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):
Grupo de prácticas y profesor de prácticas:
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: Da error de compilación ya que la cláusula default(none), obliga al programador a especificar como se comparten los atributos (privados o compartidos), y en este caso no se especifica para la variable n (i no importa por ser el indice del bucle for).

   CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

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

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

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

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

printf("Después de parallel for:\n");

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

   CAPTURAS DE PANTALLA:

```
File  Edit  View  Search  Terminal  Help
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer1] 2020-03-30 lu
nes
$gcc -O2 -fopenmp -o shared-clause shared-clause.c
shared-clause.c:6:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^~~~
shared-clause.c: In function ‘main’:
shared-clause.c:14:9: error: ‘n’ not specified in enclosing ‘parallel’
 #pragma omp parallel for default(none) shared(a)
         ^~~
shared-clause.c:14:9: error: enclosing ‘parallel’
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer1] 2020-03-30 lu
nes
$
```

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: Fuera del parallel el código imprime siempre suma=0. Si la variable suma se inicializa fuera del parallel, al mostrar dicha variable, al final del código, fuera del parallel, esta conserva el valor al que se inicializó, ya que las operaciones que se realizan dentro del parallel las realiza cada thread sobre una copia de suma que es privada a cada thread, y estas operaciones no tienen efecto sobre la variable suma fuera del parallel.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`

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

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

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

#pragma omp parallel private(suma)
{
        suma=10;
        #pragma omp for
        for(i=0; i < n; i++)
        {
                suma=suma + a[i];
                printf("thread %d suma a[%d]/", omp_get_thread_num(),i);
        }

        printf("\n* thread %d suma= %d", omp_get_thread_num(),suma);
}

printf("\nSuma = %d", suma);

printf("\n");
}
```

Para la segunda parte del ejercicio 2 inicializo suma con int suma=5.(No pongo captura pues el código es igual excepto en esa parte).

CAPTURAS DE PANTALLA:

Ejecución sin inicializar suma fuera del parallel.

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer2] 2020-03-30 lu
nes
$gcc -O2 -fopenmp -o private-clause private-clause.c
private-clause.c:8:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^~~~
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer2] 2020-03-30 lu
nes
$export OMP_DYNAMIC=FALSE
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer2] 2020-03-30 lu
nes
$export OMP_NUM_THREADS=4
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer2] 2020-03-30 lu
nes
$./private-clause
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 s
uma a[1]/thread 3 suma a[6]/
* thread 1 suma= 15
* thread 2 suma= 19
* thread 0 suma= 11
* thread 3 suma= 16
Suma = 0
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer2] 2020-03-30 lu
nes
$
```

Ejecución con suma inicializado a 5 fuera del parallel.

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer2] 2020-03-30 lu
nes
$gcc -O2 -fopenmp -o private-clause private-clause.c
private-clause.c:8:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^~~~
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer2] 2020-03-30 lu
nes
$./private-clause
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 s
uma a[1]/thread 3 suma a[6]/
* thread 2 suma= 19
* thread 0 suma= 11
* thread 1 suma= 15
* thread 3 suma= 16
Suma = 5
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer2] 2020-03-30 lu
nes
$
```

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

   RESPUESTA: Ocurre que todos los threads muestran el mismo valor para suma, y además el valor de suma tras el parallel, también es el mismo que el de los threads. Esto se debe a que, al tener la directiva for, una barrera implícita al final, el valor obtenido por cada thread para suma no se muestra hasta que todas han terminado la ejecución del bucle. Como la variable suma no está marcada como privada, se entiende como una variable compartida por todas las hebras. Por tanto al mostrar lo obtenido por cada hebra realmente se está mostrando lo que se ha obtenido al terminar de ejecutar todas, y más concretamente, el último valor que se haya almacenado en la variable suma. Además al ya no ser privada, fuera del parallel se mostrará el mismo valor que se ha obtenido para todas las hebras.

   CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

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

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

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

#pragma omp parallel
{
        suma=10;
        #pragma omp for
        for(i=0; i < n; i++)
        {
                suma=suma + a[i];
                printf("thread %d suma a[%d]/", omp_get_thread_num(),i);
        }

        printf("\n* thread %d suma= %d", omp_get_thread_num(),suma);
}

printf("\nSuma = %d", suma);

printf("\n");
}
```

CAPTURAS DE PANTALLA:

