COMS4040 Group Project

Hairong Wang

Hand-Out Date: March 17, 2017
Due Date: 24:00 pm, May 21, 2017

General Guidelines

1. You are expected to work in groups of 1 - 2 persons for this project. Some of the projects are suitable for those who are keen on working alone.
2. Each group is requested to work on one of the projects using
   (a) For Honours students, CUDA and MPI;
   (b) For Masters students, OpenMP, CUDA and MPI.
3. The final hand-in should include your source code and a detailed report.
4. In your report, proper citations and references must be given where necessary.
5. Note that the following descriptions of the problems are only outlines. Further elaborations are necessary.
6. Formulating your own project is allowed. It can be a particular problem given in other courses, or a problem close to your current Honours or Masters research, where you would like to bring in high performance computing. However, your final report to this project should have minimal overlap with your Honours or Masters research report. Send me a brief problem statement if this is the case for you.
7. Start early and plan your time efficiently. **The deadlines are going to be strictly applied with penalties of -4 to -8 marks for missing any of the deadlines**.

Deliverables and Evaluation

1. Presentation — Each group will be given a 15-20 minutes to present their project to the instructor or the class.
2. Report —
   - Problem statement
   - Methodology
   - Experimental setup
   - Results and discussions
3. Source codes with `Makefile` and `readme` files.
4. Total mark 20: Presentation (20%) + Report and Source code (80%).

Deadlines

There are three deadlines:

1. Friday, March 24, 2017 — group members and selection of a topic; Send me an email on this information by the deadline.
2. Friday, May 19, 2017 — a draft of report and presentation; The draft of report is to help you for your presentation, it can be in the form of presentation slides.
3. Sunday, 24:00 pm, May 21, 2017 — submission of final report and source code.

Problems

Projects with a '*' are suitable for students who prefer working alone.

Project 1: **Clustering*.** What *clustering* algorithms do is to form groups, given a set of data points in d-dimensional space. Clustering is used in diverse fields, such as data analysis, image processing etc. Your task in this project is to parallelize one of the clustering algorithms — *k-means* algorithm. See [7] for an efficient algorithm for k-means. Choose one of the data sets from UCI machine learning repository [13] to evaluate your implementations.

Project 2: **Sparse matrix multiplication*.** Sparse matrix is a matrix where the majority of the elements are zeros. If most of the elements in a matrix are non-zeros, then the matrix is dense. The dense matrix multiplication methods are usually considered not optimal for sparse matrix multiplication. This project explores implementing sparse matrix multiplication algorithms using proper sparse matrix representation methods. Read [14, 1] to understand the problem further.

Project 3: **K-Nearest neighbours search.** Given a set of N reference points $X := \{x_j\}_{j=1}^{N}$ and a query point x in a $d-$dimensional space, the nearest neighbour problem aims to find the set N_x of the k nearest neighbours of x. Nearest neighbour search has applications in data analytics, non-parametric statistics, and machine learning. This project explores a naive parallel solution for this problem [8].

Project 4: **Parallel graph algorithms.** Graph representations are common in many scientific and engineering applications, and the problems requiring graph data analytics are growing rapidly. The following projects explores the challenges and limitations of parallel graph algorithms for shared memory and distributed memory systems. (You only need to choose **ONE** of the following problems.)

.1 **Single-Source Shortest Paths***. Many problems can be expressed in terms of graphs, and can be solved using standard graph algorithms. This project will focus on parallelization of one of these algorithms. For a weighted graph $G = (V, E, w)$, the *single-source shortest paths* problem is to find the shortest paths from a vertex $v \in V$ to all other vertices in V. A *shortest path* from u to v is a minimum-weight path. In this project, you are to explore parallel formulation of *Dijkstra's single-source shortest paths algorithm* [5, Chapter 10] for undirected graphs with non-negative weights.

.2 **Connected component labeling.** Write both serial and parallel programs to solve the *connected component labelling problem*. Connected component labeling is used in computer vision to detect connected regions in binary digital images. A binary image is stored as an $n \times n$ array of 0s and 1s. The 1s represent objects, while the 0s represent empty space between objects. The connected component labelling problem is to associate a unique positive integer with every object. When the program completes, every 1-pixel will have a positive integer label. A pair of 1-pixels have the same label if and only if they are in the same component, where they are linked by a path of 1-pixels. Two 1-pixels are contiguous if they are adjacent to each other, either horizontally or vertically. For example, given the following table (Table 1), a valid output could be: Note that a 0 in a particular position of the input image results in a 0 in the same position in the output image. If 2 positions in the output image have the same integer value, it means there is a path of 1s between the two positions in the input image. An easy to follow explanation can be found at [2], and a divide and conquer approach is proposed in [11].

Original								Labeled							
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	2	0	2	0	0	0	0
0	1	1	1	0	0	0	0	0	2	2	2	0	0	0	0
0	1	1	0	1	1	1	1	0	2	2	0	10	10	10	10
0	0	0	0	1	0	0	1	0	0	0	0	10	0	0	10
1	1	1	0	1	1	0	1	31	31	31	0	10	10	0	10
1	1	1	1	0	1	1	1	31	31	31	31	0	10	10	10
0	0	0	0	0	0	1	1	0	0	0	0	0	0	10	10

Table 1: An example of connected component labelling

.3 **N-Queens problem.** The N-Queens problem is to place N queens on an $N \times N$ chessboard so that no two queens attack each other. If two queens are on the same row or column, they attack each other. For example, a solution for 4-queens problem is shown in Figure 1. It is obvious that the solution is not unique. Then the question is in how many different ways they can be placed. A naive algorithm for N-queens is to use brute force enumeration with pruning. It tries every possible arrangements of N-queens and checks if any of them satisfies the criteria. On a $N \times N$ board, there are N^2 locations. There will be N^2 possible locations for the first queen, $N^2 - 1$ for the second one, $N^2 - 2$ for the third , and so on. Thus, in a naive approach, we have to choose from

$$\binom{N^2}{N} = \frac{N^2!}{(N^2-N)!N!} \tag{1}$$

possible solutions. For $N = 10$, this number is $1.73e13$, and for $N = 21$, there are 314,666,222,712 possible solutions! Alternatively, we can place the queens one by one in different columns (or rows), starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false — *backtracking algorithm*. The purpose of this project is for you to explore parallelizing backtracking algorithm. That is, solving the N-queens problem in parallel. There are numerous discussions about implementing N-queens problem. A good place to start with is [3, 4].

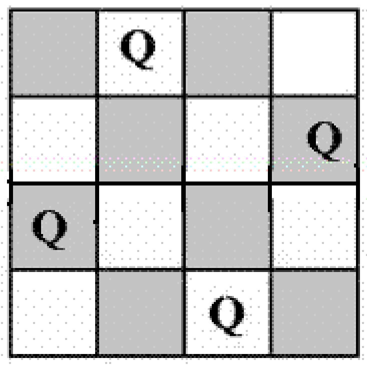

Figure 1: A solution of 4-Queens problem

.4 **Parallel breadth-first search** Breadth first search (BFS) finds applications in state space searching, graph partitioning, and is one of the most used graph operation in practical graph algorithms. Given a graph $G = (V,E)$ with a set V of n vertices, a set E of m directed edges, and a source vertex v_s, our goal is to traverse the vertices of G in breadth-first order at v_s. Each newly discovered vertex v_i will be labelled by its distance d_i from v_s and the predecessor vertex p_i immediately preceding it on the shortest path to v_s [9]. This project explores implementing parallel BFS algorithms using various parallel programming models.

.5 **Parallel branch and bound.** The branch-and-bound method is often used to find optimal solution to many optimization problem, especially in discrete and combinatorial optimization. It is a state space search method in which all the children of a node are generated before expanding any of its children. This project explores parallel branch and bound method for 15-puzzle, where 15 tiles, numbered 1-15, and a hole on a 4×4 grid. Read [12, Chapter 16] for further details. Other references are [6].

Project 5: **Feed-Forward Fully Connected Neural Network** An artificial neural network is an information processing method that was inspired by the way biological nervous systems function, such as brain, to process information. A neural network consists of two kinds of elements, neurons and connections, and it often has multiple layers. The connections between the neurons are assigned with weight values. These weight values need to be learned by training a neural network with sample inputs and predefined loss functions. This project explores parallel implementations of fully connected neural network for different parallel computing systems [10].

References

[1] Aydin Buluç, Jeremy T. Fineman, Matteo Frigo, John R. Gilbert, and Charles E. Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the Twenty-first Annual Symposium on Parallelism in Algorithms and Architectures*, SPAA '09, pages 233–244, New York, NY, USA, 2009. ACM.

[2] Code Project. http://www.codeproject.com/Articles/336915/Connected-Component-Labeling-Algorithm. Accessed on 2017-03-12.

[3] Dr.Dobb's. http://www.drdobbs.com/multicore-enabling-the-n-queens-problem/221600649?queryText=N-queens Accessed on 2017-03-12.

[4] GeeksforGeeks. http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/. Accessed on 2017-03-12.

[5] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing*. Addison Wesley, 2003.

[6] Pawan Harish and P. J. Narayanan. Accelerating large graph algorithms on the gpu using cuda. In *Proceedings of the 14th International Conference on High Performance Computing*, HiPC'07, pages 197–208, Berlin, Heidelberg, 2007. Springer-Verlag.

[7] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):881–892, JUly 2002.

[8] S. Liang, Y. Liu, C. Wang, and L. Jian. Design and evaluation of a parallel k-nearest neighbor algorithm on cuda-enabled gpu. In *2010 IEEE 2nd Symposium on Web Society*, pages 53–60, Aug 2010.

[9] Duane Merrill, Michael Garland, and Andrew Grimshaw. High-performance and scalable gpu graph traversal. *ACM Trans. Parallel Comput.*, 1(2):14:1–14:30, February 2015.

[10] Mike O'Neill. https://www.codeproject.com/kb/library/neuralnetrecognition.aspx. Accessed on 2017-03-11.

[11] JM Park, CG Looney, and HC Chen. Fast connected componenet labeling algorithm using a divide and conquer technique. In *Conference on computers and their applications*, 2000.

[12] Michael J. Quinn. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Education Group, 2003.

[13] UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/datasets/Iris. Accessed on 2017-03-12.

[14] Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, July 2005.