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): Paula Ruiz García
Grupo de prácticas: D1
Fecha de entrega: 13 de Mayo de 2018
Fecha evaluación en clase: 14 de Mayo de 2018

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

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>

int main(int argc, char **argv)
{
  int i, n=20, tid;
  int a[n],suma=0,sumalocal;
  if(argc < 3)  {
    fprintf(stderr,"[ERROR]-Falta iteraciones\n");
    exit(-1);
   }
  n = atoi(argv[1]); if (n>20) n=20;
  for (i=0; i<n; i++) {
    a[i] = i;
  }
  int x;
  x = atoi(argv[2]);

 #pragma omp parallel num_threads(x) if(n>4) default(none) \
            private(sumalocal,tid) shared(a,suma,n)
 {  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);
```

```
  }

  return(0);
}
```

CAPTURAS DE PANTALLA:

If-clause sin modificar:

```
[Paula Ruiz García Paula@Pompitas:~/AC/P3/Codigo] 2018-04-30 Monday
$gcc -O2 -fopenmp if-clause.c -o if-clause
[Paula Ruiz García Paula@Pompitas:~/AC/P3/Codigo] 2018-04-30 Monday
$./if-clause 8
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 2 suma de a[4]=4 sumalocal=4
 thread 2 suma de a[5]=5 sumalocal=9
 thread 1 suma de a[2]=2 sumalocal=2
 thread 1 suma de a[3]=3 sumalocal=5
 thread 3 suma de a[6]=6 sumalocal=6
 thread 3 suma de a[7]=7 sumalocal=13
thread master=0 imprime suma=28
```

If-clause modificado:

```
[Paula Ruiz García Paula@Pompitas:~/AC/P3/Codigo] 2018-04-30 Monday
$gcc -O2 -fopenmp if-clauseModificado.c -o if-clauseM
[Paula Ruiz García Paula@Pompitas:~/AC/P3/Codigo] 2018-04-30 Monday
$./if-clauseM 8
[ERROR]-Falta iteraciones
[Paula Ruiz García Paula@Pompitas:~/AC/P3/Codigo] 2018-04-30 Monday
$./if-clauseM 8 3
 thread 1 suma de a[3]=3 sumalocal=3
 thread 1 suma de a[4]=4 sumalocal=7
 thread 1 suma de a[5]=5 sumalocal=12
 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 2 suma de a[6]=6 sumalocal=6
 thread 2 suma de a[7]=7 sumalocal=13
thread master=0 imprime suma=28
[Paula Ruiz García Paula@Pompitas:~/AC/P3/Codigo] 2018-04-30 Monday
$./if-clauseM 8 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 0 suma de a[3]=3 sumalocal=6
 thread 1 suma de a[4]=4 sumalocal=4
 thread 1 suma de a[5]=5 sumalocal=9
 thread 1 suma de a[6]=6 sumalocal=15
 thread 1 suma de a[7]=7 sumalocal=22
thread master=0 imprime suma=28
[Paula Ruiz García Paula@Pompitas:~/AC/P3/Codigo] 2018-04-30 Monday
$
```

RESPUESTA:
Lo que ilustra la cláusula `num_threads(x)` es que podemos decidir el número de hebras que van a paralelizar el código sin tener que recompilar.

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

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

RESPUESTA:

Usando static todas las tareas se van a repartir equitativamente usando Round-Robin.
Con dynamic y guided el reparto no se sabe como se distribuirá, pero se sabe que el tamaño de las tareas vendrá definido por el chunk, es decir, cada hebra hará chunk iteraciones.
Se supone que en el guided es el mas eficiente de todos porque repartirá además las tareas equilibradamente.

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

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
 #include <omp.h>
#else
  #define omp_get_thread_num() 0
  #define omp_get_num_threads() 1
  #define omp_set_num_threads(int)
#endif

int main(int argc, char **argv) {
 int i, n=200,chunk,a[n],suma=0;
 omp_sched_t kind;
 int chunk_value;

  if(argc < 3){
    fprintf(stderr,"\nFalta iteraciones o chunk \n");
    exit(-1);
  }

  n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);
  for (i=0; i<n; i++)      a[i] = i;

  #pragma omp parallel
  {
   #pragma omp for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
   for (i=0; i<n; i++)
   {  suma = suma + a[i];
      printf(" thread %d suma a[%d]=%d suma=%d \n",
           omp_get_thread_num(),i,a[i],suma);
   }

   #pragma omp single
   {
    printf("Dentro de 'parallel for'\n");
    omp_get_schedule(&kind, &chunk_value);
    printf("dyn-var: %d\n nthreads-var: %d\n thread-limit-var: %d\n run-sched-var(kind: %d, modifier: %d)\n",
omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit() , kind, chunk_value);
   }
  }
  printf("Fuera de 'parallel for' suma=%d\n",suma);
  omp_get_schedule(&kind, &chunk_value);
  printf("dyn-var: %d\n nthreads-var: %d\n thread-limit-var: %d\n run-sched-var(kind: %d, modifier: %d)\n",
omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit() , kind, chunk_value);
  return(0);
}
```

CAPTURAS DE PANTALLA:

```
[Paula Ruiz García Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$gcc -O2 -fopenmp scheduled-clauseModificado.c -o scheduled-clauseM
[Paula Ruiz García Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$./scheduled-clauseM 8 2
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[1]=1 suma=1
 thread 2 suma a[4]=4 suma=4
 thread 2 suma a[5]=5 suma=9
 thread 3 suma a[2]=2 suma=2
 thread 3 suma a[3]=3 suma=5
 thread 1 suma a[6]=6 suma=6
 thread 1 suma a[7]=7 suma=13
Dentro de 'parallel for'
dyn-var: 0
 nthreads-var: 4
 thread-limit-var: 2147483647
 run-sched-var(kind: 2, modifier: 1)
Fuera de 'parallel for' suma=13
dyn-var: 0
 nthreads-var: 4
 thread-limit-var: 2147483647
 run-sched-var(kind: 2, modifier: 1)
[Paula Ruiz García Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$
```

RESPUESTA:
Comprobamos que los valores son los mismos dentro y fuera de la región paralela.

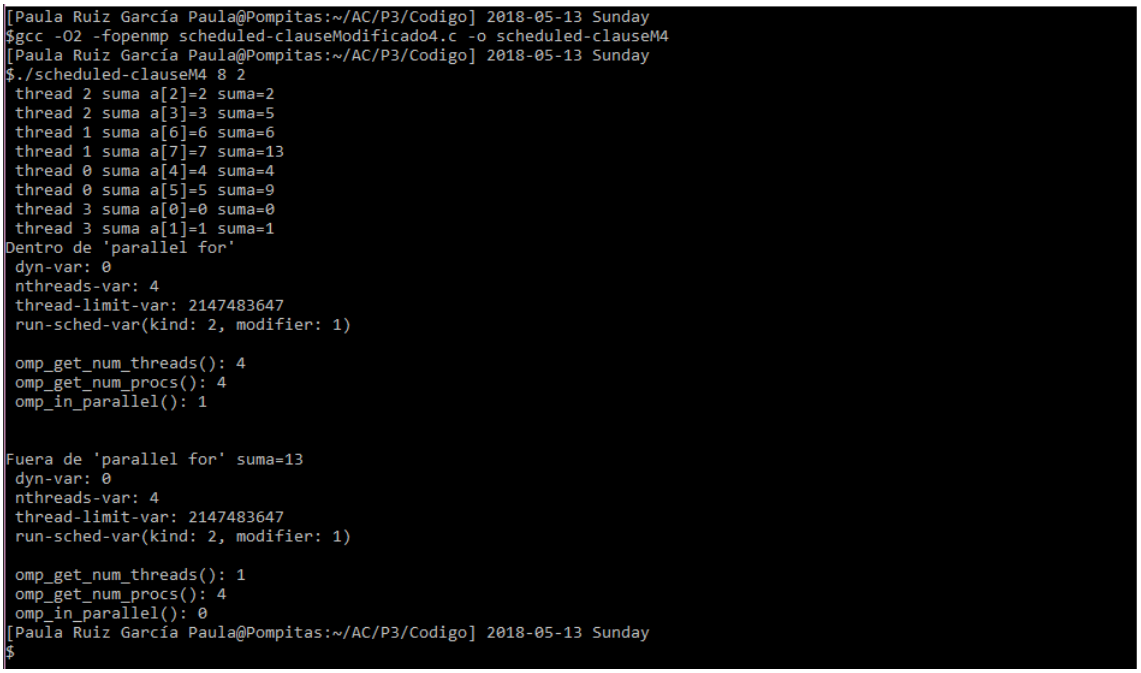
4. Usar en el ejemplo anterior las funciones omp_get_num_threads(), omp_get_num_procs() y omp_in_parallel() dentro y fuera de la región paralela. Imprimir los valores que obtienen estas funciones dentro (lo debe imprimir sólo uno de los threads) y fuera de la región paralela. Incorporar en su cuaderno 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

```
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
 #include <omp.h>
#else
  #define omp_get_thread_num() 0
  #define omp_get_num_threads() 1
  #define omp_set_num_threads(int)
  #define omp_in_parallel() 0
#endif

int main(int argc, char **argv) {
 int i, n=200,chunk,a[n],suma=0;
 omp_sched_t kind;
 int chunk_value;

  if(argc < 3)    {
    fprintf(stderr,"\nFalta iteraciones o chunk \n");
    exit(-1);
   }
  n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);
  for (i=0; i<n; i++)      a[i] = i;

  #pragma omp parallel
  {
   #pragma omp for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
   for (i=0; i<n; i++)
   {  suma = suma + a[i];
      printf(" thread %d suma a[%d]=%d suma=%d \n",
          omp_get_thread_num(),i,a[i],suma);
```

```
    }
   #pragma omp single
    {
      printf("Dentro de 'parallel for'\n");
      omp_get_schedule(&kind, &chunk_value);
      printf(" dyn-var: %d\n nthreads-var: %d\n thread-limit-var: %d\n run-sched-var(kind: %d, modifier: %d)\n",
omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit() , kind, chunk_value);
      printf("\n omp_get_num_threads(): %d\n omp_get_num_procs(): %d\n omp_in_parallel(): %d\n",
omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
    }
  }
  printf("\n\nFuera de 'parallel for' suma=%d\n",suma);
  omp_get_schedule(&kind, &chunk_value);
  printf(" dyn-var: %d\n nthreads-var: %d\n thread-limit-var: %d\n run-sched-var(kind: %d, modifier: %d)\n",
omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit() , kind, chunk_value);
  printf("\n omp_get_num_threads(): %d\n omp_get_num_procs(): %d\n omp_in_parallel(): %d\n",
omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
  return(0);
}
```

CAPTURAS DE PANTALLA:



RESPUESTA:

La única que se mantiene igual dentro y fuera de la región paralela es omp_get_num_procs(), las otras dos sí que cambian según el lugar en el que estén.

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

```
 #include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
```

```
 #include <omp.h>
#else
  #define omp_get_thread_num() 0
  #define omp_get_num_threads() 1
  #define omp_set_num_threads(int)
  #define omp_in_parallel() 0
  #define omp_set_dynamic(int)
#endif

int main(int argc, char **argv) {
 int i, n=200,chunk,a[n],suma=0;
 omp_sched_t kind;
 int chunk_value;

  if(argc < 3)    {
    fprintf(stderr,"\nFalta iteraciones o chunk \n");
    exit(-1);
   }
  n = atoi(argv[1]); if (n>200) n=200; chunk = atoi(argv[2]);

  for (i=0; i<n; i++)      a[i] = i;

  //Antes del cambio
  printf("Antes del cambio\n");
  omp_get_schedule(&kind, chunk_value);
  printf(" dyn-var: %d\n nthreads-var: %d\n thread-limit-var: %d\n run-sched-var(kind: %d, modifier: %d)\n",
omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit() , kind, chunk_value);
  printf("\n omp_get_num_threads(): %d\n omp_get_num_procs(): %d\n omp_in_parallel(): %d\n",
omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());

  //Cambio de valores
  omp_set_dynamic(2);
  omp_set_num_threads(2);
  omp_set_schedule(2, 1);

  #pragma omp parallel
  {
   #pragma omp for firstprivate(suma) lastprivate(suma) schedule(dynamic,chunk)
   for (i=0; i<n; i++)
   {   suma = suma + a[i];
       printf(" thread %d suma a[%d]=%d suma=%d \n",
            omp_get_thread_num(),i,a[i],suma);
   }

   #pragma omp single
   {
     printf("Dentro de 'parallel for'\n");
     omp_get_schedule(&kind, &chunk_value);
     printf(" dyn-var: %d\n nthreads-var: %d\n thread-limit-var: %d\n run-sched-var(kind: %d, modifier: %d)\n",
omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit() , kind, chunk_value);
     printf("\n omp_get_num_threads(): %d\n omp_get_num_procs(): %d\n omp_in_parallel(): %d\n",
omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
   }
  }
  printf("\n\nFuera de 'parallel for' suma=%d\n",suma);
  omp_get_schedule(&kind, &chunk_value);
  printf(" dyn-var: %d\n nthreads-var: %d\n thread-limit-var: %d\n run-sched-var(kind: %d, modifier: %d)\n",
```

```
omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit() , kind, chunk_value);
  printf("\n omp_get_num_threads(): %d\n omp_get_num_procs(): %d\n omp_in_parallel(): %d\n",
omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
  return(0);
}
```

CAPTURAS DE PANTALLA:

```
[Paula Ruiz García Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$gcc -O2 -fopenmp scheduled-clauseModificado5.c -o scheduled-clauseM5
[Paula Ruiz García Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$./scheduled-clauseM5 8 2
Antes del cambio
 dyn-var: 0
 nthreads-var: 4
 thread-limit-var: 2147483647
 run-sched-var(kind: 2, modifier: 1)

 omp_get_num_threads(): 1
 omp_get_num_procs(): 4
 omp_in_parallel(): 0
 thread 0 suma a[0]=0 suma=0
 thread 0 suma a[1]=1 suma=1
 thread 1 suma a[2]=2 suma=2
 thread 1 suma a[3]=3 suma=5
 thread 1 suma a[6]=6 suma=11
 thread 1 suma a[7]=7 suma=18
 thread 0 suma a[4]=4 suma=5
 thread 0 suma a[5]=5 suma=10
Dentro de 'parallel for'
 dyn-var: 1
 nthreads-var: 2
 thread-limit-var: 2147483647
 run-sched-var(kind: 2, modifier: 1)

 omp_get_num_threads(): 2
 omp_get_num_procs(): 4
 omp_in_parallel(): 1


Fuera de 'parallel for' suma=18
 dyn-var: 1
 nthreads-var: 2
 thread-limit-var: 2147483647
 run-sched-var(kind: 2, modifier: 1)

 omp_get_num_threads(): 1
 omp_get_num_procs(): 4
 omp_in_parallel(): 0
[Paula Ruiz García Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$
```

RESPUESTA:

Como podemos comprobar, si cambiamos los valores, vemos que las funciones devuelven los nuevos valores establecidos.

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

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char **argv){
  int i, j;

  //Leer argumentos de entrada.
  if(argc < 2){
```

```
  fprintf(stderr, "Falta size\n");
  exit(-1);
}

unsigned int N = atoi(argv[1]);

//Inicializamos la matriz triangular.
int *vector, *resultado, **matriz;
vector = (int *) malloc(N*sizeof(int));
resultado = (int *) malloc(N*sizeof(int));
matriz = (int **) malloc(N*sizeof(int *));
for(i = 0; i<N; i++){
  matriz[i]=(int *) malloc(N*sizeof(int));
}

//Añadimos valores a la matriz
for(i=0; i<N; i++){
  for(j=i; j<N; j++){
    matriz[i][j]=2;
    vector[i]=6;
    resultado[i]=0;
  }
}

//Pintamos la matriz
printf("Matriz:\n");
for(i=0; i<N; i++){
  for(j=0; j<N; j++){
    if(j >= i){
      printf("%d ", matriz[i][j]);
    }
    else{
      printf("0 ");
    }
  }
  printf("\n");
}

//Pintamos el vector
printf("Vector:\n");
for (i=0; i<N; i++){
  printf("%d ", vector[i]);
}
printf("\n");

// Obtenemos los resaultados
for (i=0; i<N; i++)
  for (j=i; j<N; j++)
    resultado[i] += matriz[i][j] * vector[j];

// Pintamos los resultados
printf("Resultado:\n");
for (i=0; i<N; i++)
  printf("%d ", resultado[i]);
printf("\n");

// Liberamos la memoria
```

```
  for (i=0; i<N; i++)
    free(matriz[i]);
  free(matriz);
  free(vector);
  free(resultado);
return 0;
}
```

CAPTURAS DE PANTALLA:

```
[Paula Ruiz García Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$gcc -O2 pmtv-secuencial.c -o pmtv
[Paula Ruiz García Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$./pmtv 10
Matriz:
2 2 2 2 2 2 2 2 2 2
0 2 2 2 2 2 2 2 2 2
0 0 2 2 2 2 2 2 2 2
0 0 0 2 2 2 2 2 2 2
0 0 0 0 2 2 2 2 2 2
0 0 0 0 0 2 2 2 2 2
0 0 0 0 0 0 2 2 2 2
0 0 0 0 0 0 0 2 2 2
0 0 0 0 0 0 0 0 2 2
0 0 0 0 0 0 0 0 0 2
Vector:
6 6 6 6 6 6 6 6 6 6
Resultado:
120 108 96 84 72 60 48 36 24 12
[Paula Ruiz García Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$
```

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 tiem pos 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)

Los valores por defecto son:

Static: 0 (no tienen ningún chiunk ya que se divide en partes iguales).

Dynamic: 1

Guided: 1

(B)

En cada fila se hace una suma menos que multiplicaciones y se hacen i multiplicaciones en cada fila (desde i=0 hasta 15360).

Las iteraciones por filas se reparten de forma equitativa entre los 12 threads y se da a cada thread un conjunto de filas consecutivas, de lo que obtendremos que el primer thread tiene ejecuciones desde i=0 hasta i=x y el siguiente desde i=x+1 hasta i=2x, ocurrirá que habrá threads que ejecuten muchas operaciones y threads que ejecuten pocas instrucciones.

(C)
Si la asignación se hace con dynamic o guided, el numero de operaciones de multiplicación y suma deberá estabilizarse y, por tanto, debería reducirse el tiempo de ejecución.
En ambos tendrán repartos distintos para distintas ejecuciones, lo que da lugar a que tiempos diferentes de una ejecución a otra.
Pero, sin embargo, para este caso, ya que la duración de las iteraciones que se buscan paralelizar es heterogénea, ambos darán mejores resultados que static.

CAPTURA CÓDIGO FUENTE: pmtv-OpenMP.c

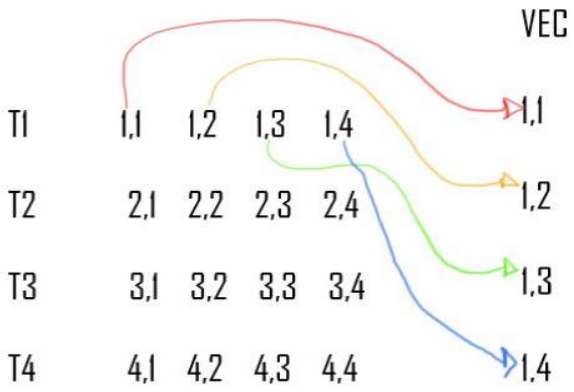
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#ifdef _OPENMP
 #include <omp.h>
#else
 #define omp_get_thread_num() 0
 #define omp_get_num_threads() 1
 #define omp_set_num_threads(omp_get_num_procs())
 #define omp_in_parallel() 0
 #define omp_set_dynamic(int)
#endif

int main(int argc, char **argv){
 int i, j;
 double t1, t2, final;

 //Leer argumentos de entrada.
 if(argc < 2){
  fprintf(stderr, "Falta size\n");
  exit(-1);
 }

 unsigned int N = atoi(argv[1]);

 //Inicializamos la matriz triangular.
 int *vector, *resultado, **matriz;
 vector = (int *) malloc(N*sizeof(int));
 resultado = (int *) malloc(N*sizeof(int));
 matriz = (int **) malloc(N*sizeof(int *));
 for(i = 0; i<N; i++){
  matriz[i]=(int *) malloc(N*sizeof(int));
 }

 //Añadimos valores a la matriz
 for(i=0; i<N; i++){
  for(j=i; j<N; j++){
   matriz[i][j]=2;
   vector[i]=6;
   resultado[i]=0;
```

```
  }
 }

 if(N<20){
  //Pintamos la matriz
  printf("Matriz:\n");
  for(i=0; i<N; i++){
   for(j=0; j<N; j++){
    if(j >= i){
     printf("%d ", matriz[i][j]);
    }
    else{
     printf("0 ");
    }
   }
   printf("\n");
  }

  //Pintamos el vector
  printf("Vector:\n");
  for (i=0; i<N; i++){
   printf("%d ", vector[i]);
  }
  printf("\n");
 }

 #pragma omp parallel
 {
  #pragma omp single
  t1 = omp_get_wtime();

  // Obtenemos los resultados
  #pragma omp for private(j) schedule(runtime)
  for (i=0; i<N; i++)
   for (j=i; j<N; j++)
    resultado[i] += matriz[i][j] * vector[j];

  #pragma omp single
  t2 = omp_get_wtime();
 }

 final = t2 - t1;

 if(N<20){
  // Pintamos los resultados
  printf("Resultado:\n");
  for (i=0; i<N; i++)
   printf("%d ", resultado[i]);
  printf("\n");
 }

 printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n", final, resultado[0], resultado[N-1]);

 // Liberamos la memoria
 for (i=0; i<N; i++)
  free(matriz[i]);
 free(matriz);
```

```
  free(vector);
  free(resultado);
return 0;
}
```

DESCOMPOSICIÓN DE DOMINIO:

Cada thread es asignada a una o varias filas, unos i determinados, en la cual, para cada fila su ejecuta suma Matriz[i][j] con todos los j que le corresponden (desde j=1 hasta j<N). Y de hay sacamos un vector final con la solución.



CAPTURAS DE PANTALLA:
Prueba 1:

```
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$cat ej7.e78601
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$cat ej7.o78601
Id/home/D1estudiante18 usuario del trabajo: D1estudiante18
Id/home/D1estudiante18 del trabajo: 78601.atcgrid
Nombre del trabajo especificado por usuario: ej7
Nodo que ejecuta qsub: atcgrid
Directorio en el que se ha ejecutado qsub: /home/D1estudiante18
Cola: ac
Nodos asignados al trabajo:
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
static y chunk por defecto
Tiempo = 0.039151871     Primera = 184320        Ultima=12
static y chunk 1
Tiempo = 0.036267017     Primera = 184320        Ultima=12
static y chunk 64
Tiempo = 0.032415683     Primera = 184320        Ultima=12
dynamic y chunk por defecto
Tiempo = 0.037052428     Primera = 184320        Ultima=12
dynamic y chunk 1
Tiempo = 0.037214105     Primera = 184320        Ultima=12
dynamic y chunk 64
Tiempo = 0.029451097     Primera = 184320        Ultima=12
guided y chunk por defecto
Tiempo = 0.032117923     Primera = 184320        Ultima=12
guided y chunk 1
Tiempo = 0.037564018     Primera = 184320        Ultima=12
guided y chunk 64
Tiempo = 0.039031605     Primera = 184320        Ultima=12
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
```

Prueba 2:

```
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$cat ej7.e78602
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$cat ej7.o78602
Id/home/D1estudiante18 usuario del trabajo: D1estudiante18
Id/home/D1estudiante18 del trabajo: 78602.atcgrid
Nombre del trabajo especificado por usuario: ej7
Nodo que ejecuta qsub: atcgrid
Directorio en el que se ha ejecutado qsub: /home/D1estudiante18
Cola: ac
Nodos asignados al trabajo:
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
atcgrid2
static y chunk por defecto
Tiempo = 0.040047094     Primera = 184320        Ultima=12
static y chunk 1
Tiempo = 0.034167702     Primera = 184320        Ultima=12
static y chunk 64
Tiempo = 0.032207792     Primera = 184320        Ultima=12
dynamic y chunk por defecto
Tiempo = 0.033801220     Primera = 184320        Ultima=12
dynamic y chunk 1
Tiempo = 0.038777399     Primera = 184320        Ultima=12
dynamic y chunk 64
Tiempo = 0.029457848     Primera = 184320        Ultima=12
guided y chunk por defecto
Tiempo = 0.034706016     Primera = 184320        Ultima=12
guided y chunk 1
Tiempo = 0.042534261     Primera = 184320        Ultima=12
guided y chunk 64
Tiempo = 0.036508310     Primera = 184320        Ultima=12
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
```

atcgrid:

```
sftp> put pmtv-OpenMP
Uploading pmtv-OpenMP to /home/D1estudiante18/pmtv-OpenMP
pmtv-OpenMP                                   100%   13KB  13.3KB/s   00:01
sftp> put pmvt-OpenMP_PCaula.sh
Uploading pmvt-OpenMP_PCaula.sh to /home/D1estudiante18/pmvt-OpenMP_PCaula.sh
pmvt-OpenMP_PCaula.sh                         100% 1103     0.4KB/s   00:03
sftp> get ej*
Fetching /home/D1estudiante18/ej7.e78601 to ej7.e78601
Fetching /home/D1estudiante18/ej7.e78602 to ej7.e78602
Fetching /home/D1estudiante18/ej7.o78601 to ej7.o78601
/home/D1estudiante18/ej7.o78601               100% 1145     0.6KB/s   00:02
Fetching /home/D1estudiante18/ej7.o78602 to ej7.o78602
/home/D1estudiante18/ej7.o78602               100% 1145     0.6KB/s   00:02
sftp>
```

```
[D1estudiante18@atcgrid ~]$ qsub pmvt-OpenMP_PCaula.sh -q ac
78601.atcgrid
[D1estudiante18@atcgrid ~]$ qsub pmvt-OpenMP_PCaula.sh -q ac
78602.atcgrid
[D1estudiante18@atcgrid ~]$ qstat
Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
78601.atcgrid              ej7              D1estudiante18  00:00:00 C ac
78602.atcgrid              ej7              D1estudiante18         0 R ac
[D1estudiante18@atcgrid ~]$ qstat
[D1estudiante18@atcgrid ~]$ ls
ej7.e78601  ej7.o78601  pmtv-OpenMP
ej7.e78602  ej7.o78602  pmvt-OpenMP_PCaula.sh
[D1estudiante18@atcgrid ~]$
```

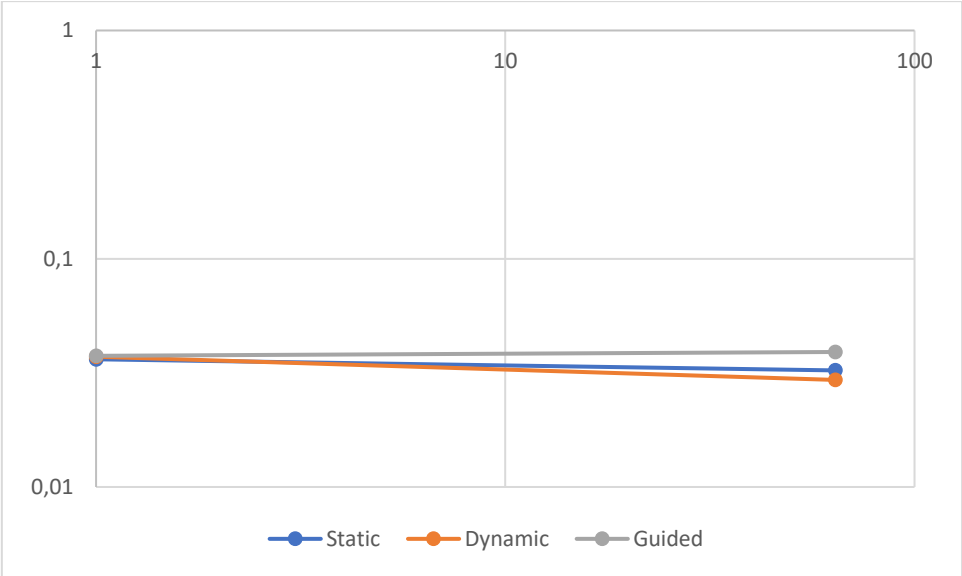
TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-OpenMP_PCaula.sh

```
#!/bin/bash
#PBS -N ej7
#PBS -q ac
echo "Id$PBS_O_WORKDIR usuario del trabajo: $PBS_O_LOGNAME"
echo "Id$PBS_O_WORKDIR del trabajo: $PBS_JOBID"
echo "Nombre del trabajo especificado por usuario: $PBS_JOBNAME"
echo "Nodo que ejecuta qsub: $PBS_O_HOST"
echo "Directorio en el que se ha ejecutado qsub: $PBS_O_WORKDIR"
echo "Cola: $PBS_QUEUE"
echo "Nodos asignados al trabajo:"
cat $PBS_NODEFILE
export OMP_SCHEDULE="static"
echo "static y chunk por defecto"
./pmtv-OpenMP 15360
export OMP_SCHEDULE="static,1"
echo "static y chunk 1"
./pmtv-OpenMP 15360
export OMP_SCHEDULE="static,64"
echo "static y chunk 64"
./pmtv-OpenMP 15360
export OMP_SCHEDULE="dynamic"
echo "dynamic y chunk por defecto"
./pmtv-OpenMP 15360
export OMP_SCHEDULE="dynamic,1"
echo "dynamic y chunk 1"
./pmtv-OpenMP 15360
export OMP_SCHEDULE="dynamic,64"
echo "dynamic y chunk 64"
./pmtv-OpenMP 15360
export OMP_SCHEDULE="guided"
echo "guided y chunk por defecto"
./pmtv-OpenMP 15360
export OMP_SCHEDULE="guided,1"
echo "guided y chunk 1"
./pmtv-OpenMP 15360
export OMP_SCHEDULE="guided,64"
echo "guided y chunk 64"
./pmtv-OpenMP 15360
```

Tabla 3 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño N= 15360 , 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0.039151871	0.037052428	0.032117923
1	0.036267017	0.037214105	0.037564018
64	0.032415683	0.029451097	0.039031605
Chunk	**Static**	**Dynamic**	**Guided**
por defecto	0.040047094	0.033801220	0.034706016
1	0.034167702	0.038777399	0.042534261
64	0.032207792	0.029457848	0.036508310



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.

CAPTURA CÓDIGO FUENTE: pmm-secuencial.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(int argc, char **argv)
{
  unsigned i, j, k;
  struct timespec cgt1,cgt2; double ncgt;

  if(argc < 2){
    fprintf(stderr, "./pmm-secuencial [TAM]\n");
    exit(-1);
  }

  unsigned int N = atoi(argv[1]);

  int **a, **b, **c;
  a = (int **) malloc(N*sizeof(int*));
  b = (int **) malloc(N*sizeof(int*));
  c = (int **) malloc(N*sizeof(int*));
  for (i=0; i<N; i++){
    a[i] = (int *) malloc(N*sizeof(int));
    b[i] = (int *) malloc(N*sizeof(int));
    c[i] = (int *) malloc(N*sizeof(int));
  }

  // Inicializamos las matrices
  for (i=0; i<N; i++){
    for (j=0; j<N; j++){
      a[i][j] = 0;
```

```
    b[i][j] = 2;
    c[i][j] = 2;
  }
 }

 clock_gettime(CLOCK_REALTIME,&cgt1);

 // Multiplicacion
 for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    for (k=0; k<N; k++)
      a[i][j] += b[i][k] * c[k][j];

 clock_gettime(CLOCK_REALTIME,&cgt2);

 ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+( double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

 if(N<10){
  //Pintamos la matriz
  printf("Matriz b:\n");
  for(i=0; i<N; i++){
    for(j=0; j<N; j++){
      printf("%d ", b[i][j]);
    }
    printf("\n");
  }

  printf("Matriz c:\n");
  for(i=0; i<N; i++){
    for(j=0; j<N; j++){
      printf("%d ", c[i][j]);
    }
    printf("\n");
  }

  //Pintamos la matriz solucion
  printf("Matriz a:\n");
  for(i=0; i<N; i++){
    for(j=0; j<N; j++){
      printf("%d ", a[i][j]);
    }
    printf("\n");
  }
 }

 // Pitamos la primera y la ultima linea de la matriz resultante
 printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",ncgt,a[0][0],a[N-1][N-1]);

 // Liberamos la memoria
 for (i=0; i<N; i++)
 {
  free(a[i]);
  free(b[i]);
  free(c[i]);
 }
 free(a);
 free(b);
```

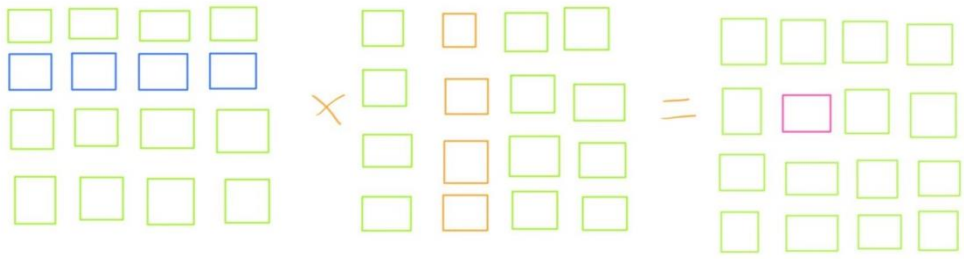
```
  free(c);

  return 0;
}
```

CAPTURAS DE PANTALLA:

```
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$gcc -O2 pmm-secuencial.c -o pmm -lrt
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$./pmm 10
Tiempo = 0.000001900      Primera = 40    Ultima=40
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$./pmm 20
Tiempo = 0.000006400      Primera = 80    Ultima=80
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$./pmm 50
Tiempo = 0.000097000      Primera = 200   Ultima=200
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$./pmm 100
Tiempo = 0.000798200      Primera = 400   Ultima=400
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$
```

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:



CAPTURA CÓDIGO FUENTE: pmm-OpenMP.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#ifdef _OPENMP
  #include <omp.h>
#else
  #define omp_get_thread_num() 0
  #define omp_get_num_threads() 1
  #define omp_set_num_threads(omp_get_num_procs())
  #define omp_in_parallel() 0
  #define omp_set_dynamic(int)
#endif

int main(int argc, char **argv)
{
  unsigned i, j, k;
  double t1, t2, final;
```

```
 if(argc < 2){
  fprintf(stderr, "./pmm-secuencial [TAM]\n");
  exit(-1);
 }

 unsigned int N = atoi(argv[1]);

 int **a, **b, **c;
 a = (int **) malloc(N*sizeof(int*));
 b = (int **) malloc(N*sizeof(int*));
 c = (int **) malloc(N*sizeof(int*));
 for (i=0; i<N; i++){
  a[i] = (int *) malloc(N*sizeof(int));
  b[i] = (int *) malloc(N*sizeof(int));
  c[i] = (int *) malloc(N*sizeof(int));
 }


// Inicializamos las matrices
 #pragma omp parallel for private(j)
  for (i=0; i<N; i++){
   for (j=0; j<N; j++){
    a[i][j] = 0;
    b[i][j] = 2;
    c[i][j] = 2;
   }
  }
  t1 = omp_get_wtime();

  // Multiplicacion
 #pragma omp parallel for private(k,j)
  for (i=0; i<N; i++)
   for (j=0; j<N; j++)
    for (k=0; k<N; k++)
     a[i][j] += b[i][k] * c[k][j];

  t2 = omp_get_wtime();
 final = t2 - t1;

 if(N<10){
  //Pintamos la matriz
  printf("Matriz b:\n");
  for(i=0; i<N; i++){
   for(j=0; j<N; j++){
    printf("%d ", b[i][j]);
   }
   printf("\n");
  }

  printf("Matriz c:\n");
  for(i=0; i<N; i++){
   for(j=0; j<N; j++){
    printf("%d ", c[i][j]);
   }
   printf("\n");
  }
```

```
  //Pintamos la matriz solucion
  printf("Matriz a:\n");
  for(i=0; i<N; i++){
    for(j=0; j<N; j++){
      printf("%d ", a[i][j]);
    }
    printf("\n");
  }
 }

 // Pitamos la primera y la ultima linea de la matriz resultante
 printf("Tiempo = %11.9f\t Primera = %d\t Ultima=%d\n",final,a[0][0],a[N-1][N-1]);

 // Liberamos la memoria
 for (i=0; i<N; i++)
 {
  free(a[i]);
  free(b[i]);
  free(c[i]);
 }
 free(a);
 free(b);
 free(c);

 return 0;
}
```

CAPTURAS DE PANTALLA:

```
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$gcc -O2 -fopenmp pmm-OpenMP.c -o pmm-OpenMP -lrt
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$./pmm-OpenMP 10
Tiempo = 0.000006000     Primera = 40    Ultima=40
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$./pmm-OpenMP 20
Tiempo = 0.000008000     Primera = 80    Ultima=80
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$./pmm-OpenMP 50
Tiempo = 0.000073000     Primera = 200   Ultima=200
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$./pmm-OpenMP 100
Tiempo = 0.000419000     Primera = 400   Ultima=400
[Paula Ruiz Garcia Paula@Pompitas:~/AC/P3/Codigo] 2018-05-13 Sunday
$
```

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.

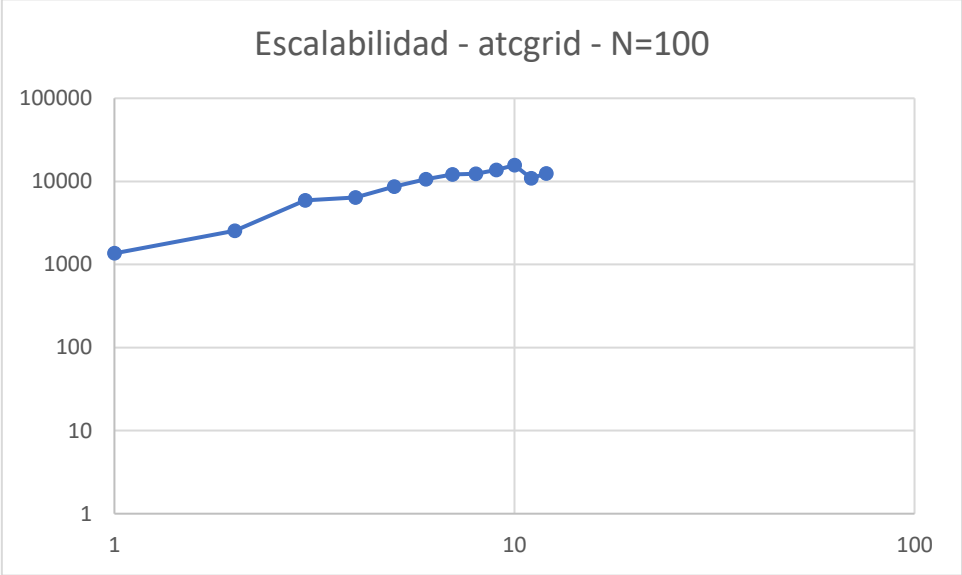
ESTUDIO DE ESCALABILIDAD EN atcgrid:

SCRIPT: pmm-OpenMP_atcgrid.sh

```
#!/bin/bash
echo "secuencial"
./pmm-secuencial 750
./pmm-secuencial 1000
```

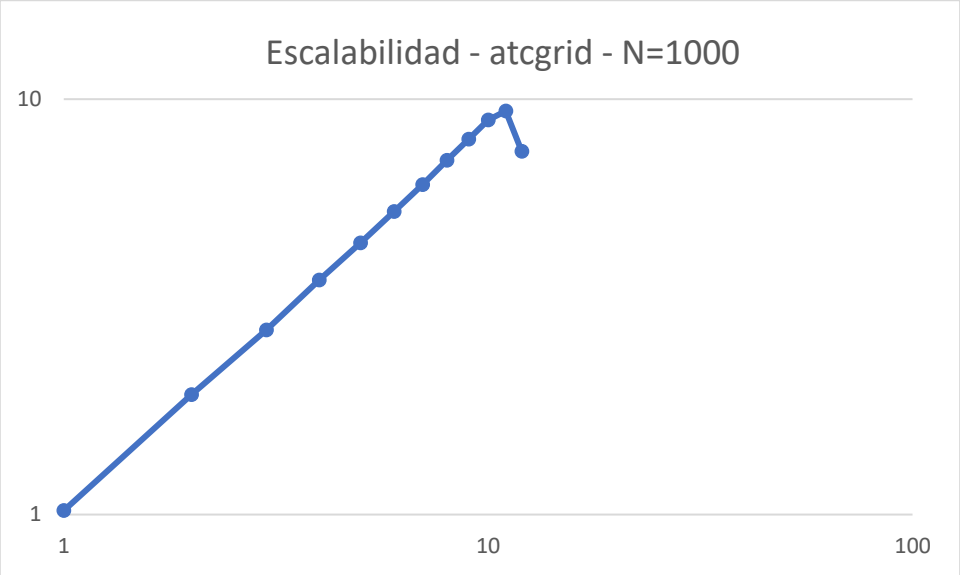
```
for ((i=1; i<13; i= i + 1))
do
echo $i "threads"
export OMP_NUM_THREADS=$i
./pmm-OpenMP 100
./pmm-OpenMP 1000
done
```

Valor	Threads	Tiempo	Ganancia
100	1	0.00277696	1360.91575
100	2	0.00149114	2534.45009
100	3	0.0006415	5891.24053
100	4	0.00058904	6415.88628
100	5	0.00043536	8680.68862
100	6	0.00035591	10618.5544
100	7	0.00031297	12075.4948
100	8	0.00030576	12360.2063
100	9	0.00027434	13775.8844
100	10	0.00024136	15658.2278
100	11	0.00034566	10933.2763
100	12	0.00030435	12417.3881



Valor	Threads	Tiempo	Ganancia
1000	1	9.50593665	1.02218699
1000	2	5.00330049	1.94208698
1000	3	3.49342096	2.78146975
1000	4	2.64696709	3.67093522
1000	5	2.15669644	4.50542995
1000	6	1.81118619	5.36490659
1000	7	1.55969513	6.22996413

1000	8	1.36466075	7.12033722
1000	9	1.21343735	8.00770202
1000	10	1.0913046	8.90387954
1000	11	1.03775846	9.36330093
1000	12	1.29881865	7.48129445

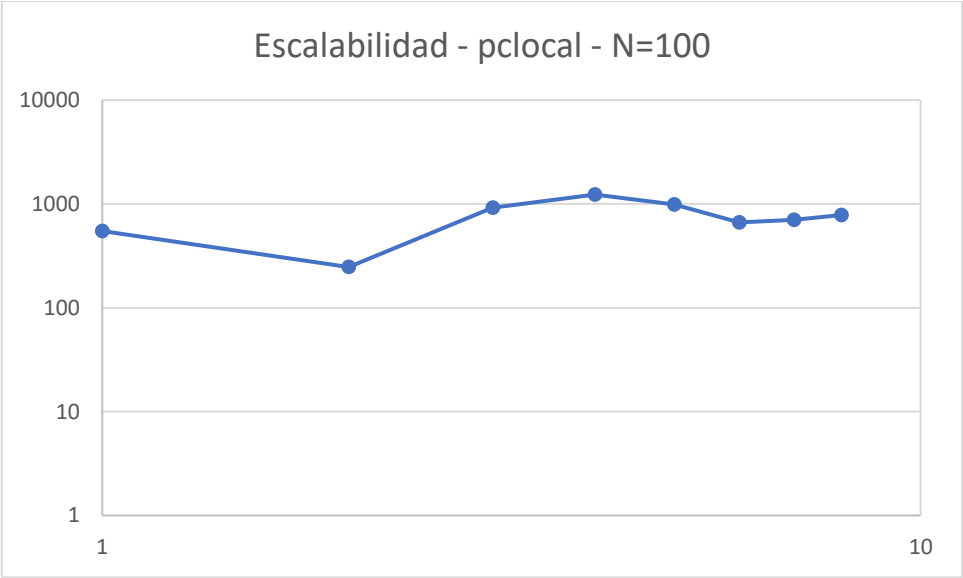


ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pclocal.sh

```
#!/bin/bash

echo "secuencial"
./pmm-secuencial 750
./pmm-secuencial 1000

for ((i=1; i<9; i= i + 1))
do
echo $i "threads"
export OMP_NUM_THREADS=$i
./pmm-OpenMP 100
./pmm-OpenMP 1000
done
```

Valor	Threads	Tiempo	Ganancia
100	1	0.000935	547.058503
100	2	0.002073	246.743705
100	3	0.000554	923.284657
100	4	0.000416	1229.56659
100	5	0.000519	985.548555
100	6	0.000768	666.015234
100	7	0.00073	700.684521
100	8	0.000655	780.915573



Valor	Threads	Tiempo	Ganancia
1000	1	1.997379	0.762337593
1000	2	1.258036	1.210360514
1000	3	1.352634	1.125712573
1000	4	2.068047	0.736287473
1000	5	2.05459	0.741109954
1000	6	2.048822	0.743196383
1000	7	2.024909	0.751973101
1000	8	2.04	0.746410343