

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): Álvarez Ocete José Antonio

Grupo de prácticas: 2

Fecha de entrega:

Fecha evaluación en clase:

#### 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:** 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:** 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());
}

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

```

4. 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:** 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 inicialización a: ");
scanf("%d", &a );
printf("Single 1 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("Single 2 ejecutada por el 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\n",i,b[i]);
}
}
return 0;
}

```

**CAPTURAS DE PANTALLA:**

```
jose@jaaopc:~/Escritorio/DGIIM/AC/P1$ ./singleModificado
Introduce valor de inicialización a: 32
Single 1 ejecutada por el thread 2
Single 2 ejecutada por el thread 3
Después de la región parallel:
b[0] = 32
b[1] = 32
b[2] = 32
b[3] = 32
b[4] = 32
b[5] = 32
b[6] = 32
b[7] = 32
b[8] = 32
jose@jaaopc:~/Escritorio/DGIIM/AC/P1$
```

4. 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:** 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;

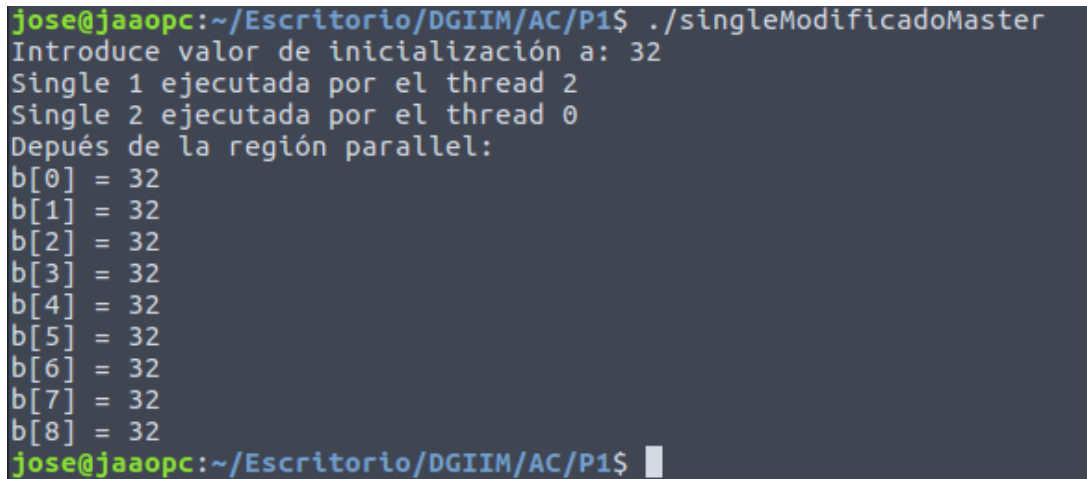
    #pragma omp parallel
    {
        #pragma omp single
        {
            printf("Introduce valor de inicialización a: ");
            scanf("%d", &a );
            printf("Single 1 ejecutada por el thread %d\n",
                omp_get_thread_num());
        }
        #pragma omp for
        for (i=0;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

        #pragma omp master
        {
            printf("Single 2 ejecutada por el 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\\n",i,b[i]);
    }
}
return 0;
}

```

**CAPTURAS DE PANTALLA:**


```

jose@jaaopc:~/Escritorio/DGIIM/AC/P1$ ./singleModificadoMaster
Introduce valor de inicialización a: 32
Single 1 ejecutada por el thread 2
Single 2 ejecutada por el thread 0
Depués de la región parallel:
b[0] = 32
b[1] = 32
b[2] = 32
b[3] = 32
b[4] = 32
b[5] = 32
b[6] = 32
b[7] = 32
b[8] = 32
jose@jaaopc:~/Escritorio/DGIIM/AC/P1$

```

**RESPUESTA A LA PREGUNTA:**

La única diferencial apreciable en este ejercicio es que es la hebra master (la 0) la que ejecuta el bloque estructurado de la directiva *master*, mientras que en la directiva *single* el bloque estructurado es ejecutado por la primera hebra que alcanza dicho punto del código.

Aquí no podemos apreciar la otra diferencia entre *single* y *master*. Esto es el hecho de que *single* presenta una barrera implícita al final del bloque. En este ejercicio no podemos apreciar esta barrera ya que el bloque *single/master* se encuentra al final del programa.

- . ¿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 la hebra master termina la sección *atomic* antes que el resto de hebras e imprime el resultado antes de que algunas de ellas hayan ejecutado la suma, el valor que se imprimirá por pantalla será la calculada hasta ese momento, por lo que será errónea. Esto ocurre porque la directiva *atomic* no tiene una barrera implícita al final del bloque.

---

## Resto de ejercicios

- . 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 el PC local, 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 mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

**CAPTURAS DE PANTALLA:**

```
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/Programas$ time ./SumaVectores 10
000000
Tiempo(seg.):0.034342239      / Tamaño Vectores:10000000      /V1[0]
+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) / /V1[9999
99]+V2[999999]=V3[999999](1999999.900000+0.100000=2000000.000000) /

real    0m0.096s
user    0m0.088s
sys     0m0.004s
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/Programas$
```

```
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/Programas$ time ./SumaVectores 10
000000
Tiempo(seg.):0.034438404      / Tamaño Vectores:10000000      /V1[0]
+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) / /V1[9999
99]+V2[999999]=V3[999999](1999999.900000+0.100000=2000000.000000) /

real    0m0.090s
user    0m0.052s
sys     0m0.036s
```

```
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/Programas$ time ./SumaVectores 10
000000
Tiempo(seg.):0.036220252      / Tamaño Vectores:10000000      /V1[0]
+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) / /V1[9999
99]+V2[999999]=V3[999999](1999999.900000+0.100000=2000000.000000) /

real    0m0.096s
user    0m0.080s
sys     0m0.016s
```

La suma de los tiempos de CPU del usuario (*user*) y del sistema (*system*) es siempre **menor o igual** que el tiempo real (*elapsed/real*). Obviamente el tiempo de ejecución del programa no puede ser mayor que el tiempo que el proceso pasa en modo usuario más el tiempo que pasa en modo kernel. Esto se cumple siempre. Las tres capturas son ejemplos de la ejecución en los que se cumple lo anteriormente razonado.

- . 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). Incorpore **el código ensamblador de la parte de la suma de vectores** en el cuaderno.

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

Sea  $N$  el nº de componentes del vector, estudiaremos los casos  $N = 10$  y  $N = 10M$ .

En primer lugar calculamos el número de instrucciones que se ejecutan. Hay 3 fuera del bucle (sin contar las llamadas a las funciones `clock_gettime`) y 6 dentro del bucle por lo que tendremos un total de  $3 + 6 \cdot N$  instrucciones.

En segunda lugar, calculamos el número de operaciones en coma flotante realizadas por segundo, 2 por iteración en el bucle. Mirando el código ensamblador contamos las instrucciones terminadas en “d”, que son las realizadas en coma flotante. Únicamente tenemos 3 de estas instrucciones por lo que se ejecutarán  $3 \cdot N$

En tercer lugar y último lugar calculamos los MIPS y los MFLOPS para cada caso (N = 10 y N = 10M), utilizando los tiempos obtenidos mediante la ejecución en atcgrid.

Para N = 10:

$$\text{MIPS} = \frac{3+6*10}{0.000003322*10^6} = 18,96447923 \text{ MIPS}$$

$$\text{MFLOPS} = \frac{3*10}{0.000003322*10^6} = 9,030704395 \text{ MIPS}$$

Para N = 10M:

$$\text{MIPS} = \frac{3+6*10^7}{0.075242381*10^7} = 797,422971 \text{ MIPS}$$

$$\text{MFLOPS} = \frac{3*10^7}{0.075242381*10^6} = 398,7114656 \text{ MIPS}$$

### CAPTURAS DE PANTALLA:

```
[E2estudiante2@atcgrid ~]$ qsub ./SumaVectores.sh
49978.atcgrid
[E2estudiante2@atcgrid ~]$ cat SumaVectores
SumaVectores      SumaVectores.o49978
SumaVectores.e49978 SumaVectores.sh
[E2estudiante2@atcgrid ~]$ cat SumaVectores.o49978
Id. usuario del trabajo: E2estudiante2
Id. del trabajo: 49978.atcgrid
Nombre del trabajo especificado por usuario: SumaVectores
Nodo que ejecuta qsub: atcgrid
Directorio en el que se ha ejecutado qsub: /home/E2estudiante2
Cola: ac
Nodos asignados al trabajo:
Para N = 10:
Tiempo(seg.):0.000003322      / Tamaño Vectores:10      /V1[0]+V2[0]=V
3[0](1.000000+1.000000=2.000000) / /V1[9]+V2[9]=V3[9](1.900000+0.10000
0=2.000000) /
Para N = 10M:
Tiempo(seg.):0.075242381      / Tamaño Vectores:10000000      /V1[0]
+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) / /V1[99999
99]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /
[E2estudiante2@atcgrid ~]$
```

**RESPUESTA:** código ensamblador generado de la parte de la suma de vectores

call	clock_gettime
	xorl %eax, %eax
	.p2align 4,,10
	.p2align 3
.L5:	
	movsd v1(%rax), %xmm0
	addq \$8, %rax
	addsd v2-8(%rax), %xmm0
	movsd %xmm0, v3-8(%rax)
	cmpq %rax, %rbx
	jne .L5
.L6:	
	leaq 16(%rsp), %rsi

```

xorl    %edi, %edi
call    clock_gettime

/*
Estas son las líneas 70-84 del archivo adjunto SumaVectores_CodigoEnsamblador,
volcado de SumaVectores.s
*/

```

- . 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:** código fuente implementado

```

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

// #define PRINTF_ALL

#define VECTOR_GLOBAL
#define MAX 67108864 // = 2^26
double v1[MAX], v2[MAX], v3[MAX];

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

    int i;
    double t_inicial, t_final; // para tiempo de ejecución

    // Leer argumento de entrada (nº de componentes del vector)
    if (argc < 2){
        printf("Faltan nº componentes del vector \n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N = 2^32 - 1 = 4294967295
                                    // (sizeof(unsigned int) = 4 B)

    if (N > MAX)
        N = MAX;

    // Inicializar vectores g

```

```

#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;
}
//los valores dependen de N

t_inicial = omp_get_wtime();

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

t_final = omp_get_wtime() - t_inicial;

//Imprimir resultado de la suma y el tiempo de ejecución
#ifdef PRINTF_ALL
    printf("Tiempo(seg.):%11.9f \t/ Tamaño Vectores:%u \t/", t_final, N);
    for (i=0; i<N; i++)
        printf("v3[%d] = %11.9f\n", i, v3[i]);
#else
    printf("Tiempo(seg.):%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",
        t_final, N, v1[0], v2[0], v3[0], N-1, N-1, N-1, v1[N-1], v2[N-1],
v3[N-1]);
#endif

return 0;
}

```



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

```

jose@jaaopc: ~/Escritorio/DGIIM/AC/P1/SumaVectores
SalidasATCGRID/ SumaVectoresParaleloTime.sh
SalidasMiPc/ SumaVectoresSections
Scripts/ SumaVectoresSections.c
SumaVectores SumaVectores.sh
SumaVectores.c SumaVectoresTime.sh
SumaVectores_CodigoEmsamblador
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/SumaVectores$ gcc -fopenmp -O2 SumaVectoresSections.c -o SumaVectoresSections -lrt
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/SumaVectores$ gcc -fopenmp -O2 SumaVectoresParalelo.c -o SumaVectoresParalelo -lrt
gcc: error: SumaVectoresParalelo.c: No existe el archivo o el directorio
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/SumaVectores$ gcc -fopenmp -O2 SumaVectoresParalelo.c -o SumaVectoresParalelo -lrt
gcc: error: SumaVectoresParalelo.c: No existe el archivo o el directorio
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/SumaVectores$ gcc -fopenmp -O2 SumaVectoresParalelo.c -o SumaVectoresParalelo -lrt
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/SumaVectores$ gcc -fopenmp -O2 SumaVectoresParalelo.c -o SumaVectoresParalelo -lrt
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/SumaVectores$ █

jose@jaaopc: ~
sftp> rm *.o*
Removing /home/E2estudiante2/SumaVectoresPruebaParalelo.o52869
sftp> ls
SumaVectoresPruebaParalelo
sftp> put SumaVectoresParalelo
Uploading SumaVectoresParalelo to /home/E2estudiante2/SumaVectoresParalelo
SumaVectoresParalelo 100% 9232 9.0KB/s 00:01
sftp> ls
SumaVectoresParalelo SumaVectoresPruebaParalelo
sftp> ls
SumaVectoresParalelo SumaVectoresPruebaParalelo
SumaVectoresPruebaParalelo.e52874 SumaVectoresPruebaParalelo.o52874
sftp> █

[E2estudiante2@atcgrid ~]$ qsub SumaVectoresPruebaParalelo
52874.atcgrid
[E2estudiante2@atcgrid ~]$ cat SumaVectoresPruebaParalelo.o*
Id. usuario del trabajo: E2estudiante2
Id. del trabajo: 52874.atcgrid
Nombre del trabajo especificado por usuario: SumaVectoresPruebaParalelo
Nodo que ejecuta qsub: atcgrid
Directorio en el que se ha ejecutado qsub: /home/E2estudiante2
Cola: ac
Nodos asignados al trabajo:
Para N = 10:
Tiempo(seg.):0.004425755 / Tamaño Vectores:8 /v3[0] = 1.6000000000000000
v3[1] = 1.6000000000
v3[2] = 1.6000000000
v3[3] = 1.6000000000
v3[4] = 1.6000000000
v3[5] = 1.6000000000
v3[6] = 1.6000000000
v3[7] = 1.6000000000
Para N = 10M:
Tiempo(seg.):0.004637871 / Tamaño Vectores:11 /v3[0] = 2.2000000000000000
v3[1] = 2.2000000000
v3[2] = 2.2000000000
v3[3] = 2.2000000000
v3[4] = 2.2000000000
v3[5] = 2.2000000000
v3[6] = 2.2000000000
v3[7] = 2.2000000000
v3[8] = 2.2000000000
v3[9] = 2.2000000000
v3[10] = 2.2000000000
[E2estudiante2@atcgrid ~]$ █

```

4. 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,  $N = 8$  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:** código fuente implementado

```

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

#define PRINTF_ALL
#define VECTOR_GLOBAL
#define MAX 67108864 //2^26
double v1[MAX], v2[MAX], v3[MAX];

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

    int i, tope;

```

```

double t_inicial, t_final;    //para tiempo de ejecución

//Leer argumento de entrada (nº de componentes del vector)
if (argc<2){
    printf("Faltan nº componentes del vector \n");
    exit(-1);
}

unsigned int N = atoi(argv[1]);    // Máximo N =2^32 -1=4294967295
(sizeof(unsigned int) = 4 B)
if (N>MAX)
    N=MAX;

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

t_inicial = omp_get_wtime();

//Calcular suma de vectores
#pragma omp parallel sections
{
    #pragma omp section
    {

```

```

        tope = N/4;
        for (i=0; i<tope; i++) {
            v3[i] = v1[i] + v2[i];
        }
    }
    #pragma omp section
    {
        tope = N/2;
        for (i=N/4; i<tope; i++) {
            v3[i] = v1[i] + v2[i];
        }
    }
    #pragma omp section
    {
        tope = 3*N/4;
        for (i=N/2; i<tope; i++) {
            v3[i] = v1[i] + v2[i];
        }
    }
    #pragma omp section
    {
        tope = N;
        for (i=3*N/4; i<tope; i++) {
            v3[i] = v1[i] + v2[i];
        }
    }
}

t_final = omp_get_wtime() - t_inicial;

//Imprimir resultado de la suma y el tiempo de ejecución
#ifdef PRINTF_ALL
    printf("Tiempo(seg.):%11.9f \t/ Tamaño Vectores:%u \t/", t_final, N);
    for (i=0; i<N; i++)
        printf("v3[%d] = %11.9f\n", i, v3[i]);
#else
    printf("Tiempo(seg.):%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",
        t_final, N, v1[0], v2[0], v3[0], N-1, N-1, N-1, v1[N-1], v2[N-1],
v3[N-1]);
#endif

    return 0;
}

```

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

```

jose@jaaopc: ~/Escritorio/DGIIM/AC/P1/SumaVectores
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/SumaVectores$ ls
Gráfica.ods          SumaVectoresParalelo
SalidasATCGRID       SumaVectoresParalelo.c
SalidasMiPc          SumaVectoresParaleloTime.sh
Scripts              SumaVectoresSections
SumaVectores         SumaVectoresSections.c
SumaVectores.c       SumaVectores.sh
SumaVectores_CodigoEmsamblador SumaVectoresTime.sh
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/SumaVectores$ gcc -fopenmp -O2 SumaVectoresSections.c -o SumaVectoresParalelo
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/SumaVectores$ gcc -fopenmp -O2 SumaVectoresSections.c -o SumaVectoresSections -lrt
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/SumaVectores$

jose@jaaopc: ~
sftp> rm .e*
Removing /home/E2estudiante2/.e*
Couldn't delete file: No such file or directory
sftp> rm *.e*
Removing /home/E2estudiante2/SumaVectoresPruebaSections.e52854
sftp> rm *.o*
Removing /home/E2estudiante2/SumaVectoresPruebaSections.o52854
sftp> ls
SumaVectoresPruebaSections      SumaVectoresSections
sftp> ls
SumaVectoresPruebaSections      SumaVectoresPruebaSections.e52859
SumaVectoresPruebaSections.o52859 SumaVectoresSections
sftp>

```

```

E2estudiante2@atcgrid:~
[E2estudiante2@atcgrid ~]$ qsub SumaVectoresPruebaSections
52859.atcgrid
[E2estudiante2@atcgrid ~]$ cat SumaVectoresPruebaSections.o52854
cat: SumaVectoresPruebaSections.o52854: No such file or directory
[E2estudiante2@atcgrid ~]$ cat SumaVectoresPruebaSections.o*
Id. usuario del trabajo: E2estudiante2
Id. del trabajo: 52859.atcgrid
Nombre del trabajo especificado por usuario: SumaVectoresPruebaSections
Nodo que ejecuta qsub: atcgrid
Directorio en el que se ha ejecutado qsub: /home/E2estudiante2
Cola: ac
Nodos asignados al trabajo:
Para N = 8:
Tiempo(seg.):0.004236175      / Tamaño Vectores:8      /v3[0] = 1.600
000000
v3[1] = 1.600000000
v3[2] = 1.600000000
v3[3] = 1.600000000
v3[4] = 1.600000000
v3[5] = 1.600000000
v3[6] = 1.600000000
v3[7] = 1.600000000
Para N = 11:
Tiempo(seg.):0.005591657      / Tamaño Vectores:11     /v3[0] = 2.200
000000
v3[1] = 2.200000000
v3[2] = 2.200000000
v3[3] = 2.200000000
v3[4] = 2.200000000
v3[5] = 2.200000000
v3[6] = 2.200000000
v3[7] = 2.200000000
v3[8] = 2.200000000
v3[9] = 2.200000000
v3[10] = 2.200000000
[E2estudiante2@atcgrid ~]$

```

- . ¿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 la versión implementada en el ejercicio 7, podremos utilizar tantas hebras (y cores) como tenga el procesador ya que se realiza una distribución dinámica de las iteraciones del bucle. En el caso de atcgrid utilizaríamos su 24 hebras (sus 12 cores) mientras que en el ordenador personal, con 2 cores y 2 hebras por core, también aprovecharía al máximo dichos recursos.

Por otro lado, la distribución que realiza la versión implementada en el ejercicio 8 es estática. Esto significa que independientemente del número de hebras de las que disponga la máquina en la que se ejecute el programa, unicamente podremos sacar partido a, como máximo, 4 de dichas hebras. En el caso de mi ordenador personal se saca máximo partido a sus recursos pues se utilizan sus 4 hebras. Sin embargo solamente podremos utilizar 4 herbas de atcgrid (2 cores), por lo que el tiempo de ejecución será mayor para grandes tamaños del vector. Podremos apreciar esto en el siguiente ejercicio.

- . Rellenar una tabla como la Tabla 2Error: no se encontró el origen de la referencia para atcgrid y otra para el PC local 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. Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

**RESPUESTA:** Adjunto las tablas con los datos y las gráficas. He puesto los ejes en formato logarítmico para que se vean mejor. La única apreciación a destacar es la mejora que se produce al

paralelizar el cálculo mediante el for frente a paralelizarlo mediante sections en atcgrid ya que como se ha comentado en el ejercicio anterior, atcgrid aprovecha mucho mejor la paralelización en este caso. Este detalle es apenas distinguible en la gráfica pero fácilmente apreciable en lat-  
abla

Esto no se puede apreciar para el PC personal ya que debido a la restricción sobre el número de hebras la paralelización es en esencia la misma, únicamente cambio la forma de distribuir las iteraciones.

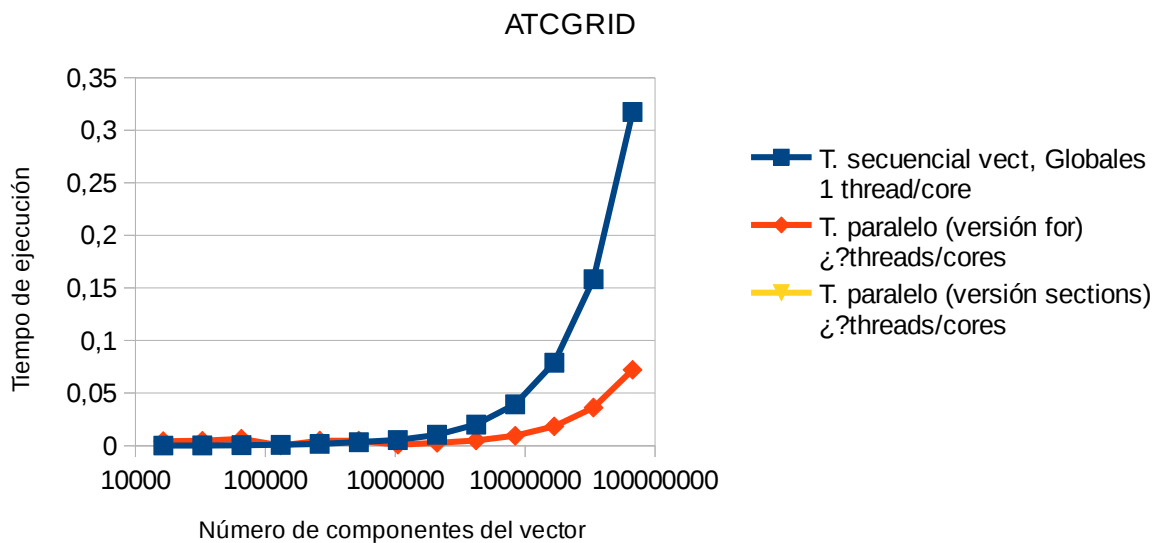
#### ATCGRID:

Nº de Componentes	T. secuencial vect, Globales 1 thread/core	T. paralelo (versión for) 24 threads/cores	T. paralelo (versión sections) 4 threads/cores
16384	0,00011488	0,00418877	0.006757118
32768	0,000194418	0,004623909	0.002147380
65536	0,000387518	0,006497946	0.002363225
131072	0,000788074	0,000167502	0.004625522
262144	0,001584133	0,004762385	0.000651298
524288	0,003239701	0,004706856	0.001576111
1048576	0,005383457	0,000966482	0.004711697
2097152	0,010300564	0,002605997	0.005762571
4194304	0,020062675	0,004969185	0.012401124
8388608	0,039292277	0,009445636	0.022575624
16777216	0,078920846	0,018401407	0.041518308
33554432	0,158294406	0,036223626	0.077475183
67108864	0,317379843	0,072185487	0.165436605

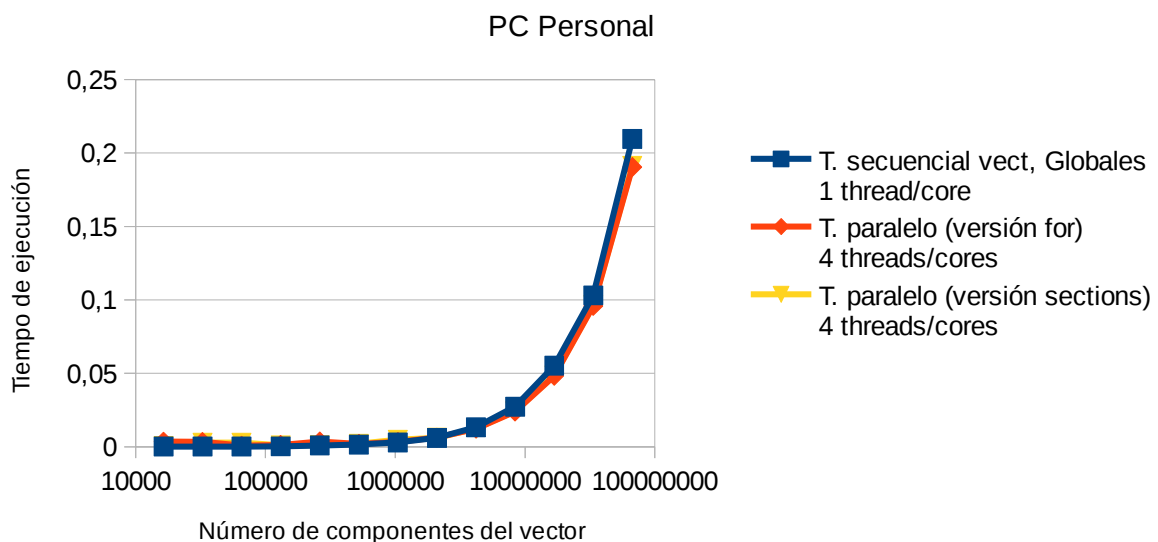
#### PC Personal:

Nº de Componentes	T. secuencial vect, Globales 1 thread/core	T. paralelo (versión for) 4 threads/cores	T. paralelo (versión sections) 4 threads/cores
16384	0,000187331	0,003562781	0,00071311
32768	0,000207951	0,003373672	0,002874773
65536	0,0002359	0,000812517	0,002855686
131072	0,000329453	0,001316516	0,00100356
262144	0,000931368	0,003637744	0,001223535
524288	0,001550678	0,002010716	0,002141825
1048576	0,003011745	0,003427914	0,00481576
2097152	0,006105224	0,006237251	0,006196455
4194304	0,013335993	0,012375529	0,01241685
8388608	0,027334643	0,024000631	0,024189966
16777216	0,055180816	0,048284426	0,048575172
33554432	0,103127566	0,095858743	0,096467783
67108864	0,209657819	0,190501258	0,191349509

## Tiempo de ejecución respecto del número de componentes del vector



## Tiempo de ejecución respecto del número de componentes del vector



- . Rellenar una tabla como la Tabla 3 para el PC local 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:** En la ejecución del programa secuencial se puede apreciar como el tiempo de CPU nunca supera al tiempo real (*elapsed*) obtenido. Sin embargo, para el programa con el for paralelizado el tiempo de CPU es la suma de los tiempos de ejecución de cada hebra. De esta forma el tiempo de CPU puede tomar cualquier valor respecto del tiempo de ejecución (mayor, menor o igual).

En este programa específico podemos observar como los tiempos de CPU de todas las ejecuciones superan el tiempo real (y en algunos casos hasta lo triplican).

Nota: Añado al final del documento capturas de pantalla de las ejecuciones de las cuales he obtenido los tiempos. También adjunto volcados de dichos datos en los documentos “TimesSecuencial” y “TimesParalelo” respectivamente.

**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 4 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	0,007	0,004	0,000	0,019	0,044	0,000
131072	0,007	0,004	0,000	0,006	0,008	0,000
262144	0,670	0,000	0,004	0,011	0,012	0,008
524288	0,020	0,000	0,004	0,007	0,008	0,008
1048576	0,014	0,004	0,008	0,009	0,024	0,000
2097152	0,019	0,016	0,000	0,020	0,048	0,012
4194304	0,040	0,036	0,000	0,037	0,092	0,020
8388608	0,080	0,060	0,020	0,061	0,188	0,028
16777216	0,144	0,108	0,032	0,114	0,304	0,100
33554432	0,288	0,224	0,060	0,218	0,592	0,216
67108864	0,291	0,228	0,060	0,218	0,636	0,176

```
jose@jaaopc: ~/Escritorio/DGIIM/AC/P1/SumaVectores
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/SumaVectores$ ./SumaVectoresTime.sh
Componentes del vector: 65536

real    0m0.007s
user    0m0.004s
sys     0m0.000s
Componentes del vector: 131072

real    0m0.007s
user    0m0.004s
sys     0m0.000s
Componentes del vector: 262144

real    0m0.670s
user    0m0.000s
sys     0m0.004s
Componentes del vector: 524288

real    0m0.020s
user    0m0.000s
sys     0m0.004s
Componentes del vector: 1048576

real    0m0.014s
user    0m0.004s
sys     0m0.008s
Componentes del vector: 2097152

real    0m0.019s
user    0m0.016s
sys     0m0.000s
Componentes del vector: 4194304

real    0m0.040s
user    0m0.036s
sys     0m0.000s
Componentes del vector: 8388608

real    0m0.080s
user    0m0.060s
sys     0m0.020s
Componentes del vector: 16777216

real    0m0.144s
user    0m0.108s
sys     0m0.032s
Componentes del vector: 33554432

real    0m0.288s
user    0m0.224s
sys     0m0.060s
Componentes del vector: 67108864

real    0m0.291s
user    0m0.228s
sys     0m0.060s
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/SumaVectores$
```



```
jose@jaaopc: ~/Escritorio/DGIIM/AC/P1/SumaVectores
jose@jaaopc:~/Escritorio/DGIIM/AC/P1/SumaVectores$ ./SumaVectores
ParaleloTime.sh
Componentes del vector: 65536

real    0m0.019s
user    0m0.044s
sys     0m0.000s
Componentes del vector: 131072

real    0m0.006s
user    0m0.008s
sys     0m0.000s
Componentes del vector: 262144

real    0m0.011s
user    0m0.012s
sys     0m0.008s
Componentes del vector: 524288

real    0m0.007s
user    0m0.008s
sys     0m0.008s
Componentes del vector: 1048576

real    0m0.009s
user    0m0.024s
sys     0m0.000s
Componentes del vector: 2097152

real    0m0.020s
user    0m0.048s
sys     0m0.012s
Componentes del vector: 4194304

real    0m0.037s
user    0m0.092s
sys     0m0.020s
Componentes del vector: 8388608

real    0m0.061s
user    0m0.188s
sys     0m0.028s
Componentes del vector: 16777216

real    0m0.114s
user    0m0.304s
sys     0m0.100s
Componentes del vector: 33554432

real    0m0.218s
user    0m0.592s
sys     0m0.216s
Componentes del vector: 67108864

real    0m0.218s
user    0m0.636s
sys     0m0.176s
```