THE UNIVERSITY OF SYDNEY

COMP2129 Assignment 3

Due: 6:00pm Friday, 22 May 2015

This assignment is worth 10% of your final assessment

Task description

In this assignment we will implement a matrix computation engine that is able to construct matrices and perform matrix computations in the C programming language. The aim is to use a variety of parallel programming, algorithmic and code optimisation techniques to achieve peak performance.

You have been provided with some well documented scaffold code. Your task is to implement the incomplete functions and then analyse the running time of the operations under various inputs with an aim to make each operation run as fast as possible. Most of the matrix operations can be improved by using parallelisation, more efficient algorithms and code optimisation techniques. It is up to you to determine whether you will focus on tuning certain operations that are dramatically slower than the others, or to focus on each operation equally. Make sure to cache matrices and results where possible.

You are encouraged to ask questions on ed using the assignments category. As with any assignment, make sure that your work is your own, and you do not share your code or solutions with other students.

Working on your assignment

You can work on this assignment on your own computer or the lab machines. Students can also take advantage of the online code editor on ed. Simply navigate to the assignment page and you are able to run, edit and submit code from within your browser. We recommend that you use Safari or Chrome.

It is important that you continually back up your assignment files onto your own machine, flash drives, external hard drives and cloud storage providers. You are encouraged to submit your assignment while you are in the process of completing it. By submitting you will obtain some feedback of your progress.

Academic declaration

By submitting this assignment you declare the following:

I declare that I have read and understood the University of Sydney Academic Dishonesty and Plagiarism in Coursework Policy, and except where specifically acknowledged, the work contained in this assignment or project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the the Academic Dishonesty and Plagiarism in Coursework Policy, can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Information Technologies, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and or communicate a copy of this assignment to a plagiarism checking service or in house computer program, and that a copy of the assignment may be maintained by the service or the School of IT for the purpose of future plagiarism checking.

Implementation details

Write a program in C that implements Matrix DB based on the included scaffold code. You can assume only valid input will be tested. All matrices will be square and have the same dimensions between $1 \leqslant order \leqslant 10000$. All matrix rows, columns and elements will be indexed from 1. Matrix elements range from 0 up to the maximum value of `uint32_t`. We expect some operations to cause unsigned integer overflows, this behaviour is well defined and those results will wrap around from 0.

Your program must be contained in `matrix.c` and `matrix.h` and produce no errors when built and run on the lab machines and ed. Reading from standard input and writing to standard output.

Your program output must match the exact output format shown in the prototype and on ed. You are encouraged to submit your assignment while you are working on it, so you can obtain some feedback.

The contents of the header file `matrix.h` are shown below

```c
#ifndef MATRIX_H
#define MATRIX_H

#include <stdint.h>

/* utility functions */

uint32_t fast_rand(void);

void set_seed(uint32_t value);
void set_nthreads(ssize_t count);
void set_dimensions(ssize_t width);

void display(const uint32_t* matrix);
void display_row(const uint32_t* matrix, ssize_t row);
void display_column(const uint32_t* matrix, ssize_t column);
void display_element(const uint32_t* matrix, ssize_t row, ssize_t column);

/* matrix operations */

uint32_t* new_matrix(void);
uint32_t* identity_matrix(void);
uint32_t* random_matrix(uint32_t seed);
uint32_t* uniform_matrix(uint32_t value);
uint32_t* sequence_matrix(uint32_t start, uint32_t step);

uint32_t* cloned(const uint32_t* matrix);
uint32_t* sorted(const uint32_t* matrix);
uint32_t* rotated(const uint32_t* matrix);
uint32_t* reversed(const uint32_t* matrix);
uint32_t* transposed(const uint32_t* matrix);

uint32_t* scalar_add(const uint32_t* matrix, uint32_t scalar);
uint32_t* scalar_mul(const uint32_t* matrix, uint32_t scalar);

uint32_t* matrix_add(const uint32_t* matrix_a, const uint32_t* matrix_b);
uint32_t* matrix_mul(const uint32_t* matrix_a, const uint32_t* matrix_b);
uint32_t* matrix_pow(const uint32_t* matrix, uint32_t exponent);

/* compute operations */

uint32_t get_sum(const uint32_t* matrix);
uint32_t get_mode(const uint32_t* matrix);
uint32_t get_trace(const uint32_t* matrix);
uint32_t get_median(const uint32_t* matrix);
uint32_t get_minimum(const uint32_t* matrix);
uint32_t get_maximum(const uint32_t* matrix);
uint32_t get_frequency(const uint32_t* matrix, uint32_t value);

#endif
```

Running the prototype

We have provided you with a prototype implemention of Matrix DB that is accessible from the lab machines and undergraduate servers by running:

```
$ /labcommon/comp2129/bin/matrix
```

We suggest that you experiment with the prototype implementation, and also run and compare your own test cases against it. Your program must match the output of the prototype for all valid input.

Writing your own testcases

We have provided you with some test cases but these do not not test all the functionality described in the assignment. It is important that you thoroughly test your code by writing your own test cases.

You should place all of your test cases in the `tests/` directory. Ensure that each test case has the `.in` input file along with a corresponding `.out` output file. We recommend that the names of your test cases are descriptive so that you know what each is testing, e.g. `sum.in` and `median.in`

Submission details

You must submit your code in the assignment page on ed. To submit, simply place your code into the code editor, click the run button to check your program works and then click the submit button.

You are encouraged to submit multiple times, but only your last submission will be marked.

Marking

In this assignment correctness and performance are equally weighted. However, your submission will only be eligible for the performance component if it passes all of the correctness test cases.

5 **marks** are assigned based on automatic tests for the *correctness* of your program.
This component will use our own hidden test cases that cover every aspect of the specification. To pass test cases your solution must produce the correct output within the imposed time limit.

5 **marks** are assigned based on the *performance* of your code relative to other students.
This component is tested on a separate machine in a consistent manner. The fastest submission will receive 5 marks, with successively slower solutions receiving lower marks. Any student who is faster than our basic parallel reference implementation will receive at least 2.5 marks.

Your program will be marked automatically, so make sure that you carefully follow the assignment specifications. Your program must match the exact output in the examples and the test cases on ed.

Warning: Any attempts to deceive or disrupt the marking system will result in an immediate zero for the entire assignment. Negative marks can be assigned if you do not properly follow the assignment specification, or your code is unnecessarily or deliberately obfuscated.