Investigating Parallel and Vectorized Matrix Multiplication Techniques

Gabriel Felipe Bernal Pinto

December 1, 2024

Abstract

Matrix multiplication is a fundamental operation in many computational fields, yet it is computationally expensive for large matrices. This study investigates the performance improvements achieved through parallel computing and vectorization techniques. Using a basic algorithm as a baseline, parallel and vectorized implementations were tested on a high-performance PC (Intel(R) Core i7-7700K CPU @ 4.20 GHz, 16 GB of RAM). Results show significant speedup and efficient resource utilization, highlighting the potential of these techniques for scalable and high-speed computations. The findings provide a foundation for further exploration of optimization strategies in computational linear algebra.

1 Introduction

Matrix multiplication is widely used in applications such as scientific computing, graphics, and machine learning. Due to its computational complexity, optimizing matrix multiplication algorithms is crucial for improving performance. This paper explores parallel and vectorized matrix multiplication techniques to achieve faster execution times while maintaining accuracy.

The focus of this study is to compare the performance of three approaches: a basic implementation, a parallelized version leveraging multi-threading, and a vectorized implementation using Java's parallel utilities. Metrics such as execution time, memory usage, speedup, and efficiency were analyzed.

2 Methodology

The experiments were conducted on a PC with the following specifications:

- Processor: Intel(R) Core i7-7700K CPU @ 4.20 GHz
- Memory: 16 GB of RAM

Three implementations of matrix multiplication were tested:

1. **Basic Algorithm**: A straightforward triple-nested loop implementation.
2. **Parallel Algorithm**: A multi-threaded approach using Java's ExecutorService.

3. **Vectorized Algorithm**: A technique employing Java's parallel stream utilities to optimize computations.

The tests were performed on randomly generated 1024×1024 matrices. Metrics such as execution time, memory usage, speedup, and efficiency were measured and analyzed.

3 Experiments and Results

3.1 Memory Usage

The memory consumption for each approach was measured:

- Basic Algorithm: 3 MB
- Parallel Algorithm: 12 MB
- Vectorized Algorithm: 14 MB

3.2 Execution Time

Execution times (in milliseconds) were recorded as follows:

- Basic Algorithm: 8878.52 ms
- Parallel Algorithm: 3162.62 ms
- Vectorized Algorithm: 2712.79 ms

3.3 Speedup and Efficiency

- Parallel Speedup: 2.81
- Parallel Efficiency: 70.18%
- Vectorized Speedup: 3.27
- Vectorized Efficiency: 78.96%

The results demonstrate that both parallel and vectorized implementations significantly outperform the basic algorithm. The vectorized algorithm achieved the highest speedup, emphasizing the efficiency of modern parallel processing utilities.

4 Conclusion

This study successfully demonstrated the performance benefits of parallel and vectorized matrix multiplication techniques. By utilizing multi-threading and vectorization, execution time was reduced by up to 69%, with acceptable memory overhead. These findings highlight the potential of parallel and vectorized methods for large-scale matrix computations.

5 Future Work

Future research can focus on:

- Exploring additional vectorization techniques using SIMD or GPU acceleration.
- Optimizing resource utilization in multi-threaded implementations.
- Extending these methods to sparse matrices or other matrix-related operations.

References

You can find the source code for this project on GitHub:

- Parallel (and Vectorized) Matrix Multiplication.