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): Christian Andrades Molina
Grupo de prácticas: B2
Fecha de entrega: 01/04/14
Fecha evaluación en clase: 09/04/14

---

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>
#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: código fuente `sectionsModificado.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());
}
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>
#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
               { printf("Añadiendo directiva single dentro de parallel para result:\n");
               for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
               printf("\n");
               }
     }
}
```

CAPTURAS DE PANTALLA:

```
christian@christian-530U4E-540U4E:~/Escritorio$ gcc -O2 -fopenmp singleModificad
o.c -o singlemodi
christian@christian-530U4E-540U4E:~/Escritorio$ export OMP_DYNAMIC=FALSE
christian@christian-530U4E-540U4E:~/Escritorio$ export OMP_NUM_THREADS=8
christian@christian-530U4E-540U4E:~/Escritorio$ ./singlemodi
Introduce valor de inicialización a: 23
Single ejecutada por el thread 1
Añadiendo directiva single dentro de parallel para result:
Single ejecutada por el thread 0
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
christian@christian-530U4E-540U4E:~/Escritorio$ export OMP_NUM_THREADS=3
christian@christian-530U4E-540U4E:~/Escritorio$ ./singlemodi
Introduce valor de inicialización a: 12
Single ejecutada por el thread 0
Añadiendo directiva single dentro de parallel para result:
Single ejecutada por el thread 1
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
```

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 `masterModificado2.c`

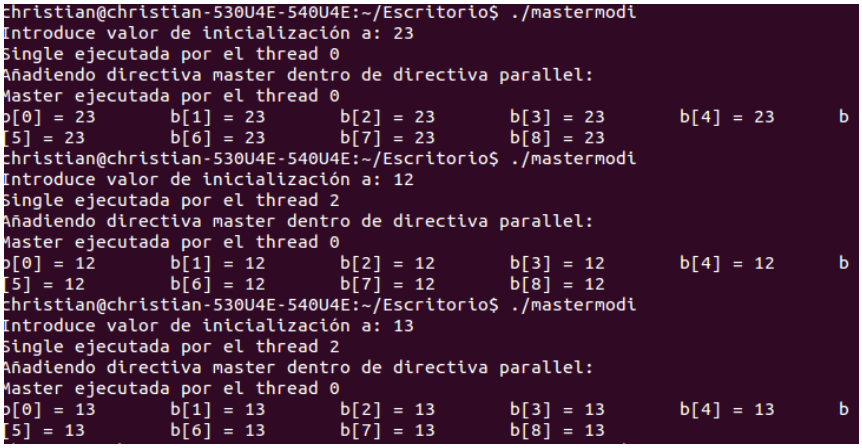
```
#include <stdio.h>
#include <omp.h>

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

  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
               {tid=omp_get_thread_num();
               printf("Añadiendo directiva master dentro de directiva parallel:\n");
               printf("Master ejecutada 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:



RESPUESTA A LA PREGUNTA: Imprimir los resultados añadiendo una directiva Single la hace cualquiera de las hebras disponibles, pero si usamos la directiva Master, la ejecuta la hebra master (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: Master no tiene barrera implicita en ninguno de los casos, por lo que no esperaría a la ejecución de todas las hebras disponibles realizando el calculo y mostraría (a veces), una suma total errónea.

1.1.1 Resto de ejercicios

En clase presencial y en casa, usando plataformas (procesadores + SO) de 64 bits**,** se trabajarán los siguientes ejercicios y cuestiones:

1. El programa secuencial C del Listado 1calcula 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 el tiempo de ejecución (*elapsed time*) y el tiempo de CPU del usuario y del sistema para el ejecutable en atcgrid y en el PC local. Obtenga los tiempos para vectores con 10000000 componentes.

   CAPTURAS DE PANTALLA:

   PC

```
christian@christian-530U4E-540U4E:~$ cd Escritorio
christian@christian-530U4E-540U4E:~/Escritorio$ time ./SumaVectoresC 10000000
Tiempo(seg.):0.038432469         / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3
[0](1000000.000000+1000000.000000=2000000.0000000) / / V1[9999999]+V2[9999999]=V
3[9999999](1999999.900000+0.100000=2000000.000000)/

real    0m0.151s
user    0m0.068s
sys     0m0.083s
```

   ATCGRID

```
-bash-4.2$ echo 'time ./practica2/ejer11 10000000  10000000' | qsub -q ac
32240.atcgrid
-bash-4.2$ ls -lag
total 36
drwxrwxr-x  2 B2estudiante2 4096 mar 26 11:42 .
drwx------ 11 B2estudiante2 4096 mar 26 11:40 ..
-rwxr-xr-x  1 B2estudiante2 8685 mar 26 11:29 ejer11
-rw-------  1 B2estudiante2   42 mar 26  2014 STDIN.e32232
-rw-------  1 B2estudiante2   42 mar 26  2014 STDIN.e32240
-rw-------  1 B2estudiante2  202 mar 26  2014 STDIN.o32232
-rw-------  1 B2estudiante2  202 mar 26  2014 STDIN.o32240
-bash-4.2$ cat STDIN.e32240

real    0m0.120s
user    0m0.067s
sys     0m0.048s
-bash-4.2$ cat STDIN.o32240
Tiempo(seg.):0.040027060         / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](1000000.000000
+1000000.000000=2000000.0000000) / / V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2
000000.000000)/
```

2. 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:

   Tiempo(seg.):**0.000403730** / Tamaño Vectores:10 / V1[0]+V2[0]=V3[0] (1.000000+1.000000=2.0000000) / / V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000)/
   Tiempo(seg.):**0.039443247** / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0] (1000000.000000+1000000.000000=2000000.0000000) / / V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000)/

MIPS (10) = 10 / 0.000403730 seg * 10^6 = 0.024 MIPS
MIPS (10000000) = 10000000 / 0.039443247 * 10^6 = 253.52 MIPS
MFLOPS (10) = 3*10 / 0.000403730 seg * 10^6 = 0.074 MFLOPS
MFLOPS(10000000) = 3*10000000 / 0.039443247 * 10^6 = 760.5 MFLOPS

RESPUESTA: código ensamblador generado de la parte de la suma de vectores

```
                call            clock_gettime
                xorl            %eax, %eax
                .p2align 4,,10
                .p2align 3
.L7:
                movsd           v1(%rax), %xmm0
                addsd           v2(%rax), %xmm0
                movsd           %xmm0, v3(%rax)
                addq            $8, %rax
                cmpq            %rbp, %rax
                jne             .L7
.L8:
                leaq            16(%rsp), %rsi
                xorl            %edi, %edi
                call            clock_gettime
```

3. 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.c
Suma de dos vectores: v3 = v1 + v2
Para compilar usar (-lrt: real time library):
gcc -O2 SumaVectores.c -o SumaVectores -lrt
Para ejecutar use: SumaVectoresC 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
//Sólo puede estar definida una de las tres constantes VECTOR_ (sólo uno de los ...
//tres defines siguientes puede estar descomentado):
//#define VECTOR_LOCAL // descomentar para que los vectores sean variables ...
// locales (si se supera el tamaño de la pila se ...
// generará el error "Violación de Segmento")

//#define VECTOR_GLOBAL// descomentar para que los vectores sean variables ...
 // globales (su longitud no estará limitada por el ...
 // tamaño de la pila del programa)
```

```
 #define VECTOR_DYNAMIC // descomentar para que los vectores sean variables ...
 // dinámicas (memoria reutilizable durante la ejecución)
 #ifdef VECTOR_GLOBAL
 #define MAX 33554432 //=2^25
 double v1[MAX], v2[MAX], v3[MAX];
 #endif

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

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)
 #ifdef VECTOR_LOCAL
 double v1[N], v2[N], v3[N];
// Tamaño variable local en tiempo de ejecución ...
// disponible en C a partir de actualización C99
 #endif

 #ifdef VECTOR_GLOBAL
 if (N>MAX) N=MAX;
 #endif

 #ifdef VECTOR_DYNAMIC
 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);
 }
 #endif

               //Seccion para inicializacion de vectores
               #pragma omp parallel for
               for(i=0; i<N; i++){
               //printf("\nthread %d realiza la iteración %d del bucle inicializacion
",
               //omp_get_thread_num(),i);
                v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de N
               }


               cgt1 = omp_get_wtime();

               #pragma omp parallel for
               //Seccion para suma de vectores
               for(i=0; i<N; i++){
               //printf("\nthread %d realiza la iteración %d del bucle suma ",
               //omp_get_thread_num(),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=
%8.6f0) / / 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
 #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:

```
christian@christian-530U4E-540U4E:~/Escritorio$ OMP_DYNAMIC=FALSE
christian@christian-530U4E-540U4E:~/Escritorio$ OMP_NUM_THREADS=8
christian@christian-530U4E-540U4E:~/Escritorio$ gcc -O2 -fopenmp -o ejer Ejer7.c

christian@christian-530U4E-540U4E:~/Escritorio$ time ./ejer 8

thread 2 realiza la iteración 4 del bucle inicializacion
thread 2 realiza la iteración 5 del bucle inicializacion
thread 3 realiza la iteración 6 del bucle inicializacion
thread 3 realiza la iteración 7 del bucle inicializacion
thread 1 realiza la iteración 2 del bucle inicializacion
thread 1 realiza la iteración 3 del bucle inicializacion
thread 0 realiza la iteración 0 del bucle inicializacion
thread 0 realiza la iteración 1 del bucle inicializacion
thread 2 realiza la iteración 4 del bucle suma
thread 2 realiza la iteración 5 del bucle suma
thread 3 realiza la iteración 6 del bucle suma
thread 3 realiza la iteración 7 del bucle suma
thread 1 realiza la iteración 2 del bucle suma
thread 0 realiza la iteración 0 del bucle suma
thread 0 realiza la iteración 1 del bucle suma
thread 1 realiza la iteración 3 del bucle suma Tiempo(seg.):0.000163115  / Tamañ
o Vectores:8    / V1[0]+V2[0]=V3[0](0.800000+0.800000=1.6000000) / / V1[7]+V2[7]
=V3[7](1.500000+0.100000=1.600000)/

real    0m0.004s
user    0m0.000s
sys     0m0.008s
```

4. 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.c
Suma de dos vectores: v3 = v1 + v2
Para compilar usar (-lrt: real time library):
gcc -O2 SumaVectores.c -o SumaVectores -lrt
Para ejecutar use: SumaVectoresC 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
//Sólo puede estar definida una de las tres constantes VECTOR_ (sólo uno de los ...
//tres defines siguientes puede estar descomentado):
//#define VECTOR_LOCAL // descomentar para que los vectores sean variables ...
// locales (si se supera el tamaño de la pila se ...
// generará el error "Violación de Segmento")

//#define VECTOR_GLOBAL// descomentar para que los vectores sean variables ...
 // globales (su longitud no estará limitada por el ...
 // tamaño de la pila del programa)

#define VECTOR_DYNAMIC // descomentar para que los vectores sean variables ...
 // dinámicas (memoria reutilizable durante la ejecución)
 #ifdef VECTOR_GLOBAL
 #define MAX 33554432 //=2^25
 double v1[MAX], v2[MAX], v3[MAX];
 #endif


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)
 #ifdef VECTOR_LOCAL
 double v1[N], v2[N], v3[N];
// Tamaño variable local en tiempo de ejecución ...
// disponible en C a partir de actualización C99
 #endif

 #ifdef VECTOR_GLOBAL
 if (N>MAX) N=MAX;
 #endif

 #ifdef VECTOR_DYNAMIC
 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);
 }
 #endif


#pragma omp parallel sections
{



               #pragma omp section
               {
               //Seccion para inicializacion de vectores
               printf("\nthread %d realiza seccion inicializacion",
omp_get_thread_num());
               for(i=0; i<N; i++)
                v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
               }


               #pragma omp section
               {
               cgt1 = omp_get_wtime();
               //Seccion para suma de vectores
               printf("\nthread %d realiza seccion suma\n", omp_get_thread_num());
               for(i=0; 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=
%8.6f0) / / 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
 #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:

```
christian@christian-530U4E-540U4E:~/Escritorio$ OMP_DYNAMIC=FALSE
christian@christian-530U4E-540U4E:~/Escritorio$ OMP_NUM_THREADS=4
christian@christian-530U4E-540U4E:~/Escritorio$ time ./ejer8 8

thread 2 realiza seccion inicializacion
thread 1 realiza seccion suma
Tiempo(seg.):0.000132019        / Tamaño Vectores:8    / V1[0]+V2[0]=V3[0](0.80
0000+0.000000=0.8000000) / / V1[7]+V2[7]=V3[7](1.500000+0.000000=1.500000)/

real    0m0.004s
user    0m0.004s
sys     0m0.002s
christian@christian-530U4E-540U4E:~/Escritorio$
```

5. ¿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:
   Ejer7: 8 threads y 4 cores (1 procesador con 4 cores i5)
   Ejer8: 4 threads y 4 cores (1 procesador con 4 cores i5)

6. Rellenar una tabla como la Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas 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 (*elapsed time*) 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. (Usar dinámico).
   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.

Nº de Componentes PC-LOCAL	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión `for`) 8 threads/cores	T. paralelo (versión `sections`) 4 threads/cores
65536	0m0.005s	0m0.011s	0m0.012s
131072	0m0.008s	0m0.004s	0m0.003s
262144	0m0.010s	0m0.005s	0m0.004s
524288	0m0.014s	0m0.010s	0m0.008s
1048576	0m0.017s	0m0.010s	0m0.015s
2097152	0m0.033s	0m0.019s	0m0.025s
4194304	0m0.063s	0m0.035s	0m0.046s
8388608	0m0.131s	0m0.067s	0m0.092s
16777216	0m0.230s	0m0.123s	0m0.168s
33554432	0m0.446s	0m0.237s	0m0.327s
67108864	0m0.874s	0m0.477s	0m0.656s

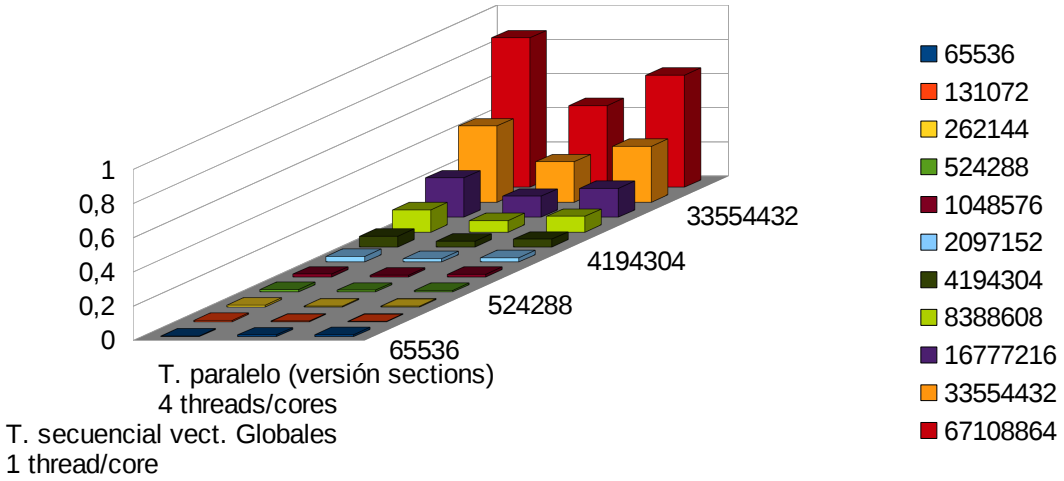
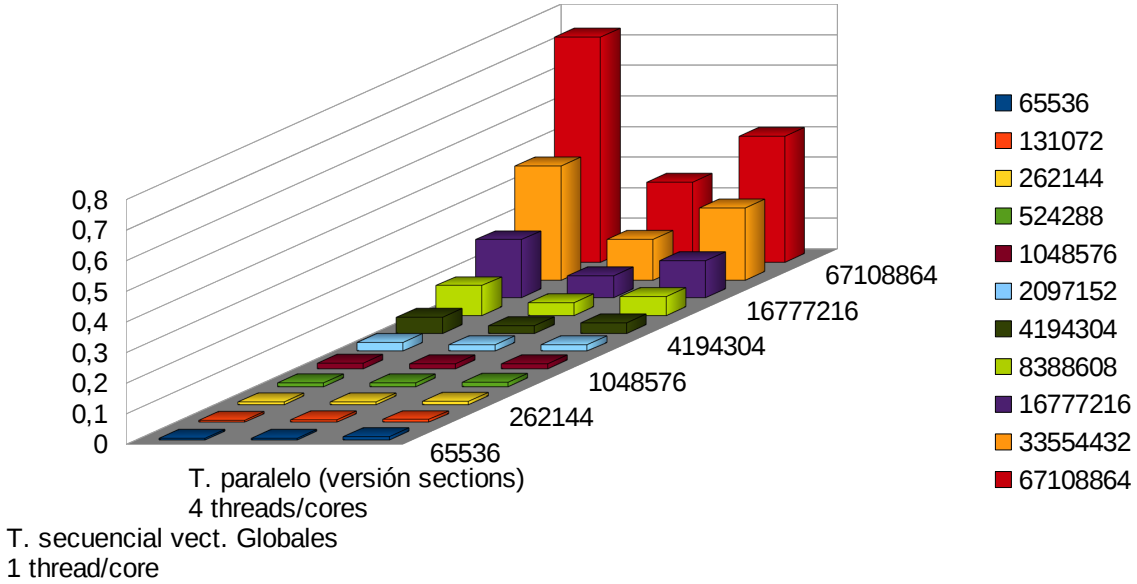


Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados.

Nº de Componentes ATCGRID	**T. secuencial vect. Globales** **1 thread/core**	**T. paralelo (versión `for`)** **8 threads/cores**	**T. paralelo (versión `sections`)** **4 threads/cores**
65536	0m0.006s	0m0.006s	0m0.011s
131072	0m0.006s	0m0.008s	0m0.009s
262144	0m0.009s	0m0.008s	0m0.010s
524288	0m0.013s	0m0.012s	0m0.014s
1048576	0m0.019s	0m0.017s	0m0.017s
2097152	0m0.029s	0m0.021s	0m0.021s
4194304	0m0.052s	0m0.026s	0m0.035s
8388608	0m0.098s	0m0.042s	0m0.062s
16777216	0m0.190s	0m0.072s	0m0.121s
33554432	0m0.372s	0m0.133s	0m0.236s
67108864	0m0.736s	0m0.262s	0m0.411s



7. Rellenar una tabla como la Tabla 3. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas 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.
   RESPUESTA:

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

Nº de Componentes	**Tiempo secuencial vect. Globales 1 thread/core**			**Tiempo paralelo/versión `for` 8 Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	real 0m0.005s	user 0m0.000s	sys 0m0.005s	real 0m0.005s	user 0m0.012s	sys 0m0.000s
131072	real 0m0.006s	user 0m0.003s	sys 0m0.003s	real 0m0.010s	user 0m0.027s	sys 0m0.000s
262144	real 0m0.005s	user 0m0.005s	sys 0m0.000s	real 0m0.008s	user 0m0.014s	sys 0m0.010s
524288	real 0m0.008s	user 0m0.000s	sys 0m0.008s	real 0m0.009s	user 0m0.021s	sys 0m0.004s
1048576	real 0m0.016s	user 0m0.004s	sys 0m0.012s	real 0m0.014s	user 0m0.038s	sys 0m0.004s
2097152	real 0m0.031s	user 0m0.015s	sys 0m0.016s	real 0m0.020s	user 0m0.026s	sys 0m0.037s
4194304	real 0m0.057s	user 0m0.032s	sys 0m0.025s	real 0m0.034s	user 0m0.072s	sys 0m0.038s
8388608	real 0m0.111s	user 0m0.047s	sys 0m0.063s	real 0m0.062s	user 0m0.115s	sys 0m0.092s
16777216	real 0m0.220s	user 0m0.080s	sys 0m0.140s	real 0m0.122s	user 0m0.202s	sys 0m0.194s
33554432	real 0m0.432s	user 0m0.206ssys	0m0.223s	real 0m0.232s	user 0m0.386s	sys 0m0.378s
67108864	real 0m0.873s	user 0m0.428s	sys 0m0.441s	real 0m0.466s	user 0m0.670s	sys 0m0.825s