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 Gámiz Delgado
Grupo de prácticas: B
Fecha de entrega: 05/04/2018
Fecha evaluación en clase: –/--/--

Ejercicios basados en los ejemplos del seminario práctico

1. 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.

```c
// Antonio Gamiz Delgado

#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 de iteraciones\n");
        exit(-1);
    }

    n = atoi(argv[1]);

    #pragma omp parallel for
        for(i=0;i < n; i++)
            printf(" thread %d ejecuta la iteracion %d del bucle \n",
            omp_get_thread_num(), i);
    return(0);
}
```

```c
// Antonio Gamiz Delgados

#include <stdio.h>
#include <omp.h>

void funcA()
{
    printf("En funcA: esta seccion la ejecuta el thread %d\n", omp_get_thread_num());
}

void funcB()
{
    printf("En funcB: esta seccion 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.

```c
//Antonio Gamiz Delgado

#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 el valor de inicializacion 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("Despues de la region parallel: \n");
            for(i=0;i<n; i++) printf("b[%d]=%d\t",i,b[i]);
            printf("\nSingle ejecutada por el thread %d\n", omp_get_thread_num());
        }
    }
}

```

```
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ gcc -O2 -fopenmp ./source/singleModificado.c -o ./bin/singleModificado
./source/singleModificado.c:4:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ export OMP_DYNAMIC=FALSE
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ export OMP_NUM_THREADS=8
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ ./bin/singleModificado
Introduce el valor de inicializacion a: 5
Single ejecutada por el thread 0
Despues de la region parallel:
b[0]=5  b[1]=5  b[2]=5  b[3]=5  b[4]=5  b[5]=5  b[6]=5  b[7]=5  b[8]=5
Single ejecutada por el thread 2
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ ./bin/singleModificado
Introduce el valor de inicializacion a: 4
Single ejecutada por el thread 7
Despues de la region parallel:
b[0]=4  b[1]=4  b[2]=4  b[3]=4  b[4]=4  b[5]=4  b[6]=4  b[7]=4  b[8]=4
Single ejecutada por el thread 4
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$
```

. 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?

```
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ gcc -O2 -fopenmp ./source/singleModificado2.c -o ./bin/singleModificado2
./source/singleModificado2.c:6:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ export OMP_DYNAMIC=FALSE
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ export OMP_NUM_THREADS=8
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ ./bin/singleModificado2
Introduce el valor de inicializacion a: 34
Single ejecutada por el thread 6
Despues de la region parallel:
b[0]=34 b[1]=34 b[2]=34 b[3]=34 b[4]=34 b[5]=34 b[6]=34 b[7]=34 b[8]=34
Single ejecutada por el thread 0
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$ ./bin/singleModificado2
Introduce el valor de inicializacion a: 345
Single ejecutada por el thread 6
Despues de la region parallel:
b[0]=345        b[1]=345        b[2]=345        b[3]=345        b[4]=345        b[5]=345        b[6]=345        b[7]=345        b[8]=345
Single ejecutada por el thread 0
[antoniogamizdelgado antonio@antonio:~/ArquitecturaDeComputadores/Práctica 2] 2018-03-12 lunes
$
```

```c
//Antonio Gamiz Delgado

#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 el valor de inicializacion 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("Despues de la region parallel: \n");
            for(i=0;i<n; i++) printf("b[%d]=%d\t",i,b[i]);
            printf("\nSingle ejecutada por el thread %d\n", omp_get_thread_num());
        }
    }
}
```

Vemos que al añadir la directiva master, la hebra que siempre ejecuta el print de salida es la hebra 0, es decir, la 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:

Si eliminámaramos la directiva barrier del programa, los threads no se sincronizarían correctamente, es decir, como los threads se ejecutan en ordénes aleatorioas, podría darse el caso de que el primer thread que terminara fuera el 0, sin haber terminado los otros, por lo que el resultado que imprimiría sería incorrecto por los demás threads no han actuado todavía sobre la variable suma.

En cambio, si ponemos la directiva barrier, expresamos explicitamente que los threads que terminen antes se esperen hasta que terminen los demás, y así obtener el resultado correcto.

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:

```
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ gcc -O2 ./source/listado1.c -o ./bin/listado1 -lrt
```

Compilamos el programa.

```
sftp> lcd ArquitecturaDeComputadores/Práctica\ 2/
sftp> lpwd
Local working directory: /home/antonio/ArquitecturaDeComputadores/Práctica 2
sftp> pwd
Remote working directory: /home/E2estudiante14
sftp> cd practica2/
sftp> ls
sftp> put ./bin/listado1
Uploading ./bin/listado1 to /home/E2estudiante14/practica2/listado1
./bin/listado1                                                100% 8968     8.8KB/s   00:00
sftp>
```

Subimos el programa a atcgrid.

```
[E2estudiante14@atcgrid practica2]$ ls
listado1
[E2estudiante14@atcgrid practica2]$ echo 'time practica2/listado1 10000000' | qsub -q ac
68311.atcgrid
[E2estudiante14@atcgrid practica2]$
```

Ejecutamos la orden ‘time’,

```
sftp> get STDIN.o68311
Fetching /home/E2estudiante14/practica2/STDIN.o68311 to STDIN.o68311
/home/E2estudiante14/practica2/STDIN.o68311                   100%  205     0.2KB/s   00:00
sftp> get STDIN.e68311
Fetching /home/E2estudiante14/practica2/STDIN.e68311 to STDIN.e68311
/home/E2estudiante14/practica2/STDIN.e68311                   100%   42     0.0KB/s   00:00
sftp>
```

Traemos los ficheros desde atcgrid a local.

```
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ cat ./outputs/STDIN.o68311
Tiempo (seg.): 0.054919933      / Tamaño vectores: 10000000    / v1[0]+v2[0]=v3[0](1000000.000000+1000000.000000=2000
000.000000) / / v1[9999999]+v2[9999999]=v3[9999999](1999999.900000+0.100000=2000000.000000) /
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ cat ./outputs/STDIN.e68311

real    0m0.163s
user    0m0.063s
sys     0m0.096s
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶
```

Como vemos, la orden ‘time’ muestra su resultado en la salido de error.

Vemos que el CPU time es 0.159s (=user+sys=0.063+0.096), que es mayor que el tiempo real, 0.163s.

Esto es debido a que el tiempo ‘user’ indica el tiempo que ha pasado el programa en tiempo de ejecución de espacio del usuario, y ‘sys’ en el nivel kernel del SO, mientras que el elapsed time también tiene ese valor más el asociado a interrupciones que sufre el programa, ya sea por la espera de I/O o ejecuaciones de otros procesos del SO.

. 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:

```
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ gcc -O2 -S ./source/listado1.c
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ ls
bin  guion  listado1.s  outputs  screenshots  source
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶
```

```
[E2estudiante14@atcgrid practica2]$ echo 'time practica2/listado1 10' | qsub -q ac
68312.atcgrid
[E2estudiante14@atcgrid practica2]$
```

```
sftp> get STDIN.o68312
Fetching /home/E2estudiante14/practica2/STDIN.o68312 to STDIN.o68312
/home/E2estudiante14/practica2/STDIN.o68312                   100%  151     0.2KB/s   00:00
sftp> get STDIN.e68312
Fetching /home/E2estudiante14/practica2/STDIN.e68312 to STDIN.e68312
/home/E2estudiante14/practica2/STDIN.e68312                   100%   42     0.0KB/s   00:00
sftp>
```

```
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ cat ./outputs/STDIN.o68312
Tiempo (seg.): 0.000002262      / 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) /
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ cat ./outputs/STDIN.e68312

real    0m0.004s
user    0m0.001s
sys     0m0.000s
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶
```

Generamos el código ensamblador asociado al listado1.c con la opción -O2.

Ejecutamos el programa listado1 con n=10 ya que para el caso 10000000 podemos usar los datos del ejercicio 5

RESPUESTA: cálculo de los MIPS y los MFLOPS

Para el cálculo de los MIPS necesitamos el número de instrucciones, por lo que contamos el número de línes del código ensamblador que tiene el bucle que realiza la suma. En este caso, buscamos la primera llamada a clock_gettime y vemos que el bucle es .L5, el cuál cuenta con 6 instrucciones, por lo que la fórmula del cálculo de los MIPS de teoría nos dice que:

MIPS(n=10)=(6*10)/(0.001*10^6)=0,06 MIPS

MIPS(n=10000000)=(6*10^7)/(0.159*10^6)=377,35 MIPS

Para calcular los MFLOPS primero vemos cuantas operaciones en coma flotante son realizadas en la suma, en este caso 1. Luego aplicanco la fórmula para el cálculo de los MFLOPS de teoría tenemos:

MFPLOPS(n=10)=10/(0.001*10^6)=0.01 MFLOPS

MFLOPS(n=10000000)=10^7/(0.159*10^6)=62,89 MFLOPS

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

```
                call            clock_gettime
                xorl            %eax, %eax
                .p2align 4,,10
                .p2align 3
.L5:
                movsd           v1(%rax), %xmm0
                addq            $8, %rax
                addsd           v2-8(%rax), %xmm0
                movsd           %xmm0, v3-8(%rax)
                cmpq            %rax, %rbx
                jne             .L5
.L6:
                leaq            16(%rsp), %rsi
                xorl            %edi, %edi
                call            clock_gettime
                movq            24(%rsp), %rax
                subq            8(%rsp), %rax
                movl            %r12d, %edx
                pxor            %xmm0, %xmm0
                movl            %r12d, %ecx
                movsd           v3(,%rdx,8), %xmm6
                movl            %r12d, %r9d
                movsd           v2(,%rdx,8), %xmm5
                movl            %r12d, %r8d
                cvtsi2sdq       %rax, %xmm0
                movq            16(%rsp), %rax
                subq            (%rsp), %rax
                movsd           v1(,%rdx,8), %xmm4
```

```
                movsd   v3(%rip), %xmm3
                movl    %ebp, %edx
                movsd   v2(%rip), %xmm2
                movl    $.LC3, %esi
                movl    $1, %edi
                movapd  %xmm0, %xmm1
                pxor    %xmm0, %xmm0
                divsd   .LC2(%rip), %xmm1
                cvtsi2sdq       %rax, %xmm0
                movl    $7, %eax
                addsd   %xmm1, %xmm0
                movsd   v1(%rip), %xmm1
                call    __printf_chk
                xorl    %eax, %eax
                movq    40(%rsp), %rcx
                xorq    %fs:40, %rcx
                jne     .L15
                addq    $48, %rsp
                .cfi_remember_state
                .cfi_def_cfa_offset 32
                popq    %rbx
                .cfi_def_cfa_offset 24
                popq    %rbp
                .cfi_def_cfa_offset 16
                popq    %r12
                .cfi_def_cfa_offset 8
                ret
.L3:
                .cfi_restore_state
                movq    %rsp, %rsi
                xorl    %edi, %edi
                orl     $-1, %r12d
                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

```
//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
}

#pragma omp barrier

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];
#pragma omp barrier

double end = omp_get_wtime();

ncgt= (double)(cgt2.tv_sec - cgt1.tv_sec) + (double) ((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));

//Imprimir resultado de la suma y el tiempo de ejecución
printf("Tiempo(seg): %11.9f\t / tamaño vectores: %u\n", (float)(end-start), 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]);
```

```
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ ./bin/listado1_omp 8
Tiempo(seg): 0.000018881         / 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) /
```

```
antoniogamizdelgado 2018-03-22 jueves  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ ./bin/listado1_omp 11
Tiempo(seg): 0.000005690         / 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

```
#pragma omp parallel
{
  #pragma omp sections
  {
    #pragma omp section
    for(int 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(int 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(int 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(int 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 barrier


double start = omp_get_wtime();

#pragma omp parallalel
{
  #pragma omp sections
  {
    #pragma omp section
    for(int i=0; i<N/4; i++) v3[i] = v1[i] + v2[i];
    #pragma omp section
    for(int i=N/4; i<N/2; i++) v3[i] = v1[i] + v2[i];
    #pragma omp section
    for(int i=N/2; i<3*N/4; i++) v3[i] = v1[i] + v2[i];
    #pragma omp section
    for(int i=3*N/4; i<N; i++) v3[i] = v1[i] + v2[i];
  }
}
```

```
antoniogamizdelgado 2018-04-04 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ gcc -fopenmp -O2 ./source/listado1_omp2.c -o ./bin/listado1_omp2 -lrt
```

```
antoniogamizdelgado 2018-04-04 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ ./bin/listado1_omp2 8
Tiempo(seg): 0.000005149         / 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) /
antoniogamizdelgado 2018-04-04 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ ./bin/listado1_omp2 11
Tiempo(seg): 0.000010564         / 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) /
antoniogamizdelgado 2018-04-04 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶
```

. 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.

En el ejercicio 7, al igual que en el 8, el número máximo de cores que podemos poner es el número de cores físicos que tengamos, por lo que en el caso de ATCGRID es 12 físicos (24 lógicos) y en mi PC 4 físicos (8 lógicos).

En el caso de los treads, ejercicio 7, el número de threads podría ser como mucho N, pero esto no tendría sentido porque entonces no sería totalmente paralelo sino concurrente. Por lo que el número máximo recomendado sería el número de cores lógicos de los que dispongas. En el ejercicio 8 es diferente ya que usamos parallalel y sections, en concreto 4 secciones, por lo que el número máximo de threads sería 4 ya que cada section la ejecuta un thread.

. 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:

```
antoniogamizdelgado 2018-04-04 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ gcc -fopenmp -O2 ./source/listado1_omp.c -o ./bin/listado1_omp_for -lrt
antoniogamizdelgado 2018-04-04 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ gcc -fopenmp -O2 ./source/listado1_omp2.c -o ./bin/listado1_omp_sections -lrt
antoniogamizdelgado 2018-04-04 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ gcc -O2 ./source/listado1.c -o ./bin/listado1_secuencial -lrt
antoniogamizdelgado 2018-04-04 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶
```

```
antoniogamizdelgado 2018-04-04 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ bash ./scripts/listado1_secuencial.sh
Tiempo (seg.): 0.000240947      / Tamaño vectores: 16384      / v1[0]+v2[0]=v3[0](1638.400000+1638.400000=3276.800000) / / v1[16383]+v2[16383]=v3[16383](3276.700000+0.100000=3276.800000) /
Tiempo (seg.): 0.000472349      / Tamaño vectores: 32768      / v1[0]+v2[0]=v3[0](3276.800000+3276.800000=6553.600000) / / v1[32767]+v2[32767]=v3[32767](6553.500000+0.100000=6553.600000) /
Tiempo (seg.): 0.000936146      / Tamaño vectores: 65536      / v1[0]+v2[0]=v3[0](6553.600000+6553.600000=13107.200000) / / v1[65535]+v2[65535]=v3[65535](13107.100000+0.100000=13107.200000) /
Tiempo (seg.): 0.002417616      / Tamaño vectores: 131072     / v1[0]+v2[0]=v3[0](13107.200000+13107.200000=26214.400000) / / v1[131071]+v2[131071]=v3[131071](26214.300000+0.100000=26214.400000) /
Tiempo (seg.): 0.004762777      / Tamaño vectores: 262144     / v1[0]+v2[0]=v3[0](26214.400000+26214.400000=52428.800000) / / v1[262143]+v2[262143]=v3[262143](52428.700000+0.100000=52428.800000) /
Tiempo (seg.): 0.006928669      / Tamaño vectores: 524288     / v1[0]+v2[0]=v3[0](52428.800000+52428.800000=104857.600000) / / v1[524287]+v2[524287]=v3[524287](104857.500000+0.100000=104857.600000) /
Tiempo (seg.): 0.007855402      / Tamaño vectores: 1048576    / v1[0]+v2[0]=v3[0](104857.600000+104857.600000=209715.200000) / / v1[1048575]+v2[1048575]=v3[1048575](209715.100000+0.100000=209715.200000) /
Tiempo (seg.): 0.007705816      / Tamaño vectores: 2097152    / v1[0]+v2[0]=v3[0](209715.200000+209715.200000=419430.400000) / / v1[2097151]+v2[2097151]=v3[2097151](419430.300000+0.100000=419430.400000) /
Tiempo (seg.): 0.015389869      / Tamaño vectores: 4194304    / v1[0]+v2[0]=v3[0](419430.400000+419430.400000=838860.800000) / / v1[4194303]+v2[4194303]=v3[4194303](838860.700000+0.100000=838860.800000) /
Tiempo (seg.): 0.028458654      / Tamaño vectores: 8388608    / v1[0]+v2[0]=v3[0](838860.800000+838860.800000=1677721.600000) / / v1[8388607]+v2[8388607]=v3[8388607](1677721.500000+0.100000=1677721.600000) /
Tiempo (seg.): 0.057921890      / Tamaño vectores: 16777216   / v1[0]+v2[0]=v3[0](1677721.600000+1677721.600000=3355443.200000) / / v1[16777215]+v2[16777215]=v3[16777215](3355443.100000+0.100000=3355443.200000) /
Tiempo (seg.): 0.114348130      / Tamaño vectores: 33554432   / v1[0]+v2[0]=v3[0](3355443.200000+3355443.200000=6710886.400000) / / v1[33554431]+v2[33554431]=v3[33554431](6710886.300000+0.100000=6710886.400000) /
Tiempo (seg.): 0.229495705      / Tamaño vectores: 67108864   / v1[0]+v2[0]=v3[0](6710886.400000+6710886.400000=13421772.800000) / / v1[67108863]+v2[67108863]=v3[67108863](13421772.700000+0.100000=13421772.800000) /
```

```
antoniogamizdelgado 2018-04-04 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ bash ./scripts/listado1_omp_for.sh
Tiempo(seg): 0.000045804        / tamaño vectores: 16384      / v1[0]+v2[0]=v3[0](1638.400000+1638.400000=3276.800000) / / v1[16383]+v2[16383]=v3[16383](3276.700000+0.100000=3276.800000) /
Tiempo(seg): 0.000030642        / tamaño vectores: 32768      / v1[0]+v2[0]=v3[0](3276.800000+3276.800000=6553.600000) / / v1[32767]+v2[32767]=v3[32767](6553.500000+0.100000=6553.600000) /
Tiempo(seg): 0.000053606        / tamaño vectores: 65536      / v1[0]+v2[0]=v3[0](6553.600000+6553.600000=13107.200000) / / v1[65535]+v2[65535]=v3[65535](13107.100000+0.100000=13107.200000) /
Tiempo(seg): 0.000104791        / tamaño vectores: 131072     / v1[0]+v2[0]=v3[0](13107.200000+13107.200000=26214.400000) / / v1[131071]+v2[131071]=v3[131071](26214.300000+0.100000=26214.400000) /
Tiempo(seg): 0.000294293        / tamaño vectores: 262144     / v1[0]+v2[0]=v3[0](26214.400000+26214.400000=52428.800000) / / v1[262143]+v2[262143]=v3[262143](52428.700000+0.100000=52428.800000) /
Tiempo(seg): 0.002911939        / tamaño vectores: 524288     / v1[0]+v2[0]=v3[0](52428.800000+52428.800000=104857.600000) / / v1[524287]+v2[524287]=v3[524287](104857.500000+0.100000=104857.600000) /
Tiempo(seg): 0.002716797        / tamaño vectores: 1048576    / v1[0]+v2[0]=v3[0](104857.600000+104857.600000=209715.200000) / / v1[1048575]+v2[1048575]=v3[1048575](209715.100000+0.100000=209715.200000) /
Tiempo(seg): 0.012331035        / tamaño vectores: 2097152    / v1[0]+v2[0]=v3[0](209715.200000+209715.200000=419430.400000) / / v1[2097151]+v2[2097151]=v3[2097151](419430.300000+0.100000=419430.400000) /
Tiempo(seg): 0.008631895        / tamaño vectores: 4194304    / v1[0]+v2[0]=v3[0](419430.400000+419430.400000=838860.800000) / / v1[4194303]+v2[4194303]=v3[4194303](838860.700000+0.100000=838860.800000) /
Tiempo(seg): 0.021988427        / tamaño vectores: 8388608    / v1[0]+v2[0]=v3[0](838860.800000+838860.800000=1677721.600000) / / v1[8388607]+v2[8388607]=v3[8388607](1677721.500000+0.100000=1677721.600000) /
Tiempo(seg): 0.031336799        / tamaño vectores: 16777216   / v1[0]+v2[0]=v3[0](1677721.600000+1677721.600000=3355443.200000) / / v1[16777215]+v2[16777215]=v3[16777215](3355443.100000+0.100000=3355443.200000) /
Tiempo(seg): 0.059867896        / tamaño vectores: 33554432   / v1[0]+v2[0]=v3[0](3355443.200000+3355443.200000=6710886.400000) / / v1[33554431]+v2[33554431]=v3[33554431](6710886.300000+0.100000=6710886.400000) /
Tiempo(seg): 0.118390948        / tamaño vectores: 67108864   / v1[0]+v2[0]=v3[0](6710886.400000+6710886.400000=13421772.800000) / / v1[67108863]+v2[67108863]=v3[67108863](13421772.700000+0.100000=13421772.800000) /
```

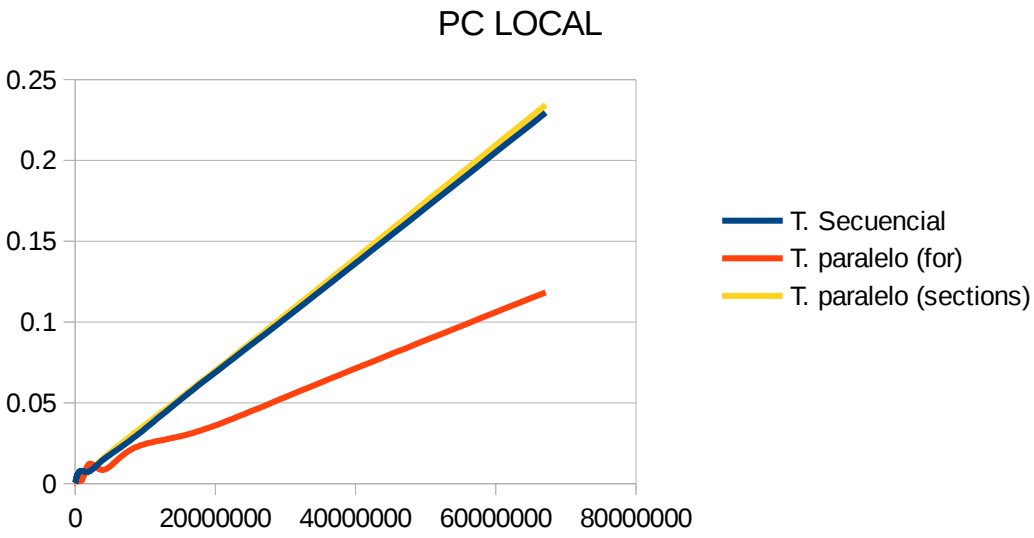
```
antoniogamizdelgado 2018-04-04 miércoles  @ antonio ~/ArquitecturaDeComputadores/Práctica 2 (master)
└─ $ ▶ bash ./scripts/listado1_omp_sections.sh
Tiempo(seg): 0.000264266        / tamaño vectores: 16384      / v1[0]+v2[0]=v3[0](1638.400000+1638.400000=3276.800000) / / v1[16383]+v2[16383]=v3[16383](3276.700000+0.100000=3276.800000) /
Tiempo(seg): 0.000514873        / tamaño vectores: 32768      / v1[0]+v2[0]=v3[0](3276.800000+3276.800000=6553.600000) / / v1[32767]+v2[32767]=v3[32767](6553.500000+0.100000=6553.600000) /
Tiempo(seg): 0.001087604        / tamaño vectores: 65536      / v1[0]+v2[0]=v3[0](6553.600000+6553.600000=13107.200000) / / v1[65535]+v2[65535]=v3[65535](13107.100000+0.100000=13107.200000) /
Tiempo(seg): 0.002036407        / tamaño vectores: 131072     / v1[0]+v2[0]=v3[0](13107.200000+13107.200000=26214.400000) / / v1[131071]+v2[131071]=v3[131071](26214.300000+0.100000=26214.400000) /
Tiempo(seg): 0.002327607        / tamaño vectores: 262144     / v1[0]+v2[0]=v3[0](26214.400000+26214.400000=52428.800000) / / v1[262143]+v2[262143]=v3[262143](52428.700000+0.100000=52428.800000) /
Tiempo(seg): 0.003025020        / tamaño vectores: 524288     / v1[0]+v2[0]=v3[0](52428.800000+52428.800000=104857.600000) / / v1[524287]+v2[524287]=v3[524287](104857.500000+0.100000=104857.600000) /
Tiempo(seg): 0.004276281        / tamaño vectores: 1048576    / v1[0]+v2[0]=v3[0](104857.600000+104857.600000=209715.200000) / / v1[1048575]+v2[1048575]=v3[1048575](209715.100000+0.100000=209715.200000) /
Tiempo(seg): 0.008158196        / tamaño vectores: 2097152    / v1[0]+v2[0]=v3[0](209715.200000+209715.200000=419430.400000) / / v1[2097151]+v2[2097151]=v3[2097151](419430.300000+0.100000=419430.400000) /
Tiempo(seg): 0.016338427        / tamaño vectores: 4194304    / v1[0]+v2[0]=v3[0](419430.400000+419430.400000=838860.800000) / / v1[4194303]+v2[4194303]=v3[4194303](838860.700000+0.100000=838860.800000) /
Tiempo(seg): 0.030971842        / tamaño vectores: 8388608    / v1[0]+v2[0]=v3[0](838860.800000+838860.800000=1677721.600000) / / v1[8388607]+v2[8388607]=v3[8388607](1677721.500000+0.100000=1677721.600000) /
Tiempo(seg): 0.058975976        / tamaño vectores: 16777216   / v1[0]+v2[0]=v3[0](1677721.600000+1677721.600000=3355443.200000) / / v1[16777215]+v2[16777215]=v3[16777215](3355443.100000+0.100000=3355443.200000) /
Tiempo(seg): 0.116722323        / tamaño vectores: 33554432   / v1[0]+v2[0]=v3[0](3355443.200000+3355443.200000=6710886.400000) / / v1[33554431]+v2[33554431]=v3[33554431](6710886.300000+0.100000=6710886.400000) /
Tiempo(seg): 0.234364092        / tamaño vectores: 67108864   / v1[0]+v2[0]=v3[0](6710886.400000+6710886.400000=13421772.800000) / / v1[67108863]+v2[67108863]=v3[67108863](13421772.700000+0.100000=13421772.800000) /
```

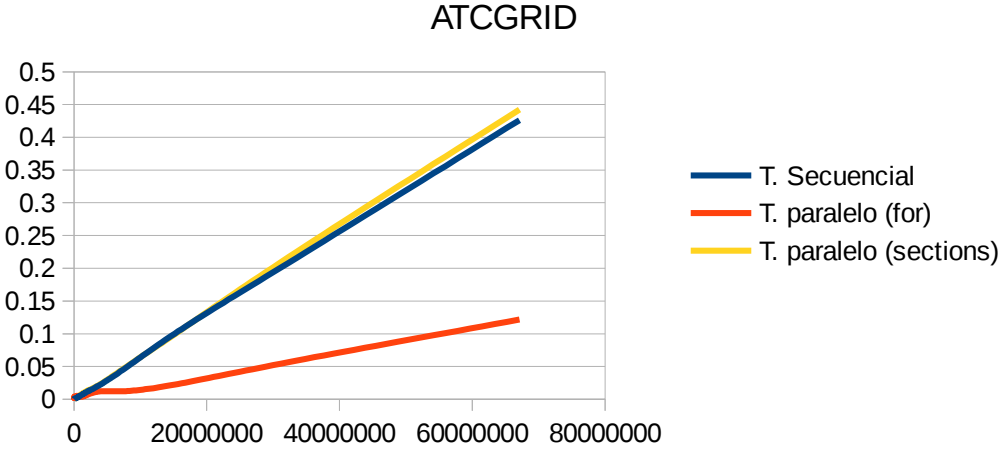
```
sftp> put listado1_secuencial
Uploading listado1_secuencial to /home/E2estudiante14/listado1_secuencial
listado1_secuencial                                                                   100% 8968     8.8KB/s   00:00
sftp> put listado1_omp_for
Uploading listado1_omp_for to /home/E2estudiante14/listado1_omp_for
listado1_omp_for                                                                      100%   13KB  13.1KB/s   00:00
sftp> put listado1_omp_sections
Uploading listado1_omp_sections to /home/E2estudiante14/listado1_omp_sections
listado1_omp_sections                                                                 100%   13KB  13.1KB/s   00:00
sftp> lcd ..
sftp> lcd scripts/
sftp> lls
atcgrid_listado1_omp_for.sh       atcgrid_listado1_secuencial.sh  listado1_omp_sections.sh
atcgrid_listado1_omp_sections.sh  listado1_omp_for.sh             listado1_secuencial.sh
sftp> put atcgrid_lis
atcgrid_listado1_omp_for.sh            atcgrid_listado1_omp_sections.sh       atcgrid_listado1_secuencial.sh

sftp> put atcgrid_listado1_secuencial.sh
Uploading atcgrid_listado1_secuencial.sh to /home/E2estudiante14/atcgrid_listado1_secuencial.sh
atcgrid_listado1_secuencial.sh                                                        100%  638     0.6KB/s   00:00
sftp> put atcgrid_listado1_omp_for.sh
Uploading atcgrid_listado1_omp_for.sh to /home/E2estudiante14/atcgrid_listado1_omp_for.sh
atcgrid_listado1_omp_for.sh                                                           100%  687     0.7KB/s   00:00
sftp> put atcgrid_listado1_omp_sections.sh
Uploading atcgrid_listado1_omp_sections.sh to /home/E2estudiante14/atcgrid_listado1_omp_sections.sh
atcgrid_listado1_omp_sections.sh                                                      100%  691     0.7KB/s   00:00
sftp>
```

```
[E2estudiante14@atcgrid ~]$ qsub atcgrid_listado1_secuencial.sh -q ac
71535.atcgrid
[E2estudiante14@atcgrid ~]$ qsub atcgrid_listado1_omp_for.sh -q ac
71536.atcgrid
[E2estudiante14@atcgrid ~]$ qsub atcgrid_listado1_omp_sections.sh -q ac
71537.atcgrid
[E2estudiante14@atcgrid ~]$ ls
atcgrid_listado1_omp_for.sh       listado1_omp_for         listado1_omp_sections       listado1_secuencial.o71535
atcgrid_listado1_omp_sections.sh  listado1_omp_for.e71536  listado1_secuencial
atcgrid_listado1_secuencial.sh    listado1_omp_for.o71536  listado1_secuencial.e71535
[E2estudiante14@atcgrid ~]$
```

```
sftp> get listado1_omp_for.o
listado1_omp_for.o71530      listado1_omp_for.o71533
sftp> get listado1_secuencial.o71535
Fetching /home/E2estudiante14/listado1_secuencial.o71535 to listado1_secuencial.o71535
/home/E2estudiante14/listado1_secuencial.o71535                    100% 3026     3.0KB/s   00:00
sftp> get listado1_omp_for.o71536
Fetching /home/E2estudiante14/listado1_omp_for.o71536 to listado1_omp_for.o71536
/home/E2estudiante14/listado1_omp_for.o71536                       100% 2997     2.9KB/s   00:00
sftp> get listado1_omp_sections.o71537
Fetching /home/E2estudiante14/listado1_omp_sections.o71537 to listado1_omp_sections.o71537
/home/E2estudiante14/listado1_omp_sections.o71537                  100% 3002     2.9KB/s   00:00
sftp>
```

PC LOCAL			
NºComponentes	T. Secuencial	T. paralelo (for)	T. paralelo (sections)
	8 threads 4 cores	8 threads 4 cores	4 threads 4 cores
16384	0.000240947	0.000045804	0.000264266
32768	0.000472349	0.000030642	0.000514873
65536	0.000936146	0.000053606	0.001087604
131072	0.002417616	0.000104791	0.002036407
262144	0.004762777	0.000294293	0.002327607
524288	0.006928669	0.002911939	0.003025020
1048576	0.007855402	0.002716797	0.004276281
2097152	0.007705816	0.012331035	0.008158196
4194304	0.015389869	0.008631895	0.016338427
8388608	0.028458654	0.021988427	0.030971842
16777216	0.057921890	0.031336799	0.058975976
33554432	0.114348130	0.059867896	0.116722323
67108864	0.229495705	0.118390948	0.234364092
ATCGRID			
NºComponentes	T. Secuencial	T. paralelo (for)	T. paralelo (sections)
	8 threads 4 cores	8 threads 4 cores	4 threads 4 cores
16384	0.000118087	0.004862905	0.000113438
32768	0.000237989	0.002143496	0.000179786
65536	0.000339675	0.003983061	0.000439332
131072	0.000919120	0.004576684	0.000963906
262144	0.001804331	0.004469179	0.001923010
524288	0.002811519	0.005095257	0.003262042
1048576	0.006082975	0.003591091	0.007254517
2097152	0.012044718	0.007286641	0.013262895
4194304	0.023672660	0.012298913	0.024393795
8388608	0.052281780	0.012774510	0.053040043
16777216	0.110859475	0.025379978	0.110343464
33554432	0.216051509	0.058994718	0.225261644
67108864	0.426167381	0.121697918	0.442281306


PC LOCAL
0.25
0.2
0.15
0.1
0.05
0
0
20000000
40000000
60000000
80000000
T. Secuencial
T. paralelo (for)
T. paralelo (sections)


ATCGRID
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
20000000
40000000
60000000
80000000
T. Secuencial
T. paralelo (for)
T. paralelo (sections)

. Rellenar una tabla como la Error: Reference source not found 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.

```
sftp> lcd ArquitecturaDeComputadores/Práctica\ 2/scripts/
sftp> lls
atcgrid_listado1_omp_for.sh       atcgrid_listado1_secuencial.sh       listado1_omp_sections.sh
atcgrid_listado1_omp_for_TIME.sh  atcgrid_listado1_secuencial_TIME.sh  listado1_secuencial.sh
atcgrid_listado1_omp_sections.sh  listado1_omp_for.sh
sftp> put atcgrid_listado1_omp_for_TIME.sh
Uploading atcgrid_listado1_omp_for_TIME.sh to /home/E2estudiante14/atcgrid_listado1_omp_for_TIME.sh
atcgrid_listado1_omp_for_TIME.sh                                      100%  679     0.7KB/s   00:00
sftp> put atcgrid_listado1_secuencial_TIME.sh
Uploading atcgrid_listado1_secuencial_TIME.sh to /home/E2estudiante14/atcgrid_listado1_secuencial_TIME.sh
atcgrid_listado1_secuencial_TIME.sh                                   100%  636     0.6KB/s   00:00
sftp>
```

```
[E2estudiante14@atcgrid ~]$ ls
atcgrid_listado1_omp_for.sh       atcgrid_listado1_secuencial.sh       listado1_omp_sections
atcgrid_listado1_omp_for_TIME.sh  atcgrid_listado1_secuencial_TIME.sh  listado1_secuencial
atcgrid_listado1_omp_sections.sh  listado1_omp_for
[E2estudiante14@atcgrid ~]$ qsub atcgrid_listado1_secuencial_TIME.sh -q ac
71546.atcgrid
[E2estudiante14@atcgrid ~]$ qsub atcgrid_listado1_omp_for_TIME.sh -q ac
71547.atcgrid
[E2estudiante14@atcgrid ~]$ ls
atcgrid_listado1_omp_for.sh       atcgrid_listado1_secuencial_TIME.sh  listado1_omp_sections
atcgrid_listado1_omp_for_TIME.sh  listado1_omp_for                     listado1_secuencial
atcgrid_listado1_omp_sections.sh  listado1_omp_for.e71547              listado1_secuencial.e71546
atcgrid_listado1_secuencial.sh    listado1_omp_for.o71547              listado1_secuencial.o71546
[E2estudiante14@atcgrid ~]$
```

```
sftp> get listado1_omp_for.e71547
Fetching /home/E2estudiante14/listado1_omp_for.e71547 to listado1_omp_for.e71547
/home/E2estudiante14/listado1_omp_for.e71547                          100%  546     0.5KB/s   00:01
sftp> get listado1_secuencial.e71546
Fetching /home/E2estudiante14/listado1_secuencial.e71546 to listado1_secuencial.e71546
/home/E2estudiante14/listado1_secuencial.e71546                       100%  546     0.5KB/s   00:00
sftp>
```

NºComponentes	Tiempo versión secuencial 1 thread/core				Tiempo versión for 1 thread/core			
	Elapsed	CPU-user	CPU-sys	CPU-total	Elapsed	CPU-user	CPU-sys	CPU-total
16384	0.001	0	0.001	0.001	0.012	0.174	0.001	0.175
32768	0.001	0.001	0	0.001	0.01	0.156	0.004	0.16
65536	0.002	0	0.002	0.002	0.011	0.172	0.004	0.176
131072	0.003	0	0.003	0.003	0.011	0.173	0.001	0.174
262144	0.005	0.001	0.003	0.004	0.012	0.179	0.002	0.181
524288	0.01	0.004	0.005	0.009	0.011	0.168	0.003	0.171
1048576	0.017	0.007	0.01	0.017	0.014	0.199	0.022	0.221
2097152	0.034	0.01	0.023	0.033	0.019	0.216	0.057	0.273
4194304	0.07	0.018	0.049	0.067	0.033	0.309	0.109	0.418
8388608	0.133	0.041	0.088	0.129	0.046	0.322	0.272	0.594
16777216	0.306	0.096	0.207	0.303	0.09	0.548	0.509	1.057
33554432	0.614	0.191	0.415	0.606	0.174	1.129	1.072	2.201
67108864	1.235	0.442	0.78	1.222	0.316	2.231	2.387	4.618

Vemos que en la versión secuencial el tiempo de CPU y el tiempo real son prácticament e iguales. Esto se debe a que no hay ni entradas ni salidas, sólo se realizan sumas e inicializaciones.

En cambio, en la versión de omp con for, vemos que el tiempo de CPU total es mayor que el real, lo que parece una contradicción. Esto se debe a que al usar varias hebras se suma el tiempo de ejecución de cada una de ellas dando lugar a un tiempo de CPU muy alto y un tiempo real de ejecución bastante bajo comparado con la versión secuencial.