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): Alejandro Molina Criado
Grupo de prácticas y profesor de prácticas: A2
Fecha de entrega: 15/04/2020
Fecha evaluación en clase: 17/04/2020

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:

```
alex@alex-CX62-6QD:~/practicas_AC/bp2/ej1$ gcc -O2 -fopenmp -o sharedModificado sharedModifica
do.c
sharedModificado.c: In function ‘main’:
sharedModificado.c:26:13: error: ‘n’ not specified in enclosing ‘parallel’
     #pragma omp parallel for shared(a) , default(none)
             ^~~
sharedModificado.c:26:13: error: enclosing ‘parallel’
```

Nos resulta un error , esto es debido a que default especifica el comportamiento de las variables sin ámbito en una región paralela . Si añadimos esta cláusula a la región parallel , tenemos que especificar en la cláusula shared todas las variables que van a compartirse dentro de la región paralela , en este caso la que nos falta es la variable n.

CAPTURAS DE PANTALLA:

```
sudo apt install default-jre
```

```
int main(){

    int i , n = 8 ,
        a[n];

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

    #pragma omp parallel for shared(a,n) , default(none)
    for(i=0;i<n;++i){
        a[i]+=i; //[1+0,2+1,3+2,....]
    }

    printf("Fin de la sección parallel for:\n");

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

    return 0;
}
```

2. Añadir a lo necesario a `private-clause.c` para que imprima suma fuera de la región `parallel` e inicializar `suma` a un valor distinto de 0. Ejecute varias veces el código ¿Qué imprime el código fuera del parallel? (muéstrelo con una captura de pantalla) ¿Qué ocurre si en esta versión de `private-clause.c` se inicia la variable `suma` fuera de la construcción `parallel` en lugar de dentro? Razone su respuesta (añada capturas de pantalla que muestren lo que ocurre). Añadir el código con las modificaciones al cuaderno de prácticas.

RESPUESTA PREGUNTA 1:

La cláusula private implica que cada subproceso tiene **su propia instancia** de la variable suma , si suma está dentro de la región parallel , todos los threads inicializará su instancia de suma a 8.
Para la región externa al parallel , la variable suma no está inicializada , por lo que su valor es 0.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`

```
33  int main(){
34      int i , n = 7,
35          a[n],
36          suma;
37
38      for(i = 0 ; i < n ; ++i) a[i] = i;
39
40      #pragma omp parallel private(suma)
41      {
42          suma = 8;
43          #pragma omp for
44          for(i = 0 ; i < n ; ++i){
45              suma = suma + a[i];
46              printf("thread %d suma a[%d] (suma = %d)\n",omp_get_thread_num() , i , suma);
47          }
48
49          printf("\n* thread %d suma=%d",omp_get_thread_num(),suma);
50
51      }
52
53
54      printf("\n---FUERA DE LA REGIÓN PARALLEL---");
55      printf("\nSuma = %d",suma);
56
57      printf("\n");
58
59      return 0;
60  }
```

CAPTURAS DE PANTALLA (compilación y ejecución):

```
alex@DESKTOP-KB48M7J:~/practicas_AC/bp2/ej2$ gcc -O2 -fopenmp -o privateModificado privateModificado.c
alex@DESKTOP-KB48M7J:~/practicas_AC/bp2/ej2$ ./privateModificado
thread 3 suma a[6] (suma = 14)
thread 2 suma a[4] (suma = 12)
thread 2 suma a[5] (suma = 17)
thread 1 suma a[2] (suma = 10)
thread 1 suma a[3] (suma = 13)
thread 0 suma a[0] (suma = 8)
thread 0 suma a[1] (suma = 9)

* thread 3 suma=14
* thread 1 suma=13
* thread 0 suma=9
* thread 2 suma=17
---FUERA DE LA REGIÓN PARALLEL---
Suma = 0
alex@DESKTOP-KB48M7J:~/practicas_AC/bp2/ej2$ ./privateModificado
thread 2 suma a[4] (suma = 12)
thread 2 suma a[5] (suma = 17)
thread 3 suma a[6] (suma = 14)
thread 1 suma a[2] (suma = 10)
thread 1 suma a[3] (suma = 13)
thread 0 suma a[0] (suma = 8)
thread 0 suma a[1] (suma = 9)

* thread 2 suma=17
* thread 3 suma=14
* thread 1 suma=13
* thread 0 suma=9
---FUERA DE LA REGIÓN PARALLEL---
Suma = 0
```

RESPUESTA PREGUNTA 2:

Si inicializamos la variable suma fuera , veremos como las distintas instancias de suma dentro de la región parallel toman valores indeterminados , esto es porque la cláusula private tiene que asignar un valor a cada instancia , valor que no hemos indicado.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`

```c
int main(){
    int i , n = 7,
        a[n],
        suma = 8; //<-- Inicializamos suma fuera de la región parallel

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


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

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

    }


    printf("\n---FUERA DE LA REGIÓN PARALLEL---");
    printf("\nSuma = %d",suma);

    printf("\n");

    return 0;
}
```

CAPTURAS DE PANTALLA (compilación y ejecución):

```
alex@DESKTOP-KB48M7J:~/practicas_AC/bp2/ej2$ gcc -O2 -fopenmp -o privateModificado privateModificado.c
alex@DESKTOP-KB48M7J:~/practicas_AC/bp2/ej2$ ./privateModificado
thread 2 suma a[4] (suma = -812637356)
thread 2 suma a[5] (suma = -812637351)
thread 1 suma a[2] (suma = -812637358)
thread 1 suma a[3] (suma = -812637355)
thread 0 suma a[0] (suma = -693879168)
thread 0 suma a[1] (suma = -693879167)
thread 3 suma a[6] (suma = -812637354)

* thread 1 suma=-812637355
* thread 0 suma=-693879167
* thread 2 suma=-812637351
* thread 3 suma=-812637354
---FUERA DE LA REGIÓN PARALLEL---
Suma = 8
alex@DESKTOP-KB48M7J:~/practicas_AC/bp2/ej2$ ./privateModificado
thread 1 suma a[2] (suma = -1033559214)
thread 1 suma a[3] (suma = -1033559211)
thread 3 suma a[6] (suma = -1033559210)
thread 2 suma a[4] (suma = -1033559212)
thread 2 suma a[5] (suma = -1033559207)
thread 0 suma a[0] (suma = -897086448)
thread 0 suma a[1] (suma = -897086447)

* thread 2 suma=-1033559207
* thread 3 suma=-1033559210
* thread 1 suma=-1033559211
* thread 0 suma=-897086447
---FUERA DE LA REGIÓN PARALLEL---
Suma = 8
```

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

RESPUESTA:

En el momento en el que eliminamos la cláusula private , la variable suma pasa a ser compartida por todos los procesos , en cada thread suma primero se inicializa a cero por lo que si un thread anterior había modificado el valor de suma este se pierde , dando un resultado erróneo.

Debido a esto , el valor de suma en todos los threads al final de la sección parrallel for es el del último thread ejecutado.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

```c
int main(){
    int i , n = 7,
        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)\n" , omp_get_thread_num() , suma);
            printf("thread %d suma a[%d]\n",omp_get_thread_num() , i);
        }

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

    }

    printf("\n");

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
alex@DESKTOP-KB48M7J:~/practicas_AC/bp2/ej3$ gcc -O2 -fopenmp -o privateModificado privateModificado.c
alex@DESKTOP-KB48M7J:~/practicas_AC/bp2/ej3$ ./privateModificado
(thread 1 suma = 2)
thread 1 suma a[2]
(thread 1 suma = 3)
thread 1 suma a[3]
(thread 2 suma = 4)
thread 2 suma a[4]
(thread 2 suma = 8)
thread 2 suma a[5]
(thread 0 suma = 0)
thread 0 suma a[0]
(thread 0 suma = 9)
thread 0 suma a[1]
(thread 3 suma = 6)
thread 3 suma a[6]

* thread 2 suma=9
* thread 0 suma=9
* thread 1 suma=9
* thread 3 suma=9
```

```
alex@DESKTOP-KB48M7J:~/practicas_AC/bp2/ej3$ ./privateModificado
(thread 1 suma = 2)
thread 1 suma a[2]
(thread 1 suma = 7)
thread 1 suma a[3]
(thread 0 suma = 0)
thread 0 suma a[0]
(thread 0 suma = 8)
thread 0 suma a[1]
(thread 3 suma = 6)
thread 3 suma a[6]
(thread 2 suma = 4)
thread 2 suma a[4]
(thread 2 suma = 13)
thread 2 suma a[5]

* thread 1 suma=13
* thread 3 suma=13
* thread 0 suma=13
* thread 2 suma=13
```

Por ejemplo , en la primera ejecución el último thread en ejecutarse ha sido el 0 (ya que suma al final es 9) , y en la segunda ejecución , el último thread en ejecutarse es el 2 (ya que suma al final es 13)

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: Con un número de threads mayor o igual a 4 , la suma siempre da como resultado 6. Esto es debido a que lastprivate obtiene el valor que toma la variable en la última iteración del bucle for (no confundir con el último thread ejecutado)

   Si el número threads es menor a 4 , observaremos como el resultado de la suma ya no es 6. Esto es debido a que la directiva for le asigna más iteraciones a menos threads , por lo que un mismo thread realizará más veces la operación suma.

   En las capturas de a continuación se demuestra lo desarrollado anteriormente.

   CAPTURAS DE PANTALLA:

```
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej4$ gcc -O2 -fopenmp -o firstlastprivate firstlastprivate.c
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej4$ export OMP_NUM_THREADS=4
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej4$ ./firstlastprivate
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9
thread 3 suma a[6] suma=6
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5

fuera de la seccion parallel suma=6
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej4$ ./firstlastprivate
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 3 suma a[6] suma=6

fuera de la seccion parallel suma=6
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej4$ export OMP_NUM_THREADS=3
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej4$ ./firstlastprivate
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[5] suma=5
thread 2 suma a[6] suma=11
thread 1 suma a[3] suma=3
thread 1 suma a[4] suma=7

fuera de la seccion parallel suma=11
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej4$
```

Con 4 threads la suma da como resultado 6, mientras que con 3 da 11.

