Architecture of ML Systems
07 Hardware Accelerators

Matthias Boehm

Graz University of Technology, Austria
Computer Science and Biomedical Engineering
Institute of Interactive Systems and Data Science
BMVIT endowed chair for Data Management

Last update: May 08, 2020

Announcements/Org

- **#1 Video Recording**
  - Link in TeachCenter & TUbe (lectures will be public)
  - **Live streaming through TUbe**, starting May 08
  - Questions: https://tugraz.webex.com/meet/m.boehm
  - Online teaching extended until Jun 30; **exams via webex**

- **#2 AMLS Programming Projects**
  - **Status:** all project discussions w/ **15 students**
  - Awesome mix of projects (algorithms, compiler, runtime)
  - Email to m.boehm@tugraz.at if no project discussed yet
  - Soft deadline: **June 30**

- **#3 IBM Quantum Challenge**
  - 4 scored exercises on quantum circuits; until **May 8, 3pm**
  - https://quantum-computing.ibm.com/challenges/4anniversary

Categories of Execution Strategies

Batch **SIMD/SPMD**	Batch/Mini-batch, Independent Tasks **MIMD**	Mini-batch
05_a Data-Parallel Execution [Apr 03]	**05_b Task-Parallel Execution** [Apr 03]	**06 Parameter Servers** (data, model) [Apr 24]

07 Hybrid Execution and HW Accelerators [May 08]

---

08 Caching, Partitioning, Indexing, and Compression [May 15]

Agenda

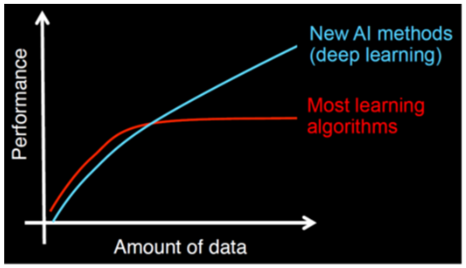
- **Motivation and Terminology**
- **GPUs in ML Systems**
- **FPGAs in ML Systems**
- **ASICs and other HW Accelerators**

Motivation and Terminology

Recap: Driving Factors for ML

- **Improved Algorithms and Models**
  - Success across data and application domains (e.g., health care, finance, transport, production)
  - More complex models which leverage large data

[**Credit:** Andrew Ng'14]



- **Availability of Large Data Collections**
  - Increasing automation and monitoring ➔ data (simplified by cloud computing & services)
  - Feedback loops, **data programming/augmentation**

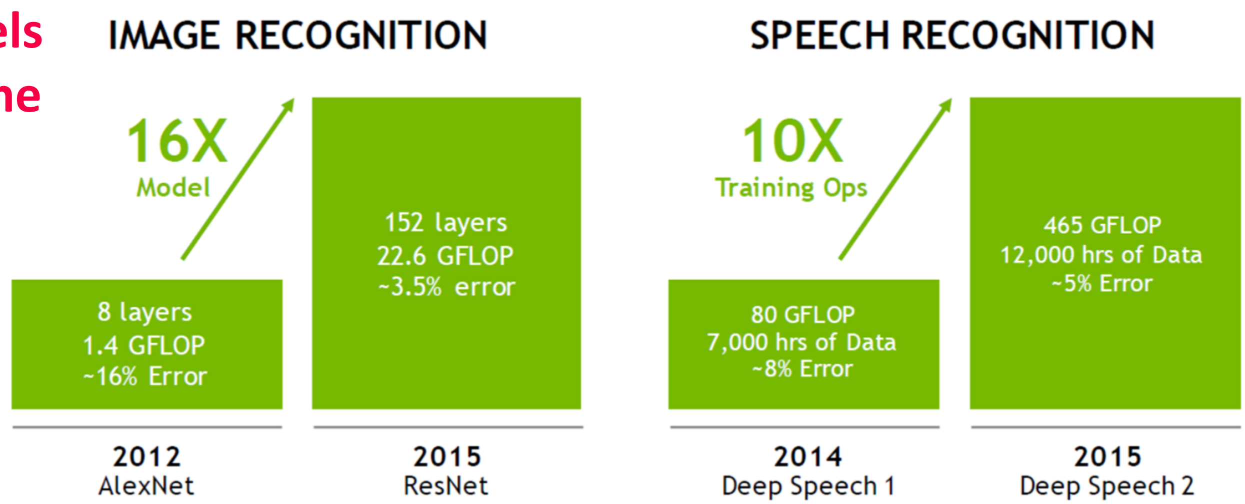
Feedback Loop

Data → Model → Usage → Data

- **HW & SW Advancements**
  - Higher performance of hardware and infrastructure (cloud)
  - Open-source large-scale computation frameworks, ML systems, and vendor-provides libraries

DNN Challenges

- **#1 Larger Models and Scoring Time**



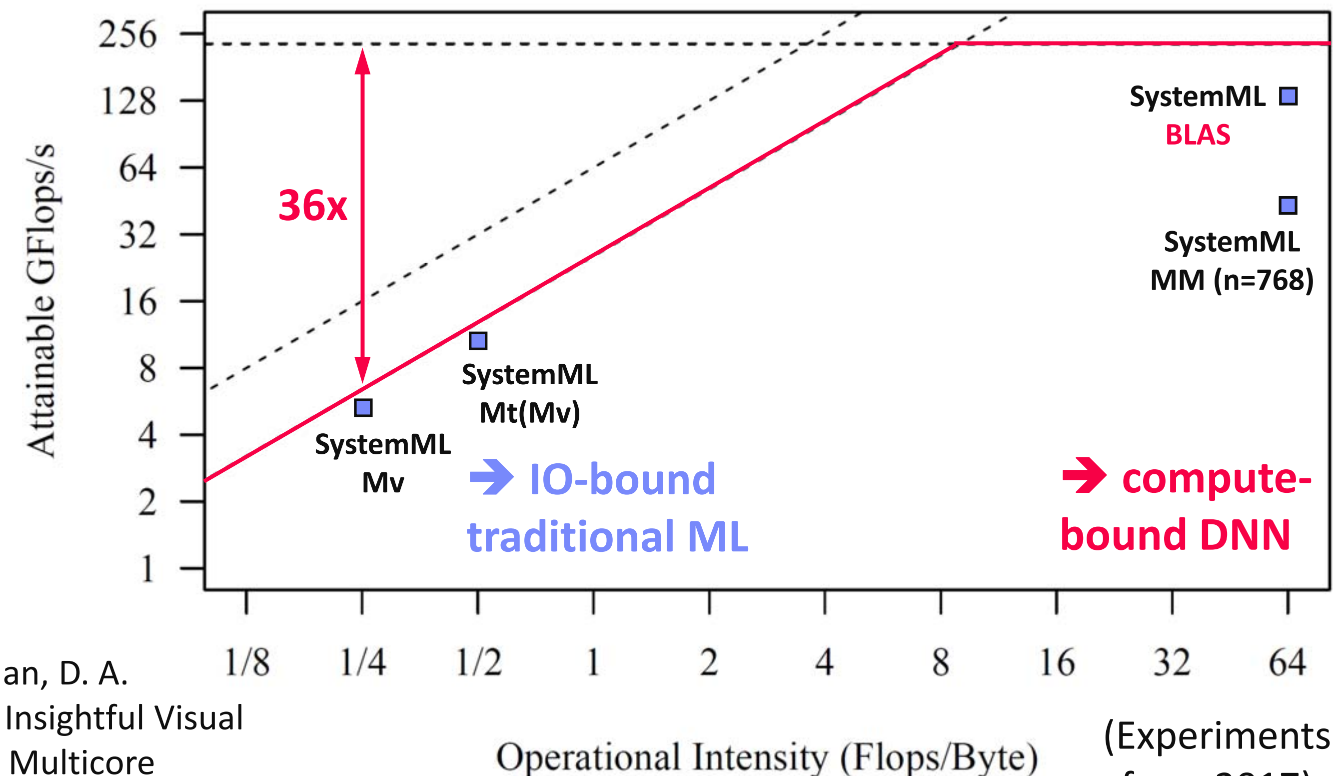
- **#2 Training Time**
  - **ResNet18:** 10.76% error, 2.5 days training
  - **ResNet50:** 7.02% error, 5 days training
  - **ResNet101:** 6.21% error, 1 week training
  - **ResNet152:** 6.16% error, **1.5 weeks training**

[Song Han: Efficient Methods and Hardware for Deep Learning, Stanford cs231n, 2017]

- **#3 Energy Efficiency**

Excursus: Roofline Analysis

- **Setup:** 2x6 E5-2440 @2.4GHz–2.9GHz, DDR3 RAM @1.3GHz (ECC)
  - Max mem bandwidth (local): 2 sock x 3 chan x 8B x 1.3G trans/s → **2 x 32GB/s**
  - Max mem bandwidth (QPI, full duplex) → **2 x 12.8GB/s**
  - Max floating point ops: 12 cores x 2*4dFP-units x 2.4GHz → **2 x 115.2GFlops/s**
- **Roofline Analysis**
  - Off-chip memory traffic
  - Peak compute



(Experiments from 2017)

[S. Williams, A. Waterman, D. A. Patterson: Roofline: An Insightful Visual Performance Model for Multicore Architectures. **Commun. ACM 2009**]

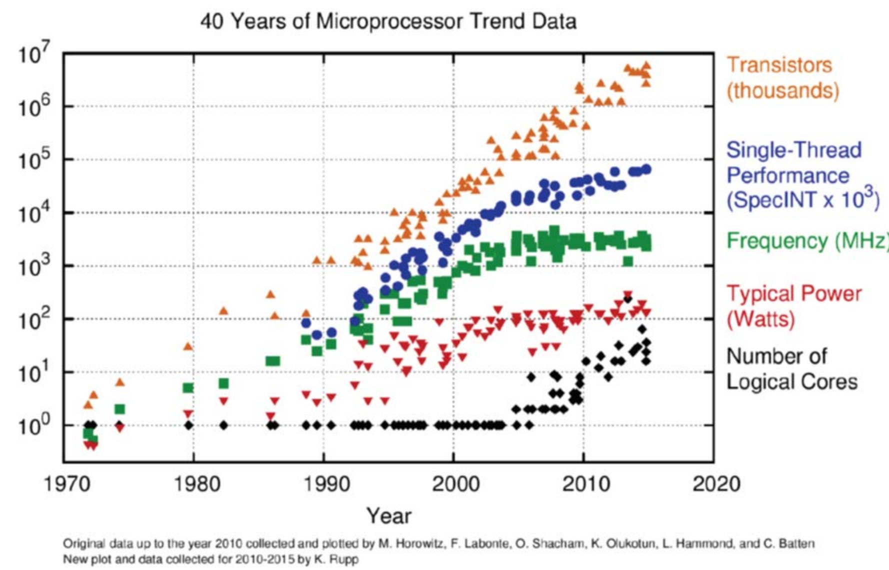
HW Challenges

[S. Markidis, E. Laure, N. Jansson, S. Rivas-Gomez and S. W. D. Chien: Moore's Law and Dennard Scaling]

- **#1 End of Dennard Scaling** (~2005)
  - Law: power stays proportional to the area of the transistor
  - Ignored leakage current / threshold voltage
    → **increasing power density S^2** (power wall, heat) → stagnating frequency

$P = \alpha\ CFV^2$ (**power density 1**)
(P .. Power, C .. Capacity, F .. Frequency, V .. Voltage)

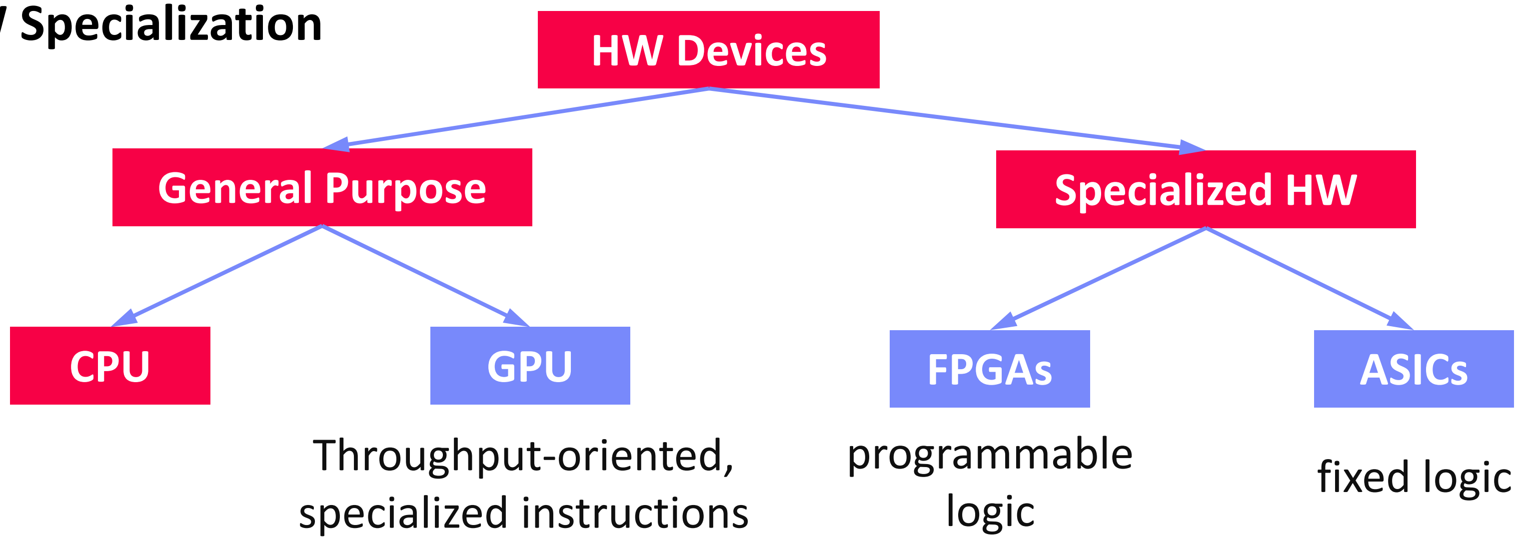
- **#2 End of Moore's Law** (~2010-20)
  - Law: #transistors/performance/ CPU frequency doubles every 18/24 months
  - Original: # transistors per chip doubles every two years **at constant costs**
  - Now increasing costs



→ **Consequences: Dark Silicon and Specialization**

Towards Specialized Hardware

- **HW Specialization**



- **Additional Specialization**
  - **Data Transfer & Types:** e.g., low-precision, quantization
  - **Sparsity Exploitation:** e.g., sparsification, defer weight decompression just before instruction execution
  - **Near-Data Processing:** e.g., operations in main memory, storage class memory (SCM), secondary storage (e.g., SSDs), and tertiary storage (e.g., tapes)

08 Caching, Indexing and Compression

Graphics Processing Units (GPUs) in ML Systems

NVIDIA Volta V100 – Specifications

- **Tesla V100 NVLink**
  - FP64: **7.8 TFLOPs**, FP32: **15.7 TFLOPs**
  - DL FP16: **125 TFLOPs**
  - NVLink: 300GB/s
  - Device HBM: 32 GB (**900 GB/s**)
  - Power: 300 W

- **Tesla V100 PCIe**
  - FP64: 7 TFLOPs, FP32: 14 TFLOPs
  - DL FP16: 112 TFLOPs
  - PCIe: 32 GB/s
  - Device HBM: 16 GB (900 GB/s)
  - Power: **250 W**

[Credit: https://nvidia.com/de-de/
data-center/tesla-v100/]

NVIDIA Volta V100 – Architecture

- **6 GPU Processing Clusters (GPCs)**
  - 7 Texture Processing Clusters (TPC)
  - 14 Streaming Multiprocessors (SM)

[NVIDIA Tesla V100 GPU Architecture, Whitepaper, **Aug 2017**]

NVIDIA Volta V100 – SM Architecture

- **FP64 cores: 32**
- **FP32 cores: 64**
- **INT32 cores: 64**
- **“Tensor cores”: 8**
- **Max warps /SM: 64**
- **Threads/warp: 32**

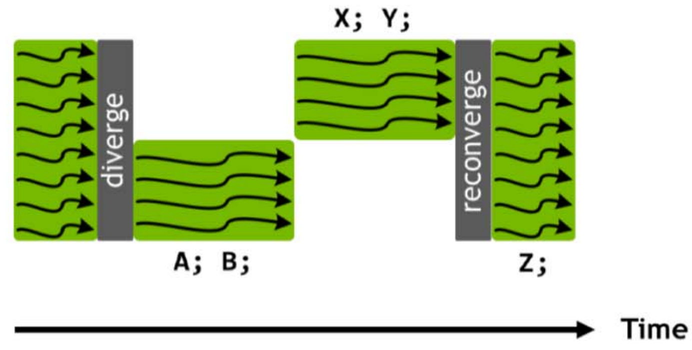
Single Instruction Multiple Threads (SIMT)

- **32 Threads grouped to warps and execute in SIMT model**

- **Pascal P100 Execution Model**
  - Warps use a single program counter + active mask

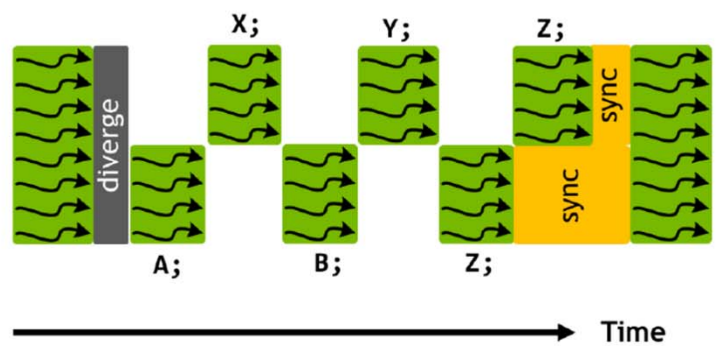
```
if (threadIdx.x < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```

Thread Divergence



- **Volta V100 Execution Model**
  - Independent thread scheduling
  - Per-thread program counters and call stacks

```
if (threadIdx.x < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
__syncwarp()
```



  - New **`__syncwarp()`** primitive (if needed) + **convergence optimizer**

NVIDIA Volta V100 – Tensor Cores

- **“Tensor Core”**
  - Specialized instruction for **4x4 by 4x4 fused matrix multiply**
  - Two FP16 inputs and FP32 accumulator
  - Exposed as warp-level matrix operations w/ special load, mm, acc, and store

[Bill Dally: Hardware for Deep Learning. **SysML 2018**]

64 FMA operations

D = A %*% B + C

$$\mathbf{D} = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

FP16 or FP32 | FP16 | FP16 | FP16 or FP32

Excursus: Amdahl's Law

- **Amdahl's law**
  - Given a fixed problem size, **Amdahl's law gives the maximum speedup**
  - T is the execution time, **s is the serial fraction**, and p the number of processors

Execution Time $$T_p = \frac{(1-s)T}{p} + sT$$

Speedup $$S_p = \frac{T}{T_p}$$

Upper-Bound Speedup $$\overline{S_p} = \lim_{p \to \infty} S_p = \frac{1}{s}$$

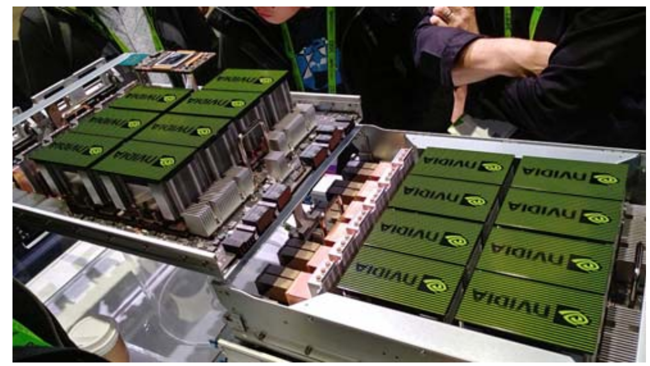
- **Examples**
  - Serial fraction s = 0.01 → max S_p = 100
  - Serial fraction s = **0.05** → max S_p = **20**
  - Serial fraction s = **0.1** → max S_p = **10**
  - Serial fraction s = 0.5 → max S_p = 2

GPUs for DNN Training

- **GPUs for DNN Training** (2009)
  - Deep belief networks
  - Sparse coding

[Rajat Raina, Anand Madhavan, Andrew Y. Ng: Large-scale deep unsupervised learning using graphics processors. **ICML 2009**]

- **Multi-GPU Learning** (Now)
  - Exploit multiple GPUs with a mix of **data- and model-parallel parameter servers**
  - Dedicated ML systems for multi-GPU learning
  - Dedicated HW: e.g., NVIDIA DGX-1 (8xP100), **NVIDIA DGX-2 (16xV100, NVSwitch)**



- **DNN Framework support**
  - All specialized DNN frameworks have very good support for GPU training
  - Most of them also support multi-GPU training

Recap: DNN Benchmarks

[**MLPerf** v0.6: https://mlperf.org/training-results-0-6/]

Closed Division Times

#	Submitter	System	Processor	#	Accelerator	#	Software	Benchmark results (minutes) Image classification ImageNet ResNet-50 v1.5	Object detection, light-weight COCO SSD w/ ResNet-34	Object detection, heavy-wt. COCO Mask-R-CNN	Translation, recurrent WMT E-G NMT	Translation, non-recur. WMT E-G Transformer	Recom-mendation MovieLens-20M NCF	Reinforcement Learning Go Mini Go	Details	Code	Notes
Available in cloud																	
0.6-1	Google	TPUv3.32			TPUv3	16	TensorFlow, TPU 1.14.1.dev	42.19	12.61	107.03	12.25	10.20	[1]		details	code	none
0.6-2	Google	TPUv3.128			TPUv3	64	TensorFlow, TPU 1.14.1.dev	11.22	3.89	57.46	4.62	3.85	[1]		details	code	none
0.6-3	Google	TPUv3.256			TPUv3	128	TensorFlow, TPU 1.14.1.dev	6.86	2.76	35.60	3.53	2.81	[1]		details	code	none
0.6-4	Google	TPUv3.512			TPUv3	256	TensorFlow, TPU 1.14.1.dev	3.85	1.79		2.51	1.58	[1]		details	code	none
0.6-5	Google	TPUv3.1024			TPUv3	512	TensorFlow, TPU 1.14.1.dev	2.27	1.34		2.11	1.05	[1]		details	code	none
0.6-6	Google	TPUv3.2048			TPUv3	1024	TensorFlow, TPU 1.14.1.dev	1.28	1.21			0.85	[1]		details	code	none
Available on-premise																	
0.6-7	Intel	32x 2S CLX 8260L	CLX 8260L	64			TensorFlow						[1]	14.43	details	code	none
0.6-8	NVIDIA	DGX-1			Tesla V100	8	MXNet, NGC19.05	115.22					[1]		details	code	none
0.6-9	NVIDIA	DGX-1			Tesla V100	8	PyTorch, NGC19.05		22.36	207.48	20.55	20.34	[1]		details	code	none
0.6-10	NVIDIA	DGX-1			Tesla V100	8	TensorFlow, NGC19.05						[1]	27.39	details	code	none
0.6-11	NVIDIA	3x DGX-1			Tesla V100	24	TensorFlow, NGC19.05						[1]	13.57	details	code	none
0.6-12	NVIDIA	24x DGX-1			Tesla V100	192	PyTorch, NGC19.05			22.03			[1]		details	code	none
0.6-13	NVIDIA	30x DGX-1			Tesla V100	240	PyTorch, NGC19.05		2.67				[1]		details	code	none
0.6-14	NVIDIA	48x DGX-1			Tesla V100	384	PyTorch, NGC19.05				1.99		[1]		details	code	none
0.6-15	NVIDIA	60x DGX-1			Tesla V100	480	PyTorch, NGC19.05					2.05	[1]		details	code	none
0.6-16	NVIDIA	130x DGX-1			Tesla V100	1040	MXNet, NGC19.05	1.69					[1]		details	code	none
0.6-17	NVIDIA	DGX-2			Tesla V100	16	MXNet, NGC19.05	57.87									
0.6-18	NVIDIA	DGX-2			Tesla V100	16	PyTorch, NGC19.05		12.21	101.00	10.94	11.04					
0.6-19	NVIDIA	DGX-2H			Tesla V100	16	MXNet, NGC19.05	52.74									
0.6-20	NVIDIA	DGX-2H			Tesla V100	16	PyTorch, NGC19.05		11.41	95.20	9.87	9.80					
0.6-21	NVIDIA	4x DGX-2H			Tesla V100	64	PyTorch, NGC19.05		4.78	32.72							
0.6-22	NVIDIA	10x DGX-2H			Tesla V100	160	PyTorch, NGC19.05					2.41					
0.6-23	NVIDIA	12x DGX-2H			Tesla V100	192	PyTorch, NGC19.05			18.47							
0.6-24	NVIDIA	15x DGX-2H			Tesla V100	240	PyTorch, NGC19.05		2.56								
0.6-25	NVIDIA	16x DGX-2H			Tesla V100	256	PyTorch, NGC19.05				2.12						
0.6-26	NVIDIA	24x DGX-2H			Tesla V100	384	PyTorch, NGC19.05				1.80						
0.6-27	NVIDIA	30x DGX-2H, 8 chips each			Tesla V100	240	PyTorch, NGC19.05		2.23								
0.6-28	NVIDIA	30x DGX-2H			Tesla V100	480	PyTorch, NGC19.05					1.59					
0.6-29	NVIDIA	32x DGX-2H			Tesla V100	512	MXNet, NGC19.05	2.59									
0.6-30	NVIDIA	96x DGX-2H			Tesla V100	1536	MXNet, NGC19.05	1.33									

96 x DGX-2H = 96 * 16 = 1536 V100 GPUs

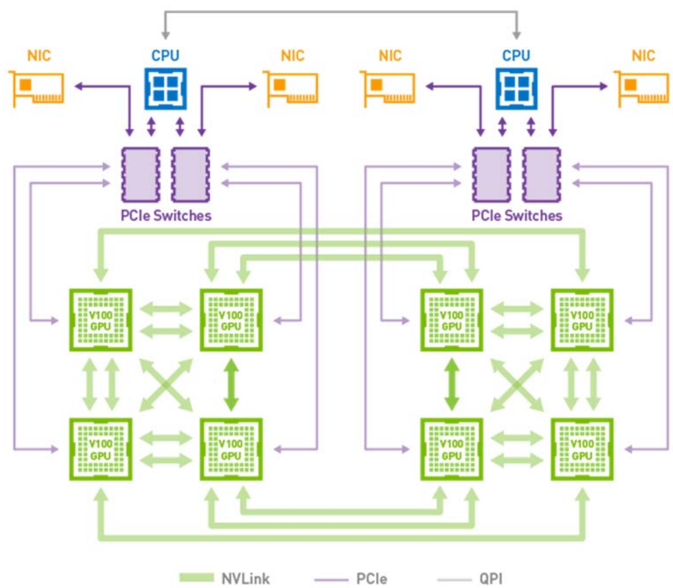
➔ ~ 96 * $400K = **$35M – $40M**

[https://www.forbes.com/sites/tiriasresearch/2019/06/19/nvidia-offers-a-turnkey-supercomputer-the-dgx-superpod/#693400f43ee5]

GPU Link Technologies

- **Classic PCI Express**
  - Peripheral Component Interconnect Express (default)
  - **v3 x16 lanes: 16GB/s**, v4 (2017) x16 lanes: 32GB/s, v5 (2019) x16 lanes: 64GB/s

- **#1 NVLink**
  - Proprietary technology
  - Requires NVLink-enabled CPU (e.g., IBM Power 8/9)
  - Connect GPU-GPU and GPU-CPU
  - NVLink 1: 80+80 GB/s
  - NVLink 2: 150+150 GB/s



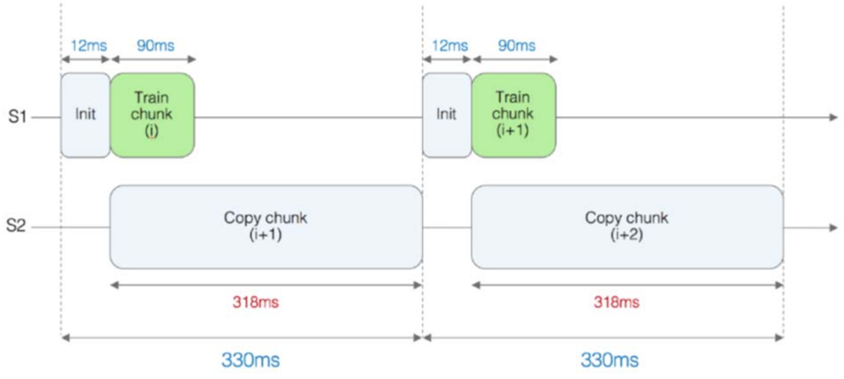
- **#1 NVSwitch**
  - Fully connected GPUs, each communicating at 300GB/s

GPU Link Technologies, cont.

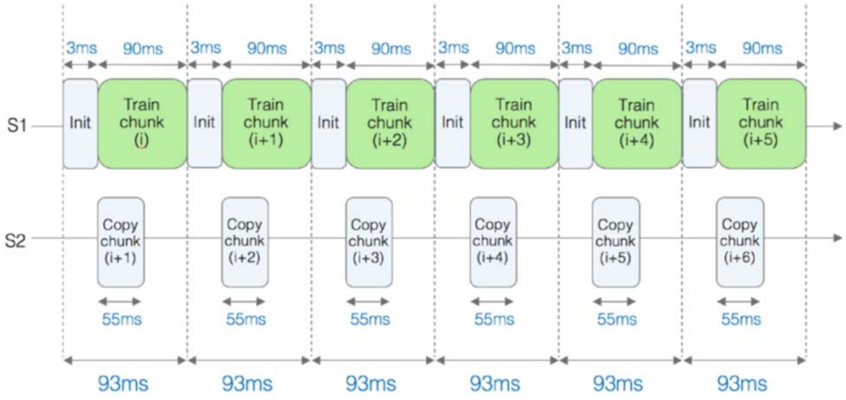
- **Recap: Amdahl's Law**
- **Experimental Setup**
  - **SnapML**, 4 IBM Power x 4 V100 GPUs, NVLink 2.0
  - 200 million training examples of the Criteo dataset (> GPU mem)
  - Train a logistic regression model

[Celestine Dünner et al.: Snap ML: A Hierarchical Framework for Machine Learning. **NeurIPS 2018**]

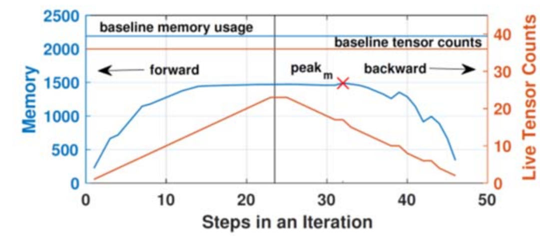
PCIe v3 Interconnect



NVLink Interconnect



Handling Memory Constraints



[Linnan Wang et al: Superneurons: dynamic GPU memory management for training deep neural networks. **PPOPP 2018**]

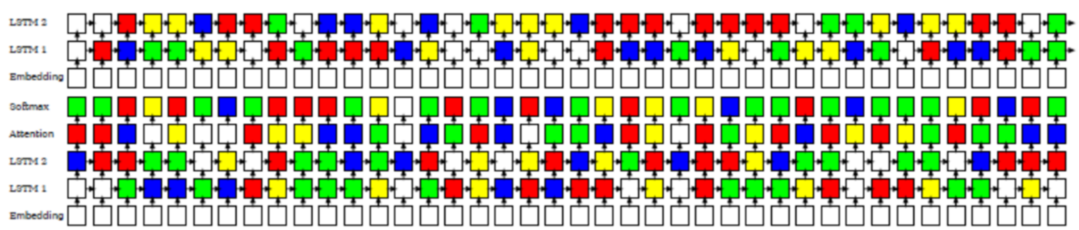
- **Problem: Limited Device Memory**
  - Large models and activations during training
- **#1 Live Variable Analysis**
  - Remove intermediates that are no longer needed
  - Examples: SystemML, TensorFlow, MXNet, Superneurons
- **#2 GPU-CPU Eviction**
  - Evict variables from GPU to CPU memory under memory pressure
  - Examples: SystemML, Superneurons, GeePS, (TensorFlow)
- **#3 Recomputation**
  - Recompute inexpensive operations (e.g., activations of forward pass)
  - Examples: MXNet, Superneurons
- **#4 Reuse Allocations**
  - Reuse allocated matrices and tensors via free lists, but **fragmentation**
  - Examples: SystemML, Superneurons

Hybrid CPU/GPU Execution

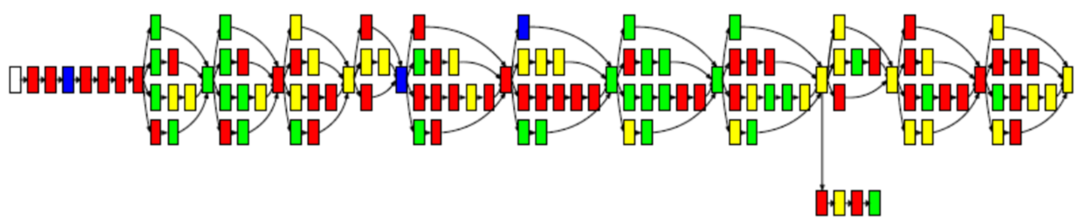
- **Manual Placement**
  - Most DNN frameworks allow manual placement of variables and operations on individual CPU/GPU devices
  - **Heuristics and intuition of human experts**
- **Automatic Placement**
  - Sequence-to-sequence model to predict which operations should run on which device
  - Examples:

[Azalia Mirhoseini et al: Device Placement Optimization with Reinforcement Learning. **ICML 2017**]

Neural MT graph



Inception V3



Sparsity in DNN

- **State-of-the-art**
  - **Very limited support of sparse tensors** in TensorFlow, PyTorch, etc
  - GPU operations for basic linear algebra (**cuSparse**), early support in ASICs
  - Research on specific operations and code generation
  - Problem: **Irregular structures of sparse matrices/tensors**

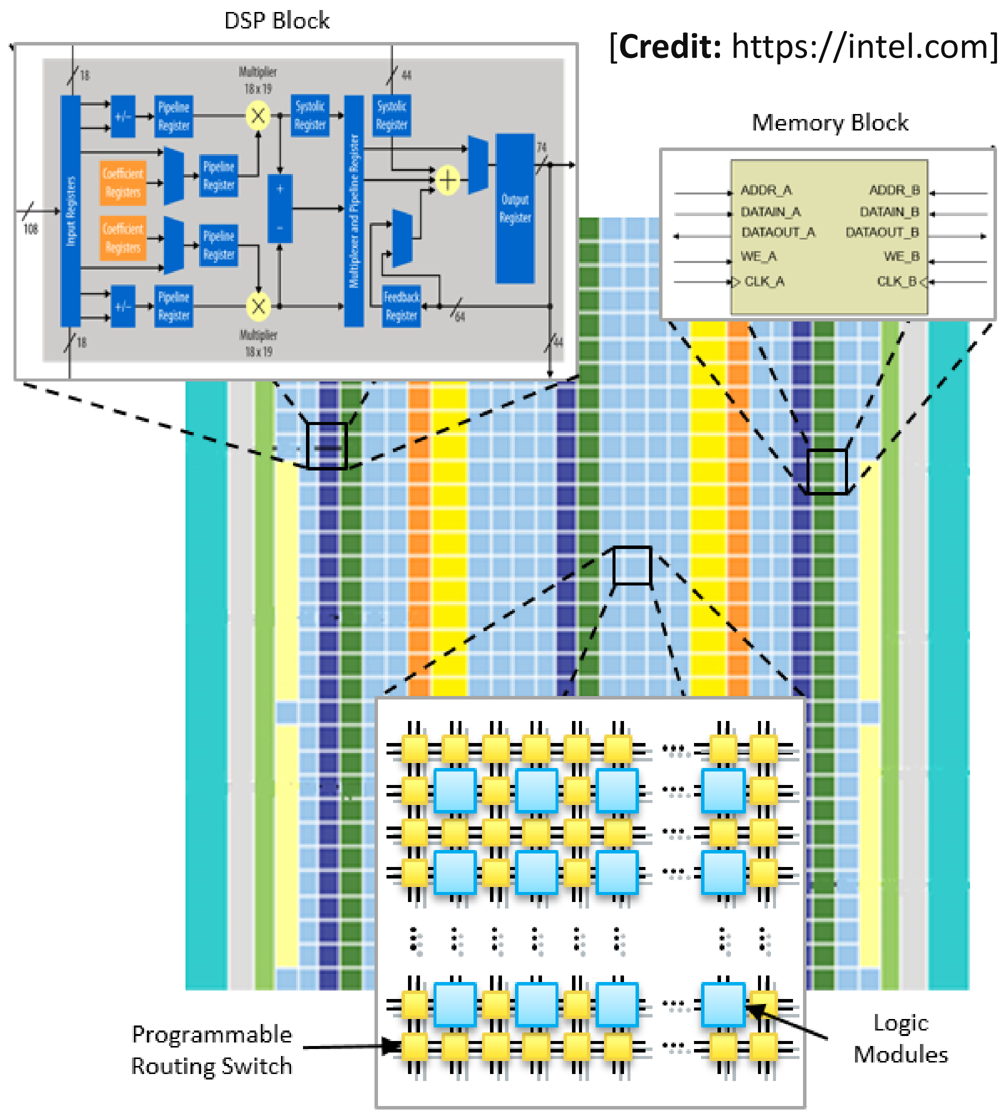
- **Common Techniques**
  - #1: **Blocking/clustering** of rows/columns by number of non-zeros
  - #2: **Padding rows/columns** to common number of non-zeros

- **Open Problem**
  - Many sources of sparsity (inputs, transformations, selections)
  - Broader support for **efficient sparsity exploitation** required

Field-Programmable Gate Arrays (FPGAs) in ML Systems

FPGA Overview

[**Credit:** https://intel.com]



- **FPGA Definition**
  - Integrated circuit that allows **configuring custom hardware designs**
  - Reconfiguration in <1s
  - HW description language: e.g.., VHDL, Verilog
- **FPGA Components**
  - **#1 lookup table** (LUT) as logic gates
  - **#2 flip-flops** (registers)
  - **#3 interconnect network**
  - Additional memory and DSP blocks

Example FPGA Characteristics

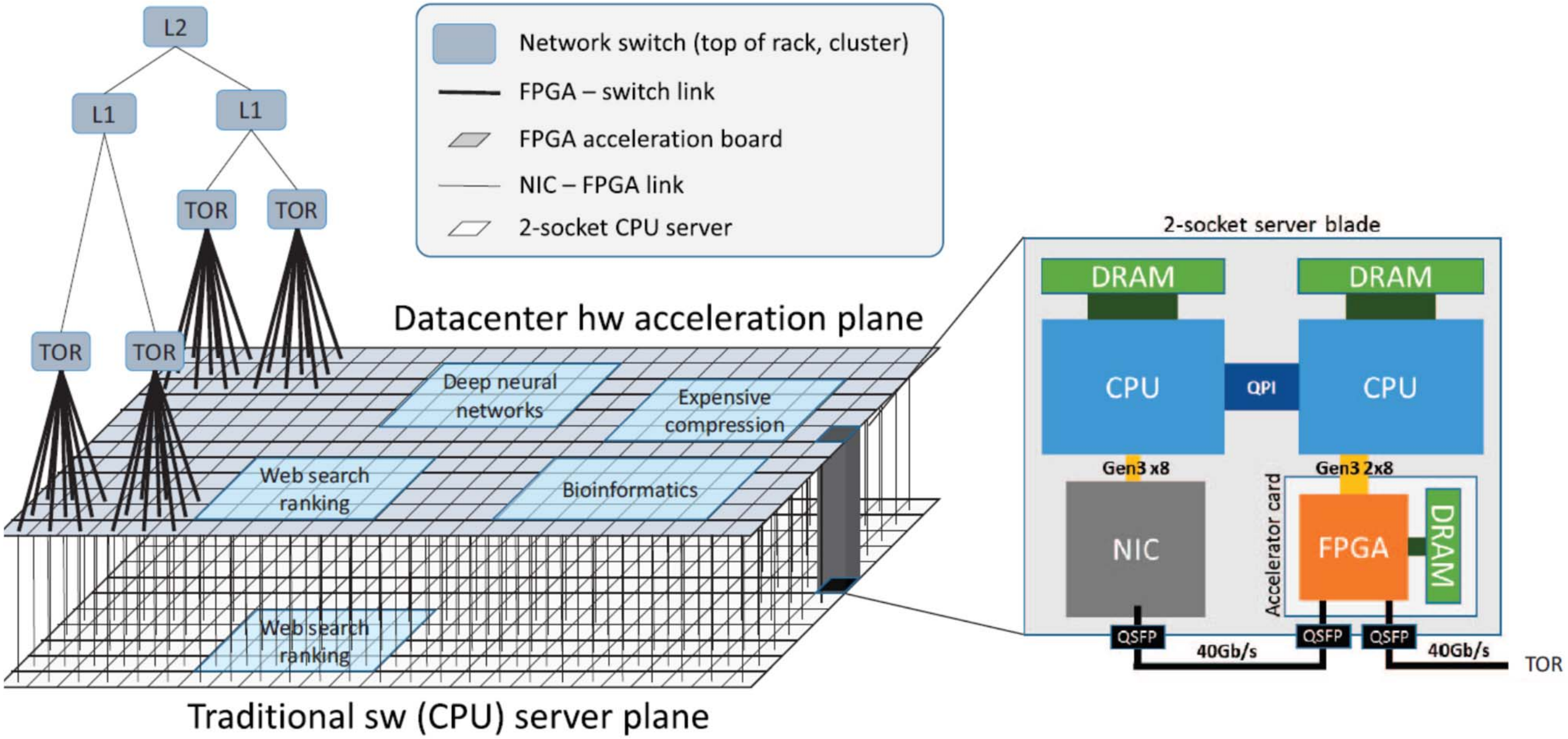
- **Intel (Altera) Stratix 10 SoC FPGA**
  - 64bit quad-core ARM
  - 10 TFLOPs FP32
  - 80GFLOPs/W
  - Other configurations w/ HBM2

- **Xilinx Virtex UltraSCALE+**
  - DSP: 21.2 TMACs
  - 64MB on-chip memory
  - 8GB HBM2 w/ 460GB/s

FPGAs in Microsoft's Data Centers

- **Microsoft Catapult**
  - Dual-socket Xeon w/ PCIe-attached FPGA
  - Pre-filtering neural networks, compression, and other workloads

[Adrian M. Caulfield et al.: A cloud-scale acceleration architecture. **MICRO 2016**]

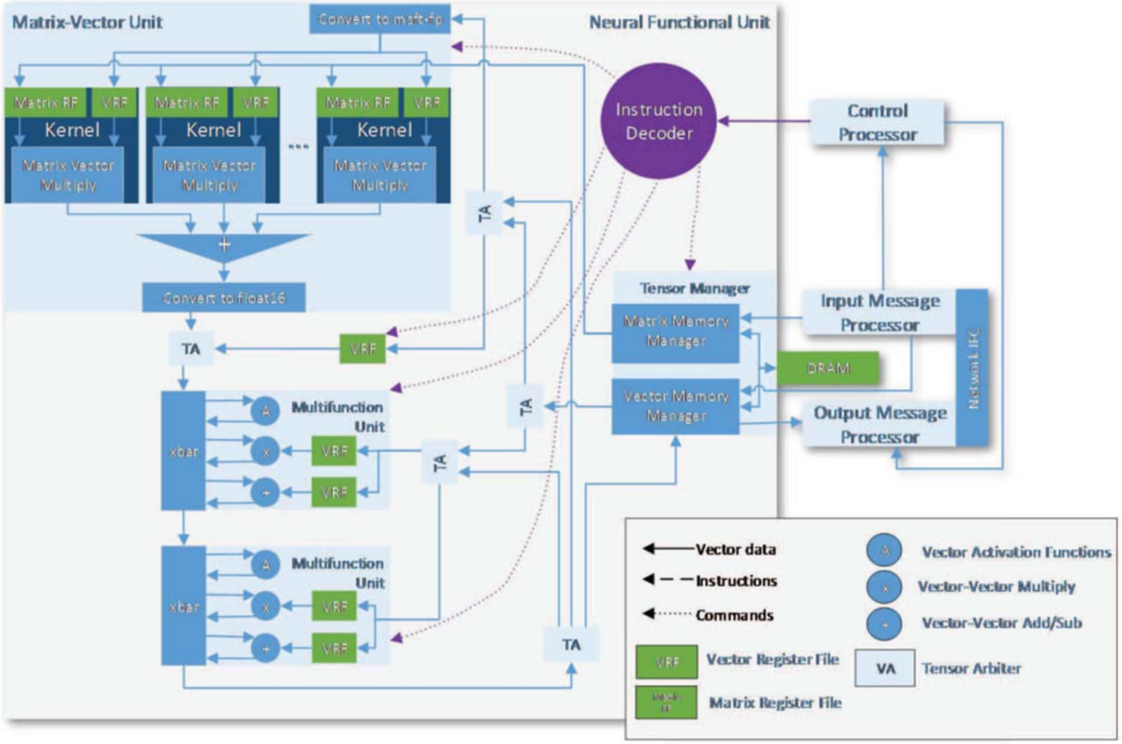


FPGAs in Microsoft's Data Centers, cont.

- **Microsoft Brainwave**
  - ML serving w/ low latency (e.g., Bing)
  - Intel Stratix 10 FPGA
  - Distributed model parallelism, precision-adaptable
  - Peak 39.5 TFLOPs

[Eric S. Chung et al: Serving DNNs in Real Time at Datacenter Scale with Project Brainwave. **IEEE Micro 2018**]

- **Brainwave NPU**
  - Neural processing unit
  - Dense matrix-vector multiplication



FPGAs in other ML Systems

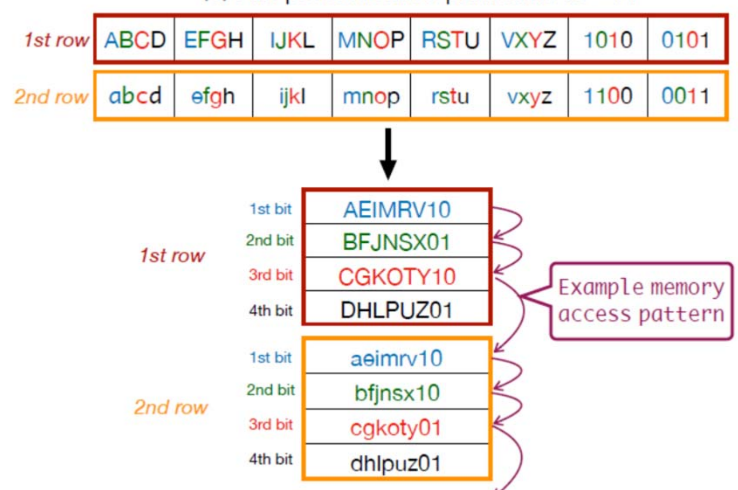
- **In-DB Acceleration of Advanced Analytics (DAnA)**
  - Compilation of python DSL into micro instructions for multi-threaded FPGA-execution engine
  - Striders to directly **interact with the buffer pool**

[Divya Mahajan et al: In-RDBMS Hardware Acceleration of Advanced Analytics. **PVLDB 2018**]

- **MLWeaving**
  - Adapted **BitWeaving** to numeric matrices
  - Data layout basis for **Any-Precision Learning**
  - Related FPGA implementation of SGD, matrix-vector multiplication for GLM
  - **Manual Selection** + Heuristics

[Zeke Wang et al: Accelerating Generalized Linear Models with MLWeaving. **PVLDB 2019**]

(a) Full-precision fixed-point table TS = T4



(b) BWeaving memory layout

- **Efficient FPGA implementations of specific operations and algorithms**
- **Specialized neural network topologies**

Application-Specific Integrated Circuit (ASICs) and other HW Accelerators

Overview ASICs

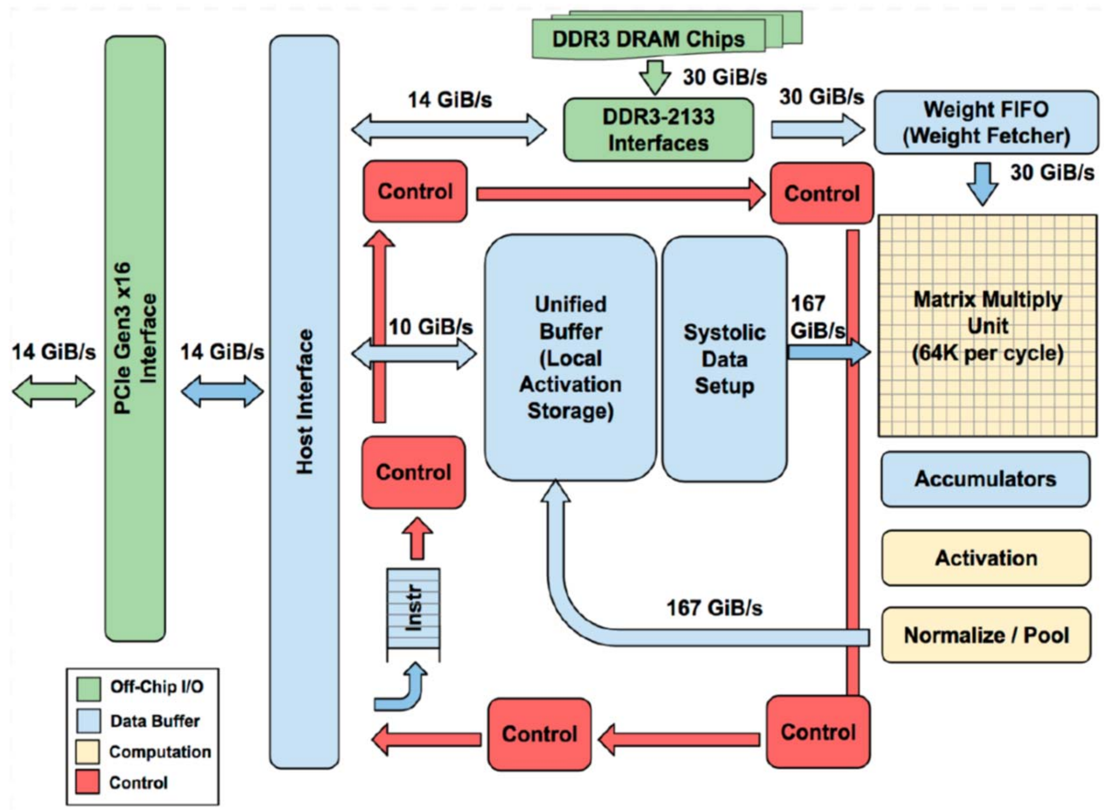
- **Motivation**
  - Additional improvements of performance, power/energy
  - ➔ **Additional specialization via custom hardware**
- **#1 General ASIC DL Accelerators**
  - HW support for matrix multiply, convolution and activation functions
  - Examples: **Google TPU**, **NVIDIA DLA** (in NVIDIA Xavier SoC), **Intel Nervana NNP**
- **#2 Specialized ASIC Accelerators**
  - Custom instructions for specific domains such as computer vision
  - Example: (Cadence) **Tensilica Vision processor** (image processing)
- **#3 Other Accelerators/Technologies** (some skepticism)
  - a) **Neuromorphic computing / spiking neural networks**
    (e.g., SyNAPSE → IBM TrueNorth, HP memristor for computation storage)
  - b) **Analog computing** (especially for ultra-low precision/quantization)

Tensor Processing Unit (TPU v1)

- **Motivation**
  - Cost-effective ML scoring (no training)
  - Latency- and throughput-oriented
  - **Improve cost-performance over GPUs by 10x**

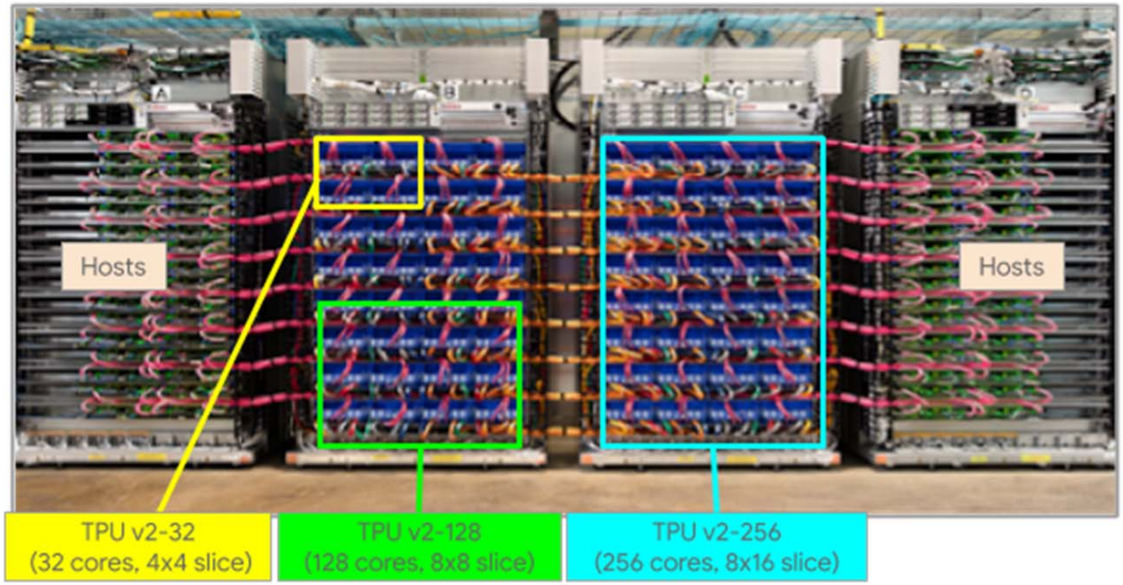
[Norman P. Jouppi et al: In-Datacenter Performance Analysis of a Tensor Processing Unit. **ISCA 2017**]

- **Architecture**
  - 256x256 8bit matrix multiply unit (systolic array → pipelining)
  - 64K MAC per cycle (92 TOPs at 8 bit)
  - 50% if one input 16bit
  - 25% if all inputs 16 bit



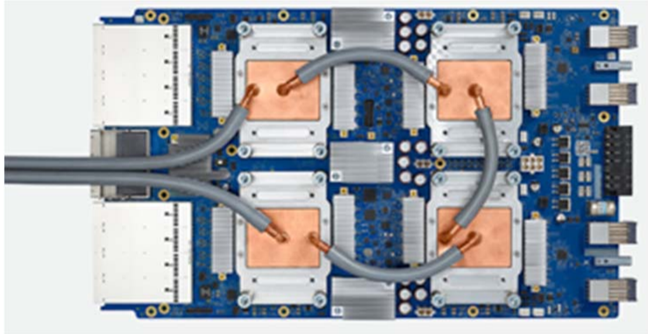
Tensor Processing Unit (TPU v2)

- **Motivation**
  - Cost effective ML training (not scoring) because edge device w/ custom inference but training in data centers
  - Unveiled at Google I/O 2017
  - Board w/ 4 TPU chips
  - Pod w/ 64 boards and custom high-speed network
  - Shelf w/ 2 boards or 1 processor



- **Cloud Offering (beta)**
  - Min 32 cores
  - Max 512 cores

Tensor Processing Unit (TPU v3)

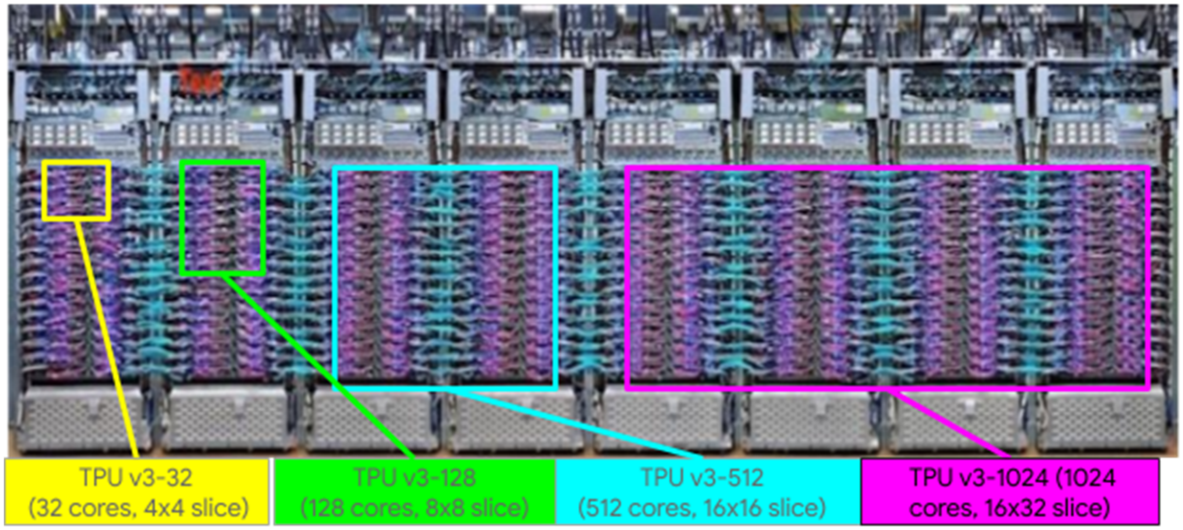


- **Motivation**
  - Competitive cost-performance compared to state-of-the-art GPUs
  - Unveiled at **Google I/O 2018**
  - Added **liquid cooling**
  - Twice as many racks per pod, twice as many TPUs per rack

  ➔ TPUv3 promoted as **8x higher performance** than TPUv2

- **Cloud Offering (beta)**
  - Min 32 cores
  - Max 2048 cores (~100PFLOPs)

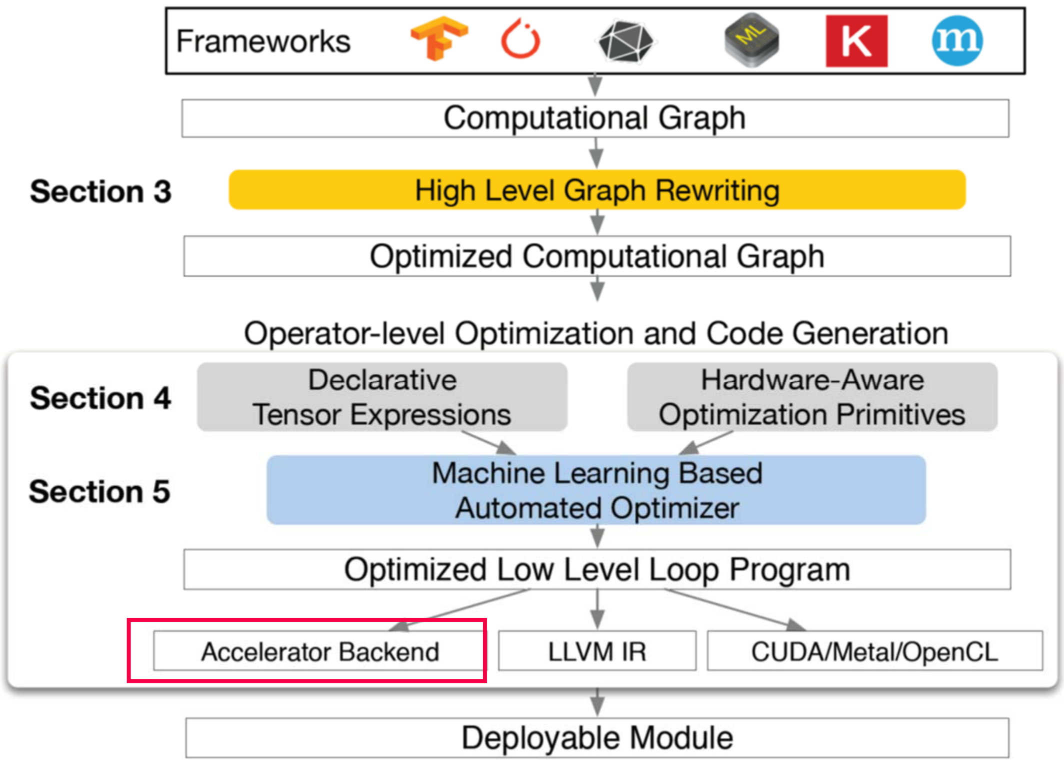
[TOP 500 Supercomputers: Summit @ Oak Ridge NL ('18): **200.7 PFLOP/s (2.4M cores)**]



Recap: Operator Fusion and Code Generation

- **TVM: Code Generation for HW Accelerators**
  - Graph- /operator-level optimizations for **embedded and HW accelerators**
  - **Lack of low-level instruction set!**
  - Schedule Primitives
    - Loop Transform
    - Thread Binding
    - Compute Locality
    - Tensorization
    - Latency Hiding

[Tianqi Chen et al: TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. **OSDI 2018**]



➔ **Apache** tvm

Excursus: Quantum Machine Learning

- **Background** (Schrödinger's cat)
  - Concepts: superposition, entanglement, de-coherence / uncertainty
- **IBM Q**
  - Hardware and software stack for quantum computing
  - **Qiskit:** OSS Python framework [https://qiskit.org/]
  - Experiment w/ quantum computers up to 20 qubit
  - **Gates:** Hadamard, NOT, Phases, Pauli, barriers transposed conjugate, if, measurement

- **Early ML (Systems) Work**
  - **Training quantum neural networks** (relied on quantum search in O(√N))
  - **SVM classification** w/ large feature space
  - **TensorFlow Quantum** (TFQ), on OSS **Cirq** for hybrid models [https://www.tensorflow.org/quantum]

[Bob Ricks, Dan Ventura: Training a Quantum Neural Network. **NIPS 2003**]

[Vojtěch Havlíček et al: Supervised learning with quantum-enhanced feature spaces. **Nature 2019**]

ML Hardware Fallacies and Pitfalls

- **Recommended Reading**
  - [Jeff Dean, David A. Patterson, Cliff Young: A New Golden Age in Computer Architecture: Empowering the Machine-Learning Revolution. **IEEE Micro 2018**]

- **#1 Fallacy: Throughput over Latency**
  - Given the large size of the ML problems, the HW focus should be op/s (throughput) rather than time to solution (latency)
- **#2 Fallacy: Runtime over Accuracy**
  - Given large speedup, ML researchers would be willing to sacrifice accuracy
- **#3 Pitfall: Designing HW using last year's models**
- **#4 Pitfall: Designing ML HW assuming the ML system is untouchable**

- **Trend: Chip Placement**
  - Chip placement as part of chip design process
  - SRAM, logic → optimize power, performance, area

[Azalia Mirhoseini, Anna Goldie, et al: Chip Placement with Deep Reinforcement Learning. **CoRR 2020**]

Summary and Conclusions

- **Different Levels of Hardware Specialization**
  - General-purpose CPUs and GPUs
  - FPGAs, DNN ASICs, and other technologies

Increasing importance of specialization:
End of Moore's Law
End of Dennard Scaling

- **Next Lectures**
  - **08 Caching, Partitioning, Indexing and Compression** [May 15]
  - **May 21/22:** Ascension Day (Christi Himmelfahrt)

(Part A: Overview and ML System Internals)

  - **09 Data Acquisition, Cleaning, and Preparation** [May 29]
  - **10 Model Selection and Management** [Jun 05]
  - **11 Model Debugging Techniques** [Jun 12]
  - **12 Model Serving Systems and Techniques** [Jun 19]
  - ~~13 Trends and Research Directions 2020 [Jun 26]~~

(Part B: ML Lifecycle Systems)