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): Juan Carlos Ruiz García
Grupo de prácticas: C2
Fecha de entrega: 26/03/2017
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: código fuente `bucle-forModificado.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: código fuente `sectionsModificado.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());
}
int 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: código fuente `singleModificado.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 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\n",i,b[i]);
      printf("Identificador del thread %d\n",omp_get_thread_num());
    }
  }
  return(0);
}
```

CAPTURAS DE PANTALLA:

```
juanka1995@juanka-desktop ~/Dropbox/INGENIER
icas/practica_1 $ export OMP_DYNAMIC=FALSE
juanka1995@juanka-desktop ~/Dropbox/INGENIER
icas/practica_1 $ export OMP_NUM_THREADS=8
juanka1995@juanka-desktop ~/Dropbox/INGENIER
icas/practica_1 $ ./single
Introduce valor de inicialización a: 23
Single ejecutada por el thread 2
b[0] = 23
b[1] = 23
b[2] = 23
b[3] = 23
b[4] = 23
b[5] = 23
b[6] = 23
b[7] = 23
b[8] = 23
Identificador del thread 2
juanka1995@juanka-desktop ~/Dropbox/INGENIER
icas/practica_1 $ ./single
Introduce valor de inicialización a: 14
Single ejecutada por el thread 2
b[0] = 14
b[1] = 14
b[2] = 14
b[3] = 14
b[4] = 14
b[5] = 14
b[6] = 14
b[7] = 14
b[8] = 14
Identificador del thread 2
```

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: código fuente `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 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\n",i,b[i]);
      printf("Identificador del thread %d\n",omp_get_thread_num());
    }
  }
  return(0);
}
```

CAPTURAS DE PANTALLA:

```
juanka1995@juanka-desktop ~/Dropbox/INGENIERIA INFORMATICA/2016-2017/2o CUAT
icas/practica_1 $ gcc -O2 -fopenmp singleModificado2.c -o singleModificado2
juanka1995@juanka-desktop ~/Dropbox/INGENIERIA INFORMATICA/2016-2017/2o CUAT
icas/practica_1 $ export OMP_DYNAMIC=FALSE
juanka1995@juanka-desktop ~/Dropbox/INGENIERIA INFORMATICA/2016-2017/2o CUAT
icas/practica_1 $ export OMP_NUM_THREADS=8
juanka1995@juanka-desktop ~/Dropbox/INGENIERIA INFORMATICA/2016-2017/2o CUAT
icas/practica_1 $ ./singleModificado2
Introduce valor de inicialización a: 23
Single ejecutada por el thread 2
b[0] = 23
b[1] = 23
b[2] = 23
b[3] = 23
b[4] = 23
b[5] = 23
b[6] = 23
b[7] = 23
b[8] = 23
Identificador del thread 0
juanka1995@juanka-desktop ~/Dropbox/INGENIERIA INFORMATICA/2016-2017/2o CUAT
icas/practica_1 $ ./singleModificado2
Introduce valor de inicialización a: 14
Single ejecutada por el thread 1
b[0] = 14
b[1] = 14
b[2] = 14
b[3] = 14
b[4] = 14
b[5] = 14
b[6] = 14
b[7] = 14
b[8] = 14
Identificador del thread 0
```

RESPUESTA A LA PREGUNTA: La única diferencia es que la hebra que ejecuta la *directiva master* es siempre la 0 (que es la master y siempre se identifica con este número). En el ejercicio anterior, la hebra que ejecutaba la *directiva single* era la primera que llegaba a dicha directiva, por eso el identificador de la hebra cambiaba en las distintas ejecuciones del programa.

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

RESPUESTA: Esto se debe a que la *directiva master* no tiene barreras implícitas. Imaginemos el caso en el que la hebra master llega antes que las demas a la *directiva master*,sin que todas las demás hayan terminado de pasar por la *directiva atomic*, esto produciría que la suma no se haya completado aún pero la hebra master mostraría el resultado que ella conoce hasta ese momento, siendo un resultado de la suma erróneo.

