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): Daniel Monjas Miguélez
Grupo de prácticas y profesor de prácticas:Miércoles, Mancia Anguita
Fecha de entrega: 25 de marzo de 2020
Fecha evaluación en clase: 25 de marzo de 2020

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

Ejercicios basados en los ejemplos del seminario práctico

1. Usar la directiva `parallel` combinada con directivas de trabajo compartido en los ejemplos `bucle-for.c` y `sections.c` del seminario. Incorporar el código fuente resultante al cuaderno de prácticas.

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

```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`

```c
#include <stdio.h>
#include <omp.h>
void funcA() {

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

printf("En funcB: esta sección la ejecuta el thread %d\n", omp_get_thread_num());

}

main() {

#pragma omp parallel sections

{

#pragma omp section
(void) funcA();

#pragma omp section
(void) funcB();

}
}
```

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`

CAPTURAS DE PANTALLA:

```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 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
{
for (i=0; i<n; i++){
        printf("b[%d] = %d\t Hebra: %d\t",i,b[i],omp_get_thread_num());
}
}

}


printf("\n");

}
```

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp1/ejer2] 2020-03-11 miércoles
$./singleModificado
Introduce valor de inicialización a: 345
Single ejecutada por el thread 6
b[0] = 345       Hebra: 7
b[1] = 345       Hebra: 7
b[2] = 345       Hebra: 7
b[3] = 345       Hebra: 7
b[4] = 345       Hebra: 7
b[5] = 345       Hebra: 7
b[6] = 345       Hebra: 7
b[7] = 345       Hebra: 7
b[8] = 345       Hebra: 7
```

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`

CAPTURAS DE PANTALLA:

```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 inicialización a: ");

scanf("%d", &a );

printf("Single ejecutada por el thread %d\n",omp_get_thread_num());

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

#pragma omp master
{
for (i=0; i<n; i++){
        printf("b[%d] = %d\t Hebra: %d\n",i,b[i],omp_get_thread_num());
}
}

}


printf("\n");

}
```

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp1/ejer3] 2020-03-11 mi
ércoles
$./singleMaster
Introduce valor de inicialización a: 20
Single ejecutada por el thread 7
b[0] = 20        Hebra: 0
b[1] = 20        Hebra: 0
b[2] = 20        Hebra: 0
b[3] = 20        Hebra: 0
b[4] = 20        Hebra: 0
b[5] = 20        Hebra: 0
b[6] = 20        Hebra: 0
b[7] = 20        Hebra: 0
b[8] = 20        Hebra: 0

