

BP2RodriguezJimenezFrancisco.pdf

**CAZZ****Arquitectura de Computadores****2º Grado en Ingeniería Informática****Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación
Universidad de Granada**

Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.





**KEEP
CALM
AND
ESTUDIA
UN POQUITO**



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Francisco Rodríguez Jiménez

Grupo de prácticas y profesor de prácticas: C1 Juan José Escobar

Fecha de entrega:

Fecha evaluación en clase:

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con las 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:

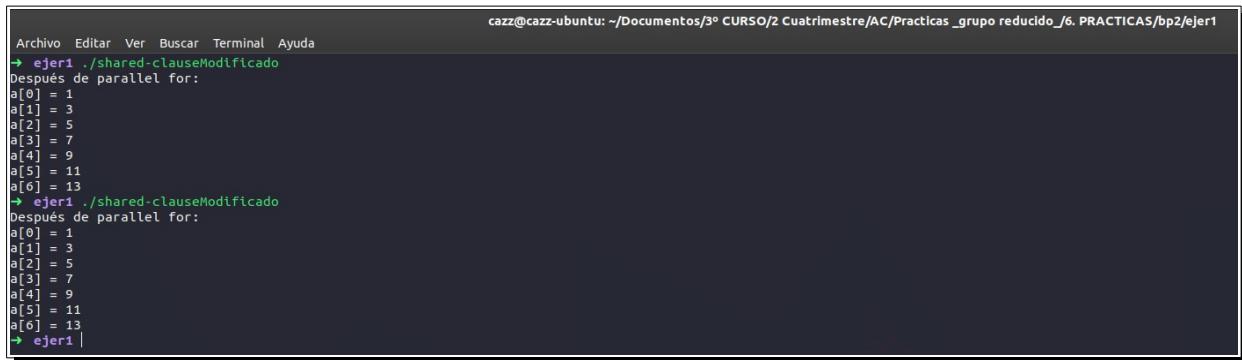
Lo que ocurre, es que con la cláusula `default(None)` se debe especificar el alcance de todas las variables usadas en la construcción. En este caso, se debe especificar que el vector `a` es compartido, y la variable `n` también. La variable `i`, como se usa como índice en el bucle `for` no hace falta especificarlo, siempre va a ser privado.

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

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

`shared-clauseModificado.c`

CAPTURAS DE PANTALLA:



```
cazz@cazz-ubuntu: ~/Documentos/3º CURSO/2 Cuatrimestre/AC/Prácticas _grupo reducido_/6. PRACTICAS_bp2/ejer1
Archivo Editar Ver Buscar Terminal Ayuda
→ ejer1 ./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
→ ejer1 ./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
→ 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`? (muéstrela 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:

El código fuera del `parallel` imprime el valor de `suma` = 0, ya que al establecer privada la variable `suma`, con la clausula `private`, cada hebra calcula su suma privada, y al salir de la región paralela no se guarda el resultado en la variable `suma` original.

Si se inicializa la variable `suma` fuera de la construcción ocurre lo mismo, no se guarda el resultado en la variable `suma` original, e imprime el valor con la que se inicializo en el trozo secuencial.

CAPTURA CÓDIGO FUENTE: private-clauseModificado.c

```
C private-clauseModificado.c x
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     int i, n = 7;
10    int a[n], suma;
11
12    for (i=0; i<n; i++)
13        a[i] = i;
14
15    #pragma omp parallel private(suma)
16    {
17        suma=5;
18        #pragma omp for
19        for (i=0; i<n; i++){
20            suma = suma + a[i];
21            printf("thread %d suma a[%d] / ", omp_get_thread_num(), i);
22        }
23    }
24    printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
25    printf("\n");
26}
27 }
```

código private-clauseModificado.c (v.1)

```
C private-clauseModificado.c x
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     int i, n = 7;
10    int a[n], suma = 5;
11
12    for (i=0; i<n; i++)
13        a[i] = i;
14
15    #pragma omp parallel private(suma)
16    {
17        //suma=5;
18        #pragma omp for
19        for (i=0; i<n; i++){
20            suma = suma + a[i];
21            printf("thread %d suma a[%d] / ", omp_get_thread_num(), i);
22        }
23    }
24    printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
25    printf("\n");
26}
27 }
```

código private-clauseModificado.c (v.2)

CAPTURAS DE PANTALLA:

```
cazz@cazz-ubuntu: ~/Documentos/3º CURSO/2 Cuatrimestre/AC/Prácticas_grupo reducido_6. PRACTICAS/bp2/ejer2
Archivo Editar Ver Buscar Terminal Ayuda
→ ejer2 ./private-clauseModificado
thread 1 suma a[2] / thread 1 suma a[3] / thread 2 suma a[4] / thread 2 suma a[5] / thread 0 suma a[0] / thread 0 suma a[1] / thread 3 suma a[6] /
* thread 0 suma= 0
→ ejer2 ./private-clauseModificado
thread 1 suma a[2] / thread 1 suma a[3] / thread 2 suma a[4] / thread 2 suma a[5] / thread 0 suma a[0] / thread 0 suma a[1] / thread 3 suma a[6] /
* thread 0 suma= 0
→ ejer2 |
ejecución v1
```

```
cazz@cazz-ubuntu: ~/Documentos/3º CURSO/2 Cuatrimestre/AC/Prácticas_grupo reducido_6. PRACTICAS/bp2/ejer2
Archivo Editar Ver Buscar Terminal Ayuda
→ ejer2 ./private-clauseModificado
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= 5
→ ejer2 ./private-clauseModificado
thread 1 suma a[2] / thread 1 suma a[3] / thread 2 suma a[4] / thread 2 suma a[5] / thread 0 suma a[0] / thread 0 suma a[1] / thread 3 suma a[6] /
* thread 0 suma= 5
→ ejer2 |
ejecución v2
```

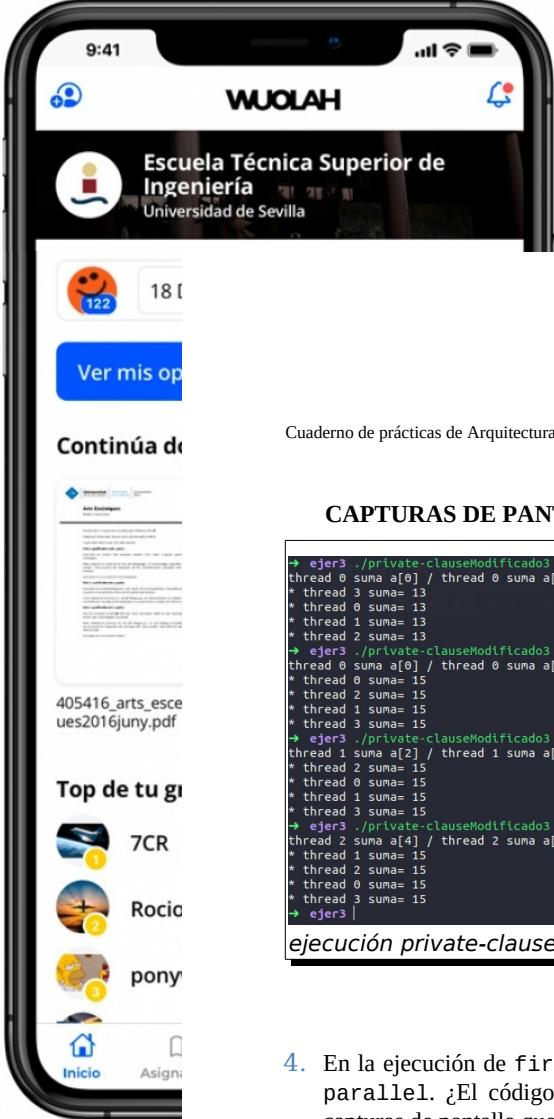
3. ¿Qué ocurre si en `private-clause.c` se elimina la cláusula `private(suma)`? ¿A qué cree que es debido?

RESPUESTA:

Que suma se vuelve variable compartida para todas las hebras, y tendrá un valor incorrecto de la última hebra que escribió en dicha variable.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

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



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática

CAPTURAS DE PANTALLA:

```
↳ ejer3 ./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 3 suma= 13
* thread 0 suma= 13
* thread 1 suma= 13
* thread 2 suma= 13
↳ ejer3 ./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
↳ ejer3 ./private-clauseModificado3
thread 1 suma a[2] / thread 1 suma a[3] / thread 2 suma a[4] / thread 2 suma a[5] / thread 0 suma a[0] / thread 0 suma a[1] / thread 3 suma a[6] /
* thread 2 suma= 15
* thread 0 suma= 15
* thread 1 suma= 15
* thread 3 suma= 15
↳ ejer3 ./private-clauseModificado3
thread 2 suma a[4] / thread 2 suma a[5] / thread 1 suma a[2] / thread 1 suma a[3] / thread 0 suma a[0] / thread 0 suma a[1] / thread 3 suma a[6] /
* thread 1 suma= 15
* thread 2 suma= 15
* thread 0 suma= 15
* thread 3 suma= 15
↳ ejecución v2
↳ ejecución v2
↳ ejecución private-clauseModificado3.c
```

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:

No, eso es debido a que también se ha añadido una directiva `lastprivate`, la cual copia (al salir de región paralela) del último valor (en una ejecución secuencial) de las variables de la lista.
En el caso de un bucle, el valor de la última iteración.

CAPTURAS DE PANTALLA:

```
C firstprivate-clause.c ✘
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      int i, n = 7;
10     int a[n], suma=0;
11
12     for (i=0; i<n; i++)
13         a[i] = i;
14
15     #pragma omp parallel for firstprivate(suma) lastprivate(suma)
16     for (i=0; i<n; i++){
17         suma = suma + a[i];
18         printf(" thread %d suma a[%d] suma=%d \n", omp_get_thread_num(),i,suma);
19     }
20
21     printf("\nFuera de la construcción parallel suma=%d\n", suma);
22 }
```

código *firstprivate-clause.c* (mod)

```
→ ejer4 ./firstprivate-clause
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
→ ejer4 ./firstprivate-clause
```

```
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9
thread 3 suma a[6] suma=6
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
```

```
Fuera de la construcción parallel suma=6
→ ejer4 □
```

ejecución *firstprivate-clause.c* (mod)

Si quitamos la clausula **lastprivate**(suma), obtendremos valor 0 como resultado, el cual se inicializo la variable suma fuera de la región paralela. La clausula **firstprivate** realizaría una difusión del valor suma a todas las variables privadas suma de cada hebra. No guarda el valor de la suma en la variable global suma. Cada hebra guarda su valor parcial en su variable privada suma.

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:

Los resultados a la hora de inicializar las componentes del vector, no son todas las del valor introducido, ya que la clausula **copyprivate** permite que una variable privada de un thread se copie a las variables privadas del mismo nombre del resto de threads (difusión), si quitamos esta clausula, las variable privada del mismo nombre no tendrá el mismo valor en otro thread.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```
C copyprivate-clause.c ×
1  #include <stdio.h>
2  #include <stdlib.h>
3  #ifdef _OPENMP
4      #include <omp.h>
5  #else
6      #define omp_get_thread_num() 0
7  #endif
8
9  int main() {
10     int n = 9, i, b[n];
11
12     for (i=0; i<n; i++)
13         b[i] = -1;
14
15     #pragma omp parallel
16     {
17         int a;
18         #pragma omp single //copyprivate(a)
19         {
20             printf("\nIntroduce valor de inicialización a: ");
21             scanf("%d", &a );
22             printf("\nSingle ejecutada por el thread %d\n",
23                   omp_get_thread_num());
24         }
25         #pragma omp for
26             for (i=0; i<n; i++)
27                 b[i] = a;
28     }
29
30     printf("Después de la región parallel:\n");
31     for (i=0; i<n; i++)
32         printf("b[%d] = %d\t,i,b[i]);"
33
34     printf("\n");
35
36 }
```

CAPTURAS DE PANTALLA:

```
→ ejer5 ./copyprivate-clause
Introduce valor de inicialización a: 5
Single ejecutada por el thread 1
Depois da rexión paralela:
b[0] = 0      b[1] = 0      b[2] = 0      b[3] = 5      b[4] = 5      b[5] = 0      b[6] = 0      b[7] = 0      b[8] = 0
→ ejer5 ./copyprivate-clause
Introduce valor de inicialización a: 3
Single ejecutada por el thread 3
Depois da rexión paralela:
b[0] = 0      b[1] = 0      b[2] = 0      b[3] = 0      b[4] = 0      b[5] = 0      b[6] = 0      b[7] = 3      b[8] = 3
```

ejecución copyprivate-clause.c

6. En el ejemplo `reduction-clause.c` sustituya `suma=0` por `suma=10`. ¿Qué resultado se imprime ahora? Justifique el resultado (añada capturas de pantalla que muestren lo que ocurre)

RESPUESTA:

Imprime el resultado de la suma realizado, pero sumándole 10 al valor correcto. Por eso en la cláusula de tipo **reduction**, el operador de reducción para suma, el valor inicial de la variable donde se va almacenar la suma debe ser 0.

CAPTURA CÓDIGO FUENTE: reduction-clauseModificado.c

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



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the App Store GET IT ON Google Play

Continúa de

Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática

CAPTURAS DE PANTALLA:

```
→ ejer6 g++ -O2 -fopenmp reduction-clause.c -o reduction-clause
→ ejer6 ./reduction-clause 3 ; ./reduction-clause 10
Tras 'parallel' suma=3
Tras 'parallel' suma=45
→ ejer6 g++ -O2 -fopenmp reduction-clause.c -o reduction-clause
→ ejer6 ./reduction-clause 3 ; ./reduction-clause 10
Tras 'parallel' suma=13
Tras 'parallel' suma=55
→ ejer6
```

ejecución reduction-clause.c (mod)

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

CAPTURAS DE PANTALLA:

```

→ ejer7 g++ -O2 -fopenmp reduction-clause.c -o reduction-clause
→ ejer7 ./reduction-clause 5
Tras 'parallel' suma=9
→ ejer7 ./reduction-clause 5
Tras 'parallel' suma=5
→ ejer7 ./reduction-clause 5
Tras 'parallel' suma=7
→ ejer7 g++ -O2 -fopenmp reduction-clause.c -o reduction-clause
→ ejer7 ./reduction-clause 5
Tras 'parallel' suma=10
→ ejer7 ./reduction-clause 5
Tras 'parallel' suma=10
→ ejer7 ./reduction-clause 5
Tras 'parallel' suma=10
→ ejer7 █

```

ejecución reduction-clause.c (mod) (sin reduction),(sin reduction y con atomic)

Resto de ejercicios

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

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

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

CAPTURA CÓDIGO FUENTE: pmv-secuencial.c

CAPTURAS DE PANTALLA:

- 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 paraleliza el bucle que recorre las filas de la matriz y
- b. una segunda que paraleliza el bucle que recorre las columnas.

Use las directivas que estime oportunas y las cláusulas que sean necesarias **excepto la cláusula reduction**. Se debe parallelizar 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 -OpenMP - 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: