

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):

Grupo de prácticas:

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: Captura que muestre el código fuente `bucle-forModificado.c`

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <omp.h>
4
5 int main(int argc, char **argv) {
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    for (i=0; i<n; i++)
14        printf("thread %d ejecuta la iteración %d del bucle\n", omp_get_thread_num(), i);
15
16    return(0);
17 }
```

RESPUESTA: Captura que muestre el código fuente `sectionsModificado.c`

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

- 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`

```
1 #include <stdio.h>
2 #include <omp.h>
3 void main() {
4
5     int n = 9, i, a, b[n];
6     for (i=0; i<n; i++) b[i] = -1;
7
8     #pragma omp parallel
9     {
10         #pragma omp single
11         {
12             printf("Introduce valor de inicialización a: ");
13             scanf("%d", &a );
14             printf("Single Inicial ejecutada por el thread %d\n", omp_get_thread_num());
15         }
16
17         #pragma omp for
18         for (i=0; i<n; i++)
19             b[i] = a;
20         //printf("Iteración Inicial ejecutada por el thread %d\n", omp_get_thread_num());
21
22         //#pragma omp barrier
23         //hay barrera implícita después del for
24
25         #pragma omp single
26         {
27             printf("Imprimo dentro de la región parallel con la directiva single:\n");
28             printf("Single Final ejecutada por el thread %d\n", omp_get_thread_num());
29             for (i=0; i<n; i++){
30                 //printf("Iteración Final ejecutada por el thread %d\n", omp_get_thread_num());
31                 printf("b[%d] = %d\n",i,b[i]);
32             }
33         }
34     }
35
36 }
37
38
39
40 }
```

CAPTURAS DE PANTALLA:

```
guillesiesta@guillesiesta:~$ gcc -O2 singleModificado.c -o singleMod -fopenmp
guillesiesta@guillesiesta:~$ ./singleMod
Introduce valor de inicialización a: 666
Single Inicial ejecutada por el thread 0
Imprimo dentro de la región parallel con la directiva single:
Single Final ejecutada por el thread 7
b[0] = 666
b[1] = 666
b[2] = 666
b[3] = 666
b[4] = 666
b[5] = 666
b[6] = 666
b[7] = 666
b[8] = 666
```

- 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`

```

1 #include <stdio.h>
2 #include <omp.h>
3 void main() {
4
5     int n = 9, i, a, b[n];
6     for (i=0; i<n; i++) b[i] = -1;
7
8     #pragma omp parallel
9     {
10         #pragma omp single
11         {
12             printf("Introduce valor de inicialización a: ");
13             scanf("%d", &a );
14             printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
15         }
16
17         #pragma omp for
18         for (i=0; i<n; i++)
19             b[i] = a;
20             printf("For ejecutada por el thread %d\n", omp_get_thread_num());
21
22         // #pragma omp barrier
23         // hay barrera implícita después del for
24
25         #pragma omp master
26         {
27             printf("MASTER->%d:\n", omp_get_thread_num());
28             for (i=0; i<n; i++) printf("b[%d] = %d\n", i, b[i]);
29             printf("\n");
30         }
31     }
32 }

```

CAPTURAS DE PANTALLA:

```

guillesiesta@guillesiesta:~$ gcc -O2 singleModificado2.c -o singleMod2 -fopenmp
guillesiesta@guillesiesta:~$ ./singleMod2
Introduce valor de inicialización a: 666
Single ejecutada por el thread 5
MASTER->0:
b[0] = 666
b[1] = 666
b[2] = 666
b[3] = 666
b[4] = 666
b[5] = 666
b[6] = 666
b[7] = 666
b[8] = 666

```

RESPUESTA A LA PREGUNTA: La principal diferencia es que al poner la directiva `single`, ese trozo de código lo puede ejecutar cualquier hebra, pero si ponemos `master`, lo ejecuta la hebra 0, que es la máster.

- 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: `Barrier` se usa para poner una barrera antes de la directiva `master` porque ésta no tiene barrera implícita, por lo que si la quitamos, la hebra 0 imprime resultados de la suma que no son correctos pues no espera a que todas las hebras sumen su `sumalocal` a `suma` (la suma total)

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:

```

B3estudiante30@atcgrid:~
-rwxrwxr-x 1 B3estudiante30 B3estudiante30 8968 mar  8 12:57 SumaVectoresCGlob
-rwxrwxr-x 1 B3estudiante30 B3estudiante30 756 mar  8 12:57 SumaVectoresDlna.sh
-rwxrwxr-x 1 B3estudiante30 B3estudiante30 756 mar  8 12:57 SumaVectoresGlob.sh
-rwxrwxr-x 1 B3estudiante30 B3estudiante30 752 mar  8 12:36 SumaVectores.sh
[GuilleMuriel B3estudiante30@atcgrid:~] 2018-04-03 martes
$gstat
[GuilleMuriel B3estudiante30@atcgrid:~] 2018-04-03 martes
$echo 'time ./SumaVectoresCGlob 10000000' | qsub -q ac
71412.atcgrid
[GuilleMuriel B3estudiante30@atcgrid:~] 2018-04-03 martes
$cat STDIN.o71412
Tiempo(seg.):0.061648147 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=20
00000.000000) / / V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /
[GuilleMuriel B3estudiante30@atcgrid:~] 2018-04-03 martes
$cat STDIN.e71412
real    0m0.175s
user    0m0.055s
sys     0m0.117s
[GuilleMuriel B3estudiante30@atcgrid:~] 2018-04-03 martes
$

```

La suma del tiempo de CPU de usuario y de tiempo de CPU de sistema es:

$$0,055+0,117 = 0,172$$

$$T^{\circ}\text{CPU_USER} + T^{\circ}\text{CPU_SYS} < T^{\circ}\text{REAL}$$

El resultado de la suma anterior es menor que el tiempo real (0,175). Esto se debe a que se ha añadido un tiempo de 0,003 debido a las esperas debidas a la Entrada y Salida o esperas asociadas a la ejecución de otros programas.

- 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:

```
[GuilleMuriel B3estudiante30@atcgrid:~] 2018-04-03 martes
$echo './SumaVectoresCGlob 10' | qsub -q ac
71417.atcgrid
[GuilleMuriel B3estudiante30@atcgrid:~] 2018-04-03 martes
$cat STDIN.o71417
Tiempo(seg.):0.000002875 / 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) /
[GuilleMuriel B3estudiante30@atcgrid:~] 2018-04-03 martes
$echo './SumaVectoresCGlob 10000000' | qsub -q ac
71418.atcgrid
[GuilleMuriel B3estudiante30@atcgrid:~] 2018-04-03 martes
$cat STDIN.o71418
Tiempo(seg.):0.061888188 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=20
00000.000000) / / V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /
[GuilleMuriel B3estudiante30@atcgrid:~] 2018-04-03 martes
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

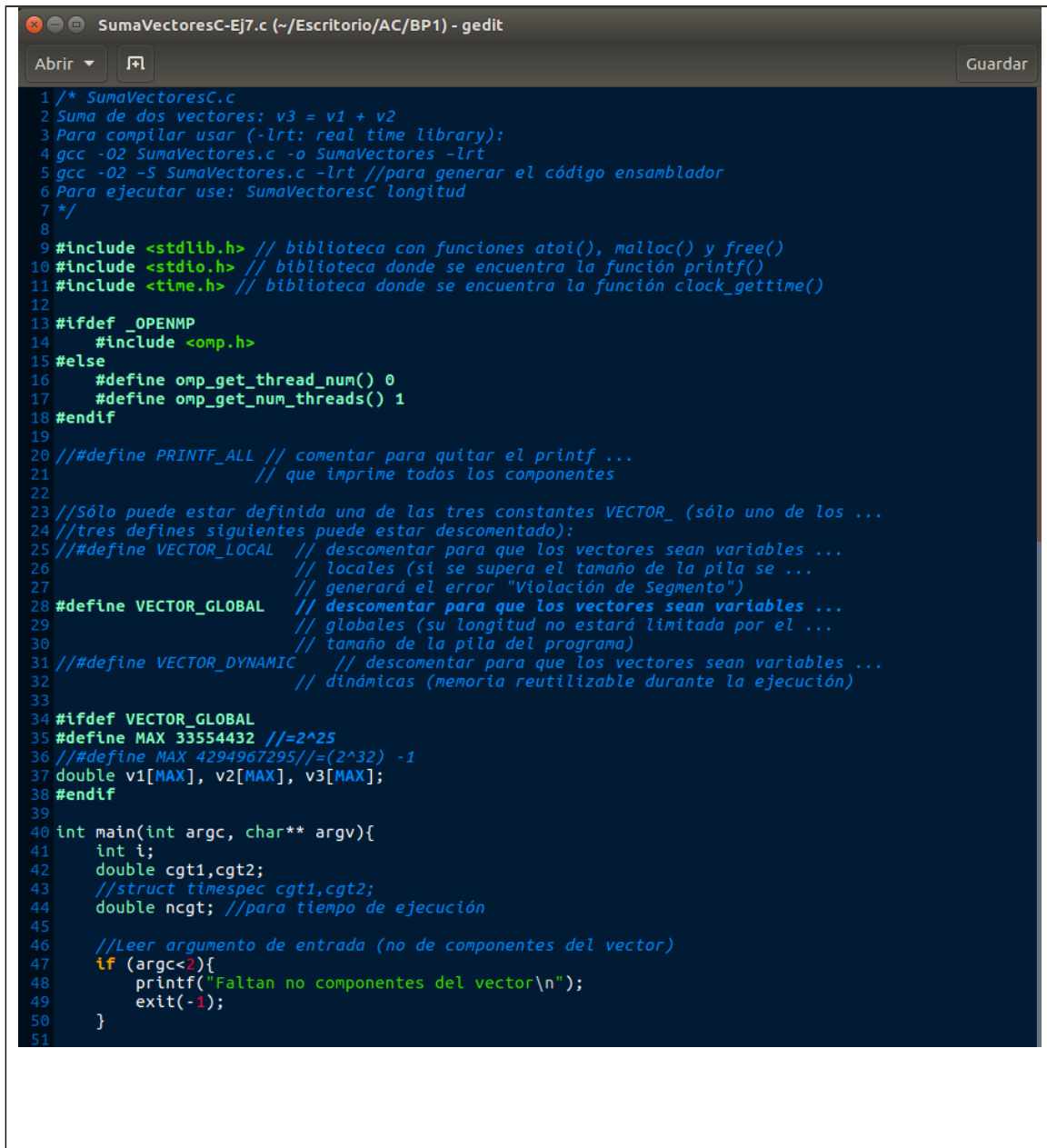
Tamaño vector =10	Tamaño vector =10000000
Número de Instrucciones: 3 +(6*10)	Número de Instrucciones: 3+(6*10000000)
iteraciones = 63	iteraciones) = 60000003
Operaciones en coma flotante: 3 *10	Operaciones en coma flotante:
iteraciones= 30	3*10000000=30000000
Tiempo(seg.): 0.000002875	Tiempo(seg.): 0.061888188
MIPS:63/(0.000002875*10^6)=21,913043478	MIPS:60000003/
MFLOPS:30/0.000002875*10^6=10,4347826	(0.061888188*10^6)=969,490381589
09	MFLOPS:30000000/
	(0.061888188*10^6)=484,745166557

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

<pre> 70 call clock_gettime 71 xorl %eax, %eax 72 .p2align 4,,10 73 .p2align 3 74 .L5: 75 movsd v1(%rax), %xmm0 76 addq \$8, %rax 77 addsd v2-8(%rax), %xmm0 78 movsd %xmm0, v3-8(%rax) 79 cmpq %rax, %rbx 80 jne .L5 81 .L6: 82 leaq 16(%rsp), %rsi 83 xorl %edi, %edi 84 call clock_gettime </pre>
--

- 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



```
1 /* SumaVectoresC.c
2 Suma de dos vectores: v3 = v1 + v2
3 Para compilar usar (-lrt: real time library):
4 gcc -O2 SumaVectores.c -o SumaVectores -lrt
5 gcc -O2 -S SumaVectores.c -lrt //para generar el código ensamblador
6 Para ejecutar use: SumaVectoresC longitud
7 */
8
9 #include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
10 #include <stdio.h> // biblioteca donde se encuentra la función printf()
11 #include <time.h> // biblioteca donde se encuentra la función clock_gettime()
12
13 #ifdef _OPENMP
14     #include <omp.h>
15 #else
16     #define omp_get_thread_num() 0
17     #define omp_get_num_threads() 1
18 #endif
19
20 // #define PRINTF_ALL // comentar para quitar el printf ...
21 // que imprime todos los componentes
22
23 // Sólo puede estar definida una de las tres constantes VECTOR_ (sólo uno de los ...
24 // tres defines siguientes puede estar descomentado):
25 // #define VECTOR_LOCAL // descomentar para que los vectores sean variables ...
26 // locales (si se supera el tamaño de la pila se ...
27 // generará el error "Violación de Segmento")
28 #define VECTOR_GLOBAL // descomentar para que los vectores sean variables ...
29 // globales (su longitud no estará limitada por el ...
30 // tamaño de la pila del programa)
31 // #define VECTOR_DYNAMIC // descomentar para que los vectores sean variables ...
32 // dinámicas (memoria reutilizable durante la ejecución)
33
34 #ifdef VECTOR_GLOBAL
35 #define MAX 33554432 // = 2^25
36 // #define MAX 4294967295 // = (2^32) - 1
37 double v1[MAX], v2[MAX], v3[MAX];
38 #endif
39
40 int main(int argc, char** argv){
41     int i;
42     double cgt1, cgt2;
43     // struct timespec cgt1, cgt2;
44     double ncgt; // para tiempo de ejecución
45
46     // Leer argumento de entrada (no de componentes del vector)
47     if (argc < 2){
48         printf("Faltan no componentes del vector\n");
49         exit(-1);
50     }
51 }
```

```

51 unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
52
53
54 #ifdef VECTOR_LOCAL
55 double v1[N], v2[N], v3[N]; // Tamaño variable local en tiempo de ejecución ...
56                               // disponible en C a partir de actualización C99
57 #endif
58
59 #ifdef VECTOR_GLOBAL
60 if (N>MAX) N=MAX;
61 #endif
62
63 #ifdef VECTOR_DYNAMIC
64 double *v1, *v2, *v3;
65 v1 = (double*) malloc(N*sizeof(double)); // malloc necesita el tamaño en bytes
66 v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc devuelve NULL
67 v3 = (double*) malloc(N*sizeof(double));
68
69 if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
70     printf("Error en la reserva de espacio para los vectores\n");
71     exit(-2);
72 }
73 #endif
74
75 //Inicializar vectores
76 #pragma omp parallel
77 {
78     #pragma omp for
79     for(i=0; i<N; i++){
80         v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de N
81     }
82
83     #pragma omp single
84     {
85         cgt1 = omp_get_wtime();
86     }
87     //Calcular suma de vectores
88     #pragma omp for
89     for(i=0; i<N; i++){
90         v3[i] = v1[i] + v2[i];
91     }
92
93     #pragma omp single
94     {
95         cgt2 = omp_get_wtime();
96     }
97     ncgt = cgt2-cgt1; //calculo el tiempo que ha transcurrido
98
99
100
101
102
103
104 //Imprimir resultado de la suma y el tiempo de ejecución
105 #ifdef PRINTF_ALL
106 printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
107 for(i=0; i<N; i++)
108     printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n", i,i,i,v1[i],v2[i],v3[i]);
109 #else
110 printf("Tiempo(seg.):%11.9f\n/ Tamaño Vectores:%u\n/ V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) /\n/ 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]);
111 #endif
112
113 #ifdef VECTOR_DYNAMIC
114 free(v1); // libera el espacio reservado para v1
115 free(v2); // libera el espacio reservado para v2
116 free(v3); // libera el espacio reservado para v3
117 #endif
118 return 0;
119 }
120

```

C Anchura de la pestaña: 4 Ln 121, Col 99 INS

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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


```

guillesiesta@guillesiesta: ~/Escritorio/AC/BP1
guillesiesta@guillesiesta:~$ gcc -O2 SumaVectoresC-Ej7.c -o sumaej7 -fopenmp
guillesiesta@guillesiesta:~$ ./sumaej7 8
Tiempo(seg.):0.000038613
/ Tamaño Vectores:8
/ V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) /
/ V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
guillesiesta@guillesiesta:~$ ./sumaej7 11
Tiempo(seg.):0.000006854
/ 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) /
guillesiesta@guillesiesta:~$

```

- 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: Captura que muestre el código fuente implementado

```

40 int main(int argc, char** argv){
41     int i;
42     double cgt1,cgt2;
43     double ncgt; //para tiempo de ejecución
44
45     //Leer argumento de entrada (no de componentes del vector)
46     if (argc<2){
47         printf("Faltan no componentes del vector\n");
48         exit(-1);
49     }
50
51     unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
52
53     #ifdef VECTOR_LOCAL
54     double v1[N], v2[N], v3[N]; // Tamaño variable local en tiempo de ejecución ...
55                                 // disponible en C a partir de actualización C99
56     #endif
57
58     #ifdef VECTOR_GLOBAL
59     if (N>MAX) N=MAX;
60     #endif
61
62     #ifdef VECTOR_DYNAMIC
63     double *v1, *v2, *v3;
64     v1 = (double*) malloc(N*sizeof(double)); // malloc necesita el tamaño en bytes
65     v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc devuelve NULL
66     v3 = (double*) malloc(N*sizeof(double));
67
68     if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
69         printf("Error en la reserva de espacio para los vectores\n");
70         exit(-2);
71     }
72     #endif
73

```



```

74 //Inicializar vectores
75 #pragma omp parallel private(i)
76 {
77     #pragma omp sections //divido el codigo en 4 secciones
78     {
79
80         #pragma omp section // de 0 a la cuarta parte
81         {
82             for(i=0; i<N/4; i++){
83                 v1[i] = N*0.1+i*0.1;v2[i] = N*0.1-i*0.1; //los valores dependen de N
84             }
85         }
86
87         #pragma omp section //de la cuarta parte hasta la mitad
88         {
89             for(i=N/4; i<N/2; i++){
90                 v1[i] = N*0.1+i*0.1;v2[i] = N*0.1-i*0.1; //los valores dependen de N
91             }
92         }
93
94         #pragma omp section //de la mitad hasta un cuarto mas de la mitad
95         {
96             for(i=N/2; i<3*N/4; i++){
97                 v1[i] = N*0.1+i*0.1;v2[i] = N*0.1-i*0.1; //los valores dependen de N
98             }
99         }
100
101         #pragma omp section //desde un cuarto mas de la mitad hasta el final
102         {
103             for(i=3*N/4; i<N; i++){
104                 v1[i] = N*0.1+i*0.1;v2[i] = N*0.1-i*0.1; //los valores dependen de N
105             }
106         }
107     }
108 } //fin del omp sections
109
110
111 #pragma omp single
112 {
113     cgt1 = omp_get_wtime();
114 }
115
116

```

```

117 //Calcular suma de vectores
118 #pragma omp sections
119 {
120     //Uso la misma division del vector descrita anteriormente
121     #pragma omp section
122     for(i=0; i<N/4; i++){
123         v3[i] = v1[i] + v2[i];
124     }
125
126     #pragma omp section
127     for(i=N/4; i<N/2; i++){
128         v3[i] = v1[i] + v2[i];
129     }
130
131     #pragma omp section
132     for(i=N/2; i<3*N/4; i++){
133         v3[i] = v1[i] + v2[i];
134     }
135
136     #pragma omp section
137     for(i=3*N/4; i<N; i++){
138         v3[i] = v1[i] + v2[i];
139     }
140 } //fin del omp sections
141
142 #pragma omp single
143 {
144     cgt2 = omp_get_wtime();
145 }
146
147 } //fin del omp parallel private(i)
148
149 ncgt = cgt2-cgt1;
150
151 //Imprimir resultado de la suma y el tiempo de ejecución
152 #ifdef PRINTF_ALL
153 printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
154 for(i=0; i<N; i++)
155     printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n", i,i,i,v1[i],v2[i],v3[i]);
156
157 #else
158 printf("Tiempo(seg.):%11.9f\n / Tamaño Vectores:%u\n / V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) /\n",
159        ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
160 #endif
161
162 #ifdef VECTOR_DYNAMIC
163 free(v1); // libera el espacio reservado para v1
164 free(v2); // libera el espacio reservado para v2
165 free(v3); // libera el espacio reservado para v3
166 #endif
167 return 0;
168 }

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
guillesiesta@guillesiesta:~$ gcc -O2 SumaVectoresCEj8.c -o sumaej8 -fopenmp
guillesiesta@guillesiesta:~$ ./sumaej8 8
Tiempo(seg.):0.000013017
/ Tamaño Vectores:8
/ V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) /
/ V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
guillesiesta@guillesiesta:~$ ./sumaej8 11
Tiempo(seg.):0.000037675
/ 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) /
guillesiesta@guillesiesta:~$
```

- 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: En ninguno de los ejercicios he especificado el número de cores a usar, es decir, no he usado la variable de entorno OMP_NUM_THREADS. Por lo que se usarán todos los cores que tenga el computador. En el ejercicio 8 he dividido el código en 4 secciones, por lo que al ejecutarlo intuyo que habrá threads que no trabajen, al tener mi ordenador 8 núcleos, es posible que habiéndolo dividido en 8 secciones, ningún thread esté ocioso.

- 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. Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute código que imprima todos los componentes del resultado cuando este número sea elevado.

RESPUESTA:

TIEMPO EN MI PC LOCAL

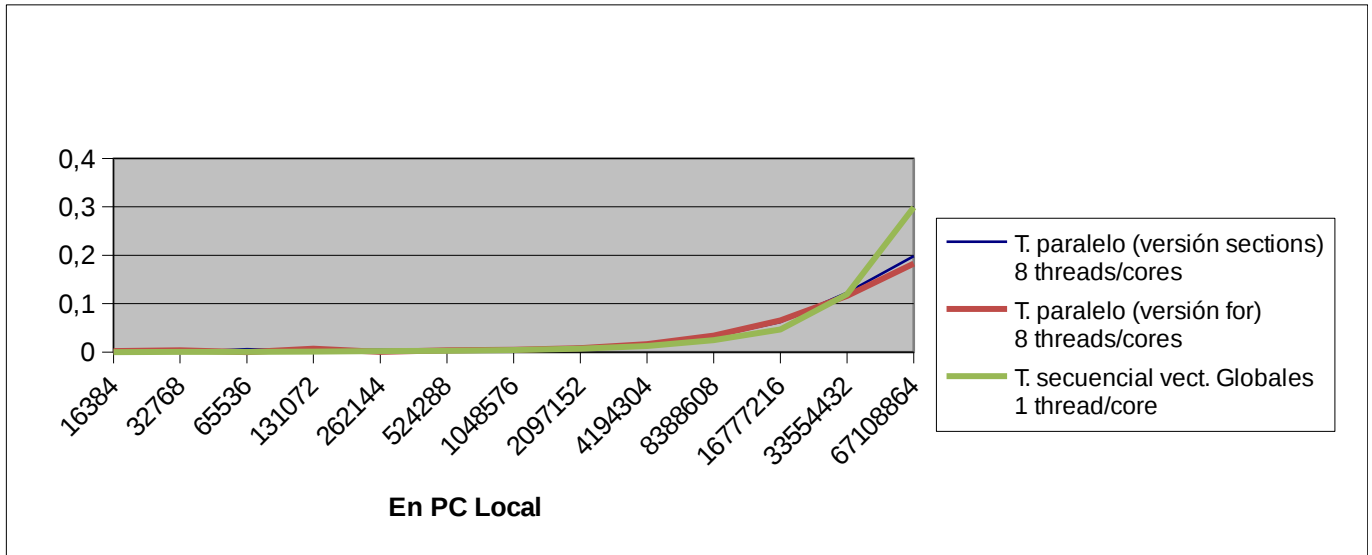
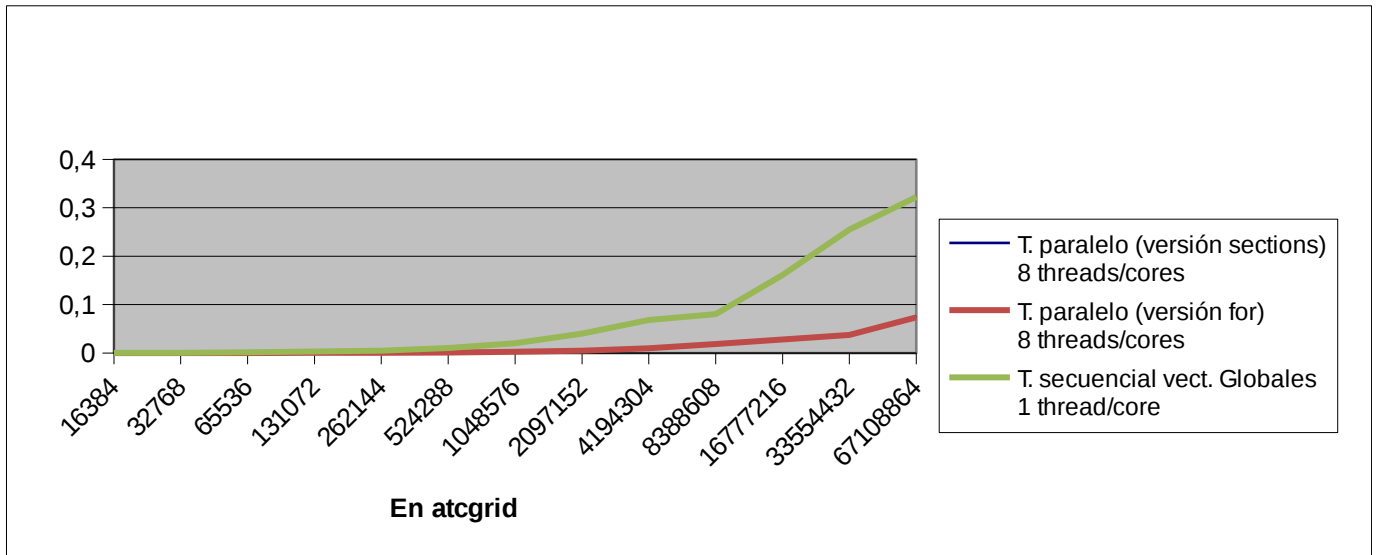


Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla “¿?” por el número de threads utilizados, que debe coincidir con el número de cores físicos utilizados.

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 2threads/4cores	T. paralelo (versión sections) 2threads/8cores
16384	0.000102853	0.002236595	0.000061381
32768	0.000211663	0.003851543	0.000079682
65536	0.000415705	0.000111469	0.006066501
131072	0.000892690	0.006801332	0.002802002
262144	0.001830221	0.000465347	0.004581504
524288	0.002655319	0.003902073	0.002289261
1048576	0.004403539	0.004644754	0.005638796
2097152	0.007212331	0.008258130	0.010035782
4194304	0.012702194	0.016182887	0.015627361
8388608	0.024585486	0.034161870	0.032376483
16777216	0.046468380	0.065212488	0.062109770
33554432	0.119703077	0.115841070	0.122699798
67108864	0.299096512	0.182640647	0.198413871

Tiempo para atcgrid



Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 8 threads/cores	T. paralelo (versión sections) 8 threads/cores
16384	0.000384565	0.000099245	0.000079160
32768	0.000558849	0.000092828	0.000198332
65536	0.001618834	0.000109775	0.001940106
131072	0.002975974	0.000199119	0.000390546
262144	0.005031271	0.000379909	0.000919906
524288	0.010184400	0.000917650	0.002587858
1048576	0.020127374	0.002620294	0.003748913
2097152	0.039815546	0.005034747	0.005055267
4194304	0.067845984	0.009562334	0.009574408
8388608	0.080375195	0.018589062	0.018893468
16777216	0.161253386	0.027895122	0.029545417
33554432	0.254574136	0.037549197	0.037309025
67108864	0.321929547	0.073428125	0.075909540

- 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:

El tiempo de CPU usado en el programa secuencial es igual al tiempo real, ya que sólo se usa un procesador. En la versión paralela, sin embargo, el tiempo de CPU es mayor que el tiempo real. Esto se debe a que el tiempo de CPU es igual a la suma de los tiempos de cada núcleo, mientras que el tiempo real es el tiempo que realmente ha tardado el programa en su ejecución.

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	real 0m0.005s	user 0m0.000s	sys 0m0.005s	real 0m0.005s	user 0m0.012s	sys 0m0.000s
131072	real 0m0.006s	user 0m0.003s	sys 0m0.003s	real 0m0.010s	user 0m0.027s	sys 0m0.000s
262144	real 0m0.005s	user 0m0.005s	sys 0m0.000s	real 0m0.008s	user 0m0.014s	sys 0m0.010s
524288	real 0m0.008s	user 0m0.000s	sys 0m0.008s	real 0m0.009s	user 0m0.021s	sys 0m0.004s
1048576	real 0m0.016s	user 0m0.004s	sys 0m0.012s	real 0m0.014s	user 0m0.038s	sys 0m0.004s
2097152	real 0m0.031s	user 0m0.015s	sys 0m0.016s	real 0m0.020s	user 0m0.026s	sys 0m0.037s
4194304	real 0m0.057s	user 0m0.032s	sys 0m0.025s	real 0m0.034s	user 0m0.072s	sys 0m0.038s
8388608	real 0m0.111s	user 0m0.047s	sys 0m0.063s	real 0m0.062s	user 0m0.115s	sys 0m0.092s
16777216	real 0m0.220s	user 0m0.080s	sys 0m0.140s	real 0m0.122s	user 0m0.202s	sys 0m0.194s
33554432	real 0m0.432s	user 0m0.206s	sys 0m0.223s	real 0m0.232s	user 0m0.386s	sys 0m0.378s
67108864	real 0m0.873s	user 0m0.428s	sys 0m0.441s	real 0m0.466s	user 0m0.670s	sys 0m0.825s