

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): David Sánchez Jiménez

Grupo de prácticas: A3

Fecha de entrega: 05/04/2018

Fecha evaluación en clase: 06/04/2018

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`

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.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 iteracion %d del bucle\n",
            omp_get_thread_num(), i);
    }

    return 0;
}
```

RESPUESTA: Captura que muestre el código fuente sectionsModificado.c

```
#include <omp.h>
#include <stdio.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(int argc, char const *argv[]) {
    #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: Captura que muestre el código fuente `singleModificado.c`

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

int main(int argc, char const *argv[]) {
    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("Despues de la región parallel:\n");
            for (i = 0; i < n; i++) {
                printf("b[%d] = %d\t", i, b[i]);
            }
            printf("\n");
        }
        return 0;
    }
}
```

CAPTURAS DE PANTALLA:

```
jue 5 abr - 20:34 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_1
@ david ./singleModificado
Introduce valor de inicializacion a:23
Single ejecutada por el thread 2
Despues de la región parallel:
b[0] = 23    b[1] = 23    b[2] = 23    b[3] = 23    b[4] = 23    b[5] = 23    b[6] = 23    b[7] = 23    b[8] = 23

jue 5 abr - 20:35 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_1
@ david ./singleModificado
Introduce valor de inicializacion a:12
Single ejecutada por el thread 1
Despues de la región parallel:
b[0] = 12    b[1] = 12    b[2] = 12    b[3] = 12    b[4] = 12    b[5] = 12    b[6] = 12    b[7] = 12    b[8] = 12
```

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: Captura que muestre el código fuente `singleModificado2.c`

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

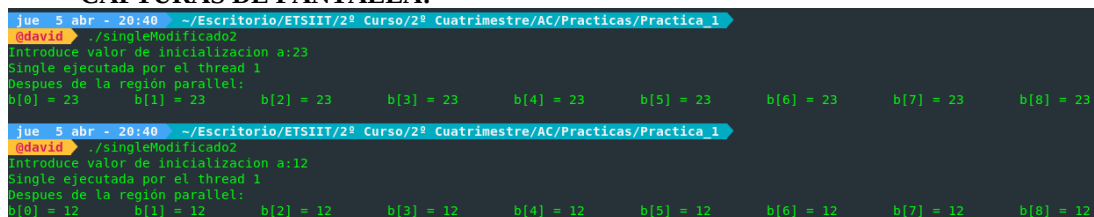
int main(int argc, char const *argv[]) {
    int n = 9, i, a, b[n];

    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("Despues de la región parallel:\n");
            for (i = 0; i < n; i++) {
                printf("b[%d] = %d\t", i, b[i]);
            }
            printf("\n");
        }
        return 0;
    }
```

CAPTURAS DE PANTALLA:



```
jue 5 abr - 20:40 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_1
@david ./singleModificado2
Introduce valor de inicializacion a:23
Single ejecutada por el thread 1
Despues de la región parallel:
b[0] = 23    b[1] = 23    b[2] = 23    b[3] = 23    b[4] = 23    b[5] = 23    b[6] = 23    b[7] = 23    b[8] = 23

jue 5 abr - 20:40 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_1
@david ./singleModificado2
Introduce valor de inicializacion a:12
Single ejecutada por el thread 1
Despues de la región parallel:
b[0] = 12    b[1] = 12    b[2] = 12    b[3] = 12    b[4] = 12    b[5] = 12    b[6] = 12    b[7] = 12    b[8] = 12
```

RESPUESTA A LA PREGUNTA: Tras varias ejecuciones se puede observar que los resultados siempre son imprimidos por la misma hebra (la hebra 1) mientras que en el caso anterior cualquier hebra podía imprimir los resultados.

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: Debido a que la directiva `barrier` es la encargada de esperar a que todas las hebras hayan sumado su resultado a la variable `suma`, por tanto, si no se produce dicha espera no se realiza correctamente la suma y se imprime un resultado erróneo.

Resto de ejercicios

5. El programa secuencial C del Listado 1 calcula la suma de dos vectores ($v3 = v1 + v2$; $v3(i) = v1(i) + v2(i)$, $i=0, \dots, N-1$). 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:

```
[A3estudiante24@atcgrid ~]$ time echo 'sesion1/SumaVectores 10000000' | qsub -q ac
72092 atcgrid
real    0m0.034s
user    0m0.012s
sys     0m0.009s
[A3estudiante24@atcgrid ~]$ cat STDIN.q72092
Tiempo(seg.): 0.06405045      Tamaño Vectores: 10000000      V1[0]+V2[0]=V3[0](1000000.000000+1000000.000000=2000000.000000) V1[9999999]=
0210000001+031000000111000000 000000+0 100000+000000 000000
```

El tiempo de CPU = `user` + `sys` es el tiempo que tarda en ejecutarse el código correspondiente a las llamadas del sistema y el código del programa mientras que el tiempo real (*elapsed*) también incluye las esperas de E/S que no forman parte del tiempo de CPU por lo que este último es mayor.

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:

```
[A3estudiante24@atcgrid ~]$ echo 'sesion1/SumaVectores 10' | qsub -q ac
72097.atcgrid
[A3estudiante24@atcgrid ~]$ cat STDIN.o72097
Tiempo(seg.):0.000002573      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)
[A3estudiante24@atcgrid ~]$ echo 'sesion1/SumaVectores 10000000' | qsub -q ac
72098.atcgrid
[A3estudiante24@atcgrid ~]$ cat STDIN.o72098
Tiempo(seg.):0.066704136      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)
```

RESPUESTA: cálculo de los MIPS y los MFLOPS

Para 10 componentes obtengo el valor de 0.000002573 segundos. En el código ensamblador tenemos 7 instrucciones fuera del bucle y dentro del bucle tenemos 6 instrucciones, que multiplicadas por el número de iteraciones que es 10 nos da un total de 60 instrucciones, por tanto, tenemos 67 instrucciones

$$MIPS = \frac{67}{0.000002573 \cdot 10^6} \cdot 10^9 = 26,039642441$$

En atcgrid se ejecutan 30 millones de instrucciones por segundo. De esas 67 instrucciones 30 son de coma flotante por lo que:

$$MFLOPS = \frac{30}{0.000002573 \cdot 10^6} \cdot 10^9 = 11,659541391$$

Ahora repito los calculos para 10000000 componentes

$$MIPS = \frac{60000007}{0.066704136 \cdot 10^6} \cdot 10^9 = 899494553081,386137735$$

$$MFLOPS = \frac{30000000}{0.066704136 \cdot 10^6} \cdot 10^9 = 449747224070,183594013$$

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

```
call    clock_gettime@PLT
xorl    %eax, %eax
.p2align 4,,10
.p2align 3
.L5:
    movsd    (%r12,%rax,8), %xmm0
    addsd    0(%rbp,%rax,8), %xmm0
    movsd    %xmm0, 0(%r13,%rax,8)
    addq     $1, %rax
    cmpl     %eax, %ebx
    ja       .L5
.L6:
    leaq     16(%rsp), %rsi
    xorl     %edi, %edi
    call     clock_gettime@PLT
```

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

```
// 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
}

double start = omp_get_wtime();

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

double end = omp_get_wtime();
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
jue 5 abr - 22:30 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_1
@dauid gcc -O2 -fopenmp SumaVectoresCModificado.c -o SumaVectoresCModificado -lrt

jue 5 abr - 22:30 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_1
@dauid ./SumaVectoresCModificado 8
Tiempo(seg.):0.000003985 / 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) /

jue 5 abr - 22:30 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_1
@dauid ./SumaVectoresCModificado 11
Tiempo(seg.):0.000004037 / 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) /
```

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: Captura que muestre el código fuente implementado

```
// Calcular suma de vectores
#pragma omp parallel sections
{
    #pragma omp section
    {
        for (primer_trozo = 0; primer_trozo < (N / 4); primer_trozo++)
            v3[primer_trozo] = v1[primer_trozo] + v2[primer_trozo];
    }
    #pragma omp section
    {
        for (segundo_trozo = (N / 4); segundo_trozo < (N / 2); segundo_trozo++)
            v3[segundo_trozo] = v1[segundo_trozo] + v2[segundo_trozo];
    }
    #pragma omp section
    {
        for (tercer_trozo = (N / 2); tercer_trozo < ((3 * N) / 4); tercer_trozo++)
            v3[tercer_trozo] = v1[tercer_trozo] + v2[tercer_trozo];
    }
    #pragma omp section
    {
        for (cuarto_trozo = ((3 * N) / 4); cuarto_trozo < N; cuarto_trozo++)
            v3[cuarto_trozo] = v1[cuarto_trozo] + v2[cuarto_trozo];
    }
}

end = omp_get_wtime();
tiempo = end - start;
printf("%11.9f\n", tiempo);
```


(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```
jue 5 abr - 22:46 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_1
@david gcc -O2 -fopenmp SumaVectoresCModificado2.c -o SumaVectoresCModificado2 -lrt

jue 5 abr - 22:46 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_1
@david ./SumaVectoresCModificado2 8
0.000002806
Tiempo(seg.):0.000002806 / 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) /

jue 5 abr - 22:46 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_1
@david ./SumaVectoresCModificado2 11
0.000004197
Tiempo(seg.):0.000004197 / 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) /
```

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 la directiva for crea tantas hebras como se le indique con la constante OMP_NUM_THREADS que se corresponde con el numero de cores del PC mientras que en el 8 hemos dividido el bucle en 4 partes y por tanto se crean 4 hebras.

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. Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute código que imprima todos los componentes del resultado cuando este número sea elevado.

RESPUESTA:

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

Nº de Componentes	T. secuencial 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:

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						
131072						
262144						
524288						
1048576						
2097152						
4194304						
8388608						
16777216						
33554432						
67108864						