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 David Villegas Yeguas
Grupo de prácticas y profesor de prácticas: B2
Fecha de entrega:
Fecha evaluación en clase:

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

Ejercicios basados en los ejemplos del seminario práctico

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

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

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

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

   int i, n =  9;
   if(argc < 2){
      fprintf(stderr,"\n[ERROR] - Falta nº iteraciones \n");
      exit(- 1);
   }

   n = atoi(argv[1 ]);

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


   return(0);

}
```

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

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

void funcA(){
   printf("En funcA: esta sección la ejecuta el 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();

   }
   return 0;
}

```

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

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

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

int main(){
   int n = 9, i, a, b[n];

   for (i = 0; i < n; i++) b[i] = -1;
   #pragma omp parallel
   {
      #pragma omp single
      {
         printf("Introduce valor de 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("Dentro de la region parallel, usando single:\n");
         for (i = 0; i < n; i++) printf("b[%d] = %d\t", i, b[i]);
         printf("\n");
      }
   }
}
```

CAPTURAS DE PANTALLA:

```
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer2] 2019-03-28 jueves
$gcc -O2 -fopenmp singleModificado.c -o singleModificado
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer2] 2019-03-28 jueves
$export OMP_DYNAMIC=FALSE
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer2] 2019-03-28 jueves
$export OMP_NUM_THREADS=4
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer2] 2019-03-28 jueves
$./singleModificado
Introduce valor de inicializacion a: 23
Single ejecutada por el thread 0
Dentro de la region parallel, usando single:
b[0] = 23       b[1] = 23       b[2] = 23       b[3] = 23       b[4] = 23       b[5] = 23       b[6] = 23       b[7] = 23       b[8] = 23
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer2] 2019-03-28 jueves
$
```

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

RESPUESTA: Captura que muestre el código fuente `singleModificado2.c`

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

int main(){
   int n = 9, i, a, b[n];

   for (i = 0; i < n; i++) b[i] = -1;
   #pragma omp parallel
   {
      #pragma omp single
      {
         printf("Introduce valor de 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 barrier
      #pragma omp master
      {
         printf("Dentro de la region parallel, usando master, ejecutado por \
               el thread: %d\n", omp_get_thread_num());
         for (i = 0; i < n; i++) printf("b[%d] = %d\t", i, b[i]);
         printf("\n");
      }
   }
}
```

CAPTURAS DE PANTALLA:

```
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer3] 2019-03-28 jueves
$gcc -O2 -fopenmp singleModificado2.c -o singleModificado2
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer3] 2019-03-28 jueves
$export OMP_DYNAMIC=FALSE
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer3] 2019-03-28 jueves
$export OMP_NUM_THREADS=4
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer3] 2019-03-28 jueves
$./singleModificado2
Introduce valor de inicializacion a: 12
Single ejecutada por el thread 0
Dentro de la region parallel, usando master, ejecutado por                el thread: 0
b[0] = 12       b[1] = 12       b[2] = 12       b[3] = 12       b[4] = 12       b[5] = 12       b[6] = 12       b[7] = 12       b[8] = 12
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer3] 2019-03-28 jueves
$./singleModificado2
Introduce valor de inicializacion a: 32
Single ejecutada por el thread 0
Dentro de la region parallel, usando master, ejecutado por                el thread: 0
b[0] = 32       b[1] = 32       b[2] = 32       b[3] = 32       b[4] = 32       b[5] = 32       b[6] = 32       b[7] = 32       b[8] = 32
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer3] 2019-03-28 jueves
$
```

RESPUESTA A LA PREGUNTA:

La seccion master es ejecuta siempre el thread 0, debido a que es la thread master. A

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 la directiva master no tiene una barrera implicita, al principio, ni la instruccion atomic al final, luego puede ser que la hebra master se adelante a las demas hebras, e imprima algo por pantalla que las demas hebras todavia no han ejecutado

---

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:

```
sftp> put ejer5/SumaVectoresC.c ejer5/
Uploading ejer5/SumaVectoresC.c to /home/B1estudiante25/BP1/ejer5/SumaVectoresC.c
ejer5/SumaVectoresC.c                                                                100% 3305   536.1KB/s   00:00
sftp> put ejer5/script_sumavectorc.sh ejer5/
Uploading ejer5/script_sumavectorc.sh to /home/B1estudiante25/BP1/ejer5/script_sumavectorc.sh
ejer5/script_sumavectorc.sh                                                          100%  769    63.2KB/s   00:00
sftp>
```

```
#!/bin/bash
#Todos los scripts que se hagan para atcgrid deben incluir lo siguiente:
#Se asigna al trabajo el nombre SumaVectoresC_vglobales
#PBS -N SumaVectoresC_vglobales
#Se asigna al trabajo la cola ac
#PBS -q ac
#Se imprime información del trabajo usando variables de entorno de PBS
echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
echo "Id. del trabajo: $PBS_JOBID"
echo "Nombre  del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"
echo "Nodos asignados al trabajo:"
cat $PBS_NODEFILE
# FIN del trozo que deben incluir todos los scripts


export OMP_DYNAMIC=FALSE
export OMP_NUM_THREADS=4

$PBS_O_WORKDIR/SumaVectoresC 10000000

```

```
B1estudiante25@atcgrid:~/BP1/ejer5

antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer5 ×   B1estudiante25@atcgrid:~/BP1/ejer5 ×   sftp B1estudiante25@atcgrid.ugr.es ×

[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer5] 2019-04-02 martes
$gcc -O2 SumaVectoresC.c -o SumaVectoresC -lrt
[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer5] 2019-04-02 martes
$cat script_sumavectorc.sh
#!/bin/bash
#Todos los scripts que se hagan para atcgrid deben incluir lo siguiente:
#Se asigna al trabajo el nombre SumaVectoresC_vglobales
#PBS -N SumaVectoresC_vglobales
#Se asigna al trabajo la cola ac
#PBS -q ac
#Se imprime información del trabajo usando variables de entorno de PBS
echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
echo "Id. del trabajo: $PBS_JOBID"
echo "Nombre  del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"
echo "Nodos asignados al trabajo:"
cat $PBS_NODEFILE
# FIN del trozo que deben incluir todos los scripts


export OMP_DYNAMIC=FALSE
export OMP_NUM_THREADS=4

time $PBS_O_WORKDIR/SumaVectoresC 10000000
[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer5] 2019-04-02 martes
$qsub script_sumavectorc.sh -q ac
17372.atcgrid
[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer5] 2019-04-02 martes
$ls -l
total 28
-rwxr-xr-x 1 B1estudiante25 B1estudiante25  774 abr  1 15:29 script_sumavectorc.sh
-rwxrwxr-x 1 B1estudiante25 B1estudiante25 8512 abr  2 15:06 SumaVectoresC
-rw-r--r-- 1 B1estudiante25 B1estudiante25 3305 abr  1 15:28 SumaVectoresC.c
-rw------- 1 B1estudiante25 B1estudiante25   42 abr  2 15:04 SumaVectoresC_vglobales.e17372
-rw------- 1 B1estudiante25 B1estudiante25  523 abr  2 15:04 SumaVectoresC_vglobales.o17372
[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer5] 2019-04-02 martes
$cat SumaVectoresC_vglobales.e17372

real    0m0.113s
user    0m0.054s
sys     0m0.056s
[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer5] 2019-04-02 martes
$cat SumaVectoresC_vglobales.o17372
Id. usuario del trabajo: B1estudiante25
Id. del trabajo: 17372.atcgrid
Nombre  del trabajo especificado por usuario: SumaVectoresC_vglobales
Nodo que ejecuta qsub: atcgrid
Directorio en el que se ha ejecutado qsub: /home/B1estudiante25/BP1/ejer5
Cola: ac
Nodos asignados al trabajo:
atcgrid3
Tama□ Vectores:10000000 (4 B)
Tiempo:0.040361605      / Tama□ Vectores:10000000     / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /
[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer5] 2019-04-02 martes
```

La suma es menor, debido a que tiene que hay tiempos de espera, en los que el programa no se ejecuta en CPU

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

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

```
[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer6] 2019-04-02 martes
$gcc -O2 -S SumaVectoresC.c -lrt
[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer6] 2019-04-02 martes
$gcc -O2 SumaVectoresC.c -o SumaVectoresC -lrt
[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer6] 2019-04-02 martes
$
```

```
[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer6] 2019-04-02 martes
$cat script_sumavectorc.sh
#!/bin/bash
#Todos los scripts que se hagan para atcgrid deben incluir lo siguiente:
#Se asigna al trabajo el nombre SumaVectoresC_vglobales
#PBS -N SumaVectoresC_vglobales
#Se asigna al trabajo la cola ac
#PBS -q ac
#Se imprime información del trabajo usando variables de entorno de PBS
echo "Id. usuario del trabajo: $PBS_O_LOGNAME"
echo "Id. del trabajo: $PBS_JOBID"
echo "Nombre  del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"
echo "Nodos asignados al trabajo:"
cat $PBS_NODEFILE
# FIN del trozo que deben incluir todos los scripts

export OMP_DYNAMIC=FALSE
export OMP_NUM_THREADS=4

time $PBS_O_WORKDIR/SumaVectoresC 10
time $PBS_O_WORKDIR/SumaVectoresC 10000000
[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer6] 2019-04-02 martes
$
```

```
[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer6] 2019-04-02 martes
$qsub script_sumavectorc.sh -q ac
17384.atcgrid
[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer6] 2019-04-02 martes
$ls -l
total 36
-rwxr-xr-x 1 B1estudiante25 B1estudiante25  810 abr  2 15:43 script_sumavectorc.sh
-rwxrwxr-x 1 B1estudiante25 B1estudiante25 8512 abr  2 15:28 SumaVectoresC
-rw-r--r-- 1 B1estudiante25 B1estudiante25 3305 abr  2 15:26 SumaVectoresC.c
-rw-rw-r-- 1 B1estudiante25 B1estudiante25 4389 abr  2 15:27 SumaVectoresC.s
-rw------- 1 B1estudiante25 B1estudiante25   84 abr  2 16:00 SumaVectoresC_vglobales.e17384
-rw------- 1 B1estudiante25 B1estudiante25  693 abr  2 16:00 SumaVectoresC_vglobales.o17384
[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer6] 2019-04-02 martes
$cat SumaVectoresC_vglobales.e17384

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

real    0m0.110s
user    0m0.066s
sys     0m0.044s
[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer6] 2019-04-02 martes
$cat SumaVectoresC_vglobales.o17384
Id. usuario del trabajo: B1estudiante25
Id. del trabajo: 17384.atcgrid
Nombre  del trabajo especificado por usuario: SumaVectoresC_vglobales
Nodo que ejecuta qsub: atcgrid
Directorio en el que se ha ejecutado qsub: /home/B1estudiante25/BP1/ejer6
Cola: ac
Nodos asignados al trabajo:
atcgrid1
Tama□ Vectores:10 (4 B)
Tiempo:0.000000202       / Tama□ 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) /
Tama□ Vectores:10000000 (4 B)
Tiempo:0.040979990       / Tama□ Vectores:10000000     / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /
[AntonioDavidVillegasYeguas B1estudiante25@atcgrid:~/BP1/ejer6] 2019-04-02 martes
$
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Vemos en ensamblador que el bucle esta compuesto por 9 instrucciones, de las cuales 3 de ellas son instrucciones Float (movsd, addsd movsd)

Para el calculo de los MIPS seria MIPS = N_intrucciones / (T_CPU * 10^6) = 9 / (10^6)

MIPS = (9 * 10000000) / (0,04097999 * 10^6) = 2.196,1938

Para el calculo de los MFLOPS seria MFLOPS = N_intrucciones_float / (T_CPU * 10^6)

MIPS = (3 * 10000000) / (0,04097999 * 10^6) = 732,064

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

```
        call    clock_gettime
        xorl    %eax, %eax
        .p2align 4,,10
        .p2align 3
.L9:
        movsd   0(%rbp,%rax), %xmm0
        addsd   (%r14,%rax), %xmm0
        movsd   %xmm0, (%r12,%rax)
        addq    $8, %rax
        cmpq    %r15, %rax
        jne     .L9
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime
        movq    24(%rsp), %rax
```

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

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

 Para compilar usar (-lrt: real time library, es posible que no sea necesario usar -lrt):
   gcc  -O2  SumaVectores.c -o SumaVectores -lrt
   gcc  -O2 -S SumaVectores.c -lrt

 Para ejecutar use: SumaVectoresC longitud

*/

#include <stdlib.h>  // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h>   // biblioteca donde se encuentra la funci□n printf()
#include <time.h> // biblioteca donde se encuentra la funci□n clock_gettime()

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

//S□lo puede estar definida una de las tres constantes VECTOR_ (s□lo uno de los ...
//tres defines siguientes puede estar descomentado):
//#define VECTOR_LOCAL  // descomentar para que los vectores sean variables ...
          // locales (si se supera el tama□o de la pila se ...
          // generar□ el error "Violaci□n de Segmento")
//#define VECTOR_GLOBAL // descomentar para que los vectores sean variables ...
          // globales (su longitud no estar□ limitada por el ...
          // tama□o de la pila del programa)
#define VECTOR_DYNAMIC  // descomentar para que los vectores sean variables ...
          // din□micas (memoria reutilizable durante la ejecuci□n)

#ifdef VECTOR_GLOBAL
#define MAX 33554432 //=2^25

double v1[MAX], v2[MAX], v3[MAX];
#endif
int main(int argc, char** argv){

  struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecuci□n

  //Leer argumento de entrada (n□ de componentes del vector)
  if (argc<2){
    printf("Faltan n□ componentes del vector\n");
    exit(-1);
  }

  unsigned int N = atoi(argv[1]);   // M□ximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
  printf("Tama□o Vectores:%u (%u B)\n",N, sizeof(unsigned int));


  #ifdef VECTOR_DYNAMIC
  double *v1, *v2, *v3;
  v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tama□o en bytes
  v2 = (double*) malloc(N*sizeof(double));
  v3 = (double*) malloc(N*sizeof(double));
  if ((v1 == NULL) || (v2 == NULL) || (v3 == NULL)) {
    printf("No hay suficiente espacio para los vectores \n");
    exit(-2);
  }
  #endif


  //Inicializar vectores
  #pragma omp parallel for
     for(int i=0; i<N; i++){
       v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
     }

  double start = omp_get_wtime();
  //Calcular suma de vectores

  #pragma omp parallel for
     for(int i=0; i<N; i++){
       v3[i] = v1[i] + v2[i];
     }

  double end = omp_get_wtime();

  double time = end - start;

  //Imprimir resultado de la suma y el tiempo de ejecuci□n
  if (N<10) {
  printf("Tiempo:%11.9f\t / Tama□o Vectores:%u\n",time,N);
  for(int 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:%11.9f\t / Tama□o 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",
           time,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);


  #ifdef VECTOR_DYNAMIC
  free(v1); // libera el espacio reservado para v1
  free(v2); // libera el espacio reservado para v2
  free(v3); // libera el espacio reservado para v3
  #endif
  return 0;
}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer7] 2019-04-02 martes
$gcc -O2 SumaVectoresC_openMP.c -o SumaVectoresC_openMP -lrt -fopenmp
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer7] 2019-04-02 martes
$export OMP_NUM_THREADS=4
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer7] 2019-04-02 martes
$export OMP_DYNAMIC=FALSE
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer7] 2019-04-02 martes
$./SumaVectoresC_openMP 8
Tama□ Vectores:8 (4 B)
Tiempo:0.000006953       / Tama□ 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) /
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer7] 2019-04-02 martes
$./SumaVectoresC_openMP 11
Tama□ Vectores:11 (4 B)
Tiempo:0.000007147       / Tama□ 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) /
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer7] 2019-04-02 martes
$
```

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

RESPUESTA: Captura que muestre el código fuente implementado

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

 Para compilar usar (-lrt: real time library, es posible que no sea necesario usar -lrt):
   gcc  -O2  SumaVectores.c -o SumaVectores -lrt
   gcc  -O2 -S SumaVectores.c -lrt

 Para ejecutar use: SumaVectoresC longitud

*/

#include <stdlib.h>  // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h>   // biblioteca donde se encuentra la funci\u00f3n printf()
#include <time.h> // biblioteca donde se encuentra la funci\u00f3n clock_gettime()

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

//S\u00f3lo puede estar definida una de las tres constantes VECTOR_ (s\u00f3lo uno de los ...
//tres defines siguientes puede estar descomentado):
//#define VECTOR_LOCAL  // descomentar para que los vectores sean variables ...
        // locales (si se supera el tama\u00f1o de la pila se ...
        // generar\u00e1 el error 'Violaci\u00f3n de Segmento')
//#define VECTOR_GLOBAL // descomentar para que los vectores sean variables ...
        // globales (su longitud no estar\u00e1 limitada por el ...
        // tama\u00f1o de la pila del programa)
#define VECTOR_DYNAMIC  // descomentar para que los vectores sean variables ...
        // din\u00e1micas (memoria reutilizable durante la ejecuci\u00f3n)

#ifdef VECTOR_GLOBAL
#define MAX 33554432 //=2^25

double v1[MAX], v2[MAX], v3[MAX];
#endif
int main(int argc, char** argv){

  struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecuci\u00f3n

  //Leer argumento de entrada (n\u00ba de componentes del vector)
  if (argc<2){
    printf("Faltan n\u00ba componentes del vector\n");
    exit(-1);
  }

  unsigned int N = atoi(argv[1]);   // M\u00e1ximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
  printf("Tama\u00f1o Vectores:%u (%u B)\n",N, sizeof(unsigned int));


  #ifdef VECTOR_DYNAMIC
  double *v1, *v2, *v3;
  v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tama\u00f1o en bytes
  v2 = (double*) malloc(N*sizeof(double));
  v3 = (double*) malloc(N*sizeof(double));
  if ((v1 == NULL) || (v2 == NULL) || (v3 == NULL)) {
    printf("No hay suficiente espacio para los vectores \n");
    exit(-2);
  }
  #endif


  //Inicializar vectores
  #pragma omp parallel 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;
       }
  }


  double start = omp_get_wtime();
  //Calcular suma de vectores

  #pragma omp parallel 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];
       }
  }


  double end = omp_get_wtime();

  double time = end - start;

  //Imprimir resultado de la suma y el tiempo de ejecuci\u00f3n
  if (N<10) {
  printf("Tiempo:%11.9f\t / Tama\u00f1o Vectores:%u\n",time,N);
  for(int 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:%11.9f\t / Tama\u00f1o 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",
           time,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);


  #ifdef VECTOR_DYNAMIC
  free(v1); // libera el espacio reservado para v1
  free(v2); // libera el espacio reservado para v2
  free(v3); // libera el espacio reservado para v3
  #endif
  return 0;
}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer8] 2019-04-02 martes
$gcc -O2 SumaVectoresC_sections.c -o SumaVectoresC_sections -lrt -fopenmp
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer8] 2019-04-02 martes
$export OMP_NUM_THREADS=4
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer8] 2019-04-02 martes
$export OMP_DYNAMIC=FALSE
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer8] 2019-04-02 martes
$./SumaVectoresC_sections 8
Tama□ Vectores:8 (4 B)
Tiempo:0.000009188      / Tama□ 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) /
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer8] 2019-04-02 martes
$./SumaVectoresC_sections 11
Tama□ Vectores:11 (4 B)
Tiempo:0.000005739      / Tama□ 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) /
[AntonioDavidVillegasYeguas antonio@antonio:~/Documentos/Universidad/2do/2do_cuatri/AC/Practicas/BP1/ejer8] 2019-04-02 martes
$
```

9. ¿Cuántos threads y cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 7? Razone su respuesta. ¿Cuántos threads y cuantos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta.

   RESPUESTA: La versión del ejercicio 7 podria implementar N threads como máximo, ya que cada thread podria ejecutar una única iteración del bucle for, si tuvieramos M threads con M > N, algunos threads quedarían inactivos

   En la implementación del ejercicio 8 solo podriamos tener 4 threads, ya que he dividido en 4 sections el calculo, y aunque tuvieramos más threads, solo 4 entrarian en cada hebra

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

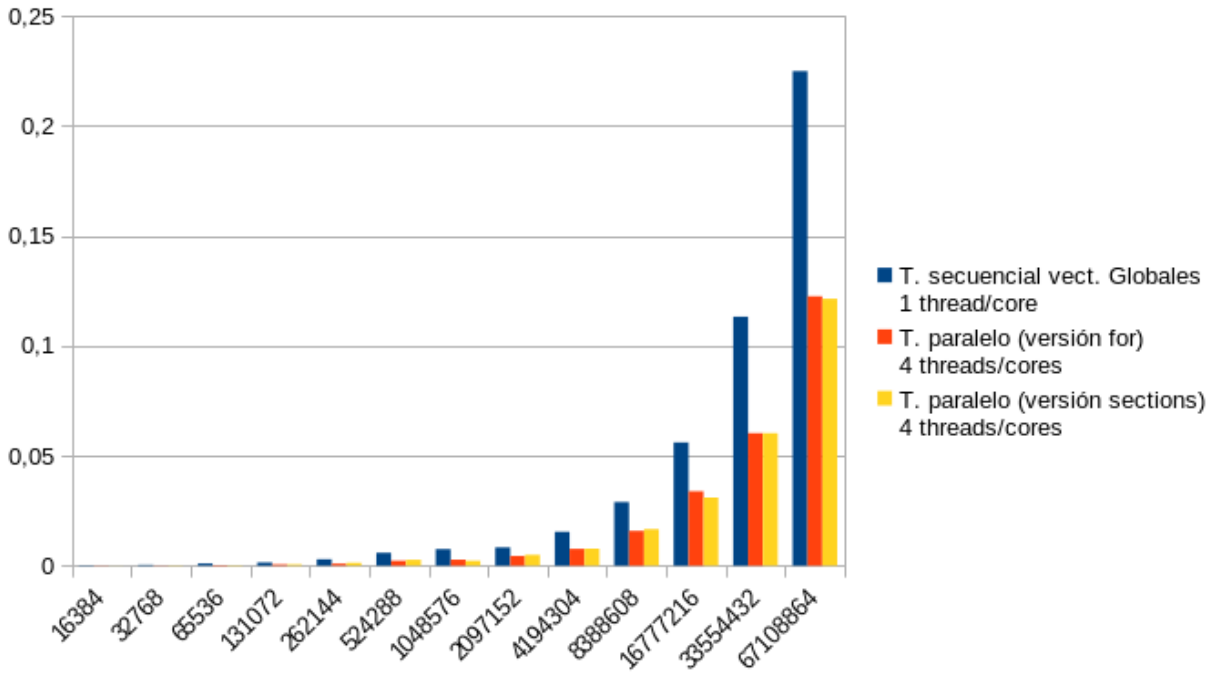
RESPUESTA:

Mi PC **Intel i5-8250U**

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 4 threads/cores	T. paralelo (versión sections) 4 threads/cores
16384	0,00020064	6,8951E-05	5,8487E-05
32768	0,000345707	0,000114316	0,000143378
65536	0,000943194	0,000222505	0,000211699
131072	0,001411803	0,000576929	0,000549028
262144	0,00289788	0,000927311	0,001302597
524288	0,0058604	0,002236288	0,002638852
1048576	0,007499382	0,002692097	0,00223478
2097152	0,008292539	0,004401032	0,004919438
4194304	0,015374117	0,007629657	0,007750333
8388608	0,0289445	0,015823796	0,016613158
16777216	0,056007118	0,033802067	0,03094324
33554432	0,113174536	0,060184144	0,060233511
67108864	0,224890717	0,122471963	0,121353418

ATCgrid

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 4 threads/cores	T. paralelo (versión sections) 4 threads/cores
16384	0,000122624	4,6314E-05	5,4928E-05
32768	0,000238818	0,000162764	7,8444E-05
65536	0,000473247	0,000136748	0,000158723
131072	0,00094761	0,000272179	0,000331622
262144	0,001892547	0,000589282	0,000628533
524288	0,002871337	0,001021006	0,000996346
1048576	0,005647381	0,002308663	0,002066028
2097152	0,010000441	0,004009223	0,004602092
4194304	0,018214631	0,006732957	0,007566364
8388608	0,033410888	0,013093939	0,013711518
16777216	0,066655803	0,042461181	0,027287619
33554432	0,131965512	0,071881953	0,069966638
67108864	0,262236944	0,09671648	0,115302753


0,25
0,2
0,15
0,1
0,05
0
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
T. secuencial vect. Globales
1 thread/core
T. paralelo (versión for)
4 threads/cores
T. paralelo (versión sections)
4 threads/cores

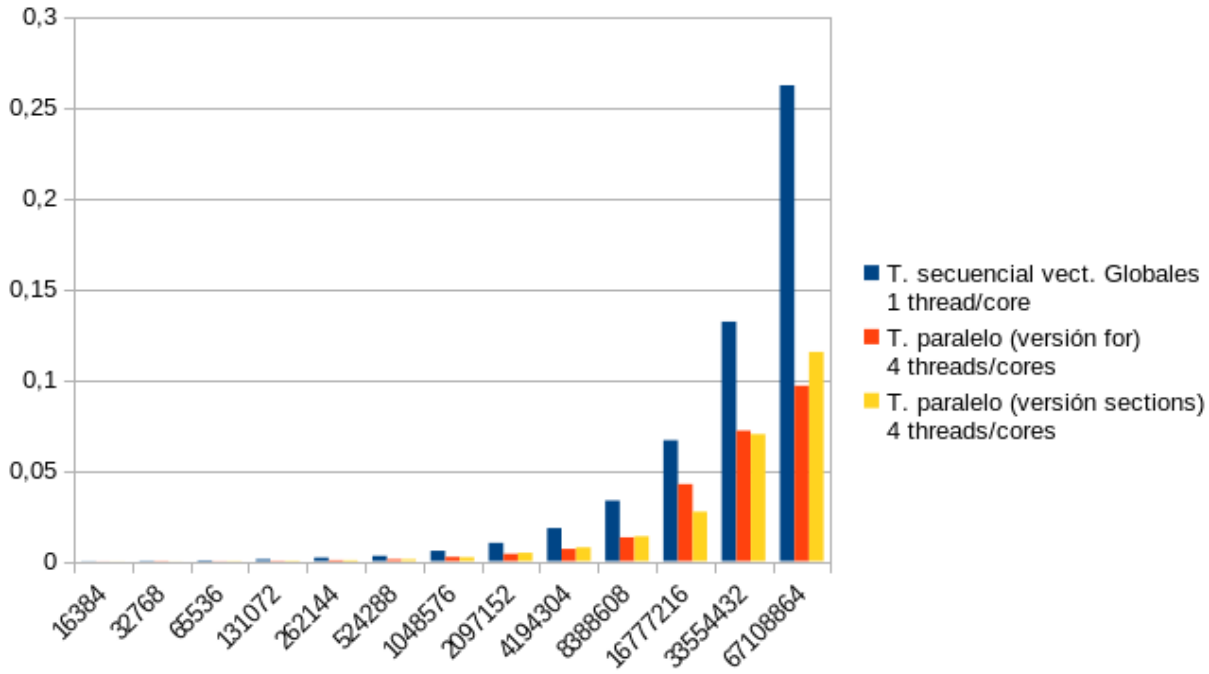

0,3
0,25
0,2
0,15
0,1
0,05
0
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
T. secuencial vect. Globales
1 thread/core
T. paralelo (versión for)
4 threads/cores
T. paralelo (versión sections)
4 threads/cores

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

Nº de Componentes	**T. secuencial vect. Globales** **1 thread/core**	**T. paralelo (versión `for`)** **¿?threads/cores**	**T. paralelo (versión `sections`)** **¿?threads/cores**
16384			
32768			
65536			
131072			
262144			
524288			
1048576			
2097152			
4194304			
8388608			
16777216			
33554432			
67108864			

11. Rellenar una tabla como la Tabla 3 para atcgrid con el tiempo de ejecución, tiempo de CPU del usuario y tiempo CPU del sistema obtenidos con `time` para el ejecutable del ejercicio 7 y para el programa secuencial del Listado 1. Ponga en la tabla el número de threads/cores que usan los códigos. ¿El tiempo de CPU que se obtiene es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.
RESPUESTA:

	Tiempo secuencial vect. Globales 1 thread/core			**Tiempo paralelo/versión for 4 Threads/cores**		
Nº de Componentes	**Elapsed**	**CPU-user**	**CPU- sys**	**Elapsed**	**CPU-user**	**CPU- sys**
65536	elapsed 0.00	user 0.00	system 0.00	elapsed 0.00	user 0.00	system 0.00
131072	elapsed 0.00	user 0.00	system 0.00	elapsed 0.00	user 0.00	system 0.00
262144	elapsed 0.00	user 0.00	system 0.00	elapsed 0.00	user 0.00	system 0.00
524288	elapsed 0.01	user 0.00	system 0.00	elapsed 0.00	user 0.01	system 0.00
1048576	elapsed 0.01	user 0.00	system 0.00	elapsed 0.00	user 0.02	system 0.00
2097152	elapsed 0.02	user 0.01	system 0.01	elapsed 0.01	user 0.03	system 0.01
4194304	elapsed 0.04	user 0.02	system 0.01	elapsed 0.02	user 0.04	system 0.03
8388608	elapsed 0.07	user 0.05	system 0.02	elapsed 0.04	user 0.09	system 0.06
16777216	elapsed 0.15	user 0.10	system 0.05	elapsed 0.09	user 0.18	system 0.12
33554432	elapsed 0.33	user 0.18	system 0.14	elapsed 0.11	user 0.24	system 0.18
67108864	elapsed 0.61	user 0.32	system 0.29	elapsed 0.26	user 0.53	system 0.41

Vemos como en la ejecución de 1 thread la suma de los tiempos de CPU es menor que el tiempo real, sin embargo en la versión de 4 hreads, el tiempo de CPU es mayor que el real, debido a que en el real se cuenta desde el inicio hasta el fin del programa, y en los tiempos de CPU se cuenta el tiempo de CPU de cada thread.

Tabla 3. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados.

Nº de Componentes	**Tiempo secuencial vect. Globales 1 thread/core**			**Tiempo paralelo/versión `for` ¿? Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536						
131072						
262144						
524288						
1048576						
2097152						
4194304						
8388608						
16777216						
33554432						
67108864						