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): Ivan Ortega Alba
Grupo de prácticas: C2
Fecha de entrega: 09/04/2014
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());
	}
int main() {
	int i = 0, j = 5, n = 10;

	#pragma omp parallel
	{
		#pragma omp sections
		{
			#pragma omp section
			{
			(void) funcA();
			for (i; i<n/2; i++)
				printf("thread %d ejecuta la iteración %d del
bucle\n",omp_get_thread_num(),i);
			}
			#pragma omp section
			{
			(void) funcB();
			for (j; j<n; j++)
				printf("thread %d ejecuta la iteración %d del
bucle\n",omp_get_thread_num(),j);
			}
		}
	}
}
```

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>
int main() {
  int n = 9, i, a, b[n];
  for (i=0; i<n; i++)
    b[i] = -1;

  #pragma omp parallel
  {
    #pragma omp single
    {
    printf("Dentro de la región parallel:\n");
  }
    #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
    {

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

CAPTURAS DE PANTALLA:

```
MacBook-Pro-de-Ivan:p1 ivanortegaalba$ ./singlemod
Dentro de la región parallel:
Introduce valor de inicialización a:2
Single ejecutada por el thread 1
b[0] = 2        b[1] = 2        b[2] = 2        b[3] = 2        b[4] = 2        b
[5] = 2 b[6] = 2        b[7] = 2        b[8] = 2
Single ejecutada por el thread 0
Después de la región parallel:
Single ejecutada por el thread 0
b[0] = 2        b[1] = 2        b[2] = 2        b[3] = 2        b[4] = 2        b
[5] = 2 b[6] = 2        b[7] = 2        b[8] = 2
MacBook-Pro-de-Ivan:p1 ivanortegaalba$
```

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>
int main() {
  int n = 9, i, a, b[n];
  for (i=0; i<n; i++)
    b[i] = -1;

  #pragma omp parallel
  {
    #pragma omp single
```

```
    {
    printf("Dentro de la región parallel:\n");
  }
    #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
    {

      for (i=0; i<n; i++)
        printf("b[%d] = %d\t",i,b[i]);
        printf("\n");
        printf("Master ejecutada por el thread %d\n",
omp_get_thread_num());
    }
  printf("Después de la región parallel:\n");
  printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
  for (i=0; i<n; i++)
    printf("b[%d] = %d\t",i,b[i]);
    printf("\n");
}
```

CAPTURAS DE PANTALLA:

```
MacBook-Pro-de-Ivan:p1 ivanortegaalba$ ./singlemodmaster
Dentro de la región parallel:
Introduce valor de inicialización a:4
Single ejecutada por el thread 0
b[0] = 4        b[1] = 4        b[2] = 4        b[3] = 4        b[4] = 4        b
[5] = 4 b[6] = 4        b[7] = 4        b[8] = 4
Master ejecutada por el thread 0
Después de la región parallel:
Single ejecutada por el thread 0
b[0] = 4        b[1] = 4        b[2] = 4        b[3] = 4        b[4] = 4        b
[5] = 4 b[6] = 4        b[7] = 4        b[8] = 4
MacBook-Pro-de-Ivan:p1 ivanortegaalba$
```

RESPUESTA A LA PREGUNTA:

La diferencia reside en que el ejercicio anterior la impresión de resultados final podía ser ejecutado por cualquier thread, mientras que en éste ejercicio, el print final siempre será ejecutado por el thread master (el 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 la directiva `atomic` no tiene una barrera implícita al final, por lo que si el thread `master` llega a la directiva `master` antes que el resto, ejecutará su `print`

correspondiente, y puede que la suma no esté completa, porque no todos los thread hayan alcanzado dicha directiva `master`

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 para el ejecutable 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.

   RESPUESTA:

   Es 0.01 segundo menos ya que también hay que tener en cuenta el tiempo empleado en I/O.

   CAPTURAS DE PANTALLA:

   Realizado en la máquina de un amigo, un Linux 64 bits con un i7 de 8 cores virtuales. (No tengo Linux nativo y no quiero hacerlo en una VM)

```
nuwanda@nuwanda-HP-Pavilion-g6-Notebook-PC:/media/windows/Users/Nuwanda/Desktop/Universidad/2
racticas/P1/5$ g++  -O2 SumaVectoresCpp.cpp -o SumaVectoresCpp -lrt
nuwanda@nuwanda-HP-Pavilion-g6-Notebook-PC:/media/windows/Users/Nuwanda/Desktop/Universidad/2
/Cuatrimestre II/AC, Practicas/Practicas/P1/5$ time ./SumaVectoresCpp 10000000
Tiempo(seg.):0.0324472  / Tamaño Vectores:10000000      / V1[0]+V2[0]=V3[0](1e+06+1e+06=2e+06
) / / V1[9999999]+V2[9999999]=V3[9999999](2e+06+0.1=2e+06)/


real    0m0.122s
user    0m0.051s
sys     0m0.070s
nuwanda@nuwanda-HP-Pavilion-g6-Notebook-PC:/media/windows/Users/Nuwanda/Desktop/Universidad/2
/Cuatrimestre II/AC, Practicas/Practicas/P1/5$
```

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:

```
[B3estudiante27@atcgrid P1]$ gcc -S -O2 -fopenmp SumaVectoresC.c
[B3estudiante27@atcgrid P1]$ ls
bfm-for                     for-mod                   singlemodmaster
bfm-sec                     secmod                    SumaVectoresC.c
bucle-for-modificado.c      singlemod                 SumaVectoresC.s
bucle-sections-master.c     single-modificado.c
bucle-sections-modificado.c single-modificado-master.c
[B3estudiante27@atcgrid P1]$ cat SumaVectoresC.s
        .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:
```

```
MacBook-Pro-de-Ivan:p1 ivanortegaalba$ ./SumaVectoresC 10
Tiempo(seg.):0.000002861         / Tamaño Vectores:10   / V1[0]+V2[0]=V3[0](1.00
0000+1.000000=2.000000)//V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /
MacBook-Pro-de-Ivan:p1 ivanortegaalba$ ./SumaVectoresC 10000000
Tiempo(seg.):0.062431097         / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3
[0](1000000.000000+1000000.000000=2000000.000000)//V1[9999999]+V2[9999999]=V3[99
99999](1999999.900000+0.100000=2000000.000000) /
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

NI (MIPS): 3+(6*N)

NI (MFLOPS): 3*N

10 componentes

MIPS=3+(6*10)/0.000002861*10^6 = 22.0202726319

MFLOPS=(3+10)/0.000002861*10^6 =45.438657812

10000000 componentes

MIPS=(3+(6*10000000))/(0.062431097*(10^6)) = 961.059566197

MFLOPS=10000003/(0.062431097*(10^6)) = 160.17663441

RESPUESTA: código ensamblador generado de la parte de la suma de vectores

```
                call            clock_gettime                   *AQUI
                xorl            %eax, %eax
                .p2align 4,,10
                .p2align 3
.L7:                            //Bucle for
                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                 *AQUI
```

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 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()
#include <omp.h>
//#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;
  // struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución
  // //Leer argumento de entrada (n° de componentes del vector)
  double start, end, elapsed;

  if (argc<2){
    printf("Faltan n° 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
    }

  //Calcular suma de vectores
  start= omp_get_wtime( );

  #pragma omp parallel for
    for(i=0; i<N; i++)
      v3[i] = v1[i] + v2[i];

  end =  omp_get_wtime( );
  elapsed = end - start;
  //Imprimir resultado de la suma y el tiempo de ejecución
  #ifdef PRINTF_ALL
  printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",elapsed,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",elapsed,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
  #endif
  printf("Vector resultante v3[8],v3[11]:\n \t V3[8]=%d,
V3[11]=%d\n",v3[8],v3[11]);
  #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:

```
MacBook-Pro-de-Ivan:p1 ivanortegaalba$ ./SumaVectoresParalelizado 20
Tiempo(seg.):0.000047922        / Tamaño Vectores:20   / V1[0]+V2[0]=V3[0](2.00
0000+2.000000=4.000000)//V1[19]+V2[19]=V3[19](3.900000+0.100000=4.000000) /
Vector resultante v3[8],v3[11]:
        V3[8]=256, V3[11]=1923617064
MacBook-Pro-de-Ivan:p1 ivanortegaalba$
```

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 <time.h> // biblioteca donde se encuentra la función clock_gettime()
#include <omp.h>
//#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;
  // //Leer argumento de entrada (n° de componentes del vector)
  double start, end, elapsed;
```

```
  if (argc<2){
    printf("Faltan n° componentes del vector\n");
    exit(-1);
  }
  unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)
  unsigned int N1 = (N / 2) , N2 = (N / 2) + (N % 2);
  #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 paralelizado
  #pragma omp parallel sections
  {
    #pragma omp section
      for(int i=0; i<N/8; i++){
        v1[i] = N*0.1+i*0.1;
      }
    #pragma omp section
      for(int i=0; i<N/4; i++){
        v1[i] = N*0.1+i*0.1;
      }
    #pragma omp section
      for(int i=0; i<(3*N/8); i++){
        v1[i] = N*0.1+i*0.1;
      }
    #pragma omp section
      for(int i=0; i<N/2; i++){
        v1[i] = N*0.1+i*0.1;
      }
    #pragma omp section
      for(int i=0; i<(5*N/8); i++){
        v1[i] = N*0.1+i*0.1;
      }
    #pragma omp section
      for(int i=0; i<(3*N/4); i++){
        v1[i] = N*0.1+i*0.1;
      }
    #pragma omp section
      for(int i=0; i<(7*N/8); i++){
        v1[i] = N*0.1+i*0.1;
```

```
      }
    #pragma omp section
      for(int i=0; i<N; i++){
        v1[i] = N*0.1+i*0.1;
      }
  }
  //Calcular suma de vectores
  start = omp_get_wtime();

  #pragma omp parallel sections
  {
    #pragma omp section
      for(int i=0; i<N/8; i++){
        v3[i] = v1[i] + v2[i];
      }
    #pragma omp section
      for(int i=0; i<N/4; i++){
        v3[i] = v1[i] + v2[i];
      }
    #pragma omp section
      for(int i=0; i<(3*N/8); i++){
        v3[i] = v1[i] + v2[i];
      }
      #pragma omp section
        for(int i=0; i<N/2; i++){
          v3[i] = v1[i] + v2[i];
        }
      #pragma omp section
        for(int i=0; i<(5*N/8); i++){
          v3[i] = v1[i] + v2[i];
        }
      #pragma omp section
        for(int i=0; i<(3*N/4); i++){
          v3[i] = v1[i] + v2[i];
      }
      #pragma omp section
        for(int i=0; i<(7*N/8); i++){
          v3[i] = v1[i] + v2[i];
        }
      #pragma omp section
        for(int i=0; i<N; i++){
          v3[i] = v1[i] + v2[i];
        }
    }

  end =  omp_get_wtime();

  elapsed = end - start;
  //Imprimir resultado de la suma y el tiempo de ejecución
  #ifdef PRINTF_ALL
  printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",elapsed,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",elapsed,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
  #endif

  printf("Vector resultante v3[8],v3[11]:\n \t V3[8]=%d,
```

```
V3[11]=%d\n",v3[8],v3[11]);

  #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:

```
MacBook-Pro-de-Ivan:p1 ivanortegaalba$ ./SumaVectoresSections 20
Tiempo(seg.):0.000086069         / Tamaño Vectores:20   / V1[0]+V2[0]=V3[0](2.00
0000+0.000000=2.000000)//V1[19]+V2[19]=V3[19](3.900000+0.000000=3.900000) /
Vector resultante v3[8],v3[11]:
         V3[8]=256, V3[11]=1923617064
```

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:

Tantas hebras como hebras máximas tengamos definidas, ya que dividirá las interacciones del `for` entre las hebras máximas o disponibles. A no ser que el número de iteraciones sea menor que el numero de hebras máximas o disponibles. Donde en tal caso serán tantas como iteraciones de el bucle.

Para el ejercicio 8, utilizará tantas como máximas o disponibles haya, si el número de `sections` es mayor. De lo contrario usará una hebra por cada `section`.

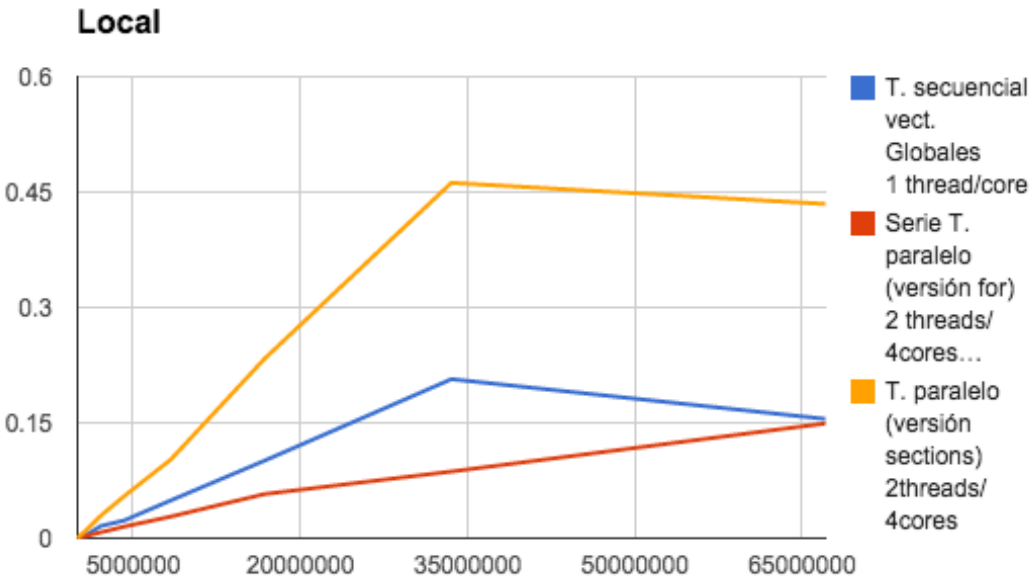
10. Rellenar una tabla como la Tabla 2 para atcgrid y otra para el PC local con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución del trozo 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. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados, que debe coincidir con el número de cores físicos utilizados.

TIEMPOS EN LOCAL

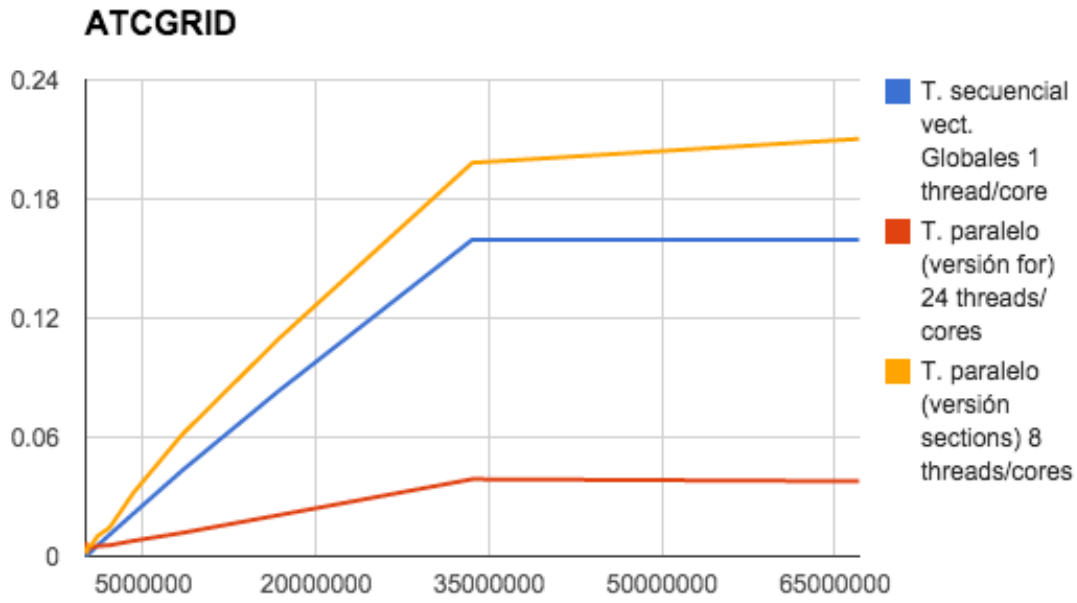
Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión `for`) 8 threads/cores	T. paralelo (versión `sections`) 8 threads/cores
16384	0.000349045	0.000110865	0.000280857
32768	0.000201941	0.000277996	0.000562191
65536	0.000399113	0.000392199	0.000846148
131072	0.000717878	0.000532150	0.001754045
262144	0.001586199	0.000963926	0.003453970
524288	0.003388882	0.002132177	0.006901979
1048576	0.006395102	0.003816128	0.013885021
2097152	0.015784025	0.007569075	0.029118061
4194304	0.023000002	0.015020847	0.054545879
8388608	0.049160004	0.028101921	0.102366924
16777216	0.100707054	0.057315111	0.233089924
33554432	0.206789970	0.086694002	0.462047100
67108864	0.155066013	0.149203062	0.434586048



TIEMPOS EN ATCGRID

Nº de	T. secuencial	T. paralelo	T. paralelo (versión

Componentes	vect. Globales 1 thread/core	(versión `for`) 8 threads/cores	`sections`) 8 threads/cores
16384	0.000101225	0.007222363	0.001799191
32768	0.000204246	0.003993563	0.002377362
65536	0.000401394	0.004612314	0.004092966
131072	0.000813813	0.001875110	0.003044942
262144	0.001195317	0.004514269	0.005318363
524288	0.002524927	0.004231805	0.005624921
1048576	0.005491663	0.005209098	0.010196138
2097152	0.010927182	0.005564016	0.014486940
4194304	0.021725446	0.007850455	0.032244172
8388608	0.043230731	0.011791516	0.061375547
16777216	0.083389303	0.020750833	0.109642552
33554432	0.159547245	0.038841992	0.198247817
67108864	0.159516953	0.037830352	0.210291847



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 es menor ya que el elapsed incluye el tiempo del user y el tiempo empleado en I/O

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	0.02-0.00-0.00			0.01-0.00-0.00		
131072	0.00-0.00-0.00			0.00-0.00-0.00		
262144	0.00-0.00-0.00			0.00-0.00-0.00		
524288	0.01-0.00-0.00			0.00-0.00-0.01		
1048576	0.02-0.00-0.00			0.01-0.01-0.01		
2097152	0.03-0.01-0.01			0.02-0.01-0.03		
4194304	0.06-0.03-0.03			0.04-0.03-0.07		
8388608	0.13-0.06-0.06			0.07-0.07-0.14		
16777216	0.30-0.12-0.15			0.16-0.13-0.30		
33554432	0.76-0.25-0.30			0.26-0.30-0.44		
67108864	0.58-0.22-0.30			0.42-0.31-0.46		