| | |
|---|---|
| 2º curso / 2º cuatr.<br>Grado Ing. Inform. | **Arquitectura de Computadores (AC)**<br>**Cuaderno de prácticas.**<br>**Bloque Práctico 3. Programación paralela III: Interacción con el entorno en OpenMP**<br><br>Estudiante (nombre y apellidos): Carmen García Romero<br>Grupo de prácticas: 2C-C1 Christian Morillas<br>Fecha de entrega: 17/05/21<br>Fecha evaluación en clase:18/05/21 |

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

## Ejercicios basados en los ejemplos del seminario práctico

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

**CAPTURA CÓDIGO FUENTE**: if-clauseModificado.c

if-clauseModificado.c

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

int main(int argc, char **argv)
{
  int i, n=20, tid, numero_threads;
  int a[n],suma=0,sumalocal;

  if(argc < 2) {
    fprintf(stderr,"[ERROR]-Falta iteraciones\n");
    exit(-1);
  }

  if(argc < 3) {
    fprintf(stderr,"[ERROR]-Falta num de threads\n");
    exit(-1);
  }

  numero_threads = atoi(argv[2]);
  n = atoi(argv[1]); if (n>20) n=20;
  for (i=0; i<n; i++) {
    a[i] = i;
  }


  #pragma omp parallel if(n>4) num_threads(numero_threads) 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);
    }
}
```

**CAPTURAS DE PANTALLA:**

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp3/ejer1  2021-04-28 miércoles
$gcc -O2 -fopenmp -o if-clauseModificado if-clauseModificado.c
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp3/ejer1  2021-04-28 miércoles
$./if-clauseModificado 7 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 master=0 imprime suma=21
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp3/ejer1  2021-04-28 miércoles
$./if-clauseModificado 7 3
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 thread 0 suma de a[2]=2 sumalocal=3
 thread 1 suma de a[3]=3 sumalocal=3
 thread 1 suma de a[4]=4 sumalocal=7
 thread 2 suma de a[5]=5 sumalocal=5
 thread 2 suma de a[6]=6 sumalocal=11
thread master=0 imprime suma=21
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/prac
ticas/bp3/ejer1  2021-04-28 miércoles
$./if-clauseModificado 7 4
 thread 0 suma de a[0]=0 sumalocal=0
 thread 0 suma de a[1]=1 sumalocal=1
 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 3 suma de a[6]=6 sumalocal=6
thread master=0 imprime suma=21
```

**RESPUESTA**:
Ante un mismo número de iteraciones, al aumentar el número de hebras vemos que el valor de las sumas locales va disminuyendo.

**2.** Rellenar la Tabla 1 (se debe poner en la tabla el id del *thread* que ejecuta cada iteración) usando scheduler-clause.c con tres *threads* (0,1,2) y un número de iteraciones de 16 (0 a 15 en la tabla). Con este ejercicio se pretende comparar distintas alternativas de planificación de bucles. Se van a usar distintos tipos (static, dynamic, guided), modificadores (monotonic y nonmonotonic) y tamaños de chunk (x = 1, 2 y 4).

**Tabla 1 .** Tabla schedule. Rellenar esta tabla ejecutando scheduler-clause.c asignando previamente a la variable de entorno OMP_SCHEDULE los valores que se indican en la tabla (por ej.: export OMP_SCHEDULE="nonmonotonic:static,2). En la segunda fila, 1, 2 4 representan el tamaño del chunk

| Iteración | "monotonic:static,x" | | | "nonmonotonic:static,x" | | | "monotonic:dynamic,x" | | | "monotonic:guided,x" | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | x=1 | x=2 | x=4 | x=1 | x=2 | x=4 | x=1 | x=2 | x=4 | x=1 | x=2 | x=4 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 1 | 2 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| 7 | 1 | 2 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| 8 | 2 | 2 | 2 | 1 | 1 | 2 | 0 | 0 | 1 | 0 | 2 | 2 |
| 9 | 2 | 2 | 2 | 1 | 1 | 2 | 0 | 0 | 1 | 0 | 2 | 2 |
| 10 | 2 | 1 | 2 | 1 | 1 | 2 | 0 | 0 | 1 | 2 | 1 | 2 |
| 11 | 2 | 1 | 2 | 2 | 1 | 2 | 0 | 0 | 1 | 2 | 1 | 2 |
| 12 | 2 | 1 | 1 | 2 | 2 | 1 | 0 | 1 | 2 | 2 | 1 | 1 |
| 13 | 1 | 1 | 1 | 2 | 2 | 1 | 0 | 1 | 2 | 2 | 1 | 1 |
| 14 | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 | 2 | 2 | 1 | 2 | 2 | 2 | 1 | 1 | 1 |

Destacar las diferencias entre las 4 alternativas de planificación de la tabla, en particular, las que hay entre static, dynamic y guided y las diferencias entre usar monotonic y nonmonotonic.

**RESPUESTA**: Con la cláusula static el reparto es más equitativo, ya que las iteraciones que realiza cada thread dependen únicamente del chunk que se le pasa como argumento. SIn embargo, con dynamic como con guided no sabemos exactamente cuántas iteración hará cada thread al ser otro tipo de asignación.
Cuando se usa monotonic cada hebras se ejecutarán según el chunk indicado, en orden lógico creciente.
Usando nonmonotonic, no hay orden a priori de que iteraciones se asignará primero.

**3.** ¿Qué valor por defecto usa OpenMP para chunk y modifier con static, dynamic y guided? Explicar qué ha hecho para contestar a esta pregunta.

- static: no tiene var por defecto
- dynamic → 1
- guided → 1
- Estos valores se pueden obtener , asignando la alternativa correspondiente a la variable de entorno export OMP_SCHEDULE, y ejecutando la siguiente instrucción:

```
omp_get_schedule(&schedule_type, &chunk_value);
printf(" run-sched-var: %d\n ;  chunk:%d\n",  schedule_type, chunk_value);
```

**4.** 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

scheduled-clauseModificado.c

```
{
  int i, n=200, chunk, a[n], suma=0;
  omp_sched_t schedule_type;
  int chunk_value, cnt=0;


  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 for firstprivate(suma)  \
  lastprivate(suma) schedule(dynamic, chunk)
  for (i=0; i<n; i++) {
    suma = suma + a[i];

     if(i == 0 ){
       printf("\nDentro de 'parallel for' suma=%d\n", suma);


       omp_get_schedule(&schedule_type, &chunk_value);
       printf(" dyn-var: %d\n nthreads-var: %d\n thread-limit-var: %d\n run-sched-var: %d\n \
       chunk:%d\n", \
            omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(),
            schedule_type, chunk_value);

      printf(" static=1 ;  dynamic= 2 ; guided = 3; autor = 4\n");
      }
    }

    printf("\nFuera de 'parallel for' suma=%d\n", suma);

    omp_get_schedule(&schedule_type, &chunk_value);
    printf(" dyn-var: %d\n nthreads-var: %d\n thread-limit-var: %d\n run-sched-var: %d\n \
         chunk:%d\n", \
         omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(),
         schedule_type, chunk_value);
    printf(" static=1 ;  dynamic= 2 ; guided = 3; autor = 4\n");
}
```

**CAPTURAS DE PANTALLA:**

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp3/e
jer4  2021-05-17 lunes
$export OMP_DYNAMIC= FALSE
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp3/e
jer4  2021-05-17 lunes
$export OMP_DYNAMIC=FALSE
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp3/e
jer4  2021-05-17 lunes
$export OMP_NUM_THREADS=4
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp3/e
jer4  2021-05-17 lunes
$export OMP_SCHEDULE="dynamic,3"
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp3/e
jer4  2021-05-17 lunes
$./scheduled-clauseModificado  12 4

Dentro de 'parallel for' suma=0
 dyn-var: 0
 nthreads-var: 4
 thread-limit-var: 2147483647
 run-sched-var: 2
        chunk:3
 static=1 ;  dynamic= 2 ; guided = 3; autor = 4

Fuera de 'parallel for' suma=38
 dyn-var: 0
 nthreads-var: 4
 thread-limit-var: 2147483647
 run-sched-var: 2
          chunk:3
 static=1 ;  dynamic= 2 ; guided = 3; autor = 4
```

**RESPUESTA**: Nos devuelve los mismo valores dentro y fuera de la región paralela.

**5.** 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-clauseModificado5.c

scheduled-clauseModificado5.c

```
  omp_sched_t schedule_type;
  int chunk_value, cnt=0;

  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 for firstprivate(suma)  \
  lastprivate(suma) schedule(dynamic, chunk)
  for (i=0; i<n; i++) {
    suma = suma + a[i];

     if(i == 0){
       printf("\nDentro de 'parallel for' suma=%d\n", suma);
       omp_get_schedule(&schedule_type, &chunk_value);
       printf(" dyn-var: %d\n nthreads-var: %d\n thread-limit-var: %d\n run-sched-var: %d\n \
       chunk:%d\n", \
            omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(),
            schedule_type, chunk_value);

        printf(" omp_get_threads: %d, get_num_procs: %d, in_parallel: %d\n", \
              omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());

        printf(" static=1 ;  dynamic= 2 ; guided = 3; autor = 4\n");
      }
    }

    printf("\nFuera de 'parallel for' suma=%d\n", suma);
    printf(" static=1 ;  dynamic= 2 ; guided = 3; autor = 4\n");
    omp_get_schedule(&schedule_type, &chunk_value);
    printf(" dyn-var: %d\n nthreads-var: %d\n thread-limit-var: %d\n run-sched-var: %d \
         chunk:%d\n", \
         omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(),
         schedule_type, chunk_value);

     printf(" omp_get_threads: %d, get_num_procs: %d, in_parallel: %d\n", \
          omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());

     printf(" static=1 ;  dynamic= 2 ; guided = 3; autor = 4\n");
}
```

**CAPTURAS DE PANTALLA:**

```
CarmenGarciaRomero c1estudiante11@carmen:~/Escritorio/Facultad/2/2Cuatri/AC/practicas/bp3/ejer5
 2021-05-04 martes
$./scheduled-clauseModificado5 5 2

Dentro de 'parallel for' suma=0
 dyn-var: 0
 nthreads-var: 12
 thread-limit-var: 2147483647
 run-sched-var: 2
        chunk:1
 omp_get_threads: 12, get_num_procs: 12, in_parallel: 1
 static=1 ;  dynamic= 2 ; guided = 3; autor = 4

Fuera de 'parallel for' suma=4
 static=1 ;  dynamic= 2 ; guided = 3; autor = 4
 dyn-var: 0
 nthreads-var: 12
 thread-limit-var: 2147483647
 run-sched-var: 2          chunk:1
omp_get_threads: 1, get_num_procs: 12, in_parallel: 0
 static=1 ;  dynamic= 2 ; guided = 3; autor = 4
```

**RESPUESTA**: Obtengo  los mismos resultados.

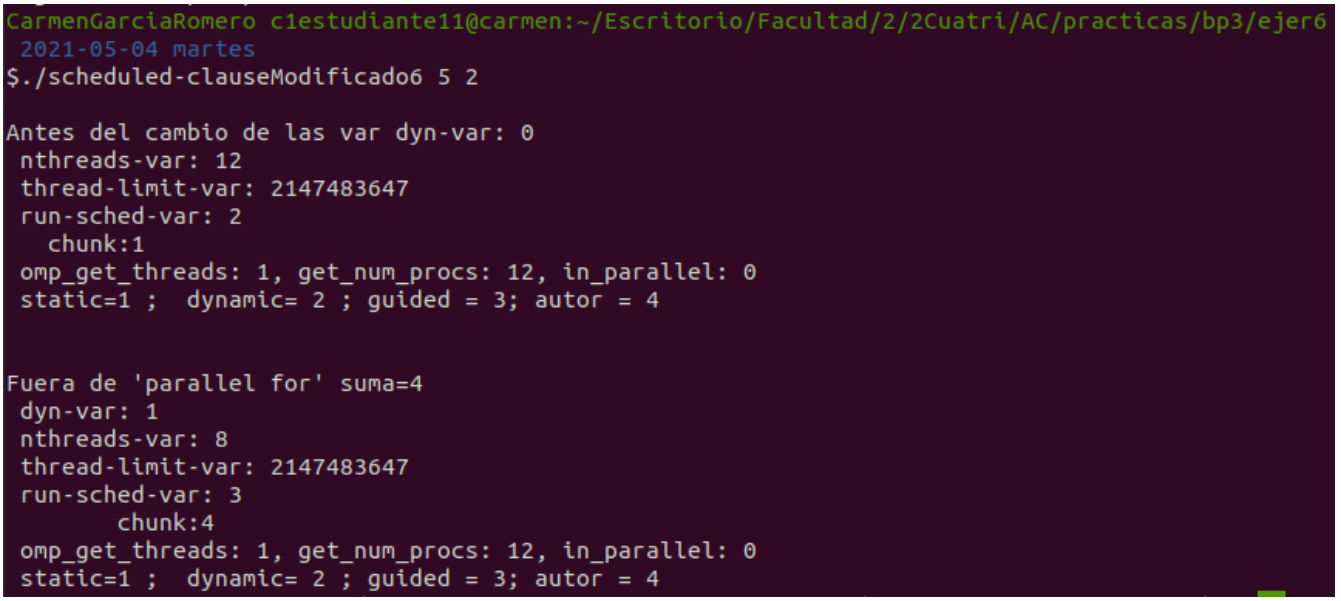
**6.** Añadir al programa scheduled-clause.c lo necesario para, usando funciones, modificar las variables de control dyn-var, nthreads-var y run-sched-var dentro de la región paralela y fuera de la región paralela. En la modificación de run-sched-var se debe usar un valor de kind distinto al utilizado en la cláusula schedule(). Añadir lo necesario para imprimir el contenido de estas variables antes y después de cada una de las dos modificaciones. Comentar los resultados.

**CAPTURA CÓDIGO FUENTE**: scheduled-clauseModificado6.c

scheduled-clauseModificado6.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;

  printf("\nAntes del cambio de las var");
  printf("\nstatic=1 ;  dynamic= 2 ; guided = 3; autor = 4\n");

  omp_get_schedule(&schedule_type, &chunk_value);
  printf(" dyn-var: %d\n nthreads-var: %d\n thread-limit-var: %d\n run-sched-var: %d\n \
  chunk:%d\n", \
       omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(),
       schedule_type, chunk_value);

  printf(" omp_get_threads: %d, get_num_procs: %d, in_parallel: %d\n", \
         omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());

         //revisar estas var con lo que dice el enuciado del kind
         //por lo demas puede estar done
         // Cambiamos los valores
         omp_set_dynamic(4);
         omp_set_num_threads(8);
         omp_set_schedule(2, 4);

  #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]=%d suma=%d \n", omp_get_thread_num(),i,a[i],suma);
  }

  printf("Fuera de 'parallel for' suma=%d\n", suma);
  printf(" static=1 ;  dynamic= 2 ; guided = 3; autor = 4\n");
  omp_get_schedule(&schedule_type, &chunk_value);
  printf(" dyn-var: %d\n nthreads-var: %d\n thread-limit-var: %d\n run-sched-var: %d\n \
       chunk:%d\n", \
       omp_get_dynamic(), omp_get_max_threads(), omp_get_thread_limit(),
       schedule_type, chunk_value);

   printf("\nomp_get_threads: %d, get_num_procs: %d, in_parallel: %d\n", \
        omp_get_num_threads(), omp_get_num_procs(), omp_in_parallel());
}
```

**CAPTURAS DE PANTALLA:**




**RESPUESTA**:
Tras el cambio, los valores son los proporcionados por las funciones en tiempo de ejecución.

## Resto de ejercicios (usar en atcgrid la cola ac a no ser que se tenga que usar atcgrid4)

**7.** Implementar un programa secuencial en C que multiplique una matriz triangular inferior por un vector (use variables dinámicas y tipo de datos double). Comparar el orden de complejidad y el número total de operaciones (sumas y productos) de este código respecto al 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

pmtv-secuencial.c

```
19      int *vector, *result, **matriz;
20      vector = (double *) malloc(N*sizeof(double));
21      result = (double *) malloc(N*sizeof(double));
22      matriz = (double **) malloc(N*sizeof(double*));
23
24      for (i=0; i<N; i++)
25          matriz[i] = (double*) malloc(N*sizeof(double));
26
27      for (i=0; i<N; i++){
28       for (j=i; j<N; j++){
29         if(j<=i)
30            matriz[i][j] = 0;
31         else
32            matriz[i][j] = 2;
33       }
34        vector[i] = 4;
35        result[i]=0;
36       }
37
38      // Pintamos la matriz
39      printf("Matriz:\n");
40      for (i=0; i<N; i++){
41       for (j=0; j<N; j++)
42         printf("%d ", matriz[i][j]);
43        printf("\n");
44      }
45
46      // Pintamos el vector
47      printf("Vector:\n");
48      for (i=0; i<N; i++)
49         printf("%d ", vector[i]);
50       printf("\n");
51
52       // Obtenemos los resaultados
53       for (i=0; i<N; i++)
54          for (j=i; j<N; j++)
55             result[i] += matriz[i][j] * vector[j];
56
57       // Pintamos los resultados
58       printf("Resultado:\n");
59       for (i=0; i<N; i++)
60          printf("%d ", result[i]);
61       printf("\n");
```

**CAPTURAS DE PANTALLA:**

```
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp3/ejer7  2021-05-15 sábado
$sbatch -p ac --wrap "./pmtv-secuencial 5"
Submitted batch job 105698
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp3/ejer7  2021-05-15 sábado
$cat slurm-105698.out
Matriz:
0 2 2 2 2
0 0 2 2 2
0 0 0 2 2
0 0 0 0 2
0 0 0 0 0
Vector:
4 4 4 4 4
Resultado:
32 24 16 8 0
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp3/ejer7  2021-05-15 sábado
$cat ssbatch -p ac --wrap "./pmtv-secuencial 7"
Submitted batch job 105699
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp3/ejer7  2021-05-15 sábado
$sbatccat slurm-105699.out
Matriz:
0 2 2 2 2 2 2
0 0 2 2 2 2 2
0 0 0 2 2 2 2
0 0 0 0 2 2 2
0 0 0 0 0 2 2
0 0 0 0 0 0 2
0 0 0 0 0 0 0
Vector:
4 4 4 4 4 4 4
Resultado:
48 40 32 24 16 8 0
```

**8.** Implementar en paralelo la multiplicación de una matriz triangular inferior 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. La inicialización de los datos la debe hacer el thread 0. 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. Mostrar en una captura de pantalla que el código resultante funciona correctamente. NOTA: usar para generar los valores aleatorios, por ejemplo, drand48_r().

**CAPTURA CÓDIGO FUENTE**: pmtv-OpenMP.c

pmtv-OpenMP.c

```
22      matriz = (double **) malloc(N*sizeof(double*));
23
24      for (i=0; i<N; i++)
25          matriz[i] = (double*) malloc(N*sizeof(double));
26
27      //Inicializacion de los datos por el thread 0
28      #pragma omp master
29      for (i=0; i<N; i++){
30       for (j=i; j<N; j++){
31         if(j<=i)
32            matriz[i][j] = 0;
33         else
34            matriz[i][j] = 2;
35       }
36        vector[i] = 4;
37        result[i]=0;
38       }
39
40      // Pintamos la matriz
41      printf("Matriz:\n");
42      for (i=0; i<N; i++){
43       for (j=0; j<N; j++)
44         printf("%d ", matriz[i][j]);
45        printf("\n");
46      }
47
48      // Pintamos el vector
49      printf("Vector:\n");
50      for (i=0; i<N; i++)
51         printf("%d ", vector[i]);
52       printf("\n");
53
54       // Obtenemos los resaultados
55       //Usamos runtime para que el usuario pueda fijar la planificacion de tareas
56       #pragma omp parallel for private(j) schedule(runtime)
57        for (i=0; i<N; i++)
58          for (j=i; j<N; j++)
59             result[i] += matriz[i][j] * vector[j];
60
61       // Pintamos los resultados
62       printf("Resultado:\n");
63       for (i=0; i<N; i++)
64          printf("%d ", result[i]);
65       printf("\n");
```

**DESCOMPOSICIÓN DE DOMINIO:**

Suponemos, por ejemplo, que tenemos una matriz, NxN y la multiplicamos por un vector(Nx1) hacemos una asignación estática fijando el chunk = x. Cada threads multiplicará por el vector x filas de la matriz, de forma consecutiva.

**CAPTURAS DE PANTALLA:**

```
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp3/ejer8  2021-05-17 lunes
$sbatch -p ac --wrap "./pmtv-OpenMP 7"
Submitted batch job 106345
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp3/ejer8  2021-05-17 lunes
$cat slurm-106345.out
Matriz:
0 2 2 2 2 2 2
0 0 2 2 2 2 2
0 0 0 2 2 2 2
0 0 0 0 2 2 2
0 0 0 0 0 2 2
0 0 0 0 0 0 2
0 0 0 0 0 0 0
Vector:
4 4 4 4 4 4 4
Resultado:
48 40 32 24 16 8 0
```

```
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp3/ejer8  2021-05-17 lunes
$sbatch -p ac --wrap "./pmtv-OpenMP 12"
Submitted batch job 106347
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp3/ejer8  2021-05-17 lunes
$cat slurm-106347
cat: slurm-106347: No existe el fichero o el directorio
CarmenGarciaRomero c1estudiante11@atcgrid:~/bp3/ejer8  2021-05-17 lunes
$cat slurm-106347.out
Matriz:
0 2 2 2 2 2 2 2 2 2 2 2
0 0 2 2 2 2 2 2 2 2 2 2
0 0 0 2 2 2 2 2 2 2 2 2
0 0 0 0 2 2 2 2 2 2 2 2
0 0 0 0 0 2 2 2 2 2 2 2
0 0 0 0 0 0 2 2 2 2 2 2
0 0 0 0 0 0 0 2 2 2 2 2
0 0 0 0 0 0 0 0 2 2 2 2
0 0 0 0 0 0 0 0 0 2 2 2
0 0 0 0 0 0 0 0 0 0 2 2
0 0 0 0 0 0 0 0 0 0 0 2
0 0 0 0 0 0 0 0 0 0 0 0
Vector:
4 4 4 4 4 4 4 4 4 4 4 4
Resultado:
88 80 72 64 56 48 40 32 24 16 8 0
```

**Nota:** al usar la función drand48_r() me daba errores, por tanto tuve que dejar los valores con los que había inicializamos las variables.

**9.** Contestar a las siguientes preguntas sobre el código del ejercicio anterior:

**(a)** ¿Qué número de operaciones de multiplicación y qué número de operaciones de suma realizan cada uno de los threads en la asignación static con monotonic y un chunk de 1?

**RESPUESTA**: Una operación cada hebra.

**(b)** 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**: Variará en función de la disponibilidad de las hebras, cada hebras ejecutará un número de multiplicaciones y sumas distintas, dependiendo de cuales estén ocupadas y cuales no.

**(c)** ¿Qué alternativa ofrece mejores prestaciones? Razonar la respuesta.

**RESPUESTA**: Es mejor la alternativa dynamic, ya que las iteraciones se dividen en unidades de chunk iteraciones y estas unidades se asignan en tiempo de ejecución según la disponibilidad de cada hebra, en cada momento. De esta forma los threads más rápidos que quedan libres pueden seguir ejecutando otras instrucciones, acelerando así el proceso y obteniendo mejores tiempos.

**10.** 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 (con monotonic en todos los casos). Usar un tamaño de vector N múltiplo del número de cores y de 64 que esté entre 11520 y 23040. El número de threads en las ejecuciones debe coincidir con el número de núcleos del computador. 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 (representar los valores de las dos tablas). Incluir los scripts utilizado en el cuaderno de prácticas. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

**CAPTURA CÓDIGO FUENTE**: pmtv-OpenMP.c

**DESCOMPOSICIÓN DE DOMINIO:**

**CAPTURAS DE PANTALLA:**

**TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid**

**SCRIPT:** pmvt-OpenMP_atcgrid.sh

**Tabla 2 .** Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector para vectores de tamaño N= (solo se ha paralelizado el producto, no la inicialización de los datos).

| Chunk | Static | Dynamic | Guided |
|---|---|---|---|
| por defecto | | | |
| 1 | | | |
| 64 | | | |

| Chunk | Static | Dynamic | Guided |
|---|---|---|---|
| por defecto | | | |
| 1 | | | |
| 64 | | | |