Grai2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 2. Programación paralela II: Cláusulas 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. ¿Qué ocurre si en el ejemplo del seminario `shared-clause.c` se añade a la directiva `parallel` la cláusula `default(none)`? (añada una captura de pantalla que muestre lo que ocurre) **(b)** Resuelva el problema generado sin eliminar `default(none)`.Añada el código con la modificación al cuaderno de prácticas.

RESPUESTA: Da un error de compilación

CÓDIGO FUENTE: `shared-clauseModificado.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 = 7;
            int a[n];
```

```
                for (i=0; i<n; i++)
                                a[i] = i+1;
                #pragma omp parallel default(none)
                {int n=7,a[n];
                #pragma omp for
                for (i=0; i<n; i++) a[i] += i;
                                printf("Después de parallel for:\n");
                }
                for (i=0; i<n; i++)
                                printf("a[%d] = %d\n",i,a[i]);

}
```

CAPTURAS DE PANTALLA:

2. ¿Qué ocurre si en `private-clause.c` se inicializa la variable `suma` fuera de la construcción `parallel` en lugar de dentro? (inicialice suma a un valor distinto de 0 dentro y fuera de `parallel`) Razone su respuesta. Añada el código con la modificación al cuaderno de prácticas.

RESPUESTA: Si se inicializa la variable suma fuera de la construcción parallel la suma final será un valor basura, ya que la cláusula private hace que se reinicien todas las variables de manera que esta variable no está inicializada.

CÓDIGO FUENTE: `private-clauseModificado.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 = 7;
                int a[n], suma=7;
                for (i=0; i<n; i++)
                                a[i] = i;
                #pragma omp parallel private(suma)
                {
                                suma=9;
                                #pragma omp for
                                for (i=0; i<n; i++){
                                                suma = suma + a[i];
                                                printf("thread %d suma a[%d] / ",
omp_get_thread_num(), i);
                                }
                                printf(
                                "\n* thread %d suma= %d", omp_get_thread_num(),
suma);
                }
                printf("\n");
}
```

CAPTURAS DE PANTALLA:

3. ¿Qué ocurre si en `private-clause.c` se elimina la cláusula `private(suma)`? ¿A qué cree que es debido?

RESPUESTA: Si se elimina la cláusula private(suma), la variable suma no será privada dentro del parallel, de manera que dentro de la directiva no hace falta inicializarla ni sacar su valor antes de que esta termine.

CÓDIGO FUENTE: `private-clauseModificado3.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 = 7;
             int a[n], suma;
             for (i=0; i<n; i++)
                          a[i] = i;
             #pragma omp parallel
             {
                          suma=0;
                          #pragma omp for
                          for (i=0; i<n; i++){
                                       suma = suma + a[i];
                                       printf("thread %d suma a[%d] / ",
omp_get_thread_num(), i);
                          }
                          printf(
                          "\n* thread %d suma= %d", omp_get_thread_num(),
suma);
             }
             printf("\n");
}
```

CAPTURAS DE PANTALLA:

4. En la ejecución de `firstlastprivate.c` de la pag. 21 del seminario se imprime un 6 fuera de la región `parallel`. ¿El código imprime siempre 6 fuera de la región `parallel`? Razone su respuesta.

RESPUESTA: Sí, ya que al utilizar la cláusula last-private se almacena el último valor que se le dio, y siendo n=7, la última iteración es siempre 6.

CAPTURAS DE PANTALLA:

5. ¿Qué ocurre si en `copyprivate-clause.c` se elimina la cláusula `copyprivate(a)` en la directiva `single`? ¿A qué cree que es debido?

RESPUESTA: No se copia el valor de “a” a las variables privadas del mismo nombre, de manera que hay algunos datos basura. Sin embargo, hay tres valores con el esperado, esto se debe a que esos valores se han ejecutado con la hebra que ha ejecutado el single.

CÓDIGO FUENTE: `copyprivate-clauseModificado.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, b[n];
                        for (i=0; i<n; i++) b[i] = -1;
                                    #pragma omp parallel
                                    { int a;
                                                #pragma omp single
//copyprivate(a)
                                                {

            printf("\nIntroduce valor de inicialización a: ");
                                                                    scanf("%d",
&a );

            printf("\nSingle ejecutada por el thread %d\n",

            omp_get_thread_num());
                                                }
                                    #pragma omp for
                                    for (i=0; i<n; i++) b[i] = a;
                                    }
                        printf("Depués de la región parallel:\n");
                        for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
                                    printf("\n");
}
```