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 el PC local, 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 mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

CAPTURAS DE PANTALLA:

```
juanka1995@juanka-desktop ~/Dropbox/INGENIERIA INFORMATICA/2016-2017/2o CUATRIMESTRE/practicas/AC/Pract
icas/practica_1 $ gcc -O2 SumaVectores.c -o SumaVectores -lrt
juanka1995@juanka-desktop ~/Dropbox/INGENIERIA INFORMATICA/2016-2017/2o CUATRIMESTRE/practicas/AC/Pract
icas/practica_1 $ time ./SumaVectores 10000000
Tiempo(seg.):0.110522536        Tamaño Vectores:10000000        V1[0]+V2[0]=V3[0](1000000.000000+10000
00.000000=2000000.000000) V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000)

real    0m0.440s
user    0m0.360s
sys     0m0.076s
```

CPU time = user + sys = 0,360s + 0,076s = 0,436s

Elapsed = 0,440s

Elapsed >= CPU Time

Esto es debido a que solo hay **un** flujo de control del programa y es de tipo secuencial, por lo que la suma del **tiempo user** + **tiempo sys** siempre va aser >= que el **tiempo real**.

- Con un flujo de control
  - elapsed >= CPU time

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:

Ejecución en atcgrid con 10 y 10.000.000

```
[C2estudiante17@atcgrid ~]$ echo './SumaVectores 10' | qsub -q ac
52235.atcgrid
[C2estudiante17@atcgrid ~]$ qstat
Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
52235.atcgrid              STDIN            C2estudiante17  00:00:00 C ac
[C2estudiante17@atcgrid ~]$ ls -l
total 16
-rw------- 1 C2estudiante17 C2estudiante17    0 Mar 26 09:37 STDIN.e52235
-rw------- 1 C2estudiante17 C2estudiante17  140 Mar 26 09:37 STDIN.o52235
-rwxr-xr-x 1 C2estudiante17 C2estudiante17 8968 Mar 26 09:34 SumaVectores
[C2estudiante17@atcgrid ~]$ echo './SumaVectores 10000000' | qsub -q ac
52236.atcgrid
[C2estudiante17@atcgrid ~]$ qstat
Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
52236.atcgrid              STDIN            C2estudiante17  00:00:00 C ac
[C2estudiante17@atcgrid ~]$ ls -l
total 20
-rw------- 1 C2estudiante17 C2estudiante17    0 Mar 26 09:37 STDIN.e52235
-rw------- 1 C2estudiante17 C2estudiante17    0 Mar 26 09:37 STDIN.e52236
-rw------- 1 C2estudiante17 C2estudiante17  140 Mar 26 09:37 STDIN.o52235
-rw------- 1 C2estudiante17 C2estudiante17  194 Mar 26 09:37 STDIN.o52236
-rwxr-xr-x 1 C2estudiante17 C2estudiante17 8968 Mar 26 09:34 SumaVectores
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

	Tamaño 10	**Tamaño 10.000.000**
MIPS	NI = 6x10+3 = 63 Tcpu = 0.000003262 MIPS = $\frac{63}{0.000003262 * 10^6}$ = 19313,3047	NI = 6x10,000,000+3 = 60,000,003 Tcpu = 0.048564205 MIPS = $\frac{63000003}{0.048564205 * 10^6}$ = 1235,47792
MFLOPS	NIcf = ? Tcpu = 0.000003262 MFLOPS = $\frac{NIcf}{0.000003262 * 10^6}$	NIcf = ? Tcpu = 0.048564205 MFLOPS = $\frac{NIcf}{0.000003262 * 10^6}$

RESPUESTA:

Código ensamblador generado de la parte de la suma de vectores

```
                xorl    %eax, %eax
                .p2align 4,,10
                .p2align 3
.L5:
                movsd   v1(%rax), %xmm0
                addq    $8, %rax
                addsd   v2-8(%rax), %xmm0
                movsd   %xmm0, v3-8(%rax)
                cmpq    %rax, %rbx
                jne     .L5
.L6:
                leaq    16(%rsp), %rsi
                xorl    %edi, %edi
```

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: código fuente implementado

```
/* SumaVectoresOpenMP.c
Suma de dos vectores: v3 = v1 + v2
Para compilar usar (-lrt: real time library):
gcc -O2 SumaVectoresOpenMP.c -o SumaVectoresOpenMP -lrt
gcc -O2 –S SumaVectoresOpenMP.c –lrt //para generar el código ensamblador
Para ejecutar use: SumaVectoresOpenMP longitud
*/
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <omp.h>
#define PRINTF_ALL// comentar para quitar el printf ...
```

```
// que imprime todos los componentes
#define MAX 33554432 //=2^25
double v1[MAX], v2[MAX], v3[MAX];
int main(int argc, char** argv)
{
		int i;
		double cgt1,cgt2;
		double ncgt; //para tiempo de ejecución
		//Leer argumento de entrada (nº de componentes del vector)
		if (argc<2)
		{
			printf("Faltan nº componentes del vector\n");
			exit(-1);
		}
		unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)
		if (N>MAX) N=MAX;
		//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; //los
valores dependen de N
			}
		cgt1 = omp_get_wtime();
		//Calcular suma de vectores
		#pragma omp parallel for
			for(i=0; i<N; i++)
				v3[i] = v1[i] + v2[i];
		cgt2 = omp_get_wtime();
		ncgt = cgt2 - cgt1;
		//Imprimir resultado de la suma y el tiempo de ejecución
#ifdef PRINTF_ALL
		printf("Tiempo(seg.):%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(seg.):%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]);
#endif
		return 0;
}
```

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

```
juanka1995@juanka-desktop ~/Dropbox/INGENIERIA INFORMATICA/2016-2017/2o CUATRIMESTI
icas/practica_1/ejer7 $ gcc -O2 -fopenmp SumaVectoresEJ7.c -o SumaVectoresEJ7 -lrt
juanka1995@juanka-desktop ~/Dropbox/INGENIERIA INFORMATICA/2016-2017/2o CUATRIMESTI
icas/practica_1/ejer7 $ ./SumaVectoresEJ7 8
Tiempo(seg.):0.002096583        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)
juanka1995@juanka-desktop ~/Dropbox/INGENIERIA INFORMATICA/2016-2017/2o CUATRIMESTI
icas/practica_1/ejer7 $ ./SumaVectoresEJ7 11
Tiempo(seg.):0.003027273        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)
```

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: código fuente implementado

```
/* SumaVectoresOpenMP.c
Suma de dos vectores: v3 = v1 + v2
Para compilar usar (-lrt: real time library):
gcc -O2 SumaVectoresOpenMP.c -o SumaVectoresOpenMP -lrt
gcc -O2 –S SumaVectoresOpenMP.c –lrt //para generar el código ensamblador
Para ejecutar use: SumaVectoresOpenMP longitud
*/
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <omp.h>
#define PRINTF_ALL// comentar para quitar el printf ...
// que imprime todos los componentes
#define MAX 33554432 //=2^25
double v1[MAX], v2[MAX], v3[MAX];
int main(int argc, char** argv)
{
            int i;
            double cgt1,cgt2;
            double ncgt; //para tiempo de ejecución
```

```
		//Leer argumento de entrada (nº de componentes del vector)
		if (argc<2)
		{
				printf("Faltan nº componentes del vector\n");
				exit(-1);
		}
		unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)
		if (N>MAX) N=MAX;
		//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; //los valores dependen de N
						}
						#pragma omp section
						for (i = N/4; i < N/2; i++) {
								v1[i] = N*0.1+i*0.1;
v2[i] = N*0.1-i*0.1; //los valores dependen de N
						}
						#pragma omp section
						for (i = N/2; i < 3*N/4; i++) {
								v1[i] = N*0.1+i*0.1;
v2[i] = N*0.1-i*0.1; //los valores dependen de N
						}
						#pragma omp section
						for (i = 3*N/4; i < N; i++) {
								v1[i] = N*0.1+i*0.1;
v2[i] = N*0.1-i*0.1; //los valores dependen de N
						}
				}
		}
		cgt1 = 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 < 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];
						}
				}
		}
		cgt2 = omp_get_wtime();
		ncgt = cgt2 - cgt1;
```

```
            //Imprimir resultado de la suma y el tiempo de ejecución
#ifdef PRINTF_ALL
            printf("Tiempo(seg.):%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(seg.):%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]);
#endif
            return 0;
}
```

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

```
juanka1995@juanka-desktop ~/Dropbox/INGENIERIA INFORMATICA/2016-2017/2o CUATRIMEST
icas/practica_1/ejer8 $ gcc -O2 -fopenmp SumaVectoresEJ8.c -o SumaVectoresEJ8 -lrt
juanka1995@juanka-desktop ~/Dropbox/INGENIERIA INFORMATICA/2016-2017/2o CUATRIMEST
icas/practica_1/ejer8 $ ./SumaVectoresEJ8 8
Tiempo(seg.):0.002776847         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)
juanka1995@juanka-desktop ~/Dropbox/INGENIERIA INFORMATICA/2016-2017/2o CUATRIMEST
icas/practica_1/ejer8 $ ./SumaVectoresEJ8 11
Tiempo(seg.):0.001702770         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)
```

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: Tanto en el ejercicio 7 como en el ejercicio 8 no hemos hecho **export OMP_NUM_THREADS** por lo que nuestra máquina utilizará todos los cores y threads que tenga disponibles en ese momento. Sin embargo, en el ejercicio 8 al utilizar sections, se utilizaran tantos threads como sections hayamos definido en el código (en mi caso 4), por lo que puede que queden threads en desuso.

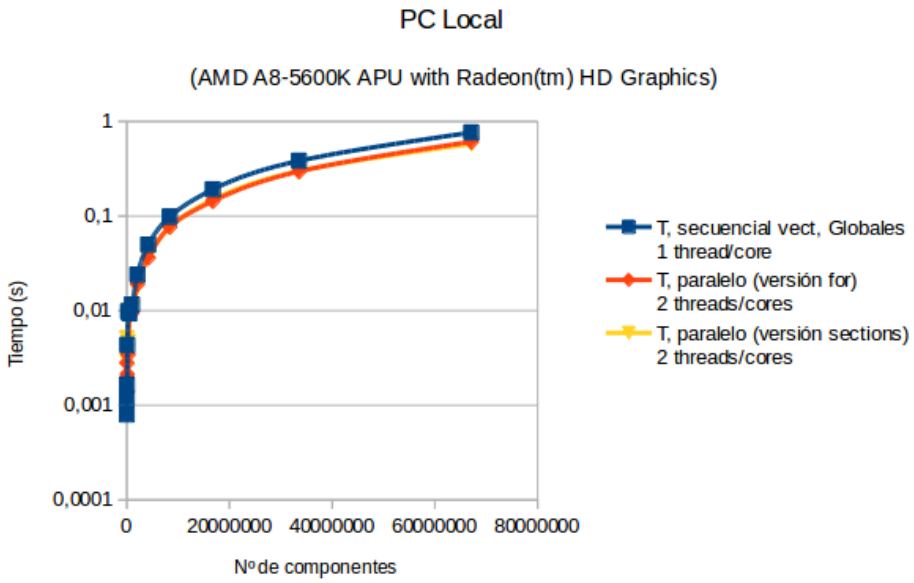
10. Rellenar una tabla como la Tabla 2 para atcgrid y otra para el PC local con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución del trozo 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 en atcgrid código que imprima todos los componentes del resultado.

RESPUESTA:

PC LOCAL

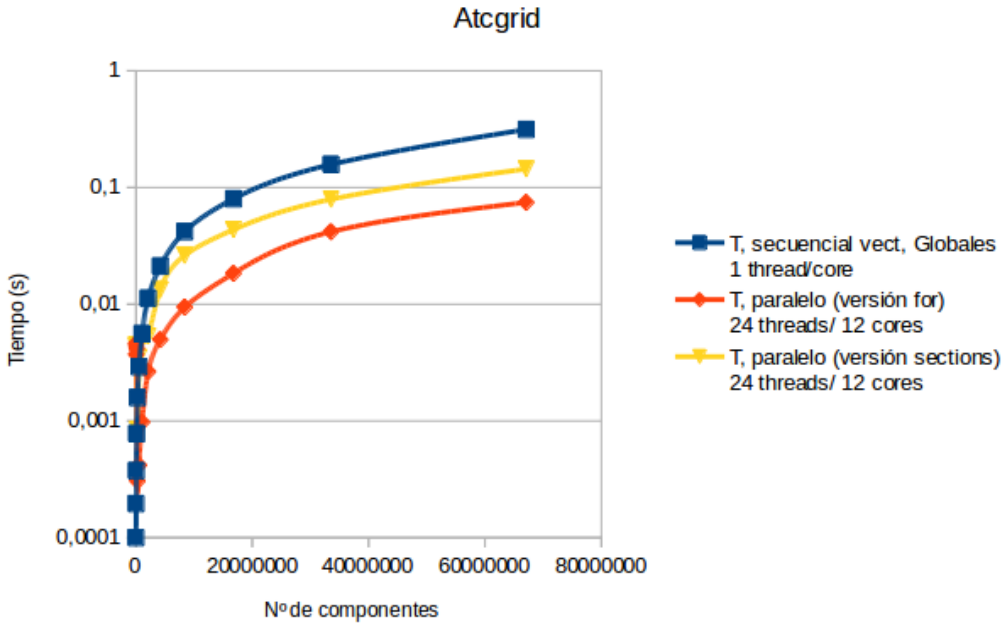
(AMD A8-5600K APU with Radeon(tm) HD Graphics)

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	0,000798579	0,001379727	0,002976734
32768	0,001189539	0,00193291	0,00298422
65536	0,001651769	0,002790903	0,005099633
131072	0,004328575	0,002120306	0,003717702
262144	0,009787753	0,003440307	0,004656051
524288	0,009338949	0,008844462	0,010017703
1048576	0,011635092	0,009495011	0,01051474
2097152	0,024040342	0,019091922	0,018305373
4194304	0,049816319	0,036267292	0,036922804
8388608	0,098518089	0,076758339	0,076855946
16777216	0,1911266	0,14375991	0,149487966
33554432	0,383119935	0,296357484	0,298367352
67108864	0,763405872	0,61012572	0,5782419



ATCGRID

Nº de Componentes	T, secuencial vect, Globales 1 thread/core	T, paralelo (versión for) 24 threads/ 12 cores	T, paralelo (versión sections) 24 threads/ 12 cores
16384	0,000100082	0,004572673	0,004344225
32768	0,000195696	0,003696362	0,004478766
65536	0,000374597	0,004338201	0,004420709
131072	0,000778127	0,004132904	0,004045537
262144	0,001582336	0,000305014	0,000840843
524288	0,002911182	0,00041802	0,003504353
1048576	0,005570166	0,000975866	0,005090065
2097152	0,011221338	0,002643786	0,005326105
4194304	0,021156504	0,004971661	0,013351476
8388608	0,041843213	0,009427441	0,026217362
16777216	0,079921668	0,01835346	0,043485239
33554432	0,157405147	0,041724296	0,079189209
67108864	0,314008397	0,074446525	0,144195233



11. Rellenar una tabla como la Tabla 3 para el PC local 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: En el secuencial el tiempo real es siempre >= que el CPUtime, esto es debido a lo ya explicado en el ejercicio 5.
Sin embargo en el paralelo el el tiempo real es siempre < que el CPUtime y esto se debe a que el CPUtime es la suma de los tiempos de cada core mientras que el tiempo real es el tiempo que verdaderamente a tardado el programa terminar su ejecución.

Nº de Componentes	T, secuencial vect, Globales 1 thread/core			T, paralelo (versión for)2 threads/cores		
	Elapsed	CPU-user	CPU-sys	Elapsed	CPU-user	CPU-sys
65536	0.009	0.004	0.000	0.006	0.004	0.004
131072	0.011	0.008	0.000	0.008	0.008	0.000
262144	0.018	0.008	0.008	0.012	0.008	0.008
524288	0.027	0.024	0.000	0.019	0.028	0.004
1048576	0.058	0.044	0.012	0.036	0.052	0.012
2097152	0.104	0.084	0.016	0.061	0.092	0.024
4194304	0.186	0.164	0.020	0.111	0.180	0.036
8388608	0.368	0.328	0.036	0.219	0.360	0.068
16777216	0.729	0.628	0.096	0.448	0.716	0.168
33554432	1.434	1.168	0.264	0.867	1.432	0.284
67108864	2.861	2.380	0.476	1.673	2.536	0.748