

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 Gómez Luzón

Grupo de prácticas: C1

Fecha de entrega:

Fecha evaluación en clase:

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`.

Imagen 1: Muestra del código `for 2.c` (`bucle-forModificado.c`).



```
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2/pruebas]2018-03-18domingo
$cat for_2.c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

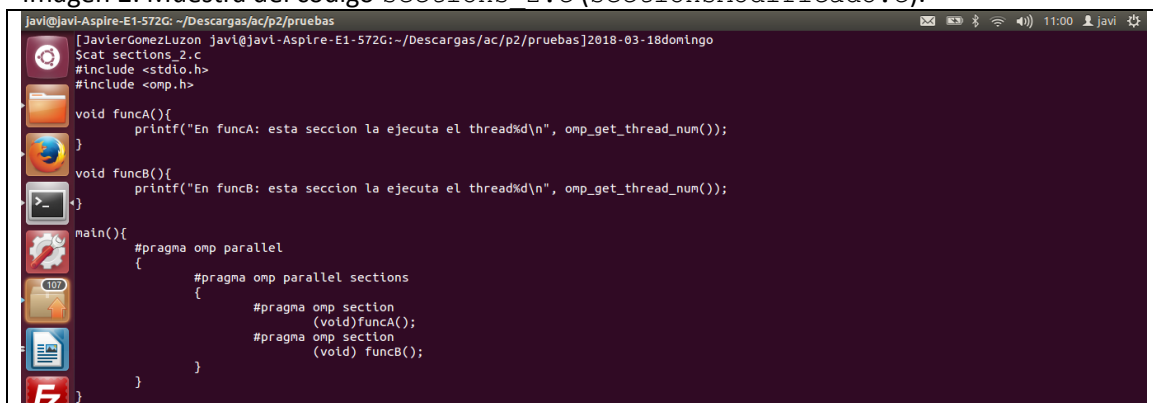
int main(int argc, char ** argv){
    int i, n=9;

    if(argc<2){
        fprintf(stderr, "\n[ERROR] -Falta nº iteraciones\n");
        exit(-1);
    }
    n=atoi(argv[1]);

    #pragma omp parallel
    {
        #pragma omp parallel for
        for(i=0;i<n;i++){
            printf("Thread %d ejecuta la iteracion %d del bucle\n", omp_get_thread_num(),i);
        }
    }
    return(0);
}
```

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

Imagen 2: Muestra del código `sections 2.c` (`sectionsModificado.c`).



```
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2/pruebas]2018-03-18domingo
$cat sections_2.c
#include <omp.h>

void funcA(){
    printf("En funcA: esta seccion la ejecuta el thread%d\n", omp_get_thread_num());
}

void funcB(){
    printf("En funcB: esta seccion la ejecuta el thread%d\n", omp_get_thread_num());
}

main(){
    #pragma omp parallel
    {
        #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`

Imagen 3: Muestra del código `single_2.c` (`singleModificado.c`).



```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2/pruebas
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-18domingo
Scat single_2.c
#include <stdio.h>
#include <omp.h>

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 inicializacion a: ");
            scanf("%d", &a);
            printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
        }

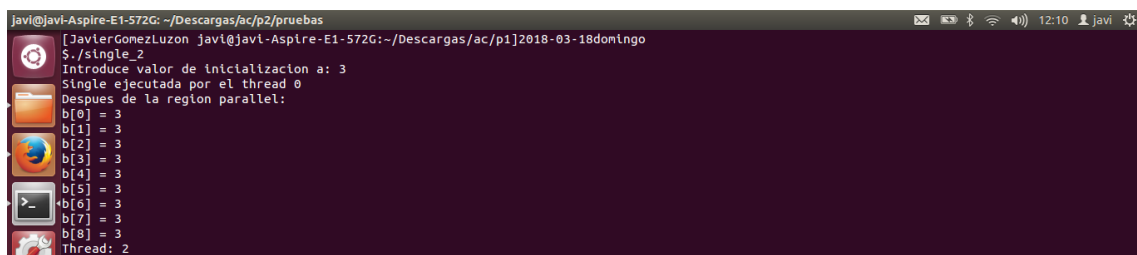
        #pragma omp for
        for(i=0;i<n;i++){
            b[i]=a;
        }

        #pragma omp single
        {
            printf("Despues de la region parallel:\n");
            for(i=0;i<n;i++) printf("b[%d] = %d\n", i, b[i]);
            printf("Thread: %d\n", omp_get_thread_num());
        }
    }
}

```

CAPTURAS DE PANTALLA:

Imagen 4: Muestra de la ejecución de `single_2.c` en local.



```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2/pruebas
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-18domingo
$ ./single_2
Introduce valor de inicializacion a: 3
Single ejecutada por el thread 0
Despues de la region 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
Thread: 2

```

- 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`

Imagen 5: Muestra del código `single_3.c` (`singleModificado2.c`).

```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2/pruebas
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-18domingo
$ cat single_3.c
#include <stdio.h>
#include <omp.h>

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 inicializacion 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("Despues de la region parallel:\n");
            for(i=0;i<n;i++) printf("b[%d] = %d\n", i, b[i]);
            printf("Thread: %d\n", omp_get_thread_num());
        }
    }
}

```

CAPTURAS DE PANTALLA:Imagen 6: Muestra de la ejecución de `single_3.c` en local.

```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2/pruebas
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-18domingo
$ ./single_3
Introduce valor de inicializacion a: 3
Single ejecutada por el thread 0
Despues de la region 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
Thread: 0

```

RESPUESTA A LA PREGUNTA:

Al usar la directiva master la hebra 0 (la hebra master) ejecutará siempre la parte del código donde se muestra el vector. Por lo tanto será siempre la hebra master quien muestre el vector.

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 que si por casualidad la hebra master se ejecutara más rápido el vector no habría terminado de inicializarse correctamente y la hebra master mostrará valores incorrectos o basura cuando muestre el vector.

1.1.1**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:

Imagen 7: Muestra de la ejecución de `SumaVectoresCG.c` en atcgrid con time para 10M de elementos.

```
Ciestudiante9@atcgrid:~
[JavierGomezLuzon Ciestudiante9@atcgrid:~]2018-03-24sábado
Secho 'time ./SumaVectoresCG 10000000' | qsub -q ac
68879.atcgrid
[JavierGomezLuzon Ciestudiante9@atcgrid:~]2018-03-24sábado
$ls
STDIN.e68879  STDIN.o68879  SumaVectoresCG
```

Imagen 8: Muestra de los resultados de la ejecución anterior.

```
javi@javi-Aspire-E1-572G: ~
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$cat STDIN.o*
Tiempo(seg.):0.057240731      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)
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$cat STDIN.e*
real    0m0.194s
user    0m0.089s
sys     0m0.110s
```

La suma de los tiempos `user` y `sys` es 0.190s y el tiempo real es 0.194s. El tiempo es practicamente el mismo ya que usamos una sola hebra del procesador.

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

CAPTURAS DE PANTALLA:

Imagen 9: Muestra de la ejecución de `SumaVectoresCG.c` en atcgrid con time para 10 elementos.

```
Ciestudiante9@atcgrid:~
[JavierGomezLuzon Ciestudiante9@atcgrid:~]2018-03-24sábado
Secho 'time ./SumaVectoresCG 10' | qsub -q ac
68878.atcgrid
[JavierGomezLuzon Ciestudiante9@atcgrid:~]2018-03-24sábado
$ls
STDIN.e68878  STDIN.o68878  SumaVectoresCG
```

Imagen 10: Muestra de los resultados de la ejecución anterior.

```
javi@javi-Aspire-E1-572G: ~
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$cat STDIN.o*
Tiempo(seg.):0.000002745      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)
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$cat STDIN.e*
real    0m0.003s
user    0m0.001s
sys     0m0.001s
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

NI(10M elementos)= $3+10000000*6=60000003$ instrucciones.

NI(10 elementos)= $3+10*6=63$ instrucciones.

NI_F(10M elementos)= $3*10000000=30000000$ instrucciones de coma flotante.

NI_F(10 elementos)= $3+10=30$ instrucciones de coma flotante.

T(10M elementos)=0.0572s.

$T(10 \text{ elementos}) = 0.000002745 \text{ s.}$

$\text{MIPS}(10 \text{ M elementos}) = \text{NI}(10 \text{ M elementos}) / T(10 \text{ M elementos}) * 10^6 = 1048.95 = 1049 \text{ MIPS}$

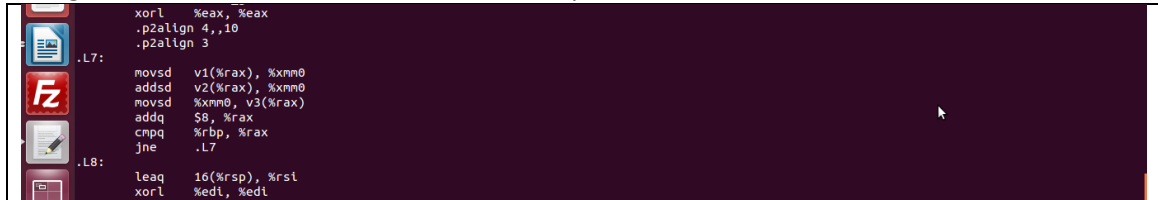
$\text{MIPS}(10 \text{ elementos}) = \text{NI}(10 \text{ elementos}) / T(10 \text{ elementos}) * 10^6 = 22.95 = 23 \text{ MIPS}$

$\text{MFLOPS}(10 \text{ M elementos}) = \text{NI_F}(10 \text{ M elementos}) / T(10 \text{ M elementos}) * 10^6 = 524.47 = 525 \text{ MFLOPS}$

$\text{MFLOPS}(10 \text{ elementos}) = \text{NI_F}(10 \text{ elementos}) / T(10 \text{ elementos}) * 10^6 = 10.92 = 11 \text{ MFLOPS}$

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

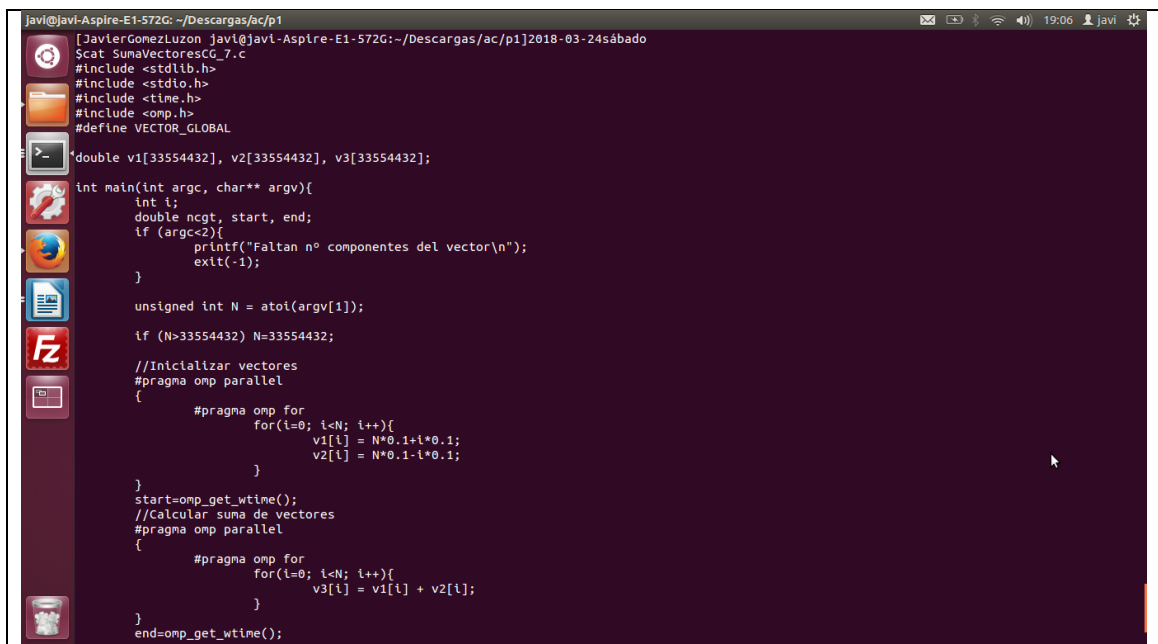
Imagen 11: Cat de SumaVectoresCG.s en la parte de la suma de vectores.



- 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

Imágenes 12 y 13: Cat de SumaVectoresCG_7.c



```

for(i=0;i<N;i++){
    printf("V3[%i]=%f\n", i, v3[i]);
}
printf("Tiempo que se ha tardado en hacer la suma-->%11.9f\n", end-start);
printf("v1[0]=%f\nv1[N-1]=%f\nv2[0]=%f\nv2[N-1]=%f\nv3[0]=%f\nv3[N-1]=%f\n", v1[0], v1[N-1], v2[0], v2[N-1], v3[0], v3[N-1]);
return 0;
}
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$

```

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

Imagen 14: Ejecución en atcgrid de SumaVectoresCG_7.c con 8 elementos.

```

Ciestudiante9@atcgrid:~
[JavierGomezLuzon Ciestudiante9@atcgrid:~]2018-03-24sábado
$ echo './SumaVectoresCG_7 8' | qsub -q ac
68846.atcgrid
[JavierGomezLuzon Ciestudiante9@atcgrid:~]2018-03-24sábado
$ ls
STDIN.e68846  STDIN.o68846  SumaVectoresCG_7

```

Imagen 15: Muestra de los resultados de la ejecución anterior.

```

javi@javi-Aspire-E1-572G:~/Descargas/ac/p1
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$ cat STDIN.o*
V3[0]=1.600000
V3[1]=1.600000
V3[2]=1.600000
V3[3]=1.600000
V3[4]=1.600000
V3[5]=1.600000
V3[6]=1.600000
V3[7]=1.600000
Tiempo que se ha tardado en hacer la suma-->0.003177041
V1[0]=0.800000
V1[N-1]=1.500000
V2[0]=0.800000
V2[N-1]=0.100000
V3[0]=1.600000
V3[N-1]=1.600000
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$ cat STDIN.e*
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$

```

Imagen 16: Ejecución en atcgrid de SumaVectoresCG_7.c para 11 elementos.

```

Ciestudiante9@atcgrid:~
[JavierGomezLuzon Ciestudiante9@atcgrid:~]2018-03-24sábado
$ echo './SumaVectoresCG_7 11' | qsub -q ac
68847.atcgrid
[JavierGomezLuzon Ciestudiante9@atcgrid:~]2018-03-24sábado
$ ls
STDIN.e68847  STDIN.o68847  SumaVectoresCG_7

```

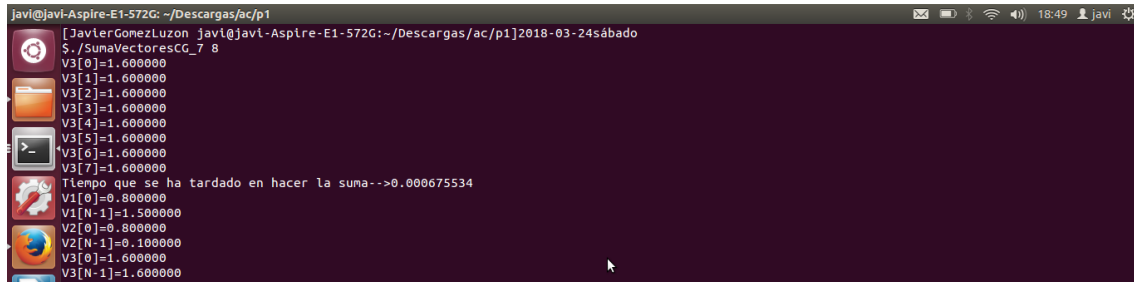
Imagen 17: Muestra de los resultados de la ejecución anterior.

```

javi@javi-Aspire-E1-572G:~/Descargas/ac/p1
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$ cat STDIN.o*
V3[0]=2.200000
V3[1]=2.200000
V3[2]=2.200000
V3[3]=2.200000
V3[4]=2.200000
V3[5]=2.200000
V3[6]=2.200000
V3[7]=2.200000
V3[8]=2.200000
V3[9]=2.200000
V3[10]=2.200000
Tiempo que se ha tardado en hacer la suma-->0.004140035
V1[0]=1.100000
V1[N-1]=2.100000
V2[0]=1.100000
V2[N-1]=0.100000
V3[0]=2.200000
V3[N-1]=2.200000
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$ cat STDIN.e*
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$

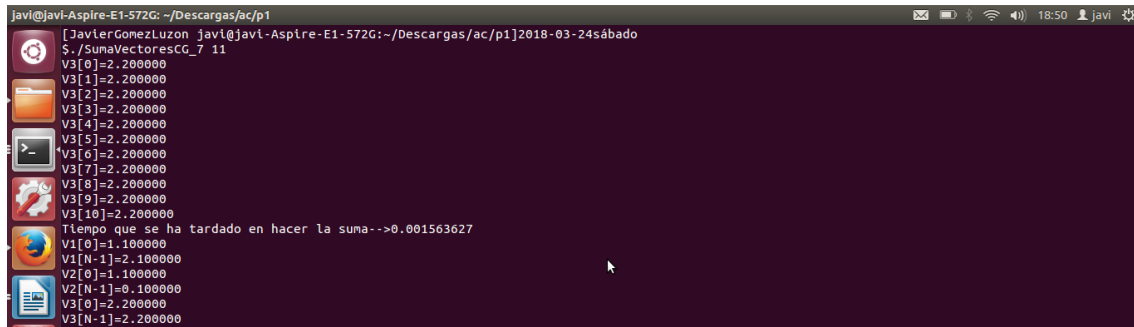
```

Imagen 18: Ejecución en local de SumaVectoresCG_7.c para 8 elementos.



```
javi@javi-Aspire-E1-572G: ~/Descargas/ac/p1
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$ ./SumaVectoresCG_7 8
V3[0]=1.600000
V3[1]=1.600000
V3[2]=1.600000
V3[3]=1.600000
V3[4]=1.600000
V3[5]=1.600000
V3[6]=1.600000
V3[7]=1.600000
Tiempo que se ha tardado en hacer la suma-->0.000675534
V1[0]=0.800000
V1[N-1]=1.500000
V2[0]=0.800000
V2[N-1]=0.100000
V3[0]=1.600000
V3[N-1]=1.600000
```

Imagen 19: Ejecución en local de SumaVectoresCG_7.c para 11 elementos.



```
javi@javi-Aspire-E1-572G: ~/Descargas/ac/p1
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$ ./SumaVectoresCG_7 11
V3[0]=2.200000
V3[1]=2.200000
V3[2]=2.200000
V3[3]=2.200000
V3[4]=2.200000
V3[5]=2.200000
V3[6]=2.200000
V3[7]=2.200000
V3[8]=2.200000
V3[9]=2.200000
V3[10]=2.200000
Tiempo que se ha tardado en hacer la suma-->0.001563627
V1[0]=1.100000
V1[N-1]=2.100000
V2[0]=1.100000
V2[N-1]=0.100000
V3[0]=2.200000
V3[N-1]=2.200000
```

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` 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

Imágenes 20, 21 y 22: Cat de SumaVectoresCG_8.c

```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p1
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$cat SumaVectoresCG_8.c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <omp.h>
#define VECTOR_GLOBAL

double v1[33554432], v2[33554432], v3[33554432];

int main(int argc, char** argv){
    int i;
    double ncgt, start, end;
    if (argc<2){
        printf("Faltan n° componentes del vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);

    if (N>33554432) N=33554432;

    //Inicializar vectores
    #pragma omp parallel private(i)
    {
        #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(i=(N/4);i<2*(N/4);i++){
                v1[i] = N*0.1+i*0.1;
                v2[i] = N*0.1-i*0.1;
            }

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

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

        start=omp_get_wtime();
        //Calcular suma de vectores
        #pragma omp parallel private(i)
        {
            #pragma omp sections
            {
                #pragma omp section
                for(i=0;i<(N/4);i++){
                    v3[i] = v1[i] + v2[i];
                }

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

                #pragma omp section
                for(i=2*(N/4);i<3*(N/4);i++){
                    v3[i] = v1[i] + v2[i];
                }

                #pragma omp section
                for(i=3*(N/4);i<4*(N/4);i++){
                    v3[i] = v1[i] + v2[i];
                }
            }
        }

        end=omp_get_wtime();
        for(i=0;i<N;i++){
            printf("V3[%i]=%f\n", i, v3[i]);
        }

        printf("Tiempo que se ha tardado en hacer la suma-->%11.9f\n", end-start);
        printf("v1[0]=%f\nv1[N-1]=%f\nv2[0]=%f\nv2[N-1]=%f\nv3[0]=%f\nv3[N-1]=%f\n", v1[0], v1[N-1], v2[0], v2[N-1], v3[0], v3[N-1]);
        return 0;
    }
}
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

Imagen 23: Ejecución en atcgrid de SumaVectoresCG_8.c para 8 elementos.

```

C1estudiante9@atcgrid:~
[javierGomezLuzon C1estudiante9@atcgrid:~]2018-03-24sábado
$echo './SumaVectoresCG_8 8' | qsub -q ac
68849.atcgrid
[javierGomezLuzon C1estudiante9@atcgrid:~]2018-03-24sábado
$ls
STDIN.e68849  STDIN.o68849  SumaVectoresCG_8
[javierGomezLuzon C1estudiante9@atcgrid:~]2018-03-24sábado

```


Imagen 24: Mostrar los resultados de la ejecución anterior.

```
javi@javi-Aspire-E1-572G: ~/Descargas/ac/p1
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
Scat STDIN.o*
V3[0]=1.600000
V3[1]=1.600000
V3[2]=1.600000
V3[3]=1.600000
V3[4]=1.600000
V3[5]=1.600000
V3[6]=1.600000
V3[7]=1.600000
Tiempo que se ha tardado en hacer la suma-->0.003874971
V1[0]=0.800000
V1[N-1]=1.500000
V2[0]=0.800000
V2[N-1]=0.100000
V3[0]=1.600000
V3[N-1]=1.600000
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
Scat STDIN.e*
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$
```

Imagen 25: Ejecución en atcgrid de SumaVectoresCG_8.c para 11 elementos.

```
Ciestudiante9@atcgrid:~
[JavierGomezLuzon Ciestudiante9@atcgrid:~]2018-03-24sábado
Secho './SumaVectoresCG_8 11' | qsub -q ac
68850.atcgrid
[JavierGomezLuzon Ciestudiante9@atcgrid:~]2018-03-24sábado
$ls
STDIN.e68850 STDIN.o68850 SumaVectoresCG_8
[JavierGomezLuzon Ciestudiante9@atcgrid:~]2018-03-24sábado
$
```

Imagen 26: Mostrar los resultados de la ejecución anterior.

```
javi@javi-Aspire-E1-572G: ~/Descargas/ac/p1
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
Scat STDIN.o*
V3[0]=2.200000
V3[1]=2.200000
V3[2]=2.200000
V3[3]=2.200000
V3[4]=2.200000
V3[5]=2.200000
V3[6]=2.200000
V3[7]=2.200000
V3[8]=0.800000
V3[9]=0.800000
V3[10]=0.800000
Tiempo que se ha tardado en hacer la suma-->0.003743057
V1[0]=1.100000
V1[N-1]=0.800000
V2[0]=1.100000
V2[N-1]=0.800000
V3[0]=2.200000
V3[N-1]=0.800000
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
Scat STDIN.e*
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$
```

Imagen 27: Ejecución en local de SumaVectoresCG_8.c para 8 elementos.

```
javi@javi-Aspire-E1-572G: ~/Descargas/ac/p1
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$. ./SumaVectoresCG_8 8
V3[0]=1.600000
V3[1]=1.600000
V3[2]=1.600000
V3[3]=1.600000
V3[4]=1.600000
V3[5]=1.600000
V3[6]=1.600000
V3[7]=1.600000
Tiempo que se ha tardado en hacer la suma-->0.001270108
V1[0]=0.800000
V1[N-1]=1.500000
V2[0]=0.800000
V2[N-1]=0.100000
V3[0]=1.600000
V3[N-1]=1.600000
$
```

Imagen 28: Ejecución en local de SumaVectoresCG_8.c para 11 elementos.

```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p1
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p1]2018-03-24sábado
$ ./SumaVectoresCG_8 11
V3[0]=2.200000
V3[1]=2.200000
V3[2]=2.200000
V3[3]=2.200000
V3[4]=2.200000
V3[5]=2.200000
V3[6]=2.200000
V3[7]=2.200000
V3[8]=0.000000
V3[9]=0.000000
V3[10]=0.000000
Tiempo que se ha tardado en hacer la suma-->0.001103188
V1[0]=1.100000
V1[N-1]=0.000000
V2[0]=1.100000
V2[N-1]=0.000000
V3[0]=2.200000
V3[N-1]=0.000000

```

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:

Para la implementación del ejercicio 7 se podrán usar tantos threads y cores como disponga el ordenador, aunque si se usan más que iteraciones tengan los bucles algunos threads y cores quedaran inutilizados.

Para la implementación del ejercicio 8 se podran usar hasta 4 threads.

10. Rellenar una tabla como la Tabla 2 para atcgrid y otra para su PC con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución del trozo de código que realiza la suma en paralelo (este es el tiempo que deben imprimir los programas). Ponga en la tabla el número de threads/cores que usan los códigos. Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute código que imprima todos los componentes del resultado cuando este número sea elevado.

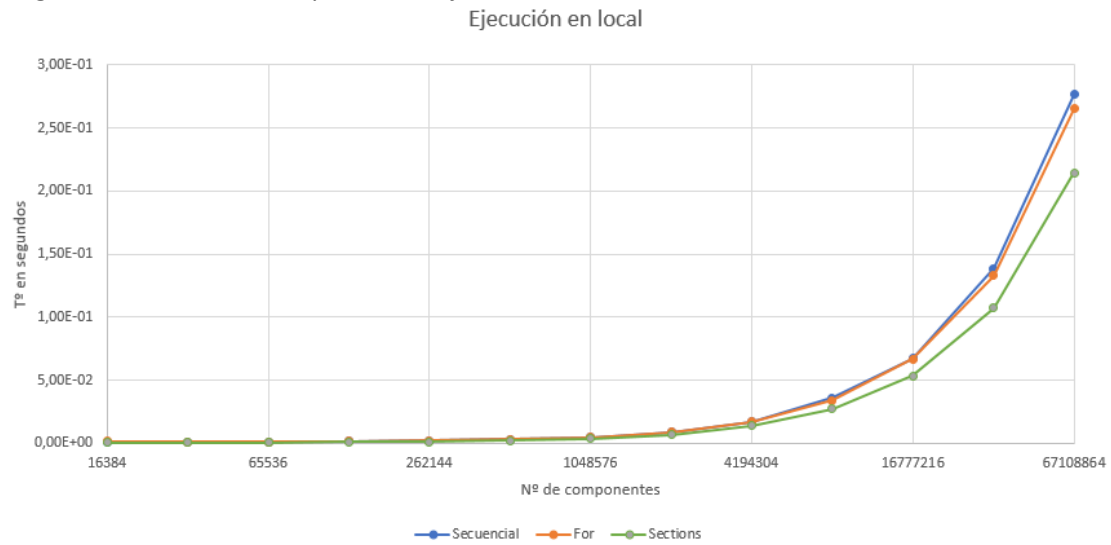
RESPUESTA:

Local

Nº de Componentes	T. secuencial vect. dinamicos 1 thread/core	T. paralelo (versión for) 2 threads/ 2 cores	T. paralelo (versión sections) 2 threads/ 2 cores
16384	0,000906961	0,001342636	0,000373139
32768	0,001145848	0,001170823	0,000407621
65536	0,000990901	0,001154826	0,000572156
131072	0,001530295	0,001503259	0,000838976
262144	0,002250432	0,002352562	0,001051175
524288	0,002876027	0,002929313	0,001983320
1048576	0,004884081	0,004461728	0,003787044
2097152	0,008836981	0,008937639	0,006868197
4194304	0,016978000	0,017083201	0,013646227
8388608	0,035727707	0,033806418	0,027404866
16777216	0,067110095	0,066768425	0,053588323

33554432	0,138400036	0,132727585	0,107275865
67108864	0,276800093	0,26545698	0,214551350

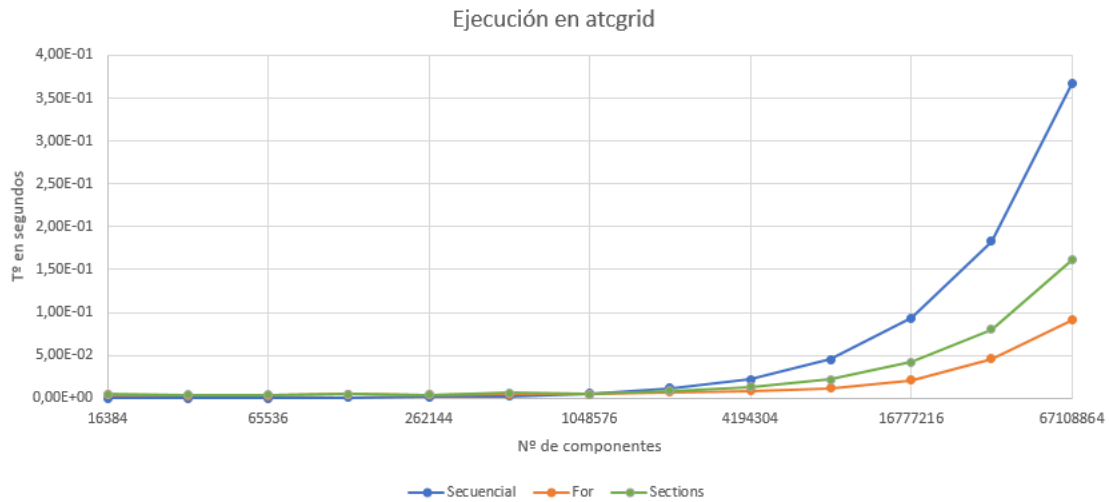
Imagen 29: Gráfico de tiempos de las ejecuciones en local.



Atcgrid

Nº de Componentes	T. secuencial vect. dinamicos 1 thread/core	T. paralelo (versión for) 24 threads/ 12 cores	T. paralelo (versión sections) 24 threads/ 12 cores
16384	0,000107215	0,003886839	0,004561996
32768	0,000211873	0,002957159	0,004351837
65536	0,000386961	0,002827300	0,003883575
131072	0,000793499	0,004427627	0,004363129
262144	0,001294151	0,004209854	0,003850685
524288	0,002699953	0,004756911	0,005940518
1048576	0,005663317	0,005101529	0,004463917
2097152	0,011404337	0,007429161	0,008250421
4194304	0,022260493	0,008585249	0,013629277
8388608	0,045424122	0,011677544	0,021665397
16777216	0,093289052	0,021448501	0,042001222
33554432	0,183319164	0,045732503	0,080446724
67108864	0,366638342	0,091465096	0,160893453

Imagen 30: Gráfico de tiempos de las ejecuciones en atcgrid.



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

RESPUESTA:

El tiempo Elapsed es igual al tiempo de CPU ya que para la versión secuencial el programa utiliza una única hebra. Para la versión en paralelo con la directiva for el tiempo Elapsed es mucho menor al tiempo de CPU, ya que el tiempo de CPU es la suma de los tiempos que ha tardado cada hebra en ejecutarse mientras que el tiempo Elapsed es lo que ha tardado la hebra principal en ejecutarse, por eso es mucho menor.

Nº de Componentes	Tiempo secuencial vect. Globales 1 thread/core			Tiempo paralelo/versión for 12 Threads/cores		
	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>
65536	0,003	0,003	0,000	0,010	0,168	0,030
131072	0,004	0,003	0,001	0,012	0,207	0,016
262144	0,005	0,002	0,002	0,013	0,216	0,018
524288	0,012	0,007	0,004	0,014	0,222	0,015
1048576	0,024	0,015	0,008	0,016	0,236	0,031
2097152	0,040	0,018	0,022	0,023	0,273	0,064
4194304	0,082	0,037	0,044	0,036	0,293	0,112
8388608	0,160	0,065	0,093	0,045	0,342	0,264
16777216	0,307	0,126	0,178	0,081	0,546	0,515
33554432	0,602	0,259	0,338	0,146	0,893	1,031
67108864	1,233	0,564	0,662	0,294	1,785	2,030