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): Laura Sánchez Parra.
Grupo de prácticas: Miércoles.
Fecha de entrega: Martes, 10 Abril 2018.
Fecha evaluación en clase: Miércoles, 11 Abril 2018.

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_$

                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_threa$
}
void funcB() {
printf("En funcB: esta sección la ejecuta el thread %d\n", omp_get_thread_num()$
}
main() {
        #pragma omp parallel sections
{
                #pragma omp section
                (void) funcA();
                #pragma omp section
                (void) funcB();
        }
        return(0);
}
                                [ 20 líneas leídas ]
```

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>

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 de 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("DEspués de la región del parallerl:\n");
                        for(i=0;i<n;i++) printf("c[%d] = %d impreso por el thread %d\t", i, b[i], omp_get_thread_num());
                        printf("\n");
                        }
                }
        }
```

CAPTURAS DE PANTALLA:

```
laura@laura-portatil:~/Escritorio/Uni/AC/practica2$ gcc -fopenmp singleMODIFICADO.c -o sigleMODIFICADO
```

```
laura@laura-portatil:~/Escritorio/Uni/AC/practica2$ ./sigleMODIFICADO
Introduce valor de inicialización de a: 20
Single ejecutada por el thread 1
DEspués de la región del parallerl:
c[0] = 20 impreso por el thread 0       c[1] = 20 impreso por el thread 0       c[2] = 20 impreso por el thread 0       c[3] = 20 impreso por
el thread 0     c[4] = 20 impreso por el thread 0       c[5] = 20 impreso por el thread 0       c[6] = 20 impreso por el thread 0       c[7] =
 20 impreso por el thread 0     c[8] = 20 impreso por el thread 0
laura@laura-portatil:~/Escritorio/Uni/AC/practica2$
```

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>

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 de 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
                {
                        printf("Después de la región del parallel:\n");
                        for(i=0;i<n;i++) printf("c[%d] = %d impreso por el thread %d\t", i, b[i], omp_get_thread_num());
                        printf("\n");
                        }
                }
        }
```

CAPTURAS DE PANTALLA:

```
laura@laura-portatil:~/Escritorio/Uni/AC/practica2$ gcc -fopenmp SingleMaster.c -o SingleMaster
```

```
laura@laura-portatil:~/Escritorio/Uni/AC/practica2$ ./SingleMaster
Introduce valor de inicialización de a: 20
Single ejecutada por el thread 0
Después de la región del parallel:
c[0] = 20 impreso por el thread 0       c[1] = 20 impreso por el thread 0       c[2] = 20 impreso por el thread 0       c[3] = 20 impreso p
el thread 0     c[4] = 20 impreso por el thread 0       c[5] = 20 impreso por el thread 0       c[6] = 20 impreso por el thread 0       c[7
 20 impreso por el thread 0     c[8] = 20 impreso por el thread 0
```

RESPUESTA A LA PREGUNTA:

Aparentemente los resultados son los mismos.

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:

   Recordemos que la directiva single tiene una barrera implícita al final, es decir, conforme cada hebra vaya acabando la tarea que le ha sido asignada, espera a que el resto acabe en ese punto para continuar con la ejecución. Sin embargo, la directiva master no tiene ninguna barrera, de forma que es necesario que el programador coloque manua7lmente dicha barrera para evitar errores.

   En nuestro caso, no existe ninguna barrera después de la directiva master, por lo que podrían llegar a darse errores de dependencia de datos.

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:

```
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-03-31 sábado
$echo 'time ./sumavectoresc 67108864' | qsub -q ac
70223.atcgrid
```

```
real    0m1.205s
user    0m0.402s
sys     0m0.792s
```

Como podemos comprobar, la suma de tiempo de usuario y el tiempo de sistema es de **0m 1,194s ,** mientras que el tiempo real es de 0m 1,205s. Está claro que la suma de tiempo de usuario más tiempo de sistema es menos que el tiempo real.

Esto se debe a que al ejecutar el código de manera secuencial, se invierte más tiempo en realizar operaciones, además del tiempo de usuario ,por lo tanto, el tiempo empleado es mayor.

Como veremos en los próximos ejercicios, el introducir bloques de código que se ejecuten en paralelo hará a la diferencia de tiempo cada vez más pequeña e incluso que el tiempo real sea menor que la suma de tiempo de sistema y usuario. Esto se debe a que el comando time contabiliza la suma de tiempo de ejecución aunque el código se ejecute en paralelo, por lo tanto, el tiempo de ejecución será menor que el "tiempo de cómputo".

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). Incorpore **el código ensamblador de la parte de la suma de vectores** en el cuaderno.

   CAPTURAS DE PANTALLA:

   Generamos código ensamblador:

```
laura@laura-portatil:~/Uni/AC/practica2$ gcc -S sumavectoresc.c
```

   Código ensamblador de la suma:

```
        call    clock_gettime@PLT
        xorl    %eax, %eax
        .p2align 4,,10
        .p2align 3
.L5:
        movsd   (%r12,%rax,8), %xmm0
        addsd   0(%rbp,%rax,8), %xmm0
        movsd   %xmm0, 0(%r13,%rax,8)
        addq    $1, %rax
        cmpl    %eax, %ebx
        ja      .L5
.L6:
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Para N=10

```
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$echo './sumavectoresc 10' | qsub -q ac
```

```
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$ls -l
total 32
-rw------- 1 E1estudiante25 E1estudiante25     0 abr  1 16:44 STDIN.e70439
-rw------- 1 E1estudiante25 E1estudiante25   148 abr  1 16:44 STDIN.o70439
-rwxr-xr-x 1 E1estudiante25 E1estudiante25  8616 abr  1 16:44 sumavectoresc
```

```
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$cat STDIN.o70439
Tiempo(seg.):0.000001633        / Tamaño Vectores:10    /V1[0]+V2[0]=V3[0](1.000
000+1.000000=2.000000) / /V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /
```

MIPS=3+6*10 / 0,000001633*10^6 = 38,57

MFLOPS=3*10/0,000001633*10^6= 18,37

Para N=10000000

```
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$echo './sumavectoresc 10000000' | qsub -q ac
70442.atcgrid
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$ls -l
total 32
-rw------- 1 E1estudiante25 E1estudiante25     0 abr  1 16:49 STDIN.e70442
-rw------- 1 E1estudiante25 E1estudiante25   202 abr  1 16:49 STDIN.o70442
-rwxr-xr-x 1 E1estudiante25 E1estudiante25  8616 abr  1 16:44 sumavectoresc
-rwxr-xr-x 1 E1estudiante25 E1estudiante25 13200 abr  1 16:41 sumavectorescmodif
icado
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$cat STDIN.o70442
Tiempo(seg.):0.063329956        / Tamaño Vectores:10000000      /V1[0]+V2[0]=V3[
0](1000000.000000+1000000.000000=2000000.000000) / /V1[9999999]+V2[9999999]=V3[9
999999](1999999.900000+0.100000=2000000.000000) /
```

MIPS= 3+6*10000000/0,063329956*10^6= 994,789

MFLOPS= 3*10000000/0,063329956*10^6 = 473,709

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

```c
#include <stdlib.h>
#include<stdio.h>
#include"omp.h"

#define VECTOR_GLOBAL
#define MAX 67108864    //=2^26
double v1[MAX], v2[MAX], v3[MAX];

int main(int argc,char** argv){
  int  i;


  double start, end, time;
  //Leer argumento de entrada (nº de componentes del vector)

  if  (argc<2){
    printf("Faltan nº componentes del vector \n");
    exit(-1);
  }

  unsigned int N = atoi(argv[1]);   // Máximo N =2^32  -1=4294967295 (sizeof(unsigned int) = 4 B)
  if (N>MAX)
    N=MAX;
  #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;
    //los valores dependen de N
  }

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

  end=omp_get_wtime( );
  time=end-start;

  //Imprimir resultado de la suma y el tiempo de ejecución

  printf("Tiempo(seg.):%0.5f \t/ Tamaño Vectores:%u \t/"
      "V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) / /"
      "V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) / \n",
      time, N, v1[0], v2[0], v3[0], N-1, N-1, N-1, v1[N-1], v2[N-1], v3[N-1]);

  return  0;
}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
laura@laura-portatil:~/Uni/AC/practica2$ gcc -O2 -fopenmp sumavectorescmodificad
o.c -o sumavectorescmodificado
[1]+  Hecho                    gedit sumavectorescmodificado.c
laura@laura-portatil:~/Uni/AC/practica2$
```

```
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$echo './sumavectorescmodificado 8' | qsub -q ac
70448.atcgrid
```

```
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$ls -l
total 32
-rw------- 1 E1estudiante25 E1estudiante25     0 abr  1 16:53 STDIN.e70448
-rw------- 1 E1estudiante25 E1estudiante25   143 abr  1 16:53 STDIN.o70448
-rwxr-xr-x 1 E1estudiante25 E1estudiante25  8616 abr  1 16:44 sumavectoresc
-rwxr-xr-x 1 E1estudiante25 E1estudiante25 13200 abr  1 16:41 sumavectorescmodif
icado
```

```
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$cat STDIN.o70448
Tiempo(seg.):0.00000     / Tamaño Vectores:8     /V1[0]+V2[0]=V3[0](0.800000+0.80
0000=1.600000) / /V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
```

```
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$echo './sumavectorescmodificado 10' | qsub -q ac
70450.atcgrid
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$ls -l
total 32
-rw------- 1 E1estudiante25 E1estudiante25     0 abr  1 16:53 STDIN.e70448
-rw------- 1 E1estudiante25 E1estudiante25   143 abr  1 16:53 STDIN.o70448
-rwxr-xr-x 1 E1estudiante25 E1estudiante25  8616 abr  1 16:44 sumavectoresc
-rwxr-xr-x 1 E1estudiante25 E1estudiante25 13200 abr  1 16:41 sumavectorescmodif
icado
```

```
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$cat STDIN.o70448
Tiempo(seg.):0.00000    / Tamaño Vectores:8     /V1[0]+V2[0]=V3[0](0.800000+0.80
0000=1.600000) / /V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
```

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 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

```
10 #include <stdlib.h>
11 #include<stdio.h>
12 #include<omp.h>
13
14 #define VECTOR_GLOBAL
15 #define MAX 67108864    //=2^26
16 double v1[MAX], v2[MAX], v3[MAX];
17
18 int main(int argc,char** argv){
19   int  i;
20
21
22   double start, end, time;
23   //Leer argumento de entrada (nº de componentes del vector)
24
25   if  (argc<2){
26     printf("Faltan nº componentes del vector \n");
27     exit(-1);
28   }
29
30   unsigned int N = atoi(argv[1]);   // Máximo N =2^32  -1=4294967295
31   if (N>MAX)
32     N=MAX;
```

```
33   #pragma omp parallel sections
34   {
35
36   //Inicializar vectores
37       #pragma omp section
38       {
39         for  (i=0; i<N; i+=4){
40           v1[i] =  N*0.1+  i*0.1; v2[i] = N*0.1  -  i*0.1;
41           //los valores dependen de N
42         }
43       }
44       #pragma omp section
45       {
46         for  (i=1; i<N; i+=4){
47           v1[i] =  N*0.1+  i*0.1; v2[i] = N*0.1  -  i*0.1;
48         }
49       }
50       #pragma omp section
51       {
52         for  (i=2; i<N; i+=4){
53           v1[i] =  N*0.1+  i*0.1; v2[i] = N*0.1  -  i*0.1;
54         }
55       }
56   }
57
58   start=omp_get_wtime( );
59   //Calcular suma de vectores
60   #pragma omp parallel sections
61   {
62       #pragma omp section
63       {
64         for  (i=0; i<N; i+=4)
65           v3[i] = v1[i] + v2[i];
66       }
67       #pragma omp section
68       {
69        for  (i=1; i<N; i+=4)
70           v3[i] = v1[i] + v2[i];
71     }
72     #pragma omp section
73     {
74      for  (i=2; i<N; i+=4)
75         v3[i] = v1[i] + v2[i];
76   }
77  }
78
79
80   end=omp_get_wtime( );
81   time=end-start;
82
83   //Imprimir resultado de la suma y el tiempo de ejecución
84
85   printf("Tiempo(seg.):%0.5f \t/ Tamaño Vectores:%u \t/"
86       "V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) / /"
87       "V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) / \n",
88       time, N, v1[0], v2[0], v3[0], N-1, N-1, N-1, v1[N-1], v2[N-1], v3[N-1]);
89
90   return  0;
91 }
```

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

```
laura@laura-portatil:~/Uni/AC/practica2$ gcc -O2 -fopenmp sumavectorescsections.
c -o sumavectorescsections
```

```
sftp> put sumavectorescsections
Uploading sumavectorescsections to /home/E1estudiante25/sumavectorescsections
sumavectorescsections                         100%   13KB 258.7KB/s   00:00
```

```
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$echo './sumavectorescmodificado 8' | qsub -q ac
70503.atcgrid
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$ls -l
total 56
-rw------- 1 E1estudiante25 E1estudiante25     0 abr  1 16:53 STDIN.e70448
-rw------- 1 E1estudiante25 E1estudiante25     0 abr  1 16:56 STDIN.e70450
-rw------- 1 E1estudiante25 E1estudiante25     0 abr  1 19:09 STDIN.e70503
-rw------- 1 E1estudiante25 E1estudiante25   143 abr  1 16:53 STDIN.o70448
-rw------- 1 E1estudiante25 E1estudiante25   144 abr  1 16:56 STDIN.o70450
-rw------- 1 E1estudiante25 E1estudiante25   143 abr  1 19:09 STDIN.o70503
-rwxr-xr-x 1 E1estudiante25 E1estudiante25  8616 abr  1 16:44 sumavectoresc
-rwxr-xr-x 1 E1estudiante25 E1estudiante25 13200 abr  1 16:41 sumavectorescmodif
icado
-rwxr-xr-x 1 E1estudiante25 E1estudiante25 13208 abr  1 19:08 sumavectorescsecti
ons
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$cat STDIN.o70503
Tiempo(seg.):0.00001    / Tamaño Vectores:8     /V1[0]+V2[0]=V3[0](0.800000+0.80
0000=1.600000) / /V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
```

```
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$echo './sumavectorescmodificado 10' | qsub -q ac
70509.atcgrid
```

```
[LauraSánchezParra E1estudiante25@atcgrid: ~] 2018-04-01 domingo
$ cat STDIN.o70509
Tiempo(seg.):0.00000    / Tamaño Vectores:10    /V1[0]+V2[0]=V3[0](1.000000+1.00
0000=2.000000) / /V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /
```

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:

El código del ejercicio 7 usa la directiva for, de manera que usará todos las hebras disponibles. En nuestro caso, como estamos ejecutando en atcgrid, tenemos disponibles 24 hebras y ningún otro proceso en cola, por lo que en la ejecución del ejercicio 7 se usarán 24 hebras, consecuentemente 12 cores lógicos.

En el ejercicio 8, sin embargo, implementamos el mismo programa con la directiva sections. En nuestra implementación, hemos dividido el programa en 3 bloques de sección, de manera que para cada operación, tanto para asignar los valores del vector, como para calcular la suma utilizaremos como máximo 3 hebras, en consecuencia 2 cores lógicos.

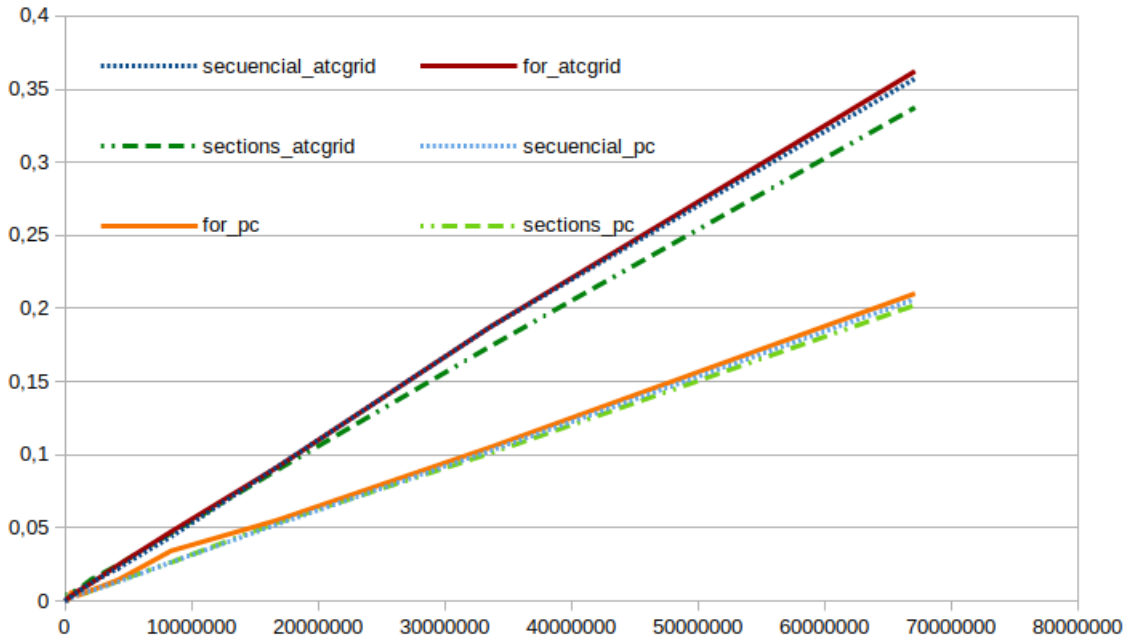
10. Rellenar una tabla como la Tabla 2Error: no se encontró el origen de la referencia para atcgrid y otra para su PC con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe 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. Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute código que imprima todos los componentes del resultado cuando este número sea elevado.

RESPUESTA: DATOS ATCGRID

Nº de Componentes	**T. secuencial vect. Globales 1 thread/core**	**T. paralelo (versión `for`) ¿? threads/cores**	**T. paralelo (versión `sections`) ¿?threads/cores**
16384	0,000138591	0,00012	0,00395
32768	0,000133540	0,00021	0,00450
65536	0,000430926	0,00045	0,00424
131072	0,000828730	0,00089	0,00443
262144	0,001338132	0,00181	0,00376
524288	0,003514294	0,00334	0,00571
1048576	0,006115808	0,00680	0,00769
2097152	0,012529154	0,01214	0,01474
4194304	0,022278978	0,02443	0,02453
8388608	0,043932141	0,04729	0,04626
16777216	0,091650322	0,09156	0,08970
33554432	0,187177045	0,18725	0,17402
67108864	0,357260221	0,36231	0,33748

DATOS PC

Nº de Componentes	**T. secuencial vect. Globales 1 thread/core**	**T. paralelo (versión `for`) ¿? threads/cores**	**T. paralelo (versión `sections`) ¿?threads/cores**
16384	0,000221305	0,00014	0,00026
32768	0,000083248	0,00012	0,00009
65536	0,000281296	0,00028	0,00018
131072	0,000351037	0,00041	0,00037
262144	0,000758243	0,00089	0,00088
524288	0,001771985	0,00582	0,00460
1048576	0,004697580	0,00373	0,00362
2097152	0,006827147	0,00728	0,00695
4194304	0,013075369	0,01430	0,01361
8388608	0,026387770	0,03423	0,02643
16777216	0,052430101	0,05525	0,05370
33554432	0,102648653	0,10501	0,10094
67108864	0,206496475	0,21029	0,20229



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` ¿? Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	0m0.004s	0m0.001s	0m0.001s	0m0.014s	0m0.166s	0m0.002s
131072	0m0.004s	0m0.001s	0m0.003s	0m0.011s	0m0.146s	0m0.015s
262144	0m0.006s	0m0.001s	0m0.005s	0m0.013s	0m0.144s	0m0.026s
524288	0m0.010s	0m0.004s	0m0.006s	0m0.014s	0m0.187s	0m0.004s
1048576	0m0.023s	0m0.008s	0m0.014s	0m0.020s	0m0.183s	0m0.014s
2097152	0m0.032s	0m0.011s	0m0.019s	0m0.020s	0m0.183s	0m0.014s
4194304	0m0.032s	0m0.011s	0m0.019s	0m0.045s	0m0.203s	0m0.089s
8388608	0m0.139s	0m0.055s	0m0.082s	0m0.079s	0m0.237s	0m0.209s
16777216	0m0.272s	0m0.080s	0m0.190s	0m0.149s	0m0.323s	0m0.355s
33554432	0m0.542s	0m0.186s	0m0.353s	0m0.281s	0m0.469s	0m0.762s
67108864	0m1.054s	0m0.355s	0m0.690s	0m0.552s	0m0.732s	0m1.559s