m0.110s user 0m0.057s Sys 0m0.038s			real 0m0.132s user 0m0.098s sys 0m0.061s		
16777216	real 0m0.171s user 0m0.098s Sys 0m0.061s			real 0m0.178s user 0m0.174s sys 0m0.115s		
33554432	real 0m0.324s user 0m0.175s Sys 0m0.140s			real 0m0.277s user 0m0.328s sys 0m0.199s		
67108864	real 0m0.652s user 0m0.343s sys 0m0.299s			real 0m0.517s user 0m0.660s sys 0m0.346s		