2º curso / 2º cuatr. Grados Ingeniería Informática	**Arquitectura de Computadores (AC)** **1.1.1 Cuaderno de prácticas.** **1.1.2 Bloque Práctico 1. Programación paralela I: Directivas OpenMP** Estudiante (nombre y apellidos): Jesús Losada Arauzo Grupo de prácticas y profesor de prácticas: B2 Javier Medina

2 Parte I. 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

```
*/
int main(int argc, char **argv)
{
   int i, n = 9;

  if(argc < 2)     {
    fprintf(stderr,"\n[ERROR] - Falta nº de iteraciones \n");
         exit(-1);
     }
   n = atoi(argv[1]);
#pragma omp parallel
{
   #pragma omp for
   for (i=0; i<n; i++)
      printf("Hebra %d ejecuta la iteración %d del bucle\n",
              omp_get_thread_num(),i);
}

   return(0);
}
```

RESPUESTA: Captura que muestre el código fuente sectionsModificado.c

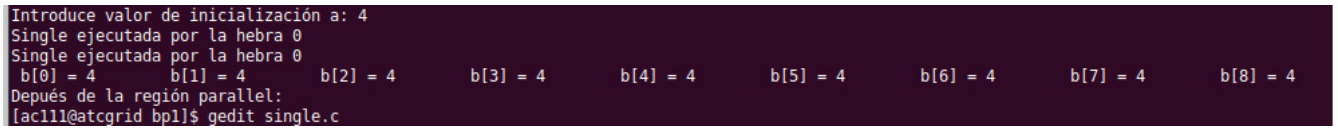
```
void funcA()
{
    printf("En funcA: esta sección la ejecuta la hebra %d\n",
          omp_get_thread_num());
}
void funcB()
{
    printf("En funcB: esta sección la ejecuta la hebra %d\n",
          omp_get_thread_num());
}

int main()
{
#pragma omp parallel
{
    #pragma omp 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

```
*/
int main()
{
   int n = 9;
   int 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 la hebra %d\n",
             omp_get_thread_num());
   }

   #pragma omp for
   for (i=0; i<n; i++)
       b[i] = a;
#pragma omp single
{
        printf("Single ejecutada por la hebra %d\n",
             omp_get_thread_num());
   #pragma omp for
   for (i=0; i<n; i++)
       printf(" b[%d] = %d\t",i,b[i]);
       printf("\n");
}
}

   printf("Depués de la región parallel:\n");

   return(0);
}
```

CAPTURAS DE PANTALLA:



La captura no es mia ya que aun teniendo el mismo codigo que el resto de mis compañeros no me compilaba

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

```
*/
int main()
{
    int n = 9;
    int 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 la hebra %d\n",
                omp_get_thread_num());
    }

    #pragma omp for

      for (i=0; i<n; i++)
        b[i] = a;


    #pragma omp master
          //printf("Master ejecutada por la hebra %d\n",omp_get_thread_num());
          for (int k = 0; k < n; k++){
                  printf(" b[%d] = %d\t",k,b[k]);
          }
          printf("\n");



}
    return(0);
}
```

CAPTURAS DE PANTALLA:

```
[ac125@atcgrid Practica1]$ gcc -fopenmp -O2 -o single singleModif2.c
[ac125@atcgrid Practica1]$ srun single
5
Introduce valor de inicialización a: Single ejecutada por la hebra 0
 b[0] = 5        b[1] = 5        b[2] = 5        b[3] = 5        b[4] = 5        b[5] = 5        b[6] = 5        b[7]
 = 5     b[8] = 5

[ac125@atcgrid Practica1]$
```

RESPUESTA A LA PREGUNTA:

Ahora solo va a ejecutar la hebra 0 el for para sacar las cosas dentro del parallel.

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 se espera a que este todo sumado a veces dara una suma incorrecta porque faltaran terminos

3 Parte II. 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:

```
jesus@jesus-HP-EliteBook-830-G7-Notebook-PC:~/AC/Practicas/Practica1$ time ./sumavect 10000000
Tiempo(seg.):0.045744290         / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0 (0.201682+0.684696
=0.886378) / / V1[9999999]+V2[9999999]=V3[9999999](0.694726+0.976769=1.671495) /

real    0m0,250s
user    0m0,186s
sys     0m0,065s
```

```
[ac125@atcgrid Practica1]$ srun --hint=nomultithread --cpus-per-task=1 time ./sumavec 10000000
Tiempo(seg.):0.134973777         / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0 (0.431624+0.069935=0.501559) / / V
1[9999999]+V2[9999999]=V3[9999999](0.523783+0.910611=1.434394) /
0.33user 0.04system 0:00.74elapsed 50%CPU (0avgtext+0avgdata 234828maxresident)k
0inputs+0outputs (0major+1549minor)pagefaults 0swaps
Tiempo(seg.):0.155344029         / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0 (0.431624+0.069935=0.501559) / / V
1[9999999]+V2[9999999]=V3[9999999](0.523783+0.910611=1.434394) /
0.33user 0.04system 0:00.76elapsed 49%CPU (0avgtext+0avgdata 234896maxresident)k
0inputs+0outputs (0major+2060minor)pagefaults 0swaps
[ac125@atcgrid Practica1]$ srun sumavec 10000000
Tiempo(seg.):0.073356685         / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0 (0.914833+0.888297=1.803130) / / V
1[9999999]+V2[9999999]=V3[9999999](0.951134+0.521196=1.472330) /
[ac125@atcgrid Practica1]$
```

RESPUESTA:

Debería ser menor igual porque el elapsed time y tiene el tiempo real como tal y en cambio el otro es solo el tiempo de ejecución

6. Generar el código ensamblador a partir del programa secuencial C del Listado 1 (ver cuaderno de BP0) 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):

```
[ac125@atcgrid Practica1]$ gcc -S -O2 sumavec_ass SumaVectores.c
gcc: warning: sumavec_ass: linker input file unused because linking not done
[ac125@atcgrid Practica1]$ ls
atomic.c     critical.c      parallel.c  single          singleModif.c  SumaVectores.c
barrier.c    critical_sin.c  sections    single.c        sumavec        SumaVectores.s
bucle-for.c  master.c        sections.c  singleModif2.c  sumavec_ass
[ac125@atcgrid Practica1]$ cat SumaVectores.s
```

```
[ac125@atcgrid Practica1]$ ls
atomic.c     critical.c      parallel.c  single          singleModif.c  SumaVectores.c
barrier.c    critical_sin.c  sections    single.c        sumavec        SumaVectores.s
bucle-for.c  master.c        sections.c  singleModif2.c  sumavec_ass
[ac125@atcgrid Practica1]$ srun sumavec 10000000
Tiempo(seg.):0.073448734         / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0 (0.955024+0.438797=1.393821) / / V
1[9999999]+V2[9999999]=V3[9999999](0.823048+0.290349=1.113397) /
[ac125@atcgrid Practica1]$ srun sumavec 100
Tiempo(seg.):0.000000810         / Tamaño Vectores:100  / V1[0]+V2[0]=V3[0 (0.438233+0.257159=0.695392) / / V1[99]+V2
[99]=V3[99](0.559842+0.056622=0.616463) /
[ac125@atcgrid Practica1]$ srun sumavec 10
Tiempo(seg.):0.000000260         / Tamaño Vectores:10   / V1[0]+V2[0]=V3[0 (0.050640+0.370931=0.421571) / / V1[9]+V2[
9]=V3[9](0.777084+0.769588=1.546673) /
[ac125@atcgrid Practica1]$ srun --hint=nomultithread --cpus-per-task=1 time ./sumavec 10000000
Tiempo(seg.):0.158834821         / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0 (0.275454+0.598473=0.873927) / / V
1[9999999]+V2[9999999]=V3[9999999](0.711938+0.524751=1.236689) /
0.33user 0.04system 0:00.74elapsed 50%CPU (0avgtext+0avgdata 234828maxresident)k
0inputs+0outputs (0major+1549minor)pagefaults 0swaps
Tiempo(seg.):0.147358098         / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0 (0.275454+0.598473=0.873927) / / V
1[9999999]+V2[9999999]=V3[9999999](0.711938+0.524751=1.236689) /
0.32user 0.04system 0:00.74elapsed 50%CPU (0avgtext+0avgdata 234896maxresident)k
0inputs+0outputs (0major+2059minor)pagefaults 0swaps
[ac125@atcgrid Practica1]$ srun --hint=nomultithread --cpus-per-task=1 time ./sumavec 10
Tiempo(seg.):0.000000360         / Tamaño Vectores:10   / V1[0]+V2[0]=V3[0 (0.629465+0.121426=0.750891) / / V1[9]+V2[
9]=V3[9](0.777679+0.401745=1.179423) /
0.00user 0.00system 0:00.00elapsed 50%CPU (0avgtext+0avgdata 580maxresident)k
0inputs+0outputs (0major+186minor)pagefaults 0swaps
Tiempo(seg.):0.000000337         / Tamaño Vectores:10   / V1[0]+V2[0]=V3[0 (0.629465+0.121426=0.750891) / / V1[9]+V2[
9]=V3[9](0.777679+0.401745=1.179423) /
0.00user 0.00system 0:00.00elapsed 0%CPU (0avgtext+0avgdata 584maxresident)k
0inputs+0outputs (0major+187minor)pagefaults 0swaps
[ac125@atcgrid Practica1]$
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

MIPS = $13*10^7/0.36 * 10^6 = 361,111111$

MFLOPS = $3*10^7/0.36 * 10^6 = 83,33333$

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

```
        .p2align 3
.L12:
        movsd   0(%r13,%rbx,8), %xmm0
        movl    %ebx, %esi
        movl    %ebx, %ecx
        movl    %ebx, %edx
        movsd   (%r12,%rbx,8), %xmm2
        movsd   (%r14,%rbx,8), %xmm1
        movl    $.LC3, %edi
        movl    $3, %eax
        addq    $1, %rbx
        call    printf
        cmpl    %ebx, %ebp
        ja      .L12
        jmp     .L13
.L5:
```

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); (4) se debe imprimir el tamaño de los vectores y el número de hilos; (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

```
[ac125@atcgrid Practica1]$ gcc -fopenmp -O2 -o sp-open sp-OpenMP-for.c
[ac125@atcgrid Practica1]$ time ./sp-open 11
Tiempo(seg.):0.001432521         / Tamaño Vectores:11   / V1[0]+V2[0]=V3[0 (0.610475+0.719651=1.33012
6) / / V1[10]+V2[10]=V3[10](0.097617+0.000763=0.098380) n hilos:0.000000/

real    0m0.004s
user    0m0.013s
sys     0m0.000s
[ac125@atcgrid Practica1]$ time ./sp-open 8
Tiempo(seg.):0.000436853         / Tamaño Vectores:8     / n hilos:5273155.000000
/ 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) /

real    0m0.003s
user    0m0.006s
sys     0m0.001s
[ac125@atcgrid Practica1]$
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

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); (4) se debe imprimir el tamaño de los vectores y el número de hilos; (5) 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

```
[ac125@atcgrid Practica1]$ gcc -fopenmp -O2 -o sp-openSec sp-OpenMP-sections.c
[ac125@atcgrid Practica1]$ time ./sp-openSec 8
Tiempo(seg.):0.009101038        / Tamaño Vectores:8     / n hilos:5274157.000000
/ 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) /

real    0m0.012s
user    0m0.064s
sys     0m0.002s
[ac125@atcgrid Practica1]$
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

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: En el ejercicio 7 se podrían usar tantas tantas thread como elementos tenga el vector, en cambio en el ejercicio 8 se podrían usar simplemente 2 por como es mi código ya que parto a la mitad el dulce for dependiendo del tamaño

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. 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

```
 GNU nano 2.3.1                Fichero: sp-OpenMP-script10.sh

#!/bin/bash
#Órdenes para el Gestor de carga de trabajo (no intercalar instrucciones del script):
#1. Asigna al trabajo un nombre
#SBATCH --job-name=helloOMP
#2. Asignar el trabajo a una cola (partición)
#SBATCH --partition=ac
#3. Asignar el trabajo a un account
#SBATCH --account=ac
#4. Para que el trabajo no comparta recursos
#SBATCH -exclusive
#5. Para que se genere un único proceso del SO que pueda usar un máximo de 12 núcleos
#SBATCH --ntasks 1 --cpus-per-task 12
#Obtener información de las variables del entorno del Gestor de carga de trabajo:
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 "Nº CPUs disponibles para el trabajo en el nodo: $SLURM_JOB_CPUS_PER_NODE"
#Instrucciones del script para ejecutar código:
echo -e "\n 1. Ejecución helloOMP una vez sin cambiar nº de threads (valor por defecto):\n"
srun ./sp-open
#srun ./sp-openSec
echo -e "\n 2. Ejecución helloOMP varias veces con distinto nº de threads:\n"
for ((P=67108864;P>0;P=P/2))
do
export OMP_NUM_THREADS=$P
echo -e "\n - Para $P threads:"
srun --hint=nomultithread ./sumavec $P
#srun --hint=noultithread ./sp-openSec $P
done
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

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

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**)** **¿?threads = cores lógicos = cores físicos**	**T. paralelo (versión** sections**)** **¿?threads = cores lógicos = cores físicos**
16384	0.000296914	Resource unavailable temporary	Resource unavailable temporary
32768	0.000437734	Resource unavailable temporary	Resource unavailable temporary
65536	0.000897533	Resource unavailable temporary	Resource unavailable temporary
131072	0.001800269	Vioalcion de segmento	Vioalcion de segmento
262144	0.003394229	Vioalcion de segmento	Vioalcion de segmento
524288	0.005219778	Vioalcion de segmento	Vioalcion de segmento

1048576	0.009257568	Vioalcion de segmento	Vioalcion de segmento
2097152	0.016055479	Vioalcion de segmento	Vioalcion de segmento
4194304	0.031340202	Vioalcion de segmento	Vioalcion de segmento
8388608	0.061933520	Vioalcion de segmento	Vioalcion de segmento
16777216	0.123269096	Vioalcion de segmento	Vioalcion de segmento
33554432	0.245683780	Vioalcion de segmento	Vioalcion de segmento
67108864	0.460919101	Vioalcion de segmento	Vioalcion de segmento

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 (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

```
 GNU nano 2.3.1                Fichero: sp-OpenMP-script11.sh

#!/bin/bash
#Órdenes para el Gestor de carga de trabajo (no intercalar instrucciones del script):
#1. Asigna al trabajo un nombre
#SBATCH --job-name=helloOMP
#2. Asignar el trabajo a una cola (partición)
#SBATCH --partition=ac
#3. Asignar el trabajo a un account
#SBATCH --account=ac
#4. Para que el trabajo no comparta recursos
#SBATCH -exclusive
#5. Para que se genere un único proceso del SO que pueda usar un máximo de 12 núcleos
#SBATCH --ntasks 1 --cpus-per-task 12
#Obtener información de las variables del entorno del Gestor de carga de trabajo:
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 "Nº CPUs disponibles para el trabajo en el nodo: $SLURM_JOB_CPUS_PER_NODE"
#Instrucciones del script para ejecutar código:
echo -e "\n 1. Ejecución helloOMP una vez sin cambiar nº de threads (valor por defecto):\n"
srun ./sp-open
#srun ./sp-openSec
echo -e "\n 2. Ejecución helloOMP varias veces con distinto nº de threads:\n"
for ((P=67108864;P>0;P=P/2))
do
export OMP_NUM_THREADS=$P
echo -e "\n - Para $P threads:"
srun --hint=nomultithread time ./sp-open $P
#srun --hint=noultithread ./sp-openSec $P
done
```

(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	00.30	0.26	0.03	19.70	0.20	2.60
16777216	00.60	0.52	0.07	37.39	0.38	5.06
33554432	01.19	1.04	0.14	46.43	0.76	6.71
67108864	02.37	2.07	0.29	46.60	1.50	6.55