

# Arquitectura de Computadores (AC)

## Cuaderno de prácticas.

### Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): David Sánchez Jiménez

Grupo de prácticas: A3

Fecha de entrega: 19/04/2018

Fecha evaluación en clase: 20/04/2018

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:** Nos devuelve un error porque no hemos especificado el ámbito de la variable `n`, por lo que debemos añadir `n` a las variables compartidas.

**CAPTURA CÓDIGO FUENTE:** `shared-clauseModificado.c`

```
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#endif

int main() {
    int i, n = 7;
    int a[n];

    for (i = 0; i < n; i++) {
        a[i] = i + 1;
    }

    #pragma omp parallel for default(none) shared(a, n)
    for (i = 0; i < n; i++) {
        a[i] += i;
    }

    printf("Despues de parallel for:\n");

    for (i = 0; i < n; i++) {
        printf("a[%d] = %d\n", i, a[i]);
    }

    return 0;
}
```

#### CAPTURAS DE PANTALLA:

```
jue 19 abr - 21:53 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_2
@david gcc -fopenmp -O2 shared-clauseModificado.c -o shared-clauseModificado
shared-clauseModificado.c: In function 'main':
shared-clauseModificado.c:13:9: error: 'n' not specified in enclosing 'parallel'
  #pragma omp parallel for default(none) shared(a)
          ^~~
shared-clauseModificado.c:13:9: error: enclosing 'parallel'
```

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:** Se obtienen resultados de suma erróneos ya que la variable `suma` toma un valor desconocido al entrar en el `parallel`

**CAPTURA CÓDIGO FUENTE:** `private-clauseModificado.c`

```
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif

int main() {
    int i, n = 7;
    int a[n], suma;

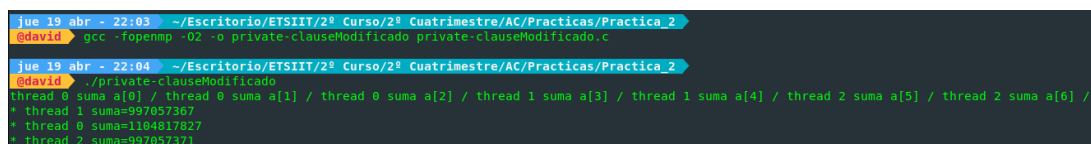
    for (i = 0; i < n; i++) {
        a[i] = i;
    }

    suma = 0;
#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");
}
```

#### CAPTURAS DE PANTALLA:



```
jue 19 abr - 22:03 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica 2
@david gcc -fopenmp -O2 -o private-clauseModificado private-clauseModificado.c

jue 19 abr - 22:04 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica 2
@david ./private-clauseModificado
thread 0 suma a[0] / thread 0 suma a[1] / thread 0 suma a[2] / thread 1 suma a[3] / thread 1 suma a[4] / thread 2 suma a[5] / thread 2 suma a[6] /
* thread 1 suma=997057367
* thread 0 suma=1104817827
* thread 2 suma=997057371
```

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

**RESPUESTA:** Obtenemos un resultado incorrecto ya que todas las hebras trabajan sobre la misma variable `suma` debido a que es definida fuera de la región `parallel` es una variable compartida.

**CAPTURA CÓDIGO FUENTE:** `private-clauseModificado3.c`

```
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif

int main() {
    int i, n = 7;
    int a[n], suma;

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

    suma = 0;
#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");
    }
}
```

**CAPTURAS DE PANTALLA:**

```
vie 20 abr - 00:08 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica 2
@david gcc -fopenmp -O2 -o private-clauseModificado3 private-clauseModificado3.c

vie 20 abr - 00:08 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica 2
@david ./private-clauseModificado3
thread 0 suma a[0] / thread 0 suma a[1] / thread 0 suma a[2] / thread 2 suma a[5] / thread 2 suma a[6] / thread 1 suma a[3] / thread 1 suma a[4] /
* thread 2 suma=21
* thread 0 suma=21
* thread 1 suma=21
```

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, debido a que `lastprivate` hace que suma se quede con el valor del ultimo thread (el 3), el cual siempre tiene valor 6.

#### CAPTURAS DE PANTALLA:

```
vie 20 abr - 00:15 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_2
@david ./firstlastprivate-clang
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 3 suma a[6] suma=6
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9
Fuera de la construccion parallel suma=6

vie 20 abr - 00:15 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_2
@david ./firstlastprivate-clang
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 3 suma a[6] suma=6
Fuera de la construccion parallel suma=6

vie 20 abr - 00:15 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_2
@david ./firstlastprivate-clang
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9
thread 3 suma a[6] suma=6
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
Fuera de la construccion parallel suma=6
```

5. ¿Qué se observa en los resultados de ejecución de `copyprivate-clause.c` cuando se elimina la cláusula `copyprivate(a)` en la directiva `single`? ¿A qué cree que es debido?

**RESPUESTA:** Se observa que la difusión del valor introducido por teclado no se hace y por tanto en vez de inicializarse todo el vector `b` con el valor introducido por teclado se inicializa solo la parte del thread que ejecutó el `single` y el resto se inicializa a un valor indeterminado.

**CAPTURA CÓDIGO FUENTE:** `copyprivate-clauseModificado.c`

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

int main() {
    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");
}
```

#### CAPTURAS DE PANTALLA:

```
vie 20 abr - 00:21 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica 2
@david gcc -fopenmp -O2 -o copyprivate-clauseModificado copyprivate-clauseModificado.c

vie 20 abr - 00:21 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica 2
@david ./copyprivate-clauseModificado

Introduce valor de inicialización a:3

Single ejecutada por el thread 1
Después de la región parallel:
b[0] = 0      b[1] = 0      b[2] = 0      b[3] = 3      b[4] = 3      b[5] = 0      b[6] = 0      b[7] = 0      b[8] = 0
```

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

**RESPUESTA:** El resultado final de la suma es 10 unidades mayor debido a que la clausula `reduction` despues de sumar todos los valores de las iteraciones se los suma al valor original de la variable `suma`. Por eso, el resultado es la suma de las iteraciones del bucle `for` y el valor original que tuviera `suma`.

**CAPTURA CÓDIGO FUENTE:** `reduction-clauseModificado.c`

```
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif

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:**

```
vie 20 abr - 00:38 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_2
@david gcc -fopenmp -O2 -o reduction-clause reduction-clause.c

vie 20 abr - 00:38 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_2
@david ./reduction-clause 10
Tras 'parallel' suma = 45

vie 20 abr - 00:38 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_2
@david gcc -fopenmp -O2 -o reduction-clauseModificado reduction-clauseModificado.c

vie 20 abr - 00:38 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_2
@david ./reduction-clauseModificado 10
Tras 'parallel' suma = 55
```

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:**

**CAPTURA CÓDIGO FUENTE:** `reduction-clauseModificado7.c`

```
int main(int argc, char const *argv[]) {
    int i, n = 20, a[n], suma_final = 0, suma_pri = 0;

    if (argc < 2) {
        fprintf(stderr, "Falta iteraciones\n");
        exit(-1);
    }

    n = atoi(argv[1]);

    if (n > 20) {
        n = 20;
        printf("n = %d\n", n);
    }

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

    #pragma omp parallel firstprivate(suma_pri)
    {
        #pragma omp for
        for (i = 0; i < n; i++) {
            suma_pri += a[i];
        }

        #pragma omp atomic
        suma_final += suma_pri;
    }

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

**CAPTURAS DE PANTALLA:**

```
vie 20 abr - 00:43 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_2
@david gcc -fopenmp -O2 -o reduction-clauseModificado7 reduction-clauseModificado7.c

vie 20 abr - 00:43 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_2
@david ./reduction-clauseModificado7 10
Tras 'parallel' suma = 45

vie 20 abr - 00:43 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_2
@david ./reduction-clauseModificado7 10
Tras 'parallel' suma = 45

vie 20 abr - 00:43 ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_2
@david ./reduction-clauseModificado7 10
Tras 'parallel' suma = 45
```



## 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).

**CAPTURA CÓDIGO FUENTE:** pmv-secuencial.c

### 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).

**CAPTURA CÓDIGO FUENTE :** pmv-OpenMP-a.c

**CAPTURA CÓDIGO FUENTE:** pmv-OpenMP-b.c

### 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).

**CAPTURA CÓDIGO FUENTE:** `pmv-OpenmMP-reduction.c`

**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 su PC 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.

**CAPTURAS DE PANTALLA (que justifique el código elegido):**

**TABLA Y GRÁFICA (por ejemplo para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N-: un N entre 30000 y 100000, y otro entre 5000 y 30000):**

**COMENTARIOS SOBRE LOS RESULTADOS:**