

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): Juan Manuel Rubio Rodríguez

Grupo de prácticas: D1

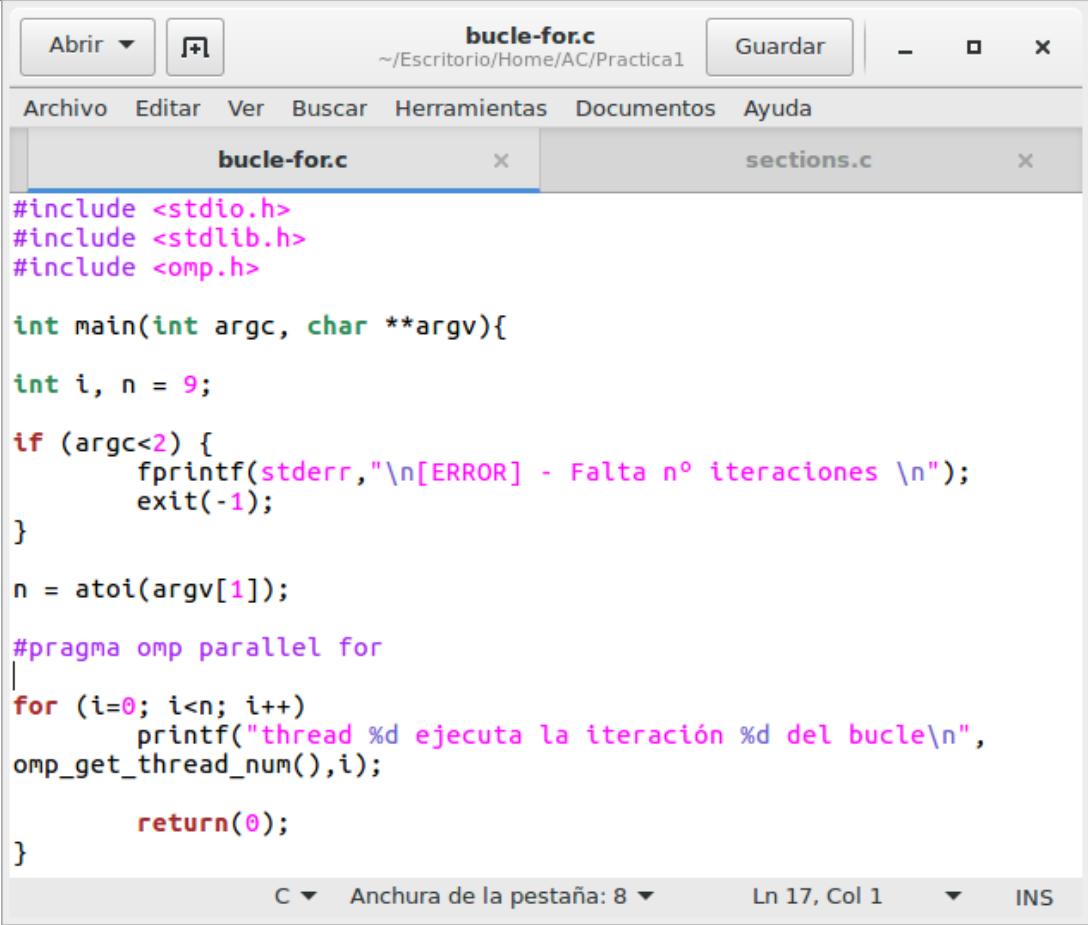
Fecha de entrega:

Fecha evaluación en clase:

Ejercicios basados en los ejemplos del seminario práctico

- 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.

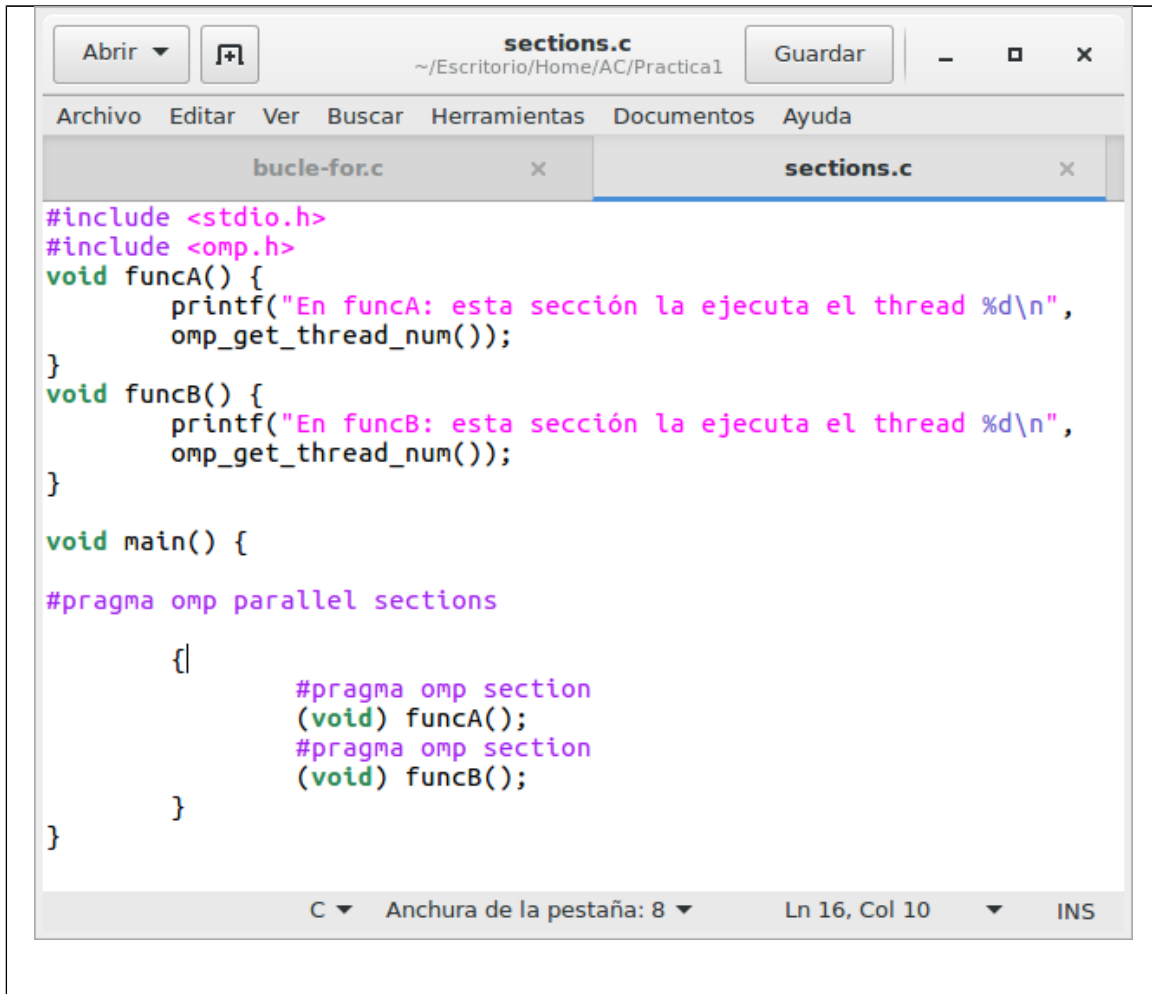
RESPUESTA: Captura que muestre el código fuente `bucle-forModificado.c`



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

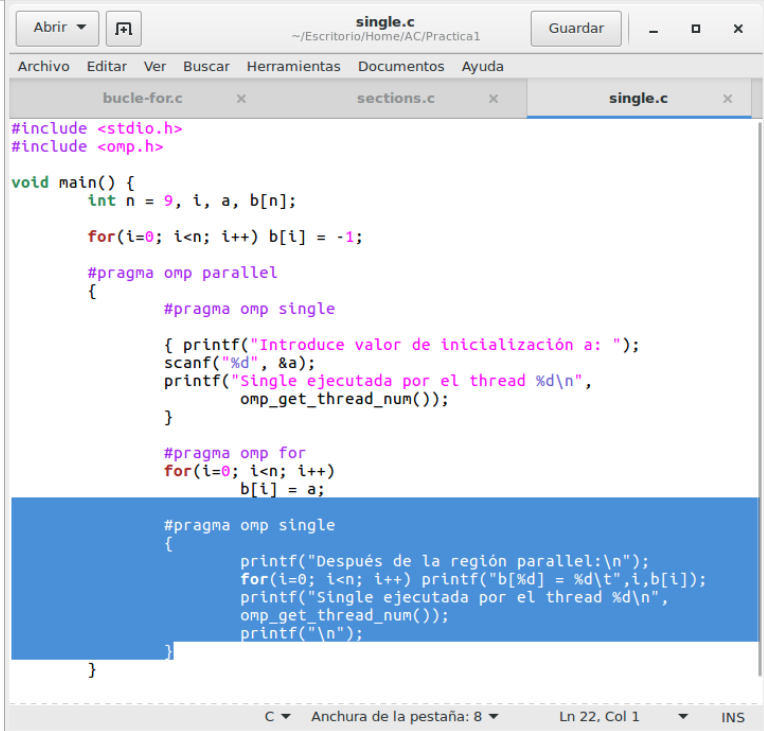


```
#include <stdio.h>
#include <omp.h>
void funcA() {
    printf("En funcA: esta sección la ejecuta el thread %d\n",
        omp_get_thread_num());
}
void funcB() {
    printf("En funcB: esta sección la ejecuta el thread %d\n",
        omp_get_thread_num());
}

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

- . 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`



```

single.c
~/Escritorio/Home/AC/Practica1
Guardar

Archivo Editar Ver Buscar Herramientas Documentos Ayuda
bucle-for.c x sections.c x single.c x

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

void main() {
    int n = 9, i, a, b[n];

    for(i=0; i<n; i++) b[i] = -1;

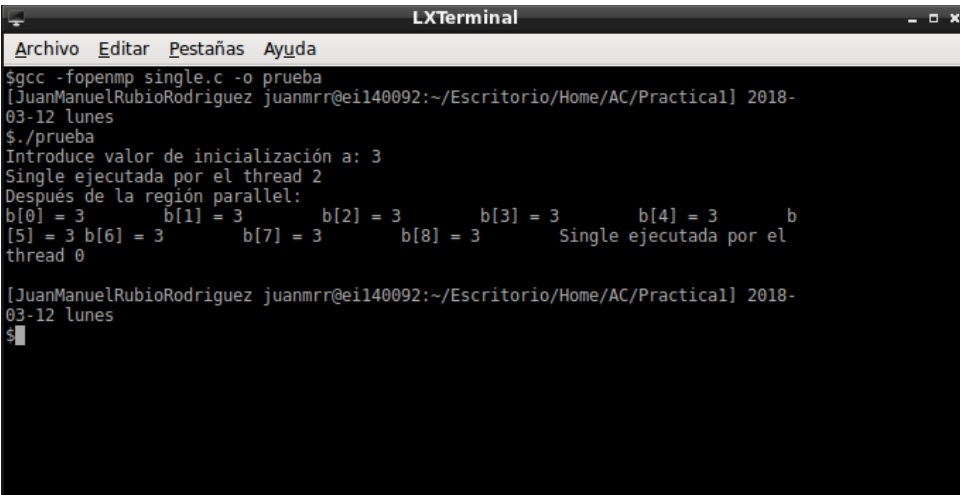
    #pragma omp parallel
    {
        #pragma omp single
        { printf("Introduce valor de inicialización a: ");
          scanf("%d", &a);
          printf("Single ejecutada por el thread %d\n",
                omp_get_thread_num());
        }

        #pragma omp for
        for(i=0; i<n; i++)
            b[i] = a;

        #pragma omp single
        {
            printf("Después de la región parallel:\n");
            for(i=0; i<n; i++) printf("b[%d] = %d\t", i, b[i]);
            printf("Single ejecutada por el thread %d\n",
                  omp_get_thread_num());
            printf("\n");
        }
    }
}
C Anchura de la pestaña: 8 Ln 22, Col 1 INS

```

CAPTURAS DE PANTALLA:



```

LXTerminal
Archivo Editar Pestañas Ayuda
$gcc -fopenmp single.c -o prueba
[JuanManuelRubioRodriguez juanmrr@eil140092:~/Escritorio/Home/AC/Practica1] 2018-03-12 lunes
$./prueba
Introduce valor de inicialización a: 3
Single ejecutada por el thread 2
Después de la región parallel:
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    Single ejecutada por el
thread 0

[JuanManuelRubioRodriguez juanmrr@eil140092:~/Escritorio/Home/AC/Practica1] 2018-03-12 lunes
$

```

- . 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`

```

singleModificado2.c
~/Escritorio/Home/AC/Practica1

Archivo Editar Ver Buscar Herramientas Documentos Ayuda
bucle-for.c x sections.c x singleModificado2.c x

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

void main() {
    int n = 9, i, a, b[n];

    for(i=0; i<n; i++) b[i] = -1;

    #pragma omp parallel
    {
        #pragma omp master

        { 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("Después de la región parallel:\n");
            for(i=0; i<n; i++) printf("b[%d] = %d\t", i, b[i]);
            printf("Single ejecutada por el thread %d\n",
                  omp_get_thread_num());
            printf("\n");
        }
    }
}
C Anchura de la pestaña: 8 Ln 17, Col 18 INS

```

CAPTURAS DE PANTALLA:

```

LXTerminal
Archivo Editar Pestañas Ayuda
[juanmanuelrubiorodriguez@juanmrr@ei140092:~/Escritorio/Home/AC/Practica1] 2018-03-12 lunes
$gcc -fopenmp singleModificado2.c -o prueba
[juanmanuelrubiorodriguez@juanmrr@ei140092:~/Escritorio/Home/AC/Practica1] 2018-03-12 lunes
$./prueba
Introduce valor de inicialización a: 2
Single ejecutada por el thread 0
Después de la región parallel:
b[0] = 2      b[1] = 2      b[2] = 2      b[3] = 0      b[4] = 0      b[5] = 0      b
[6] = 0 b[7] = 0      b[8] = 0      Single ejecutada por el thread 0
[juanmanuelrubiorodriguez@juanmrr@ei140092:~/Escritorio/Home/AC/Practica1] 2018-03-12 lunes
$

```

RESPUESTA A LA PREGUNTA:

La directiva master provoca que la hebra que se ejecute dentro de ella sea la cero.

- . ¿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:

Puede que imprima la suma antes de haber realizado todas las sumas parciales, pues nada le impide seguir con la ejecución.

Resto de ejercicios

- . El programa secuencial C del Listado 1 calcula la suma de dos vectores ($v3 = v1 + v2$; $v3(i) = v1(i) + v2(i)$, $i=0, \dots, N-1$). Generar el ejecutable del programa del Listado 1 para **vectores globales**. Usar time (Lección 3/ Tema 1) en la línea de comandos para obtener, en 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:

```

LXTerminal
Archivo Editar Pestañas Ayuda
[JuanManuelRubioRodriguez juanmrr@eil42094:~/Escritorio/Home/AC/Practical1] 2018-
03-19 lunes
$cat STDIN.e67233

real    0m0.164s
user    0m0.053s
sys     0m0.108s
[JuanManuelRubioRodriguez juanmrr@eil42094:~/Escritorio/Home/AC/Practical1] 2018-
03-19 lunes
$cat STDIN.o67233
Tiempo(seg.):0.054524366          / 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) /
[JuanManuelRubioRodriguez juanmrr@eil42094:~/Escritorio/Home/AC/Practical1] 2018-
03-19 lunes
$

```

El tiempo real es mayor debido a las esperas asociadas a operaciones de I/O o a la ejecución de otros programas.

- . 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:

```

LXTerminal
Archivo Editar Pestañas Ayuda
[JuanManuelRubioRodriguez juanmrr@eil42094:~/Escritorio/Home/AC/Practica1] 2018-
03-19 lunes
$cat STDIN.e67233

real    0m0.164s
user    0m0.053s
sys      0m0.108s
[JuanManuelRubioRodriguez juanmrr@eil42094:~/Escritorio/Home/AC/Practica1] 2018-
03-19 lunes
$cat STDIN.o67233
Tiempo(seg.):0.054524366 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3
[0](1000000.000000+1000000.000000=2000000.000000) V1[999999]+V2[999999]=V3[999
999](1999999.900000+0.100000=2000000.000000) /
[JuanManuelRubioRodriguez juanmrr@eil42094:~/Escritorio/Home/AC/Practica1] 2018-
03-19 lunes
$

```

```

juanma@juanma-X550VX: ~
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/Practica1] 2018-0
3-26 lunes
$cat STDIN.o68990
Tiempo(seg.):0.000002737 / Tamaño Vectores:10 / V1[0]+V2[0]=V3[0](1.00
0000+1.000000=2.000000) V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/Practica1] 2018-0
3-26 lunes
$

```

RESPUESTA: cálculo de los MIPS y los MFLOPS

$$\text{MIPS} = \text{NI} / \text{TCPU} * 10^6$$

$$\text{MIPS} = 6 * 10^7 / 0,054524366 * 10^6 = 1,10042554186 * 10^3$$

$$\text{MIPS} = 6 * 10^7 / 0,000002737 * 10^6 = 2,37486298867 * 10^1$$

$$\text{MFLOPS} = \text{N}^{\circ} \text{op.flotante} / \text{TCPU} * 10^6$$

$$\text{MFLOPS} = 1 * 10^7 / 0,054524366 * 10^6 = 1,83404241693 * 10^2$$

$$\text{MFLOPS} = 1 * 10^7 / 0,000002737 * 10^6 = 3,65363536719$$

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

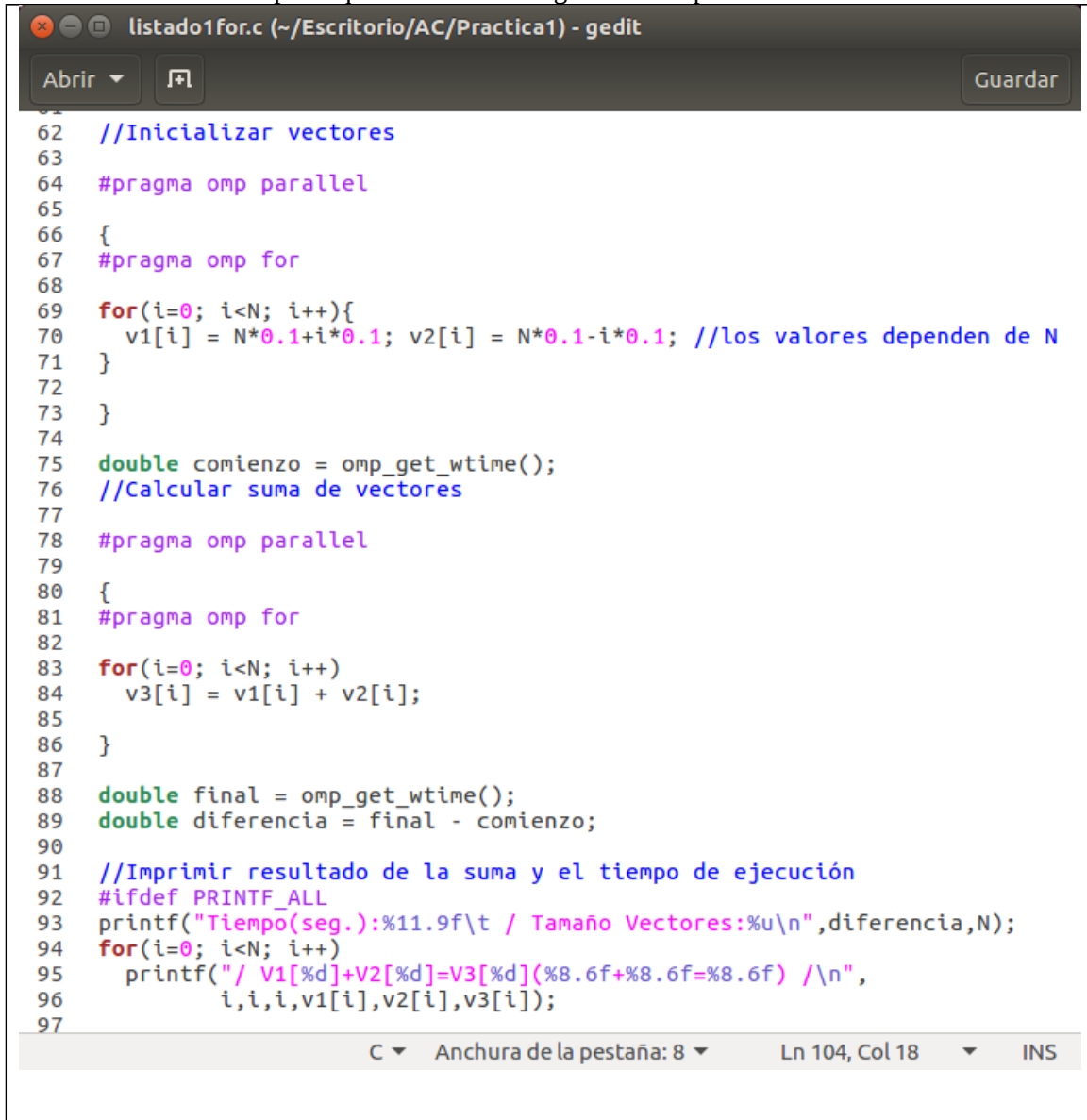
```

57      cvtsi2sd      %ebx, %xmm0
58      mulsd      %xmm3, %xmm0
59      movapd      %xmm0, %xmm2
60      subsd      %xmm0, %xmm7
61      addsd      %xmm1, %xmm2
62      movsd      %xmm7, v2(,%rbx,8)
63      movsd      %xmm2, v1(,%rbx,8)
64      addq      $1, %rbx
65      cmpq      %rax, %rbx
66      jne      .L4
67      movq      %rsp, %rsi
68      xorl      %edi, %edi
69      salq      $3, %rbx
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
85      movq      24(%rsp), %rax
86      subq      8(%rsp), %rax
87      movl      %rax, %edi

```

Texto plano ▾ Anchura de la pestaña: 8 ▾ Ln 71, Col 1 ▾ INS

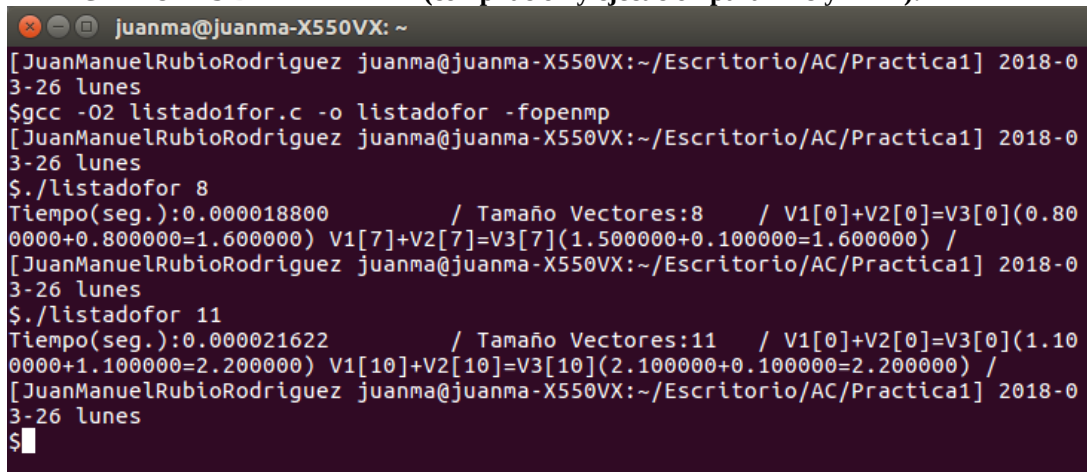
1. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores ($v3 = v1 + v2$; $v3(i) = v1(i) + v2(i)$, $i = 0, \dots, N-1$) usando las directivas `parallel` y `for`. Se debe paralelizar también las tareas asociadas a la inicialización 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


```

62 //Inicializar vectores
63
64 #pragma omp parallel
65 {
66     #pragma omp for
67     for(i=0; i<N; i++){
68         v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de N
69     }
70 }
71
72 //Calcular suma de vectores
73
74 #pragma omp parallel
75 {
76     #pragma omp for
77     for(i=0; i<N; i++){
78         v3[i] = v1[i] + v2[i];
79     }
80 }
81
82 double comienzo = omp_get_wtime();
83 //Calcular suma de vectores
84
85 #pragma omp parallel
86 {
87     #pragma omp for
88     for(i=0; i<N; i++){
89         v3[i] = v1[i] + v2[i];
90     }
91 }
92
93 double final = omp_get_wtime();
94 double diferencia = final - comienzo;
95
96 //Imprimir resultado de la suma y el tiempo de ejecución
97 #ifdef PRINTF_ALL
98 printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",diferencia,N);
99 for(i=0; i<N; i++){
100     printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
101         i,i,i,v1[i],v2[i],v3[i]);
102 }
103 #endif

```

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


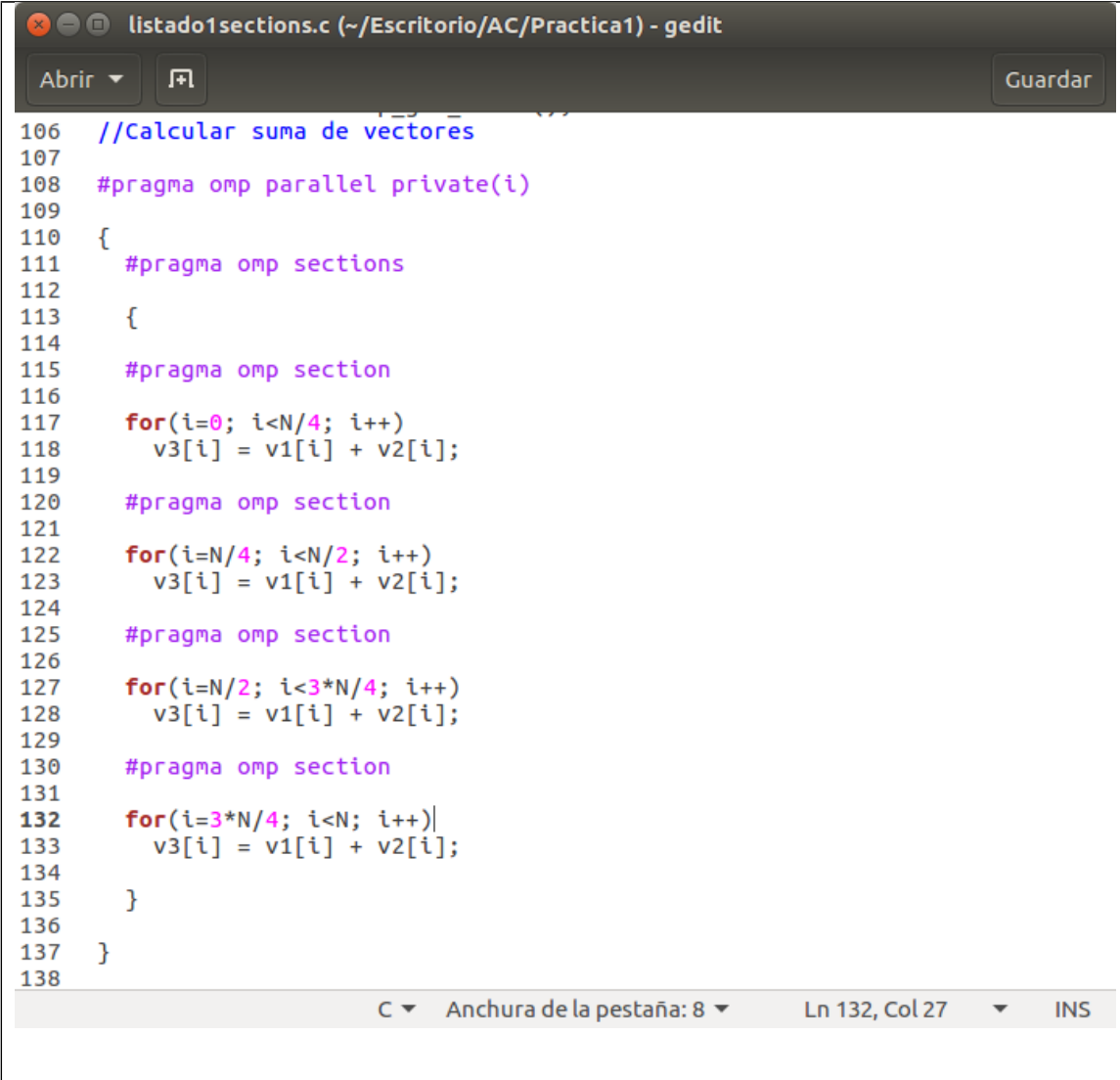
```

juanma@juanma-X550VX: ~
[juanma@juanma-X550VX:~/Escritorio/AC/Practica1] 2018-0
3-26 lunes
$gcc -O2 listado1for.c -o listadofor -fopenmp
[juanma@juanma-X550VX:~/Escritorio/AC/Practica1] 2018-0
3-26 lunes
$./listadofor 8
Tiempo(seg.):0.000018800 / Tamaño Vectores:8 / V1[0]+V2[0]=V3[0](0.80
0000+0.800000=1.600000) V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
[juanma@juanma-X550VX:~/Escritorio/AC/Practica1] 2018-0
3-26 lunes
$./listadofor 11
Tiempo(seg.):0.000021622 / Tamaño Vectores:11 / V1[0]+V2[0]=V3[0](1.10
0000+1.100000=2.200000) V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
[juanma@juanma-X550VX:~/Escritorio/AC/Practica1] 2018-0
3-26 lunes
$

```


- . 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



The screenshot shows a gedit editor window titled "listado1sections.c (~/Escritorio/AC/Practica1) - gedit". The window has a menu bar with "Abrir" and "Guardar" buttons. The code is as follows:

```
106 //Calcular suma de vectores
107
108 #pragma omp parallel private(i)
109 {
110     #pragma omp sections
111     {
112         #pragma omp section
113         for(i=0; i<N/4; i++)
114             v3[i] = v1[i] + v2[i];
115
116         #pragma omp section
117         for(i=N/4; i<N/2; i++)
118             v3[i] = v1[i] + v2[i];
119
120         #pragma omp section
121         for(i=N/2; i<3*N/4; i++)
122             v3[i] = v1[i] + v2[i];
123
124         #pragma omp section
125         for(i=3*N/4; i<N; i++)
126             v3[i] = v1[i] + v2[i];
127     }
128 }
129
130
131
132
133
134
135
136
137
138
```

The status bar at the bottom indicates "C", "Anchura de la pestaña: 8", "Ln 132, Col 27", and "INS".

```

63     printf("Error en la reserva de espacio para los vectores\n"),
64     exit(-2);
65 }
66 #endif
67
68 //Inicializar vectores
69
70 #pragma omp parallel
71 {
72     #pragma omp sections private(i)
73     {
74         #pragma omp section
75         {
76             #pragma omp section
77             for(i=0; i<N/4; i++){
78                 v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de
79                 N
80             }
81             #pragma omp section
82             for(i=N/4; i<N/2; i++){
83                 v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de
84                 N
85             }
86             #pragma omp section
87             for(i=N/2; i<3*N/4; i++){
88                 v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de
89                 N
90             }
91             #pragma omp section
92             for(i=3*N/4; i<N; i++){
93                 v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de
94                 N
95             }
96         }
97     }
98     double comienzo = omp_get_wtime();
99     //Calcular suma de vectores
100
101     #pragma omp parallel private(i)
102     {
103         #pragma omp sections
104         {
105             #pragma omp section

```

C ▾ Anchura de la pestaña: 8 ▾ Ln 103, Col 4 ▾ INS

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```

juanma@juanma-X550VX: ~
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/Practica1] 2018-0
3-26 lunes
$gcc -O2 listado1sections.c -o listadosections -fopenmp
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/Practica1] 2018-0
3-26 lunes
$./listadosections 8
Tiempo(seg.):0.000024837 / Tamaño Vectores:8 / V1[0]+V2[0]=V3[0](0.80
0000+0.800000=1.600000) V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/Practica1] 2018-0
3-26 lunes
$./listadosections 11
Tiempo(seg.):0.000020995 / Tamaño Vectores:11 / V1[0]+V2[0]=V3[0](1.10
0000+1.100000=2.200000) V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/Practica1] 2018-0
3-26 lunes
$

```

- . ¿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:

Como no hemos fijado la variable de entorno OMP_NUM_THREADS, en ambos casos se ejecutarán con el máximo de cores lógicos disponibles. En el caso del ejercicio 8, está programado para aprovechar 4 threads.

- . 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:

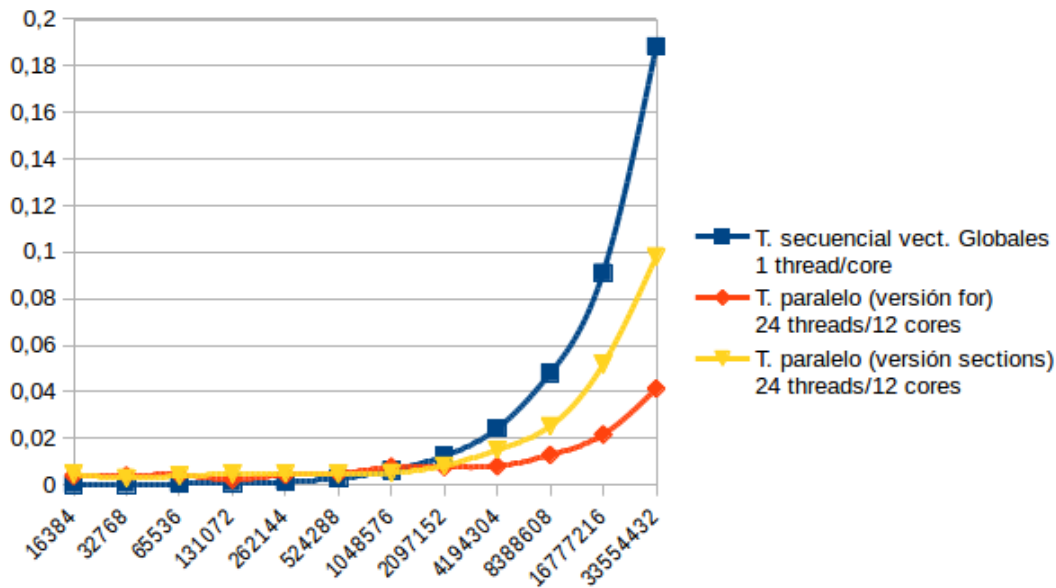


Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla “?” por el número de threads utilizados, que debe coincidir con el número de cores físicos utilizados.

' de Component	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 24 threads/12 cores	T. paralelo (versión sections) 24 threads/12 cores
16384	0,00011024	0,003916312	0,004636964
32768	0,000214805	0,003927053	0,002933783
65536	0,000440798	0,004223738	0,003855501
131072	0,000858666	0,002273383	0,004608277
262144	0,001322313	0,004263617	0,004989085
524288	0,002820559	0,004880893	0,004731708
1048576	0,006159536	0,007761521	0,005203183
2097152	0,012407851	0,007738019	0,008153208
4194304	0,024263801	0,008003552	0,015064378
8388608	0,047937118	0,012863912	0,025248748
16777216	0,091195253	0,021655632	0,051777942
33554432	0,188316078	0,041361358	0,098129817
67108864	0,181112623	0,045260213	0,099727063

Esta es la gráfica y la tabla para atcgrid.

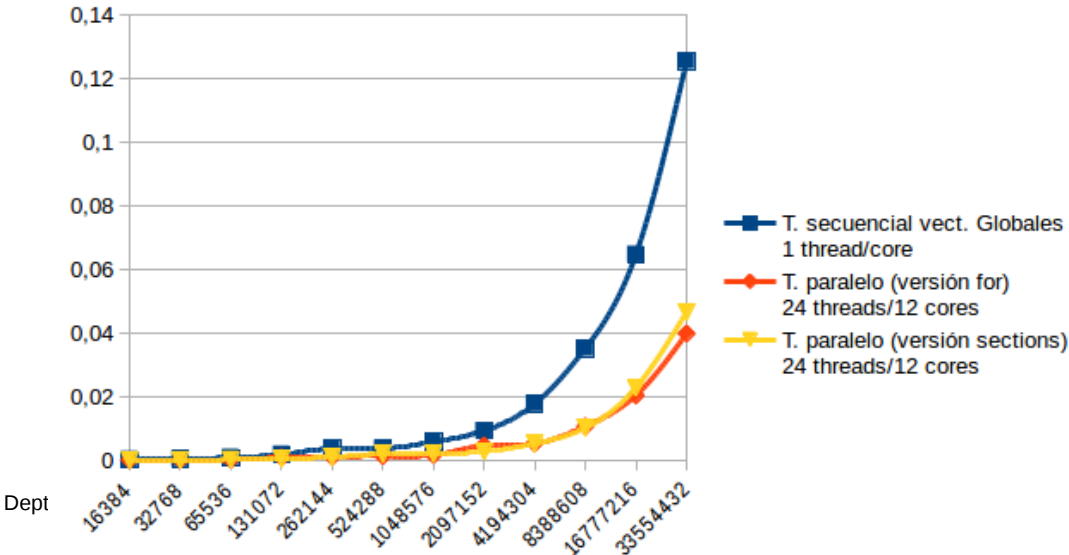


Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla “¿?” por el número de threads utilizados, que debe coincidir con el número de cores físicos utilizados.

' de Component	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 24 threads/12 cores	T. paralelo (versión sections) 24 threads/12 cores
16384	0,00011024	0,003916312	0,004636964
32768	0,000214805	0,003927053	0,002933783
65536	0,000440798	0,004223738	0,003855501
131072	0,000858666	0,002273383	0,004608277
262144	0,001322313	0,004263617	0,004989085
524288	0,002820559	0,004880893	0,004731708
1048576	0,006159536	0,007761521	0,005203183
2097152	0,012407851	0,007738019	0,008153208
4194304	0,024263801	0,008003552	0,015064378
8388608	0,047937118	0,012863912	0,025248748
16777216	0,091195253	0,021655632	0,051777942
33554432	0,188316078	0,041361358	0,098129817
67108864	0,181112623	0,045260213	0,099727063

Esta es la gráfica y la tabla para mi pc. La gráfica tiene sentido para estos datos obtenidos con el comando `lscpu`:

```

juanma@juanma-X550VX: ~
3554431](6710886.300000+0.100000=6710886.400000) /
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/Practica1] 2018-0
3-27 martes
$ lscpu
Arquitectura:          x86_64
modo(s) de operación de las CPUs:32-bit, 64-bit
Orden de bytes:       Little Endian
CPU(s):               4
On-line CPU(s) list:  0-3
Hilo(s) de procesamiento por núcleo:1
Núcleo(s) por «socket»:4
Socket(s):             1
Modo(s) NUMA:         1
ID de fabricante:     GenuineIntel
Familia de CPU:        6
Modelo:               94
Model name:           Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz
Revisión:             3
CPU MHz:              2300.000
CPU max MHz:          3200,0000
CPU min MHz:          800,0000
BogoMIPS:             4608.00
Virtualización:       VT-x
Caché L1d:            32K
Caché L1i:            32K
Caché L2:             256K

```

- . 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 real en el caso secuencial prácticamente coincide con la suma de los tiempos de usuario y sistema, pues se ejecuta con una sola hebra y nunca es menor. En el caso del código paralelo, el tiempo real es sensiblemente menor que la suma del tiempo de usuario y de sistema, algo de esperar pues la tarea se divide en tantas hebras como cores lógicos tiene nuestra máquina y se trabaja de manera paralela; en CPU-user se acumula todo el tiempo de procesamiento de usuario entre todos los hilos. Con lo cual, la suma es siempre mayor que el tiempo real.

* La última iteración no es un dato relevante, pues el vector global es de tamaño 33554432 (2^{25}) y no se ha ejecutado sobre el ejecutable `listado1` modificado con el tamaño de 2^{26} .

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 24 Threads/cores		
	Elapsed	CPU-user	CPU- sys	Elapsed	CPU-user	CPU- sys
65536	0m0.003s	0m0.001s	0m0.001s	0m0.019s	0m0.233s	0m0.005s
131072	0m0.004s	0m0.001s	0m0.003s	0m0.013s	0m0.211s	0m0.025s
262144	0m0.007s	0m0.001s	0m0.005s	0m0.014s	0m0.206s	0m0.002s
524288	0m0.009s	0m0.006s	0m0.003s	0m0.014s	0m0.226s	0m0.012s
1048576	0m0.020s	0m0.007s	0m0.013s	0m0.014s	0m0.199s	0m0.022s
2097152	0m0.037s	0m0.008s	0m0.029s	0m0.021s	0m0.256s	0m0.065s
4194304	0m0.074s	0m0.021s	0m0.053s	0m0.034s	0m0.293s	0m0.129s
8388608	0m0.135s	0m0.048s	0m0.084s	0m0.046s	0m0.366s	0m0.258s
16777216	0m0.280s	0m0.105s	0m0.173s	0m0.083s	0m0.565s	0m0.534s
33554432	0m0.552s	0m0.207s	0m0.341s	0m0.153s	0m0.867s	0m1.023s
67108864	0m0.544s	0m0.179s	0m0.360s	0m0.144s	0m0.859s	0m0.966s