Master Informatics Eng.

2018/19

A.J.Proença

Data Parallelism 3 (*GPU/CUDA, Neural Nets, ...*)

(most slides are borrowed)

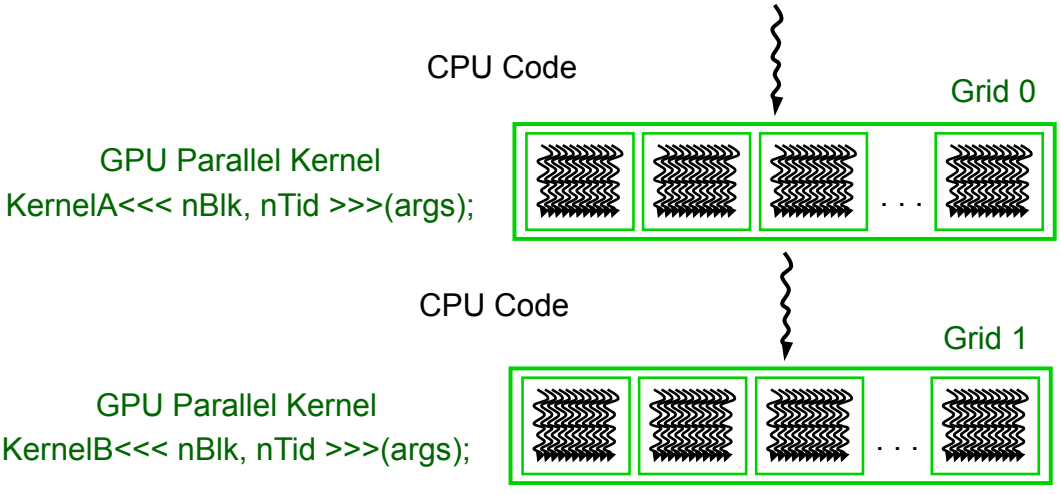
- ***C**ompute **U**nified **D**evice **A**rchitecture*
- CUDA is a recent programming model, designed for
  - a multicore CPU ***host*** coupled to a many-core ***device***, where
  - *devices* have wide SIMD/SIMT parallelism, and
  - the *host* and the *device* do not share memory
- CUDA provides:
  - a thread abstraction to deal with SIMD
  - synchr. & data sharing between small groups of threads
- CUDA programs are written in C with extensions
- OpenCL inspired by CUDA, but hw & sw vendor neutral
  - programming model essentially identical

CUDA Devices and Threads

- A compute device
  - is a coprocessor to the CPU or host
  - has its own DRAM (device memory)
  - runs many threads in parallel
  - is typically a GPU but can also be another type of parallel processing device
- Data-parallel portions of an application are expressed as device kernels which run on many threads - **SIMT**
- Differences between GPU and CPU threads
  - GPU threads are extremely lightweight
    - very little creation overhead, requires LARGE register bank
  - GPU needs 1000s of threads for full efficiency
    - multi-core CPU needs only a few

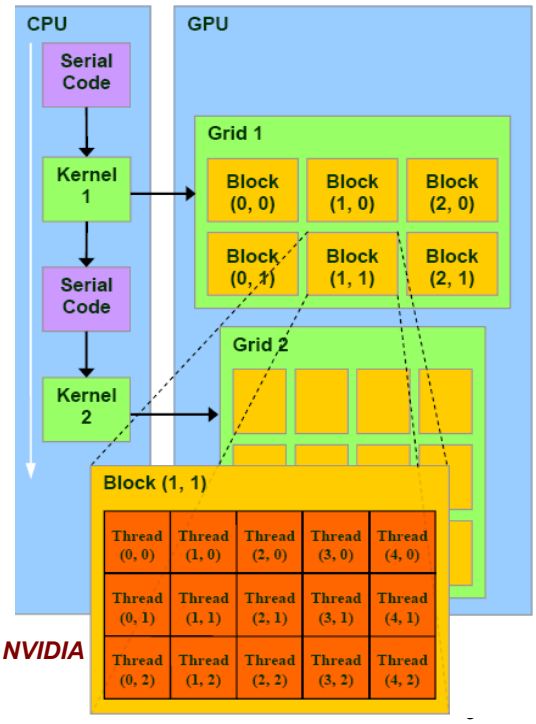
CUDA basic model: Single-Program Multiple-Data (SPMD)

- CUDA integrated CPU + GPU application C program
  - Serial C code executes on CPU
  - Parallel Kernel C code executes on GPU thread blocks



Programming Model: SPMD + SIMT/SIMD

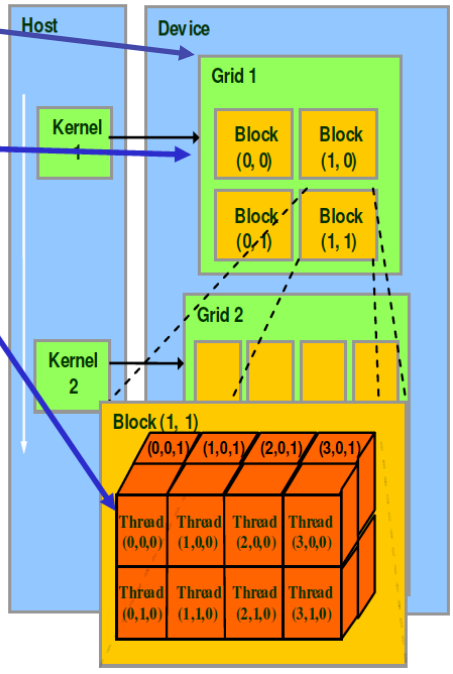
- Hierarchy
  - Device => Grids
  - Grid => Blocks
  - Block => Warps
  - Warp => Threads
- Single kernel runs on multiple blocks (SPMD)
- Threads within a warp are executed in a lock-step way called single-instruction multiple-thread (SIMT)
- Single instruction are executed on multiple threads (SIMD)
  - Warp size defines SIMD granularity (32 threads)
- Synchronization within a block uses shared memory



Courtesy NVIDIA

The Computational Grid: Block IDs and Thread IDs

- A kernel runs on a computational grid of thread blocks
  - Threads share global memory
- Each thread uses IDs to decide what data to work on
  - Block ID: 1D or 2D
  - Thread ID: 1D, 2D, or 3D
- A thread block is a batch of threads that can cooperate by:
  - Sync their execution w/ barrier
  - Efficiently sharing data through a low latency shared memory
  - Two threads from two different blocks cannot cooperate



Example

- Multiply two vectors of length 8192
  - Code that works over all elements is the grid
  - Thread blocks break this down into manageable sizes
    - 512 threads per block
  - SIMD instruction executes 32 elements at a time
  - Thus grid size = 16 blocks
  - Block is analogous to a strip-mined vector loop with vector length of 32
  - Block is assigned to a *multithreaded SIMD processor* by the *thread block scheduler*
  - Current-generation GPUs (Fermi) have 7-16 multithreaded SIMD processors

C with CUDA Extensions: C with a few keywords

```
void saxpy_serial(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}
// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

Standard C Code

```
__global__ void saxpy_parallel(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)  y[i] = a*x[i] + y[i];
}
// Invoke parallel SAXPY kernel with 256 threads/block
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

Parallel C Code

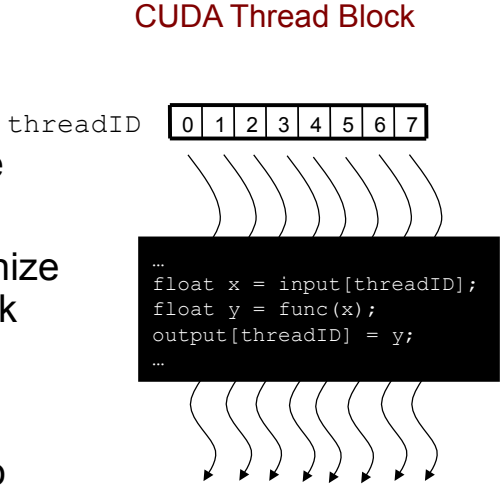
NVIDIA Confidential

Terminology *(and in NVidia)*

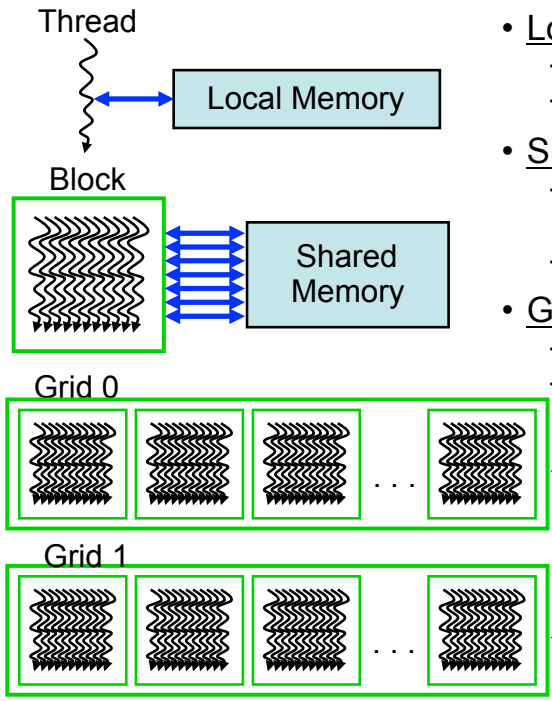
- Threads of SIMD instructions *(**warps**)*
  - Each has its own IP (up to 48/64 per SIMD processor, Fermi/Kepler)
  - Thread scheduler uses scoreboard to dispatch
  - No data dependencies between threads!
  - Threads are organized into blocks & executed in groups of 32 threads *(**thread block**)*
    - Blocks are organized into a grid
- The thread block scheduler schedules blocks to SIMD processors *(**Streaming Multiprocessors**)*
- Within each SIMD processor:
  - 32 SIMD lanes *(**thread processors**)*
  - Wide and shallow compared to vector processors

CUDA Thread Block

- Programmer declares (Thread) Block:
  - Block size 1 to **512** concurrent threads
  - Block shape 1D, 2D, or 3D
  - Block dimensions in threads
- All threads in a Block execute the same thread program
- Threads share data and synchronize while doing their share of the work
- Threads have thread id numbers within Block
- Thread program uses thread id to select work and address shared data



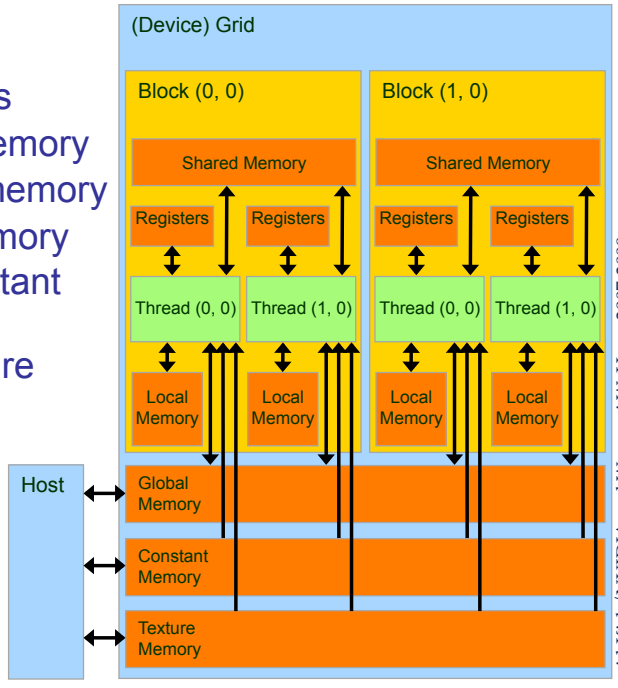
Parallel Memory Sharing



- Local Memory: **per-thread**
  - Private per thread
  - Auto variables, register spill
- Shared Memory: **per-block**
  - Shared by threads of the same block
  - Inter-thread communication
- Global Memory: **per-application**
  - Shared by all threads
  - Inter-Grid communication

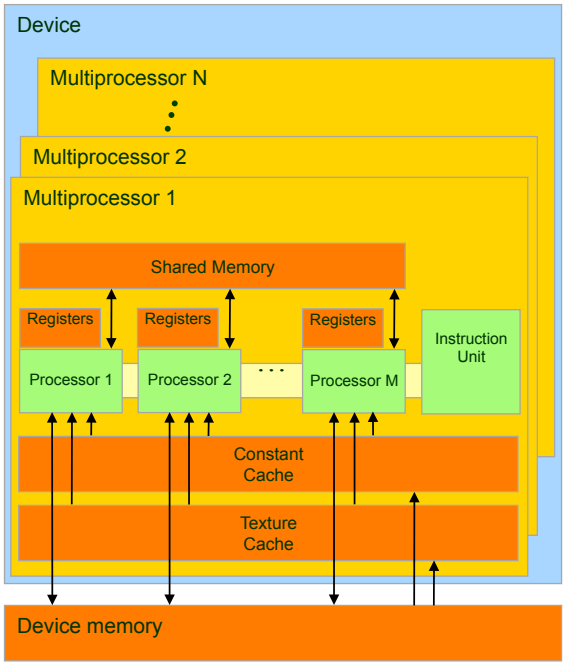
CUDA Memory Model Overview

- Each thread can:
  - R/W per-thread registers
  - R/W per-thread local memory
  - R/W per-block shared memory
  - R/W per-grid global memory
  - Read only per-grid constant memory
  - Read only per-grid texture memory
- The host can R/W global, constant, and texture memories



Hardware Implementation: Memory Architecture

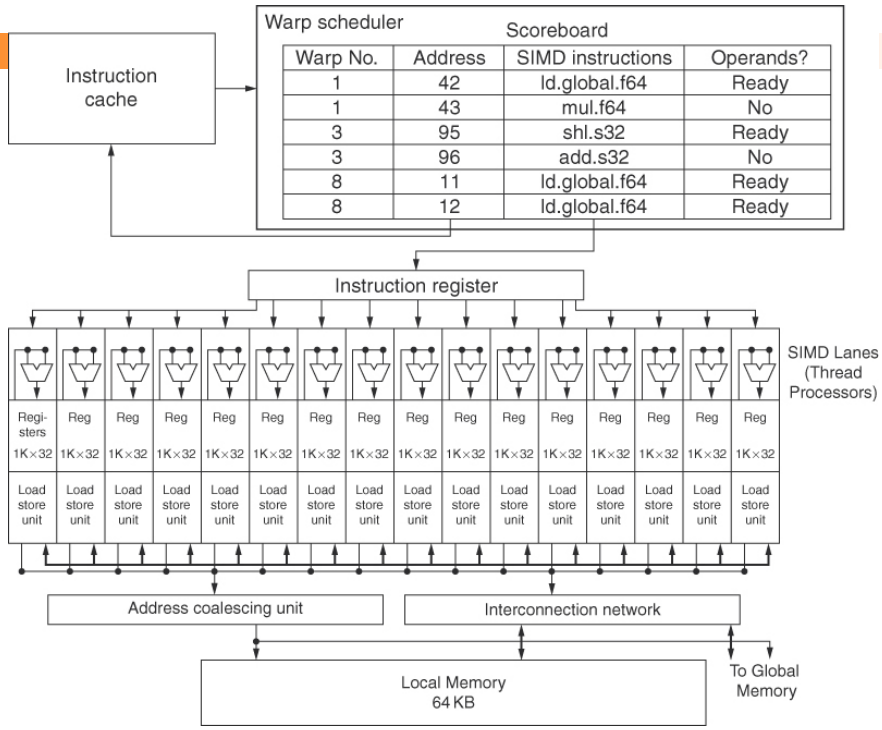
- Device memory (DRAM)
  - Slow (2~300 cycles)
  - Local, global, constant, and texture memory
- On-chip memory
  - Fast (1 cycle)
  - Registers, shared memory, constant/texture cache



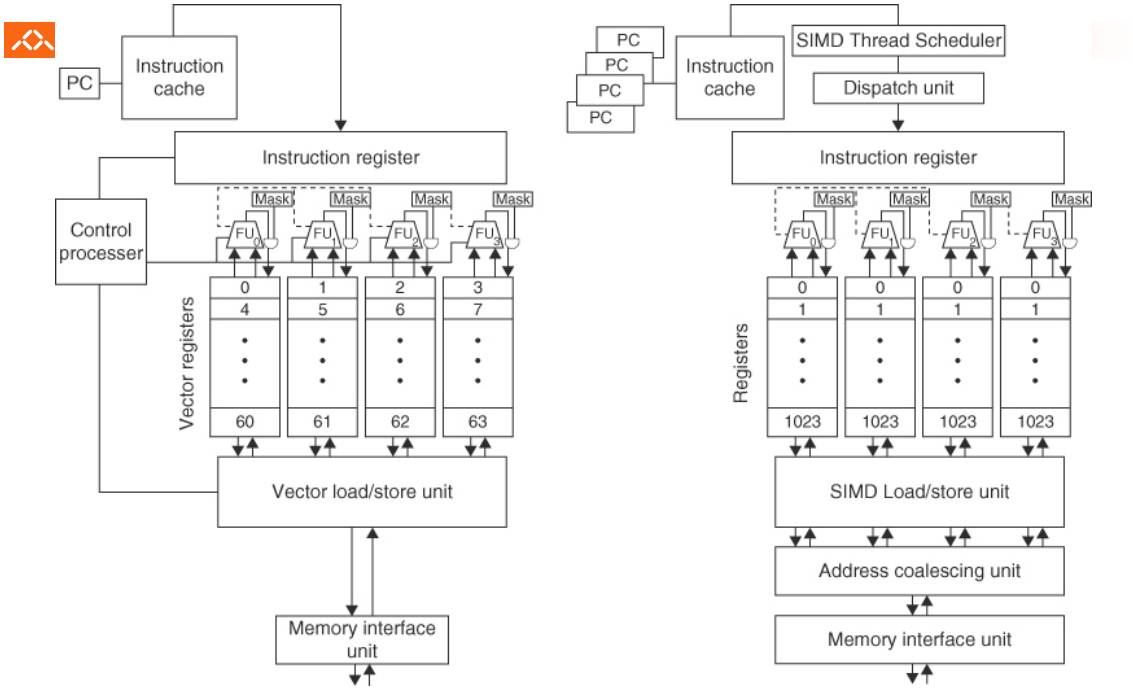
Courtesy NVIDIA

Graphical Processing Units

Example



Vector Processor versus CUDA core



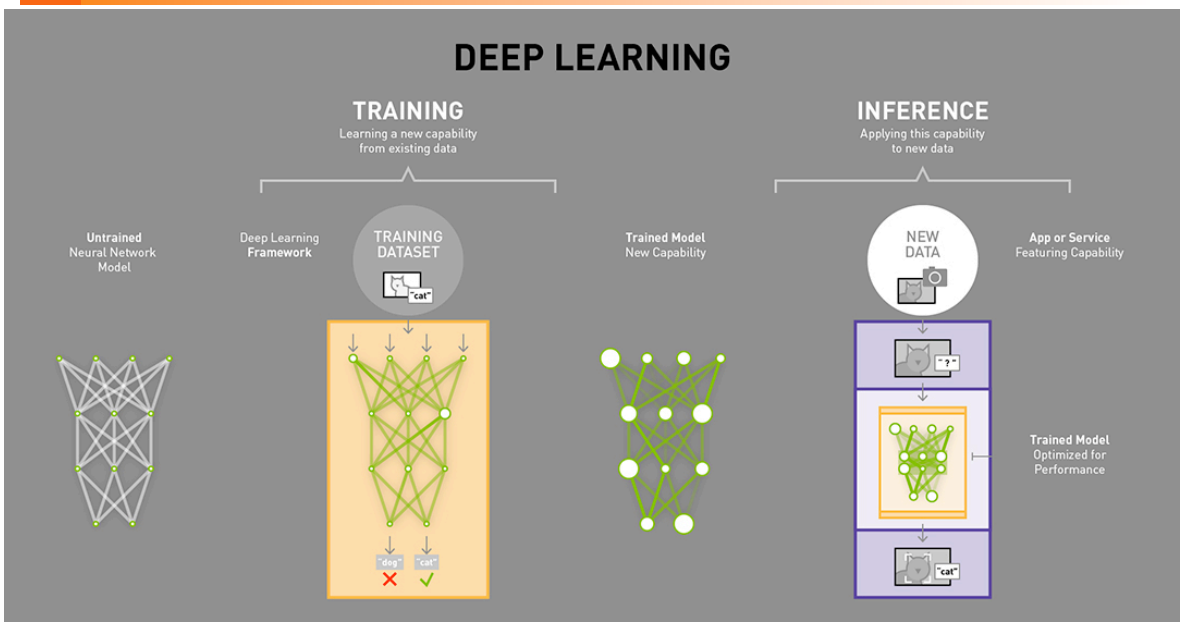
Conditional Branching

- Like vector architectures, GPU branch hardware uses internal masks
- Also uses
  - Branch synchronization stack
    - Entries consist of masks for each SIMD lane
    - I.e. which threads commit their results (all threads execute)
  - Instruction markers to manage when a branch diverges into multiple execution paths
    - Push on divergent branch
  - …and when paths converge
    - Act as barriers
    - Pops stack
- Per-thread-lane 1-bit predicate register, specified by programmer

Beyond Vector/SIMD architectures

- Vector/SIMD-extended architectures are hybrid approaches
  - mix (super)**scalar + vector** op capabilities on a single device
  - **highly pipelined** approach to reduce memory access penalty
  - **tightly-closed access to shared memory**: lower latency
- Evolution of Vector/SIMD-extended architectures
  - **PU (Processing Unit) cores with wider vector units**
    - x86 many-core: **Intel** MIC / Xeon KNL (*more slides later*)
    - other many-core: **IBM** Power BlueGene/Q Compute, **ShenWay** 260
  - **coprocessors (require a host scalar processor): accelerator devices**
    - on disjoint physical memories (e.g., **Xeon KNC** with PCI-Expr, **PEZY-SC**)
    - ISA-free architectures, code compiled to silica: **FPGA**
    - focus on SIMT/SIMD to hide memory latency: **GPU**-type approach
    - focus on tensor/neural nets cores: **NVidia**, **IBM**, **Intel** NNP, **Google** TPU
  - **heterogeneous PUs in a SoC: multicore PUs with GPU-cores**
    - …

Machine learning w/ neural nets & deep learning...



Key algorithms to train & classify use matrix products,
but require lower precision numbers!

