Comparative Study of code optimization and multithreading in Java, Go and Python

Abhishek Deshmukh

Spring 2023 Adv Prog Language Principles — Under the guidance of Prof. Thomas Austin

Abstract

As software development requirements have become increasingly more complex, using computational resources efficiently has become a critical aspect in the development cycle. The primary method to improve and optimize computation is through parallel processing which allows execution of multiple processes simultaneously. Nonetheless, the execution time of a program does not solely rely on algorithmic logic but also depends on the multi-threading capabilities of the programming language.

Contents

1 Introduction

In today's fast-paced computing environment, software developers are responsible for optimizing the performance of their applications. Parallel processing is one of the most commonly used optimization techniques, and multithreading is a popular execution model for achieving this. However, the performance of a multithreaded application is not solely determined by the implementation logic. Choosing the right programming language is also critical for achieving optimal performance. In this context, we propose a comparative analysis of the matrix multiplication algorithm implemented using three different programming languages: Java, Go, and Python. The aim of this analysis is to evaluate the performance of each language in terms of the time taken and CPU utilization during end-to-end execution. The results of this comparison will provide valuable insights into the suitability of each language for multithreaded applications, thus aiding software developers in selecting the best language for their specific optimization needs.

1.1 Language Selection

The study aims to evaluate the multithreading performance of three different programming languages: Java, Go, and Python. Java was chosen for its capability to provide developers with the ability to manage thread execution through interface implementations, while Go simplifies multithreading by abstracting its complexity from users. Python, on the other hand, was selected due to its popularity and straightforward coding syntax, making it an accessible choice for beginners and a commonly used language for multithreaded applications. The goal is to compare the three languages' multithreading capabilities and provide insights into their respective strengths and weaknesses.

2 Programming Model

2.1 Matrix Multiplication

Matrix multiplication is a critical operation in the field of linear algebra, involving the multiplication of two matrices to produce a third matrix. This operation is computationally demanding and requires significant computational resources. Despite this, matrix multiplication has a broad range of applications across various fields, including population modeling, coordinate system transformation, network theory, and many others.

In population modeling, matrix multiplication can be used to simulate disease spread and estimate population growth. In coordinate system transformation, matrix multiplication can convert data from one coordinate system to another, enabling interoperability between different systems. In network theory, matrix multiplication is used to model network connections and analyze network behavior, making it a valuable tool in network analysis.

As matrix multiplication is such a computationally intensive operation, it is essential to optimize its performance through parallel processing. However, the choice of programming language can significantly impact the efficiency of parallel processing. By comparing the implementation of matrix multiplication in different programming languages, we can gain valuable insights into the strengths and weaknesses of each language and make informed decisions in optimizing matrix multiplication for our applications.

2.2 Implementation

Matrix multiplication is a significant mathematical operation that is utilized in various mathematical applications, especially in linear algebra. To optimize the process of computation, multithreading is used. Each row of matrix A is assigned a new thread to compute the multiplication process in parallel with other threads. The use of multithreading accelerates the computation of matrix multiplication between matrices A and B. The resulting matrix C is calculated by adding up the products of corresponding elements of matrices A and B. By utilizing multithreading, the algorithm reduces the computation time and maximizes the utilization of computing resources.

```
matrixMultiply()
    for i from 0 to n
        spawn new thread
        parallelMultiply(A[i,n], B[n,n])

parallelMultiply(A[1,n], B[n,n])
    for i from 0 to n
        for j from 0 to n
            for k from 0 to n
                C[i,j] = C[i,j]+A[i,k]*B[k,j]
            end for
        end for
    end for
```

3 Execution Methodology

3.1 Execution Model

In order to conduct a comprehensive analysis of the Matrix Multiplication algorithm, we implemented and executed the algorithm using three different programming languages: Java, Go, and Python. The goal was to compare the performance of these languages in terms of their execution time for matrix multiplication tasks.

To ensure a fair comparison, we designed three distinct scenarios for each language, involving matrix sizes of 4x4, 256x256, and

1024x1024. The number of threads spawned during execution was determined based on the number of rows in the first input matrix, allowing for parallel processing and optimization of performance.

Each scenario was executed multiple times, specifically three times, to account for any variations in execution time. This helped in obtaining more reliable and consistent results. The average response time across the three runs was calculated and used as a performance metric for comparison.

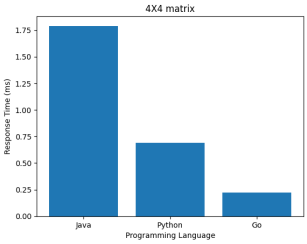
During the execution of each scenario, we also monitored the CPU utilization to assess the efficiency of resource utilization by each language. This provided insights into the impact of parallel processing on the overall utilization of computing resources.

