Assignment 2 – CSE 436/536, Summer 2016

**Due: 11:55PM June 8th Wednesday, 2016**

**OpenMP Parallelization for MM and Sum**

In this assignment, you will implement and experiments the OpenMP version of matrix multiplication (mm) and sum. All matrices are stored in row major. Please create the two programs in two files. For each program, a sequential version is provided. You will implement the versions that use OpenMP “parallel”, and “parallel for” for decomposing the problem for parallel execution. For mm, you will need to implement row-wise, col-wise and rowcol-wise decomposition mechanisms.

**/\* Sum up all the numbers in array A[N] \*/**

REAL sum (int N, REAL \*A);

REAL sum\_omp\_parallel (int N, REAL \*A, int num\_tasks);

REAL sum\_omp\_parallel\_for (int N, REAL \*A, int num\_tasks);

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

void mm(int N, int K, int M, REAL \* A, REAL \* B, REAL \* C);

void mm\_parallel\_row(int N, int K, int M, REAL \* A, REAL \* B, REAL \* C, int num\_tasks);

void mm\_parallel\_col(int N, int K, int M, REAL \* A, REAL \* B, REAL \* C, int num\_tasks);

void mm\_parallel\_rowcol(int N, int K, int M, REAL \* A, REAL \* B, REAL \* C, int num\_tasks);

void mm\_parallel\_for\_row(int N, int K, int M, REAL \* A, REAL \* B, REAL \* C, int num\_tasks);

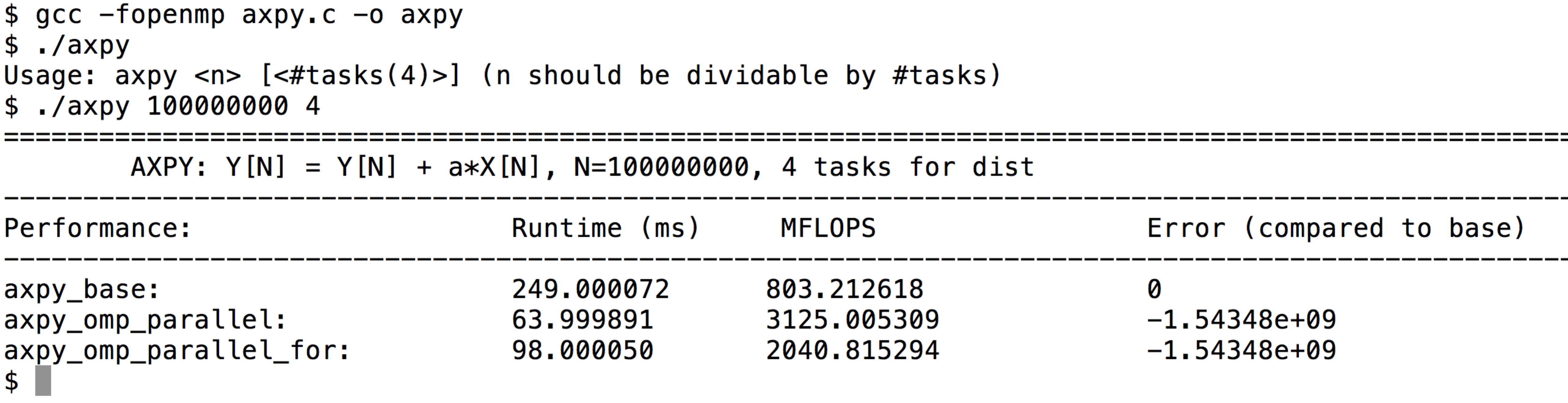
void mm\_parallel\_for\_col(int N, int K, int M, REAL \* A, REAL \* B, REAL \* C, int num\_tasks);

void mm\_parallel\_for\_rowcol(int N, int K, int M, REAL \* A, REAL \* B, REAL \* C, int num\_tasks);

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 are stored in row major. Memory should be dynamically allocated and freed (using malloc and free) for the matrices (A, B, and C) so you can handle large matrices. For sum, you will also experiment different schedule policy for the omp\_parallel\_for version.

For each program (sum.c, and mm.c), you can compile and build the executable using gcc compiler (e.g. gcc -fopenmp mm.c -o mm), in which the -fopenmp flag is to enable OpenMP. Your executable should be able to run with three 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), and num\_tasks for specifying the number of tasks/threads. Default values could be specified in your program for the three arguments.

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



Your output should include all the entries for each functions of the program.

**Submission:** Your submission should include three files: 1) The two source files that contain your implementations 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 5 figures when running with 1024x1024 for mm and 10M (million) for sum, both with 1, 2, 4 and 8 threads. Figure 1 (sum), and 2 (mm) each report the execution times in ms for each version of program using static (default) loop schedule policy. Figure 3 (sum) and 4 (mm) report the speedup for each of the program. Speedup is measured as the ratio of sequential execution time (1 thread) to the parallel execution time (2, 4, 8 threads). Figure 5 report the execution time of sum\_omp\_parallel\_for when running with the following choice of loop schedule policy: static, “static, 2000”, “static, 200”, “static,20”, “dynamic, 2000”, “dynamic, 200”, “dynamic,20”, and “guided, 200000”. Please note that you are welcome to and should tune the policy and chunk size in order to generate any interesting and insightful result. An Excel sheet template is included for you to fill in your data so the figures will be automatically generated. *In your report, you should explain the reasons of the performance differences between different sequential version and the parallel version, and the reasons of performance differences between different parallel versions, and the speedups of each program when increasing the number of tasks(threads).*
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 (30 for each of the two programs).**

**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.