

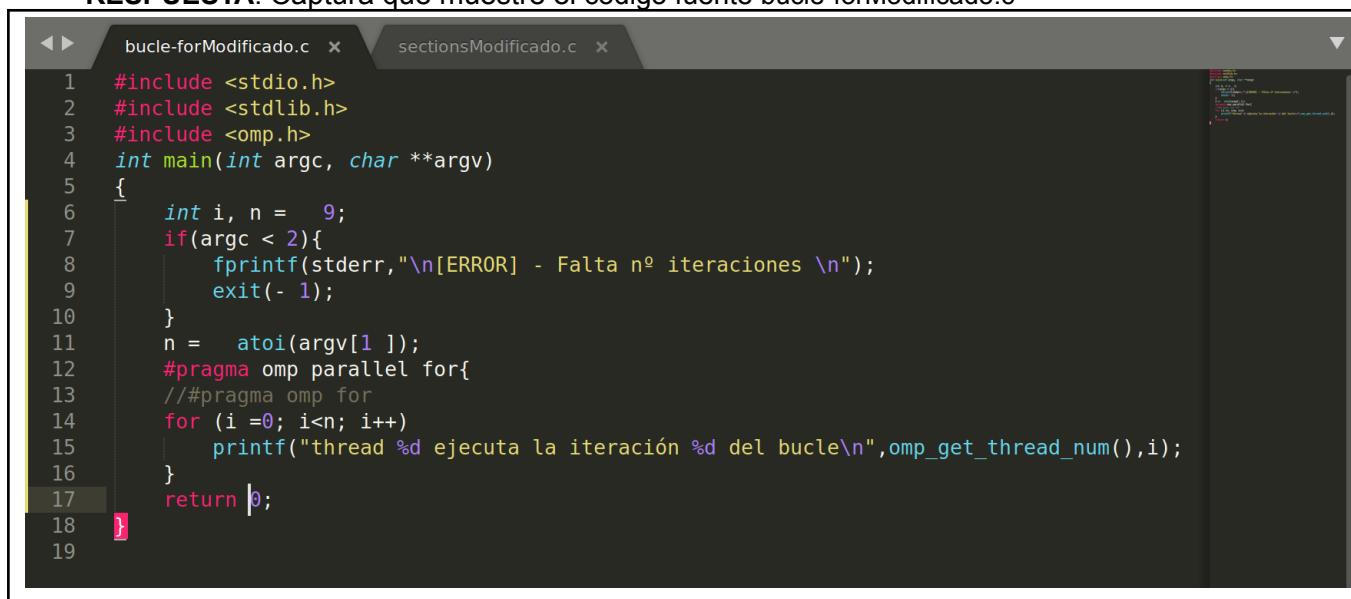
<p>2º curso / 2º cuatr.</p> <p>Grado Ing. Inform.</p> <p>Doble Grado Ing. Inform. y Mat.</p>	<h2>Arquitectura de Computadores (AC)</h2> <h3>Cuaderno de prácticas.</h3> <h3>Bloque Práctico 1. Programación paralela I: Directivas OpenMP</h3> <p>Estudiante (nombre y apellidos): Amador Carmona Méndez</p> <p>Grupo de prácticas y profesor de prácticas: B1 Jesús González</p> <p>Fecha de entrega: 18 abril 2021</p> <p>Fecha evaluación en clase:</p>
--	---

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



```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <omp.h>
4  int main(int argc, char **argv)
5  {
6      int i, n = 9;
7      if(argc < 2){
8          fprintf(stderr, "\n[ERROR] - Falta nº iteraciones \n");
9          exit(-1);
10     }
11     n = atoi(argv[1]);
12     #pragma omp parallel for{
13         //#pragma omp for
14         for (i = 0; i < n; i++)
15             printf("thread %d ejecuta la iteración %d del bucle\n", omp_get_thread_num(), i);
16     }
17     return 0;
18 }
19

```

RESPUESTA: Captura que muestre el código fuente sectionsModificado.c

```

1  #include <stdio.h>
2  #include <omp.h>
3  void funcA()
4  {
5      printf("En funcA: esta sección la ejecuta el thread%d\n",omp_get_thread_num());
6  }
7  void funcB()
8  {
9      printf("En funcB: esta sección la ejecuta el thread%d\n",omp_get_thread_num());
10 }
11 main()
12 {
13     #pragma omp parallel sections
14     {
15         {
16             #pragma omp section
17             (void) funcA();
18             #pragma omp section
19             (void) funcB();
20         }
21     }
22     return 0;
23 }

```

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

```

[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer2] 2021-04-18 domingo
$cat 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 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 hebra:%d\t",i,b[i],omp_get_thread_num());
            printf("\n");
        }
    }
    return 0;
}
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer2] 2021-04-18 domingo
$

```

CAPTURAS DE PANTALLA:

La directiva `single` hace que sea una única hebra la que ejecute la sección de código a la que se refiere, y como hay dos `single`, son dos hebras las que se ven cuando se ejecuta el programa.

```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer2] 2021-04-18 domingo
$gcc -O2 -fopenmp -o single singleModificado.c
singleModificado.c:3:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
^
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer2] 2021-04-18 domingo
$./single
Introduce valor de inicialización a: 100
Single ejecutada por el thread 2
Después de la región parallel:
b[0] = 100 hebra:2    b[1] = 100 hebra:2    b[2] = 100 hebra:2    b[3] = 100 hebra:2    b[4] = 100 hebra:2    b[5] = 100 hebra:2 b
[6] = 100 hebra:2    b[7] = 100 hebra:2    b[8] = 100 hebra:2
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer2] 2021-04-18 domingo
$./single
Introduce valor de inicialización a: 500
Single ejecutada por el thread 2
Después de la región parallel:
b[0] = 500 hebra:3    b[1] = 500 hebra:3    b[2] = 500 hebra:3    b[3] = 500 hebra:3    b[4] = 500 hebra:3    b[5] = 500 hebra:3 b
[6] = 500 hebra:3    b[7] = 500 hebra:3    b[8] = 500 hebra:3
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer2] 2021-04-18 domingo
$./single
Introduce valor de inicialización a: 1000
Single ejecutada por el thread 2
Después de la región parallel:
b[0] = 1000 hebra:3    b[1] = 1000 hebra:3    b[2] = 1000 hebra:3    b[3] = 1000 hebra:3    b[4] = 1000 hebra:3    b[5] = 1000 hebra:3b
[6] = 1000 hebra:3    b[7] = 1000 hebra:3    b[8] = 1000 hebra:3
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer2] 2021-04-18 domingo
$./single
Introduce valor de inicialización a: 10000
Single ejecutada por el thread 1
Después de la región parallel:
b[0] = 10000 hebra:1    b[1] = 10000 hebra:1    b[2] = 10000 hebra:1    b[3] = 10000 hebra:1    b[4] = 10000 hebra:1    b[5] = 10000 hebra:1
b[6] = 10000 hebra:1    b[7] = 10000 hebra:1    b[8] = 10000 hebra:1
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer2] 2021-04-18 domingo
$
```

- 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`

```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer3] 2021-04-18 domingo
$cat singleModificado2.c
cat: singleModificado2.c: No existe el archivo o el directorio
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer3] 2021-04-18 domingo
$cat 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 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 hebra:%d\t",i,b[i],omp_get_thread_num());
            printf("\n");
        }
    }

    return 0;
}
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer3] 2021-04-18 domingo
$
```

CAPTURAS DE PANTALLA:

```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer3] 2021-04-18 domingo
$gcc -O2 -fopenmp -o single singleModificado2.c
singleModificado2.c:3:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
^~~~~~
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer3] 2021-04-18 domingo
$./single
Introduce valor de inicialización a: 100
Single ejecutada por el thread 1
Después de la región parallel:
b[0] = 100 hebra:0    b[1] = 100 hebra:0    b[2] = 100 hebra:0    b[3] = 100 hebra:0    b[4] = 100 hebra:0    b[5] = 100 hebra:0 b
[6] = 100 hebra:0    b[7] = 100 hebra:0    b[8] = 100 hebra:0
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer3] 2021-04-18 domingo
$./single
Introduce valor de inicialización a: 500
Single ejecutada por el thread 0
Después de la región parallel:
b[0] = 500 hebra:0    b[1] = 500 hebra:0    b[2] = 500 hebra:0    b[3] = 500 hebra:0    b[4] = 500 hebra:0    b[5] = 500 hebra:0 b
[6] = 500 hebra:0    b[7] = 500 hebra:0    b[8] = 500 hebra:0
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer3] 2021-04-18 domingo
$
```

RESPUESTA A LA PREGUNTA: La parte de la directiva master que antes era single ahora se ejecuta siempre en la hebra 0 que es la master y antes se ejecutaba en una sola hebra pero iba variando la 1,2 o 3. con la directiva master te aseguras que se ejecute con 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: si quitamos la directiva barrier, el programa no espera a que se ejecuten todas las hebras por ello cuando una queda libre imprime sin que la suma quede realizada entera y por ello da error. La directiva barrier nos da el control sobre las hebras cuando queremos que todas las hebras esperen a que las demas se hayan ejecutado.

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:

```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer5] 2021-04-18 domingo
$gcc -O2 -fopenmp -o suma sumaVectoresC.c
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer5] 2021-04-18 domingo
$time ./suma 10000000
Tiempo:0.057708177 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](9.049965+11.905485=20.955450) / / V1[9999999]+V2[9999999]=V3[9999999](0.230008+0.947594=1.177602) /

real    0m0.721s
user    0m0.609s
sys     0m0.112s
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer5] 2021-04-18 domingo
$
```

RESPUESTA: en mi caso son iguales la suma de los tiempos de usuario y del sistema (sys y user en la imagen) son iguales a la de tiempo real (real en la imagen) , y siempre será igual o menor que la real. Esto es así ya que el tiempo real es el tiempo transcurrido en la ejecución, sys el tiempo que se ha ejecutado en el núcleo del SO y user el tiempo ejecutado fuera del núcleo del SO.

6. Generar el código ensamblador a partir del programa secuencial C del Listado 1 para **vectores globales** (para generar el código ensamblador tiene que compilar usando `-S` en lugar de `-o`). Utilice el fichero con el código fuente ensamblador generado y el fichero ejecutable generado en el ejercicio 5 para obtener para atcgrid los MIPS (*Millions of Instructions Per Second*) y los MFLOPS (*Millions of Floating-point Per Second*) del código que obtiene la suma de vectores (código entre las funciones `clock_gettime()`); el cálculo se debe hacer para 10 y 10000000 componentes en los vectores (consulte la Lección 3/Tema1 AC). Razonar cómo se han obtenido los valores que se necesitan para calcular los MIPS y MFLOPS. Incorporar **el código ensamblador de la parte de la suma de vectores** (no de todo el programa) en el cuaderno.

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

```

[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer6] 2021-04-18 domingo
$ls
sumaVectoresC.c
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer6] 2021-04-18 domingo
$ls
sumaVectoresC.c
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer6] 2021-04-18 domingo
$gcc -O2 -fopenmp -S sumaVectoresC.c
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer6] 2021-04-18 domingo
$ls
sumaVectoresC.c sumaVectoresC.s
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer6] 2021-04-18 domingo
$gcc -O2 -fopenmp -o suma sumaVectoresC.c
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer6] 2021-04-18 domingo
$ls
suma sumaVectoresC.c sumaVectoresC.s
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer6] 2021-04-18 domingo
$
suma sumaVectoresC.c sumaVectoresC.s
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer6] 2021-04-18 domingo
$./suma 100000000
Tamaño Vectores:100000000 / V1[0]+V2[0]=V3[0](1.030366+0.845644=1.876010) / / V1[9999999]+V2[9999999]=V3[99999
99](0.357067+1.436034=1.793100) /
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer6] 2021-04-18 domingo
$./suma 10
Tamaño Vectores:10 / V1[0]+V2[0]=V3[0](0.632577+1.564922=2.197499) / / V1[9]+V2[9]=V3[9](0.373014+0.001633=0.37
4647) /
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer6] 2021-04-18 domingo
$

```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Para los MIPS serían 6 instrucciones para el bucle y 3 de inicio entre los tiempos. Y los MFLOPS 7 operaciones en coma flotante.

$$\text{MIPS} = 3 + (6 * 100000000) / 0,06887579 * 10^6 = 8,7 * 10^{14}$$

$$\text{MFLOPS} = 7 * 100000000 / 0,06887579 * 10^6 = 1,01 * 10^{15}$$

$$\text{MIPS} = 3 + (6 * 10) / 0.000000229 * 10^6 = 2,75 * 10^{14}$$

$$\text{MFLOPS} = 7 * 10 / 0.000000229 * 10^6 = 3.05 * 10^{14}$$

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

```

        movl    4(%rsp), %eax
        leaq    8(,%rax,8), %rdx
        xorl    %eax, %eax
        .p2align 4,,10
        .p2align 3
.L10:
        movsd   0(%rbp,%rax), %xmm0
        addsd   (%r12,%rax), %xmm0
        movsd   %xmm0, 0(%r13,%rax)
        addq    $8, %rax
        cmpq    %rax, %rdx
        jne     .L10
        leaq    32(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
        movq    40(%rsp), %rax

```

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`


```

61     exit(-2);
62 }
63 #endif
64
65 //Inicializar vectores
66 #pragma omp parallel for
67 for (i = 0; i < N; i++)
68 {
69     v1[i] = N * 0.1 + i * 0.1;
70     v2[i] = N * 0.1 - i * 0.1;
71 }
72
73
74 double time_start=omp_get_wtime();
75 //Calcular suma de vectores
76 #pragma omp parallel for
77 for(i=0; i<N; i++)
78     v3[i] = v1[i] + v2[i];
79
80 ncgt=(double)((omp_get_wtime()-time_start)*1000.0);
81
82 //Imprimir resultado de la suma y el tiempo de ejecución
83 if (N<10) {
84     printf("Tiempo:%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
85     for(i=0; i<N; i++)
86         printf("/ V1[%d]+V2[%d]=V3[%d] (%8.6f+%8.6f=%8.6f) /\n",
87             i, i, i, v1[i], v2[i], v3[i]);

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```

[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer7] 2021-04-18 domingo
$gcc -O2 -fopenmp -o suma sumaVectoresC.c
sumaVectoresC.c: In function 'main':
sumaVectoresC.c:46:35: 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));
                                ~^
                                %lu
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer7] 2021-04-18 domingo
$./suma 11
Tamaño Vectores:11 (4 B)
Tiempo:0.006118000 / 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) /
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer7] 2021-04-18 domingo
$./suma 8
Tamaño Vectores:8 (4 B)
Tiempo:4.030010001 / 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) /
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer7] 2021-04-18 domingo
$

```

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

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

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)**CAPTURAS DE PANTALLA (compilación y ejecución para N=8 y N=11):**

```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer8] 2021-04-18 domingo
$gcc -O2 -fopenmp -o suma sumaVectoresC.c
sumaVectoresC.c: In function 'main':
sumaVectoresC.c:46:35: 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));
                                ~^
                                %lu
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer8] 2021-04-18 domingo
$./suma 8
Tamaño Vectores:8 (4 B)
Tiempo:0.039981001 / 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) /
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer8] 2021-04-18 domingo
$./suma 11
Tamaño Vectores:11 (4 B)
Tiempo:0.016607002 / 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) /
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP1/ejer8] 2021-04-18 domingo
$
```

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 se podrían utilizar tantas hebras como tenga el ordenador y en el 8 se utilizaran solo 4 ya que solo hemos puesto 4 section en cada parte paralelizada.

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. 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

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

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

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión for) ¿?threads = cores lógicos = cores físicos	T. paralelo (versión sections) ¿?threads = cores lógicos = cores físicos
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 (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

(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-</i>	

<i>sys</i>	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>
8388608			
16777216			
33554432			
67108864			