

<p>Grai2º curso / 2º cuatr.</p> <p>Grado Ing. Inform.</p>	<h2>Arquitectura de Computadores (AC)</h2> <h3>Cuaderno de prácticas.</h3> <h3>Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP</h3> <p>Estudiante (nombre y apellidos):</p> <p>Grupo de prácticas y profesor de prácticas:</p> <p>Fecha de entrega:</p> <p>Fecha evaluación en clase:</p>
---	---

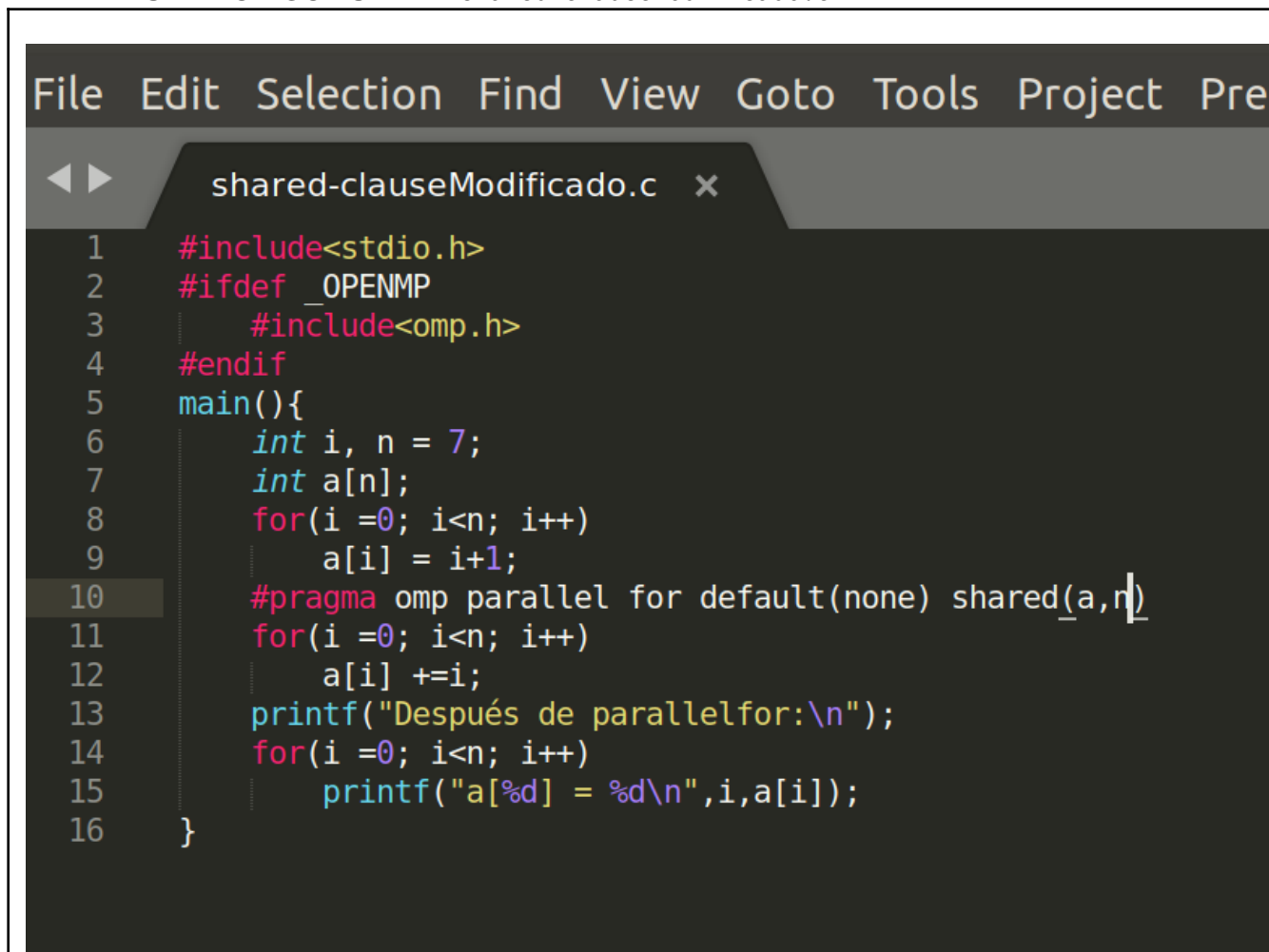
Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

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: Da un error de compilación, y para solucionarlo hay que poner `shared(a,n)`, es decir especificar que se comparten estas variables.

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`



```

File Edit Selection Find View Goto Tools Project Pre
shared-clauseModificado.c x
1  #include<stdio.h>
2  #ifdef _OPENMP
3      #include<omp.h>
4  #endif
5  main(){
6      int i, n = 7;
7      int a[n];
8      for(i=0; i<n; i++)
9          a[i] = i+1;
10     #pragma omp parallel for default(none) shared(a,n)
11     for(i=0; i<n; i++)
12         a[i] +=i;
13     printf("Después de parallelfor:\n");
14     for(i=0; i<n; i++)
15         printf("a[%d] = %d\n",i,a[i]);
16 }

```

CAPTURAS DE PANTALLA:

```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer1] 202
1-05-01 sábado
$
gcc -O2 -fopenmp -o shared shared-clauseModificado.c
shared-clauseModificado.c:5:1: warning: return type defaults to 'int'
[-Wimplicit-int]
main(){
^~~~
shared-clauseModificado.c: In function 'main':
shared-clauseModificado.c:10:10: error: 'n' not specified in enclosing
'parallel'
#pragma omp parallel for default(none) shared(a)
      ^~~
shared-clauseModificado.c:10:10: error: enclosing 'parallel'
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer1] 202
1-05-01 sábado
$
```

```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer1] 202
1-05-01 sábado
$
gcc -O2 -fopenmp -o shared shared-clauseModificado.c
shared-clauseModificado.c:5:1: warning: return type defaults to 'int'
[-Wimplicit-int]
main(){
^~~~
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer1] 202
1-05-01 sábado
$
./shared
Después de parallelfor:
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 7
a[4] = 9
a[5] = 11
a[6] = 13
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer1] 202
1-05-01 sábado
$
```

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:** Cada hebra hacen su suma y las sumas pierden el valor cuando salen del `parallel`

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

```

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

```

CAPTURAS DE PANTALLA:

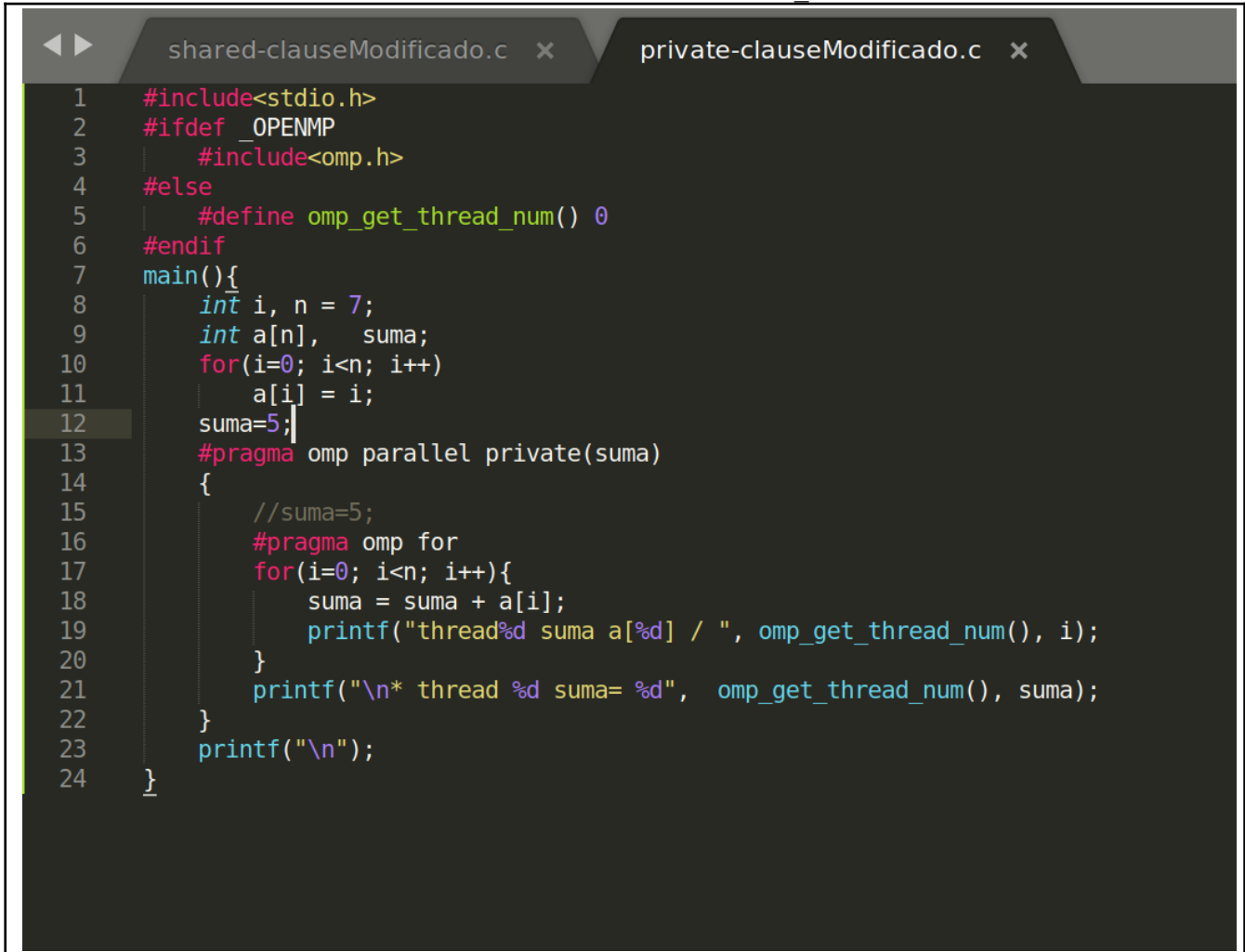
```

[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer2] 202
1-05-01 sábado
$
./private
thread1 suma a[2] / thread1 suma a[3] / thread2 suma a[4] / thread2 s
uma a[5] / thread0 suma a[0] / thread0 suma a[1] / thread3 suma a[6]
/
* thread 0 suma= 6
* thread 3 suma= 11
* thread 1 suma= 10
* thread 2 suma= 14
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer2] 202
1-05-01 sábado
$

```

(b) **RESPUESTA:** Cada suma en cada thread se desborda

CAPTURA CÓDIGO FUENTE: private-clauseModificado_b.c



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

CAPTURAS DE PANTALLA:

```

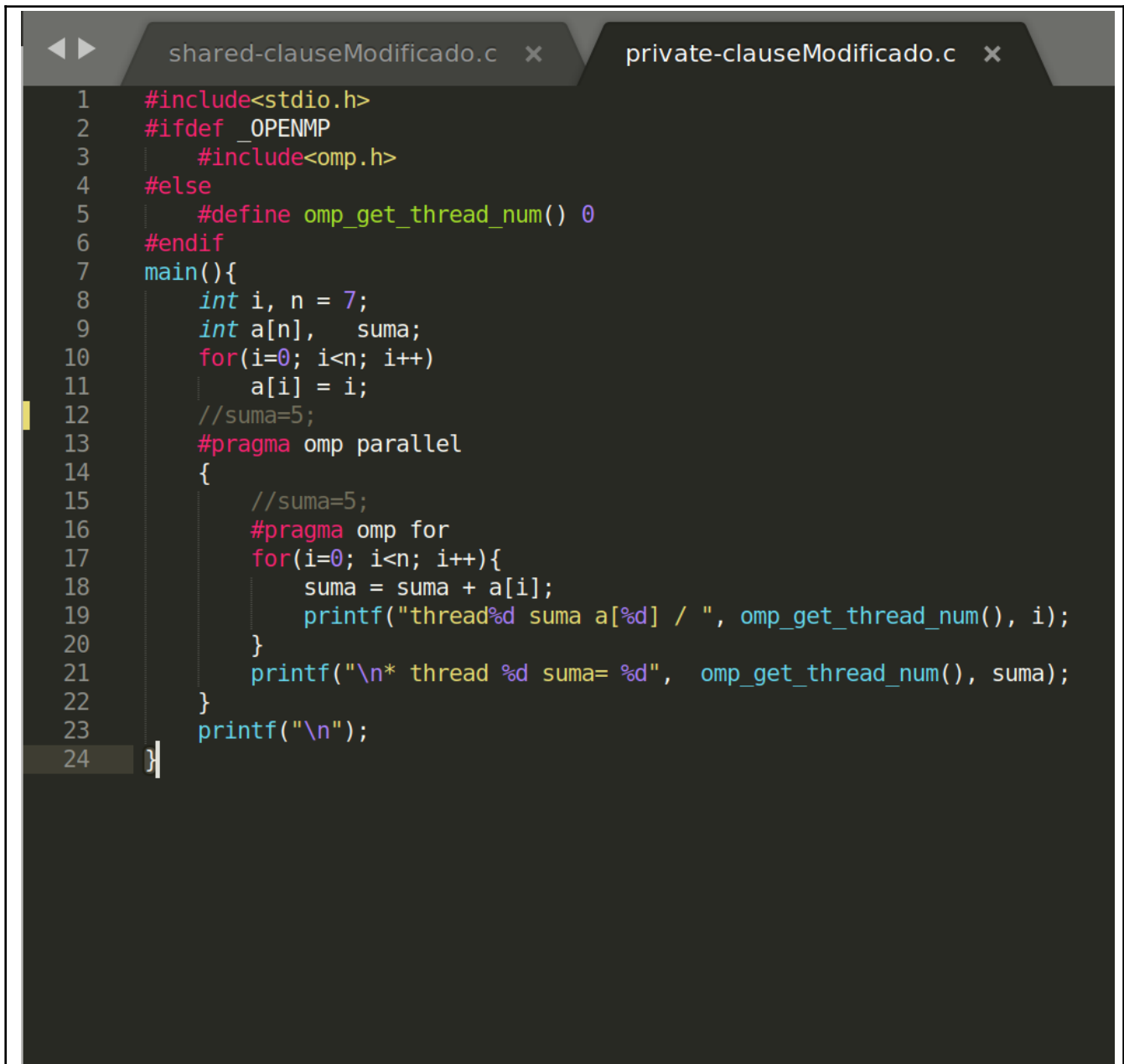
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer2] 202
1-05-01 sábado
$
gcc -O2 -fopenmp -o private private-clauseModificado.c
private-clauseModificado.c:7:1: warning: return type defaults to 'int'
[-Wimplicit-int]
main(){
^~~~
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer2] 202
1-05-01 sábado
$
./private
thread0 suma a[0] / thread0 suma a[1] / thread1 suma a[2] / thread1 s
uma a[3] / thread3 suma a[6] / thread2 suma a[4] / thread2 suma a[5]
/
* thread 2 suma= -1530158247
* thread 1 suma= -1530158251
* thread 3 suma= -1530158250
* thread 0 suma= -1543210911
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer2] 202
1-05-01 sábado
$

```

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

RESPUESTA: La variable `suma` es compartida si le quitamos el `private`, por lo que se almacena todas las sumas de las hebras en ella

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`



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

CAPTURAS DE PANTALLA:

```

[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer3] 202
1-05-01 sábado
$
./private
thread0 suma a[0] / thread0 suma a[1] / thread2 suma a[4] / thread2 s
uma a[5] / thread1 suma a[2] / thread1 suma a[3] / thread3 suma a[6]
/
* thread 1 suma= 21
* thread 0 suma= 21
* thread 2 suma= 21
* thread 3 suma= 21
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer3] 202
1-05-01 sábado
$
./private
thread0 suma a[0] / thread0 suma a[1] / thread1 suma a[2] / thread1 s
uma a[3] / thread2 suma a[4] / thread2 suma a[5] / thread3 suma a[6]
/
* thread 1 suma= 19
* thread 2 suma= 19
* thread 0 suma= 19
* thread 3 suma= 19
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer3] 202
1-05-01 sábado
$
./private
thread0 suma a[0] / thread0 suma a[1] / thread3 suma a[6] / thread2 s
uma a[4] / thread2 suma a[5] / thread1 suma a[2] / thread1 suma a[3]
/

```

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: Aumenta el tamaño del vector por tanto el tamaño de la suma también.

CAPTURAS DE PANTALLA:

```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer4] 202
1-05-02 domingo
$./first
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 3 suma a[6] suma=6
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9

Fuerade la construcción parallelsoma=0
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer4] 202
1-05-02 domingo
$./first
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 3 suma a[6] suma=6
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5

Fuerade la construcción parallelsoma=0
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer4] 202
1-05-02 domingo
$./first
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 3 suma a[6] suma=6
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9

Fuerade la construcción parallelsoma=0
```

(b) RESPUESTA: Como se reinicia el valor de la suma con cada ejecución la suma siempre vuelve al mismo valor.

CAPTURAS

DE

PANTALLA:


```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer4] 202
1-05-02 domingo
$./first
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 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

Fuerade la construcción parallelsuma=0
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer4] 202
1-05-02 domingo
$./first
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 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

Fuerade la construcción parallelsuma=0
```

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: La cláusula `copyprivate` hace que se copie el valor de `a` en todas las hebras de forma privada por lo que cuando lo quitamos solo se guarda en algunas hebras.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```

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

```

CAPTURAS DE PANTALLA:

```

[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer5] 202
1-05-02 domingo
$gcc -O2 -fopenmp -o copy copyprivate-clause.c
copyprivate-clause.c:3:1: warning: return type defaults to 'int' [-Wimplicit-int]
main(){
^~~~
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer5] 202
1-05-02 domingo
$./copy

Introduce valor de inicialización a: 5

Single ejecutada por el thread3
Depuésde la región parallel:
b[0] = 5      b[1] = 5      b[2] = 5      b[3] = 5      b[4]
= 5      b[5] = 5      b[6] = 5      b[7] = 5      b[8] = 5
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer5] 202
1-05-02 domingo
$gcc -O2 -fopenmp -o copy copyprivate-clause.c
copyprivate-clause.c:3:1: warning: return type defaults to 'int' [-Wimplicit-int]
main(){
^~~~
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer5] 202
1-05-02 domingo
$./copy

Introduce valor de inicialización a: 5

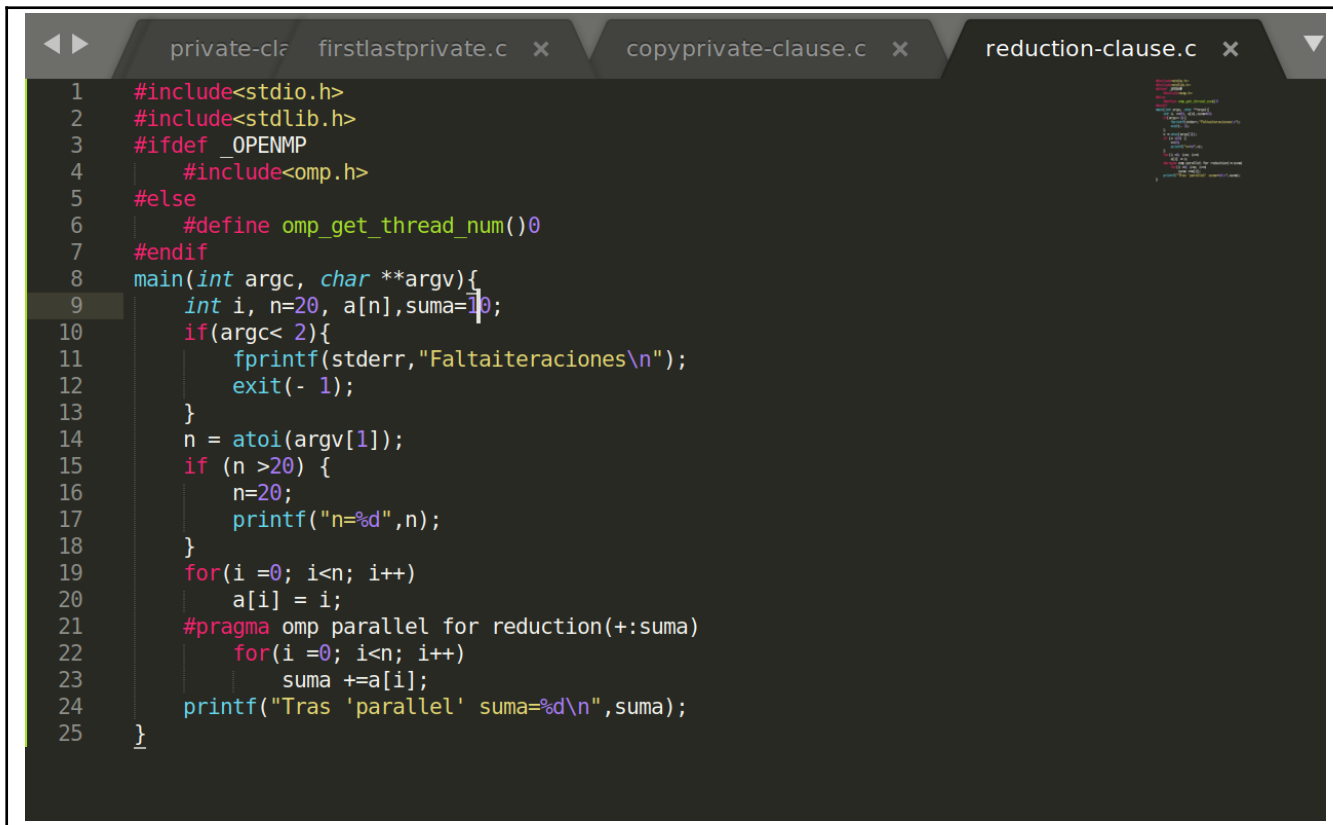
Single ejecutada por el thread2
Depuésde la región parallel:
b[0] = 0      b[1] = 0      b[2] = 0      b[3] = 0      b[4]
= 0      b[5] = 5      b[6] = 5      b[7] = 0      b[8] = 0
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer5] 202
1-05-02 domingo
$

```

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: Imprime el resultado más diez (que es el valor al que lo hemos inicializado), por ello la cláusula especifica que se inicialice a 0, ya que imprime la suma + el valor de inicio.

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_thread_num()0
7  #endif
8  main(int argc, char **argv){
9      int i, n=20, a[n], suma=10;
10     if(argc< 2){
11         fprintf(stderr, "Faltaiteraciones\n");
12         exit(- 1);
13     }
14     n = atoi(argv[1]);
15     if (n >20) {
16         n=20;
17         printf("n=%d", n);
18     }
19     for(i =0; i<n; i++)
20         a[i] = i;
21     #pragma omp parallel for reduction(+:suma)
22     for(i =0; i<n; i++)
23         suma +=a[i];
24     printf("Tras 'parallel' suma=%d\n", suma);
25 }

```

CAPTURAS DE PANTALLA:

```

[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer6] 202
1-05-02 domingo
$gcc -O2 -fopenmp -o reduction reduction-clause.c
reduction-clause.c:8:1: warning: return type defaults to 'int' [-Wimplicit-int]
  main(int argc, char **argv){
  ^~~~
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer6] 202
1-05-02 domingo
$./reduction
Falta iteraciones
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer6] 202
1-05-02 domingo
$./reduction 10
Tras 'parallel' suma=55
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer6] 202
1-05-02 domingo
$./reduction 10
Tras 'parallel' suma=55
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer6] 202
1-05-02 domingo
$./reduction 20
Tras 'parallel' suma=200
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer6] 202
1-05-02 domingo
$./reduction 30
n=20 Tras 'parallel' suma=200
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer6] 202
1-05-02 domingo
$./reduction 5
Tras 'parallel' suma=20
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer6] 202
1-05-02 domingo
$

```

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: Al quitar `reduction` tenemos que añadir `atomic`, para evitar que se utilice a la vez la variable, así pueden escribir y utilizar su valor todas las hebras sin dar valores nulos o indefinidos.

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_thread_num()0
7  #endif
8  main(int argc, char **argv){
9      int i, n=20, a[n], suma=10;
10     if(argc< 2){
11         fprintf(stderr, "Faltaiteraciones\n");
12         exit(- 1);
13     }
14     n = atoi(argv[1]);
15     if (n >20) {
16         n=20;
17         printf("n=%d",n);
18     }
19     for(i=0; i<n; i++)
20         a[i] = i;
21     #pragma omp parallel for //reduction(+:suma)
22     for(i=0; i<n; i++)
23         #pragma omp atomic
24         suma +=a[i];
25     printf("Tras 'parallel' suma=%d\n",suma);
26 }

```

CAPTURAS DE PANTALLA:

```
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer7] 202
1-05-02 domingo
$gcc -O2 -fopenmp -o reduction reduction-clauseModificado.c
reduction-clauseModificado.c:8:1: warning: return type defaults to 'i
nt' [-Wimplicit-int]
main(int argc, char **argv){
^~~~
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer7] 202
1-05-02 domingo
$./reduction 10
Tras 'parallel' suma=55
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer7] 202
1-05-02 domingo
$./reduction 20
Tras 'parallel' suma=200
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer7] 202
1-05-02 domingo
$./reduction 30
n=20Tras 'parallel' suma=200
[AmadorCarmonaMendez amador@MacBookPro:~/Escritorio/AC/BP2/ejer7] 202
1-05-02 domingo
$
```

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):

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

NOTAS: (1) el número de filas /columnas N de la matriz deben ser argumentos de entrada al programa; (2) se debe inicializar la matriz y el vector antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, **v3**, para tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que calcula el producto matriz vector y, al menos, el primer y último componente del resultado (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

CAPTURA CÓDIGO FUENTE: pmv-secuencial.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <malloc.h>
5
6  #define GLOBAL
7
8  #ifdef GLOBAL
9      #define MAX 33554432
10 #endif
11
12 int main(int argc, char const *argv[]){
13     if(argc != 2){
14         printf("Faltan argumentos %s", argv[0]);
15         return(EXIT_FAILURE);
16     }
17
18     struct timespec cgt1, cgt2;
19     double ncgt;
20     int N = atoi(argv[1]);
21
22     #ifdef GLOBAL
23         if(N > MAX) N = MAX;
24         int matriz[N][N];
25         int vector[N];
26         int vector_resultado[N];
27         printf("Ejecutado GLOBAL\n");
28     #endif
29
30     #ifdef DINAMIC
31         int **matriz, *vector, *vector_resultado;
32         matriz = (int**) malloc(N * sizeof(int*));
33         for(int i = 0; i < N; ++i)
34             matriz[i] = (int*) malloc(N * sizeof(int));
35
36         vector = (int*) malloc(N * sizeof(int));
37         vector_resultado = (int*) malloc(N * sizeof(int));
38         printf("Ejecutado DINAMICO\n");
39     #endif
40
41     for(int i = 0; i < N; ++i){
42         vector[i] = i;
43         for(int j = 0; j < N; ++j)
44             matriz[i][j] = i + j;
45     }

```



```

56     clock_gettime(CLOCK_REALTIME, &cgt2);
57     ncgt = (double) (cgt2.tv_sec - cgt1.tv_sec) + (double) (cgt2.tv_nsec - cgt1.
58
59     printf("Tiempo(seg.): %11.9f\t / Tamaño vectores: %u\n", ncgt, N);
60     if(N < 15)
61         for(int i = 0; i < N; i++){
62             printf("VECTOR_RESULTADO[%d] = %d ", i, vector_resultado[i]);
63             printf("\n");
64         }
65     else{
66         printf("VECTOR_RESULTADO[0] = %d ", vector_resultado[0]);
67         printf("VECTOR_RESULTADO[%d] = %d ", N - 1, vector_resultado[N - 1]);
68     }
69
70     #ifdef DINAMIC
71     for(int i = 0; i < N; i++)
72         free(matriz[i]);
73
74     free(matriz); free(vector); free(vector_resultado);
75     #endif
76
77     return 0;
78 }

```

CAPTURAS DE PANTALLA:

```

[AmadorCarmonaMendez b1estudiante4@atcgrid:~/BP2] 2021-05-02 domingo
$cd ejer8
[AmadorCarmonaMendez b1estudiante4@atcgrid:~/BP2/ejer8] 2021-05-02 domingo
$srn ./pmv 8
Ejecutado GLOBAL
Tiempo(seg.): 0.000000300 / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140 VECTOR_RESULTADO[1] = 168 VECTOR_RESULTADO[2] = 196 VE
CTOR_RESULTADO[3] = 224 VECTOR_RESULTADO[4] = 252 VECTOR_RESULTADO[5] = 280 VECT
OR_RESULTADO[6] = 308 VECTOR_RESULTADO[7] = 336 [AmadorCarmonaMendez b1estudiante
4@atcgrid:~/BP2/ejer8] 2021-05-02 domingo
$srn ./pmv 11
Ejecutado GLOBAL
Tiempo(seg.): 0.000000433 / Tamaño vectores: 11
VECTOR_RESULTADO[0] = 385 VECTOR_RESULTADO[1] = 440 VECTOR_RESULTADO[2] = 495 VE
CTOR_RESULTADO[3] = 550 VECTOR_RESULTADO[4] = 605 VECTOR_RESULTADO[5] = 660 VECT
OR_RESULTADO[6] = 715 VECTOR_RESULTADO[7] = 770 VECTOR_RESULTADO[8] = 825 VECTOR
_RESULTADO[9] = 880 VECTOR_RESULTADO[10] = 935 [AmadorCarmonaMendez b1estudiante4
@atcgrid:~/BP2/ejer8] 2021-05-02 domingo
$

```

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

```
#pragma omp parallel for
for(int i = 0; i < N; ++i){
    vector[i] = i;
    #pragma omp parallel for
    for(int j = 0; j < N; ++j)
        matriz[i][j] = i + j;
}

clock_gettime(CLOCK_REALTIME, &cgt1);
#pragma omp parallel for
for(int i = 0; i < N; i++){
    int suma = 0;
    for(int j = 0; j < N; j++)
        suma += matriz[i][j] * vector[j];

    vector_resultado[i] = suma;
}
```

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

```
#pragma omp parallel for
for(int i = 0; i < N; ++i){
    vector[i] = i;
    #pragma omp parallel for
    for(int j = 0; j < N; ++j)
        matriz[i][j] = i + j;
}

clock_gettime(CLOCK_REALTIME, &cgt1);
for(int i = 0; i < N; i++){
    int suma = 0;
    #pragma omp parallel for
    for(int j = 0; j < N; j++)
        #pragma omp atomic
        suma += matriz[i][j] * vector[j];

    vector_resultado[i] = suma;
}
```

RESPUESTA: Los errores de compilación que he tenido han sido los típicos de punto y coma y algún error en la sintaxis, si es verdad que se me olvidó el atomic en el pmv-b.c y eso si me dio errores en la ejecución pero luego encontré el error y lo solucioné.

CAPTURAS

DE

PANTALLA:

```
[AmadorCarmonaMendez b1estudiante4@atcgrid:~/BP2/ejer9] 2021-05-02 domingo
$srn ./pmv-a 8
Ejecutado GLOBAL
Tiempo(seg.): 0.000003080 / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140 VECTOR_RESULTADO[1] = 168 VECTOR_RESULTADO[2] = 196 VE
CTOR_RESULTADO[3] = 224 VECTOR_RESULTADO[4] = 252 VECTOR_RESULTADO[5] = 280 VECT
OR_RESULTADO[6] = 308 VECTOR_RESULTADO[7] = 336 [AmadorCarmonaMendez b1estudiante
4@atcgrid:~/BP2/ejer9] 2021-05-02 domingo
$srn ./pmv-a 11
Ejecutado GLOBAL
Tiempo(seg.): 0.000003310 / Tamaño vectores: 11
VECTOR_RESULTADO[0] = 385 VECTOR_RESULTADO[1] = 440 VECTOR_RESULTADO[2] = 495 VE
CTOR_RESULTADO[3] = 550 VECTOR_RESULTADO[4] = 605 VECTOR_RESULTADO[5] = 660 VECT
OR_RESULTADO[6] = 715 VECTOR_RESULTADO[7] = 770 VECTOR_RESULTADO[8] = 825 VECTOR
_RESULTADO[9] = 880 VECTOR_RESULTADO[10] = 935 [AmadorCarmonaMendez b1estudiante4
@atcgrid:~/BP2/ejer9] 2021-05-02 domingo
$srn ./pmv-b 8
Ejecutado DINAMICO
Tiempo(seg.): 0.000019197 / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140 VECTOR_RESULTADO[1] = 168 VECTOR_RESULTADO[2] = 196 VE
CTOR_RESULTADO[3] = 224 VECTOR_RESULTADO[4] = 252 VECTOR_RESULTADO[5] = 280 VECT
OR_RESULTADO[6] = 308 VECTOR_RESULTADO[7] = 336 [AmadorCarmonaMendez b1estudiante
4@atcgrid:~/BP2/ejer9] 2021-05-02 domingo
$srn ./pmv-b 11
Ejecutado DINAMICO
Tiempo(seg.): 0.000014440 / Tamaño vectores: 11
VECTOR_RESULTADO[0] = 385 VECTOR_RESULTADO[1] = 440 VECTOR_RESULTADO[2] = 495 VE
CTOR_RESULTADO[3] = 550 VECTOR_RESULTADO[4] = 605 VECTOR_RESULTADO[5] = 660 VECT
OR_RESULTADO[6] = 715 VECTOR_RESULTADO[7] = 770 VECTOR_RESULTADO[8] = 825 VECTOR
_RESULTADO[9] = 880 VECTOR_RESULTADO[10] = 935 [AmadorCarmonaMendez b1estudiante4
@atcgrid:~/BP2/ejer9] 2021-05-02 domingo
$
```

```

[b1estudiante4@atcgrid BP2]$ cd ejer9
[b1estudiante4@atcgrid ejer9]$ srun ./pmv-a 8
Ejecutado GLOBAL
Tiempo(seg.): 0.000003138 / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140 VECTOR_RESULTADO[1] = 168 VECTOR_RESULTADO[2] = 196 VE
CTOR_RESULTADO[3] = 224 VECTOR_RESULTADO[4] = 252 VECTOR_RESULTADO[5] = 280 VECT
OR_RESULTADO[6] = 308 VECTOR_RESULTADO[7] = 336 [b1estudiante4@atcgrid ejer9]$ sr
un ./pmv-a 11
Ejecutado GLOBAL
Tiempo(seg.): 0.000003182 / Tamaño vectores: 11
VECTOR_RESULTADO[0] = 385 VECTOR_RESULTADO[1] = 440 VECTOR_RESULTADO[2] = 495 VE
CTOR_RESULTADO[3] = 550 VECTOR_RESULTADO[4] = 605 VECTOR_RESULTADO[5] = 660 VECT
OR_RESULTADO[6] = 715 VECTOR_RESULTADO[7] = 770 VECTOR_RESULTADO[8] = 825 VECTOR
_RESULTADO[9] = 880 VECTOR_RESULTADO[10] = 935 [b1estudiante4@atcgrid ejer9]$ sru
n ./pmv-b 8
Ejecutado DINAMICO
Tiempo(seg.): 0.000018379 / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140 VECTOR_RESULTADO[1] = 168 VECTOR_RESULTADO[2] = 196 VE
CTOR_RESULTADO[3] = 224 VECTOR_RESULTADO[4] = 252 VECTOR_RESULTADO[5] = 280 VECT
OR_RESULTADO[6] = 308 VECTOR_RESULTADO[7] = 336 [b1estudiante4@atcgrid ejer9]$ sr
un ./pmv-b 11
Ejecutado DINAMICO
Tiempo(seg.): 0.000025609 / Tamaño vectores: 11
VECTOR_RESULTADO[0] = 385 VECTOR_RESULTADO[1] = 440 VECTOR_RESULTADO[2] = 495 VE
CTOR_RESULTADO[3] = 550 VECTOR_RESULTADO[4] = 605 VECTOR_RESULTADO[5] = 660 VECT
OR_RESULTADO[6] = 715 VECTOR_RESULTADO[7] = 770 VECTOR_RESULTADO[8] = 825 VECTOR
_RESULTADO[9] = 880 VE[b1e[b1e[b1e[b1e[b1e[b1estud[b1e[b1e[b1estud[b1e[b1e[b1e[b1e

```

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`

```

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

clock_gettime(CLOCK_REALTIME, &cgt1);
for(int i = 0; i < N; i++){
    int suma = 0;
    #pragma omp parallel for reduction(+:suma)
    for(int j = 0; j < N; j++)
        // #pragma omp atomic
        suma += matriz[i][j] * vector[j];

    vector_resultado[i] = suma;
}

```

RESPUESTA: No me ha surgido ningún problema ni de ejecución ni de compilación, ya que solamente he añadido al código anterior ya depurado la clausula `reduction(+:suma)` y he comentado la directiva `atomic`.

CAPTURAS**DE****PANTALLA:**

```

[AmadorCarmonaMendez b1estudiante4@atcgrid:~/BP2] 2021-05-02 domingo
$cd ejer10
[AmadorCarmonaMendez b1estudiante4@atcgrid:~/BP2/ejer10] 2021-05-02 domingo
$srn ./pmv 8
Ejecutado DINAMICO
Tiempo(seg.): 0.000018824 / Tamaño vectores: 8
VECTOR_RESULTADO[0] = 140 VECTOR_RESULTADO[1] = 168 VECTOR_RESULTADO[2] = 196 VE
CTOR_RESULTADO[3] = 224 VECTOR_RESULTADO[4] = 252 VECTOR_RESULTADO[5] = 280 VECT
OR_RESULTADO[6] = 308 VECTOR_RESULTADO[7] = 336 [AmadorCarmonaMendez b1estudiante
4@atcgrid:~/BP2/ejer10] 2021-05-02 domingo
$srn ./pmv 11
Ejecutado DINAMICO
Tiempo(seg.): 0.000025167 / Tamaño vectores: 11
VECTOR_RESULTADO[0] = 385 VECTOR_RESULTADO[1] = 440 VECTOR_RESULTADO[2] = 495 VE
CTOR_RESULTADO[3] = 550 VECTOR_RESULTADO[4] = 605 VECTOR_RESULTADO[5] = 660 VECT
OR_RESULTADO[6] = 715 VECTOR_RESULTADO[7] = 770 VECTOR_RESULTADO[8] = 825 VECTOR
_RESULTADO[9] = 880 VECTOR_RESULTADO[10] = 935 [AmadorCarmonaMendez b1estudiante4
@atcgrid:~/BP2/ejer10] 2021-05-02 domingo

```

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):

JUSTIFICAR AHORA EN BASE AL CÓDIGO LA DIFERENCIA EN TIEMPOS:

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

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

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

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)
Código Secuencial		----		----		----		----		----		----
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
32												

COMENTARIOS SOBRE LOS RESULTADOS: