

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): Javier Galera Garrido

Grupo de prácticas: B3

Fecha de entrega:

Fecha evaluación en clase:

[-RECORDATORIO, quitar todo este texto en rojo del cuaderno definitivo-

1. COMENTARIOS

1) Este cuaderno de prácticas se utilizará para asignarle una puntuación durante la evaluación continua de prácticas y también lo utilizará como material de estudio y repaso para preparar el examen de prácticas escrito. Luego redáctelo con cuidado, y sea ordenado y claro.

2) No use máquinas virtuales. Se piden obtener resultados en atcgrid y en su PC (PC del aula o PC personal).

3) Debe modificar el prompt en los computadores que utilice en prácticas para que aparezca su nombre y apellidos, su usuario (\u), el computador (\h), el directorio de trabajo del bloque práctico (\w), la fecha (\D) completa (%F) y el día (%A) . Para modificar el prompt utilice lo siguiente (si es necesario, use export delante):

```
PS1="[NombreApellidos \u@\h:\w] \D{%F %A}\n$"
```

donde NombreApellidos es su nombre seguido de sus apellidos, por ejemplo: Juan Ortuño Vilariño

2. NORMAS SOBRE EL USO DE LA PLANTILLA

1) Usar **interlineado SENCILLO**.

2) Respetar los tipos de letra y tamaños indicados:

- Calibri-11 o Liberation Serif-11 para el texto

- **Courier New-10 o Liberation Mono-10 para nombres de fichero, comandos, variables de entorno, etc., cuando se usan en el texto.**

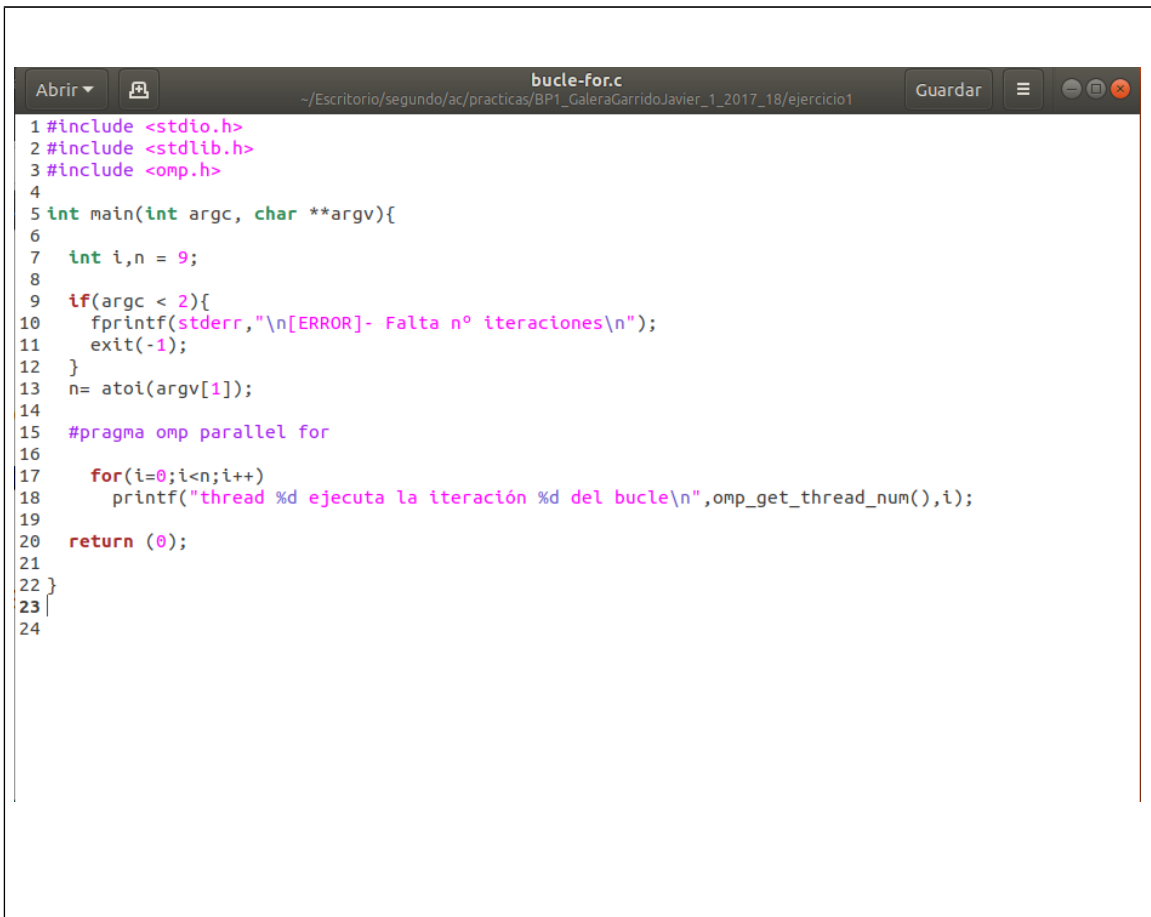
3) Insertar las capturas de pantalla donde se pidan y donde se considere oportuno. En particular, los listados de código se deben insertar como capturas de pantalla. En todas las capturas de pantalla, incluidas las de los listados de código, debe aparecer el directorio y usuario. El tamaño de letra en las capturas debe ser similar al tamaño que se está usando en el texto.

Recuerde que debe adjuntar al zip de entrega, el pdf de este fichero, todos los ficheros con código fuente implementados/utilizados y el resto de ficheros que haya implementado/utilizado (scripts, hojas de cálculo, etc.)]

1. 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`

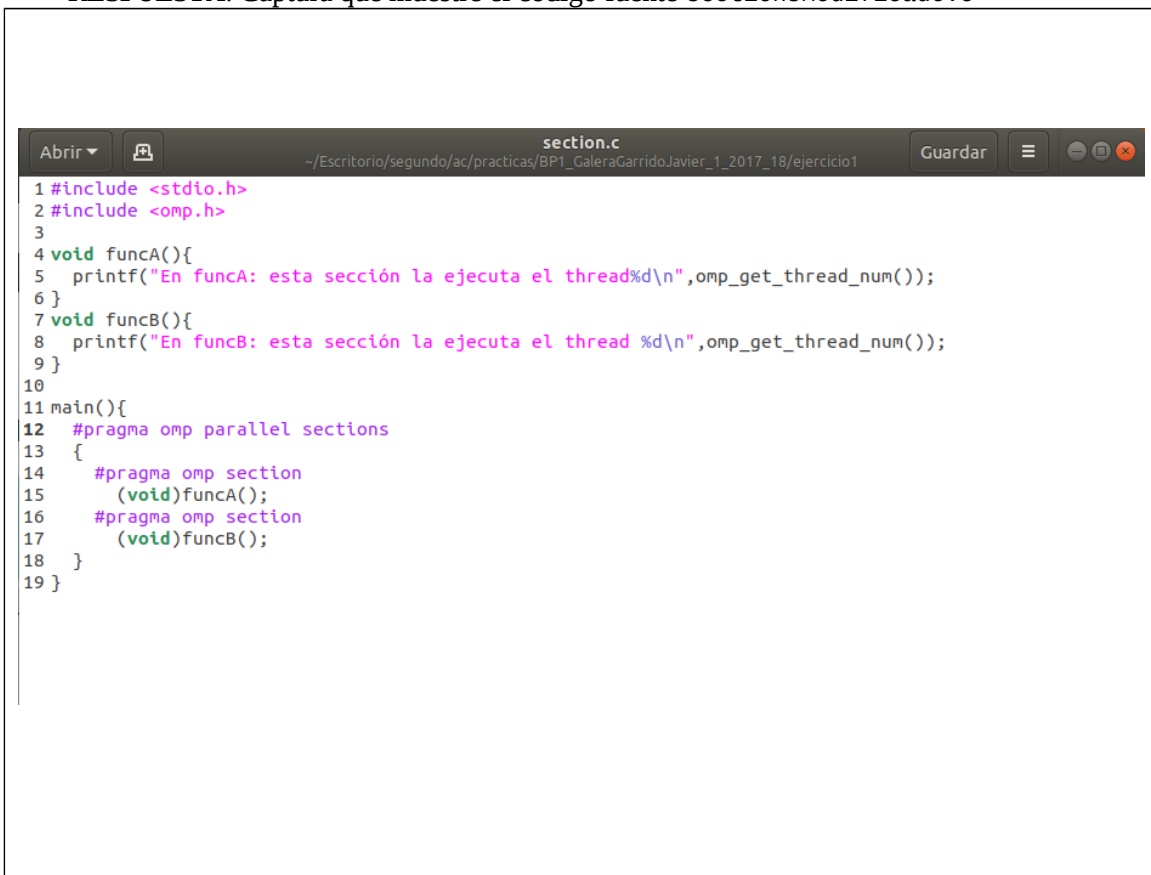


```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <omp.h>
4
5 int main(int argc, char **argv){
6
7     int i,n = 9;
8
9     if(argc < 2){
10         fprintf(stderr, "\n[ERROR]- Falta nº iteraciones\n");
11         exit(-1);
12     }
13     n= atoi(argv[1]);
14
15     #pragma omp parallel for
16
17     for(i=0;i<n;i++)
18         printf("thread %d ejecuta la iteración %d del bucle\n",omp_get_thread_num(),i);
19
20     return (0);
21 }
22
23
24

```

RESPUESTA: Captura que muestre el código fuente sectionsModificado.c



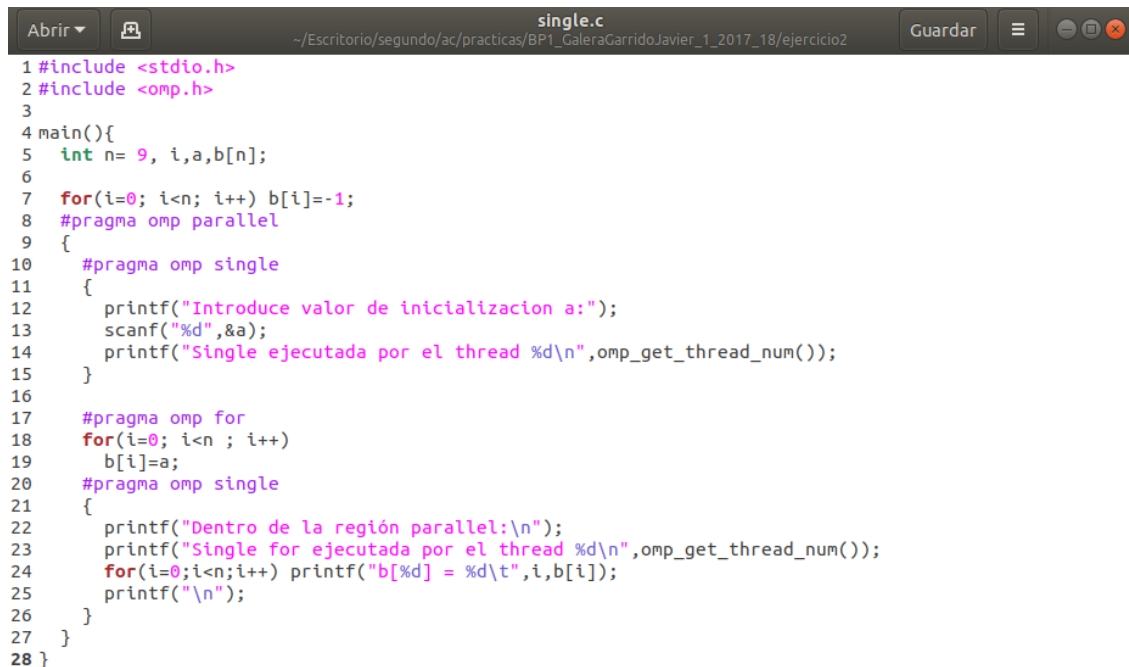
```

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 void funcB(){
8     printf("En funcB: esta sección la ejecuta el thread %d\n",omp_get_thread_num());
9 }
10
11 main(){
12     #pragma omp parallel sections
13     {
14         #pragma omp section
15         (void)funcA();
16         #pragma omp section
17         (void)funcB();
18     }
19 }

```

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
4 main(){
5     int n= 9, i,a,b[n];
6
7     for(i=0; i<n; i++) b[i]=-1;
8     #pragma omp parallel
9     {
10         #pragma omp single
11         {
12             printf("Introduce valor de inicializacion 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         #pragma omp single
21         {
22             printf("Dentro de la región parallel:\n");
23             printf("Single for ejecutada por el thread %d\n",omp_get_thread_num());
24             for(i=0;i<n;i++) printf("b[%d] = %d\t",i,b[i]);
25             printf("\n");
26         }
27     }
28 }
```

CAPTURAS DE PANTALLA:

```

~/Escritorio/segundo/ac/practicas/BP1_GaleraGarridoJavier_1_2017_18/ejercicio2 @ 12:51:06
$ ./single
Introduce valor de inicializacion a:5
Single ejecutada por el thread 2
Dentro de la región parallel:
Single for ejecutada por el thread 3
b[0] = 5    b[1] = 5    b[2] = 5    b[3] = 5    b[4] = 5    b[5] = 5    b[6] = 5    b[7] = 5    b[8] = 5
~/Escritorio/segundo/ac/practicas/BP1_GaleraGarridoJavier_1_2017_18/ejercicio2 @ 12:51:17
$

```

- 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`

```

~/Escritorio/segundo/ac/practicas/BP1_GaleraGarridoJavier_1_2017_18 — Atom
Help
Telemetry Consent x single.c — ejercicio2 single.c — ejercicio3
1 #include <stdio.h>
2 #include <omp.h>
3
4 main(){
5     int n= 9, i,a,b[n];
6
7     for(i=0; i<n; i++) b[i]=-1;
8     #pragma omp parallel
9     {
10        #pragma omp single
11        {
12            printf("Introduce valor de inicializacion 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        #pragma omp master
21        {
22            printf("Dentro de la región parallel:\n");
23            printf("Single for ejecutada por el thread %d\n",omp_get_thread_num());
24            for(i=0;i<n;i++) printf("b[%d] = %d\t",i,b[i]);
25            printf("\n");
26        }
27    }
28 }
29

```

CAPTURAS DE PANTALLA:

```

~/Escritorio/segundo/ac/practicas/BP1_GaleraGarridoJavier_1_2017_18/ejercicio3 @ 13:09:27
$ gcc -O2 -fopenmp single.c -o single_master
single.c:4:1: warning: return type defaults to 'int' [-Wimplicit-int]
main(){
^
~/Escritorio/segundo/ac/practicas/BP1_GaleraGarridoJavier_1_2017_18/ejercicio3 @ 13:09:29
$ ls
single.c  single_master
~/Escritorio/segundo/ac/practicas/BP1_GaleraGarridoJavier_1_2017_18/ejercicio3 @ 13:09:31
$ ./single_master
Introduce valor de inicializacion a:3
Single ejecutada por el thread 0
Dentro de la región parallel:
Single for ejecutada por el thread 0
b[0] = 3  b[1] = 3  b[2] = 3  b[3] = 3  b[4] = 3  b[5] = 3  b[6] = 3  b[7] = 3  b[8] = 3
~/Escritorio/segundo/ac/practicas/BP1_GaleraGarridoJavier_1_2017_18/ejercicio3 @ 13:09:39
$
+ Console
ejercicio3 JotaGalera@jota-Erazer-P6679-MD60359: jota@jota-Erazer-P6679-MD60359: ~/Escritorio/segundo/ac/practicas/BP1_GaleraGarridoJavier_1_2017_18/ejercicio3:~/Escritorio/segundo/ac/practicas/BP1_

```

RESPUESTA A LA PREGUNTA:

En el caso anterior solo se ejecutaba una vez, pero podía entrar cualquier hebra en condición de la carrera, en el primer ejemplo entró la 3, en el caso de la master, ÚNICAMENTE puede entrar la hebra Master, siendo esta la “0”, por eso por más que ejecutemos el programa `single_master`, una y otra vez siempre obtenemos que el bucle `for` ha sido realizado por la misma hebra.

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:

Esto es debido a que las anteriores hebras tienen cada una un calculo independiente de las otras, por tanto, si no esperamos a que todas terminen sus calculos para hacer la suma total, esto podría ocasionar una suma realizada equivocadamente en lugar de la que debería de mostrar como resultado.

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:

```

[B3estudiante21@atcgrid ejercicio5]$ time ./listado1 10000000
Tiempo(seg.):0.189290566 / Tamaño Vectores:10000000 /
V1[0]+V2[0]=V3[0] (1000000.000000+1000000.000000=2000000.000000) / /
V1[9999999]+V2[9999999]=V3[9999999] (1999999.900000+0.100000=2000000.000000) /

real    0m0,494s
user    0m0,258s
sys      0m0,233s
[B3estudiante21@atcgrid ejercicio5]$ time ./listado1 10000000
Tiempo(seg.):0.188282048 / Tamaño Vectores:10000000 /
V1[0]+V2[0]=V3[0] (1000000.000000+1000000.000000=2000000.000000) / /
V1[9999999]+V2[9999999]=V3[9999999] (1999999.900000+0.100000=2000000.000000) /

real    0m0,496s
user    0m0,240s
sys      0m0,253s
[B3estudiante21@atcgrid ejercicio5]$ time ./listado1 10000000
Tiempo(seg.):0.190795521 / Tamaño Vectores:10000000 /
V1[0]+V2[0]=V3[0] (1000000.000000+1000000.000000=2000000.000000) / /
V1[9999999]+V2[9999999]=V3[9999999] (1999999.900000+0.100000=2000000.000000) /

real    0m0,496s
user    0m0,235s
sys      0m0,259s
[B3estudiante21@atcgrid ejercicio5]$

```

En ningún caso CPU user + sys > real(elapsed), esto es debido a que el tiempo real(elapsed) es aquel asociado a las esperas por Entrada/Salida o a las ejecuciones 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:

```
[B3estudiante21@atcgrid ejercicio5]$ time ./listado1 10
Tiempo(seg.):0.000007263 / 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) /

real    0m0,003s
user    0m0,000s
sys      0m0,002s
[B3estudiante21@atcgrid ejercicio5]$ time ./listado1 10000000
Tiempo(seg.):0.182278449 / Tamaño Vectores:10000000 /
V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) / /
V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /

real    0m0,490s
user    0m0,255s
sys      0m0,232s
[B3estudiante21@atcgrid ejercicio5]$
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Formula de MIPS: $NI / (T_{cpu} * 10^6)$

Formula de MFLOPS: $N_{operaciones} / (T_{cpu} * 10^6)$

Los tiempos los obtenemos de:

Por tanto:

$MIPS(10) \rightarrow (7+6*10) / 0.000007263 * 10^6 = 67 / 7,263 = 9,2248MIPS \Rightarrow$ Siendo 7 las instrucciones fuera del bucle, 6 las que se repiten 10 veces, el número de veces que se repetirán las instrucciones.

$MIPSS(10000000) \rightarrow (7+6*10000000) / 0,182278449 * 10^6 = 60000007 / 182278,449 = 329,166762879MIPS.$

En el caso de los FLOPS, miramos el número de operaciones en coma flotante en un rango de tiempo, como podemos observar en nuestro código ensamblador hay 3 en el bucle, por tanto:

$MFLOPS(10) \rightarrow (3*10) / 0.000007263 * 10^6 = 30 / 7,263 = 0,02149 MFLOPS.$

$MFLOPS(10000000) \rightarrow (3*10000000) / 0,182278449 * 10^6 = 30000000 / 182278,449 = 164,583362238 MFLOPS.$

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

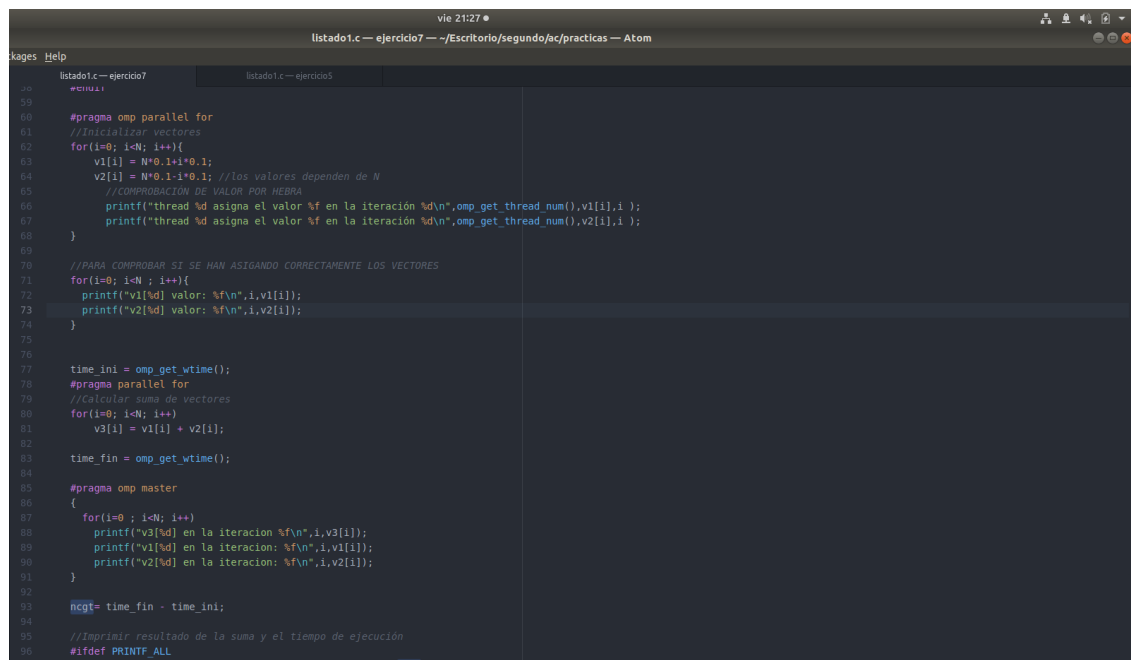
```

Listado1:
Listado1:
96 addsd %xmm0, %xmm2
97 subss %xmm0, %xmm7
98 movsd %xmm2, 0(%rbp,%rax,8)
99 movsd %xmm7, (%r12,%rax,8)
100 addq $1, %rax
101 cmpq %rax, %rbx
102 jne .L8
103 movq %rsp, %rsi
104 xorl %edi, %edi
105 salq $3, %rbx
106 call clock_gettime@PLT
107 xorl %eax, %eax
108 .p2align 4,.l0
109 .p2align 3
110 .L9:
111 movsd 0(%rbp,%rax), %xmm0
112 addsd (%r12,%rax), %xmm0
113 movsd %xmm0, (%r14,%rax)
114 addq $8, %rax
115 cmpq %rax, %rbx
116 jne .L9
117 .L10:
118 leaq 16(%rsp), %rsi
119 xorl %edi, %edi
120 call clock_gettime@PLT
121 movq 24(%rsp), %rax
122 subq 8(%rsp), %rax
123 movl %r15d, %ecx
124 pxor %xmm0, %xmm0
125 leaq .LC4(%rip), %rsi
126 pxor %xmm1, %xmm1
127 movl %r15d, %ecx
128 movsd (%r14,%rdx,8), %xmm6
129 movl %r15d, %r9d
130 cvtsi2sdq %rax, %xmm0
131 movq 16(%rsp), %rax
132 subq (%rsp), %rax
133 movsd (%r12,%rdx,8), %xmm5
134 movsd 0(%rbp,%rdx,8), %xmm4

```

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



```
vi@ 21:27 ●
listado1.c — ejercicio7 — /Escritorio/segundo/ac/practicas — Atom

kages Help

listado1.c — ejercicio7
listado1.c — ejercicio5

20 #include
21
22 #pragma omp parallel for
23 //Inicializar vectores
24 for(i=0; i<N; i++){
25     v1[i] = N*0.1+i*0.1;
26     v2[i] = N*0.1-i*0.1; //los valores dependen de N
27 //COMPROBACION DE VALOR POR HEBRA
28 printf("thread %d asigna el valor %f en la iteración %d\n",omp_get_thread_num(),v1[i],i );
29 printf("thread %d asigna el valor %f en la iteración %d\n",omp_get_thread_num(),v2[i],i );
30 }
31
32 //PARA COMPROBAR SI SE HAN ASIGNADO CORRECTAMENTE LOS VECTORES
33 for(i=0; i<N; i++){
34     printf("v1[%d] valor: %f\n",i,v1[i]);
35     printf("v2[%d] valor: %f\n",i,v2[i]);
36 }
37
38
39 time_ini = omp_get_wtime();
40 #pragma parallel for
41 //Calcular suma de vectores
42 for(i=0; i<N; i++)
43     v3[i] = v1[i] + v2[i];
44
45 time_fin = omp_get_wtime();
46
47 #pragma omp master
48 {
49     for(i=0; i<N; i++)
50         printf("v3[%d] en la iteracion %f\n",i,v3[i]);
51         printf("v1[%d] en la iteracion: %f\n",i,v1[i]);
52         printf("v2[%d] en la iteracion: %f\n",i,v2[i]);
53     }
54
55 ncgt= time_fin - time_ini;
56
57 //Imprimir resultado de la suma y el tiempo de ejecución
58 #ifdef PRINTF_ALL
```

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

```

des
v1[3] valor: 1.100000
v2[3] valor: 0.500000
v1[4] valor: 1.200000
v2[4] valor: 0.400000
v1[5] valor: 1.300000
v2[5] valor: 0.300000
v1[6] valor: 1.400000
v2[6] valor: 0.200000
v1[7] valor: 1.500000
v2[7] valor: 0.100000
v3[0] en la iteración 1.600000
v3[1] en la iteración 1.600000
v3[2] en la iteración 1.600000
v3[3] en la iteración 1.600000
v3[4] en la iteración 1.600000
v3[5] en la iteración 1.600000
v3[6] en la iteración 1.600000
v3[7] en la iteración 1.600000
v1[8] en la iteración: 0.000000
v2[8] en la iteración: 0.000000
Tiempo(seg.):0.000003565 / 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)
~/Escritorio/segundo/ac/practicas/BP1_GaleraGarridoJavier_1_2017_18/ejercicio7 @ 21:28:07
$
+ Consola Consola número 2 Consola número 3
ejercicio7 JotaGalera@jota-Erazer-P6679-MD60359:~/Escritorio/segundo/ac/practicas/BP1_GaleraGarridoJavier_1_2017_18/ejercicio7:~/Escritorio/segundo/ac/practicas/BP1

```

·N = 11

```

des
v2[7] valor: 0.400000
v1[8] valor: 1.900000
v2[8] valor: 0.300000
v1[9] valor: 2.000000
v2[9] valor: 0.200000
v1[10] valor: 2.100000
v2[10] valor: 0.100000
v3[0] en la iteración 2.200000
v3[1] en la iteración 2.200000
v3[2] en la iteración 2.200000
v3[3] en la iteración 2.200000
v3[4] en la iteración 2.200000
v3[5] en la iteración 2.200000
v3[6] en la iteración 2.200000
v3[7] en la iteración 2.200000
v3[8] en la iteración 2.200000
v3[9] en la iteración 2.200000
v3[10] en la iteración 2.200000
v1[11] en la iteración: 0.000000
v2[11] en la iteración: 0.000000
Tiempo(seg.):0.000016632 / 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)
~/Escritorio/segundo/ac/practicas/BP1_GaleraGarridoJavier_1_2017_18/ejercicio7 @ 21:29:16
$
+ Consola Consola número 2 Consola número 3
ejercicio7 JotaGalera@jota-Erazer-P6679-MD60359:~/Escritorio/segundo/ac/practicas/BP1_GaleraGarridoJavier_1_2017_18/ejercicio7:~/Escritorio/segundo/ac/practicas/BP1

```

- 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

```

116         printf("v2[%d] valor: %f\n", i, v2[i]);
117     }
118     */
119
120     time_ini = omp_get_wtime();
121     #pragma omp parallel sections
122     {
123         //Calcular suma de vectores primer cuarto
124         #pragma omp section
125         {
126             for(i=0; i<N/4; i++)
127                 v3[i] = v1[i] + v2[i];
128         }
129         //Calcular suma de vectores segundo cuarto
130         #pragma omp section
131         {
132             for(k=N/4; k<N/2; k++)
133                 v3[k] = v1[k] + v2[k];
134         }
135         //Calcular suma de vectores tercer cuarto
136         #pragma omp section
137         {
138             for(t=N/2; t<(N*3)/4; t++)
139                 v3[t] = v1[t] + v2[t];
140         }
141         //Calcular suma de vectores cuarto cuarto
142         #pragma omp section
143         {
144             for(j=(N*3)/4; j<N; j++)
145                 v3[j] = v1[j] + v2[j];
146         }
147     }
148     time_fin = omp_get_wtime();
149
150     #pragma omp master
151     {
152         for(i=0; i<N; i++)
153             printf("v3[%d] vale: %f\n", i, v3[i]);
154         printf("v1[%d] primer valor: %f\n", 0, v1[0]);
155         printf("v1[%d] ultimo valor: %f\n", N-1, v1[N-1]);
156         printf("v2[%d] primer valor: %f\n", 0, v2[0]);
157         printf("v2[%d] ultimo valor: %f\n", N-1, v2[N-1]);
158     }
159

```

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

·N = 8

```

thread 3 asigna el valor 1.480000 en la iteración 6
thread 3 asigna el valor 0.200000 en la iteración 6
thread 3 asigna el valor 1.580000 en la iteración 7
thread 3 asigna el valor 0.180000 en la iteración 7
thread 2 asigna el valor 1.200000 en la iteración 4
thread 2 asigna el valor 0.480000 en la iteración 4
thread 2 asigna el valor 1.300000 en la iteración 5
thread 2 asigna el valor 0.300000 en la iteración 5
v3[0] vale: 1.680000
v3[1] vale: 1.680000
v3[2] vale: 1.680000
v3[3] vale: 1.680000
v3[4] vale: 1.680000
v3[5] vale: 1.680000
v3[6] vale: 1.680000
v3[7] vale: 1.680000
v1[0] primer valor: 0.800000
v1[7] ultimo valor: 1.580000
v2[0] primer valor: 0.800000
v2[7] ultimo valor: 0.100000
Tiempo(seg.): 0.000004651 / Tamaño Vectores: 8 /
v1[0]+v2[0]=v3[0] (0.800000+0.800000=1.600000) / /
v1[7]+v2[7]=v3[7] (1.580000+0.100000=1.680000) / / 10
Depto. Arquitectura y Tecnología de Computadores
~/Escritorio/segundo/ac/practicas/BP1_GaleraGarridoJavier_1_2017_18/ejercicio8 @ 1.00.40
ejercicio8 JotaGalera@Jota-Erazer-P6679-MD60359: ~/Escritorio/segundo/ac/practicas/BP1_GaleraGarridoJavier_1_2017_18/ejercicio8:~

```

·N = 11

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 el ejercicio 7 en el cual utilizamos la directiva parallel-for, la cual nos permite variar el número de hebras, en el caso de utilizar export OMP_NUM_THREAD, nos permite variar hasta el máximo de cores que te permita tu hardware, en mi caso he utilizado OMP_NUM_THREAD=8.

Por otro lado en el ejercicio 8, al utilizar la directiva parallel-sections, solo nos permite utilizar el mismo número de hebras que de section declaremos, como he declarado 4 section en el código obtenemos como resultado 4 hebras.

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:

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 Componente s	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 12 threads/cores	T. paralelo (versión sections) 12 threads/cores
16384	0.000107815	0.000113080	0.004882389
32768	0.000211568	0.000251032	0.004269219
65536	0.000435112	0.000437535	0.004354554
131072	0.000869718	0.000992965	0.004288127
262144	0.001318850	0.001593913	0.004447784
524288	0.003334933	0.003527144	0.004457493
1048576	0.006769509	0.007090334	0.005450047
2097152	0.012801770	0.012923242	0.008051012
4194304	0.024947047	0.024656814	0.014579517
8388608	0.049792028	0.046539923	0.023312414
16777216	0.106627486	0.101805381	0.049710061
33554432	0.203772374	0.201958682	0.100101430
67108864	0.417851824	0.402734903	0.188579587

Resultados sobre atcgrid

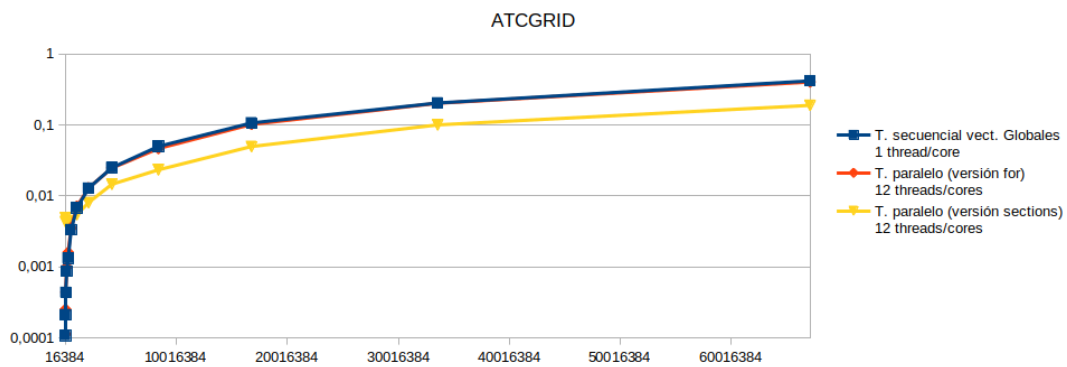
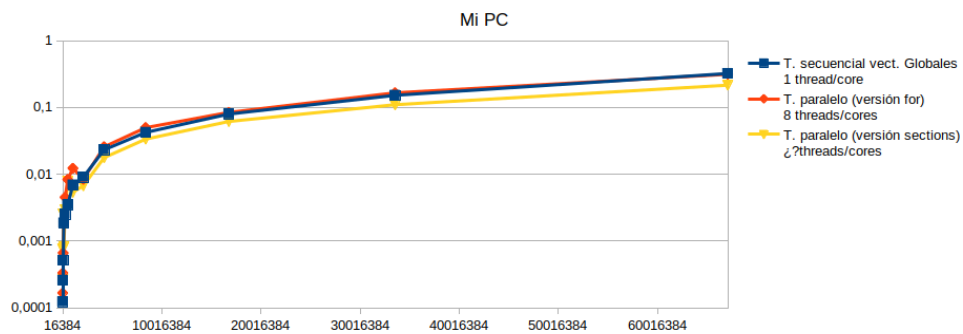


Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla “¿?” por el número de threads utilizados, que debe coincidir con el número de cores físicos utilizados.

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 8 threads/cores	T. paralelo (versión sections) 8 threads/cores
16384	0.000123426	0.000167324	0.000755097
32768	0.000259933	0.000333708	0.000266332
65536	0.000516678	0.000664120	0.002533686
131072	0.001872222	0.001903445	0.000826825
262144	0.002539430	0.004508363	0.002940374
524288	0.003486504	0.008350498	0.007847793
1048576	0.006925152	0.012309556	0.005383435
2097152	0.009086101	0.008416551	0.006737547
4194304	0.023282767	0.025827057	0.017780969
8388608	0.042436171	0.049962369	0.033666548
16777216	0.079788011	0.084608780	0.061788676
33554432	0.152388214	0.165991060	0.110177945
67108864	0.322859339	0.313994849	0.215945576

Resultados sobre mi PC



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 por lo general es menor que el tiempo real(*elapsed*) ya que el tiempo real es el tiempo de ejecución del programa incluyendo esperas de la CPU, en cambio, el tiempo de CPU es solo el tiempo que el programa se ejecuta en la CPU. Pero en algunos datos de la tabla vemos que el tiempo de CPU es mayor que el tiempo real. Esto es debido a que cuando tenemos varios flujos de control, el tiempo de CPU es la suma de todos los flujos que entran en la CPU, mientras que el tiempo real analiza solo el flujo de control principal.

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 ¿? 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	0m0,001s		real	0m0.012s	
	user	0m0,001s		user	0m0.108s	
	sys	0m0,000s		sys	0m0.002s	
131072	real	0m0,002s		real	0m0.010s	
	user	0m0,000s		user	0m0.099s	
	sys	0m0,002s		sys	0m0.019s	
262144	real	0m0,003s		real	0m0.008s	
	user	0m0,003s		user	0m0.130s	
	sys	0m0,000s		sys	0m0.003s	
524288	real	0m0,007s		real	0m0.012s	
	user	0m0,003s		user	0m0.142s	
	sys	0m0,004s		sys	0m0.006s	
1048576	real	0m0,013s		real	0m0.017s	
	user	0m0,005s		user	0m0.135s	
	sys	0m0,009s		sys	0m0.023s	
2097152	real	0m0,029s		real	0m0.026s	
	user	0m0,008s		user	0m0.147s	
	sys	0m0,020s		sys	0m0.047s	
4194304	real	0m0,045s		real	0m0.050s	
	user	0m0,021s		user	0m0.166s	
	sys	0m0,025s		sys	0m0.092s	
8388608	real	0m0,086s		real	0m0.084s	
	user	0m0,037s		user	0m0.191s	
	sys	0m0,049s		sys	0m0.217s	
16777216	real	0m0,165s		real	0m0.164s	
	user	0m0,083s		user	0m0.264s	
	sys	0m0,082s		sys	0m0.411s	
33554432	real	0m0,337s		real	0m0.310s	
	user	0m0,124s		user	0m0.403s	
	sys	0m0,212s		sys	0m0.831s	
67108864	real	0m0,650s		real	0m0.644s	
	user	0m0,280s		user	0m0.707s	
	sys	0m0,368s		sys	0m1.535s	