[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp1/ejer3] 2020-03-11 mi
ércoles
$./singleMaster
Introduce valor de inicialización a: 12
Single ejecutada por el thread 1
b[0] = 12        Hebra: 0
b[1] = 12        Hebra: 0
b[2] = 12        Hebra: 0
b[3] = 12        Hebra: 0
b[4] = 12        Hebra: 0
b[5] = 12        Hebra: 0
b[6] = 12        Hebra: 0
b[7] = 12        Hebra: 0
b[8] = 12        Hebra: 0
```

RESPUESTA A LA PREGUNTA: Se puede observar que independientemente de las veces que se ejecute el programa, es la hebra master la encargada de mostrar los resultado, a diferencia del caso anterior, en el que cualquier hebra que estuviese libre podía encargarse de la impresión de resultados

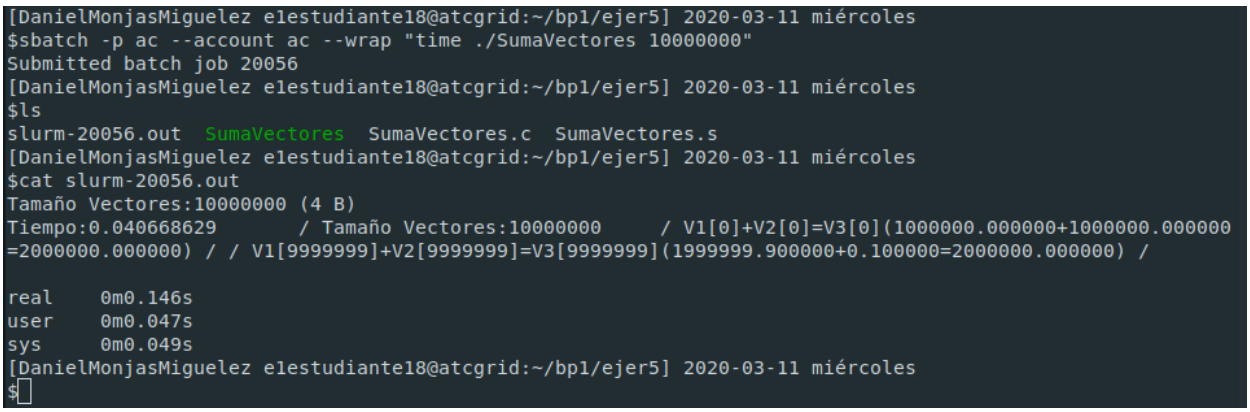
4. ¿Por qué si se elimina directiva `barrier` en el ejemplo `master.c` la suma que se calcula e imprime no siempre es correcta? Responda razonadamente.

RESPUESTA: Se debe a que al no sincronizarse las hebras ya que se omite la directiva `barrier`, la hebra `master` mostrará directamente el valor de la variable compartirda que haya en el momento, sin tener en cuenta los resultados que introduzcan posteriormente cualquiera de las hebras que esten trabajando paralelamente.

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,…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:



   RESPUESTA: El tiempo real es mayor que la suma del tiempo de usuario y del tiempo de sistema. Habrá algún tiempo añadido ya que el sistema estará ejecutando algún trabajo no relacionado con el programa que yo he pedido ejecutar, de forma que aumenta el tiempo real.

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

   CAPTURAS DE PANTALLA (que muestren la generación del código ensamblador y del código ejecutable, y la obtención de los tiempos de ejecución):

   RESPUESTA: cálculo de los MIPS y los MFLOPS

   Para el tiempo de CPU utilizaremos el obtenido con time del ejercicio anterior. En primer lugar contaremos el nº de instrucciones del bucle. Tomaremos como punto de partida desde clock_gettime, vemos que el xorl se ejecuta una sola vez, y como punto final de una iteración ja, dando un total de NI=N*6+1, donde N es el número de iteraciones del bucle. Para las operaciones con coma flotante buscamos y vemos que en el cuerpo del bucle solo hay una instruccion con coma flotante, addsd, luego Nº op.coma flotante=1.

   Con esto y el tiempo de CPU, podemos hacer el cálculo requerido

MIPS(10)=(6*10+1)/(T_CPU*10^6)=61/(0.000364401*10^6)=0.167398

MIPS(10000000)=(6*10000000+1)/(T_CPU*10^6)=60000001/(0.040668629*10^6)=1475.33867

MFLOPS(10)=(1*10)/(T_CPU*10^6)=10/(0.000364401*10^6)=0.027442295

MFLOPS(10000000)=(1*10000000)/(T_CPU*10^6)=10000000/(0.040668629*10^6)=245.8897742

RESPUESTA: Captura que muesre 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
        pxor    %xmm1, %xmm1
        movq    16(%rsp), %rax
        subq    (%rsp), %rax
        cvtsi2sdq       %rax, %xmm1
        divsd   .LC6(%rip), %xmm0
        addsd   %xmm1, %xmm0
        cmpl    $9, %ebx
        ja      .L22
        movl    %ebp, %esi
        movl    $.LC5, %edi
        movl    $1, %eax
        xorl    %ebx, %ebx
        call    printf
        .p2align 4,,10
        .p2align 3
```

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

```
#pragma omp parallel
{

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

double start=omp_get_wtime();

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

double end=omp_get_wtime();

ncgt=end - start;

//Imprimir resultado de la suma y el tiempo de ejecución
if (N<10) {
printf("Tiempo:%11.9f\t / Tamaño Vectores:%u\n",ncgt,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", ncgt,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):

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp1/ejer7] 2020-03-16 lu
nes
$gcc -O2 -fopenmp -o listado_paralelo listado_paralelo.c
listado_paralelo.c: In function ‘main’:
listado_paralelo.c:46:33: warning: format ‘%u’ expects argument of type ‘unsigned int’, but argument 3 ha
s type ‘long unsigned int’ [-Wformat=]
   printf("Tamaño Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                                ~^
                                %lu
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp1/ejer7] 2020-03-16 lu
nes
$./listado_paralelo 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000019316       / 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) /
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp1/ejer7] 2020-03-16 lu
nes
$./listado_paralelo 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000015751       / 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) /
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp1/ejer7] 2020-03-16 lu
nes
$
```

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

```
  double start = omp_get_wtime();

  #pragma omp parallel 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 < N/2; i++){
    v3[i] = v1[i] + v2[i];
  }

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

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

  }

  double end = omp_get_wtime();

  ncgt=end-start;

  //Imprimir resultado de la suma y el tiempo de ejecución
  if (N<10) {
  printf("Tiempo:%11.9f\t / Tamaño Vectores:%u\n",ncgt,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", ncgt,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):

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp1/ejer8] 2020-03-18 mi
ércoles
$gcc -O2 -fopenmp -o listado_sections listado_sections.c
listado_sections.c: In function 'main':
listado_sections.c:46:33: warning: format '%u' expects argument of type 'unsigned int', but argument 3 ha
s type 'long unsigned int' [-Wformat=]
   printf("Tamaño Vectores:%u (%u B)\n",N, sizeof(unsigned int));
                                ~^
                                %lu
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp1/ejer8] 2020-03-18 mi
ércoles
$./listado_sections 8
Tamaño Vectores:8 (4 B)
Tiempo:0.016102940       / 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) /
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp1/ejer8] 2020-03-18 mi
ércoles
$./listado_sections 12
Tamaño Vectores:12 (4 B)
Tiempo:0.020012217       / Tamaño Vectores:12   / V1[0]+V2[0]=V3[0](1.200000+1.200000=2.400000) / / V1[11
]+V2[11]=V3[11](2.300000+0.100000=2.400000) /
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp1/ejer8] 2020-03-18 mi
ércoles
$./listado_sections 15
Tamaño Vectores:15 (4 B)
Tiempo:0.009574547       / Tamaño Vectores:15   / V1[0]+V2[0]=V3[0](1.500000+1.500000=3.000000) / / V1[14
]+V2[14]=V3[14](2.900000+0.100000=3.000000) /
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp1/ejer8] 2020-03-18 mi
ércoles
$
```

9. ¿Cuántos threads y cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 7? Razone su respuesta. ¿Cuántos threads y cuantos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta.

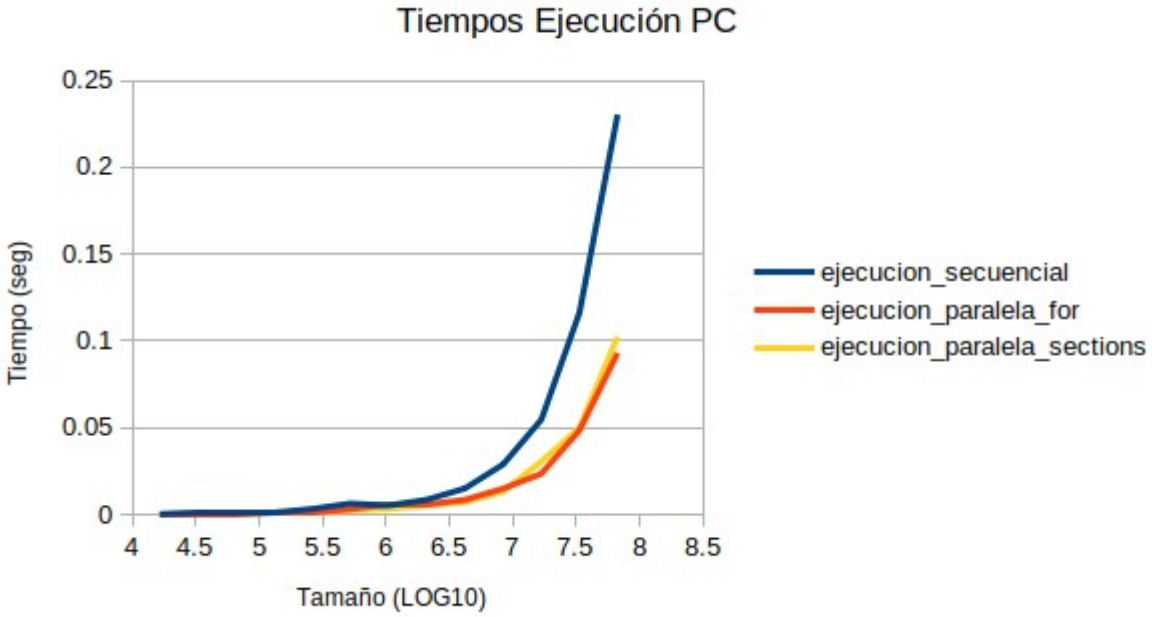
   RESPUESTA:

   En el ejercicio 7 puedo utilizar un máximo 8 threads y 4 cores físicos en mi versión del ejercicio 7 puesto que no he establecido límite al número de threads que el computador puede utilizar, por tanto utilizará todos los que requiera y de los que disponga.

   En el ejercicio 8, al dividir el for en 4 sections, cada uno de ellos se ejecutará en una hebra, dando que el máximo de hebras a utilizar son 4 que coincide con el número máximo de cores.

10. Rellenar una tabla como la Tabla 2 para atcgrid y otra para su PC con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución del trozo de código que realiza la suma en paralelo (este es el tiempo que deben imprimir los programas). Ponga en la tabla el número de threads/cores que usan los códigos (use el máximo número de cores físicos del computador que como máximo puede aprovechar el código, no use un número de threads superior al número de cores físicos). Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute código que imprima todos los componentes del resultado cuando este número sea elevado.

    RESPUESTA:


Tiempos Ejecución PC
Tiempo (seg)
0.25
0.2
0.15
0.1
0.05
0
4
4.5
5
5.5
6
6.5
7
7.5
8
8.5
Tamaño (LOG10)
ejecucion_secuencial
ejecucion_paralela_for
ejecucion_paralela_sections

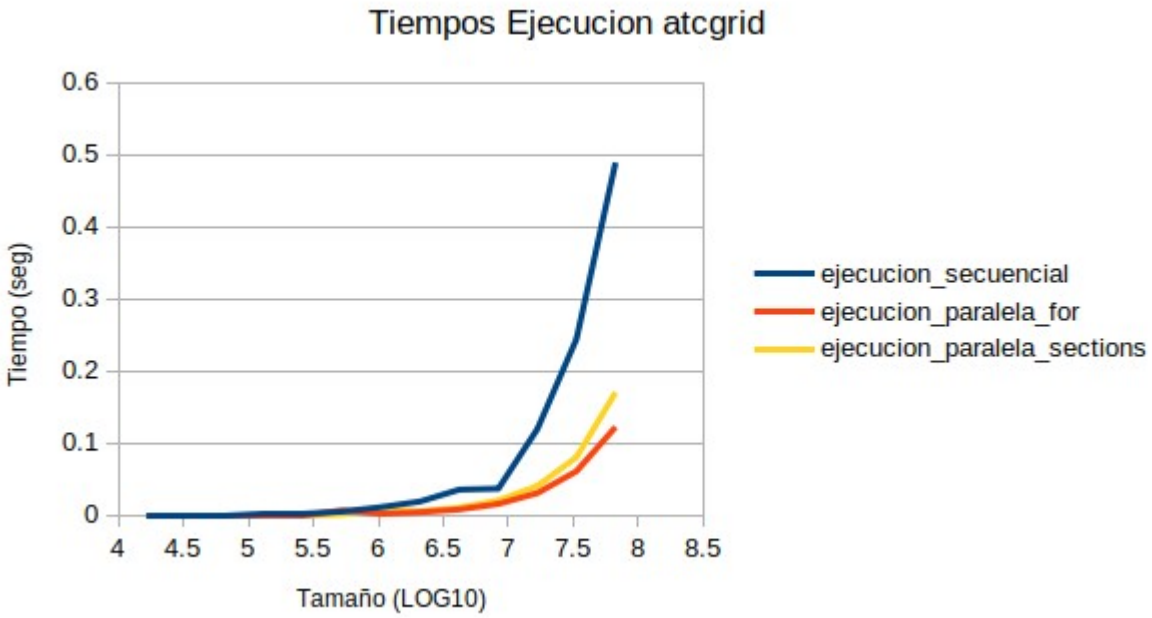

Tiempos Ejecucion atcgrid
Tiempo (seg)
0.6
0.5
0.4
0.3
0.2
0.1
0
4
4.5
5
5.5
6
6.5
7
7.5
8
8.5
Tamaño (LOG10)
ejecucion_secuencial
ejecucion_paralela_for
ejecucion_paralela_sections

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

Tabla PC

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión `for`) 4 threads/cores	T. paralelo (versión `sections`) 4 threads/cores
16384	0.000204365	0.000153576	0.000097457
32768	0.000563140	0.000269946	0.000225956
65536	0.000825127	0.000431396	0.000315288
131072	0.001579341	0.000784296	0.000624993
262144	0.003160515	0.001557885	0.001118197
524288	0.006246992	0.003074897	0.002394976
1048576	0.005194459	0.005685232	0.003070426
2097152	0.008465219	0.006212665	0.005277613
4194304	0.015018488	0.008388812	0.007169635
8388608	0.028851006	0.014888589	0.013265990
16777216	0.054606367	0.023420311	0.030442066
33554432	0.116274106	0.048379215	0.050036533
67108864	0.230419173	0.093036232	0.102360584

Tabla atcgrid

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión `for`) 12 threads/cores	T. paralelo (versión `sections`) 12 threads/cores
16384	0.000123876	0.000042621	0.000036288
32768	0.000232896	0.000121748	0.000061084
65536	0.000492944	0.000116793	0.000123464
131072	0.001077081	0.000191428	0.000221832
262144	0.002383923	0.000408312	0.000427026
524288	0.004953759	0.006924845	0.000955377
1048576	0.011174753	0.001939500	0.007813634
2097152	0.019030617	0.004242457	0.005531732
4194304	0.035507474	0.008057483	0.010589959
8388608	0.036939515	0.015638370	0.020567667
16777216	0.119714217	0.030729659	0.040521765
33554432	0.244265385	0.060902774	0.080504054
67108864	0.488991315	0.122273035	0.170243233

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: Para la ejecución secuencial el tiempo real es mayor que la suma de los tiempos de CPU, esto se debe a que probablemente haya interrupciones ajenas al programa, que aumentan el tiempo de ejecución

del mismo. Por otro lado para la ejecución en paralelo, se puede apreciar que los tiempos reales son menores que el tiempo de CPU. Esto se debe a que mientras que para el tiempo real se toma el tiempo de ejecución del programa, para los tiempos de CPU se realiza la suma de los tiempos de CPU de las distintas hebras, de forma que la suma de todos los tiempos de CPU de las distintas hebras es mayor que el tiempo de ejecución del programa.

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

Nº de Componentes	**Tiempo secuencial vect. Globales 1 thread/core**			**Tiempo paralelo/versión `for` 12 Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	0.021	0.000	0.003	0.029	0.021	0.023
131072	0.029	0.000	0.004	0.027	0.042	0.024
262144	0.029	0.004	0.002	0.032	0.017	0.046
524288	0.074	0.004	0.005	0.028	0.096	0.038
1048576	0.029	0.006	0.010	0.024	0.049	0.062
2097152	0.081	0.016	0.012	0.029	0.102	0.059
4194304	0.078	0.028	0.022	0.081	0.186	0.094
8388608	0.140	0.054	0.041	0.101	0.365	0.151
16777216	0.165	0.072	0.083	0.141	0.718	0.281
33554432	0.321	0.146	0.165	0.180	1.377	0.598
67108864	0.622	0.321	0.286	0.360	2.664	1.138