

BP3.pdf



PruebaAlien



Arquitectura de Computadores



2º Grado en Ingeniería Informática



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

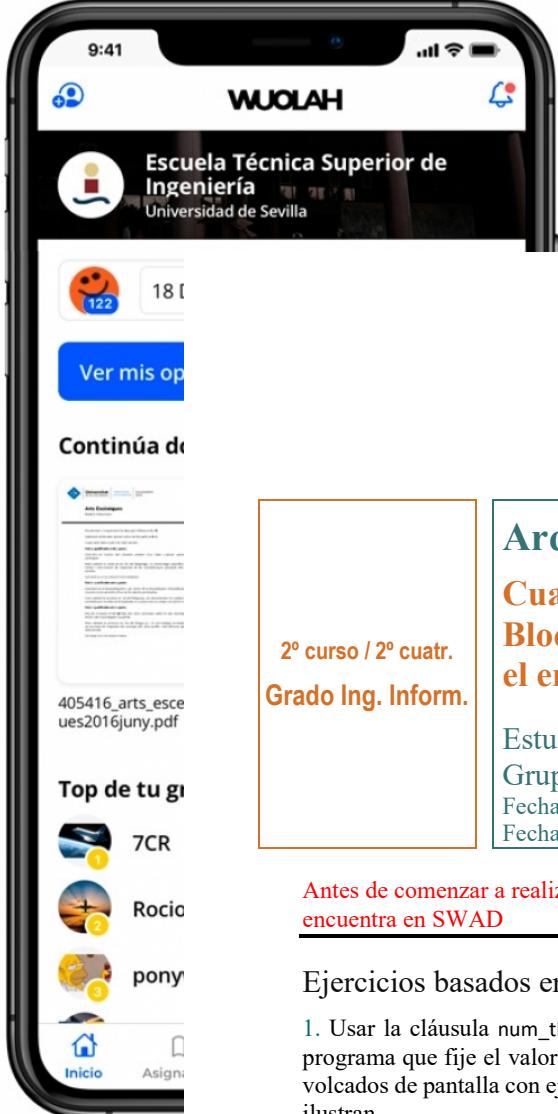


Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.





**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 3. Programación paralela III: Interacción con el entorno en OpenMP

Estudiante (nombre y apellidos):

Grupo de prácticas:

Fecha de entrega:

Fecha evaluación en clase:

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

Ejercicios basados en los ejemplos del seminario práctico

1. Usar la cláusula num_threads(x) en el ejemplo del seminario if_clause.c, y añadir un parámetro de entrada al programa que fije el valor x que se va a usar en la cláusula. Incorporar en el cuaderno de trabajo de esta práctica volcados de pantalla con ejemplos de ejecución que ilustren la funcionalidad de esta cláusula y explicar por qué lo ilustran.

CAPTURA CÓDIGO FUENTE: if-clauseModificado.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <omp.h>
4
5 //gcc -O2 -fopenmp -o single single.c
6
7 int main(int argc, char const *argv[])
8 {
9     int i, n=20, tid, x;
10    int a[n], suma=0, sumalocal;
11
12    if(argc<3){
13        fprintf(stderr, "Error sintaxis: ./execute <number_size> <number_threads>\n");
14        exit(-1);
15    }
16
17    n = atoi(argv[1]);
18    x = atoi(argv[2]);
19
20    if(n>20)n=20;
21
22    for (i = 0; i < n; ++i){
23        a[i] = i;
24    }
25
26    #pragma omp parallel if(n>4) num_threads(x) default(none) private(sumalocal,tid) shared(a, suma, n)
27 {
28        sumalocal = 0;
29        tid = omp_get_thread_num();
30
31        #pragma omp for private(i) schedule(static) nowait
32        for (i = 0; i < n; ++i)
33        {
34            sumalocal += a[i];
35            printf("thread: %d suma de a[%d] = %d, sumalocal = %d\n", tid, i, a[i], sumalocal);
36        }
37
38        #pragma omp atomic
39        suma += sumalocal;
40        #pragma omp barrier
41        #pragma omp master
42        printf("thread master = %d imprime suma = %d\n", tid, suma);
43    }
44    return 0;
45 }
```

CAPTURAS DE PANTALLA:

```

ruben:~/Escritorio/AC/BP3/ejer1$ ./if-clause 5 4
thread: 2 suma de a[3] = 3, sumalocal = 3
thread: 0 suma de a[0] = 0, sumalocal = 0
thread: 0 suma de a[1] = 1, sumalocal = 1
thread: 3 suma de a[4] = 4, sumalocal = 4
thread: 1 suma de a[2] = 2, sumalocal = 2
thread master = 0 imprime suma = 10
ruben:~/Escritorio/AC/BP3/ejer1$ ./if-clause 5 2
thread: 0 suma de a[0] = 0, sumalocal = 0
thread: 0 suma de a[1] = 1, sumalocal = 1
thread: 0 suma de a[2] = 2, sumalocal = 3
thread: 1 suma de a[3] = 3, sumalocal = 3
thread: 1 suma de a[4] = 4, sumalocal = 7
thread master = 0 imprime suma = 10
ruben:~/Escritorio/AC/BP3/ejer1$ ./if-clause 3 4
thread: 0 suma de a[0] = 0, sumalocal = 0
thread: 0 suma de a[1] = 1, sumalocal = 1
thread: 0 suma de a[2] = 2, sumalocal = 3
thread master = 0 imprime suma = 3
ruben:~/Escritorio/AC/BP3/ejer1$ 
```

RESPUESTA:

Básicamente lo que hace es indicarle el tamaño del vector y el numero de hebras que van a trabajar en paralelo por parámetro y calcula la suma de forma paralela con el numero de hebras **num_threads(x)** que van a trabajar de forma paralela, repartiendo el trabajo entre las hebras asignadas.

Pero si yo le indico que el tamaño es < 4 y le indico el numero de hebras, hace la suma pero con una unica hebra, por lo tanto no se hace en paralelo, haciendo la suma la misma **hebra 0**, por la condición **if** puesta.

- 2. (a)** Rellenar la Tabla 1 (se debe poner en la tabla el id del *thread* que ejecuta cada iteración) ejecutando los ejemplos del seminario *schedule-clause.c*, *scheduled-clause.c* y *scheduleg-clause.c* con dos *threads* (0,1) y unas entradas de:

- iteraciones: 16 (0,...15)
- chunck= 1, 2 y 4

Tabla 1 . Tabla schedule. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

Iteración	schedule-clause.c			schedule-clause.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0
5	1	0	1	0	0	0	0	0	0
6	0	1	1	0	0	0	0	0	0
7	1	1	1	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0
10	0	1	0	0	0	0	0	0	0
11	1	1	0	0	0	0	0	0	0
12	0	0	1	0	0	1	1	1	1
13	1	0	1	0	0	1	1	1	1
14	0	1	1	0	1	1	1	1	1
15	1	1	1	1	1	1	1	1	1

- (b)** Rellenar otra tabla como la de la figura pero esta vez usando cuatro *threads* (0,1,2,3).

Tabla 2 .

Tabla schedule. En la segunda fila, 1, 2 4 representan el tamaño del chunk (consulte seminario)

Iteración	schedule-clause.c			schedule-clause.d.c			schedule-clauseg.c		
	1	2	4	1	2	4	1	2	4
0	0	1	2	0	1	0	2	0	0
1	1	1	2	0	1	0	2	0	0
2	3	0	2	0	1	0	2	0	0
3	2	0	2	0	1	0	2	0	0
4	0	2	1	0	1	2	2	0	3
5	1	2	1	0	1	2	2	0	3
6	3	3	1	0	1	2	2	0	3
7	2	3	1	0	1	2	2	0	3
8	0	1	0	0	1	3	3	3	2
9	1	1	0	0	1	3	3	3	2
10	3	0	0	0	0	3	3	3	2
11	2	0	0	0	0	3	0	1	2
12	0	2	3	0	3	1	0	1	1
13	1	2	3	3	3	1	0	1	1
14	3	3	3	2	2	1	1	2	1
15	2	3	3	1	2	1	1	2	1

Escriba en el cuaderno de prácticas las diferencias en el comportamiento de `schedule()` con `static`, `dynamic` y `guided`.

RESPUESTA:

STATIC: Divide las iteraciones en unidades **chunk**, asignándole las unidades en **round-robin** y se le asigna un único **chunk** a cada **thread**. Distribuyendo las iteraciones entre las hebras disponibles, realizando su iteración acorde a su identificador.

DYNAMIC: En este caso no se puede saber en principio las iteraciones que realiza cada hebra, pero la distribución se realiza en tiempo de ejecución. Entonces, sabiendo las iteraciones chunk sabremos su eficiencia **O(n/chunk)**, siendo n el numero de iteraciones. De esta forma los threads mas rápidos ejecutarán mas iteraciones que los demás, pero con un mínimo chunk iteraciones para cada hebra. Con esto indica que no sigue un orden de identificación, sino que se le asigna la iteración a la primera hebra libre. Con dynamic hay que tener cuidado, puesto que hay sobrecarga adicional.

GUIDED: Este realiza la distribución en tiempo de ejecución, utilizando el valor de chunk como tamaño mínimo del bloque, comenzando con un bloque grande y reduciendo el tamaño siempre que no sea igual que el valor de chunk, llegando a no ser mas pequeño que chunk. Este también tiene sobrecarga adicional, pero menor que **dynamic**. Otra cosa que tiene en común con **dynamic** es que las hebras más rápidas realizan más iteraciones que el resto.



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the App Store

GET IT ON Google Play



122

Ver mis op

Continúa d...

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



405416_arts_esce
ues2016juniy.pdf

Top de tu gu...



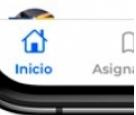
7CR



Rocio



pony



Inicio

Asigni...

3. Añadir al programa scheduled-clause.c lo necesario para que imprima el valor de las variables de control dyn-var, nthreads-var, thread-limit-var y run-sched-var dentro (debe imprimir sólo un thread) y fuera de la región paralela. Realizar varias ejecuciones usando variables de entorno para modificar estas variables de control antes de la ejecución. Incorporar en su cuaderno de prácticas volcados de pantalla de estas ejecuciones. ¿Se imprimen valores distintos dentro y fuera de la región paralela?

CAPTURA CÓDIGO FUENTE: scheduled-clauseModificado.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <stdbool.h>
5
6 #ifdef _OPENMP
7     #include <omp.h>
8 #else
9     #define omp_get_thread_num() 0
10 #endif
11
12 //gcc -O2 -fopenmp -o single single.c
13
14 int main(int argc, char const *argv[])
15 {
16     int i, n=200, chunk, a[n], suma=0;
17     omp_sched_t tipo;
18     int modif;
19     char type[10];
20     bool first = true;
21
22     if(argc<3){
23         fprintf(stderr, "Error sintaxis: ./execute <size> <number_chunk>\n");
24         exit(-1);
25     }
26
27     chunk = atoi(argv[2]);
28
29     n = atoi(argv[1]);
30
31     if(n>200) n = 200;
32
33     for (i = 0; i < n; ++i) a[i] = i;
34
35 #pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
36     for (i = 0; i < n; ++i)
37     {
38         #pragma single
39         {
40             if(first){
41                 first = false;
42                 omp_get_schedule(&tipo, &modif);
43                 printf("\n-----\n");
44                 printf("dentro del \\"parallel for \\"\n");
45                 printf("dyn-var: %d\nnthreads-var: %d\nthread-limit-var: %d\nrun-sched-var: %d\n",
46                         omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), modif);
47                 switch(tipo){
48                     case omp_sched_static:
49                         strcpy(type, "ESTATICO");
50                         break;
51                     case omp_sched_dynamic:
52                         strcpy(type, "DINAMICO");
53                         break;
54                     case omp_sched_guided:
55                         strcpy(type, "GUIDED");
56                         break;
57                     case omp_sched_auto:
58                         strcpy(type, "AUTO");
59                         break;
59                 }
59             }
59         }
59     }
59 }
```

```
60         }
61         printf("tipo: %s\n", type);
62         printf("-----\n\n");
63     }
64 }
65
66 suma += a[i];
67 printf("thread: %d suma de a[%d], suma = %d\n", omp_get_thread_num(), i, suma);
68 }
69
70 omp_get_schedule(&tipo, &modif);
71
72 printf("\nfuera del \"parallel for \" imprime suma = %d\n", suma);
73 printf("dyn-var: %d\nthreads-var: %d\nthread-limit-var: %d\nrun-sched-var: %d\n",
74       omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(), modif);
75
76 if(tipo == omp_sched_static)
77     strcpy(type, "ESTATICO");
78 else if(tipo == omp_sched_dynamic)
79     strcpy(type, "DINAMICO");
80 else if (tipo == omp_sched_guided)
81     strcpy(type, "GUIDED");
82 printf("tipo: %s\n", type);
83 return 0;
84 }
85 }
```

CAPTURAS DE PANTALLA:

```
ruben:~/Escritorio/AC/BP3/ejer3$ export OMP_DYNAMIC=TRUE          4
ruben:~/Escritorio/AC/BP3/ejer3$ ./schedule-clauseModificado 6 3
thread: 1 suma de a[3], suma = 3
thread: 1 suma de a[4], suma = 7
thread: 1 suma de a[5], suma = 12

-----
dentro del "parallel for "
dyn-var: 1
nthreads-var: 8
thread-limit-var: 2147483647
run-sched-var: 1
tipo: DINAMICO
-----

thread: 3 suma de a[0], suma = 0
thread: 3 suma de a[1], suma = 1
thread: 3 suma de a[2], suma = 3

fuera del "parallel for " imprime suma = 12
dyn-var: 1
nthreads-var: 8
thread-limit-var: 2147483647
run-sched-var: 1
tipo: DINAMICO
ruben:~/Escritorio/AC/BP3/ejer3$ █
run-sched-var: 1
tipo: DINAMICO
ruben:~/Escritorio/AC/BP3/ejer3$ █
```

Depto. Arquitectura y Tecnología de Computadores

```
ruben:~/Escritorio/AC/BP3/ejer3$ export OMP_THREAD_LIMIT=2
ruben:~/Escritorio/AC/BP3/ejer3$ ./schedule-clauseModificado 5 2
thread: 1 suma de a[2], suma = 2
thread: 1 suma de a[3], suma = 5

-----
dentro del "parallel for "
thread: 1 suma de a[4], suma = 9
dyn-var: 0
nthreads-var: 3
thread-limit-var: 2
run-sched-var: 1
tipo: DINAMICO
-----

thread: 0 suma de a[0], suma = 0
thread: 0 suma de a[1], suma = 1

fuera del "parallel for " imprime suma = 9
dyn-var: 0
nthreads-var: 3
thread-limit-var: 2
run-sched-var: 1
tipo: DINAMICO
ruben:~/Escritorio/AC/BP3/ejer3$ 
thread-limit-var: 2147483647
run-sched-var: 1
tipo: DINAMICO
ruben:~/Escritorio/AC/BP3/ejer3$ 
```



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

```
ruben:~/Escritorio/AC/BP3/ejer3$ export OMP_SCHEDULE="static, 4"
ruben:~/Escritorio/AC/BP3/ejer3$ ./schedule-clauseModificado 5 2

-----
dentro del "parallel for "
dyn-var: 0
nthreads-var: 3
thread-limit-var: 2
run-sched-var: 4
tipo: ESTATICO

-----
thread: 0 suma de a[0], suma = 0
thread: 0 suma de a[1], suma = 1
thread: 0 suma de a[4], suma = 5
thread: 1 suma de a[2], suma = 2
thread: 1 suma de a[3], suma = 5

fuera del "parallel for " imprime suma = 5
dyn-var: 0
nthreads-var: 3
thread-limit-var: 2
run-sched-var: 4
tipo: ESTATICO
ruben:~/Escritorio/AC/BP3/ejer3$
```

RESPUESTA:

Dentro y fuera de la región paralela imprime los mismos valores (dinámico, etc), que se han establecido en el parallel for. Como se puede ver en las distintas ejecuciones imprime lo mismo

4. Usar en el ejemplo anterior las funciones `omp_get_num_threads()`, `omp_get_num_procs()` y `omp_in_parallel()` dentro y fuera de la región paralela. Imprimir los valores que obtienen estas funciones dentro (lo debe imprimir sólo uno de los threads) y fuera de la región paralela. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos. Indicar en qué funciones se obtienen valores distintos dentro y fuera de la región paralela.

