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):
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`

```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("Dentro de la región parallel:\n");
       for (i=0; i<n; i++)   printf("b[%d] = %d\t",i,b[i]);
       printf("\n");
       printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
     }
  }



  return 0;
}
```

CAPTURAS DE PANTALLA:

```
[IgnacioSanchezHerrera nacho@debian:~/Dropbox/Universidad/2º/S2/Informática/AC/P
rácticas/practica2/codigo] 2019-03-23 sábado
$./single
Introduce valor de inicialización a: 10
Single ejecutada por el thread 1
Dentro de la región parallel:
b[0] = 10       b[1] = 10       b[2] = 10       b[3] = 10       b[4] = 10       b
[5] = 10        b[6] = 10       b[7] = 10       b[8] = 10
Single ejecutada por el thread 2
```

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
     {
       printf("Dentro de la región parallel:\n");
       for (i=0; i<n; i++)   printf("b[%d] = %d\t",i,b[i]);
       printf("\n");
       printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
     }
  }



  return 0;
}
```

CAPTURAS DE PANTALLA:

```
[IgnacioSanchezHerrera nacho@debian:~/Disco2/Universidad/practicas_AC/bp1/ejer3]
 2019-03-23 sábado
$./singleModificado2
Introduce valor de inicialización a: 10
Single ejecutada por el thread 2
Dentro de la región parallel:
b[0] = 10       b[1] = 10       b[2] = 10       b[3] = 10       b[4] = 10       b
[5] = 10        b[6] = 10       b[7] = 10       b[8] = 10
Single ejecutada por el thread 0
```

RESPUESTA A LA PREGUNTA:

La única diferencia es: el thread que muestra los valores del vector es la hebra 0 (hebra master) ya que en el ejercicio anterior los mostraba la primera hebra que llegase a esta región del programa, mientras que en esta siempre lo ejecuta la hebra master (hebra nº 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 si no, la hebra master puede mostrar el resultado antes de que el resto de hebras hayan acabado de sumar el resultado de su suma local a la variable resultado.

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:

```
[IgnacioSanchezHerrera E3estudiante25@atcgrid:~/bp1/ejer5] 2019-03-23 sábado
$echo 'time ~/bp1/ejer5/SumaVectoresC 10000000' | qsub -q ac
13115.atcgrid
```

```
[IgnacioSanchezHerrera E3estudiante25@atcgrid:~/bp1/ejer5] 2019-03-23 sábado
$cat STDIN.e13115

real    0m0.217s
user    0m0.170s
sys     0m0.043s
```

RESPUESTA:

La suma de los tiempos de CPU del usuario y del sistema es menor que el tiempo real, ya que en el tiempo real también se incluye el tiempo asociado a las esperas debidas a I/O o asociados a la ejecución de otros programas.

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):

```
[IgnacioSanchezHerrera nacho@debian:~/Disco2/Universidad/practicas_AC/bp1/ejer6] 2019-03-25 lunes
$gcc  -O2  SumaVectoresC.c -o SumaVectoresC -lrt
```

```
[IgnacioSanchezHerrera nacho@debian:~/Disco2/Universidad/practicas_AC/bp1/ejer6] 2019-03-25 lunes
$gcc  -O2 -S SumaVectoresC.c -lrt
```

```
[IgnacioSanchezHerrera E3estudiante25@atcgrid:~/bp1/ejer6] 2019-03-25 lunes
$echo '~/bp1/ejer6/SumaVectoresC 10' | qsub -q ac
13450.atcgrid
[IgnacioSanchezHerrera E3estudiante25@atcgrid:~/bp1/ejer6] 2019-03-25 lunes
$cat STDIN.o13450
Tamaño Vectores:10 (4 B)
Tiempo:0.000398722      / Tamaño Vectores:10   / V1[0]+V2[0]=V3[0](1.000000+1.000000=2.000000) / / V1[9]+V2[9]=V3[9](1.900000+0.100000=2.00
0000) /
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

MIPS:

