

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): Guillermo Sandoval Schmidt

Grupo de prácticas y profesor de prácticas: C3

Fecha de entrega:

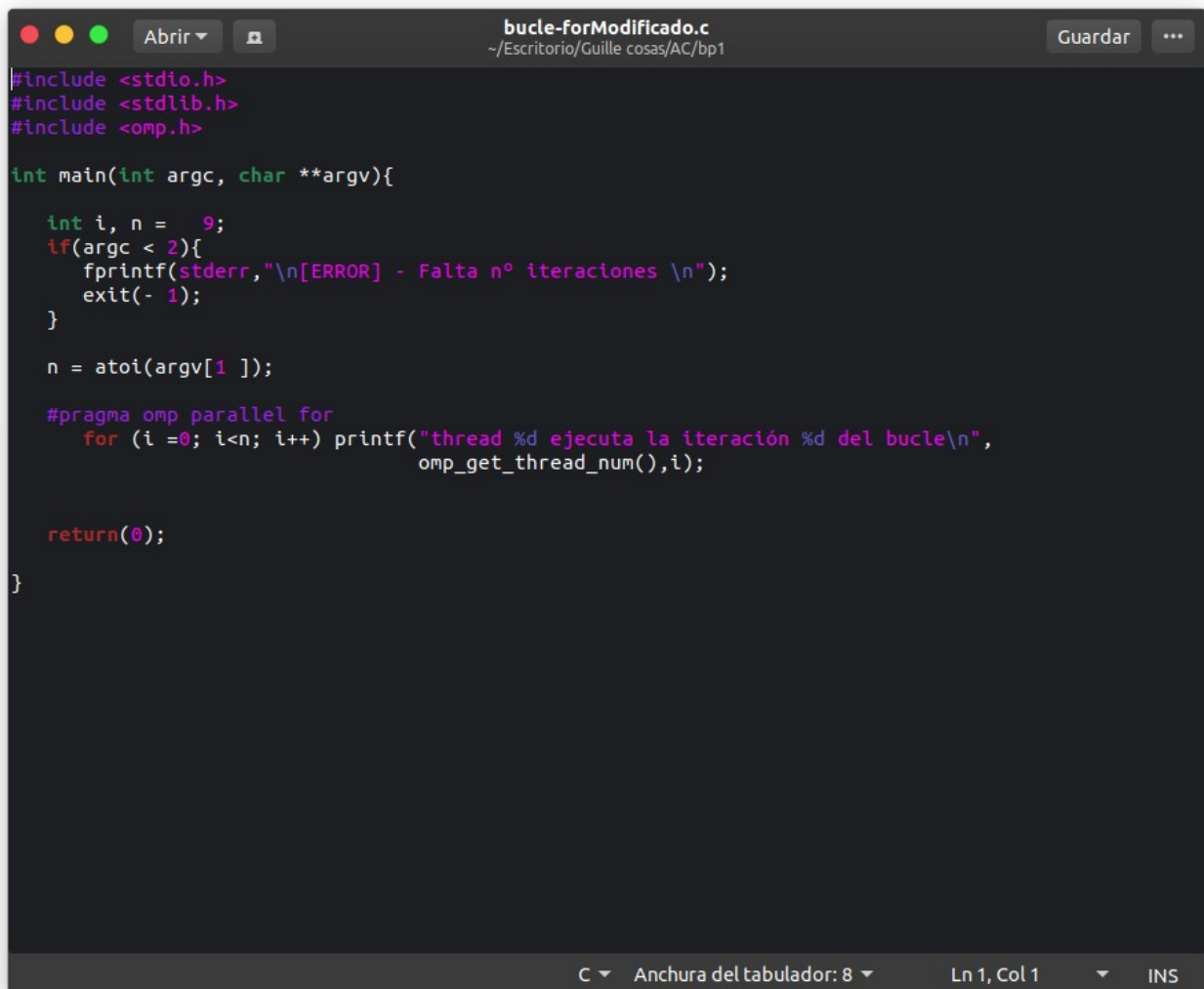
Fecha evaluación en clase:

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){

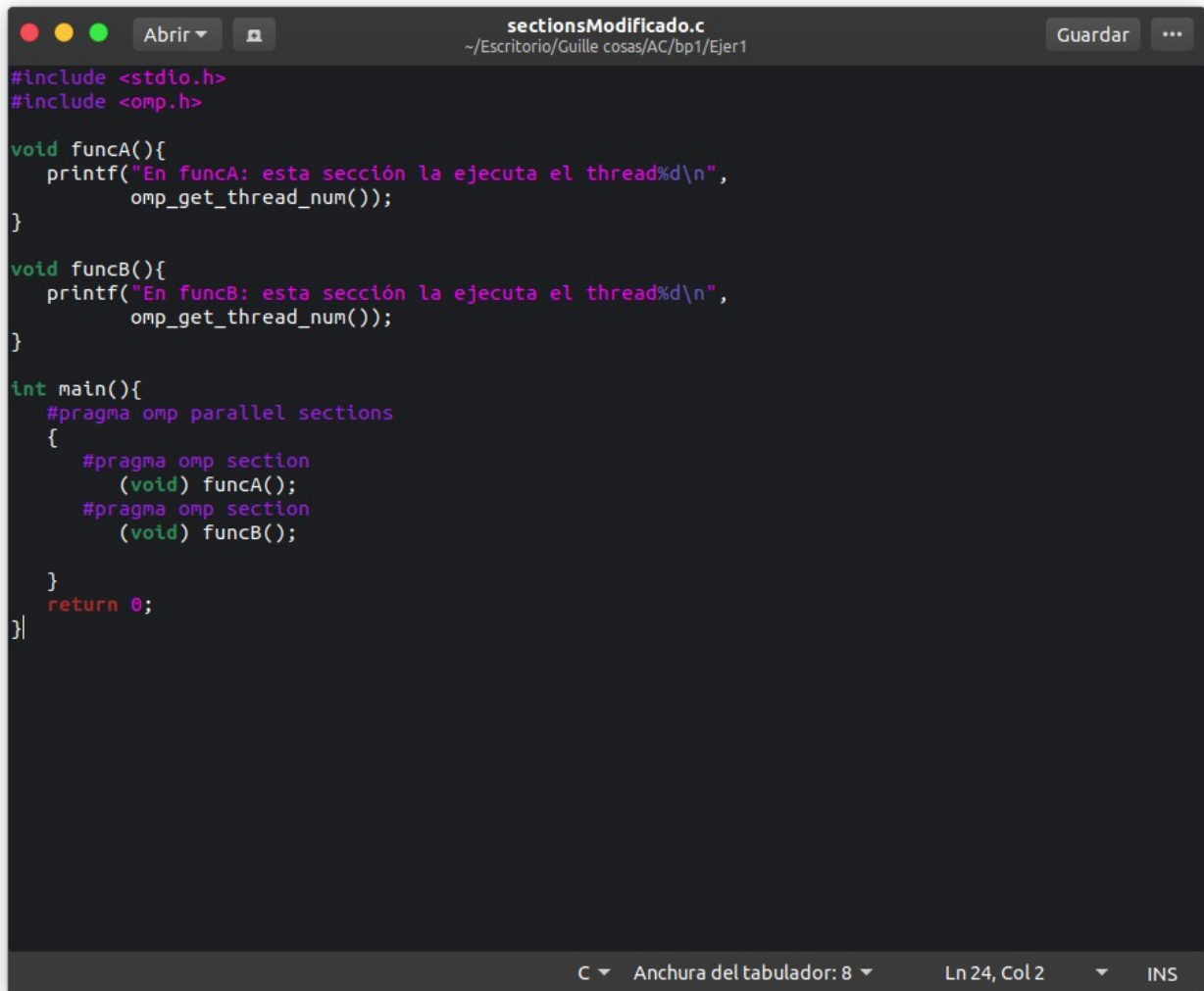
    int i, n = 9;
    if(argc < 2){
        fprintf(stderr, "\n[ERROR] - Falta nº iteraciones \n");
        exit(- 1);
    }

    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>

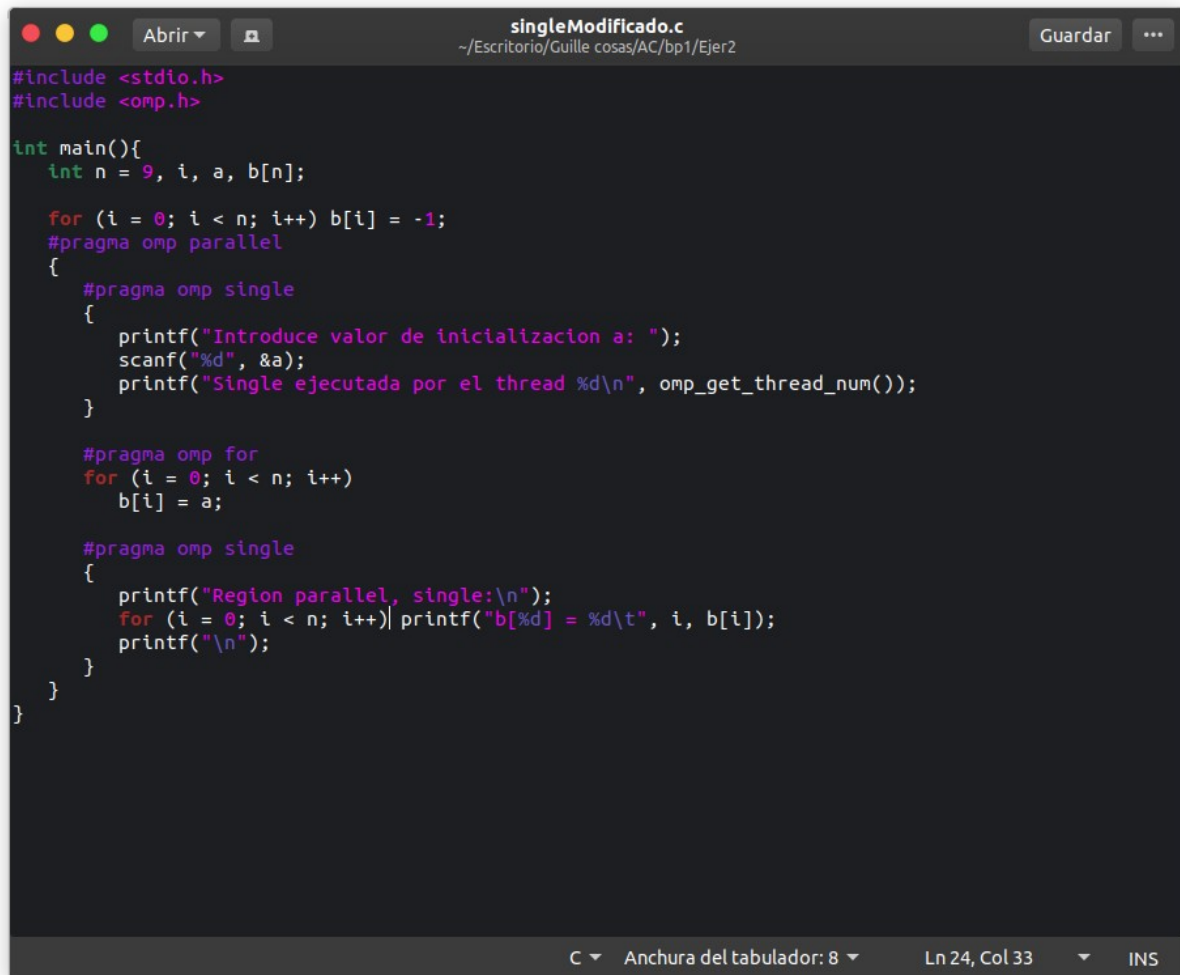
void funcA(){
    printf("En funcA: esta sección la ejecuta el thread%d\n",
           omp_get_thread_num());
}

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 <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 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("Region parallel, single:\n");
            for (i = 0; i < n; i++) printf("b[%d] = %d\t", i, b[i]);
            printf("\n");
        }
    }
}
```

C Anchura del tabulador: 8 Ln 24, Col 33 INS

```
Archivo Editar Ver Buscar Terminal Ayuda
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer2$ gcc -O2 -fopenmp singleModificado.c -o single
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer2$ ./single
Introduce valor de inicializacion a: 12
Single ejecutada por el thread 1
Region parallel, single:
b[0] = 12      b[1] = 12      b[2] = 12      b[3] = 12      b[4] = 12      b
[5] = 12      b[6] = 12      b[7] = 12      b[8] = 12
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer2$
```

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

**CAPTURAS DE PANTALLA:**

```

Archivo Editar Ver Buscar Terminal Ayuda
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer3$ gcc -O2 -fopenmp singleModificado.c -o single
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer3$ ./single
Introduce valor de inicializacion a: 10
Single ejecutada por el thread 1
Region parallel, master hebra 0:
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
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer3$

```

#### RESPUESTA A LA PREGUNTA:

La directiva master siempre imprime la hebra 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:

Al eliminar la directiva barrier, la directiva master, al no tener barreras implícitas, podría acabar antes que las otras hebras e imprimir un resultado parcial, que por tanto, sería erróneo.

### 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 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:

Es ligeramente menor, ya que el tiempo real incluye el tiempo de comunicación.

```

Archivo Editar Ver Buscar Terminal Ayuda
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer3$ gcc -O2 -fopenmp singleModificado.c -o single
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer3$ ./single
Introduce valor de inicializacion a: 10
Single ejecutada por el thread 1
Region parallel, master hebra 0:
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
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer3$

```

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. 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):

```
Archivo Editar Ver Buscar Terminal Ayuda
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer6$ gcc -O2 SumaVectores
C.c -S -lrt
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer6$ gcc -O2 SumaVectores
C.c -o ejecucion -lrt
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer6$ ./ejecucion 10
Tamaño Vectores:10 (4 B)
Tiempo:0.000000261 / Tamaño Vectores:10 / V1[0]+V2[0]=V3[0](1.000000+1.0
00000=2.000000) / / V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer6$ ./ejecucion 10000000
Tamaño Vectores:10000000 (4 B)
Tiempo:0.039042721 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](1000
000.000000+1000000.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3[999999
9](1999999.900000+0.100000=2000000.000000) /
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer6$
```

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

$(6 \cdot 10000000) / (0.039042721 \cdot 10^6) = 1536,778 \text{ MIPS}$

$(3 \cdot 10000000) / (0.039042721 \cdot 10^6) = 768,389 \text{ MFLOPS}$

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

```

subsd    %xmm0, %xmm7
movsd    %xmm2, 0(%rbp,%rax,8)
movsd    %xmm7, (%r14,%rax,8)
addq     $1, %rax
cmpq     %rax, %r15
jne      .L8
movq     %rsp, %rsi
xorl     %edi, %edi
salq     $3, %r15
call     clock_gettime@PLT
xorl     %eax, %eax
.p2align 4,,10
.p2align 3
.L9:
movsd    0(%rbp,%rax), %xmm0
addsd    (%r14,%rax), %xmm0
movsd    %xmm0, (%r12,%rax)
addq     $8, %rax
cmpq     %r15, %rax
jne      .L9
leaq     16(%rsp), %rsi
xorl     %edi, %edi
call     clock_gettime@PLT
movq     24(%rsp), %rax
subq     8(%rsp), %rax
pxor     %xmm0, %xmm0
pxor     %xmm1, %xmm1
cvtsi2sdq    %rax, %xmm0
movq     16(%rsp), %rax
subq     (%rsp), %rax
cmpl     $9, %r13d
cvtsi2sdq    %rax, %xmm1
divsd    .LC6(%rip), %xmm0
addsd    %xmm1, %xmm0
jbe      .L28
movl     %ebx, %eax
movsd    (%r12), %xmm3

```

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



```

SumaVectoresC.c
~/Escritorio/Guille cosas/AC/bp1/Ejer7

unsigned int N = atoi(argv[1]); // Tamaño N = 2550 >= 409600/256 (sizeof(unsigned int) * 4 B)
printf("Tamaño Vectores: %u B\n", N, (unsigned int) sizeof(unsigned int));
#ifdef VECTOR_LOCAL
double v1[N], v2[N], v3[N]; // Tamaño variable local en tiempo de ejecución ...
// disponible en C a partir de C99
#endif
#ifdef VECTOR_GLOBAL
if (N > MAX) N = MAX;
#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 (int i=0; i<N; i++){
    v1[i] = N*0.1*i*0.1; v2[i] = N*0.1*i*0.1;
}

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

double res = omp_get_wtime() - ini;

//Imprimir resultado de la suma y el tiempo de ejecución
if (N<10) {
    printf("Tiempo: %11.9f / Tamaño Vectores: %u\n", res, N);
    for (int i=0; i<N; i++){
        printf("/ V1[%d]+V2[%d]=V3[%d](%.6f+%.6f=%.6f) / \n",
            i, i, v1[i], v2[i], v3[i]);
    }
} else {
    printf("Tiempo: %11.9f / Tamaño Vectores: %u / V1[0]+V2[0]=V3[0](%.6f+%.6f=%.6f) / V1[%d]+V2[%d]=V3[%d](%.6f+%.6f=%.6f) / \n",
        res, N, v1[0], v2[0], v3[0], N-1, N-1, v1[N-1], v2[N-1], v3[N-1]);
}

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

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```

Archivo Editar Ver Buscar Terminal Ayuda
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer7$ gcc -O2 -fopenmp SumaVectoresC.c -o ejecutable -lrt
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer7$ export OMP_DYNAMIC=FALSE
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer7$ export OMP_NUM_THREADS=4
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer7$ ./ejecutable 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000004018 / 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) /
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer7$ ./ejecutable 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000003700 / 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) /
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer7$

```

- 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

```

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);
}

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

double int = omp_get_wtime();
//calcular suma de vectores
#pragma omp parallel sections
{
    #pragma omp section
    for(int i=0; i<N/4; i++){
        v3[i] = v1[i] + v2[i];
    }
    #pragma omp section
    for(int i=N/4; i<N/2; i++){
        v3[i] = v1[i] + v2[i];
    }
    #pragma omp section
    for(int i=N/2; i<3*N/4; i++){
        v3[i] = v1[i] + v2[i];
    }
    #pragma omp section
    for(int i=3*N/4; i<N; i++){
        v3[i] = v1[i] + v2[i];
    }
}

double res = omp_get_wtime() - int;

```

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

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

```

Archivo Editar Ver Buscar Terminal Ayuda
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer8$ gcc -O2 -fopenmp SumaVectoresC.c -o ejecutable -lrt
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer8$ export OMP_DYNAMIC=FALSE
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer8$ export OMP_NUM_THREADS=4
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer8$ ./ejecutable 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000003963 / 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) /
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer8$ ./ejecutable 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000004297 / 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) /
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp1/Ejer8$

```

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 cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta.

**RESPUESTA:**

En el ejercicio 7 podríamos aprovechar tantos threads como iteraciones tengamos (tamaño de N), mientras que en el ejercicio 8, solo podremos aprovechar 1 thread por cada section que declare el programador.

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:**

**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 ¿? 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						
131072						
262144						
524288						
1048576						
2097152						
4194304						
8388608						
16777216						
33554432						
67108864						