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): David Carrasco Chicharro
Grupo de prácticas: B1
Fecha de entrega: 27/03/2018
Fecha evaluación en clase: 09/04/2018

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_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());
}
```

```
12
13  int main()  {
14
15      #pragma omp parallel sections
16    {
17       #pragma omp section
18          (void) funcA();
19       #pragma omp section
20          (void) funcB();
21    }
22    return 0;
23  }
```

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`

```
1   #include <stdio.h>
2   #include <omp.h>
3
4   int main()  {
5     int n = 9, i, a, b[n];
6
7     for (i=0; i<n; i++)   b[i] = -1;
8     #pragma omp parallel
9     {
10      #pragma omp single
11      {
12        printf("Introduce valor de inicialización a: ");
13        scanf("%d", &a );
14        printf("Single ejecutada por el thread %d\n",
15            omp_get_thread_num());
16      }
17
18      #pragma omp for
19      for (i=0; i<n; i++)
20        b[i] = a;
21
22      #pragma omp single
23      {
24        for (i=0 ; i<n ; i++)
25          printf("b[%d] = %d\tEjecutado por la hebra %d\n",
26          i, b[i], omp_get_thread_num());
27        printf("\n");
28      }
29    }
30
31    return 0;
32  }
```

CAPTURAS DE PANTALLA:

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP1/ej2:
2018-03-19 lunes
$ ./singleModificado
Introduce valor de inicialización a: 23
Single ejecutada por el thread 0
b[0] = 23       Ejecutado por la hebra 1
b[1] = 23       Ejecutado por la hebra 1
b[2] = 23       Ejecutado por la hebra 1
b[3] = 23       Ejecutado por la hebra 1
b[4] = 23       Ejecutado por la hebra 1
b[5] = 23       Ejecutado por la hebra 1
b[6] = 23       Ejecutado por la hebra 1
b[7] = 23       Ejecutado por la hebra 1
b[8] = 23       Ejecutado por la hebra 1
```

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP1/ej2:
2018-03-19 lunes
$ ./singleModificado
Introduce valor de inicialización a: 10
Single ejecutada por el thread 2
b[0] = 10       Ejecutado por la hebra 0
b[1] = 10       Ejecutado por la hebra 0
b[2] = 10       Ejecutado por la hebra 0
b[3] = 10       Ejecutado por la hebra 0
b[4] = 10       Ejecutado por la hebra 0
b[5] = 10       Ejecutado por la hebra 0
b[6] = 10       Ejecutado por la hebra 0
b[7] = 10       Ejecutado por la hebra 0
b[8] = 10       Ejecutado por la hebra 0
```

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());
    }

```

```
18        #pragma omp for
19        for (i=0; i<n; i++)
20          b[i] = a;
21
22        #pragma omp master
23        {
24          for (i=0 ; i<n ; i++)
25            printf("b[%d] = %d\tEjecutado por la hebra master %d\n",
26            i, b[i], omp_get_thread_num());
27          printf("\n");
28        }
29      }
30
31      return 0;
32    }
```

CAPTURAS DE PANTALLA:

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP1/ej3:
2018-03-19 lunes
$ ./singleModificado2
Introduce valor de inicialización a: 23
Single ejecutada por el thread 2
b[0] = 23       Ejecutado por la hebra master 0
b[1] = 23       Ejecutado por la hebra master 0
b[2] = 23       Ejecutado por la hebra master 0
b[3] = 23       Ejecutado por la hebra master 0
b[4] = 23       Ejecutado por la hebra master 0
b[5] = 23       Ejecutado por la hebra master 0
b[6] = 23       Ejecutado por la hebra master 0
b[7] = 23       Ejecutado por la hebra master 0
b[8] = 23       Ejecutado por la hebra master 0
```

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP1/ej3:
2018-03-19 lunes
$ ./singleModificado2
Introduce valor de inicialización a: 55
Single ejecutada por el thread 1
b[0] = 55       Ejecutado por la hebra master 0
b[1] = 55       Ejecutado por la hebra master 0
b[2] = 55       Ejecutado por la hebra master 0
b[3] = 55       Ejecutado por la hebra master 0
b[4] = 55       Ejecutado por la hebra master 0
b[5] = 55       Ejecutado por la hebra master 0
b[6] = 55       Ejecutado por la hebra master 0
b[7] = 55       Ejecutado por la hebra master 0
b[8] = 55       Ejecutado por la hebra master 0
```

RESPUESTA A LA PREGUNTA:

En el ejercicio anterior el bloque de la directiva *single* era ejecutada por cualquier hebra del proceso, pero en este siempre es la hebra *master* (que siempre lleva asociada el número 0) la que ejecuta el sección de código con la directiva *single*.

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:

   Una barrera es un punto en el código en el que los *threads* se esperan entre sí. Al eliminar dicha barrera la variable `suma` no almacena siempre todos los valores calculados en cada variable `sumalocal`, ya que cada hebra termina sin esperar al resto y puede no darle tiempo a realizar el cálculo, con lo que se imprime un valor incorrecto.

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:

```
David Carrasco Chicharro B1estudiante4@atcgrid:~:2018-03-19 lunes
$ echo 'time ./Listado1 10000000' | qsub -q ac
67197.atcgrid
David Carrasco Chicharro B1estudiante4@atcgrid:~:2018-03-19 lunes
$ cat STDIN.o67197
Tiempo(seg.):0.056382006          / 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=20000
00.000000) /
David Carrasco Chicharro B1estudiante4@atcgrid:~:2018-03-19 lunes
$ cat STDIN.e67197

real    0m0.167s
user    0m0.057s
sys     0m0.107s
```

   El tiempo de CPU del usuario es 0,057s (tiempo en ejecución en el espacio de usuario).
   El tiempo de CPU del sistema es 0,107s (tiempo en el nivel del kernel del S.O.).
   La suma de ambos tiempos es el tiempo de CPU, y suponen 0,164 mientras que el tiempo real (*elapsed time)* es 0,167s.

   0,164s < 0,167s → La suma de los tiempos de CPU del usuario y del sistema es menor que el tiempo real.

   Además tenemos que el tiempo asociado a las esperas debidas a E/S o asociados a la ejecución de otros programas en el sistema es de 0,003s (3ms).

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:

```
David Carrasco Chicharro B1estudiante4@atcgrid:~:2018-03-19 lunes
$ echo 'time ./Listado1 10' | qsub -q ac
67288.atcgrid
David Carrasco Chicharro B1estudiante4@atcgrid:~:2018-03-19 lunes
$ cat STDIN.o67288
Tiempo(seg.):0.000002823         / 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=
2.000000) /
```

```
David Carrasco Chicharro B1estudiante4@atcgrid:~:2018-03-19 lunes
$ echo 'time ./Listado1 10000000' | qsub -q ac
67197.atcgrid
David Carrasco Chicharro B1estudiante4@atcgrid:~:2018-03-19 lunes
$ cat STDIN.o67197
Tiempo(seg.):0.056382006         / 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=20000
00.000000) /
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Cálculo de MIPS

Tal y como se puede apreciar en el código en ensamblador hay 6 instrucciones en la etiqueta .L5, que corresponden a la suma del cálculo de los vectores.

Para 10 componentes en el vector:

$$MIPS = \frac{Num\,Instr}{T_{CPU}\cdot 10^6} = \frac{3 + \sum_{0}^{9} 6 + 2}{2{,}823\cdot 10^{-6}\cdot 10^6} = \frac{65}{2{,}823} = 23{,}025$$

Para 10000000 componentes:

$$MIPS = \frac{Num\,Instr}{T_{CPU}\cdot 10^6} = \frac{3 + \sum_{0}^{9999999} 6 + 2}{0{,}056382006\cdot 10^6} = \frac{60000005}{56382{,}006} = 1064{,}17$$

Cálculo de MFLOPS

Para el cálculo de MFLOPS hay que tener en cuenta aquellas instrucciones en las que aparece %xmm0, que son un total de 3.

Para 10 componentes:

$$MFLOPS = \frac{Num\,opers\,coma\,flotante}{T_{CPU}\cdot 10^6} = \frac{\sum_{0}^{9} 3}{2{,}823\cdot 10^{-6}\cdot 10^6} = \frac{30}{2{,}823} = 10{,}627$$

Para 10000000 componentes:

$$MFLOPS = \frac{Num\,opers\,coma\,flotante}{T_{CPU}\cdot 10^6} = \frac{\sum_{0}^{9999999} 3}{0{,}056382006\cdot 10^6} = \frac{30000000}{56382{,}006} = 532{,}08$$

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

