

2° curso / 2° 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): Rodrigo Raya Castellano

Grupo de prácticas: 2

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\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\n", omp_get_thread_num());
}
void funcB() {
    printf("En funcB: esta sección la ejecuta el thread\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 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("Single que muestra resultados ejecutada por el thread %d\n",
                omp_get_thread_num());
            for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
            printf("\n");
        }
    }
}
```

CAPTURAS DE PANTALLA:

```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/2.2/AC/practicas/practica1$ ./singleModificado
Introduce valor de inicializacion a: 7
Single ejecutada por el thread 0
Single que muestra resultados ejecutada por el thread 2
b[0] = 7      b[1] = 7      b[2] = 7      b[3] = 7      b[4] = 7      b[5] = 7      b[6] = 7      b[7] = 7      b[8] = 7
```

3. Imprimir los resultados del programa `single.c` usando una directiva `master` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `master` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `master`. Incorpore en su cuaderno el código fuente y volcados de pantalla con los resultados de ejecución obtenidos. ¿Qué diferencia observa con respecto a los resultados de ejecución del ejercicio anterior?

RESPUESTA: código fuente `singleModificado2.c`

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

```

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 master
    {
        printf("Single que muestra resultados ejecutada por el thread %d\n",
            omp_get_thread_num());
        for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
        printf("\n");
    }
}
}

```

CAPTURAS DE PANTALLA:

```

rjraya@rjraya-SATELLITE-C55-A-1EL:~/2.2/AC/practicas/practica1$ ./singleModificado
Introduce valor de inicializacion a: 8
Single ejecutada por el thread 0
Single que muestra resultados ejecutada por el thread 0
b[0] = 8    b[1] = 8    b[2] = 8    b[3] = 8    b[4] = 8    b[5] = 8    b[6] = 8    b[7] = 8    b[8] = 8
rjraya@rjraya-SATELLITE-C55-A-1EL:~/2.2/AC/practicas/practica1$ ./singleModificado
Introduce valor de inicializacion a: 9
Single ejecutada por el thread 2
Single que muestra resultados ejecutada por el thread 0
b[0] = 9    b[1] = 9    b[2] = 9    b[3] = 9    b[4] = 9    b[5] = 9    b[6] = 9    b[7] = 9    b[8] = 9
rjraya@rjraya-SATELLITE-C55-A-1EL:~/2.2/AC/practicas/practica1$ ./singleModificado
Introduce valor de inicializacion a: 1
Single ejecutada por el thread 2
Single que muestra resultados ejecutada por el thread 0
b[0] = 1    b[1] = 1    b[2] = 1    b[3] = 1    b[4] = 1    b[5] = 1    b[6] = 1    b[7] = 1    b[8] = 1
rjraya@rjraya-SATELLITE-C55-A-1EL:~/2.2/AC/practicas/practica1$

```

RESPUESTA A LA PREGUNTA:

Cuando se ejecutaba la versión con single el thread encargado de imprimir los resultados podía variar de una ejecución a otra. Sin embargo en este caso el comportamiento es determinista, siempre ejecuta este código la 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:

Cuando las hebras actualizan la suma no se esperan entre ellas (no hay sincronización), en particular, la hebra cero, que hace de master, no espera a las demás con lo que no imprime el último valor de suma.

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, en el PC local, el tiempo de ejecución (*elapsed time*) y el tiempo de CPU del usuario y del sistema para el ejecutable generado. Obtenga los tiempos para vectores con 10000000 componentes. ¿La suma de los tiempos de CPU del usuario y del sistema es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

RESPUESTA:

Siguiendo a Chapman se observa que el tiempo real es ligeramente mayor que el tiempo de CPU. De entre las muchas razones por las que puede ocurrir esto, hay una común: la aplicación no obtuvo un procesador completamente para él porque había cierta carga en el sistema.

En ocasiones con paralelismo puede ocurrir que el tiempo real sea menor que el tiempo de CPU, si bien puede haber sobrecargas adicionales. Esto lo veremos en ejercicios posteriores.

CAPTURAS DE PANTALLA:

```
rjraya@eil40091:~/Escritorio/Home/AC/pl$ time ./listado1 10000000
Tiempo(seg.):0.024954261 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000)v1[9999999]+v2[9999999]=v3[9999999](1999999.900000+0.100000=2000000.000000) /

real    0m0.100s
user    0m0.054s
sys     0m0.044s
rjraya@eil40091:~/Escritorio/Home/AC/pl$
```

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:

```
[E2estudiante9@atcgrid l3]$ cat STDIN.o9617
Tiempo(seg.):0.000003283 / 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) /
[E2estudiante9@atcgrid l3]$ echo 'l3/listado1 10000000' | qsub -q ac
9618.atcgrid
[E2estudiante9@atcgrid l3]$ cat STDIN.o9618
Tiempo(seg.):0.046845963 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000)v1[9999999]+v2[9999999]=v3[9999999](1999999.900000+0.100000=2000000.000000) /
[E2estudiante9@atcgrid l3]$
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Tamaño (N)	10	10000000
Tiempo	0,000003283	0,046845963
MIPS	20,41	1280,79
MFLOPS	3,045994517	213,465565859

No es inmediata una explicación de por qué los MIPS difieren tanto de un tamaño a otro. Pero podemos suponer un comportamiento lineal tipo $A+B*N$ para el número de instrucciones a realizar y otro comportamiento lineal para el tiempo de ejecución $A'+B'*N$. Entonces, para ejecuciones de pocos elementos el número de MIPS sería similar al cociente A/A' y para ejecuciones con muchos elementos sería similar al cociente B/B' .

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

```

call    clock_gettime
    leal    -1(%rbx), %eax
    leaq    8(,%rax,8), %rdx
    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    %rdx, %rax
    jne     .L7
.L8:
    movq    %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 ($v3 = v1 + v2$; $v3(i)=v1(i)+v2(i)$, $i=0,...N-1$) 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

```

/*SumaVectoresC.c
    Suma de dos vectores: v3 = v1 + v2
    Para compilar usar (-lrt: real time library):
    gcc -O2 SumaVectores.c -o SumaVectores -lrt
    Para ejecutar use: SumaVectoresC longitud
*/
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
#include <omp.h>

//#define PRINTF_ALL // comentar para quitar el printf ...
// que imprime todos los componentes

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

//#define VECTOR_LOCAL // descomentar para que los vectores sean variables ...
// locales (si se supera el tamaño de la pila se ...
// generará el error "Violación de Segmento")
#define VECTOR_GLOBAL // descomentar para que los vectores sean variables ...
// globales (su longitud no estará limitada por el ...
// tamaño de la pila del programa)
//#define VECTOR_DYNAMIC // descomentar para que los vectores sean
variables ...
// dinámicas (memoria reutilizable durante la ejecución)
#ifdef VECTOR_GLOBAL

#define MAX 33554432
    double v1[MAX], v2[MAX], v3[MAX];
#endif

int main(int argc, char** argv){
    int i;
    struct timespec; double ncgt,cgt1,cgt2; //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)

        //Segun el array que se haya elegido se ajusta el tamaño de una
        forma u otra
        #ifdef VECTOR_LOCAL
            double v1[N], v2[N], v3[N];
            // Tamaño variable local en tiempo de ejecución ...
            // disponible en C a partir de actualización C99
        #endif
        #ifdef VECTOR_GLOBAL
            if (N>MAX) N=MAX;
        #endif
        #ifdef VECTOR_DYNAMIC
            double *v1, *v2, *v3;
            v1 = (double*) malloc(N*sizeof(double)); // malloc
necesita el tamaño en bytes
            v2 = (double*) malloc(N*sizeof(double)); //si no
hay espacio suficiente malloc devuelve NULL
            v3 = (double*) malloc(N*sizeof(double));
            if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
                printf("Error en la reserva de espacio
para los vectores\n");
                exit(-2);
            }
        #endif

        //Inicializar vectores
        #pragma omp parallel for
        for(i=0; i<N; i++){
            v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los
valores dependen de N
        }

        cgt1 = omp_get_wtime();
        //Calcular suma de vectores
        #pragma omp parallel for
        for(i=0; i<N; i++)
            v3[i] = v1[i] + v2[i];

        cgt2 = omp_get_wtime();
        ncgt=(cgt2-cgt1)+((cgt2-cgt1)/(1.e+9));
        //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\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

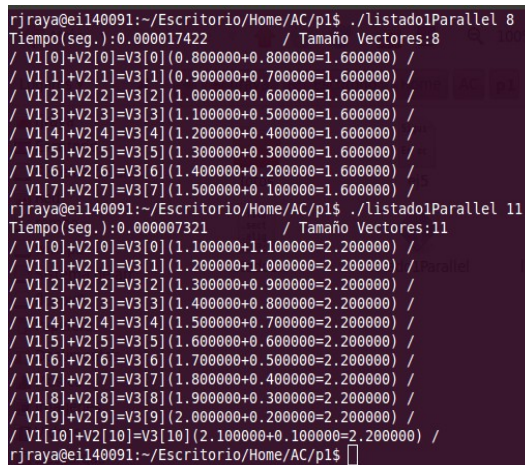
return 0;
}

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA:

Tamaños pequeños para comprobar que calcula el resultado correctamente:



```

rjraya@eil40091:~/Escritorio/Home/AC/pl$ ./listado1Parallel 8
Tiempo(seg.):0.000017422 / Tamaño Vectores:8
/ V1[0]+V2[0]=V3[0] (0.800000+0.800000=1.600000) /
/ V1[1]+V2[1]=V3[1] (0.900000+0.700000=1.600000) /
/ V1[2]+V2[2]=V3[2] (1.000000+0.600000=1.600000) /
/ V1[3]+V2[3]=V3[3] (1.100000+0.500000=1.600000) /
/ V1[4]+V2[4]=V3[4] (1.200000+0.400000=1.600000) /
/ V1[5]+V2[5]=V3[5] (1.300000+0.300000=1.600000) /
/ V1[6]+V2[6]=V3[6] (1.400000+0.200000=1.600000) /
/ V1[7]+V2[7]=V3[7] (1.500000+0.100000=1.600000) /
rjraya@eil40091:~/Escritorio/Home/AC/pl$ ./listado1Parallel 11
Tiempo(seg.):0.000007321 / Tamaño Vectores:11
/ V1[0]+V2[0]=V3[0] (1.100000+1.100000=2.200000) /
/ V1[1]+V2[1]=V3[1] (1.200000+1.000000=2.200000) /
/ V1[2]+V2[2]=V3[2] (1.300000+0.900000=2.200000) /
/ V1[3]+V2[3]=V3[3] (1.400000+0.800000=2.200000) /
/ V1[4]+V2[4]=V3[4] (1.500000+0.700000=2.200000) /
/ V1[5]+V2[5]=V3[5] (1.600000+0.600000=2.200000) /
/ V1[6]+V2[6]=V3[6] (1.700000+0.500000=2.200000) /
/ V1[7]+V2[7]=V3[7] (1.800000+0.400000=2.200000) /
/ V1[8]+V2[8]=V3[8] (1.900000+0.300000=2.200000) /
/ V1[9]+V2[9]=V3[9] (2.000000+0.200000=2.200000) /
/ V1[10]+V2[10]=V3[10] (2.100000+0.100000=2.200000) /
rjraya@eil40091:~/Escritorio/Home/AC/pl$

```

Tamaños mayores para medir tiempos:



```

rjraya@eil40091:~/Escritorio/Home/AC/pl$ gcc -O2 -fopenmp listado1Parallel.c -o listado1Parallel -lrt
rjraya@eil40091:~/Escritorio/Home/AC/pl$ time ./listado1Parallel 1000000
Tiempo(seg.):0.015726829 / Tamaño Vectores:1000000 / V1[0]+V2[0]=V3[0] (1000000.000000+1000000.000000=2000000.000000) v1[999999]+v2[999999]=v3[999999] (199
9999.900000+0.100000=2000000.000000) /

real    0m0.043s
user    0m0.083s
sys     0m0.054s
rjraya@eil40091:~/Escritorio/Home/AC/pl$

```


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 directivas `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, v_3 , 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 v_1 , v_2 y v_3 (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

RESPUESTA: código fuente implementado

```

/*SumaVectoresC.c
    Suma de dos vectores: v3 = v1 + v2
    Para compilar usar (-lrt: real time library):
    gcc -O2 SumaVectores.c -o SumaVectores -lrt
    Para ejecutar use: SumaVectoresC longitud
*/
#include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h> // biblioteca donde se encuentra la función clock_gettime()
#include <omp.h>

//#define PRINTF_ALL // comentar para quitar el printf ...
// que imprime todos los componentes

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

//#define VECTOR_LOCAL // descomentar para que los vectores sean variables ...
// locales (si se supera el tamaño de la pila se ...
// generará el error "Violación de Segmento")
#define VECTOR_GLOBAL // descomentar para que los vectores sean variables ...
// globales (su longitud no estará limitada por el ...
// tamaño de la pila del programa)
//#define VECTOR_DYNAMIC // descomentar para que los vectores sean
variables ...
// dinámicas (memoria reutilizable durante la ejecución)
#ifdef VECTOR_GLOBAL

#define MAX 33554432
    double v1[MAX], v2[MAX], v3[MAX];
#endif

int main(int argc, char** argv){
    int i;
    struct timespec; double ncgt,cgt1,cgt2; //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)

//Segun el array que se haya elegido se ajusta el tamaño de una
forma u otra
#ifdef VECTOR_LOCAL
    double v1[N], v2[N], v3[N];
    // Tamaño variable local en tiempo de ejecución ...
    // disponible en C a partir de actualización C99
#endif
#ifdef VECTOR_GLOBAL
    if (N>MAX) N=MAX;
#endif
#ifdef VECTOR_DYNAMIC
    double *v1, *v2, *v3;
    v1 = (double*) malloc(N*sizeof(double)); // malloc
necesita el tamaño en bytes
    v2 = (double*) malloc(N*sizeof(double)); //si no
hay espacio suficiente malloc devuelve NULL
    v3 = (double*) malloc(N*sizeof(double));
    if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
        printf("Error en la reserva de espacio
para los vectores\n");
        exit(-2);
    }
#endif

//Inicializar vectores
#pragma omp parallel for
for(i=0; i<N; i++){
    v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1; //los
valores dependen de N
}

cgt1 = omp_get_wtime();

//Calcular suma de vectores (versión continua)
#pragma omp parallel sections
{
    #pragma omp section
    for (i = 0; i < 1*(N/4); i++)
        v3[i] = v1[i] + v2[i];
    #pragma omp section
    for (i = 1*(N/4); i < 2*(N/4); i++)
        v3[i] = v1[i] + v2[i];
    #pragma omp section
    for (i = 2*(N/4); 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];
}

```

```

        cgt2 = omp_get_wtime();
        ncgt=(cgt2-cgt1)+((cgt2-cgt1)/(1.e+9));
        //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\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

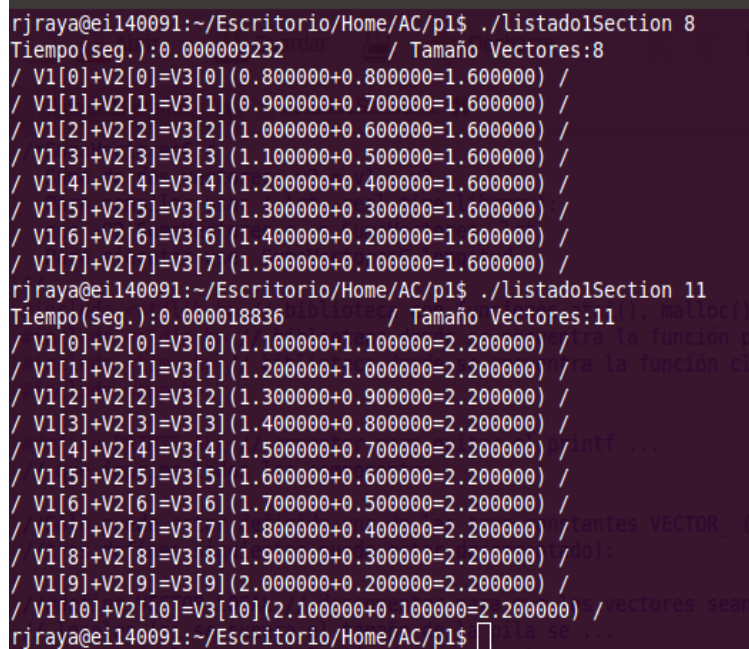
        #ifdef VECTOR_DYNAMIC
            free(v1); // libera el espacio reservado para v1
            free(v2); // libera el espacio reservado para v2
            free(v3); // libera el espacio reservado para v3
        #endif

return 0;
}

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)**CAPTURAS DE PANTALLA:**

Tamaños pequeños para comprobar que calcula el resultado correctamente:



```

rjraya@eil40091:~/Escritorio/Home/AC/pl$ ./listado1Section 8
Tiempo(seg.):0.000009232 / Tamaño Vectores:8
/ V1[0]+V2[0]=V3[0] (0.800000+0.800000=1.600000) /
/ V1[1]+V2[1]=V3[1] (0.900000+0.700000=1.600000) /
/ V1[2]+V2[2]=V3[2] (1.000000+0.600000=1.600000) /
/ V1[3]+V2[3]=V3[3] (1.100000+0.500000=1.600000) /
/ V1[4]+V2[4]=V3[4] (1.200000+0.400000=1.600000) /
/ V1[5]+V2[5]=V3[5] (1.300000+0.300000=1.600000) /
/ V1[6]+V2[6]=V3[6] (1.400000+0.200000=1.600000) /
/ V1[7]+V2[7]=V3[7] (1.500000+0.100000=1.600000) /
rjraya@eil40091:~/Escritorio/Home/AC/pl$ ./listado1Section 11
Tiempo(seg.):0.000018836 / Tamaño Vectores:11
/ V1[0]+V2[0]=V3[0] (1.100000+1.100000=2.200000) /
/ V1[1]+V2[1]=V3[1] (1.200000+1.000000=2.200000) /
/ V1[2]+V2[2]=V3[2] (1.300000+0.900000=2.200000) /
/ V1[3]+V2[3]=V3[3] (1.400000+0.800000=2.200000) /
/ V1[4]+V2[4]=V3[4] (1.500000+0.700000=2.200000) /
/ V1[5]+V2[5]=V3[5] (1.600000+0.600000=2.200000) /
/ V1[6]+V2[6]=V3[6] (1.700000+0.500000=2.200000) /
/ V1[7]+V2[7]=V3[7] (1.800000+0.400000=2.200000) /
/ V1[8]+V2[8]=V3[8] (1.900000+0.300000=2.200000) /
/ V1[9]+V2[9]=V3[9] (2.000000+0.200000=2.200000) /
/ V1[10]+V2[10]=V3[10] (2.100000+0.100000=2.200000) /
rjraya@eil40091:~/Escritorio/Home/AC/pl$

```

Tamaños mayores para medir tiempos:

```

rjraya@eil40091:~/Escritorio/Home/AC/pl$ ./listado1Section 10000000
Tiempo(seg.):0.014935329 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000)v1[9999999]+v2[9999999]=2000000.000000 /
rjraya@eil40091:~/Escritorio/Home/AC/pl$ time ./listado1Section 10000000
Tiempo(seg.):0.015119974 / Tamaño Vectores:10000000 / V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000)v1[9999999]+v2[9999999]=2000000.000000 /
real    0m0.044s
user    0m0.008s
sys     0m0.057s
rjraya@eil40091:~/Escritorio/Home/AC/pl$

```

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:

Sections fija el número de hebras a 4 (número de “sections” que utilizo).

Parallel for se reparte las iteraciones entre todas las hebras.

Así, en atcgrid, con un N alto se utilizarían todas las hebras.

10. Rellenar una tabla como la Tabla 2 para atcgrid y otra para el PC local con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución del trozo de código que realiza la suma en paralelo (este es el tiempo que deben imprimir los programas). Ponga en la tabla el número de threads/cores que usan los códigos. Represente en una gráfica los tres tiempos.

RESPUESTA:

Tabla 1. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas en el PC del aula de prácticas.

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	0,000254949	0,000641256	0,00062443
131072	0,000512377	0,000700225	0,00065232
262144	0,000769406	0,001229595	0,001015458
524288	0,001614732	0,004081765	0,001581263
1048576	0,002815389	0,002593487	0,00614131
2097152	0,005701782	0,005164376	0,007011527
4194304	0,011205567	0,010069653	0,009718028
8388608	0,022454632	0,019210002	0,021518402
16777216	0,042459965	0,038063089	0,037549191
33554432	0,083992262	0,075313098	0,074580815
67108864	0,083046877	0,075601614	0,074726436

Diagrama 1. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas en el PC del aula de prácticas. Se puede apreciar que la mejor versión es la paralela que usa la directiva “section” y la peor es la versión secuencial.

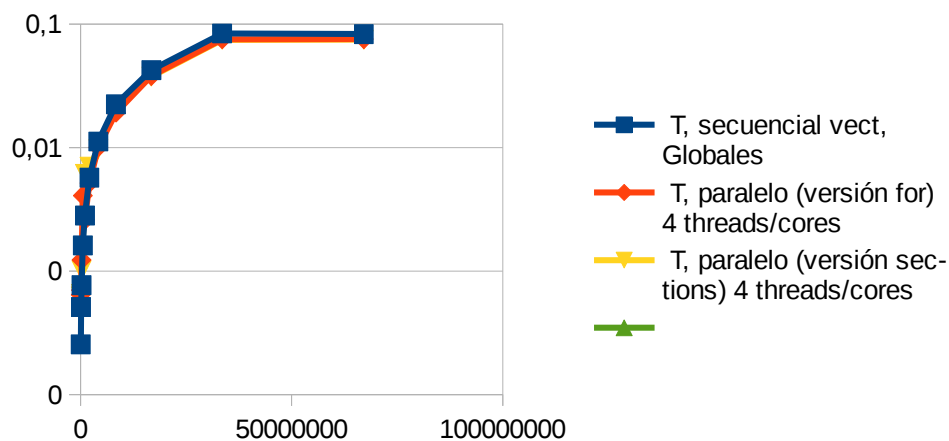
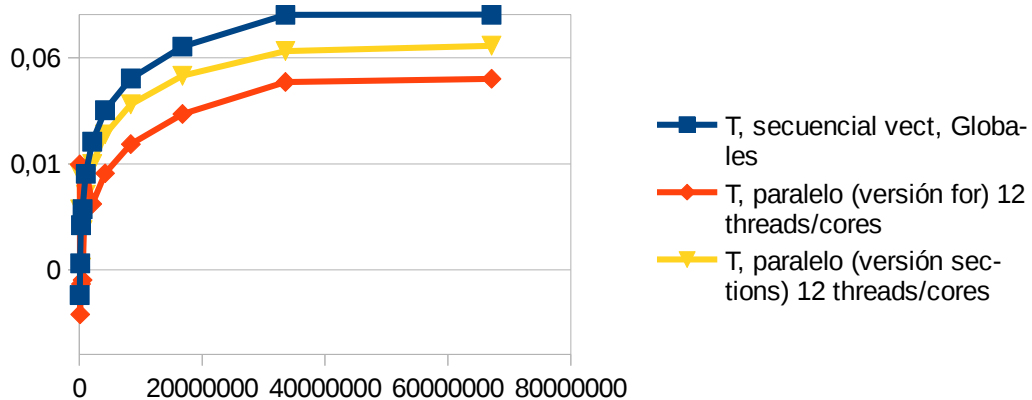


Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas en ATCGRID.

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 12 threads/cores	T. paralelo (versión sections) 12 threads/cores
65536	0,000361707	0,006176826	0,00511327
131072	0,000721784	0,00023851	0,002325342
262144	0,00165224	0,000460639	0,000658956
524288	0,002337578	0,000501337	0,001477011
1048576	0,005037839	0,005576255	0,003206917
2097152	0,010112513	0,002624333	0,006245741
4194304	0,020057947	0,005098094	0,011864214
8388608	0,039884558	0,009593646	0,022760271
16777216	0,079840965	0,01844974	0,042302396
33554432	0,159152572	0,037017049	0,072389336
67108864	0,159446193	0,039702963	0,081215749

Diagrama 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas en ATCGRID. Se puede apreciar que la mejor versión es la paralela que usa la directiva “parallel for” y la peor es la versión secuencial. Aquí ya no estamos limitados por el número de threads y las directivas “section” dejan de tener ventaja.

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:

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/cores			Tiempo paralelo/versión for 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	0,004	0	0	0,004	0,004	0
131072	0,004	0	0	0,007	0,012	0,004
262144	0,005	0	0	0,009	0,012	0,008
524288	0,006	0,004	0	0,011	0,024	0
1048576	0,009	0,004	0,004	0,01	0,016	0,012
2097152	0,015	0,004	0,008	0,013	0,02	0,024
4194304	0,028	0,008	0,016	0,028	0,048	0,032
8388608	0,051	0,04	0,008	0,047	0,116	0,028
16777216	0,097	0,06	0,036	0,086	0,2	0,108
33554432	0,19	0,128	0,06	0,17	0,428	0,212
67108864	0,191	0,124	0,064	0,17	0,412	0,228

Siguiendo a Chapman se observa que el tiempo real es ligeramente mayor que el tiempo de CPU (en el caso secuencial). De entre las muchas razones por las que puede ocurrir esto, hay una común: la aplicación no obtuvo un procesador completamente para él porque había cierta carga en el sistema.

En el caso del paralelismo de la versión paralela `for` ocurre que el tiempo real es menor que el tiempo de CPU, puesto que las tareas están repartidas entre los distintos threads.