# ICS, Fall 2016
# Lab 4: Code Optimization

November 23, 2016

## 1 Introduction

In this lab, you will have a chance to apply the various code optimization techniques that you have learned in class. You will be given a naive implementation of matrix multiplication in C. On the premises of maintaining the correctness of the original program, you are required to optimize it with your maximum effort. On completion of this lab, you will gain insights on valuable tricks that accelerate programs.

## 2 Background

In mathematics, matrix multiplication is defined as a binary operation that produces a matrix from two matrices. Suppose that $A$ is an $n \times m$ matrix and $B$ is an $m \times p$ matrix, their matrix product $AB$ is an $n \times p$ matrix, in which the $m$ entries across a row of $A$ are multiplied with the $m$ entries down a columns of $B$ and summed to produce an entry of $AB$. In more detail,

$$A = \begin{pmatrix} A_{11} & A_{12} & ... & A_{1m} \\ A_{21} & A_{22} & ... & A_{2m} \\ ... & ... & ... & ... \\ A_{n1} & A_{n2} & ... & A_{nm} \end{pmatrix} \quad B = \begin{pmatrix} B_{11} & B_{12} & ... & B_{1p} \\ B_{21} & B_{22} & ... & B_{2p} \\ ... & ... & ... & ... \\ B_{m1} & B_{m2} & ... & B_{mp} \end{pmatrix}$$

$$AB = \begin{pmatrix} AB_{11} & AB_{12} & ... & AB_{1p} \\ AB_{21} & AB_{22} & ... & AB_{2p} \\ ... & ... & ... & ... \\ AB_{n1} & AB_{n2} & ... & AB_{np} \end{pmatrix}$$

each $i$, $j$ entry of $AB$ is given by multiplying the entries $A_{ik}$ (across row $i$ of $A$) by the entries $B_{kj}$ (down column $j$ of $B$), for $k = 1, 2, ..., m$, and summing the results over k:

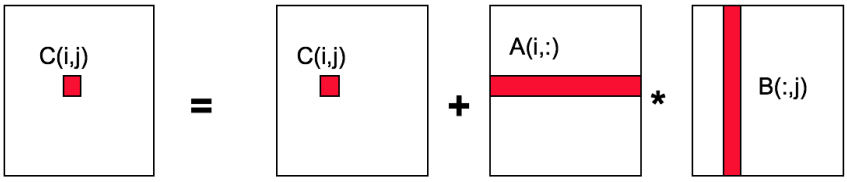
$$(AB)_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj}$$

# 3 Implementation

A *naive* implementation of matrix multiplication in C is given below.

```
typedef double data_t;

void MM(size_t m, size_t n, size_t h,
            data_t *C, data_t *A, data_t *B) {
    size_t i, j k;
    for (i = 0; i < m; ++i)
        for (size_t j = 0; j < n; ++j)
            for (size_t k = 0; k < h; ++k)
                C[i * n + j] += A[i * h + k] * B[k * n + j];
}
```




You are required to provide your own implementation of matrix multiplication in C and make it run as fast as possible. You are free to use any optimization technique introduced in class, including moving *vec_length* out of loop, avoiding bounds check on each cycle, accumulating in temporary, loop unrolling, multiple accumulators, SIMD operations, etc.

# 4 Evaluation

The primary objective in writing a program is to ensure that it works correctly under all possible conditions. Code optimization that does not preserve the correctness of the original program serves no useful purpose.

As a sanity check of your code, you must ensure that the optimized program yields exactly the same result as the *naive* implementation of matrix multiplication. Recall that in the $4th$ lecture we have learned that, while floating-point addition and multiplication are both commutative ($a+b = b+a$ and $a \times b = b \times a$), they are not necessarily associative (i.e., $(a + b) + c$ is not necessarily equal to $a + (b + c)$), as rounding errors are introduced when dissimilar-sized values are joined together. Therefore each element in the result matrix $C$ is allowed to have an error threshold of $1e-3$. That is to say, $-1e-3 < C[i,j] - C'[i,j] < 1e-3$ holds true for all possible conditions, in which $C[i,j]$ and $C'[i,j]$ stand for the $i$, $j$ entry of the result matrix given by the *naive* implementation and your optimized implementation, respectively. Submissions that failed to pass the

sanity check will be graded as incorrect, and not be eligible for the performance evaluations. *Do be careful.*

The performance of your submitted code is evaluated in terms of *cycles per element*, which is abbreviated as *CPE*. As described in the lecture, this measure assumes that the run time, measured in clock cycles, for an array of length $n$ is a function of the form $T = CPE \times n + Overhead$. You may refer to the $10th$ lecture for details.

Apart from the ultimate code, you are also required to submit a technical report which includes all the optimization techniques that you have applied in your program. You should demonstrate in your report how much extra performance will be gained through the adoption of each trick(in terms of the *CPE* reduction in percentile). In addition, for techniques like loop unrolling, you should specify how you could find the optimal parameter for the optimization. It would be better if you could demonstrate it with a table *Hint: you may refer to page 43 of the slides.*

This assignment is graded both on the performance of your code and the technical report that you submitted, and the ratio is 7 : 3. Student whose program wins the first place will get full credit for the performance part of this assignment, and the rest will receive normalized score according to their ranks.

# 5 Hand-In Instructions

The *deadline* for this assignment is November 30, 2016. If you have any question about this assignment, feel free to contact your TAs.

After you have completed this assignment, please pack your source code and your technical report into a single file and send it to $luyc13@fudan.edu.cn$.