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): José Manuel Pérez Lendínez
Grupo de prácticas:A2
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());
}
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`

```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
    {
      printf("Resultados mostrados por el thread %d\n",omp_get_thread_num());
      for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
      printf("\n");
    }
  }

  return(0);
}
```

**CAPTURAS DE PANTALLA:**

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica1/codigos] 2018-03-22 Thursday
$./single.eje
Introduce valor de inicialización a: 14
Single ejecutada por el thread 1
Resultados mostrados 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
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica1/codigos] 2018-03-22 Thursday
```

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica1/codigos] 2018-03-22 Thursday
$./single.eje
Introduce valor de inicialización a: 14
Single ejecutada por el thread 0
Resultados mostrados por el thread 3
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
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica1/codigos] 2018-03-22 Thursday
```

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`

```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
    {
      printf("Resultados mostrados por el thread %d\n",omp_get_thread_num());
      for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
      printf("\n");
    }
  }

  return(0);
}
```

**CAPTURAS DE PANTALLA:**

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica1/codigos] 2018-03-22 Thursday
$./single2.eje
Introduce valor de inicialización a: 13
Single ejecutada por el thread 0
Resultados mostrados por el thread 0
b[0] = 13       b[1] = 13       b[2] = 13       b[3] = 13       b[4] = 13       b[5] = 13       b[6] = 13       b[7] = 13       b[8] = 13
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica1/codigos] 2018-03-22 Thursday
$
```

**RESPUESTA A LA PREGUNTA:**

La diferencia es que en este caso siempre sera ejecutada esa parte por la hebra 0 que es la master, mientras en el caso anterior podía ser ejecutada por cualquiera.

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:**

   La directiva barrier obliga a que todas las hebras lleguen a ese punto antes de poder avanzar. En este caso si no se pone la directiva barrier podrí a darse el caso en el que la hebra master termine la suma, continué hacia adelante y muestre el resultado antes de que todas las hebras realizaran su suma.

---

## 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:**

```
$time ./SumaVectoresC.eje 10000000
Tiempo(seg.):0.041945000        Tamaño Vectores:10000000        V1[0]+V2[0]=V
3[0](1000000.000000+1000000.000000=2000000.000000) V1[9999999]+V2[9999999]=V3[
9999999](1999999.900000+0.100000=2000000.000000)

real    0m0.157s
user    0m0.109s
sys     0m0.016s
```

   T. de cpu = user + sys , no coincide con el tiempo real porque en el tiempo de cpu no esta incluido el tiempo que el programa puede estar sin ejecutar debido a los dispositivos de entrada/salida por ejemplo.

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:**

```
[jose manuel perez lendinez A2estudiante18@atcgrid:~] 2018-04-02 Monday
$echo './SumaVectoresC.eje 10' |qsub -q ac
71059.atcgrid
[jose manuel perez lendinez A2estudiante18@atcgrid:~] 2018-04-02 Monday
$ls
STDIN.e71055  STDIN.e71059  STDIN.o71056  SumaVectoresC.eje
STDIN.e71056  STDIN.o71055  STDIN.o71059  SumaVectoresCparalela.eje
[jose manuel perez lendinez A2estudiante18@atcgrid:~] 2018-04-02 Monday
$cat STDIN.o71059
Tiempo(seg.):0.000002213        Tamaño Vectores:10      V1[0]+V2[0]=V3[0](1.0
00000+1.000000=2.000000) V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000)
[jose manuel perez lendinez A2estudiante18@atcgrid:~] 2018-04-02 Monday
$
```

```
[jose manuel perez lendinez A2estudiante18@atcgrid:~] 2018-04-02 Monday
$echo './SumaVectoresC.eje 10000000' |qsub -q ac
71060.atcgrid
[jose manuel perez lendinez A2estudiante18@atcgrid:~] 2018-04-02 Monday
$cat STDIN.o71060
Tiempo(seg.):0.067332370         Tamaño Vectores:10000000        V1[0]+V2[0]=V
3[0](1000000.000000+1000000.000000=2000000.000000) V1[9999999]+V2[9999999]=V3[
9999999](1999999.900000+0.100000=2000000.000000)
[jose manuel perez lendinez A2estudiante18@atcgrid:~] 2018-04-02 Monday
$
```

**RESPUESTA:** cálculo de los MIPS y los MFLOPS

| | Tamaño 10 | Tamaño 10000000 |
|---|---|---|
| Nº de instrucciones | 6*10+3 = 63 | 6*10000000+3=60000003 |
| Tiempo(seg) | 0,000002213 | 0,067332370 |
| Nº de instrucciones de coma flotante | 3*10 = 30 | 3*10000000= 30000000 |
| MIPS | 63/(0,000002213*10^6) = **28,468** | 60000003/ (0,067332370*10^6)=**891,102** |
| MFLOPS | 30/(0,000002213*10^6) = **13,556** | 30000000/ (0,067332370*10^6)=**445,551** |

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

```
71    xorl  %eax, %eax
72    .p2align 4,,10
73    .p2align 3
74  .L5:
75    movsd v1(%rax), %xmm0
76    addq  $8, %rax
77    addsd v2-8(%rax), %xmm0
78    movsd %xmm0, v3-8(%rax)
79    cmpq  %rax, %rbx
80    jne .L5
81  .L6:
82    leaq  16(%rsp), %rsi
83    xorl  %edi, %edi
84    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

```
/* SumaVectoresC.c
Suma de dos vectores: v3 = v1 + v2
Para compilar usar (-lrt: real time library):
gcc -O2 SumaVectores.c -o SumaVectores -lrt
gcc -O2 –S SumaVectores.c –lrt //para generar el código ensamblador
Para ejecutar use: SumaVectoresC longitud
*/
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
#ifdef _OPENMP
              #include <omp.h>
#else
              #define omp_get_thread_num() 0
              #define omp_get_num_threads() 1
#endif


//#define PRINTF_ALL// comentar para quitar el printf ...
 // que imprime todos los componentes
//Sólo puede estar definida una de las tres constantes VECTOR_ (sólo uno de
los ...
//tres defines siguientes puede estar descomentado):
//#define VECTOR_LOCAL // descomentar para que los vectores sean variables ...
 // locales (si se supera el tamaño de la pila se ...
 // generará el error "Violación de Segmento")
#define VECTOR_GLOBAL// descomentar para que los vectores sean variables ...
 // globales (su longitud no estará limitada por el ...
 // tamaño de la pila del programa)
//#define VECTOR_DYNAMIC // descomentar para que los vectores sean
variables ...
 // dinámicas (memoria reutilizable durante la ejecución)
#ifdef VECTOR_GLOBAL
#define MAX 33554432 //=2^25
double v1[MAX], v2[MAX], v3[MAX];
#endif


int main(int argc, char** argv)
{
              int i;
              double cgt1,cgt2;
              double ncgt; //para tiempo de ejecución
              //Leer argumento de entrada (n° de componentes del vector)
              if (argc<2)
              {
                            printf("Faltan n° componentes del vector\n");
                            exit(-1);
              }


              unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)
#ifdef VECTOR_LOCAL
              double v1[N], v2[N], v3[N]; // Tamaño variable local en tiempo
de ejecución ...
              // disponible en C a partir de actualización C99
#endif
#ifdef VECTOR_GLOBAL
              if (N>MAX) N=MAX;
#endif
#ifdef VECTOR_DYNAMIC
```

```
            double *v1, *v2, *v3;
            v1 = (double*) malloc(N*sizeof(double));// malloc necesita el
tamaño en bytes
            v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio
suficiente malloc devuelve NULL
            v3 = (double*) malloc(N*sizeof(double));
            if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
            printf("Error en la reserva de espacio para los vectores\n");
            exit(-2);
            }
            #endif
            //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
            if(N <= 11){
                        printf("Tiempo(seg.):%11.9f\t  Tamaño Vectores:
%u\n",ngct,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
#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
#endif
            return 0;
}
```

**(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

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

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica1/codigo
s] 2018-04-02 Monday
$./SumaVectoresCejer7.eje 11
Tiempo(seg.):0.000013000          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
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica1/codigo
s] 2018-04-02 Monday
$./SumaVectoresCejer7.eje 8
Tiempo(seg.):0.000012000          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
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica1/codigo
s] 2018-04-02 Monday
$
```

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 private(i)
{
  #pragma omp sections
  {
    #pragma omp section
    for(i=0; i<N/4; i++){
      v1[i] = N*0.1+i*0.1;v2[i] = N*0.1-i*0.1;
    }

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

    #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;
    }

    #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;
    }
  }

  #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];

  }

  #pragma omp single
  {
    cgt2 = omp_get_wtime();
  }
}
```

**(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

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

```
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica1/codigo
s] 2018-04-03 Tuesday
$./SumaVectoresCejer7.eje 8
Tiempo(seg.):0.000014000          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
[jose manuel perez lendinez jmplz14@DESKTOP-KGK4SOA:/mnt/d/AC/practica1/codigo
s] 2018-04-03 Tuesday
$./SumaVectoresCejer7.eje 11
Tiempo(seg.):0.000012000          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.

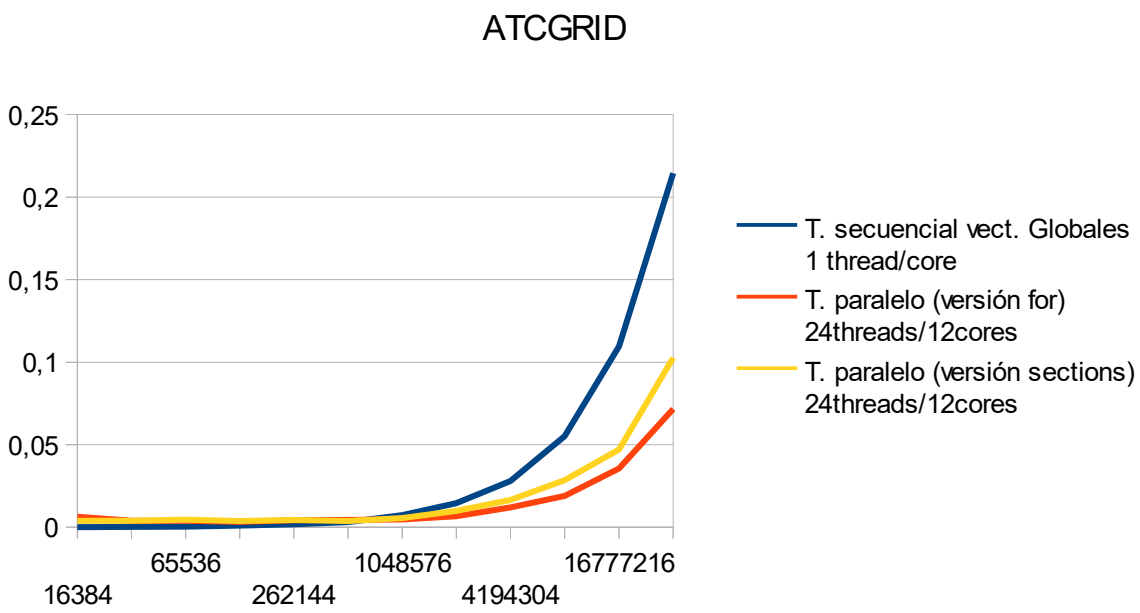
**RESPUESTA:**

Como mi ordenador tiene 4 núcleos con un solo hilo, lo he realizado de forma que se tenga 4 secciones en el vector en el ejercicio 8. Aun con esto se podría ejecutar con cualquier numero de cores lógicos o físicos puesto que la propia directiva sección limita a que solo lo ejecuten como máximo 4, si se tiene menos coresalguno ejecutar mas de una sección.

En el ejercicio 7 se puede ejecutar con cualquier numeor de cores logicos o fisicos. La propia directiva for reparte entre las hebras las partes a ejecutar.

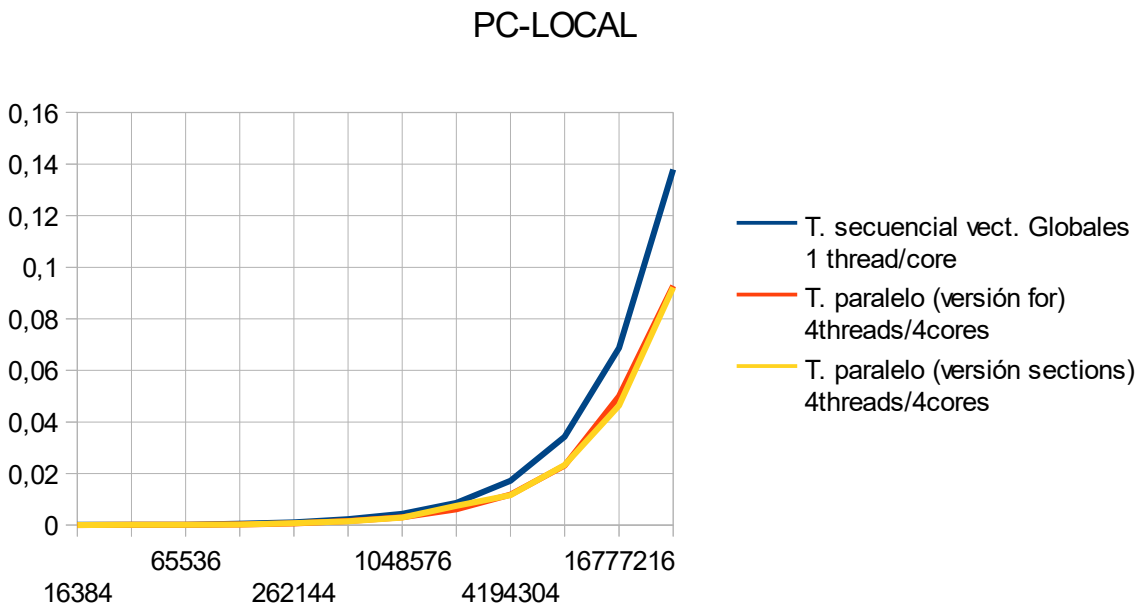
10. Rellenar una tabla como la Tabla 2 para atcgrid y otra para su PC con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución del trozo de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

**RESPUESTA:**



**ATCGRID**

| Nº de Componentes | T. secuencial vect. Globales 1 thread/core | T. paralelo (versión `for`) 24threads/12cores | T. paralelo (versión `sections`) 24threads/12cores |
|---|---|---|---|
| **16384** | 0,000071898 | 0,006336311 | 0,003899575 |
| **32768** | 0,000213378 | 0,003900629 | 0,003869961 |
| **65536** | 0,000448517 | 0,003924091 | 0,004538864 |
| **131072** | 0,000948987 | 0,003207621 | 0,003839629 |
| **262144** | 0,001830145 | 0,004210751 | 0,004356352 |
| **524288** | 0,003082300 | 0,004373626 | 0,003738595 |
| **1048576** | 0,007281402 | 0,004737129 | 0,005466596 |
| **2097152** | 0,014558969 | 0,006687012 | 0,009899840 |
| **4194304** | 0,027932363 | 0,011929186 | 0,016523199 |
| **8388608** | 0,055043281 | 0,018979694 | 0,028521501 |
| **16777216** | 0,109547357 | 0,035588778 | 0,047127643 |
| **33554432** | 0,214453949 | 0,071696632 | 0,102572302 |



PC LOCAL

| **Nº de Componentes** | **T. secuencial vect. Globales** **1 thread/core** | **T. paralelo (versión `for`) 4threads/4cores** | **T. paralelo (versión `sections`) 4threads/4cores** |
|---|---|---|---|
| **16384** | 0,000091500 | 0,000050000 | 0,000068000 |
| **32768** | 0,000154600 | 0,000091000 | 0,000090000 |
| **65536** | 0,000154600 | 0,000168000 | 0,000161000 |
| **131072** | 0,000556000 | 0,000255000 | 0,000258000 |
| **262144** | 0,001080400 | 0,000679000 | 0,000705000 |
| **524288** | 0,002304500 | 0,001560000 | 0,001506000 |
| **1048576** | 0,004353800 | 0,002968000 | 0,002944000 |
| **2097152** | 0,008629800 | 0,006119000 | 0,007445000 |
| **4194304** | 0,017214600 | 0,011772000 | 0,011623000 |
| **8388608** | 0,034284500 | 0,023109000 | 0,023316000 |
| **16777216** | 0,068755100 | 0,049997000 | 0,046394000 |
| **33554432** | 0,137860900 | 0,092819000 | 0,092164000 |

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**:En el caso secuencial el tiempo de cpu y el tiempo real coinciden porque solo se ejecuta en un core. En la versión con hebras estas se están ejecutando en paralelo en varios cores distintos. Esto hace que el tiempo real sea menor que el tiempo de cpu. Esto es porque las hebras suman sus tiempos de cpu pero el tiempo real solo se tiene en cuenta hasta que la ultima hebra termina.

| **Nº de Componentes** | **Tiempo secuencial vect. Globales 1 thread/core** | | | **Tiempo paralelo/versión `for` 24 Threads/12cores** | | |
|---|---|---|---|---|---|---|
| | ***Elapsed*** | ***CPU-user*** | ***CPU- sys*** | ***Elapsed*** | ***CPU-user*** | ***CPU- sys*** |
| **65536** | real 0m0.004s | | | real 0m0.012s | | |
| | user 0m0.000s | | | user 0m0.147s | | |
| | sys 0m0.003s | | | sys 0m0.021s | | |
| **131072** | real 0m0.004s | | | real 0m0.010s | | |
| | user 0m0.000s | | | user 0m0.177s | | |
| | sys 0m0.003s | | | sys 0m0.002s | | |
| **262144** | real 0m0.006s | | | real 0m0.011s | | |
| | user 0m0.002s | | | user 0m0.155s | | |
| | sys 0m0.004s | | | sys 0m0.034s | | |
| **524288** | real 0m0.010s | | | real 0m0.015s | | |
| | user 0m0.002s | | | user 0m0.193s | | |
| | sys 0m0.008s | | | sys 0m0.005s | | |
| **1048576** | real 0m0.020s | | | real 0m0.014s | | |
| | user 0m0.009s | | | user 0m0.191s | | |
| | sys 0m0.010s | | | sys 0m0.026s | | |
| **2097152** | real 0m0.040s | | | real 0m0.023s | | |
| | user 0m0.012s | | | user 0m0.221s | | |
| | sys 0m0.027s | | | sys 0m0.082s | | |
| **4194304** | real 0m0.078s | | | real 0m0.031s | | |
| | user 0m0.027s | | | user 0m0.289s | | |
| | sys 0m0.049s | | | sys 0m0.174s | | |
| **8388608** | real 0m0.162s | | | real 0m0.057s | | |
| | user 0m0.057s | | | user 0m0.462s | | |
| | sys 0m0.102s | | | sys 0m0.283s | | |
| **16777216** | real 0m0.303s | | | real 0m0.092s | | |
| | user 0m0.113s | | | user 0m0.673s | | |
| | sys 0m0.187s | | | sys 0m0.558s | | |
| **33554432** | real 0m0.604s | | | real 0m0.163s | | |
| | user 0m0.223s | | | user 0m1.229s | | |
| | sys 0m0.377s | | | sys 0m1.270s | | |