

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): Mario Garcia Marquez

Grupo de prácticas y profesor de prácticas: Maribel Garcia Arenas

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`

```
File: bucle-for.c
1  #include <omp.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main(int argc, char **argv) {
6
7      int i, n = 9;
8
9      if (argc < 2) {
10         fprintf(stderr, "\n[ERROR] - Falta nº iteraciones \n");
11         exit(-1);
12     }
13     n = atoi(argv[1]);
14
15     #pragma omp parallel for
16     for (i = 0; i < n; i++)
17         printf("thread %d ejecuta la iteración %d del bucle\n",
18             omp_get_thread_num(), i);
19     return (0);
20 }
```

RESPUESTA: Captura que muestre el código fuente `sectionsModificado.c`

```

File: sections.c
1  #include <omp.h>
2  #include <stdio.h>
3
4  void funcA() {
5      printf("En funcA: esta sección la ejecuta el thread %d\n",
6             omp_get_thread_num());
7  }
8  void funcB() {
9      printf("En funcB: esta sección la ejecuta el thread %d\n",
10             omp_get_thread_num());
11  }
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`

```

[MarioGarciaMarquez elestudiente9@atcgird:~/bp1/codigo] 2021-03-25 jueves
$srunc -p ac -A ac ./single
10
Introduce valor de inicialización a: Single ejecutada por el thread 1
Single ejecutada por el thread 0
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

```

CAPTURAS DE PANTALLA:

```

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

```

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`

File: **single.c**

```

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

```

CAPTURAS DE PANTALLA:

```

[MarioGarciaMarquez elestudiente9@atcgriid:~/bpl/codigo] 2021-03-25 jueves
$srn -p ac -A ac ./single
10
Introduce valor de inicialización a: Single ejecutada por el thread 1
Single ejecutada por el thread 0
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

```

RESPUESTA A LA PREGUNTA:

El principal cambio es que el bloque master siempre es ejecutado por la hebra numero 0 mientras que el bloque single lo puede hacer cualquiera.

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 barrier fuerza a que acaben todos los hilos evitando así una situación de carrera. Al quitarlo puede ocurrir que se imprima el resultado sin haber acabado todos los computos necesarios. Es decir, barrier hace de elemento de sincronización en este código.

1.1.1

Resto de ejercicios (usar en atcgrid la cola ac a no ser que se tenga que usar atcgrid4)

5. El programa secuencial C del Listado 1 calcula la suma de dos vectores ($v3 = v1 + v2$; $v3(i) = v1(i) + v2(i)$, $i=0, \dots, N-1$). 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:

```
[MarioGarciaMarquez elestudiente9@atcgrid:~/bp1/ejer5] 2021-03-25 jueves
$ sbatch -p ac -A ac script.sh
Submitted batch job 76338
[MarioGarciaMarquez elestudiente9@atcgrid:~/bp1/ejer5] 2021-03-25 jueves
$ cat slurm-76338.out
Tiempo(seg.):0.000413733 / Tamaño Vectores:10000 / V1[0]+V2[0]=V3[0]
(3.169472+0.120862=3.290334) / / V1[9999]+V2[9999]=V3[9999] (0.970053+2.
775853=3.745905) /

real    0m0.076s
user    0m0.001s
sys      0m0.002s
[MarioGarciaMarquez elestudiente9@atcgrid:~/bp1/ejer5] 2021-03-25 jueves
$
```

RESPUESTA:

Es menor, esto se debe a que real también representa los tiempos de espera de entrada y salida de datos cosa que no tiene presente ni sys ni 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. Incorporar **el código ensamblador de la parte de la suma de vectores** (no de todo el programa) 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):

```
[MarioGarciaMarquez elestudiente9@atcgrid:~/bp1/ejer6] 2021-04-01 jueves
$ gcc listado1.c -O2 -s
[MarioGarciaMarquez elestudiente9@atcgrid:~/bp1/ejer6] 2021-04-01 jueves
$ gcc listado1.c -O2
[MarioGarciaMarquez elestudiente9@atcgrid:~/bp1/ejer6] 2021-04-01 jueves
$ ls
a.out  assembler.png  listado1.c  listado1.s
```

```
[MarioGarciaMarquez elestudiente9@atcgrid:~/bpl/ejer6] 2021-04-01 jueves
$ssrun a.out 10
Tiempo(seg.):0.000404534 / Tamaño Vectores:10 / V1[0]+V2[0]=V3[0] (0.761543+
0.401155=1.162698) / / V1[9]+V2[9]=V3[9] (0.129158+0.400250=0.529408) /
[MarioGarciaMarquez elestudiente9@atcgrid:~/bpl/ejer6] 2021-04-01 jueves
$ssrun a.out 10000000
Tiempo(seg.):0.039665501 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0] (1
.392758+1.083805=2.476564) / / V1[9999999]+V2[9999999]=V3[9999999] (2.559253
+1.183971=3.743224) /
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

MIPS:

Dado que cada iteración supone 6 instrucciones, para calcular los mips multiplicaremos las instrucciones por el número de iteraciones y pasándolo a millones, y lo dividiremos por el tiempo. Así para 10 tenemos 0.14 MIPS y para 10 millones obtenemos 1500 MIPS.

FLOPS:

Cada iteración supone una operación de coma flotante, así obtenemos para 10 iteraciones 0.024 MFLOPS y para 10 millones de iteraciones 250 MFLOPS.

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

```
15 .L8:
14 movsd 0(%r13,%rax,8), %xmm0
13 addsd (%r12,%rax,8), %xmm0
12 movsd %xmm0, (%r14,%rax,8)
11 addq $1, %rax
10 cmpl %eax, %ebp
9 ja .L8
8 leaq 32(%rsp), %rsi
7 xorl %edi, %edi
6 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 ($v3 = v1 + v2$; $v3(i) = v1(i) + v2(i)$, $i = 0, \dots, N-1$) 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 `sp-OpenMP-for.c`


```

63 // Inicializar vectores
64 if (N < 9)
65 #pragma omp parallel for
66     for (i = 0; i < N; i++) {
67         v1[i] = N * 0.1 + i * 0.1;
68         v2[i] = N * 0.1 - i * 0.1;
69     }
70 else {
71     srand(time(0));
72 #pragma omp parallel for
73     for (i = 0; i < N; i++) {
74         v1[i] = rand() / ((double)rand());
75         v2[i] = rand() / ((double)rand());
76         // printf("%d:%f,%f/", i, v1[i], v2[i]);
77     }
78 }
79 clock_gettime(CLOCK_REALTIME, &cgt1);
80 double time = omp_get_wtime();
81
82 // Calcular suma de vectores
83 #pragma omp parallel for
84     for (i = 0; i < N; i++)
85         v3[i] = v1[i] + v2[i];
86 double time2 = omp_get_wtime();
87
88 // Imprimir resultado de la suma y el tiempo de ejecución
89 if (N < 10) {
90     printf("Tiempo(seg.):%f\t / Tamaño Vectores:%lu\n", time2 - time, N);
91     for (i = 0; i < N; i++)
92         printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n", i, i, i, v1[i],
93             v2[i], v3[i]);
94 } else
95     printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\t/ "
96         "V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) / / "
97         "V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
98         time2 - time, N, v1[0], v2[0], v3[0], N - 1, N - 1, N - 1, v1[N - 1],
99         v2[N - 1], v3[N - 1]);
100 #ifdef VECTOR_DYNAMIC
101     free(v1); // libera el espacio reservado para v1
102     free(v2); // libera el espacio reservado para v2
103     free(v3); // libera el espacio reservado para v3
104 #endif
105 return 0;
106 }

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```

[MarioGarciaMarquez elestudiente9@atcgrid:~/bp1/ejer7] 2021-04-01 jueves
$gcc -O2 -fopenmp sp-OpenMP-for.c
[MarioGarciaMarquez elestudiente9@atcgrid:~/bp1/ejer7] 2021-04-01 jueves
$ls
a.out codigo.png sp-OpenMP-for.c
[MarioGarciaMarquez elestudiente9@atcgrid:~/bp1/ejer7] 2021-04-01 jueves
$srunc a.out 8
Tiempo(seg.):0.000815 / 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) /
[MarioGarciaMarquez elestudiente9@atcgrid:~/bp1/ejer7] 2021-04-01 jueves
$srunc a.out 11
Tiempo(seg.):0.000718616 / Tamaño Vectores:11 / V1[0]+V2[0]=V3[0] (2.628352+
5.749347=8.377699) / / V1[10]+V2[10]=V3[10] (0.739614+1.386647=2.126261) /

```

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 `sp-OpenMP-sections.c`


```

65 // Inicializar vectores
66 ~ if (N < 9) {
67 ~ #pragma omp parallel sections
68 ~ {
69 ~ #pragma omp section
70 + for (i = 0; i < N; i++)
71 ~ v1[i] = N * 0.1 + i * 0.1;
72 ~ #pragma omp section
73 + for (i = 0; i < N; i++)
74 - v2[i] = N * 0.1 - i * 0.1;
75 ~ }
76 ~
77 ~ } else {
78 ~ srand(time(0));
79 ~ #pragma omp parallel sections
80 - {
81 ~ #pragma omp section
82 ~ for (i = 0; i < N; i++)
83 ~ v1[i] = rand() / ((double)rand());
84 ~ #pragma omp section
85 ~ for (i = 0; i < N; i++) {
86 ~ v2[i] = rand() / ((double)rand());
87 ~ // printf("%d:%f,%f/", i, v1[i], v2[i]);
88 ~ }
89 ~ }
90 ~ }
91 clock_gettime(CLOCK_REALTIME, &cgt1);
92 double time = omp_get_wtime();
93
94 // Calcular suma de vectores
95 #pragma omp parallel
96 - {
97 - #pragma omp sections
98 - {
99 ~ #pragma omp section
100 ~ for (i = 0; i < N; i++) {
101 ~ v3[i] = v1[i] + v2[i];
102 ~ }
103 ~
104 ~ #pragma omp section
105 ~ for (i = N / 2; i < N; i++)
106 ~ v3[i] = v1[i] + v2[i];
107 ~ }
108 ~ }

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[MarioGarciaMarquez elestudiente9@atcgrid:~/bpl/ejer8] 2021-04-01 jueves
$gcc -O2 -fopenmp sp-OpenMP-sections.c
[MarioGarciaMarquez elestudiente9@atcgrid:~/bpl/ejer8] 2021-04-01 jueves
$srunc -p ac -A ac a.out 8
Tiempo(seg.):0.000653 / 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) /
[MarioGarciaMarquez elestudiente9@atcgrid:~/bpl/ejer8] 2021-04-01 jueves
$srunc -p ac -A ac a.out 11
Tiempo(seg.):0.000662699 / Tamaño Vectores:11 / V1[0]+V2[0]=V3[0] (0.000000+
2.314580=2.314580) / / V1[10]+V2[10]=V3[10] (1.363198+0.000000=1.363198) /
```

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 cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta. NOTA: Al contestar piense sólo en el código, no piense en el computador en el que lo va a ejecutar.

RESPUESTA: En el ejercicio 7 si van a usar como un total de $\min(\text{iteraciones}, \text{threads disponibles})$ threads. Esto se debe a que for repartira las iteraciones a los threads tanto como sea posible. Sin embargo, en el 8 seran $\min(\text{secciones en el codigo}, \text{threads})$ ya que cada seccion es ejecutada por un unico thread.

10. Rellenar una tabla como la Tabla 212 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. Escribir un script para realizar las ejecuciones necesarias utilizando como base el script del seminario de BP0 (se deben imprimir en el script al menos las variables de entorno que ya se imprimen en el script de BP0). 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. Observar que el número de componentes en la tabla llega hasta 67108864.

RESPUESTA: Captura del script implementado sp-OpenMP-script10.sh

```

Alacritty
14  echo "Directorio de trabajo (en el que se ejecuta el script): $SLURM_S
    UBMIT_DIR"
15  echo "Cola: $SLURM_JOB_PARTITION"
16  echo "Nodo que ejecuta este trabajo:$SLURM_SUBMIT_HOST"
17  echo "Nº de nodos asignados al trabajo: $SLURM_JOB_NUM_NODES"
18  echo "Nodos asignados al trabajo: $SLURM_JOB_NODELIST"
19  echo "CPUs por nodo: $SLURM_JOB_CPUS_PER_NODE"
20  #Instrucciones del script para ejecutar código:
21  echo -e "\n 1. Ejecución listado1:\n"
22
23
24  echo -e "\n\nlistado1:\n"
25  for ((P=16384;P<=67108864;P=P*2))
26  do
27      echo -e "\n tam: $P"
28      ./listado1 $P
29
30  done
31
32  echo -e "\n\nsp-OpenMP-for:\n"
33  for ((P=16384;P<=67108864;P=P*2))
34  do
35      echo -e "\n tam: $P"
36      ./sp-OpenMP-for $P
37
38  done
39
40  echo -e "\n\nsp-OpenMP-sections:\n"
41  for ((P=16384;P<=67108864;P=P*2))
42  do
43      echo -e "\n tam: $P"
44      ./sp-OpenMP-sections $P
45
46  done
(END)

```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

CAPTURAS DE PANTALLA (mostrar la ejecución en atcgrid – envío(s) a la cola):

```

[MarioGarciaMarquez elestudiente9@atcgrid:~/bp1/ejer10] 2021-04-01 jueves
$ sbatch -p ac -A ac -c12 -nl --hint=nomultithread sp-OpenMP-script10.sh
Submitted batch job 77552

```

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 y cores lógicos utilizados.

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión for) 16 threads = cores lógicos = cores físicos	T. paralelo (versión sections) 2 threads = cores lógicos = cores físicos
16384	0.00004276	0.001095	0.001866
32768	0.000080010	0.00112811	0.001877
65536	0.004863	0.001155	0.0015
131072	0.000763719	0.00249	0.00163
262144	0.001789649	0.00308	0.0025914
524288	0.001688901	0.001144	0.001835215
1048576	0.004478011	0.00206	0.007033264
2097152	0.007492873	0.00389	0.008638779
4194304	0.013049351	0.006473	0.0164
8388608	0.028882984	0.01360	0.031023
16777216	0.057809930	0.023306	0.062638
33554432	0.12521	0.0429	0.1286
67108864			

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión for) 12 threads = cores lógicos = cores físicos	T. paralelo (versión sections) 2 threads = cores lógicos = cores físicos
16384	0.000114	0.000062373	0.006973639
32768	0.0002184	0.00515	0.002713095
65536	0.000450881	0.004513	0.000581
131072	0.0012222	0.004742932	0.000828516
262144	0.00135324	0.006223328	0.0041712
524288	0.0023936	0.005751267	0.0073939
1048576	0.004915	0.006126631	0.005753580
2097152	0.008518	0.006837044	0.01221959
4194304	0.0175734	0.011022851	0.025560591
8388608	0.034338507	0.017155796	0.046601009
16777216	0.3433857	0.033447322	0.046601009
33554432	0.067728712	0.061593477	0.100206062
67108864			

11. 11. Rellenar una tabla como la 13Tabla 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 (que debe coincidir con el número cores físicos y lógicos) que usan los códigos. Escribir un script para realizar las ejecuciones necesarias utilizando como base el script del seminario de BP0 (se deben imprimir en el script al menos las variables de entorno que ya se imprimen en el script de BP0) ¿El tiempo de CPU que se obtiene es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

RESPUESTA: Captura del script implementado `sp-OpenMP-script11.sh`

```
#!/bin/bash
sizes=(8388608 16777216 33554432 67108864)

for i in "${sizes[@]"; do
    time ./listado1 $i
done

for i in "${sizes[@]"; do
    time ./sp-OpenMP-fo $i
done
```

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 = 1 core lógico = 1 core físico			Tiempo paralelo/versión for 12 Threads = cores lógicos=cores físicos		
	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>
8388608	0.524	0.451	0.037	6.987	9.872	2m11.458
16777216	0.943	0.841	0.068	14	19.9	4m34.410
33554432	1.878	1.662	0.157	28	40	9m16
67108864	1.889	1.662	0.157	28	39.7	9m24.361