```
70          call    clock_gettime
71          xorl    %eax, %eax
72          .p2align 4,,10
73          .p2align 3
74      .L5:
75          movsd   v1(%rax), %xmm0
76          addq    $8, %rax
77          addsd   v2-8(%rax), %xmm0
78          movsd   %xmm0, v3-8(%rax)
79          cmpq    %rax, %rbx
80          jne .L5
81      .L6:
82          leaq    16(%rsp), %rsi
83          xorl    %edi, %edi
84          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

```
10  #include <stdlib.h>
11  #include <stdio.h>
12
13  #ifdef _OPENMP
14    #include <omp.h>    // biblioteca para usar OpenMP
15  #else
16    #define omp_get_thread_num() 0
17    #define omp_get_num_threads() 1
18  #endif
19  //#define PRINTF_ALL    // comentar para quitar el printf
20
21  #define VECTOR_GLOBAL
22  #define MAX 33554432      //=2^25
23
24  double v1[MAX], v2[MAX], v3[MAX];
25
26
27  int main(int argc, char** argv){
28    int i;
29    double t_ini, t_fin, t_elapsed; //para tiempo de ejecución
30
31    //Leer argumento de entrada (nº de componentes del vector)
32    if (argc<2){
33      printf("Faltan nº componentes del vector\n");
34      exit(-1);
35    }
36
37    unsigned int N = atoi(argv[1]);    // Máximo N =2^32-1=4294967295
38
39    if (N>MAX) N=MAX;
40
```

```
41      //Inicializar vectores
42      #pragma omp parallel for
43      for(i=0; i<N; i++){
44        v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de N
45      }
46
47      t_ini = omp_get_wtime();
48
49      #pragma omp parallel for
50      for(i=0; i<N; i++)
51        v3[i] = v1[i] + v2[i];
52
53      t_fin = omp_get_wtime();
54      t_elapsed = t_fin - t_ini;
55
56
57      //Imprimir resultado de la suma y el tiempo de ejecución
58      #ifdef PRINTF_ALL
59      printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",t_elapsed,N);
60      for(i=0; i<N; i++)
61        printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",i,i,i,v1[i],v2[i],v3[i]);
62      #else
63        printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",t_elapsed,N);
64        for (int i=0; i<N ; i++)
65          printf("V1[%d]+V2[%d]=V3[%d] \t(%8.6f+%8.6f=%8.6f)\n",i,i,i,v1[i],v2[i],v3[i]);
66      #endif
67
68      return 0;
69    }
70
```

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

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP1/ej7:
2018-03-19 lunes
$ gcc -O2 -fopenmp -o Listado1_ej7 Listado1_ej7.c
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP1/ej7:
2018-03-19 lunes
$ ./Listado1_ej7 8
Tiempo(seg.):0.000034099         / 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)
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP1/ej7:
2018-03-19 lunes
$ ./Listado1_ej7 11
Tiempo(seg.):0.000007691         / 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()`.

RESPUESTA: Captura que muestre el código fuente implementado

```
10    #include <stdlib.h>
11    #include <stdio.h>
12
13    #ifdef _OPENMP
14      #include <omp.h>    // biblioteca para usar OpenMP
15    #else
16      #define omp_get_thread_num() 0
17      #define omp_get_num_threads() 1
18    #endif
19    //#define PRINTF_ALL
20
21    #define VECTOR_GLOBAL
22    #define MAX 33554432      //=2^25
23
24    double v1[MAX], v2[MAX], v3[MAX];
25
26
27  int main(int argc, char** argv){
28    int i, j, k, l;
29    double t_ini, t_fin, t_elapsed; //para tiempo de ejecución
30
31    //Leer argumento de entrada (nº de componentes del vector)
32    if (argc<2){
33      printf("Faltan nº componentes del vector\n");
34      exit(-1);
35    }
36
37    unsigned int N = atoi(argv[1]);
38
39    if (N>MAX) N=MAX;
40
41    //Inicializar vectores
42    #pragma omp parallel sections
43    {
44      #pragma omp section
45      for(i=0; i<N/4; i++){
46        v1[i] = N*0.1+i*0.1;
47        v2[i] = N*0.1-i*0.1;
48      }
49
50      #pragma omp section
51      for(j=N/4; j<N/2; j++){
52        v1[j] = N*0.1+j*0.1;
53        v2[j] = N*0.1-j*0.1;
54      }
55
56      #pragma omp section
57      for(k=N/2; k<3*N/4; k++){
58        v1[k] = N*0.1+k*0.1;
59        v2[k] = N*0.1-k*0.1;
60      }
61
62      #pragma omp section
63      for(l=3*N/4; l<N; l++){
64        v1[l] = N*0.1+l*0.1;
65        v2[l] = N*0.1-l*0.1;
66      }
67    }
```

```
68
69      t_ini = omp_get_wtime();
70      #pragma omp parallel sections
71      {
72        #pragma omp section
73        for(i=0; i<N/4; i++)
74          v3[i] = v1[i] + v2[i];
75
76        #pragma omp section
77        for(j=N/4; j<N/2; j++)
78          v3[j] = v1[j] + v2[j];
79
80        #pragma omp section
81        for(k=N/2; k<3*N/4; k++)
82          v3[k] = v1[k] + v2[k];
83
84        #pragma omp section
85        for(l=3*N/4; l<N; l++)
86          v3[l] = v1[l] + v2[l];
87      }
88
89      t_fin = omp_get_wtime();
90      t_elapsed = t_fin - t_ini;
91
92      //Imprimir resultado de la suma y el tiempo de ejecución
93      #ifdef PRINTF_ALL
94      printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",t_elapsed,N);
95      for(i=0; i<N; i++)
96        printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
97          i,i,i,v1[i],v2[i],v3[i]);
98      #else
99        printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",t_elapsed,N);
100       for (int i=0; i<N ; i++)
101         printf("V1[%d]+V2[%d]=V3[%d] \t(%8.6f+%8.6f=%8.6f)\n",
102           i,i,i,v1[i],v2[i],v3[i]);
103     #endif
104
105     return 0;
```

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

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP1/
ej8:2018-03-20 martes
$ gcc -O2 -fopenmp -o Listado1_ej8 Listado1_ej8.cDavid Carrasco Chicharro
 david@david:~/Uni/AC/Practicas/5.Realizadas/BP1/ej8:2018-03-20 martes
$ ./Listado1_ej8 8
Tiempo(seg.):0.000022577         / 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)
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP1/
ej8:2018-03-20 martes
$ ./Listado1_ej8 11
Tiempo(seg.):0.000021500         / 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:

   En el ejercicio 7 como se utilizan 4 cores y 4 threads, pues es el máximo recurso del que dispone mi PC.

   En el ejercicio 8 el número de threads que ejecutan el trabajo coincide con el número de sections, es decir, 4 en este caso, que es también el máximo disponible de mi computadora tal y como se ha mencionado anteriormente.

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. Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute código que imprima todos los componentes del resultado cuando este número sea elevado.

    RESPUESTA:

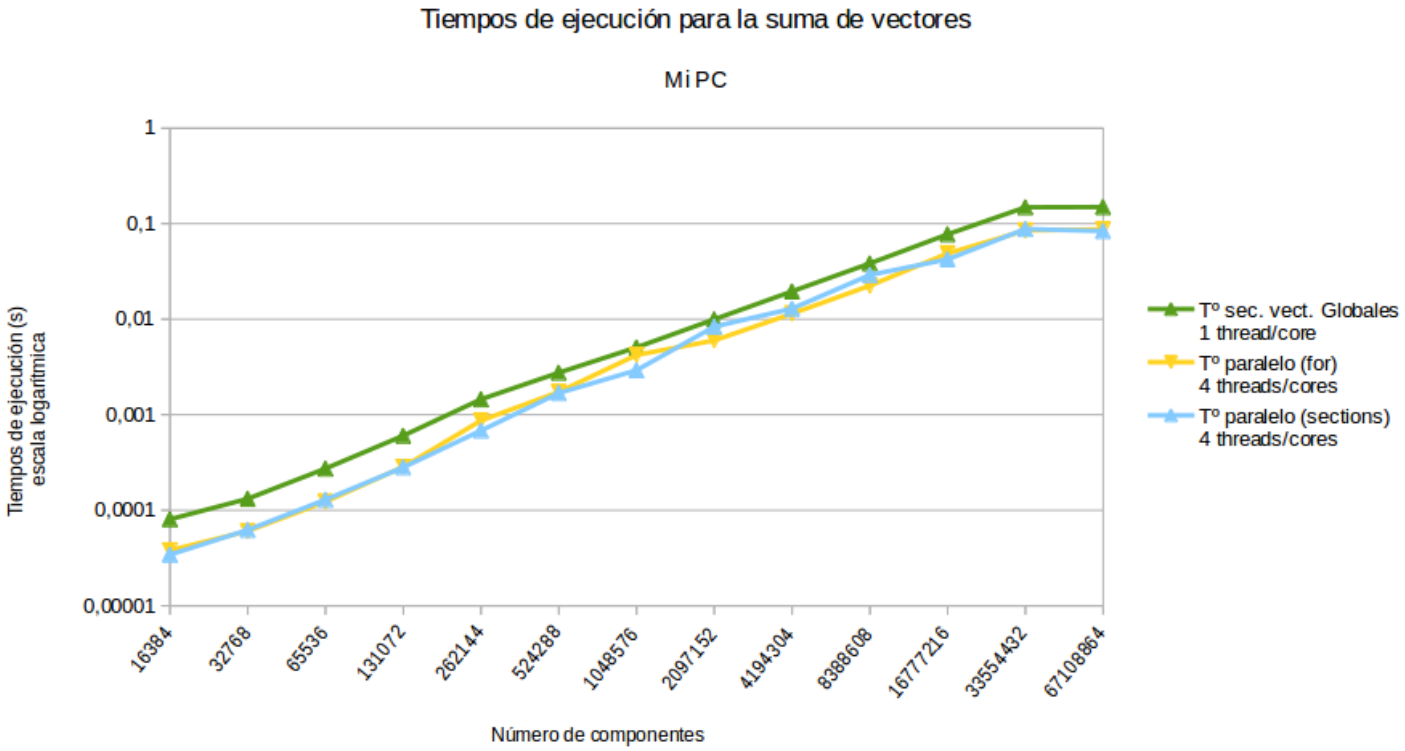
Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados, que debe coincidir con el número de cores físicos utilizados.

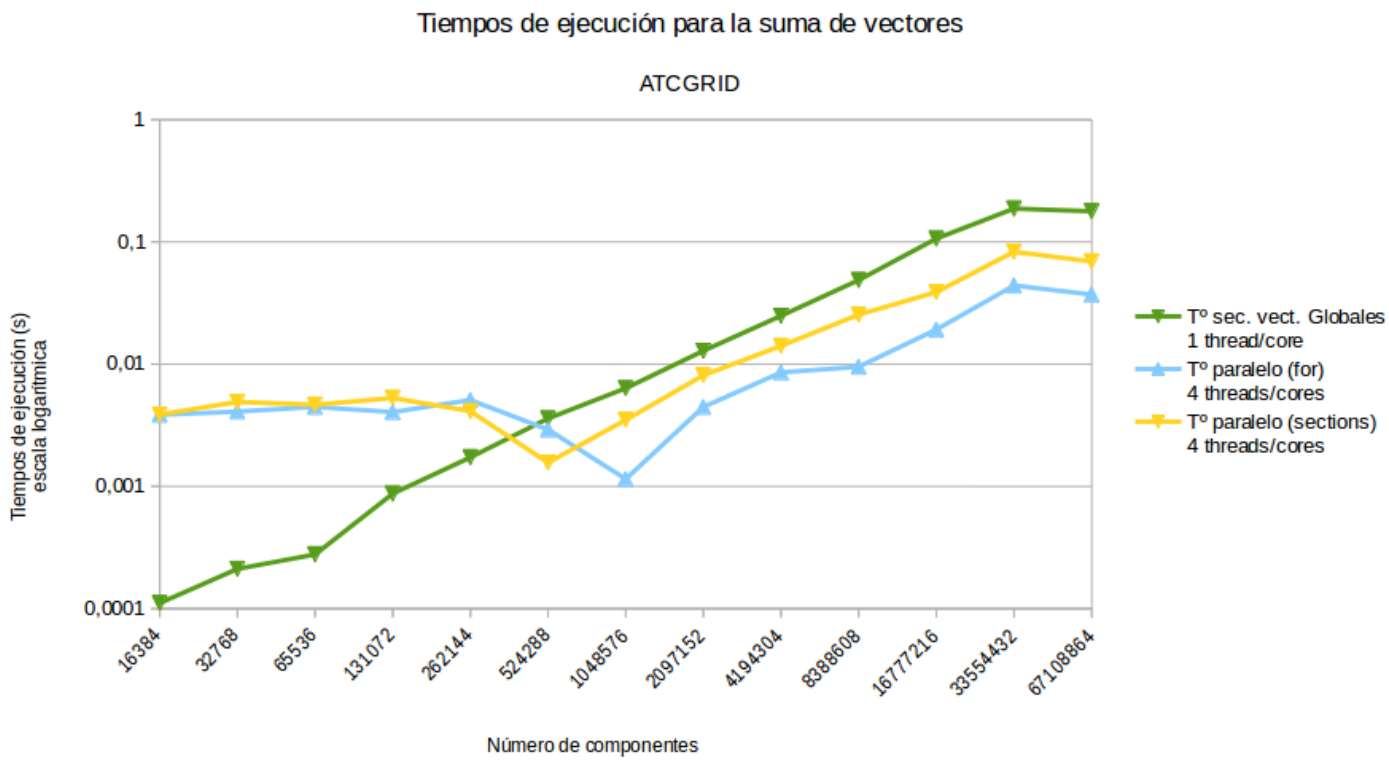
Nº de Componentes	**T. secuencial vect. Globales** **1 thread/core**	**T. paralelo (versión `for`)** **4 threads/cores**	**T. paralelo (versión `sections`)** **4 threads/cores**
16384	0.000080480	0.000038053	0.000034285
32768	0.000131934	0.000060627	0.000062217
65536	0.000273266	0.000122892	0.000129112
131072	0.000603463	0.000284586	0.000281665
262144	0.001450765	0.000870184	0.000682964
524288	0.002753120	0.001729780	0.001687166
1048576	0.005067706	0.004220881	0.002921372
2097152	0.009961358	0.005991540	0.008363337
4194304	0.019539669	0.011442157	0.012903201
8388608	0.038369076	0.022442108	0.029028983
16777216	0.077345143	0.048775790	0.042345660
33554432	0.148359392	0.084837487	0.087889059
67108864	0.149115025	0.087264811	0.083460786

