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): Juan Miguel Castro Guerrero
Grupo de prácticas: B3
Fecha de entrega: 01/04/2014
Fecha evaluación en clase:

---

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();
    }
}
```

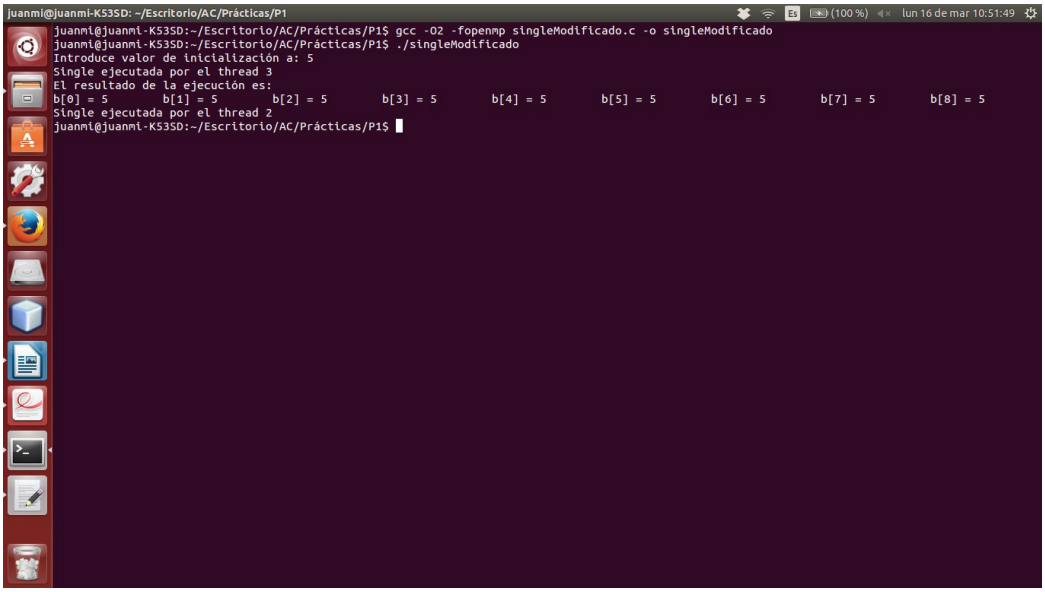
2. Imprimir los resultados del programa single.c usando una directiva single dentro de la construcción parallel en lugar de imprimirlos fuera de la región parallel. Añadir lo necesario, dentro de la nueva directiva single incorporada, para que se imprima el identificador del thread que ejecuta el bloque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


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("El resultado de la ejecución es:\n");
                for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
```

```
                printf("\n");
                printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
        }
    }
}
```

CAPTURAS DE PANTALLA:



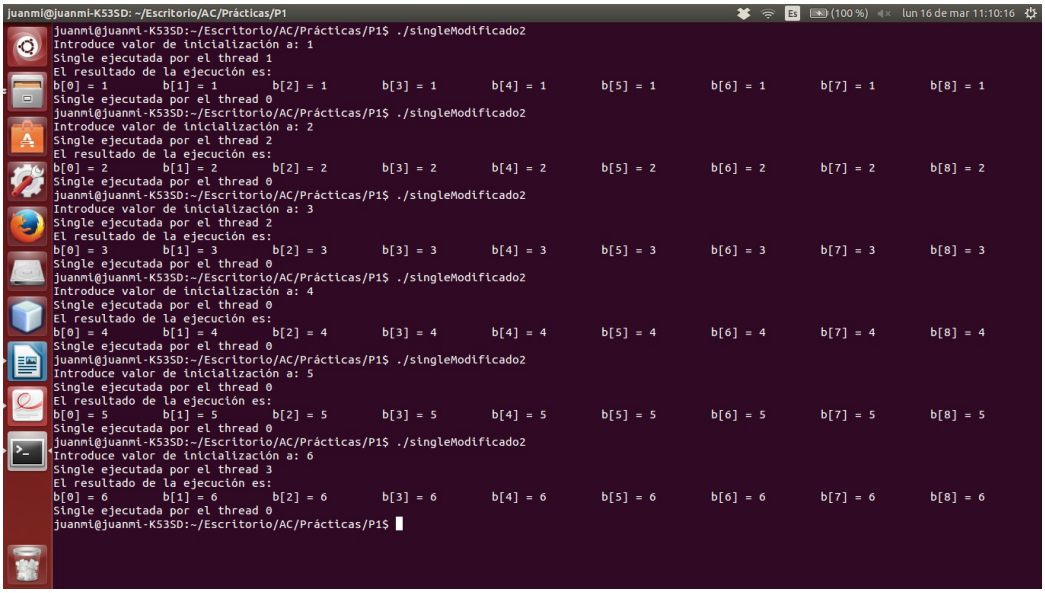
3. Imprimir los resultados del programa single.c usando una directiva master dentro de la construcción parallel en lugar de imprimirlos fuera de la región parallel. Añadir lo necesario, dentro de la nueva directiva master incorporada, para que se imprima el identificador del thread que ejecuta el bloque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


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("El resultado de la ejecución es:\n");
                for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
                printf("\n");
                printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
        }
    }
}
```

CAPTURAS DE PANTALLA:



RESPUESTA A LA PREGUNTA: La principal diferencia se encuentra en que la hebra padre, es decir, la hebra 0 siempre ejecuta el fragmento asociado a la directiva 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 master no tiene barreras implícitas por lo que necesita una directiva barrier para que se esperen los resultados de ejecución de las demás hebras y poder ejecutar correctamente el código asociado a la directiva master, en este caso la suma de todas las hebras.

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 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.

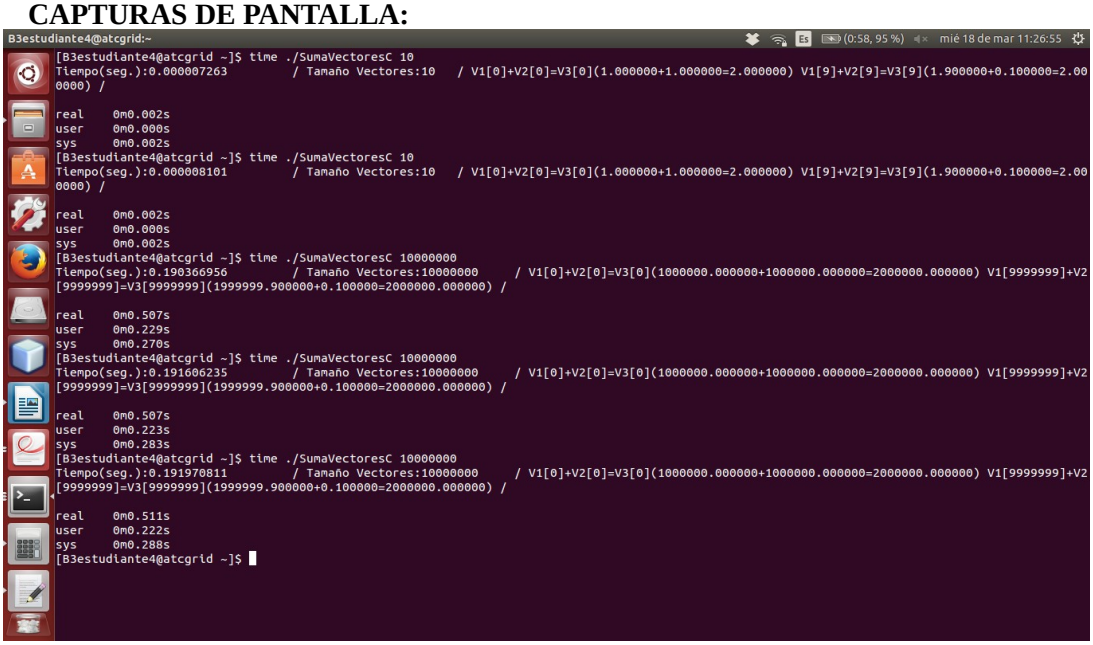
CAPTURAS DE PANTALLA:

RESPUESTA: La suma de de los tiempos de CPU de usuario y del sistema es igual al tiempo real.

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.

RESPUESTA: El código correspondiente entre las funciones clock_gettime() en ensamblador se compone de 11 instrucciones. Para vectores de tamaño 10 en atcgrid se tardan 0,002 segundos en la ejecución de 47 instrucciones por lo que serían 0,0235 MIPS. Para vectores de tamaño 10000000 se tardan 0,5 segundos en la ejecución de 40000007 intrucciones por lo que serían 80,000015 MIPS

CAPTURAS DE PANTALLA:



RESPUESTA:

código ensamblador generado de la parte de la suma de vectores

```
                .file   "SumaVectoresC.c"
                .section        .rodata.str1.8,"aMS",@progbits,1
                .align 8
.LC0:
                .string "Faltan n\302\272 componentes del vector"
                .align 8
.LC3:
                .string "Tiempo(seg.):%11.9f\t / Tama\303\261o 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"
                .section        .text.startup,"ax",@progbits
                .p2align 4,,15
                .globl  main
                .type   main, @function
main:
.LFB39:
                .cfi_startproc
                pushq   %r12
                .cfi_def_cfa_offset 16
                .cfi_offset 12, -16
                pushq   %rbp
                .cfi_def_cfa_offset 24
                .cfi_offset 6, -24
                pushq   %rbx
                .cfi_def_cfa_offset 32
                .cfi_offset 3, -32
                subq    $32, %rsp
                .cfi_def_cfa_offset 64
                cmpl    $1, %edi
```

```
                jle     .L11
                movq    8(%rsi), %rdi
                movl    $10, %edx
                xorl    %esi, %esi
                movl    $33554432, %ebp
                call    strtol
                cmpl    $33554432, %eax
                cmovbe  %eax, %ebp
                testl   %ebp, %ebp
                je      .L3
                cvtsi2sd        %ebp, %xmm1
                leal    -1(%rbp), %r12d
                movsd   .LC1(%rip), %xmm3
                xorl    %ebx, %ebx
                movl    %r12d, %eax
                addq    $1, %rax
                mulsd   %xmm3, %xmm1
                .p2align 4,,10
                .p2align 3
.L5:
                cvtsi2sd        %ebx, %xmm0
                movapd  %xmm1, %xmm7
                mulsd   %xmm3, %xmm0
                movapd  %xmm0, %xmm2
                subsd   %xmm0, %xmm7
                addsd   %xmm1, %xmm2
                movsd   %xmm7, v2(,%rbx,8)
                movsd   %xmm2, v1(,%rbx,8)
                addq    $1, %rbx
                cmpq    %rax, %rbx
                jne     .L5
                movq    %rsp, %rsi
                xorl    %edi, %edi
                salq    $3, %rbx
                call    clock_gettime
                xorl    %eax, %eax
                .p2align 4,,10
                .p2align 3
.L7:
                movsd   v1(%rax), %xmm0
                addq    $8, %rax
                addsd   v2-8(%rax), %xmm0
                movsd   %xmm0, v3-8(%rax)
                cmpq    %rbx, %rax
                jne     .L7
.L6:
                leaq    16(%rsp), %rsi
                xorl    %edi, %edi
                call    clock_gettime
                movq    16(%rsp), %rdx
```

```
        subq    (%rsp), %rdx
        movl    %r12d, %eax
        movsd   v3(,%rax,8), %xmm6
        movl    %r12d, %r9d
        movsd   v2(,%rax,8), %xmm5
        movl    %r12d, %r8d
        movsd   v1(,%rax,8), %xmm4
        movl    %r12d, %ecx
        cvtsi2sdq       %rdx, %xmm0
        movq    24(%rsp), %rdx
        subq    8(%rsp), %rdx
        movsd   v3(%rip), %xmm3
        movsd   v2(%rip), %xmm2
        movl    $.LC3, %esi
        movl    $1, %edi
        movl    $7, %eax
        cvtsi2sdq       %rdx, %xmm1
        movl    %ebp, %edx
        divsd   .LC2(%rip), %xmm1
        addsd   %xmm1, %xmm0
        movsd   v1(%rip), %xmm1
        call    __printf_chk
        addq    $32, %rsp
        .cfi_remember_state
        .cfi_def_cfa_offset 32
        xorl    %eax, %eax
        popq    %rbx
        .cfi_def_cfa_offset 24
        popq    %rbp
        .cfi_def_cfa_offset 16
        popq    %r12
        .cfi_def_cfa_offset 8
        ret
.L3:
        .cfi_restore_state
        movq    %rsp, %rsi
        xorl    %edi, %edi
        orl     $-1, %r12d
        call    clock_gettime
        jmp     .L6
.L11:
        movl    $.LC0, %edi
        call    puts
        orl     $-1, %edi
        call    exit
        .cfi_endproc
.LFE39:
        .size   main, .-main
        .comm   v3,268435456,32
        .comm   v2,268435456,32
```

```
                .comm   v1,268435456,32
                .section        .rodata.cst8,"aM",@progbits,8
                .align 8
.LC1:
                .long   2576980378
                .long   1069128089
                .align 8
.LC2:
                .long   0
                .long   1104006501
                .ident  "GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2"
                .section        .note.GNU-stack,"",@progbits
```

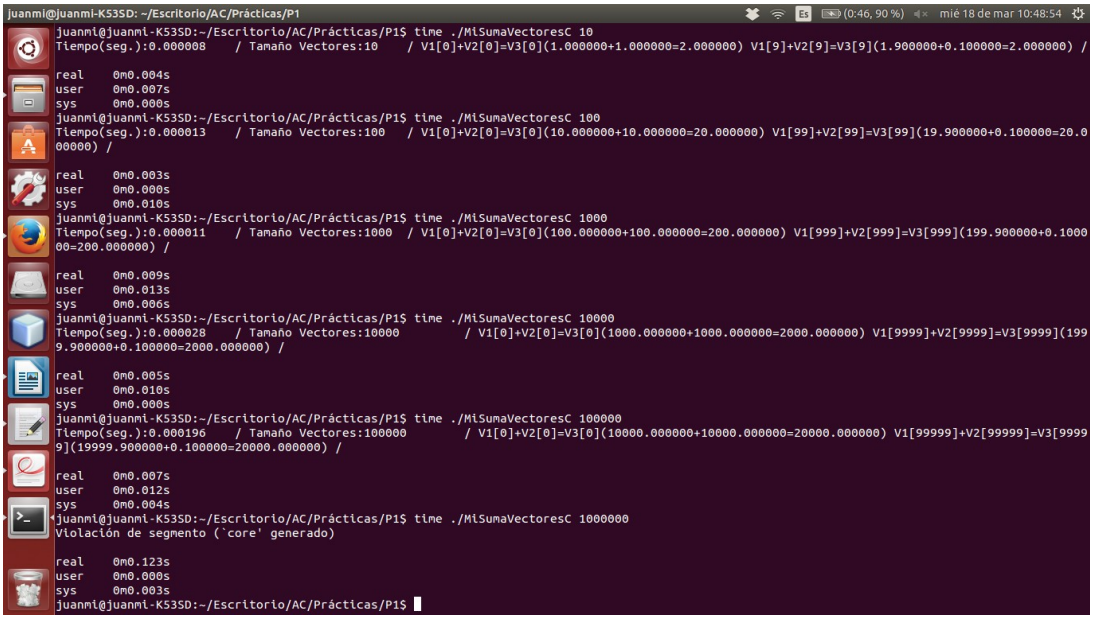
7. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores (v3 = v1 + v2; v3(i)=v1(i)+v2(i), i=0,…N-1) usando las directivas parallel y for. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Como en 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

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


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


  int i;
  double inicio,final,diferencia; //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)
  double v1[N], v2[N], v3[N];   // Tamaño variable local en tiempo de ejecución ...


  //Inicializar vectores
```

```
  #pragma omp parallel for
  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
  }


  inicio = omp_get_wtime();
  //Calcular suma de vectores
  #pragma omp parallel for
  for(i=0; i<N; i++)
    v3[i] = v1[i] + v2[i];


  final = omp_get_wtime();
  diferencia = final - inicio;


  //Imprimir resultado de la suma y el tiempo de ejecución
  printf("Tiempo(seg.):%f\t / Tamaño Vectores:%u\t / V1[0]+V2[0]=V3[0](%f+%f=%f)
V1[%d]+V2[%d]=V3[%d](%f+%f=%f) /\n",diferencia,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-
1],v2[N-1],v3[N-1]);


  return 0;
}
```

CAPTURAS DE PANTALLA:



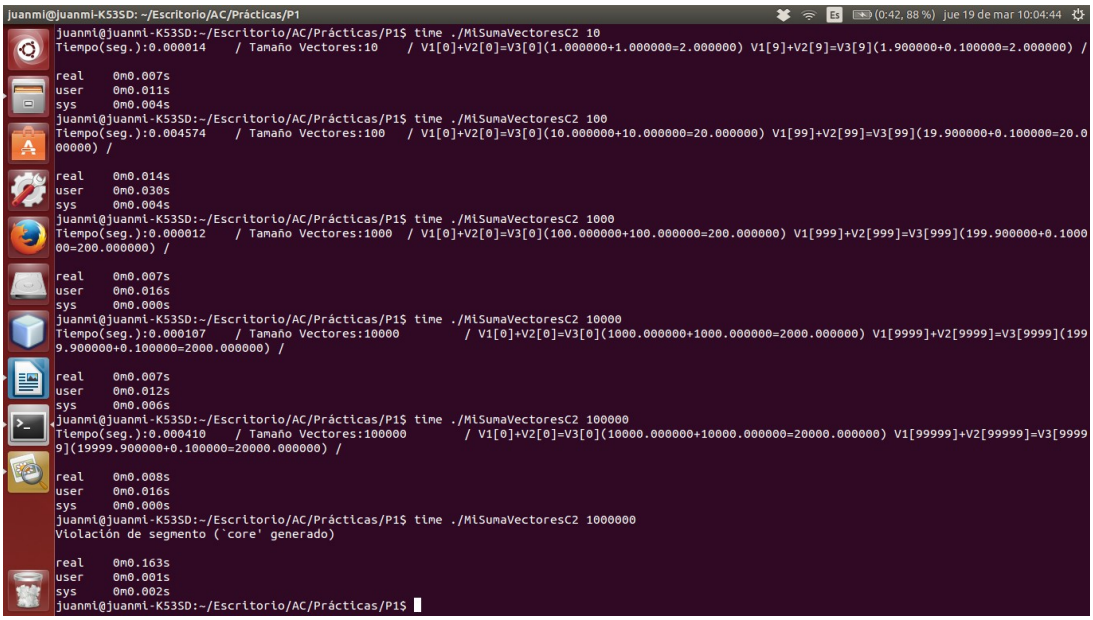
8. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores usando las parallel y sections/section (se debe aprovechar el paralelismo de datos usando estas directivas en lugar de la directiva for); es decir, hay que repartir el trabajo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

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


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


  int i;
  double inicio,final,diferencia; //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)
  double v1[N], v2[N], v3[N];   // Tamaño variable local en tiempo de ejecución ...


  //Inicializar vectores
  #pragma omp parallel sections
  {
    #pragma omp section
                    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
    }
  }


  inicio = omp_get_wtime();


  //Calcular suma de vectores
```

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


  final = omp_get_wtime();
  diferencia = final - inicio;


  //Imprimir resultado de la suma y el tiempo de ejecución
  printf("Tiempo(seg.):%f\t / Tamaño Vectores:%u\t / V1[0]+V2[0]=V3[0](%f+%f=%f)
V1[%d]+V2[%d]=V3[%d](%f+%f=%f) /\n",diferencia,N,v1[0],     v2[0],v3[0],N-1,N-1,N-
1,v1[N-1],v2[N-1],v3[N-1]);


  return 0;
}
```

CAPTURAS DE PANTALLA:



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 podemos utilizar tantos cores como tenga nuestro procesador y utilizaremos tantas hebras como número de iteraciones tenga cada bucle for.

   En el ejercicio 8 utilizaremos el mínimo número posible de hebras y cores, nuestro procesador ejecutará en paralelo el código asociado a la directiva section por lo que se utilizarán tantas hebras como sections haya, en este caso dos.

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.

    RESPUESTA:

Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas en 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	0.000624310	0.000225	0.000204
32768	0.000794100	0.000387	0.000364
65536	0.000657396	0.000504	0.000356
131072	0.000745797	0.000735	0.000766
262144	0.001304893	0.001798	0.001114
524288	0.003547075	-	-
1048576	0.004915191	-	-
2097152	0.008173045	-	-
4194304	0.015099397	-	-
8388608	0.029851060	-	-
16777216	0.058573322	-	-
33554432	0.117744569	-	-
67108864	-	-	-

Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas en ATCGRID

Nº de Componentes	**T. secuencial vect. Globales** **1 thread/core**	**T. paralelo (versión `for`) 4 threads/cores**	**T. paralelo (versión `sections`) 4 threads/cores**
16384	0,000361782	0,000309	0,000323
32768	0,000658471	0,000749	0,000696
65536	0,001428131	0,001445	0,001700
131072	0,002632487	0,002982	0,002834
262144	0,005072489	0,005066	0,005077
524288	0,009992722	-	-
1048576	0,020099146	-	-
2097152	0,039944347	-	-
4194304	0,079171275	-	-
8388608	0,157318946	-	-
16777216	0,314725893	-	-
33554432	0,649796497	-	-
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: El tiempo de CPU tanto para vectores globales y 1 core como para el código asociado al ejercicio 7 y 4 cores es igual al tiempo real.

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` 4 Threads/cores		
	Elapsed	***CPU-user***	***CPU-sys***	***Elapsed***	***CPU-user***	***CPU-sys***
65536	0,004	0,004	0,000	0,005	0,005	0,000
131072	0,007	0,004	0,003	0,006	0,000	0,006
262144	0,007	0,007	0,000	0,007	0,007	0,000
524288	0,009	0,003	0,006	-	-	-
1048576	0,013	0,013	0,000	-	-	-
2097152	0,025	0,021	0,004	-	-	-
4194304	0,049	0,044	0,004	-	-	-
8388608	0,080	0,068	0,012	-	-	-
16777216	0,153	0,132	0,020	-	-	-
33554432	0,295	0,222	0,076	-	-	-
67108864	-	-	-	-	-	-

Nota: Para el código asociado al ejercicio 7 se produce violación de segmento a partir del tamaño 262144 por lo que los valores de tiempo real se disparan.

Nota: Para el código del listado 1 perteneciente a vectores globales el tamaño máximo de los vectores es 33554432 por lo que no se ha podido obtener el tiempo para el tamaño 67108864.