

Grai2º curso / 2º  
cuatr.  
Grado Ing. Inform.

# Arquitectura de Computadores (AC)

## Cuaderno de prácticas.

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

Estudiante (nombre y apellidos): Valentino Lugli

Grupo de prácticas y profesor de prácticas: C1, Christian Morillas

Fecha de entrega: 26/04/21

Fecha evaluación en clase: 27/04/21

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

### Ejercicios basados en los ejemplos del seminario práctico

1. (a) Añadir la cláusula `default(none)` a la directiva `parallel` del ejemplo del seminario `shared-clause.c`? ¿Qué ocurre? ¿A qué se debe? (b) Resolver el problema generado sin eliminar `default(none)`. Incorporar el código con la modificación al cuaderno de prácticas. (Añadir capturas de pantalla que muestren lo que ocurre)

**RESPUESTA:** Lo que sucede es que se genera un fallo de compilación, se debe a que la cláusula `default(none)` indica que todas las variables que se utilicen en una construcción deben ser especificadas, exceptuando en índice `i` del bucle `for`, en este caso `n` no está siendo especificada y por lo tanto sucede el error.

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

```
main()
{
    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]);
}
```

### CAPTURAS DE PANTALLA:

```
File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer1] 2021-04-13 martes
>>make
gcc -O2 -fopenmp -o shared-clauseModificado_EXE shared-clauseModificado.c
shared-clauseModificado.c:5:1: warning: return type defaults to 'int' [-Wimplicit-int]
5 | main()
  | ^~~~~
shared-clauseModificado.c: In function 'main':
shared-clauseModificado.c:13:13: error: 'n' not specified in enclosing 'parallel'
13 |     #pragma omp parallel for shared(a) default(none)
    |     ~~~~~
shared-clauseModificado.c:13:13: error: enclosing 'parallel'
make: *** [makefile:3: shared-clauseModificado_EXE] Error 1
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer1] 2021-04-13 martes
>>|
```

2. (a) Añadir a lo necesario a `private-clause.c` para que imprima suma fuera de la región `parallel`. Inicializar suma dentro del `parallel` a un valor distinto de 0. Ejecutar varias veces el código ¿Qué imprime el código fuera del `parallel`? (mostrar lo que ocurre con una captura de pantalla) Razonar respuesta. (b) Modificar el código del apartado (a) para que se inicialice suma fuera del `parallel` en lugar de dentro ¿Qué ocurre? Comparar todo lo que imprime el código ahora con la salida en (a) (mostrar la salida con una captura de pantalla) Razonar respuesta.

**(a) RESPUESTA:**

El código imprime que la variable `suma = 0`, aunque dentro se ha inicializado al valor 16. Esto se debe a que la variable está imprimiendo basura aunque haya sido utilizada e inicializada dentro de las directiva `parallel`, lo que indica que no sufre cambios una vez que termina el paralelismo.

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

```
main()
{
    int i, n = 7;
    int a[n], suma;
    for (i=0; i<n; i++)
        a[i] = i;
    #pragma omp parallel private(suma)
    {
        suma=16;
        #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");
    printf("Suma fuera de parallel: %d\n", suma);
}
```

**CAPTURAS DE PANTALLA:**

```
File Edit View Bookmarks Settings Help
Suma fuera de parallel: 0
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer2] 2021-04-13 martes
>>make
gcc -O2 -fopenmp -o private-clauseModificado_a_EXE private-clauseModificado_a.c
private-clauseModificado_a.c:9:1: warning: return type defaults to 'int' [-Wimplicit-int]
  9 | main()
    | ^~~~~
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer2] 2021-04-13 martes
>>./private-clauseModificado_a_EXE
thread 3 suma a[3] / thread 4 suma a[4] / thread 6 suma a[6] / thread 1 suma a[1] / thread 2 suma a[2] /
thread 0 suma a[0] / thread 5 suma a[5] /
* thread 4 suma= 20
* thread 3 suma= 19
* thread 6 suma= 22
* thread 1 suma= 17
* thread 2 suma= 18
* thread 0 suma= 16
* thread 5 suma= 21
* thread 7 suma= 16
Suma fuera de parallel: 0
```

```

File Edit View Bookmarks Settings Help
* thread 7 suma= 16
Suma fuera de parallel: 0
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer2] 2021-04-13 martes
>>./private-clauseModificado_a_EXE
thread 4 suma a[4] / thread 3 suma a[3] / thread 6 suma a[6] / thread 0 suma a[0] / thread 5 suma a[5] /
thread 2 suma a[2] / thread 1 suma a[1] /
* thread 3 suma= 19
* thread 6 suma= 22
* thread 4 suma= 20
* thread 7 suma= 16
* thread 5 suma= 21
* thread 2 suma= 18
* thread 1 suma= 17
* thread 0 suma= 16
Suma fuera de parallel: 0
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer2] 2021-04-13 martes
>>./private-clauseModificado_a_EXE
thread 2 suma a[2] / thread 5 suma a[5] / thread 6 suma a[6] / thread 1 suma a[1] / thread 4 suma a[4] /
thread 3 suma a[3] / thread 0 suma a[0] /
* thread 5 suma= 21
* thread 7 suma= 16
* thread 6 suma= 22
* thread 2 suma= 18
* thread 3 suma= 19
* thread 0 suma= 16
* thread 4 suma= 20
* thread 1 suma= 17
Suma fuera de parallel: 0
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer2] 2021-04-13 martes
>>./private-clauseModificado_a_EXE
thread 5 suma a[5] / thread 4 suma a[4] / thread 3 suma a[3] / thread 2 suma a[2] / thread 0 suma a[0] /
thread 6 suma a[6] / thread 1 suma a[1] /
* thread 0 suma= 16
* thread 3 suma= 19
* thread 6 suma= 22
* thread 7 suma= 16
* thread 2 suma= 18
* thread 4 suma= 20
* thread 5 suma= 21
* thread 1 suma= 17
Suma fuera de parallel: 0

```

**(b) RESPUESTA:**

Ahora, internamente dentro de la cláusula `parallel` la variable `suma` no está inicializada porque la cláusula `private` realiza una copia de la variable pero no su contenido, por ello imprime basura en paralelo pero imprime 16 al final, fuera del paralelismo.

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

```

main()
{
    int i, n = 7;
    int a[n], suma;
    for (i=0; i<n; i++)
        a[i] = i;
    #pragma omp parallel private(suma)
    {
        suma=16;
        #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");
    printf("Suma fuera de parallel: %d\n", suma);
}

```

**CAPTURAS DE PANTALLA:**

```

File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer2] 2021-04-13 martes
>>make private-clauseModificado_b_EXE
gcc -O2 -fopenmp -o private-clauseModificado_b_EXE private-clauseModificado_b.c
private-clauseModificado_b.c:9:1: warning: return type defaults to 'int' [-Wimplicit-int]
  9 | main()
    | ^~~~~
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer2] 2021-04-13 martes
>>./private-clauseModificado_b_EXE
thread 1 suma a[1] / thread 2 suma a[2] / thread 3 suma a[3] / thread 4 suma a[4] / thread 5 suma a[5] /
thread 6 suma a[6] / thread 0 suma a[0] /
* thread 4 suma= 795779780
* thread 5 suma= 795779781
* thread 6 suma= 795779782
* thread 3 suma= 795779779
* thread 2 suma= 795779778
* thread 1 suma= 795779777
* thread 0 suma= 8
* thread 7 suma= 795779776
Suma fuera de parallel: 16
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer2] 2021-04-13 martes
>>./private-clauseModificado_b_EXE
thread 0 suma a[0] / thread 5 suma a[5] / thread 3 suma a[3] / thread 2 suma a[2] / thread 6 suma a[6] /
thread 1 suma a[1] / thread 4 suma a[4] /
* thread 6 suma= -1599200570
* thread 7 suma= -1599200576
* thread 5 suma= -1599200571
* thread 3 suma= -1599200573
* thread 0 suma= 8
* thread 1 suma= -1599200575
* thread 4 suma= -1599200572
* thread 2 suma= -1599200574
Suma fuera de parallel: 16
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer2] 2021-04-13 martes
>>./private-clauseModificado_b_EXE
thread 6 suma a[6] / thread 4 suma a[4] / thread 3 suma a[3] / thread 5 suma a[5] / thread 2 suma a[2] /
thread 1 suma a[1] / thread 0 suma a[0] /
* thread 2 suma= -373767486
* thread 5 suma= -373767483
* thread 1 suma= -373767487
* thread 7 suma= -373767488
* thread 0 suma= 8
* thread 6 suma= -373767482
* thread 4 suma= -373767484
* thread 3 suma= -373767485
Suma fuera de parallel: 16
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer2] 2021-04-13 martes
>>

```

3. (a) Eliminar la cláusula `private(suma)` en `private-clause.c`. Ejecutar el código resultante. ¿Qué ocurre? (b) ¿A qué es debido?

**RESPUESTA:**

Ahora la suma que tiene cada hilo es la misma, esto sucede ya que la variable `suma` ahora está compartida, (esto debido a que se removió la cláusula `private(suma)`) entre todos los hilos, como la cláusula `for` tiene una barrera implícita, todos hilos esperarán que el resto termine y por lo tanto todos los hilos tendrán la suma total del bucle. Aquí a diferencia de los otros dos ejercicios, la variable está inicializada y se mantiene así pero se vuelve compartida entre los hilos.

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

```

main()
{
    int i, n = 7;
    int a[n], suma=16;
    for (i=0; i<n; i++)
        a[i] = i;
}

```

```

#pragma omp parallel
{
    #pragma omp for
    for (i=0; i<n; i++)
    {
        suma = suma + a[i];
        printf("thread %d a[%d] suma=%d / ", omp_get_thread_num(), i, suma);
    }
    printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
}
printf("\n");
printf("Suma fuera de parallel: %d\n", suma);
}

```

**CAPTURAS DE PANTALLA:**

```

File Edit View Bookmarks Settings Help
Suma fuera de parallel: 35
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer2] 2021-04-13 martes
>>make private-clauseModificado_c_EXE
gcc -O2 -fopenmp -o private-clauseModificado_c_EXE private-clauseModificado_c.c
private-clauseModificado_c.c:9:1: warning: return type defaults to 'int' [-Wimplicit-int]
9 | main()
  | ^~~~~
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer2] 2021-04-13 martes
>>./private-clauseModificado_c_EXE
thread 3 a[3] suma=19 / thread 1 a[1] suma=22 / thread 5 a[5] suma=21 / thread 2 a[2] suma=18 / thread 6
uma=24 / thread 0 a[0] suma=18 / thread 4 a[4] suma=20 /
* thread 1 suma= 24
* thread 3 suma= 24
* thread 2 suma= 24
* thread 4 suma= 24
* thread 5 suma= 24
* thread 0 suma= 24
* thread 7 suma= 24
* thread 6 suma= 24
Suma fuera de parallel: 24
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer2] 2021-04-13 martes
>>./private-clauseModificado_c_EXE
thread 2 a[2] suma=18 / thread 0 a[0] suma=24 / thread 4 a[4] suma=20 / thread 3 a[3] suma=19 / thread 5
uma=21 / thread 1 a[1] suma=25 / thread 6 a[6] suma=24 /
* thread 7 suma= 25
* thread 2 suma= 25
* thread 5 suma= 25
* thread 6 suma= 25
* thread 0 suma= 25
* thread 3 suma= 25
* thread 4 suma= 25
* thread 1 suma= 25
Suma fuera de parallel: 25
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer2] 2021-04-13 martes

```

4. En la ejecución de `firstlastprivate.c` de la pag. 21 del seminario se imprime un 6 fuera de la región parallel. **(a)** Cambiar el tamaño del vector a 10. Razonar lo que imprime el código en su PC con esta modificación. (añadir capturas de pantalla que muestren lo que ocurre). **(b)** Sin cambiar el tamaño del vector ¿podría imprimir el código otro valor? Razonar respuesta (añadir capturas de pantalla que muestren lo que ocurre).

**(a) RESPUESTA:**

Se está imprimiendo el resultado que obtiene la ejecución de la hebra que ha realizado la última iteración del bucle, como lo especifica la documentación de [OpenMP 5.1](#). En este caso se obtiene 9 ya que la la novena iteración es la última del bucle y fue ejecutada por el hilo 7.

**CAPTURAS DE PANTALLA:**

```

File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer4] 2021-04-20 martes
>>./firstlastprivate_EXE
thread 4 suma a[6] suma=6
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 3 suma a[5] suma=5
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 2 suma a[4] suma=4
thread 6 suma a[8] suma=8
thread 5 suma a[7] suma=7
thread 7 suma a[9] suma=9

Fuera de la construcción parallel suma=9
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer4] 2021-04-20 martes
>>./firstlastprivate_EXE
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 2 suma a[4] suma=4
thread 5 suma a[7] suma=7
thread 7 suma a[9] suma=9
thread 4 suma a[6] suma=6
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 6 suma a[8] suma=8
thread 3 suma a[5] suma=5

Fuera de la construcción parallel suma=9
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer4] 2021-04-20 martes
>>

```

**(b) RESPUESTA:**

Este valor puede variar dependiendo de que esa misma hebra haya realizado iteraciones anteriores, ya que en ese caso acumula su valor.

**CAPTURAS DE PANTALLA:**

```

File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer4] 2021-04-20 martes
>>./firstlastprivate_EXE
thread 4 suma a[6] suma=6
thread 5 suma a[7] suma=7
thread 2 suma a[4] suma=4
thread 7 suma a[9] suma=9
thread 6 suma a[8] suma=8
thread 3 suma a[5] suma=5
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

Fuera de la construcción parallel suma=9
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer4] 2021-04-20 martes
>>export OMP_NUM_THREADS=3
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer4] 2021-04-20 martes
>>./firstlastprivate_EXE
thread 1 suma a[4] suma=4
thread 1 suma a[5] suma=9
thread 1 suma a[6] suma=15
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 2 suma a[7] suma=7
thread 2 suma a[8] suma=15
thread 2 suma a[9] suma=24
thread 0 suma a[3] suma=6

```

```

Fuera de la construcción parallel suma=24
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer4] 2021-04-20 martes
>>export OMP_NUM_THREADS=4
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer4] 2021-04-20 martes
>>./firstlastprivate_EXE
thread 2 suma a[6] suma=6
thread 2 suma a[7] suma=13
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7
thread 1 suma a[5] suma=12
thread 3 suma a[8] suma=8
thread 3 suma a[9] suma=17
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3

Fuera de la construcción parallel suma=17
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer4] 2021-04-20 martes
>>

```

5. (a) ¿Qué se observa en los resultados de ejecución de copyprivate-clause.c cuando se elimina la cláusula copyprivate(a) en la directiva single? (b) ¿A qué cree que es debido? (añadir una captura de pantalla que muestre lo que ocurre)

**RESPUESTA:**

(a) Se observa que las variables están inicializadas a valores basura, excepto aquella del hilo que ejecutó la cláusula single.

(b) El valor de a es indefinido para el resto de los hilos debido a que sin el copyprivate, el valor que se introduce por pantalla no es difundido.

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

```

#include <stdio.h>
#include <omp.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
        {
            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:**

```

File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer5] 2021-04-20 martes
>>make
gcc -O2 -fopenmp -o copyprivate-clauseModificado_EXE copyprivate-clauseModificado.c
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer5] 2021-04-20 martes
>>./copyprivate-clauseModificado_EXE

Introduce valor de inicialización a: 16

Single ejecutada por el thread 2
Depués de la región parallel:
b[0] = 22094    b[1] = 22094    b[2] = 0        b[3] = 16        b[4] = 0        b[5] = 0        b[6] = 0
b[7] = 0        b[8] = 0
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer5] 2021-04-20 martes
>>

```

6. En el ejemplo `reduction-clause.c` sustituya `suma=0` por `suma=10`. ¿Qué resultado se imprime ahora? Justifique el resultado (añada capturas de pantalla que muestren lo que ocurre)

**RESPUESTA:**

Se imprimen los valores originales más 10, lo cual tiene sentido debido a que se ha inicializado el valor de suma a 10 y lo que realiza el código con la cláusula `reduction` es ir acumulando el valor de cada hilo en esa variable suma manteniendo su valor original de inicialización.

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

```

#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
    #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);
    }

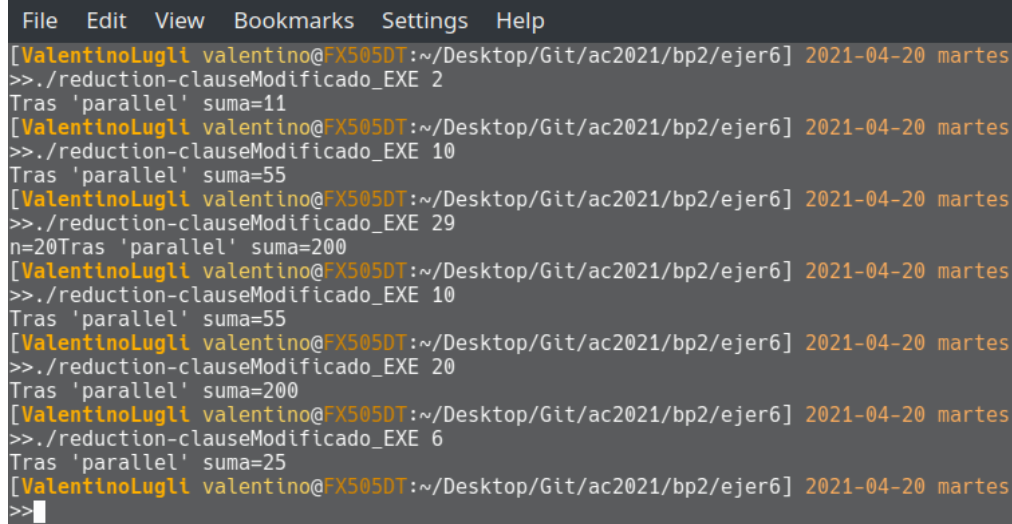
    for (i=0; i<n; i++)
        a[i] = i;

    #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:**


```
File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer6] 2021-04-20 martes
>>./reduction-clauseModificado_EXE 2
Tras 'parallel' suma=11
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer6] 2021-04-20 martes
>>./reduction-clauseModificado_EXE 10
Tras 'parallel' suma=55
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer6] 2021-04-20 martes
>>./reduction-clauseModificado_EXE 29
n=20Tras 'parallel' suma=200
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer6] 2021-04-20 martes
>>./reduction-clauseModificado_EXE 10
Tras 'parallel' suma=55
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer6] 2021-04-20 martes
>>./reduction-clauseModificado_EXE 20
Tras 'parallel' suma=200
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer6] 2021-04-20 martes
>>./reduction-clauseModificado_EXE 6
Tras 'parallel' suma=25
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer6] 2021-04-20 martes
>>|
```

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 añadir más directivas de trabajo compartido (añada capturas de pantalla que muestren lo que ocurre).

