# CPU and GPU Performance for Matrix Multiplication

Course: CSI 4650: Parallel and Distributed Computation, Fall 2024

Instructor: Justin Kur

Group 6: Willow Connelly, Rumana Yeasmin, Emma Gabos

Date: 12-05-24

**Project Overview**

This project investigates the performance differences between CPU and GPU computations, focusing on matrix multiplication using both sequential and parallel approaches. The program supports a single-threaded CPU version with brute force and Strassen methods, loop unrolling optimizations, and GPU-based computation using PyTorch.

The benchmarking involves:

- CPU Brute Force: Baseline implementation for sequential processing.
- GPU with PyTorch: Utilizes GPU acceleration for parallel processing.

We measure and compare computational latency for increasing matrix sizes and explore performance variations due to hardware constraints.

**Matrix Multiplication:**

A foundational operation in numerous fields, including:

- Linear algebra
- Machine learning
- Scientific simulations

**Use cases:**

- **Machine Learning and AI**:

  Matrix multiplication is a cornerstone of deep learning, powering tasks such as neural network training, convolutional layers, and embedding operations. Parallelized GPU-based tensor computations allow scalable training for large datasets, enabling breakthroughs in AI applications like computer vision, natural language processing, and reinforcement learning.

- **Scientific Computing**:

  High-performance matrix multiplication is essential in solving differential equations, molecular simulations, and astrophysics modeling. For example, weather prediction models and climate simulations rely on parallel computation to process immense data sets in real time.

- **Data Analytics**:

  Efficient data transformation and computation pipelines, such as dimensionality reduction (e.g., PCA), correlation analysis, and data clustering, benefit from parallelized matrix operations, making it possible to process millions of rows in seconds.

- **Graphics and Gaming**:

  Real-time rendering of complex scenes and simulations in gaming requires heavy matrix computations for 3D transformations, collision detection, and particle simulations. GPUs handle these tasks to deliver seamless visuals and physics at high frame rates.

- **Autonomous Vehicles**:

  Self-driving systems use parallelized matrix operations for sensor fusion, path planning, and real-time environment mapping, enabling safe and efficient navigation.

**Benchmarking Goals:**

The primary goal of benchmarking in this project is to measure and analyze the computational latency of matrix multiplication across a range of matrix sizes. By doing so, we aim to evaluate the impact of parallelization on performance efficiency when using both CPUs and GPUs. This involves comparing single-threaded execution to parallelized methods, highlighting how different approaches influence computation times. Additionally, the benchmarking seeks to explore the trade-offs between sequential and parallel execution, considering variations in hardware capabilities and data size to provide insights into scalability and optimization opportunities.

**CPU Parallelization Methods**

**Threading**:

- ○ Threads share the same memory space, making them efficient for tasks that require frequent data sharing.
- ○ Best for tasks that have low memory overhead and benefit from lightweight, quick communication between threads.

**PyTorch Parallelization**

- **GPU** Execution:

  PyTorch leverages CUDA cores to perform operations on tensors in parallel.

  - Operations like matrix multiplication are optimized for batch processing.
  - Automatic allocation of tasks across multiple GPU cores ensures efficient resource usages

## <u>Tools and Technologies</u>

**Tools and Platforms:** In order to run the program, we have decide to use the free Google Colab environment.

- **Google Colab**:

    - Cloud-based environment for executing and testing code.
    - Provides access to GPUs for parallel computation experiments.

**Libraries**

- **PyTorch**:
    - GPU-optimized library for tensor computations and matrix multiplication.
    - Used to leverage CUDA for GPU-based acceleration.

**Environment:**

- CPU: Intel Xeon
- Physical cores: 1
- Logical processors (threads): 2
- GPU: NVIDIA Tesla T4 - 2560 CUDA cores

**Challenges**

- Figuring out Google Colab's environment specs
    - Understanding Google Colab's environment specifications was a crucial step for accurate benchmarking and performance evaluation. This involved identifying the number of CPU cores available (typically 2 by default), GPU availability, and memory constraints.
- Correctly timing/benchmarking the code. Using PyTorch's timing functions was necessary.
- Building a parallelizable yet efficient matrix multiplication function that doesn't just use pytorch's or NumPy's built-in matmult functions.

## Future Improvements

- Explore additional GPU optimization techniques.
- Use a better environment; one with a better GPU and more CPU threads to test on.
- Integrate more matrix multiplication algorithms.
- Automate matrix size and iteration count testing for dynamic benchmarking.