|  |  |  |
| --- | --- | --- |
| **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 claúsula 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).

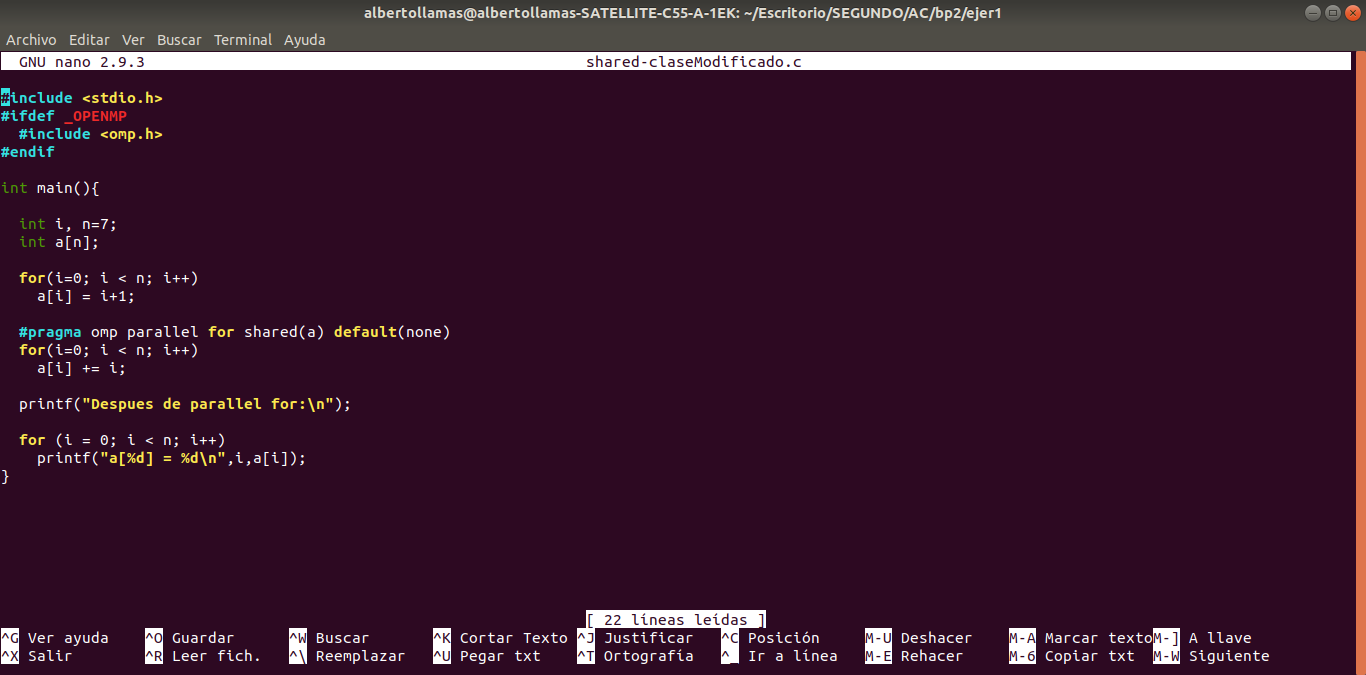
**b)** Se corrige incluyendo la variable “n” en la lista de la claúsula shared.

**CAPTURA CÓDIGO FUENTE**: shared-clauseModificado.c

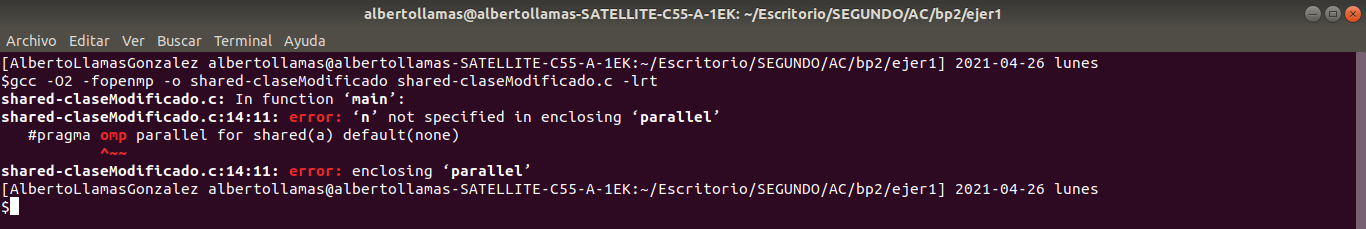
|  |
| --- |
| capturas/ejer1/ocdigoBUeno.png |

**CAPTURAS DE PANTALLA:**

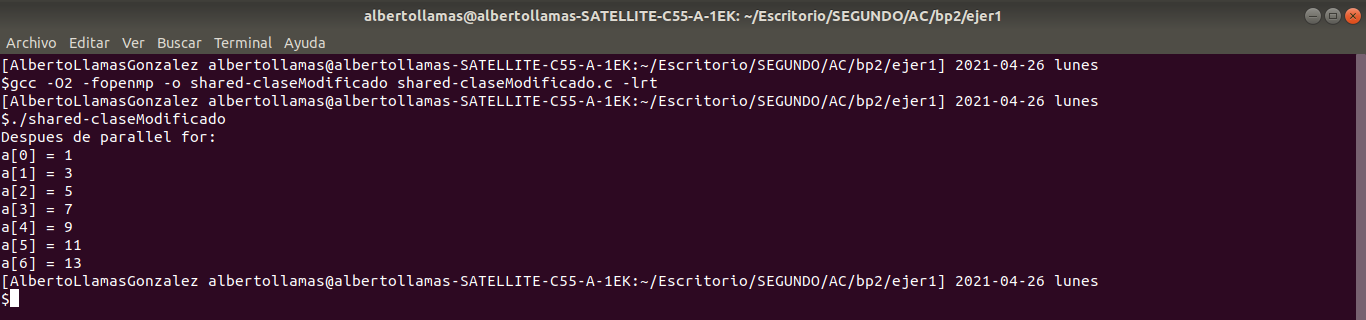
Código donde se produce el error



Error:



Ejecución correcta:



1. **(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.

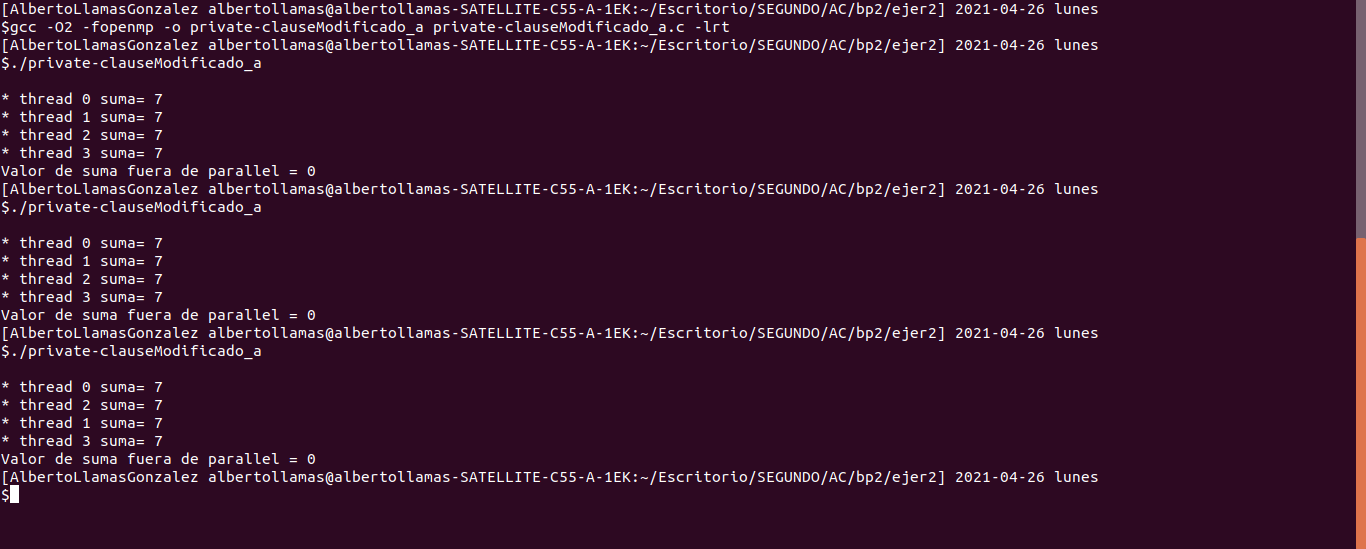
**(a) RESPUESTA**:

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

|  |
| --- |
| capturas/ejer2/codigoA.png |

**CAPTURAS DE PANTALLA:**

****

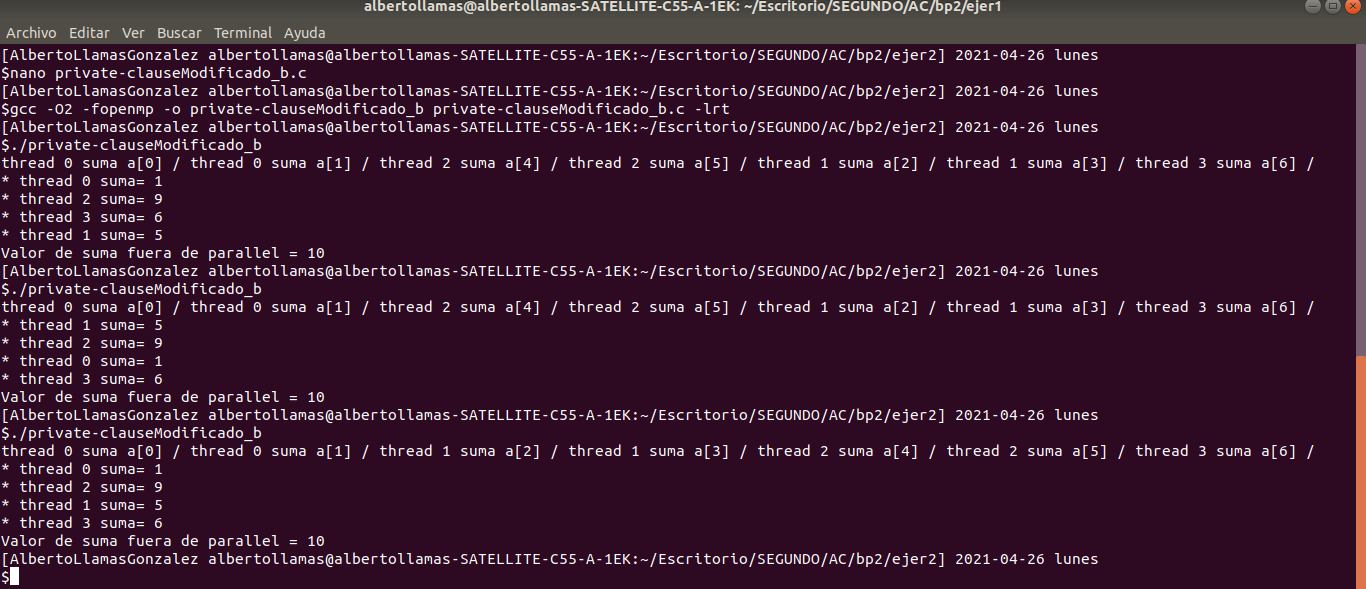
**(b) RESPUESTA**:

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

|  |
| --- |
| capturas/ejer2/codigoB.png |

**CAPTURAS DE PANTALLA:**



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

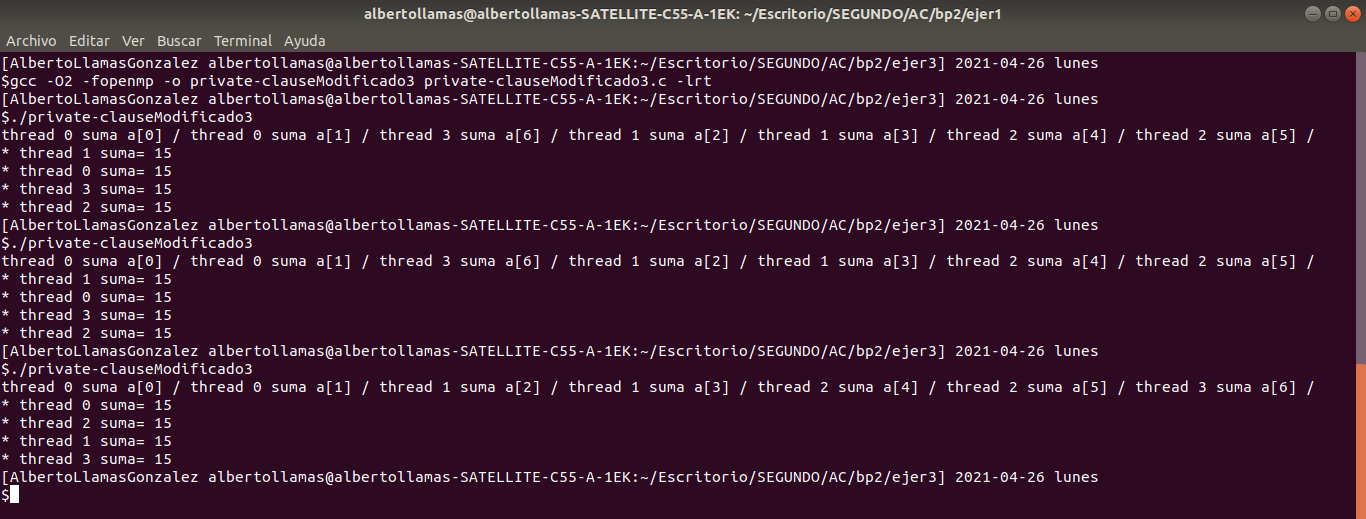
**RESPUESTA**:

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

|  |
| --- |
| capturas/ejer3/codigo.png |

**CAPTURAS DE PANTALLA:**

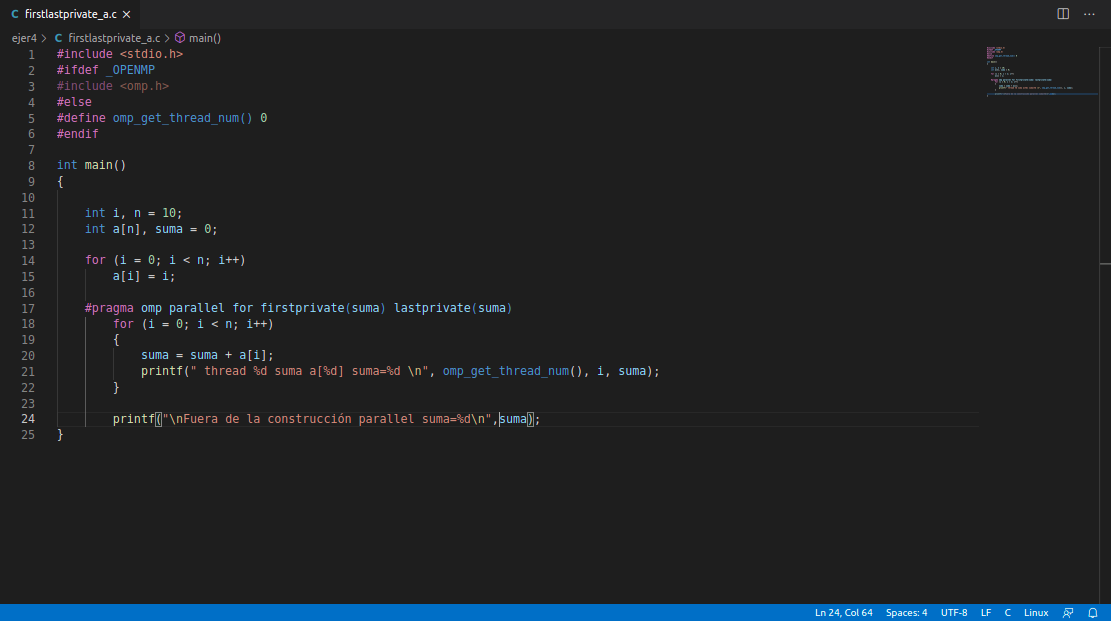


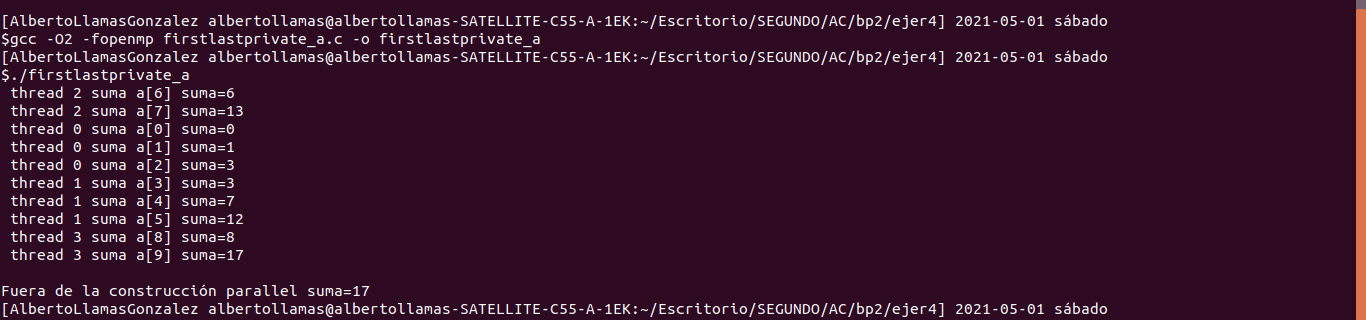
1. En la ejecución de firstlastprivate.c de la pag. 21 del seminario se imprime un 6 fuera de la región parallel. **(a)** Cambiar el tamaño del vector a 10. Razonar lo que imprime el código en su PC con esta modificación. (añadir capturas de pantalla que muestren lo que ocurre). **(b)** Sin cambiar el 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:**

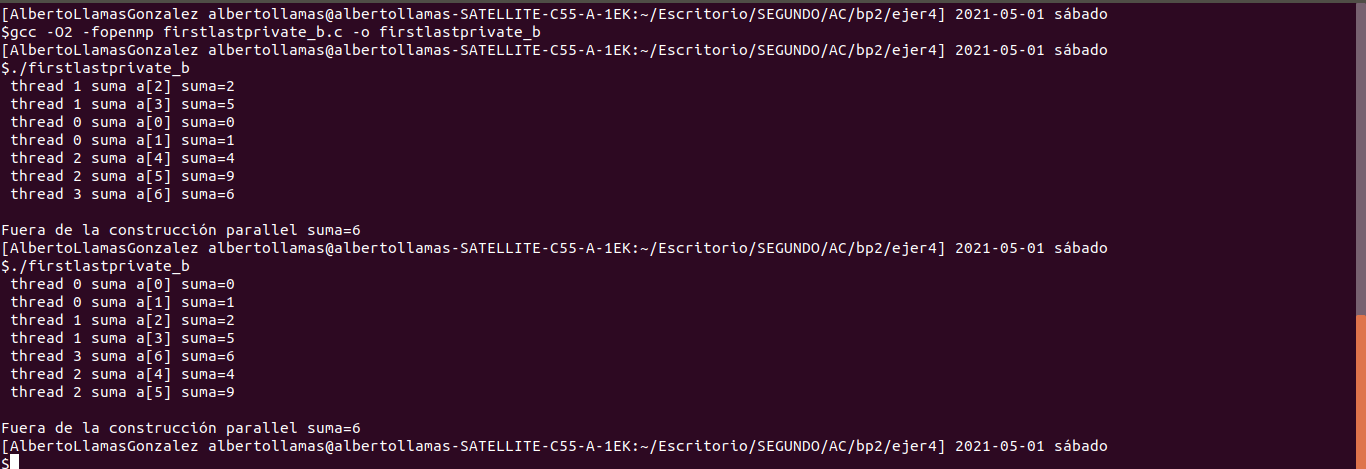
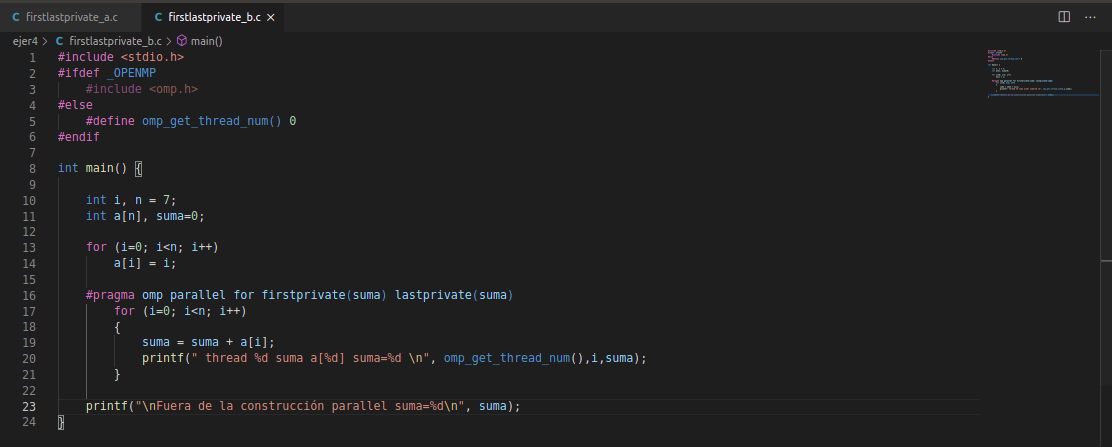
****

****

**(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:**



1. **(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)

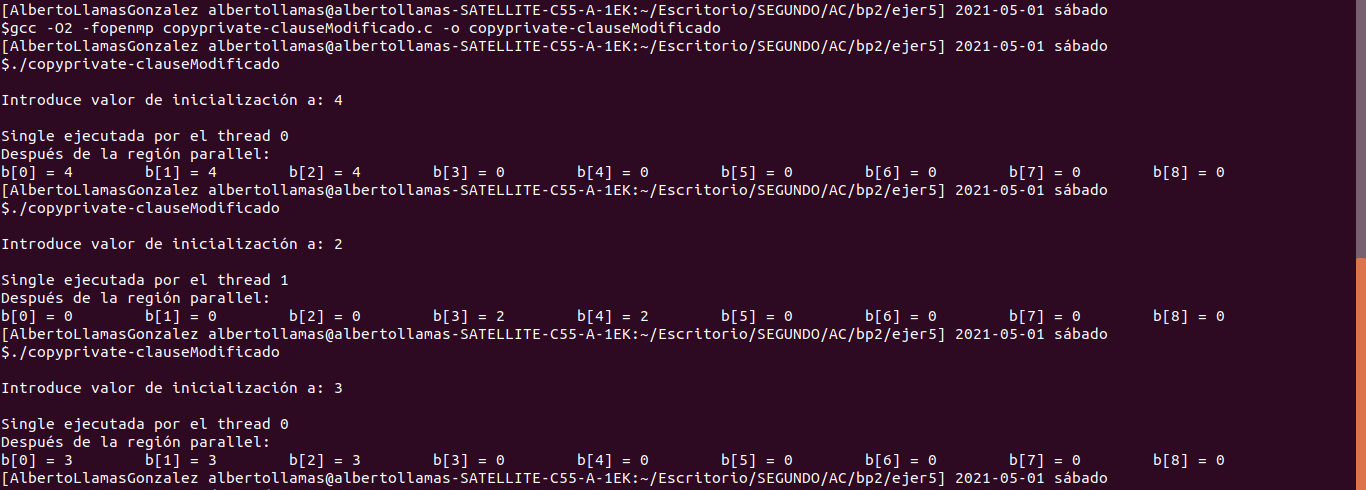
**RESPUESTA**:

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

|  |
| --- |
| capturas/ejer5/Captura%20de%20pantalla%20de%202021-05-01%2016-31-20.png |

**CAPTURAS DE PANTALLA:**



1. 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)

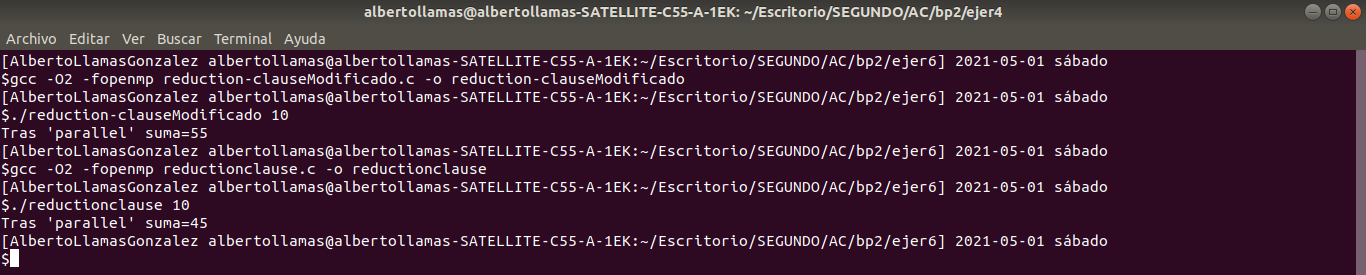
**RESPUESTA**: Queda el valor de suma + 10, porque master se queda con el valor 10. Inicializa las variables locales, no el valor de suma fuera.

**CAPTURA CÓDIGO FUENTE**: reduction-clauseModificado.c

|  |
| --- |
| capturas/ejer6/Captura%20de%20pantalla%20de%202021-05-01%2018-52-42.png |

**CAPTURAS DE PANTALLA:**

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

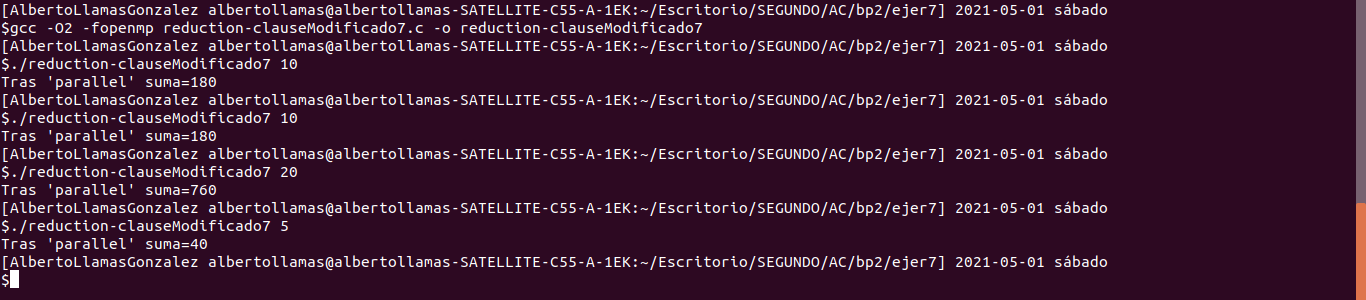


1. En el ejemplo reduction-clause.c, elimine reduction() de #pragma omp parallel for reduction(+:suma) y haga las modificaciones necesarias para que se siga realizando la suma de los componentes del vector a en paralelo sin añadir más directivas de trabajo compartido (añada capturas de pantalla que muestren lo que ocurre).

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

|  |
| --- |
| capturas/ejer7/Captura%20de%20pantalla%20de%202021-05-01%2018-53-41.png |

**CAPTURAS DE PANTALLA:**



# Resto de ejercicios (usar en atcgrid la cola ac a no ser que se tenga que usar atcgrid4)

1. 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):



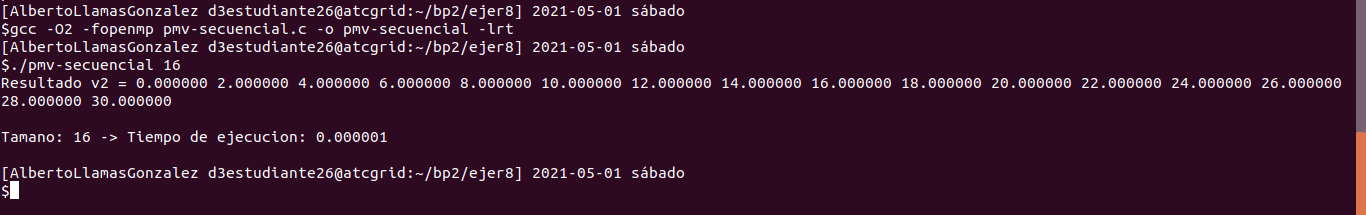
NOTAS: (1) el número de filas /columnas N de la matriz deben ser argumentos de entrada al programa; (2) se debe inicializar la matriz y el vector antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para tamaños 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

|  |
| --- |
| capturas/ejer8/Captura%20de%20pantalla%20de%202021-05-01%2018-56-29.pngcapturas/ejer8/Captura%20de%20pantalla%20de%202021-05-01%2018-56-40.pngcapturas/ejer8/Captura%20de%20pantalla%20de%202021-05-01%2018-56-53.pngcapturas/ejer8/Captura%20de%20pantalla%20de%202021-05-01%2018-57-34.png |

**CAPTURAS DE PANTALLA:**



1. 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):
   1. una primera que paralelice el bucle que recorre las filas de la matriz y
   2. 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:



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

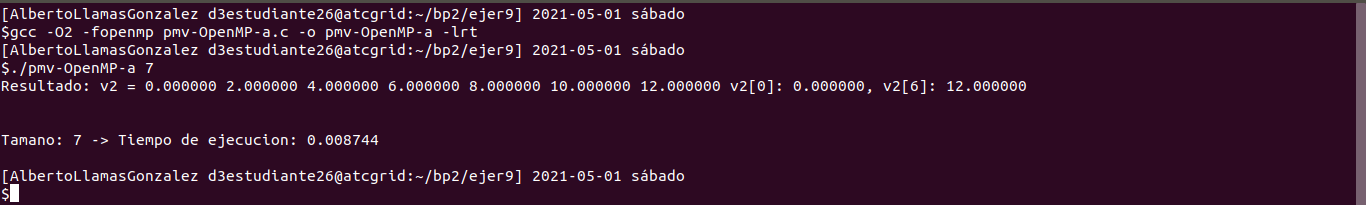
|  |
| --- |
| capturas/ejer9/Captura%20de%20pantalla%20de%202021-05-01%2018-58-19.png |

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

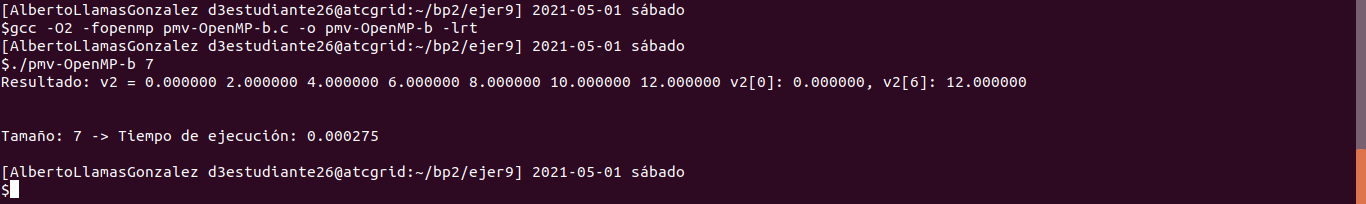
|  |
| --- |
| capturas/ejer9/Captura%20de%20pantalla%20de%202021-05-01%2018-58-31.png |

**CAPTURAS DE PANTALLA:**

Ejecución **a)**



Ejecución **b)**

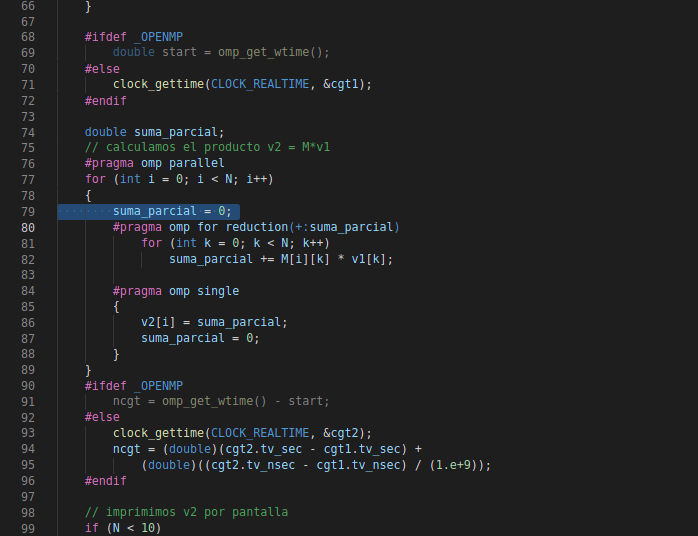


1. A partir de la segunda versión de código paralelo desarrollado en el ejercicio anterior, implementar una versión paralela del producto matriz por vector con OpenMP que use para comunicación/sincronización la cláusula reduction. Respecto a este ejercicio:
   * Anote en su cuaderno de prácticas todos 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:



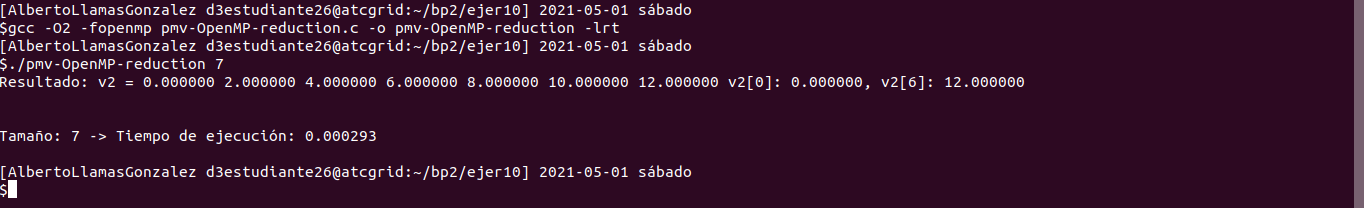
El problema era que el resultado de v2 era siempre 0 salvo en la última posición por lo que para solucionarlo, simplemente había que eliminar dicha línea de código.

Se muestra la parte módificada del código anterior:

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

|  |
| --- |
| capturas/ejer10/Captura%20de%20pantalla%20de%202021-05-01%2018-59-38.png |

**CAPTURAS DE PANTALLA:**



1. (**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:**