

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): Arturo Cortés Sánchez

Grupo de prácticas y profesor de prácticas: C2

Fecha de entrega:

Fecha evaluación en clase:

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

Ejercicios basados en los ejemplos del seminario práctico

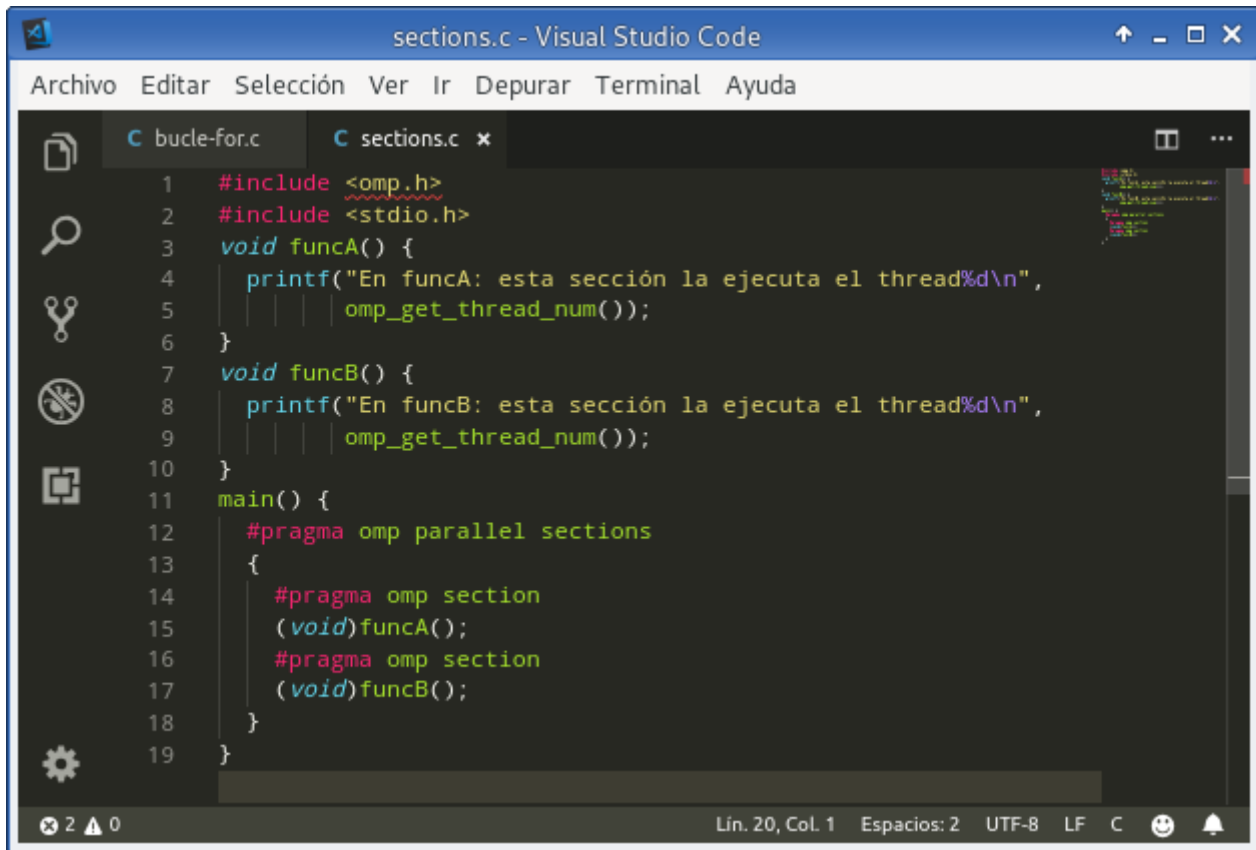
1. Usar la directiva `parallel` combinada con directivas de trabajo compartido en los ejemplos `bucle-for.c` y `sections.c` del seminario. Incorporar el código fuente resultante al cuaderno de prácticas.

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



```
1 #include <omp.h>
2 #include <stdio.h>
3 #include <stdlib.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",
15              omp_get_thread_num(), i);
16    }
17
18    return (0);
19 }
```

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



```

1  #include <omp.h>
2  #include <stdio.h>
3  void funcA() {
4      printf("En funcA: esta sección la ejecuta el thread%d\n",
5          omp_get_thread_num());
6  }
7  void funcB() {
8      printf("En funcB: esta sección la ejecuta el thread%d\n",
9          omp_get_thread_num());
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 <omp.h>
2  #include <stdio.h>
3  int main() {
4      int n = 9, i, a, b[n];
5      for (i = 0; i < n; i++)
6          b[i] = -1;
7      #pragma omp parallel
8      {
9          #pragma omp single
10         {
11             printf("Introduce valor de inicialización a : ");
12             scanf("%d", &a);
13             printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
14         }
15         #pragma omp for
16         for (i = 0; i < n; i++)
17             b[i] = a;
18
19         #pragma omp single
20         {
21             printf("Dentro de la región parallel, hebra: %d\n", omp_get_thread_num());
22             for (i = 0; i < n; i++)
23                 printf("b[%d] = %d\t", i, b[i]);
24             printf("\n");
25         }
26     }
27 }

```

CAPTURAS DE PANTALLA:

```

arturo@mior-pc: ~/Escritorio
arturo@mior-pc:~/Escritorio 136x8
[ArturoCortésSánchez arturo@mior-pc:~/Escritorio] 2019-03-13 miércoles
$ ./single
Introduce valor de inicialización a : 5
Single ejecutada por el thread 1
Dentro de la región parallel, hebra: 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
[ArturoCortésSánchez arturo@mior-pc:~/Escritorio] 2019-03-13 miércoles
$

```

- Imprimir los resultados del programa `single.c` usando una directiva `master` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `master` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `master`. Incorpore en su cuaderno el código fuente y volcados de pantalla con los resultados de ejecución obtenidos. ¿Qué diferencia observa con respecto a los resultados de ejecución del ejercicio anterior?

RESPUESTA: Captura que muestre el código fuente `singleModificado2.c`

CAPTURAS DE PANTALLA:

```

single.c - Visual Studio Code
Archivo Editar Selección Ver Ir Depurar Terminal Ayuda

1  #include <omp.h>
2  #include <stdio.h>
3  int main() {
4      int n = 9, i, a, b[n];
5      for (i = 0; i < n; i++)
6          b[i] = -1;
7      #pragma omp parallel
8      {
9          #pragma omp single
10         {
11             printf("Introduce valor de inicialización a : ");
12             scanf("%d", &a);
13             printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
14         }
15         #pragma omp for
16         for (i = 0; i < n; i++)
17             b[i] = a;
18         #pragma omp master
19         {
20             printf("Dentro de la región parallel, hebra: %d\n", omp_get_thread_num());
21             for (i = 0; i < n; i++)
22                 printf("b[%d] = %d\t", i, b[i]);
23             printf("\n");
24         }
25     }
26 }
27

```

```

arturo@mior-pc: ~/Escritorio
arturo@mior-pc:~/Escritorio 136x8
[ArturoCortésSánchez arturo@mior-pc:~/Escritorio] 2019-03-13 miércoles
$ ./single
Introduce valor de inicialización a : 4
Single ejecutada por el thread 0
Dentro de la región parallel, hebra: 0
b[0] = 4      b[1] = 4      b[2] = 4      b[3] = 4      b[4] = 4      b[5] = 4      b[6] = 4      b[7] = 4      b[8] = 4
[ArturoCortésSánchez arturo@mior-pc:~/Escritorio] 2019-03-13 miércoles
$

```

RESPUESTA A LA PREGUNTA:

Usando master la hebra asignada siempre es la 0.

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

RESPUESTA:

Porque master no implica que las hebras tengan que esperar a que todas acaben, para eso necesita un barrier, por tanto si lo quitamos puede que la suma se imprima antes de que acabe.

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:

```

thread 3 suma de a[19]=19 sumalocal=85
thread master=0 imprime suma=190
[ArturoCortésSánchez arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2] 2
$ssh C2estudiante3@atcgrid.ugr.es
C2estudiante3@atcgrid.ugr.es$ C2estudiante3@atcgrid.ugr.es's password:
Last login: Wed Mar  6 15:18:55 2019 from cv1086055.ugr.es
[C2estudiante3@atcgrid ~]$ PS1="[ArturoCortésSánchez \u@h:\w] \D{%F %A}\n$ "
[ArturoCortésSánchez C2estudiante3@atcgrid:~] 2019-03-13 miércoles
$ls
bp0
[ArturoCortésSánchez C2estudiante3@atcgrid:~] 2019-03-13 miércoles
$mkdir bp1
[ArturoCortésSánchez C2estudiante3@atcgrid:~] 2019-03-13 miércoles
$ls
bp0 bp1
[ArturoCortésSánchez C2estudiante3@atcgrid:~] 2019-03-13 miércoles
$cd bp1
[ArturoCortésSánchez C2estudiante3@atcgrid:~/bp1] 2019-03-13 miércoles
$ls
[ArturoCortésSánchez C2estudiante3@atcgrid:~/bp1] 2019-03-13 miércoles
$ls
[ArturoCortésSánchez C2estudiante3@atcgrid:~/bp1] 2019-03-13 miércoles
$ls
[ArturoCortésSánchez C2estudiante3@atcgrid:~/bp1] 2019-03-13 miércoles
$ls
SumaVectoresC
[ArturoCortésSánchez C2estudiante3@atcgrid:~/bp1] 2019-03-13 miércoles
$./SumaVectoresC 10000000
Tamaño Vectores:10000000 (4 B)
Tiempo:0.027672278 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](1000000.000000+1000
000.000000=2000000.000000) // V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000
000.000000) /
[ArturoCortésSánchez C2estudiante3@atcgrid:~/bp1] 2019-03-13 miércoles
$echo "time ./SumaVectoresC 10000000" | qsub -q ac
11320.atcgrid
[ArturoCortésSánchez C2estudiante3@atcgrid:~/bp1] 2019-03-13 miércoles
$ls
STDIN.e11320 STDIN.o11320 SumaVectoresC
[ArturoCortésSánchez C2estudiante3@atcgrid:~/bp1] 2019-03-13 miércoles
$ls -lag
total 36
drwxrwxr-x 2 C2estudiante3 4096 mar 13 17:19 .
drwx----- 9 C2estudiante3 4096 mar 13 17:17 ..
-rw----- 1 C2estudiante3 42 mar 13 17:15 STDIN.e11320
-rw----- 1 C2estudiante3 230 mar 13 17:15 STDIN.o11320
-rwxr-xr-x 1 C2estudiante3 16928 mar 13 17:18 SumaVectoresC
[ArturoCortésSánchez C2estudiante3@atcgrid:~/bp1] 2019-03-13 miércoles
$

SumaVectoresC
100% 17KB 581.8KB/s 00:00
sftp> get 5
STDIN.e11320 SumaVectoresC
sftp> get 6T
STDIN.e11320 STDIN.o11320
sftp> get STDIN.*
Fetching /home/C2estudiante3/bp1/STDIN.e11320 to STDIN.e11320
/home/C2estudiante3/bp1/STDIN.e11320 100% 42 5.2KB/s 00:00
Fetching /home/C2estudiante3/bp1/STDIN.o11320 to STDIN.o11320
/home/C2estudiante3/bp1/STDIN.o11320 100% 230 38.2KB/s 00:00
sftp>
$

arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2.95x32
[ArturoCortésSánchez arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2] 2
019-03-13 miércoles
'$Captura de pantalla 2019-03-13 16-11-44.png' ejer2.png STDIN.e11320 SumaVectoresC.c
'$Captura de pantalla_2019-03-13_16-12-42.png' ejer3-2.png STDIN.o11320
ejer2-2.png ejer3.png SumaVectoresC
[ArturoCortésSánchez arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2] 2
019-03-13 miércoles
$cat STDIN.*

real 0m0.105s
user 0m0.053s
sys 0m0.050s
Tamaño Vectores:10000000 (4 B)
Tiempo:0.042318264 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](1000000.000000+1000
000.000000=2000000.000000) // V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000
000.000000) /
[ArturoCortésSánchez arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2] 2
019-03-13 miércoles
$

```

- Generar el código ensamblador a partir del programa secuencial C del Listado 1 para **vectores globales** (para generar el código ensamblador tiene que compilar usando -s en lugar de -o). Utilice el fichero con el código fuente ensamblador generado y el fichero ejecutable generado en el ejercicio 5 para obtener para atcgrid los MIPS (*Millions of Instructions Per Second*) y los MFLOPS (*Millions of Floating-point Per Second*) del código que obtiene la suma de vectores (código entre las funciones `clock_gettime()`); el cálculo se debe hacer para 10 y 10000000 componentes en los vectores (consulte la Lección 3/Tema1 AC). Razonar cómo se han obtenido los valores que se necesitan para calcular los MIPS y MFLOPS. Incorpore **el código ensamblador de la parte de la suma de vectores** en el cuaderno.

CAPTURAS DE PANTALLA (que muestren la generación del código ensamblador y del código ejecutable, y la obtención de los tiempos de ejecución):

```

C2estudiante3@atcgriid:~/bp1/ejer6 105x46
[arturo@miroir-pc ~]$ ssh C2estudiante3@atcgriid.ugr.es
C2estudiante3@atcgriid.ugr.es's password:
Last login: Wed Mar 13 17:10:44 2019 from cv1087138.ugr.es
C2estudiante3@atcgriid ~$ cd bp1
C2estudiante3@atcgriid bp1$ mkdir ejer6
C2estudiante3@atcgriid bp1$ ls
ejer6  STDIN.e11320  STDIN.o11320  SumaVectoresC
C2estudiante3@atcgriid bp1$ cd ejer6/
C2estudiante3@atcgriid ejer6$ ls
SumaVectoresC
C2estudiante3@atcgriid ejer6$ echo "SumaVectoresC" | qsub -q ac
12588.atcgriid
C2estudiante3@atcgriid ejer6$ ls -ls
total 24
4 -rw-rw-r-- 1 C2estudiante3 97 mar 20 16:04 STDIN.e12588
0 -rw-rw-r-- 1 C2estudiante3 0 mar 20 16:04 STDIN.o12588
20 -rwxr-xr-x 1 C2estudiante3 16928 mar 20 16:04 SumaVectoresC
C2estudiante3@atcgriid ejer6$ nano STDIN.e12588
C2estudiante3@atcgriid ejer6$ echo "./SumaVectoresC" | qsub -q ac
12589.atcgriid
C2estudiante3@atcgriid ejer6$ ls -ls
total 28
4 -rw-rw-r-- 1 C2estudiante3 97 mar 20 16:04 STDIN.e12588
0 -rw-rw-r-- 1 C2estudiante3 0 mar 20 16:02 STDIN.e12589
0 -rw-rw-r-- 1 C2estudiante3 0 mar 20 16:04 STDIN.o12588
4 -rw-rw-r-- 1 C2estudiante3 35 mar 20 16:02 STDIN.o12589
20 -rwxr-xr-x 1 C2estudiante3 16928 mar 20 16:04 SumaVectoresC
C2estudiante3@atcgriid ejer6$ nano STDIN.e12589
C2estudiante3@atcgriid ejer6$ nano STDIN.o12589
C2estudiante3@atcgriid ejer6$ rm STDIN.*
C2estudiante3@atcgriid ejer6$ echo "SumaVectoresC 10" | qsub -q ac
12590.atcgriid
C2estudiante3@atcgriid ejer6$ nano STDIN.o12590
C2estudiante3@atcgriid ejer6$ nano STDIN.e12590
C2estudiante3@atcgriid ejer6$ rm STDIN.*
C2estudiante3@atcgriid ejer6$ echo "SumaVectoresC 10" | qsub -q ac
12591.atcgriid
C2estudiante3@atcgriid ejer6$ nano STDIN.e12590
C2estudiante3@atcgriid ejer6$ nano STDIN.o12590
C2estudiante3@atcgriid ejer6$ nano STDIN.e12591
C2estudiante3@atcgriid ejer6$ nano STDIN.o12591
C2estudiante3@atcgriid ejer6$ PS1="[ArturoCortesSanchez ugriid:~$]"
[ArturoCortesSanchez C2estudiante3@atcgriid:~/bp1/ejer6] 2019-03-20 miércoles
Secho "/SumaVectoresC 10000000" | qsub -q ac
12594.atcgriid
[ArturoCortesSanchez C2estudiante3@atcgriid:~/bp1/ejer6] 2019-03-20 miércoles

arturo@miroir-pc:~$ 85x11
Fetching /home/C2estudiante3/bp1/ejer6/STDIN.e12591 to STDIN.e12591
Fetching /home/C2estudiante3/bp1/ejer6/STDIN.o12591 to STDIN.o12591
/home/C2estudiante3/bp1/ejer6/STDIN.o12591 100% 170 26.9KB/s 00:00
sftp> get STDIN.*
Fetching /home/C2estudiante3/bp1/ejer6/STDIN.e12591 to STDIN.e12591
Fetching /home/C2estudiante3/bp1/ejer6/STDIN.o12591 to STDIN.o12591
/home/C2estudiante3/bp1/ejer6/STDIN.o12591 100% 170 20.1KB/s 00:00
Fetching /home/C2estudiante3/bp1/ejer6/STDIN.o12594 to STDIN.o12594
/home/C2estudiante3/bp1/ejer6/STDIN.o12594 100% 230 22.5KB/s 00:00
sftp>

arturo@miroir-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2 85x32
[arturo@miroir-pc Practica 2]$ ls
BP1_ApellidoApellidoNombre_Y.odt
Captura de pantalla 2019-03-13_16-11-44.png
Captura de pantalla 2019-03-13_16-12-42.png
ejer2-2.png
ejer2.png
ejer3-2.png
ejer3.png
ejer5.png
STDIN.e11320
STDIN.o11320
SumaVectoresC
SumaVectoresC.c
SumaVectoresC.s
[arturo@miroir-pc Practica 2]$ gcc -O2 -S SumaVectoresC.c
[arturo@miroir-pc Practica 2]$ gcc -O2 -S SumaVectoresC.c
[arturo@miroir-pc Practica 2]$ gcc -O2 SumaVectoresC.c -o SumaVectoresC
[arturo@miroir-pc Practica 2]$ ./SumaVectoresC
Faltan n6 componentes del vector
[arturo@miroir-pc Practica 2]$ ./SumaVectoresC 10
Tamaño Vectores:10 (4 B)
Tiempo:0.000005736 / 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) /
[arturo@miroir-pc Practica 2]$ ./SumaVectoresC 10000000
Tamaño Vectores:10000000 (4 B)
Tiempo:0.061390509 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](1000000.0
00000+1000000.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3[9999999](1999999.
900000+0.100000=2000000.000000) /
[arturo@miroir-pc Practica 2]$ PS1="[ArturoCortesSanchez ugriid:~$]"
[ArturoCortesSanchez arturo@miroir-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Pra
ctica 2] 2019-03-20 miércoles

```

RESPUESTA: cálculo de los MIPS y los MFLOPS

$$MIPS = \frac{\text{Numero instrucciones}}{\text{Tiempo CPU} \cdot 10^6} \rightarrow MIPS = \frac{1 + 6 \cdot N + 2}{\text{Tiempo CPU} \cdot 10^6}$$

$$MIPS = \frac{1 + 6 \cdot 10 + 2}{0,000387753 \cdot 10^6} = 0,16247456499369445$$

$$MIPS = \frac{1 + 6 \cdot 10000000 + 2}{0,040183463 \cdot 10^6} = 1493,1516230943062$$

$$MFLOPS = \frac{\text{Numero instrucciones en coma flotante}}{\text{Tiempo CPU} \cdot 10^6}$$

$$MFLOPS = \frac{3 \cdot 10}{0,000387753 \cdot 10^6} = 0,07736884047318783$$

$$MFLOPS = \frac{3 \cdot 10000000}{0,040183463 \cdot 10^6} = 746,5757742183644$$

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

```

92      call    clock_gettime@PLT
93      xorl    %eax, %eax
94      .p2align 4,,10
95      .p2align 3
96      .L6:
97      movsd   (%r12,%rax,8), %xmm0
98      addsd   0(%r13,%rax,8), %xmm0
99      movsd   %xmm0, (%r14,%rax,8)
100     addq    $1, %rax
101     cmpl    %eax, %ebp
102     ja      .L6
103     leaq    16(%rsp), %rsi
104     xorl    %edi, %edi
105     call    clock_gettime@PLT

```

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

```

70 // Inicializar vectores
71 #pragma omp parallel for
72     for (i = 0; i < N; i++) {
73         v1[i] = N * 0.1 + i * 0.1;
74         v2[i] = N * 0.1 - i * 0.1;
75     }
76     double start_time = omp_get_wtime();
77 // clock_gettime(CLOCK_REALTIME,&cgt1);
78 // Calcular suma de vectores
79 #pragma omp parallel for
80     for (i = 0; i < N; i++)
81         v3[i] = v1[i] + v2[i];
82
83 // clock_gettime(CLOCK_REALTIME,&cgt2);
84 // ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
85 //         (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
86
87 ncgt = omp_get_wtime() - start_time;
88
89 // Imprimir resultado de la suma y el tiempo de ejecución
90 if (N < 10) {
91     printf("Tiempo:%11.9f\t / Tamaño Vectores:%u\n", ncgt, N);
92     for (i = 0; i < N; i++)
93         printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n", i, i, i, v1[i],
94             v2[i], v3[i]);
95 } else
96     printf("Tiempo:%11.9f\t / Tamaño Vectores:%u\t/ "
97         "V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) / / "
98         "V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
99         ncgt, N, v1[0], v2[0], v3[0], N - 1, N - 1, N - 1, v1[N - 1],
100         v2[N - 1], v3[N - 1]);
101
102 printf("Primeros elementos de v1, v2 y v3: %f, %f, %f \n", v1[0], v2[0],
103     v3[0]);
104 printf("Últimos elementos de v1, v2 y v3: %f, %f, %f \n", v1[N - 1],
105     v2[N - 1], v3[N - 1]);

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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


```

arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2/ejer7
arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2/ejer7 165x25

[ArturoCortesSanchez arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2/ejer7] 2019-03-20 miércoles
$gcc SumaVectores_paralelo.c -o SumaVectores_paralelo -fopenmp
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2/ejer7] 2019-03-20 miércoles
$./SumaVectores_paralelo 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000019003 / Tamaño Vectores:8
/ V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) /
/ V1[1]+V2[1]=V3[1](0.900000+0.700000=1.600000) /
/ V1[2]+V2[2]=V3[2](1.000000+0.600000=1.600000) /
/ V1[3]+V2[3]=V3[3](1.100000+0.500000=1.600000) /
/ V1[4]+V2[4]=V3[4](1.200000+0.400000=1.600000) /
/ V1[5]+V2[5]=V3[5](1.300000+0.300000=1.600000) /
/ V1[6]+V2[6]=V3[6](1.400000+0.200000=1.600000) /
/ V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
Primeros elementos de v1, v2 y v3: 0.800000, 0.800000, 1.600000
Últimos elementos de v1, v2 y v3: 1.500000, 0.100000, 1.600000
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2/ejer7] 2019-03-20 miércoles
$./SumaVectores_paralelo 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000016579 / 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) /
Primeros elementos de v1, v2 y v3: 1.100000, 1.100000, 2.200000
Últimos elementos de v1, v2 y v3: 2.100000, 0.100000, 2.200000
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2/ejer7] 2019-03-20 miércoles
$

```

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, $N = 8$); (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

```

70 // Inicializar vectores
71 #pragma omp parallel sections private(i)
72 {
73     #pragma omp section
74     {
75         for (i = 0; i < N / 4; i++) {
76             v1[i] = N * 0.1 + i * 0.1;
77             v2[i] = N * 0.1 - i * 0.1;
78         }
79     }
80     #pragma omp section
81     {
82         for (i = N / 4; i < N / 2; i++) {
83             v1[i] = N * 0.1 + i * 0.1;
84             v2[i] = N * 0.1 - i * 0.1;
85         }
86     }
87     #pragma omp section
88     {
89         for (i = N / 2; i < 3 * N / 4; i++) {
90             v1[i] = N * 0.1 + i * 0.1;
91             v2[i] = N * 0.1 - i * 0.1;
92         }
93     }
94     #pragma omp section
95     {
96         for (i = 3 * N / 4; i < N; i++) {
97             v1[i] = N * 0.1 + i * 0.1;
98             v2[i] = N * 0.1 - i * 0.1;
99         }
100     }
101 }

```

```

109 #pragma omp parallel sections private(i)
110 {
111     #pragma omp section
112     {
113         for (i = 0; i < N / 4; i++) {
114             v3[i] = v1[i] + v2[i];
115         }
116     }
117     #pragma omp section
118     {
119         for (i = N / 4; i < N / 2; i++) {
120             v3[i] = v1[i] + v2[i];
121         }
122     }
123     #pragma omp section
124     {
125         for (i = N / 2; i < 3 * N / 4; i++) {
126             v3[i] = v1[i] + v2[i];
127         }
128     }
129     #pragma omp section
130     {
131         for (i = 3 * N / 4; i < N; i++) {
132             v3[i] = v1[i] + v2[i];
133         }
134     }
135 }

```


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

```

arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2/ejer8
arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2/ejer8 165x25
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2/ejer8] 2019-03-20 miércoles
$g++ SumaVectores_paralelo.c -o SumaVectores_paralelo -O2 -fopenmp
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2/ejer8] 2019-03-20 miércoles
$./SumaVectores_paralelo 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000012608 / Tamaño Vectores:8
/ V1[0]+V2[0]=V3[0] (0.800000+0.800000=1.600000) /
/ V1[1]+V2[1]=V3[1] (0.900000+0.700000=1.600000) /
/ V1[2]+V2[2]=V3[2] (1.000000+0.600000=1.600000) /
/ V1[3]+V2[3]=V3[3] (1.100000+0.500000=1.600000) /
/ V1[4]+V2[4]=V3[4] (1.200000+0.400000=1.600000) /
/ V1[5]+V2[5]=V3[5] (1.300000+0.300000=1.600000) /
/ V1[6]+V2[6]=V3[6] (1.400000+0.200000=1.600000) /
/ V1[7]+V2[7]=V3[7] (1.500000+0.100000=1.600000) /
Primeros elementos de v1, v2 y v3: 0.800000, 0.800000, 1.600000
Últimos elementos de v1, v2 y v3: 1.500000, 0.100000, 1.600000
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2/ejer8] 2019-03-20 miércoles
$./SumaVectores_paralelo 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000015537 / 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) /
Primeros elementos de v1, v2 y v3: 1.100000, 1.100000, 2.200000
Últimos elementos de v1, v2 y v3: 2.100000, 0.100000, 2.200000
[ArturoCortesSanchez arturo@mior-pc:~/Escritorio/Facultad/Segundo cuatrimestre/AC/Practica 2/ejer8] 2019-03-20 miércoles
$

```

9. ¿Cuántos threads y cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 7? Razone su respuesta. ¿Cuántos threads y cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta.

RESPUESTA:

En el ejercicio 7, con parallel for se se asignará como máximo una hebra a cada elemento del array que se procesa, por tanto el máximo de hebras será igual al tamaño del array. Para el ejercicio 8 solo se podrán usar 4 hebras porque solo he puesto cuatro sections.

10. Rellenar una tabla como la Tabla 2 para atcgrid y otra para su PC con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución del trozo de código que realiza la suma en paralelo (este es el tiempo que deben imprimir los programas). Ponga en la tabla el número de threads/cores que usan los códigos (use el máximo número de cores físicos del computador que como máximo puede aprovechar el código, no use un número de threads superior al número de cores físicos). Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute código que imprima todos los componentes del resultado cuando este número sea elevado.

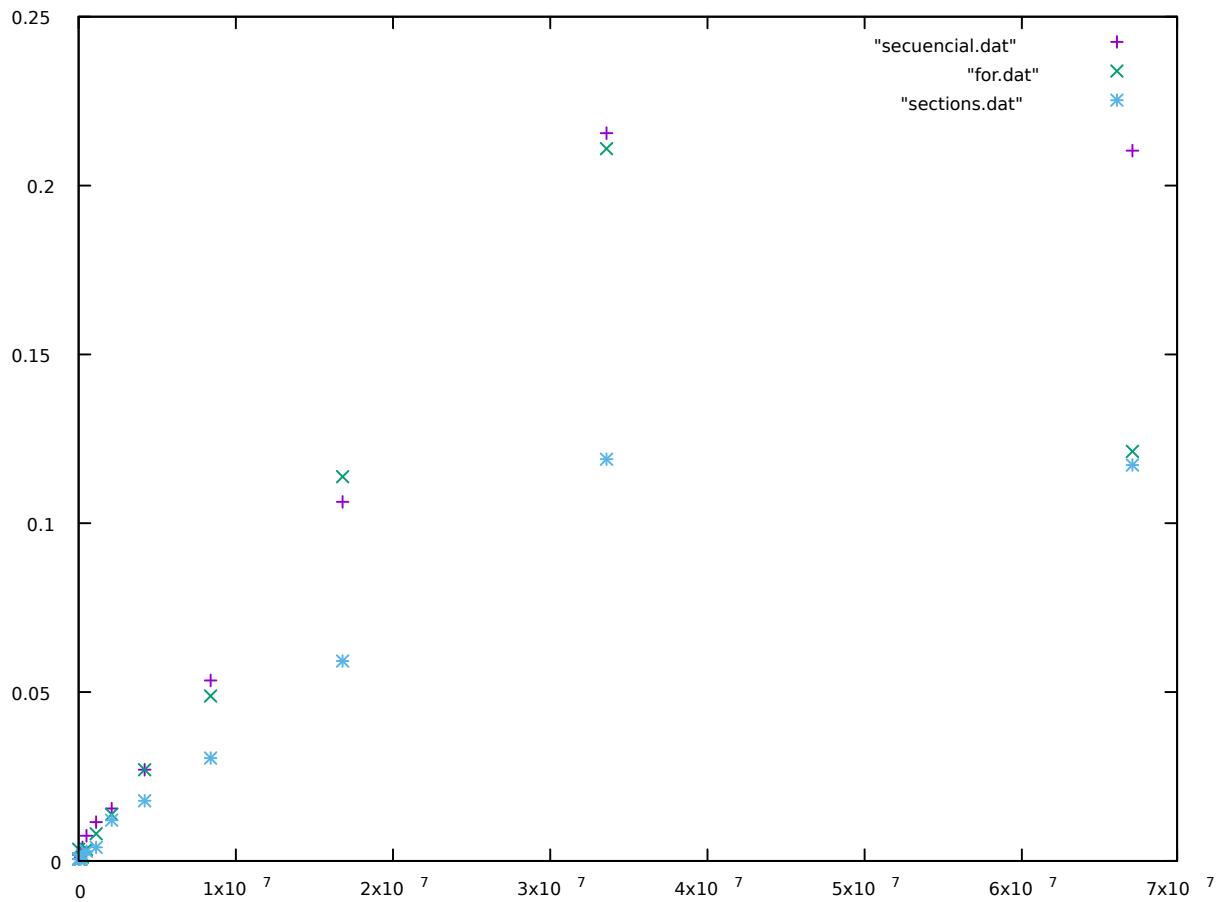
RESPUESTA:

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

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) ¿?threads/cores	T. paralelo (versión sections) 4 threads/cores
16384			
32768			
65536			
131072			
262144			
524288			
1048576			
2097152			
4194304			
8388608			
16777216			
33554432			
67108864			

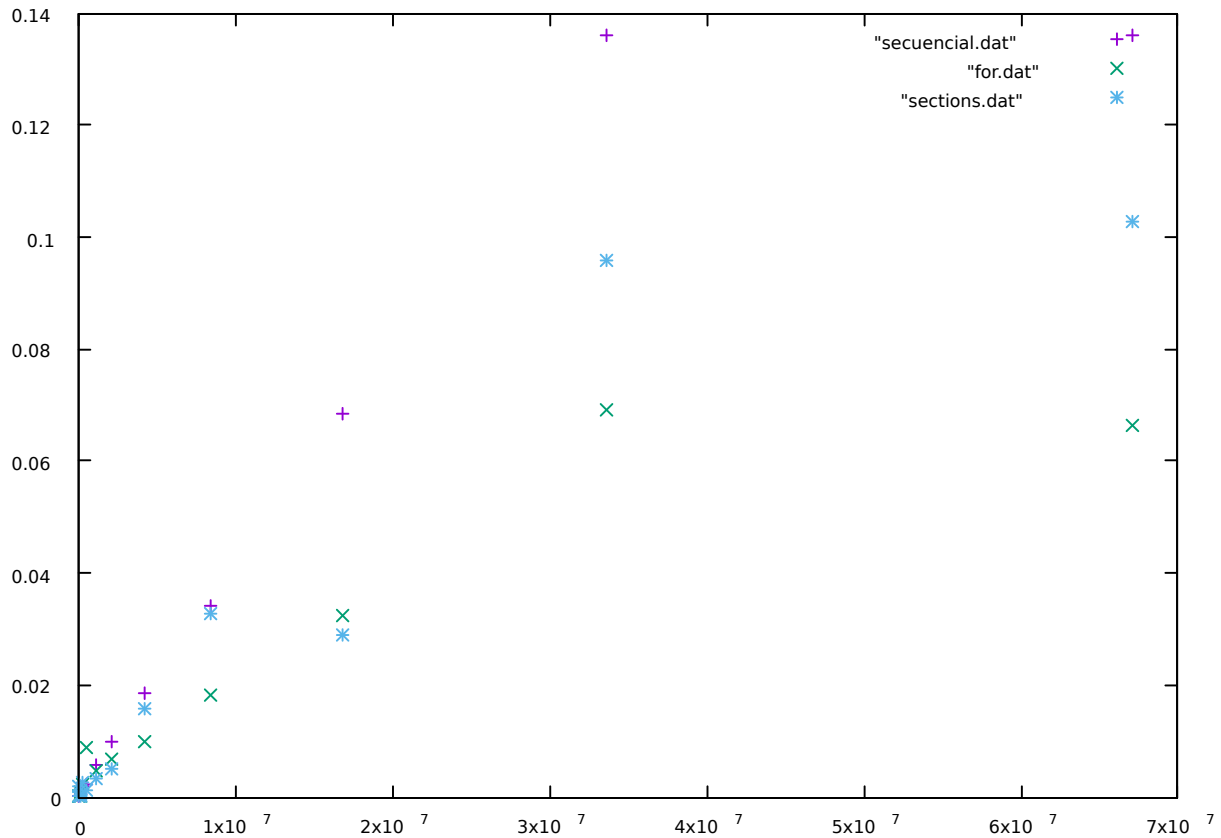
PC:

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	0.000182867	0.000068429	0.000074507
32768	0.000481493	0.003456867	0.000140453
65536	0.000781626	0.000218293	0.000211306
131072	0.001222395	0.000533998	0.000527985
262144	0.003517747	0.001121919	0.001612833
524288	0.007435114	0.003091967	0.002577993
1048576	0.011006772	0.007937677	0.004049805
2097152	0.015507071	0.013702680	0.011658519
4194304	0.026971008	0.026498454	0.017384292
8388608	0.053373113	0.048655939	0.030371346
16777216	0.105855197	0.113649198	0.058855202
33554432	0.215125636	0.210577082	0.118498536
67108864	0.210247551	0.120796377	0.117012371



Atcgrid:

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 12 threads/cores	T. paralelo (versión sections) 4 threads/cores
16384	0.000129479	0.000061381	0.001785604
32768	0.000230643	0.000087616	0.000094245
65536	0.000472086	0.000120472	0.000179406
131072	0.001086004	0.000796721	0.000375585
262144	0.002032990	0.002109870	0.002446681
524288	0.002253463	0.008745358	0.001298852
1048576	0.005797214	0.004572833	0.003250765
2097152	0.009728710	0.006690020	0.005024543
4194304	0.018456538	0.009932669	0.015859092
8388608	0.034257023	0.018315713	0.032865037
16777216	0.068519032	0.032591699	0.028884643
33554432	0.135920893	0.069228205	0.095702855
67108864	0.136183280	0.066498255	0.102860822



11. Rellenar una tabla como la Tabla 3 para atcgrid con el tiempo de ejecución, tiempo de CPU del usuario y tiempo CPU del sistema obtenidos con `time` para el ejecutable del ejercicio 7 y para el programa secuencial del Listado 1. Ponga en la tabla el número de threads/cores que usan los códigos. ¿El tiempo de CPU que se obtiene es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

RESPUESTA:

Tabla 3. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla “¿?” por el número de threads utilizados.

Nº de 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						
131072						
262144						
524288						
1048576						
2097152						
4194304						
8388608						
16777216						
33554432						
67108864						

Nº de Componentes	Tiempo secuencial vect. Globales 1 thread/core			Tiempo paralelo/versión for 12 Threads/cores		
	Elapsed	CPU-user	CPU- sys	Elapsed	CPU-user	CPU- sys
65536	0.009	0.004	0.002	0.013	0.059	0.144
131072	0.006	0.001	0.004	0.010	0.038	0.143
262144	0.007	0.003	0.004	0.014	0.088	0.184
524288	0.010	0.006	0.004	0.014	0.086	0.181
1048576	0.017	0.007	0.010	0.016	0.088	0.240
2097152	0.028	0.015	0.013	0.023	0.264	0.204
4194304	0.049	0.020	0.028	0.027	0.389	0.206
8388608	0.086	0.047	0.039	0.048	0.667	0.302
16777216	0.165	0.088	0.077	0.088	1.244	0.557
33554432	0.323	0.174	0.148	0.166	2.343	0.960
67108864	0.322	0.176	0.146	0.166	2.539	1.026

En el caso secuencial, el tiempo de CPU suele ser menor o igual que el tiempo real, pero en el caso de la versión paralela, el tiempo de CPU es mayor que el real, debido a la sobrecarga que producen las hebras