

In this assignment, you will implement a **sparse matrix-vector multiplication** algorithm of your choice using Pthreads. Unlike dense matrix operations, sparse matrix operations will force you to use more sophisticated data structures and force you to think about load balance. 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:

- [http://netlib.org/linalg/html\\_templates/node98.html#SECTION00932100000000000000](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\\_csapes08.pdf](http://www.cs.sandia.gov/CSRI/Workshops/2008/CSCAPES/presentations/wolf_csapes08.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/>

## 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 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 seperated 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 Longhorn Linux cluster at TACC to grade your solutions, so be sure that what you send us works on Longhorn.

**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, any insights gained, problems encountered, etc. Your report should include performance results for your solution.
2. Your source code, instructions to build it on Longhorn, and instructions to run the programs (along with the arguments, etc).
3. More detailed instructions will be mailed by the TA.

## Extra Credit

To receive extra credit, run your solution on the supplied inputs (which will be made available no later than Friday the 18th) and submit your code and your results by 11:59pm on Saturday February 19. The TA will post some of the best times on her website, 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 19th, as well as extra credit for the best times for your final submissions (which the TA will run).