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): Francisco Carrillo Pérez
Grupo de prácticas: A2
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: 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 = 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`

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

```
/* INTERLINEADO SENCILLO */

#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();


             }
}
```

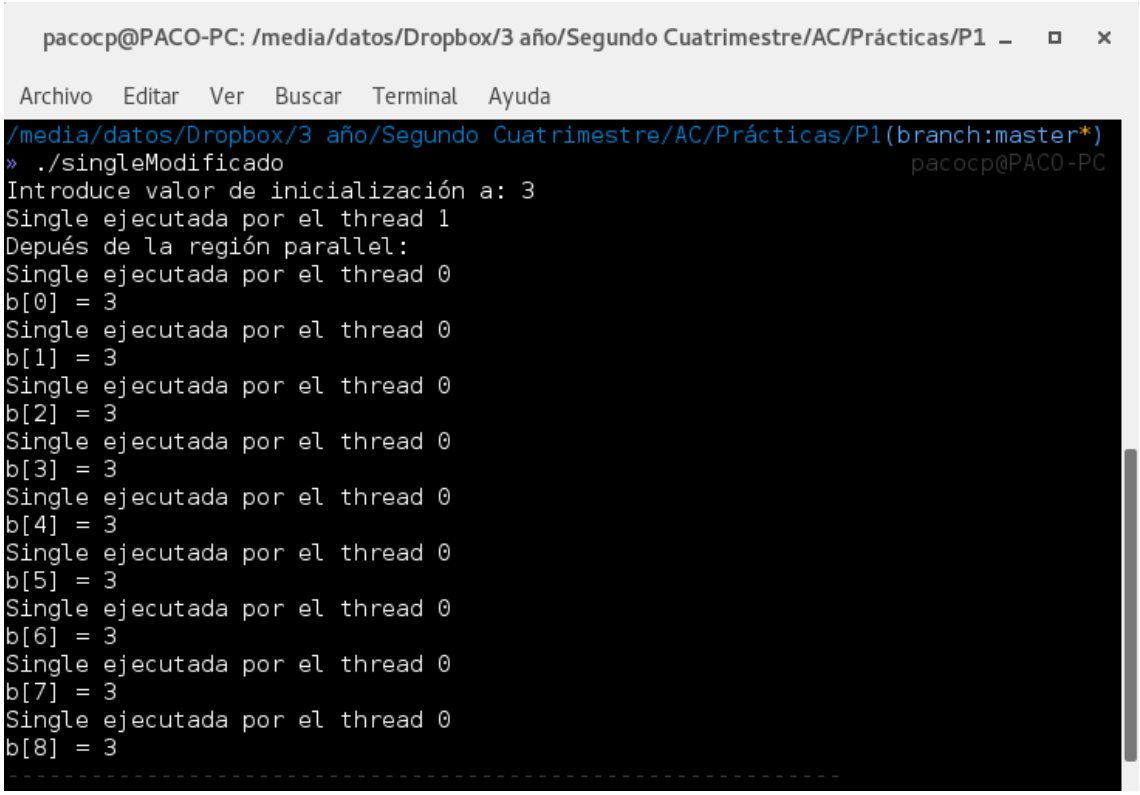
. Imprimir los resultados del programa `single.c` usando una directiva `single` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `single` incorporada, para que se imprima el identificador del thread que ejecuta 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 `single`Modificado.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 <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("Depués de la región parallel:\n");
                        for (i=0; i<n; i++)
                                    {
                                                printf("Single ejecutada
por el thread %d\n",
                                                omp_get_thread_num());
                                                printf("b[%d] =
%d\t",i,b[i]);
                                                printf("\n");
                                    }
            }


 }


}
```

CAPTURAS DE PANTALLA:

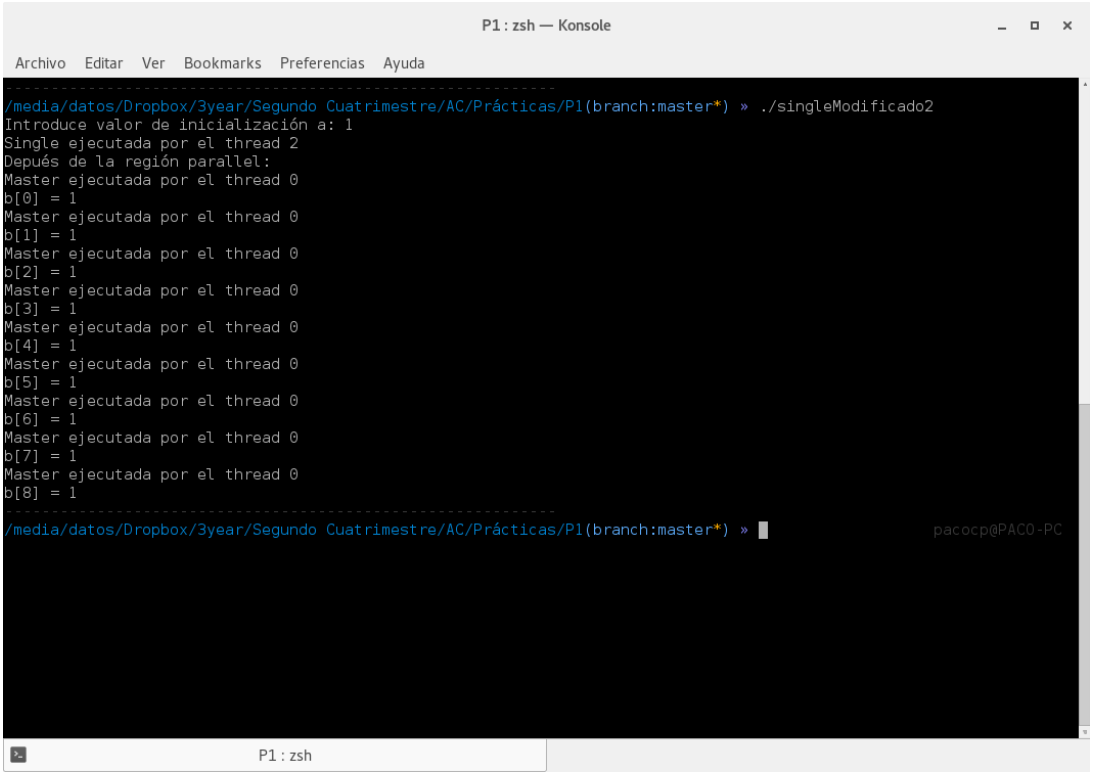


. Imprimir los resultados del programa `single.c` usando una directiva `master` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `master` incorporada, para que se imprima el identificador del thread que ejecuta 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 <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("Depués de la región parallel:\n");
                        for (i=0; i<n; i++)
                                    {
                                                printf("Master
ejecutada por el thread %d\n",
                                                omp_get_thread_num());
                                                printf("b[%d] =
%d\t",i,b[i]);
                                                printf("\n");
                                    }


            }
 }


}
```

CAPTURAS DE PANTALLA:



RESPUESTA A LA PREGUNTA:

La diferencia es que en este caso la directiva master hace que siempre la hebra que imprima los resultados sea la hebra master, en este caso la hebra 0.

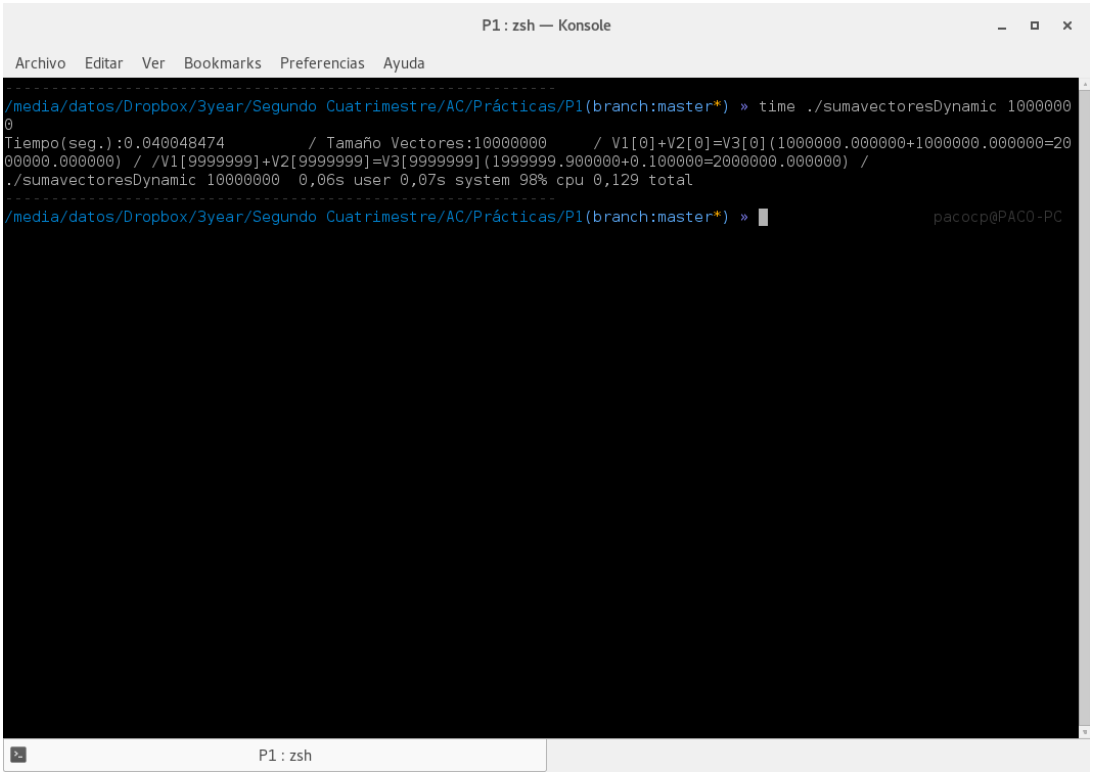
. ¿Por qué si se elimina directiva `barrier` en el ejemplo `master.c` la suma que se calcula e imprime no siempre es correcta? Responda razonadamente.

RESPUESTA: Porque master no tiene barreras implícitas, por lo que si no esperamos a que todas as hebras hayan añadido su suma local a la suma global, el resultado que imprima la hebra master puede no ser correcto.

Resto de ejercicios

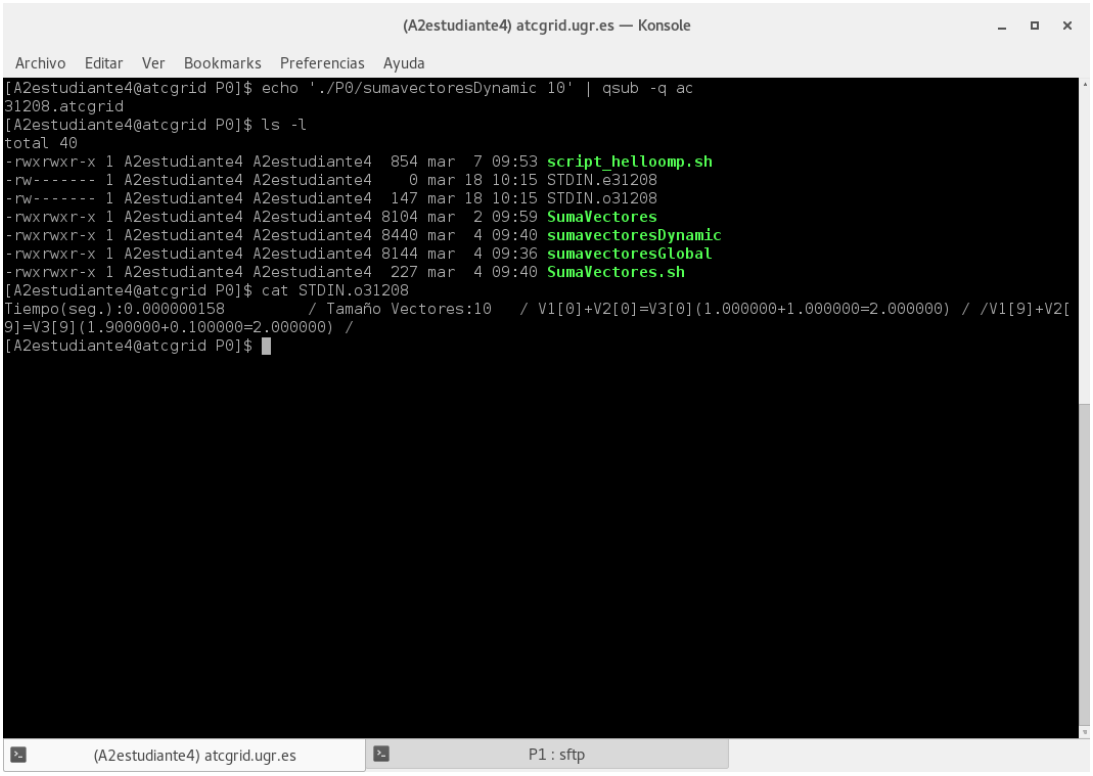
. El programa secuencial C del Listado 1 calcula la suma de dos vectores (v3 = v1 + v2; v3(i) = v1(i) + v2(i), i=0,…N-1). Generar el ejecutable del programa del Listado 1 para **vectores dinámicos**. Usar `time` (Lección 3/ Tema 1) en la línea de comandos para obtener, en el PC local, el tiempo de ejecución (*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.

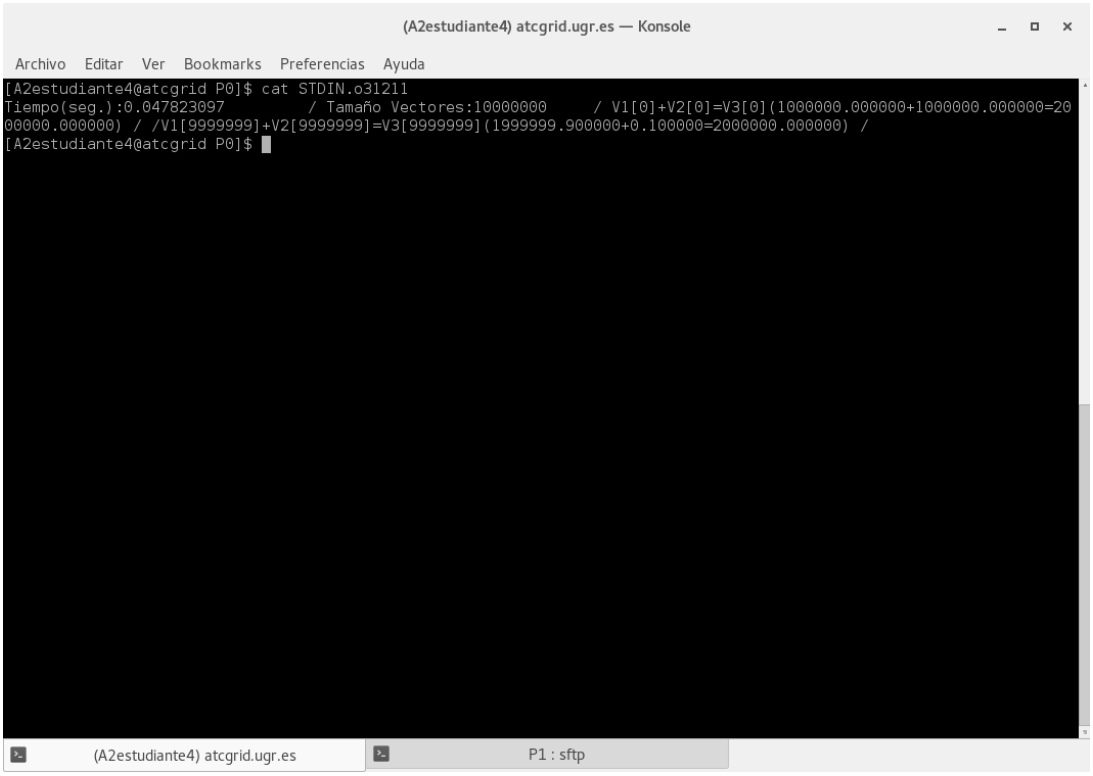
CAPTURAS DE PANTALLA:



. 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

MIPS10 = 63 / (0.000000158 * 10^6) = 398.73 MIPS

MFLOPS10 = 30 / (0.000000158 * 10^6) = 189.87 MFLOPS

MIPS 10000000 = (3 + 6 *10000000) / (0.047823097 * 10^6) = 1254.65 MIPS

MFLOPS 10000000 = (3 *10000000) / (0.047823097 * 10^6) = 627.31 MFLOPS

RESPUESTA:

código ensamblador generado de la parte de la suma de vectores

```
            .file   "sumavectores.c"
            .section    .rodata.str1.8,"aMS",@progbits,1
            .align 8
.LC0:
            .string "Faltan no componentes del vector"
            .align 8
.LC1:
            .string "Error en la reserva de espacio para los vectores"
            .align 8
.LC4:
            .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.unlikely,"ax",@progbits
.LCOLDB5:
            .section    .text.startup,"ax",@progbits
.LHOTB5:
            .p2align 4,,15
            .globl      main
            .type       main, @function
main:
.LFB21:
            .cfi_startproc
            pushq       %r15
            .cfi_def_cfa_offset 16
            .cfi_offset 15, -16
            pushq       %r14
            .cfi_def_cfa_offset 24
            .cfi_offset 14, -24
            pushq       %r13
            .cfi_def_cfa_offset 32
            .cfi_offset 13, -32
            pushq       %r12
            .cfi_def_cfa_offset 40
            .cfi_offset 12, -40
            pushq       %rbp
            .cfi_def_cfa_offset 48
            .cfi_offset 6, -48
            pushq       %rbx
            .cfi_def_cfa_offset 56
            .cfi_offset 3, -56
            subq        $40, %rsp
            .cfi_def_cfa_offset 96
            cmpl        $1, %edi
            jle         .L18
            movq        8(%rsi), %rdi
            movl        $10, %edx
            xorl        %esi, %esi
            call        strtol
            movl        %eax, %ebx
            movq        %rax, %r13
            movl        %eax, %r14d
            leaq        0(,%rbx,8), %r15
            movq        %r15, %rdi
            call        malloc
            movq        %r15, %rdi
            movq        %rax, %rbp
            call        malloc
            movq        %r15, %rdi
```

```
                movq          %rax, %r12
                call          malloc
                testq         %rbp, %rbp
                movq          %rax, %r15
                sete          %dl
                testq         %r12, %r12
                sete          %al
                orb           %al, %dl
                jne           .L3
                testq         %r15, %r15
                je            .L3
                testl         %r13d, %r13d
                je            .L19
                pxor          %xmm1, %xmm1
                xorl          %eax, %eax
                movsd         .LC2(%rip), %xmm3
                cvtsi2sdq     %rbx, %xmm1
                mulsd         %xmm3, %xmm1
                .p2align 4,,10
                .p2align 3
.L8:
                pxor          %xmm0, %xmm0
                movapd        %xmm1, %xmm7
                cvtsi2sd      %eax, %xmm0
                mulsd         %xmm3, %xmm0
                movapd        %xmm0, %xmm2
                subsd         %xmm0, %xmm7
                addsd         %xmm1, %xmm2
                movsd         %xmm7, (%r12,%rax,8)
                movsd         %xmm2, 0(%rbp,%rax,8)
                addq          $1, %rax
                cmpl          %eax, %r14d
                ja            .L8
                movq          %rsp, %rsi
                xorl          %edi, %edi
                call          clock_gettime
                xorl          %eax, %eax
                .p2align 4,,10
                .p2align 3
.L9:
                movsd         0(%rbp,%rax,8), %xmm0
                addsd         (%r12,%rax,8), %xmm0
                movsd         %xmm0, (%r15,%rax,8)
                addq          $1, %rax
                cmpl          %eax, %r14d
                ja            .L9
.L10:
                leaq          16(%rsp), %rsi
                xorl          %edi, %edi
                call          clock_gettime
                movq          24(%rsp), %rax
                subq          8(%rsp), %rax
                leal          -1(%r13), %ecx
                pxor          %xmm0, %xmm0
                movl          %r13d, %esi
                pxor          %xmm1, %xmm1
                movq          %rcx, %rdx
                movsd         (%r15,%rcx,8), %xmm6
                movl          %ecx, %r8d
                cvtsi2sdq     %rax, %xmm0
                movq          16(%rsp), %rax
                subq          (%rsp), %rax
```

```
                movsd         (%r12,%rcx,8), %xmm5
                movsd         0(%rbp,%rcx,8), %xmm4
                movl          $.LC4, %edi
                movsd         (%r15), %xmm3
                movsd         (%r12), %xmm2
                cvtsi2sdq     %rax, %xmm1
                movl          $7, %eax
                divsd         .LC3(%rip), %xmm0
                addsd         %xmm1, %xmm0
                movsd         0(%rbp), %xmm1
                call          printf
                movq          %rbp, %rdi
                call          free
                movq          %r12, %rdi
                call          free
                movq          %r15, %rdi
                call          free
                addq          $40, %rsp
                .cfi_remember_state
                .cfi_def_cfa_offset 56
                xorl          %eax, %eax
                popq          %rbx
                .cfi_def_cfa_offset 48
                popq          %rbp
                .cfi_def_cfa_offset 40
                popq          %r12
                .cfi_def_cfa_offset 32
                popq          %r13
                .cfi_def_cfa_offset 24
                popq          %r14
                .cfi_def_cfa_offset 16
                popq          %r15
                .cfi_def_cfa_offset 8
                ret
.L19:
                .cfi_restore_state
                movq          %rsp, %rsi
                xorl          %edi, %edi
                call          clock_gettime
                jmp           .L10
.L3:
                movl          $.LC1, %edi
                call          puts
                movl          $-2, %edi
                call          exit
.L18:
                movl          $.LC0, %edi
                call          puts
                orl           $-1, %edi
                call          exit
                .cfi_endproc
.LFE21:
                .size         main, .-main
                .section      .text.unlikely
.LCOLDE5:
                .section      .text.startup
.LHOTE5:
                .section      .rodata.cst8,"aM",@progbits,8
                .align 8
.LC2:
                .long         2576980378
                .long         1069128089
```

```
            .align 8
.LC3:
            .long       0
            .long       1104006501
            .ident      "GCC: (GNU) 5.3.0"
            .section    .note.GNU-stack,"",@progbits
```

. 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

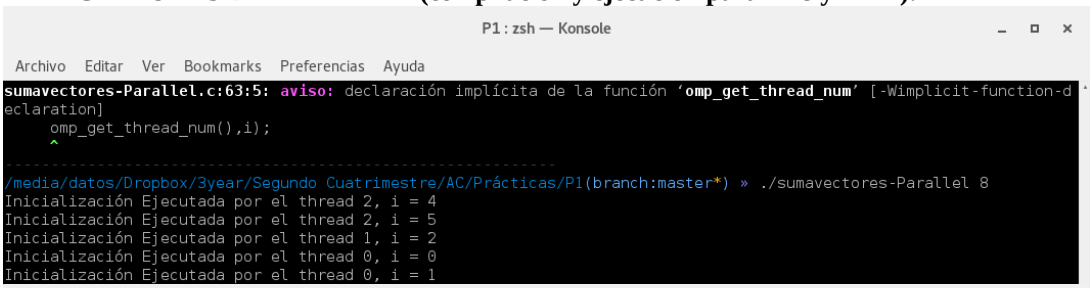
```
/* SumaVectoresC.c
Suma de dos vectores: v3 = v1 + v2
Para compilar usar (-lrt: real time library):
gcc -O2 SumaVectores.c -o SumaVectores -lrt
Para ejecutar use: SumaVectoresC longitud
*/
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
#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  4294967295//=2^32-1
double v1[MAX], v2[MAX], v3[MAX];
#endif




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


int i;
struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución
//Leer argumento de entrada (no de componentes del vector)
```

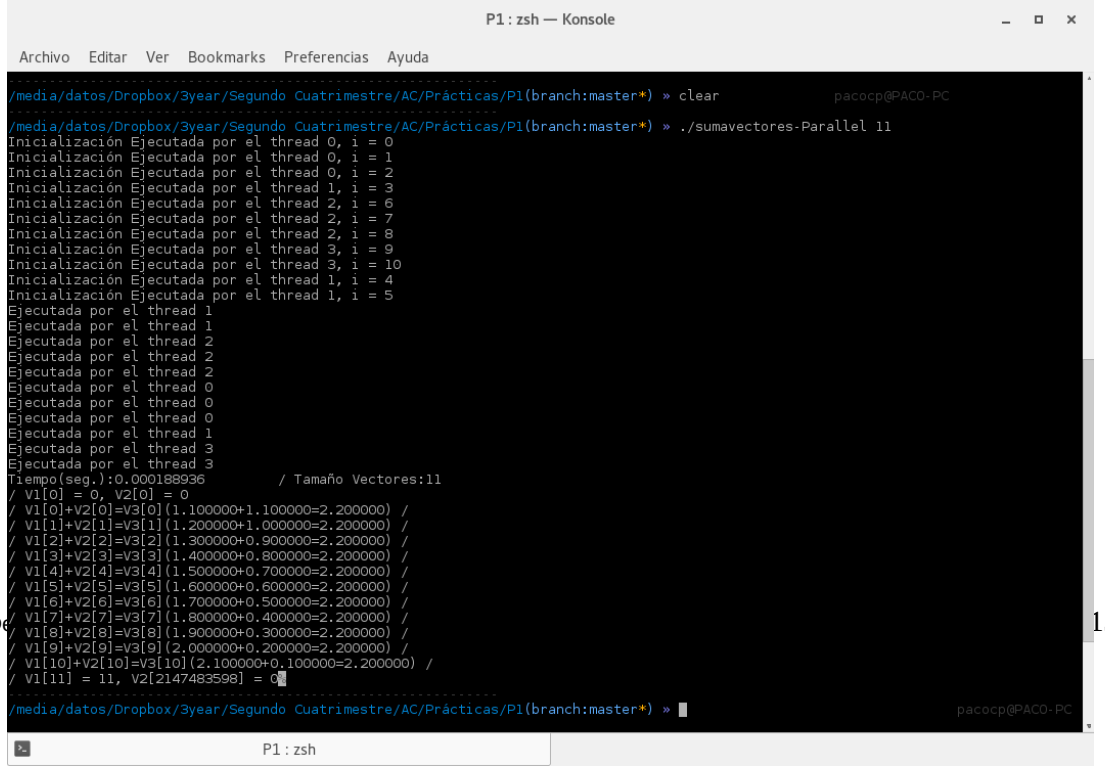
```
if (argc<2){
printf("Faltan no componentes del vector\n");
exit(-1);
}
unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)
#ifdef VECTOR_LOCAL
double v1[N], v2[N], v3[N];
 // Tamaño variable local en tiempo de ejecución ...
// disponible en C a partir de actualización 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
//Inicializando los vectores
#pragma omp parallel
{


#pragma omp for
for(i=0; i<N; i++){
             printf("Inicialización Ejecutada por el thread %d, i = %d\n",
                                                          omp_get_thread_num(),i);
v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los valores dependen de N
}


#pragma omp single
{
clock_gettime(CLOCK_REALTIME,&cgt1);
}
//Calcular suma de vectores
#pragma omp  for
             for(i=0; i<N; i++)
                          {
                                       printf("Ejecutada por el thread %d\n",


             omp_get_thread_num());
                                       v3[i] = v1[i] + v2[i];
                          }




#pragma omp sigle
                          {
                                       clock_gettime(CLOCK_REALTIME,&cgt2);
                          }
```

```
}
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(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);
printf("/ V1[0] = %d, V2[0] = %d\n", v1[0],v2[0]);
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]);
}
printf("/ V1[%d] = %d, V2[%d] = %d", N,v1[N],N,v2[N]);


#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 (compilación y ejecución para N=8 y N=11):





. 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

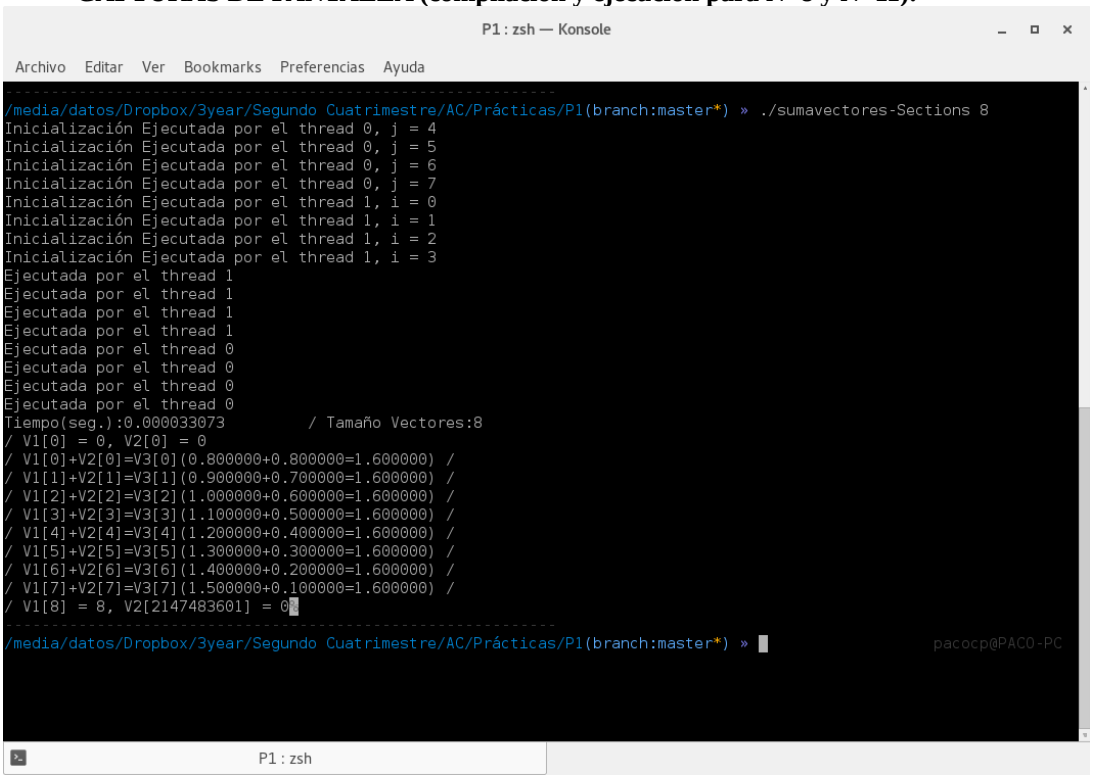
```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

/* SumaVectoresC.c
Suma de dos vectores: v3 = v1 + v2
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 <time.h> // biblioteca donde se encuentra la función clock_gettime()
#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  4294967295//=2^32-1
double v1[MAX], v2[MAX], v3[MAX];
#endif




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


int i,j;
struct timespec 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);
}
```

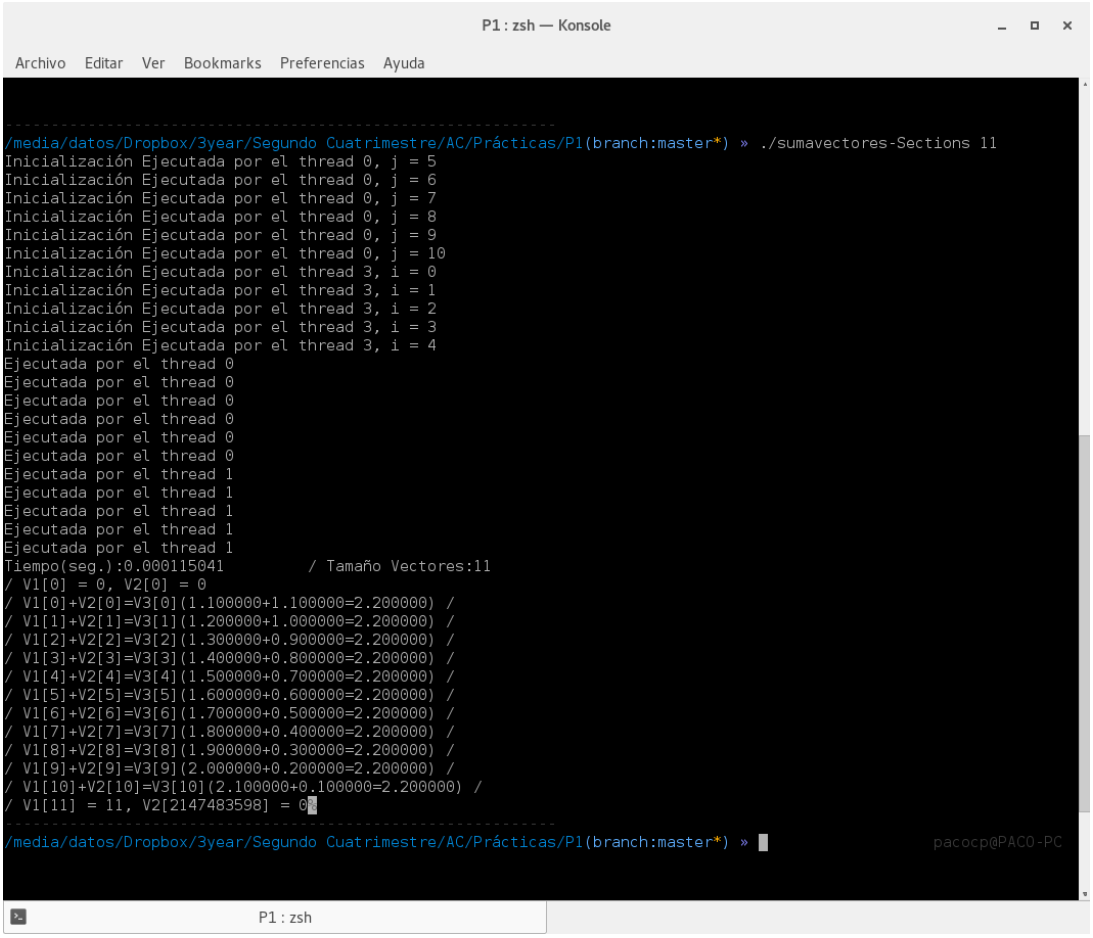
```
unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)
#ifdef VECTOR_LOCAL
double v1[N], v2[N], v3[N];
 // Tamaño variable local en tiempo de ejecución ...
// disponible en C a partir de actualización C99
#endif
#ifdef VECTOR_GLOBAL
if (N>MAX) N=MAX;
#endif
#ifdef 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
//Inicializando los vectores
#pragma omp parallel
{


#pragma omp sections
{
             #pragma omp section
             for(i=0; i<N/2; i++){
                          printf("Inicialización Ejecutada por el thread %d,
i = %d\n",


             omp_get_thread_num(),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(j=N/2; j<N; j++){
                          printf("Inicialización Ejecutada por el thread %d,
j = %d\n",


             omp_get_thread_num(),j);
             v1[j] = N*0.1+j*0.1; v2[j] = N*0.1-j*0.1; //los valores dependen
de N
             }
}
i = 0;
j = 0;
#pragma omp single
{
clock_gettime(CLOCK_REALTIME,&cgt1);
}
//Calcular suma de vectores
#pragma omp  sections
{
             #pragma omp section
             for(i=0; i<N/2; i++)
                          {
                                       printf("Ejecutada por el thread %d\n",
```

```
            omp_get_thread_num());
                                    v3[i] = v1[i] + v2[i];
                        }
            #pragma omp section
            for(j=N/2; j<N; j++)
                        {
                                    printf("Ejecutada por el thread %d\n",


            omp_get_thread_num());
                                    v3[j] = v1[j] + v2[j];
                        }
}




#pragma omp sigle
                        {
                                    clock_gettime(CLOCK_REALTIME,&cgt2);
                        }


}
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(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);
printf("/ V1[0] = %d, V2[0] = %d\n", v1[0],v2[0]);
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]);
}
printf("/ V1[%d] = %d, V2[%d] = %d", N,v1[N],N,v2[N]);


#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 (compilación y ejecución para N=8 y N=11):



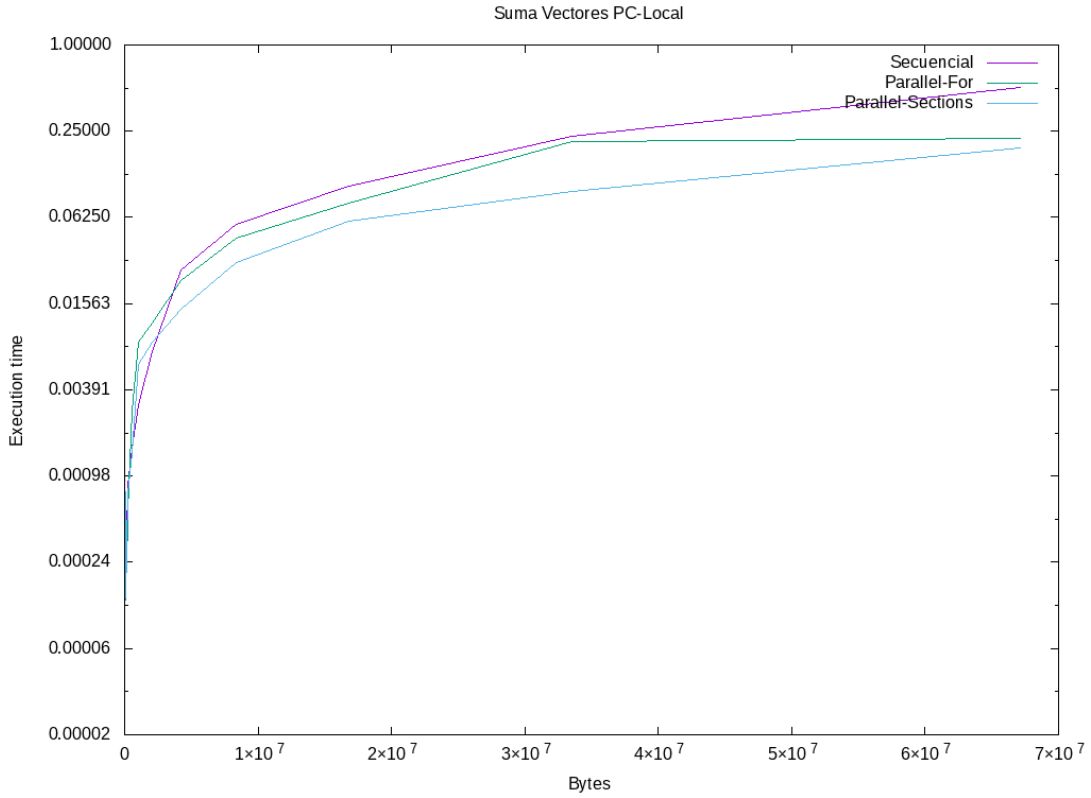


. ¿Cuántos threads y cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 7? Razone su respuesta. ¿Cuántos threads y cuantos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta.

RESPUESTA: En el ejercicio 7 podrías tener un máximo de N hebras, siendo N el número que le pasamos como parámetro. En el ejercicio 8 podríamos utilizar como máximo el número de sections que podamos tener, ese sería el número de hebras máximo, ya que estamos repartiendo las tareas.

