

Grai2º curso / 2º  
cuatr.

Grado Ing. Inform.

Doble Grado Ing.  
Inform. y Mat.

# Arquitectura de Computadores (AC)

## Cuaderno de prácticas.

### Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos):

Grupo de prácticas:

Fecha de entrega:

Fecha evaluación en clase

## Ejercicios basados en los ejemplos del seminario práctico

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

**RESPUESTA:** al utilizar la clausula `default(none)` todas las variables declaradas dentro de la directiva `parallel` deben tener su ámbito especificado, por lo que el compilador no devuelve error.

**CAPTURA CÓDIGO FUENTE:** `shared-clauseModificado.c`

**a)**

```
#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) 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]);
}
```

**b)**

```
#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;
```

```
[SERGIO AGUILERA RAMIREZ usuario@cvi065155:~/Desktop/BP2-AC] 2018-04-03 martes
$gcc-7 -O2 -fopenmp shared-clauseModificado.c -o shared-clauseModificado
[SERGIO AGUILERA RAMIREZ usuario@cvi065155:~/Desktop/BP2-AC] 2018-04-03 martes
$./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
```

```
        #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]);
    }
```

#### CAPTURAS DE PANTALLA:

a)

```
[SERGIO AGUILERA RAMIREZ usuario@cvi065155:~/Desktop/BP2-AC] 2018-04-03 martes
$gcc-7 -O2 -fopenmp shared-clauseModificado.c -o shared-clauseModificado
shared-clauseModificado.c: In function 'main':
shared-clauseModificado.c:15:12: error: 'n' not specified in enclosing 'parallel'
    #pragma omp parallel for shared(a) default(none)
                        ^~~~~~
shared-clauseModificado.c:15:12: error: enclosing 'parallel'
shared-clauseModificado.c:15:12: error: enclosing 'parallel'
```

b)

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

#### RESPUESTA:

Al inicializar la variable fuera de la sección `parallel` esta contiene un número indeterminado, por lo que esta debemos de inicializarla dentro de la región.

**CAPTURA CÓDIGO FUENTE:** `private-clauseModificado.c`

```
[SERGIO AGUILERA RAMIREZ usuario@cvi065155:~/Desktop/BP2-AC] 2018-04-03 martes
$gcc-7 -O2 -fopenmp private-clauseModificado.c -o private-clauseModificado
[SERGIO AGUILERA RAMIREZ usuario@cvi065155:~/Desktop/BP2-AC] 2018-04-03 martes
$./private-clauseModificado
thread 0 suma a[0] / thread 1 suma a[4] / thread 0 suma a[1] / thread 1 suma a[5] / thread 0 suma a[2] / thread 1 suma a[6] / thread 0 suma a[3] /
* thread 0 suma= -375104842
* thread 1 suma= -1996488689
```

```
#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;

    suma=10;

    #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");
}
```

### CAPTURAS DE PANTALLA:

a) si inicializamos la variable fuera de la directiva parallel esta tendra un valor erróneo

b) si inicializamos la variable suma dentro de la directiva parallel se obtendra el valor correcto

```
$gcc-7 -O2 -fopenmp private-clauseModificado.c -o private-clauseModificado
[SERGIO AGUILERA RAMIREZ usuario@cvi065155:~/Desktop/BP2-AC] 2018-04-03 martes
$./private-clauseModificado
thread 0 suma a[0] / thread 1 suma a[4] / thread 0 suma a[1] / thread 1 suma a[5] / thread 0 suma a[2] / thread 1 suma a[6] / thread 0 suma a[3] /
* thread 1 suma= 25
* thread 0 suma= 16
```

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

**RESPUESTA:**

La variable `suma` pasaría a ser compartida por lo que esta podría ser sobrescrita por alguna hebra.

**CAPTURA CÓDIGO FUENTE:** `private-clauseModificado3.c`

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

main()
{
    int i, n = 7;
    int a[n], suma;

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

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

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

**CAPTURAS DE PANTALLA:**

```
[SERGIO AGUILERA RAMIREZ usuario@MacBook-Pro-de-Sergio-Aguilera:~/Desktop/BP2-AC] 2018-04
-09 lunes
$gcc-7 -O2 -fopenmp private-clauseModificado3.c -o private-clauseModificado3
[SERGIO AGUILERA RAMIREZ usuario@MacBook-Pro-de-Sergio-Aguilera:~/Desktop/BP2-AC] 2018-04
-09 lunes
$./private-clauseModificado3
thread 0 suma a[0] / thread 3 suma a[6] / thread 1 suma a[2] / thread 2 suma a[4] / thread
d 0 suma a[1] / thread 1 suma a[3] / thread 2 suma a[5] /
* thread 3 suma= 13
* thread 1 suma= 13
* thread 0 suma= 13
* thread 2 suma= 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.

**RESPUESTA:**

La cláusula `lastprivate` asigna a la variable `suma` el último valor en la ejecución secuencial

```
MacBook-Pro-de-Sergio-Aguilera:codigos usuario$ ./firstlastprivate-clause
thread 0 suma a[0] suma=0
thread 3 suma a[6] suma=6
thread 1 suma a[2] suma=2
thread 2 suma a[4] suma=4
thread 0 suma a[1] suma=1
thread 1 suma a[3] suma=5
thread 2 suma a[5] suma=9
Fuera de la construcción parallel suma=6
```

que  
sería

siempre 6.

### CAPTURAS DE PANTALLA:

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?

#### RESPUESTA:

Esto es incorrecto ya que la variable `a` podría contener basura ya que esta la hemos leído en una hebra pero no la hemos escrito en las demás hebras, la clausula `copyprivate` tiene la función de copiar el valor de la variable en el resto de hebras después de la ejecución del `single`.

#### CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.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
        {
            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");
}
```

### CAPTURAS DE PANTALLA:

```
[SERGIO AGUILERA RAMIREZ usuario@MacBook-Pro-de-Sergio-Aguilera:~/Desktop/BP2-AC/codigos]
2018-04-09 lunes
$gcc-7 -O2 -fopenmp copyprivate-clause.c -o copyprivate-clause
[SERGIO AGUILERA RAMIREZ usuario@MacBook-Pro-de-Sergio-Aguilera:~/Desktop/BP2-AC/codigos]
2018-04-09 lunes
$./copyprivate-clause

Introduce valor de inicialización a: 10

Single ejecutada por el thread 2
Después de la región parallel:
b[0] = 1      b[1] = 1      b[2] = 1      b[3] = 32647      b[4] = 32647      b[5] = 10
      b[6] = 10      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

#### RESPUESTA:

el código suma todos los números desde el 0 hasta n, por lo que si inicializamos suma a 10 el resultado sería la suma de los n números más 10.

#### CAPTURA CÓDIGO FUENTE: `reduction-clauseModificado.c`

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

int main(int argc, char **argv)
{
    int i, n=20, a[n], suma=10;

    if(argc < 2) {
        fprintf(stderr, "Falta iteraciones\n");
        exit(-1);
    }

    n = atoi(argv[1]);
    if (n>20)
    {
        n=20;
        printf("n=%d",n);
    }

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

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

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

**CAPTURAS DE PANTALLA:**

```
[SERGIO AGUILERA RAMIREZ usuario@MacBook-Pro-de-Sergio-Aguilera:~/Desktop/BP2-AC/codigos]
2018-04-09 lunes
$./reduction-clauseModificado 10
Tras 'parallel' suma=55
[SERGIO AGUILERA RAMIREZ usuario@MacBook-Pro-de-Sergio-Aguilera:~/Desktop/BP2-AC/codigos]
2018-04-09 lunes
$./reduction-clauseModificado 40
n=20Tras 'parallel' suma=200
```

7. En el ejemplo `reduction-clause.c`, elimine `reduction()` de `#pragma omp parallel for reduction(+:suma)` y haga las modificaciones necesarias para que se siga realizando la suma de los componentes del vector `a` en paralelo sin usar directivas de trabajo compartido.

**RESPUESTA:****CAPTURA CÓDIGO FUENTE:** `reduction-clauseModificado7.c`**CAPTURAS DE PANTALLA:**

## Resto de ejercicios

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

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

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

**CAPTURA CÓDIGO FUENTE:** `pmv-secuencial.c`**GLOBALES:**

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

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

```
#define omp_get_thread_num() 0
#endif
```

```

#define MAX 1000

int main (int argc, char** argv)
{
    double tiempo1, tiempo2, total;

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

    unsigned int N = atoi(argv[1]);

    if ( N>MAX )
    {
        N=MAX;
    }

    double vector1[MAX], vector2[MAX], matriz[MAX][MAX];

    for ( int i=0; i<N; i++ ){
        vector1[i]=i;
        vector2[i]=0;

        for(int j=0; j<N; j++){
            matriz[i][j]=i+j;
        }
    }

    tiempo1=omp_get_wtime();

    for(int k=0; k<N; k++)
    {
        for ( int v=0; v<N; v++ )
        {
            vector2[k] += matriz[k][v] * vector1[v];
        }
    }

    tiempo2=omp_get_wtime();
    total = tiempo2 - tiempo1;

    printf("Tiempo: %11.9f\t / Tamaño:%u\t/ vector2[0]=%8.6f vector2[%d]=%8.6f\n", total,N,vector2[0], N-1, vector2[N-1]);

    return 0;
}

```

**DINAMICOS:**

```

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

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

#define MAX 1000

int main (int argc, char** argv)
{
    double tiempo1, tiempo2, total;

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

    unsigned int N = atoi(argv[1]);

    double *vector1, *vector2, **matriz;

```



```

vector1 = (double*) malloc(N*sizeof(double));
vector2 = (double*) malloc(N*sizeof(double));
matriz = (double**) malloc(N*sizeof(double *));

if ( (vector1==NULL) || (vector2==NULL) || (matriz==NULL) ){
    printf("Error en la reserva de memoria para los vectores");
    exit(-2);
}

for ( int i=0; i<N; i++){
    matriz[i] = (double*) malloc(N*sizeof(double));
    if(matriz[i]==NULL){
        printf("Error en la reserva de memoria para los vectores");
        exit(-2);
    }
}

for ( int i=0; i<N; i++){
    vector1[i]=i;
    vector2[i]=0;

    for(int j=0; j<N; j++){
        matriz[i][j]=i+j;
    }
}

tiempo1=omp_get_wtime();

for(int k=0; k<N; k++)
{
    for ( int v=0; v<N; v++ )
    {
        vector2[k] += matriz[k][v] * vector1[v];
    }
}

tiempo2=omp_get_wtime();
total = tiempo2 - tiempo1;

printf("Tiempo: %11.9f\t / Tamaño:%u\t/ vector2[0]=%8.6f vector2[%d]=%8.6f\n", total,N,vector2[0], N-1, vector2[N-1]);

if (N<20)
{
    for(int m=0; m<N; m++)
    {
        printf(" vector2[%d]=%5.2f\n", m, vector2[m]);
    }
}

free(vector1);
free(vector2);

for(int z=0; z<N; z++)
{
    free(matriz[z]);
}

free(matriz);

return 0;
}

```

### CAPTURAS DE PANTALLA:

#### Vectores Globales:

```

[SERGIO AGUILERA RAMIREZ usuario@cvi096172:~/Desktop/BP2-AC/codigos] 2018-04-10 martes
$gcc-7 -O2 -fopenmp pmv-secuencial.c -o pmv-secuencial
[SERGIO AGUILERA RAMIREZ usuario@cvi096172:~/Desktop/BP2-AC/codigos] 2018-04-10 martes
$./pmv-secuencial 10000
Tiempo: 0.001444000 / Tamaño:1000 / vector2[0]=332833500.000000 vector2[999]=831834000.000000

```

#### Vectores Dinámicos:

```
[SERGIO AGUILERA RAMIREZ usuario@cvi096172:~/Desktop/BP2-AC/codigos] 2018-04-10 martes
$gcc-7 -O2 -fopenmp pmv-secuencial-Dinamico.c -o pmv-secuencial-Dinamico
[SERGIO AGUILERA RAMIREZ usuario@cvi096172:~/Desktop/BP2-AC/codigos] 2018-04-10 martes
$./pmv-secuencial-Dinamico 10000
Tiempo: 0.614839000 / Tamaño:10000 / vector2[0]=333283335000.000000 vector2[9999]=8331833400
00.000000
```

9. Implementar en paralelo el producto matriz por vector con OpenMP a partir del código escrito en el ejercicio anterior usando la directiva `for`. Debe implementar dos versiones del código (consulte la lección 5/Tema 2):

- una primera que paralelice el bucle que recorre las filas de la matriz y
- una segunda que paralelice el bucle que recorre las columnas.

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

- Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
- Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).

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

#### CAPTURA CÓDIGO FUENTE : `pmv-OpenMP-a.c`

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

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

#define MAX 1000

int main (int argc, char** argv)
{
    double tiempo1, tiempo2, total;
    int p;

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

    unsigned int N = atoi(argv[1]);

    double *vector1, *vector2, **matriz;
    vector1 = (double*) malloc(N*sizeof(double));
    vector2 = (double*) malloc(N*sizeof(double));
    matriz = (double**) malloc(N*sizeof(double *));
```

```

if ( (vector1==NULL) || (vector2==NULL) || (matriz==NULL) ){
    printf("Error en la reserva de memoria para los vectores");
    exit(-2);
}

for ( int i=0; i<N; i++){
    matriz[i] = (double*) malloc(N*sizeof(double));
    if(matriz[i]==NULL){
        printf("Error en la reserva de memoria para los vectores");
        exit(-2);
    }
}

#pragma omp parallel
{
    #pragma omp for private(p)
    for ( int m=0; m<N; m++)
    {
        vector1[m]=m;
        vector2[m]=m;
        for(p=0; p<N; p++)
        {
            matriz[m][p]= m+p;
        }
    }

    #pragma omp single
    tiempo1 = omp_get_wtime();

    #pragma omp for private(p)
    for(int w=0; w<N; w++)
    {
        for(int r=0; r<N; r++ )
        {
            vector2[w] += matriz[w][r] * vector1[r];
        }
    }

    #pragma omp single
    tiempo2 = omp_get_wtime();
}

total = tiempo2 - tiempo1;

printf("Tiempo: %11.9f\t / Tamaño:%u\t/ vector2[0]=%8.6f vector2[%d]=%8.6f\n", total,N,vector2[0], N-1, vector2[N-1]);

if (N<20)
{
    for(int m=0; m<N; m++)
    {
        printf(" vector2[%d]=%5.2f\n", m, vector2[m]);
    }
}

free(vector1);
free(vector2);

for(int z=0; z<N; z++)
{
    free(matriz[z]);
}

free(matriz);

return 0;
}

```

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

```

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

#ifdef _OPENMP
#include <omp.h>

```

```

#else
#define omp_get_thread_num() 0
#endif

#define MAX 1000

int main (int argc, char** argv)
{
    double tiempo1, tiempo2, total;
    int p;

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

    unsigned int N = atoi(argv[1]);

    double *vector1, *vector2, **matriz;
    vector1 = (double*) malloc(N*sizeof(double));
    vector2 = (double*) malloc(N*sizeof(double));
    matriz = (double**) malloc(N*sizeof(double *));

    if ( (vector1==NULL) || (vector2==NULL) || (matriz==NULL) ){
        printf("Error en la reserva de memoria para los vectores");
        exit(-2);
    }

    for ( int i=0; i<N; i++){
        matriz[i] = (double*) malloc(N*sizeof(double));
        if(matriz[i]==NULL){
            printf("Error en la reserva de memoria para los vectores");
            exit(-2);
        }
    }

    for(int m=0; m<N; m++){

        vector1[m]=m;
        vector2[m]=0;

        #pragma omp parallel for
        for ( int k=0; k<N; k++){
            matriz[m][k]=m+k;
        }
    }

    tiempo1= omp_get_wtime();

    for ( int t=0; t<N; t++){
        #pragma omp parallel
        {
            double tmp=0;
            #pragma omp for
            for(int n=0; n<N; n++){
                tmp+=matriz[t][n] * vector1[n];
            }

            #pragma omp critical
            vector2[t]+=tmp;
        }
    }

    tiempo2= omp_get_wtime();

    total = tiempo2 - tiempo1;

    printf("Tiempo: %11.9f\t / Tamaño:%u\t/ vector2[0]=%8.6f vector2[%d]=%8.6f\n", total,N,vector2[0], N-1, vector2[N-1]);

    if (N<20)
    {

```

```

for(int m=0; m<N; m++)
{
    printf(" vector2[%d]=%5.2f\n", m, vector2[m]);
}

free(vector1);
free(vector2);

for(int z=0; z<N; z++)
{
    free(matriz[z]);
}

free(matriz);

return 0;

}

```

**RESPUESTA:**

El código pmv-OpenMP-b.c me daba números incorrectos ya que no utilizaba ninguna forma de sumar de forma atómica por lo que use la directiva critical por lo que ya me daba resultados correctores en la ejecución.

**CAPTURAS DE PANTALLA:**

pmv-OpenMP-a.c :

```

[SERGIO AGUILERA RAMIREZ usuario@cvi096172:~/Desktop/BP2-AC/codigos] 2018-04-10 martes
$gcc-7 -O2 -fopenmp pmv-OpenMP-a.c -o pmv-OpenMP-a
[SERGIO AGUILERA RAMIREZ usuario@cvi096172:~/Desktop/BP2-AC/codigos] 2018-04-10 martes
$./pmv-OpenMP-a 10000
Tiempo: 0.608604000 / Tamaño:10000 / vector2[0]=333283335000.000000 vector2[9999]=8331833499
99.000000

```

pmv-OpenMP-b.c :

```

[SERGIO AGUILERA RAMIREZ usuario@cvi096172:~/Desktop/BP2-AC/codigos] 2018-04-10 martes
$gcc-7 -O2 -fopenmp pmv-OpenMP-b.c -o pmv-OpenMP-b
[SERGIO AGUILERA RAMIREZ usuario@cvi096172:~/Desktop/BP2-AC/codigos] 2018-04-10 martes
$./pmv-OpenMP-b 10000
Tiempo: 1.279800000 / Tamaño:10000 / vector2[0]=333283335000.000000 vector2[9999]=8331833400
00.000000

```

10. A partir de la segunda versión de código paralelo desarrollado en el ejercicio anterior, implementar una versión paralela del producto matriz por vector con OpenMP que use para comunicación/sincronización la cláusula `reduction`. Respecto a este ejercicio:

- Anote en su cuaderno de prácticas todos los errores de compilación que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).
- Anote todos los errores en tiempo de ejecución que se han generado durante la realización del ejercicio y explique cómo los ha resuelto (especifique qué ayudas externas ha usado o recibido).

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

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

```

#include <stdio.h>

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

#define MAX 1000

int main (int argc, char** argv)
{
    double tiempo1, tiempo2, total;

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

    unsigned int N = atoi(argv[1]);

    double *vector1, *vector2, **matriz;
    vector1 = (double*) malloc(N*sizeof(double));
    vector2 = (double*) malloc(N*sizeof(double));
    matriz = (double**) malloc(N*sizeof(double *));

    if ( (vector1==NULL) || (vector2==NULL) || (matriz==NULL) ){
        printf("Error en la reserva de memoria para los vectores");
        exit(-2);
    }

    for ( int i=0; i<N; i++){
        matriz[i] = (double*) malloc(N*sizeof(double));
        if(matriz[i]==NULL){
            printf("Error en la reserva de memoria para los vectores");
            exit(-2);
        }
    }

    for ( int i=0; i<N; i++){
        vector1[i]=i;
        vector2[i]=0;

        for(int j=0; j<N; j++){
            matriz[i][j]=i+j;
        }
    }

    tiempo1=omp_get_wtime();

    for(int k=0; k<N; k++)
    {
        int tmp=0;
        #pragma omp parallel for reduction(+:tmp)
        for ( int c=0; c<N; c++){
            tmp+=matriz[k][c] * vector1[c];
        }
        vector2[k] = tmp;
    }

    tiempo2=omp_get_wtime();
    total = tiempo2 - tiempo1;

    printf("Tiempo: %11.9f\t / Tamaño:%u\t/ vector2[0]=%8.6f vector2[%d]=%8.6f\n", total,N,vector2[0], N-1, vector2[N-1]);

```

```

if (N<20)
{
    for(int m=0; m<N; m++)
    {
        printf(" vector2[%d]=%5.2f\n", m, vector2[m]);
    }
}

free(vector1);
free(vector2);

for(int z=0; z<N; z++)
{
    free(matriz[z]);
}

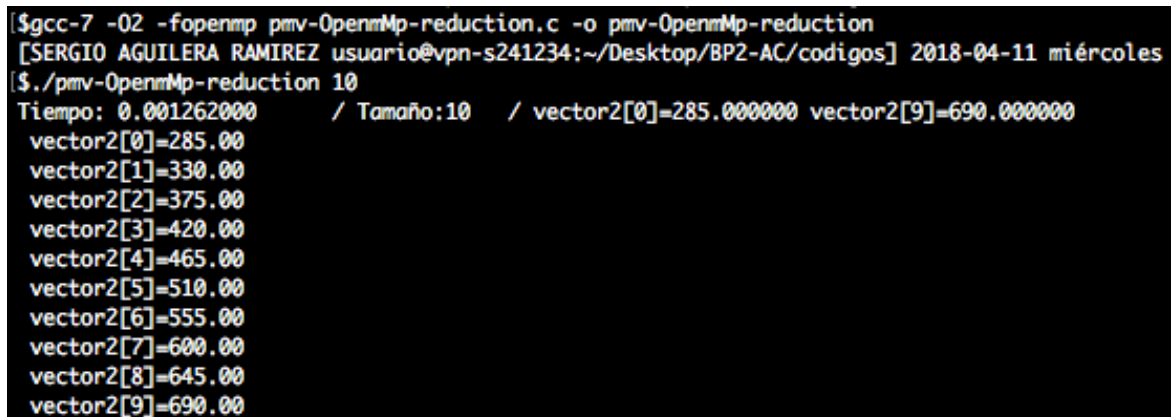
free(matriz);

return 0;
}

```

**RESPUESTA:**

Tuve problemas en compilación ya que intente hacer reduction directamente en el vector cosa que la biblioteca OpenMP no permite por lo que cree una variable para almacenar la suma.

**CAPTURAS DE PANTALLA:**


```

$gcc-7 -O2 -fopenmp pmv-OpenmMp-reduction.c -o pmv-OpenmMp-reduction
[SERGIO AGUILERA RAMIREZ usuario@vpn-s241234:~/Desktop/BP2-AC/codigos] 2018-04-11 miércoles
$./pmv-OpenmMp-reduction 10
Tiempo: 0.001262000 / Tamaño:10 / vector2[0]=285.000000 vector2[9]=690.000000
vector2[0]=285.00
vector2[1]=330.00
vector2[2]=375.00
vector2[3]=420.00
vector2[4]=465.00
vector2[5]=510.00
vector2[6]=555.00
vector2[7]=600.00
vector2[8]=645.00
vector2[9]=690.00

```

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):**

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

#### **COMENTARIOS SOBRE LOS RESULTADOS:**

He utilizado el código del ejercicio-9-a ( filas ), para mi pc local he ejecutado para dos tamaños diferentes ( 10000 y 30000 ) y para las hebras de 1 a 4 obteniendo los tiempos representados en la tabla, en la tabla asociada podemos ver la escalabilidad de las distintas hebras. Para atcgrid he utilizado los mismos tamaños pero en su ejecución he utilizado el número de hebras de la 1 a la 12 obteniendo los tiempos mostrados en la tabla, he de decir que para el tamaño N=30000 he obtenido unos tiempos menores que en mi pc local.

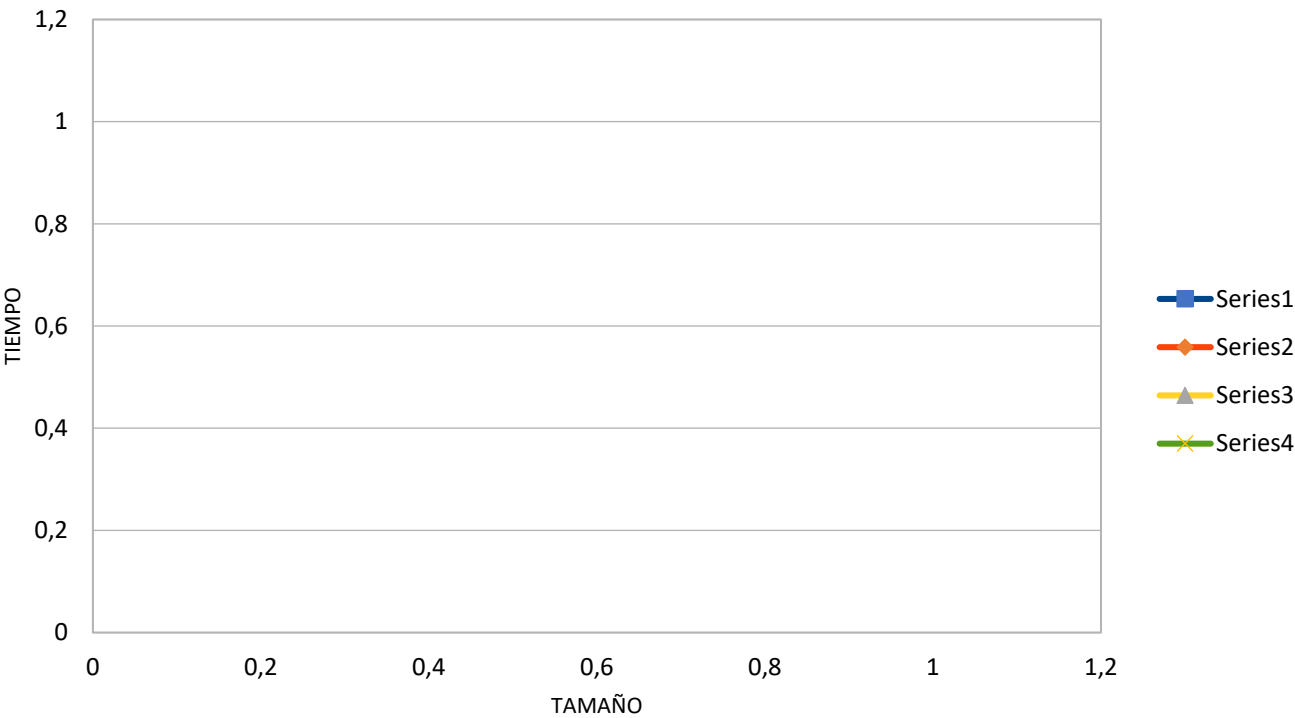
**Pc local: Mac – Procesador: 2,8 GHz Intel Core i7**

**pmv-OpenMP-a**

**PC LOCAL:**

TAMAÑO	1 threads	2 threads	3 threads	4 threads
10000	0,909723000	0,059639000	0,059944000	0,049452000
30000	39,826792000	12,490093000	17,159844000	9,091296000





**ATCGRID:**

TAM ÑAO	1 threa ds	2 threa ds	3 threa ds	4 threa ds	5 threa ds	6 threa ds	7 threa ds	8 threa ds	9 threa ds	10 threa ds	11 threa ds	12 threa ds
1000 0	0,147 5367 39	0,085 2530 04	0,057 3655 37	0,048 6021 61	0,044 6346 69	0,045 6952 28	0,038 1602 83	0,033 9193 96	0,035 5004 15	0,033 0998 53	0,042 5319 04	0,039 2234 58
3000 0	1,206 8037 35	1,093 8057 15	0,752 8725 42	0,681 6715 23	0,471 0245 31	0,455 3409 99	0,513 3823 99	0,591 0620 44	0,314 7588 52	0,435 6149 61	0,310 7167 92	0,348 4720 71

