

2º 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): Francisco Javier Bolívar Lupiáñez

Grupo de prácticas: B1

Fecha de entrega: 25/04/2014

Fecha evaluación en clase:

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: Con `none` se ha de especificar el alcance de todas las variables de la construcción.

CÓDIGO FUENTE: `shared-clauseModificado.c`

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

int main(int argc, char ** argv)
{
    int i, n = 7;
    int a[n];
    for (i=0; i<n; i++)
        a[i] = i+1;
    #pragma omp parallel for default(none) shared(a,n)
        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:



```
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P2$ ./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
```

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: Si no se inicializa dentro. Esta variable coge un valor “basura” porque es una variable privada en esa sección. Por eso siempre se debe inicializar dentro. Si la inicializas fuera no pasará nada siempre y cuando la hayas inicializado también dentro.

CÓDIGO FUENTE: `private-clauseModificado.c`

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

int main(int argc, char ** argv)
{
    int i, n = 7;
    int a[n], suma=8;

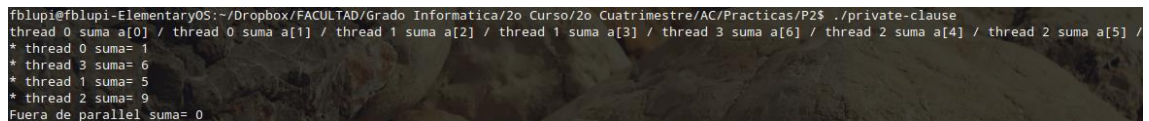
    for (i=0; i<n; i++)
        a[i] = i;

    #pragma omp parallel private(suma)
```

```

{
    //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("\nFuera de parallel suma= %d", suma);
printf("\n");
}

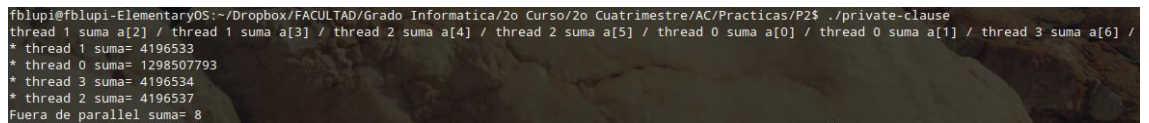
```

CAPTURAS DE PANTALLA:


```

fbilupi@fbilupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P2$ ./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 0 suma= 1
* thread 3 suma= 6
* thread 1 suma= 5
* thread 2 suma= 9
Fuera de parallel suma= 0

```



```

fbilupi@fbilupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P2$ ./private-clause
thread 1 suma a[2] / thread 1 suma a[3] / 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= 4196533
* thread 0 suma= 1298507793
* thread 3 suma= 4196534
* thread 2 suma= 4196537
Fuera de parallel suma= 8

```

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

RESPUESTA: Al no ser una variable privada, todos los threads usarían la misma variable y guardarían su valor calculado en el mismo lugar por lo que aparecerá que todos “han calculado” lo mismo.

CÓDIGO FUENTE: `private-clauseModificado3.c`

```

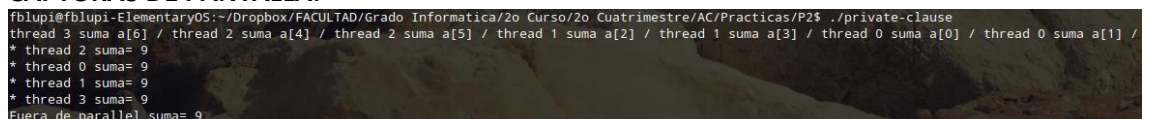
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{
    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("\nFuera de parallel suma= %d", suma);
    printf("\n");
}

```

CAPTURAS DE PANTALLA:


```

fbilupi@fbilupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P2$ ./private-clause
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 a[0] / thread 0 suma a[1] /
* thread 2 suma= 9
* thread 0 suma= 9
* thread 1 suma= 9
* thread 3 suma= 9
Fuera de parallel suma= 9

```

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, porque al tener la cláusula `last`. Se devolverá el valor local de la hebra que haga la última iteración. En este caso, la hebra 6.

CAPTURAS DE PANTALLA:

```
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P2$ ./firstlastprivate-clause
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 3 suma a[6] suma=6
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

Fuera de la construcción parallel suma=6
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P2$ ./firstlastprivate-clause
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
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1

Fuera de la construcción parallel suma=6
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P2$ ./firstlastprivate-clause
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
```

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: No se copia el valor de esa variable en el resto de threads, por tanto están inicializados a “basura”.

CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

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

int main(int argc, char ** argv)
{
    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:

```
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P2$ ./copyprivate-clause
Introduce valor de inicialización a: 2

Single ejecutada por el thread 2
Después de la región parallel:
b[0] = 32767   b[1] = 32767   b[2] = 32767   b[3] = 32568   b[4] = 32568   b[5] = 32568   b[6] = 2       b[7] = 2       b[8] = 2
```

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

RESPUESTA: 10 veces más del que imprimiría antes ya que inicializa el valor de `suma` a 10.

CÓDIGO FUENTE: `reduction-clauseModificado.c`

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

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:

```
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P2$ ./reduction-clause 5
Tras 'parallel' suma=10
```

```
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P2$ ./reduction-clause 5
Tras 'parallel' suma=20
```

7. 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.

RESPUESTA: He usado una variable local para cada hebra y después he hecho la suma de estas variables locales.

CÓDIGO FUENTE: `reduction-clauseModificado7.c`

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

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", n);}

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

    #pragma omp parallel
    {
        int sumalocal=0;
        #pragma omp for schedule(static)
        for (i=0; i<n; i++) {
            sumalocal += a[i];
        }
    }
}
```

```

    }
    #pragma omp atomic
    suma+=sumalocal;
}
printf("Tras 'parallel' suma=%d\n",suma);
}

```

CAPTURAS DE PANTALLA:

```

fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P2$ ./reduction-clause 20
Tras 'parallel' suma=190

```

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

```

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

#define TIMES
// #define PRINTF_ALL

main(int argc, char **argv) {
    // 1. Lectura valores de entrada
    if(argc < 2) {
        fprintf(stderr, "Falta num\n");
        exit(-1);
    }
    int n = atoi(argv[1]);
    if (n > 10000) {
        n = 10000;
        printf("n=%d", n);
    }
    int i, j;
    struct timespec ini, fin; double transcurrido;

    // 2. Creación e inicialización de vector y matriz
    // Creación
    int *v1, *v2;
    v1 = (int*) malloc(n*sizeof(double));
    v2 = (int*) malloc(n*sizeof(double));

    int **M;
    M = (int**) malloc(n*sizeof(int*));
    for(i=0; i<n; i++)
        M[i] = (int*) malloc(n*sizeof(int));

    // Inicialización
    for(i=0; i<n; i++)

```

```

    v1[i]=i;

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

// 3. Impresión de vector y matriz
#ifndef TIMES
#ifdef PRINTF_ALL
    printf("Vector inicial:\n");
    for (i=0; i<n; i++) printf("%d ",v1[i]);
    printf("\n");

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

// 4. Cálculo resultado
clock_gettime(CLOCK_REALTIME,&ini);
for (i=0; i<n; i++) {
    v2[i]=0;
    for (j=0; j<n; j++) {
        v2[i]+=M[i][j]*v1[j];
    }
}
clock_gettime(CLOCK_REALTIME,&fin);
transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

// 5. Impresión de vector resultado
#ifdef TIMES
    printf("%d %11.9f\n",n,transcurrido);
#else
#ifdef PRINTF_ALL
    printf("Tiempo: %11.9f\n",transcurrido);
    printf("Vector resultado (M x v1):\n");
    for (i=0; i<n; i++) printf("%d ",v2[i]);
    printf("\n");
#else
    printf("Tiempo: %11.9f\n",transcurrido);
    printf("v2[0]: %d, v2[n-1]: %d\n",v2[0],v2[n-1]);
#endif
#endif

// 6. Eliminar de memoria
free(M);
free(v1);
free(v2);
}

```

CAPTURAS DE PANTALLA:

```

fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P2$ ./pmv-secuencial 5
Vector inicial:
0 1 2 3 4
Matriz inicial:
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
20 21 22 23 24
Tiempo: 0.000000707
Vector resultado (M x v1):
30 80 130 180 230

```

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

```

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

#define TIMES
// #define PRINTF_ALL

main(int argc, char **argv) {
    // 1. Lectura valores de entrada
    if(argc < 2) {
        fprintf(stderr, "Falta num\n");
        exit(-1);
    }
    int n = atoi(argv[1]);
    if (n>10000) {
        n=10000;
        printf("n=%d",n);
    }
    int i, j;
    struct timespec ini,fin; double transcurrido;

```



```

// 2. Creación e inicialización de vector y matriz
// Creación
int *v1,*v2;
v1 = (int*) malloc(n*sizeof(double));
v2 = (int*) malloc(n*sizeof(double));

int **M;
M = (int**) malloc(n*sizeof(int*));
for(i=0;i<n;i++)
    M[i] = (int*)malloc(n*sizeof(int));

// Inicialición
for(i=0;i<n;i++)
    v1[i]=i;

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

// 3. Impresión de vector y matriz
#ifdef TIMES
#ifdef PRINTF_ALL
    printf("Vector inicial:\n");
    for (i=0; i<n; i++) printf("%d ",v1[i]);
    printf("\n");

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

// 4. Cálculo resultado
clock_gettime(CLOCK_REALTIME,&ini);
#pragma omp parallel for default(none) private(i,j) shared(n,v1,M,v2)
for (i=0; i<n; i++) {
    v2[i]=0;
    for (j=0; j<n; j++) {
        v2[i]+=M[i][j]*v1[j];
    }
}
clock_gettime(CLOCK_REALTIME,&fin);
transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

// 5. Impresión de vector resultado
#ifdef TIMES
    printf("%d %11.9f\n",n,transcurrido);
#else
#ifdef PRINTF_ALL
    printf("Tiempo: %11.9f\n",transcurrido);
    printf("Vector resultado (M x v1):\n");
    for (i=0; i<n; i++) printf("%d ",v2[i]);
    printf("\n");

```



```

    #else
        printf("Tiempo: \%11.9f\n",transcurrido);
        printf("v2[0]: %d, v2[n-1]: %d\n",v2[0],v2[n-1]);
    #endif
#endif

// 6. Eliminar de memoria
free(M);
free(v1);
free(v2);
}

```

CÓDIGO FUENTE: pmv-OpenMP-b.c

```

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

#define TIMES
// #define PRINTF_ALL

main(int argc, char **argv) {
    // 1. Lectura valores de entrada
    if(argc < 2) {
        fprintf(stderr,"Falta num\n");
        exit(-1);
    }
    int n = atoi(argv[1]);
    if (n>10000) {
        n=10000;
        printf("n=%d",n);
    }
    int i, j, sumalocal;
    struct timespec ini,fin; double transcurrido;

    // 2. Creación e inicialización de vector y matriz
    // Creación
    int *v1,*v2;
    v1 = (int*) malloc(n*sizeof(double));
    v2 = (int*) malloc(n*sizeof(double));

    int **M;
    M = (int**) malloc(n*sizeof(int*));
    for(i=0;i<n;i++)
        M[i] = (int*)malloc(n*sizeof(int));

    // Inicialición
    for(i=0;i<n;i++)
        v1[i]=i;

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

    // 3. Impresión de vector y matriz
    #ifndef TIMES
    #ifdef PRINTF_ALL
        printf("Vector inicial:\n");
    #endif
    #endif
}

```

```

        for (i=0; i<n; i++) printf("%d ",v1[i]);
        printf("\n");

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

// 4. Cálculo resultado
clock_gettime(CLOCK_REALTIME,&ini);
for (i=0; i<n; i++) {
    v2[i]=0;
    #pragma omp parallel default(none) shared(i,n,v1,M,v2) private(sumalocal)
    {
        sumalocal=0;
        #pragma omp for schedule(static)
        for (j=0; j<n; j++) {
            sumalocal+=M[i][j]*v1[j];
        }
        #pragma omp atomic
        v2[i]+=sumalocal;
    }
}
clock_gettime(CLOCK_REALTIME,&fin);
transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

// 3. Impresión de vector resultado
#ifdef TIMES
    printf("%d %11.9f\n",n,transcurrido);
#else
    #ifdef PRINTF_ALL
        printf("Tiempo: %11.9f\n",transcurrido);
        printf("Vector resultado (M x v1):\n");
        for (i=0; i<n; i++) printf("%d ",v2[i]);
        printf("\n");
    #else
        printf("Tiempo: %11.9f\n",transcurrido);
        printf("v2[0]: %d, v2[n-1]: %d\n",v2[0],v2[n-1]);
    #endif
#endif
#endif

// 6. Eliminar de memoria
free(M);
free(v1);
free(v2);
}

```

RESPUESTA:pmv-OpenMP-a (por filas)

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 0 \cdot 0 + 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 3 \\ 0 \cdot 4 + 1 \cdot 5 + 2 \cdot 6 + 3 \cdot 7 \\ 0 \cdot 8 + 1 \cdot 9 + 2 \cdot 10 + 3 \cdot 11 \\ 0 \cdot 12 + 1 \cdot 13 + 2 \cdot 14 + 3 \cdot 15 \end{pmatrix}$$

Diagram illustrating the row-major calculation of the matrix-vector product. The matrix is multiplied by the vector $\begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \end{pmatrix}$. The result is shown as a column vector where each element is the dot product of a row of the matrix with the vector, labeled as thread 0, thread 1, thread 2, and thread 3 respectively.

pmv-OpenMP-b (por columnas)

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 0 \cdot 0 + 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 3 \\ 0 \cdot 4 + 1 \cdot 5 + 2 \cdot 6 + 3 \cdot 7 \\ 0 \cdot 8 + 1 \cdot 9 + 2 \cdot 10 + 3 \cdot 11 \\ 0 \cdot 12 + 1 \cdot 13 + 2 \cdot 14 + 3 \cdot 15 \end{pmatrix}$$

Diagram illustrating the column-major calculation of the matrix-vector product. The matrix is multiplied by the vector $\begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \end{pmatrix}$. The result is shown as a column vector where each element is the dot product of a row of the matrix with the vector, labeled as thread 0, thread 1, thread 2, and thread 3 respectively.

CAPTURAS DE PANTALLA:

```

fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P2$ ./pmv-OpenMP-a 4
Tiempo: 0.000003964
v2[0]: 14, v2[n-1]: 86
fblupi@fblupi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P2$ ./pmv-OpenMP-b 4
Tiempo: 0.000018507
v2[0]: 14, v2[n-1]: 86

```

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`

```

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

#define TIMES
// #define PRINTF_ALL

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

```

```

    exit(-1);
}
int n = atoi(argv[1]);
if (n>10000) {
    n=10000;
    printf("n=%d",n);
}
int i, j, sumalocal;
struct timespec ini,fin; double transcurrido;

// 2. Creación e inicialización de vector y matriz
// Creación
int *v1,*v2;
v1 = (int*) malloc(n*sizeof(double));
v2 = (int*) malloc(n*sizeof(double));

int **M;
M = (int**) malloc(n*sizeof(int*));
for(i=0;i<n;i++)
    M[i] = (int*)malloc(n*sizeof(int));

// Inicialición
for(i=0;i<n;i++)
    v1[i]=i;

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

// 3. Impresión de vector y matriz
#ifdef TIMES
#ifdef PRINTF_ALL
    printf("Vector inicial:\n");
    for (i=0; i<n; i++) printf("%d ",v1[i]);
    printf("\n");

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

// 4. Cálculo resultado
clock_gettime(CLOCK_REALTIME,&ini);
for (i=0; i<n; i++) {
    sumalocal=0;
    #pragma omp parallel for reduction(+:sumalocal)
    for (j=0; j<n; j++) {
        sumalocal+=M[i][j]*v1[j];
    }
    v2[i]=sumalocal;
}
clock_gettime(CLOCK_REALTIME,&fin);
transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

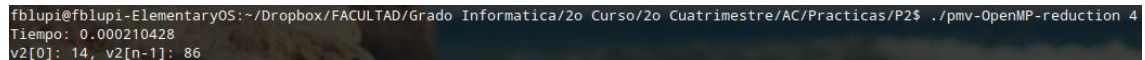
```

```
// 5. Impresión de vector resultado
#ifdef TIMES
    printf("%d %11.9f\n",n,transcurrido);
#else
    #ifdef PRINTF_ALL
        printf("Tiempo: %11.9f\n",transcurrido);
        printf("Vector resultado (M x v1):\n");
        for (i=0; i<n; i++) printf("%d ",v2[i]);
        printf("\n");
    #else
        printf("Tiempo: %11.9f\n",transcurrido);
        printf("v2[0]: %d, v2[n-1]: %d\n",v2[0],v2[n-1]);
    #endif
#endif

// 6. Eliminar de memoria
free(M);
free(v1);
free(v2);
}
```

RESPUESTA: En un principio intenté hacerlo sin la variable `sumalocal`, usando `v2[i]` en el `reduction`, pero falló al compilar y añadí esta variable.

CAPTURAS DE PANTALLA:



```
fb1upi@fb1upi-ElementaryOS:~/Dropbox/FACULTAD/Grado Informatica/2o Curso/2o Cuatrimestre/AC/Practicas/P2$ ./pmv-OpenMP-reduction 4
Tiempo: 0.000210428
v2[0]: 14, v2[n-1]: 86
```

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). Usar `-O2` al compilar.

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

Tiempos y ganancia en PC

1. TIEMPOS

pmv-OpenMP-a

	cores			
tamaño	1	2	3	4
100	0,000068331	0,00017722	0,00411991	0,01007688
1000	0,003245161	0,0021832	0,0016088	0,00125108
10000	0,224953663	0,15776021	0,12475877	0,10695591

pmv-OpenMP-b

	cores			
tamaño	1	2	3	4
100	0,000208147	0,00048075	0,01091477	0,0229232
1000	0,003032945	0,00399684	0,00529901	0,00746249
10000	0,192225877	0,14822163	0,12326514	0,10897176

pmv-OpenMP-reduction

	cores			
tamaño	1	2	3	4
100	0,000178394	0,000414404	0,00899875	0,018816192
1000	0,003542121	0,004516328	0,005836953	0,008071535
10000	0,193374741	0,152600852	0,126606216	0,109787544

2. GANANCIAS

pmv-OpenMP-a

		cores		
tamaño	1	2	3	4
100	1	0,38557574	0,016585563	0,00678097
1000	1	1,486425057	2,017136415	2,593894843
10000	1	1,425921429	1,803109074	2,103237395

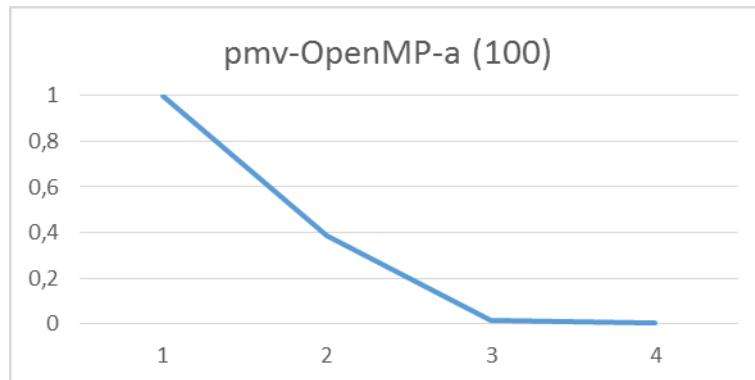
pmv-OpenMP-b

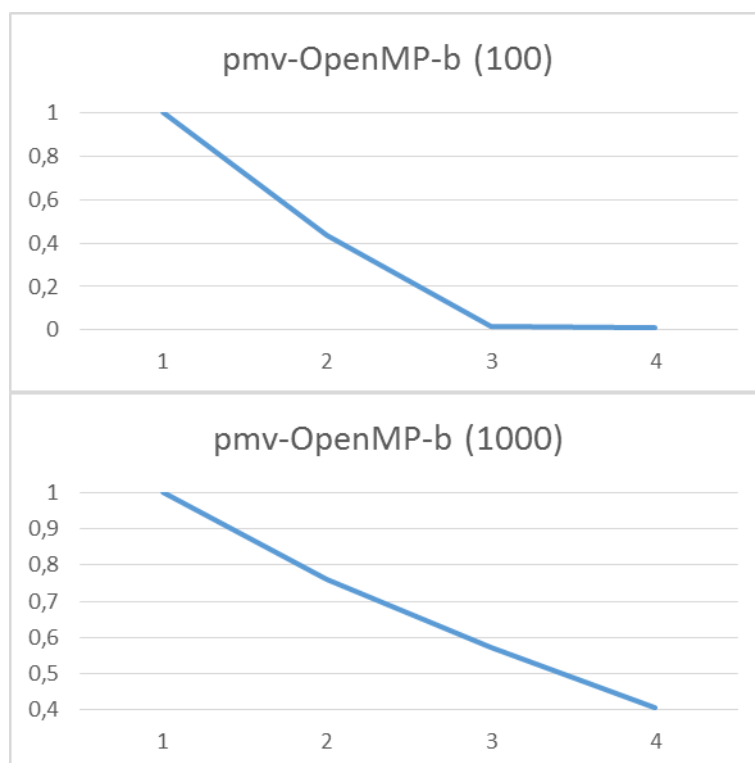
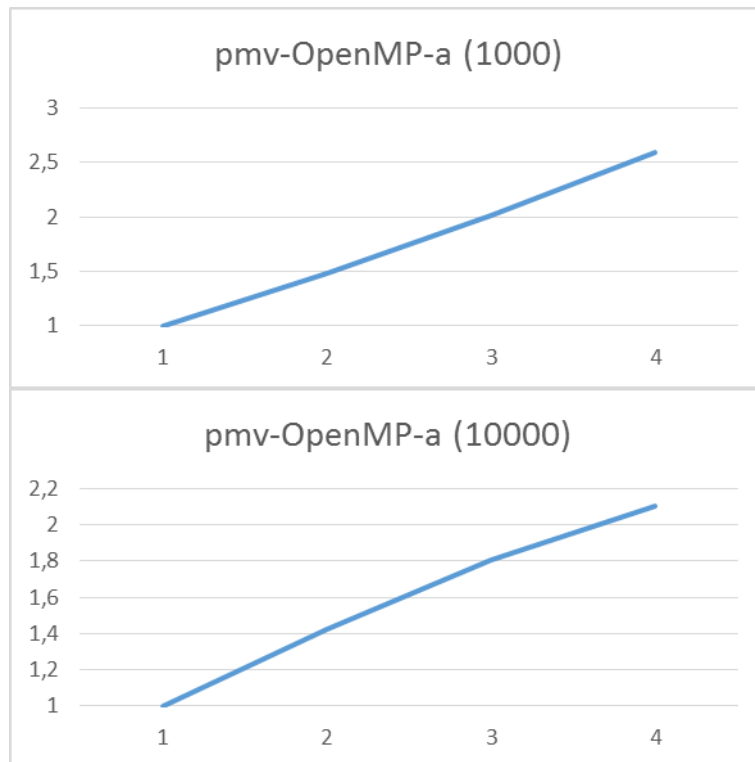
		cores		
tamaño	1	2	3	4
100	1	0,432961768	0,01907016	0,009080169
1000	1	0,758835691	0,572360343	0,406425479
10000	1	1,29688145	1,559450469	1,76399717

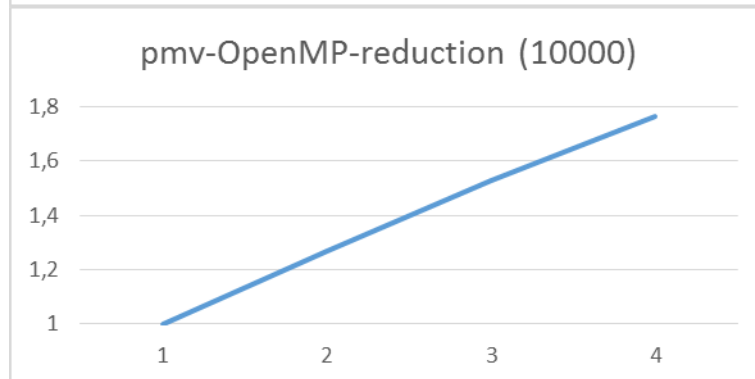
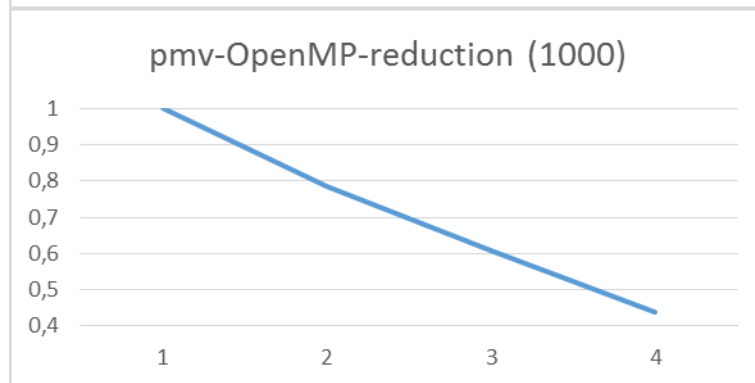
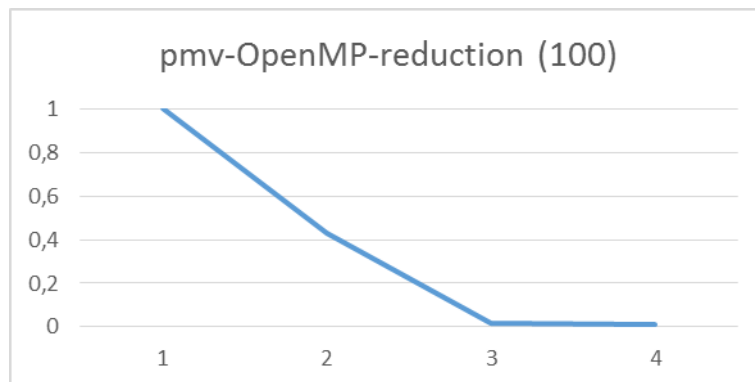
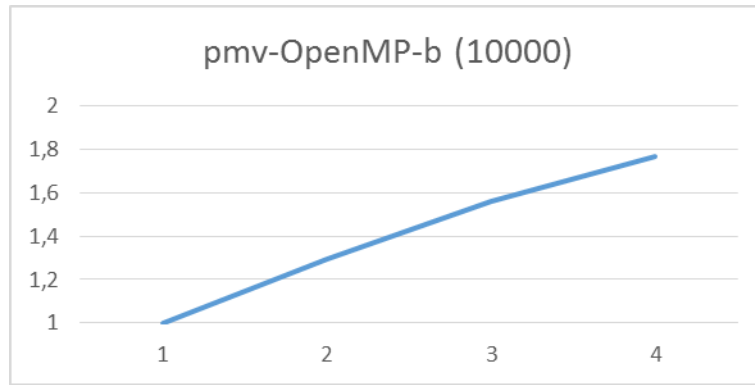
pmv-OpenMP-reduction

		cores		
tamaño	1	2	3	4
100	1	0,430483753	0,019824353	0,009480898
1000	1	0,784292148	0,60684416	0,438841072
10000	1	1,267193058	1,527371619	1,761354098

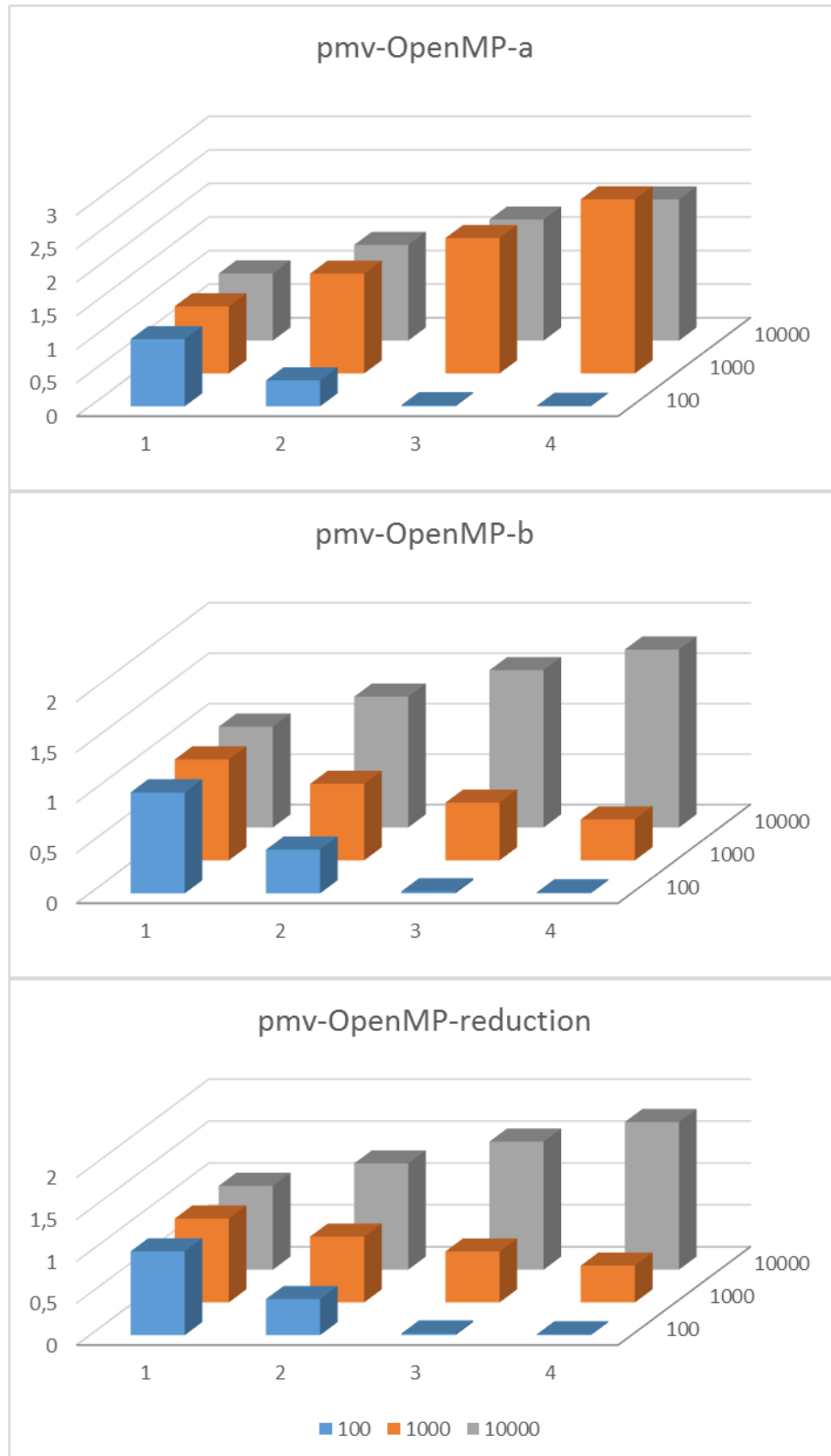
3. GRÁFICOS INDEPENDIENTES







4. GRÁFICOS COMPARATIVOS



Tiempos y ganancia en ATCGRID

1. TIEMPOS

Pmv-OpenMP-a

tamaño	cores											
	1	2	3	4	5	6	7	8	9	10	11	12
100	4E-05	7E-05	0,0001	0,0002	0,0002	0,0002	0,0003	0,0003	0,0003	0,0003	0,0004	0,0004
1000	0,002	0,0013	0,0009	0,0008	0,0007	0,0006	0,0006	0,0006	0,0006	0,0006	0,0006	0,0006
10000	0,141	0,0768	0,0532	0,0424	0,0355	0,032	0,0293	0,028	0,0271	0,0265	0,0263	0,026

Pmv-OpenMP-b

tamaño	cores											
	1	2	3	4	5	6	7	8	9	10	11	12
100	0,0001	0,0002	0,0004	0,0006	0,0007	0,0008	0,0008	0,0009	0,0009	0,0009	0,0009	0,001
1000	0,0028	0,0037	0,0043	0,0048	0,0053	0,0056	0,0058	0,0061	0,0064	0,0067	0,0068	0,0071
10000	0,141	0,1007	0,085	0,0787	0,0745	0,0723	0,071	0,0701	0,0699	0,0696	0,069	0,069

Pmv-OpenMP-reduction

tamaño	cores											
	1	2	3	4	5	6	7	8	9	10	11	12
100	9E-05	0,0002	0,0004	0,0005	0,0006	0,0007	0,0007	0,0007	0,0008	0,0008	0,0009	0,0009
1000	0,0025	0,0032	0,0037	0,004	0,0042	0,0044	0,0045	0,0047	0,0049	0,005	0,0052	0,0054
10000	0,1364	0,0941	0,0763	0,0679	0,0651	0,0636	0,0634	0,0631	0,0629	0,0625	0,0621	0,062

2. GANANCIA

Pmv-OpenMP-a

tamaño	cores											
	1	2	3	4	5	6	7	8	9	10	11	12
100	1	0,4943	0,287	0,1993	0,166	0,1415	0,1341	0,1197	0,1068	0,1005	0,0993	0,0959
1000	1	1,5581	2,1051	2,5514	2,9255	3,1641	3,379	3,4951	3,5542	3,5741	3,5925	3,6182
10000	1	1,8353	2,6486	3,323	3,9678	4,4019	4,8175	5,0408	5,2062	5,3114	5,3686	5,4336

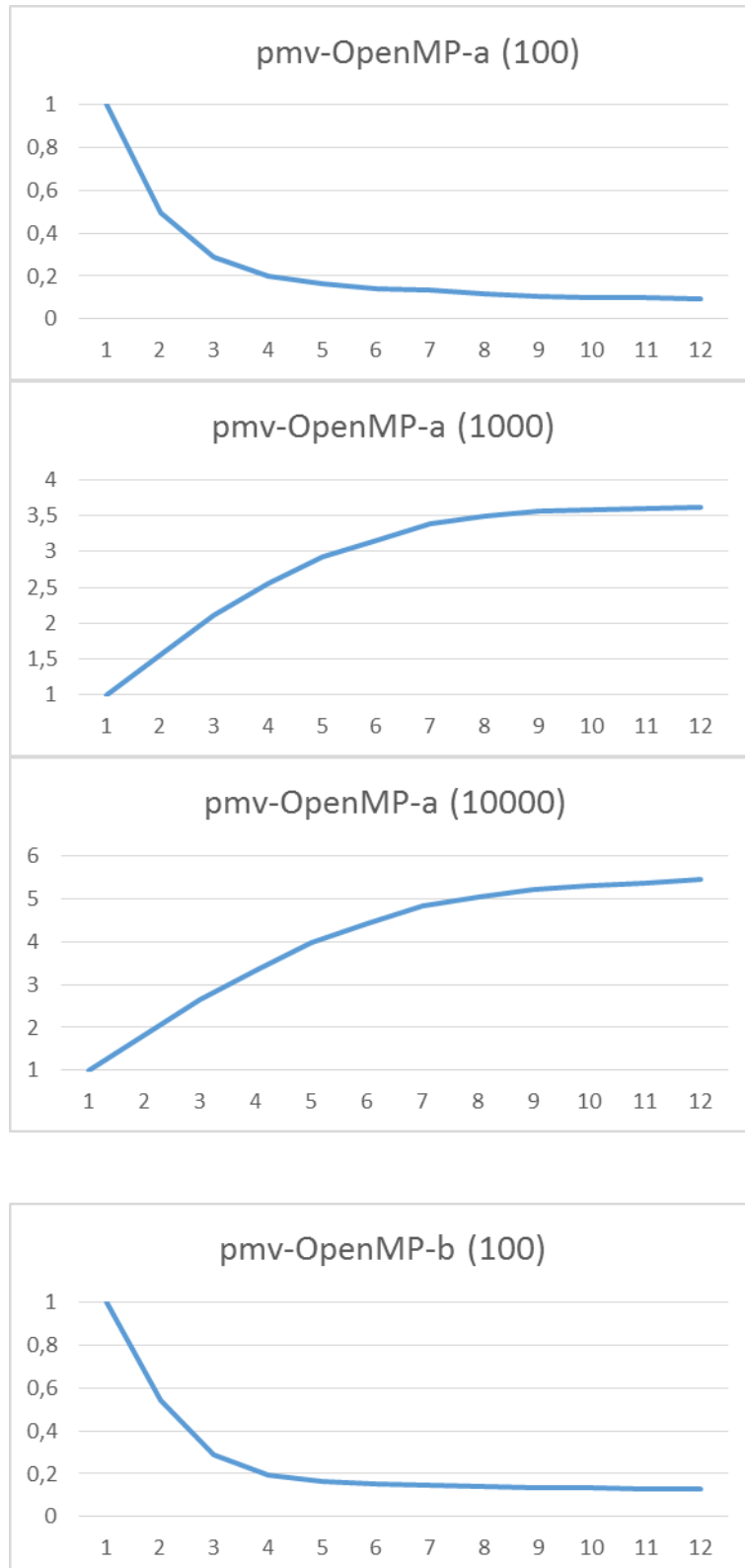
Pmv-OpenMP-b

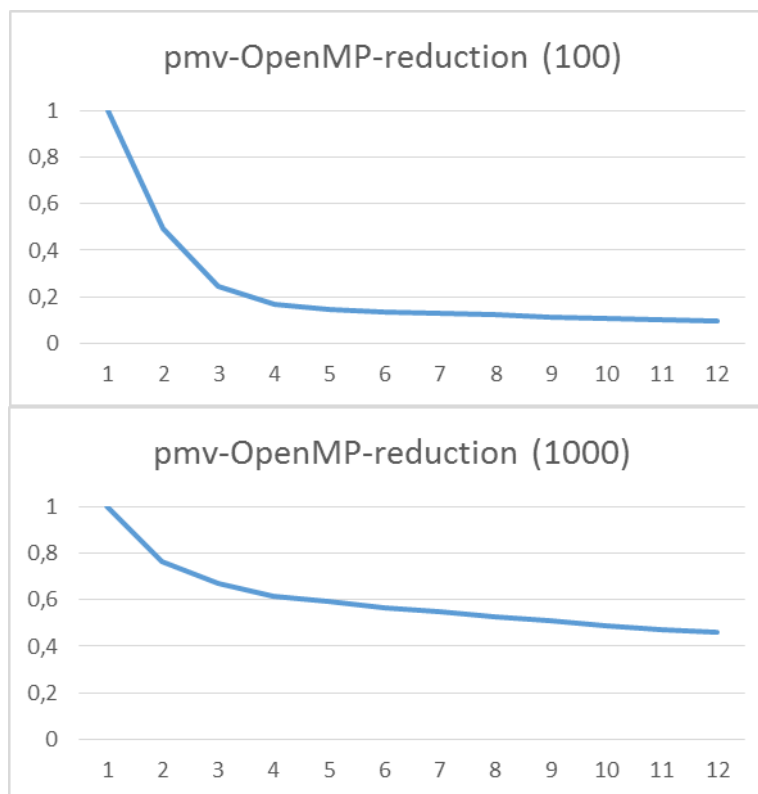
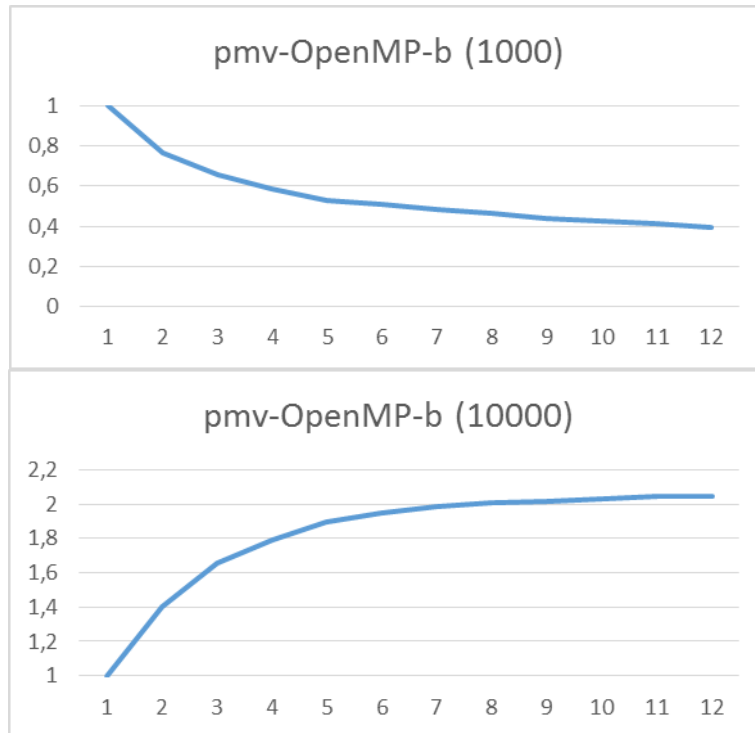
tamaño	cores											
	1	2	3	4	5	6	7	8	9	10	11	12
100	1	0,5414	0,2855	0,1966	0,1647	0,1502	0,148	0,1409	0,1338	0,1331	0,1306	0,1284
1000	1	0,7654	0,6556	0,5839	0,5312	0,507	0,4849	0,4639	0,4411	0,4226	0,414	0,3958
10000	1	1,4005	1,6593	1,7903	1,892	1,9507	1,9843	2,0107	2,0171	2,0264	2,0428	2,0441

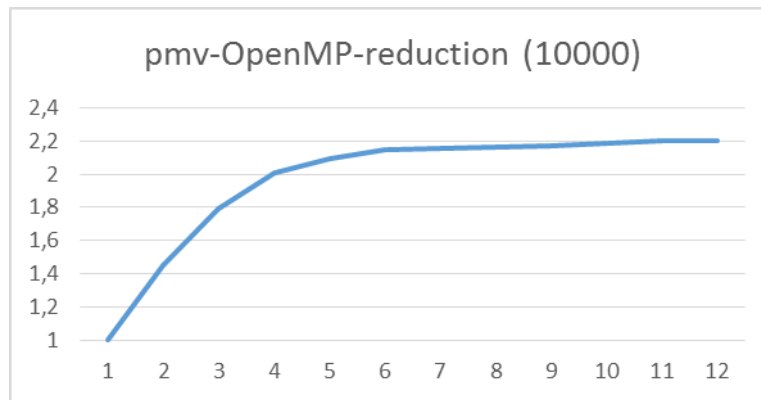
Pmv-OpenMP-reduction

tamaño	cores											
	1	2	3	4	5	6	7	8	9	10	11	12
100	1	0,4935	0,2468	0,1686	0,1453	0,1333	0,1268	0,1212	0,1145	0,107	0,1022	0,0972
1000	1	0,7643	0,6685	0,6144	0,5919	0,5673	0,5498	0,5267	0,5083	0,4895	0,4734	0,4589
10000	1	1,4498	1,7889	2,0097	2,0964	2,1451	2,1529	2,1623	2,1702	2,184	2,1978	2,1998

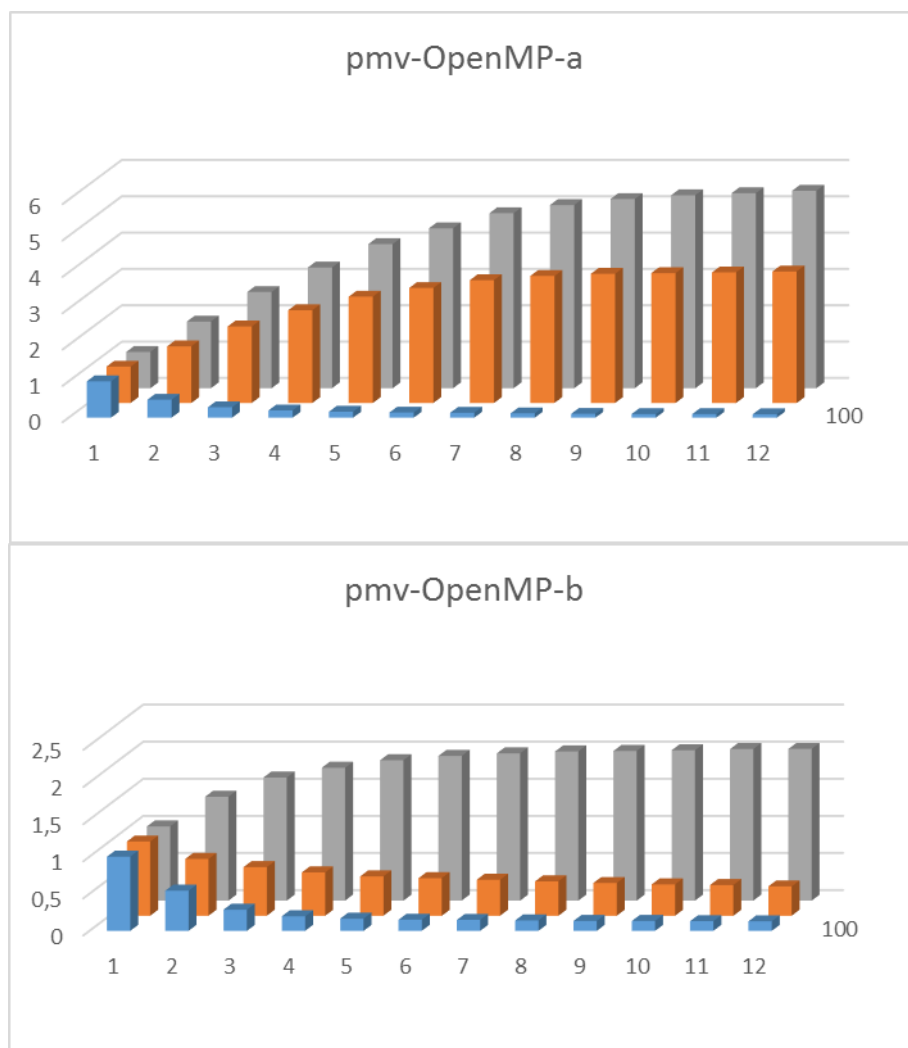
3. GRÁFICOS INDEPENDIENTES

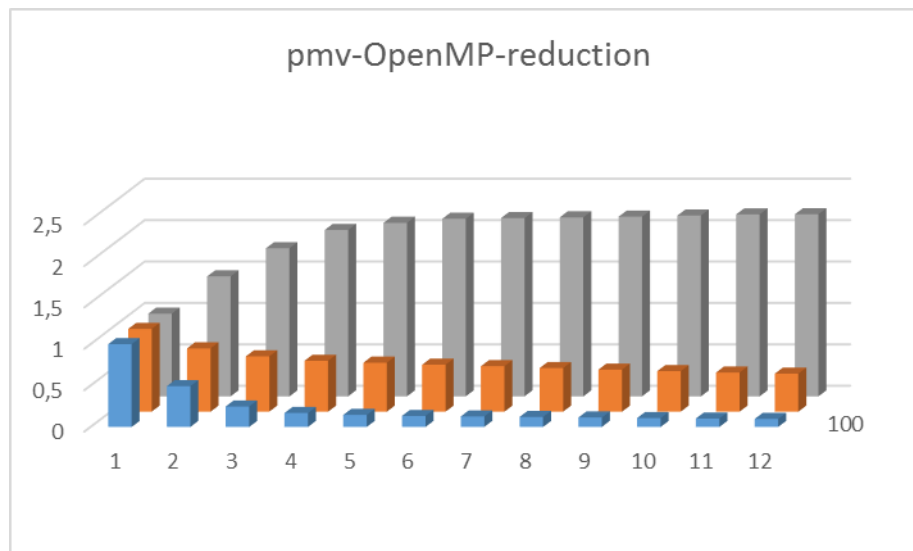






4. GRÁFICOS COMPARATIVOS





COMENTARIOS SOBRE LOS RESULTADOS: Con los gráficos obtenidos con los tiempos del PC apenas se pueden sacar conclusiones. Tan solo que para un tamaño pequeño de entrada (100) el tiempo empeora a más procesadores incorpores. Esto puede ser debido al coste de tiempo que hay en la creación y comunicación de hebras. Este comportamiento se repite en la paralelización por columnas que se hace en el b y en el reduction, en ambas hay que crear una variable privada para cada hebra y realizar en exclusión mutua la suma de todas estas variables privadas. Esta operación constituye una pérdida de tiempo que se puede ver cuando el tamaño de entrada es 1000. Ya para 10000 vemos claramente que se mejoran los tiempos y en un principio parece que de forma lineal. Para ver cómo se comporta esta ganancia, es preciso verla en más procesadores. Para ello tomamos tiempos en ATCGRID con 1-12 procesadores. Vemos que el comportamiento que ocurría en el PC se repite, pero en esta ocasión podemos determinar con más seguridad el comportamiento de la ganancia.

En paralelización por filas:

- Tamaño 100: Hay una asíntota vertical en 0,01 y a más procesadores, menos diferencia hay entre los procesadores P_n y P_{n-1} .
- Tamaño 1000: Lo que en el PC parecía un crecimiento lineal pasa a verse logarítmico. Con la asíntota más o menos en el 3,6.
- Tamaño 10000: El crecimiento sigue siendo logarítmico pero se ve en la tabla comparativa que la ganancia es mayor que cuando el tamaño es 1000, la asíntota está aproximadamente en 5,5.

En paralelización por columnas (tanto apartado b como reduction cuyo comportamiento es similar):

- Tamaño 100: Mismo comportamiento que en la paralelización por filas.
- Tamaño 1000: Decrecimiento no tan brusco como el que se produce con tamaño 100. Es decir, la diferencia entre el procesador P_1 y el P_{12} es menor.
- Tamaño 10000: Mismo comportamiento que en la paralelización por filas.

En conclusión, vemos que hay un límite en el que la mejora al incorporar más procesadores deja de ser tan superior, si hubiésemos incorporado tiempos con 24 procesadores, los tiempos entre P_{12} y P_{24} no habrían variado demasiado.