

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 Ramirez Pulido

Grupo de prácticas y profesor de prácticas: c2 José Sánchez Garrido

Fecha de entrega:

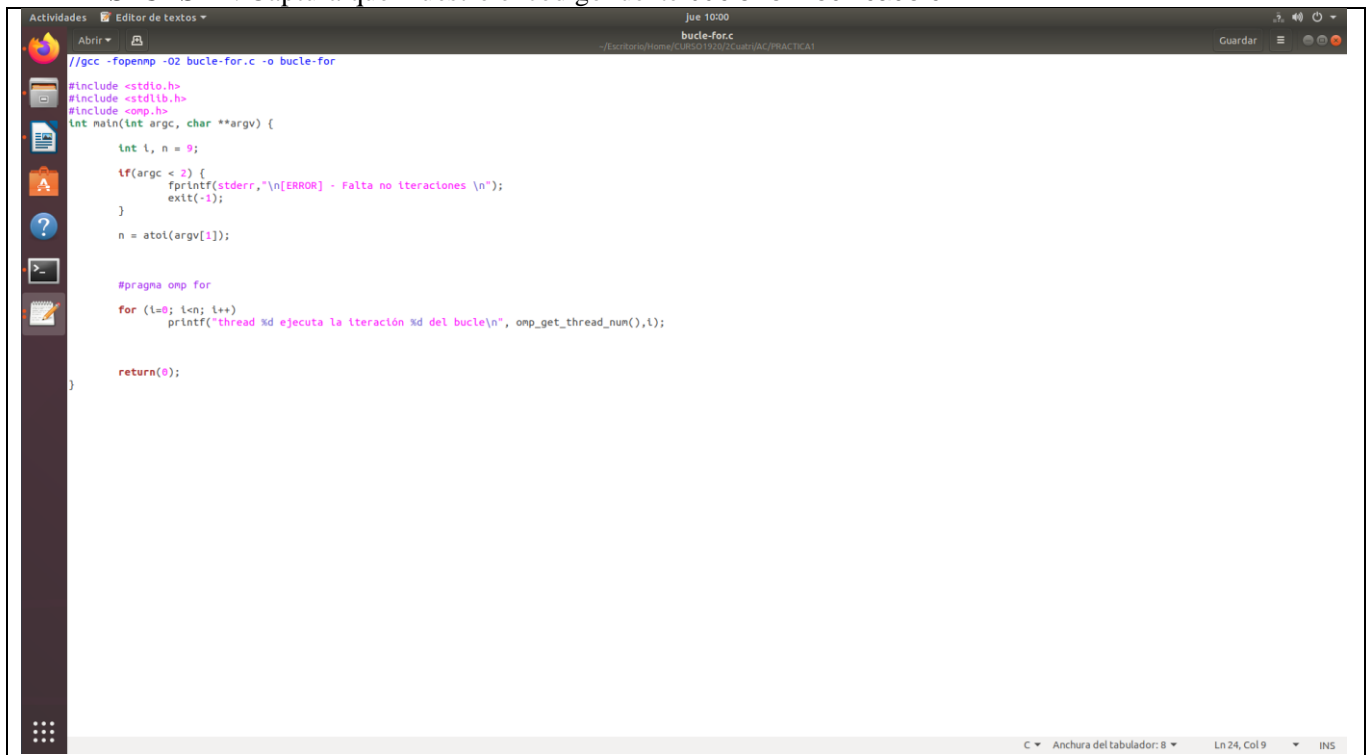
Fecha evaluación en clase:

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



```
//gcc -fopenmp -O2 bucle-for.c -o bucle-for
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main(int argc, char **argv) {
    int i, n = 0;
    if(argc < 2) {
        fprintf(stderr, "\n[ERROR] - Falta no iteraciones \n");
        exit(-1);
    }
    n = atoi(argv[1]);
    #pragma omp 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

```

//gcc -fopenmp -O2 sections.c -o sections
#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());
}
int natn() {
    #pragma omp sections
    {
        #pragma omp section
        (void) funcA();
        #pragma omp section
        (void) funcB();
    }
    return 0;
}

```

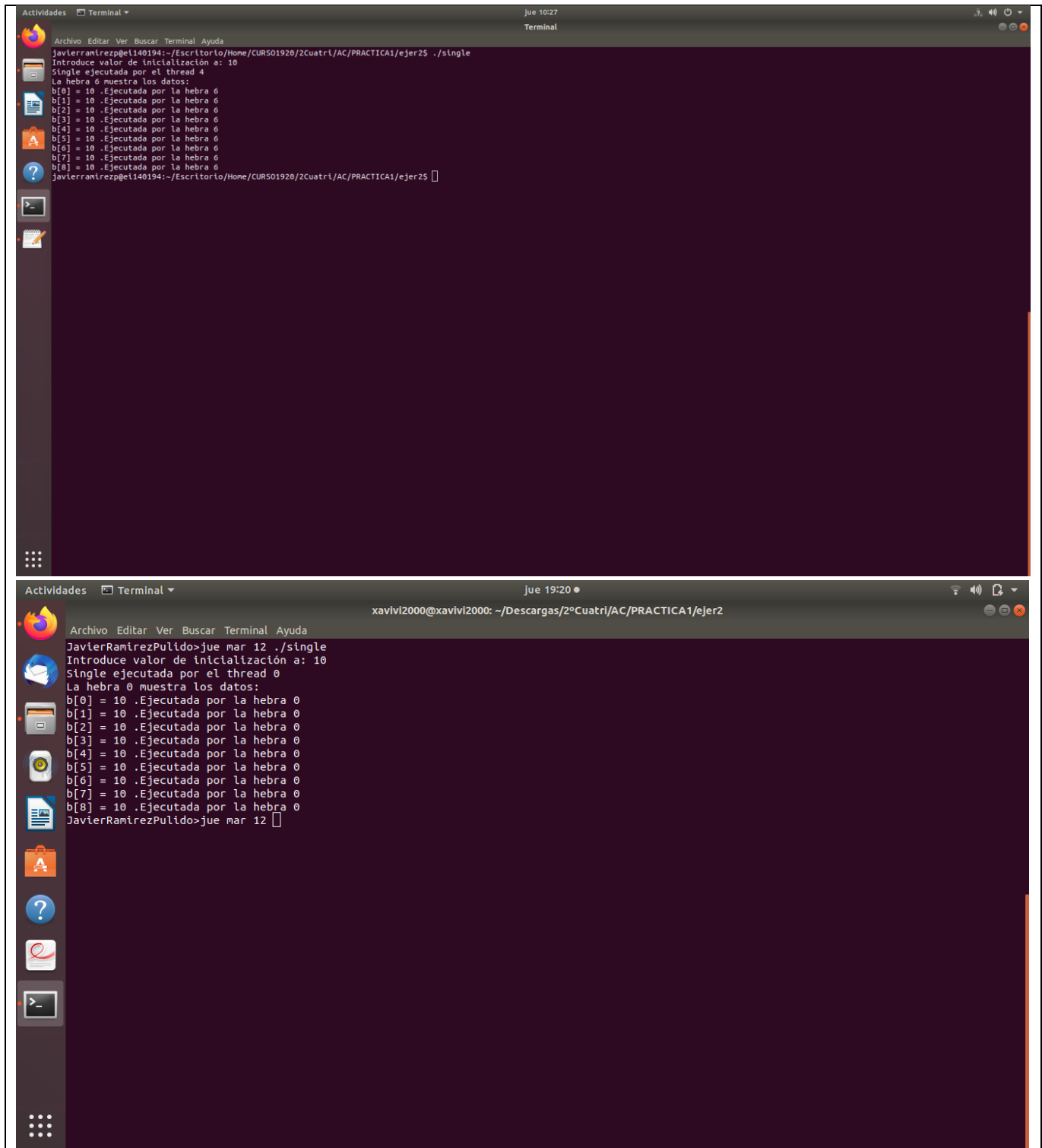
- Imprimir los resultados del programa single.c usando una directiva single dentro de la construcción parallel en lugar de imprimirlos fuera de la región parallel. Añadir lo necesario, dentro de la nueva directiva single incorporada, para que se imprima el identificador del thread que ejecuta el bloque 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`

```

//gcc -fopenmp -O2 single.c -o single
#include <stdio.h>
#include <omp.h>
int natn() {
    int n = 9, i, a, b[n];
    for (i=0; i<n; i++)
        b[i] = -i;
    #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("La hebra %d muestra los datos:\n", omp_get_thread_num());
            for(i=0; i<n; i++) printf("b[%d] = %d .Ejecutada por la hebra %d\n", i, b[i], omp_get_thread_num());
        }
    }
}

```



```
Actividades Terminal Jue 10:27
Archivo Editar Ver Buscar Terminal Ayuda
javerramirezp@140194:~/Escritorio/Home/CURSO1920/2Cuatri/AC/PRACTICA1/ejer2$ ./single
Introduce valor de inicialización a: 10
Single ejecutada por el thread 4
La hebra 6 muestra los datos:
b[0] = 10 .Ejecutada por la hebra 6
b[1] = 10 .Ejecutada por la hebra 6
b[2] = 10 .Ejecutada por la hebra 6
b[3] = 10 .Ejecutada por la hebra 6
b[4] = 10 .Ejecutada por la hebra 6
b[5] = 10 .Ejecutada por la hebra 6
b[6] = 10 .Ejecutada por la hebra 6
b[7] = 10 .Ejecutada por la hebra 6
b[8] = 10 .Ejecutada por la hebra 6
javerramirezp@140194:~/Escritorio/Home/CURSO1920/2Cuatri/AC/PRACTICA1/ejer2$

Actividades Terminal Jue 19:20
xavivi2000@xavivi2000: ~/Descargas/2ºCuatri/AC/PRACTICA1/ejer2
Archivo Editar Ver Buscar Terminal Ayuda
JavierRamirezPulido>jue mar 12 ./single
Introduce valor de inicialización a: 10
Single ejecutada por el thread 0
La hebra 0 muestra los datos:
b[0] = 10 .Ejecutada por la hebra 0
b[1] = 10 .Ejecutada por la hebra 0
b[2] = 10 .Ejecutada por la hebra 0
b[3] = 10 .Ejecutada por la hebra 0
b[4] = 10 .Ejecutada por la hebra 0
b[5] = 10 .Ejecutada por la hebra 0
b[6] = 10 .Ejecutada por la hebra 0
b[7] = 10 .Ejecutada por la hebra 0
b[8] = 10 .Ejecutada por la hebra 0
JavierRamirezPulido>jue mar 12
```

La línea `#pragma omp single` provoca que una hebra sea la que ejecute la impresión de los datos resultantes, además de ser una sola hebra la que realiza el bucle `for` completo.

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 Linux desktop environment with a text editor and a terminal window.

Text Editor (single.c):

```
//gcc -fopenmp -O2 single.c -o single

#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
        {
            printf("La hebra %d muestra los datos:\n", omp_get_thread_num());
            for(i=0;i<n;i++) printf("b[%d] = %d .Ejecutada por la hebra %d\n",i,b[i],omp_get_thread_num());
        }

    }

}
```

Terminal:

```
xavivi2000@xavivi2000: ~/Descargas/2ºCuatri/AC/PRACTICA1/ejer3/single.c...
JavierRamirezPulido>jue mar 12 ./single
Introduce valor de inicialización a: 23
Single ejecutada por el thread 0
La hebra 0 muestra los datos:
b[0] = 23 .Ejecutada por la hebra 0
b[1] = 23 .Ejecutada por la hebra 0
b[2] = 23 .Ejecutada por la hebra 0
b[3] = 23 .Ejecutada por la hebra 0
b[4] = 23 .Ejecutada por la hebra 0
b[5] = 23 .Ejecutada por la hebra 0
b[6] = 23 .Ejecutada por la hebra 0
b[7] = 23 .Ejecutada por la hebra 0
b[8] = 23 .Ejecutada por la hebra 0
JavierRamirezPulido>jue mar 12 ./single
Introduce valor de inicialización a: 3
Single ejecutada por el thread 0
La hebra 0 muestra los datos:
b[0] = 3 .Ejecutada por la hebra 0
b[1] = 3 .Ejecutada por la hebra 0
b[2] = 3 .Ejecutada por la hebra 0
b[3] = 3 .Ejecutada por la hebra 0
b[4] = 3 .Ejecutada por la hebra 0
b[5] = 3 .Ejecutada por la hebra 0
b[6] = 3 .Ejecutada por la hebra 0
b[7] = 3 .Ejecutada por la hebra 0
b[8] = 3 .Ejecutada por la hebra 0
JavierRamirezPulido>jue mar 12
```

RESPUESTA A LA PREGUNTA:

La hebra principal (también conocida como master) es la encargada de imprimir los resultados por la especificación `#pragma omp master`, pero al no tener barreras de forma implícita (como `#pragma omp single`) puede que los resultados sean erróneos.

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:

Por la falta de las barreras implícitas en la directiva `master`. Esto provoca la posibilidad de errores ya que la hebra principal encargada de la suma puede terminar antes de que lo hagan las demás. `Barrier` las detiene forzándolas a esperarse entre ellas.

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:

The screenshot shows a terminal window with the following content:

```

Actividades Terminal
c3estudiante18@atcgrid:~/bp1/ejer5
xavivi2000@xavivi2000: ~
JavierRamirezPulido>jue mar 12 time sbatch -p ac --wrap "SumaVectores" 10000000
sbatch: error: Script arguments not permitted with --wrap option

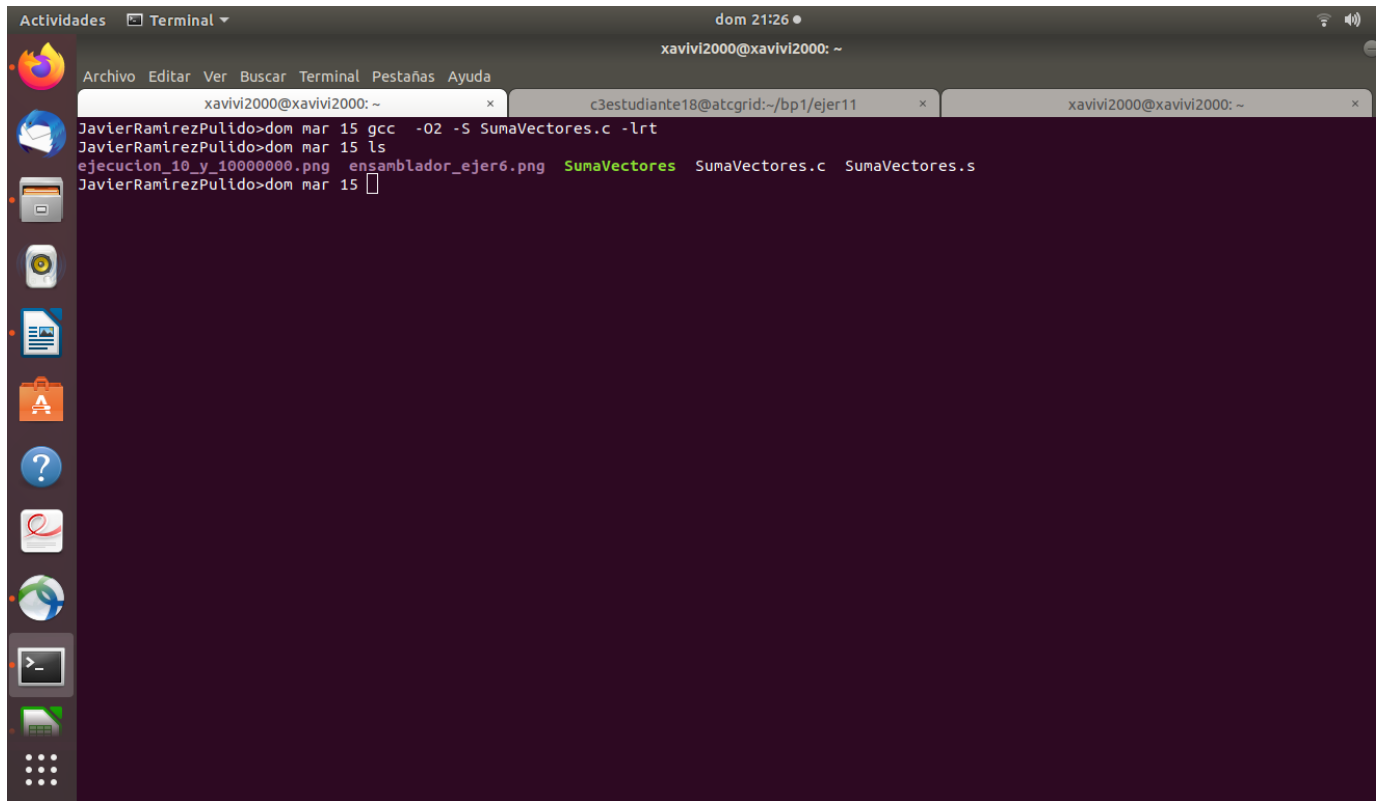
real    0m0.006s
user    0m0.004s
sys     0m0.002s
JavierRamirezPulido>jue mar 12
  
```

El tiempo de CPU es la suma del tiempo de CPU del usuario y del sistema. Además, el tiempo de ejecución es la suma del tiempo de CPU total + tiempo asociado entradas, salidas o interrupciones. De esa forma, el tiempo de CPU puede ser igual o menor que el real, pero nunca mayor. En mi caso, el tiempo de CPU es 0.064 (usuario) + 0.046(sistema) = 0.110, pero el tiempo real es 0.122s, lo cual significa que hay 0.012s perdidos que no son de CPU.

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):



```

dom 21:26
xavivi2000@xavivi2000: ~
Archivo Editar Ver Buscar Terminal Pestañas Ayuda
xavivi2000@xavivi2000: ~
c3estudiante18@atcgrid:~/bp1/ejer11
xavivi2000@xavivi2000: ~
JavierRamirezPulido>dom mar 15 gcc -O2 -S SumaVectores.c -lrt
JavierRamirezPulido>dom mar 15 ls
ejecucion_10_y_10000000.png ensamblador_ejer6.png SumaVectores SumaVectores.c SumaVectores.s
JavierRamirezPulido>dom mar 15

```



```

SumaVectores.s
~/Descargas/2°Cuatri/AC/PRACTICA1/ejer6
Guardar
subsd    %xmm0, %xmm7
movsd    %xmm2, (%r12,%rax,8)
movsd    %xmm7, 0(%r13,%rax,8)
addq     $1, %rax
cmpl     %eax, %ebp
ja       .L4
movq     %rsp, %rsi
xorl     %edi, %edi
leaq     v3(%rip), %r14
call     clock_gettime@PLT
xorl     %eax, %eax
.p2align 4,,10
.p2align 3
.L5:
movsd    (%r12,%rax,8), %xmm0
addsd    0(%r13,%rax,8), %xmm0
movsd    %xmm0, (%r14,%rax,8)
addq     $1, %rax
cmpl     %eax, %ebp
ja       .L5
leaq     16(%rsp), %rsi
xorl     %edi, %edi
call     clock_gettime@PLT
movq     24(%rsp), %rax
subq     8(%rsp), %rax
pxor     %xmm0, %xmm0
pxor     %xmm1, %xmm1
cvtsi2sdq %rax, %xmm0
movq     16(%rsp), %rax
subq     (%rsp), %rax
cmpl     $9, %ebx
cvtsi2sdq %rax, %xmm1
divsd    .LC5(%rip), %xmm0
addsd    %xmm1, %xmm0
jbe      .L22
leal     -1(%rbp), %eax
movsd    (%r14), %xmm3

```

```

Actividades Terminal
c3estudiante18@atcgriid:~/bp1/ejer6

Archivo Editar Ver Buscar Terminal Pestañas Ayuda

xavivi2000@xavivi2000: ~
xavivi2000@xavivi2000: ~
c3estudiante18@atcgriid:~/bp1/ejer6
xavivi2000@xavivi2000: ~

JavierRamirezPulido>vie mar 13 time ./SumaVectores 10
Tamaño Vectores:10 (4 B)
Tiempo:0.000204571 / 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    0m0.002s
user    0m0.000s
sys     0m0.002s
JavierRamirezPulido>vie mar 13 time ./SumaVectores 10000000
Tamaño Vectores:10000000 (4 B)
Tiempo:0.014960026 / 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    0m0.040s
user    0m0.026s
sys     0m0.014s
JavierRamirezPulido>vie mar 13

```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Fórmulas necesarias:

$$\text{MIPS} = \text{NI} / (\text{Tcpu} * 10^6)$$

$$\text{MFLOPS} = \text{Operaciones_en_coma_flotante} / (\text{Tcpu} * 10^6)$$

CALCULO PARA NUMERO DE ITERACIONES 10:

Tras la ejecución vemos el tiempo que tarda el bucle que será el Tcpu. En este caso es: $0.000 + 0.002 = 0.002\text{s}$

Para el numero de instrucciones miramos en el .s del ensamblador. Entre las dos llamadas a clock_gettime ocurre el bucle que tiene dentro 6 instrucciones que se ejecutan el número de veces que pongas de iteraciones. Además tiene encima 3 instrucciones y 2 despues. Entones $\text{NI} = 6 * \text{N} + 5$. Como las iteraciones son 10, $\text{NI} = 65$.

$$\text{MIPS} = \text{NI} / (\text{Tcpu} * 10^6) = 65 / (0.002 * 10^6) = 0.0325$$

Para los MFLOPS miramos el manual en ensamblador para saber las operaciones con coma flotante que son movsd (2 veces) y addsd. Como están dentro de un ciclo, es $3 * \text{N}$ iteraciones. Operaciones con coma flotante = $3 * 10 = 30$

$$\text{MFLOPS} = \text{Operaciones_en_coma_flotante} / (\text{Tcpu} * 10^6) = 30 / (0.002 * 10^6) = 0.015$$

CALCULO PARA NUMERO DE ITERACIONES 10000000:

Tras la ejecución vemos el tiempo que tarda el bucle que será el Tcpu. En este caso es: $0.026 + 0.014 = 0.040\text{s}$

Para el numero de instrucciones miramos en el .s del ensamblador. Entre las dos llamadas a clock_gettime ocurre el bucle que tiene dentro 6 instrucciones que se ejecutan el número de veces que pongas de iteraciones. Además tiene encima 3 instrucciones y 2 despues. Entones $\text{NI} = 6 * \text{N} + 5$. Como las iteraciones son 10000000, $\text{NI} = 60000005$.

$$\text{MIPS} = \text{NI} / (\text{Tcpu} * 10^6) = 60000005 / (0.040 * 10^6) = 1500.00013$$

Para los MFLOPS miramos el manual en ensamblador para saber las operaciones con coma flotante que son movsd (2 veces) y addsd. Como están dentro de un ciclo, es $3 \cdot N$ iteraciones. Operaciones con coma flotante = $3 \cdot 10000000 = 30000000$

$\text{MFLOPS} = \text{Operaciones_en_coma_flotante} / (\text{Tcpu} \cdot 10^6) = 30000000 / (0.040 \cdot 10^6) = 750$

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

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

```

//Inicializar vectores
#pragma omp parallel for
for(i=0; i<N; i++){
    v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
}

double tiempo1 = omp_get_wtime();

//Calcular suma de vectores
#pragma omp parallel for
for(i=0; i<N; i++){
    v3[i] = v1[i] + v2[i];
}

double tiempo2 = omp_get_wtime();

double tiempo_total = tiempo2 - tiempo1;

//Imprimir resultado de la suma y el tiempo de ejecución
#ifdef PRINTF_ALL
printf("Tiempo:%11.9f\t / Tamaño Vectores:%u\n", tiempo_total, N);
for(i=0; i<N; i++){
    printf(" / v1[%d]+v2[%d]=v3[%d](%8.6f+%8.6f=%8.6f) /\n",
        i, i, v1[i], v2[i], v3[i]);
}
#else
printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n", tiempo_total, N);
for(int i = 0; i<N; i++){
    printf("v1[%d]+v2[%d]=v3[%d](%8.6f+%8.6f=%8.6f)\n", i, i, v1[i], v2[i], v3[i]);
}
#endif

return 0;
  
```



```

xavivi2000@xavivi2000: ~
JavierRamirezPulido>vie mar 13 gcc -O2 -fopenmp SumaVectores.c -o SumaVectores
SumaVectores.c: In function 'main':
SumaVectores.c:70:20: warning: implicit declaration of function 'omp_get_wtime'; did you mean 'timer_gettime'? [-Wimplicit-function-declariatio
n]
double tiempo1 = omp_get_wtime();
                    ^~~~~~
                    timer_gettime
JavierRamirezPulido>vie mar 13 ./SumaVectores 8
Tamaño Vectores:8 (4 B)
Tiempo(seg.):0.000000000 / 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)
JavierRamirezPulido>vie mar 13 ./SumaVectores 11
Tamaño Vectores:11 (4 B)
Tiempo(seg.):0.000000000 / Tamaño Vectores:11
V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000)
V1[1]+V2[1]=V3[1](1.200000+1.000000=2.200000)
V1[2]+V2[2]=V3[2](1.300000+0.900000=2.200000)
V1[3]+V2[3]=V3[3](1.400000+0.800000=2.200000)
V1[4]+V2[4]=V3[4](1.500000+0.700000=2.200000)
V1[5]+V2[5]=V3[5](1.600000+0.600000=2.200000)
V1[6]+V2[6]=V3[6](1.700000+0.500000=2.200000)
V1[7]+V2[7]=V3[7](1.800000+0.400000=2.200000)
V1[8]+V2[8]=V3[8](1.900000+0.300000=2.200000)
V1[9]+V2[9]=V3[9](2.000000+0.200000=2.200000)
V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000)
JavierRamirezPulido>vie mar 13

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

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

```

}
#endif

//Inicializar vectores
double tiempo_inicial, tiempo_final, tiempo_total;
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        for(i=0; i<N/4; i++){
            v1[i] = N*0.1+i*0.1;
            v2[i] = N*0.1-i*0.1;
        }

        #pragma omp section
        for(j=N/4; j<N/2; j++){
            v1[j] = N*0.1+j*0.1;
            v2[j] = N*0.1-j*0.1;
        }

        #pragma omp section
        for(k=N/2; k<3*N/4; k++){
            v1[k] = N*0.1+k*0.1;
            v2[k] = N*0.1-k*0.1;
        }

        #pragma omp section
        for(l=3*N/4; l<N; l++){
            v1[l] = N*0.1+l*0.1;
            v2[l] = N*0.1-l*0.1;
        }
    }

    tiempo_inicial=omp_get_wtime();
}

```

```

tiempo_inicial=omp_get_wtime();

#pragma omp sections
{
    #pragma omp section
    for(i=0; i<N/4; i++){
        v3[i] = v1[i]+v2[i];
    }

    #pragma omp section
    for(j=N/4; j<N/2; j++){
        v3[j] = v1[j]+v2[j];
    }

    #pragma omp section
    for(k=N/2; k<3*N/4; k++){
        v2[k] = v1[k]+v2[k];
    }

    #pragma omp section
    for(l=3*N/4; l<N; l++){
        v2[l] = v1[l]+v2[l];
    }
}

tiempo_final=omp_get_wtime();
}

tiempo_total = tiempo_final - tiempo_inicial;

//Imprimir resultado de la suma y el tiempo de ejecución
#ifdef PRINTF_ALL
printf("Tiempo:%11.9f\t / Tamaño Vectores:%u\n", tiempo_total, N);
for(i=0; i<N; i++)
    printf("/ v1[%d]+v2[%d]=v3[%d](%8.6f+%8.6f=%8.6f) /\n",
        i, i, i, v1[i], v2[i], v3[i]);
#endif

```

The screenshot shows a Linux desktop with a text editor (gedit) and a terminal window. The text editor is open to a file named 'SumaVectores.c' located at '~/.Descargas/2ºCuatr/AC/PRACTICA1/ejer8'. The code in the editor is as follows:

```
#pragma omp section
for(k=N/2; k<3*N/4; k++)
    v2[k] = v1[k]+v2[k];

#pragma omp section
for(l=3*N/4; l<N; l++)
    v2[l] = v1[l]+v2[l];
}

tiempo_final=omp_get_wtime();
}

tiempo_total = tiempo_final - tiempo_inicial;

//Imprimir resultado de la suma y el tiempo de ejecución
#ifdef PRINTF_ALL
printf("Tiempo:%11.9f\t / Tamaño Vectores:%u\n", tiempo_total, N);
for(i=0; i<N; i++)
    printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
        i, i, v1[i], v2[i], v3[i]);
}
#else
printf("Tiempo:%11.9f\t / Tamaño Vectores:%u\n", tiempo_total, N);
for(int i = 0; i<N; i++)
    printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
        i, i, v1[i], v2[i], v3[i]);
#endif

return 0;
}
```

The terminal window shows the execution of the program for N=8 and N=11. The output for N=8 is:

```
JavierRamirezPulido>vie mar 13 ./SumaVectores 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000000000 / 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+1.600000=0.000000) /
/ V1[5]+V2[5]=V3[5](1.300000+1.600000=0.000000) /
/ V1[6]+V2[6]=V3[6](1.400000+1.600000=0.000000) /
/ V1[7]+V2[7]=V3[7](1.500000+1.600000=0.000000) /
JavierRamirezPulido>vie mar 13 ./SumaVectores 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000000000 / Tamaño Vectores:11
/ V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) /
/ V1[1]+V2[1]=V3[1](1.200000+1.000000=2.200000) /
/ V1[2]+V2[2]=V3[2](1.300000+0.900000=2.200000) /
/ V1[3]+V2[3]=V3[3](1.400000+0.800000=2.200000) /
/ V1[4]+V2[4]=V3[4](1.500000+0.700000=2.200000) /
/ V1[5]+V2[5]=V3[5](1.600000+2.200000=0.000000) /
/ V1[6]+V2[6]=V3[6](1.700000+2.200000=0.000000) /
/ V1[7]+V2[7]=V3[7](1.800000+2.200000=0.000000) /
/ V1[8]+V2[8]=V3[8](1.900000+2.200000=0.000000) /
/ V1[9]+V2[9]=V3[9](2.000000+2.200000=0.000000) /
/ V1[10]+V2[10]=V3[10](2.100000+2.200000=0.000000) /
JavierRamirezPulido>vie mar 13
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA (compilación y ejecución para N=8 y 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 cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta.

RESPUESTA: `omp_set_num_threads(N°)` permite cambiar el número de hebras o cores que ejecutan el programa, pero de no usarlo, el número de hebras coincidirán con los de tu procesador. Esto ocurre tanto en el 7 como en el 8. Otras funciones de apoyo que podrían ayudarnos a saber esta información son `omp_get_max_threads()` y `omp_get_num_procs()`, que devuelven el mayor número de hebras disponibles en la region paralela y el número de procesadores disponibles para el programa respectivamente.

10. Rellenar una tabla como la Tabla 22 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:

```

Actividades Terminal dom 19:26
c3estudiante18@atcgrid:~/bp1/ejer10

xavivi2000@xavivi2000: ~
Tiempo:0.000199370 / Tamaño Vectores:16384 / V1[0]+V2[0]=V3[0](1638.400000+1638.400000=3276.800000) / / V1[16383]+V2[16383]=V3[16383]
383](3276.700000+0.100000=3276.800000) /
Tamaño Vectores:32768 (4 B)
Tiempo:0.000173214 / Tamaño Vectores:32768 / V1[0]+V2[0]=V3[0](3276.800000+3276.800000=6553.600000) / / V1[32767]+V2[32767]=V3[32767]
767](6553.500000+0.100000=6553.600000) /
Tamaño Vectores:65536 (4 B)
Tiempo:0.000475102 / Tamaño Vectores:65536 / V1[0]+V2[0]=V3[0](6553.600000+6553.600000=13107.200000) / / V1[65535]+V2[65535]=V3[65535]
5535](13107.100000+0.100000=13107.200000) /
Tamaño Vectores:131072 (4 B)
Tiempo:0.000348089 / Tamaño Vectores:131072 / V1[0]+V2[0]=V3[0](13107.200000+13107.200000=26214.400000) / / V1[131071]+V2[131071]=V3[131071]
26214](26214.300000+0.100000=26214.400000) /
Tamaño Vectores:262144 (4 B)
Tiempo:0.000798925 / Tamaño Vectores:262144 / V1[0]+V2[0]=V3[0](26214.400000+26214.400000=52428.800000) / / V1[262143]+V2[262143]=V3[262143]
52428](52428.700000+0.100000=52428.800000) /
Tamaño Vectores:524288 (4 B)
Tiempo:0.001319581 / Tamaño Vectores:524288 / V1[0]+V2[0]=V3[0](52428.800000+52428.800000=104857.600000) / / V1[524287]+V2[524287]=V3[524287]
104857](104857.500000+0.100000=104857.600000) /
Tamaño Vectores:1048576 (4 B)
Tiempo:0.001852908 / Tamaño Vectores:1048576 / V1[0]+V2[0]=V3[0](104857.600000+104857.600000=209715.200000) / / V1[1048575]+V2[1048575]=V3[1048575]
209715](209715.100000+0.100000=209715.200000) /
Tamaño Vectores:2097152 (4 B)
Tiempo:0.003562579 / Tamaño Vectores:2097152 / V1[0]+V2[0]=V3[0](209715.200000+209715.200000=419430.400000) / / V1[2097151]+V2[2097151]=V3[2097151]
419430](419430.300000+0.100000=419430.400000) /
Tamaño Vectores:4194304 (4 B)
Tiempo:0.006391067 / Tamaño Vectores:4194304 / V1[0]+V2[0]=V3[0](419430.400000+419430.400000=838860.800000) / / V1[4194303]+V2[4194303]=V3[4194303]
838860](838860.700000+0.100000=838860.800000) /
Tamaño Vectores:8388608 (4 B)
Tiempo:0.012828608 / Tamaño Vectores:8388608 / V1[0]+V2[0]=V3[0](838860.800000+838860.800000=1677721.600000) / / V1[8388607]+V2[8388607]=V3[8388607]
1677721](1677721.500000+0.100000=1677721.600000) /
Tamaño Vectores:16777216 (4 B)
Tiempo:0.025156677 / Tamaño Vectores:16777216 / V1[0]+V2[0]=V3[0](1677721.600000+1677721.600000=3355443.200000) / / V1[16777215]+V2[16777215]=V3[16777215]
3355443](3355443.100000+0.100000=3355443.200000) /
Tamaño Vectores:33554432 (4 B)
Tiempo:0.050300233 / Tamaño Vectores:33554432 / V1[0]+V2[0]=V3[0](3355443.200000+3355443.200000=6710886.400000) / / V1[33554431]+V2[33554431]=V3[33554431]
6710886](6710886.300000+0.100000=6710886.400000) /
Tamaño Vectores:67108864 (4 B)
Tiempo:0.050140767 / Tamaño Vectores:33554432 / V1[0]+V2[0]=V3[0](3355443.200000+3355443.200000=6710886.400000) / / V1[33554431]+V2[33554431]=V3[33554431]
6710886](6710886.300000+0.100000=6710886.400000) /
Tamaño Vectores:33554432 (4 B)

```

Ejecución primera columna atcgrid

```

Actividades Terminal dom 19:21
c3estudiante18@atcgrid:~/bp1/ejer10

xavivi2000@xavivi2000: ~
xavivi2000@xavivi2000: ~
c3estudiante18@atcgrid:~/bp1/ejer10
xavivi2000@xavivi2000: ~

JavierRamirezPulido>dom mar 15 ./script_ejer7_atcgrid.sh
Tamaño Vectores:16384 (4 B) / Tamaño Vectores:16384
Tiempo(seg.):0.000787761
Tamaño Vectores:32768 (4 B) / Tamaño Vectores:32768
Tiempo(seg.):0.000742381
Tamaño Vectores:65536 (4 B) / Tamaño Vectores:65536
Tiempo(seg.):0.004140100
Tamaño Vectores:131072 (4 B) / Tamaño Vectores:131072
Tiempo(seg.):0.000921486
Tamaño Vectores:262144 (4 B) / Tamaño Vectores:262144
Tiempo(seg.):0.001121100
Tamaño Vectores:524288 (4 B) / Tamaño Vectores:524288
Tiempo(seg.):0.001490142
Tamaño Vectores:1048576 (4 B) / Tamaño Vectores:1048576
Tiempo(seg.):0.002132216
Tamaño Vectores:2097152 (4 B) / Tamaño Vectores:2097152
Tiempo(seg.):0.003646074
Tamaño Vectores:4194304 (4 B) / Tamaño Vectores:4194304
Tiempo(seg.):0.007158415
Tamaño Vectores:8388608 (4 B) / Tamaño Vectores:8388608
Tiempo(seg.):0.014712920
Tamaño Vectores:16777216 (4 B) / Tamaño Vectores:16777216
Tiempo(seg.):0.027980246
Tamaño Vectores:33554432 (4 B) / Tamaño Vectores:33554432
Tiempo(seg.):0.054354956
Tamaño Vectores:67108864 (4 B) / Tamaño Vectores:67108864
Tiempo(seg.):0.054865796
JavierRamirezPulido>dom mar 15

```

Ejecución columna 2 atcgrid

```

Actividades Terminal dom 19:23
c3estudiante18@atcgrid:~/bp1/ejer10

xavivi2000@xavivi2000: ~
xavivi2000@xavivi2000: ~
c3estudiante18@atcgrid:~/bp1/ejer10
xavivi2000@xavivi2000: ~

JavierRamirezPulido>dom mar 15 ./script_ejer8_atcgrid.sh
Tamaño Vectores:16384 (4 B) / Tamaño Vectores:16384
Tiempo:0.000409780
Tamaño Vectores:32768 (4 B) / Tamaño Vectores:32768
Tiempo:0.000295592
Tamaño Vectores:65536 (4 B) / Tamaño Vectores:65536
Tiempo:0.000466483
Tamaño Vectores:131072 (4 B) / Tamaño Vectores:131072
Tiempo:0.000452418
Tamaño Vectores:262144 (4 B) / Tamaño Vectores:262144
Tiempo:0.000556547
Tamaño Vectores:524288 (4 B) / Tamaño Vectores:524288
Tiempo:0.000951324
Tamaño Vectores:1048576 (4 B) / Tamaño Vectores:1048576
Tiempo:0.001765635
Tamaño Vectores:2097152 (4 B) / Tamaño Vectores:2097152
Tiempo:0.003086539
Tamaño Vectores:4194304 (4 B) / Tamaño Vectores:4194304
Tiempo:0.005881060
Tamaño Vectores:8388608 (4 B) / Tamaño Vectores:8388608
Tiempo:0.011726249
Tamaño Vectores:16777216 (4 B) / Tamaño Vectores:16777216
Tiempo:0.023056237
Tamaño Vectores:33554432 (4 B) / Tamaño Vectores:33554432
Tiempo:0.044520874
Tamaño Vectores:67108864 (4 B) / Tamaño Vectores:67108864
Tiempo:0.044965448
JavierRamirezPulido>dom mar 15

```

Ejecución columna 3 atcgrid



Ejecución columna 1 en pc

The screenshot shows a Linux terminal window with the title 'Terminal'. The system clock in the top right corner indicates 11:05. The terminal content shows a user named 'JavierRanirezPulido' running a script named 'script.sh' in a directory named 'mar 13'. The script outputs a series of commands and their results, including file size and execution time for various vector operations.

```
JavierRanirezPulido>vte mar 13 ./script.sh
Tamaño Vectores:16384 (4 B)
Tiempo(seg.):0.000084929 / Tamaño Vectores:16384
Tamaño Vectores:32768 (4 B)
Tiempo(seg.):0.000175037 / Tamaño Vectores:32768
Tamaño Vectores:65536 (4 B)
Tiempo(seg.):0.000344118 / Tamaño Vectores:65536
Tamaño Vectores:131072 (4 B)
Tiempo(seg.):0.000684672 / Tamaño Vectores:131072
Tamaño Vectores:262144 (4 B)
Tiempo(seg.):0.001610324 / Tamaño Vectores:262144
Tamaño Vectores:524288 (4 B)
Tiempo(seg.):0.002736723 / Tamaño Vectores:524288
Tamaño Vectores:1048576 (4 B)
Tiempo(seg.):0.003929989 / Tamaño Vectores:1048576
Tamaño Vectores:2097152 (4 B)
Tiempo(seg.):0.008164034 / Tamaño Vectores:2097152
Tamaño Vectores:4194304 (4 B)
Tiempo(seg.):0.014698768 / Tamaño Vectores:4194304
Tamaño Vectores:8388608 (4 B)
Tiempo(seg.):0.027882922 / Tamaño Vectores:8388608
Tamaño Vectores:16777216 (4 B)
Tiempo(seg.):0.056287808 / Tamaño Vectores:16777216
Tamaño Vectores:33554432 (4 B)
Tiempo(seg.):0.111316221 / Tamaño Vectores:33554432
Tamaño Vectores:67108864 (4 B)
Tiempo(seg.):0.110611259 / Tamaño Vectores:67108864
JavierRanirezPulido>vte mar 13
```

Ejecución columna 2 en pc

```

Actividades Terminal vie 11:08
Terminal
Archivo Editar Ver Buscar Terminal Ayuda
JavierRamirezPulido>vie mar 13 ./script.sh
Tamaño Vectores:16384 (4 B)
Tiempo:0.000027300 / Tamaño Vectores:16384
Tamaño Vectores:32768 (4 B)
Tiempo:0.000051336 / Tamaño Vectores:32768
Tamaño Vectores:65536 (4 B)
Tiempo:0.000093162 / Tamaño Vectores:65536
Tamaño Vectores:131072 (4 B)
Tiempo:0.000187948 / Tamaño Vectores:131072
Tamaño Vectores:262144 (4 B)
Tiempo:0.000469772 / Tamaño Vectores:262144
Tamaño Vectores:524288 (4 B)
Tiempo:0.001605274 / Tamaño Vectores:524288
Tamaño Vectores:1048576 (4 B)
Tiempo:0.002911803 / Tamaño Vectores:1048576
Tamaño Vectores:2097152 (4 B)
Tiempo:0.005501186 / Tamaño Vectores:2097152
Tamaño Vectores:4194304 (4 B)
Tiempo:0.011573034 / Tamaño Vectores:4194304
Tamaño Vectores:8388608 (4 B)
Tiempo:0.021197616 / Tamaño Vectores:8388608
Tamaño Vectores:16777216 (4 B)
Tiempo:0.039127499 / Tamaño Vectores:16777216
Tamaño Vectores:33554432 (4 B)
Tiempo:0.077907602 / Tamaño Vectores:33554432
Tamaño Vectores:67108864 (4 B)
Tiempo:0.077763705 / Tamaño Vectores:67108864
JavierRamirezPulido>vie mar 13

```

Ejecución columna 3 en pc

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 2 threads/cores	T. paralelo (versión sections) 2 threads/cores
16384	0,00165682	0,000084329	0,0000273
32768	0,000339599	0,000175037	0,000051336
65536	0,000444747	0,000344118	0,000093162
131072	0,000733119	0,000684672	0,000187948
262144	0,001527918	0,001610324	0,000469772
524288	0,001982264	0,002736723	0,001605274
1048576	0,003756661	0,003929989	0,002911803
2097152	0,007546716	0,008164034	0,005501186
4194304	0,014551251	0,014698768	0,011573034
8388608	0,028316301	0,027882922	0,021197616
16777216	0,055931344	0,056228708	0,039127499
33554432	0,10910511	0,111316221	0,077907602
67108864	0,120236684	0,110611259	0,077763705

cpu

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 24 threads/cores	T. paralelo (versión sections) 24 threads/cores
16384	0,00019937	0,000787761	0,00040978
32768	0,000173214	0,000742381	0,000295592
65536	0,000475102	0,0041401	0,000466483
131072	0,000348089	0,00092486	0,000452418
262144	0,000798925	0,0011211	0,000556547
524288	0,001319581	0,001490142	0,000951324
1048576	0,003562579	0,002132216	0,001765635
2097152	0,006391067	0,003646074	0,003086539
4194304	0,012828608	0,007158415	0,00588106
8388608	0,025156677	0,01471292	0,011726249
16777216	0,050300233	0,027980246	0,023056237
33554432	0,050306562	0,054354956	0,044520874
67108864	0,050140767	0,054865796	0,044965448

atcgrid

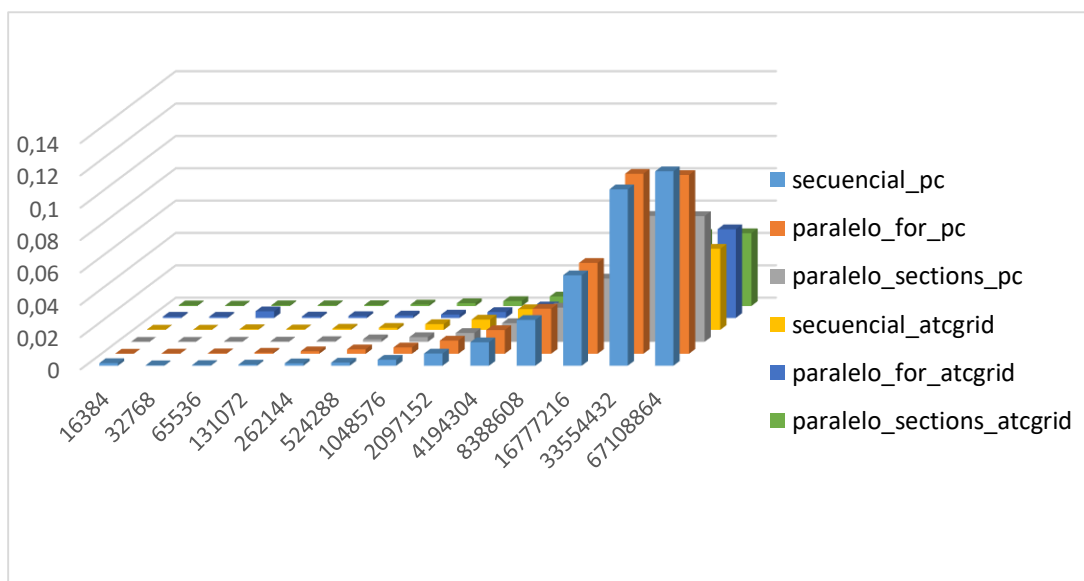


Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla “¿?” por el número de threads utilizados, que debe coincidir con el número de cores físicos del computador que como máximo puede aprovechar el código.

11. Rellenar una tabla como la **¡Error! Marcador no definido.** 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: Pues en la mayoría de casos es igual exceptuando algún otro en el que el tiempo de CPU es

mayor que el elapsed.

Nº de Componentes	Tiempo secuencial vect. Globales 1 thread/core			Tiempo paralelo/versión for 24 Threads/cores		
	Elapsed	CPU-user	CPU- sys	sed	CPU-user	Elap- CPU- sys
65536	0m0.002s	0m0.000s	0m0.002s	0m0.008s	0m0.029s	0m0.020s
131072	0m0.004s	0m0.001s	0m0.003s	0m0.013s	0m0.070s	0m0.016s
262144	0m0.004s	0m0.001s	0m0.002s	0m0.016s	0m0.095s	0m0.013s
524288	0m0.006s	0m0.003s	0m0.003s	0m0.017s	0m0.096s	0m0.018s
1048576	0m0.006s	0m0.004s	0m0.002s	0m0.006s	0m0.026s	0m0.017s
2097152	0m0.009s	0m0.005s	0m0.004s	0m0.018s	0m0.119s	0m0.010s
4194304	0m0.017s	0m0.017s	0m0.004s	0m0.019s	0m0.103s	0m0.034s
8388608	0m0.033s	0m0.024s	0m0.009s	0m0.032s	0m0.169s	0m0.078s
16777216	0m0.064s	0m0.046s	0m0.018s	0m0.064s	0m0.376s	0m0.123s
33554432	0m0.125s	0m0.091s	0m0.034s	0m0.124s	0m0.667s	0m0.309s
67108864	0m0.242s	0m0.170s	0m0.072s	0m0.253s	0m1.359s	0m0.628s

Nº de Componentes	Tiempo secuencial vect. Globales 1 thread/core			Tiempo paralelo/versión for ¿? Threads/cores		
	Elapsed	CPU-user	CPU- sys	Elapsed	CPU-user	CPU- sys
65536						
131072						
262144						
524288						
1048576						
2097152						
4194304						
8388608						
16777216						
33554432						
67108864						