

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

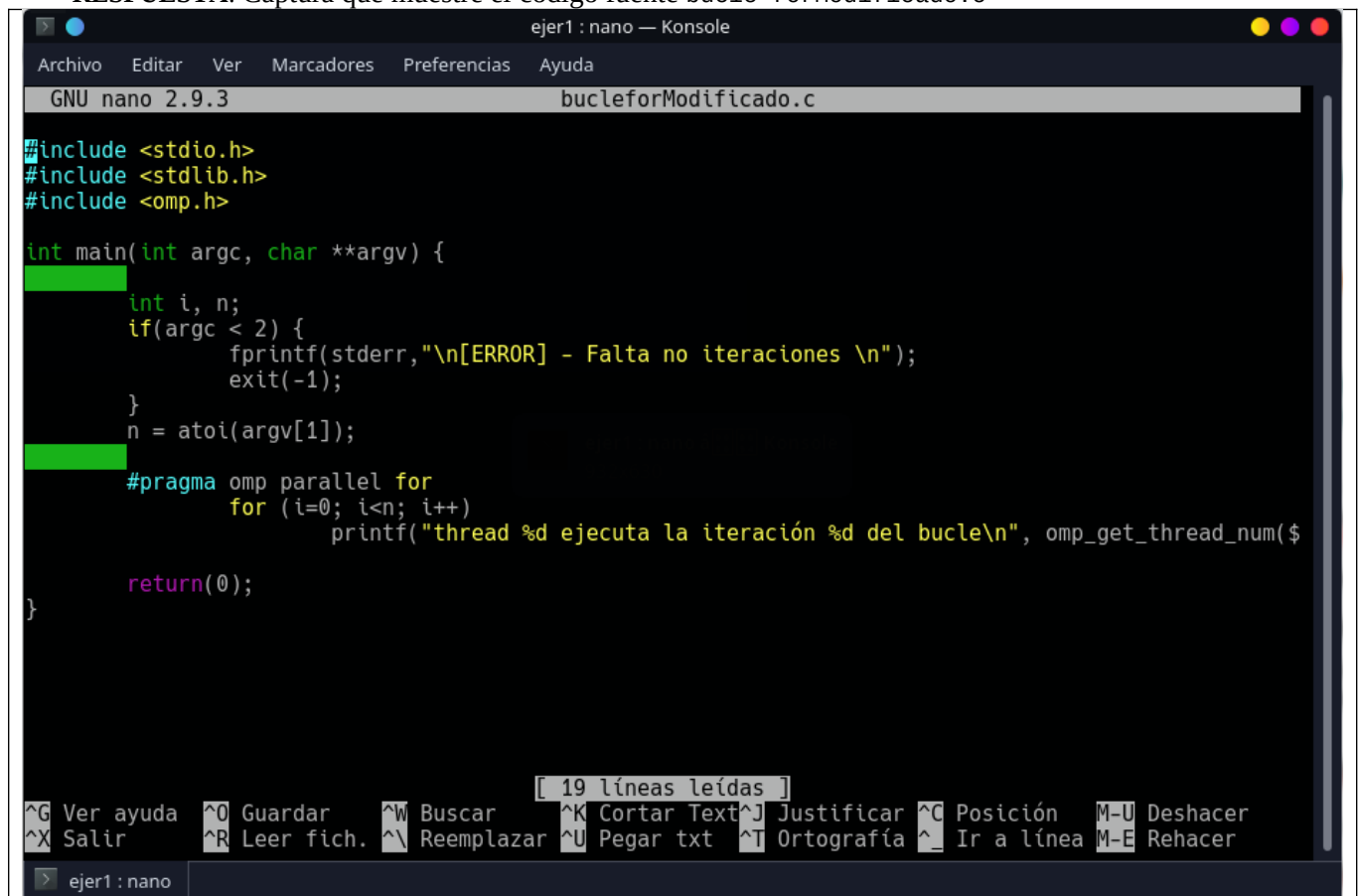
Estudiante (nombre y apellidos): Miguel Ángel Fernández Gutiérrez
Grupo de prácticas y profesor de prácticas: GIM2, Francisco Barranco
Fecha de entrega: 3 abril 2019
Fecha evaluación en clase: 4 abril 2019

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

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: Captura que muestre el código fuente `bucle-forModificado.c`



```
GNU nano 2.9.3 bucleforModificado.c

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

int main(int argc, char **argv) {
    int i, n;
    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($

    return(0);
}
```

[19 líneas leídas]

Ver ayuda Guardar Buscar Cortar Text Justificar Posición Deshacer
Salir Leer fich. Reemplazar Pegar txt Ortografía Ir a línea Rehacer

RESPUESTA: Captura que muestre el código fuente `sectionsModificado.c`

```

ejer1 : nano — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
GNU nano 2.9.3  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());
}

int main() {
    #pragma omp parallel sections
    {
        #pragma omp section
        (void) funcA();
        #pragma omp section
        (void) funcB();
    }
}

[ 20 líneas leídas ]
^G Ver ayuda  ^O Guardar    ^W Buscar     ^K Cortar Text^J Justificar  ^C Posición   M-U Deshacer
^X Salir       ^R Leer fich. ^\ Reemplazar ^U Pegar txt  ^T Ortografía ^_ Ir a línea   M-E Rehacer
> ejer1 : nano

```

- 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: Captura que muestre el código fuente `singleModificado.c`

```

GNU nano 2.9.3 singleModificado.c

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

int 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("En la región parallel:\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());
        }
    }
}

^G Ver ayuda  ^O Guardar    ^W Buscar    ^K Cortar Text^J Justificar  ^C Posición  M-U Deshacer
^X Salir      ^R Leer fich. ^\ Reemplazar^K Pegar txt  ^T Ortografía ^_ Ir a línea  M-E Rehacer

> ejer2 : nano

```

CAPTURAS DE PANTALLA:

```

ejer2 : bash — Konsole

Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda

[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp1/ejer2] 2019-04-03 miércoles
$ ./singleModificado
Introduce valor de inicialización a: 14
Single ejecutada por el thread 1
En 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
Single ejecutada por el thread 2
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp1/ejer2] 2019-04-03 miércoles
$

> ejer2 : bash

```

- 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: Captura que muestre el código fuente singleModificado2.c

```

ejer3 : nano — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
GNU nano 2.9.3  singleModificado2.c

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

int 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("En la región parallel:\n");
            for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
            printf("\n");
            printf("Master ejecutada por el thread %d\n", omp_get_thread_num());
        }
    }
}

^G Ver ayuda  ^O Guardar  ^W Buscar  ^K Cortar Text  ^J Justificar  ^C Posición  M-U Deshacer
^X Salir      ^R Leer fich.  ^\ Reemplazar  ^U Pegar txt  ^T Ortografía  ^_ Ir a línea  M-E Rehacer
> ejer3 : nano

```

CAPTURAS DE PANTALLA:

```

ejer3 : bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp1/ejer3] 2019-04-03 miércoles
$ ./singleModificado2
Introduce valor de inicialización a: 7
Single ejecutada por el thread 1
En la región parallel:
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
Master ejecutada por el thread 0
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp1/ejer3] 2019-04-03 miércoles
$ ./singleModificado2
Introduce valor de inicialización a: 7
Single ejecutada por el thread 6
En la región parallel:
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
Master ejecutada por el thread 0
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp1/ejer3] 2019-04-03 miércoles
$
> ejer3 : bash

```

RESPUESTA A LA PREGUNTA:

La diferencia es que todas las instrucciones contenidas en una directiva “master” son ejecutadas por el thread maestro, que corresponde al identificador 0. Podemos ver que ejecutando el código varias veces, el identificador que se imprime tras el resultado siempre es 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:

Esto ocurre porque tras “atomic” no hay barrera implícita. Esto, en consecuencia, implica que es posible ejecutar el “printf” antes de que se hayan acumulado todas las sumas, por lo que el resultado sería incorrecto (uno de los hilos, tan pronto como no esté ocupado, ejecutará dicho “printf”). La barrera impide esto: espera a que todos los hilos hayan realizado la instrucción de adición acumulativa sobre “suma”, y a continuación se ejecuta el “printf”, teniendo en consecuencia un resultado correcto.

Resto de ejercicios nmtbpdptb

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 atcgrid, 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 menor, mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

CAPTURAS DE PANTALLA:

```

(E2estudiante29) atcgrid.ugr.es — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[MiguelÁngelFernándezGutiérrez E2estudiante29@atcgrid:~] 2019-04-03 miércoles
$echo "time bp1/ejer5/Listado1 10000000" | qsub -q ac
17888.atcgrid
[MiguelÁngelFernándezGutiérrez E2estudiante29@atcgrid:~] 2019-04-03 miércoles
$cat *e17888

real    0m0.112s
user    0m0.059s
sys     0m0.050s
[MiguelÁngelFernándezGutiérrez E2estudiante29@atcgrid:~] 2019-04-03 miércoles
$cat *o17888
Tiempo(seg.):0.043403624      / Tamaño Vectores:10000000      / V1[0]+V2[0]=V3[0](1000000.000000+1
000000.000000=2000000.000000) / / V1[999999]+V2[999999]=V3[999999](199999.900000+0.100000=200000
0.000000) /
[MiguelÁngelFernándezGutiérrez E2estudiante29@atcgrid:~] 2019-04-03 miércoles
$

```

Vemos que la suma de tiempo de usuario y sistema es $0.059+0.050 = 0.109s < 0.112s$. Esto es así debido al tiempo de espera debida a operaciones, por ejemplo, de E/S.

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). Razonar cómo se han

obtenido los valores que se necesitan para calcular los MIPS y MFLOPS. Incorpore **el código ensamblador de la parte de la suma de vectores** en el cuaderno.

CAPTURAS DE PANTALLA (que muestren la generación del código ensamblador y del código ejecutable, y la obtención de los tiempos de ejecución):

RESPUESTA: cálculo de los MIPS y los MFLOPS

Calcularemos los MIPS y MFLOPS

Para 10 componentes:

$$\text{MIPS} = \text{NI} / (\text{T_cpu} * 10^6) = 14 / 0.036 / 10^6 = 0.00038$$

$$\text{MFLOPS} = \text{NI_float} / (\text{T_cpu} * 10^6) = 4 / 0.036 / 10^6 = 0.00011$$

Para 10000000 componentes:

$$\text{MIPS} = \text{NI} / (\text{T_cpu} * 10^6) = 14 / 0.112 / 10^6 = 0.000125$$

$$\text{MFLOPS} = \text{NI_float} / (\text{T_cpu} * 10^6) = 4 / 0.112 / 10^6 = 0.0000357$$

RESPUESTA: Captura que muestre el código ensamblador generado de la parte de la suma de vectores

```

ejer6 : nano — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
GNU nano 2.9.3  Listado1.s

    andl    $1, %eax
    orq     %rax, %rdx
    cvtsi2sdq    %rdx, %xmm0
    addsd   %xmm0, %xmm0
.L8:
    movsd   .LC1(%rip), %xmm1
    mulsd   %xmm1, %xmm0
    cvtsi2sd    -64(%rbp), %xmm1
    movsd   .LC1(%rip), %xmm2
    mulsd   %xmm2, %xmm1
    subsd   %xmm1, %xmm0
    movl    -64(%rbp), %eax
    cltq
    leaq    0(,%rax,8), %rdx
    leaq    v2(%rip), %rax
    movsd   %xmm0, (%rdx,%rax)
    addl    $1, -64(%rbp)
.L4:
    movl    -64(%rbp), %eax
    cmpl    %eax, -60(%rbp)
    ja      .L9
    leaq    -48(%rbp), %rax
    movq     %rax, %rsi
    movl    $0, %edi
    call    clock_gettime@PLT
    movl    $0, -64(%rbp)
    jmp     .L10
.L11:
    movl    -64(%rbp), %eax
    cltq
    leaq    0(,%rax,8), %rdx
    leaq    v1(%rip), %rax
    movsd   (%rdx,%rax), %xmm1
    movl    -64(%rbp), %eax
    cltq
    leaq    0(,%rax,8), %rdx
    leaq    v2(%rip), %rax
    movsd   (%rdx,%rax), %xmm0
    addsd   %xmm1, %xmm0
    movl    -64(%rbp), %eax
    cltq
    leaq    0(,%rax,8), %rdx
    leaq    v3(%rip), %rax
    movsd   %xmm0, (%rdx,%rax)
    addl    $1, -64(%rbp)
.L10:
    movl    -64(%rbp), %eax
    cmpl    %eax, -60(%rbp)
    ja      .L11
    leaq    -32(%rbp), %rax
    movq     %rax, %rsi
    movl    $0, %edi
    call    clock_gettime@PLT
    movq     -32(%rbp), %rdx
    movq     -48(%rbp), %rax
    subq     %rax, %rdx
    movq     %rdx, %rax

```

[^]G Ver ayuda [^]O Guardar [^]W Buscar [^]K Cortar Text [^]J Justificar [^]C Posición ^M-U Deshacer
[^]X Salir [^]R Leer fich. [^]\ Reemplazar [^]U Pegar txt [^]T Ortografía [^] Ir a línea ^M-E Rehacer

```

> ejer6 : nano

```

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: Captura que muestre el código fuente implementado

NOTA: se muestra la parte que difiere del Listado1.c


```

ejer7 : nano — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
GNU nano 2.9.3  Listado1Ejer7.c

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
v3 = (double*) malloc(N*sizeof(double));
if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
    printf("Error en la reserva de espacio para los vectores\n");
    exit(-2);
}
#endif

#pragma omp parallel for
//Inicializar vectores
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
}

double start = omp_get_wtime();

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

ncgt=omp_get_wtime() - start;

//Imprimir resultado de la suma y el tiempo de ejecución
if (N<10) {
    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) $
ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);

#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;

^G Ver ayuda ^O Guardar ^W Buscar ^K Cortar Text ^J Justificar ^C Posición M-U Deshacer
^X Salir ^R Leer fich. ^\ Reemplazar ^U Pegar txt ^T Ortografía ^_ Ir a línea M-E Rehacer
> ejer7 : nano

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA (compilación y ejecución para N=8 y N=11):

```

ejer7 : bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp1/ejer7] 2019-04-03 miércoles
$time ./Listado1Ejer7 8
Tiempo(seg.):0.000000000 / 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) /

real    0m0.005s
user    0m0.000s
sys     0m0.015s
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp1/ejer7] 2019-04-03 miércoles
$time ./Listado1Ejer7 11
Tiempo(seg.):0.000000000 / 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) /

real    0m0.005s
user    0m0.000s
sys     0m0.017s
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp1/ejer7] 2019-04-03 miércoles
ejer7 : bash

```

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, $N = 8$); (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: Captura que muestre el código fuente implementado

NOTA: se muestra lo que difiere de Listado1.c

```

ejer8: nano — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
GNU nano 2.9.3  Listado1Ejer8.c

#pragma omp parallel sections
{
    //Inicializar vectores
    #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; //los valores dependen de N
    }
    #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; //los valores dependen de N
    }
    #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; //los valores dependen de N
    }
    #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; //los valores dependen de N
    }
}

double start = omp_get_wtime();

#pragma omp parallel sections
{
    //Calcular suma de vectores
    #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/2; i++)
        v3[i] = v1[i] + v2[i];
    #pragma omp section
    for(i=3*N/2; i<N; i++)
        v3[i] = v1[i] + v2[i];
}

^G Ver ayuda  ^O Guardar  ^W Buscar  ^K Cortar Text  ^J Justificar  ^C Posición  M-U Deshacer
^X Salir      ^R Leer fich.  ^\ Reemplazar  ^U Pegar txt  ^T Ortografía  ^_ Ir a línea  M-E Rehacer
> ejer8: nano

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA (compilación y ejecución para N=8 y N=11):

```

ejer8 : bash — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp1/ejer8] 2019-04-03 miércoles
$time ./Listado1Ejer8 8
Tiempo(seg.):0.000000000 / 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) /

real    0m0.005s
user    0m0.020s
sys     0m0.001s
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp1/ejer8] 2019-04-03 miércoles
$time ./Listado1Ejer8 11
Tiempo(seg.):0.000000000 / 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) /

real    0m0.004s
user    0m0.006s
sys     0m0.001s
[MiguelÁngelFernándezGutiérrez mianfg@mianfg-PE62-7RD:~/AC_practicas/bp1/ejer8] 2019-04-03 miércoles
$

```

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:

En el ejercicio 7, podríamos utilizar un máximo de N threads, sin embargo, este número se limita por el número de cores lógicos del ordenador en el que estemos trabajando, o de la constante definida por OpenMP. Por otra parte, el ejercicio 8 siempre usará un máximo de 4 threads, porque hemos dividido el “for” en cuatro secciones paralelas.

10. Rellenar una tabla como la Tabla 2 para atcgrid y otra para su PC 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 (use el máximo número de cores físicos del computador que como máximo puede aprovechar el código, no use un número de threads superior al número de cores físicos). Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute código que imprima todos los componentes del resultado cuando este número sea elevado.

RESPUESTA:

Las tablas son:

TABLA PARA PC

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) ¿?threads/cores	T. paralelo (versión sections) ¿?threads/cores
16384	0,000160219	5,8026E-05	0,00016861
32768	0,000343571	7,8749E-05	4,0197E-05
65536	0,000640855	0,000151106	7,715E-05
131072	0,001242147	0,000340679	0,001567013
262144	0,002490441	0,000574024	0,00123543
524288	0,005010069	0,001152271	0,000889739
1048576	0,004224835	0,001746891	0,00190161
2097152	0,007028418	0,005891664	0,003761898
4194304	0,012448721	0,009359407	0,010216318
8388608	0,02434209	0,016472493	0,017184797
16777216	0,048921489	0,028106719	0,029148952
33554432	0,10422794	0,062284827	0,057714343
67108864	0,101154211	0,056429033	0,063015746

GRÁFICA PARA PC

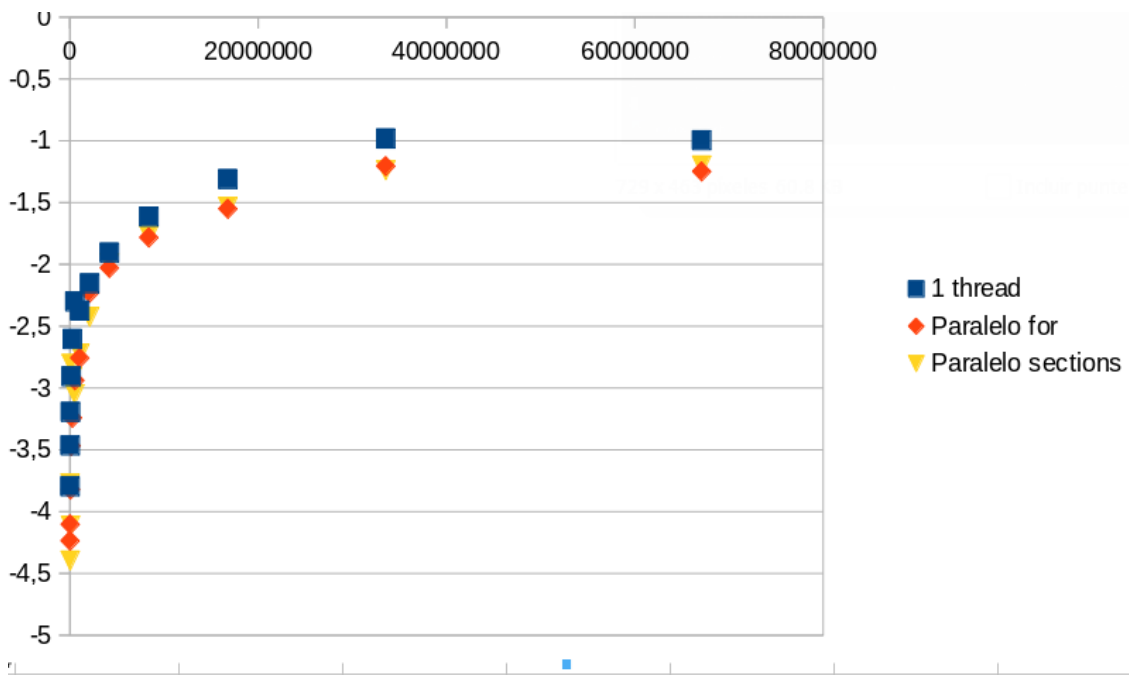


TABLA PARA ATCGRID

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) 24 threads/cores	T. paralelo (versión sections) 4 threads/cores
16384	0,000442263	0,004733769	0,000721092
32768	0,000489119	0,003093275	0,003944734
65536	0,000358651	0,003186956	0,004547087
131072	0,000503664	0,003056407	0,004546847
262144	0,001441406	0,003544667	0,004549864
524288	0,002684234	0,003986245	0,0010266
1048576	0,0057366	0,005836043	0,003119511
2097152	0,010166336	0,006889466	0,008367067
4194304	0,01861657	0,010090537	0,011574421
8388608	0,034639633	0,1671189	0,33201216
16777216	0,067816011	0,03492474	0,043756628
33554432	0,135165787	0,064476436	0,107116851
67108864	0,134723415	0,06460002	0,10837692

GRÁFICA PARA ATCGRID

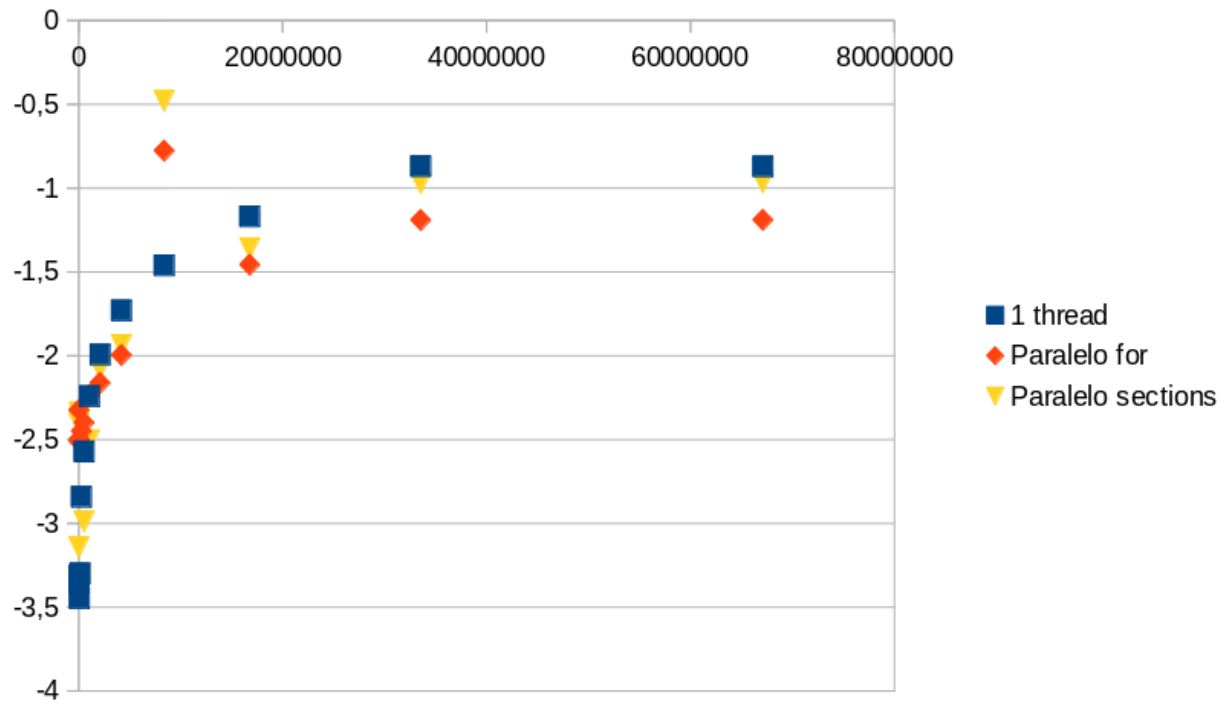


Tabla 2. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla “¿?” por el número de threads utilizados, que debe coincidir con el número de cores físicos del computador que como máximo puede aprovechar el código.

Nº de Componentes	T. secuencial vect. Globales 1 thread/core	T. paralelo (versión for) ¿?threads/cores	T. paralelo (versión sections) ¿?threads/cores
16384			
32768			
65536			
131072			
262144			
524288			
1048576			
2097152			
4194304			
8388608			
16777216			
33554432			
67108864			

11. Rellenar una tabla como la Tabla 3 para atcgrid 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: En este caso, el tiempo será menor porque no estamos contando el tiempo debido a esperas de E/S o a la espera de ejecución de otros programas.

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		
	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>	<i>Elapsed</i>	<i>CPU-user</i>	<i>CPU- sys</i>
65536	0,003	0,001	0,002	0,011	0,068	0,138
131072	0,003	0,002	0,001	0,010	0,032	0,155
262144	0,006	0,002	0,004	0,015	0,106	0,187
524288	0,009	0,001	0,008	0,015	0,124	0,167
1048576	0,0015	0,006	0,009	0,018	0,141	0,212
2097152	0,027	0,013	0,014	0,024	0,266	0,197
4194304	0,046	0,023	0,023	0,028	0,358	0,195
8388608	0,084	0,048	0,036	0,042	0,662	0,258
16777216	0,160	0,088	0,071	0,086	1,246	0,513
33554432	0,316	0,166	0,150	0,160	1,246	2,367
67108864	0,316	0,186	0,130	0,159	2,475	0,960