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): Salvador Romero Cortés
Grupo de prácticas y profesor de prácticas: A1 Niceto Rafael Luque
Fecha de entrega:
Fecha evaluación en clase:

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

Ejercicios basados en los ejemplos del seminario práctico

1. Usar la directiva parallel combinada con directivas de trabajo compartido en los ejemplos bucle-for.c y sections.c del seminario. Incorporar el código fuente resultante al cuaderno de prácticas.

RESPUESTA: Captura que muestre el código fuente bucle-forModificado.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

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

    if (argc < 2) {
        fprintf(stderr, "\n[ERROR] - Falta el nº de iteraciones\n");
        exit(-1);
    }
    n = atoi(argv[1]);


    #pragma omp parallel for
    for (i=0; i<n; i++)
        printf("thread %d ejecuta la iteración %d del bucle\n",
            omp_get_thread_num(),i);


    return(0);
}
```

RESPUESTA: Captura que muestre el código fuente sectionsModificado.c

```c
#include <stdio.h>
#include <omp.h>

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

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

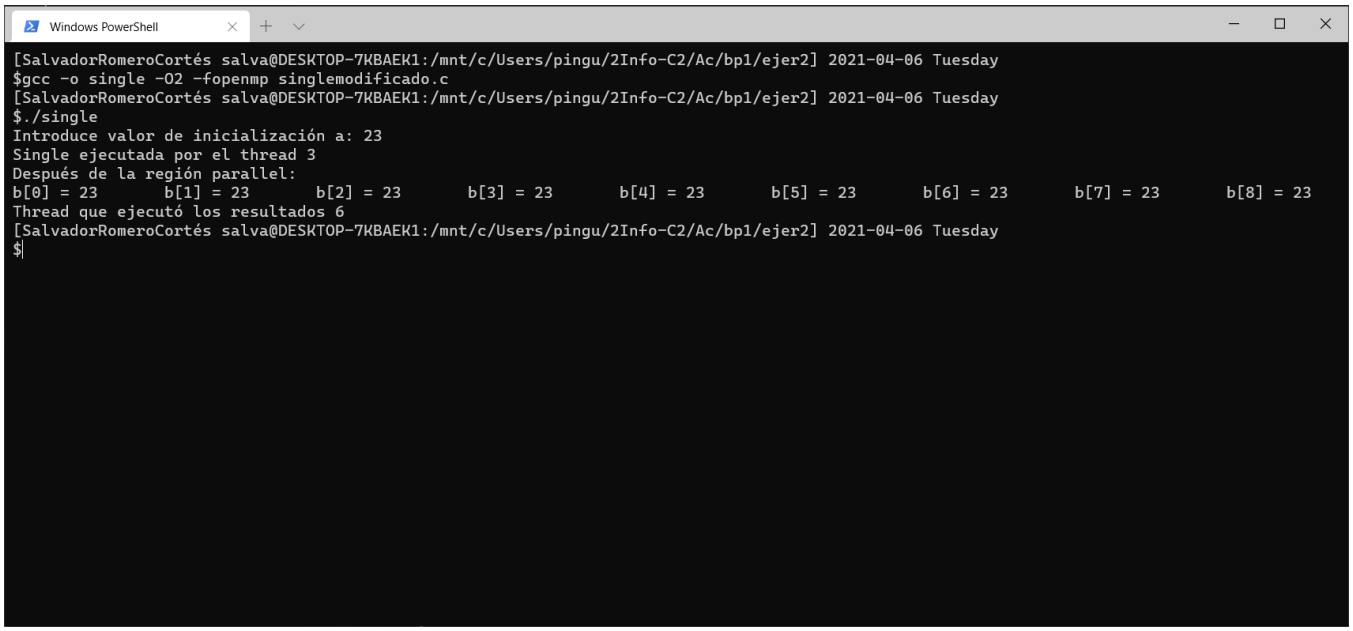
2. Imprimir los resultados del programa single.c usando una directiva single dentro de la construcción parallel en lugar de imprimirlos fuera de la región parallel. Añadir lo necesario, dentro de la nueva directiva single incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva single. Incorpore en su cuaderno de trabajo el código fuente y volcados de pantalla con los resultados de ejecución obtenidos.

   RESPUESTA: Captura que muestre el código fuente singleModificado.c

```c
#include <stdio.h>
#include <omp.h>

int main(){
    int n = 9, i, a, b[n];

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

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

        #pragma omp single
        {
            printf("Después de la región parallel:\n");
            for (i=0; i < n; i++) printf("b[%d] = %d\t",i,b[i]);
            printf("\n");
            printf("Thread que ejecutó los resultados %d\n", omp_get_thread_num());
        }
    }
}
```

CAPTURAS DE PANTALLA:

Resultados de ejecución:

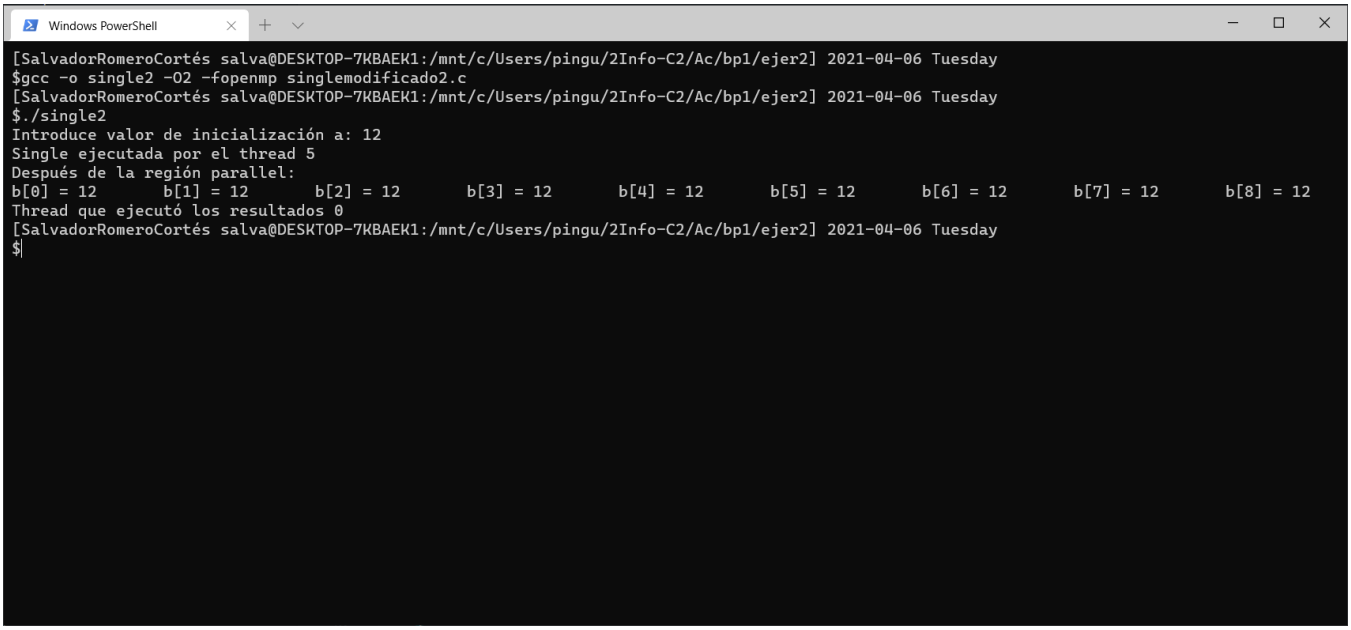


3. Imprimir los resultados del programa single.c usando una directiva master dentro de la construcción parallel en lugar de imprimirlos fuera de la región parallel. Añadir lo necesario, dentro de la nueva directiva master incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva master. Incorpore en su cuaderno el código fuente y volcados de pantalla con los resultados de ejecución obtenidos. ¿Qué diferencia observa con respecto a los resultados de ejecución del ejercicio anterior?

   RESPUESTA: Captura que muestre el código fuente singleModificado2.c

```c
#include <stdio.h>
#include <omp.h>

int main(){
    int n = 9, i, a, b[n];

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

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

        #pragma omp master
        {
            printf("Después de la región parallel:\n");
            for (i=0; i < n; i++) printf("b[%d] = %d\t",i,b[i]);
            printf("\n");
            printf("Thread que ejecutó los resultados %d\n", omp_get_thread_num());
        }
    }
}
```

CAPTURAS DE PANTALLA:



RESPUESTA A LA PREGUNTA:

La diferencia está en que en este último ejercicio, al usar la directiva master, el thread que ejecuta los resultados es siempre el thread 0, el principal.

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:

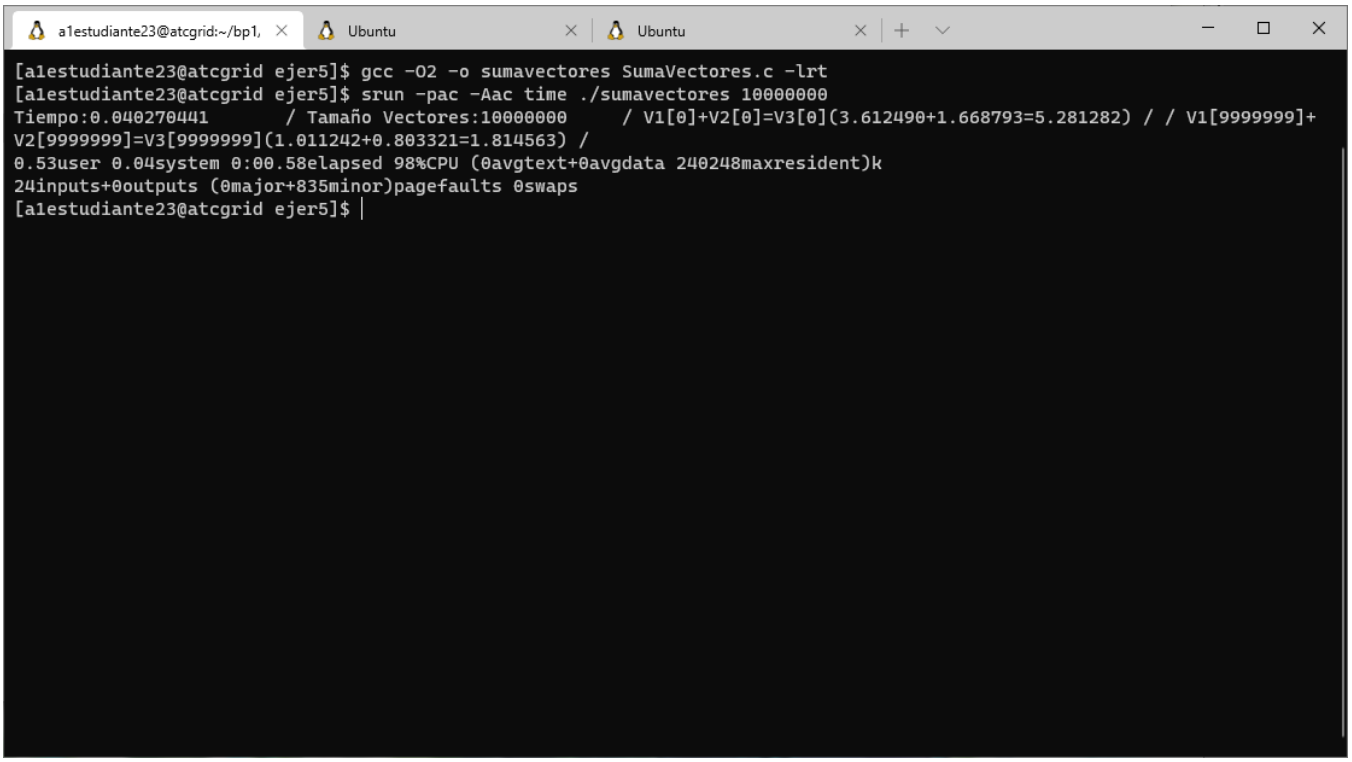
La directiva barrier hace que los threads esperen para sincronizarse. Si la eliminamos, el thread master no esperará al resto de threads y ejecutará la instrucción para mostrar por pantalla con el valor de la suma que tenga en ese momento. Esto ocurre porque la directive master no tiene una barrera implícita y por tanto es necesario expresarlo de manera explícita si queremos que funcione correctamente.

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,…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:



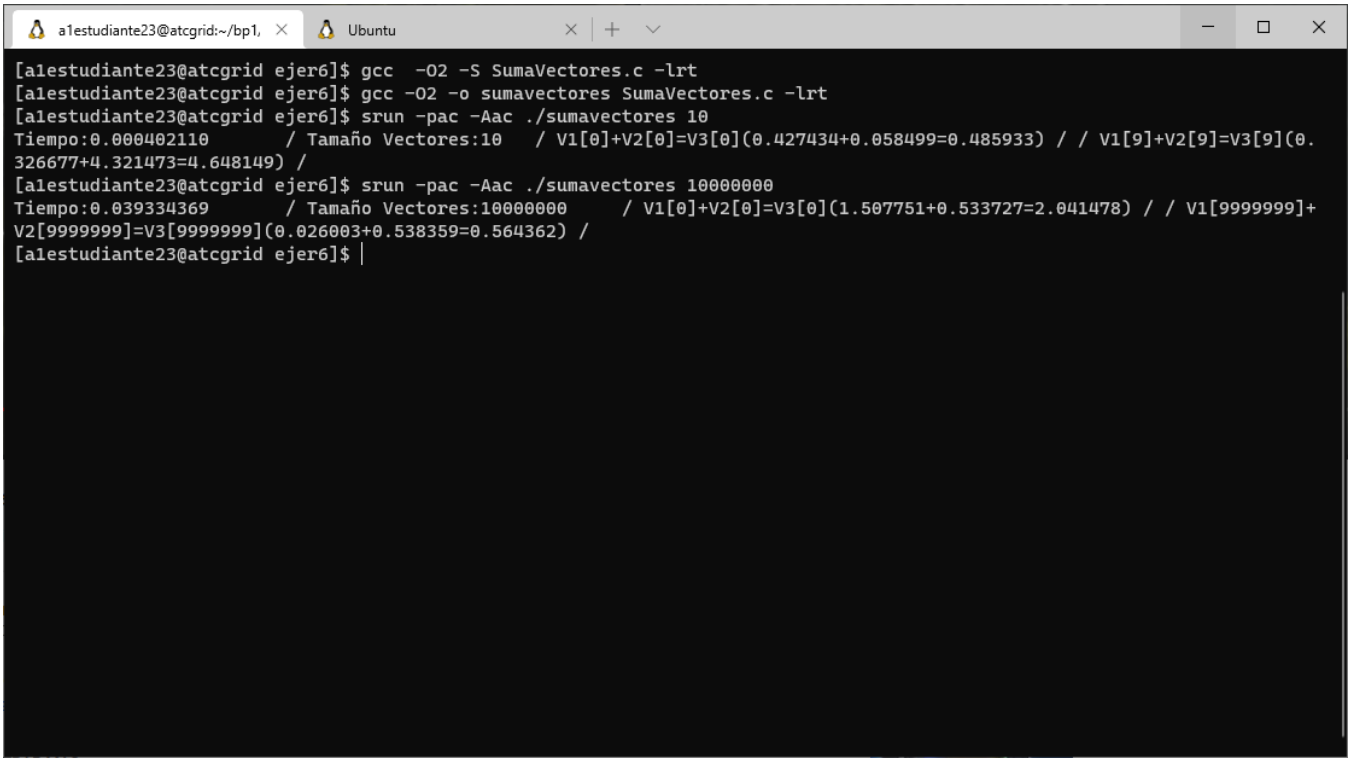
RESPUESTA:

La suma de los tiempos de CPU de usuario y de sistema son siempre menor o igual que el tiempo real (elapsed). Esto es así porque el tiempo de usuario y de sistema no considera variables ajenas a la ejecución del programa como cambios de contexto o suspensiones del proceso. Si, por ejemplo, se producen cambios

de contexto, no se tendrán en cuenta en los tiempos indicados pero sí en el tiempo real. Por tanto, en caso de que se ejecute perfectamente sin ningún tipo de proceso externo o cualquier otra variable, la suma será igual al tiempo real. En cualquier otro caso, será menor.

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):



Se puede observar, compilación con -S para generar el código ensamblador. Compilación para obtener binario y ejecución con 10 y con 10000000 elementos (el propio programa nos indica el tiempo).

RESPUESTA: cálculo de los MIPS y los MFLOPS

Como se puede observar en la captura del código ensamblador, entre las dos llamadas a `clock_gettime()` hay 9 instrucciones.6 de ellas pertenecen al bucle y las otras 3 son independientes de este. Por lo tanto podemos calcular el número de instrucciones con 6*n +3. Siendo n el tamaño del vector.

Por tanto, MIPS con 10 y 10000000 elementos

MIPS_10 = (6*10 + 3) / (0.00040211 * 10^6) = 0.15667 MIPS

MIPS_10000000 = (6*10000000 + 3) / (0.039334369 * 10^6) = 1525.38364 MIPS

Para calcular los MFLOPS, nos fijaremos en las instrucciones que usen los registros de coma flotante. Esto es, los registros `xmm`. Como vemos, hay 3 instrucciones que usan estos registros, que además están dentro del bucle. Por tanto:

MFLOPS_10 = 10*3 / (0.00040211 * 10^6) = 0.07460 MFLOPS

MFLOPS_10000000 = 10000000 * 3 / (0.039334369 * 10^6) = 762.69178 MFLOPS

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

```
    call    clock_gettime
    xorl    %eax, %eax
    .p2align 4,,10
    .p2align 3
.L8:
    movsd   v1(,%rax,8), %xmm0
    addsd   v2(,%rax,8), %xmm0
    movsd   %xmm0, v3(,%rax,8)
    addq    $1, %rax
    cmpl    %eax, %ebp
    ja  .L8
    leaq    16(%rsp), %rsi
    xorl    %edi, %edi
    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).

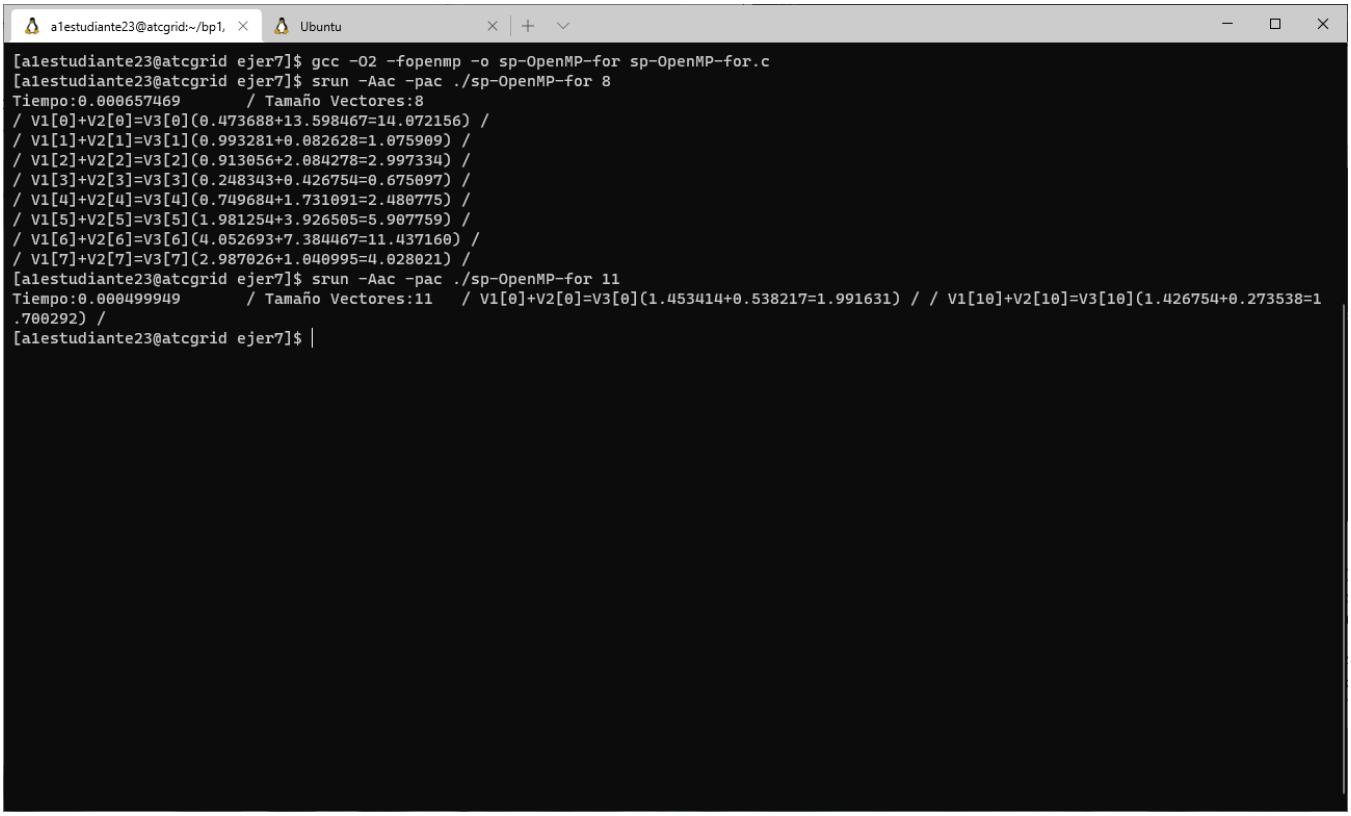
RESPUESTA: Captura que muestre el código fuente implementado sp-OpenMP-for.c

Captura de lo modificado en el código:

```
//Inicializar vectores
srand(time(0));
#pragma omp parallel for
for (int i = 0; i < N; i++)
{
    v1[i] = rand() / ((double)rand());
    v2[i] = rand() / ((double)rand()); //printf("%d:%f,%f/",i,v1[i],v2[i]);
}

ini = omp_get_wtime();

//Calcular suma de vectores
#pragma omp parallel for
for (int i = 0; i < N; i++)
    v3[i] = v1[i] + v2[i];
fin = omp_get_wtime();

ncgt = fin - ini;

//Imprimir resultado de la suma y el tiempo de ejecución
if (N < 10)
{
    printf("Tiempo:%11.9f\t / Tamaño Vectores:%u\n", ncgt, N);
    for (int i = 0; i < N; i++)
        printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
            i, i, i, v1[i], v2[i], v3[i]);
}
else
    printf("Tiempo:%11.9f\t / Tamaño Vectores:%u\t/ V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) / / V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
        ncgt, N, v1[0], v2[0], v3[0], N - 1, N - 1, N - 1, v1[N - 1], v2[N - 1], v3[N - 1]);
```

(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); (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

```
    //Inicializar vectores
    srand(time(0));

    /*
        Debido a que trabajamos con sections, tenemos que fijar el número de estas
        en el código. Por ejemplo, ponemos 4 sections tanto para inicializar el vector
        como para sumarlos.
    */
#pragma omp parallel sections private(i)
    {
#pragma omp section
        for (i = 0; i < N; i += 4)
        {
            v1[i] = rand() / ((double)rand());
            v2[i] = rand() / ((double)rand()); //printf("%d:%f,%f/",i,v1[i],v2[i]);
        }

#pragma omp section
        for (i = 1; i < N; i += 4)
        {
            v1[i] = rand() / ((double)rand());
            v2[i] = rand() / ((double)rand()); //printf("%d:%f,%f/",i,v1[i],v2[i]);
        }

#pragma omp section
        for (i = 2; i < N; i += 4)
        {
            v1[i] = rand() / ((double)rand());
            v2[i] = rand() / ((double)rand()); //printf("%d:%f,%f/",i,v1[i],v2[i]);
        }

#pragma omp section
        for (i = 3; i < N; i += 4)
        {
            v1[i] = rand() / ((double)rand());
            v2[i] = rand() / ((double)rand()); //printf("%d:%f,%f/",i,v1[i],v2[i]);
        }
    }
```

Código de la zona de inicialización

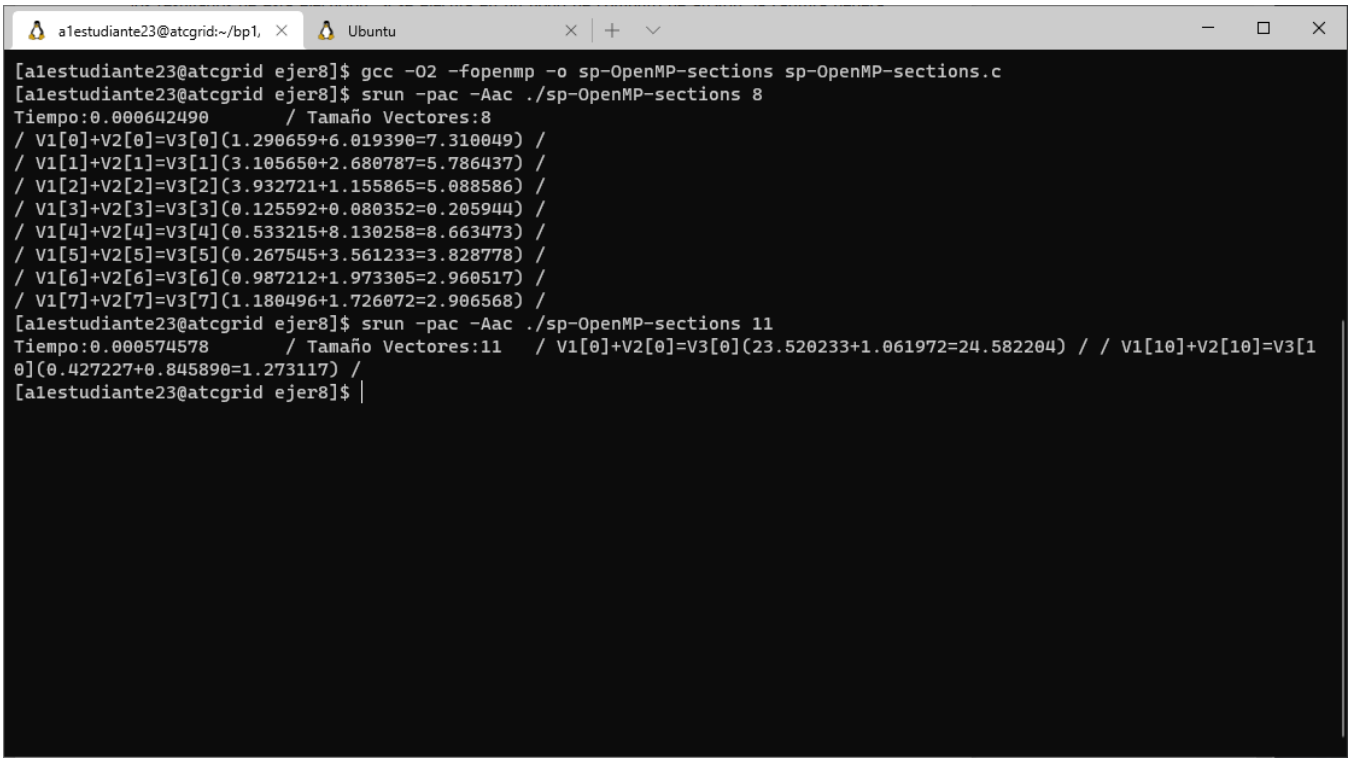
```
    ini = omp_get_wtime();

    //Calcular suma de vectores
#pragma omp parallel sections private(i)
{
#pragma omp section
    {
        for (i = 0; i < N; i+=4)
            v3[i] = v1[i] + v2[i];
    }

#pragma omp section
    {
        for (i = 1; i < N; i+=4)
            v3[i] = v1[i] + v2[i];
    }

#pragma omp section
    {
        for (i = 2; i < N; i+=4)
            v3[i] = v1[i] + v2[i];
    }

#pragma omp section
    {
        for (i = 3; i < N; i+=4)
            v3[i] = v1[i] + v2[i];
    }
}
        fin = omp_get_wtime();

        ncgt = fin - ini;
```

Código de la zona de la suma

El resto del código no se ha modificado.

(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:

   Puesto que en el ejercicio 7 hemos hecho uso de las directivas parallel for, no hay límite de threads para usar el programa puesto que se va repartiendo la tarea entre los distintos hilos. El único problema sería en el caso de que hubiera más threads que elementos en el vector, en cuyo caso, el rendimiento no aumentaría al tener hebras ociosas.

   En el caso del ejercicio 8, al igual que está comentado en el propio código, el número máximo de threads que se va a usar es 4. Esto se debe a haber dividido el código en 4 secciones paralelas. Estas secciones siempre son 4 y no aumentan con el número de threads o de cores disponibles.

10. Rellenar una tabla como la Tabla 217 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

Script binario ejercicio 7

```
#!/bin/bash
#Órdenes para el Gestor de carga de un trabajo:
#1. Asigna al trabajo un nombre
#SBATCH --job-name=SumaLocal
#2. Asignar el trabajo a una partición (cola)
#SBATCH --partition=ac
#3. Asignar el trabajo a un account
#SBATCH --account=ac

#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 trbajo (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 BINARIO EJER7
for ((P=16384;P<=67108864;P=P*2))
do
    srun ./sp-OpenMP-for $P
done
```

Script binario ejercicio 8

```
#!/bin/bash
#Órdenes para el Gestor de carga de un trabajo:
#1. Asigna al trabajo un nombre
#SBATCH --job-name=SumaLocal
#2. Asignar el trabajo a una partición (cola)
#SBATCH --partition=ac
#3. Asignar el trabajo a un account
#SBATCH --account=ac

#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 trbajo (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 BINARIO EJER8
for ((P=16384;P<=67108864;P=P*2))
do
    srun ./sp-OpenMP-sections $P
done
```

Script binario secuencial:

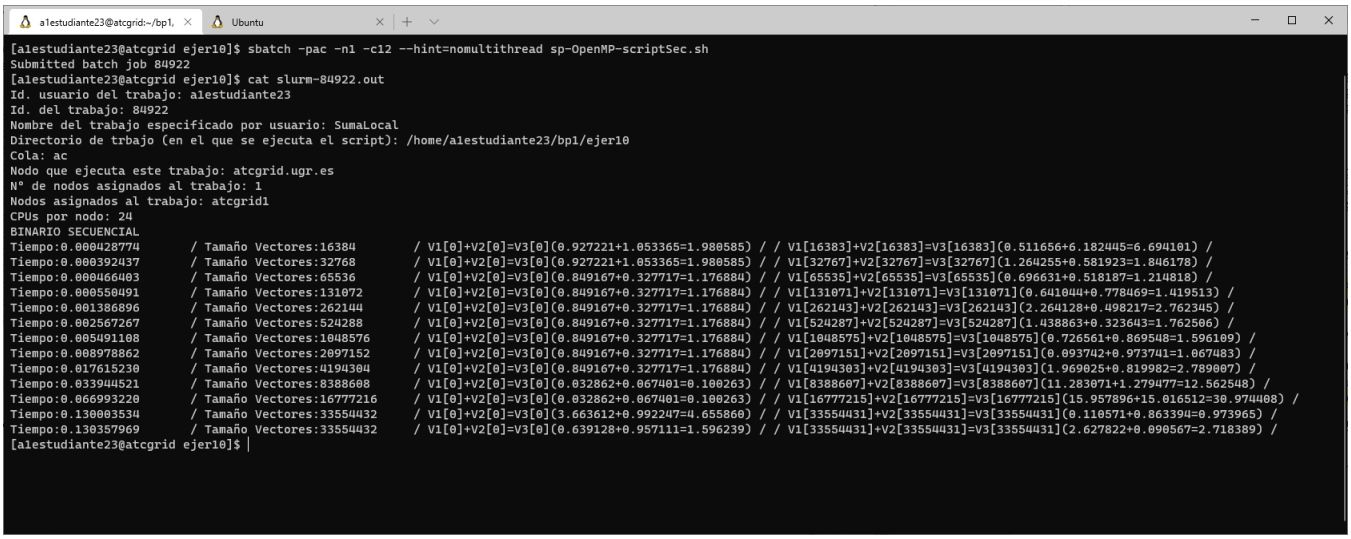
```
#!/bin/bash
#Órdenes para el Gestor de carga de un trabajo:
#1. Asigna al trabajo un nombre
#SBATCH --job-name=SumaLocal
#2. Asignar el trabajo a una partición (cola)
#SBATCH --partition=ac
#3. Asignar el trabajo a un account
#SBATCH --account=ac

#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 trbajo (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 BINARIO SECUENCIAL
for ((P=16384;P<=67108864;P=P*2))
do
    srun ./sumavectores $P
done
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

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

ATCGRID:

Ejecución secuencial:



Ejecución binario ejercicio 7:

```
[a1estudiante23@atcgrid ejer10]$ sbatch -pac -n1 -c12 --hint=nomultithread sp-OpenMP-script7.sh
Submitted batch job 84927
[a1estudiante23@atcgrid ejer10]$ cat slurm-84927.out
Id. usuario del trabajo: a1estudiante23
Id. del trabajo: 84927
Nombre del trabajo especificado por usuario: SumaLocal
Directorio de trbajo (en el que se ejecuta el script): /home/a1estudiante23/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
BINARIO EJER7
Tiempo:0.001178082       / Tamaño Vectores:16384        / V1[0]+V2[0]=V3[0](1.022908+0.530634=1.553541) / / V1[16383]+V2[16383]=V3[16383](1.916164+1.349435=3.265599) /
Tiempo:0.001191583       / Tamaño Vectores:32768        / V1[0]+V2[0]=V3[0](0.655898+0.583138=1.239035) / / V1[32767]+V2[32767]=V3[32767](0.694464+4.350748=5.045211) /
Tiempo:0.004652940       / Tamaño Vectores:65536        / V1[0]+V2[0]=V3[0](0.486768+1.031854=1.518622) / / V1[65535]+V2[65535]=V3[65535](8.491502+7.291950=15.783452) /
Tiempo:0.003950506       / Tamaño Vectores:131072       / V1[0]+V2[0]=V3[0](1.753284+0.970588=2.723872) / / V1[131071]+V2[131071]=V3[131071](2.934955+1.235654=4.170610) /
Tiempo:0.005074792       / Tamaño Vectores:262144       / V1[0]+V2[0]=V3[0](1.897849+1.678788=3.576638) / / V1[262143]+V2[262143]=V3[262143](1.113325+0.130050=1.243375) /
Tiempo:0.005820032       / Tamaño Vectores:524288       / V1[0]+V2[0]=V3[0](1.897849+1.678788=3.576638) / / V1[524287]+V2[524287]=V3[524287](1.566049+0.791325=2.357374) /
Tiempo:0.006058585       / Tamaño Vectores:1048576      / V1[0]+V2[0]=V3[0](0.501558+0.737055=1.238613) / / V1[1048575]+V2[1048575]=V3[1048575](0.072221+0.277029=0.349250) /
Tiempo:0.005826052       / Tamaño Vectores:2097152      / V1[0]+V2[0]=V3[0](0.040923+8.426289=8.467212) / / V1[2097151]+V2[2097151]=V3[2097151](1.537242+2.723540=4.260781) /
Tiempo:0.009822287       / Tamaño Vectores:4194304      / V1[0]+V2[0]=V3[0](1.286459+4.065572=5.352030) / / V1[4194303]+V2[4194303]=V3[4194303](0.755653+3.321209=4.076862) /
Tiempo:0.016397648       / Tamaño Vectores:8388608      / V1[0]+V2[0]=V3[0](2.908826+0.826526=3.735352) / / V1[8388607]+V2[8388607]=V3[8388607](1.994805+0.726117=2.720922) /
Tiempo:0.033309288       / Tamaño Vectores:16777216     / V1[0]+V2[0]=V3[0](0.166729+10.702736=10.869465) / / V1[16777215]+V2[16777215]=V3[16777215](0.312211+3.153437=3.465648) /
Tiempo:0.072183140       / Tamaño Vectores:33554432     / V1[0]+V2[0]=V3[0](1.844633+0.180790=2.025422) / / V1[33554431]+V2[33554431]=V3[33554431](0.585667+0.893341=1.479009) /
srun: Job step aborted: Waiting up to 32 seconds for job step to finish.
slurmstepd: error: *** JOB 84927 ON atcgrid1 CANCELLED AT 2021-04-10T01:30:59 DUE TO TIME LIMIT ***
slurmstepd: error: *** STEP 84927.12 ON atcgrid1 CANCELLED AT 2021-04-10T01:30:59 DUE TO TIME LIMIT ***
srun: error: atcgrid1: task 0: Terminated
[a1estudiante23@atcgrid ejer10]$
```

Ejecución binario ejercicio 8:

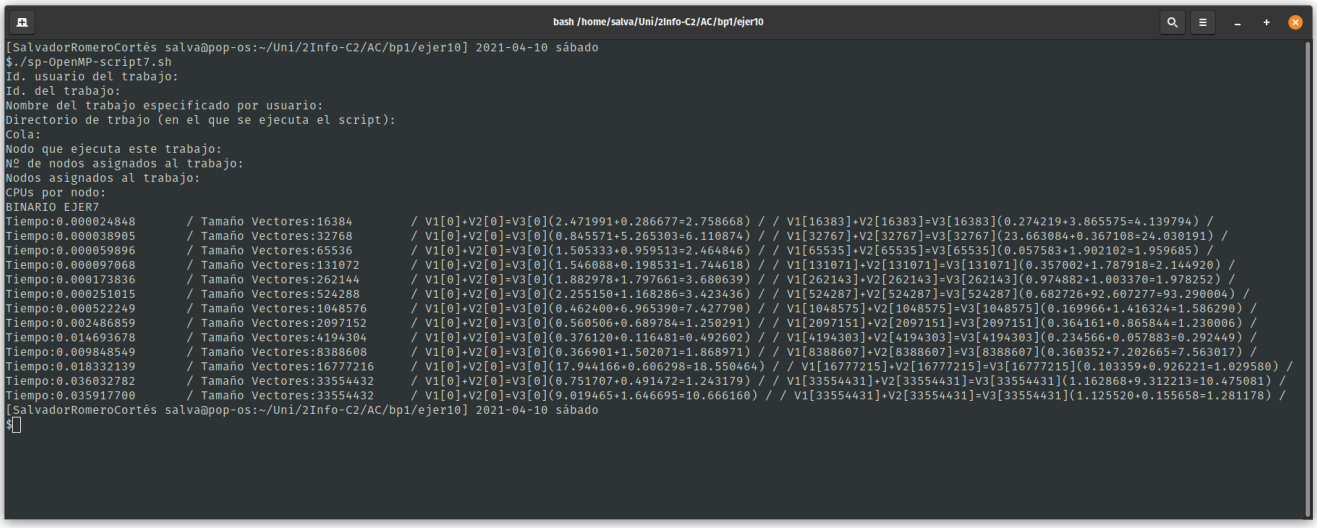
```
[a1estudiante23@atcgrid ejer10]$ sbatch -pac -n1 -c12 --hint=nomultithread sp-OpenMP-script8.sh
Submitted batch job 84928
[a1estudiante23@atcgrid ejer10]$ cat slurm-84928.out
Id. usuario del trabajo: a1estudiante23
Id. del trabajo: 84928
Nombre del trabajo especificado por usuario: SumaLocal
Directorio de trbajo (en el que se ejecuta el script): /home/a1estudiante23/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
BINARIO EJER8
Tiempo:0.000519849       / Tamaño Vectores:16384        / V1[0]+V2[0]=V3[0](5.901318+0.135096=6.036414) / / V1[16383]+V2[16383]=V3[16383](0.210194+2.748627=2.958821) /
Tiempo:0.000746462       / Tamaño Vectores:32768        / V1[0]+V2[0]=V3[0](5.901318+0.140130=6.041448) / / V1[32767]+V2[32767]=V3[32767](0.800362+1.141361=1.941723) /
Tiempo:0.000745662       / Tamaño Vectores:65536        / V1[0]+V2[0]=V3[0](2.308932+1.393533=3.702465) / / V1[65535]+V2[65535]=V3[65535](1.819527+0.401056=2.220583) /
Tiempo:0.002642091       / Tamaño Vectores:131072       / V1[0]+V2[0]=V3[0](2.308932+1.896313=4.205246) / / V1[131071]+V2[131071]=V3[131071](2.530819+32.749825=35.280644) /
Tiempo:0.002401795       / Tamaño Vectores:262144       / V1[0]+V2[0]=V3[0](2.308932+0.865240=3.174172) / / V1[262143]+V2[262143]=V3[262143](0.505431+0.493084=0.998516) /
Tiempo:0.004123513       / Tamaño Vectores:524288       / V1[0]+V2[0]=V3[0](1.295830+0.218123=1.513954) / / V1[524287]+V2[524287]=V3[524287](0.605185+2.061221=2.666406) /
Tiempo:0.007251058       / Tamaño Vectores:1048576      / V1[0]+V2[0]=V3[0](1.075900+0.791015=1.866915) / / V1[1048575]+V2[1048575]=V3[1048575](2.705492+0.234008=2.939500) /
Tiempo:0.013017103       / Tamaño Vectores:2097152      / V1[0]+V2[0]=V3[0](2.731664+1.466788=4.198452) / / V1[2097151]+V2[2097151]=V3[2097151](0.859237+8.194857=9.054094) /
Tiempo:0.025916759       / Tamaño Vectores:4194304      / V1[0]+V2[0]=V3[0](0.851799+0.778917=1.630716) / / V1[4194303]+V2[4194303]=V3[4194303](13.559051+3.718548=17.277599) /
Tiempo:0.045680687       / Tamaño Vectores:8388608      / V1[0]+V2[0]=V3[0](1.552700+0.563069=2.115769) / / V1[8388607]+V2[8388607]=V3[8388607](0.644117+1.919304=2.563421) /
Tiempo:0.105596114       / Tamaño Vectores:16777216     / V1[0]+V2[0]=V3[0](0.164341+0.042147=0.206487) / / V1[16777215]+V2[16777215]=V3[16777215](1.161729+0.041654=1.203384) /
Tiempo:0.180050734       / Tamaño Vectores:33554432     / V1[0]+V2[0]=V3[0](1.141822+0.209920=1.351741) / / V1[33554431]+V2[33554431]=V3[33554431](1.209058+1.044680=2.253738) /
srun: Job step aborted: Waiting up to 32 seconds for job step to finish.
slurmstepd: error: *** STEP 84928.12 ON atcgrid1 CANCELLED AT 2021-04-10T01:34:29 DUE TO TIME LIMIT ***
slurmstepd: error: *** JOB 84928 ON atcgrid1 CANCELLED AT 2021-04-10T01:34:29 DUE TO TIME LIMIT ***
[a1estudiante23@atcgrid ejer10]$
```

PC

Ejecución secuencial:

```
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp1/ejer10] 2021-04-10 sábado
$./sp-OpenMP-scriptSec.sh
Id. usuario del trabajo:
Id. del trabajo:
Nombre del trabajo especificado por usuario:
Directorio de trbajo (en el que se ejecuta el script):
Cola:
Nodo que ejecuta este trabajo:
Nº de nodos asignados al trabajo:
Nodos asignados al trabajo:
CPUs por nodo:
BINARIO SECUENCIAL
Tiempo:0.000047801       / Tamaño Vectores:16384        / V1[0]+V2[0]=V3[0](1.043947+1.304284=2.348231) / / V1[16383]+V2[16383]=V3[16383](0.075852+0.918580=0.994432) /
Tiempo:0.000103014       / Tamaño Vectores:32768        / V1[0]+V2[0]=V3[0](1.043947+1.304284=2.348231) / / V1[32767]+V2[32767]=V3[32767](0.813596+9.110241=9.923837) /
Tiempo:0.000209876       / Tamaño Vectores:65536        / V1[0]+V2[0]=V3[0](1.043947+1.304284=2.348231) / / V1[65535]+V2[65535]=V3[65535](1.022845+0.596696=1.619542) /
Tiempo:0.000227780       / Tamaño Vectores:131072       / V1[0]+V2[0]=V3[0](1.043947+1.304284=2.348231) / / V1[131071]+V2[131071]=V3[131071](1.674009+1.141057=2.815067) /
Tiempo:0.000441794       / Tamaño Vectores:262144       / V1[0]+V2[0]=V3[0](1.043947+1.304284=2.348231) / / V1[262143]+V2[262143]=V3[262143](3.601816+10.181577=13.783393) /
Tiempo:0.001064419       / Tamaño Vectores:524288       / V1[0]+V2[0]=V3[0](1.043947+1.304284=2.348231) / / V1[524287]+V2[524287]=V3[524287](0.080054+0.561809=0.641863) /
Tiempo:0.002828438       / Tamaño Vectores:1048576      / V1[0]+V2[0]=V3[0](1.043947+1.304284=2.348231) / / V1[1048575]+V2[1048575]=V3[1048575](1.154750+0.065359=1.220109) /
Tiempo:0.005681633       / Tamaño Vectores:2097152      / V1[0]+V2[0]=V3[0](1.043947+1.304284=2.348231) / / V1[2097151]+V2[2097151]=V3[2097151](4.606380+15.836596=20.442976) /
Tiempo:0.010261667       / Tamaño Vectores:4194304      / V1[0]+V2[0]=V3[0](1.043947+1.304284=2.348231) / / V1[4194303]+V2[4194303]=V3[4194303](1.756381+2.152324=3.908705) /
Tiempo:0.020470273       / Tamaño Vectores:8388608      / V1[0]+V2[0]=V3[0](1.043947+1.304284=2.348231) / / V1[8388607]+V2[8388607]=V3[8388607](0.789560+5.199150=5.988711) /
Tiempo:0.040117954       / Tamaño Vectores:16777216     / V1[0]+V2[0]=V3[0](0.958424+16.925343=17.883767) / / V1[16777215]+V2[16777215]=V3[16777215](8.757254+1.235273=9.992527) /
Tiempo:0.080487347       / Tamaño Vectores:33554432     / V1[0]+V2[0]=V3[0](0.958424+16.925343=17.883767) / / V1[33554431]+V2[33554431]=V3[33554431](1.177637+0.780291=1.957928) /
Tiempo:0.080448319       / Tamaño Vectores:33554432     / V1[0]+V2[0]=V3[0](1.819044+1.513965=3.333009) / / V1[33554431]+V2[33554431]=V3[33554431](0.205243+0.359369=0.564612) /
[SalvadorRomeroCortés salva@pop-os:~/Uni/2Info-C2/AC/bp1/ejer10] 2021-04-10 sábado
$
```

Ejecución binario ejercicio 7



Ejecución binario ejercicio 8

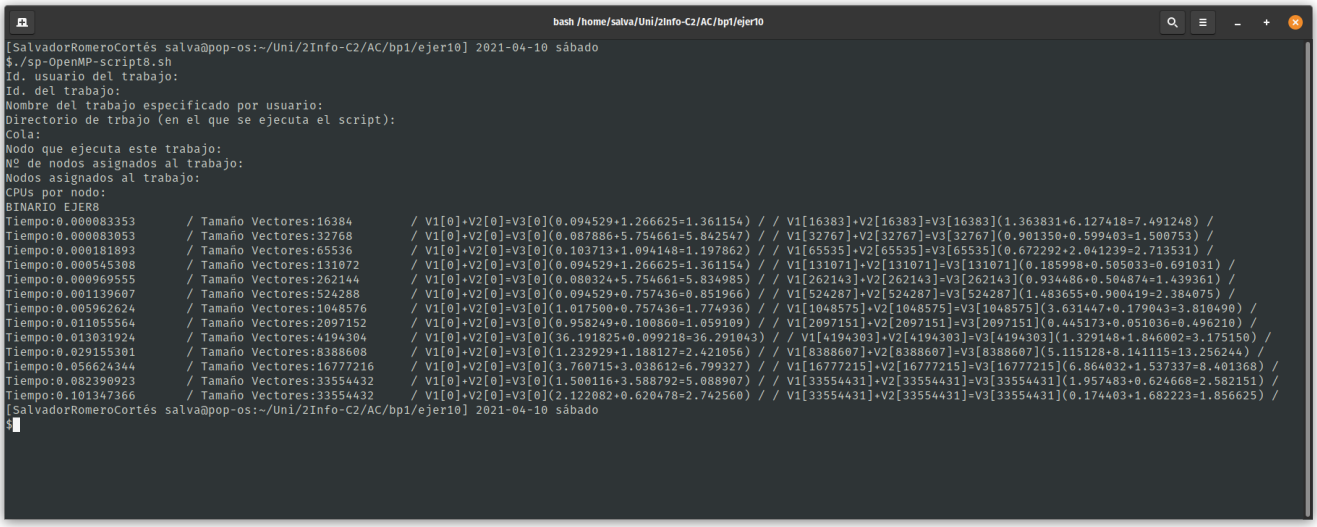


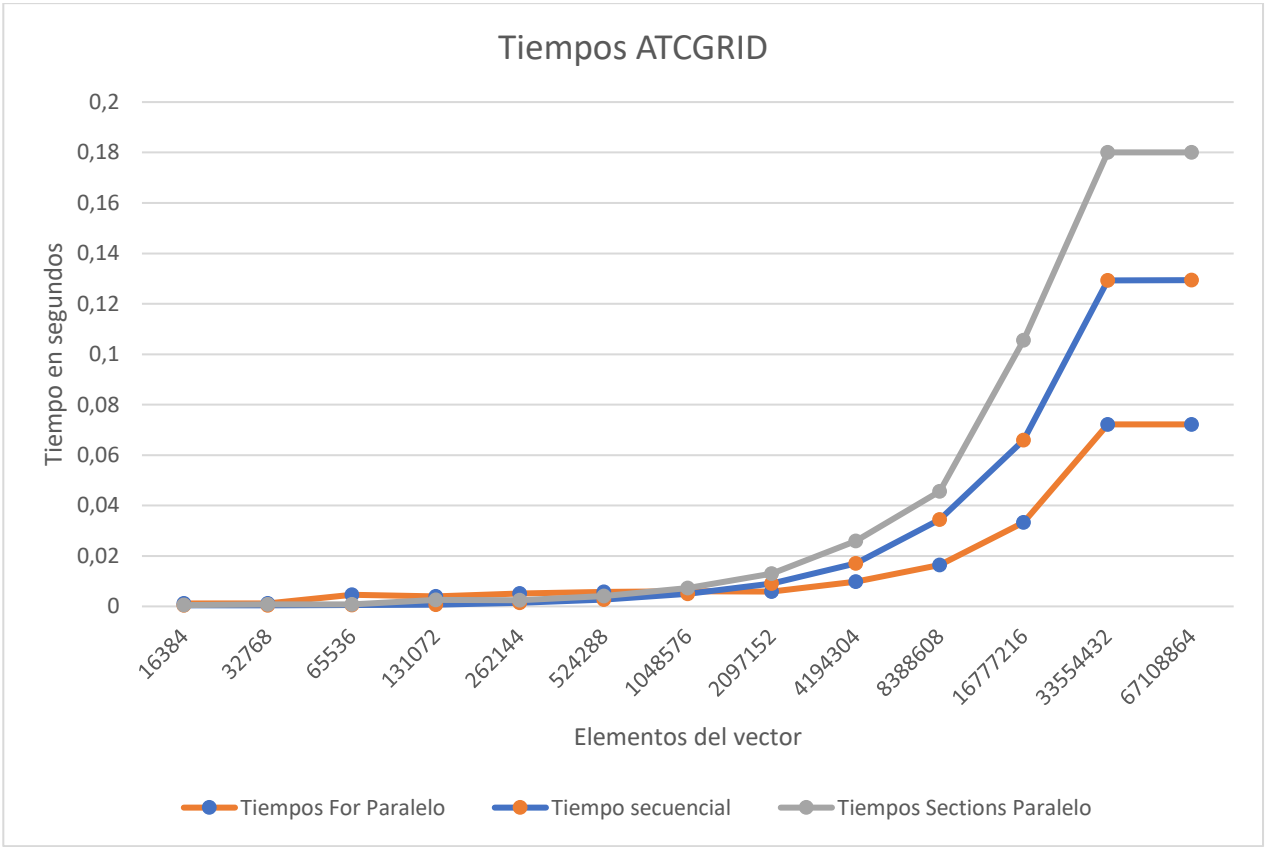
Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados, que debe coincidir con el número de cores físicos y cores lógicos utilizados.

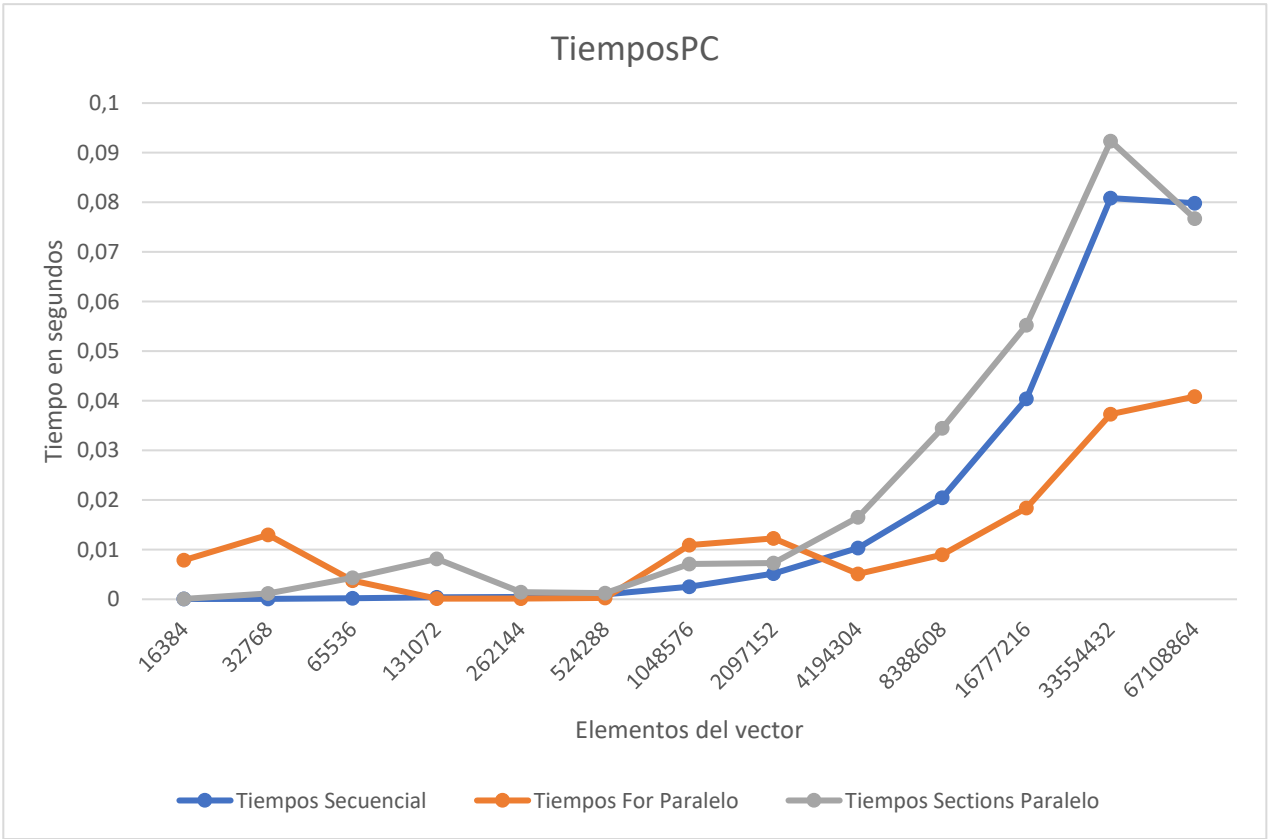
ATCGRID

Nº de Componentes	**T. secuencial vect. Globales** **1 thread=core**	**T. paralelo (versión** for**)** **12 threads = cores lógicos = cores físicos**	**T. paralelo (versión** sections**)** **4 threads = cores lógicos = cores físicos**
16384	0.000442056	0.001178082	0.000519849
32768	0.000462920	0.001191583	0.000746462
65536	0.000564351	0.004652940	0.000745662
131072	0.000676787	0.003950506	0.002642091
262144	0.001381205	0.005074792	0.002401795
524288	0.002672260	0.005820032	0.004123513
1048576	0.004993129	0.006058585	0.007251058
2097152	0.009081609	0.005826052	0.013017103
4194304	0.017112380	0.009822287	0.025916759
8388608	0.034459608	0.016397648	0.045680687
16777216	0.065936852	0.033309288	0.105596114
33554432	0.129233605	0.072183140	0.180050734
67108864	0.129384642	0.072183140	0.180050734

PC

Nº de Componentes	**T. secuencial vect. Globales** **1 thread=core**	**T. paralelo (versión** for**)** **12 threads**	**T. paralelo (versión** sections**)** **4 threads**
16384	0.000054181	0.007879549	0.000077054
32768	0.000090950	0.012999560	0.001207203
65536	0.000213508	0.003738213	0.004316088
131072	0.000379177	0.000131756	0.008173647
262144	0.000439279	0.000173754	0.001405995
524288	0.000960711	0.000277197	0.001244081
1048576	0.002514647	0.010920345	0.007093573
2097152	0.005195233	0.012266949	0.007329155
4194304	0.010363908	0.005131737	0.016510991
8388608	0.020444119	0.009015211	0.034457895
16777216	0.040421427	0.018423528	0.055257150
33554432	0.080844701	0.037298735	0.092366452
67108864	0.079852677	0.040856397	0.076730505


Tiempos ATCGRID
Tiempo en segundos
0,2
0,18
0,16
0,14
0,12
0,1
0,08
0,06
0,04
0,02
0
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
Elementos del vector
Tiempos For Paralelo
Tiempo secuencial
Tiempos Sections Paralelo


TiemposPC
Tiempo en segundos
0,1
0,09
0,08
0,07
0,06
0,05
0,04
0,03
0,02
0,01
0
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
Elementos del vector
Tiempos Secuencial
Tiempos For Paralelo
Tiempos Sections Paralelo

11. Rellenar una tabla como la 21Tabla 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

Script binario ejercicio 7:

```
#!/bin/bash
#Órdenes para el Gestor de carga de un trabajo:
#1. Asigna al trabajo un nombre
#SBATCH --job-name=SumaLocal
#2. Asignar el trabajo a una partición (cola)
#SBATCH --partition=ac
#3. Asignar el trabajo a un account
#SBATCH --account=ac

#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 trbajo (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 BINARIO EJER7
for ((P=8388608;P<=67108864;P=P*2))
do
    srun time ./sp-OpenMP-for $P
done
```

Script binario secuencial:

```
#!/bin/bash
#Órdenes para el Gestor de carga de un trabajo:
#1. Asigna al trabajo un nombre
#SBATCH --job-name=SumaLocal
#2. Asignar el trabajo a una partición (cola)
#SBATCH --partition=ac
#3. Asignar el trabajo a un account
#SBATCH --account=ac

#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 trbajo (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 BINARIO SECUENCIAL
for ((P=8388608;P<=67108864;P=P*2))
do
    srun time ./sumavectores $P
done
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

CAPTURAS DE PANTALLA (ejecución en atcgrid):

Envío y ejecución del binario del ejercicio 7:

```
[a1estudiante23@atcgrid ejer11]$ sbatch -pac -Aac -n1 -c12 --hint=nomultithread ./sp-OpenMP-script7.sh
Submitted batch job 84980
[a1estudiante23@atcgrid ejer11]$ cat slurm-84980.out
Id. usuario del trabajo: a1estudiante23
Id. del trabajo: 84980
Nombre del trabajo especificado por usuario: SumaLocal
Directorio de trbajo (en el que se ejecuta el script): /home/a1estudiante23/bp1/ejer11
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
BINARIO EJER7
Tiempo:0.019505050       / Tamaño Vectores:8388608      / V1[0]+V2[0]=V3[0](1.254480+2.165835=3.420315) / / V1[8388607]+V2[8388607]=V3[8388607](0.331921+0.217840=0.549761) /
8.48user 64.19system 0:06.92elapsed 1048%CPU (0avgtext+0avgdata 203632maxresident)k
32inputs+0outputs (0major+1519minor)pagefaults 0swaps
Tiempo:0.034794204       / Tamaño Vectores:16777216     / V1[0]+V2[0]=V3[0](0.556479+1.620647=2.177125) / / V1[16777215]+V2[16777215]=V3[16777215](0.530747+4.209535=4.740282) /
16.66user 130.53system 0:13.84elapsed 1063%CPU (0avgtext+0avgdata 400236maxresident)k
0inputs+0outputs (0major+1810minor)pagefaults 0swaps
Tiempo:0.065324470       / Tamaño Vectores:33554432     / V1[0]+V2[0]=V3[0](1.355236+0.745205=2.100442) / / V1[33554431]+V2[33554431]=V3[33554431](0.977682+0.658735=1.636417) /
32.77user 256.76system 0:27.25elapsed 1062%CPU (0avgtext+0avgdata 787324maxresident)k
0inputs+0outputs (0major+2377minor)pagefaults 0swaps
Tiempo:0.066412561       / Tamaño Vectores:33554432     / V1[0]+V2[0]=V3[0](5.907043+1.256755=7.163798) / / V1[33554431]+V2[33554431]=V3[33554431](1.683908+0.410936=2.094844) /
31.08user 254.95system 0:26.68elapsed 1071%CPU (0avgtext+0avgdata 787320maxresident)k
0inputs+0outputs (0major+2317minor)pagefaults 0swaps
[a1estudiante23@atcgrid ejer11]$
```

Envío y ejecución del binario del programa secuencial:

```
[a1estudiante23@atcgrid ejer11]$ sbatch -pac -Aac -n1 -c12 --hint=nomultithread ./sp-OpenMP-scriptSec.sh
Submitted batch job 84979
[a1estudiante23@atcgrid ejer11]$ cat slurm-84979.out
Id. usuario del trabajo: a1estudiante23
Id. del trabajo: 84979
Nombre del trabajo especificado por usuario: SumaLocal
Directorio de trbajo (en el que se ejecuta el script): /home/a1estudiante23/bp1/ejer11
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
BINARIO SECUENCIAL
Tiempo:0.033041427       / Tamaño Vectores:8388608      / V1[0]+V2[0]=V3[0](0.213685+1.478480=1.692164) / / V1[8388607]+V2[8388607]=V3[8388607](1.359824+1.150449=2.510274) /
0.43user 0.04system 0:00.48elapsed 98%CPU (0avgtext+0avgdata 203388maxresident)k
0inputs+0outputs (0major+817minor)pagefaults 0swaps
Tiempo:0.067032713       / Tamaño Vectores:16777216     / V1[0]+V2[0]=V3[0](0.213685+1.478480=1.692164) / / V1[16777215]+V2[16777215]=V3[16777215](2.520088+0.722235=3.242323) /
0.84user 0.06system 0:00.91elapsed 99%CPU (0avgtext+0avgdata 400000maxresident)k
0inputs+0outputs (0major+915minor)pagefaults 0swaps
Tiempo:0.129529282       / Tamaño Vectores:33554432     / V1[0]+V2[0]=V3[0](0.046162+0.498450=0.544612) / / V1[33554431]+V2[33554431]=V3[33554431](7.570858+1.908610=9.479468) /
1.66user 0.14system 0:01.81elapsed 99%CPU (0avgtext+0avgdata 787080maxresident)k
0inputs+0outputs (0major+1105minor)pagefaults 0swaps
Tiempo:0.129679516       / Tamaño Vectores:33554432     / V1[0]+V2[0]=V3[0](17.817626+1.364780=19.182406) / / V1[33554431]+V2[33554431]=V3[33554431](0.208888+2.046961=2.255848) /
1.65user 0.16system 0:01.81elapsed 99%CPU (0avgtext+0avgdata 787080maxresident)k
0inputs+0outputs (0major+1106minor)pagefaults 0swaps
[a1estudiante23@atcgrid ejer11]$
```

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**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
8388608	0.48	0.43	0.04	6.92	8.48	64.19
16777216	0.91	0.84	0.06	13.84	16.6	130.53
33554432	1.81	1.66	0.14	27.25	32.77	256.76
67108864	1.81	0.16	1.65	26.68	31.08	254.95

Como podemos observar, las mediciones de tiempo usando “time” para los programas paralelos producen resultados extraños. Por ejemplo, que la suma de CPU user y CPU system sea superior a la elapsed. O que aparezcan unos números tan altos cuando el tiempo de ejecución (medido por la orden `omp_get_wtime()`) son bastante mejores.

Podemos concluir que hay que tener cuidado a la hora de realizar este tipo de mediciones ya que funciones como “time” suman el total de tiempo de cada hebra en lugar del tiempo que ha tardado en total.