

Grai2º curso / 2º
cuatr.

Grado Ing. Inform.

Doble Grado Ing.
Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos): Javier Gómez Luzón

Grupo de prácticas: C1

Fecha de entrega:

Fecha evaluación en clase:

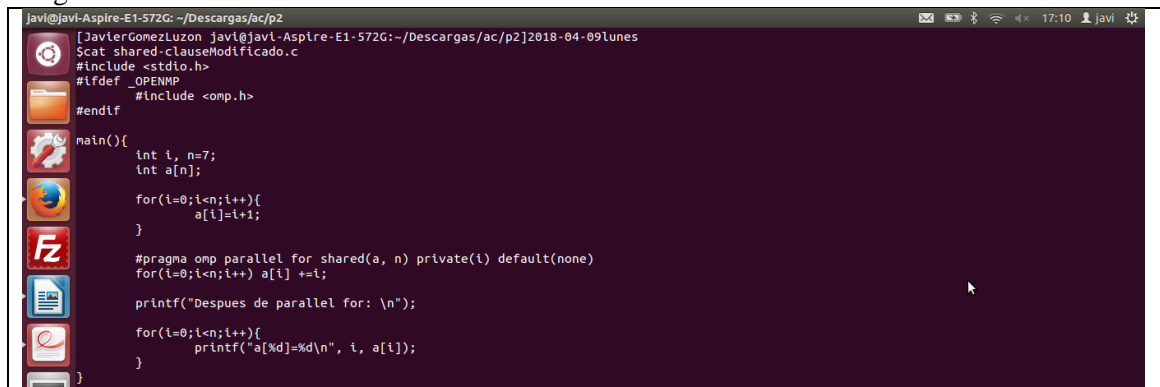
Ejercicios basados en los ejemplos del seminario práctico

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

RESPUESTA: La clausula `default(none)` hace que se tengan que especificar los ambitos de todas las variables involucradas en la región paralela.

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

Imagen 1: Cat de `shared-clauseModificado.c`.



```
javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-09lunes
$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;
    }

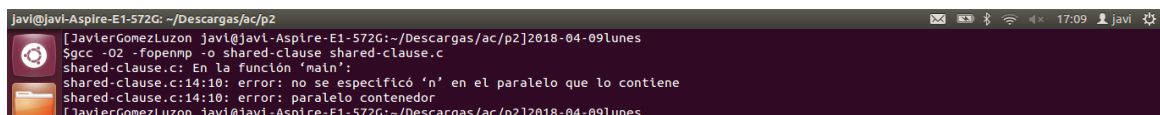
    #pragma omp parallel for shared(a, n) private(i) default(none)
    for(i=0;i<n;i++) a[i] +=i;

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

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

CAPTURAS DE PANTALLA:

Imagen 2: Error al compilar `shared-clauseModificado.c`.



```
javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-09lunes
$gcc -O2 -fopenmp -o shared-clause shared-clause.c
shared-clause.c: En la función 'main':
shared-clause.c:14:10: error: no se especificó 'n' en el paralelo que lo contiene
shared-clause.c:14:10: error: paralelo contenedor
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-09lunes
```

2. ¿Qué ocurre si en `private-clause.c` se inicializa la variable `suma` fuera de la construcción `parallel` en lugar de dentro? (inicialice `suma` a un valor distinto de 0 dentro y fuera de `parallel`) Razone su respuesta. Añada el código con la modificación al cuaderno de prácticas.

RESPUESTA: 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.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`

Imagen 3: Cat de private-clauseModificado.c.

```

Archivo Editar Ver Buscar Terminal Ayuda
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16lunes
$ cat private-clauseModificado.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=3;

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

    #pragma omp parallel private (suma)
    {
        suma=4;
        #pragma omp for
        for(i=0;i<n;i++){
            suma=suma+a[i];
            printf("thread %d suma %d/", omp_get_thread_num(),i);
            printf("\nthread %d suma=%d", omp_get_thread_num(), suma);
        }
    }
    printf("\n");
}

```

CAPTURAS DE PANTALLA:

Imagen 4: Ejecución de private-clauseModificado.c.