Nº instrucciones del fragmento = 21

Tamaño = 10 => Nº instrucciones = 10*21 = 210

Tiempo = 0.000398722 s => MIPS = ((10*21)/0.000397330) * 10^{-6} = 0.526683 MIPS

Tamaño = 10000000 => Nº instrucciones = 21*10^7

Tiempo = 0. 043612749 s => MIPS = ((10000000*21)/0.043612749) * 10^{-6} = 4815.105 MIPS

MFLOPS:

Nº instrucciones con punto flotante en el fragmento = 1 (La suma)

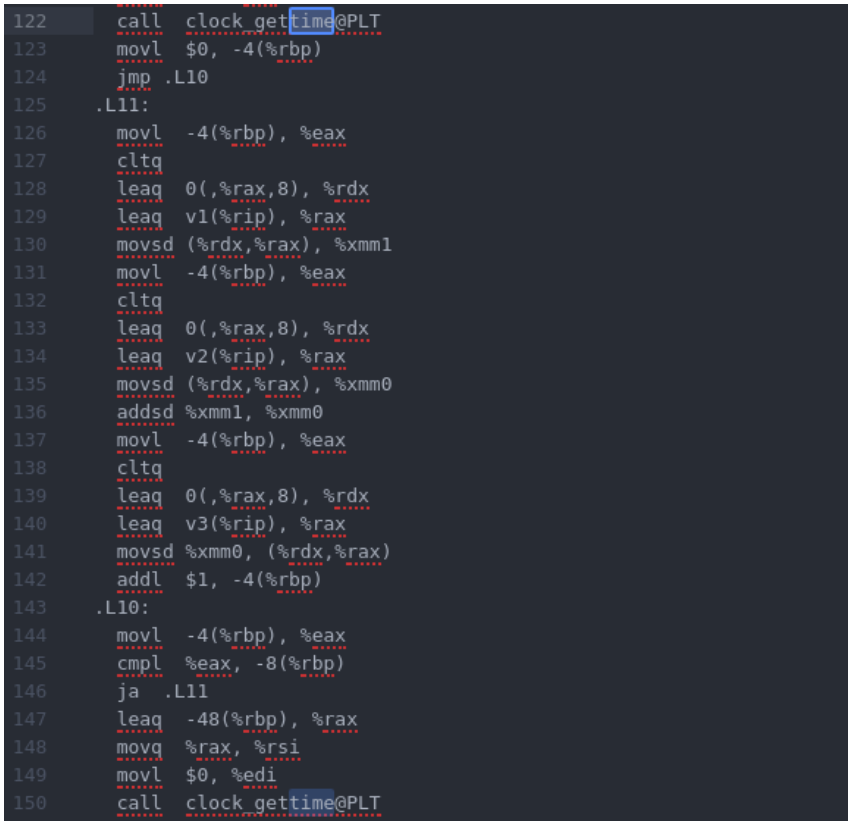
Tamaño = 10 => Nº instrucciones FP = 10

Tiempo = 0.000398722 s => MFLOPS = (10/0.000398722)*10^{-6} = 0.0250801 MFLOPS

Tamaño = 10000000 => Nº instrucciones FP = 10000000

Tiempo = 0. 043612749 s => MFLOPS = (10000000/ 0.043612749)*10^{-6} = 229.29 MFLOPS

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



```
122     call  clock_gettime@PLT
123     movl  $0, -4(%rbp)
124     jmp .L10
125   .L11:
126     movl  -4(%rbp), %eax
127     cltq
128     leaq  0(,%rax,8), %rdx
129     leaq  v1(%rip), %rax
130     movsd (%rdx,%rax), %xmm1
131     movl  -4(%rbp), %eax
132     cltq
133     leaq  0(,%rax,8), %rdx
134     leaq  v2(%rip), %rax
135     movsd (%rdx,%rax), %xmm0
136     addsd %xmm1, %xmm0
137     movl  -4(%rbp), %eax
138     cltq
139     leaq  0(,%rax,8), %rdx
140     leaq  v3(%rip), %rax
141     movsd %xmm0, (%rdx,%rax)
142     addl  $1, -4(%rbp)
143   .L10:
144     movl  -4(%rbp), %eax
145     cmpl  %eax, -8(%rbp)
146     ja  .L11
147     leaq  -48(%rbp), %rax
148     movq  %rax, %rsi
149     movl  $0, %edi
150     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

```
#pragma omp parallel
{
  #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;
  }

  t_ini = omp_get_wtime();

  #pragma omp for
  for(i=0; i<N; i++)
    v3[i] = v1[i] + v2[i];

  t_fin = omp_get_wtime();

}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[IgnacioSanchezHerrera nacho@debian:~/Disco2/Universidad/practicas_AC/bp1/ejer7] 2019-03-25 lunes
$gcc -O2 -fopenmp SumaVectoresC_paralela.c -o SumaVectoresC_paralela -lrt
```

```
[IgnacioSanchezHerrera E3estudiante25@atcgrid:~/bp1/ejer7] 2019-03-25 lunes
$echo '/home/E3estudiante25/bp1/ejer7/SumaVectoresC_paralela 8' | qsub -q ac
13590.atcgrid
[IgnacioSanchezHerrera E3estudiante25@atcgrid:~/bp1/ejer7] 2019-03-25 lunes
$cat STDIN.o13590
Tamaño Vectores:8 (4 B)
Tiempo:0.002476787       / 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) /
```

```
[IgnacioSanchezHerrera E3estudiante25@atcgrid:~/bp1/ejer7] 2019-03-25 lunes
$echo '/home/E3estudiante25/bp1/ejer7/SumaVectoresC_paralela 11' | qsub -q ac
13591.atcgrid
[IgnacioSanchezHerrera E3estudiante25@atcgrid:~/bp1/ejer7] 2019-03-25 lunes
$cat STDIN.o13591
Tamaño Vectores:11 (4 B)
Tiempo:0.002805386       / Tamaño Vectores:11
/ V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) /
/ V1[1]+V2[1]=V3[1](1.200000+1.000000=2.200000) /
/ V1[2]+V2[2]=V3[2](1.300000+0.900000=2.200000) /
/ V1[3]+V2[3]=V3[3](1.400000+0.800000=2.200000) /
/ V1[4]+V2[4]=V3[4](1.500000+0.700000=2.200000) /
/ V1[5]+V2[5]=V3[5](1.600000+0.600000=2.200000) /
/ V1[6]+V2[6]=V3[6](1.700000+0.500000=2.200000) /
/ V1[7]+V2[7]=V3[7](1.800000+0.400000=2.200000) /
/ V1[8]+V2[8]=V3[8](1.900000+0.300000=2.200000) /
/ V1[9]+V2[9]=V3[9](2.000000+0.200000=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

```
void InicializarVectores(int ini, int fin, int tam){
  for(int i=ini; i<fin; ++i){
    v1[i] = N*0.1+i*0.1;
    v2[i] = N*0.1-i*0.1;
  }
}

void SumaVectores(int ini, int fin){
  for(int i=ini; i<fin; ++i)
    v3[i] = v1[i] + v2[i];
}
```

```
#pragma omp parallel
{
  #pragma omp sections
  {
    #pragma omp section
      InicializarVectores(INI_0, INI_0+N_ELEM_THREAD, N);
    #pragma omp section
      InicializarVectores(INI_1, INI_1+N_ELEM_THREAD, N);
    #pragma omp section
      InicializarVectores(INI_2, INI_2+N_ELEM_THREAD, N);
    #pragma omp section
      InicializarVectores(INI_3, N, N);
  }



  t_ini = omp_get_wtime();

  #pragma omp sections
  {
    #pragma omp section
      SumaVectores(INI_0, INI_0+N_ELEM_THREAD);
    #pragma omp section
      SumaVectores(INI_1, INI_1+N_ELEM_THREAD);
    #pragma omp section
      SumaVectores(INI_2, INI_2+N_ELEM_THREAD);
    #pragma omp section
      SumaVectores(INI_3, N);
  }

  t_fin = omp_get_wtime();

}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[IgnacioSanchezHerrera nacho@debian:~/Disco2/Universidad/practicas_AC/bp1/ejer8] 2019-03-26 martes
$gcc -O2 -fopenmp SumaVectoresC_parallelSections.c -o SumaVectoresC_parallelSections
```

```
$echo 'bp1/ejer8/SumaVectoresC_parallelSections 8' | qsub -q ac
14307.atcgrid
```

```
[IgnacioSanchezHerrera E3estudiante25@atcgrid:~/bp1/ejer8] 2019-03-26 martes
$cat STDIN.o14307
Tamaño Vectores:8 (4 B)
Tiempo:0.000014072       / 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) /
```

```
[IgnacioSanchezHerrera E3estudiante25@atcgrid:~] 2019-03-26 martes
$echo 'bp1/ejer8/SumaVectoresC_parallelSections 11' | qsub -q ac
14308.atcgrid
```

```
[IgnacioSanchezHerrera E3estudiante25@atcgrid:~/bp1/ejer8] 2019-03-26 martes
$cat STDIN.o14308
Tamaño Vectores:11 (4 B)
Tiempo:0.002514751       / Tamaño Vectores:11
/ V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) /
/ V1[1]+V2[1]=V3[1](1.200000+1.000000=2.200000) /
/ V1[2]+V2[2]=V3[2](1.300000+0.900000=2.200000) /
/ V1[3]+V2[3]=V3[3](1.400000+0.800000=2.200000) /
/ V1[4]+V2[4]=V3[4](1.500000+0.700000=2.200000) /
/ V1[5]+V2[5]=V3[5](1.600000+0.600000=2.200000) /
/ V1[6]+V2[6]=V3[6](1.700000+0.500000=2.200000) /
/ V1[7]+V2[7]=V3[7](1.800000+0.400000=2.200000) /
/ V1[8]+V2[8]=V3[8](1.900000+0.300000=2.200000) /
/ V1[9]+V2[9]=V3[9](2.000000+0.200000=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:

   La implementación del ejercicio 7 utiliza tantas hebras como indique la variable OMP_NUM_THREADS, mientras que la implementación del ejercicio 8 usa tantas hebras como directivas *section* incluyamos. Sin embargo, en ambos casos el máximo de hebras que se pueden usar, es el mismo.

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:

PC:

Nº de Componentes	T. secuencial Vect. Globales 1 thread/core	T. paralelo (versión for) 2 threads/cores	T. paralelo (versión sections) 2 threads/cores
16384	0.000635116	0.000714968	0.000471683
32768	0.000210137	0.001410038	0.000494671
65536	0.000253506	0.000689649	0.000847689
131072	0.000383991	0.001570792	0.000636097
262144	0.001378063	0.001222993	0.001517198
524288	0.001591422	0.001513234	0.001895328
1048576	0.003372394	0.002679722	0.002730258
2097152	0.008351292	0.007621834	0.005656960
4194304	0.008640422	0.010073193	0.009818901
8388608	0.016990711	0.022958051	0.018323672
16777216	0.033171477	0.036140537	0.036901998
33554432	0.062331649	0.071607872	0.068841583
67108864	0.199839912	0.145291887	0.146019495

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	0.000429852	0.001138790	0.001881473
32768	0.000265312	0.000971534	0.001449351
65536	0.000350989	0.001076182	0.002127709
131072	0.00113173	0.001307608	0.003329625
262144	0.001204485	0.001584041	0.005030553
524288	0.002698951	0.002366032	0.009473459
1048576	0.005767551	0.003646625	0.015388751
2097152	0.010631061	0.004773756	0.025917484
4194304	0.01915922	0.011095268	0.051959002
8388608	0.036726446	0.018151679	0.097995573
16777216	0.073386948	0.034357449	Segmentation fault
33554432	0.145871697	0.075388644	Segmentation fault
67108864	0.273920916	0.143431236	Segmentation fault