

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): Juan Carlos Ruiz Fernández

Grupo de prácticas y profesor de prácticas: B3 Mancia Anguita

Fecha de entrega: 27/04/21

Fecha evaluación en clase: 28/04/21

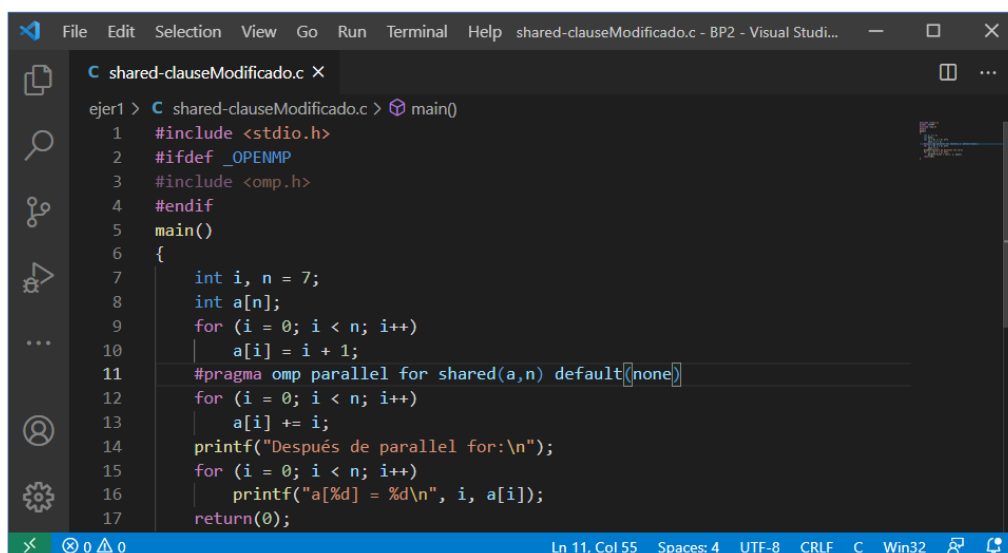
#### Ejercicios basados en los ejemplos del seminario práctico

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:

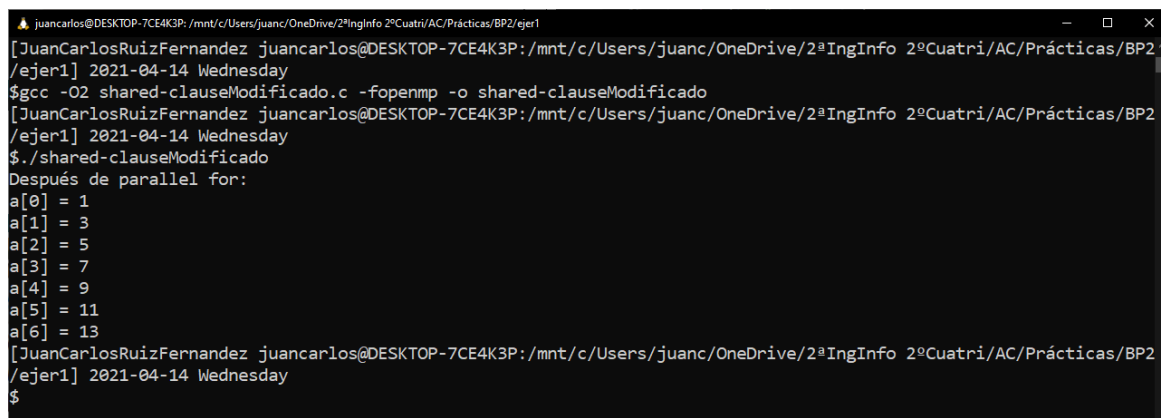
Default(none) provoca que las variables creadas fuera del `parallel` no se compartan, por lo que en al compilar provoca error de que `n` no esta especificada en la sección. Para solucionarlo se añade a variable `n` en la directiva `shared`.

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



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

#### CAPTURAS DE PANTALLA:



```
juancarlos@DESKTOP-7CE4K3P: /mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer1
[juancarlos@DESKTOP-7CE4K3P: /mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer1] 2021-04-14 Wednesday
$gcc -O2 shared-clauseModificado.c -fopenmp -o shared-clauseModificado
[juancarlos@DESKTOP-7CE4K3P: /mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer1] 2021-04-14 Wednesday
$./shared-clauseModificado
Después de parallel for:
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 7
a[4] = 9
a[5] = 11
a[6] = 13
[juancarlos@DESKTOP-7CE4K3P: /mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer1] 2021-04-14 Wednesday
$
```

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:

**Private** hace que cada thread tenga una copia de la variable. Como hemos puesto que *suma* se imprima fuera del `parallel` no representará ningún cabio realizado dentro del `parallel`. Pues obtenemos en varias ejecuciones el valor 0.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado_a.c`

```

1  File Edit Selection View Go Run Terminal Help private-clause_a.c - BP2 - Visual Studio Co...
2  C private-clause_a.c X C private-clause_b.c
3  ejer2 > C private-clause_a.c > main()
4
5  9 int i, n = 7;
6  10 int a[n], suma;
7  11 for (i = 0; i < n; i++)
8  12     a[i] = i;
9  13 #pragma omp parallel private(suma)
10 14 {
11 15     suma = 5;
12 16     #pragma omp for
13 17     for (i = 0; i < n; i++)
14 18     {
15 19         suma = suma + a[i];
16 20         printf( "thread %d suma a[%d] / ", omp_get_thread_num(), i);
17 21     }
18 22 }
19 23 printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
20 24 printf("\n");
21 25 return(0);
22
23 Ln 15, Col 18 Spaces: 4 UTF-8 CRLF C Win32

