

HW1: Matrix-Vector Multiplication

Course: ENEE759K/CMSC751
Title: Matrix-vector multiplication (matvec)
Date Assigned: February 10th, 2009
Date Due: February 24th, 2009
Contact: Fuat Keceli – keceli (at) umd (dot) edu

1 Assignment

Your assignment is to implement a parallel algorithm in XMTC to multiply a sparse matrix with a dense vector¹. Your parallel algorithm should be as fast as possible. Use the data structures described in the next section. Your implementation should satisfy the following:

- Each row will be handled by a single thread.
- No thread will handle more than one row.

2 Data structures

In this assignment the sparse matrix is represented using the following three data structures:

- **rowptr array:** For each row i of the sparse matrix, $rowptr[i]$ contains the index number of the first nonzero element in this row. This index can be used in **col_ind** and **values** arrays (see below). If row i does not contain a non-zero element (i.e. it is all zeros), then $rowptr[i] == rowptr[i + 1]$. This array has $m + 1$ elements where m is the number of rows in the matrix. The last element of this row points to the outside of the **col_ind** and **values** array to indicate that there are no more non-zero numbers.
- **col_ind array:** This array contains the column indices of the non-zero elements. If the matrix element at row i column j is a non-zero element, then for some k such that $0 \leq k < rowptr[i + 1] - rowptr[i]$, $col_ind[rowptr[i] + k] == j$.
- **values array:** This array contains the values of non-zero elements. This array is indexed similar to the **col_ind array**. If the matrix element at row i column j has the non-zero value v , then for some k such that $0 \leq k < rowptr[i + 1] - rowptr[i]$, $values[rowptr[i] + k] == v$.

Consider the 6x7 sparse matrix in Figure 1. The above described data structures corresponding to this matrix can be seen in Figure 2.

¹A related assignment was first given at the University of California, Santa Barbara at the end of a graduate course based on the parallel programming language MPI. The current assignment was developed to allow comparative study of program development-time for XMTC vs MPI. See <http://www.cs.umd.edu/basili/publications/proceedings/P119.pdf>

$$A = \begin{bmatrix} 0 & 0 & 3 & 0 & 1 & 4 & 0 \\ 0 & 2 & 0 & 0 & 6 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 2 & 0 \end{bmatrix}$$

Figure 1: Example 6x7 sparse matrix

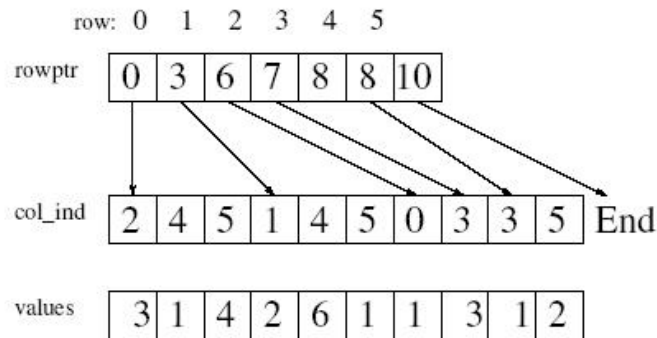


Figure 2: Implementation

2.1 Setting up the environment

The header files and the binary files can be downloaded from \sim *george/xmtdata*. To get the data files, log in to your account in the class server and copy the *matvec.tgz* file from directory using the following commands:

```
$ cp ~george/xmtdata/matvec.tgz ~
$ tar xzvf matvec.tgz
```

This will create the directory *matvec* with following folders: *data*, *src*, and *doc*. Data files are available in data directory. Put your *c* files to *src*, and *txt* files to *doc*.

3 Questions

1. Serial implementation:

- Describe the serial algorithm of *matvec* in file algorithm.s.txt
- Provide a brief work and time complexity analysis of this algorithm. Append this analysis to the file algorithm.s.txt
- Write the XMTC serial program that executes this algorithm. Use matvec.s.c that is given to you in src. Write your code to the place indicated in the file. Please do not modify the marked region, you will use that region to check the correctness of your program.
- Run this program using 4 sets of data given in the Input section.
- Collect the number of clock cycles for each run and fill out the table in doc/table.txt using this information (see Output section).

2. Parallel implementation:

- (a) Describe the parallel algorithm of matvec in file algorithm.p.txt
- (b) Provide a brief work and time complexity analysis of this algorithm. Append this analysis to the file algorithm.p.txt
- (c) Write the XMTC parallel program that executes this algorithm. Use matvec.p.c that is given to you in src. Write your code to the place indicated in the file. Please do not modify the marked region, you will use that region to check the correctness of your program.
- (d) Run this program using 4 sets of data given in the Input section.
- (e) Collect the number of clock cycles for each run and fill out the table in doc/table.txt using this information (see Output section).

3.1 Input format

<code>#define m</code>	The number of rows in the sparse matrix
<code>#define n</code>	The number of columns in the sparse matrix
<code>#define nnz</code>	The number of non-zero elements in the sparse matrix
<code>int rowptr[m+1]</code>	For each row this arrays contains an index number for <code>col_ind</code> and values arrays
<code>int col_ind[nnz]</code>	This array contains the column indices of each non-zero element
<code>int values[nnz]</code>	This array contains the values of each non-zero element
<code>int vector[n]</code>	This array contains the values of each element of the column vector, that you're going to multiply with the sparse matrix
<code>int result[m]</code>	The result vector. The result of the matrix-vector multiplication will be written into this vector

Table 1: Input format

You can declare any number of global arrays and variables in your program as needed. The number of elements in the arrays (m , n , and nnz) are declared as a constant in each dataset, and you can use those to declare auxiliary arrays. For example, this is valid XMTC code:

```
#define N 16384

int temp1[16384];
int temp2[2*N];
int pointer;

int main() {
    //...
}
```

3.2 Data sets

Run all your programs (serial and parallel) using the following data files. You can directly include the header file into your XMTC code with `#include` or you can include the header file with the compile option `-include`. The binary file that contains the data (.xbo file) should be passed to the compiler as shown in “Testing the program” section.

Description	Data Set	Header File	Binary file	Max. non-zero elements / row
Small	$m = 50, n = 100$ $nnz = 110$	data/small/matvec.h	data/small/matvec.xbo	5
Medium	$m = 400, n = 100$ $nnz = 826$	data/medium/matvec.h	data/medium/matvec.xbo	9
Large	$m = 10000, n = 100$ $nnz = 19872$	data/large/matvec.h	data/large/matvec.xbo	10
X-Large	$m = 30000, n = 100$ $nnz = 60130$	data/xlarge/matvec.h	data/xlarge/matvec.xbo	10

Table 2: Header files

4 Testing the program

You can test the correctness of your programs with the result data given in data sets as follows:

```
> xmtcc matvec.p.c -include ../data/small/matvec.h ../data/small/matvec.xbo \
-D PRINT_RESULT -quiet -o matvec.p
> xmtfpga matvec.p.b -o myFileSmall.txt
> diff -b myFileSmall.txt ../data/small/resultFileSmall.txt
```

If the *diff* command does not give any output, it means that you have the same result value and your program is right. Don't forget the *-b* option.

IMPORTANT: The FPGA has limited reserved space for standard output right now. This causes problems when while using printf statements to test the output of larger programs, such as matvec for the XLarge testcase. In particular, if you run your reference solution with that dataset and compare the outputs, you will notice that the result is truncated and *diff* with the provided result will fail. **Use the standard output only to test the first 3 dataset (small, medium and large), and not the largest one (xlarge).**

4.1 Output

Fill the following table: A text file named `table.txt` in `doc` is already created for you. Fill out the table in this text file using white spaces to indent the fields. This text file will be parsed automatically by a script so it is important to adhere to the format. **Remove any *printf* statements from your code while taking these measurements.** Printf statements increase the clock count. Therefore the measurements with printf statements may not reflect the actual time and work done.

Input size	Small	Medium	Large	X-large
matvec.s.c				
matvec.p.c				

Table 3: Clock cycles will be written to table.txt

Note that, a part of your grading criteria is the performance of your parallel implementation. Therefore you should try to obtain the fastest running parallel program. As a guideline, following are the cycle counts for our reference serial and parallel implementations on the FPGA computer. Serial implementation: 3653342 clock cycles, parallel implementation: 214319 clock cycles, parallel speedup: $\sim 17x$.

4.2 Submission

The use of the make utility for submission *make submit* is required. Make sure that you have the correct files at correct locations (*src* and *doc* directories) using the *make submitcheck* command. Run following commands to submit the assignment:

```
$ make submitcheck  
$ make submit
```