A Survey Comparing Specialized Hardware and Evolution in CPU, GPU AND TPU for Neural Network

FRANCIS CHIGOZIE EMMANUEL[1], ODIKWA NDUBUISI HENRY[2], ONYEMAOBI BETHRAM CHIBUZO[3]
[1]*Department of Computer Science, Clifford University*
[2]*Department of Computer Science, Abia State University*
[3]*Department of Computer Science, Gregory University Uturu*

Abstract- ***This survey paper is focused on the genesis of GPU AND TPU from first generation and their architectures. The paper compares CPUs, GPUs, and TPUs, their hardware architectures, their similarities and differences were extensively discussed. The batch of data are immensely used these days but they require more time, computation and energy. Due to the greater demand and attractive options for architects to explore, companies are continuously working to reduce training and inference response time. Pertinent to the demands and cost factors different kinds of ASICs (application specific integrated circuits) are developed and research is increased in this area. Many models of CPUs, GPUs and TPUs have been developed to support these networks and to improve training and inference phase. The hardware of CPUs and GPUs can be sold to businesses while Google offers TPU processing for everyone from the cloud. When the data is away from the computational source, it increases the overall cost and to reduce this cost companies implements memory management and caching techniques close to ALUs.***

Indexed Terms— GPU, TPU, CPU, Deep Learning, ALU, framework.

I. INTRODUCTION

The concept of number-crunching on the GPU is, almost, as old as the GPU itself. Early solutions revolved around the idea of manipulating pixel data through shader language, as a means of forming simple floating-point calculations in a dummy graphics shell. Essentially, where in rendering we perform per-pixel operations in the context of colour space, early GPU computation used those colour components to conceal the numerical data which needed processing. In some cases, just to make this function, the researchers then had to force the GPU to render something to get the results out the other end (generally two triangles).

Around 2004, researchers began taking this idea very seriously. A lot of problems, particularly simulation problems, have significant amounts of physical data to consider; in computing terms, physical points are often handled as three element vectors. It is not difficult to see how this mapped conveniently to the colour variables in rendering. Similarly, many of the problems researchers focused upon were relatively straightforward mathematics and, where scale was a problem rather than complexity, it was believed that the GPU offered a cost-effective improvement to performance.

Windows Vista changed the playing field with DirectX 10s unified shader model. Prior to this, shader cores had very specific tasks and were largely incapable of performing any other task (different instruction sets for different shader types). With this move towards unified shaders came an industry sea-change in favour of more generally capable shaders all-round if the instruction sets needed to be generalized in terms of vertex, pixel and geometry shader need, why not generalize them as far as possible beyond that.

With the advent of CUDA, and later FireStream, researchers gained access to easily programmable APIs (relative to performing GPU computation using shader language) and ever-more-capable hardware. The issue then became one of identifying problems that the GPU could solve well, and deploying those solutions; similarly, avoiding deploying problems to the GPU which did not lend themselves to its strengths.

This survey focuses on the specialized hardware that are used for machine learning and other data processing. For the domain of AI, understanding the relative advantages of these technologies are most and the constraints such as weight, size and power should be considered. The other factors include the memory bandwidth for the model parameters which are used for loading and updating the data.

II. LITERATURE REVIEW

2.1 The central Processing Unit

The full form of CPU is Central Processing Unit. It is a brain of the computer. All types of data processing operations and all the important functions of a computer are performed by the CPU. It helps input and output devices to communicate with each other and perform their respective operations. It also stores data which is input, intermediate results in between processing, and instructions.

A Central Processing Unit is the most important component of a computer system. A CPU is a hardware that performs data input/output, processing and storage functions for a computer system. A CPU can be installed into a CPU socket. These sockets are generally located on the motherboard. CPU can perform various data processing operations. CPU can store data, instructions, programs, and intermediate results.

Now, the CPU consists of 3 major units, which are:

1. Memory or Storage Unit
2. Control Unit
3. ALU (Arithmetic Logic Unit)

2.1.1 Memory or Storage Unit

As the name suggests this unit can store instructions, data, and intermediate results. The memory unit is responsible for transferring information to other units of the computer when needed. It is also known as an internal storage unit or the main memory or the primary storage or Random Access Memory (RAM) as all these are storage devices.

Its size affects speed, power, and performance. There are two types of memory in the computer, which are primary memory and secondary memory. Some main functions of memory units are listed below:

i. Data and instructions are stored in memory units which are required for processing.
ii. It also stores the intermediate results of any calculation or task when they are in process.
iii. The final results of processing are stored in the memory units before these results are released to an output device for giving the output to the user.
iv. All sorts of inputs and outputs are transmitted through the memory unit.

2.1.2 Control Unit

As the name suggests, a control unit controls the operations of all parts of the computer but it does not carry out any data processing operations. For executing already stored instructions, it instructs the computer by using the electrical signals to instruct the computer system. It takes instructions from the memory unit and then decodes the instructions after that it executes those instructions. So, it controls the functioning of the computer. Its main task is to maintain the flow of information across the processor. Some main functions of the control unit are listed below:

i. Controlling of data and transfer of data and instructions is done by the control unit among other parts of the computer.
ii. The control unit is responsible for managing all the units of the computer.
iii. The main task of the control unit is to obtain the instructions or data which is input from the memory unit, interprets them, and then directs the operation of the computer according to that.
iv. The control unit is responsible for communication with Input and output devices for the transfer of data or results from memory.
v. The control unit is not responsible for the processing of data or storing data.

2.1.3 ALU (Arithmetic Logic Unit)

ALU (Arithmetic Logic Unit) is responsible for performing arithmetic and logical functions or operations. It consists of two subsections, which are:

- Arithmetic Section
- Logic Section

Now, let us know about these subsections:

1. Arithmetic Section: By arithmetic operations, we mean operations like addition, subtraction, multiplication, and division, and all these operation and functions are performed by ALU. Also, all the complex operations are done by making repetitive use of the mentioned operations by ALU.
2. Logic Section: By Logical operations, we mean operations or functions like selecting, comparing, matching, and merging the data, and all these are performed by ALU.

Note: CPU may contain more than one ALU and it can be used for maintaining timers that help run the computer system.

The main function of a computer processor is to execute instruction and produce an output. CPU works are Fetch, Decode and Execute are the fundamental functions of the computer.

- Fetch: the first CPU gets the instruction. That means binary numbers that are passed from RAM to CPU.
- Decode: When the instruction is entered into the CPU, it needs to decode the instructions. with the help of ALU(Arithmetic Logic Unit) the process of decode begins.
- Execute: After decode step the instructions are ready to execute
- Store: After execute step the instructions are ready to store in the memory.

Types of CPU

We have three different types of CPU:

- Single Core CPU: The oldest type of computer CPUs is single core CPU. These CPUs were used in the 1970s. These CPUs only have a single core that preform different operations. This means that the single core CPU can only process one operation at a single time. Single core CPU is not suitable for multitasking.

- Dual-Core CPU: Dual-Core CPUs contain a single Integrated Circuit with two cores. Each core has its cache and controller. These controllers and cache are work as a single unit. Dual core CPUs can work faster than the single-core processors.

- Quad-Core CPU: Quad-Core CPUs contain two dual-core processors present within a single integrated circuit (IC) or chip. A quad-core processor contains a chip with four independent cores. These cores read and execute various instructions provided by the CPU. Quad Core CPU increases the overall speed for programs. Without even boosting the overall clock speed it results in higher performance. Figure 1 shows the architecture of a CPU. CPUs works on the principle of fetch and execute. CPUs revolutionized from transistor CPUs to small-scale integrated CPU, large-scale integrated CPU and microprocessors.

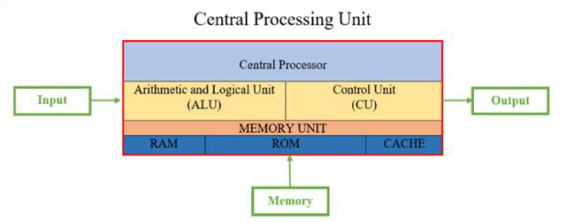

Fig. 1 Central Processing Unit

2.2 Tensor Processing Units (TPU)

With machine learning gaining its relevance and importance every day, the conventional microprocessors have proven to be unable to effectively handle it, be it training on neural network processing. GPUs, with their highly parallel architecture designed for fast graphic processing proved to be way more useful than CPUs for the purpose, but were somewhat lacking. Therefore, in order to combat this situation, Google developed an AI accelerator integrated circuit which would be used by its TensorFlow AI framework. This device has been named TPU (Tensor Processing Unit). The chip has been designed for Tensorflow Framework. TensorFlow is an open-source library developed by Google for its internal use. Its main usage is in

machine learning and dataflow programming. TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays. These arrays are referred to as “tensors”. TensorFlow is available for Linux distributions, Windows, and MacOS.

TPU Architecture

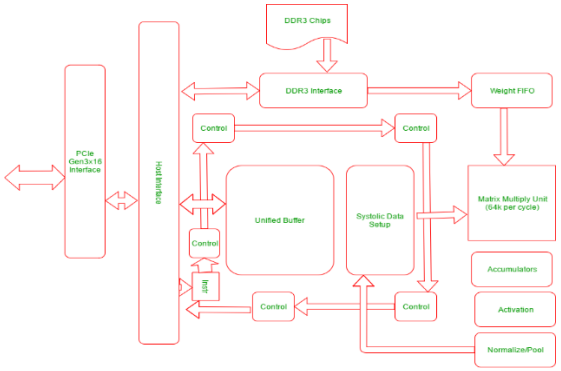

Fig. 2: The following diagram explains the physical architecture of the units in a TPU: (source: computer.org. Retrieved October 31, 2020 from https://www.computer.org/publications/tech-news/chasing-pixels/isit-time-to-rename-the-gpu)

The TPU includes the following computational resources depicted in figure 2:

i. Matrix Multiplier Unit (MXU): 65, 536 8-bit multiply-and-add units for matrix operations.
ii. Unified Buffer (UB): 24MB of SRAM that work as registers
iii. Activation Unit (AU): Hardwired activation functions.

There are 5 major high-level instruction sets devised to control how the above resources work shown in table 1.

Table 1: High Level Instruction Set

TPU Instruction	Function
Read_Host_Memory	Read data from memory
Read_Weights	Read weights from memory
MatrixMultiply/Convolve	Multiply or convolve with the data and weights, accumulate the results
Activate	Apply activation functions
Write_Host_Memory	Write result to memory

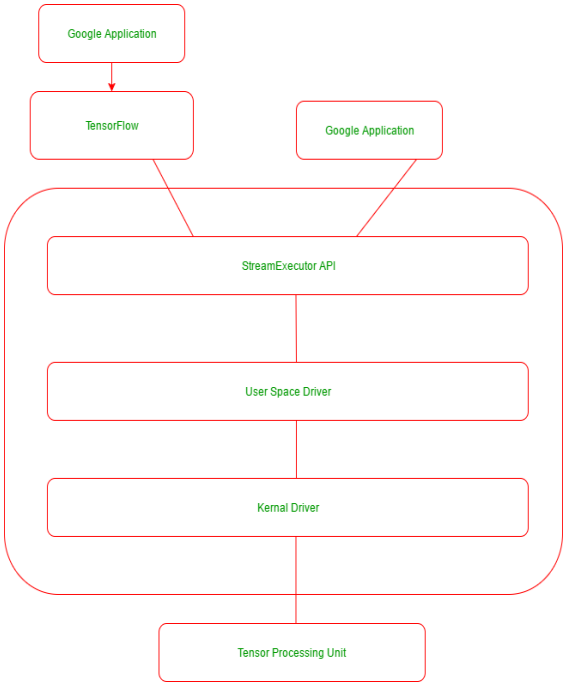

Fig. 3: TensorFlow and TPU: (source; computer.org. Retrieved October 31, 2020 from https://www.computer.org/publications/tech-news/chasing-pixels/isit-time-to-rename-the-gpu)

2.2.1 Advantages of TPU

The following are some notable advantages of TPUs:

1. Accelerates the performance of linear algebra computation, which is used heavily in machine learning applications.
2. Minimizes the time-to-accuracy when you train large, complex neural network models.
3. Models that previously took weeks to train on other hardware platforms can converge in hours on TPUs.

When to use a TPU

- Models dominated by matrix computations.
- Models with no custom TensorFlow operations inside the main training loop.
- Models that train for weeks or months
- Larger and very large models with very large effective batch sizes.

2.3 Graphics Processing Unit (GPU)

As we enter the era of GPU computing, demanding applications with substantial parallelism increasingly use the massively parallel computing capabilities of GPUs to achieve superior performance and efficiency. Today GPU computing enables applications that we previously thought infeasible because of long execution times. With the GPU's rapid evolution from a configurable graphics processor to a programmable parallel processor, the unubiquitous GPU in every PC, laptop, desktop, and workstation is a many-core multi-threaded multiprocessor that excels at both graphics and computing applications. Today's GPUs use hundreds of parallel processor cores executing tens of thousands of parallel threads to rapidly solve large problems having substantial inherent parallelism. They're now the most pervasive massively parallel processing platform ever available, as well as the most cost-effective. Using NVIDIA GPUs as examples, this article describes the evolution of GPU computing and its parallel computing model, the enabling architecture and software developments, how computing applications use CPU & GPU co-processing, example application performance speedups, and trends in GPU computing.

2.3.1 GPU Computing's Evolution

Why have GPUs evolved to have large numbers of parallel threads and many cores? The driving force continues to be the real-time graphics performance needed to render complex, high-resolution 3D scenes at interactive frame rates for games.

Rendering high-definition graphics scenes is a problem with tremendous inherent parallelism. A graphics programmer writes a single-thread program that draws one pixel, and the GPU runs multiple instances of this thread in parallel—drawing multiple pixels in parallel. Graphics programs, written in shading languages such as Cg or High-Level Shading Language (HLSL), thus scale transparently over a wide range of thread and processor parallelism. Also, GPU computing programs—written in C or C++ with the CUDA parallel computing model, or using a parallel computing API inspired by CUDA such as Direct-Compute or Open C scale transparently over a wide range of parallelism. Software scalability, too, has enabled GPUs to rapidly increase their parallelism and performance with increasing transistor density.

Table 2. NVIDIA GPU technology development

Date	Product	Transistors	CUDA cores	Technology
1997	RIVA 128	3 million	—	3D graphics accelerator
1999	GeForce 256	25 million	—	First GPU, programmed with DX7 and OpenGL
2001	GeForce 3	60 million	—	First programmable shader GPU, programmed
				with DX8 and OpenGL
2002	GeForce FX	125 million	—	32-bit floating-point (FP) programmable GPU with
				Cg programs, DX9, and OpenGL
2004	GeForce 6800	222 million	—	32-bit FP programmable scalable GPU, GPGPU
				Cg programs, DX9, and OpenGL
2006	GeForce 8800	681 million	128	First unified graphics and computing GPU,
				programmed in C with CUDA
2007	Tesla T8, C870	681 million	128	First GPU computing system programmed

				in C
				with CUDA
2008	GeForce GTX 280	1.4 billion	240	Unified graphics and computing GPU, IEEE FP,
				CUDA C, OpenCL, and DirectCompute
2008	Tesla T10, S1070	1.4 billion	240	GPU computing clusters, 64-bit IEEE FP, 4-Gbyte
				memory, CUDA C, and OpenCL
2009	Fermi	3.0 billion	512	GPU computing architecture, IEEE 754-2008 FP,
				64-bit unified addressing, caching, ECC memory, CUDA C, C++, OpenCL, and DirectCompute

The table 2 lists significant milestones in NVIDIA GPU technology development that drove the evolution of unified graphics and computing GPUs. GPU transistor counts increased exponentially, doubling roughly every 18 months with increasing semiconductor density. Since their 2006 introduction, CUDA parallel computing cores per GPU also doubled nearly every 18 months.

In the early 1990s, there were no GPUs. Video graphics array (VGA) controllers generated 2D graphics displays for PCs to accelerate graphical user interfaces. In 1997, NVIDIA released the RIVA 128 3D single-chip graphics accelerator for games and 3D visualization applications, programmed with Microsoft Direct3D and OpenGL. Evolving to modern GPUs involved adding programmability incrementally—from fixed function pipelines to micro coded processors, configurable processors, programmable processors, and scalable parallel processors.

Early GPUs

The first GPU was the GeForce 256, a single-chip 3D real-time graphics processor introduced in 1999 that included nearly every feature of high-end workstation 3D graphics pipelines of that era. It contained a configurable 32-bit floating-point vertex transform and lighting processor, and a configurable integer pixel-fragment pipeline, programmed with OpenGL and Microsoft DirectX 7 (DX7) APIs. GPUs first used floating-point arithmetic to calculate 3D geometry and vertices, then applied it to pixel lighting and color values to handle high-dynamic-range scenes and to simplify programming. They implemented accurate floating-point rounding to eliminate frame-varying artifacts on moving polygon edges that would otherwise sparkle at real-time frame rates. As programmable shaders emerged, GPUs became more flexible and programmable. In 2001, the GeForce 3 introduced the first programmable vertex processor that executed vertex shader programs, along with a configurable 32-bit floating-point pixel-fragment pipeline, programmed with OpenGL and DX8. The ATI Radeon 9700, introduced in 2002, featured a programmable 24-bit floating-point pixel-fragment processor programmed with DX9 and OpenGL. The GeForce FX and GeForce 6800 featured programmable 32-bit floating-point pixel-fragment processors and vertex processors, programmed with Cg programs, DX9, and OpenGL. These processors were highly multi-threaded, creating a thread and executing a thread program for each vertex and pixel fragment. The GeForce 6800 scalable processor core architecture facilitated multiple GPU implementations with different numbers of processor cores. Developing the Cg language for programming GPUs provided a scalable parallel programming model for the programmable floating-point vertex and pixel-fragment processors of GeForce FX, GeForce 6800, and subsequent GPUs. A Cg program resembles a C program for a single thread that draws a single vertex or single pixel. The multi-threaded GPU created independent threads that executed a shader program to draw every vertex and pixel fragment.

2.3.2 Unified Computing and Graphics GPUs

The GeForce 8800 introduced in 2006 featured the

first unified graphics and computing GPU architecture programmable in C with the CUDA parallel computing model, in addition to using DX10 and OpenGL. Its unified streaming processor cores executed vertex, geometry, and pixel shader threads for DX10 graphics programs, and also executed computing threads for CUDA C programs. Hardware multithreading enabled the GeForce 8800 to efficiently execute up to 12,288 threads concurrently in 128 processor cores. NVIDIA deployed the scalable architecture in a family of GeForce GPUs with different numbers of processor cores for each market segment.

The GeForce 8800 was the first GPU to use scalar thread processors rather than vector processors, matching standard scalar languages like C, and eliminating the need to manage vector registers and program vector

2.3.3 GPU Computing Systems

At first, users built personal supercomputers by adding multiple GPU cards to PCs and workstations, and assembled clusters of GPU computing nodes. In 2007, responding to demand for GPU computing systems, NVIDIA introduced the Tesla C870, D870, and S870 GPU card, desk side, and rack-mount GPU computing systems containing one, two, and four T8 GPUs. The T8 GPU was based on the GeForce 8800 GPU, configured for parallel computing.

The second-generation Tesla C1060 and S1070 GPU computing systems introduced in 2008 used the T10 GPU, based on the GPU in GeForce GTX 280. The T10 featured 240 processor cores, 1-teraflop-per-second peak single-precision floating-point rate, IEEE 754-2008 double-precision 64-bit floating-point arithmetic, and 4-Gbyte DRAM memory. Today there are Tesla S1070 systems with thousands of GPUs widely deployed in high-performance computing systems in production and research.

NVIDIA introduced the third-generation Fermi GPU computing architecture in 2009. Based on user experience with prior generations, it addressed several key areas to make GPU computing more broadly applicable. Fermi implemented IEEE 754-2008 and significantly increased double-precision performance. It added error-correcting code (ECC) memory protection for large-scale GPU computing, 64-bit unified addressing, cached memory hierarchy, and instructions for C, C++, Fortran, OpenCL, and Direct Compute

2.3.4 GPU Computing Architecture

To address different market segments, GPU architectures scale the number of processor cores and memories to implement different products for each segment while using the same scalable architecture and software. NVIDIA's scalable GPU computing architecture varies the number of streaming multi-processors to scale computing performance, and varies the number of DRAM memories to scale memory bandwidth and capacity.

Each multithreaded streaming multiprocessor provides sufficient threads, processor cores, and shared memory to execute one or more CUDA thread blocks. The parallel processor cores within a streaming multi-processor execute instructions for parallel threads. Multiple streaming multiprocessors provide coarse-grained scalable data and task parallelism to execute multiple coarse-grained thread blocks (possibly running different kernels) in parallel. Multithreading and parallel-pipelined processor cores within each streaming multiprocessor implement

III. COMPARISON OF GPU VS TPU VS NPU

It is no secret that artificial intelligence (AI) is driving innovation, particularly when it comes to processing data at scale. Machine learning (ML) and deep learning (DL) algorithms, designed to solve complex problems and self-learn over time, are exploding the possibilities of what computers are capable of. As the problems we ask the computers to solve get more complex, there's also an unavoidable, explosive growth in the number of processes they run. This growth has led to the rise of specialized processors and a whole host of new acronyms.

Joining the ranks of central processing units (CPUs), which you may already be familiar with, are neural processing units (NPUs), graphics processing units (GPUs), and tensile processing units (TPUs).

So, let's dig in to understand how some of these specialized processor's work, and how they're

different from each other. If you're still with me after that, stick around for an IT history lesson. I'll get into some of the more technical concepts about the combination of hardware and software developments in the last 100 or so years.

3.1 Pros and Cons

Flexibility: GPUs can handle a wide range of tasks, including graphics rendering, simulations, and scientific computing, in addition to machine learning workloads.

Maturity: GPUs have been widely adopted for deep learning, and there is a vast ecosystem of software and tools built around them, such as CUDA, cuDNN, and popular deep learning frameworks like TensorFlow and PyTorch.

Precision: GPUs offer a range of precision options, from low-precision FP16 to high-precision FP64, making them suitable for various workloads with different accuracy requirements.

Power Consumption: GPUs typically consume more power than TPUs, which can be a concern for large-scale deployments and energy efficiency.

Cost: High-performance GPUs can be expensive, especially for small businesses or individual researchers.

Performance: TPUs are designed specifically for tensor operations, resulting in faster training and inference times for neural networks compared to GPUs.

Energy Efficiency: TPUs are more power-efficient than GPUs, making them a better choice for large-scale machine learning deployments.

Ease of Use: TPUs are integrated with popular machine learning frameworks like TensorFlow, making it easy for developers to leverage their capabilities.

Limited Ecosystem: The TPU ecosystem is less mature than that of GPUs, with fewer software and tools available.

Availability: TPUs are primarily available through Google Cloud Platform, which may not be suitable for all users and organizations.

3.1.1 The Rise of Specialized Processors

When a computer is given a task, the first thing the processor has to do is communicate with the memory, including program memory (ROM)—designed for more fixed tasks like startup—and data memory (RAM)—designed for things that change more often like loading applications, editing a document, and browsing the internet. The thing that allows these elements to talk is called the bus, and it can only access one of the two types of memory at one time. In the past, processors ran more slowly than memory access, but that's changed as processors have gotten more sophisticated. Now, when CPUs are asked to do a bunch of processes on large amounts of data, the CPU ends up waiting for memory access because of traffic on the bus. In addition to slower processing, it also uses a ton of energy. Folks in computing call this the Von Neumann bottleneck, and as compute tasks like those for AI have become more complex, we've had to work out ways to solve this problem. One option is to create chips that are optimized to specific tasks. Specialized chips are designed to solve the processing difficulties machine learning algorithms present to CPUs. In the race to create the best AI processor, big players like Google, IBM, Microsoft, and Nvidia have solved this with specialized processors that can execute more logical queries (and thus more complex logic). They achieve this in a few different ways. The ways are as follow

IV. LIMITATIONS OF THIS WORK

This study didn't cover the cloud overhead, multi-node systems, accuracy, or convergence. Those areas are for future work, as each deserves in-depth study. For example, evaluating inference entails different metrics, such as latency, and a different experimental setup, as network overhead may have a large effect. NVIDIA's eight-node DGX-1 or Google's 256-TPU systems are not studied here. Studying multi-node systems involves more system parameters, including numbers of nodes, inter-node bandwidth, inter-connect topology, and synchronization mechanisms. Cloud system overhead also becomes more acute in multi-node systems.

The validity of extrapolating training throughput to time to-accuracy remains an open question. Recent work studied the number of training steps to accuracy as a function of batch sizes It shows that very large batch size results in sub-linear scaling, but the best batch size depends largely on the model and optimizer. In a multi-node system, synchronization becomes more complicated, which results in different convergence behaviour.

CONCLUSION

In conclusion, GPUs and TPUs each have their pros and cons when working with neural networks. GPUs are versatile and supported by a mature ecosystem, while TPUs excel in performance and energy efficiency for machine learning tasks. The choice between them depends on your specific requirements, budget, and development environment. Assess the advantages and limitations of each option to determine the best fit for your project.

We present architectural bottlenecks of the TPU platform and provide suggestions for future improvement. we compare the hardware and software of the TPU, GPU, and CPU platforms. We present several new observations and insights into the design of specialized hardware and software for deep learning and motivate the need for further work in this field.

REFERENCES

[1] "TensorFlow: Using JIT compilation https://www.tensorflow.org/xla/jit," 2018.

[2] "https://cloud.google.com/tpu/docs/system-architecture," *Google Cloud Documentation*, 2018.

[3] Adolf, R. Rama, S. Reagen, B., and Brooks, D. "Fathom: Reference workloads for modern deep learning methods," in *Workload Characterization (IISWC), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 1–10.

[4] Amodei, D. Ananthanarayanan, S. Anubhai, R. Bai, J. Battenberg, E. Case, C. Casper, J. Catanzaro, B. Cheng, Q. and Chen G., "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International Conference on Machine Learning*, 2016, pp. 173–182.

[5] Bahrampour, S. Ramakrishnan, N. Schott, L. and Shah, M. "Comparative study of Caffe, Neon, Theano, and Torch for deep learning," in *ICLR*, 2016.

[6] Banner, R. Hubara, I. Hoffer, E. and Soudry, D. "Scalable methods for 8-bit training of neural networks," *arXiv preprint arXiv:1805.11046*, 2018.

[7] Bienia, C. Kumar, S. Singh, J. P. and Li, K. "The PARSEC benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 72–81.

[8] Blog, G. A. "Introducing GPipe, an open source library for efficiently training large-scale neural network models," *https://ai.googleblog.com/2019/03/introducing-gpipe-open-sourcelibrary.html*, 2019.

[9] Case, L. "Volta Tensor Core GPU achieves new AI performance milestones," *Nvidia Developer Blog*, 2018.

[10] Che, S. Boyer, M. Meng, J. Tarjan, D. Sheaffer, J. W. Lee, S.-H. and Skadron, K. "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. Ieee, 2009, pp. 44–54.

[11] Chen, T. Moreau, T. Jiang, Z. Shen, H. Yan, E. Wang, L. Hu, Y. Ceze, L. Guestrin, C. and Krishnamurthy, A. "TVM: End-to-end optimization stack for deep learning," *arXiv preprint arXiv:1802.04799*, 2018.

[12] Chen, T. Chen, Y. Duranton, M. Guo, Q. Hashmi, A. Lipasti, M. Nere, A. Qiu, S. ebag, M. S and Temam, O. "BenchNN: On the broad potential application scope of hardware neural network accelerators," in *Workload Characterization (IISWC), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 36–45.

[13] Onyemaobi, C.B. and Ajah, I.A. (2017) "Comparative analysis of web development languages performances", Int. J. Web Science, Vol. 3, No. 1, pp.16–31.

[14] Onyemaobi B.C. (2012). Design and Implementation of Aerospace Information System. Lokoja: Lambert Academic Publishing.

[15] O. B. Chibuzo and D. O. Isiaka, “Design and Implementation of Secure Browser for ComputerBased Tests,” Int. J. Innov. Sci. Res. Technol., vol. 5, no. 8, pp. 1347–1356, 2020, doi: 10.38124/ijisrt20aug526.

[16] Bethram Chibuzo, O., Philip Omoniyi, A.: Network and complex systems developing a signal booster for improved communication in remote areas (2019). https://doi.org/10.7176/NCS