

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): Álvaro Fernández García

Grupo de prácticas: A2

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 <stdlib.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(int argc, char ** argv) {
    #pragma omp parallel sections
    {
```

```

#pragma omp section
    (void) funcA();
#pragma omp section
    (void) funcB();
}
}

```

2. Imprimir los resultados del programa `single.c` usando una directiva `single` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `single` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `single`. Incorpore en su cuaderno de trabajo el código fuente y volcados de pantalla con los resultados de ejecución obtenidos.

RESPUESTA: 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 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");
            printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
            for (i=0; i<n; i++){ printf("b[%d] = %d\t",i,b[i]);
                printf("\n");
            }
        }
    }
}

```

CAPTURAS DE PANTALLA:

```

alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/P
áctica1$ gcc -O2 -fopenmp singleModificado.c -o single
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/P
áctica1$ ./single
Introduce valor de inicialización a: 8
Single ejecutada por el thread 1
Después de la región parallel:
Single ejecutada por el thread 2
b[0] = 8
b[1] = 8
b[2] = 8
b[3] = 8
b[4] = 8
b[5] = 8
b[6] = 8
b[7] = 8
b[8] = 8
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/P

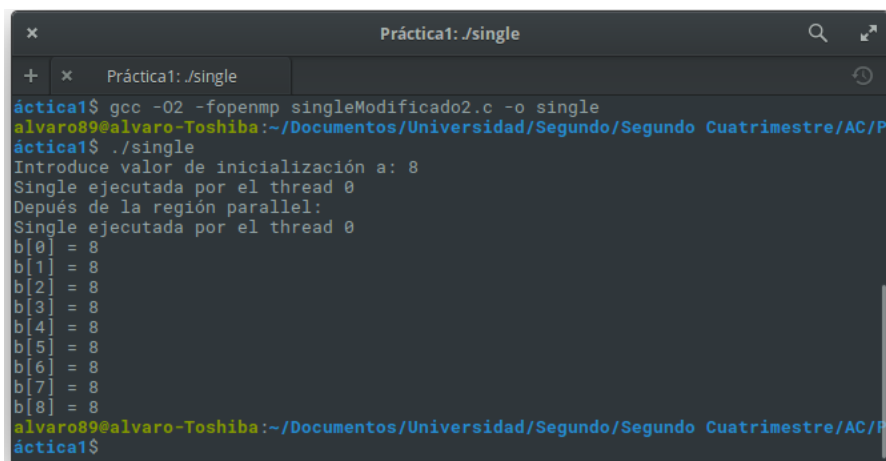
```

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: código fuente `singleModificado2.c`

```
#include <stdio.h>
#include <omp.h>
int main() {
    int n = 9, i, a, b[n];
    for (i=0; i<n; i++)
        b[i] = -1;
    #pragma omp parallel
    {
        #pragma omp single
        {
            printf("Introduce valor de inicialización a: ");
            scanf("%d", &a );
            printf("Single ejecutada por el thread %d\n",    omp_get_thread_num());
        }
        #pragma omp for
        for (i=0; i<n; i++)
            b[i] = a;
        #pragma omp master
        {
            printf("Depués de la región parallel:\n");
            printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
            for (i=0; i<n; i++){ printf("b[%d] = %d\t",i,b[i]);
                                printf("\n");
            }
        }
    }
}
```

CAPTURAS DE PANTALLA:



```
Práctica1: ./single
álctica1$ gcc -O2 -fopenmp singleModificado2.c -o single
álvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/P
álctica1$ ./single
Introduce valor de inicialización a: 8
Single ejecutada por el thread 0
Depués de la región parallel:
Single ejecutada por el thread 0
b[0] = 8
b[1] = 8
b[2] = 8
b[3] = 8
b[4] = 8
b[5] = 8
b[6] = 8
b[7] = 8
b[8] = 8
álvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/P
álctica1$
```

RESPUESTA A LA PREGUNTA:

La diferencia entre la directiva `single` y la directiva `master`, es que en la directiva `single` el bloque estructurado lo ejecuta una hebra cualquiera (en el ejercicio anterior ha sido la hebra 2), mientras que por el contrario en la directiva `master`, el bloque estructurado lo ejecuta

obligatoriamente la hebra maestra (lo cual es cierto puesto que en este ejercicio está ejecutando el bloque la hebra 0 (hebra maestra)). Otra diferencia es que master no tiene barrera implícita al final, single sí la tiene, aunque en este ejemplo eso es irrelevante.

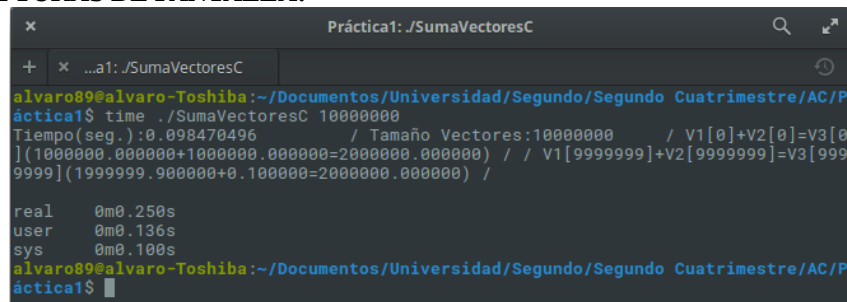
4. ¿Por qué si se elimina directiva barrier en el ejemplo master.c la suma que se calcula e imprime no siempre es correcta? Responda razonadamente.

RESPUESTA: Porque antes de poder imprimir el resultado de la suma, debemos de asegurarnos de que todas las hebras han añadido su parte calculada a la variable total, de ahí que se aparezca la directiva barrier, para esperar a que todas las hebras hayan llegado a ella. Por el contrario, al eliminarla, cuando la hebra master añada la sumalocal a suma, ejecutará el bloque estructurado del master e imprimirá el resultado. Por tanto la suma no presentará el valor de las hebras que todavía no han sumado su valor a la variable suma y de ahí que el resultado no sea correcto.

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 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:



```

x      Práctica1: ./SumaVectoresC
+ x    ...a1: ./SumaVectoresC
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/P
áctica1$ time ./SumaVectoresC 10000000
Tiempo(seg.):0.098470496 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0
](1000000.000000+1000000.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3[999
9999](1999999.900000+0.100000=2000000.000000) /
real    0m0.250s
user    0m0.136s
sys     0m0.100s
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/P
áctica1$

```

La suma de los tiempos de CPU del usuario y del sistema es menor que el tiempo real (236) esto se debe fundamentalmente a los tiempos de espera, que en este caso suma un total de 14.

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

CAPTURAS DE PANTALLA:

```

x A2estudiante6@atcgrid:~
+ Práctica1: gcc x ...diente6@atcgrid:~ ...e6@atcgrid.ugr.es
[A2estudiante6@atcgrid ~]$ echo './SumaVectoresC 10' | qsub -q ac
50280.atcgrid
[A2estudiante6@atcgrid ~]$ cat STDIN.o50280
Tiempo(seg.):0.000000167 / 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
) /
[A2estudiante6@atcgrid ~]$

```

```

x A2estudiante6@atcgrid:~
+ Práctica1: gcc x ...diente6@atcgrid:~ ...e6@atcgrid.ugr.es
[A2estudiante6@atcgrid ~]$ echo './SumaVectoresC 100000000' | qsub -q ac
50281.atcgrid
[A2estudiante6@atcgrid ~]$ cat STDIN.o50281
Tiempo(seg.):0.475321877 / Tamaño Vectores:100000000 / V1[0]+V2[0
]=V3[0](10000000.000000+10000000.000000=20000000.000000) / / V1[99999999]+V2
[99999999]=V3[99999999](19999999.900000+0.100000=20000000.000000) /
[A2estudiante6@atcgrid ~]$

```

RESPUESTA: cálculo de los MIPS y los MFLOPS

	MIPS	MFLOPS
Para tamaño 10	$\frac{65}{0,000000167 * 10^6} = 389,221$	$\frac{30}{0,000000167 * 10^6} = 179,640$
Para tamaño 10000000	$\frac{6 * 10^7 + 5}{0,475321877 * 10^6} = 126,2303$	$\frac{3 * 10^7}{0,475321877 * 10^6} = 63,115$

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

	call	clock_gettime
	xorl	%eax, %eax
	.p2align 4,,10	
	.p2align 3	
.L9:	movsd	0(%rbp,%rax,8), %xmm0
	addsd	(%r12,%rax,8), %xmm0
	movsd	%xmm0, (%r15,%rax,8)
	addq	\$1, %rax
	cmpl	%eax, %r14d
	ja	.L9
.L10:	leaq	16(%rsp), %rsi
	xorl	%edi, %edi
	call	clock_gettime

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: código fuente implementado

```
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <omp.h>

int main ( int argc, char ** argv){
    int i;
    double cgt1,cgt2;
    double ncgt; //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
    (sizeof(unsigned int) = 4B)

    double *v1, *v2, *v3;
    v1 = (double*) malloc(N*sizeof(double));
    v2 = (double*) malloc(N*sizeof(double));
    v3 = (double*) malloc(N*sizeof(double));
    if( (v1 == NULL) || (v2 == NULL) || (v3 == NULL)){
        printf("Error en la reserva del espacio para los
vecres\n");
        exit(-2);
    }

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

        #pragma omp single
        { cgt1 = omp_get_wtime();}

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

        #pragma omp single
        {cgt2 = omp_get_wtime();}
```

```

    }
    ncgt = cgt2 - cgt1;

    // Imprimir
    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",ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);

    free(v1); // libera el espacio reservado para v1
    free(v2); // libera el espacio reservado para v2
    free(v3); // libera el espacio reservado para v3

    return 0;
}

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```

Códigos Fuente: ./Listado1OMP
+ x ...uente: ./Listado1OMP
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctica1/Código
s Fuente$ gcc -O2 -fopenmp -o Listado1OMP Listado1-OMP.c
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctica1/Código
s Fuente$ ./Listado1OMP 8
Tiempo(seg.):0.000004815 / Tamaño Vectores:8 / V1[0]+V2[0]=V3[0](0.800000+0.800000=1.60
0000) / / V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctica1/Código
s Fuente$

```

```

Códigos Fuente: ./Listado1OMP
+ x ...uente: ./Listado1OMP
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctica1/Código
s Fuente$ gcc -O2 -fopenmp -o Listado1OMP Listado1-OMP.c
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctica1/Código
s Fuente$ ./Listado1OMP 11
Tiempo(seg.):0.000003677 / Tamaño Vectores:11 / V1[0]+V2[0]=V3[0](1.100000+1.100000=2.20
0000) / / V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctica1/Código
s Fuente$

```

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 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> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <omp.h>

int main ( int argc, char ** argv){
    int i;
    double cgt1,cgt2;
    double ncgt; //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
    (sizeof(unsigned int) = 4B)

    double *v1, *v2, *v3;
    v1 = (double*) malloc(N*sizeof(double));
    v2 = (double*) malloc(N*sizeof(double));
    v3 = (double*) malloc(N*sizeof(double));
    if( (v1 == NULL) || (v1 == NULL) || (v1 == NULL)){
        printf("Error en la reserva del espacio para los
vecorres\n");
        exit(-2);
    }

    int num_it = N / 4; //Repartir las iteraciones entre las cuatro
secciones.

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

```



```

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

#pragma omp single
{ cgt1 = omp_get_wtime();}

// Calcular la suma de vectores
#pragma omp sections private(i)
{
    #pragma omp section
    for(i = 0; i < num_it; i++){
        v3[i] = v1[i] + v2[i];
    }
    #pragma omp section
    for(i = num_it; i < num_it*2; i++){
        v3[i] = v1[i] + v2[i];
    }
    #pragma omp section
    for(i = num_it*2; i < num_it*3; i++){
        v3[i] = v1[i] + v2[i];
    }
    #pragma omp section
    for(i = num_it*3; i < N; i++){
        v3[i] = v1[i] + v2[i];
    }
}

#pragma omp single
{cgt2 = omp_get_wtime();}

}
ncgt = cgt2 - cgt1;

// Imprimir
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",ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);

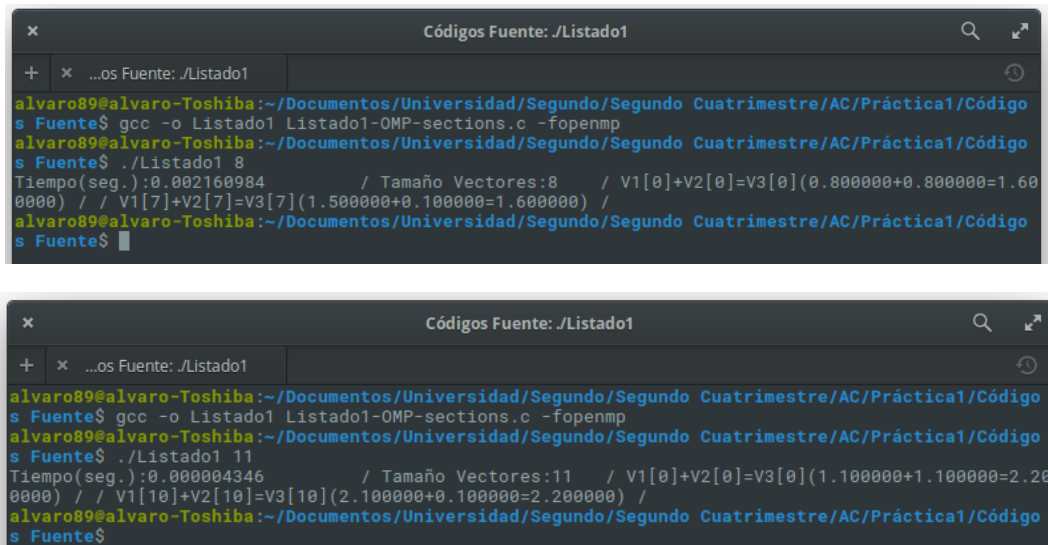
free(v1); // libera el espacio reservado para v1
free(v2); // libera el espacio reservado para v2
free(v3); // libera el espacio reservado para v3

return 0;
}

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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



```

Códigos Fuente: ./Listado1
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctica1/Código
s Fuente$ gcc -o Listado1 Listado1-OMP-sections.c -fopenmp
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctica1/Código
s Fuente$ ./Listado1 8
Tiempo(seg.):0.002160984 / Tamaño Vectores:8 / V1[0]+V2[0]=V3[0](0.800000+0.800000=1.60
0000) / / V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctica1/Código
s Fuente$

Códigos Fuente: ./Listado1
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctica1/Código
s Fuente$ gcc -o Listado1 Listado1-OMP-sections.c -fopenmp
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctica1/Código
s Fuente$ ./Listado1 11
Tiempo(seg.):0.000004346 / Tamaño Vectores:11 / V1[0]+V2[0]=V3[0](1.100000+1.100000=2.20
0000) / / V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
alvaro89@alvaro-Toshiba:~/Documentos/Universidad/Segundo/Segundo Cuatrimestre/AC/Práctica1/Código
s Fuente$

```

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: Con respecto a la versión implementada en el ejercicio 7, lo que realiza la directiva de trabajo compartido for es repartir el número de iteraciones del bucle entre las hebras disponibles, por tanto como mínimo una hebra tendrá que realizar una iteración. En consecuencia el número máximo de threads del ejercicio 7 será N, que representa el tamaño del vector. Sin embargo el número de cores que pueden ejecutar dicha aplicación en paralelo será el número de cores físicos de los que disponga el computador, en este caso 4 cores. Por otra parte en el ejercicio 8, el número de threads máximo será igual al número de secciones utilizadas en el código (dado que cada sección está ejecutada por una sola hebra). En este caso se ha optado por una implementación con 4 secciones, por tanto el número máximo de hebras es 4. Con respecto a los cores se repite lo descrito en el caso anterior.

10. Rellenar una tabla como la Tabla 2 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: A continuación se muestran las tablas de tiempos obtenidas para Atcgrid y para el PC local. Las gráficas se encuentran en la hoja de cálculo “Tablas y gráficas.ods”

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 utilizados.

Ejercicio 10 (PC LOCAL)			
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	0,000162588	0,000063221	0,001913702
32768	0,000342636	0,002856027	0,002781329
65536	0,00065247	0,004656402	0,00415584
131072	0,001763162	0,000809298	0,000682173
262144	0,003149612	0,001527356	0,001544234
524288	0,005270367	0,004216166	0,006336929
1048576	0,011152091	0,007949723	0,011438336
2097152	0,020686549	0,01746194	0,016105072
4194304	0,035964291	0,03201191	0,031951178
8388608	0,069817454	0,063312955	0,063846474
16777216	0,136288326	0,130056199	0,134874551
33554432	0,270505468	0,260844405	0,258994989
67108864	0,543524631	0,513367823	0,512341998

Ejercicio 10 (ATCGRID)			
Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 12 threads/cores	T. paralelo (versión sections) 12 threads/cores
16384	0,000096481	0,000032216	0,000073781
32768	0,000188164	0,001072584	0,000089934
65536	0,000355159	0,000166306	0,000137879
131072	0,000786	0,000146477	0,000217047
262144	0,001442283	0,000273231	0,000732644
524288	0,003184004	0,005513623	0,001555469
1048576	0,005123726	0,001828974	0,002750626
2097152	0,010077425	0,003305376	0,004427582
4194304	0,0198329	0,005050261	0,007606344
8388608	0,039620155	0,012035219	0,016545875
16777216	0,078859038	0,022837428	0,035026152
33554432	0,159564842	0,036609815	0,062896669
67108864	0,31562866	0,073918981	0,13693789

11. 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 el caso de la ejecución secuencial se verifica que el tiempo de CPU (user + sys) es menor o igual que tiempo real, debido fundamentalmente a la existencia de esperas. Sin embargo en el caso de la ejecución paralela, se cumple que el tiempo de CPU es mayor estricto que el tiempo real. Esto se debe a que el tiempo de CPU se calcula como la suma del tiempo que han estado en CPU cada una de las hebras, sin embargo como se están ejecutando en paralelo, es decir, al mismo tiempo, el tiempo real es menor. No obstante si se realizara el promedio (dividir el tiempo de CPU entre el número de hebras) volvería a cumplirse lo dicho en el primer caso.

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 Componente s	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	0m0.004s	0m0.000s	0m0.000s	0m0.009s	0m0.016s	0m0.004s
131072	0m0.006s	0m0.004s	0m0.000s	0m0.009s	0m0.008s	0m0.008s
262144	0m0.010s	0m0.004s	0m0.008s	0m0.009s	0m0.016s	0m0.004s
524288	0m0.015s	0m0.004s	0m0.008s	0m0.016s	0m0.024s	0m0.020s
1048576	0m0.029s	0m0.008s	0m0.020s	0m0.029s	0m0.032s	0m0.032s
2097152	0m0.055s	0m0.028s	0m0.024s	0m0.050s	0m0.080s	0m0.040s
4194304	0m0.103s	0m0.052s	0m0.048s	0m0.091s	0m0.132s	0m0.084s
8388608	0m0.199s	0m0.124s	0m0.072s	0m0.173s	0m0.372s	0m0.148s
16777216	0m0.383s	0m0.192s	0m0.188s	0m0.333s	0m0.624s	0m0.352s
33554432	0m0.689s	0m0.440s	0m0.248s	0m0.647s	0m1.472s	0m0.652s
67108864	0m1.295s	0m0.796s	0m0.492s	0m1.230s	0m2.508s	0m1.580s