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): David Martinez Diaz
Grupo de prácticas y profesor de prácticas: 2 – Juan José Escobar Perez
Fecha de entrega: 05/04/2021
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`

bucle-forModificado.c

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

int main(int argc, char **argv) {

    int i, n = 9;
    if(argc < 2) {

        fprintf(stderr,"\n[ERROR] - Falta no iteraciones \n");
        exit(-1);
    }

    n = atoi(argv[1]);
    #pragma omp parallel for
        for (i=0; i<n; i++)
            printf("thread %d ejecuta la iteración %d del bucle\n", omp_get_thread_num(),i);

    return(0);
}
```

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

sectionsModificado.c

```c
#include <stdio.h>
#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());
}


int main(){

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

    return 0;
}
```

2. Imprimir los resultados del programa `single.c` usando una directiva `single` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `single` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `single`. Incorpore en su cuaderno de trabajo el código fuente y volcados de pantalla con los resultados de ejecución obtenidos.

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

single.c | singleModificado.c

D: > Usuario > Desktop > Inf + Ade > 2 curso > 2 cuatrimestre > AC > AC_Practicas > bp1_MartinezDiazDavid > ejer2 > C singleModificado.c > main()

```c
#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 inicializacion a: ");
            scanf("%d", &a);
            printf("Single(introducir valor) 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("\n Impresion ejecutada por el thread %d\n", omp_get_thread_num());
            for(i = 0; i < n; i++) printf("b[%d] = %d [hebra %d]\n", i, b[i], omp_get_thread_num());
        }

    }

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
 DavidMartinezDiaz - 21-04-01 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp1_MartinezDiazDavid     cd ejer2
 DavidMartinezDiaz - 21-04-01 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer2     ls     ✓ | 22:15:16
single.c  singleModificado  singleModificado.c
 DavidMartinezDiaz - 21-04-01 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer2     ./singleModificado
Introduce valor de inicializacion a: 7
Single(introducir valor) ejecutada por el thread 0

 Impresion ejecutada por el thread 8
b[0] = 7 [hebra 8]
b[1] = 7 [hebra 8]
b[2] = 7 [hebra 8]
b[3] = 7 [hebra 8]
b[4] = 7 [hebra 8]
b[5] = 7 [hebra 8]
b[6] = 7 [hebra 8]
b[7] = 7 [hebra 8]
b[8] = 7 [hebra 8]
 DavidMartinezDiaz - 21-04-01 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer2     |     ✓ | 22:15:32
```

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`

singleModificado2.c

```c
#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 inicializacion a: ");
            scanf("%d", &a);
            printf("Single(introducir valor) 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("\n Impresion ejecutada por el thread %d\n", omp_get_thread_num());
            for(i = 0; i < n; i++) printf("b[%d] = %d [hebra %d]\n", i, b[i], omp_get_thread_num());
        }

    }

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
 DavidMartinezDiaz - 21-04-01 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer3     ls      | 22:16:05
single.c  singleModificado2  singleModificado2.c
 DavidMartinezDiaz - 21-04-01 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer3     ./singleModificado2
Introduce valor de inicializacion a: 4
Single(introducir valor) ejecutada por el thread 0

 Impresion ejecutada por el thread 0
b[0] = 4 [hebra 0]
b[1] = 4 [hebra 0]
b[2] = 4 [hebra 0]
b[3] = 4 [hebra 0]
b[4] = 4 [hebra 0]
b[5] = 4 [hebra 0]
b[6] = 4 [hebra 0]
b[7] = 4 [hebra 0]
b[8] = 4 [hebra 0]
 DavidMartinezDiaz - 21-04-01 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer3     |       | 22:16:12
```

RESPUESTA A LA PREGUNTA:

La principal diferencia es que en la directiva single se ejecuta cualquier hebra de las disponibles y en la directiva master siempre se ejecuta la hebra principal, que en este caso es la hebra 0.

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

RESPUESTA:

Esto se debe a que la directiva master no tiene unas barreras implicitas, si quitamos dicha barrera, la que se pone antes de que se ejecute la directiva master, puede que a la hora de calcular la suma se realice antes de que terminen las demas hebras, dando lugar a resultados erroneos. Es decir lo que hace la directiva barrier es esperar a que terminen todas las hebras, y luego continua.

1.1.1

Resto de ejercicios (usar en atcgrid la cola ac a no ser que se tenga que usar atcgrid4)

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,…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:

```
[e2estudiante14@atcgrid ejer5]$ ls
SumaVectoresC.c
[e2estudiante14@atcgrid ejer5]$ gcc -O2 -fopenmp SumaVectoresC.c -o SumaVect
oresC
[e2estudiante14@atcgrid ejer5]$ ls
SumaVectoresC  SumaVectoresC.c
[e2estudiante14@atcgrid ejer5]$ sbatch -p ac --wrap "time ./SumaVectoresC 10
000000"
Submitted batch job 78201
[e2estudiante14@atcgrid ejer5]$ ls
SumaVectoresC  SumaVectoresC.c  slurm-78201.out
[e2estudiante14@atcgrid ejer5]$ cat slurm-78201.out
Tamanio Vectores:10000000 (4 B)
Tiempo:0.043746392       / Tamanio 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    0m0.150s
user    0m0.064s
sys     0m0.050s
[e2estudiante14@atcgrid ejer5]$
```

RESPUESTA:

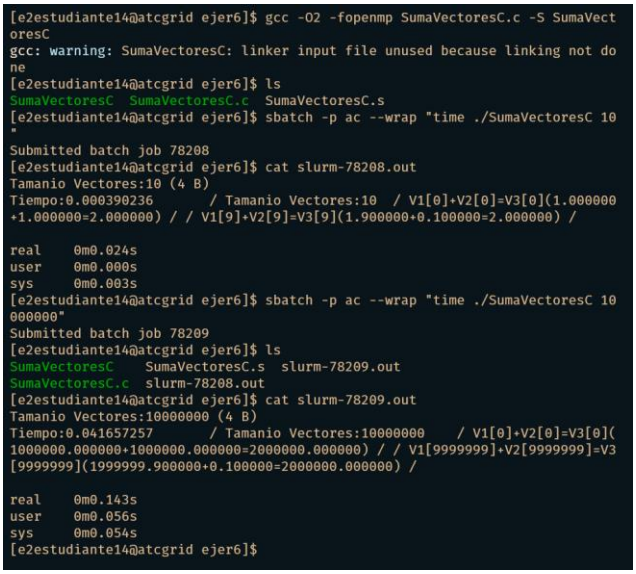
Esto se debe a que el tiempo real aunque sea mayor, no significa que tarde eso el cpu del usuario y la cpu del sistema, ya que el programa puede estar esperando sin estar ejecutando nada y aun asi el tiempo real estaría contado ese tiempo de espera.

Además existe el caso en el que el tiempo sea menor, ya que se podrían dividir el trabajo los hijos o threads que al final se sumarian al usuario y al sistema.

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. Incorporar **el código ensamblador de la parte de la suma de vectores** (no de todo el programa) 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

Caso: N 10;

Nº de Instrucciones: 6;

Tiempo: 0,000390236 s

MIPS = 60/0,000390236 * 10^6 ➔0.1537;

MFLOPS = 10/0,000390236*10^6 ➔2,56 * 10^(-3);

Caso: N 10000000;

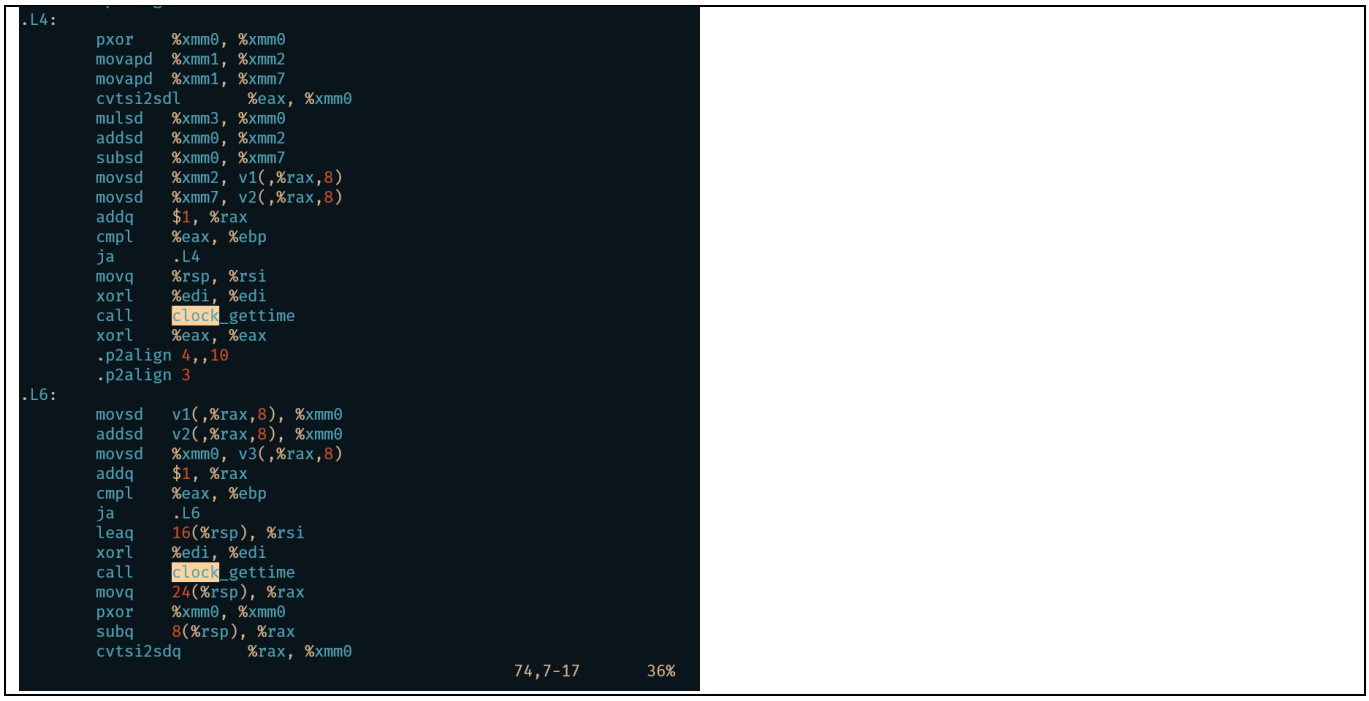
Nº de Instrucciones: 6;

Tiempo: 0,041657257 s

MIPS = 60000000/0, 041657257 * 10^6 ➔144,032;

MFLOPS = 10000000/0,041657257*10^6 ➔240,0542119;

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



```
.L4:
        pxor    %xmm0, %xmm0
        movapd  %xmm1, %xmm2
        movapd  %xmm1, %xmm7
        cvtsi2sdl       %eax, %xmm0
        mulsd   %xmm3, %xmm0
        addsd   %xmm0, %xmm2
        subsd   %xmm0, %xmm7
        movsd   %xmm2, v1(,%rax,8)
        movsd   %xmm7, v2(,%rax,8)
        addq    $1, %rax
        cmpl    %eax, %ebp
        ja      .L4
        movq    %rsp, %rsi
        xorl    %edi, %edi
        call    clock_gettime
        xorl    %eax, %eax
        .p2align 4,,10
        .p2align 3
.L6:
        movsd   v1(,%rax,8), %xmm0
        addsd   v2(,%rax,8), %xmm0
        movsd   %xmm0, v3(,%rax,8)
        addq    $1, %rax
        cmpl    %eax, %ebp
        ja      .L6
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime
        movq    24(%rsp), %rax
        pxor    %xmm0, %xmm0
        subq    8(%rsp), %rax
        cvtsi2sdq       %rax, %xmm0
                                              74,7-17       36%
```

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,…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 `sp-OpenMP-for.c`

```c
#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;
}

// Creamos variable para el omp_time
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];

// Recogemos el tiempo que tarda en calcular la suma de vectores
double tiempo2 = omp_get_wtime();
double tiempo_total = tiempo2 - tiempo1;

//Imprimir resultado de la suma y el tiempo de ejecucion
if (N<10) {
  printf("Tiempo:%11.9f\t / Tamanio Vectores:%u\n",tiempo_total,N);
  #pragma omp parallel for
  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]);
}
else
  printf("Tiempo:%11.9f\t / Tamanio Vectores:%u\t/ V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) / / V1[%d]+V2[%d]
        tiempo_total,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);

#ifdef VECTOR_DYNAMIC
  free(v1); // libera el espacio reservado para v1
  free(v2); // libera el espacio reservado para v2
  free(v3); // libera el espacio reservado para v3
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
 DavidMartinezDiaz - 21-04-03 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/
2/2/AC/A/bp1/ejer7     gcc -fopenmp -O2 sp-OpenMP-for.c -o sp-OpenMP-for
 DavidMartinezDiaz - 21-04-03 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/
2/2/AC/A/bp1/ejer7     ./sp-OpenMP-for 8                    ✓ | 11:29:53
Tamanio Vectores:8 (4 B)
Tiempo:0.000867900       / Tamanio Vectores:8
/ V1[6]+V2[6]=V3[6](1.400000+0.200000=1.600000) /
/ V1[4]+V2[4]=V3[4](1.200000+0.400000=1.600000) /
/ V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) /
/ V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
/ V1[1]+V2[1]=V3[1](0.900000+0.700000=1.600000) /
/ V1[3]+V2[3]=V3[3](1.100000+0.500000=1.600000) /
/ V1[2]+V2[2]=V3[2](1.000000+0.600000=1.600000) /
/ V1[5]+V2[5]=V3[5](1.300000+0.300000=1.600000) /
 DavidMartinezDiaz - 21-04-03 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/
2/2/AC/A/bp1/ejer7     ./sp-OpenMP-for 11                   ✓ | 11:29:58
Tamanio Vectores:11 (4 B)
Tiempo:0.000012000       / Tamanio Vectores:11  / V1[0]+V2[0]=V3[0](1.100000
+1.100000=2.200000) / / V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
 DavidMartinezDiaz - 21-04-03 | dmartinez01@LAPTOP-H62PMCCC | /mnt/d/U/De/I/
2/2/AC/A/bp1/ejer7     |                                    ✓ | 11:30:23
```

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 `sp-OpenMP-sections.c`

```c
// Inicializamos los 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;
    }
  }

  #pragma omp single
    tiempo_inicial = omp_get_wtime();
```

```c
  #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++)
      v3[k] = v1[k] + v2[k];

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

  #pragma omp single
    tiempo_final = omp_get_wtime();
}

tiempo_total = tiempo_final - tiempo_inicial;

//Imprimir resultado de la suma y el tiempo de ejecución
if (N<10) {
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]);
}
else
  printf("Tiempo:%11.9f\t / Tamaño Vectores:%u\t/ 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",
         tiempo_total,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
 DavidMartinezDiaz - 21-04-03 | dmartinez01@LAPTOP-H62PMCC
C | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer8     gcc -O2 -fopenmp
sp-OpenMP-sections.c -o sp-OpenMP-sections
 DavidMartinezDiaz - 21-04-03 | dmartinez01@LAPTOP-H62PMCC
C | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer8     ./sp-OpenMP-secti
ons 8
Tiempo:0.000003900       / 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) /
 DavidMartinezDiaz - 21-04-03 | dmartinez01@LAPTOP-H62PMCC
C | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer8     ./sp-OpenMP-secti
ons 11
Tiempo:0.000051500       / Tamaño Vectores:11   / V1[0]+V2
[0]=V3[0](1.100000+1.100000=2.200000) / / V1[10]+V2[10]=V3
[10](2.100000+0.100000=2.200000) /
 DavidMartinezDiaz - 21-04-03 | dmartinez01@LAPTOP-H62PMCC
C | /mnt/d/U/De/I/2/2/AC/A/bp1/ejer8
```

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. NOTA: Al contestar piense sólo en el código, no piense en el computador en el que lo va a ejecutar.

RESPUESTA:

En el ejercicio 7, vamos a poder utilizar todos los cores que tenga mi pc, ya que en el codigo no se ha implementado la function “omp-set-num-threads”, que establece el número de threads que vamos a utilizar en el programa.

Sin embargo en el ejercicio 8, como cada bucle for lo ejecuta una sola hebra, su grado de paralelizacion ya no es el mismo, por lo que no se llegara a utilizer el maximo creo yo, que en este caso, serian 4 cores fisicos, uno para cada bucle for.

10. Rellenar una tabla como la Tabla 210 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. Escribir un script para realizar las ejecuciones necesarias utilizando como base el script del seminario de BP0 (se deben imprimir en el script al menos las variables de entorno que ya se imprimen en el script de BP0). 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. Observar que el número de componentes en la tabla llega hasta **67108864.**

RESPUESTA: Captura del script implementado sp-OpenMP-script10.sh

```
#!/bin/bash

#Todos los scripts que se hagan para atcgrid deben incluir lo siguiente:
#Se asigna al trabajo el nombre SumaVectoresC_vlocales
#PBS -N SumaVectoresC_vlocales
#Se asigna al trabajo la cola ac
#PBS -q ac
#Se imprime información del trabajo usando variables de entorno de PBS
echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
echo "Id. del trabajo: $PBS_JOBID"

echo "Nombre  del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"
echo "Nodos asignados al trabajo:"
cat $PBS_NODEFILE
# FIN del trozo que deben incluir todos los scripts
for (( i = 16384; i <= 67108864 ; i*=2 )); do
    ./$1 $i
done
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

CAPTURAS DE PANTALLA (mostrar la ejecución en atcgrid – envío(s) a la cola):

```
[e2estudiante14@atcgrid ejer10]$ sbatch -p ac ./sp-OpenMP-script10.sh
Submitted batch job 78338
[e2estudiante14@atcgrid ejer10]$ cat slurm-78338.out
Tamanio Vectores:16384 (4 B)
Tiempo:0.000432747      / Tamanio Vectores:16384      / V1[0]+V2[0]=V3[0](1638.400000
+1638.400000=3276.800000) / / V1[16383]+V2[16383]=V3[16383](3276.700000+0.100000=3276.8
00000) /
Tamanio Vectores:32768 (4 B)
Tiempo:0.000881658      / Tamanio Vectores:32768      / V1[0]+V2[0]=V3[0](3276.800000
+3276.800000=6553.600000) / / V1[32767]+V2[32767]=V3[32767](6553.500000+0.100000=6553.6
00000) /
Tamanio Vectores:65536 (4 B)
Tiempo:0.000488072      / Tamanio Vectores:65536      / V1[0]+V2[0]=V3[0](6553.600000
+6553.600000=13107.200000) / / V1[65535]+V2[65535]=V3[65535](13107.100000+0.100000=1310
7.200000) /
Tamanio Vectores:131072 (4 B)
Tiempo:0.000603504      / Tamanio Vectores:131072      / V1[0]+V2[0]=V3[0](13107.20000
0+13107.200000=26214.400000) / / V1[131071]+V2[131071]=V3[131071](26214.300000+0.100000
=26214.400000) /
```

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 y cores lógicos utilizados.

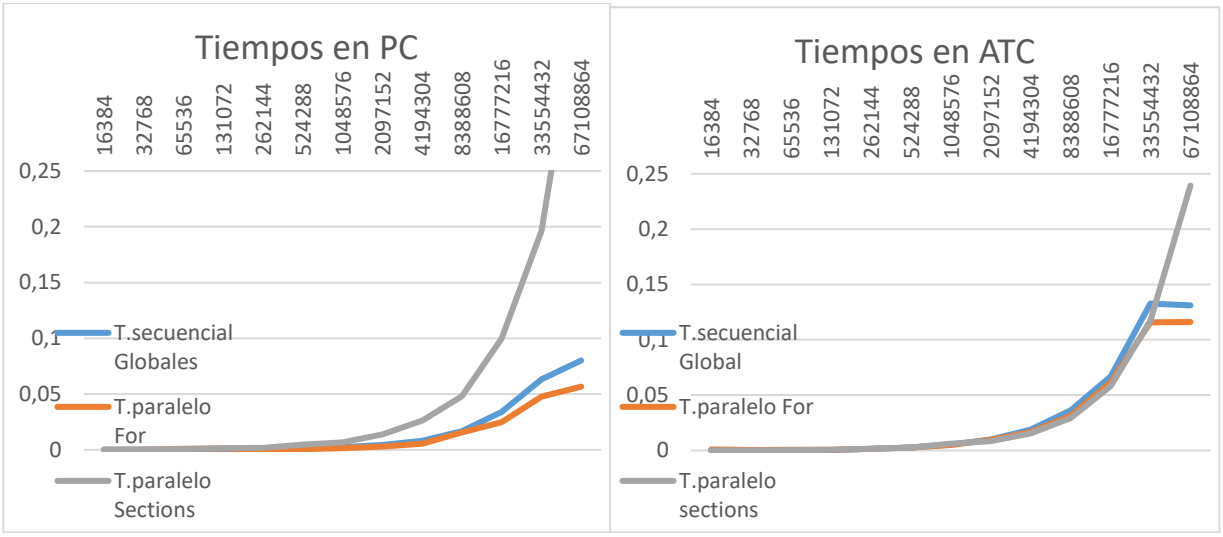
Para PC:

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión `for`) 6 threads = cores lógicos = cores físicos	T. paralelo (versión `sections`) 6 threads = cores lógicos = cores físicos
16384	0.000097600	0.000031400	0.000135800
32768	0.000140400	0.000055600	0.000320000
65536	0.000132900	0.000879900	0.000524700
131072	0.000283000	0.001213900	0.001614100
262144	0.000495600	0.000312000	0.001664800
524288	0.000967200	0.000600500	0.004641700
1048576	0.002228500	0.001345800	0.006590600
2097152	0.004278200	0.003030000	0.013758000
4194304	0.008132800	0.005573200	0.026234300
8388608	0.016687300	0.015657900	0.048014300
16777216	0.033854200	0.024733300	0.099617300
33554432	0.063229200	0.047648600	0.196372300
67108864	0.080136900	0.056642900	0.413890500

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 y cores lógicos utilizados.

Para ATC:

Nº de Componentes	T. secuencial vect. Globales 1 thread=core	T. paralelo (versión `for`) 12 threads = cores lógicos = cores físicos	T. paralelo (versión `sections`) 12 threads = cores lógicos = cores físicos
16384	0.000430682	0.000893410	0.000103626
32768	0.000418737	0.000469036	0.000179239
65536	0.000391069	0.000509299	0.000409625
131072	0.000443401	0.000736352	0.000749394
262144	0.001443790	0.001429945	0.001445636
524288	0.002761257	0.002568111	0.002555914
1048576	0.005237209	0.005067702	0.006035995
2097152	0.009870978	0.009554561	0.008564293
4194304	0.018729252	0.016653799	0.015439760
8388608	0.036239037	0.031263638	0.029247530
16777216	0.066939411	0.061516255	0.058253612
33554432	0.132744833	0.115652569	0.117332138
67108864	0.131187172	0.116130535	0.239396375



11. Rellenar una tabla como la 12Tabla 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 (que debe coincidir con el número cores físicos y lógicos) que usan los códigos. Escribir un script para realizar las ejecuciones necesarias utilizando como base el script del seminario de BP0 (se deben imprimir en el script al menos las variables de entorno que ya se imprimen en el script de BP0) ¿El tiempo de CPU que se obtiene es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

RESPUESTA: Captura del script implementado `sp-OpenMP-script11.sh`

```
#!/bin/bash

#Todos los scripts que se hagan para atcgrid deben incluir lo siguiente:
#Se asigna al trabajo el nombre SumaVectoresC_vlocales
#PBS -N SumaVectoresC_vlocales
#Se asigna al trabajo la cola ac
#PBS -q ac
#Se imprime información del trabajo usando variables de entorno de PBS
echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
echo "Id. del trabajo: $PBS_JOBID"

echo "Nombre  del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"
echo "Nodos asignados al trabajo:"
cat $PBS_NODEFILE
# FIN del trozo que deben incluir todos los scripts
for (( i = 8388608; i <= 67108864 ; i*=2 )); do
    time ./SumaVectoresC $i
done
```

(RECUERDE ADJUNTAR LOS CÓDIGOS AL .ZIP)

CAPTURAS DE PANTALLA (ejecución en atcgrid):

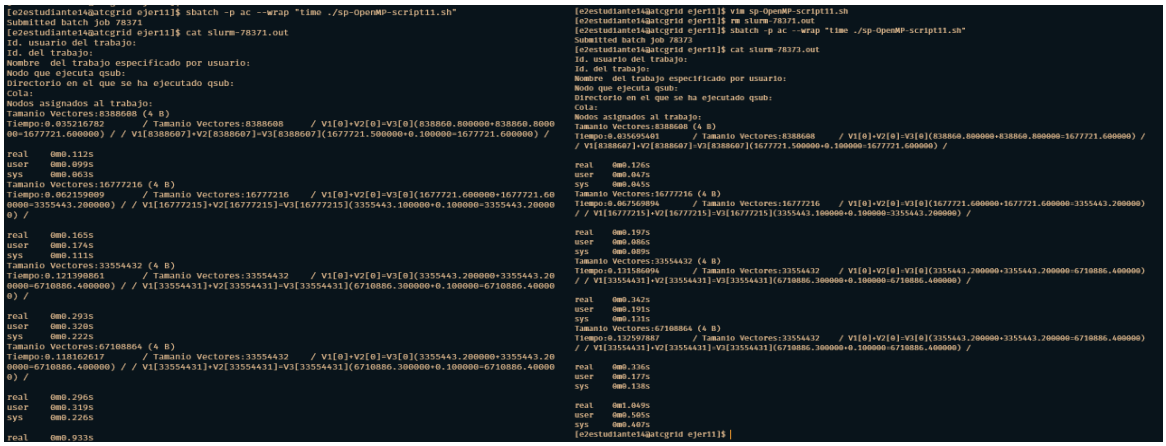


Tabla 3. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados.

Nº de Componentes	Tiempo secuencial vect. Globales 1 thread = 1 core lógico = 1 core físico			Tiempo paralelo/versión `for` 8 Threads = cores lógicos=cores físicos		
	Elapsed	*CPU-user*	*CPU- sys*	*Elapsed*	*CPU-user*	*CPU- sys*
8388608	real 0m0.125s			real 0m0.112s		
	user 0m0.031s			user 0m0.099s		
	sys 0m0.045s			sys 0m0.063s		
16777216	real 0m0.176s			real 0m0.165s		
	user 0m0.083s			user 0m0.174s		
	sys 0m0.073s			sys 0m0.111s		
33554432	real 0m0.327s			real 0m0.293s		
	user 0m0.160s			user 0m0.320s		
	sys 0m0.148s			sys 0m0.222s		
67108864	real 0m0.330s			real 0m0.296s		
	user 0m0.169s			user 0m0.319s		
	sys 0m0.141s			sys 0m0.226s		

Como ya hemos comentado en algún ejercicio anterior, el tiempo real puede que sea mayor o menor que el tiempo de ejecución (usuario + sistema), ya que el programa puede estar esperando sin estar ejecutando nada y aun así el tiempo real estaría contado ese tiempo de espera.

Además, existe el caso en el que el tiempo sea menor, ya que se podrían dividir el trabajo los hijos o threads que al final se sumarian al usuario y al sistema.

Aunque generalmente, según los valores que nos ha ido dando el tiempo de ejecución es mayor que el tiempo real, ya sea por el tiempo de espera o por simplemente el tiempo que tardan en realizar los cálculos y operaciones de los de los vectores.