Optimizing Matrix Multiplication Through Parallelization

Carlos Santana Esplá

December 2, 2023

Abstract

This paper addresses the efficient multiplication of matrices using a parallelized approach. The experiment involves benchmarking the performance of matrix multiplication, examining the impact of utilizing multi-threading. The implemented solution leverages Java, employing parallel threads to expedite the computation. The primary objective is to reduce execution time significantly.

1 Introduction

Matrix multiplication is a fundamental operation in various computational tasks, serving as the backbone for numerous scientific and engineering applications. As the sizes of matrices grow, the computational complexity increases, demanding efficient algorithms to ensure timely results. This paper delves into the optimization of matrix multiplication through the application of parallelization techniques.

1.1 Background and Motivation

Benchmarking programming languages for computational tasks is a common practice to identify the most suitable environment for specific applications. In the context of matrix multiplication, Java's multi-threading capabilities make it a compelling candidate. The motivation behind this research is to explore how parallelization can be leveraged to enhance the performance of matrix multiplication in Java.

1.2 Literature Review

Previous approaches to optimizing matrix multiplication have primarily focused on algorithmic improvements. While these methods have shown success, there is a growing interest in exploiting hardware parallelism to achieve even greater efficiency. This paper contributes to this evolving field by introducing a novel method that combines algorithmic optimization with parallelization.

1.3 Research Contribution

The main contribution of this paper is the introduction of a parallelized matrix multiplication method implemented in Java. The method utilizes multi-threading to concurrently process matrix elements, aiming to significantly reduce execution time. By combining algorithmic improvements with parallelization, this research contributes to the ongoing efforts to optimize matrix multiplication.

2 Problem Statement

Efficient matrix multiplication is essential for a wide range of applications, including scientific simulations, machine learning, and numerical analysis. Traditional sequential methods may not fully exploit the computational capabilities of modern hardware. The problem addressed in this paper is the need for an optimized approach that leverages parallelization to enhance the efficiency of matrix multiplication.

3 Solution/Methodology

The proposed solution involves the utilization of multi-threading to parallelize the matrix multiplication process. The `MatrixMultiplierThread` class divides the matrix into partitions, and each partition is processed concurrently by separate threads. This approach aims to capitalize on the parallel processing capabilities of modern multi-core processors.

To ensure the reproducibility of the experiments, the methodology includes a detailed description of the experimental setup. The matrices used for multiplication are read from external files using the `Reader` class. The `System.currentTimeMillis()` method is employed to measure the execution time accurately.

4 Experiments

A comprehensive set of experiments was conducted to evaluate the performance of the proposed parallelized matrix multiplication method. Matrices of varying sizes were used, sourced from `"./rdb200/rdb200.mtx"` and `"./Trefethen_200/Trefethen_200.mtx"`. These matrices were read from files using the `readMatrixFromFile` method of the `Reader` class.

The execution time of the matrix multiplication process was measured using the `System.currentTimeMillis()` method. The experiments were repeated with different matrix sizes to analyze the scalability of the parallelized approach. The obtained results were meticulously recorded, and charts and tables were generated to visualize the impact of multi-threading on execution time.

4.1 Experimental Setup

The experimental setup involved a machine with specific configurations detailed in the `Reader` class. The matrices were read, and the multiplication process was executed in a controlled environment to ensure the reliability of the results.

5 Code Integration

The parallelized matrix multiplication method is seamlessly integrated into the code, enhancing the efficiency of the matrix multiplication process. The `MatrixMultiplier` class orchestrates parallel execution by dividing the matrix into partitions and distributing them among multiple threads. Each thread, represented by the `MatrixMultiplierThread` class, processes a distinct partition concurrently, harnessing the power of multi-core processors.

The integration ensures a modular and maintainable implementation, allowing for easy scalability and adaptation to varying matrix sizes.

5.1 Results and Analysis

The results indicated a substantial reduction in execution time when utilizing multi-threading for matrix multiplication. The impact was more pronounced with larger matrices, demonstrating the scalability of the proposed approach. Charts and tables presented a clear visual representation of the efficiency gains achieved through parallelization.

6 Conclusion

This research presents a novel approach to optimizing matrix multiplication through parallelization in Java. The integration of multi-threading into the matrix multiplication process showcases substantial improvements in execution time. The proposed method contributes to the ongoing efforts in the field of matrix computation optimization.

7 Future Work

Future research could explore alternative implementations of matrix multiplication, investigating different algorithms and optimizations. Additionally, further experimentation on diverse hardware configurations and matrix sizes could provide insights into the scalability of the proposed method. Exploring the application of the method to different types of matrices and computational tasks would contribute to a more comprehensive understanding of its potential.

In conclusion, this paper highlights the effectiveness of utilizing execution time as a metric for optimizing matrix multiplication. The incorporation of multi-threading yields substantial improvements, showcasing the potential for enhancing performance in computational tasks involving matrix operations.