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
/* --Elena Merelo Molina--
Compilamos con gcc -fopenmp bucle-for.c -o bucle-for, ejecutamos con ./bucle 5
y obtenemos:
thread 0 ejecuta la iteración 0 del bucle
thread 2 ejecuta la iteración 2 del bucle
thread 4 ejecuta la iteración 4 del bucle
thread 1 ejecuta la iteración 1 del bucle
thread 3 ejecuta la iteración 3 del bucle
*/

#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);
}
```

RESPUESTA: Captura que muestre el código fuente `sectionsModificado.c`

```c
// Elena Merelo Molina

#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();
  }
}
```

Resultado de la ejecución antes y después de modificar el código:

```
2018-03-14 23:07:01 elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
→ gcc -fopenmp sections.c -o sections

2018-03-14 23:07:20 elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
→ ./sections
En funcB: esta sección la ejecuta el thread 0
En funcA: esta sección la ejecuta el thread 1

2018-03-14 23:07:24 elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
→ gcc -fopenmp sections.c -o sections

2018-03-14 23:08:56 elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
→ ./sections
En funcB: esta sección la ejecuta el thread 2
En funcA: esta sección la ejecuta el thread 4
```

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
//Elena Merelo Molina

#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("Single ejecutada por el thread %d\n", omp_get_thread_num());
      }
    }
  }
}
```

Al compilarlo y ejecutarlo:

```
2018-03-15 17:53:33 elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
→ gcc -fopenmp single.c -o single

2018-03-15 17:53:54 elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
→ ./single
Introduce valor de inicialización a: 32
Single ejecutada por el thread 1
b[0] = 32       Single ejecutada por el thread 0
b[1] = 32       Single ejecutada por el thread 0
b[2] = 32       Single ejecutada por el thread 0
b[3] = 32       Single ejecutada por el thread 0
b[4] = 32       Single ejecutada por el thread 0
b[5] = 32       Single ejecutada por el thread 0
b[6] = 32       Single ejecutada por el thread 0
b[7] = 32       Single ejecutada por el thread 0
b[8] = 32       Single ejecutada por el thread 0
```

Probamos y ejecutamos más veces:

```
 2018-03-15 17:53:56  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → export OMP_DYNAMIC=FALSE

 2018-03-15 20:22:38  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → export OMP_NUM_THREADS=8

 2018-03-15 20:22:49  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → ./single
Introduce valor de inicialización a: 32
Single ejecutada por el thread 0
b[0] = 32       Single ejecutada por el thread 4
b[1] = 32       Single ejecutada por el thread 4
b[2] = 32       Single ejecutada por el thread 4
b[3] = 32       Single ejecutada por el thread 4
b[4] = 32       Single ejecutada por el thread 4
b[5] = 32       Single ejecutada por el thread 4
b[6] = 32       Single ejecutada por el thread 4
b[7] = 32       Single ejecutada por el thread 4
b[8] = 32       Single ejecutada por el thread 4
```

```
 2018-03-15 20:23:01  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → export OMP_NUM_THREADS=3

 2018-03-15 20:24:06  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → ./single
Introduce valor de inicialización a: 32
Single ejecutada por el thread 2
b[0] = 32       Single ejecutada por el thread 0
b[1] = 32       Single ejecutada por el thread 0
b[2] = 32       Single ejecutada por el thread 0
b[3] = 32       Single ejecutada por el thread 0
b[4] = 32       Single ejecutada por el thread 0
b[5] = 32       Single ejecutada por el thread 0
b[6] = 32       Single ejecutada por el thread 0
b[7] = 32       Single ejecutada por el thread 0
b[8] = 32       Single ejecutada por el thread 0
```

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
//Elena Merelo Molina

#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("Single ejecutada por el thread %d\n", omp_get_thread_num());
      }
    }
  }
}
```

Resultado de la ejecución:

```
2018-03-15 20:24:21 elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
→ gcc -fopenmp single_modificado_2.c -o single_modificado_2

2018-03-15 20:37:38 elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
→ ./single_modificado_2
Introduce valor de inicialización a: 32
Single ejecutada por el thread 0
b[0] = 32       Single ejecutada por el thread 0
b[1] = 32       Single ejecutada por el thread 0
b[2] = 32       Single ejecutada por el thread 0
b[3] = 32       Single ejecutada por el thread 0
b[4] = 32       Single ejecutada por el thread 0
b[5] = 32       Single ejecutada por el thread 0
b[6] = 32       Single ejecutada por el thread 0
b[7] = 32       Single ejecutada por el thread 0
b[8] = 32       Single ejecutada por el thread 0
```

Haciendo más pruebas:

```
 2018-03-15 20:37:48  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → export OMP_DYNAMIC=FALSE

 2018-03-15 20:39:40  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → export OMP_NUM_THREADS=8

 2018-03-15 20:39:43  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → ./single_modificado_2
Introduce valor de inicialización a: 32
Single ejecutada por el thread 2
b[0] = 32       Single ejecutada por el thread 0
b[1] = 32       Single ejecutada por el thread 0
b[2] = 32       Single ejecutada por el thread 0
b[3] = 32       Single ejecutada por el thread 0
b[4] = 32       Single ejecutada por el thread 0
b[5] = 32       Single ejecutada por el thread 0
b[6] = 32       Single ejecutada por el thread 0
b[7] = 32       Single ejecutada por el thread 0
b[8] = 32       Single ejecutada por el thread 0
```

```
 2018-03-15 20:39:47  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → export OMP_NUM_THREADS=3

 2018-03-15 20:40:08  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1
○ → ./single_modificado_2
Introduce valor de inicialización a: 32
Single ejecutada por el thread 0
b[0] = 32       Single ejecutada por el thread 0
b[1] = 32       Single ejecutada por el thread 0
b[2] = 32       Single ejecutada por el thread 0
b[3] = 32       Single ejecutada por el thread 0
b[4] = 32       Single ejecutada por el thread 0
b[5] = 32       Single ejecutada por el thread 0
b[6] = 32       Single ejecutada por el thread 0
b[7] = 32       Single ejecutada por el thread 0
b[8] = 32       Single ejecutada por el thread 0
```

RESPUESTA A LA PREGUNTA:

Lo que hay dentro de #pragma omp master es siempre ejecutado por el thread 0, a diferencia del programa anterior, ya que con #pragma omp single le indicas que lo que haya dentro de los corchetes ha de ser ejecutado por una única thread cada vez, mientras que con la directiva master además esta thread ha de ser la principal, la 0, y no pone una barrera implícita.

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: Al quitar la directiva barrier siempre devuelve que el tiempo es 0, exportemos el número de threads que exportemos, ya que no esperan a los tres segundos cuando se cumple que tid < omp_get_num_threads()/2, con lo que time(NULL) es 0. En definitiva la respuesta no siempre es correcta porque la directiva barrier identifica un punto de sincronización en el cual las threads de una región parallel no ejecutarán más allá de dicha #pragma omp barrier hasta que el resto hayan completado la tarea especificada. En este programa pues no esperan los tres segundos y por eso devuelven 0.

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:

Compilamos primeramente desde nuestro pc local:

```
2018-03-23 12:18:06  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1/src
→gcc -O2 listado1.c -o listado1 -lrt
```

Nos metemos desde sftp en la carpeta donde hemos generado el ejecutable

```
sftp> lcd Escritorio/University\ stuff/2º/2º\ Cuatrimestre/AC/Prácticas/BP1/src/
```

Mandamos el trabajo al front-end:

```
sftp> put listado1
Uploading listado1 to /home/E2estudiante10/listado1
listado1                                      100% 8968     8.8KB/s   00:00
```

Ejecutamos finalmente en atcgrid con la opción time y para un vector de tamaño 10000000 para obtener el tiempo de ejecución (*elapsed time*) y el tiempo de CPU del usuario y del sistema generado:

```
[ElenaMereloMolina E2estudiante10@atcgrid:~] 2018-03-23 viernes
$echo 'time ./listado1 10000000' | qsub -q ac
68392.atcgrid
```

El output generado es:

```
[ElenaMereloMolina E2estudiante10@atcgrid:~] 2018-03-23 viernes
$cat STDIN.o68392
Tiempo(seg.): 0.054006252       /Tamaño vectores:10000000       /V1[0] + V2[0]= V3[0]
(1000000.000000+1000000.000000=2000000.000000)// V1[9999999] + V2[9999999]= V3[999999
9](1999999.900000+0.100000=2000000.000000)/
```

Y los tiempo de ejecución:

```
[ElenaMereloMolina E2estudiante10@atcgrid:~] 2018-03-23 viernes
$cat STDIN.e68392

real    0m0.159s
user    0m0.047s
sys     0m0.103s
```

La suma de los tiempos de CPU del usuario y del sistema es 0m0.150s, que es menor que el tiepo real (elapsed) ya que ha habido un tiempo de espera asociado a la ejecución de otros programas o entrada/salida. Tiene sentido que nos salga esto teniendo en cuenta que en atcgrid se usan múltiples flujos de control, múltiples thread, luego pueden tener que esperarse unas a que terminen otras.

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:

   Generamos el código ensamblador:

```
2018-03-23 13:11:36  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1/src
→ gcc -O2 -S listado1.c -lrt
```

Lo mandamos al front-end:

```
sftp> put listado1.s
Uploading listado1.s to /home/E2estudiante10/listado1.s
listado1.s                                    100% 3022     3.0KB/s   00:00
```

Buscamos en el código ensamblador el primer clock_gettime() e identificamos el bucle for un poco más abajo:

```
        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
```

Éste .L5 se ejecutará 10000000 veces, el tamaño del vector, al realizar la suma para todas sus componentes, y consta de 6 instrucciones, luego MIPS = Número de instrucciones / (tiempo de cpu * 10 ^6). El tiempo de CPU a su vez es, del ejercicio anterior, user+sys time= 0m0.150s. Tenemos entonces que MIPS = (6 * 10^7) / (0.150 * 10^6) = 60 / 0.15 = 400 MIPS.

Si hubieran sido 10 componentes, MIPS = (6 * 10) / (0.15 * 10^6) = 1 / 2500 = 0.0004 MIPS.

Para hallar los MFLOPS, nos fijamos en las operaciones en coma flotante realizadas, que son 2, hay dos add, pero uno de ellos es para cambiar de registro, luego tenemos una operación en coma flotante, si hay 10 componentes en el vector los MFLOPS= 10 / (tiempo_cpu * 10^6) = 10 / (0.15 * 10^6) = 1 / 15000 = 0.000066666...(el 6 es periódico).

Si tenemos 10^7 componentes: MFLOPS= 10^7 / (0.15 * 10^6) = 10 / 0.15 = 200 / 3 = 66.67 (redondeando).

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

```
/*
File: listado1_omp.c
Para compilar usar (-lrt: real time library):
gcc -O2 - fopenmp listado1_omp.c -o listado1_omp -lrt
gcc -O2 -S -fopenmp listado1_omp.c -lrt    (para generar el código
ensamblador)
Para ejecutar: ./listado1_omp longitud
*/
```

```
#include <stdlib.h>   //biblioteca para funciones atoi(), malloc() y free()
#include <stdio.h>    //biblioteca donde se encuentra la función printf()
#include <time.h>     //biblioteca donde se encuentra la función
clock_gettime()
#include <math.h>


#ifdef _OPENMP  //Si se compila con la opción -fopenmp
  #include <omp.h>
#else
  #define omp_get_thread_num() 0
  #define omp_get_num_threads() 1
#endif


//#define PRINTF_ALL


#define MAX 67108865
double v1[MAX], v2[MAX], v3[MAX];




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


  //Leer argumento de entrada (nº de componentes del vector)
  if (argc<2){
    printf("Faltan nºcomponentes del vector\n");
    exit(-1);
  }


  unsigned int N= atoi(argv[1]);    //Máximo N= 2^32-1= 4294967295
(sizeof(unsigned int)= 4B)


  if (N> MAX) N= MAX;


  //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 //para que se asignen correctamente los valores


  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();
  double diff= 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", diff, N);
  for(i= 0; i<N; i++)
    printf("/V1[%d] + V2[%d] = V3[%d] (%8.6f + %8.6f = %8.6f) /\n", i, i, i,
v1[i], v2[i], v3[i]);
  #else
    printf("Tiempo(seg): %11.9f\t / tamaño vectores: %u\t/ v1[0]+v2[0]=v3[0]
(%8.6f+%8.6f=%8.6f) / / v1[%d]+v2[%d]=v3[%d](%8.6f+%8.6f=%8.6f) /\n", diff, N,
v1[0], v2[0], v3[0], N-1, N-1, N-1, v1[N-1], v2[N-1], v3[N-1]);
  #endif


}
```

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

Compilamos:

```
2018-04-05 22:17:46  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1/src
→ gcc -O2 -fopenmp listado1_omp.c -o listado1_omp -lrt
listado1_omp.c: In function 'main':
listado1_omp.c:76:10: warning: format '%d' expects argument of type 'int', but argument 2 has type 'double' [-Wf
ormat=]
   printf("\nv1[0]: %d, v1[%d]: %d, v2[0]: %d, v2[%d]: %d, v3[0]: %d, v3[%d]: %d", v
          ^
listado1_omp.c:76:10: warning: format '%d' expects argument of type 'int', but argument 3 has type 'double' [-Wf
ormat=]
listado1_omp.c:76:10: warning: format '%d' expects argument of type 'int', but argument 4 has type 'double' [-Wf
ormat=]
listado1_omp.c:76:10: warning: format '%d' expects argument of type 'int', but argument 5 has type 'double' [-Wf
ormat=]
listado1_omp.c:76:10: warning: format '%d' expects argument of type 'int', but argument 6 has type 'double' [-Wf
ormat=]
listado1_omp.c:76:10: warning: format '%d' expects argument of type 'int', but argument 7 has type 'double' [-Wf
ormat=]
listado1_omp.c:76:10: warning: format '%d' expects a matching 'int' argument [-Wformat=]
listado1_omp.c:76:10: warning: format '%d' expects a matching 'int' argument [-Wformat=]
listado1_omp.c:76:10: warning: format '%d' expects a matching 'int' argument [-Wformat=]
```

Ejecutamos para N=8 y N=11

```
2018-04-05 22:32:20  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1/src
→ ./listado1_omp 8
Tiempo(seg): 0.003934771         / 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) /

v1[0]: 477718400, v1[2147483593]: 0, v2[0]: 58, v2[0]: 2048343983, v3[0]: 8, v3[-1777136128]: 0
```

```
 2018-04-05 22:32:23  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1/src
○ → ./listado1_omp 11
Tiempo(seg): 0.000584452        / 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) /

v1[0]: -2002069632, v1[2147483590]: 0, v2[0]: 61, v2[0]: 141791370, v3[0]: 11, v3[-181385728]: 0
```

Desde sftp subo al front-end el ejecutable:

```
sftp> lcd Escritorio/University\ stuff/2º/2º\ Cuatrimestre/AC/Prácticas/BP1/src/

sftp> lls
barrier.c     listado1      listado1_omp.c  single.c
bucle_for.c   listado1.c    listado1.s      single_modificado_1.c
hello.c       listado1_omp  sections.c      single_modificado_2.c
sftp> put lis
listado1        listado1.c      listado1.s      listado1_omp    listado1_omp.c

sftp> put listado1_
listado1_omp    listado1_omp.c
sftp> put listado1_omp
Uploading listado1_omp to /home/E2estudiante10/listado1_omp
listado1_omp                                  100%   13KB  13.1KB/s   00:00
```

Ejecutamos ahora desde atcgrid:

```
[ElenaMereloMolina E2estudiante10@atcgrid:~] 2018-04-05 jueves
$echo './listado1_omp 8' | qsub -q ac
72104.atcgrid
[ElenaMereloMolina E2estudiante10@atcgrid:~] 2018-04-05 jueves
$echo './listado1_omp 11' | qsub -q ac
72105.atcgrid
```

Resultado para N= 8:

```
[ElenaMereloMolina E2estudiante10@atcgrid:~] 2018-04-05 jueves
$cat STDIN.o72104
Tiempo(seg): 0.003828537        / 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) /

v1[0]: -783898816, v1[0]: 4197656, v2[0]: 1, v2[-783904072]: 106426814, v3[0]: 8, v3[1807015936]: 0
```

Y para N= 11:

```
[ElenaMereloMolina E2estudiante10@atcgrid:~] 2018-04-05 jueves
$cat STDIN.o72105
Tiempo(seg): 0.003898053        / 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) /

v1[0]: -1547360448, v1[0]: 4197656, v2[0]: 1, v2[-1547365704]: 367313450, v3[0]: 11, v3[980654080]:
 0[ElenaMereloMolina E2estudiante10@atcgrid:~] 2018-04-05 jueves
```

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

```
/* listado1_sections.c
  Suma de dos vectores utilizando la directiva sections:
  v3 = v1 + v2
  Para compilar usar (-lrt: real time library):
    gcc  -O2 listado1_sections.c -o listado1_sections –lrt
    gcc  -O2 –S listado1_sections.c –lrt  //para generar el código ensamblador
  Para ejecutar: ./listado1_sections tamanio
*/


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


//#define PRINTF_ALL


#define VECTOR_GLOBAL
#define MAX 67108865
double v1[MAX], v2[MAX], v3[MAX];


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


  int i, tope;
  double t_inicial, t_final;   //para tiempo de ejecución


  //Leer argumento de entrada (nº de componentes del vector)
  if (argc<2){
    printf("Faltan nº componentes del vector \n");
    exit(-1);
  }


  unsigned int N = atoi(argv[1]);   // Máximo N =2^32  -1=4294967295
(sizeof(unsigned int) = 4 B)
  if (N>MAX)
    N=MAX;


  //Inicializar vectores
  #pragma omp parallel sections
```

```
  {
    #pragma omp section
    {
      tope = N/4;
      for  (i=0; i<tope; i++) {
        v1[i] = N*0.1 + i*0.1;
        v2[i] = N*0.1 - i*0.1;
      }
    }
    #pragma omp section
    {
      tope = N/2;
      for  (i=N/4; i<tope; i++) {
        v1[i] = N*0.1 + i*0.1;
        v2[i] = N*0.1 - i*0.1;
      }
    }
    #pragma omp section
    {
      tope = 3*N/4;
      for  (i=N/2; i<tope; i++) {
        v1[i] = N*0.1 + i*0.1;
        v2[i] = N*0.1 - i*0.1;
      }
    }
    #pragma omp section
    {
      tope = N;
      for  (i=3*N/4; i<tope; i++) {
        v1[i] = N*0.1 + i*0.1;
        v2[i] = N*0.1 - i*0.1;
      }
    }
  }


  t_inicial = omp_get_wtime();


  //Calcular suma de vectores
  #pragma omp parallel sections
  {
    #pragma omp section
    {
      tope = N/4;
      for  (i=0; i<tope; i++) {
        v3[i] = v1[i] + v2[i];
      }
    }
    #pragma omp section
    {
      tope = N/2;
      for  (i=N/4; i<tope; i++) {
        v3[i] = v1[i] + v2[i];
      }
    }
    #pragma omp section
    {
      tope = 3*N/4;
      for  (i=N/2; i<tope; i++) {
        v3[i] = v1[i] + v2[i];
      }
```

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



  t_final = omp_get_wtime() - t_inicial;


  //Imprimir resultado de la suma y el tiempo de ejecución
  #ifdef PRINTF_ALL
    printf("Tiempo(seg.):%11.9f \t/ Tamaño Vectores:%u \t/", t_final, N);
    for (i=0; i<N; i++)
      printf("v3[%d] = %11.9f\n", i, v3[i]);
  #else
    printf("Tiempo(seg.):%11.9f \t/ Tamaño Vectores:%u \t/"
        "V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) / /"
        "V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) / \n",
        t_final, N, v1[0], v2[0], v3[0], N-1, N-1, N-1, v1[N-1], v2[N-1],
v3[N-1]);
  #endif


  return  0;
}
```

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

Compilación y ejecución para n= 8

```
 2018-04-05 22:41:27  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1/src
○→gcc -O2 -fopenmp listado1_sections.c -o listado1_sections -lrt

 2018-04-05 22:42:33  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1/src
○→./listado1_sections 8
Tiempo(seg.):0.002399460        / Tamaño Vectores:8     /v3[0] = 1.600000000
v3[1] = 1.600000000
v3[2] = 1.600000000
v3[3] = 1.600000000
v3[4] = 1.600000000
v3[5] = 1.600000000
v3[6] = 1.600000000
v3[7] = 1.600000000
```

Para N=11:

```
 2018-04-05 22:42:34  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1/src
○→./listado1_sections 11
Tiempo(seg.):0.002992762        / Tamaño Vectores:11    /v3[0] = 2.200000000
v3[1] = 2.200000000
v3[2] = 2.200000000
v3[3] = 2.200000000
v3[4] = 2.200000000
v3[5] = 2.200000000
v3[6] = 2.200000000
v3[7] = 2.200000000
v3[8] = 2.200000000
v3[9] = 2.200000000
v3[10] = 2.200000000
```

Lo subimos ahora al front-end:

```
sftp> put listado1_sections
Uploading listado1_sections to /home/E2estudiante10/listado1_sections
listado1_sections                             100%   13KB   1.6KB/s   00:08
```

Y ejecutamos desde atcgrid:

```
ElenaMereloMolina E2estudiante10@atcgrid:~] 2018-04-05 jueves
$echo './listado1_sections 8' | qsub -q ac
72110.atcgrid
[ElenaMereloMolina E2estudiante10@atcgrid:~] 2018-04-05 jueves
$echo './listado1_sections 11' | qsub -q ac
72111.atcgrid
```

Obteniendo:

```
[ElenaMereloMolina E2estudiante10@atcgrid:~] 2018-04-05 jueves
$cat STDIN.o72110
Tiempo(seg.):0.003938973        / Tamaño Vectores:8     /v3[0] = 1.600000000
v3[1] = 1.600000000
v3[2] = 1.600000000
v3[3] = 1.600000000
v3[4] = 1.600000000
v3[5] = 1.600000000
v3[6] = 1.600000000
v3[7] = 1.600000000
[ElenaMereloMolina E2estudiante10@atcgrid:~] 2018-04-05 jueves
$cat STDIN.o72111
Tiempo(seg.):0.006679346        / Tamaño Vectores:11    /v3[0] = 2.200000000
v3[1] = 2.200000000
v3[2] = 2.200000000
v3[3] = 2.200000000
v3[4] = 2.200000000
v3[5] = 2.200000000
v3[6] = 2.200000000
v3[7] = 2.200000000
v3[8] = 2.200000000
v3[9] = 2.200000000
v3[10] = 2.200000000
```

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:

   Para el ejercicio 7 podemos usar tantas hebras (y cores) como tenga el procesador dado que se realiza una distribución dinámica de las iteraciones del bucle, esto es, en atcgrid utilizaríamos sus 24 hebras (12 cores) mientras que en mi portátil, que tiene 4 cpus, también aprovecharía al máximo dichos recursos.

   No obstante, la distribución que se realiza en el ejercicio 8 es estática, independientemente del número de hebras de las que disponga la máquina en la que se ejecuta el programa, solo podremos sacar partido a, como máximo, 4 de dichas hebras al haber 4 secciones. Así, en el caso de mi portátil se saca máximo partido a sus recursos al utilizar sus cuatro hebras, pero en atcgrid el tiempo de ejecución será mayor para grandes tamaño del vector, al tener hebras desocupadas, se usan únicamente 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:

Tras compilar y generar el ejecutable de listado1_omp.c, programa correspondiente al ejercicio 7, ejecutamos el script ejer7_tabla2_local.sh, obteniendo:

```
2018-04-17 20:57:06  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1/src
→ g++ -lrt -fopenmp -O2 listado1_omp.c -o ../bin/listado1_omp
```

```
2018-04-17 21:04:52  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1/scripts
→ bash ejer7_tabla2_local.sh
Tiempo(seg): 0.000137662          / 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): 0.000270505          / 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): 0.000887913          / 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): 0.000392618          / 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): 0.000628411          / 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): 0.001197741          / 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): 0.001811933          / 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): 0.004147278          / 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): 0.006101044          / 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): 0.012220636          / 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): 0.024242750          / 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): 0.048166828          / 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): 0.146182268          / 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) /
```

Metemos estos datos en la tabla y hacemos lo mismo para listado1_sections.c y luego listado1.c:

```
2018-04-17 22:00:51  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1/src
→ g++ -lrt -fopenmp -O2 listado1_sections.c -o ../bin/listado1_sections
```

```
2018-04-17 22:01:35  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1/scripts
→ bash ejer8_tabla2_local.sh
Tiempo(seg.):0.000161355        / 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.):0.000351815        / 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.):0.022177524        / 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.):0.000210604        / 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.):0.000427867        / 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.):0.001025107        / 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.):0.001606143        / 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.):0.003427715        / 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.):0.006736259        / 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.):0.012537288        / 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.):0.027084534        / 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.):0.048912363        / 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.):0.100487461        / 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) /
```

```
2018-04-17 22:04:27  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1/src
→ g++ -lrt -fopenmp -O2 listado1.c -o ../bin/listado1
```

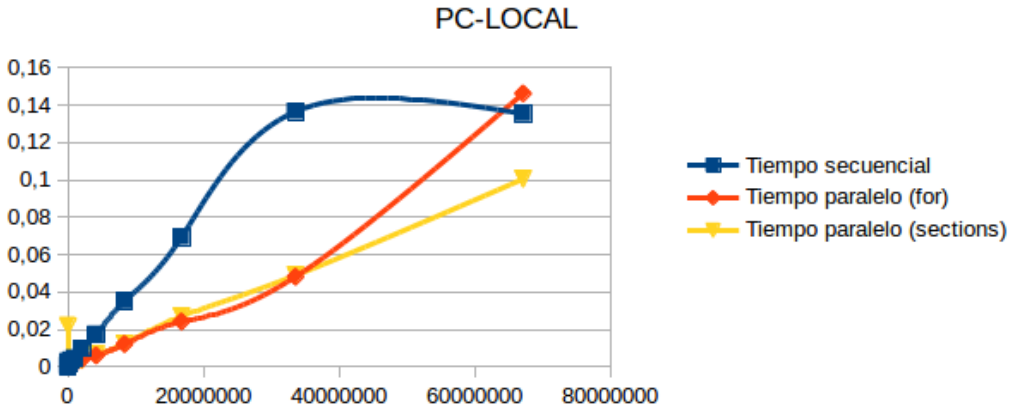
```
2018-04-17 22:05:32  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestre/AC/Prácticas/BP1/scripts
→ bash listado1_secuencial.sh
Tiempo(seg.): 0.000359066       /Tamaño vectores:16384  /V1[0] + V2[0]= V3[0](1638.400000+1638.400000=3276
.800000)// V1[16383] + V2[16383]= V3[16383](3276.700000+0.100000=3276.800000)/
Tiempo(seg.): 0.000739514       /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.): 0.001499248       /Tamaño vectores:65536  /V1[0] + V2[0]= V3[0](6553.600000+6553.600000=1310
7.200000)// V1[65535] + V2[65535]= V3[65535](13107.100000+0.100000=13107.200000)/
Tiempo(seg.): 0.002878951       /Tamaño vectores:131072 /V1[0] + V2[0]= V3[0](13107.200000+13107.200000=26
214.400000)// V1[131071] + V2[131071]= V3[131071](26214.300000+0.100000=26214.400000)/
Tiempo(seg.): 0.002230084       /Tamaño vectores:262144 /V1[0] + V2[0]= V3[0](26214.400000+26214.400000=52
428.800000)// V1[262143] + V2[262143]= V3[262143](52428.700000+0.100000=52428.800000)/
Tiempo(seg.): 0.003544448       /Tamaño vectores:524288 /V1[0] + V2[0]= V3[0](52428.800000+52428.800000=10
4857.600000)// V1[524287] + V2[524287]= V3[524287](104857.500000+0.100000=104857.600000)/
Tiempo(seg.): 0.004648286       /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.): 0.009792565       /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.): 0.017361179       /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.): 0.035143529       /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.): 0.069365449       /Tamaño vectores:16777216       /V1[0] + V2[0]= V3[0](1677721.600000+16777
21.600000=3355443.200000)// V1[16777215] + V2[16777215]= V3[16777215](3355443.100000+0.100000=3355443.2000
00)/
Tiempo(seg.): 0.136505656       /Tamaño vectores:33554432       /V1[0] + V2[0]= V3[0](3355443.200000+33554
43.200000=6710886.400000)// V1[33554431] + V2[33554431]= V3[33554431](6710886.300000+0.100000=6710886.4000
00)/
Tiempo(seg.): 0.135385132       /Tamaño vectores:33554432       /V1[0] + V2[0]= V3[0](3355443.200000+33554
43.200000=6710886.400000)// V1[33554431] + V2[33554431]= V3[33554431](6710886.300000+0.100000=6710886.4000
00)/
```

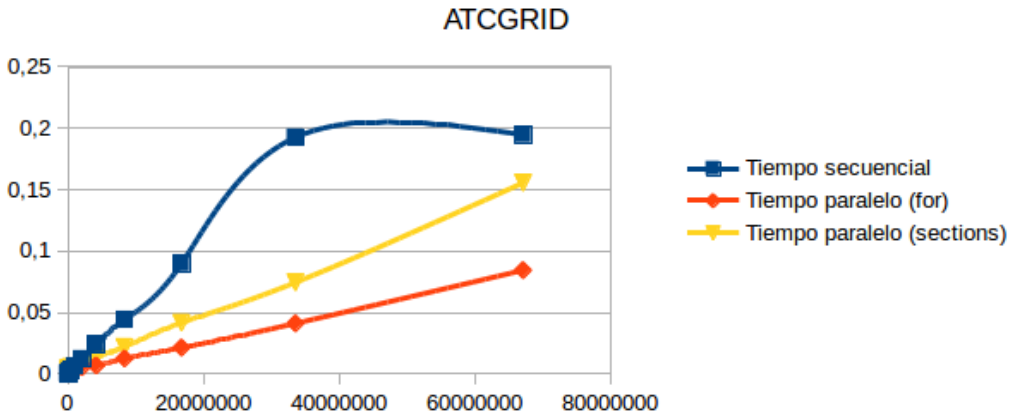
Subimos al frontend los scripts y ejecutables de éste y el siguiente ejercicio, y ejecutamos los scripts, obteniendo los correspondientes tiempos de ejecución, que colocamos en la tabla.

```
[ElenaMereloMolina E2estudiante10@atcgrid:~] 2018-04-17 martes
$ls
atcgrid_listado1_secuencial.sh       ejer8_tabla2_local.sh
atcgrid_listado1_secuencial_TIME.sh  listado1
ejer7_tabla2_atcgrid.sh              listado1_omp
ejer7_tabla2_local.sh                listado1_sections
ejer8_tabla2_atcgrid.sh              listado1_secuencial.sh
[ElenaMereloMolina E2estudiante10@atcgrid:~] 2018-04-17 martes
$qsub atcgrid_listado1_secuencial.sh -q ac
74295.atcgrid
```

ATCGRID ~ 12 threads/core			
N.º componentes	Tiempo secuencial	Tiempo paralelo (for)	Tiempo paralelo (sections)
16384	0.000111504	0.003935250	0.004126758
32768	0.000211555	0.004400079	0.003768297
65536	0.000426327	0.004194752	0.004180857
131072	0.000857215	0.004061962	0.003944567
262144	0.001689410	0.004388661	0.005055143
524288	0.003388732	0.004916015	0.005610716
1048576	0.006409406	0.004774413	0.005053353
2097152	0.012103938	0.005341516	0.007893194
4194304	0.023849091	0.006769757	0.013809685
8388608	0.044213675	0.012537950	0.022073274
16777216	0.089785193	0.021363975	0.041752336
33554432	0.192617632	0.041071397	0.074396430
67108864	0.194785822	0.084381192	0.155626928

Y sus gráficos:





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:

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	0,001	0,001	0,002	0,012	0,174	0,001	0,175
32768	0,001	0,001	0,001	0,002	0,01	0,156	0,004	0,16
65536	0,002	0,001	0,002	0,003	0,011	0,172	0,004	0,176
131072	0,003	0,001	0,003	0,004	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