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): Álvaro Vega Romero
Grupo de prácticas y profesor de prácticas: B2 – Niceto Rafael Luque Sola
Fecha de entrega: 01/04/2021
Fecha evaluación en clase: 09/04/2021

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());
}
main() {

#pragma omp parallel sections
 {
 #pragma omp section
 (void) funcA();
 #pragma omp section
 (void) funcB();
 }
}
```

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 argc, char **argv)
{
 int n = 9, i, a, b[n];
 for (i=0; i<n; i++) b[i] = -1;
 #pragma omp parallel
 {
 #pragma omp single
 {
        printf("Introduce valor de inicialización a: ");
        scanf("%d", &a );
        printf("Single ejecutada por el thread %d\n",omp_get_thread_num());
 }
 #pragma omp for
        for (i=0; i<n; i++)
                b[i] = a;

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

CAPTURAS DE PANTALLA:

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer2] 2021-03-12 Friday
$gcc -O2 -fopenmp -o singleModificado singleModificado.c
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer2] 2021-03-12 Friday
$./singleModificado
Introduce valor de inicialización a: 1
Single ejecutada por el thread 1
b[0] = 1        - Thread: 7
 b[1] = 1       - Thread: 7
 b[2] = 1       - Thread: 7
 b[3] = 1       - Thread: 7
 b[4] = 1       - Thread: 7
 b[5] = 1       - Thread: 7
 b[6] = 1       - Thread: 7
 b[7] = 1       - Thread: 7
 b[8] = 1       - Thread: 7
 [AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer2] 2021-03-12 Friday
```

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 argc, char **argv)
{
 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 barrier
 #pragma omp master
 {
        for (i=0; i<n; i++)
                printf("b[%d] = %d\t - Thread: %d\n ",i,b[i], omp_get_thread_num());
        }
 }
```

CAPTURAS DE PANTALLA:

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer3] 2021-03-12 Friday
$gcc -O2 -fopenmp -o singleModificado2 singleModificado2.c
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer3] 2021-03-12 Friday
$./singleModificado2
Introduce valor de inicialización a: 12
Single ejecutada por el thread 3
b[0] = 12        - Thread: 0
 b[1] = 12       - Thread: 0
 b[2] = 12       - Thread: 0
 b[3] = 12       - Thread: 0
 b[4] = 12       - Thread: 0
 b[5] = 12       - Thread: 0
 b[6] = 12       - Thread: 0
 b[7] = 12       - Thread: 0
 b[8] = 12       - Thread: 0
 [AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer3] 2021-03-12 Friday
```

RESPUESTA A LA PREGUNTA: La hebra/thread que imprime los resultados, siempre es la maestra, la cual es la 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 podría mostrar el resultado antes de que el resto de hebras terminasen su trabajo, el cual es sumar a sumalocal cada componente del vector y luego sumalocal sumarla al resultado.

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,…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:

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer5] 2021-03-19 Friday
$gcc  -O2  SumaVectores.c -o SumaVectores -lrt
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer5] 2021-03-19 Friday
```

```
[AlvaroVega b2estudiante24@atcgrid:~/bp1/ejer5] 2021-03-19 Friday
$srun -p ac time ./SumaVectores 10000000
Tiempo:0.040924254       / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](0.358223+0.620124=0.978347) / / V1[9999999]+V2[9999999]=V3[9999999](1.045386+1.214657=2.260044) /
0.52user 0.04system 0:00.57elapsed 99%CPU (0avgtext+0avgdata 244448maxresident)k
32inputs+0outputs (0major+352minor)pagefaults 0swaps
```

RESPUESTA:

La suma es menor ya que al ejecutar un programa secuencial se cumple: Treal>=Tuser+Tsystem.

En el caso anterior sería: 0,59 >= 0,52+0,04

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):

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer5] 2021-03-19 Friday
$gcc  -O2 -S SumaVectores.c -lrt
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer5] 2021-03-19 Friday
$ls
SumaVectores  SumaVectores.c  SumaVectores.s
```

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer5] 2021-03-19 Friday
$gcc  -O2  SumaVectores.c -o SumaVectores -lrt
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer5] 2021-03-19 Friday
```

```
[ÁlvaroVega b2estudiante24@atcgrid:~/bp1/ejer5] 2021-03-19 Friday
$srun -p ac time ./SumaVectores 10
Tiempo:0.000474053       / Tamaño Vectores:10   / V1[0]+V2[0]=V3[0](1.025890+0.558789=1.584679) / / V1[9]+V2[9]=V3[9](0.325606+0.803547=1.129153) /
0.00user 0.00system 0:00.00elapsed 66%CPU (0avgtext+0avgdata 4740maxresident)k
0inputs+0outputs (0major+212minor)pagefaults 0swaps
[ÁlvaroVega b2estudiante24@atcgrid:~/bp1/ejer5] 2021-03-19 Friday
```

```
[AlvaroVega b2estudiante24@atcgrid:~/bp1/ejer5] 2021-03-19 Friday
$srun -p ac time ./SumaVectores 10000000
Tiempo:0.040924254       / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](0.358223+0.620124=0.978347) / / V1[9999999]+V2[9999999]=V3[9999999](1.045386+1.214657=2.260044) /
0.52user 0.04system 0:00.57elapsed 99%CPU (0avgtext+0avgdata 244448maxresident)k
32inputs+0outputs (0major+352minor)pagefaults 0swaps
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Tenemos 6 instrucciones que son el bucles de la suma que son: (se ejecutan en cada interacción)

```
movsd   0(%rbp,%rax), %xmm0
addsd   (%r12,%rax), %xmm0
movsd   %xmm0, (%r15,%rax)
addq    $8, %rax
cmpq    %rdx, %rax
jne     .L9
```

Y tras el bucle 2 instrucciones previas al call clock_gettime. Debajo del primer call clock_gettime, en mi caso no tengo ninguna instrucción, por lo que tengo 1 instrucción menos

Para los MFLOPS, tenemos que ver las operaciones en coma flotante y podemos ver 3 (instrucciones) en la captura adjuntada del código en ensamblador. Ahí se ve como se usa el registro %xmm0, asumimos que se usan para FLOPS

Cuando n=10

Num Instrucciones= 6*10+2=62

Tejecucion=0,00047s

MIPS=NI/(Tej*10^6) = 62/ (0,00047*10^6) = 0,1319 MIPS

NI=3*10=30

MFLOPS=30 / (Tej*10^6) = 0,064 MFLOPS

Cuando n=10M

Num Insutrucciones=6*10M+2=60000002

Tejecucion=0,0409

MIPS=1466,993 MIPS

NI = 3*10M = 30M

MFLOPS = 63829,787 MFLOPS

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

Vemos que es la parte que se encarga de la suma de vectores porque aparecen las instrucciones de suma y esta entre dos llamadas a clock_gettime. (no hay instrucciones debajo de clock_gettime como vemos en la segunda imagen)

```
.L9:
        movsd   0(%rbp,%rax), %xmm0
        addsd   (%r12,%rax), %xmm0
        movsd   %xmm0, (%r15,%rax)
        addq    $8, %rax
        cmpq    %rdx, %rax
        jne     .L9
        leaq    32(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
        movq    40(%rsp), %rax
        subq    24(%rsp), %rax


 ------------------------------------------------------------------------------
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        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 sp-OpenMP-for.c

```
65    //Inicializar vectores
66
67    #pragma omp parallel for
68    for(i=0 ; i<N ; i++)
69    {
70      v1[i]=N*0.1+i*0.1;
71      v2[i]=N*0.1-0.1*i;
72    }
73
74    cgt1 = omp_get_wtime();
75
76    //Calcular suma de vectores
77
78    #pragma omp parallel for
79    for(i=0 ; i<N; i++)
80      v3[i]=v1[i]+v2[i];
81
82    cgt2 = omp_get_wtime();
83
84    ncgt = cgt2 - cgt1;
85
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer7] 2021-03-19 Friday
$gcc -fopenmp -O2  sp-OpenMP-for.c -o sp-OpenMP-for -lrt
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer7] 2021-03-19 Friday
```

```
[ÁlvaroVega b2estudiante24@atcgrid:~/bp1/ejer7] 2021-03-19 Friday
$srun -p ac sp-OpenMP-for 8
Tiempo:0.000657234      / 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) /
[ÁlvaroVega b2estudiante24@atcgrid:~/bp1/ejer7] 2021-03-19 Friday
$srun -p ac sp-OpenMP-for 11
Tiempo:0.000514481      / Tamaño Vectores:11   / V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) / / V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
[ÁlvaroVega b2estudiante24@atcgrid:~/bp1/ejer7] 2021-03-19 Friday
```

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
67    #pragma omp parallel sections private(i)
68    {
69      #pragma omp section
70      for(i=0; i<N ;i+=4)
71      {
72          v1[i]=N*0.1+i*0.1;
73          v2[i]=N*0.1-0.1*i;
74      }
75
76      #pragma omp section
77      for(i=1; i<N ;i+=4)
78      {
79          v1[i]=N*0.1+i*0.1;
80          v2[i]=N*0.1-0.1*i;
81      }
82
83      #pragma omp section
84      for(i=2; i<N ;i+=4)
85      {
86          v1[i]=N*0.1+i*0.1;
87          v2[i]=N*0.1-0.1*i;
88      }
89
90      #pragma omp section
91      for(i=3; i<N ;i+=4)
92      {
93          v1[i]=N*0.1+i*0.1;
94          v2[i]=N*0.1-0.1*i;
95      }
96    }
97
98    cgt1 = omp_get_wtime();
99
```

```
100   //Calcular suma de vectores
101
102   #pragma omp parallel sections private(i)
103   {
104     #pragma omp section
105     for(i=0; i<N ;i+=4)
106     {
107         v3[i]=v2[i]+v1[i];
108     }
109
110     #pragma omp section
111     for(i=1; i<N ;i+=4)
112     {
113         v3[i]=v2[i]+v1[i];
114     }
115
116     #pragma omp section
117     for(i=2; i<N ;i+=4)
118     {
119         v3[i]=v2[i]+v1[i];
120     }
121
122     #pragma omp section
123     for(i=3; i<N ;i+=4)
124     {
125         v3[i]=v2[i]+v1[i];
126     }
127   }
128
129   cgt2 = omp_get_wtime();
130
131   ncgt = cgt2 - cgt1;
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer8] 2021-03-19 Friday
$gcc -fopenmp -O2  sp-OpenMP-sections.c -o sp-OpenMP-sections -lrt
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer8] 2021-03-19 Friday
$ls
sp-OpenMP-sections  sp-OpenMP-sections.c
[AlvaroVega zuki@LAPTOP-9IDMGVGU:~/bp1/ejer8] 2021-03-19 Friday
```

```
[ÁlvaroVega b2estudiante24@atcgrid:~/bp1/ejer8] 2021-03-19 Friday
$srun -p ac sp-OpenMP-sections 8
Tiempo:0.000603516       / 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) /
[ÁlvaroVega b2estudiante24@atcgrid:~/bp1/ejer8] 2021-03-19 Friday
$srun -p ac sp-OpenMP-sections 11
Tiempo:0.000485241       / Tamaño Vectores:11   / V1[0]+V2[0]=V3[0](1.100000+1.100000=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. NOTA: Al contestar piense sólo en el código, no piense en el computador en el que lo va a ejecutar.

   RESPUESTA:

   La versión del 7: Como máximo, podríamos tener 11 hebras, asignándole a cada interacción (ya que tenemos 11 interacciones) una hebra y cuando son 8 interacciones pues 8 hebras.

   La versión del 8: Idealmente podríamos tener 4 threads como máximo, uno para cada section y por tanto tener para ambos casos 4 hebras.

10. Rellenar una tabla como la Tabla 210 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

```
#!/bin/bash

echo "Id. usuario del trabajo: $SLURM_JOB_USER"
echo "Id. del trabajo: $SLURM_JOBID"
echo "Nombre del trabajo especificado por usuario: $SLURM_JOB_NAME"
echo "Directorio de trabajo (en el que se ejecuta el script): $SLURM_SUBMIT_DIR"
echo "Cola: $SLURM_JOB_PARTITION"
echo "Nodo que ejecuta este trabajo:$SLURM_SUBMIT_HOST"
echo "Nº de nodos asignados al trabajo: $SLURM_JOB_NUM_NODES"
echo "Nodos asignados al trabajo: $SLURM_JOB_NODELIST"
echo "CPUs por nodo: $SLURM_JOB_CPUS_PER_NODE"

#Instrucciones del script para ejecutar código:
echo -e Ejecucion de SumaVectores_global, con for paralelizado y sections

for ((P=16384;P<=67108864;P=P*2))
do
        echo -e "\n Ejecutando con for con $P componentes"
        ./sp-OpenMP-for $P
        echo -e "\n Ejecutando con sections con $P componentes"
        ./sp-OpenMP-sections $P
        echo -e "\n Ejecutando secuencial con $P componentes"
        ./SumaVectoresC_global $P
done
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

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

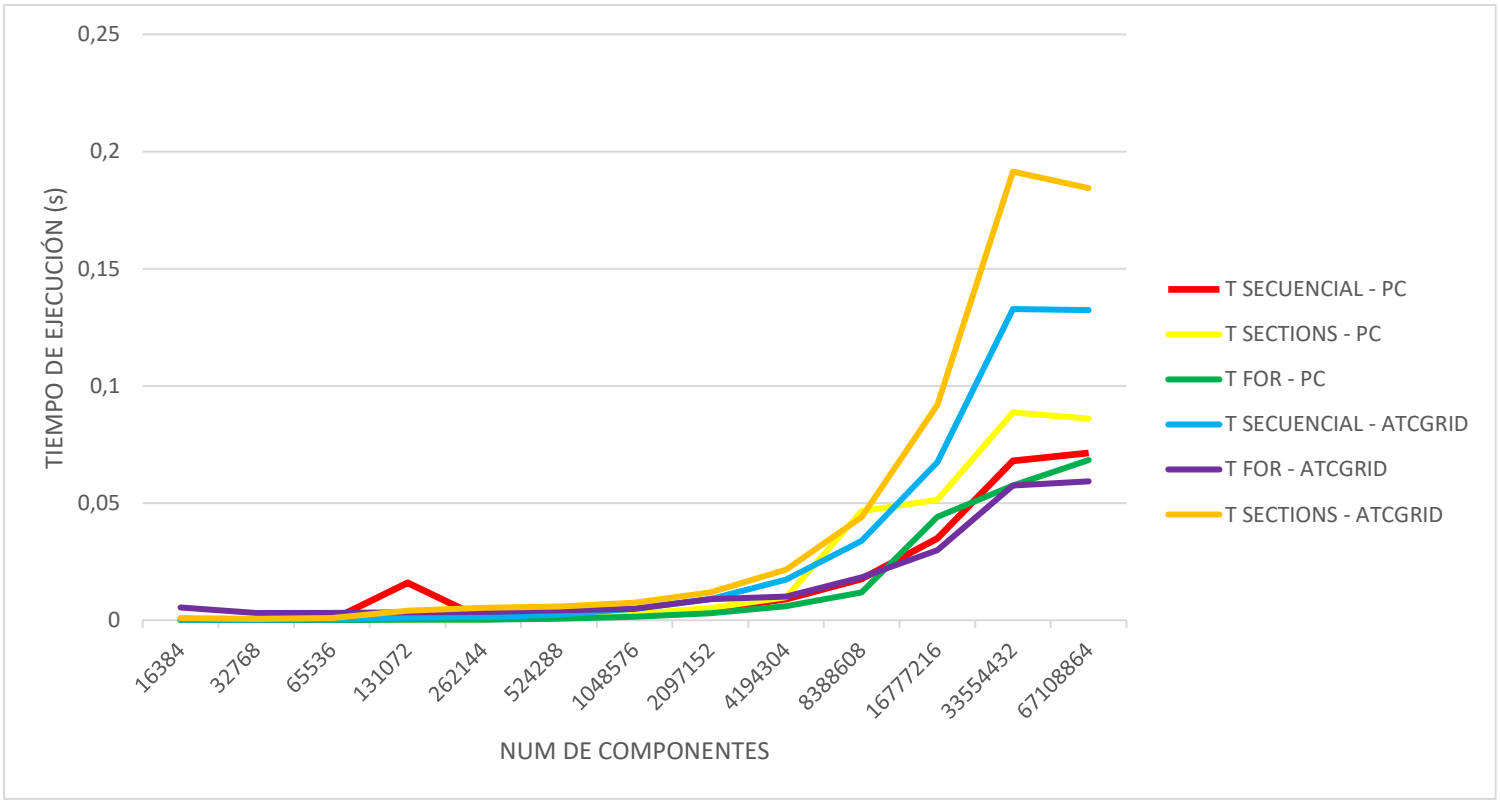
```
[ÁlvaroVega b2estudiante24@atcgrid:~/bp1/ejer10] 2021-03-26 Friday
$sbatch -pac -n1 -c12 --hint=nomultithread sp-OpenMP-script10.sh
Submitted batch job 76448
[ÁlvaroVega b2estudiante24@atcgrid:~/bp1/ejer10] 2021-03-26 Friday
$ls
SumaVectoresC_global  slurm-76448.out  sp-OpenMP-for  sp-OpenMP-script10.sh  sp-OpenMP-sections
[ÁlvaroVega b2estudiante24@atcgrid:~/bp1/ejer10] 2021-03-26 Friday
$cat slurm-76448.out
Id. usuario del trabajo: b2estudiante24
Id. del trabajo: 76448
Nombre del trabajo especificado por usuario: sp-OpenMP-script10.sh
Directorio de trabajo (en el que se ejecuta el script): /home/b2estudiante24/bp1/ejer10
Cola: ac
Nodo que ejecuta este trabajo:atcgrid.ugr.es
Nº de nodos asignados al trabajo: 1
Nodos asignados al trabajo: atcgrid1
CPUs por nodo: 24
Ejecucion de SumaVectores_global, con for paralelizado y sections

 Ejecutando con for con 16384 componentes
Tiempo:0.005452082      / Tamaño Vectores:16384        / V1[0]+V2[0]=V3[0](1638.400000+1638.400000=3276.800000) / / V1[16383]+V2[16383]=V3[16383](3276.700000+0.100000=3276.800000) /

 Ejecutando con sections con 16384 componentes
Tiempo:0.000939451      / Tamaño Vectores:16384        / V1[0]+V2[0]=V3[0](1638.400000+1638.400000=3276.800000) / / V1[16383]+V2[16383]=V3[16383](3276.700000+0.100000=3276.800000) /

 Ejecutando secuencial con 16384 componentes
```

Tabla 2 PC. 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**)** 8 threads / core	**T. paralelo (versión** sections**)** 4 threads / core
16384	0,0000391	0,000034	0,0000632
32768	0,000058	0,0000589	0,0000887
65536	0,0001383	0,00000941	0,0001324
131072	0,0159418	0,00019	0,0002306
262144	0,0004793	0,0003073	0,0003665
524288	0,0010855	0,0006905	0,0013368
1048576	0,0024311	0,0015074	0,0027233
2097152	0,0045113	0,0030005	0,0052119
4194304	0,0090014	0,0059688	0,0098241
8388608	0,0175407	0,0118947	0,0465995
16777216	0,0350017	0,044107	0,0513666
33554432	0,067977	0,0574758	0,0886634
67108864	0,0713704	0,0683439	0,0860289

Tabla 3 ATCGRID. 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**)** **12 threads / core**	**T. paralelo (versión** sections**)** **4 threads / core**
16384	0,000230856	0,005452082	0,000939451
32768	0,000264150	0,003081955	0,000596758
65536	0,00035905	0,003104832	0,000962541
131072	0,001047645	0,003395963	0,004071094
262144	0,001370548	0,003533382	0,005282596
524288	0,002447511	0,004149929	0,005787924
1048576	0,004947309	0,004913222	0,007498335
2097152	0,008979142	0,009027977	0,011906628
4194304	0,017317302	0,010016378	0,021578982
8388608	0,033868584	0,018266719	0,043969467
16777216	0,067450999	0,0299231	0,091964308
33554432	0,13277725	0,057443816	0,191405777
67108864	0,132307161	0,059261966	0,184437539



11. Rellenar una tabla como la 12Tabla 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

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

CAPTURAS DE PANTALLA (ejecución en atcgrid):

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 **¿? Threads = cores lógicos=cores físicos**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
8388608						
16777216						
33554432						
67108864						