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): César Muñoz Reinoso
Grupo de prácticas: Grupo 2
Fecha de entrega:
Fecha evaluación en clase:

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`

```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\n");
    exit(-1);
  }
  n = atoi(argv[1]);
  x = atoi(argv[2]);
  if(n>20) n=20;
  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:

```
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-04-20 viernes
$gcc -fopenmp -O2 if-clauseModificado.c -o if-clauseModificado
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-04-20 viernes
$./if-clauseModificado 15 1
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 0 suma de a[4]=4 sumalocal=10
thread 0 suma de a[5]=5 sumalocal=15
thread 0 suma de a[6]=6 sumalocal=21
thread 0 suma de a[7]=7 sumalocal=28
thread 0 suma de a[8]=8 sumalocal=36
thread 0 suma de a[9]=9 sumalocal=45
thread 0 suma de a[10]=10 sumalocal=55
thread 0 suma de a[11]=11 sumalocal=66
thread 0 suma de a[12]=12 sumalocal=78
thread 0 suma de a[13]=13 sumalocal=91
thread 0 suma de a[14]=14 sumalocal=105
thread master=0 imprime suma=105
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-04-20 viernes
$./if-clauseModificado 10 5
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 4 suma de a[8]=8 sumalocal=8
thread 4 suma de a[9]=9 sumalocal=17
thread 3 suma de a[6]=6 sumalocal=6
thread 3 suma de a[7]=7 sumalocal=13
thread 0 suma de a[0]=0 sumalocal=0
thread 0 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=45
```

RESPUESTA: Con la función usada, num_threads, podemos elegir en cada ejecución el número de threads que queramos utilizar. Ej(1 thread,5 threads,...)

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

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

RESPUESTA:

Cuando la planificación es estática, las ejecuciones del bucle se van ejecutando

secuencialmente por los threads en orden (0-15 thread 1, 16-30 thread 2, …). En cambio cuando la planificación es guiada, se van ejecutando sececuencialmente como en estática pero las iteracciones no van en orden desde 0. Cuando es dinámica, se van ejecutando por orden de llegada, no se sigue ningún orden.

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`

```
int main(int argc, char ** argv){
  int i,n=200, chunk;
  int a[n], suma=0;
  omp_sched_t kind;
  if(argc<3){
    fprintf(stderr,"\nFalta 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 += a[i];
      printf("thread %d suma a[%d] suma=%d\n", omp_get_thread_num
(),i,suma);

    }
    omp_get_schedule(&kind, &chunk);
    #pragma omp single
    printf("\nLa variable dyn-var:%d\n La variable nthreads-var:%d\n La
variable thread-limit-var:%d\n La variable run-sched-var:%d \n
\n",omp_get_dynamic(),omp_get_max_threads(),omp_get_thread_limit(),kind);
  }
      printf("Fuera de 'parallel for' suma=%d\n",suma);
      printf("\nLa variable dyn-var:%d\n La variable nthreads-var:%d\n
La variable thread-limit-var:%d\n La variable run-sched-var:%d \n
\n",omp_get_dynamic(),omp_get_max_threads(),omp_get_thread_limit(),kind);
```

CAPTURAS DE PANTALLA:

```
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-04-27 viernes
$gcc -fopenmp -O2 scheduled-clauseModificado.c -o scheduled-clauseModificado
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-04-27 viernes
$export OMP_DYNAMIC=TRUE
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-04-27 viernes
$export OMP_NUM_THREADS=6
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-04-27 viernes
$export OMP_THREAD_LIMIT=8
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-04-27 viernes
$export OMP_SCHEDULE="dynamic"
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-04-27 viernes
$./scheduled-clauseModificado 16 2
thread 1 suma a[0] suma=0
thread 1 suma a[1] suma=1
thread 1 suma a[8] suma=9
thread 1 suma a[9] suma=18
thread 1 suma a[10] suma=28
thread 1 suma a[11] suma=39
thread 1 suma a[12] suma=51
thread 1 suma a[13] suma=64
thread 1 suma a[14] suma=78
thread 1 suma a[15] suma=93
thread 2 suma a[2] suma=2
thread 2 suma a[3] suma=5
thread 0 suma a[6] suma=6
thread 0 suma a[7] suma=13
thread 3 suma a[4] suma=4
thread 3 suma a[5] suma=9

La variable dyn-var:1
 La variable nthreads-var:6
 La variable thread-limit-var:8
 La variable run-sched-var:2

Fuera de 'parallel for' suma=93

La variable dyn-var:1
 La variable nthreads-var:6
 La variable thread-limit-var:8
```

RESPUESTA:

Aunque no nos encontremos en la misma región de parallel, los valores son los mismos, ya que el número de threads, limite de threads y el tipo de planificación es el mismo.

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

CAPTURA CÓDIGO FUENTE: `scheduled-clauseModificado4.c`

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

  if(argc<3){
    fprintf(stderr,"\nFalta 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 for firstprivate(suma) \
          lastprivate(suma) schedule(dynamic,chunk)
    for(i=0;i<n;i++){
      suma += a[i];
      printf("thread %d suma a[%d] suma=%d\n", omp_get_thread_num
(),i,suma);

    }
    #pragma omp single
    printf("\nNumero de threads:%d \n Numero de procs:%d \n  Zona
parallel:%d\n\n",omp_get_num_threads(),omp_get_num_procs
(),omp_in_parallel());
  }
      printf("Fuera de 'parallel for' suma=%d\n",suma);
      printf("\nNumero de threads:%d \n Numero de procs:%d \n  Zona
parallel:%d\n\n",omp_get_num_threads(),omp_get_num_procs
(),omp_in_parallel());
}
```

CAPTURAS DE PANTALLA:

```
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-04-20 viernes
$gcc -fopenmp -O2 scheduled-clauseModificado4.c -o scheduled-clauseModificado4
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-04-20 viernes
$./scheduled-clauseModificado4 16 2
thread 2 suma a[0] suma=0
thread 2 suma a[1] suma=1
thread 0 suma a[0] suma=0
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
thread 0 suma a[7] suma=28
thread 0 suma a[8] suma=36
thread 1 suma a[0] suma=0
thread 1 suma a[10] suma=55
thread 3 suma a[0] suma=0
thread 3 suma a[12] suma=78
thread 3 suma a[13] suma=91
thread 3 suma a[14] suma=105
thread 3 suma a[15] suma=120
thread 0 suma a[9] suma=45
thread 1 suma a[11] suma=66
thread 2 suma a[2] suma=3

Numero de threads:4
 Numero de procs:4
  Zona parallel:1

Fuera de 'parallel for' suma=120

Numero de threads:1
 Numero de procs:4
  Zona parallel:0
```

RESPUESTA:

El número de threads que se están usando en la región paralela cambia de 4 dentro a 1 fuera, ya que dentro de la region paralela tenemos 4 threads trabajando y fuera solo esta actuando el thread master(0).

El número de procesadores disponibles no varia, es 4 fuera y dentro de la región parallel. La función omp_in_parallel() devuelve true si se encuentra dentro de la región parallel y false si se encuentra fuera.

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`

```
    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 += a[i];
      printf("thread %d suma a[%d] suma=%d\n", omp_get_thread_num
(),i,suma);

    }
    #pragma omp single
    {
      printf("ANTES DE LA MODIFICACIÓN");
      omp_get_schedule(&kind, &chunk);
      printf("\nLa variable dyn-var:%d\n La variable nthreads-var:%d\n
La variable thread-limit-var:%d\n La variable run-sched-var:%d \n
\n",omp_get_dynamic(),omp_get_max_threads(),omp_get_thread_limit(),kind);
      omp_set_dynamic(9);
      omp_set_num_threads(15);
      omp_set_schedule(3, 3);
      printf("DESPUÉS DE LA MODIFICACIÓN");
      omp_get_schedule(&kind, &chunk);
      printf("\nLa variable dyn-var:%d\n La variable nthreads-var:%d\n
La variable thread-limit-var:%d\n La variable run-sched-var:%d \n
\n",omp_get_dynamic(),omp_get_max_threads(),omp_get_thread_limit(),kind);
    }
```

CAPTURAS DE PANTALLA:

```
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-04-27 viernes
$gcc -fopenmp -O2 scheduled-clauseModificado5.c -o scheduled-clauseModificado5
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-04-27 viernes
$./scheduled-clauseModificado5 16 1
thread 2 suma a[0] suma=0
thread 2 suma a[4] suma=4
thread 2 suma a[5] suma=9
thread 2 suma a[6] suma=15
thread 1 suma a[1] suma=1
thread 1 suma a[8] suma=9
thread 1 suma a[9] suma=18
thread 1 suma a[10] suma=28
thread 1 suma a[11] suma=39
thread 1 suma a[12] suma=51
thread 1 suma a[13] suma=64
thread 1 suma a[14] suma=78
thread 3 suma a[2] suma=2
thread 0 suma a[3] suma=3
thread 1 suma a[15] suma=93
thread 2 suma a[7] suma=22
ANTES DE LA MODIFICACIÓN
La variable dyn-var:1
 La variable nthreads-var:6
 La variable thread-limit-var:8
 La variable run-sched-var:2

DESPUÉS DE LA MODIFICACIÓN
La variable dyn-var:1
 La variable nthreads-var:15
 La variable thread-limit-var:8
 La variable run-sched-var:3
```

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`

```
vector = (int*) malloc(N*sizeof(int));
matrix = (int**) malloc(N*sizeof(int*));
salida = (int*) malloc(N*sizeof(int));

if((vector == NULL) || (matrix == NULL) || (salida == NULL)){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
}

// RESERVAMOS MEMORIA PARA LA MATRIZ
for (int i = 0;i < N;i++){
        matrix[i] = (int*)malloc(N*sizeof(int));
        if(matrix[i] == NULL) perror("Error: ");
}

//INICIALIZAMOS MATRIZ Y VECTOR
for(int i = 0;i < N;i++){
        for(int j = 0;j <= i;j++){
                        matrix[i][j]=i+j+3;
        }
}

for(int i = 0;i < N;i++){
        vector[i] = i;
}

for(int i = 0;i < N;i++){
        for(int j = 0;j < N;j++){
                printf("%i  ",matrix[i][j]);
        }
        printf("\n");
}

clock_gettime(CLOCK_REALTIME,&cgt1);

//MULTIPLICACIÓN
for(int i = 0; i< N; i++){
        for(int j = 0; j< N; j++){
                suma += matrix[i][j]*vector[j];
        }
        salida[i] = suma;
        suma = 0;
}
clock_gettime(CLOCK_REALTIME,&cgt2);

//SALIDA
printf("Vector resultante: ");
for(int i = 0;i < N;i++){
                printf("%d ",salida[i]);
        }
```

CAPTURAS DE PANTALLA:

```
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-05-03 jueves
$gcc -O2 pmtv-secuencial.c -o pmtv-secuencial
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-05-03 jueves
$./pmtv-secuencial 15
3  0  0  0  0  0  0  0  0  0  0  0  0  0  0
4  5  0  0  0  0  0  0  0  0  0  0  0  0  0
5  6  7  0  0  0  0  0  0  0  0  0  0  0  0
6  7  8  9  0  0  0  0  0  0  0  0  0  0  0
7  8  9  10  11  0  0  0  0  0  0  0  0  0  0
8  9  10  11  12  13  0  0  0  0  0  0  0  0  0
9  10  11  12  13  14  15  0  0  0  0  0  0  0  0
10  11  12  13  14  15  16  17  0  0  0  0  0  0  0
11  12  13  14  15  16  17  18  19  0  0  0  0  0  0
12  13  14  15  16  17  18  19  20  21  0  0  0  0  0
13  14  15  16  17  18  19  20  21  22  23  0  0  0  0
14  15  16  17  18  19  20  21  22  23  24  25  0  0  0
15  16  17  18  19  20  21  22  23  24  25  26  27  0  0
16  17  18  19  20  21  22  23  24  25  26  27  28  29  0
17  18  19  20  21  22  23  24  25  26  27  28  29  30  31
Vector resultante: 0 5 20 50 100 175 280 420 600 825 1100 1430 1820 2275 2800 Tiempo(seg.):0.000008735
  Tamaño:15
 salida[0]=0 / / salida[14]=2800
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-05-03 jueves
$./pmtv-secuencial 20
3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
4  5  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
5  6  7  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
6  7  8  9  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
7  8  9  10  11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
8  9  10  11  12  13  0  0  0  0  0  0  0  0  0  0  0  0  0  0
9  10  11  12  13  14  15  0  0  0  0  0  0  0  0  0  0  0  0  0
10  11  12  13  14  15  16  17  0  0  0  0  0  0  0  0  0  0  0  0
11  12  13  14  15  16  17  18  19  0  0  0  0  0  0  0  0  0  0  0
12  13  14  15  16  17  18  19  20  21  0  0  0  0  0  0  0  0  0  0
13  14  15  16  17  18  19  20  21  22  23  0  0  0  0  0  0  0  0  0
14  15  16  17  18  19  20  21  22  23  24  25  0  0  0  0  0  0  0  0
15  16  17  18  19  20  21  22  23  24  25  26  27  0  0  0  0  0  0  0
16  17  18  19  20  21  22  23  24  25  26  27  28  29  0  0  0  0  0  0
17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  0  0  0  0  0
18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  0  0  0  0
19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  0  0  0
20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  0  0
21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  0
22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41
Vector resultante: 0 5 20 50 100 175 280 420 600 825 1100 1430 1820 2275 2800 3400 4080 4845 5700 6650 Tiempo(seg.):0.000001779
  Tamaño:20
 salida[0]=0 / / salida[19]=6650
```

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

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

RESPUESTA:

Como podemos ver, con la planificación guided, el tiempo por ejecución es menor en todos los casos, por lo que es la que ofrece mejor prestaciones.

Dentro de la asignación de planificación static, cada thread ejecuta una operación de suma y otra de multiplicación por cada interacción, una por cada fila de la matriz. Si implementamos una asignación dynamic o guided, los chunks por iteracción son menores.

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

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

  vector = (int*) malloc(N*sizeof(int));
  matrix = (int**) malloc(N*sizeof(int*));
  salida = (int*) malloc(N*sizeof(int));

  if((vector == NULL) || (matrix == NULL) || (salida == NULL)){
          printf("Error en la reserva de espacio para los vectores\n");
          exit(-2);
  }

  // RESERVAMOS MEMORIA PARA LA MATRIZ
  #pragma omp parallel for
  for (int i = 0;i < N;i++){
          matrix[i] = (int*)malloc(N*sizeof(int));
          if(matrix[i] == NULL) perror("Error: ");
  }

  //INICIALIZAMOS MATRIZ Y VECTOR
  for(int i = 0;i < N;i++){
          for(int j = 0;j <= i;j++){
                          matrix[i][j]=i+j+3;
          }
  }

  for(int i = 0;i < N;i++){
          vector[i] = i;
  }



  clock_gettime(CLOCK_REALTIME,&cgt1);

  //MULTIPLICACIÓN
  #pragma omp parallel for schedule(runtime) reduction(+:suma)
  for(int i = 0; i< N; i++){
          for(int j = 0; j< N; j++){
                  suma += matrix[i][j]*vector[j];
          }
          salida[i] = suma;
          suma = 0;
  }
  clock_gettime(CLOCK_REALTIME,&cgt2);


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

  printf("Tiempo(seg.):%11.9f\n  Tamaño:%u\n salida[0]=%d / / salida[%d]=%d \n", ncgt,N,salida
l,salida[N-1]);
```

DESCOMPOSICIÓN DE DOMINIO:

La descomposición de dominio se basa en la multiplicación de cada fila por columna y su posterior suma, por lo que cada thread realiza una multiplicación y suma de cada multiplicación.

CAPTURAS DE PANTALLA:

```
$echo 'sh pmvt-OpenMP_PCaula.sh pmtv-OpenMP'| qsub -q ac
76682.atcgrid
[César Muñoz Reinoso E2estudiante17@atcgrid:~] 2018-05-03 jueves
$echo 'sh pmvt-OpenMP_PCaula.sh pmtv-OpenMP'| qsub -q ac
76683.atcgrid
[César Muñoz Reinoso E2estudiante17@atcgrid:~] 2018-05-03 jueves
$cat STDIN.76682
cat: STDIN.76682: No such file or directory
[César Muñoz Reinoso E2estudiante17@atcgrid:~] 2018-05-03 jueves
$cat STDIN.o76682
Tiempo(seg.):0.065167924
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
Tiempo(seg.):0.068260557
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
Tiempo(seg.):0.069198443
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
Tiempo(seg.):0.076615687
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
Tiempo(seg.):0.076764113
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
Tiempo(seg.):0.066850966
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
Tiempo(seg.):0.063668238
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
Tiempo(seg.):0.073153814
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
Tiempo(seg.):0.059829928
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
[César Muñoz Reinoso E2estudiante17@atcgrid:~] 2018-05-03 jueves
$cat STDIN.o76683
Tiempo(seg.):0.065158806
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
Tiempo(seg.):0.068350367
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
Tiempo(seg.):0.069675418
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
Tiempo(seg.):0.074888500
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
Tiempo(seg.):0.077260812
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
Tiempo(seg.):0.067354679
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
Tiempo(seg.):0.063504288
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
Tiempo(seg.):0.059674776
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
Tiempo(seg.):0.065340831
  Tamaño:19200
 salida[0]=0 / / salida[19199]=1249886592
```

SCRIPT: pmvt-OpenMP_PCaula.sh

```
#!/bin/bash


export OMP_SCHEDULE="static"
./$1 19200
export OMP_SCHEDULE="static,1"
./$1 19200
export OMP_SCHEDULE="static,64"
./$1 19200

export OMP_SCHEDULE="dynamic"
./$1 19200
export OMP_SCHEDULE="dynamic,1"
./$1 19200
export OMP_SCHEDULE="dynamic,64"
./$1 19200

export OMP_SCHEDULE="guided"
./$1 19200
export OMP_SCHEDULE="guided,1"
./$1 19200
export OMP_SCHEDULE="guided,64"
./$1 19200
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

Tabla 3 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector r para vectores de tamaño N= 19200, 12 threads

Chunk	Static	Dynamic	Guided
por defecto	0.065167924	0.076615687	0.063668238
1	0.068260557	0.076764113	0.073153814
64	0.069198443	0.066850966	0.059829928
Chunk	**Static**	**Dynamic**	**Guided**
por defecto	0.065158806	0.074888500	0.063504288
1	0.068350367	0.077260812	0.059674776
64	0.069675418	0.067354679	0.065340831

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`

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

        matrix1 = (int**) malloc(N*sizeof(int*));
        matrix2 = (int**) malloc(N*sizeof(int*));
        salida = (int**) malloc(N*sizeof(int*));

        if((matrix1 == NULL) || (matrix2 == NULL) || (salida == NULL)){
                printf("Error en la reserva de espacio para los vectores\n");
                exit(-2);
        }

        // RESERVAMOS MEMORIA PARA LA MATRIZ
        for (int i = 0;i < N;i++){
                matrix1[i] = (int*)malloc(N*sizeof(int));
                matrix2[i] = (int*)malloc(N*sizeof(int));
                salida[i] = (int*)malloc(N*sizeof(int));
                if(matrix1[i] == NULL) perror("Error: ");
                if(matrix2[i] == NULL) perror("Error: ");
                if(salida[i] == NULL) perror("Error: ");
        }

        //INICIALIZAMOS MATRIZ Y VECTOR
        for(int i = 0;i < N;i++){
                for(int j = 0;j < N;j++){
                                matrix1[i][j]=i+j;
                                matrix2[i][j]=i+j;
                }
        }

        clock_gettime(CLOCK_REALTIME,&cgt1);

        //MULTIPLICACIÓN
        for(int i = 0; i< N; i++){
                for(int j = 0; j< N; j++){
                        for(int z=0; z<N; z++){
                                salida[i][j] += matrix1[i][z]*matrix2[z][i];
                        }
                }
        }
        clock_gettime(CLOCK_REALTIME,&cgt2);

        //SALIDA
        printf("Matriz resultante: ");
        for(int i = 0;i < N;i++){
                for(int j=0;j<N;j++){
                        printf("%d ",salida[i][j]);
                }
        }

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

CAPTURAS DE PANTALLA:

```
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-05-03 jueves
$gcc -O2 -fopenmp pmm-secuencial.c -o pmm-secuencial
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-05-03 jueves
$./pmm-secuencial
Faltan nº componentes del vector
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-05-03 jueves
$./pmm-secuencial 5
0  1  2  3  4
1  2  3  4  5
2  3  4  5  6
3  4  5  6  7
4  5  6  7  8


0  1  2  3  4
1  2  3  4  5
2  3  4  5  6
3  4  5  6  7
4  5  6  7  8
Matriz resultante: 30 30 30 30 30 55 55 55 55 55 90 90 90 90 90 135 135 135 135 135 190 190 190 190 190 Tiempo(seg.):0.000001884
  Tamaño:5
 salida[0][0]=30 / / salida[4][4]=190
```

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:

Como en la ejecución anterior, se realiza una multiplicación y una posterior suma de cada interacción del bucle por lo que cada thread ejecuta un número concreto de iteraciones, iniciando en el thread 0 con la interacción 0.

CAPTURA CÓDIGO FUENTE: `pmm-OpenMP.c`

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

matrix1 = (int**) malloc(N*sizeof(int*));
matrix2 = (int**) malloc(N*sizeof(int*));
salida = (int**) malloc(N*sizeof(int*));

if((matrix1 == NULL) || (matrix2 == NULL) || (salida == NULL)){
        printf("Error en la reserva de espacio para los vectores\n");
        exit(-2);
}

// RESERVAMOS MEMORIA PARA LA MATRIZ
for (int i = 0;i < N;i++){
        matrix1[i] = (int*)malloc(N*sizeof(int));
        matrix2[i] = (int*)malloc(N*sizeof(int));
        salida[i] = (int*)malloc(N*sizeof(int));
        if(matrix1[i] == NULL) perror("Error: ");
        if(matrix2[i] == NULL) perror("Error: ");
        if(salida[i] == NULL) perror("Error: ");
}

//INICIALIZAMOS MATRIZ Y VECTOR
#pragma omp parallel for
for(int i = 0;i < N;i++){
        for(int j = 0;j < N;j++){
                        matrix1[i][j]=i+j;
                        matrix2[i][j]=i+j;
        }
}

clock_gettime(CLOCK_REALTIME,&cgt1);

//MULTIPLICACIÓN
#pragma omp parallel for reduction(+:suma)
for(int i = 0; i< N; i++){
        for(int j = 0; j< N; j++){
                for(int z=0; z<N; z++){
                        salida[i][j] += matrix1[i][z]*matrix2[z][i];
                }
        }
}
clock_gettime(CLOCK_REALTIME,&cgt2);
```

CAPTURAS DE PANTALLA:

```
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-05-03 jueves
$gcc -O2 -fopenmp pmm-OpenMP.c -o pmm-OpenMP
[César Muñoz Reinoso cesar@cesar-X550CA:~/Escritorio/UNIVERSIDAD/SEGUNDO/2º Cuatrimestre/AC/Practicas/Archivos/BP3] 2018-05-03 jueves
$./pmm-OpenMP 5
0  1  2  3  4
1  2  3  4  5
2  3  4  5  6
3  4  5  6  7
4  5  6  7  8


0  1  2  3  4
1  2  3  4  5
2  3  4  5  6
3  4  5  6  7
4  5  6  7  8
Matriz resultante:
719716907 30 1879572542 1702061456 29836
1879572567 1769173926 6646937 134217783 17410
26575 67108954 132 90 1262248926
7561712 134217863 179 135 1845114642
1953722653 6649119 2069 190 2102464603
Tiempo(seg.):0.000011080
  Tamaño:5
 salida[0][0]=719716907 / / salida[4][4]=2102464603
```

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

OMP_NUM_THREADS=12
for ((N=100; N<1500; N=N+100))
do
        "./$1" $N
done
```

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pclocal.s

```
#!/bin/bash


OMP_NUM_THREADS=4
for ((N=100; N<1500; N=N+100))
do
        "./$1" $N

done
```