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): José Javier Alonso Ramos
Grupo de prácticas:A3
Fecha de entrega:
Fecha evaluación en clase:

Ejercicios basados en los ejemplos del seminario práctico

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

```c
#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());
}
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: Captura que muestre el código fuente `singleModificado.c`

CAPTURAS DE PANTALLA:

```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("Depués de la inicialización:\n");
            printf("->Hebra nº %d\n",omp_get_thread_num() );
            for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
        }
    }

    printf("\n");
}
```

```
jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º Semestre/AC/PRACTICAS/Practica_1/ejercicios$ ./singleModificado
Introduce valor de inicialización a: 15
Single ejecutada por el thread 1
Depués de la inicialización:
->Hebra nº 3
b[0] = 15       b[1] = 15       b[2] = 15       b[3] = 15       b[4] = 15       b[5] = 15       b[6] = 15       b[7] = 15       b[8] = 15
jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º Semestre/AC/PRACTICAS/Practica_1/ejercicios$ ./singleModificado
Introduce valor de inicialización a: 9
Single ejecutada por el thread 2
Depués de la inicialización:
->Hebra nº 0
b[0] = 9        b[1] = 9        b[2] = 9        b[3] = 9        b[4] = 9        b[5] = 9        b[6] = 9        b[7] = 9        b[8] = 9
jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º Semestre/AC/PRACTICAS/Practica_1/ejercicios$
```

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`

   CAPTURAS DE PANTALLA:

```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 master
        {
            printf("Depués de la inicialización:\n");
            printf("->Hebra nº %d\n",omp_get_thread_num() );
            for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
        }
    }

    printf("\n");
}
```

```
jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º Semestre/AC/PRACTICAS/Practica_1/ejercicios$ ./singleModificado
Introduce valor de inicialización a: 5
Single ejecutada por el thread 1
Depués de la inicialización:
->Hebra nº 0
b[0] = 5        b[1] = 5        b[2] = 5        b[3] = 5        b[4] = 5        b[5] = 5        b[6] = 5        b[7] = 5        b[8] = 5
jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º Semestre/AC/PRACTICAS/Practica_1/ejercicios$
jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º Semestre/AC/PRACTICAS/Practica_1/ejercicios$ ./singleModificado
Introduce valor de inicialización a: 1234
Single ejecutada por el thread 2
Depués de la inicialización:
->Hebra nº 0
b[0] = 1234     b[1] = 1234     b[2] = 1234     b[3] = 1234     b[4] = 1234     b[5] = 1234     b[6] = 1234     b[7] = 1234     b[8] = 1234
jjavier98@jjavier98-Lenovo-ideapad-310-15IKB:~/GII/2º/2º Semestre/AC/PRACTICAS/Practica_1/ejercicios$
```

RESPUESTA A LA PREGUNTA:

La única diferencia respecto al ejercicio anterior es que la hebra que muestra los resultados es siempre la nº 0 (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.

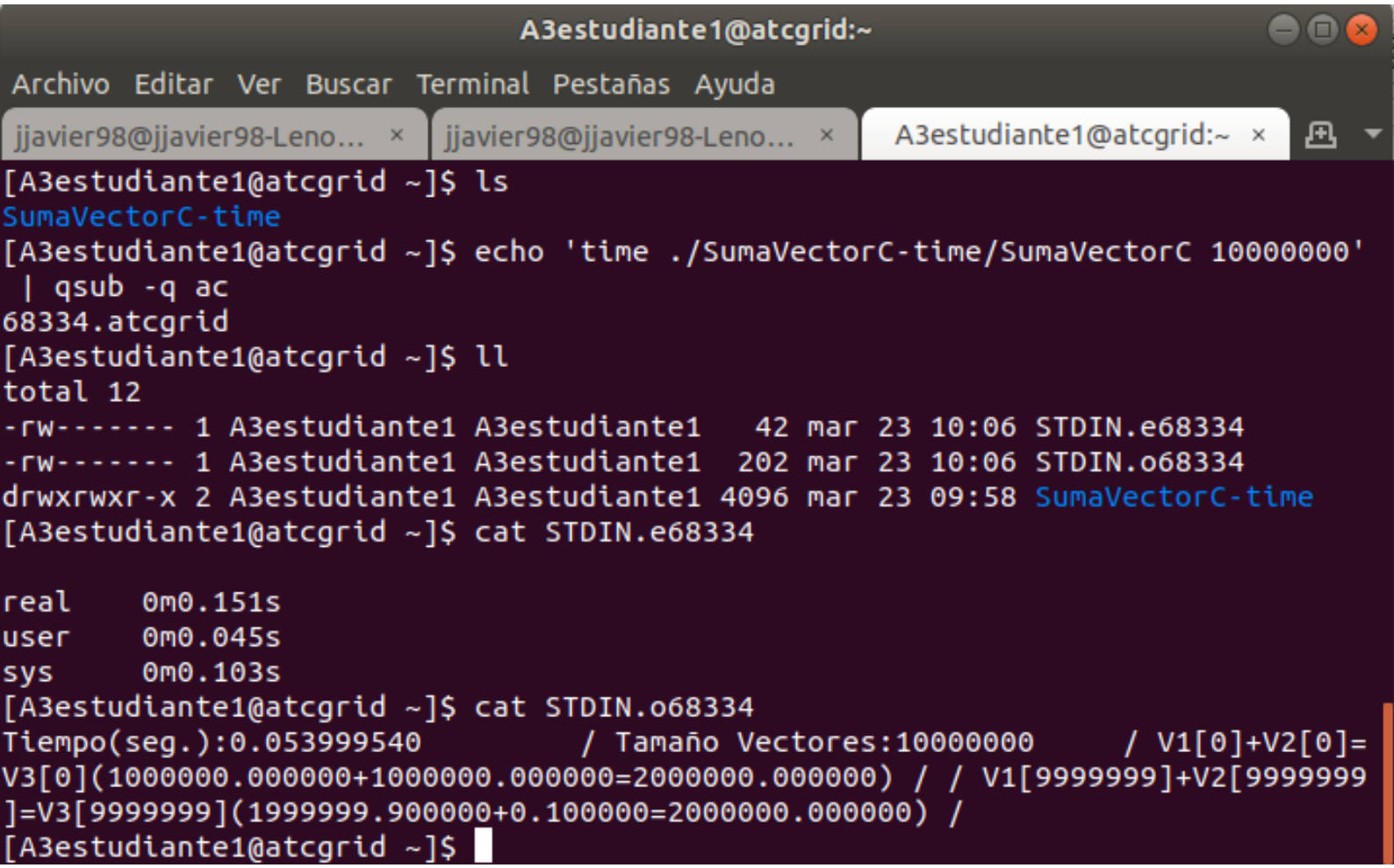
RESPUESTA:

Porque la directiva barrier retiene todas las hebras hasta que hayan completado su ejecución y hayan llegado hasta esta directiva. La siguiente instrucción la realiza la hebra 'master' según aparece en el código. Si omitimos 'barrier' y la hebra master acaba antes que cualquiera de las demás hebras imprimirá un valor de 'suma' al que todavía no se le han añadido todos los valores calculados.

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:



   La suma de tiempos de CPU de usuario y de sistema es menor que el tiempo real ya que no tenemos en cuenta el tiempo de E/S.

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:

RESPUESTA: cálculo de los MIPS y los MFLOPS

MIPS: (6 instrucciones en el bucle * N repeticiones + 3 instrucciones fuera del bucle) / (tiempo * 10^6)

MFLOPS: (3 instrucciones en el bucle * N repeticiones + 0 instrucciones fuera del bucle) / (tiempo * 10^6)

```
        call    clock_gettime@PLT
        xorl    %eax, %eax
        .p2align 4,,10
        .p2align 3
.L5:
        movsd   (%r12,%rax,8), %xmm0
        addsd   0(%rbp,%rax,8), %xmm0
        movsd   %xmm0, 0(%r13,%rax,8)
        addq    $1, %rax
        cmpl    %eax, %ebx
        ja      .L5
.L6:
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
```

- N=10

  Tiempo calculado: 0.000002822s

  MIPS = (6*10+3)/(0.000002822 * 10^6) = 22,32459249

  MFLOPS = (3*10)/(0.000002822 * 10^6) = 10,63075833

- N=10.000.000

  Tiempo calculado: 0.051715503s

  MIPS = (6*10.000.000+3)/(0.051715503 * 10^6) = 1160,193743

  MFLOPS = (3*10.000.000)/(0.051715503 * 10^6) = 580,0968425

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

```
[A3estudiante1@atcgrid SumaVectorC-time]$ echo './SumaVectorC-time/SumaVectorC 10000000' | qsub -q ac
68364.atcgrid
[A3estudiante1@atcgrid SumaVectorC-time]$ ll
total 20
-rw------- 1 A3estudiante1 A3estudiante1     0 mar 23 10:47 STDIN.e68364
-rw------- 1 A3estudiante1 A3estudiante1   202 mar 23 10:47 STDIN.o68364
-rwxr-xr-x 1 A3estudiante1 A3estudiante1 12712 mar 23 09:55 SumaVectorC
[A3estudiante1@atcgrid SumaVectorC-time]$ cat STDIN.o68364
Tiempo(seg.):0.051715503         / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) / / V1[9999999]
+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /
[A3estudiante1@atcgrid SumaVectorC-time]$ ^C
[A3estudiante1@atcgrid SumaVectorC-time]$ echo './SumaVectorC-time/SumaVectorC 10' | qsub -q ac
68365.atcgrid
[A3estudiante1@atcgrid SumaVectorC-time]$ ll
total 24
-rw------- 1 A3estudiante1 A3estudiante1     0 mar 23 10:47 STDIN.e68364
-rw------- 1 A3estudiante1 A3estudiante1     0 mar 23 10:47 STDIN.e68365
-rw------- 1 A3estudiante1 A3estudiante1   202 mar 23 10:47 STDIN.o68364
-rw------- 1 A3estudiante1 A3estudiante1   148 mar 23 10:47 STDIN.o68365
-rwxr-xr-x 1 A3estudiante1 A3estudiante1 12712 mar 23 09:55 SumaVectorC
[A3estudiante1@atcgrid SumaVectorC-time]$ cat STDIN.o68365
Tiempo(seg.):0.000002822         / Tamaño 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) /
[A3estudiante1@atcgrid SumaVectorC-time]$
```

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):
 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 Open MP
//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);
    }
```

```
37      unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
38      #ifdef VECTOR_LOCAL
39      double v1[N], v2[N], v3[N]; // Tamaño variable local en tiempo de ejecución ...
40      // disponible en C a partir de actualización C99
41      #endif
42      #ifdef VECTOR_GLOBAL
43      if (N>MAX) N=MAX;
44      #endif
45      #ifdef VECTOR_DYNAMIC
46      double *v1, *v2, *v3;
47      v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño en bytes
48      v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc devuelve NULL
49      v3 = (double*) malloc(N*sizeof(double));
50      if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
51      printf("Error en la reserva de espacio para los vectores\n");
52      exit(-2);
53      }
54      #endif
55
56      //Inicializar vectores
57      #pragma omp parallel for
58          for(i=0; i<N; i++)
59          {
60              v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de N
61          }
62
63      cgt1 = omp_get_wtime();
64      //Calcular suma de vectores
65      #pragma omp parallel for
66          for(i=0; i<N; i++)
67              v3[i] = v1[i] + v2[i];
68
69      cgt2 = omp_get_wtime();
70      ncgt= cgt2 - cgt1;
71
72      //Imprimir resultado de la suma y el tiempo de ejecución
73      if(N < 20)
74      {
75          printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
76          for(i=0; i<N; i++)
77          printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",i,i,i,v1[i],v2[i],v3[i]);
78
79          printf("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",
80                  v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
81      }
82      else
83      {
84          printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\t/ V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) / /
85           V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
86                  ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
87      }
88
89      #ifdef VECTOR_DYNAMIC
90      free(v1); // libera el espacio reservado para v1
91      free(v2); // libera el espacio reservado para v2
92      free(v3); // libera el espacio reservado para v3
93      #endif
94      return 0;
95  }
96
```

```
[A3estudiante1@atcgrid ~]$ ls
SumaVectorC_omp
[A3estudiante1@atcgrid ~]$ echo './SumaVectorC_omp 10' | qsub -q ac
68381.atcgrid
[A3estudiante1@atcgrid ~]$ ll
total 20
-rw------- 1 A3estudiante1 A3estudiante1     0 mar 23 11:40 STDIN.e68381
-rw------- 1 A3estudiante1 A3estudiante1   646 mar 23 11:40 STDIN.o68381
-rwxr-xr-x 1 A3estudiante1 A3estudiante1 13184 mar 23 11:38 SumaVectorC_omp
[A3estudiante1@atcgrid ~]$ cat STDIN.o68381
Tiempo(seg.):0.003818204         / Tamaño Vectores:10
/ V1[0]+V2[0]=V3[0](1.000000+1.000000=2.000000) /
/ V1[1]+V2[1]=V3[1](1.100000+0.900000=2.000000) /
/ V1[2]+V2[2]=V3[2](1.200000+0.800000=2.000000) /
/ V1[3]+V2[3]=V3[3](1.300000+0.700000=2.000000) /
/ V1[4]+V2[4]=V3[4](1.400000+0.600000=2.000000) /
/ V1[5]+V2[5]=V3[5](1.500000+0.500000=2.000000) /
/ V1[6]+V2[6]=V3[6](1.600000+0.400000=2.000000) /
/ V1[7]+V2[7]=V3[7](1.700000+0.300000=2.000000) /
/ V1[8]+V2[8]=V3[8](1.800000+0.200000=2.000000) /
/ V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /
V1[0]+V2[0]=V3[0](1.000000+1.000000=2.000000) / / V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /
[A3estudiante1@atcgrid ~]$ echo './SumaVectorC_omp 10000000' | qsub -q ac
68385.atcgrid
[A3estudiante1@atcgrid ~]$ ll
total 24
-rw------- 1 A3estudiante1 A3estudiante1     0 mar 23 11:40 STDIN.e68381
-rw------- 1 A3estudiante1 A3estudiante1     0 mar 23 11:44 STDIN.e68385
-rw------- 1 A3estudiante1 A3estudiante1   646 mar 23 11:40 STDIN.o68381
-rw------- 1 A3estudiante1 A3estudiante1   202 mar 23 11:44 STDIN.o68385
-rwxr-xr-x 1 A3estudiante1 A3estudiante1 13184 mar 23 11:38 SumaVectorC_omp
[A3estudiante1@atcgrid ~]$ cat STDIN.o68385
Tiempo(seg.):0.016245375         / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) / / V1[9999999]
+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000) /
[A3estudiante1@atcgrid ~]$
[A3estudiante1@atcgrid ~]$ echo './SumaVectorC_omp 8' | qsub -q ac
68387.atcgrid
[A3estudiante1@atcgrid ~]$ cat STDIN.o68387
Tiempo(seg.):0.004513159         / Tamaño Vectores:8
/ V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) /
/ V1[1]+V2[1]=V3[1](0.900000+0.700000=1.600000) /
/ V1[2]+V2[2]=V3[2](1.000000+0.600000=1.600000) /
/ V1[3]+V2[3]=V3[3](1.100000+0.500000=1.600000) /
/ V1[4]+V2[4]=V3[4](1.200000+0.400000=1.600000) /
/ V1[5]+V2[5]=V3[5](1.300000+0.300000=1.600000) /
/ V1[6]+V2[6]=V3[6](1.400000+0.200000=1.600000) /
/ V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) / / V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
[A3estudiante1@atcgrid ~]$ echo './SumaVectorC_omp 11' | qsub -q ac
68388.atcgrid
[A3estudiante1@atcgrid ~]$ cat STDIN.o68388
Tiempo(seg.):0.004894238         / Tamaño Vectores:11
/ V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) /
/ V1[1]+V2[1]=V3[1](1.200000+1.000000=2.200000) /
/ V1[2]+V2[2]=V3[2](1.300000+0.900000=2.200000) /
/ V1[3]+V2[3]=V3[3](1.400000+0.800000=2.200000) /
/ V1[4]+V2[4]=V3[4](1.500000+0.700000=2.200000) /
/ V1[5]+V2[5]=V3[5](1.600000+0.600000=2.200000) /
/ V1[6]+V2[6]=V3[6](1.700000+0.500000=2.200000) /
/ V1[7]+V2[7]=V3[7](1.800000+0.400000=2.200000) /
/ V1[8]+V2[8]=V3[8](1.900000+0.300000=2.200000) /
/ V1[9]+V2[9]=V3[9](2.000000+0.200000=2.200000) /
/ V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) / / V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
[A3estudiante1@atcgrid ~]$
```

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).

```c
/* SumaVectoresC.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 Open MP
//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);
    }
```

RESPUESTA: Captura que muestre el código fuente implementado

```
37      unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)
38      #ifdef VECTOR_LOCAL
39      double v1[N], v2[N], v3[N]; // Tamaño variable local en tiempo de ejecución ...
40      // disponible en C a partir de actualización C99
41      #endif
42      #ifdef VECTOR_GLOBAL
43      if (N>MAX) N=MAX;
44      #endif
45      #ifdef VECTOR_DYNAMIC
46      double *v1, *v2, *v3;
47      v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño en bytes
48      v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio suficiente malloc devuelve NULL
49      v3 = (double*) malloc(N*sizeof(double));
50      if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
51      printf("Error en la reserva de espacio para los vectores\n");
52      exit(-2);
53      }
54      #endif
55
56      //Inicializar vectores
57      #pragma omp parallel sections
58      {
59          #pragma omp section
60          {
61              for(i=0; i<N/4; i++)
62              {
63                  v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de N
64              }
65          }
66          #pragma omp section
67          {
68              for(i=N/4; i<N/2; i++)
69              {
70                  v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de N
71              }
```

```
72          }
73          #pragma omp section
74          {
75              for(i=N/2; i<3*N/4; i++)
76              {
77                  v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de N
78              }
79          }
80          #pragma omp section
81          {
82              for(i=3*N/4; i<N; i++)
83              {
84                  v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de N
85              }
86          }
87      }
88
89      cgt1 = omp_get_wtime();
90      //Calcular suma de vectores
91      #pragma omp parallel sections
92      {
93          #pragma omp section
94          {
95              for(i=0; i<N/4; i++)
96                  v3[i] = v1[i] + v2[i];
97          }
98          #pragma omp section
99          {
100             for(i=N/4; i<N/2; i++)
101                 v3[i] = v1[i] + v2[i];
102         }
103         #pragma omp section
104         {
105             for(i=N/2; i<3*N/4; i++)
106                 v3[i] = v1[i] + v2[i];
107         }
```

```
108         #pragma omp section
109         {
110             for(i=3*N/4; i<N; i++)
111                 v3[i] = v1[i] + v2[i];
112         }
113     }
114
115     cgt2 = omp_get_wtime();
116     ncgt= cgt2 - cgt1;
117
118     //Imprimir resultado de la suma y el tiempo de ejecución
119     if(N < 20)
120     {
121         printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
122         for(i=0; i<N; i++)
123         printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",i,i,i,v1[i],v2[i],v3[i]);
124
125         printf("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",
126                 v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
127     }
128     else
129     {
130         printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\t/ V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) / /
131         V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
132                 ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
133     }
134
135     #ifdef VECTOR_DYNAMIC
136     free(v1); // libera el espacio reservado para v1
137     free(v2); // libera el espacio reservado para v2
138     free(v3); // libera el espacio reservado para v3
139     #endif
140     return 0;
141 }
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
[A3estudiante1@atcgrid ~]$ echo './SumaVectorC_sections 11' | qsub -q ac
68407.atcgrid
[A3estudiante1@atcgrid ~]$ cat STDIN.o68407
Tiempo(seg.):0.003839001        / Tamaño Vectores:11
/ V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) /
/ V1[1]+V2[1]=V3[1](1.200000+1.000000=2.200000) /
/ V1[2]+V2[2]=V3[2](1.300000+0.900000=2.200000) /
/ V1[3]+V2[3]=V3[3](1.400000+0.800000=2.200000) /
/ V1[4]+V2[4]=V3[4](1.500000+0.700000=2.200000) /
/ V1[5]+V2[5]=V3[5](1.600000+0.600000=2.200000) /
/ V1[6]+V2[6]=V3[6](1.700000+0.500000=2.200000) /
/ V1[7]+V2[7]=V3[7](1.800000+0.400000=2.200000) /
/ V1[8]+V2[8]=V3[8](1.900000+0.300000=2.200000) /
/ V1[9]+V2[9]=V3[9](2.000000+0.200000=2.200000) /
/ V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) / / V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
[A3estudiante1@atcgrid ~]$ echo './SumaVectorC_sections 8' | qsub -q ac
68408.atcgrid
[A3estudiante1@atcgrid ~]$ cat STDIN.o68408
Tiempo(seg.):0.004174888        / Tamaño Vectores:8
/ V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) /
/ V1[1]+V2[1]=V3[1](0.900000+0.700000=1.600000) /
/ V1[2]+V2[2]=V3[2](1.000000+0.600000=1.600000) /
/ V1[3]+V2[3]=V3[3](1.100000+0.500000=1.600000) /
/ V1[4]+V2[4]=V3[4](1.200000+0.400000=1.600000) /
/ V1[5]+V2[5]=V3[5](1.300000+0.300000=1.600000) /
/ V1[6]+V2[6]=V3[6](1.400000+0.200000=1.600000) /
/ V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) / / V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
[A3estudiante1@atcgrid ~]$ echo './SumaVectorC_sections 100' | qsub -q ac
68410.atcgrid
[A3estudiante1@atcgrid ~]$ cat STDIN.o68410
Tiempo(seg.):0.004037254        / Tamaño Vectores:100  / V1[0]+V2[0]=V3[0](10.000000+10.000000=20.000000) / / V1[99]+V2[99]=V3[99](19.900000+0.1
00000=20.000000) /
[A3estudiante1@atcgrid ~]$
```

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 threads/cores que se puede utilizar es ilimitado y dependerá de la máquina. Podrá usar todos los threads/cores de los que disponga. La directiva *parallel for* no limita el número de hebras y las distribuye dinámicamente.
   - En el ejercicio 8, según mi implementación, el máximo de hebras que se pueden utilizar es 4 ya que cada directiva *sections* incluye 4 directivas *section* las cuales ejecuta 1 hebra cada una.

10. Rellenar una tabla como la Tabla 2 para atcgrid y otra para su PC con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución del trozo de código que realiza la suma en paralelo (este es el tiempo que deben imprimir los programas). Ponga en la tabla el número de threads/cores que usan los códigos. Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute código que imprima todos los componentes del resultado cuando este número sea elevado.

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	0.000283002	0.000058530	0.000028117
32768	0.000562442	0.000075205	0.000235967
65536	0.000946665	0.000431233	0.000447513
131072	0.000480507	0.000245134	0.000215492
262144	0.005106932	0.000651222	0.002872958
524288	0.002102740	0.001344627	0.001164048
1048576	0.005972728	0.002545664	0.024511504
2097152	0.010535858	0.016737670	0.015492773
4194304	0.018779638	0.018031620	0.016994959
8388608	0.032209643	0.020513873	0.020860382
16777216	0.063350831	0.038002034	0.038286754
33554432	0.126497402	0.074421012	0.074707491
67108864	0.126162005	0.166833607	0.149476369

Tabla de mi PC

Tabla 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	0.000110585	0.006748580	0.004342328
32768	0.000212657	0.004726598	0.003839270
65536	0.000293419	0.003968476	0.003944567
131072	0.000561247	0.003982289	0.003868929
262144	0.001150001	0.004057179	0.002759937
524288	0.002465538	0.003004980	0.003742654
1048576	0.006298575	0.004842584	0.005110522
2097152	0.012614323	0.005099161	0.006521923
4194304	0.023929901	0.008727039	0.012175326
8388608	0.050084908	0.012132419	0.020811721
16777216	0.113783645	0.023883984	0.041757095
33554432	0.216906585	0.053959887	0.093371272
67108864	0.222051974	0.117788407	0.181550450

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:
En la suma secuencial el tiempo real es ligeramente mayor debido a los tiempos de E/S.
En cambio en la suma paralela el tiempo real es mucho menor debido a la creación de hebras.

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` 24 Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	0m0.004s	0m0.000s	0m0.003s	0m0.014s	0m0.191s	0m0.003s
131072	0m0.004s	0m0.001s	0m0.003s	0m0.012s	0m0.113s	0m0.015s
262144	0m0.007s	0m0.003s	0m0.004s	0m0.016s	0m0.215s	0m0.019s
524288	0m0.011s	0m0.005s	0m0.006s	0m0.014s	0m0.205s	0m0.020s
1048576	0m0.022s	0m0.007s	0m0.015s	0m0.017s	0m0.247s	0m0.031s
2097152	0m0.040s	0m0.010s	0m0.029s	0m0.021s	0m0.256s	0m0.069s
4194304	0m0.073s	0m0.018s	0m0.055s	0m0.033s	0m0.261s	0m0.261s
8388608	0m0.144s	0m0.047s	0m0.096s	0m0.046s	0m0.403s	0m0.240s
16777216	0m0.288s	0m0.102s	0m0.184s	0m0.081s	0m0.583s	0m0.522s
33554432	0m0.579s	0m0.216s	0m0.358s	0m0.153s	0m1.084s	0m1.283s
67108864	0m0.592s	0m0.221s	0m0.366s	0m0.320s	0m2.067s	0m2.269s