Homework #6 CS 1645/2045
<u>Due: February 26, 2018</u> Spring 2017

Question 1:
Consider the loop:

```
a[0] = 0;
for (int i = 1; i < n; i++)
    a[i] = a[i-1] + i;
```

a. What is the loop-carried data dependence?
b. Can you see a way to eliminate this dependence and parallelize the loop?

Question 2:
Write a parallel program using OpenMP that implements Conway's Game of Life. The world is represented by a two-dimensional array. In each coordinate (x, y) a cell can be either live or dead.

A population of cells evolve according to the following rules:

I. Any live cell with fewer than two live neighbors dies (loneliness).
II. Any live cell with two or three live neighbors lives on to the next generation.
III. Any live cell with more than three live neighbors dies (crowding).
IV. Any dead cell with exactly three live neighbors becomes a live cell (reproduction).

Your program must apply the rules above to a community of cells that evolve through M generations. You should consider the 8 neighbors of each cell (i.e., including diagonal cells) when applying the evolution rules. If a neighbor of a cell is outside the limits of the world, it is considered to be dead. Make sure you follow these instructions:

a. Implement the game using life.cc as your starting point.
b. You should add OpenMP pragmas into the code in order to find the best way to parallelize it.
c. Generate a sequential version of your code by compiling without the -fopenmp flag.
    i. Note, this means you need to protect your imports and calls to omp methods. See slides for details on to accomplish this using `#ifdef _OPENMP`.
d. Your program must run with the following parameters:
    ./life <N> <M>
    or
    ./life <N> <M> <filename>
    The first case will create a random two-dimensional world size N × N, while the second case will read the cells from a file. In both cases, the program must simulate the evolution of the world during M generations.

e. Use the following command to analyze the performance of your code using different numbers of workers: export NUM_THREADS=<thread_count>

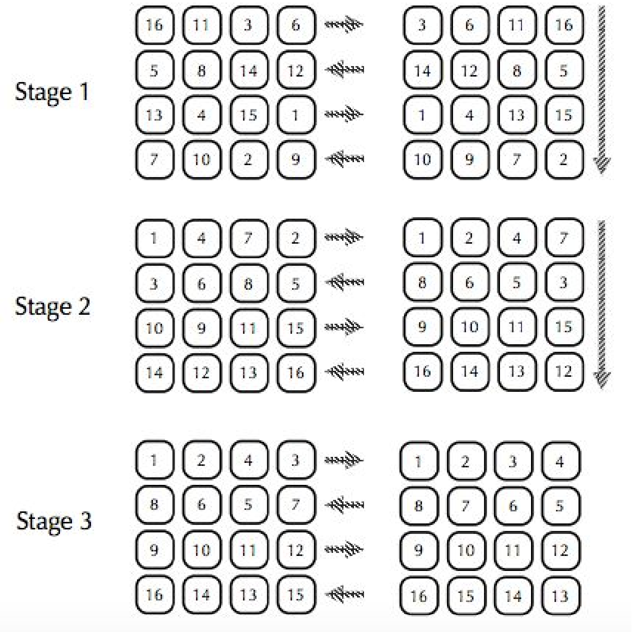
f. What parallel pattern does the algorithm follow? See slides for list of patterns.

Question 3:

The Shear Sort algorithm (also called Snake Sort) is a parallel sorting algorithm originally designed to run on a two-dimensional mesh of processors. However, the same algorithm can be used to sort N values (with N a square number) on a multiprocessor computer using any number of threads. Shear Sort arranges the original array of N = M^2 elements into a square M × M matrix A. Then, it proceeds to execute log2(N) stages. In each stage, the rows of the matrix are sorted (alternating increasing and decreasing order) and then the columns are sorted (all in increasing order). The following pseudo-code summarizes Shear Sort:

```
function shearSort(A, M):
  repeat log2(M*M) times:
    sortRowsAlternateDirection(A,M)
    sortColumns(A,M)
```

The final matrix A contains the elements sorted if the matrix is traversed row by row alternating directions (starting left-to-right). The figure below presents an example with the first 3 stages of Shear Sort on a 16-element input.



Stage 1	16	11	3	6	→	3	6	11	16
	5	8	14	12	←	14	12	8	5
	13	4	15	1	→	1	4	13	15
	7	10	2	9	←	10	9	7	2
Stage 2	1	4	7	2	→	1	2	4	7
	3	6	8	5	←	8	6	5	3
	10	9	11	15	→	9	10	11	15
	14	12	13	16	←	16	14	13	12
Stage 3	1	2	4	3	→	1	2	3	4
	8	6	5	7	←	8	7	6	5
	9	10	11	12	→	9	10	11	12
	16	14	13	15	←	16	15	14	13

a. Implement shear sort in shear.cc file.

b. Add OpenMP pragma commands to parallelize the algorithm, beware of loop dependencies.

c. Generate a sequential version of your code by compiling without flag -fopenmp.

d. Your program must run with the following parameters:
./shear <N>

or

```
./shear <N> <filename>
```

The first case will create a random two-dimensional matrix size sqrt(N) × sqrt(N) (making a total of N elements), while the second case will read the matrix from a file. In both cases, the program must sort the numbers in the matrix in “snake sort” order. See example result outputs for examples.

e. Use the following command to analyze the performance of your code using different numbers of threads: OMP_NUM_THREADS=16 ./shear
   You can update the shear.batch file to see the effect on stampede.

Write-up and Code Submission

1. Please submit all code in the dropbox as previously done.
2. Please provide a written report which answers the questions asked in the homework, including any analysis.