Introduction to Tensor Processing Unit

Lecture 5
August 25th, 2017

Jae W. Lee (jaewlee@snu.ac.kr)
Computer Science and Engineering
Seoul National University

Download this lecture slides at https://goo.gl/rJPMQU

Slide credits: David A. Patterson (Google Brain), "In-Data Center Performance Analysis of a Tensor Processing Unit™"
Disclaimer

- All of the following slides are taken from the following:
  - David Patterson, "Evaluation of the Tensor Processing Unit: A Deep Neural Network Accelerator for the Datacenter", *NAE Regional Meeting*, April 2017.
  - Original link: https://sites.google.com/view/naeregionalsymposium

- This work has been published at ISCA 2017.
In-Data Center Performance Analysis of a Tensor Processing Unit™ *

David Patterson and the Google TPU Team
davidpatterson@google.com

April 5, 2017

*4/5/17 Google published a blog on the TPU. A 17-page technical paper with same title will be on arXiv.org. (Paper will also appear at the International Symposium on Computer Architecture on June 26, 2017.)

A Golden Age in Microprocessor Design

- Stunning progress in microprocessor design 40 years $\approx 10^6 \times$ faster!

- Three architectural innovations ($\approx 1000x$)
  - Width: 8->16->32 ->64 bit ($\approx 8x$)
  - Instruction level parallelism:
    - 4-10 clock cycles per instruction to 4+ instructions per clock cycle ($\approx 10-20x$)
    - Multicore: 1 processor to 16 cores ($\approx 16x$)

- Clock rate: 3 to 4000 MHz ($\approx 1000x$ thru technology & architecture)

- Made possible by IC technology:
  - Moore’s Law: growth in transistor count (2X every 1.5 years)
  - Dennard Scaling: power/transistor shrinks at same rate as transistors are added (constant per mm$^2$ of silicon)

Changes Converge

• Technology
  • End of Dennard scaling: power becomes the key constraint
  • Slowdown (retirement) of Moore’s Law: transistors cost

• Architectural
  • Limitation and inefficiencies in exploiting instruction level parallelism end the uniprocessor era in 2004
  • Amdahl’s Law and its implications end “easy” multicore era

• Products
  • PC/Server ⇒ Client/Cloud

End of Growth of Performance?

40 years of Processor Performance

End of Moore’s Law ⇒ 2X / 20 yrs (3%/yr)

RISC 2X / 1.5 yrs (52%/yr)

CISC 2X / 3.5 yrs (22%/yr)

End of Dennard Scaling ⇒
Multicore 2X / 3.5 yrs (23%/yr)

Am- dahl’s Law ⇒ 2X / 6 yrs (12%/yr)

End of Moore’s Law ⇒ 2X / 20 yrs (3%/yr)

Based on SPECintCPU. Source: John Hennessy and David Patterson, Computer Architecture: A Quantitative Approach, 6/e. 2018
What’s Left?

Since

- Transistors not getting much better
- Power budget not getting much higher
- Already switched from 1 inefficient processor/chip to N efficient processors/chip

Only path left is *Domain Specific Architectures*

- Just do a few tasks, but extremely well
What is Deep Learning?

- Loosely based on (what little) we know about the brain

Slide from "Large-Scale Deep Learning with TensorFlow for Building Intelligent Systems," by Jeff Dean, ACM Webinar, 7/7/16
The Artificial Neuron

\[ y = F \left( \sum_i w_i x_i \right) \]

\[ F(x) = \max(0, x) \]

\[ F: \text{a nonlinear differentiable function} \]
IS THIS A CAT or DOG?

CAT DOG

OUTPUT LAYER

ACTIVATED NEURONS

INPUT LAYER

Deep Neural Network
Key NN Concepts for Architects

- *Training* or learning (development) vs. *Inference* or prediction (production)

- *Batch size*
  - Problem: DNNs have millions of weights that take a long time to load from memory (DRAM)
  - Solution: Large batch $\Rightarrow$ Amortize weight-fetch time by inferring (or training) many input examples at a time

- Floating-Point vs. Integer ("Quantization")
  - Training in Floating Point on GPUs popularized DNNs
  - Inferring in Integers faster, lower energy, smaller
● 2013: Prepare for success-disaster of new DNN apps
  ● Scenario with users speaking to phones 3 minutes per day:
    If only CPUs, need 2X-3X times whole fleet
  ● Unlike some hardware targets, DNNs applicable to a wide range of problems, so can reuse for solutions in speech, vision, language, translation, search ranking, ...

● Custom hardware to reduce the TCO of DNN inference phase by **10X** vs. GPUs
  ● Must run existing apps developed for CPUs and GPUs

● A very short development cycle
  ● Started project 2014, running in datacenter 15 months later:
    Architecture invention, compiler invention, hardware design, build, test, deploy

● Google CEO Sundar Pichai reveals Tensor Processing Unit at Google I/O on May 18, 2016 as “10X performance/Watt”
  cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html
● TPU Card to replace a disk
● Up to 4 cards / server
3 Types of NNs

1. Multilayer Perceptrons
   • Each new layer applies nonlinear function $F$ to weighted sum of all outputs from prior layer ("fully connected") $x_n = F(Wx_{n-1})$

2. Convolutional Neural Network
   • Like MLPs, but same weights used on nearby subsets of outputs from prior layer

3. Recurrent NN/"Long Short-Term Memory"
   • Each new layer a NL function of weighted sums of past state and prior outputs; same weights used across time steps
3 Types of NNs

- Convolutional Neural Network (CNN)
  - Based on convolution operations

Source: http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html
3 Types of NNs

- **Recurrent Neural Network (RNN)/LSTM**
  - Each layer is a function of both inputs from previous layers and past state.

Source: Deeplearning4j
## Inference Datacenter Workload (95%)

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>FC</th>
<th>Conv</th>
<th>Vector</th>
<th>Pool</th>
<th>Total</th>
<th>Nonlinear function</th>
<th>Weights</th>
<th>TPU Ops / Weight Byte</th>
<th>TPU Batch Size</th>
<th>% Deployed</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP0</td>
<td>0.1k</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>ReLU</td>
<td>20M</td>
<td>200</td>
<td>200</td>
<td>61%</td>
</tr>
<tr>
<td>MLP1</td>
<td>1k</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>ReLU</td>
<td>5M</td>
<td>168</td>
<td>168</td>
<td></td>
</tr>
<tr>
<td>LSTM0</td>
<td>1k</td>
<td>24</td>
<td>34</td>
<td></td>
<td></td>
<td>58</td>
<td>sigmoid, tanh</td>
<td>52M</td>
<td>64</td>
<td>64</td>
<td>29%</td>
</tr>
<tr>
<td>LSTM1</td>
<td>1.5k</td>
<td>37</td>
<td>19</td>
<td></td>
<td></td>
<td>56</td>
<td>sigmoid, tanh</td>
<td>34M</td>
<td>96</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>CNN0</td>
<td>1k</td>
<td>16</td>
<td></td>
<td></td>
<td>16</td>
<td></td>
<td>ReLU</td>
<td>8M</td>
<td>2888</td>
<td>8</td>
<td>5%</td>
</tr>
<tr>
<td>CNN1</td>
<td>1k</td>
<td>4</td>
<td>72</td>
<td>13</td>
<td></td>
<td>89</td>
<td>ReLU</td>
<td>100M</td>
<td>1750</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>
- Add as accelerators to existing servers
  - So connect over I/O bus ("PCIe")
  - TPU ≈ matrix accelerator on I/O bus
- Host server sends it instructions like a Floating Point Unit
  - Unlike GPU that fetches and executes own instructions
- The Matrix Unit: 65,536 (256x256) 8-bit multiply-accumulate units
- 700 MHz clock rate
- Peak: 92T operations/second
  - $65,536 \times 2 \times 700M$
- >25X as many MACs vs GPU
- >100X as many MACs vs CPU
- 4 MiB of on-chip Accumulator memory
- 24 MiB of on-chip Unified Buffer (activation memory)
- 3.5X as much on-chip memory vs GPU
- Two 2133MHz DDR3 DRAM channels
- 8 GiB of off-chip weight DRAM memory

**TPU: High-level Chip Architecture**
TPU: a Neural Network Accelerator Chip

Unified Buffer for Local Activations (96Kx256x8b = 24 MiB) 29% of chip

Matrix Multiply Unit (256x256x8b=64K MAC) 24%

Host Interf. 2%

Accumulators (4Kx256x32b = 4 MiB) 6%

Control 2%

Activation Pipeline 6%

PCle Interface 3%

Misc. I/O 1%

DRAM port ddr3 3%
• 5 main (CISC) instructions
  Read_Host_Memory
  Write_Host_Memory
  Read_Weights
  MatrixMultiply/Convolve
  Activate(ReLU, Sigmoid, Maxpool, LRN, ...)

• Average Clock cycles per instruction: >10
• 4-stage overlapped execution, 1 instruction type / stage
  • Execute other instructions while matrix multiplier busy
• Complexity in SW: No branches, in-order issue,
  SW controlled buffers, SW controlled pipeline synchronization
• Problem: energy/ time for repeated SRAM accesses of matrix multiply

• Solution: “Systolic Execution” to compute data on the fly in buffers by pipelining control and data
  • Relies on data from different directions arriving at cells in an array at regular intervals and being combined
Systolic Execution: Control and Data are pipelined
Systolic Execution

- Reuse input values
  - Read each input value once, but use it for many different operations
  - Wires only connect spatially adjacent ALUs -> area- and energy-efficient
  - Example: Multiplying an input vector by a weight matrix

Systolic Execution

- **Data flows in waves at matrix multiply unit**
  - Trade registers, control and operational flexibility for efficiency and much higher operation density.
  - Not well suited for general-purpose computation
  - Example: Multiplying an input matrix by a weight matrix

heig-vd/snu summer university 2017: how modern processors work?
Can now ignore pipelining in matrix
Pretend each 256B input read at once, & they instantly update 1 location of each of 256 accumulator RAMs.
## Relative Performance: 3 Contemporary Chips

<table>
<thead>
<tr>
<th>Processor</th>
<th>mm²</th>
<th>Clock MHz</th>
<th>TDP Watts</th>
<th>Idle Watts</th>
<th>Memory GB/sec</th>
<th>Peak TOPS/chip</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU: Haswell (18 core)</td>
<td>662</td>
<td>2300</td>
<td>145</td>
<td>41</td>
<td>51</td>
<td>2.6</td>
</tr>
<tr>
<td>GPU: Nvidia K80 (2 / card)</td>
<td>561</td>
<td>560</td>
<td>150</td>
<td>25</td>
<td>160</td>
<td>--</td>
</tr>
<tr>
<td>TPU</td>
<td>&lt;331*</td>
<td>700</td>
<td>75</td>
<td>28</td>
<td>34</td>
<td>91.8</td>
</tr>
</tbody>
</table>

*TPU is less than half die size of the Intel Haswell processor

K80 and TPU in 28 nm process; Haswell fabbed in Intel 22 nm process

These chips and platforms chosen for comparison because widely deployed in Google data centers
### GPUs and TPUs added to CPU server

<table>
<thead>
<tr>
<th>Processor</th>
<th>Chips/Server</th>
<th>DRAM</th>
<th>TDP Watts</th>
<th>Idle Watts</th>
<th>Observed Busy Watts in datacenter</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU: Haswell (18 cores)</td>
<td>2</td>
<td>256 GB</td>
<td>504</td>
<td>159</td>
<td>455</td>
</tr>
<tr>
<td>NVIDIA K80 (13 cores)</td>
<td>8</td>
<td>256 GB (host) + 12GB x 8</td>
<td>1838</td>
<td>357</td>
<td>991</td>
</tr>
<tr>
<td>TPU (1 core)</td>
<td>4</td>
<td>256GB (host) + 8GB x 4</td>
<td>861</td>
<td>290</td>
<td>384</td>
</tr>
</tbody>
</table>

These chips and platforms chosen for comparison because widely deployed in Google datacenters
2 Limits to performance:
1. Peak Computation
2. Peak Memory Bandwidth
   (For apps with large data that don’t fit in cache)

Arithmetic Intensity (FLOP/byte or reuse) determines which limit

Weight-reuse = Arithmetic Intensity for DNN roofline

Roofline Visual Performance Model

GFLOP/s = Min(Peak GFLOP/s, Peak GB/s x AI)
TPU Die Roofline

TeraOps/sec (log scale)

Operational Intensity: Ops/weight byte (log scale)
K80 (GPU) Die Roofline

- **Operational Intensity:** Ops/weight byte (log scale)
- **TeraOps/sec (log scale)**

The graph illustrates the roofline model for K80 GPUs, showing the relationship between operational intensity and TeraOps/sec under a log-log scale.
Why so far below Rooflines? (MLP0)

<table>
<thead>
<tr>
<th>Type</th>
<th>Batch</th>
<th>99th% Response</th>
<th>Inf/s (IPS)</th>
<th>% Max IPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>16</td>
<td>7.2 ms</td>
<td>5,482</td>
<td>42%</td>
</tr>
<tr>
<td>CPU</td>
<td>64</td>
<td>21.3 ms</td>
<td>13,194</td>
<td>100%</td>
</tr>
<tr>
<td>GPU</td>
<td>16</td>
<td>6.7 ms</td>
<td>13,461</td>
<td>37%</td>
</tr>
<tr>
<td>GPU</td>
<td>64</td>
<td>8.3 ms</td>
<td>36,465</td>
<td>100%</td>
</tr>
<tr>
<td>TPU</td>
<td>200</td>
<td>7.0 ms</td>
<td>225,000</td>
<td>80%</td>
</tr>
<tr>
<td>TPU</td>
<td>250</td>
<td>10.0 ms</td>
<td>280,000</td>
<td>100%</td>
</tr>
</tbody>
</table>
Log Rooflines for CPU, GPU, TPU

-Star = TPU
-Triangle = GPU
-Circle = CPU
Linear Rooflines for CPU, GPU, TPU

- Star = TPU
- Triangle = GPU
- Circle = CPU
# TPU & GPU Relative Performance to CPU

<table>
<thead>
<tr>
<th>Type</th>
<th>MLP 0</th>
<th>MLP 1</th>
<th>LSTM 0</th>
<th>LSTM 1</th>
<th>CNN 0</th>
<th>CNN 1</th>
<th>Weighted Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>2.5</td>
<td>0.3</td>
<td>0.4</td>
<td>1.2</td>
<td>1.6</td>
<td>2.7</td>
<td>1.9</td>
</tr>
<tr>
<td>TPU</td>
<td>41.0</td>
<td>18.5</td>
<td>3.5</td>
<td>1.2</td>
<td>40.3</td>
<td>71.0</td>
<td>29.2</td>
</tr>
<tr>
<td>Ratio</td>
<td>16.7</td>
<td>60.0</td>
<td>8.0</td>
<td>1.0</td>
<td>25.4</td>
<td>26.3</td>
<td>15.3</td>
</tr>
</tbody>
</table>
Perf/Watt TPU vs CPU & GPU

~80X incremental perf/W of Haswell CPU
~30X incremental perf/W of K80 GPU
• Current DRAM
  • 2 DDR3 2133 ⇒ 34 GB/s
• Replace with GDDR5 like in K80 ⇒ 180 GB/s
  • Move Ridge Point from 1400 to 256

Improving TPU: Move “Ridge Point” to the left
Revised TPU Raises Roofline

Improves performance 4X for LSTM1, LSTM0, MLP1, MLP0
Perf/Watt Original & Revised TPU

~200X incremental perf/W of Haswell CPU
~70X incremental perf/W of K80 GPU
Related Work

Two survey articles discuss the custom NN ASICs go back at least 25 years (Bartels et al.[6]). For example, CNNs chips contained 64 MIPs and were implemented in a 0.35μm CMOS technology. The system contained in it a neural-like learning algorithm that allowed it to adapt to various learning tasks.

Twelve diverse 16-bit W-synthesizers, commercialized by the 30 custom ASICs, were deployed starting in 1990 to do both NN training and inference research. The resulting products were the early days of the NN architecture, where a software system was to be designed to control an AI system. This early work was motivated, in part, by the need to create an architecture that could be used for both learning and inference.

The more recent DNNs family of NN architectures mimics human attention and accesses both the chip or direct interface to external DRAM via efficient architectural support for the memory access patterns that appear in NN applications (Ehrenberg et al.[13]). This design strategy delivers similar improvements to the state-of-the-art approaches when those are applied to CNNs as well as to DNNs. This elegant solution could produce up to 13.2 times higher performance than DNNs and 14× speedup on DNNs. The solutions are not new and the CPU's AVX instructions, and there is no reprogramming ever (see table 1) [Ehrenberg et al.].

Catastrophe is the most widely deployed example of a catastrophic failure to support DNNs, which many propose (Strub et al.[21]). The FPGA-based attack on the chip's DRAM interface, which creates a single malicious DNN that can attack the chip and cause it to fail. The catastrophic attack is used to show that the FPGAs are not only more effective than the SIMD processors, and within 2× to 3× of custom hardware designed just for the application.

The Falcon benchmark paper seems to report results contrary to our own, with the CPU running inference much faster than the FPGA. However, the CPU is not as resource-efficient as an SIMD processor, and within 2× to 3× of custom hardware designed just for the application.

Catastrophe is the most widely deployed example of a catastrophic failure to support DNNs, which many propose (Strub et al.[21]). The FPGA-based attack on the chip's DRAM interface, which creates a single malicious DNN that can attack the chip and cause it to fail. The catastrophic attack is used to show that the FPGAs are not only more effective than the SIMD processors, and within 2× to 3× of custom hardware designed just for the application.

As first introduced in 2014 (Nielson et al.[17]), a CPU is a CPU that simply adds one more dimension to the matrix of a neural network. As shown in table 1, a CPU cannot be used to implement a fully connected neural network. In its place, CPU's were used to implement a fully connected neural network. In its place, CPU's were used to implement a fully connected neural network. In its place, CPU's were used to implement a fully connected neural network.

Catastrophe is the most widely deployed example of a catastrophic failure to support DNNs, which many propose (Strub et al.[21]). The FPGA-based attack on the chip's DRAM interface, which creates a single malicious DNN that can attack the chip and cause it to fail. The catastrophic attack is used to show that the FPGAs are not only more effective than the SIMD processors, and within 2× to 3× of custom hardware designed just for the application.

As first introduced in 2014 (Nielson et al.[17]), a CPU is a CPU that simply adds one more dimension to the matrix of a neural network. As shown in table 1, a CPU cannot be used to implement a fully connected neural network. In its place, CPU's were used to implement a fully connected neural network. In its place, CPU's were used to implement a fully connected neural network. In its place, CPU's were used to implement a fully connected neural network.
TPU succeeded because of

- Large matrix multiply unit
- Substantial software-controlled on-chip memory
- Run whole inference models to reduce host CPU
- Single-threaded, deterministic execution model
- Good match to 99th-percentile response time
- Enough flexibility to match NNs of 2017 vs. 2013
- Omission of GP features ⇒ small, low power die
- Use of 8-bit integers in the quantized apps
- Apps in TensorFlow, so easy to port at speed
Conclusions (2/2)

- Inference prefers latency over throughput
- K80 GPU relatively poor at inference (vs. training)
- Small redesign improves TPU at low cost
- 15-month design & live on I/O bus yet TPU
  15X-30X faster Haswell CPU, K80 GPU (inference),
  <½ die size, ½ Watts
  - 65,536 (8-bit) TPU MACs cheaper, lower energy, &
    faster 576 (32-bit) CPU MACs, 2496 GPU (32-bit) MACs
- 10X difference in computer products are rare
Questions?

*4/5/17 Google published a blog on the TPU. A 17-page technical paper with same title will be on arXiv.org. (Paper will also appear at the *International Symposium on Computer Architecture* on June 26, 2017.)