Raymond Zhu

Project 2 Report

Experimental Results:

To begin this experiment, I started with 1,000x1,000 matrix multiplication at both 1% density and 0.1% density as shown in the first two images below. I performed multiple tests for comparison and came to the conclusion that there was not much of a difference (around 0.02sec to 0.06sec). Next, I conducted the 10,000x10,000 matrix multiplication at both 1% and 0.1% density. As one can see below in the next two figures, both these tests took over 3.5 hours (13000 seconds). Since there was no way I was willing to perform these two tests more than once, it seems that adjusting the density of the matrix does not affect the time and speed of performance as much as the overall sizes of the matrices.

1000 x 1000; 1% density:

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 1000 density .01
Size: 1000x1000
Density: 1% x 1%
Threads: 1

Matrix multiplication completed in 8.64022 seconds.
```

1000 x 1000; 0.1% density:

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 1000 density .001
Size: 1000x1000
Density: 0.1% x 0.1%
Threads: 1

Matrix multiplication completed in 8.68634 seconds.
```

10000 x 10000; 1% density:

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 10000 density .01
Size: 10000x10000
Density: 1% x 1%
Threads: 1

Matrix multiplication completed in 13001.2 seconds.
```

10000 x 10000; 0.1% density:

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 10000 density .001
Size: 10000x10000
Density: 0.1% x 0.1%
Threads: 1

Matrix multiplication completed in 13136.6 seconds.
```

Moving on to the next part of the experiment, I started by finding 2 baselines and the control times for each of them. I decided to use 5000x5000 sized matrices, 10% density, and 1 thread as one arbitrary control and 2500x2500, 50%, and 1 thread for the other.

Baseline:

5000 x 5000; 10% density; 1 thread

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 5000
Size: 5000x5000
Density: 10% x 10%
Threads: 1

Matrix multiplication completed in 1550.15 seconds.
```

2500 x 2500; 50% density; 1 thread

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 2500 density 0.5
Size: 2500x2500
Density: 50% x 50%
Threads: 1

Matrix multiplication completed in 185.38 seconds.
```

Multithreading:

The first optimization technique that I tested was multithreading. As seen below in the next 3 images, I performed the test for the 5000x5000 size matrices with 4 threads, 8 threads, and 12 threads. My laptop could access up to 12 threads so I decided to test in increments of 4. As seen below, with just 4 threads, the time was cut from around 26 mins to around 7 mins. With 8 threads, the time was shortened again to around 5 mins. Finally with 12 threads, the time was shortened again to around 4 mins.

5000 x 5000; 10% density; 4 threads

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 5000 threads 4
Size: 5000x5000
Density: 10% x 10%
Threads: 4

Matrix multiplication completed in 440.308 seconds.
```

5000 x 5000; 10% density; 8 threads

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 5000 threads 8
Size: 5000x5000
Density: 10% x 10%
Threads: 8

Matrix multiplication completed in 299.469 seconds.
```

5000 x 5000; 10% density; 12 threads

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 5000 threads 12
Size: 5000x5000
Density: 10% x 10%
Threads: 12

Matrix multiplication completed in 252.874 seconds.
```

I then tested on the second baseline of 2500x2500. As seen below, with 4 threads, the time was cut down to around 52 seconds. With 8 threads, the time went down to around 33 seconds. With 12 threads, the time went down to around 28 seconds.

2500 x 2500; 50% density; 4 threads

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 2500 density 0.5 threads 4
Size: 2500x2500
Density: 50% x 50%
Threads: 4

Matrix multiplication completed in 52.054 seconds.
```

2500 x 2500; 50% density; 8 threads

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 2500 density 0.5 threads 8
Size: 2500x2500
Density: 50% x 50%
Threads: 8

Matrix multiplication completed in 32.9745 seconds.
```

2500 x 2500; 50% density; 12 threads

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 2500 density 0.5 threads 12
Size: 2500x2500
Density: 50% x 50%
Threads: 12

Matrix multiplication completed in 28.0835 seconds.
```

The second optimization technique I tested for was SIMD. As seen in the figure below, the time for the 5000x5000 compared to the baseline was shortened to around 6-7 mins. For the 2500x2500, time was shortened to around 54 seconds.

SMID:

5000 x 5000; 10% density

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 5000 simd
Size: 5000x5000
Density: 10% x 10%
Threads: 1

Matrix multiplication completed in 393.846 seconds.
```

2500 x 2500; 50% density

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 2500 density 0.5 simd
Size: 2500x2500
Density: 50% x 50%
Threads: 1

Matrix multiplication completed in 54.1921 seconds.
```

The third optimization technique I tested for was cache miss optimization. As seen below, the time for the 5000x5000 was only shortened to around 17 mins. The time for the 2500x2500 was shortened to around 2 mins.

Cache:

5000 x 5000; 10% density

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 5000 cache
Size: 5000x5000
Density: 10% x 10%
Threads: 1

Matrix multiplication completed in 1045.78 seconds.
```

2500 x 2500; 50% density

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 2500 density 0.5 cache
Size: 2500x2500
Density: 50% x 50%
Threads: 1

Matrix multiplication completed in 127.969 seconds.
```

Finally, I performed all 3 optimization techniques together for each of the baselines. For the 5000x5000, the time ended up around 33 seconds. For the 2500x2500, the time ended up around 4 seconds.

All Three Together:

5000 x 5000; 10% density

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 5000 all 12
Size: 5000x5000
Density: 10% x 10%
Threads: 12

Matrix multiplication completed in 33.2116 seconds.
```

2500 x 2500; 50% density

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 2500 density 0.5 all 12
Size: 2500x2500
Density: 50% x 50%
Threads: 12

Matrix multiplication completed in 3.97889 seconds.
```

Comparing back to both baselines, it can be seen that the optimization that had the most impact was multithreading with 12 threads. Multithreading with 12 threads was able to cut the 5000x5000 matrix size time by roughly 22 mins (85%) and the 2500x2500 matrix size time by roughly 2 mins (66%). The time was lowered each time with the addition of more and more threads, but the amount of time that was saved each time was also diminished.

Dense (100%) x Dense (100%):

For dense vs dense, I chose to test 2500x2500, 3250x3250, and 5000x5000 at 100% density and with all 3 optimization techniques to speed things up. As the matrix size increased, the time needed also increased.

2500 x 2500; 100% density; 12 threads; Everything

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 2500 density_A 1 density_B 1 all 12
Size: 2500x2500
Density: 100% x 100%
Threads: 12

Matrix multiplication completed in 3.97529 seconds.
```

3250 x 3250; 100% density; 12 threads; Everything

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 3250 density 1 all 12
Size: 3250x3250
Density: 100% x 100%
Threads: 12

Matrix multiplication completed in 9.07588 seconds.
```

5000 x 5000; 100% density; 12 threads; Everything

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 5000 density_A 1 density_B 1 all 12
Size: 5000x5000
Density: 100% x 100%
Threads: 12

Matrix multiplication completed in 33.5683 seconds.
```

Dense (100%) x Sparse (1%):

For the first part for dense vs sparse, I stuck with 2500x2500, but changed the densities of each matrix. I started at 95% and 5%, went to 90% and 10%, and ended at 85% and 15%. As you can see below, the time slightly went up with each test, increasing at about 0.5 seconds.

2500 x 2500; 95% x 5% density; 12 threads; Everything

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 2500 density_A 0.95 density_B 0.05 all 12
Size: 2500x2500
Density: 95% x 5%
Threads: 12

Matrix multiplication completed in 3.99376 seconds.
```

2500 x 2500; 90% x 10% density; 12 threads; Everything

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 2500 density_A 0.90 density_B 0.10 all 12
Size: 2500x2500
Density: 90% x 10%
Threads: 12

Matrix multiplication completed in 4.05507 seconds.
```

2500 x 2500; 85% x 15% density; 12 threads; Everything

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 2500 density_A 0.85 density_B 0.15 all 12
Size: 2500x2500
Density: 85% x 15%
Threads: 12

Matrix multiplication completed in 4.10604 seconds.
```

The second part for dense vs sparse was to change the matrix size. Sticking with the same sizes as the dense vs dense tests, I conducted tests for 2500x2500, 3250x3250, and 5000x5000 for 100% x 1% density. As seen below, the times for dense vs sparse increased as the matrix size increased. Not only were the times around the same as the dense vs dense test, but the times also increased at around the same intervals.

2500 x 2500; 100% x 1% density; 12 threads; Everything

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 2500 density_A 1 density_B 0.01 all 12
Size: 2500x2500
Density: 100% x 1%
Threads: 12

Matrix multiplication completed in 4.09871 seconds.
```

3250 x 3250; 100% x 1% density; 12 threads; Everything

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 3250 density_A 1 density_B 0.01 all 12
Size: 3250x3250
Density: 100% x 1%
Threads: 12

Matrix multiplication completed in 9.03613 seconds.
```

5000 x 5000; 100% x 1% density; 12 threads; Everything

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 5000 density_A 1 density_B 0.01 all 12
Size: 5000x5000
Density: 100% x 1%
Threads: 12

Matrix multiplication completed in 33.6797 seconds.
```

Sparse (1%) x Sparse (1%):

The experiment for sparse vs sparse was also split into the same two parts as dense vs sparse. For part 1, I stuck with 2500x2500 and decreased the density of each test. I started at 0.1% x 0.1%, went to 0.01% x 0.01%, and ended at 0.001% x 0.001%. As I performed these tests multiple times, I noticed that there was not much difference or correlation as the density decreased. All times were basically at around 4 seconds.

2500 x 2500; 0.1% x 0.1% density; 12 threads; Everything

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 2500 density_A 0.001 density_B 0.001 all 12
Size: 2500x2500
Density: 0.1% x 0.1%
Threads: 12

Matrix multiplication completed in 4.24131 seconds.
```

2500 x 2500; 0.01% x 0.01% density; 12 threads; Everything

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 2500 density_A 0.0001 density_B 0.0001 all 12
Size: 2500x2500
Density: 0.01% x 0.01%
Threads: 12

Matrix multiplication completed in 4.05678 seconds.
```

2500 x 2500; 0.001% x 0.001% density; 12 threads; Everything

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 2500 density_A 0.00001 density_B 0.00001 all 12
Size: 2500x2500
Density: 0.001% x 0.001%
Threads: 12

Matrix multiplication completed in 4.11831 seconds.
```

For the second part of sparse vs sparse, I stuck with 2500x2500, 3250x3250, and 5000x5000 matrix sizes and kept the densities at 1% x 1%. As the matrix size increased, the time increased. Here you can see that like the dense vs dense and dense vs sparse tests, the times and time increase intervals were around the same again.

2500 x 2500; 1% x 1% density; 12 threads; Everything

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 2500 density_A 0.01 density_B 0.01 all 12
Size: 2500x2500
Density: 1% x 1%
Threads: 12

Matrix multiplication completed in 3.95473 seconds.
```

3250 x 3250; 1% x 1% density; 12 threads; Everything

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 3250 density_A 0.01 density_B 0.01 all 12
Size: 3250x3250
Density: 1% x 1%
Threads: 12

Matrix multiplication completed in 9.25339 seconds.
```

5000 x 5000; 1% x 1% density; 12 threads; Everything

```
C:\Users\Ray Zhu\Code>Project_02_test_02 size 5000 density_A 0.01 density_B 0.01 all 12
Size: 5000x5000
Density: 1% x 1%
Threads: 12

Matrix multiplication completed in 33.77 seconds.
```

Final Analysis and Conclusion:

Project 2 involved matrix-matrix multiplication, where various optimization techniques of multithreading, SIMD (Single Instruction, Multiple Data), and cache miss minimization, were applied. The experimentation covered matrix sizes ranging from 1000x1000 to 10000x10000 with different levels of matrix sparsities, and focused on how these optimizations affected performance.

Key Findings:

1. **Multithreading:**
    - Multithreading was the most impactful optimization technique. By increasing the number of threads, the computational time was drastically reduced, especially for larger matrices. For example, with 12 threads, the time for the 5000x5000 matrix was cut down by 85% compared to the single-thread baseline, from 26 minutes to about 4 minutes. Similarly, the 2500x2500 matrix saw a 66% reduction in computational time when increasing the thread count.
2. **SIMD Optimization:**
    - SIMD also contributed significantly to performance improvement. While not as effective as multithreading, SIMD reduced the time for the 5000x5000 matrix to around 6-7 minutes from the baseline time of 26 minutes. This technique was particularly useful when working with matrices of medium sizes and densities.
3. **Cache Miss Minimization:**
    - Cache optimization had the least impact among the three. It reduced the time by a smaller margin compared to the other techniques. For instance, the 5000x5000 matrix time was reduced from 26 minutes to around 17 minutes, highlighting that cache optimization primarily enhances performance when other bottlenecks, such as computational power, have already been addressed.
4. **Dense-Dense Multiplication:**
    - For dense matrices, the overall computation time increased linearly with matrix size, even with optimizations applied. The experiments on matrix sizes from 2500x2500 to 5000x5000 at 100% density showed consistent performance improvement with the combined optimization techniques but with diminishing returns as matrix size increased.
5. **Dense-Sparse Multiplication:**
    - The time increase in dense-sparse multiplication was marginal compared to dense-dense multiplication, which indicates that the sparsity of one matrix helps in reducing computational complexity. The sparsity configurations (95%-5%, 90%-10%, and 85%-15%) only resulted in slight performance degradation, as the matrix sparsity increased.
6. **Sparse-Sparse Multiplication:**
    - Sparse-sparse multiplication exhibited almost no performance improvement as sparsity decreased (from 0.1% to 0.001%). This suggests that at extremely low sparsity levels, the multiplication operation is inherently fast, and additional optimizations have limited effect.

Conclusion:

The results from Project 2 confirmed that multithreading was the most effective optimization for large matrix multiplications.The experiments demonstrate a significant decrease in computation time for both dense and sparse matrices, with multithreading showing the most pronounced benefits. However, the efficiency of these optimizations diminishes with extremely sparse matrices, where the computational load is already minimal. This suggests that careful consideration of matrix size and density is essential when determining which optimizations to apply for maximum performance improvement.

In practical terms, for large-scale applications involving matrix-matrix multiplication—such as those in machine learning or scientific computing—multithreading should be prioritized, followed by SIMD and cache optimizations as supplementary techniques. As matrices become larger and more computationally expensive, the benefits of multithreading and SIMD scale well, ensuring faster execution times and better system resource utilization.