ENEE459P/699 Parallel Algorithms XMT-HW4 or OpenMP HW3

[MP7 for UIUC CS 420/CSE 402/ECE 492 Introduction to Parallel Programming for Science and Engineering]

Due Wednesday, December 8th 2010 at 11:59 p.m.

For questions about the assignment, please write to James Edwards, jedward5 at umd.edu

This is an individual assignment.

Problem Statement

You are required to implement Floyd's algorithm. To read about this algorithm you can refer to chapter 6 of the UIUC course textbook, or Chapter 25.2 in the CLRS textbook http://en.wikipedia.org/wiki/Introduction_to_Algorithms or the UIUC class lecture on November 3rd (the algorithm is on slide 6). For this exercise **you are given the option to choose to write your code either using OpenMP or using XMTC**. After you have made your choice, describe the reasons and the logic behind your decision of choosing one of the OpenMP and XMTC. Submit this description, your source code, and a table of your timings in as described in the Submission section.

Below are instructions for implementing the code using XMTC.

If you are implementing the code using OpenMP, please refer to MP 7 on the homepage of the UIUC course https://agora.cs.illinois.edu/display/cs420fa10/Assignments

The number of rows (equal to the number of columns) is stored in the input variable *N*. The input to your program is an *N*-by-*N* matrix stored in the two-dimensional array *a*[*N*][*N*], where *a*[*i*][*j*] is the element in the i^{th} row and the j^{th} column. The output of your program should be stored in that same array.

Four input data sets are provided. Data set d0 is provided only to aid you in debugging your program; your program will not be tested on this input.

Dataset	**d0**	**d1**	**d2**	**d3**
N	5	30	100	400
Serial clock cycles	N/A	306,156	14,003,113	1,270,113,072
Parallel clock cycles	N/A	60,200	1,246,758	62,096,567
Speedup	N/A	5.1	11.2	20.5

Language and Machines

Log into the class server (paraleap.ece.umd.edu) via SSH, copy the *floyd.tgz* file from the */opt/xmt/class/xmtdata/* directory, and extract it using the following commands:

cp /opt/xmt/class/xmtdata/floyd.tgz ~
tar xzvf floyd.tgz

This will create the directory *floyd* with *src* and *doc* folders. Put your *c* files in *src*, and your *txt* files in *doc*.

Use the XMTC language for coding your program. Write a serial version of Floyd's algorithm and save it as *floyd.s.c*. Write a parallel version and save it as *floyd.p.c*.

Hint: The following factors affect performance. If the number of threads is low relative to the number of processors (TCUs), increasing it improves performance. If the number of threads is sufficient to keep the TCUs busy, increasing it hurts performance as it increases the overall overhead costs of creating and terminating threads.

Testing for XMT version

- You have to implement the code using only one of the two choices – OpenMP or XMTC.
- Use the provided makefile (in the *src* directory) to compile your program
- Input data files are stored in the */opt/xmt/class/xmtdata/floyd* directory on the class server. This location is referenced in the makefile, so you do not need to refer to it explicitly.
- The output of your program should be the transformed adjacency matrix that contains the shortest path lengths.
- To run your code, execute the following command (replace *floyd.s.c* with the name of your source file and *d1* with the name of the data set):

  make run INPUT=floyd.s.c DATA=d1

- Put any calls to *printf* inside an *#ifdef PRINT_RESULT…#endif* block so that they do not affect the performance of your code while you are taking measurements. To see the output of your *printf* calls, append "PRINT_RESULT=1" to the end of your *make* command:

  make run INPUT=floyd.s.c DATA=d1 PRINT_RESULT=1

Submission

- Put your explanation of why you chose to implement this assignment in XMTC in a file named *README* in your *doc* directory.
- Create a table of your serial and parallel cycle counts for the d1, d2, and d3 data sets and save it as *table.txt* in your *doc* directory.
- Before you submit, check that you have the correct files in the correct locations with the following command:

  make submitcheck

- To submit, run the command

  make submit

- You may submit multiple times, but only your last submission will be graded.