CS 380P Parallel Systems — Spring 2013 **Programming Assignment #3**
Sparse Matrix-Vector Multiplication using Pthreads **Due February 18, 2013**

In this assignment, you will implement a **sparse matrix-vector multiplication** algorithm of your choice using Pthreads. This assignment will force you to think about load balance and the use of sophisticated data structures. You should optimize your code to run as fast as possible on Lonestar and Stampede. To prevent you from over-specializing your solution for specific inputs, the TA will post the inputs that you should use for timings shortly before the due date. You may, however, submit different solutions for the two machines. Note that the TA may run your program on other inputs that are not made available to you.

You may work with a partner if you wish.

1 Background

A sparse matrix is one that is populated with many zeros. Large sparse matrices are often encountered in scientific computations, and special data structures are used to compactly represent such matrices. Sparse matrices can have a wide range of sparsity, which is defined as the percentage of zero entries with respect to the total number of entries in the matrix. Thus, a sparsity value of 90% means that 90% of the matrix entries are zeros. For this assignment, you can assume that sparsities are at least 70%.

2 Sparse Matrix-Vector Multiplication Algorithms

You may use any sparse matrix-vector multiplication algorithm that you wish, so you are free to consult the literature or devise your own algorithm. However, you are not allowed to reuse existing code except to translate and generate matrices. To help you get started, you might find the following material useful[1]:

- `http://netlib.org/linalg/html_templates/node98.html#SECTION00932100000000000000`
  You will also find other types of representations of a sparse matrix in memory.
- `http://www.eecs.harvard.edu/~ellard/Q-97/HTML/root/node20.html#smmAlg`

If you'd like a challenge, read the following for ideas:

- "Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms."
  Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, James Demmel. SC, Nov 2007.
- `http://www.cs.sandia.gov/CSRI/Workshops/2008/CSCAPES/presentations/wolf_cscapes08.pdf`
- "Efficient Sparse Matrix-Vector Multiplication on CUDA"
  Nathan Bell and Michael Garland, in, "NVIDIA Technical Report NVR-2008-004", December 2008

3 Sparse Matrix Representation

Many sparse matrix representations have been proposed, and you may use the format of your choice. In a day or two, we will provide a few input sparse matrices that use the Matrix Market Exchange Format (MMEF), which you can use to test your correctness and performance. Of course, when we grade your solutions, we will not limit ourselves to these few test cases. To learn more about sparse matrix formats, including the MMEF, please see the following URL: `http://math.nist.gov/MatrixMarket/formats.html`.

Because we are allowing you to choose your representation, you may find the following library useful, because it claims to be able to translate among matrices of multiple formats: `http://bebop.cs.berkeley.edu/smc/`

[1] In case your PDF reader is having problems with any of these links, the TA will post all of these links to our Piazza page

4 Generating Large Matrices

You can expect the sparse matrix dimensions to range from 1000 to 10000. The TA's page will point you to a tool that randomly generates sparse matrices. You can use this tool to generate input matrices with varying sparsities.

4.1 Matrix Assumptions

- You should not assume any special matrix properties, For example, you cannot assume that the matrices are symmetric, diagonal, banded, etc.
- You can assume that the input matrices will be in MMEF's Coordinate format, but you can **not** assume that the rows and columns will be sorted.
- You can assume that each matrix elements is a **double** data type.
- The sparsity may be as low as 70%, which is to say that you can't make any strong assumptions about sparsity.

4.2 Running code

Your program should accept 3 command line parameters.

- command_line_param1
  Input file containing the sparse matrix in MMEF format.
- command_line_param2
  Input file containing the vector.
  The format is as follows:
  number_of_elements(first line of the file)
  1.1 2.4 ...... (elements separated by space)
- command_line_param3
  Output file name

5 Details

As usual, you may develop your code on any platform, but we will use the Lonestar and Stampede clusters to grade your solutions, so be sure that what you send us works on these machines.

Important: Your solution should include code that reports the elapsed time for your multiplication, excluding file I/O time. To allow us to check for correctness, your solution should write the final result to a file in MMEF's **Coordinate Format**, and all row coordinates should be listed in increasing order, and within each row all column coordinates should be given in increasing order.

What to Turn In

This assignment is due at 11:59pm on the due date. Use the **turnin** program to submit your solution, which should include the following:

1. A written report of the assignment in either plain text or PDF format. This is your chance to explain your approach, your optimizations, any insights gained, problems encountered, etc. Your report should include performance results for your solution on both Lonestar and Stampede.
2. Your source code, instructions to build it on Lonestar and Stampede, and instructions to run the programs (along with the arguments, etc). You may submit different codes for Lonestar and Stampede, in which case your report should explain why your codes differ for the two machines.
3. More detailed instructions will be provided by the TA via Piazza.

Extra Credit

To receive extra credit, run your solution on the supplied inputs and submit your code and your results by 11:59pm on Friday February 15. The TA will post some of the best times on the Piazza page, which will hopefully show other students what to shoot for. Of course, you will still have a few days to optimize your code, so your results here do not have to be your final results. We will give additional extra credit for the best times that are submitted on the 15th, as well as extra credit for the best times for your final submissions (which the TA will run).