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): Antonio Miguel Morillo Chica
Grupo de prácticas: D1
Fecha de entrega: 06/04/2016
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`

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

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

  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
```

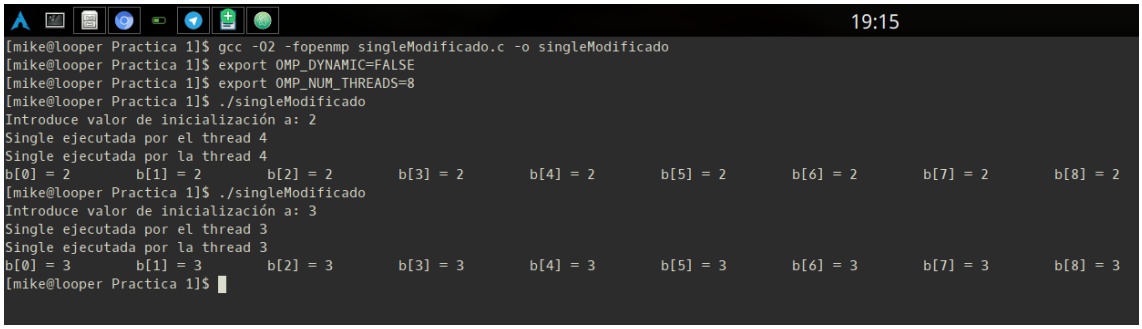
```
%d\n",omp_get_thread_num());
  }


  #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 `singleModificado.c`

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

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

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("Single el ejecutada por la 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:

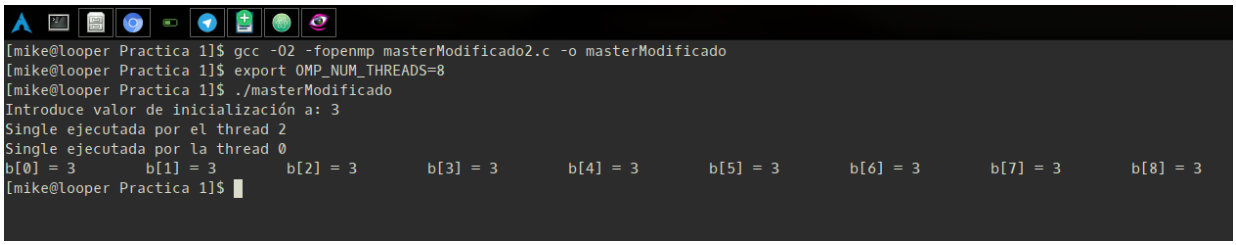


. Imprimir los resultados del programa `single.c` usando una directiva `master` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `master` incorporada, para que se imprima el identificador del thread que ejecuta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


int main(int argc, char ** argv){
  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("Single ejecutada por la 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:



RESPUESTA A LA PREGUNTA: Con la directiva single el thread que ejecuta los elementos del vector es cualquiera y con la directiva master el thread que lo ejecuta es al que denominamos padre, es decir, el que a partir de él se dividen los demás threads, y es el threads 0.

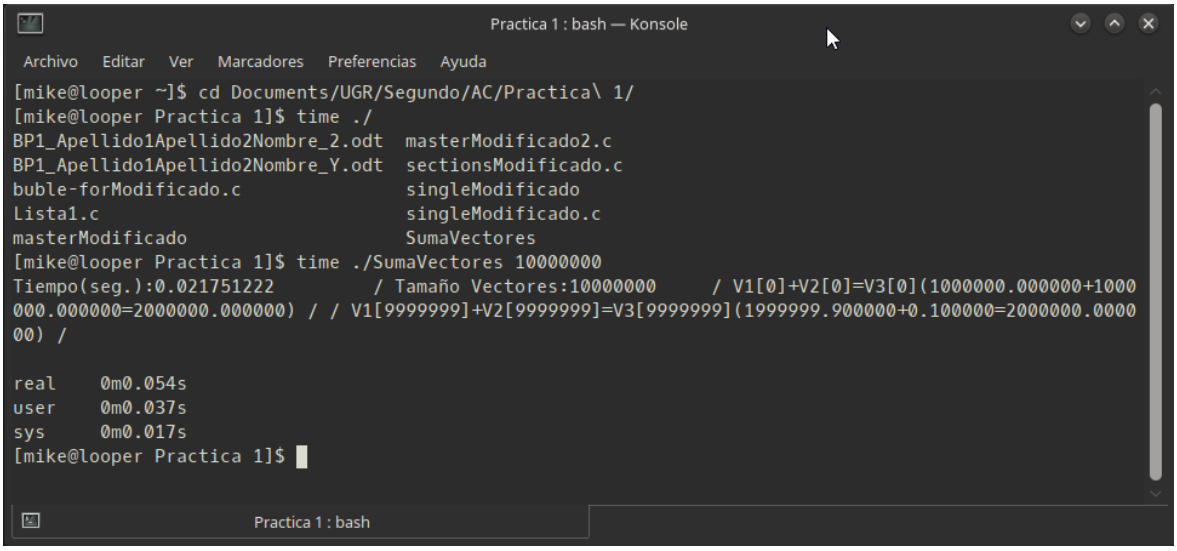
. ¿Por qué si se elimina directiva `barrier` en el ejemplo `master.c` la suma que se calcula e imprime no siempre es correcta? Responda razonadamente.

RESPUESTA: Cada hilo espera a que acabe el de antes para poder entrar. Cuando quitas barrier quitas la trestrinción y, por tanto, puede entrar un hilo sin que el otro hubiese acabado de ejecutar (ejemplo: hilos 0,1,2,3, sin barrier podría terminar antes de que todos hayan entrado), por lo que se cambiaría el valor de la suma y no se imprimiría la correcta.

Resto de ejercicios

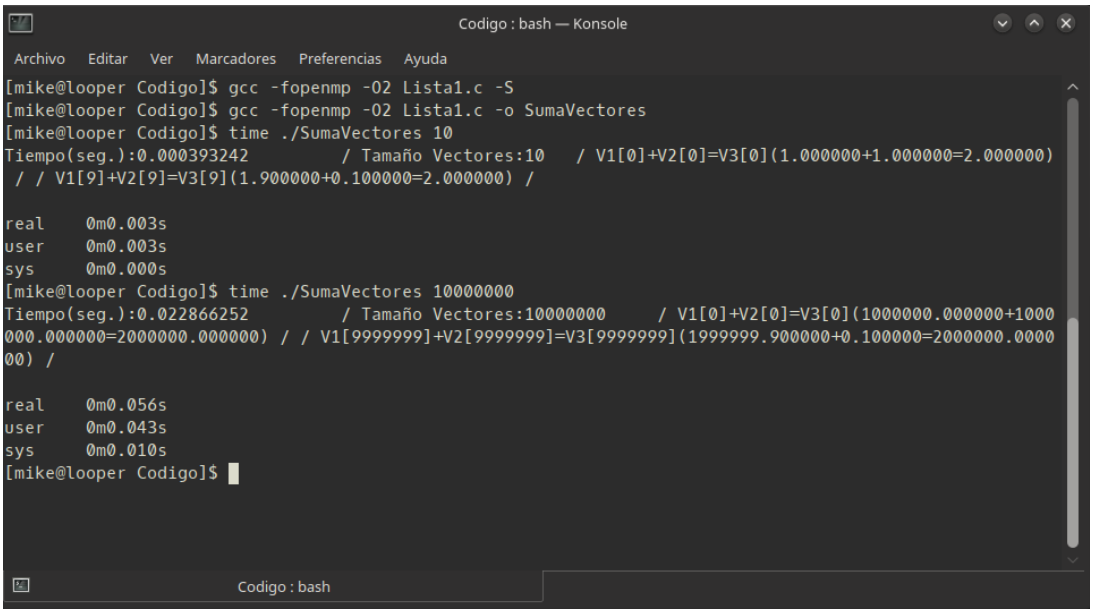
. El programa secuencial C del Listado 1 calcula la suma de dos vectores (v3 = v1 + v2; v3(i) = v1(i) + v2(i), i=0,…N-1). Generar el ejecutable del programa del Listado 1 para **vectores globales**. Usar `time` (Lección 3/ Tema 1) en la línea de comandos para obtener, en el PC local, el tiempo de ejecución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

CAPTURAS DE PANTALLA:



. 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: Con 10000000 componentes:

MIPS = NI / T_CPU * 10^6 = Frecuencia / CPIx10^6

MFLOPS = Operaciones_coma_flotante / T_CPUx10^6

Hay 11 instrucciones entre las dos funciones clock_gettime que será ejecutadas cada componente del vector que sabemos que hay 10000000.

NI = 110000000 instuciones (multiplicar 11 * vector)

T_CPU = 0,022866252

MIPS = 110000000 / (0,022866 * 1000000) = 4810,526

Hay 3 instrucciones en como flotante.

NI = 30000000

MFLOPS = 30000000 / (0,022866 * 10000000) = 1311,962

RESPUESTA: Con 10 componentes

NI = 11*10=110

T_CPU = 0,000393242

MIPS = 110 / (0,00039324 * 1000000) = 0,279726

Con tres instrucciones en coma flotante

NI = 30 instrucciones

MFLOPS = 30 / (0,00039324 * 1000000) = 0,0762889

Código ensamblador generado de la parte de la suma de vectores

```
                Call            clock_gettime
                movl            $0, -20(%rbp)
.               jump            .L10

.L11:
                movl            -20(%rbd),%edx
                movl            -20(%rbp),%eax
                cltq
                movsd           v1(,%rax,8),%xmm1
                movl            -20(%rbp),%eax
                cltq
                movsd           v2(,%raw,8),%xmm0
                addsd           %xmm1,%xmm0
                movslq          %edx,%rax
                movsd           %xmm0, v3(,%rax,8)
                addl            $1,-20(%rbp)
.L10:
                movl            -20(%rbp),%eax
                cmpl            -24(%rbp),%eax
.               jb              .L11
                leaq            -64(%rbp),%rax
                movq            %rax,%rsi
                movl            $0,%edi
                call            clock_gettime
```

