


<p>2º curso / 2º cuatr. Grado Ing. Inform. Doble Grado Ing. Inform. y Mat.</p>	<h2>Arquitectura de Computadores (AC)</h2> <h3>Cuaderno de prácticas.</h3> <h3>Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP</h3> <p>Estudiante (nombre y apellidos): Carlos de la Torre</p> <p>Grupo de prácticas: A2</p> <p>Fecha de entrega: 22/04 23:59</p> <p>Fecha evaluación en clase: 30/04</p>	
--	---	---

1. ¿Qué ocurre si en el ejemplo del seminario `shared-clause.c` se añade a la directiva `parallel` la cláusula `default(none)`? Si se plantea algún problema, resuélvalo sin eliminar `default(none)`. Añada el código con la modificación al cuaderno de prácticas.

RESPUESTA:

Da un error de compilación por que no encuentra la variable `N` que esta definida fuera del `parallel` osea con `default(none)` lo que estamos haciendo es obligar al programador a definir explícitamente el alcance las variables que definimos, por lo tanto para que `N` pueda ser vista en la sección del `parallel`, hay que agregarla a la lista de `shared` osea que sea compartida por todas las hebras de la ejecución.

CÓDIGO FUENTE: `shared-clauseModificado.c`

```

/*
 * shared-clause.c
 *
 * Created on: 02/04/2014
 * Author: Carlos de la Torre
 */
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#endif
int 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) default(none)
    for (i = 0; i < n; i++)
        a[i] += i;
    printf("Después de parallel for:\n");
    for (i = 0; i < n; i++)
        printf("a[%d] = %d\n", i, a[i]);
    return 0;
}

```

CAPTURAS DE PANTALLA:

```
[usuario@portatil Practica02_AC]$ ./bin/shared-clause
Después de parallel for:
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 7
a[4] = 9
a[5] = 11
a[6] = 13
[usuario@portatil Practica02_AC]$ ./bin/shared-clauseModificado
Después de parallel for:
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 7
a[4] = 9
a[5] = 11
a[6] = 13
[usuario@portatil Practica02_AC]$
```

2. ¿Qué ocurre si en `private-clause.c` se inicializa la variable `suma` fuera de la construcción `parallel` en lugar de dentro? Razone su respuesta. Añada el código con la modificación al cuaderno de prácticas.

RESPUESTA:

Lo que sucede es que cuando ejecutamos el código con esta modificación no salen los resultados correctos puesto que al no inicializar dentro de la sección `parallel` la variable que vamos a usar OMP no puede asegurarnos cual es el valor de la variable que explícitamente marcamos como privada para cada una de las hebras, en las capturas de pantalla se puede apreciar que en la hebra 0 de casualidad el resultado es 1 osea que estaría correcto pero en el resto de hebras el valor que se le asigna a la variable `suma` es indeterminado y lo que tiene dicha variable es "basura" antes de realizar las sumas.

CÓDIGO FUENTE: `private-clauseModificado2.c`

```
/*
 * private-clause.c
 *
 * Created on: 02/04/2014
 * Author: Carlos de la Torre
 */
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
int main() {
    int i, n = 7;
    int a[n];
    int suma=0;
    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] / ", omp_get_thread_num(), i);
        }
        printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
    }
    printf("\n");
}
```

```
return 0;
}
```

CAPTURAS DE PANTALLA:

```
[usuario@portatil Practica02_AC]$ ./bin/private-clause
thread 0 suma a[0] / thread 0 suma a[1] / thread 1 suma a[2] / thread 1 suma a[3] / thread 3 suma a[6] / thread 2 suma a[4] / thread 2
suma a[5] /
* thread 1 suma= 5
* thread 0 suma= 1
* thread 2 suma= 9
* thread 3 suma= 6
[usuario@portatil Practica02_AC]$ ./bin/private-clauseModificado
thread 0 suma a[0] / thread 0 suma a[1] / thread 1 suma a[2] / thread 1 suma a[3] / thread 3 suma a[6] / thread 2 suma a[4] / thread 2
suma a[5] /
* thread 1 suma= 4196485
* thread 3 suma= 4196486
* thread 0 suma= 1
* thread 2 suma= 4196489
[usuario@portatil Practica02_AC]$
```

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

RESPUESTA:

Lo que ocurre es que todas las sumas son incorrectas puesto que al quitar la variable `suma` del estado `private` o sea que cada una de las hebras pueda acceder a ella de manera exclusiva los accesos a la misma no los puede controlar OMP puesto que la variable esta definida fuera de la sentencia `parallel` aunque si este inicializada dentro del `parallel` al estar definida fuera no es posible asegurar la sincronización de las hebras ha dicha variable.

CÓDIGO FUENTE: `private-clauseModificado3.c`

```
/*
 * private-clause.c
 *
 * Created on: 02/04/2014
 * Author: Carlos de la Torre
 */
#include <stdio.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
int 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 a[%d] / ", omp_get_thread_num(), i);
        }
        printf("\n* thread %d suma= %d", omp_get_thread_num(), suma);
    }
    printf("\n");
    return 0;
}
```

CAPTURAS DE PANTALLA:

```
[usuario@portatil Practica02_AC]$ ./bin/private-clauseModificado3
thread 1 suma a[2] / thread 1 suma a[3] / thread 0 suma a[0] / thread 0 suma a[1] / thread 3 suma a[6] / thread 2 suma a[4] / thread 2
suma a[5] /
* thread 0 suma= 15
* thread 3 suma= 15
* thread 1 suma= 15
* thread 2 suma= 15
[usuario@portatil Practica02_AC]$ ./bin/private-clauseModificado3
thread 0 suma a[0] / thread 0 suma a[1] / thread 3 suma a[6] / thread 2 suma a[4] / thread 2 suma a[5] / thread 1 suma a[2] / thread 1
suma a[3] /
* thread 0 suma= 11
* thread 1 suma= 11
* thread 3 suma= 11
* thread 2 suma= 11
[usuario@portatil Practica02_AC]$ ./bin/private-clauseModificado3
thread 0 suma a[0] / thread 0 suma a[1] / thread 2 suma a[4] / thread 2 suma a[5] / thread 1 suma a[2] / thread 1 suma a[3] / thread 3
suma a[6] /
* thread 1 suma= 15
* thread 2 suma= 15
* thread 0 suma= 15
* thread 3 suma= 15
[usuario@portatil Practica02_AC]$ ./bin/private-clauseModificado3
thread 2 suma a[4] / thread 2 suma a[5] / thread 0 suma a[0] / thread 0 suma a[1] / thread 3 suma a[6] / thread 1 suma a[2] / thread 1
suma a[3] /
* thread 0 suma= 9
* thread 2 suma= 9
* thread 3 suma= 9
* thread 1 suma= 9
[usuario@portatil Practica02_AC]$ ./bin/private-clauseModificado3
thread 0 suma a[0] / thread 0 suma a[1] / thread 3 suma a[6] / thread 2 suma a[4] / thread 2 suma a[5] / thread 1 suma a[2] / thread 1
suma a[3] /
* thread 2 suma= 11
* thread 3 suma= 11
* thread 1 suma= 11
* thread 0 suma= 11
[usuario@portatil Practica02_AC]$
```

4. En la ejecución de `firstlastprivate.c` de la pag. 21 del seminario se imprime un 6. ¿El código imprime siempre 6? Razone su respuesta.

RESPUESTA:

Sí, Siempre imprimirá un 6 por que la sentencia `firstprivate(suma)` hace que la variable `suma` tenga el valor de inicialización que tiene fuera de la sentencia `parallel`, osea que si la inicializamos fuera, ese es el valor que contendrá en cada una de las hebras que la utilicen puesto que es `private`, y con la sentencia `lastprivate` nos estamos asegurando que a la salida de la sección `parallel` tenga el valor de la ultima iteración que debería tener la variable `suma` y como la ultima iteración, sea cual sea la cantidad de hebras en la que se ejecute nuestro programa, siempre será la misma, osea la que llegue al valor $N-1$ por lo tanto siempre sera el mismo valor.

CAPTURAS DE PANTALLA:

```
[A2estudiante7@atcgrid ~]$ echo './02_Practica2_OMP/firstlastprivate' | qsub -q ac
thread 4 suma a[4] suma=4
thread 2 suma a[2] suma=2
thread 0 suma a[0] suma=0
thread 6 suma a[6] suma=6
thread 3 suma a[3] suma=3
thread 1 suma a[1] suma=1
thread 5 suma a[5] suma=5

Fuera de la construcción parallel suma=6
[A2estudiante7@atcgrid ~]$
```

```

thread 0 suma a[1] suma=1
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5

Fuera de la construcción parallel suma=6
[usuario@portatil Practica02_AC]$ ./bin/firstlastprivate
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 3 suma a[6] suma=6
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

Fuera de la construcción parallel suma=6
[usuario@portatil Practica02_AC]$ ./bin/firstlastprivate
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9
thread 3 suma a[6] suma=6

Fuera de la construcción parallel suma=6
[usuario@portatil Practica02_AC]$ ./bin/firstlastprivate
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 3 suma a[6] suma=6
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

Fuera de la construcción parallel suma=6
[usuario@portatil Practica02_AC]$ █

```

5. ¿Qué ocurre si en `copyprivate-clause.c` se elimina la cláusula `copyprivate(a)` en la directiva `single`? ¿A qué cree que es debido?

RESPUESTA:

Lo que ocurre es que los valores que deberían haber en las posiciones del vector no son correctas, puesto que en algunas posiciones hay “basura” y solamente en las posiciones que se asignan en la hebra cuya ejecución hace la petición del valor de la variable “a” es en la que se encuentran los valores correctos en la asignación de las posiciones del vector.

Espero que se entienda lo que quiero decir con esto ya que es bastante complicado de explicar.

El que los valores que tendrían que haber en las diferentes posiciones del vector estén errores es debido a que sin la clausula `copyprivate`, el valor leído por la hebra que se encarga de capturar el valor de la variable “a” solo permanece en esa hebra y por lo tanto las demas hebras no conocen el valor de la variable “a” puesto que aunque si se ha definido dentro de la clausula `parallel` no se ha inicializado en cada una de las hebras que usa dicha variable, por lo tanto quitando esa clausula solo habrá una hebra que tenga el valor correcto de la variable “a”.

CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```

/*
 * copyprivate-clipse.c
 *
 * Created on: 09/04/2014
 * Author: Carlos de la Torre
 */

#include <stdio.h>
#include <omp.h>
int main() {
    int n = 9, i, b[n];
    for (i = 0; i < n; i++)
        b[i] = -1;
    #pragma omp parallel
    {
        int a;
        #pragma omp single
        {
            printf("\nIntroduce valor de inicialización a: ");
            scanf("%d", &a );
            printf("\nSingle ejecutada por el thread
%d\n",omp_get_thread_num());
        }
        #pragma omp for
        for (i = 0; i < n; i++)
            b[i] = a;
    }
    printf("Después de la región parallel:\n");
    for (i = 0; i < n; i++)
        printf("b[%d] = %d\t", i, b[i]);
    printf("\n");
    return 0;
}

```

CAPTURAS DE PANTALLA:

```

[usuario@portatil Practica02_AC]$ ./bin/copyprivate-clipseModificado
Introduce valor de inicialización a: 4
Single ejecutada por el thread 0
Después de la región parallel:
b[0] = 4      b[1] = 4      b[2] = 4      b[3] = 60      b[4] = 60      b[5] = 60      b[6] = 60      b[7] = 0      b[8] =
0
[usuario@portatil Practica02_AC]$ ./bin/copyprivate-clipseModificado
Introduce valor de inicialización a: 5
Single ejecutada por el thread 0
Después de la región parallel:
b[0] = 5      b[1] = 5      b[2] = 5      b[3] = 60      b[4] = 60      b[5] = 0      b[6] = 0      b[7] = 0      b[8] =
0
[usuario@portatil Practica02_AC]$ ./bin/copyprivate-clipse
Introduce valor de inicialización a: 4
Single ejecutada por el thread 3
Después de la región parallel:
b[0] = 4      b[1] = 4      b[2] = 4      b[3] = 4      b[4] = 4      b[5] = 4      b[6] = 4      b[7] = 4      b[8] =
4
[usuario@portatil Practica02_AC]$ ./bin/copyprivate-clipse
Introduce valor de inicialización a: 5
Single ejecutada por el thread 0
Después de la región parallel:
b[0] = 5      b[1] = 5      b[2] = 5      b[3] = 5      b[4] = 5      b[5] = 5      b[6] = 5      b[7] = 5      b[8] =
5
[usuario@portatil Practica02_AC]$ █

```

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

RESPUESTA:

El resultado anterior era 45 y ahora el resultado es 55 y esta claro si inicializamos la

variable suma con 10 unidades el programa seguirá sumando los 45 que suma al recorrer el vector mas los 10 que contiene la variable.

CÓDIGO FUENTE: reduction-clauseModificado2.c

```
/*
 * reduction-clause.c
 *
 * Created on: 09/04/2014
 * Author: Carlos de la Torre
 */

#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", 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);
    return 0;
}
```

CAPTURAS DE PANTALLA:

```
[usuario@portatil Practica02_AC]$ ./bin/reduction-clauseModificado 10
Tras 'parallel' suma=55
[usuario@portatil Practica02_AC]$ ./bin/reduction-clause 10
Tras 'parallel' suma=45
[usuario@portatil Practica02_AC]$ █
```

- En el ejemplo reduction-clause.c, elimine for 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.

CÓDIGO FUENTE: reduction-clauseModificado3.c

```
/*
 * reduction-clause.c
 *
 * Created on: 09/04/2014
 * Author: Carlos de la Torre
 */

#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", n);
    }
    for (i = 0; i < n; i++)
        a[i] = i;
    #pragma omp parallel
    {
        for (i = 0; i < n; i++)
            suma += a[i];
    }
    printf("Tras 'parallel' suma=%d\n", suma);
    return 0;
}
```

```

    }
    n = atoi(argv[1]);
    if (n > 20) {
        n = 20;
        printf("n=%d\n", n);
    }
    for (i = 0; i < n; i++)
        a[i] = i;
#pragma omp parallel sections reduction(+:suma) private(i)
    {
        #pragma omp section
        for (i = 0; i < n/4; i++)
            suma += a[i];
        #pragma omp section
        for (i = n/4; i < n/2; i++)
            suma += a[i];
        #pragma omp section
        for (i = n/2; i < n/2+n/4; i++)
            suma += a[i];
        #pragma omp section
        for (i = n/2+n/4; i < n; i++)
            suma += a[i];
    }
    printf("Tras 'parallel' suma=%d\n", suma);
return 0;
}

```

CAPTURAS DE PANTALLA:

```

[usuario@portatil Practica02_AC]$ ./bin/reduction-clauseModificado2 5
Tras 'parallel' suma=10
[usuario@portatil Practica02_AC]$ ./bin/reduction-clauseModificado2 6
Tras 'parallel' suma=15
[usuario@portatil Practica02_AC]$ ./bin/reduction-clauseModificado2 7
Tras 'parallel' suma=21
[usuario@portatil Practica02_AC]$ ./bin/reduction-clauseModificado2 8
Tras 'parallel' suma=28
[usuario@portatil Practica02_AC]$ ./bin/reduction-clauseModificado2 9
Tras 'parallel' suma=36
[usuario@portatil Practica02_AC]$ ./bin/reduction-clauseModificado2 10
Tras 'parallel' suma=45
[usuario@portatil Practica02_AC]$ ./bin/reduction-clauseModificado3 5
Tras 'parallel' suma=10
[usuario@portatil Practica02_AC]$ ./bin/reduction-clauseModificado3 6
Tras 'parallel' suma=15
[usuario@portatil Practica02_AC]$ ./bin/reduction-clauseModificado3 7
Tras 'parallel' suma=21
[usuario@portatil Practica02_AC]$ ./bin/reduction-clauseModificado3 8
Tras 'parallel' suma=28
[usuario@portatil Practica02_AC]$ ./bin/reduction-clauseModificado3 9
Tras 'parallel' suma=36
[usuario@portatil Practica02_AC]$ ./bin/reduction-clauseModificado3 10
Tras 'parallel' suma=45
[usuario@portatil Practica02_AC]$ █

```

8. Implementar un programa secuencial en C que calcule el producto de una matriz cuadrada, M, por un vector, v1:

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

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).

CÓDIGO FUENTE: pmv-secuencial.c

```

/*
 * pmv-secuencial.c
 *
 * Created on: 12/04/2014
 * Author: Carlos de la Torre
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define PRINT_ALL_MIN 15
// Ponemos que los elementos mínimos para que se
// impriman todos los valores de la matriz sea 15

int main(int argc, char* argv[]) {
    int i,j,k,N,TIME,acumulador;
    double tr;
    struct timespec t1, t2;

    switch (argc){ // coneste switch nos aseguramos de que la entrada de parametros
sea correcta
        case 1:
            printf("Faltan las filas/columnas de la Matriz, y el tamaño del
vector\n");
            printf("\nUso: %s [numero] [0/1]\n",argv[0]);
            printf("\nDonde numero es el tamaño de las filas y las columnas
de la matriz y el tamaño del vector\n");
            printf("y el 0 o el 1 especifica si queremos solo los tiempos (1)
o no\n");
            exit(-1);
            break;
        case 2:
            N = atoi(argv[1]); // Este sera el tamaño del vector y de las
filas/columnas de la matriz
            TIME = 0;
            break;
        case 3:
            N = atoi(argv[1]); // Este sera el tamaño del vector y de las
filas/columnas de la matriz
            TIME = atoi(argv[2]); // si tiene un valor de 0 se imprime toda
la info si tiene un valor de 1 se imprime solo el tiempo
            break;
        default:
            printf("La cantidad de parametros es incorrecta\n");
            exit(-1);
            break;
    }

    int *vector, *Vresultado;
    int **Matriz;
    Matriz = (int**) malloc(N * sizeof(int*));
    for (i = 0; i < N; i++)
        Matriz[i] = (int*) malloc(N * sizeof(int));
    vector = (int*) malloc(N * sizeof(int)); //si no hay espacio suficiente malloc
devuelve NULL
    Vresultado = (int*) malloc(N * N * sizeof(int));
    if ((Matriz == NULL) || (vector == NULL) || (Vresultado == NULL)) {
        printf("Error en la reserva de espacio para los Vectores o Matriz\n");
        exit(-2);
    }

    srand(time(NULL)); // esta es la semilla que se usa para los random
    // Inicializamos la Matriz y el vector
    for (i = 0; i < N; i++){
        for (j = 0; j < N; j++){
            Matriz[i][j] = rand()%10;
        }
        vector[i] = rand()%10;
    }
}

```

```

// imprimimos la matriz y el vector si el tamaño de N < PRINT_ALL_MIN
if (N <= PRINT_ALL_MIN && TIME!=1){
    printf ("\nEsta es la matriz: \n");
    for (i = 0; i < N; i++){
        for (j = 0; j < N; j++){
            printf ("%d ",Matriz[i][j]);
        }
        printf ("\n");
    }
    printf ("\nEste es el vector: \n");
    for (i = 0; i < N; i++)
        printf ("%d ",vector[i]);
    printf("\n\n");
}

// Calcular la multiplicación de una matriz por un vector
clock_gettime(CLOCK_REALTIME, &t1);

for (i = 0; i < N; i++)
    for (j = 0; j < N; j++) {
        acumulador = 0;
        for (k = 0; k < N; k++) {
            acumulador += Matriz[i][j] * vector[k];
        }
        Vresultado[(i*k)+j] = acumulador;
    }

clock_gettime(CLOCK_REALTIME, &t2);

// calculamos el tiempo que hemos tardado en calcular la multiplicación
tr = (double) (t2.tv_sec - t1.tv_sec) + (double) ((t2.tv_nsec - t1.tv_nsec) /
(1.e+9));

// Ahora imprimimos por pantalla los resultados obtenidos segun las
restricciones del problema
if (N <= PRINT_ALL_MIN){
    printf("Tiempo(seg.):%11.9f\nTamaño Matriz y Vector:%u\n",tr,N); // si
queremos imprimir datos completos y N < PRINT_ALL_MIN
    printf ("Este es el vector resultante: \n");
    printf("{");
    for (i = 0; i < N*N; i++){
        printf ("VR[%d]=%d, ",i,Vresultado[i]);
    }
    printf("}\n");
}else if (TIME==1) // si queremos imprimir unicamente el tiempo de cálculo
    printf("%11.9f\n",tr); //
else{ // y si queremos imprimir el tiempo la primera y la ultima multiplicación
    printf("Tiempo(seg.):%11.9f\n",tr);
    printf("Tamaño Matriz y Vector:%u\n",N);
    printf("(Matriz[0][0]=%d)*(Vector[0]=%d)=%d\n",Matriz[0]
[0],vector[0],Matriz[0][0]*vector[0]);
    printf("(Matriz[%d][%d]=%d)*(Vector[%d]=%d)=%d\n",N-1,N-1,Matriz[N-1][N-
1],N-1,vector[N-1],Matriz[N-1][N-1]*vector[N-1]);
}
free(vector);
free(Vresultado);
for(i=0; i<N; i++)
    free(Matriz[i]);
free(Matriz);
return 0;
}

```

CAPTURAS DE PANTALLA:

```
[usuario@portatil Practica02_AC]$ ./bin/pmv-secuencial 5

Esta es la matriz:
4 0 4 6 7
2 4 0 4 0
0 8 7 8 4
2 5 6 9 5
9 5 6 5 2

Este es el vector:
4 4 6 6 5

Tiempo(seg.):0.000001226
Tamaño Matriz y Vector:5
Este es el vector resultante:
{VR[0]=100, VR[1]=0, VR[2]=100, VR[3]=150, VR[4]=175, VR[5]=50, VR[6]=100, VR[7]=0, VR[8]=100, VR[9]=0, VR[10]=0, VR[11]=200, VR[12]=175, VR[13]=200, VR[14]=100, VR[15]=50, VR[16]=125, VR[17]=150, VR[18]=225, VR[19]=125, VR[20]=225, VR[21]=125, VR[22]=150, VR[23]=125, VR[24]=50, }
[usuario@portatil Practica02_AC]$ ./bin/pmv-secuencial 7

Esta es la matriz:
3 0 8 1 2 2 9
9 9 6 3 1 4 8
4 7 8 4 1 6 8
5 5 4 1 4 3 8
5 9 8 7 3 7 8
8 5 7 1 1 8 9
5 7 1 6 5 1 9

Este es el vector:
1 6 3 7 2 7 1

Tiempo(seg.):0.000004847
Tamaño Matriz y Vector:7
Este es el vector resultante:
{VR[0]=81, VR[1]=0, VR[2]=216, VR[3]=27, VR[4]=54, VR[5]=54, VR[6]=243, VR[7]=243, VR[8]=243, VR[9]=162, VR[10]=81, VR[11]=27, VR[12]=108, VR[13]=216, VR[14]=108, VR[15]=189, VR[16]=216, VR[17]=108, VR[18]=27, VR[19]=162, VR[20]=216, VR[21]=135, VR[22]=135, VR[23]=108, VR[24]=27, VR[25]=108, VR[26]=81, VR[27]=216, VR[28]=135, VR[29]=243, VR[30]=216, VR[31]=189, VR[32]=81, VR[33]=189, VR[34]=216, VR[35]=216, VR[36]=135, VR[37]=189, VR[38]=27, VR[39]=27, VR[40]=216, VR[41]=243, VR[42]=135, VR[43]=189, VR[44]=27, VR[45]=162, VR[46]=135, VR[47]=27, VR[48]=243, }
[usuario@portatil Practica02_AC]$
```

9. Implementar en paralelo el producto matriz por vector con OpenMP a partir del código escrito en el ejercicio anterior. 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

```
/*
 * pmv-OpenMp-a.c
 *
 * Created on: 12/04/2014
 * Author: Carlos de la Torre
 */
#include <stdio.h>
```

```

#include <stdlib.h>
#include <time.h>
#include <omp.h> // biblioteca para programas paralelos
#define PRINT_ALL_MIN 15
// Ponemos que los elementos mínimos para que se
// impriman todos los valores de la matriz sea 15

int main(int argc, char* argv[]) {
    int i,j,k,N,TIME;
    double tr;
    // estas son las variables que sirven para medir el tiempo
    double t1, t2;
    switch (argc){ // coneste switch nos aseguramos de que la entrada de parametros
sea correcta
        case 1:
            printf("Faltan las filas/columnas de la Matriz, y el tamaño del
vector\n");
            printf("\nUso: %s [numero] [0/1]\n",argv[0]);
            printf("\nDonde numero es el tamaño de las filas y las columnas
de la matriz y el tamaño del vector\n");
            printf("y el 0 o el 1 especifica si queremos solo los tiempos (1)
o no\n");
            exit(-1);
            break;
        case 2:
            N = atoi(argv[1]); // Este sera el tamaño del vector y de las
filas/columnas de la matriz
            TIME = 0;
            break;
        case 3:
            N = atoi(argv[1]); // Este sera el tamaño del vector y de las
filas/columnas de la matriz
            TIME = atoi(argv[2]); // si tiene un valor de 0 se imprime toda
la info si tiene un valor de 1 se imprime solo el tiempo
            break;
        default:
            printf("La cantidad de parametros es incorrecta\n");
            exit(-1);
            break;
    }

    int *vector, *Vresultado;
    int **Matriz;
    Matriz = (int**) malloc(N * sizeof(int*));
    for (i = 0; i < N; i++)
        Matriz[i] = (int*) malloc(N * sizeof(int));
    vector = (int*) malloc(N * sizeof(int)); //si no hay espacio suficiente malloc
devuelve NULL
    Vresultado = (int*) malloc(N * N * sizeof(int));
    if ((Matriz == NULL) || (vector == NULL) || (Vresultado == NULL)) {
        printf("Error en la reserva de espacio para los Vectores o Matriz\n");
        exit(-2);
    }

    srand(time(NULL)); // esta es la semilla que se usa para los random

#pragma omp parallel for private(i,j) // Inicializamos la Matriz y el vector
    for (i = 0; i < N; i++){
        for (j = 0; j < N; j++){
            Matriz[i][j] = 2;
        }
        vector[i] = 4;
    }

    // imprimimos la matriz y el vector si el tamaño de N es menor de PRINT_ALL_MIN
    if (N <= PRINT_ALL_MIN && TIME!=1){
        printf ("\nEsta es la matriz: \n");
        for (i = 0; i < N; i++){
            for (j = 0; j < N; j++){
                printf ("%d ",Matriz[i][j]);
            }
            printf ("\n");
        }
        printf ("\nEste es el vector: \n");
        for (i = 0; i < N; i++)

```

```

        printf ("%d ",vector[i]);
        printf("\n\n");
    }

    t1 = omp_get_wtime(); // Calcular la multiplicación de una matriz por un vector

    double acumulador=0;
    for (k = 0; k < N; k++){
        acumulador += vector[k];
    }

#pragma omp parallel private (j)
    {
        #pragma omp for
        for (i = 0; i < N; i++){
            for (j = 0; j < N; j++){
                Vresultado[j+i*N] = Matriz[i][j]*acumulador;
            }
        }
    }

    t2 = omp_get_wtime();
    tr = t2 - t1; // Calculo el tiempo que he tardado en multiplicarlo

    // Ahora imprimimos por pantalla los resultados obtenidos segun las
    restricciones del problema
    if (N <= PRINT_ALL_MIN){
        printf("Tiempo(seg.):%11.9f\nTamaño Matriz y Vector:%u\n",tr,N); // si
        queremos imprimir datos completos y N < PRINT_ALL_MIN
        printf ("Este es el vector resultante: \n");
        printf("{");
        for (i = 0; i < N*N; i++){
            printf ("VR[%d]=%d, ",i,Vresultado[i]);
        }
        printf("}\n");
    }else if (TIME==1) // si queremos imprimir unicamente el tiempo de cálculo
        printf("%11.9f\n",tr); //
    else{ // y si queremos imprimir el tiempo la primera y la ultima multiplicación
        printf("Tiempo(seg.):%11.9f\n",tr);
        printf("Tamaño Matriz y Vector:%u\n",N);
        printf("Suma del vector completo: %d\n",vector[0]*N);
        printf("(Matriz[0][0]=%d)*%d=%d\n",Matriz[0][0],vector[0]*N,Matriz[0]
[0]*vector[0]*N);
        printf("(Matriz[%d][%d]=%d)*%d=%d\n",N-1,N-1,Matriz[N-1][N-1],vector[N-
1]*N,Matriz[N-1][N-1]*vector[N-1]*N);
    }
    free(vector);
    free(Vresultado);
    for(i=0; i<N; i++)
        free(Matriz[i]);
    free(Matriz);
    return 0;
}

```

CÓDIGO FUENTE: pmv-OpenMP-b.c

```

/*
 * pmv-OpenMp-b.c
 *
 * Created on: 12/04/2014
 * Author: Carlos de la Torre
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h> // biblioteca para programas paralelos
#define PRINT_ALL_MIN 15
// Ponemos que los elementos mínimos para que se
// impriman todos los valores de la matriz sea 15

int main(int argc, char* argv[]) {
    int i,j,k,N,TIME;
    double tr;
    // estas son las variables que sirven para medir el tiempo
    double t1, t2;
    switch (argc){ // coneste switch nos aseguramos de que la entrada de parametros
sea correcta
        case 1:
            printf("Faltan las filas/columnas de la Matriz, y el tamaño del
vector\n");
            printf("\nUso: %s [numero] [0/1]\n",argv[0]);
            printf("\nDonde numero es el tamaño de las filas y las columnas
de la matriz y el tamaño del vector\n");
            printf("y el 0 o el 1 especifica si queremos solo los tiempos (1)
o no\n");
            exit(-1);
            break;
        case 2:
            N = atoi(argv[1]); // Este sera el tamaño del vector y de las
filas/columnas de la matriz
            TIME = 0;
            break;
        case 3:
            N = atoi(argv[1]); // Este sera el tamaño del vector y de las
filas/columnas de la matriz
            TIME = atoi(argv[2]); // si tiene un valor de 0 se imprime toda
la info si tiene un valor de 1 se imprime solo el tiempo
            break;
        default:
            printf("La cantidad de parametros es incorrecta\n");
            exit(-1);
            break;
    }

    int *vector, *Vresultado;
    int **Matriz;
    Matriz = (int**) malloc(N * sizeof(int*));
    for (i = 0; i < N; i++)
        Matriz[i] = (int*) malloc(N * sizeof(int));
    vector = (int*) malloc(N * sizeof(int)); //si no hay espacio suficiente malloc
devuelve NULL
    Vresultado = (int*) malloc(N * N * sizeof(int));
    if ((Matriz == NULL) || (vector == NULL) || (Vresultado == NULL)) {
        printf("Error en la reserva de espacio para los Vectores o Matriz\n");
        exit(-2);
    }

    srand(time(NULL)); // esta es la semilla que se usa para los random

#pragma omp parallel for private(i,j) // Inicializamos la Matriz y el vector
    for (i = 0; i < N; i++){
        for (j = 0; j < N; j++){
            Matriz[i][j] = 2;
        }
        vector[i] = 4;
    }
}

```

```

// imprimimos la matriz y el vector si el tamaño de N es menor de PRINT_ALL_MIN
if (N <= PRINT_ALL_MIN && TIME!=1){
    printf ("\nEsta es la matriz: \n");
    for (i = 0; i < N; i++){
        for (j = 0; j < N; j++){
            printf ("%d ",Matriz[i][j]);
        }
        printf ("\n");
    }
    printf ("\nEste es el vector: \n");
    for (i = 0; i < N; i++)
        printf ("%d ",vector[i]);
    printf("\n\n");
}

t1 = omp_get_wtime(); // Calcular la multiplicación de una matriz por un vector

double acumulador=0;
for (k = 0; k < N; k++){
    acumulador += vector[k];
}

#pragma omp parallel private (i)
for (i = 0; i < N; i++){
    #pragma omp for
    for (j = 0; j < N; j++){
        Vresultado[j+i*N] = Matriz[i][j]*acumulador;
    }
}

t2 = omp_get_wtime();
tr = t2 - t1; // Calculo el tiempo que he tardado en multiplicarlo

// Ahora imprimimos por pantalla los resultados obtenidos segun las
restricciones del problema
if (N <= PRINT_ALL_MIN){
    printf("Tiempo(seg.):%11.9f\nTamaño Matriz y Vector:%u\n",tr,N); // si
    queremos imprimir datos completos y N < PRINT_ALL_MIN
    printf ("Este es el vector resultante: \n");
    printf("{");
    for (i = 0; i < N*N; i++){
        printf ("VR[%d]=%d, ",i,Vresultado[i]);
    }
    printf("}\n");
}else if (TIME==1) // si queremos imprimir unicamente el tiempo de cálculo
    printf("%11.9f\n",tr); //
else{ // y si queremos imprimir el tiempo la primera y la ultima multiplicación
    printf("Tiempo(seg.):%11.9f\n",tr);
    printf("Tamaño Matriz y Vector:%u\n",N);
    printf("Suma del vector completo: %d\n",vector[0]*N);
    printf("(Matriz[0][0]=%d)*%d=%d\n",Matriz[0][0],vector[0]*N,Matriz[0]
[0]*vector[0]*N);
    printf("(Matriz[%d][%d]=%d)*%d=%d\n",N-1,N-1,Matriz[N-1][N-1],vector[N-
1]*N,Matriz[N-1][N-1]*vector[N-1]*N);
}
free(vector);
free(Vresultado);
for(i=0; i<N; i++)
    free(Matriz[i]);
free(Matriz);
return 0;
}

```

RESPUESTA:

En realidad no tuve problemas en la compilación de los ejecutables donde si los tuve fue en la ejecución de los mismos por que había cometido un error en la reserva de memoria y la había hecho global en vez de dinámica, pero cuando me di cuenta simplemente tuve que cambiar la reserva de memoria con malloc y me funciono correctamente.

CAPTURAS DE PANTALLA:

```
[usuario@portatil Practica02_AC]$ ./bin/pmv-OpenMP-a 5

Esta es la matriz:
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2

Este es el vector:
4 4 4 4 4

Tiempo(seg.):0.000008315
Tamaño Matriz y Vector:5
Este es el vector resultante:
{VR[0]=40, VR[1]=40, VR[2]=40, VR[3]=40, VR[4]=40, VR[5]=40, VR[6]=40, VR[7]=40, VR[8]=40, VR[9]=40, VR[10]=40, VR[11]=40, VR[12]=40, VR[13]=40, VR[14]=40, VR[15]=40, VR[16]=40, VR[17]=40, VR[18]=40, VR[19]=40, VR[20]=40, VR[21]=40, VR[22]=40, VR[23]=40, VR[24]=40, }
[usuario@portatil Practica02_AC]$ ./bin/pmv-OpenMP-a 7

Esta es la matriz:
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2

Este es el vector:
4 4 4 4 4 4 4

Tiempo(seg.):0.000010271
Tamaño Matriz y Vector:7
Este es el vector resultante:
{VR[0]=56, VR[1]=56, VR[2]=56, VR[3]=56, VR[4]=56, VR[5]=56, VR[6]=56, VR[7]=56, VR[8]=56, VR[9]=56, VR[10]=56, VR[11]=56, VR[12]=56, VR[13]=56, VR[14]=56, VR[15]=56, VR[16]=56, VR[17]=56, VR[18]=56, VR[19]=56, VR[20]=56, VR[21]=56, VR[22]=56, VR[23]=56, VR[24]=56, VR[25]=56, VR[26]=56, VR[27]=56, VR[28]=56, VR[29]=56, VR[30]=56, VR[31]=56, VR[32]=56, VR[33]=56, VR[34]=56, VR[35]=56, VR[36]=56, VR[37]=56, VR[38]=56, VR[39]=56, VR[40]=56, VR[41]=56, VR[42]=56, VR[43]=56, VR[44]=56, VR[45]=56, VR[46]=56, VR[47]=56, VR[48]=56, }
[usuario@portatil Practica02_AC]$
```

```
[usuario@portatil Practica02_AC]$ ./bin/pmv-OpenMP-b 5

Esta es la matriz:
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2

Este es el vector:
4 4 4 4 4

Tiempo(seg.):0.000014013
Tamaño Matriz y Vector:5
Este es el vector resultante:
{VR[0]=40, VR[1]=40, VR[2]=40, VR[3]=40, VR[4]=40, VR[5]=40, VR[6]=40, VR[7]=40, VR[8]=40, VR[9]=40, VR[10]=40, VR[11]=40, VR[12]=40, VR[13]=40, VR[14]=40, VR[15]=40, VR[16]=40, VR[17]=40, VR[18]=40, VR[19]=40, VR[20]=40, VR[21]=40, VR[22]=40, VR[23]=40, VR[24]=40, }
[usuario@portatil Practica02_AC]$ ./bin/pmv-OpenMP-b 7

Esta es la matriz:
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2
2 2 2 2 2 2 2

Este es el vector:
4 4 4 4 4 4 4

Tiempo(seg.):0.000014885
Tamaño Matriz y Vector:7
Este es el vector resultante:
{VR[0]=56, VR[1]=56, VR[2]=56, VR[3]=56, VR[4]=56, VR[5]=56, VR[6]=56, VR[7]=56, VR[8]=56, VR[9]=56, VR[10]=56, VR[11]=56, VR[12]=56, VR[13]=56, VR[14]=56, VR[15]=56, VR[16]=56, VR[17]=56, VR[18]=56, VR[19]=56, VR[20]=56, VR[21]=56, VR[22]=56, VR[23]=56, VR[24]=56, VR[25]=56, VR[26]=56, VR[27]=56, VR[28]=56, VR[29]=56, VR[30]=56, VR[31]=56, VR[32]=56, VR[33]=56, VR[34]=56, VR[35]=56, VR[36]=56, VR[37]=56, VR[38]=56, VR[39]=56, VR[40]=56, VR[41]=56, VR[42]=56, VR[43]=56, VR[44]=56, VR[45]=56, VR[46]=56, VR[47]=56, VR[48]=56, }
[usuario@portatil Practica02_AC]$
```

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).

CÓDIGO FUENTE: pmv-OpenmMP-reduction.c

```

/*
 * pmv-OpenMp-b.c
 *
 * Created on: 12/04/2014
 * Author: Carlos de la Torre
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h> // biblioteca para programas paralelos
#define PRINT_ALL_MIN 15
// Ponemos que los elementos mínimos para que se
// impriman todos los valores de la matriz sea 15

int main(int argc, char* argv[]) {
    int i,j,N,TIME;
    double tr, acumulador=0;
    // estas son las variables que sirven para medir el tiempo
    double t1, t2;
    switch (argc){ // coneste switch nos aseguramos de que la entrada de parametros
sea correcta
        case 1:
            printf("Faltan las filas/columnas de la Matriz, y el tamaño del
vector\n");
            printf("\nUso: %s [numero] [0/1]\n",argv[0]);
            printf("\nDonde numero es el tamaño de las filas y las columnas
de la matriz y el tamaño del vector\n");
            printf("y el 0 o el 1 especifica si queremos solo los tiempos (1)
o no\n");
            exit(-1);
            break;
        case 2:
            N = atoi(argv[1]); // Este sera el tamaño del vector y de las
filas/columnas de la matriz
            TIME = 0;
            break;
        case 3:
            N = atoi(argv[1]); // Este sera el tamaño del vector y de las
filas/columnas de la matriz
            TIME = atoi(argv[2]); // si tiene un valor de 0 se imprime toda
la info si tiene un valor de 1 se imprime solo el tiempo
            break;
        default:
            printf("La cantidad de parametros es incorrecta\n");
            exit(-1);
            break;
    }

    int *vector, *Vresultado;
    int **Matriz;
    Matriz = (int**) malloc(N * sizeof(int*));
    for (i = 0; i < N; i++)
        Matriz[i] = (int*) malloc(N * sizeof(int));
    vector = (int*) malloc(N * sizeof(int)); //si no hay espacio suficiente malloc
devuelve NULL
    Vresultado = (int*) malloc(N * N * sizeof(int));
    if ((Matriz == NULL) || (vector == NULL) || (Vresultado == NULL)) {
        printf("Error en la reserva de espacio para los Vectores o Matriz\n");
        exit(-2);
    }

    srand(time(NULL)); // esta es la semilla que se usa para los random

#pragma omp parallel for private(i,j)// Inicializamos la Matriz y el vector
    for (i = 0; i < N; i++){
        for (j = 0; j < N; j++){
            Matriz[i][j] = 2;
        }
    }

```

```

        vector[i] = 4;
    }

    // imprimimos la matriz y el vector si el tamaño de N es menor de PRINT_ALL_MIN
    if (N <= PRINT_ALL_MIN && TIME!=1){
        printf ("\nEsta es la matriz: \n");
        for (i = 0; i < N; i++){
            for (j = 0; j < N; j++){
                printf ("%d ",Matriz[i][j]);
            }
            printf ("\n");
        }
        printf ("\nEste es el vector: \n");
        for (i = 0; i < N; i++)
            printf ("%d ",vector[i]);
        printf("\n\n");
    }

    t1 = omp_get_wtime(); // Calcular la multiplicación de una matriz por un vector
#pragma omp parallel for reduction(+:acumulador)
    for (j = 0; j < N; j++){
        acumulador += vector[j];
    }
#pragma omp parallel private (i)
    for (i = 0; i < N; i++){
        #pragma omp for
        for (j = 0; j < N; j++){
            Vresultado[j+i*N] = Matriz[i][j]*acumulador;
        }
    }

    t2 = omp_get_wtime();
    tr = t2 - t1; // Calculo el tiempo que he tardado en multiplicarlo

    // Ahora imprimimos por pantalla los resultados obtenidos segun las
    restricciones del problema
    if (N <= PRINT_ALL_MIN){
        printf("Tiempo(seg.):%11.9f\nTamaño Matriz y Vector:%u\n",tr,N); // si
        queremos imprimir datos completos y N < PRINT_ALL_MIN
        printf ("Este es el vector resultante: \n");
        printf("{");
        for (i = 0; i < N*N; i++){
            printf ("VR[%d]=%d, ",i,Vresultado[i]);
        }
        printf("}\n");
    }else if (TIME==1) // si queremos imprimir unicamente el tiempo de cálculo
        printf("%11.9f\n",tr); //
    else{ // y si queremos imprimir el tiempo la primera y la ultima multiplicación
        printf("Tiempo(seg.):%11.9f\n",tr);
        printf("Tamaño Matriz y Vector:%u\n",N);
        printf("Suma del vector completo: %d\n",vector[0]*N);
        printf("(Matriz[0][0]=%d)*%d=%d\n",Matriz[0][0],vector[0]*N,Matriz[0]
[0]*vector[0]*N);
        printf("(Matriz[%d][%d]=%d)*%d=%d\n",N-1,N-1,Matriz[N-1][N-1],vector[N-
1]*N,Matriz[N-1][N-1]*vector[N-1]*N);
    }
    free(vector);
    free(Vresultado);
    for(i=0; i<N; i++)
        free(Matriz[i]);
    free(Matriz);
    return 0;
}

```

RESPUESTA:**CAPTURAS DE PANTALLA:**

```
[usuario@fedora20 Practica02_AC]$ ./bin/pmv-OpenMP-reduction 100
Tiempo(seg.):0.003286951
Tamaño Matriz y Vector:100
Suma del vector completo: 400
(Matriz[0][0]=2)*400=800
(Matriz[99][99]=2)*400=800
[usuario@fedora20 Practica02_AC]$ ./bin/pmv-OpenMP-reduction 100
Tiempo(seg.):0.000135158
Tamaño Matriz y Vector:100
Suma del vector completo: 400
(Matriz[0][0]=2)*400=800
(Matriz[99][99]=2)*400=800
[usuario@fedora20 Practica02_AC]$ ./bin/pmv-OpenMP-reduction 100
Tiempo(seg.):0.000136020
Tamaño Matriz y Vector:100
Suma del vector completo: 400
(Matriz[0][0]=2)*400=800
(Matriz[99][99]=2)*400=800
[usuario@fedora20 Practica02_AC]$ ./bin/pmv-OpenMP-reduction 100
```

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 el PC del aula de prácticas de los tres códigos implementados en los ejercicios anteriores para tres tamaños (N) distintos (consulte la Lección 6/Tema 2). Recuerde usar -O2 al compilar.

TABLA Y GRÁFICA (por ejemplo para 1-4 hebras PC aula, y para 1-12 hebras en atcgrid, tamaños-N: 100, 1.000, 10.000):

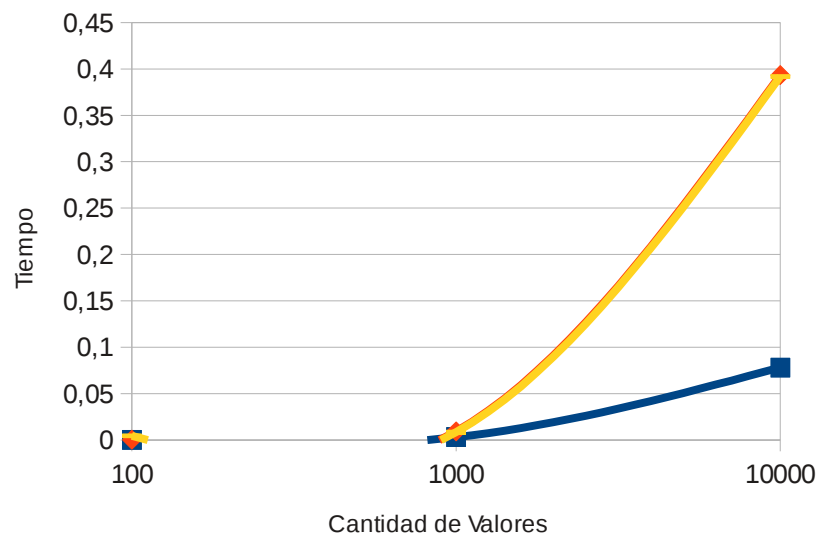
ATCGRID

Tamaños de N	Pmv-OpenMP-a	Pmv-OpenMP-b	Pmv-OpenMP-reduction
100	0,000057811	0,000313166	0,004486664
1000	0,003018287	0,009369392	0,00839488
10000	0,078032804	0,39335534	0,391870047

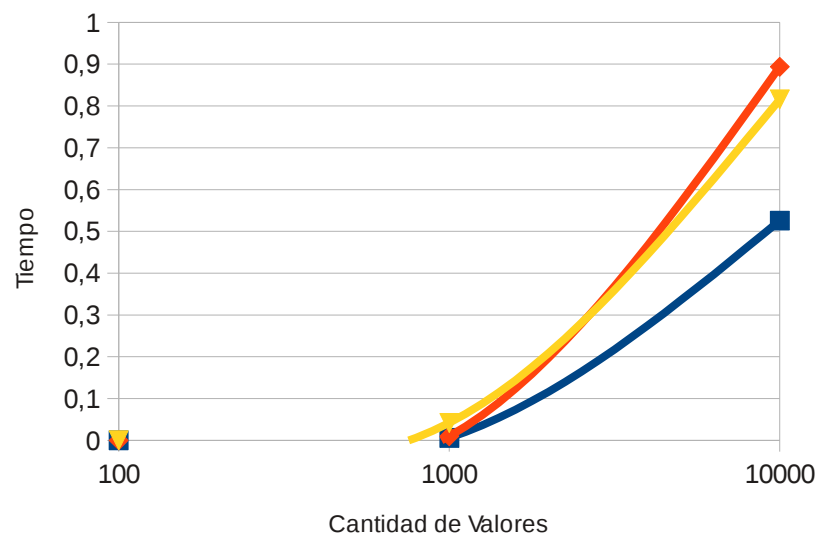
Ordenador Local

Tamaños de N	Pmv-OpenMP-a	Pmv-OpenMP-b	Pmv-OpenMP-reduction
100	0,000038771	0,000130234	0,000130256
1000	0,006119436	0,009597601	0,041709973
10000	0,525658766	0,893670617	0,815693785

Tiempos de Ejecución ATCGRID



Tiempos de Ejecución Locales



COMENTARIOS SOBRE LOS RESULTADOS: