

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): Jose Luis Pedraza Román

Grupo de prácticas y profesor de prácticas: A2 – Christian Morillas

Fecha de entrega: 15/4/2020

Fecha evaluación en clase: 16/4/2020

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

Ejercicios basados en los ejemplos del seminario práctico

1. ¿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: Cuando usamos la clausula “default(none)” debemos especificar en la cláusula “shared” todas las variables implicadas en la región parallel. De no se así, recibiremos un error del compilador diciendo que “n” no está especificada en la región parallel.

CAPTURA CÓDIGO FUENTE: shared-clauseModificado.c

```
#include <stdio.h>

#ifdef _OPENMP
    #include <omp.h>
#endif

int main(int argc, char const *argv[])
{
    int i, n=7;
    int a[n];

    #pragma omp parallel for default(none) shared(a,n)
    for(i=0;i<n;++i) a[i] = i + 1;

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

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

    return 0;
}
```

CAPTURAS DE PANTALLA:

Error:

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer1] 2020-04-12 domingo
$gcc -fopenmp -O2 -o shared_clause shared_clause.c
shared_clause.c: In function 'main':
shared_clause.c:12:10: error: 'n' not specified in enclosing 'parallel'
    #pragma omp parallel for default(none) shared(a)
               ^~~~
shared_clause.c:12:10: error: enclosing 'parallel'
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer1] 2020-04-12 domingo
$
```

Incluyendo “n” en “shared”:

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer1] 2020-04-12 domingo
$gcc -fopenmp -O2 -o shared_clauseModificado shared_clauseModificado.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer1] 2020-04-12 domingo
$./shared_clauseModificado
Despues del parallel for:
a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5
a[5] = 6
a[6] = 7
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer1] 2020-04-12 domingo
$
```

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`? (muéstrelolo 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: A la primera pregunta, el código imprime 0 fuera de `parallel` aun habiendo inicializado “suma” a 10 dado que por defecto se inicializa a 0 fuera de `parallel`.

A la segunda pregunta, es completamente necesario inicializar la variable dentro de la región `parallel` ya que es una variable privada para esta sección. Por lo tanto esta variable almacenará “basura” si la inicializamos fuera (si la inicializamos dentro funcionará correctamente).

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`

```
#include <stdio.h>

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

//gcc -O2 -fopenmp -o private_clause private_clause.c

int main(int argc, char const *argv[])
{
    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 + i;
            printf("thread %d, suma a[%d] = %d\n", omp_get_thread_num(),i,a[i]);
        }
    }
```

```

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

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

    return 0;
}

```

CAPTURAS DE PANTALLA:

A la primera pregunta:

```

[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer2] 2020-04-12 domingo
$gcc -fopenmp -O2 -o private_clause private_clause.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer2] 2020-04-12 domingo
$./private_clause
thread 4, suma a[4] = 4
thread 0, suma a[0] = 0
thread 1, suma a[1] = 1
thread 3, suma a[3] = 3
thread 5, suma a[5] = 5
thread 2, suma a[2] = 2
thread 6, suma a[6] = 6

thread 5 suma = 15
thread 4 suma = 14
thread 3 suma = 13
thread 7 suma = 10
thread 1 suma = 11
thread 0 suma = 10
thread 2 suma = 12
thread 6 suma = 16

suma = 0
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer2] 2020-04-12 domingo
$

```

A la segunda pregunta:

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer2] 2020-04-12 domingo
$gcc -fopenmp -O2 -o private_clause private_clause.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer2] 2020-04-12 domingo
$./private_clause
thread 0, suma a[0] = 0
thread 1, suma a[1] = 1
thread 6, suma a[6] = 6
thread 5, suma a[5] = 5
thread 2, suma a[2] = 2
thread 4, suma a[4] = 4
thread 3, suma a[3] = 3

thread 2 suma = -118
thread 4 suma = -116
thread 5 suma = -115
thread 1 suma = -119
thread 7 suma = -120
thread 3 suma = -117
thread 6 suma = -114
thread 0 suma = 510769456

suma = 10
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer2] 2020-04-12 domingo
$
```

Añadir a lo necesario a `private_clause.c` para que imprima suma fuera de la región `parallel` e imprimiendo suma a un valor distinto de 0. Ejecutar varias veces el código. ¿Qué imprime el código fuera del `parallel` (comparado con una captura de pantalla). ¿Qué ocurre si en esta versión de `private_clause.c` se declara la variable fuera fuera de la construcción `parallel` en lugar de dentro? Hazme un resumen tabular de capturas de pantalla que muestren lo que ocurre. Añadir el código con las modificaciones al cuaderno de prácticas.

RESPUESTA: A la primera pregunta, el código imprime 0 fuera de `parallel` aun habiendo inicializado "suma" a 10 dado que por defecto se imprime a 0 fuera de `parallel`.

A la segunda pregunta, es completamente necesario declarar la variable dentro de la región `parallel` ya que es una variable privada para esta sección. Por lo tanto esta variable almacenará "suma" si la inicializamos fuera (o la inicializamos dentro inmediatamente en su lugar).

CAPTURA CÓDIGO FUENTE: `private_clauseModificado.c`

CAPTURAS DE PANTALLA:

A la primera pregunta:

Ejecución correcta `private_clauseModificado`:

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer2] 2020-04-12 domingo
$gcc -fopenmp -O2 -o private_clauseModificado private_clauseModificado.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer2] 2020-04-12 domingo
$./private_clauseModificado
thread 0, suma a[0] = 0
thread 3, suma a[3] = 3
thread 5, suma a[5] = 5
thread 2, suma a[2] = 2
thread 4, suma a[4] = 4
thread 6, suma a[6] = 6
thread 1, suma a[1] = 1

thread 6 suma = 16
thread 2 suma = 12
thread 4 suma = 14
thread 3 suma = 13
thread 7 suma = 10
thread 0 suma = 10
thread 1 suma = 11
thread 5 suma = 15

suma = 0
```

Añadir a lo necesario a `private_clause.c` para que imprima suma fuera de la región `parallel` e imprimiendo suma a un valor distinto de 0. Ejecutar varias veces el código. ¿Qué imprime el código fuera del `parallel` (comparado con una captura de pantalla). ¿Qué ocurre si en esta versión de `private_clause.c` se declara la variable fuera fuera de la construcción `parallel` en lugar de dentro? Hazme un resumen tabular de capturas de pantalla que muestren lo que ocurre. Añadir el código con las modificaciones al cuaderno de prácticas.

RESPUESTA: A la primera pregunta, el código imprime 0 fuera de `parallel` aun habiendo inicializado "suma" a 10 dado que por defecto se imprime a 0 fuera de `parallel`.

A la segunda pregunta, es completamente necesario declarar la variable dentro de la región `parallel` ya que es una variable privada para esta sección. Por lo tanto esta variable almacenará "suma" si la inicializamos fuera (o la inicializamos dentro inmediatamente en su lugar).

CAPTURA CÓDIGO FUENTE: `private_clauseModificado.c`

CAPTURAS DE PANTALLA:

A la primera pregunta:

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

RESPUESTA: Dejará de ser una variable privada para la sección paralela (será compartida), por lo tanto todas las hebras almacenarán su valor calculado en la misma variable (por lo que vemos en la salida que todas muestran el mismo valor, dado que las distintas hebras sobrescriben el estado previo, mostrando el último valor calculado).

CAPTURA CÓDIGO FUENTE: `private-clauseModificado2.c`

```
#include <stdio.h>

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

//gcc -O2 -fopenmp -o private_clause private_clause.c

int main(int argc, char const *argv[])
{
    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 + i;
            printf("thread %d, suma a[%d] =
%d\n", omp_get_thread_num(),i,a[i]);
        }

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

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

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer3] 2020-04-12 domingo
$gcc -fopenmp -O2 -o private_clauseModificado2 private_clauseModificado2.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer3] 2020-04-12 domingo
$./private_clauseModificado2
thread 0, suma a[0] = 0
thread 6, suma a[6] = 6
thread 1, suma a[1] = 1
thread 2, suma a[2] = 2
thread 5, suma a[5] = 5
thread 4, suma a[4] = 4
thread 3, suma a[3] = 3
thread 5 suma = 10
thread 2 suma = 10
thread 1 suma = 10
thread 0 suma = 10
thread 6 suma = 10
thread 7 suma = 10
thread 3 suma = 10
thread 4 suma = 10
suma = 10
```

RESPUESTA:

CAPTURA CÓDIGO FUENTE: private_clauseModificado2.c

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

/*gcc -O2 -fopenmp -o private_clause private_clause.c
int main(int argc, char const *argv[])
{
    int i, n=7;
    int *a[n], suma;
    for (i = 0; i < n; ++i)
    {
```


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: Al tener la cláusula “`lastprivate`”, siempre imprimirá el valor de la variable “`suma`” de la hebra que realice la última iteración.

CAPTURAS DE PANTALLA:

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer4] 2020-04-12 domingo
$gcc -fopenmp -O2 -o firstlastprivate firstlastprivate.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer4] 2020-04-12 domingo
$./firstlastprivate
thread 0, suma a[0] SUMA = 0
thread 3, suma a[3] SUMA = 3
thread 5, suma a[5] SUMA = 5
thread 2, suma a[2] SUMA = 2
thread 4, suma a[4] SUMA = 4
thread 6, suma a[6] SUMA = 6
thread 1, suma a[1] SUMA = 1

Fuera de la region parallel suma = 6
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer4] 2020-04-12 domingo
$./firstlastprivate
thread 0, suma a[0] SUMA = 0
thread 1, suma a[1] SUMA = 1
thread 4, suma a[4] SUMA = 4
thread 3, suma a[3] SUMA = 3
thread 6, suma a[6] SUMA = 6
thread 2, suma a[2] SUMA = 2
thread 5, suma a[5] SUMA = 5

Fuera de la region parallel suma = 6
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer4] 2020-04-12 domingo
$./firstlastprivate
thread 0, suma a[0] SUMA = 0
thread 6, suma a[6] SUMA = 6
thread 3, suma a[3] SUMA = 3
thread 4, suma a[4] SUMA = 4
thread 1, suma a[1] SUMA = 1
thread 2, suma a[2] SUMA = 2
thread 5, suma a[5] SUMA = 5

Fuera de la region 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)

RESPUESTA: No funciona correctamente dado que no se realiza correctamente la difusión a todas las hebras del valor de “`a`” ya que no copia la variable privada, entonces la variable “`a`” en algunos casos contendrá “`basura/val_inicial`” porque la hemos leído en una hebra pero no la hemos copiado en las demás (el valor solo se guardará en la hebra que ejecute la región “`single`”), por lo que es necesario el “`copyprivate`” para una vez terminado el “`single`”, se copie la variable “`a`” a las demás hebras mediante esta difusión.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```
#include <stdio.h>

#ifdef _OPENMP
    #include <omp.h>
#else
```

```

        #define omp_get_thread_num()0
#endif

int main(int argc, char const *argv[])
{
    int i, n=9, b[n];

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

    #pragma omp parallel
    {
        int a;
        #pragma omp single /*copyprivate(a)*/
        {
            printf("Introduce valor de a: ");
            scanf("%d",&a);
            printf("\nSingle ejecutada por la hebra: %d\n", omp_get_thread_num());
        }
        #pragma omp for
        for(i=0;i<n;++i) b[i]=a;
    }

    printf("\nFuera de la region parallel:\n");
    for (i = 0; i < n; ++i)
    {
        printf("b[%d] = %d\n", i, b[i]);
    }

    return 0;
}

```

CAPTURAS DE PANTALLA:

```

[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer5] 2020-04-12 domingo
$gcc -fopenmp -O2 -o copyprivate_clauseModificado copyprivate_clauseModificado.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer5] 2020-04-12 domingo
$./copyprivate_clauseModificado
Introduce valor de a: 10

Single ejecutada por la hebra: 1

Fuera de la region parallel:
b[0] = 0
b[1] = 0
b[2] = 10
b[3] = 0
b[4] = 0
b[5] = 0
b[6] = 0
b[7] = 0
b[8] = 0

```


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: Obtendremos como resultado “suma” + 10, ya que sumamos todos los números de 0 a n guardandolos en la variable “suma”, por lo que si cambiamos el valor inicial de dicha variable vemos cómo el resultado incrementa en ese valor inicial.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

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

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer6] 2020-04-12 domingo
$gcc -fopenmp -O2 -o reduction-clause reduction-clause.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer6] 2020-04-12 domingo
$./reduction-clause 10
Tras 'parallel' suma=45
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer6] 2020-04-12 domingo
$gcc -fopenmp -O2 -o reduction-clauseModificado reduction-clauseModificado.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer6] 2020-04-12 domingo
$./reduction-clauseModificado 10
Tras 'parallel' suma=55
```

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: Comienzo el bucle con `i=numerodehebra` (`i` privado para cada hebra) hasta `n`, y con incrementos del número total de hebras, para que el trabajo que realiza cada una sea distribuido.

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
int 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) private(i)
        for (i=omp_get_thread_num(); i<n; i+=omp_get_num_threads()) suma += a[i];

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

CAPTURAS DE PANTALLA:

```
JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer7] 2020-04-12 domingo
gcc -fopenmp -O2 -o reduction-clauseModificado2 reduction-clauseModificado2.c
JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer7] 2020-04-12 domingo
$ ./reduction-clauseModificado2 10
Tras 'parallel' suma=45
```

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):

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

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

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

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

#define GLOBAL
// #define DINAMIC
#ifdef GLOBAL
    #define MAX 33554432
#endif

// comentar para mostrar unicamente primer y ultimo componente del vector resultado v2
#define PRINTF_ALL

int main(int argc, char **argv) {
    //Lectura valores de entrada
    if(argc < 2) {
        fprintf(stderr, "Falta num\n");
        exit(-1);
    }

    unsigned int n = atoi(argv[1]); // Máximo n= 2^32-1= 4294967295 (sizeof(unsigned int) = 4 B)

    int i, j;

    struct timespec ini, fin;
    double transcurrido;

    //Creación e inicialización de vector y matriz
    // Creación
    #ifdef GLOBAL
        if(n > MAX){
            n=MAX;
        }
        int M[n][n];
        int v1[n];
        int v2[n];
        printf("\n\t\tGLOBAL:\n\n");
    #endif
}
```

```

#endif

#ifdef DINAMIC
    int *v1,*v2;
    v1 = (int*) malloc(n*sizeof(int));
    v2 = (int*) malloc(n*sizeof(int));

    int **M;
    M = (int**) malloc(n*sizeof(int*));
    for(i=0;i<n;i++){
        M[i] = (int*)malloc(n*sizeof(int));
        printf("\n\t\tDINAMICO:\n\n");
    }
#endif

// Inicialización
for(i=0;i<n;i++){
    v1[i]=i;
    v2[i]=0;
}
for(i=0;i<n;i++){
    for(j=0;j<n;j++){
        M[i][j]=v1[j]+n*i;
    }
}

//Impresión de vector y matriz iniciales
#ifdef PRINTF_ALL
    printf("Vector inicial:\n");
    for (i=0; i<n; i++) printf("%d ",v1[i]);
    printf("\n");

    printf("Matriz inicial:\n");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            if(M[i][j]<10) printf(" %d ",M[i][j]);
            else printf("%d ",M[i][j]);
        }
        printf("\n");
    }
#endif

//Cálculo resultado en v2 y toma de tiempos
clock_gettime(CLOCK_REALTIME,&ini);
for (i=0; i<n; i++) {
    v2[i]=0;
    for (j=0; j<n; j++) {
        v2[i]+=M[i][j]*v1[j];
    }
}
clock_gettime(CLOCK_REALTIME,&fin);

transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double)
((fin.tv_nsec-ini.tv_nsec)/(1.e+9));

//Impresión del tiempo y vector resultado
#ifdef PRINTF_ALL
    printf("Tamaño: %d \tTiempo: %11.9f\n",n,transcurrido);
    printf("Vector resultado (M x v1):\n");
    for (i=0; i<n; i++) printf("%d ",v2[i]);
    printf("\n");
#else
    printf("Tamaño: %d \tTiempo: %11.9f\n",n,transcurrido);
#endif

```

```

        printf("v2[0]: %d \nv2[n-1]: %d\n", v2[0], v2[n-1]);
    #endif

    //Vaciar memoria
    #ifdef DINAMIC
        for(int i=0; i<n;i++){
            free(M[i]);
        }
        free(M);
        free(v1);
        free(v2);
    #endif

    return(0);
}

```

CAPTURAS DE PANTALLA:

```

[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer8] 2020-04-13 lunes
$gcc -fopenmp -O2 -o pmv-secuencial pmv-secuencial.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer8] 2020-04-13 lunes
$./pmv-secuencial 8

```

```

                DINAMICO:

Vector inicial:
0 1 2 3 4 5 6 7
Matriz inicial:
 0  1  2  3  4  5  6  7
 8  9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63
Tamaño: 8          Tiempo: 0.000000248
Vector resultado (M x v1):
140 364 588 812 1036 1260 1484 1708

```

```

[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer8] 2020-04-13 lunes
$gcc -fopenmp -O2 -o pmv-secuencial pmv-secuencial.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer8] 2020-04-13 lunes
$./pmv-secuencial 8

```

```

                GLOBAL:

Vector inicial:
0 1 2 3 4 5 6 7
Matriz inicial:
 0  1  2  3  4  5  6  7
 8  9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63
Tamaño: 8          Tiempo: 0.000000528
Vector resultado (M x v1):
140 364 588 812 1036 1260 1484 1708

```

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):

- una primera que paralelice el bucle que recorre las filas de la matriz y
- 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`

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

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

//define GLOBAL
#define DINAMIC
#ifdef GLOBAL
    #define MAX 33554432
#endif

//comentar para mostrar unicamente primer y ultimo componente del vector
resultado v2
#define PRINTF_ALL

int main(int argc, char **argv) {
    //Lectura valores de entrada
    if(argc < 2) {
        fprintf(stderr, "Falta num\n");
        exit(-1);
    }

    unsigned int n = atoi(argv[1]); // Máximo n= 2^32-1= 4294967295
    (sizeof(unsigned int) = 4 B)
```



```

int i,j;

struct timespec ini,fin;
double transcurrido;

//Creación e inicialización de vector y matriz
// Creación
#ifdef GLOBAL
    if(n > MAX){
        n=MAX;
    }
    int M[n][n];
    int v1[n];
    int v2[n];
    printf("\n\t\tGLOBAL:\n\n");
#endif

#ifdef DINAMIC
    int *v1,*v2;
    v1 = (int*) malloc(n*sizeof(int));
    v2 = (int*) malloc(n*sizeof(int));

    int **M;
    M = (int**) malloc(n*sizeof(int*));
    for(i=0;i<n;i++)
        M[i] = (int*)malloc(n*sizeof(int));
    printf("\n\t\tDINAMICO:\n\n");
#endif

// Inicialición
//omp_set_num_threads(n);
#pragma omp parallel private(j)
{
    for(i=0;i<n;i++){
        v1[i]=i;
        v2[i]=0;
    }
    #pragma omp for
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            M[i][j]=v1[j]+n*i;
        }
    }
}
//Impresión de vector y matriz iniciales
#ifdef PRINTF_ALL
    printf("Vector inicial:\n");
    for (i=0; i<n; i++) printf("%d ",v1[i]);
    printf("\n");

    printf("Matriz inicial:\n");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            if(M[i][j]<10) printf(" %d ",M[i][j]);
            else printf("%d ",M[i][j]);
        }
    }
}

```

```

        printf("\n");
    }
#endif

//Cálculo resultado en v2 y toma de tiempos

    clock_gettime(CLOCK_REALTIME,&ini);
#pragma omp parallel private(j)
{
    #pragma omp for
    for (i=0; i<n; i++) {
        v2[i]=0;
        for (j=0; j<n; j++) {
            v2[i]+=M[i][j]*v1[j];
        }
    }
}
    clock_gettime(CLOCK_REALTIME,&fin);

    transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

//Impresión del tiempo y vector resultado
#ifdef PRINTF_ALL
    printf("Tamaño: %d \tTiempo: %11.9f\n",n,transcurrido);
    printf("Vector resultado (M x v1):\n");
    for (i=0; i<n; i++) printf("%d ",v2[i]);
    printf("\n");
#else
    printf("Tamaño: %d \tTiempo: %11.9f\n",n,transcurrido);
    printf("v2[0]: %d \nv2[n-1]: %d\n",v2[0],v2[n-1]);
#endif

//Vaciar memoria
#ifdef DINAMIC
    for(int i=0; i<n;i++){
        free(M[i]);
    }
    free(M);
    free(v1);
    free(v2);
#endif

return(0);
}

```

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

```

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

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

```

```

// #define GLOBAL
#define DINAMIC
#ifdef GLOBAL
    #define MAX 33554432
#endif

// comentar para mostrar unicamente primer y ultimo componente del vector resultado v2
#define PRINTF_ALL

int main(int argc, char **argv) {
    // Lectura valores de entrada
    if(argc < 2) {
        fprintf(stderr, "Falta num\n");
        exit(-1);
    }

    unsigned int n = atoi(argv[1]); // Máximo n= 2^32-1= 4294967295 (sizeof(unsigned int) =
4 B)

    int i, j;
    struct timespec ini, fin;
    double transcurrido;

    // Creación e inicialización de vector y matriz
    // Creación
#ifdef GLOBAL
        if(n > MAX){
            n=MAX;
        }
        int M[n][n];
        int v1[n];
        int v2[n];
        printf("\n\t\tGLOBAL:\n\n");
    #endif

    #ifdef DINAMIC
        int *v1, *v2;
        v1 = (int*) malloc(n*sizeof(int));
        v2 = (int*) malloc(n*sizeof(int));

        int **M;
        M = (int**) malloc(n*sizeof(int*));
        for(i=0; i<n; i++){
            M[i] = (int*) malloc(n*sizeof(int));
            printf("\n\t\tDINAMICO:\n\n");
        }
    #endif

    // Inicialización
    // omp_set_num_threads(n);
    #pragma omp parallel private(i)
    {
        for(i=0; i<n; i++){
            v1[i]=i;
            v2[i]=0;
        }

        for(i=0; i<n; i++){
            #pragma omp for
            for(j=0; j<n; j++){
                M[i][j]=v1[j]+n*i;
            }
        }
    }
}

```

```

    }
}
//Impresión de vector y matriz iniciales
#ifdef PRINTF_ALL
    printf("Vector inicial:\n");
    for (i=0; i<n; i++) printf("%d ",v1[i]);
    printf("\n");

    printf("Matriz inicial:\n");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            if(M[i][j]<10) printf(" %d ",M[i][j]);
            else printf("%d ",M[i][j]);
        }
        printf("\n");
    }
#endif

//Cálculo resultado en v2 y toma de tiempos

    clock_gettime(CLOCK_REALTIME,&ini);

    for (i=0; i<n; i++) {
#pragma omp parallel
    {
        int aux=0;
        #pragma omp for
        for (j=0; j<n; j++) {
            aux+=M[i][j]*v1[j];
        }
        #pragma omp critical
        v2[i]+=aux;
    }
    clock_gettime(CLOCK_REALTIME,&fin);

    transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double)
((fin.tv_nsec-ini.tv_nsec)/(1.e+9));

//Impresión del tiempo y vector resultado
#ifdef PRINTF_ALL
    printf("Tamaño: %d \tTiempo: %11.9f\n",n,transcurrido);
    printf("Vector resultado (M x v1):\n");
    for (i=0; i<n; i++) printf("%d ",v2[i]);
    printf("\n");
#else
    printf("Tamaño: %d \tTiempo: %11.9f\n",n,transcurrido);
    printf("v2[0]: %d \nv2[n-1]: %d\n",v2[0],v2[n-1]);
#endif

//Vaciar memoria
#ifdef DINAMIC
    for(int i=0; i<n;i++){
        free(M[i]);
    }
    free(M);
    free(v1);
    free(v2);
#endif

return(0);
}
}

```

RESPUESTA:**CAPTURAS DE PANTALLA:**

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer9] 2020-04-15 miércoles
$ ./pmv-OpenMP-a 8

DINAMICO:

Vector inicial:
0 1 2 3 4 5 6 7
Matriz inicial:
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63
Tamaño: 8      Tiempo: 0.000002346
Vector resultado (M x v1):
140 364 588 812 1036 1260 1484 1708
```

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer9] 2020-04-15 miércoles
$ ./pmv-OpenMP-b 8

DINAMICO:

Vector inicial:
0 1 2 3 4 5 6 7
Matriz inicial:
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63
Tamaño: 8      Tiempo: 0.000076700
Vector resultado (M x v1):
140 364 588 812 1036 1260 1484 1708
```

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`

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

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

```

#endif

//#define GLOBAL
#define DINAMIC
#ifdef GLOBAL
    #define MAX 33554432
#endif

//comentar para mostrar unicamente primer y ultimo componente del vector
resultado v2
#define PRINTF_ALL

int main(int argc, char **argv) {
    //Lectura valores de entrada
    if(argc < 2) {
        fprintf(stderr, "Falta num\n");
        exit(-1);
    }

    unsigned int n = atoi(argv[1]); // Máximo n= 2^32-1= 4294967295
    (sizeof(unsigned int) = 4 B)

    int i,j;

    struct timespec ini,fin;
    double transcurrido;

    //Creación e inicialización de vector y matriz
    // Creación
    #ifdef GLOBAL
        if(n > MAX){
            n=MAX;
        }
        int M[n][n];
        int v1[n];
        int v2[n];
        printf("\n\t\tGLOBAL:\n\n");
    #endif

    #ifdef DINAMIC
        int *v1,*v2;
        v1 = (int*) malloc(n*sizeof(int));
        v2 = (int*) malloc(n*sizeof(int));

        int **M;
        M = (int**) malloc(n*sizeof(int*));
        for(i=0;i<n;i++)
            M[i] = (int*)malloc(n*sizeof(int));
        printf("\n\t\tDINAMICO:\n\n");
    #endif

    // Inicialición
    //omp_set_num_threads(n);

    for(i=0;i<n;i++){

```



```

        v1[i]=i;
        v2[i]=0;
    }

    for(i=0;i<n;i++){
        #pragma omp parallel for
        for(j=0;j<n;j++){
            M[i][j]=v1[j]+n*i;
        }
    }

//Impresión de vector y matriz iniciales
#ifdef PRINTF_ALL
    printf("Vector inicial:\n");
    for (i=0; i<n; i++) printf("%d ",v1[i]);
    printf("\n");

    printf("Matriz inicial:\n");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            if(M[i][j]<10) printf(" %d ",M[i][j]);
            else printf("%d ",M[i][j]);
        }
        printf("\n");
    }
#endif

//Cálculo resultado en v2 y toma de tiempos

    clock_gettime(CLOCK_REALTIME,&ini);

    int aux=0;
    #pragma omp parallel for reduction(+:aux)
    for (j=0; j<n; j++) {
        aux+=M[i][j]*v1[j];
    }
    #pragma omp critical
    v2[i]+=aux;
}
clock_gettime(CLOCK_REALTIME,&fin);

    transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

//Impresión del tiempo y vector resultado
#ifdef PRINTF_ALL
    printf("Tamaño: %d \tTiempo: %11.9f\n",n,transcurrido);
    printf("Vector resultado (M x v1):\n");
    for (i=0; i<n; i++) printf("%d ",v2[i]);
    printf("\n");
#else
    printf("Tamaño: %d \tTiempo: %11.9f\n",n,transcurrido);
    printf("v2[0]: %d \nv2[n-1]: %d\n",v2[0],v2[n-1]);
#endif

```

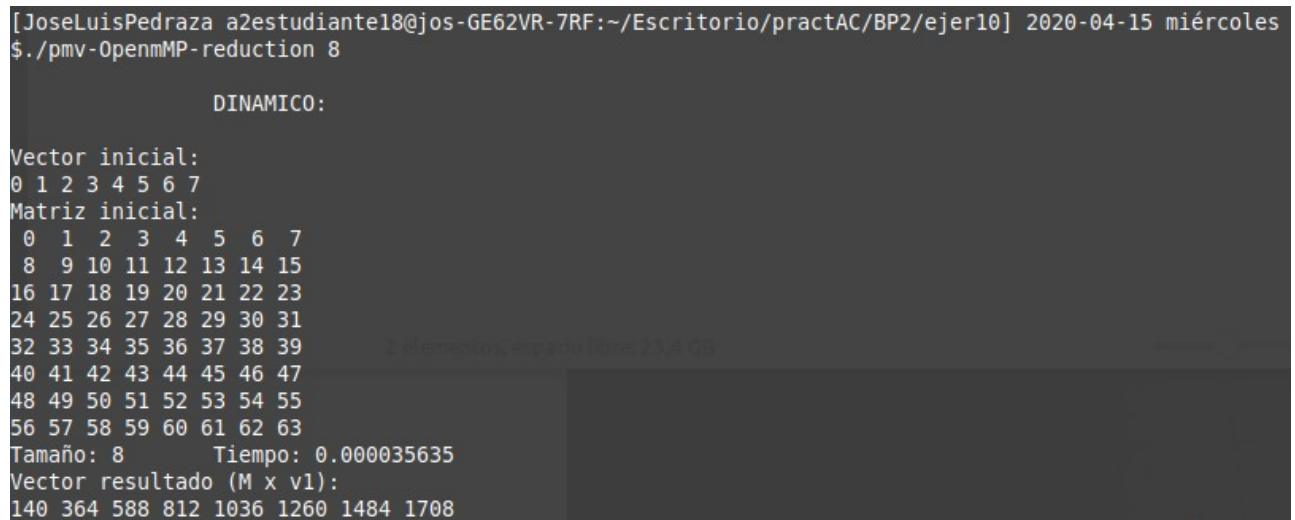
```

    for (i=0; i<n; i++) {
    //Vaciar memoria
#ifdef DINAMIC
        for(int i=0; i<n;i++){
            free(M[i]);
        }
        free(M);
        free(v1);
        free(v2);
    #endif
return(0);
}

```

RESPUESTA:

CAPTURAS DE PANTALLA:



```

[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer10] 2020-04-15 miércoles
$ ./pmv-OpenMP-reduction 8

          DINAMICO:

Vector inicial:
0 1 2 3 4 5 6 7
Matriz inicial:
 0  1  2  3  4  5  6  7
 8  9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63
Tamaño: 8      Tiempo: 0.000035635
Vector resultado (M x v1):
140 364 588 812 1036 1260 1484 1708

```

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.

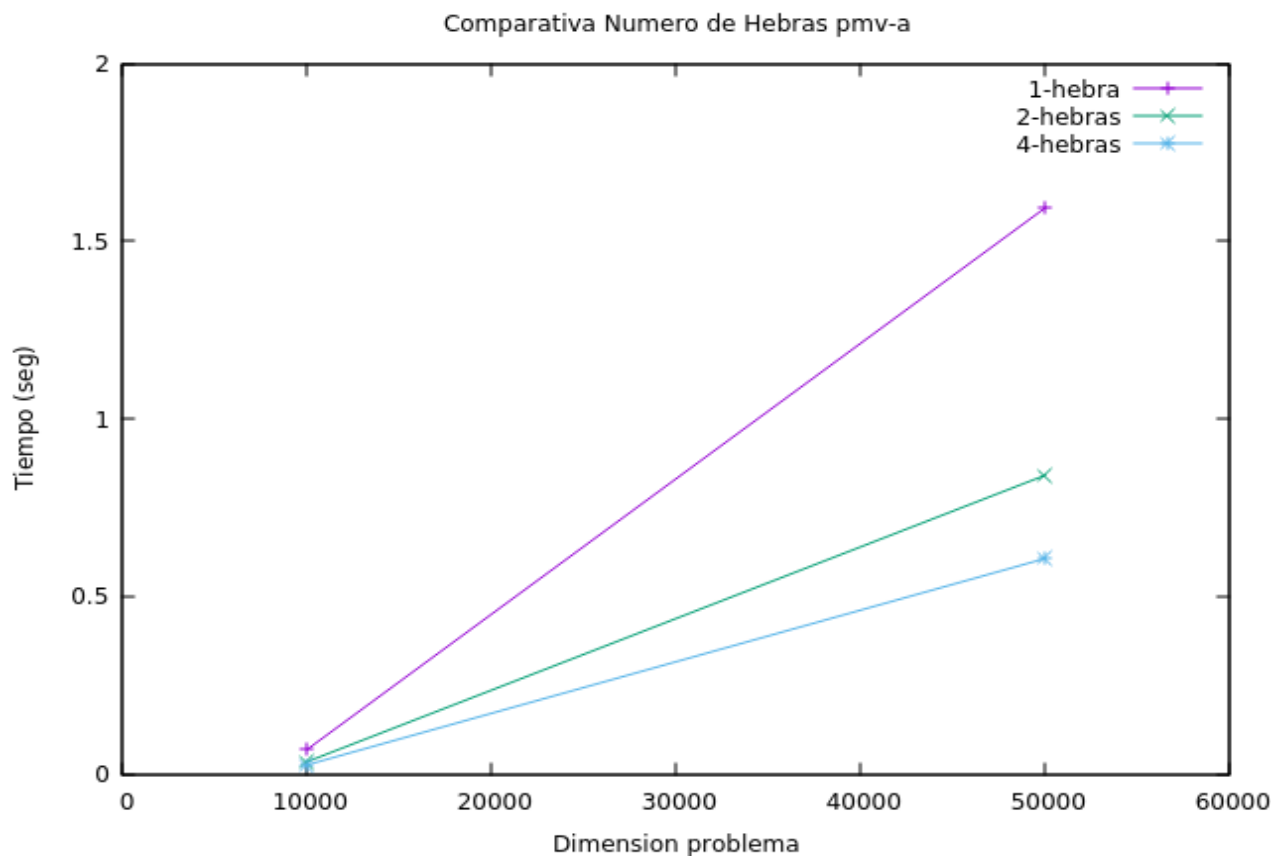
CAPTURAS DE PANTALLA (que justifique el código elegido): Escojo el código pmv-a dado que al paralelizar la matriz por filas obtenemos mejores resultados en comparación con el b y el reduction, ya que estos paralelizan por columnas y necesitan de una variable auxiliar para realizar los cálculos correctamente en una sección crítica.

```
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer11/PC] 2020-04-15 miércoles
$gcc -fopenmp -O2 -o pmv-OpenMP-a pmv-OpenMP-a.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer11/PC] 2020-04-15 miércoles
$gcc -fopenmp -O2 -o pmv-OpenMP-b pmv-OpenMP-b.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer11/PC] 2020-04-15 miércoles
$gcc -fopenmp -O2 -o pmv-OpenMP-reduction pmv-OpenMP-reduction.c
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer11/PC] 2020-04-15 miércoles
$./pmv-OpenMP-a 50000; ./pmv-OpenMP-b 50000; ./pmv-OpenMP-reduction 50000
Tamaño: 50000 Tiempo: 0.601060301
Tamaño: 50000 Tiempo: 0.746300653
Tamaño: 50000 Tiempo: 0.679977930
[JoseLuisPedraza a2estudiante18@jos-GE62VR-7RF:~/Escritorio/practAC/BP2/ejer11/PC] 2020-04-15 miércoles
$
```

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):

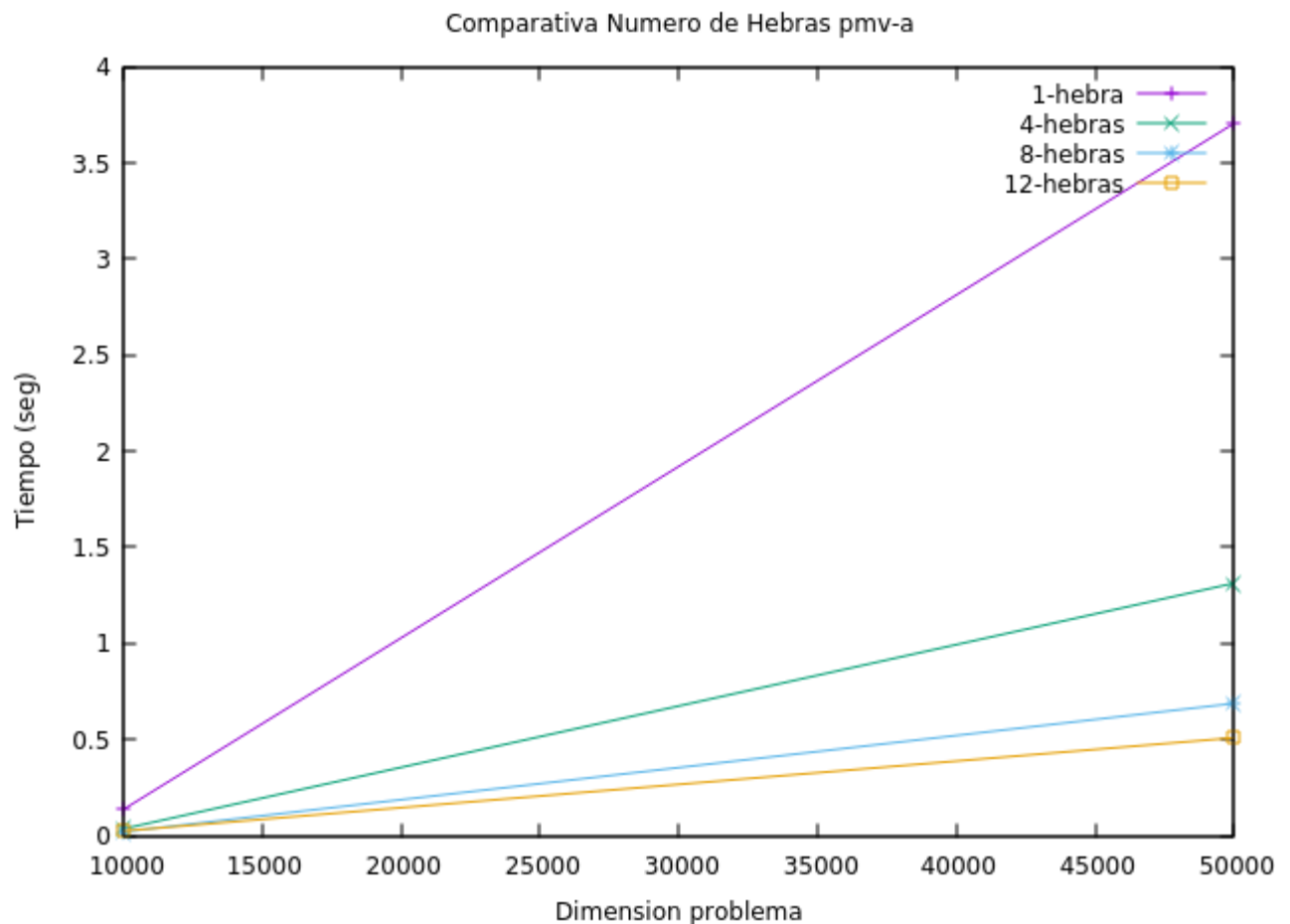
- MIPC: pmv-a:

Hebras	Tamaño N= 10000	Tamaño N= 50000
1	0.067790452	1.594270023
2	0.033407935	0.840847549
4	0.024630695	0.606247814



- ATCGRID: pmv-a

Hebras	Tamaño N= 10000	Tamaño N= 50000
1	0.134949141	3.705145153
4	0.035448716	1.313051625
8	0.020109044	0.687105942
12	0.023874660	0.509043195



COMENTARIOS SOBRE LOS RESULTADOS:

En principio, los tiempos para estos dos valores tomados son mejores en mi PC que en ATCGRID. Y como se aprecia claramente, el tiempo disminuye a medida que aumentamos el número de hebras, al menos hasta un N= 50000, tanto en mi PC como en ATCGRID, lo cual es lógico, ya que en el código paralelizamos ciertas partes y cuantas más hebras, menor carga de trabajo de nuestro código tendrán que realizar cada una, disminuyendo así el tiempo total.