Advanced Computer Architecture

高 级 计 算 机 体 系 结 构

第18讲：Domain Specific Arch (2)

张献伟

xianweiz.github.io

DCS5637, 12/28/2022

Review Questions

- Differences between coherence and consistency?

  Same vs different location, eventually vs when, cache vs. mem, …

- What are possible values of *data* in TSO processors? Give the ordering.

  1: ①②③④

P0	P1
// flag = 0; data = 0;	

data = 1; ①
flag = 1; ②
	while (flag == 0); ③
	print data; ④

- What about PSO processors?

  1: ①②③④/②①③④/②③①④

  0: ②③④①

- Why DSA?

  Ending of Moore's law; limited perf impr of general-purpose.

- DSA design guidelines?

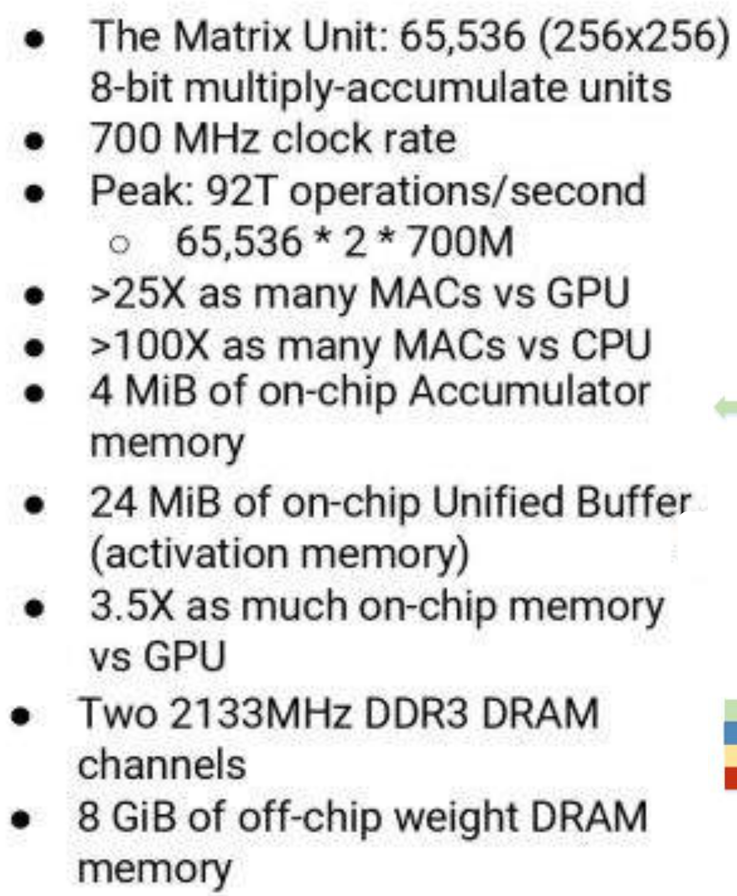
  Dedicated memories, larger ALUs, easy parallelism, smaller data size, domain-specific language.

- Target applications of TPU?

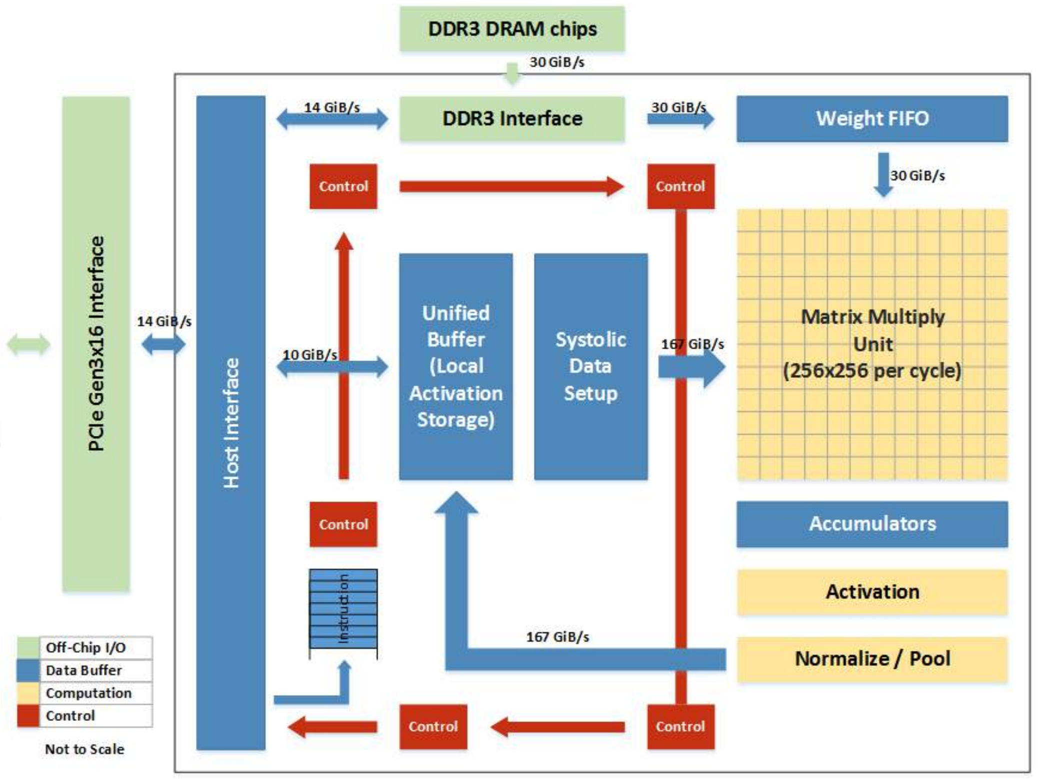
  DNN: MLP, CNN, LSTM.

TPU Architecture[架构]

- A coprocessor on the PCIe I/O bus
- A large software-managed on-chip memory



- The Matrix Unit: 65,536 (256x256) 8-bit multiply-accumulate units
- 700 MHz clock rate
- Peak: 92T operations/second
  - 65,536 * 2 * 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

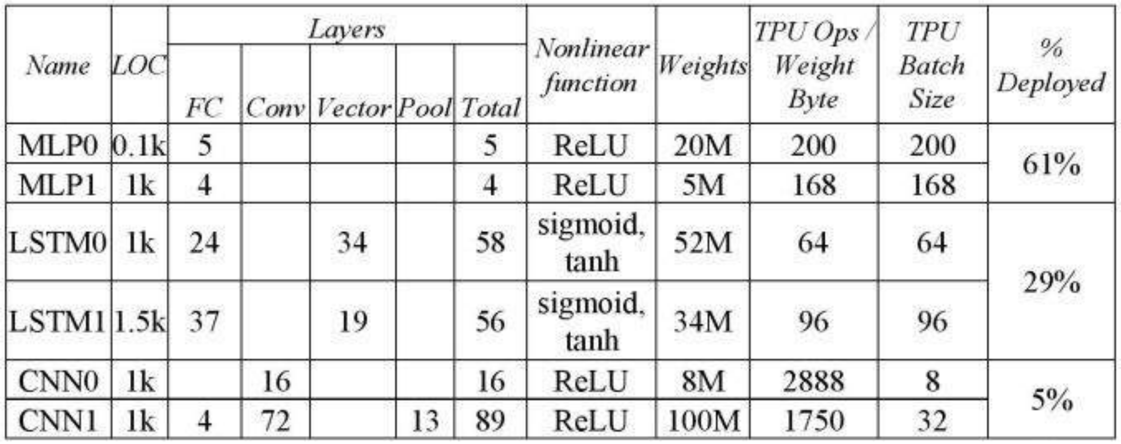


https://www.anandtech.com/show/11749/hot-chips-google-tpu-performance-analysis-live-blog-3pm-pt-10pm-utc

TPU Performance[性能]

- Compare using six benchmarks
  - Representing 95% of TPU inference workload in Google data center in 2016
  - Typically written in TensorFlow, pretty short (100-1500 LOCs)
- Chips/servers being compared
  - CPU server: Intel 18-core, dual-socket Haswell; host server for GPUs/TPUs
  - GPU accelerator: Nvidia K80

Inference Datacenter Workload (95%)



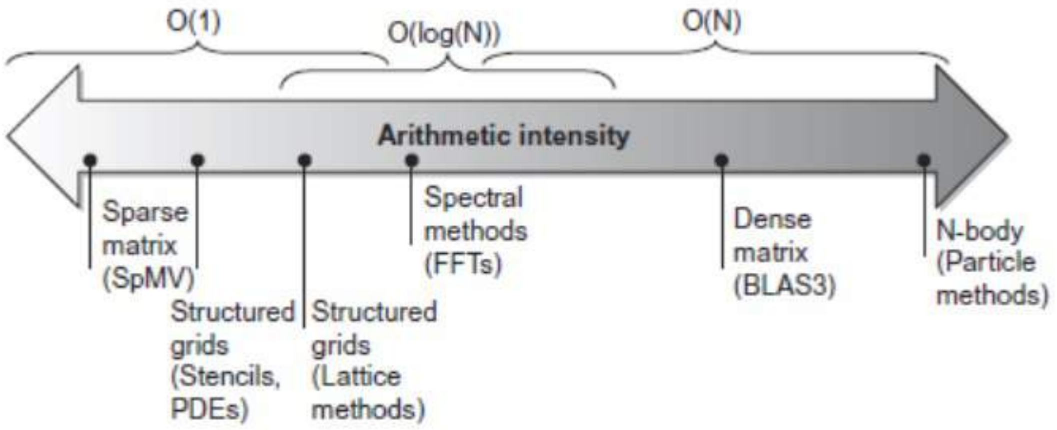
Name	*LOC*	*Layers*					*Nonlinear function*	*Weights*	*TPU Ops / Weight Byte*	*TPU Batch Size*	*% Deployed*
		FC	*Conv*	*Vector*	*Pool*	*Total*					
MLP0	0.1k	5				5	ReLU	20M	200	200	61%
MLP1	1k	4				4	ReLU	5M	168	168	
LSTM0	1k	24		34		58	sigmoid, tanh	52M	64	64	29%
LSTM1	1.5k	37		19		56	sigmoid, tanh	34M	96	96	
CNN0	1k		16			16	ReLU	8M	2888	8	5%
CNN1	1k	4	72		13	89	ReLU	100M	1750	32	

Roofline Performance Model[屋顶线]

- The roofline model was introduced in 2009
  - Samuel Williams, Andrew Waterman, and David Patterson. 2009. *Roofline: an insightful visual performance model for multicore architectures*. Commun. ACM
- It provides an easy way to get performance bounds for compute and memory bandwidth bound computations
- It relies on the concept of **Computational Intensity** (CI)
  - Sometimes also called Arithmetic or Operational Intensity
- The model provides a relatively simple way for performance estimates based on the computational kernel and hardware characteristics
  - Performance [GF/s] = function (hardware and software characteristics)

Roofline Performance Model(cont.)

- Basic idea
  - Plot peak FP throughput as a function of arithmetic intensity
  - Ties together FP performance and memory performance for a target machine
- Arithmetic intensity[运算密度/算存比]
  - Ratio of FP operations per byte of memory accessed
    - (total #FP operations for a program) / (total data bytes transferred to main memory during program execution)



Arithmetic Intensity[运算密度]

- $A.I. = \frac{W}{Q}$ (FLOP/Byte)
  - W: amount of work, i.e floating point operations required
  - Q: memory transfer, i.e access from DRAM to lowest level cache
- Examples

```
for (i = 0; i < N; ++i)
    z[i] = x[i]+y[i]
```

1 ADD
2 (8 byte) loads
1 (8 byte) write
AI = 1 / (2*8 + 8) = 1/24

```
for (i = 0; i < N; ++i)
    z[i] = x[i]+y[i]*x[i]
```

1 ADD
1 MUL
2 (8 byte) loads
1 (8 byte) write
AI = 2 / (2*8 + 8) = 1/12

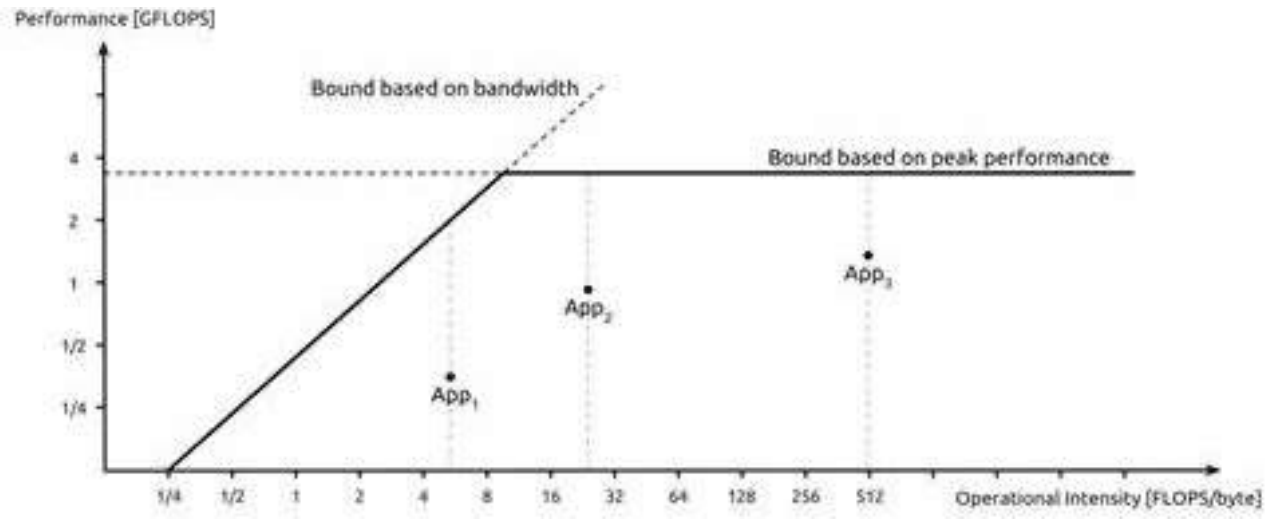
Example

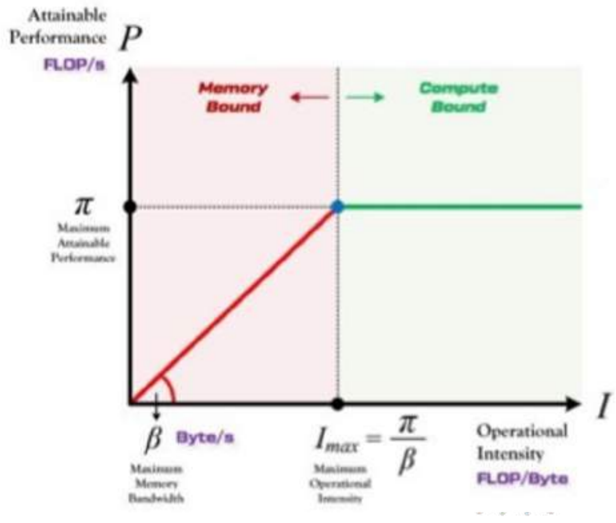
```
float in[N], out[N];
for (int i=1; i<N-1; i++)
    out[i] = in[i-1]-2*in[i]+in[i+1];
```

- Amount of FLOPS: 3(N-2)
  - For every i: out[i] = in[i-1]-2*in[i]+in[i+1] → 3 flop
  - Loop over: for (int i=1; i<N-1; i++) → (N-2) repetitions
- Memory accesses Q: depends on cache size
  - No cache (read directly from slow memory) → every data accessed is counted
    - 4 accesses x (N-2) repetitions x 4 bytes → A.I. = 3/16
  - Perfect cache (infinite sized cache) → data is read & written only once
    - 2 accesses x (N-2) repetitions x 4 bytes → A.I. = 3/8

Roofline Analysis

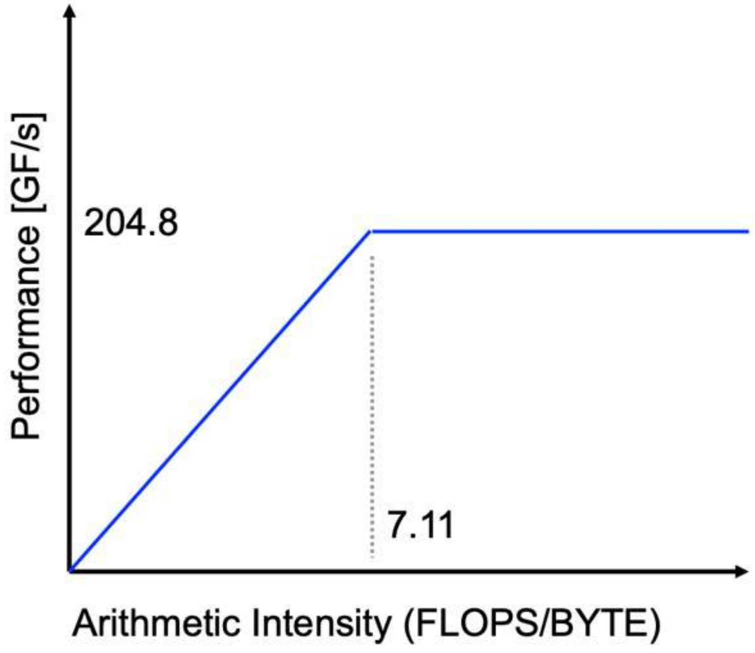
- "**Roofline**" sets an upper bound on perf of a kernel depending on its arithmetic intensity
  - Think of arithmetic intensity as a pole that hits the roof
    - Hits the flat part: perf is computationally limited
    - Hits the slanted part: perf is ultimately limited by memory bandwidth
- **Ridge point**: the diagonal and horizontal roofs meet
  - Far to right: only very intensive kernels can achieve max perf
  - Far to left: almost any kernel can potentially hit max perf





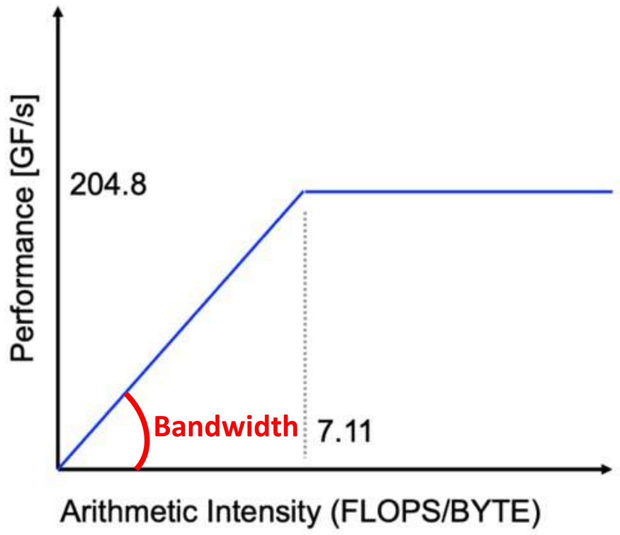
Example

- Consider: for (i = 0; i < N; ++i) y[i] = a*x[i]+y[i]
  - For each “i” :
    - 1 addition, 1 multiplication
    - 2 loads of 8 bytes each
    - 1 store
- Execution on BlueGene/Q
  - Peak 204.8 GFLOP/node
- Performance estimates:
  - AI = 2/(3*8) = 1 / 12 1/12 < 7.11 → limited area on the Roofline plot
  - 7.11/(1/12)= 85.32
  - 204.8 / 85.32 = 2.4 GF/s



Example (cont.)

- Peak double precision floating-point performance
  - 204.8 GFLOPS
- Peak memory bandwidth
  - 204.8/7.11 = 28.8 GB/s
  - The steady state bandwidth potential of the memory in a computer, not the pin bandwidth of the DRAM chips
  - Common way is to measure it with benchmarks like STREAM

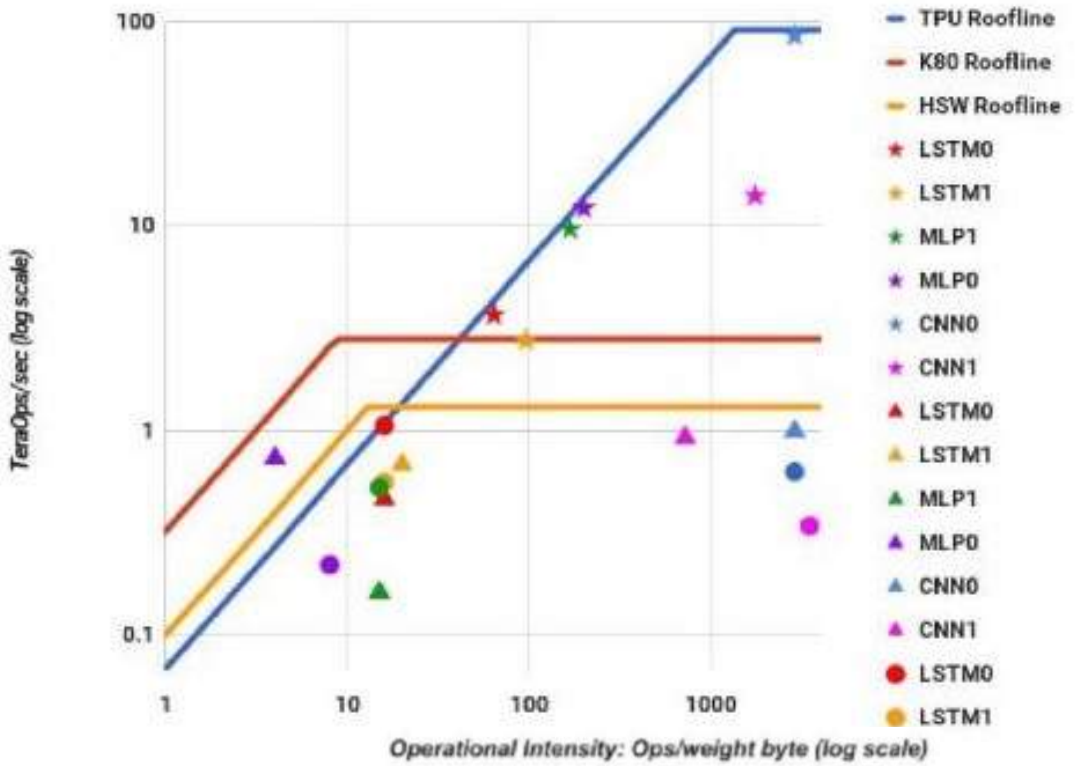


- Lawrence Livermore National Laboratory (LLNL)
- IBM BlueGene/Q
- Top500 2012/6 #1, 16.3PFLOPS (efficiency 81%), 1.57Mcore, 7.89MW, 2.07GFLOPS/W
- 4 BlueGene/Q were listed in top10
- 18core/chip, 1.6GHz, 4way SMT, 204.8GFLOPS/55W, L2:32MB eDRAM, mem: 16GB, 42.5GB/s
- 32chip/node, 32node/rack, 96rack

http://www.hpcs.cs.tsukuba.ac.jp/~taisuke/tmp/ACHPC-LN9-4cut.pdf

TPU Roofline Performance

- TPU: its ridge point is far to the right at 1350
  - CNN1 is much further below its Roofline than the other DNNs
    - Waiting for weights to be loaded into the matrix unit
  - Ridge point comparison:
    - CPU: 13, GPU: 9 → better balanced, but perf a lot lower



☆Star = TPU
△Triangle = GPU
○Circle = CPU

Cost-Performance

- Cost metric: performance per watt
  - "Total": includes the power consumed by the host CPU server when calculating perf/watt for the GPU and TPU
  - "Incremental": subtracts the host CPU power from the total
- Total: GPU is 2.1x CPU, TPU is 34x CPU
- Incremental: TPU is 83x CPU, 29x GPU

