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("Depués de la región parallel (ejecutado 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");

        }

    }

}
```

**CAPTURAS DE PANTALLA:**

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MacBook-Pro-de-Alejandro:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de
computadores/AC---Practicas/Practica 1] 2018-03-06 martes
$./singleModificadoMac
Introduce valor de inicialización a: 3
Single ejecutada por el thread 1
Depués de la región parallel (ejecutado por el thread 3):
b[0] = 3        b[1] = 3        b[2] = 3        b[3] = 3        b[4] = 3        b[5] = 3        b[6] = 3        b[7] = 3
 b[8] = 3
```

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>

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("Depués de la región parallel (ejecutado 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");

        }

    }

}
```

**CAPTURAS DE PANTALLA:**

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MacBook-Pro-de-Alejandro:~/Dropbox/Carrera/2ºCuatrimestre/Arquitectura de
 computadores/AC---Practicas/Practica 1] 2018-03-06 martes
$./singleModificado2Mac
Introduce valor de inicialización a: 2
Single ejecutada por el thread 1
Depués de la región parallel (ejecutado por el thread 0):
b[0] = 2        b[1] = 2        b[2] = 2        b[3] = 2        b[4] = 2        b[5] = 2        b[6] = 2        b[7] = 2
b[8] = 2
```

**RESPUESTA A LA PREGUNTA:**

La diferencia está en que con master siempre ejecutará la instrucción el thread 0, el thread master.

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:**

Porque la directiva `barrier` crea una barrera implicita en el programa donde todas las hebras deben esperar a que lleguen las demas. Cuando han llegado todas a la barrera, el programa continua. Es por esto, que si quitamos la directiva `barrier`, la hebra master imprimirá el resultado que la variable `suma` contenga en ese momento.

## 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:**

```
[AlejandroGarciaVallecillo D1estudiante8@atcgrid:~/P1] 2018-03-12 lunes
$time ./SumaVectoresC 10000000
Tiempo(seg.):0.205837546        / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.00
0000) V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /

real    0m0,569s
user    0m0,287s
sys     0m0,225s
```

   La suma del tiempo de usuario y el tiempo del sistema es 0,512 y el tiempo real es 0,569, por lo que la suma es menor. Esto significa que la diferencia, 0,057, es tiempo que el sistema consume en esperas debido a I/O o asociados a la ejecución de otros programas.

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:**

```
[AlejandroGarciaVallecillo D1estudiante8@atcgrid:~/P1] 2018-03-12 lunes
$time ./SumaVectoresC 10000000
Tiempo(seg.):0.205837546        / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.00
0000) V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /

real    0m0,569s
user    0m0,287s
sys     0m0,225s
[AlejandroGarciaVallecillo D1estudiante8@atcgrid:~/P1] 2018-03-12 lunes
$time ./SumaVectoresC 10
Tiempo(seg.):0.000006705        / Tamaño Vectores:10   / V1[0]+V2[0]=V3[0](1.000000+1.000000=2.000000) V1[9]+V2[9]=V3[9](1.
900000+0.100000=2.000000) /

real    0m0,002s
user    0m0,001s
sys     0m0,001s
```

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

   10 componentes: Teniendo en cuenta que el bucle for consta de 65 instrucciones para 10 componentes:

   MIPS = NI / (Tc * 10^6) = 65 / (0.000006705 * 10^6) = 9,694 MIPS

   MFLOPS = NFLO / (Tc * 10^6) = 10 / (0.000006705 * 10^6) = 1,491 MFLOPS

   10000000 componentes: Teniendo en cuenta que el bucle for consta de 65 instrucciones para 10000000 componentes:

   MIPS = NI / (Tc * 10^6) = 60000005 / (0.205837546 * 10^6) = 291,492 MIPS

   MFLOPS = NFLO / (Tc * 10^6) = 10000000 / (0.205837546 * 10^6) = 48,582 MFLOPS

**RESPUESTA:** Captura que muestre el 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:** Captura que muestre el código fuente implementado

```
//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
    //printf("thread %d ejecuta la iteración %d del bucle\n", omp_get_thread_num(),i);
  }

double start = omp_get_wtime();

//Calcular suma de vectores
#pragma omp parallel for
  for(i=0; i<N; i++) {
    v3[i] = v1[i] + v2[i];
    printf("thread %d ejecuta la iteración %d del bucle\n", omp_get_thread_num(),i);
  }

double end = omp_get_wtime( );

tTotal = end - start;

//Imprimir resultado de la suma y el tiempo de ejecución
#ifdef PRINTF_ALL
printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",tTotal,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]);
```

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

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

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@cvi040083:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC
---Practicas/Practica 1] 2018-03-16 viernes
$gcc-7 -O2 -fopenmp -o SumaVectoresOMP SumaVectoresOMP.c
```

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@cvi040083:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC
---Practicas/Practica 1] 2018-03-16 viernes
$./SumaVectoresOMP 8
thread 3 ejecuta la iteración 6 del bucle
thread 1 ejecuta la iteración 2 del bucle
thread 0 ejecuta la iteración 0 del bucle
thread 2 ejecuta la iteración 4 del bucle
thread 3 ejecuta la iteración 7 del bucle
thread 1 ejecuta la iteración 3 del bucle
thread 0 ejecuta la iteración 1 del bucle
thread 2 ejecuta la iteración 5 del bucle
Tiempo(seg.):0.000580000         / 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) /
```

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@cvi040083:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de computadores/AC
---Practicas/Practica 1] 2018-03-16 viernes
$./SumaVectoresOMP 11
thread 2 ejecuta la iteración 6 del bucle
thread 1 ejecuta la iteración 3 del bucle
thread 3 ejecuta la iteración 9 del bucle
thread 0 ejecuta la iteración 0 del bucle
thread 2 ejecuta la iteración 7 del bucle
thread 1 ejecuta la iteración 4 del bucle
thread 3 ejecuta la iteración 10 del bucle
thread 0 ejecuta la iteración 1 del bucle
thread 2 ejecuta la iteración 8 del bucle
thread 1 ejecuta la iteración 5 del bucle
thread 0 ejecuta la iteración 2 del bucle
Tiempo(seg.):0.000217000         / 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:** Captura que muestre el código fuente implementado

```
void iniVectores(unsigned int N){

    int i;
     //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
        printf("thread %d ejecuta la inicializacion %d de los vectores\n", omp_get_thread_num(),i);
    }

}

void sumarVectores(unsigned int N){

    int i;
    for(i=0; i<N; i++) {
        v3[i] = v1[i] + v2[i];
        printf("thread %d ejecuta la iteración %d del bucle\n", omp_get_thread_num(),i);
    }

}
```

```
#pragma omp parallel sections
{

  #pragma omp section
      (void) iniVectores(N);

  #pragma omp section
  {
      start = omp_get_wtime();
      (void) sumarVectores(N);
      end = omp_get_wtime( );
  }

}
```

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

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

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MacBook-Pro-de-Alejandro:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de
computadores/AC---Practicas/Practica 1] 2018-03-16 viernes
$gcc-7 -O2 -fopenmp -o SumaVectoresOMP2 SumaVectoresOMP2.c
```

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MacBook-Pro-de-Alejandro:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de
computadores/AC---Practicas/Practica 1] 2018-03-16 viernes
$./SumaVectoresOMP2 8
thread 2 ejecuta la inicializacion 0 de los vectores
thread 0 ejecuta la iteración 0 del bucle
thread 2 ejecuta la inicializacion 1 de los vectores
thread 0 ejecuta la iteración 1 del bucle
thread 2 ejecuta la inicializacion 2 de los vectores
thread 0 ejecuta la iteración 2 del bucle
thread 2 ejecuta la inicializacion 3 de los vectores
thread 0 ejecuta la iteración 3 del bucle
thread 2 ejecuta la inicializacion 4 de los vectores
thread 0 ejecuta la iteración 4 del bucle
thread 2 ejecuta la inicializacion 5 de los vectores
thread 0 ejecuta la iteración 5 del bucle
thread 2 ejecuta la inicializacion 6 de los vectores
thread 0 ejecuta la iteración 6 del bucle
thread 2 ejecuta la inicializacion 7 de los vectores
thread 0 ejecuta la iteración 7 del bucle
Tiempo(seg.):0.000223000         / 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) /
```

```
[AlejandroGarciaVallecillo alejandrogarciavallecillo@MacBook-Pro-de-Alejandro:~/Dropbox/Carrera/2°Cuatrimestre/Arquitectura de
computadores/AC---Practicas/Practica 1] 2018-03-16 viernes
$./SumaVectoresOMP2 11
thread 0 ejecuta la inicializacion 0 de los vectores
thread 1 ejecuta la iteración 0 del bucle
thread 0 ejecuta la inicializacion 1 de los vectores
thread 1 ejecuta la iteración 1 del bucle
thread 0 ejecuta la inicializacion 2 de los vectores
thread 1 ejecuta la iteración 2 del bucle
thread 0 ejecuta la inicializacion 3 de los vectores
thread 1 ejecuta la iteración 3 del bucle
thread 0 ejecuta la inicializacion 4 de los vectores
thread 1 ejecuta la iteración 4 del bucle
thread 0 ejecuta la inicializacion 5 de los vectores
thread 1 ejecuta la iteración 5 del bucle
thread 0 ejecuta la inicializacion 6 de los vectores
thread 1 ejecuta la iteración 6 del bucle
thread 0 ejecuta la inicializacion 7 de los vectores
thread 1 ejecuta la iteración 7 del bucle
thread 0 ejecuta la inicializacion 8 de los vectores
thread 1 ejecuta la iteración 8 del bucle
thread 0 ejecuta la inicializacion 9 de los vectores
thread 1 ejecuta la iteración 9 del bucle
thread 0 ejecuta la inicializacion 10 de los vectores
thread 1 ejecuta la iteración 10 del bucle
Tiempo(seg.):0.000301000         / 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:**

   - N thread, ya que si utilizamos más thread que número de iteraciones del for, estas no se utilizarían.

   - 2 thread como máximo, ya que depende del número de sections y los creados a partir de los dos primeros no se utilizarían.

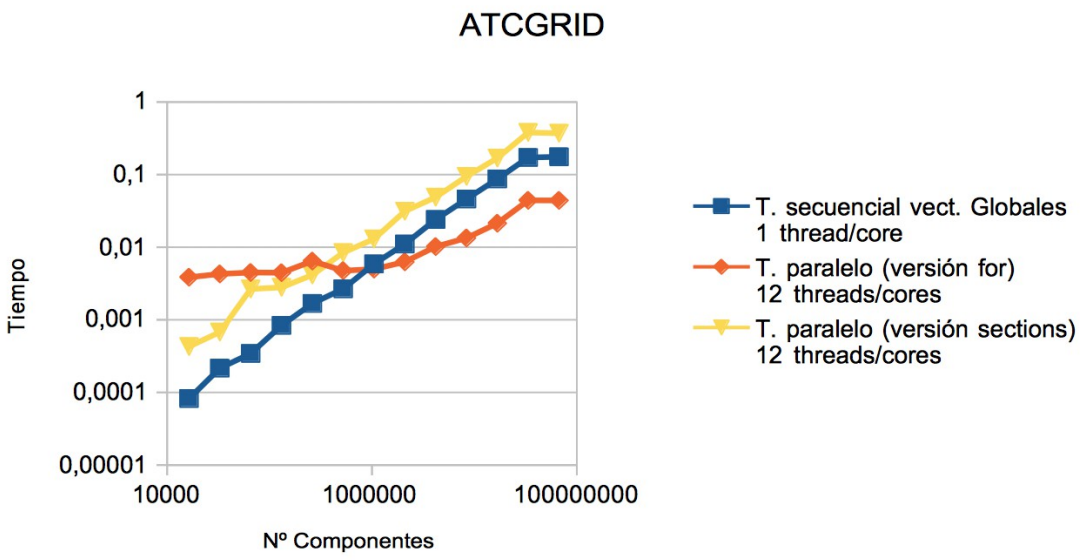
   El número de cores sería el número de thread/2.

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`) 12 threads/cores** | **T. paralelo (versión `sections`) 12 threads/cores** |
| **16384** | 0.000082145 | 0.003852291 | 0.000427592 |
| **32768** | 0.000216143 | 0.004282184 | 0.000678245 |
| **65536** | 0,000343706 | 0.004484751 | 0.002656266 |
| **131072** | 0,000837749 | 0.004447266 | 0.002799995 |
| **262144** | 0,001681404 | 0.006425226 | 0.004147946 |
| **524288** | 0,002668151 | 0.004745692 | 0.008422083 |
| **1048576** | 0,005886096 | 0.004992273 | 0.012987511 |
| **2097152** | 0,011046379 | 0.006277094 | 0.031282829 |
| **4194304** | 0,024098284 | 0.010185304 | 0.048830401 |
| **8388608** | 0,046148912 | 0.013386141 | 0.094306388 |
| **16777216** | 0,087200568 | 0.021371388 | 0.169097663 |
| **33554432** | 0,172027762 | 0.044158599 | 0.380108895 |
| **67108864** | 0,175857023 | 0.044098930 | 0.370404025 |

| | PC LOCAL | | |
|---|---|---|---|
| **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** | 0.000142000 | 0.000103000 | 0.000193000 |
| **32768** | 0.000252000 | 0.000155000 | 0.000427000 |
| **65536** | 0,000341 | 0.000271000 | 0.000787000 |
| **131072** | 0,000729 | 0.000458000 | 0.001542000 |
| **262144** | 0,001382 | 0.001081000 | 0.003117000 |
| **524288** | 0,002486 | 0.001683000 | 0.005681000 |
| **1048576** | 0,00461 | 0.003057000 | 0.011567000 |
| **2097152** | 0,008394 | 0.006182000 | 0.024186000 |
| **4194304** | 0,020582 | 0.013080000 | 0.041935000 |
| **8388608** | 0,033785 | 0.025108000 | 0.081782000 |
| **16777216** | 0,072742 | 0.043113000 | 0.157745000 |
| **33554432** | 0,15057 | 0.063498000 | 0.314937000 |
| **67108864** | 0,14756 | 0.060415000 | 0.303457000 |



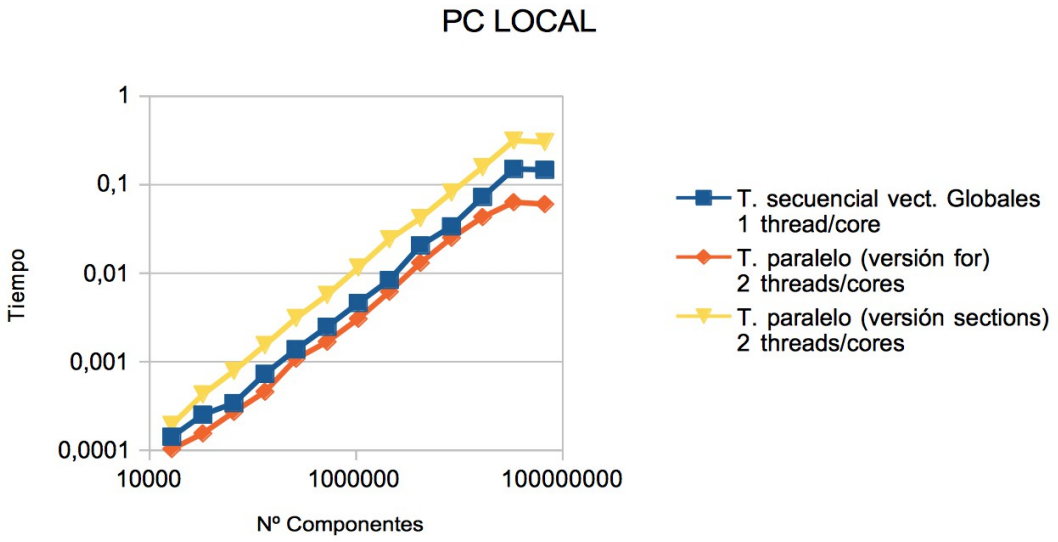



**Tabla 2.** Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados, que debe coincidir con el número de cores físicos utilizados.

| Nº de Componentes | T. secuencial vect. Globales 1 thread/core | T. paralelo (versión `for`) ¿?threads/cores | T. paralelo (versión `sections`) ¿?threads/cores |
|---|---|---|---|
| **16384** | | | |
| **32768** | | | |
| **65536** | | | |
| **131072** | | | |
| **262144** | | | |
| **524288** | | | |
| **1048576** | | | |
| **2097152** | | | |
| **4194304** | | | |
| **8388608** | | | |
| **16777216** | | | |
| **33554432** | | | |
| **67108864** | | | |

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**:

Normalmente es menor que el tiempo real, puesto que en éste último también influyen la interrupciones y demás del sistema.

| 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 | 0m0,026s | 0m0,002s | 0m0,003s | 0m0,023s | 0m0,002s | 0m0,005s |
| 131072 | 0m0,009s | 0m0,003s | 0m0,006s | 0m0,011s | 0m0,003s | 0m0,007s |
| 262144 | 0m0,016s | 0m0,007s | 0m0,008s | 0m0,018s | 0m0,008s | 0m0,008s |
| 524288 | 0m0,029s | 0m0,013s | 0m0,015s | 0m0,031s | 0m0,013s | 0m0,017s |
| 1048576 | 0m0,055s | 0m0,028s | 0m0,027s | 0m0,059s | 0m0,027s | 0m0,030s |
| 2097152 | 0m0,108s | 0m0,054s | 0m0,054s | 0m0,114s | 0m0,057s | 0m0,055s |
| 4194304 | 0m0,213s | 0m0,105s | 0m0,107s | 0m0,225s | 0m0,126s | 0m0,097s |
| 8388608 | 0m0,422s | 0m0,221s | 0m0,198s | 0m0,452s | 0m0,243s | 0m0,205s |
| 16777216 | 0m0,919s | 0m0,459s | 0m0,388s | 0m0,885s | 0m0,443s | 0m0,437s |
| 33554432 | 0m1,866s | 0m0,890s | 0m0,797s | 0m1,750s | 0m0,914s | 0m0,829s |
| 67108864 | 0m1,674s | 0m0,843s | 0m0,802s | 0m1,744s | 0m0,937s | 0m0,798s |

**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 | | | | | | |