

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): David Carrasco Chicharro

Grupo de prácticas: B1

Fecha de entrega: 21/04/2018

Fecha evaluación en clase: 23/04/2018

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:

Con la directiva `default()` usando como argumento `none` las variables que no están incluidas dentro de la cláusula `shared` no tienen ámbito dentro de la región paralela, que debe especificarse obligatoriamente. Para solucionar el problema debe escribirse: `shared(a, n)`

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

```
1  #include <stdio.h>
2  #ifdef _OPENMP
3      #include <omp.h>
4  #endif
5
6  int main()
7  {
8      int i, n = 7;
9      int a[n];
10
11     for (i=0; i<n; i++)
12         a[i] = i+1;
13
14     #pragma omp parallel for shared(a, n), default(none)
15     for (i=0; i<n; i++)    a[i] += i;
16
17     printf("Después de parallel for:\n");
18
19     for (i=0; i<n; i++)
20         printf("a[%d] = %d\n", i, a[i]);
21 }
```

CAPTURAS DE PANTALLA:

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_Carrasco
ChicharroDavid_2017_18:2018-04-15 domingo
$ gcc -O2 -fopenmp -o shared-clauseModificado 1-shared-clauseModificado.c
1-shared-clauseModificado.c: In function 'main':
1-shared-clauseModificado.c:14:12: error: 'n' not specified in enclosing parallel
      #pragma omp parallel for shared(a), default(none)
      ^
1-shared-clauseModificado.c:14:12: error: enclosing parallel
```

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_Carrasco
ChicharroDavid_2017_18:2018-04-15 domingo
$ ./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
```

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:

Si se inicializa la variable `suma` fuera de la región `parallel`, al llegar a dicha sección de código, por ser privada, toma un valor indefinido.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`

```
1  #include <stdio.h>
2  #ifdef _OPENMP
3      #include <omp.h>
4  #else
5      #define omp_get_thread_num() 0
6  #endif
7
8  int main()
9  {
10     int i, n = 7;
11     int a[n], suma;
12
13     for (i=0; i<n; i++)
14         a[i] = i;
15
16     //suma=20;
17     #pragma omp parallel private(suma)
18     {
19         suma=20;
20         #pragma omp for
21         for (i=0; i<n; i++)
22         {
23             suma = suma + a[i];
24             printf(
25                 "thread %d suma a[%d] / ", omp_get_thread_num(), i);
26         }
27         printf(
28             "\n* thread %d suma= %d", omp_get_thread_num(), suma);
29     }
30
31     printf("\n");
32
33     return 0;
34 }
```

CAPTURAS DE PANTALLA:

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_Carrasco
ChicharroDavid_2017_18:2018-04-15 domingo
$ ./private-clauseModificado thread 1 suma a[2] / thread 1 suma a[3] / thread 3 s
uma a[6] / thread 0 suma a[0] / thread 0 suma a[1] / thread 2 suma a[4] / thread
2 suma a[5] /
* thread 1 suma= 4196565
* thread 3 suma= 4196566
* thread 0 suma= 5
* thread 2 suma= 4196569
```

Inicialización de suma fuera de la región parallel

```
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_Carrasco
ChicharroDavid_2017_18:2018-04-15 domingo
$ ./private-clauseModificado
thread 2 suma a[4] / thread 2 suma a[5] / thread 3 suma a[6] / thread 0 suma a[0]
/ thread 0 suma a[1] / thread 1 suma a[2] / thread 1 suma a[3] /
* thread 2 suma= 9
* thread 0 suma= 1
* thread 3 suma= 6
* thread 1 suma= 5
```

Inicialización de suma dentro de la región parallel

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

RESPUESTA:

Todas las componentes del vector almacenan el mismo valor, ya que la variable `suma` se convierte en una variable compartida por todas la hebras.

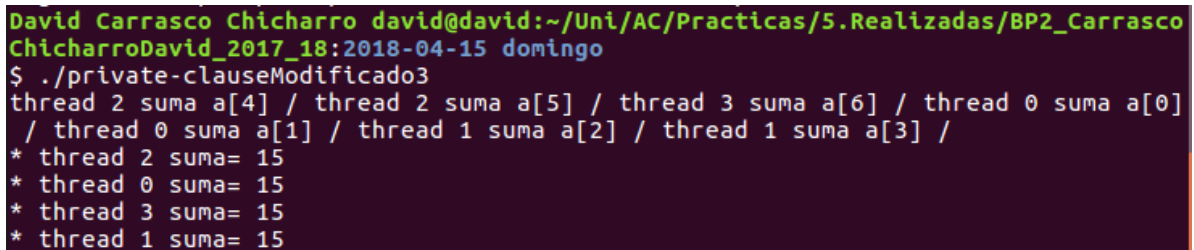
CAPTURA CÓDIGO FUENTE: `private-clauseModificado3.c`

```
1  #include <stdio.h>
2  #ifdef _OPENMP
3      #include <omp.h>
4  #else
5      #define omp_get_thread_num() 0
6  #endif
7
8  int main()
9  {
10     int i, n = 7;
11     int a[n], suma;
12
13     for (i=0; i<n; i++)
14         a[i] = i;
15
16     //suma=20;
17     #pragma omp parallel
18     {
19         suma=20;
20         #pragma omp for
21         for (i=0; i<n; i++)
22         {
23             suma = suma + a[i];
24             printf(
25                 "thread %d suma a[%d] / ", omp_get_thread_num(), i);
```

```

26     }
27     printf(
28         "\n* thread %d suma= %d", omp_get_thread_num(), suma);
29 }
30
31     printf("\n");
32
33     return 0;
34 }

```

CAPTURAS DE PANTALLA:


```

David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_Carrasco
ChicharroDavid_2017_18:2018-04-15 domingo
$ ./private-clauseModificado3
thread 2 suma a[4] / thread 2 suma= 15 / thread 3 suma a[6] / thread 0 suma a[0]
/ thread 0 suma a[1] / thread 1 suma a[2] / thread 1 suma a[3] /
* thread 2 suma= 15
* thread 0 suma= 15
* thread 3 suma= 15
* thread 1 suma= 15

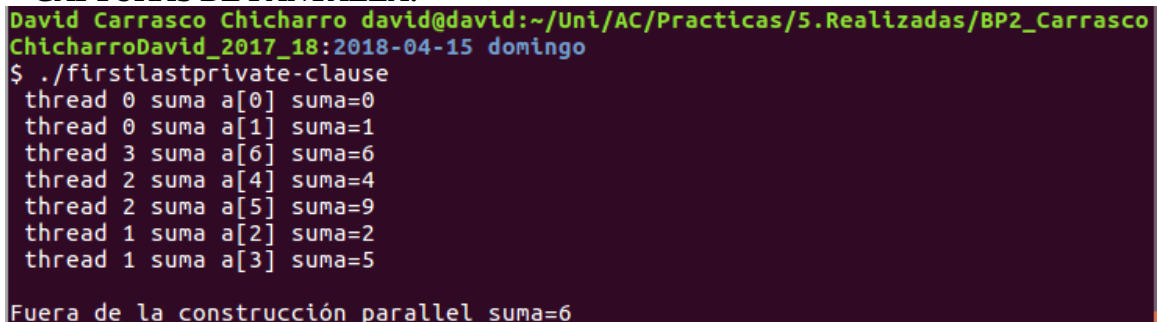
```

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:

Con la cláusula firstprivate(suma) se indica que la variable suma sea privada para todas las hebras y que además esté inicializada con el valor asignado en la declaración.

Con la cláusula lastprivate(suma) se indica que la variable suma, cuando se acabe la región paralela, debe tener como contenido el valor producido en dicha variable en la última iteración del bucle. Por esto, siempre se imprimirá un 6 fuera de la región parallel, independientemente de la hebra que ejecute la última iteración del bucle (siempre que no se modifique el valor de n en el código, que es el número de iteraciones).

CAPTURAS DE PANTALLA:


```

David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_Carrasco
ChicharroDavid_2017_18:2018-04-15 domingo
$ ./firstlastprivate-clause
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 3 suma a[6] suma=6
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
Fuera de la construcción parallel suma=6

```

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:

Al eliminar la cláusula, la hebra single, que tiene el valor de a, no copia ese valor en el resto de hebras debido a que es privada, con lo que los demás threads almacenarán un contenido indeterminado de a.

CAPTURA CÓDIGO FUENTE: copyprivate-clauseModificado.c

```

1  #include <stdio.h>
2  #include <omp.h>
3
4  int main() {
5      int n = 9, i, b[n];
6
7      for (i=0; i<n; i++)    b[i] = -1;
8
9      #pragma omp parallel
10 {   int a;
11     #pragma omp single
12     {
13         printf("\nIntroduce valor de inicialización a: ");
14         scanf("%d", &a );
15         printf("\nSingle ejecutada por el thread %d\n",
16             omp_get_thread_num());
17     }
18     #pragma omp for
19     for (i=0; i<n; i++)    b[i] = a;
20 }
21
22 printf("Después de la región parallel:\n");
23 for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
24 printf("\n");
25
26 return 0;
27 }

```

CAPTURAS DE PANTALLA:

```

David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_
CarrascoChicharroDavid_2017_18:2018-04-15 domingo
$ ./copyprivate-clauseModificado

Introduce valor de inicialización a: 10

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

Si se ejecuta el programa pasándole como argumento ./reduction-clause 10, el resultado de suma al final de la ejecución es de suma=45. Tras sustituir la inicialización por suma=10, el resultado será suma=55.

Esto se debe a que la cláusula reduction(+:suma), además de acumular el resultado de las sumas, añade el valor de la inicialización; en este caso al haberse inicializado suma=10, el valor final será de 10 unidades más que en la ejecución en la que se inicializaba a suma=0.

CAPTURA CÓDIGO FUENTE: reduction-clauseModificado.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #ifdef _OPENMP
4      #include <omp.h>
5  #else
6      #define omp_get_thread_num() 0
7  #endif
8
9  int main(int argc, char **argv) {
10     int i, n=20, a[n], suma=10;
11
12     if(argc < 2) {
13         fprintf(stderr, "Falta iteraciones\n");
14         exit(-1);
15     }
16     n = atoi(argv[1]); if (n>20) {n=20; printf("n=%d",n);}
17
18     for (i=0; i<n; i++)    a[i] = i;
19
20     #pragma omp parallel for reduction(+:suma)
21     for (i=0; i<n; i++)    suma += a[i];
22
23     printf("Tras 'parallel' suma=%d\n", suma);
24 }

```

CAPTURAS DE PANTALLA:


```

David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_
CarrascoChicharroDavid_2017_18:2018-04-15 domingo
$ ./reduction-clause 10
Tras 'parallel' suma=45
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_
CarrascoChicharroDavid_2017_18:2018-04-15 domingo
$ ./reduction-clauseModificado 10
Tras 'parallel' suma=55

```

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:

Si se elimina reduction(+:suma) una posible alternativa es usar la cláusula firstprivate() con la variable suma y después la directiva atomic para almacenar en suma_final el resultado de todas las sumas parciales realizadas por cada hebra asegurando la exclusión mutua.

CAPTURA CÓDIGO FUENTE: reduction-clauseModificado7.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #ifdef _OPENMP
4      #include <omp.h>
5  #else
6      #define omp_get_thread_num() 0
7  #endif

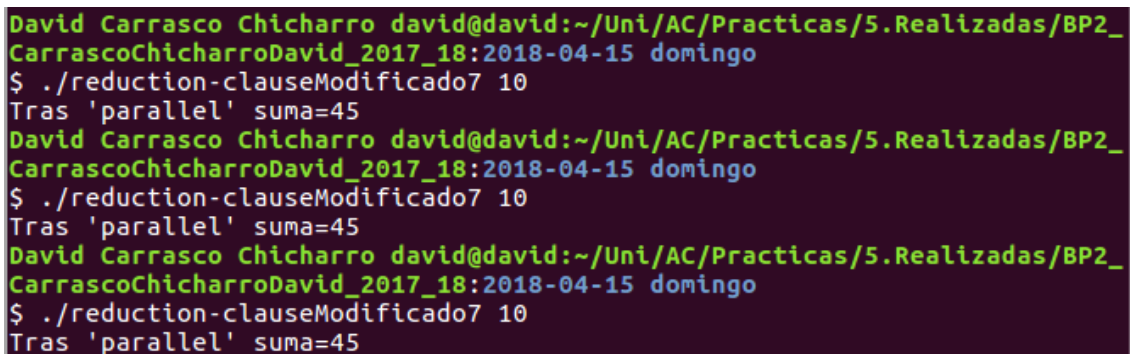
```

```

8
9 ▾ int main(int argc, char **argv) {
10     int i, n=20, a[n], suma=0, suma_final=0;
11
12 ▾     if(argc < 2) {
13         fprintf(stderr, "Falta iteraciones\n");
14         exit(-1);
15     }
16     n = atoi(argv[1]);
17     if (n>20) {n=20; printf("n=%d", n);}
18
19     for (i=0; i<n; i++)
20         a[i] = i;
21
22     #pragma omp parallel firstprivate(suma)
23 ▾ {
24     #pragma omp for
25         for (i=0; i<n; i++) suma += a[i];
26
27     #pragma omp atomic
28         suma_final += suma;
29     }
30
31     printf("Tras 'parallel' suma=%d\n", suma_final);
32 }

```

CAPTURAS DE PANTALLA:



```

David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_
CarrascoChicharroDavid_2017_18:2018-04-15 domingo
$ ./reduction-clauseModificado7 10
Tras 'parallel' suma=45
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_
CarrascoChicharroDavid_2017_18:2018-04-15 domingo
$ ./reduction-clauseModificado7 10
Tras 'parallel' suma=45
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_
CarrascoChicharroDavid_2017_18:2018-04-15 domingo
$ ./reduction-clauseModificado7 10
Tras 'parallel' suma=45

```

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):

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

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).

CAPTURA CÓDIGO FUENTE: pmv-secuencial.c

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <time.h>
4
5  //#define VERSION_GLOBAL
6  #define VERSION_DYNAMIC
7
8  #ifdef VERSION_GLOBAL
9  #define MAX 33554432
10 int M[MAX][MAX], v1[MAX], v2[MAX];
11 #endif
12
13 int main(int argc, char **argv) {
14
15     struct timespec cgt1,cgt2;
16     double ncgt; //para tiempo de ejecución
17
18     if(argc < 2){
19         fprintf(stderr,"Falta el tamaño\n");
20         exit(-1);
21     }
22
23     unsigned int N = atoi(argv[1]);
24     int fil, col, suma;
25
26     #ifdef VERSION_GLOBAL
27     if (N>MAX) N=MAX;
28     #endif
29
30     #ifdef VERSION_DYNAMIC
31     int **M, *v1, *v2;
32     M = (int**) malloc(N*sizeof(int*));
33     for(int i=0 ; i<N ; i++)
34         M[i] = (int*) malloc(N*sizeof(int));
35
36     v1 = (int*) malloc(N*sizeof(int));
37     v2 = (int*) malloc(N*sizeof(int));
38
39     if ( (M==NULL) || (v1==NULL) || (v2==NULL) ){
40         printf("Error en la reserva de espacio para los vectores\n");
41         exit(-2);
42     }
43     #endif
44
45     // Inicialización de la matriz y los vectores
46     for(fil=0 ; fil<N ; fil++){
47         for(int col=0 ; col<N ; col++)
48             M[fil][col]=fil+col;
49
50         v1[fil]=fil;
51         v2[fil]=0;
52     }

```



```

53
54     clock_gettime(CLOCK_REALTIME,&cgt1);
55     // Cálculo de v2
56     for (fil=0 ; fil<N; fil++){
57         suma=0;
58         for(col=0 ; col<N ; col++)
59             suma += M[fil][col] * v1[col];
60
61         v2[fil] = suma;
62     }
63
64     clock_gettime(CLOCK_REALTIME,&cgt2);
65     ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+
66         (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
67
68     printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
69     if(N<20)
70         for(fil=0 ; fil<N ; fil++)
71             printf("V2[%d]=%d  ", fil, v2[fil]);
72     else{
73         printf("V2[0]=%d  ", v2[0]);
74         printf("V2[%d]=%d  ", N-1, v2[N-1]);
75     }
76     printf("\n");
77
78     #ifdef VERSION_DYNAMIC
79     for(int i=0 ; i<N ; i++)
80         free(M[i]);
81     free(M);   free(v1);   free(v2);
82     #endif
83
84     return 0;
85 }

```

CAPTURAS DE PANTALLA:


```

ChicharroDavid_2017_18:2018-04-15 domingo
$ gcc -O2 8-pmv-secuencial.c -o pmv-secuencial -lrt
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_Carrasco
ChicharroDavid_2017_18:2018-04-15 domingo
$ ./pmv-secuencial 8
Tiempo(seg.):0.000000647          / Tamaño Vectores:8
V2[0]=168 V2[1]=336 V2[2]=504 V2[3]=672 V2[4]=840 V2[5]=1008 V2[6]=1176 V2
[7]=1344
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_Carrasco
ChicharroDavid_2017_18:2018-04-15 domingo
$ ./pmv-secuencial 11
Tiempo(seg.):0.000000757          / Tamaño Vectores:11
V2[0]=440 V2[1]=880 V2[2]=1320 V2[3]=1760 V2[4]=2200 V2[5]=2640 V2[6]=3080
V2[7]=3520 V2[8]=3960 V2[9]=4400 V2[10]=4840

```

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):
 - 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).

CAPTURA CÓDIGO FUENTE : pmv-OpenMP-a.c

```

20 ▾ int main(int argc, char **argv) {
21
22     double t_ini, t_fin, t_elapsed; //para tiempo de ejecución
23
24 ▾     if(argc < 2){
25         fprintf(stderr,"Falta el tamaño\n");
26         exit(-1);
27     }
28
29     unsigned int N = atoi(argv[1]);
30     int fil, col, suma;
31
32     #ifdef VERSION_GLOBAL
33     if (N>MAX) N=MAX;
34     #endif
35
36     #ifdef VERSION_DYNAMIC
37     int **M, *v1, *v2;
38     M = (int**) malloc(N*sizeof(int*));
39     for(int i=0 ; i<N ; i++)
40         M[i] = (int*) malloc(N*sizeof(int));
41
42     v1 = (int*) malloc(N*sizeof(int));
43     v2 = (int*) malloc(N*sizeof(int));
44
45 ▾     if ( (M==NULL) || (v1==NULL) || (v2==NULL) ){
46         printf("Error en la reserva de espacio para los vectores\n");
47         exit(-2);
48     }
49     #endif
50

```

```

51     #pragma omp parallel
52     {
53         // Inicialización de la matriz y los vectores
54         #pragma omp for private(fil,col)
55         for(fil=0 ; fil<N ; fil++){
56             for(int col=0 ; col<N ; col++)
57                 M[fil][col]=fil+col;
58
59             v1[fil]=fil;
60             v2[fil]=0;
61         }
62
63         #pragma omp single
64         t_ini = omp_get_wtime();
65
66         // Cálculo de v2
67         #pragma omp for private(fil,col)
68         for (fil=0 ; fil<N; fil++){
69             suma=0;
70             for(col=0 ; col<N ; col++)
71                 suma += M[fil][col] * v1[col];
72
73             v2[fil] = suma;
74         }
75
76         #pragma omp single
77         t_fin = omp_get_wtime();
78     }
79
80     t_elapsed = t_fin - t_ini;
81
82     printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",t_elapsed,N);
83     if(N<20)
84         for(fil=0 ; fil<N ; fil++)
85             printf("V2[%d]=%d ", fil, v2[fil]);
86     else{
87         printf("V2[0]=%d ", v2[0]);
88         printf("V2[%d]=%d ", N-1, v2[N-1]);
89     }
90     printf("\n");
91
92     #ifdef VERSION_DYNAMIC
93     for(int i=0 ; i<N ; i++)
94         free(M[i]);
95     free(M); free(v1); free(v2);
96     #endif
97
98     return 0;
99 }

```

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

```

51 // Inicialización de la matriz y los vectores
52 for(fil=0 ; fil<N ; fil++){
53     #pragma omp parallel for
54     for(int col=0 ; col<N ; col++)
55         M[fil][col]=fil+col;
56
57     v1[fil]=fil;
58     v2[fil]=0;
59 }
60
61 t_ini = omp_get_wtime();
62
63 // Cálculo de v2
64 for (fil=0 ; fil<N; fil++){
65     suma=0;
66     #pragma omp parallel firstprivate(suma)
67     {
68         #pragma omp for private(col)
69         for(col=0 ; col<N ; col++)
70             suma += M[fil][col]*v1[col];
71
72         #pragma omp critical
73         v2[fil] += suma;
74     }
75 }
76
77 t_fin = omp_get_wtime();
78 t_elapsed = t_fin - t_ini;
79
80
81 printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",t_elapsed,N);
82 if(N<20)
83     for(fil=0 ; fil<N ; fil++)
84         printf("V2[%d]=%d ", fil, v2[fil]);
85 else{
86     printf("V2[0]=%d ", v2[0]);
87     printf("V2[%d]=%d ", N-1, v2[N-1]);
88 }
89 printf("\n");
90
91 #ifdef VERSION_DYNAMIC
92 for(int i=0 ; i<N ; i++)
93     free(M[i]);
94 free(M);
95 free(v1);
96 free(v2);
97 #endif
98
99 return 0;
100 }

```

RESPUESTA:

No he tenido ningún error de compilación en ninguno de los dos apartados, pero para el apartado 9b he tenido que compilar varias veces por cambios de código, puesto que las ejecuciones no daban el resultado esperado al paralelizar mal las hebras o incluyendo cláusulas erróneas.

CAPTURAS DE PANTALLA:

```

David_2017_18:2018-04-16 lunes
$ gcc -O2 -fopenmp 9-pmv-OpenMP-a.c -o pmv-OpenMP-a -lrt
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_CarrascoChicharro
David_2017_18:2018-04-16 lunes
$ ./pmv-OpenMP-a 8
Tiempo(seg.):0.000002561 / Tamaño Vectores:8
V2[0]=140 V2[1]=168 V2[2]=196 V2[3]=224 V2[4]=252 V2[5]=280 V2[6]=308 V2[7]=336
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_CarrascoChicharro
David_2017_18:2018-04-16 lunes
$ ./pmv-OpenMP-a 11
Tiempo(seg.):0.000001688 / Tamaño Vectores:11
V2[0]=385 V2[1]=440 V2[2]=495 V2[3]=550 V2[4]=605 V2[5]=660 V2[6]=715 V2[7]=770 V2
[8]=825 V2[9]=880 V2[10]=935

```

Ejecuciones 9a

```

David_2017_18:2018-04-16 lunes
$ gcc -O2 -fopenmp 9-pmv-OpenMP-b.c -o pmv-OpenMP-b -lrt
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_CarrascoChicharro
David_2017_18:2018-04-16 lunes
$ ./pmv-OpenMP-b 8
Tiempo(seg.):0.000051210 / Tamaño Vectores:8
V2[0]=140 V2[1]=168 V2[2]=196 V2[3]=224 V2[4]=252 V2[5]=280 V2[6]=308 V2[7]=336
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_CarrascoChicharro
David_2017_18:2018-04-16 lunes
$ ./pmv-OpenMP-b 11
Tiempo(seg.):0.000031883 / Tamaño Vectores:11
V2[0]=385 V2[1]=440 V2[2]=495 V2[3]=550 V2[4]=605 V2[5]=660 V2[6]=715 V2[7]=770 V2
[8]=825 V2[9]=880 V2[10]=935

```

Ejecuciones 9b

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`

```

61     t_ini = omp_get_wtime();
62
63     // Cálculo de v2
64     for (fil=0 ; fil<N; fil++){
65         suma=0;
66         #pragma omp parallel
67         {
68             #pragma omp for reduction(+:suma)
69             for(col=0 ; col<N ; col++)
70                 suma += M[fil][col]*v1[col];
71
72             v2[fil] = suma;
73         }
74     }
75
76     t_fin = omp_get_wtime();
77     t_elapsed = t_fin - t_ini;

```

RESPUESTA:

He tenido un error de compilación la primera vez:

```
10-pmv-OpenmMP-reduction.c: In function 'main':
10-pmv-OpenmMP-reduction.c:68:14: error: reduction variable
'suma' is private in outer context
    #pragma omp for reduction(+:suma)
```

Se debe a que tenía escrito en el código “#pragma omp parallel firstprivate (suma)” en la paralelización de las columnas, y la variable suma no puede ser privada para la cláusula reduction.

CAPTURAS DE PANTALLA:

```
David_2017_18:2018-04-16 lunes
$ gcc -O2 -fopenmp 10-pmv-OpenmMP-reduction.c -o pmv-OpenMP-reduction -lrt
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_CarrascoChicharro
David_2017_18:2018-04-16 lunes
$ ./pmv-OpenMP-reduction 8
Tiempo(seg.):0.000110624 / Tamaño Vectores:8
V2[0]=140 V2[1]=168 V2[2]=196 V2[3]=224 V2[4]=252 V2[5]=280 V2[6]=308 V2[7]=336
David Carrasco Chicharro david@david:~/Uni/AC/Practicas/5.Realizadas/BP2_CarrascoChicharro
David_2017_18:2018-04-16 lunes
$ ./pmv-OpenMP-reduction 11
Tiempo(seg.):0.000070025 / Tamaño Vectores:11
V2[0]=385 V2[1]=440 V2[2]=495 V2[3]=550 V2[4]=605 V2[5]=660 V2[6]=715 V2[7]=770 V2
[8]=825 V2[9]=880 V2[10]=935
```

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.

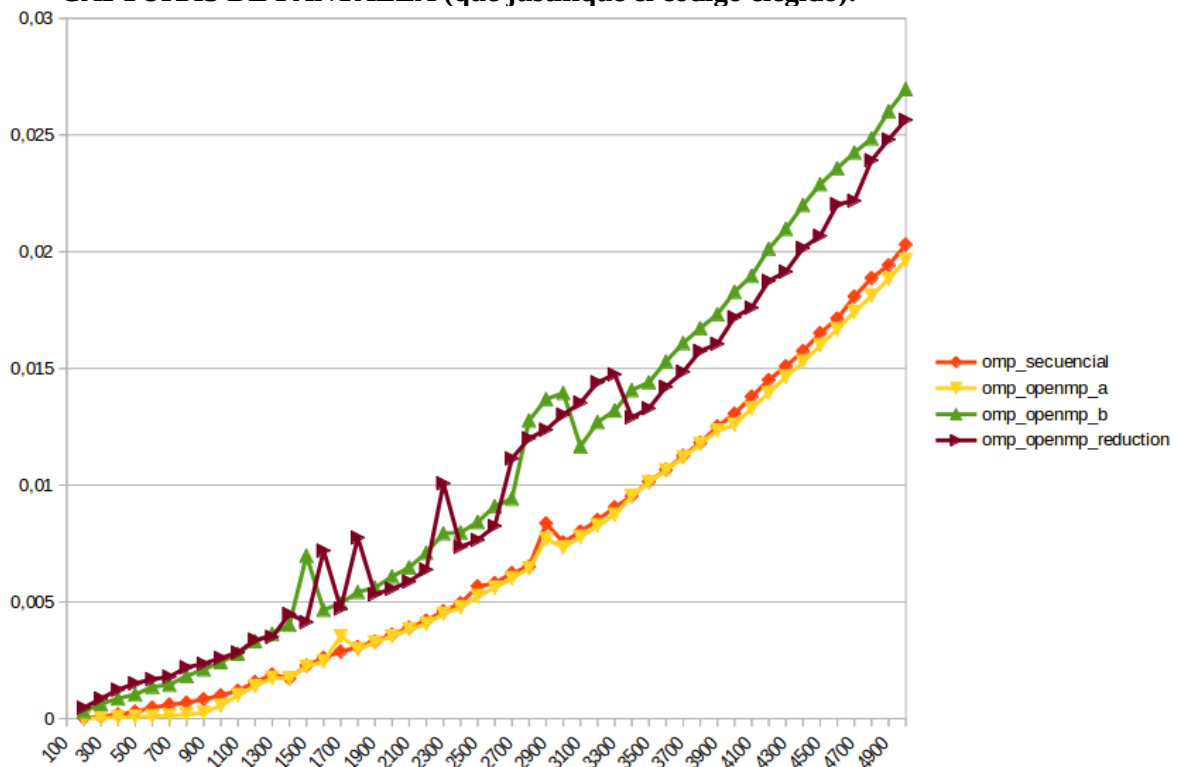
CAPTURAS DE PANTALLA (que justifique el código elegido):

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):

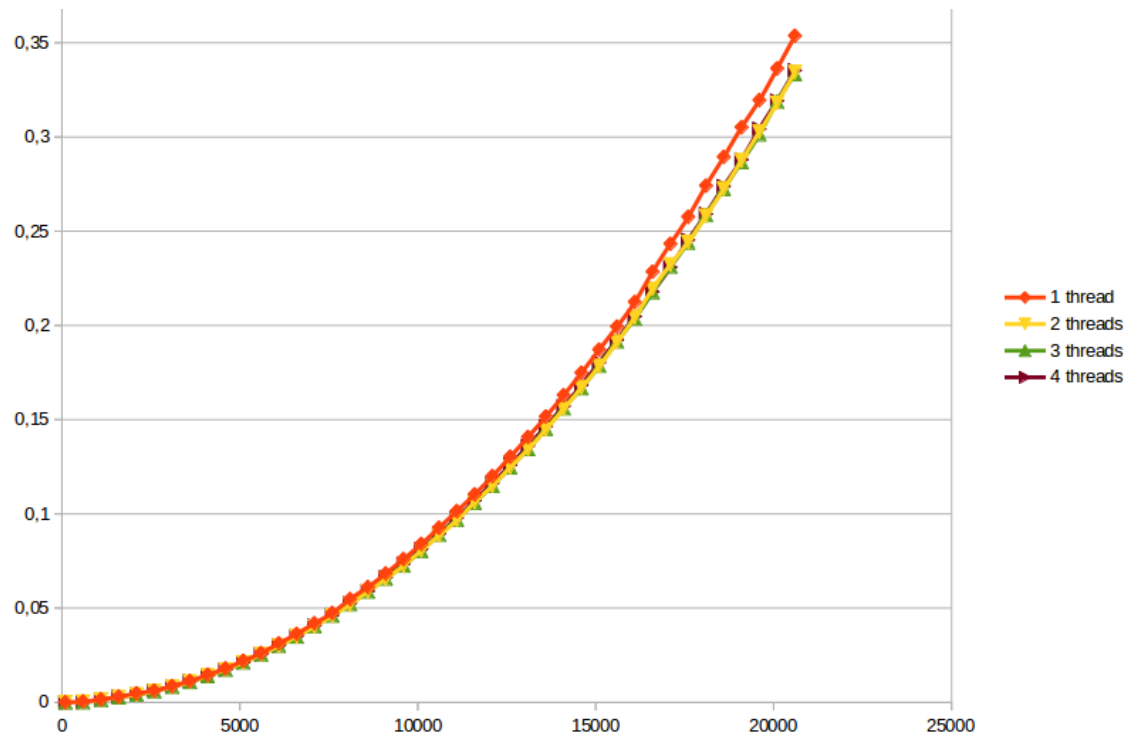


Gráfico en mi PC

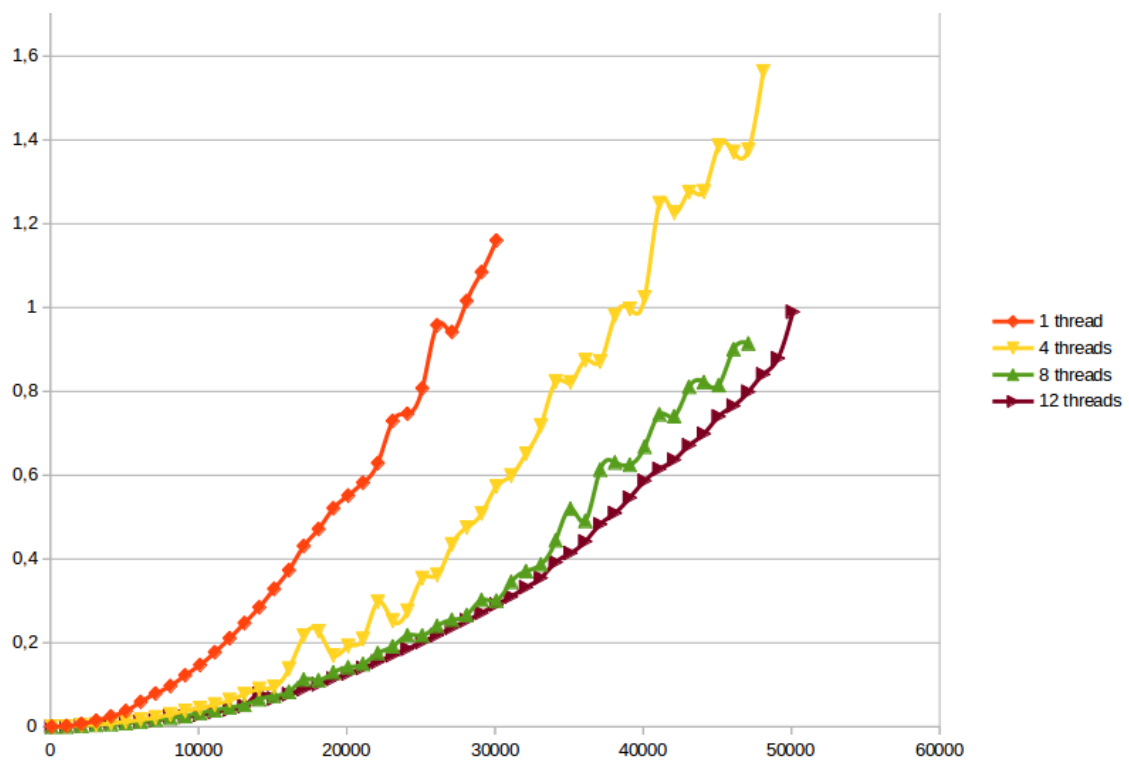


Gráfico en ATCGRID

Num. Iters.	1 thread	2 threads	3 threads	4 threads	Ganancia
100	1,9649E-05	1,4319E-05	3,007E-06	4,898E-06	4,011637403
600	0,000429467	0,000307132	7,3809E-05	7,5606E-05	5,6803296035
1100	0,001534221	0,001413574	0,001386513	0,001451651	1,0568800628
1600	0,002981221	0,002698592	0,002712875	0,002971673	1,0032130049
2100	0,004811119	0,004128211	0,004224674	0,004172002	1,1531919208
2600	0,006228677	0,005939357	0,005929499	0,006146038	1,013445898
3100	0,008640113	0,00825159	0,008330639	0,008515512	1,0146322382
3600	0,011317696	0,010931653	0,010904496	0,011119545	1,0178200637
4100	0,014612164	0,014206708	0,014032454	0,014178352	1,0305967859
4600	0,018001429	0,017452725	0,017391562	0,017382523	1,035605073
5100	0,021984803	0,021192739	0,021291751	0,021208451	1,0366057851
5600	0,026124538	0,025497942	0,025498097	0,025483507	1,0251547403
6100	0,031302169	0,029924827	0,030094398	0,030039701	1,04202665
6600	0,036315456	0,035025623	0,034971895	0,03506814	1,0355683535
7100	0,041957727	0,040121798	0,040494425	0,040573536	1,0341156117
7600	0,047401951	0,0458907	0,046070664	0,046252792	1,0248451812
8100	0,054781081	0,051864835	0,052153393	0,052470884	1,04402817
8600	0,061157848	0,058847819	0,058856821	0,05905509	1,0356067191
9100	0,068389162	0,065416837	0,065700483	0,06618686	1,0332740064
9600	0,076008864	0,072531679	0,072681429	0,073459201	1,0347085588
10100	0,084059782	0,080566581	0,080413434	0,081172088	1,0355749627
10600	0,092654997	0,088503595	0,089039007	0,089417272	1,0362091677
11100	0,101487993	0,096680568	0,096993435	0,097861433	1,0370581126
11600	0,110346242	0,106221242	0,105961144	0,107062557	1,0306707134
12100	0,120057991	0,114895392	0,115022957	0,116280592	1,0324852061
12600	0,130323999	0,124765851	0,124716018	0,126008637	1,0342465572
13100	0,140790399	0,134236621	0,134518154	0,136022724	1,0350505773
13600	0,151626717	0,144714803	0,145185181	0,146464059	1,0352486339
14100	0,162989408	0,155674376	0,156142929	0,157113385	1,037399888
14600	0,174854122	0,1670261	0,166767788	0,168323629	1,0387972446
15100	0,187155267	0,178377704	0,178716029	0,180313063	1,0379462469
15600	0,199345351	0,191085791	0,191530103	0,192381797	1,0361965327
16100	0,212434159	0,204367456	0,203645754	0,204735279	1,0376040712
16600	0,228473369	0,219377301	0,217671799	0,217830332	1,0488592975
17100	0,243288385	0,232235146	0,231320718	0,230912463	1,0535957299
17600	0,25758788	0,244264523	0,244013565	0,245317637	1,0500177776
18100	0,27412842	0,258082003	0,258666037	0,259138343	1,0578458472
18600	0,289383954	0,272470449	0,272451665	0,273828568	1,0568070239
19100	0,305159895	0,287552237	0,286759394	0,287984516	1,0596399391
19600	0,319503397	0,302711847	0,301434215	0,3042247	1,0502217506
20100	0,336336653	0,318047989	0,318787177	0,319200414	1,0536848896
20600	0,353681963	0,334367087	0,333358686	0,33526646	1,0549279609

Tabla en mi PC

Num. Iters.	1 thread	4 threads	8 threads	12 threads	Ganancia
100	1,9827E-05	8,802E-06	1,22E-05	1,1195E-05	1,7710585083
1100	0,001631998	0,000544793	0,000298398	0,000335108	4,8700657698
2100	0,00650961	0,002942592	0,002252813	0,002780065	2,3415315829
3100	0,014301687	0,004961214	0,004510639	0,004545248	3,1465141176
4100	0,024644529	0,007870479	0,00601049	0,007583793	3,2496310224
5100	0,037462707	0,011996274	0,008884105	0,010060482	3,723748723
6100	0,059024159	0,016336082	0,012430357	0,011166476	5,2858358358
7100	0,07903693	0,021646168	0,01680379	0,014856072	5,3201768274
8100	0,096973809	0,028790815	0,020595698	0,025475603	3,8065363556
9100	0,122577221	0,037305551	0,025507307	0,023785713	5,1533969572
10100	0,147026169	0,043824307	0,032901307	0,028778336	5,1089183544
11100	0,177458841	0,052270488	0,03953685	0,035067653	5,0604709987
12100	0,210826164	0,063027689	0,046165114	0,040990606	5,1432799993
13100	0,247144891	0,077037045	0,052356656	0,049474272	4,9954224895
14100	0,284781855	0,089545576	0,065296524	0,078646403	3,6210410666
15100	0,328544706	0,094522516	0,073831677	0,067141216	4,8933386312
16100	0,373252993	0,137821275	0,083358765	0,07737072	4,8242150648
17100	0,430888427	0,216992904	0,112522005	0,08966078	4,8057626423
18100	0,471477893	0,227467554	0,110818009	0,101181992	4,6597016295
19100	0,52107242	0,169701971	0,129432079	0,114428427	4,5536973081
20100	0,551361899	0,192161942	0,141707971	0,126734266	4,3505353083
21100	0,582103726	0,208826415	0,149478916	0,140207686	4,151724792
22100	0,628918464	0,297372648	0,175485732	0,154647755	4,0667804327
23100	0,729392014	0,252625591	0,191520925	0,170329676	4,2822368429
24100	0,746630796	0,275900014	0,218084731	0,185831186	4,0177906199
25100	0,8073125	0,35395266	0,217737814	0,202036512	3,9958742705
26100	0,95783251	0,361601016	0,240341563	0,218251172	4,3886706368
27100	0,941741792	0,434714812	0,254730046	0,236040809	3,989741418
28100	1,015749265	0,475015294	0,266366469	0,253468085	4,0074049757
29100	1,084651018	0,508584789	0,302429667	0,271399281	3,9965139701
30100	1,160135766	0,573201403	0,300767309	0,29133672	3,9821130889
31100		0,598597989	0,345479598	0,310288446	
32100		0,65134946	0,37115734	0,332615562	
33100		0,7179938	0,386849063	0,354952544	
34100		0,822999313	0,444662183	0,392163781	
35100		0,821281983	0,519976434	0,414235444	
36100		0,874073026	0,490882589	0,442047669	
37100		0,871024966	0,61323454	0,482997043	
38100		0,981049996	0,63047764	0,509734049	
39100		0,996597209	0,624982851	0,546330379	
40100		1,024309632	0,667994614	0,586524856	
41100		1,24873112	0,745313005	0,615029247	
42100		1,226304837	0,740893042	0,636912055	
43100		1,275243681	0,810763463	0,671255452	
44100		1,276559037	0,822276832	0,698938706	
45100		1,386057469	0,814974035	0,740505126	
46100		1,370718532	0,90047513	0,76565414	
47100		1,376428117	0,914161814	0,798883667	
48100		1,563309176		0,840522099	
49100				0,879078586	
50100				0,989880947	

Tabla en ATCGRID

COMENTARIOS SOBRE LOS RESULTADOS:

La mejor elección es la ejecución del programa 9a: pmv-OpenMP-a

En mi PC se ha ejecutado el programa para 1, 2, 3 y 4 hebras, donde la diferencia en el tiempo de cálculo del vector v2 ha sido mínima. Sin embargo en ATCGRID, donde se ha ejecutado el programa para 1, 4, 8 y 12 hebras, la diferencia en el tiempo de cálculo ha sido más notorio, mejorándose el resultado cuanto mayor es el número de hebras que ejecutan el programa