By following this methodology, we aimed to obtain accurate and meaningful results that would enable a fair comparison of the performance of Java, Go, and Python in terms of their execution time for matrix multiplication tasks. The methodology ensured that the analysis was conducted under similar conditions, allowing us to draw valid conclusions regarding the performance of each language.

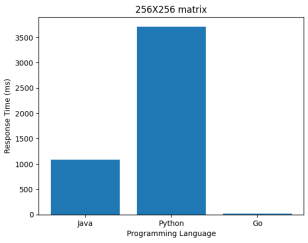
In the subsequent sections, we will present the findings of our analysis, including the average response times and CPU utilization for each language in the different matrix multiplication scenarios. This will provide a comprehensive understanding of the performance characteristics of Java, Go, and Python in handling computationally intensive matrix multiplication operations.

3.2 Results

The findings from the comparative analysis of matrix multiplication implementation using Java, Go, and Python offer valuable insights into the performance and efficiency of these programming languages.

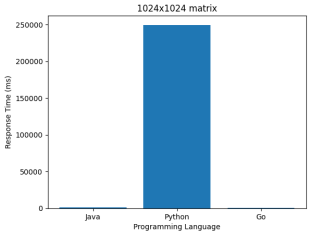


In this particular scenario, the sequential execution outperforms parallel execution in terms of time due to the relatively small size of the input dataset. When the data is divided into smaller chunks and processed in parallel, the overhead introduced by parallelization outweighs the benefits gained from parallel execution. Therefore, in this specific case, the sequential approach proves to be more efficient and effective in terms of overall execution time.

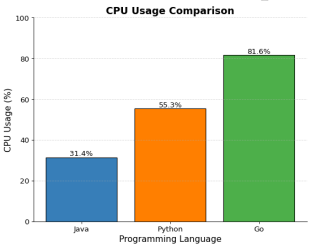


Among the three languages, Go demonstrates the fastest response time for the multiplication of a 256 x 256 matrix. Java performs moderately well in comparison. However, Python exhibits the slowest execution time in this case. Interestingly, the time taken by Python to compute a 256 x 256 matrix is significantly higher compared to its execution time for a 4 x 4 matrix. This slowdown can be attributed to the Global Interpreter Lock (GIL) implemented by Python.

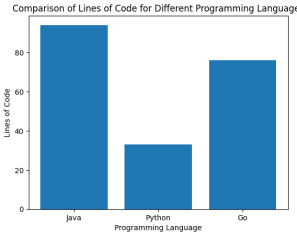
Since matrix multiplication is a CPU-bound process, the introduction of parallelization does not lead to an improvement in response time. This limitation arises from the GIL, which restricts multiple threads from simultaneously entering the critical section. Consequently, the benefits of parallel execution are not fully realized in Python for matrix multiplication tasks.



When considering a larger matrix size, such as 1024 x 1024, Go continues to demonstrate optimal performance compared to the other two languages. Java maintains a moderate execution time, while Python remains the slowest language due to the overhead imposed by the Global Interpreter Lock (GIL). The presence of GIL in Python introduces limitations that hinder the efficient utilization of resources for parallel execution, resulting in slower computation times for matrix multiplication tasks.



During the execution of matrix multiplication for a 1024 x 1024 matrix, Go exhibits the highest CPU consumption among the three languages. On the other hand, Java demonstrates the lowest CPU usage in this particular scenario.



In terms of implementing matrix multiplication, Java is known for its verbosity, often requiring a larger amount of code compared to other languages. On the other hand, Python stands out as a more compact language, allowing for a concise implementation of multithreading logic for matrix multiplication with fewer lines of code.

4 Conclusions

In conclusion, the use of multithreading implementation can significantly improve the performance time of CPU-bound and compute-intensive applications. Based on the comparison of the three programming languages, Go provides the fastest response times, but its CPU consumption increases for larger datasets. Java, while verbose, is still a reliable choice for multithreading and offers flexibility for developers. On the other hand, Python may not be ideal for CPU-bound applications due to its slow performance caused by the Global Interpreter Locking mechanism. Therefore, selecting the right programming language is essential in optimizing software performance. Developers should carefully consider the specific requirements of their applications before choosing the language for implementation to achieve the desired performance and resource utilization.

References

[1] Wuxue Jiang, Qi Li, Zhiming Wang, and Jianfeng Luo. A framework of concurrent mechanism based on java multithread. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 11, 09 2013.

[2] Radu Rugina and Martin Rinard. Pointer analysis for multithreaded programs. *ACM SIGPLAN Notices*, 34(5):77–90, 1999.

[3] Naohiro Togashi and Vitaly Klyuev. Concurrency in go and java: Performance analysis.

In *2014 4th IEEE International Conference on Information Science and Technology*, pages 213–216, 2014.