# ***Matrix Multiplication Speedup Comparison***

**Name: Abdullah Fayyaz**

**RollNumber:21l-5208**

**Section:6-A**

***Part 0***:

CPU Information:

Model Name: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz

Processor Family: Pentium Pro (AKA the P6)

Processor Vendor: GenuineIntel

Vector Capabilities: AVX2

No. of Cores: 4

No. of hyper Threads: 8

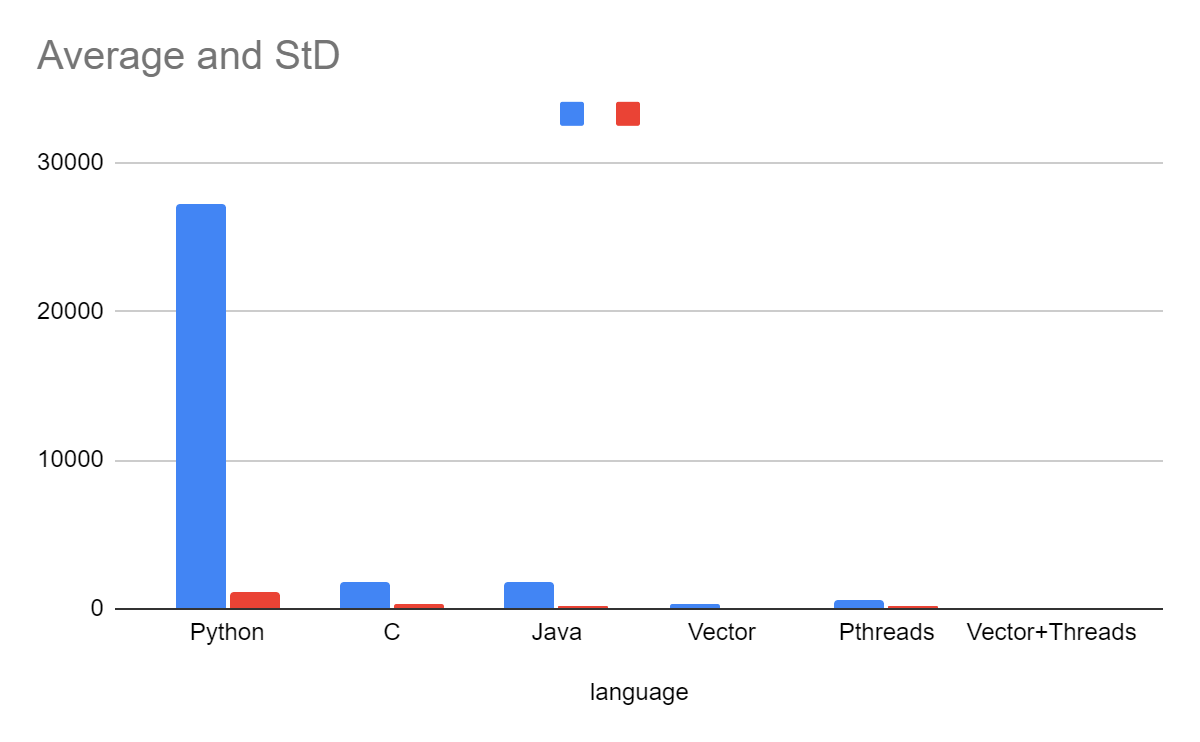
Ram Amount: 5022 Mb

Disk Type: SSD

CPU Utilization:1%(Of VirtualMachine)

Memory Utilization:0.5%(Of VirtualMachine)

*Matrix Multiplication SpeedUp Graph:*



Red: Standard Deviation

Blue: Average Execution time

**Readings**

|  |  |  |
| --- | --- | --- |
| **Language** | **Average Execution Time** | **Standard Deviation** |
| Python | 27160.73 | 1127.0421660553 |
| C | 1811.587 | 371.84802149035 |
| Java | 1814.74625 | 228.92671360987 |
| C Vector Instructions | 389.93305 | 29.210204851191 |
| C Pthreads | 565.742475 | 152.66125117328 |
| C Pthreads+Vector Instructions | 113.29675 | 47.021839250501 |

# ***Part 5***

# ***(A)***

This assignment proved to be immensely instructive, deepening my understanding of the concept of speed and illustrating the remarkable transformation in matrix multiplication runtime. Initially, the operation took an extensive 7.5 hours when implemented in Python. However, through strategic optimizations, the execution time dramatically reduced to a mere 2 minutes when employing C programming, shedding light on the significant impact of programming languages on speed.

Furthermore, the exploration of parallelization techniques, such as utilizing pthreads and vector instructions, yielded even more substantial improvements. The runtime, initially reduced to 30 minutes by transitioning from Python to C, was further optimized to a mere 2 minutes through the implementation of pthreads and vector instructions. This comprehensive analysis underscored not only the role of programming languages but also the pivotal impact of advanced optimization techniques in achieving remarkable speedup in computational tasks.

# ***(B)***

Switching programming languages proved to be a pivotal factor in maximizing speedup, with the execution time plummeting from a daunting 7.5 hours to a more manageable 30 minutes upon transitioning from Python to C. This initial transformation underscores the inherent efficiency gains associated with lower-level languages like C.

However, the narrative doesn't stop there—additional layers of optimization, particularly through the integration of vector instructions and multithreading, further contributed to substantial speed improvements. It becomes evident that these techniques are not exclusive to a particular programming language; rather, they serve as universal tools for performance enhancement across diverse language environments.

Upon introducing vector instructions to the C implementation, the execution time experienced another noteworthy reduction, plummeting from 30 minutes to an impressive 6.5 minutes. This emphasizes the critical role that hardware-level optimizations, such as vectorization, play in squeezing out additional speed from a given algorithm.

Multithreading, a parallelization strategy, also played a significant role in the quest for speed. By leveraging eight threads, the execution time decreased from 30 minutes to a commendable 9.5 minutes. This showcases the effectiveness of parallel computation in distributing the workload across multiple cores and achieving faster results.

However, the most striking revelation lies in the combined effect of vector instructions and multithreading. The synergistic implementation of both techniques resulted in an extraordinary reduction of the execution time from 30 minutes to an astonishing 2 minutes. This not only reinforces the cumulative impact of optimization strategies but also underscores the potential for achieving unparalleled speedup when leveraging multiple approaches concurrently.

In essence, this journey from Python to C, accompanied by the strategic integration of vector instructions and multithreading, provides a comprehensive roadmap for achieving optimal performance across programming languages. It highlights the adaptability of optimization techniques and their ability to transcend language barriers, ultimately leading to a profound and multifaceted acceleration of computational tasks.

# ***(C)***

The Karp-Flatt metric, which assesses the performance improvement by considering the serial part and overhead while varying the number of processors, can be insightful in evaluating the quality of speedup achieved through language changes, vectorization, and multithreading. However, it's important to note that the metric traditionally focuses on the impact of increasing the number of physical processors.

If we extend the definition of "processor" to include the parallelization techniques like vectorization and multithreading, we can gain insights into the quality of speedup across the entire optimization spectrum. Here's an analysis based on the extended definition:

**Language Change (Python to C):**

The initial speedup from 7.5 hours to 30 minutes with the transition from Python to C suggests a substantial improvement. However, as this is primarily due to the efficiency of C as a lower-level language, it may not directly correlate with an increase in the number of processors.

**Vectorization (C with Vector Instructions):**

The additional speedup from 30 minutes to 6.5 minutes with vector instructions implies an improvement in parallel processing capabilities. While vectorization doesn't introduce physical processors, it enhances the efficiency of operations by simultaneously processing multiple data elements. The quality of speedup remains high, indicating effective utilization of available resources.

**Multithreading (C with Multithreading):**

The shift to multithreading further contributes to speedup, reducing the execution time from 30 minutes to 9.5 minutes. Multithreading introduces parallelism by utilizing multiple threads, effectively increasing the computational capacity. The quality of speedup remains notable, but the rate of improvement might be influenced by factors like thread synchronization and overhead.

**Combined Effect (Vector + Multithreading):**

The most significant reduction in execution time, from 30 minutes to 2 minutes, occurs when combining vectorization and multithreading. This suggests a synergistic effect, leveraging both vector instructions and multithreading concurrently. The quality of speedup remains high, indicating a well-balanced and effective utilization of parallelization techniques.

In conclusion, the extended definition of "processor" to include vectorization and multithreading provides a nuanced perspective on the quality of speedup. The observations suggest that the speedup quality remains substantial throughout the optimization stages, with the combined effect demonstrating the potential for continued improvement. However our speedups did parallelize after the change in language as it do not changes as high as before so we can say that plateau is present there.