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): David Gil Bautista
Grupo de prácticas: C1
Fecha de entrega:
Fecha evaluación en clase:

Ejercicios basados en los ejemplos del seminario práctico

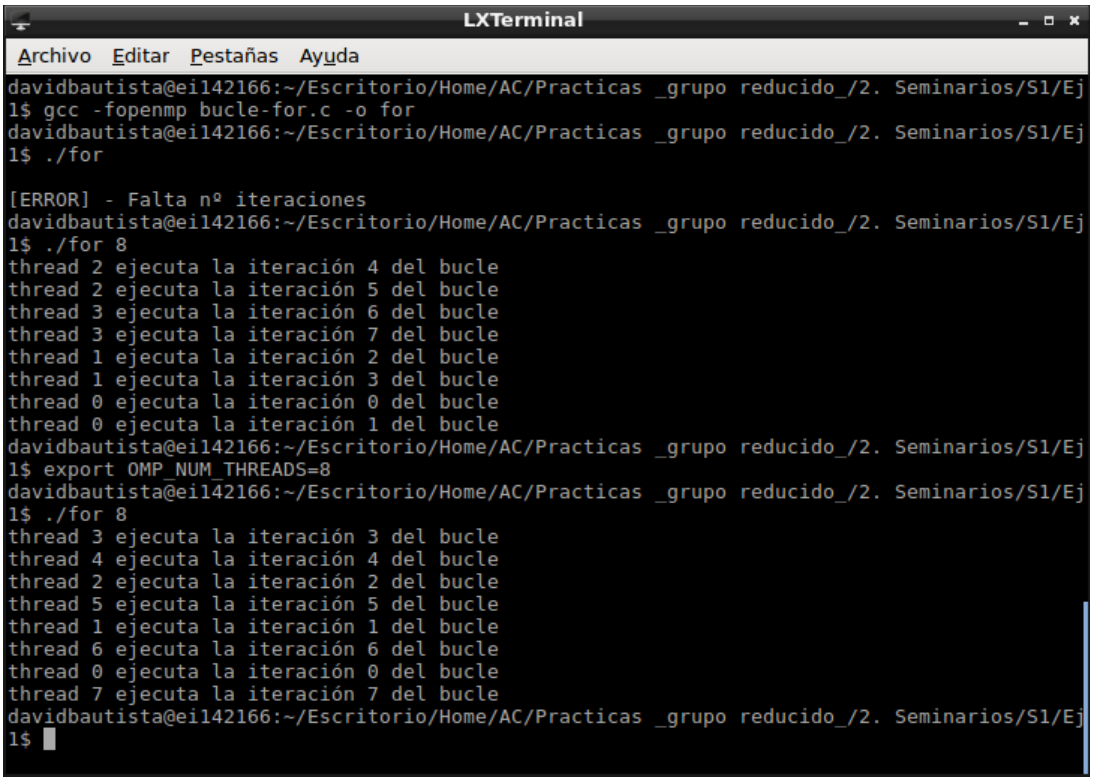
. Usar la directiva `parallel` combinada con directivas de trabajo compartido en los ejemplos `bucle-for.c` y `sections.c` del seminario. Incorporar el código fuente resultante al cuaderno de prácticas.

RESPUESTA: código fuente `bucle-forModificado.c`

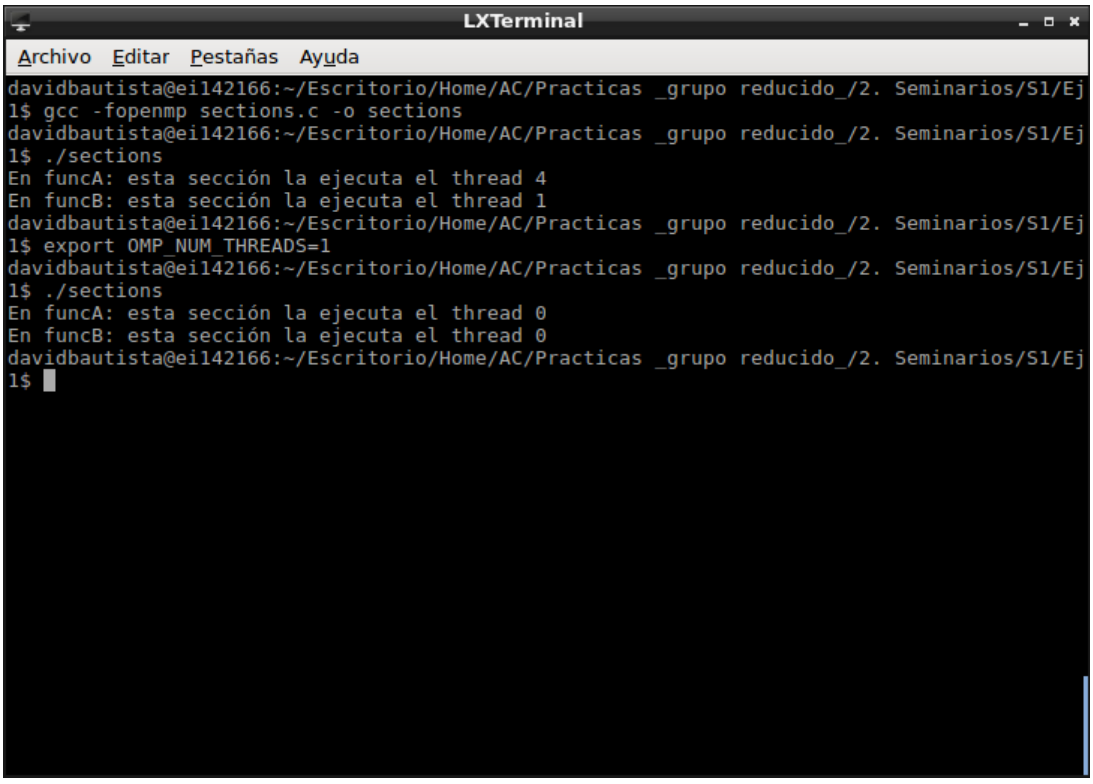
```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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



int main(int argc, char ** argv)
{
   int i, n = 8;

   if(argc < 2)     {
      printf("No. Argumentos incorrecto, uso; ./for nIteraciones");
      exit(-1);
   }
   n = atoi(argv[1]);

#pragma omp parallel
{
   #pragma omp 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`

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

void func1(){ // ... }
void func2(){ // ... }
void funcN(){ // ... }

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

#pragma omp parallel
{
     #pragma omp sections
     {
         #pragma omp section
            (void) func1();
         #pragma omp section
            (void) func2();

         //     ...

         #pragma omp section
            (void) funcN();
     }
}
   return 0;

}
```



Bucle for modificado



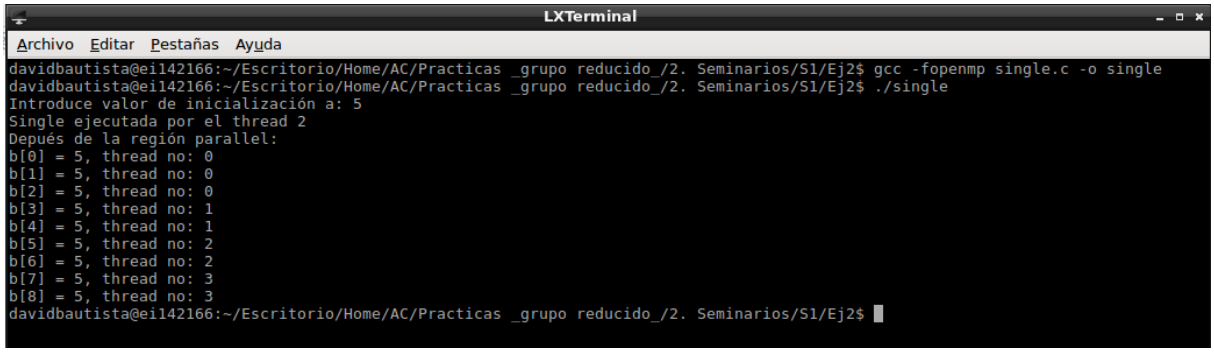
Sections modificado

. 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`

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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



