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):Jose Teodosio Lorente Vallecillos
Grupo de prácticas y profesor de prácticas: A3 Mancia Anguita
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 nº iteraciones \n");
        exit(-1);
    }

    n = atoi(argv[1]);

    #pragma omp parallel
    {
        #pragma omp 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
    {
        #pragma omp 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("Depués de la región parallel:\n");
            for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
            printf("\n");
            printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
        }
    }
}
```

CAPTURAS DE PANTALLA:

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp1/ejer2] 2021-03-19 viernes
$ls
single.c  singleModificado.c
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp1/ejer2] 2021-03-19 viernes
$gcc -fopenmp -O2 singleModificado.c  -o singleModificado && ./singleModificado
Introduce valor de inicialización a: 22
Single ejecutada por el thread 0
Depués de la región parallel:
b[0] = 22       b[1] = 22       b[2] = 22       b[3] = 22       b[4] = 22       b[5] = 22       b[6] = 22
b[7] = 22       b[8] = 22
Single ejecutada por el thread 1
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp1/ejer2] 2021-03-19 viernes
$
```

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("Depués de la región parallel:\n");
            for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
            printf("\n");
            printf("Master ejecutada por el thread %d\n", omp_get_thread_num());
        }
    }
}
```

CAPTURAS DE PANTALLA:

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp1/ejer3] 2021-03-19 viernes
$ls
singleModificado2.c
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp1/ejer3] 2021-03-19 viernes
$gcc -fopenmp -O2 singleModificado2.c  -o singleModificado2 && ./singleModificado2
Introduce valor de inicialización a: 22
Single ejecutada por el thread 3
Depués de la región parallel:
b[0] = 22       b[1] = 22       b[2] = 22       b[3] = 22       b[4] = 22       b[5] = 22       b[6] = 22
b[7] = 22       b[8] = 22
Master ejecutada por el thread 0
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp1/ejer3] 2021-03-19 viernes
$
```

RESPUESTA A LA PREGUNTA:

La diferencia es que la directiva master la ejecuta la hebra 0, sin embargo la directiva single la ejecuta cualquier hebra.

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 la directiva barrier hace que las hebras se queden bloqueadas hasta que todas hayan acabado. En el caso de master.c si las hebras no es esperan a que terminen las demás sale un resultado erróneo de la suma

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.

```
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer5] 2021-04-01 jueves
$sbatch -p ac --wrap "time ./SumaVectoresCglobales 10000000"
Submitted batch job 77799
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer5] 2021-04-01 jueves
$ls
slurm-77799.out  SumaVectoresCglobales
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer5] 2021-04-01 jueves
$cat slurm-77799.out
Tiempo(seg.):0.060425613         / Tamaño Vectores:10000000
/ V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) /
/ V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /

real    0m0.224s
user    0m0.171s
sys     0m0.043s
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer5] 2021-04-01 jueves
$
```

CAPTURAS DE PANTALLA:

RESPUESTA: El tiempo de CPU del usuario + el tiempo de CPU del sistema = 0.214s, mientras que el tiempo de CPU real es 0.224s. El tiempo real es mayor debido a que un programa puede estar esperando sin ejecutar nada, de modo que el tiempo real cuenta el tiempo total. El tiempo de usuario y el del sistema solo cuentan el tiempo de ejecución.

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.

```
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer6] 2021-04-01 jueves
$sbatch -p ac --wrap "./SumaVectoresCglobales 10"
Submitted batch job 77929
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer6] 2021-04-01 jueves
$cat slurm-77929.out
Tiempo(seg.):0.000387276         / Tamaño Vectores:10
/ V1[0]+V2[0]=V3[0](1.000000+1.000000=2.000000) /
/ V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer6] 2021-04-01 jueves
$sbatch -p ac --wrap "./SumaVectoresCglobales 10000000"
Submitted batch job 77930
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer6] 2021-04-01 jueves
$cat slurm-77930.out
Tiempo(seg.):0.060056943         / Tamaño Vectores:10000000
/ V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) /
/ V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer6] 2021-04-01 jueves
$
```

CAPTURAS DE PANTALLA (que muestren la generación del código ensamblador y del código ejecutable, y la obtención de los tiempos de ejecución):

RESPUESTA: cálculo de los MIPS y los MFLOPS

Tamaño 10

Tiempo: 0.000387276s

NI: 6*10+3= 63

MIPS: $63/(0.000387276*10^6)= 0.162674682$

FPO: 3*10= 30

MFLOPS: $30/(0.000387276*10^6)= 0.077464134$

Tamaño 10000000

Tiempo: 0.060056943s

NI: 6*10000000+3= 60000003

MIPS: $60000003/(0.060056943*10^6)= 999.051899795$

FPO: 3*10000000= 30000000

MFLOPS: $30000000/(0.060056943*10^6)= 499.525924921$

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

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`

```c
  //Inicializar vectores
  #pragma omp parallel for
  for (i = 0; i < N; i++) {
      v1[i] = N * 0.1 + i * 0.1;
      v2[i] = N * 0.1 - i * 0.1;
  }

  double t1 = omp_get_wtime();

//  clock_gettime(CLOCK_REALTIME,&cgt1);

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

  double t2 = omp_get_wtime();

  double tt = t2 - t1;

/*  clock_gettime(CLOCK_REALTIME,&cgt2);
  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
       (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));*/

  //Imprimir resultado de la suma y el tiempo de ejecución
  if (N<10) {
    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%lu\n",tt,N);
    #pragma omp parallel for
    for(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(seg.):%11.9f\t / Tamaño Vectores:%u\t\n/ V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) / \n/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
           tt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
  }

  #ifdef VECTOR_DYNAMIC
  free(v1); // libera el espacio reservado para v1
  free(v2); // libera el espacio reservado para v2
  free(v3); // libera el espacio reservado para v3
  #endif
  return 0;
}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp1/ejer7] 2021-04-01 jueves
$gcc -fopenmp -O2 -o sp-OpenMP-for sp-OpenMP-for.c
sp-OpenMP-for.c: In function ‘main’:
sp-OpenMP-for.c:89:56: warning: format ‘%lu’ expects argument of type ‘long unsigned int’, but argument 3
 has type ‘unsigned int’ [-Wformat=]
   89 |     printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%lu\n",tt,N);
      |                                                      ~~^       ~
      |                                                        |       |
      |                                                        |       unsigned int
      |                                                        long unsigned int
      |                                                      %u
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp1/ejer7] 2021-04-01 jueves
$ls
sp-OpenMP-for  sp-OpenMP-for.c
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp1/ejer7] 2021-04-01 jueves
$./sp-OpenMP-for 8
Tamaño Vectores:8 (4 B)
Tiempo(seg.):0.000006110         / Tamaño Vectores:8
/ V1[6]+V2[6]=V3[6](1.400000+0.200000=1.600000) /
/ V1[7]+V2[7]=V3[7](1.500000+0.100000=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[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) /
/ V1[1]+V2[1]=V3[1](0.900000+0.700000=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) /
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp1/ejer7] 2021-04-01 jueves
$./sp-OpenMP-for 11
Tamaño Vectores:11 (4 B)
Tiempo(seg.):0.000007413         / 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) /
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp1/ejer7] 2021-04-01 jueves
$
```

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`

```c
//Inicializar vectores
#pragma omp parallel sections
{
    #pragma omp section
    for (i = 0; i < N/4; i++) {
        v1[i] = N * 0.1 + i * 0.1;
        v2[i] = N * 0.1 - i * 0.1;
    }
    #pragma omp section
    for (i = N/4; i < N/2; i++) {
        v1[i] = N * 0.1 + i * 0.1;
        v2[i] = N * 0.1 - i * 0.1;
    }
    #pragma omp section
    for (i = N/2; i < N*3/4; i++) {
        v1[i] = N * 0.1 + i * 0.1;
        v2[i] = N * 0.1 - i * 0.1;
    }
    #pragma omp section
    for (i = N*3/4; i < N; i++) {
        v1[i] = N * 0.1 + i * 0.1;
        v2[i] = N * 0.1 - i * 0.1;
    }

}

double t1 = omp_get_wtime();

//  clock_gettime(CLOCK_REALTIME,&cgt1);
#pragma omp parallel sections
{
    #pragma omp section
    for (i = 0; i < N/4; i++)
        v3[i] = v1[i] + v2[i];

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

    #pragma omp section
    for (i = N/2; i < N*3/4; i++)
        v3[i] = v1[i] + v2[i];

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

double t2 = omp_get_wtime();

double tt = t2 - t1;
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp1/ejer8] 2021-04-01 jueves
$gcc -fopenmp -O2 -o sp-OpenMP-sections sp-OpenMP-sections.c
sp-OpenMP-sections.c: In function ‘main’:
sp-OpenMP-sections.c:121:56: warning: format ‘%lu’ expects argument of type ‘long unsigned int’, but argu
ment 3 has type ‘unsigned int’ [-Wformat=]
  121 |     printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%lu\n",tt,N);
      |                                                      ~~^       ~
      |                                                        |       |
      |                                                        |       unsigned int
      |                                                        long unsigned int
      |                                                      %u
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp1/ejer8] 2021-04-01 jueves
$ls
sp-OpenMP-sections  sp-OpenMP-sections.c
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp1/ejer8] 2021-04-01 jueves
$./sp-OpenMP-sections 8
Tamaño Vectores:8 (4 B)
Tiempo(seg.):0.000005309         / Tamaño Vectores:8
/ 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[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) /
/ V1[1]+V2[1]=V3[1](0.900000+0.700000=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) /
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp1/ejer8] 2021-04-01 jueves
$./sp-OpenMP-sections 11
Tamaño Vectores:11 (4 B)
Tiempo(seg.):0.000006789         / 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) /
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp1/ejer8] 2021-04-01 jueves
$
```

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: Todas las hebras y cores que tenga el PC, esto es debido a no haber definido el omp_num_threads. Esto es tanto en el ejercicio 7 como en el 8, salvo que en el 8 habrá hebras que no ejecuten nada debido a sections.

10. Rellenar una tabla como la Tabla 215 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.**

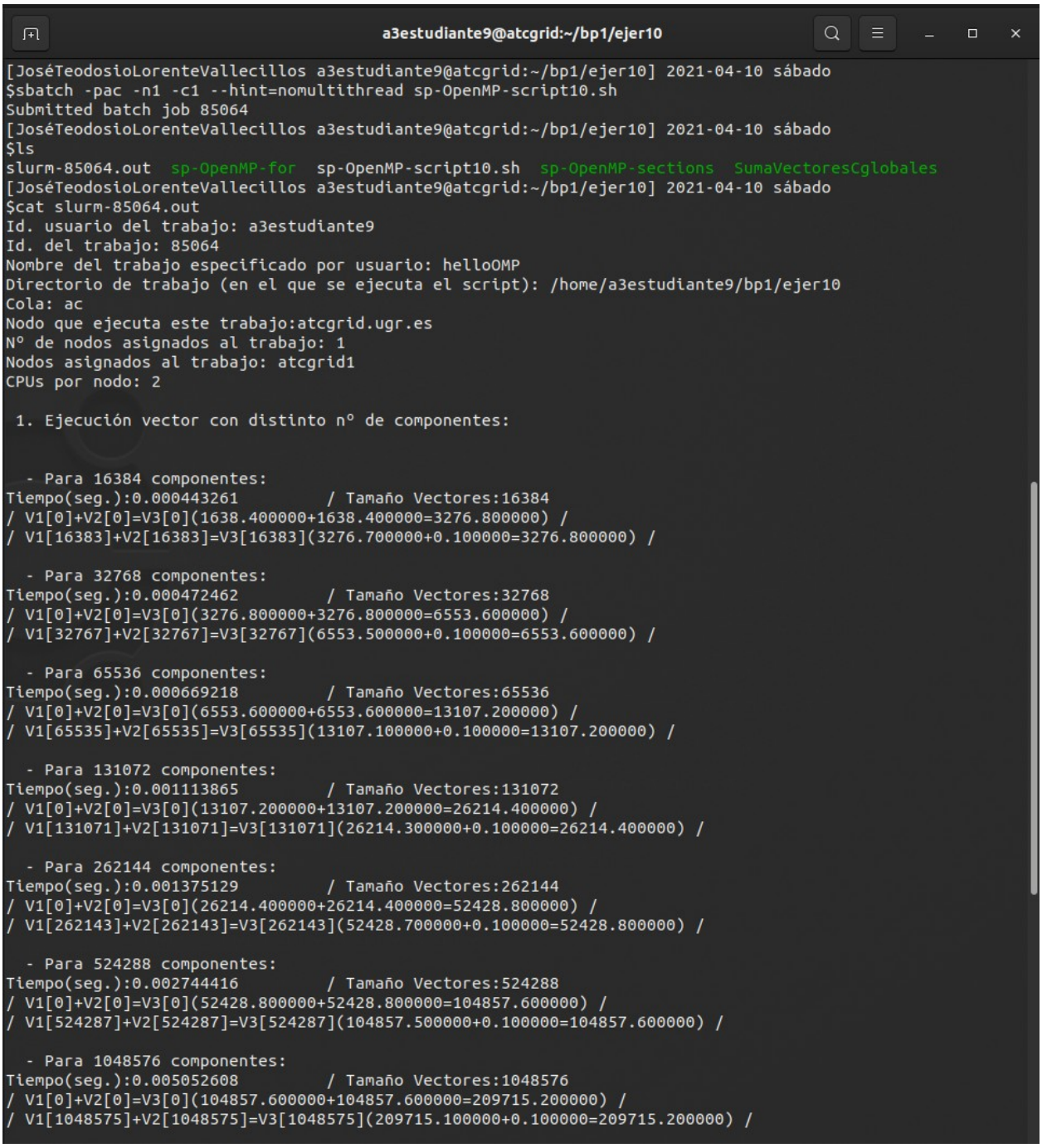
RESPUESTA: Captura del script implementado `sp-OpenMP-script10.sh`

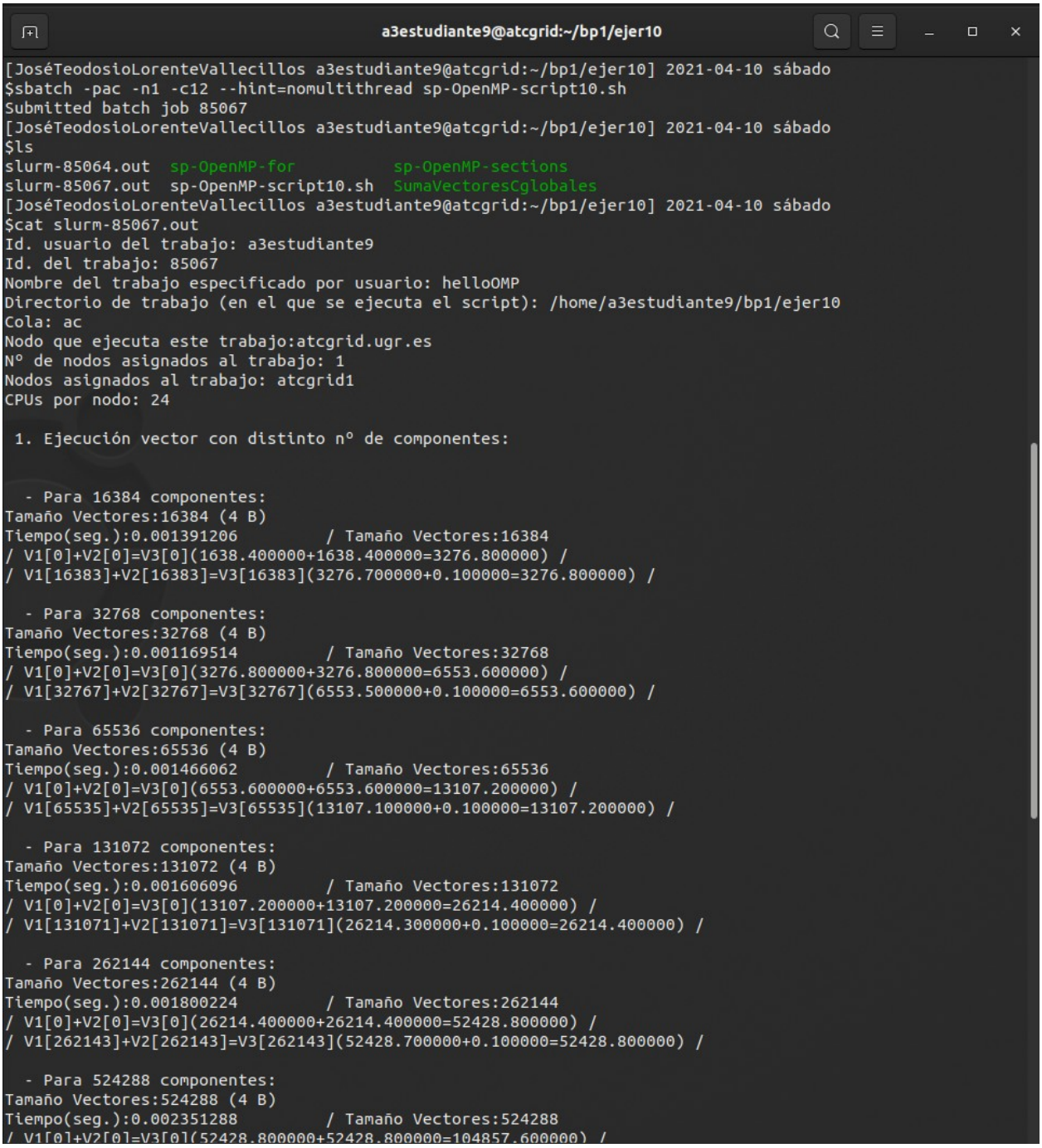
sp-OpenMP-script1... | slurm-85048.out | script-PC | SumaVectoresCglo... | salidasp-OpenMP-s... | salidasp-OpenMP-f... | salidaVectoresGlob...

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

#Obtener información de las variables del entorno del sistema de colas:
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:

a=16384
echo -e "\n 1. Ejecución vector con distinto nº de componentes:\n"
for ((P=14;P<=26;P++))
do
   export OMP_NUM_THREADS=1
   #export OMP_NUM_THREADS=12
   echo -e "\n  - Para $a componentes:"
   srun ./SumaVectoresCglobales $a
   #srun ./sp-OpenMP-for $a
   #srun ./sp-OpenMP-sections $a
   ((a=a*2))
done
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

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


a3estudiante9@atcgrid:~/bp1/ejer10
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer10] 2021-04-10 sábado
$sbatch -pac -n1 -c1 --hint=nomultithread sp-OpenMP-script10.sh
Submitted batch job 85064
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer10] 2021-04-10 sábado
$ls
slurm-85064.out sp-OpenMP-for sp-OpenMP-script10.sh sp-OpenMP-sections SumaVectoresCglobales
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer10] 2021-04-10 sábado
$cat slurm-85064.out
Id. usuario del trabajo: a3estudiante9
Id. del trabajo: 85064
Nombre del trabajo especificado por usuario: helloOMP
Directorio de trabajo (en el que se ejecuta el script): /home/a3estudiante9/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: 2
1. Ejecución vector con distinto nº de componentes:
- Para 16384 componentes:
Tiempo(seg.):0.000443261 / 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) /
- Para 32768 componentes:
Tiempo(seg.):0.000472462 / Tamaño Vectores:32768
/ V1[0]+V2[0]=V3[0](3276.800000+3276.800000=6553.600000) /
/ V1[32767]+V2[32767]=V3[32767](6553.500000+0.100000=6553.600000) /
- Para 65536 componentes:
Tiempo(seg.):0.000669218 / Tamaño Vectores:65536
/ V1[0]+V2[0]=V3[0](6553.600000+6553.600000=13107.200000) /
/ V1[65535]+V2[65535]=V3[65535](13107.100000+0.100000=13107.200000) /
- Para 131072 componentes:
Tiempo(seg.):0.001113865 / Tamaño Vectores:131072
/ V1[0]+V2[0]=V3[0](13107.200000+13107.200000=26214.400000) /
/ V1[131071]+V2[131071]=V3[131071](26214.300000+0.100000=26214.400000) /
- Para 262144 componentes:
Tiempo(seg.):0.001375129 / Tamaño Vectores:262144
/ V1[0]+V2[0]=V3[0](26214.400000+26214.400000=52428.800000) /
/ V1[262143]+V2[262143]=V3[262143](52428.700000+0.100000=52428.800000) /
- Para 524288 componentes:
Tiempo(seg.):0.002744416 / Tamaño Vectores:524288
/ V1[0]+V2[0]=V3[0](52428.800000+52428.800000=104857.600000) /
/ V1[524287]+V2[524287]=V3[524287](104857.500000+0.100000=104857.600000) /
- Para 1048576 componentes:
Tiempo(seg.):0.005052608 / Tamaño Vectores:1048576
/ V1[0]+V2[0]=V3[0](104857.600000+104857.600000=209715.200000) /
/ V1[1048575]+V2[1048575]=V3[1048575](209715.100000+0.100000=209715.200000) /


a3estudiante9@atcgrid:~/bp1/ejer10
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer10] 2021-04-10 sábado
$sbatch -pac -n1 -c12 --hint=nomultithread sp-OpenMP-script10.sh
Submitted batch job 85067
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer10] 2021-04-10 sábado
$ls
slurm-85064.out sp-OpenMP-for sp-OpenMP-sections
slurm-85067.out sp-OpenMP-script10.sh SumaVectoresCglobales
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer10] 2021-04-10 sábado
$cat slurm-85067.out
Id. usuario del trabajo: a3estudiante9
Id. del trabajo: 85067
Nombre del trabajo especificado por usuario: helloOMP
Directorio de trabajo (en el que se ejecuta el script): /home/a3estudiante9/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
1. Ejecución vector con distinto nº de componentes:
- Para 16384 componentes:
Tamaño Vectores:16384 (4 B)
Tiempo(seg.):0.001391206 / 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) /
- Para 32768 componentes:
Tamaño Vectores:32768 (4 B)
Tiempo(seg.):0.001169514 / Tamaño Vectores:32768
/ V1[0]+V2[0]=V3[0](3276.800000+3276.800000=6553.600000) /
/ V1[32767]+V2[32767]=V3[32767](6553.500000+0.100000=6553.600000) /
- Para 65536 componentes:
Tamaño Vectores:65536 (4 B)
Tiempo(seg.):0.001466062 / Tamaño Vectores:65536
/ V1[0]+V2[0]=V3[0](6553.600000+6553.600000=13107.200000) /
/ V1[65535]+V2[65535]=V3[65535](13107.100000+0.100000=13107.200000) /
- Para 131072 componentes:
Tamaño Vectores:131072 (4 B)
Tiempo(seg.):0.001606096 / Tamaño Vectores:131072
/ V1[0]+V2[0]=V3[0](13107.200000+13107.200000=26214.400000) /
/ V1[131071]+V2[131071]=V3[131071](26214.300000+0.100000=26214.400000) /
- Para 262144 componentes:
Tamaño Vectores:262144 (4 B)
Tiempo(seg.):0.001800224 / Tamaño Vectores:262144
/ V1[0]+V2[0]=V3[0](26214.400000+26214.400000=52428.800000) /
/ V1[262143]+V2[262143]=V3[262143](52428.700000+0.100000=52428.800000) /
- Para 524288 componentes:
Tamaño Vectores:524288 (4 B)
Tiempo(seg.):0.002351288 / Tamaño Vectores:524288

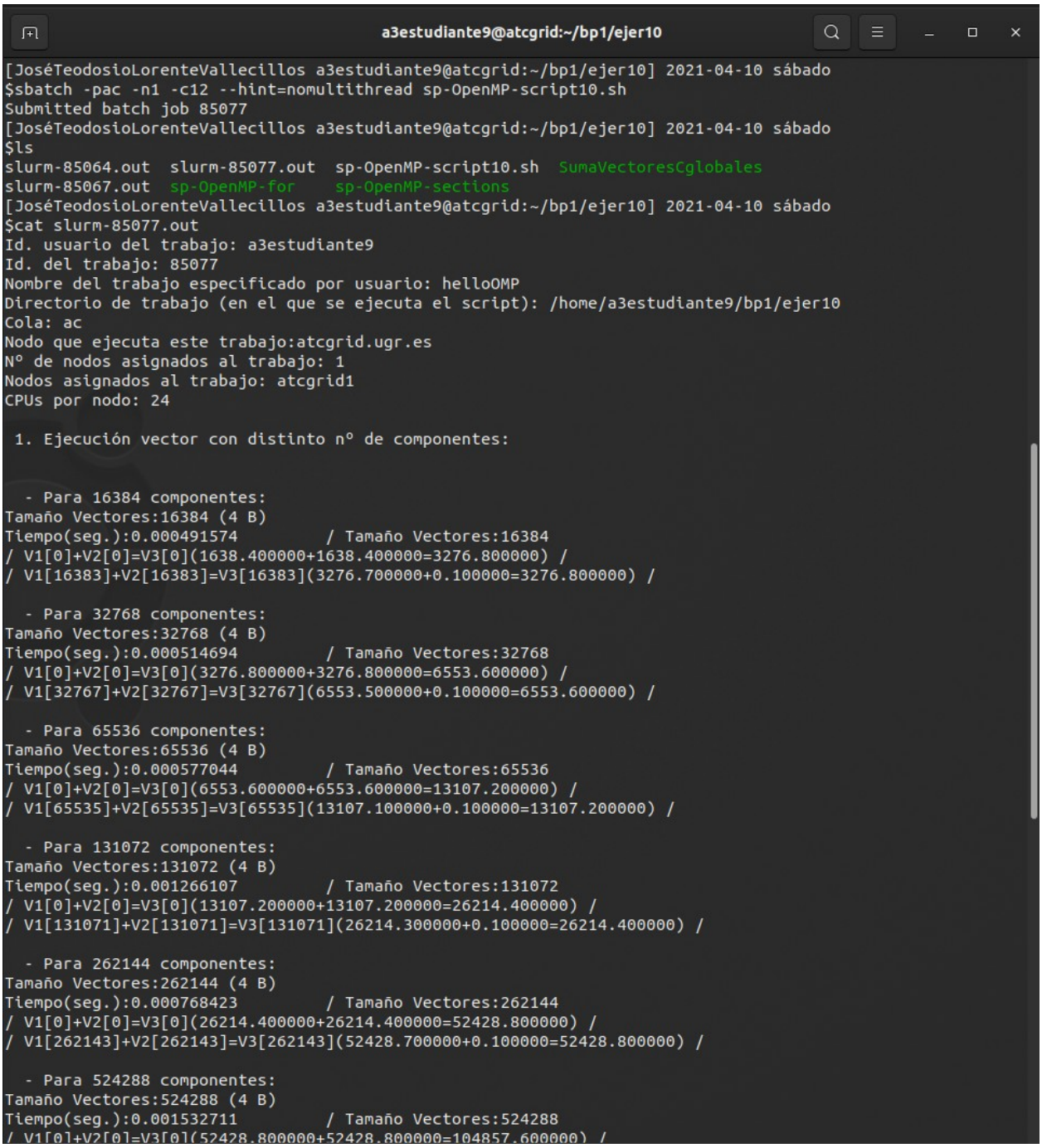
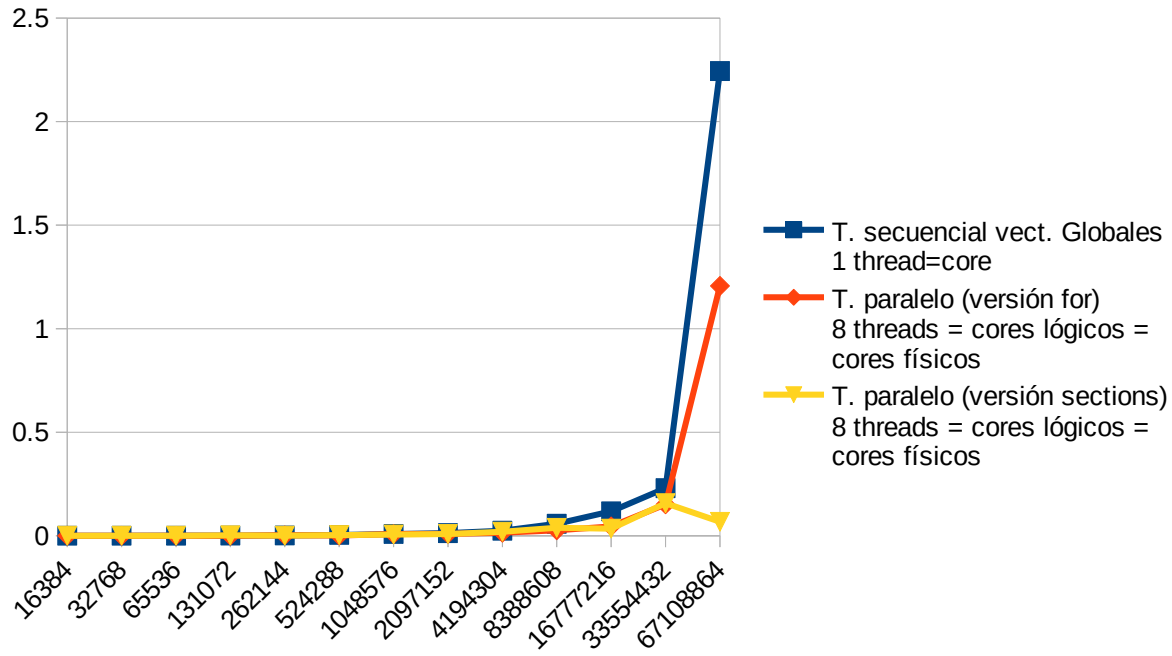

a3estudiante9@atcgrid:~/bp1/ejer10
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer10] 2021-04-10 sábado
$sbatch -pac -n1 -c12 --hint=nomultithread sp-OpenMP-script10.sh
Submitted batch job 85077
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer10] 2021-04-10 sábado
$ls
slurm-85064.out slurm-85077.out sp-OpenMP-script10.sh SumaVectoresCglobales
slurm-85067.out sp-OpenMP-for sp-OpenMP-sections
[JoséTeodosioLorenteVallecillos a3estudiante9@atcgrid:~/bp1/ejer10] 2021-04-10 sábado
$cat slurm-85077.out
Id. usuario del trabajo: a3estudiante9
Id. del trabajo: 85077
Nombre del trabajo especificado por usuario: helloOMP
Directorio de trabajo (en el que se ejecuta el script): /home/a3estudiante9/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
1. Ejecución vector con distinto nº de componentes:
- Para 16384 componentes:
Tamaño Vectores:16384 (4 B)
Tiempo(seg.):0.000491574 / 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) /
- Para 32768 componentes:
Tamaño Vectores:32768 (4 B)
Tiempo(seg.):0.000514694 / Tamaño Vectores:32768
/ V1[0]+V2[0]=V3[0](3276.800000+3276.800000=6553.600000) /
/ V1[32767]+V2[32767]=V3[32767](6553.500000+0.100000=6553.600000) /
- Para 65536 componentes:
Tamaño Vectores:65536 (4 B)
Tiempo(seg.):0.000577044 / Tamaño Vectores:65536
/ V1[0]+V2[0]=V3[0](6553.600000+6553.600000=13107.200000) /
/ V1[65535]+V2[65535]=V3[65535](13107.100000+0.100000=13107.200000) /
- Para 131072 componentes:
Tamaño Vectores:131072 (4 B)
Tiempo(seg.):0.001266107 / Tamaño Vectores:131072
/ V1[0]+V2[0]=V3[0](13107.200000+13107.200000=26214.400000) /
/ V1[131071]+V2[131071]=V3[131071](26214.300000+0.100000=26214.400000) /
- Para 262144 componentes:
Tamaño Vectores:262144 (4 B)
Tiempo(seg.):0.000768423 / Tamaño Vectores:262144
/ V1[0]+V2[0]=V3[0](26214.400000+26214.400000=52428.800000) /
/ V1[262143]+V2[262143]=V3[262143](52428.700000+0.100000=52428.800000) /
- Para 524288 componentes:
Tamaño Vectores:524288 (4 B)
Tiempo(seg.):0.001532711 / Tamaño Vectores:524288

Tabla 1.



Tiempos PC:

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión `for`) 8 threads = cores lógicos = cores físicos	T. paralelo (versión `sections`) 8 threads = cores lógicos = cores físicos
16384	0.000091779	0.000039326	0.000042413
32768	0.000252945	0.000056376	0.000080501
65536	0.000586872	0.000103670	0.000118754
131072	0.000921690	0.000236505	0.002136971
262144	0.001751483	0.002341285	0.000709007
524288	0.004276354	0.001124355	0.003563170
1048576	0.008615794	0.008014133	0.005998104
2097152	0.013070811	0.009432942	0.008495787
4194304	0.024500067	0.014929951	0.019503978
8388608	0.057928717	0.026239688	0.038411438
16777216	0.118045320	0.045472020	0.034285456
33554432	0.229881835	0.151722239	0.157239282
67108864	2.244027137	1.206551842	0.067684256

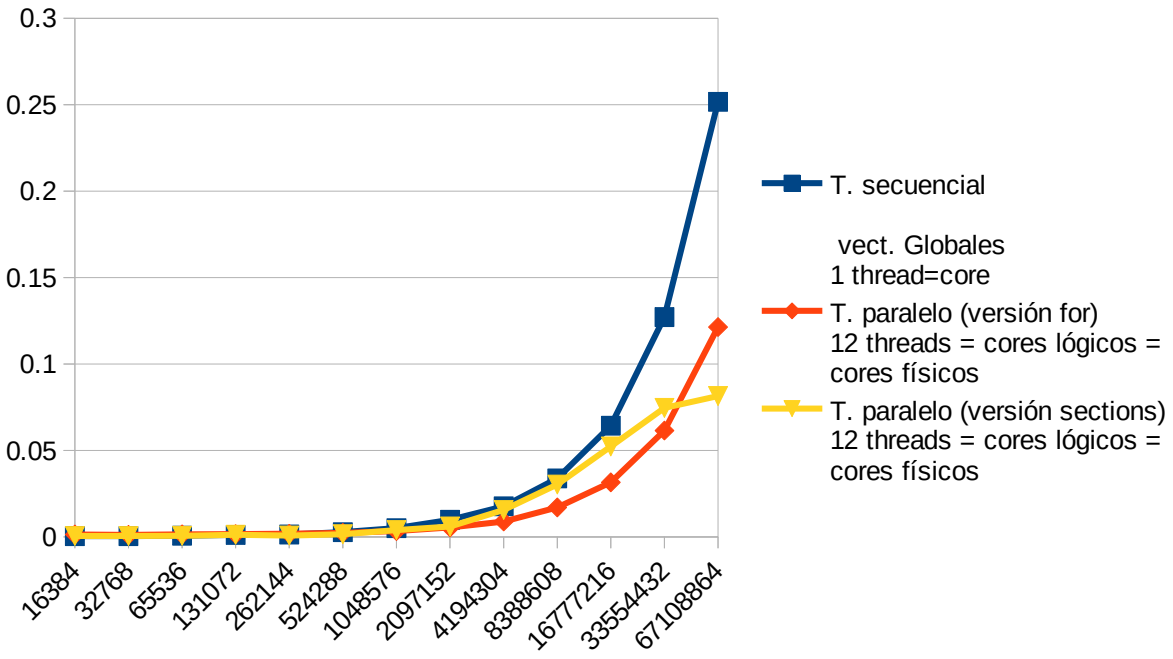


Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados, que debe coincidir con el número de cores físicos y cores lógicos utilizados.

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión `for`) 12 threads = cores lógicos = cores físicos	T. paralelo (versión `sections`) 12 threads = cores lógicos = cores físicos
16384	0.000443261	0.001391206	0.000491574
32768	0.000472462	0.001169514	0.000514694
65536	0.000669218	0.001466062	0.000577044
131072	0.001113865	0.001606096	0.001266107
262144	0.001375129	0.001800224	0.000768423
524288	0.002744416	0.002351288	0.001532711
1048576	0.005052608	0.003407158	0.003952663
2097152	0.009867515	0.005464401	0.006034359
4194304	0.017703911	0.008895148	0.015534885
8388608	0.033819668	0.017041732	0.030279819
16777216	0.064337943	0.031547613	0.052269075
33554432	0.127112658	0.061507672	0.074600261
67108864	0.251663615	0.121404909	0.081456032

11. Rellenar una tabla como la 18Tabla 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 impri-

men en el script de BP0) ¿El tiempo de CPU que se obtiene es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

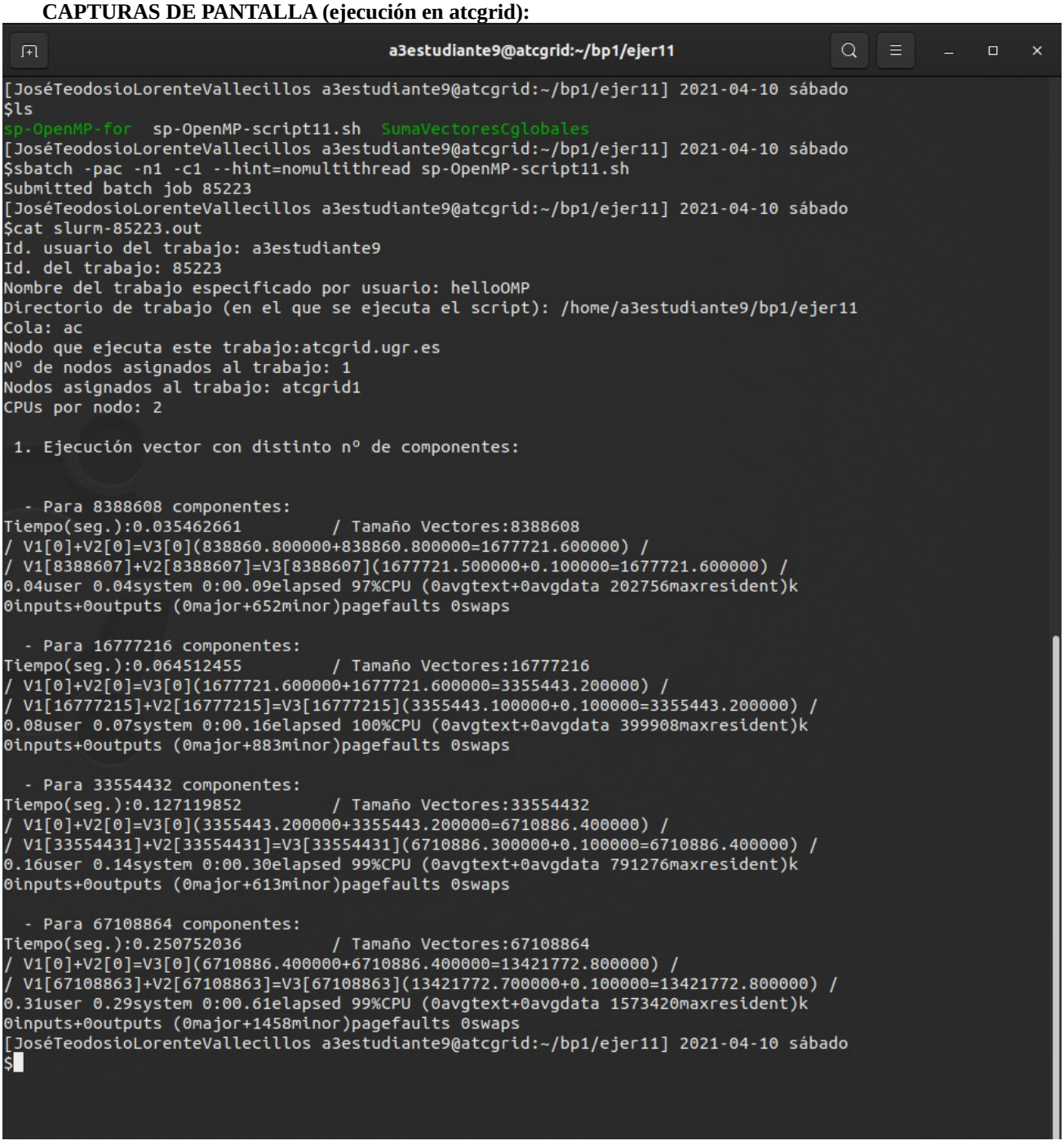
RESPUESTA: Captura del script implementado `sp-OpenMP-script11.sh`

sp-OpenMP-script11.sh

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

#Obtener información de las variables del entorno del sistema de colas:
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:

a=8388608
echo -e "\n 1. Ejecución vector con distinto nº de componentes:\n"
for ((P=23;P<=26;P++))
do
   export OMP_NUM_THREADS=1
   #export OMP_NUM_THREADS=12
   echo -e "\n  - Para $a componentes:"
   srun ./SumaVectoresCglobales $a
   #srun ./sp-OpenMP-for $a
   ((a=a*2))
done
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

CAPTURAS DE PANTALLA (ejecución en atcgrid):



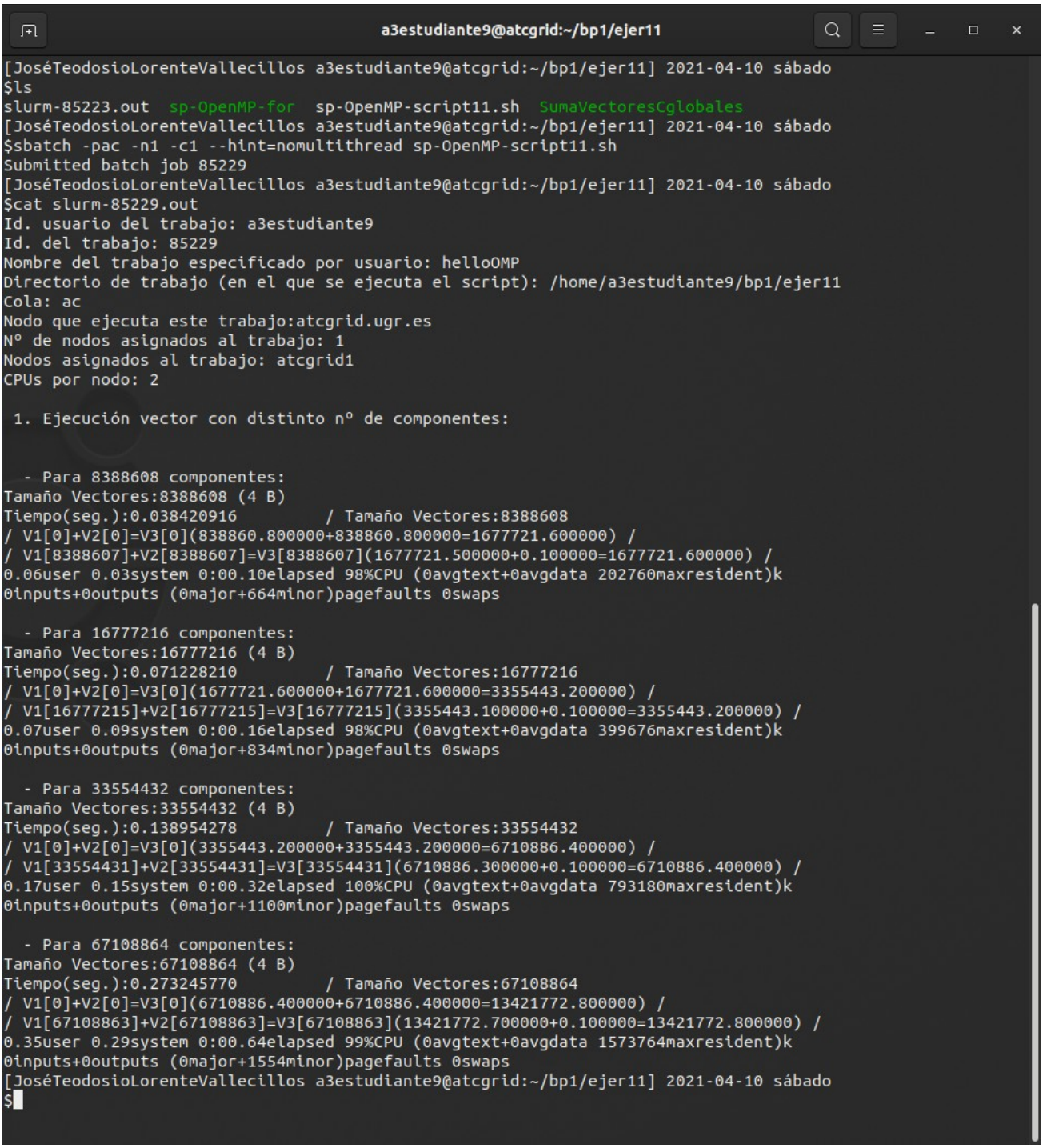


Tabla 3. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados.

Nº de Componentes	Tiempo secuencial vect. Globales 1 thread = 1 core lógico = 1 core físico			Tiempo paralelo/versión `for` 12 Threads = cores lógicos=cores físicos		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
8388608	0.09 elapsed	0.04 user	0.04 system	0.10 elapsed	0.06 user	0.03 system
16777216	0.16 elapsed	0.08 user	0.07 system	0.16 elapsed	0.07 user	0.09 system
33554432	0.30 elapsed	0.16 user	0.14 system	0.32 elapsed	0.17 user	0.15 system
67108864	0.61 elapsed	0.31 user	0.29 system	0.64 elapsed	0.35 user	0.29 system