

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): Alberto Llamas González

Grupo de prácticas y profesor de prácticas: D3, Juan Carlos Gómez López

Fecha de entrega: 2 mayo 2020

Fecha evaluación en clase: 3 mayo 2020

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

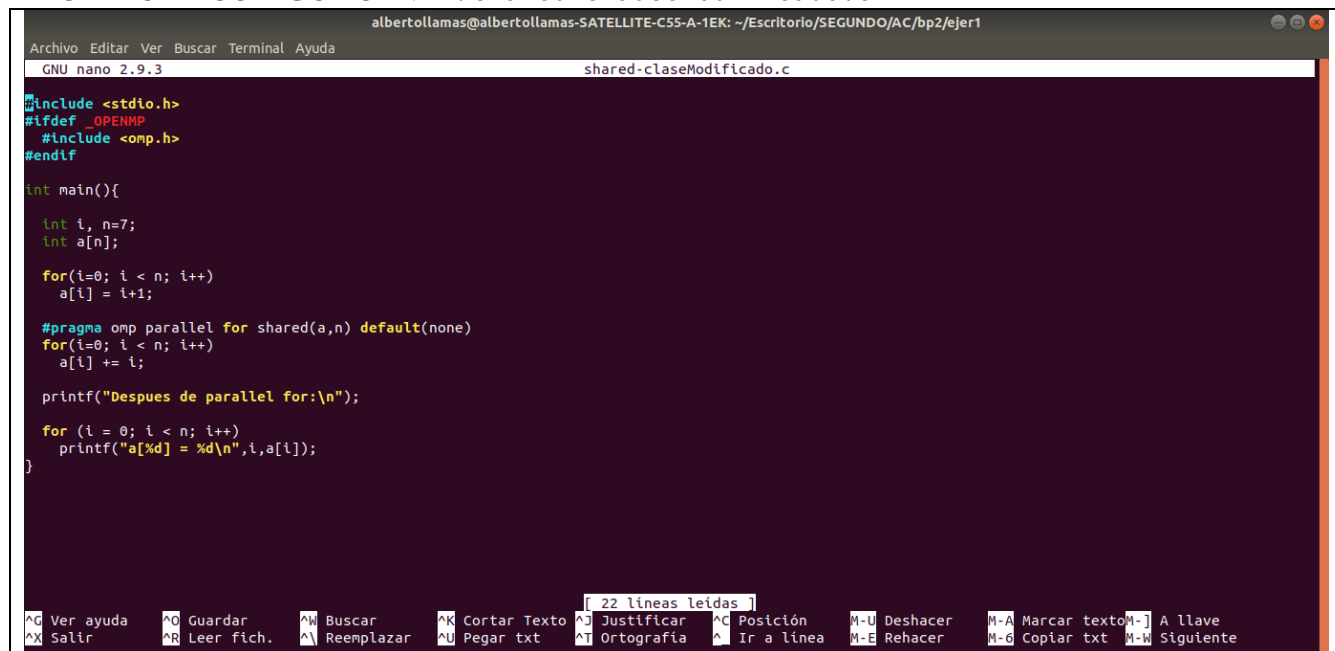
1. (a) Añadir la cláusula `default(none)` a la directiva `parallel` del ejemplo del seminario `shared-clause.c`? ¿Qué ocurre? ¿A qué se debe? (b) Resolver el problema generado sin eliminar `default(none)`. Incorporar el código con la modificación al cuaderno de prácticas. (Añadir capturas de pantalla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

RESPUESTA:

a) La compilación da error y no se logra crear el ejecutable. Esto se debe a que dentro de las regiones `parallel`, al haber usado `default(none)`, por defecto las variables declaradas previamente (y no especificadas dentro de la cláusula `shared`), como por ejemplo la variable “n”, no son compartidas y no se puede trabajar con ellas dentro de estas regiones paralelas (se desconoce su valor concreto y no es común a todas las hebras).

b) Se corrige incluyendo la variable “n” en la lista de la cláusula `shared`.

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`



```
albertollamas@albertollamas-SATELLITE-C55-A-1EK: ~/Escritorio/SEGUNDO/AC/bp2/ejer1
GNU nano 2.9.3 shared-claseModificado.c

#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#endif

int main(){
    int i, n=7;
    int a[n];

    for(i=0; i < n; i++)
        a[i] = i+1;

    #pragma omp parallel for shared(a,n) default(none)
    for(i=0; i < n; i++)
        a[i] += i;

    printf("Despues de parallel for:\n");

    for (i = 0; i < n; i++)
        printf("a[%d] = %d\n",i,a[i]);
}
```

CAPTURAS DE PANTALLA:

Código donde se produce el error

```

#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#endif

int main(){
    int i, n=7;
    int a[n];

    for(i=0; i < n; i++)
        a[i] = i+1;

    #pragma omp parallel for shared(a) default(none)
    for(i=0; i < n; i++)
        a[i] += i;

    printf("Despues de parallel for:\n");

    for (i = 0; i < n; i++)
        printf("a[%d] = %d\n",i,a[i]);
}

```

Error:

```

[AlbertollamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer1] 2021-04-26 lunes
$gcc -O2 -fopenmp -o shared-claseModificado shared-claseModificado.c -lrt
shared-claseModificado.c: In function 'main':
shared-claseModificado.c:14:11: error: 'n' not specified in enclosing 'parallel'
    #pragma omp parallel for shared(a) default(none)
    ^~~~~~
shared-claseModificado.c:14:11: error: enclosing 'parallel'
[AlbertollamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer1] 2021-04-26 lunes
$

```

Ejecución correcta:

```

[AlbertollamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer1] 2021-04-26 lunes
$gcc -O2 -fopenmp -o shared-claseModificado shared-claseModificado.c -lrt
[AlbertollamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer1] 2021-04-26 lunes
$./shared-claseModificado
Despues de parallel for:
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 7
a[4] = 9
a[5] = 11
a[6] = 13
[AlbertollamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer1] 2021-04-26 lunes
$

```

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:

Fuera del `parallel`, el código solo imprime el resultado de una suma, la suma del thread 0 y siempre igual a cero, aunque la inicialicemos a otro valor dentro de la región paralela. Eso es porque la clausula `private` solo afecta a la región paralela, por lo que fuera de ella, la variable suma ya no tiene las modificaciones realizadas

dentro de la región paralela y por tanto contiene el valor que tuviera al declararse e inicializarse fuera (que es cero por defecto).

CAPTURA CÓDIGO FUENTE: private-clauseModificado_a.c

```

C private-clauseModificado_a.c X
ejer2 > C private-clauseModificado_a.c
1  #include <stdio.h>
2
3  #ifdef _OPENMP
4      #include <omp.h>
5  #else
6      #define omp_get_thread_num() 0
7  #endif
8
9  int main()
10 {
11     int i, n;
12     int a[n], suma;
13
14     for (i=0; i<n; i++)
15         a[i] = i;
16
17     #pragma omp parallel private(suma)
18     {
19         suma=7;
20         #pragma omp for
21         for (i=0; i<n; i++)
22         {
23             suma = suma + a[i];
24             printf("thread %d suma a[%d] / ", omp_get_thread_num(), i);
25         }
26         printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
27     }
28
29     printf("\nValor de suma fuera de parallel = %d", suma);
30
31     printf("\n");
32 }

```

CAPTURAS DE PANTALLA:

```

[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer2] 2021-04-26 lunes
$gcc -O2 -fopenmp -o private-clauseModificado_a private-clauseModificado_a.c -lrt
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer2] 2021-04-26 lunes
$./private-clauseModificado_a
* thread 0 suma= 7
* thread 1 suma= 7
* thread 2 suma= 7
* thread 3 suma= 7
Valor de suma fuera de parallel = 0
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer2] 2021-04-26 lunes
$./private-clauseModificado_a
* thread 0 suma= 7
* thread 1 suma= 7
* thread 2 suma= 7
* thread 3 suma= 7
Valor de suma fuera de parallel = 0
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer2] 2021-04-26 lunes
$./private-clauseModificado_a
* thread 0 suma= 7
* thread 2 suma= 7
* thread 1 suma= 7
* thread 3 suma= 7
Valor de suma fuera de parallel = 0
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer2] 2021-04-26 lunes
$

```

(b) RESPUESTA:

Ahora ocurre que la suma que se imprime al final del programa (fuera del parallel) por el thread master (0) sí es el valor al que se había inicializado *suma* previamente ya que ahora lo hemos hecho fuera de la región paralela.