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

  int n = 9, i, a;
  int b[n];
  int t[n];

  for (i=0; i<n; i++)   b[i] = -3;
  #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;
             t[i] = omp_get_thread_num();
     }

  }
  printf("Depués de la región parallel:\n");
  for (i=0; i<n; i++){
             printf("b[%d] = %d, thread no: %d\t",i,b[i],t[i]);
             printf("\n");
  }
  return 0;

}
```

CAPTURAS DE PANTALLA:



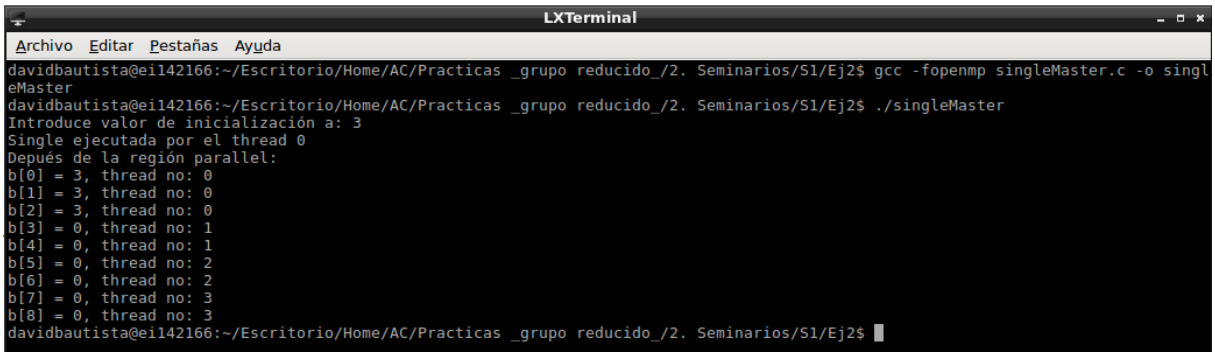
. Imprimir los resultados del programa `single.c` usando una directiva `master` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `master` incorporada, para que se imprima el identificador del thread que ejecuta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

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

int main(int argc, char ** argv)
{
  int n = 9, i, a;
  int b[n];
  int t[n];

  for (i=0; i<n; i++)   b[i] = -3;
  #pragma omp parallel
  {

     #pragma omp master
     {  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;
             t[i] = omp_get_thread_num();
     }

  }
  printf("Depués de la región parallel:\n");
  for (i=0; i<n; i++){
```

```
            printf("b[%d] = %d, thread no: %d\t",i,b[i],t[i]);
            printf("\n");
  }
  return 0;
}
```

CAPTURAS DE PANTALLA:



RESPUESTA A LA PREGUNTA: La hebra principal siempre ejecuta la sección de **#pragma omp master.**

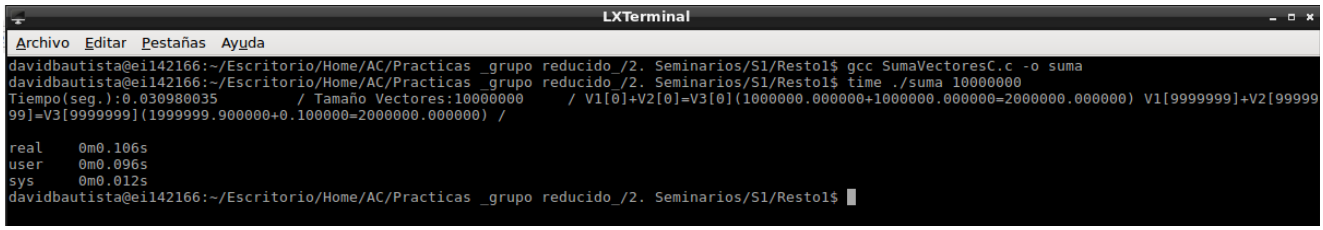
. ¿Por qué si se elimina directiva `barrier` en el ejemplo `master.c` la suma que se calcula e imprime no siempre es correcta? Responda razonadamente.

