School of Computer Science and Applied Mathematics
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

COMS3008A Course Project

Hand-out date: October 12, 2020
Due date: Friday 23:55, Nov 6, 2020

Contents

1 Introduction

1. You are expected to work individually on this project.
2. The course project consists of TWO mini-projects, you are requested to complete both of them.
3. Hand-ins:
   (a) Source codes with `Makefile`, run or job scripts (`run.sh`), and `readme` files. (**Do not submit any code which does not compile!**)
   (b) Report —
   - General introduction
   - Parallelization approaches; you may use pseudo codes to describe your parallelization
   - Experimental setup (includes experimental data and performance evaluation approaches among others.)
   - Evaluation results and discussions.
4. In your report, proper citations and references must be given where necessary.
5. Total mark 100 which makes up 20% course weight.
6. Start early and plan your time effectively. Meet the deadline to avoid late submission penalties (20% to 40%).

2 Due Date

Friday, 23:55, November 6, 2020 — the submission of final report and source codes on sakai course site. (There will be two separate links provided for submission, one for report (with a Turnitin report), and the other for source codes).

3 Projects

1. **Mini-project one: Parallel Conway's Game of Life.** The Conway's Game of Life can be pictured as taking place on a two-dimensional board of cells. The cells, similar to biological cells, can live or die. The rules of Game of Life is simple. They determine when a live cell remains alive or dies from one generation to the next, and when a dead cell comes back to life. The number of alive neighbouring cells in a surrounding grid points (include 8 immediate horizontal, vertical, and diagonal neighbours) that holds a live cell is used to determine its state in the next generation according to the following:

   - A live cell remains alive if there are 2 or 3 live neighbours;
   - A live cell dies if the number of live neighbours is less than 2 (loneliness) or greater than 3 (over-crowding); and
   - A dead cell will birth a new cell if there are exactly 3 live neighbours.

   Implementing a Game of Life simulation in serial is trivial. **In this mini-project, you are requested to implement a Game of Life using MPI for a distributed memory system.** To do this, we can decompose the game board into smaller sub-boards, and then evaluate the cells in each sub-board in parallel. Apparently, the parallelization can happen within each time step (or generation), not across different generations. The only problem is the border cells. To update the border cells in each sub-board, we need information from the neighbouring sub-boards. So this means some information need to be communicated before the evaluation of the border cells in the sub-boards.

   The goal of this mini-project is to apply efficient data communication between different MPI processes to achieve performance; and evaluate the performance of your implementation via differing board size, and the number of MPI processes (or machine size).

2. **Mini-project two: Parallel Bitonic Sort.** The bitonic sort is based on the idea of sorting network. The bitonic sorting algorithm is suitable for parallel processing, especially for GPU sorting. **However, in this mini-project, you are requested to implement parallel bitonic sorting of integers using OpenMP and MPI, respectively.**

   The following is a brief introduction of bitonic sorting taken from [1, Chap. 9].

   - A sequence of keys $(a_0, a_1, \ldots, a_{n-1})$ is bitonic if
     (a) there exists an index m, $0 \leq m \leq n-1$ such that

     $$a_0 \leq a_1 \leq \cdots \leq a_m \geq a_{m+1} \geq \ldots a_{n-1},$$

     (b) or there exists a cyclic shift σ of $(0, 1, \ldots, n-1)$ such that the sequence $(a_{\sigma(0)}, a_{\sigma(1)}, \ldots, a_{\sigma(n-1)})$ satisfies condition 1. A cyclic shift sends each index i to $(i+s)$ mod n, for some integers s.
   - A bitonic sequence has two tones – increasing and decreasing, or vice versa. Any cyclic rotation of such networks is also considered bitonic.
   - $(1,2,4,7,6,0)$ is a bitonic sequence, because it first increases and then decreases. $(8,9,2,1,0,4)$ is another bitonic sequence, because it is a cyclic shift of $(0,4,8,9,2,1)$. Similarly, the sequence $(1,5,6,9,8,7,3,0)$ is bitonic, as is the sequence $(6,9,8,7,3,0,1,5)$, since it can be obtained from the first by a cyclic shift.
   - If sequence $A = (a_0, a_1, \ldots, a_{n-1})$ is bitonic, then we can form two bitonic sequences from A as

     $$A_{min} = (min(a_0, a_{n/2}), min(a_1, a_{n/2+1}), \ldots, min(a_{n/2-1}, a_{n-1})),$$

     and

     $$A_{max} = (max(a_0, a_{n/2}), max(a_1, a_{n/2+1}), \ldots, max(a_{n/2-1}, a_{n-1})).$$

     A_{min} and A_{max} are bitonic sequences, and each element of A_{min} is less that every element in A_{max}.
   - We can apply the procedure recursively on A_{min} and A_{max} to get the sorted sequence.
   - For example, $A = (6,9,8,7,3,0,1,5)$ is a bitonic sequence. We can split it into two bitonic sequence by finding $A_{min} = (min(6,3), min(9,0), min(8,1), min(7,5))$, which is $A_{min} = (3,0,1,5)$ (first decrease, then increase), and $A_{max} = (max(6,3), max(9,0), max(8,1), max(7,5))$, which is $A_{max} = (6,9,8,7)$ (first increase, then decrease).

Original sequence	3	5	8	9	10	12	14	20	95	90	60	40	35	23	18	0
1st Split	3	5	8	9	10	12	14	0	95	90	60	40	35	23	18	20
2nd Split	3	5	8	0	10	12	14	9	35	23	18	20	95	90	60	40
3rd Split	3	0	8	5	10	9	14	12	18	20	35	23	60	40	95	90
4th Split	0	3	5	8	9	10	12	14	18	20	23	35	40	60	90	95

Table 1: Merging a 16-element bitonic sequence through a series of $\log 16$ bitonic splits.

- The kernel of the network is the rearrangement of a bitonic sequence into a sorted sequence.
- We can easily build a sorting network to implement this bitonic merge algorithm.
- Such a network is called a *bitonic merging network*. See Table 1 for an example.
- The network contains $\log n$ columns. Each column contains $n/2$ comparators and performs one step of the bitonic merge.
- We denote a bitonic merging network with n inputs by $\oplus$BM[n].
- Replacing the $\oplus$ comparators by $\ominus$ comparators results in a decreasing output sequence; such a network is denoted by $\ominus$BM[n]. **(Here, a comparator refers to a device with two inputs x and y and two outputs x' and y'. For an increasing comparator, denoted by $\oplus$, $x' = min(x,y)$ and $y' = max(x,y)$, and vice versa for decreasing comparator, denoted by $\ominus$.)**
- The depth of the network is $\Theta(\log^2 n)$. Each stage of the network contains $n/2$ comparators. A serial implementation of the network would have complexity $\Theta(n \log^2 n)$. On the other hand, a parallel bitonic sorting network sorts n elements in $\Theta(\log^2 n)$ time. (The comparators within each stage are independent of one another, i.e., can be done in parallel.)
- How do we sort an unsorted sequence using a bitonic merge?
  - We must first build a single bitonic sequence from the given sequence. See Figure 1 for an illustration of building a bitonic sequence from an input sequence.
    - A sequence of length 2 is a bitonic sequence.
    - A bitonic sequence of length 4 can be built by sorting the first two elements using $\oplus$BM[2] and next two, using $\ominus$BM[2].
    - This process can be repeated to generate larger bitonic sequences.
  - Once we have turned our input into a bitonic sequence, we can apply a bitonic merge process to obtain a sorted list. Figure 3 shows an example.

To implement bitonic sorting in MPI, the basic idea is that you have much more elements than the number of PEs. For example sorting one million or even more data elements using a small number of MPI processes in our case, say 4, 8, or 16 processes. So, you need to fit the bitonic sorting idea into this kind of framework, not that you are sorting 16 elements only using 16 PEs. In your implementation, the main element from the perspective of distributed programming model is to figure out how to pair up a PE with its correct partner at each step. The overall idea is to begin by dividing the data elements among the PEs evenly first, then each PE sorts its share of elements in increasing or decreasing orders depending on its MPI process rank. In this way, the goal is to generate a global bitonic sequence owned by the entire MPI processes. For example, if you have 4 MPI processes, then you can have processes 0 and 1 jointly own a monotonic (say increasing) sequence; and processes 2 and 3 jointly own another monotonic (say decreasing) sequence. Then processes 0, 1, 2, and 3 jointly own a bitonic sequence. The remaining work will be then how to carry out the bitonic split steps. Lastly, note that you need to assume both the number of elements to be sorted and the number of PEs to be **power of two**.

4 General marking guideline

The weights of two mini-projects are distributed as mini-project one – 30% and mini-project two – 70%. Within each project, the mark will be distributed approximately as the following.

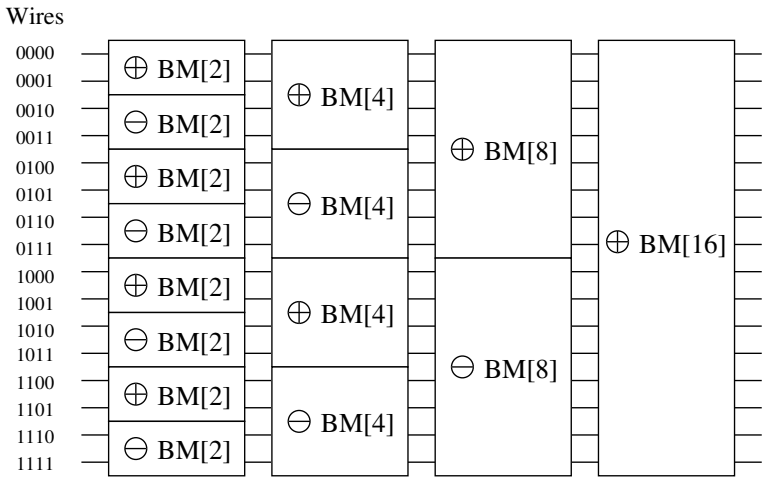

Figure 1: A schematic representation of a network that converts an input sequence into a bitonic sequence. In this example, $\oplus$BM[k] and $\ominus$BM[k] denote bitonic merging networks of input size k that use $\oplus$ and $\ominus$ comparators, respectively. The last merging network ($\oplus$BM[16]) sorts the input. In this example, $n = 16$.

1. In each implementation of the problems listed above, you are expected to include
   - a (correct) baseline (could be a serial implementation); [20%]
   - a parallel implementation with a validation of the correctness; (it is easy to add a function to verify your output list is sorted in the case of sorting; in the case of game of life, you can compare the outputs of serial and parallel outputs by starting from the same initial states etc.) [40%]
2. Report (only one combined report should be submitted): [30%]
   - Adequate writing to describe clearly the approach to solve the problem, i.e., parallelization methods, and other components for a short report;
   - performance evaluation by i) comparing the performance (speedup, efficiency etc.) of the baseline and parallel version; ii) testing the scalability by varying the input size or number of processing elements; iii) or others. (proper tables, line graphs, or bar graphs are good ways of presenting such results.)
3. Makefiles, run scripts, and job scripts (where applicable) are provided. [5%]
4. Other interesting inputs include but not limited to novelty, simulation of results (for game of life), and so on. [5%]

References

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

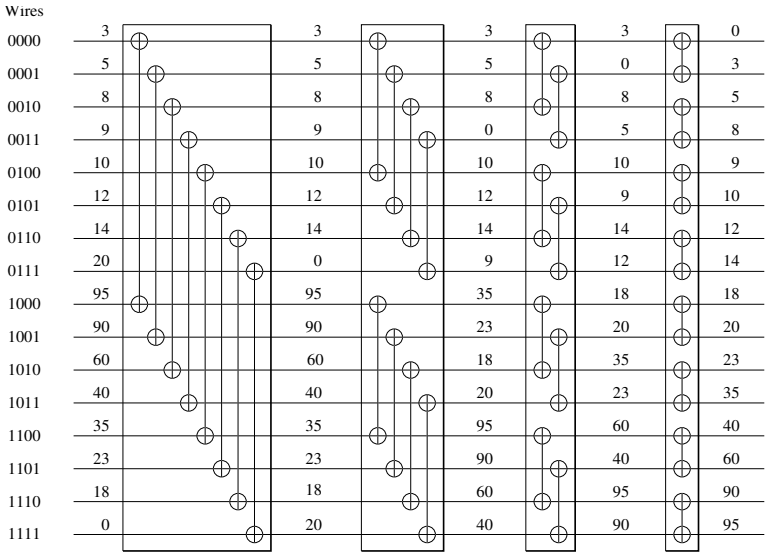

Figure 2: A bitonic merging network for $n = 16$. The input wires are numbered $0, 1 \ldots, n-1$, and the binary representation of these numbers is shown. Each column of comparators is drawn separately; the entire figure represents a $\oplus$BM[16] bitonic merging network. The network takes a bitonic sequence and outputs it in sorted order.

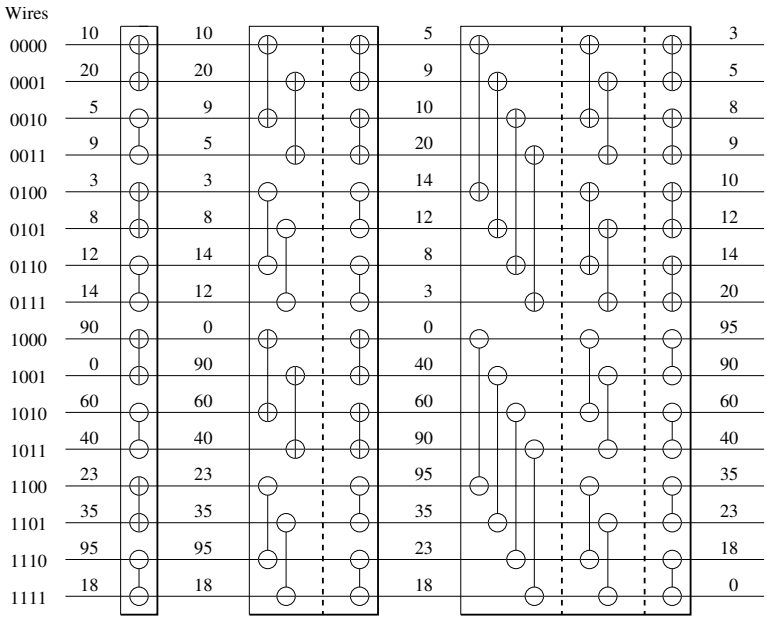

Figure 3: The comparator network that transforms an input sequence of 16 unordered numbers into a bitonic sequence.