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): Elena Merelo Molina
Grupo de prácticas: 2
Fecha de entrega:
Fecha evaluación en clase:

Ejercicios basados en los ejemplos del seminario práctico

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

CAPTURA CÓDIGO FUENTE: `if_clause_modif.c`

```c
//Elena Merelo Molina

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

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

  if(argc != 3) {
    fprintf(stderr,"Número de argumentos incorrecto, ha de introducir iteraciones y threads\n");
    exit(-1);
  }

  n = atoi(argv[1]);
  x= atoi(argv[2]);

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

  /*Al imprimir omp_get_max_threads en mi pc sale 4, es el máximo número de threads
  con los que puedo trabajar.*/
  if( x > 4)
    x= 4;

  if (n>20)
    n=20;
```

```
  /*No se ejecuta en paralelo si n<= 4. Con default(none) hacemos que no se compartan
  o por defecto sean privadas las variables, de manera que nosotros mismos podamos
  establecer su visibilidad, poniendo sumalocal y tid privadas(con private(sumalocal, tid))
  y a, suma, n compartidas.*/
  #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();
    /*Con private(i) hacemos que cada hebra tenga un valor de i propio, schedule(static)
    hace que openMP divida el número de iteraciones de forma equitativa entre las hebras,
    aunque al poner nowait no han de esperarse unas a otras.*/
    #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 //Se espera a que todas las hebras terminen para imprimir el resultado
    #pragma omp master
    printf("thread master=%d imprime suma=%d\n",tid,suma);
  }
}
```

CAPTURAS DE PANTALLA:

Compilamos y generamos el ejecutable:

```
 2018-04-20 12:37:16  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
→ gcc -fopenmp -O2 if_clause_modif.c -o ../bin/if_clause_modif
```

Y ejecutamos con un número de threads igual a 14 (en mi portátil hay 4 threads/core, por ello le he puesto que si se introduce un número de threads mayor que 4 ponga x a 4):

```
 2018-04-20 13:13:25  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
→ ../bin/if_clause_modif 14 14
 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 1 suma de a[7]=7 sumalocal=22
 thread 2 suma de a[8]=8 sumalocal=8
 thread 2 suma de a[9]=9 sumalocal=17
 thread 2 suma de a[10]=10 sumalocal=27
 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 3 suma de a[11]=11 sumalocal=11
 thread 3 suma de a[12]=12 sumalocal=23
 thread 3 suma de a[13]=13 sumalocal=36
thread master=0 imprime suma=91
```

Se ve reflejado en que solo usa de las threads 0 a la 3, mientras que si por ejemplo establezco el número de threads a 2 solo usa las threads 0 y 1:

```
2018-04-20 13:14:04  elena in ~/Escritorio/University stuff/2º/2º Cuatrimestr
e/AC/Prácticas/BP3/src
○ → ../bin/if_clause_modif 14 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 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 1 suma de a[7]=7 sumalocal=7
 thread 1 suma de a[8]=8 sumalocal=15
 thread 1 suma de a[9]=9 sumalocal=24
 thread 1 suma de a[10]=10 sumalocal=34
 thread 1 suma de a[11]=11 sumalocal=45
 thread 1 suma de a[12]=12 sumalocal=57
 thread 1 suma de a[13]=13 sumalocal=70
thread master=0 imprime suma=91
```

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

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

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

RESPUESTA:

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`

CAPTURAS DE PANTALLA:

RESPUESTA:

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`

CAPTURAS DE PANTALLA:

RESPUESTA:

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`

CAPTURAS DE PANTALLA:

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`

CAPTURAS DE PANTALLA:

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).

10.

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