4003-531/4005-735	**Parallel Computing I**	**Assignment 3**
Term 20102		**January 19, 2011**

Assignment 3

Due: February 2, 2011

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 the `thug`N backend nodes, submit it to the job queue running on `paranoia`. **Directly logging into the `thug`N backend nodes is not allowed.**

1. (30pts)

   Building Parallel Programs, Part III Exercises:

   (a) Exercise 21 (p. 574)

   (b) Exercise 22 (p. 574)

   (c) Exercise 23 (p. 574)

   (d) Exercise 24 (p. 574)

   (e) Exercise 25 (p. 574)

   (f) Exercise 26 (p. 574)

   Submission Submit a plain text file named `hw3-1.txt` or a PDF file named `hw3-1.pdf`. The `hw3-1` file should contain the solutions to the exercises.

2. (30pts)

Review the elementary cellular automata problem from Assignment 1. You may download the reference solution files `ElementaryCASeq.java` and `ElementaryCASmp.java` from the course website.

Recall the following:

- The `ElementaryCASeq` program requires $O(\textit{gridSize} \cdot \textit{numSteps} + \textit{gridSize})$ computation.
- The `ElementaryCASeq` and `ElementaryCASmp` programs allocated exactly two arrays of size *gridSize*.
- The `ElementaryCASmp` program used a parallel `for`-loop to parallelize the update of the cellular automaton state and a parallel `for`-loop to parallelize the summation of the number of `1` cells in the final configuration.
- A `fixed` schedule sufficed for the `ElementaryCASmp` program, because each individual cell update requires the same computation.

Write Java programs called `ElementaryCASeq`[1] (a sequential program) and `ElementaryCAClu` (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 $\textit{gridSize} - 1$) and the cell to the right of the last cell of the grid (i.e., at index $\textit{gridSize} - 1$) 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; in the cluster parallel program, only rank 0 should print out the total number of `1` cells in the final configuration, but each rank should print out the running time of the program. For instance, here are a sample executions of `ElementaryCASeq` and `ElementaryCAClu`:

```
[mtf@paranoia code]$ java ElementaryCASeq 30 200000 2000
Job 49, thug20
2021
Running time: 33414 msec
[mtf@paranoia code]$ java -Dpj.np=4 ElementaryCAClu 30 200000 2000
Job 50, thug21, thug22, thug23, thug24
Running time: 11017 msec; rank(3)
Running time: 11070 msec; rank(2)
Running time: 11085 msec; rank(1)
2021
Running time: 11081 msec; rank(0)
```

[1] You may use the provided reference solution, but a sequential program should begin timing *before* calling `Comm.init` when it is being compared to a cluster parallel program.

One advantage of cluster parallel computers over SMP parallel computers is that they may scale to much larger total memory; hence, with a cluster parallel computer, one may solve problems that require more total memory than can be accommodated in a single SMP parallel computer. One requirement of the `ElementaryCAClu.java` program is that it should not allocate (significantly) more than two arrays of size $gridSize/K$, where K is the number of processes in the cluster parallel program.

Considering `ElementaryCAClu`, give an expression for the amount of data (measured in bits) sent during the execution of the program. This formula should be a function of *gridSize*, *numSteps*, and K. Be sure to note if there are any special cases. You may ignore the communication-layer overhead.[2]

Measure the running time T for `ElementaryCASeq` with command-line arguments `90` *`gridSize numSteps`* on the `thug`N backend nodes and `ElementaryCAClu` with the same command-line arguments on the `thug`N backend nodes using 1, 2, 4, and 8 processors. Calculate *Speedup*, *Eff*, and *EDSF*. When measuring the running time T for `ElementaryCAClu`, be sure to record the running time of the rank which has the longest running time. Submit your results in a table organized as follows:

gridSize	*numSteps*	K	T (msec)	*Speedup*	*Eff*	*EDSF*
20000000	200	seq		xxx	xxx	xxx
20000000	200	1				xxx
20000000	200	2				
20000000	200	4				
20000000	200	8				
20000000	200	16				
2000000	2000	seq		xxx	xxx	xxx
2000000	2000	1				xxx
2000000	2000	2				
2000000	2000	4				
2000000	2000	8				
2000000	2000	16				
20000	200000	seq		xxx	xxx	xxx
20000	200000	1				xxx
20000	200000	2				
20000	200000	4				
20000	200000	8				
20000	200000	16				

Explain your results.

Submission Submit `ElementaryCASeq.java`, `ElementaryCAClu.java`, and a plain text file named `hw3-2.txt` or a PDF file named `hw3-2.pdf`. The `hw3-2` file should contain the formula for amount of data sent during the execution of the cluster parallel program, the tabulated results, and an explaination.

[2] That is, if your program performs a `world.send(toRank, intBuf)`, where `intBuf` is a `IntegerItemBuf`, then count that as 32 bits of data sent (one `int`), and not as 32 bits plus Parallel Java header bits plus TCP header bits plus IP header bits plus .... Note that Parallel Java sends a `boolean` as one (8 bits) `byte`.

Submission

Submit a single ZIP file named `hw3.zip` to the `Homework 3` Dropbox on MyCourses by the due date. The `hw3.zip` file should contain:

- `hw3-1.txt` or `hw3-1.pdf`
- `ElementaryCASeq.java`
- `ElementaryCAClu.java`
- `hw3-2.txt` or `hw3-2.pdf`

The `hw3.zip` file should contain no additional files.

Document History

January 19, 2011
Original version