

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): Javier Galera Garrido

Grupo de prácticas: B3

Fecha de entrega:

Fecha evaluación en clase:

[RECORDATORIO, quitar todo este texto en rojo del cuaderno definitivo–

1. COMENTARIOS

1) Este cuaderno de prácticas se utilizará para asignarle una puntuación durante la evaluación continua de prácticas y también lo utilizará como material de estudio y repaso para preparar el examen de prácticas escrito. Luego redáctelo con cuidado, y sea ordenado y claro.

2) No use máquinas virtuales. Se piden obtener resultados en atcgrid y en su PC (PC del aula o PC personal).

3) Debe modificar el prompt en los computadores que utilice en prácticas para que aparezca su nombre y apellidos, su usuario (\u), el computador (\h), el directorio de trabajo del bloque práctico (\w), la fecha (\D) completa (%F) y el día (%A) . Para modificar el prompt utilice lo siguiente (si es necesario, use export delante):

PS1=" [NombreApellidos \u@\h:\w] \D{%-F %A}\n\$"

donde NombreApellidos es su nombre seguido de sus apellidos, por ejemplo: Juan Ortúñoz Vilariño

2. NORMAS SOBRE EL USO DE LA PLANTILLA

1) Usar interlineado SENCILLO.

2) Respetar los tipos de letra y tamaños indicados:

- Calibri-11 o Liberation Serif-11 para el texto

- Courier New-10 o Liberation Mono-10 para nombres de fichero, comandos, variables de entorno, etc., cuando se usan en el texto.

3) Insertar las capturas de pantalla donde se pidan y donde se considere oportuno. En particular, los listados de código se deben insertar como capturas de pantalla. En todas las capturas de pantalla, incluidas las de los listados de código, debe aparecer el directorio y usuario. El tamaño de letra en las capturas debe ser similar al tamaño que se está usando en el texto.

Recuerde que debe adjuntar al zip de entrega, el pdf de este fichero, todos los ficheros con código fuente implementados/utilizados y el resto de ficheros que haya implementado/utilizado (scripts, hojas de cálculo, etc.)]

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 compilador de Ubuntu 17, versión 7.2.0 de gcc, el resultado de la primera compilación no es erronea aunque debería de serlo

```
~/Escritorio/segundo/ac/practicas/practica3/ejercicio1 ① 12:47:26
$ gcc shared-clause.c -o shared-clause
shared-clause.c:1:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
^~~~~
shared-clause.c:1:1: note: previous declaration of 'main' was here
main()
^~~~~
~/Escritorio/segundo/ac/practicas/practica3/ejercicio1 ② 12:54:05
$ ./shared-clause
jota@jota-Erazer-P6679-MD60359:jota@jota-Erazer-P6679-MD60359: ~/Escritorio/segundo/ac/practicas/practica3/ejercicio1:-/Escritorio/segundo/ac/practicas/practica3/ejercicio1 zsh - Drop- X
+ Consola
ejercicio1 JotaGalera@jota-Erazer-P6679-MD60359:jota@jota-Erazer-P6679-MD60359: ~/Escritorio/segundo/ac/practicas/practica3/ejercicio1 zsh - Drop- X
```

```
1 #include <stdio.h>
2 #ifdef _OPENMP
3 #include <omp.h>
4 #endif
5
6 main()
{
7     int i,n = 7;
8     int a[n];
9
10    for(i = 0; i<n ; i++)
11        a[i] = i+1;
12
13    #pragma omp parallel for shared(a) default(none)
14    for (i=0 ; i<n;i++) a[i] += i;
15
16    printf("Después de parallel for: \n");
17
18    for(i=0 ; i<n ; i++)
19        printf("a[%d]= %d\n",i,a[i]);
20
21 }
22 |
```

CAPTURA CÓDIGO FUENTE: shared-clauseModificado.c

```
1 #include <stdio.h>
2 #ifdef _OPENMP
3 #include <omp.h>
4 #endif
5
6 main()
{
7     int i,n = 7;
8     int a[n];
9
10    for(i = 0; i<n ; i++)
11        a[i] = i+1;
12
13    #pragma omp parallel for shared(a,n) default(none)
14    for (i=0 ; i<n;i++) a[i] += i;
15
16    printf("Después de parallel for: \n");
17
18    for(i=0 ; i<n ; i++)
19        printf("a[%d]= %d\n",i,a[i]);
20
21 }
22 |
```

CAPTURAS DE PANTALLA:

```

~/Escritorio/segundo/ac/practicas/practica3/ejercicio1 o 12:58:31
$ gcc shared-clause.c -o shared-clause
shared-clause.c:6:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
^
shared-clause.c:6:1: note: C99 section 6.5.2.2 [enabled by default]
shared-clause.c:6:1: note: #include <limits.h> might help
shared-clause.c:6:1: note: use -fint-type=long if you need long
~/Escritorio/segundo/ac/practicas/practica3/ejercicio1 o 12:58:52
$ ./shared-clause
Desarrollado por: Jota Galera
a[0]= 1
a[1]= 3
a[2]= 5
a[3]= 7
a[4]= 9
a[5]= 11
a[6]= 13
~/Escritorio/segundo/ac/practicas/practica3/ejercicio1 o 12:58:54
$ ./shared-clause
Desarrollado por: Jota Galera
a[0]= 1
a[1]= 3
a[2]= 5
a[3]= 7
a[4]= 9
a[5]= 11
a[6]= 13

```

Aunque no debería de ser en el primer caso, ambos me muestran el mismo resultado.

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: el resultado es distinto porque la variable suma al estar inicializada fuera, tiene basura y, por tanto, el resultado final de “Suma”, no es fiable.

