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):
Grupo de prácticas:
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`

```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`

```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
            {

            printf("Resultado:\n");
            for (i=0; i<n; i++)
            printf("b[%d] = %d\t",i,b[i]);
            printf("\n");
            printf("Single ha sido ejecutado por la hebra %d\n"
                   omp_get_thread_num());
            }
  }

  printf("Depués de la región parallel:\n");
  for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
  printf("\n");
            }
```

CAPTURAS DE PANTALLA:

```
[Sergio Aguilera Ramirez sergioaguilera@ei143091:~/Escritorio/P1AC] 2018-03-20 martes
$gcc -O2 -fopenmp -o single\(Modificado\) single\(Modificado\).c
[Sergio Aguilera Ramirez sergioaguilera@ei143091:~/Escritorio/P1AC] 2018-03-20 martes
$./single\(Modificado\)
Introduce valor de inicialización a: 23
Single ejecutada por el thread 2
Resultado:
b[0] = 23       b[1] = 23       b[2] = 23       b[3] = 23       b[4] = 23       b[5] = 23b
[6] = 23        b[7] = 23       b[8] = 23
Single ha sido ejecutado por la hebra 1
Depués de la región parallel:
b[0] = 23       b[1] = 23       b[2] = 23       b[3] = 23       b[4] = 23       b[5] = 23b
[6] = 23        b[7] = 23       b[8] = 23
[Sergio Aguilera Ramirez sergioaguilera@ei143091:~/Escritorio/P1AC] 2018-03-20 martes
$
```

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`

```
#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
          {

          printf("Resultado:\n");
          for (i=0; i<n; i++)
          printf("b[%d] = %d\t",i,b[i]);
          printf("\n");
          printf("Master ha sido ejecutado por la hebra %d\n",
                 omp_get_thread_num());
          }
  }

  printf("Depués de la región parallel:\n");
  for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
  printf("\n");
          }
```

CAPTURAS DE PANTALLA

```
[Sergio Aguilera Ramirez sergioaguilera@ei143091:~/Escritorio/P1AC] 2018-03-20 martes
$gcc -O2 -fopenmp -o single\(Modificado2\) single\(Modificado2\).c
[Sergio Aguilera Ramirez sergioaguilera@ei143091:~/Escritorio/P1AC] 2018-03-20 martes
$./single\(Modificado2\) 4
Introduce valor de inicialización a: 23
Single ejecutada por el thread 0
Resultado:
b[0] = 23       b[1] = 23       b[2] = 23       b[3] = 23       b[4] = 23       b[5] = 23b
[6] = 23        b[7] = 23       b[8] = 23
Master ha sido ejecutado por la hebra 0
Depués de la región parallel:
b[0] = 23       b[1] = 23       b[2] = 23       b[3] = 23       b[4] = 23       b[5] = 23b
[6] = 23        b[7] = 23       b[8] = 23
```

RESPUESTA A LA PREGUNTA: cuando utilizamos la directiva master obsevamos que dicha directiva es siempre ejecutada por la hebra 0, ha diferencia de la directiva single que esta puede ser ejecutada por cualquier hebra.

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: Si se eliminase la directiva barrier del codigo, esto supondria que en caso

   de que la hebra 0 fuese mas rapida que las demas hebras , esta suma daria un valor incorrecto ya que la hebra 0 no tiene por que esperar a las demas hebras y pasaria a la impresion del resultado sin haber sumado las hebras anteriores.

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:

```
[Sergio Aguilera Ramirez A2estudiante1@atcgrid:~] 2018-03-13 martes
echo "time ./SumaVectoresC 10000000" | qsub -q ac
66403.atcgrid
[Sergio Aguilera Ramirez A2estudiante1@atcgrid:~] 2018-03-13 martes
ls -lag
total 80
drwx------    5 A2estudiante1  4096 mar 13 10:33 .
drwxr-xr-x. 504 root          20480 feb 19 12:43 ..
-rw-------    1 A2estudiante1  5886 mar 13 10:20 .bash_history
-rw-r--r--    1 A2estudiante1    18 ene 16  2015 .bash_logout
-rw-r--r--    1 A2estudiante1   193 ene 16  2015 .bash_profile
-rw-r--r--    1 A2estudiante1   231 ene 16  2015 .bashrc
drwxr-xr-x    3 A2estudiante1  4096 feb 26  2016 .local
drwxr-xr-x    4 A2estudiante1  4096 ene 30  2015 .mozilla
drwxr-xr-x    2 A2estudiante1  4096 feb  5  2015 .ssh
-rw-------    1 A2estudiante1    42 mar 13 10:33 STDIN.e66403
-rw-------    1 A2estudiante1   202 mar 13 10:33 STDIN.o66403
-rwxr-xr-x    1 A2estudiante1  8968 mar 13 10:33 SumaVectoresC
-rw-------    1 A2estudiante1    60 feb 26  2016 .Xauthority
[Sergio Aguilera Ramirez A2estudiante1@atcgrid:~] 2018-03-13 martes
cat STDIN.o66403
Tiempo(seg.):0.056581516         / 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) /
[Sergio Aguilera Ramirez A2estudiante1@atcgrid:~] 2018-03-13 martes
cat STDIN.e66403

real    0m0.165s
user    0m0.056s
sys     0m0.105s
```

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:

```
[Sergio Aguilera Ramirez A2estudiante1@atcgrid:~] 2018-03-20 martes
$echo "./SumaVectoresC 10" | qsub -q ac
67321.atcgrid
[Sergio Aguilera Ramirez A2estudiante1@atcgrid:~] 2018-03-20 martes
$cat STDIN.o67321
Tiempo(seg.):0.000002813         / Tamaño Vectores:10    / V1[0]+V2[0]=V3[0](1.000000+1.000
000=2.000000) / / V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /
[Sergio Aguilera Ramirez A2estudiante1@atcgrid:~] 2018-03-20 martes
$echo "./SumaVectoresC 10000000" | qsub -q ac
67322.atcgrid
[Sergio Aguilera Ramirez A2estudiante1@atcgrid:~] 2018-03-20 martes
$cat STDIN.o67322
Tiempo(seg.):0.055446491         / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](100000
0.000000+1000000.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3[9999999](1999999.90
0000+0.100000=2000000.000000) /
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Tamaño : 10 Tiempo: 0.000002813 segundo NI: 6 * 10 +3 = 63 FPO: 3 * 10 = 30 MIPS = 63 / (0.000002813 * 10^6) = 22.396 MFLOPS = 30 / (0.000002813 * 10^6) = 10.664	Tamaño: 10000000 Tiempo: 0.055446491 segundos NI: 6 * 10000000 + 3 = 60000003 FPO: 3 * 10000000 = 30000000 MIPS = 60000003 / (0.05544691 * 10^6) = 1082.1162 MFLOPS = 30000000 / (0.05544691 * 10^6) = 541.05

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

```
call            clock_gettime
                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
                call            clock_gettime
```

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
                //Inicializar vectores
                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
                }


        //Calcular suma de vectores
        #pragma omp single
        {
                cgt1 = omp_get_wtime();
        }


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


        //Calcular suma de vectores
        #pragma omp single
        {
                cgt2 = omp_get_wtime();
        }
}

        ncgt = cgt2 - cgt1;
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[Sergio Aguilera Ramirez sergioaguilera@ei143091:~/Escritorio/P1AC] 2018-03-20 martes
$gcc -O2 -w -fopenmp Listado1\(Modificado\).c -o Listado1\(Modificado\) -lrt
[Sergio Aguilera Ramirez sergioaguilera@ei143091:~/Escritorio/P1AC] 2018-03-20 martes
$./Listado1\(Modificado\) 8
Tiempo(seg.):0.000001941        / Tamaño Vectores:8    / V1[0]+V2[0]=V3[0](0.800000+%  8.
6f=0.800000) / / V1[7]+V2[7]=V3[7](1.600000+1.500000=0.100000) /
[Sergio Aguilera Ramirez sergioaguilera@ei143091:~/Escritorio/P1AC] 2018-03-20 martes
$./Listado1\(Modificado\) 11
Tiempo(seg.):0.000002138        / Tamaño Vectores:11   / V1[0]+V2[0]=V3[0](1.100000+%  8.
6f=1.100000) / / V1[10]+V2[10]=V3[10](2.200000+2.100000=0.100000) /
```

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

```
#pragma omp parallel
{

               #pragma omp sections
               {

               #pragma omp section
               //Inicializar vectores
               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
               //Inicializar vectores
               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
               //Inicializar vectores
               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
               //Inicializar vectores
               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
                }

               }

               //Calcular suma de vectores
               #pragma omp single
               {
               cgt1 = omp_get_wtime();
               }



               #pragma omp sections
               {

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

```
                #pragma omp section
                for(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];
                }
                }

                //Calcular suma de vectores
                #pragma omp single
                {
                cgt2 = omp_get_wtime();
                }
                }

                ncgt = cgt2 - cgt1;
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[Sergio Aguilera Ramirez sergioaguilera@ei143091:~/Escritorio/P1AC] 2018-03-20 martes
$gcc -O2 -w -fopenmp Listado1\(Ejercicio8\).c -o Listado1\(Ejercicio8\) -lrt
[Sergio Aguilera Ramirez sergioaguilera@ei143091:~/Escritorio/P1AC] 2018-03-20 martes
$./Listado1\(Ejercicio8\) 8
Tiempo(seg.):0.000002807        / Tamaño Vectores:8    / V1[0]+V2[0]=V3[0](0.800000+%  8.
6f=0.800000) / / V1[7]+V2[7]=V3[7](1.600000+1.500000=0.100000) /
[Sergio Aguilera Ramirez sergioaguilera@ei143091:~/Escritorio/P1AC] 2018-03-20 martes
$./Listado1\(Ejercicio8\) 11
Tiempo(seg.):0.000003562        / Tamaño Vectores:11   / V1[0]+V2[0]=V3[0](1.100000+%  8.
6f=1.100000) / / V1[10]+V2[10]=V3[10](2.200000+2.100000=0.100000) /
```

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 las dos implementaciones se utilizaran los cores y threads de lo que disponga la maquina con la que se haya realizado la práctica, aunque en el ejercicio 8 al declarar diferentes secciones puede que algunos threads no realicen ninguna tarea.

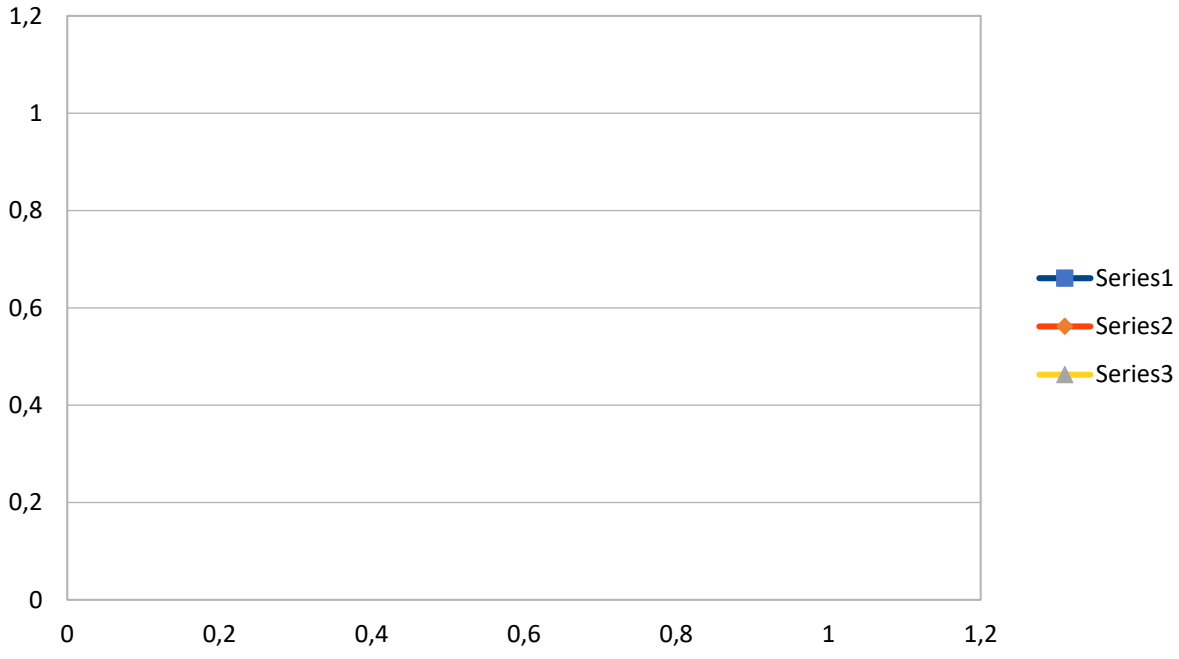
10. Rellenar una tabla como la Tabla 214 para atcgrid y otra para su PC con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución del trozo 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:

