

2º curso / 2º cuatr.
Grado Ing. Inform.
Doble Grado Ing.
Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 1. Programación paralela I: Directivas OpenMP

Estudiante (nombre y apellidos): Francisco Javier Bolívar Lupiáñez

Grupo de prácticas: B1

Fecha de entrega: 01/04/2014

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());
}

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>
#include <omp.h>

main()
{
    int n = 9, i, a, b[n];

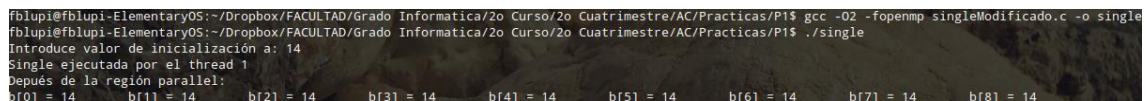
    for (i=0; i<n; i++)
        b[i] = -1;
    #pragma omp parallel
    {
        #pragma omp single
        {
            printf("Introduce valor de inicialización a: ");
            scanf("%d", &a );
            printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
        }

        #pragma omp for
        for (i=0; i<n; i++)
            b[i] = a;

        #pragma omp single
        {
            for (i=0; i<n; i++)
                printf("b[%d] = %d\t", i, b[i]);
            printf("\n");
        }
    }
}

```

CAPTURAS DE PANTALLA:



```

fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P1$ gcc -O2 -fopenmp singleModificado.c -o single
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P1$ ./single
Introduce valor de inicialización a: 14
Single ejecutada por el thread 1
Depués de la región parallel:
b[0] = 14    b[1] = 14    b[2] = 14    b[3] = 14    b[4] = 14    b[5] = 14    b[6] = 14    b[7] = 14    b[8] = 14

```

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 masterModificado2.c

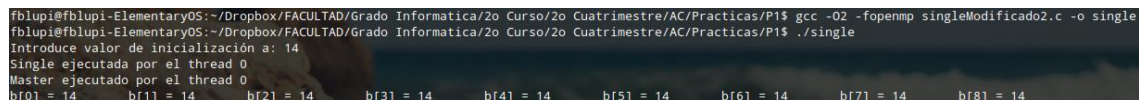
```
#include <stdio.h>
#include <omp.h>

main()
{
    int n = 9, i, a, b[n];

    for (i=0; i<n; i++)
        b[i] = -1;
    #pragma omp parallel
    {
        #pragma omp single
        {
            printf("Introduce valor de inicialización a: ");
            scanf("%d", &a );
            printf("Single ejecutada por el thread %d\n", omp_get_thread_num());
        }

        #pragma omp for
        for (i=0; i<n; i++)
            b[i] = a;

        #pragma omp barrier
        #pragma omp master
        {
            printf("Master ejecutado 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:


```
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P1$ gcc -O2 -fopenmp singleModificado2.c -o single
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P1$ ./single
Introduce valor de inicialización a: 14
Single ejecutada por el thread 0
Master ejecutado por el thread 0
b[0] = 14    b[1] = 14    b[2] = 14    b[3] = 14    b[4] = 14    b[5] = 14    b[6] = 14    b[7] = 14    b[8] = 14
```

RESPUESTA A LA PREGUNTA: El `printf` lo ejecutará siempre la hebra maestra, (hebra 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 no habría una barrera que esperase a que se sumasen todas las sumas locales antes de hacer el `printf`.

Resto de ejercicios

En clase presencial y en casa, usando plataformas (procesadores + SO) de 64 bits, se trabajarán los siguientes ejercicios y cuestiones:

5. El programa secuencial C del Listado 1 calcula la suma de dos vectores ($v3 = v1 + v2$; $v3(i) = v1(i) + v2(i)$, $i=0, \dots, N-1$). Generar el ejecutable del programa del Listado 1 para **vectores globales**. Usar `time` (Lección 3/ Tema 1) en la línea de comandos para obtener el tiempo de ejecución (*elapsed time*) y el tiempo de CPU del usuario y del sistema para el ejecutable en `atcgri` y en el PC local. Obtenga los tiempos para vectores con 10000000 componentes.

CAPTURAS DE PANTALLA:

```
fbilupi@fbilupi-ElementaryOS:~/Dropbox/FACULTAD/Grado_Informatica/2o_Curso/2o_Cuatrimestre/AC/Practicas/P1$ time ./sumaVectoresGlobal 10000000
Tiempo(seg.):0.065373986 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) / / V1[999999]+V2[999999]=V3[999999](1999999.900000+0.100000=2000000.000000) /
real    0m0.249s
user    0m0.112s
sys     0m0.136s

[B1estudiante3@atcgrid hello]$ echo 'time hello/SumaVectoresGlobal 10000000' | qsub -q ac
30313.atcgrid
[B1estudiante3@atcgrid hello]$ cat STDIN.e30313

real    0m0.122s
user    0m0.073s
sys     0m0.044s
```

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:

```
[B1estudiante3@atcgrid hello]$ echo 'hello/SumaVectoresGlobal 10' | qsub -q ac
30433.atcgrid
[B1estudiante3@atcgrid hello]$ echo 'hello/SumaVectoresGlobal 10000000' | qsub -q ac
30434.atcgrid
[B1estudiante3@atcgrid hello]$ cat STDIN.o30433
Tiempo(seg.):0.000403428 / Tamaño Vectores:10 / V1[0]+V2[0]=V3[0](1.000000+1.000000=2.000000) / / V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /
[B1estudiante3@atcgrid hello]$ cat STDIN.o30434
Tiempo(seg.):0.040160941 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) / / V1[999999]+V2[999999]=V3[999999](1999999.900000+0.100000=2000000.000000) /

[B1estudiante3@atcgrid hello]$ echo 'hello/SumaVectoresGlobal 1000' | qsub -q ac
31588.atcgrid
[B1estudiante3@atcgrid hello]$ echo 'hello/SumaVectoresGlobal 1000000' | qsub -q ac
31589.atcgrid
[B1estudiante3@atcgrid hello]$ cat STDIN.o31588
Tiempo(seg.):0.000405310 / Tamaño Vectores:1000 / V1[0]+V2[0]=V3[0](100.000000+100.000000=200.000000) / / V1[999]+V2[999]=V3[999](199.900000+0.100000=200.000000) /
[B1estudiante3@atcgrid hello]$ cat STDIN.o31589
Tiempo(seg.):0.000623282 / Tamaño Vectores:100000 / V1[0]+V2[0]=V3[0](10000.000000+10000.000000=20000.000000) / / V1[99999]+V2[99999]=V3[99999](19999.900000+0.100000=20000.000000) /
```

RESPUESTA: Lógicamente, con 10 no se aprovecha todo el paralelismo que nos ofrece ATCGRID, además, el gasto que consumen las operaciones de `clock_gettime` también influyen y hacen que se vea tanta diferencia en algo que no debería tener tanta. Por eso probé con más valores: 1000 y 1000000.

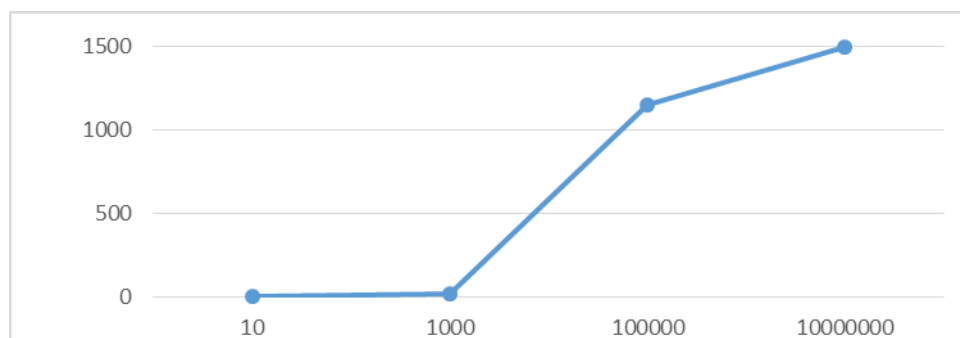
MIPS

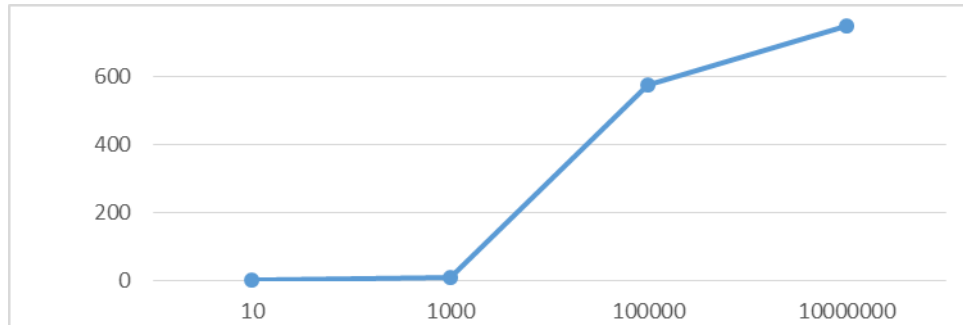
$MIPS_{10}=0.16607672248828$

$MIPS_{1000}=14.82075448422195$

$MIPS_{100000}=1146.62266235032$

$MIPS_{10000000}=1493.9890726166$



MFLOPSMFLPOS₁₀=0.074362711561914MFLPOS₁₀₀₀=7.40174187658829MFLOPS₁₀₀₀₀₀=573.3046426209959MFLOPS₁₀₀₀₀₀₀₀=746.9944491589478**RESPUESTA:** código ensamblador generado de la parte de la suma de vectores

```

call    clock_gettime
xorl    %eax, %eax
.p2align 4,,10
.p2align 3
.L7:
movsd   v1(%rax), %xmm0
addsd   v2(%rax), %xmm0
movsd   %xmm0, v3(%rax)
addq    $8, %rax
cmpq    %rbp, %rax
jne     .L7
.L8:
leaq    16(%rsp), %rsi
xorl    %edi, %edi
call    clock_gettime

```

7. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores ($v3 = v1 + v2$; $v3(i) = v1(i) + v2(i)$, $i=0, \dots, N-1$) 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

```

// gcc -O2 -fopenmp sumaVectoresFor.c -o sumaVectoresFor

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

// #define PRINTF_ALL // Comentar para quitar el print...

```

```

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

    unsigned int N = atoi(argv[1]);

    double *v1, *v2, *v3;
    v1 = (double*) malloc(N*sizeof(double));
    v2 = (double*) malloc(N*sizeof(double));
    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);
    }

    int i;

    double cgt1, cgt2, ncgt;

    if (argc<2){
        printf("Faltan no componentes del vector\n");
        exit(-1);
    }

    // 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;
        //printf("La hebra %d ejecuta la iteracion %d del primer
bucle\n",omp_get_thread_num(),i);
    }
    cgt1=omp_get_wtime();

    // Calcular suma de vectores
    #pragma omp parallel for
    for(i=0; i<N; i++) {
        v3[i] = v1[i] + v2[i];
        //printf("La hebra %d ejecuta la iteracion %d del segundo bucle\n",
omp_get_thread_num(),i);
    }
    cgt2=omp_get_wtime();
    ncgt=(cgt2-cgt1);

    #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\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

    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;
}

```

CAPTURAS DE PANTALLA:

```
fblupie@fblupie-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P1$ ./sumaVectores 11
Tiempo(seg.):0.000026505 / 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) /
fblupie@fblupie-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P1$ ./sumaVectores 8
Tiempo(seg.):0.000021533 / 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) /
```

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

```
// gcc -O2 -fopenmp sumaVectoresSection.c -o sumaVectoresSection

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

#define SECTIONS 4
// #define PRINTF_ALL // Comentar para quitar el print...

void f(double *v1, double *v2, double *v3, int act, int N) {
    int i;
    for(i=act; i<N; i=i+SECTIONS) {
        v3[i] = v1[i] + v2[i];
        //printf("La hebra %d ejecuta la iteracion %d del segundo
bucle\n",omp_get_thread_num(),i);
    }
}

void iniciaV1(double *v1, int N) {
    int i;
    for(i=0; i<N; i++) {
        v1[i] = N*0.1+i*0.1;
        //printf("La hebra %d ejecuta la iteracion %d del primer bucle
V1\n",omp_get_thread_num(),i);
    }
}

void iniciaV2(double *v2, int N) {
    int i;
    for(i=0; i<N; i++) {
        v2[i] = N*0.1-i*0.1;
        //printf("La hebra %d ejecuta la iteracion %d del primer bucle
V2\n",omp_get_thread_num(),i);
    }
}

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

    unsigned int N = atoi(argv[1]);

    double *v1, *v2, *v3;
```

```

v1 = (double*) malloc(N*sizeof(double));
v2 = (double*) malloc(N*sizeof(double));
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);
}

int i;

double cgt1, cgt2, ncgt;

if (argc<2){
    printf("Faltan no componentes del vector\n");
    exit(-1);
}

// Inicializar vectores
#pragma omp parallel sections
{
    #pragma omp section
        (void)iniciaV1(v1,N);
    #pragma omp section
        (void)iniciaV2(v2,N);
}
cgt1=omp_get_wtime();

// Calcular suma de vectores
#pragma omp parallel sections
{
    #pragma omp section
        (void)f(v1,v2,v3,0,N);
    #pragma omp section
        (void)f(v1,v2,v3,1,N);
    #pragma omp section
        (void)f(v1,v2,v3,2,N);
    #pragma omp section
        (void)f(v1,v2,v3,3,N);
}
cgt2=omp_get_wtime();
ncgt=(cgt2-cgt1);

#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\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

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;
}

```

CAPTURAS DE PANTALLA:


```
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P1$ ./sumaVectoresSection 8
Tiempo(seg.):0.000292529 / Tamaño Vectores:8 / V1[0]+V2[0]=V3[0](0.800000+0.800000=0.000000) / V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P1$ ./sumaVectoresSection 11
Tiempo(seg.):0.000351587 / Tamaño Vectores:11 / V1[0]+V2[0]=V3[0](1.100000+1.100000=0.000000) / 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 cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta.

RESPUESTA: No existe un número fijo de threads y cores en ninguno de los casos. En el caso del ejercicio 7, dependería del número de iteraciones que hiciese el bucle. Sería una tontería usar más threads que este número de iteraciones o que el número de cores totales. En el caso del ejercicio 8, nosotros fijamos las sections, y usar más threads que sections no tendría mucho sentido. Lo ideal sería usar tantas threads como sections.

10. Rellenar una tabla como la 0 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 (*elapsed time*) 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.

Mi PC			
Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 4 threads/cores	T. paralelo (versión sections) 4 threads/cores
65536	0.000691984	0.009293241	0.009086779
131072	0.001776454	0.004292056	0.003678596
262144	0.003026093	0.000972495	0.002141481
524288	0.006195593	0.010737913	0.004341564
1048576	0.007034391	0.013772339	0.019668285
2097152	0.014201907	0.012702962	0.037282649
4194304	0.027511037	0.017678589	0.052121586
8388608	0.054142262	0.037345468	0.066386879
16777216	0.109533503	0.065587392	0.163916816
33554432	0.222417465	0.106485788	0.320137855
67108864	0.439385165	0.201262258	0.558560499

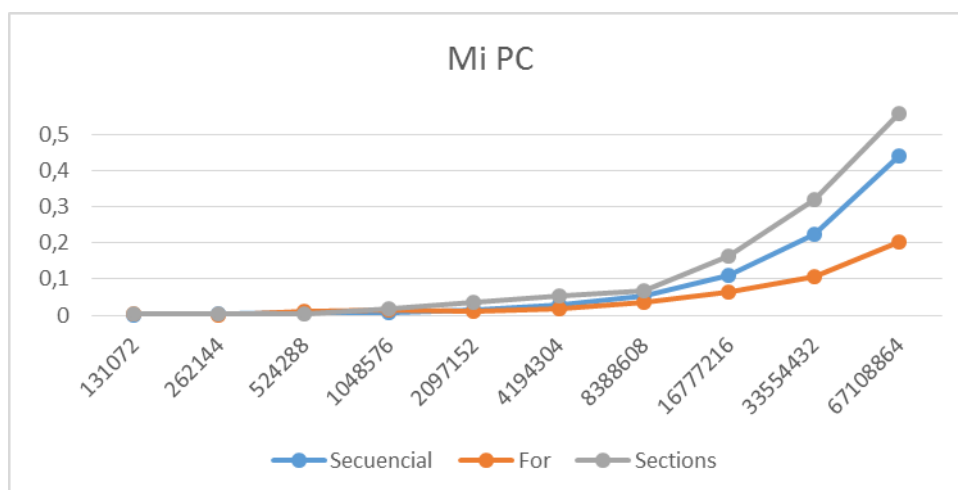
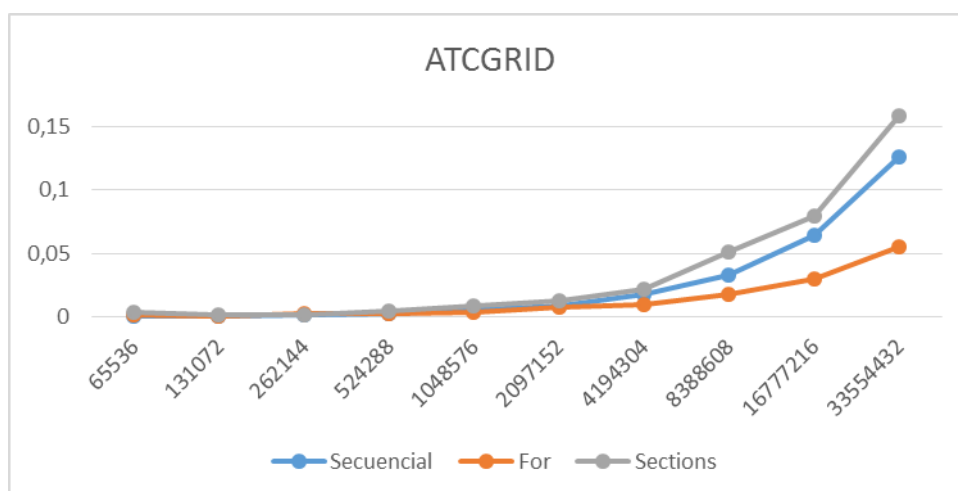


Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas.

ATCGRID

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 24 threads/cores	T. paralelo (versión sections) 24 threads/cores
65536	0.000556825	0.001717393	0.003851940
131072	0.000493246	0.000143548	0.001068264
262144	0.001433038	0.002859207	0.001785220
524288	0.002679134	0.002479588	0.004303319
1048576	0.005131432	0.003941232	0.008441180
2097152	0.008972716	0.007089039	0.012741302
4194304	0.017244894	0.009330989	0.021440588
8388608	0.032592473	0.017260491	0.050876519
16777216	0.064401751	0.029466876	0.080059414
33554432	0.126639787	0.055516305	0.159002917
67108864	0.250765808	0.108251665	0.302757850



11. Rellenar una tabla como la 0 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.

RESPUESTA:

Tabla 3. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas.

Nº de Componentes	Tiempo secuencial vect. Globales 1 thread/core			Tiempo paralelo/versión <code>for</code> 4 Threads/cores		
	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>
65536	0m0.005s	0m0.004s	0m0.004s	0m0.015s	0m0.040s	0m0.004s
131072	0m0.008s	0m0.004s	0m0.000s	0m0.011s	0m0.016s	0m0.004s
262144	0m0.013s	0m0.008s	0m0.004s	0m0.016s	0m0.028s	0m0.008s
524288	0m0.015s	0m0.004s	0m0.008s	0m0.011s	0m0.016s	0m0.016s
1048576	0m0.035s	0m0.012s	0m0.020s	0m0.035s	0m0.068s	0m0.020s
2097152	0m0.060s	0m0.028s	0m0.028s	0m0.037s	0m0.084s	0m0.032s
4194304	0m0.117s	0m0.048s	0m0.068s	0m0.057s	0m0.116s	0m0.068s
8388608	0m0.213s	0m0.096s	0m0.116s	0m0.105s	0m0.152s	0m0.176s
16777216	0m0.409s	0m0.200s	0m0.200s	0m0.193s	0m0.272s	0m0.348s
33554432	0m0.870s	0m0.348s	0m0.508s	0m0.380s	0m0.544s	0m0.676s
67108864	0m1.668s	0m0.772s	0m0.880s	0m0.775s	0m1.056s	0m1.292s