Ejecución en mi 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	0,000110855	0,003844371	0,003867872
32768	0,000211253	0,004089086	0,004904039
65536	0,00027728	0,004461302	0,004647095
131072	0,000870926	0,004047087	0,005280385
262144	0,001728536	0,00507704	0,004120003
524288	0,003588514	0,002912226	0,001561533
1048576	0,006346683	0,001147501	0,003496595
2097152	0,012832569	0,004456311	0,008088511
4194304	0,024814317	0,008546806	0,014089909
8388608	0,048686425	0,009505896	0,025376134
16777216	0,106376531	0,019175385	0,038794261
33554432	0,187116624	0,043993953	0,08289305
67108864	0,177906927	0,037091077	0,069129485

Ejecución en atcgrid





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.

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` 4 Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	0,004	0,000	0,003	0,006	0,005	0,005
131072	0,004	0,000	0,003	0,005	0,005	0,004
262144	0,009	0,009	0,000	0,007	0,007	0,011
524288	0,015	0,005	0,010	0,008	0,004	0,017
1048576	0,022	0,004	0,018	0,015	0,041	0,005
2097152	0,035	0,018	0,017	0,021	0,029	0,037
4194304	0,040	0,008	0,023	0,037	0,051	0,056
8388608	0,127	0,032	0,094	0,072	0,133	0,078
16777216	0,242	0,089	0,153	0,135	0,174	0,229
33554432	0,454	0,161	0,293	0,267	0,330	0,462
67108864	0,460	0,172	0,288	0,266	0,384	0,404

Tiempos en mi PC

Nº de Componentes	Tiempo secuencial vect. Globales 1 thread/core			Tiempo paralelo/versión `for` 4 Threads/cores		
	Elapsed	*CPU-user*	*CPU- sys*	*Elapsed*	*CPU-user*	*CPU- sys*
65536	0,006	0,001	0,003	0,011	0,172	0,015
131072	0,008	0,003	0,003	0,013	0,188	0,002
262144	0,014	0,007	0,006	0,006	0,051	0,023
524288	0,022	0,009	0,013	0,013	0,197	0,014
1048576	0,034	0,012	0,021	0,016	0,206	0,038
2097152	0,076	0,026	0,049	0,020	0,231	0,068
4194304	0,145	0,048	0,096	0,024	0,135	0,161
8388608	0,275	0,092	0,178	0,054	0,328	0,221
16777216	0,563	0,179	0,378	0,082	0,513	0,435
33554432	0,566	0,193	0,368	0,155	0,784	1,093
67108864	0,543	0,185	0,352	0,151	0,875	1,032

Tiempos en atcgrid

RESPUESTA:
Cuando se usa un sólo thread/core el tiempo de CPU (CPU-user + CPU-sys) es menor que el tiempo real del sistema. Sin embargo cuando se usa una versión en paralelo (con 4 threads en este caso), el tiempo de CPU es siempre mayor que el tiempo real. Esto es debido a que cuando se ejecuta una versión en paralelo el *elapsed time* mide lo que tarda en ejecutarse el programa desde el principio hasta el final, mientras que el tiempo de CPU mide la suma de todos los tiempos de ejecución de los diferentes *threads*, siendo dicha suma mayor que el tiempo real.