CAPTURA CÓDIGO FUENTE: private-clauseModificado_b.c

```

C private-clauseModificado_b.c X
ejer2 > C private-clauseModificado_b.c
1 #include <stdio.h>
2
3 #ifdef _OPENMP
4 #include <omp.h>
5 #else
6 #define omp_get_thread_num() 0
7 #endif
8
9 int main()
10 {
11     int i, n = 7;
12     int a[n], suma = 10;
13
14     for (i=0; i<n; i++)
15         a[i] = i;
16
17     #pragma omp parallel private(suma)
18     {
19         suma = 0;
20         #pragma omp for
21         for (i=0; i<n; i++)
22         {
23             suma = suma + a[i];
24             printf("thread %d suma a[%d] / ", omp_get_thread_num(), i);
25         }
26         printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
27     }
28
29     printf("\nValor de suma fuera de parallel = %d", suma);
30
31     printf("\n");
32 }

```

CAPTURAS DE PANTALLA:

```

albertollamas@albertollamas-SATELLITE-C55-A-1EK: ~/Escritorio/SEGUNDO/AC/bp2/ejer1
Archivo Editar Ver Buscar Terminal Ayuda
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer2] 2021-04-26 lunes
$ nano private-clauseModificado_b.c
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer2] 2021-04-26 lunes
$ gcc -O2 -fopenmp -o private-clauseModificado_b private-clauseModificado_b.c -lrt
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer2] 2021-04-26 lunes
$ ./private-clauseModificado_b
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 0 suma= 1
* thread 2 suma= 9
* thread 3 suma= 6
* thread 1 suma= 5
Valor de suma fuera de parallel = 10
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer2] 2021-04-26 lunes
$ ./private-clauseModificado_b
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 1 suma= 5
* thread 2 suma= 9
* thread 0 suma= 1
* thread 3 suma= 6
Valor de suma fuera de parallel = 10
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer2] 2021-04-26 lunes
$ ./private-clauseModificado_b
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 0 suma= 1
* thread 2 suma= 9
* thread 1 suma= 5
* thread 3 suma= 6
Valor de suma fuera de parallel = 10
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer2] 2021-04-26 lunes

```

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

RESPUESTA:

Lo que ocurre es que al no privatizar la variable `suma` se ha vuelto compartida con todas las hebras por lo que el `printf` que se haga será el del último valor asignado a `suma` debido a las condiciones de carrera.

CAPTURA CÓDIGO FUENTE: private-clauseModificado3.c

```

C private-clauseModificado3.c X
ej3 > C private-clauseModificado3.c > ...
1  #include <stdio.h>
2
3  #ifdef _OPENMP
4  #include <omp.h>
5  #else
6  #define omp_get_thread_num() 0
7  #endif
8
9  int main()
10 {
11     int i, n = 7;
12     int a[n], suma;
13
14     for (i=0; i<n; i++)
15         a[i] = i;
16
17     #pragma omp parallel
18     {
19         suma=0;
20         #pragma omp for
21         for (i=0; i<n; i++)
22         {
23             suma = suma + a[i];
24             printf("thread %d suma a[%d] / ", omp_get_thread_num(), i);
25         }
26         printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
27     }
28
29     printf("\n");
30 }
31

```

CAPTURAS DE PANTALLA:

```

albertollamas@albertollamas-SATELLITE-C55-A-1EK: ~/Escritorio/SEGUNDO/AC/bp2/ejer1
Archivo Editar Ver Buscar Terminal Ayuda
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer3] 2021-04-26 lunes
$gcc -O2 -fopenmp -o private-clauseModificado3 private-clauseModificado3.c -lrt
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer3] 2021-04-26 lunes
$./private-clauseModificado3
thread 0 suma a[0] / thread 0 suma a[1] / thread 3 suma a[6] / thread 1 suma a[2] / thread 1 suma a[3] / thread 2 suma a[4] / thread 2 suma a[5] /
* thread 1 suma= 15
* thread 0 suma= 15
* thread 3 suma= 15
* thread 2 suma= 15
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer3] 2021-04-26 lunes
$./private-clauseModificado3
thread 0 suma a[0] / thread 0 suma a[1] / thread 3 suma a[6] / thread 1 suma a[2] / thread 1 suma a[3] / thread 2 suma a[4] / thread 2 suma a[5] /
* thread 1 suma= 15
* thread 0 suma= 15
* thread 3 suma= 15
* thread 2 suma= 15
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer3] 2021-04-26 lunes
$./private-clauseModificado3
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 0 suma= 15
* thread 2 suma= 15
* thread 1 suma= 15
* thread 3 suma= 15
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer3] 2021-04-26 lunes
$

```

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:

El código imprimirá primero las sumas realizadas por las hebras: cada hebra tiene un contenido propio en la variable suma y todas comienzan inicializadas a cero (debido al uso de firstprivate) y van sumando el índice de la iteración que realizan a “su” variable suma. Por último el código imprime como resultado el contenido de suma, que al haber usado “lastprivate (suma)”, es el contenido en la variable suma de la hebra que la haya modificado tras la última iteración, es decir, el contenido que tenga la variable suma en la hebra que haya realizado la última iteración del bucle.

CAPTURAS DE PANTALLA:

```

C firstlastprivate_a.c x
ejer4 > C firstlastprivate_a.c > main()
1  #include <stdio.h>
2  #ifdef _OPENMP
3  #include <omp.h>
4  #else
5  #define omp_get_thread_num() 0
6  #endif
7
8  int main()
9  {
10
11     int i, n = 10;
12     int a[n], suma = 0;
13
14     for (i = 0; i < n; i++)
15         a[i] = i;
16
17     #pragma omp parallel for firstprivate(suma) lastprivate(suma)
18     for (i = 0; i < n; i++)
19     {
20         suma = suma + a[i];
21         printf(" thread %d suma a[%d] suma=%d \n", omp_get_thread_num(), i, suma);
22     }
23
24     printf("\nFuera de la construcción parallel suma=%d\n", suma);
25 }

```

```

[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer4] 2021-05-01 sábado
$gcc -O2 -fopenmp firstlastprivate_a.c -o firstlastprivate_a
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer4] 2021-05-01 sábado
$./firstlastprivate_a
thread 2 suma a[6] suma=6
thread 2 suma a[7] suma=13
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 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
Fuera de la construcción parallel suma=17
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer4] 2021-05-01 sábado

```

(b) RESPUESTA:

El tamaño del vector es 7 y en el for el programa solo tiene que realizar 7 iteraciones donde cada una de ellas la realiza una hebra. Como mi pc hace uso de 4 hebras (0-3) y las iteraciones son 7, hay necesidad de que una hebra tenga que hacer dos sumas por lo que la última hebra será la que asigne el valor a suma debido al uso de last private en suma. Es decir, 6.

CAPTURAS DE PANTALLA:

```

C firstlastprivate_a.c  C firstlastprivate_b.c x
ejer4 > C firstlastprivate_b.c > main()
1  #include <stdio.h>
2  #ifdef _OPENMP
3      #include <omp.h>
4  #else
5      #define omp_get_thread_num() 0
6  #endif
7
8  int main() {
9
10     int i, n = 7;
11     int a[n], suma=0;
12
13     for (i=0; i<n; i++)
14         a[i] = i;
15
16     #pragma omp parallel for firstprivate(suma) lastprivate(suma)
17     for (i=0; i<n; i++)
18     {
19         suma = suma + a[i];
20         printf(" thread %d suma a[%d] suma=%d \n", omp_get_thread_num(),i,suma);
21     }
22
23     printf("\nFuera de la construcción parallel suma=%d\n", suma);
24 }

[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer4] 2021-05-01 sábado
$gcc -O2 -fopenmp firstlastprivate_b.c -o firstlastprivate_b
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer4] 2021-05-01 sábado
$./firstlastprivate_b
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9
thread 3 suma a[6] suma=6

Fuera de la construcción parallel suma=6
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer4] 2021-05-01 sábado
$./firstlastprivate_b
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

Fuera de la construcción parallel suma=6
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer4] 2021-05-01 sábado

```

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:

Que sólo una de las hebras tendrá el valor de a actualizado, ya que hemos declarado a en el `parallel` y se trata de una variable local a cada una de las hebras. De este modo, el vector se inicializará algunas componentes a 0, y otras al valor de a insertado, en función de la distribución que haga OpenMP en el `for`.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```

C copyprivate-clauseModificado.c X
ejer5 > 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         #pragma omp single
15         {
16             printf("\nIntroduce valor de inicialización a: ");
17             scanf("%d", &a);
18             printf("\nSingle ejecutada por el thread %d\n",
19                 omp_get_thread_num());
20         }
21         #pragma omp for
22         for (i = 0; i < n; i++)
23             b[i] = a;
24     }
25     printf("\nDespués de la región parallel:\n");
26     for (i = 0; i < n; i++)
27         printf("b[%d] = %d\t", i, b[i]);
28     printf("\n");
29 }

```

CAPTURAS DE PANTALLA:

```

[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer5] 2021-05-01 sábado
$gcc -O2 -fopenmp copyprivate-clauseModificado.c -o copyprivate-clauseModificado
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer5] 2021-05-01 sábado
$./copyprivate-clauseModificado

Introduce valor de inicialización a: 4

Single ejecutada por el thread 0
Después de la región parallel:
b[0] = 4      b[1] = 4      b[2] = 4      b[3] = 0      b[4] = 0      b[5] = 0      b[6] = 0      b[7] = 0      b[8] = 0
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer5] 2021-05-01 sábado
$./copyprivate-clauseModificado

Introduce valor de inicialización a: 2

Single ejecutada por el thread 1
Después de la región parallel:
b[0] = 0      b[1] = 0      b[2] = 0      b[3] = 2      b[4] = 2      b[5] = 0      b[6] = 0      b[7] = 0      b[8] = 0
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer5] 2021-05-01 sábado
$./copyprivate-clauseModificado

Introduce valor de inicialización a: 3

Single ejecutada por el thread 0
Después de la región parallel:
b[0] = 3      b[1] = 3      b[2] = 3      b[3] = 0      b[4] = 0      b[5] = 0      b[6] = 0      b[7] = 0      b[8] = 0
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer5] 2021-05-01 sábado

```

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: Queda el valor de suma + 10, porque master se queda con el valor 10. Inicializa las variables locales, no el valor de suma fuera.

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     n = atoi(argv[1]);
19     if (n > 20)
20     {
21         n = 20;
22         printf("n=%d", n);
23     }
24     for (i = 0; i < n; i++)
25         a[i] = i;
26     #pragma omp parallel for reduction(+: suma)
27     for (i = 0; i < n; i++)
28         suma += a[i];
29     printf("Tras 'parallel' suma=%d\n", suma);
30 }

```

CAPTURAS DE PANTALLA:

Se muestra también una ejecución de *reduction-clause.c*.

```

albertollamas@albertollamas-SATELLITE-C55-A-1EK: ~/Escritorio/SEGUNDO/AC/bp2/ejer4
Archivo Editar Ver Buscar Terminal Ayuda
[AlbertollamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer6] 2021-05-01 sábado
$ gcc -O2 -fopenmp reduction-clauseModificado.c -o reduction-clauseModificado
[AlbertollamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer6] 2021-05-01 sábado
$ ./reduction-clauseModificado 10
Tras 'parallel' suma=55
[AlbertollamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer6] 2021-05-01 sábado
$ gcc -O2 -fopenmp reductionclause.c -o reductionclause
[AlbertollamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer6] 2021-05-01 sábado
$ ./reductionclause 10
Tras 'parallel' suma=45
[AlbertollamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer6] 2021-05-01 sábado
$

```

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).

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     n = atoi(argv[1]);
19     if (n > 20)
20     {
21         n = 20;
22         printf("n=%d", n);
23     }
24     for (i = 0; i < n; i++)
25         a[i] = i;
26
27     #pragma omp parallel
28     {
29         int suma_local = 0;
30         #pragma omp parallel for
31         for (i = 0; i < n; i++)
32             suma_local += a[i];
33
34         #pragma omp atomic
35         suma += suma_local;
36     }
37     printf("Tras 'parallel' suma=%d\n", suma);
38 }

```

CAPTURAS DE PANTALLA:

```
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer7] 2021-05-01 sábado
$gcc -O2 -fopenmp reduction-clauseModificado7.c -o reduction-clauseModificado7
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer7] 2021-05-01 sábado
$./reduction-clauseModificado7 10
Tras 'parallel' suma=180
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer7] 2021-05-01 sábado
$./reduction-clauseModificado7 10
Tras 'parallel' suma=180
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer7] 2021-05-01 sábado
$./reduction-clauseModificado7 20
Tras 'parallel' suma=760
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer7] 2021-05-01 sábado
$./reduction-clauseModificado7 5
Tras 'parallel' suma=40
[AlbertoLlamasGonzalez albertollamas@albertollamas-SATELLITE-C55-A-1EK:~/Escritorio/SEGUNDO/AC/bp2/ejer7] 2021-05-01 sábado
$
```

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):

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

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).

*Nota: He decidido usar el código para dinámicas. Por otra parte, podemos ver que el código es correcto teniendo en cuenta que hemos inicializado la matriz a $M=2*Id$. Por tanto, $v2 = 2Id*v1 = 2v1$.*

CAPTURA CÓDIGO FUENTE: pmv-secuencial.c

```
C pmv-secuencial.c x
ejer8 > C pmv-secuencial.c > main(int, char **)
1  /* pmv-secuencial.c
2  Calcula:
3  - el producto de matriz cuadrada(n) por un vector(n): v2 = M x v1
4  - el tiempo que tarda en realizar dicho producto
5
6  Para compilar usar: gcc -O2 -fopenmp pmv-secuencial.c -o pmv-secuencial
7  Para ejecutar use: pmv-secuencial "num filas/columnas"
8
9  @author Alberto Llamas González
10
11  */
12
13  #include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
14  #include <stdio.h> // biblioteca donde se encuentra la función printf()
15  #include <time.h> // biblioteca donde se encuentra la función clock_gettime()
16
17  // #define VECTOR_GLOBAL // descomentar para que los vectores sean variables Dinamicas
18  #define VECTOR_DYNAMIC // descomentar para que los vectores sean variables globales
19
20  #ifdef VECTOR_GLOBAL
21  #define MAX 33554432 //2^25
22
23  double v1[MAX], v2[MAX], v3[MAX];
24  #endif
25
26  int main(int argc, char **argv)
27  {
28
29      int i;
30
31      struct timespec cgt1, cgt2;
32      double ncgt; //para tiempo de ejecución
33
```

```

C pmv-secuencial.c x
ejer8 > C pmv-secuencial.c > main(int, char**)
33
34 //Leer argumento de entrada (nº de componentes del vector)
35 if (argc < 2)
36 {
37     printf("ERROR: Falta el tamaño N de la matriz NxN y del vector v\n");
38     exit(-1);
39 }
40
41 unsigned int N = atoi(argv[1]); // Tamaño del vector
42
43 #ifdef VECTOR_DYNAMIC
44 double *v1, *v2, **M;
45 v1 = (double *)malloc(N * sizeof(double)); // malloc necesita el tamaño en bytes
46 v2 = (double *)malloc(N * sizeof(double));
47 M = (double **)malloc(N * sizeof(double*));
48
49 for (int i = 0; i < N; i++)
50     M[i] = (double*) malloc(N*sizeof(double));
51
52 if ((v1 == NULL) || (v2 == NULL) || (M == NULL))
53 {
54     printf("No hay suficiente espacio para los vectores \n");
55     exit(-2);
56 }
57 #endif
58
59 //Inicializar vectores y matriz
60 for (int i = 0; i < N; i++){
61     v1[i] = 1;
62     v2[i] = 0;
63 }
64
65 //Haremos que la matriz M sea 2*Id donde Id = matriz identidad, para poder
66 //comprobar el resultado simplemente viendo que v2 = 2*v1;
67 for (int j = 0; j < N; j++){
68     if (j == 0)
69         M[i][j] = 2;
70     else
71         M[i][j] = 0;
72 }
73
74 clock_gettime(CLOCK_REALTIME, &cgt1);
75 //Calcular el producto v2 = M*v1;
76
77 for (i = 0; i < N; i++)
78     for (int k = 0; k < N; k++)
79         v2[i] += M[i][k]*v1[k];
80
81 clock_gettime(CLOCK_REALTIME, &cgt2);
82 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
83         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
84
85 //Imprimimos v2 por pantalla y el tiempo de ejecución
86 printf("Resultado v2 = ");
87 for (int i = 0; i < N; i++)
88     printf("%f ", v2[i]);
89 printf("\n\nTamaño: %d -> Tiempo de ejecucion: %f\n\n", N, ncgt);
90
91 #ifdef VECTOR_DYNAMIC
92 free(v1); // libera el espacio reservado para v1
93 free(v2); // libera el espacio reservado para v2
94 for (int i = 0; i < N; i++)
95     free(M[i]);
96 free(M);
97 #endif
98 return 0;
99 }

```

CAPTURAS DE PANTALLA:

```
[AlbertoLlamasGonzalez d3estudiante26@atcgrid:~/bp2/ejer8] 2021-05-01 sábado
$gcc -O2 -fopenmp pmv-secuencial.c -o pmv-secuencial -lrt
[AlbertoLlamasGonzalez d3estudiante26@atcgrid:~/bp2/ejer8] 2021-05-01 sábado
$./pmv-secuencial 16
Resultado v2 = 0.000000 2.000000 4.000000 6.000000 8.000000 10.000000 12.000000 14.000000 16.000000 18.000000 20.000000 22.000000 24.000000 26.000000
28.000000 30.000000

Tamano: 16 -> Tiempo de ejecucion: 0.000001
[AlbertoLlamasGonzalez d3estudiante26@atcgrid:~/bp2/ejer8] 2021-05-01 sábado
$
```

9. Implementar en paralelo el producto matriz por vector con OpenMP a partir del código escrito en el ejercicio anterior usando la directiva for. Debe implementar dos versiones del código (consulte la lección 5/Tema 2):

- una primera que paralelice el bucle que recorre las filas de la matriz y
- una segunda que paralelice el bucle que recorre las columnas.

Use las directivas que estime oportunas y las cláusulas que sean necesarias **excepto la cláusula reduction**. Se debe paralelizar también la inicialización de las matrices. Respecto a este ejercicio:

- Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
- Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).

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

RESPUESTA:

Como ayuda externa se han utilizado los documentos de los seminarios de la BP1 y BP2-

Se muestra la parte modificada del código original de *pmv-secuencial.c* . También se incluye al principio de cada modificación el siguiente código:

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

CAPTURA CÓDIGO FUENTE : pmv-OpenMP-a.c

```

C pmv-OpenMP-a.c x
ejer9 > C pmv-OpenMP-a.c > main(int, char **)
72 }
73 #ifdef _OPENMP
74 double start = omp_get_wtime();
75 #else
76 clock_gettime(CLOCK_REALTIME, &cgt1);
77 #endif
78
79 //Calcular el producto v2 = M*v1;
80 #pragma omp parallel for
81 for (i = 0; i < N; i++)
82     for (int k = 0; k < N; k++)
83         v2[i] += M[i][k] * v1[k];
84
85 #ifdef _OPENMP
86 ncgt = omp_get_wtime() - start;
87 #else
88 clock_gettime(CLOCK_REALTIME, &cgt2);
89 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
90         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
91 #endif
92
93 // imprimimos v2 por pantalla
94 if (N < 10)
95 {
96     printf("Resultado: v2 = ");
97     for (int i = 0; i < N; i++)
98         printf("%f ", v2[i]);
99 }
100
101 // imprimimos finales
102 printf("v2[0]: %f, v2[%u]: %f\n", v2[0], N - 1, v2[N - 1]);
103 //imprimimos tiempo de ejecucion
104 printf("\n\nTamano: %d -> Tiempo de ejecucion: %f\n\n", N, ncgt);
105

```

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

```

C pmv-OpenMP-b.c x
ejer9 > C pmv-OpenMP-b.c > main(int, char **)
68 #ifdef _OPENMP
69 double start = omp_get_wtime();
70 #else
71 clock_gettime(CLOCK_REALTIME, &cgt1);
72 #endif
73 double suma_parcial;
74 // calculamos el producto v2 = M*v1
75 #pragma omp parallel private(suma_parcial)
76 for (int i = 0; i < N; i++)
77 {
78     suma_parcial = 0;
79 #pragma omp for
80     for (int k = 0; k < N; k++)
81         suma_parcial += M[i][k] * v1[k];
82 #pragma omp critical
83     v2[i] += suma_parcial;
84 }
85 #ifdef _OPENMP
86 ncgt = omp_get_wtime() - start;
87 #else
88 clock_gettime(CLOCK_REALTIME, &cgt2);
89 ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
90         (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
91 #endif
92
93 // imprimimos v2 por pantalla
94 if (N < 10)
95 {
96     printf("Resultado: v2 = ");
97     for (int i = 0; i < N; i++)
98         printf("%f ", v2[i]);
99 }
100
101 // imprimimos finales
102 printf("v2[0]: %f, v2[%u]: %f\n", v2[0], N - 1, v2[N - 1]);

```

CAPTURAS DE PANTALLA:**Ejecución a)**

```

[AlbertoLlamasGonzalez d3estudiante26@atcgrid:~/bp2/ejer9] 2021-05-01 sábado
$gcc -O2 -fopenmp pmv-OpenMP-a.c -o pmv-OpenMP-a -lrt
[AlbertoLlamasGonzalez d3estudiante26@atcgrid:~/bp2/ejer9] 2021-05-01 sábado
$./pmv-OpenMP-a 7
Resultado: v2 = 0.000000 2.000000 4.000000 6.000000 8.000000 10.000000 12.000000 v2[0]: 0.000000, v2[6]: 12.000000

Tamano: 7 -> Tiempo de ejecucion: 0.008744

[AlbertoLlamasGonzalez d3estudiante26@atcgrid:~/bp2/ejer9] 2021-05-01 sábado
$

```

Ejecución b)

```
[AlbertoLlamasGonzalez d3estudiante26@atcgrid:~/bp2/ejer9] 2021-05-01 sábado
$gcc -O2 -fopenmp pmv-OpenMP-b.c -o pmv-OpenMP-b -lrt
[AlbertoLlamasGonzalez d3estudiante26@atcgrid:~/bp2/ejer9] 2021-05-01 sábado
$./pmv-OpenMP-b 7
Resultado: v2 = 0.000000 2.000000 4.000000 6.000000 8.000000 10.000000 12.000000 v2[0]: 0.000000, v2[6]: 12.000000

Tamaño: 7 -> Tiempo de ejecución: 0.000275

[AlbertoLlamasGonzalez d3estudiante26@atcgrid:~/bp2/ejer9] 2021-05-01 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).

RESPUESTA:**Errores:**

El principal error que me ha tenido un tiempo pensando cómo solucionarlo ya que no encontraba la línea de código, ha sido el que se muestra en la imagen siguiente:

```
66     }
67
68     #ifdef _OPENMP
69     |     double start = omp_get_wtime();
70     #else
71     |     clock_gettime(CLOCK_REALTIME, &cgt1);
72     #endif
73
74     double suma_parcial;
75     // calculamos el producto v2 = M*v1
76     #pragma omp parallel
77     for (int i = 0; i < N; i++)
78     {
79     |     suma_parcial = 0;
80     |     #pragma omp for reduction(+:suma_parcial)
81     |     for (int k = 0; k < N; k++)
82     |     |     suma_parcial += M[i][k] * v1[k];
83     |
84     |     #pragma omp single
85     |     {
86     |     |     v2[i] = suma_parcial;
87     |     |     suma_parcial = 0;
88     |     }
89     }
90     #ifdef _OPENMP
91     |     ncgt = omp_get_wtime() - start;
92     #else
93     |     clock_gettime(CLOCK_REALTIME, &cgt2);
94     |     ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
95     |     (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
96     #endif
97
98     // imprimimos v2 por pantalla
99     if (N < 10)
```

El problema era que el resultado de `v2` era siempre 0 salvo en la última posición por lo que para solucionarlo, simplemente había que eliminar dicha línea de código.

Se muestra la parte modificada del código anterior:

CAPTURA CÓDIGO FUENTE: pmv-OpenmMP-reduction.c

```

1 //
2 #ifdef _OPENMP
3     double start = omp_get_wtime();
4 #else
5     clock_gettime(CLOCK_REALTIME, &cgt1);
6 #endif
7
8 double suma_parcial;
9 // calculamos el producto v2 = M*v1
10 #pragma omp parallel
11 for (int i = 0; i < N; i++)
12 {
13     #pragma omp for reduction(+:suma_parcial)
14     for (int k = 0; k < N; k++)
15         suma_parcial += M[i][k] * v1[k];
16 }
17
18 #pragma omp single
19 {
20     v2[i] = suma_parcial;
21     suma_parcial = 0;
22 }
23 }
24 #ifdef _OPENMP
25     ncgt = omp_get_wtime() - start;
26 #else
27     clock_gettime(CLOCK_REALTIME, &cgt2);
28     ncgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
29           (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
30 #endif
31
32 // imprimimos v2 por pantalla
33 if (N < 10)
34 {
35     printf("Resultado: v2 = ");
36     for (int i = 0; i < N; i++)
37         printf("%f ", v2[i]);
38 }

```

CAPTURAS DE PANTALLA:

```

[AlbertoLlamasGonzalez d3estudiante26@atcgrid:~/bp2/ejer10] 2021-05-01 sábado
$gcc -O2 -fopenmp pmv-OpenmMP-reduction.c -o pmv-OpenmMP-reduction -lrt
[AlbertoLlamasGonzalez d3estudiante26@atcgrid:~/bp2/ejer10] 2021-05-01 sábado
$./pmv-OpenmMP-reduction 7
Resultado: v2 = 0.000000 2.000000 4.000000 6.000000 8.000000 10.000000 12.000000 v2[0]: 0.000000, v2[6]: 12.000000

Tamaño: 7 -> Tiempo de ejecución: 0.000293
[AlbertoLlamasGonzalez d3estudiante26@atcgrid:~/bp2/ejer10] 2021-05-01 sábado
$

```

11.(NO REALIZADO) 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: