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): Daniel Díaz Pareja
Grupo de prácticas: A2
Fecha de entrega: 07/04/2016
Fecha evaluación en clase: 08/04/2016

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>
#include <omp.h>

int main(int argc, char ** argv)
{
 int i, n = 9;
 if(argc < 2) {
   fprintf(stderr,"\n[ERROR] - Falta n° 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());
}
```

```
int main(int argc, char ** argv) {
 #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: código fuente `singleModificado.c`

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


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


 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("Single 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");
   }
 }
```

```
 return(0);
}
```

CAPTURAS DE PANTALLA:

```
dani@dani-Aspire-5750G:$export OMP_NUM_THREADS=4
dani@dani-Aspire-5750G:$./singleModificado
Introduce valor de inicialización a: 10
Single ejecutada por el thread 1
Single ejecutada por el thread 0
b[0] = 10       b[1] = 10       b[2] = 10       b[3] = 10       b[4] = 10
b[6] = 10       b[7] = 10       b[8] = 10
dani@dani-Aspire-5750G:$
```

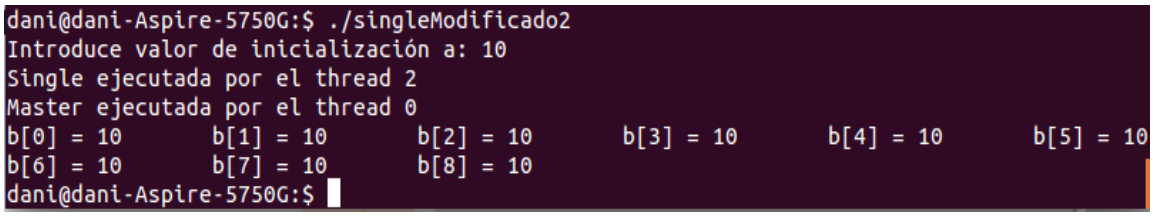
3. Imprimir los resultados del programa `single.c` usando una directiva `master` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `master` incorporada, para que se imprima el identificador del thread que ejecuta 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>
#include <omp.h>


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


 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("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");
    }
  }


 return(0);
}
```

CAPTURAS DE PANTALLA:



RESPUESTA A LA PREGUNTA: La diferencia es que en este caso, nos aseguramos de que la parte Master la va a ejecutar la hebra master del programa (la 0), mientras que en el anterior la parte single puede ejecutar cualquier hebra (antes ha sido casualidad que la ejecute la hebra master)

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

RESPUESTA: La directiva atomic no tiene una barrera implícita, por lo que si se elimina la directiva barrier que hace la función de sincronización entre la suma local de todas las hebras, puede que la hebra master imprima su resultado antes de que la suma total se haya completado.

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 10.000.000 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:

```
dani@dani-Aspire-5750G:$ time ./SumaVectoresCGlobales 10000000

real    0m0.155s
user    0m0.104s
sys     0m0.048s
dani@dani-Aspire-5750G:$ time ./SumaVectoresCGlobales 10000000

real    0m0.161s
user    0m0.132s
sys     0m0.028s
dani@dani-Aspire-5750G:$ time ./SumaVectoresCGlobales 10000000

real    0m0.140s
user    0m0.096s
sys     0m0.040s
dani@dani-Aspire-5750G:$ time ./SumaVectoresCGlobales 10000000

real    0m0.165s
user    0m0.136s
sys     0m0.028s
dani@dani-Aspire-5750G:$ time ./SumaVectoresCGlobales 10000000

real    0m0.159s
user    0m0.128s
sys     0m0.028s
dani@dani-Aspire-5750G:$
```

RESPUESTA: En la mayoría de casos, el CPU time (user+sys) es menor o igual que el elapsed time. Esto es porque este programa no aprovecha paralalelismo, todo se hace en un flujo de control, en una sola cpu, por lo que el tiempo de cpu al final es muy parecido al tiempo real. Si usase varias cpu's, el tiempo total de cpu sería la suma de todos los tiempos consumidos por cada cpu, por lo que el elapsed sería menor que el tiempo de user+sys.

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:

   Comando 'time' para vectores de 10 elementos:

```
[A2estudiante8@atcgrid practica1]$ echo 'time practica1/SumaVectoresCGlobales 10' | qsub -q ac
33089.atcgrid
[A2estudiante8@atcgrid practica1]$ cat STDIN.e33089

real    0m0.003s
user    0m0.000s
sys     0m0.002s
```

   Comando 'time' para vectores de 10.000.000 elementos:

```
[A2estudiante8@atcgrid practica1]$ echo 'time practica1/SumaVectoresCGlobales 10000000' | qsub -q ac
33090.atcgrid
[A2estudiante8@atcgrid practica1]$ cat STDIN.e33090

real    0m0.170s
user    0m0.064s
sys     0m0.104s
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Para vectores de 10 elementos:

- Tcpu = user + sys = 0.002s
- NI = (nº de instrucciones de la suma de los vectores) * tamaño vectores = 6 * 10 = 60
- MIPS = NI/(Tcpu * 10^6) = 60 / (0.002 * 10^6) = 0,03 MIPS
- Operaciones en coma flotante = iteraciones del bucle (número de elementos del vector) * operaciones realizadas (una suma) = 10 * 1 = 10
- MFLOPS = OP_coma_flotante / (Tcpu * 10^6) = 10 / (0.002 * 10^6) = 0,005 MFLOPS

Para vectores de 10.000.000 elementos:

- Tcpu = user + sys = 0,064s + 0,104s = 0,168s
- NI = (nº de instrucciones de la suma de los vectores) * tamaño vectores = 6 * 10.000.000 = 60.000.000
- MIPS = NI/(Tcpu * 10^6) = 60.000.000 / (0.168 * 10^6) = 357 MIPS
- Operaciones en coma flotante = iteraciones del bucle (número de elementos del vector) * operaciones realizadas (una suma) = 10 * 1 = 10.000.000

  MFLOPS = OP_coma_flotante / (Tcpu * 10^6) = 10.000.000 / (0.168 * 10^6) = 59 MFLOPS

RESPUESTA:

código ensamblador generado de la parte de la suma de vectores

```
movsd         v1(%rax), %xmm0
addq          $8, %rax
addsd         v2-8(%rax), %xmm0
movsd         %xmm0, v3-8(%rax)
cmpq          %rbx, %rax
jne           .L7
```

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

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

```
Para compilar usar (-lrt: real time library):
gcc -O2 SumaVectores.c -o SumaVectores -lrt
gcc -O2 -S SumaVectores.c -lrt //para generar el código ensamblador
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> // biblioteca donde se encuentran las funciones OMP.
//#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;
struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución
//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]); // 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


//Inicializar vectores


#pragma omp parallel for // Combinamos las directivas para usar solo 1
barrera implícita
    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
        printf("La hebra [%d] escribe V1[%f], V2[%f].\n",
omp_get_thread_num(),v1[i],v2[i]);
    }


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];


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]);


if (N<=11)
    for(i=0; i<N; i++)
        printf("v1[%d]=[%f], v2[%d]=[%f],
v3[%d]=[%f]\n",i,v1[i],i,v2[i],i,v3[i]);




printf("\n");
#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 (compilación y ejecución para N=8 y N=11):

N=8

```
dani@dani-Aspire-5750G:$ gcc -O2 -fopenmp -o SumaVectoresCOMPfor SumaVectoresCOMPf
or.c
dani@dani-Aspire-5750G:$ ./SumaVectoresCOMPfor 8
La hebra 3 escribe V1[6]=1.400000, V2[6]=0.200000.
La hebra 3 escribe V1[7]=1.500000, V2[7]=0.100000.
La hebra 2 escribe V1[4]=1.200000, V2[4]=0.400000.
La hebra 2 escribe V1[5]=1.300000, V2[5]=0.300000.
La hebra 1 escribe V1[2]=1.000000, V2[2]=0.600000.
La hebra 1 escribe V1[3]=1.100000, V2[3]=0.500000.
La hebra 0 escribe V1[0]=0.800000, V2[0]=0.800000.
La hebra 0 escribe V1[1]=0.900000, V2[1]=0.700000.
Tiempo(seg.):0.001570060         / Tama◆o Vectores:8    / V1[0]+V2[0]=V3[0](0.8000
00+0.800000=1.600000) /        V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
v1[0]=0.800000, v2[0]=0.800000, v3[0]=1.600000
v1[1]=0.900000, v2[1]=0.700000, v3[1]=1.600000
v1[2]=1.000000, v2[2]=0.600000, v3[2]=1.600000
v1[3]=1.100000, v2[3]=0.500000, v3[3]=1.600000
v1[4]=1.200000, v2[4]=0.400000, v3[4]=1.600000
v1[5]=1.300000, v2[5]=0.300000, v3[5]=1.600000
v1[6]=1.400000, v2[6]=0.200000, v3[6]=1.600000
v1[7]=1.500000, v2[7]=0.100000, v3[7]=1.600000
```

N = 11

```
dani@dani-Aspire-5750G:$ gcc -O2 -fopenmp -o SumaVectoresCOMPfor SumaVectoresCOMPf
or.c
dani@dani-Aspire-5750G:$ ./SumaVectoresCOMPfor 11
La hebra 1 escribe V1[3]=1.400000, V2[3]=0.800000.
La hebra 1 escribe V1[4]=1.500000, V2[4]=0.700000.
La hebra 1 escribe V1[5]=1.600000, V2[5]=0.600000.
La hebra 0 escribe V1[0]=1.100000, V2[0]=1.100000.
La hebra 0 escribe V1[1]=1.200000, V2[1]=1.000000.
La hebra 0 escribe V1[2]=1.300000, V2[2]=0.900000.
La hebra 3 escribe V1[9]=2.000000, V2[9]=0.200000.
La hebra 3 escribe V1[10]=2.100000, V2[10]=0.100000.
La hebra 2 escribe V1[6]=1.700000, V2[6]=0.500000.
La hebra 2 escribe V1[7]=1.800000, V2[7]=0.400000.
La hebra 2 escribe V1[8]=1.900000, V2[8]=0.300000.
Tiempo(seg.):0.001379972         / Tama◆o Vectores:11   / V1[0]+V2[0]=V3[0](1.1000
00+1.100000=2.200000) /        V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
v1[0]=1.100000, v2[0]=1.100000, v3[0]=2.200000
v1[1]=1.200000, v2[1]=1.000000, v3[1]=2.200000
v1[2]=1.300000, v2[2]=0.900000, v3[2]=2.200000
v1[3]=1.400000, v2[3]=0.800000, v3[3]=2.200000
v1[4]=1.500000, v2[4]=0.700000, v3[4]=2.200000
v1[5]=1.600000, v2[5]=0.600000, v3[5]=2.200000
v1[6]=1.700000, v2[6]=0.500000, v3[6]=2.200000
v1[7]=1.800000, v2[7]=0.400000, v3[7]=2.200000
v1[8]=1.900000, v2[8]=0.300000, v3[8]=2.200000
v1[9]=2.000000, v2[9]=0.200000, v3[9]=2.200000
v1[10]=2.100000, v2[10]=0.100000, v3[10]=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

```
/* SumaVectoresCSections.c
Suma de dos vectores: v3 = v1 + v2
Para compilar usar (-lrt: real time library):
gcc -O2 SumaVectores.c -o SumaVectores -lrt
gcc -O2 -S SumaVectores.c -lrt //para generar el código ensamblador
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> // biblioteca donde se encuentran las funciones OMP.
//#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;
struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución
//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]); // 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


//Inicializar vectores


#pragma omp parallel // Combinamos las directivas para usar solo 1 barrera
implícita
{
    #pragma omp sections private (i)
    {
        #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; //los valores dependen de N
                printf("La hebra %d escribe V1[%d]=%f, V2[%d]=%f.\n",
omp_get_thread_num(),i,v1[i],i,v2[i]);
            }


        #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; //los valores dependen de N
                printf("La hebra %d escribe V1[%d]=%f, V2[%d]=%f.\n",
omp_get_thread_num(),i,v1[i],i,v2[i]);
            }


        #pragma omp section
            for(i=N/2; i < N*3/4; i++){
                v1[i] = N*0.1+i*0.1;
                v2[i] = N*0.1-i*0.1; //los valores dependen de N
                printf("La hebra %d escribe V1[%d]=%f, V2[%d]=%f.\n",
omp_get_thread_num(),i,v1[i],i,v2[i]);
```

```
            }


        #pragma omp section
            for(i=N*3/4; i < N; i++){
                v1[i] = N*0.1+i*0.1;
                v2[i] = N*0.1-i*0.1; //los valores dependen de N
                printf("La hebra %d escribe V1[%d]=%f, V2[%d]=%f.\n",
omp_get_thread_num(),i,v1[i],i,v2[i]);
            }


    }
}


double start = omp_get_wtime();
//Calcular suma de vectores
#pragma omp parallel
{
    #pragma omp sections private(i)
    {
        #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<N*3/4; i++)
                v3[i] = v1[i] + v2[i];




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


    }


}




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]);


if (N<=11)
    for(i=0; i<N; i++)
        printf("v1[%d]=%f, v2[%d]=%f, v3[%d]=%f\n",i,v1[i],i,v2[i],i,v3[i]);


printf("\n");
#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 (compilación y ejecución para N=8 y N=11):

N=8

```
dani@dani-Aspire-5750G:$ gcc -O2 -fopenmp -o SumaVectoresCSections SumaVectoresCSe
ctions.c
dani@dani-Aspire-5750G:$ ./SumaVectoresCSections 8
La hebra 1 escribe V1[0]=0.800000, V2[0]=0.800000.
La hebra 1 escribe V1[1]=0.900000, V2[1]=0.700000.
La hebra 0 escribe V1[4]=1.200000, V2[4]=0.400000.
La hebra 0 escribe V1[5]=1.300000, V2[5]=0.300000.
La hebra 3 escribe V1[6]=1.400000, V2[6]=0.200000.
La hebra 3 escribe V1[7]=1.500000, V2[7]=0.100000.
La hebra 2 escribe V1[2]=1.000000, V2[2]=0.600000.
La hebra 2 escribe V1[3]=1.100000, V2[3]=0.500000.
Tiempo(seg.):0.001781177         / Tama◆o Vectores:8    / V1[0]+V2[0]=V3[0](0.8000
00+0.800000=1.600000) /        V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
v1[0]=0.800000, v2[0]=0.800000, v3[0]=1.600000
v1[1]=0.900000, v2[1]=0.700000, v3[1]=1.600000
v1[2]=1.000000, v2[2]=0.600000, v3[2]=1.600000
v1[3]=1.100000, v2[3]=0.500000, v3[3]=1.600000
v1[4]=1.200000, v2[4]=0.400000, v3[4]=1.600000
v1[5]=1.300000, v2[5]=0.300000, v3[5]=1.600000
v1[6]=1.400000, v2[6]=0.200000, v3[6]=1.600000
v1[7]=1.500000, v2[7]=0.100000, v3[7]=1.600000
```

N=11

```
dani@dani-Aspire-5750G:$ gcc -O2 -fopenmp -o SumaVectoresCSections SumaVectoresCSe
ctions.c
dani@dani-Aspire-5750G:$ ./SumaVectoresCSections 11
La hebra 2 escribe V1[2]=1.300000, V2[2]=0.900000.
La hebra 2 escribe V1[3]=1.400000, V2[3]=0.800000.
La hebra 2 escribe V1[4]=1.500000, V2[4]=0.700000.
La hebra 0 escribe V1[8]=1.900000, V2[8]=0.300000.
La hebra 0 escribe V1[9]=2.000000, V2[9]=0.200000.
La hebra 0 escribe V1[10]=2.100000, V2[10]=0.100000.
La hebra 1 escribe V1[0]=1.100000, V2[0]=1.100000.
La hebra 1 escribe V1[1]=1.200000, V2[1]=1.000000.
La hebra 3 escribe V1[5]=1.600000, V2[5]=0.600000.
La hebra 3 escribe V1[6]=1.700000, V2[6]=0.500000.
La hebra 3 escribe V1[7]=1.800000, V2[7]=0.400000.
Tiempo(seg.):0.001696200         / Tama◆o Vectores:11   / V1[0]+V2[0]=V3[0](1.1000
00+1.100000=2.200000) /        V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
v1[0]=1.100000, v2[0]=1.100000, v3[0]=2.200000
v1[1]=1.200000, v2[1]=1.000000, v3[1]=2.200000
v1[2]=1.300000, v2[2]=0.900000, v3[2]=2.200000
v1[3]=1.400000, v2[3]=0.800000, v3[3]=2.200000
v1[4]=1.500000, v2[4]=0.700000, v3[4]=2.200000
v1[5]=1.600000, v2[5]=0.600000, v3[5]=2.200000
v1[6]=1.700000, v2[6]=0.500000, v3[6]=2.200000
v1[7]=1.800000, v2[7]=0.400000, v3[7]=2.200000
v1[8]=1.900000, v2[8]=0.300000, v3[8]=2.200000
v1[9]=2.000000, v2[9]=0.200000, v3[9]=2.200000
v1[10]=2.100000, v2[10]=0.100000, v3[10]=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: En el ejercicio 7 el número de hebras siempre será el máximo que deseemos, ya que la directiva for reparte las iteraciones de forma equitativa entre cuantas hebras hayamos definido en la variable de entorno OMP_NUM_THREADS.

   En el ejercicio 8 el número de hebras máximo a utilizar serán 4, ya que el programa tiene 4 directivas "section", es decir, puede asignar un total de 4 trabajos distintos como máximo a las hebras. Por tanto, esta directiva nos limita en cuanto al número máximo de hebras aprovechables.

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:

    Tiempos para el PC local:

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	0.000593	0.005200	0.001614
32768	0.000683	0.001684	0.002375

65536	0.001488	0.003854	0.001596
131072	0.000933	0.001803	0.002990
262144	0.001757	0.002541	0.002785
524288	0.003328	0.003421	0.003423
1048576	0.006816	0.009253	0.009110
2097152	0.014740	0.013377	0.011997
4194304	0.021084	0.021016	0.022376
8388608	0.039769	0.038975	0.039174
16777216	0.079547	0.077376	0.077366
33554432	0.160022	0.154197	0.154631
67108864	0.167867	0.161956	0.163290

Tiempos ATCGRID:

Nº de Componentes	**T. secuencial vect. Globales** **1 thread/core**	**T. paralelo (versión `for`)** **24 threads/cores**	**T. paralelo (versión `sections`)** **4 threads/cores**
16384	0.000096	0.000059	0.002094
32768	0.000194	0.000073	0.005258
65536	0.000369	0.001883	0.000232
131072	0.000666	0.002049	0.000305
262144	0.001345	0.000468	0.000777
524288	0.002307	0.002133	0.001248
1048576	0.005244	0.000941	0.004064
2097152	0.010520	0.002586	0.006242
4194304	0.020670	0.006701	0.013569
8388608	0.040889	0.011086	0.019414
16777216	0.081165	0.018397	0.045932
33554432	0.160310	0.036678	0.083476
67108864	0.161116	0.037802	0.079916

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.

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 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:

Nº de Componentes	Tiempo secuencial vect. Globales 1 thread/core		Tiempo paralelo/versión `for` ¿? Threads/cores	
65536	real	0m0.005s	real	0m0.006s
	user	0m0.000s	user	0m0.000s
	sys	0m0.004s	sys	0m0.016s
131072	real	0m0.006s	real	0m0.011s
	user	0m0.004s	user	0m0.028s
	sys	0m0.000s	sys	0m0.000s
262144	real	0m0.012s	real	0m0.015s
	user	0m0.008s	user	0m0.028s
	sys	0m0.000s	sys	0m0.012s
524288	real	0m0.017s	real	0m0.010s
	user	0m0.016s	user	0m0.024s
	sys	0m0.000s	sys	0m0.008s
1048576	real	0m0.023s	real	0m0.013s
	user	0m0.012s	user	0m0.028s
	sys	0m0.008s	sys	0m0.016s
2097152	real	0m0.042s	real	0m0.031s
	user	0m0.040s	user	0m0.072s
	sys	0m0.000s	sys	0m0.016s
4194304	real	0m0.062s	real	0m0.051s
	user	0m0.048s	user	0m0.108s
	sys	0m0.012s	sys	0m0.064s
8388608	real	0m0.117s	real	0m0.098s
	user	0m0.100s	user	0m0.224s

	sys	0m0.016s	sys	0m0.116s
16777216	real	0m0.201s	real	0m0.195s
	user	0m0.164s	user	0m0.444s
	sys	0m0.036s	sys	0m0.220s
33554432	real	0m0.401s	real	0m0.371s
	user	0m0.308s	user	0m0.860s
	sys	0m0.092s	sys	0m0.464s
67108864	real	0m0.440s	real	0m0.371s
	user	0m0.344s	user	0m0.844s
	sys	0m0.092s	sys	0m0.464s

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						