

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Rodrigo Raya Castellano

Grupo de prácticas: 2

Fecha de entrega:

Fecha evaluación en clase:

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:

Se produce un error que nos informa de que no se ha definido el alcance de la variable `n` cuando la directiva `default(none)` exige que todos los ámbitos los defina el programador.

Para solucionarlo, añadimos `shared(n)` de modo que `n` sea una variable compartida por todos los threads.

CÓDIGO FUENTE: `shared-clauseModificado.c`

```
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#endif
main()
{
    int i, n = 7;
    int a[n];
    for (i=0; i<n; i++)
        a[i] = i+1;
    #pragma omp parallel for shared(a) shared(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]);
}
```

CAPTURAS DE PANTALLA:

```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas$ gcc -fopenmp -O2 -o shared-clause shared-clause.c
shared-clause.c: In function 'main':
shared-clause.c:11:10: error: 'n' not specified in enclosing parallel
    #pragma omp parallel for shared(a) default(none)
    ^
shared-clause.c:11:10: error: enclosing parallel
    for (i=0; i<n; i++) a[i] += i;
```

2. ¿Qué ocurre si en `private-clause.c` se inicializa la variable `suma` fuera de la construcción `parallel` en lugar de dentro? Razone su respuesta. Añada el código con la modificación al cuaderno de prácticas.

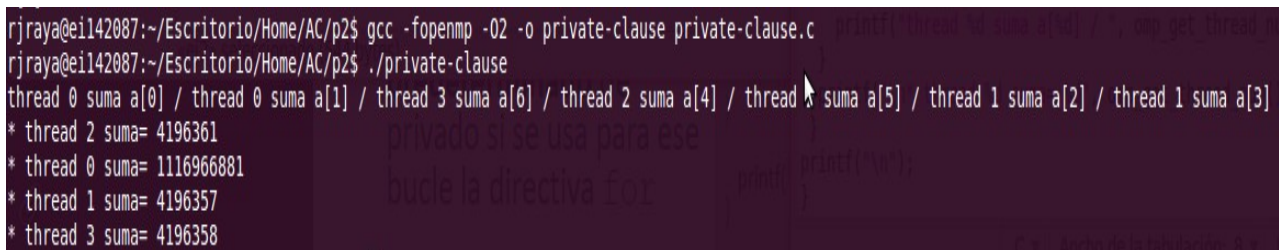
RESPUESTA:

El valor de entrada estará indefinido y las diferentes copias de la variable `suma` contienen basura, luego no se imprime el valor correcto.

CÓDIGO FUENTE: `private-clauseModificado.c`

```
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
main()
{
    int i, n = 7;
    int a[n], suma = 0;
    for (i=0; i<n; i++)
        a[i] = i;
    #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:



```
rjraya@ei142087:~/Escritorio/Home/AC/p2$ gcc -fopenmp -O2 -o private-clause private-clause.c
rjraya@ei142087:~/Escritorio/Home/AC/p2$ ./private-clause
thread 0 suma a[0] / thread 0 suma a[1] / thread 3 suma a[6] / thread 2 suma a[4] / thread 1 suma a[5] / thread 1 suma a[2] / thread 1 suma a[3]
* thread 2 suma= 4196361
* thread 0 suma= 1116966881
* thread 1 suma= 4196357
* thread 3 suma= 4196358
```

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

RESPUESTA:

Todos los threads comparten la misma variable suma y por tanto al final se imprime la misma. También observamos que variando el número de threads ,varía la suma que se obtiene.

CÓDIGO FUENTE: private-clauseModificado3.c

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

```
rjraya@eil42087:~/Escritorio/Home/AC/p2$ export OMP_NUM_THREADS=6
rjraya@eil42087:~/Escritorio/Home/AC/p2$ ./private_clause
thread 2 suma a[4] / thread 2 suma a[5] / thread 0 suma a[0] / thread 0 suma a[1] / thread 1 suma a[2] / thread 1 suma a[3] / thread 3 suma a[6]
* thread 2 suma= 3
* thread 1 suma= 3
* thread 4 suma= 3
* thread 0 suma= 3
* thread 3 suma= 3
* thread 5 suma= 3
rjraya@eil42087:~/Escritorio/Home/AC/p2$ export OMP_NUM_THREADS=8
rjraya@eil42087:~/Escritorio/Home/AC/p2$ ./private_clause
thread 3 suma a[3] / thread 2 suma a[2] / thread 0 suma a[0] / thread 6 suma a[6] / thread 5 suma a[5] / thread 1 suma a[1] / thread 4 suma a[4]
* thread 7 suma= 0
* thread 4 suma= 0
* thread 1 suma= 0
* thread 0 suma= 0
* thread 6 suma= 0
* thread 2 suma= 0
* thread 5 suma= 0
* thread 3 suma= 0
```

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:

La cláusula devuelve el valor en la última iteración del bucle. Como `a[6] = 6` devuelve 6. Si cambiara el número de iteraciones y el tamaño del array entonces cambiaría el resultado.

CAPTURAS DE PANTALLA:

```
rjraya@eil42087:~/Escritorio/Home/AC/p2$ ./firstlast-private
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 3 suma a[6] suma=6
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

Fuera de la construcción parallel suma=6
rjraya@eil42087:~/Escritorio/Home/AC/p2$ ./firstlast-private
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 construcción parallel suma=6
```

Con otro número de iteraciones:

```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas$ gcc -fopenmp -O2 -o
firstlast-private firstlast-private.c
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas$ ./firstlast-private
thread 2 suma a[6] suma=6
thread 2 suma a[7] suma=13
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 3 suma a[8] suma=8
thread 3 suma a[9] suma=17
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7
thread 1 suma a[5] suma=12

Fuera de la construcción parallel suma=17
```

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:

Algunas iteraciones (las de hebra que ejecuta el `single`) obtienen el valor correcto y las demás un valor basura. La razón es que no se les hace llegar (no se difunde) el valor de la variable leída.

CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```
#include <stdio.h>
#include <omp.h>
main() {
    int n = 9, i, b[n];
    for (i=0; i<n; i++) b[i] = -1;
    #pragma omp parallel
    { int a;
      #pragma omp single
      {
          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:

```
rjraya@eil42087:~/Escritorio/Home/AC/p2$ gcc -fopenmp -o copyprivate-clause copyprivate-clause.c
rjraya@eil42087:~/Escritorio/Home/AC/p2$ ./copyprivate-clause

Introduce valor de inicialización a: 3

Single ejecutada por el thread 0
Después de la región parallel:
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
rjraya@eil42087:~/Escritorio/Home/AC/p2$ gcc -fopenmp -o copyprivate-clause copyprivate-clause.c
rjraya@eil42087:~/Escritorio/Home/AC/p2$ ./copyprivate-clause

Introduce valor de inicialización a: 3

Single ejecutada por el thread 0
Después de la región parallel:
b[0] = 3    b[1] = 3    b[2] = 3    b[3] = 141943589    b[4] = 141943589    b[5] = 141943589    b[6] = 0    b[7] = 0    b[8] = 0
rjraya@eil42087:~/Escritorio/Home/AC/p2$
```

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

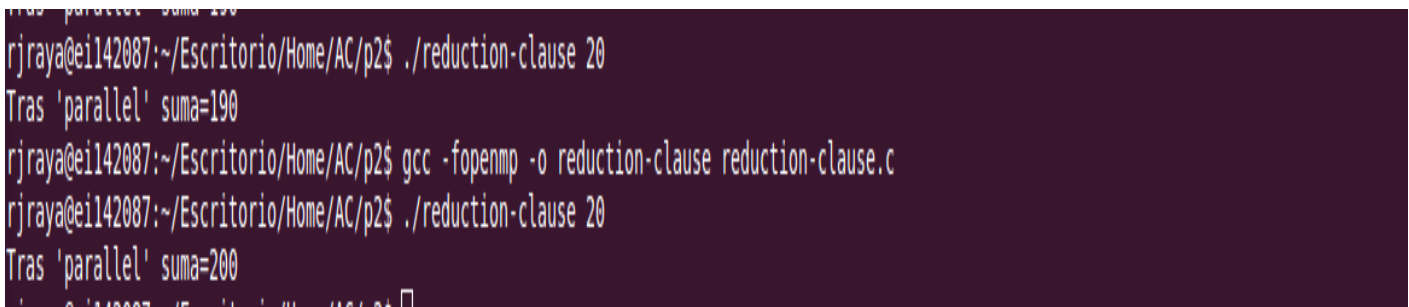
RESPUESTA:

Con 20 iteraciones y con `suma = 0` imprime 190 y con `suma = 10` imprime 200 . Esto ocurre así porque también se reduce la variable compartida.

CÓDIGO FUENTE: `reduction-clauseModificado.c`

```
#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, 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:



```
rjraya@ei142087:~/Escritorio/Home/AC/p2$ ./reduction-clause 20
Tras 'parallel' suma=190
rjraya@ei142087:~/Escritorio/Home/AC/p2$ gcc -fopenmp -o reduction-clause reduction-clause.c
rjraya@ei142087:~/Escritorio/Home/AC/p2$ ./reduction-clause 20
Tras 'parallel' suma=200
```

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.

RESPUESTA:

Lo que he hecho ha sido repartir explícitamente las iteraciones del bucle en modo round-robin.

CÓDIGO FUENTE: `reduction-clauseModificado7.c`

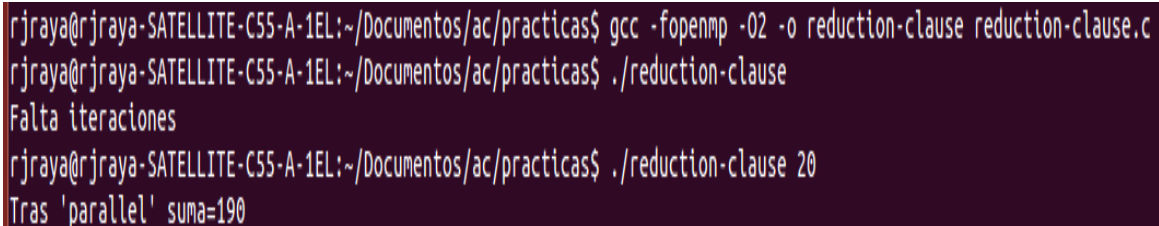
```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
main(int argc, char **argv) {
    int i, n=20, a[n], suma=0;
    if(argc < 2) {
        fprintf(stderr, "Falta iteraciones\n");
        exit(-1);
    }
    n = atoi(argv[1]); if (n>20) {n=20; printf("n=%d", n);}

    int tid = -1;

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

    #pragma omp parallel reduction(+:suma) private(tid)
    {
        int nT = omp_get_num_threads();
        tid = omp_get_thread_num();
        for (i=tid; i<n; i = i + nT){
            suma += a[i];
        }
    }
    printf("Tras 'parallel' suma=%d\n", suma);
}
```

CAPTURAS DE PANTALLA:



```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas$ gcc -fopenmp -O2 -o reduction-clause reduction-clause.c
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas$ ./reduction-clause
Falta iteraciones
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas$ ./reduction-clause 20
Tras 'parallel' suma=190
```

8. Implementar un programa secuencial en C que calcule el producto de una matriz cuadrada, M, por un vector, v1:

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

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 del aula de prácticas de los tres códigos implementados en los ejercicios anteriores para tres tamaños (N) distintos (consulte la Lección 6/Tema 2). Usar `-O2` al compilar.

TABLA Y GRÁFICA (por ejemplo para 1-4 threads PC aula, y para 1-12 threads en atcgrid, tamaños-N-: 1.000, 10.000, 100.000):

COMENTARIOS SOBRE LOS RESULTADOS: