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): Ernesto Serrano Collado
Grupo de prácticas: A2
Fecha de entrega: 07/04/2016
Fecha evaluación en clase: 08/04/2016

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>

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

main()
{
        #pragma omp parallel sections
        {
```

```
            #pragma omp section
                    (void) funcA();

                    #pragma omp section
                    (void) funcB();
            }
}
```

2. Imprimir los resultados del programa `single.c` usando una directiva `single` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `single` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `single`. Incorpore en su cuaderno de trabajo el código fuente y volcados de pantalla con los resultados de ejecución obtenidos.

   RESPUESTA: código fuente `singleModificado.c`

```
#include <stdio.h>

#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 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:

```
bender:practica1 ernesto$ bin/singleModificado
Introduce valor de inicialización a: 100
Single ejecutada por el thread 0
Resultados:
b[0] = 100      b[1] = 100      b[2] = 100      b[3] = 100      b[4] = 100      b
[5] = 100       b[6] = 100      b[7] = 100      b[8] = 100
Single ejecutada por el thread 3
bender:practica1 ernesto$
```

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>

#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:

```
bender:practica1 ernesto$ bin/singleModificado2
Introduce valor de inicialización a: 100
Single ejecutada por el thread 1
Resultados:
b[0] = 100      b[1] = 100      b[2] = 100      b[3] = 100      b[4] = 100      b
[5] = 100       b[6] = 100      b[7] = 100      b[8] = 100
Single ejecutada por el thread 0
bender:practica1 ernesto$
```

RESPUESTA A LA PREGUNTA: Al utilizar la directiva master los resultados siempre los pintará el thread 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 si el thread 0 se ejecutara mas rápido pintaría los resultados sin haber esperado a que terminen los otros hilos.

Resto de ejercicios

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 dinamicos**. 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.

   RESPUESTA: La suma de los tiempos de usuario y sistema es igual al tiempo real porque se está usando un solo núcleo del procesador.

   CAPTURAS DE PANTALLA:

```
[[ernesto@ubuntuserver 13:33:35 practica1]$ time bin/SumaVectores 10000000
Tiempo(seg.):0.154536434         Tamaño Vectores:10000000        V1[0]+V2[0]=V3[
0](1000000.000000+1000000.000000=2000000.000000) V1[9999999]+V2[9999999]=V3[9999
999](1999999.900000+0.100000=2000000.000000)

real    0m0.354s
user    0m0.195s
sys     0m0.158s
```

6. Generar el código ensamblador a partir del programa secuencial C del Listado 1 para **vectores dinamicos** (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:

```
[[A2estudiante29@atcgrid 13:48:08 ~]$ echo './SumaVectores 10' | qsub -q ac
31793.atcgrid
[[A2estudiante29@atcgrid 13:50:46 ~]$ cat STDIN.o31793
Tiempo(seg.):0.000000160         Tamaño Vectores:10      V1[0]+V2[0]=V3[0](1.000000+1.000000=2.0
+0.100000=2.000000)
[[A2estudiante29@atcgrid 13:51:15 ~]$ echo './SumaVectores 10000000' | qsub -q ac
31794.atcgrid
[[A2estudiante29@atcgrid 13:51:45 ~]$ cat STDIN.o31794
Tiempo(seg.):0.048853831         Tamaño Vectores:10000000        V1[0]+V2[0]=V3[0](1000000.00000
V1[9999999]+V2[9999999]=V3[9999999](1999999.900000+0.100000=2000000.000000)
```

   RESPUESTA: cálculo de los MIPS y los MFLOPS

Tamaño vector = 10: NI: 63 (6*10+3) FPO: 30 (3*10) Tiempo(seg.): 0.000000160 MIPS: 63 / (0.000000160*10^6) =393.75 MFLOPS: 30 / (0.000000160*10^6) =187.5	Tamaño vector = 10000000: NI: 60000003 (6*10000000 +3) FPO: 30000000 (3*10000000) Tiempo(seg.):0.048853831 MIPS: 60000003 / (0.048853831*10^6) = 1228.15348913 MFLOPS: 30000000 / (0.048853831*10^6) = 614.076713861

   RESPUESTA:

código ensamblador generado de la parte de la suma de vectores

```
        xorl    %eax, %eax
        .p2align 4,,10
        .p2align 3
.L9:
        movsd   0(%rbp,%rax,8), %xmm0
        addsd   (%rbx,%rax,8), %xmm0
        movsd   %xmm0, 0(%r13,%rax,8)
        addq    $1, %rax
        cmpl    %eax, %r12d
        ja      .L9
.L10:
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
```

7. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores (v3 = v1 + v2; v3(i)=v1(i)+v2(i), i=0,…N-1) usando las directivas `parallel` y `for`. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Como en el código del Listado 1 se debe obtener el tiempo (*elapsed time*) que supone el cálculo de la suma. Para obtener este tiempo usar la función `omp_get_wtime()`, que proporciona el estándar OpenMP, en lugar de `clock_gettime()`. NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para varios tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de v1, v2 y v3 (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

RESPUESTA: código fuente implementado

```
#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 (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)

    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):

```
[ernesto@ubuntuserver 16:21:11 practica1]$ bin/SumaVectoresE7 8
Tiempo(seg.):0.000010293        Tamaño Vectores:8
 V1[0]+V2[0]=V3[0]      (0.800000+0.800000=1.600000)
 V1[7]+V2[7]=V3[7]      (1.500000+0.100000=1.600000)
[ernesto@ubuntuserver 16:21:19 practica1]$ bin/SumaVectoresE7 11
Tiempo(seg.):0.000011274        Tamaño Vectores:11
 V1[0]+V2[0]=V3[0]      (1.100000+1.100000=2.200000)
 V1[10]+V2[10]=V3[10]   (2.100000+0.100000=2.200000)
```

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

```
#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 (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)

        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):

```
[[ernesto@ubuntuserver 16:30:47 practica1]$ bin/SumaVectoresE8 8
Tiempo(seg.):0.000010784
 Tamaño Vectores:8       V1[0]+V2[0]=V3[0]       (0.800000+0.800000=1.600000)
 V1[7]+V2[7]=V3[7]      (1.500000+0.100000=1.600000)
[[ernesto@ubuntuserver 16:32:10 practica1]$ bin/SumaVectoresE8 11
Tiempo(seg.):0.000011763
 Tamaño Vectores:11      V1[0]+V2[0]=V3[0]       (1.100000+1.100000=2.200000)
 V1[10]+V2[10]=V3[10]   (2.100000+0.100000=2.200000)
```

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.

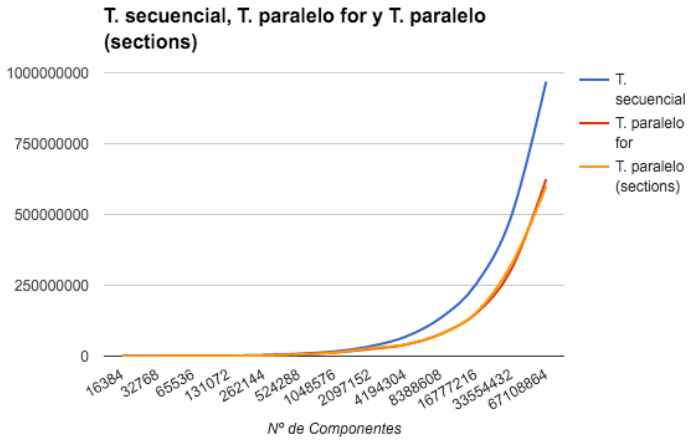
RESPUESTA: En ambos casos como no hemos definido la variable de entorno OMP_NUM_THREADS se usarán todos los cores/threads que tenga disponibles la máquina, aqunque en el ejercicio 8 al definir las secciones de forma fija habrán threads que no hagan nada.

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. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

RESPUESTA:

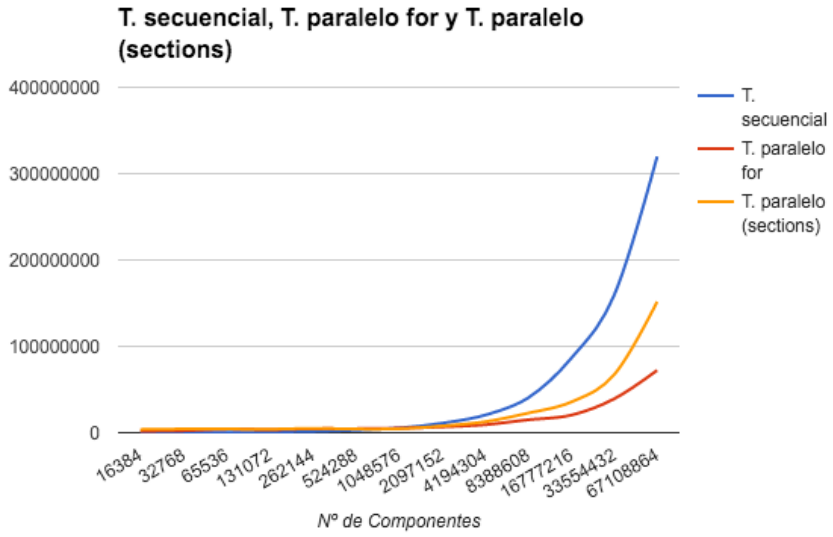
PC LOCAL (`AMD Athlon(tm) X2 Dual-Core QL-60`)

Nº de Componentes	T. secuencial vect. dinamicos 1 thread/core	T. paralelo (versión `for`) 2 threads/ 2 cores	T. paralelo (versión `sections`) 2 threads/ 2 cores
16384	0.000364175	0.000198997	0.000262716
32768	0.000871961	0.000474947	0.000448478
65536	0.001726275	0.000987143	0.001027825
131072	0.002196320	0.001551295	0.001477284
262144	0.003822117	0.002834973	0.002779588
524288	0.008433360	0.006488972	0.006242921
1048576	0.016577535	0.012340760	0.012581419
2097152	0.034020660	0.024855030	0.027722353
4194304	0.067463934	0.039780302	0.040645891
8388608	0.133610860	0.077351051	0.076422234
16777216	0.251523007	0.148771399	0.150854987
33554432	0.488708420	0.302782403	0.320336789
67108864	0.968969758	0.624633235	0.598744988



ATCGRID

Nº de Componentes	T. secuencial vect. dinamicos 1 thread/core	T. paralelo (versión `for`) 24 threads/ 12 cores	T. paralelo (versión `sections`) 24 threads/ 12 cores
16384	0.000093374	0.001846500	0.003699739
32768	0.000175027	0.002395201	0.004251290
65536	0.000397032	0.004168388	0.004264597
131072	0.000796611	0.003509805	0.004020311
262144	0.001444324	0.004544973	0.004937742
524288	0.003044522	0.004585367	0.004211366
1048576	0.005346107	0.005140271	0.004511990
2097152	0.010889973	0.006662808	0.007708970
4194304	0.020307471	0.009173129	0.012616109
8388608	0.040151177	0.014905810	0.022641912
16777216	0.085907184	0.020254489	0.035415940
33554432	0.159008779	0.039140988	0.067056276
67108864	0.319819459	0.072207924	0.151632909



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 usado en el programa secuencial es igual al tiempo real, ya que sólo se usa un procesador. En la versión paralela, sin embargo, el tiempo de CPU es mayor que el tiempo real. Esto es debido a que el tiempo de CPU es igual a la suma de los tiempos de cada núcleo, mientras que el tiempo real es el tiempo que realmente ha tardado el programa en ejecutarse.

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. dinamicos 1 thread/core**			**Tiempo paralelo/versión `for` 2 Threads/ 2 cores**		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	real 0m0.009s	user 0m0.000s	sys 0m0.010s	real 0m0.007s	user 0m0.007s	sys 0m0.003s
131072	real 0m0.010s	user 0m0.005s	sys 0m0.005s	real 0m0.009s	user 0m0.007s	sys 0m0.006s
262144	real 0m0.014s	user 0m0.005s	sys 0m0.009s	real 0m0.011s	user 0m0.013s	sys 0m0.007s
524288	real 0m0.026s	user 0m0.013s	sys 0m0.013s	real 0m0.020s	user 0m0.013s	sys 0m0.021s
1048576	real 0m0.045s	user 0m0.020s	sys 0m0.023s	real 0m0.032s	user 0m0.028s	sys 0m0.032s
2097152	real 0m0.081s	user 0m0.065s	sys 0m0.016s	real 0m0.061s	user 0m0.067s	sys 0m0.050s
4194304	real 0m0.152s	user 0m0.044s	sys 0m0.107s	real 0m0.111s	user 0m0.122s	sys 0m0.090s
8388608	real 0m0.292s	user 0m0.168s	sys 0m0.123s	real 0m0.197s	user 0m0.226s	sys 0m0.162s
16777216	real 0m0.567s	user 0m0.355s	sys 0m0.211s	real 0m0.376s	user 0m0.458s	sys 0m0.282s
33554432	real 0m1.109s	user 0m0.664s	sys 0m0.437s	real 0m0.702s	user 0m0.781s	sys 0m0.605s
67108864	real 0m2.167s	user 0m1.218s	sys 0m0.942s	real 0m1.394s	user 0m1.572s	sys 0m1.181s