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: Si eliminamos la cláusula copyprivate los threads ya no compartirán el valor de la variable a introducida por teclado , por lo que el valor de dicha variable en los threads será indefinido (mi compilador por defecto los pone a 0)

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```c
int main(){
    int n = 9 ,
        i,
        b[n];
    for(i=0;i<n;i++) b[i] = -1;
    printf("\nAntes de la sección parallel:\n");
    for(i=0;i<n;++i) printf("b[%d]=%d || ",i,b[i]);

    #pragma omp parallel
    {
        int a ;

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

        #pragma omp for
        for(i=0;i<n;i++) b[i] = a;
    }
    printf("\nDespues de la sección parallel:\n");
    for(i=0;i<n;++i) printf("b[%d]=%d || ",i,b[i]);
    printf("\n");
    return 0;
}
```

CAPTURAS DE PANTALLA:

```
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej5$ gcc -O2 -fopenmp -o copyprivateModificado copyprivateModificado.c
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej5$ ./copyprivateModificado

Antes de la sección parallel:
b[0]=-1 || b[1]=-1 || b[2]=-1 || b[3]=-1 || b[4]=-1 || b[5]=-1 || b[6]=-1 || b[7]=-1 || b[8]=-1 ||
Introduce el valor de inicializacion a :8

Sección sigle ejecutada por el thread 1
Despues de la sección parallel:
b[0]=0 || b[1]=0 || b[2]=0 || b[3]=8 || b[4]=8 || b[5]=0 || b[6]=0 || b[7]=0 || b[8]=0 ||
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej5$ ./copyprivateModificado

Antes de la sección parallel:
b[0]=-1 || b[1]=-1 || b[2]=-1 || b[3]=-1 || b[4]=-1 || b[5]=-1 || b[6]=-1 || b[7]=-1 || b[8]=-1 ||
Introduce el valor de inicializacion a :10

Sección sigle ejecutada por el thread 1
Despues de la sección parallel:
b[0]=0 || b[1]=0 || b[2]=0 || b[3]=10 || b[4]=10 || b[5]=0 || b[6]=0 || b[7]=0 || b[8]=0 ||
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej5$
```

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: Al inicializar suma a 10 , al resultado final se le añade este valor.
Las caputas de a continuación muestran la impresión antes de la modificación y después.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

```c
int main(int argc , char **argv){
    int i,
        n = 20,
        a[n],
        suma = 10;
    if (argc < 2){
        fprintf(stderr,"ERROR: 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;

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

    printf("\nFinal de la sección parralel , suma = %d\n" , suma);

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej6$ gcc -O2 -fopenmp -o reduction reduction.c
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej6$ ./reduction 20

Final de la sección parralel , suma = 190
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej6$ gcc -O2 -fopenmp -o reductionModificado reductionModificado.c
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej6$ ./reductionModificado 20

Final de la sección parralel , suma = 200
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej6$
```

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: Si eliminamos reduction y queremos seguir obteniendo la suma correcta con una ejecución en paralelo , tenemos que asegurar la exclusión mutua , algo que la cláusula atomic nos asegura.

CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado7.c`

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

    #pragma omp parallel for
    for(i=0;i<n;++i){
        #pragma omp atomic
        suma+=a[i];
    }

    printf("\nFinal de la sección parralel , suma = %d\n" , suma);

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej7$ gcc -o reductionModificado reductionModificado.c
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej7$ ./reductionModificado 10

Final de la sección parralel , suma = 45
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej7$ ./reductionModificado 20

Final de la sección parralel , suma = 190
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej7$ ./reductionModificado 5

Final de la sección parralel , suma = 10
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej7$
```

Resto de ejercicios

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

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

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

CÓDIGO FUENTE: `pmv-secuencial.c`

```
/*
   Nombre: Alejandro Molina Criado
   Fecha: 01 / 04 / 2020

   para compilar: gcc -O2 -fopenmp -o pmv-secuencial pmv-secuencial.c
*/
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

//#define MATRIX_GLOBAL
#define MATRIX_DYNAMIC

#ifdef MATRIX_GLOBAL
#define MAX 20000

double m[MAX][MAX], v1[MAX], vf[MAX];
#endif
int main(int argc, char** argv)
{
        if (argc<2)
        {
        printf("ERROR: Introduce nº columnas\n");
        exit(-1);
        }

        int N = atoi(argv[1]);

        #ifdef MATRIX_GLOBAL
           if (N>MAX) N=MAX;
        #endif
        #ifdef MATRIX_DYNAMIC
           double *v1, *vf;
           double **m;

        v1 = (double*) malloc(N * sizeof(double));
        vf = (double*) malloc(N * sizeof(double));
        m = (double**) malloc(N * sizeof(double*));
```

```
	for (int i=0; i<N; i++)
	m[i] = (double*)malloc(N * sizeof(double));

	if ((v1 == NULL) || (vf == NULL) || (m == NULL))
	{
	printf("No hay suficiente espacio para reservar memoria \n");
	exit(-2);
	}
	#endif

	struct timespec cgt1,cgt2;
	double ncgt;

	printf("Tamaño Matriz:%u x %u (%lu B)\n", N, N, sizeof(unsigned int));


	for (int i=0; i<N; ++i){
	v1[i] = N*0.1+i*0.1;
	for (int j=0; j<N; ++j)
		m[i][j] = N*0.1+i*0.1;
	}

	//Multiplicacion
	clock_gettime(CLOCK_REALTIME,&cgt1);

	for (int i=0; i<N; ++i)
	for (int j=0; j<N; ++j)
		vf[i] += m[i][j] * v1[j];

	clock_gettime(CLOCK_REALTIME,&cgt2);

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

	if (N<12)
	{
	printf("Tiempo:%11.9f\t / Tamaño columnas:%u\n",ncgt,N);
	for (int i=0; i<N; ++i)
	{
		printf("/ ");
		for (int j=0; j<N-1; ++j)
			printf("(%8f*%8f)+", m[i][j], v1[j]);
		printf("(%8f*%8f) =", m[i][N-1], v1[N-1]);
		printf(" V2[%d] = %8f /\n", i, vf[i]);
	}
	}
	else
	printf("Tiempo:%11.9f\t / Tamaño columnas:%u\t/ V2[0] = %8.6f / / V2[%d] = %8.6f /\n", ncgt, N, vf[0], N-1, vf[N-1]);
	#ifdef MATRIX_DYNAMIC
	for(int i = 0; i < N; ++i)
		free(m[i]);
	free(m);
	free(v1);
	free(vf);
	#endif
	return 0;
}
```

CAPTURAS DE PANTALLA:

```
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej8$ gcc -O2 -fopenmp -o pmv-secuencial pmv-secuencial.c
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej8$ ./pmv-secuencial 10
Tamaño Matriz:10 x 10 (4 B)
Tiempo:0.000000764       / Tamaño columnas:10
/ (1.000000*1.000000)+(1.000000*1.100000)+(1.000000*1.200000)+(1.000000*1.300000)+(1.000000*1.400000)+(1.000000*1.500000)+(1.000000*1.600000)+(1.000000*1.700000)+(1.000000*1.800000)+(1.000000*1.900000) = V2[0] = 14.500000 /
/ (1.100000*1.000000)+(1.100000*1.100000)+(1.100000*1.200000)+(1.100000*1.300000)+(1.100000*1.400000)+(1.100000*1.500000)+(1.100000*1.600000)+(1.100000*1.700000)+(1.100000*1.800000)+(1.100000*1.900000) = V2[1] = 15.950000 /
/ (1.200000*1.000000)+(1.200000*1.100000)+(1.200000*1.200000)+(1.200000*1.300000)+(1.200000*1.400000)+(1.200000*1.500000)+(1.200000*1.600000)+(1.200000*1.700000)+(1.200000*1.800000)+(1.200000*1.900000) = V2[2] = 17.400000 /
/ (1.300000*1.000000)+(1.300000*1.100000)+(1.300000*1.200000)+(1.300000*1.300000)+(1.300000*1.400000)+(1.300000*1.500000)+(1.300000*1.600000)+(1.300000*1.700000)+(1.300000*1.800000)+(1.300000*1.900000) = V2[3] = 18.850000 /
/ (1.400000*1.000000)+(1.400000*1.100000)+(1.400000*1.200000)+(1.400000*1.300000)+(1.400000*1.400000)+(1.400000*1.500000)+(1.400000*1.600000)+(1.400000*1.700000)+(1.400000*1.800000)+(1.400000*1.900000) = V2[4] = 20.300000 /
/ (1.500000*1.000000)+(1.500000*1.100000)+(1.500000*1.200000)+(1.500000*1.300000)+(1.500000*1.400000)+(1.500000*1.500000)+(1.500000*1.600000)+(1.500000*1.700000)+(1.500000*1.800000)+(1.500000*1.900000) = V2[5] = 21.750000 /
/ (1.600000*1.000000)+(1.600000*1.100000)+(1.600000*1.200000)+(1.600000*1.300000)+(1.600000*1.400000)+(1.600000*1.500000)+(1.600000*1.600000)+(1.600000*1.700000)+(1.600000*1.800000)+(1.600000*1.900000) = V2[6] = 23.200000 /
/ (1.700000*1.000000)+(1.700000*1.100000)+(1.700000*1.200000)+(1.700000*1.300000)+(1.700000*1.400000)+(1.700000*1.500000)+(1.700000*1.600000)+(1.700000*1.700000)+(1.700000*1.800000)+(1.700000*1.900000) = V2[7] = 24.650000 /
/ (1.800000*1.000000)+(1.800000*1.100000)+(1.800000*1.200000)+(1.800000*1.300000)+(1.800000*1.400000)+(1.800000*1.500000)+(1.800000*1.600000)+(1.800000*1.700000)+(1.800000*1.800000)+(1.800000*1.900000) = V2[8] = 26.100000 /
/ (1.900000*1.000000)+(1.900000*1.100000)+(1.900000*1.200000)+(1.900000*1.300000)+(1.900000*1.400000)+(1.900000*1.500000)+(1.900000*1.600000)+(1.900000*1.700000)+(1.900000*1.800000)+(1.900000*1.900000) = V2[9] = 27.550000 /
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej8$
```

```
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej8$ ./pmv-secuencial 8
Tamaño Matriz:8 x 8 (4 B)
Tiempo:0.000000693       / Tamaño columnas:8
/ (0.800000*0.800000)+(0.800000*0.900000)+(0.800000*1.000000)+(0.800000*1.100000)+(0.800000*1.200000)+(0.800000*1.300000)+(0.800000*1.400000)+(0.800000*1.500000) = V2[0] = 7.360000 /
/ (0.900000*0.800000)+(0.900000*0.900000)+(0.900000*1.000000)+(0.900000*1.100000)+(0.900000*1.200000)+(0.900000*1.300000)+(0.900000*1.400000)+(0.900000*1.500000) = V2[1] = 8.280000 /
/ (1.000000*0.800000)+(1.000000*0.900000)+(1.000000*1.000000)+(1.000000*1.100000)+(1.000000*1.200000)+(1.000000*1.300000)+(1.000000*1.400000)+(1.000000*1.500000) = V2[2] = 9.200000 /
/ (1.100000*0.800000)+(1.100000*0.900000)+(1.100000*1.000000)+(1.100000*1.100000)+(1.100000*1.200000)+(1.100000*1.300000)+(1.100000*1.400000)+(1.100000*1.500000) = V2[3] = 10.120000 /
/ (1.200000*0.800000)+(1.200000*0.900000)+(1.200000*1.000000)+(1.200000*1.100000)+(1.200000*1.200000)+(1.200000*1.300000)+(1.200000*1.400000)+(1.200000*1.500000) = V2[4] = 11.040000 /
/ (1.300000*0.800000)+(1.300000*0.900000)+(1.300000*1.000000)+(1.300000*1.100000)+(1.300000*1.200000)+(1.300000*1.300000)+(1.300000*1.400000)+(1.300000*1.500000) = V2[5] = 11.960000 /
/ (1.400000*0.800000)+(1.400000*0.900000)+(1.400000*1.000000)+(1.400000*1.100000)+(1.400000*1.200000)+(1.400000*1.300000)+(1.400000*1.400000)+(1.400000*1.500000) = V2[6] = 12.880000 /
/ (1.500000*0.800000)+(1.500000*0.900000)+(1.500000*1.000000)+(1.500000*1.100000)+(1.500000*1.200000)+(1.500000*1.300000)+(1.500000*1.400000)+(1.500000*1.500000) = V2[7] = 13.800000 /
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej8$
```

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).

CÓDIGO FUENTE : `pmv-OpenMP-a.c`

```
/*
   Nombre: Alejandro Molina Criado
   Fecha: 01 / 04 / 2020

   para compilar: gcc -O2 -fopenmp -o pmv-secuencial pmv-secuencial.c
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <malloc.h>
#include <omp.h>

#define GLOBAL

#ifdef GLOBAL
        #define MAX 33554432
#endif

int main(int argc, char const *argv[]){
        if(argc != 2){
      printf("Error de argumentos %s", argv[0]);
      return(EXIT_FAILURE);
   }

   struct timespec cgt1, cgt2;
   double ncgt;

   int N = atoi(argv[1]);

   #ifdef GLOBAL
        if(N > MAX) N = MAX;
      int matriz[N][N];
      int vector[N];
      int vector_resultado[N];
      printf("Ejecutado de forma global\n");
   #endif

   #ifdef DINAMIC
      int **matriz, *vector, *vector_resultado;
      matriz = (int**) malloc(N * sizeof(int*));
      for(int i = 0; i < N; ++i)
         matriz[i] = (int*) malloc(N * sizeof(int));

      vector = (int*) malloc(N * sizeof(int));
      vector_resultado = (int*) malloc(N * sizeof(int));
      printf("Ejecutado de forma dinámica\n");
   #endif

        #pragma omp parallel for
        for(int i = 0; i < N; ++i){
        vector[i] = i;
        #pragma omp parallel for
        for(int j = 0; j < N; ++j)
        matriz[i][j] = i + j;
        }
```

```
        clock_gettime(CLOCK_REALTIME, &cgt1);
        //Calculamos el vector resultado
        #pragma omp parallel for
        for(int i = 0; i < N; i++){
        int suma = 0;
        for(int j = 0; j < N; j++)
                suma += matriz[i][j] * vector[j];

        vector_resultado[i] = suma;
        }

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

        printf("Tiempo: %11.9f seg.\t / Tamaño vectores: %u\n", ncgt, N);
        if(N < 15)
        for(int i = 0; i < N; i++){
                printf("V_RESULTADO[%d] = %d  ", i, vector_resultado[i]);
                printf("\n");
        }
        else{
        printf("V_RESULTADO[0] = %d  ", vector_resultado[0]);
        printf("V_RESULTADO[%d] = %d  ", N - 1, vector_resultado[N - 1]);
        printf("\n");
        }

        #ifdef DINAMIC
        for(int i = 0; i < N; i++)
        free(matriz[i]);

        //liberamos espacio
        free(matriz); free(vector); free(vector_resultado);
        #endif

    return 0;
}
```

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

```
/*
   Nombre: Alejandro Molina Criado
   Fecha: 01 / 04 / 2020

   para compilar: gcc -O2 -fopenmp -o pmv-secuencial pmv-secuencial.c
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <malloc.h>

//#define GLOBAL
#define DINAMIC

#ifdef GLOBAL
        #define MAX 33554432
#endif

int main(int argc, char const *argv[]){
        if(argc != 2){
    printf("Error de argumentos %s", argv[0]);
    return(EXIT_FAILURE);
  }

  struct timespec cgt1, cgt2;
  double ncgt;

  int N = atoi(argv[1]);

  #ifdef GLOBAL
        if(N > MAX) N = MAX;
    int matriz[N][N];
    int vector[N];
    int vector_resultado[N];
    printf("Ejecutado GLOBALMENTE\n");
  #endif

  #ifdef DINAMIC
    int **matriz, *vector, *vector_resultado;
    matriz = (int**) malloc(N * sizeof(int*));
    for(int i = 0; i < N; ++i)
       matriz[i] = (int*) malloc(N * sizeof(int));

    vector = (int*) malloc(N * sizeof(int));
    vector_resultado = (int*) malloc(N * sizeof(int));
    printf("Ejecutado DINAMICAMENTE\n");
  #endif

  #pragma omp parallel for
  for(int i = 0; i < N; ++i){
        vector[i] = i;
        #pragma omp parallel for
    for(int j = 0; j < N; ++j)
        matriz[i][j] = i + j;
        }
```

```
        clock_gettime(CLOCK_REALTIME, &cgt1);
        //Calculamos el vector resultado
        for(int i = 0; i < N; i++){
        int suma = 0;
        #pragma omp parallel for
        for(int j = 0; j < N; j++)
                #pragma omp atomic
                suma += matriz[i][j] * vector[j];

        vector_resultado[i] = suma;
        }

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

        printf("Tiempo: %11.9f seg.\t / Tamaño vectores: %u\n", ncgt, N);
        if(N < 15)
        for(int i = 0; i < N; i++){
                printf("VRESULTADO[%d] = %d  ", i, vector_resultado[i]);
                printf("\n");
        }
        else{
        printf("V_RESULTADO[0] = %d  ", vector_resultado[0]);
        printf("V_RESULTADO[%d] = %d  ", N - 1, vector_resultado[N - 1]);
        printf("\n");
        }

        #ifdef DINAMIC
        for(int i = 0; i < N; i++)
        free(matriz[i]);

    //Liberamos espacio
        free(matriz); free(vector); free(vector_resultado);
        #endif

    return 0;
}
```

RESPUESTA: No he tenido problemas en la compilación ni en el resultado esperado.

CAPTURAS DE PANTALLA:

`pmv-OpenMP-a.c`

```
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej9$ gcc -O2 -fopenmp -o pmv-OpenMP-a pmv-OpenMP-a.c
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej9$ ./pmv-OpenMP-a 10
Ejecutado de forma global
Tiempo(seg.): 0.000007416        / Tamaño vectores: 10
V_RESULTADO[0] = 285
V_RESULTADO[1] = 330
V_RESULTADO[2] = 375
V_RESULTADO[3] = 420
V_RESULTADO[4] = 465
V_RESULTADO[5] = 510
V_RESULTADO[6] = 555
V_RESULTADO[7] = 600
V_RESULTADO[8] = 645
V_RESULTADO[9] = 690
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej9$ ./pmv-OpenMP-a 5
Ejecutado de forma global
Tiempo(seg.): 0.000007391        / Tamaño vectores: 5
V_RESULTADO[0] = 30
V_RESULTADO[1] = 40
V_RESULTADO[2] = 50
V_RESULTADO[3] = 60
V_RESULTADO[4] = 70
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej9$
```

`pmv-OpenMP-b.c`

```
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej9$ gcc -O2 -fopenmp -o pmv-OpenMP-b pmv-OpenMP-b.c
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej9$ ./pmv-OpenMP-b 5
Ejecutado DINAMICAMENTE
Tiempo: 0.000032561 seg.         / Tamaño vectores: 5
VRESULTADO[0] = 30
VRESULTADO[1] = 40
VRESULTADO[2] = 50
VRESULTADO[3] = 60
VRESULTADO[4] = 70
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej9$ ./pmv-OpenMP-b 10
Ejecutado DINAMICAMENTE
Tiempo: 0.000062747 seg.         / Tamaño vectores: 10
VRESULTADO[0] = 285
VRESULTADO[1] = 330
VRESULTADO[2] = 375
VRESULTADO[3] = 420
VRESULTADO[4] = 465
VRESULTADO[5] = 510
VRESULTADO[6] = 555
VRESULTADO[7] = 600
VRESULTADO[8] = 645
VRESULTADO[9] = 690
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej9$
```

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`

```
/*
   Nombre: Alejandro Molina Criado
   Fecha: 01 / 04 / 2020

   para compilar: gcc -O2 -fopenmp -o nombre nombre.c

   Con la cláusula atomic asegura la exclusión mutua en una región de memoria
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <malloc.h>

//#define GLOBAL
#define DINAMIC

#ifdef GLOBAL
        #define MAX 33554432
#endif

int main(int argc, char const *argv[]){
        if(argc != 2){
      printf("Error de argumentos %s", argv[0]);
      return(EXIT_FAILURE);
   }

   struct timespec cgt1, cgt2;
   double ncgt;

   int N = atoi(argv[1]);

   #ifdef GLOBAL
        if(N > MAX) N = MAX;
      int matriz[N][N];
      int vector[N];
      int vector_resultado[N];
      printf("Ejecutado GLOBALMENTE\n");
   #endif

   #ifdef DINAMIC
      int **matriz, *vector, *vector_resultado;
      matriz = (int**) malloc(N * sizeof(int*));
      for(int i = 0; i < N; ++i)
         matriz[i] = (int*) malloc(N * sizeof(int));

      vector = (int*) malloc(N * sizeof(int));
      vector_resultado = (int*) malloc(N * sizeof(int));
      printf("Ejecutado DINAMICAMENTE\n");
   #endif
```

```
    #pragma omp parallel for
    for(int i = 0; i < N; ++i){
            vector[i] = i;
            #pragma omp parallel for
       for(int j = 0; j < N; ++j)
            matriz[i][j] = i + j;
            }

            clock_gettime(CLOCK_REALTIME, &cgt1);
            int suma;
            //Resultado
            for(int i = 0; i < N; i++){
            suma = 0;
            #pragma omp parallel for reduction(+:suma)
            for(int j = 0; j < N; j++)
                        suma += vector[j] * matriz[i][j];

            vector_resultado[i] = suma;
            }

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

            printf("Tiempo: %11.9f seg.\t / Tamaño vectores: %u\n", ncgt, N);
            if(N < 15)
            for(int i = 0; i < N; i++){
                        printf("V_RESULTADO[%d] = %d  ", i, vector_resultado[i]);
                        printf("\n");
            }
            else{
            printf("V_RESULTADO[0] = %d  ", vector_resultado[0]);
            printf("V_RESULTADO[%d] = %d  ", N - 1, vector_resultado[N - 1]); printf("\n");
            }

            #ifdef DINAMIC
            for(int i = 0; i < N; i++)
            free(matriz[i]);

            //LIBERAMOS ESPACIO
            free(matriz); free(vector); free(vector_resultado);
            #endif

    return 0;
}
```

CAPTURAS DE PANTALLA:

```
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej10$ ./pmv-OpenmMP-reduction 5000
Ejecutado DINAMICAMENTE
Tiempo: 0.033783826 seg.          / Tamaño vectores: 5000
V_RESULTADO[0] = -1295505460  V_RESULTADO[4999] = 1049954896
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej10$ ./pmv-OpenmMP-reduction 1000
Ejecutado DINAMICAMENTE
Tiempo: 0.001965589 seg.          / Tamaño vectores: 1000
V_RESULTADO[0] = 332833500  V_RESULTADO[999] = 831834000
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej10$ ./pmv-OpenmMP-reduction 6000
Ejecutado DINAMICAMENTE
Tiempo: 0.040971937 seg.          / Tamaño vectores: 6000
V_RESULTADO[0] = -1032443032  V_RESULTADO[5999] = -442622432
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej10$ ./pmv-OpenmMP-reduction 9000
Ejecutado DINAMICAMENTE
Tiempo: 0.090418826 seg.          / Tamaño vectores: 9000
V_RESULTADO[0] = -1853634372  V_RESULTADO[8999] = 1788117264
[1]+  Terminado (killed)      ./pmv-OpenmMP-reduction 50000
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej10$
```

RESPUESTA: No he tenido ningún problema de ejecución.

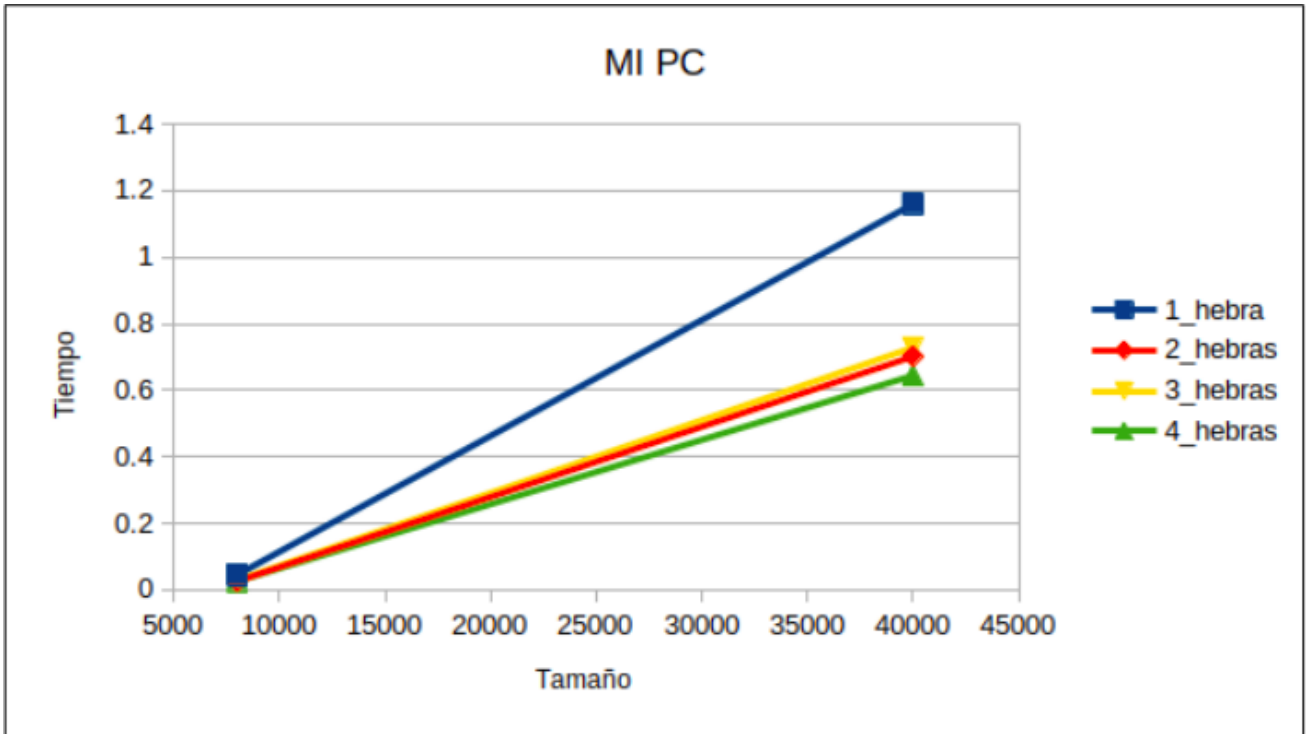
11. Ayudándose de una hoja de cálculo (recuerde que en las aulas está instalado OpenOffice) realice una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del mejor código paralelo de los tres implementados en los ejercicios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

```
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej11$ ./pmv-OpenMP-reduction 12000
Tiempo(seg.): 0.054615627         / Tamaño vectores: 12000
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej11$ ./pmv-OpenMP-reduction 4000
Tiempo(seg.): 0.012610891         / Tamaño vectores: 4000
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej11$ ./pmv-OpenMP-reduction 9000
Tiempo(seg.): 0.034428960         / Tamaño vectores: 9000
alex@alex-pc:~/Escritorio/UNI/AC/PRÁCTICAS/OPEN_MPI/bp2/ej11$
```

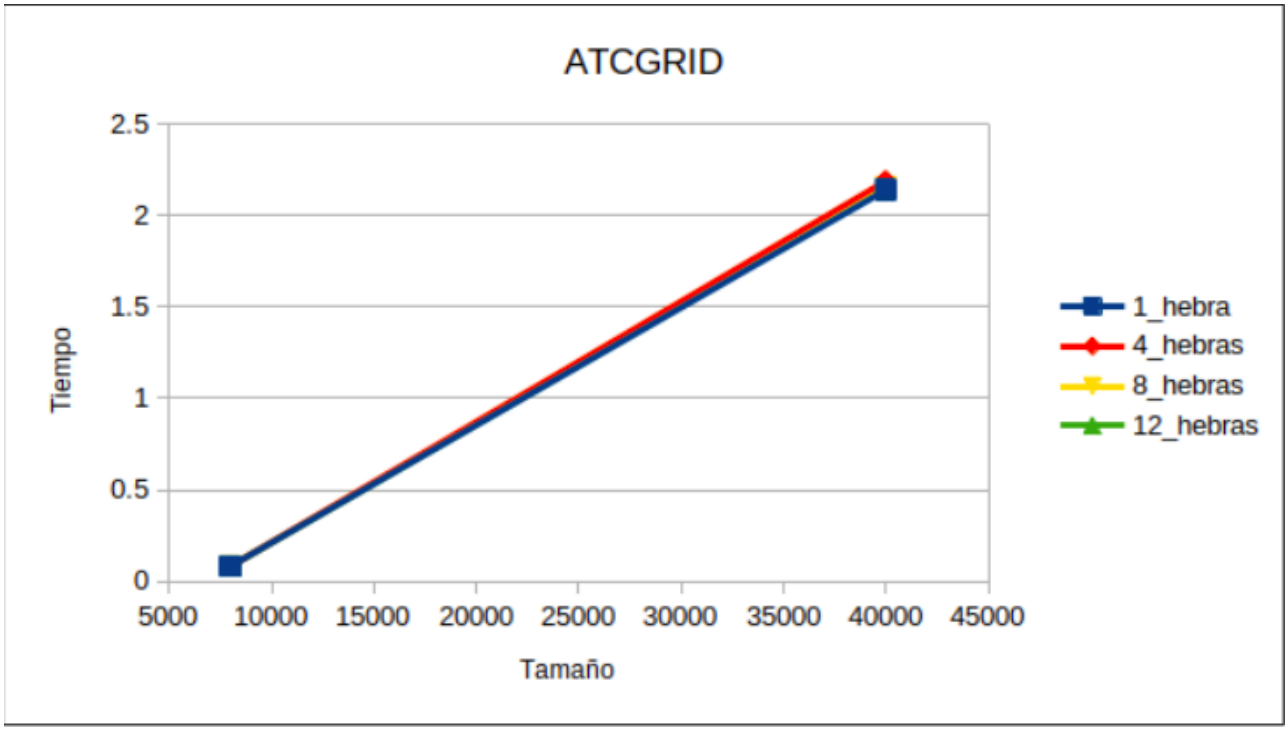
TABLA (con tiempos y ganancia) Y GRÁFICA (con ganancia) (para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N-: un N entre 20000 y 100000, y otro entre 5000 y 20000):

N.º Thread	Tam = 800	Tam = 800
1	0.04652455	1.1595677213
2	0.024508203	0.702323784
3	0.025408401	0.728383977
4	0.024978965	0.643711738



N.º Thread	Tam = 800	Tam = 800
1	0.08312455	2.1395677213
2	0.083508203	2.172323784
3	0.085408401	0.028383977
4	0.083578965	2.143711738

COMENTARIOS SOBRE LOS RESULTADOS: El código más efectivo es el de pmv-OpenMP-a


ATCGRID
Tiempo
2.5
2
1.5
1
0.5
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
Tamaño
1_hebra
4_hebras
8_hebras
12_hebras