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): David Sánchez Jiménez
Grupo de prácticas: A3
Fecha de entrega: 03/05/2018
Fecha evaluación en clase: 04/05/2018

Ejercicios basados en los ejemplos del seminario práctico

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

CAPTURA CÓDIGO FUENTE: `if-clauseModificado.c`

```
#pragma omp parallel if (n > 4)                                    \
    num_threads(num_hebras) 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:

```
mar  1 may - 19:31  ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_3
@david  gcc -O2 -fopenmp -o if-clauseModificado if-clauseModificado.c

mar  1 may - 19:31  ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_3
@david  ./if-clauseModificado 3 4
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 master=0 imprime suma=3

mar  1 may - 19:31  ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_3
@david  ./if-clauseModificado 5 6
thread 0 suma de a[0]=0 sumalocal=0
thread 4 suma de a[4]=4 sumalocal=4
thread 3 suma de a[3]=3 sumalocal=3
thread 2 suma de a[2]=2 sumalocal=2
thread 1 suma de a[1]=1 sumalocal=1
thread master=0 imprime suma=10

mar  1 may - 19:31  ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_3
@david  ./if-clauseModificado 6 8
thread 5 suma de a[5]=5 sumalocal=5
thread 1 suma de a[1]=1 sumalocal=1
thread 4 suma de a[4]=4 sumalocal=4
thread 0 suma de a[0]=0 sumalocal=0
thread 2 suma de a[2]=2 sumalocal=2
thread 3 suma de a[3]=3 sumalocal=3
thread master=0 imprime suma=15
```

RESPUESTA: if ahorra la creación innecesaria de hebras cuando tenemos pocas iteraciones y, no sale rentable perder tiempo creando, sincronizando y destruyendo hebras. Por ejemplo, como se ve en la captura de pantalla, cuando sólo tenemos 3 iteraciones, la hebra master se encarga de ellas. Cuando ya tenemos un número mayor, sí que se hace de forma paralela. Con num_threads podemos decidir las hebras que se crearán, si pasamos este número por consola nos ahorramos tener que recompilar el programa cada bez que queramos cambiar dicho número.

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

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

RESPUESTA: La principal diferencia se ve en el reparto de iteraciones, ya que con static todas las hebras hacen las mismas iteraciones en round-robin. Con dynamic el orden y el reparto no se puede saber, lo unico que se puede saber es que como mínimo cada hebra hará chunk iteraciones. Lo mismo pasa con guided, con la diferencia de que las iteraciones están más “equilibradas” entre las hebras y el número de iteraciones que hace cada hebra eno es múltiplo de chunk.

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`

```
    if (omp_get_thread_num() == 0) {
        printf("Dentro de 'parallel':\n");
        printf("\tdyn-var: %d\n", omp_get_dynamic());
        printf("\tnthreads-var: %d\n", omp_get_max_threads());
        printf("\tthread-limit-var: %d\n", omp_get_thread_limit());
        omp_sched_t schedule_type;
        int chunk_size;
        omp_get_schedule(&schedule_type, &chunk_size);
        printf("\trun-sched-var: ");
        if (schedule_type == omp_sched_static) {
            printf("omp_sched_static\n");
        } else if (schedule_type == omp_sched_dynamic) {
            printf("omp_sched_dynamic\n");
        } else if (schedule_type == omp_sched_guided) {
            printf("omp_sched_guided\n");
        } else {
            printf("omp_sched_auto\n");
        }
        printf("\tchunk: %d\n", chunk_size);
    }
}

printf("Fuera de 'parallel' suma=%d\n", suma);
printf("\tdyn-var: %d\n", omp_get_dynamic());
printf("\tnthreads-var: %d\n", omp_get_max_threads());
printf("\tthread-limit-var: %d\n", omp_get_thread_limit());
omp_sched_t schedule_type;
int chunk_size;
omp_get_schedule(&schedule_type, &chunk_size);
printf("\trun-sched-var: ");
if (schedule_type == omp_sched_static) {
    printf("omp_sched_static\n");
} else if (schedule_type == omp_sched_dynamic) {
    printf("omp_sched_dynamic\n");
} else if (schedule_type == omp_sched_guided) {
    printf("omp_sched_guided\n");
} else { /*if (schedule_type == omp_sched_auto)*/
    printf("omp_sched_auto\n");
}
printf("\tchunk: %d\n", chunk_size);
```

CAPTURAS DE PANTALLA:

```
mié  9 may - 20:17  ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_3
@david  ./scheduled-clauseModificado 4 3
thread 1 suma a[3]=3 suma=3
thread 2 suma a[0]=0 suma=0
thread 2 suma a[1]=1 suma=1
thread 2 suma a[2]=2 suma=3
Dentro de 'parallel':
        dyn-var: 0
        nthreads-var: 4
        thread-limit-var: 2147483647
un-sched-var: omp_sched_dynamic
        chunk: 1
Fuera de 'parallel' suma=3
        dyn-var: 0
        nthreads-var: 4
        thread-limit-var: 2147483647
        run-sched-var: omp_sched_dynamic
        chunk: 1

mié  9 may - 20:17  ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_3
@david  export OMP_NUM_THREADS=2

mié  9 may - 20:18  ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_3
@david  ./scheduled-clauseModificado 4 3
thread 0 suma a[0]=0 suma=0
thread 0 suma a[1]=1 suma=1
thread 0 suma a[2]=2 suma=3
thread 1 suma a[3]=3 suma=3
Dentro de 'parallel':
        dyn-var: 0
        nthreads-var: 2
        thread-limit-var: 2147483647
un-sched-var: omp_sched_dynamic
        chunk: 1
Fuera de 'parallel' suma=3
        dyn-var: 0
        nthreads-var: 2
        thread-limit-var: 2147483647
        run-sched-var: omp_sched_dynamic
        chunk: 1
```

RESPUESTA: Se imprimen los mismos valores tanto dentro como fera del parallel, que son los indicados en las modificaciones de las variables de entorno de el shell.

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`

```
printf("\tomp_get_num_threads: %d\n", omp_get_num_threads());
printf("\tomp_get_num_procs: %d\n", omp_get_num_procs());
printf("\tomp_in_parallel: ");
if (omp_in_parallel()) {
  printf("true\n");
} else {
  printf("false\n");
}
```

CAPTURAS DE PANTALLA:

```
mié  9 may - 20:33  ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_3
@david  ./scheduled-clauseModificado4 3 4
thread 3 suma a[0]=0 suma=0
thread 3 suma a[1]=1 suma=1
thread 3 suma a[2]=2 suma=3

Dentro de 'parallel':
        omp_get_num_threads: 4
        omp_get_num_procs: 8
        omp_in_parallel: true
        dyn-var: 0
        nthreads-var: 4
        thread-limit-var: 2147483647
        run-sched-var: omp_sched_dynamic
        chunk: 1

Fuera de 'parallel' suma=3
        omp_get_num_threads: 1
        omp_get_num_procs: 8
        omp_in_parallel: false
        dyn-var: 0
        nthreads-var: 4
        thread-limit-var: 2147483647
        run-sched-var: omp_sched_dynamic
        chunk: 1
```

RESPUESTA: De las nuevas variables de este ejercicio, sólo se mantiene constante omp_get_num_procs(). El resto varían según estemos dentro o fuera de la región paralela.

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`

```
47    int set_dyn;
48    do {
49      printf("Introduce dyn-var: ");
50      scanf("%i", &set_dyn);
51      omp_set_dynamic(set_dyn);
52    } while (set_dyn > 1);
53
54    int n_threads;
55    printf("Introduce nthreads-var: ");
56    scanf("%i", &n_threads);
57    omp_set_num_threads(n_threads);
58
59    char sched_t[20];
60    printf("Introduce schedule_type: ");
61    scanf("%s", sched_t);
62    printf("Introduce chunk_size: ");
63    scanf("%i", &chunk_size);
64    if (strcmp(sched_t, "omp_sched_static") == 0) {
65      schedule_type = omp_sched_static;
66    } else if (strcmp(sched_t, "omp_sched_dynamic") == 0) {
67      schedule_type = omp_sched_dynamic;
68    } else if (strcmp(sched_t, "omp_sched_guided") == 0) {
69      schedule_type = omp_sched_guided;
70    } else if (sched_t == "omp_sched_auto") {
71      schedule_type = omp_sched_auto;
72    }
73    omp_set_schedule(schedule_type, chunk_size);
```

CAPTURAS DE PANTALLA:

```
jue 10 may - 21:44  ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_3
@david  ./scheduled-clauseModificado5 6 4
dyn-var: 0
nthreads-var: 8
run-sched-var:
        omp_sched_dynamic
        chunk: 1
Introduce dyn-var: 1
Introduce nthreads-var: 14
Introduce schedule_type: omp_sched_guided
Introduce chunk_size: 6
thread 0 suma a[0] suma=0
thread 0 suma a[1] suma=1
thread 0 suma a[2] suma=2
thread 0 suma a[3] suma=3
thread 5 suma a[4] suma=4
thread 5 suma a[5] suma=5
Fuera de 'parallel for' suma = 9
dyn-var: 1
nthreads-var: 14
run-sched-var:
        omp_sched_guided
        chunk: 6
```

RESPUESTA:

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`

```
for (i = 0; i < N; i++) {
  for (j = i; j < N; j++) {
    sol[i] += matriz[i][j] * vector[j];
  }
}

sol[N - 1] = matriz[N - 1][N - 1] * vector[N - 1];
```

CAPTURAS DE PANTALLA:

```
vie 11 may - 00:23  ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_3
@david  ./pmtv-secuencial 3
Matriz:
3 3 3
0 3 3
0 0 3
Vector:
5 5 5
Resultado:
45 30 15

vie 11 may - 00:23  ~/Escritorio/ETSIIT/2º Curso/2º Cuatrimestre/AC/Practicas/Practica_3
@david  ./pmtv-secuencial 7
Matriz:
3 3 3 3 3 3 3
0 3 3 3 3 3 3
0 0 3 3 3 3 3
0 0 0 3 3 3 3
0 0 0 0 3 3 3
0 0 0 0 0 3 3
0 0 0 0 0 0 3
Vector:
5 5 5 5 5 5 5
Resultado:
105 90 75 60 45 30 15
```

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:

CAPTURA CÓDIGO FUENTE: `pmtv-OpenMP.c`

DESCOMPOSICIÓN DE DOMINIO:

CAPTURAS DE PANTALLA:

TABLA RESULTADOS, SCRIPT Y GRÁFICA atcgrid

SCRIPT: pmvt-OpenMP_PCaula.sh

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

Chunk	Static	Dynamic	Guided
por defecto			
1			
64			

Chunk	Static	Dynamic	Guided
por defecto			
1			
64			

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`

CAPTURAS DE PANTALLA:

9. Implementar en paralelo la multiplicación de matrices cuadradas con OpenMP a partir del código escrito en el ejercicio anterior. Use las directivas, las cláusulas y las funciones de entorno que considere oportunas. Se debe paralelizar también la inicialización de las matrices. Dibuje en su cuaderno de prácticas la descomposición de dominio que ha utilizado en el código paralelo implementado para asignar tareas a los threads (Lección 4/Tema 2,Lección 5/Tema 2).

DESCOMPOSICIÓN DE DOMINIO:

CAPTURA CÓDIGO FUENTE: `pmm-OpenMP.c`

CAPTURAS DE PANTALLA:

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

ESTUDIO DE ESCALABILIDAD EN PCLOCAL:

SCRIPT: pmm-OpenMP_pclocal.sh