CAPTURAS DE PANTALLA:

6. En el ejemplo `reduction-clause.c` sustituya `suma=0` por `suma=10`. ¿Qué resultado se imprime ahora? Justifique el resultado

RESPUESTA: El resultado que se obtiene es el mismo + 10, ya que la cláusula reducción acumula cada una de las hebras añadiéndoles la inicialización.

CÓDIGO FUENTE: `reduction-clauseModificado.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=20, a[n],suma=10;
                        if(argc < 2) {
                                    fprintf(stderr,"Falta iteraciones\n");
                                    exit(-1);
                        }
                        n = atoi(argv[1]); if (n>20) {n=20; printf("n=
%d",n);}
                        for (i=0; i<n; i++) a[i] = i;
                                    #pragma omp parallel for
```

```
reduction(+:suma)
                                        for (i=0; i<n; i++) suma += a[i];

                                                    printf("Tras 'parallel'
suma=%d\n",suma);
}
```

CAPTURAS DE PANTALLA:

7. En el ejemplo `reduction-clause.c`, elimine `for` de `#pragma omp parallel for reduction(+:suma)` y haga las modificaciones necesarias para que se siga realizando la suma de los componentes del vector a en paralelo sin usar directivas de trabajo compartido .

RESPUESTA:

CÓDIGO FUENTE: `reduction-clauseModificado7.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=20, a[n],suma=0;
                            if(argc < 2) {
                                        fprintf(stderr,"Falta iteraciones\n");
                                        exit(-1);
                            }
                            n = atoi(argv[1]); if (n>20) {n=20; printf("n=
%d",n);}
                            for (i=0; i<n; i++) a[i] = i;
                                        #pragma omp parallel private(i)
reduction(+:suma)
                                                    for
(i=omp_get_thread_num()*n/omp_get_num_threads(); i<(omp_get_thread_num()
+1)*n/omp_get_num_threads(); i++) suma += a[i];

                                                    printf("Tras 'parallel'
suma=%d\n",suma);
}
```

CAPTURAS DE PANTALLA:

Resto de ejercicios

8. Implementar un programa secuencial en C que calcule el producto de una matriz cuadrada, M, por un vector, v1 (implemente una versión para variables globales y otra para variables dinámicas, use una de estas versiones en los siguientes ejercicios):

$$v2 = M \bullet v1;\ v2(i) = \sum_{k=0}^{N-1} M(i,k) \bullet v(k),\ i = 0,\ldots N-1$$

NOTAS: (1) el número de filas /columnas N de la matriz deben ser argumentos de entrada al programa; (2) se debe inicializar la matriz y el vector 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 calcula el producto

matriz vector y, al menos, el primer y último componente del resultado (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

CÓDIGO FUENTE: `pmv-secuencial.c`

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

DINÁMICOS

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

int main(int argc, char ** argv)
{
int i,j,N;
            if(argc < 2) {
                                    fprintf(stderr,"Falta componentes\n");
                                    exit(-1);
            }
            N=atoi(argv[1]);
            int* V1 = (int*) malloc(N*sizeof(int));// malloc necesita el
tamaño en bytes
            int* V2 = (int*) malloc(N*sizeof(int)); //si no hay espacio
suficiente malloc devuelve NULL
            int ** M = (int**) malloc(N*sizeof(int*));
            for(i=0; i<N; i++)
                        M[i] = (int*) malloc(N*sizeof(int));

            for (i=0; i<N; i++) V1[i] = i;

            for(i=0; i<N; i++)
                        for(j=0; j<N; j++)
                                    M[i][j]=j;

            for(i=0; i<N; i++)
                        for(j=0; j<N; j++)
                                    V2[i]+=M[i][j]*V1[j];

            for(i=0; i<N; i++)
                        printf("%d\n",V2[i]);


            free(V1);
            free(V2);
            for(i=0; i<N; i++)
                        free(M[i]);
            free(M);
}
GLOBALES
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
main(int argc, char **argv) {
            int i,j,N, V1[N], V2[N], M[N][N];
```

```
            if(argc < 2) {
                                    fprintf(stderr,"Falta
componentes\n");
                                    exit(-1);
            }
            N=atoi(argv[1]);


            for (i=0; i<N; i++) V1[i] = i;

            for(i=0; i<N; i++)
                        for(j=0; j<N; j++)
                                    M[i][j]=j;

            for(i=0; i<N; i++)
                        for(j=0; j<N; j++)
                                    V2[i]+=M[i][j]*V1[j];

            for(i=0; i<N; i++)
                        printf("%d\n",V2[i]);

}
```

CAPTURAS DE PANTALLA:

9. Implementar en paralelo el producto matriz por vector con OpenMP a partir del código escrito en el ejercicio anterior usando la directiva `for` . Debe implementar dos versiones del código (consulte la lección 5/Tema 2):

    a. una primera que paralelice el bucle que recorre las filas de la matriz y

    b. una segunda que paralelice el bucle que recorre las columnas.

    Use las directivas que estime oportunas y las cláusulas que sean necesarias **excepto la cláusula `reduction`**. Se debe paralelizar también la inicialización de las matrices. Respecto a este ejercicio:

    - Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
    - Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).

    NOTAS: (1) el número de filas /columnas N de la matriz deben ser argumentos de entrada; (2) se debe inicializar la matriz y el vector 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 que calcula el producto matriz vector y, al menos, el primer y último componente del resultado (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

CÓDIGO FUENTE : `pmv-OpenMP-a.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)
{
…
}
```

CÓDIGO FUENTE: pmv-OpenMP-b.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,j,N,m;
            if(argc < 2) {
                                    fprintf(stderr,"Falta componentes\n");
                                    exit(-1);
            }
            N=atoi(argv[1]);
                        int* V1 = (int*) malloc(N*sizeof(int));// malloc
necesita el tamaño en bytes
            int* V2 = (int*) malloc(N*sizeof(int)); //si no hay espacio
suficiente malloc devuelve NULL
            int ** M = (int**) malloc(N*sizeof(int*));
            for(i=0; i<N; i++)
                        M[i] = (int*) malloc(N*sizeof(int));

            for (i=0; i<N; i++) V1[i] = i;
            for (i=0; i<N; i++) V2[i] = 0;

            for(i=0; i<N; i++)
                        for(j=0; j<N; j++)
                                    M[i][j]=j;
            #pragma omp parallel for private(m) private(j)
            for(i=0; i<N; i++){
                        for(j=0; j<N; j++){
                                    m+=M[i][j]*V1[j];

                        }V2[i]=m;
            }

            for(i=0; i<N; i++)
                        printf("%d\n",V2[i]);

            free(V1);
            free(V2);
            for(i=0; i<N; i++)
                        free(M[i]);
            free(M);
}
```

RESPUESTA:

CAPTURAS DE PANTALLA:

10. A partir de la segunda versión de código paralelo desarrollado en el ejercicio anterior, implementar una versión paralela del producto matriz por vector con OpenMP que use para comunicación/sincronización la cláusula `reduction`. Respecto a este ejercicio:
    - Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
    - Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).

    CÓDIGO FUENTE: `pmv-OpenmMP-reduction.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)
{
…
}
```

RESPUESTA:

CAPTURAS DE PANTALLA:

11. 11. Ayudándose de una hoja de cálculo (recuerde que en las aulas está instalado OpenOffice) realice una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en el PC local del mejor código paralelo de los tres implementados en los ejercicios anteriores para dos tamaños (N) distintos (consulte la Lección 6/Tema 2). Usar –O2 al compilar. Justificar por qué el código escogido es el mejor. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

TABLA Y GRÁFICA (por *ejemplo* para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N-: algúno del orden de cientos de miles):

COMENTARIOS SOBRE LOS RESULTADOS: