

Grai2º curso / 2º  
cuatr.  
Grado Ing. Inform.

# Arquitectura de Computadores (AC)

## Cuaderno de prácticas.

## Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Félix Ramírez García

Grupo A3 con profesor Christian Morillas

Fecha de entrega: 28-04-2019

Fecha evaluación en clase: 29-04-2019

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)

La clausula `default(none)` hace que se tengan que especificar los ámbitos de todas las variables involucradas en la región paralela. En este caso he indicado que la variable `a` y `n` son compartidas y la variable `i` es privada para la zona paralela.

La siguiente imagen muestra el programa `shared-clauseModificado.c` :

```
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGar
ciaFelix/ejer1] 2019-04-20 Saturday
$ls
shared-clauseModificado.c
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGar
ciaFelix/ejer1] 2019-04-20 Saturday
$cat shared-clauseModificado.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 shared(a, n) private(i) default(none)
    for(i=0;i<n;i++) a[i] +=i;

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

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

La siguiente imagen muestra lo que ocurre al compilar el programa , que es un warning al no especificar que la función `main` devuelve un tipo entero , una vez solventado esto he vuelto a compilar el programa y no da ningún tipo de error ni aviso, por lo que procedo a su ejecución:

```
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGar
ciaFelix/ejer1] 2019-04-20 Saturday
$gcc -O2 -fopenmp -o shared-clauseModificado shared-clauseModificado.c
shared-clauseModificado.c:6:1: warning: return type defaults to 'int' [-Wimplicit-int]
main() {
~~~~~
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGar
ciaFelix/ejer1] 2019-04-20 Saturday
$gcc -O2 -fopenmp -o shared-clauseModificado shared-clauseModificado.c
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGar
ciaFelix/ejer1] 2019-04-20 Saturday
$ ./shared-clauseModificado
Despues de parallel for:
a[0]=1
a[1]=3
a[2]=5
a[3]=7
a[4]=9
a[5]=11
a[6]=13
```

Si compilamos el programa sin especificar los ámbitos de las variables obtendríamos los errores comentados anteriormente que se ilustran en la siguiente imagen:

```
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGar
ciaFelix/ejer1] 2019-04-20 Saturday
$gcc -O2 -fopenmp -o shared-clause shared-clause.c
shared-clause.c: In function 'main':
shared-clause.c:14:10: error: 'n' not specified in enclosing 'parallel'
#pragma omp parallel for shared(a) default(none)
               ^
shared-clause.c:14:10: error: enclosing 'parallel'
shared-clause.c:14:10: error: enclosing 'parallel'
```

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.

El código de `private-clauseModificado.c` se muestra en la siguiente imagen:

```
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=3;
11
12 |     for(i=0;i<n;i++){
13 |         a[i]=i;
14 |     }
15
16 |     #pragma omp parallel private (suma)
17 |     {
18 |         suma=4;
19 |         #pragma omp for
20 |         for(i=0;i<n;i++){
21 |             suma=suma+a[i];
22 |             printf("thread %d suma a[%d]/", omp_get_thread_num(), i);
23 |             printf("\n*thread %d suma=%d", omp_get_thread_num(), suma);
24 |         }
25 |     }
26 |     printf("\n");
27
28 |     return 0;
29 }
30
```

Si la variable se inicializa fuera, al llegar a la región paralela la variable contendrá basura. En el caso de que iniciemos la variable tanto dentro como fuera de la región paralela, la variable se inicializará con el valor con el que se inicializó dentro de la región paralela

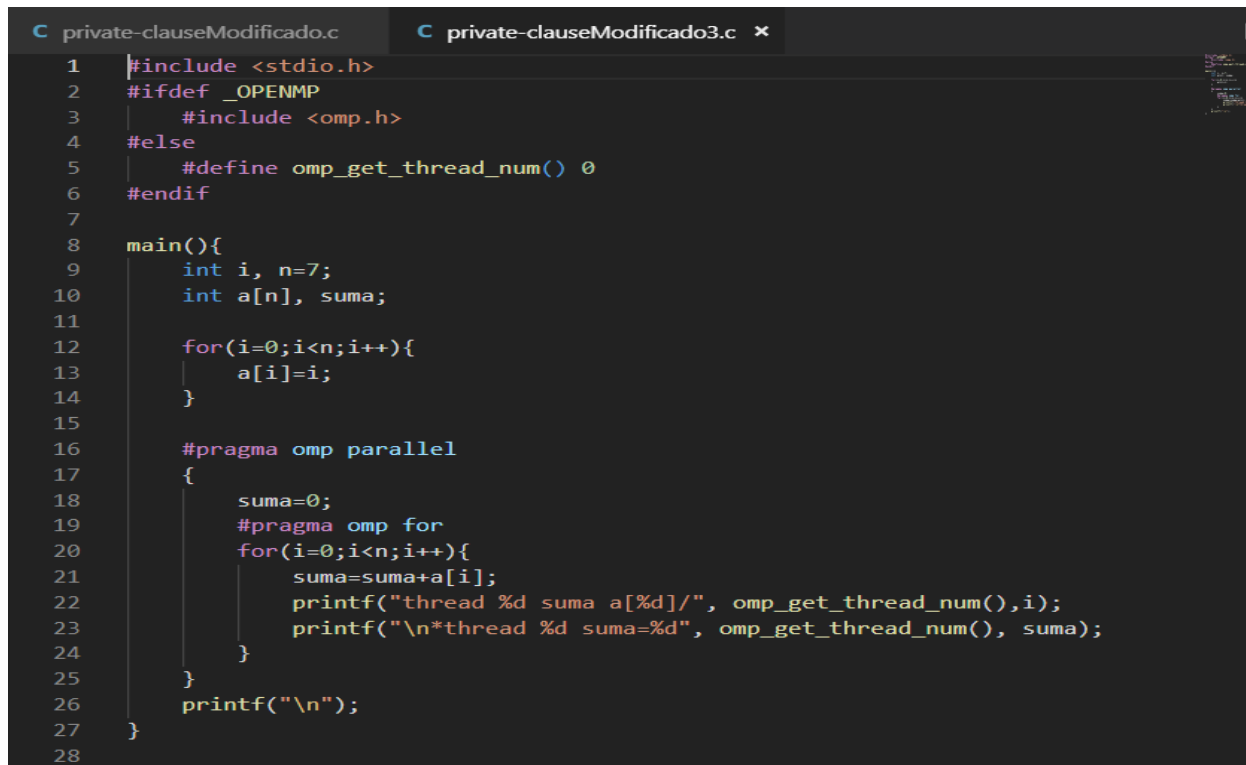
A continuación se muestra la captura de pantalla de la ejecución de `private-clauseModificado`:

```
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGarciaFelix/ejer2] 2019-04-28 Sunday
$gcc -O2 -fopenmp -o private-clauseModificado private-clauseModificado.c
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGarciaFelix/ejer2] 2019-04-28 Sunday
$./private-clauseModificado
thread 0 suma a[0]/
*thread 0 suma=4thread 0 suma a[1]/
*thread 0 suma=5thread 2 suma a[3]/
*thread 2 suma=7thread 3 suma a[4]/
*thread 3 suma=8thread 5 suma a[6]/
*thread 5 suma=10thread 4 suma a[5]/
*thread 4 suma=9thread 1 suma a[2]/
*thread 1 suma=6
```

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

Lo que ocurre es que la variable pasa a ser compartida y todas las hebras podrán modificarla, los valores irían machacándose unos a otros .

El código de `private-clauseModificado3.c` es el que muestra la siguiente imagen:



```
C private-clauseModificado.c  C private-clauseModificado3.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  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
16     #pragma omp parallel
17     {
18         suma=0;
19         #pragma omp for
20         for(i=0;i<n;i++){
21             suma=suma+a[i];
22             printf("thread %d suma a[%d]/", omp_get_thread_num(),i);
23             printf("\n*thread %d suma=%d", omp_get_thread_num(), suma);
24         }
25     }
26     printf("\n");
27 }
28
```

El resultado de la ejecución de este apartado es el siguiente:

```
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGar
ciaFelix/ejer3] 2019-04-28 Sunday
$gcc -O2 -fopenmp -o private-clauseModificado3 private-clauseModificado3.c
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGar
ciaFelix/ejer3] 2019-04-28 Sunday
$./private-clauseModificado3
thread 0 suma a[0]/
*thread 0 suma=4thread 0 suma a[1]/
*thread 0 suma=5thread 1 suma a[2]/
*thread 1 suma=5thread 3 suma a[4]/
*thread 3 suma=5thread 4 suma a[5]/
*thread 4 suma=5thread 2 suma a[3]/
*thread 2 suma=5thread 5 suma a[6]/
*thread 5 suma=5
```

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).

Eso es porque `lastprivate` asigna a la variable el ultimo valor que se tendría y en una ejecución secuencial  $(0+6)$  siempre será 6.

El código es el siguiente:

```
C firstlastprivate-clause.c x
1  #include <stdio.h>
2  #include <omp.h>
3
4  main(){
5      int i, n=7;
6      int a[n], suma=0;
7
8      for(i=0;i<n;i++) a[i]=i;
9
10     #pragma omp parallel for firstprivate(suma) lastprivate(suma)
11     for(i=0;i<n;i++){
12         suma=suma+a[i];
13         printf("thread %d suma a[%d] suma=%d\n", omp_get_thread_num(), i, suma);
14     }
15
16     printf("\nFuera de la construccion parallel suma=%d\n", suma);
17 }
18
```

Y el resultado de su ejecución se muestra en la siguiente imagen:

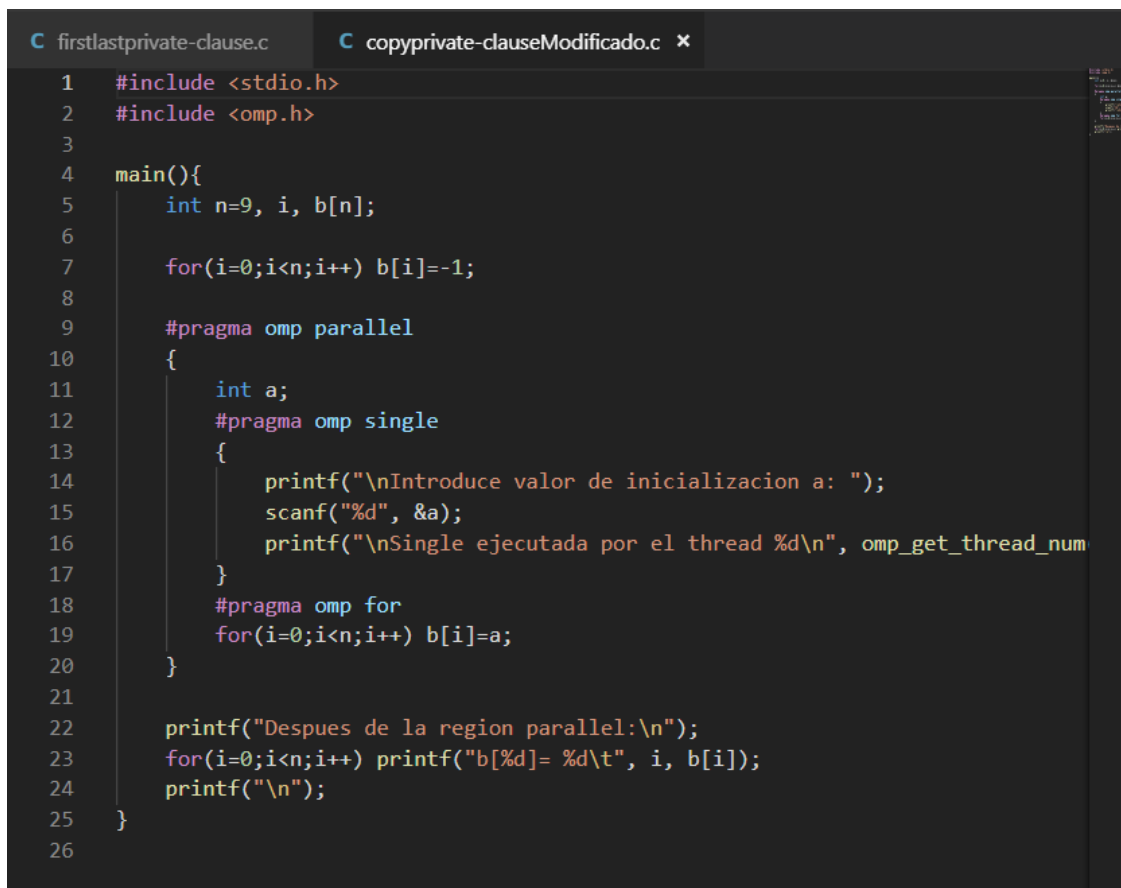
```
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGar
ciaFelix/ejer4] 2019-04-28 Sunday
$gcc -O2 -fopenmp -o firstlastprivate-clause firstlastprivate-clause.c
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGar
ciaFelix/ejer4] 2019-04-28 Sunday
$ ./firstlastprivate-clause
thread 5 suma a[6] suma=6
thread 3 suma a[4] suma=4
thread 1 suma a[2] suma=2
thread 2 suma a[3] suma=3
thread 4 suma a[5] suma=5
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1

Fuera de la construccion parallel suma=6
```

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)

Lo que ocurre es que solo un thread tiene la variable bien inicializada y los demás tienen basura. Al no tener la cláusula `copyprivate`, al terminar el `single`, no se copia el valor de la variable a los otros threads y por eso la variable no está inicializada en todos. En las capturas de pantalla podemos observar el cambio de resultado de ambas ejecuciones.

El código de `copyprivate-clauseModificado.c` es el siguiente:



```
C firstlastprivate-clause.c  C copyprivate-clauseModificado.c x
1  #include <stdio.h>
2  #include <omp.h>
3
4  main(){
5      int n=9, i, b[n];
6
7      for(i=0;i<n;i++) b[i]=-1;
8
9      #pragma omp parallel
10     {
11         int a;
12         #pragma omp single
13         {
14             printf("\nIntroduce valor de inicializacion a: ");
15             scanf("%d", &a);
16             printf("\nSingle ejecutada por el thread %d\n", omp_get_thread_num);
17         }
18         #pragma omp for
19         for(i=0;i<n;i++) b[i]=a;
20     }
21
22     printf("Despues de la region parallel:\n");
23     for(i=0;i<n;i++) printf("b[%d]= %d\t", i, b[i]);
24     printf("\n");
25 }
26
```

La ejecución de `copyprivate-clause.c` es la siguiente:

```

Codigo.FRM copyprivate-clause.c copyprivate-clauseModificado.c
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGar
ciaFelix/ejer5] 2019-04-28 Sunday
$gcc -O2 -fopenmp -o copyprivate-clause copyprivate-clause.c
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGar
ciaFelix/ejer5] 2019-04-28 Sunday
$./copyprivate-clause

Introduce valor de inicialización a: 3

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

```

Y la ejecución de `copyprivate-clauseModificado.c` es la siguiente:

```

= 3      b[8] = 3
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGar
ciaFelix/ejer5] 2019-04-28 Sunday
$gcc -O2 -fopenmp -o copyprivate-clauseModificado copyprivate-clauseModificado.c
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGar
ciaFelix/ejer5] 2019-04-28 Sunday
$./copyprivate-clauseModificado

Introduce valor de inicializacion a: 3

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

```

- 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)

El programa suma los números desde 0 hasta `n`, teniendo `n` como máximo el 20, si la variable en vez de estar inicializada a 0 lo esta a 10, el resultado será el mismo que el anterior pero sumándole 10.

El código es el que se muestra en la siguiente imagen:

```
#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;

    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 'paralel' suma=%d\n", suma);
}
```

Ejecución de reduction-clipse.c con parámetro 10.

```
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/m
ciaFelix/ejer6] 2019-04-28 Sunday
$gcc -O2 -fopenmp -o reduction-clipse reduct
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/m
ciaFelix/ejer6] 2019-04-28 Sunday
$./reduction-clipse 10
Tras 'paralel' suma=45
```

Ejecución de reduction-clipseModificado.c con parámetro 10

```
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/m
ciaFelix/ejer6] 2019-04-28 Sunday
$gcc -O2 -fopenmp -o reduction-clipseModif
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/m
ciaFelix/ejer6] 2019-04-28 Sunday
$./reduction-clipseModificado 10
Tras 'paralel' suma=45
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/m
```

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.

Debemos encontrar una forma de dividir el trabajo entre los threads. He optado por poner el número de hebras a `n`. Y que cada hebra, dentro de una región `parallel` vayan sumando dentro de la misma variable. Como vemos en las capturas de pantalla el resultado sigue siendo el mismo.

El código de `reduction-clauseModificado7.c` es el que se muestra en la siguiente imagen:

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

#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif

int main(int argc, char **argv){
    int i, n=20, a[n], suma=0;
    if(argc<2){
        fprintf(stderr, "Faltan 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;
    }

    omp_set_num_threads(n);
    #pragma omp parallel
    {
        suma+=a[omp_get_thread_num()];
    }

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

La siguiente imagen muestra varias ejecuciones de `reduction-clause.c` y `reductionclauseModificado.c`.



```

[FelixRamirezGarcia felix@DESKTOP-7
ciaFelix/ejer7] 2019-04-28 Sunday
$gcc -O2 -fopenmp -o reduction-clau
[FelixRamirezGarcia felix@DESKTOP-7
ciaFelix/ejer7] 2019-04-28 Sunday
$./reduction-clause 10
Tras 'parallel' suma=45
[FelixRamirezGarcia felix@DESKTOP-7
ciaFelix/ejer7] 2019-04-28 Sunday
$./reduction-clauseModificado7 10
Tras parallel suma=45
[FelixRamirezGarcia felix@DESKTOP-7
ciaFelix/ejer7] 2019-04-28 Sunday
$./reduction-clause 50
n=20Tras 'parallel' suma=190
[FelixRamirezGarcia felix@DESKTOP-7
ciaFelix/ejer7] 2019-04-28 Sunday
$./reduction-clauseModificado7 50
n=20Tras parallel suma=190

```

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):

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

- (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);
- (4) 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).

El código de pmv-secuencial.c se muestra en la siguiente imagen:

```

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

#ifdef _OPENMP
    #include <omp.h>
#else
    #define omp_get_thread_num() 0
    #define omp_get_num_threads() 1
#endif

int main(int argc, char ** argv){
    int i, j;
    double begin, end, t;

    if(argc<2){
        printf("Falta el tamaño de la matriz y el del vector\n");
        exit(-1);
    }

    unsigned int N=atoi(argv[1]);
    double *v1, *v2, **m;

    v1=(double*) malloc(N*sizeof(double));
    v2=(double*) malloc(N*sizeof(double));
    m=(double**) malloc(N*sizeof(double*));

    if( (v1==NULL) || (v2==NULL) || (m==NULL)){
        printf("Error al reservar el espacio\n");
        exit(-2);
    }

    for(i=0;i<N;i++){
        m[i]=(double*) malloc(N*sizeof(double));
        if(m[i]==NULL){
            printf("Error en la reserva de espacio\n");
            exit(-2);
        }
    }

    for(i=0;i<N;i++){
        v1[i]=i;
        v2[i]=0;
        for(j=0;j<N;j++){
            m[i][j]=i+j;
        }
    }

    begin=omp_get_wtime();

    for(i=0;i<N;i++){
        for(j=0;j<N;j++){
            v2[i]=m[i][j]*v1[j];
        }
    }

    end=omp_get_wtime();

    t=end-begin;

    printf("Tiempo: %11.9f\tTamaño:%u\tv2[0]=%8.6fv2[%d]=%8.6f\n", t, N, v2[0], N-1, v2[N-1]);

    if(N==8 || N==11){
        for(i=0;i<N;i++){
            printf("v2[%d]=%5.2f\n",i,v2[i]);
        }
    }

    free(v1);
    free(v2);

    for(i=0;i<N;i++){
        free(m[i]);
    }
    free(m);
}

```

La ejecución de `pmv-secuencial-dinamico.c` se muestra en la siguiente imagen:

```
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGarciaFelix/ejer8] 2019-04-28 Sunday
$gcc -O2 -fopenmp -o pmv-secuencial-dinamico pmv-secuencial-dinamico.c
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGarciaFelix/ejer8] 2019-04-28 Sunday
$./pmv-secuencial-dinamico 8
Tiempo: 0.000002900    Tamaño:8    v2[0]=49.000000v2[7]=98.000000
v2[0]=49.00
v2[1]=56.00
v2[2]=63.00
v2[3]=70.00
v2[4]=77.00
v2[5]=84.00
v2[6]=91.00
v2[7]=98.00
```

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).

- 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  $N=11$ );
- (4) 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).

El código de `pmv-OpenMP-a.c` se muestra en la siguiente imagen:

```

C pmv-OpenMP-a.c x
1
2
3
4 #ifdef _OPENMP
5 #include <omp.h>
6 #else
7 #define omp_get_thread_num() 0
8 #define omp_get_num_threads() 1
9 #endif
10
11 int main(int argc, char ** argv){
12     int i, j;
13     double begin, end, t;
14
15     if(argc<2){
16         printf("Falta el tamaño de la matriz y el del vector\n");
17         exit(-1);
18     }
19
20     unsigned int N=atoi(argv[1]);
21     double *v1, *v2, **m;
22
23     v1=(double*) malloc(N*sizeof(double));
24     v2=(double*) malloc(N*sizeof(double));
25     m=(double**) malloc(N*sizeof(double*));
26
27     if( (v1==NULL) || (v2==NULL) || (m==NULL)){
28         printf("Error al reservar el espacio\n");
29         exit(-2);
30     }
31
32     for(i=0;i<N;i++){
33         m[i]=(double*) malloc(N*sizeof(double));
34         if(m[i]==NULL){
35             printf("Error en la reserva de espacio\n");
36             exit(-2);
37         }
38     }
39
40     #pragma omp for
41     for(i=0;i<N;i++){
42         v1[i]=i;
43         v2[i]=0;
44         for(j=0;j<N;j++){
45             m[i][j]=i+j;
46         }
47     }
48
49     begin=omp_get_wtime();
50
51     #pragma omp for
52     for(i=0;i<N;i++){
53         for(j=0;j<N;j++){
54             v2[i]=m[i][j]*v1[j];
55         }
56     }
57
58     end=omp_get_wtime();
59
60     t=end-begin;
61
62     printf("Tiempo: %11.9f\tTamaño:%u\tv2[0]=%8.6fv2[%d]=%8.6f\n", t, N, v2[0], N-1, v2[N-1]);
63
64     if(N==8 || N==11){
65         for(i=0;i<N;i++){
66             printf("v2[%d]=%5.2f\n",i,v2[i]);
67         }
68     }
69
70     free(v1);
71     free(v2);
72
73     for(i=0;i<N;i++){
74         free(m[i]);
75     }
76     free(m);
77 }
78

```

Y el código de pmv-OpenMP-b.c se muestra en la siguiente imagen:

```

#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#define omp_get_num_threads() 1
#endif

int main(int argc, char ** argv){
    int i, j;
    double begin, end, t;

    if(argc<2){
        printf("Falta el tamaño de la matriz y el del vector\n");
        exit(-1);
    }

    unsigned int N=atoi(argv[1]);
    double *v1, *v2, **m;

    v1=(double*) malloc(N*sizeof(double));
    v2=(double*) malloc(N*sizeof(double));
    m=(double**) malloc(N*sizeof(double*));

    if( (v1==NULL) || (v2==NULL) || (m==NULL)){
        printf("Error al reservar el espacio\n");
        exit(-2);
    }

    for(i=0;i<N;i++){
        m[i]=(double*) malloc(N*sizeof(double));
        if(m[i]==NULL){
            printf("Error en la reserva de espacio\n");
            exit(-2);
        }
    }

    for(i=0;i<N;i++){
        v1[i]=i;
        v2[i]=0;
        #pragma omp for
        for(j=0;j<N;j++){
            m[i][j]=i+j;
        }
    }

    begin=omp_get_wtime();

    for(i=0;i<N;i++){
        #pragma omp for
        for(j=0;j<N;j++){
            v2[i]=m[i][j]*v1[j];
        }
    }

    end=omp_get_wtime();

    t=end-begin;

    printf("Tiempo: %11.9f\tTamaño:%u\tv2[0]=%8.6fv2[%d]=%8.6f\n", t, N, v2[0], N-1, v2[N-1]);

    if(N==8 || N==11){
        for(i=0;i<N;i++){
            printf("v2[%d]=%5.2f\n",i,v2[i]);
        }
    }

    free(v1);
    free(v2);

    for(i=0;i<N;i++){
        free(m[i]);
    }
    free(m);
}

```

Casualmente no obtuve ni un solo error en ninguna de las versiones, ni de compilación ni de ejecución. La siguiente imagen muestra la compilación sin errores.

```
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGarciaFelix/ejer9] 2019-04-28 Sunday
$gcc -O2 -fopenmp -o pmv-OpenMP-a pmv-OpenMP-a.c
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGarciaFelix/ejer9] 2019-04-28 Sunday
$gcc -O2 -fopenmp -o pmv-OpenMP-b pmv-OpenMP-b.c
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGarciaFelix/ejer9] 2019-04-28 Sunday
```

Y la siguiente imagen muestra la ejecución de pmv-secuencial-dinamico.c y de ambas versiones para la comparación de tiempos entre las 3, que no varia mucho

```
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGarciaFelix/ejer9] 2019-04-28 Sunday
$./pmv-secuencial-dinamico 1000
Tiempo: 0.000008800 Tamaño:1000 v2[0]=998001.000000v2[999]=1996002.000000
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGarciaFelix/ejer9] 2019-04-28 Sunday
$./pmv-OpenMP-a 1000
Tiempo: 0.000009800 Tamaño:1000 v2[0]=998001.000000v2[999]=1996002.000000
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGarciaFelix/ejer9] 2019-04-28 Sunday
$./pmv-OpenMP-b 1000
Tiempo: 0.000013500 Tamaño:1000 v2[0]=998001.000000v2[999]=1996002.000000
```

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).

El código de `pmv-OpenmMP-reduction.c` se muestra en la siguiente imagen:

```

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

#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#define omp_get_num_threads() 1
#endif

int main(int argc, char ** argv){
    int i, j;
    double begin, end, t;

    if(argc<2){
        printf("Falta el tamaño de la matriz y el del vector\n");
        exit(-1);
    }

    unsigned int N=atoi(argv[1]);
    double *v1, *v2, **m;

    v1=(double*) malloc(N*sizeof(double));
    v2=(double*) malloc(N*sizeof(double));
    m=(double**) malloc(N*sizeof(double*));

    if( (v1==NULL) || (v2==NULL) || (m==NULL)){
        printf("Error al reservar el espacio\n");
        exit(-2);
    }

    for(i=0;i<N;i++){
        m[i]=(double*) malloc(N*sizeof(double));
        if(m[i]==NULL){
            printf("Error en la reserva de espacio\n");
            exit(-2);
        }
    }

    for(i=0;i<N;i++){
        v1[i]=i;
        v2[i]=0;
        #pragma omp for
        for(j=0;j<N;j++){
            m[i][j]=i+j;
        }
    }

    begin=omp_get_wtime();

    for(i=0;i<N;i++){
        int aux=0;
        #pragma omp parallel for reduction (+:aux)
        for(j=0;j<N;j++){
            aux+=m[i][j]*v1[j];
        }
        v2[i]=aux;
    }

    end=omp_get_wtime();

    t=end-begin;

    printf("Tiempo: %11.9f\tTamaño:%u\tv2[0]=%8.6fv2[%d]=%8.6f\n", t, N, v2[0], N-1, v2[N-1]);

    if(N==8 || N==11){
        for(i=0;i<N;i++){
            printf("v2[%d]=%5.2f\n",i,v2[i]);
        }
    }

    free(v1);
    free(v2);

    for(i=0;i<N;i++){
        free(m[i]);
    }
    free(m);
}

```



No he tenido ningún tipo de error en compilación o en ejecución. La siguiente imagen lo muestra:

```
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGarciaFelix/ejer10] 2019-04-28 Sunday
$gcc -O2 -fopenmp -o pmw-OpenMP-reduction pmw-OpenMP-reduction.c
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGarciaFelix/ejer10] 2019-04-28 Sunday
$./pmw-OpenMP-reduction 10
Tiempo: 0.023761400      Tamaño:10      v2[0]=285.000000v2[9]=690.000000
```

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.

Para justificar el código seleccionado voy a ejecutar las tres versiones distintas con tres tamaños del problema distintos y seleccionare la que a priori mejor evolución tenga. En la siguiente imagen se puede apreciar que la mejor de todas es el modelo a , ya que tiene en los tres casos menor tiempo de ejecución.

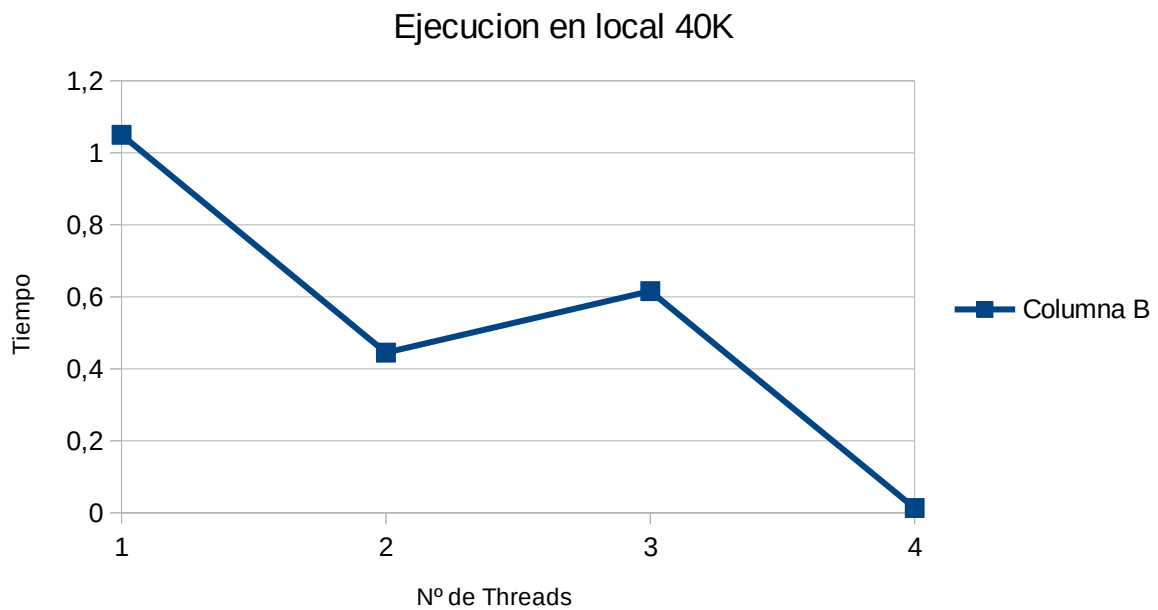
```
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp2_RamirezGarciaFelix/ejer11] 2019-04-28 Sunday
$./script-ejecutar-todos
Modelo a 1k
Tiempo: 0.000011300      Tamaño:1000      v2[0]=998001.000000v2[999]=1996002.000000
Modelo b 1k
Tiempo: 0.000014400      Tamaño:1000      v2[0]=998001.000000v2[999]=1996002.000000
Modelo reduction 1k
Tiempo: 0.018795200      Tamaño:1000      v2[0]=332833500.000000v2[999]=831834000.000000
Modelo a 7.5k
Tiempo: 0.000136700      Tamaño:7500      v2[0]=56235001.000000v2[7499]=112470002.000000
Modelo b 7.5k
Tiempo: 0.000157100      Tamaño:7500      v2[0]=56235001.000000v2[7499]=112470002.000000
Modelo reduction 7.5k
Tiempo: 0.584763900      Tamaño:7500      v2[0]=-1611364696.000000v2[7499]=736401900.000000
Modelo a 15k
Tiempo: 0.000337700      Tamaño:15000     v2[0]=224970001.000000v2[14999]=449940002.000000
Modelo b 15k
Tiempo: 0.000377900      Tamaño:15000     v2[0]=224970001.000000v2[14999]=449940002.000000
Modelo reduction 15k
Tiempo: 1.137857500      Tamaño:15000     v2[0]=-313900351.000000v2[14999]=-161510248.000000
```

La siguiente tabla muestra los resultados de la ejecución del modelo a en local para 1-4 threads y tamaño 40000:

Threads	Tamaño	Tiempo
1	40000	1.050074800
2	40000	0.444853200
3	40000	0.615770500
4	40000	0.013431900



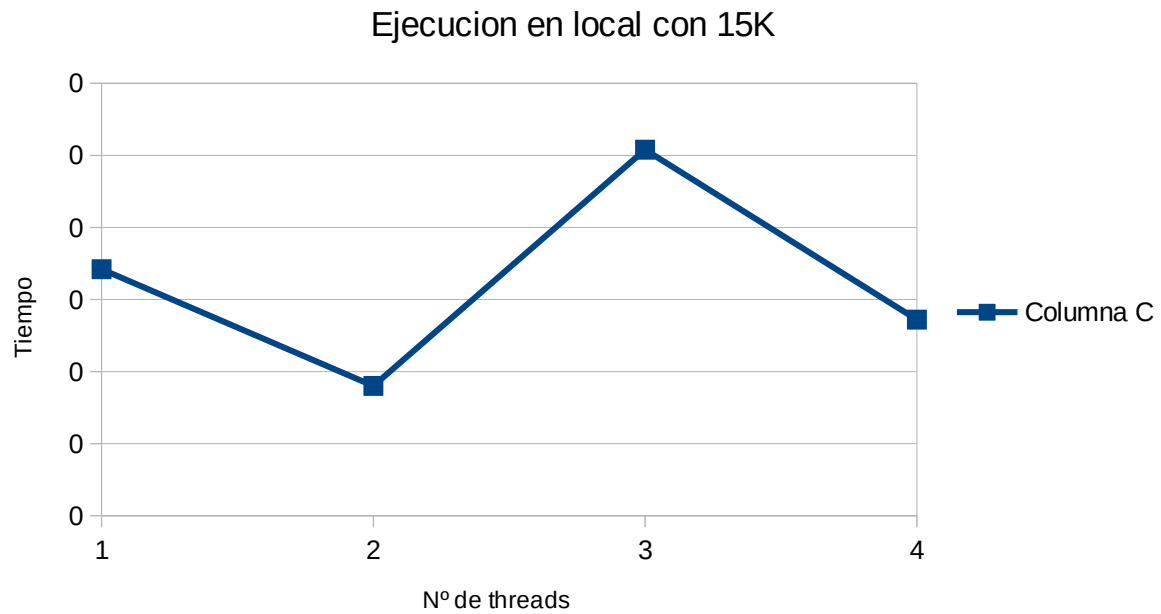
La siguiente imagen muestra la grafica de la ejecución de pmv-OpenMP-a.c con 40k elementos



La siguiente tabla muestra los resultados de la ejecución del modelo a en local para 1-4 threads y tamaño 15000:

Threads	Tamaño	Tiempo
1	15000	0.000327100
2	15000	0.000319000
3	15000	0.000335400
4	15000	0.000323600

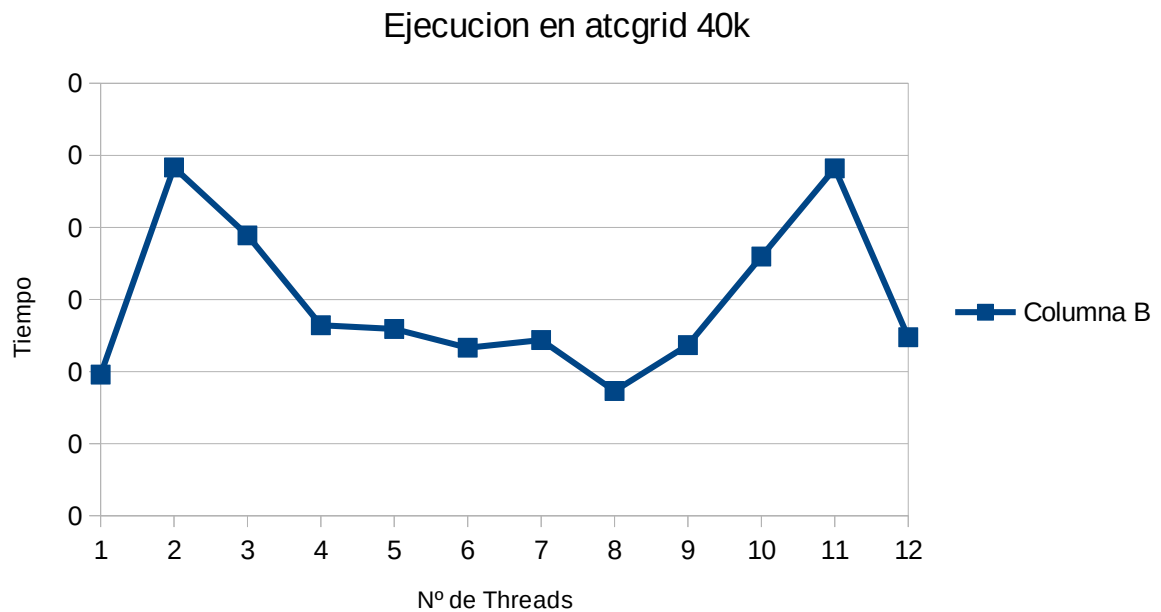
La siguiente imagen muestra la grafica de la ejecución de pmv-OpenMP-a.c con 15k elementos



Ahora pasamos alas ejecuciones en atcgrid, primero mostramos los resultados de la ejecución del modelo a en atcgrid para 1-12 threads y tamaño 40000:

Threads	Tamaño	Tiempo
1	40000	0.001239154
2	40000	0.001296670
3	40000	0.001277795
4	40000	0.001252859
5	40000	0.001251832
6	40000	0.001246629
7	40000	0.001248747
8	40000	0.001234621
9	40000	0.001247365
10	40000	0.001271929
11	40000	0.001296398
12	40000	0.001249545

La siguiente imagen muestra la grafica de la ejecución de pmv-OpenMP-a.c en atcgrid con 40k elementos :



Ahora pasamos alas ejecuciones en atcgrid, primero mostramos los resultados de la ejecución del modelo a en atcgrid para 1-12 threads y tamaño 15000

Threads	Tamaño	Tiempo
1	40000	0.000446752
2	40000	0.000445444
3	40000	0.000456025
4	40000	0.000437343
5	40000	0.000441643
6	40000	0.000439782
7	40000	0.000444902
8	40000	0.000447130
9	40000	0.000425289
10	40000	0.000440080
11	40000	0.000447138
12	40000	0.000436532

La siguiente imagen muestra la grafica de la ejecución de pmv-OpenMP-a.c en atcgrid con 15k elementos :

