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): Jose Miguel Hernández García
Grupo de prácticas: C3
Fecha de entrega: 29-03-17
Fecha evaluación en clase: 30-03-17

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){
        printf(stderr, "\n[ERROR] - Falta n° iteraciones \n");
        exit(-1);
    }
    n = atoi(argv[1]);

    #pragma omp parallel for
        for(i = 0; i < n; i++)
            printf("thread %d ejecuta la iteracion %d del bucle \n",
                    omp_get_thread_num(), i);
    return 0;
}
```

RESPUESTA: código fuente `sectionsModificado.c`

```
#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif

void funcA(){
    printf("En funcA: esta seccion la ejecuta el thread %d\n",
            omp_get_thread_num());
}

void funcB(){
    printf("En funcB: esta seccion la ejecuta el thread %d\n",
            omp_get_thread_num());
}

void main(){
```

```
    #pragma omp sections
    {
        #pragma omp section
            (void)funcA();
        #pragma omp section
            (void)funcB();
    }
}
```

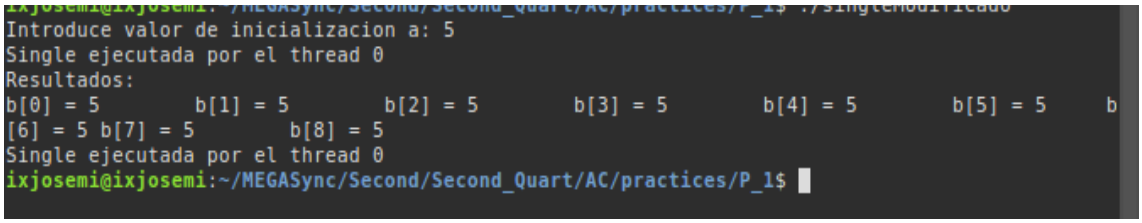
. Imprimir los resultados del programa `single.c` usando una directiva `single` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `single` incorporada, para que se imprima el identificador del thread que ejecuta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

```
#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif


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 inicializacion 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("Resultados:\n");


            for(i = 0; i < n; i++)
                printf("b[%d] = %d\t", i b[i]);


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

```
}
```

CAPTURAS DE PANTALLA:

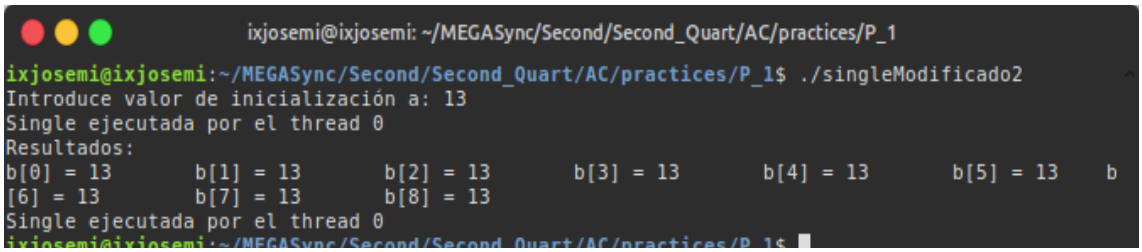


. Imprimir los resultados del programa `single.c` usando una directiva `master` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `master` incorporada, para que se imprima el identificador del thread que ejecuta 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>
#ifdef _OPENMP
                #include <omp.h>
#else
                #define omp_get_thread_num() 0
                #define omp_get_num_threads() 1
#endif

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("Resultados:\n");

            for(i=0; i<n; i++)
               printf("b[%d] = %d\t", i, b[i]);

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

CAPTURAS DE PANTALLA:



RESPUESTA A LA PREGUNTA:

La diferencia que se observa es que al utilizar la directiva master, los resultados siempre los pintará el thread 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.

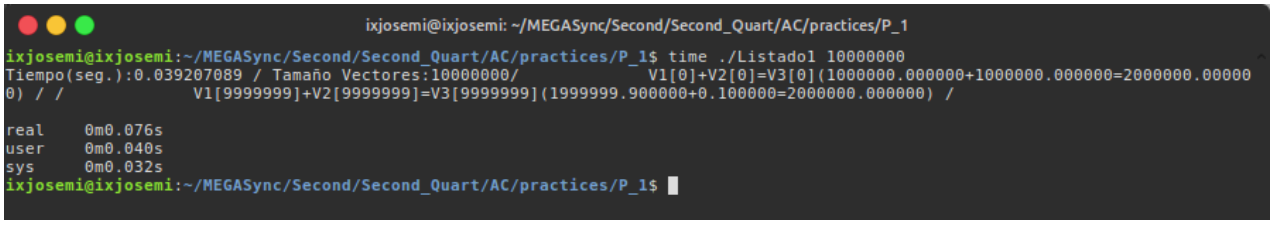
RESPUESTA:

Porque si el thread numero 0 se ejecuta antes que el resto, pintará el resultado sin esperar a que el resto de hilos termine.

1.1.1 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: cálculo de los MIPS y los MFLOPS

Disponemos de 6 instrucciones dentro del bucle y 7 fuera, por tanto, si tenemos un tamaño de vector 10, el numero de instrucciones será 6*10 +7.

Tam_vector = 10

Tiempo(s) = 0.000006705

MIPS = 67 / (0.000006705 * 10^6) = 9,992542878 millones de instrucciones por segundo.

MFLOPS = 30 / (0.000006705 * 10^6) = 4,474272931 millones de operaciones en coma flotante por segundo.

Disponemos de 6 instrucciones dentro del bucle y 7 fuera, por tanto, si tenemos un tamaño de vector 10000000 , el numero de instrucciones será 6*10000000 +7.

Tam_vector = 10000000

Tiempo(s) = 0.287034231

MIPS = 60000007 / (0.287034231*10^6) = 209,0343259 millones de instrucciones por segundo.

MFLOPS = 30000000 / (0.287034231*10^6) = 104,5171508 millones de operaciones en coma flotante por segundo.

RESPUESTA:

código ensamblador generado de la parte de la suma de vectores

```
                call            clock_gettime
                xorl            %eax, %eax
                .p2align 4,,10
                .p2align 3
.L5:
                movsd           v1(%rax), %xmm0
                addq            $8, %rax
                addsd           v2-8(%rax), %xmm0
                movsd           %xmm0, v3-8(%rax)
                cmpq            %rax, %rbx
                jne             .L5
.L6:
                leaq            16(%rsp), %rsi
                xorl            %edi, %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).

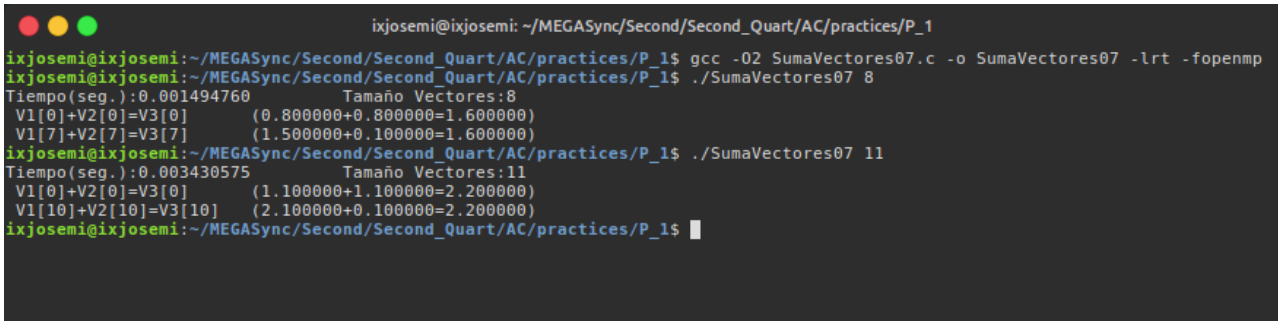
RESPUESTA: código fuente implementado

```
/* SumaVectores07.c
Suma de dos vectores: v3 = v1 + v2
Para compilar usar (-lrt: real time library):
gcc -O2 SumaVectores07.c -o SumaVectores07 -lrt
gcc -O2 –S SumaVectores07.c –lrt //para generar el código ensamblador
Para ejecutar use: SumaVectores07 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

#ifdef _OPENMP
               #include <omp.h>
#else
               #define omp_get_thread_num() 0
               #define omp_get_num_threads() 1
#endif

int main(int argc, char** argv){
    int i;
    double cgt1,cgt2;
    double ncgt; // para tiempo de ejecución

    // Leer argumento de entrada (n° de componentes del vector)
    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)

    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);
    }

    // Inicializar vectores
    #pragma omp parallel
    {
        #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
        }

        #pragma omp single
        {
```

```
            cgt1 = omp_get_wtime();
        }

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

        #pragma omp single
        {
            cgt2 = omp_get_wtime();
        }
    }
    ncgt = 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);

        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\n V1[0]+V2[0]=V3[0]\t(%8.6f+
%8.6f=%8.6f)\n V1[%d]+V2[%d]=V3[%d]\t(%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

    free(v1); // libera el espacio reservado para v1
    free(v2); // libera el espacio reservado para v2
    free(v3); // libera el espacio reservado para v3

    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

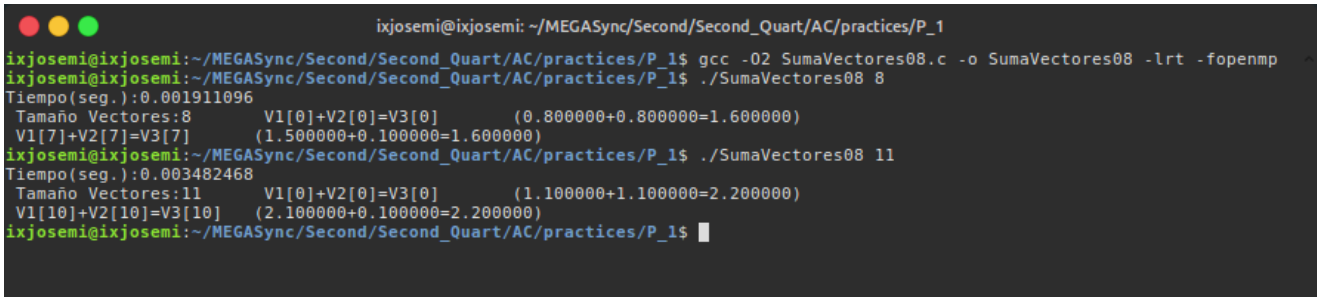
```
/* SumaVectores08.c
Suma de dos vectores: v3 = v1 + v2
Para compilar usar (-lrt: real time library):
gcc -O2 SumaVectores08.c -o SumaVectores08 -lrt
gcc -O2 –S SumaVectores08.c –lrt //para generar el código ensamblador
Para ejecutar use: SumaVectores08 longitud
*/

#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

#ifdef _OPENMP
    #include <omp.h>
#else
               #define omp_get_thread_num() 0
               #define omp_get_num_threads() 1
#endif

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

    int i;
    double cgt1,cgt2;
    double ncgt; // para tiempo de ejecución

    // Leer argumento de entrada (n° de componentes del vector)
    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)

    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);
               }

    // Inicializar vectores
    #pragma omp parallel private(i)
    {
        #pragma omp sections
        {
            #pragma omp section
            for(i=0; i<N/4; i++){
                v1[i] = N*0.1+i*0.1;
```

```
                v2[i] = N*0.1-i*0.1;
            }

            #pragma omp section
            for(i=N/4; i<N/2; i++){
                v1[i] = N*0.1+i*0.1;
                v2[i] = N*0.1-i*0.1;
            }

            #pragma omp section
            for(i=N/2; i<3*N/4; i++){
                v1[i] = N*0.1+i*0.1;
                v2[i] = N*0.1-i*0.1;
            }

            #pragma omp section
            for(i=3*N/4; i<N; i++){
                v1[i] = N*0.1+i*0.1;
                v2[i] = N*0.1-i*0.1;
            }
        }

        #pragma omp single
        {
            cgt1 = omp_get_wtime();
        }

        // Calcular suma de vectores
        #pragma omp sections
        {
            // Dividimos las iteraciones for de forma manual en 4 pedazos
            #pragma omp section
            for(i=0; i<N/4; i++)
                v3[i] = v1[i] + v2[i];

            #pragma omp section
            for(i=N/4; i<N/2; i++)
                v3[i] = v1[i] + v2[i];

            #pragma omp section
            for(i=N/2; i<3*N/4; i++)
                v3[i] = v1[i] + v2[i];

            #pragma omp section
            for(i=3*N/4; i<N; i++)
                v3[i] = v1[i] + v2[i];
        }

        #pragma omp single
        {
            cgt2 = omp_get_wtime();
        }
    }
    ncgt = 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);

        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\n Tamaño Vectores:%u\t V1[0]+V2[0]=V3[0] \t (%8.6f+
%8.6f=%8.6f)\n V1[%d]+V2[%d]=V3[%d]\t(%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

    free(v1); // libera el espacio reservado para v1
    free(v2); // libera el espacio reservado para v2
    free(v3); // libera el espacio reservado para v3

    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 los dos casos, al no haber definido la variable de entorno OMP_NUM_THREADS, se usarán todos los cores o threads que tenga disponible nuestra máquina, sin embargo, en el ejercicio 8, al haber definido las secciones de forma fija, habrán threads que no hagan nada.

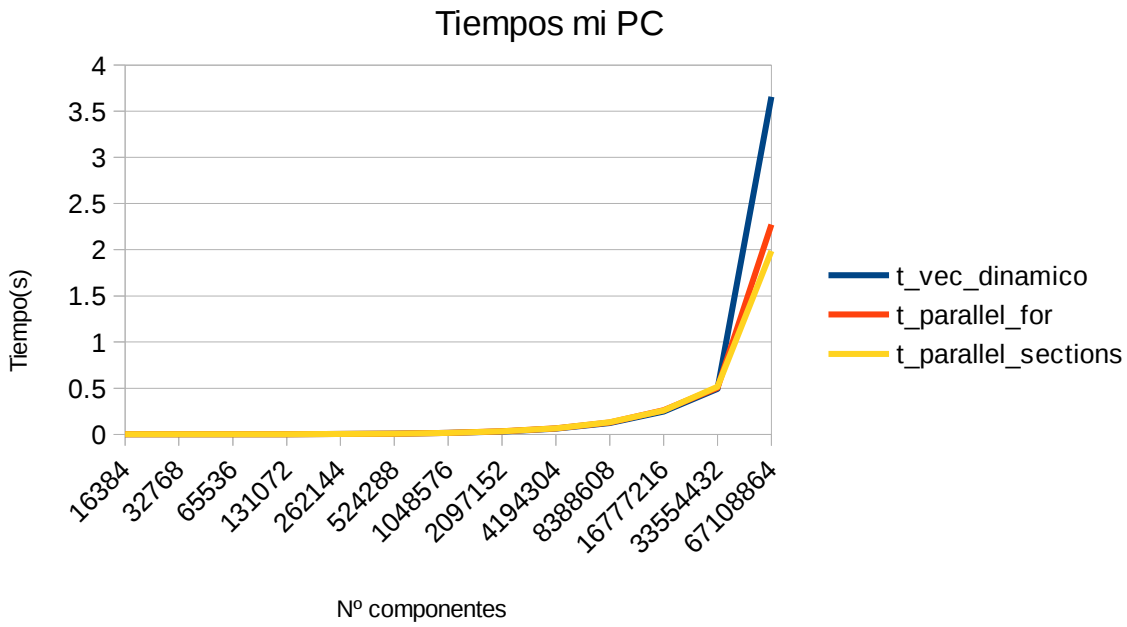
. Rellenar una tabla como la Tabla 2 para atcgrid y otra para el PC local con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución del trozo 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:

PC LOCAL → Intel Core i7 – 4710HQ.

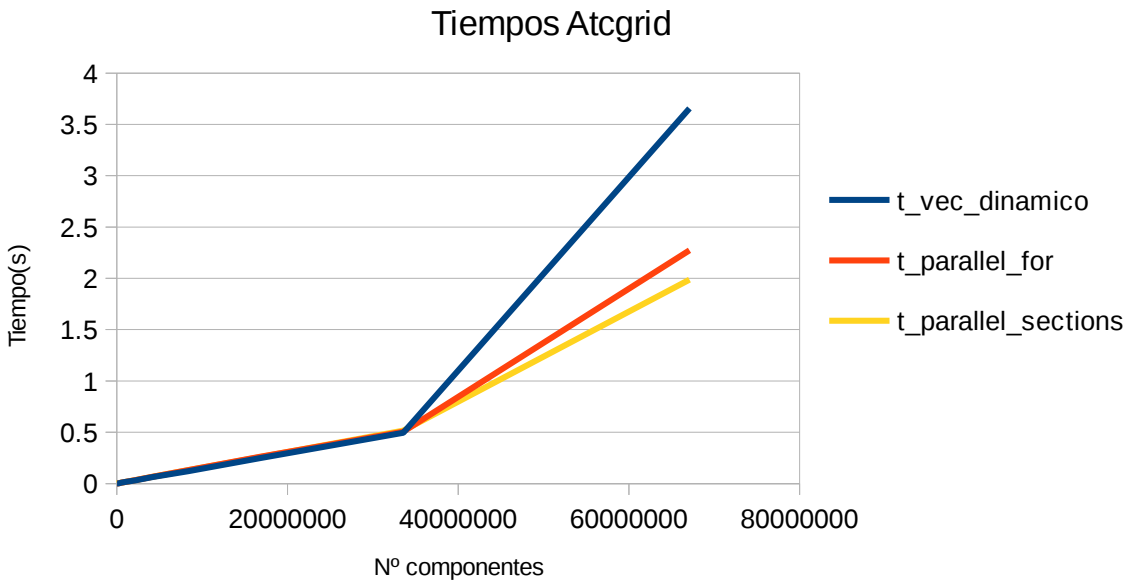
Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados, que debe coincidir con el número de cores físicos utilizados.

Nº de Componentes	T. secuencial vec. Dinamicos 1 thread/core	T. paralelo (versión `for`) ¿?threads/cores	T. paralelo (versión `sections`) ¿?threads/cores
16384	0.000036906	0.000726474	0.003412577
32768	0.000080682	0.001253049	0.003507248
65536	0.000143133	0.000036557	0.003534916
131072	0.000331660	0.000091267	0.002980733
262144	0.001048510	0.003574092	0.001826291
524288	0.001733763	0.005051306	0.002648893
1048576	0.005043845	0.006848639	0.006111208
2097152	0.006670403	0.009339732	0.010355434
4194304	0.015092629	0.020839788	0.022191192
8388608	0.029710133	0.041504727	0.042188799
16777216	0.059323836	0.079377017	0.078795664
33554432	0.133216473	0.157664800	0.167763925
67108864	0.235362682	0.324384315	0.356522307



ATCGRID

Nº de Componentes	T. secuencial vec. Dinamicos 1 thread/core	T. paralelo (versión `for`) ¿?threads/cores	T. paralelo (versión `sections`) ¿?threads/cores
16384	0.000242212	0.000210645	0.000195557
32768	0.000723843	0.000615450	0.000562649
65536	0.001200724	0.001140103	0.001146806
131072	0.002187173	0.002149180	0.002211759
262144	0.004089670	0.004198903	0.004173201
524288	0.007490977	0.008296397	0.008263150
1048576	0.015902193	0.016486071	0.016386058
2097152	0.030741390	0.032921299	0.033111550
4194304	0.061933680	0.065835616	0.065212065
8388608	0.122100358	0.129882995	0.131670672
16777216	0.247079605	0.261696704	0.257492771
33554432	0.494411478	0.505471384	0.516335452
67108864	3.656314280	2.272380412	1.986127527



. 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` ¿? Threads/cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	0m0.001s	0m0.000s	0m0.000s	0m0.012s	0m0.044s	0m0.000s
131072	0m0.002s	0m0.000s	0m0.000s	0m0.012s	0m0.048s	0m0.000s
262144	0m0.004s	0m0.000s	0m0.000s	0m0.008s	0m0.032s	0m0.000s
524288	0m0.006s	0m0.000s	0m0.004s	0m0.011s	0m0.052s	0m0.004s
1048576	0m0.011s	0m0.004s	0m0.004s	0m0.019s	0m0.072s	0m0.024s
2097152	0m0.028s	0m0.020s	0m0.004s	0m0.027s	0m0.128s	0m0.040s
4194304	0m0.032s	0m0.024s	0m0.004s	0m0.051s	0m0.224s	0m0.064s
8388608	0m0.055s	0m0.044s	0m0.008s	0m0.101s	0m0.408s	0m0.108s
16777216	0m0.105s	0m0.068s	0m0.036s	0m0.190s	0m0.756s	0m0.300s
33554432	0m0.248s	0m0.172s	0m0.076s	0m0.379s	0m1.448s	0m0.652s
67108864	0m0.252s	0m0.176s	0m0.072s	0m0.763s	0m2.948s	0m1.104s

En el caso del Ejercicio Listado1 el tiempo Real siempre supera a los tiempos de CPU excepto en el caso del n.º de componentes = 33554432 donde los tiempos son iguales.

Por otro lado, en el caso de ejecución del ejercicio 7, los tiempos de CPU-user siempre superan al tiempo Real, mientras que los tiempos de CPU-sys comienzan a superar al tiempo real a partir de n.º componentes = 1048576.