```

CAPTURAS DE PANTALLA:

```

1  Seleccionar juancarlos@DESKTOP-7CE4K3P: /mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer1
2
3  [JuanCarlosRuizFernandez juancarlos@DESKTOP-7CE4K3P:/mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer2] 2021-04-14 Wednesday
4  $gcc -O2 -fopenmp private-clause_a.c -o private-clause_a
5  [JuanCarlosRuizFernandez juancarlos@DESKTOP-7CE4K3P:/mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer2] 2021-04-14 Wednesday
6  $./private-clause_a
7  thread 1 suma a[1] / thread 5 suma a[5] / thread 4 suma a[4] / thread 6 suma a[6] / thread 0 suma a[0] / thread 3 suma a[3] / thread 2 suma a[2] /
8  * thread 0 suma= 0
9  [JuanCarlosRuizFernandez juancarlos@DESKTOP-7CE4K3P:/mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer2] 2021-04-14 Wednesday
10 $./private-clause_a
11 thread 4 suma a[4] / thread 5 suma a[5] / thread 1 suma a[1] / thread 3 suma a[3] / thread 6 suma a[6] / thread 2 suma a[2] / thread 0 suma a[0] /
12 * thread 0 suma= 0
13 [JuanCarlosRuizFernandez juancarlos@DESKTOP-7CE4K3P:/mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer2] 2021-04-14 Wednesday
14 $./private-clause_a
15 thread 0 suma a[0] / thread 1 suma a[1] / thread 3 suma a[3] / thread 2 suma a[2] / thread 5 suma a[5] / thread 6 suma a[6] / thread 4 suma a[4] /
16 * thread 0 suma= 0
17 [JuanCarlosRuizFernandez juancarlos@DESKTOP-7CE4K3P:/mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer2] 2021-04-14 Wednesday
18 $

```

**(b) RESPUESTA:**

De igual manera obtenemos reiteradamente el mismo valor, que en este caso es el mismo al que hemos inicializado suma fuera del parallel. Esto se debe a que hacer private, como he dicho antes, cada thread utiliza *suma* como variable local propia haciendo que al terminar la sección parallel “no se guarde ningún cambio” teniendo *suma* el valor que le hemos inicializado.

**CAPTURA CÓDIGO FUENTE:** private-clauseModificado\_b.c

```

File Edit Selection View Go Run Terminal Help private-clause_b.c - BP2 - Visual Studio Co...
C private-clause_b.c X
ejer2 > C private-clause_b.c > main()
9   int i, n = 7;
10  int a[n], suma;
11  for (i = 0; i < n; i++)
12  |   a[i] = i;
13  suma = 7;
14  #pragma omp parallel private(suma)
15  {
16      #pragma omp for
17      for (i = 0; i < n; i++)
18      {
19          suma = suma + a[i];
20          printf( "thread %d suma a[%d] / ", omp_get_thread_num(), i);
21      }
22  }
23  printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
24  printf("\n");
25  return(0);
Ln 13, Col 14 Spaces: 4 UTF-8 CRLF C Win32

```

**CAPTURAS DE PANTALLA:**

```

juancarlos@DESKTOP-7CE4K3P: /mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer1
$gcc -O2 -fopenmp private-clause_b.c -o private-clause_b
[juancarlos@DESKTOP-7CE4K3P: /mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer2] 2021-04-14 Wednesday
$./private-clause_b
thread 5 suma a[5] / thread 1 suma a[1] / thread 2 suma a[2] / thread 4 suma a[4] / thread 0 suma a[0] / thread 6 suma a[6] / thread 3 suma a[3] /
* thread 0 suma= 7
[juancarlos@DESKTOP-7CE4K3P: /mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer2] 2021-04-14 Wednesday
$./private-clause_b
thread 3 suma a[3] / thread 5 suma a[5] / thread 1 suma a[1] / thread 6 suma a[6] / thread 4 suma a[4] / thread 2 suma a[2] / thread 0 suma a[0] /
* thread 0 suma= 7
[juancarlos@DESKTOP-7CE4K3P: /mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer2] 2021-04-14 Wednesday
$./private-clause_b
thread 3 suma a[3] / thread 1 suma a[1] / thread 5 suma a[5] / thread 4 suma a[4] / thread 6 suma a[6] / thread 0 suma a[0] / thread 2 suma a[2] /
* thread 0 suma= 7
[juancarlos@DESKTOP-7CE4K3P: /mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer2] 2021-04-14 Wednesday
$

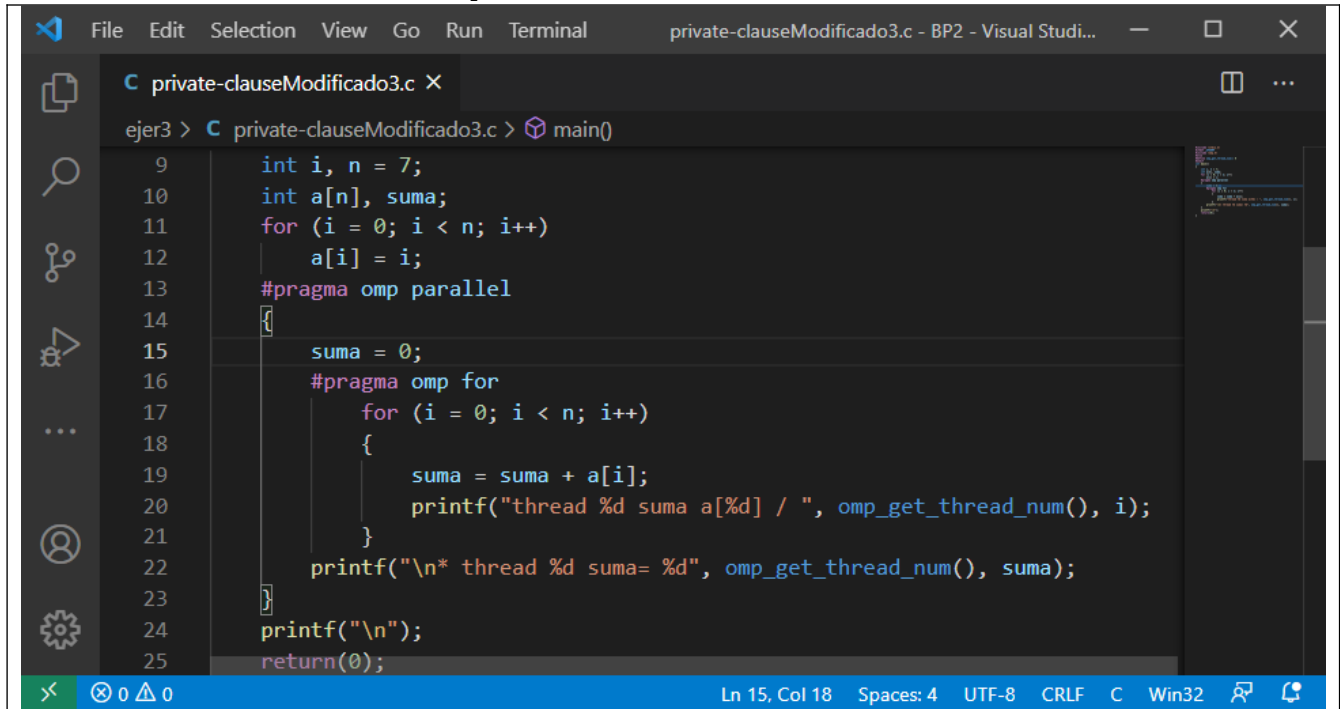
```

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 variable `suma` es compartida entre todas las hebras, produciendo cierto problema de carrera y los resultados se pisen los unos con los otros. Para la ejecución mostrada se ve todos los resultados como ceros, en otras ejecuciones salen todo tres, todo seis, todo cuatro... depende de la hebra ultima se haya ejecutado y seteado la variable

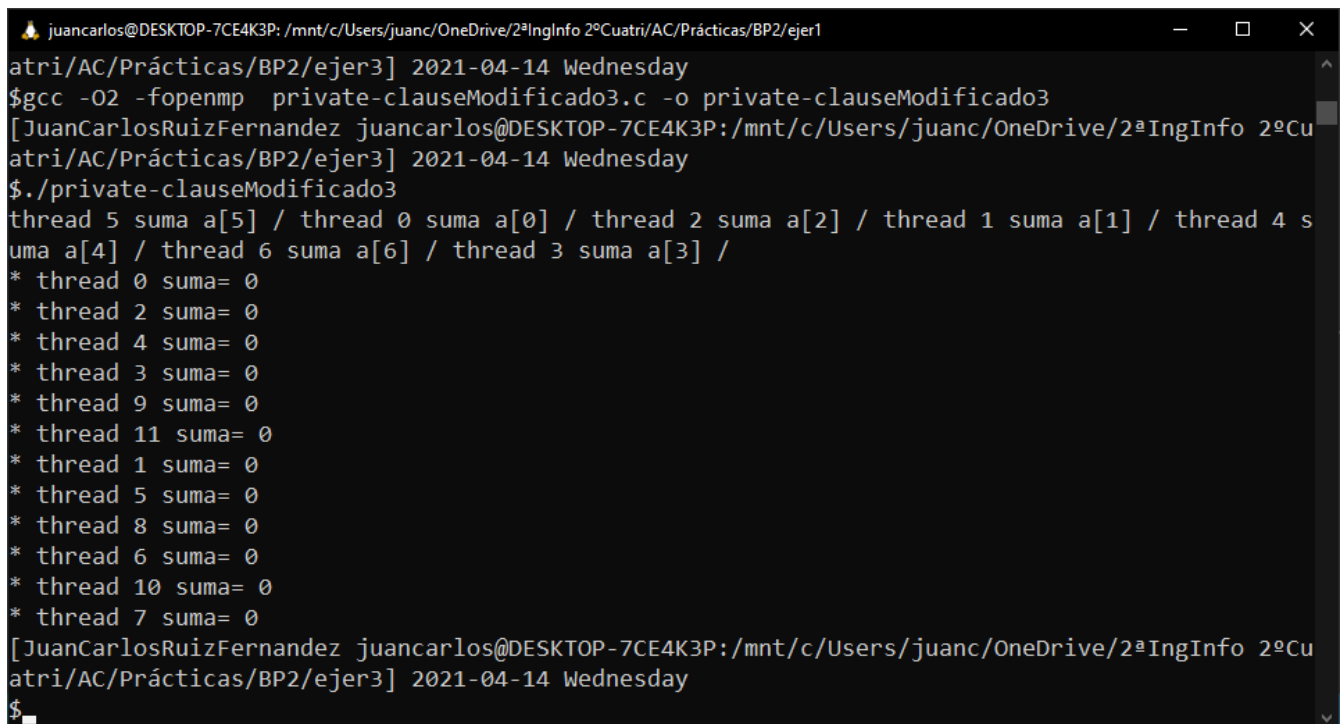
**CAPTURA CÓDIGO FUENTE:** `private-clauseModificado3.c`



```

9      int i, n = 7;
10     int a[n], suma;
11     for (i = 0; i < n; i++)
12         a[i] = i;
13     #pragma omp parallel
14     {
15         suma = 0;
16         #pragma omp for
17         for (i = 0; i < n; i++)
18         {
19             suma = suma + a[i];
20             printf("thread %d suma a[%d] / ", omp_get_thread_num(), i);
21         }
22         printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
23     }
24     printf("\n");
25     return(0);
  
```

**CAPTURAS DE PANTALLA:**

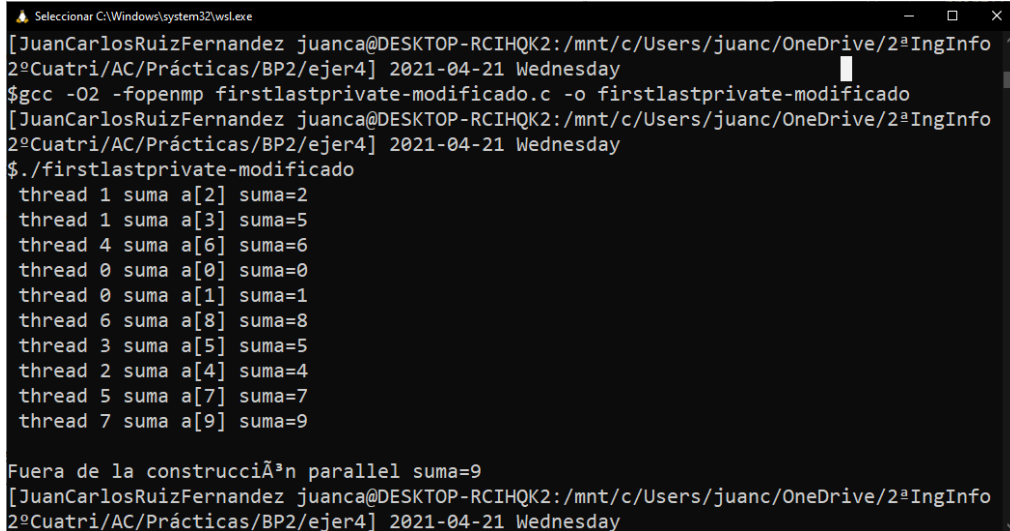


```

juancarlos@DESKTOP-7CE4K3P: /mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer1
[JuanCarlosRuizFernandez juancarlos@DESKTOP-7CE4K3P: /mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer1] 2021-04-14 Wednesday
$gcc -O2 -fopenmp private-clauseModificado3.c -o private-clauseModificado3
[JuanCarlosRuizFernandez juancarlos@DESKTOP-7CE4K3P: /mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer1] 2021-04-14 Wednesday
$./private-clauseModificado3
thread 5 suma a[5] / thread 0 suma a[0] / thread 2 suma a[2] / thread 1 suma a[1] / thread 4 s
uma a[4] / thread 6 suma a[6] / thread 3 suma a[3] /
* thread 0 suma= 0
* thread 2 suma= 0
* thread 4 suma= 0
* thread 3 suma= 0
* thread 9 suma= 0
* thread 11 suma= 0
* thread 1 suma= 0
* thread 5 suma= 0
* thread 8 suma= 0
* thread 6 suma= 0
* thread 10 suma= 0
* thread 7 suma= 0
[JuanCarlosRuizFernandez juancarlos@DESKTOP-7CE4K3P: /mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer1] 2021-04-14 Wednesday
$
  
```

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:** Con `lastprivate` hacemos que predomine el valor modificado solo por la ultima hebra que ejecuta la ultima iteración en el vector (9).



```

Selecciónar C:\Windows\system32\wsl.exe
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ªIngInfo
2ºCuatri/AC/Prácticas/BP2/ejer4] 2021-04-21 Wednesday
$gcc -O2 -fopenmp firstlastprivate-modificado.c -o firstlastprivate-modificado
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ªIngInfo
2ºCuatri/AC/Prácticas/BP2/ejer4] 2021-04-21 Wednesday
$./firstlastprivate-modificado
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 4 suma a[6] suma=6
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 6 suma a[8] suma=8
thread 3 suma a[5] suma=5
thread 2 suma a[4] suma=4
thread 5 suma a[7] suma=7
thread 7 suma a[9] suma=9

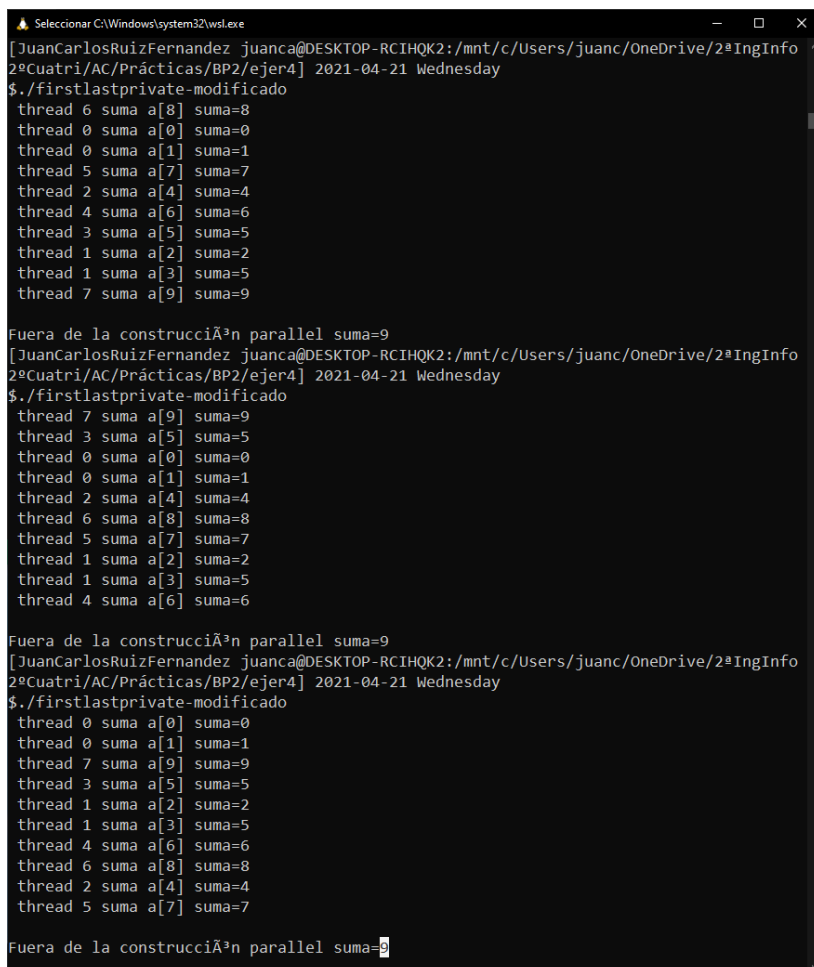
Fuera de la construcción parallel suma=9
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ªIngInfo
2ºCuatri/AC/Prácticas/BP2/ejer4] 2021-04-21 Wednesday

```

**CAPTURAS DE PANTALLA:**

**(b) RESPUESTA:** No, ya que `lastprivate` obliga a que la ultima iteración del bucle es la que mantiene el valor de la variable.

**CAPTURAS DE PANTALLA:**



```

Selecciónar C:\Windows\system32\wsl.exe
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ªIngInfo
2ºCuatri/AC/Prácticas/BP2/ejer4] 2021-04-21 Wednesday
$./firstlastprivate-modificado
thread 6 suma a[8] suma=8
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 5 suma a[7] suma=7
thread 2 suma a[4] suma=4
thread 4 suma a[6] suma=6
thread 3 suma a[5] suma=5
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 7 suma a[9] suma=9

Fuera de la construcción parallel suma=9
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ªIngInfo
2ºCuatri/AC/Prácticas/BP2/ejer4] 2021-04-21 Wednesday
$./firstlastprivate-modificado
thread 7 suma a[9] suma=9
thread 3 suma a[5] suma=5
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 2 suma a[4] suma=4
thread 6 suma a[8] suma=8
thread 5 suma a[7] suma=7
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 4 suma a[6] suma=6

Fuera de la construcción parallel suma=9
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ªIngInfo
2ºCuatri/AC/Prácticas/BP2/ejer4] 2021-04-21 Wednesday
$./firstlastprivate-modificado
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 7 suma a[9] suma=9
thread 3 suma a[5] suma=5
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 4 suma a[6] suma=6
thread 6 suma a[8] suma=8
thread 2 suma a[4] suma=4
thread 5 suma a[7] suma=7

Fuera de la construcción parallel suma=9

```

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) Observo que el vector no se inicializa al valor que le introduzco (con `single` debería iniciarse todo por igual) solo la correspondiente iteración a la hebra que ejecuta el `single` que es la única que ha cambiado el valor. b) Es debido, parafraseando lo último que he dicho justo antes, a que “a” se crea dentro del *parallel*, por lo que es privada de cada thread. Como en la sección *single* solo lo ejecuta un thread solo para este thread se cambiara el valor.

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

```

ej5 > C copyprivate-clauseModificado.c > main()
1  #include <stdio.h>
2  #include <omp.h>
3
4  int main()
5  {
6      int n = 9, i, b[n];
7
8      for (i = 0; i < n; i++)
9          b[i] = -1;
10
11     #pragma omp parallel
12     {
13         int a;
14
15         #pragma omp single
16         {
17             printf("\nIntroduce valor de inicializaciÃ³n a: ");
18             scanf("%d", &a);
19             printf("\nSingle ejecutada por el thread %d\n", omp_get_thread_num());
20         }
21
22         #pragma omp for
23         for (i = 0; i < n; i++)
24             b[i] = a;
25     }
26
27     printf("DepuÃ³s de la regiÃ³n parallel:\n");
28
29     for (i = 0; i < n; i++)
30         printf("b[%d] = %d\t", i, b[i]);
31
32     printf("\n");
33 }

```

#### CAPTURAS DE PANTALLA:

```

Seleccionar C:\Windows\system32\cmd.exe
[juan@juanc:~/OneDrive/2º Ingeniería Informática/Prácticas/2º Cuatrimestre/Prácticas/Práctica 2] 2021-04-21 Wednesday
$ gcc -O2 -fopenmp copyprivate-clauseModificado.c -o copyprivate-clauseModificado
[juan@juanc:~/OneDrive/2º Ingeniería Informática/Prácticas/2º Cuatrimestre/Prácticas/Práctica 2] 2021-04-21 Wednesday
$ ./copyprivate-clauseModificado

Introduce valor de inicializaciÃ³n a: 5

Single ejecutada por el thread 7
DepuÃ³s de la regiÃ³n parallel:
b[0] = 32767
b[1] = 32767
b[2] = 0
b[3] = 0
b[4] = 0
b[5] = 0
b[6] = 0
b[7] = 0
b[8] = 5

```

- 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:** Era de esperar que sean los mismo resultados que la diapositiva 27 del seminario 2 pero con 10 puntos de incremento

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

```

C reduction-clauseModificado.c X
ejer6 > C reduction-clauseModificado.c > main(int, char **)
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
9  int main(int argc, char **argv)
10 {
11     int i, n = 20, a[n], suma = 10;
12
13     if (argc < 2)
14     {
15         fprintf(stderr, "Falta iteraciones\n");
16         exit(-1);
17     }
18
19     n = atoi(argv[1]);
20
21     if (n > 20)
22     {
23         n = 20;
24         printf("n=%d", n);
25     }
26
27     for (i = 0; i < n; i++)
28         a[i] = i;
29
30     #pragma omp parallel for reduction(+: suma)
31     for (i = 0; i < n; i++)
32         suma += a[i];
33
34     printf("Tras 'parallel' suma=%d\n", suma);
35 }

```

#### CAPTURAS DE PANTALLA:

```

C:\Windows\system32\wsl.exe
[juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo
2ºCuatri/AC/Prácticas/BP2/ejer6] 2021-04-21 Wednesday
$gcc -O2 -fopenmp reduction-clauseModificado.c -o reduction-clauseModificado
[juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo
2ºCuatri/AC/Prácticas/BP2/ejer6] 2021-04-21 Wednesday
$./reduction-clauseModificado 10
Tras 'parallel' suma=55
[juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo
2ºCuatri/AC/Prácticas/BP2/ejer6] 2021-04-21 Wednesday
$./reduction-clauseModificado 20
Tras 'parallel' suma=200
[juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo
2ºCuatri/AC/Prácticas/BP2/ejer6] 2021-04-21 Wednesday
$./reduction-clauseModificado 6
Tras 'parallel' suma=25
[juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo
2ºCuatri/AC/Prácticas/BP2/ejer6] 2021-04-21 Wednesday
$

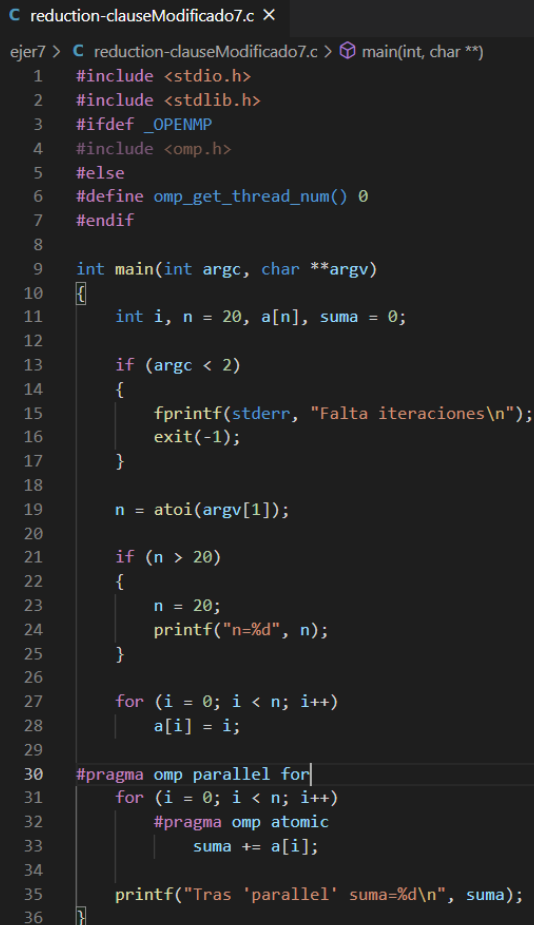
```



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:** El resultado usando `atomic` es correcto como el programa original con `reduction`.

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

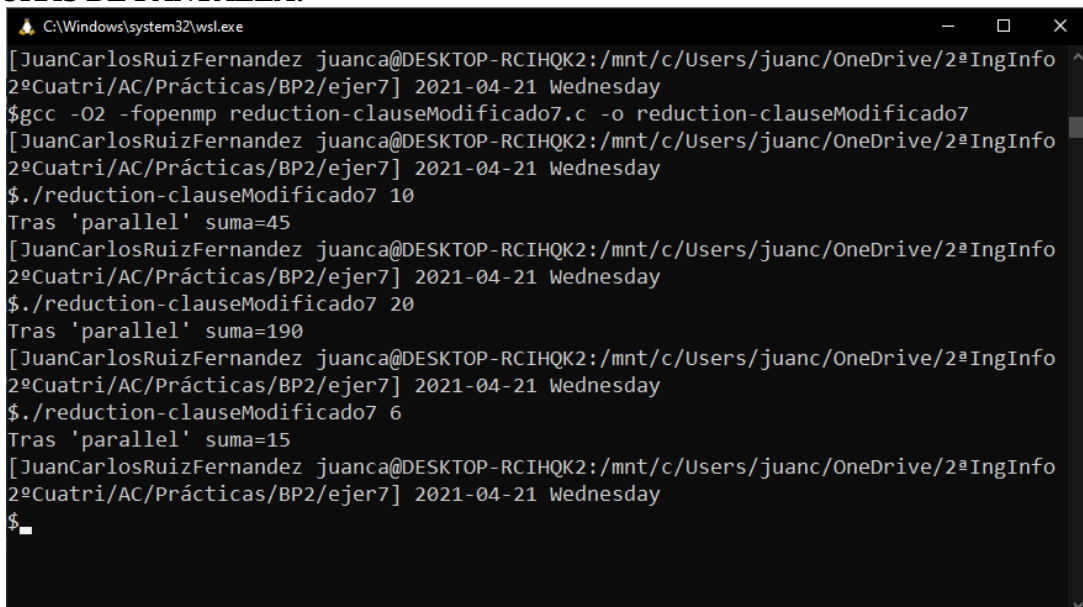


```

C reduction-clauseModificado7.c X
ejer7 > C reduction-clauseModificado7.c > main(int, char **)
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
9  int main(int argc, char **argv)
10 {
11     int i, n = 20, a[n], suma = 0;
12
13     if (argc < 2)
14     {
15         fprintf(stderr, "Falta iteraciones\n");
16         exit(-1);
17     }
18
19     n = atoi(argv[1]);
20
21     if (n > 20)
22     {
23         n = 20;
24         printf("n=%d", n);
25     }
26
27     for (i = 0; i < n; i++)
28         a[i] = i;
29
30     #pragma omp parallel for
31     for (i = 0; i < n; i++)
32     {
33         #pragma omp atomic
34         suma += a[i];
35     }
36     printf("Tras 'parallel' suma=%d\n", suma);

```

**CAPTURAS DE PANTALLA:**



```

C:\Windows\system32\wsl.exe
[juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer7] 2021-04-21 Wednesday
$gcc -O2 -fopenmp reduction-clauseModificado7.c -o reduction-clauseModificado7
[juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer7] 2021-04-21 Wednesday
$./reduction-clauseModificado7 10
Tras 'parallel' suma=45
[juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer7] 2021-04-21 Wednesday
$./reduction-clauseModificado7 20
Tras 'parallel' suma=190
[juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer7] 2021-04-21 Wednesday
$./reduction-clauseModificado7 6
Tras 'parallel' suma=15
[juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP2/ejer7] 2021-04-21 Wednesday
$

```





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):**

5.a una primera que paralelice el bucle que recorre las filas de la matriz y

5.b 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, **el número de hilos que usa** y, al menos, el primer y último componente del resultado (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

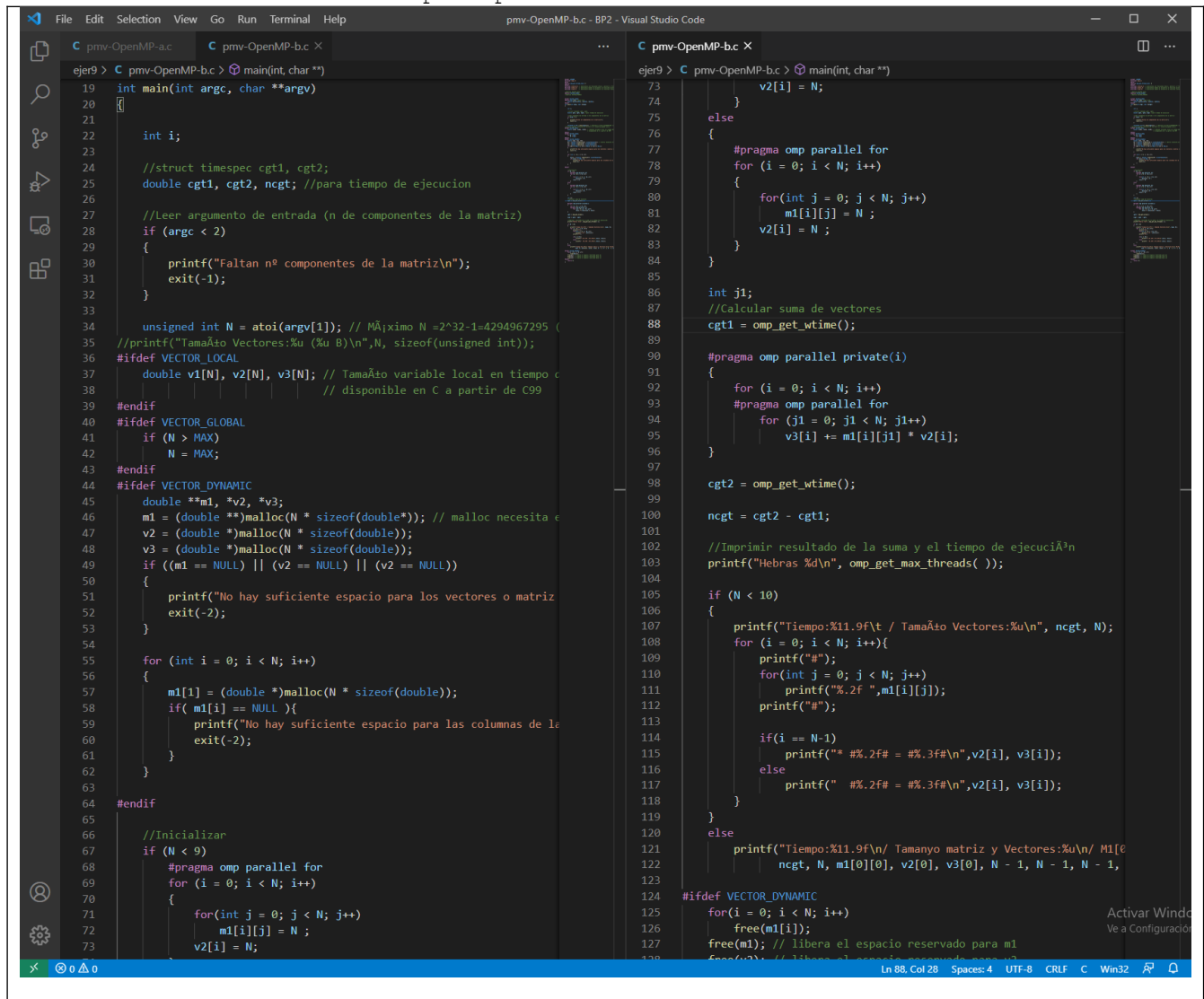
### CAPTURA CÓDIGO FUENTE : pmv-OpenMP-a.c

```

10 //define VECTOR_LOCAL
11 //define VECTOR_GLOBAL
12 //define VECTOR_DYNAMIC
13 #ifdef VECTOR_GLOBAL
14 #define MAX 5792 //sqrt 2^25
15 double m1[MAX][MAX], v2[MAX], v3[MAX];
16 #endif
17 int main(int argc, char **argv)
18 {
19     int i;
20
21     //struct timespec cgt1, cgt2;
22     double cgt1, cgt2, ncgt; //para tiempo de ejecucion
23
24     //Leer argumento de entrada (n de componentes de la matriz)
25     if (argc < 2)
26     {
27         printf("Faltan n° componentes de la matriz\n");
28         exit(-1);
29     }
30
31     unsigned int N = atoi(argv[1]); // Máximo N = 2^32-1=4294967295
32     //printf("Tamaño Vectores: %u (%u B)\n", N, sizeof(unsigned int));
33
34     #ifdef VECTOR_LOCAL
35     double v1[N], v2[N], v3[N]; // Tamaño variable local en tiempo de ejecución
36     // disponible en C a partir de C99
37     #endif
38     #ifdef VECTOR_GLOBAL
39     if (N > MAX)
40     {
41         N = MAX;
42     }
43     #endif
44     #ifdef VECTOR_DYNAMIC
45     double **m1, *v2, *v3;
46     m1 = (double **)malloc(N * sizeof(double*)); // malloc necesita
47     v2 = (double *)malloc(N * sizeof(double));
48     v3 = (double *)malloc(N * sizeof(double));
49     if ((m1 == NULL) || (v2 == NULL) || (v3 == NULL))
50     {
51         printf("No hay suficiente espacio para los vectores o matriz\n");
52         exit(-2);
53     }
54     for (int i = 0; i < N; i++)
55     {
56         m1[i] = (double *)malloc(N * sizeof(double));
57         if (m1[i] == NULL) {
58             printf("No hay suficiente espacio para las columnas de la matriz\n");
59             exit(-2);
60         }
61     }
62     #endif
63
64     //Inicializar
65
66     for (i = 0; i < N; i++)
67     {
68         for (int j = 0; j < N; j++)
69         {
70             m1[i][j] = N;
71             v2[i] = N;
72         }
73     }
74
75     #pragma omp parallel for
76     for (i = 0; i < N; i++)
77     {
78         for (int j = 0; j < N; j++)
79         {
80             m1[i][j] = N;
81             v2[i] = N;
82         }
83     }
84
85     //Calcular suma de vectores
86     cgt1 = omp_get_wtime();
87     #pragma omp parallel for
88     for (i = 0; i < N; i++)
89     {
90         for (int j = 0; j < N; j++)
91         {
92             v3[i] += m1[i][j] * v2[j];
93         }
94     }
95     cgt2 = omp_get_wtime();
96
97     ncgt = cgt2 - cgt1;
98
99     //Imprimir resultado de la suma y el tiempo de ejecución
100     printf("Hebras %d\n", omp_get_max_threads());
101
102     if (N < 10)
103     {
104         printf("Tiempo: %11.9f\t / Tamaño Vectores: %u\n", ncgt, N);
105         for (i = 0; i < N; i++) {
106             printf("#");
107             for (int j = 0; j < N; j++)
108             {
109                 printf("%.2f ", m1[i][j]);
110                 printf("#");
111             }
112             if (i == N-1)
113                 printf(" %.2f = %.3f\n", v2[i], v3[i]);
114             else
115                 printf(" %.2f = %.3f\n", v2[i], v3[i]);
116         }
117     }
118     else
119     {
120         printf("Tiempo: %11.9f\n / Tamaño matriz y Vectores: %u\n / M[0][0] = %d, v2[0] = %d, v3[0] = %d, N-1 = %d, N-1 = %d, N-1 = %d\n", ncgt, N, m1[0][0], v2[0], v3[0], N-1, N-1, N-1);
121     }
122
123     #ifdef VECTOR_DYNAMIC
124     for (i = 0; i < N; i++)
125     {
126         free(m1[i]);
127     }
128     free(m1); // libera el espacio reservado para m1
129     free(v2); // libera el espacio reservado para v2
130     free(v3); // libera el espacio reservado para v3
131 }

```

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



**RESPUESTA:**

Solo he tenido problemas de cara a setear el numero de hebras, no recordaba que funcion usar para conseguir el numero de hebras maximas disponibles por lo que lo busque en google y lo encuentre aquí:

<https://docs.microsoft.com/es-es/cpp/parallel/openmp/reference/openmp-functions?view=msvc-160#omp-get-max-threads>

Al ejecutar no tuve problema.

## CAPTURAS DE PANTALLA:

## Version A

```

b3estudiante23@atcgrid:~/BP2
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP2/ejer9] 2021-04-27 Tuesday
$ sbatch -pac --wrap "./pmv-OpenMP-a 8"
Submitted batch job 100335
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP2/ejer9] 2021-04-27 Tuesday
$ cat slurm-100335.out
Hebras 4
Tiempo:0.000026532 / Tamaño Vectores:8
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #512.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #512.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #512.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #512.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #512.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #512.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #512.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 #* #8.00# = #512.000#
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP2/ejer9] 2021-04-27 Tuesday
$ sbatch -pac --wrap "./pmv-OpenMP-a 11"
Submitted batch job 100336
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP2/ejer9] 2021-04-27 Tuesday
$ cat slurm-100336.out
Hebras 4
Tiempo:0.000024736
/ Tamaño matriz y Vectores:11
/ M1[0][0] += V2[0]=V3[0](11.0000000 += 11.0000000=1331.000000) /
/ M1[10][10] += V2[10]=V3[10](11.0000000 += 11.0000000 = 1331.000000) /
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP2/ejer9] 2021-04-27 Tuesday
$

```

## Version B

```

b3estudiante23@atcgrid:~/BP2
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP2/ejer9] 2021-04-27 Tuesday
$ sbatch -pac --wrap "./pmv-OpenMP-b 8"
Submitted batch job 100338
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP2/ejer9] 2021-04-27 Tuesday
$ cat slurm-100338.out
Hebras 4
Tiempo:0.000127804 / Tamaño Vectores:8
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #1536.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #1536.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #1536.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #1536.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #1536.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #1536.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #1536.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 #* #8.00# = #1536.000#
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP2/ejer9] 2021-04-27 Tuesday
$ sbatch -pac --wrap "./pmv-OpenMP-b 11"
Submitted batch job 100339
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP2/ejer9] 2021-04-27 Tuesday
$ cat slurm-100339.out
Hebras 4
Tiempo:0.000143580
/ Tamaño matriz y Vectores:11
/ M1[0][0] += V2[0]=V3[0](11.0000000 += 11.0000000=3993.000000) /
/ M1[10][10] += V2[10]=V3[10](11.0000000 += 11.0000000 = 3993.000000) /
[JuanCarlosRuizFernandez b3estudiante23@atcgrid:~/BP2/ejer9] 2021-04-27 Tuesday
$

```

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:

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

```

19 int main(int argc, char **argv)
20 {
21     int i;
22     //struct timespec cgt1, cgt2;
23     double cgt1, cgt2, ncgt; //para tiempo de ejecucion
24     //Leer argumento de entrada (n de componentes de la matriz)
25     if (argc < 2)
26     {
27         printf("Faltan n° componentes de la matriz\n");
28         exit(-1);
29     }
30     unsigned int N = atoi(argv[1]); // MÃximo N =2^32-1=4294967295
31     //printf("Tamaño Vectores: %u (%u B)\n", N, sizeof(unsigned int));
32     #ifdef VECTOR_LOCAL
33     double v1[N], v2[N], v3[N]; // Tamaño variable local en tiempo de ejecución
34     // disponible en C a partir de C99
35     #endif
36     #ifdef VECTOR_GLOBAL
37     if (N > MAX)
38         N = MAX;
39     #endif
40     #ifdef VECTOR_DYNAMIC
41     double *m1, *v2, *v3;
42     m1 = (double *)malloc(N * sizeof(double)); // malloc necesita
43     v2 = (double *)malloc(N * sizeof(double));
44     v3 = (double *)malloc(N * sizeof(double));
45     if ((m1 == NULL) || (v2 == NULL) || (v3 == NULL))
46     {
47         printf("No hay suficiente espacio para los vectores o matriz\n");
48         exit(-2);
49     }
50     for (int i = 0; i < N; i++)
51     {
52         m1[i] = (double *)malloc(N * sizeof(double));
53         if (m1[i] == NULL)
54         {
55             printf("No hay suficiente espacio para las columnas de la matriz\n");
56             exit(-2);
57         }
58     }
59     #endif
60     //Inicializar
61     if (N < 9)
62     {
63         #pragma omp parallel for
64         for (i = 0; i < N; i++)
65         {
66             for (int j = 0; j < N; j++)
67                 m1[i][j] = N;
68             v2[i] = N;
69         }
70     }
71     v2[i] = N;
72 }

```

```

73     v2[i] = N;
74 }
75 else
76 {
77     #pragma omp parallel for
78     for (i = 0; i < N; i++)
79     {
80         for (int j = 0; j < N; j++)
81             m1[i][j] = N;
82         v2[i] = N;
83     }
84 }
85 int j1;
86 double suma;
87 //Calcular suma de vectores
88 cgt1 = omp_get_wtime();
89 #pragma omp parallel private(i)
90 {
91     for (i = 0; i < N; i++)
92     {
93         #pragma omp parallel for reduction(+:v3[i])
94         for (j1 = 0; j1 < N; j1++)
95             v3[i] += m1[i][j1] * v2[j1];
96     }
97 }
98 cgt2 = omp_get_wtime();
99 ncgt = cgt2 - cgt1;
100 //Imprimir resultado de la suma y el tiempo de ejecución
101 printf("Hebras %d\n", omp_get_max_threads());
102 if (N < 10)
103 {
104     printf("Tiempo: %11.9f\t / Tamaño Vectores: %u\n", ncgt, N);
105     for (i = 0; i < N; i++)
106     {
107         printf("#");
108         for (int j = 0; j < N; j++)
109             printf("%.2f ", m1[i][j]);
110         printf("#");
111         if (i == N-1)
112             printf(" %.2f# = %.3f#\n", v2[i], v3[i]);
113         else
114             printf(" %.2f# = %.3f#\n", v2[i], v3[i]);
115     }
116 }
117 else
118 {
119     printf("Tiempo: %11.9f\n / Tamaño matriz y Vectores: %u\n / M1[0]
120         ncgt, N, m1[0][0], v2[0], v3[0], N - 1, N - 1, N - 1,
121         #ifdef VECTOR_DYNAMIC
122         for (i = 0; i < N; i++)
123             free(m1[i]);
124 }

```

**RESPUESTA:** Fallos no he tenido, solo he tenido que recordar con el seminario la nomenclatura del `reduction`.

Aunque me acabo de dar cuenta que no he realizado bien la multiplicación de matrices ni en este ni los anteriores. Pero no me da tiempo para cambiarlo.

**CAPTURAS DE PANTALLA:**

```

$gcc -O2 -fopenmp pmv-OpenMP-reduction.c -o pmv-OpenMP-reduction
[JuanCarlosRuizFernandez b3estudiante23@atcgriid:~/BP2/ejer10] 2021-04-27 Tuesday
$batch -pac --wrap "/pmv-OpenMP-reduction 11"
Submitted batch job 100368
[JuanCarlosRuizFernandez b3estudiante23@atcgriid:~/BP2/ejer10] 2021-04-27 Tuesday
$cat slurm-100368.out
Hebras 4
Tiempo:0.000151664
/ Tamaño matriz y Vectores:11
/ M1[0][0] * V2[0]=V3[0](11.000000 * 11.000000=5324.000000) /
/ M1[0][10] * V2[10]=V3[10](11.000000 * 11.000000 = 5324.000000) /
[JuanCarlosRuizFernandez b3estudiante23@atcgriid:~/BP2/ejer10] 2021-04-27 Tuesday
$batch -pac --wrap "/pmv-OpenMP-reduction 8"
Submitted batch job 100369
[JuanCarlosRuizFernandez b3estudiante23@atcgriid:~/BP2/ejer10] 2021-04-27 Tuesday
$cat slurm-100369.out
Hebras 4
Tiempo:0.000130758 / Tamaño Vectores:8
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #2048.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #2048.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #2048.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #2048.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #2048.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #2048.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #2048.000#
#8.00 8.00 8.00 8.00 8.00 8.00 8.00 8.00 # #8.00# = #2048.000#
[JuanCarlosRuizFernandez b3estudiante23@atcgriid:~/BP2/ejer10] 2021-04-27 Tuesday

```

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)
<b>Código Secuencial</b>		----		----		----		----		----		----
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												

## **COMENTARIOS SOBRE LOS RESULTADOS:**