Summary of In-Datacenter Performance Analysis of a Tensor Processing Unit

Runyu Zheng (runyuz), Shangquan Sun (sunsean)

Problem and Motivation

The paper is generally a performance and power analysis of the Tensor Processing Unit (TPU) architecture at Google. TPU achieves an improvements of cost-energy-performance in neural network inference comparing to CPU and GPU. Since inference tasks demand strict latency requirements, TPU is more suitful for inference tasks. On the other hand, GPUs are optimized for the training of deep neural networks. And CPUs have none of the advantages of domain specific architecture. Therefore, the analysis and evaluation of TPU and the comparison between TPU, GPU and CPU can indicate the the improvement of TPU's performance.

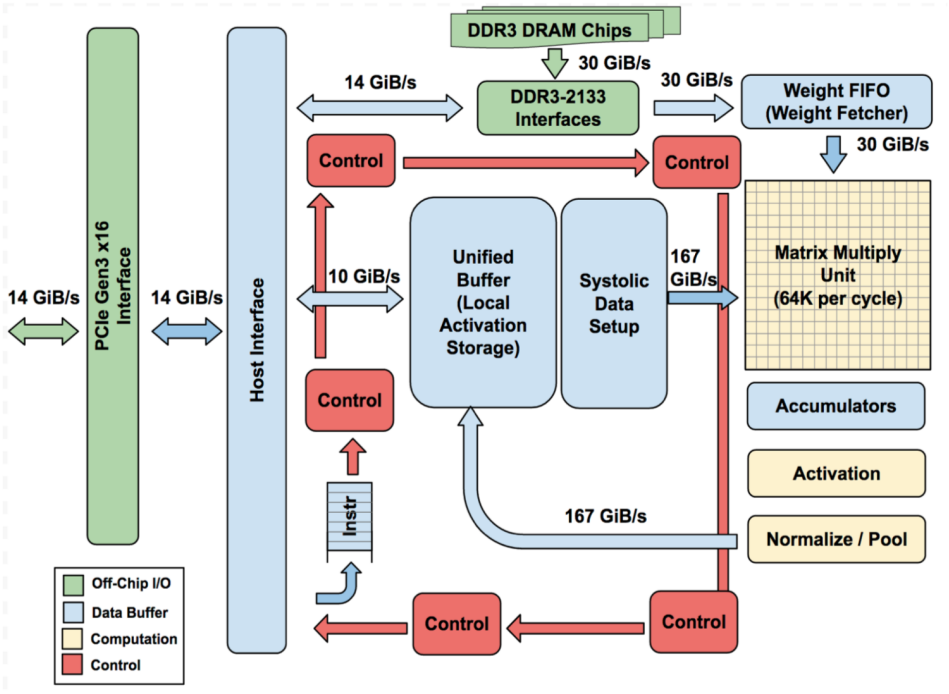
Hypothesis

TPU achieves higher performance in inference of deep neural network, while GPUs are optimized for training. Below is a table shows the details of each deep neural network models that are inferenced on Google's TPU in 2016.

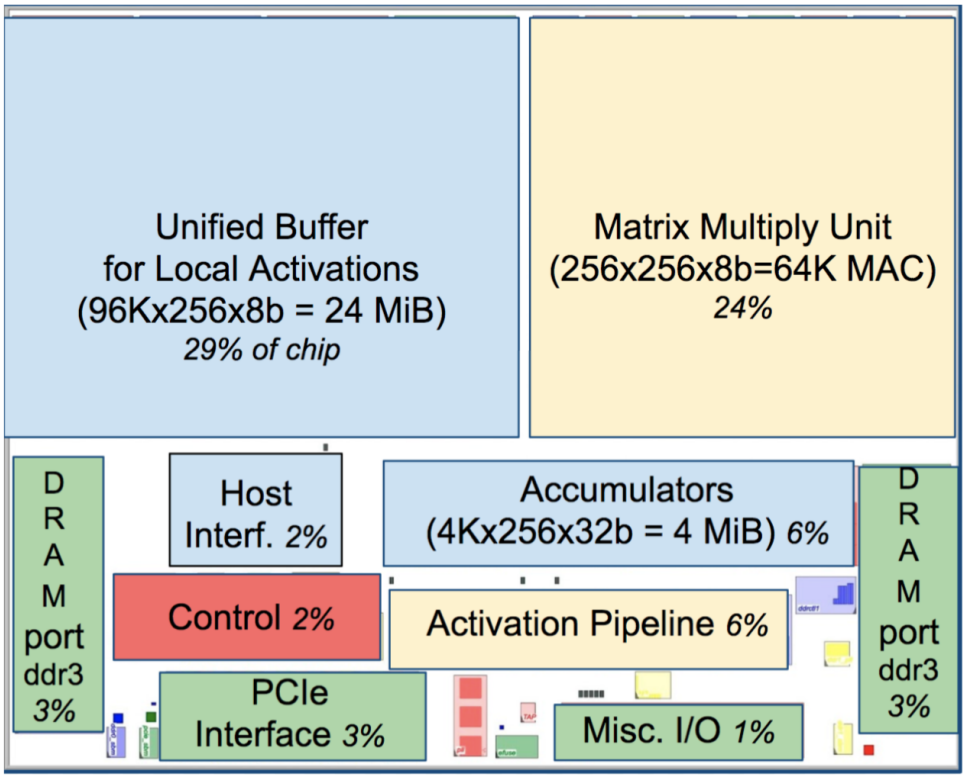
Name	*LOC*	*Layers*					*Nonlinear function*	*Weights*	*TPU Ops / Weight Byte*	*TPU Batch Size*	*% of Deployed TPUs in July 2016*
		FC	*Conv*	*Vector*	*Pool*	*Total*					
MLP0	100	5				5	ReLU	20M	200	200	61%
MLP1	1000	4				4	ReLU	5M	168	168	
LSTM0	1000	24		34		58	sigmoid, tanh	52M	64	64	29%
LSTM1	1500	37		19		56	sigmoid, tanh	34M	96	96	
CNN0	1000		16			16	ReLU	8M	2888	8	5%
CNN1	1000	4	72		13	89	ReLU	100M	1750	32	

Solution Overview

To reflect the increasing amount of inference tasks like speech recognition, Google started to design TPU in 2013 aiming to improve cost-performance by 10X over GPUs in inference, and completed the design in 15 months. The block diagram of the TPU is shown below:

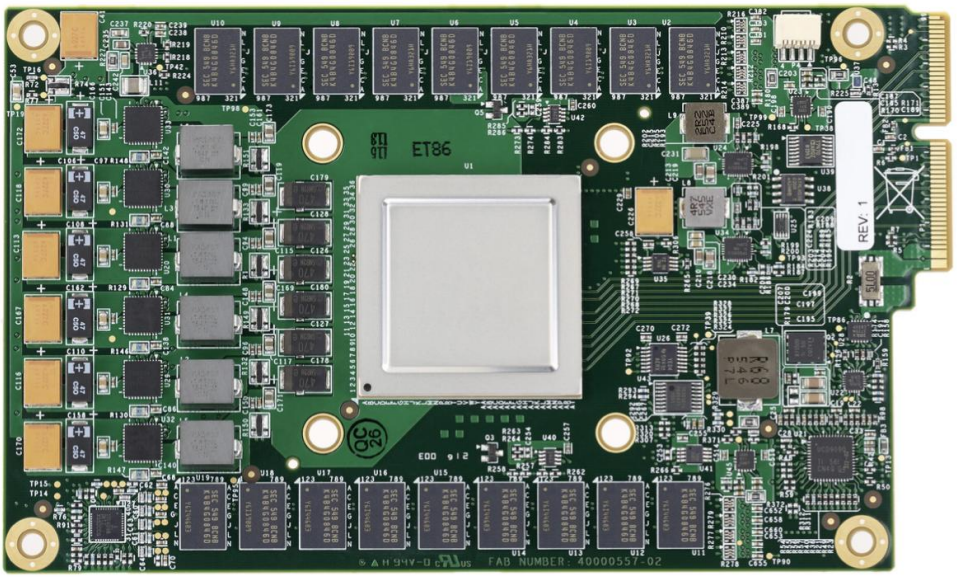


The Matrix Multiply Unit on the right hand side is the key part of the TPU. It can “perform 8-bit multiply-and-adds on signed or unsigned integers. The 16-bit products are collected in the 4 MiB of 32-bit Accumulators below the matrix unit. The matrix unit can produce 256-element partial sum per clock cycle”. The authors “pick 4096 by first noting that the operations per byte needed to reach peak performance (roofline knee in Section 4) is ~1350, so they rounded that up to 2048 and then duplicated it so that the compiler could use double buffering while running at peak performance.” Also the TPU matrix multiply unit is designed for dense matrices. “The weights for the matrix unit are staged through an on-chip Weight FIFO that reads from an off-chip 8 GiB DRAM called Weight Memory”. The floor plan of TPU die is shown in the figure below:



The PCIe Gen3 x16 bus sends TPU instructions to an instruction buffer. And there are five key instructions, including Read_Host_Memory, Read_Weights, MatrixMultiply/Convolve, Activate and Write_Host_Memory. Read_Host_Memory and Write_Host_Memory are to read and write data “from the CPU host memory into the unified buffer”. Read_Weights is to “read weights from weight memory into the Weight FIFO as input to the Matrix Unit”. MatrixMultiply/Convolve “causes the Matrix Unit to perform a matrix multiply or a convolution from the Unified Buffer into the Accumulators”. And Activate “performs the

nonlinear function of the artificial neuron, with options for ReLU, Sigmoid, and so on". The TPU printed circuit board is shown in the figure below:



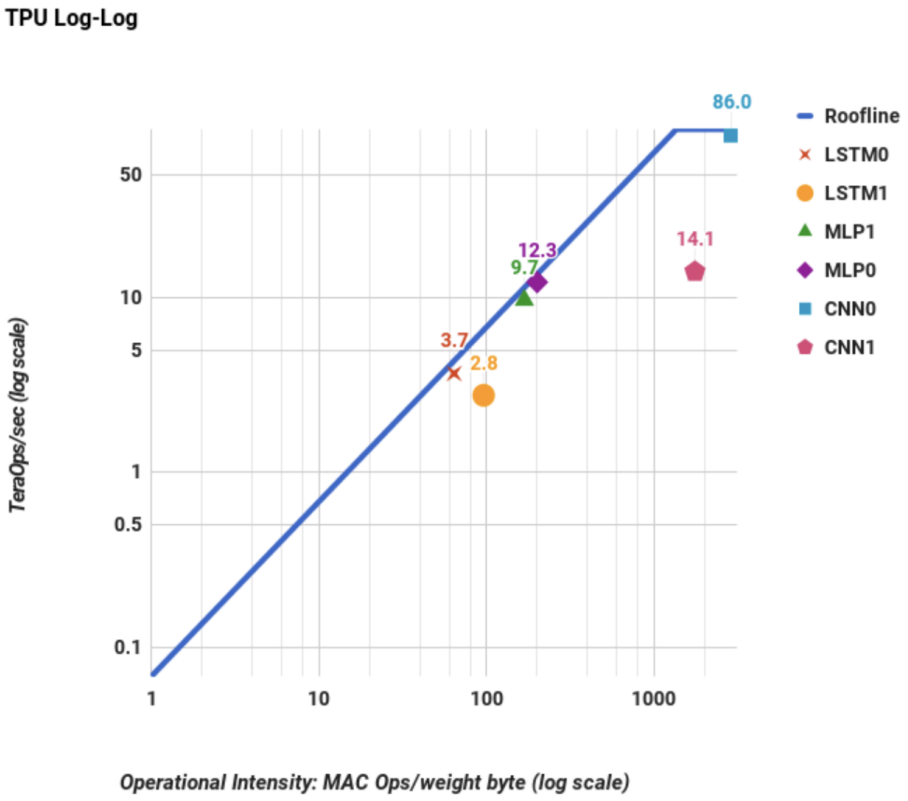
Limitations and Possible Improvements

The use of PCIe Gen3 x16 bus cause low memory bandwidth that limits the performance of TPU in 4 of the neural network tasks. They could improve bus or change to a better bus. TPU is only designed for one task of inference, while GPU or CPU can process multiple kinds of tasks. And the topic of the paper is limited to inference tasks. The TPU is limited in Tensorflow. They could make it adaptive for more libraries.

Summary of Class Discussion

- Q: How does the roofline graph look like this?
  A: For HPC, the Y-axis is floating-point operations per second (TeraOps/sec), which cause the "flat" part of the roofline. The X-axis is floating-point operations per DRAM

byte accessed. And since memory bandwidth is bytes per second, there is a “slanted” part in the roofline graph because (TeraOps/sec) / (TeraOps/Byte) = Bytes/sec.



- Q: Why does the matrix unit adopt Systolic execution?
  A: The TPU uses systolic execution to save energy by reducing reads and writes of the Unified Buffer.

Summary Serving DNNs in Real Time at Datacenter Scale with Project Brainwave

Runyu Zheng (runyuz), Shangquan Sun (sunsean)

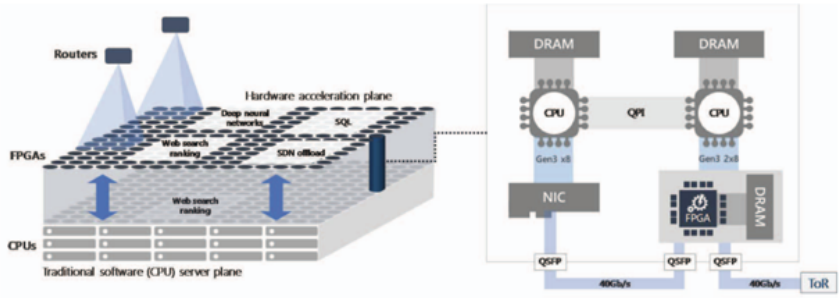
Problem and Motivation

- Cloud service providers deploy DNN-infused applications that ingest live data streams such as search queries and videos. For example, Bing’s intelligent search features (www.bing.com/explore/intelligentsearch (http://www.bing.com/explore/intelligentsearch)), leverage state-of-the-art machine reading comprehension models to analyze and understand billions of documents from the web to provide more relevant answers faster and directly to users.
- Computational demands of DNNs are outpacing performance growth of traditional CPUS. There is an explosion of processors like GPUs, FPGAs, and NPUs for AI workloads.
- There is a clear need for low latency and high throughput. Real-time latency constraints severely limit the size, complexity, and quality of such models that can be deployed on conventional hardware at datacenter scale. While throughput-oriented architectures such as GPGPUs and batch-oriented NPUs are popular for offline training and serving, they are not efficient for online, low-latency serving of DNN models.
- Specialized hardware can improve efficiency and performance

Hypothesis

FPGAs - Field-Programmable Gate Array

FPGA is an integrated circuit designed to be reprogrammable to implement different logic functions. With FPGA, one can exploit the model level parallelism targeted for a specific algorithm. Multiple FPGAs can be allocated as a single shared hardware microservice with no software in the loop.



Solution Overview

- The goal of Project Brainwave is to enable users without hardware expertise to automatically deploy and accelerate the serving of state-of-the-art DNN models in real time and at low cost.
- To meet real-time requirements on conventional hardware systems, such memory-intensive models must often be trimmed down in dimensionality and parameters, sacrificing quality and accuracy.
- To solve the challenges with serving memory-intensive DNNs, the Brainwave system exploits model parallelism and on-chip pinning at scale to achieve ultra-low latency serving of DNN models while preserving model accuracy.
- During offline compilation, the Brainwave tool flow splits DNN models into subgraphs, each of which can fit into on-chip FPGA memory or run on the CPU.

Brainwave Stack

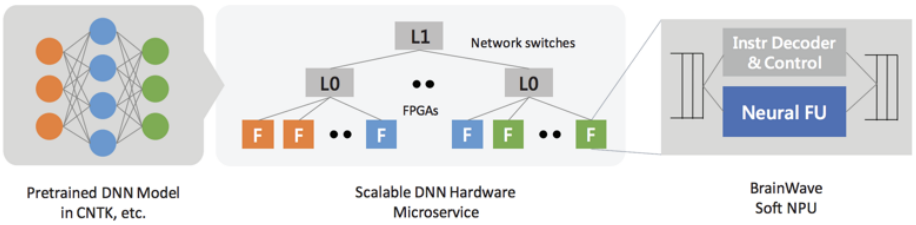


Figure 2. The three major layers of the Brainwave system: (left) a tool flow that converts pre-trained DNN models to a deployment package, (middle) a scalable DNN hardware microservice with network-attached FPGAs, and (right) the programmable Brainwave NPU hosted on FPGA.

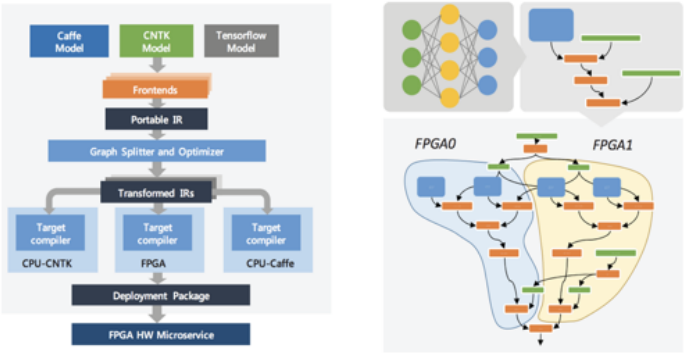


Figure 3. The framework-neutral Brainwave tool flow accepts models from different DNN toolchains and exports them into a common intermediate graph representation. Tool flow optimizes the intermediate representation and partitions it into sub-graphs assigned to different CPUs and FPGAs. Device-specific backends generate device assembly and are linked together by a federated runtime that gets deployed into a live FPGA hardware microservice.

The tool flow enables the user to partition the graph IR into different sub-graphs based on

the constraints imposed by the target system, such as the type of layer (CNN vs. RNN), available number of FPGAs in a hardware microservice, the supported set of operators that can be acceler- ated on FPGA, and the available on-chip memory capacity per FPGA:

- CNNs high intensity mapped to single FGPA.
- Weights of bandwidth-limited RNNs pinned greedily.
- Unsupported or not profitable operators assigned to CPUs.

Brainwave NPU

The Brainwave NPU is a parameterized soft vector processor at the heart of Brainwave system, with the following features:

- the use of compile-time narrow precision data types to extract higher performance than what is possible using conventional float and integer types without losses in accuracy;
- a simple, single-threaded programming model with an extensible ISA that can adapt to fast-changing DNN algorithms; and
- a scalable microarchitecture that maximizes hardware efficiency at low batch sizes.

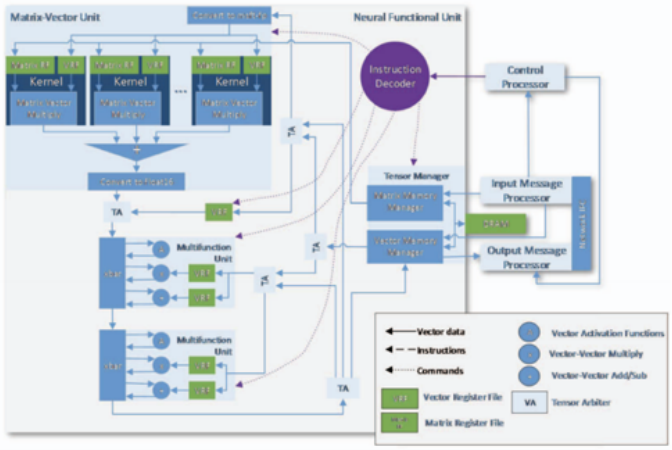


Figure 4. The Brainwave NPU is a "mega-SIMD" vector processor architecture. A sequentially programmed control processor asynchronously controls the neighboring NFU, optimized for fast DNN operations. The heart of the NFU is a dense matrix-vector multiplication unit (MVU) capable of processing single DNN requests at low batch with high utilization. The MVU is joined to secondary multifunctional units (MFUs) that perform element-wise vector-vector operations and activation functions.

Hardware Organization

- Control Processor: an off-the-shelf Nios embedded processor. Responsible for issuing CISC instruction to the decoupled NFU through an asynchronous instruction queue.
- NFU: consists of specialized functional units for accelerating common-case DNN operations.
- Nearby multi-ported vector register files: provide temporary storage between operations.
- MVU: the largest functional unit in the Brainwave NPU, receives encoded micro-ops from a distributed decoder that specifies op behavior (such as matrix size) and their dependences (such as source and target vector register).

Narrow Precision

Based on accuracy-sensitivity stud- ies of internal production and public DNN models (such as ResNet), the authors have developed proprietary "neural"-optimized data formats based on 8- and 9-bit floating point, where mantissas are trimmed to 2 or 3 bits. These formats, referred to as ms-fp8 and ms-fp9, exploit efficient packing into reconfigurable resources

and are comparable in FPGA area to low bit-width integer operations but can achieve higher dynamic range and accuracy than pure fixed-point implementations.

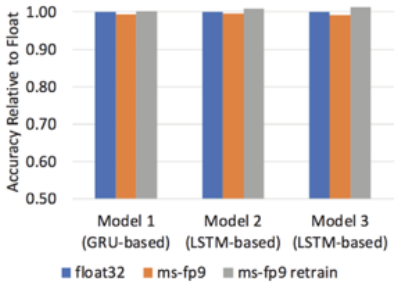


Figure 6. The impact to accuracy of various DNN production models is negligible with post-trained quantization into narrow precision ms-fp. With re-training, accuracy can be fully recovered.

Limitations and Possible Improvements

Microsoft successfully exploits FPGAs on a datacenter-scale fabric for real-time serving of state-of-the-art DNNs. In the furture, with the call for AI adaptive to live data, converged architectures for both low-latency inferencing and high-throughput training should be invented. The new architecture should also support state-of-art deep learning algorithm.

Summary of Class Discussion

- Q: Why using FPGA instead of CPU/GPU?
  A: FPGAs are more efficient than CPU/GPU, while also relatively flexible. It takes care of throughput and can exploit parallelism. It can achieve near-peak processing efficiencies at low batch sizes, which is important for real-time AI services. However, programming the FPGAs are difficult since both the hardware and software are moving parts. It could make debugging and fault tolerance hard.
- Q: How ms-fp8 and ms-fp9 stores floating point with fewer bits?
  A: I find in this link: https://en.wikichip.org/wiki/microsoft/msfp8 (https://en.wikichip.org/wiki/microsoft/msfp8)
  It follows standard IEEE754 single-precision floating-point, but truncates the mantissa field.