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): Emilio Chica Jiménez
Grupo de prácticas: B2
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();
 }
```

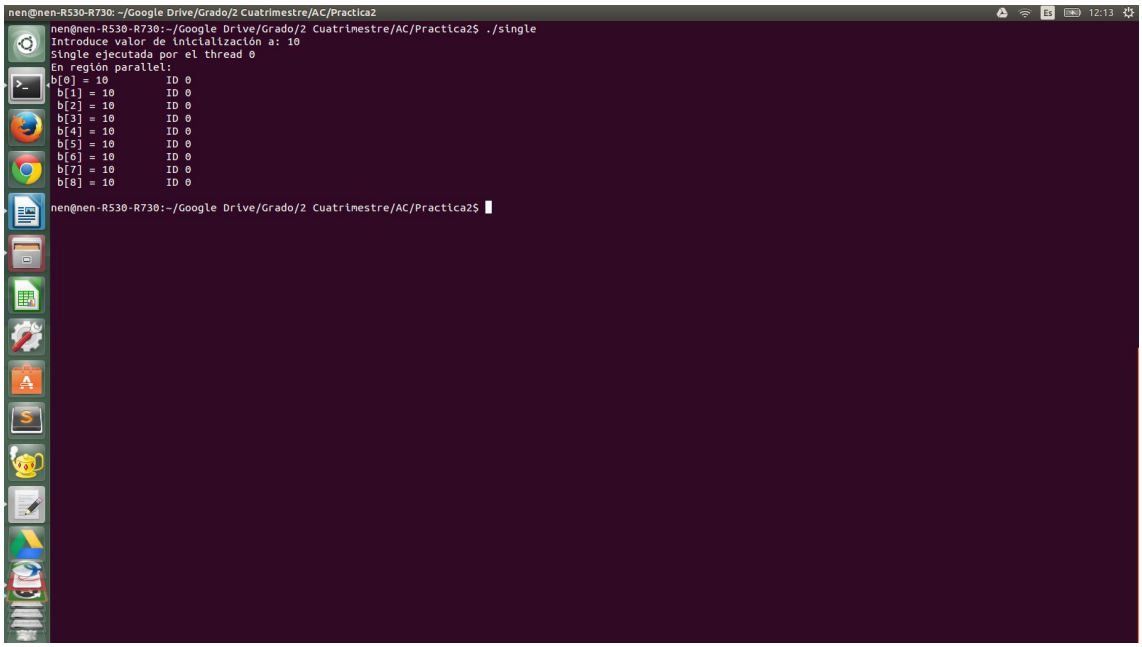
```
}
```

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("En región parallel:\n");
                                       for (i=0; i<n; i++) printf("b[%d] =
%d\t  ID %d\n ",i,b[i],omp_get_thread_num());
                                       printf("\n");
                          }
             }


}
```

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("En región parallel:\n");
                                    for (i=0; i<n; i++) printf("b[%d]
```

```
= %d\t  ID %d\n ",i,b[i],omp_get_thread_num());
                                        printf("\n");
                            }
                }


}
```

CAPTURAS DE PANTALLA:

RESPUESTA A LA PREGUNTA: Difiere de la anterior en que la ejecución de la sección master sólo la realiza la hebra 0.

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

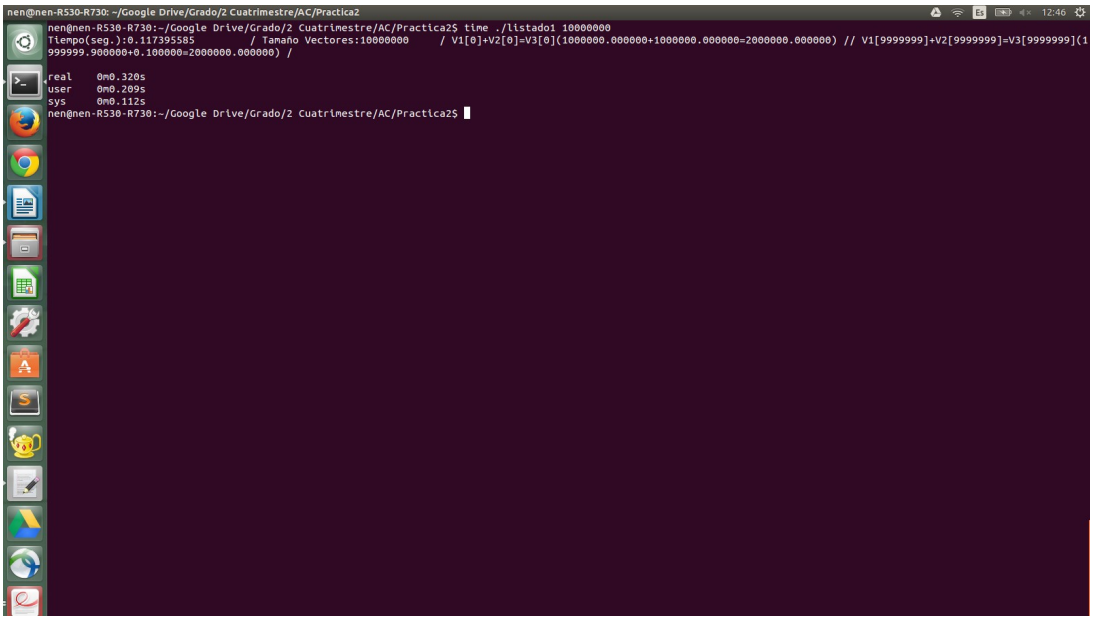
   RESPUESTA: Porque atomic no tiene una barrera implicita por lo tanto pueden terminar con el código de la suma local unas antes que otras y ejecutarse la hebra master antes de terminar la suma total probocará que alguno de los resultados no haya sido sumado por lo tanto será erróneo.

---

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.

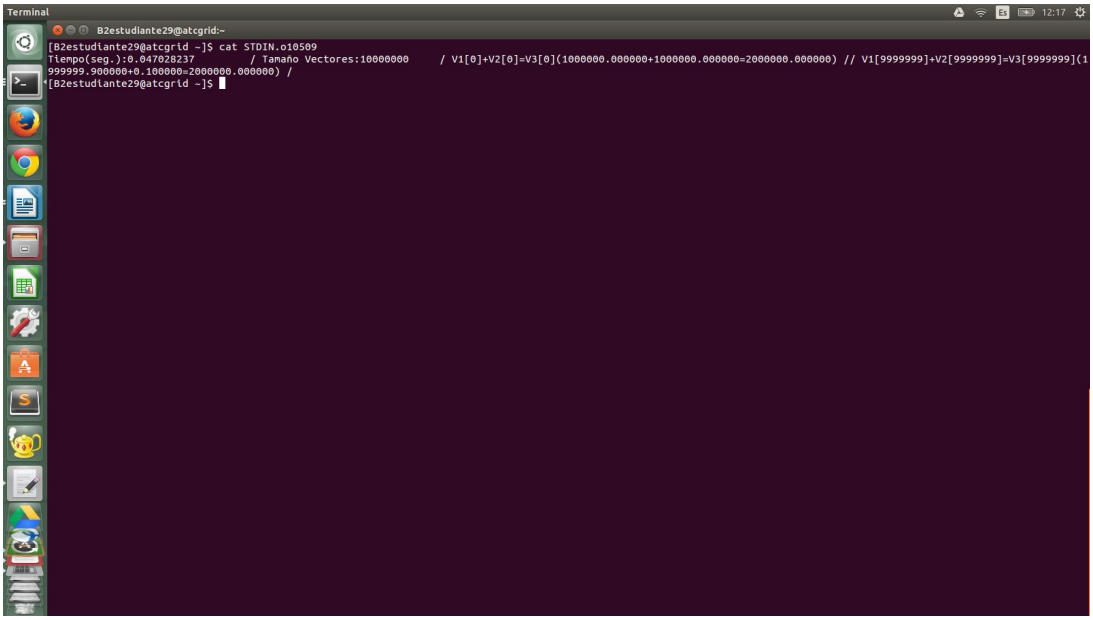
   No es mayor porque la suma total del tiempo de CPU de usuario y de sistema menos el tiempo total que nos muestra es el tiempo que dedica el sistema a las operaciones de E/S. En este caso muestra que es un milisegundo mayor debido al redondeo de los decimales que realiza time.

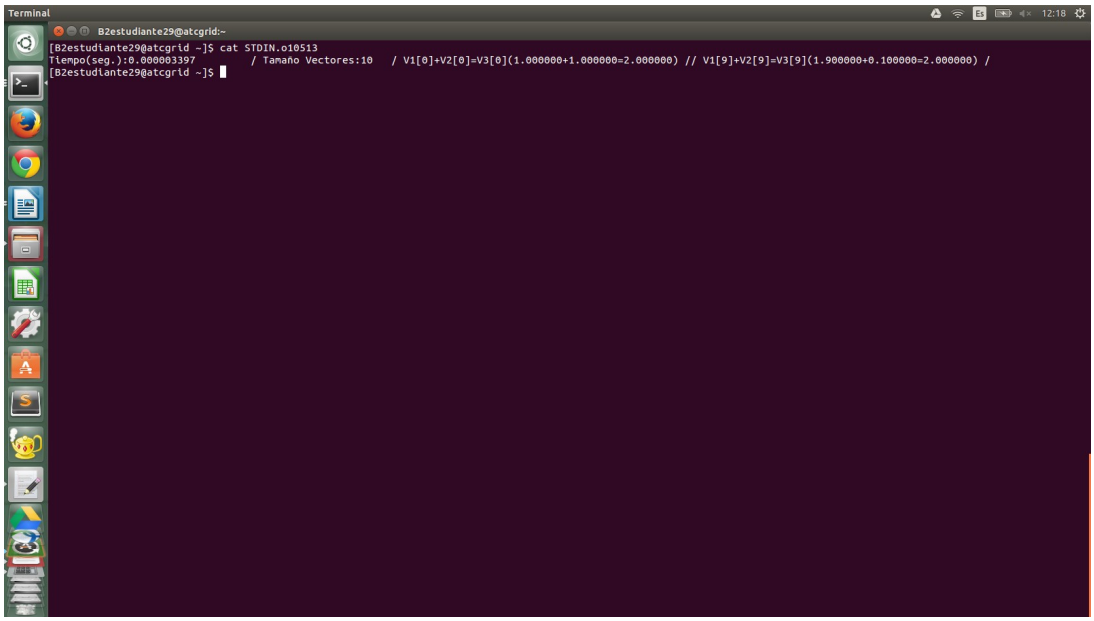
   CAPTURAS DE PANTALLA:



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

RESPUESTA:

MIPS con 10000000 componentes: 1275,829307

MIPS con 10 componentes: 18,54577

MFLOPS con 10000000 componentes: 63,791462

MFLOPS con 10 componentes: 8,831321

código ensamblador generado de la parte de la suma de vectores

```
                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
```

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).

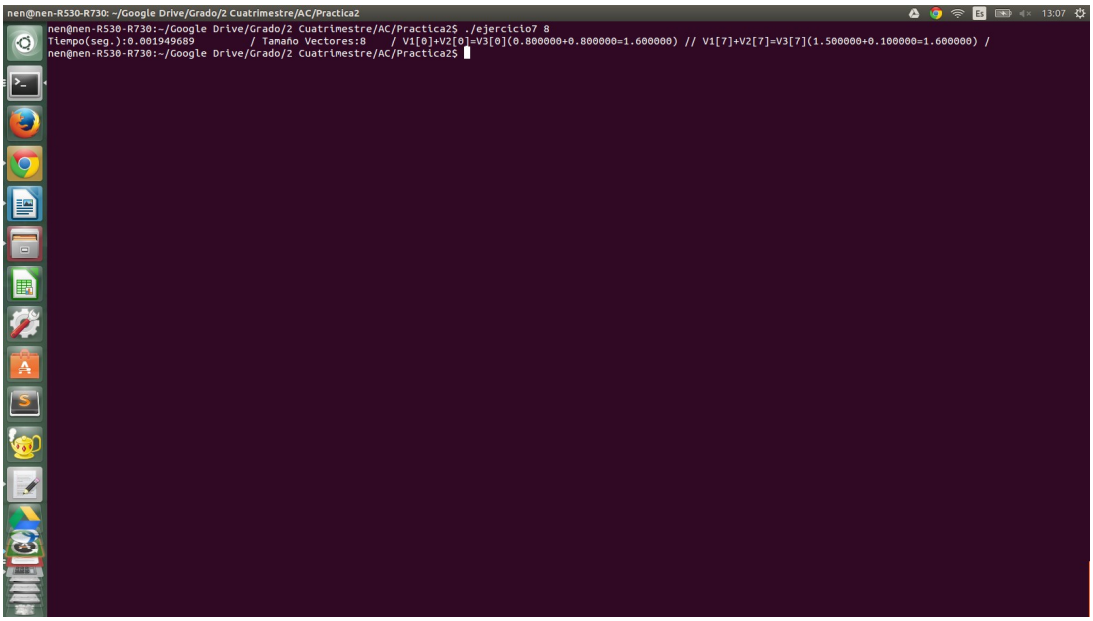
RESPUESTA: código fuente implementado

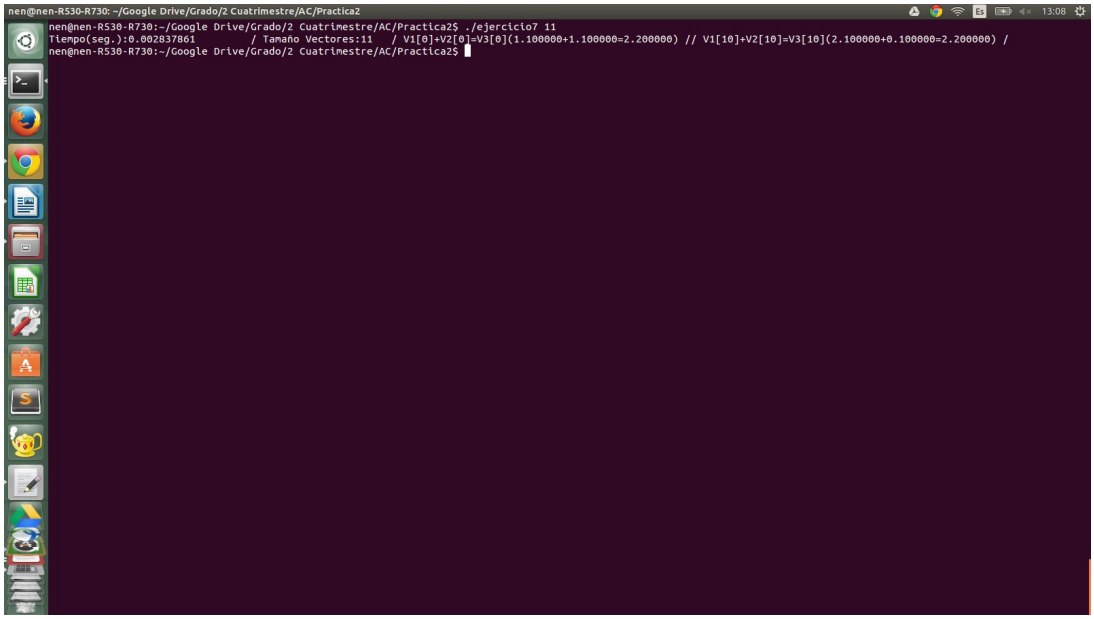
```
Para ejecutar use: SumaVectoresC longitud
*/
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <omp.h>// biblioteca donde se encuentra las funciones paralelas
//#define PRINTF_ALL // comentar para quitar el printf ...
// que imprime todos los componentes
//Sólo puede estar definida una de las tres constantes VECTOR_ (sólo uno de
los ...
//tres defines siguientes puede estar descomentado):
//#define VECTOR_LOCAL // descomentar para que los vectores sean variables ...
// locales (si se supera el tamaño de la pila se ...
// generará el error "Violación de Segmento")
#define VECTOR_GLOBAL// descomentar para que los vectores sean variables ...
// globales (su longitud no estará limitada por el ...
// tamaño de la pila del programa)
//#define VECTOR_DYNAMIC // descomentar para que los vectores sean
variables ...
// dinámicas (memoria reutilizable durante la ejecución)
#ifdef VECTOR_GLOBAL
#define MAX 33554432 //=2^25
double v1[MAX], v2[MAX], v3[MAX];
#endif
int main(int argc, char** argv){
    int i;
    double cgt1,cgt2; double ncgt; //para tiempo de ejecución
    //Leer argumento de entrada (no de componentes del vector)
    if (argc<2){
        printf("Faltan no componentes del vector\n");
        exit(-1);
    }
    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)
    #ifdef VECTOR_LOCAL
        double v1[N], v2[N], v3[N];
        // Tamaño variable local en tiempo de ejecución ...
        // disponible en C a partir de actualización C99
    #endif
    #ifdef VECTOR_GLOBAL
        if (N>MAX) N=MAX;
    #endif
    #ifdef VECTOR_DYNAMIC
        double *v1, *v2, *v3;
        v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño
en bytes
        v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio
```

```
suficiente malloc devuelve NULL
        v3 = (double*) malloc(N*sizeof(double));
        if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
            printf("Error en la reserva de espacio para los vectores\n");
            exit(-2);
        }
    #endif
    //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
                }


    cgt1 = omp_get_wtime();
    //Calcular suma de vectores
    #pragma omp parallel for
                for(i=0; i<N; i++)
                        v3[i] = v1[i] + v2[i];


    cgt2 = omp_get_wtime();
    ncgt=(cgt2-cgt1)/*/(1.e+9)*/;
    //Imprimir resultado de la suma y el tiempo de ejecución
    #ifdef PRINTF_ALL


            printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
            for(i=0; i<N; i++)
                printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f)
/\n",i,i,i,v1[i],v2[i],v3[i]);
    #else
        printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\t/
V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) // V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=
%8.6f) /\n",
            ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
    #endif
    #ifdef VECTOR_DYNAMIC
        free(v1); // libera el espacio reservado para v1
        free(v2); // libera el espacio reservado para v2
        free(v3); // libera el espacio reservado para v3
    #endif
    return 0;
}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA:


nen@nen-R530-R730: ~/Google Drive/Grado/2 Cuatrimestre/AC/Practica2
13:07
nen@nen-R530-R730:~/Google Drive/Grado/2 Cuatrimestre/AC/Practica2$ ./ejercicio7 8
Tiempo(seg.):0.001949689 / Tamaño Vectores:8 / V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) // V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
nen@nen-R530-R730:~/Google Drive/Grado/2 Cuatrimestre/AC/Practica2$


nen@nen-R530-R730: ~/Google Drive/Grado/2 Cuatrimestre/AC/Practica2
13:08
nen@nen-R530-R730:~/Google Drive/Grado/2 Cuatrimestre/AC/Practica2$ ./ejercicio7 11
Tiempo(seg.):0.002837861 / Tamaño Vectores:11 / V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) // V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
nen@nen-R530-R730:~/Google Drive/Grado/2 Cuatrimestre/AC/Practica2$

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

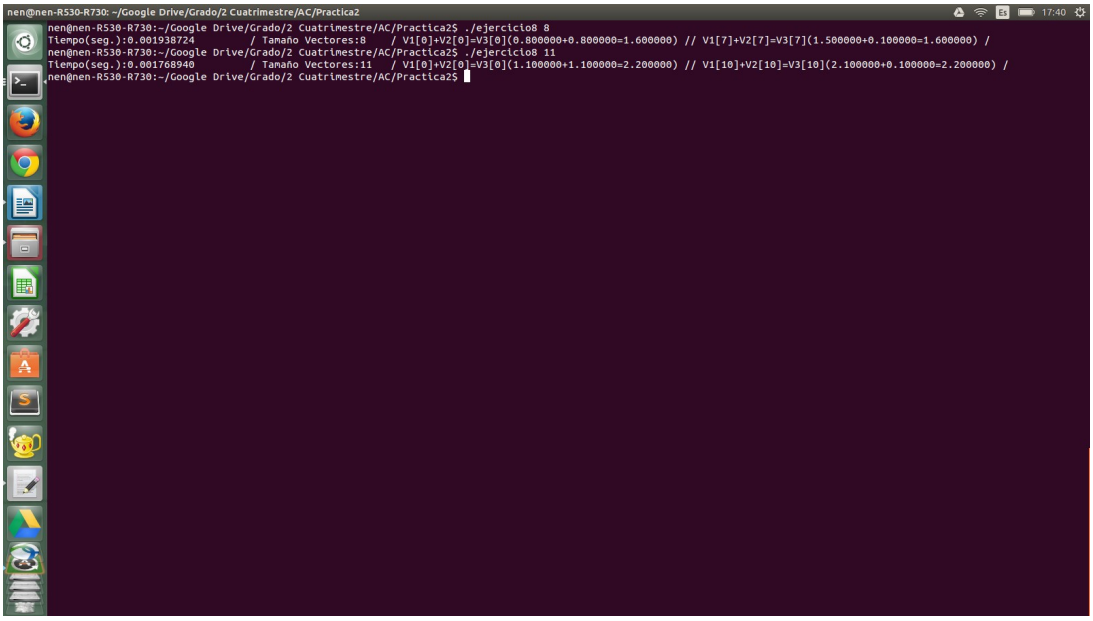
```
/*Para compilar usar (-lrt: real time library):
gcc -O2 SumaVectores.c -o SumaVectores -lrt
Para ejecutar use: SumaVectoresC longitud
*/
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <omp.h>// biblioteca donde se encuentra las funciones paralelas
//#define PRINTF_ALL // comentar para quitar el printf ...
// que imprime todos los componentes
//Sólo puede estar definida una de las tres constantes VECTOR_ (sólo uno de
los ...
//tres defines siguientes puede estar descomentado):
//#define VECTOR_LOCAL // descomentar para que los vectores sean variables ...
// locales (si se supera el tamaño de la pila se ...
// generará el error "Violación de Segmento")
#define VECTOR_GLOBAL// descomentar para que los vectores sean variables ...
// globales (su longitud no estará limitada por el ...
// tamaño de la pila del programa)
//#define VECTOR_DYNAMIC // descomentar para que los vectores sean
variables ...
// dinámicas (memoria reutilizable durante la ejecución)
#ifdef VECTOR_GLOBAL
#define MAX 33554432 //=2^25
double v1[MAX], v2[MAX], v3[MAX];
#endif
int main(int argc, char** argv){
    int i;
    double cgt1,cgt2; double ncgt; //para tiempo de ejecución
    //Leer argumento de entrada (no de componentes del vector)
    if (argc<2){
        printf("Faltan no componentes del vector\n");
        exit(-1);
    }
    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)
    #ifdef VECTOR_LOCAL
        double v1[N], v2[N], v3[N];
        // Tamaño variable local en tiempo de ejecución ...
        // disponible en C a partir de actualización C99
    #endif
    #ifdef VECTOR_GLOBAL
        if (N>MAX) N=MAX;
    #endif
```

```
    #ifdef VECTOR_DYNAMIC
        double *v1, *v2, *v3;
        v1 = (double*) malloc(N*sizeof(double));// malloc necesita el tamaño
en bytes
        v2 = (double*) malloc(N*sizeof(double)); //si no hay espacio
suficiente malloc devuelve NULL
        v3 = (double*) malloc(N*sizeof(double));
        if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
            printf("Error en la reserva de espacio para los vectores\n");
            exit(-2);
        }
    #endif
    //Inicializar vectores
    #pragma omp parallel
    {
              #pragma omp sections
              {
                          #pragma omp section
                              for(i=0; 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
                                        }
                          #pragma omp section
                              for(i=(N/2); i<N; i++){
                                        v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-
i*0.1; //los valores dependen de N
                    }
              }
     }


    cgt1 = omp_get_wtime();
    //Calcular suma de vectores
    #pragma omp parallel
    {
              #pragma omp sections
              {
                          #pragma omp section
                              for(i=0; i<N/2; i++)
                                        v3[i] = v1[i] + v2[i];
                          #pragma omp section
                              for(i=(N/2); i<N; i++)
                                        v3[i] = v1[i] + v2[i];
              }
   }
    cgt2 = omp_get_wtime();
    ncgt=(cgt2-cgt1)/*/(1.e+9)*/;
    //Imprimir resultado de la suma y el tiempo de ejecución
    #ifdef PRINTF_ALL


              printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
              for(i=0; i<N; i++)
                  printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f)
/\n",i,i,i,v1[i],v2[i],v3[i]);
    #else
        printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\t/
V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) // V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=
%8.6f) /\n",
              ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
    #endif
    #ifdef VECTOR_DYNAMIC
```

```
        free(v1); // libera el espacio reservado para v1
        free(v2); // libera el espacio reservado para v2
        free(v3); // libera el espacio reservado para v3
    #endif
    return 0;
}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA:



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: Debido a que el ordenador donde lo he ejecutado tiene dos cores, como máximo podría estar usando estos dos, y en cuanto a hebras el SO podría lanzar su máximo pues no especificamos cuantas hebras queremos que ejecuten nuestro código y por ello el SO las crea dinámicamente, pero útiles depende de los datos con los que vayamos a trabajar con el programa.

   En cuanto a cores la respuesta anterior es válida para el ejercicio 8 también, ya que sigo teniendo los mismos cores. Pero con respecto a threads en este caso sólo tendremos 2 ya que he creado 2 sections los cuales se deducen en la creación de 2 hebras. Por lo que se elimina el tiempo de creación o destrucción de más hebras innecesarias.

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.

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.

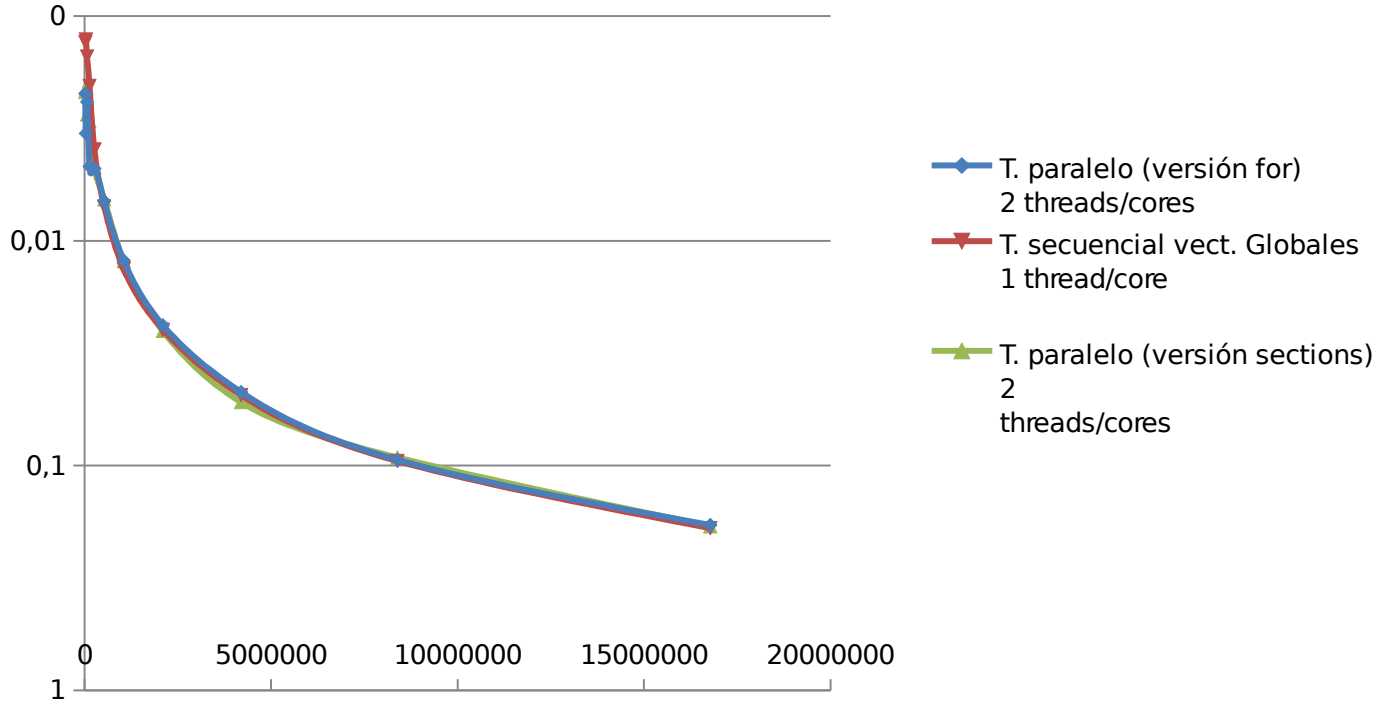
LOCAL

Nº de Componentes	**T. secuencial vect. Globales** **1 thread/core**	**T. paralelo (versión `for`) 2 threads/cores**	**T. paralelo (versión `sections`) 2 threads/cores**
16384	0.001308686	0.002210896	0.002181422
32768	0.001271041	0.003332617	0.002128763
65536	0.001515766	0.002408825	0.002740781
131072	0.002047397	0.004666096	0.002934521
262144	0.003924731	0.004769252	0.004721900
524288	0.007042795	0.006659367	0.006548040
1048576	0.013355749	0.012245971	0.012337183
2097152	0.024841916	0.023733814	0.025165910
4194304	0.048598359	0.047096352	0.052192461
8388608	0.095482044	0.094324355	0.092044107
16777216	0.189935606	0.183680201	0.186246310
33554432	0.380225097	0.371568061	0.363300180
67108864	0.384790133	0.362170287	0.371409869

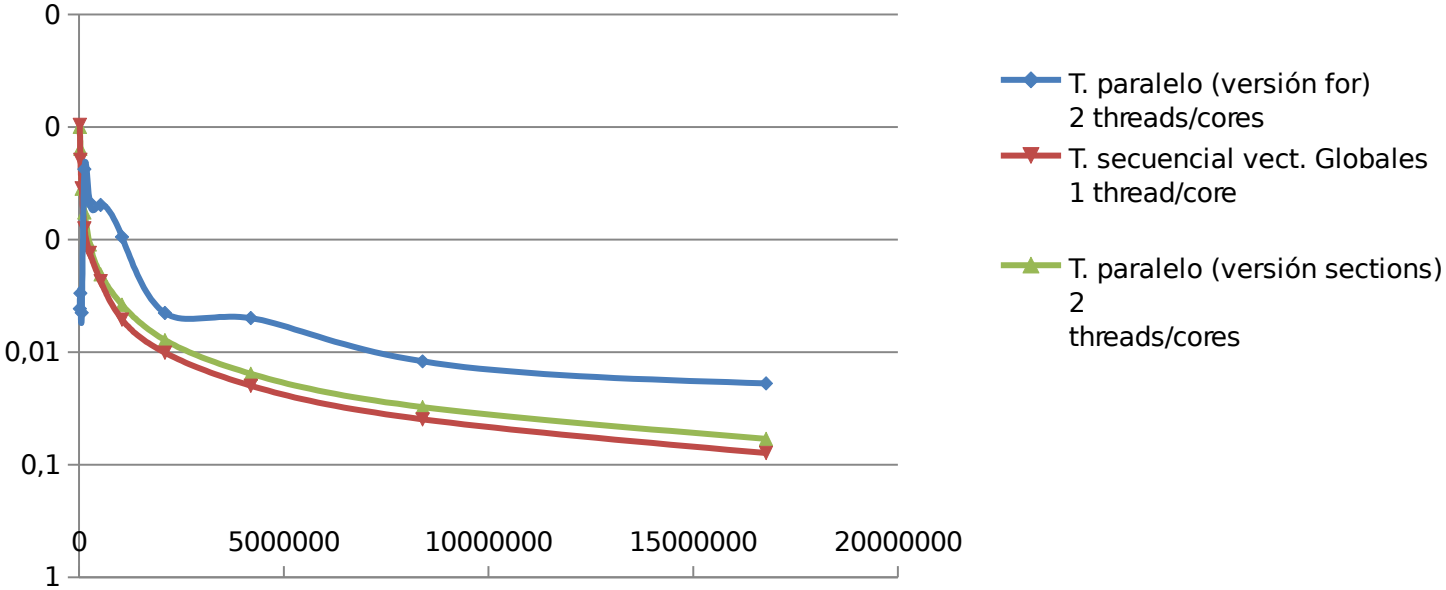
ATCGRID

Nº de Componentes	**T. secuencial vect. Globales** **1 thread/core**	**T. paralelo (versión `for`) 1 a 24 threads/cores**	**T. paralelo (versión `sections`) 2 threads/cores**
16384	0.000095971	0.004101126	0.000099557
32768	0.000195341	0.003000158	0.000155517
65536	0.000351640	0.004456974	0.000353351
131072	0.000794076	0.000236724	0.000572239
262144	0.001312471	0.000467597	0.001101266
524288	0.002335675	0.000491038	0.002029370
1048576	0.005126983	0.000946385	0.003787681
2097152	0.010115864	0.004478408	0.007806665
4194304	0.019833539	0.004980425	0.015534148
8388608	0.039466471	0.012065224	0.030728744
16777216	0.078703030	0.018911896	0.058710770
33554432	0.157152167	0.036884294	0.110205064
67108864	0.158139610	0.036826201	0.122721335

PC LOCAL



ATCGRID



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, ya que el tiempo real es el tiempo de CPU+Tiempo de entrada y salida por lo tanto debería ser siempre menor o igual.

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.

LOCAL

Nº de Componentes	**Tiempo secuencial vect. Globales 1 thread/core**		**Tiempo paralelo/versión `for` 2 Threads/cores**	
	Elapsed	***CPU-user CPU- sys***	***Elapsed***	***CPU-user CPU-sys***
65536	real 0m0.004s user 0m0.000s sys 0m0.003s		real 0m0.005s user 0m0.000s sys 0m0.009s	
131072	real 0m0.004s user 0m0.000s sys 0m0.003s		real 0m0.009s user 0m0.000s sys 0m0.010s	
262144	real 0m0.004s user 0m0.000s sys 0m0.004s		real 0m0.009s user 0m0.002s sys 0m0.010s	
524288	real 0m0.010s user 0m0.003s sys 0m0.003s		real 0m0.017s user 0m0.009s sys 0m0.005s	
1048576	real 0m0.014s user 0m0.000s sys 0m0.012s		real 0m0.013s user 0m0.009s sys 0m0.014s	
2097152	real 0m0.023s user 0m0.004s sys 0m0.015s		real 0m0.028s user 0m0.011s sys 0m0.023s	
4194304	real 0m0.039s user 0m0.017s sys 0m0.023s		real 0m0.041s user 0m0.032s sys 0m0.024s	
8388608	real 0m0.066s user 0m0.031s sys 0m0.031s		real 0m0.063s user 0m0.064s sys 0m0.036s	
16777216	real 0m0.131s user 0m0.070s sys 0m0.059s		real 0m0.119s user 0m0.115s sys 0m0.087s	
33554432	real 0m0.238s user 0m0.187s sys 0m0.049s		real 0m0.228s user 0m0.254s sys 0m0.164s	
67108864	real 0m0.461s user 0m0.348s sys 0m0.112s		real 0m0.461s user 0m0.480s sys 0m0.329s	