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): Cristina María Garrido López
Grupo de prácticas: A1
Fecha de entrega:
Fecha evaluación en clase:

[-RECORDATORIO, quitar todo este texto en rojo del cuaderno definitivo–
1. COMENTARIOS
1) **Esta plantilla no sustituye al guion de prácticas, se ha preparado para ahorrarles trabajo.** Las preguntas de esta plantilla se han extraído del **guion** de prácticas de programación paralela, si tiene dudas sobre el enunciado consulte primero el **guion**.
2) Este cuaderno de prácticas se utilizará para asignarle una puntuación durante la evaluación continua de prácticas y también lo utilizará como material de estudio y repaso para preparar el examen de prácticas escrito. Luego redáctelo con cuidado, y sea ordenado y claro.
3) No use máquinas virtuales.

2. NORMAS SOBRE EL USO DE LA PLANTILLA
1) Usar **interlineado SENCILLO**.
2) Respetar los tipos de letra y tamaños indicados:
- Calibri-11 o Liberation Serif-11 para el texto
- **Courier New-10 o Liberation Mono-10 para nombres de fichero, comandos, variables de entorno, etc., cuando se usan en el texto.**
- **Courier New o Liberation Mono de tamaño 8 o 9 para el código fuente en los listados de código fuente.**
- Formatee el código fuente de los listados para que sea legible, limpio y claro. Consulte, como ejemplo, los Listados 1 y 2 del guion (tabule, comente, …)
3) Insertar las capturas de pantalla donde se pidan y donde se considere oportuno
Recuerde que debe adjuntar al zip de entrega, el pdf de este fichero, todos los ficheros con código fuente implementados/utilizados y el resto de ficheros que haya implementado/utilizado (scripts, hojas de cálculo, etc.), lea la Sección 1.4 del guion]

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`

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#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
bucle\n",omp_get_num_threads(),i);

return(0);
}
```

RESPUESTA: código fuente `sectionsModificado.c`

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#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`

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#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 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:

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`

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#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 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:

RESPUESTA A LA PREGUNTA: La hebra master siempre es la misma.

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: Porque si no la llevase, las hebras no se esperarían, ya que la directiva master no lleva barreras implícitas y podrían estar imprimiéndose datos incorrectos.

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,…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. No, porque se pueden estar ejecutando otros procesos al mismo tiempo, por lo tanto tarda más.

   CAPTURAS DE PANTALLA:

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

RESPUESTA: MIPS(10)=(8+23*10)/0.000000225*10^6=1057,7778 y MIPS2(10m)=(8+23*10000000)/0.088006085*10^6=2613,4557

MFLOPS(10)=4*10/ 0.000000225*10^6=177,7778 y MFLOPS(10m)=4*10000000/0.088006085*10^6=454,514

código ensamblador generado de la parte de la suma de vectores

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */
            call        clock_gettime
            movl        $0, -104(%rbp)
            jmp         .L11
.L12:
            movl        -104(%rbp), %eax
            cltq
            leaq        0(,%rax,8), %rdx
            movq        -80(%rbp), %rax
            addq        %rax, %rdx
            movl        -104(%rbp), %eax
            cltq
            leaq        0(,%rax,8), %rcx
            movq        -96(%rbp), %rax
            addq        %rcx, %rax
            movsd       (%rax), %xmm1
            movl        -104(%rbp), %eax
            cltq
            leaq        0(,%rax,8), %rcx
            movq        -88(%rbp), %rax
            addq        %rcx, %rax
            movsd       (%rax), %xmm0
            addsd       %xmm0, %xmm1
            movq        %xmm1, %rax
            movq        %rax, (%rdx)
            addl        $1, -104(%rbp)
.L11:
            movl        -104(%rbp), %eax
            cmpl        -100(%rbp), %eax
            jb          .L12
            leaq        -48(%rbp), %rax
            movq        %rax, %rsi
            movl        $0, %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

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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


int main(int argc, char** argv){
int i;
             if (argc<2){
                          printf("Faltan nº componentes del vector\n");
                          exit(-1);
             }
             double tm,tm1,tm2;
             unsigned int N = atoi(argv[1]);


             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);
             }
                          #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 master
                          tm1=omp_get_wtime();
                          #pragma omp barrier
                          #pragma omp for //calculando suma
                          for(i=0; i<N; i++)
                                       v3[i] = v1[i] + v2[i];
                          #pragma omp master
                          tm2=omp_get_wtime();
                          #pragma omp barrier
                          }




             tm=(tm2-tm1);
             printf("Tiempo(seg.):%f\t / Tamaño Vectores:%u\n",tm,N);
             for(i=0;i<N;i++)
                          printf("%f\t",v3[i]);
```

```
                free(v1);
                free(v2);
                free(v3);


                return 0;
}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

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: código fuente implementado

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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


int main(int argc, char** argv){
                int i,k,l,j;
                if (argc<2){
                                printf("Faltan nº componentes del vector\n");
                                exit(-1);
                }
                double tm,tm1,tm2;
                unsigned int N = atoi(argv[1]);


                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);
```

```
            }


                        #pragma omp parallel
                        {
                                    #pragma omp sections
                                    {
                                                #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;
                                                }
                                                #pragma omp section
                                                for(j=N/4;j<N/2;j++){
                                                            v1[j] =
N*0.1+j*0.1;
                                                            v2[j] =
N*0.1-j*0.1;
                                                }
                                                #pragma omp section
                                                for(k=2*N/4;k<3*N/4;k++)
{
                                                            v1[k] =
N*0.1+k*0.1;
                                                            v2[k] =
N*0.1-k*0.1;
                                                }
                                                #pragma omp section
                                                for(l=3*N/4; l<N;l++){
                                                            v1[l] =
N*0.1+l*0.1;
                                                            v2[l] =
N*0.1-l*0.1;
                                                }
                                    }


                                    #pragma omp single
                                    tm1=omp_get_wtime();
                                    #pragma omp sections
                                    {


                                                #pragma omp section
                                                for(i=0;i<N/4;i++){
                                                            v3[i] =
v1[i] + v2[i];
                                                }
                                                #pragma omp section
                                                for(j=N/4;j<N/2;j++){
                                                            v3[j] =
v1[j] + v2[j];
                                                }
                                                #pragma omp section
                                                for(k=2*N/4;k<3*N/4;k++)
{
                                                            v3[k] =
v1[k] + v2[k];
                                                }
                                                #pragma omp section
```

```
                                                    for(l=3*N/4; l<N;l++){
                                                                v3[l] =
v1[l] + v2[l];
                                                    }
                                            }
                                            #pragma omp single
                                            tm2=omp_get_wtime();
                                }


                tm=(tm2-tm1);
                printf("Tiempo(seg.):%f\t / Tamaño Vectores:%u\n",tm,N);
                for(i=0;i<N;i++)
                                printf("%f\t",v3[i]);


                free(v1);
                free(v2);
                free(v3);


                return 0;
}
```

(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 se necesitan n threads donde n es el tamaño del vector. En el ejercicio 8 pueden estar ejecutando 4 threads a la vez como máximo.

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. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

    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	0.000051056	0.000170	0.002688
32768	0.000152752	0.000053	0.002060
65536	0.000225245	0.002855	0.002769
131072	0.000503669	0.002703	0.000188
262144	0.002015313	0.005012	0.002604
524288	0.002325959	0.004670	0.003509
1048576	0.003708519	0.008328	0.004799
2097152	0.009091187	0.010137	0.012390
4194304	0.020020253	0.018915	0.018980
8388608	0.034794909	0.033031	0.034020
16777216	0.066886933	0.066337	0.068005
33554432	0.116385514	0.134620	0.133580
67108864	0.306587726	0.265901	0.268401

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/core			Tiempo paralelo/versión `for` n Threads/cores		
	Elapsed	***CPU-user***	***CPU- sys***	***Elapsed***	***CPU-user***	***CPU- sys***
65536	0,002s	0,000s	0,000s	0,294s	0,076s	0,116s
131072	0,006s	0,004s	0,000s	0,561s	0,092s	0,216s
262144	0,009s	0,000s	0,008s	1,077s	0,172s	0,404s
524288	0,013s	0,012s	0,000s	2,102s	0,264s	0,840s
1048576	0,038s	0,028s	0,008s	4,171s	0,608s	1,592s
2097152	0,030s	0,028s	0,000s	8,515s	1,228s	3,228s
4194304	0,095s	0,088s	0,004s	16,674s	2,464s	6,212s
8388608	0,147s	0,124s	0,020s	34,282s	5,200s	12,668s
16777216	0,231s	0,176s	0,052s	1m18,153s	12,272s	28,636s
33554432	0,540s	0,452s	0,088s	2m37,726s	24,364s	58,016s
67108864	1,288s	1,096s	0,188s	5m18,337s	47,568s	1m57,816s