

BP2.pdf



PruebaAlien



Arquitectura de Computadores



2º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación
Universidad de Granada



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.





**KEEP
CALM
AND
ESTUDIA
UN POQUITO**



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 2. Programación paralela II: Cláusulas OpenMP

Estudiante (nombre y apellidos):

Grupo de prácticas y profesor de prácticas:

Fecha de entrega:

Fecha evaluación en clase:

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

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)`? (b) Resuelva el problema generado sin eliminar `default(none)`.Añada el código con la modificación al cuaderno de prácticas. (Añada capturas de pantalla que muestren lo que ocurre)

RESPUESTA: Al poner `default(none)` estamos especificando que las variables que estan fuera del parallel no se compartan, dando un error de compilación, para solucionarlo hay que decirle que las variables que se usan en parallel que se inicializan fuera del parallel se comparta, puesto que se usa dentro del parallel. La variable `i` como se usa como indice no hace falta indicarle el shared, puesto que es privada.

CAPTURA CÓDIGO FUENTE: `shared-clauseModificado.c`

```
1  #include <stdio.h>
2
3  #ifdef _OPENMP
4      #include <omp.h>
5  #endif
6
7  //gcc -O2 -fopenmp -o shared_clause shared_clause.c
8
9  int main(int argc, char const *argv[])
10 {
11     int i, n=7;
12     int a[n];
13
14     #pragma omp parallel for default(none) shared(a,n)
15     for(i=0;i<n;++i) a[i] = i + 1;
16
17     printf("Despues del parallel for:\n");
18
19     for (i = 0; i < n; ++i) printf("a[%d] = %d\n",i,a[i] );
20
21     return 0;
22 }
```

CAPTURAS DE PANTALLA:

```
[b4estudiante10@atcgrid ejer1]$ ./shared_clause
Despues del parallel for:
a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5
a[5] = 6
a[6] = 7
[b4estudiante10@atcgrid ejer1]$
```

2. Añadir a lo necesario a `private-clause.c` para que imprima suma fuera de la región `parallel` e inicializar `suma` a un valor distinto de 0. Ejecute varias veces el código ¿Qué imprime el código fuera del `parallel`? (muéstrello con una captura de pantalla) ¿Qué ocurre si en esta versión de `private-clause.c` se inicia la variable `suma` fuera de la construcción `parallel` en lugar de dentro? Razona su respuesta (añada capturas de pantalla que muestren lo que ocurre). Añadir el código con las modificaciones al cuaderno de prácticas.

RESPUESTA: Cuando inicializo el valor dentro de `parallel` el valor que muestra de `suma` fuera de `parallel` es 0 puesto que lo hemos puesto privado en `parallel`, de tal forma que cada hebra tendrá su propia `suma` y al salir fuera de `parallel` se pierde el valor que tuviese `suma` dentro de `parallel`.

En caso de que se inicialice `suma` fuera ocurriría lo mismo, pero en vez de mostrar 0 mostraría 4 que es el valor con el que se ha inicializado.

CAPTURA CÓDIGO FUENTE: `private-clauseModificado.c`

```
1 #include <stdio.h>
2
3 #ifdef _OPENMP
4     #include <omp.h>
5 #else
6     #define omp_get_thread_num() 0
7 #endif
8
9 //gcc -O2 -fopenmp -o private_clause private_clause.c
10
11 int main(int argc, char const *argv[])
12 {
13     int i, n=7;
14     int a[n], suma;
15
16     for (i = 0; i < n; ++i)
17     {
18         a[i]=i;
19     }
20
21     #pragma omp parallel private(suma)
22     {
23         suma=4;
24         #pragma omp for
25         for(i=0;i<n;+i){
26             suma = suma + a[i];
27             printf("thread %d, suma a[%d] = %d / ", omp_get_thread_num(),i,a[i]);
28         }
29
30     }
31
32     printf("\nthread %d suma = %d \n", omp_get_thread_num(), suma);
33
34
35     return 0;
36 }
```

Inicializamos
suma dentro
de parallel con
otro valor

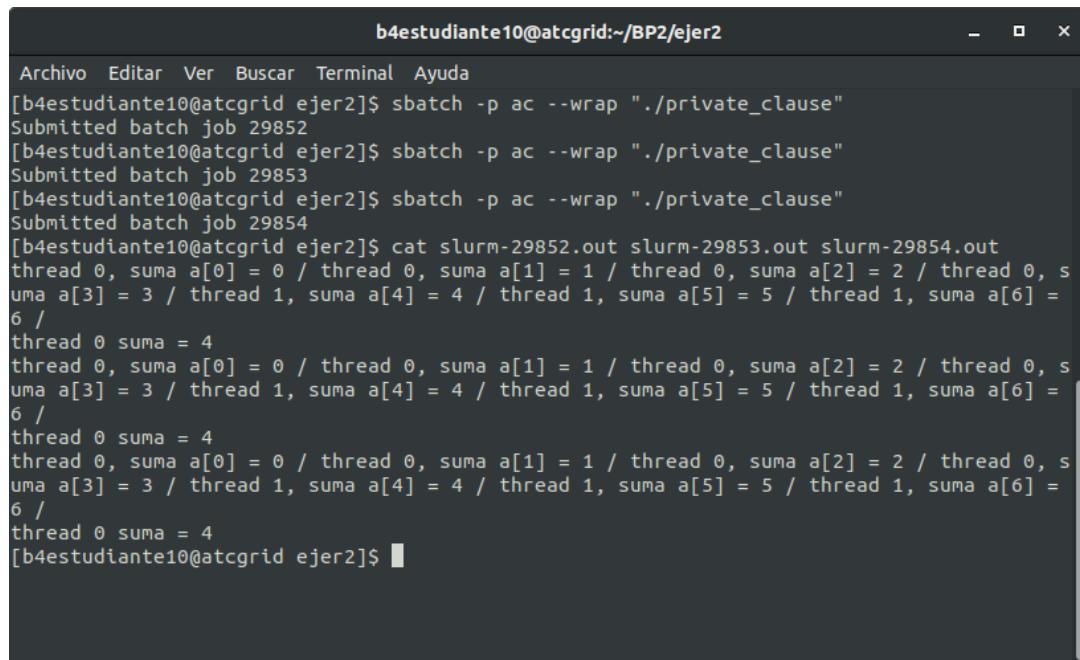
Inicializamos
suma fuera de
parallel con
otro valor

```
[b4estudiante10@atcgrid ejer2]$ sbatch -p ac --wrap "./private_clause"
Submitted batch job 29849
[b4estudiante10@atcgrid ejer2]$ cat slurm-29849.out
thread 0, suma a[0] = 0 / thread 0, suma a[1] = 1 / thread 0, suma a[2] = 2 / thread 0, s
uma a[3] = 3 / thread 1, suma a[4] = 4 / thread 1, suma a[5] = 5 / thread 1, suma a[6] =
6 /
thread 0 suma = 0
[b4estudiante10@atcgrid ejer2]$ █
1 #include <stdio.h>
2
3 #ifdef _OPENMP
4     #include <omp.h>
5 #else
6     #define omp_get_thread_num() 0
7 #endif
8
9 //gcc -O2 -fopenmp -o private_clause private_clause.c
10
11 int main(int argc, char const *argv[])
12 {
13     int i, n=7;
14     int a[n], suma;
15
16     for (i = 0; i < n; ++i)
17     {
18         a[i]=i;
19     }
20
21     suma=4;
22
23 #pragma omp parallel private(suma)
24 {
25     //suma=4;
26     #pragma omp for
27     for(i=0;i<n;++i){
28         suma = suma + a[i];
29         printf("thread %d, suma a[%d] = %d / ", omp_get_thread_num(),i,a[i]);
30     }
31
32 }
33
34
35     printf("\nthread %d suma = %d \n", omp_get_thread_num(), suma);
36
37     return 0;
38 }
```

CAPTURAS DE PANTALLA:

Inicializacion suma dentro de parallel con otro valor

Inicializo suma fuera de parallel con otro valor



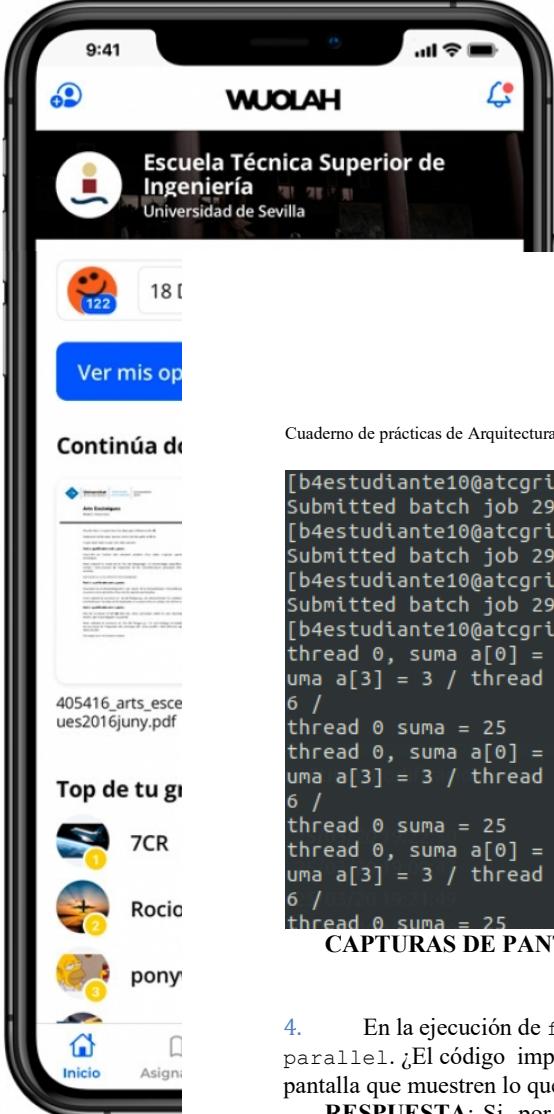
```
b4estudiante10@atcgrid:~/BP2/ejer2
Archivo Editar Ver Buscar Terminal Ayuda
[b4estudiante10@atcgrid ejer2]$ sbatch -p ac --wrap "./private_clause"
Submitted batch job 29852
[b4estudiante10@atcgrid ejer2]$ sbatch -p ac --wrap "./private_clause"
Submitted batch job 29853
[b4estudiante10@atcgrid ejer2]$ sbatch -p ac --wrap "./private_clause"
Submitted batch job 29854
[b4estudiante10@atcgrid ejer2]$ cat slurm-29852.out slurm-29853.out slurm-29854.out
thread 0, suma a[0] = 0 / thread 0, suma a[1] = 1 / thread 0, suma a[2] = 2 / thread 0, s
uma a[3] = 3 / thread 1, suma a[4] = 4 / thread 1, suma a[5] = 5 / thread 1, suma a[6] =
6 /
thread 0 suma = 4
thread 0, suma a[0] = 0 / thread 0, suma a[1] = 1 / thread 0, suma a[2] = 2 / thread 0, s
uma a[3] = 3 / thread 1, suma a[4] = 4 / thread 1, suma a[5] = 5 / thread 1, suma a[6] =
6 /
thread 0 suma = 4
thread 0, suma a[0] = 0 / thread 0, suma a[1] = 1 / thread 0, suma a[2] = 2 / thread 0, s
uma a[3] = 3 / thread 1, suma a[4] = 4 / thread 1, suma a[5] = 5 / thread 1, suma a[6] =
6 /
thread 0 suma = 4
[b4estudiante10@atcgrid ejer2]$
```

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

RESPUESTA: Lo que ocurre que al quitarle `private` la variable `suma` se ha vuelto compartida por defecto con todas las hebras, obteniendo la suma de total de todas las hebras.

CAPTURA CÓDIGO FUENTE: private-clauseModificado3.c

```
1 #include <stdio.h>
2
3 #ifdef _OPENMP
4     #include <omp.h>
5 #else
6     #define omp_get_thread_num() 0
7 #endif
8
9 //gcc -O2 -fopenmp -o private_clause private_clause.c
10
11 int main(int argc, char const *argv[])
12 {
13     int i, n=7;
14     int a[n], suma;
15
16     for (i = 0; i < n; ++i)
17     {
18         a[i]=i;
19     }
20
21     suma=4;
22
23     #pragma omp parallel
24     {
25
26         #pragma omp for
27         for(i=0;i<n;++i){
28             suma = suma + a[i];
29             printf("thread %d, suma a[%d] = %d / ", omp_get_thread_num(),i,a[i]);
30         }
31
32     }
33
34     printf("\nthread %d suma = %d \n", omp_get_thread_num(), suma);
35
36     return 0;
37 }
```



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the App Store GET IT ON Google Play

Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática

```
[b4estudiante10@atcgrid ejer2]$ sbatch -p ac --wrap "./private_clause_modificado3"
Submitted batch job 29920
[b4estudiante10@atcgrid ejer2]$ sbatch -p ac --wrap "./private_clause_modificado3"
Submitted batch job 29921
[b4estudiante10@atcgrid ejer2]$ sbatch -p ac --wrap "./private_clause_modificado3"
Submitted batch job 29922
[b4estudiante10@atcgrid ejer2]$ cat slurm-29920.out slurm-29921.out slurm-29922.out
thread 0, suma a[0] = 0 / thread 0, suma a[1] = 1 / thread 0, suma a[2] = 2 / thread 0, s
uma a[3] = 3 / thread 1, suma a[4] = 4 / thread 1, suma a[5] = 5 / thread 1, suma a[6] =
6 /
thread 0 suma = 25
thread 0, suma a[0] = 0 / thread 0, suma a[1] = 1 / thread 0, suma a[2] = 2 / thread 0, s
uma a[3] = 3 / thread 1, suma a[4] = 4 / thread 1, suma a[5] = 5 / thread 1, suma a[6] =
6 /
thread 0 suma = 25
thread 0, suma a[0] = 0 / thread 0, suma a[1] = 1 / thread 0, suma a[2] = 2 / thread 0, s
uma a[3] = 3 / thread 1, suma a[4] = 4 / thread 1, suma a[5] = 5 / thread 1, suma a[6] =
6 /
thread 0 suma = 25
```

CAPTURAS DE PANTALLA:

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 (añada capturas de pantalla que muestren lo que ocurre).

RESPUESTA: Si, por que guarda el ultimo valor de la ultima hebra que a accedido a la sección `parallel` machacando el valor del `firstprivate`. De tal forma si le quitamos el `lastprivate` obtendría suma el valor de la primera hebra que a accedido a la sección `parallel`.

CAPTURAS DE PANTALLA:

```
[b4estudiante10@atcgrid ejer3]$ ./firstlastprivate
thread 0, suma a[0] SUMA = 0
thread 2, suma a[2] SUMA = 2
thread 5, suma a[5] SUMA = 5
thread 1, suma a[1] SUMA = 1
thread 3, suma a[3] SUMA = 3
thread 4, suma a[4] SUMA = 4
thread 6, suma a[6] SUMA = 6
Fuera de la region parallel suma = 6
[b4estudiante10@atcgrid ejer3]$ ./firstlastprivate
thread 0, suma a[0] SUMA = 0
thread 6, suma a[6] SUMA = 6
thread 3, suma a[3] SUMA = 3
thread 4, suma a[4] SUMA = 4
thread 1, suma a[1] SUMA = 1
thread 2, suma a[2] SUMA = 2
thread 5, suma a[5] SUMA = 5
Fuera de la region parallel suma = 6
[b4estudiante10@atcgrid ejer3]$ ./firstlastprivate
thread 4, suma a[4] SUMA = 4
thread 2, suma a[2] SUMA = 2
thread 0, suma a[0] SUMA = 0
thread 3, suma a[3] SUMA = 3
thread 6, suma a[6] SUMA = 6
thread 1, suma a[1] SUMA = 1
thread 5, suma a[5] SUMA = 5
Fuera de la region parallel suma = 6
[b4estudiante10@atcgrid ejer3]$
```

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? (añada una captura de pantalla que muestre lo que ocurre)

RESPUESTA: Ocurre lo mismo, pero al final lo solucione, tenia un problema de cache y no se borraba bien, así que lo que ocurre al quitarle `copyprivate(a)` la variable `a` solo se copia en 1 hebra de tal forma que al modificar el vector `b` todos tendrán valor basura por que no guarda el valor en `a` en todas las hebras, excepto en la hebra donde contiene el valor pasado, que lo guarda en una posición del vector. Es decir `copyprivate` permite que se copie el valor de la variable en este caso `a` de una hebra en todas las hebras de forma privada.

CAPTURA CÓDIGO FUENTE: `copyprivate-clauseModificado.c`

```
1 #include <stdio.h>
2
3 #ifdef _OPENMP
4     #include <omp.h>
5 #else
6     #define omp_get_thread_num() 0
7 #endif
8
9 //gcc -O2 -fopenmp -o firstlastprivate firstlastprivate.c
10
11 int main(int argc, char const *argv[])
12 {
13     int i, n=9, b[n];
14
15     for (i = 0; i < n; ++i) b[i]=-1;
16
17     #pragma omp parallel
18     {
19         int a;
20         #pragma omp single //copyprivate(a)
21         {
22             printf("Introduce valor de a: ");
23             scanf("%d",&a);
24             printf("\nSingle ejecutada por la hebra: %d\n", omp_get_thread_num());
25         }
26         #pragma omp for
27             for(i=0;i<n;++i) b[i]=a;
28     }
29
30     printf("\nFuera de la region parallel:\n");
31     for (i = 0; i < n; ++i)
32     {
33         printf("b[%d] = %d\n", i, b[i]);
34     }
35
36     return 0;
37 }
```

CAPTURAS DE PANTALLA:

```
ruben:~/Escritorio/AC/BP2/ejer5$ gcc -O2 -fopenmp -o copyprivate_clause copyprivate_clause.c
ruben:~/Escritorio/AC/BP2/ejer5$ ./copyprivate_clause
Introduce valor de a: 2

Single ejecutada por la hebra: 0

Fuera de la region parallel:
b[0] = 2
b[1] = 2
b[2] = 0
b[3] = 0
b[4] = 0
b[5] = 0
b[6] = 0
b[7] = 0
b[8] = 0
ruben:~/Escritorio/AC/BP2/ejer5$ ./copyprivate_clause
Introduce valor de a: 3

Single ejecutada por la hebra: 1

Fuera de la region parallel:
b[0] = 0
b[1] = 0
b[2] = 3
b[3] = 0
b[4] = 0
b[5] = 0
b[6] = 0
b[7] = 0
b[8] = 0
ruben:~/Escritorio/AC/BP2/ejer5$ ./copyprivate_clause
Introduce valor de a: 6

Single ejecutada por la hebra: 0

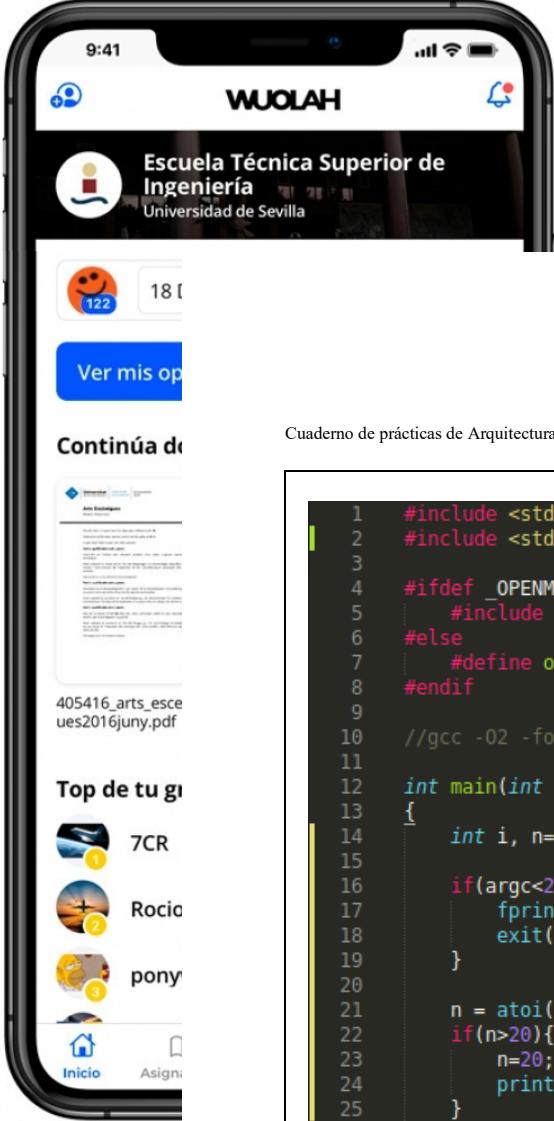
Fuera de la region parallel:
b[0] = 6
b[1] = 6
b[2] = 0
b[3] = 0
b[4] = 0
b[5] = 0
b[6] = 0
b[7] = 0
b[8] = 0
```

6. En el ejemplo reduction-clause.c sustituya `suma=0` por `suma=10`. ¿Qué resultado se imprime ahora? Justifique el resultado (añada capturas de pantalla que muestren lo que ocurre)

RESPUESTA:

Imprime el resultado, es decir la suma de los valores + el valor de inicio, en este caso 10. Por eso en la clausula especifica que la inicialización tiene que ser 0.

CAPTURA CÓDIGO FUENTE: reduction-clauseModificado.c



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #ifdef _OPENMP
5     #include <omp.h>
6 #else
7     #define omp_get_thread_num() 0
8 #endif
9
10 //gcc -O2 -fopenmp -o firstlastprivate firstlastprivate.c
11
12 int main(int argc, char const *argv[])
13 {
14     int i, n=20, a[n], suma=10;
15
16     if(argc<2){
17         fprintf(stderr, "Error falta recibir por paramentro un numero.\n");
18         exit(-1);
19     }
20
21     n = atoi(argv[1]);
22     if(n>20){
23         n=20;
24         printf("\nn = %d\n", n);
25     }
26
27     for (i = 0; i < n; ++i) a[i]=i;
28
29 #pragma omp parallel for reduction(+:suma)
30     for(i=0;i<n;++i) suma+=a[i];
31
32     printf("\nFuera de la region parallel: suma = %d\n", suma);
33
34     return 0;
35 }
```

CAPTURAS DE PANTALLA:

```
[b4estudiante10@atcgrid ejer6]$ gcc -O2 -fopenmp -o reduction_clause reduction_clause.c
[b4estudiante10@atcgrid ejer6]$ nano reduction_clause.c
[b4estudiante10@atcgrid ejer6]$ ./reduction_clause 1
Fuera de la region parallel: suma = 10
[b4estudiante10@atcgrid ejer6]$ ./reduction_clause 2
Fuera de la region parallel: suma = 11
[b4estudiante10@atcgrid ejer6]$ ./reduction_clause 3
Fuera de la region parallel: suma = 13
[b4estudiante10@atcgrid ejer6]$ ./reduction_clause 4
Fuera de la region parallel: suma = 16
[b4estudiante10@atcgrid ejer6]$ ./reduction_clause 5
Fuera de la region parallel: suma = 20
[b4estudiante10@atcgrid ejer6]$ ./reduction_clause 30
n = 20
Fuera de la region parallel: suma = 200
[b4estudiante10@atcgrid ejer6]$ █
```

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 añadir más directivas de trabajo compartido (añada capturas de pantalla que muestren lo que ocurre).

RESPUESTA: He usado `atomic` para sincronizar las hebras, actualizando el valor de la variable automáticamente, garantizando que 1 hebra actualice la variable compartida a la vez, evitando errores.

CAPTURA CÓDIGO FUENTE: reduction-clauseModificado7.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #ifdef _OPENMP
5     #include <omp.h>
6 #else
7     #define omp_get_thread_num() 0
8 #endif
9
10 //gcc -O2 -fopenmp -o firstlastprivate firstlastprivate.c
11
12 int main(int argc, char const *argv[])
13 {
14     int i, n=20, a[n], suma=10;
15
16     if(argc<2){
17         fprintf(stderr, "Error falta recibir por paramentro un numero.\n");
18         exit(-1);
19     }
20
21     n = atoi(argv[1]);
22     if(n>20){
23         n=20;
24         printf("\nn = %d\n", n);
25     }
26
27     for (i = 0; i < n; ++i) a[i]=i;
28
29 #pragma omp parallel for
30     for(i=0;i<n;++i){
31         #pragma omp atomic
32             suma+=a[i];
33     }
34
35     printf("\nFuera de la region parallel: suma = %d\n", suma);
36
37     return 0;
38 }
```

CAPTURAS DE PANTALLA:

```
[b4estudiante10@atcgrid ejer7]$ ./reduction_clause_modificado7 30
n = 20
Fuera de la region parallel: suma = 200
[b4estudiante10@atcgrid ejer7]$ ./reduction_clause_modificado7 10
Fuera de la region parallel: suma = 55
[b4estudiante10@atcgrid ejer7]$ ./reduction_clause_modificado7 5
Fuera de la region parallel: suma = 20
[b4estudiante10@atcgrid ejer7]$ ./reduction_clause_modificado7 3
Fuera de la region parallel: suma = 13
[b4estudiante10@atcgrid ejer7]$ █
```

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):

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

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

CAPTURA CÓDIGO FUENTE: pmv-secuencial.c

Lo he ejecutado en versión dinámica.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #ifdef _OPENMP
5     #include <omp.h>
6 #else
7     #define omp_get_thread_num() 0
8 #endif
9
10 //#define GLOBAL //para vectores y matriz globales
11 #define DYNAMIC //para vectores y matriz dinamicos
12
13 //gcc -O2 -fopenmp -o firstlastprivate firstlastprivate.c
14
15 /*
16     el numero de filas y columnas de la matriz m es n que recibe por parametro
17     inicializar el vector v1 y la matriz antes
18     la suma debe hacerse correcta
19
20     probar con N=8 y N=11
21
22     IMPRIMIR
23         -el tiempo de ejecución del código paralelo que calcula el producto matriz vector
24         -el primer y ultimo componente del resultado
25 */
26
27 #ifdef GLOBAL
28     #define MAX 67108864    //=2^26
29 #endif
30
31
32 int main(int argc, char const *argv[])
33 {
34
35     if(argc<2){
36         fprintf(stderr, "Error falta recibir por paramentro un numero.\n");
37         exit(-1);
38     }
39
40     unsigned int n=atoi(argv[1]);
41
42     #ifdef GLOBAL
43         if(n>MAX){
44             n=MAX;
45             printf("n = %d\n", n);
46         }
47         int m[n][n], v1[n], v2[n];
48     #endif
49
50     #ifdef DYNAMIC
51         int **m, *v1, *v2;
52         v1 = (int*) malloc(n*sizeof(int));// malloc necesita el tamaño en bytes
53         v2 = (int*) malloc(n*sizeof(int));
54         m = (int**) malloc(n*sizeof(int *));
55
56         if ((v1 == NULL) || (v2 == NULL) || (m == NULL)) {
57             printf("No hay suficiente espacio para los vectores o matriz \n");
58             exit(-2);
59         }
60         for (int i = 0; i < n; i++) {
61             for (int e = 0; e < n; e++) {
62                 m[i][e] = 0;
63             }
64         }
65     #endif
66
67     for (int i = 0; i < n; i++) {
68         for (int e = 0; e < n; e++) {
69             m[i][e] = i + e;
70         }
71     }
72
73     for (int i = 0; i < n; i++) {
74         v1[i] = 0;
75     }
76
77     #ifdef DYNAMIC
78         for (int i = 0; i < n; i++) {
79             v1[i] = i;
80         }
81     #endif
82
83     #ifdef _OPENMP
84         #pragma omp parallel for
85         for (int i = 0; i < n; i++) {
86             #pragma omp parallel for
87             for (int e = 0; e < n; e++) {
88                 v2[i] += m[i][e];
89             }
90         }
91     #else
92         for (int i = 0; i < n; i++) {
93             for (int e = 0; e < n; e++) {
94                 v2[i] += m[i][e];
95             }
96         }
97     #endif
98
99     #ifdef DYNAMIC
100        free(v1);
101        free(v2);
102    #endif
103
104    printf("El resultado es:\n");
105    for (int i = 0; i < n; i++) {
106        printf("%d ", v2[i]);
107    }
108
109    return 0;
110 }
111
112
113
114
115
116
117
118

```



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the App Store GET IT ON Google Play

Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática

```
119     if(midle==i)
120         printf("x |%d| = |%d|\n", v1[i],v2[i]);
121     else
122         printf("  |%d|   |%d|\n", v1[i],v2[i]);
123     }
124 }
125
126 printf("V2[%d] = %d\n", 0, v2[0]);
127 printf("V2[%d] = %d\n", n-1, v2[n-1]);
128
129 #ifdef VECTOR_DYNAMIC
130     free(v1); // libera el espacio reservado para v1
131     free(v2); // libera el espacio reservado para v2
132
133     for (int i = 0; i < n; ++i)
134     {
135         free(m[i]); // libera el espacio reservado de las columnas de la matriz
136     }
137
138     free(m); // libera el espacio reservado de las filas de la matriz
139 #endif
140
141     return 0;
142 }
```

CAPTURAS DE PANTALLA:

```
[b4estudiante10@atcgrid ejer8]$ srun ./pmv_secuencial 8
Tiempo (seg.): 0.000000326
|2 2 2 2 2 2 2 2| |3| |48| Nombre de archivo > Tam
|2 2 2 2 2 2 2 2| |3| |48| ejer1
|2 2 2 2 2 2 2 2| |3| |48| ejer2
|2 2 2 2 2 2 2 2| |3| |48| ejer3
|2 2 2 2 2 2 2 2| |3| |48| ejer5
|2 2 2 2 2 2 2 2| |3| |48| ejer6
V2[0] = 48
V2[7] = 48
[b4estudiante10@atcgrid ejer8]$ srun ./pmv_secuencial 11
Tiempo (seg.): 0.000000346
|2 2 2 2 2 2 2 2 2 2| |3| |66| Nombre de archivo > Tam
|2 2 2 2 2 2 2 2 2 2| |3| |66| ejer1
|2 2 2 2 2 2 2 2 2 2| |3| |66| ejer2
|2 2 2 2 2 2 2 2 2 2| |3| |66| ejer3
|2 2 2 2 2 2 2 2 2 2| |3| |66| ejer5
|2 2 2 2 2 2 2 2 2 2| |3| |66| ejer6
|2 2 2 2 2 2 2 2 2 2| |3| |66| ejer7
V2[0] = 66 satisfactorios (16)
V2[10] = 66
[b4estudiante10@atcgrid ejer8]$
```

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 parallelizar 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).

Al final lo he hecho con la versión dinámica.

CAPTURA CÓDIGO FUENTE : pmv-OpenMP-a.c

```

76 //inicialización
77
78 #pragma omp parallel for private(e)
79     for(i=0;i<n;++i){
80         for (e = 0; e < n; ++e){
81             m[i][e]=2;
82         }
83         v1[i]=3;
84         v2[i]=0;
85     }
86
87 //calculo
88
89 double time_start = omp_get_wtime();
90 #pragma omp parallel for
91     for(int i=0;i<n;++i){
92         for (e = 0; e < n; ++e){
93             v2[i]+=m[i][e] * v1[e];
94         }
95     }
96
97 double total_time = (double) (omp_get_wtime() - time_start);
98
99 //resultado
100 printf("Tiempo (seg.): %11.9f\n", total_time);
101
102
103 /*if(n>25){
104     printf("(");
105     for(int i=0;i<6;++i){
106         if(i+1==6)
107             printf("..., %d)\n", v2[n-1]);
108         else if(i<5)
109             printf("%d, ", v2[i]);
110     }
111 }else{
112     for(int i=0;i<n;++i){
113         printf("|");
114         for (int e = 0; e < n; ++e){
115             if(e+1==n)
116                 printf("%d| ", m[i][e]);
117             else
118                 printf("%d ", m[i][e]);
119         }
120         if(midle==i)
121             printf("x |%d| = |%d|\n", v1[i],v2[i]);
122         else
123             printf(" |%d| |%d|\n", v1[i],v2[i]);
124     }
125 }*/
126
127 printf("V2[%d] = %d\n", 0, v2[0]);
128 printf("V2[%d] = %d\n", n-1, v2[n-1]);
129
130 #ifdef VECTOR_DYNAMIC
131     free(v1); // libera el espacio reservado para v1
132     free(v2); // libera el espacio reservado para v2

```

CAPTURA CÓDIGO FUENTE: pmv-OpenMP-b.c

```
77     //inicialización
78
79     #pragma omp parallel private(i)
80     {
81         #pragma omp for private(e)
82         for(i=0;i<n;++i){
83             for (e = 0; e < n; ++e){
84                 m[i][e]=2;
85             }
86             v1[i]=3;
87             v2[i]=0;
88         }
89
90         #pragma omp single
91         {
92             time_start = omp_get_wtime();
93         }
94
95         double suma = 0;
96
97         for(i=0;i<n;++i){
98
99             suma=0;
100
101             #pragma omp for
102             for (e = 0; e < n; ++e){
103                 suma+=(m[i][e] * v1[e]);
104             }
105             #pragma omp atomic
106             v2[i]+=suma;
107         }
108
109         #pragma omp single
110         {
111             total_time = (double) (omp_get_wtime() - time_start);
112         }
113     }
114
115
116     //resultado
117
118     printf("Tiempo (seg.): %11.9f\n", total_time);
119
120
121     if(n>25){
122         printf("(");
123         for(int i=0;i<6;++i){
124             if(i+1==6)
125                 printf("..., %d)\n", v2[n-1]);
126             else if(i<5)
127                 printf("%d, ", v2[i]);
128         }
129     }else{
130         for(int i=0;i<n;++i){
131             printf("|");
132             for (int e = 0; e < n; ++e){
133                 if(e+1==n)
134                     printf("%d| ", m[i][e]);
135             }
136         }
137     }
138 }
```



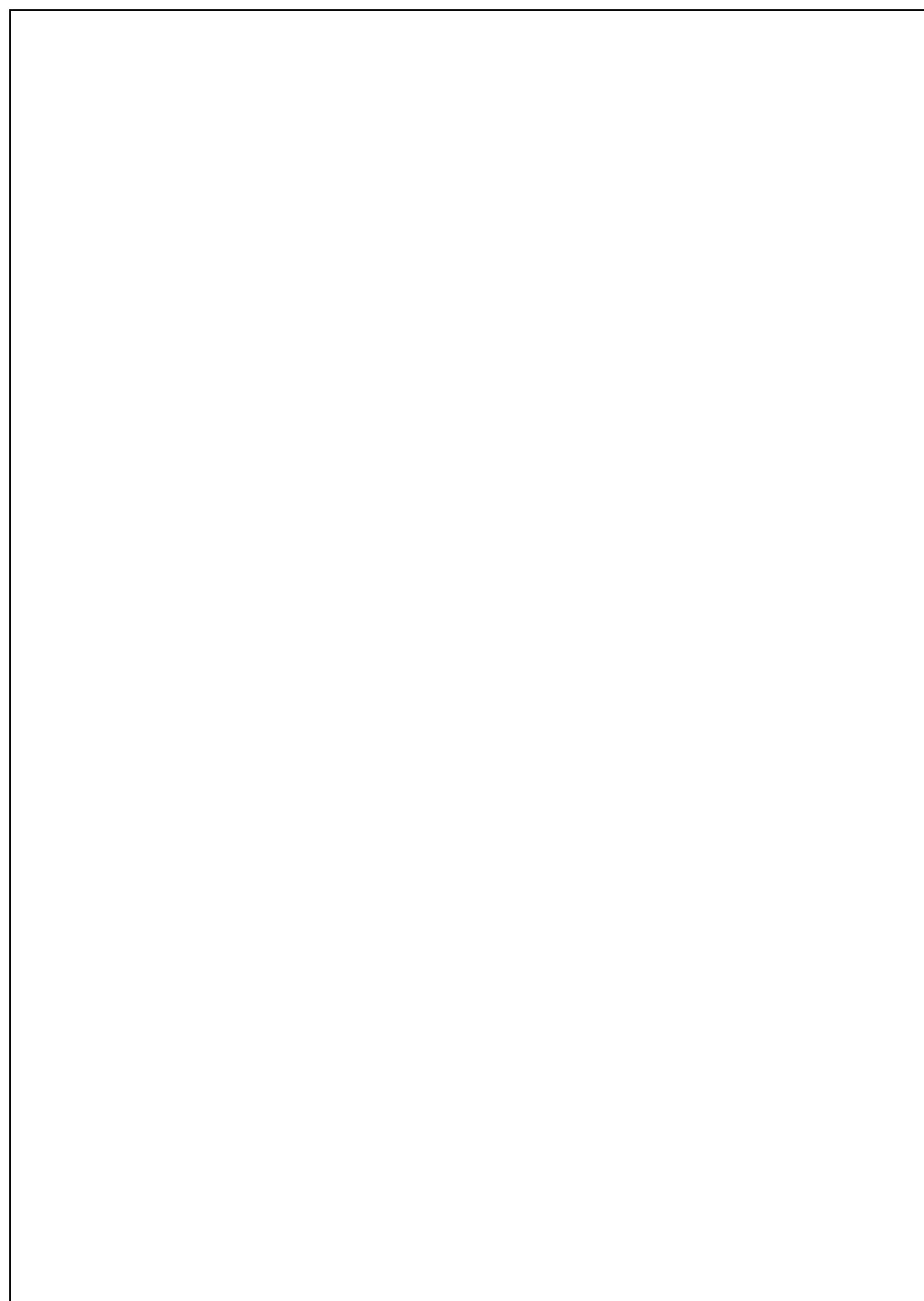
Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática



```
135         else
136             printf("%d ", m[i][e]);
137     }
138     if(midle==i)
139         printf("x |%d| = |%d|\n", v1[i],v2[i]);
140     else
141         printf(" |%d| - |%d|\n", v1[i],v2[i]);
142     }
143 }
144
145 printf("V2[%d] = %d\n", 0, v2[0]);
146 printf("V2[%d] = %d\n", n-1, v2[n-1]);
147
148 #ifdef VECTOR_DYNAMIC
149     free(v1); // libera el espacio reservado para v1
150     free(v2); // libera el espacio reservado para v2
151
152     for (int i = 0; i < n; ++i)
153     {
154         free(m[i]); // libera el espacio reservado de las columnas de la matriz
155     }
156
157     free(m); // libera el espacio reservado de las filas de la matriz
158 #endif
159
160     return 0;
161 }
```

RESPUESTA:

CAPTURAS DE PANTALLA:

```
[b4estudiante10@atcgrid ejer9]$ ./pmv_OpenMP_a 8
Tiempo (seg.): 0.000005173
|2 2 2 2 2 2 2 2| |3| |48|
|2 2 2 2 2 2 2 2| |3| |48|
|2 2 2 2 2 2 2 2| |3| |48|
|2 2 2 2 2 2 2 2| |3| |48|
|2 2 2 2 2 2 2 2| |3| |48|
|2 2 2 2 2 2 2 2| x |3| = |48|
|2 2 2 2 2 2 2 2| |3| |48|
|2 2 2 2 2 2 2 2| |3| |48|
|2 2 2 2 2 2 2 2| |3| |48|
V2[0] = 48
V2[7] = 48
[b4estudiante10@atcgrid ejer9]$ ./pmv_OpenMP_a 11
Tiempo (seg.): 0.000005748
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| x |3| = |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
V2[0] = 66
V2[10] = 66
[b4estudiante10@atcgrid ejer9]$ ■
[b4estudiante10@atcgrid ejer9]$ ./pmv_OpenMP_b 8
Tiempo (seg.): 0.000033712
|2 2 2 2 2 2 2 2| |3| |48|
|2 2 2 2 2 2 2 2| |3| |48|
|2 2 2 2 2 2 2 2| |3| |48|
|2 2 2 2 2 2 2 2| |3| |48|
|2 2 2 2 2 2 2 2| x |3| = |48|
|2 2 2 2 2 2 2 2| |3| |48|
|2 2 2 2 2 2 2 2| |3| |48|
|2 2 2 2 2 2 2 2| |3| |48|
V2[0] = 48
V2[7] = 48
[b4estudiante10@atcgrid ejer9]$ ./pmv_OpenMP_b 11
Tiempo (seg.): 0.000046015
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| x |3| = |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
|2 2 2 2 2 2 2 2 2| |3| |66|
V2[0] = 66
V2[10] = 66
[b4estudiante10@atcgrid ejer9]$ ■
```

Los únicos errores que dio era que en el b los resultados no coincidian y tuve que poner la sección de la multiplicación de matrices atomic.

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-OpenMP-reduction.c

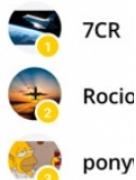


[Ver mis op](#)

Continúa d



Top de tu gu



Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática

```

75     //inicialización
76
77     int e, i, suma=0;
78     double time_start, total_time;
79
80     #pragma omp parallel private(i)
81     {
82         #pragma omp for private(e)
83         for(i=0;i<n;++i){
84             for (e = 0; e < n; ++e){
85                 m[i][e]=2;
86             }
87             v1[i]=3;
88             v2[i]=0;
89         }
90
91         //calcula
92
93         #pragma omp single
94         {
95             time_start = omp_get_wtime();
96         }
97
98         for(i=0;i<n;++i){
99             #pragma omp for reduction(+:suma)
100                for (e = 0; e < n; ++e){
101                    suma += m[i][e] * v1[e];
102                }
103                #pragma omp single
104                {
105                    v2[i] = suma;
106                    suma = 0;
107                }
108            }
109
110            #pragma omp single
111            {
112                total_time = (double) (omp_get_wtime() - time_start);
113            }
114        }
115
116        //resultado
117
118        printf("Tiempo (seg.): %11.9f\n", total_time);
119
120
121        if(n>25){
122            printf("(");
123            for(int i=0;i<6;++i){
124                if(i+1==6)
125                    printf("..., %d)\n", v2[n-1]);
126                else if(i<5)
127                    printf("%d, ", v2[i]);
128            }
129        }else{
130            for(int i=0;i<n;++i){
131                printf("|");
132                for (int e = 0; e < n; ++e){
133                    if(e+1==n)

```

Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

```
134         printf("%d| ", m[i][e]);
135     else
136         printf("%d ", m[i][e]);
137     }
138     if(midle==i)
139         printf("x |%d| = |%d|\n", v1[i],v2[i]);
140     else
141         printf(" |%d| |%d|\n", v1[i],v2[i]);
142     }
143 }
144
145 printf("V2[%d] = %d\n", 0, v2[0]);
146 printf("V2[%d] = %d\n", n-1, v2[n-1]);
147
148 #ifdef VECTOR_DYNAMIC
149     free(v1); // libera el espacio reservado para v1
150     free(v2); // libera el espacio reservado para v2
151
152     for (int i = 0; i < n; ++i)
153     {
154         free(m[i]); // libera el espacio reservado de las columnas de la matriz
155     }
156
157     free(m); // libera el espacio reservado de las filas de la matriz
158 #endif
159
160     return 0;
161 }
```

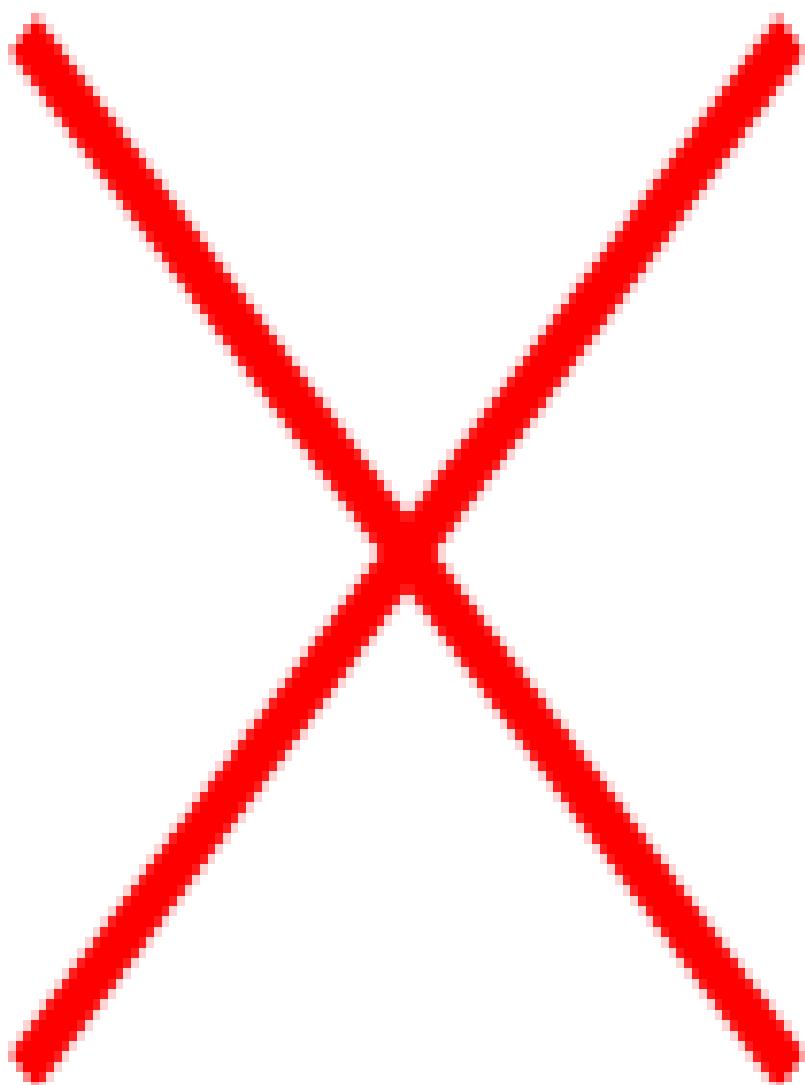
RESPUESTA: El único problema que he tenido ha sido como poner el **reduction**, probando sin usar variables auxiliares, pero al final lo que he hecho con una variable auxiliar que acceda al **reduction** y guarde la suma y luego usando **single** guardo de forma segura el valor de suma en **V2**.

CAPTURAS DE PANTALLA:

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 (con tiempos y ganancia) Y GRÁFICA (con ganancia) (para 1-4 threads PC local, y para 1-12 threads en atcgrid, tamaños-N: un N entre 20000 y 100000, y otro entre 5000 y 20000):





Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play



122

18

Ver mis op

Continúa d

Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática



405416_arts_esce_ues2016juniy.pdf

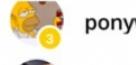
Top de tu gr



7CR



Rocio



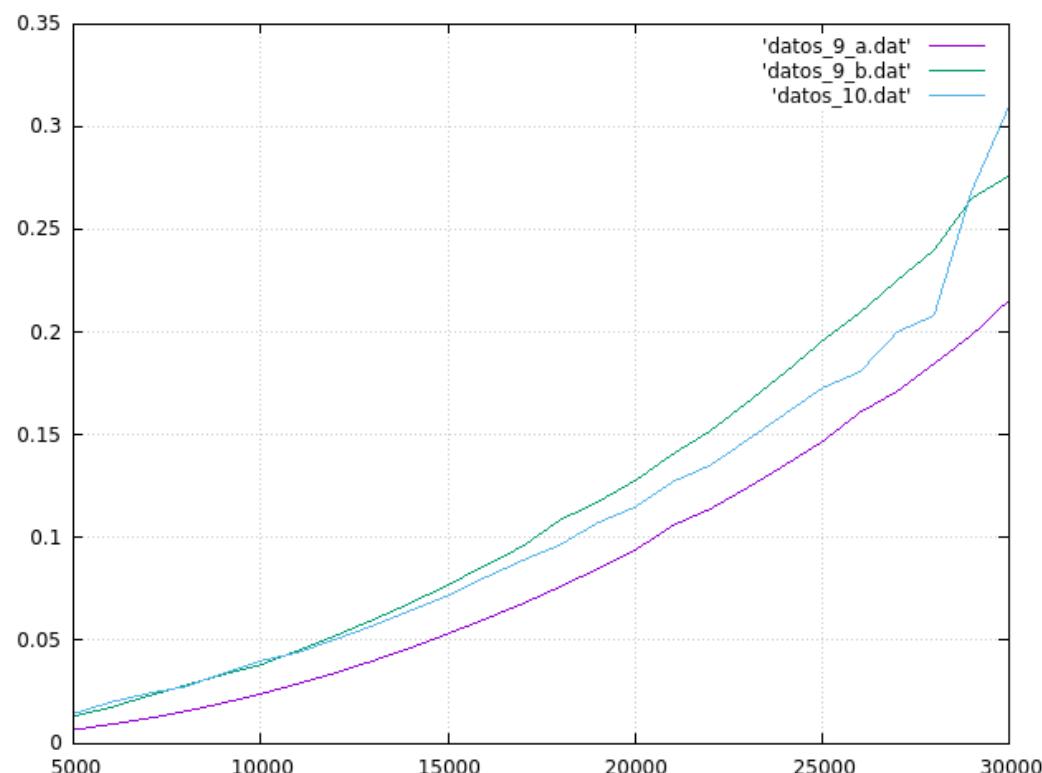
pony



Inicio

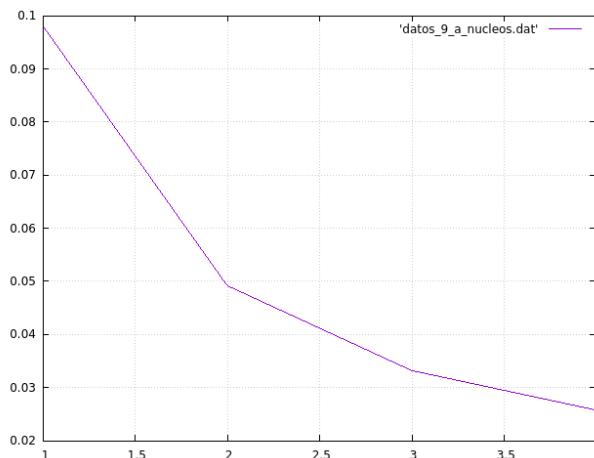


Asigni

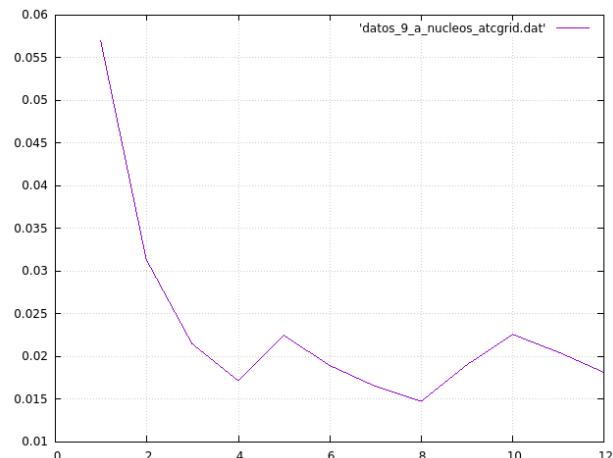


Ejercicio 9 a n = 10000

PC

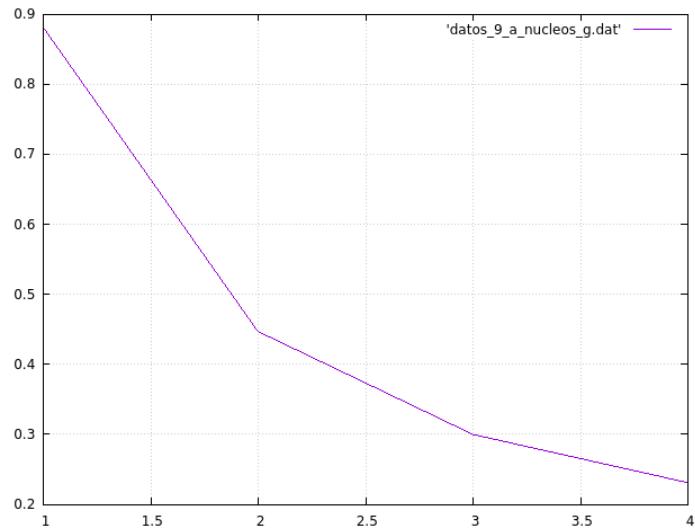


ATCGRID

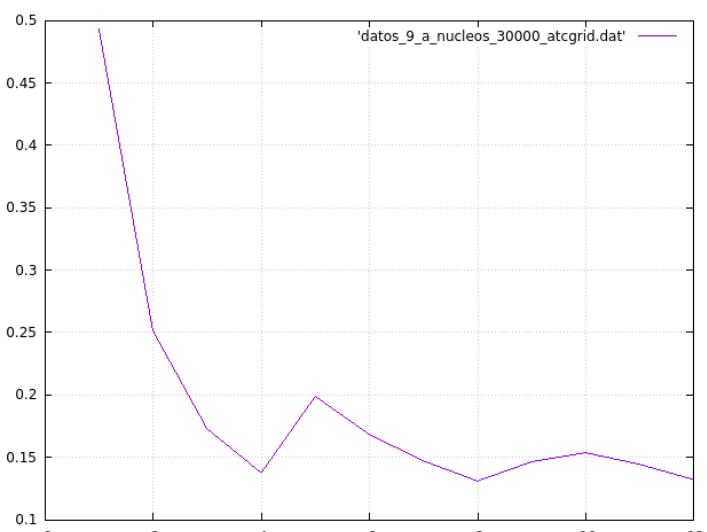


Ejercicio 9 a n = 30000

PC



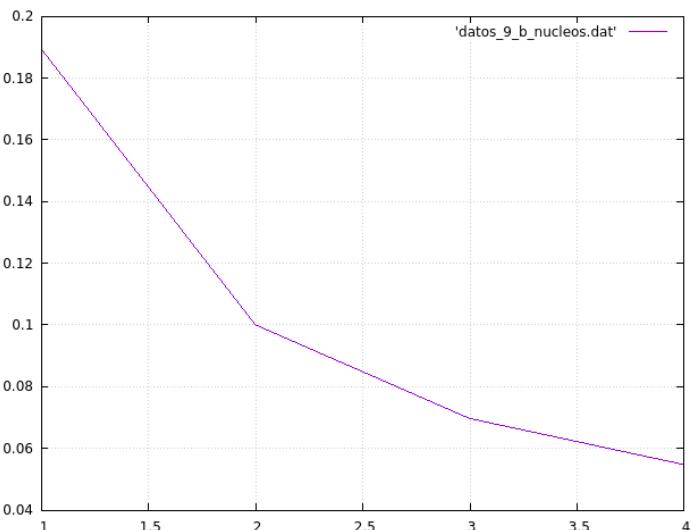
ATCGRID



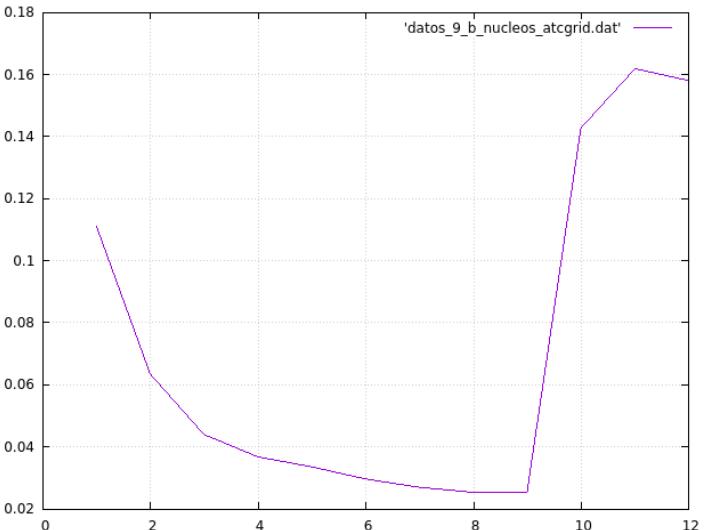
Como podemos ver en las 2 graficas de atcgrid al llegar mas de 4 hebras se inestabiliza teniendo peor y mejor rendimiento.

Ejercicio 9 b n = 10000

PC

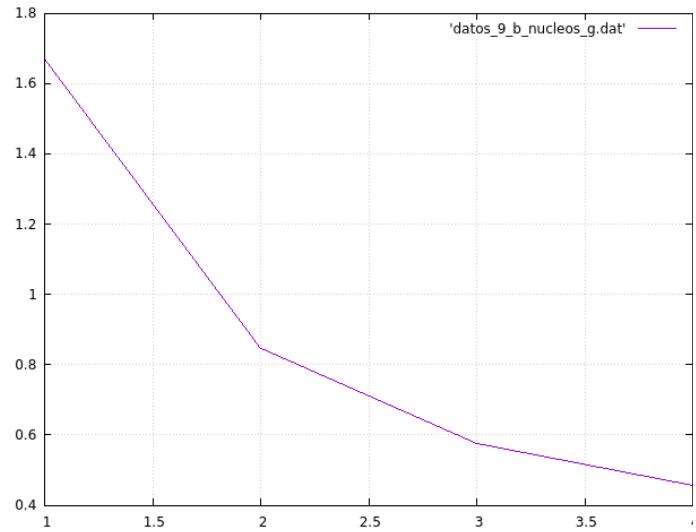


ATCGRID

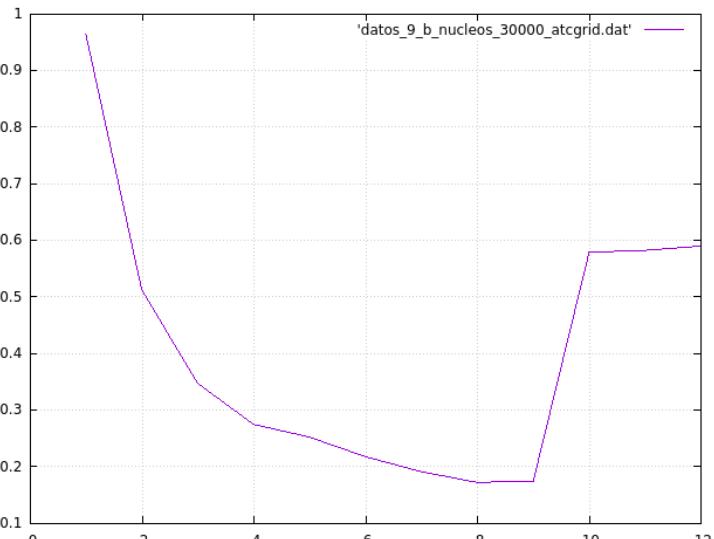


Ejercicio 9 b n = 30000

PC

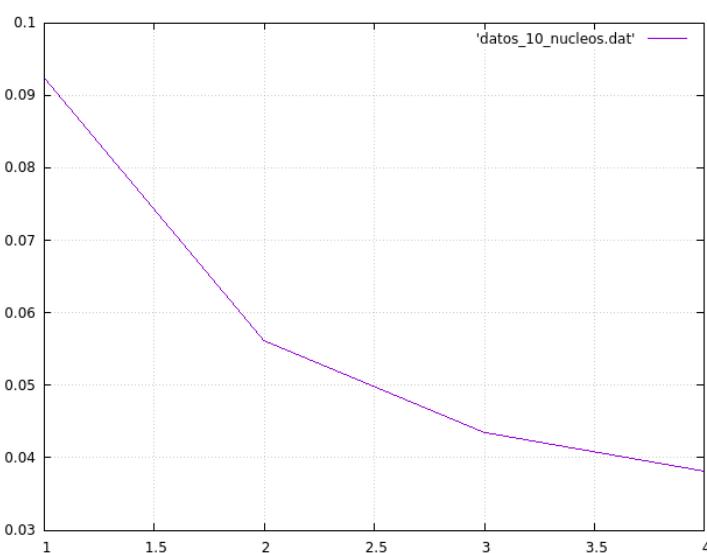


ATCGRID

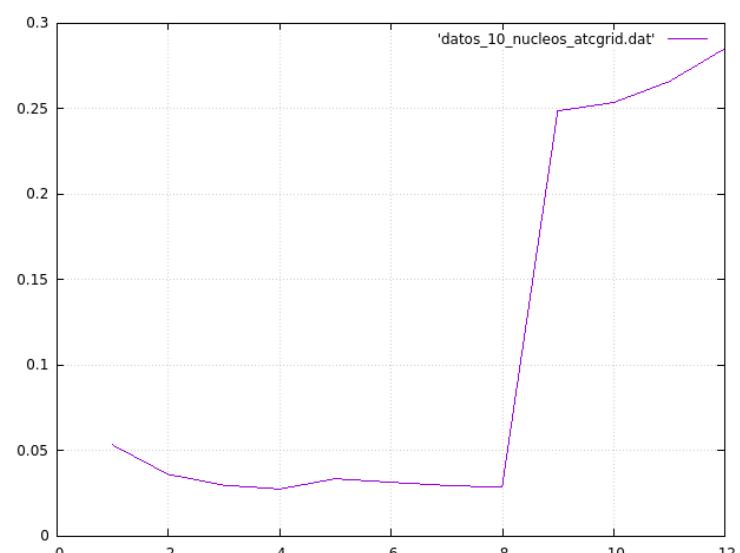


Ejercicio 10 n = 10000

PC



ATCGRID



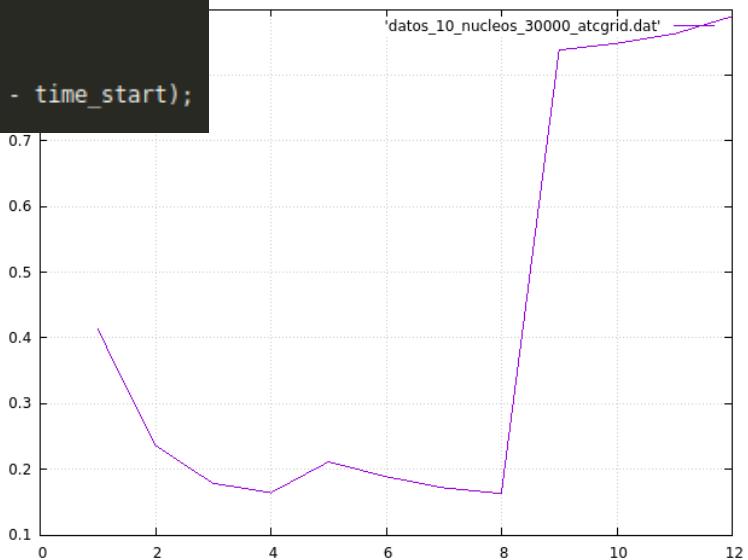
```
//calculo  
  
double time_start = omp_get_wtime();  
#pragma omp parallel for  
for(int i=0;i<n;++i){  
    for (e = 0; e < n; ++e){  
        v2[i]+=m[i][e] * v1[e];  
    }  
}  
  
double total_time = (double) (omp_get_wtime() - time_start);
```



Ejercicio 10 n = 30000

PC

ATCGRID



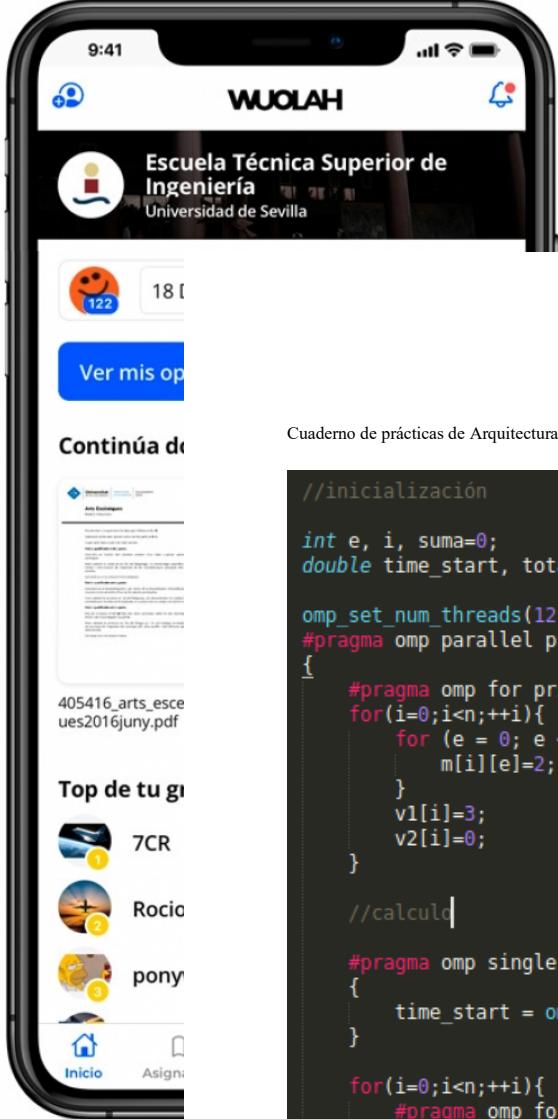
En cambio en estos graficos cuando llega a mas de 4 hebras empeora, pero lo mas curioso que cuando tiene mas de 8 hebras empeora bastante, consiquiendo un peor rendimiento.

COMENTARIOS SOBRE LOS RESULTADOS:

Viendo los resultados el **ejercicio 9 a** es el mas rapido que los otros, entre los valores 5000 y 30000 con 4 hebras, con una diferencia de medio milisegundo, por que en el **ejercicio 9 a** paraleliza los 2 bucles aprovechando más tiempo, en cambio los otros solo paralelizan 1 bucle en el calculo y el otro no, desaprovechando tiempo.

Ejercicio 9 a

Ejercicio 9 b



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the App Store

GET IT ON Google Play

Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática

```
//inicialización

int e, i, suma=0;
double time_start, total_time;

omp_set_num_threads(12);
#pragma omp parallel private(i)
{
    #pragma omp for private(e)
    for(i=0;i<n;++i){
        for (e = 0; e < n; ++e){
            m[i][e]=2;
        }
        v1[i]=3;
        v2[i]=0;
    }

    //calcula

    #pragma omp single
    {
        time_start = omp_get_wtime();
    }

    for(i=0;i<n;++i){
        #pragma omp for reduction(+:suma)
        for (e = 0; e < n; ++e){
            suma += m[i][e] * v1[e];
        }
        #pragma omp single
        {
            v2[i] = suma;
            suma = 0;
        }
    }

    #pragma omp single
    {
        total_time = (double) (omp_get_wtime() - time_start);
    }
}
```

Ejercicio 10