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): Antonio Jesús Heredia Castillo
Grupo de prácticas: A2
Fecha de entrega: 3-04-2018
Fecha evaluación en clase: 3-04-2018

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

            }
        }
    }

}
```

```
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica1/codigo] 2018-03-27 martes
$./singleModificado.out
Introduce valor de inicialización a:12
Single ejecutada por el thread 1
b[0] = 12       Ejecutado por el thread 4
b[1] = 12       Ejecutado por el thread 4
b[2] = 12       Ejecutado por el thread 4
b[3] = 12       Ejecutado por el thread 4
b[4] = 12       Ejecutado por el thread 4
b[5] = 12       Ejecutado por el thread 4
b[6] = 12       Ejecutado por el thread 4
b[7] = 12       Ejecutado por el thread 4
b[8] = 12       Ejecutado por el thread 4
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica1/codigo] 2018-03-27 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`

   CAPTURAS DE PANTALLA:

```c
#include <stdio.h>
#include <omp.h>
int main() {
    int n = 9, i, a, b[n];
    for (i = 0; i < n; i++)
        b[i] = -1;
    #pragma omp parallel
    {
        #pragma omp single
        {
            printf("Introduce valor de inicialización a:");
            scanf("%d",  &a);
            printf("Single ejecutada por el thread %d\n",
            omp_get_thread_num());
        }
        #pragma omp for
            for (i = 0; i < n; i++)
                b[i] = a;

        #pragma omp master
        {
            for (i = 0; i < n; i++){
                printf("b[%d] = %d \t", i, b[i]);
                printf("Ejecutado por el thread %d",omp_get_thread_num() );
                printf("\n");

            }
        }
    }

}
```

```
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica1/codigo] 2018-03-27 martes
$./singleModificado2.out
Introduce valor de inicialización a:1
Single ejecutada por el thread 7
b[0] = 1        Ejecutado por el thread 0
b[1] = 1        Ejecutado por el thread 0
b[2] = 1        Ejecutado por el thread 0
b[3] = 1        Ejecutado por el thread 0
b[4] = 1        Ejecutado por el thread 0
b[5] = 1        Ejecutado por el thread 0
b[6] = 1        Ejecutado por el thread 0
b[7] = 1        Ejecutado por el thread 0
b[8] = 1        Ejecutado por el thread 0
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica1/codigo] 2018-03-27 martes
$./singleModificado2.out
Introduce valor de inicialización a:21244
Single ejecutada por el thread 1
b[0] = 21244    Ejecutado por el thread 0
b[1] = 21244    Ejecutado por el thread 0
b[2] = 21244    Ejecutado por el thread 0
b[3] = 21244    Ejecutado por el thread 0
b[4] = 21244    Ejecutado por el thread 0
b[5] = 21244    Ejecutado por el thread 0
b[6] = 21244    Ejecutado por el thread 0
b[7] = 21244    Ejecutado por el thread 0
b[8] = 21244    Ejecutado por el thread 0
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica1/codigo] 2018-03-27 martes
$
```

RESPUESTA A LA PREGUNTA:

Pues que en este caso, el thread que muestra la inicialización del vector, siempre es el 0, que corresponde con 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:

Esto se debe a que master no tiene barrera implícita al inicio y por tanto la impresión por pantalla se puede hacer antes de que se haya realizado la suma debido a que se estas dos sentencias se realizan en paralelo.

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:

```
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica1/codigo] 2018-03-28 miér
coles
$time ./listado1.out 10000000
Tiempo(seg.):0.050059586        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)

real    0m0.245s
user    0m0.153s
sys     0m0.092s
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica1/codigo] 2018-03-28 miér
coles
$
```

Esto se debe a que al ejecutarse en un solo núcleo, el comando time puede registrar todo el tiempo que nuestro proceso pasa en el procesador.

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:

   RESPUESTA: cálculo de los MIPS y los MFLOPS

```
[AntonioJesusHerediaCastillo A2estudiante7@atcgrid:~] 2018-03-28 miércoles
$echo './listado1.out 10' | qsub -q ac
69408.atcgrid
[AntonioJesusHerediaCastillo A2estudiante7@atcgrid:~] 2018-03-28 miércoles
$echo './listado1.out 10000000' | qsub -q ac
69410.atcgrid
[AntonioJesusHerediaCastillo A2estudiante7@atcgrid:~] 2018-03-28 miércoles
$cat STDIN.o69408
Tiempo(seg.):0.000000265         Tamaño Vectores:10      V1[0]+V2[0]=V3[0](1.00000
0+1.000000=2.000000) V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000)
[AntonioJesusHerediaCastillo A2estudiante7@atcgrid:~] 2018-03-28 miércoles
$cat STDIN.o69410
Tiempo(seg.):0.093759368         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)
```

Tamaño	10
NI	18*10=180
FPO	4*10 = 40
Tiempo	0.000000265
MIPS	180/(0.000000265*10^6)=679.245283
MFLOPS	40/(0.000000265*10^6)=150.943396226

Tamaño	10000000
NI	18*10000000=180000000
FPO	4*10 = 40000000
Tiempo	0.000000265
MIPS	180000000/(0.093759368*10^6)=1919.80816253
MFLOPS	40000000/(0.093759368*10^6)=426.624036118

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

```
194     movl    $0, %edi
195     call    clock_gettime
196     movl    $0, -160(%rbp)
197 .L10:
198     movl    -160(%rbp), %eax
199     cmpl    -156(%rbp), %eax
200     jnb .L9
201     movq    -144(%rbp), %rax
202     movl    -160(%rbp), %edx
203     movslq  %edx, %rdx
204     movsd   (%rax,%rdx,8), %xmm1
205     movq    -128(%rbp), %rax
206     movl    -160(%rbp), %edx
207     movslq  %edx, %rdx
208     movsd   (%rax,%rdx,8), %xmm0
209     addsd   %xmm1, %xmm0
210     movq    -112(%rbp), %rax
211     movl    -160(%rbp), %edx
212     movslq  %edx, %rdx
213     movsd   %xmm0, (%rax,%rdx,8)
214     addl    $1, -160(%rbp)
215     jmp .L10
216 .L9:
217     leaq    -80(%rbp), %rax
218     movq    %rax, %rsi
219     movl    $0, %edi
220     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

```
    printf("Error en la reserva de espacio para los vectores\n");
    exit(-2);
    }
    double start, end , diferencia;
    #pragma omp parallel
    {
        //Inicializar vectores
        #pragma omp 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
        }
        #pragma single
            start = omp_get_wtime();
        //Calcular suma de vectores
        #pragma omp for
        for(i=0; i<N; i++)
            v3[i] = v1[i] + v2[i];

        #pragma single
        end = omp_get_wtime();
    }
    diferencia=end-start;
    //Imprimir resultado de la suma y el tiempo de ejecución
#ifdef PRINTF_ALL
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica1/codigo] 2018-03-30 viernes
$gcc -fopenmp listado1_parallel.c -o listado1_parallel.out
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica1/codigo] 2018-03-30 viernes
$./listado1_parallel.out 8
Tiempo(seg.):0.000004211        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)
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica1/codigo] 2018-03-30 viernes
$./listado1_parallel.out 11
Tiempo(seg.):0.008074775        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)
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica1/codigo] 2018-03-30 viernes
```

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

```
31      #pragma omp parallel
32      {
33          //Inicializar vectores
34          #pragma omp sections
35          {
36              #pragma omp section
37                  for(int i=0; i<N/4; i++){
38                      v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
39                  }
40              #pragma omp section
41                  for(int i=N/4; i<N/2; i++){
42                      v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
43                  }
44              #pragma omp section
45                  for(int i=N/2; i<(3*N)/4; i++){
46                      v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
47                  }
48              #pragma omp section
49                  for(int i=(3*N)/4; i<N; i++){
50                      v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
51                  }
52          }
53          #pragma single
54              start = omp_get_wtime();
55          //Calcular suma de vectores
56          #pragma omp sections
57          {
58              #pragma omp section
59                  for(int i=0; i<N/4; i++)
60                      v3[i] = v1[i] + v2[i];
61              #pragma omp section
62                  for(int i=N/4; i<N/2; i++)
63                      v3[i] = v1[i] + v2[i];
64              #pragma omp section
65                  for(int i=N/2; i<(3*N)/4; i++)
66                      v3[i] = v1[i] + v2[i];
67              #pragma omp section
68                  for(int i=(3*N)/4; i<N; i++)
69                      v3[i] = v1[i] + v2[i];
70          }
71
72          #pragma single
73          end = omp_get_wtime();
```

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

```
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica1/codigo] 2018-03-31 sábado
$gcc -fopenmp listado1_section.c -o listado1_section.out
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica1/codigo] 2018-03-31 sábado
$./listado1_section.out 8
Tiempo(seg.):0.000004051        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)
[AntonioJesusHerediaCastillo antonio@antonio-Lenovo:~/Documentos/git/AC/Practica1/codigo] 2018-03-31 sábado
$./listado1_section.out 11
Tiempo(seg.):0.000003137        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:

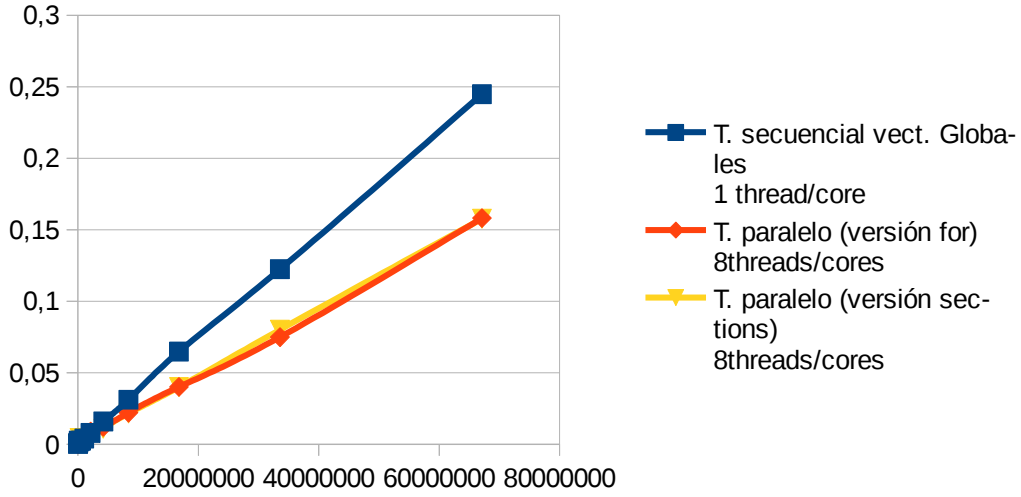
En la implementación que he hecho, como maximo se van a usar 4 thread, ya que en cada sections del programa, solo tengo 4 section y por tanto solo se podría paralelizar de 4 en 4.

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:

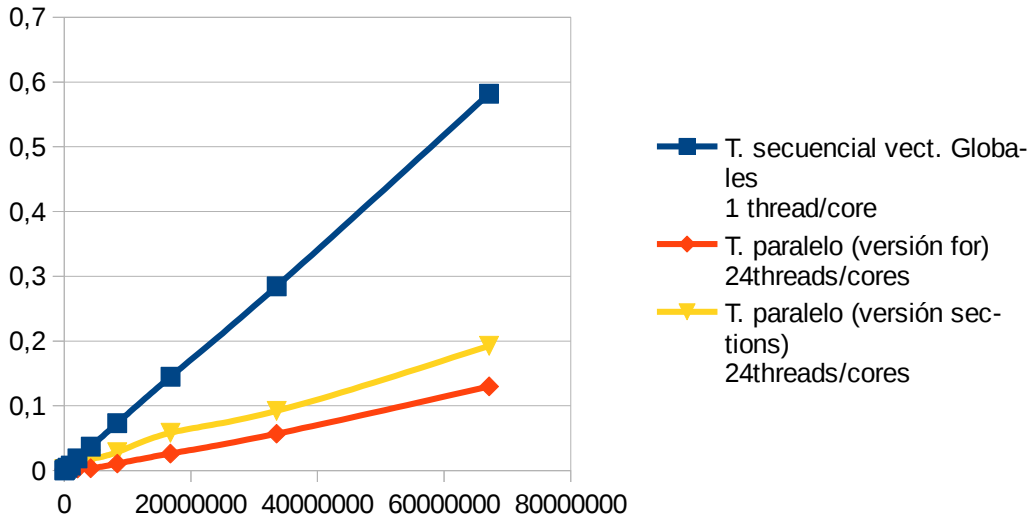
PC

Nº de Componentes	**T. secuencial vect. Globales** **1 thread/core**	**T. paralelo (versión `for`)** **8threads/cores**	**T. paralelo (versión `sections`)** **8threads/cores**
16384	0.000274702	0.000107806	0.000058419
32768	0.000537567	0.000216586	0.000071818
65536	0.001115792	0.000461290	0.000132523
131072	0.002504408	0.000737147	0.000220478
262144	0.002215816	0.001077633	0.004517707
524288	0.002528973	0.001440026	0.001018609
1048576	0.004063949	0.002827901	0.001924182
2097152	0.007982115	0.008881472	0.004181355
4194304	0.016045593	0.012175077	0.010873697
8388608	0.031191136	0.022170801	0.021621807
16777216	0.064883159	0.040143195	0.040023039
33554432	0.122388820	0.074980931	0.080085626
67108864	0.244924885	0.158192586	0.157748644



ATCGRID

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión `for`) 24threads/cores	T. paralelo (versión `sections`) 24threads/cores
16384	0.000165977	0.001993754	0.002421947
32768	0.000365556	0.001987238	0.002181734
65536	0.000732465	0.002264841	0.002002745
131072	0.001461952	0.002226536	0.002037567
262144	0.002158999	0.002202116	0.002090927
524288	0.004410715	0.002280049	0.002577502
1048576	0.007683628	0.002313775	0.003662886
2097152	0.018832358	0.002919654	0.007585611
4194304	0.036948177	0.003483543	0.017490341
8388608	0.073198832	0.010752238	0.028463577
16777216	0.144711451	0.026214737	0.058380680
33554432	0.284444731	0.056924541	0.092189562
67108864	0.581811219	0.129826763	0.192626789



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.

En la ejecucion secuencial, el tiempo de CPU y el real coincide, ya que el tiempo que pasa el programa en el procesador es todo el tiempo que esta ejecutandose.

En cambio en la version con ejecución paralela esto no es asi, ya que el tiempo real es desde que empieza a ejecutarse hasta que termina la ejecucion y pero en este caso el tiempo de CPU es la suma de el tiempo que pasa en cada nucleo. Por tanto tiene que ser mayor el tiempo de CPU que el tiempo de ejecución.

RESPUESTA:

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` 24 Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	real	0m0.007s		real	0m0.012s	
	user	0m0.004s		user	0m0.171s	
	sys	0m0.003s		sys	0m0.002s	
131072	real	0m0.003s		real	0m0.014s	
	user	0m0.003s		user	0m0.204s	
	sys	0m0.000s		sys	0m0.013s	
262144	real	0m0.011s		real	0m0.012s	
	user	0m0.004s		user	0m0.170s	
	sys	0m0.007s		sys	0m0.005s	
524288	real	0m0.019s		real	0m0.015s	
	user	0m0.013s		user	0m0.185s	
	sys	0m0.006s		sys	0m0.005s	
1048576	real	0m0.035s		real	0m0.013s	
	user	0m0.022s		user	0m0.172s	
	sys	0m0.012s		sys	0m0.023s	
2097152	real	0m0.064s		real	0m0.017s	
	user	0m0.035s		user	0m0.183s	
	sys	0m0.029s		sys	0m0.091s	
4194304	real	0m0.124s		real	0m0.032s	
	user	0m0.072s		user	0m0.296s	
	sys	0m0.051s		sys	0m0.140s	
8388608	real	0m0.243s		real	0m0.048s	
	user	0m0.152s		user	0m0.427s	
	sys	0m0.090s		sys	0m0.305s	
16777216	real	0m0.473s		real	0m0.082s	
	user	0m0.280s		user	0m0.646s	
	sys	0m0.190s		sys	0m0.637s	
33554432	real	0m0.946s		real	0m0.159s	
	user	0m0.592s		user	0m1.219s	
	sys	0m0.348s		sys	0m1.244s	
67108864	real	0m1.884s		real	0m0.347s	
	user	0m1.120s		user	0m2.227s	
	sys	0m0.753s		sys	0m2.677s	