

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

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

Estudiante (nombre y apellidos): Clara María Romero Lara

Grupo de prácticas: D1

Fecha de entrega: 06-05-2020

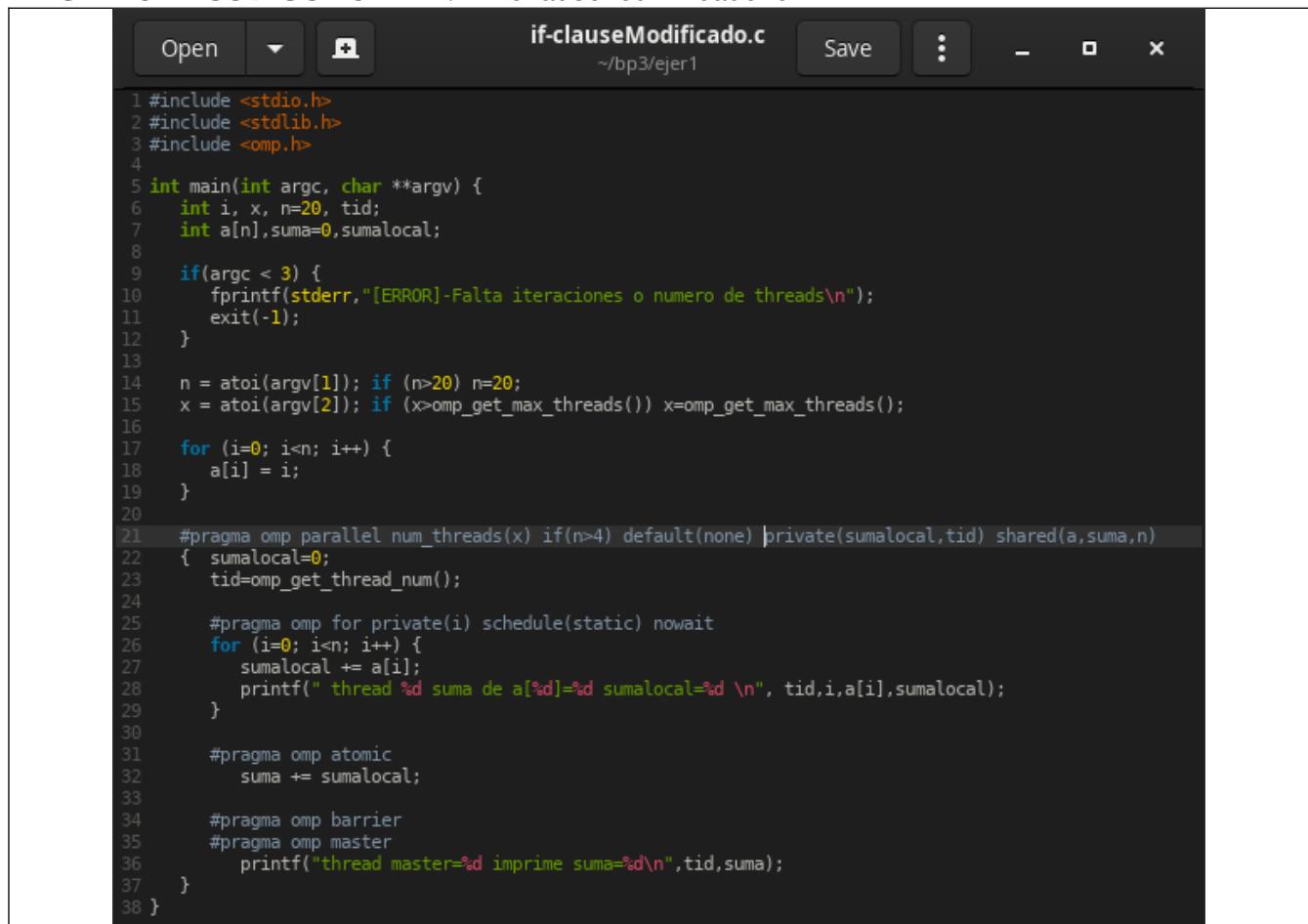
Fecha evaluación en clase: 08-05-2020

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. 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



```
Open ▾ + if-clauseModificado.c ~/bp3/ejer1 Save ⋮ - □ ×
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <omp.h>
4
5 int main(int argc, char **argv) {
6     int i, x, n=20, tid;
7     int a[n], suma=0,sumalocal;
8
9     if(argc < 3) {
10         fprintf(stderr, "[ERROR]-Falta iteraciones o numero de threads\n");
11         exit(-1);
12     }
13
14     n = atoi(argv[1]); if (n>20) n=20;
15     x = atoi(argv[2]); if (x>omp_get_max_threads()) x=omp_get_max_threads();
16
17     for (i=0; i<n; i++) {
18         a[i] = i;
19     }
20
21     #pragma omp parallel num_threads(x) if(n>4) default(None) private(sumalocal,tid) shared(a,suma,n)
22     { sumalocal=0;
23         tid=omp_get_thread_num();
24
25         #pragma omp for private(i) schedule(static) nowait
26         for (i=0; i<n; i++) {
27             sumalocal += a[i];
28             printf(" thread %d suma de a[%d]=%d sumalocal=%d \n", tid,i,a[i],sumalocal);
29         }
30
31         #pragma omp atomic
32         suma += sumalocal;
33
34         #pragma omp barrier
35         #pragma omp master
36         printf("thread master=%d imprime suma=%d\n",tid,suma);
37     }
38 }
```

CAPTURAS DE PANTALLA:

```
~/bp3/ejer1
→ gcc -O2 -fopenmp if-clauseModificado.c -o ifclauseMod

~/bp3/ejer1
→ ./ifclauseMod 10 5
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 3 suma de a[6]=6 sumalocal=6
thread 3 suma de a[7]=7 sumalocal=13
thread 4 suma de a[8]=8 sumalocal=8
thread 4 suma de a[9]=9 sumalocal=17
thread 1 suma de a[2]=2 sumalocal=2
thread 1 suma de a[3]=3 sumalocal=5
thread 2 suma de a[4]=4 sumalocal=4
thread 2 suma de a[5]=5 sumalocal=9
thread master=0 imprime suma=45

~/bp3/ejer1
→ ./ifclauseMod 10 10
thread 2 suma de a[4]=4 sumalocal=4
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread 4 suma de a[6]=6 sumalocal=6
thread 6 suma de a[8]=8 sumalocal=8
thread 7 suma de a[9]=9 sumalocal=9
thread 3 suma de a[5]=5 sumalocal=5
thread 5 suma de a[7]=7 sumalocal=7
thread 1 suma de a[2]=2 sumalocal=2
thread 1 suma de a[3]=3 sumalocal=5
thread master=0 imprime suma=45
```

RESPUESTA: La cláusula nos permite seleccionar cuántas hebras deseamos que ejecute el programa. En la ejecución de 10 iteraciones y 5 hebras vemos que las hebras 0-5 ejecutan el programa. En la segunda ejecución hemos indicado 10 hebras cuando los cores lógicos de mi ordenador son 8, así que automáticamente el número de hebras pasa a ser 8 (0-7).

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

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

RESPUESTA:

Con static: las hebras ejecutan un total del mismo número de iteraciones. Se asigna en Round Robin.

Con dynamic: cada hebra ejecuta, al menos, un número de iteraciones igual al chunk. Los threads más rápidos ejecutan más ciclos.

Con guided: cada hebra ejecuta también al menos tantas iteraciones como tamaño del chunk. Comienza con un bloque largo, y luego se va reduciendo.

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`

```

11
12     if(argc < 3) {
13         fprintf(stderr, "\nFalta iteraciones o chunk \n");
14         exit(-1);
15     }
16
17     n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);
18
19     int dyn_var = omp_get_dynamic();
20     int nthreads_var = omp_get_max_threads();
21     int thread_limit_var = omp_get_thread_limit();
22     int run_sched_var;
23     omp_sched_t kind;
24     omp_get_schedule(&kind, &run_sched_var);
25
26     for (i=0; i<n; i++) a[i] = i;
27
28     printf("\nFUERA DE REGION PARALELA:");
29     printf("\n\tdyn_var: %d \
30 \tnthreads_var: %d \
31 \tthread_limit_var: %d \
32 \trun_sched_var: %d \
33 ", dyn_var, nthreads_var, thread_limit_var, run_sched_var);
34
35 #pragma omp parallel for firstprivate(suma) \
36           lastprivate(suma) schedule(dynamic,chunk)
37     for (i=0; i<n; i++) {
38         suma = suma + a[i];
39         printf(" thread %d suma a[%d]=%d suma=%d \n", omp_get_thread_num(),i,a[i],suma);
40     }
41
42     #pragma omp single
43     {
44         dyn_var = omp_get_dynamic();
45         nthreads_var = omp_get_max_threads();
46         thread_limit_var = omp_get_thread_limit();
47         omp_get_schedule(&kind, &run_sched_var);
48
49         printf("\nDENTRO DE REGION PARALELA:");
50         printf("\n\tdyn_var: %d \
51 \tnthreads_var: %d \
52 \tthread_limit_var: %d \
53 \trun_sched_var: %d \
54 ", dyn_var, nthreads_var, thread_limit_var, run_sched_var);
55     }
56
57     printf("Fuera de 'parallel for' suma=%d\n",suma);
58 }
```

CAPTURAS DE PANTALLA:

```

~/bp3/ejer3
→ ./scheduledModificado 5 5

FUERA DE REGION PARALELA:
    dyn_var: 0
    nthreads_var: 6
    thread_limit_var: 2147483647
    run_sched_var: 1
    thread 0 suma a[0]=0 suma=0
    thread 0 suma a[1]=1 suma=1
    thread 0 suma a[2]=2 suma=3
    thread 0 suma a[3]=3 suma=6
    thread 0 suma a[4]=4 suma=10

DENTRO DE REGION PARALELA:
    dyn_var: 0
    nthreads_var: 6
    thread_limit_var: 2147483647
    run_sched_var: 1
    Fuera de 'parallel for' suma=10

~/bp3/ejer3
→ export OMP_NUM_THREADS=10

~/bp3/ejer3
→ ./scheduledModificado 5 5

FUERA DE REGION PARALELA:
    dyn_var: 1
    nthreads_var: 10
    thread_limit_var: 2147483647
    run_sched_var: 1
    thread 1 suma a[0]=0 suma=0
    thread 1 suma a[1]=1 suma=1
    thread 1 suma a[2]=2 suma=3
    thread 1 suma a[3]=3 suma=6
    thread 1 suma a[4]=4 suma=10

DENTRO DE REGION PARALELA:
    dyn_var: 1
    nthreads_var: 10
    thread_limit_var: 2147483647
    run_sched_var: 1
    Fuera de 'parallel for' suma=10

~/bp3/ejer3
→ export OMP_DYNAMIC=true

~/bp3/ejer3
→ ./scheduledModificado 5 5

FUERA DE REGION PARALELA:
    dyn_var: 1
    nthreads_var: 6
    thread_limit_var: 2147483647
    run_sched_var: 1
    thread 4 suma a[0]=0 suma=0
    thread 4 suma a[1]=1 suma=1
    thread 4 suma a[2]=2 suma=3
    thread 4 suma a[3]=3 suma=6
    thread 4 suma a[4]=4 suma=10

DENTRO DE REGION PARALELA:
    dyn_var: 1
    nthreads_var: 6
    thread_limit_var: 2147483647
    run_sched_var: 1
    Fuera de 'parallel for' suma=10

~/bp3/ejer3
→ export OMP_THREAD_LIMIT=4

~/bp3/ejer3
→ ./scheduledModificado 5 5

FUERA DE REGION PARALELA:
    dyn_var: 1
    nthreads_var: 10
    thread_limit_var: 4
    run_sched_var: 1
    thread 1 suma a[0]=0 suma=0
    thread 1 suma a[1]=1 suma=1
    thread 1 suma a[2]=2 suma=3
    thread 1 suma a[3]=3 suma=6
    thread 1 suma a[4]=4 suma=10

DENTRO DE REGION PARALELA:
    dyn_var: 1
    nthreads_var: 10
    thread_limit_var: 4
    run_sched_var: 1
    Fuera de 'parallel for' suma=10

~/bp3/ejer3
→ export OMP_SCHEDULE="static, 2"

~/bp3/ejer3
→ ./scheduledModificado 5 5

FUERA DE REGION PARALELA:
    dyn_var: 1
    nthreads_var: 10
    thread_limit_var: 4
    run_sched_var: 2
    thread 2 suma a[0]=0 suma=0
    thread 2 suma a[1]=1 suma=1
    thread 2 suma a[2]=2 suma=3
    thread 2 suma a[3]=3 suma=6
    thread 2 suma a[4]=4 suma=10

DENTRO DE REGION PARALELA:
    dyn_var: 1
    nthreads_var: 10
    thread_limit_var: 4
    run_sched_var: 2
    Fuera de 'parallel for' suma=10

```

RESPUESTA: Los valores dentro y fuera de la región paralela son los mismos

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

```

8
9 int main(int argc, char **argv) {
10    int i, n=200,chunk,a[n],suma=0;
11
12    if(argc < 3) {
13        fprintf(stderr,"\\nFalta iteraciones o chunk \\n");
14        exit(-1);
15    }
16
17    n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);
18
19    int num_threads = omp_get_num_threads();
20    int num_procs = omp_get_num_procs();
21    int in_parallel = omp_in_parallel();
22
23
24    for (i=0; i<n; i++) a[i] = i;
25
26    printf("\\nFUERA DE REGION PARALELA:");
27    printf("\\n\\tnum_threads: %d \\
28    \\tnum_procs: %d \\
29    \\tin_parallel: %d \\
30    \\n", num_threads, num_procs, in_parallel);
31
32
33    #pragma omp parallel for firstprivate(suma) \
34        lastprivate(suma) schedule(dynamic,chunk)
35    for (i=0; i<n; i++) {
36        suma = suma + a[i];
37        printf(" thread %d suma a[%d]=%d suma=%d \\n", omp_get_thread_num(),i,a[i],suma);
38    }
39
40    #pragma omp single
41    {
42        num_threads = omp_get_num_threads();
43        num_procs = omp_get_num_procs();
44        in_parallel = omp_in_parallel();
45
46        printf("\\nDENTRO DE REGION PARALELA:");
47        printf("\\n\\tnum_threads: %d \\
48            \\tnum_procs: %d \\
49            \\tin_parallel: %d \\
50            \\n", num_threads, num_procs, in_parallel);
51    }
52
53    printf("Fuera de 'parallel for' suma=%d\\n",suma);
54 }
55

```

CAPTURAS DE PANTALLA:

```
~/bp3/ejer4
$ gcc -O2 -fopenmp scheduled-clauseModificado4.c -o scheduledModificado4

~/bp3/ejer4
$ ./scheduledModificado4 16 2

FUERA DE REGION PARALELA:
    num_threads: 1
    num_procs: 8
    in_parallel: 0
    thread 0 suma a[10]=10 suma=10
    thread 0 suma a[11]=11 suma=21
    thread 7 suma a[8]=8 suma=8
    thread 7 suma a[9]=9 suma=17
    thread 5 suma a[2]=2 suma=2
    thread 5 suma a[3]=3 suma=5
    thread 4 suma a[0]=0 suma=0
    thread 4 suma a[1]=1 suma=1
    thread 6 suma a[14]=14 suma=14
    thread 6 suma a[15]=15 suma=29
    thread 1 suma a[12]=12 suma=12
    thread 1 suma a[13]=13 suma=25
    thread 2 suma a[4]=4 suma=4
    thread 2 suma a[5]=5 suma=9
    thread 3 suma a[6]=6 suma=6
    thread 3 suma a[7]=7 suma=13

DENTRO DE REGION PARALELA:
    num_threads: 1
    num_procs: 8
    in_parallel: 0
    Fuera de 'parallel for' suma=29
```

RESPUESTA: No cambia ninguna respecto a dentro/fuera del parallel

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`

```

16 n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);
17
18 int dyn_var = omp_get_dynamic();
19 int nthreads_var = omp_get_max_threads();
20 int thread_limit_var = omp_get_thread_limit();
21 int run_sched_var;
22 omp_sched_t kind;
23 omp_get_schedule(&kind, &run_sched_var);
24
25 for (i=0; i<n; i++) a[i] = i;
26
27 printf("\nFUERA DE REGION PARALELA, sin modificar:");
28 printf("\n\tdyn_var: %d \
29 \tnthreads_var: %d \
30 \tthread_limit_var: %d \
31 \trun_sched_var: %d \
32 \n", dyn_var, nthreads_var, thread_limit_var, run_sched_var);
33
34 #pragma omp parallel for firstprivate(suma) \
35         lastprivate(suma) schedule(dynamic,chunk)
36 for (i=0; i<n; i++) {
37     suma = suma + a[i];
38     printf(" thread %d suma a[%d]=%d suma=%d \n", omp_get_thread_num(),i,a[i],suma);
39 }
40
41 #pragma omp single
42 {
43     omp_set_dynamic(4);
44     omp_set_num_threads(6);
45     omp_set_schedule(omp_sched_dynamic, 6);
46
47     dyn_var = omp_get_dynamic();
48     nthreads_var = omp_get_max_threads();
49     thread_limit_var = omp_get_thread_limit();
50     omp_get_schedule(&kind, &run_sched_var);
51
52     printf("\nDENTRO DE REGION PARALELA, modificado:");
53     printf("\n\tdyn_var: %d \
54 \tnthreads_var: %d \
55 \tthread_limit_var: %d \
56 \trun_sched_var: %d \
57 \n", dyn_var, nthreads_var, thread_limit_var, run_sched_var);
58 }
59
60
61 printf("Fuerza de 'parallel for' suma=%d\n",suma);
62 }

```

CAPTURAS DE PANTALLA:

```

~/bp3/ejer5
→ gcc -O2 -fopenmp scheduled-clauseModificado5.c -o scheduledModificado5

~/bp3/ejer5
→ ./scheduledModificado5 5 5

FUERA DE REGION PARALELA, sin modificar:
    dyn_var: 0
    nthreads_var: 8
    thread_limit_var: 2147483647
    run_sched_var: 1
    thread 5 suma a[0]=0 suma=0
    thread 5 suma a[1]=1 suma=1
    thread 5 suma a[2]=2 suma=3
    thread 5 suma a[3]=3 suma=6
    thread 5 suma a[4]=4 suma=10

DENTRO DE REGION PARALELA, modificado:
    dyn_var: 1
    nthreads_var: 6
    thread_limit_var: 2147483647
    run_sched_var: 6
    Fuerza de 'parallel for' suma=10

```

RESPUESTA: Las variables se modifican durante la ejecución

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 <time.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4
5 //#define GLOBAL
6 #define DINAMICO
7
8 const int MAX = 4294967295;
9
10 int main(int argc, char** argv){
11     struct timespec cgt1, cgt2;           //Medicion de tiempo
12     double ncgt;
13
14     if(argc < 2) {
15         fprintf(stderr,"Falta el tamaño de la matriz\n");
16         exit(-1);
17     }
18
19     unsigned int n = atoi(argv[1]);
20     int i, j;
21
22     #ifdef GLOBAL
23         printf("\nEjecutando de forma GLOBAL\n");
24         if(n > MAX) {
25             n = MAX;
26         }
27
28         int A[n][n];
29         int B[n];
30         int C[n];
31
32     #endif
33
34     #ifdef DINAMICO
35         printf("\nEjecutando de forma DINAMICA\n");
36         int **A = (int**) malloc(n * sizeof(int*));
37         int *B = (int*) malloc(n * sizeof(int));
38         int *C = (int*) malloc(n * sizeof(int));
39
40         for(int i = 0; i < n; i++) {
41             A[i] = (int*) malloc(n * sizeof(int));
42         }
43     #endif
44
45     for(i = 0; i < n; i++){
46         B[i] = n * (i+i) - 2*i + 1;
47         C[i] = 0;
48
49     }
50
51     ncgt = ncgt / n;
52
53     if(ncgt < 0)
54         ncgt = -ncgt;
55
56     if(ncgt > 1)
57         ncgt = 1;
58
59     if(ncgt < 0)
60         ncgt = -ncgt;
61
62     if(ncgt < 0)
63         ncgt = -ncgt;
64
65     if(ncgt < 0)
66         ncgt = -ncgt;
67
68     if(ncgt < 0)
69         ncgt = -ncgt;
70
71     if(ncgt < 0)
72         ncgt = -ncgt;
73
74     if(ncgt < 0)
75         ncgt = -ncgt;
76
77     if(ncgt < 0)
78         ncgt = -ncgt;
79
80     if(ncgt < 0)
81         ncgt = -ncgt;
82
83     if(ncgt < 0)
84         ncgt = -ncgt;
85
86     if(ncgt < 0)
87         ncgt = -ncgt;
88
89     if(ncgt < 0)
90         ncgt = -ncgt;
91
92     if(ncgt < 0)
93         ncgt = -ncgt;
94
95     if(ncgt < 0)
96         ncgt = -ncgt;
97
98     if(ncgt < 0)
99         ncgt = -ncgt;
100
101     if(ncgt < 0)
102         ncgt = -ncgt;
103
104     if(ncgt < 0)
105         ncgt = -ncgt;
106
107     if(ncgt < 0)
108         ncgt = -ncgt;
109
110     if(ncgt < 0)
111         ncgt = -ncgt;
112
113     if(ncgt < 0)
114         ncgt = -ncgt;
115
116     if(ncgt < 0)
117         ncgt = -ncgt;
118
119     if(ncgt < 0)
120         ncgt = -ncgt;
121
122     if(ncgt < 0)
123         ncgt = -ncgt;
124
125     if(ncgt < 0)
126         ncgt = -ncgt;
127
128     if(ncgt < 0)
129         ncgt = -ncgt;
130
131     if(ncgt < 0)
132         ncgt = -ncgt;
133
134     if(ncgt < 0)
135         ncgt = -ncgt;
136
137     if(ncgt < 0)
138         ncgt = -ncgt;
139
140     if(ncgt < 0)
141         ncgt = -ncgt;
142
143     if(ncgt < 0)
144         ncgt = -ncgt;
145
146     if(ncgt < 0)
147         ncgt = -ncgt;
148
149     if(ncgt < 0)
150         ncgt = -ncgt;
151
152     if(ncgt < 0)
153         ncgt = -ncgt;
154
155     if(ncgt < 0)
156         ncgt = -ncgt;
157
158     if(ncgt < 0)
159         ncgt = -ncgt;
160
161     if(ncgt < 0)
162         ncgt = -ncgt;
163
164     if(ncgt < 0)
165         ncgt = -ncgt;
166
167     if(ncgt < 0)
168         ncgt = -ncgt;
169
170     if(ncgt < 0)
171         ncgt = -ncgt;
172
173     if(ncgt < 0)
174         ncgt = -ncgt;
175
176     if(ncgt < 0)
177         ncgt = -ncgt;
178
179     if(ncgt < 0)
180         ncgt = -ncgt;
181
182     if(ncgt < 0)
183         ncgt = -ncgt;
184
185     if(ncgt < 0)
186         ncgt = -ncgt;
187
188     if(ncgt < 0)
189         ncgt = -ncgt;
190
191     if(ncgt < 0)
192         ncgt = -ncgt;
193
194     if(ncgt < 0)
195         ncgt = -ncgt;
196
197     if(ncgt < 0)
198         ncgt = -ncgt;
199
200     if(ncgt < 0)
201         ncgt = -ncgt;
202
203     if(ncgt < 0)
204         ncgt = -ncgt;
205
206     if(ncgt < 0)
207         ncgt = -ncgt;
208
209     if(ncgt < 0)
210         ncgt = -ncgt;
211
212     if(ncgt < 0)
213         ncgt = -ncgt;
214
215     if(ncgt < 0)
216         ncgt = -ncgt;
217
218     if(ncgt < 0)
219         ncgt = -ncgt;
220
221     if(ncgt < 0)
222         ncgt = -ncgt;
223
224     if(ncgt < 0)
225         ncgt = -ncgt;
226
227     if(ncgt < 0)
228         ncgt = -ncgt;
229
230     if(ncgt < 0)
231         ncgt = -ncgt;
232
233     if(ncgt < 0)
234         ncgt = -ncgt;
235
236     if(ncgt < 0)
237         ncgt = -ncgt;
238
239     if(ncgt < 0)
240         ncgt = -ncgt;
241
242     if(ncgt < 0)
243         ncgt = -ncgt;
244
245     if(ncgt < 0)
246         ncgt = -ncgt;
247
248     if(ncgt < 0)
249         ncgt = -ncgt;
250
251     if(ncgt < 0)
252         ncgt = -ncgt;
253
254     if(ncgt < 0)
255         ncgt = -ncgt;
256
257     if(ncgt < 0)
258         ncgt = -ncgt;
259
260     if(ncgt < 0)
261         ncgt = -ncgt;
262
263     if(ncgt < 0)
264         ncgt = -ncgt;
265
266     if(ncgt < 0)
267         ncgt = -ncgt;
268
269     if(ncgt < 0)
270         ncgt = -ncgt;
271
272     if(ncgt < 0)
273         ncgt = -ncgt;
274
275     if(ncgt < 0)
276         ncgt = -ncgt;
277
278     if(ncgt < 0)
279         ncgt = -ncgt;
280
281     if(ncgt < 0)
282         ncgt = -ncgt;
283
284     if(ncgt < 0)
285         ncgt = -ncgt;
286
287     if(ncgt < 0)
288         ncgt = -ncgt;
289
290     if(ncgt < 0)
291         ncgt = -ncgt;
292
293     if(ncgt < 0)
294         ncgt = -ncgt;
295
296     if(ncgt < 0)
297         ncgt = -ncgt;
298
299     if(ncgt < 0)
300         ncgt = -ncgt;
301
302     if(ncgt < 0)
303         ncgt = -ncgt;
304
305     if(ncgt < 0)
306         ncgt = -ncgt;
307
308     if(ncgt < 0)
309         ncgt = -ncgt;
310
311     if(ncgt < 0)
312         ncgt = -ncgt;
313
314     if(ncgt < 0)
315         ncgt = -ncgt;
316
317     if(ncgt < 0)
318         ncgt = -ncgt;
319
320     if(ncgt < 0)
321         ncgt = -ncgt;
322
323     if(ncgt < 0)
324         ncgt = -ncgt;
325
326     if(ncgt < 0)
327         ncgt = -ncgt;
328
329     if(ncgt < 0)
330         ncgt = -ncgt;
331
332     if(ncgt < 0)
333         ncgt = -ncgt;
334
335     if(ncgt < 0)
336         ncgt = -ncgt;
337
338     if(ncgt < 0)
339         ncgt = -ncgt;
340
341     if(ncgt < 0)
342         ncgt = -ncgt;
343
344     if(ncgt < 0)
345         ncgt = -ncgt;
346
347     if(ncgt < 0)
348         ncgt = -ncgt;
349
350     if(ncgt < 0)
351         ncgt = -ncgt;
352
353     if(ncgt < 0)
354         ncgt = -ncgt;
355
356     if(ncgt < 0)
357         ncgt = -ncgt;
358
359     if(ncgt < 0)
360         ncgt = -ncgt;
361
362     if(ncgt < 0)
363         ncgt = -ncgt;
364
365     if(ncgt < 0)
366         ncgt = -ncgt;
367
368     if(ncgt < 0)
369         ncgt = -ncgt;
370
371     if(ncgt < 0)
372         ncgt = -ncgt;
373
374     if(ncgt < 0)
375         ncgt = -ncgt;
376
377     if(ncgt < 0)
378         ncgt = -ncgt;
379
380     if(ncgt < 0)
381         ncgt = -ncgt;
382
383     if(ncgt < 0)
384         ncgt = -ncgt;
385
386     if(ncgt < 0)
387         ncgt = -ncgt;
388
389     if(ncgt < 0)
390         ncgt = -ncgt;
391
392     if(ncgt < 0)
393         ncgt = -ncgt;
394
395     if(ncgt < 0)
396         ncgt = -ncgt;
397
398     if(ncgt < 0)
399         ncgt = -ncgt;
400
401     if(ncgt < 0)
402         ncgt = -ncgt;
403
404     if(ncgt < 0)
405         ncgt = -ncgt;
406
407     if(ncgt < 0)
408         ncgt = -ncgt;
409
410     if(ncgt < 0)
411         ncgt = -ncgt;
412
413     if(ncgt < 0)
414         ncgt = -ncgt;
415
416     if(ncgt < 0)
417         ncgt = -ncgt;
418
419     if(ncgt < 0)
420         ncgt = -ncgt;
421
422     if(ncgt < 0)
423         ncgt = -ncgt;
424
425     if(ncgt < 0)
426         ncgt = -ncgt;
427
428     if(ncgt < 0)
429         ncgt = -ncgt;
430
431     if(ncgt < 0)
432         ncgt = -ncgt;
433
434     if(ncgt < 0)
435         ncgt = -ncgt;
436
437     if(ncgt < 0)
438         ncgt = -ncgt;
439
440     if(ncgt < 0)
441         ncgt = -ncgt;
442
443     if(ncgt < 0)
444         ncgt = -ncgt;
445
446     if(ncgt < 0)
447         ncgt = -ncgt;
448
449     if(ncgt < 0)
450         ncgt = -ncgt;
451
452     if(ncgt < 0)
453         ncgt = -ncgt;
454
455     if(ncgt < 0)
456         ncgt = -ncgt;
457
458     if(ncgt < 0)
459         ncgt = -ncgt;
460
461     if(ncgt < 0)
462         ncgt = -ncgt;
463
464     if(ncgt < 0)
465         ncgt = -ncgt;
466
467     if(ncgt < 0)
468         ncgt = -ncgt;
469
470     if(ncgt < 0)
471         ncgt = -ncgt;
472
473     if(ncgt < 0)
474         ncgt = -ncgt;
475
476     if(ncgt < 0)
477         ncgt = -ncgt;
478
479     if(ncgt < 0)
480         ncgt = -ncgt;
481
482     if(ncgt < 0)
483         ncgt = -ncgt;
484
485     if(ncgt < 0)
486         ncgt = -ncgt;
487
488     if(ncgt < 0)
489         ncgt = -ncgt;
490
491     if(ncgt < 0)
492         ncgt = -ncgt;
493
494     if(ncgt < 0)
495         ncgt = -ncgt;
496
497     if(ncgt < 0)
498         ncgt = -ncgt;
499
500     if(ncgt < 0)
501         ncgt = -ncgt;
502
503     if(ncgt < 0)
504         ncgt = -ncgt;
505
506     if(ncgt < 0)
507         ncgt = -ncgt;
508
509     if(ncgt < 0)
510         ncgt = -ncgt;
511
512     if(ncgt < 0)
513         ncgt = -ncgt;
514
515     if(ncgt < 0)
516         ncgt = -ncgt;
517
518     if(ncgt < 0)
519         ncgt = -ncgt;
520
521     if(ncgt < 0)
522         ncgt = -ncgt;
523
524     if(ncgt < 0)
525         ncgt = -ncgt;
526
527     if(ncgt < 0)
528         ncgt = -ncgt;
529
530     if(ncgt < 0)
531         ncgt = -ncgt;
532
533     if(ncgt < 0)
534         ncgt = -ncgt;
535
536     if(ncgt < 0)
537         ncgt = -ncgt;
538
539     if(ncgt < 0)
540         ncgt = -ncgt;
541
542     if(ncgt < 0)
543         ncgt = -ncgt;
544
545     if(ncgt < 0)
546         ncgt = -ncgt;
547
548     if(ncgt < 0)
549         ncgt = -ncgt;
550
551     if(ncgt < 0)
552         ncgt = -ncgt;
553
554     if(ncgt < 0)
555         ncgt = -ncgt;
556
557     if(ncgt < 0)
558         ncgt = -ncgt;
559
560     if(ncgt < 0)
561         ncgt = -ncgt;
562
563     if(ncgt < 0)
564         ncgt = -ncgt;
565
566     if(ncgt < 0)
567         ncgt = -ncgt;
568
569     if(ncgt < 0)
570         ncgt = -ncgt;
571
572     if(ncgt < 0)
573         ncgt = -ncgt;
574
575     if(ncgt < 0)
576         ncgt = -ncgt;
577
578     if(ncgt < 0)
579         ncgt = -ncgt;
580
581     if(ncgt < 0)
582         ncgt = -ncgt;
583
584     if(ncgt < 0)
585         ncgt = -ncgt;
586
587     if(ncgt < 0)
588         ncgt = -ncgt;
589
590     if(ncgt < 0)
591         ncgt = -ncgt;
592
593     if(ncgt < 0)
594         ncgt = -ncgt;
595
596     if(ncgt < 0)
597         ncgt = -ncgt;
598
599     if(ncgt < 0)
600         ncgt = -ncgt;
601
602     if(ncgt < 0)
603         ncgt = -ncgt;
604
605     if(ncgt < 0)
606         ncgt = -ncgt;
607
608     if(ncgt < 0)
609         ncgt = -ncgt;
610
611     if(ncgt < 0)
612         ncgt = -ncgt;
613
614     if(ncgt < 0)
615         ncgt = -ncgt;
616
617     if(ncgt < 0)
618         ncgt = -ncgt;
619
620     if(ncgt < 0)
621         ncgt = -ncgt;
622
623     if(ncgt < 0)
624         ncgt = -ncgt;
625
626     if(ncgt < 0)
627         ncgt = -ncgt;
628
629     if(ncgt < 0)
630         ncgt = -ncgt;
631
632     if(ncgt < 0)
633         ncgt = -ncgt;
634
635     if(ncgt < 0)
636         ncgt = -ncgt;
637
638     if(ncgt < 0)
639         ncgt = -ncgt;
640
641     if(ncgt < 0)
642         ncgt = -ncgt;
643
644     if(ncgt < 0)
645         ncgt = -ncgt;
646
647     if(ncgt < 0)
648         ncgt = -ncgt;
649
650     if(ncgt < 0)
651         ncgt = -ncgt;
652
653     if(ncgt < 0)
654         ncgt = -ncgt;
655
656     if(ncgt < 0)
657         ncgt = -ncgt;
658
659     if(ncgt < 0)
660         ncgt = -ncgt;
661
662     if(ncgt < 0)
663         ncgt = -ncgt;
664
665     if(ncgt < 0)
666         ncgt = -ncgt;
667
668     if(ncgt < 0)
669         ncgt = -ncgt;
670
671     if(ncgt < 0)
672         ncgt = -ncgt;
673
674     if(ncgt < 0)
675         ncgt = -ncgt;
676
677     if(ncgt < 0)
678         ncgt = -ncgt;
679
680     if(ncgt < 0)
681         ncgt = -ncgt;
682
683     if(ncgt < 0)
684         ncgt = -ncgt;
685
686     if(ncgt < 0)
687         ncgt = -ncgt;
688
689     if(ncgt < 0)
690         ncgt = -ncgt;
691
692     if(ncgt < 0)
693         ncgt = -ncgt;
694
695     if(ncgt < 0)
696         ncgt = -ncgt;
697
698     if(ncgt < 0)
699         ncgt = -ncgt;
700
701     if(ncgt < 0)
702         ncgt = -ncgt;
703
704     if(ncgt < 0)
705         ncgt = -ncgt;
706
707     if(ncgt < 0)
708         ncgt = -ncgt;
709
710     if(ncgt < 0)
711         ncgt = -ncgt;
712
713     if(ncgt < 0)
714         ncgt = -ncgt;
715
716     if(ncgt < 0)
717         ncgt = -ncgt;
718
719     if(ncgt < 0)
720         ncgt = -ncgt;
721
722     if(ncgt < 0)
723         ncgt = -ncgt;
724
725     if(ncgt < 0)
726         ncgt = -ncgt;
727
728     if(ncgt < 0)
729         ncgt = -ncgt;
730
731     if(ncgt < 0)
732         ncgt = -ncgt;
733
734     if(ncgt < 0)
735         ncgt = -ncgt;
736
737     if(ncgt < 0)
738         ncgt = -ncgt;
739
740     if(ncgt < 0)
741         ncgt = -ncgt;
742
743     if(ncgt < 0)
744         ncgt = -ncgt;
745
746     if(ncgt < 0)
747         ncgt = -ncgt;
748
749     if(ncgt < 0)
750         ncgt = -ncgt;
751
752     if(ncgt < 0)
753         ncgt = -ncgt;
754
755     if(ncgt < 0)
756         ncgt = -ncgt;
757
758     if(ncgt < 0)
759         ncgt = -ncgt;
760
761     if(ncgt < 0)
762         ncgt = -ncgt;
763
764     if(ncgt < 0)
765         ncgt = -ncgt;
766
767     if(ncgt < 0)
768         ncgt = -ncgt;
769
770     if(ncgt < 0)
771         ncgt = -ncgt;
772
773     if(ncgt < 0)
774         ncgt = -ncgt;
775
776     if(ncgt < 0)
777         ncgt = -ncgt;
778
779     if(ncgt < 0)
780         ncgt = -ncgt;
781
782     if(ncgt < 0)
783         ncgt = -ncgt;
784
785     if(ncgt < 0)
786         ncgt = -ncgt;
787
788     if(ncgt < 0)
789         ncgt = -ncgt;
790
791     if(ncgt < 0)
792         ncgt = -ncgt;
793
794     if(ncgt < 0)
795         ncgt = -ncgt;
796
797     if(ncgt < 0)
798         ncgt = -ncgt;
799
800     if(ncgt < 0)
801         ncgt = -ncgt;
802
803     if(ncgt < 0)
804         ncgt = -ncgt;
805
806     if(ncgt < 0)
807         ncgt = -ncgt;
808
809     if(ncgt < 0)
810         ncgt = -ncgt;
811
812     if(ncgt < 0)
813         ncgt = -ncgt;
814
815     if(ncgt < 0)
816         ncgt = -ncgt;
817
818     if(ncgt < 0)
819         ncgt = -ncgt;
820
821     if(ncgt < 0)
822         ncgt = -ncgt;
823
824     if(ncgt < 0)
825         ncgt = -ncgt;
826
827     if(ncgt < 0)
828         ncgt = -ncgt;
829
830     if(ncgt < 0)
831         ncgt = -ncgt;
832
833     if(ncgt < 0)
834         ncgt = -ncgt;
835
836     if(ncgt < 0)
837         ncgt = -ncgt;
838
839     if(ncgt < 0)
840         ncgt = -ncgt;
841
842     if(ncgt < 0)
843         ncgt = -ncgt;
844
845     if(ncgt < 0)
846         ncgt = -ncgt;
847
848     if(ncgt < 0)
849         ncgt = -ncgt;
850
851     if(ncgt < 0)
852         ncgt = -ncgt;
853
854     if(ncgt < 0)
855         ncgt = -ncgt;
856
857     if(ncgt < 0)
858         ncgt = -ncgt;
859
860     if(ncgt < 0)
861         ncgt = -ncgt;
862
863     if(ncgt < 0)
864         ncgt = -ncgt;
865
866     if(ncgt < 0)
867         ncgt = -ncgt;
868
869     if(ncgt < 0)
870         ncgt = -ncgt;
871
872     if(ncgt < 0)
873         ncgt = -ncgt;
874
875     if(ncgt < 0)
876         ncgt = -ncgt;
877
878     if(ncgt < 0)
879         ncgt = -ncgt;
880
881     if(ncgt < 0)
882         ncgt = -ncgt;
883
884     if(ncgt < 0)
885         ncgt = -ncgt;
886
887     if(ncgt < 0)
888         ncgt = -ncgt;
889
890     if(ncgt < 0)
891         ncgt = -ncgt;
892
893     if(ncgt < 0)
894         ncgt = -ncgt;
895
896     if(ncgt < 0)
897         ncgt = -ncgt;
898
899     if(ncgt < 0)
900         ncgt = -ncgt;
901
902     if(ncgt < 0)
903         ncgt = -ncgt;
904
905     if(ncgt < 0)
906         ncgt = -ncgt;
907
908     if(ncgt < 0)
909         ncgt = -ncgt;
910
911     if(ncgt < 0)
912         ncgt = -ncgt;
913
914     if(ncgt < 0)
915         ncgt = -ncgt;
916
917     if(ncgt < 0)
918         ncgt = -ncgt;
919
920     if(ncgt < 0)
921         ncgt = -ncgt;
922
923     if(ncgt < 0)
924         ncgt = -ncgt;
925
926     if(ncgt < 0)
927         ncgt = -ncgt;
928
929     if(ncgt < 0)
930         ncgt = -ncgt;
931
932     if(ncgt < 0)
933         ncgt = -ncgt;
934
935     if(ncgt < 0)
936         ncgt = -ncgt;
937
938     if(ncgt < 0)
939         ncgt = -ncgt;
940
941     if(ncgt < 0)
942         ncgt = -ncgt;
943
944     if(ncgt < 0)
945         ncgt = -ncgt;
946
947     if(ncgt < 0)
948         ncgt = -ncgt;
949
950     if(ncgt < 0)
951         ncgt = -ncgt;
952
953     if(ncgt < 0)
954         ncgt = -ncgt;
955
956     if(ncgt < 0)
957         ncgt = -ncgt;
958
959     if(ncgt < 0)
960         ncgt = -ncgt;
961
962     if(ncgt < 0)
963         ncgt = -ncgt;
964
965     if(ncgt < 0)
966         ncgt = -ncgt;
967
968     if(ncgt < 0)
969         ncgt = -ncgt;
970
971     if(ncgt < 0)
972         ncgt = -ncgt;
973
974     if(ncgt < 0)
975         ncgt = -ncgt;
976
977     if(ncgt < 0)
978         ncgt = -ncgt;
979
980     if(ncgt < 0)
981         ncgt = -ncgt;
982
983     if(ncgt < 0)
984         ncgt = -ncgt;
985
986     if(ncgt < 0)
987         ncgt = -ncgt;
988
989     if(ncgt < 0)
990         ncgt = -ncgt;
991
992     if(ncgt < 0)
993         ncgt = -ncgt;
994
995     if(ncgt < 0)
996         ncgt = -ncgt;
997
998     if(ncgt < 0)
999         ncgt = -ncgt;
999
1000    if(ncgt < 0)
1001        ncgt = -ncgt;
1002
1003    if(ncgt < 0)
1004        ncgt = -ncgt;
1005
1006    if(ncgt < 0)
1007        ncgt = -ncgt;
1008
1009    if(ncgt < 0)
1010        ncgt = -ncgt;
1011
1012    if(ncgt < 0)
1013        ncgt = -ncgt;
1014
1015    if(ncgt < 0)
1016        ncgt = -ncgt;
1017
1018    if(ncgt < 0)
1019        ncgt = -ncgt;
1020
1021    if(ncgt < 0)
1022        ncgt = -ncgt;
1023
1024    if(ncgt < 0)
1025        ncgt = -ncgt;
1026
1027    if(ncgt < 0)
1028        ncgt = -ncgt;
1029
1030    if(ncgt < 0)
1031        ncgt = -ncgt;
1032
1033    if(ncgt < 0)
1034        ncgt = -ncgt;
1035
1036    if(ncgt < 0)
1037        ncgt = -ncgt;
1038
1039    if(ncgt < 0)
1040        ncgt = -ncgt;
1041
1042    if(ncgt < 0)
1043        ncgt = -ncgt;
1044
1045    if(ncgt < 0)
1046        ncgt = -ncgt;
1047
1048    if(ncgt < 0)
1049        ncgt = -ncgt;
1050
1051    if(ncgt < 0)
1052        ncgt = -ncgt;
1053
1054    if(ncgt < 0)
1055        ncgt = -ncgt;
1056
1057    if(ncgt < 0)
1058        ncgt = -ncgt;
1059
1060    if(ncgt < 0)
1061        ncgt = -ncgt;
1062
1063    if(ncgt < 0)
1064        ncgt = -ncgt;
1065
1066    if(ncgt < 0)
1067        ncgt = -ncgt;
1068
1069    if(ncgt < 0)
1070        ncgt = -ncgt;
1071
1072    if(ncgt < 0)
1073        ncgt = -ncgt;
1074
1075    if(ncgt < 0)
1076        ncgt = -ncgt;
1077
1078    if(ncgt < 0)
1079        ncgt = -ncgt;
1080
1081    if(ncgt < 0)
1082        ncgt = -ncgt;
1083
1084    if(ncgt < 0)
1085        ncgt = -ncgt;
1086
1087    if(ncgt < 0)
1088        ncgt = -ncgt;
1089
1090    if(ncgt < 0)
1091        ncgt = -ncgt;
1092
1093    if(ncgt < 0)
1094        ncgt = -ncgt;
1095
1096    if(ncgt < 0)
1097        ncgt = -ncgt;
1098
1099    if(ncgt < 0)
1100        ncgt = -ncgt;
1101
1102    if(ncgt < 0)
1103        ncgt = -ncgt;
1104
1105    if(ncgt < 0)
1106        ncgt = -ncgt;
1107
1108    if(ncgt < 0)
1109        ncgt = -ncgt;
1110
1111    if(ncgt < 0)
1112        ncgt = -ncgt;
1113
1114    if(ncgt < 0)
1115        ncgt = -ncgt;
1116
1117    if(ncgt < 0)
1118        ncgt = -ncgt;
1119
1120    if(ncgt < 0)
1121        ncgt = -ncgt;
1122
1123    if(ncgt < 0)
1124        ncgt = -ncgt;
1125
1126    if(ncgt < 0)
1127        ncgt = -ncgt;
1128
1129    if(ncgt < 0)
1130        ncgt = -ncgt;
1131
1132    if(ncgt < 0)
1133        ncgt = -ncgt;
1134
1135    if(ncgt < 0)
1136        ncgt = -ncgt;
1137
1138    if(ncgt < 0)
1139        ncgt = -ncgt;
1140
1141    if(ncgt < 0)
1142        ncgt = -ncgt;
1143
11
```

```

48     for(j = i; j < n; j++){
49         A[i][j] = n * (2*j) - i + 1;
50     }
51 }
52
53
54 if(n<=11){
55     printf("\nMatriz inicial:\n");
56     for(i = 0; i < n; i++){
57         for(j = 0; j < n; j++){
58             printf(" %d ", A[i][j]);
59         }
60         printf("\n");
61     }
62
63
64     printf("\nVector inicial: |");
65     for(i = 0; i < n; i++){
66         printf(" %d |", B[i]);
67     }
68 }
69
70 clock_gettime(CLOCK_REALTIME,&cgt1);
71
72 for(i = 0; i < n; i++) {
73     for(j = i; j < n; j++) {
74         C[i] += A[i][j] * B[j];
75     }
76 }
77
78 clock_gettime(CLOCK_REALTIME,&cgt2);
79
80 ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-cgt1.tv_nsec) / (1.e+9));
81
82 printf("\nTiempo matriz tamanio %d: %11.9f", n, ncgt);
83
84 if(n<=11){
85     printf("\nVector resultado: |");
86     for(i = 0; i < n; i++){
87         printf(" %d |", C[i]);
88     }
89 }
90
91 #ifdef DYNAMIC
92     for(int i = 0; i < n; i++){
93         free(A[i]);
94     }
95 }

```

C ▾ Tab Width: 3 ▾ Ln 74, Col 31 ▾ INS

CAPTURAS DE PANTALLA:

```

~/bp3/ejer6
→ gcc -O2 pmtv-secuencial.c -o pmtv-sec

~/bp3/ejer6
→ ./pmtv-sec 4

Ejecutando de forma DINAMICA

Matriz inicial:
 1  9  17  25
 0  8  16  24
 0  0  15  23
 0  0  0   22

Vector inicial: | 5 | 7 | 9 | 11 |
Tiempo matriz tamanio 4: 0.0000000131

```

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.

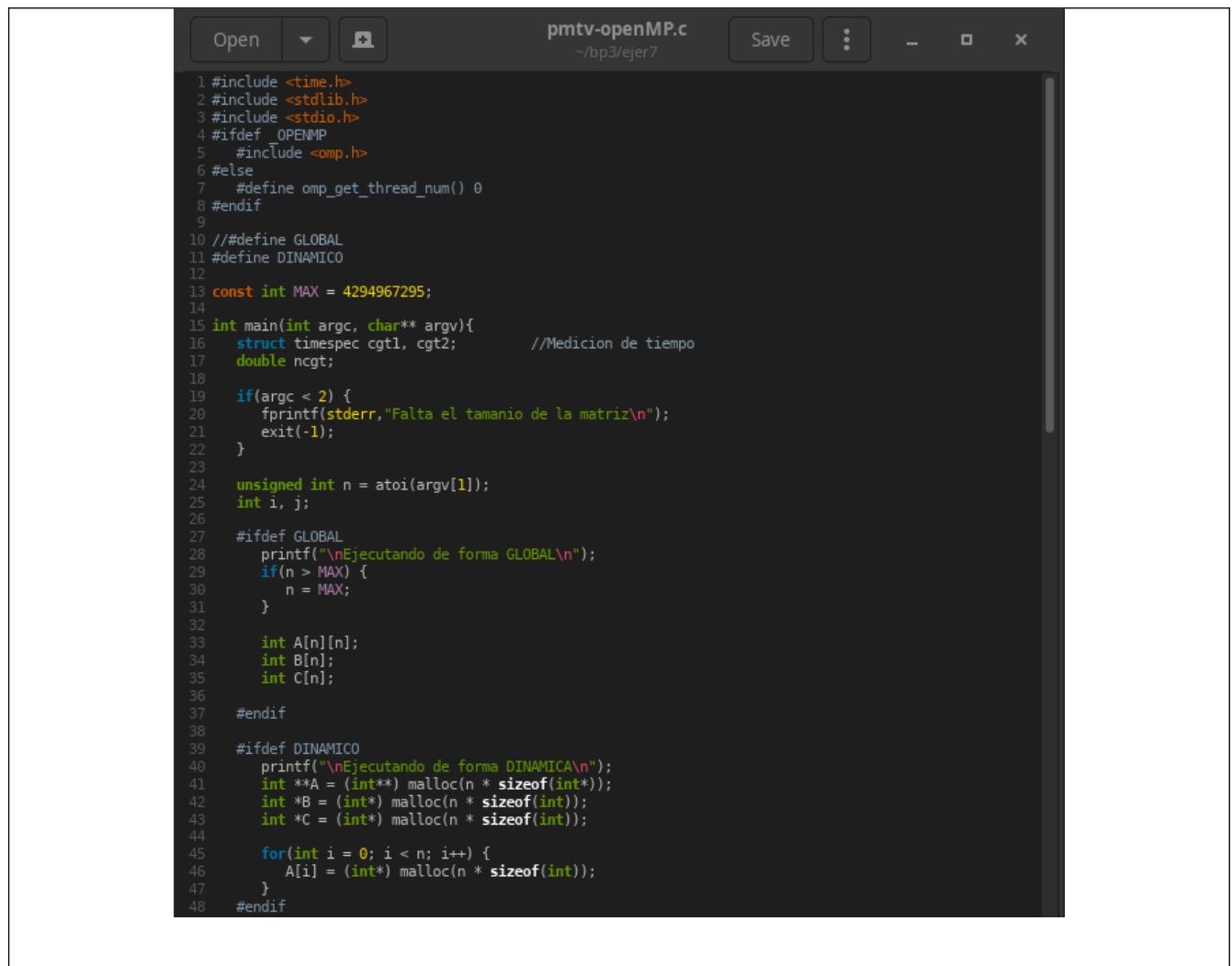
Conteste a las siguientes preguntas: (a) ¿Qué valor por defecto usa OpenMP para chunk con `static`, `dynamic` y `guided`? Indique qué ha hecho para obtener este valor por defecto para cada alternativa. (b) ¿Qué número de operaciones de multiplicación y suma realizan cada uno de los threads en la asignación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

RESPUESTAS:

A) Se obtiene con `omp_get_schedule`. Es 1 para `dynamic` y `guided`; y 0 para `static`.

C) Que no será equitativo. En `dynamic` la hebra más rápida ejecutará más, y en `guided` la primera que ejecute se llevará un bloque mucho más grande

CAPTURA CÓDIGO FUENTE: pmtv-OpenMP.c



```
1 #include <time.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #ifndef OPENMP
5     #include <omp.h>
6 #else
7     #define omp_get_thread_num() 0
8 #endif
9
10 // #define GLOBAL
11 #define DINAMICO
12
13 const int MAX = 4294967295;
14
15 int main(int argc, char** argv){
16     struct timespec cgt1, cgt2;           //Medicion de tiempo
17     double ncgt;
18
19     if(argc < 2) {
20         fprintf(stderr,"Falta el tamano de la matriz\n");
21         exit(-1);
22     }
23
24     unsigned int n = atoi(argv[1]);
25     int i, j;
26
27     #ifdef GLOBAL
28         printf("\nEjecutando de forma GLOBAL\n");
29         if(n > MAX) {
30             n = MAX;
31         }
32
33         int A[n][n];
34         int B[n];
35         int C[n];
36
37     #endif
38
39     #ifdef DINAMICO
40         printf("\nEjecutando de forma DINAMICA\n");
41         int **A = (int**) malloc(n * sizeof(int*));
42         int *B = (int*) malloc(n * sizeof(int));
43         int *C = (int*) malloc(n * sizeof(int));
44
45         for(int i = 0; i < n; i++) {
46             A[i] = (int*) malloc(n * sizeof(int));
47         }
48     #endif
```

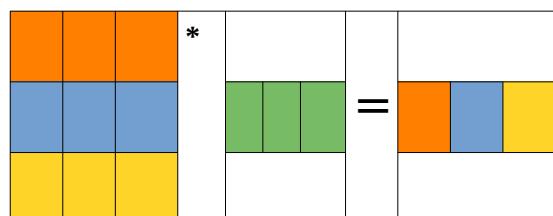
```

49  #pragma omp parallel for
50  for(i = 0; i < n; i++){
51      B[i] = n * (1+i) - 2*i + 1;
52      C[i] = 0;
53      for(j = i; j < n; j++){
54          A[i][j] = n * (2*j) - i + 1;
55      }
56  }
57 }
58
59
60 if(n<=11){
61     printf("\nMatriz inicial:\n");
62     for(i = 0; i < n; i++){
63         for(j = 0; j < n; j++){
64             printf(" %d ", A[i][j]);
65         }
66         printf("\n");
67     }
68
69
70     printf("\nVector inicial: |");
71     for(i = 0; i < n; i++){
72         printf(" %d |", B[i]);
73     }
74 }
75
76 clock_gettime(CLOCK_REALTIME,&cgt1);
77 |
78
79 #pragma omp parallel shared(A, B, C) private(i, j)
80 {
81     #pragma omp for
82     for(i = 0; i < n; i++) {
83         for(j = i; j < n; j++) {
84             C[i] += A[i][j] * B[j];
85         }
86     }
87 }
88
89 clock_gettime(CLOCK_REALTIME,&cgt2);
90
91 ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-cgt1.tv_nsec) / (1.e+9));
92
93 printf("\nTiempo matriz tamano %d: %11.9f", n, ncgt);
94
95 if(n<=11){
96     printf("\nVector resultado: |");

```

C ▾ Tab Width: 3 ▾ Ln 77, Col 4 ▾ INS

DESCOMPOSICIÓN DE DOMINIO: Cada fila de la matriz (A) se asigna a una hebra. A cada fila de la matriz le corresponde una posición en el vector solución (C).



CAPTURAS DE PANTALLA:

```
~/bp3/ejer7
→ gcc -O2 -fopenmp pmtv-openMP.c -o pmtv-open

~/bp3/ejer7
→ ./pmtv-open 4

Ejecutando de forma DINAMICA

Matriz inicial:
 1  9  17  25
 0  8  16  24
 0  0  15  23
 0  0  0  22

Vector inicial: | 5 | 7 | 9 | 11 |
Tiempo matriz tamaño 4: 0.000001784
Vector resultado: | 496 | 464 | 388 | 242 |%
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

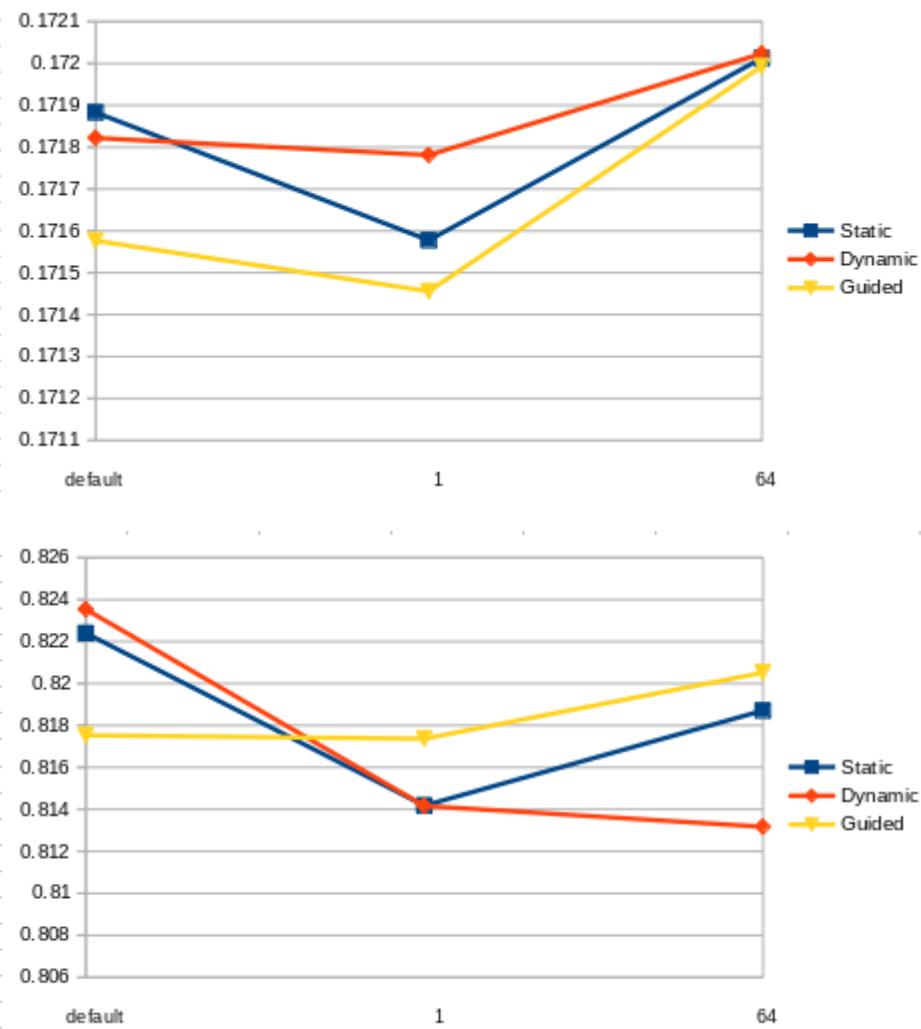
SCRIPT: pmvt-OpenMP_atcgrid.sh

```
#!/bin/bash
# Id. usuario del trabajo: $SLURM_JOB_USER
# Id. del trabajo: $SLURM_JOBID
# Nombre del trabajo especificado por usuario: $SLURM_JOB_NAME
# Directorio de trabajo (en el que se ejecuta el script): $SLURM_SUBMIT_DIR
# Cola: $SLURM_JOB_PARTITION
# Nodo que ejecuta este trabajo: $SLURM_SUBMIT_HOST
# No de nodos asignados al trabajo: $SLURM_JOB_NUM_NODES
# Nodos asignados al trabajo: $SLURM_JOB_NODELIST
# CPUs por nodo: $SLURM_JOB_CPUS_PER_NODE
# N=16000
# 
# 
# STATIC
# export OMP_SCHEDULE="static"
# ./pmtv-open $N
# export OMP_SCHEDULE="static,1"
# ./pmtv-open $N
# export OMP_SCHEDULE="static,64"
# ./pmtv-open $N
# 
# 
# DYNAMIC
# export OMP_SCHEDULE="dynamic"
# ./pmtv-open $N
# export OMP_SCHEDULE="dynamic,1"
# ./pmtv-open $N
# export OMP_SCHEDULE="dynamic,64"
# ./pmtv-open $N
# 
# 
# GUIDED
# export OMP_SCHEDULE="guided"
# ./pmtv-open $N
# export OMP_SCHEDULE="guided,1"
# ./pmtv-open $N
# export OMP_SCHEDULE="guided,64"
# ./pmtv-open $N
# 
# 
# N=36000
# 
# 
# STATIC
# export OMP_SCHEDULE="static"
# ./pmtv-open $N
# export OMP_SCHEDULE="static,1"
# ./pmtv-open $N
# export OMP_SCHEDULE="static,64"
# ./pmtv-open $N
# 
# 
# DYNAMIC
# export OMP_SCHEDULE="dynamic"
# ./pmtv-open $N
# export OMP_SCHEDULE="dynamic,1"
# ./pmtv-open $N
# export OMP_SCHEDULE="dynamic,64"
# ./pmtv-open $N
# 
# 
# GUIDED
# export OMP_SCHEDULE="guided"
# ./pmtv-open $N
# export OMP_SCHEDULE="guided,1"
# ./pmtv-open $N
# export OMP_SCHEDULE="guided,64"
# ./pmtv-open $N
```

Tabla 3 .Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño N=16000, 36000 , 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0.171882813	0.171822063	0.171576714
1	0.171577729	0.171781251	0.171455927
64	0.172012578	0.172023527	0.171993181

Chunk	Static	Dynamic	Guided
por defecto	0.822397772	0.823533586	0.817531695
1	0.814180432	0.814166323	0.817366535
64	0.818708209	0.813170908	0.820519730

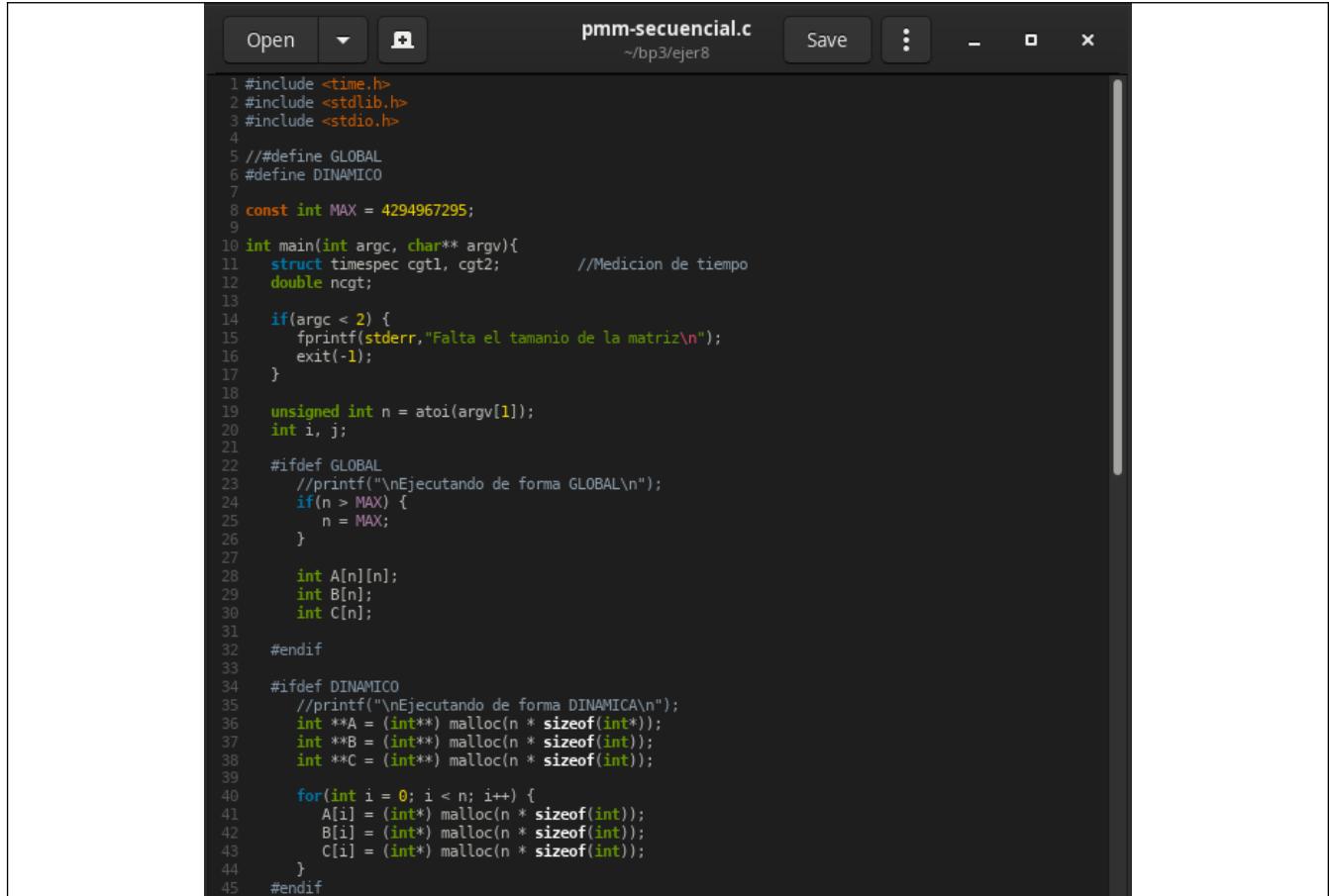


8. Implementar un programa secuencial en C que calcule la multiplicación de matrices cuadradas, B y C:

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

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 <time.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4
5 // #define GLOBAL
6 #define DINAMICO
7
8 const int MAX = 4294967295;
9
10 int main(int argc, char** argv){
11     struct timespec cgt1, cgt2;           //Medicion de tiempo
12     double ncgt;
13
14     if(argc < 2) {
15         fprintf(stderr,"Falta el tamaño de la matriz\n");
16         exit(-1);
17     }
18
19     unsigned int n = atoi(argv[1]);
20     int i, j;
21
22     #ifdef GLOBAL
23         //printf("\nEjecutando de forma GLOBAL\n");
24         if(n > MAX) {
25             n = MAX;
26         }
27
28         int A[n][n];
29         int B[n];
30         int C[n];
31
32     #endif
33
34     #ifdef DINAMICO
35         //printf("\nEjecutando de forma DINAMICA\n");
36         int **A = (int**) malloc(n * sizeof(int));
37         int **B = (int**) malloc(n * sizeof(int));
38         int **C = (int**) malloc(n * sizeof(int));
39
40         for(int i = 0; i < n; i++) {
41             A[i] = (int*) malloc(n * sizeof(int));
42             B[i] = (int*) malloc(n * sizeof(int));
43             C[i] = (int*) malloc(n * sizeof(int));
44         }
45     #endif

```

```

47     for(i = 0; i < n; i++){
48         C[i] = 0;
49         for(j = i; j < n; j++){
50             A[i][j] = n * (2*j) - i + 1;
51             B[i][j] = n * (2*i) - j + 1;
52         }
53     }
54
55     if(n<=11){
56         printf("\nMatriz A:\n");
57         for(i = 0; i < n; i++){
58             for(j = 0; j < n; j++){
59                 printf(" %d ", A[i][j]);
60             }
61             printf("\n");
62         }
63     }
64
65     printf("\nMatriz B:\n");
66     for(i = 0; i < n; i++){
67         for(j = 0; j < n; j++){
68             printf(" %d ", B[i][j]);
69         }
70         printf("\n");
71     }
72 }
73
74 clock_gettime(CLOCK_REALTIME,&cgt1);
75
76 for(i = 0; i < n; i++) {
77     for(j = 0; j < n; j++) {
78         for(k = 0; k < n; k++){
79             C[i][j] += A[i][k] * B[k][j];
80         }
81     }
82 }
83
84 clock_gettime(CLOCK_REALTIME,&cgt2);
85
86 ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-cgt1.tv_nsec) / (1.e+9));
87
88 printf("\nTiempo matriz tamaño %d: %11.9f", n, ncgt);
89
90 if(n<=11){
91     printf("\nMatriz resultado:\n");
92     for(i = 0; i < n; i++){
93         for(j = 0; j < n; j++){
94

```

C ▾ Tab Width: 3 ▾ Ln 105, Col 21 ▾ INS

CAPTURAS DE PANTALLA:

```

~/bp3/ejer8
→ gcc -O2 pmm-secuencial.c -o pmm-sec

~/bp3/ejer8
→ ./pmm-sec 4

Matriz A:
 5  7  9  11
 4  6  8  10
 3  5  7  9
 2  4  6  8

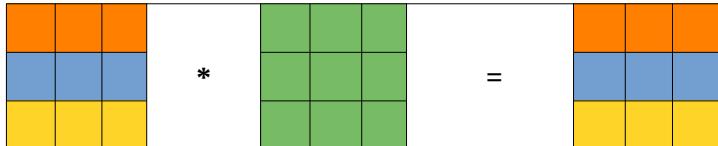
Matriz B:
 5  7  9  11
 4  6  8  10
 3  5  7  9
 2  4  6  8

Tiempo matriz tamaño 4: 0.0000001147
Matriz resultado:
 102 166 230 294
 88 144 200 256
 74 122 170 218
 60 100 140 180

```

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 parallelizar 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).

DESCOMPOSICIÓN DE DOMINIO:



CAPTURA CÓDIGO FUENTE: pmm-OpenMP.c

```

  Open  Save  pmm-openMP.c
  ~~/bp3/ejer9
1 #include <time.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #ifndef _OPENMP
5 #include <omp.h>
6 #else
7 #define omp_get_thread_num() 0
8 #endif
9
10
11 //#define GLOBAL
12 #define DINAMICO
13
14 const int MAX = 4294967295;
15
16 int main(int argc, char** argv){
17     struct timespec cgt1, cgt2;           //Medicion de tiempo
18     double ncgt;
19
20     if(argc < 2) {
21         fprintf(stderr,"Falta el tamaño de la matriz\n");
22         exit(-1);
23     }
24
25     unsigned int n = atoi(argv[1]);
26     int i, j;
27
28     #ifdef GLOBAL
29         //printf("\nEjecutando de forma GLOBAL\n");
30         if(n > MAX) {
31             n = MAX;
32         }
33
34         int A[n][n];
35         int B[n][n];
36         int C[n][n];
37
38     #endif
39
40     #ifdef DINAMICO
41         //printf("\nEjecutando de forma DINAMICA\n");
42         int **A = (int**) malloc(n * sizeof(int*));
43         int **B = (int**) malloc(n * sizeof(int*));
44         int **C = (int**) malloc(n * sizeof(int*));
45
46         for(int i = 0; i < n; i++) {
47             A[i] = (int*) malloc(n * sizeof(int));
48             B[i] = (int*) malloc(n * sizeof(int));
49             C[i] = (int*) malloc(n * sizeof(int));
50
51         }
52
53     #endif
54
55     //Inicilizar matrices
56
57     //Multiplicación
58
59     //Almacenar resultado
60
61     //Libertar memoria
62
63     //Medición de tiempo
64
65     //Imprimir resultado
66
67     //Finalizar
68 }
```

```

51  #endif
52
53 #pragma omp parallel for
54 for(i = 0; i < n; i++){
55     for(j = 0; j < n; j++){
56         A[i][j] = n + (2*j) - i + 1;
57         B[i][j] = n + (2*j) - i + 1;
58         C[i][j] = 0;
59     }
60 }
61
62
63 if(n<=11){
64     printf("\nMatriz A:\n");
65     for(i = 0; i < n; i++){
66         for(j = 0; j < n; j++){
67             printf(" %d ", A[i][j]);
68         }
69         printf("\n");
70     }
71
72
73     printf("\nMatriz B:\n");
74     for(i = 0; i < n; i++){
75         for(j = 0; j < n; j++){
76             printf(" %d ", B[i][j]);
77         }
78         printf("\n");
79     }
80 }
81
82 clock_gettime(CLOCK_REALTIME,&cgt1);
83
84 int k;
85
86 #pragma omp parallel shared(A, B, C) private(i, j, k)
87 {
88     #pragma omp for
89     for(i = 0; i < n; i++) {
90         for(j = 0; j < n; j++) {
91             for(int k = 0; k < n; k++){
92                 C[i][j] += A[i][k] * B[k][j];
93             }
94         }
95     }
96 }
97
98 clock_gettime(CLOCK_REALTIME,&cgt2);
99 ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-cgt1.tv_nsec) / (1.e+9));
100
101 printf("\nTiempo matriz tamaño %d: %11.9f", n, ncgt);
102
103 if(n<=11){
104     printf("\nMatriz resultado:\n");
105     for(i = 0; i < n; i++){
106         for(j = 0; j < n; j++){
107             printf(" %d ", C[i][j]);
108         }
109         printf("\n");
110     }
111 }
112
113
114 #ifdef DYNAMIC
115     for(int i = 0; i < n; i++){
116         free(A[i]);
117         free(B[i]);
118         free(C[i]);
119     }
120
121     free(A);
122     free(B);
123     free(C);
124 #endif
125
126 return 0;
127 }

```

C ▼ Tab Width: 3 ▼ Ln 84, Col 8 ▼ INS

CAPTURAS DE PANTALLA:

```
~/bp3/ejer9
→ gcc -O2 -fopenmp pmm-openMP.c -o pmm-open

~/bp3/ejer9
→ ./pmm-open 4

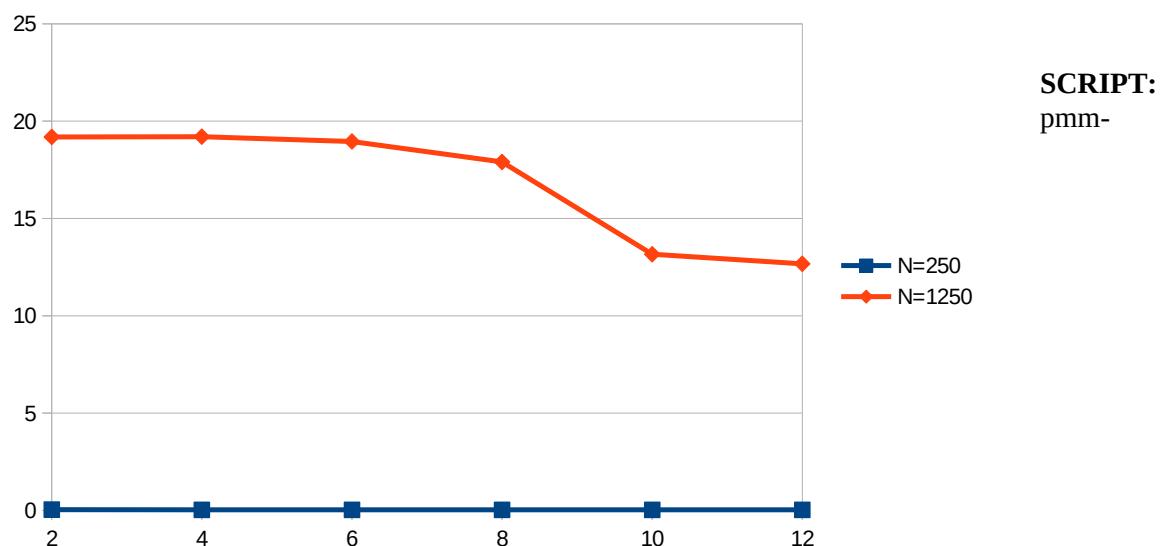
Matriz A:
5 7 9 11
4 6 8 10
3 5 7 9
2 4 6 8

Matriz B:
5 7 9 11
4 6 8 10
3 5 7 9
2 4 6 8

Tiempo matriz tamaño 4: 0.000001955
Matriz resultado:
102 166 230 294
88 144 200 256
74 122 170 218
60 100 140 180
```

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 -02 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:



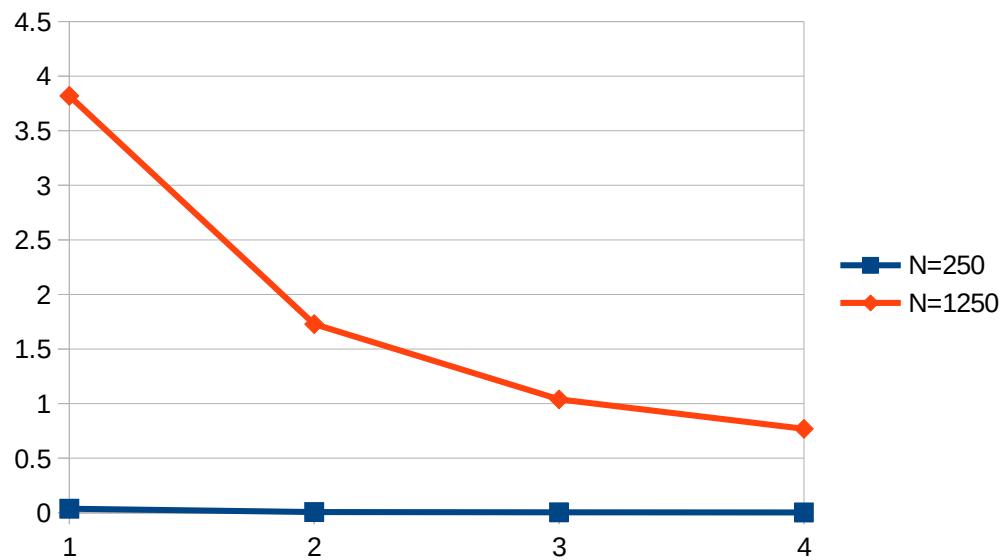
OpenMP_atcgrid.sh

```

1#!/bin/bash
2echo "Id. usuario del trabajo: $SLURM_JOB_USER"
3echo "Id. del trabajo: $SLURM_JOBID"
4echo "Nombre del trabajo especificado por usuario: $SLURM_JOB_NAME"
5echo "Directorio de trabajo (en el que se ejecuta el script):"
6$SLURM_SUBMIT_DIR
7echo "Cola: $SLURM_JOB_PARTITION"
8echo "Nodo que ejecuta este trabajo:$SLURM_SUBMIT_HOST"
9echo "No de nodos asignados al trabajo: $SLURM_JOB_NUM_NODES"
10echo "Nodos asignados al trabajo: $SLURM_JOB_NODELIST"
11echo "CPUs por nodo: $SLURM_JOB_CPUS_PER_NODE"
12
13echo "n = 250" >> salidaatc
14for ((N=2;N<13;N=N+2))
15do
16    export OMP_NUM_THREADS=$N
17    echo "threads = $N" >> salidaatc
18    srun ./pmm-open 250 >> salidaatc
19    echo "" >> salidaatc
20done
21
22echo "n = 1250">> salidaatc
23for ((N=2;N<13;N=N+2))
24do
25    export OMP_NUM_THREADS=$N
26    echo "threads = $N" >> salidaatc
27    srun ./pmm-open 1250 >> salidaatc
28    echo "" >> salidaatc
29done

```

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:



SCRIPT: pmm-OpenMP_pclocal.sh

```
1 #!/bin/bash
2
3 echo "N = 250"
4 for ((N=1;N<5;N=N+1))
5 do
6     export OMP_NUM_THREADS=$N
7     echo "threads = $N"
8     ./pmm-open 250
9     echo ""
10 done
11
12 echo ""
13
14 echo "tamano = 1250"
15 for ((N=1;N<5;N=N+1))
16 do
17     export OMP_NUM_THREADS=$N
18     echo "threads = $N"
19     ./pmm-open 1250
20     echo ""
21 done
```