

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): Daniel Alconchel Vázquez

Grupo de prácticas y profesor de prácticas: 1

Fecha de entrega:

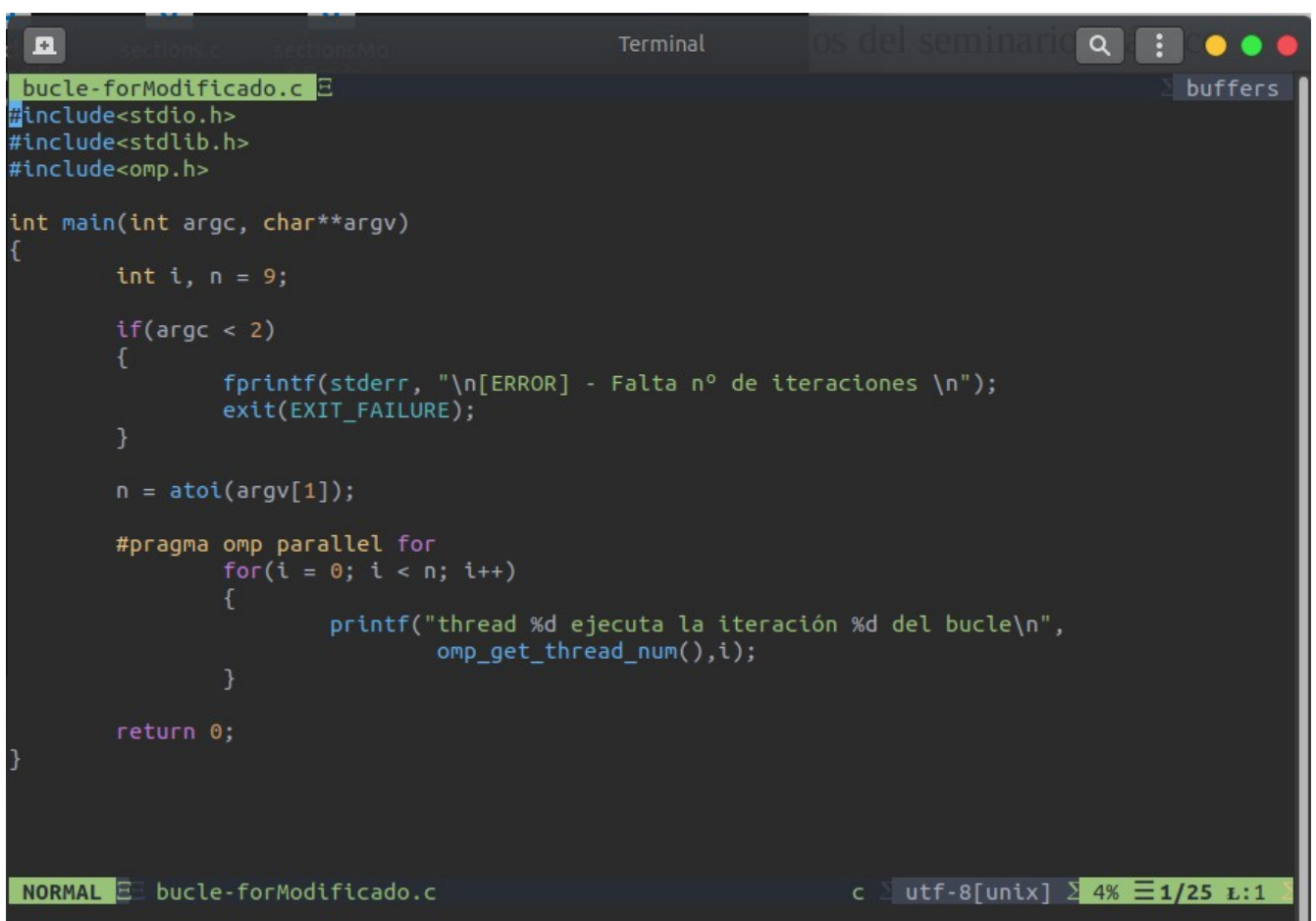
Fecha evaluación en clase:

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`



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

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

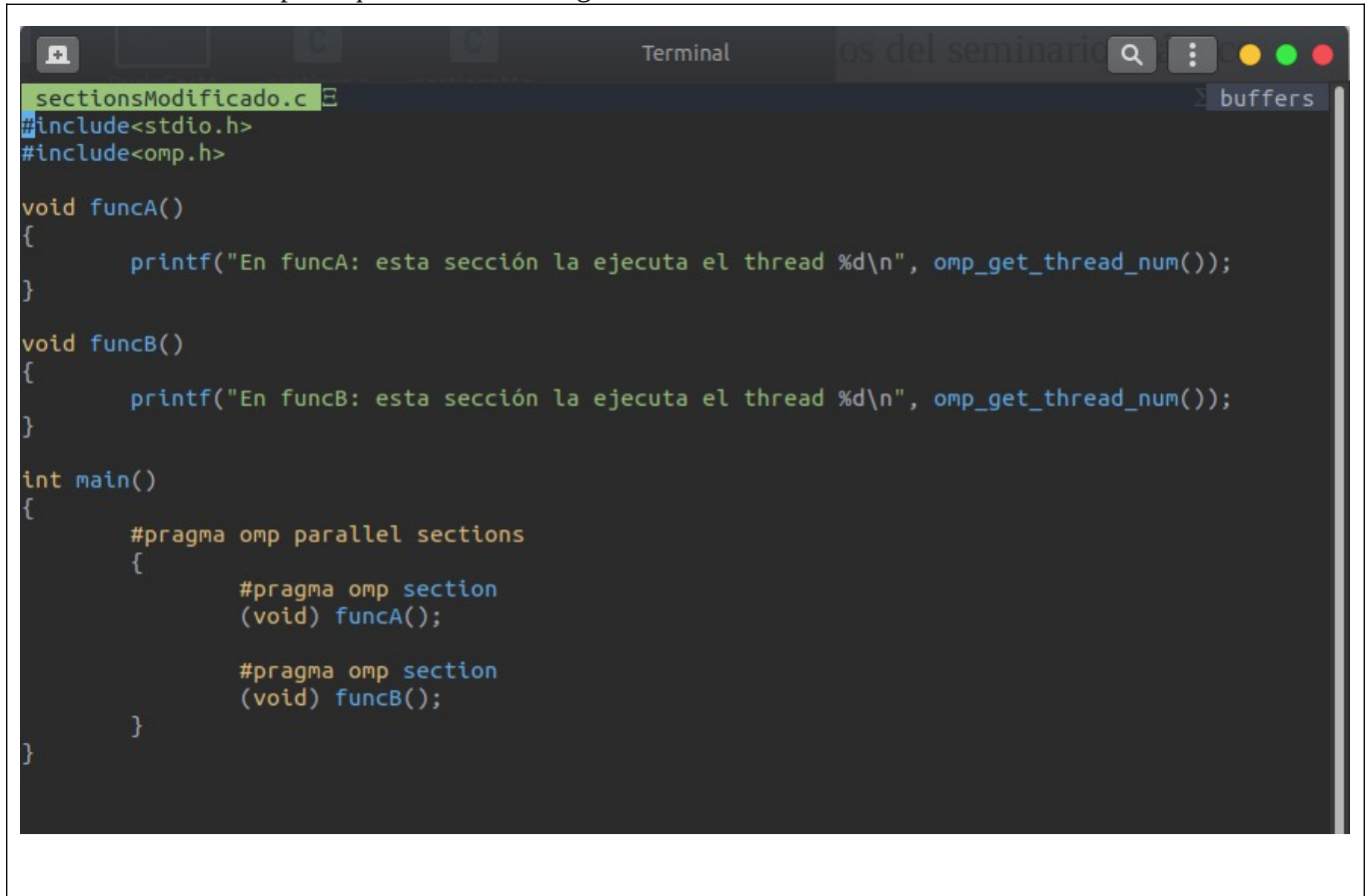
    if(argc < 2)
    {
        fprintf(stderr, "\n[ERROR] - Falta nº de iteraciones \n");
        exit(EXIT_FAILURE);
    }

    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`



```
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();
    }
}
```

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`

```

singleModificado.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("MODIFICACION CODIGO!! ID THREAD = %d\n", omp_get_thread_num());
            printf("Después de la región parallel:\n");
            for(i=0;i<n;i++) printf("b[%d] = %d\t",i,b[i]);
            printf("\n");
        }
    }

    return(0);
}

```

## CAPTURAS DE PANTALLA:

```

[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp1/ejer2] 2021-04-13 martes
$ gcc -fopenmp -O2
single.c singleModificado.c SingleModificado.png
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp1/ejer2] 2021-04-13 martes
$ gcc -fopenmp -O2 singleModificado.c
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp1/ejer2] 2021-04-13 martes
$ ls
a.out single.c singleModificado.c SingleModificado.png
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp1/ejer2] 2021-04-13 martes
$ ./a.out
Introduce valor de inicialización a: 14
Single ejecutada por el thread 5
MODIFICACION CODIGO!! ID THREAD = 3
Después de la región parallel:
b[0] = 14    b[1] = 14    b[2] = 14    b[3] = 14    b[4] = 14    b[5] = 14    b[6] = 14    b[7] = 14    b[8] = 14
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp1/ejer2] 2021-04-13 martes
$

```

- 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`

```

singleModificado2.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("MODIFICACION [2] CODIGO!! ID THREAD = %d\n", omp_get_thread_num());
            printf("Después de la región parallel:\n");
            for(i=0;i<n;i++) printf("b[%d] = %d\t",i,b[i]);
            printf("\n");
        }
    }

    return(0);
}

```

### CAPTURAS DE PANTALLA:

```

Terminal
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp1/ejer3] 2021-04-13 martes
$ gcc -fopenmp -O2 singleModificado2.c
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp1/ejer3] 2021-04-13 martes
$ ls
a.out singleModificado2.c singleModificado2.png
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp1/ejer3] 2021-04-13 martes
$ ./a.out
Introduce valor de inicialización a: 8
Single ejecutada por el thread 5
MODIFICACION [2] CODIGO!! ID THREAD = 0
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
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp1/ejer3] 2021-04-13 martes
$

```

```

Terminal
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp1/ejer3] 2021-04-13 martes
$ ./a.out
Introduce valor de inicialización a:
1
Single ejecutada por el thread 2
MODIFICACION [2] CODIGO!! ID THREAD = 0
Después de la región parallel:
b[0] = 1    b[1] = 1    b[2] = 1    b[3] = 1    b[4] = 1    b[5] = 1 b[6] = 1    b[7] = 1    b[8] = 1
[DanielAlconchelVázquez daniel@daniel-GL63-8SE:~/Git/DGIIM/Segundo/2 Cuatrimestre/AC/Prácticas/bp1/ejer3] 2021-04-13 martes
$

```

**RESPUESTA A LA PREGUNTA:** En el caso del ejercicio anterior, la hebra que ejecutaba el programa podría ser cualquiera de las disponibles, sin embargo, con la directiva **master**, obligamos a que siempre sea la hebra 0 la que ejecute dicha sección de código

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 **master** no dispone de barrera implícita, por lo que es necesario añadir una directiva **barrier** antes para su correcto funcionamiento. El código de **master.c** realiza la suma de las componentes de un vector en paralelo, para ello, se calcula la suma con un bucle **for** al que se le aplica la directiva **omp for**, por tanto, cada hebra de las disponibles en el sistema se encarga de realizar una parte de las iteraciones del bucle (las que le asignen). Es posible que alguna de las hebras termina antes que otra, por lo que escribe el valor de **sumalocal** en la variable **suma**, mientras que las demás pueden seguir calculando.

Como en el código modificado, no disponemos de una directiva **barrier**, la hebra que finalice antes saltará a la directiva **master** sin esperar a los que no han acabado e imprimirá el mensaje del resultado por pantalla.

Es por eso por lo que es necesario la directiva **barrier** antes de **master**, para evitar condición de carrera y que todas las hebras se sincronicen adecuadamente.

### 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:**

```
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer5] 2021-04-13 martes
$ ls
GLOBAL_SumaVectoresC.c
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer5] 2021-04-13 martes
$ gcc -fopenmp -O2 GLOBAL_SumaVectoresC.c
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer5] 2021-04-13 martes
$ ls
a.out GLOBAL_SumaVectoresC.c
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer5] 2021-04-13 martes
$ time srun -p ac a.out 10000000
Tamaño Vectores:10000000 (4 B)
Tiempo:0.041968369 || 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    0m0.218s
user    0m0.008s
sys      0m0.007s
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer5] 2021-04-13 martes
$
```

**RESPUESTA:** Vemos que el real time es la suma del user time y el sys time .

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 có-

algo 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):

```
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer6] 2021-04-13 martes
$ gcc -fopenmp GLOBAL_SumaVectoresC.c
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer6] 2021-04-13 martes
$ srun -p ac a.out 10000000
Tamaño Vectores:10000000 (4 B)
Tiempo:0.074617460 || 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) ||
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer6] 2021-04-13 martes
$ srun -p ac a.out 10
Tamaño Vectores:10 (4 B)
Tiempo:0.000393934 || 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) ||
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer6] 2021-04-13 martes
$
```

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

- MIPS (10 elementos):  $[10 \cdot 6 + 3] / (0.000393934 \cdot 10^6) = 0.1599252667$  MIPS
- MIPS (10000000 elementos):  $[10000000 \cdot 6 + 3] / (0.074617460 \cdot 10^6) = 80.41002391$  MIPS
- MFLOPS (10 elementos):  $3 \cdot 10 / (0.000393934 \cdot 10^6) = 0.07615488889$  MFLOPS
- MFLOPS (10000000 elementos):  $3 \cdot 10000000 / (0.074617460 \cdot 10^6) = 402.0499185$  MFLOPS

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

```
.L4:
    pxor    %xmm0, %xmm0
    movapd  %xmm1, %xmm2
    movapd  %xmm1, %xmm7
    cvtsi2sdl    %eax, %xmm0
    mulsd   %xmm3, %xmm0
    addsd   %xmm0, %xmm2
    subsd   %xmm0, %xmm7
    movsd   %xmm2, v1(,%rax,8)
    movsd   %xmm7, v2(,%rax,8)
    addq    $1, %rax
    cmpl    %eax, %ebp
    ja      .L4
    movq    %rsp, %rsi
    xorl    %edi, %edi
    call    clock_gettime
```

- 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

```
/**
 * @brief MODIFICACION: INICIALIZACION DE VECTORES CON PARALELIZACION
 */
#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;
}

cgt1 = omp_get_wtime();

/**
 * @brief MODIFICACION: CALCULO DE LA SUMA DE VECTORES CON PARALELIZACION
 */
#pragma omp parallel for
for(i=0; i<N; i++){
    v3[i] = v1[i] + v2[i];
}

cgt2 = omp_get_wtime();
ncgt= cgt2-cgt1;

//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,v1[i],v2[i],v3[i]);
}
else
    printf("Tiempo:%11.9f\t || Tamaño Vectores:%u\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",
        ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
```

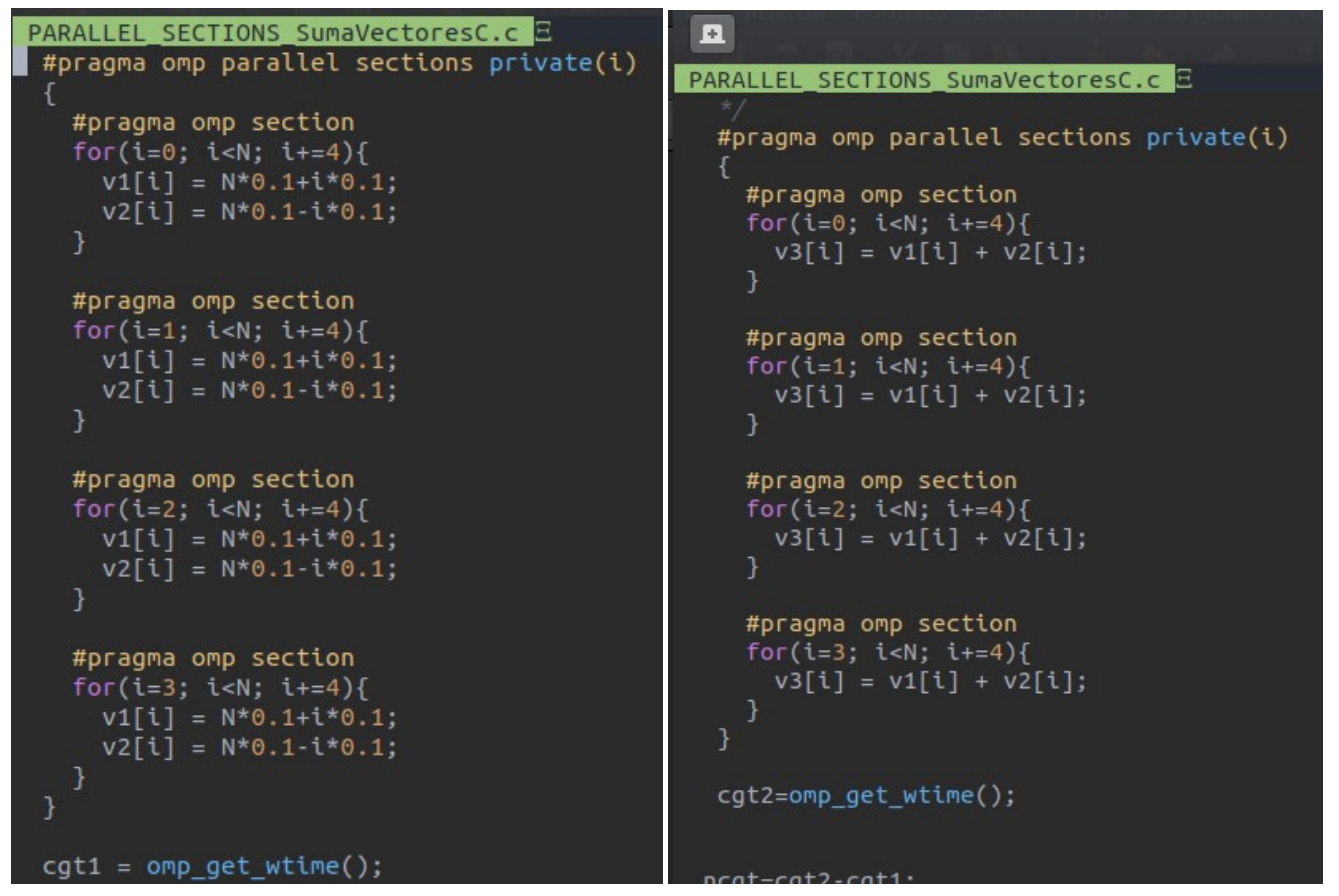
(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer7] 2021-04-13 martes
$ gcc -fopenmp PARALLEL_FOR_SumaVectoresC.c
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer7] 2021-04-13 martes
$ ls
a.out PARALLEL_FOR_SumaVectoresC.c
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer7] 2021-04-13 martes
$ srun -p ac a.out 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000440076 || 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) ||
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer7] 2021-04-13 martes
$ srun -p ac a.out 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000612702 || 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) ||
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer7] 2021-04-13 martes
$
```

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`



The image shows two side-by-side screenshots of a code editor with a dark background. The left screenshot shows the initialization of two vectors, `v1` and `v2`, in four parallel sections. The right screenshot shows the calculation of the sum vector `v3` in four parallel sections, followed by a timing measurement using `omp_get_wtime()`.

```

PARALLEL SECTIONS SumaVectoresC.c
#pragma omp parallel sections private(i)
{
    #pragma omp section
    for(i=0; i<N; 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+=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+=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+=4){
        v1[i] = N*0.1+i*0.1;
        v2[i] = N*0.1-i*0.1;
    }
}

cgt1 = omp_get_wtime();

PARALLEL SECTIONS SumaVectoresC.c
/*
#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];
    }
}

cgt2=omp_get_wtime();

ncgt=cgt2-cgt1;

```



(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer8] 2021-04-13 martes
$ gcc -fopenmp PARALLEL_SECTIONS_SumaVectoresC.c
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer8] 2021-04-13 martes
$ srun -p ac a.out 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000663292      || 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) ||
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer8] 2021-04-13 martes
$ srun -p ac a.out 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000571422      || 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) ||
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer8] 2021-04-13 martes
$
```

9. ¿Cuántos threads y cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 7? Razone su respuesta. ¿Cuántos threads y cuantos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta. NOTA: Al contestar piense sólo en el código, no piense en el computador en el que lo va a ejecutar.

**RESPUESTA:** En el ejercicio 7 podríamos utilizar un máximo de N threads, sin embargo, este número se limita por el número de cores lógicos del ordenador en el que estemos trabajando, o de la constante definida por OpenMP. Por otra parte, el ejercicio 8 siempre usará un máximo de 4 threads, porque hemos dividido el “for” en cuatro secciones paralelas.

10. Rellenar una tabla como la Tabla 211 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**

```

Terminal
script_parallel_atcgrid.sh
#script_parallel_atcgrid.sh
#SECUENCIAL=SECUENCIAL_
FILENO1=$OMP_PARALLELSOMP_FORSTARGET
FILENO2=$OMP_PARALLELSOMP_SECTIONSSTARGET
FILENO3=$SECUENCIALSTARGET
#DIRECTORY="b1n"
#DEFINICION DE LOS TIEMPOS MIN Y MAX
MIN=16384
MAX=67108864
#EJECUCION DE SCRIPT: TIEMPOS FOR
echo -e ">>> EJECUCION DE PROGRAMA CON OMP PARALLEL [FOR] <<<"
for((N=$MIN; N<=$MAX; N=N*2))
do
    srun ./SFILENO1 $N
done
#EJECUCION DE SCRIPT: TIEMPOS SECTIONS
echo -e "\n\n>>> EJECUCION DE PROGRAMA CON OMP PARALLEL [SECTIONS] <<<"
for((N=$MIN; N<=$MAX; N=N*2))
do
    srun ./SFILENO2 $N
done
NORMAL: script_parallel_atcgrid.sh sh utf-8[unix] 100% 50/50 L:1 5/5 tra

script_secuencial_atcgrid.sh
#script_secuencial_atcgrid.sh
#Obtengo información de las variables del entorno de las 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: $SLURM_JOB_CPUS_PER_NODE"
#DEFINICION DE MINIMO Y MAXIMO
MIN=16384
MAX=67108864
#CONSTRUCCION DE NOMBRES DE ARCHIVO EJECUTABLES
TARGETS="SumaVectoresC"
OMP_PARALLEL="PARALLEL_"
OMP_FOR="FOR_"
OMP_SECTIONS="SECTIONS_"
SECUENCIAL="SECUENCIAL_"
FILENO1=$OMP_PARALLELSOMP_FORSTARGET
FILENO2=$OMP_PARALLELSOMP_SECTIONSSTARGET
FILENO3=$SECUENCIALSTARGET
NORMAL: script_secuencial_atcgrid.sh sh utf-8[unix] 22% 10/44 L:1 5/5 tra

Terminal
script_pc.sh
#script_pc.sh
#BORRADO DE ARCHIVOS POR SI EXISTIAN PREVIAMENTE
rm -r $OUT_DIRECTORY 2> /dev/null
#COMPILEACION DE OBJETOS
echo -e ">>> COMPILANDO OBJETOS <<<"
make 2> /dev/null
#EJECUCION DE SCRIPT: TIEMPOS SECUENCIAL
mkdirt -p $OUT_DIRECTORY
echo -e "\n\n>>> EJECUCION DE PROGRAMA CON OMP PARALLEL [SECUENCIAL] <<<"
for((N=$MIN; N<=$MAX; N=N*2))
do
    echo -e "--> OBTENIENDO TIEMPO PARA DIMENSION [SN] DE [$MAX]"
    ./SDIRECTORY/SFILENO3 $N >> $OUT_DIRECTORY/$OUT_FILENO3
    echo -e ">>> $OUT_DIRECTORY/$OUT_FILENO3"
done
#DEFINICION DE HEBRAS A EJECUTAR EN PC
NUM_HEBRAS=4
set omp_num_threads $NUM_HEBRAS
#EJECUCION DE SCRIPT: TIEMPOS FOR
mkdirt -p $OUT_DIRECTORY
echo -e "\n\n>>> EJECUCION DE PROGRAMA CON OMP PARALLEL [FOR] <<<"
echo -e "NUM HEBRAS DE EJECUCION: $NUM_HEBRAS\n" >> $OUT_DIRECTORY/$OUT_FILENO1
for((N=$MIN; N<=$MAX; N=N*2))
do
    echo -e "--> OBTENIENDO TIEMPO PARA DIMENSION [SN] DE [$MAX]"
    ./SDIRECTORY/SFILENO1 $N >> $OUT_DIRECTORY/$OUT_FILENO1
    echo -e ">>> $OUT_DIRECTORY/$OUT_FILENO1"
done
#EJECUCION DE SCRIPT: TIEMPOS SECTIONS
mkdirt -p $OUT_DIRECTORY
echo -e "\n\n>>> EJECUCION DE PROGRAMA CON OMP PARALLEL [SECTIONS] <<<"
echo -e "NUM HEBRAS DE EJECUCION: $NUM_HEBRAS\n" >> $OUT_DIRECTORY/$OUT_FILENO2
for((N=$MIN; N<=$MAX; N=N*2))
do
    echo -e "--> OBTENIENDO TIEMPO PARA DIMENSION [SN] DE [$MAX]"
    ./SDIRECTORY/SFILENO2 $N >> $OUT_DIRECTORY/$OUT_FILENO2
    echo -e ">>> $OUT_DIRECTORY/$OUT_FILENO2"
done
#ELIMINACION DE CODIGOS EJECUTABLES
echo -e "\n\n>>> ELIMINANDO OBJETOS <<<"
make clean 2> /dev/null
NORMAL: script_pc.sh sh utf-8[unix] 98% 86/87 L:1
  
```

**(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)****CAPTURAS DE PANTALLA (mostrar la ejecución en atcgrid – envío(s) a la cola):**

```

[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer10] 2021-04-13 martes
$ gcc -O2 -fopenmp PARALLEL_FOR_SumaVectoresC.c -o PARALLEL_FOR_SumaVectoresC -lrt
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer10] 2021-04-13 martes
$ gcc -O2 -fopenmp PARALLEL_SECTIONS_SumaVectoresC.c -o PARALLEL_SECTIONS_SumaVectoresC -lrt
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer10] 2021-04-13 martes
$ gcc -O2 -fopenmp SECUENCIAL_SumaVectoresC.c -o SECUENCIAL_SumaVectoresC -lrt
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer10] 2021-04-13 martes
$ ls
PARALLEL_FOR_SumaVectoresC PARALLEL_SECTIONS_SumaVectoresC script_parallel_atcgrid.sh script_secuencial_atcgrid.sh SECUENCIAL_SumaVectoresC
PARALLEL_FOR_SumaVectoresC.c PARALLEL_SECTIONS_SumaVectoresC.c script_pc.sh script.sh SECUENCIAL_SumaVectoresC.c
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer10] 2021-04-13 martes
$ sbatch -p ac script_secuencial_atcgrid.sh --cpus-per-task=1 -n1 --hint=nomultithread
Submitted batch job 89672
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer10] 2021-04-13 martes
$ sbatch -p ac script_parallel_atcgrid.sh --cpus-per-task=12 -n1 --hint=nomultithread
Submitted batch job 89673
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer10] 2021-04-13 martes
$ ls slurm*
slurm-89672.out slurm-89673.out
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer10] 2021-04-13 martes
$
  
```

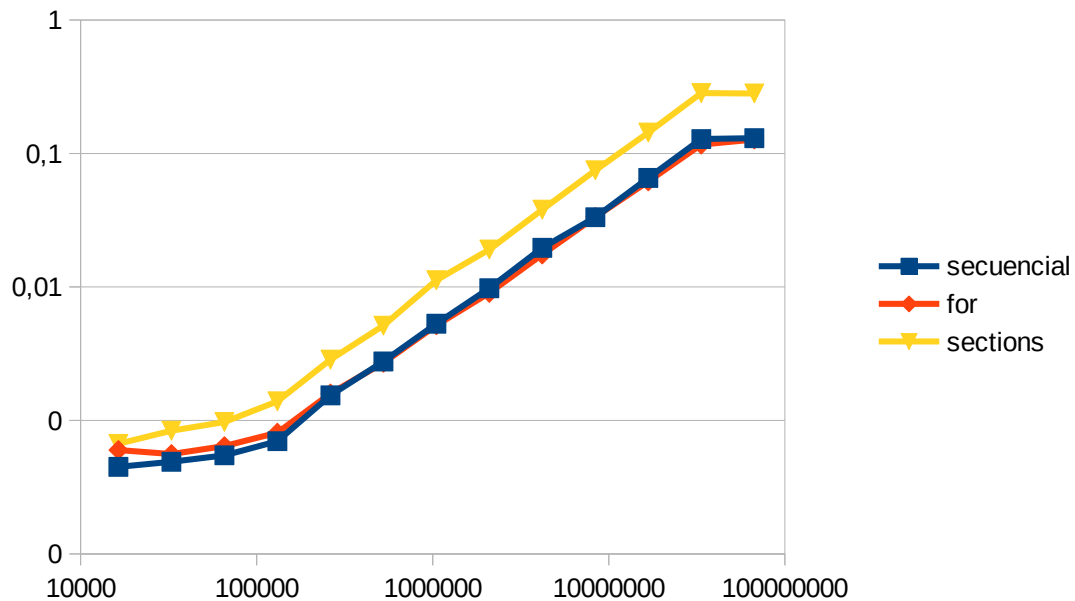
Tabla 2. ATCGRID

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión for) ¿?threads = cores lógicos = cores físicos	T. paralelo (versión sections) ¿?threads = cores lógicos = cores físicos
16384	0.000448477	0.000600278	0.000667289
32768	0.000489302	0.000559647	0.000837926
65536	0.000547549	0.000642359	0.000973675
131072	0.000699152	0.000808634	0.001389045
262144	0.001539467	0.001592282	0.002853766
524288	0.002764394	0.002699208	0.005128793
1048576	0.005307320	0.005162366	0.011213049
2097152	0.009778892	0.009024363	0.019060675
4194304	0.019621829	0.017501481	0.037919253
8388608	0.033273680	0.033412635	0.074806694
16777216	0.065479011	0.061793864	0.143204108
33554432	0.128302326	0.116410881	0.283238929
67108864	0.130105611	0.127027225	0.281467564

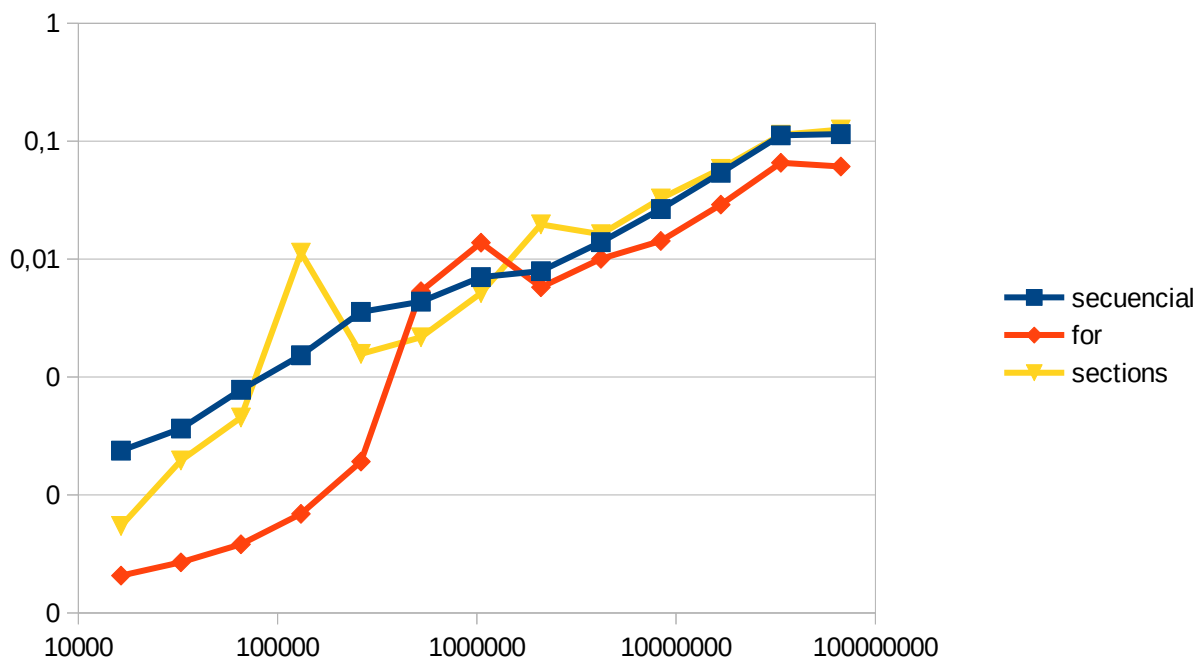
Tabla 2. PC

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión for) ¿?threads = cores lógicos = cores físicos	T. paralelo (versión sections) ¿?threads = cores lógicos = cores físicos
16384	0.000237405	0.000020686	0.000054503
32768	0.000365535	0.000026845	0.000196031
65536	0.000780026	0.000038180	0.000453229
131072	0.001529545	0.000069145	0.011345412
262144	0.003568252	0.000192076	0.001569505
524288	0.004360590	0.005339374	0.002173137
1048576	0.007037373	0.013763870	0.005141597
2097152	0.007907395	0.005774981	0.019686984
4194304	0.013920437	0.010062030	0.016251668
8388608	0.026572006	0.014253940	0.032679719
16777216	0.053917251	0.028992832	0.058223839
33554432	0.112057229	0.065540334	0.113405367
67108864	0.114658298	0.060971111	0.125329773

**atcgrid:**



**pc:**



11. Rellenar una tabla como la 13Tabla 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`

```

#DEFINICION DE MINIMO Y MAXIMO
MIN=16384
MAX=67108864

#CONSTRUCCION DE NOMBRES DE ARCHIVO EJECUTABLES
TARGET="SumaVectoresC"
OMP_PARALLEL="PARALLEL_"
OMP_FOR="FOR_"
OMP_SECTIONS="SECTIONS_"
SECUENCIAL="SECUENCIAL_"

FILEN01=$OMP_PARALLEL$OMP_FOR$TARGET
FILEN02=$OMP_PARALLEL$OMP_SECTIONS$TARGET
FILEN03=$SECUENCIAL$TARGET

DIRECTORY="bin"

#EJECUCION DE SCRIPT: TIEMPOS SECUENCIAL
echo -e "\n>>> EJECUCION DE PROGRAMA [SECUENCIAL] <<<"
for((N=$MIN; N<=$MAX; N=N*2))
do
    time ./FILEN03 $N
done

```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

CAPTURAS DE PANTALLA (ejecución en atcgrid):

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

Nº de Componentes	Tiempo secuencial vect. Globales 1 thread = 1 core lógico = 1 core físico			Tiempo paralelo/versión for ¿? Threads = cores lógicos=cores físicos		
	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>
8388608	0m0.070s	0m0.091s	0m0.047s	0m0.069s	0m0.085s	0m0.051s
16777216	0m0.133s	0m0.163s	0m0.098s	0m0.132s	0m0.169s	0m0.092s
33554432	0m0.258s	0m0.325s	0m0.186s	0m0.259s	0m0.321s	0m0.192s
67108864	0m0.257s	0m0.329s	0m0.181s	0m0.261s	0m0.314s	0m0.203s



```
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer11] 2021-04-14 miércoles
$ srun -p ac ./script_parallel_atcgrid.sh >> salida.txt --cpus-per-task=1 --hint=nomultithread -n1

real    0m0.006s
user    0m0.000s
sys     0m0.006s

real    0m0.003s
user    0m0.000s
sys     0m0.005s

real    0m0.003s
user    0m0.002s
sys     0m0.003s

real    0m0.004s
user    0m0.003s
sys     0m0.003s

real    0m0.007s
user    0m0.004s
sys     0m0.007s

real    0m0.010s
user    0m0.008s
sys     0m0.008s
```

```
[DanielAlconchelVázquez e1estudiante1@atcgrid:~/bp1/ejer11] 2021-04-14 miércoles
$ srun -p ac ./script_parallel_atcgrid.sh >> salida.txt --cpus-per-task=12 --hint=nomultithread -n1

real    0m0.007s
user    0m0.000s
sys     0m0.005s

real    0m0.003s
user    0m0.000s
sys     0m0.004s

real    0m0.004s
user    0m0.003s
sys     0m0.002s

real    0m0.004s
user    0m0.005s
sys     0m0.001s

real    0m0.007s
user    0m0.005s
sys     0m0.006s

real    0m0.010s
user    0m0.009s
sys     0m0.008s

real    0m0.015s
user    0m0.015s
sys     0m0.011s
```