CAPTURA CÓDIGO FUENTE: scheduled-clauseModificado4.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <stdbool.h>
5
6 #ifdef _OPENMP
7     #include <omp.h>
8 #else
9     #define omp_get_thread_num() 0
10#endif
11
12 //gcc -O2 -fopenmp -o single single.c
13
14 int main(int argc, char const *argv[])
15 {
16     int i, n=200, chunk, a[n], suma=0;
17     bool first = true;
18
19     if(argc<3){
20         fprintf(stderr, "Error sintaxis: ./execute <size> <number_chunk>\n");
21         exit(-1);
22     }
23
24     chunk = atoi(argv[2]);
25
26     n = atoi(argv[1]);
27
28     if(n>200) n = 200;
29
30     for (i = 0; i < n; ++i) a[i] = i;
31
32 #pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
33     for (i = 0; i < n; ++i)
34     {
35         #pragma single
36         {
37             if(first){
38                 first = false;
39                 printf("\n-----\n");
40                 printf("dentro del \"parallel for \"\n");
41                 printf("num_threads: %d\nnum_procs: %d\nin_parallel: %d\n",
42                         omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
43             }
44         }
45
46         suma += a[i];
47         printf("thread: %d suma de a[%d], suma = %d\n", omp_get_thread_num(), i, suma);
48     }
49
50     printf("\nfuera del \"parallel for \" imprime suma = %d\n", suma);
51     printf("num_threads: %d\nnum_procs: %d\nin_parallel: %d\n",
52             omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
53     return 0;
54 }
```

CAPTURAS DE PANTALLA:

```
ruben:~/Escritorio/AC/BP3/ejer4$ ./schedule-clauseModificado4 10 4
thread: 7 suma de a[4], suma = 4
thread: 2 suma de a[8], suma = 8
thread: 2 suma de a[9], suma = 17
thread: 7 suma de a[5], suma = 9
thread: 7 suma de a[6], suma = 15
thread: 7 suma de a[7], suma = 22

-----
dentro del "parallel for "
num_threads: 8
um_procs: 8
in_parallel: 1
thread: 0 suma de a[0], suma = 0
thread: 0 suma de a[1], suma = 1
thread: 0 suma de a[2], suma = 3
thread: 0 suma de a[3], suma = 6

fuera del "parallel for " imprime suma = 17
num_threads: 1
um_procs: 8
in_parallel: 0
ruben:~/Escritorio/AC/BP3/ejer4$
```

RESPUESTA:

Como se puede ver los valores son distintos en el numero de hebras y si esta en una región paralela o no, dentro de la región paralela se ve que usa 8 hebras, indica con 1 que esta en una región paralela y en ambos tanto dentro como fuera de la región el numero de procesos en ambos es 8. Fuera de la región paralela el numero de hebras cambia a 1 e indica un 0 diciendo que no esta en una región en paralelo.

5. Añadir al programa scheduled-clause.c lo necesario para modificar las variables de control dyn-var, nthreads-var y run-sched-var y para poder imprimir el valor de estas variables antes y después de dicha modificación. Incorporar en su cuaderno de prácticas volcados de pantalla con los resultados de ejecución obtenidos.

CAPTURA CÓDIGO FUENTE: scheduled-clauseModificado5.c

```
35 #pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
36     for (i = 0; i < n; ++i)
37     {
38         #pragma single
39         {
40             if(first){
41                 first = false;
42                 omp_get_schedule(&tipo, &modif);
43                 printf("\n-----\n");
44                 printf("dentro del \\"parallel for ANTES DE MODIFICALO\\"\\n");
45                 printf("dyn-var: %d\\nnthreads-var: %d\\nrun-sched-var: %d\\n",
46                         omp_get_dynamic(), omp_get_max_threads(), modif);
47                 switch(tipo){
48                     case omp_sched_static:
49                         strcpy(type, "ESTATICO");
50                         break;
51                     case omp_sched_dynamic:
52                         strcpy(type, "DINAMICO");
53                         break;
54                     case omp_sched_guided:
55                         strcpy(type, "GUIDED");
56                         break;
57                     case omp_sched_auto:
58                         strcpy(type, "AUTO");
59                         break;
60                 }
61                 printf("tipo: %s\\n", type);
62                 printf("-----\\n\\n");
63             }
64         }
65     }
66
67     if(i==2){
68         //modifico las variables
69         omp_set_dynamic(3);
70         omp_set_num_threads(2);
71         omp_set_schedule(omp_sched_static,2);
72
73         omp_get_schedule(&tipo, &modif);
74         printf("\n-----\n");
75         printf("dentro del \\"parallel for DESPUES DE MODIFICALO\\"\\n");
76         printf("dyn-var: %d\\nnthreads-var: %d\\nrun-sched-var: %d\\n",
77                         omp_get_dynamic(), omp_get_max_threads(), modif);
78         switch(tipo){
79             case omp_sched_static:
80                 strcpy(type, "ESTATICO");
81                 break;
82             case omp_sched_dynamic:
83                 strcpy(type, "DINAMICO");
84                 break;
85             case omp_sched_guided:
86                 strcpy(type, "GUIDED");
87                 break;
88             case omp_sched_auto:
89                 strcpy(type, "AUTO");
90                 break;
91         }
92         printf("tipo: %s\\n", type);
93     }
```



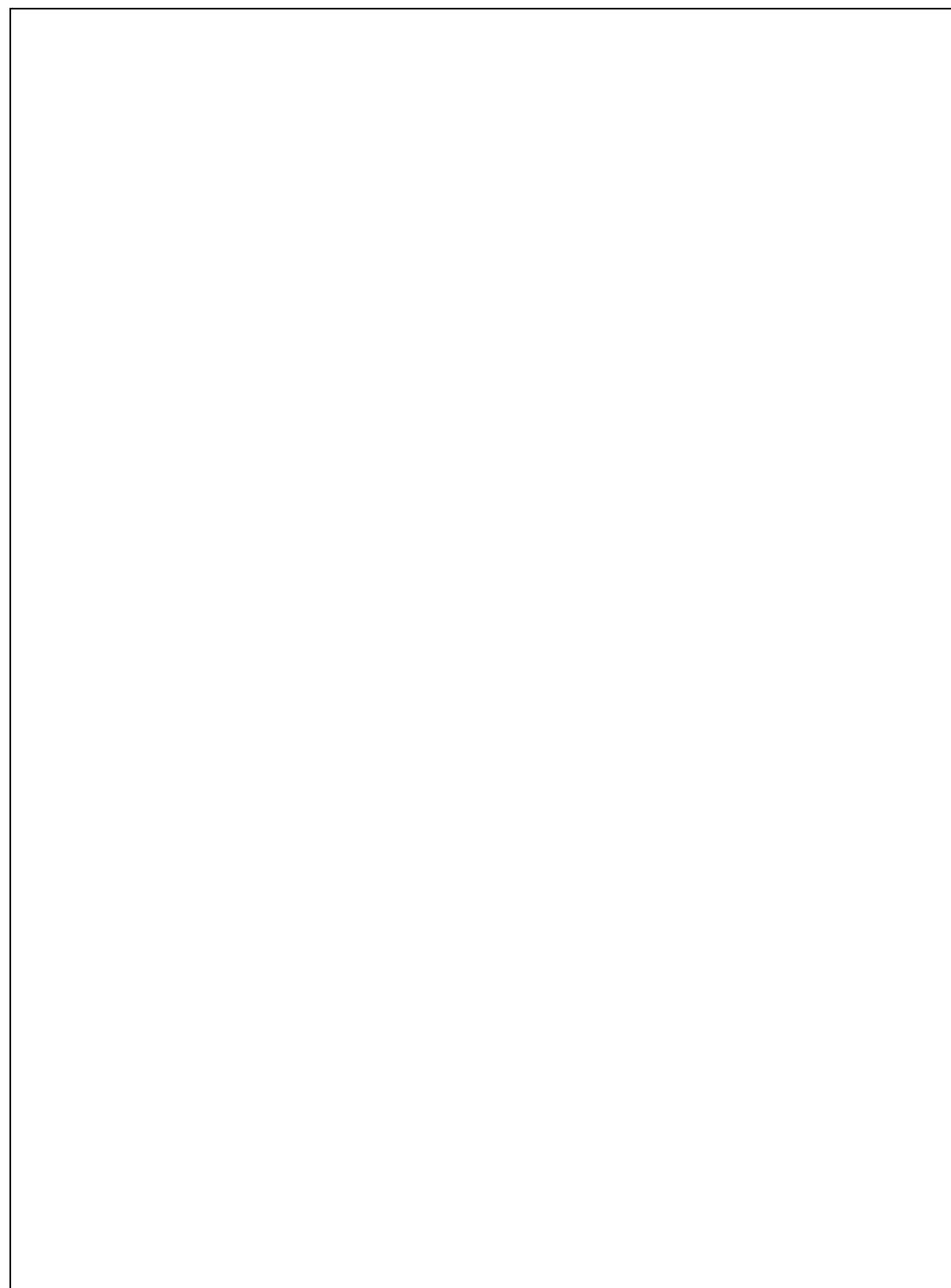
Descarga la APP de Wuolah.

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



```
ruben:~/Escritorio/AC/BP3/ejer5$ gcc -O2 -fopenmp -o schedule-clauseModificados
schedule-clauseModificado5.c
rulen:~/Escritorio/AC/BP3/ejer5$ ./schedule-clauseModificado5 10 4

-----
dentro del "parallel for ANTES DE MODIFICALO"
dyn-var: 0
nthreads-var: 8
run-sched-var: 4
tipo: DINAMICO
thread: 1 suma de a[8], suma = 8
thread: 1 suma de a[9], suma = 17
-----

thread: 4 suma de a[0], suma = 0
thread: 4 suma de a[1], suma = 1

-----
dentro del "parallel for DESPUES DE MODIFICALO"
dyn-var: 1
nthreads-var: 2
run-sched-var: 2
tipo: ESTATICO
-----

thread: 4 suma de a[2], suma = 3
thread: 4 suma de a[3], suma = 6
thread: 2 suma de a[4], suma = 4
thread: 2 suma de a[5], suma = 9
thread: 2 suma de a[6], suma = 15
thread: 2 suma de a[7], suma = 22

fuera del "parallel for " imprime suma = 17
dyn-var: 0
nthreads-var: 8
run-sched-var: 4
tipo: DINAMICO
rulen:~/Escritorio/AC/BP3/ejer5$ [REDACTED]

94         printf("-----\n\n");
95     }
96 }
97
98     suma += a[i];
99     printf("thread: %d suma de a[%d], suma = %d\n", omp_get_thread_num(), i, suma);
100 }
101
102 printf("\nfuera del \"parallel for \" imprime suma = %d\n", suma);
103 return 0;
104 }
```

CAPTURAS DE PANTALLA:

RESPUESTA:

Como se puede ver cambia los valores des pues de unos valores definidos, antes dinámico, chunk = 4 y el numero de hebras 8, después lo cambie a estático, con 2 hebras y chunk = 2. De tal forma que antes de modificar las variables de control, sus valores eran las variables de entorno y después de la modificación son

los proporcionados por las funciones en tiempo de ejecución. Pero es curioso que después de la región parallel vuelven a tener los valores que tenía antes de ser modificados.

Resto de ejercicios

- 6.** Implementar un programa secuencial en C que multiplique una matriz triangular por un vector (use variables dinámicas). Compare el orden de complejidad del código que ha implementado con el código que implementó para el producto matriz por vector.

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se debe inicializar las matrices antes del cálculo; (3) se debe imprimir siempre la primera y última componente del resultado antes de que termine el programa.

CAPTURA CÓDIGO FUENTE: pmtv-secuencial.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <stdbool.h>
5 #include <time.h>
6
7 int main(int argc, char const *argv[])
8 {
9
10    if(argc<2){
11        fprintf(stderr, "Error sintaxis: ./execute <size>\n");
12        exit(-1);
13    }
14
15    int i, n, e;
16    int **m, *v, *v_result;
17    struct timespec cgt1, cgt2;
18    double ncgt;
19
20    n = atoi(argv[1]);
21
22    v = (int*) malloc(n*sizeof(int));// malloc necesita el tamaño en bytes
23    v_result = (int*) malloc(n*sizeof(int));
24    m = (int**) malloc(n*sizeof(int *));
25
26
27    if ((v == NULL) || (v_result == NULL) || (m == NULL)) {
28        printf("No hay suficiente espacio para los vectores o matriz \n");
29        exit(-2);
30    }
31
32    for (int i = 0; i < n; ++i)
33    {
34        m[i] = (int*) malloc(n*sizeof(int));
35        if (m[i] == NULL) {
36            printf("No hay suficiente espacio para la matriz en la columna %d\n",i);
37            exit(-2);
38        }
39    }
40
41    //inicialización
42    for(i=0;i<n;++i){
43
44        for (e = 0; e < n; ++e){
45            if(i<=e)
46                m[i][e]=2;
47            else
48                m[i][e]=0;
49        }
50
51        v[i]=3;
52        v_result[i]=0;
53    }
54 }
```



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

```
55 //calculo
56 clock_gettime(CLOCK_REALTIME, &cgt1);
57
58 for(i=0;i<n;++i){
59     for (e = 0; e < n; ++e){
60         v_result[i] += m[i][e] * v[e];
61     }
62 }
63 clock_gettime(CLOCK_REALTIME, &cgt2);
64
65 ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
66 printf("Tiempo (seg.): %11.9f\n", ncgt);
67
68 int midle = n/2;
69
70 /*if(n>25){
71     printf("(");
72     for(int i=0;i<6;++i){
73         if(i+1==6)
74             printf("..., %d)\n", v_result[n-1]);
75         else if(i<5)
76             printf("%d, ", v_result[i]);
77     }
78 }else{
79     for(int i=0;i<n;++i){
80         printf("|");
81         for (int e = 0; e < n; ++e){
82             if(e+1==n)
83                 printf("%d| ", m[i][e]);
84             else
85                 printf("%d| ", m[i][e]);
86         }
87         if(midle==i)
88             printf("x |%d| = |%d|\n", v[i],v_result[i]);
89         else
90             printf(" |%d| |%d|\n", v[i],v_result[i]);
91     }
92 }*/
93
94 printf("v_result[%d] = %d\n", 0, v_result[0]);
95 printf("v_result[%d] = %d\n", n-1, v_result[n-1]);
96
97 free(v); // libera el espacio reservado para v1
98 free(v_result); // libera el espacio reservado para v2
99
100 for (int i = 0; i < n; ++i)
101     free(m[i]); // libera el espacio reservado de las columnas de la matriz
102
103 free(m); // libera el espacio reservado de las filas de la matriz
104
105 }
```

```
ruben:~/Escritorio/AC/BP3/ejer6$ ./pmtv-secuencial 10
Tiempo (seg.): 0.000000514
v_result[0] = 60
v_result[9] = 6
ruben:~/Escritorio/AC/BP3/ejer6$ ./pmtv-secuencial 500
Tiempo (seg.): 0.000340530
v_result[0] = 3000
v_result[499] = 6
ruben:~/Escritorio/AC/BP3/ejer6$ ./pmtv-secuencial 10000
Tiempo (seg.): 0.076258271
v_result[0] = 60000
v_result[9999] = 6
ruben:~/Escritorio/AC/BP3/ejer6$ ./pmv_secuencial 10
Tiempo (seg.): 0.000000782
V2[0] = 60
V2[9] = 60
ruben:~/Escritorio/AC/BP3/ejer6$ ./pmv_secuencial 500
Tiempo (seg.): 0.000396398
V2[0] = 3000
V2[499] = 3000
ruben:~/Escritorio/AC/BP3/ejer6$ ./pmv_secuencial 10000
Tiempo (seg.): 0.095766496
V2[0] = 60000
V2[9999] = 60000
ruben:~/Escritorio/AC/BP3/ejer6$
```

CAPTURAS DE PANTALLA:

7. Implementar en paralelo la multiplicación de una matriz triangular por un vector a partir del código secuencial realizado para el ejercicio anterior utilizando la directiva `for` de OpenMP. El código debe repartir entre los threads las iteraciones del bucle que recorre las filas. Dibujar en el cuaderno de prácticas la descomposición de dominio utilizada (Lección 4/Tema 2) en el código paralelo implementado para asignar tareas a los threads (Lección 5/Tema 2). Añadir lo necesario para que el usuario pueda fijar la planificación de tareas usando la variable de entorno `OMP_SCHEDULE`. Obtener en atcgrid los tiempos de ejecución del código paralelo (usando, como siempre, `-O2` al compilar) que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para chunk de 1, 64 y el chunk por defecto para la alternativa. Use un tamaño de vector N múltiplo del número de cores y de 64 que no sea inferior a 15360. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 3 dos veces con los tiempos obtenidos. Representar el tiempo para `static`, `dynamic` y `guided` en función del tamaño del chunk en una gráfica. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

Conteste a las siguientes preguntas: (a) ¿Qué valor por defecto usa OpenMP para chunk con `static`, `dynamic` y `guided`? Indique qué ha hecho para obtener este valor por defecto para cada alternativa. (b) ¿Qué número de operaciones de multiplicación y suma realizan cada uno de los threads en la asignación `static` para cada uno de los chunks? (c) Con la asignación `dynamic` y `guided`, ¿qué cree que debe ocurrir con el número de operaciones de multiplicación y suma que realizan cada uno de los threads?

RESPUESTA:

- (a) Por defecto usa el valor 0 en `static`, y 1 para `dynamic` y `guided`, pero saber que valores daba he imprimido como se puede ver en las capturas de pantalla el chunk y si es estático, dinámico o `guided`.
- (b) Depende según la fila de la matriz, puesto que la matriz es triangular, de tal forma que la fila 1 hace N

operaciones, la fila 2 hace N-1 operaciones, y así hasta la ultima fila que hace 1 operación.

(c) Eso depende de las hebras disponibles, es decir cada hebra hará x operaciones distintas, dependiendo de la disponibilidad estarán libres u ocupadas.

CAPTURA CÓDIGO FUENTE: pmtv-OpenMP.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 int main(int argc, char const *argv[])
11 {
12
13     if(argc<2){
14         fprintf(stderr, "Error sintaxis: ./execute <size>\n");
15         exit(-1);
16     }
17
18     int i, n, e, suma;
19     int **m, *v, *v_result;
20     double t_ini, t_fin, t_elap;
21
22     n = atoi(argv[1]);
23
24     v = (int*) malloc(n*sizeof(int)); // malloc necesita el tamaño en bytes
25     v_result = (int*) malloc(n*sizeof(int));
26     m = (int**) malloc(n*sizeof(int *));
27
28     if ((v == NULL) || (v_result == NULL) || (m == NULL)) {
29         printf("No hay suficiente espacio para los vectores o matriz \n");
30         exit(-2);
31     }
32
33     for (int i = 0; i < n; ++i)
34     {
35         m[i] = (int*) malloc(n*sizeof(int));
36         if (m[i] == NULL) {
37             printf("No hay suficiente espacio para la matriz en la columna %d\n",i);
38             exit(-2);
39         }
40     }
41
42
43     //inicialización
44     for(i=0;i<n;++i){
45
46         for (e = 0; e < n; ++e){
47             if(i<=e)
48                 m[i][e]=2;
49             else
50                 m[i][e]=0;
51         }
52
53         v[i]=3;
54         v_result[i]=0;
55     }
56 }
```



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

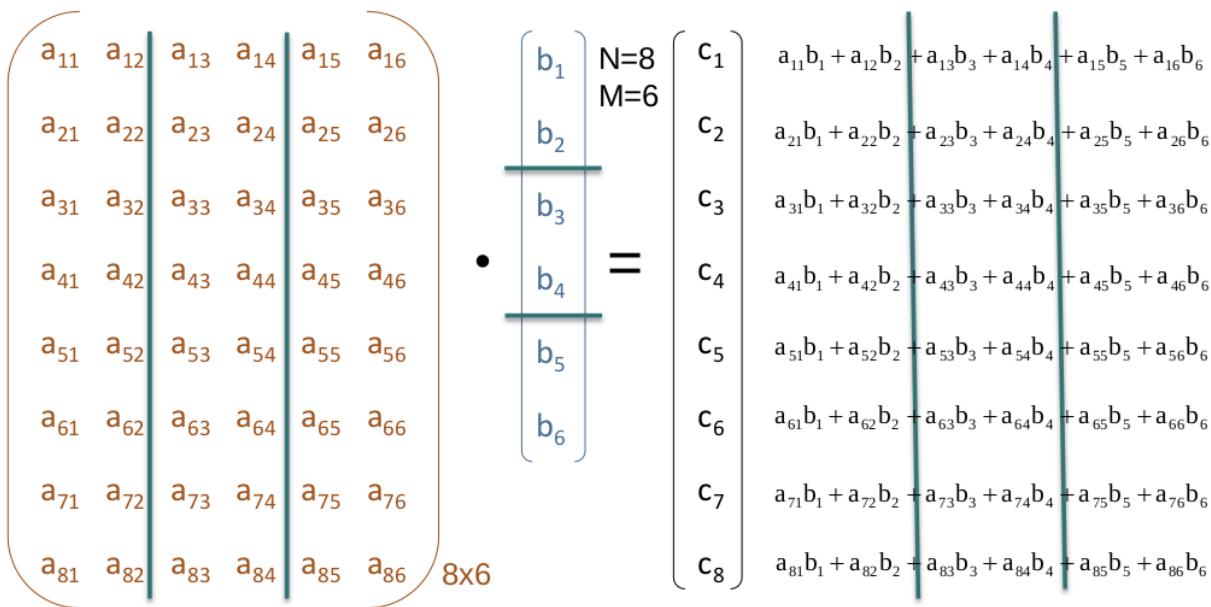
Continúa d...

```
57 //calculo
58 #pragma omp parallel private(e)
59 {
60     t_ini = omp_get_wtime();
61
62     #pragma omp for firstprivate(suma) schedule(runtime)
63     for(i=0;i<n;++i){
64         suma = 0;
65         for (e = 0; e < n; ++e){
66             suma += m[i][e] * v[e];
67         }
68         #pragma single
69         v_result[i] = suma;
70     }
71
72     t_fin = omp_get_wtime();
73 }
74
75 t_elap = t_fin - t_ini;
76 printf("Tiempo (seg.): %11.9f \t tamaño: %d\n", t_elap, n);
77
78 printf("v_result[%d] = %d\n", 0, v_result[0]);
79 printf("v_result[%d] = %d\n", n-1, v_result[n-1]);
80
81 free(v); // libera el espacio reservado para v1
82 free(v_result); // libera el espacio reservado para v2
83
84 for (int i = 0; i < n; ++i)
85     free(m[i]); // libera el espacio reservado de las columnas de la matriz
86
87 free(m); // libera el espacio reservado de las filas de la matriz
88 return 0;
89 }
```

DESCOMPOSICIÓN DE DOMINIO:

Para 4 hebras y le hemos asignado estatica, con chunk a 2. De tal forma que cada hebra multiplicara 2 filas de la matriz por el vector de forma consecutiva. (esta explicación grafica lo he sacado de las transparencias de teoria)

$$c = A \bullet b; \quad c_i = \sum_{k=0}^{M-1} a_{ik} \bullet b_k = a_i^T \bullet b, \quad c(i) = \sum_{k=0}^{M-1} A(i, k) \bullet b(k), \quad i = 0, \dots, N-1$$



Tambien puede ser por columnas, no tiene por que ser en filas.

Pero en mi código lo he hecho por filas.

CAPTURAS DE PANTALLA:

```
static y chunk 64 -----
Tiempo (seg.): 0.023959676      tamaño: 15400
v_result[0] = 92400
v_result[15399] = 6
chunk: 64

CÁPTURA CÓDIGO FUENTE myt0001.c

dynamic y chunk por defecto -----
Tiempo (seg.): 0.011035033      tamaño: 15400
v_result[0] = 92400
v_result[15399] = 6
DINAMICO      chunk: 1

dynamic y chunk 1 -----
Tiempo (seg.): 0.021852145      tamaño: 15400
v_result[0] = 92400
v_result[15399] = 6
DINAMICO      chunk: 1

dynamic y chunk 64 -----
Tiempo (seg.): 0.000589630      tamaño: 15400
v_result[0] = 92400
v_result[15399] = 6
DINAMICO      chunk: 64

guided y chunk por defecto -----
Tiempo (seg.): 0.014435258      tamaño: 15400
v_result[0] = 92400
v_result[15399] = 6
GUIDED     chunk: 1

guided y chunk 1 -----
Tiempo (seg.): 0.016602084      tamaño: 15400
v_result[0] = 92400
v_result[15399] = 6
GUIDED     chunk: 1

guided y chunk 64 -----
Tiempo (seg.): 0.020945717      tamaño: 15400
v_result[0] = 92400
v_result[15399] = 6
GUIDED     chunk: 64

[b4estudiante10@atcgrid ejer7]$ █
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-OpenMP_atcgrid.sh

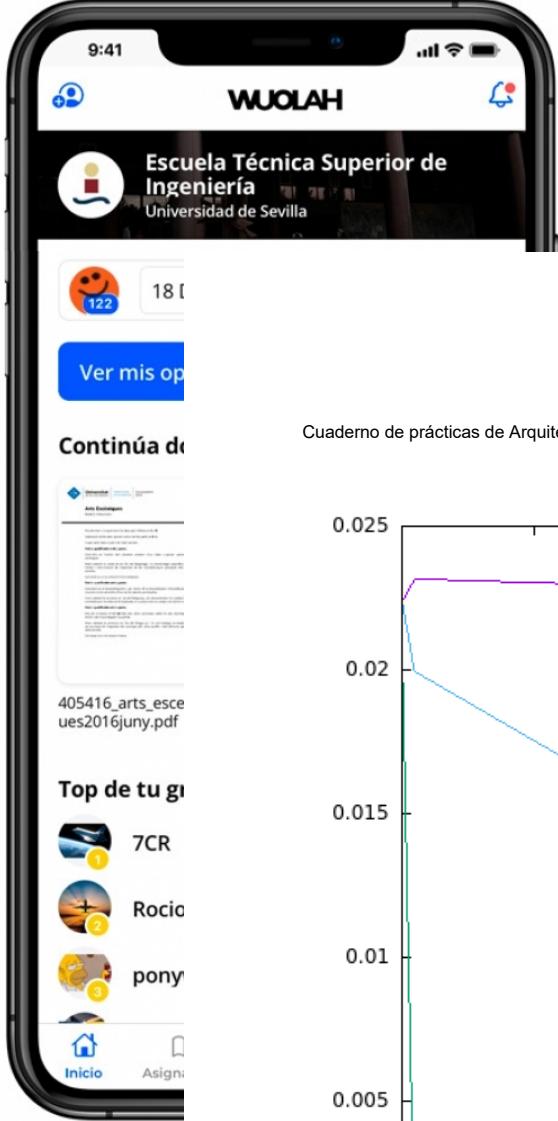
```

1  #!/bin/bash
2
3  export OMP_NUM_THREADS=12
4
5  declare -a planificacion=("static" "dynamic" "guided")
6
7  for i in "${planificacion[@]}"
8  do
9      for (( e = 0; e < 3; ++e ));| do
10         if [ $e -eq 0 ]
11         then
12             echo "$i y chunk por defecto -----"
13             export OMP_SCHEDULE=$i
14         elif [ $e -eq 1 ]
15         then
16             echo "$i y chunk 1 -----"
17             export OMP_SCHEDULE="$i,1"
18         else
19             echo "$i y chunk 64 -----"
20             export OMP_SCHEDULE="$i,64"
21         fi
22         ./pmvt-OpenMP 15400
23         echo -e "\n"
24     done
25 done

```

Tabla 3 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño N= 15400 , 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0,022298416	0,022041062	0,009693813
1	0,023113169	0,000083709	0,023058625
64	0,022222614	0,020940898	0,02329722
Chunk	Static	Dynamic	Guided
por defecto	0,016982589	0,022317173	0,022532305
1	0,027866177	0,000027670	0,019903677
64	0,024292115	0,021909261	0,003108142



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Viendo la grafica vemos que la que tiene mejores prestaciones es la planificación guided, puesto que reduce bastante el tiempo de ejecución, pero de forma teórica es mejor la planificación dynamica, por que aprovechala las iteraciones dividiéndolas por el numero de chunk iteraciones y se le asignan en tiempo de ejecución.

8. Implementar un programa secuencial en C que calcule la multiplicación de matrices cuadradas, B y C:

$$A = B \bullet C; A(i, j) = \sum_{k=0}^{N-1} B(i, k) \bullet C(k, j), i, j = 0, \dots, N - 1$$

NOTAS: (1) el número de filas/columnas debe ser un argumento de entrada; (2) se deben inicializar las matrices antes del cálculo; (3) se debe imprimir siempre las componentes (0,0) y (N-1, N-1) del resultado antes de que termine el programa.

CAPTURA CÓDIGO FUENTE: pmm-secuencial.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  #ifdef _OPENMP
6      #include <omp.h>
7  #else
8      #define omp_get_thread_num() 0
9  #endif
10
11
12 int main(int argc, char const *argv[])
13 {
14
15     if(argc<2){
16         fprintf(stderr, "Error falta recibir por parametro un numero.\n");
17         exit(-1);
18     }
19
20     unsigned int n=atoi(argv[1]);
21
22     struct timespec cgt1, cgt2;
23     double ncgt;
24
25     int **m_r, **a, **b;
26     b = (int**) malloc(n*sizeof(int *));// malloc necesita el tamaño en bytes
27     a = (int**) malloc(n*sizeof(int *));
28     m_r = (int**) malloc(n*sizeof(int *));
29
30     if ((a == NULL) || (b == NULL) || (m_r == NULL)) {
31         printf("No hay suficiente espacio para los vectores o matriz \n");
32         exit(-2);
33     }
34
35     for (int i = 0; i < n; ++i){
36         m_r[i] = (int*) malloc(n*sizeof(int));
37         a[i] = (int*) malloc(n*sizeof(int));
38         b[i] = (int*) malloc(n*sizeof(int));
39         if (m_r[i] == NULL || a[i] == NULL || b[i] == NULL) {
40             printf("No hay suficiente espacio para la matriz en la columna %d\n",i);
41             exit(-2);
42         }
43     }
44
45     //inicialización
46
47     for(int i=0;i<n;++i){
48         for (int e = 0; e < n; ++e){
49             b[i][e]=2;
50             a[i][e]=3;
51             m_r[i][e]=0;
52         }
53     }

```




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

```
55      //calculo
56
57      clock_gettime(CLOCK_REALTIME, &cgt1);
58
59      for(int i=0;i<n;++i){
60          for (int e = 0; e < n; ++e){
61              for (int k = 0; k < n; ++k)
62                  m_r[i][e]+=a[i][k] * b[k][e];
63          }
64      }
65
66      clock_gettime(CLOCK_REALTIME, &cgt2);
67
68      //resultado
69
70      ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
71      printf("Tiempo (seg.): %11.9f\n", ncgt);

109     printf("m_r[%d] = %d\n", 0, m_r[0][0]);
110    printf("m_r[%d] = %d\n", n-1, m_r[n-1][n-1]);
111
112    for (int i = 0; i < n; ++i)
113    {
114        free(m_r[i]); // libera el espacio reservado de las columnas de la matriz
115        free(a[i]);
116        free(b[i]);
117    }
118
119    free(m_r); // libera el espacio reservado de las filas de la matriz
120    free(a);
121    free(b);
122
123    return 0;
124 }
```

CAPTURAS DE PANTALLA:

```

ruben:~/Escritorio/AC/BP3/ejer8$ gcc -O2 -fopenmp -o pmm_secuencial pmm_secuencial.c
ruben:~/Escritorio/AC/BP3/ejer8$ ./pmm_secuencial 100
Tiempo (seg.): 0.001578731
V2[0] = 600
V2[99] = 600
ruben:~/Escritorio/AC/BP3/ejer8$ ./pmm_secuencial 5
Tiempo (seg.): 0.000000729
V2[0] = 30
V2[4] = 30
ruben:~/Escritorio/AC/BP3/ejer8$ ./pmm_secuencial 1000
Tiempo (seg.): 1.475295426
V2[0] = 6000
V2[999] = 6000
ruben:~/Escritorio/AC/BP3/ejer8$ ./pmm_secuencial 4
Tiempo (seg.): 0.000000633
V2[0] = 24
V2[3] = 24
ruben:~/Escritorio/AC/BP3/ejer8$ 

```

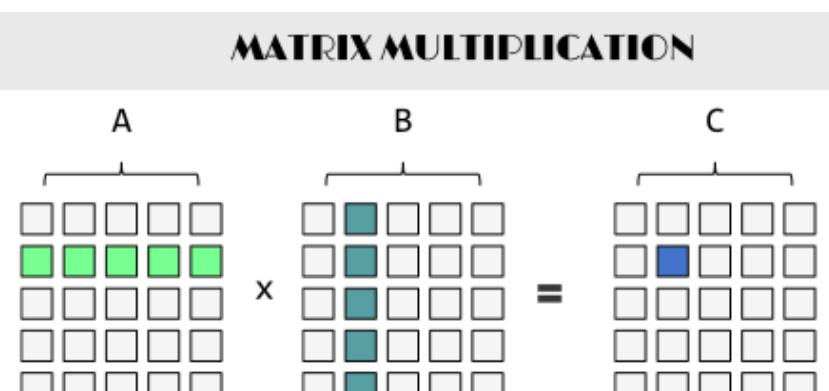
9. Implementar en paralelo la multiplicación de matrices cuadradas con OpenMP a partir del código escrito en el ejercicio anterior. Use las directivas, las cláusulas y las funciones de entorno que considere oportunas. Se debe paralelizar también la inicialización de las matrices. Dibuje en su cuaderno de prácticas la descomposición de dominio que ha utilizado en el código paralelo implementado para asignar tareas a los threads (Lección 4/Tema 2, Lección 5/Tema 2).

10.

DESCOMPOSICIÓN DE DOMINIO:

Básicamente lo que hace cada hebra es multiplicar los valores de cada casilla de la fila de la matriz A, por los valores de la columna de la matriz B y sumar todos esos valores y almacenarlos en la casilla correspondiente en la matriz M_R . De tal forma que cada hebra se corresponde con una fila de la matriz A y una columna de la matriz B.

Para 4 hebras y la planificación estática:



CAPTURA CÓDIGO FUENTE: pmm-OpenMP.c

```

45     #pragma omp parallel private(e,k)
46     {
47         //inicialización
48         #pragma omp for schedule(runtime)
49         for(i=0;i<n;++i){
50             for (e = 0; e < n; ++e){
51                 b[i][e]=2;
52                 a[i][e]=3;
53                 m_r[i][e]=0;
54             }
55         }
56
57         //calculo
58         #pragma omp single
59         t_ini = omp_get_wtime();
60
61         #pragma omp for schedule(runtime)
62         for(i=0;i<n;++i){
63             for (e = 0; e < n; ++e){
64                 for (k = 0; k < n; ++k)
65                     m_r[i][e]+=a[i][k] * b[k][e];
66             }
67         }
68
69         #pragma omp single
70         t_fin = omp_get_wtime();
71     }

```

CAPTURAS DE PANTALLA:

```

ruben:~/Escritorio/AC/BP3/ejer9$ gcc -O2 -fopenmp -o pmm-OpenMP pmm-OpenMP.c
ruben:~/Escritorio/AC/BP3/ejer9$ ./pmm-OpenMP 10
Tiempo (seg.): 0.000007929      tamaño: 10
m_r[0] = 60
m_r[9] = 60
ruben:~/Escritorio/AC/BP3/ejer9$ ./pmm-OpenMP 500
Tiempo (seg.): 0.040859628      tamaño: 500
m_r[0] = 3000
m_r[499] = 3000
ruben:~/Escritorio/AC/BP3/ejer9$ ./pmm-OpenMP 1000
Tiempo (seg.): 1.454275652      tamaño: 1000
m_r[0] = 6000
m_r[999] = 6000
ruben:~/Escritorio/AC/BP3/ejer9$ ./pmm-OpenMP 1500
Tiempo (seg.): 5.705300162      tamaño: 1500
m_r[0] = 9000
m_r[1499] = 9000
ruben:~/Escritorio/AC/BP3/ejer9$ █

```

10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en su PC del código paralelo implementado para dos tamaños de las matrices. Debe recordar usar `-O2` al compilar. El número de núcleos máximo en este estudio debe ser el igual al de núcleos físicos del computador. Presente los resultados del estudio en tablas de valores y en gráficas. Escoger los tamaños de manera que se observe diferentes

curvas de escalabilidad en las gráficas que entregue en su cuaderno de prácticas (pruebe con valores de N entre 100 y 1500). Consulte la Lección 6/Tema 2. Incluya los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

ESTUDIO DE ESCALABILIDAD EN atcgrid:

SCRIPT: pmm-OpenMP_atcgrid.sh

```

1 #!/bin/bash
2
3  export OMP_NUM_THREADS=1
4
5  for (( e = 100; e <= 1500 ; e = e + 100 )); do
6
7      ./pmm-OpenMP $e
8      echo -e "\n"
9  done
10

```

Tiempos de ejecución:

Tamaños	1 hebra	2 hebras	4 hebras	8 hebras	16 hebras	24 hebras
100	0.001453415	0.000829415	0.000422927	0.000166405	0.000455186	0.000647755
200	0.007807642	0.004596261	0.003276037	0.001350099	0.003258729	0.003338225
300	0.018493509	0.009363594	0.006780051	0.004586741	0.006648259	0.007882832
400	0.045345290	0.023154516	0.011881378	0.011515182	0.011739828	0.011978656
500	0.100594858	0.051030086	0.026319094	0.023910260	0.025198599	0.025336085
600	0.180071285	0.091696102	0.047232695	0.045520525	0.049535481	0.046897084
700	0.333272137	0.172539147	0.091992447	0.089687333	0.090507902	0.093148636
800	0.421684226	0.216793250	0.114044540	0.273850404	0.258940941	0.244630709
900	0.593866553	0.311264288	0.160416961	0.506304309	0.464158978	0.489409789
1000	0.859047033	0.448778873	0.225660043	0.909835901	0.922047397	0.928290565
1100	1.432968775	0.815826226	0.422963534	1.278183822	1.263212323	1.273579665
1200	2.692079440	1.544218032	0.791155843	1.633454110	1.651012048	1.649968738
1300	2.271405801	1.241683695	0.634848485	2.236647142	2.243738631	2.247868417
1400	3.543054465	2.044599589	0.984461902	2.825086622	2.843732260	2.850090539
1500	6.333550520	3.369916352	1.678959187	3.505608818	3.553548656	3.551300410

Ganancia en prestaciones Ts/Tp(p)

Tamaños	2 hebras	4 hebras	8 hebras	16 hebras	24 hebras
---------	----------	----------	----------	-----------	-----------



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

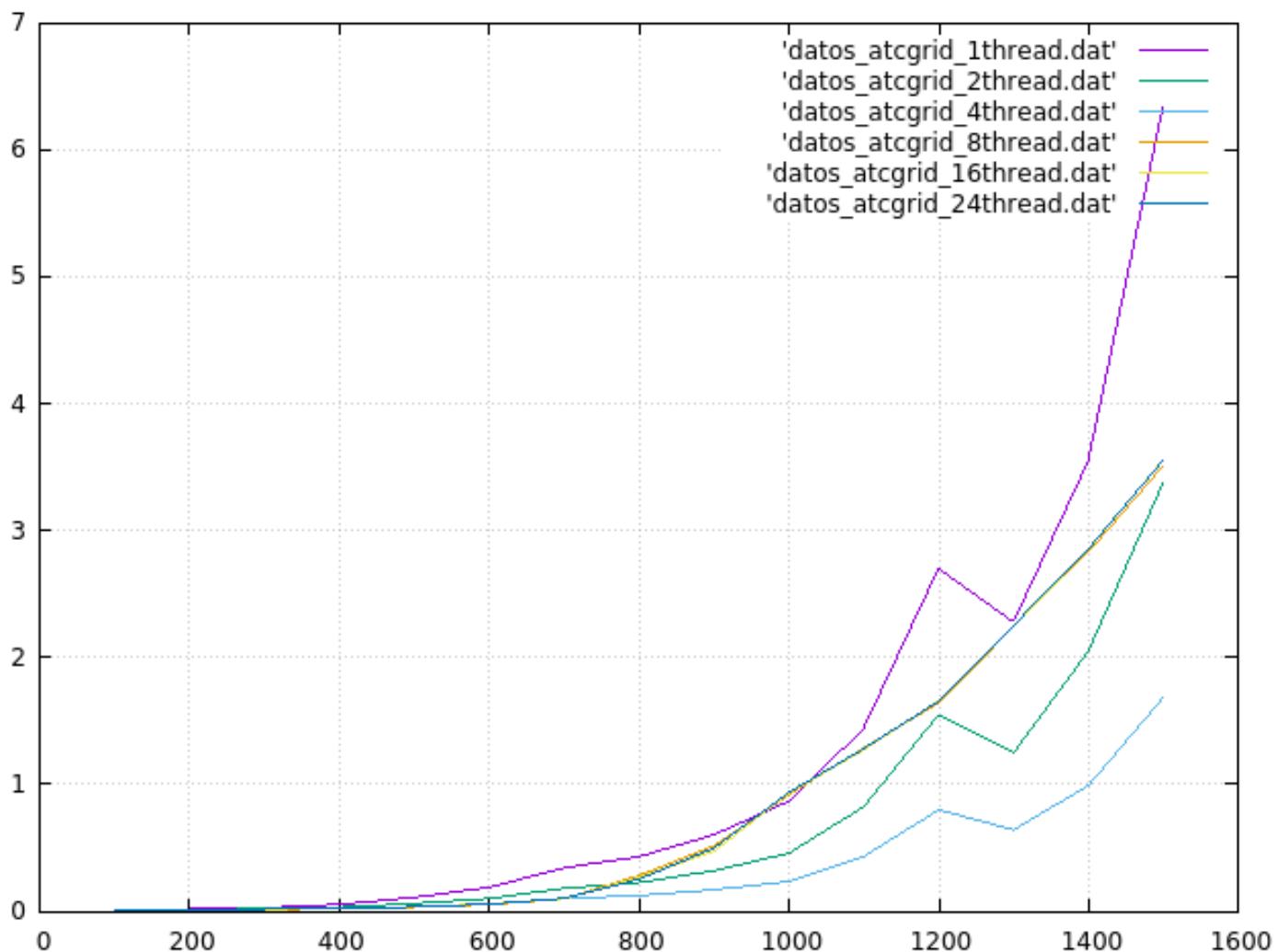
Cuaderno de prácticas de Arquitectura de Computadores, Grado en Ingeniería Informática

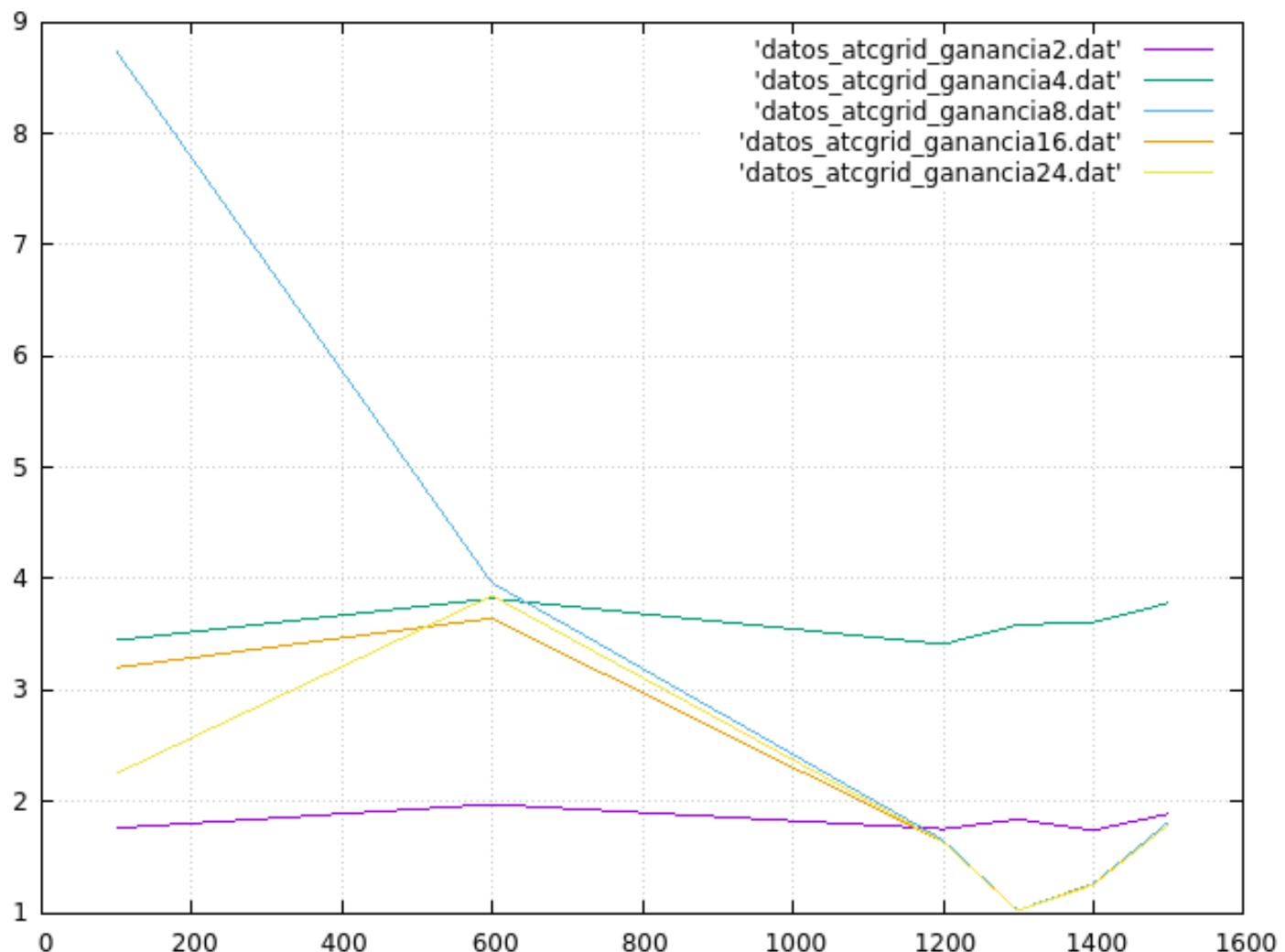
100	1,752337491	3,436562338	8,734202698	3,193013406	2,243772723
600	1,963783422	3,812428764	3,95582619	3,635198071	3,839711761
1200	1,743328587	3,402717004	1,648090034	1,630563171	1,63159421
1300	1,829295021	3,577870712	1,01554052	1,012330835	1,010470979
1400	1,732884269	3,598975702	1,254140116	1,245917035	1,243137513
1500	1,879438496	3,772307611	1,806690606	1,782317096	1,783445439

Top de tu grupo

- 7CR
- Rocio
- pony

Inicio Asignaciones





ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pclocal.sh

```

1  #!/bin/bash
2
3  export OMP_NUM_THREADS=5
4  i=100
5  while [ "$i" -le 1500 ]; do
6
7      ./pmm-OpenMP $i
8      i=$(( i + 100 ))
9  done
10

```

Tiempos de ejecución

Tamaños	1 hebra	2 hebras	3 hebras	4 hebras
100	0.001891071	0.001099754	0.000708320	0.000740285
200	0.011020156	0.006337533	0.004731924	0.004489026
300	0.031087738	0.015593183	0.011224406	0.010249125

400	0.077015236	0.038963996	0.027200822	0.025843903
500	0.167636577	0.086123964	0.056615137	0.043729230
600	0.302412617	0.151542650	0.100323743	0.087867942
700	0.596301900	0.298433102	0.201199836	0.150341024
800	0.731942578	0.371414529	0.249370011	0.185468438
900	1.036909047	0.540877735	0.363091912	0.278430381
1000	1.498174227	0.829173068	0.516203380	0.393119517
1100	2.750567964	1.255527722	0.900908814	0.636755962
1200	4.580440992	2.644752052	1.719189643	1.266876836
1300	4.024508730	2.045002918	1.289623761	0.996984588
1400	6.021341397	3.387337269	2.077776550	1.748240992
1500	10.534797572	5.723399523	3.736361998	2.786624162

Ganancia en prestaciones Ts/Tp(p)

Tamaños	2 hebras	3 hebras	4 hebras
100	1,719540006	2,669797549	2,554517517
600	1,995561098	3,014367367	3,441671787
1200	1,731898077	2,664302342	3,615537724
1300	1,967972121	3,120684382	4,036680986
1400	1,777603149	2,897973508	3,44422847
1500	1,840653886	2,819533433	3,780487414

9:41



WUOLAH



WUOLAH



Escuela Técnica Superior de
Ingeniería
Universidad de Sevilla



122



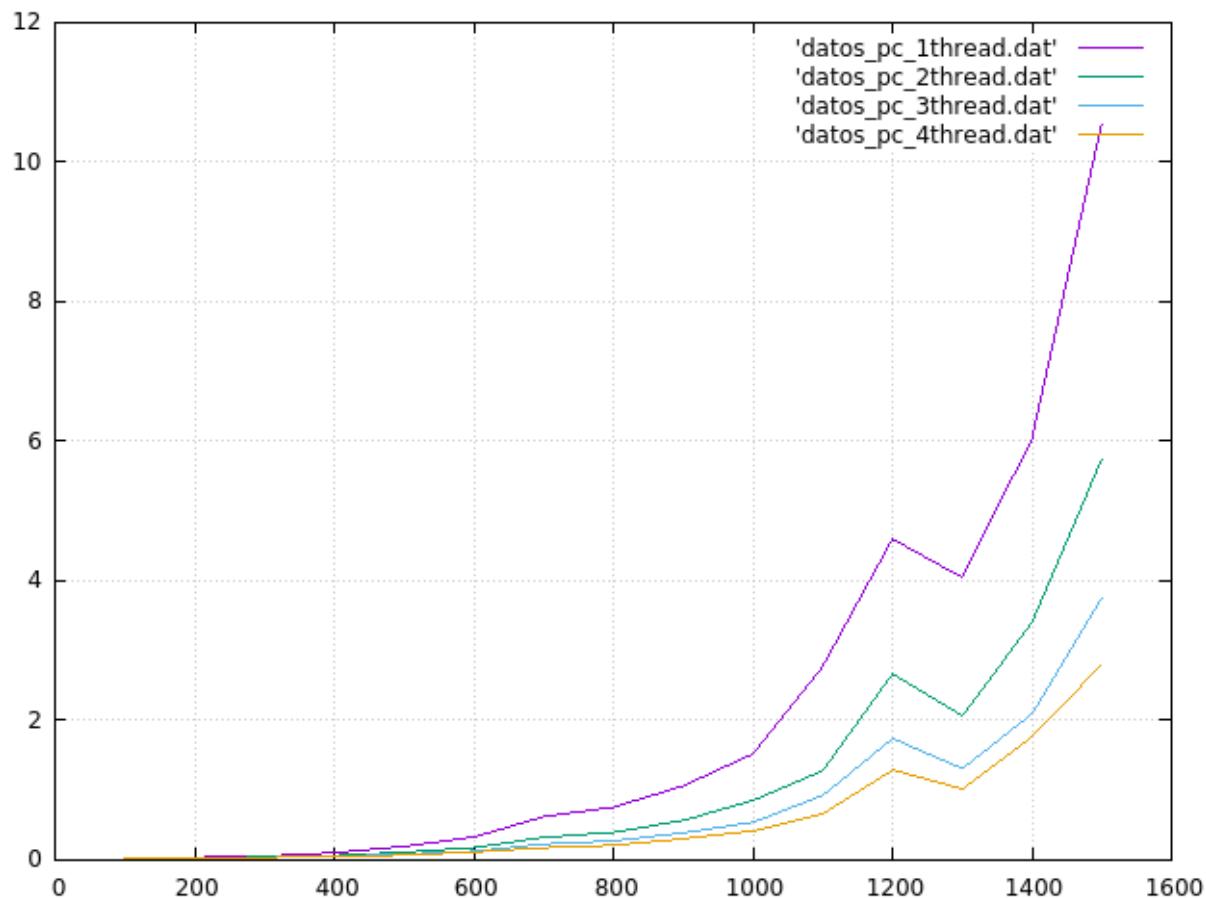
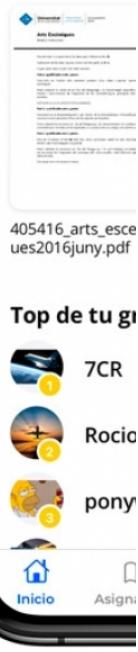
18



Ver mis op

Continúa d

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



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