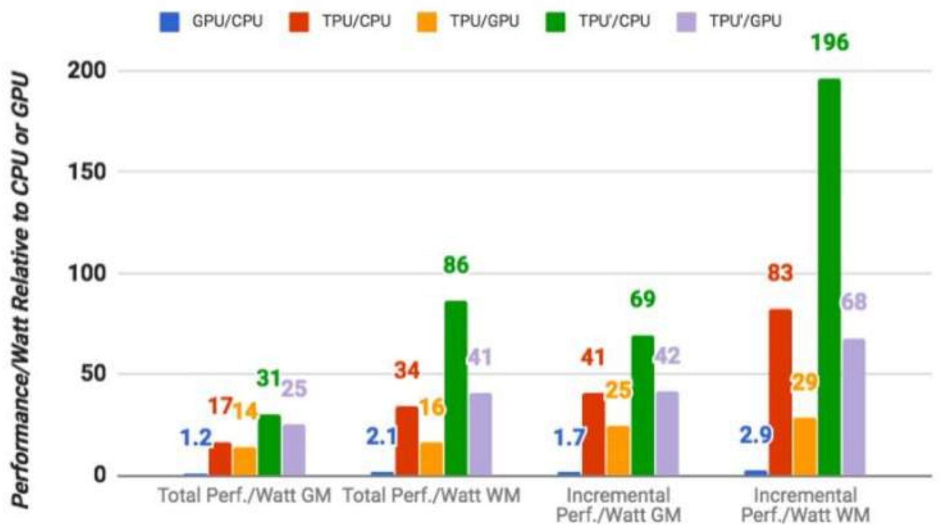
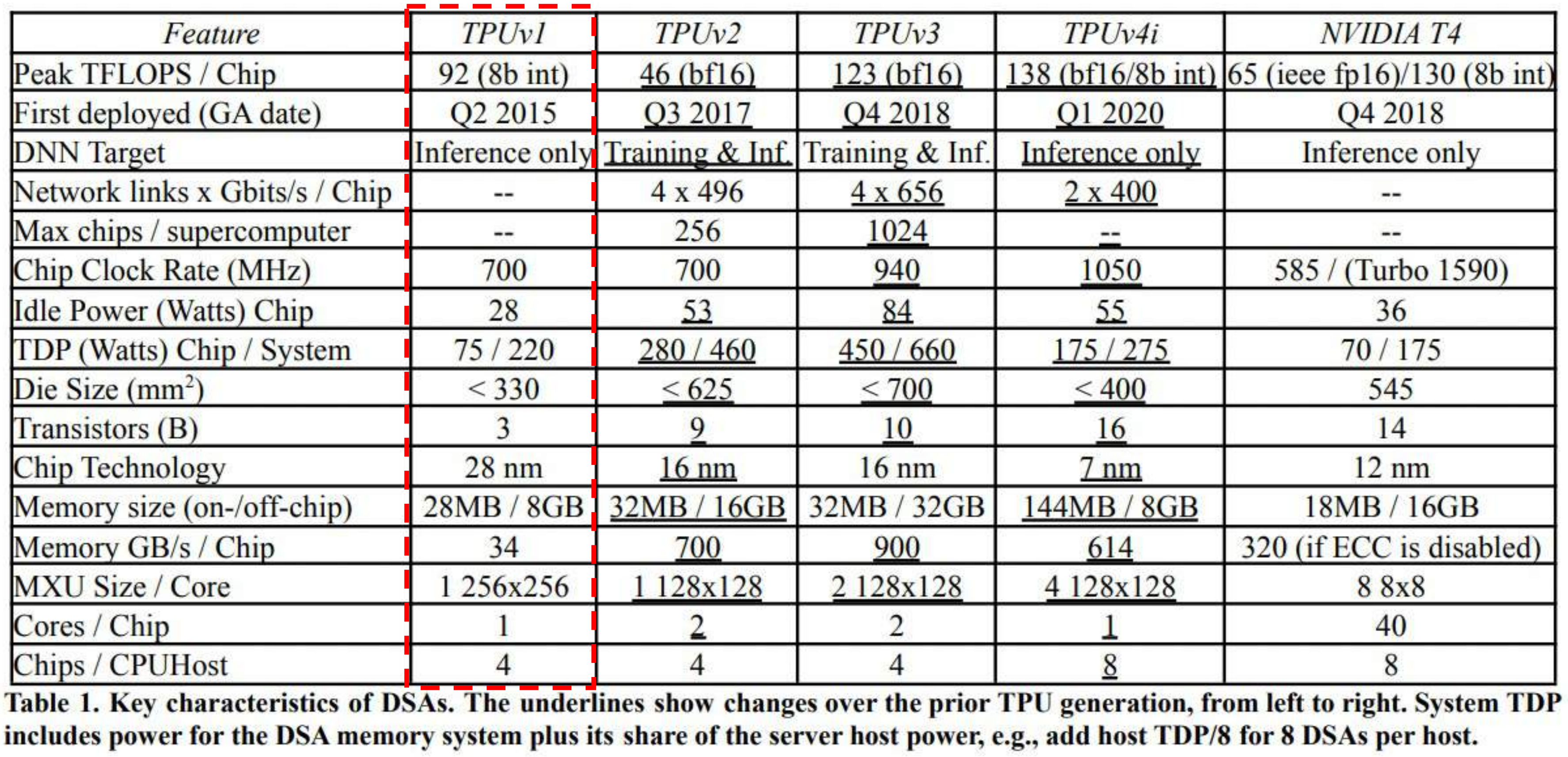


Figure 9. Relative performance/Watt (TDP) of GPU server (blue bar) and TPU server (red bar) to CPU server, and TPU server to GPU server (orange bar). TPU' is an improved TPU (Sec. 7). The green bar shows its ratio to the CPU server and the lavender bar shows its relation to the GPU server. Total includes host server power, but incremental doesn't. GM and WM are the geometric and weighted means.

https://www.extremetech.com/computing/247199-googles-dedicated-tensorflow-processor-tpu-makes-hash-intel-nvidia-inference-workloads

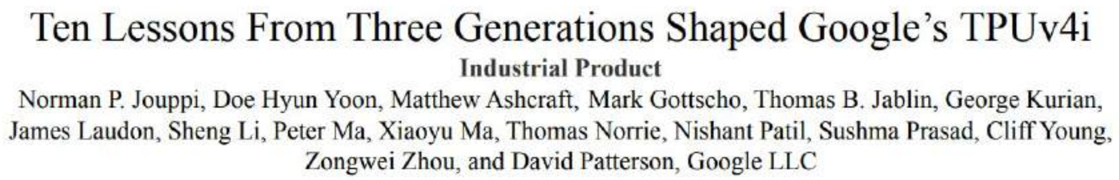
TPU Generations



Feature	*TPUv1*	*TPUv2*	*TPUv3*	*TPUv4i*	*NVIDIA T4*
Peak TFLOPS / Chip	92 (8b int)	46 (bf16)	123 (bf16)	138 (bf16/8b int)	65 (ieee fp16)/130 (8b int)
First deployed (GA date)	Q2 2015	Q3 2017	Q4 2018	Q1 2020	Q4 2018
DNN Target	Inference only	Training & Inf.	Training & Inf.	Inference only	Inference only
Network links x Gbits/s / Chip	--	4 x 496	4 x 656	2 x 400	--
Max chips / supercomputer	--	256	1024	--	--
Chip Clock Rate (MHz)	700	700	940	1050	585 / (Turbo 1590)
Idle Power (Watts) Chip	28	53	84	55	36
TDP (Watts) Chip / System	75 / 220	280 / 460	450 / 660	175 / 275	70 / 175
Die Size (mm^2)	< 330	< 625	< 700	< 400	545
Transistors (B)	3	9	10	16	14
Chip Technology	28 nm	16 nm	16 nm	7 nm	12 nm
Memory size (on-/off-chip)	28MB / 8GB	32MB / 16GB	32MB / 32GB	144MB / 8GB	18MB / 16GB
Memory GB/s / Chip	34	700	900	614	320 (if ECC is disabled)
MXU Size / Core	1 256x256	1 128x128	2 128x128	4 128x128	8 8x8
Cores / Chip	1	2	2	1	40
Chips / CPUHost	4	4	4	8	8

Table 1. Key characteristics of DSAs. The underlines show changes over the prior TPU generation, from left to right. System TDP includes power for the DSA memory system plus its share of the server host power, e.g., add host TDP/8 for 8 DSAs per host.

2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)



Ten Lessons From Three Generations Shaped Google's TPUv4i

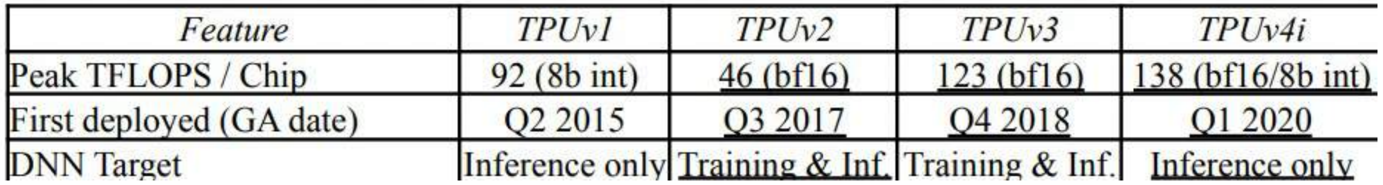
Industrial Product

Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou, and David Patterson, Google LLC

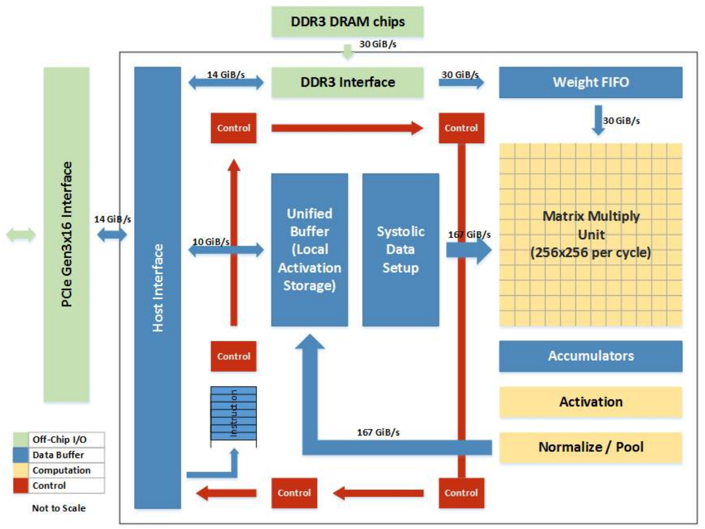
https://www.gwern.net/docs/ai/scaling/hardware/2021-jouppi.pdf

中山大學 SUN YAT-SEN UNIVERSITY

TPU Generations (cont.)

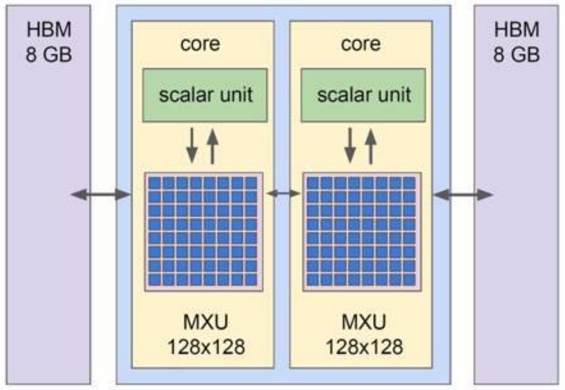


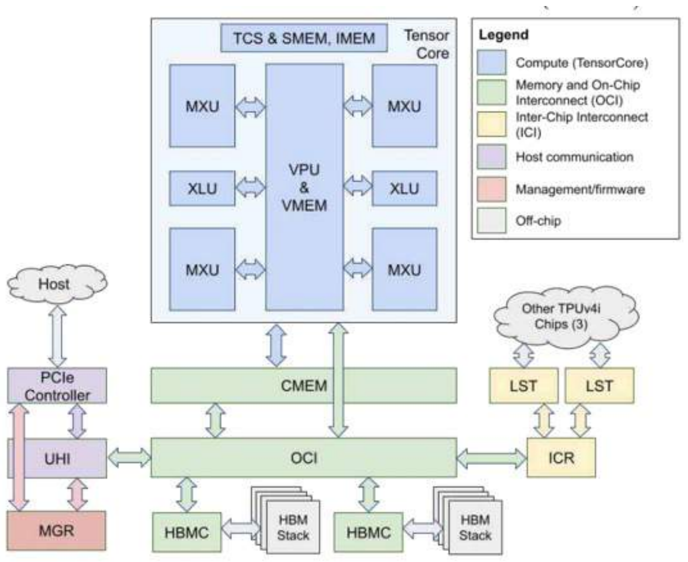
Feature	*TPUv1*	*TPUv2*	*TPUv3*	*TPUv4i*
Peak TFLOPS / Chip	92 (8b int)	46 (bf16)	123 (bf16)	138 (bf16/8b int)
First deployed (GA date)	Q2 2015	Q3 2017	Q4 2018	Q1 2020
DNN Target	Inference only	Training & Inf.	Training & Inf.	Inference only



TPUv2 Chip

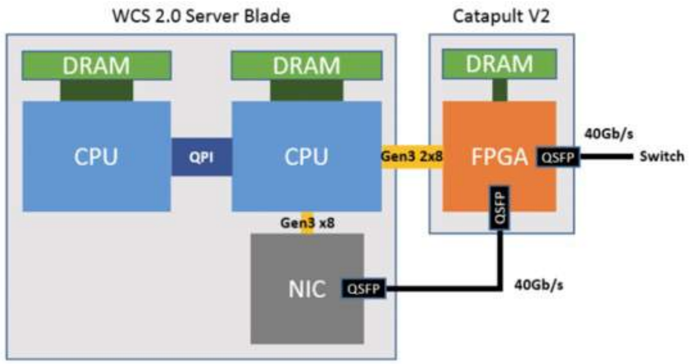
- 16 GB of HBM
- 600 GB/s mem BW
- Scalar unit: 32b float
- MXU: 32b float accumulation but reduced precision for multipliers
- 45 TFLOPS





Microsoft Catapult

- Project Catapult
  - To transform cloud computing by augmenting CPUs with an interconnected and configurable compute layer composed of programmable silicon
- FPGAs offer a unique combination of speed and flexibility
  - FPGAs could deliver efficiency and performance without the cost, complexity, and risk of developing custom ASICs
  - The FPGA can act as a local compute accelerator, an inline processor, or a remote accelerator for distributed computing

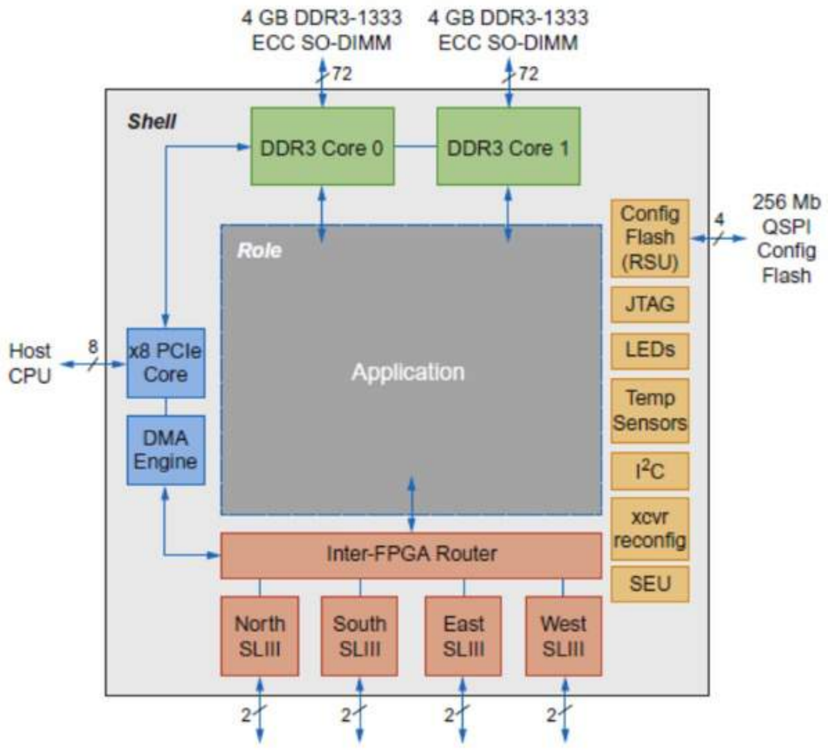


Catapult v2 Mezzanine card

https://www.microsoft.com/en-us/research/project/project-catapult/

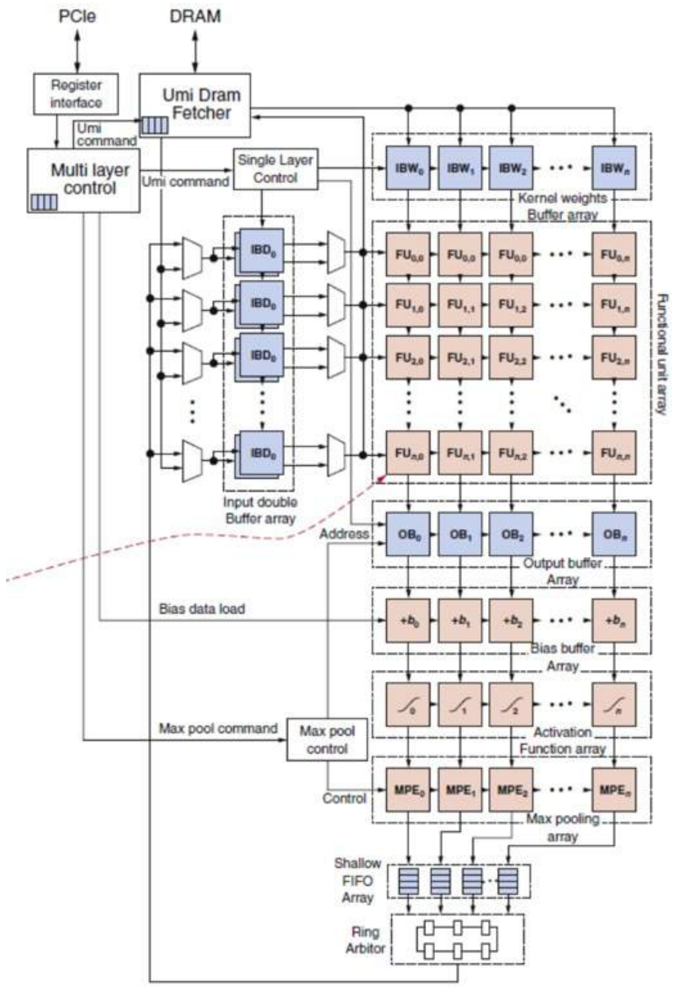
Microsoft Catapult (cont.)

- Needed to be general purpose and power efficient
  - Uses FPGA PCIe board with dedicated 20 Gbps network in 6 x 8 torus
  - Each of the 48 servers in half the rack has a Catapult board
  - Limited to 25 watts
  - 32 MB Flash memory
  - Two banks of DDR3-1600 (11 GB/s) and 8 GB DRAM
  - FPGA (unconfigured) has 3962 18-bit ALUs and 5 MB of on-chip memory
  - Programmed in Verilog RTL
  - Shell is 23% of the FPGA

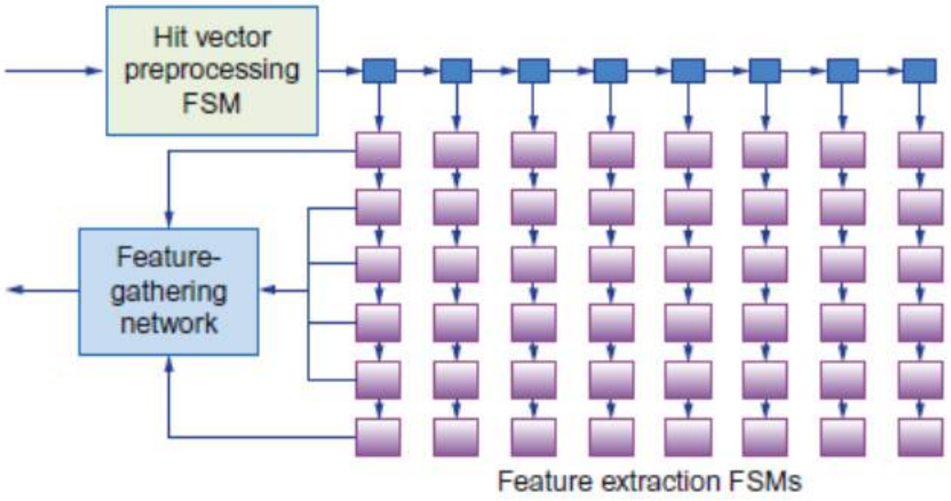


Catapult Applications

- The processing element (PE) of the CNN Accelerator for Catapult



- The architecture of FPGA implementation of the Feature Extraction stage in search acceleration



How Catapult Follows the Guidelines

- *Use dedicated memories*
  - 5 MB dedicated memory
- *Invest resources in arithmetic units and dedicated memories*
  - 3926 ALUs
- *Use the easiest form of parallelism that matches the domain*
  - 2D SIMD for CNN, MISD parallelism for search scoring
- *Reduce the data size and type needed for the domain*
  - Uses mixture of 8-bit integers and 64-bit floating-point
- *Use a domain-specific programming language*
  - Uses Verilog RTL; Microsoft did not follow this guideline

Advanced Computer Architecture

高级计算机体系结构

第18讲：CXL & Computing

张献伟

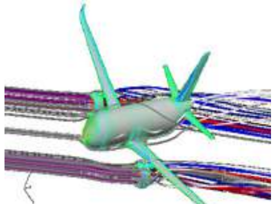
xianweiz.github.io

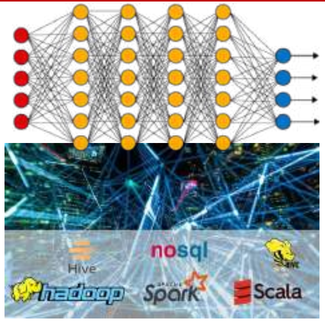
DCS5637, 12/28/2022

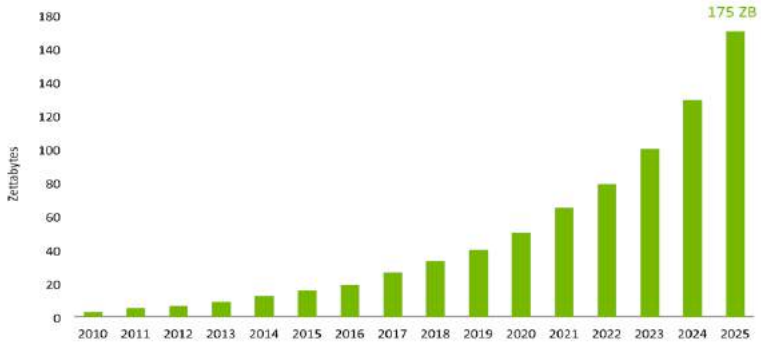
OUTLINE

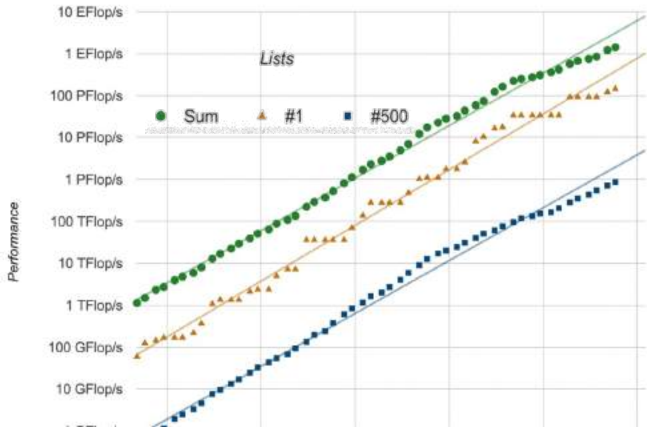
Applications Keep Growing

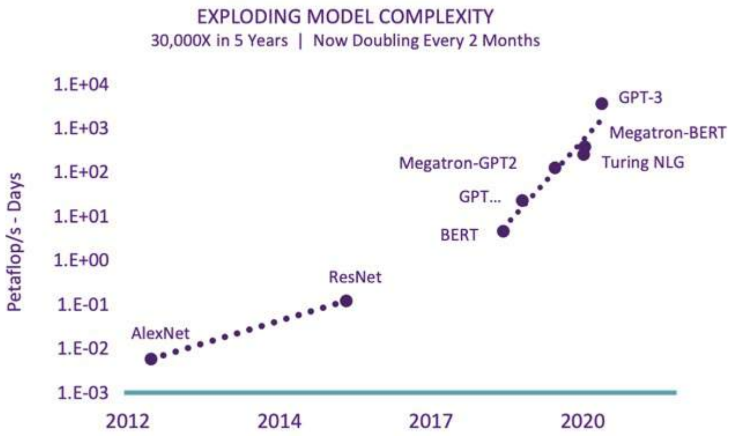
- High-performance Computing
  - FLOPS: mega - giga - tera - petra – exa
- Artificial Intelligence
  - More complicated model: AlexNet - ResNet - BERT – GPT
- Big Data
  - Exponential growth on volume







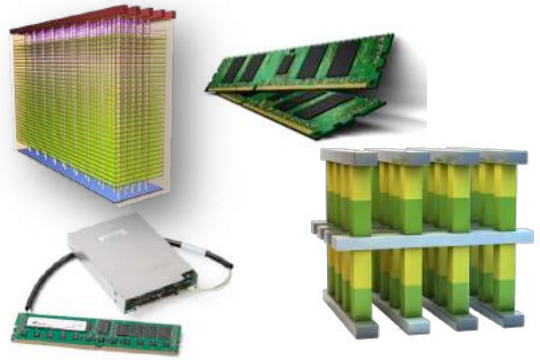




System Evolution

- Computing
  - Scalar → vector
  - Homogeneous → heterogeneous: CPU + GPU/accelerator/FPGA/DPU

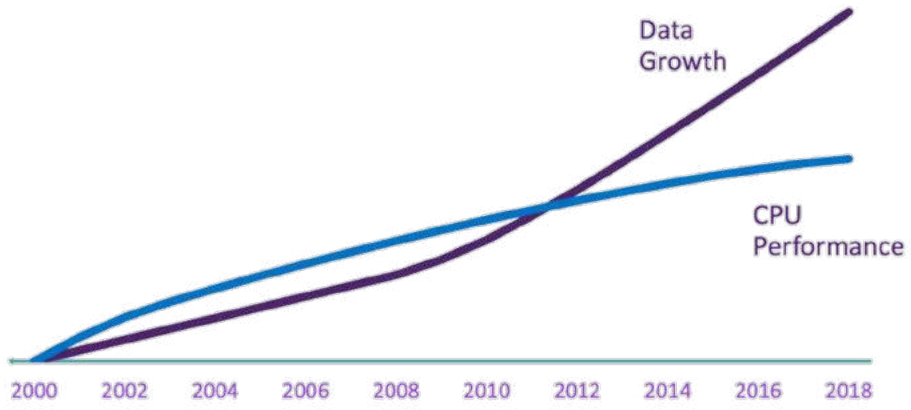
- Memory
  - DRAM, GDDR, HBM
  - HDD, SSD, NVM



- Interconnect
  - Intra-node: PCI-e, CAPI/OpenCAPI, CCIX, UPI/QPI, NVLink, Gen-Z
  - Inter-node: Ethernet, InfiniBand, Omnipath, HPC Ethernet/Slingshot

Memory Wall

- Memory lags behind compute
  - Bandwidth wall: improvement rate in processing far exceeds that of memory
  - Capacity wall: growing imbalance in compute-memory ratio in data centers
- Memory subsystem becomes one of the most crucial system-level performance bottlenecks

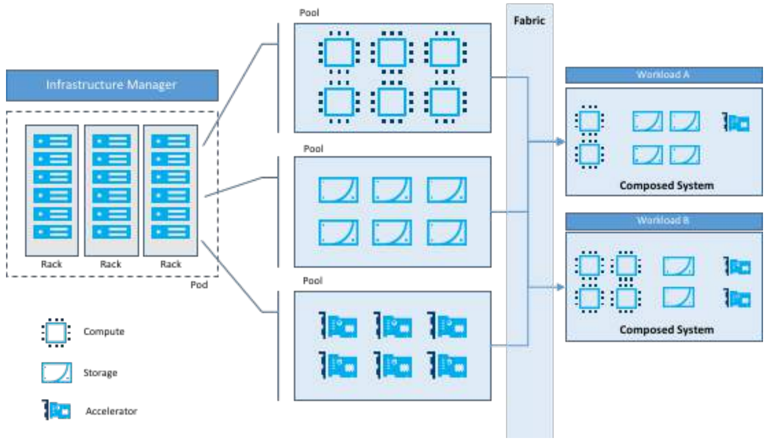


Supercomputer	Compute (PFLOPS)	Memory (PB)	C-M Ratio
Frontier (2022.6)	1102	9.2	118:1
Fugaku (2020.6)	415	4.85	86:1
Summit (2018.6)	122	2.8+3.7	19:1
Sunway TaihuLight (2016.6)	105	1.3	81:1
Tianhe-2 (2013.6)	34	1.4	24:1

https://www.top500.org/lists/top500/

Memory Demands

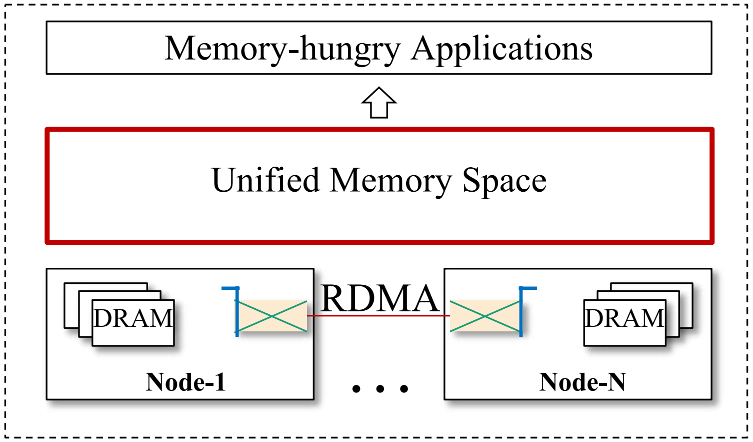
- Large capacity
  - To meet apps need of large space
- Fast access
  - Low latency, higher bandwidth
- Unified
  - Heterogeneous processors with own memory
- Scalable
  - Rack scale, multi-node
- Easy to use programming models, efficient management

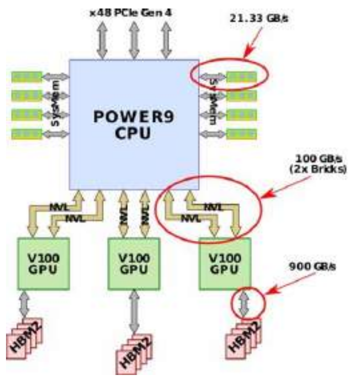


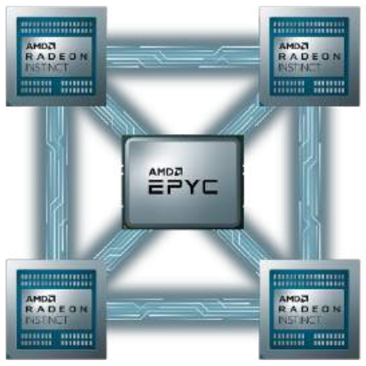
- Large-scale Memory Pooling
  - Unified memory space across nodes

Existing Solutions

- RDMA-based distributed memory
  - Based on Ethernet/IB
- Unified memory
  - Enabled with high-speed interconnects, e.g., NVLink, Infinity Fabric
  - Coherent memory space between CPUs and GPUs
  - Already used in supercomputing





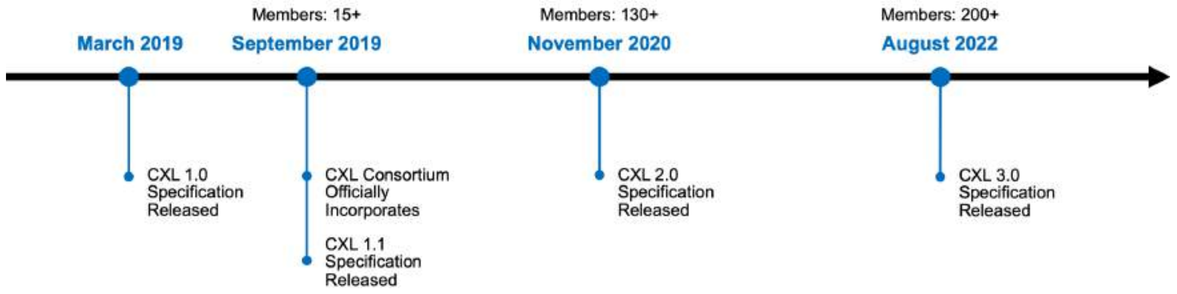


OUTLINE

The Enabler: Compute Express Link (CXL)

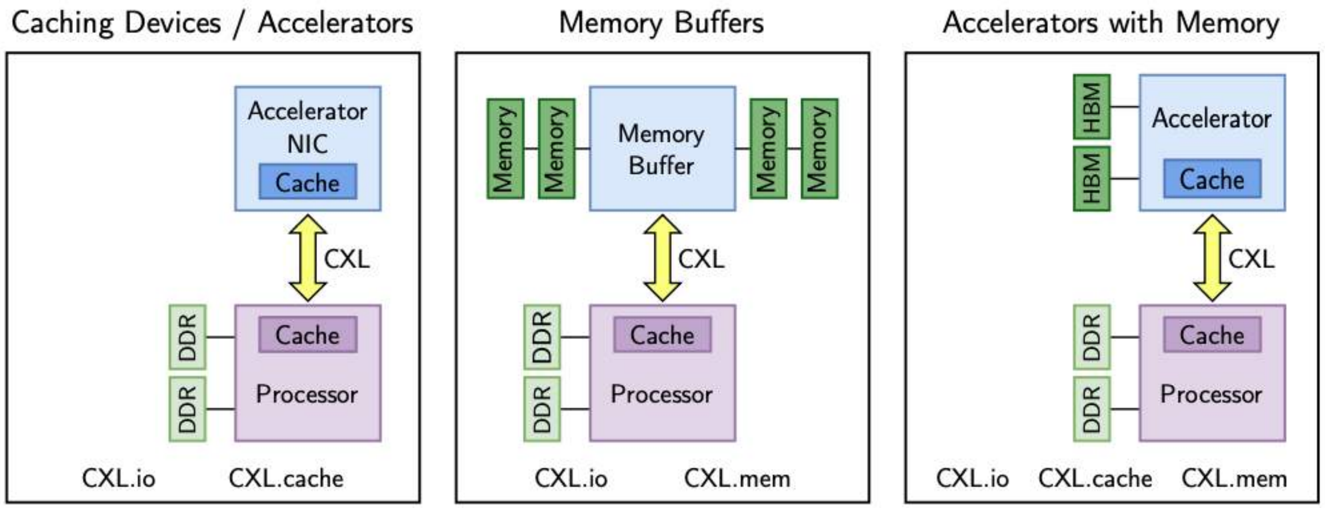
- Open industry standard processor interconnect
  - Unified, coherent memory space between the CPU and any memory attached CXL device
  - High-bandwidth, low-latency connection between host and devices including accelerators, memory expansion, and smart I/O devices
  - Utilizes PCI Express 5.0 physical layer infrastructure and the PCIe alternate protocol
  - Designed to meet demanding needs of HPC work in AI, ML, communication systems through enablement of coherency and memory semantics across heterogeneous processing and memory systems

OpenCAPI
GENZ
CXL
Compute Express Link


March 2019
CXL 1.0 Specification Released
Members: 15+
September 2019
CXL Consortium Officially Incorporates
CXL 1.1 Specification Released
Members: 130+
November 2020
CXL 2.0 Specification Released
Members: 200+
August 2022
CXL 3.0 Specification Released

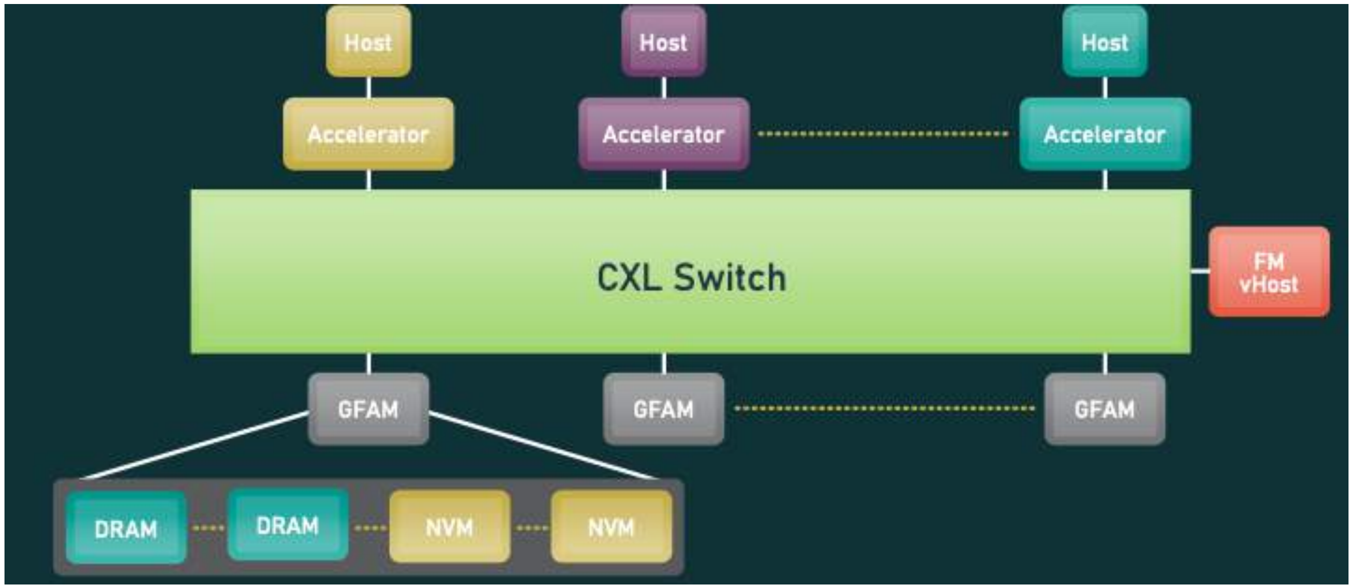
Compute Express Link (cont.)

- The CXL transaction layer is comprised of three dynamically multiplexed sub-protocols on a single link
  - CXL.io: functionally equivalent to the PCIe 5.0 protocol
  - CXL.cache: for devices to cache data from the CPU memory
  - CXL.memory: for processor to access the memory of attached devices



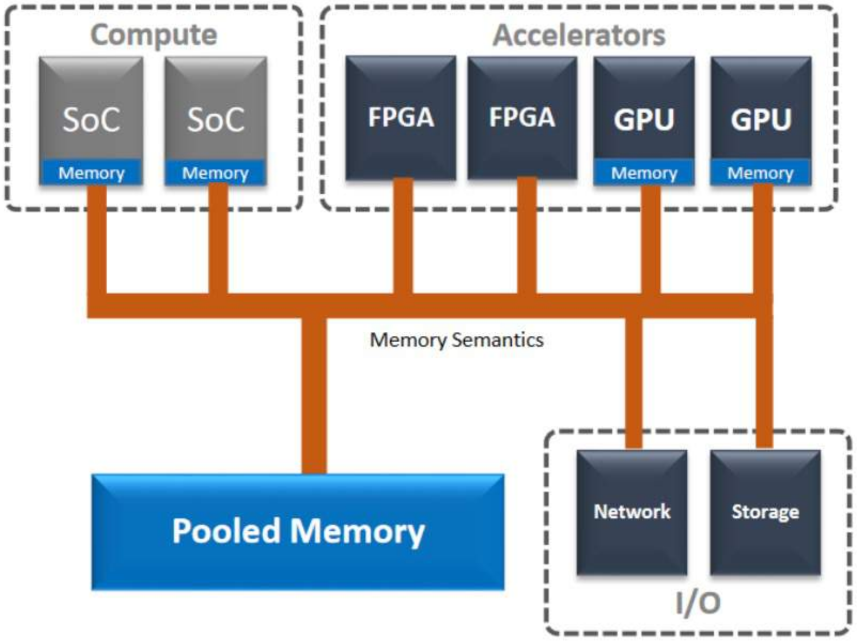
CXL-based Memory

- Unified sharing memory between host and devices
  - No longer use host memory as an intermediary for communication
- Scalable to provide more memory types and capacity
  - Just attached via CXL fabric
  - GFAM: Global Fabric Attached Memory



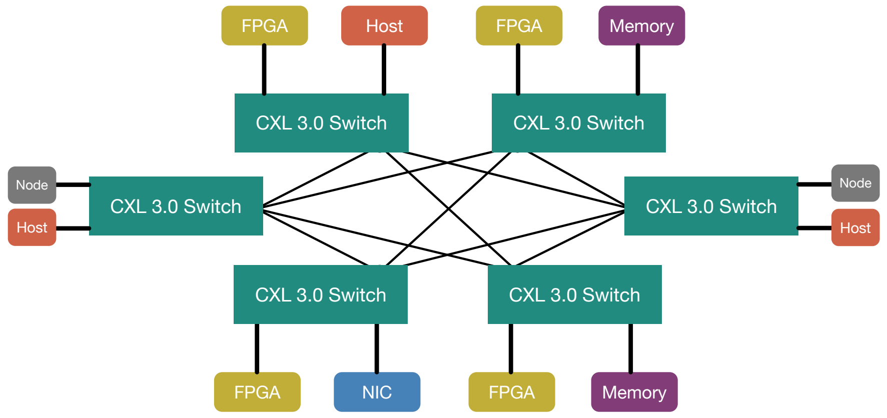
CXL-based Memory (cont.)

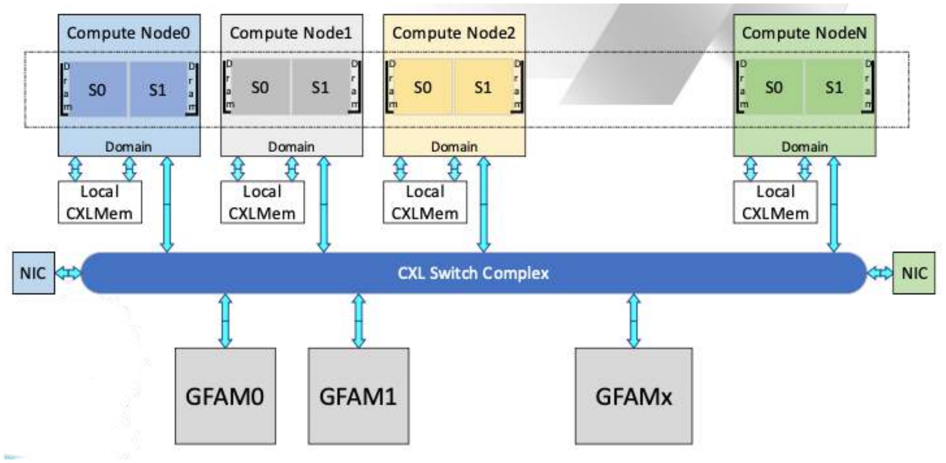
- Memory semantic
  - Handles all communications as memory operations such as load/store, put/get and atomic operations typically used by a processor
- Memory speed
  - Communications at the speed of memory, i.e., sub-microsecond latencies



CXL-based System

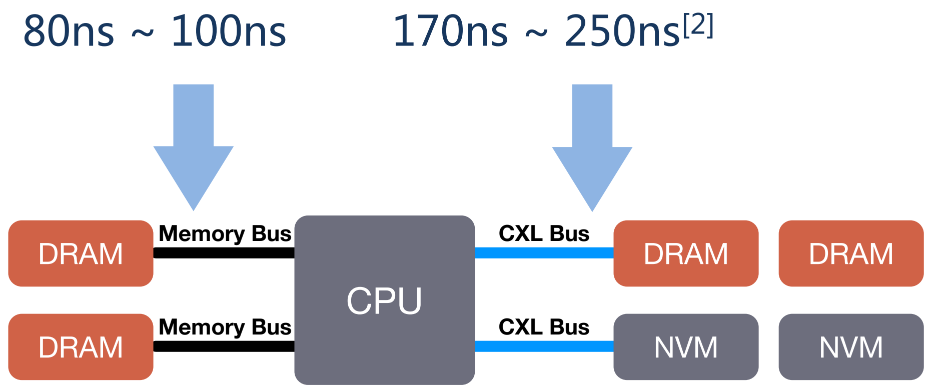
- Multi-tiered switching
  - Enables the implementation of switch fabrics
  - Switches can connect to other switches
- Rack-scale memory fabric
  - Fine-grained resource sharing across multiple domains

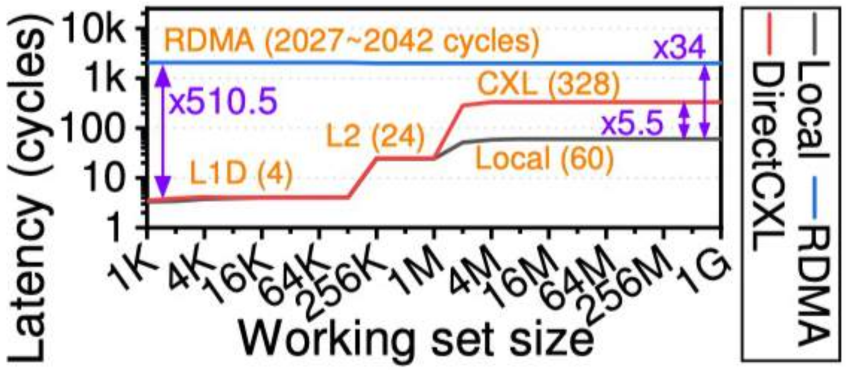




Feature: Fast Access

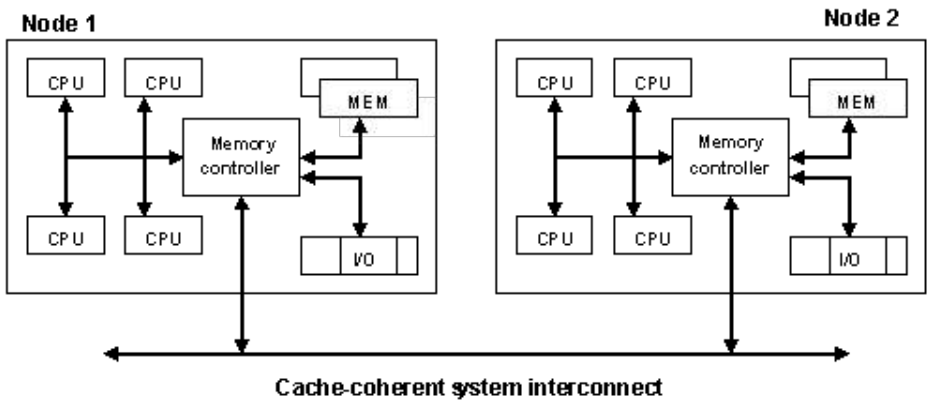
- As memory bus, CXL is slower than DRAM
  - Contributing factors: CXL phy/ctr - 40ns, Retimer - 20 ns, Propagation - 7n, EMC - 20ns
- As inter-node link, CXL is much faster than RDMA
  - Particularly at small granularities

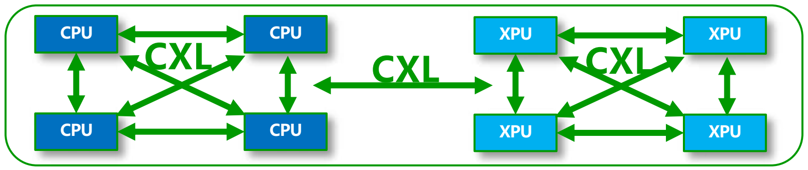




Feature: Data Coherence

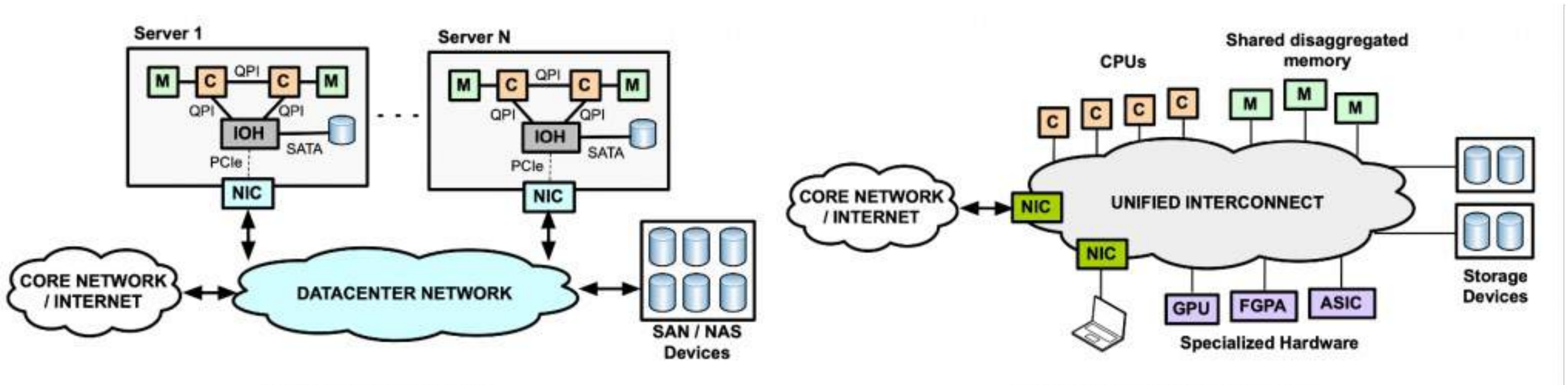
- Coherency between the CPU memory space and memory on attached devices
  - Allows resource sharing for higher performance
  - Reduced complexity and lower overall system cost
  - Permits to focus on target workloads vs. redundant memory management
  - Every processor in the rack is on the same page about what is happening simultaneously





Feature: Resource Disaggregation

- Data centers to scale out differently with dis-aggregation of resources with separate pools of memory, storage, and accelerators
  - You just grab what you need and compose your resources based on the workload characteristics

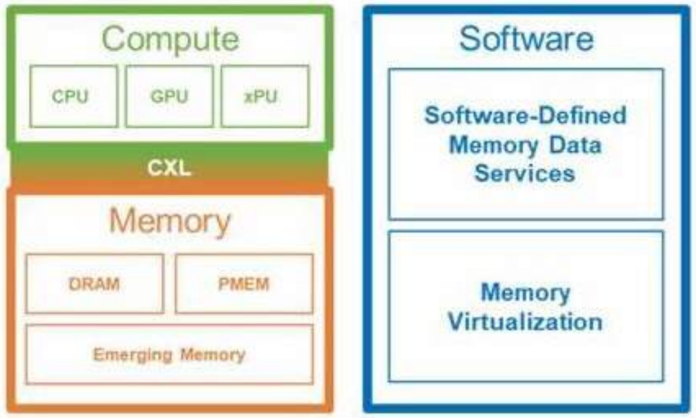


(a) Current datacenter (b) Disaggregated datacenter

OUTLINE

Big Memory Computing

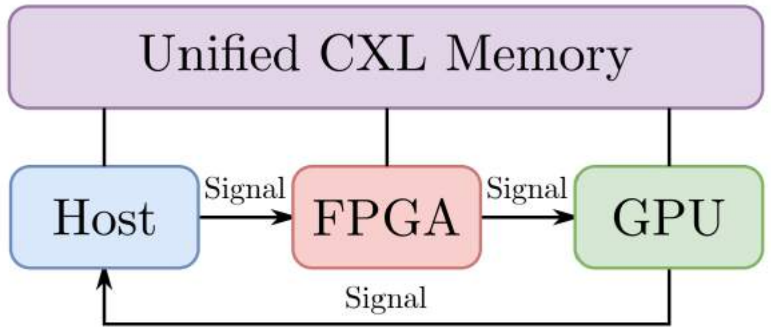
- Compute
  - Enhanced movement of operands and results between processors
  - Heterogeneous programming can be easier with unified memory
- Memory
  - Boosted capacity and bandwidth, highly scalable
- Software
  - Critical to maximize resource utilization and bridge applications to hardware

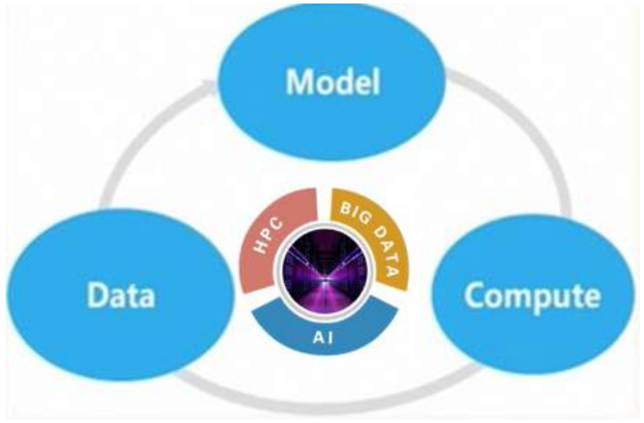


Application for Big Memory

- Q: How to adapt applications to the system?

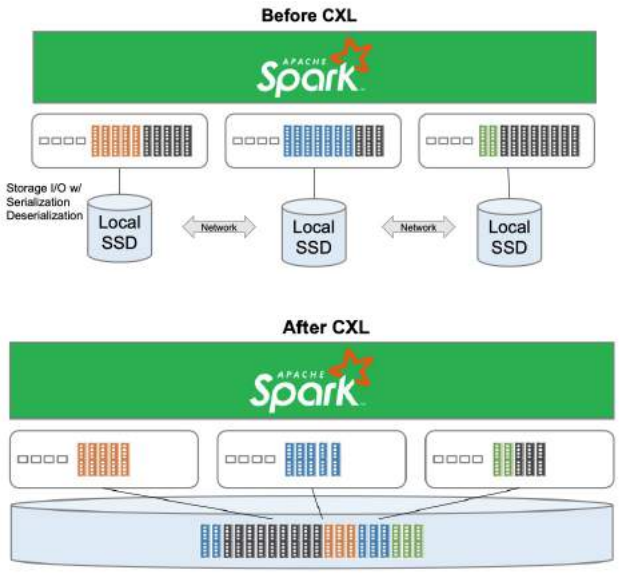
- Key features should be utilized to accelerate applications
  - Unified, coherent memory space between host and devices
    - Including accelerators, memory expansion, and smart I/O devices
  - Fine-grained, memory-semantic data sharing are now possible
    - Assessing and improving operations in real time

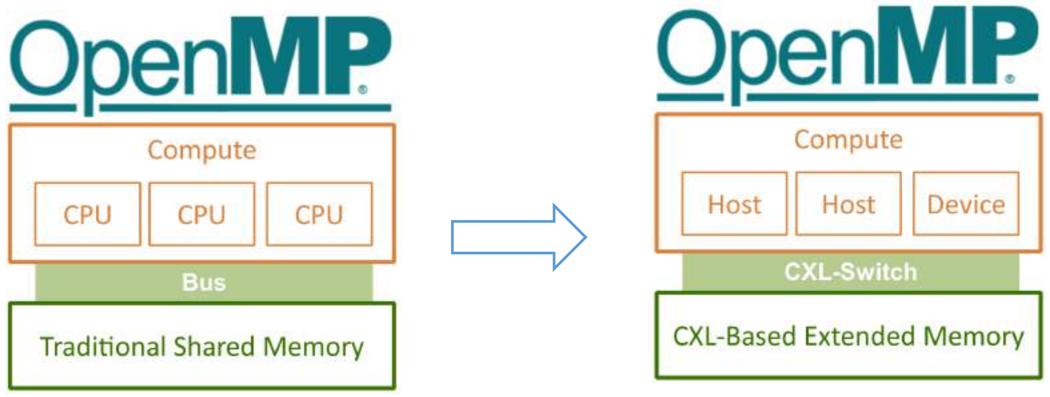




Application for Big Memory (cont.)

- Develop new applications
  - Eased programming models, via simplified mem management
  - Mostly use load/store for data access, in-mem data exchange
- Migrate legacy applications
  - Compiling and runtime support to translate the deprecated usages. E.g.,
    - Get rid of explicit data copies, e.g., cudaMemcpyDeviceToHost()
    - IO and RDMA to load/store

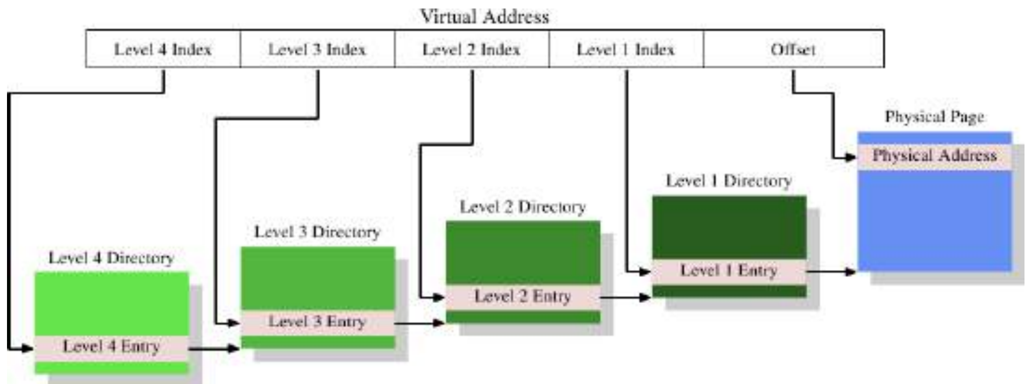


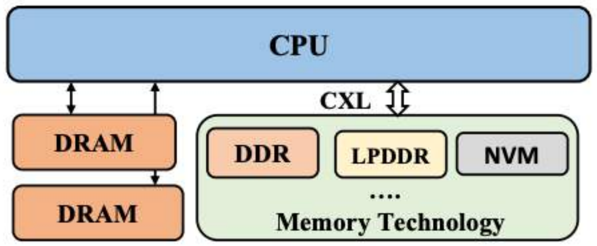


Manage the Big Memory

- Q: How to manage the huge size memory?

- Current virtual memory system and allocators are insufficient
  - 4KB paging? Too fine-grained, intolerable translation overheads
  - 4MB paging? Too coarse-grained, wasted memory space
- Memory heterogeneity should be well utilized
  - DRAM: low latency, high bandwidth, high cost
  - NVM: high latency, low bandwidth, low cost





Manage the Big Memory (cont.)

- File-system based management
  - Memory-semantic file access
  - Transparent hybrid memory allocation
  - Shared memory between process and file system
    - In-situ data processing within file system

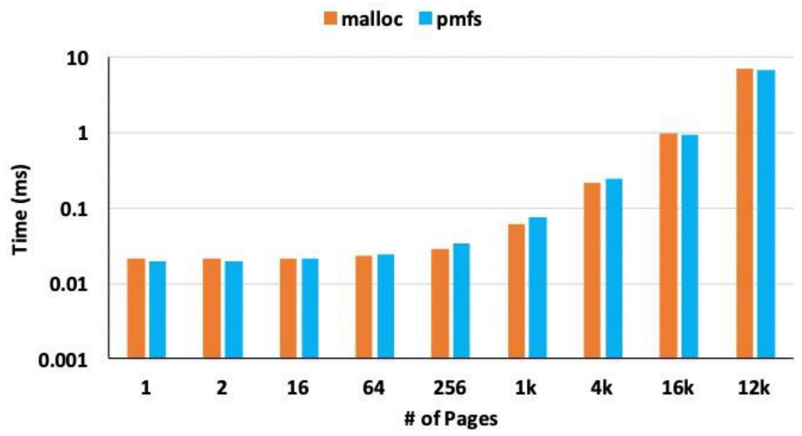


Figure 7: Time to allocate memory pages using anonymous memory (malloc) and memory mapping a file in PMFS.

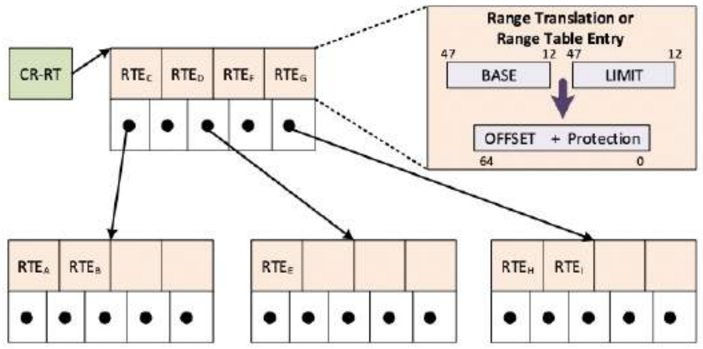


Figure 9: Range table and entries

Access Delay

- Q: How to effectively use w.r.t the access delay?

- CXL bus is slower than DRAM
  - Inter-node connection can be even slower with more nodes
    - Switch, signal skew
  - Applications have varying delay tolerance
    - Some may seriously suffer from performance loss

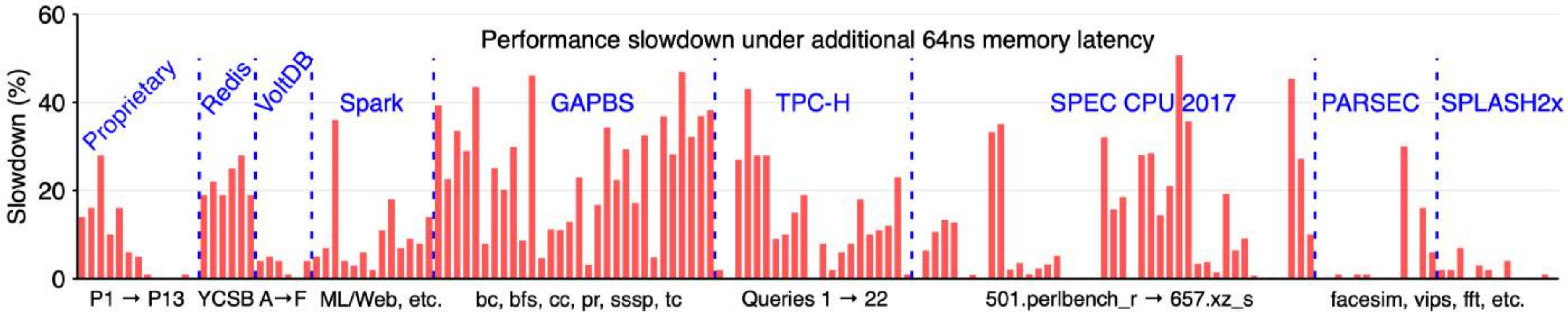
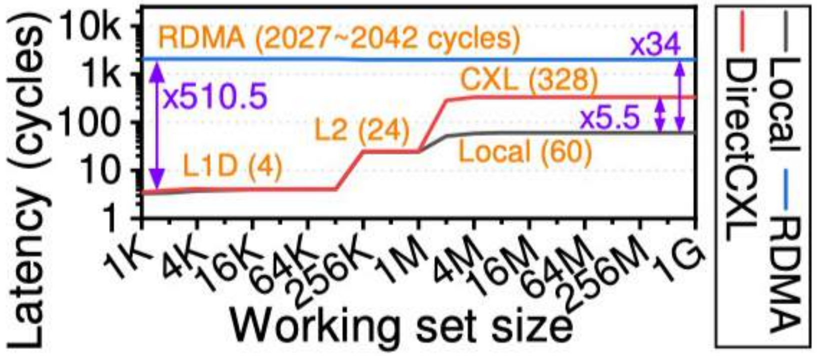
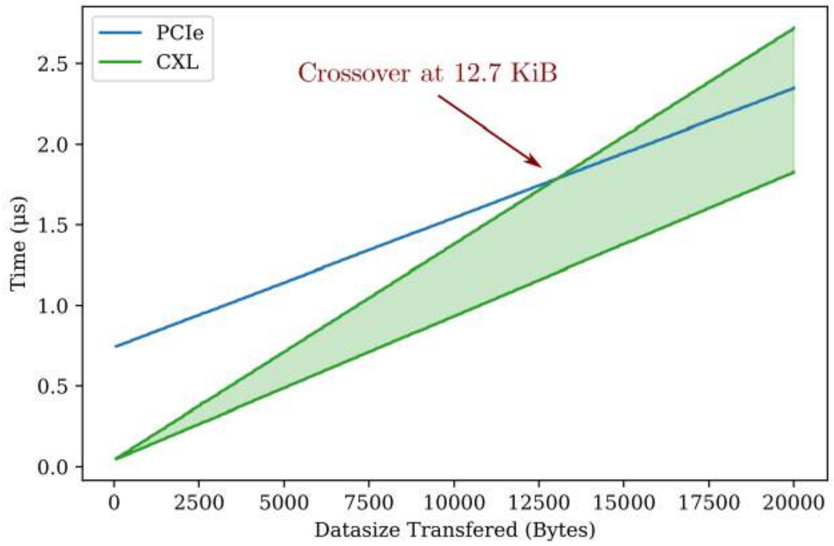


Figure 2: **Workload performance slowdown under additional 64ns memory latency (§3.3).** *This graph shows the performance*

Access Delay (cont.)

- Granularity-aware access
  - The high-speed link favors fine-grained data transfer
  - For larger size, PCIe or RDMA can still be a better choice
- NUMA- and application-aware data placement
  - Local or remote?
  - Sensitive or not?

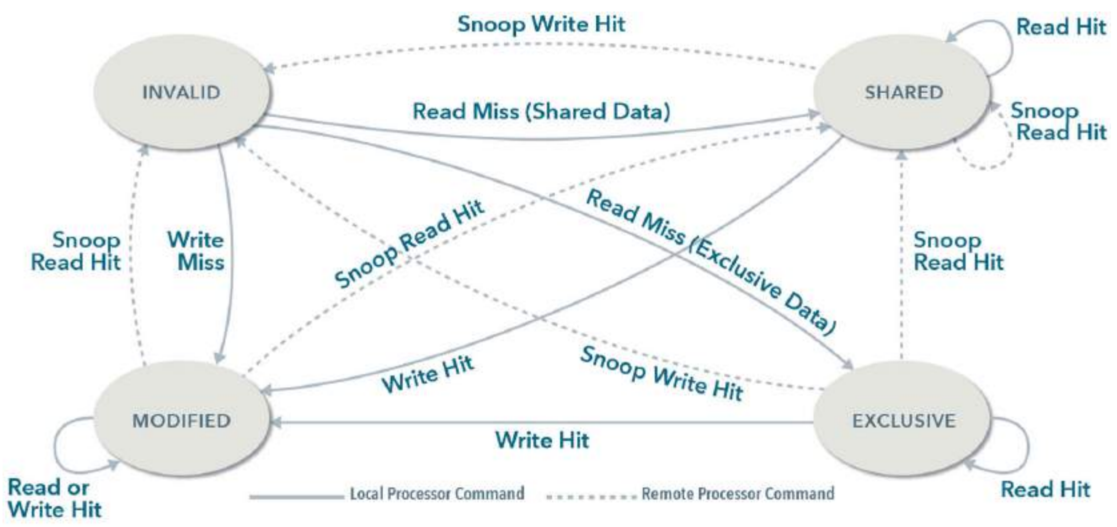




Coherence Guarantee

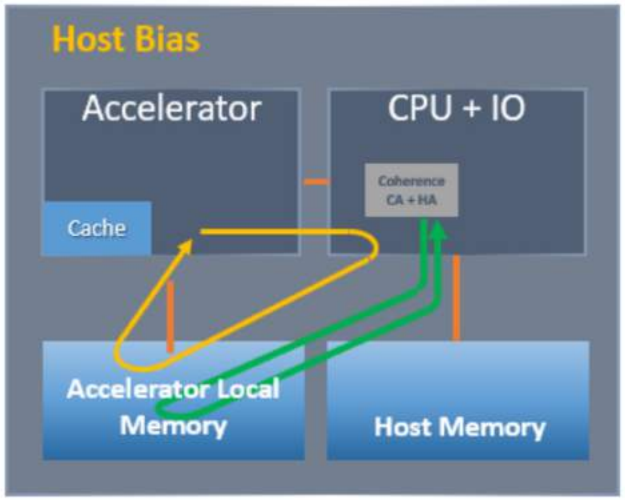
- Q: How to guarantee coherence in an efficient way?

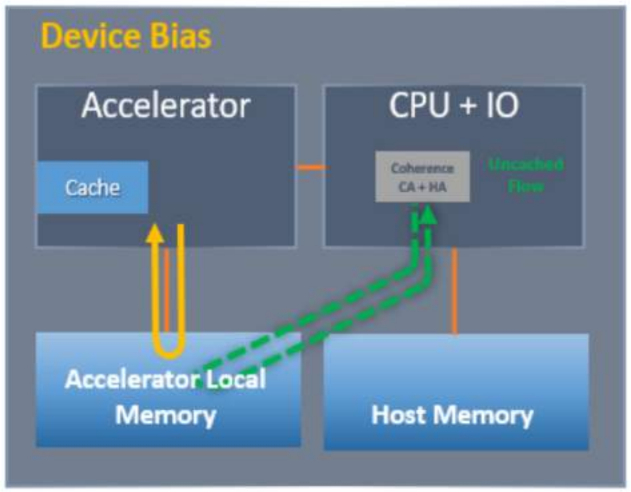
- Coherence overhead is high among massive entities
  - Strict coherence requires the data update to be seen by all
    - May lead to frequent invalidations
  - The overhead can be unacceptably high in large data center



Coherence Guarantee (cont.)

- Coherence bias
  - Allows a device to access its memory coherently without visiting the processor
    - Device bias: pages being used exclusively by the device
    - Host bias: pages being used by the host or shared between host and device
  - Zone-based coherence
    - Strict coherence in small zones, use SW to control coherence in large regions

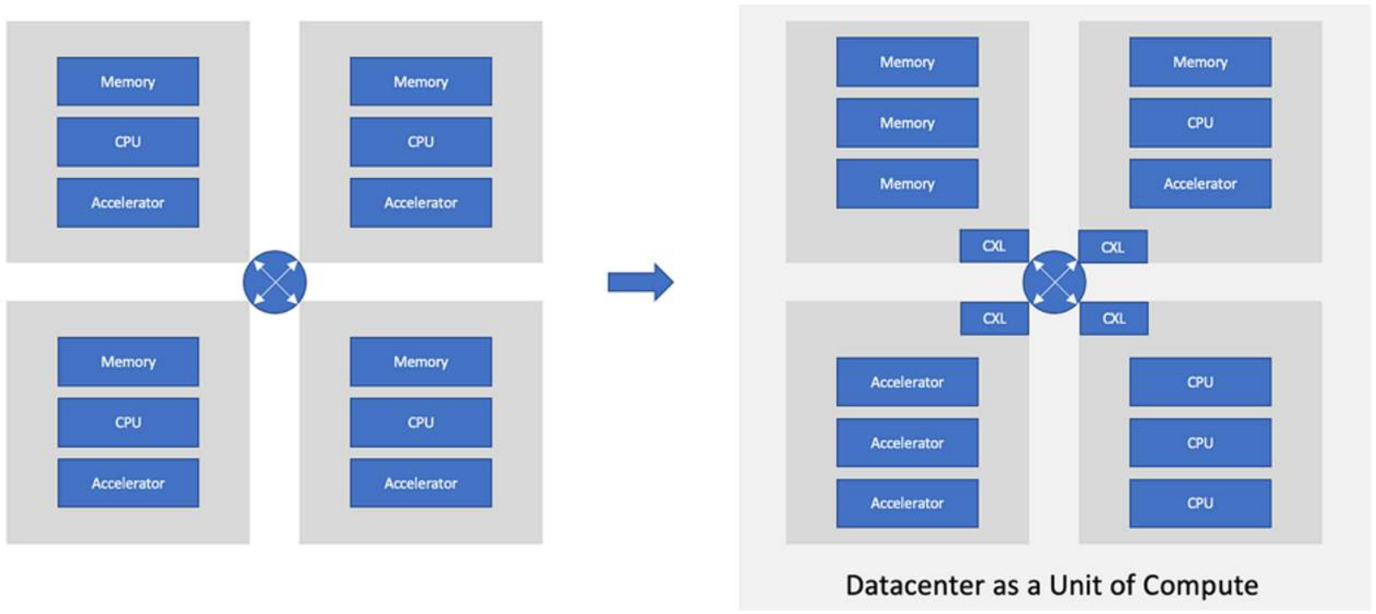




Disaggregated Resources

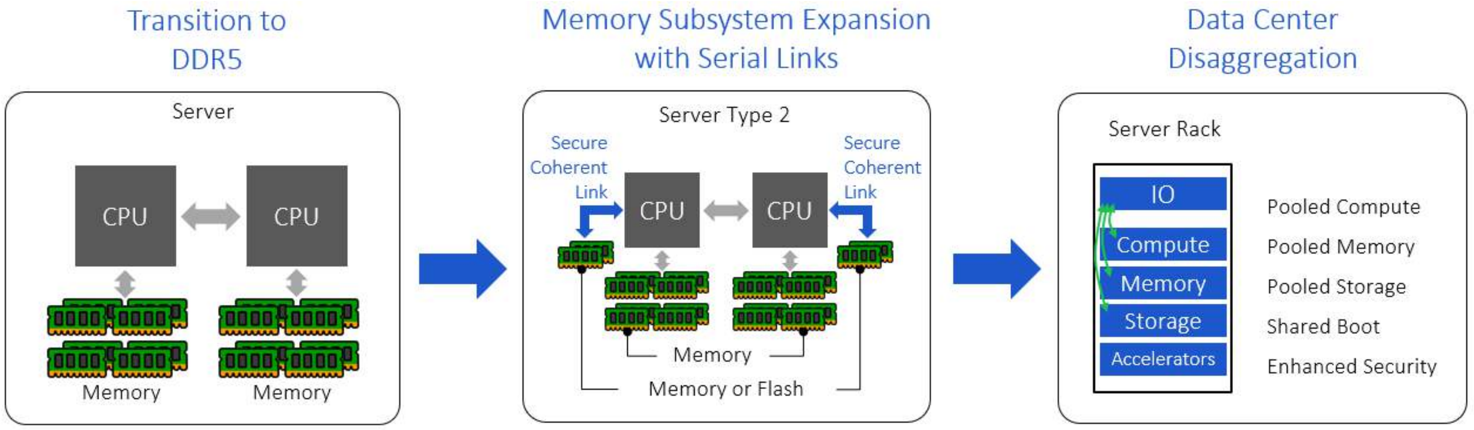
- Q: How to configure and adjust the composable resources?

- No one-size-fits-all setting
  - Applications have varying resource demands
  - Demanded resources are dynamically changing



Disaggregated Resources (cont.)

- Compiler or loader estimates the compute and memory demands
  - Compose the initial resources for the application
- Runtime monitors the dynamic behaviors
  - Adjust the resource amounts
- Intertwining effects should be taken into account
  - Application characteristics are varying
    - Co-existing or conflicting?



OUTLINE

SUN YAT-SEN UNIVERSITY

Summary

- Ever high demands on memory capacity and bandwidth
  - CXL-based memory pooling is the promising solution
- CXL-based memory pooling is the promising direction
  - Unified memory, easing programming burden
  - High-speed comparable to memory bus, enabling memory semantic access
  - Also allows resource disaggregation and composition
- Open issues to be explored
  - Application porting
  - Software-level management
  - Performance optimizations (latency, coherence, etc)
  - …