```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16lunes
$ ./private-clauseModificado
thread 0 suma a[0]/
*thread 0 suma=4thread 0 suma a[1]/
*thread 0 suma=5thread 3 suma a[6]/
*thread 3 suma=10thread 2 suma a[4]/
*thread 2 suma=8thread 2 suma a[5]/
*thread 2 suma=13thread 1 suma a[2]/
*thread 1 suma=6thread 1 suma a[3]/
*thread 1 suma=9

```

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

RESPUESTA: Que la variable pasa a ser compartida y todas las hebras podrán modificarla, los valores irían machacándose unos a otros.

CAPTURA CÓDIGO FUENTE: private-clauseModificado3.c

Imagen 5: Cat de private-clauseModificado3.c.

```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16lunes
$ cat private-clauseModificado3.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=0;
        #pragma omp for
        for(i=0;i<n;i++){
            suma=suma+a[i];
            printf("thread %d suma %d/", omp_get_thread_num(),i);
            printf("\nthread %d suma=%d", omp_get_thread_num(), suma);
        }
    }
    printf("\n");
}

```

CAPTURAS DE PANTALLA:

Imagen 6: Ejecución de private-clauseModificado3.c.

```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16lunes
$ ./private-clauseModificado3
thread 0 suma a[0]/
*thread 0 suma=6thread 0 suma a[1]/
*thread 0 suma=7thread 1 suma a[2]/
*thread 1 suma=7thread 1 suma a[3]/
*thread 1 suma=10thread 3 suma a[6]/
*thread 3 suma=10thread 2 suma a[4]/
*thread 2 suma=10thread 2 suma a[5]/
*thread 2 suma=15

```

- 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.

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

CAPTURAS DE PANTALLA:

Imagen 7: Multiple ejecución de `firstlastprivate-clause.c`

```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7
thread 1 suma a[5] suma=12
thread 2 suma a[6] suma=6
Fuera de la construccion parallel suma=6
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16lunes
$ ./firstlastprivate-clause
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 2 suma a[6] suma=6
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7
thread 1 suma a[5] suma=12
Fuera de la construccion parallel suma=6
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16lunes
$ ./firstlastprivate-clause
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=3
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7
thread 1 suma a[5] suma=12
thread 2 suma a[6] suma=6
Fuera de la construccion parallel suma=6
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16lunes

```

- ¿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?

RESPUESTA: Lo que ocurre es que solo un thread tiene la variable bien inicializada y los demás tienen basura. Al no tener la clausula `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.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

Imagen 8: Cat de `copyprivate-clauseModificado.c`.

```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16lunes
$ cat copyprivate-clauseModificado.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 inicializacion 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("\nDespues de la region parallel:\n");
        for(i=0;i<n;i++) printf("b[%d]= %d\t", i, b[i]);
        printf("\n");
    }
}

```

CAPTURAS DE PANTALLA:

Imagen 9: Ejecución de `copyprivate-clause.c`

```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16lunes
$ ./copyprivate-clause
Introduce valor de inicializacion a: 3
Single ejecutada por el thread 0
Despues de la region 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
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16lunes
$

```

Imagen 10: Ejecución de `copyprivate-clauseModificado.c`.

```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16lunes
$ ./copyprivate-clauseModificado
Introduce valor de inicializacion a: 3
Single ejecutada por el thread 0
Despues de la region parallel:
b[0]= 3 b[1]= 3 b[2]= 3 b[3]= 32663 b[4]= 32663 b[5]= 32663 b[6]= 32663 b[7]= 32663 b[8]= 32663
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16lunes
$

```

6. En el ejemplo `reduction-clause.c` sustituya `suma=0` por `suma=10`. ¿Qué resultado se imprime ahora? Justifique el resultado

RESPUESTA: 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.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

Imagen 11: Cat de `reduction-clauseModificado.c`.

```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16domingo
$ cat reduction-clauseModificado.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;

    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);
}

```

CAPTURAS DE PANTALLA:

Imagen 12: Ejecución de `reduction-clause.c` con parámetro 10.

```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16lunes
$ ./reduction-clause 10
Tras 'paralel' suma=45
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16lunes

```

Imagen 13: Ejecución de `reduction-clauseModificado.c` con parámetro 10.

```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16lunes
$ ./reduction-clauseModificado 10
Tras 'paralel' suma=55
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-16lunes

```

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 usar directivas de trabajo compartido.

RESPUESTA: 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.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`

Imagen 14: Cat de reduction-clauseModificado7.c.

```

C:\estudiante9@atcgriid:-
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-22domingo
$ cat reduction-clauseModificado7.c
#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);
}

```

CAPTURAS DE PANTALLA:

Imagen 15: Varias ejecuciones de reduction-clause.c y reduction-clauseModificado.c.

```

C:\estudiante9@atcgriid:-
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-22domingo
$ ./reduction-clause 10
Tras parallel suma=45
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-22domingo
$ ./reduction-clauseModificado7 10
Tras parallel suma=45
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-22domingo
$ ./reduction-clause 50
n=20Tras parallel suma=190
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-22domingo
$ ./reduction-clauseModificado7 50
n=20Tras parallel suma=190

```

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):

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

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

CAPTURA CÓDIGO FUENTE: pmv-secuencial.c

Imagen 16 y 17: Cat de pmv-secuencial-dinamico.c.

```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-21sábado
$ cat pmv-secuencial-dinamico.c
#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:\xu\tv2[0]=%8.6fv2[2]=%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);
}

```

CAPTURAS DE PANTALLA:

Imagen 18: Ejecucion de pmv-secuencial-dinamico.c.

```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-21sábado
$ ./pmv-secuencial-dinamico 8
Tiempo: 0.000001116      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
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-21sábado
$ ./pmv-secuencial-dinamico 11
Tiempo: 0.000001151      Tamaño:11     v2[0]=100.000000v2[10]=200.000000
v2[0]=100.00
v2[1]=110.00
v2[2]=120.00
v2[3]=130.00
v2[4]=140.00
v2[5]=150.00
v2[6]=160.00
v2[7]=170.00
v2[8]=180.00
v2[9]=190.00
v2[10]=200.00
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-21sábado

```

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.

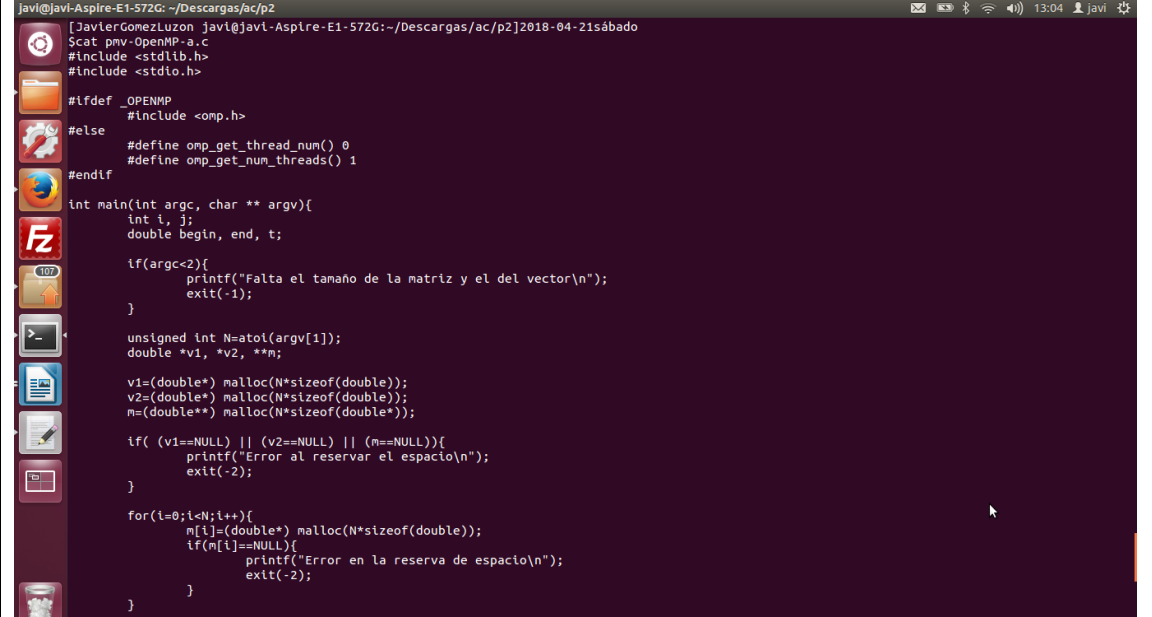
Use las directivas que estime oportunas y las cláusulas que sean necesarias **excepto la cláusula `reduction`**. Se debe paralelizar también la inicialización de las matrices. Respecto a este ejercicio:

- 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

Imagen 19 y 20: Cat pmv-OpenMP-a.c.



```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-21sábado
$ cat pmv-OpenMP-a.c
#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);
        }
    }
}

```



```
#pragma omp for
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();

#pragma omp for
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);
}
```

[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-21sábado

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

Imagen 21 y 22: Cat pmv-OpenMP-b.c.

```
javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[JavierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-21sábado
$cat pmv-OpenMP-b.c
#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;

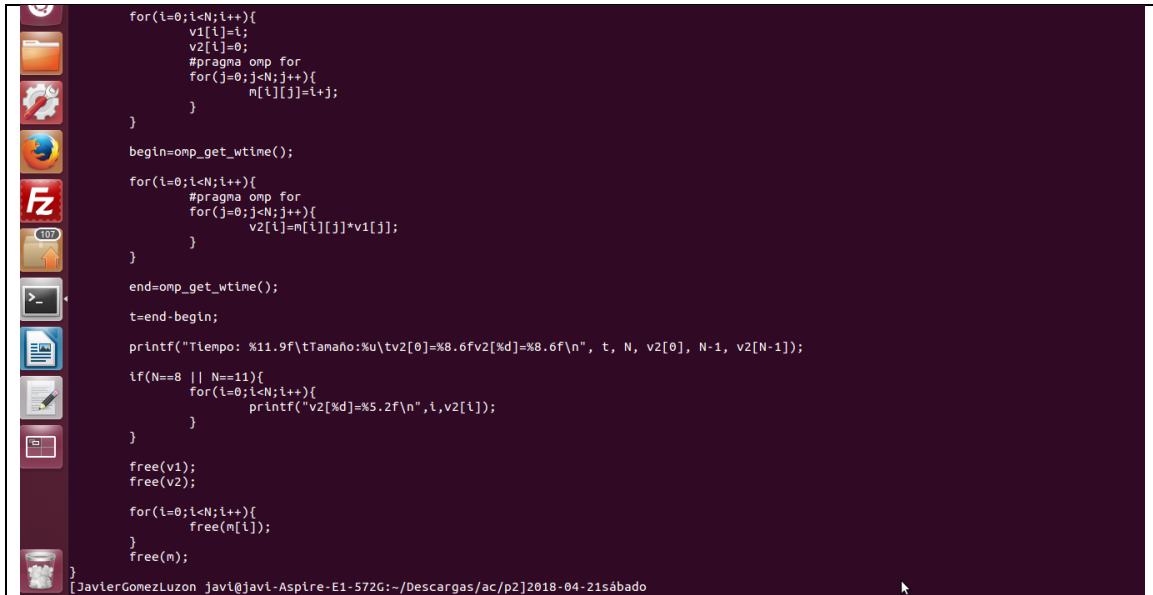
    if(argc<2){
        printf("Falta el tamaño de la matriz y el del vector\n");
        exit(-1);
    }

    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++){
        n[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]=n[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);

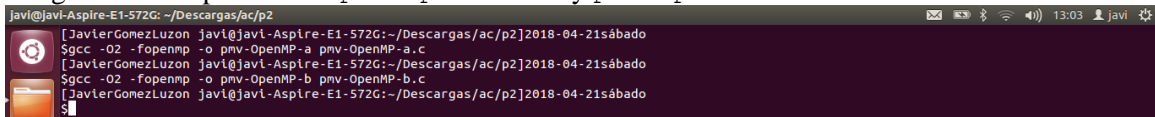
for(i=0;i<N;i++){
    free(n[i]);
}
free(n);
}

```

RESPUESTA: Casualmente no obtuve ni un solo error en ninguna de las versiones, ni de compilación ni de ejecución. La imagen 23 muestra la ejecución sin errores. La imagen 24 muestra la ejecución de ambas versiones y la comparación de tiempos entre las 3, que no varia mucho.

CAPTURAS DE PANTALLA:

Imagen 23: Compilación de `pmv-OpenMP-a.c` y `pmv-OpenMP-b.c`.

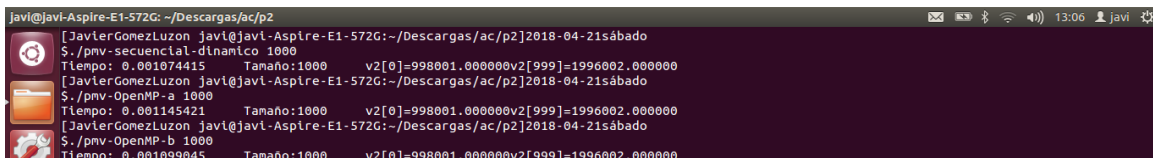


```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
$ gcc -O2 -fopenmp -o pmv-OpenMP-a pmv-OpenMP-a.c
$ gcc -O2 -fopenmp -o pmv-OpenMP-b pmv-OpenMP-b.c

```

Imagen 24: Ejecución de `pmv-secuencial-dinamico.c`, `pmv-OpenMP-a.c` y `pmv-OpenMP-b.c`.



```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
$ ./pmv-secuencial-dinamico 1000
Tiempo: 0.001074415 Tamaño:1000 v2[0]=998001.000000v2[999]=1996002.000000
$ ./pmv-OpenMP-a 1000
Tiempo: 0.001145421 Tamaño:1000 v2[0]=998001.000000v2[999]=1996002.000000
$ ./pmv-OpenMP-b 1000
Tiempo: 0.001099045 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).

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

Imagen 25 y 26: Cat de pmv-OpenMP-reduction.c.



```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-21sábado
$ cat pmv-OpenMP-reduction.c
#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);

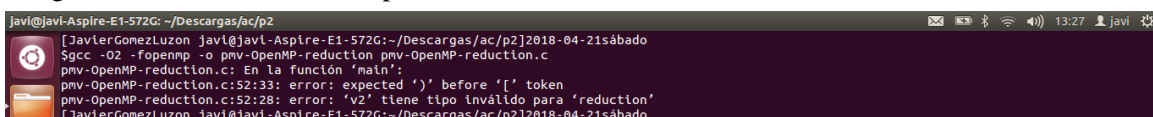
    for(i=0;i<N;i++){
        free(m[i]);
    }
    free(m);
}

```

RESPUESTA: La imagen 27 nos muestra el primer error, el error es porque OpenMP no permite hacer reduction con vectores. Lo solucioné guardando el valor en una variable y luego asignando esa variable al vector. No he utilizado ninguna ayuda para detectar este error ya que el mismo compilador te dice porque no funciona.

CAPTURAS DE PANTALLA:

Imagen 27: Primer error de compilación de pmv-OpenMP-reduction.c.



```

javi@javi-Aspire-E1-572G: ~/Descargas/ac/p2
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-21sábado
$ gcc -O2 -fopenmp -o pmv-OpenMP-reduction pmv-OpenMP-reduction.c
pmv-OpenMP-reduction.c: En la función 'main':
pmv-OpenMP-reduction.c:52:33: error: expected ')' before '[' token
pmv-OpenMP-reduction.c:52:28: error: 'v2' tiene tipo inválido para 'reduction'
[javierGomezLuzon javi@javi-Aspire-E1-572G:~/Descargas/ac/p2]2018-04-21sábado

```

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):

Imagen 28: Ejecución de los 3 modelos con distintos tamaños.

```

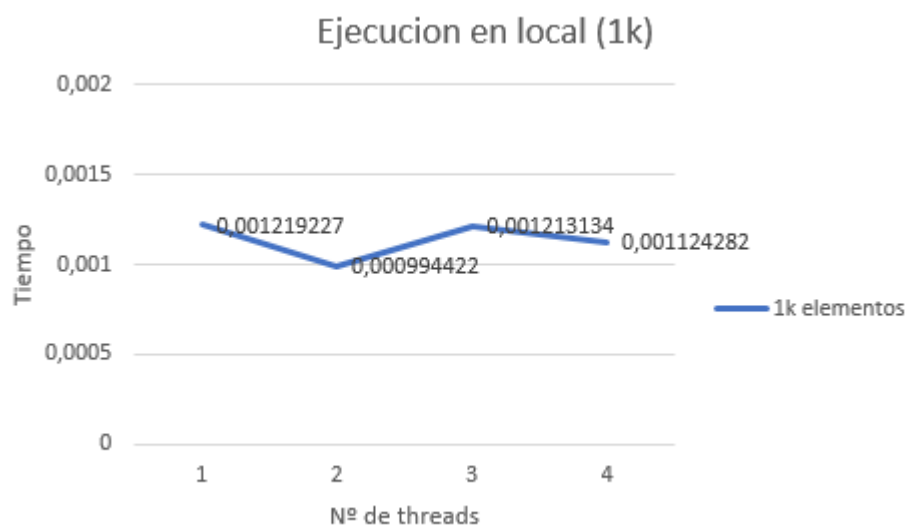
javi@javi-Aspire-E1-572G: ~
$ ./script-ejecutar-todos
a 1k
Tiempo: 0.000916227 Tamaño:1000 v2[0]=998001.000000v2[999]=1996002.000000
b 1k
Tiempo: 0.001010605 Tamaño:1000 v2[0]=998001.000000v2[999]=1996002.000000
reduction 1k
Tiempo: 0.004472201 Tamaño:1000 v2[0]=332833500.000000v2[999]=831834000.000000
a 7.5k
Tiempo: 0.038547236 Tamaño:7500 v2[0]=56235001.000000v2[7499]=112470002.000000
b 7.5k
Tiempo: 0.043307316 Tamaño:7500 v2[0]=56235001.000000v2[7499]=112470002.000000
reduction 7.5k
Tiempo: 0.074914206 Tamaño:7500 v2[0]=-1764014177.000000v2[7499]=670887372.000000
a 15k
Tiempo: 0.152247147 Tamaño:15000 v2[0]=224970001.000000v2[14999]=449940002.000000
b 15k
Tiempo: 0.172283197 Tamaño:15000 v2[0]=224970001.000000v2[14999]=449940002.000000
reduction 15k
Tiempo: 0.267472487 Tamaño:15000 v2[0]=-642846265.000000v2[14999]=79980676.000000
javi@javi-Aspire-E1-572G: ~

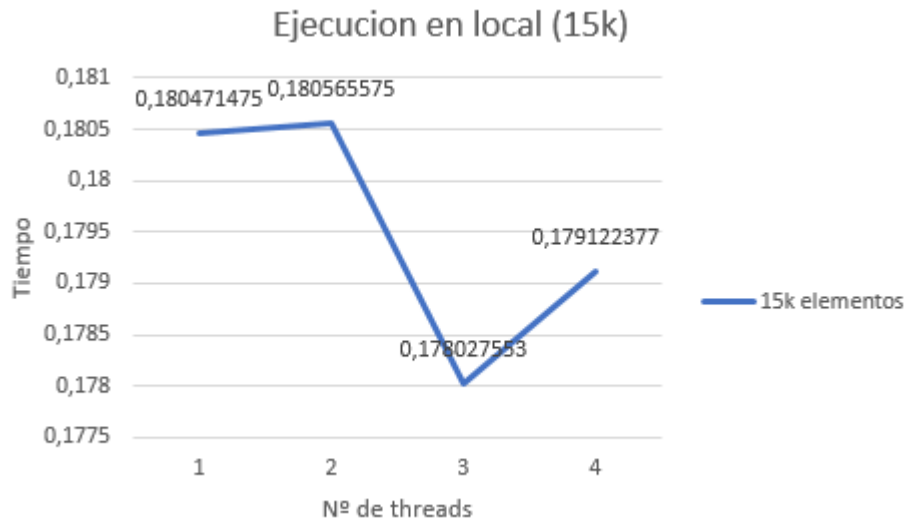
```

TABLA Y GRÁFICA (por ejemplo para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N-: un N entre 30000 y 100000, y otro entre 5000 y 30000):

| Ejecución en local | | |
|--------------------|--------|-------------|
| Threads | Tamaño | Tiempo |
| 1 | 1k | 0,001219227 |
| 2 | 1k | 0,000994422 |
| 3 | 1k | 0,001213134 |
| 4 | 1k | 0,001124282 |
| 1 | 15k | 0,180471475 |
| 2 | 15k | 0,180565575 |
| 3 | 15k | 0,178027553 |
| 4 | 15k | 0,179122377 |

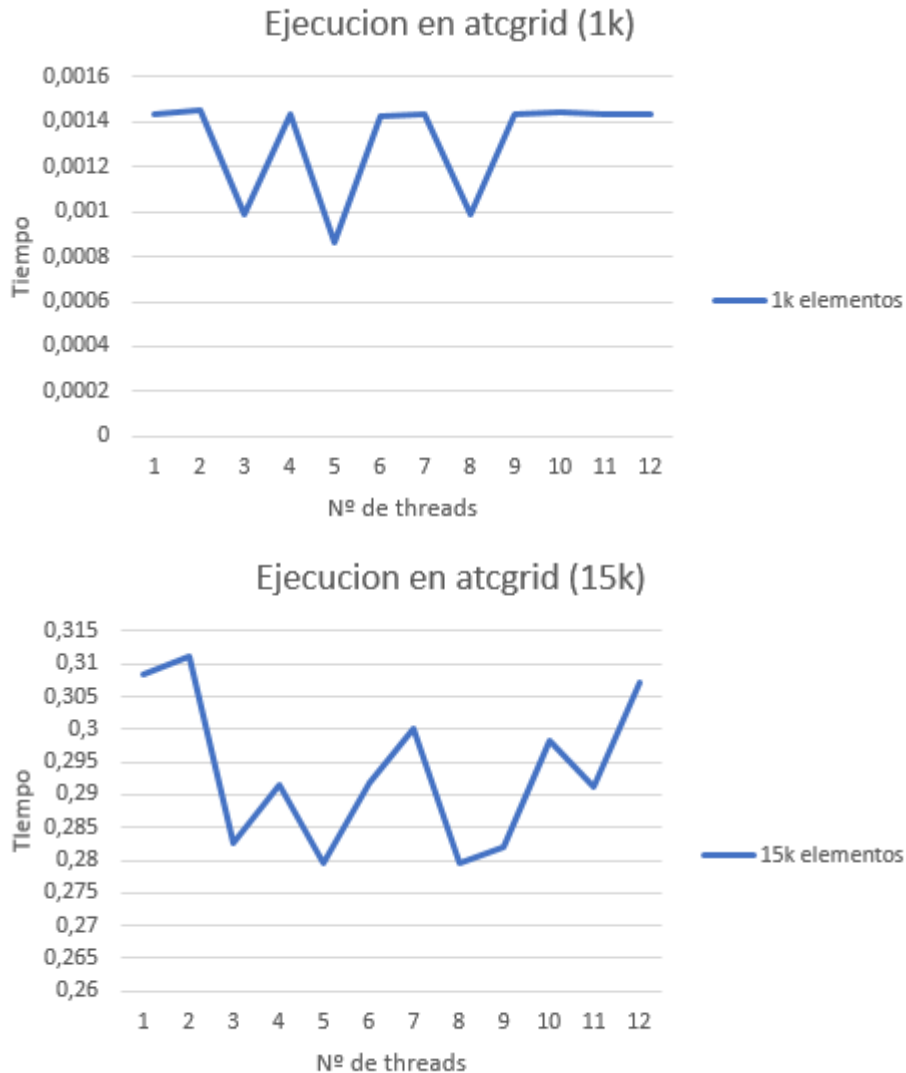
Imagen 29: Gráficos de la ejecución de pmv-OpenMP-a.c con 1k elementos de 1 a 4 threads y la ejecución de pmv-OpenMP-a.c con 15k elementos de 1 a 4 threads en local.





| Ejecución en atcgrid | | |
|----------------------|--------|-------------|
| Cores | Tamaño | Tiempo |
| 1 | 1k | 0,001436935 |
| 2 | 1k | 0,001454554 |
| 3 | 1k | 0,000992343 |
| 4 | 1k | 0,001438038 |
| 5 | 1k | 0,000864409 |
| 6 | 1k | 0,001429858 |
| 7 | 1k | 0,001433291 |
| 8 | 1k | 0,000987449 |
| 9 | 1k | 0,001437763 |
| 10 | 1k | 0,001445004 |
| 11 | 1k | 0,001433333 |
| 12 | 1k | 0,001434143 |
| 1 | 15k | 0,308376599 |
| 2 | 15k | 0,311042013 |
| 3 | 15k | 0,282776085 |
| 4 | 15k | 0,291396450 |
| 5 | 15k | 0,279698925 |
| 6 | 15k | 0,291860985 |
| 7 | 15k | 0,300205236 |
| 8 | 15k | 0,279701190 |
| 9 | 15k | 0,282010047 |
| 10 | 15k | 0,298419325 |
| 11 | 15k | 0,291088920 |
| 12 | 15k | 0,307254425 |

Imagen 30: Gráficos de la ejecución de `pmv-OpenMP-a.c` con 1k elementos de 1 a 12 threads y la ejecución de `pmv-OpenMP-a.c` con 15k elementos de 1 a 12 threads en `atcgrid`.



COMENTARIOS SOBRE LOS RESULTADOS: He escogido el código `pmv-OpenMP-a.c` porque es el que menos tiempo ha tardado en ejecutar con los 3 diferentes elementos.

Como vemos cuando hay 1k elementos va oscilando el tiempo, supongo que es porque son muy pocos elementos para que haya una diferencia notoria en el tiempo.

Cuando hay 15k elementos el tiempo si disminuye, aunque va oscilando hacia arriba también. Luego me ha sorprendido que con 12 threads en `atcgrid` tardara un tiempo similar al que tardo con 1.

No he podido realizar las pruebas con mas de 30k porque mi ordenador mataba el proceso.