Cuaderno de prácticas
de Arquitectura de Computadores
Grado en Ingeniería Informática

Memoria
Bloque Práctico 2

Alumno: Manuel Jesús García Manday
DNI: 48893432D
Grupo: *D3*

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)`.
   RESPUESTA: Nos da un error de compilación ya que nos obliga a identificar explícitamente el ambito de cada variable, y solo hemos los hemos echo con el vector a, pero no con la variable n, ya que la varibale i viene ya identificado su ambito de privada.

   CÓDIGO FUENTE: `shared-clauseModificado.c`

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#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 shared(a) shared(n) private(i) 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:

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario2 $ ./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.
   RESPUESTA: Si la inicializamos fuera, la copia de esa variable para cada hebra tomará basura como valor de inicialización por lo que la suma no se haría correctamente.

   CÓDIGO FUENTE: `private-clauseModificado.c`

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#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;

  suma=0;
  #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 suma = %d\n", suma);
}
```

   CAPTURAS DE PANTALLA:

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario2 $ ./private-clause
thread 1 suma a[4] / thread 1 suma a[5] / thread 1 suma a[6] / thread 0 suma a[0] / thread 0 suma a[1] / thread 0 su
ma a[2] / thread 0 suma a[3] /
* thread 1 suma= 4196463
* thread 0 suma= -1093455658
 suma = 0
```

3. ¿Qué ocurre si en `private-clause.c` se elimina la cláusula `private(suma)`? ¿A qué cree que es debido?
   RESPUESTA: La variable pasaría a ser compartida y el resultado de la suma de cada hebra puede que no coincida con el total, ya que podría darse el caso de que varias hebras leyeran y escribieran a la vez, por lo que no habría consistencia en el resultado. Para solucionarlo añadimos una variable local a cada hebra y la suma de cada una de ellas mediante la clausula atomic.

   CÓDIGO FUENTE: `private-clauseModificado3.c`

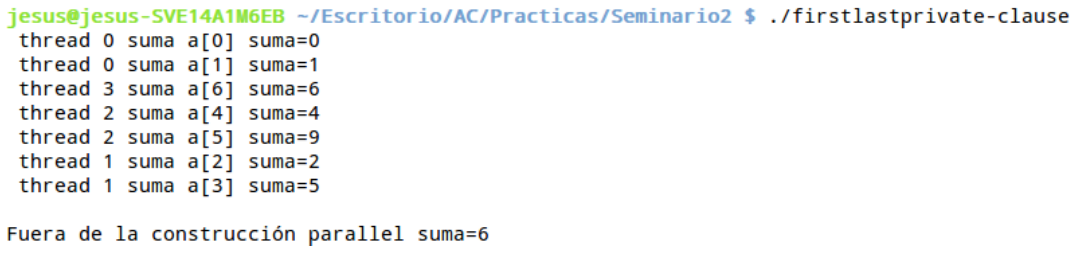
```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

     #pragma omp atomic
     suma += sumaLocal;
   }
   printf("\n suma = %d\n", suma);
}
```

   CAPTURAS DE PANTALLA:

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario2 $ ./private-clause
thread 1 suma a[4] / thread 1 suma a[5] / thread 1 suma a[6] / thread 0 suma a[0] / thread 0 suma a[1] / thread 0 su
ma a[2] / thread 0 suma a[3] /
* thread 1 suma= 21
* thread 0 suma= 21
 suma = 21
```

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: No, en este caso es porque la ultima hebra solo realiza una iteración, por lo que no realiza ninguna suma, pero si la última hebra realizara las dos ultimas iteraciones el resultado sería distinto.

   CAPTURAS DE PANTALLA:



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: Que el valor desde teclado solo lo tendrá en su variable privada la hebra que ejecute el single, por lo que al imprimir los valores del vector despues del codigo paralelizado solo las posiciones en las que esa hebra iteró tendra almacenado el valor de entrada, mientras que el resto contendrá basura.

   CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#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 ejecutadda por el thread %d\n",
omp_get_thread_num());
     }

     #pragma omp for
     for(i = 0; i < n; i++)
        b[i] = a;
  }

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

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

CAPTURAS DE PANTALLA:

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario2 $ ./copyprivate-clause

Introduce valor de inicialización a: 5

Single ejecutadda por el thread 1
Despues de region parallel:
b[0]=32767      b[1]=32767      b[2]=32767      b[3]=5  b[4]=5  b[5]=5  b[6]=32593      b[7]=32593      b[8
]=32593
```

6. En el ejemplo `reduction-clause.c` sustituya `suma=0` por `suma=10`. ¿Qué resultado se imprime ahora? Justifique el resultado
   RESPUESTA: Imprimirá el resultado de antes mas 10, es decir, si con 20 iteraciones el resultado es 190, pues imprimirá 200, ya que la variable global suma la hemos inicializado a 10, y las hebras que trabajen y calculen la suma parcial, la irán sumando a la que ya contenía.

   CÓDIGO FUENTE: `reduction-clauseModificado.c`

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#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:

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario2 $ ./reduction-clause 20
Tras '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.

   RESPUESTA:

   CÓDIGO FUENTE: `reduction-clauseModificado7.c`

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

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
  {
    int sumaLocal = 0;
    #pragma omp for
```

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

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

CAPTURAS DE PANTALLA:

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario2 $ ./reduction-clause 20
Tras 'parallel' suma=190
```

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

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

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

CÓDIGO FUENTE: `pmv-secuencial.c`

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

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

            int n, i, j;
            struct timespec cgt1,cgt2;
            double ncgt;
            n = atoi(argv[1]); /* tomamos el tamaño de filas y
columnas de la matriz */
            int m[n][n], v1[n], v3[n]; /* nos creamos la matriz y el
vector */

            srand(time(NULL));
            /* inicializamos la matriz y el vector */
            for(i = 0; i < n; i++){
                        v1[i] = (rand() % 3) + 1;
                        v3[i] = 0;
                        for(j = 0; j < n; j++)
                                    m[i][j] = rand() % (3+1);
```

```
            }

            clock_gettime(CLOCK_REALTIME,&cgt1);
            /* realizamos la multiplicacion */
            for(i = 0; i < n; i++){
                        for(j = 0; j < n; j++)
                                    v3[i] += m[i][j] * v1[j];
            }

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

            printf("Tiempo(seg.):%11.9f\t / Tamaño Vector:
%u\n",ncgt,n);
            printf("Vector resultante del producto: ");
            for(i = 0; i < n; i++)
                        printf("%d ", v3[i]);
            printf("\n");

}
```

CAPTURAS DE PANTALLA:

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario2/BloquePractico2_GarciaMandayManuelJesus $ ./pmv-secuencial 8
Tiempo(seg.):0.000000816         / Tamaño Vector:8
Vector resultante del producto: 21 33 29 27 25 18 20 38
```

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario2/BloquePractico2_GarciaMandayManuelJesus $ ./pmv-secuencial 11
Tiempo(seg.):0.000001133         / Tamaño Vector:11
Vector resultante del producto: 20 13 19 23 17 13 20 19 23 14 17
```

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).

    CÓDIGO FUENTE : `pmv-OpenMP-a.c`
    (RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

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

            int n, i, j;
            struct timespec cgt1,cgt2;
            double ncgt;
            n = atoi(argv[1]); /* tomamos el tamaño de filas y
columnas de la matriz */
            int m[n][n], v1[n], v3[n]; /* nos creamos la matriz y el
vector */

            srand(time(NULL));
            /* inicializamos la matriz y el vector */
            #pragma omp parallel for private(i) private(j) shared(n)
shared(m) shared(v1) shared(v3) default(none)
            for(i = 0; i < n; i++){ ;
                        v1[i] = (rand() % 3) + 1;
                        v3[i] = 0;
                        for(j = 0; j < n; j++)
                                    m[i][j] = rand() % (3+1);
            }

            clock_gettime(CLOCK_REALTIME,&cgt1);
            /* realizamos la multiplicacion */
            #pragma omp parallel for private(i) private(j) shared(n)
shared(m) shared(v1) shared(v3) default(none)
            for(i = 0; i < n; i++){
                        for(j = 0; j < n; j++)
                                    v3[i] += m[i][j] * v1[j];
            }

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

            printf("Tiempo(seg.):%11.9f\t / Tamaño Vector:
%u\n",ncgt,n);
            printf("Vector resultante del producto: ");
            for(i = 0; i < n; i++)
                        printf("%d ", v3[i]);
            printf("\n");
}
```

CÓDIGO FUENTE: pmv-OpenMP-b.c
(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
/* Tipo de letra Courier New. Tamaño 10.*/
```

```
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

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

            int n, i, j;
            struct timespec cgt1,cgt2;
            double ncgt;
            n = atoi(argv[1]); /* tomamos el tamaño de filas y
columnas de la matriz */
            int m[n][n], v1[n], v3[n]; /* nos creamos la matriz y el
vector */

            srand(time(NULL));
            /* inicializamos la matriz y el vector */
            for(i = 0; i < n; i++){
                        v1[i] = (rand() % 3) + 1;
                        v3[i] = 0;
                        #pragma omp parallel for shared(i) private(j)
shared(n) shared(m) default(none)
                        for(j = 0; j < n; j++)
                                    m[i][j] = rand() % (3+1);
            }


            clock_gettime(CLOCK_REALTIME,&cgt1);
            /* realizamos la multiplicacion */
            for(i = 0; i < n; i++){
                        #pragma omp parallel {
                           #pragma omp for shared(i) private(j)
shared(n) shared(m) shared(v1) shared(v3) default(none)

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

                        #pragma omp atomic
                             v3[i] += sumLoc ;

            }

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

            printf("Tiempo(seg.):%11.9f\t / Tamaño Vector:
%u\n",ncgt,n);
            printf("Vector resultante del producto: ");
            for(i = 0; i < n; i++)
                        printf("%d ", v3[i]);
```

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

RESPUESTA: En la ejecucion de la segunda version me daba resultados errones ya que al expresar el ambito de la variable del primer for (i) puse que era de ambito privada, cuando su verdadero ambito es shared, ya que esa variable solo la debe tener la hebra principal de programa.

CAPTURAS DE PANTALLA:

a)

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario2/BloquePractico2_GarciaMandayManuelJesus $ ./pmv-OpenMP-a 8
Tiempo(seg.):0.000006645         / Tamaño Vector:8
Vector resultante del producto: 27 25 29 31 29 28 25 32
```

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario2/BloquePractico2_GarciaMandayManuelJesus $ ./pmv-OpenMP-a 11
Tiempo(seg.):0.000018041         / Tamaño Vector:11
Vector resultante del producto: 36 11 21 34 34 32 36 27 31 46 31
```

b)

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario2/BloquePractico2_GarciaMandayManuelJesus $ ./pmv-OpenMP-b 8
Tiempo(seg.):0.000028925         / Tamaño Vector:8
Vector resultante del producto: 19 16 27 41 21 23 31 33
```

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario2/BloquePractico2_GarciaMandayManuelJesus $ ./pmv-OpenMP-b 11
Tiempo(seg.):0.000044341         / Tamaño Vector:11
Vector resultante del producto: 30 26 26 25 16 26 21 19 23 25 28
```

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-OpenMP-reduction.c`
(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
/* Tipo de letra Courier New. Tamaño 10.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

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

```
#include <time.h>

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

            int n, i, j, suma;
            struct timespec cgt1,cgt2;
            double ncgt;
            n = atoi(argv[1]); /* tomamos el tamaño de filas y
columnas de la matriz */
            int m[n][n], v1[n], v3[n]; /* nos creamos la matriz y el
vector */

            srand(time(NULL));
            /* inicializamos la matriz y el vector */
            for(i = 0; i < n; i++){
                        v1[i] = (rand() % 3) + 1;
                        v3[i] = 0;
                        #pragma omp parallel for shared(i) private(j)
shared(n) shared(m) default(none)
                        for(j = 0; j < n; j++)
                                    m[i][j] = rand() % (3+1);
            }

            clock_gettime(CLOCK_REALTIME,&cgt1);
            /* realizamos la multiplicacion */
            for(i = 0; i < n; i++){
                        #pragma omp parallel for shared(i) private(j)
shared(n) shared(m) shared(v1) shared(v3) reduction(+:suma)
default(none)
                        for(j = 0; j < n; j++){
                                    suma = m[i][j] * v1[j];
                        }
                        v3[i] += suma;
                        suma = 0;
            }
            clock_gettime(CLOCK_REALTIME,&cgt2);
            ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
                        (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/
(1.e+9));

            printf("Tiempo(seg.):%11.9f\t / Tamaño Vector:
%u\n",ncgt,n);
            printf("Vector resultante del producto: ");
            for(i = 0; i < n; i++)
                        printf("%d ", v3[i]);
            printf("\n");
}
```

RESPUESTA:

CAPTURAS DE PANTALLA:

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario2/BloquePractico2_GarciaMandayManuelJesus $ ./pmv-OpenMP-reduction 8
Tiempo(seg.):0.000026027        / Tamaño Vector:8
Vector resultante del producto: 14 4 11 10 6 14 16 15
```

```
jesus@jesus-SVE14A1M6EB ~/Escritorio/AC/Practicas/Seminario2/BloquePractico2_GarciaMandayManuelJesus $ ./pmv-OpenMP-reduction 11
Tiempo(seg.):0.000036902          / Tamaño Vector:11
Vector resultante del producto: 8 10 15 10 5 7 11 16 4 16 6
```

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).

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):

pclocal 1-4 hebras. Tiempos de ejecución.

N	pmv-OpenMP-a	pmv-OpenMP-b	pmv-OpenMP-reduction
100	0.000030830	0.000138489	0.000114995
1000	0.000689706	0.002174291	0.001556106
10000	0.064748690	0.066206061	0.062886644

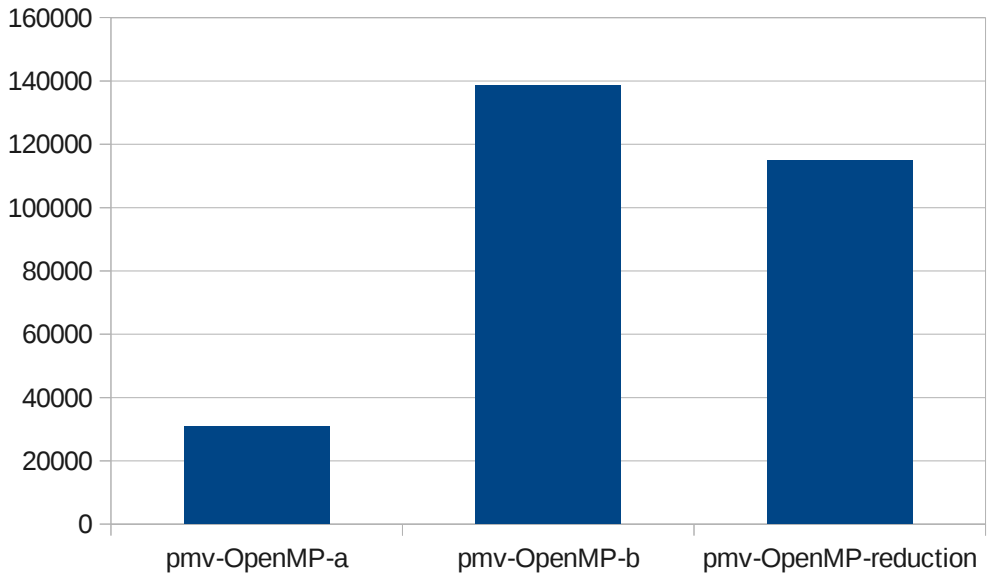
Atcgrid 1-12 hebras. Tiempo de ejecución.

N	pmv-OpenMP-a	pmv-OpenMP-b	pmv-OpenMP-reduction
100	0.000023948	0.000747113	0.000621145
1000	0.000160561	0.007850292	0.006376843
10000	0.023152428	0.096411167	0.075829123

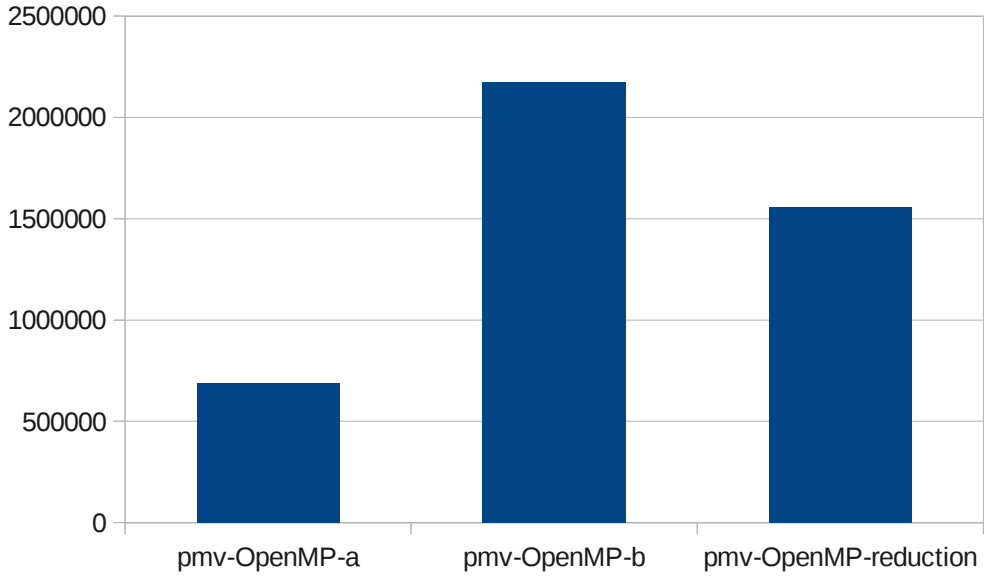
COMENTARIOS SOBRE LOS RESULTADOS: Podemos ver como en ambas máquinas si realizamos el producto de la matriz por el

vector paralelizado por filas es más rápido que si lo realizamos por columnas (para cualquier valor de N). Eso es debido en parte a la sincronización causada por la cláusula atomic en la version b, y el reduction en la siguiente, ya que en esas versiones existen puntos de sincronización que en la versión a no hay. En la versión b tenemos el atomic, por el que las hebras deben esperarse mutuamente para realizar de forma exclusiva las operaciones pertinentes, mientras que en la versión siguiente con el reduction, una vez echo el trabajo por separado cada hebra, se tiene que unir en uno solo (resultado) lo que también produce retardo en el tiempo total de ejecución.

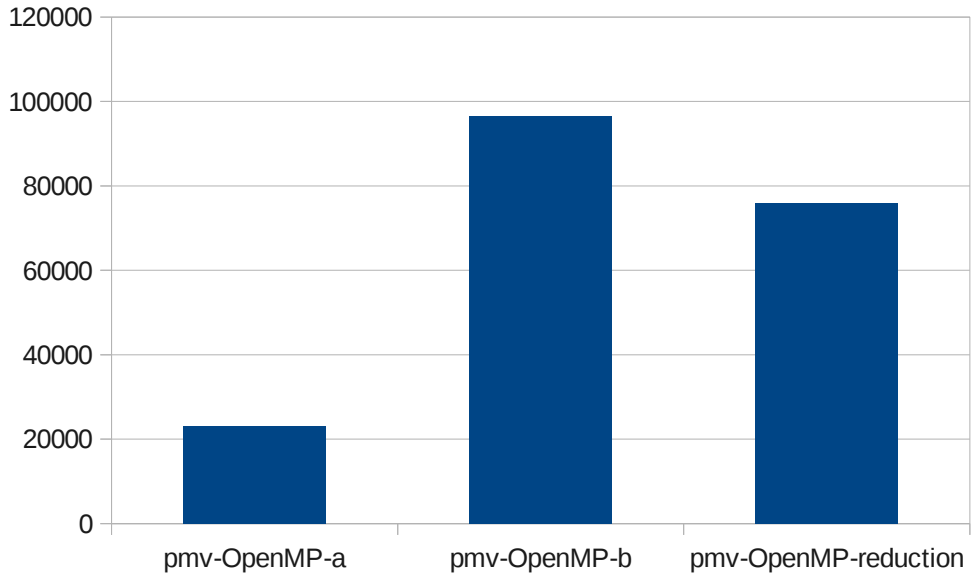
N = 100



N = 1000


2500000
2000000
1500000
1000000
500000
0
pmv-OpenMP-a
pmv-OpenMP-b
pmv-OpenMP-reduction

N = 10000


120000
100000
80000
60000
40000
20000
0
pmv-OpenMP-a
pmv-OpenMP-b
pmv-OpenMP-reduction

La gráficas tienen una escala 1:1000000