RESPUESTA: Porque las hebras pueden seguir ejecutándose. Barrier hace que todas las hebras se detengan en un punto y luego que continúen una vez hayan llegado todas.

Resto de ejercicios

. El programa secuencial C del Listado 1 calcula la suma de dos vectores (v3 = v1 + v2; v3(i) = v1(i) + v2(i), i=0,…N-1). Generar el ejecutable del programa del Listado 1 para **vectores globales**. Usar `time` (Lección 3/ Tema 1) en la línea de comandos para obtener, en el PC local, el tiempo de ejecución (*elapsed time*) y el tiempo de CPU del usuario y del sistema generado. Obtenga los tiempos para vectores con 10000000 componentes. ¿La suma de los tiempos de CPU del usuario y del sistema es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

CAPTURAS DE PANTALLA:



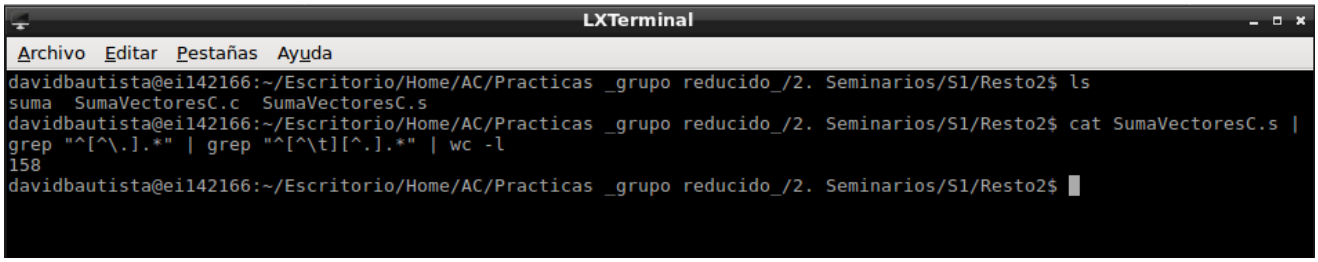
No, ya que el real es el tiempo de usuario (programa) + sistema (llamadas) + el tiempo de la esperas de las entradas y salidas.

. 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

RESPUESTA:



N = 158 instrucciones

Vector de 10 elementos.

Vector de 10000000 elementos.

MIPS = NI / Tcpu*10^6

Para 10 elementos.

MIPS = 158*10/0,003*10^6 = 0,526 MIPS

Para 10 000 000 elementos.

MIPS = 158*10^7/0,218*10^6 = 7,247 * 10^3 MIPS

MFLOPS = Float Ops. / Tcpu * 10^6

Para 10 componentes

MFLOPS = OpFloat / (TCPU * 106) = 6*10 / (0.000003240 * 106) = 18,5185185185

Para 10000000 componentes

MFLOPS = OpFloat / (TCPU * 106) = 6*10000000 / (0.047347371 * 106) = 1267,22981092 MFLOPS

Código ensamblador generado de la parte de la suma de vectores: **Resto2/SumaVectoresC.s**

. 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).

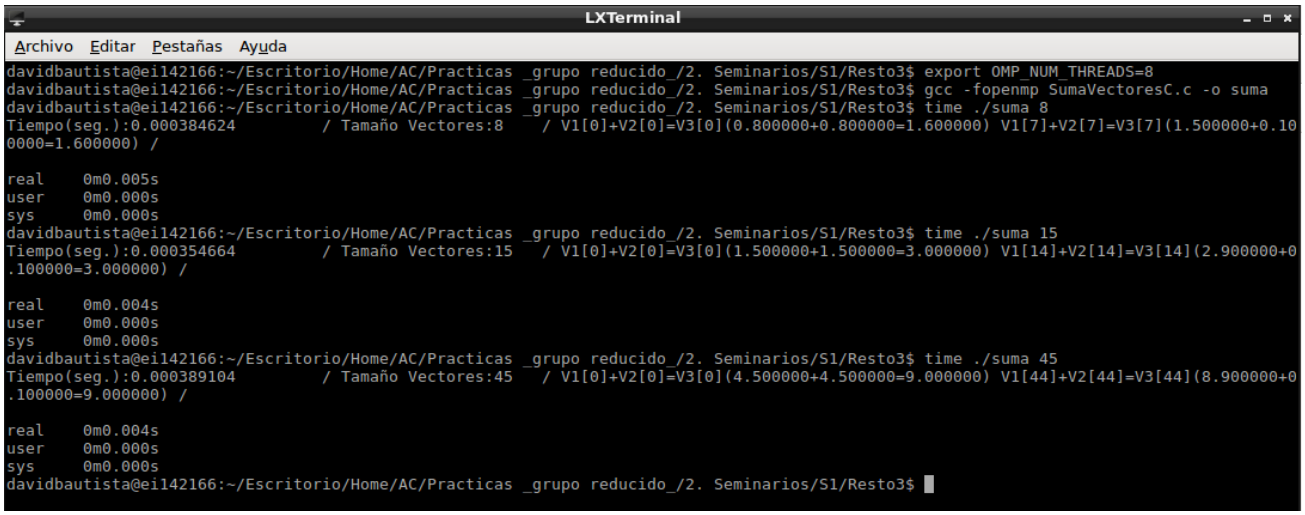
RESPUESTA: código fuente implementado

./Resto4/suma.c

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

```
#include <omp.h>

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

CAPTURAS DE PANTALLA :



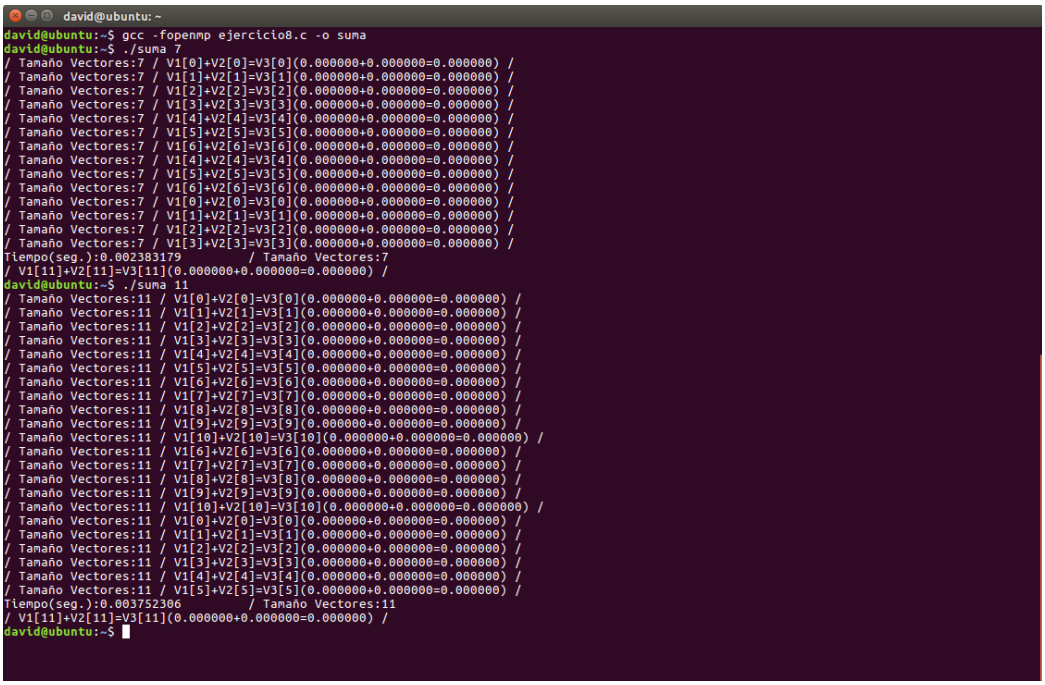
. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores usando las `parallel` y `sections/section` (se debe aprovechar el paralelismo de datos usando estas directivas en lugar de la directiva `for`); es decir, hay que repartir el trabajo (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 → resto5/suma2.c

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

int main(int argc, char ** argv)
{
 unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)

             #pragma omp sections
             {

             #pragma omp section
                           for(i=0; i<N/4; i++){
                                        v3[i] = v1[i] + v2[i];
                                        printf("/ Tamaño Vectores:%u / V1[%d]
+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",N,i,i,i,v1[i],v2[i],v3[i]);
                           }
             #pragma omp section
                           for(i=N/4; i<N/3; i++){
                                        v3[i] = v1[i] + v2[i];
                                        printf("/ Tamaño Vectores:%u / V1[%d]
+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",N,i,i,i,v1[i],v2[i],v3[i]);
                           }

             #pragma omp section
                           for(i=N/3; i<N/2; i++){
                                        v3[i] = v1[i] + v2[i];
                                        printf("/ Tamaño Vectores:%u / V1[%d]
+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",N,i,i,i,v1[i],v2[i],v3[i]);
                           }

             #pragma omp section
                           for(i=N/2; i<N; i++){
                                        v3[i] = v1[i] + v2[i];
                                        printf("/ Tamaño Vectores:%u / V1[%d]
+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",N,i,i,i,v1[i],v2[i],v3[i]);
                           }

             }

double start, end;
  start=  omp_get_wtime();

 //Calcular suma de vectores
 #pragma omp parallel for
 for(i=0; i<N; i++){
 v3[i] = v1[i] + v2[i];
printf("/ Tamaño Vectores:%u / V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f)
/\n",N,i,i,i,v1[i],v2[i],v3[i]);
}

 end= omp_get_wtime();

}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA (compilación y ejecución para N=7 y N=11):



. ¿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: El *for* utiliza todas las threads que tenga el ordenador, mientras que el sections ejecuta tantas hebras como sections haya dividiendo el código, en mi caso 4.

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

RESPUESTA:

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

Nº de Componentes	**T. secuencial vect. Globales** **1 thread/core**	**T. paralelo (versión `for`)** **8 threads/cores**	**T. paralelo (versión `sections`)** **4 threads/cores**
16384	0.00101473	0.008008047	0.009622917
32768	0.00195717	0.008139227	0.011590914
65536	0.00396539	0.008860120	0.008300863
131072	0.00799729	0.007474956	0.009608031
262144	0.01551490	0.008546875	0.006906714
524288	0.02898526	0.007131474	0.006749673
1048576	0.05500206	0.009154747	0.008595951
2097152	0.10014429	0.011886034	0.006093693
4194304	0.20116821	0.015038341	0.011894468
8388608	0.42606206	0.024233161	0.022358800
16777216	0.81666607	0.033663495	0.028673174
33554432	1.58284837	0.057203881	0.050205605

. 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: Menor ya que el tiempo real es una composición del tiempo real más otros tiempos.

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	0m0.002s	0m0.000s	0m0.000s	0m0.006s	0m0.0028s	0m0.000s
131072	0m0.002s	0m0.000s	0m0.000s	0m0.006s	0m0.028s	0m0.000s
262144	0m0.003s	0m0.000s	0m0.000s	0m0.008s	0m0.012s	0m0.012s
524288	0m0.006s	0m0.004s	0m0.000s	0m0.009s	0m0.000s	0m0.036s
1048576	0m0.009s	0m0.004s	0m0.004s	0m0.013s	0m0.028s	0m0.028s
2097152	0m0.019s	0m0.008s	0m0.008s	0m0.023s	0m0.080s	0m0.000s
4194304	0m0.038s	0m0.028s	0m0.008s	0m0.047s	0m0.148s	0m0.020s
8388608	0m0.069s	0m0.048s	0m0.032s	0m0.081s	0m0.232s	0m0.052s
16777216	0m0.121s	0m0.088s	0m0.024s	0m0.142s	0m0.484s	0m0.080s
33554432	0m0.231s	0m0.180s	0m0.092s	0m0.269s	0m0.880s	0m0.196s
67108864	0m0.231s	0m0.180s	0m0.052s	0m0.240s	0m0.060s	0m0.068s