AMATH 483 / 583 (Roche) - Homework Set 4

Due Friday May 2, 5pm PT

April 25, 2025

Homework 4 (90 points)

1. (+10) Given matrix $A = \begin{pmatrix} 1 & 2 \\ 0 & 2 \end{pmatrix}$, evaluate the action of A on the unit balls of $\mathbb{R}^2$ defined by the 1-norm, 2-norm, and ∞-norm (induced matrix norms). Submit your work and drawings.

2. (+20) Compiler Optimization of Matrix Multiplication Loop Permutations. Implement C++ templated gemm, $C \leftarrow \alpha AB + \beta C$ ($A \in \mathbb{T}^{m\times p}$, $B \in \mathbb{T}^{p\times n}$, $C \in \mathbb{T}^{m\times n}$, $\alpha, \beta \in \mathbb{T}$) for $\{kij\}$ and $\{jki\}$ loop permutations using the specifications provided here:

   -
     ```
     template<typename T>
     void mm_jki(T a, const std::vector<T>& A, const std::vector<T>& B, T b,
     std::vector<T>& C, int m, int p, int n);
     ```

   -
     ```
     template<typename T>
     void mm_kij(T a, const std::vector<T>& A, const std::vector<T>& B, T b,
     std::vector<T>& C, int m, int p, int n);
     ```

   You will explore matrix multiply performance applying compiler optimization levels *-O0* and *-O3* for square matrices of dimension $n = 2$ to $n = 512$, stride one to these functions. For reference, recall the loop order for $C_{i,j} \leftarrow \sum_{k=0}^{p-1} A_{i,k} * B_{k,j} \, \forall [i = 0, ..., m-1][j = 0, ..., n-1]$ is $\{ijk\}$, i outer loop, j middle loop, k inner loop. Let each n be measured *ntrial* times and plot the average performance in FLOPs (flop count / time(seconds)) for each case versus n, $ntrial \geq 3$. Submit plots for both permutation variants that include both FP32 (*float*) and FP64 (*double*) compiler optimization results.

3. (+20) **Row major Matrix class**. Reference the file `matrix_class.hpp` for the starter code for a Matrix class template for row major index referencing and `std::vector<T>` for the matrix storage scheme. Please put your function implementations in file `matrix_class.hpp`, and submit `matrix_class.hpp`.

   - (+5) Matrix transpose for $A \in \mathbb{R}^{m\times n}$ is defined $A^T_{i,j} = A_{j,i}$ and so $A^T \in \mathbb{R}^{n\times m}$. This method returns a matrix as defined by the class.

     -
       ```
       Matrix<T> transpose() const{}
       ```

   - (+5) Matrix infinity norm for $A \in \mathbb{R}^{m\times n}$ is defined $\|A\|_\infty = \max\limits_{0\leq i\leq m} \sum_{j=1}^{j=n} |A_{i,j}|$. This method returns a number.

     -
       ```
       T infinityNorm() const{}
       ```

   - (+5) Write the method to operator overload multiplication $*$ for the matrix class I provided. This method returns a matrix as defined by the class.

     -
       ```
       Matrix<T> operator*(const Matrix<T> &other) const{}
       ```

   - (+5) Write the method to operator overload addition + for the matrix class I provided. This method returns a matrix as defined by the class.

     -
       ```
       template <typename T>
       Matrix<T> Matrix<T>::operator+(const Matrix<T> &other) const{}
       ```

4. (+10) **Extremum**. Consider the surface defined by $xy + 2xz = 5\sqrt{5}$ and $(x, y, z) \in \mathbb{R}^3$. Find (a) the coordinate instance(s) affiliated with the minimum distance from a point on the surface to the origin, and (b) the value of the minimum distance. You will find (may safely assume) the domain $[-3, 3] \times [-3, 3] \times [-3, 3] \in \mathbb{R}^3$ holds the correct coordinate instance(s).

5. (+10) **IO bandwidth**. Write a C++ function that `write`s type `double` square matrices in column major order to file in binary. Measure the time required to complete the `write` for matrices of dimension 32, 64, 128, ... 16384 (2GB). Write a C++ function that `read`s binary matrices from file to type `double` matrices in memory. Measure the time required to complete each `read` for the same dimensions. (a) Make a single plot of the *read* and *write* measurements with the bandwidth (bytes per second) on the y-axis, and the problem dimension on the x-axis. Submit your plot.

6. (+20) **File access time**. (a) Write C++ functions for the given function declarations that perform *row* and *column* swap operations on a type `double` matrix stored in a file in column major index order. Test the swapping capabilities for correctness. Put the functions you write in file **file_swaps.hpp**. (b) Conduct a performance test for square matrix dimensions 16, 32, 64, 128, ... 8192, measuring the time required to conduct file-based *row* and *column* swaps separately. Let each operation be measured *ntrial* times, *ntrial* ≥ 3. Make a single plot of the *row* and *column* swap average times on the y-axis ($log_{10}(time)$) and the problem dimension on the x-axis. Submit your header file **file_swaps.hpp** and plot. You may find the code snippet helpful.

```
void swapRowsInFile(std::fstream &file, int nRows, int nCols, int i, int j);
void swapColsInFile(std::fstream &file, int nRows, int nCols, int i, int j);

// snippet
#include <iostream>
#include <fstream>
#include <vector>
#include <utility>
#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <cstdio>
#include <chrono>
#include "file_swaps.hpp"

int main(int argc, char *argv[])
{
// Generate the matrix
std::vector<double> matrix(numRows * numCols);
// init matrix elements in column major order
// write the matrix to a file
std::fstream file(filename, std::ios::out | std::ios::binary);
file.write(reinterpret_cast<char *>(&matrix[0]), numRows * numCols * sizeof(double));
file.close();
// Open the file in read-write mode for swapping
std::fstream fileToSwap(filename, std::ios::in | std::ios::out | std::ios::binary);
// Get random indices i and j for row swapping
// Measure the time required for row swapping using file I/O
auto startTime = std::chrono::high_resolution_clock::now();
// Swap rows i and j in the file version of the matrix
swapRowsInFile(fileToSwap, numRows, numCols, i, j);
auto endTime = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> duration = endTime - startTime;
// Close the file after swapping
fileToSwap.close();
//...
// after each problem size delete the test file
std::remove(filename.c_str());
// ...
}
```