. 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:



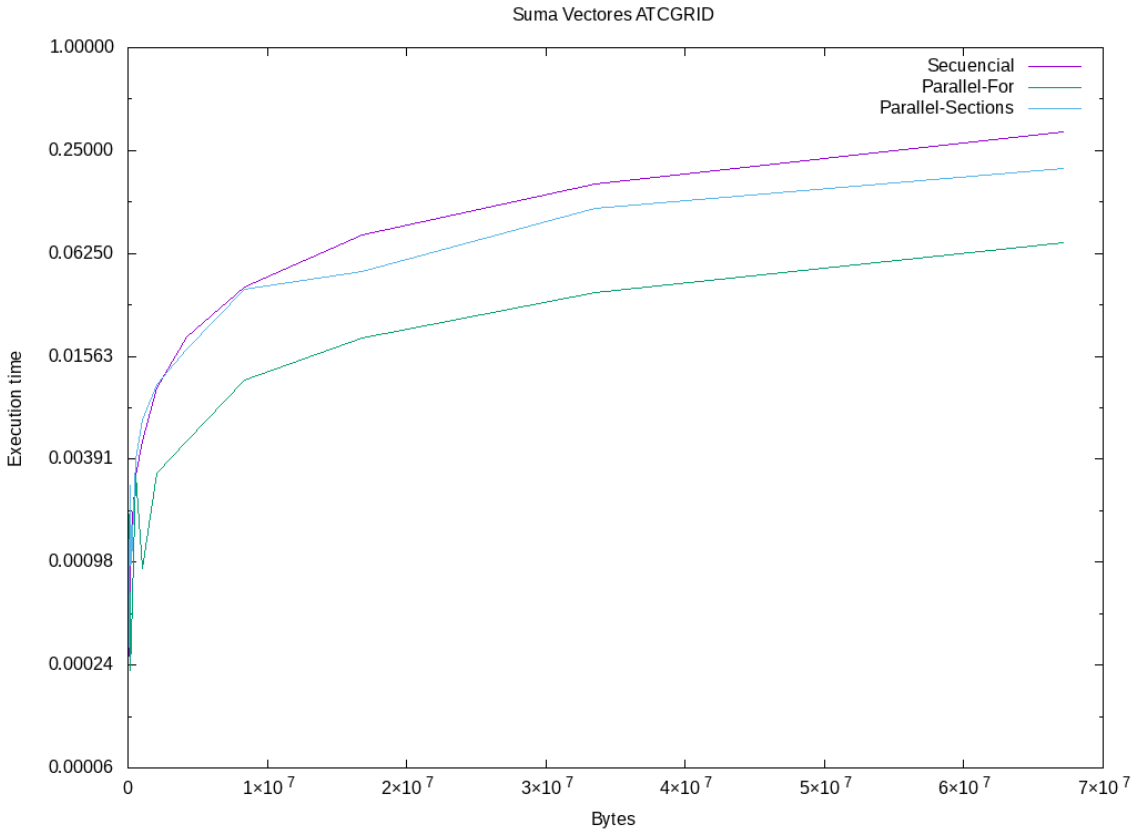

Suma Vectores ATCGRID
Secuencial
Parallel-For
Parallel-Sections
Execution time
1.00000
0.25000
0.06250
0.01563
0.00391
0.00098
0.00024
0.00006
0
1×10 7
2×10 7
3×10 7
4×10 7
5×10 7
6×10 7
7×10 7
Bytes

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. PCLOCAL

Nº de Componentes	**T. secuencial vect. Dynámicos** **1 thread/core**	**T. paralelo (versión `for`)** **2 threads/cores**	**T. paralelo (versión `sections`)** **2 threads/cores**
16384	0.000047946	0.000022760	0.000039827
32768	0.000097061	0.000051076	0.004193416
65536	0.000189542	0.000336358	0.000137150
131072	0.000377851	0.000207543	0.000309705
262144	0.000839413	0.000555377	0.000745363
524288	0.001523382	0.002768770	0.001413316
1048576	0.003053517	0.008475130	0.005897371
2097152	0.007244013	0.011501902	0.008415367
4194304	0.027049520	0.022491466	0.014291642
8388608	0.055713120	0.044926836	0.030455263
16777216	0.102839355	0.078825724	0.058926929
33554432	0.231690915	0.212583449	0.095663266
67108864	0.508441478	0.222583449	0.189932324

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. ATCGRID

Nº de Componentes	T. secuencial vect. Dynámicos 1 thread/core	T. paralelo (versión `for`) 4 threads/cores	T. paralelo (versión `sections`) 2 threads/cores
16384	0.000093281	0.005774570	0.007034690
32768	0.000186296	0.000079145	0.005653929
65536	0.000398285	0.001865455	0.000308584
131072	0.000713780	0.000225610	0.002759208
262144	0.001438568	0.000343292	0.000948331
524288	0.003134933	0.003125697	0.003888329
1048576	0.005024901	0.000895230	0.006673508
2097152	0.010202518	0.003246863	0.010758951
4194304	0.020170575	0.004969477	0.017266343
8388608	0.039859308	0.011270663	0.038468449
16777216	0.080052877	0.020095688	0.048696542
33554432	0.159873026	0.036663525	0.115540072
67108864	0.321719829	0.072489168	0.196610802

. 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:

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. Dynamic 1 thread/core**			**Tiempo paralelo/versión `for` 2 Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	0m0.002s	0m0.003s	0m0.000s	0m0.044s	0m0.010s	0m0.007s
131072	0m0.003s	0m0.000s	0m0.000s	0m0.002s	0m0.000s	0m0.003s
262144	0m0.003s	0m0.000s	0m0.000s	0m0.003s	0m0.000s	0m0.003s
524288	0m0.005s	0m0.003s	0m0.000s	0m0.005s	0m0.007s	0m0.003s
1048576	0m0.009s	0m0.003s	0m0.003s	0m0.008s	0m0.023s	0m0.000s
2097152	0m0.017s	0m0.007s	0m0.007s	0m0.019s	0m0.047s	0m0.010s
4194304	0m0.036s	0m0.023s	0m0.010s	0m0.038s	0m0.083s	0m0.023s
8388608	0m0.060s	0m0.027s	0m0.030s	0m0.057s	0m0.120s	0m0.063s
16777216	0m0.113s	0m0.080s	0m0.030s	0m0.104s	0m0.250s	0m0.107s
33554432	0m0.224s	0m0.150s	0m0.070s	0m0.207s	0m0.487s	0m0.230s
67108864	0m0.997s	0m0.310s	0m0.457s	0m0.421s	0m0.773s	0m0.713s