2º curso / 2º cuatr. Grados Ingeniería Informática	**Arquitectura de Computadores (AC)** **Cuaderno de prácticas.** **Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP** Estudiante (nombre y apellidos): Grupo de prácticas y profesor de prácticas:

Parte I. Ejercicios basados en los ejemplos del seminario práctico

1. **(a)** Añadir la cláusula default(none) a la directiva parallel del ejemplo del seminario shared-clause.c? ¿Qué ocurre? ¿A qué se debe? **(b)** Resolver el problema generado sin eliminar default(none). Incorporar el código con la modificación al cuaderno de prácticas. (Añadir capturas de pantalla que muestren lo que ocurre)

   RESPUESTA: Con default(none) el programador especifica el alcance de todas las variables usadas en la construcción y en ningún momento se especifica el alcance de n, por lo que da un error de compilación.

   CAPTURA CÓDIGO FUENTE: shared-clauseModificado.c

```c
#include <stdio.h>
#ifndef _OPENMP
        #include <omp.h>
#endif

int main (){

        int i, n=7;
        int a[n];

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

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

CAPTURAS DE PANTALLA:

```
[ac125@atcgrid Practica2]$ gcc -O2 -fopenmp shared-clauseModificado.c -o shared-
clauseModificado
[ac125@atcgrid Practica2]$ srun shared-clauseModificado
Despues de parallel for:
a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5
a[5] = 6
a[6] = 7
[ac125@atcgrid Practica2]$
```

2. **(a)** Añadir a lo necesario a private-clause.c para que imprima suma fuera de la región parallel. Inicializar suma dentro del parallel a un valor distinto de 0. Ejecutar varias veces el código ¿Qué imprime el código fuera del parallel? (mostrar lo que ocurre con una captura de pantalla) Razonar respuesta. **(b)** Modificar el código del apartado (a) para que se inicialice suma fuera del parallel en lugar de dentro ¿Qué ocurre? Comparar todo lo que imprime el código ahora con la salida en (a) (mostrar la salida con una captura de pantalla) Razonar respuesta.

(a) RESPUESTA:Como fuera del parallel el valor de suma no se inicializa, nos muestra un valor basura porque las operaciones dentro del parallel no afectan fuera de este.

CAPTURA CÓDIGO FUENTE: private-clauseModificado_a.c

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

int main (){

        int i, n=7,suma;
        int a[n];

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

        #pragma omp parallel private(suma)
        {
                suma = 5;
                #pragma omp for
                for (i = 0; i < n ; i++)
                {
                        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\n* suma fuera de parallel= %d",suma );
        printf("\n\n");

}
```

CAPTURAS DE PANTALLA:

```
[ac125@atcgrid Practica2]$ clear

[ac125@atcgrid Practica2]$ gcc -O2 -fopenmp private-clauseModificado.c -o private-clauseModificado
[ac125@atcgrid Practica2]$ srun private-clauseModificado
thread 0 suma a[0] / thread 0 suma a[1] / thread 0 suma a[2] / thread 0 suma a[3] / thread 0 suma a[4] / thread 0 suma a[5] / thread 0 suma a[6] /
* thread 0 suma=15
* thread 0 suma=23

* suma fuera de parallel= 0

[ac125@atcgrid Practica2]$ srun private-clauseModificado
thread 0 suma a[4] / thread 0 suma a[5] / thread 0 suma a[6] / thread 0 suma a[0] / thread 0 suma a[1] / thread 0 suma a[2] / thread 0 suma a[3] /
* thread 0 suma=23
* thread 0 suma=15

* suma fuera de parallel= 0

[ac125@atcgrid Practica2]$
```

(b) RESPUESTA:Ahora ocurre lo contrario. El valor inicializado fuera del parallel es el que se imprime al final después de este, mientras que dentro del parallel, al no inicializarse suma en ningún momento, nos muestra valores indefinidos

CAPTURA CÓDIGO FUENTE: private-clauseModificado_b.c

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

int main (){

        int i, n=7,suma;
        int a[n];

        for (i = 0; i < n; i++)
                a[i] = i;
        suma = 5;
        #pragma omp parallel private(suma)
        {

                #pragma omp for
                for (i = 0; i < n ; i++)
                {
                        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\n* suma fuera de parallel= %d",suma );
        printf("\n\n");

}
```

CAPTURAS DE PANTALLA:

```
[ac125@atcgrid Practica2]$ gcc -O2 -fopenmp private-clauseModificado.c -o private-clauseModificado
[ac125@atcgrid Practica2]$ srun private-clauseModificado
thread 0 suma a[4] / thread 0 suma a[5] / thread 0 suma a[6] / thread 0 suma a[0] / thread 0 suma a[1] / thread 0 suma a[2] / thread 0 suma a[3] /
* thread 0 suma=4198943
* thread 0 suma=8

* suma fuera de parallel= 5

[ac125@atcgrid Practica2]$
```

3. **(a)** Eliminar la cláusula private(suma) en private-clause.c. Ejecutar el código resultante. ¿Qué ocurre? **(b)** ¿A qué es debido?

RESPUESTA:Si quitamos la cláusula la variable se comparte para todas las hebras provocando

condiciones de carrera, implicando eso resultados impredecibles y diferentes con cada ejecución.

CAPTURA CÓDIGO FUENTE: private-clauseModificado3.c

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

int main (){

        int i, n=7,suma;
        int a[n];

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

        #pragma omp parallel
        {
                suma = 0;
                #pragma omp for
                for (i = 0; i < n ; i++)
                {
                        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\n* suma fuera de parallel= %d",suma );
        printf("\n\n");

}
```

CAPTURAS DE PANTALLA:

```
[ac125@atcgrid Practica2]$ gcc -O2 -fopenmp private-clauseModificado.c -o private-clauseModificado
[ac125@atcgrid Practica2]$ srun private-clauseModificado
thread 0 suma a[4] / thread 0 suma a[5] / thread 0 suma a[6] / thread 0 suma a[0] / thread 0 suma a[1] / thread 0 suma a[2] / thread 0 suma a[3] /
* thread 0 suma=21
* thread 0 suma=21

* suma fuera de parallel= 21
```

Deberían salir valores impredecibles

4. En la ejecución de firstlastprivate.c de la pag. 21 del seminario se imprime un 6 fuera de la región parallel. **(a)** Cambiar el tamaño del vector a 10. Razonar lo que imprime el código en su PC con esta modificación. (añadir capturas de pantalla que muestren lo que ocurre). **(b)** Sin cambiar el tamaño del vector ¿podría imprimir el código otro valor? Razonar respuesta (añadir capturas de pantalla que muestren lo que ocurre).

(a) RESPUESTA: Con el firstprivate, suma adquiere el valor adquirido inicialmente, es decir, 0. Con las lastprivate, suma adquiere finalmente el valor de la última asignación que se le ha hecho, la de la última iteración, es decir, 9

CAPTURAS DE PANTALLA:

```
[ac125@atcgrid Practica2]$ gcc -O2 -fopenmp firstlastprivate.c -o firstlastprivate
[ac125@atcgrid Practica2]$ srun firstlastprivate
 thread 1 suma a[4] suma=4
 thread 1 suma a[5] suma=9
 thread 1 suma a[6] suma=15
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 0 suma a[2] suma=3
 thread 0 suma a[3] suma=6

Fuera de la construcciÃ³n parallel suma=15
[ac125@atcgrid Practica2]$
```

(b) RESPUESTA:Si cambiamos el número de threads, el valor final de suma es diferente

CAPTURAS DE PANTALLA:

```
[ac125@atcgrid Practica2]$ export OMP_NUM_THREADS=4
[ac125@atcgrid Practica2]$ srun firstlastprivate
 thread 1 suma a[2] suma=2
 thread 1 suma a[3] suma=5
 thread 2 suma a[4] suma=4
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 2 suma a[5] suma=9
 thread 3 suma a[6] suma=6

Fuera de la construcciÃ³n parallel suma=6
[ac125@atcgrid Practica2]$
```

5. **(a)** ¿Qué se observa en los resultados de ejecución de copyprivate-clause.c cuando se elimina la cláusula copyprivate(a) en la directiva single? **(b)** ¿A qué cree que es debido? (añadir una captura de pantalla que muestre lo que ocurre)

RESPUESTA: copyprivate(a) permite que una variable privada de un thread se copie a las variables privadas del mismo nombre del resto de threads. Al quitarse, el valor a solo se inicializa en un thread y en los demás el valor es indefinido.

CAPTURA CÓDIGO FUENTE: copyprivate-clauseModificado.c

```c
#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 //copyprivate(a)
        {
            printf("\nIntroduce valor de inicializaciÃ³n 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("DepuÃ©s de la regiÃ³n parallel:\n");

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

    printf("\n");
}
```

CAPTURAS DE PANTALLA:

```
[ac125@atcgrid Practica2]$ gcc -O2 -fopenmp copyprivate-clause.c -o copyprivate-clause
[ac125@atcgrid Practica2]$ srun copyprivate-clause
2

Introduce valor de inicializaciÃ³n a:
Single ejecutada por el thread 1
DepuÃ©s de la regiÃ³n parallel:
b[0] = 0        b[1] = 0        b[2] = 0        b[3] = 2        b[4] = 2        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:Lo mismo pero se le suma 10 más al resultado final

CAPTURA CÓDIGO FUENTE: reduction-clauseModificado.c

```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:

```
jesus@jesus-HP-EliteBook-830-G7-Notebook-PC:~/AC/Practicas/Practica2$ gcc -O2 -fopenmp reduction-clauseMo
if.c -o redMod
jesus@jesus-HP-EliteBook-830-G7-Notebook-PC:~/AC/Practicas/Practica2$ gcc -O2 -fopenmp reduction-clause.c
-o red
jesus@jesus-HP-EliteBook-830-G7-Notebook-PC:~/AC/Practicas/Practica2$ ./red
Falta iteraciones
jesus@jesus-HP-EliteBook-830-G7-Notebook-PC:~/AC/Practicas/Practica2$ ./red 2
Tras 'parallel' suma=1
jesus@jesus-HP-EliteBook-830-G7-Notebook-PC:~/AC/Practicas/Practica2$ ./red 2
red     redMod
jesus@jesus-HP-EliteBook-830-G7-Notebook-PC:~/AC/Practicas/Practica2$ ./redMod 2
Tras 'parallel' suma=11
jesus@jesus-HP-EliteBook-830-G7-Notebook-PC:~/AC/Practicas/Practica2$
```

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:Se hace lo mismo simplemente reduction es más eficiente que

CAPTURA CÓDIGO FUENTE: reduction-clauseModificado7.c

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

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

CAPTURAS DE PANTALLA:

```
jesus@jesus-HP-EliteBook-830-G7-Notebook-PC:~/AC/Practicas/Practica2$ gedit reduction-Mod2.c
jesus@jesus-HP-EliteBook-830-G7-Notebook-PC:~/AC/Practicas/Practica2$ gcc -O2 -fopenmp reduction-Mod2.c -o
 redMod2
jesus@jesus-HP-EliteBook-830-G7-Notebook-PC:~/AC/Practicas/Practica2$ ./red
red      redMod   redMod2
jesus@jesus-HP-EliteBook-830-G7-Notebook-PC:~/AC/Practicas/Practica2$ ./redMod2
Falta iteraciones
jesus@jesus-HP-EliteBook-830-G7-Notebook-PC:~/AC/Practicas/Practica2$ ./redMod2 5
Tras 'parallel' suma=10
jesus@jesus-HP-EliteBook-830-G7-Notebook-PC:~/AC/Practicas/Practica2$ ./red 5
Tras 'parallel' suma=10
jesus@jesus-HP-EliteBook-830-G7-Notebook-PC:~/AC/Practicas/Practica2$
```

Parte II. Resto de ejercicios (usar en atcgrid la cola ac a no

ser que se tenga que usar atcgrid4)

8. Implementar en paralelo el producto matriz por vector con OpenMP usando la directiva for. Partir del código secuencial disponible en SWAD. 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, v2, para tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (4) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código que calcula el producto matriz vector, el número de hilos que usa 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

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

#define MAX 1000

int main(int argc, char** argv) {
    unsigned int N;
    struct timespec cgt1, cgt2; double ncgt;
    int i, j;

    if (argc < 2) {
        printf("Usage: %s size\n", argv[0]);
        exit (EXIT_FAILURE);
    }

    N = atoi(argv[1]);
#ifdef VECTOR_GLOBAL
    if ( N > MAX) N = MAX;
#endif

#ifdef VECTOR_DYNAMIC
    double *v1, *v2, **m;
    v1 = (double*)malloc(N* sizeof(double));
    v2 = (double*)malloc(N* sizeof(double));
    m = (double**)malloc(N* sizeof(double*));
    if ( (v1 == NULL) || (v2 == NULL) || (m == NULL)) {
        printf("Not enough space for v1, v2 and m\n");
        exit (EXIT_FAILURE);
    }
    for (i = 0; i < N; i++) {
        m[i] = (double*)malloc(N* sizeof(double));
        if (m[i] == NULL) {
            printf("Not enough space for m\n");
            exit(EXIT_FAILURE);
        }
    }
#endif
```

```
    // Initialize vector and matriz
#pragma omp parallel for private (j)
    for (i=0; i < N; i++){
        v1[i] = 0.1*i;
        v2[i] = 0;
        for (j = 0; j < N; j++)
            m[i][j] = i*N+j;
    }

        // Calculate v2 = m * v1
        clock_gettime (CLOCK_REALTIME, &cgt1);
#pragma omp parallel private(i)
        for (i = 0; i < N ; i++){
                //#pragma omp for
                for(j = 0; j < N; j++){
                   // #pragma omp critical
                    v2[i] += m[i][j] * v1[j];
                }


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

        // Print results
        if (N < 10) {
            printf("Time: %11.9f\t Size: %u\n", ncgt, N);
            printf("Matrix:\n\t");
            for (i = 0; i < N; i++) {
                for (j = 0; j < N; j++)
                    printf("%8.6f\t",m[i][j]);
                printf("\n\t");
            }
            printf("Vector:\n\t");
            for (i = 0; i < N; i++)
                printf("%8.6f", v1[i]);
            printf("\n\nVector resultado:\n\t");
            for (i = 0; i < N; i++)
                printf("%8.6f", v2[i]);
            printf("\n");
        } else {
            printf("Time: %11.9f \t Size: %u\t v2[0]: %8.6f \t v2 [N-1]: %8.6f \n", ncgt, N, v2[0],
v2 [N-1]);
        }


    return(0);
}
```

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

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

#define MAX 1000

int main(int argc, char** argv) {
    unsigned int N;
    struct timespec cgt1, cgt2; double ncgt;
    int i, j;

    if (argc < 2) {
        printf("Usage: %s size\n", argv[0]);
        exit (EXIT_FAILURE);
    }

    N = atoi(argv[1]);
#ifdef VECTOR_GLOBAL
    if ( N > MAX) N = MAX;
#endif

#ifdef VECTOR_DYNAMIC
    double *v1, *v2, **m;
    v1 = (double*)malloc(N* sizeof(double));
    v2 = (double*)malloc(N* sizeof(double));
    m = (double**)malloc(N* sizeof(double*));
    if ( (v1 == NULL) || (v2 == NULL) || (m == NULL)) {
        printf("Not enough space for v1, v2 and m\n");
        exit (EXIT_FAILURE);
    }
    for (i = 0; i < N; i++) {
        m[i] = (double*)malloc(N* sizeof(double));
        if (m[i] == NULL) {
            printf("Not enough space for m\n");
            exit(EXIT_FAILURE);
        }
    }
#endif

    // Initialize vector and matriz
#pragma omp parallel for private (j)
    for (i=0; i < N; i++){
        v1[i] = 0.1*i;
        v2[i] = 0;
        for (j = 0; j < N; j++)
            m[i][j] = i*N+j;
    }
```

```
    // Initialize vector and matriz
#pragma omp parallel for private (j)
    for (i=0; i < N; i++){
        v1[i] = 0.1*i;
        v2[i] = 0;
        for (j = 0; j < N; j++)
            m[i][j] = i*N+j;
    }

        // Calculate v2 = m * v1
        clock_gettime (CLOCK_REALTIME, &cgt1);
#pragma omp parallel private(i)
        for (i = 0; i < N ; i++){
                #pragma omp for
                for(j = 0; j < N; j++){
                    #pragma omp critical
                    v2[i] += m[i][j] * v1[j];
                }


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

        // Print results
        if (N < 10) {
            printf("Time: %11.9f\t Size: %u\n", ncgt, N);
            printf("Matrix:\n\t");
            for (i = 0; i < N; i++) {
                for (j = 0; j < N; j++)
                    printf("%8.6f\t",m[i][j]);
                printf("\n\t");
            }
            printf("Vector:\n\t");
            for (i = 0; i < N; i++)
                printf("%8.6f", v1[i]);
            printf("\n\nVector resultado:\n\t");
            for (i = 0; i < N; i++)
                printf("%8.6f", v2[i]);
            printf("\n");
        } else {
            printf("Time: %11.9f \t Size: %u\t v2[0]: %8.6f \t v2 [N-1]: %8.6f \n", ncgt, N, v2[0],
v2 [N-1]);
        }


    return(0);
}
```

RESPUESTA:

CAPTURAS DE PANTALLA:

```
[ac125@atcgrid Practica2]$ gcc -O2 -fopenmp -DVECTOR_DYNAMIC pmv-OpenMP-a.c -o redMod2
[ac125@atcgrid Practica2]$ srun redMod2 6
Time: 0.000000000        Size: 6
Matrix:
        0.000000        1.000000        2.000000        3.000000        4.000000        5.000000
        6.000000        7.000000        8.000000        9.000000        10.000000       11.000000
        12.000000       13.000000       14.000000       15.000000       16.000000       17.000000
        18.000000       19.000000       20.000000       21.000000       22.000000       23.000000
        24.000000       25.000000       26.000000       27.000000       28.000000       29.000000
        30.000000       31.000000       32.000000       33.000000       34.000000       35.000000
        Vector:
        0.0000000.1000000.2000000.3000000.4000000.500000

Vector resultado:
        22.00000058.00000094.000000130.000000166.000000202.000000
[ac125@atcgrid Practica2]$
```

```
[ac125@atcgrid Practica2]$ gcc -O2 -fopenmp -DVECTOR_DYNAMIC pmv-OpenMP-b.c -o redMod2
[ac125@atcgrid Practica2]$ srun redMod2 6
Time: 0.000000000        Size: 6
Matrix:
        0.000000        1.000000        2.000000        3.000000        4.000000        5.000000
        6.000000        7.000000        8.000000        9.000000        10.000000       11.000000
        12.000000       13.000000       14.000000       15.000000       16.000000       17.000000
        18.000000       19.000000       20.000000       21.000000       22.000000       23.000000
        24.000000       25.000000       26.000000       27.000000       28.000000       29.000000
        30.000000       31.000000       32.000000       33.000000       34.000000       35.000000
        Vector:
        0.0000000.1000000.2000000.3000000.4000000.500000

Vector resultado:
        5.50000014.50000023.50000032.50000041.50000050.500000
[ac125@atcgrid Practica2]$
```

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

```
#pragma omp parallel for private (j)
    for (i=0; i < N; i++){
        v1[i] = 0.1*i;
        v2[i] = 0;
        for (j = 0; j < N; j++)
            m[i][j] = i*N+j;
    }

        // Calculate v2 = m * v1
        clock_gettime (CLOCK_REALTIME, &cgt1);
#pragma omp parallel private(i)
        for (i = 0; i < N ; i++){
                #pragma omp for reduction(+:v2[i])
                for(j = 0; j < N; j++){
                    //#pragma omp critical
                    v2[i] += m[i][j] * v1[j];
                }


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

        // Print results
        if (N < 10) {
            printf("Time: %11.9f\t Size: %u\n", ncgt, N);
            printf("Matrix:\n\t");
            for (i = 0; i < N; i++) {
                for (j = 0; j < N; j++)
                    printf("%8.6f\t",m[i][j]);
                printf("\n\t");
            }
            printf("Vector:\n\t");
            for (i = 0; i < N; i++)
                printf("%8.6f", v1[i]);
            printf("\n\nVector resultado:\n\t");
            for (i = 0; i < N; i++)
                printf("%8.6f", v2[i]);
            printf("\n");
```

El unico cambio es lo comentado, y el reduction el resto es igual ya que asi hace lo mismo que el b

RESPUESTA:

CAPTURAS DE PANTALLA:

```
[ac125@atcgrid Practica2]$ gcc -O2 -fopenmp -DVECTOR_DYNAMIC pmv-OpenMP-reduction.c -o redMod2
[ac125@atcgrid Practica2]$ srun redMod2 6
Time: 0.000000000        Size: 6
Matrix:
        0.000000        1.000000        2.000000        3.000000        4.000000        5.000000
        6.000000        7.000000        8.000000        9.000000        10.000000       11.000000
        12.000000       13.000000       14.000000       15.000000       16.000000       17.000000
        18.000000       19.000000       20.000000       21.000000       22.000000       23.000000
        24.000000       25.000000       26.000000       27.000000       28.000000       29.000000
        30.000000       31.000000       32.000000       33.000000       34.000000       35.000000
        Vector:
        0.0000000.1000000.2000000.3000000.4000000.500000

Vector resultado:
        5.50000014.50000023.50000032.50000041.50000050.500000
[ac125@atcgrid Practica2]$
```

10. Realizar una tabla y una gráfica que permitan comparar la escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid4, en uno de los nodos de la cola ac 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):

JUSTIFICAR AHORA EN BASE AL CÓDIGO LA DIFERENCIA EN TIEMPOS:

CAPTURA DE PANTALLA del script pmv-OpenmMP-script.sh

CAPTURAS DE PANTALLA (mostrar la ejecución en atcgrid – envío(s) a la cola):

TABLA (con tiempos y ganancia) Y GRÁFICA (con ganancia):

Tabla 1. Tiempos de ejecución del código secuencial y de la versión paralela para atcgrid y para el PC personal

	atcgrid1, atcgrid2 o atcgrid3				**atcgrid4**				**PC**			
	Tamaño= entre 5000 y 10000		**Tamaño= entre 10000 y 100000**		**Tamaño= entre 5000 y 10000**		**Tamaño= entre 10000 y 100000**		**Tamaño= entre 5000 y 10000**		**Tamaño= entre 10000 y 100000**	
Nº de núcleos (p)	**T(p)**	**S(p)**	**T(p)**	**S(p)**	**T(p)**	**S(p)**	**T(p)**	**S(p)**	**T(p)**	**S(p)**	**T(p)**	**S(p)**
Código Secuencial		----		----		----		----		----		----
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
32												

COMENTARIOS SOBRE LOS RESULTADOS: