

BP1.pdf



Nashor



Arquitectura de Computadores



2º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación
Universidad de Granada

NEW

WUOLAH Print

Lo que faltaba en Wuolah



Imprimir



- Todos los apuntes que necesitas están aquí
- Al mejor precio del mercado, desde **2 cent.**
- Recoge los apuntes en tu copistería más cercana o recíbelos en tu casa
- Todas las anteriores son correctas

2º curso / 2º cuatr.
Grado Ing. Inform.
Doble Grado Ing.
Inform. y Mat.

Imprimir



PDF

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 1. Programación paralela I: Directivas OpenMP

Estudiante (nombre y apellidos): Ignacio Carvajal Herrera

Grupo de prácticas y profesor de prácticas: D2 Jorge Sánchez Garrido

Fecha de entrega: 21/03/2020

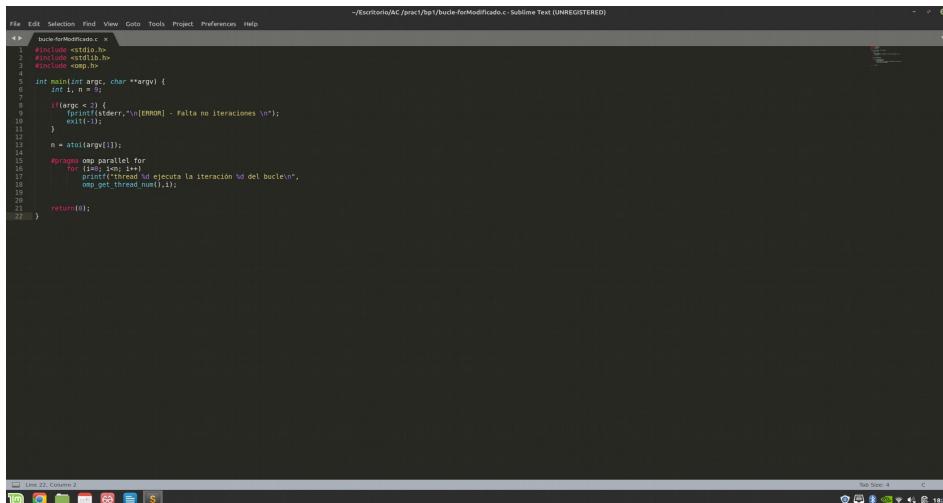
Fecha evaluación en clase: 24/03/2020

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`

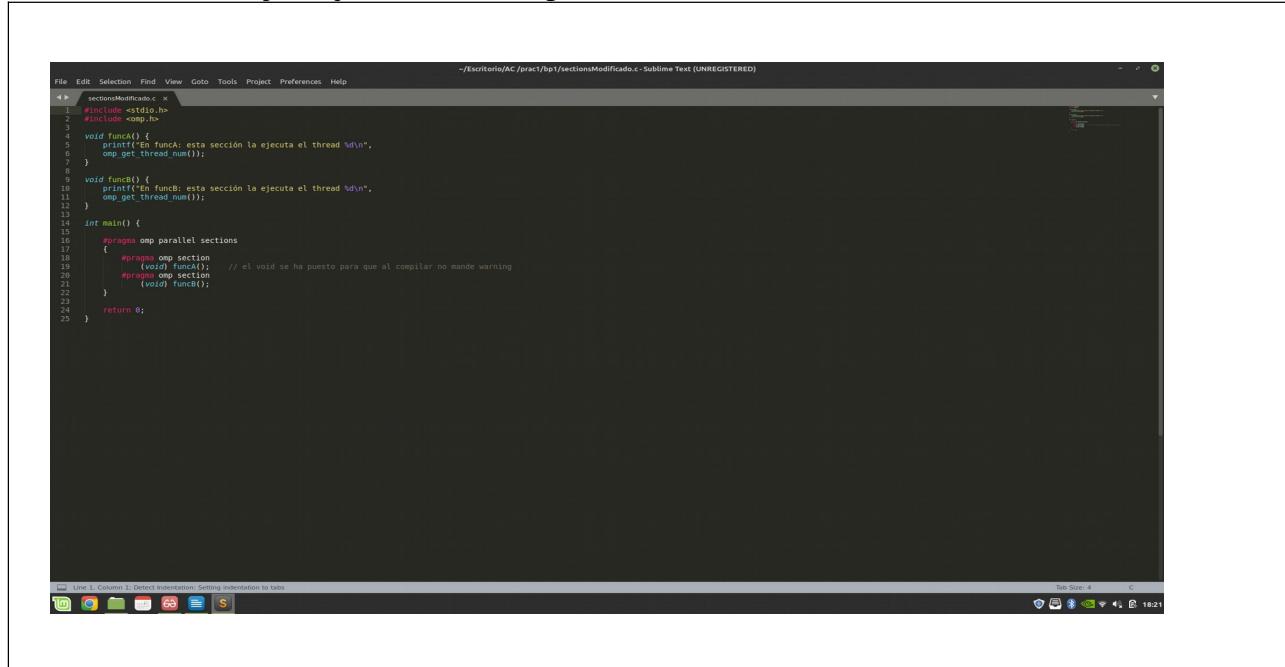


```

File Edit Selection Find View Goto Tools Project Preferences Help
bucle-forModificado.c < /Escritorio/AC /pract1/hpt1/bucle-forModificado.c - Sublime Text (UNREGISTERED)
1 // Ejemplo de uso de OpenMP
2 #include <stdio.h>
3 #include <omp.h>
4
5 int main(int argc, char **argv) {
6     int i, n;
7     if(argc < 2) {
8         fprintf(stderr, "Usage: %s [NOMIN] - Falta n iteraciones\n");
9         exit(1);
10    }
11    n = atoi(argv[1]);
12
13    #pragma omp parallel for
14    for (i=0; i<n; i++)
15        #pragma omp critical
16        printf("Thread %d ejecuta la iteración %d del bucle\n",
17               omp_get_thread_num(), i);
18
19
20
21    return(0);
22 }

```

RESPUESTA: Captura que muestre el código fuente sectionsModificado.c

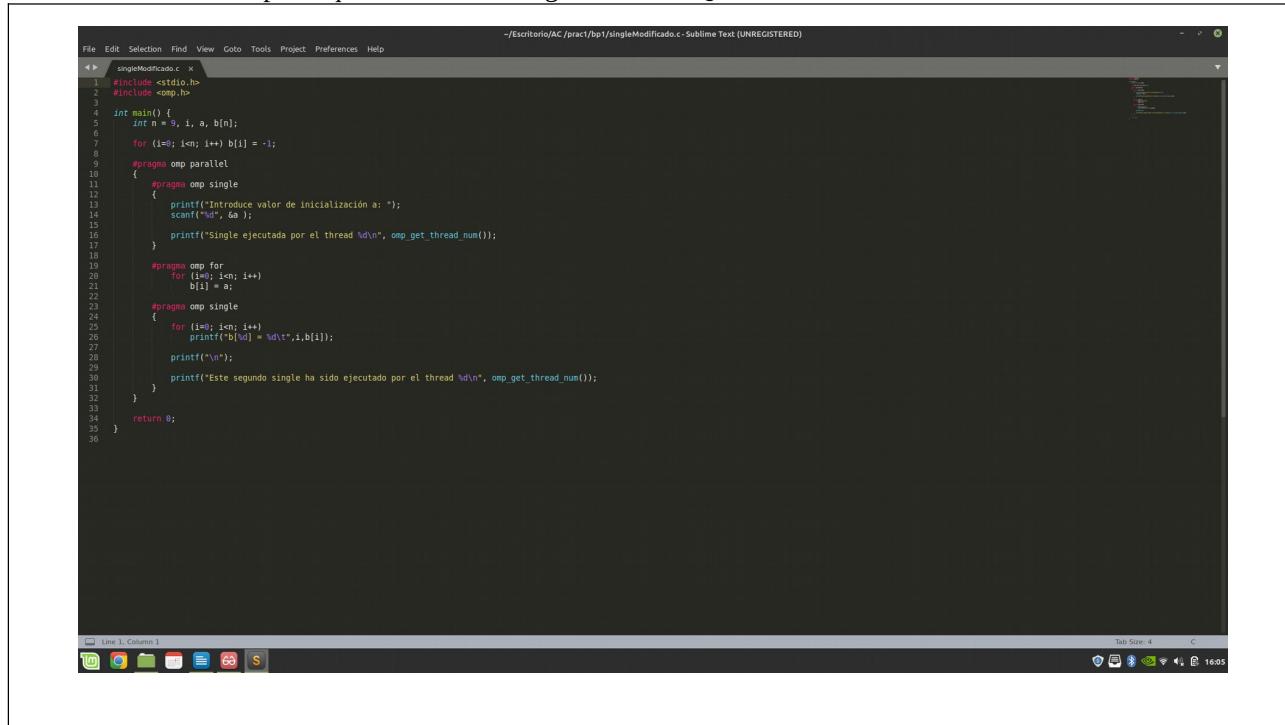


```

File Edit Selection Find View Goto Tools Project Preferences Help
sectionsModificado.c x -/Escritorio/AC/pract/bpt/sectionsModificado.c - Sublime Text (UNREGISTERED)
1 #include <stdio.h>
2 #include <omp.h>
3
4 void funcA() {
5     printf("En la sección funcA: esta sección la ejecuta el thread %d\n",
6         omp_get_thread_num());
7 }
8
9 void funcB() {
10    printf("En la sección funcB: esta sección la ejecuta el thread %d\n",
11        omp_get_thread_num());
12 }
13
14 int main() {
15     #pragma omp parallel sections
16     {
17         #pragma omp section
18         void funcA(); // el void se ha puesto para que al compilar no mande warning
19         #pragma omp section
20         void funcB();
21     }
22 }
23
24 return 0;
25 }
```

- Imprimir los resultados del programa single.c usando una directiva single dentro de la construcción parallel en lugar de imprimirlas 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



```

File Edit Selection Find View Goto Tools Project Preferences Help
singleModificado.c x -/Escritorio/AC/pract/bpt/singleModificado.c - Sublime Text (UNREGISTERED)
1 #include <stdio.h>
2 #include <omp.h>
3
4 int main() {
5     int n = 9, i, a, b[n];
6
7     for (i=0; i<n; i++) b[i] = -1;
8
9     #pragma omp parallel
10    {
11         #pragma omp single
12         {
13             printf("Introduce valor de inicialización a: ");
14             scanf("%d", &a);
15
16             printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
17         }
18
19         #pragma omp for
20         for (i=0; i<n; i++)
21             b[i] = a;
22
23         #pragma omp single
24         {
25             for (i=0; i<n; i++)
26                 printf("b[%d] = %d\n", i, b[i]);
27
28             printf("\n");
29
30             printf("Este segundo single ha sido ejecutado por el thread %d\n", omp_get_thread_num());
31
32         }
33
34     }
35
36 }
```

NEW

WUOLAH Print

Lo que faltaba en Wuolah



Imprimir



- Todos los apuntes que necesitas están aquí
- Al mejor precio del mercado, desde **2 cent.**
- Recoge los apuntes en tu copistería más cercana o recíbelos en tu casa
- Todas las anteriores son correctas

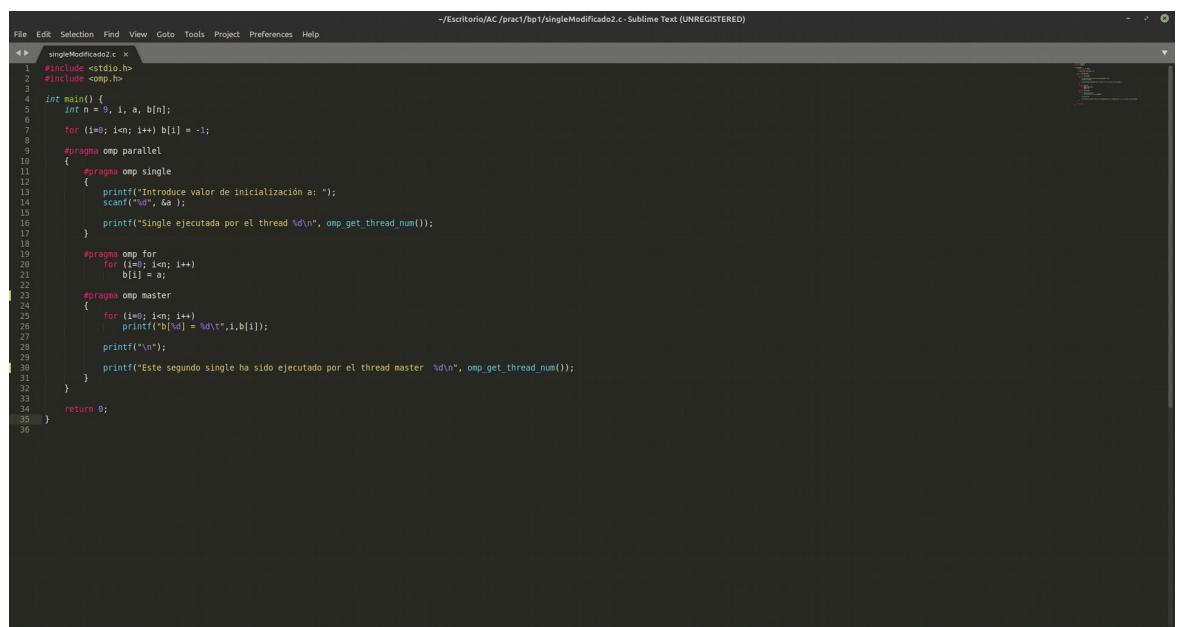


CAPTURAS DE PANTALLA:

```
nacho@GE63-Raider:~ - nacho@GE63-Raider:~ - nacho@GE63-Raider:~ -  
[IgnacioCarvalhoHerrera nacho@GE63-Raider:~/Escritorio/AC /praci/bpl] 2020-03-11 miércoles  
$export OMP_NUM_THREADS=8  
[IgnacioCarvalhoHerrera nacho@GE63-Raider:~/Escritorio/AC /praci/bpl] 2020-03-11 miércoles  
$OMP_NUM_THREADS=8  
$OMP_NUM_THREADS=8  
Introduce valor de inicialización: a: 23  
Single ejecutada por el thread 5: b[2] = 23      b[3] = 23      b[4] = 23      b[5] = 23      b[6] = 23      b[7] = 23      b[8] = 23  
Este segundo single ha sido ejecutado por el thread 5  
[IgnacioCarvalhoHerrera nacho@GE63-Raider:~/Escritorio/AC /praci/bpl] 2020-03-11 miércoles  
$
```

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



The screenshot shows a Sublime Text window with the following details:

- Title Bar:** Shows the file path: "/Escritorio/AC/pract/bpt/singleModificado2.c" and "Sublime Text (UNREGISTERED)".
- Menu Bar:** File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, Help.
- Code Area:** The main content displays a C program named "singleModificado2.c". The code includes several OpenMP directives: parallel regions (lines 10-11), single execution (line 12), and master threads (line 24). It also contains standard I/O operations like printf and scanf.
- Status Bar:** Shows "Line 35 Column 2" and "Tab Size: 4".
- System Icons:** A dock at the bottom includes icons for file operations, terminal, browser, and system status.

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

int main() {
    int n = 9, i, a, b[n];
    for (i=0; i<n; i++) b[i] = -1;
    #pragma omp parallel
    {
        #pragma omp single
        {
            printf("Introduce valor de inicialización a: ");
            scanf("%d", &a);
            printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
        }
        #pragma omp for
        for (i=0; i<n; i++)
            b[i] = a;
        #pragma omp master
        {
            for (i=0; i<n; i++)
                printf("%d[%d] = %d\n", i, b[i]);
            printf("\n");
        }
        printf("Este segundo single ha sido ejecutado por el thread master %d\n", omp_get_thread_num());
    }
    return 0;
}
```

CAPTURAS DE PANTALLA:

```

nacho@GE63-Raider-8SE: ~
nacho@GE63-Raider-8SE: ~
nacho@GE63-Raider-8SE: ~

[IgnacioCarvaljalHerrera nacho@GE63-Raider-8SE:~/Escritorio/AC /prac1/bpl] 2020-03-11 miércoles
$export OMP_NUM_THREADS=8
[IgnacioCarvaljalHerrera nacho@GE63-Raider-8SE:~/Escritorio/AC /prac1/bpl] 2020-03-11 miércoles
$bin/singleModificado2
Introduce valor de inicialización a: 26
Single ejecutada por el thread 2
b[0] = 26    b[1] = 26    b[2] = 26    b[3] = 26    b[4] = 26    b[5] = 26    b[6] = 26    b[7] = 26    b[8] = 26
Este segundo single ha sido ejecutado por el thread master. 0
[IgnacioCarvaljalHerrera nacho@GE63-Raider-8SE:~/Escritorio/AC /prac1/bpl] 2020-03-11 miércoles
$
```

RESPUESTA A LA PREGUNTA:

La diferencia con respecto a la ejecución del ejercicio 2 es que el thread que imprime los resultados de la ejecución del programa dentro de la construcción parallel es el thread master, es decir, la hebra numero 0.

Esto no influye sobre los resultados del programa.

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:

Se suman las componentes paralelizando las thread las sumas, la suma total es la suma de las sumas parciales y se actualiza la variable global compartida para ello se tiene que asegurar que se accede a ella de forma ordenada con critica,l en el ejemplo se usa atomic que es lo mismo. Como con parallel for hay barrera implícita pues entonces cuando se calcule la suma total se sabe que se ha realizado bien.

Pero en el ejemplo de master.c se quiere hacer dentro del código que paraleliza entonces como atomic no tiene barrera implícita entonces tenemos que usar barrier para saber que se respeta la sección crítica y asegurarnos de que suma tiene sumado todos los valores de las sumas locales. De no poner barrier la sección crítica se va a respetar pero se puede mostrar por pantalla el valor de la suma total faltando por sumar algunas sumas parciales.

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:

```

nacho@GE63-Raider-8SE: ~
nacho@GE63-Raider-8SE: ~
nacho@GE63-Raider-8SE: ~

[IgnacioCarvaljalHerrera d2estudiante6@atcgrid:~/bp1] 2020-03-11 miércoles
$time ./sumarVectoresC global 10000000
Stime srunti SumarVectoresC global 10000000
Tamaño Vectores:10000000 ( 4 )
Tiempo:0.84022094 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) / V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+1000000.200000=2000000.100000)
real   0m0.152s
user   0m0.007s
sys    0m0.007s
[IgnacioCarvaljalHerrera d2estudiante6@atcgrid:~/bp1] 2020-03-11 miércoles
$
```

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.

- Todos los apuntes que necesitas están aquí
 - Al mejor precio del mercado, desde **2 cent.**
 - Recoge los apuntes en tu copistería más cercana o recíbelos en tu casa
 - Todas las anteriores son correctas

Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática

La suma de los tiempos de CPU del usuario y del sistema es menor que el tiempo real ya que el tiempo que falta es el asociado a las esperas debidas a E/S o asociadas a la ejecución de otros programas.

0m0.007s + 0m0.007s = 0m0.014s es el 9.21% del tiempo transcurrido (0m0.152s)

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). 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):



RESPUESTA: cálculo de los MIPS y los MFLOPS.

Todo lo que hay entre las 2 llamadas a `clock_gettime` es la suma además de que aparece la etiqueta del bucle (L5)

Consultando un manual de código ensamblador observamos que las operaciones en coma flotante son movsd y addsd, es decir las 3 primeras de .L5

El valor del número de instrucciones viene dado por la expresión $NI = 3 + 6*N$

Desde el comienzo de la etiqueta .L5 hasta el salto a ésta hay seis instrucciones. Por lo tanto tendremos que multiplicar por 6 el número de componentes. Le sumamos 3 a la expresión debido a las tres instrucciones que hay al acabar el bucle y hasta llamar a `clock_gettime`.

Tamaño de 10 componentes.

$$NI = 6*10 + 3 = 63$$

$$\text{Oper. Coma flotante} = 3 * 10 = 30$$

$$\text{Tiempo} = 0.000369806 \text{ s}$$

$$\text{MIPS} = 63 / (0.000369806 * 10^6) = 0.17035959$$

$$\text{MFLOPS} = 30 / (0.000369806 * 10^6) = 0.081123616$$

Tamaño de 10000000 componentes.

$$NI = 6*10000000 + 3 = 60000003$$

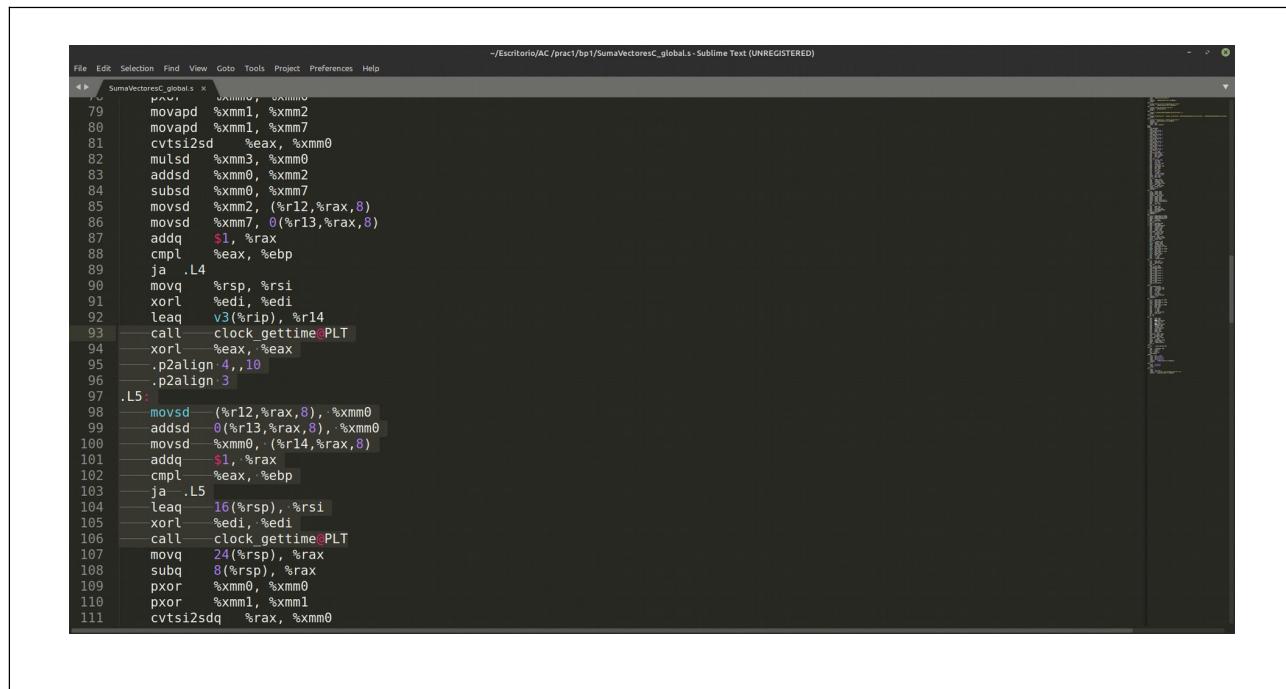
$$\text{Oper. Coma flotante} = 3 * 10000000 = 30000000$$

$$\text{Tiempo} = 0.042068983$$

$$\text{MIPS} = 60000003 / (0.042068983 * 10^6) = 1426.2289$$

$$\text{MFLOPS} = 30000000 / (0.042068983 * 10^6) = 713.1144577$$

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



The screenshot shows a Sublime Text editor window with the file "SumaVectoresC_global.s" open. The assembly code is as follows:

```

File Edit Selection Find View Goto Tools Project Preferences Help
SumaVectoresC_global.s x -/Escritorio/AC/pract/bp1/SumaVectoresC_global.s - Sublime Text (UNREGISTERED)
48    movsd  %xmm0, %xmm0
49    movapd %xmm1, %xmm2
50    movapd %xmm1, %xmm7
51    cvtsi2sd  %eax, %xmm0
52    mulsd  %xmm3, %xmm0
53    addsd  %xmm0, %xmm2
54    subsd  %xmm0, %xmm7
55    movsd  %xmm2, (%r12,%rax,8)
56    movsd  %xmm1, 0(%r13,%rax,8)
57    addq   $1, %rax
58    cmpl  %eax, %ebp
59    ja  .L4
60    movq  %rsp, %rsi
61    xorl  %edi, %edi
62    leaq   v3(%rip), %rl4
63    call  clock_gettime@PLT
64    xorl  %eax, %eax
65    .p2align 4,,10
66    .p2align 3
67 .L5:
68    movsd  (%r12,%rax,8), %xmm0
69    addsd  0(%r13,%rax,8), %xmm0
70    movsd  %xmm0, (%r14,%rax,8)
71    addq   $1, %rax
72    cmpl  %eax, %ebp
73    ja  .L5
74    leaq   16(%rsp), %rsi
75    xorl  %edi, %edi
76    call  clock_gettime@PLT
77    movq  24(%rsp), %rax
78    subq  8(%rsp), %rax
79    pxor  %xmm0, %xmm0
80    pxor  %xmm1, %xmm1
81    cvtsi2sdq %rax, %xmm0

```

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 parallelizar 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

```

File Edit Selection Find View Goto Tools Project Preferences Help
-SumaVectoresC_globalModificado.c - /Escritorio/A/C/prac1/bp1/SumaVectoresC_globalModificado.c - Sublime Text (UNREGISTERED)
46 }
47
48 unsigned int N = atoi(argv[1]); // Máximo N = 2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
49 printf("Tamaño Vectores:%u (%u B)\n",N, sizeof(unsigned int));
50 #ifdef VECTOR_DYNAMIC
51 double v1[N], v2[N], v3[N]; // Tamaño variable local en tiempo de ejecución ...
52 #endif
53 #endif
54 #ifndef VECTOR_GLOBAL
55 #define MAX_N 100000000
56 #endif
57 #ifndef VECTOR_DYNAMIC
58 double *v1, *v2, *v3;
59 v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño en bytes
60 v2 = (double*) malloc(N*sizeof(double));
61 v3 = (double*) malloc(N*sizeof(double));
62 if ((v1 == NULL) || (v2 == NULL) || (v3 == NULL)) {
63     printf("No hay suficiente espacio para los vectores \n");
64     exit(-2);
65 } endif
66 #endif
67 //Inicializar vectores
68 #pragma omp parallel for
69 for(i=0; i<N; i++){
70     v1[i] = N*0.1*i*0.1; v2[i] = N*0.1-i*0.1;
71 }
72
73 start = omp_get_wtime();
74
75 //Calcular suma de vectores
76 #pragma omp parallel for
77 for(i=0; i<N; i++)
78     v3[i] = v1[i] + v2[i];
79
80 end = omp_get_wtime();
81
82 ngt = end - start;
83
84 //Imprimir resultado de la suma y el tiempo de ejecución
85 if (N>0) {
86     printf("Tiempo:%11.9f\n / Tamaño Vectores:%u\n",ngt,N);
87     for(i=0; i<N; i++)
88         printf(" / V1[%d]+V2[%d]=V3[%d] (%8.6f+%8.6f=%8.6f) \n",
89                i,i,v1[i],v2[i],v3[i]);
90 }
91
92 else
93     printf("Tiempo:%11.9f\n / Tamaño Vectores:%u\n / V1[0]+V2[0]=V3[0] (%8.6f+%8.6f=%8.6f) / V1[%d]+V2[%d]=V3[%d] (%8.6f+%8.6f=%8.6f) \n",
94                N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
95
96
97 #endif VECTOR_DYNAMIC

```

Line 85, Column 59 Tab Size: 4 C

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

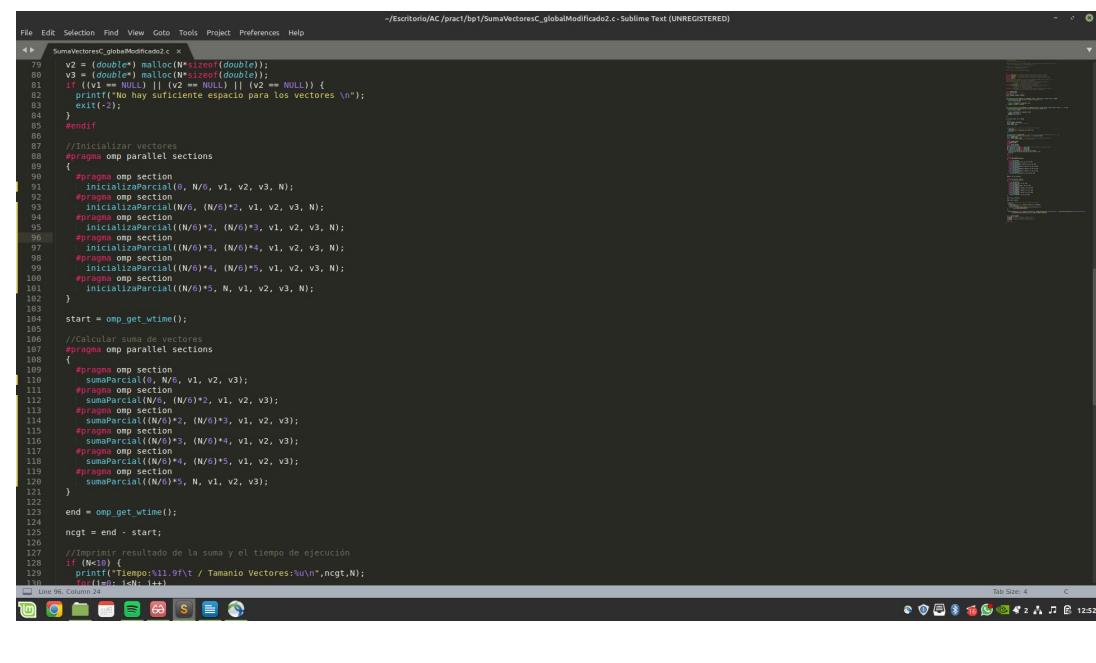
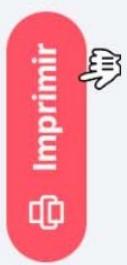
```
nacho@GE63-Raider-8SE:~/Escritorio/AC/pract1/bp1
Archivo Editar Ver Buscar Terminal Ayuda
[IgnacioCarvajalHerrera nacho@GE63-Raider-8SE:~/Escritorio/AC/pract1/bp1] 2020-03-19 jueves
snake
gcc -O2 -fopenmp ./SumaVectoresC_globalModificado.c -o bin/SumaVectoresC_globalModificado -lrt
./SumaVectoresC_globalModificado.c: In function `main':
./SumaVectoresC_globalModificado.c:49:33: warning: format `"%u"' expects argument of type `unsigned int', but argument 3 has type `long unsigned int' [-Wformat=]
    printf("Tamaño Vectores:%u (%u B)\n",N,sizeof(unsigned int));
          ^~~~~
[IgnacioCarvajalHerrera nacho@GE63-Raider-8SE:~/Escritorio/AC/pract1/bp1] 2020-03-19 jueves
$bin/SumaVectoresC_globalModificado 8
Tamaño Vectores:8 (% 8 B)
Tiempo: 0.000023098
[Tamaño Vectores:8
 / V1[0]+V2[0]=V3[0](0.800000+0.800000i-1.600000) /
 / V1[1]+V2[1]=V3[1](0.800000+0.700000i-0.600000) /
 / V1[2]+V2[2]=V3[2](1.100000+0.500000i-0.100000) /
 / V1[3]+V2[3]=V3[3](1.100000+0.500000i-1.600000) /
 / V1[4]+V2[4]=V3[4](1.200000+0.400000i-1.600000) /
 / V1[5]+V2[5]=V3[5](1.100000+0.400000i-0.100000) /
 / V1[6]+V2[6]=V3[6](1.400000+0.200000i-1.600000) /
 / V1[7]+V2[7]=V3[7](1.500000+0.100000i-1.600000) /
[IgnacioCarvajalHerrera nacho@GE63-Raider-8SE:~/Escritorio/AC/pract1/bp1] 2020-03-19 jueves
$bin/SumaVectoresC_globalModificado 11
Tamaño Vectores:11 (% 11 B)
Tiempo: 0.000007770
[Tamaño Vectores:11
 / V1[0]+V2[0]=V3[0](1.100000+1.200000i-2.200000) /
 / V1[1]+V2[1]=V3[1](2.100000+0.100000i-2.000000) /
[IgnacioCarvajalHerrera nacho@GE63-Raider-8SE:~/Escritorio/AC/pract1/bp1] 2020-03-19 jueves
$
```

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

- Todos los apuntes que necesitas están aquí
 - Al mejor precio del mercado, desde **2 cent.**
 - Recoge los apuntes en tu copistería más cercana o recíbelos en tu casa
 - Todas las anteriores son correctas

Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática



(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

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 que usamos las directivas parallel y for, el reparto de trabajo entre threads es automático. El número de threads depende del valor que tenga la variable OMP_NUM_THREADS. Por ejemplo, si OMP_NUM_THREADS=4 el for se ejecutará en paralelo por parte de las 4 hebras, siempre y cuando el número de iteraciones sea mayor o igual que el valor de esta variable de entorno.

La asignación de hebras a iteraciones es:

iteraciones/num_threads y el resto se reparte consecutivamente entre las hebras.

En el ejercicio 8, en el que usamos las directivas parallel y sections, el trabajo lo realizaran tantas threads como secciones tengamos, si solo tenemos 4 secciones puess solo podemos usar 4 threads a pesar de que hayamos hecho previamente `OMP_NUM_THREADS=8`. Si por el contrario, al programa que usa el for le asignamos 12 threads entonces el bucle lo lanzan 12 threads, esa es la diferencia entre las 2 directivas.

Con sections como lo que estamos mandando son funciones o bloques de código pues entonces el reparto hace el programador de forma explícita.

Para darnos cuenta de esto, podemos hacer export OMP_NUM_THREADS=4 y lanzar el programa sections.c y observar como solo se usan 2 threads y el resto de threads no se pueden aprovechar.

Sections NO REPARTE DE FORMA DINAMICA.

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.

Tabla de tiempos de ejecución en mi PC

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 6 threads/cores	T. paralelo (versión sections) 6 threads/cores
16384	0,000190045	0,000084471	0,000076975
32768	0,000375032	0,000126911	0,000117894
65536	0,000839076	0,000207142	0,000235574
131072	0,001139920	0,000078188	0,000102221
262144	0,000729018	0,000178210	0,000176871
524288	0,001563546	0,000739064	0,000767115
1048576	0,003366435	0,001823192	0,001759171
2097152	0,007041477	0,003557105	0,003444979
4194304	0,013660131	0,006611004	0,006548425
8388608	0,026643879	0,013005072	0,012980130
16777216	0,051358160	0,025937472	0,025442404
33554432	0,100514741	0,051249230	0,051372230
67108864	0,100790391	0,051499432	0,052533365

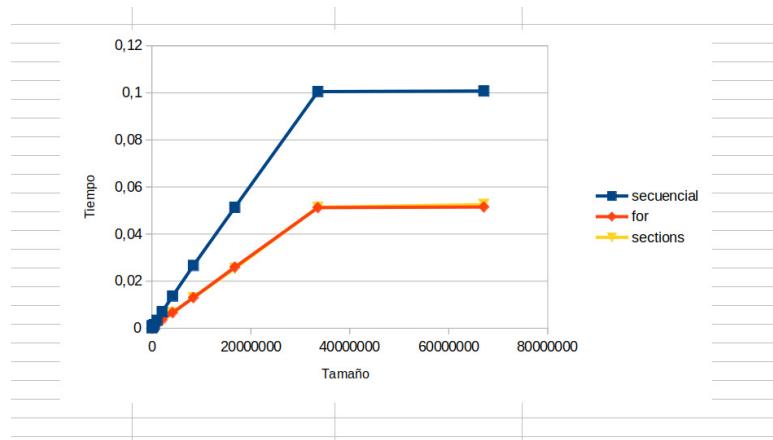
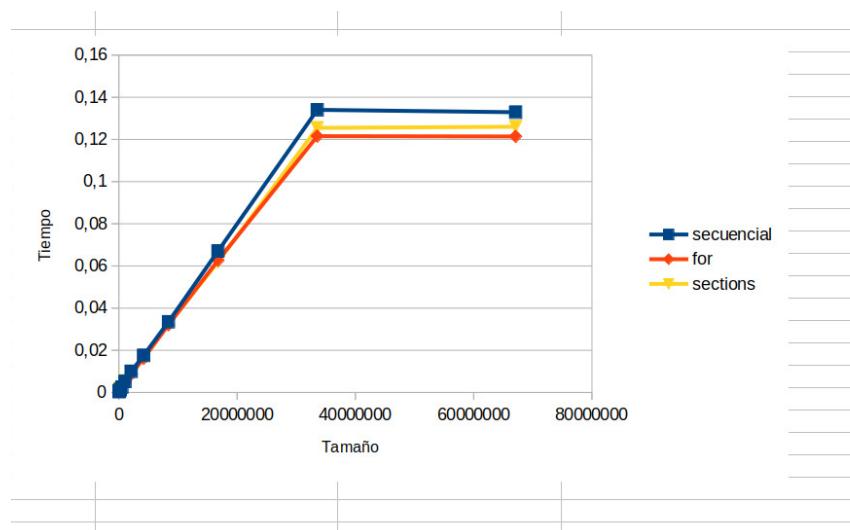


Tabla de tiempos de ejecución para atcgrid

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 6 threads/cores	T. paralelo (versión sections) 6 threads/cores
16384	0,000405580	0,000594202	0,000559209
32768	0,000404251	0,000614922	0,000570366
65536	0,000506079	0,000524860	0,001152063
131072	0,000648967	0,000783708	0,001320418
262144	0,001351708	0,001358787	0,001799067
524288	0,002551364	0,002452519	0,002492221
1048576	0,005203819	0,004904266	0,005501581
2097152	0,010015630	0,008716540	0,009313658
4194304	0,017574304	0,016206751	0,016350947
8388608	0,033516979	0,032110965	0,032422625
16777216	0,067098667	0,062681235	0,061939908
33554432	0,134049690	0,121554499	0,125496328
67108864	0,132928373	0,121426435	0,126042314



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.

- Todos los apuntes que necesitas están aquí
- Al mejor precio del mercado, desde **2 cent**.
- Recoge los apuntes en tu copistería más cercana o recíbelos en tu casa
- Todas las anteriores son correctas

Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática

Nº de Componentes	Tiempo secuencial vect. Globales 1 thread/core			Tiempo paralelo/versión for 6 Threads/cores		
	Elapsed	CPU-user	CPU-sys	Elapsed	CPU-user	CPU-sys
65536	0m0.087s	0m0.010s	0m0.009s	0m0.109s	0m0.011s	0m0.007s
131072	0m0.105s	0m0.010s	0m0.008s	0m0.105s	0m0.008s	0m0.010s
262144	0m0.084s	0m0.007s	0m0.011s	0m0.116s	0m0.009s	0m0.010s
524288	0m0.126s	0m0.010s	0m0.008s	0m0.088s	0m0.009s	0m0.010s
1048576	0m0.095s	0m0.011s	0m0.008s	0m0.092s	0m0.009s	0m0.009s
2097152	0m0.101s	0m0.009s	0m0.009s	0m0.101s	0m0.008s	0m0.010s
4194304	0m0.151s	0m0.011s	0m0.007s	0m0.160s	0m0.009s	0m0.009s
8388608	0m0.195s	0m0.012s	0m0.007s	0m0.182s	0m0.008s	0m0.010s
16777216	0m0.263s	0m0.007s	0m0.009s	0m0.284s	0m0.009s	0m0.008s
33554432	0m0.430s	0m0.008s	0m0.006s	0m0.398s	0m0.009s	0m0.006s
67108864	0m0.436s	0m0.007s	0m0.008s	0m0.377s	0m0.004s	0m0.010s

Es evidente que la suma del tiempo de usuario (tiempo de ejecución en espacio de usuario) y el tiempo del sistema (tiempo en el nivel del kernel del SO) es menor que el tiempo real. Esto se debe a que existe un tiempo asociado a las esperas debidas a E/S o asociado a la ejecución de otros programas.

Si aplicamos la fórmula de la ganancia:

$$T_1 = \text{Media aritmética tiempo ejecución secuencial} = 0m0.188$$

$$T_p = \text{Media aritmética tiempo ejecución paralelizado} = 0m0.182$$

$$S = T_1 / T_p = 0m0.188 / 0m0.182 = 1.032$$

Por lo que podemos ver que hay algo de ganancia.