

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): Guillermo Sandoval Schmidt

Grupo de prácticas y profesor de prácticas: C3

Fecha de entrega:

Fecha evaluación en clase:

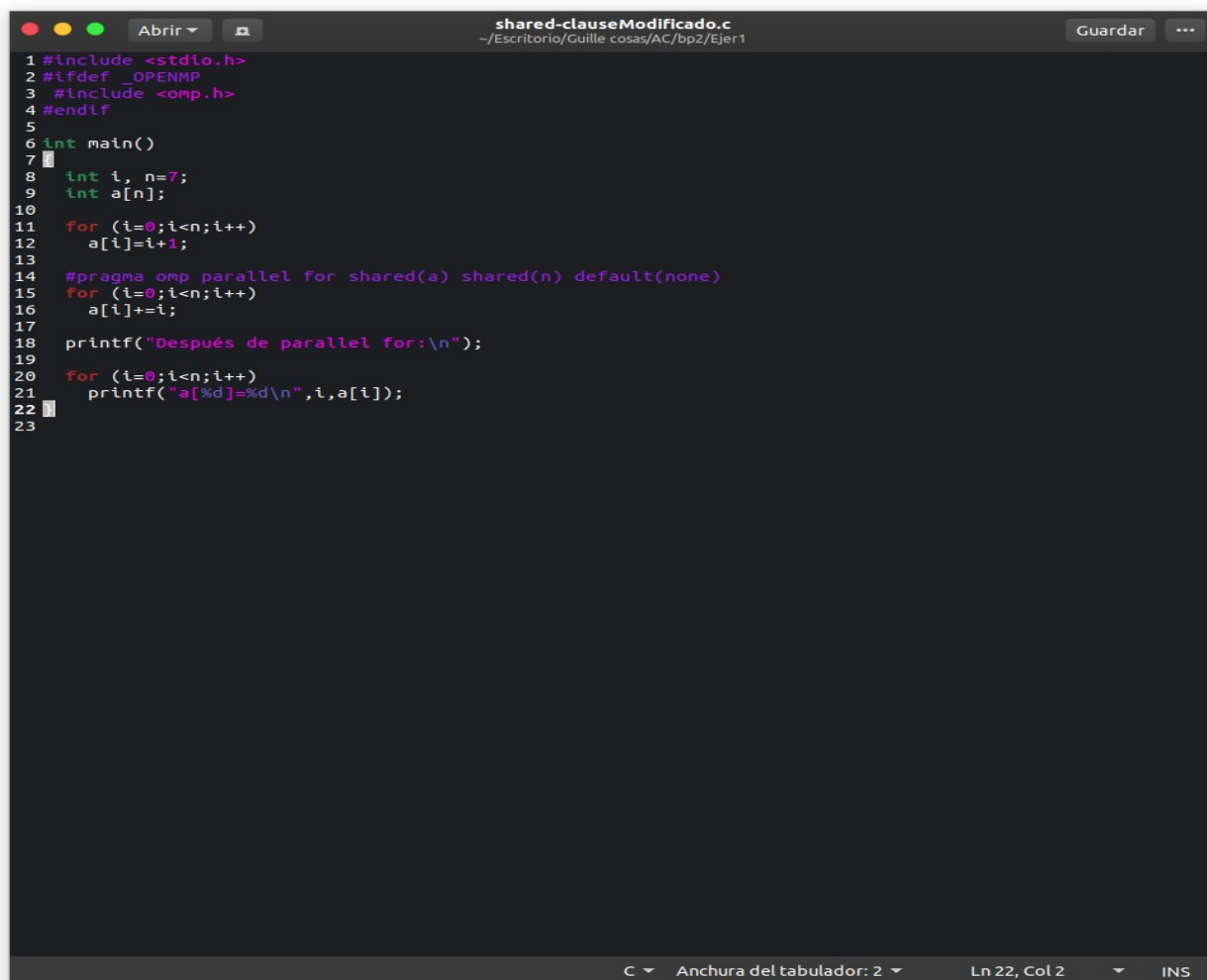
Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

Ejercicios basados en los ejemplos del seminario práctico

1. ¿Qué ocurre si en el ejemplo del seminario `shared-clause.c` se añade a la directiva `parallel` la cláusula `default(none)`? **(b)** Resuelva el problema generado sin eliminar `default(none)`. Añada el código con la modificación al cuaderno de prácticas. (Añada capturas de pantalla que muestren lo que ocurre)

RESPUESTA: Si solo se añade esa cláusula, no compila el programa.

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`



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

CAPTURAS DE PANTALLA:

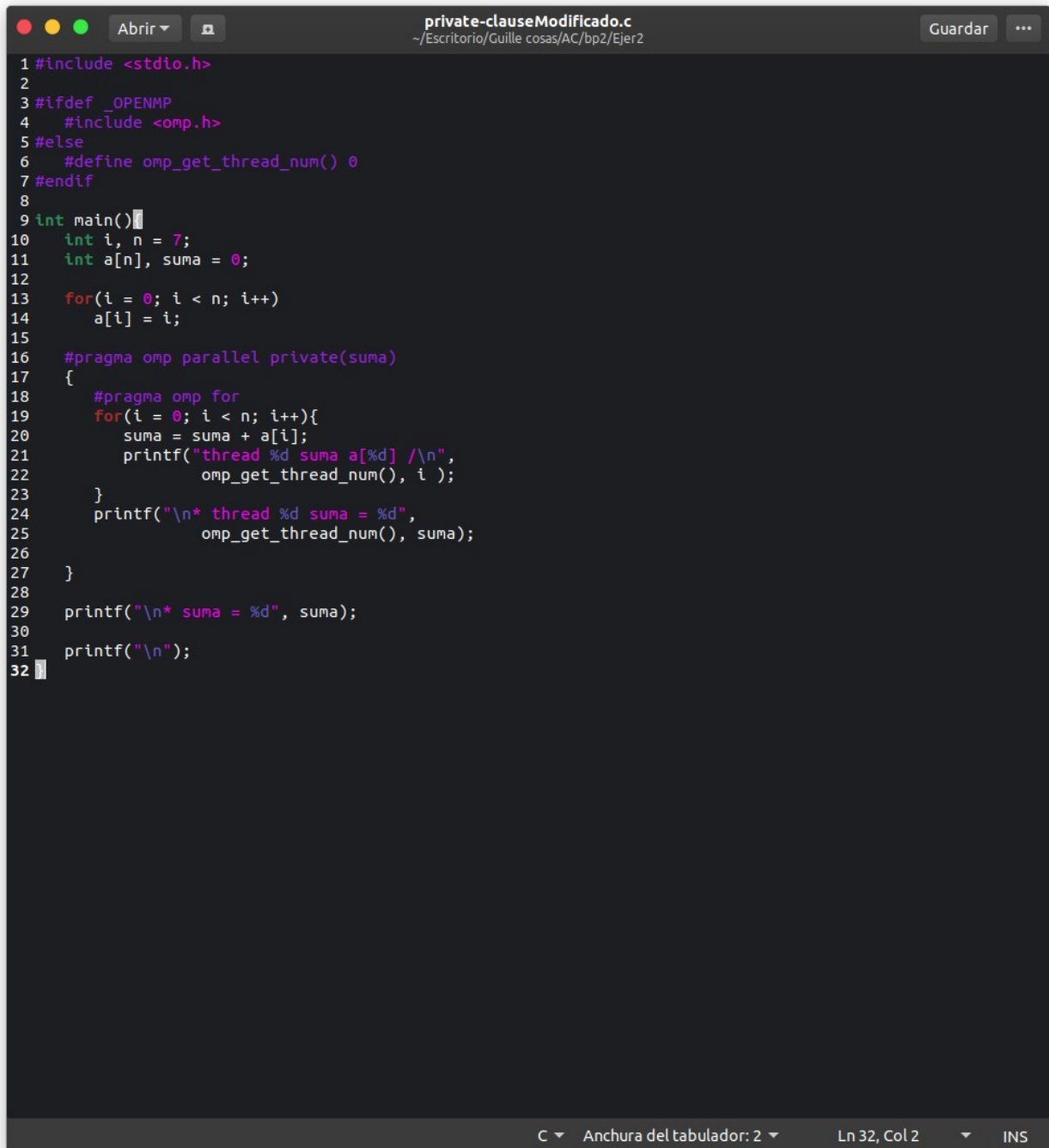
```
Archivo Editar Ver Buscar Terminal Ayuda
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer1$ gcc -O2 -fopenmp sha
red-clause.c -o ejecutable
shared-clause.c: In function 'main':
shared-clause.c:14:11: error: 'n' not specified in enclosing 'parallel'
    #pragma omp parallel for shared(a) default(none)
                ^~~
shared-clause.c:14:11: error: enclosing 'parallel'
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer1$
```

```
Archivo Editar Ver Buscar Terminal Ayuda
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer1$ gcc -O2 -fopenmp sha
red-clauseModificado.c -o ejecutable
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer1$ ./ejecutable
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
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer1$
```

2. Añadir a lo necesario a `private-clause.c` para que imprima suma fuera de la región `parallel` e inicializar suma a un valor distinto de 0. Ejecute varias veces el código ¿Qué imprime el código fuera del `parallel`? (muéstrelo con una captura de pantalla) ¿Qué ocurre si en esta versión de `private-clause.c` se inicia la variable suma fuera de la construcción `parallel` en lugar de dentro? Razone su respuesta (añada capturas de pantalla que muestren lo que ocurre). Añadir el código con las modificaciones al cuaderno de prácticas.

RESPUESTA: Si lo inicializamos fuera, tendrá un valor nulo a la hora de usarlo dentro de la región paralela

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`



```

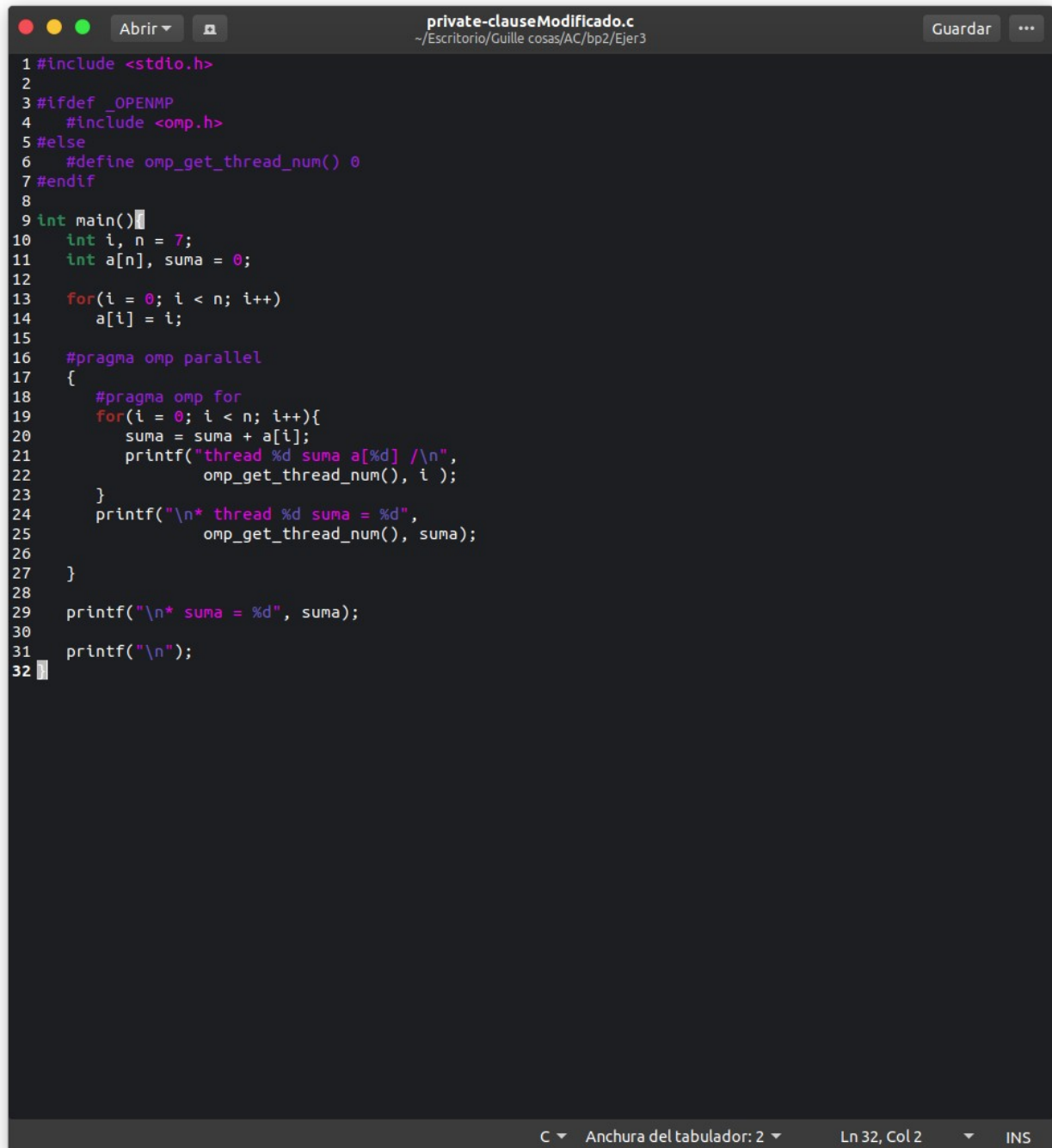
1 #include <stdio.h>
2
3 #ifdef _OPENMP
4     #include <omp.h>
5 #else
6     #define omp_get_thread_num() 0
7 #endif
8
9 int main()
10 {
11     int i, n = 7;
12     int a[n], suma = 0;
13
14     for(i = 0; i < n; i++)
15         a[i] = i;
16
17     #pragma omp parallel private(suma)
18     {
19     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

```

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

RESPUESTA: Se convierte en una variable compartida y hay problemas de dependencias, dando resultados distintos en cada ejecución

CAPTURA CÓDIGO FUENTE: `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 = 0;
13
14     for(i = 0; i < n; i++)
15         a[i] = i;
16
17     #pragma omp parallel
18     {
19         #pragma omp for
20         for(i = 0; i < n; i++){
21             suma = suma + a[i];
22             printf("thread %d suma a[%d] /\n",
23                   omp_get_thread_num(), i );
24         }
25         printf("\n* thread %d suma = %d",
26               omp_get_thread_num(), suma);
27     }
28
29     printf("\n* suma = %d", suma);
30
31     printf("\n");
32 }
```

CAPTURAS DE PANTALLA:

```
Archivo Editar Ver Buscar Terminal Ayuda
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer3$ gcc -o2 -fopenmp private-clauseModificado.c -o ejecutable
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer3$ ./ejecutable
thread 0 suma a[0] /
thread 4 suma a[4] /
thread 2 suma a[2] /
thread 5 suma a[5] /
thread 6 suma a[6] /
thread 3 suma a[3] /
thread 1 suma a[1] /

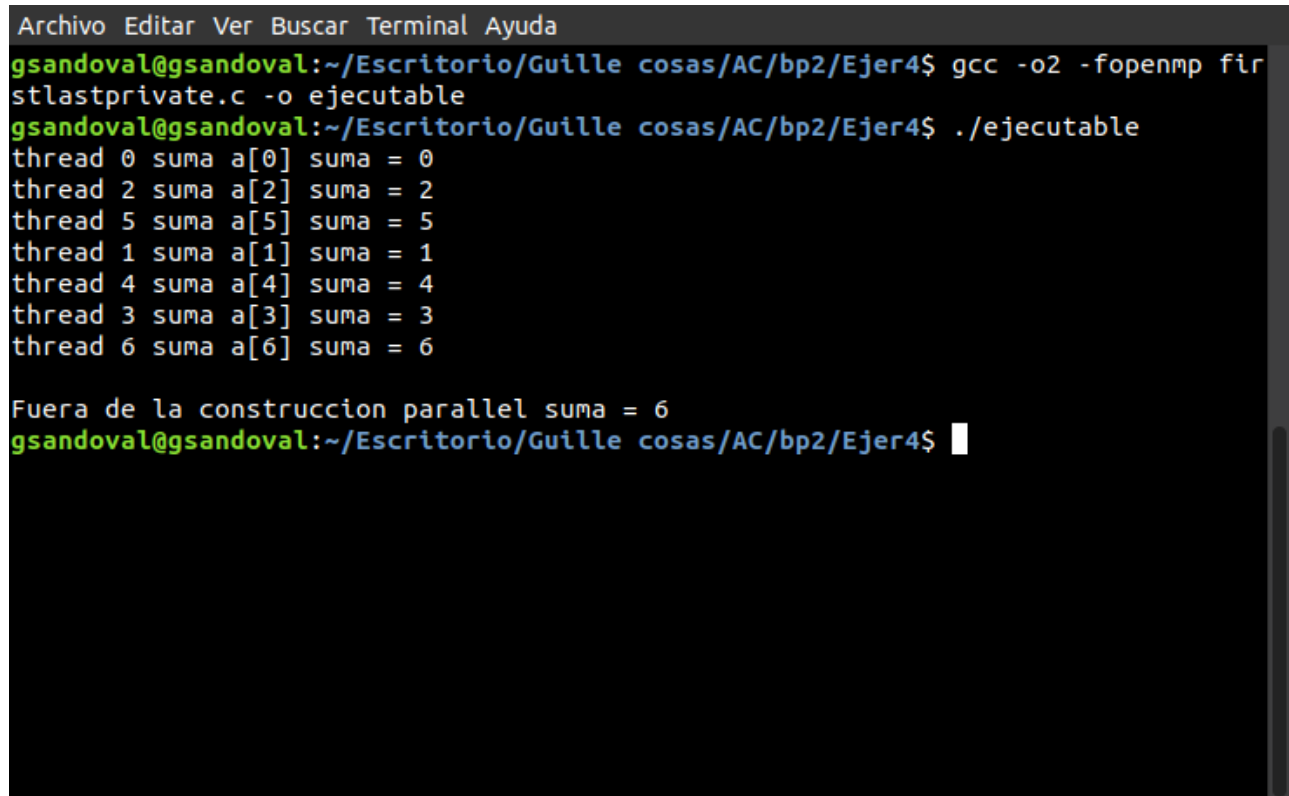
* thread 3 suma = 12
* thread 5 suma = 12
* thread 6 suma = 12
* thread 4 suma = 12
* thread 7 suma = 12
* thread 2 suma = 12
* thread 1 suma = 12
* thread 0 suma = 12
* suma = 12
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer3$ ./ejecutable
thread 0 suma a[0] /
thread 1 suma a[1] /
thread 6 suma a[6] /
thread 3 suma a[3] /
thread 5 suma a[5] /
thread 4 suma a[4] /
thread 2 suma a[2] /

* thread 5 suma = 9
* thread 1 suma = 9
* thread 7 suma = 9
* thread 0 suma = 9
* thread 6 suma = 9
* thread 2 suma = 9
* thread 4 suma = 9
* thread 3 suma = 9
* suma = 9
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer3$
```

4. En la ejecución de `firstlastprivate.c` de la pag. 21 del seminario se imprime un 6 fuera de la región `parallel`. ¿El código imprime siempre 6 fuera de la región `parallel`? Razone su respuesta (añada capturas de pantalla que muestren lo que ocurre).

RESPUESTA: Siempre imprime 6 porque la clausula `lastprivate` le daa el valor del último valor que tome la variable en una ejecución secuencial.

CAPTURAS DE PANTALLA:



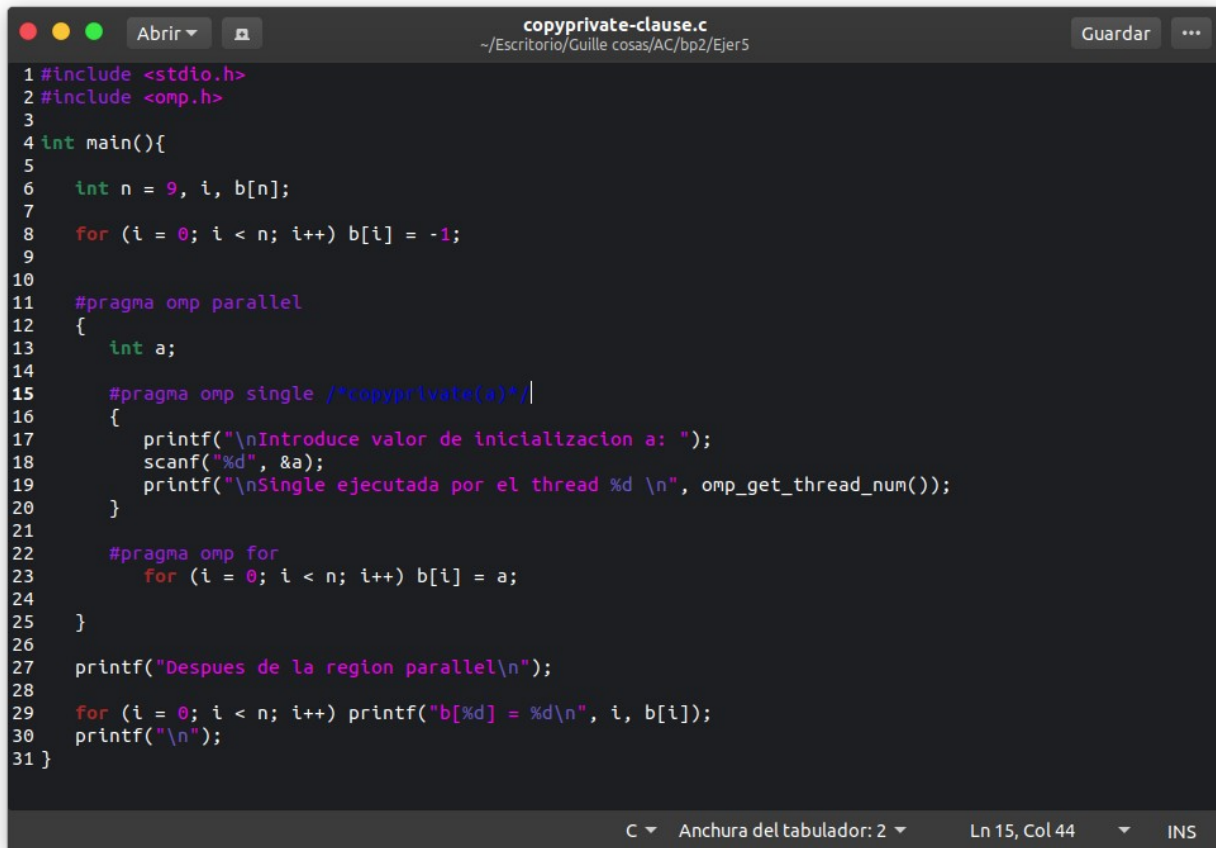
```
Archivo Editar Ver Buscar Terminal Ayuda
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer4$ gcc -o2 -fopenmp fir
stlastprivate.c -o ejecutable
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer4$ ./ejecutable
thread 0 suma a[0] suma = 0
thread 2 suma a[2] suma = 2
thread 5 suma a[5] suma = 5
thread 1 suma a[1] suma = 1
thread 4 suma a[4] suma = 4
thread 3 suma a[3] suma = 3
thread 6 suma a[6] suma = 6

Fuera de la construccion parallel suma = 6
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer4$
```

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

RESPUESTA: Si lo mantenemos, copia todos los valores al valor inicializado, si no, solo lo usa en una de las hebras.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

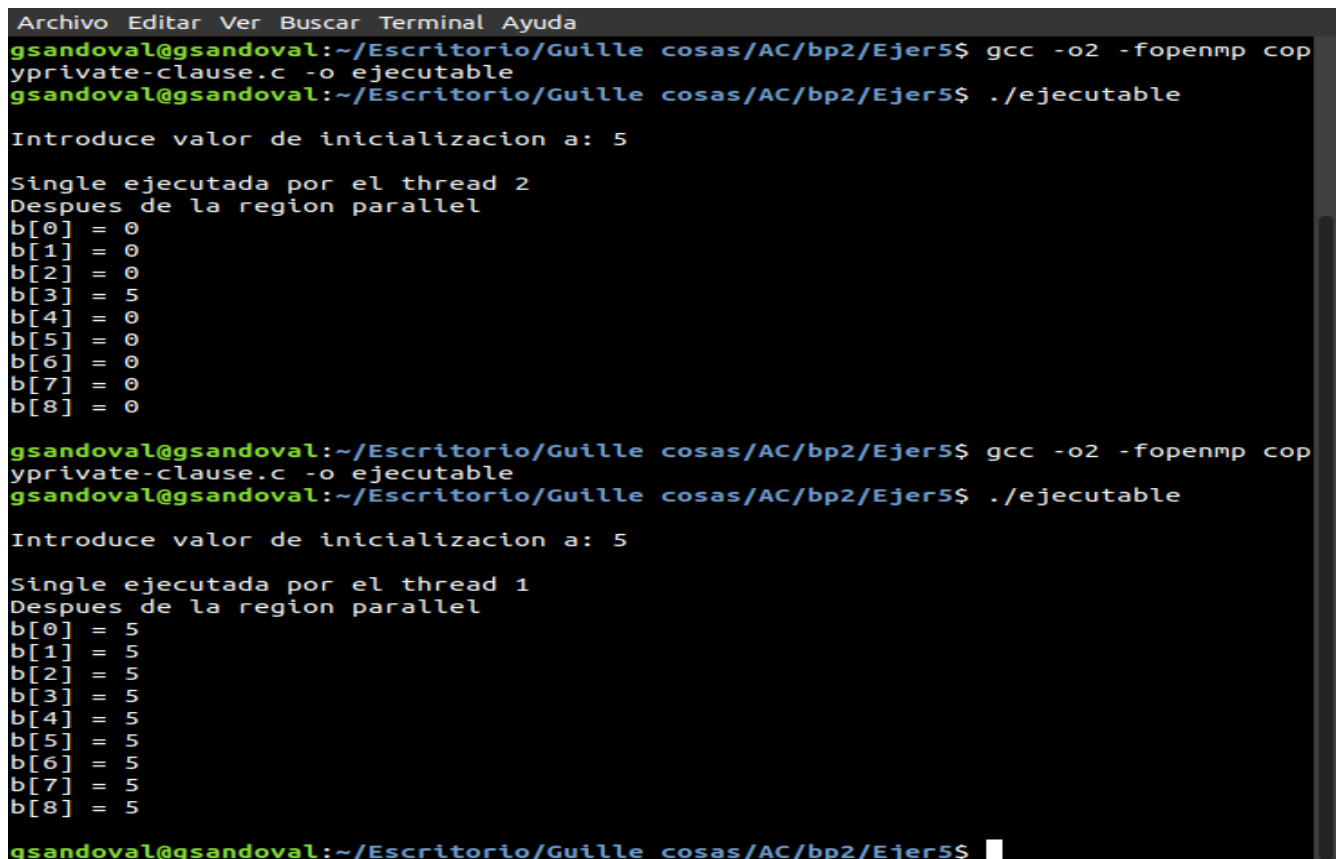


```

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++) b[i] = -1;
9
10
11     #pragma omp parallel
12     {
13         int a;
14
15         #pragma omp single /*copyprivate(a)*/
16         {
17             printf("\nIntroduce valor de inicializacion 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++) b[i] = a;
24     }
25
26     printf("Despues de la region parallel\n");
27
28     for (i = 0; i < n; i++) printf("b[%d] = %d\n", i, b[i]);
29     printf("\n");
30 }
31

```

CAPTURAS DE PANTALLA:



```

Archivo Editar Ver Buscar Terminal Ayuda
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer5$ gcc -o2 -fopenmp cop
yprivate-clause.c -o ejecutable
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer5$ ./ejecutable

Introduce valor de inicializacion a: 5

Single ejecutada por el thread 2
Despues de la region parallel
b[0] = 0
b[1] = 0
b[2] = 0
b[3] = 5
b[4] = 0
b[5] = 0
b[6] = 0
b[7] = 0
b[8] = 0

gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer5$ gcc -o2 -fopenmp cop
yprivate-clause.c -o ejecutable
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer5$ ./ejecutable

Introduce valor de inicializacion a: 5

Single ejecutada por el thread 1
Despues de la region 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

gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer5$

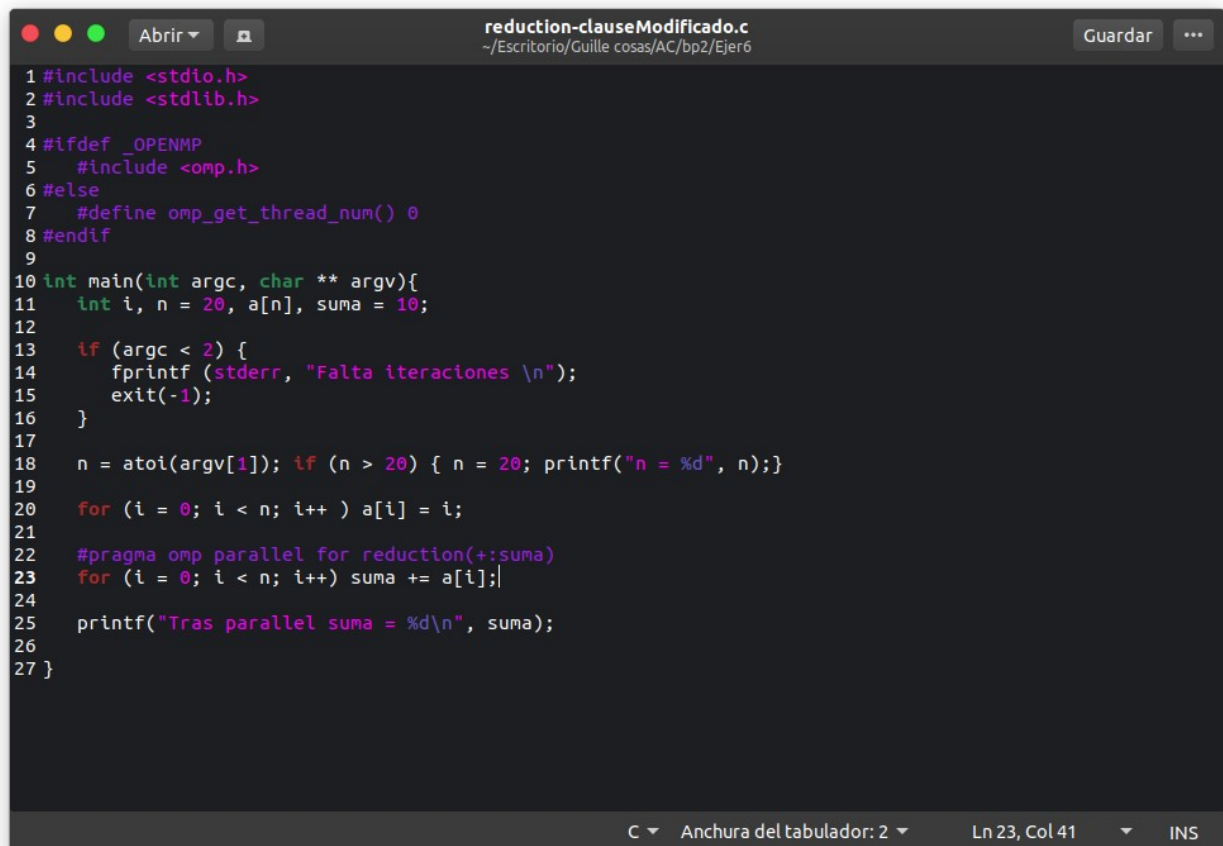
```


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: Porque la cláusula utilizada suma el valor inicial de la variable más el valor de todas las iteraciones privadas

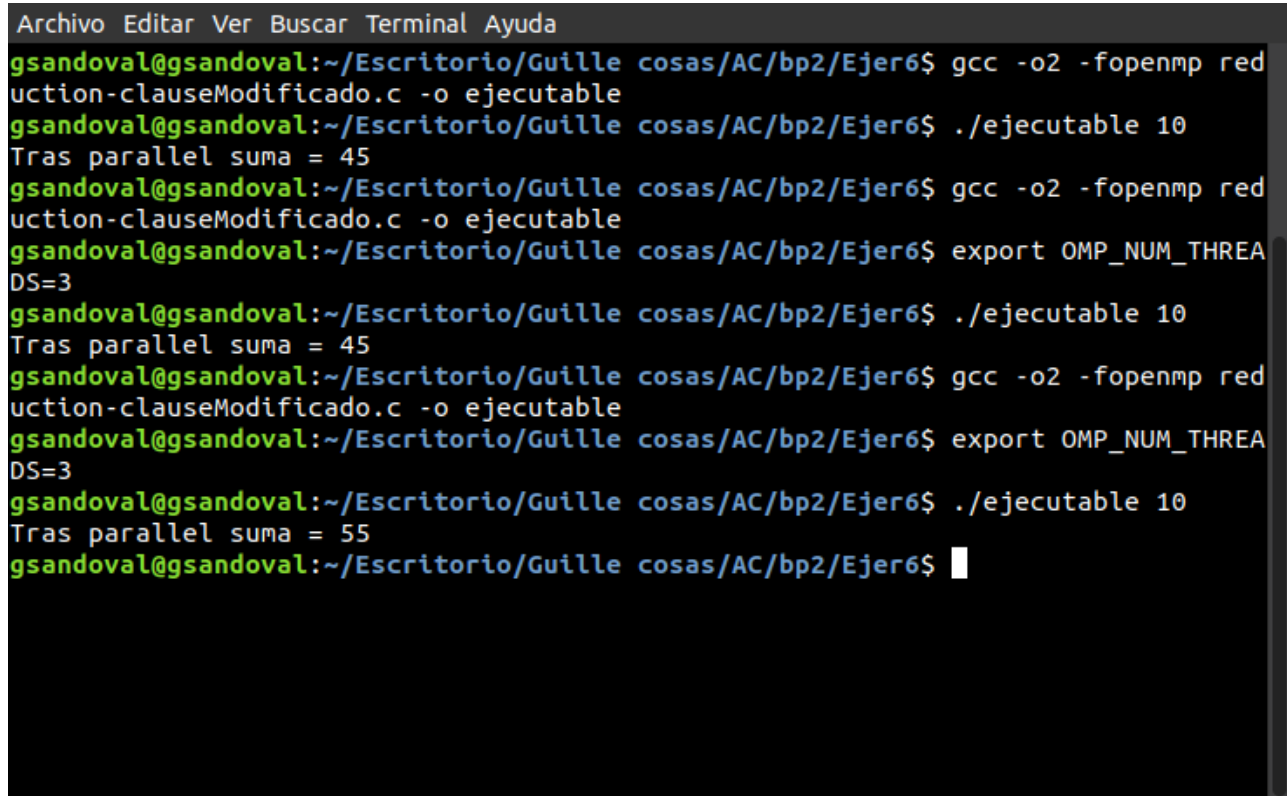
CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`



```

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

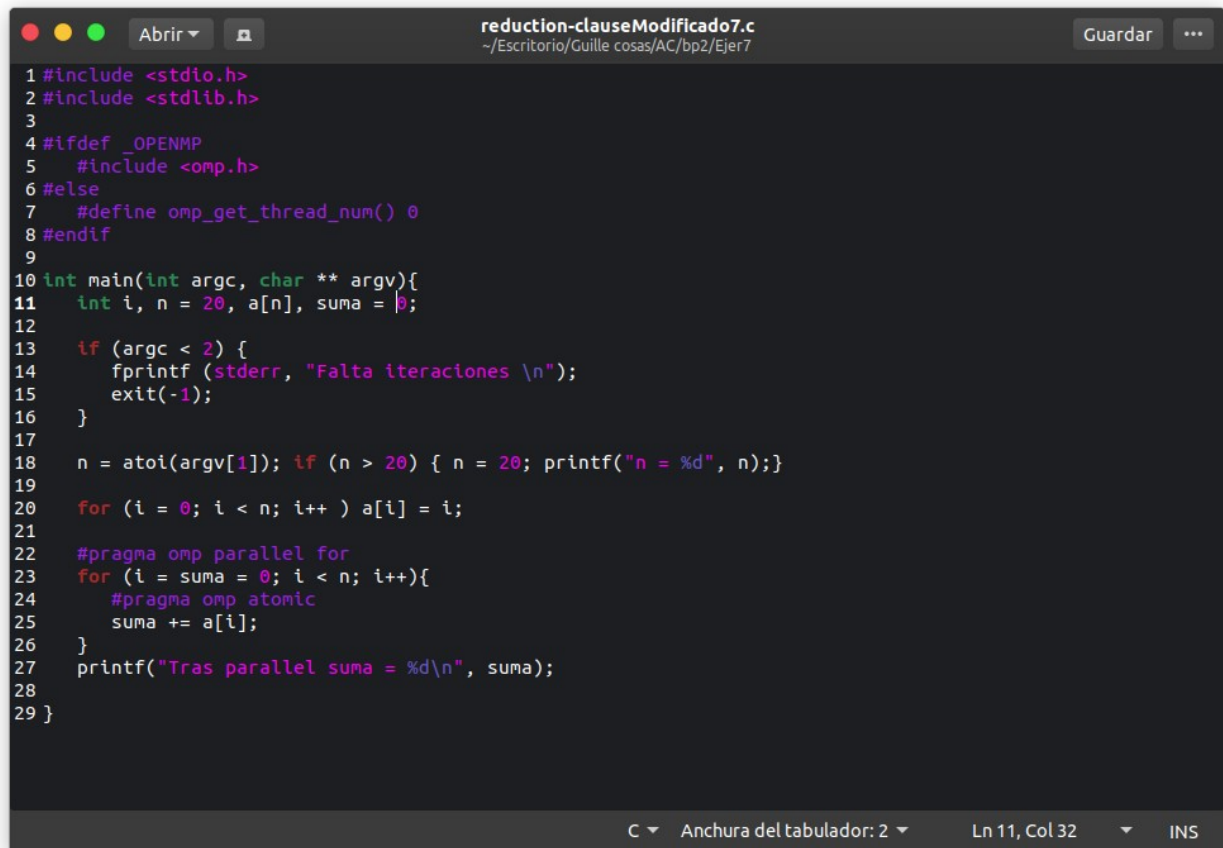

CAPTURAS DE PANTALLA:



```
Archivo Editar Ver Buscar Terminal Ayuda
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer6$ gcc -o2 -fopenmp reduction-clauseModificado.c -o ejecutable
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer6$ ./ejecutable 10
Tras parallel suma = 45
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer6$ gcc -o2 -fopenmp reduction-clauseModificado.c -o ejecutable
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer6$ export OMP_NUM_THREADS=3
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer6$ ./ejecutable 10
Tras parallel suma = 45
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer6$ gcc -o2 -fopenmp reduction-clauseModificado.c -o ejecutable
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer6$ export OMP_NUM_THREADS=3
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer6$ ./ejecutable 10
Tras parallel suma = 55
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer6$
```

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



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

CAPTURAS DE PANTALLA:

Resto de ejercicios

8. Implementar un programa secuencial en C que calcule el producto de una matriz cuadrada, M, por un vector, v1 (implemente una versión para variables globales y otra para variables dinámicas, use una de estas versiones en los siguientes ejercicios):

```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer7$ gcc -o2 -fopenmp reduction-clauseModificado7.c -o ejecutable
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer7$ export OMP_NUM_THREADS=3
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer7$ export OMP_DYNAMIC=FALSE
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer7$ ./ejecutable 5
Tras parallel suma = 4
gsandoval@gsandoval:~/Escritorio/Guille cosas/AC/bp2/Ejer7$ █

```

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

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

CAPTURA CÓDIGO FUENTE: pmv-secuencial.c

CAPTURAS DE PANTALLA:

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):
 - a. una primera que paralelice el bucle que recorre las filas de la matriz y
 - 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 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

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

RESPUESTA:

CAPTURAS DE PANTALLA:

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-OpenmpMP-reduction.c

RESPUESTA:

CAPTURAS DE PANTALLA:

11. Ayudándose de una hoja de cálculo (recuerde que en las aulas está instalado OpenOffice) realice una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del mejor código paralelo de los tres implementados en los ejercicios anteriores para dos tamaños (N) distintos (consulte la Lección 6/Tema 2). Usar `-O2` al compilar. Justificar por qué el código escogido es el mejor. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

CAPTURAS DE PANTALLA (que justifique el código elegido):

TABLA (con tiempos y ganancia) Y GRÁFICA (con ganancia) (para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N-: un N entre 20000 y 100000, y otro entre 5000 y 20000):

COMENTARIOS SOBRE LOS RESULTADOS: