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): Miguel Angel Robles Urquiza
Grupo de prácticas: A1
Fecha de entrega: 30/3/2017
Fecha evaluación en clase: 31/03/2017

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: código fuente `bucle-forModificado.c`

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
  #include <omp.h>
#else
  #define omp_get_thread_num() 0
  #define omp_get_num_threads() 1
#endif


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
  or (i=0; i<n; i++)
    printf("thread %d ejecuta la iteración %d del bucle\n",omp_get_thread_num(),i);


 return(0);
}
```

RESPUESTA: código fuente `sectionsModificado.c`

```
#include <stdio.h>
#ifdef _OPENMP
 #include <omp.h>
#else
 #define omp_get_thread_num() 0
 #define omp_get_num_threads() 1
#endif
void funcA(){
 printf("En funcA: esta sección la ejecuta el thread %d\n",  omp_get_thread_num());
 }
void funcB(){
 printf("En funcB: esta sección la ejecuta el thread %d\n", omp_get_thread_num());
}
```

```
main(){
 #pragma omp parallel sections{
   #pragma omp section
    (void) funcA();
   #pragma omp section
    (void) funcB();
 }
}
```

2. Imprimir los resultados del programa `single.c` usando una directiva `single` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `single` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `single`. Incorpore en su cuaderno de trabajo el código fuente y volcados de pantalla con los resultados de ejecución obtenidos.

   RESPUESTA: código fuente `singleModificado.c`

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

CAPTURAS DE PANTALLA:

```
migue@RoblesPC:~/Dropbox/2017DGIIM/2ºCuatri/AC/BP1$ ./single
Introduce valor de inicialización a: 5
Single ejecutada por el thread 2
Depués de la región 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

migue@RoblesPC:~/Dropbox/2017DGIIM/2ºCuatri/AC/BP1$ ./singleModificado
Introduce valor de inicialización a: 5
Single ejecutada por el thread 3
Resultados:
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 by thread 3
```

3. Imprimir los resultados del programa `single.c` usando una directiva `master` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `master` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `master`. Incorpore en su cuaderno el código fuente y volcados de pantalla con los resultados de ejecución obtenidos. ¿Qué diferencia observa con respecto a los resultados de ejecución del ejercicio anterior?

   RESPUESTA: código fuente `singleModificado2.c`

```
#include <stdio.h>
#ifdef _OPENMP
 #include <omp.h>
#else
 #define omp_get_thread_num() 0
 #define omp_get_num_threads() 1
#endif


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

```
      }
      #pragma omp for
        for (i=0; i<n; i++)
          b[i] = a;
      #pragma omp master{
       printf("Resultados:\n");
       for (i=0; i<n; i++)
        printf("b[%d] = %d\n",i,b[i]);
       printf("\n");
       printf("Master by thread %d\n", omp_get_thread_num());
    }
  }
}
```

CAPTURAS DE PANTALLA:

```
migue@RoblesPC:~/Dropbox/2017DGIIM/2ºCuatri/AC/BP1$ ./singleModificado2
Introduce valor de inicialización a: 5
Single ejecutada por el thread 3
Resultados:
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

Master by thread 0
```

RESPUESTA A LA PREGUNTA:

El single se ejecuta por una hebra diferente al master, ya que vemos que el single se ejecuta por la hebra 3 y el master se ejecuta por la hebra 0

4. ¿Por qué si se elimina directiva `barrier` en el ejemplo `master.c` la suma que se calcula e imprime no siempre es correcta? Responda razonadamente.

   RESPUESTA:

   Porque especificamos que va a ser la hebra master la que va a imprimir el resultado en pantalla. Si esta hebra es mas rápida, y no espera a las demás, mostrará el resultado de la suma antes de que hayan acabado las demás hebras, por lo tanto la suma no siempre será correcta.

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 el PC local, 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 mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

   CAPTURAS DE PANTALLA:

```
migue@RoblesPC:~/Dropbox/2017DGIIM/2ºCuatri/AC/BP1$ time ./SumaVectore
s 10000000
Tiempo(seg.):0.033247625          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=2000000.00
0000)

real    0m0.082s
user    0m0.052s
sys     0m0.028s
```

RESPUESTA A LA PREGUNTA:

La suma de los tiempos de CPU de usuario y del sistema es ligeramente menor al tiempo real porque el código es secuencial, es decir, solo estamos ejecutando 1 thread

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:

```
[E1estudiante17@atcgrid ~]$ echo './SumaVectores 10' | qsub -q ac
51856.atcgrid
[E1estudiante17@atcgrid ~]$ cat STDIN.o51856
Tiempo(seg.):0.000003350          Tamaño Vectores:10                V1[0]+V
2[0]=V3[0](1.000000+1.000000=2.000000)  V1[9]+V2[9]=V3[9](1.900000+0.10000
0=2.000000)
[E1estudiante17@atcgrid ~]$ echo './SumaVectores 10000000' | qsub -q ac
51858.atcgrid
[E1estudiante17@atcgrid ~]$ cat STDIN.o51858
Tiempo(seg.):0.075073123          Tamaño Vectores:10000000
 V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000)  V1[99999
99]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000)
[E1estudiante17@atcgrid ~]$
```

RESPUESTA:

Tamaño	10	10000000
Nº Instrucciones	6*10 = 60	6*10000000= 60000000
Operaciones en coma flotante	3*10 = 30	3*10000000 = 30000000
Tiempo(seg)	0,000003350	0,075073123
MIPS	60/(0,000003350*10^6)	60000000/(0,075073123*10^6)
MFLOPS	30/(0,000003350*10^6)	30000000/(0,075073123*10^6)

RESPUESTA:

código ensamblador generado de la parte de la suma de vectores

```
.L5:
            movsd       (%r12,%rax), %xmm0
            addsd       0(%rbp,%rax), %xmm0
            movsd       %xmm0, 0(%r13,%rax)
            addq        $8, %rax
            cmpq        %rax, %rbx
            jne         .L5
```

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: código fuente implementado

```
/* SumaVectoresC_7.c
   Suma de dos vectores: v3 = v1 + v2


   Para compilar usar (-lrt: real time library):
              gcc -O2 -lrt -fopenmp SumaVectoresC_7.c -o SumaVectoresC_7
   gcc -O2 –S SumaVectoresC_7.c –lrt
   //para generar el código ensamblador


   Para ejecutar use: SumaVectoresC_7 longitud
*/
#include <stdlib.h> // biblioteca con 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 <omp.h>


//#define PRINTF_ALL // comentar para quitar el printf ...
                     // que imprime todos los componentes


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


  int i;


  double cgt1,cgt2;
             double ncgt; //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)


  double *v1, *v2, *v3;
  v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño en
bytes
  v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente
malloc devuelve NULL
  v3 = (double*) malloc(N*sizeof(double));
  if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
    printf("Error en la reserva de espacio para los vectores\n");
    exit(-2);
  }
             #pragma omp parallel
             {
                          #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 omp single
                                        {
                                                    cgt1 = omp_get_wtime();
                                        }
                                        #pragma omp for
                                        for(i=0; i<N; i++){
                                        v3[i] = v1[i] + v2[i];
                                                    }
                                        #pragma omp single
                                        {
                                                    cgt2 = omp_get_wtime();
                                        }
                          }
                                                    ncgt = cgt2-cgt1;



//Imprimir resultado de la suma y el tiempo de ejecución
#ifdef PRINTF_ALL
  printf("Tiempo(seg.):%11.9f\t  Tamaño Vectores:%u\n",ncgt,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", ncgt,N,v1[0],v2[0],v3[0],N-1,N-
1,N-1,v1[N-1],v2[N-1],v3[N-1]);
#endif
```

```
  free(v1); // libera el espacio reservado para v1
  free(v2); // libera el espacio reservado para v2
  free(v3); // libera el espacio reservado para v3


  return 0;
}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
migue@RoblesPC:~/GIT/AC/BP1$ ./SumaVectoresC_7 8
Tiempo(seg.):0.000003709          Tamaño Vectores:8
V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000)  V1[7]+V2[7]=V3[7](1.
500000+0.100000=1.600000)
migue@RoblesPC:~/GIT/AC/BP1$ ./SumaVectoresC_7 11
Tiempo(seg.):0.000002265          Tamaño Vectores:11
V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000)  V1[10]+V2[10]=V3[10]
(2.100000+0.100000=2.200000)
```

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

```
/* SumaVectoresC_8.c
   Suma de dos vectores: v3 = v1 + v2


   Para compilar usar (-lrt: real time library):
   gcc -O2 -lrt -fopenmp SumaVectoresC_8.c -o SumaVectoresC_8
   gcc -O2 –S SumaVectoresC_8.c –lrt
   //para generar el código ensamblador


   Para ejecutar use: SumaVectoresC_8 longitud
*/
#include <stdlib.h> // biblioteca con 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 <omp.h>


//#define PRINTF_ALL // comentar para quitar el printf ...
                     // que imprime todos los componentes


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

```
  int i;


  double cgt1,cgt2; double ncgt; //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)


  double *v1, *v2, *v3;
  v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño en
bytes
  v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente
malloc devuelve NULL
  v3 = (double*) malloc(N*sizeof(double));
  if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
    printf("Error en la reserva de espacio para los vectores\n");
    exit(-2);
  }
  #pragma omp parallel private(i)
            {
                        #pragma omp sections
                        {
                                    #pragma omp section
                                    for(i=0; i<N/4; i++)
                                    {
                                                v1[i] = N*0.1+i*0.1;
                                                v2[i] = N*0.1-i*0.1;
                                    }


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


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


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

```
                                        }
                            }
                                        cgt1 = omp_get_wtime();


    #pragma omp sections
                            {
                                        #pragma omp section
                                        for(i=0; i<N/4; i++)
                                                    v3[i] = v1[i] + v2[i];


                                        #pragma omp section
                                        for(i=N/4; i<N/2; i++)
                                                    v3[i] = v1[i] + v2[i];


                                        #pragma omp section
                                        for(i=N/2; i<3*N/4; i++)
                                                    v3[i] = v1[i] + v2[i];


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


                            }


                                        cgt2 = omp_get_wtime();
                            }
                                        ncgt = cgt2-cgt1;



//Imprimir resultado de la suma y el tiempo de ejecución
#ifdef PRINTF_ALL
  printf("Tiempo(seg.):%11.9f\t  Tamaño Vectores:%u\n",ncgt,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", ncgt,N,v1[0],v2[0],v3[0],N-1,N-
1,N-1,v1[N-1],v2[N-1],v3[N-1]);
#endif


  free(v1); // libera el espacio reservado para v1
  free(v2); // libera el espacio reservado para v2
  free(v3); // libera el espacio reservado para v3


  return 0;
}
```

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

```
migue@RoblesPC:~/GIT/AC/BP1$ ./SumaVectoresC_8 8
Tiempo(seg.):9878.996093971        Tamaño Vectores:8
V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000)  V1[7]+V2[7]=V3[7](1.
500000+0.100000=1.600000)
migue@RoblesPC:~/GIT/AC/BP1$ ./SumaVectoresC_8 11
Tiempo(seg.):9880.683988891        Tamaño Vectores:11
V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000)  V1[10]+V2[10]=V3[10]
(2.100000+0.100000=2.200000)
```

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:

   - No hemos definido la variable de entorno OMP_NUM_THREADS, asi que se utilizaran todos los cores/threads que tenga disponibles el PC para el ejercicio 7.
   - No hemos definido la variable de entorno OMP_NUM_THREADS, asi que se utilizaran todos los cores/threads que tenga disponibles el PC. En este ejrcicio, el 8, hemos dividido en secciones fijas, por lo que habrá hebras que no realicen ningun tranajo, asi que como maximo, el numero de cores/threads que tenga disponibles el PC

10. Rellenar una tabla como la Tabla 2 para atcgrid y otra para el PC local 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 en atcgrid código que imprima todos los componentes del resultado.

    RESPUESTA:

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

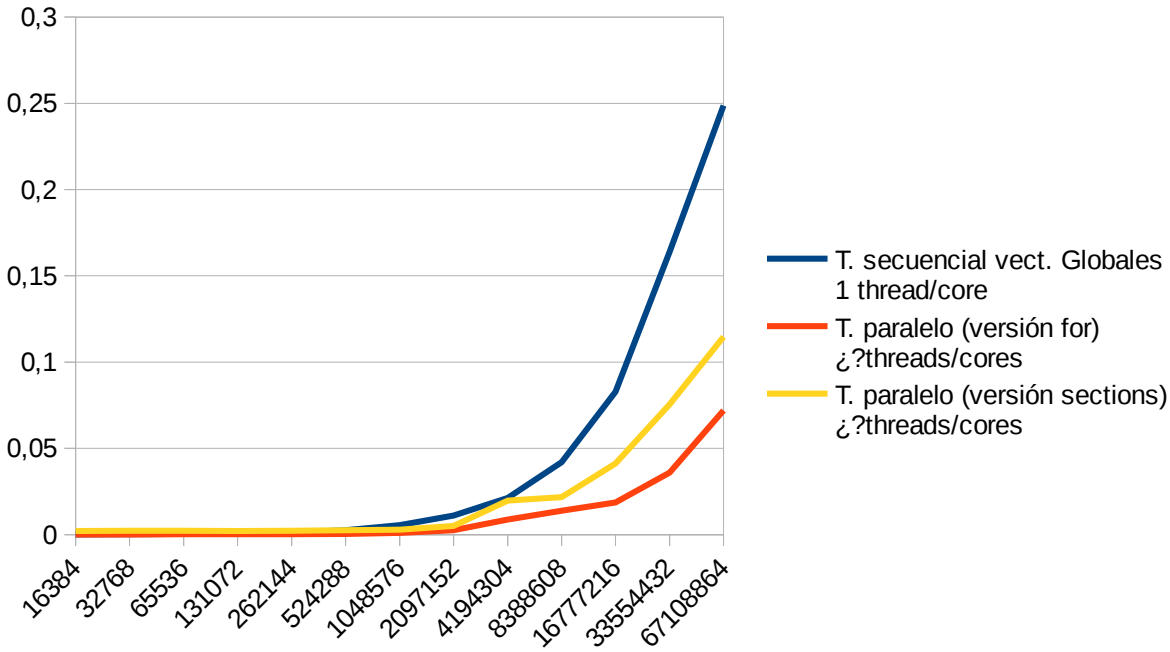
ATCGRID

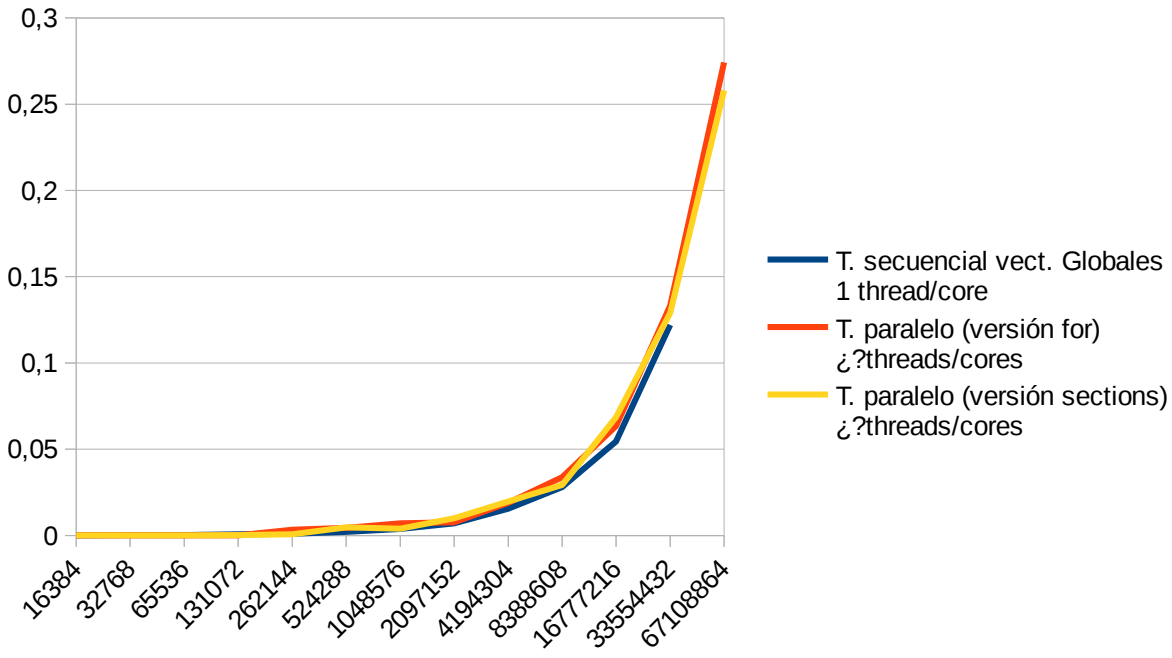
Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión `for`) ¿?threads/cores	T. paralelo (versión `sections`) ¿?threads/cores
16384	0,000099281	0,000008949	0,002154486
32768	0,000196134	0,000013547	0,002246585
65536	0,000400912	0,000199854	0,002288874
131072	0,000781347	0,000206725	0,002150713
262144	0,001225502	0,000238748	0,002212951
524288	0,002530856	0,000395950	0,002531035
1048576	0,005536854	0,000937210	0,002900945
2097152	0,011058754	0,002594899	0,004992671
4194304	0,021269760	0,008658936	0,019747402
8388608	0,041931747	0,013892073	0,021741901
16777216	0,082777000	0,018631654	0,041264853
33554432	0,164039175	0,036000643	0,075596185
67108864	0,248672493	0,072056994	0,114661349

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

PC LOCAL

Nº de Componentes	**T. secuencial vect. Globales 1 thread/core**	**T. paralelo (versión `for`) ¿?threads/cores**	**T. paralelo (versión `sections`) ¿?threads/cores**
16384	0,000159709	0,000020004	0,000004503
32768	0,000190783	0,000037861	0,000086327
65536	0,000272691	0,000078126	0,000088259
131072	0,000679270	0,000172879	0,000188877
262144	0,001005068	0,003380064	0,000630842
524288	0,002110556	0,004424162	0,004646026
1048576	0,003765257	0,007105729	0,003893605
2097152	0,007061106	0,007695018	0,009991919
4194304	0,015494183	0,018899974	0,019747402
8388608	0,028061611	0,033750317	0,029320982
16777216	0,054427364	0,063489113	0,068568945
33554432	0,122099354	0,133261708	0,129088576
67108864	0.242099354	0,274257563	0,258006070





11. Rellenar una tabla como la Tabla 3 para el PC local 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:El tiempo de CPU usado en el programa secuencial es menor que el tiempo real, ya que sólo se usa un procesador. En la versión paralela, sin embargo, el tiempo de CPU es mayor que el tiempo real. Esto es debido a que el tiempo de CPU es igual a la suma de los tiempos de cada núcleo, mientras que el tiempo real es el tiempo que realmente ha tardado el programa en ejecutarse.

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.

PC LOCAL

Nº de Componentes	**Tiempo secuencial vect. Globales 1 thread/core**			**Tiempo paralelo/versión `for` ¿? Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	0m0.004s	0m0.000s	0m0.000s	0m0.007s	0m0.012	0m0.000s
131072	0m0.005s	0m0.000s	0m0.000s	0m0.004s	0m0.008s	0m0.000s
262144	0m0.007s	0m0.004s	0m0.000s	0m0.007s	0m0.016s	0m0.000s
524288	0m0.011s	0m0.008s	0m0.000s	0m0.009s	0m0.012s	0m0.012s
1048576	0m0.016s	0m0.008s	0m0.004s	0m0.014s	0m0.032s	0m0.000s
2097152	0m0.027s	0m0.024s	0m0.000s	0m0.027s	0m0.064s	0m0.016s
4194304	0m0.035s	0m0.024s	0m0.008s	0m0.041s	0m0.104s	0m0.024s
8388608	0m0.071s	0m0.048s	0m0.020s	0m0.085s	0m0.180s	0m0.064s
16777216	0m0.124s	0m0.080s	0m0.040s	0m0.181s	0m0.348s	0m0.112s
33554432	0m0.254s	0m0.172s	0m0.080s	0m0.327s	0m0.708s	0m0.220s
67108864	0m0.258s	0m0.192s	0m0.064s	0m0.615s	0m1.584s	0m0.572s