

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

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

Estudiante (nombre y apellidos): Juan Carlos Ruiz Fernández

Grupo de prácticas: B3

Fecha de entrega: 25 de mayo del 2021

Fecha evaluación en clase:

2º curso / 2º cuatr.
Grado Ing. Inform.

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`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <omp.h>
4  int main(int argc, char **argv)
5  {
6      int i, n = 20, threads, tid;
7      int a[n], suma = 0, sumalocal;
8
9      if (argc < 3){
10         fprintf(stderr, "[ERROR]-Falta iteraciones , Numero Hebras\n");
11         exit(-1);
12     }
13
14     n = atoi(argv[1]);
15     threads = atoi(argv[2]);
16
17     if (n > 20)
18         n = 20;
19
20     for (i = 0; i < n; i++){
21         a[i] = i;
22     }
23
24     #pragma omp parallel if (n > 4) num_threads(threads) default(none) private(sumalocal, tid) shared(a, suma, n)
25     {
26         sumalocal = 0;
27         tid = omp_get_thread_num();
28
29         #pragma omp for private(i) schedule(static) nowait
30         for (i = 0; i < n; i++){
31             sumalocal += a[i];
32             printf(" thread %d suma de a[%d]=%d sumalocal=%d \n", tid, i, a[i], sumalocal);
33         }
34
35         #pragma omp atomic
36         suma += sumalocal;
37
38         #pragma omp barrier
39         #pragma omp master
40         printf("thread master=%d imprime suma=%d\n", tid, suma);
41     }
42
43 }
```

CAPTURAS DE PANTALLA:

```
juanca@DESKTOP-RCIHQK2: /mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP3/ejer1$ ./if-clauseModificado 6 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 master=0 imprime suma=15
juanca@DESKTOP-RCIHQK2: /mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP3/ejer1$ ./if-clauseModificado 6 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 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 master=0 imprime suma=15
juanca@DESKTOP-RCIHQK2: /mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP3/ejer1$ ./if-clauseModificado 6 3
thread 2 suma de a[4]=4 sumalocal=4
thread 2 suma de a[5]=5 sumalocal=9
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 master=0 imprime suma=15
juanca@DESKTOP-RCIHQK2: /mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP3/ejer1$
```

RESPUESTA:

Al ser un argumento del programa, facilita ver las diferencias entre las ejecuciones para 6 iteraciones con 1, 2 o 3 hebras. La salida muestra como se reparte el trabajo las hebras y que parte de éste: Para una hebra, obviamente, se lleva todo el trabajo; para dos hebras se reparte equitativamente entre ellas y de igual forma con tres hebras. Si se diera un numero de hebra no múltiplo de las iteraciones se repartiría por igual entre las `n-1` hebras dejando a solo una hebra con distinta carga restante.

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="non-monotonic: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	0	0	0	0	2	2	2	2
1	1	0	1	0	1	2	2	2
2	2	1	2	1	0	1	2	2
3	0	1	0	1	0	1	2	2
4	1	2	1	2	1	0	2	2
5	2	2	2	2	1	0	2	2
6	0	0	0	0	1	2	1	1
7	1	0	1	0	1	2	1	1
8	2	1	2	1	1	0	1	1
9	0	1	0	1	1	0	1	1
10	1	2	1	2	1	0	0	0
11	2	2	2	2	1	0	0	0
12	0	0	0	0	1	0	1	2
13	1	0	1	0	1	0	1	2
14	2	1	2	1	1	0	2	0
15	0	1	0	1	1	0	2	0

RESPUESTA:

En esta tabla no se representa la diferencia entre *monotonic* y *nonmonotonic* porque está ordenado en funcion de la iteracion. Entre ellos dos se diferencian en que en que el primero lo realiza en orden creciente y el segundo lo pueden hacer de forma creciente o decreciente durante la ejecucion.

Entre *static*, *dynamic* y *guided* la diferencia reside en el como y cuando asigna *trabajo a las hebras*. *Static*, como su nombre indica, ya esta establecido y siempre será igual la distribucion del trabajo ejecucion tras ejecucion. *Dynamic* en cambio lo hace en funcion de velocidad, si una hebra es mas rapida le asigna mas iteraciones. *Guided* cambia el chunk, al principio la granularidad es mas grande y asigna mas tareas a cada hebra y la va reduciendo con forme se acaban, poniendo el limite en el chunk especificado (se aprecia en la ultima columna que hasta la iteracion 9 las hebras 2 y 1 ejecutan muchas mas iteraciones que las especificadas, hasta el resto que se reparte por el chunk = 2).

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.

Al principio pensé que se seteaba por defecto a 1. Al comprobarlo me di cuenta que para `dynamic` y `guided` lo cumplian pero no `static`. Buscando en el manual de *OpenMP3.1 Complete Specifications* de la pagina oficial (<https://www.openmp.org/wp-content/uploads/OpenMP3.1.pdf> pagina 52 del pdf, 44 del documento) encontré que efectivamente que el valor por defecto en `dynamic` y `guided` es 1 pero para `static` se reparte por igual entre el numero de iteraciones, mas o menos, ya que seguro que alguna de las hebras se llevará una iteración mas dependiendo de el numero de éstas y de hebras.

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`

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #ifdef _OPENMP
5 #include <omp.h>
6 #else
7 #define omp_get_thread_num() 0
8 #endif
9
10 int main(int argc, char **argv){
11     int i, n = 200, chunk, a[n], suma = 0, dynamic, nthreads, limit, chunk2;
12     omp_sched_t tipo;
13
14     if (argc < 3){
15         fprintf(stderr, "Falta iteraciones o chunk\n");
16         exit(-1);
17     }
18
19     n = atoi(argv[1]);
20     if (n > 200) n = 200;
21     chunk = atoi(argv[2]);
22
23     for (i = 0; i < n; i++) a[i] = i;
24
25     #pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic, chunk)
26     for (i = 0; i < n; i++)
27     {
28         if(i==0){
29             printf("\n DENTRO DE LA REGION PARALEL\n \tdyn-var: %d\n", omp_get_dynamic());
30             printf("\tnthreads-var: %d\n", omp_get_max_threads());
31             printf("\tthreads-limit-var: %d\n", omp_get_max_threads());
32             omp_get_schedule(&tipo,&chunk2);
33             printf("\trun-chd-var: %s\n", tipo);
34         }
35         suma = suma + a[i];
36         printf(" thread %d suma a[%d]=%d suma=%d \n",
37             omp_get_thread_num(), i, a[i], suma);
38     }
39
40     printf("\nfuera de 'parallel for' suma=%d\n", suma);
41     printf("\tdyn-var: %d\n", omp_get_dynamic());
42     printf("\tnthreads-var: %d\n", omp_get_max_threads());
43     printf("\tthreads-limit-var: %d\n", omp_get_max_threads());
44     omp_get_schedule(&tipo,&chunk2);
45     printf("\trun-chd-var: %s\n", tipo);
46 }

```

CAPTURAS DE PANTALLA:

```

[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2*IngInfo 2*Cuatri/A
C/Prácticas/BP3/ejer4] 2021-05-05 Wednesday
$gcc -O2 -fopenmp scheduled-clauseModificado.c -o scheduled-clauseModificado $5 ./scheduled-clau
seModificado $ 2

DENTRO DE LA REGION PARALEL
dyn-var: 1
nthreads-var: 8
threads-limit-var: 8
run-chd-var: 80000002

thread 7 suma a[0]=0 suma=0
thread 7 suma a[1]=1 suma=1
thread 3 suma a[2]=2 suma=2
thread 3 suma a[3]=3 suma=5
thread 0 suma a[4]=4 suma=4

Fuera de 'parallel for' suma=4
dyn-var: 1
nthreads-var: 8
threads-limit-var: 2147483647
run-chd-var: 80000002

[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2*IngInfo 2*Cuatri/A
C/Prácticas/BP3/ejer4] 2021-05-05 Wednesday
$export OMP_DYNAMIC=TRUE $5 export OMP_NUM_THREADS=4
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2*IngInfo 2*Cuatri/A
C/Prácticas/BP3/ejer4] 2021-05-05 Wednesday
$gcc -O2 -fopenmp scheduled-clauseModificado.c -o scheduled-clauseModificado $5 ./scheduled-clau
seModificado $ 2

DENTRO DE LA REGION PARALEL
dyn-var: 1
nthreads-var: 4
threads-limit-var: 4
run-chd-var: 80000002

thread 0 suma a[0]=0 suma=0
thread 0 suma a[1]=1 suma=1
thread 3 suma a[2]=2 suma=4
thread 1 suma a[3]=3 suma=5

Fuera de 'parallel for' suma=4
dyn-var: 1
nthreads-var: 4
threads-limit-var: 2147483647
run-chd-var: 80000002

[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2*IngInfo 2*Cuatri/A

```

RESPUESTA:

La única diferencia entre fuera y dentro de la región paralela es *thread-limit-var*, la que fuera de la región queda indefinida aparentemente (ha cogido el máximo valor de int en 32bits). Haciendo export de dos variables entre ejecuciones si las hace cambiar, pero pasa lo mismo dentro y fuera de la región.

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`

```

25 #pragma omp parallel for firstprivate(suma) lastprivate(suma) schedule(dynamic, chunk)
26 for (i = 0; i < n; i++)
27 {
28     if(i==0){
29         printf("\n DENTRO DE LA REGION PARALEL\n \tomp_get_num_threads(): %d\n", omp_get_num_threads());
30         printf("\tomp_get_num_procs(): %d\n", omp_get_num_procs());
31         printf("\tomp_in_parallel(): %d\n", omp_in_parallel());
32     }
33     suma = suma + a[i];
34     printf(" thread %d suma a[%d]=%d suma=%d \n",
35         omp_get_thread_num(), i, a[i], suma);
36 }
37
38 printf("\nfuera de 'parallel for' suma=%d\n", suma);
39 printf("\n FUERA DE LA REGION PARALEL\n \tomp_get_num_threads(): %d\n", omp_get_num_threads());
40 printf("\tomp_get_num_procs(): %d\n", omp_get_num_procs());
41 printf("\tomp_in_parallel(): %d\n", omp_in_parallel());
42 }

```

CAPTURAS DE PANTALLA:

```
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/A
C/Prácticas/BP3/ejer5] 2021-05-05 Wednesday
$gcc -O2 -fopenmp scheduled-clauseModificado4.c -o scheduled-clauseModificado4 && ./scheduled-cl
auseModificado4 5 2
thread 0 suma a[2]=2 suma=2
thread 0 suma a[3]=3 suma=5
thread 2 suma a[4]=4 suma=4

DENTRO DE LA REGION PARALLEL
omp_get_num_threads(): 4
omp_get_num_procs(): 8
omp_in_parallel(): 1

thread 3 suma a[0]=0 suma=0
thread 3 suma a[1]=1 suma=1

Fuera de 'parallel for' suma=4

FUERA DE LA REGION PARALLEL
omp_get_num_threads(): 1
omp_get_num_procs(): 8
omp_in_parallel(): 0
[juan@DESKTOP-RCIHQK2:/mnt/c/Users/juanc/OneDrive/2ªIngInfo 2ºCuatri/A
C/Prácticas/BP3/ejer5] 2021-05-05 Wednesday
```

RESPUESTA:

Se obtienen diferencias para *num_threads* y para *in_parallel*. Dentro del *parallel* tiene el valor que yo exporte en la captura del ejercicio anterior (4) y es la cantidad de threads ejecutandose; pues fuera de la región se convierte en 1. Para *in_parallel* como no tenemos condicionales en la directiva *parallel* sabemos que siempre se va a ejecutar en modo paralelo, por eso dentro de la región muestra un 1 y fuera un 0, esta ultima demostrando que no hay región paralela activa.

6. Añadir al programa `scheduled-clone.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

[illegible]

CAPTURAS DE PANTALLA:

```

juanca@DESKTOP-RCIHQK2: /r  × + -
[JuanCarlosRuizFernandez juanca@DESKTOP-RCIHQK2:
/mnt/c/Users/juanc/OneDrive/2ºIngInfo 2ºCuatri/AC/Prácticas/BP3/ejer6] 2021-05-19 Wednesday
$gcc -fopenmp -O2 schedule-clauseModificado5.c -o schedule-clauseModificado5 && ./schedule-clauseModificado5 2

-----

Antes del cambio DENTRO PARALLEL:
    dyn-var: 0
    nthreads-var: 8
    run-sched-var: dinamico

-----

DESPUES del cambio DENTRO PARALLEL:
    dyn-var: 1
    nthreads-var: 4
    run-sched-var: estatico

-----

thread 1 suma a[0] suma=0
thread 1 suma a[1] suma=1
thread 6 suma a[6] suma=6
thread 5 suma a[2] suma=2
thread 5 suma a[3] suma=5
thread 0 suma a[4] suma=4
thread 0 suma a[5] suma=9
Fuera de 'parallel for' suma=6

-----

Antes del cambio FUERA PARALLEL:
    dyn-var: 0
    nthreads-var: 8
    run-sched-var: dinamico

-----

DESPUES del cambio FUERA PARALLEL:
    dyn-var: 1
    nthreads-var: 4
    run-sched-var: estatico

```

RESPUESTA:

Los cambios se realizan correctamente pero después de salir de la region *parallel* recuperan los valores anteriores al cambio.

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

```

1  #include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
2  #include <stdio.h> // biblioteca donde se encuentra la función printf()
3  #include <time.h> // biblioteca donde se encuentra la función clock_gettime()
4
5  //Define VECTOR_LOCAL
6  #define VECTOR_LOCAL
7  #define VECTOR_DYNAMIC
8  #ifdef VECTOR_LOCAL
9  #define MAX 5792 //sqrt(2^25)
10 double m1[MAX][MAX], v2[MAX], v3[MAX];
11 #endif
12 int main(int argc, char **argv){
13     int i;
14     struct timespec cgt1, cgt2;
15     double mgt; //para tiempo de ejecución
16
17     if (argc < 2){ //Leer argumento de entrada (n de componentes de la matriz)
18         printf("Faltan n componentes de la matriz\n");
19         exit(-1);
20     }
21     unsigned int N = atoi(argv[1]); // Máximo N=2^12-1=4294967295 (sizeof(unsigned int) - 4)
22     //printf("Tamaño Vectores: %u (%d B)\n", N, sizeof(unsigned int));
23     #ifdef VECTOR_LOCAL
24     double v1[N], v2[N], v3[N]; // Tamaño variable local en tiempo de ejecución ...
25     // disponible en C a partir de C99
26     #endif
27     #ifdef VECTOR_DYNAMIC
28     if (N > MAX)
29         N = MAX;
30     #endif
31     #ifdef VECTOR_DYNAMIC
32     double *m1, *v2, *v3;
33     m1 = (double **)malloc(N * sizeof(double)); // malloc necesita el tamaño en bytes
34     v2 = (double *)malloc(N * sizeof(double));
35     v3 = (double *)malloc(N * sizeof(double));
36     if ((m1 == NULL) || (v2 == NULL) || (v3 == NULL))
37     {
38         printf("No hay suficiente espacio para los vectores o matriz\n");
39         exit(-2);
40     }
41     for (int i = 0; i < N; i++)
42     {
43         m1[i] = (double *)malloc(N * sizeof(double));
44         if (m1[i] == NULL){
45             printf("No hay suficiente espacio para las columnas de la matriz\n");
46             exit(-2);
47         }
48     }
49     #endif
50     //Inicializar triangular INFERIOR
51     if (N < 9)
52         return 0;
53
54     //Inicializar triangular INFERIOR
55     if (N < 9)
56         return 0;
57
58     for (i = 0; i < N; i++){
59         for (int j = 0; j < N; j++){
60             if (j < (i+1)) //Solo la parte inferior
61                 m1[i][j] = N * 0.1 + i * 0.1;
62             else m1[i][j] = 0;
63         }
64         v2[i] = N * 0.1 - i * 0.1;
65     }
66     srand(time(0));
67     for (i = 0; i < N; i++)
68     {
69         for (int j = 0; j < N; j++){
70             m1[i][j] = rand() / ((double)rand());
71             v2[i] = rand() / ((double)rand()); //printf("%d.%f,%d.%f", i, v1[i], v2[i]);
72         }
73     }
74
75     //Calcular suma de vectores
76     clock_gettime(CLOCK_REALTIME, &cgt1);
77     for (i = 0; i < N; i++)
78     {
79         for (int j = 0; j < (i+1); j++){
80             v3[i] += m1[i][j] * v2[j];
81         }
82         clock_gettime(CLOCK_REALTIME, &cgt2);
83     }
84     mgt = (double)(cgt2.tv_sec - cgt1.tv_sec) +
85           (double)((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
86     //Imprimir resultado de la suma y el tiempo de ejecución
87     if (N < 10)
88     {
89         printf("Tiempo: %11.9f / Tamaño Vectores: %u\n", mgt, N);
90         for (i = 0; i < N; i++){
91             printf("v3: ");
92             for (int j = 0; j < N; j++){
93                 printf("%11.9f", m1[i][j]);
94             }
95             if (i == N-1)
96                 printf("\n");
97             else
98                 printf(" ");
99         }
100     }
101     else
102     {
103         printf("Tiempo: %11.9f / Tamaño matriz y Vectores: %u\n", m1[0][0] * v2[0], v3[0] * (N * 0.6 + 0.3 * N * 0.6) /
104               mgt, N, m1[0][0], v2[0], v3[0], N - 1, N - 1, N - 1, N - 1, m1[N - 1][N - 1], v2[N - 1], v3[N -
105     #ifdef VECTOR_DYNAMIC
106     for (i = 0; i < N; i++)
107         free(m1[i]);
108     free(m1); // libera el espacio reservado para m1
109     free(v2); // libera el espacio reservado para v2
110     free(v3); // libera el espacio reservado para v3
111     #endif
112     return 0;
113 }

```

CAPTURAS DE PANTALLA:

```

b3estudiante23@atcgird:~/BP3, x + v
[JuanCarlosRuizFernandez b3estudiante23@atcgird:~/BP3/ejer7] 2021-05-19 Wednesday
$gcc -O2 pmtv-secuencial.c -o pmtv-secuencial && ./pmtv-secuencial 4
Tiempo:0.000000108 / Tamaño Vectores:4
#0.40 0.00 0.00 0.00 # #0.40# = #0.160#
#0.50 0.50 0.00 0.00 # #0.30# = #0.300#
#0.60 0.60 0.60 0.00 # #0.20# = #0.360#
#0.70 0.70 0.70 0.70 #* #0.10# = #0.280#
[JuanCarlosRuizFernandez b3estudiante23@atcgird:~/BP3/ejer7] 2021-05-19 Wednesday

```

Para el ejemplo de la matriz completa seguía un orden $O(n^2)$, $N*N$ sumas y multiplicaciones. Claramente en este caso tenemos que número de cálculos es menor siguiendo un orden aproximado de $O(n \cdot \log n)$. Haciendo el cálculo para el primer ejemplo con matriz de 4x4 tendríamos 32 cálculos, con la matriz triangular obtenemos 20. Para 8x8, 128 para la completa y 72 para la triangular. Podemos deducir que siempre tendremos un número menor de cálculos para este caso.

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

DESCOMPOSICIÓN DE DOMINIO:

$$c = A \bullet b; \quad c_i = \sum_{k=0}^{M-1} a_{ik} \bullet b_k = a_i^T \bullet b, \quad c(i) = \sum_{k=0}^{M-1} A(i,k) \bullet b(k), \quad i = 0, \dots, N-1$$

$ \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} & a_{86} \end{bmatrix} $	$ \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix} $	$ \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \end{bmatrix} $	$ \begin{aligned} c(1) &= \sum_{k=0}^{M-1} A(1,k) \bullet b(k) \\ c(2) &= \sum_{k=0}^{M-1} A(2,k) \bullet b(k) \\ c(3) &= \sum_{k=0}^{M-1} A(3,k) \bullet b(k) \\ c(4) &= \sum_{k=0}^{M-1} A(4,k) \bullet b(k) \\ c(5) &= \sum_{k=0}^{M-1} A(5,k) \bullet b(k) \\ c(6) &= \sum_{k=0}^{M-1} A(6,k) \bullet b(k) \\ c(7) &= \sum_{k=0}^{M-1} A(7,k) \bullet b(k) \\ c(8) &= \sum_{k=0}^{M-1} A(8,k) \bullet b(k) \end{aligned} $
---	--	--	---

CAPTURAS DE PANTALLA:

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: Cada thread ejecutará una cantidad diferente ya que es una matriz triangular inferior. Por lo que la primera hebra ejecutará 1, la siguiente ejecutará 2 y así sucesivamente hasta la última hebra que ejecutará la cantidad igual al número de filas o columnas.

(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: Dependerá de la velocidad de como se ejecute cada thread, pero será más equitativo ya que por ejemplo la primera no solo ejecutará una iteración, si puede ejecutará más.

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

RESPUESTA: Según la teoría, `guided` dará mejores respuestas ya que provoca menos sobrecarga que `dynamic`. Además como empieza con mayores asignaciones al principio y las va reduciendo, se equilibra el reparto a los threads al tener las primeras iteraciones menos cálculos y las últimas más. Como se aprecia en las últimas columnas de la tabla del ejercicio 2.

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 utilizados 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