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

//Sólo puede estar definida una de las tres constantes VECTOR_ (sólo uno de los ...
//tres defines siguientes puede estar descomentado):

//#define VECTOR_LOCAL
#define VECTOR_GLOBAL
//#define VECTOR_DYNAMIC

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)

// No sé porque pero no me funciona con ifdef y debería..
// Da un error de compilación que no sabía resolver. Y la única manera que he
// encontrado ha sido decomentando estas lineas.
// #ifdef VECTOR_LOCAL
double v1[N], v2[N], v3[N]; // Tamaño variable local
// #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

#pragma omp parallel
{
//Inicializar vectores
  #pragma omp 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
    //printf("/ V1[%d] y V2[%d](%8.6f y %8.6f) \n", i,i,v1[i],v2[i]);
}

#pragma omp master
cgt1=omp_get_wtime();

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

#pragma omp master
cgt2=omp_get_wtime();
ncgt=(double) (cgt2-cgt1);

//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);
#pragma omp master
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

#pragma omp master
   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;

}
```

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


//#define VECTOR_LOCAL
#define VECTOR_GLOBAL
//#define VECTOR_DYNAMIC

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
#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


#pragma omp parallel private(i)
{
double v1[N], v2[N], v3[N];


#pragma omp sections
{
#pragma omp section
  for(i=0; i<N/2; i++){
   v1[i] = N*0.1+i*0.1;
}
#pragma omp section
  for(i=N/2; i<N; i++){
  v1[i] = N*0.1+i*0.1;
}
#pragma omp section
  for(i=0; i<N/2; i++){
  v2[i] = N*0.1-i*0.1;
}
#pragma omp section
  for(i=N/2; i<N; i++){
  v2[i] = N*0.1-i*0.1;
  }
}


cgt1=omp_get_wtime();


//Calcular suma de vectores
#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=(double) (cgt2-cgt1);


//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);
  #pragma omp master
  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
#pragma omp master
  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;
}
```

. ¿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 lo realizamos con la directiva for, por lo tanto podría utilizarse tantos threads y cores como el ordenador tenga porque se divide todas las iteraciones entre los cores disponibles. En cuando al ejercicio 8 se utiliza el mismo numero de cores como directivas section tengamos, como lo hemos dividido en 4 se utilizarán como máximo 4 ya que en la siguiente región sections solo tenemos 2 section dentro.

. 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

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. PC LOCAL

Nº de Componentes	**T. secuencial vect. Globales 1 thread/core**	**T. paralelo (versión `for`) 4threads/cores**	**T. paralelo (versión `sections`) 4threads/cores**
65536	0.000692543	0.000364832	0.000971466
131072	0.001292720	0.000703536	0.001184314
262144	0.002826540	0.003088300	0.004509690
524288	0.003377562	0.006385484	0.009203388
1048576	0.004852049	0.007830096	0.007634332
2097152	0.008998421	0.017574048	0.015261752
4194304	0.017900689	0.029185456	0.027913788
8388608	0.035608380	0.053873132	0.057765652
16777216	0.071126819	0.105498462	0.110294452
33554432	0.140789624	0.209360758	0.206373390
67108864	0.269156392	0.380979226	0.382059140

los códigos. Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

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. Globales 1 thread/core**	**T. paralelo (versión `for`) 4threads/cores**	**T. paralelo (versión `sections`) 4threads/cores**
65536	0.000551998	0.005007286	0.0000001610
131072	0.000704173	0.003297825	0.000724921
262144	0.001449972	0.00386299	0.001072048
524288	0.002627603	0.004496637	0.001963218
1048576	0.005202697	0.005172230	0.002417654
2097152	0.008788232	0.006518500	0.008656132
4194304	0.017245525	0.009985239	0.009965483
8388608	0.034113234	0.016266697	0.022513803
16777216	0.066960004	0.030133680	0.042147916
33554432	0.130894112	0.058518531	0.087501103
67108864	0.253834767	0.112556787	0.135472637

. 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. Globales 1 thread/core**			**Tiempo paralelo/versión `for` 4 Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	0.027s	0.000s	0.004s	0.014s	0.000s	0.000s
131072	0.023s	0.004s	0.004s	0.017s	0.000s	0.004s
262144	0.030s	0.004s	0.008s	0.022s	0.004s	0.004s
524288	0.036s	0.000s	0.012s	0.032s	0.004s	0.008s
1048576	0.077s	0.012s	0.016s	0.086s	0.004s	0.028s
2097152	0.126s	0.012s	0.040s	0.135s	0.012s	0.048s
4194304	0.191s	0.016s	0.092s	0.198s	0.020s	0.092s
8388608	0.333s	0.040s	0.168s	0.453s	0.056s	0.260s
16777216	0.910s	0.104s	0.344s	0.832s	0.068s	0.548s
33554432	1.646s	0.208s	0.668s	1.364s	0.212s	0.872s
67108864	2.909s	0.444s	1.132s	1.642s	0.296s	1.068s