CAPTURA CÓDIGO FUENTE: private-clauseModificado.c

DENTRO:

```

private-clause.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(int argc, char** argv){
9     int i, n=7;
10    int a[n],suma;
11
12    for(i=0;i<n;i++)
13        a[i]=i;
14    #pragma omp parallel private(suma)
15    {
16        suma=1;
17        #pragma omp for
18        for(i=0;i<n;i++){
19            suma = suma + a[i];
20            printf("\nthread %d suma a[%d]/ ",omp_get_thread_num(),i);
21        printf("\n* thread %d suma = %d", omp_get_thread_num(), suma);
22    }
23    printf("\n* thread %d suma = %d", omp_get_thread_num(), suma);
24 }
25 printf("\n");
26 return 0;
27 }
28

```

FUERA:

```
private-clause.c           Telemetry Consent
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(int argc, char** argv){
9     int i, n=7;
10    int a[n], suma;
11    suma=1;
12    for(i=0;i<n;i++)
13        a[i]=i;
14    #pragma omp parallel private(suma)
15    {
16
17        #pragma omp for
18        for(i=0;i<n;i++){
19            suma = suma + a[i];
20            printf("\nthread %d suma a[%d]/ ",omp_get_thread_num(),i);
21        printf("\n* thread %d suma = %d", omp_get_thread_num(), suma);
22    }
23    printf("\n* thread %d suma = %d", omp_get_thread_num(), suma);
24    }
25    printf("\n");
26    return 0;
27 }
28
```

CAPTURAS DE PANTALLA:

DENTRO:

```
-/Escritorio/segundo/ac/practicas/practica3/ejercicio2 ⊕ 17:02:55
$ ./private_clause_in
private-clause.c           Telemetry Consent
thread 1 suma =1/
* thread 1 suma = 3
thread 0 suma =3[3]
* thread 1 suma = 6
thread 0 suma =6[6]
* thread 0 suma =1
thread 0 suma =1[1]
* thread 0 suma =3
thread 3 suma =1[1]
* thread 3 suma = 7
thread 2 suma =1[1]
* thread 2 suma = 5
thread 1 suma =1[1]
* thread 2 suma = 10
* thread 1 suma = 6
* thread 3 suma = 7
* thread 0 suma = 2
* thread 2 suma = 10
-/Escritorio/segundo/ac/practicas/practica3/ejercicio2 ⊕ 17:02:57
$ [ ]
+ Consola ejercito2 JotaGalera@jota-Erazer-P6679-MD60359:jota@jota-Erazer-P6679-MD60359: ~/Escritorio/segundo/ac/practicas/practica3/ejercicio2:~/Escritorio/segundo/ac/practicas/practica3/ejercicio2 zsh - Drop-Drag-Copy
```

FUERA:

```
-/Escritorio/segundo/ac/practicas/practica3/ejercicio2 ⊕ 17:00:24
$ ./private_clause_out
private-clause.c           Telemetry Consent
thread 0 suma =0[0]
* thread 0 suma = -757436928
thread 0 suma =0[1]
* thread 0 suma = -757436927
thread 0 suma =0[2]
* thread 1 suma =1506550610
thread 1 suma =0[3]
* thread 1 suma =1506550613
thread 1 suma =0[4]
* thread 1 suma =1506550614
thread 2 suma =0[5]
* thread 2 suma =1506550612
thread 2 suma =0[6]
* thread 2 suma =1506550617
* thread 3 suma = 1506550617
* thread 3 suma = -757436927
* thread 3 suma =1506550614
* thread 1 suma =1506550613
* thread 2 suma = 1506550617
-/Escritorio/segundo/ac/practicas/practica3/ejercicio2 ⊕ 17:01:33
$ [ ]
+ Consola ejercito2 JotaGalera@jota-Erazer-P6679-MD60359:jota@jota-Erazer-P6679-MD60359: ~/Escritorio/segundo/ac/practicas/practica3/ejercicio2:~/Escritorio/segundo/ac/practicas/practica3/ejercicio2 zsh - Drop-Drag-Copy
```

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

RESPUESTA: Ocurre que todas las hebras utilizan la misma variable y por tanto, el valor de suma es el mismo para todos los threads.

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(int argc, char** argv){
9      int i, n=7;
10     int a[n], suma;
11
12     for(i=0;i<n;i++)
13         a[i]=i;
14     #pragma omp parallel
15     {
16         suma=1;
17         #pragma omp for
18         for(i=0;i<n;i++){
19             suma = suma + a[i];
20             printf("\nthread %d suma a[%d]/ ",omp_get_thread_num(),i);
21             printf("\n* thread %d suma = %d", omp_get_thread_num(), suma);
22         }
23         printf("\n* thread %d suma = %d", omp_get_thread_num(), suma);
24     }
25     printf("\n");
26     return 0;
27 }
28

```

CAPTURAS DE PANTALLA:

```

$ ./private-clause
thread 0 suma a[0]/
thread 0 suma = 7
thread 0 suma a[1]/
* thread 0 suma = 8
thread 0 suma a[2]/
* thread 0 suma = 8
thread 1 suma a[3]/
* thread 1 suma = 11
thread 1 suma a[4]/
thread 1 suma a[5]/
* thread 1 suma = 11
thread 2 suma a[6]/
thread 2 suma a[7]/
* thread 2 suma = 16
* thread 2 suma = 16
thread 2 suma a[8]/
* thread 2 suma = 16

```

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: Si puesto que a la variable “común” se le introduce el valor del último thread, que en este caso siempre valdrá 6.

CAPTURAS DE PANTALLA:

```
./Escritorio/segundo/ac/practicas/practica3/ejercicio1 o 20:58:21
$ ./firstlastprivate
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[5] suma=4
thread 2 suma a[2] suma=9
thread 1 suma a[2] suma=2
thread 1 suma a[3] suma=5
Fuera de 'parallel' suma = 6
./Escritorio/segundo/ac/practicas/practica3/ejercicio1 o 20:59:23
$ 
+ Consola
ejercicio4 JotaGalera@jota-Erazer-P6679-MD60359:jota@jota-Erazer-P6679-MD60359: ~/Escritorio/segundo/ac/practicas/practica3/ejercicio4:~/Escritorio/segundo/ac/practicas/practica3/ejercicio4 zsh - Drop- x
```

```
int main(int argc, char** argv){
    int i, suma;
    int a[6];
    suma=0;
    for(i=0;i<6;i++)
        a[i]=i;
    #pragma omp parallel for firstprivate(suma) lastprivate(suma)
    for(i=0;i<6;i++){
        suma+=a[i];
    }
    printf("thread %d suma a[%d] suma=%d\n",omp_get_thread_num(),i,suma);
}
```

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: La cláusula `copyprivate` se utiliza en un `single` para que una thread consiga un valor para una variable y luego transmitirselo al resto de threads, por tanto si la omitimos en este código, lo que ocurre es que el valor que introducimos(`input`) no es compartido y, por tanto, solo la thread que ejecute la directiva `single` será la única con ese valor.

CAPTURA CÓDIGO FUENTE: copyprivate-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(int argc, char** argv){
9
10    int n=9, i, b[n];
11
12    for(i=0;i<n;i++) b[i]=-1;
13
14    #pragma omp parallel
15    {
16        int a;
17        #pragma omp single
18        {
19            printf("\nIntroduce valor de inicialización a: ");
20            scanf("%d",&a);
21            printf("\nSingle ejecutada por el thread %d\n", omp_get_thread_num());
22        }
23        #pragma omp for
24        for(i=0; i<n; i++) b[i]=a;
25    }
26    printf("Despues de la region parallel:\n");
27    for(i=0;i<n;i++)printf("b[%d]=%d\t",i,b[i]);
28
29    printf("\n");
30    return 0;
31 }
32

```

CAPTURAS DE PANTALLA:

```

~/Escritorio/segundo/ac/practicas/practica3/ejercicio5 ⌘ 21:04:36 ✚ git pull todos los threads
$ ./copyprivate-clause
Introduce valor de inicialización a: 3
Comunicación colectiva todos a uno (Lección 4/Tema2)
Single ejecutada por el thread 2
Despues de la region parallel:
b[0]=0 b[1]=0 b[2]=0 b[3]=0 b[4]=0 b[5]=3 b[6]=3 b[7]=0 b[8]=0
~/Escritorio/segundo/ac/practicas/practica3/ejercicio5 ⌘ 21:05:37
$ 

```

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

RESPUESTA: Simplemente antes ejecutandolo nos con el parámetro 10, nos mostraba un resultado de 45, y ahora uno de 55. Esto es debido porque la clausula reduction utiliza la variable y suma a partir de su valor el resto de los valores que tienen las threads.

CAPTURA CÓDIGO FUENTE: reduction-clauseModificado.c

```
1 #include <stdlib.h>
2 #include <stdio.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;
11     int a[n],suma=10;
12
13     if(argc <2){
14         fprintf(stderr, "Faltan iteraciones\n");
15         exit(-1);
16     }
17
18     n= atoi(argv[1]);
19     if(n>20)
20         n=20;
21
22     for(i=0;i<n;i++)
23         a[i]=i;
24
25 #pragma omp parallel for reduction(+:suma)
26
27     for(i=0;i<n;i++){
28         suma += a[i];
29         printf("\n Valor de a[%d]: %d",i,a[i]);
30     }
31
32     printf("\nFuera de 'parallel' suma = %d",suma);
33
34     printf("\n");
35     return 0;
36 }
37
```

CAPTURAS DE PANTALLA:

```

~/Escritorio/segundo/ac/practicas/practica3/ejercicio6 ⑥ 21:21:58
$ ./reduction-clause 10
Valor de a[0]: 0
Valor de a[1]: 1
Valor de a[2]: 2
Valor de a[3]: 3
Valor de a[4]: 4
Valor de a[5]: 5
Valor de a[6]: 6
Valor de a[7]: 7
Valor de a[8]: 8
Valor de a[9]: 9
Fuera de 'parallel' suma = 55
~/Escritorio/segundo/ac/practicas/practica3/ejercicio6 ⑥ 21:23:32
$ [REDACTED]
+ Consola
ejercicio6 JotaGalera@jota-Erazer-P6679-MD60359:jota@jota-Erazer-P6679-MD60359: ~/Escritorio/segundo/ac/practic
Cláusula reduction. Salida
AC RTC
○ mancia@mancia-ubuntu: ~/docencia/OpenMP/lec [ ]
Archivo Editar Ver Terminal Ayuda
$ gcc -O2 -fopenmp -o reduction-clause reduction-clause.c
$ export OMP_NUM_THREADS=3
$ reduction-clause 10

```

- 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

```

1 #include <stdlib.h>
2 #include <stdio.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;
11     int a[n], suma=10;
12     int suma_privada=0;
13
14     if(argc <2){
15         fprintf(stderr, "Faltan iteraciones\n");
16         exit(-1);
17     }
18
19     n= atoi(argv[1]);
20     if(n>20)
21         n=20;
22
23     for(i=0;i<n;i++)
24         a[i]=i;
25
26 #pragma omp parallel firstprivate(suma_privada)
27 {
28     #pragma omp for
29     for(i=0;i<n;i++){
30         suma += a[i];
31         printf("\n Valor de a[%d]: %d",i,a[i]);
32     }
33     #pragma omp critical
34         suma+=suma_privada;
35 }
36
37
38     printf("\nFuera de 'parallel' suma = %d",suma);
39
40     printf("\n");
41     return 0;
42 }
43

```

CAPTURAS DE PANTALLA:

```

$ ./reduction-clause 10
Clausula reduction. Salida
Valor de a[0]: 0
Valor de a[1]: 1
Valor de a[2]: 2
Valor de a[3]: 3
Valor de a[4]: 4
Valor de a[5]: 5
Valor de a[6]: 6
Valor de a[7]: 7
Fuera de 'parallel' suma = 55

$ ./reduction-clause 10
Clausula reduction. Salida
Valor de a[0]: 0
Valor de a[1]: 1
Valor de a[2]: 2
Valor de a[3]: 3
Valor de a[4]: 4
Valor de a[5]: 5
Valor de a[6]: 6
Valor de a[7]: 7
Fuera de 'parallel' suma = 55

```

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

```

matrix_secuencial.c
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <time.h>
4
5 #define VECTOR_GLOBAL
6 //##define VECTOR_DYNAMIC
7
8 #ifdef VECTOR_GLOBAL
9 #define MAX 1000 //=2^25
10 int v1[MAX], v2[MAX], m[MAX][MAX];
11#endif
12
13
14 int main(int argc, char* argv[]){
15     int i,j;
16
17     struct timespec cgt1,cgt2; // para tiempos de ejecución
18     double ncgt;
19
20     //Leer argumento de entrada
21     if(argc<2){
22         printf("Falta tamaño de matriz\n");
23         exit(1);
24     }
25     unsigned int N = atoi(argv[1]); // MAXIMO N = 2^32-1
26
27
28     #ifdef VECTOR_GLOBAL
29     if (N>MAX)
30     #endif
31     #ifdef VECTOR_DYNAMIC
32     int *v1,*v2,*m;
33     v1 = (int *) malloc(N*sizeof(int));
34     v2 = (int *) malloc(N*sizeof(int));
35     m = (int *) malloc(N*sizeof(int));
36     for(i=0 ; i<N ; i++)
37         m[i] = (int *) malloc(N*sizeof(int));
38
39     if( (v1==NULL) || (v2==NULL) || (m==NULL)){
40         printf("Error en la reserva de memoria para los vectores o matriz\n");
41         exit(1);
42     }
43     #endif
44
45
46     //Inicialización de vectores
47     for(i=0; i<N ; i++){
48         v2[i] = i+1;
49         printf("[%d]",v2[i]);
50     }
51     printf("\n");
52     for(i=0;i<N;i++){
53         for(j=0; j<N ; j++){
54             m[i][j] = i+j+2;
55             if(m[i][j]>=10){
56                 printf("[%d]",m[i][j]);
57             }
58             else{
59                 printf("[%d]",m[i][j]);
60             }
61         }
62     }
63
64     printf("\n\n");
65     clock_gettime(CLOCK_REALTIME,&cg1);
66
67     //Suma de vectores
68     for(i=0; i<N ; i++)
69         for(j=0; j<N ; j++)
70             v1[i]=m[i][j]*v2[j];
71
72     clock_gettime(CLOCK_REALTIME,&cg2);
73     ncgt=(double) (cg2.tv_sec-cg1.tv_sec)+(double)
74         ((cg2.tv_nsec-cg1.tv_nsec)/(1.e9));
75
76     //IMPRIME RESULTADO Y TIEMPO DE EJECUCION
77     printf("Tiempos(seg):%f\t Tamaño Matriz:%d\t \n V1[0]=%d\t V2[0]=(%d=%d)\t / \n V1[%d]=%d\t V2[%d]=(%d=%d)\t / \n",ncgt,N,v1[0],v2[0],N-1,N-1,v1[N-1],v2[N-1],N-1);
78
79     #ifdef VECTOR_DYNAMIC
80     free(v1);
81     free(v2);
82     for(i=0; i<N ; i++)
83         free(m[i]);
84     free(m);
85     #endif
86
87     return 0;
88 }

```

CAPTURAS DE PANTALLA:

N=8

```

$ gcc matrix_secuencial.c -o matriz
[Escriptorio/segundo/ac/practicas/practica3/ejercicios o 0:40:52]
[1][2][3][4][5][6][7][8]
[02][03][04][05][06][07][08][09] New [10] [11] [12] [13] [14] [15] [16]
[03][04][05][06][07][08][09][10][11]
[04][05][06][07][08][09][10][11][12]
[05][06][07][08][09][10][11][12][13]
[06][07][08][09][10][11][12][13][14]
[07][08][09][10][11][12][13][14][15]
[08][09][10][11][12][13][14][15][16]
Tiempo(seg): 0.000002138      / Tamaño Matriz:8
V1[0]=0    / V2[0]=(240=24)   /
V1[7]=7    / V2[7]=(492=168)  /
[Escriptorio/segundo/ac/practicas/practica3/ejercicios o 0:40:54]
$ ./matriz_11
[1][2][3][4][5][6][7][8][9][10][11]
[02][03][04][05][06][07][08][09][10][11][12]
[03][04][05][06][07][08][09][10][11][12][13]
[04][05][06][07][08][09][10][11][12][13][14]

```

N=11

```

./Escriptorio/segundo/ac/practicas/practica3/ejercicio8 o 0:40:54
$ ./matriz*vector <matriz> <vector> <resultado>
[1][2][3][4][5][6][7][8][9][10][11]
[02][03][04][05][06][07][08][09][10][11][12][13]
[03][04][05][06][07][08][09][10][11][12][13][14]
[04][05][06][07][08][09][10][11][12][13][14][15]
[05][06][07][08][09][10][11][12][13][14][15][16]
[06][07][08][09][10][11][12][13][14][15][16][17]
[07][08][09][10][11][12][13][14][15][16][17][18]
[08][09][10][11][12][13][14][15][16][17][18][19]
[09][10][11][12][13][14][15][16][17][18][19][20]
[10][11][12][13][14][15][16][17][18][19][20][21]
[11][12][13][14][15][16][17][18][19][20][21][22]
[12][13][14][15][16][17][18][19][20][21][22]

Tiempo(seg., j: 0.09900367           / Tamaño Matriz:11
V1[0]=M[0][0]*V[0] (57*22)*21   /
V1[18]=M[1][0]*V[18] V2[18]=(252+22)*11 /
~/Escriptorio/segundo/ac/practicas/practica3/ejercicio8 o 0:43:07
$ 
ejercicio8 JotaGalera@jota-Erazer-P6679-MD60359:jota@jota-Erazer-P6679-MD60359: ~/Escriptorio/segundo/ac/practicas/practica3/ejercicio8:~/Escriptorio/segundo/ac/practicas/practica3/ejercicio8 zsh - Drop-Box x

```

CAPTURA CODIGO FUENTE: pmv-secuencial.c

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 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 eliminan el código de la suma).

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 eliminan el código de la suma).

CAPTURA CÓDIGO FUENTE : pmv-OpenMP-a.c

```

1  /*
2   Producto de Matriz cuadrada por Vector
3
4   Para compilar usar: -lrt; real time library;
5   gcc -O2 pmv-secuencial.c -o pmv-secuencial -lrt
6   gcc -O2 -S pmv-secuencial.c -lrt
7   Para ejecutar use:pmv-secuencial <tamaño matriz>
8 */
9 #include <stdlib.h>
10 #include <stdio.h>
11 #include <time.h>
12 #include <omp.h>
13
14 //Define VECTOR_GLOBAL
15 #define VECTOR_DYNAMIC
16
17 #define PRINT_ALL
18
19 #ifdef VECTOR_GLOBAL
20     #define MAX 100000
21     int v1[MAX], v2[MAX], m[MAX][MAX];
22 #endif
23
24
25
26 int main(int argc, char** argv){
27     int i,j,suma_local;
28
29     struct timespec cgt1,cgt2; double ncp; //para tiempo de ejecución
30
31     //Leer argumento de entrada (nº de componentes del vector)
32     if (argc<2){
33         printf("Falta tamaño de la matriz\n");
34         exit(1);
35     }
36
37     unsigned int N = atoi(argv[1]); // Máximo N <2^32-1=429496729 (sizeof(unsigned int) = 4 B)
38
39     #ifdef VECTOR_GLOBAL
40         if (N>MAX) N=MAX;
41     #endif
42     #ifdef VECTOR_DYNAMIC
43         int *v1, *v2, *m;
44         v1 = (int*) malloc(N*sizeof(int));// malloc necesita el tamaño en bytes
45         v2 = (int*) malloc(N*sizeof(int)); //si no hay espacio suficiente malloc devuelve NULL
46         m = (int**)malloc(N*sizeof(int*));
47         for(i=0; i<N; i++){
48             m[i] = (int*) malloc(N*sizeof(int));
49
50             if ( (v1==NULL) || (v2==NULL) || (m==NULL)){
51                 printf("Error en la reserva de espacio para los vectores/matriz\n");
52                 exit(-2);
53             }
54         }
55     #endif
56
57     //Inicializar vectores de forma paralela
58     //repartiendo el for en las distintas hebras
59     #pragma omp parallel for
60     for(i=0; i<N; i++){
61         v2[i] = i+1;
62     }
63     //Inicializar matriz
64     //privatizar la "j", puesto que cada hebra se
65     //encargará de poner valor a cada columna
66
67     //SE PODRIA PONER TAMBIEN LA "i", PERO DE ESTA MANERA CADA COLUMNA TENDRA UN +
68     //EN CADA POSICION HACIA LA DERECHA
69     #pragma omp parallel for private(j)
70     for(i=0;i<N;i++){
71         for(j=0;j<N;j++){
72             m[i][j] = i+j+2;
73             printf("Para la hebra:%d , el valor de j es:%d\n",omp_get_thread_num(),j );
74         }
75     }
76
77     #ifdef PRINT_ALL
78         //Mostrar vector
79         for(i=0;i<N;i++)
80             printf("[%d]",v2[i]);
81         printf("\n");
82         //Mostrar matriz
83         for(i=0;i<N;i++){
84             printf("\n");
85             for(j=0;j<N;j++){
86                 if(m[i][j]>10)
87                     printf("[%d]",m[i][j]);
88                 else {
89

```

```

00         }
01     }
02     printf("\n\n");
03 #endif
04     clock_gettime(CLOCK_REALTIME,&cgt1);
05
06     //Calcular suma de vectores
07     //Valor suma_local como private inicializada a 0
08     //dentro, la idea es almacenar en cada posición de v1
09     //el valor de cada vector para realizar una suma total después
10     #pragma omp parallel for private(suma_local)
11     for(i=0; i<N; i++){
12         suma_local = 0;
13         for(j=0; j<N; j++)
14             suma_local+=m[i][j];
15         v1[i]=suma_local*v2[i];
16     }
17
18
19     clock_gettime(CLOCK_REALTIME,&cgt2);
20     ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
21
22
23
24
25     //Imprimir resultado de la suma y el tiempo de ejecución
26     //printf("Tiempo(seg.):%11.9f\t v1[0]=%d\t v1[%d]=%d\n", ncgt, v1[0], N-1, v1[N-1]);
27     printf("Tiempo(seg.):%11.9f\t / Tamaño Matriz:%u\t \n V1[0]=%u\t v1[%d]=%u\t v2[0]=%u\t v2[%d]=%u\t / \n v1[%d]=%u\t m[%d][%d]=%u\t / \n", ncgt, N, v1[0], m[0][0], v2[0], N-1, N-1, v1[N-1], m[N-1]);
28
29     #ifdef PRINT_ALL
30     for(i=0;i<N;i++)printf("v1[%d]=%d\n",i,v1[i]);
31     #endif
32
33
34
35     #ifndef VECTOR_DYNAMIC
36     free(v1); // Libera el espacio reservado para v1
37     free(v2); // libera el espacio reservado para v2
38     for(i=0;i<N;i++)
39         free(m[i]);
40     free(m);
41     #endif
42     return 0;
43 }
44

```

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

```

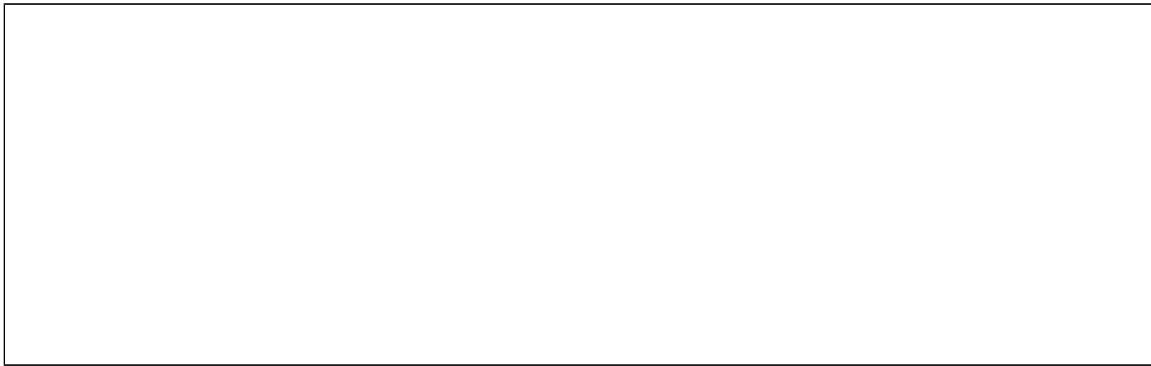
/*
2  Producto de Matriz cuadrada por Vector
3
4  Para compilar usar:-lrt: real time library:
5  gcc -O2 pmv-secuencial.c -o pmv-secuencial -lrt
6  gcc -O2 -S pmv-secuencial.c -lrt
7  Para ejecutar use:pmv-secuencial <tamaño matriz>
8 */
9 #include <stdlib.h>
10 #include <stdio.h>
11 #include <time.h>
12 #include <omp.h>
13 //Define VECTOR GLOBAL
14 #define VECTOR_DYNAMIC
15
16
17 #define PRINT_ALL
18
19 #ifndef VECTOR_GLOBAL
20 #define MAX 100000
21 int v1[MAX], v2[MAX], m[MAX][MAX];
22 #endif
23
24
25 int main(int argc, char** argv){
26     int i,j,suma_local,suma_privada;
27
28     struct timespec cgt1,cgt2; double ncgt; //para tiempo de ejecución
29
30     //Leer argumento de entrada (no de componentes del vector)
31     if (argc<2){
32         printf("Falta tamaño de la matriz\n");
33         exit(-1);
34     }
35
36     unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=429496729 (sizeof(unsigned int) = 4 B)
37
38     #ifndef VECTOR_GLOBAL
39     if (N>MAX) N=MAX;
40     #endif
41     #ifndef VECTOR_DYNAMIC
42     int *v1, *v2, **m;
43     v1 = (int*) malloc(N*sizeof(int));// malloc necesita el tamaño en bytes
44     v2 = (int*) malloc(N*sizeof(int)); // si no hay espacio suficiente malloc devuelve NULL
45

```

```

46     for(i=0; i<N;i++)
47         m[i] = (int*) malloc(N*sizeof(int));
48
49     if ( (v1==NULL) || (v2==NULL) || (m==NULL)){
50         printf("Error en la reserva de espacio para los vectores/matriz\n");
51         exit(-2);
52     }
53 #endif
54
55 //Inicializar vectores
56 #pragma omp parallel for
57 for(i=0; i<N; i++){
58     v2[i] = i*i;
59 }
60
61 //Inicializar matriz
62 for(i=0;i<N;i++){
63     //Igual idea del apartado, pero en este caso
64     //no necesitamos el private, puesto que con
65     //#pragma omp parallel for repartimos directamente
66     //el trabajo en las diferentes hebras que se crean.
67     #pragma omp parallel for
68     for(j=0;j<N;j++){
69         m[i][j] = i+j+2;
70     }
71
72 #ifdef PRINT_ALL
73     //Mostrar vector
74     for(i=0;i<N;i++)
75         printf("%d",v2[i]);
76     printf("\n");
77     //Mostrar matriz
78     for(i=0;i<N;i++){
79         printf("\n");
80         for(j=0;j<N;j++){
81             if(m[i][j]>10){
82                 printf("%d",m[i][j]);
83             }
84             else {
85                 printf("%d",m[i][j]);
86             }
87         }
88     }
89     printf("\n\n");
90 #endif
91     CLOCK_gettime(CLOCK_REALTIME,&cgt1);
92
93 //Calcular suma de vectores
94
95 for(i=0; i<N; i++){
96     suma_local = 0;
97     suma_privada = 0;
98     #pragma omp parallel firstprivate(suma_privada)
99     {
100
101         #pragma omp for
102         for(j=0;j<N;j++){
103             suma_privada+=m[i][j];
104         }
105
106         #pragma omp atomic
107         suma_local+=suma_privada;
108
109     }
110
111     v1[i]=suma_local*v2[i];
112 }
113
114
115
116
117 clock_gettime(CLOCK_REALTIME,&cgt2);
118 ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
119
120
121
122 //Imprimir resultado de la suma y el tiempo de ejecución
123
124 printf("Tiempo(seg.):%11.9f\n", v1[0]=#d,v1[N-1],ncgt, v1[0],N-1,v1[N-1]);
125 //printf("Tiempo(seg.):%11.9f / Tamaño Matriz:%u(t/ (n V1[0]*m[0]/(v2[0]/(%d=%d)*%d)) / (n V1[%d]=M[%d]*%d)*V2[%d]/(%d=%d*%d) )\n", ncgt,N,v1[0],m[0],v2[0],N-1,N-1,N-1,v1[N-1],m[N-1]);
126
127 #ifdef PRINT_ALL
128     for(i=0;i<N;i++)printf("v1[%d]=%d\n",i,v1[i]);
129 #endif
130
131
132 #ifdef VECTOR_DYNAMIC
133 free(v1); // Libera el espacio reservado para v1
134 free(v2); // libera el espacio reservado para v2
135 #endif
136
137
138 #ifdef PRINT_ALL
139     for(i=0;i<N;i++)printf("v1[%d]=%d\n",i,v1[i]);
140 #endif
141
142 #ifdef VECTOR_DYNAMIC
143 free(v1); // libera el espacio reservado para v1
144 free(v2); // libera el espacio reservado para v2
145 for(i=0;i<N;i++)
146     free(m[i]);
147 free(m);
148 #endif
149 return 0;
150 }
151

```



CAPTURAS DE PANTALLA:

```
./Escritorio/segundo/ac/practicas/practica3/ejercicio9 @ 1:51:18
$ ./matriz_B
matrix_b.c -- ~/Escritorio/segundo/ac/practicas -- Atom
[1][2][3][4][5][6][7][8]

[02][03][04][05][06][07][08][09]
[03][04][05][06][07][08][09][10] PRINT_ALL
[04][05][06][07][08][09][10][11]
[05][06][07][08][09][10][11][12]
[06][07][08][09][10][11][12][13]
[07][08][09][10][11][12][13][14]
[08][09][10][11][12][13][14][15]
[09][10][11][12][13][14][15][16] m[1][1] = 1+i+j

Tiempo(seg.):0.00000524 / Tamaño Matriz:8
V1[0]=160
V1[1]=180
V1[2]=200
V1[3]=220
V1[4]=380
V1[5]=500
V1[6]=644
V1[7]=800
+ Consola
ejercicio9 JotaGalera@jota-Erazer-P6679-MD60359:jota@jota-Erazer-P6679-MD60359: ~/Escritorio/segundo/ac/practicas/practica3/ejercicio9:-/Escritorio/segundo/ac/practicas/practica3/ejercicio9 zsh - Drop- @
```

Ejecución matriz a

```
./Escritorio/segundo/ac/practicas/practica3/ejercicio9 @ 1:52:33
$ ./matriz_B
matrix_b.c -- ~/Escritorio/segundo/ac/practicas -- Atom
[1][2][3][4][5][6][7][8]

[02][03][04][05][06][07][08][09]
[03][04][05][06][07][08][09][10] PRINT_ALL
[04][05][06][07][08][09][10][11]
[05][06][07][08][09][10][11][12]
[06][07][08][09][10][11][12][13]
[07][08][09][10][11][12][13][14]
[08][09][10][11][12][13][14][15]
[09][10][11][12][13][14][15][16]

Tiempo(seg.):0.00004698 v1[0]=44 v1[7]=800
v1[0]=44
v1[1]=160
v1[2]=180
v1[3]=220
v1[4]=380
v1[5]=500
v1[6]=644
v1[7]=800
+ Consola
ejercicio9 JotaGalera@jota-Erazer-P6679-MD60359:jota@jota-Erazer-P6679-MD60359: ~/Escritorio/segundo/ac/practicas/practica3/ejercicio9:-/Escritorio/segundo/ac/practicas/practica3/ejercicio9 zsh - Drop- @
```

Ejecución matriz b

- 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

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <time.h>
4 #include <omp.h>
5 //##define VECTOR_GLOBAL
6 #define VECTOR_DYNAMIC
7
8
9 //##define PRINT_ALL
10
11 #ifdef VECTOR_GLOBAL
12     #define MAX 100000
13     int v1[MAX], v2[MAX], m[MAX][MAX];
14 #endif
15
16
17 int main(int argc, char** argv){
18     int i,j,suma_local;
19
20     struct timespec cgt1,cgt2; double nrgt; //para tiempo de ejecución
21
22     //Leer argumento de entrada (no de componentes del vector)
23     if (argc<2){
24         printf("Falta tamaño de la matriz\n");
25         exit(-1);
26     }
27
28     unsigned int N = atoi(argv[1]); // Máximo N = 2^32-1=429496729 (sizeof(unsigned int) = 4 B)
29
30     #ifdef VECTOR_GLOBAL
31     if (N>MAX) N=MAX;
32     #endif
33     #ifdef VECTOR_DYNAMIC
34         int *v1, *v2, **m;
35         v1 = (int*) malloc(N*sizeof(int));// malloc necesita el tamaño en bytes
36         v2 = (int*) malloc(N*sizeof(int)); //si no hay espacio suficiente malloc devuelve NULL
37         m = (int**)malloc(N*sizeof(int*));
38         for(i=0; i<N; i++){
39             m[i] = (int*) malloc(N*sizeof(int));
40
41         if ( (v1==NULL) || (v2==NULL) || (m==NULL)){
42             printf("Error en la reserva de espacio para los vectores/matriz\n");
43             exit(-2);
44         }
45     }
46
47     #endif
48     //Inicializar vectores
49     #pragma omp parallel for
50     for(i=0; i<N; i++){
51         v2[i] = i+1;
52     }
53     //Inicializar matriz
54     for(i=0;i<N;i++){
55         #pragma omp parallel for
56         for(j=0;j<N;j++){
57             m[i][j] = i+j*2;
58         }
59     }
60
61     #ifdef PRINT_ALL
62         //Mostrar vector
63         for(i=0;i<N;i++){
64             printf("%d",v2[i]);
65             printf("\n");
66         //Mostrar matriz
67         for(i=0;i<N;i++){
68             printf("\n");
69             for(j=0;j<N;j++){
70                 if(m[i][j]>=10){
71                     printf("%d",m[i][j]);
72                 }
73                 else {
74                     printf("%d",m[i][j]);
75                 }
76             }
77             printf("\n\n");
78         #endif
79         clock_gettime(CLOCK_REALTIME,&cg1);
80
81         //Calcular suma de vectores
82
83         for(i=0; i<N; i++){
84             suma_local = 0;
85             #pragma omp parallel for reduction(+:suma_local)
86             for(j=0;j<N;j++){
87                 suma_local+=m[j][i];
88             }
89             v1[i]=suma_local*v2[i];
90     }

```

```

73         printf("%d",m[i][j]);
74     }
75   }
76   printf("\n\n");
77 #endif
78 clock_gettime(CLOCK_REALTIME,&cgt1);
79
80 //Calcular suma de vectores
81
82 for(i=0; i<N; i++){
83   suma_local = 0;
84   #pragma omp parallel for reduction(+:suma_local)
85   for(j=0;j<N;j++)
86     suma_local+=m[j][i];
87   v1[i]=suma_local+v2[i];
88 }
89
90
91 clock_gettime(CLOCK_REALTIME,&cgt2);
92 ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
93
94
95
96
97 //Imprimir resultado de la suma y el tiempo de ejecución
98
99 printf("Tiempo(seg.):%.11.9f\n", v1[0]); //d\nt v1[%d]=%d\n", ncgt, v1[0],N-1,v1[N-1]);
100 //printf("Tiempo(seg.):%.11.9f / Tamaño Matriz:%d\n",v1[0]*v2[0]/(N*N));
101
102 #ifdef PRINT_ALL
103   for(i=0;i<N;i++)printf("v1[%d]=%d\n",i,v1[i]);
104 #endif
105
106
107 #ifdef VECTOR_DYNAMIC
108   free(v1); // Libera el espacio reservado para v1
109   free(v2); // libera el espacio reservado para v2
110   for(i=0;i<N;i++)
111     free(m[i]);
112   free(m);
113 #endif
114   return 0;
115 }
116

```

CAPTURAS DE PANTALLA:

```

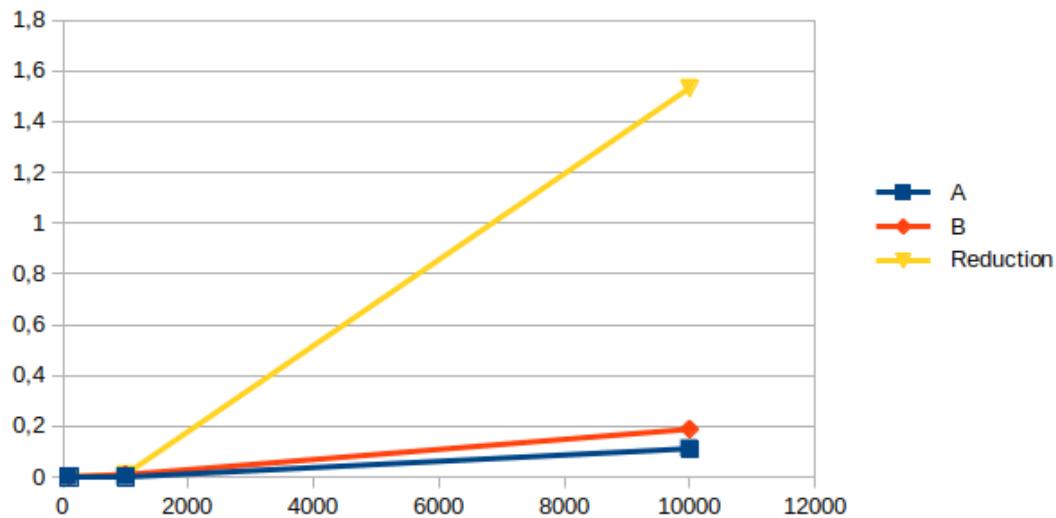
~/Escritorio/segundo/ac/practicas/practica3/ejercicio10 ◊ 2:00:00
$ ./matriz_reduc 8
Tiempo(seg.):0.000000637    v1[0]=44    v1[7]=800      CAPTURAS DE PANTALLA:
~/Escritorio/segundo/ac/practicas/practica3/ejercicio10 ◊ 2:00:12
$ [ ] Ayudándose de una hoja de cálculo (recuerde que en las aulas está instalado OpenOffice)
+ Consola ejercicio10 JotaGalera@jota-Erazer-P6679-MD60359:jota@jota-Erazer-P6679-MD60359: ~/Escritorio/segundo/ac/practicas/practica3/ejercicio10:~/Escritorio/segundo/ac/practicas/practica3/ejercicio10 zsh - Dr0-0-0-0

```

- 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):

ATCGRID



Mi PC

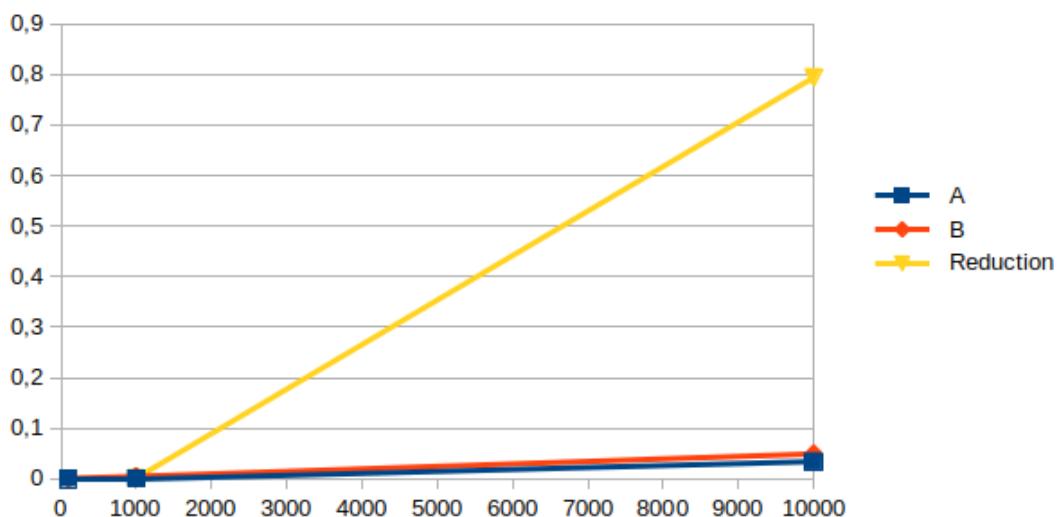


TABLA Y GRÁFICA (por ejemplo para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N: un N entre 30000 y 100000, y otro entre 5000 y 30000):

COMENTARIOS SOBRE LOS RESULTADOS:

Como podemos observar en las gráficas, el tiempo de ejecución del programa A es menor que el del B y el de Reducion, por tanto es el más eficiente.