

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): Juan Manuel Salcedo Serrano

Grupo de prácticas: D1

Fecha de entrega: 30/03/2017

Fecha evaluación en clase: 31/03/2017

#### 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:** 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 n.º 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:** código fuente `sectionsModificado.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main(int argc, char ** 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:** código fuente `singleModificado.c`

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

```

#include <omp.h>
main() {
    int n = 9, i, a, b[n];
    for (i=0; i<n; i++){
        b[i] = -1;
    }
    #pragma omp parallel
    {
        #pragma omp single
        {
            printf("Introduce valor de inicialización a: ");
            scanf("%d", &a);
            printf("Single ejecutada por el thread %d\n",
omp_get_thread_num());
        }

        #pragma omp for
        for (i=0; i<n; i++){
            b[i] = a;
        }
        #pragma omp single
        {
            printf("Single nuevo 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:**

```

odeclas@ei142158:~/Escritorio/Home/2Cuat/AC/BP1$ gcc -O2 -fopenmp -o single single.c
odeclas@ei142158:~/Escritorio/Home/2Cuat/AC/BP1$ export OMP_DYNAMIC=FALSE
odeclas@ei142158:~/Escritorio/Home/2Cuat/AC/BP1$ export OMP_NUM_THREADS=8
odeclas@ei142158:~/Escritorio/Home/2Cuat/AC/BP1$ ./single
Introduce valor de inicialización a: 9
Single ejecutada por el thread 0
Single nuevo ejecutada por el thread 3
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
odeclas@ei142158:~/Escritorio/Home/2Cuat/AC/BP1$

```

4. 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 <stdlib.h>
#include <omp.h>
int main(int argc, char ** 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("\nImprimidos por el thread: %d\n",omp_g et_thread_num());
    for(i=0;i<n;i++){
        printf("b[%d] =%d\t",i,b[i]);
    }
}
}
}

```

#### CAPTURAS DE PANTALLA:

```

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP1$ gcc -fopenmp -O2 -o single2 singleModificado2.c
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP1$ ./single2
Introduce valor de inicializacion a: 3
Single ejecutada por el thread 2

Imprimidos por el thread: 0
b[0] =3 b[1] =3 b[2] =3 b[3] =3 b[4] =3 b[5] =3 b[6] =3 b[7] =3 b[8] =3
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP1$

```

**RESPUESTA A LA PREGUNTA:** Usando la directiva master se ejecuta siempre con la hebra 0

- . ¿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:** Sin barrier las hebras no se esperan entre si en ese punto, por tanto las sumas parciales no tienen por qué ser correctas.

### Resto de ejercicios

- . 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 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.

#### CAPTURAS DE PANTALLA:

```

odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP1$ time ./listado1 10000000
tiempo(seg.):0.029717817 / 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 0m0.052s
sys 0m0.044s
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP1$ time ./listado1 10000000
tiempo(seg.):0.030936431 / 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 0m0.052s
sys 0m0.048s
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP1$ time ./listado1 10000000
tiempo(seg.):0.030065307 / 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 0m0.099s
user 0m0.040s
sys 0m0.056s

```

## METER CAPTURA DE EJECUCIÓN EN ATCGRID

**RESPUESTA:** Se ejecuta en una única hebra, es igual.

- . 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.

**RESPUESTA:** Para 10 componentes:

$$\text{MIPS} = (5+6*10)/(0.000002420*10^6) = 26.8595041322$$

$$\text{MFLOPS} = (10)/(0.000002420*10^6) = 4.13223140496$$

Para 10000000 componentes:

$$\text{MIPS} = (5+6*10000000)/(0.074507170*10^6) = 805.291692061$$

$$\text{MFLOPS} = (10000000)/(0.074507170*10^6) = 134.215270826$$

**RESPUESTA:**

código ensamblador generado de la parte de la suma de vectores

```
call    clock_gettime
xorl    %eax, %eax
.p2align 4,,10
.p2align 3
.L7:
movsd   v1(%rax), %xmm0
addq    $8, %rax
addsd   v2-8(%rax), %xmm0
movsd   %xmm0, v3-8(%rax)
cmpq    %rbx, %rax
jne     .L7
.L6:
leaq    16(%rsp), %rsi
xorl    %edi, %edi
call    clock_gettime
```

- . 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

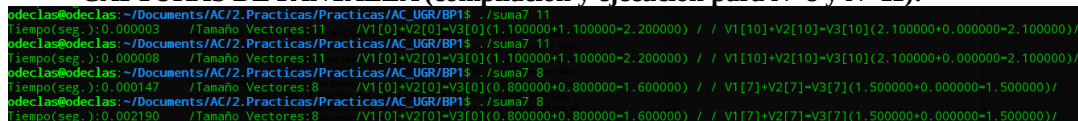
menos, el primer y último componente de  $v_1$ ,  $v_2$  y  $v_3$  (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

**RESPUESTA:** código fuente implementado

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define MAX 33554432
double v1[MAX],v2[MAX],v3[MAX];
int main(int argc, char ** argv){
    int i;
    double inicio, final;
    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;
        #pragma omp single
        {
            inicio = omp_get_wtime();
        }
        #pragma omp for
        for(i=0; i< N; i++){
            v3[i] = v1[i] + v2[i];
        }
        #pragma omp single
        {
            final = omp_get_wtime();
        }
    }
    printf("Tiempo(seg.):%f\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", final-
inicio,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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



Las capturas de pantalla muestran la ejecución del programa en un terminal. Se ven dos líneas de comandos y su salida correspondiente. La primera línea es para N=8 y la segunda para N=11. Ambas muestran el tiempo de ejecución en segundos y la suma de los componentes de los vectores.

1. 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,  $v_3$ , 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  $v1$ ,  $v2$  y  $v3$  (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

**RESPUESTA:** código fuente implementado

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define MAX 33554432
double v1[MAX],v2[MAX],v3[MAX];
int main(int argc, char ** argv){
    int a, b, c, d, i;
    double inicio, final;
    unsigned int N = atoi(argv[1]);
    if(N > MAX){
        N=MAX;
    }

    #pragma omp parallel
    {
        #pragma omp sections
        {
            #pragma omp section
            {
                for(a=0;a < (N/4);a++){
                    v1[a] =N*0.1+a*0.1;
                }v2[a] = N*0.1-a*0.1;
            }
            #pragma omp section
            {
                for(b=(N/4);b <(N/4)*2;b++){
                    v1[b] =N*0.1+b*0.1;
                }v2[b] = N*0.1-b*0.1;
            }
            #pragma omp section
            {
                for(c=(N/4)*2;c <(N/4)*3;c++){
                    v1[c] =N*0.1+c*0.1;
                }
                v2[c] = N*0.1-c*0.1;
            }
            #pragma omp section
            {
                for(d=(N/4)*3;d < N;d++){
                    v1[d] =N*0.1+d*0.1;
                }v2[d] = N*0.1-d*0.1;
            }
        }
        #pragma omp single
        {
            inicio = omp_get_wtime();
        }
        #pragma omp sections
        {
            #pragma omp section
            {
                for(a=0;a < (N/4);a++){
                    v3[a] =v1[a] + v2[a];
                }
            }
        }
    }
}
```

```

    }
    #pragma omp section
    {
        for(b=(N/4);b <(N/4)*2;b++){
            v3[b] =v1[b] + v2[b];
        }
    }
    #pragma omp section
    {
        for(c=(N/4)*2;c <(N/4)*3;c++){
            v3[c] =v1[c] + v2[c];
        }
    }
    #pragma omp section
    {
        for(d=(N/4)*3;d < N;d++){
            v3[d] =v1[d] + v2[d];
        }
    }
}
#pragma omp single
{
    final = omp_get_wtime();
}
}
if(N < 10){
    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.):%f\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",final-
inicio,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-1],v3[N-1]);
}
}
}

```

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

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

- . ¿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 la versión del ejercicio 7 se usa una directiva pragma omp for por tanto se pueden usar tantos threads como pueda el ordenador/sistema. En el ejercicio 8 se han usado 4 pragma omp section, por tanto se usarían 4 threads como máximo.