

Grai2º curso / 2º
cuatr.

Grado Ing. Inform.

Doble Grado Ing.
Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Juan Manuel Rubio Rodríguez

Grupo de prácticas: D1

Fecha de entrega:

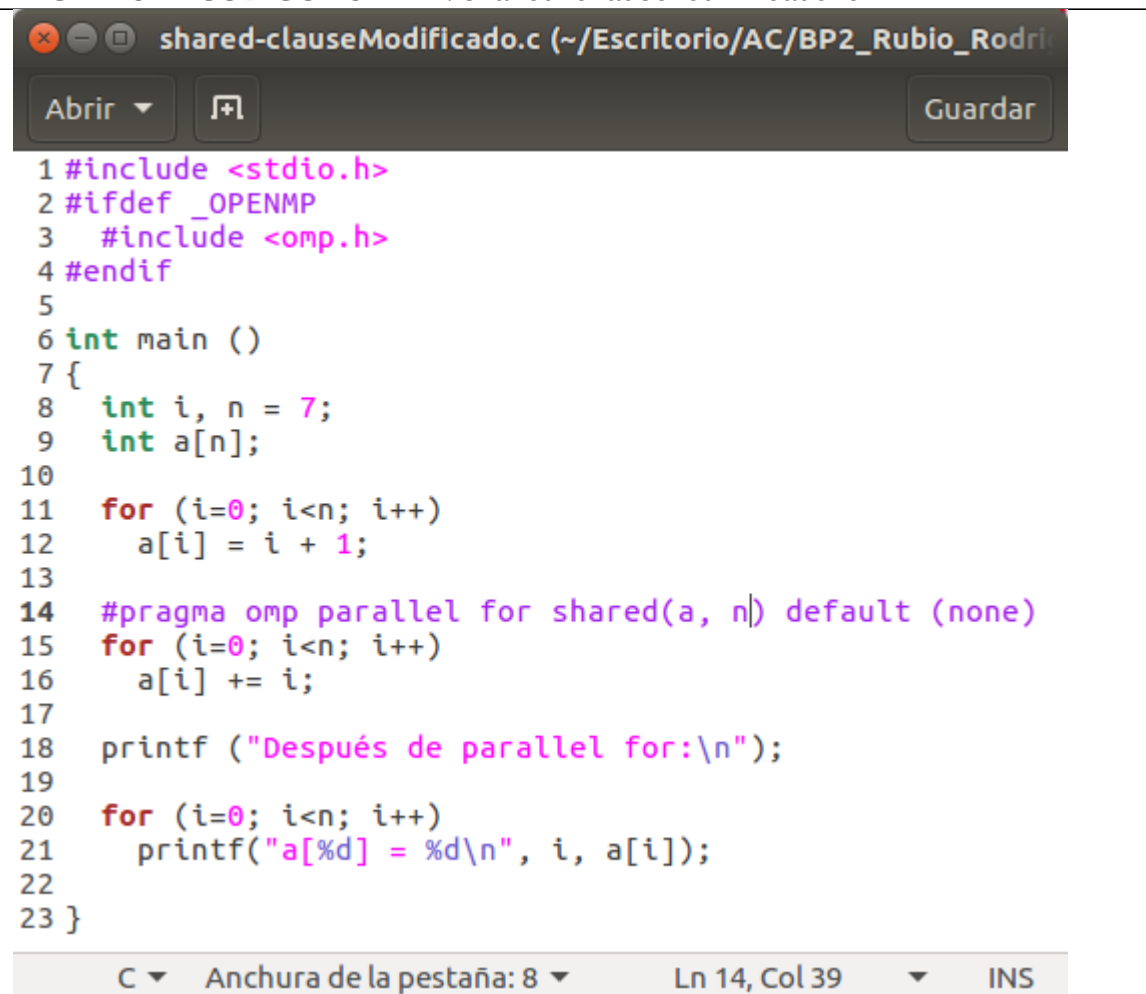
Fecha evaluación en clase:

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 la cláusula `default(none)`, es el programador el que debe especificar el alcance de las variables utilizadas en la construcción, en este caso, de `n`.

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`



```
1 #include <stdio.h>
2 #ifdef _OPENMP
3   #include <omp.h>
4 #endif
5
6 int main ()
7 {
8   int i, n = 7;
9   int a[n];
10
11   for (i=0; i<n; i++)
12     a[i] = i + 1;
13
14   #pragma omp parallel for shared(a, n) default (none)
15   for (i=0; i<n; i++)
16     a[i] += i;
17
18   printf ("Después de parallel for:\n");
19
20   for (i=0; i<n; i++)
21     printf("a[%d] = %d\n", i, a[i]);
22
23 }
```

C Anchura de la pestaña: 8 Ln 14, Col 39 INS

CAPTURAS DE PANTALLA:

```

juanma@juanma-X550VX: ~
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-11 miércoles
$gcc shared-clause.c -o shared-clause -fopenmp
shared-clause.c: In function 'main':
shared-clause.c:14:11: error: 'n' not specified in enclosing parallel
    #pragma omp parallel for shared(a) default (none)
                        ^
shared-clause.c:14:11: error: enclosing parallel
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-11 miércoles
$

```

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

RESPUESTA: Si se inicializa fuera de la construcción parallel, dentro de ella la variable contendrá basura. Al utilizar la cláusula private, debemos inicializar dentro de la construcción parallel la variable.

CAPTURA CÓDIGO FUENTE: private-clauseModificado.c

```

private-clauseModificado.c (~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel) - c
Abrir  Guardar
1 #include <stdio.h>
2 #ifdef _OPENMP
3     #include <omp.h>
4 #else
5     #define omp_get_num_thread() 0
6 #endif
7
8 int main ()
9 {
10     int i, n = 7;
11     int a[n], suma = 9;
12
13     for (i=0; i<n; i++)
14         a[i] = i;
15
16     #pragma omp parallel private (suma)
17     {
18         suma = 2;
19         #pragma omp for
20         for (i=0; i<n; i++)
21         {
22             suma = suma + a[i];
23             printf ("thread %d suma a[%d]/", omp_get_thread_num(), i);
24         }
25         printf ("\n thread %d suma=%d", omp_get_thread_num(), suma);
26     }
27
28     printf ("\n");
29
30 }
C  Anchura de la pestaña: 8  Ln 18, Col 5  INS

```

CAPTURAS DE PANTALLA:

```

juanma@juanma-X550VX: ~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-13 viernes
$g++ private-clauseModificado.c -o private-clause -fopenmp
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-13 viernes
$./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 0 suma=3
thread 2 suma=11
thread 3 suma=8
thread 1 suma=7
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-13 viernes
$

```

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

RESPUESTA: La variable `suma` ahora es compartida, lo que implica que cada thread puede machacar el resultado anterior y el resultado no es correcto.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

```

1 #include <stdio.h>
2 #ifdef _OPENMP
3 #include <omp.h>
4 #else
5 #define omp_get_num_thread() 0
6 #endif
7
8 int main ()
9 {
10     int i, n = 7;
11     int a[n], suma;
12
13     for (i=0; i<n; i++)
14         a[i] = i;
15
16     #pragma omp parallel
17     {
18         suma = 0;
19         #pragma omp for
20         for (i=0; i<n; i++)
21         {
22             suma = suma + a[i];
23             printf ("thread %d suma a[%d]/", omp_get_thread_num(), i);
24         }
25         printf ("\n thread %d suma=%d", omp_get_thread_num(), suma);
26     }
27
28     printf ("\n");
29
30 }

```

CAPTURAS DE PANTALLA:

```

juanma@juanma-X550VX: ~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-13 viernes
$g++ private-clauseModificado3.c -o private-clause3 -fopenmp
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-13 viernes
$./private-clause3
thread 0 suma a[0]/thread 0 suma a[1]/thread 1 suma a[2]/thread 1 suma a[3]/thread 2 suma a[4]/thread 2 suma a[5]/thread 3 suma a[6]/thread 3 suma a[7]
thread 2 suma=21
thread 1 suma=21
thread 0 suma=21
thread 3 suma=21
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-13 viernes
$

```

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: Siempre imprime 6 porque con `firstprivate` obtiene el valor al que se inicializa la variable `suma` que es '0' y con `lastprivate` obtiene el valor de la última iteración que en este caso es '6'; de tal forma que siempre tendrá '0+6'

CAPTURAS DE PANTALLA:

```

juanma@juanma-X550VX: ~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-13 viernes
$g++ firstlastprivate.c -o firstlastprivate -fopenmp
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-13 viernes
$./firstlastprivate
thread 0 suma a[0]/thread 0 suma a[1]/thread 2 suma a[4]/thread 2 suma a[5]/thread 1 suma a[2]/thread 1 suma a[3]/thread 3 suma a[6]/thread 3 suma a[7]
thread 0 suma=6
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-13 viernes
$./firstlastprivate
thread 0 suma a[0]/thread 0 suma a[1]/thread 2 suma a[4]/thread 2 suma a[5]/thread 1 suma a[2]/thread 1 suma a[3]/thread 3 suma a[6]/thread 3 suma a[7]
thread 0 suma=6
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-13 viernes
$

```

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

RESPUESTA: La cláusula `copyprivate` permite que una variable privada de un thread se copie a las variables privadas del mismo nombre del resto de threads (difusión). Cuando eliminamos la cláusula, esa variable ya no se copia en el resto de threads y por tanto, contienen basura.

CAPTURA CÓDIGO FUENTE: copyprivate-clauseModificado.c

```

1 #include <stdio.h>
2 #include <omp.h>
3
4 int main() {
5     int n = 9, i, b[n];
6
7     for (i=0; i<n; i++)
8         b[i] = -1;
9
10    #pragma omp parallel
11    { int a;
12      #pragma omp single //copyprivate(a)
13      {
14          printf("\nIntroduce valor de inicialización a: ");
15          scanf("%d", &a);
16          printf("\nSingle ejecutada por el thread %d\n", omp_get_thread_num
17      ());
18      }
19      #pragma omp for
20      for (i=0; i<n; i++)
21          b[i] = a;
22
23      printf ("Después de la región parallel:\n");
24      for (i=0; i<n; i++)
25          printf ("b[%d] = %d\t", i, b[i]);
26      printf("\n");
27 }

```

CAPTURAS DE PANTALLA:

```

juanma@juanma-X550VX: ~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-16 lunes
$g++ copyprivate-clauseModificado.c -o copyprivate -fopenmp
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-16 lunes
$./copyprivate

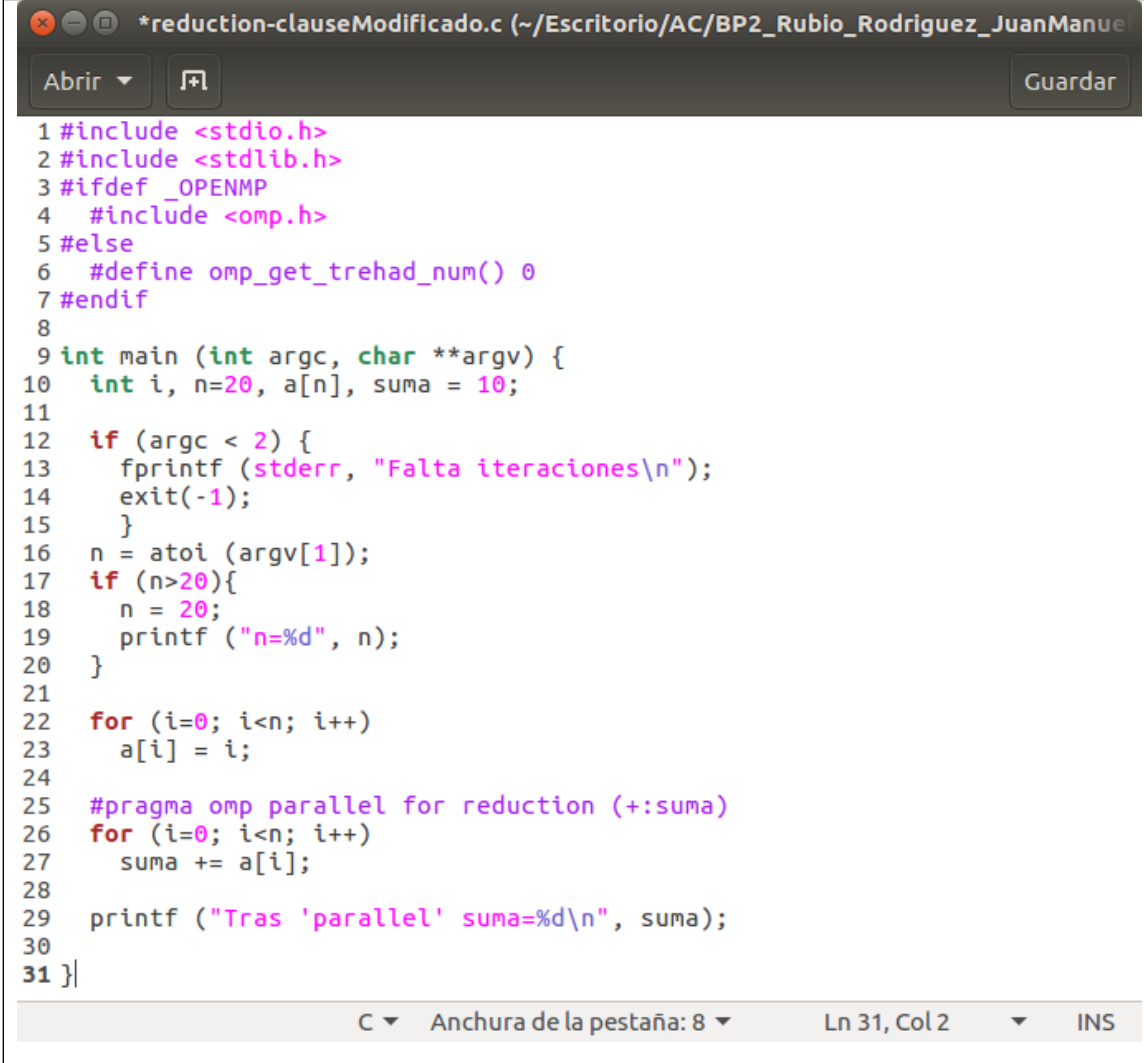
Introduce valor de inicialización a: 10

Single ejecutada por el thread 1
Después de la región parallel:
b[0] = 4197095 b[1] = 4197095 b[2] = 4197095 b[3] = 10 b[4] = 10 b[5] = 0 b[6] = 0 b[7] = 0 b[8] = 0
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-16 lunes
$

```

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

RESPUESTA: Imprime '55' cuando con suma = 0 imprime '45'. Hemos modificado el valor inicial y suma lo hemos incrementado en 10 unidades, con lo cual, el total suma también 10 unidades más.

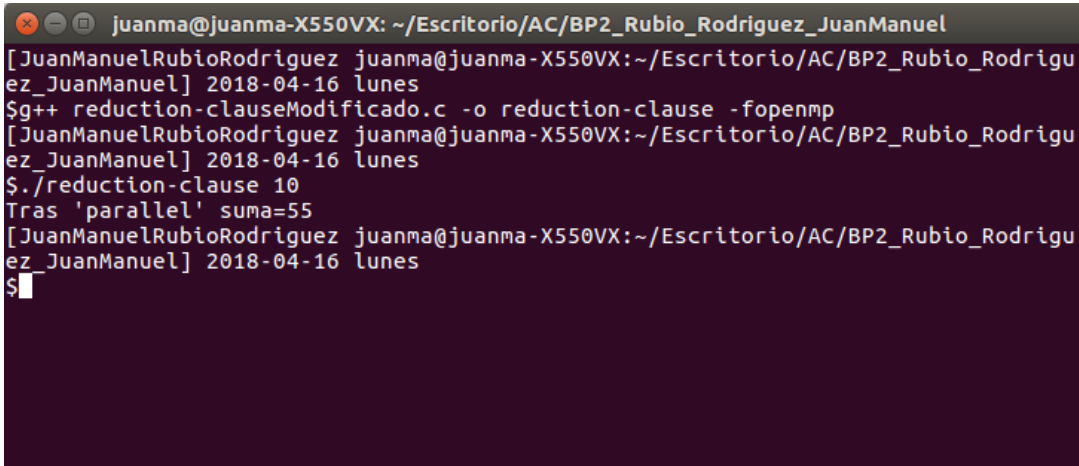
CAPTURA CÓDIGO FUENTE: reduction-clauseModificado.c


```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #ifdef _OPENMP
4     #include <omp.h>
5 #else
6     #define omp_get_trehanum() 0
7 #endif
8
9 int main (int argc, char **argv) {
10     int i, n=20, a[n], suma = 10;
11
12     if (argc < 2) {
13         fprintf (stderr, "Falta iteraciones\n");
14         exit(-1);
15     }
16     n = atoi (argv[1]);
17     if (n>20){
18         n = 20;
19         printf ("n=%d", n);
20     }
21
22     for (i=0; i<n; i++)
23         a[i] = i;
24
25     #pragma omp parallel for reduction (+:suma)
26     for (i=0; i<n; i++)
27         suma += a[i];
28
29     printf ("Tras 'parallel' suma=%d\n", suma);
30
31 }

```

C Anchura de la pestaña: 8 Ln 31, Col 2 INS

CAPTURAS DE PANTALLA:


```

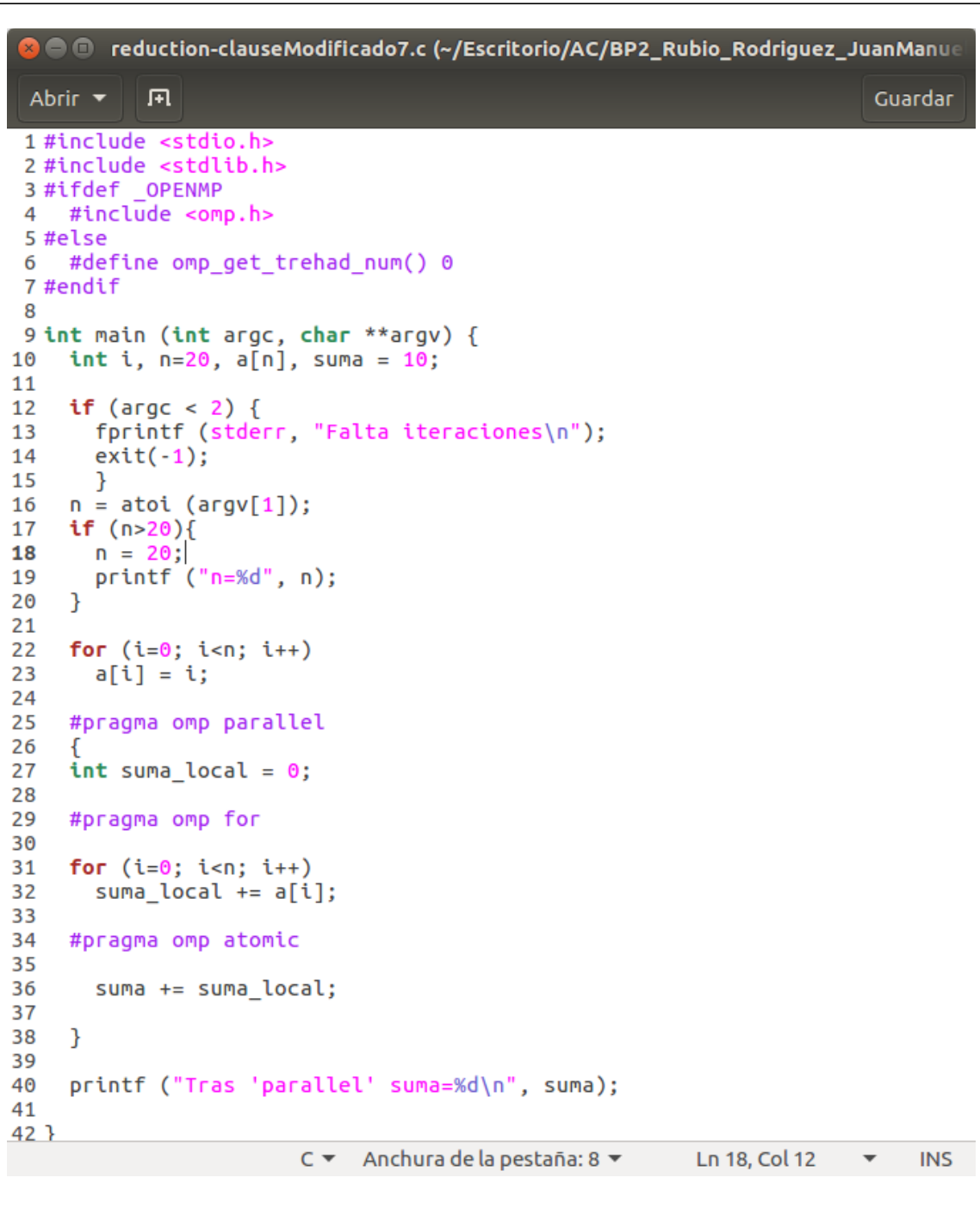
juanma@juanma-X550VX: ~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-16 lunes
$g++ reduction-clauseModificado.c -o reduction-clause -fopenmp
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-16 lunes
$./reduction-clause 10
Tras 'parallel' suma=55
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-16 lunes
$

```

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: He optado por tener una variable local `suma_local` donde ir almacenando los valores calculados en el bucle `for` y una directiva `atomic` para sumar todas esas sumas parciales de `suma_local` en la variable `suma` de forma que no se produzca ningún error al intentar acceder de forma simultanea por varios thread a `suma`.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`



```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #ifdef _OPENMP
4 #include <omp.h>
5 #else
6 #define omp_get_trehad_num() 0
7 #endif
8
9 int main (int argc, char **argv) {
10     int i, n=20, a[n], suma = 10;
11
12     if (argc < 2) {
13         fprintf (stderr, "Falta iteraciones\n");
14         exit(-1);
15     }
16     n = atoi (argv[1]);
17     if (n>20){
18         n = 20;
19         printf ("n=%d", n);
20     }
21
22     for (i=0; i<n; i++)
23         a[i] = i;
24
25     #pragma omp parallel
26     {
27         int suma_local = 0;
28
29         #pragma omp for
30
31         for (i=0; i<n; i++)
32             suma_local += a[i];
33
34         #pragma omp atomic
35             suma += suma_local;
36
37     }
38
39     printf ("Tras 'parallel' suma=%d\n", suma);
40
41 }
42

```

C Anchura de la pestaña: 8 Ln 18, Col 12 INS

CAPTURAS DE PANTALLA:

```

juanma@juanma-X550VX: ~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel
[juanmanuelrubiorodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-16 lunes
$g++ reduction-clauseModificado.c -o reduction-clause -fopenmp
[juanmanuelrubiorodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-16 lunes
$./reduction-clause 10
Tras 'parallel' suma=55
[juanmanuelrubiorodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-16 lunes
$

```

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):

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

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).

CAPTURA CÓDIGO FUENTE: pmv-secuencial.c

```

//Version para variables dinámicas
#include <stdlib.h>
#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif

int main(int argc, char** argv)
{
    int i, j;
    double t1, t2, total;

    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);

    //Reservo memoria para los vectores y la matriz

    double *v1, *v3, **M;

```



```

v1 = (double*) malloc(N*sizeof(double));
v3 = (double*) malloc(N*sizeof(double));
M = (double**) malloc(N*sizeof(double *));

for (i=0; i<N; i++){
    M[i] = (double*) malloc(N*sizeof(double));
}

//Inicializo matriz y vectores
for (i=0; i<N;i++)
{
    v1[i] = i;
    v3[i] = 0;
    for(j=0;j<N;j++)
        M[i][j] = i+j;
}

//Medida de tiempo
t1 = omp_get_wtime();

//Calculo el producto de la matriz por el vector v1
for (i=0; i<N;i++)
    for(j=0;j<N;j++)
        v3[i] += M[i][j] * v1[j];

//Medida de tiempo
t2 = omp_get_wtime();
total = t2 - t1;

//Imprimir el resultado y el tiempo de ejecución
printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ v3[0]=%8.6f\n", total,N,v3[0],N-1,v3[N-1]);

if (N<20)
    for (i=0; i<N;i++)
        printf(" v3[%d]=%5.2f\n", i, v3[i]);

//Libero memoria

free(v1);
free(v3);
for (i=0; i<N; i++)
    free(M[i]);
free(M);

return 0;
}

```

CAPTURAS DE PANTALLA:

```

juanma@juanma-X550VX: ~
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-20 viernes
$gcc pmv-secuencial.c -o pmv-secuencial -fopenmp
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-20 viernes
$./pmv-secuencial 8
Tiempo(seg.):0.000001618          / Tamaño:8          / v3[0]=140.000000 v3[7]=336.000000
v3[0]=140.00
v3[1]=168.00
v3[2]=196.00
v3[3]=224.00
v3[4]=252.00
v3[5]=280.00
v3[6]=308.00
v3[7]=336.00
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-20 viernes
$./pmv-secuencial 11
Tiempo(seg.):0.000002617          / Tamaño:11         / v3[0]=385.000000 v3[10]=935.000000
v3[0]=385.00
v3[1]=440.00
v3[2]=495.00
v3[3]=550.00
v3[4]=605.00
v3[5]=660.00
v3[6]=715.00
v3[7]=770.00
v3[8]=825.00
v3[9]=880.00
v3[10]=935.00
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-20 viernes
$

```

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 <stdlib.h>
#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif

int main(int argc, char** argv)
{
    int i, j;
    double t1, t2, total;

    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);

    //Reservo memoria para los vectores y la matriz

    double *v1, *v3, **M;
    v1 = (double*) malloc(N*sizeof(double));
    v3 = (double*) malloc(N*sizeof(double));
    M = (double**) malloc(N*sizeof(double *));

    for (i=0; i<N; i++){
        M[i] = (double*) malloc(N*sizeof(double));
    }

    #pragma omp parallel
    {

        //Inicializo matriz y vectores
        #pragma omp for private (j)
        for (i=0; i<N;i++)
        {
            v1[i] = i;
            v3[i] = 0;
            for(j=0;j<N;j++)
                M[i][j] = i+j;
        }

        //Medida de tiempo
        #pragma omp single
        t1 = omp_get_wtime();

        #pragma omp for private (j)

        //Calculo el producto de la matriz por el vector v1
        for (i=0; i<N;i++)
            for(j=0;j<N;j++)
                v3[i] += M[i][j] * v1[j];
    }
}

```

```

        //Medida de tiempo
        #pragma omp single
        t2 = omp_get_wtime();

    }

    total = t2 - t1;

    //Imprimir el resultado y el tiempo de ejecución
    printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ v3[0]=%8.6f\n", total, N, v3[0], N-1, v3[N-1]);

    if (N<20)
        for (i=0; i<N;i++)
            printf(" v3[%d]=%5.2f\n", i, v3[i]);

    //Libero memoria

    free(v1);
    free(v3);
    for (i=0; i<N; i++)
        free(M[i]);
    free(M);

    return 0;
}

```

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

```

#include <stdlib.h>
#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif

int main(int argc, char** argv)
{
    int i, j;
    double t1, t2, total;

    if (argc<2){
        printf("Falta tamaño de matriz y vector\n");
        exit(-1);
    }

    unsigned int N = atoi(argv[1]);

    //Reservo memoria para los vectores y la matriz

    double *v1, *v3, **M;
    v1 = (double*) malloc(N*sizeof(double));
    v3 = (double*) malloc(N*sizeof(double));
    M = (double**) malloc(N*sizeof(double *));

```

```

    for (i=0; i<N; i++){
        M[i] = (double*) malloc(N*sizeof(double));
    }

    //Inicializo matriz y vectores
    for (i=0; i<N;i++)
    {
        v1[i] = i;
        v3[i] = 0;
        #pragma omp parallel for
        for(j=0;j<N;j++)
            M[i][j] = i+j;
    }

    //Medida de tiempo
    t1 = omp_get_wtime();

    //Calculo el producto de la matriz por el vector v1
    for (i=0; i<N;i++)

        #pragma omp parallel
        {
            double suma_parcial = 0; //necesito una
variable donde ir almacenando los valores que calcula //el
bucle for; como ya se hizo en ejercicios anteriores
            #pragma omp for
            for(j=0;j<N;j++)
                suma_parcial += M[i][j] * v1[j];

            #pragma omp atomic
                v3[i] += suma_parcial;

        }

    //Medida de tiempo
    t2 = omp_get_wtime();
    total = t2 - t1;

    //Imprimir el resultado y el tiempo de ejecución
    printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ v3[0]=%8.6f
v3[%d]=%8.6f\n", total,N,v3[0],N-1,v3[N-1]);

    if (N<20)
        for (i=0; i<N;i++)
            printf(" v3[%d]=%5.2f\n", i, v3[i]);

    //Libero memoria

    free(v1);
    free(v3);
    for (i=0; i<N; i++)
        free(M[i]);
    free(M);

```

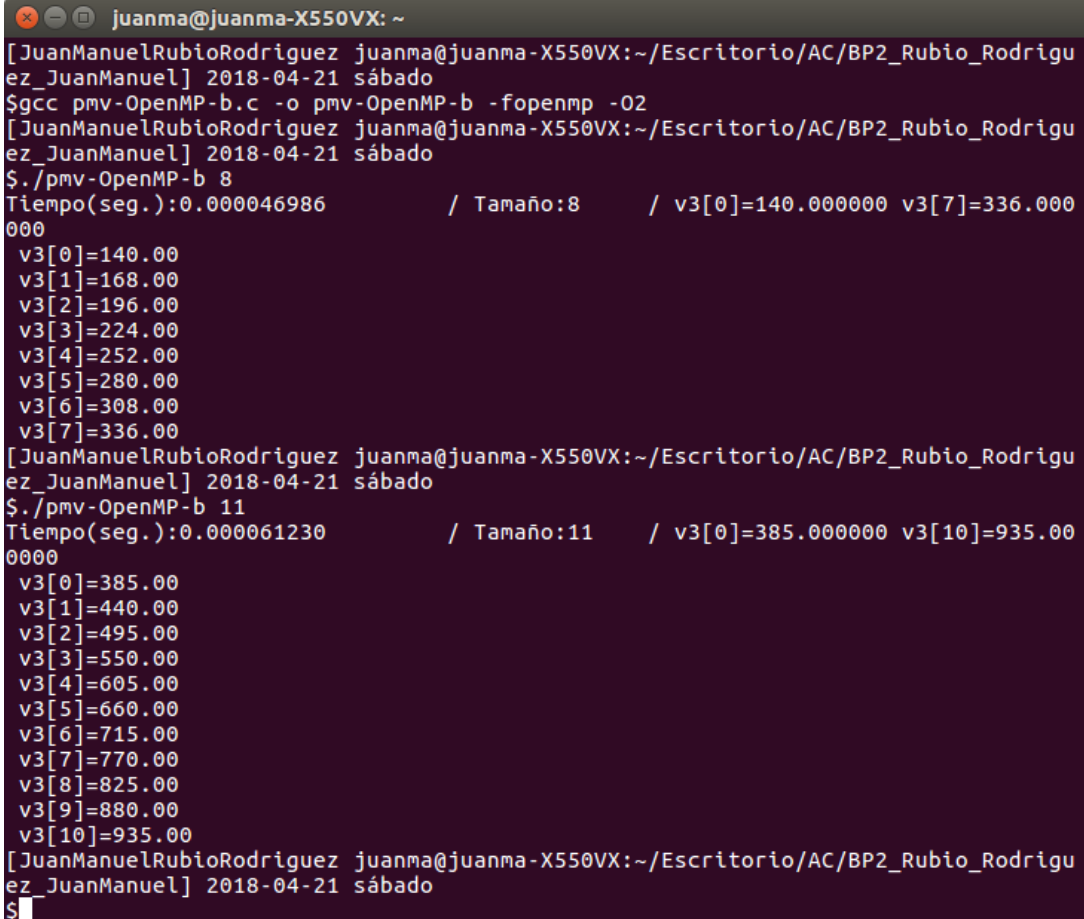
```

    return 0;
}

```

RESPUESTA: He tenido un error de ejecución con el primer código porque me faltó escribir `omp` en la directiva `#pragma omp for`, con lo cual, aunque se lo tragaba el compilador, daba errores de cálculo y de violación de segmento aleatorios.

CAPTURAS DE PANTALLA:



```

juanma@juanma-X550VX: ~
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-21 sábado
$gcc pmv-OpenMP-b.c -o pmv-OpenMP-b -fopenmp -O2
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-21 sábado
$./pmv-OpenMP-b 8
Tiempo(seg.):0.000046986 / Tamaño:8 / v3[0]=140.000000 v3[7]=336.000000
v3[0]=140.00
v3[1]=168.00
v3[2]=196.00
v3[3]=224.00
v3[4]=252.00
v3[5]=280.00
v3[6]=308.00
v3[7]=336.00
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-21 sábado
$./pmv-OpenMP-b 11
Tiempo(seg.):0.000061230 / Tamaño:11 / v3[0]=385.000000 v3[10]=935.000000
v3[0]=385.00
v3[1]=440.00
v3[2]=495.00
v3[3]=550.00
v3[4]=605.00
v3[5]=660.00
v3[6]=715.00
v3[7]=770.00
v3[8]=825.00
v3[9]=880.00
v3[10]=935.00
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-21 sábado
$

```

```

juanma@juanma-X550VX: ~
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-21 sábado
$gcc pmv-OpenMP-a.c -o pmv-OpenMP-a -fopenmp -O2
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-21 sábado
$./pmv-OpenMP-a 8
Tiempo(seg.):0.000005078          / Tamaño:8          / v3[0]=140.000000 v3[7]=336.000000
v3[0]=140.00
v3[1]=168.00
v3[2]=196.00
v3[3]=224.00
v3[4]=252.00
v3[5]=280.00
v3[6]=308.00
v3[7]=336.00
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-21 sábado
$./pmv-OpenMP-a 11
Tiempo(seg.):0.000011502          / Tamaño:11         / v3[0]=385.000000 v3[10]=935.000000
v3[0]=385.00
v3[1]=440.00
v3[2]=495.00
v3[3]=550.00
v3[4]=605.00
v3[5]=660.00
v3[6]=715.00
v3[7]=770.00
v3[8]=825.00
v3[9]=880.00
v3[10]=935.00
[JuanManuelRubioRodriguez juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-21 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 <stdlib.h>
#include <stdio.h>
#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_set_num_threads() 1
#endif

int main(int argc, char** argv)
{
    int i, j;
    double t1, t2, total;

```



```

if (argc<2){
    printf("Falta tamaño de matriz y vector\n");
    exit(-1);
}

unsigned int N = atoi(argv[1]);

    //Reservo memoria para los vectores y la matriz

double *v1, *v3, **M;
v1 = (double*) malloc(N*sizeof(double));
v3 = (double*) malloc(N*sizeof(double));
M = (double**) malloc(N*sizeof(double *));

for (i=0; i<N; i++){
    M[i] = (double*) malloc(N*sizeof(double));
}

//Inicializo matriz y vectores
for (i=0; i<N;i++)
{
    v1[i] = i;
    v3[i] = 0;
    #pragma omp parallel for
    for(j=0;j<N;j++)
        M[i][j] = i+j;
}

//Medida de tiempo
t1 = omp_get_wtime();

//Calculo el producto de la matriz por el vector v1
for (i=0; i<N;i++){
    double suma_parcial = 0;
    #pragma omp parallel for reduction
    (+:suma_parcial)
    for(j=0;j<N;j++)
        suma_parcial += M[i][j] * v1[j];

    v3[i] = suma_parcial;
}

//Medida de tiempo
t2 = omp_get_wtime();
total = t2 - t1;

//Imprimir el resultado y el tiempo de ejecución
printf("Tiempo(seg.):%11.9f\t / Tamaño:%u\t/ v3[0]=%8.6f\n", total, N, v3[0], N-1, v3[N-1]);

if (N<20)
    for (i=0; i<N;i++)
        printf(" v3[%d]=%5.2f\n", i, v3[i]);

```

```

        //Libero memoria

        free(v1);
        free(v3);
        for (i=0; i<N; i++)
            free(M[i]);
        free(M);

        return 0;
}

```

RESPUESTA: No he tenido ningún tipo de error de compilación ni de ejecución con este código.

CAPTURAS DE PANTALLA:

```

juanma@juanma-X550VX: ~
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-21 sábado
$gcc pmv-OpenMP-reduction.c -o pmv-OpenMP-reduction -fopenmp
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-21 sábado
$./pmv-OpenMP-reduction 8
Tiempo(seg.):0.000300109 / Tamaño:8 / v3[0]=140.000000 v3[7]=336.000000
v3[0]=140.00
v3[1]=168.00
v3[2]=196.00
v3[3]=224.00
v3[4]=252.00
v3[5]=280.00
v3[6]=308.00
v3[7]=336.00
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-21 sábado
$./pmv-OpenMP-reduction 11
Tiempo(seg.):0.000252707 / Tamaño:11 / v3[0]=385.000000 v3[10]=935.000000
v3[0]=385.00
v3[1]=440.00
v3[2]=495.00
v3[3]=550.00
v3[4]=605.00
v3[5]=660.00
v3[6]=715.00
v3[7]=770.00
v3[8]=825.00
v3[9]=880.00
v3[10]=935.00
[juanma@juanma-X550VX:~/Escritorio/AC/BP2_Rubio_Rodriguez_JuanManuel] 2018-04-21 sábado
$

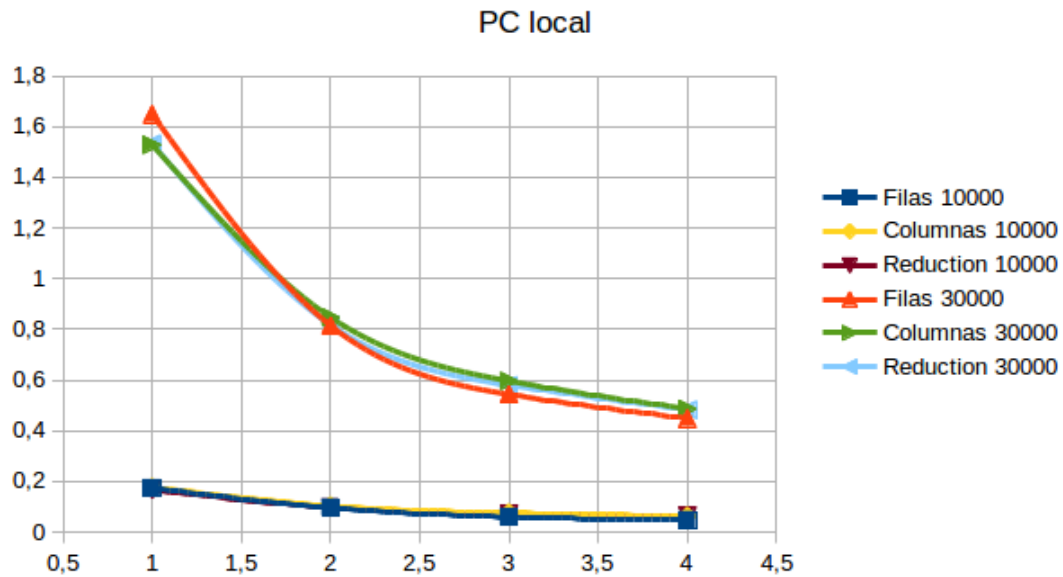
```

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):

PC local

n.º de threads	Filas 10000	Filas 30000	Columnas 10000	Columnas 30000	Reduction 10000	Reduction 30000
1	0,173078043	1,64910402	0,177049577	1,528431144	0,167161392	1,533328585
2	0,096247314	0,814406314	0,10466504	0,847292918	0,096203901	0,817054975
3	0,059513758	0,543392171	0,076751776	0,59634635	0,071048955	0,580058379
4	0,047375058	0,44844282	0,063260491	0,48625768	0,06521275	0,482668899



atcgrid

n.º de threads	Filas 10000	Filas 30000	Columnas 10000	Columnas 30000	Reduction 10000	Reduction 30000
1	0,150139773	1,220618078	0,157900508	1,276221455	0,155643554	1,251903746
2	0,080533121	1,133751295	0,103168612	0,980360693	0,087623016	0,811072422
3	0,057128056	0,892057241	0,085331309	0,771506436	0,08023898	0,609592194
4	0,043378238	0,602653957	0,083673107	0,663255213	0,084047818	0,627767895
5	0,040831053	0,560158615	0,076748574	0,585753222	0,078692593	0,556308106
6	0,034602714	0,435456356	0,095960466	0,631710917	0,098011671	0,511152049
7	0,035397704	0,398073976	0,07999201	0,5604074	0,081237878	0,640307917
8	0,03160555	0,322476462	0,085828457	0,590045393	0,080154686	0,527267324
9	0,039458249	0,318079893	0,086926165	0,588253553	0,084414586	0,555312356
10	0,02869184	0,305178498	0,085784206	0,58767473	0,084412586	0,575549166
11	0,030354971	0,312907682	0,096599801	0,586238322	0,086648209	0,623539745
12	0,028992667	0,291891227	0,097452039	0,670055798	0,088029909	0,59052648

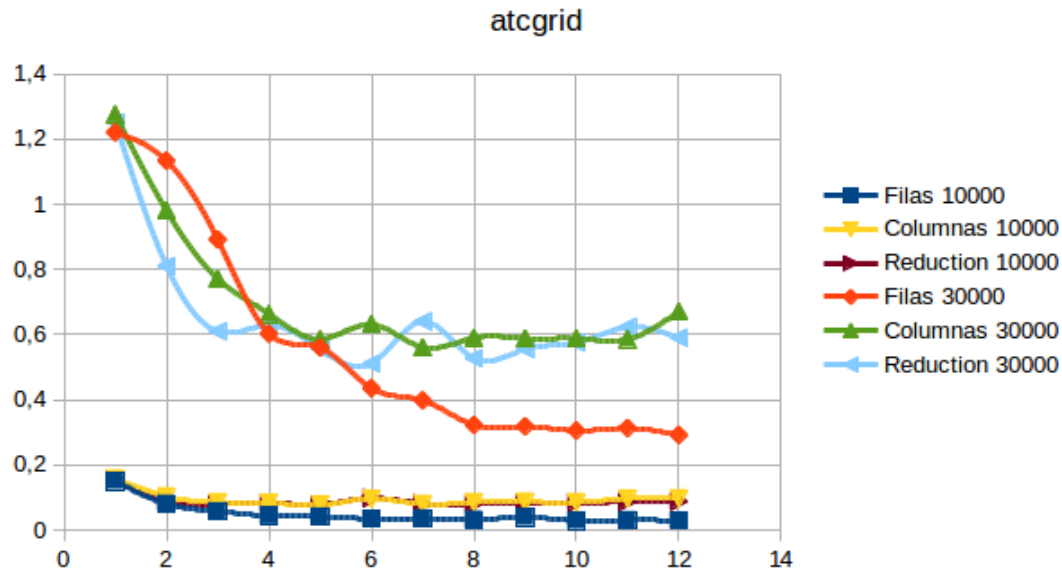
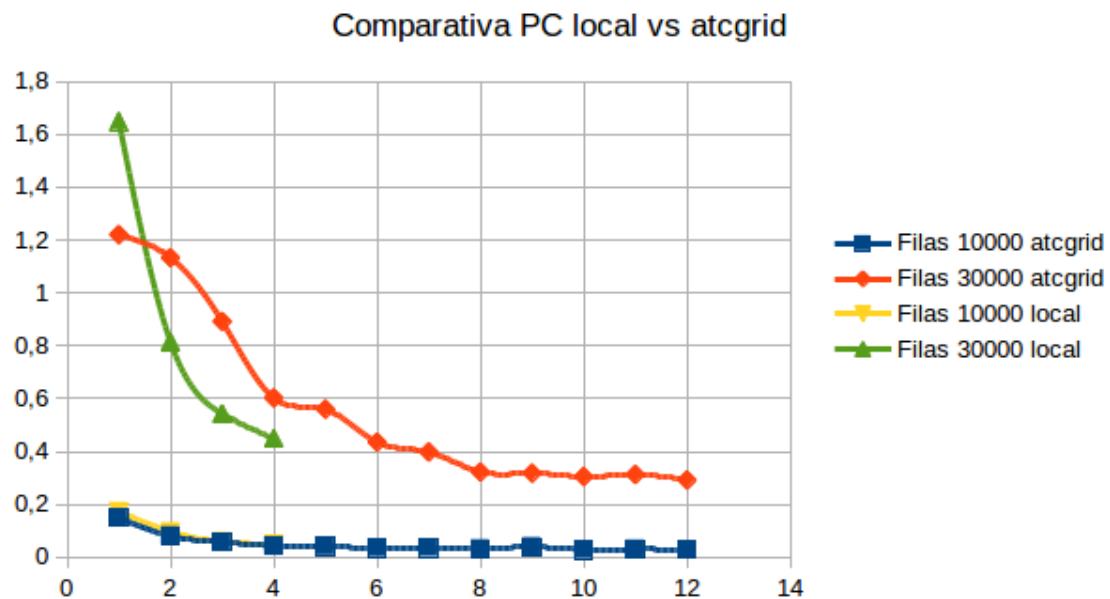


TABLA Y GRÁFICA (por ejemplo para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N-: un N entre 30000 y 100000, y otro entre 5000 y 30000):

n.º de threads	Filas 10000 atcgrid	Filas 30000 atcgrid	Filas 10000 local	Filas 30000 local
1	0,150139773	1,220618078	0,173078043	1,64910402
2	0,080533121	1,133751295	0,096247314	0,814406314
3	0,057128056	0,892057241	0,059513758	0,543392171
4	0,043378238	0,602653957	0,047375058	0,44844282
5	0,040831053	0,560158615		
6	0,034602714	0,435456356		
7	0,035397704	0,398073976		
8	0,03160555	0,322476462		
9	0,039458249	0,318079893		
10	0,02869184	0,305178498		
11	0,030354971	0,312907682		
12	0,028992667	0,291891227		



COMENTARIOS SOBRE LOS RESULTADOS: Como se observa en la gráfica, especialmente de atcgrid por contar con un número significativamente superior de cores, el código que paraleliza las operaciones con filas es más escalable que el código que paraleliza las columnas y el que utiliza la directiva reduction. Por ello, el código elegido para mostrar la comparativa entre mi pc y atcgrid ha sido el código de pmv_OpenMP-a.c que paraleliza las filas.

No hay un resultado claro que permita asegurar que el código que paraleliza las columnas es más eficiente o menos que el código que utiliza reduction.

Vemos que en este caso, en mi pc, la ganancia empírica es mayor que para atcgrid, al menos hasta 4 cores que son con los que se ha realizado esta prueba.

Los cálculos se han hecho para tamaños de 10000 y de 30000, éste último como máximo pues al aumentar el tamaño por encima de esa cifra se ralentizaba demasiado el sistema.

*Para atcgrid se ha utilizado un script que se adjunta en el .zip.