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): Francisco José Aparicio Martos
Grupo de prácticas y profesor de prácticas:
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
#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
#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
#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
      {
      printf("Single ejecutada por el thread %d\n",omp_get_thread_num());

      for (i=0; i<n; i++)   printf("b[%d] = %d\t",i,b[i]);
      printf("\n");
      }

  }

  printf("Depués de la región parallel:\n");
  for (i=0; i<n; i++)   printf("b[%d] = %d\t",i,b[i]);
  printf("\n");
  return 0;
}
```

```
[FranciscoJoseAparicioMartos francisco@francisco-GE63-7RD:~/Escritorio/AC/practi
cas/bp1/ejer1] 2020-03-09 lunes
$./single
Introduce valor de inicialización a: 4
Single ejecutada por el thread 0
Single ejecutada por el thread 3
b[0] = 4        b[1] = 4        b[2] = 4        b[3] = 4        b[4] = 4        b
[5] = 4 b[6] = 4        b[7] = 4        b[8] = 4
Depués de la región parallel:
b[0] = 4        b[1] = 4        b[2] = 4        b[3] = 4        b[4] = 4        b
[5] = 4 b[6] = 4        b[7] = 4        b[8] = 4
```

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

    #pragma omp barrier

     #pragma omp for
     for (i=0; i<n; i++)
         b[i] = a;

      #pragma omp master
      {
      printf("Single ejecutada por el thread %d\n",omp_get_thread_num());

      for (i=0; i<n; i++)   printf("b[%d] = %d\t",i,b[i]);
      printf("\n");
      }

      #pragma omp barrier
  }

  printf("Depués de la región parallel:\n");
  for (i=0; i<n; i++)   printf("b[%d] = %d\t",i,b[i]);
  printf("\n");
  return 0;
}
```

```
[FranciscoJoseAparicioMartos francisco@francisco-GE63-7RD:~/Escritorio/AC/practicas/bp1/ejer3] 2020-03-09 lunes
$./single
Introduce valor de inicialización a: 6
Single ejecutada por el thread 0
Single ejecutada por el thread 0
b[0] = 6        b[1] = 6        b[2] = 6        b[3] = 6        b[4] = 6        b[5] = 6        b[6] = 6        b[7] = 6        b[8] = 6
Depués de la región parallel:
b[0] = 6        b[1] = 6        b[2] = 6        b[3] = 6        b[4] = 6        b[5] = 6        b[6] = 6        b[7] = 6        b[8] = 6
```

RESPUESTA A LA PREGUNTA:

La diferencia en este caso es que las secciones ejecutadas bajo la directiva master son ejecutadas por la hebra maestra, es decir, la hebra número 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:

Porque la directiva master no tiene una barrera implícita por lo que en las hebras no se esperan entre sí, provocando que en las distintas ejecuciones los resultados salgan diferentes pues depende de la cantidad de hebras que terminan a tiempo.

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:

```
[FranciscJoseAparicioMartos c1estudiante12@\g:~/bp1/ejer5] 2020-03-15 domingo
$sbatch -p ac -n1 --cpus-per-task=12 --hint=nomultithread --wrap "time ./SumaVectoresC 10000000"
Submitted batch job 20756
[FranciscJoseAparicioMartos c1estudiante12@\g:~/bp1/ejer5] 2020-03-15 domingo
$cat slurm-20756.out
Tamaño Vectores:10000000 (4 B)
Tiempo:0.060138752      / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /

real    0m0.210s
user    0m0.153s
sys     0m0.048s
```

El tiempo real es mayor que la suma de los tiempos que se ha ejecutado el programa en modo usuario y del sistema pues existen pequeños tiempos en los que el programa se puede quedar en espera.

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):

```
[FranciscJoseAparicioMartos c1estudiante12@\g:~/bp1/ejer6] 2020-03-15 domingo
$gcc -S SumaVectoresC.c
```

```
[FranciscJoseAparicioMartos c1estudiante12@\g:~/bp1/ejer6] 2020-03-15 domingo
$gcc SumaVectoresC.c -o SumaVectoresC
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

MIPS = NI/(Tcpu*10^6)

MFLOPS = OP_CF/(Tcpu*10^6)

Obtención de tiempos para 10000000 componentes

```
[c1estudiante12@atcgrid ejer6]$ sbatch -p ac -n1 --cpus-per-task=12 --hint=nomul
tithread --wrap "time ./SumaVectoresC 10000000"
Submitted batch job 20975
[c1estudiante12@atcgrid ejer6]$ cat slurm-20975.out
Tamaño Vectores:10000000 (4 B)
Tiempo:0.060362248       / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](1000
000.000000+1000000.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3[9999999
](1999999.900000+0.100000=2000000.000000) /

real    0m0.225s
user    0m0.169s
sys     0m0.045s
```

Obtención de tiempo para 10 componentes

```
[c1estudiante12@atcgrid ejer6]$ sbatch -p ac -n1 --cpus-per-task=12 --hint=nomul
tithread --wrap "time ./SumaVectoresC 10"
Submitted batch job 20988
[c1estudiante12@atcgrid ejer6]$ cat slurm-20988.out
Tamaño Vectores:10 (4 B)
Tiempo:0.000399694       / Tamaño Vectores:10   / V1[0]+V2[0]=V3[0](1.000000+1.0
00000=2.000000) / / V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /

real    0m0.025s
user    0m0.000s
sys     0m0.002s
```

19 instrucciones

4 op coma flotante

Para 10000000 componentes

MIPS = (5+14*10000000)/(0,060362248*10^6) = 2319,33 MIPS

MFLOPS = (4*10000000)/(0,060362248*10^6) = 662,66 MFLOPS

Para 10 comoponentes

MIPS = (5+14*10)/(0,000399694*10^6) = 0,363 MIPS

MFLOPS = (4*10)/(0,000399694*10^6) = 0,1 MFLOPS

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

```
        call    clock_gettime
        movl    $0, -4(%rbp)
        jmp     .L10
.L11:
        movl    -4(%rbp), %eax
        cltq
        movsd   v1(,%rax,8), %xmm1
        movl    -4(%rbp), %eax
        cltq
        movsd   v2(,%rax,8), %xmm0
        addsd   %xmm1, %xmm0
        movl    -4(%rbp), %eax
        cltq
        movsd   %xmm0, v3(,%rax,8)
        addl    $1, -4(%rbp)
.L10:
        movl    -4(%rbp), %eax
        cmpl    %eax, -8(%rbp)
        ja      .L11
        leaq    -48(%rbp), %rax
        movq    %rax, %rsi
        movl    $0, %edi
        call    clock_gettime
```

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

```
[FranciscoJoseAparicioMartos c1estudiante12@atcgrid:~/bp1/ejer7] 2020-03-18 miércoles
$gcc -fopenmp SumaVectoresC.c -o SumaVectoresC
```

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

  //clock_gettime(CLOCK_REALTIME,&cgt1); lo cambio por omp_get_wtime()

  #pragma omp single
  {
    start = omp_get_wtime();
  }

  //Calcular suma de vectores
  #pragma omp for
    for(i=0; i<N; i++)
      v3[i] = v1[i] + v2[i];


  #pragma omp single
  {
    end = omp_get_wtime();
  }


}
```

```
[FranciscoJoseAparicioMartos francisco@francisco-GE63-7RD:~/Escritorio/AC/practicas/bp1/ejer7] 2020-03-20 viernes
$./SumaVectoresC 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000016725       / Tamaño 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) /
[FranciscoJoseAparicioMartos francisco@francisco-GE63-7RD:~/Escritorio/AC/practicas/bp1/ejer7] 2020-03-20 viernes
$./SumaVectoresC 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000017753       / Tamaño Vectores:11   / V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) / / V1[10]+V2[10]=V
3[10](2.100000+0.100000=2.200000) /
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

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

```
#pragma omp parallel private(i)//para que cada thread tenga su propio i
{
  //inicializo los vectores
  #pragma omp sections
  {
    #pragma omp section
    {
      for(i=0; i<N/4; i++){
          v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
        }
    }

    #pragma omp section
    {
      for(i=N/4; i<N/2; i++){
          v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
        }
    }

    #pragma omp section
    {
      for(i=N/2; i<(N*3)/4; i++){
          v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
        }
    }


    #pragma omp section
    {
      for(i=(N*3)/4; i<N; i++){
          v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
        }
    }


  }

  //guardo el momento antes de empezar la suma
  #pragma omp single
  {
    start = omp_get_wtime();
  }
```

```
#pragma omp sections
{
  #pragma omp section
  {
    for(i=0; i<N/4; i++){
        v3[i] = v1[i] + v2[i];
    }
  }

  #pragma omp section
  {
    for(i=N/4; i<N/2; i++){
        v3[i] = v1[i] + v2[i];
    }
  }

  #pragma omp section
  {
    for(i=N/2; i<(N*3)/4; i++){
        v3[i] = v1[i] + v2[i];
    }
  }


  #pragma omp section
  {
    for(i=(N*3)/4; i<N; i++){
        v3[i] = v1[i] + v2[i];
    }
  }


}

#pragma omp single
{
  end = omp_get_wtime();
}

}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

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:

La cantidad de hebras que podrá usar será N, 1 por cada iteración del bucle, y la cantidad de cores que podrá usar serán los que estén disponibles durante la ejecución, pues no hemos puesto ninguna restricción en cuanto a la cantidad de cores usados. Con respecto al ejercicio 8, se podrá usar como máximo 4 hebras, 1 por cada section

10. Rellenar una tabla como la Tabla 2 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:

En atcgrid

```
[FranciscoJoseAparicioMartos francisco@francisco-GE63-7RD:~/Escritorio/AC/practicas/bp1/ejer10] 2020-03-2
2 domingo
$gcc -O2 -fopenmp SumaVectoresC_ejer7.c -o SumaVectoresejer7
[FranciscoJoseAparicioMartos francisco@francisco-GE63-7RD:~/Escritorio/AC/practicas/bp1/ejer10] 2020-03-2
2 domingo
$gcc -O2 -fopenmp SumaVectoresC_ejer8.c -o SumaVectoresejer8
[FranciscoJoseAparicioMartos francisco@francisco-GE63-7RD:~/Escritorio/AC/practicas/bp1/ejer10] 2020-03-2
2 domingo
```

Para el programa secuencial

```
[FranciscoJoseAparicioMartos c1estudiante12@atcgrid:~/bp1/ejer10] 2020-03-22 dom
ingo
$sbatch -n1 -p ac ./SumaVectores.sh
Submitted batch job 24236
```

Para la version for

```
[FranciscoJoseAparicioMartos c1estudiante12@atcgrid:~/bp1/ejer10] 2020-03-22 domingo
$sbatch -n1 -p ac --cpus-per-task=12 --hint=nomultithread ./SumaVectores.sh
```

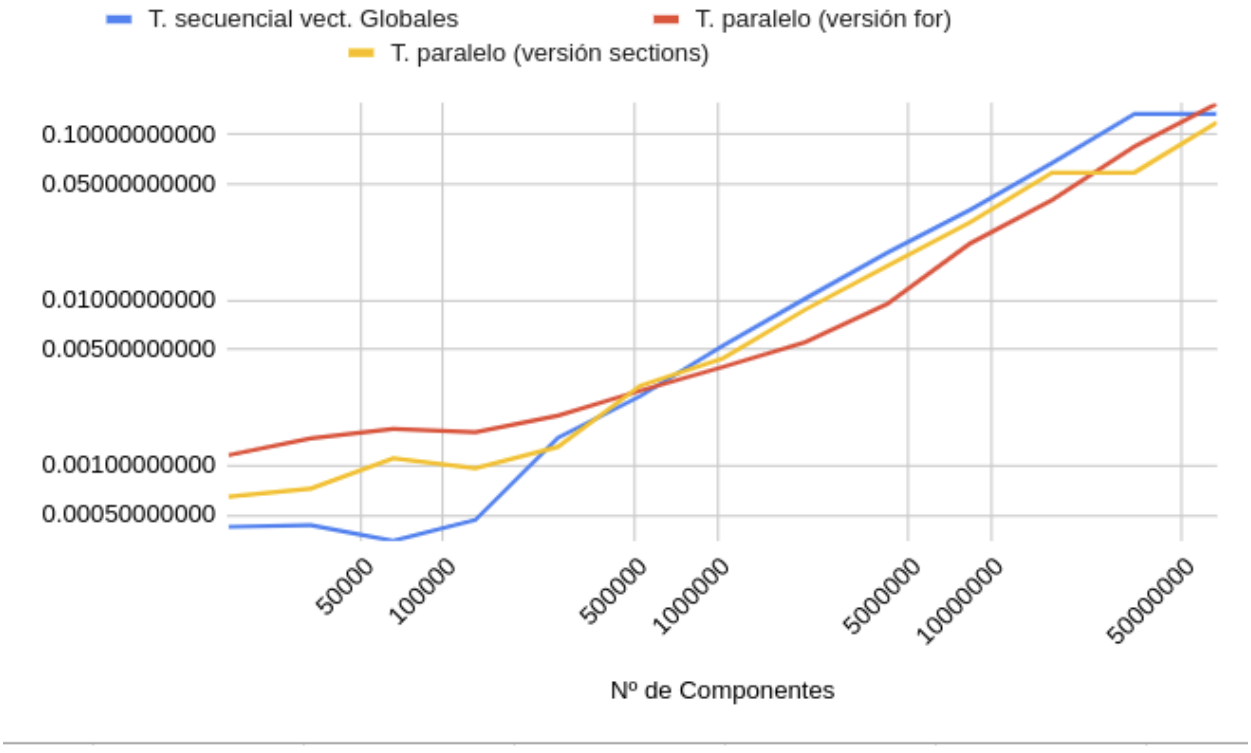
En el pc

```
[FranciscoJoseAparicio francisco@francisco-GE63-7RD:~/Escritorio/AC/practicas/bp
1/ejer10] 2020-03-22 domingo
$gcc -O2 SumaVectoresC.c -o SumaVectores
[FranciscoJoseAparicio francisco@francisco-GE63-7RD:~/Escritorio/AC/practicas/bp
1/ejer10] 2020-03-22 domingo
$gcc -fopenmp -O2 SumaVectoresC_ejer7.c -o SumaVectores7
[FranciscoJoseAparicio francisco@francisco-GE63-7RD:~/Escritorio/AC/practicas/bp
1/ejer10] 2020-03-22 domingo
$gcc -fopenmp -O2 SumaVectoresC_ejer8.c -o SumaVectores8
[FranciscoJoseAparicio francisco@francisco-GE63-7RD:~/Escritorio/AC/practicas/bp
1/ejer10] 2020-03-22 domingo
```

Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados, que debe coincidir con el número de cores físicos del computador que como máximo puede aprovechar el código.

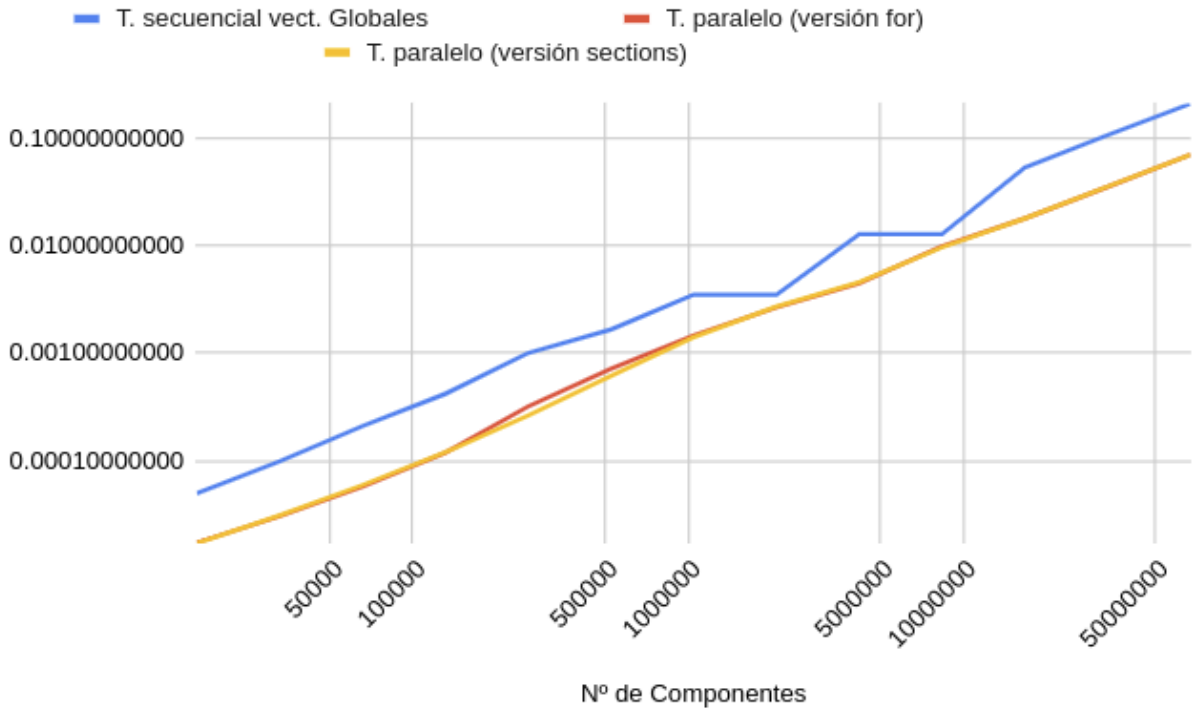
Para atcgrid

Nº de Componentes	**T. secuencial vect. Globales 1 thread/core**	**T. paralelo (versión `for`) 12 threads/cores**	**T. paralelo (versión `sections`) 12 threads/cores**
16384	0.000425532	0.001154596	0.000646805
32768	0.000435058	0.001461333	0.000723496
65536	0.000349933	0.001663221	0.001104772
131072	0.000469096	0.001586528	0.000960553
262144	0.001469020	0.001997624	0.001294363
524288	0.002622818	0.002834892	0.003026044
1048576	0.005281788	0.003936604	0.004423825
2097152	0.010179588	0.005553093	0.008728897
4194304	0.019307916	0.009447914	0.016052913
8388608	0.035089001	0.021905817	0.029407728
16777216	0.067275930	0.040220322	0.058956478
33554432	0.133391767	0.084903253	0.058956478
67108864	0.133176048	0.154164650	0.118321624



Ejecución en el pc

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión `for`) 12 threads/cores	T. paralelo (versión `sections`) 4 threads/cores
16384	0.000050002	0.000017481	0.000017158
32768	0.000099108	0.000030491	0.000031044
65536	0.000210562	0.000057721	0.000059329
131072	0.000419737	0.000118811	0.000120315
262144	0.001005272	0.000321691	0.000263022
524288	0.001665042	0.000719636	0.000615036
1048576	0.003523241	0.001472457	0.001417528
2097152	0.003523241	0.002674234	0.002721647
4194304	0.012894851	0.004483638	0.004580411
8388608	0.012894851	0.009883176	0.009731335
16777216	0.053500667	0.018018853	0.018023480
33554432	0.106666453	0.035329204	0.035528570
67108864	0.210874858	0.070428374	0.070612287



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:

Tabla 3. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados.

Nº de Componentes	**Tiempo secuencial vect. Globales 1 thread/core**			**Tiempo paralelo/versión `for` 12 Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	0,052	0	0,003	0.039	0.017	0.023
131072	0,024	0,002	0,001	0.077	0.022	0.026
262144	0,067	0,001	0,005	0.067	0.025	0.030
524288	0,067	0,004	0,006	0.079	0.042	0.040
1048576	0,066	0,006	0,01	0.067	0.038	0.061
2097152	0,067	0,016	0,012	0.089	0.079	0.066
4194304	0,093	0,029	0,022	0.086	0.146	0.073
8388608	0,114	0,044	0,048	0.108	0.349	0.182
16777216	0,197	0,094	0,082	0.101	0.517	0.271
33554432	0,333	0,162	0,149	0.161	1,039	0.584
67108864	0,333	0,176	0,136	0.337	1.920	1.043

En el secuencial el elapsed time es igual a la suma del cpu-user y el cpu-sys, como es de esperar, pero en la versión paralela, la suma de estos dos tiempos es mayor que el elapsed time, porque se esta contando el tiempo que consume cada core físico, por lo que al sumarlo es más tiempo del transcurrido realmente, pues en verdad estos tiempos son simultáneos y no secuenciales.