2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

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

Estudiante (nombre y apellidos): Rodrigo Raya Castellano
Grupo de prácticas: 2
Fecha de entrega:
Fecha evaluación en clase:

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.

CÓDIGO FUENTE: `if-clauseModificado.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main(int argc, char **argv)
{
 int i, n=20, tid, x;
 int a[n],suma=0,sumalocal;
 if(argc < 3) {
  fprintf(stderr,"[ERROR]-Falta iteraciones o Faltan num_threads\n");
  exit(-1);
 }
 n = atoi(argv[1]); if (n>20) n=20;
 x = atoi(argv[2]);

 for (i=0; i<n; i++) {
  a[i] = i;
 }
 #pragma omp parallel if(n>4) default(none) \
 private(sumalocal,tid) shared(a,suma,n) num_threads(x)
 { sumalocal=0;
   tid=omp_get_thread_num();
   #pragma omp for private(i) schedule(static) nowait
   for (i=0; i<n; i++)
   { sumalocal += a[i];
     printf(" thread %d suma de a[%d]=%d sumalocal=%d
\n",tid,i,a[i],sumalocal);
   }
   #pragma omp atomic
    suma += sumalocal;
   #pragma omp barrier
   #pragma omp master
    printf("thread master=%d imprime suma=%d\n",tid,suma);
 }
}
```

CAPTURAS DE PANTALLA:

```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ ./if-clause 4 8
 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 0 suma de a[3]=3 sumalocal=6
thread master=0 imprime suma=6
```

```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ ./if-clause 5 8
 thread 2 suma de a[2]=2 sumalocal=2
 thread 1 suma de a[1]=1 sumalocal=1
 thread 3 suma de a[3]=3 sumalocal=3
 thread 4 suma de a[4]=4 sumalocal=4
 thread 0 suma de a[0]=0 sumalocal=0
thread master=0 imprime suma=10
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ ./if-clause 5 4
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 2 suma de a[3]=3 sumalocal=3
 thread 3 suma de a[4]=4 sumalocal=4
 thread 1 suma de a[2]=2 sumalocal=2
thread master=0 imprime suma=10
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ ./if-clause 5 5
 thread 3 suma de a[3]=3 sumalocal=3
 thread 0 suma de a[0]=0 sumalocal=0
 thread 4 suma de a[4]=4 sumalocal=4
 thread 1 suma de a[1]=1 sumalocal=1
 thread 2 suma de a[2]=2 sumalocal=2
thread master=0 imprime suma=10
```

RESPUESTA:

En la primera captura tenemos que como el número de iteraciones no supera 4 la región paralela la ejecuta un único thread. En la captura posterior sí que se realiza en paralelo al haber suficientes threads y podemos fijar de distintas formas el número de threads que realizan la tarea.

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-claused.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	1	0	0	0	0	0
2	0	1	0	0	1	0	0	0	0
3	1	1	0	0	1	0	0	0	0
4	0	0	1	0	0	1	0	0	0
5	1	0	1	0	0	1	0	0	0
6	0	1	1	0	1	1	0	0	0
7	1	1	1	0	1	1	0	0	0
8	0	0	0	0	1	0	1	1	1
9	1	0	0	0	1	0	1	1	1
10	0	1	0	0	1	0	1	1	1
11	1	1	0	0	1	0	1	1	1
12	0	0	1	0	1	1	0	0	0
13	1	0	1	0	1	1	0	0	0
14	0	1	1	0	1	1	1	0	0
15	1	1	1	0	1	1	1	0	

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

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

RESPUESTA:

Con static tenemos una asignación round-robin de bloques continuos de iteraciones del tamaño del chunk. Este comportamiento es mucho más uniforme que el dynamic y el guided.

Con dynamic la distribución tiene lugar en tiempo de ejecución pero siempre se asigna el mismo tamaño de bloque.

Con guided se empieza asignando grupos de iteraciones grandes y se observa que este número va disminuyendo según va disminuyendo el número de threads (según el seminario número de iteraciones / número de threads) pero nunca se disminuye por debajo del chunck asignado. Es interesante observar que esto supone menos sobrecargas para las hebras respecto a dynamic porque claramente el tiempo que se emplea en asignar las tareas y comunicar las hebras debería ser menor.

En la versión de 4 threads con guided se aprecian claramente dos detalles. 1) Se asignan bloques del tamaño mencionado anteriormente redondeando al alza y 2) La última hebra puede quedarse con más iteraciones de las que le corresponden en algún caso.

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?

CÓDIGO FUENTE: `scheduled-clauseModificado.c`

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
 #include <omp.h>
#else
 #define omp_get_thread_num() 0
#endif
main(int argc, char **argv) {
 int i, n = 7,chunk, a[n],suma=0;
 omp_sched_t kind;
 int modifier;

 if(argc < 2) {
  fprintf(stderr,"\nFalta chunk \n");
  exit(-1);
 }
 chunk = atoi(argv[1]);
 for (i=0; i<n; i++) a[i] = i;

#pragma omp parallel for firstprivate(suma) lastprivate(suma)
schedule(dynamic,chunk)
for (i=0; i<n; i++)
{
  suma = suma + a[i];
  printf(" thread %d suma a[%d] suma=%d \n",omp_get_thread_num(),i,suma);
  if(i == n-1){
    omp_get_schedule(&kind, &modifier);
    printf("Dentro de 'parallel' dyn-var=%d \n",omp_get_dynamic());
    printf("Dentro de 'parallel' nthreads-var:%d \n",omp_get_max_threads());
    printf("Dentro de 'parallel' thread-limit-var:%d
\n",omp_get_thread_limit());
    printf("Dentro de 'parallel' run-sched-var (kind,modifier):%d,%d
\n",kind,modifier);
  }
 }

 printf("Fuera de 'parallel' suma=%d\n",suma);
 printf("Fuera de 'parallel' dyn-var=%d \n",omp_get_dynamic());
 printf("Fuera de 'parallel' nthreads-var:%d \n",omp_get_max_threads());
 printf("Fuera de 'parallel' thread-limit-var:%d \n",omp_get_thread_limit());
 omp_get_schedule(&kind, &modifier);
 printf("Fuera de 'parallel' run-sched-var (kind,modifier):%d,%d
\n",kind,modifier);
}
```

CAPTURAS DE PANTALLA:

```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ export OMP_SCHEDULE="static,1"
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ export OMP_DYNAMIC=false
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ export OMP_NUM_THREADS=8
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ export OMP_THREAD_LIMIT=8
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ ./scheduled-clause 10
 thread 1 suma a[0] suma=0
 thread 1 suma a[1] suma=1
 thread 1 suma a[2] suma=3
 thread 1 suma a[3] suma=6
 thread 1 suma a[4] suma=10
 thread 1 suma a[5] suma=15
 thread 1 suma a[6] suma=21
Dentro de 'parallel' dyn-var=0
Dentro de 'parallel' nthreads-var:8
Dentro de 'parallel' thread-limit-var:8
Dentro de 'parallel' run-sched-var (kind,modifier):1,1
Fuera de 'parallel' suma=21
Fuera de 'parallel' dyn-var=0
Fuera de 'parallel' nthreads-var:8
Fuera de 'parallel' thread-limit-var:8
Fuero de 'parallel' run-sched-var (kind,modifier):1,1
```

```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ export OMP_SCHEDULE="guided,2"
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ export OMP_DYNAMIC=true
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ export OMP_THREAD_LIMIT=4
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ export OMP_NUM_THREADS=8
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ ./scheduled-clause 10
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 0 suma a[2] suma=3
 thread 0 suma a[3] suma=6
 thread 0 suma a[4] suma=10
 thread 0 suma a[5] suma=15
 thread 0 suma a[6] suma=21
Dentro de 'parallel' dyn-var=1
Dentro de 'parallel' nthreads-var:8
Dentro de 'parallel' thread-limit-var:4
Dentro de 'parallel' run-sched-var (kind,modifier):3,2
Fuera de 'parallel' suma=21
Fuera de 'parallel' dyn-var=1
Fuera de 'parallel' nthreads-var:8
Fuera de 'parallel' thread-limit-var:4
Fuero de 'parallel' run-sched-var (kind,modifier):3,2
```

RESPUESTA:

En todas las ejecuciones se ha venido imprimiendo el mismo resultado dentro y fuera de la región paralela. Aquí he consultado por primera vez la especificación OpenMP Application Program Interface en su Versión 3.0 de Mayo de 2008. He encontrado que existe otro modo para schedule: “auto”. Cuando se exporta el valor auto o runtime no se debe especificar el tamaño del chunk. Se imprime el tipo de schedule en formato numérico donde 1 es static, 2 es dynamic y 3 es guided.

Las cláusulas no modifican el valor de las variables de control.

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.

CÓDIGO FUENTE: `scheduled-clauseModificado4.c`

```
/* Tipo de letra Courier new o Liberation Mono. Tamaño 8 o 9 .*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#include <stdio.h>
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
 #include <omp.h>
#else
 #define omp_get_thread_num() 0
#endif
main(int argc, char **argv) {
 int i, n = 7,chunk, a[n],suma=0;
 omp_sched_t kind;
 int modifier;

 if(argc < 2) {
  fprintf(stderr,"\nFalta chunk \n");
  exit(-1);
 }
 chunk = atoi(argv[1]);
 for (i=0; i<n; i++) a[i] = i;

#pragma omp parallel for firstprivate(suma) lastprivate(suma)
schedule(dynamic,chunk)
for (i=0; i<n; i++)
{
  suma = suma + a[i];
  printf(" thread %d suma a[%d] suma=%d \n",omp_get_thread_num(),i,suma);
  if(i == n-1){
    omp_get_schedule(&kind, &modifier);
    printf("Dentro de 'parallel' dyn-var=%d \n",omp_get_dynamic());
    printf("Dentro de 'parallel' nthreads-var:%d \n",omp_get_max_threads());
    printf("Dentro de 'parallel' thread-limit-var:%d
\n",omp_get_thread_limit());
    printf("Dentro de 'parallel' run-sched-var (kind,modifier):%d,%d
\n",kind,modifier);
    printf("Dentro de 'parallel' num_threads=%d \n",omp_get_num_threads());
    printf("Dentro de 'parallel' num_procs=%d \n",omp_get_num_procs());
    printf("Dentro de 'parallel' in_parallel=%d \n",omp_in_parallel());
  }
 }

 printf("Fuera de 'parallel' suma=%d\n",suma);
 printf("Fuera de 'parallel' dyn-var=%d \n",omp_get_dynamic());
 printf("Fuera de 'parallel' nthreads-var:%d \n",omp_get_max_threads());
 printf("Fuera de 'parallel' thread-limit-var:%d \n",omp_get_thread_limit());
 omp_get_schedule(&kind, &modifier);
 printf("Fuera de 'parallel' run-sched-var (kind,modifier):%d,%d
\n",kind,modifier);
 printf("Fuera de 'parallel' num_threads=%d \n",omp_get_num_threads());
```

```
 printf("Fuera de 'parallel' num_procs=%d \n",omp_get_num_procs());
 printf("Fuera de 'parallel' in_parallel=%d \n",omp_in_parallel());
}
```

CAPTURAS DE PANTALLA:

```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ ./scheduled-clause 10
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 0 suma a[2] suma=3
 thread 0 suma a[3] suma=6
 thread 0 suma a[4] suma=10
 thread 0 suma a[5] suma=15
 thread 0 suma a[6] suma=21
Dentro de 'parallel' dyn-var=1
Dentro de 'parallel' nthreads-var:8
Dentro de 'parallel' thread-limit-var:4
Dentro de 'parallel' run-sched-var (kind,modifier):2,1
Dentro de 'parallel' num_threads=4
Dentro de 'parallel' num_procs=4
Dentro de 'parallel' in_parallel=1
Fuera de 'parallel' suma=21
Fuera de 'parallel' dyn-var=1
Fuera de 'parallel' nthreads-var:8
Fuera de 'parallel' thread-limit-var:4
Fuera de 'parallel' run-sched-var (kind,modifier):2,1
Fuera de 'parallel' num_threads=1
Fuera de 'parallel' num_procs=4
Fuera de 'parallel' in_parallel=0
```

```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ OMP_NUM_THREADS_LIMIT=5
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ ./scheduled-clause 3
 thread 2 suma a[6] suma=6
Dentro de 'parallel' dyn-var=1
Dentro de 'parallel' nthreads-var:5
Dentro de 'parallel' thread-limit-var:4
Dentro de 'parallel' run-sched-var (kind,modifier):2,1
Dentro de 'parallel' num_threads=4
Dentro de 'parallel' num_procs=4
Dentro de 'parallel' in_parallel=1
 thread 1 suma a[3] suma=3
 thread 1 suma a[4] suma=7
 thread 1 suma a[5] suma=12
 thread 3 suma a[0] suma=0
 thread 3 suma a[1] suma=1
 thread 3 suma a[2] suma=3
Fuera de 'parallel' suma=6
Fuera de 'parallel' dyn-var=1
Fuera de 'parallel' nthreads-var:5
Fuera de 'parallel' thread-limit-var:4
Fuera de 'parallel' run-sched-var (kind,modifier):2,1
Fuera de 'parallel' num_threads=1
Fuera de 'parallel' num_procs=4
Fuera de 'parallel' in_parallel=0
```

RESPUESTA:

Se observa que el número de procesadores disponibles para el programa en el momento de ejecución no varía. Por otro lado, sí varía el número de threads que ejecutan la región paralela y la no paralela y por último cuando nos encontramos en una región parallel omp_in_parallel() devuelve 1 y en caso contrario un 0.

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.

CÓDIGO FUENTE: `scheduled-clauseModificado5.c`

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
 #include <omp.h>
#else
 #define omp_get_thread_num() 0
#endif
main(int argc, char **argv) {
 int i, n = 7,chunk, a[n],suma=0;
 omp_sched_t kind;
 int modifier;

 if(argc < 2) {
  fprintf(stderr,"\nFalta chunk \n");
  exit(-1);
 }
 chunk = atoi(argv[1]);
 for (i=0; i<n; i++) a[i] = i;

#pragma omp parallel for firstprivate(suma) lastprivate(suma)
schedule(dynamic,chunk)
for (i=0; i<n; i++)
{
  suma = suma + a[i];
  printf(" thread %d suma a[%d] suma=%d \n",omp_get_thread_num(),i,suma);
  if(i == n-1){
    omp_get_schedule(&kind, &modifier);
  //  printf("Dentro de 'parallel' dyn-var=%d \n",omp_get_dynamic());
  //  printf("Dentro de 'parallel' nthreads-var:%d \n",omp_get_max_threads());
  //  printf("Dentro de 'parallel' thread-limit-var:%d
\n",omp_get_thread_limit());
  //  printf("Dentro de 'parallel' run-sched-var (kind,modifier):%d,%d
\n",kind,modifier);
  //  printf("Dentro de 'parallel' num_threads=%d \n",omp_get_num_threads());
  //  printf("Dentro de 'parallel' num_procs=%d \n",omp_get_num_procs());
  //  printf("Dentro de 'parallel' in_parallel=%d \n",omp_in_parallel());
  }
 }

 //printf("Fuera de 'parallel' suma=%d\n",suma);
 //printf("Fuera de 'parallel' dyn-var=%d \n",omp_get_dynamic());
 //printf("Fuera de 'parallel' nthreads-var:%d \n",omp_get_max_threads());
 //printf("Fuera de 'parallel' thread-limit-var:%d
\n",omp_get_thread_limit());
 //omp_get_schedule(&kind, &modifier);
 //printf("Fuera de 'parallel' run-sched-var (kind,modifier):%d,%d
\n",kind,modifier);
 //printf("Fuera de 'parallel' num_threads=%d \n",omp_get_num_threads());
 //printf("Fuera de 'parallel' num_procs=%d \n",omp_get_num_procs());
 //printf("Fuera de 'parallel' in_parallel=%d \n",omp_in_parallel());

 //Si los dynamic_threads se evalúan como un valor distinto de cero, el número
de subprocesos que se utilizan para
 //ejecutar las regiones paralelas subsiguientes se puede ajustar
automáticamente
```

```
 printf("Antes de modificar dyn-var=%d \n",omp_get_dynamic());
 omp_set_dynamic(0);
 printf("Despues de modificar dyn-var=%d \n",omp_get_dynamic());
 printf("Antes de modificar nthreads-var:%d \n",omp_get_max_threads());
 omp_set_num_threads(10);
 printf("Despues de modificar nthreads-var:%d \n",omp_get_max_threads());
 printf("Antes de modificar run-sched-var (kind,modifier):%d,%d
\n",kind,modifier);
 omp_set_schedule(omp_sched_guided,1);
 printf("Despues de modificar run-sched-var (kind,modifier):%d,%d
\n",kind,modifier);
}
```

CAPTURAS DE PANTALLA:

```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ OMP_SCHEDULE="guided,2"
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3$ ./scheduled-clause 3
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 0 suma a[2] suma=3
 thread 3 suma a[3] suma=3
 thread 3 suma a[4] suma=7
 thread 3 suma a[5] suma=12
 thread 1 suma a[6] suma=6
Antes de modificar dyn-var=1
Despues de modificar dyn-var=0
Antes de modificar nthreads-var:5
Despues de modificar nthreads-var:10
Antes de modificar run-sched-var (kind,modifier):3,2
Despues de modificar run-sched-var (kind,modifier):3,2
```

RESPUESTA:

Según la página de ibm :

http://www-01.ibm.com/support/docview.wss?uid=swg1LI76992

existe un problema con la llamada a omp_set_schedule lo que probablemente motiva que en las distintas ejecuciones no se ve modificado correctamente.

6. Implementar un programa secuencial en C que multiplique una matriz triangular por un 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.

CÓDIGO FUENTE: `pmtv-secuencial.c`

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
void main(int argc,char **argv) {
...
srand(time(0));
//Inicializamos M, V
for(i=0; i<N; i++)
 for(j=0; j<i+1; j++)
 M[i][j] = 1;
for(i=0; i<N; i++)
 V[i] = i+1;
//Multiplicamos M x V = V2
t1 = omp_get_wtime();
for(i=0; i<N; i++){
 V2[i] = 0;
 for(j=0; j<i+1; j++)
  V2[i] += M[i][j] * V[j];
}
t2 = omp_get_wtime();
t2 = t2-t1;
...
}
```

CAPTURAS DE PANTALLA:

```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3/r$ ./ej6 8
M:
1 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0
1 1 1 0 0 0 0 0
1 1 1 1 0 0 0 0
1 1 1 1 1 0 0 0
1 1 1 1 1 1 0 0
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1

V:
1
2
3
4
5
6
7
8

Resultado MxV:
1 3 6 10 15 21 28 36
Tiempo en ejecutar MxV:0.0000005
```

```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3/r$ ./ej6 10
V2[0]:1
V2[N-1]:55

Tiempo en ejecutar MxV:0.0000006
```

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. 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 que multiplica una matriz triangular por un vector con las alternativas de planificación `static`, `dynamic` y `guided` para `chunk` de 2, 64, 128, 1024 y el chunk por defecto para la alternativa. No use vectores mayores de 32768 componentes ni menores de 4096 componentes. El número de threads en las ejecuciones debe coincidir con el número de cores. Rellenar la Tabla 3 con los tiempos obtenidos, ponga en la tabla el número de threads que utilizan las ejecuciones. Representar el tiempo para `static`, `dynamic` y `guided` en función del tamaño del `chunk` en una gráfica. Rellenar la tabla y realizar la gráfica también para el PC local. ¿Qué alternativa ofrece mejores prestaciones? Razone por qué.

RESPUESTA:

Por defecto para static se asigna un chunk nulo, para dynamic se asigna una unidad de chunk y para guided se asigna también una unidad de chunk.

En las gráficas hay que tener especial cuidado porque hemos representado el caso default en un chunk 500 para que se vea mejor la distribución. No tendrá importancia si aquí en la respuesta lo remarcamos.

Para el PC local los resultados son muy parecidos entre sí. En alguna ocasión la planificación guided supera a la static. Ambos son mejores que la dinámica. Parece que de algún modo funcionan mejor las planificaciones que tienen que realizar menos peticiones en la repartición de tareas. Dicho de otro modo, en la planificación estática las iteraciones que ejecuta cada thread se deciden en compilación mientras que en la planificación dinámica se realiza en ejecución. La planificación guided se salva porque asigna bloques grandes primero. Quizá estos resultados variaran si aumentamos el número de componentes (15000 para esta ejecución).

Para ATCGRID el comentario es similar. Simplemente observamos además que en ambos casos la planificación estática era peor que la guided con chunk pequeños pero que si el chunk aumenta mejora respecto de esta. Siguiendo la explicación anterior, esto podría deberse a que con un chunk de gran tamaño la planificación guided se comporta aproximadamente como una static (no puede disminuir lo asignado por debajo de chunk) pero tiene la penalización de que reparte las tareas en tiempo de ejecución.

CÓDIGO FUENTE: pmtv-OpenMP.c

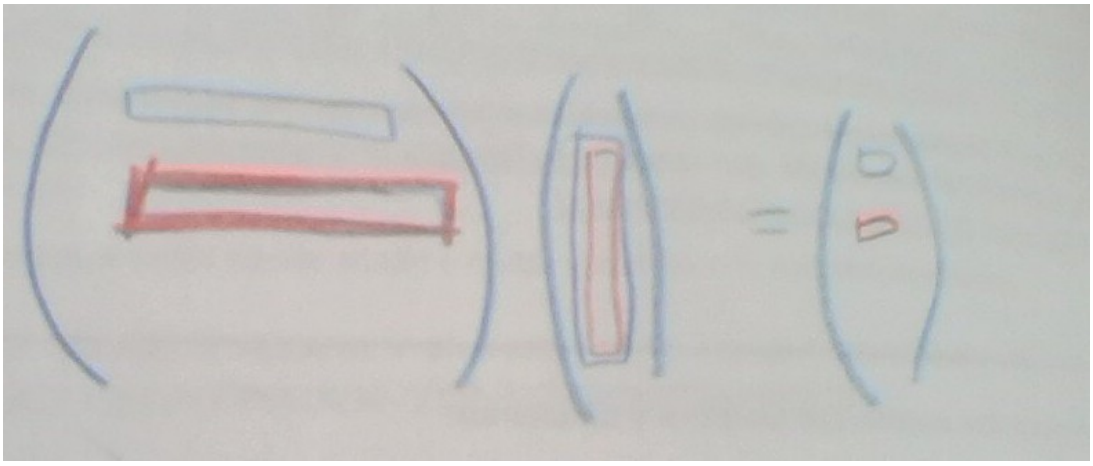
```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
void main(int argc,char **argv) {
...
srand(time(0));
//Inicializamos M y V
#pragma omp parallel for schedule(runtime) private(j)
for(i=0; i<N; i++){
 for(j=0; j<i+1; j++){ M[i][j] = 1; }
}
#pragma omp parallel for schedule(runtime)
for(i=0; i<N; i++){ V[i] = i+1; }

//Multiplicamos M x V = V2
t1 = omp_get_wtime();
#pragma omp parallel for schedule (runtime) private(j)
for(i=0; i<N; i++){
 V2[i] = 0;
 for(j=0; j<i+1; j++)
  V2[i] += M[i][j] * V[j];
}

t2 = omp_get_wtime();
t2 = t2-t1;
...
}
```

DESCOMPOSICIÓN DE DOMINIO:

En este caso, cada iteración realiza un producto escalar como se indica en este dibujo



CAPTURAS DE PANTALLA:

```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3/ejecutable$ ./ej7 8
run-sched-var=3, 1
M:
1 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0
1 1 1 0 0 0 0 0
1 1 1 1 0 0 0 0
1 1 1 1 1 0 0 0
1 1 1 1 1 1 0 0
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1

V:
1
2
3
4
5
6
7
8

Resultado MxV:
1 3 6 10 15 21 28 36
Tiempo en ejecutar MxV:0.0000049
```

```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3/ejecutable$ ./ej7 10
run-sched-var=3, 1
V2[0]:1
V2[N-1]:55

Tiempo en ejecutar MxV:0.0000055
```

TABLA RESULTADOS Y GRÁFICA ATCGRID

Tabla 3 . Tiempos de ejecución de la multiplicación de una matriz y un vector. Versión paralela. ATCGRID

Chunk	Static 12 threads	Dynamic 12 threads	Guided 12 threads
por defecto	0.0478907	0.0515784	0.0261805
2	0.0342679	0.0493022	0.0261245
64	0.0267778	0.0266884	0.0264924
128	0.0276189	0.0279916	0.0299362
1024	0.0420444	0.0424946	0.0456188

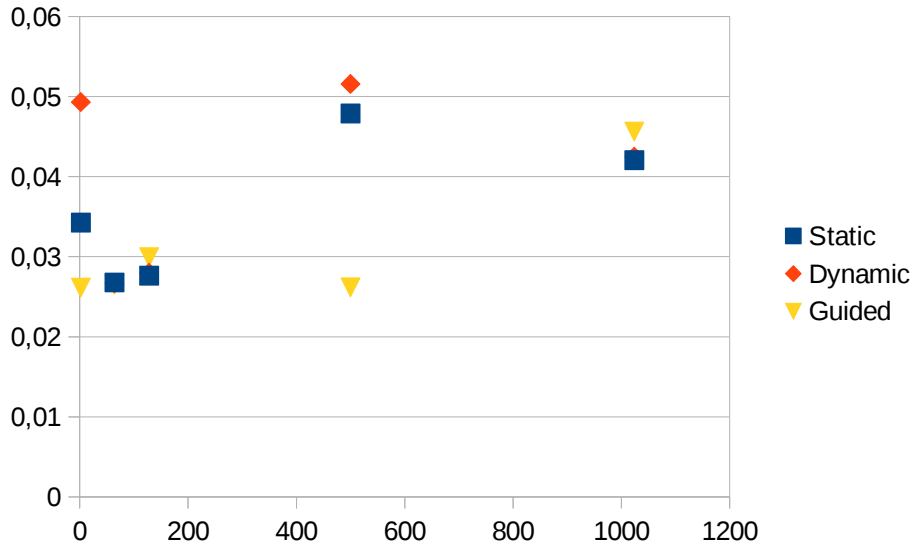
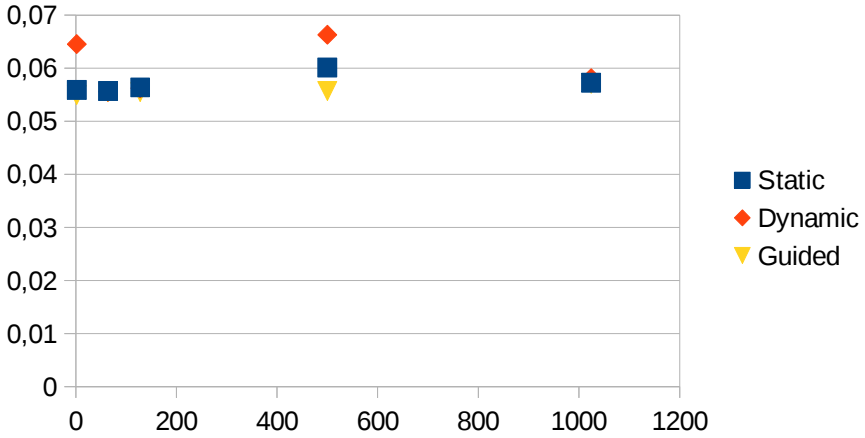


TABLA RESULTADOS Y GRÁFICA PC LOCAL

Tabla 4 . Tiempos de ejecución de la multiplicación de una matriz y un vector. Versión paralela. PC local.

Chunk	Static 4 threads	Dynamic 4 threads	Guided 4 threads
por defecto	0.0601313	0.0662982	0.0557280
2	0.0559147	0.0645444	0.0549552
64	0.0557320	0.0555740	0.0563637
128	0.0563929	0.0562218	0.0555104
1024	0.0572728	0.0581269	0.0570470



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

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

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.

CÓDIGO FUENTE: `pmm-secuencial.c`

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
void main(int argc,char **argv) {
...
srand(time(0));
//Inicializamos M1 y M2
for(i=0; i<N; i++)
 for(j=0; j<N; j++)
  M[i][j] = 0;
for(i=0; i<N; i++)
 for(j=0; j<N; j++)
  M1[i][j] = i*j+1;
for(i=0; i<N; i++)
 for(j=0; j<N; j++)
  M2[i][j] = 1;
//Multiplicamos M1 x M2 = M
t1=omp_get_wtime();
for(i=0;i<N;i++){
 for(j=0;j<N;j++) {
  for(k=0;k<N;k++) {M[i][j]=M[i][j]+M1[i][k]*M2[k][j];}
 }
}
t2 = omp_get_wtime();
t2 = t2-t1;
...
}
```

CAPTURAS DE PANTALLA:

```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3/codigos$ ./ej8 8
M1:
1 1 1 1 1 1 1 1
1 2 3 4 5 6 7 8
1 3 5 7 9 11 13 15
1 4 7 10 13 16 19 22
1 5 9 13 17 21 25 29
1 6 11 16 21 26 31 36
1 7 13 19 25 31 37 43
1 8 15 22 29 36 43 50
M2:
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1

Resultado M1xM2:
8 8 8 8 8 8 8 8
36 36 36 36 36 36 36 36
64 64 64 64 64 64 64 64
92 92 92 92 92 92 92 92
120 120 120 120 120 120 120 120
148 148 148 148 148 148 148 148
176 176 176 176 176 176 176 176
204 204 204 204 204 204 204 204

Tiempo en ejecutar M1xM2:0.0000015
```

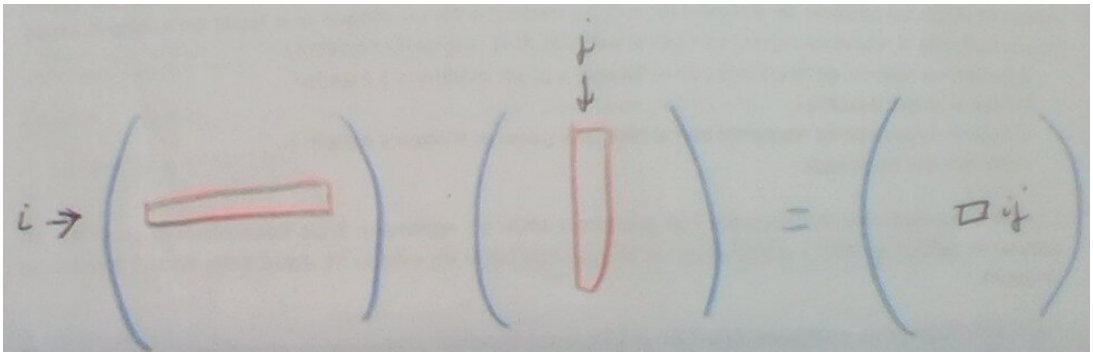
```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3/codigos$ ./ej8 10
M[0][0]:10
M[N-1][N-1]:415

Tiempo en ejecutar M1xM2:0.0000025
```

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).

DESCOMPOSICIÓN DE DOMINIO:

En este caso la descomposición del dominio reparte a cada hebra un producto escalar (por supuesto las diferentes planificaciones agruparán algunos de estos productos escalares).



CÓDIGO FUENTE: `pmm-OpenMP.c`

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif
void main(int argc,char **argv) {
...
srand(time(0));
//Inicializamos M1 y M2
#pragma omp parallel for private(j) schedule(runtime)
for(i=0; i<N; i++)
 for(j=0; j<N; j++){ M1[i][j] = i*j+1; }
#pragma omp parallel for private(j) schedule(runtime)
for(i=0; i<N; i++)
 for(j=0; j<N; j++){ M2[i][j] = 1; }
//Multiplicamos M1 x M2 = M
t1=omp_get_wtime();
#pragma omp parallel for private(j) schedule(runtime)
for(i=0;i<N;i++){
 #pragma omp parallel for private(k) schedule(runtime)
 for(j=0;j<N;j++) {
  for(k=0;k<N;k++) {M[i][j]=M[i][j]+M1[i][k]*M2[k][j];}
 }
}
t2 = omp_get_wtime();
t2 = t2-t1;
...
```

CAPTURAS DE PANTALLA:

```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3/codigos$ ./ej9 8
M1:
1 1 1 1 1 1 1 1
1 2 3 4 5 6 7 8
1 3 5 7 9 11 13 15
1 4 7 10 13 16 19 22
1 5 9 13 17 21 25 29
1 6 11 16 21 26 31 36
1 7 13 19 25 31 37 43
1 8 15 22 29 36 43 50
M2:
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1

Resultado M1xM2:
8 8 8 8 8 8 8 8
36 36 36 36 36 36 36 36
64 64 64 64 64 64 64 64
92 92 92 92 92 92 92 92
120 120 120 120 120 120 120 120
148 148 148 148 148 148 148 148
176 176 176 176 176 176 176 176
204 204 204 204 204 204 204 204

Tiempo en ejecutar M1xM2:0.0000929
```

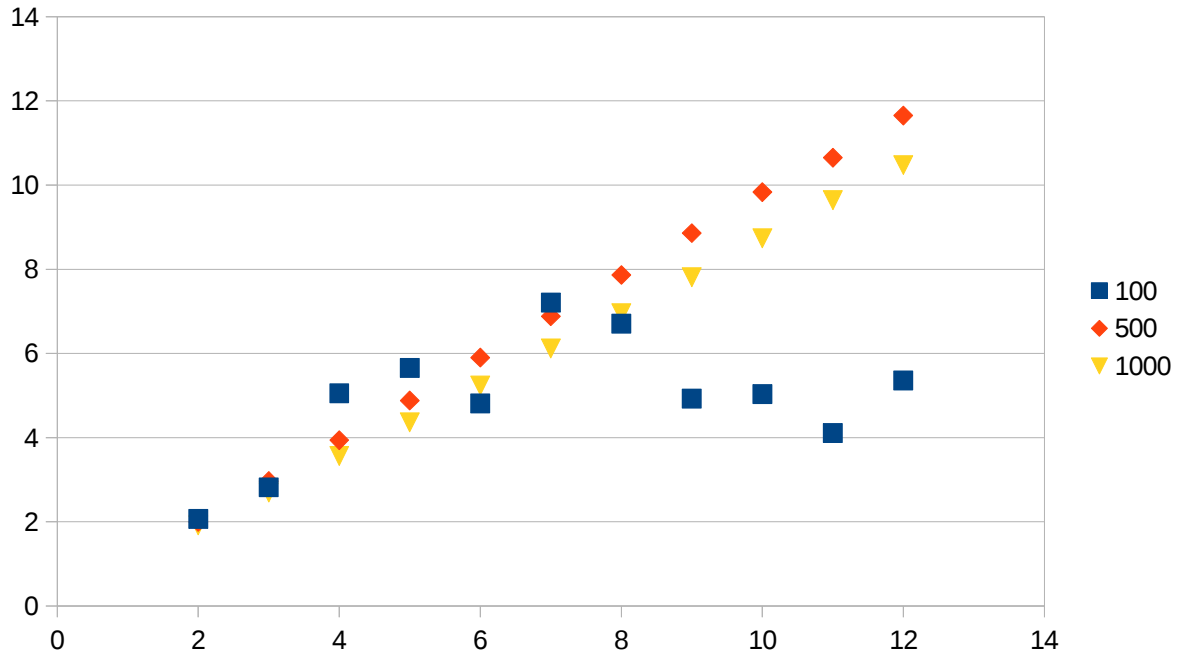
```
rjraya@rjraya-SATELLITE-C55-A-1EL:~/Documentos/ac/practicas/BloquePractico3/codigos$ ./ej9 10
M[0][0]:10
M[N-1][N-1]:415

Tiempo en ejecutar M1xM2:0.0001068
```

10. Hacer un estudio de escalabilidad (ganancia en velocidad en función del número de cores) en atcgrid y en el PC local del código paralelo implementado para tres tamaños de las matrices. Debe recordar usar –O2 al compilar. 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 . Consulte la Lección 6/Tema 2.

Se han realizado las ejecuciones con planificación de tipo guided y chunk de tamaño uno. Hemos utilizado matrices cuadradas de tamaño 100, 500 y 1000. Hemos realizado ejecuciones con entre 1-4 threads (PC local) y entre 1-12 threads (ATCGRID). Se han obtenido los siguientes resultados:

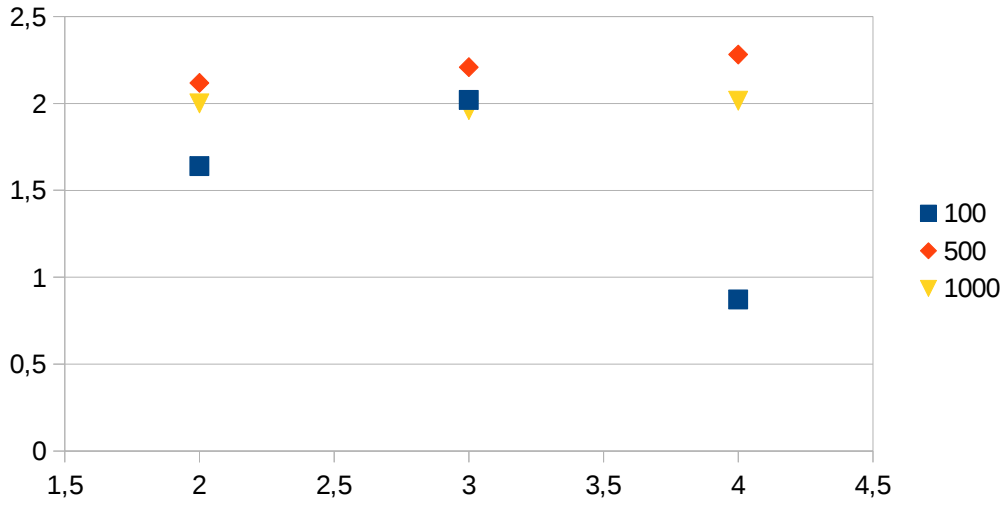
ESTUDIO DE ESCALABILIDAD EN ATCGRID:



	Ganancia ATCGRID				
Tamaño/Threa	2	3	4	5	6
100	2,0656839218	2,8187304112	5,0520667393	5,6546546547	4,8147023572
500	1,9968347527	2,9673688801	3,9385828956	4,8806694577	5,900901804
1000	1,9137671539	2,6938637361	3,5639682215	4,3604664184	5,2346574315

7	8	9	10	11	12
7,2067934458	6,7077813648	4,926901063	5,0318162839	4,107710137	5,3579939534
6,8824254373	7,8647637935	8,8592638105	9,8325395585	10,651152097	11,651342179
6,1157848699	6,954415691	7,8106470922	8,7364076723	9,6442465931	10,475634226

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:



	Ganancia-Local		
amaño/Threads	2	3	4
100	1,6403891212	2,0196823078	0,8727671527
500	2,1186800389	2,2089746738	2,2814676467
1000	2,0005035737	1,9623010057	2,0163058878