Matrix Multiplication Optimization

Hao Deng

October 2023

1 Introduction

In this assignment, we want compare the following matrix multiplication optimization methods:

1. Hoisting
2. Unroll and Jam
3. Tiling and Hoisting

We use direct multiplication as our benchmark.
We compare the performance of different optimizations under the following compilation optimization options:

1. O0
2. O2

Finally, we plot the performance for each optimization in terms of runtime and GFLOPS.

2 Experiment

First, we initialize three matrices (A, B, and C) with sizes $N \times N$, where $N = 2^a (a \in [1, 11], a \in \mathbb{N})$. Then, we perform the following multiplication using different optimization methods:

$$C = A * B$$

We record the performance and make plots.

We obtained the following result:

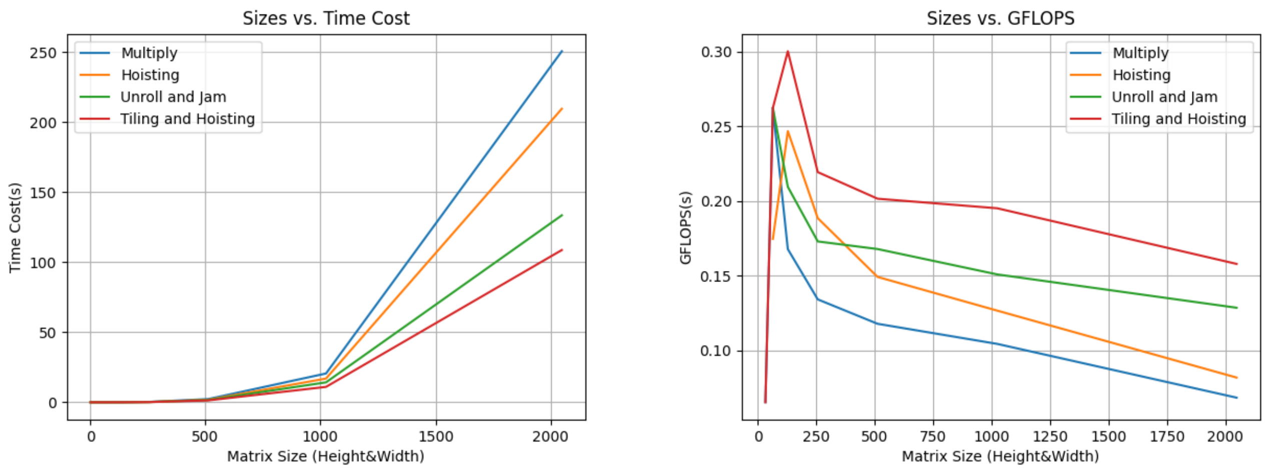

Figure 1: Compilation Optimization option: -O0

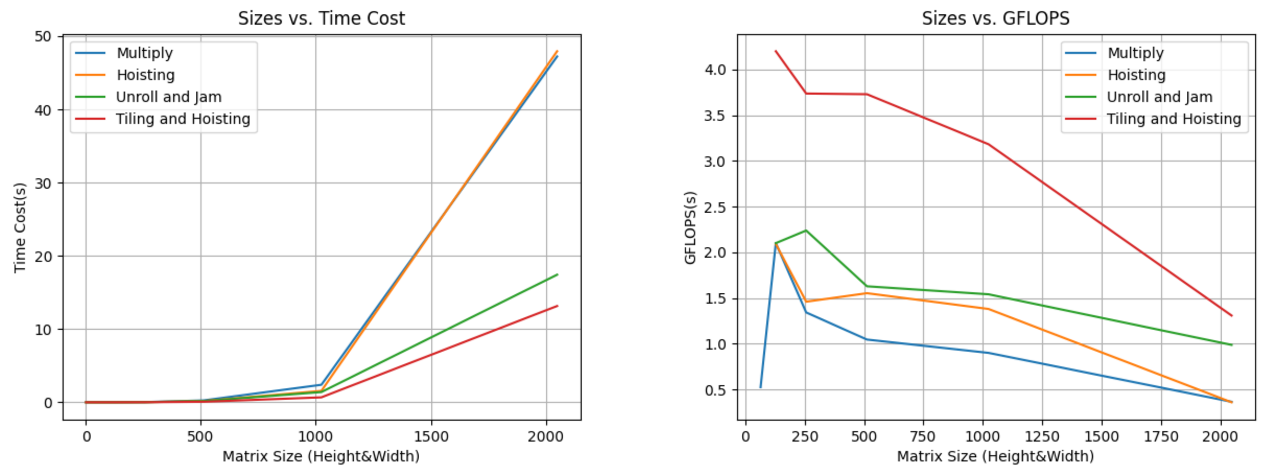

Figure 2: Compilation Optimization option: -O2

We observed that under -O0, Tiling and Hoisting has the best performance, followed by Unroll and Jam, Hoisting, and direct multiplication.

Under -O2, Tiling and Hoisting still has the best performance. Unroll and Jam is the second best. However, there is little difference between Hoisting and direction multiplication, especially in terms of runtime.

Notice that the performances for all four methods get improved under -O2.

Interestingly, we can notice that there is a significant performance drop from $N = 128$ to $N = 256$.
For $N = 128$, a matrix consists of $128 \times 128 = 16,384$ doubles, and hence 131,072 bytes.
For $N = 256$, a matrix consists of $256 \times 256 = 65,536$ doubles, and hence 524,288 bytes.
Our machine has 384KB (or 384,000 bytes) L1 cache.
We speculate the reason for the performance drop is that the data size exceeds the capacity of L1 cache at $N = 256$.
Hence, CPU had to load data from L2 cache or even farther, which led to drop in performance.

3 Conclusion

Under both optimization options, Tiling and Hoisting has the best performance. Compared to not using optimization (-O0), using compilation optimization (-O2) can improve the performance for all four methods of multiplication.