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer3] 2020-03-30 lu
nes
$gcc -O2 -fopenmp -o private-clause private-clause.c
private-clause.c:8:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^~~~
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer3] 2020-03-30 lu
nes
$./private-clause
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 2 s
uma a[4]/thread 2 suma a[5]/
* thread 3 suma= 23
* thread 0 suma= 23
* thread 1 suma= 23
* thread 2 suma= 23
Suma = 23
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer3] 2020-03-30 lu
nes
$./private-clause
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 s
uma a[5]/thread 3 suma a[6]/
* thread 0 suma= 25
* thread 1 suma= 25
* thread 2 suma= 25
* thread 3 suma= 25
Suma = 25
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer3] 2020-03-30 lu
nes
$
```

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: El código siempre imprime 6 fuera de la región parallel, ya que, aunque al llevar la cláusula firstprivate se esperaría que suma contuviese suma=suma+a[0], al llevar la cláusula lastprivate, se sustituye este primer valor asignado por el de la última iteración del bucle, es decir, suma=suma + a[6].

CAPTURAS DE PANTALLA:

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer4] 2020-03-30 lu
nes
$gcc -O2 -fopenmp -o firstlastprivate firstlastprivate.c
firstlastprivate.c:8:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^~~~
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer4] 2020-03-30 lu
nes
$./firstlastprivate
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
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer4] 2020-03-30 lu
nes
$./firstlastprivate
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 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 3 suma a[6] suma=6

Fuera de la construcción parallel suma=6
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer4] 2020-03-30 lu
nes
$./firstlastprivate
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
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer4] 2020-03-30 lu
nes
$
```

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: Se observa que el valor que hemos dado a ‘a’ se copia solo en una de las posiciones del Array b. Esto se debe a que al ejecutar la directiva for, las iteraciones del bucle, que son 8, se dividen entre las ocho cpus de mi PC, de forma que sólo adopta el valor de ‘a’ introducido, aquella posición del vector cuyo valor lo establece la hebra que ejecuta la cláusula single, mientras que para el resto de hebras ‘a’ no está inicializado y por tanto se establece a=0.

   CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```c
#include <stdio.h>
#include <omp.h>

main() {

int n = 9, i, b[n];

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


#pragma omp parallel
{ int a;

        #pragma omp single
        {
        printf("\nIntroduce valor de inicialización a:");
        scanf("%d",&a);
        printf("\nSingle ejecutada por el thread %d\n", omp_get_thread_num());
        }

        #pragma omp for
        for(i=0; i < n;i++)     b[i]=a;
}

printf("Después de la región parallel:\n");
for(i=0; i < n;i++) printf("b[%d] = %d\t", i,b[i]);
printf("\n");

}
```

CAPTURAS DE PANTALLA:

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer5] 2020-03-30 lu
nes
$gcc -O2 -fopenmp -o copyprivate-clause copyprivate-clause.c
copyprivate-clause.c:4:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main() {
 ^~~~
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer5] 2020-03-30 lu
nes
$./copyprivate-clause

Introduce valor de inicialización a:3

Single ejecutada por el thread 4
Después de la región parallel:
b[0] = 0        b[1] = 0        b[2] = 0        b[3] = 0        b[4] = 0        b[5] = 3        b[6] = 0b
[7] = 0 b[8] = 0
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer5] 2020-03-30 lu
nes
$./copyprivate-clause

Introduce valor de inicialización a:10

Single ejecutada por el thread 3
Después de la región parallel:
b[0] = 0        b[1] = 0        b[2] = 0        b[3] = 0        b[4] = 10       b[5] = 0        b[6] = 0b
[7] = 0 b[8] = 0
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer5] 2020-03-30 lu
nes
$./copyprivate-clause

Introduce valor de inicialización a:5

Single ejecutada por el thread 1
Después de la región parallel:
b[0] = 0        b[1] = 0        b[2] = 5        b[3] = 0        b[4] = 0        b[5] = 0        b[6] = 0b
[7] = 0 b[8] = 0
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer5] 2020-03-30 lu
nes
$
```

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: El resultado que se imprime ahora es el que se esperaría para suma=0, pero sumándole 10. Esto se debe a que la suma final que se obtiene es suma=suma+suma_0+suma_1+...+suma_n, y como en este caso el valor inicial de suma es 10 y no 0, la suma será 10 unidades mayor de lo esperado.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

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

main(int argc, char **argv){

int i, n=20, a[n], suma=10;

if(argc < 2){
        fprintf(stderr,"Falta iteraciones\n");
        exit(-1);
}

n=atoi(argv[1]);

if(n>20){

n=20; printf("n=%d",n);

}

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

#pragma omp parallel for reduction (+:suma)
for(i=0; i < n; i++)    suma += a[i];

printf("Tras 'parallel' suma=%d\n", suma);

}
```

CAPTURAS DE PANTALLA:

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer6] 2020-03-30 lu
nes
$gcc -O2 -fopenmp -o reduction-clause reduction-clause.c
reduction-clause.c:9:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main(int argc, char **argv){
 ^~~~
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer6] 2020-03-30 lu
nes
$./reduction-clause 10
Tras 'parallel' suma=55
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer6] 2020-03-30 lu
nes
$./reduction-clause 20
Tras 'parallel' suma=200
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer6] 2020-03-30 lu
nes
$./reduction-clause 6
Tras 'parallel' suma=25
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer6] 2020-03-30 lu
nes
$
```

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

RESPUESTA: Divido la directiva combinada parallel for, en una directiva parallel que contiene a la directiva for, y una directiva atomic. Además a la directiva parallel le añado la cláusula private a una variable auxiliar sumalocal, que contendrá el valor de la suma de la iteraciones realizada por cada hebra. Finalmente la directiva atomic se encarga de realizar una a una las suma de las variables privada sumalocal dentro de la variable suma, dando el resultado.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`

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

main(int argc, char **argv){

int i, n=20, a[n], suma=0, sumalocal;

if(argc < 2){
        fprintf(stderr,"Falta iteraciones\n");
        exit(-1);
}

n=atoi(argv[1]);

if(n>20){

n=20; printf("n=%d",n);

}

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



#pragma omp parallel private(sumalocal)
{
sumalocal=0;

#pragma omp for
for(i=0; i < n; i++)    sumalocal+=a[i];

#pragma omp atomic
suma+=sumalocal;
}

printf("Tras 'parallel' suma=%d\n", suma);

}
```

CAPTURAS DE PANTALLA:

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer7] 2020-03-30 lu
nes
$gcc -O2 -fopenmp -o reduction-clause reduction-clause.c
reduction-clause.c:9:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main(int argc, char **argv){
 ^~~~
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer7] 2020-03-30 lu
nes
$./reduction-clause 5
Tras 'parallel' suma=10
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer7] 2020-03-30 lu
nes
$./reduction-clause 6
Tras 'parallel' suma=15
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer7] 2020-03-30 lu
nes
$./reduction-clause 10
Tras 'parallel' suma=45
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer7] 2020-03-30 lu
nes
$./reduction-clause 20
Tras 'parallel' suma=190
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer7] 2020-03-30 lu
nes
$
```

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):

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

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`

```c
#include <stdlib.h>
#include <stdio.h>

//#define VECTOR_GLOBAL

#define VECTOR_DYNAMIC

#ifdef VECTOR_GLOBAL
#define MAX 15000

double v[MAX], final[MAX], matriz[MAX][MAX];
 //Inicializar vector
for(i=0; i < N; i++)
        v[i]=rand()%20;

if(N < 11){
 //Imprimimos matriz
for(i=0; i < N; i++){
        for(j=0; j < N; j++)
                printf("m[%d%d]=%f\t",i,j,matriz[i][j]);
printf("\n");
}

 //Imprimimos vector
for(i=0; i < N; i++)
        printf("v[%d]=%f\n",i,v[i]);
}

else{
printf("m[%d%d]=%f\t",0,0,matriz[0][0]);
printf("m[%d%d]=%f\t",N-1,N-1,matriz[N-1][N-1]);
printf("\nv[%d]=%f\n",0,v[0]);
printf("v[%d]=%f\n",N-1,v[N-1]);
}

clock_gettime(CLOCK_REALTIME,&cgt1);
//Producto matriz*vector
for(i=0; i < N; i++){
        for(j=0; j < N; j++)
                final[i]+=matriz[i][j]*v[j];
}

clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

printf("Tiempo: %f\nTamaño: %d\n\n",ncgt, N);

if(N < 11){
//Imprimimos resultado
printf("\n RESULTADO PRODUCTO \n");
for(i=0; i < N; i++)
        printf("final[%d]=%f\n ", i, final[i]);
}

else{
        printf("\n RESULTADO PRODUCTO \n");
        printf("final[0]=%f\n",final[0]);
        printf("final[N-1]=%f\n",final[N-1]);
}
}
```

CAPTURAS DE PANTALLA:

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer8] 2020-04-01 miércoles
$gcc -O2 -o pmv-secuencial pmv-secuencial.c
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer8] 2020-04-01 miércoles
$./pmv-secuencial 8
m[00]=19.000000 m[01]=18.000000 m[02]=8.000000  m[03]=2.000000  m[04]=9.000000  m[05]=15.000000 m[06]=19.000000 m[07]=14.000000
m[10]=0.000000  m[11]=8.000000  m[12]=13.000000 m[13]=18.000000 m[14]=5.000000  m[15]=1.000000  m[16]=2.000000  m[17]=0.000000
m[20]=11.000000 m[21]=8.000000  m[22]=11.000000 m[23]=2.000000  m[24]=3.000000  m[25]=9.000000  m[26]=7.000000  m[27]=16.000000
m[30]=9.000000  m[31]=16.000000 m[32]=17.000000 m[33]=5.000000  m[34]=16.000000 m[35]=14.000000 m[36]=18.000000 m[37]=8.000000
m[40]=12.000000 m[41]=6.000000  m[42]=10.000000 m[43]=13.000000 m[44]=13.000000 m[45]=9.000000  m[46]=19.000000 m[47]=14.000000
m[50]=9.000000  m[51]=13.000000 m[52]=4.000000  m[53]=6.000000  m[54]=6.000000  m[55]=6.000000  m[56]=6.000000  m[57]=17.000000
m[60]=6.000000  m[61]=17.000000 m[62]=12.000000 m[63]=9.000000  m[64]=19.000000 m[65]=19.000000 m[66]=18.000000 m[67]=8.000000
m[70]=15.000000 m[71]=15.000000 m[72]=5.000000  m[73]=3.000000  m[74]=9.000000  m[75]=3.000000  m[76]=11.000000 m[77]=13.000000
v[0]=1.000000
v[1]=1.000000
v[2]=7.000000
v[3]=15.000000
v[4]=3.000000
v[5]=6.000000
v[6]=1.000000
v[7]=4.000000
Tiempo: 0.000000
Tamaño: 8


 RESULTADO PRODUCTO
final[0]=315.000000
 final[1]=392.000000
 final[2]=260.000000
 final[3]=401.000000
 final[4]=451.000000
 final[5]=268.000000
 final[6]=463.000000
 final[7]=218.000000
 [DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer8] 2020-04-01 miércoles
$./pmv-secuencial 11
m[00]=9.000000  m[1010]=16.000000
v[0]=7.000000
v[10]=19.000000
Tiempo: 0.000001
Tamaño: 11


 RESULTADO PRODUCTO
final[0]=1524.000000
final[N-1]=1067.000000
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer8] 2020-04-01 miércoles
$
```

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`

```
clock_gettime(CLOCK_REALTIME,&cgt1);
//Producto matriz*vector
#pragma omp parallel for private(i,j)
for(i=0; i < N; i++){
        for(j=0; j < N; j++){
                final[i]+=matriz[i][j]*v[j];
                //printf("Producto a[%d][%d] hebra:%d\n",i,j,omp_get_thread_num());
                //printf("final[%d]+=%f*%f=%f\n",i,matriz[i][j],v[j],final[i]);
        }
}
clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

printf("Tiempo: %f\nTamaño: %d\n\n",ncgt, N);

if(N < 11){
//Imprimimos resultado
printf("\n RESULTADO PRODUCTO \n");
for(i=0; i < N; i++)
        printf("final[%d]=%f\n ", i, final[i]);
}

else{
        printf("\n RESULTADO PRODUCTO \n");
        printf("final[0]=%f\n",final[0]);
        printf("final[N-1]=%f\n",final[N-1]);
}
```

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

```
clock_gettime(CLOCK_REALTIME,&cgt1);
//Producto matriz*vector
#pragma omp parallel private(productolocal,i,j)
{

for(i=0; i < N; i++){
        productolocal=0;

        #pragma omp for
        for(j=0; j < N; j++){
                productolocal+=matriz[i][j]*v[j];
                printf("Producto a[%d][%d] hebra:%d\n",i,j,omp_get_thread_num());
        }

        #pragma omp critical
        final[i]+=productolocal;
}
}
clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

printf("Tiempo: %f\nTamaño: %d\n\n",ncgt, N);

if(N < 11){
//Imprimimos resultado
printf("\n RESULTADO PRODUCTO \n");
for(i=0; i < N; i++)
        printf("final[%d]=%f\n ", i, final[i]);
}

else{
        printf("\n RESULTADO PRODUCTO \n");
        printf("final[0]=%f\n",final[0]);
        printf("final[N-1]=%f\n",final[N-1]);
}
```

RESPUESTA: En errores de ejecución únicamente daba error por un error de sintaxis. En cuanto a errores de ejecución, en primer lugar en ambas versiones me daba error al realizar la suma por no declarar j e i como variables privadas a cada hebra. Otro error en tiempo de ejecución ha sido que al no haber inicializado los valores del vector final, la suma daba errónea. Para la versión por columnas otro error de compilación era que al haber hecho el productolocal=0 fuera del bucle for se acumulaban los valores, dando valores muy grandes. Para solucionarlo he hecho que dicha variable se iguale a 0 al principio de cada iteración del primer bucle.

CAPTURAS DE PANTALLA:

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer9] 2020-04-01 miércoles
$gcc -O2 -fopenmp -o pmv-secuencial-filas pmv-secuencial-filas.c
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer9] 2020-04-01 miércoles
$./pmv-secuencial-filas 8
m[00]=0.000000  m[01]=17.000000 m[02]=1.000000  m[03]=1.000000  m[04]=15.000000 m[05]=4.000000  m[06]=11.000000 m[07]=15.000000
m[10]=15.000000 m[11]=10.000000 m[12]=0.000000  m[13]=4.000000  m[14]=11.000000 m[15]=13.000000 m[16]=6.000000  m[17]=8.000000
m[20]=15.000000 m[21]=9.000000  m[22]=1.000000  m[23]=4.000000  m[24]=5.000000  m[25]=6.000000  m[26]=11.000000 m[27]=15.000000
m[30]=6.000000  m[31]=12.000000 m[32]=3.000000  m[33]=19.000000 m[34]=19.000000 m[35]=11.000000 m[36]=5.000000  m[37]=11.000000
m[40]=8.000000  m[41]=6.000000  m[42]=13.000000 m[43]=15.000000 m[44]=2.000000  m[45]=16.000000 m[46]=2.000000  m[47]=17.000000
m[50]=18.000000 m[51]=2.000000  m[52]=14.000000 m[53]=10.000000 m[54]=16.000000 m[55]=0.000000  m[56]=18.000000 m[57]=11.000000
m[60]=10.000000 m[61]=11.000000 m[62]=8.000000  m[63]=7.000000  m[64]=9.000000  m[65]=19.000000 m[66]=2.000000  m[67]=15.000000
m[70]=3.000000  m[71]=17.000000 m[72]=6.000000  m[73]=14.000000 m[74]=9.000000  m[75]=11.000000 m[76]=6.000000  m[77]=9.000000
v[0]=9.000000
v[1]=11.000000
v[2]=17.000000
v[3]=11.000000
v[4]=7.000000
v[5]=19.000000
v[6]=1.000000
v[7]=6.000000
Tiempo: 0.005121
Tamaño: 8


 RESULTADO PRODUCTO
final[0]=497.000000
 final[1]=667.000000
 final[2]=545.000000
 final[3]=859.000000
 final[4]=946.000000
 final[5]=728.000000
 final[6]=940.000000
 final[7]=802.000000
 [DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer9] 2020-04-01 miércoles
$./pmv-secuencial-filas 11
m[00]=14.000000 m[1010]=0.000000
v[0]=8.000000
v[10]=3.000000
Tiempo: 0.000703
Tamaño: 11


 RESULTADO PRODUCTO
final[0]=742.000000
final[N-1]=762.000000
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer9] 2020-04-01 miércoles
$
```

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer9] 2020-04-01 miércoles
$gcc -O2 -fopenmp -o pmv-secuencial_columnas pmv-secuencial_columnas.c
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer9] 2020-04-01 miércoles
$./pmv-secuencial_columnas 8
m[00]=12.000000 m[01]=7.000000  m[02]=11.000000 m[03]=7.000000  m[04]=2.000000  m[05]=1.000000  m[06]=3.000000  m[07]=8.000000
m[10]=3.000000  m[11]=6.000000  m[12]=4.000000  m[13]=5.000000  m[14]=1.000000  m[15]=11.000000 m[16]=4.000000  m[17]=10.000000
m[20]=4.000000  m[21]=11.000000 m[22]=19.000000 m[23]=2.000000  m[24]=0.000000  m[25]=0.000000  m[26]=7.000000  m[27]=6.000000
m[30]=3.000000  m[31]=19.000000 m[32]=11.000000 m[33]=14.000000 m[34]=18.000000 m[35]=19.000000 m[36]=15.000000 m[37]=3.000000
m[40]=18.000000 m[41]=6.000000  m[42]=10.000000 m[43]=12.000000 m[44]=7.000000  m[45]=13.000000 m[46]=0.000000  m[47]=11.000000
m[50]=11.000000 m[51]=17.000000 m[52]=8.000000  m[53]=13.000000 m[54]=8.000000  m[55]=4.000000  m[56]=15.000000 m[57]=4.000000
m[60]=15.000000 m[61]=7.000000  m[62]=18.000000 m[63]=8.000000  m[64]=19.000000 m[65]=18.000000 m[66]=14.000000 m[67]=2.000000
m[70]=17.000000 m[71]=5.000000  m[72]=9.000000  m[73]=8.000000  m[74]=16.000000 m[75]=4.000000  m[76]=11.000000 m[77]=15.000000
v[0]=11.000000
v[1]=13.000000
v[2]=7.000000
v[3]=10.000000
v[4]=18.000000
v[5]=0.000000
v[6]=13.000000
v[7]=9.000000
Tiempo: 0.002827
Tamaño: 8


 RESULTADO PRODUCTO
final[0]=517.000000
 final[1]=349.000000
 final[2]=485.000000
 final[3]=1043.000000
 final[4]=691.000000
 final[5]=903.000000
 final[6]=1004.000000
 final[7]=961.000000
 [DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer9] 2020-04-01 miércoles
$./pmv-secuencial_columnas 11
m[00]=8.000000  m[1010]=7.000000
v[0]=14.000000
v[10]=1.000000
Tiempo: 0.002064
Tamaño: 11


 RESULTADO PRODUCTO
final[0]=946.000000
final[N-1]=707.000000
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer9] 2020-04-01 miércoles
$
```

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-OpenmMP-reduction.c`

```
clock_gettime(CLOCK_REALTIME,&cgt1);
//Producto matriz*vector
#pragma omp parallel private(productolocal,i,j)
{

for(i=0; i < N; i++){
        productolocal=0;

        #pragma omp for reduction(+:final[i])
        for(j=0; j < N; j++){
                final[i]+=matriz[i][j]*v[j];

        }
}
}

clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

printf("Tiempo: %f\nTamaño: %d\n\n",ncgt, N);
```

RESPUESTA: No ha habido ningún error, el cambio que había que realizar era muy pequeño.

CAPTURAS DE PANTALLA:

```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer10] 2020-04-05 domingo
$gcc -O2 -fopenmp -o pmv-secuencial_columnas pmv-secuencial_columnas.c
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer10] 2020-04-05 domingo
$./pmv-secuencial_columnas 8
m[00]=14.000000 m[01]=12.000000 m[02]=18.000000 m[03]=14.000000 m[04]=19.000000 m[05]=4.000000  m[06]=14.000000 m[07]=18.000000
m[10]=0.000000  m[11]=9.000000  m[12]=8.000000  m[13]=16.000000 m[14]=3.000000  m[15]=7.000000  m[16]=13.000000 m[17]=4.000000
m[20]=9.000000  m[21]=17.000000 m[22]=6.000000  m[23]=14.000000 m[24]=5.000000  m[25]=19.000000 m[26]=19.000000 m[27]=1.000000
m[30]=4.000000  m[31]=0.000000  m[32]=5.000000  m[33]=18.000000 m[34]=5.000000  m[35]=18.000000 m[36]=11.000000 m[37]=0.000000
m[40]=3.000000  m[41]=9.000000  m[42]=6.000000  m[43]=2.000000  m[44]=6.000000  m[45]=0.000000  m[46]=12.000000 m[47]=18.000000
m[50]=2.000000  m[51]=0.000000  m[52]=6.000000  m[53]=5.000000  m[54]=19.000000 m[55]=0.000000  m[56]=1.000000  m[57]=9.000000
m[60]=17.000000 m[61]=0.000000  m[62]=15.000000 m[63]=14.000000 m[64]=19.000000 m[65]=14.000000 m[66]=16.000000 m[67]=3.000000
m[70]=6.000000  m[71]=1.000000  m[72]=13.000000 m[73]=12.000000 m[74]=12.000000 m[75]=5.000000  m[76]=4.000000  m[77]=15.000000
v[0]=6.000000
v[1]=10.000000
v[2]=17.000000
v[3]=4.000000
v[4]=2.000000
v[5]=1.000000
v[6]=2.000000
v[7]=4.000000
Tiempo: 0.007073
Tamaño: 8


 RESULTADO PRODUCTO
final[0]=708.000000
 final[1]=345.000000
 final[2]=453.000000
 final[3]=231.000000
 final[4]=326.000000
 final[5]=210.000000
 final[6]=509.000000
 final[7]=412.000000
 [DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer10] 2020-04-05 domingo
$./pmv-secuencial_columnas 11
m[00]=7.000000  m[1010]=8.000000
v[0]=4.000000
v[10]=12.000000
Tiempo: 0.002054
Tamaño: 11


 RESULTADO PRODUCTO
final[0]=687.000000
final[N-1]=1233.000000
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2/ejer10] 2020-04-05 domingo
$
```

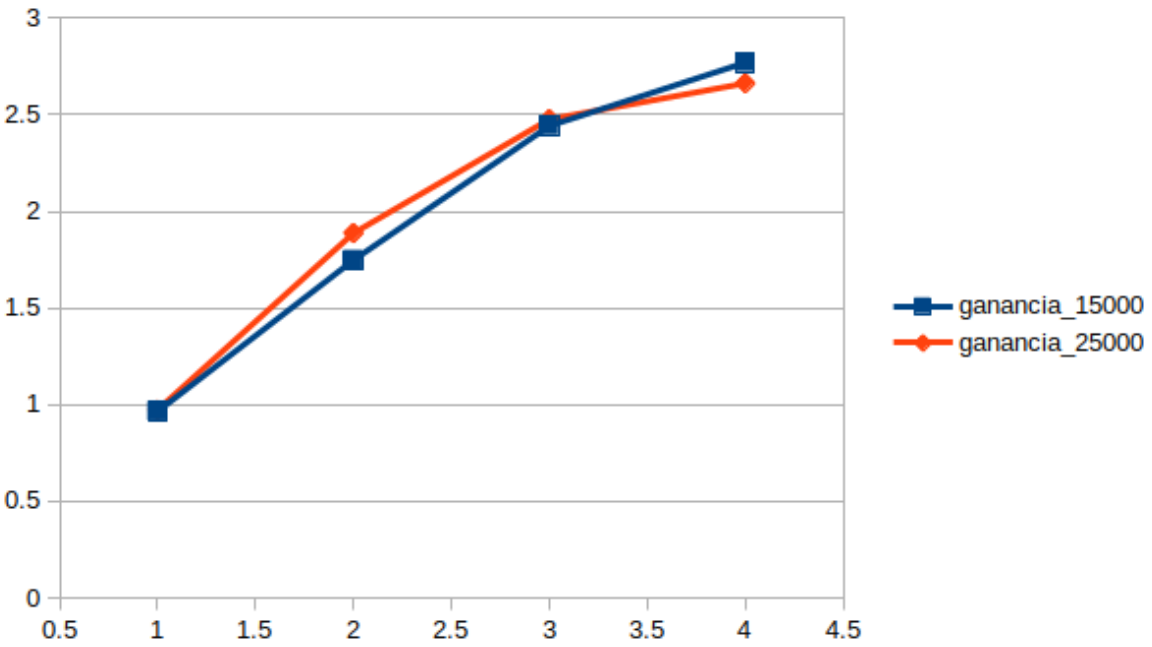
11. Ayudándose de una hoja de cálculo (recuerde que en las aulas está instalado OpenOffice) realice una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del mejor código paralelo de los tres implementados en los ejercicios 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):

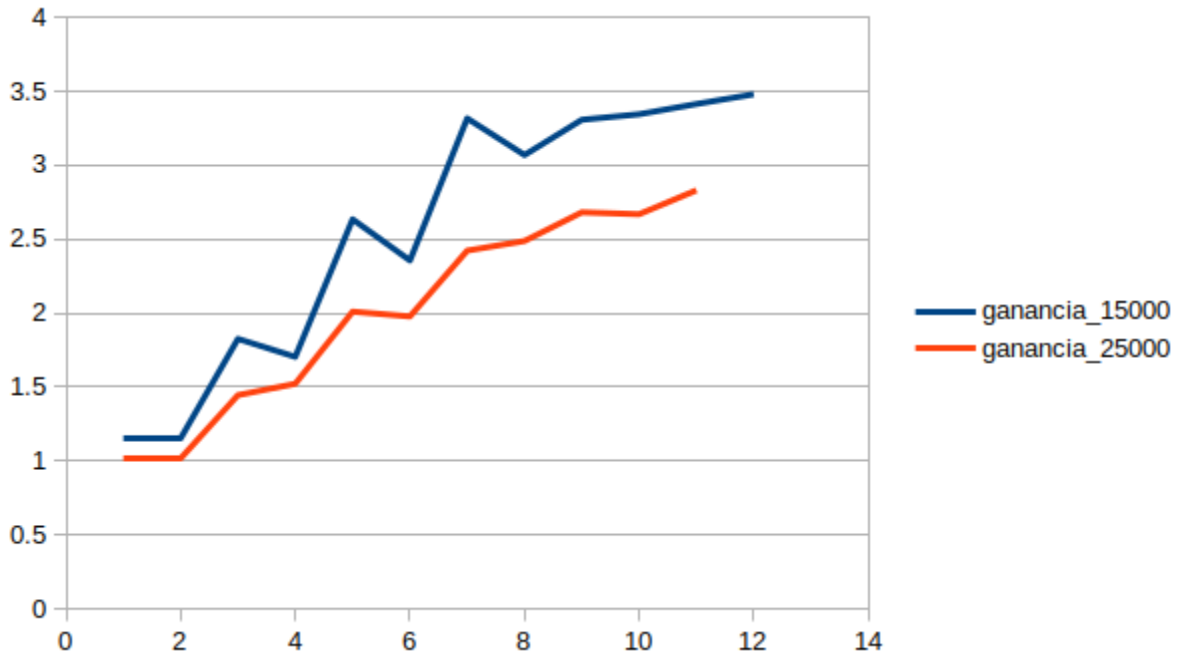
```
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2] 2020-04-05 domingo
$./ejer8/pmv-secuencial 15000
m[00]=10.000000 m[1499914999]=14.000000
v[0]=6.000000
v[14999]=12.000000
Tiempo: 0.256326
Tamaño: 15000


 RESULTADO PRODUCTO
final[0]=1364323.000000
final[N-1]=1362082.000000
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2] 2020-04-05 domingo
$./ejer9/pmv-secuencial-filas 15000
m[00]=12.000000 m[1499914999]=3.000000
v[0]=3.000000
v[14999]=10.000000
Tiempo: 0.104301
Tamaño: 15000


 RESULTADO PRODUCTO
final[0]=1380256.000000
final[N-1]=1364938.000000
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2] 2020-04-05 domingo
$./ejer10/pmv-secuencial_columnas 15000
m[00]=19.000000 m[1499914999]=8.000000
v[0]=1.000000
v[14999]=5.000000
Tiempo: 0.115612
Tamaño: 15000


 RESULTADO PRODUCTO
final[0]=1359524.000000
final[N-1]=1366399.000000
[DanielMonjasMiguelez daniel@daniel-XPS-15-9570:~/Escritorio/Daniel/AC/PRACTICAS/bp2] 2020-04-05 domingo
$
```

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):

MI PC					
		Threads			
Tamaños	Secuencial	1	2	3	4
15000	0.257587	0.267089	0.147354	0.105471	0.093014
25000	0.719276	0.739497	0.381083	0.290407	0.270153
Ganancia	Threads				
Tamaños	1	2	3	4	
15000	0.964423844	1.748082848	2.442254269	2.769335799	
25000	0.972655738	1.88745234	2.476786028	2.662476449	



ATCGRID													
		THREADS											
Tamaños	Secuencial	1	2	3	4	5	6	7	8	9	10	11	12
15000	0.500841	0.43551	0.436654	0.274258	0.293839	0.190028	0.212507	0.150946	0.163123	0.151374	0.149676	0.146634	0.143841
25000	1.085901	1.073805	1.072537	0.751172	0.713311	0.539975	0.548839	0.448041	0.436305	0.404784	0.406736	0.383461	0.39251
Ganancia		THREADS											
Tamaños	1	2	3	4	5	6	7	8	9	10	11	12	
15000	1.150010333	1.146997394	1.826167331	1.704474219	2.635616856	2.356821187	3.318014389	3.070327299	3.308632922	3.346167722	3.415585744	3.481907106	
25000	1.011264615	1.012460176	1.445608995	1.522338784	2.011020881	1.978541977	2.423664352	2.488857565	2.682667793	2.669793183	2.831842091	2.766556266	



COMENTARIOS SOBRE LOS RESULTADOS: En primer lugar, he elegido el programa de productos de vector por matriz que paraleliza el bucle de las filas, pues al probar los tres para un tamaño grande me sale que es el que menos tiempo tarda. En mi ordenador se puede observar que el tiempo de ejecución se reduce significantemente al aumentar el número de cores, pero sin embargo, al pasar de tres a cuatro cores la mejora es

muy pequeña, esto se deberá a que ya se ha obtenido toda la optimización posible con paralelismo y por consiguiente aunque añadamos más cores a la ejecución de dicha parte se tarda más tiempo en invocar las directivas de sincronización que en la propia ejecución. Por otra parte en atcgrid, se observa una mejora similar a la de mi ordenador pero de forma más escalonada. Esto se puede deber a que al disponer de dos procesadores físicos (sockets) la comunicación y sincronización entre ambos conlleva que al aumentar el número de cores no se obtenga mejora pues requieres de cores del otro procesador y la comunicación requiere más tiempo del que se optimiza la ejecución.