

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): Javier Victoria Mohamed

Grupo de prácticas y profesor de prácticas:

Fecha de entrega:

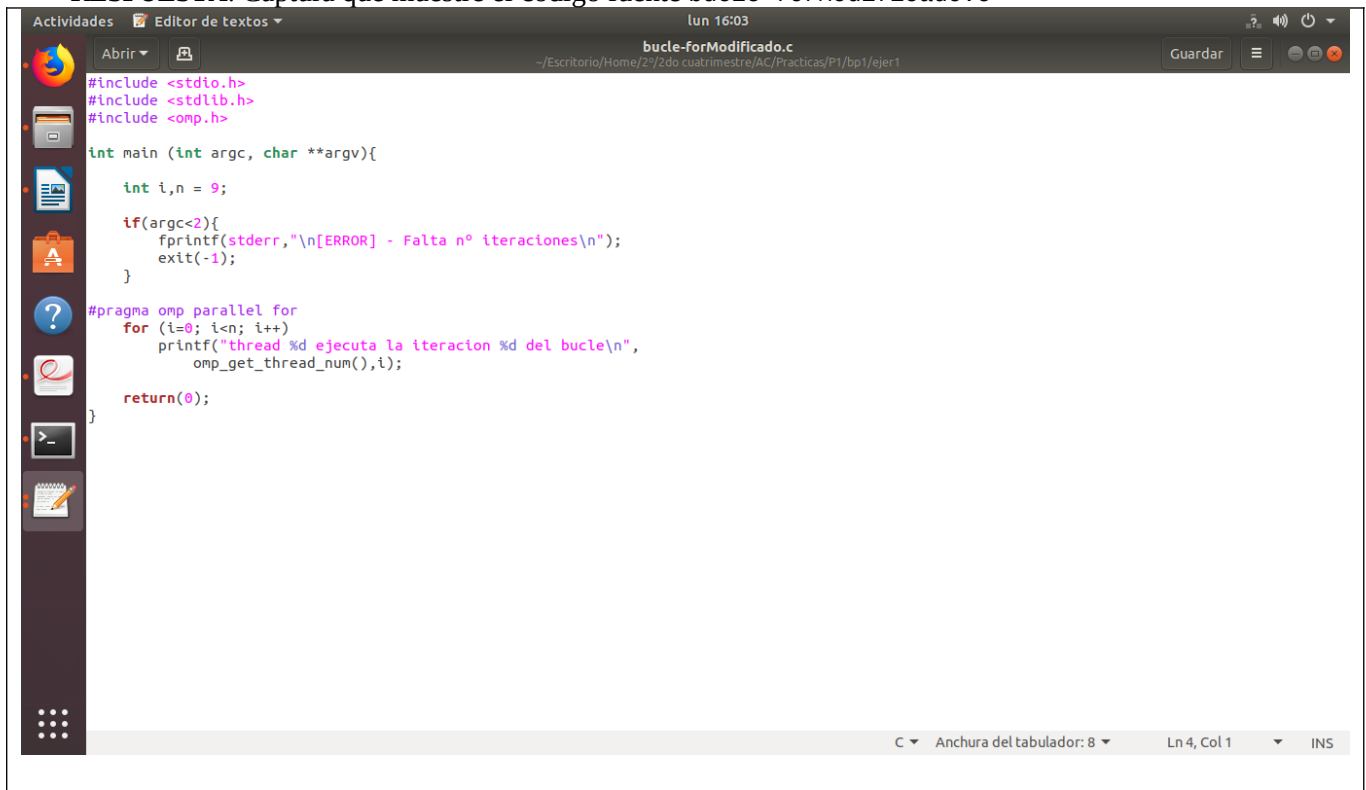
Fecha evaluación en clase:

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 iteraciones\n");
        exit(-1);
    }

    #pragma omp parallel for
    for (i=0; i<n; i++){
        printf("thread %d ejecuta la iteracion %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 seccion la ejecuta el thread %d\n",omp_get_thread_num());
}

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

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

```

Ln 19, Col 13

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

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 inicializacion a: ");
            scanf("%d",&a);
            printf("Single ejecutada por el thread %d\n",omp_get_thread_num());
        }

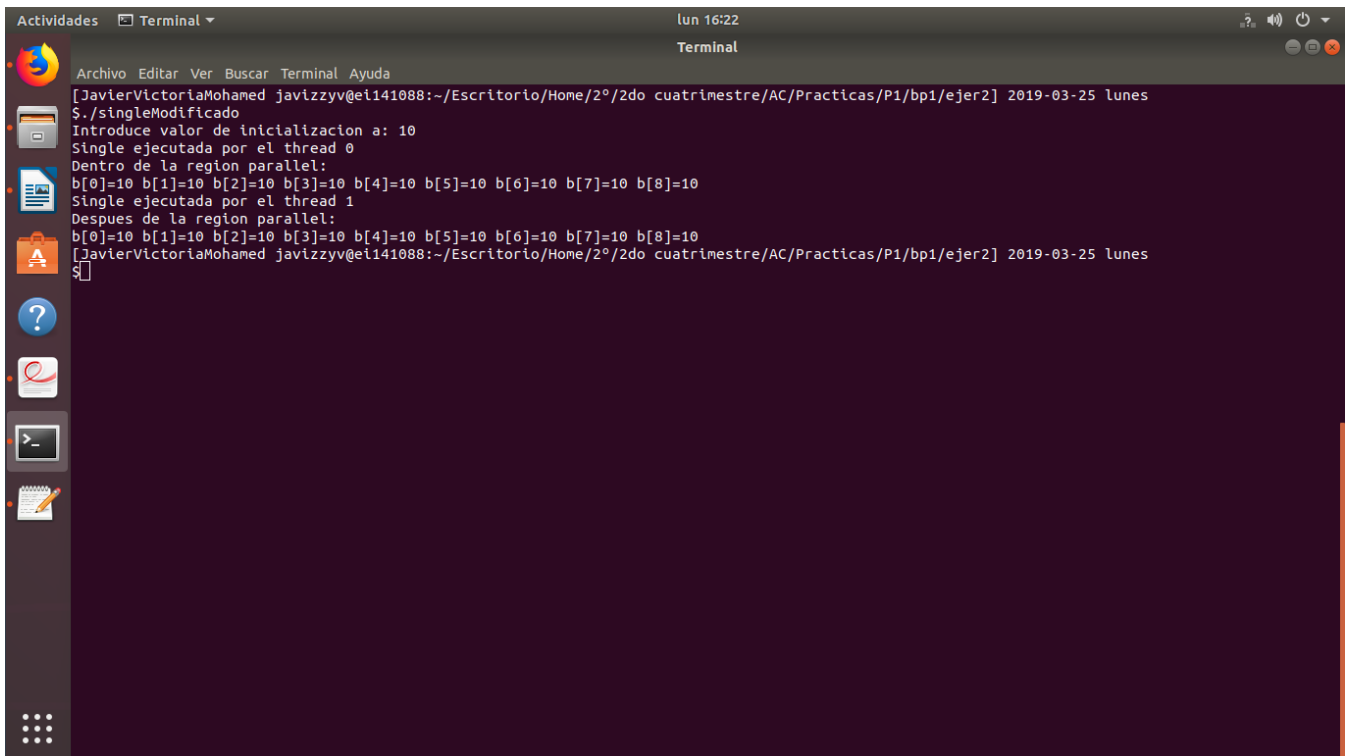
        #pragma omp for
        for(i=0;i<n;i++)
            b[i]=a;

        #pragma omp single
        {
            printf("Dentro de la region parallel:\n");
        }
        #pragma omp single
        {
            for(i=0;i<n;i++) printf("b[%d]=%d\t",i,b[i]);
            printf("\nSingle ejecutada por el thread %d\n",omp_get_thread_num());
        }
    }
    printf("Despues de la region parallel:\n");
    for(i=0;i<n;i++) printf("b[%d]=%d\t",i,b[i]);
    printf("\n");
}

```

Ln 28, Col 23

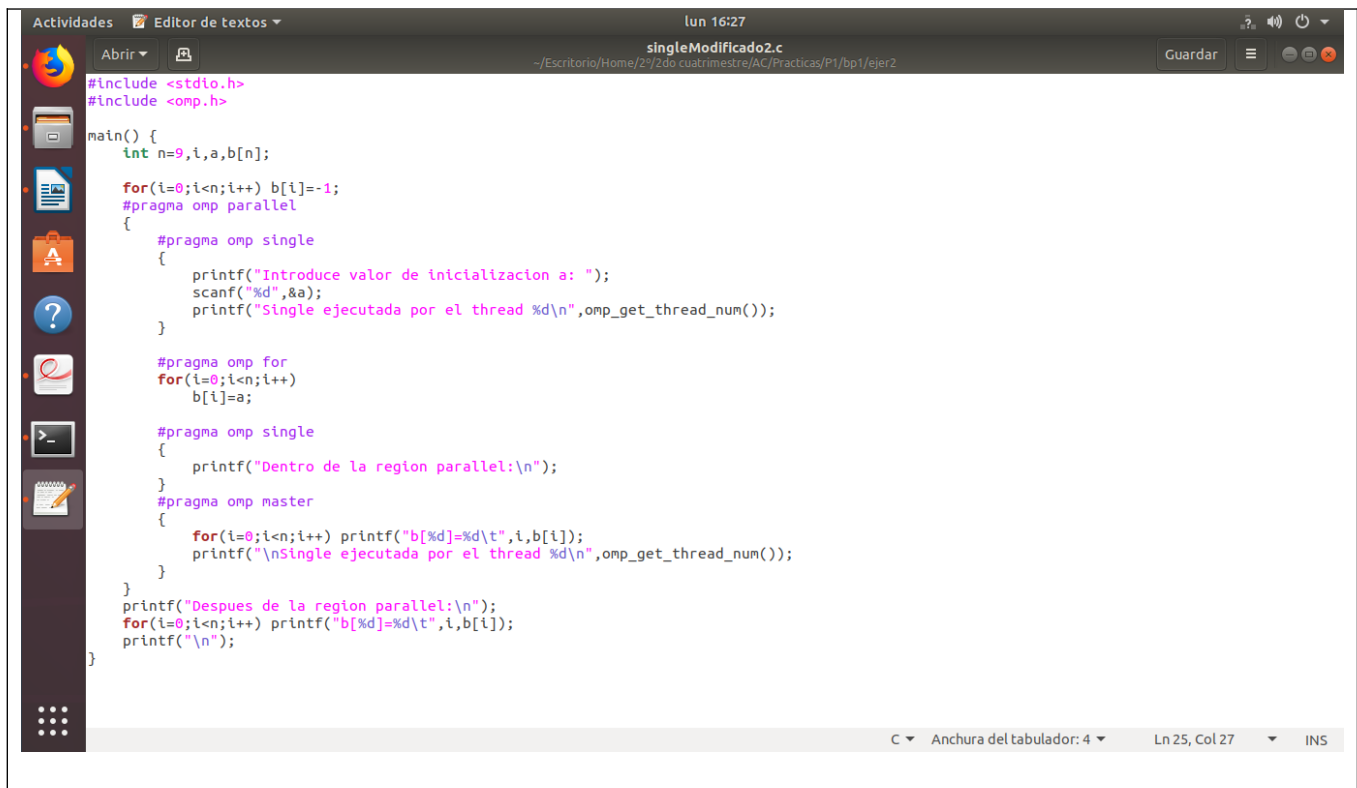
### CAPTURAS DE PANTALLA:



```
Actividades Terminal lun 16:22
Terminal
Archivo Editar Ver Buscar Terminal Ayuda
[JavierVictoriaMohamed javizzyv@ei141088:~/Escritorio/Home/2º/2do cuatrimestre/AC/Practicas/P1/bp1/ejer2] 2019-03-25 Lunes
$ ./singleModificado
Introduce valor de inicializacion a: 10
Single ejecutada por el thread 0
Dentro de la region 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
Single ejecutada por el thread 1
Despues de la region 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
[JavierVictoriaMohamed javizzyv@ei141088:~/Escritorio/Home/2º/2do cuatrimestre/AC/Practicas/P1/bp1/ejer2] 2019-03-25 Lunes
$
```

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

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

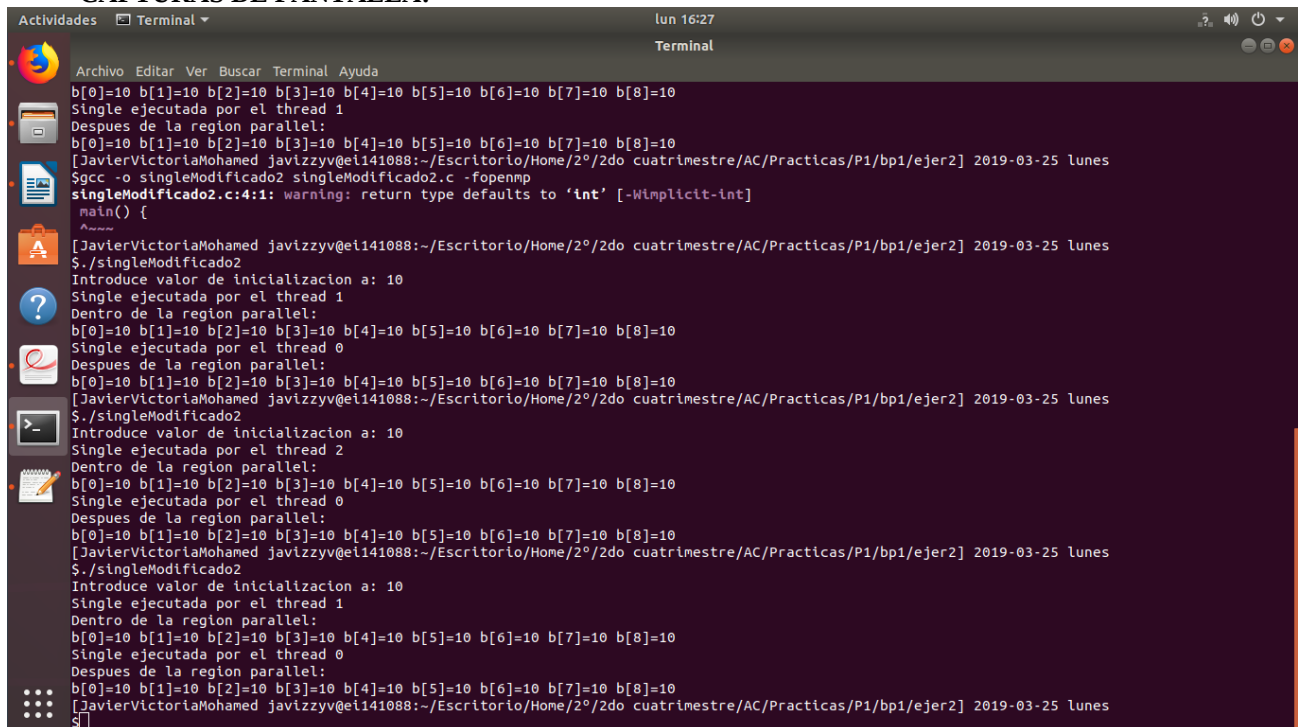
    for(i=0;i<n;i++) b[i]=-1;
    #pragma omp parallel
    {
        #pragma omp single
        {
            printf("Introduce valor de inicializacion a: ");
            scanf("%d",&a);
            printf("Single ejecutada por el thread %d\n",omp_get_thread_num());
        }

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

        #pragma omp single
        {
            printf("Dentro de la region parallel:\n");
        }
        #pragma omp master
        {
            for(i=0;i<n;i++) printf("b[%d]=%d\t",i,b[i]);
            printf("\nSingle ejecutada por el thread %d\n",omp_get_thread_num());
        }
    }
    printf("Despues de la region parallel:\n");
    for(i=0;i<n;i++) printf("b[%d]=%d\t",i,b[i]);
    printf("\n");
}

```

### CAPTURAS DE PANTALLA:



```

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
Single ejecutada por el thread 1
Despues de la region 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
[JavierVictoriaMohamed javizzzyv@ei141088:~/Escritorio/Home/2º/2do cuatrimestre/AC/Practicas/P1/bp1/ejer2] 2019-03-25 lunes
$gcc -o singleModificado2 singleModificado2.c -fopenmp
singleModificado2.c:4:1: warning: return type defaults to 'int' [-Wimplicit-int]
main() {
^
[JavierVictoriaMohamed javizzzyv@ei141088:~/Escritorio/Home/2º/2do cuatrimestre/AC/Practicas/P1/bp1/ejer2] 2019-03-25 lunes
$./singleModificado2
Introduce valor de inicializacion a: 10
Single ejecutada por el thread 1
Dentro de la region 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
Single ejecutada por el thread 0
Despues de la region 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
[JavierVictoriaMohamed javizzzyv@ei141088:~/Escritorio/Home/2º/2do cuatrimestre/AC/Practicas/P1/bp1/ejer2] 2019-03-25 lunes
$./singleModificado2
Introduce valor de inicializacion a: 10
Single ejecutada por el thread 2
Dentro de la region 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
Single ejecutada por el thread 0
Despues de la region 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
[JavierVictoriaMohamed javizzzyv@ei141088:~/Escritorio/Home/2º/2do cuatrimestre/AC/Practicas/P1/bp1/ejer2] 2019-03-25 lunes
$./singleModificado2
Introduce valor de inicializacion a: 10
Single ejecutada por el thread 1
Dentro de la region 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
Single ejecutada por el thread 0
Despues de la region 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
[JavierVictoriaMohamed javizzzyv@ei141088:~/Escritorio/Home/2º/2do cuatrimestre/AC/Practicas/P1/bp1/ejer2] 2019-03-25 lunes
$

```

### RESPUESTA A LA PREGUNTA:

Como vemos tras varias ejecuciones la hebra que ejecuta la sección requerida por el ejercicio es siempre la 0, la master. En cambio en el anterior ejercior cambiaba la hebra que ejecutaba el código pudiendo ser la 1, la 2, etc.

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 al no haber puesto una barrera de sincronización entre las hebras, puede ser que se imprima la suma antes de haber terminado de hacer todos los pasos, si ponemos `barrier` se espera a que todas las hebras hayan acabado para continuar con el código.

## Resto de ejercicios

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 `atcgird`, 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:**

```

C3estudiante19@atcgird.ugr.es's password:
[C3estudiante19@atcgird ejer5]$ PS1="[JavierVictoriaMohamed \u@\h:\w] \D{%F %A}\n$"
[JavierVictoriaMohamed C3estudiante19@atcgird:~/bp1/ejer5] 2019-03-25 lunes
$ls
SumaVectoresC SumaVectoresC.c
[JavierVictoriaMohamed C3estudiante19@atcgird:~/bp1/ejer5] 2019-03-25 lunes
$echo "time ./SumaVectoresC 10000000" | qsub -q ac
13481.atcgird
[JavierVictoriaMohamed C3estudiante19@atcgird:~/bp1/ejer5] 2019-03-25 lunes
$ls
STDIN.e13481 STDIN.o13481 SumaVectoresC SumaVectoresC.c
[JavierVictoriaMohamed C3estudiante19@atcgird:~/bp1/ejer5] 2019-03-25 lunes
$cat STDIN.o13481
[JavierVictoriaMohamed C3estudiante19@atcgird:~/bp1/ejer5] 2019-03-25 lunes
$pwd
/home/C3estudiante19/bp1/ejer5
[JavierVictoriaMohamed C3estudiante19@atcgird:~/bp1/ejer5] 2019-03-25 lunes
$AC
[JavierVictoriaMohamed C3estudiante19@atcgird:~/bp1/ejer5] 2019-03-25 lunes
$echo "time /home/C3estudiante19/bp1/ejer5/SumaVectoresC 10000000" | qsub -q ac
13483.atcgird
[JavierVictoriaMohamed C3estudiante19@atcgird:~/bp1/ejer5] 2019-03-25 lunes
$cat STDIN.o13483
Tamaño Vectores:10000000 (4 B)
Tiempo:0.046386936 / 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) /
[JavierVictoriaMohamed C3estudiante19@atcgird:~/bp1/ejer5] 2019-03-25 lunes
$cat STDIN.e13483
real    0m0.116s
user    0m0.064s
sys     0m0.050s
[JavierVictoriaMohamed C3estudiante19@atcgird:~/bp1/ejer5] 2019-03-25 lunes
$

```

Es menor porque el programa no se ejecuta el mismo tiempo en modo usuario que en modo kernel(sistema), por eso mismo el tiempo entre ellos es diferente y menor al tiempo de ejecución total, debido a que una porción del tiempo se ejecuta en modo kernel y otra en modo usuario.

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 `atcgird` 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. Incorpore **el código ensamblador de la parte de la suma de vectores** 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):

```

Actividades Terminal lun 17:17
Terminal
Archivo Editar Ver Buscar Terminal Ayuda
$time ./SumaVectoresC 10000000
Tamaño Vectores:10000000 (4 B)
Tiempo:0.042329112 / 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 0m0.117s
user 0m0.069s
sys 0m0.048s
[JavierVictoriaMohamed javizyv@ei141088:~/Escritorio/Home/2º/2do cuatrimestre/AC/Practicas/P1/bp1/ejer5] 2019-03-25 lunes
$ls
SumaVectoresC SumaVectoresC.c
[JavierVictoriaMohamed javizyv@ei141088:~/Escritorio/Home/2º/2do cuatrimestre/AC/Practicas/P1/bp1/ejer5] 2019-03-25 lunes
$gcc -S SumaVectoresC.c -o SumaVectoresC -lrt
SumaVectoresC.c: In function 'main':
SumaVectoresC.c:45:32: warning: format '%u' expects argument of type 'unsigned int', but argument 3 has type 'long unsigned int' [-Wformat=]
printf("Tamaño Vectores:%u (%u B)\n",N, sizeof(unsigned int));
~^
~^
~^
$ls
SumaVectoresC SumaVectoresC.c SumaVectoresC.s

```

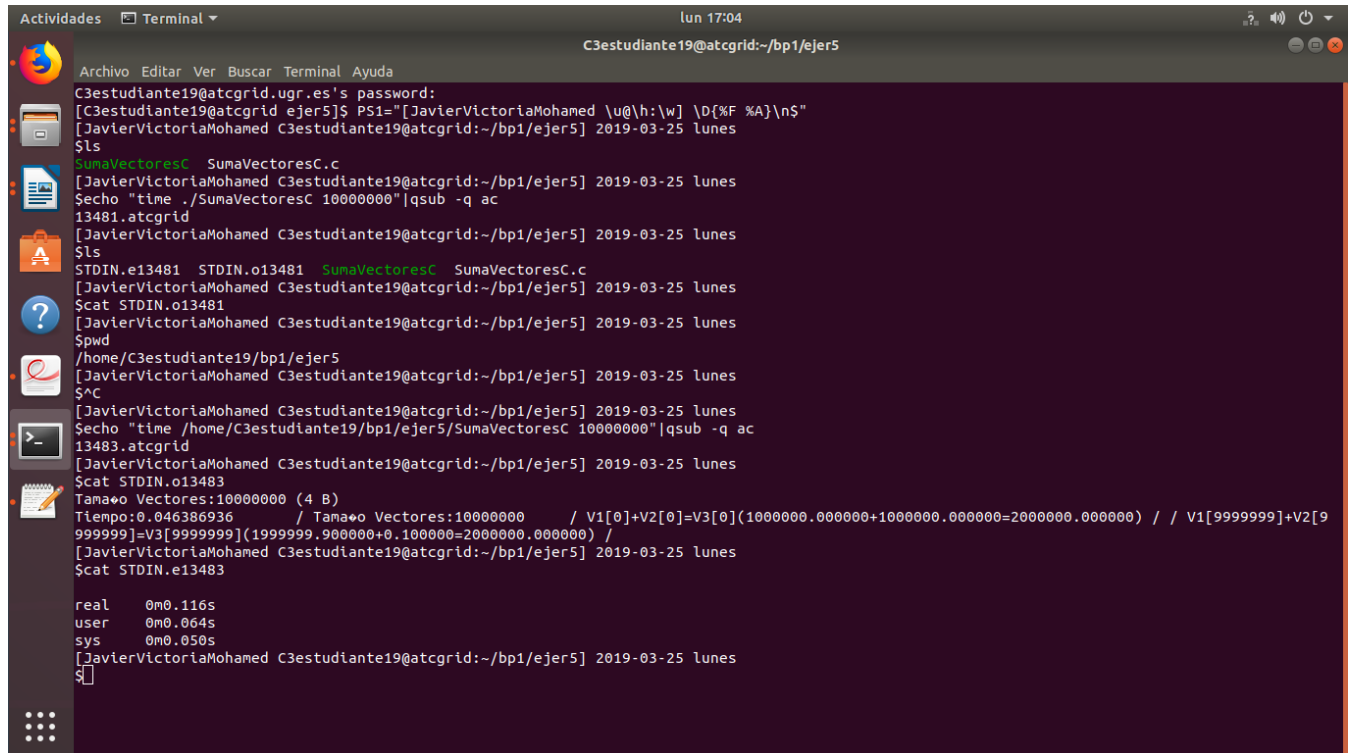
10:

```

Actividades Terminal lun 17:20
C3estudiante19@atcgrid:~/bp1/ejer5
[JavierVictoriaMohamed C3estudiante19@atcgrid:~/bp1/ejer5] 2019-03-25 lunes
$cat STDIN.o13504
Tamaño Vectores:10 (4 B)
Tiempo:0.000408552 / 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) /
[JavierVictoriaMohamed C3estudiante19@atcgrid:~/bp1/ejer5] 2019-03-25 lunes
$cat STDIN.e13504
real 0m0.005s
user 0m0.001s
sys 0m0.002s
[JavierVictoriaMohamed C3estudiante19@atcgrid:~/bp1/ejer5] 2019-03-25 lunes
$

```

10000000:



```

Actividades Terminal lun 17:04
C3estudiante19@atcgrid:~/bp1/ejer5

C3estudiante19@atcgrid.ugr.es's password:
[C3estudiante19@atcgrid ejer5]$ PS1="[JavierVictoriaMohamed \u@h:\w] \D{%F %A}\n$"
[JavierVictoriaMohamed C3estudiante19@atcgrid:~/bp1/ejer5] 2019-03-25 Lunes
$ls
SumaVectoresC SumaVectoresC.c
[JavierVictoriaMohamed C3estudiante19@atcgrid:~/bp1/ejer5] 2019-03-25 Lunes
$echo "time ./SumaVectoresC 10000000"|qsub -q ac
13481.atcgrid
[JavierVictoriaMohamed C3estudiante19@atcgrid:~/bp1/ejer5] 2019-03-25 Lunes
$ls
STDIN.e13481 STDIN.o13481 SumaVectoresC SumaVectoresC.c
[JavierVictoriaMohamed C3estudiante19@atcgrid:~/bp1/ejer5] 2019-03-25 Lunes
$cat STDIN.o13481
[JavierVictoriaMohamed C3estudiante19@atcgrid:~/bp1/ejer5] 2019-03-25 Lunes
$pwd
/home/C3estudiante19/bp1/ejer5
[JavierVictoriaMohamed C3estudiante19@atcgrid:~/bp1/ejer5] 2019-03-25 Lunes
$^C
[JavierVictoriaMohamed C3estudiante19@atcgrid:~/bp1/ejer5] 2019-03-25 Lunes
$echo "time /home/C3estudiante19/bp1/ejer5/SumaVectoresC 10000000"|qsub -q ac
13483.atcgrid
[JavierVictoriaMohamed C3estudiante19@atcgrid:~/bp1/ejer5] 2019-03-25 Lunes
$cat STDIN.o13483
Tamaño Vectores:10000000 (4 B)
Tiempo:0.046386936 / 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) /
[JavierVictoriaMohamed C3estudiante19@atcgrid:~/bp1/ejer5] 2019-03-25 Lunes
$cat STDIN.e13483
real    0m0.116s
user    0m0.064s
sys     0m0.050s
[JavierVictoriaMohamed C3estudiante19@atcgrid:~/bp1/ejer5] 2019-03-25 Lunes
$

```

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

Obtenemos los tiempos de CPU de la ejecución para tamaño 10 y 10000000, 6 instrucciones de bucle (desde L10 hasta clock\_gettime()) y 1 sola instrucción en coma flotante (addsd). Finalmente multiplicamos por  $10^6$  el TCPU para expresar el resultado en MFLOPS y MIPS.

-MIPS: Para tamaño 10:  $(6 \cdot 10) / (0.000408552 \cdot 10^6) = 0.14$  MIPS,

Para 10000000:  $(6 \cdot 10000000) / (0.046386936 \cdot 10^6) = 1293$  MIPS

-MFLOPS: Para tamaño 10:  $(1 \cdot 10) / (0.000408552 \cdot 10^6) = 0.024$  MFLOPS,

Para 10000000:  $(1 \cdot 10000000) / (0.046386936 \cdot 10^6) = 215$  MFLOPS

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

```

SumaVectoresC.c
~/.Escritorio/Home/2º/2do cuatrimestre/AC/Practicas/P1/bp1/ejer5
Guardar

SumaVectoresC.c x apuntes.txt x formato_terminal x SumaVectoresC.s x

    cml    %eax, -60(%rbp)
    ja     .L9
    leaq   -48(%rbp), %rax
    movq   %rax, %rsi
    movl   $0, %edi
    call   clock_gettime@PLT
    movl   $0, -64(%rbp)
    jmp    .L10
.L11:
    movl   -64(%rbp), %eax
    cltq
    leaq   0(,%rax,8), %rdx
    leaq   v1(%rip), %rax
    movsd  (%rdx,%rax), %xmm1
    movl   -64(%rbp), %eax
    cltq
    leaq   0(,%rax,8), %rdx
    leaq   v2(%rip), %rax
    movsd  (%rdx,%rax), %xmm0
    addsd  %xmm1, %xmm0
    movl   -64(%rbp), %eax
    cltq
    leaq   0(,%rax,8), %rdx
    leaq   v3(%rip), %rax
    movsd  %xmm0, (%rdx,%rax)
    addl   $1, -64(%rbp)
.L10:
    movl   -64(%rbp), %eax
    cml    %eax, -60(%rbp)
    ja     .L11
    leaq   -32(%rbp), %rax
    movq   %rax, %rsi
    movl   $0, %edi
    call   clock_gettime@PLT
    movq   -32(%rbp), %rdx
    movl   -48(%rbp), %rax

```

Texto plano Anchura del tabulador: 8 Ln 148, Col 17 INS

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



```

#endif
#ifdef VECTOR_DYNAMIC
double *v1, *v2, *v3;
v1 = (double*) malloc(N*sizeof(double)); // malloc necesita el tamaño en bytes
v2 = (double*) malloc(N*sizeof(double));
v3 = (double*) malloc(N*sizeof(double));
if ((v1 == NULL) || (v2 == NULL) || (v3 == NULL)) {
    printf("No hay suficiente espacio para los vectores \n");
    exit(-2);
}
#endif

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

double fin = omp_get_wtime();
double ncgt = fin - inicio;
//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\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]);

#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

```

Usando vectores dinámicos, debido a que son más eficientes.

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```

$gcc -O2 SumaVectoresC.c -o SumaVectoresC -lrt -fopenmp
SumaVectoresC.c: In function 'main':
SumaVectoresC.c:44:32: warning: format '%u' expects argument of type 'unsigned int', but argument 3 has type 'long unsigned int' [-Wformat=]
    printf("Tamaño Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                               ^~
$./SumaVectoresC 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000005622 / 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) /
$./SumaVectoresC 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000004508 / 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) /
$

```

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



```

//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<3*N/4; i++){
        v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
    }

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

double inicio = omp_get_wtime();
//Calcular suma de vectores
#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<3*N/4; i++){
        v3[i] = v1[i] + v2[i];
    }

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

double fin = omp_get_wtime();
printf("Tiempo de ejecución: %f\n", fin - inicio);
for(i=0; i<N; i++){
    printf("v3[%d] = %f\n", i, v3[i]);
}
  
```

Usando vectores dinámicos, debido a que son más eficientes.

**(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

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

```
mié 11:19
Terminal

Archivo Editar Ver Buscar Terminal Ayuda

[JavierVictoriaMohamed javizzyv@ei141054:~/Escritorio/Home/2º/2do cuatrimestre/AC/Practicas/P1/bp1/ejer8] 2019-03-27 miércoles
$gcc -O2 SumaVectoresC.c -o SumaVectoresC -lrt -fopenmp
SumaVectoresC.c: In function 'main':
SumaVectoresC.c:44:32: warning: format '%u' expects argument of type 'unsigned int', but argument 3 has type 'long unsigned int' [-Wformat=]
    printf("Tamaño Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                               ^~
                               |
                               int
[JavierVictoriaMohamed javizzyv@ei141054:~/Escritorio/Home/2º/2do cuatrimestre/AC/Practicas/P1/bp1/ejer8] 2019-03-27 miércoles
$./SumaVectoresC 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000006620 / 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) /
[JavierVictoriaMohamed javizzyv@ei141054:~/Escritorio/Home/2º/2do cuatrimestre/AC/Practicas/P1/bp1/ejer8] 2019-03-27 miércoles
$./SumaVectoresC 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000006604 / 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) /
[JavierVictoriaMohamed javizzyv@ei141054:~/Escritorio/Home/2º/2do cuatrimestre/AC/Practicas/P1/bp1/ejer8] 2019-03-27 miércoles
$
```

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.

**RESPUESTA:**

Para el ejercicio 7: Se utilizarán tantos cores y hebras como haya disponible en ese momento al no haber definido cuantos queremos utilizar y al tratarse de ‘parallel for’.

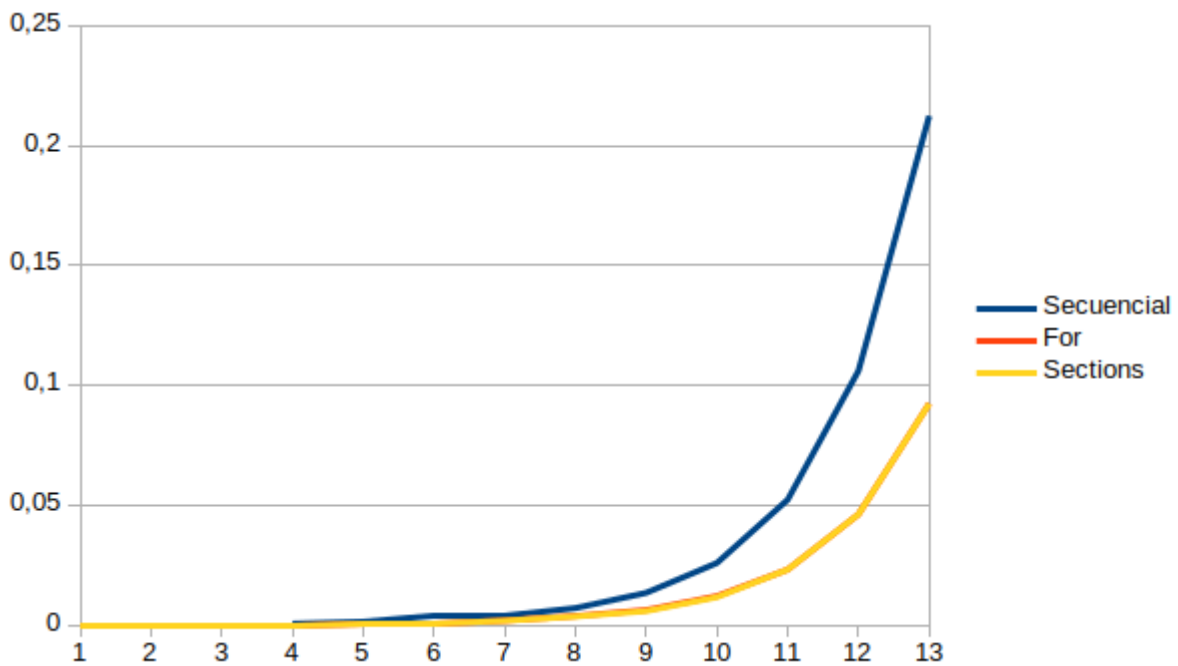
Para el ejercicio 8: Se utilizarán tantos cores y hebras como secciones hayamos definido, en este caso 4 que es el número de cores de el PC donde se realiza la práctica.

10. Rellenar una tabla como la Tabla 2 para atcgrid y otra para su PC con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. 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.

**RESPUESTA:**

Para PC:

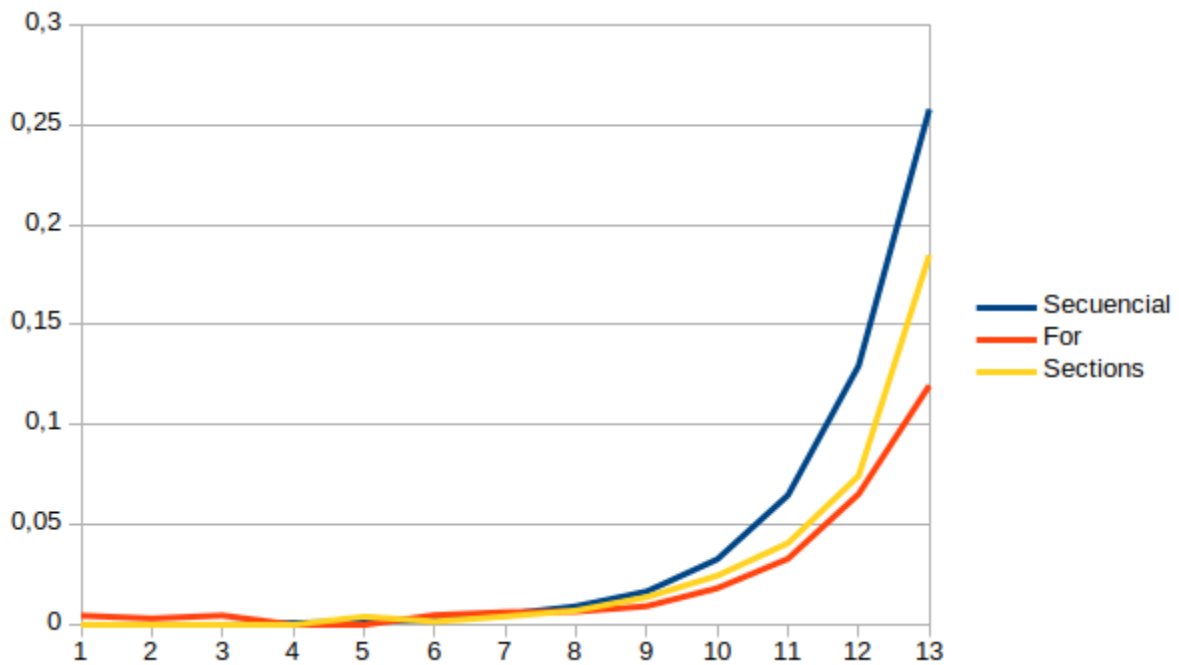
Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 4threads/cores	T. paralelo (versión sections) 4threads/cores
16384	0,000088390	0,000028601	0,000015862
32768	0,000178557	0,000044503	0,000053283
65536	0,000348237	0,000103729	0,000126895
131072	0,000709536	0,000213205	0,000219335
262144	0,001496853	0,000437111	0,000441751
524288	0,003890852	0,001070135	0,001022524
1048576	0,003971576	0,002001229	0,001934412
2097152	0,007234995	0,003891867	0,003716214
4194304	0,013581290	0,006402119	0,005968398
8388608	0,026023087	0,012222454	0,011776949
16777216	0,052400408	0,023270717	0,023285603
33554432	0,106079826	0,046260032	0,046289990
67108864	0,212375183	0,092446150	0,092361329



El 'Sections' sale casi superpuesto con el 'For' debido a que el tiempo de ejecución es casi similar en ambos casos debido a que en nuestro PC no podemos aprovechar del todo el paralelismo en el for ya que solo tiene 4 núcleos.

Para atcgrid:

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 24threads/cores	T. paralelo (versión sections) 4threads/cores
16384	0,000109471	0,004568136	0,000051823
32768	0,000215709	0,003115010	0,000120523
65536	0,000428290	0,004652941	0,000217434
131072	0,000870498	0,000142103	0,000340376
262144	0,001741344	0,000318558	0,004049385
524288	0,002618037	0,004785522	0,001633905
1048576	0,005138861	0,006267602	0,004139981
2097152	0,009229375	0,006560748	0,007068706
4194304	0,016626030	0,009259224	0,013811568
8388608	0,032709705	0,018284550	0,024546259
16777216	0,064797186	0,033001063	0,040807840
33554432	0,129533829	0,065352813	0,074504804
67108864	0,257759352	0,119642209	0,184836418



**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 del computador que como máximo puede aprovechar el código.

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) ¿?threads/cores	T. paralelo (versión sections) ¿?threads/cores
16384			
32768			
65536			
131072			
262144			
524288			
1048576			
2097152			
4194304			
8388608			
16777216			
33554432			
67108864			

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/cores que usan los códigos. ¿El tiempo de CPU que se obtiene es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

**RESPUESTA:**

**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/core			Tiempo paralelo/versión for 24 Threads/cores		
	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>
65536	0m0.003s	0m0.002s	0m0.001s	0m0.003s	0m0.026s	0m0.003s
131072	0m0.004s	0m0.002s	0m0.002s	0m0.006s	0m0.048s	0m0.034s
262144	0m0.007s	0m0.002s	0m0.005s	0m0.006s	0m0.084s	0m0.016s
524288	0m0.010s	0m0.003s	0m0.007s	0m0.016s	0m0.228s	0m0.071s
1048576	0m0.016s	0m0.006s	0m0.010s	0m0.017s	0m0.168s	0m0.165s
2097152	0m0.027s	0m0.015s	0m0.012s	0m0.023s	0m0.261s	0m0.217s
4194304	0m0.045s	0m0.023s	0m0.022s	0m0.025s	0m0.344s	0m0.201s
8388608	0m0.081s	0m0.036s	0m0.045s	0m0.044s	0m0.623s	0m0.306s
16777216	0m0.158s	0m0.096s	0m0.061s	0m0.076s	0m1.179s	0m0.543s
33554432	0m0.313s	0m0.159s	0m0.154s	0m0.142s	0m2.269s	0m0.904s
67108864	0m0.619s	0m0.336s	0m0.282s	0m0.296s	0m4.490s	0m2.037s

En paralelo es mayor y en secuencial es menor, debido a que al haber más CPUs que trabajan en paralelo, el tiempo de todas se suma y da este resultado.