VE370 Project 3 Paper, Dec 7, 2018

Why Artificial Intelligence Accelerators Are Needed for Machine Learning Inherent Tasks

Delmwin Baeka[1]*

Abstract
Application-based Artificial Intelligence demand instantaneous feedback when it comes to voice recognition, text sentiment analysis, facial recognition amongst others. Datacenters for such applications require methods that can compute results in real time. Traditional methods included the use of several CPUs and GPUs for computation, with the latter proving more promising; however, in recent times, the question for the need for AI-specific accelerators arises since a few of such devices tend to outperform CPUs and GPUs. This paper recognizes the use of CPUs, GPUs and AI accelerators (FPGAs and ASICs) by comparing their AI computation performance and architecture, with more detailed analysis on AI accelerator types. Despite, the lower flexibility for AI accelerators beyond AI tasks, a few of them outperform the traditional CPU and GPU. In terms of overall performance for AI computing, AI accelerators perform about 40 times GPUs and 80 times CPUs. Moreover, newer models that exploit the advantages of all three further intensify the performance of AI accelerators.

Keywords
Neural Network—Architecture—AI Accelerator—Performance—Artificial Intelligence—Matrix Multiplication—SIMD

[1] *Electrical and Computer Engineering, University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai, China*
***Author email**: dbaekajnr@sjtu.edu.cn

Contents

Introduction

Background

Real-time AI feedback is one of the gains necessary in user experience. Datacenters with lots of computing power are used to handle complex AI problems to provide the necessary real-time experience. Today's AI applications rely deeply on Neural Networks (NNs), and any reference to AI focuses in this field. A typical application of Neural Networks is speech recognition, which has been improved by 30% from usig AI accelerators over traditional methods used in the past [1].

Neural Networks

Neural networks are based on how the brain computes data by replication neurons artificially, coupled with the use of non-linear functions such as the ReLU, sigmoid, tanh and other functions [1][2]. NNs operate in two stages: training and inference. Training stage generally emphasizes the use of forward propagation to find the required weights that minimize the error in prediction for a labeled set of data. Once training is done on a labeled set of data and validation is done on the labeled data to ensure high prediction accuracy, inference is done on new set of data. In general, traditional hardware like the CPU and GPU can perform forward propagation as good as AI accelerators due to the dependency on concurrency [1][3]. The next step, inference, relies heavily on computation of data since the weights developed through training are represented as matrices. Each inference stage requires the multiplication of the weights matrix and the new input, and then aggregating the results before applying an activation function to find the prediction. For example, as shown in Figure 1, a neural network of three inputs and two fully connected neurons requires 6 matrix multiplications [2].

Common types of NNs are MLP, CNN and RNN[1]. MLPs use multiple layers by reusing the weights output from previous layers to compute newer weights. CNNs use weights that are spatially close to each other to compute weights in the next layer. RNNs keeps both previous weights and current weight outputs for reuse when necessary such as LSTM

[1]MLP – Multi-Layer Perceptron, CNN – Convolutional Neural Network, RNN – Recurrent Neural Network, LSTM – Long Short-Term Memory

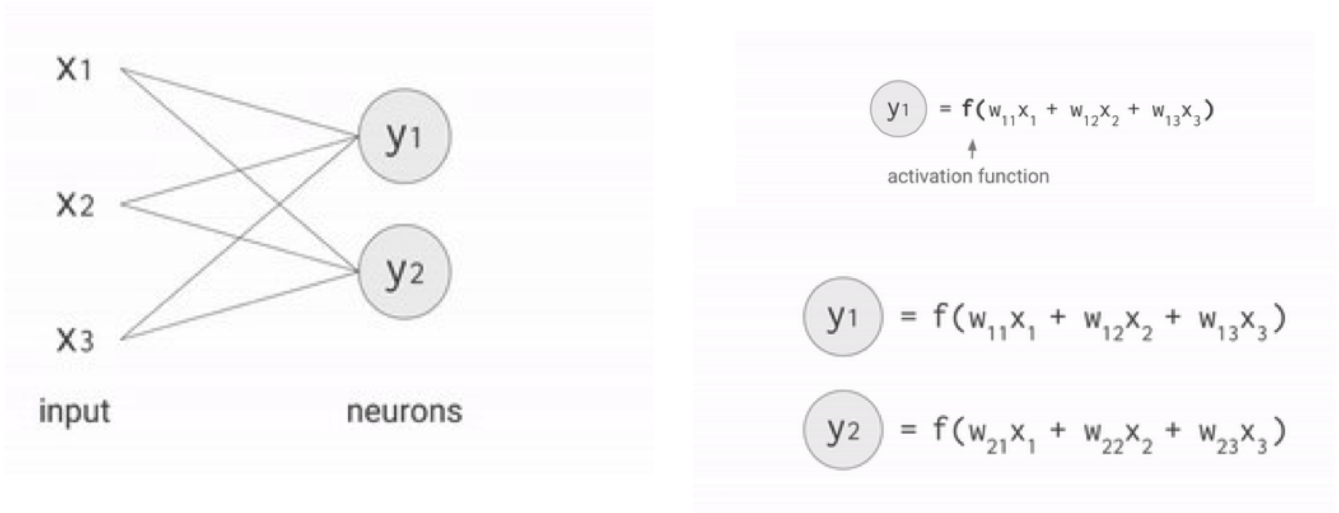
X1
X2
X3
input
neurons
$y_1 = f(w_{11}x_1 + w_{12}x_2 + w_{13}x_3)$
activation function
$y_1 = f(w_{11}x_1 + w_{12}x_2 + w_{13}x_3)$
$y_2 = f(w_{21}x_1 + w_{22}x_2 + w_{23}x_3)$

Figure 1. Structure of a simple single layer Neural Network [left] and calculations of outputs and activation function [right]

that decides whether to go long-term (previous weights) or short-term (new weights). Table 1 shows the various types of networks and how many weights are needed [2].

Table 1. Table showing number of layers and weights in different types of Neural Networks

Network Type	No. of Network Layers	No. of Weights (million)
MLP0	5	20
MLP1	4	5
LSTM0	58	52
LSTM1	56	34
CNN0	16	8
CNN1	89	100

From the table above, the weights range, in millions, from 5 to 100 and that results in a tremendous amount of computations for each inference[2]. The rest of the paper discusses the approaches in design of CPUs, GPUs and common AI accelerators such as Google's ASIC TPU and FPGA based type[2]. The results compare their relative performances and why AI accelerators are important for real-time AI needs.

1. Methods

The various types of hardware processors are investigated. First, by architecture and then their performance on a benchmarked Neural Network task is compared. The participants are the CPU, GPU and Google's TPU to represent the AI Accelerators. By investigating how each architecture does computations, their inherent nature can be observed to see how they exploit the neural network topology.

[2]CPU – Central Processing Unit, GPU – Graphical Processing Unit, FPGA – Field Programmable Gate Array, ASIC – Application Specific Integrated Circuit, TPU - Tensor Processing Unit

1.1 Central Processing Unit

CPUs thrive on complexity of operation and as such can handle a wide variety of complex tasks. Through concurrency and exploitation of Single Instruction, Multiple Data (SIMD), matrix arithmetic can be greatly sped up on CPUs [4].

1.2 Graphics Processing Unit

Popularists term GPUs as CPUs on steroids. GPUs typically contain many 100 times of cores and sometimes each possess their own cache. GPUs can handle many more threads than a typical CPU can and hence this makes GPUs great for complex calculations.

1.3 AI Accelerators

AI accelerators combine the best parts of GPGPUs[3] and CPUs. The reason is that what works best for AI in terms of computation needs might reduce the performance for other computations for other tasks on CPUs. The same for GPUs. Since AI accelerators are for AI tasks only, they do not have these drawbacks. Two kinds of AI accelerators are below.

ASIC ASIC comprise processors typically designed for specific applications such as bitcoin mining and digital voice recording. ASICs are typically very energy efficient and have high performance. Their only drawback is lack of reconfigurability [3]. The design explored in this paper is based on Google's TPU.

FPGA FPGAs are a balance between high performance, efficiency and versatility due to re-configurability. FPGAs however require lots of skill for redesigning the configuration for a different configuration. FPGAs exploit the topology of the NNs by reproducing them as hardware [3].

[3]GPGPU - General Purpose computation on a GPU

2. Results

To better understand the need for Neural network-specific accelerators, the architecture of the traditional processing units are analysed to determine their drawbacks and features. Perormance comparison gives a broader view on which type is better for Nerual Network applications.

2.1 Architecture

The architecture of processing unit shapes how it handles instructions. CPUs have the most varied form of architecture and one of the most complex due to control and predictions. GPUs are not left out of the complex order since their concurrent nature demands a lot of precision to ensure synchronous computations. AI Accelerators thrive on simplicity since that reduces the fiscal and power cost and allows for faster manipulations during computations.

2.1.1 CPU

The first architectural exploitation of modern CPUs is concurrency. Memory locality ensures that nearby memory addresses for a given memory location are also loaded into the cache of the processor for High-Performance Computing (HPC). Based on this principle, the best way to exploit the CPU will be through software. By encoding instructions such that the inner loops of matrix multiplication is stored contiguously. For Eqn 1, storing the A as row-major order and B as column-major order allows for fast matrix computation [4].

$$C = A * B \tag{1}$$

The introduction of accumulators also allows CPUs the opportunity for parallel cores to pipeline operations and distribute them over the cores after unrolling speeds up the computation[4].

The processor used in this study is the Intel Haswell processor architecture. Figure 2 shows how the structure of the multi-core CPU [5]. Instructions fetched are shared between all threads and the processor has three dispatch ports for 256-bit integer SIMD[4] executions. The access to memory is done using integer bypass network for fast forwarding register access. More information on how the Haswell architecture exploits SIMD and parallelism can be found from [5]. By improving the speed for matrix computations, neural networks can be sped up in the inference stage using low-level SIMD parallelization. However, unlike other common Machine Learning algorithms such as regression classifiers that can parallelize the input data and accumulate results from each thread to compute the final prediction, neural networks since each weight in the layer depends on the input from a different neuron or input layer. This creates a limitation for CPUs in terms of speeding up neural network computations [6].

[4]SIMD - Single Instruction, Multiple Data

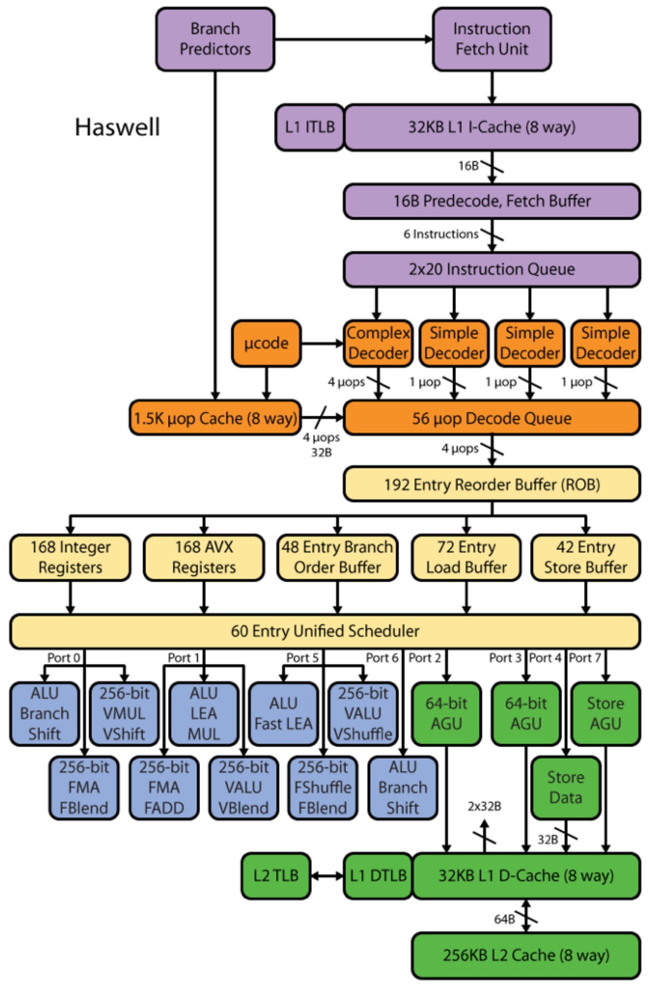

Figure 2. Intel Haswell Micro-architecture

2.1.2 GPU

Using the NVidia GeForce GTX 280 as base model for architecture analysis, Figure 3 shows the architectural layout. The GPU shown in Figure 3 has 240 total cores (30 multiprocessors with 8 stream processors each [6]. Though, detailed introduction to GPU architecture is beyond the scope of this paper, the basic ideas will be established. GPUs were originally used to handle pixel shaders that were functions for that calculated the color of each pixel based on matrix-inspired inputs [7]. The jump from just color calculations came as result of similarities between simulation calculations that also worked with grids of data points and the interactions between each data point. GPUs then got extended to handle these partial differential equations (PDEs). The creation of GPGPUs (General Purpose computation on a GPU) extended pixel shaders to compute kernels. Compute kernels extended the dimensionality from 2 to many and from single output as color to many outputs [7]. Also each multicore as shown in Figure 3. allows the compute kernels to share data through the shared memory.

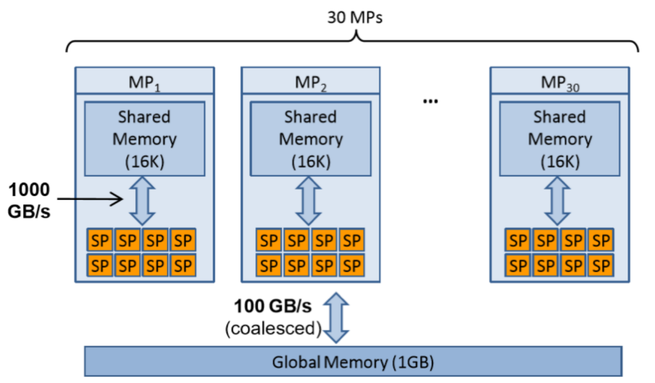

Figure 3. Simplified schematic for the Nvidia GeForce GTX 280 graphics card, with 240 total cores (30 multiprocessors with 8 stream processors each)

The NVidia CUDA[5] model typically has two-levels of parallelism: the multicore processors (MPs) and each having stream processors (SPs) for calculations. Computations can be split into blocks each containing many threads. Each MP can handle a different block and each SP handles the threads. Threaded calculations have access to the shared memory which is very fast and allows for synchronous computations. Blocks however have access to high bandwidth global memory, typically 4GB in most modern GPUs. The global memory is slower than the shared memory, however, coalesced access ensures that data is arranged consecutively, speeding up access time for each block. In terms of computations, GPUs are extremely faster than CPUs. The bottleneck that still exists is in the transfer between GPU and RAM[6]. The only way to reduce this is through GPU-specific instructions of most programs [6]. NNs are typically developed such that each input is handled by an MP and the weights calculation is sped up using the SPs. The bottleneck aforementioned is reduced by scheduling transfer between the Global Memory and RAM only when large chunks of data are needed to be transferred, and hence as much data must always be kept on the global memory.

2.1.3 TPU (ASIC)

Google's TPUs focus on the utilizing and improving the parts of CPUs and GPUs important for NNs. First is quantization. Quantization is an optimization technique that uses an 8-bit integer to approximate an arbitrary value between a preset minimum and a maximum value. Quantization can help reduce model sizes by about 4 times [2]. Compared to GPUs, TPUs have about 65,536 8-bit integer multipliers, whilst GPUs have a few thousands of 32-bit floating-point multipliers. To address the re-configurability issue, TPUs are built on top of the Complex Instruction Set Computer (CISC) for fast high-level instructions.

The TPU has 3 units as shown in Figure 4: Matrix Multiplier Unit (MXU), SRAM[7] Unified Buffer (UB) and Activation Unit (AU) for hardwired activation functions [2]. The above coupled with the TPU instruction set [2] allows for faster NNs. Whilst CPUs and GPUs exploit vectorization by processing the matrices unrolled as vectors for SIMDs, TPUs work with matrices directly. The MXU processes hundreds of thousands of matrix operations in one single clock cycle.

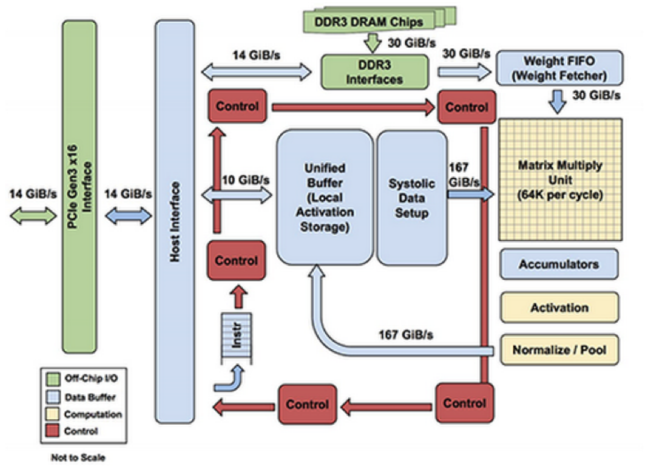

Figure 4. TPU Block Diagram Architecture

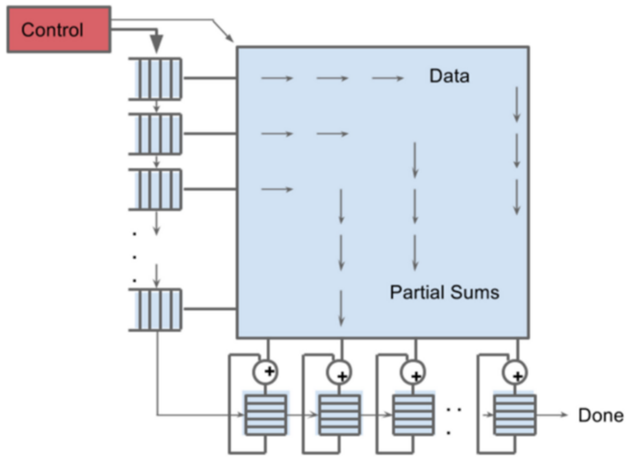

Figure 5. Matrix Multiplier Unit (MXU) of the TPU

The MXU shown in Figure 5 operates using systolic array [2]. The systolic array takes in data input once and stores it in a structure that allows it to be reused for future arithmetic. CPUs and GPUs, due to their complex structure, rely heavily on a lot of control and need to read registers every step of instruction. MXUs are thus inherently faster since a wave front arrangement of input allows fast multiplication of matrix weight and input [2].

The TPU Matrix Multiplication Unit has a systolic array mechanism (Figure 6) that contains $256 \times 256 = 65,536$ ALUs [2]. That means a TPU can process 65,536 multiply-

[5]CUDA - https://en.wikipedia.org/wiki/CUDA
[6]RAM - Random Access Memory
[7]SRAM - Static Random Access Memory

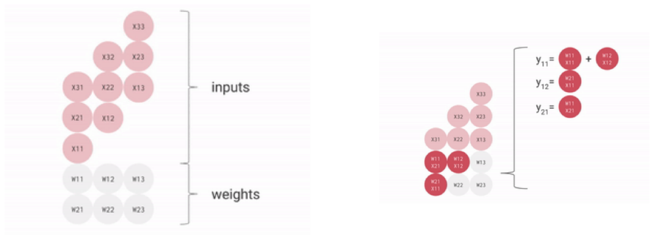

Figure 6. Systolic Array Multiplication of a Matrix

and-adds for 8-bit integers every cycle. A TPU that runs at 700MHz can compute $65,536 \times 700,000,000 = 46 \times 10^{12}$ multiply-and-add operations or 92 Teraops per second (92×10^{12}) in the matrix unit [2].

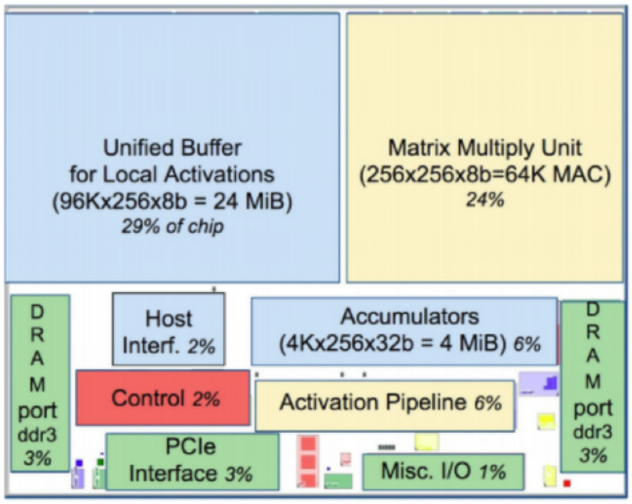
Unified Buffer for Local Activations (96Kx256x8b = 24 MiB) 29% of chip
Matrix Multiply Unit (256x256x8b=64K MAC) 24%
DRAM port ddr3 3%
Host Interf. 2%
Accumulators (4Kx256x32b = 4 MiB) 6%
DRAM port ddr3 3%
Control 2%
Activation Pipeline 6%
PCIe Interface 3%
Misc. I/O 1%

Figure 7. Floor Space Layout of TPU on a Single Die

Another design feature of TPUs is the removal of unnecessary control units found in typical CPUs and GPUs. This allows for fast design and deterministic approach since less prediction as in CPUs and GPUs. This is achieved since NNs follow a fixed flow of operations. The design of a TPU is smaller than other chips as seen from the floor space in Figure 7 [2].

2.1.4 FPGA

A typical FPGA approach design is shown in Figure 8 for the building blocks for each type of NN [3]. Here, each unit of the NN is designed into modules using any hardware description language (HDL). The various units are modeled using typical logic circuits such as registers, and muxes. Each type of NN such as MLP can combine the necessary units for operation. More details on FPGA can be found in [3][8]. The performance comparison focuses mainly on the ASIC version for AI accelerators.

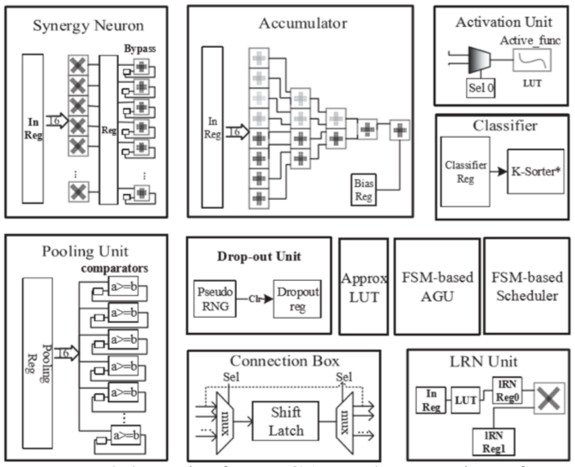

Figure 8. Module Units for FPGA Implementation of Neural Network Accelerator

2.2 Performance Comparison

After comparing[8] the architectures of CPU, GPU and TPU, various comparisons can be made based on NNs.

Operations Per Cycle Operations per cycle measures how many operations can be computed per clock cycle for a typical for a typical NN application. TPUs have the most performance using this criterion as verified from Table 2.

Table 2. Table showing Operation per cycle for the types of hardware processors

	Operations Per Cycle
CPU	Tens
GPU	Tens of Thousands
TPU	Hundreds of Thousands to 128K

Performance Per Watt Performance per watt ratio can also be used to measure the performance of each architecture as shown in Figure 9.

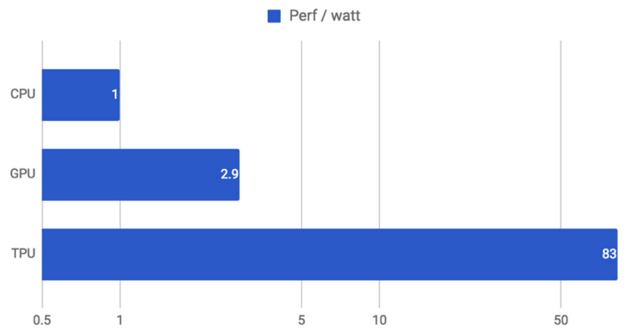

Figure 9. Performance per Watt Comparison for CPU, GPU and TPU

[8]Comparisons were made using TPUv1, Intel Haswell (CPU) and NVidia K80 (GPU) [2]

Throughput Throughput measures how much output or input is handled per second in each processing unit. Figure 10 compares the throughput in predictions (output) per second for each architecture.

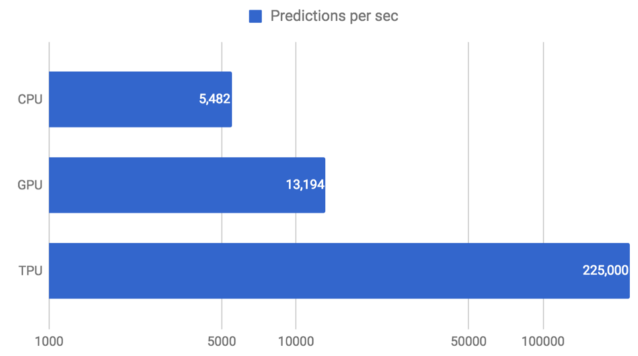

Figure 10. Throughput in predictions per sec for different processing units

Overall Comparison For typical NN tasks for each type of NN as described in Table 1, the following overall performance can be measured based on number of predictions from the NN per second. Figure 11 compares the overall performance for each processing unit using the different types of NNs shown in Table 1.

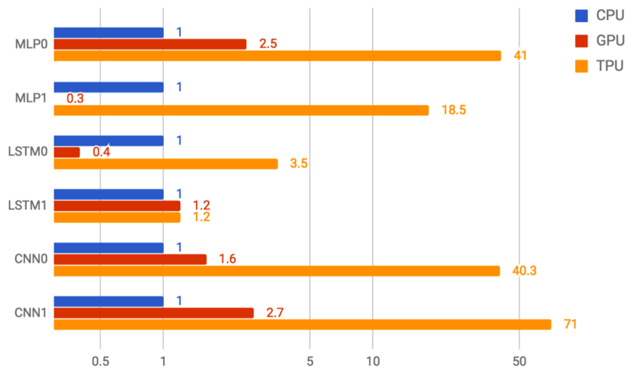

Figure 11. Overall comparison for each processing unit

3. Discussion

When it comes to meeting user expectations, latency is one issue that must be addressed. AI accelerators offer far superior performance for real-time AI capabilities such as instant voice recognition. Mobile platforms of today such as Apple's A12 bionic processor has an AI accelerator co-processor for fast AI computations, putting devices that employ it such as the iPhone X ahead of its competitors.

In terms of overall performance Google's TPU is beyond its competitors. With newer iterations of the TPU, that is the Version 3, even more performance can be squeezed out of TPUs. Albeit their limitations, when it comes to new NNs, Ai accelerators that are built on custom instruction sets can exploit the software exploit like Google does with the Tensorflow API.

AI accelerators can be improved by using some of the best technology from GPUs and CPUs such as concurrency per TPU and using GDDR5 for faster bandwidth. Another opportunity for improvement is doing away with the von Neumann computing structure where input, ALU and memory are separate. The brain, for instance, operates using the synapses for both storage and computing, reducing power consumption by 1000 times. Research from IBM suggests ways of performing computation in memory [9].

References

[1] Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, Richard C. Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. *CoRR*, abs/1704.04760, 2017.

[2] Kaz Sato, Cliff Young, and David Patterson. An in-depth look at google's first tensor processing unit (tpu) — google cloud blog, Mar 2017.

[3] Ying Wang, Jie Xu, Yinhe Han, Huawei Li, and Xiaowei Li. Deepburning: automatic generation of fpga-based learning accelerators for the neural network family. In *Proceedings of the 53rd Annual Design Automation Conference*, page 110. ACM, 2016.

[4] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.

[5] David Kanter. Intel's haswell cpu microarchitecture, Nov 2012.

[6] Rajat Raina, Anand Madhavan, and Andrew Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International*

Conference on Machine Learning, ICML '09, pages 873–880, New York, NY, USA, 2009. ACM.

[7] Igor Ostrovsky. How gpu came to be used for general computation, Mar 2018.

[8] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric S Chung. Accelerating deep convolutional neural networks using specialized hardware. *Microsoft Research Whitepaper*, 2(11), 2015.

[9] Abu Sebastian, Manuel Le Gallo, Geoffrey W. Burr, Sangbum Kim, Matthew BrightSky, and Evangelos Eleftheriou. Tutorial: Brain-inspired computing using phase-change memory devices. *Journal of Applied Physics*, 124(11):111101, 2018.