kkj2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

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

Estudiante (nombre y apellidos): Mario Garcia Marquez
Grupo de prácticas: Maribel Garcia
Fecha de entrega:
Fecha evaluación en clase:

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`

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

int main(int argc, char **argv) {
  int i, n = 20, x = 0, tid;
  int a[n], suma = 0, sumalocal;
  if (argc < 3) {
    fprintf(stderr, "[ERROR]-Falta iteraciones y threads\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)                                                \
    num_threads(x) 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:

```
ac/bp3/ejer1 on  main [!?]
❯ gcc -fopenmp -O2 if-clauseModificado.c
ac/bp3/ejer1 on  main [!?]
❯ ./a.out 50 2
 thread 1 suma de a[10]=10 sumalocal=10
 thread 1 suma de a[11]=11 sumalocal=21
 thread 1 suma de a[12]=12 sumalocal=33
 thread 1 suma de a[13]=13 sumalocal=46
 thread 1 suma de a[14]=14 sumalocal=60
 thread 1 suma de a[15]=15 sumalocal=75
 thread 1 suma de a[16]=16 sumalocal=91
 thread 1 suma de a[17]=17 sumalocal=108
 thread 1 suma de a[18]=18 sumalocal=126
 thread 1 suma de a[19]=19 sumalocal=145
 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 master=0 imprime suma=190
```

RESPUESTA:

Si n es mayor que 4 entonces se estable num_threads al numero especificado de hilos, sin embargo en caso de que no se cumple el if y este sea menor entonces la region paralela no se ejecutara en paralelo y todo se hara por la hebra master.

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=1	x=2	x=1	x=2	x=1	x=2
0	*id del thread*	0	0	0	8	9	10	5
1	1	0	1	0	15	9	3	5
2	2	1	2	1	9	6	2	7
3	3	1	3	1	1	6	0	7

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

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 static la asignacion de hebras se hace en tiempo de compilacion resultando en un orden preestablecido de hebras mientras que tanto dynamic y guided lo hacen de forma dinamica, es decir, establecen las hebras en tiempo de ejecucion resultando distintas ejecuciones en un orden distinto de hebras.

Ademas guided puede cambiar el tamano del chunk de forma dinamica para mejorar la distribucion de carga siendo el parametro chunk el tamano minimo. Aun asi esto no se ha visto en un programa tan liviano como que este.

Monotonic ejecuta las hebras en orden de iteracion mientras que nonmonotonic no. No se bien por que pero esto no se ha visto reflejado en las iteraciones.

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.

Para contestar a esta pregunta se ha buscado en la documentacion oficial de OpenMP. Por defecto en static se usa monotonic y en cualquier otro tipo nonmonotonic.

En caso de static chunk no tiene un valor especifico por defecto mientras que para los otros 2 tipos su valor por defecto es 1.

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`

File: **schedule-clauseModificado.c**

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

int main(int argc, char **argv) {

  int i, n = 16, chunk, a[n], suma = 0;

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

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

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

    if (i == 10)
      printf("dyn-var = %d, nthreads-var = %d thread-limit-var = %d "
             "run-sched-var = %d\n",
             omp_get_dynamic(), omp_get_num_threads(), omp_get_thread_limit(),
             omp_get_dynamic());
  }

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

  printf("dyn-var = %d, nthreads-var = %d thread-limit-var = %d "
         "run-sched-var = %d fuera\n",
         omp_get_dynamic(), omp_get_num_threads(), omp_get_thread_limit(),
         omp_get_dynamic());

  return (0);
}
```

CAPTURAS DE PANTALLA:

```
❯ ./a.out 2
 thread 0 suma a[0] suma=0
 thread 0 suma a[1] suma=1
 thread 0 suma a[2] suma=3
 thread 0 suma a[3] suma=6
 thread 0 suma a[4] suma=10
 thread 0 suma a[5] suma=15
 thread 0 suma a[6] suma=21
 thread 0 suma a[7] suma=28
 thread 0 suma a[8] suma=36
 thread 0 suma a[9] suma=45
 thread 0 suma a[10] suma=55
dyn-var = 0, nthreads-var = 1 thread-limit-var = 2147483647 run-sched-var = 0
 thread 0 suma a[11] suma=66
 thread 0 suma a[12] suma=78
 thread 0 suma a[13] suma=91
 thread 0 suma a[14] suma=105
 thread 0 suma a[15] suma=120
Fuera de 'parallel for' suma=120
dyn-var = 0, nthreads-var = 1 thread-limit-var = 2147483647 run-sched-var = 0 fuera
```

RESPUESTA:

La captura adjtunta pertenece a la ejecucion en caso de static. En la salida del mensaje exterior siempre permanece invariante mientras que en la interior varia tanto el numero de n-threads como los indicadores de schedule que son dyn-var y run-sched-var indicando si la distribucion de threads es dinamica o estatica.

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-clauseModificado4.c`

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

int main(int argc, char **argv) {

  int i, n = 16, chunk, a[n], suma = 0;

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

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

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

    if (i == 10)
      printf("num_procs = %d, nthreads-var = %d in_parallel = %d \n",
             omp_get_num_procs(), omp_get_num_threads(), omp_in_parallel());
  }

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

  printf("num_procs = %d, nthreads-var = %d in_parallel = %d \n",
         omp_get_num_procs(), omp_get_num_threads(), omp_in_parallel());

  return (0);
}
```

CAPTURAS DE PANTALLA:

```
❯ ./a.out 1
 thread 7 suma a[7] suma=7
 thread 4 suma a[4] suma=4
 thread 9 suma a[9] suma=9
 thread 5 suma a[5] suma=5
 thread 10 suma a[10] suma=10
 thread 15 suma a[15] suma=15
 thread 2 suma a[2] suma=2
 thread 14 suma a[14] suma=14
 thread 6 suma a[6] suma=6
 thread 0 suma a[0] suma=0
 thread 8 suma a[8] suma=8
 thread 13 suma a[13] suma=13
 thread 3 suma a[3] suma=3
 thread 12 suma a[12] suma=12
 thread 1 suma a[1] suma=1
 thread 11 suma a[11] suma=11
num_procs = 16, nthreads-var = 16 in_parallel = 1
Fuera de 'parallel for' suma=15
num_procs = 16, nthreads-var = 1 in_parallel = 0
```

RESPUESTA:

Se obtienen distintos resultados en las variables nthreads-var que indica los hilos ejecutandose en ese momento, que muestra 16(los maximos de mi PC) dentro de la region paralela mientras que fuera de ella es 1, ya que se ejecuta el programa de forma secuencial. La variable in_parallel indica si estamos en una region paralela o no.

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-clauseModificado5.c`

```
46    chunk = atoi(argv[1]);
45
44    for (i = 0; i < n; i++)
43      a[i] = i;
42  #pragma omp parallel
41    {
40
39  #pragma omp for schedule(runtime) firstprivate(suma) lastprivate(suma)
38      for (i = 0; i < n; i++) {
37        suma = suma + a[i];
36        printf(" thread %d suma a[%d] suma=%d \n", omp_get_thread_num(), i, suma);
35      }
34  #pragma omp single
33      {
32        int chunk;
31        omp_sched_t sched;
30        omp_get_schedule(&sched, &chunk);
29        printf("schedule = %d, nthreads-var = %d in_parallel = %d \n", sched,
28               omp_get_num_threads(), omp_in_parallel());
27      }
26
25      omp_set_num_threads(5);
24      omp_set_schedule(omp_sched_static, 1);
23      omp_set_dynamic(4);
22
21  #pragma omp single
20      {
19        int chunk;
18        omp_sched_t sched;
17        omp_get_schedule(&sched, &chunk);
16        printf("schedule = %d, nthreads-var = %d in_parallel = %d \n", sched,
15               omp_get_num_threads(), omp_in_parallel());
14      }
13    }
12
11    omp_set_num_threads(4);
10    omp_set_schedule(omp_sched_dynamic, 2);
 9    omp_set_dynamic(3);
 8    printf("Fuera de 'parallel for' suma=%d\n", suma);
 7
 6    omp_sched_t sched;
 5    omp_get_schedule(&sched, &chunk);
 4    printf("schedule = %d, nthreads-var = %d in_parallel = %d \n", sched,
 3           omp_get_num_threads(), omp_in_parallel());
 2
 1    return (0);
59  }
```

CAPTURAS DE PANTALLA:

```
ac/bp3/ejer6 on  main [!?]
❯ gcc -fopenmp -O2 schedule-clauseModificado5.c
ac/bp3/ejer6 on  main [!?]
❯ ./a.out 2
 thread 8 suma a[11] suma=11
 thread 7 suma a[10] suma=10
 thread 11 suma a[5] suma=5
 thread 14 suma a[15] suma=15
 thread 0 suma a[12] suma=12
 thread 3 suma a[6] suma=6
 thread 6 suma a[4] suma=4
 thread 15 suma a[8] suma=8
 thread 9 suma a[14] suma=14
 thread 10 suma a[0] suma=0
 thread 4 suma a[3] suma=3
 thread 1 suma a[13] suma=13
 thread 5 suma a[2] suma=2
 thread 13 suma a[9] suma=9
 thread 2 suma a[1] suma=1
 thread 12 suma a[7] suma=7
schedule = 2, nthreads-var = 16 in_parallel = 1
schedule = 1, nthreads-var = 16 in_parallel = 1
Fuera de 'parallel for' suma=15
schedule = 2, nthreads-var = 1 in_parallel = 0
```

RESPUESTA:

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`

```c
#include <stdio.h>
#include <stdlib.h>
#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_thread_num() 0
#endif

int main(int nargs, char **args) {
  if (nargs < 2) {
    printf("Use program with %s <matrix dimension>\n", args[0]);
    return 1;
  }

  int dim = atoi(args[1]);
  int i, j;
  double suma;
  unsigned seed;

  // Creation of matrix and vector
  double *vector = malloc(sizeof(double) * dim);
  double *resultado = malloc(sizeof(double) * dim);
  double **matrix = malloc(sizeof(double *) * dim);
  for (i = 0; i < dim; i++) {
    matrix[i] = malloc(sizeof(double) * dim);
  }

  // La matriz es triangular inferior
  // Initialization
  for (i = 0; i < dim; i++) {
    seed = 25234 + 17 * omp_get_thread_num();
    vector[i] = rand_r(&seed);
    for (j = 0; j <= i; j++)
      matrix[i][j] = rand_r(&seed);
  }

  // Producto
  for (i = 0; i < dim; i++) {
    suma = 0;
    for (j = 0; j <= i; j++) {
      suma += vector[j] * matrix[i][j];
    }
    resultado[i] = suma;
  }

```

```
20   // Producto
19   for (i = 0; i < dim; i++) {
18     suma = 0;
17     for (j = 0; j <= i; j++) {
16       suma += vector[j] * matrix[i][j];
15     }
14     resultado[i] = suma;
13   }
12
11   // Salida resultados
10   printf("resultado[0] = %f, resultado[%d] = %f\n", resultado[0], dim - 1,
 9          resultado[dim - 1]);
 8
 7   // Free
 6   free(vector);
 5   free(resultado);
 4   for (i = 0; i < dim; i++) {
 3     free(matrix[i]);
 2   }
 1   free(matrix);
57 }
```

CAPTURAS DE PANTALLA:

```
[MarioGarciaMarquez e1estudiante9@atcgrid:~/bp3/ejer7] 2021-05-12 miércoles
$gcc -O2 -fopenmp pmtv-secuencial.c
[MarioGarciaMarquez e1estudiante9@atcgrid:~/bp3/ejer7] 2021-05-12 miércoles
$srun -pac -Aac ./a.out 50
resultado[0] = 1012317583240356608.000000, resultado[49] = 642190972230321684448.000000
[MarioGarciaMarquez e1estudiante9@atcgrid:~/bp3/ejer7] 2021-05-12 miércoles
$
```

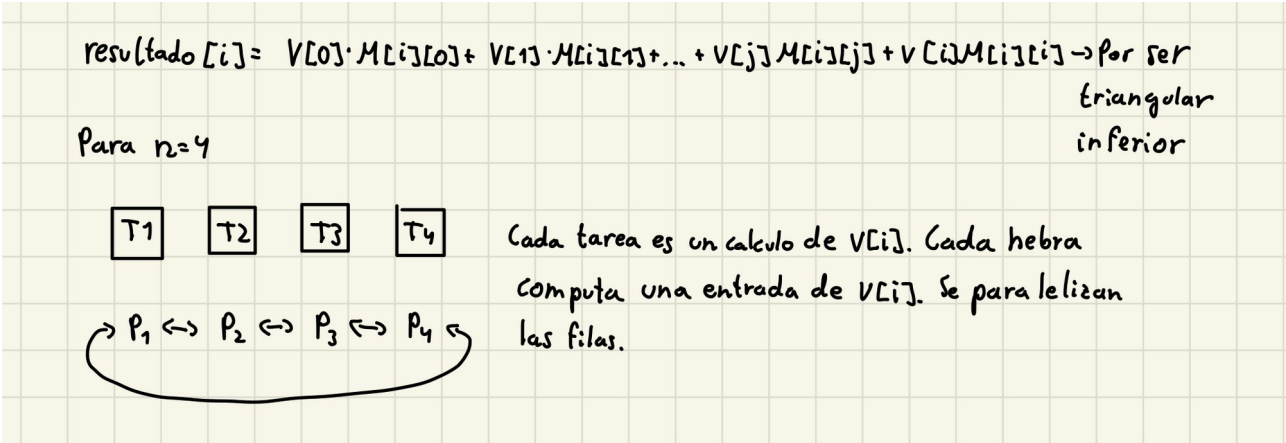
Se realizan nlog(n) operaciones mientras en el programa anterior se realizaban el orden de n^ 2 operaciones.

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`

```
35   // Creation of matrix and vector
34   double *vector = malloc(sizeof(double) * dim);
33   double *resultado = malloc(sizeof(double) * dim);
32   double **matrix = malloc(sizeof(double *) * dim);
31   for (i = 0; i < dim; i++) {
30     matrix[i] = malloc(sizeof(double) * dim);
29   }
28
27   // La matriz es triangular inferior
26   // Initialization
25   for (i = 0; i < dim; i++) {
24     seed = 25234 + 17 * omp_get_thread_num();
23     vector[i] = rand_r(&seed);
22     for (j = 0; j <= i; j++)
21       matrix[i][j] = rand_r(&seed);
20   }
19
18   // Producto
17 #pragma omp parallel for schedule(runtime) private(j, suma)
16   for (i = 0; i < dim; i++) {
15     suma = 0;
14     for (j = 0; j <= i; j++) {
13       suma += vector[j] * matrix[i][j];
12     }
11     resultado[i] = suma;
10   }
 9
 8   // Salida resultados
 7   printf("resultado[0] = %f, resultado[%d] = %f\n", resultado[0], dim - 1,
 6          resultado[dim - 1]);
 5
 4   // Free
 3   free(vector);
 2   free(resultado);
 1   for (i = 0; i < dim; i++) {
5 []   free(matrix[i]);
```

DESCOMPOSICIÓN DE DOMINIO:



CAPTURAS DE PANTALLA:

```
[MarioGarciaMarquez e1estudiante9@atcgrid:~/bp3/ejer8] 2021-05-12 miércoles
$gcc -fopenmp -O2 pmtv-OpenMP.c
[MarioGarciaMarquez e1estudiante9@atcgrid:~/bp3/ejer8] 2021-05-12 miércoles
$srun -pac -Aac ./a.out 50
resultado[0] = 1012317583240356608.000000, resultado[49] = 64219097223032168448.000000
[MarioGarciaMarquez e1estudiante9@atcgrid:~/bp3/ejer8] 2021-05-12 miércoles
$
```

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: Siendo n el numero total de operaciones a realizar, un total de n/numero de hilos

(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: Se mantienen en un numero parecido de ejecuciones aunque agrega un overhead al programa con la cantidad de computo de redistribuir la carga

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

RESPUESTA: Static ofrece una mejor alternativa ya que ofrece un balance correcto entre los procesadores y no agrega el overhead que si lo hace dynamic y guided

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`

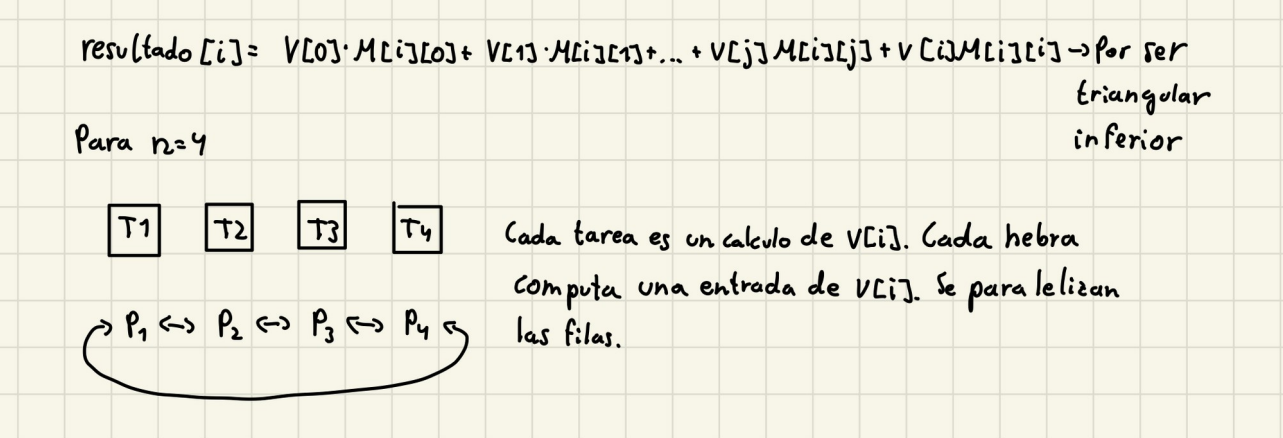
```
  }

  // La matriz es triangular inferior
  // Initialization
  for (i = 0; i < dim; i++) {
    seed = 25234 + 17 * omp_get_thread_num();
    vector[i] = rand_r(&seed);
    for (j = 0; j <= i; j++)
      matrix[i][j] = rand_r(&seed);
  }
  t1 = omp_get_wtime();
  // Producto
#pragma omp parallel for schedule(runtime) private(j, suma)
  for (i = 0; i < dim; i++) {
    suma = 0;
    for (j = 0; j <= i; j++) {
      suma += vector[j] * matrix[i][j];
    }
    resultado[i] = suma;
  }
  t2 = omp_get_wtime();
  // Salida resultados
  printf("resultado[0] = %f, resultado[%d] = %f\n", resultado[0], dim - 1,
         resultado[dim - 1]);
  printf("time = %f\n", t2 - t1);

  // Free
  free(vector);
  free(resultado);
  for (i = 0; i < dim; i++) {
    free(matrix[i]);
  }
  free(matrix);
}
```

El resto del codigo es igual que en el ejercicio 8

DESCOMPOSICIÓN DE DOMINIO:



Igual que en el ejercicio 8

CAPTURAS DE PANTALLA:

```
[MarioGarciaMarquez e1estudiante9@atcgrid:~/bp3/ejer10] 2021-05-12 miércoles
$cat slurm-104671.out
STATIC default
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.139407
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.133483
STATIC 1
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.128519
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.128348
STATIC 64
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.128231
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.127958
monotonic:dynamic default
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.128361
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.128328
monotonic:dynamic 1
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.128561
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.128657
monotonic:dynamic 64
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.128198
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.127973
monotonic:guided default
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.127913
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.128021
monotonic:guided 1
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.127590
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.128476
monotonic:guided 64
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.127807
resultado[0] = 1012317583240356608.000000, resultado[13823] = 20480936193042560319488.000000
time = 0.128405
```

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-OpenMP_atcgrid.sh

```
#!/bin/bash

echo "STATIC default"
export OMP_SCHEDULE="static"
./a.out 13824
./a.out 13824

echo "STATIC 1"
export OMP_SCHEDULE="static,1"
./a.out 13824
./a.out 13824

echo "STATIC 64"
export OMP_SCHEDULE="static,64"
./a.out 13824
./a.out 13824


echo "monotonic:dynamic default"
export OMP_SCHEDULE="monotonic:dynamic"
./a.out 13824
./a.out 13824

echo "monotonic:dynamic 1"
export OMP_SCHEDULE="monotonic:dynamic,1"
./a.out 13824
./a.out 13824

echo "monotonic:dynamic 64"
export OMP_SCHEDULE="monotonic:dynamic,64"
./a.out 13824
./a.out 13824


echo "monotonic:guided default"
export OMP_SCHEDULE="monotonic:guided"
./a.out 13824
./a.out 13824

echo "monotonic:guided 1"
export OMP_SCHEDULE="monotonic:guided,1"
./a.out 13824
./a.out 13824

echo "monotonic:guided 64"
export OMP_SCHEDULE="monotonic:guided,64"
./a.out 13824
```

Tabla 2 . Tiempos de ejecución de la versión paralela del producto de una matriz triangular por un vector para vectores de tamaño N= 13824 (solo se ha paralelizado el producto, no la inicialización de los datos).

Chunk	Static	Dynamic	Guided
por defecto	0.1394	0.128361	0.127913
1	0.1285	0.128561	0.127590
64	0.1282	0.128198	0.127807

Chunk	Static	Dynamic	Guided
por defecto	0.133483	0.128328	0.128021
1	0.1283	0.128657	0.128476
64	0.1279	0.127973	0.128405