Assignment 1 – CSE 436/536, Summer 2016

**Due: 11:55PM May 25 Wednesday, 2016**

**2-Dimensional Matrix Addition and Multiplication, and Matrix Vector Multiplication**

In this assignment, you will implement three C functions for performing 2-dimensional matrix operations: addition, multiplication and matrix vector multiplication. The function signatures are already given as follows and you will use the sum.c example as the starting point for creating your implementation of the three functions.

/\* C[N][M] = A[N][M] + B[N][M] \*/

void matrix\_addition(int N, int M, REAL \* A, REAL \* B, REAL \* C, int A\_rowMajor, int B\_rowMajor);

/\* C[N][M] = A[N][K] \* B[K][M] \*/

void matrix\_multiplication(int N, int K, int M, REAL \* A, REAL \* B, REAL \* C, int A\_rowMajor, int B\_rowMajor);

/\* C[N] = A[N][M] \* B[M] \*/

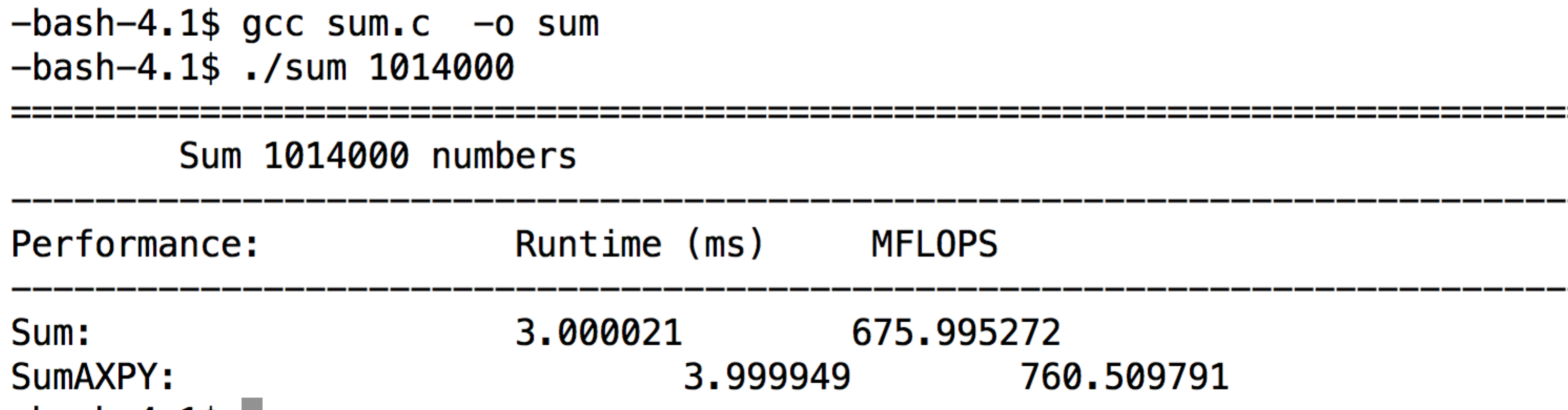
void mv\_multiplication (int N, int M, REAL \*A, REAL \*B, REAL \*C, int A\_rowMajor);

The memory of each matrix (A, B, or C) is provided using REAL \* A, and its dimensions are specified as function parameters (N, K, or M). The elements of a 2-dimensional matrix could be stored in either row major or column major, identified using A\_rowMajor (or B\_rowMajor or C\_rowMajor). If A\_rowMajor is non-zero, elements of A are stored in row major, otherwise, they are stored in column major. Your function should be able to handle any combination of the storage type of each matrix in the arguments, e.g. there are four cases for matrix\_addition and matrix\_multiplication function and two cases for mv\_multiplication function. The results in matrix C should be always in row major.

The sum.c file used during the class includes helper functions for initializing the matrix and for timing the execution, and a skeleton main function. You can use them for your implementation. In sum.c, array are statically declared (e.g. REAL X[N]). In your implementation, memory should be dynamically allocated and freed (using malloc and free) for the matrices (A, B, and C) so you can handle large matrices.

You can compile and build the executable using gcc compiler (e.g. gcc -O0 matmul.c -o matmul), in which –O0 is the flag optimization level 0 (no optimization). You can try –O2 or –O3 optimization flag. Your executable should be able to run with two arguments, N and M for specifying the sizes of the matrices (K should be always the same as N for the sake of simplicity to handle arguments).

The output of your program should include both the time of computation (ms) and FLOPs/s performance. See below example for sum.c



Your output should include the 10 entries, considering the total 10 cases of the combination of row/column major for the storage type of the matrices.

**Submission:** Your submission should include two files: 1) The matmul.c file that contains your implementations of the three functions and the main function that call the three functions with different parameters, and 2) a max 3-page report. The report should include:

1. Short description on how you implement each of the three functions
2. Performance report using three figures when running with at least 512 (N=512 and M=512), 1024x1024 and 2048x2048 matrix sizes (more sizes of configuration are welcome). Figure 1 reports the execution times in ms for matrix addition when compile your matmul.c with –O0 and –O3 optimization flags for gcc, Figure 2 reports execution times in ms for matrix multiplication when compile your matmul.c with –O0 and –O3 optimization flags for gcc, Figure 3 report execution time in ms for matrix vector addition when compile your matmul.c with –O0 and –O3 optimization flags for gcc. I included Figure samples (using dummy values) generated from an Excel sheet for your reference. The excel sheet is provided so you can simply input your execution results and the figure will be automatically populated and generated by Excel. *In your report, you should explain the reasons of the performance differences between different matrix storage types (row major or column major) and the reasons of performance differences between –O0 and –O3 optimization flags (check gcc manual page and search internet to find correct answers, and explain in your English).*
3. While the development can be done from your laptop or any other computers, the results in the report should be collected from the machine listed from <http://cto.secs.oakland.edu/docs/pdf/linuxServers.pdf> , and let me know if you need help to access (you need VPN to access those machine from home, check <http://secs.oakland.edu/docs/pdf/vpn.pdf> ). Please indicate in your report, which machine you use. Please be noted that the machine is shared resource, overloaded use of the machine the last day of the machine may cause incorrect performance results.
4. Explanation of the performance results shown in your figures and draw meaningful conclusions.

**Grading:**

**Functions implementations: 60 points (15 for each of the three functions and main).**

**Report: 40 points.**

**For non-compliable code, you only receive max 60% of function implementations points. For compliable, but with execution errors and incorrectness, you receive max 70% of function implementation points.**

Assignment policy:

Programming assignments are to be done individually. You may discuss assignments with others, but you must code your own solutions and submit them with a write-up in your own words. Indicate clearly the name(s) of student(s) you collaborated with, if any. Although homework assignments will not be pledged, per se, the submitted solutions must be your work and not copied from other students' assignments or other sources.

You may not transmit or receive code from anyone in the class in any way--visually (by showing someone your code), electronically (by emailing, posting, or otherwise sending someone your code), verbally (by reading your code to someone), or in any other way.

You may not collaborate with people who are not your classmates, TAs, or instructor in any way. For example, you may not post questions to programming forums.

You may search the web and use any information that you find. However, you cannot take more than two lines of code from an external resource and actually include it in one of your assignments. Changing variable names or rewriting code you find does not void the "two line rule."

Any violations of these rules will be reported to the honor council. Check the syllabus for the late policy and academic conduct.