NVidia Volta Architecture: the new Tensor Cores

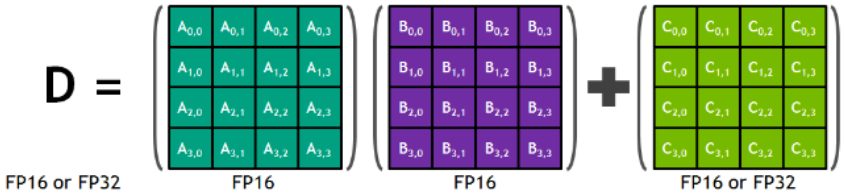


Figure 8. Tensor Core 4x4 Matrix Multiply and Accumulate

For each SM:
8x 64 FMA ops/cycle
1k FLOPS/cycle!

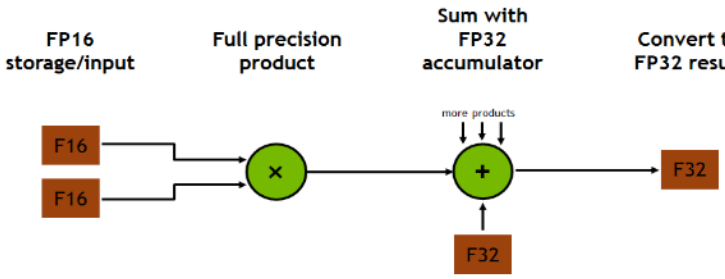


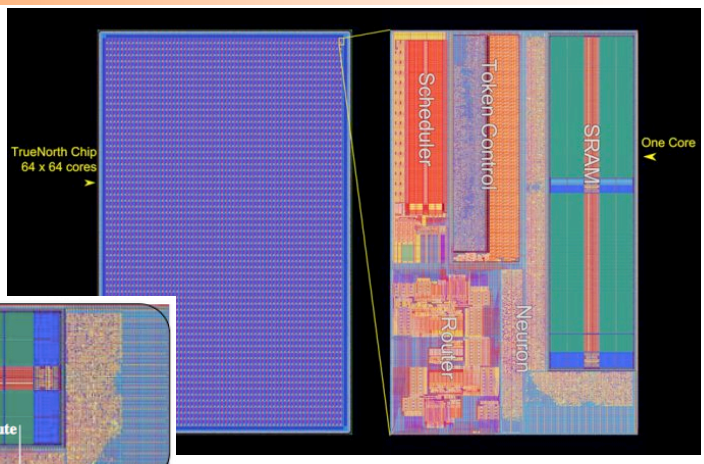
Figure 9. Mixed Precision Multiply and Accumulate in Tensor Core

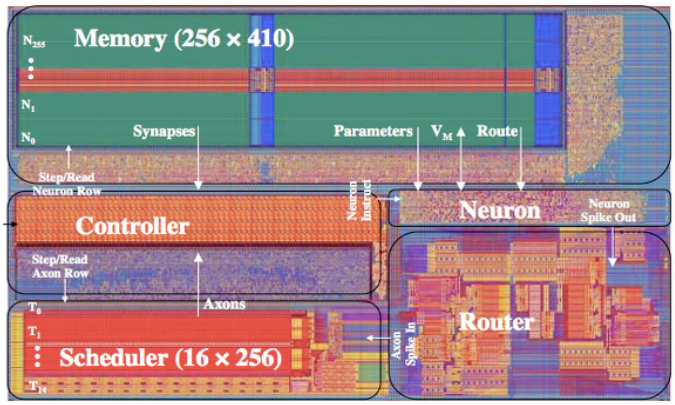
http://www.nvidia.com/content/gated-pdfs/Volta-Architecture-Whitepaper-v1.1.pdf

NVidia competitors with neural net features: IBM TrueNorth chip array (August'2014)

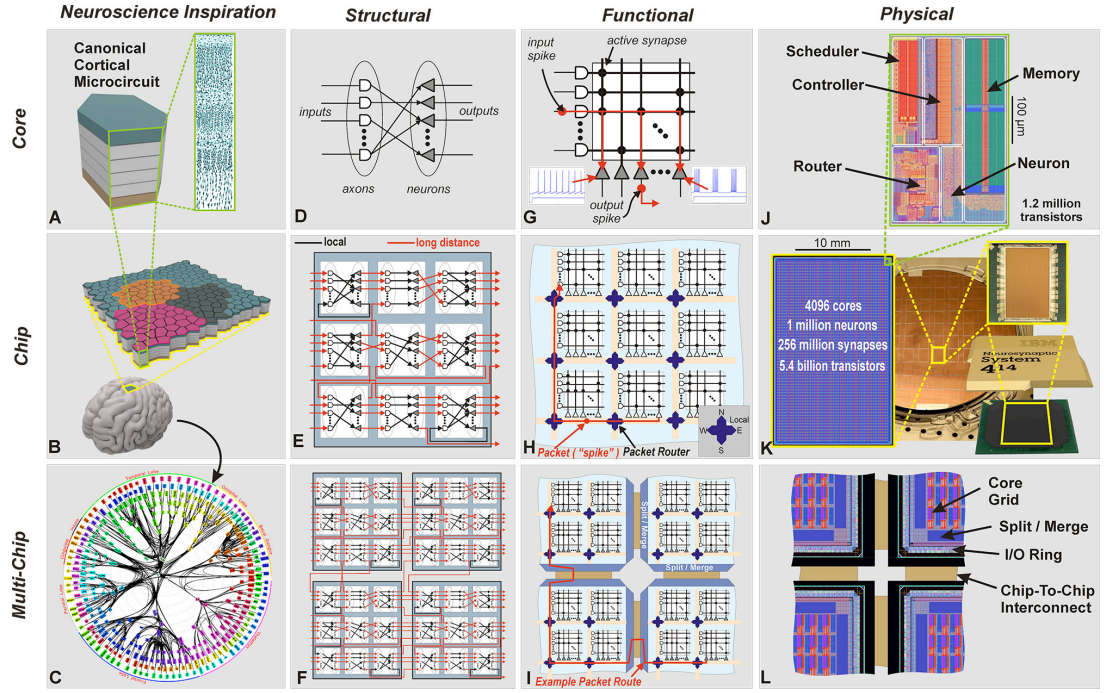
TrueNorth Chip:
- 4096 neurosynaptic cores

Each core:
- 256 inputs (axons)
- 256 outputs (neurons)
- RAM w/ data for each neuron
- router (any neuron to any axon)





NVidia competitors with neural net features: the IBM TrueNorth architecture

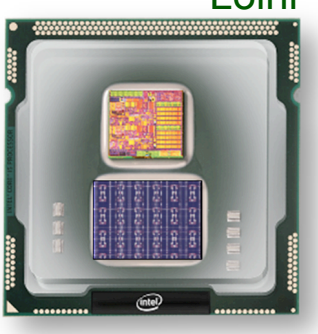


NVidia competitors with neural net features: Intel Nervana Neural Network Processor, NNP

History

- Nervana Engine announced in May'16
- Key features:
  - ASIC chip, focused on matrix multiplication,convolutions,... (for neural nets)
  - HBM2: 4x 8GB in-package storage & 1TB/sec memory access b/w
  - no h/w managed cache hierarchy (saves die area, higher compute density)
  - built-in networking (6 bi-directional high-b/w links)
  - separate pipelines for computation and data management
  - proprietary numeric format Flexpoint
    in-between floating point and fixed point precision
- Nervana acquired by Intel in August 2016:
  - renamed the project to "Lake Crest"
  - later to Nervana NNP, launched in October'17
  - Loihi test chip w/ self-learning capabilities
    announced in Sept'17, to be launched in 2018

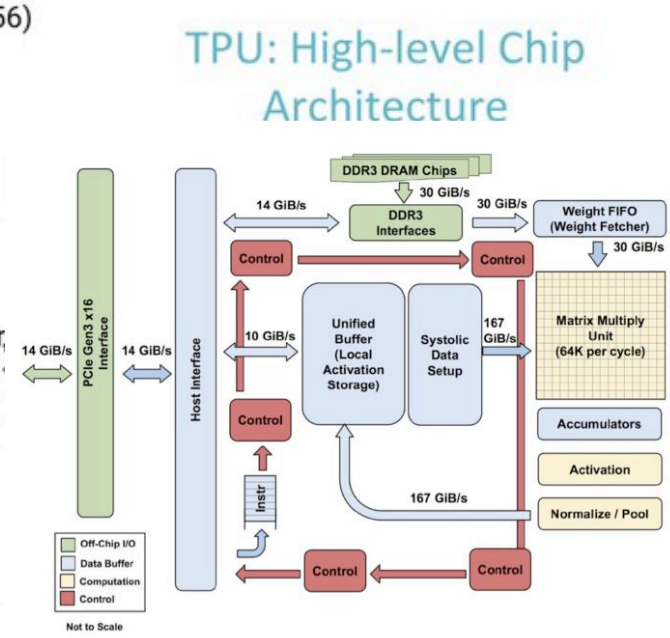
Loihi



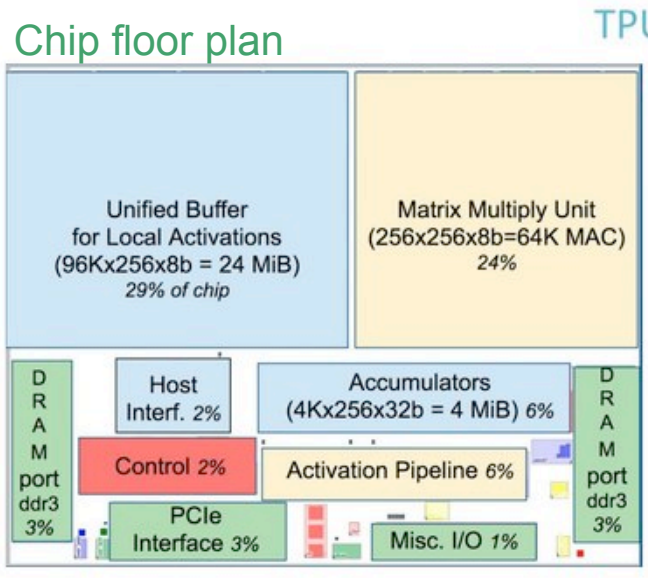
https://www.top500.org/news/intel-will-ship-first-neural-network-chip-this-year/

NVidia competitors with neural net features: Google Tensor Processing Unit, TPU *(April'17)*

- 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



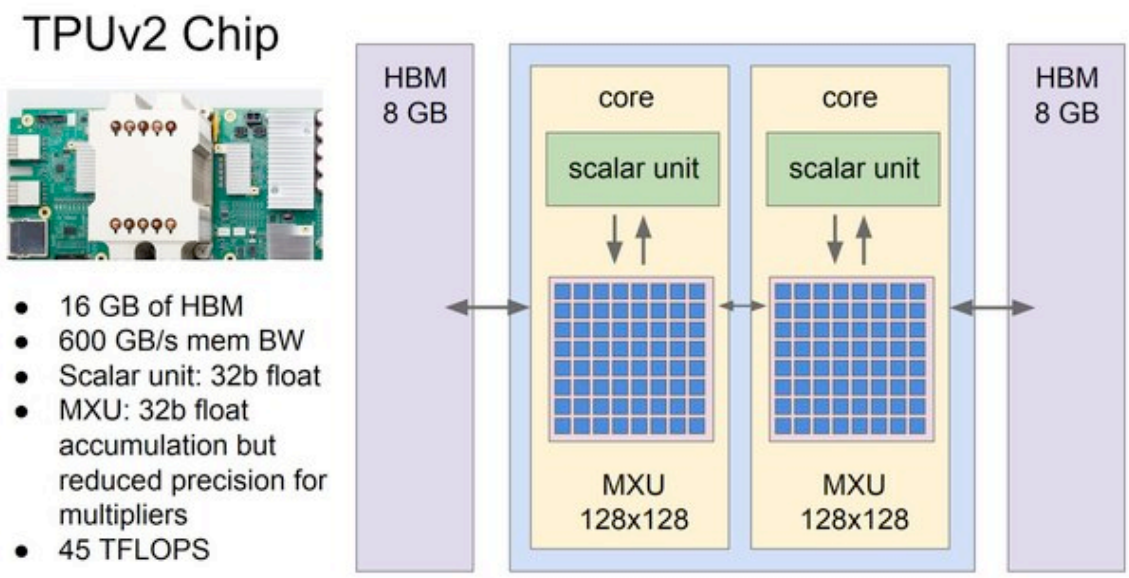
NVidia competitors with neural net features: Google Tensor Processing Unit, TPU *(April'17)*



TPUs are intensively used by Google, namely in RankBrain, StreetView & Google Translate

https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

NVidia competitors with neural net features: Google TPUv2 (September'17)



Beyond Vector/SIMD architectures

- Vector/SIMD-extended architectures are hybrid approaches
  - mix (super)**scalar + vector** op capabilities on a single device
  - **highly pipelined** approach to reduce memory access penalty
  - **tightly-closed access to shared memory**: lower latency
- Evolution of Vector/SIMD-extended architectures
  - **PU (Processing Unit) cores with wider vector units**
    - x86 many-core: **Intel** MIC / Xeon KNL
    - other many-core: **IBM** Power BlueGene/Q Compute, **ShenWay** 260
  - **coprocessors (require a host scalar processor): accelerator devices**
    - on disjoint physical memories (e.g., **Xeon KNC** with PCI-Expr, **PEZY-SC**)
    - ISA-free architectures, code compiled to silica: **FPGA**
    - focus on SIMT/SIMD to hide memory latency: **GPU**-type approach
    - focus on tensor/neural nets cores: **NVidia**, **IBM**, **Intel** NNP, **Google** TPU
  - **heterogeneous PUs in a SoC: multicore PUs with GPU-cores**
    - x86 multicore coupled with SIMT/SIMD cores: **Intel** i5/i7
    - ARMv8 cores coupled with SIMT/SIMD cores: **NVidia** Tegra