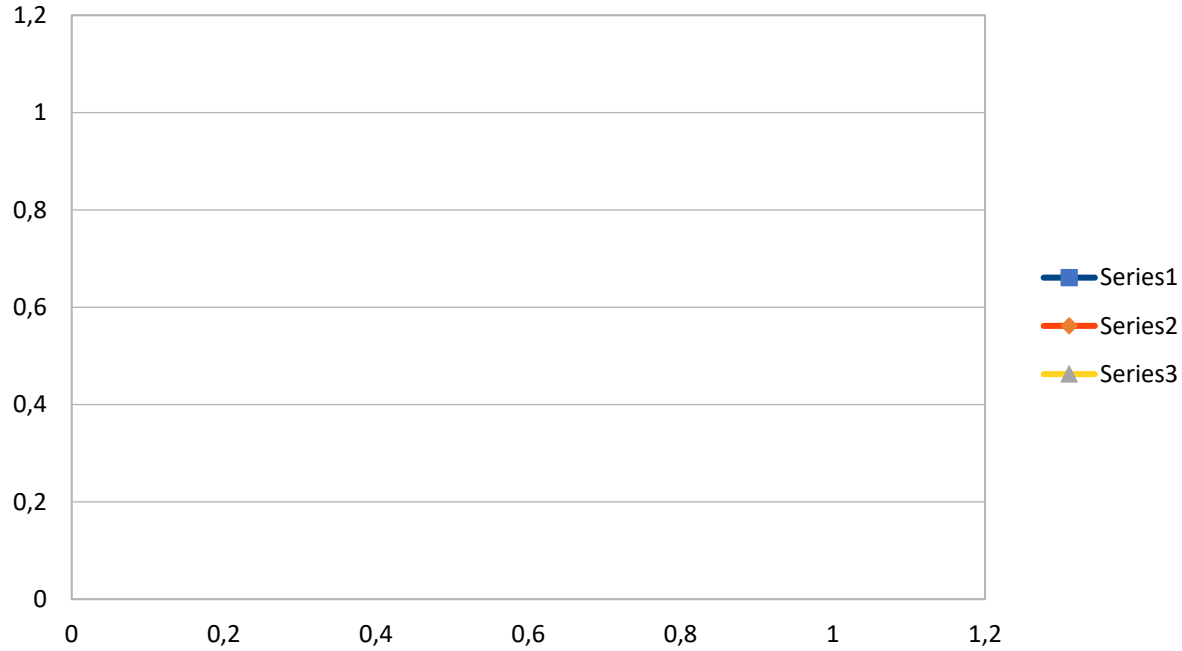
PC: 2.8 GHz Intel Core i7

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión `for`) 2 threads/ 2 cores	T. paralelo (versión `sections`) 2 threads/ 2 cores
16384	0,000275628	0,000023470	0,000026005
32768	0,000267186	0,000045335	0,000046730
65536	0,000328124	0,000080556	0,000083549
131072	0,000483976	0,000161297	0,000162812
262144	0,001328469	0,000496860	0,000523995
524288	0,002328453	0,002333061	0,002364893
1048576	0,004159411	0,004424160	0,005694990
2097152	0,007783988	0,008783495	0,009002443
4194304	0,014762724	0,017695080	0,017847847
8388608	0,027718588	0,035658652	0,035494518
16777216	0,053548159	0,070648407	0,071175361
33554432	0,106177715	0,142107310	0,141833617
67108864	0,106647564	0,282558657	0,283142347



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	0,000109435	0,003912625	0,003867927
32768	0,000141408	0,004183958	0,004301603
65536	0,000455180	0,004310786	0,003972797
131072	0,000924816	0,004531651	0,004299076
262144	0,001756158	0,004081232	0,004380108
524288	0,003416003	0,003738488	0,003303340
1048576	0,006316947	0,004906637	0,002988456
2097152	0,012704751	0,006572047	0,008157278
4194304	0,024687156	0,009852659	0,012945012
8388608	0,048833410	0,013558954	0,023506668
16777216	0,091855658	0,021913328	0,041483804
33554432	0,185198587	0,039551778	0,074346623
67108864	0,197697158	0,077460940	0,143498559



11. Rellenar una tabla como la 13 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 CPU obtenido en el programa secuencial es el mismo que el real ya que en este programa solo se utiliza un solo procesador. En el programa en paralelo el tiempo CPU es mayor que el tiempo real esto se debe a que el tiempo CPU es la suma del tiempo de cada core y el tiempo real es el tiempo tardado por la maquina en ejecutar el programa.

Nº de Componentes	**Tiempo secuencial vect. Globales 1 thread/core**			**Tiempo paralelo/versión `for` ¿? Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	Real 0m0.005s user 0m0.002s sys 0m0.002s			Real 0m0.244s user 0m0.002s sys 0m0.005s		
131072	Real 0m0.006s user 0m0.002s sys 0m0.003s			Real 0m0.007s user 0m0.003s sys 0m0.004s		

262144	Real 0m0.009s user 0m0.003s sys 0m0.004s	Real 0m0.009s user 0m0.005s sys 0m0.007s
524288	Real 0m0.016s user 0m0.006s sys 0m0.007s	Real 0m0.016s user 0m0.008s sys 0m0.014s
1048576	Real 0m0.033s user 0m0.012s sys 0m0.019s	Real 0m0.031s user 0m0.014s sys 0m0.023s
2097152	Real 0m0.053s user 0m0.021s sys 0m0.025s	Real 0m0.074s user 0m0.026s sys 0m0.040s
4194304	Real 0m0.104s user 0m0.043s sys 0m0.057s	Real 0m0.110s user 0m0.048s sys 0m0.083s
8388608	Real 0m0.287s user 0m0.076s sys 0m0.090s	Real 0m0.188s user 0m0.097s sys 0m0.162s
16777216	Real 0m0.451s user 0m0.181s sys 0m0.225s	Real 0m0.473s user 0m0.205s sys 0m0.442s
33554432	Real 0m0.794s user 0m0.310s sys 0m0.413s	Real 0m1.366 user 0m0.473s sys 0m1.138s
67108864	Real 0m0.771s user 0m0.302s sys 0m0.422s	Real 0m8.306s user 0m1.002s sys 0m3.107s

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` ¿? Threads/cores		
	Elapsed	*CPU-user*	*CPU- sys*	*Elapsed*	*CPU-user*	*CPU- sys*
65536						
131072						
262144						
524288						
1048576						
2097152						
4194304						
8388608						
16777216						
33554432						
67108864						

2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones pa**Tabla**ralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utiliza dos, que debe coincidir con el número de cores físicos utilizados.