**RESPUESTA:**

Se utiliza la cláusula shared(a) para compartir el vector a entre todos los hilos y para que accedan sin condiciones de carrera se utiliza la cláusula atomic en la suma.

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

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

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;

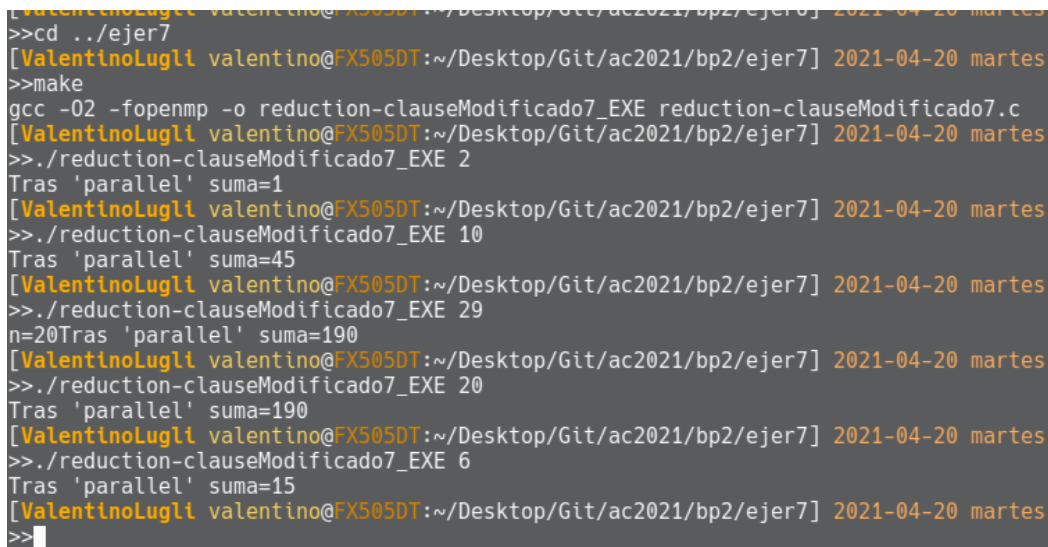
    #pragma omp parallel for shared(a)
        for (i=0; i<n; i++)
```

```

        #pragma omp atomic
        suma += a[i];

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

```

**CAPTURAS DE PANTALLA:**


```

>>cd ../ejer7
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer7] 2021-04-20 martes
>>make
gcc -O2 -fopenmp -o reduction-clauseModificado7_EXE reduction-clauseModificado7.c
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer7] 2021-04-20 martes
>>./reduction-clauseModificado7_EXE 2
Tras 'parallel' suma=1
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer7] 2021-04-20 martes
>>./reduction-clauseModificado7_EXE 10
Tras 'parallel' suma=45
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer7] 2021-04-20 martes
>>./reduction-clauseModificado7_EXE 20
Tras 'parallel' suma=190
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer7] 2021-04-20 martes
>>./reduction-clauseModificado7_EXE 6
Tras 'parallel' suma=15
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer7] 2021-04-20 martes
>>

```

Resto de ejercicios (usar en atcgrid la cola ac a no ser que se tenga que usar atcgrid4)

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):

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

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).

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

```

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

int main(int argc, char **argv)
{
    if(argc < 2)
    {
        printf("Uso: ./pmv-secuencial_EXE n\nn      Dimensión del vector y de la matriz.\n");
        exit(-1);
    }

    // Declaración de datos.
    struct timespec cgt1, cgt2; double ncgt;

```

```

int N = atoi(argv[1]);

int *v1, *v3;
v1 = (int*) malloc(N*sizeof(int));
v3 = (int*) malloc(N*sizeof(int));

int **matrix;
matrix = (int**) malloc(N*sizeof(int*));

if(v1 == NULL || matrix == NULL || v3 == NULL)
{
    printf("Error en la reserva de espacio\n");
    exit(-2);
}

for(int i=0; i<N; i++)
{
    matrix[i] = (int*) malloc(N*sizeof(int));
    if(matrix[i] == NULL)
    {
        printf("Error en la reserva de espacio\n");
        exit(-2);
    }
}

// Inicialización de datos.
for (int i = 0; i < N; i++)
{
    v1[i] = 1;
    for (int j = 0; j < N; j++)
    {
        matrix[i][j] = 1;
    }
}

// Cálculo de la multiplicación de v1 * matrix = v3
int aux;

clock_gettime(CLOCK_REALTIME,&cgt1);

for (int i = 0; i < N; i++)
{
    aux = 0;
    for (int j = 0; j < N; j++)
    {
        aux += matrix[j][i]*v1[j];
    }
    v3[i] = aux;
}

clock_gettime(CLOCK_REALTIME,&cgt2);

ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
      (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

// Imprimir los datos
printf("Dimension: %d Tiempo: %11.9f \n",N,ncgt);

printf("Resultado: ");
if(N<12)

```

```

{
    for (int i = 0; i < N; i++)
    {
        printf("[%d] = %d ", i, v3[i]);
    }


    else
    {
        printf("[0] = %d [%d] = %d", v3[0], N-1, v3[N-1]);
    }
    printf("\n");

    free(v1);
    free(v3);

    for (int i = 0; i < N; i++) free(matrix[i]);
    free(matrix);

    return 0;
}

```

**CAPTURAS DE PANTALLA:**


```

File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer8] 2021-04-24 sábado
>>make
gcc -O2 -fopenmp -o pmv-secuencial_EXE pmv-secuencial.c
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer8] 2021-04-24 sábado
>>./pmv-secuencial_EXE 4
Dimension: 4 Tiempo: 0.000000361
Resultado: [0] = 4 [1] = 4 [2] = 4 [3] = 4
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer8] 2021-04-24 sábado
>>./pmv-secuencial_EXE 8
Dimension: 8 Tiempo: 0.000000531
Resultado: [0] = 8 [1] = 8 [2] = 8 [3] = 8 [4] = 8 [5] = 8 [6] = 8 [7] = 8
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer8] 2021-04-24 sábado
>>./pmv-secuencial_EXE 20
Dimension: 20 Tiempo: 0.000001362
Resultado: [0] = 20; [19] = 20
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer8] 2021-04-24 sábado
>>

```

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

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

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

int main(int argc, char **argv)
{
    if(argc < 2)
    {
        printf("Uso: ./*_EXE n\nn    Dimensión del vector y de la matriz.\n");
        exit(-1);
    }

    // Declaración de datos.
    struct timespec cgt1,cgt2; double ncgt;

    int N = atoi(argv[1]);

    int *v1, *v3;
    v1 = (int*) malloc(N*sizeof(int));
    v3 = (int*) malloc(N*sizeof(int));

    int **matrix;
    matrix = (int**) malloc(N*sizeof(int*));

    if(v1 == NULL || matrix == NULL || v3 == NULL)
    {
        printf("Error en la reserva de espacio\n");
        exit(-2);
    }

    for(int i=0; i<N; i++)
    {
        matrix[i] = (int*) malloc(N*sizeof(int));
        if(matrix[i] == NULL)
        {
            printf("Error en la reserva de espacio\n");
            exit(-2);
        }
    }

    // Inicialización de datos.
    int j;
    #pragma omp parallel for private(j) shared(v1)
    for (int i = 0; i < N; i++)
    {
        v1[i] = 1;
        for (j = 0; j < N; j++)
        {
            matrix[i][j] = 1;
        }
    }
}
```

```

// Cálculo de la multiplicación de v1 * matrix = v3
int aux;

clock_gettime(CLOCK_REALTIME,&cgt1);

// Filas
#pragma omp parallel for shared(v3) private(j, aux)
for (int i = 0; i < N; i++)
{
    // Columnas
    aux = 0;
    for (j = 0; j < N; j++)
    {
        aux += matrix[j][i]*v1[j];
    }
    v3[i] = aux;
}

clock_gettime(CLOCK_REALTIME,&cgt2);

ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
      (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

// Imprimir los datos
printf("Dimension: %d Tiempo: %11.9f \n", N, ncgt);

printf("Resultado: ");
if(N<12)
{
    for (int i = 0; i < N; i++)
    {
        printf("[%d] = %d ", i, v3[i]);
    }
}
else
{
    printf("[0] = %d; [%d] = %d", v3[0], N-1, v3[N-1]);
}
printf("\n");

// Liberar memoria
free(v1);
free(v3);

for (int i = 0; i < N; i++) free(matrix[i]);
free(matrix);

return 0;
}

```

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

```

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

#ifdef _OPENMP
    #include <omp.h>
#else

```

```

#define omp_get_thread_num() 0
#endif

int main(int argc, char **argv)
{
    if(argc < 2)
    {
        printf("Uso: ./*_EXE n\nn    Dimensión del vector y de la matriz.\n");
        exit(-1);
    }

    // Declaración de datos.
    struct timespec cgt1,cgt2; double ncgt;

    int N = atoi(argv[1]);

    int *v1, *v3;
    v1 = (int*) malloc(N*sizeof(int));
    v3 = (int*) malloc(N*sizeof(int));

    int **matrix;
    matrix = (int**) malloc(N*sizeof(int*));

    if(v1 == NULL || matrix == NULL || v3 == NULL)
    {
        printf("Error en la reserva de espacio\n");
        exit(-2);
    }

    for(int i=0; i<N; i++)
    {
        matrix[i] = (int*) malloc(N*sizeof(int));
        if(matrix[i] == NULL)
        {
            printf("Error en la reserva de espacio\n");
            exit(-2);
        }
    }

    // Inicialización de datos.
    for (int i = 0; i < N; i++)
    {
        v1[i] = 1;
        #pragma omp parallel for
        for (int j = 0; j < N; j++)
        {
            matrix[i][j] = 1;
        }
    }

    // Cálculo de la multiplicación de v1 * matrix = v3
    int aux;

    clock_gettime(CLOCK_REALTIME,&cgt1);

    // Filas
    for (int i = 0; i < N; i++)
    {
        // Columnas
        aux = 0;

```



```

        #pragma omp parallel for shared(aux)
        for (int j = 0; j < N; j++)
        {
            #pragma omp atomic
            aux += matrix[j][i]*v1[j];
        }
        v3[i] = aux;
    }

    clock_gettime(CLOCK_REALTIME,&cgt2);

    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
        (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

    // Imprimir los datos
    printf("Dimension: %d Tiempo: %11.9f \n", N, ncgt);

    printf("Resultado: ");
    if(N<12)
    {
        for (int i = 0; i < N; i++)
        {
            printf("[%d] = %d ", i, v3[i]);
        }
    }
    else
    {
        printf("[0] = %d; [%d] = %d", v3[0], N-1, v3[N-1]);
    }
    printf("\n");

    // Liberar memoria
    free(v1);
    free(v3);

    for (int i = 0; i < N; i++) free(matrix[i]);
    free(matrix);

    return 0;
}

```

**RESPUESTA:**

Se obtuvo un error con la asignación y la liberación de memoria de la matriz dinámica, lo cual resolví yendo a Google y buscando mi problema en particular y leyendo varias entradas en StackOverflow, que era como declarar una matriz de esa manera en C; lo cual había olvidado.

El resto de problemas a la hora de compilar resultaban más normales, olvidar un punto y coma en algún sitio, o alguna variable que había olvidado declarar anteriormente.

Por parte de errores en ejecución, mayoritariamente fueron errores debido a olvidar colocar ciertas cláusulas de OpenMP; por ejemplo, olvidar marcar ciertos valores como privados o como compartidos, lo que daba sumas incorrectas, pero lo solucioné revisando las diapositivas de los Seminarios y depurando la ejecución.

**CAPTURAS DE PANTALLA:**

```

File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer9] 2021-04-24 sábado
>>make pmv-OpenMP-a_EXE
gcc -O2 -fopenmp -o pmv-OpenMP-a_EXE pmv-OpenMP-a.c
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer9] 2021-04-24 sábado
>>./pmv-OpenMP-a_EXE 4
Dimension: 4 Tiempo: 0.000003176
Resultado: [0] = 4 [1] = 4 [2] = 4 [3] = 4
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer9] 2021-04-24 sábado
>>./pmv-OpenMP-a_EXE 8
Dimension: 8 Tiempo: 0.000003566
Resultado: [0] = 8 [1] = 8 [2] = 8 [3] = 8 [4] = 8 [5] = 8 [6] = 8 [7] = 8
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer9] 2021-04-24 sábado
>>./pmv-OpenMP-a_EXE 20
Dimension: 20 Tiempo: 0.000003897
Resultado: [0] = 20; [19] = 20
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer9] 2021-04-24 sábado

File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer9] 2021-04-24 sábado
>>make pmv-OpenMP-b_EXE
gcc -O2 -fopenmp -o pmv-OpenMP-b_EXE pmv-OpenMP-b.c
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer9] 2021-04-24 sábado
>>./pmv-OpenMP-b_EXE 4
Dimension: 4 Tiempo: 0.000012153
Resultado: [0] = 4 [1] = 4 [2] = 4 [3] = 4
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer9] 2021-04-24 sábado
>>./pmv-OpenMP-b_EXE 8
Dimension: 8 Tiempo: 0.000025438
Resultado: [0] = 8 [1] = 8 [2] = 8 [3] = 8 [4] = 8 [5] = 8 [6] = 8 [7] = 8
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer9] 2021-04-24 sábado
>>./pmv-OpenMP-b_EXE 20
Dimension: 20 Tiempo: 0.000065313
Resultado: [0] = 20; [19] = 20
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer9] 2021-04-24 sábado
>>

```

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**

```

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

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

int main(int argc, char **argv)
{
    if(argc < 2)
    {
        printf("Uso: ./*_EXE n\n\n    Dimensión del vector y de la matriz.\n");
        exit(-1);
    }
}

```

```

// Declaración de datos.
struct timespec cgt1,cgt2; double ncgt;

int N = atoi(argv[1]);

int *v1, *v3;
v1 = (int*) malloc(N*sizeof(int));
v3 = (int*) malloc(N*sizeof(int));

int **matrix;
matrix = (int**) malloc(N*sizeof(int*));

if(v1 == NULL || matrix == NULL || v3 == NULL)
{
    printf("Error en la reserva de espacio\n");
    exit(-2);
}

for(int i=0; i<N; i++)
{
    matrix[i] = (int*) malloc(N*sizeof(int));
    if(matrix[i] == NULL)
    {
        printf("Error en la reserva de espacio\n");
        exit(-2);
    }
}

// Inicialización de datos.

for (int i = 0; i < N; i++)
{
    v1[i] = 1;
    #pragma omp parallel for
    for (int j = 0; j < N; j++)
    {
        matrix[i][j] = 1;
    }
}

// Cálculo de la multiplicación de v1 * matrix = v3
int aux;

clock_gettime(CLOCK_REALTIME,&cgt1);

// Filas
for (int i = 0; i < N; i++)
{
    // Columnas
    aux = 0;
    #pragma omp parallel for reduction(+:aux)
    for (int j = 0; j < N; j++)
    {
        aux += matrix[j][i]*v1[j];
    }
    v3[i] = aux;
}

clock_gettime(CLOCK_REALTIME,&cgt2);

```

```

ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
      (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

// Imprimir los datos
printf("Dimension: %d Tiempo: %11.9f \n", N, ncgt);

printf("Resultado: ");
if(N<12)
{
    for (int i = 0; i < N; i++)
    {
        printf("[%d] = %d ", i, v3[i]);
    }
}
else
{
    printf("[0] = %d; [%d] = %d", v3[0], N-1, v3[N-1]);
}
printf("\n");

// Liberar memoria
free(v1);
free(v3);

for (int i = 0; i < N; i++) free(matrix[i]);
free(matrix);

return 0;
}

```

**RESPUESTA:**

Como la adaptación del ejercicio anterior a este no comprendía un cambio significativo en el código, se modificaron las cláusulas y no hubieron errores de compilación ni de ejecución; meramente se revisó las diapositivas del Seminario para verificar bien el formato que debe utilizar la cláusula.

**CAPTURAS DE PANTALLA:**

```

File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer10] 2021-04-24 sábado
>>make
gcc -O2 -fopenmp -o pmv-OpenmMP-reduction_EXE pmv-OpenmMP-reduction.c
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer10] 2021-04-24 sábado
>>./pmv-OpenmMP-reduction_EXE 4
Dimension: 4 Tiempo: 0.000011962
Resultado: [0] = 4 [1] = 4 [2] = 4 [3] = 4
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer10] 2021-04-24 sábado
>>./pmv-OpenmMP-reduction_EXE 8
Dimension: 8 Tiempo: 0.000022833
Resultado: [0] = 8 [1] = 8 [2] = 8 [3] = 8 [4] = 8 [5] = 8 [6] = 8 [7] = 8
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer10] 2021-04-24 sábado
>>./pmv-OpenmMP-reduction_EXE 20
Dimension: 20 Tiempo: 0.000056406
Resultado: [0] = 20; [19] = 20
[ValentinoLugli valentino@FX505DT:~/Desktop/Git/ac2021/bp2/ejer10] 2021-04-24 sábado
>>

```

11. Realizar una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid4, en uno de los nodos de la cola ac 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):**

```

File Edit View Bookmarks Settings Help
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer11] 2021-04-24 sábado
>>../ejer9/pmv-OpenMP-a_EXE 20000
Dimension: 20000 Tiempo: 3.498718926
Resultado: [0] = 20000; [19999] = 20000
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer11] 2021-04-24 sábado
>>../ejer9/pmv-OpenMP-b_EXE 20000
Dimension: 20000 Tiempo: 4.868070209
Resultado: [0] = 20000; [19999] = 20000
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer11] 2021-04-24 sábado
>>../ejer10/pmv-OpenmMP-reduction_EXE 20000
Dimension: 20000 Tiempo: 0.947371865
Resultado: [0] = 20000; [19999] = 20000
[ValentinoLugli valentino@FX5050T:~/Desktop/Git/ac2021/bp2/ejer11] 2021-04-24 sábado
>>

```

Resulta evidente que la versión con la clausula de reducción es la que produce el menor tiempo de los tres códigos.

**JUSTIFICAR AHORA EN BASE AL CÓDIGO LA DIFERENCIA EN TIEMPOS:**

Los códigos A y B realizan de manera explícita la suma de cada multiplicación de la matriz, luego añadiéndose esa suma a la componente del vector de salida: esto es la reducción propiamente, en cambio con la cláusula de reducción, esto ya está implementado internamente de una manera mucho más optimizada y con menos overhead como se puede notar en la captura.

**CAPTURA DE PANTALLA del script pmv-OpenmMP-script.sh**

```

#!/bin/bash
#Ordenes para el Gestor de carga de trabajo:
#1. Asigna al trabajo un nombre
#SBATCH --job-name=bp2_11

# VERSION ATCGRID
#Instrucciones del script para ejecutar código:
echo "### Cola: $SLURM_JOB_PARTITION"

echo -e "---SECUENCIAL---"
srun ./pmv-secuencial_EXE 7500
srun ./pmv-secuencial_EXE 15000

for (( i=0; i<16; i+=1 )); do

    echo -e "---THREADS=$((i + 1))"

    export OMP_NUM_THREADS=$((i + 1))

    srun ./pmv-OpenmMP-reduction_EXE 7500
    srun ./pmv-OpenmMP-reduction_EXE 15000

done

echo -e "---THREADS=32"

export OMP_NUM_THREADS=32

echo -e "    REDUCTION"
srun ./pmv-OpenmMP-reduction_EXE 7500
srun ./pmv-OpenmMP-reduction_EXE 15000

```

**CAPTURAS DE PANTALLA (mostrar la ejecución en atcgrid – envío(s) a la cola):**

```

File Edit View Bookmarks Settings Help
[ValentinoLugli c1estudiante16@atcgrid:~/bp2/ejer11] 2021-04-24 sábado
>>sbatch -pac -Aac -n1 -c24 --exclusive pmv-OpenmMP-script.sh
Submitted batch job 97182
[ValentinoLugli c1estudiante16@atcgrid:~/bp2/ejer11] 2021-04-24 sábado
>>cat slurm-97182.out
### Cola: ac
---SECUENCIAL---
Dimension: 7500 Tiempo: 0.833558847
Resultado: [0] = 7500; [7499] = 7500
Dimension: 15000 Tiempo: 4.103812111
Resultado: [0] = 15000; [14999] = 15000
---THREADS=1
Dimension: 7500 Tiempo: 0.848910909
Resultado: [0] = 7500; [7499] = 7500
Dimension: 15000 Tiempo: 4.132495181
Resultado: [0] = 15000; [14999] = 15000
---THREADS=2
Dimension: 7500 Tiempo: 0.447431652
Resultado: [0] = 7500; [7499] = 7500
Dimension: 15000 Tiempo: 2.234594664
Resultado: [0] = 15000; [14999] = 15000
---THREADS=3
Dimension: 7500 Tiempo: 0.302565806
Resultado: [0] = 7500; [7499] = 7500
(c1estudiante16) atcgrid.ugr.es x ejer11 : sftp x ejer11 : bash x

```

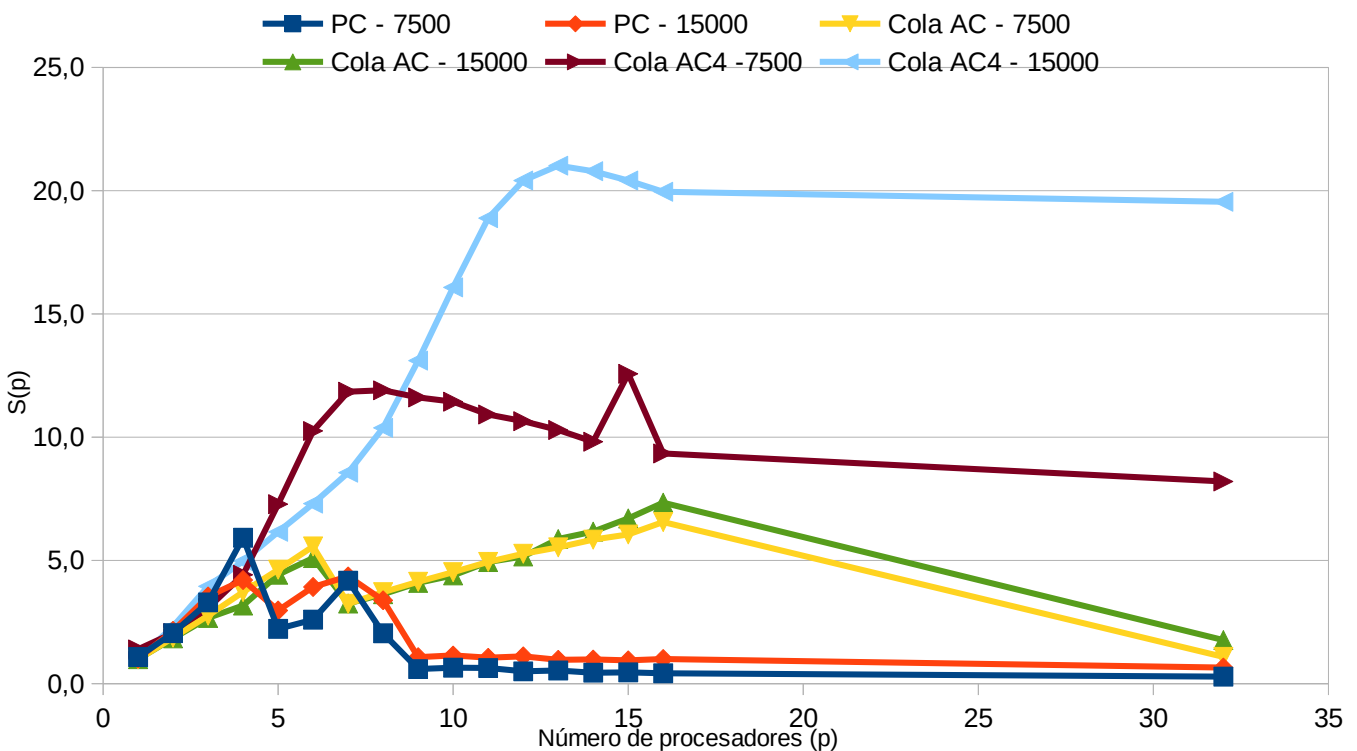
```

File Edit View Bookmarks Settings Help
[ValentinoLugli c1estudiante16@atcgrid:~/bp2/ejer11] 2021-04-24 sábado
>>sbatch -pac4 -Aac -n1 -c32 --exclusive pmv-OpenmMP-script.sh
Submitted batch job 97183
[ValentinoLugli c1estudiante16@atcgrid:~/bp2/ejer11] 2021-04-24 sábado
>>ls
pmv-OpenmMP-reduction_EXE pmv-OpenmMP-script.sh pmv-secuencial_EXE slurm-97182.out slurm-97183.out
[ValentinoLugli c1estudiante16@atcgrid:~/bp2/ejer11] 2021-04-24 sábado
>>cat slurm-97183.out
### Cola: ac4
---SECUENCIAL---
Dimension: 7500 Tiempo: 0.526907123
Resultado: [0] = 7500; [7499] = 7500
Dimension: 15000 Tiempo: 2.723854821
Resultado: [0] = 15000; [14999] = 15000
---THREADS=1
Dimension: 7500 Tiempo: 0.378975713
Resultado: [0] = 7500; [7499] = 7500
Dimension: 15000 Tiempo: 2.736341605
Resultado: [0] = 15000; [14999] = 15000
---THREADS=2
Dimension: 7500 Tiempo: 0.257181603
Resultado: [0] = 7500; [7499] = 7500
Dimension: 15000 Tiempo: 1.174337127
Resultado: [0] = 15000; [14999] = 15000
(c1estudiante16) atcgrid.ugr.es x ejer11 : sftp x

```

**TABLA (con tiempos y ganancia) Y GRÁFICA (con ganancia):**

Nº de núcleos (p)	atcgrid1, atcgrid2 o atcgrid3				atcgrid4				PC			
	Tamaño= 7500		Tamaño= 15000		Tamaño= 7500		Tamaño= 15000		Tamaño= 7500		Tamaño= 15000	
	T(p)	S(p)	T(p)	S(p)	T(p)	S(p)	T(p)	S(p)	T(p)	S(p)	T(p)	S(p)
<b>Código Secuencial</b>	0,8336	----	4,1038	----	0,5269	----	2,7239	----	0,2861	----	1,8541	----
<b>1</b>	0,8489	0,9819	4,1325	0,9931	0,3790	1,3903	2,7363	0,9954	0,2644	1,0822	1,8297	1,0134
<b>2</b>	0,4474	1,8630	2,2346	1,8365	0,2572	2,0488	1,1743	2,3195	0,1397	2,0488	0,8666	2,1395
<b>3</b>	0,3026	2,7550	1,5394	2,6658	0,1718	3,0678	0,6903	3,9459	0,0868	3,2949	0,5239	3,5389
<b>4</b>	0,2242	3,7178	1,2947	3,1698	0,1193	4,4155	0,5480	4,9707	0,0483	5,9191	0,4412	4,2025
<b>5</b>	0,1802	4,6248	0,9287	4,4190	0,0724	7,2819	0,4423	6,1581	0,1288	2,2209	0,6246	2,9686
<b>6</b>	0,1498	5,5649	0,8055	5,0945	0,0514	10,2511	0,3730	7,3032	0,1102	2,5965	0,4728	3,9215
<b>7</b>	0,2573	3,2400	1,2623	3,2510	0,0445	11,8449	0,3184	8,5557	0,0685	4,1755	0,4261	4,3510
<b>8</b>	0,2246	3,7112	1,1318	3,6260	0,0443	11,8993	0,2625	10,3759	0,1400	2,0432	0,5481	3,3830
<b>9</b>	0,2018	4,1299	1,0043	4,0862	0,0454	11,6097	0,2079	13,1049	0,4841	0,5911	1,7230	1,0761
<b>10</b>	0,1842	4,5242	0,9346	4,3909	0,0461	11,4383	0,1694	16,0750	0,4438	0,6448	1,6215	1,1434
<b>11</b>	0,1687	4,9416	0,8322	4,9311	0,0483	10,9196	0,1442	18,8855	0,4519	0,6331	1,7648	1,0506
<b>12</b>	0,1582	5,2691	0,7952	5,1605	0,0495	10,6461	0,1334	20,4135	0,5737	0,4988	1,6699	1,1103
<b>13</b>	0,1507	5,5305	0,6996	5,8659	0,0512	10,2902	0,1296	21,0130	0,5376	0,5323	1,9231	0,9641
<b>14</b>	0,1427	5,8408	0,6655	6,1663	0,0537	9,8145	0,1310	20,7908	0,6451	0,4436	1,8903	0,9809
<b>15</b>	0,1378	6,0491	0,6124	6,7015	0,0419	12,5676	0,1335	20,4106	0,6234	0,4590	1,9589	0,9465
<b>16</b>	0,1271	6,5566	0,5589	7,3429	0,0564	9,3395	0,1365	19,9565	0,6849	0,4178	1,8583	0,9977
<b>32</b>	0,7680	1,0854	2,3193	1,7694	0,0643	8,2000	0,1393	19,5474	1,0126	0,2826	2,8466	0,6513



### COMENTARIOS SOBRE LOS RESULTADOS:

Es evidente por medio del gráfico cómo la escalabilidad es afectada por la cantidad de procesadores que posee el computador donde se realiza el cálculo. En mi PC, se llega a la máxima mejora de prestaciones al rededor de 4 núcleos, lo cual concuerda con que el procesador que poseo tiene 4 núcleos físicos, existe otro pico cuando hay 8 núcleos pero este es menor, se trata del overhead producido por los núcleos lógicos.

Aunque esto no sucede de igual forma en los nodos atcgrid y actgrid4; si bien es notable como existe una escalabilidad notable, llega al máximo en los 16 procesadores, teniendo un crecimiento lineal hasta ese punto, se puede analizar que el procesador, siendo uno de servidor, es más capaz de reducir el overhead producido por tener más de una hebra por cada procesador, teniendo en cuenta que estos nodos poseen 12 núcleos físicos.

De igual manera se puede notar con atcgrid4, el cual posee una mejora en prestaciones impresionantes, sin duda por poseer un procesador mucho más poderoso, sin embargo luego de los 12 procesadores empieza a haber una especie de meseta en cuanto a la mejora, esto puede darse porque se llega al límite en el que se puede



aprovechar el grado de paralelismo, así como también que se empieza a reducir, esto debido también a la sobrecarga de la comunicación entre las hebras.

**Tabla 1.** Tiempos de ejecución del código secuencial y de la versión paralela para atcgrid y para el PC personal

	atcgrid1, atcgrid2 o atcgrid3				atcgrid4				PC			
	Tamaño= entre 5000 y 10000		Tamaño= entre 10000 y 100000		Tamaño= entre 5000 y 10000		Tamaño= entre 10000 y 100000		Tamaño= entre 5000 y 10000		Tamaño= entre 10000 y 100000	
Nº de núcleos (p)	T(p)	S(p)	T(p)	S(p)	T(p)	S(p)	T(p)	S(p)	T(p)	S(p)	T(p)	S(p)
<b>Código Secuencial</b>		----		----		----		----		----		----
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
32												