AMATH 483 / 583 - HW3

1 Problems

1. Matrix Multiplication Loop Permutations. Implement templated gemm for each $\{i, j, k\}$ loop permutation using the following specifications. Each computes $C \leftarrow \alpha AB + \beta C$, where $A \in \mathbb{R}^{m\times p}$, $B \in \mathbb{R}^{p\times n}$, $C \in \mathbb{R}^{m\times n}$, $\alpha, \beta \in \mathbb{R}$, but will exhibit distinct memory access patterns. Check these produce the correct results. Turn in the `.cpp` and `.hpp` files for each. Include the header files into another header file `hw3_p1_header.hpp` and submit this as well. Pay special attention that your matrices will now be represented within a single vector in this exercise. Please utilize column major ordering as discussed in lecture when assigning and accessing matrix elements in this format.

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

2. Compiler Optimization. Use the $\{kij\}$ and $\{jki\}$ loop permutation codes from problem 1 to explore the performance of your implementations applying compiler optimization levels *default* (no optimization or default case), *-O3*, and *-ffast-math* (or the equivalent for your compiler!) for square matrices of dimension $n = 2$ to $n = 512$, stride one. Let each n be measured *ntrial* times and plot the average performance for each case versus n, *ntrial* ≥ 3. Submit your `.cpp` test code, and two plots -one for each loop variant on your choice of data type *float* or *double*. **Extra credit:** Submit plots for both data types.

3. **(AM583 only, +5 for AM483)** Strassen. Use notes from the class lecture to implement a C++ template for the (recursive) Strassen matrix multiplication algorithm. Plot the *double* precision performance for square matrices of even dimension from $n = 2$ to $n = 512$. Let each n be measured *ntrial* times and plot the average performance versus n, *ntrial* ≥ 3. You will turn in the `.cpp` and `.hpp` files for the strassen code, your `.cpp` test code, and performance plot.

   ```
   template <typename T>
   vector<vector<T>> strassen_mm(const vector<vector<T>> &A,
   const vector<vector<T>> &B); //vector<vector<double>>C=strassen_mm(A, B);
   ```