An Introduction to Graphics Processing Units and Tensor Processing Units

Soumadeep Ghosh

Kolkata, India

Abstract

This paper provides a comprehensive introduction to Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs), two critical hardware architectures that have revolutionized computational paradigms in computer graphics, scientific computing, and artificial intelligence. We examine the architectural foundations, operational principles, performance characteristics, and application domains of these processors, with particular emphasis on their role in accelerating parallel workloads and machine learning tasks.

The paper ends with "The End"

1 Introduction

The evolution of computing has been marked by increasing specialization in processor architectures. While Central Processing Units (CPUs) excel at sequential processing and general-purpose computation, the demands of modern applications have necessitated the development of specialized accelerators. Graphics Processing Units and Tensor Processing Units represent two such evolutionary branches, each optimized for distinct but increasingly convergent workload characteristics.

GPUs emerged from the need to render complex graphics in real-time, while TPUs were purpose-built for machine learning workloads. Both architectures exploit massive parallelism to achieve performance levels unattainable by traditional CPUs for their respective domains.

2 Graphics Processing Units

2.1 Historical Context and Evolution

The GPU originated in the late 1990s as fixed-function graphics accelerators. NVIDIA's GeForce 256, introduced in 1999, was the first processor marketed as a GPU. The transition to programmable shaders in the early 2000s, followed by the introduction of general-purpose GPU (GPGPU) computing frameworks like CUDA in 2006, transformed GPUs from specialized graphics processors into powerful parallel computing engines.

2.2 Architectural Principles

The GPU architecture is fundamentally different from CPU design. Where a CPU might have 4 to 64 cores optimized for sequential performance with large caches and complex control logic, a modern GPU contains thousands of smaller, simpler cores designed for throughput rather than latency.

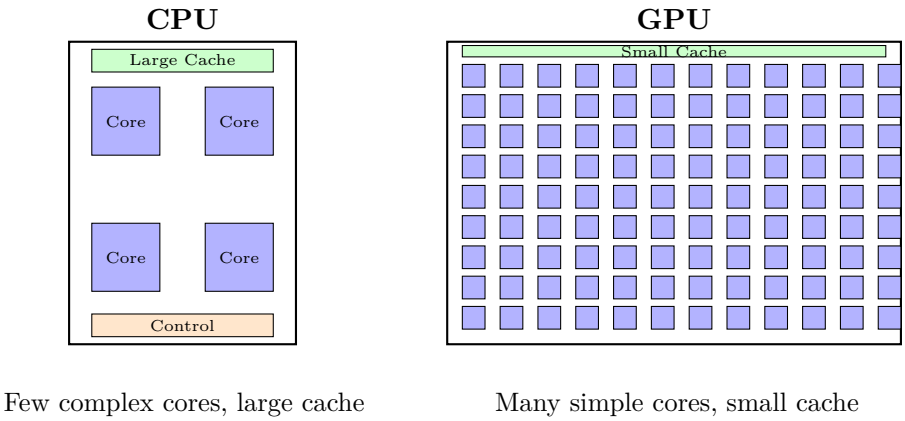

Figure 1: Architectural comparison between CPU and GPU designs

A typical modern GPU consists of multiple streaming multiprocessors (SMs), each containing numerous CUDA cores (in NVIDIA terminology) or stream processors (in AMD terminology). These cores share memory hierarchies including registers, shared memory, L1 cache, L2 cache, and global memory.

2.3 Parallel Processing Model

GPUs employ the Single Instruction Multiple Thread (SIMT) execution model. In this paradigm, multiple threads execute the same instruction on different data simultaneously. Threads are organized hierarchically into warps (typically 32 threads), blocks, and grids, allowing programmers to express massive parallelism.

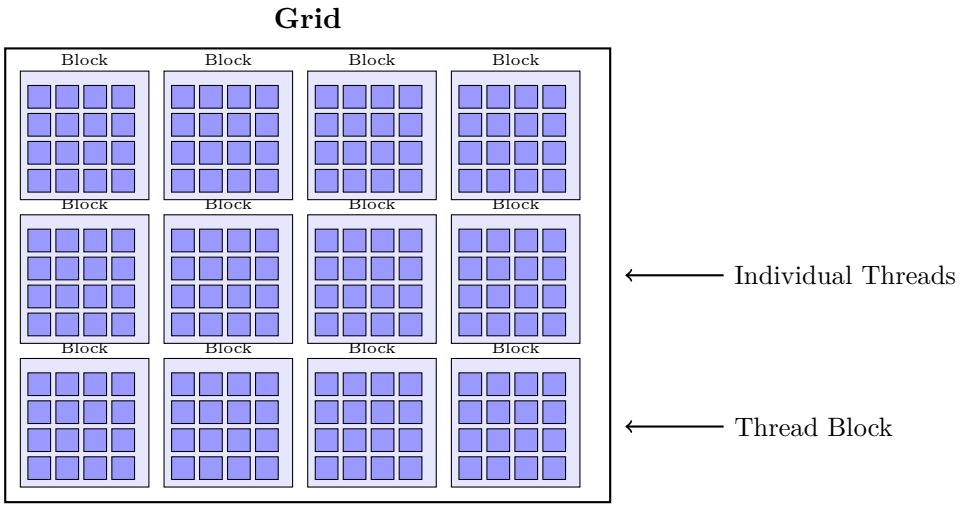

Figure 2: GPU thread hierarchy: grid of blocks containing threads

2.4 Memory Hierarchy

The GPU memory system is designed to maximize bandwidth for parallel access patterns. Global memory provides large capacity but higher latency, while shared memory within each SM offers low latency for inter-thread communication within a block. Modern GPUs also feature texture memory and constant memory with specialized caching behavior.

Memory coalescing is critical for performance. When threads in a warp access consecutive memory addresses, the hardware combines these into fewer transactions, dramatically improving bandwidth utilization.

2.5 Applications Beyond Graphics

While GPUs were designed for rendering, their parallel architecture has proven remarkably effective for diverse computational tasks including scientific simulations, molecular dynamics, computational fluid dynamics, cryptographic operations, and most notably, machine learning. The introduction of tensor cores in recent GPU generations specifically accelerates matrix operations common in neural network training and inference.

3 Tensor Processing Units

3.1 Origins and Design Philosophy

Tensor Processing Units were developed by Google and first deployed in their datacenters in 2015. Unlike GPUs which evolved from graphics processors, TPUs were designed from the ground up for machine learning workloads. The architecture reflects a singular focus on accelerating the tensor operations that dominate neural network computations.

3.2 Architectural Innovation

The TPU architecture centers on a massive systolic array of multiply-accumulate (MAC) units. A systolic array is a network of processing elements that rhythmically compute and pass data through the array, analogous to how blood pulses through the circulatory system.

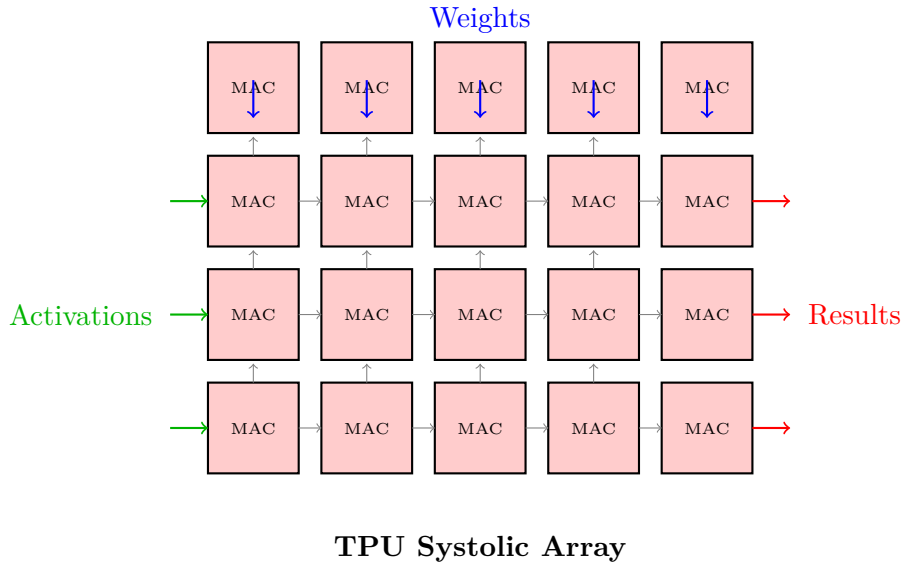

Figure 3: Simplified systolic array architecture for matrix multiplication

The first-generation TPU featured a 256×256 systolic array capable of 65,536 multiply-accumulate operations per cycle. With reduced precision arithmetic (8-bit integers for inference), this architecture achieved over 92 teraops per second, far exceeding contemporary GPUs for specific neural network operations.

3.3 Quantization and Precision

A key insight in TPU design is that neural network inference often tolerates reduced numerical precision. The TPU architecture aggressively employs quantization, using 8-bit or 16-bit integers rather than 32-bit floating point, which increases throughput and reduces power consumption while maintaining acceptable accuracy for most models.

3.4 Memory and I/O Design

TPUs feature large on-chip memory to minimize external memory access. The high bandwidth memory (HBM) provides the data throughput necessary to keep the systolic array fed. The architecture is optimized for batch processing, where multiple inputs are processed simultaneously to maximize hardware utilization.

3.5 Successive Generations

Subsequent TPU generations have expanded capabilities. TPU v2 and v3 added support for training in addition to inference, incorporated floating-point computation, and increased computational density. TPU v4, announced in 2021, delivers significant performance improvements with enhanced inter-chip interconnects for distributed training.

4 Comparative Analysis

4.1 Performance Characteristics

Both GPUs and TPUs excel at parallel computation but with different characteristics. GPUs offer greater flexibility and programmability, supporting arbitrary computational graphs and a wide variety of operations. TPUs achieve superior performance and energy efficiency for the specific matrix operations central to neural networks but are less flexible for general computation.

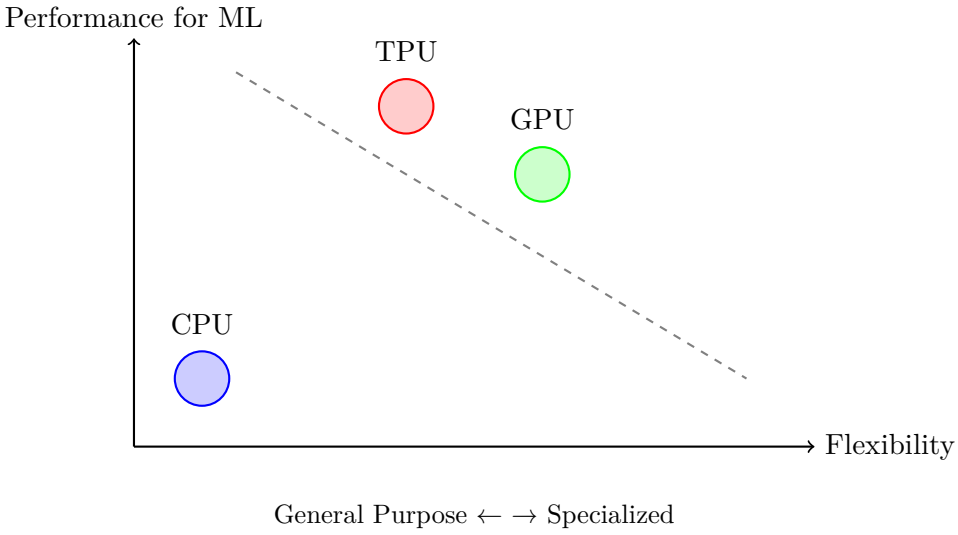

Figure 4: Conceptual positioning of processors in the flexibility-performance space

4.2 Power Efficiency

Power efficiency, measured in operations per watt, strongly favors TPUs for neural network workloads. The specialized architecture eliminates unnecessary functionality and optimizes data movement, which typically consumes more energy than computation. GPUs have improved efficiency through tensor cores and other ML-specific features but remain more power-hungry for equivalent throughput on matrix operations.

4.3 Ecosystem and Accessibility

GPUs benefit from a mature ecosystem with extensive software support, including CUDA, ROCm, OpenCL, and high-level frameworks like PyTorch and TensorFlow. Development tools, libraries, and community resources are abundant. TPUs, being Google-proprietary hardware, have more limited accessibility, primarily through Google Cloud Platform, though the open-source nature of TensorFlow provides good software support.

4.4 Cost Considerations

The total cost of ownership differs significantly between these platforms. GPUs are available for purchase and on-demand cloud rental, with pricing spanning consumer to datacenter grades. TPUs are accessed exclusively through cloud services with pricing models optimized for large-scale batch processing. For many applications, TPUs offer superior cost-performance ratios at scale, while GPUs provide better economics for development, experimentation, and smaller deployments.

5 Programming Models

5.1 GPU Programming

GPU programming typically employs CUDA (for NVIDIA hardware) or OpenCL (cross-platform). Developers write kernel functions executed by many threads in parallel. Memory management, thread synchronization, and optimization for memory coalescing require explicit attention.

Modern machine learning frameworks abstract much of this complexity, automatically generating optimized GPU code from high-level model descriptions. However, performance tuning for custom operations still often requires low-level optimization.

5.2 TPU Programming

TPU programming occurs primarily through TensorFlow and JAX frameworks. The XLA (Accelerated Linear Algebra) compiler optimizes computational graphs for TPU execution. Developers describe models in high-level Python APIs, and the framework handles placement and execution on TPU hardware.

The programming model emphasizes batching and graph optimization. Unlike GPUs where incremental computation is common, TPUs favor ahead-of-time compilation and execution of complete computational graphs.

6 Future Directions

The boundary between GPUs and TPUs is blurring as both architectures incorporate features from each other. Modern GPUs include tensor cores that function similarly to TPU systolic arrays for matrix operations. Meanwhile, TPU designs are becoming more flexible to accommodate diverse neural network architectures and operations.

Emerging trends include sparse computation support to exploit the sparsity common in neural networks, mixed-precision training that adaptively uses different numerical precisions, and tighter integration with CPU and memory systems to reduce data movement overhead. The development of domain-specific languages and intermediate representations promises to improve portability across different accelerator architectures.

As neural network models grow in size and complexity, distributed training across multiple accelerators has become essential. Both GPU and TPU ecosystems are investing heavily in high-bandwidth interconnects and sophisticated parallelization strategies to enable training of models with hundreds of billions or even trillions of parameters.

7 Conclusion

Graphics Processing Units and Tensor Processing Units represent complementary approaches to accelerating computationally intensive workloads. GPUs evolved from graphics processors into versatile parallel computing platforms, while TPUs were purpose-built for machine learning. Both architectures achieve orders-of-magnitude performance improvements over CPUs for their target applications through massive parallelism and specialized computational units.

The choice between GPU and TPU depends on specific requirements including workload characteristics, flexibility needs, ecosystem considerations, and cost constraints. As machine learning continues

to drive computational demands, we can expect continued innovation in both architectures, along with the emergence of new specialized processors targeting specific aspects of the AI computational pipeline.

Understanding these architectures is increasingly essential for practitioners in machine learning, scientific computing, and high-performance computing. As these technologies mature, they are democratizing access to computational capabilities that were inconceivable just decades ago, enabling breakthroughs across scientific disciplines and commercial applications.

References

[1] NVIDIA Corporation. *CUDA C++ Programming Guide*. NVIDIA Developer Documentation, 2023.

[2] Jouppi, N. P., Young, C., Patil, N., et al. *In-Datacenter Performance Analysis of a Tensor Processing Unit*. Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA), 2017.

[3] Hennessy, J. L. and Patterson, D. A. *Computer Architecture: A Quantitative Approach*, 6th Edition. Morgan Kaufmann, 2017.

[4] Kirk, D. B. and Hwu, W. W. *Programming Massively Parallel Processors: A Hands-on Approach*, 3rd Edition. Morgan Kaufmann, 2016.

[5] Google Cloud. *Cloud TPU System Architecture*. Google Cloud Documentation, 2023.

[6] NVIDIA Corporation. *NVIDIA Tensor Core GPU Architecture*. NVIDIA Technical Whitepaper, 2020.

[7] Patterson, D. A. *Latency Lags Bandwidth*. Communications of the ACM, Volume 47, Issue 10, 2004.

[8] Khronos Group. *OpenCL 3.0 Specification*. Khronos OpenCL Working Group, 2020.

[9] Abadi, M., Barham, P., Chen, J., et al. *TensorFlow: A System for Large-Scale Machine Learning*. Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2016.

[10] Vaswani, A., Shazeer, N., Parmar, N., et al. *Attention Is All You Need*. Advances in Neural Information Processing Systems 30 (NIPS), 2017.

[11] Sarah, R. *Machine Learning Systems: Design and Implementation*. MIT Press, 2022.

[12] Dean, J. and Ghemawat, S. *MapReduce: Simplified Data Processing on Large Clusters*. Communications of the ACM, Volume 51, Issue 1, 2008.

Glossary

ASIC Application-Specific Integrated Circuit. A chip designed for a specific purpose rather than general computation.

Bandwidth The rate at which data can be transferred, typically measured in bytes per second or gigabytes per second.

Batch Processing Processing multiple inputs simultaneously to improve hardware utilization and throughput.

Cache Fast memory close to processing cores that stores frequently accessed data to reduce latency.

CUDA Compute Unified Device Architecture. NVIDIA's parallel computing platform and programming model for GPU programming.

High Bandwidth Memory (HBM) Advanced memory technology providing much higher bandwidth than traditional DRAM through 3D stacking and wide interfaces.

Inference Using a trained neural network to make predictions on new data, as opposed to training.

Kernel A function executed on a GPU by many parallel threads.

Latency The time delay between initiating an operation and receiving its result.

Matrix Multiplication A fundamental operation in neural networks where two matrices are multiplied, forming the basis of most neural network layer computations.

Memory Coalescing Combining multiple memory accesses into fewer transactions to improve bandwidth utilization.

Quantization Reducing the precision of numerical representations, such as converting 32-bit floating point to 8-bit integers.

SIMT Single Instruction Multiple Thread. An execution model where many threads execute the same instruction on different data.

Streaming Multiprocessor (SM) The basic processing unit in NVIDIA GPU architecture containing multiple CUDA cores and shared resources.

Systolic Array A network of processing elements that rhythmically compute and pass data, optimized for matrix operations.

Tensor A multi-dimensional array of numbers, generalizing concepts like scalars, vectors, and matrices.

Tensor Core Specialized hardware units in modern GPUs designed to accelerate matrix multiplication operations.

Teraops One trillion operations per second, a measure of computational throughput.

Throughput The amount of work completed per unit time, as opposed to the time for a single operation (latency).

Warp A group of threads (typically 32) that execute together in lockstep on a GPU.

XLA Accelerated Linear Algebra. A compiler for optimizing TensorFlow computations for various hardware backends including TPUs.

The End