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 y profesor de prácticas:
Fecha de entrega:
Fecha evaluación en clase:

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

Ejercicios basados en los ejemplos del seminario práctico

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

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char **argv) {

    int i, n;
    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();
    }
}
```

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("Impresión realizada 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:

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp1/ejer2] 2019-03-19 martes
$./singlemod
Introduce valor de inicialización a: 2
Single ejecutada por el thread 0
Impresión realizada por el thread 3
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
```

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("Impresión realizada 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:

```
$./singlemod2
Introduce valor de inicialización a: 2
Single ejecutada por el thread 2
Impresión realizada 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 entre master y single es que master no tiene barrera y siempre se ejecuta por la hebra master. La diferencia que se nota respecto a los resultados de ejecución con single es que la impresión siempre es realizada por el thread 0 (la hebra 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: La directiva atomic no tiene barrera implícita, por lo tanto la suma de la sumalocal de una hebra a suma puede no ser realizada por todas las hebras antes de que alguna que ya haya realizado el cálculo imprima el valor de suma como si fuera el valor final. La directiva barrier obliga a esperar a que todas las hebras hayan terminado la suma, evitando así este problema.

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:

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp1/ejer5] 2019-03-19 martes
$gcc -O2 SumaVectoresC.c -o sumavectoresglobales -lrt
[FranciscoJavierBolívarExpósito D1estudiante3@atcgrid:~/bp1/ejer5] 2019-03-19 martes
$echo 'time ./bp1/ejer5/sumavectoresglobales 10000000' | qsub -q ac
12271.atcgrid
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp1/ejer5] 2019-03-19 martes
$cat STDIN.e12271

real    0m0.119s
user    0m0.060s
sys     0m0.055s
```

   La suma de los tiempos de cpu del usuario y el sistema es menor que el tiempo real. Como es un programa secuencial y solo hay un flujo de control se cumple que el *elapsed time* tiene que ser mayor que el tiempo de cpu.

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

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

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp1/ejer6] 2019-03-26 martes
$gcc -S ../ejer5/SumaVectoresC.c -O2 -o SumaVectoresG_Assembly.s
```

```
[FranciscoJavierBolívarExpósito D1estudiante3@atcgrid:~/bp1/ejer6] 2019-03-26 martes
$echo 'time ./bp1/ejer5/sumavectoresglobales 10' | qsub -q ac
14172.atcgrid
[FranciscoJavierBolívarExpósito D1estudiante3@atcgrid:~/bp1/ejer6] 2019-03-26 martes
$echo 'time ./bp1/ejer5/sumavectoresglobales 10000000' | qsub -q ac
14174.atcgrid
```

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp1/ejer6] 2019-03-26 martes
$cat STDIN.e14172 STDIN.e14174

real    0m0.004s
user    0m0.001s
sys     0m0.001s

real    0m0.114s
user    0m0.065s
sys     0m0.046s
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Los valores se han obtenido mirando el numero de instrucciones necesarias en el código ensamblador y los resultados de los tiempos de ejecución.

MIPS = (6*N + 3) / Tcpu * 10^6

MFLOPS = N / Tcpu * 10^6

(Usando los datos para tamaño 10)

MIPS = 63 / 0.001 * 10^6 = 0,063MIPS

MFLOPS = 10 / 0.001* 10^6 = 0,01MFLOPS

(Usando los datos para tamaño 10000000)

MIPS = 60000003 / 0.065 * 10^6 = 923,076969MIPS

MFLOPS = 10000000 / 0.065 * 10^6 = 153,846153MFLOPS

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

```
        call    clock_gettime@PLT
        xorl    %eax, %eax
        .p2align 4,,10
        .p2align 3
.L6:
        movsd   (%r12,%rax,8), %xmm0
        addsd   0(%r13,%rax,8), %xmm0
        movsd   %xmm0, (%r14,%rax,8)
        addq    $1, %rax
        cmpl    %eax, %ebp
        ja      .L6
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
```

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

```
37    double cgt1,cgt2, ncgt; //para tiempo de ejecución
38
39    //Leer argumento de entrada (nº de componentes del vector)
40    if (argc<2){
41      printf("Faltan nº componentes del vector\n");
42      exit(-1);
43    }
44
45    unsigned int N = atoi(argv[1]);  // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
46    printf("Tamaño Vectores:%u (%u B)\n",N, sizeof(unsigned int));
47    #ifdef VECTOR_LOCAL
48    double v1[N], v2[N], v3[N];   // Tamaño variable local en tiempo de ejecución ...
49          // disponible en C a partir de C99
50    #endif
51    #ifdef VECTOR_GLOBAL
52    if (N>MAX) N=MAX;
53    #endif
54    #ifdef VECTOR_DYNAMIC
55    double *v1, *v2, *v3;
56    v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño en bytes
57    v2 = (double*) malloc(N*sizeof(double));
58    v3 = (double*) malloc(N*sizeof(double));
59    if ((v1 == NULL) || (v2 == NULL) || (v2 == NULL)) {
60      printf("No hay suficiente espacio para los vectores \n");
61      exit(-2);
62    }
63    #endif
64
65      //Inicializar vectores
66      #pragma omp parallel for
67        for(i=0; i<N; i++){
68          v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
69        }
70
71    cgt1 = omp_get_wtime();
72    //Calcular suma de vectores
73      #pragma omp parallel for
74        for(i=0; i<N; i++)
75          v3[i] = v1[i] + v2[i];
76
77    cgt2 = omp_get_wtime();
78    ncgt=cgt2 - cgt1;
```

(Resto de código igual al proporcionado en el Listado 1)

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp1/ejer7] 2019-03-26 martes
$gcc SumaVectoresParalela.c -O2 -fopenmp -o sumaparalela
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp1/ejer7] 2019-03-26 martes
$./sumaparalela 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000009944      / 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) /
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp1/ejer7] 2019-03-26 martes
$./sumaparalela 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000019341      / Tamaño Vectores:11   / V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) / / V1[10]+V2[10]=V3[
10](2.100000+0.100000=2.200000) /
```

8. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores usando las `parallel` y `sections/section` (se debe aprovechar el paralelismo de datos usando estas directivas en lugar de la directiva `for`); es decir, hay que repartir el trabajo (tareas) entre varios threads usando `sections/section`. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Para obtener este tiempo usar la función `omp_get_wtime()` en lugar de `clock_gettime()`. NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para tamaños pequeños de los vectores (por ejemplo, N = 8); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de v1, v2 y v3 (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

RESPUESTA: Captura que muestre el código fuente implementado

```
//Inicializar vectores
#pragma omp parallel sections private(i)
{
    #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<(N/4)*3; i++){
        v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
    }
    }

    #pragma omp section
    {
    for(i=(N/4)*3; i<N; i++){
        v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
    }
    }
}
```

```
cgt1 = omp_get_wtime();
//Calcular suma de vectores
#pragma omp parallel sections private(i)
{
    #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<(N/4)*3; i++)
        v3[i] = v1[i] + v2[i];
    }

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

(Resto de código igual al ejer7, solo ha cambiado la inicialización y la suma de vectores)

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp1/ejer8] 2019-03-26 martes
$gcc SumaVectoresParalelaSections.c -O2 -fopenmp -o sumaparalela
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp1/ejer8] 2019-03-26 martes
$./sumaparalela 8
Tamaño Vectores:8 (4 B)
Tiempo:0.000021680       / 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) /
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp1/ejer8] 2019-03-26 martes
$./sumaparalela 11
Tamaño Vectores:11 (4 B)
Tiempo:0.000021219       / Tamaño Vectores:11   / V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) / / V1[10]+V2[10]=V3[
10](2.100000+0.100000=2.200000) /
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Prácticas/bp1/ejer8] 2019-03-26 martes
```

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 solo se divide la ejecución en 4 secciones como máximo podrá utilizar 4threads/cores.

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

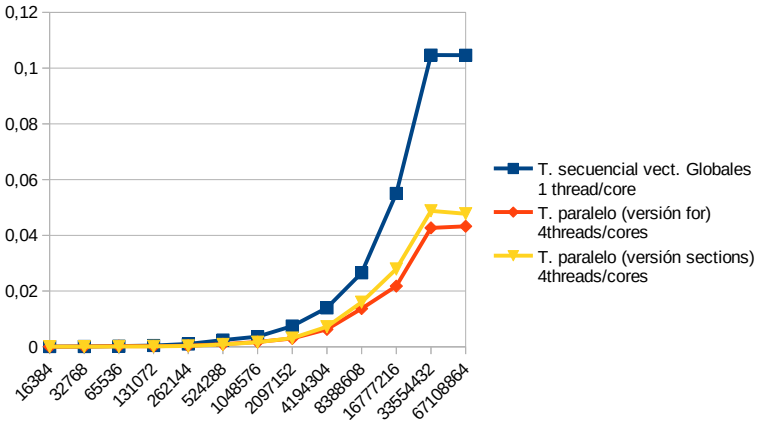
    RESPUESTA:

Nº de Componentes	**T. secuencial vect. Globales 1 thread/core**	**T. paralelo (versión `for`) 4threads/ cores**	**T. paralelo (versión `sections`) 4threads/cores**
16384	0,000066214	0,000050654	0,000049453
32768	0,000096216	0,000098200	0,000046089
65536	0,000222404	0,000228018	0,000087582
131072	0,000453914	0,000178689	0,000141362
262144	0,001009154	0,000394437	0,000297048
524288	0,002403822	0,000918583	0,000831904
1048576	0,003643432	0,001717630	0,001639852
2097152	0,007446155	0,002987181	0,003129002
4194304	0,013969847	0,006273043	0,007179576
8388608	0,026524109	0,013715319	0,016005514
16777216	0,055032949	0,021776534	0,027904833
33554432	0,104685235	0,042672667	0,048794954
67108864	0,104624595	0,043238264	0,047738707

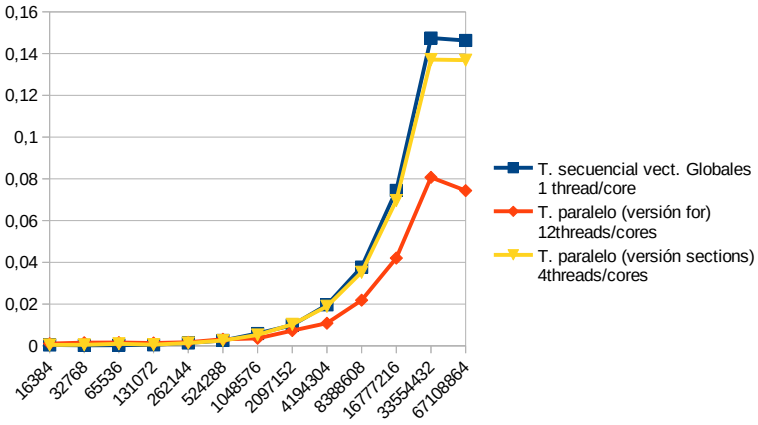
PC

Nº de Componentes	**T. secuencial vect. Globales 1 thread/core**	**T. paralelo (versión `for`) 12threads/ cores**	**T. paralelo (versión `sections`) 4threads/cores**
16384	0,000430245	0,001053127	0,000442399
32768	0,000246033	0,001511177	0,000481220
65536	0,000330268	0,001567631	0,000957006
131072	0,000469258	0,001374379	0,000576200
262144	0,001386223	0,001718451	0,001363641
524288	0,002571877	0,003166257	0,002577746
1048576	0,005910737	0,003611784	0,005214944
2097152	0,009908823	0,007307391	0,010284429
4194304	0,019635588	0,010844356	0,018938489
8388608	0,037749301	0,021871563	0,035249872
16777216	0,074488735	0,042045458	0,069647501
33554432	0,147458447	0,080711272	0,137113675
67108864	0,146286838	0,074351656	0,136848217

Atcgrid



PC



Atcgrid

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

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:

Nº de Componentes	**Tiempo secuencial vect. Globales 1 thread/core**			**Tiempo paralelo/versión `for` 12 Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	0,003	0,001	0,002	0,005	0,007	0,030
131072	0,004	0,002	0,001	0,006	0,012	0,034
262144	0,005	0,003	0,003	0,007	0,023	0,034
524288	0,009	0,004	0,005	0,008	0,023	0,041
1048576	0,015	0,011	0,004	0,010	0,048	0,049
2097152	0,028	0,012	0,016	0,013	0,086	0,049
4194304	0,048	0,028	0,020	0,026	0,179	0,081
8388608	0,086	0,046	0,040	0,041	0,290	0,150
16777216	0,164	0,089	0,075	0,074	0,514	0,320
33554432	0,324	0,172	0,152	0,150	1,078	0,586
67108864	0,324	0,179	0,144	0,168	1,075	0,574

En el caso de la suma secuencial, los tiempos de cpu son menores que el tiempo real. Como es un programa secuencial y solo hay un flujo de control se cumple que el *elapsed time* tiene que ser mayor que el tiempo de cpu.

En cambio en la suma paralela el tiempo real es mucho menor que el tiempo de cpu, ya que se calcula con la suma de los tiempos que da cada flujo paralelo aunque este se ejecuten a la vez que los otros.

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						