4003-531/4005-735 | **Parallel Computing I** | **Assignment 1**
Term 20102 | **December 6, 2010**

Assignment 1

Due: December 17, 2010

Before starting, be sure to review the instructions for running Parallel Java on the RIT CS Parallel Computers (`http://www.cs.rit.edu/~ark/runningpj.shtml`). Remember, to execute programs on `parasite`, submit it to the job queue running on `paragon`. **Directly logging into `parasite` is not allowed.**

1. (20pts)

   Download `PrimeTesterSeq.java` and `PrimeTesterSmp.java` from the course website. You may find `primes.txt` (also available from the course website) a useful starting point for your investigation.

   (a) (10pts) Find a set of 8 integers to be used as command line arguments for `PrimeTesterSeq` and `PrimeTesterSmp` that satisfy the following conditions:

   - Each integer must be different.
   - Each integer must be greater than or equal to $2^{61} = 2305843009213693952$.
   - Each integer must be less than or equal to $2^{62} = 4611686018427387904$.
   - The running time of `PrimeTesterSmp` with these integers on `parasite` using 8 processors must exceed 45000msec.
   - The efficiency of `PrimeTesterSmp` with these integers on `parasite` using 8 processors must be greater than 0.95.

   Measure the running time T for `PrimeTesterSeq` with these integers on `parasite` and `PrimeTesterSmp` with these integers on `parasite` using 8 processors. [Recall that `parasite` has 8 processors and that `PrimeTesterSmp` will create the same number of threads as integers being tested.] Calculate *Speedup* and *Eff* as a function of K (the number of processors). Submit your results in a table organized as follows:

K	T (msec)	*Speedup*	*Eff*
seq		xxx	xxx
8			

   Explain why this set of integers leads to high efficiency.

   (b) (10pts) Find a set of 8 integers to be used as command line arguments for `PrimeTesterSeq` and `PrimeTesterSmp` that satisfy the following conditions:

   - Each integer must be different.
   - Each integer must be greater than or equal to $2^{61} = 2305843009213693952$.
   - Each integer must be less than or equal to $2^{62} = 4611686018427387904$.
   - The running time of `PrimeTesterSmp` with these integers on `parasite` using 8 processors must exceed 45000msec.
   - The efficiency of `PrimeTesterSmp` with these integers on `parasite` using 8 processors must be less than 0.175.

   Measure the running time T for `PrimeTesterSeq` with these integers on `parasite` and `PrimeTesterSmp` with these integers on `parasite` using 8 processors. [Recall that `parasite` has 8 processors and that `PrimeTesterSmp` will create the same number of threads as integers being

tested.] Calculate *Speedup* and *Eff* as a function of K (the number of processors). Submit your results in a table organized as follows:

K	T (msec)	*Speedup*	*Eff*
seq		xxx	xxx
8			

Explain why this set of integers leads to low efficiency.

Submission Submit a plain text file named `hw1-1.txt` or a PDF file named `hw1-1.pdf`. The `hw1-1` file should contain the set of 8 integers, the tabulated results, and an explaination for part (a) and the set of 8 integers, the tabulated results, and an explaination for part (b).

2. (20pts)

Building Parallel Programs, Part II Exercises:

(a) Exercise 58 (p. 292)

(b) Exercise 59 (p. 292)

(c) Exercise 60 (p. 292)

Submission Submit a plain text file named `hw1-2.txt` or a PDF file named `hw1-2.pdf`. The `hw1-2` file should contain the tabulated results for part (a), the tabulated results for part (b), and the tabulated results for part (c).

3. (30pts)

A cellular automaton is a discrete dynamical system. The state of the system consists of a regular grid of cells, each of which is in one of a finite number of states. At each time step, each cell simultaneously changes state based on its state and the states of neighboring cells, according to some fixed rule. Conway's Game of Life is a famous example of a cellular automaton. For more background on cellular automaton, see `http://en.wikipedia.org/wiki/Cellular_automaton` and `http://mathworld.wolfram.com/CellularAutomaton.html`.

The simplest (non-trivial) cellular automata are the elementary cellular automata. The state of such automata consists of a one-dimensional grid of cells, each of which is in one of two states (`1` or `0`). At each time step, each cell simultaneously changes state based on the states of the cell to its left, itself, and the cell to its right. Note that there are $2^3 = 8$ possible configurations of three cells. A rule consists of deciding, for each of these 8 configurations, whether the center cell becomes `1` or `0`. Thus, there are $2^8 = 256$ possible rules. One can represent each rule as an unsigned 8-bit integer — interpret the left, center, and right cells as an unsigned 3-bit integer called n and the new state of the center cell is the n^{th} bit of the 8-bit rule. For instance, rule $30 = 00011110_2$ corresponds to the following:

configuration of cells	111	110	101	100	011	010	001	000
new state of center cell	0	0	0	1	1	1	1	0

Write Java programs called `ElementaryCASeq` (a sequential program) and `ElementaryCASmp` (a parallel program) that executes elementary cellular automata. Both programs must take the following command-line arguments:

- *rule*: the rule to execute (an integer)
- *gridSize*: the size of the grid (an integer)

- *numSteps*: the number of steps to execute (an integer)

The initial state of the cellular automaton is a grid of *gridSize* cells, where exactly one cell is `1` and all other cells are `0`. When executing the cellular automaton, assume that the grid "wraps around"; that is, the cell to the left of the first cell of the grid (i.e., at index 0) is the last cell of the grid (i.e., at index *gridSize* $-$ 1) and the cell to the right of the last cell of the grid (i.e., at index *gridSize* $-$ 1) is the first cell of the grid (i.e., at index 0). After executing the elementary cellular automaton for *numSteps* steps, the program should print out the number of `1` cells in the final configuration and the running time of the program. For instance, here is a sample execution of `ElementaryCASeq`:

```
[mtf@fenrir code]$ java ElementaryCASeq 30 1000000 10000
9964
Running time: 61736 msec
```

Considering `ElementaryCASeq`, describe the sequential dependencies (if any).

Considering `ElementaryCASmp`, describe the parallel design pattern (or patterns) used.

Measure the running time T for `ElementaryCASeq` with command-line arguments `90 1000000 10000` on `parasite` and `ElementaryCASmp` with the same command-line arguments on `parasite` using 1, 2, 4, and 8 processors. Calculate *Speedup*, *Eff*, and *EDSF* as a function of K (the number of processors). Submit your results in a table organized as follows:

K	T (msec)	*Speedup*	*Eff*	*EDSF*
seq		xxx	xxx	xxx
1				xxx
2				
4				
8				

Submission Submit `ElementaryCASeq.java`, `ElementaryCASmp.java`, and a plain text file named `hw1-3.txt` or a PDF file named `hw1-3.pdf`. The `hw1-3` file should contain a description of the sequential dependencies of `ElementaryCASeq`, a description of the parallel design pattern(s) of `ElementaryCASmp`, and the tabulated results.

Submission

Submit a single ZIP file named `hw1.zip` to the `Homework 1` Dropbox on MyCourses by the due date. The `hw1.zip` file should contain:

- `hw1-1.txt` or `hw1-1.pdf`
- `hw1-2.txt` or `hw1-2.pdf`
- `ElementaryCASeq.java`
- `ElementaryCASmp.java`
- `hw1-3.txt` or `hw1-3.pdf`

The `hw1.zip` file should contain no additional files.

Document History

December 6, 2010
Original version

December 12, 2010
Introduction:
Added: "Remember, to execute programs on `parasite`, submit it to the job queue running on `paragon`. **Directly logging into `parasite` is not allowed.**"

December 12, 2010
Problem 1:
using 1, 2, 4, and 8 processors.
$\Rightarrow$ using 8 processors.

December 12, 2010
Problem 1:
Added: "[Recall that `parasite` has 8 processors and that `PrimeTesterSmp` will create the same number of threads as integers being tested.]"

December 15, 2010
Problem 3:

K	T (msec)	*Speedup*	*Eff*	*EDSF*
seq		xxx	xxx	xxx
1				

$\Rightarrow$

K	T (msec)	*Speedup*	*Eff*	*EDSF*
seq		xxx	xxx	xxx
1				xxx