

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): David Gil Bautista

Grupo de prácticas: C1

Fecha de entrega:

Fecha evaluación en clase:

Ejercicios basados en los ejemplos del seminario práctico

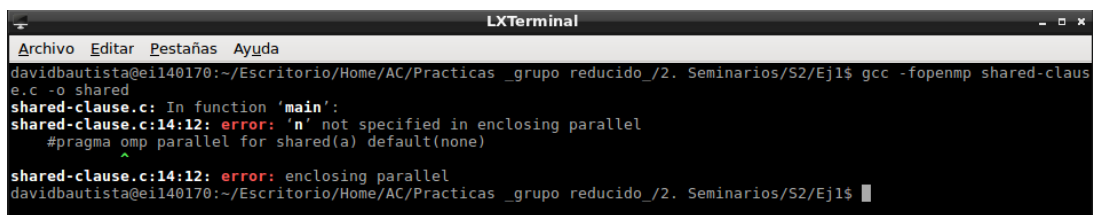
1. ¿Qué ocurre si en el ejemplo del seminario `shared-clause.c` se añade a la directiva `parallel` la cláusula `default(none)`? (añada una captura de pantalla que muestre lo que ocurre) **(b)** Resuelva el problema generado sin eliminar `default(none)`. Añada el código con la modificación al cuaderno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

RESPUESTA: Al añadir el `default(none)` todas las hebras trabajan sobre el mismo `n` pero añadiendo este `n` al `shared` las hebras lo comparten y no se sobrescribe.

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 for shared(a,n) default(none)  
    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]);  
  
    return 0;  
}
```

CAPTURAS DE PANTALLA:



```
LXTerminal  
Archivo Editar Pestañas Ayuda  
davidbautista@eil40170:~/Escritorio/Home/AC/Practicas_grupo reducido_/2. Seminarios/S2/Ej1$ gcc -fopenmp shared-clause.c -o shared  
shared-clause.c: In function 'main':  
shared-clause.c:14:12: error: 'n' not specified in enclosing parallel  
    #pragma omp parallel for shared(a) default(none)  
    ^  
shared-clause.c:14:12: error: enclosing parallel  
davidbautista@eil40170:~/Escritorio/Home/AC/Practicas_grupo reducido_/2. Seminarios/S2/Ej1$
```

```

LXTerminal
Archivo Editar Pestañas Ayuda
davidbautista@eil40170:~/Escritorio/Home/AC/Practicas_grupo reducido_/2. Seminarios/S2/Ej1$ gcc -fopenmp shared-clause-modificado.c -o sharedM
davidbautista@eil40170:~/Escritorio/Home/AC/Practicas_grupo reducido_/2. Seminarios/S2/Ej1$ ./sharedM
Después de parallel for:
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 7
a[4] = 9
a[5] = 11
a[6] = 13
davidbautista@eil40170:~/Escritorio/Home/AC/Practicas_grupo reducido_/2. Seminarios/S2/Ej1$ ./sharedM
Después de parallel for:
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 7
a[4] = 9
a[5] = 11
a[6] = 13
davidbautista@eil40170:~/Escritorio/Home/AC/Practicas_grupo reducido_/2. Seminarios/S2/Ej1$

```

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: Al sacar la `suma` fuera el `private` crea otra variable `suma` con otro valor distinto.

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;

    for (i=0; i<n; i++)
        a[i] = i;

    suma=5;

#pragma omp parallel private(suma)
    {

#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");
}

```

```
    return 0;
}
```

CAPTURAS DE PANTALLA:

Suma fuera:

```
LXTerminal
Archivo Editar Pestañas Ayuda
davidbautista@eil40170:~/Escritorio/Home/AC/Practicas_grupo reducido_/2. Seminarios/S2/ej2$ gcc -fopenmp private-clause.c -o private
davidbautista@eil40170:~/Escritorio/Home/AC/Practicas_grupo reducido_/2. Seminarios/S2/ej2$ ./private
thread 3 suma a[6] / thread 2 suma a[4] / thread 2 suma a[5] / thread 1 suma a[2] / thread 1 suma a[3] /
thread 0 suma a[0] / thread 0 suma a[1] /
* thread 2 suma= 14
* thread 3 suma= 11
* thread 0 suma= 6
* thread 1 suma= 10
davidbautista@eil40170:~/Escritorio/Home/AC/Practicas_grupo reducido_/2. Seminarios/S2/ej2$
```

Suma dentro:

```
LXTerminal
Archivo Editar Pestañas Ayuda
davidbautista@eil40170:~/Escritorio/Home/AC/Practicas_grupo reducido_/2. Seminarios/S2/ej2$ gcc -fopenmp private-clause.c -o private
davidbautista@eil40170:~/Escritorio/Home/AC/Practicas_grupo reducido_/2. Seminarios/S2/ej2$ ./private
thread 2 suma a[4] / thread 2 suma a[5] / thread 3 suma a[6] / thread 0 suma a[0] / thread 0 suma a[1] /
thread 1 suma a[2] / thread 1 suma a[3] /
* thread 0 suma= 1
* thread 2 suma= 9
* thread 1 suma= 5
* thread 3 suma= 6
davidbautista@eil40170:~/Escritorio/Home/AC/Practicas_grupo reducido_/2. Seminarios/S2/ej2$
```

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

RESPUESTA: Cuando quitas el `private` todos trabajan sobre la misma variable por lo que las distintas hebras darán el mismo resultado.

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=5;
    }
}
```

```

#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:

```

saytes@TRON:~/Dropbox/Facultad/2ºCurso/Segundo Cuatrimestre/AC/Práctica 2$ ./ejercicio3
thread 0 suma a[0]/thread 2 suma a[2]/thread 5 suma a[5]/thread 6 suma a[6]/thread 4 suma a[4]/thread 1 suma a[1]/thread 3 suma a[3]/
* thread 4 suma= 8
* thread 2 suma= 8
* thread 3 suma= 8
* thread 0 suma= 8
* thread 1 suma= 8
* thread 5 suma= 8
* thread 7 suma= 8
* thread 6 suma= 8

```

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: Si, porque es el valor de la última hebra que finaliza el proceso.

CAPTURAS DE PANTALLA:

```

saytes@TRON:~/Dropbox/Facultad/2ºCurso/Segundo Cuatrimestre/AC/Seminario2$ ./firstlastprivate-clause
thread 0 suma a[0] suma=0
thread 4 suma a[4] suma=4
thread 5 suma a[5] suma=5
thread 3 suma a[3] suma=3
thread 1 suma a[1] suma=1
thread 6 suma a[6] suma=6
thread 2 suma a[2] suma=2
Fuera de la construcción parallel suma=6
saytes@TRON:~/Dropbox/Facultad/2ºCurso/Segundo Cuatrimestre/AC/Seminario2$ ./firstlastprivate-clause
thread 1 suma a[1] suma=1
thread 0 suma a[0] suma=0
thread 4 suma a[4] suma=4
thread 5 suma a[5] suma=5
thread 6 suma a[6] suma=6
thread 3 suma a[3] suma=3
thread 2 suma a[2] suma=2
Fuera de la construcción parallel suma=6

```

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: Al no poner la cláusula no se inicializan los valores en las hebras.

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("Después 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:

```

saytes@TRON:~/Dropbox/Facultad/2ºCurso/Segundo Cuatrimestre/AC/Práctica 2$ ./ejercicio5

Introduce valor de inicialización a: 3

Single ejecutada por el thread 4
Después de la región parallel:
b[0] = 32766    b[1] = 32766    b[2] = 32673    b[3] = 0        b[4] = 0        b[5] = 3        b[6] = 0 b
[7] = 0 b[8] = 0
saytes@TRON:~/Dropbox/Facultad/2ºCurso/Segundo Cuatrimestre/AC/Práctica 2$

```

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

RESPUESTA:**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:**Poniendo suma = 0;**

```

saytes@TRON:~/Dropbox/Facultad/2*Curso/Segundo Cuatrimestre/AC/Práctica 2$ ./ejercicio6 3
Tras 'parallel' suma=3

```

Poniendo suma = 10;

```

saytes@TRON:~/Dropbox/Facultad/2*Curso/Segundo Cuatrimestre/AC/Práctica 2$ ./ejercicio6 3
Tras 'parallel' suma=13
saytes@TRON:~/Dropbox/Facultad/2*Curso/Segundo Cuatrimestre/AC/Práctica 2$

```

7. En el ejemplo `reduction-clause.c`, elimine `reduction()` 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: Al quitarlo la suma no se realiza de forma correcta por lo que para que se ejecute bien hay que añadir un `sections` y dividir el bucle `for`.

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 sections reduction(+:suma)
    {
        #pragma omp section
        for (i=0; i<n/2; i++)    suma += a[i];
        #pragma omp section

```

```

        for (i=n/2; i<n; i++)    suma += a[i];
    }

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

```

CAPTURAS DE PANTALLA:

```

saytes@TRON:~/Dropbox/Facultad/2ºCurso/Segundo Cuatrimestre/AC/Práctica 2$ gcc -O2 -fopenmp -o ejercicio7
ejercicio7.c
saytes@TRON:~/Dropbox/Facultad/2ºCurso/Segundo Cuatrimestre/AC/Práctica 2$ ./ejercicio7 3
Tras 'parallel' suma=3
saytes@TRON:~/Dropbox/Facultad/2ºCurso/Segundo Cuatrimestre/AC/Práctica 2$

```

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):

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

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 */

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

int main(int argc, char ** argv)
{
    ...
}

```

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):
- una primera que paralelice el bucle que recorre las filas de la matriz y
 - 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)
{
...
}
```

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. 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-: alguno del orden de cientos de miles):

COMENTARIOS SOBRE LOS RESULTADOS: