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 Domínguez Lorente
Grupo de prácticas y profesor de prácticas: Christian Morillas (B1)
Fecha de entrega: 21/03/2019
Fecha evaluación en clase: 4/4/2019

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`

```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
     {
        for (i=0; i<n; i++)   printf("Thread ID: %d - b[%d] = %d\t",omp_get_thread_num(),i,b[i]);
     }
  }
  return 0;
}
```

CAPTURAS DE PANTALLA:

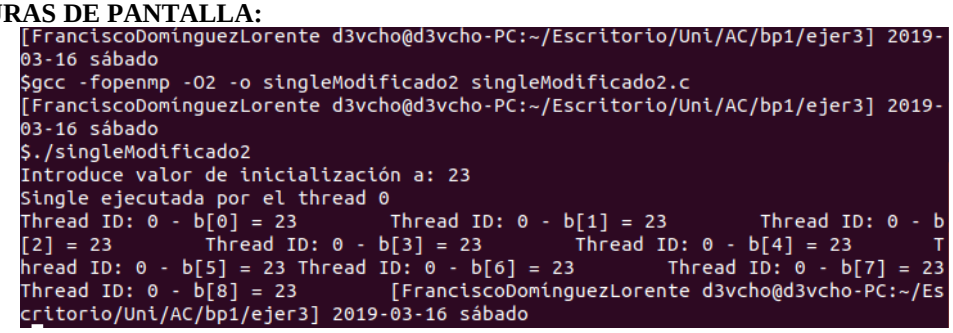
```
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp1/ejer2] 2019-
03-16 sábado
$export OMP_DYNAMIC=FALSE
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp1/ejer2] 2019-
03-16 sábado
$export OMP_NUM_THREADS=8
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp1/ejer2] 2019-
03-16 sábado
$gcc -fopenmp -O2 -o singleModificado singleModificado.c
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp1/ejer2] 2019-
03-16 sábado
$./singleModificado
Introduce valor de inicialización a: 23
Single ejecutada por el thread 0
Thread ID: 4 - b[0] = 23        Thread ID: 4 - b[1] = 23        Thread ID: 4 - b
[2] = 23        Thread ID: 4 - b[3] = 23        Thread ID: 4 - b[4] = 23       T
hread ID: 4 - b[5] = 23 Thread ID: 4 - b[6] = 23        Thread ID: 4 - b[7] = 23
Thread ID: 4 - b[8] = 23        [FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Es
critorio/Uni/AC/bp1/ejer2] 2019-03-16 sábado
```

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`

```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 master
     {
        for (i=0; i<n; i++)   printf("Thread ID: %d - b[%d] = %d\t",omp_get_thread_num(),i,b[i]);
     }
  }
  return 0;
}
```

CAPTURAS DE PANTALLA:



RESPUESTA A LA PREGUNTA: La diferencia es que obligatoriamente en este caso se ejecutará la sección single con la hebra master, es decir la hebra con identificador 0, En el anterior ejemplo, se podrá ejecutar con cualquier hebra,

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,

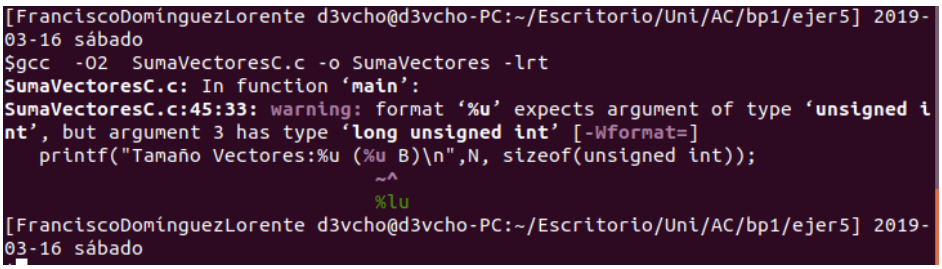
   RESPUESTA: Esto se debe a que la directiva *master* no tiene barrera implícita al final, lo que quiere decir que si se elimina la directiva *barrier*, la hebra *master* podría ejecutar el código que imrime la suma total sin que el resto de las hebras hayan actualizado el valor de la variable *suma*,

---

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:



```
[B1estudiante9@atcgrid ejer5]$ time ./SumaVectores 10000000
Tamaño Vectores:10000000 (4 B)
Tiempo:0.029366660       / 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    0m0.097s
user    0m0.028s
sys     0m0.069s
[B1estudiante9@atcgrid ejer5]$
```

RESPUESTA: En mi caso, la suma de *user* y *sys* coincide con el valor de *real*,

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):

```
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp1/ejer6] 2019-
03-16 sábado
$gcc  -O2 -S SumaVectoresC.c -lrt
SumaVectoresC.c: In function ‘main’:
SumaVectoresC.c:45:33: warning: format ‘%u’ expects argument of type ‘unsigned i
nt’, but argument 3 has type ‘long unsigned int’ [-Wformat=]
   printf("Tamaño Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                                ~^
                                %lu
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp1/ejer6] 2019-
03-16 sábado
```

```
sftp> lcd ejer6
sftp> put SumaVectores
Uploading SumaVectores to /home/B1estudiante9/bp1/ejer6/SumaVectores
SumaVectores                                  100%   12KB 180.9KB/s   00:00
sftp>
```

```
[B1estudiante9@atcgrid ejer6]$ echo './bp1/ejer6/SumaVectores 10' | qsub -
q ac
11841.atcgrid
[B1estudiante9@atcgrid ejer6]$ cat STDIN.e11841
[B1estudiante9@atcgrid ejer6]$ cat STDIN.o11841
Tamaño Vectores:10 (4 B)
Tiempo:0.000000178       / Tamaño Vectores:10    / V1[0]+V2[0]=V3[0](1.0000
00+1.000000=2.000000) / / V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /
[B1estudiante9@atcgrid ejer6]$
```

```
[B1estudiante9@atcgrid ejer6]$ echo './bp1/ejer6/SumaVectores 10000000' |
qsub -q ac
11843.atcgrid
[B1estudiante9@atcgrid ejer6]$ cat STDIN.e11843
[B1estudiante9@atcgrid ejer6]$ cat STDIN.o11843
Tamaño Vectores:10000000 (4 B)
Tiempo:0.039351385       / 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) /
[B1estudiante9@atcgrid ejer6]$
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Para calcular los MIPS tenemos que contar primeramente el número de instrucciones desde la primera llamada a clock_gettime y la segunda, La primera se produce en la línea 109 y la segunda en la línea 122, Entre estas dos llamadas, tenemos un total de 9 instrucciones,

Tenemos que tener en cuenta primeramente que las instrucciones pertenecientes al bucle for se ejecutarán tantas veces como componentes tenga el vector,

Para obtener los MIPS, tenemos que dividir el numero de instrucciones que se ejecutan entre el tiempo que ha tardado la ejecución, pasádolo previamente a millones *(multiplicando por 10^6)*, Por tanto, desarrollamos el cálculo de MIPS para tamaño N=10:

$$\frac{((6*10)+3)}{0.000000178*10^6} = 353{,}93\,MIPS$$

Para tamaño N=10000000 de igual forma tenemos:

$$\frac{((6*10000000)+3)}{0.039351385*10^6} = 1524{,}72\,MIPS$$

Para calcular los MFLOPS, seguimos el mismo razonamiento anterior pero solo teniendo en cuenta las instrucciones que contengan operaciones en coma flotante, aquellas que usen *%xmm_*,

Para N=10:

$$\frac{(3*10)}{0.000000178*10^6} = 168{,}53\,MFLOPS$$

Para N=10000000

$$\frac{(3*10000000)}{0.039351385*10^6} = 762{,}36\,MFLOPS$$

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

```
109     call    clock_gettime@PLT
110     xorl    %eax, %eax
111     .p2align 4,,10
112     .p2align 3
113 .L9:
114     movsd   0(%rbp,%rax), %xmm0
115     addsd   (%r14,%rax), %xmm0
116     movsd   %xmm0, (%r12,%rax)
117     addq    $8, %rax
118     cmpq    %r15, %rax
119     jne .L9
120     leaq    16(%rsp), %rsi
121     xorl    %edi, %edi
122     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

```
65    //Inicializar vectores
66    #pragma omp parallel for
67      for(i=0; i<N; i++){
68        v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
69      }
70
71    cgt1 = omp_get_wtime();
72    //Calcular suma de vectores
73    #pragma omp parallel for
74      for(i=0; i<N; i++)
75        v3[i] = v1[i] + v2[i];
76
77    ncgt = omp_get_wtime() - cgt1;
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL ,ZIP)

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

```
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp1/ejer7] 2019-
03-19 martes
$gcc -fopenmp -O2 -o SumaVectoresC-OMP SumaVectoresC-OMP.c
SumaVectoresC-OMP.c: In function 'main':
SumaVectoresC-OMP.c:46:33: warning: format '%u' expects argument of type 'unsign
ed int', but argument 3 has type 'long unsigned int' [-Wformat=]
   printf("Tamaño Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                                ~^
                                %lu
```

```
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp1/ejer7] 2019-
03-19 martes
$./SumaVectoresC-OMP 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000007046       / 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) /
```

```
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp1/ejer7] 2019-
03-19 martes
$./SumaVectoresC-OMP 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000006061       / Tamaño Vectores:11   / V1[0]+V2[0]=V3[0](1.100000+1.1
00000=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

```
65    //Inicializar vectores
66    #pragma omp parallel sections private(i)
67    {
68      #pragma omp section
69        for(i=0; i<N/4; i++){
70          v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
71        }
72
73      #pragma omp section
74        for(i=N/4; i<N/2; i++){
75          v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
76        }
77
78      #pragma omp section
79        for(i=N/2; i<3*N/4; i++){
80          v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
81        }
82
83      #pragma omp section
84        for(i=3*N/4; i<N; i++){
85          v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
86        }
87    }
88
89    cgt1 = omp_get_wtime();
```

```
86    //Calcular suma de vectores
87    #pragma omp parallel sections private(i)
88    {
89      #pragma omp section
90        for(i=0; i<N/4; i++)
91          v3[i] = v1[i] + v2[i];
92
93      #pragma omp section
94        for(i=N/4; i<N/2; i++)
95          v3[i] = v1[i] + v2[i];
96
97      #pragma omp section
98        for(i=N/2; i<3*N/4; i++)
99          v3[i] = v1[i] + v2[i];
100
101     #pragma omp section
102       for(i=3*N/4; i<N; i++)
103         v3[i] = v1[i] + v2[i];
104   }
105
106   ncgt = omp_get_wtime() - cgt1;
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL ,ZIP)

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

```
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp1/ejer8] 2019-
03-21 jueves
$gcc -fopenmp -O2 -o SumaVectoresC-OMP2 SumaVectoresC-OMP2.c
SumaVectoresC-OMP2.c: In function ‘main’:
SumaVectoresC-OMP2.c:46:33: warning: format ‘%u’ expects argument of type ‘unsig
ned int’, but argument 3 has type ‘long unsigned int’ [-Wformat=]
   printf("Tamaño Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                                ~^
                                %lu
```

```
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp1/ejer8] 2019-
03-21 jueves
$./SumaVectoresC-OMP2 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000006047      / 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) /
```

```
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp1/ejer8] 2019-
03-21 jueves
$./SumaVectoresC-OMP2 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000005799      / Tamaño Vectores:11   / V1[0]+V2[0]=V3[0](1.100000+1.1
00000=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: Para el ejercicio 7, como máximo se podrían ejecutar un número de hebras igual al número de componentes que tiene el vector,

   En el caso del ejercicio 8, como máximo se podrían ejecutar como máximo coincide con el número de sections que tenemos en nuestro código,

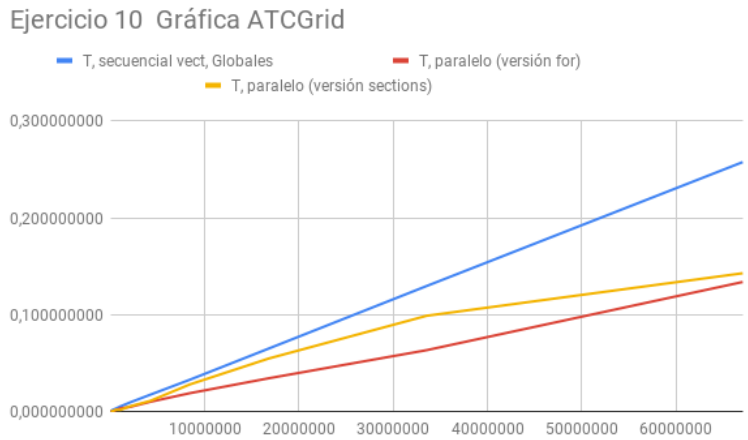
10. Rellenar una tabla como la Tabla 2 para atcgrid y otra para su PC con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1, Generar los ejecutables usando -O2, En la tabla debe aparecer el tiempo de ejecución del trozo de 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: Para PC:

Nº de Componentes	**T, secuencial vect, Globales 1 thread/core**	**T, paralelo (versión `for`) 4 threads/cores**	**T, paralelo (versión `sections`) 4 threads/cores**
16384	0,000177977	0,000022229	0,000021153
32768	0,000361778	0,000037313	0,000037984
65536	0,000808581	0,000082245	0,000077026
131072	0,002096280	0,000448797	0,000178836
262144	0,003166329	0,000544758	0,000555540
524288	0,002961855	0,001383162	0,001713424
1048576	0,004256309	0,002233312	0,002255799
2097152	0,007273798	0,004365103	0,004415647
4194304	0,014016246	0,008698808	0,009417083
8388608	0,027917273	0,017121630	0,018839051
16777216	0,054773586	0,034128064	0,033762449
33554432	0,112462780	0,065976242	0,065484649
67108864	0,219647325	0,132061872	0,130131556

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	0.000116661	0.000027520	0.000049410
32768	0.000234118	0.000047190	0.000101885
65536	0.000466850	0.000065440	0.000156640
131072	0.000937067	0.000126636	0.000347448
262144	0.001871137	0.000225195	0.000632940
524288	0.002837887	0.001409213	0.001643919
1048576	0.005272313	0.003207339	0.001655960
2097152	0.009479355	0.004746002	0.005441997
4194304	0.017257688	0.009960098	0.010891988
8388608	0.032702454	0.018837625	0.027845395
16777216	0.064946683	0.034273373	0.054676096
33554432	0.129581164	0.063324464	0.098835549
67108864	0.257566707	0.133748914	0.142779811



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:

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	0.007	0.000	0.004	0.006	0.015	0.003
131072	0.005	0.001	0.004	0.003	0.011	0.002
262144	0.007	0.003	0.004	0.003	0.018	0.002
524288	0.010	0.005	0.005	0.006	0.041	0.010
1048576	0.017	0.009	0.007	0.012	0.062	0.047
2097152	0.028	0.013	0.015	0.014	0.091	0.054
4194304	0.048	0.025	0.023	0.021	0.147	0.080
8388608	0.083	0.040	0.042	0.040	0.290	0.156
16777216	0.159	0.083	0.076	0.073	0.519	0.306
33554432	0.313	0.159	0.154	0.144	1.029	0.564
67108864	0.621	0.337	0.284	0.277	1.916	1.132