EE/CSCI 451
Spring 2016
Programming Homework 4

Assigned: February 21, 2016
Due: March 4, 2016, before 11:59 pm, submit via blackboard
Total Points: 25

1 Estimating π [10 points]

In this problem, you will use OpenMP directives to parallelize a serial program. The serial program is given in 'p1_serial.c', which uses the algorithm we discussed in Discussion 6 to estimate the value of π. You need use OpenMP to parallelize the loop between Line 28 and Line 32. To compile the program, type: gcc -lrt -fopenmp -o run p1_serial.c

1. Version 1: Use 4 threads and the **DO/for** directive to parallelize the loop. Name this version as 'p1_for.c'.
2. Version 2: Use 2 threads and the **SECTIONS** directive to parallelize the loop. Name this version as 'p1_section.c'.
3. Report the execution time of serial version, Version 1 and Version 2, respectively.

Hint: you may also need the **REDUCTION** data attribute.

2 Sorting [15 points]

In this problem, you need implement the quick sort algorithm [1] to sort an array in ascending order. Quick sort is a divide and conquer algorithm. The algorithm first divides a large array into two smaller sub-arrays: the low elements and the high elements; then recursively sorts the sub-arrays. The steps are:

- Pick an element, called a pivot, from the array.
- Reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.
- Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

In the given program 'p2.c', the array m ($size$ of $m = 16M$) which you need sort is generated.

1. Serial version: implement a Quicksort function to sort the array. The input of the function includes the pointer of the array, the index of the starting element and ending element. For example, **Quicksort(m, 20, 100)** will sort the array m from $m[20]$ to $m[100]$. Name this program as 'p2_serial.c'. Report the execution time of the serial version by calling Quicksort(m, 0, $size - 1$).

2. OpenMP version: randomly pick up an element of m, $m[rand()\%size]$, as the pivot, reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position, say f. Run 2 threads in parallel, one calling Quicksort$(m, 0, f-1)$ and the other calling Quicksort$(m, f, size-1)$. Name this program as 'p2_omp.c' and report its execution time. **Note that the Quicksort function is the same as the one used in serial version.**

Note: you can get more details about quick sort algorithm from textbook, Section 9.4.

3 Submission

You may discuss the algorithms. However, the programs have to be written individually. Submit your code and report via **Blackboard**. The report is preferred to be written in PDF. Your program should be written in C or C++. Make sure your program is runnable. Write clearly how to compile and run your code in the report as well. It is recommended to include a Makefile [2] along with your submission. If your program has error when we compile or run your program, you will lose at least 50% of credits.

References

[1] "Quicksort,"
`http://en.wikipedia.org/wiki/Quicksort`

[2] "Using make and writing Makefiles,"
`http://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html`