

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): Pablo Olivares Martínez

Grupo de prácticas y profesor de prácticas: María Isabel García

Fecha de entrega: 14/04/2021

Fecha evaluación en clase: 15/04/2021

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

---

#### Ejercicios basados en los ejemplos del seminario práctico

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

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

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

int main(int argc, char **argv) {

    int i, n = 9;

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

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

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

```
#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();
    }
    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`

```

#include <omp.h>
#include <stdio.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("Single del resultado ejecutada por el thread %d\n", omp_get_thread_num());
        }
    }
    return 0;
}

```

## CAPTURAS DE PANTALLA:

```

Terminal
(base) [Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 1 -
Herramientas de programación paralela I: Directivas OpenMP/bp1/ejer2] 2021-04-0
8 jueves
$ gcc -o singleModificado -fopenmp singleModificado.c
(base) [Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 1 -
Herramientas de programación paralela I: Directivas OpenMP/bp1/ejer2] 2021-04-0
8 jueves
$ ./singleModificado
Introduce valor de inicialización a: 8
Single ejecutada por el thread 1
Después de la región parallel:
b[0] = 8      b[1] = 8      b[2] = 8      b[3] = 8      b[4] = 8      b
[5] = 8 b[6] = 8      b[7] = 8      b[8] = 8
Single del resultado ejecutada por el thread 7
(base) [Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 1 -
Herramientas de programación paralela I: Directivas OpenMP/bp1/ejer2] 2021-04-0
8 jueves
$

```

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`

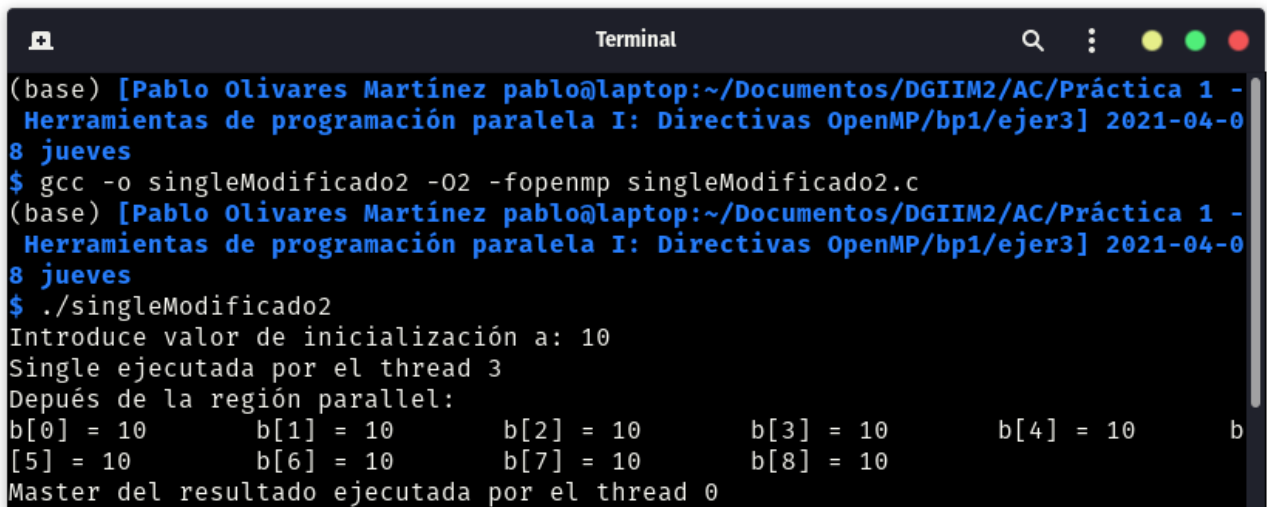
```
#include <omp.h>
#include <stdio.h>

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("Master del resultado ejecutada por el thread %d\n", omp_get_thread_num());
        }
    }
    return 0;
}
```

**CAPTURAS DE PANTALLA:**


```

Terminal
(base) [Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 1 - Herramientas de programación paralela I: Directivas OpenMP/bp1/ejer3] 2021-04-08 jueves
$ gcc -o singleModificado2 -O2 -fopenmp singleModificado2.c
(base) [Pablo Olivares Martínez pablo@laptop:~/Documentos/DGIIM2/AC/Práctica 1 - Herramientas de programación paralela I: Directivas OpenMP/bp1/ejer3] 2021-04-08 jueves
$ ./singleModificado2
Introduce valor de inicialización a: 10
Single ejecutada por el thread 3
Después de la región parallel:
b[0] = 10      b[1] = 10      b[2] = 10      b[3] = 10      b[4] = 10      b
[5] = 10      b[6] = 10      b[7] = 10      b[8] = 10
Master del resultado ejecutada por el thread 0

```

**RESPUESTA A LA PREGUNTA:**

Mientras que en el ejercicio anterior el programa ejecutaba la función en cualquier hebra disponible, con la directiva master obligamos a que se ejecute en la hebra maestra, es decir, la 0.

4. ¿Por qué si se elimina directiva barrier en el ejemplo master.c la suma que se calcula e imprime no siempre es correcta? Responda razonadamente.

**RESPUESTA:**

Esto se debe a que la directiva master carece de barrera implícita. Por eso, cuando ejecutamos el programa, la directiva parallel realiza una suma por cada una de las hebras que van a realizar la operación (sumalocal). Cuando llegan a suma, éstas se van añadiendo una a una gracias a la directiva atomic, para evitar errores y condiciones de carrera. Finalmente, la barrera hace que se espere a que todas las hebras hayan realizado la suma, sin embargo, al eliminarla, la primera hebra que llegue pasará a la impresión del resultado, la cual es errónea pues no ha esperado a sumar todas las sumas locales.

**1.1.1**

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

5. El programa secuencial C del Listado 1 calcula la suma de dos vectores ( $v3 = v1 + v2$ ;  $v3(i) = v1(i) + v2(i)$ ,  $i=0, \dots, N-1$ ). Generar el ejecutable del programa del Listado 1 para **vectores globales**. Usar time (Lección 3/ Tema 1) en la línea de comandos para obtener, en atcgrid, el tiempo de ejecución (*elapsed time*) y el tiempo de CPU del usuario y del sistema generado. Obtenga los tiempos para vectores con 10000000 componentes. ¿La suma de los tiempos de CPU del usuario y del sistema es menor, mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

**CAPTURAS DE PANTALLA:**

```
e1estudiante21@atcgrid:~/bp1/ejer5
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer5] 2021-04-08 jueves
$ srun -pac -Aac time ./SumaVectores 10000000
Tiempo:0.040824541 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](2.85
1159+0.533320=3.384480) / / V1[9999999]+V2[9999999]=V3[9999999](1.174845+1.05174
8=2.226593) /
0.53user 0.05system 0:00.59elapsed 99%CPU (0avgtext+0avgdata 239396maxresident)k
40inputs+0outputs (0major+614minor)pagefaults 0swaps
```

**RESPUESTA:**

El tiempo de ejecución (elapsed time) es igual a la suma del tiempo de usuario (user time) y tiempo del sistema (sys time), como podemos ver en las capturas.  $0.53 \text{ (user)} + 0.05 \text{ (system)} = 0.59 \text{ (elapsed)}$  (el error de la suma se debe a la falta de precisión).

- 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):

```
e1estudiante21@atcgrid:~/bp1/ejer5
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer5] 2021-04-08 jueves
$ gcc -o SumaVectores -O2 SumaVectoresC.c -lrt
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer5] 2021-04-08 jueves
$ gcc -O2 -S SumaVectoresC.c -lrt
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer5] 2021-04-08 jueves
$ srun -pac -Aac time ./SumaVectores 10
Tiempo:0.000400796 / Tamaño Vectores:10 / V1[0]+V2[0]=V3[0](4.941989+1.1
03230=6.045219) / / V1[9]+V2[9]=V3[9](0.247077+0.377707=0.624784) /
0.00user 0.00system 0:00.00elapsed 33%CPU (0avgtext+0avgdata 4736maxresident)k
24inputs+0outputs (0major+211minor)pagefaults 0swaps
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer5] 2021-04-08 jueves
$ srun -pac -Aac time ./SumaVectores 10000000
Tiempo:0.040640047 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](8.00
8916+0.448341=8.457257) / / V1[9999999]+V2[9999999]=V3[9999999](0.197405+1.06582
7=1.263232) /
0.52user 0.04system 0:00.57elapsed 99%CPU (0avgtext+0avgdata 240256maxresident)k
0inputs+0outputs (0major+836minor)pagefaults 0swaps
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer5] 2021-04-08 jueves
$
```

**RESPUESTA:** cálculo de los MIPS y los MFLOPS

$$\text{MIPS}(10) = (10 \cdot 6 + 3) / (0.000400796 \cdot 10^6) = 0,1572 \text{ MIPS}$$

$$\text{MFLOPS}(10) = 3 \cdot 10 / (0.000400796 \cdot 10^6) = 0,07485 \text{ MFLOPS}$$

$$\text{MIPS}(10000000) = (10000000 \cdot 6 + 3) / (0.040640047 \cdot 10^6) = 1476,3763 \text{ MIPS}$$

$$\text{MFLOPS}(10000000) = 3 \cdot 10000000 / (0.040640047 \cdot 10^6) = 738,1881 \text{ MFLOPS}$$

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

```
.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 ( $v3 = v1 + v2$ ;  $v3(i) = v1(i) + v2(i)$ ,  $i = 0, \dots, N-1$ ) usando las directivas `parallel` y `for`. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Como en el código del Listado 1 se debe obtener el tiempo (*elapsed time*) que supone el cálculo de la suma. Para obtener este tiempo usar la función `omp_get_wtime()`, que proporciona el estándar OpenMP, en lugar de `clock_gettime()`. NOTAS: (1) el número de componentes  $N$  de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante,  $v3$ , para varios tamaños pequeños de los vectores (por ejemplo,  $N = 8$  y  $N = 11$ ); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de  $v1$ ,  $v2$  y  $v3$  (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

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



```

//Inicializar vectores en paralelo
if (N < 9) {
    #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;
    }
} else {
    srand(time(0));
    #pragma omp parallel for
    for (i = 0; i < N; i++) {
        v1[i] = rand() / ((double)rand());
        v2[i] = rand() / ((double)rand()); //printf("%d:%f,%f/",i,v1[i],v2[i]);
    }
}

cgt1 = omp_get_wtime(); // Dato inicial para elapsed time

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

cgt2 = omp_get_wtime(); // Dato final para elapsed time
ncgt = cgt2 - cgt1; // Elapsed time

//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 (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):

```

e1estudiante21@atcgrid:~/bp1/ejer7
e1estudiante21@atcgrid:~/bp1/... x Terminal x Terminal x
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer7] 2021-04-08 jueves
$ gcc -O2 -fopenmp -o sp-OpenMP-for sp-OpenMP-for.c
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer7] 2021-04-08 jueves
$ srun -pac -Aac ./sp-OpenMP-for 8
Tiempo:0.000663541 / Tamaño Vectores:8
/ V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) /
/ V1[1]+V2[1]=V3[1](0.900000+0.700000=1.600000) /
/ V1[2]+V2[2]=V3[2](1.000000+0.600000=1.600000) /
/ V1[3]+V2[3]=V3[3](1.100000+0.500000=1.600000) /
/ V1[4]+V2[4]=V3[4](1.200000+0.400000=1.600000) /
/ V1[5]+V2[5]=V3[5](1.300000+0.300000=1.600000) /
/ V1[6]+V2[6]=V3[6](1.400000+0.200000=1.600000) /
/ V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer7] 2021-04-08 jueves
$ srun -pac -Aac ./sp-OpenMP-for 11
Tiempo:0.000650275 / Tamaño Vectores:11 / V1[0]+V2[0]=V3[0](0.509817+0.7
89911=1.299728) / / V1[10]+V2[10]=V3[10](0.685327+1.193301=1.878628) /
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer7] 2021-04-08 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`

```
//Inicializar vectores en paralelo
if (N < 9) {
    #pragma omp parallel sections private(i)
    {
        #pragma omp section
        for (i = 0; i < N; i=i+4) {
            v1[i] = N * 0.1 + i * 0.1;
            v2[i] = N * 0.1 - i * 0.1;
        }

        #pragma omp section
        for (i = 1; i < N; i=i+4) {
            v1[i] = N * 0.1 + i * 0.1;
            v2[i] = N * 0.1 - i * 0.1;
        }

        #pragma omp section
        for (i = 2; i < N; i=i+4) {
            v1[i] = N * 0.1 + i * 0.1;
            v2[i] = N * 0.1 - i * 0.1;
        }

        #pragma omp section
        for (i = 3; i < N; i=i+4) {
            v1[i] = N * 0.1 + i * 0.1;
            v2[i] = N * 0.1 - i * 0.1;
        }
    }
} else {
```

```
} else {
    srand(time(0));
    #pragma omp parallel sections private(i)
    {
        #pragma omp section
        for (i = 0; i < N; i=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=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=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=i+4) {
            v1[i] = rand() / ((double)rand());
            v2[i] = rand() / ((double)rand()); //printf("%d:%f,%f/", i, v1[i], v2[i]);
        }
    }
}
```

```

cgt1 = omp_get_wtime(); // Dato inicial para elapsed time

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

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

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

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

cgt2 = omp_get_wtime(); // Dato final para elapsed time
ncgt = cgt2 - cgt1; // Elapsed time

//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 (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):

```

e1estudiante21@atcgrid:~/bp1/ejer8
e1estudiante21@atcgrid:~/bp1/... x Terminal x Terminal x
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer8] 2021-04-08 jueves
$ gcc -O2 -fopenmp -o sp-OpenMP-sections sp-OpenMP-sections.c
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer8] 2021-04-08 jueves
$ srun -pac -Aac ./sp-OpenMP-sections 8
Tiempo:0.000685398 / Tamaño Vectores:8
/ V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) /
/ V1[1]+V2[1]=V3[1](0.900000+0.700000=1.600000) /
/ V1[2]+V2[2]=V3[2](1.000000+0.600000=1.600000) /
/ V1[3]+V2[3]=V3[3](1.100000+0.500000=1.600000) /
/ V1[4]+V2[4]=V3[4](1.200000+0.400000=1.600000) /
/ V1[5]+V2[5]=V3[5](1.300000+0.300000=1.600000) /
/ V1[6]+V2[6]=V3[6](1.400000+0.200000=1.600000) /
/ V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer8] 2021-04-08 jueves
$ srun -pac -Aac ./sp-OpenMP-sections 11
Tiempo:0.000588957 / Tamaño Vectores:11 / V1[0]+V2[0]=V3[0](0.055965+0.6
17793=0.673759) / / V1[10]+V2[10]=V3[10](0.758751+17.171594=17.930345) /
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer8] 2021-04-08 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 im-

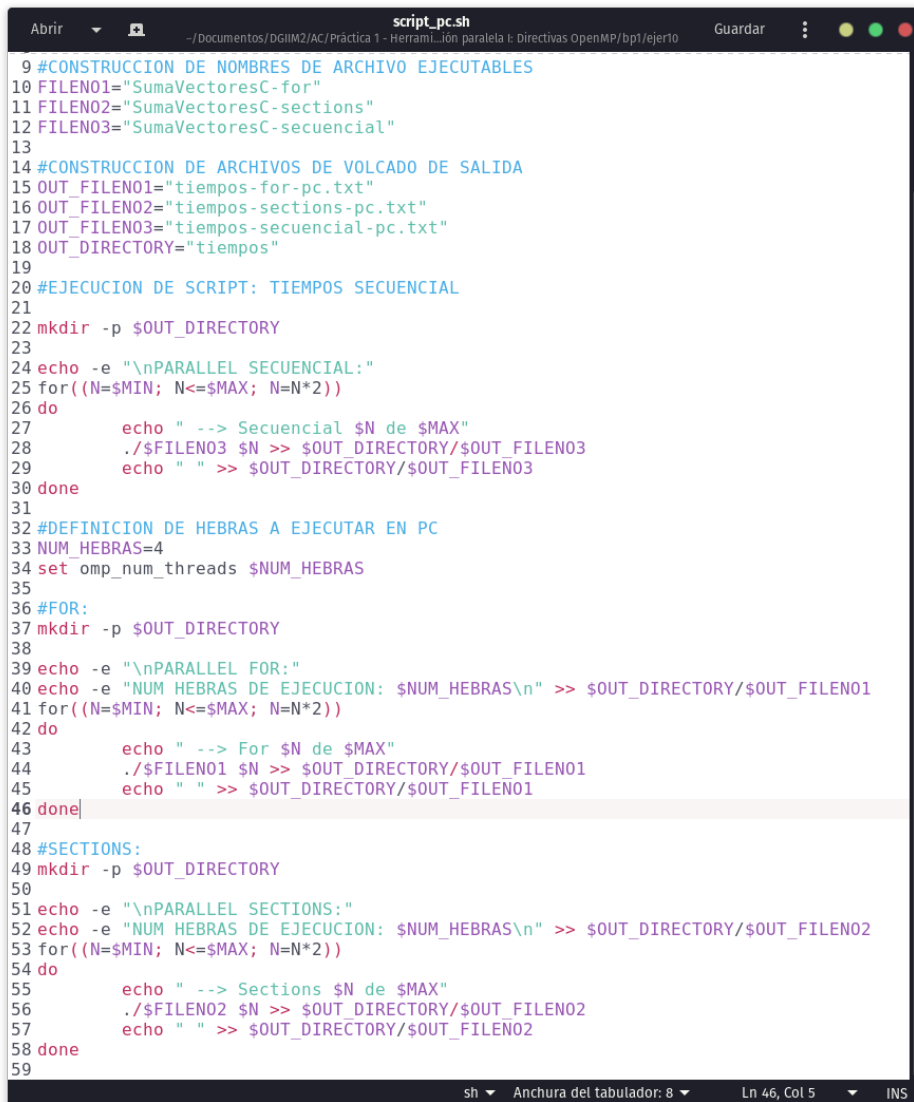
plementado en el ejercicio 8? Razone su respuesta. NOTA: Al contestar piense sólo en el código, no piense en el computador en el que lo va a ejecutar.

### RESPUESTA:

Podrían utilizar todas las hebras de las que disponga el computador utilizado. Sin embargo, para el for nos sería más útil utilizar tantas hebras como iteraciones haya, mientras que en sections nos conviene usar tantas como tareas tengamos.

10. Rellenar una tabla como la Tabla 213 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



```

9 #CONSTRUCCION DE NOMBRES DE ARCHIVO EJECUTABLES
10 FILEN01="SumaVectoresC-for"
11 FILEN02="SumaVectoresC-sections"
12 FILEN03="SumaVectoresC-secuencial"
13
14 #CONSTRUCCION DE ARCHIVOS DE VOLCADO DE SALIDA
15 OUT_FILEN01="tiempos-for-pc.txt"
16 OUT_FILEN02="tiempos-sections-pc.txt"
17 OUT_FILEN03="tiempos-secuencial-pc.txt"
18 OUT_DIRECTORY="tiempos"
19
20 #EJECUCION DE SCRIPT: TIEMPOS SECUENCIAL
21
22 mkdir -p $OUT_DIRECTORY
23
24 echo -e "\nPARALLEL SECUENCIAL:"
25 for((N=$MIN; N<=$MAX; N=N*2))
26 do
27     echo " --> Secuencial $N de $MAX"
28     ./FILEN03 $N >> $OUT_DIRECTORY/$OUT_FILEN03
29     echo " " >> $OUT_DIRECTORY/$OUT_FILEN03
30 done
31
32 #DEFINICION DE HEBRAS A EJECUTAR EN PC
33 NUM_HEBRAS=4
34 set omp_num_threads $NUM_HEBRAS
35
36 #FOR:
37 mkdir -p $OUT_DIRECTORY
38
39 echo -e "\nPARALLEL FOR:"
40 echo -e "NUM HEBRAS DE EJECUCION: $NUM_HEBRAS\n" >> $OUT_DIRECTORY/$OUT_FILEN01
41 for((N=$MIN; N<=$MAX; N=N*2))
42 do
43     echo " --> For $N de $MAX"
44     ./FILEN01 $N >> $OUT_DIRECTORY/$OUT_FILEN01
45     echo " " >> $OUT_DIRECTORY/$OUT_FILEN01
46 done
47
48 #SECTIONS:
49 mkdir -p $OUT_DIRECTORY
50
51 echo -e "\nPARALLEL SECTIONS:"
52 echo -e "NUM HEBRAS DE EJECUCION: $NUM_HEBRAS\n" >> $OUT_DIRECTORY/$OUT_FILEN02
53 for((N=$MIN; N<=$MAX; N=N*2))
54 do
55     echo " --> Sections $N de $MAX"
56     ./FILEN02 $N >> $OUT_DIRECTORY/$OUT_FILEN02
57     echo " " >> $OUT_DIRECTORY/$OUT_FILEN02
58 done
59

```

```

script_secuencial_atcgrid.sh
~/Documentos/DGIIIM2/AC/Práctica 1 - ...lela I: Directivas OpenMP/bp1/ejer10
Abrir Guardar

10 #Obtener información de las variables del entorno del sistema de colas:
11 echo "Id. usuario del trabajo: $SLURM_JOB_USER"
12 echo "Id. del trabajo: $SLURM_JOBID"
13 echo "Nombre del trabajo especificado por usuario: $SLURM_JOB_NAME"
14 echo "Directorio de trabajo (en el que se ejecuta el script): $SLURM_SUBMIT_DIR"
15 echo "Cola: $SLURM_JOB_PARTITION"
16 echo "Nodo que ejecuta este trabajo: $SLURM_SUBMIT_HOST"
17 echo "Nº de nodos asignados al trabajo: $SLURM_JOB_NUM_NODES"
18 echo "Nodos asignados al trabajo: $SLURM_JOB_NODELIST"
19 echo "CPUs por nodo: $SLURM_JOB_CPUS_PER_NODE"
20
21 #DEFINICION DE MINIMO Y MAXIMO
22 MIN=16384
23 MAX=67108864
24
25 #CONSTRUCCION DE NOMBRES DE ARCHIVO EJECUTABLES
26 FILEN03="SumaVectoresC-secuencial"
27
28 #EJECUCION DE SCRIPT: TIEMPOS SECUENCIAL
29 echo -e "\n$SECUENCIAL:"
30 for((N=$MIN; N<=$MAX; N=N*2))
31 do
32     srun ./FILEN03 $N
33 done
sh Anchura del tabulador: 8 Ln 29, Col 12 INS

```

```

script_parallel_atcgrid.sh
~/Documentos/DGIIIM2/AC/Práctica 1 - ...lela I: Directivas OpenMP/bp1/ejer10
Abrir Guardar

10 #Obtener información de las variables del entorno del sistema de colas:
11 echo "Id. usuario del trabajo: $SLURM_JOB_USER"
12 echo "Id. del trabajo: $SLURM_JOBID"
13 echo "Nombre del trabajo especificado por usuario: $SLURM_JOB_NAME"
14 echo "Directorio de trabajo (en el que se ejecuta el script): $SLURM_SUBMIT_DIR"
15 echo "Cola: $SLURM_JOB_PARTITION"
16 echo "Nodo que ejecuta este trabajo: $SLURM_SUBMIT_HOST"
17 echo "Nº de nodos asignados al trabajo: $SLURM_JOB_NUM_NODES"
18 echo "Nodos asignados al trabajo: $SLURM_JOB_NODELIST"
19 echo "CPUs por nodo: $SLURM_JOB_CPUS_PER_NODE"
20
21 #CONSTRUCCION DE NOMBRES DE ARCHIVO EJECUTABLES
22 FILEN01="SumaVectoresC-for"
23 FILEN02="SumaVectoresC-sections"
24
25 #DEFINICION DE LOS TIEMPOS MIN Y MAX
26 MIN=16384
27 MAX=67108864
28
29 #EJECUCION DE SCRIPT: TIEMPOS FOR
30 echo -e "PARALLEL FOR:"
31 for((N=$MIN; N<=$MAX; N=N*2))
32 do
33     srun ./FILEN01 $N
34 done
35
36 #EJECUCION DE SCRIPT: TIEMPOS SECTIONS
37 echo -e "\n\nPARALLEL SECTIONS:"
38 for((N=$MIN; N<=$MAX; N=N*2))
39 do
40     srun ./FILEN02 $N
41 done
sh Anchura del tabulador: 8 Ln 41, Col 5 INS

```

**(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)**

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

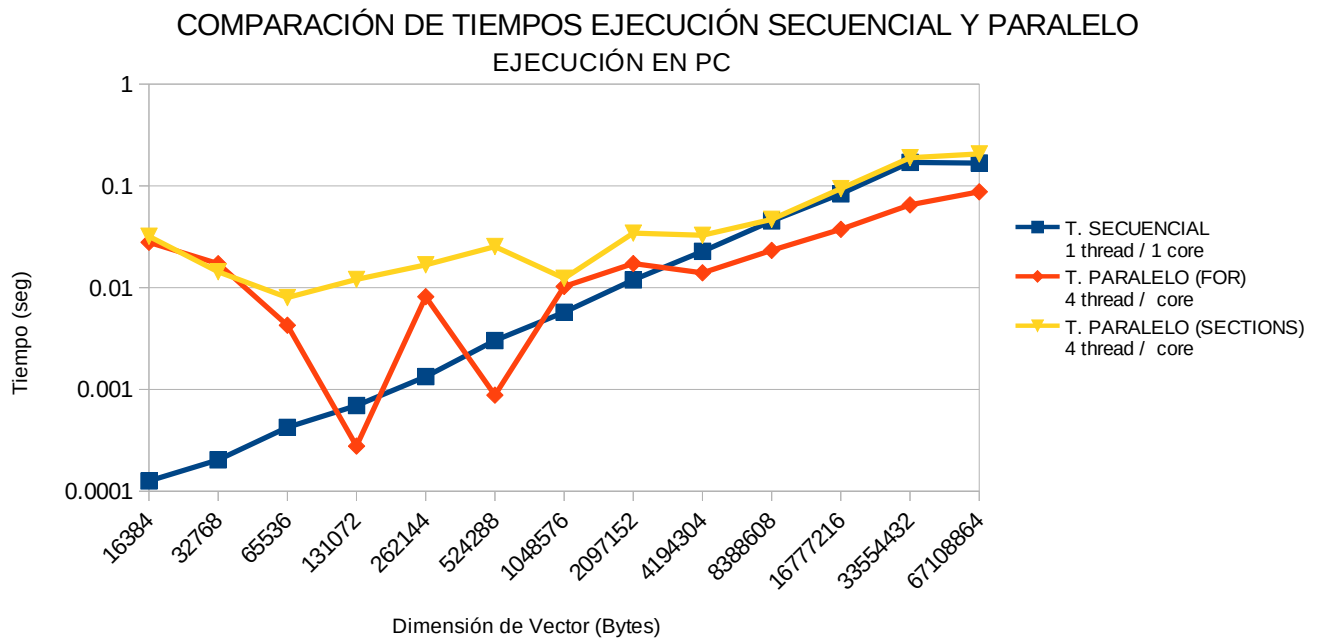
```

e1estudiante21@atcgrid:~/bp1/ejer10
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer10] 2021-04-14 miércoles
$ sbatch -p ac script_secuencial_atcgrid.sh --cpus-per-task=1 -n1 --hint=nomultithread
Submitted batch job 91231
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer10] 2021-04-14 miércoles
$ sbatch -p ac script_parallel_atcgrid.sh --cpus-per-task=12 -n1 --hint=nomultithread
Submitted batch job 91233
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer10] 2021-04-14 miércoles

```

**Tabla 2.** Tiempos de ejecución PC 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.

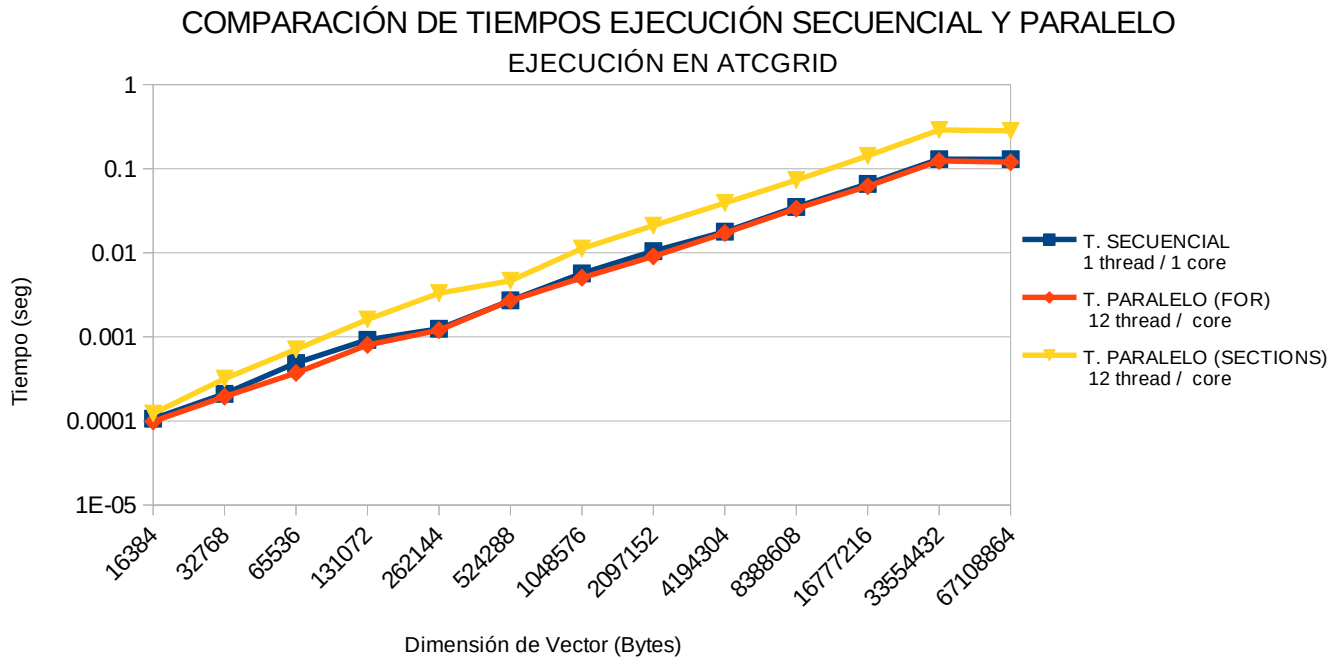
Nº de Componentes	T. secuencial vect. Globales 1 thread = 1 core	T. paralelo (versión for) 4 threads = 4 cores lógicos = 4 cores físicos	T. paralelo (versión sections) 12 threads = 12 cores lógicos = 12 cores físicos
16384	0	0.03	0.03
32768	0	0.02	0.01
65536	0	0	0.01
131072	0	0	0.01
262144	0	0.01	0.02
524288	0	0	0.03
1048576	0.01	0.01	0.01
2097152	0.01	0.02	0.03
4194304	0.02	0.01	0.03
8388608	0.05	0.02	0.05
16777216	0.08	0.04	0.09
33554432	0.17	0.07	0.19
67108864	0.17	0.09	0.21



**Tabla 3.** Tiempos de ejecución ATCGRID 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	0	0
32768	0	0	0
65536	0	0	0
131072	0	0	0
262144	0	0	0
524288	0	0	0
1048576	0.01	0.01	0.01
2097152	0.01	0.01	0.02
4194304	0.02	0.02	0.04
8388608	0.04	0.03	0.07
16777216	0.07	0.06	0.14
33554432	0.13	0.12	0.29
67108864	0.13	0.12	0.28





NOTA: Al realizar copia-pegar de los datos de la hoja de cálculo al documento, se redondea automáticamente. Los valores sin alterar se encuentran en la hoja de cálculo y el slurm.

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.

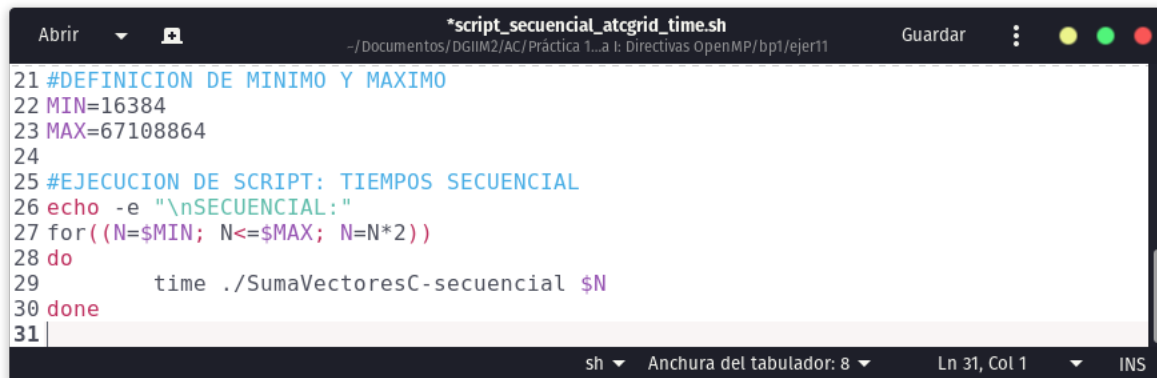
**RESPUESTA:** Captura del script implementado `sp-OpenMP-script11.sh`

```

Abrir  script_parallel_atcgrid_time.sh  Guardar
~/Documentos/DGIIIM2/AC/Práctica 1 ...la 1: Directivas OpenMP/bp1/ejer11

20
21 #DEFINICION DE LOS TIEMPOS MIN Y MAX
22 MIN=16384
23 MAX=67108864
24
25 #EJECUCION DE SCRIPT: TIEMPOS FOR
26 echo -e "PARALLEL FOR:"
27 for((N=$MIN; N<=$MAX; N=N*2))
28 do
29     time ./SumaVectoresC-for $N
30 done
31
sh  Anchura del tabulador: 8  Ln 26, Col 19  INS

```



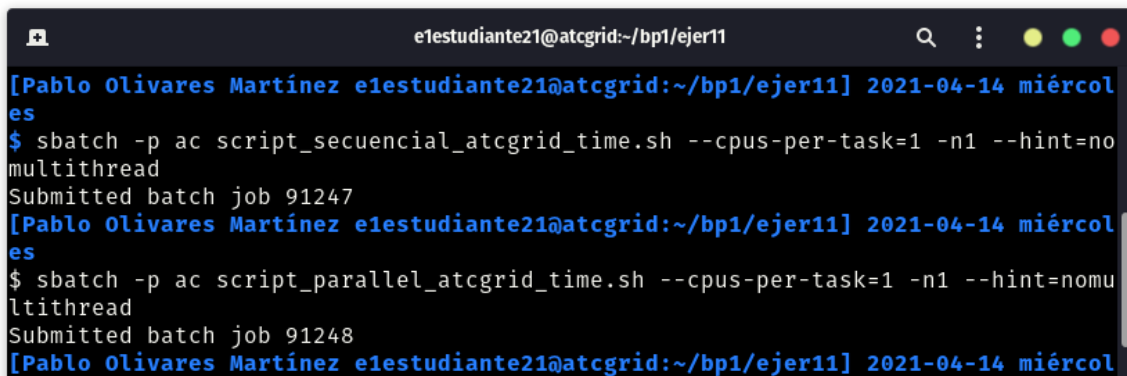
```

21 #DEFINICION DE MINIMO Y MAXIMO
22 MIN=16384
23 MAX=67108864
24
25 #EJECUCION DE SCRIPT: TIEMPOS SECUENCIAL
26 echo -e "\nSECUENCIAL:"
27 for((N=$MIN; N<=$MAX; N=N*2))
28 do
29     time ./SumaVectoresC-secuencial $N
30 done
31

```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

CAPTURAS DE PANTALLA (ejecución en atcgrid):



```

e1estudiante21@atcgrid:~/bp1/ejer11
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer11] 2021-04-14 miércoles
$ sbatch -p ac script_secuencial_atcgrid_time.sh --cpus-per-task=1 -n1 --hint=no
multithread
Submitted batch job 91247
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer11] 2021-04-14 miércoles
$ sbatch -p ac script_parallel_atcgrid_time.sh --cpus-per-task=1 -n1 --hint=nomu
ltithread
Submitted batch job 91248
[Pablo Olivares Martínez e1estudiante21@atcgrid:~/bp1/ejer11] 2021-04-14 miércoles

```

**Tabla 4.** 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 = 12 cores lógicos = 12 cores físicos		
	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>
<b>16384</b>	0m0.061s	0m0.000s	0m0.004s	0m0.063s	0m0.002s	0m0.005s
<b>32768</b>	0m0.033s	0m0.001s	0m0.001s	0m0.049s	0m0.001s	0m0.003s
<b>65536</b>	0m0.039s	0m0.001s	0m0.002s	0m0.039s	0m0.003s	0m0.003s
<b>131072</b>	0m0.039s	0m0.000s	0m0.004s	0m0.039s	0m0.000s	0m0.007s
<b>262144</b>	0m0.078s	0m0.005s	0m0.000s	0m0.056s	0m0.004s	0m0.005s
<b>524288</b>	0m0.089s	0m0.005s	0m0.004s	0m0.083s	0m0.010s	0m0.006s
<b>1048576</b>	0m0.089s	0m0.007s	0m0.007s	0m0.078s	0m0.011s	0m0.014s
<b>2097152</b>	0m0.111s	0m0.017s	0m0.010s	0m0.089s	0m0.032s	0m0.015s
<b>4194304</b>	0m0.110s	0m0.028s	0m0.022s	0m0.100s	0m0.060s	0m0.030s
<b>8388608</b>	0m0.115s	0m0.048s	0m0.043s	0m0.122s	0m0.085s	0m0.070s
<b>16777216</b>	0m0.189s	0m0.094s	0m0.074s	0m0.160s	0m0.162s	0m0.117s
<b>33554432</b>	0m0.338s	0m0.164s	0m0.152s	0m0.296s	0m0.324s	0m0.220s
<b>67108864</b>	0m0.330s	0m0.129s	0m0.181s	0m0.301s	0m0.304s	0m0.254s

En nuestro caso hay veces que el tiempo de CPU obtenido de la suma del tiempo del sistema de usuario y del sistema es mayor que el tiempo elapsed debido a que el sistema dedica tiempo a la creación y destrucción de hebras.