

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): César Muñoz Reinoso

Grupo de prácticas: Grupo 2

Fecha de entrega: 28/03

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:** Captura que muestre el 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:** Captura que muestre el 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:** Captura que muestre el código fuente 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());
            }
        }
    }
}
```

#### CAPTURAS DE PANTALLA:

```
cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas
Archivos/BP1$ ./singleModificado
Introduce valor de inicialización a: 4
Single ejecutada por el thread 1
En la región parallel:
b[0] = 4
Single ejecutada por el thread 3
b[1] = 4
Single ejecutada por el thread 3
b[2] = 4
Single ejecutada por el thread 3
b[3] = 4
Single ejecutada por el thread 3
b[4] = 4
Single ejecutada por el thread 3
b[5] = 4
Single ejecutada por el thread 3
b[6] = 4
Single ejecutada por el thread 3
b[7] = 4
Single ejecutada por el thread 3
b[8] = 4
Single ejecutada por el thread 3
```

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 <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("Single ejecutada por el thread %d\n",omp_get_thread_num());
            }
        }
    }
}
```

**CAPTURAS DE PANTALLA:**

```
cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas
Archivos/BP1$ gcc -O2 -fopenmp singleModificado2.c -o singleModificado2
cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas
Archivos/BP1$ ./singleModificado2
Introduce valor de inicialización a: 4
Single ejecutada por el thread 2
En la región parallel:
b[0] = 4
Single ejecutada por el thread 0
b[1] = 4
Single ejecutada por el thread 0
b[2] = 4
Single ejecutada por el thread 0
b[3] = 4
Single ejecutada por el thread 0
b[4] = 4
Single ejecutada por el thread 0
b[5] = 4
Single ejecutada por el thread 0
b[6] = 4
Single ejecutada por el thread 0
b[7] = 4
Single ejecutada por el thread 0
b[8] = 4
Single ejecutada por el thread 0
```

**RESPUESTA A LA PREGUNTA:** La diferencia es que en `singleModificado` con la directiva `single` se ejecuta el bloque estructurado con un único thread elegido de forma aleatoria, en cambio, en `singleModificado2` con la directiva `master` ejecuta todas las iteraciones con el thread 0.

4. ¿Por qué si se elimina directiva `barrier` en el ejemplo `master.c` la suma que se calcula e imprime no siempre es correcta? Responda razonadamente.

**RESPUESTA:**

Con la directiva `barrier` imponemos una barrera después de `suma += sumalocal`. Dicha suma no siempre es correcta ya que si no hubiera barrera puede ser que algunos threads no hayan terminado de ejecutarse y no de el resultado adecuado.

## 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:**

El tiempo de ejecución real no es solo `user+sys`, sino que también incluyen el tiempo de espera de E/S por eso el tiempo real es mayor que el de `CPU=user+sys`.

```

Terminal - E2estudiante17@atcgrid:~
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
[César Muñoz Reinoso E2estudiante17@atcgrid:~] 2018-03-16 viernes
$ls
hello SumaVectoresGlobales
[César Muñoz Reinoso E2estudiante17@atcgrid:~] 2018-03-16 viernes
$echo 'time ./SumaVectoresGlobales 10000000' |qsub -q ac
66731.atcgrid
[César Muñoz Reinoso E2estudiante17@atcgrid:~] 2018-03-16 viernes
$ls
hello STDIN.e66731 STDIN.o66731 SumaVectoresGlobales
[César Muñoz Reinoso E2estudiante17@atcgrid:~] 2018-03-16 viernes
$qstat
[César Muñoz Reinoso E2estudiante17@atcgrid:~] 2018-03-16 viernes

Abrir + STDIN.e66731 Guardar - + x
~/Escritorio/UNIVERSIDA...

```

```

real    0m0.003s
user    0m0.001s
sys     0m0.000s

```

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:

**RESPUESTA:** cálculo de los MIPS y los MFLOPS

Tiempo(seg.):0.000002338 / Tamaño Vectores:10

$MIPS = NI/T_{cpu} * 10^6 = 6 * 10 / 10^6 * 0.000002338 = 25,662959795 \text{ MIPS}$

$MFLOPS = Op_{float}/T_{cpu} * 10^6 = 2 * 10 / 10^6 * 0.000002338 = 8,554319932 \text{ MFLOPS}$

Tiempo(seg.):0.056323616 / Tamaño Vectores:10000000

$MIPS = NI/T_{cpu} * 10^6 = 6 * 10^7 / 10^6 * 0.056323616 = 10652,725137534 \text{ MIPS}$

$MFLOPS = Op_{float}/T_{cpu} * 10^6 = 2 * 10^7 / 10^6 * 0.056323616 = 3550,908379178 \text{ MFLOPS}$

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

```
.L4:
    movsd    (%rbx,%rax), %xmm0
    addsd    0(%rbp,%rax), %xmm0
    movsd    %xmm0, (%rdx,%rax)
    addq     $8, %rax
    cmpq     $232, %rax
    jne      .L4
```

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

```

int i;
double antes, despues, tiempototal;

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

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

#pragma omp parallel
{
    #pragma omp for
    for(i=0; i<N; i++){
        v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-i*0.1;
    }
}
antes = omp_get_wtime();
#pragma omp parallel
{
    #pragma omp for
    for(i=0; i<N; i++){
        v3[i] = v1[i] + v2[i];
    }
}

despues= (double) omp_get_wtime();
tiempototal = despues - antes;

#ifdef PRINTF_ALL
printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n", tiempototal, N);
for(i=0; i<N; i++)
    printf(" v1[%d] + v2[%d] = v3[%d] \n (%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]\n(%8.6f + %8.6f =
%8.6f) \n v1[%d] + v2[%d] = v3[%d]\n(%8.6f + %8.6f = %8.6f) \n", tiempototal, N, v1[0], v2[0], v3
[0], N-1, N-1, N-1, v1[N-1], v2[N-1], v3[N-1]);
#endif

return 0;

```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

```

cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/B
/Suma_Vectores$ gcc -O2 -fopenmp SumaVectoresC.c -o SumaVectoresGlobalesOMP -lrt
cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/B
/Suma_Vectores$ time ./SumaVectoresGlobalesOMP 8
Tiempo(seg.):0.000015365 / 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)

real    0m0,004s
user    0m0,002s
sys     0m0,003s
cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/B
/Suma_Vectores$ time ./SumaVectoresGlobalesOMP 11
Tiempo(seg.):0.000011015 / 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    0m0,008s
user    0m0,015s
sys     0m0,003s

```

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



```

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

#pragma omp parallel sections
{
    #pragma omp section
    for(int i=0; i<N/4; i++){
        v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-
i*0.1;
        printf("En la inicializacion del vector: el thread %d ejecuta
la iteración %d del bucle\n", omp_get_thread_num(),i);
    }
    #pragma omp section
    for(int i=N/4; i<N/2; i++){
        v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-
i*0.1;
        printf("En la inicializacion del vector: el thread %d ejecuta
la iteración %d del bucle\n", omp_get_thread_num(),i);
    }
    #pragma omp section
    for(int i=N/2; i<N*3/4; i++){
        v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-
i*0.1;
        printf("En la inicializacion del vector: el thread %d ejecuta
la iteración %d del bucle\n", omp_get_thread_num(),i);
    }
    #pragma omp section
    for(int i=N*3/4; i<N; i++){
        v1[i] = N*0.1+i*0.1; v2[i] = N*0.1-
i*0.1;
        printf("En la inicializacion del vector: el thread %d ejecuta
la iteración %d del bucle\n", omp_get_thread_num(),i);
    }

    antes = omp_get_wtime();
    #pragma omp parallel sections
    {
        #pragma omp section
        for(int i=0; i<N/4; i++){
            v3[i] = v1[i] + v2[i];
            printf("En la suma del vector: el thread %d ejecuta la iteración %d
del bucle\n", omp_get_thread_num(),i);
        }
        #pragma omp section
        for(int i=N/4; i<N/2; i++){
            v3[i] = v1[i] + v2[i];
            printf("En la suma del vector: el thread %d ejecuta la iteración %d
del bucle\n", omp_get_thread_num(),i);
        }
        #pragma omp section
        for(int i=N/2; i<N*3/4; i++){
            v3[i] = v1[i] + v2[i];
            printf("En la suma del vector: el thread %d ejecuta la iteración %d
del bucle\n", omp_get_thread_num(),i);
        }
        #pragma omp section
        for(int i=N*3/4; i<N; i++){
            v3[i] = v1[i] + v2[i];
            printf("En la suma del vector: el thread %d ejecuta la iteración %d
del bucle\n", omp_get_thread_num(),i);
        }
    }

    despues= (double) omp_get_wtime();
    tiempototal = despues - antes;
}

```

**(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)**

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

9. ¿Cuántos threads y cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 7? Razone su respuesta. ¿Cuántos threads y cuantos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta.

**RESPUESTA:** En el ejercicio 7 el procesador utiliza 1 socket de 2 núcleos y 2 hilos por núcleos = 4 threads; atcgrid tiene 2 socket de 6 núcleos y 2 hilos por núcleo = 24 threads. El procesador utiliza todos los threads posibles.

En el ejercicio 8 el procesador utilizar los threads por cada section, como hemos dividido el bucle en 4 partes, utilizara 4 threads = 2 cores.

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.

### 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
16384	0.000108015	0.003844198	0.003966422
32768	0.000217933	0.003563798	0.000745046
65536	0.000433658	0.004010354	0.004492082
131072	0.000711709	0.004447752	0.004086754
262144	0.001442829	0.002776119	0.004275029
524288	0.002775511	0.004331119	0.004306928
1048576	0.006253552	0.005039622	0.004549832
2097152	0.012650709	0.006085367	0.008528160
4194304	0.023943650	0.007773165	0.013308278
8388608	0.046473439	0.012728483	0.023512612
16777216	0.094993085	0.020667446	0.046592325
33554432	0.180255432	0.040199234	0.074260263
67108864	0.181777042	0.039682957	0.085494584

### 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
16384	0.000288880	0.000078918	0.000033599
32768	0.000628249	0.000062293	0.000060568
65536	0.000752070	0.000114686	0.000113857
131072	0.000539108	0.000309708	0.000244283
262144	0.001112670	0.000563096	0.000562496
524288	0.002316088	0.001137681	0.001160274
1048576	0.004376680	0.002195652	0.002243923
2097152	0.008920421	0.004126528	0.004351695
4194304	0.016819910	0.008337956	0.008172245
8388608	0.032979057	0.015926910	0.016155749
16777216	0.064342192	0.032199046	0.032017580
33554432	0.128650538	0.062943264	0.063903349
67108864	0.128805245	0.062678278	0.062682879

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 Componente s	Tiempo secuencial vect. Globales 1 thread/core			Tiempo paralelo/versión for 12 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	0m0.003s	0m0.001s	0m0.001s	0m0.011s	0m0.225s	0m0.000s
131072	0m0.004s	0m0.001s	0m0.003s	0m0.013s	0m0.213s	0m0.004s
262144	0m0.005s	0m0.000s	0m0.005s	0m0.013s	0m0.166s	0m0.007s
524288	0m0.010s	0m0.006s	0m0.004s	0m0.014s	0m0.238s	0m0.005s
1048576	0m0.019s	0m0.004s	0m0.014s	0m0.016s	0m0.236s	0m0.029s
2097152	0m0.036s	0m0.017s	0m0.019s	0m0.021s	0m0.257s	0m0.063s
4194304	0m0.074s	0m0.026s	0m0.048s	0m0.032s	0m0.291s	0m0.119s
8388608	0m0.141s	0m0.050s	0m0.089s	0m0.047s	0m0.364s	0m0.268s
16777216	0m0.281s	0m0.105s	0m0.174s	0m0.084s	0m0.511s	0m0.534s
33554432	0m0.542s	0m0.192s	0m0.345s	0m0.149s	0m0.849s	0m0.997s
67108864	0m0.536s	0m0.174s	0m0.359s	0m0.141s	0m0.824s	0m1.053s