Introduction to High Performance Computing Systems
CS 1645

University of Pittsburgh
Department of Computer Science
Spring, 2015
Homework 3

Instructions

- The solutions to all the problems in this homework should be the result of the student's individual effort. Presenting the words or ideas of somebody else under your name hinders your skills in academic and professional work and violates the university's policy on academic integrity: `http://www.provost.pitt.edu/info/ai1.html`
- The submission process for this homework will use the SVN repository: `https://collab.sam.pitt.edu/svn/CS1645/users/<pittID>` Create a `hw3` directory into your repository.
- You have to check-in the following files into the `hw3` directory:
  - File `determinant.cpp` containing the parallel code for computing the determinant of a matrix.
  - File `shear.cpp` containing the parallel code for the shear sort algorithm.
  - File `report.pdf` with a report summarizing your results (it needs to be a PDF file).

  by 11:59pm **Wednesday February 18, 2015**.
- Late submissions will not be accepted.
- Your code ***MUST*** run on the Stampede system with the software infrastructure provided (Makefile, main files, and other utility files). You should use the files provided in the following repository: `https://collab.sam.pitt.edu/svn/CS1645/assignments/hw3`
- Consider using interactive job submissions for developing and debugging your code: `idev -A TG-CIE140012`

1 Determinant

In linear algebra, the determinant is a value associated with a square matrix. It can be computed from the entries of the matrix by a specific arithmetic expression. The determinant provides important information: *i*) about a matrix of coefficients of a system of linear equations, or *ii*) about a matrix that corresponds to a linear transformation of a vector space. In the first case the system has a unique solution exactly when the determinant is nonzero; when the determinant is zero there are either no solutions or many solutions. In the second case the transformation has an inverse operation exactly when the determinant is nonzero. [1]

A traditional way to compute the determinant of a matrix is to use the minors of the matrix. Given a $N \times N$ matrix A, the minor M_{ij} is defined as a $(N-1) \times (N-1)$ matrix obtained after removing row i and column j from matrix A. Thus, the determinant of matrix A, denoted by $\det(A)$ or $|A|$ can be computed by:

$$\det(A) = |A| = \sum_{j=1}^{N} (-1)^{j+1} a_{1j} |M_{1j}|$$

For example, given the following matrix:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

its determinant is given by $|A| = a_{11}|M_{11}| - a_{12}|M_{12}| + a_{13}|M_{13}|$.

Write a parallel OpenMP program that computes the determinant of a square matrix.

Make sure you follow these instructions:

a) You should implement a function (may be parallel) to compute the determinant of the minors.

b) You should add OpenMP directives into the code in order to find the *best* way to parallelize it.

c) Write your code in the provided file `determinant.cpp`.

d) Generate a sequential version of your code by compiling without flag `-openmp`.

e) Your program must run with the following parameters:
`./determinant <N>`
or
`./determinant <N> <file>`
The first case will create a random matrix size $N \times N$, while the second case will read the matrix from a file.

f) Use the following command to analyze the performance of your code using different numbers of threads:
`OMP_NUM_THREADS=16 ./determinant <N>`

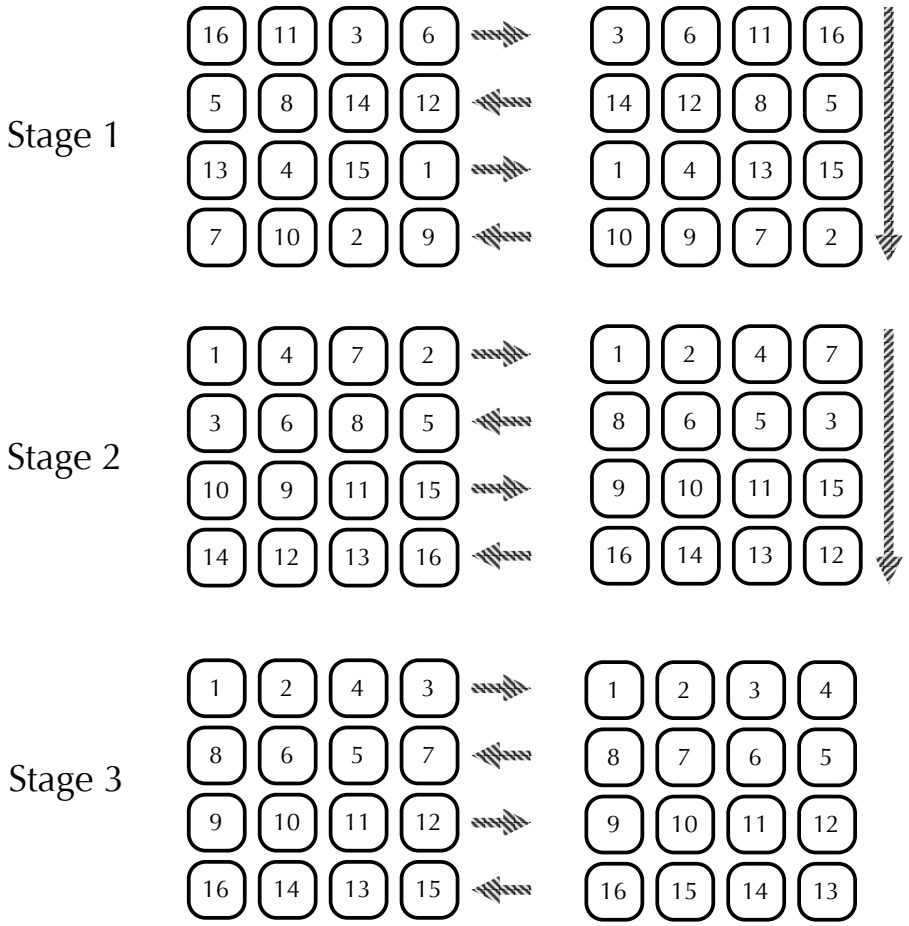
[1] Extracted from Wikipedia

2 Shear Sort

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 $M \times M$ matrix A. Then, it proceeds to execute $log_2(N)$ 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.



Write a parallel OpenMP program that implements Shear Sort.
Make sure you follow these instructions:

a) You should implement a sorting algorithm (may be parallel) to sort each row and column.

b) You should add OpenMP directives into the code in order to find the *best* way to parallelize it.

c) Write your code in the provided file `shear.cpp`.

d) Generate a sequential version of your code by compiling without flag `-openmp`.

e) Your program must run with the following parameters:
`./shear <N>`
or
`./shear <N> <file>`
The first case will create a random array (stored as matrix) size N, while the second case will read the array from a file. In any case, N must be a square number.

f) Use the following command to analyze the performance of your code using different numbers of threads:
`OMP_NUM_THREADS=16 ./shear <N>`

Report

You should create a report with the following sections for each of the two programs above:

1. A general strategy of the parallelization effort. Why did you choose those directives to parallelize the program?
2. A speedup analysis. Using 3 different *interesting* problem sizes for both problems and these number of threads $\{1, 2, 4, 8, 16\}$, make a speedup plot.
3. An efficiency analysis. Using 3 different *interesting* problem sizes for both problems and these number of threads $\{1, 2, 4, 8, 16\}$, make an efficiency plot.
4. A description of the performance bottlenecks. What is preventing the program from getting linear speedup?