Why Everyone is Talking About GPUs

The Matrix Multiplication Story

Your Name

October 11, 2025

MumPy Meetup

[NVIDIA Stock Chart]

Most Valuable Company in the World

GPUs are Everywhere Now

- ChatGPT needs thousands of GPUs
- Image generators like DALL-E run on GPUs
- Self-driving cars use GPUs
- Gaming has always used GPUs

But why?

A Relatable Problem

Question: How long will it take me to reach Andheri?

[Mumbai Map with Andheri marker]

What Affects Travel Time?

- Distance
- Time of day
- Is it raining?
- Weekday/Weekend
- Traffic conditions
- Construction zones

[Feature Icons:]
Distance, Clock,
Rain, Calendar,
Traffic, Construction

Two Approaches

Traditional: Rule-Based

IF (distance ¿ 10km AND is_raining) THEN add 30 minutes...

Machine Learning

Convert everything to numbers, learn a function

We won't cover the learning part today, just the prediction part!

Making a Prediction

For one location, we do:

$$
\begin{aligned}
\text{predicted\_time} = {} & \text{distance} \times w_1 \\
& + \text{time\_of\_day} \times w_2 \\
& + \text{is\_raining} \times w_3 \\
& + \text{is\_weekend} \times w_4
\end{aligned}
$$

The weights (w_1, w_2, w_3, w_4) are learned from data

But What If...

- We have **50 features** instead of 4?
- We want to predict for **1,000 locations** at once?
- We have **multiple layers** of transformations? (deep learning)

This becomes **matrix multiplication**

Matrix Multiplication

[Matrix Multiplication Visual]

$$[1000 \times 50] \times [50 \times 10] = [1000 \times 10]$$

Interactive demo: `http://matrixmultiplication.xyz/`

The Real Scale

Modern ML models:

- GPT-3: 175 **billion** parameters
- Billions of multiply-add operations per prediction
- Training involves trillions of operations

How do we compute this efficiently?

CPU: The Generalist

Few powerful cores

- 4-16 cores typically
- Sequential processing
- Great for general tasks

Analogy: 8 very smart people solving 10,000 problems one by one

[CPU Diagram]
8-16 powerful cores
Sequential processing

The CPU Bottleneck

[CPU Sequential Processing]
Elements computed one-by-one

Matrix multiplication is **inherently parallel**, but CPU does it **sequentially**

GPU: The Parallel Powerhouse

Thousands of smaller cores

- 10,000+ CUDA cores
- Parallel processing
- Specialized for repetitive tasks

Analogy: 10,000 people each solving one problem simultaneously

[GPU Diagram]
10,000+ small cores
Parallel processing

Parallel Matrix Multiplication

[GPU Parallel Processing]

All elements computed simultaneously

Each element computed **simultaneously**!

Why GPUs Exist

GPUs were originally designed for **graphics and gaming**

- 3D transformations: matrix operations
- Rendering millions of pixels: parallel
- Real-time requirements: fast

Turns out: AI has the same needs!

Seeing is Believing

Python Timing Examples

Demo 1: NumPy vs CuPy

Setup: Large matrix multiplication

CPU (NumPy)

```
import numpy as np
import time

size = 10000
A = np.random.rand(size, size)
B = np.random.rand(size, size)

start = time.time()
C = A @ B
end = time.time()

print(f"Time: {end-start:.2f}s")
```

GPU (CuPy)

```
import cupy as cp
import time

size = 10000
A = cp.random.rand(size, size)
B = cp.random.rand(size, size)

start = time.time()
C = A @ B
cp.cuda.Stream.null.synchronize()
end = time.time()

print(f"Time: {end-start:.2f}s")
```

Demo 1: Results

[Timing Bar Chart]
NumPy (CPU): XX.XX seconds
CuPy (GPU): X.XX seconds

50-100x speedup on GPU!

Demo 2: PyTorch CPU vs GPU

Setup: Neural network forward pass

```
import torch
import torch.nn as nn

# Create a simple neural network
model = nn.Sequential(
    nn.Linear(1000, 5000),
    nn.ReLU(),
    nn.Linear(5000, 5000),
    nn.ReLU(),
    nn.Linear(5000, 1000)
)

# Input data
x = torch.randn(1000, 1000)
```

```
# CPU version
model_cpu = model.to('cpu')
x_cpu = x.to('cpu')
output = model_cpu(x_cpu)
```

```
# GPU version
model_gpu = model.to('cuda')
x_gpu = x.to('cuda')
output = model_gpu(x_gpu)
```

Demo 2: Results

[Timing Bar Chart]
PyTorch CPU: XX.XX seconds
PyTorch GPU: X.XX seconds

100-200x speedup for deep learning!

When Should You Care About GPUs?

Use GPUs when you have:

- Large matrices (thousands of rows/columns)
- Repeated operations (training loops)
- Real-time requirements
- Deep learning models

Skip GPUs for:

- Small data (¡ 1000 elements)
- One-off calculations
- Overhead of data transfer ¿ computation time

How to Get Started

Free options:

- **Google Colab** - Free T4 GPU access
- **Kaggle Notebooks** - Free GPU hours

Cloud providers:

- AWS, Google Cloud, Azure
- Pay per hour of GPU usage

Local GPU:

- NVIDIA GPUs (required for CUDA)
- For serious/regular ML work

Python Tools for GPU Computing

PyTorch	Most popular for deep learning, easy GPU support
TensorFlow	Google's framework, mature ecosystem
CuPy	NumPy, but on GPU
JAX	Google's new framework, automatic differentiation
RAPIDS	GPU-accelerated data science (pandas-like)

Most just need `.to('cuda')` or similar!

Putting It All Together

1. Modern AI = **lots of matrix multiplication**
2. Matrix multiplication = **embarrassingly parallel**
3. CPUs = sequential, GPUs = **parallel powerhouses**
4. Result: **50-200x speedup** for ML workloads

That's why every AI breakthrough mentions "GPU hours"!

Thank You!

Questions?

Demo: `http://matrixmultiplication.xyz/`