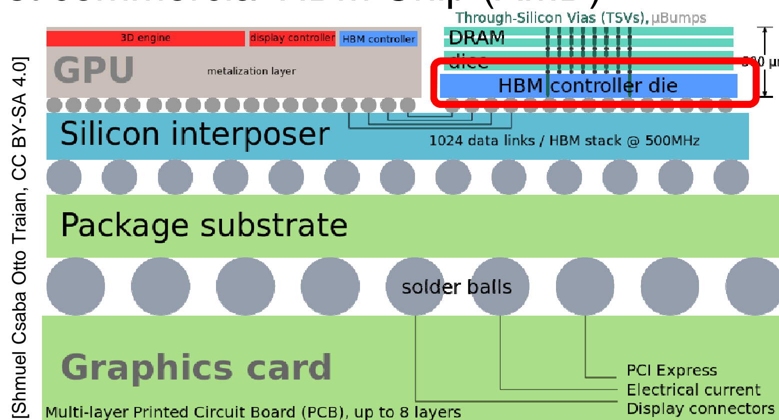
NTX: A Scalable Near-Memory Architecture for Training Deep Neural Networks on Large In-Memory Datasets

Luca Benini[1,2], Fabian Schuiki[1]

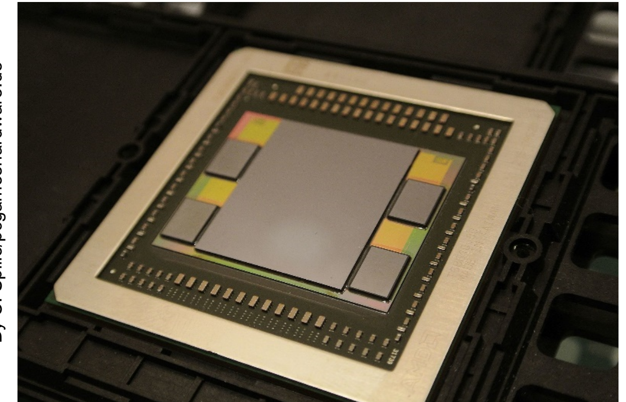
[1]ETH Zurich, [2]University of Bologna

Opportunity – 3D Memories are happening!

- **2011**: heterogeneous 3D-integration
  - 3D Memory Stacking
- **2012**: Hybrid Memory Cube (HMC) Proposal by **Micron**
- **2013**: HMC Consortium ➔ HMC V1.0 Specifications
  - **Flexible** and **Abstracted** Serial Interface
- **2015**: First commercial HBM Chip (AMD)



[Shmuel Csaba Otto Traian, CC BY-SA 4.0]

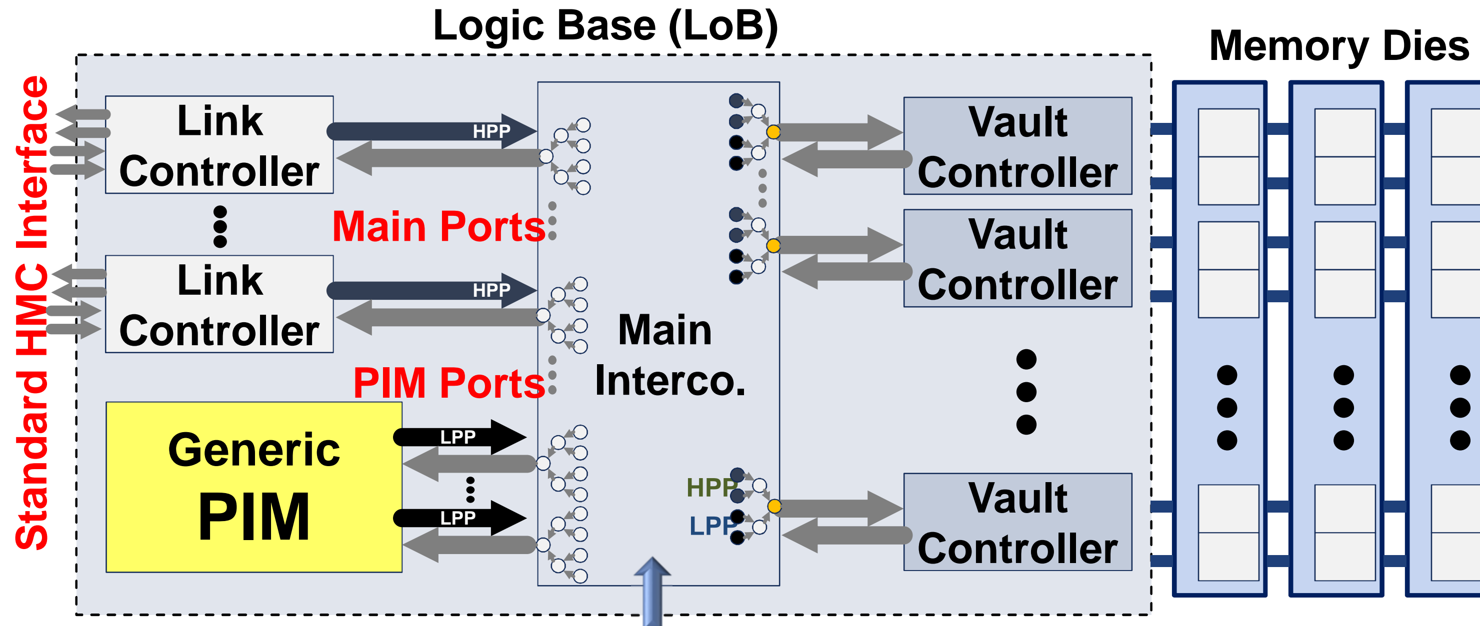


By C. Spille/pcgameshardware.de

- Revisit Near Memory Computation
  - In the HMC's (or HBM's) **controller die**

➔ We are exploring the "**Smart Memory Cube (SMC)**" concept

The Smart Memory Cube (SMC)



Featuring a specialized interconnect on the Logic Base (LoB)

- Providing sufficient bandwidth to the main links
- Extra bandwidth to any generic Processor in Memory (PIM)

Rationale

Literature:

1. **Latency reduction:** vicinity to the main storage
2. Higher **bandwidth:** TSVs instead of Pins

- Most recent works exploit motivation (2): **BANDWIDTH**

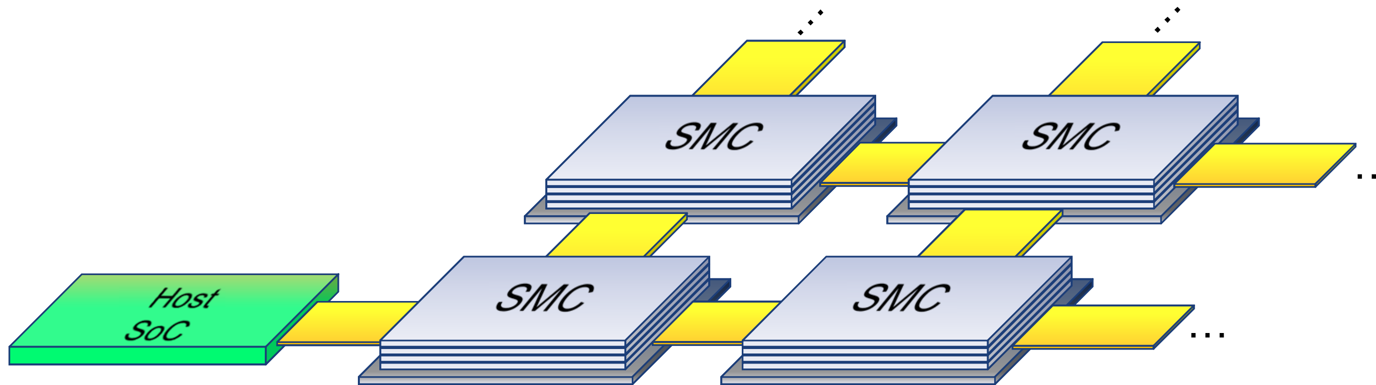
  [Ahn, ISCA'15], [Sura, CF'15], [Zhang, HPDC'14], [Islam, Euro-Par'14], …
- However, motivation (2) is not valid in HMC
- HMC Can deliver **all its internal bandwidth** to the outside world
- In current HMC the advantage of **PIM** over the **external world** is **vicinity** to the memory (**lower** access **latency and energy**) and **not** higher bandwidth.

(1) $E = E_{MEM} + E_{LINK}$

We will show that, **even in this case**, PIM **is highly competitive**

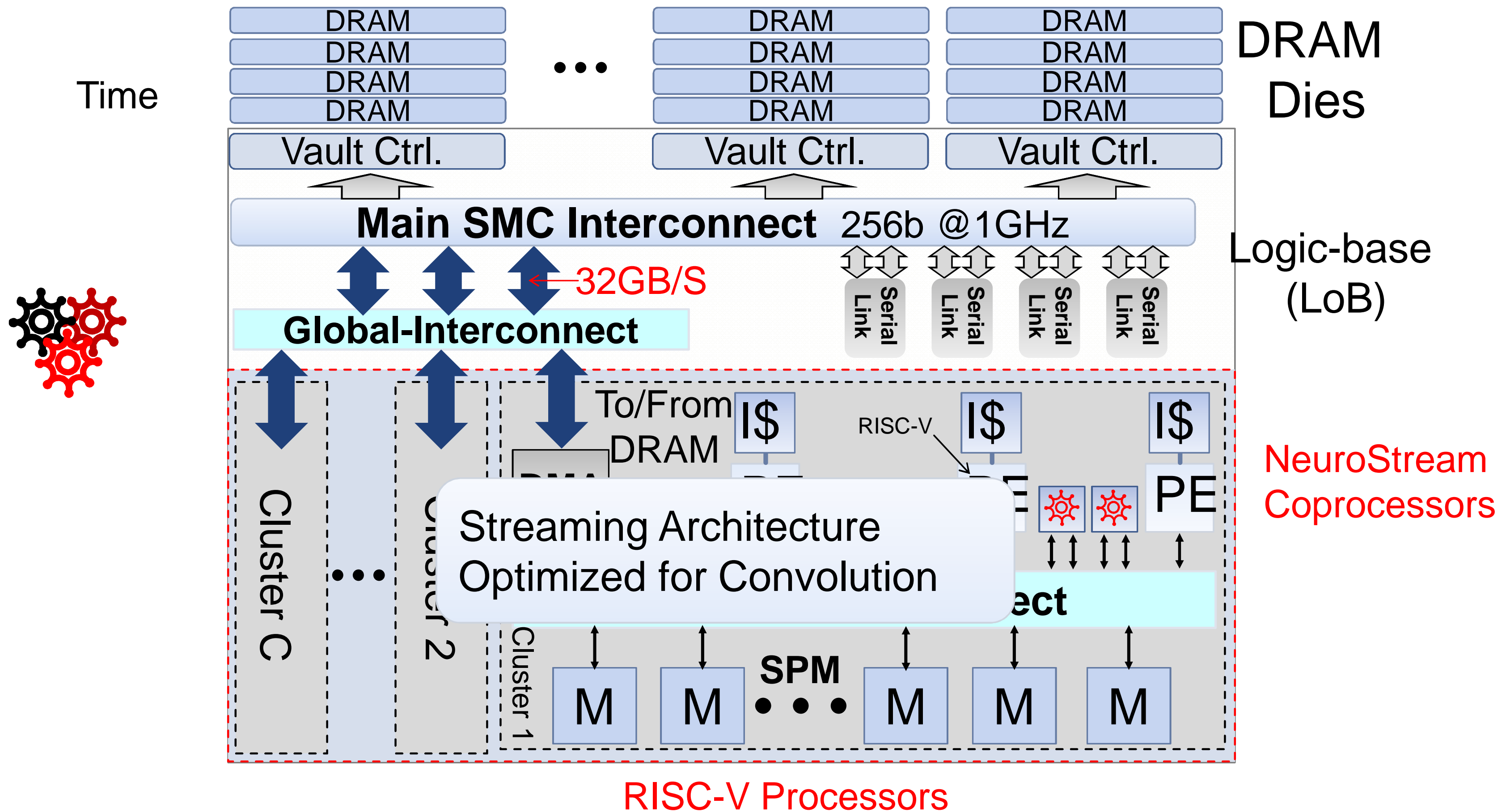
(2) $M_{BUF} = bw\ W_T$

Scalability



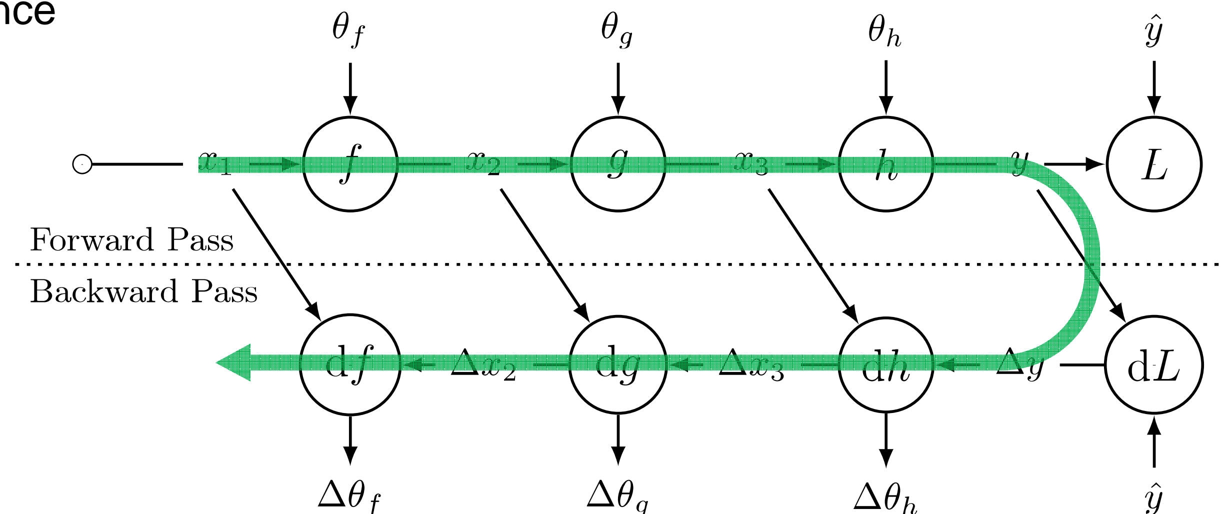
- Scalability potential for PIM
  - Much larger aggregate bandwidth seen by “distributed” PIMs
  - **E.g. each cube work on one video frame**
  - Power management in serial links
    - **Only active when necessary**

PIM for Machine Learning: NeuroStreams



Neurostream

- **Inference**
  - Well covered, but "low" memory requirements, FP precision not needed, very well-suited for systolic architecture
  - Compute layer, only keep results, advance
- **Training**
  - **Long data dependency chain**
  - Intermediate results must be stored
  - Cannot fuse ReLU with convolution
  - Derivatives tricky (ReLU, Maxpool)
  - FP precision is required

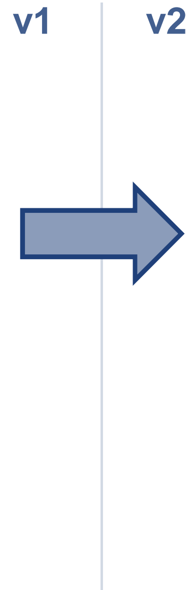


- **Offloading**
  - Only 3 nested hardware loops
  - Convolution needs **6** loops
  - Processor must issue many small operations

Focus Shift towards Training

Neurostream (NS) [1]

- Inference only
- Uses off-the-shelf ADD/MUL units
  - Limited control on internals
  - Slow, critical path in the FPU
- High control overhead
  - 3 nested hardware loops
  - 2 address generators
- 2 NST per processor core
- Requires specifically arranged data

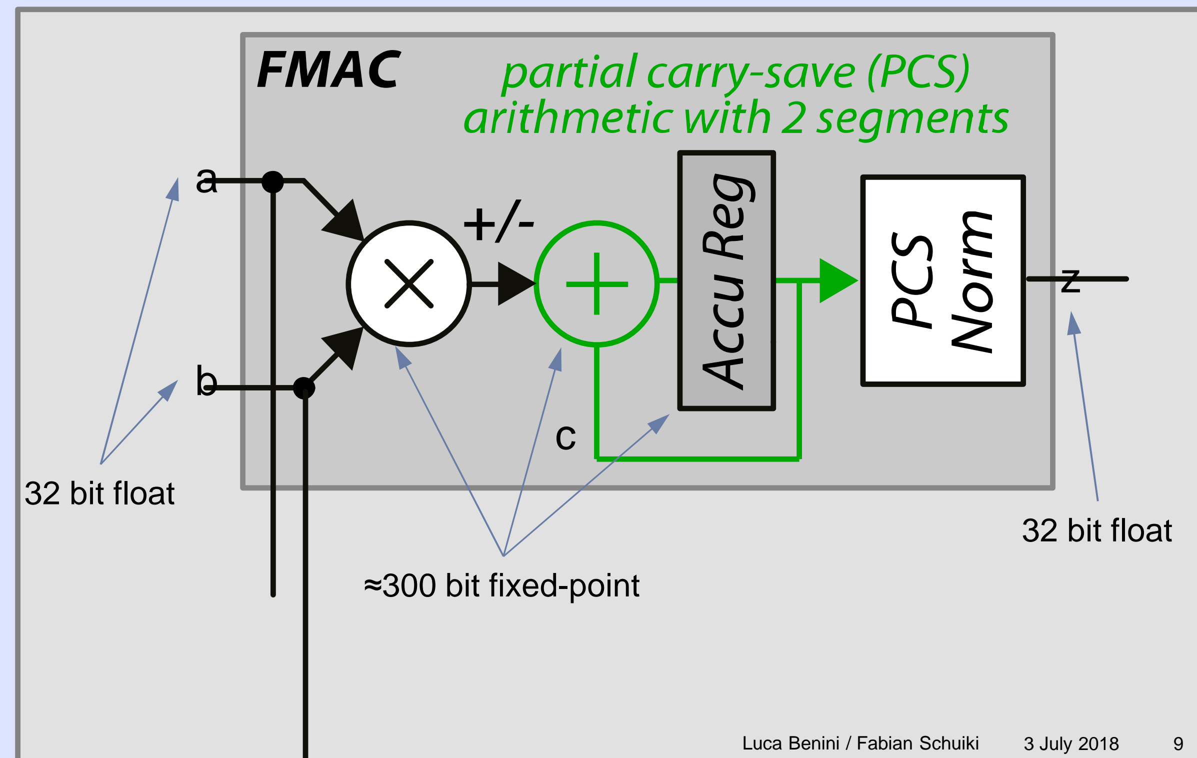


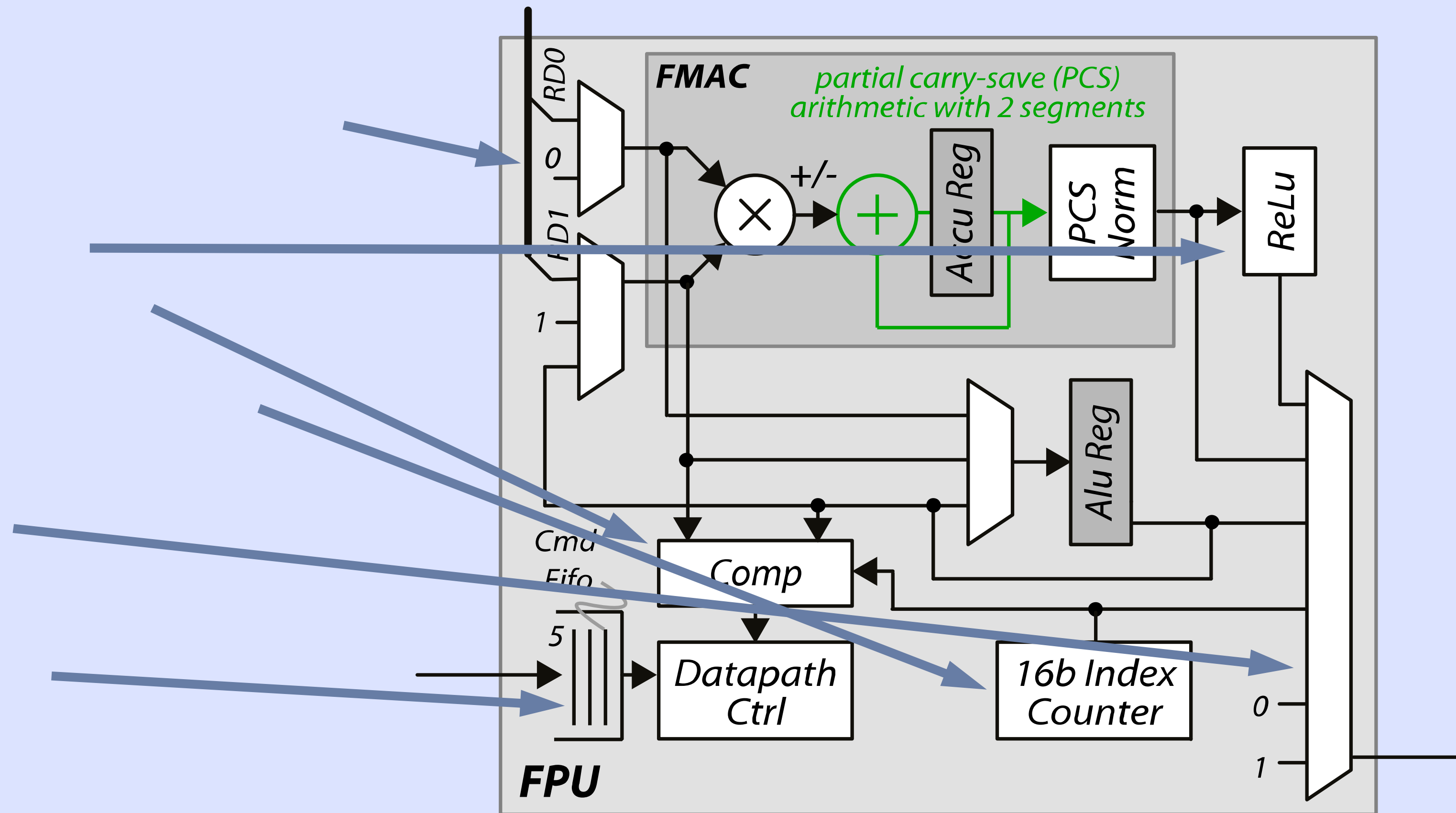
Network Training Accelerator (NTX) [2]

- Inference and **training**
- Uses **custom FMAC unit**
  - Full control on internals
  - Fast
- **Low control overhead**
  - 5 nested hardware loops
  - 3 address generators
- **8 NST** per processor core
- **No data layout** requirement

[1] Azarkhish et al, "Neurostream: Scalable and energy efficient deep learning with smart memory cubes," in IEEE TPDS 2018.
[2] Schuiki et al, "A Scalable Near-Memory Architecture for Training Deep Neural Networks on Large In-Memory Datasets," to be published.

Architecture FMAC



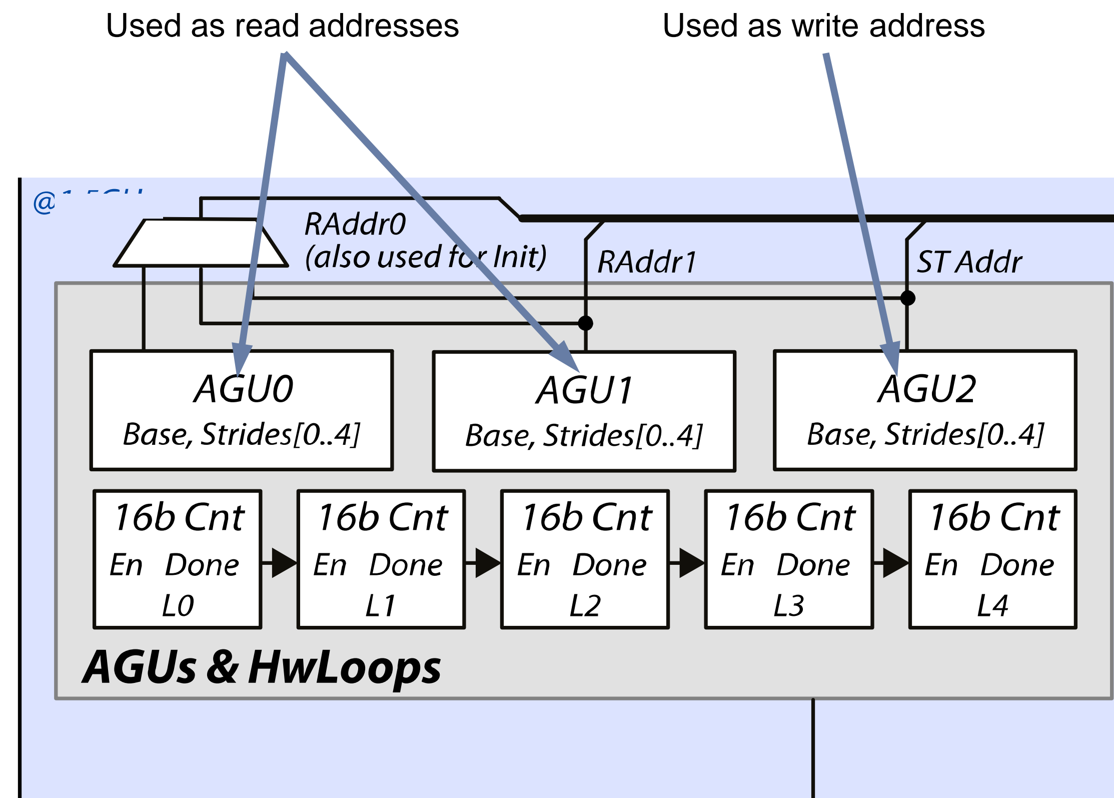

FMAC
partial carry-save (PCS)
arithmetic with 2 segments
RD0
RD1
0
1
+/-
Accu Reg
PCS
Norm
ReLu
Alu Reg
Cmd
Fifo
5
Comp
Datapath
Ctrl
16b Index
Counter
0
1
FPU

Architecture Address Generation

- 5 nested hardware loop counters
  - 16 bit counter register
  - Configurable number of iterations
  - Once last iteration reached:
    - Reset counter to 0
    - Enable next counter for one cycle

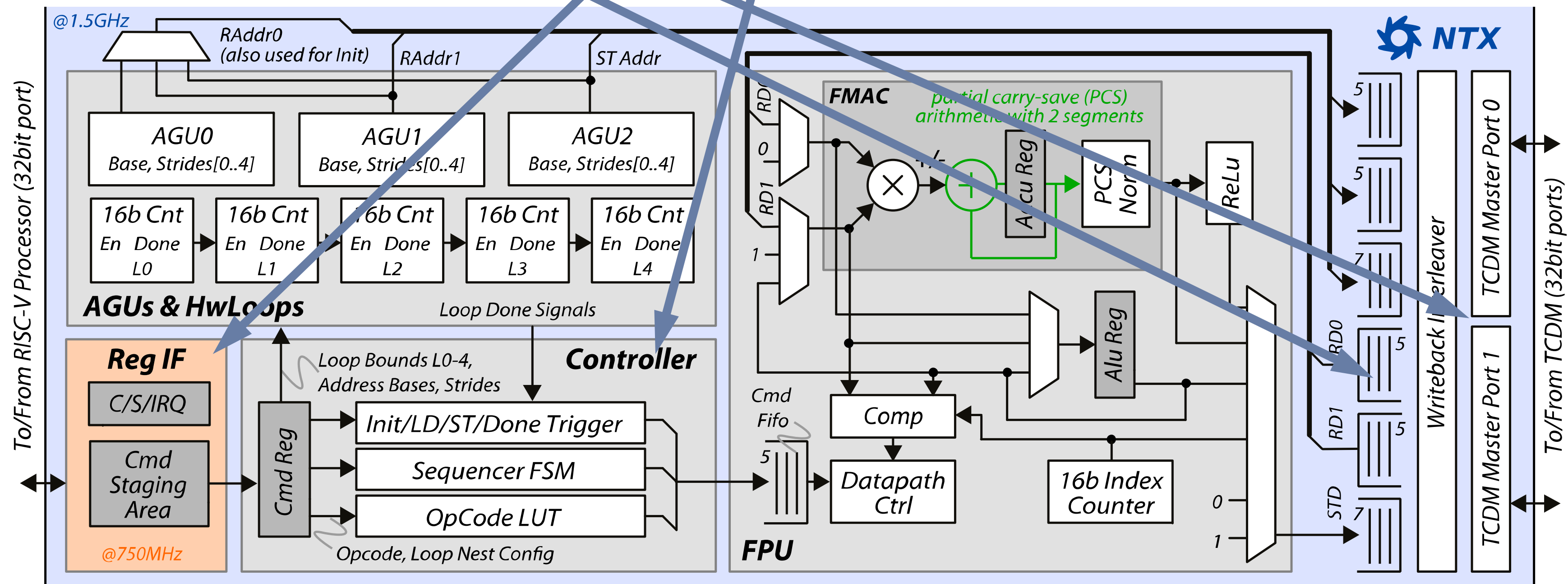
- 3 address generation units
  - 32 bit address register
  - Each has 5 configurable strides, one per loop
  - One stride added to register per cycle
  - Stride corresponds to the highest enabled loop

- Allows for complex address patterns



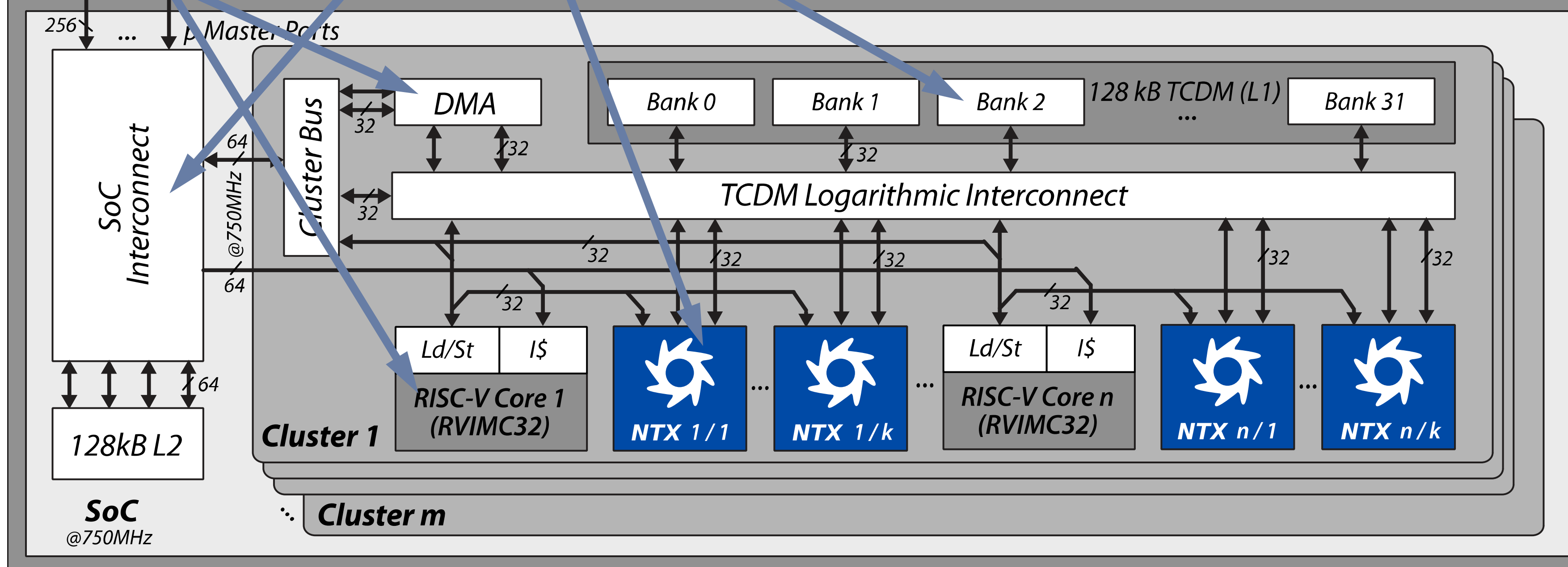
Architecture Coprocessor

- Processor configures operation via **memory-mapped registers**
- Controller issues AGU, HWL, and FPU micro-commands based on configuration
- Reads/writes data via **2 memory ports** (2 operand and 1 writeback streams)
- FIFOs help buffer data path and memory latencies



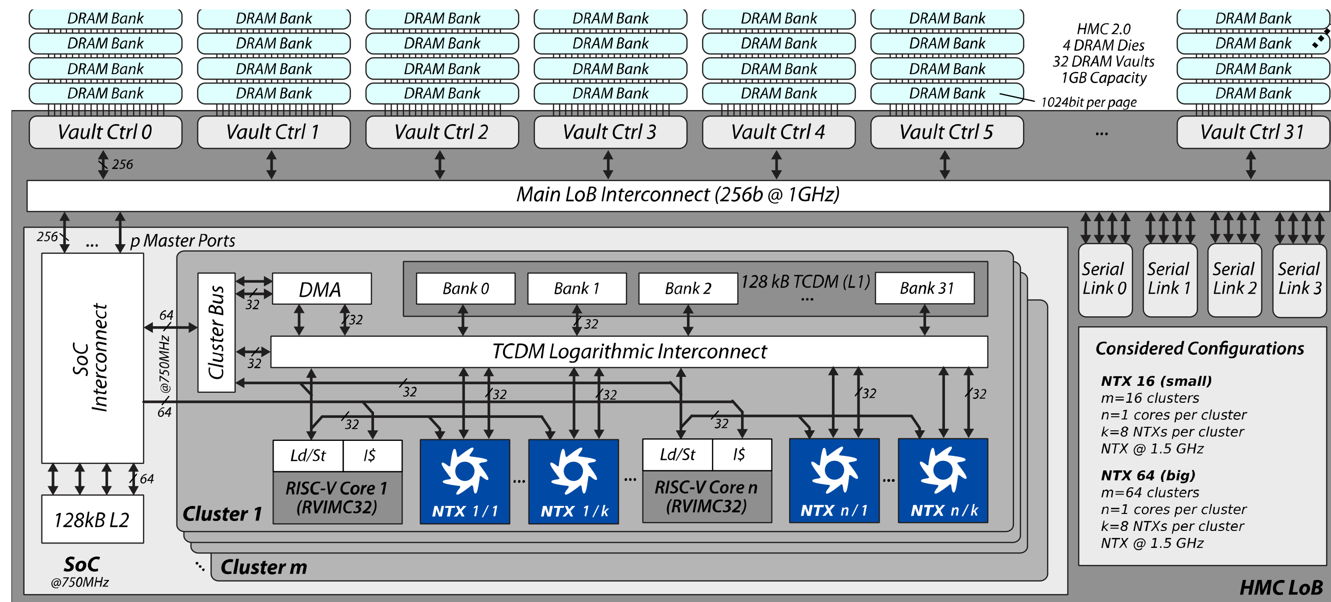
Architecture Processing Cluster

- **1 processor** core controls **8 NTX** coprocessors
- Attached to 128 kB shared **TCDM** via a logarithmic interconnect
- **DMA** engine used to transfer data (double buffering)
- Multiple clusters connected via interconnect (crossbar/NoC)



Architecture HMC Integration

- HMC is split into independent vaults (DRAM controllers)
- Main interconnect routes traffic between serial links and vaults
- Clusters attach to this interconnect
  - Full view of the HMC memory space
  - Access to other HMCs via serial links



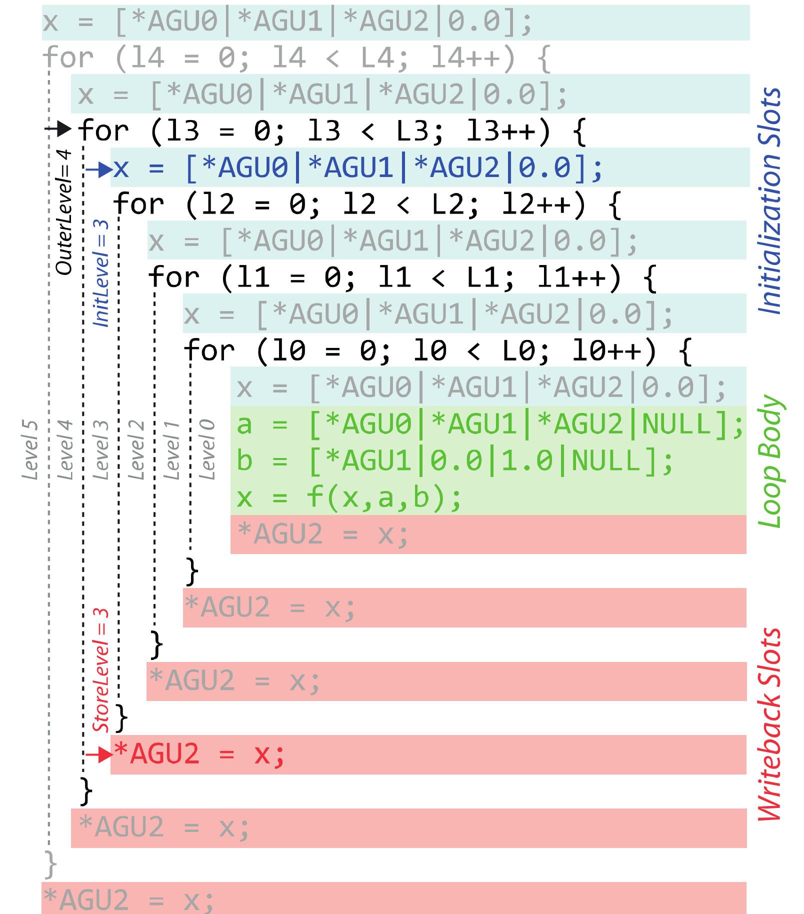
Programming Model Loops

- Up to 5 nested loops can be offloaded to NTX
  - Loops should describe a reduction for best performance
  - Covers convolutions, fully connected layers, and more
- Accumulator initialization and writeback is configurable
- For example a DNN convolution:

Perform outermost loop level on processor core.

```
for (int k = 0; k < K; ++k)
for (int n = 0; n < N; ++n)      Level 4
for (int m = 0; m < M; ++m) {    Level 3
    float a = b[k];              Init Level = 3
    for (int d = 0; d < D; ++d)  Level 2
    for (int u = 0; u < U; ++u)  Level 1
    for (int v = 0; v < V; ++v) {  Level 0
        a += x[d][n+u][m+v] * w[k][d][u][v];
    }
    y[k][n][m] = a;              Store Level = 3
}
```

NTX



Programming Model Tiling

- Cluster has limited memory (~128 kB)
- DNN data sets are usually multiple GB
- **Tile** input data into chunks that fit in 128 kB
- Use **double buffering** to hide latency while NTX are processing current chunk
  - Write back last iteration's result
  - Preload next iteration's input data
- NTX run independently; processor free to orchestrate data movement with the DMA
- Consider the tiled DNN convolution:

```
for (int tk = 0; tk < TK; ++tk)
for (int tn = 0; tn < TN; ++tn)
for (int tm = 0; tm < TM; ++tm) {
  // load tile inputs x, w, b with DMA
  for (int k = 0; k < K; ++k)
  for (int n = 0; n < N; ++n)
  for (int m = 0; m < M; ++m) {
    float a = b[k];
    for (int d = 0; d < D; ++d)
    for (int u = 0; u < U; ++u)
    for (int v = 0; v < V; ++v) {
      a += x[d][n+u][m+v] * w[k][d][u][v];
    }
    y[k][n][m] = a;
  }
  // store tile outputs y
}
```

Iterate over tiles of the input data

Iterate over pixels in the current tile

Perform convolution for current pixel

C++ API Example

Tiled convolution:

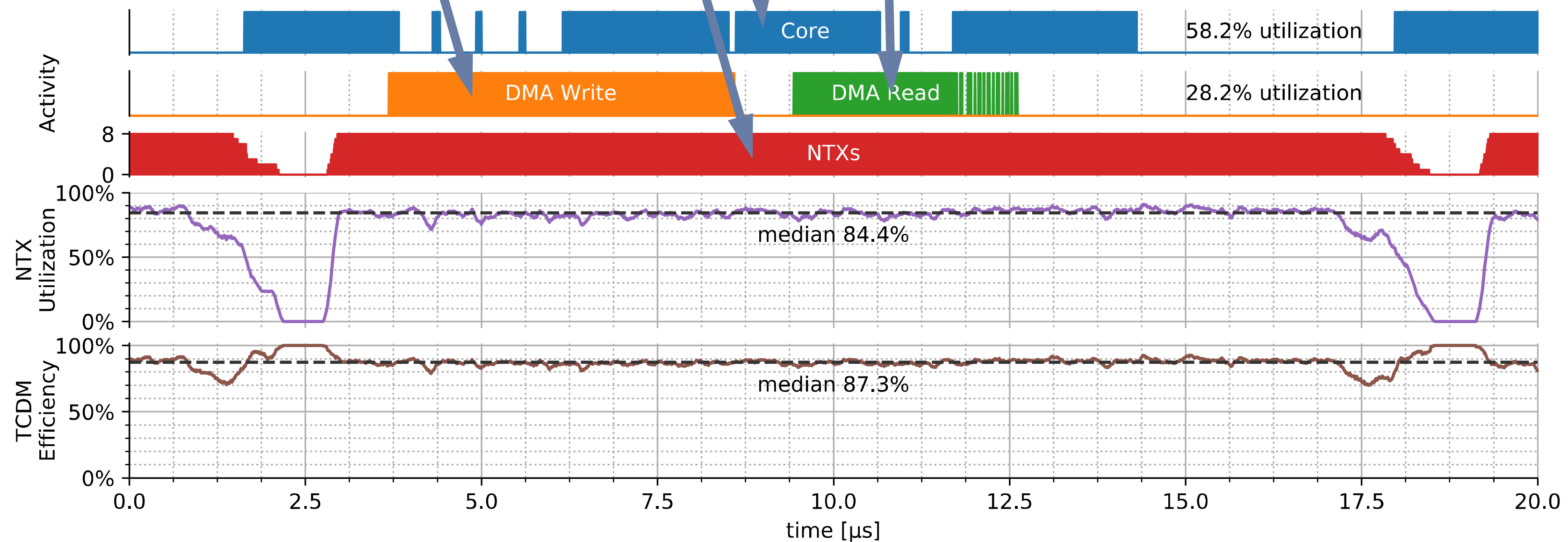
```
for (int tk = 0; tk < TK; ++tk)
for (int tn = 0; tn < TN; ++tn)
for (int tm = 0; tm < TM; ++tm) {
  load_tile(x, w, b);
  for (int k = 0; k < K; ++k)
  for (int n = 0; n < N; ++n)
  for (int m = 0; m < M; ++m) {
    float a = b[k];
    for (int d = 0; d < D; ++d)
    for (int u = 0; u < U; ++u)
    for (int v = 0; v < V; ++v) {
      a += x[d][n+u][m+v] * w[k][d][u][v];
    }
    y[k][n][m] = a;
  }
  store_tile(y);
}
```

Tiled convolution with NTX:

```
ntx_api ntx;
dma_api dma;
ntx.cfg_loops(5, {N,M,D,U,V}, ...);
for (int tk = 0; tk < TK; ++tk)
for (int tn = 0; tn < TN; ++tn)
for (int tm = 0; tm < TM; ++tm) {
  dma.start_read(x, w, b);
  for (int k = 0; k < K; ++k) {
    ntx.cfg_ptrs(x, &w[k], &y[k]);
    dma.wait_read();
    ntx.issue_cmd(ntx_api::MAC);
  }
  ntx.wait_ready();
  dma.start_write(y);
  swap_buffers();
}
```

Execution Sample

- All 8 NTX perform the main computation
- DMA writes back results of last computation and reads inputs for next
- Processor core orchestrates operation, computes addresses, pads input data
  - NTXs require no control once started
  - DMA is capable of 2D transfers; core issues multiple small transfers for 3D/4D tensors



Results versus Neurostream

- How much did we gain over the initial inference engine?

- **1.5x** speed up (fast custom accumulator)
- **4x** less control cores (less control overhead)

- Inference:
  - **1.2x** speed up
  - Same energy efficiency
- Training (NS-subset):
  - **1.6x** speed up
  - **1.4x** higher energy efficiency

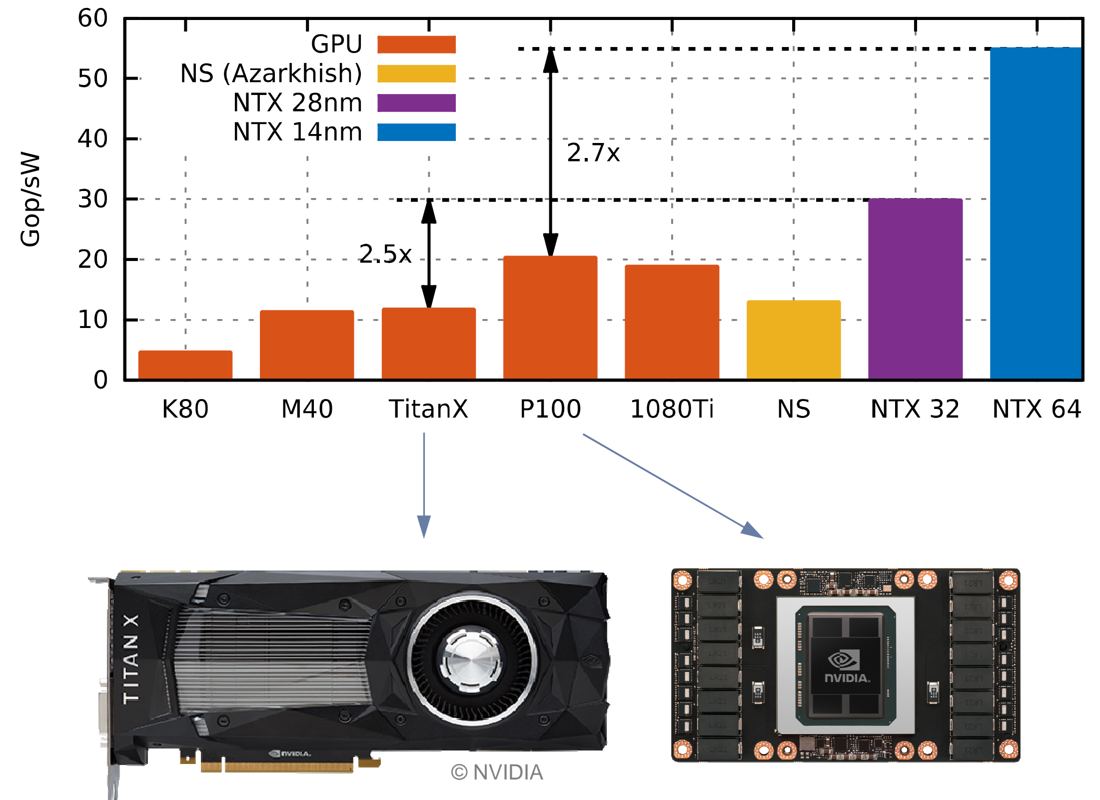
Figure of Merit	**NS** [10]	**NTX** "small"
Number of Clusters	16	16
Cores per Cluster	4	1
Accelerators per Core	2	8
Cluster Frequency [GHz]	1.0	0.75
Accelerator Frequency [GHz]	1.0	1.5
Peak Performance [Gop/s]	256	384
Core Efficiency [Gop/s W]	116	97
Inference		
Total [ms]	14.0	11.3
– Convolution [ms]	13.1	10.5
– Linear [ms]	0.08	0.07
– Pooling [ms]	0.83	0.74
Avg. Bandwidth [GB/s]	14.4	17.8
Peak Bandwidth [GB/s]	51.2	57.6
Efficiency [Gop/s W]	**20.3**	**21.4**
Training		
Total [ms]	56.8	34.8
– Convolution [ms]	54.8 †	33.1
– Linear [ms]	0.65 †	0.43
– Pooling [ms]	1.38 †	1.23
Avg. Bandwidth [GB/s]	11.3	18.5
Peak Bandwidth [GB/s]	51.2	57.6
Efficiency [Gop/s W]	**15.0**	**21.0**

Results versus GPUs

- How much Gflop/s of compute do we get per W of power?
- Comparison of NTX against GPU in similar technology node

- 28 nm: **2.5x** more vs. Nvidia TitanX
- 14 nm: **2.7x** more vs. Nvidia Tesla P100

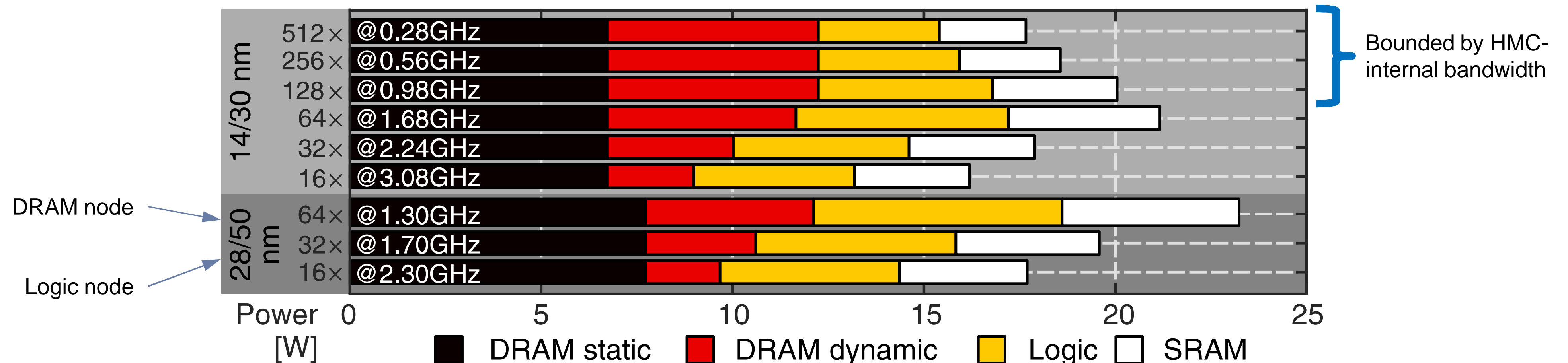
- A note on Nvidia V100:
  - Tensor cores operate on float16
  - Real float32 efficiency likely 30 Gflop/Ws
  - 12 nm NTX likely around 2.1x gain [1]



[1] O. Abdelkader et al, "The Impact of FinFET Technology Scaling on Critical Path Performance under Process Variations," at ICEAC, 2015.

Results Power Budget

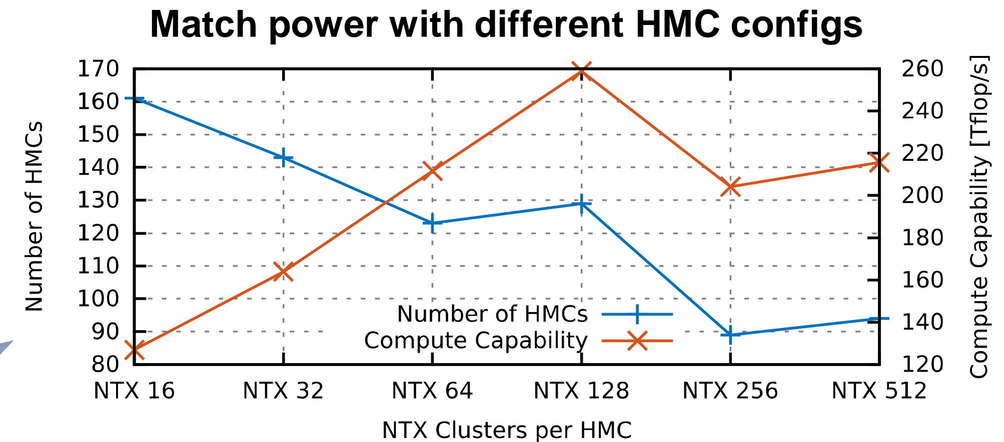
- Assume 25W of power budget in HMC [1]
- Investigate different number of NTX clusters per HMC:
  - 28 nm: 16, 32, 64
  - 14 nm: 16, 32, 64, 128, 256, 512
- Scale voltage and frequency to fit into budget:

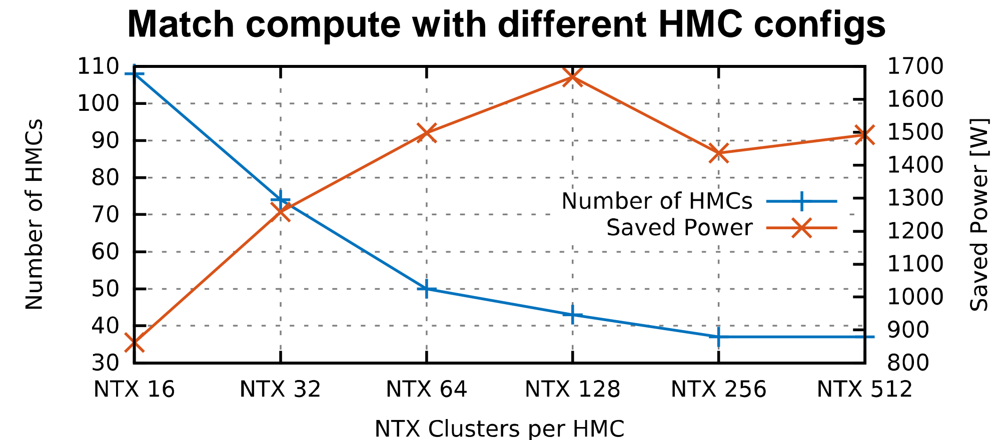


[1] Y. Eckert, N. Jayasena, and G. H. Loh, “Thermal feasibility of die- stacked processing in memory,” in WoNDP, 2014.

Results Data Center

- Match an Nvidia **DGX-1** with HMCs
  - Two Intel Xeon CPUs, eight Tesla P100
  - 3.2 kW total, 2.4 kW due to GPU
  - 84.8 Tflop/s of compute
- Scenario 1: Match **3.2 kW** power envelope
  - **3.1x** increase in compute (258.9 Tflop/s)
  - 129 HMCs, 128 NTX clusters each
- Scenario 2: Match **84.8 Tflop/s** of compute
  - **2.1x** power reduction (1.53 kW)
  - 43 HMCs, 128 NTX clusters each
  - Energy bill: –$1808 per server and year



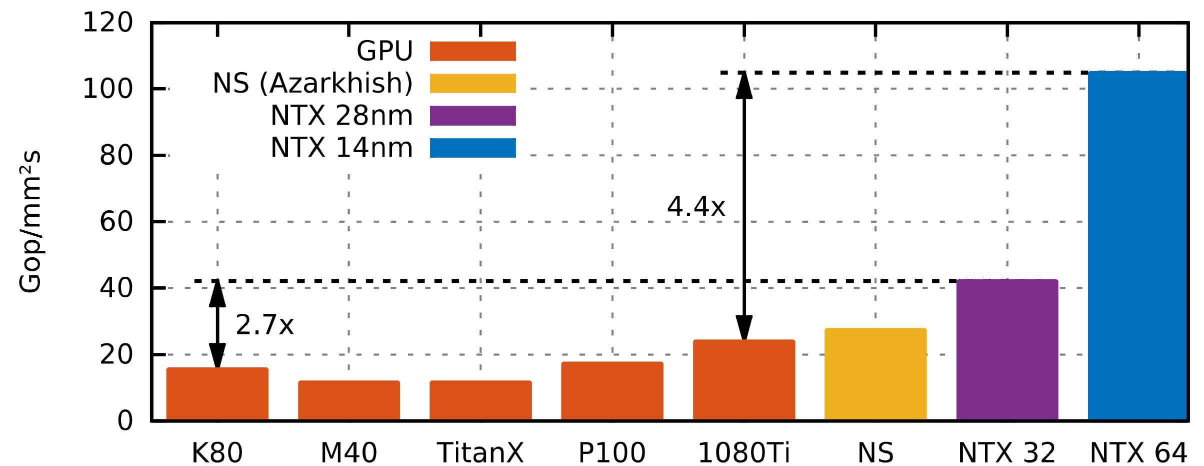


Results Deployed Silicon

- How much Gflop/s of compute do we get per mm^2 of silicon area?
- Comparison of NTX against GPU in similar technology node

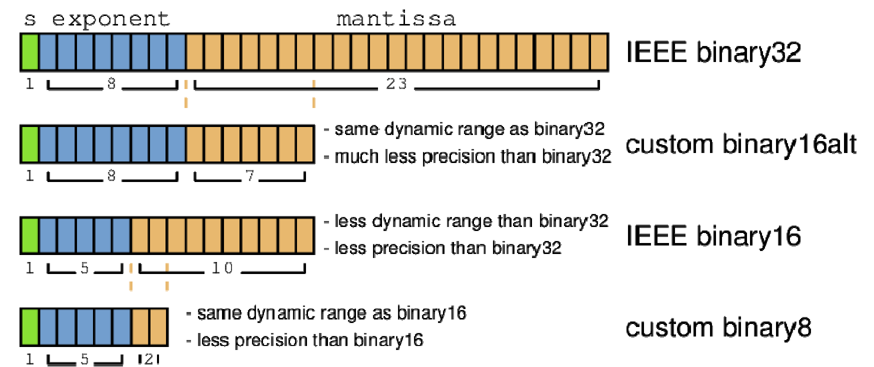
- 28nm: **2.7x** more vs. Nvidia K80
- 14nm: **4.4x** more vs. Nvidia 1080Ti

- GPU dies are huge (>500 mm^2)
- NTX fits easily into HMC (21 mm^2 each)
- Silicon in HMC manufactured anyway, but is unused; virtually zero additional cost

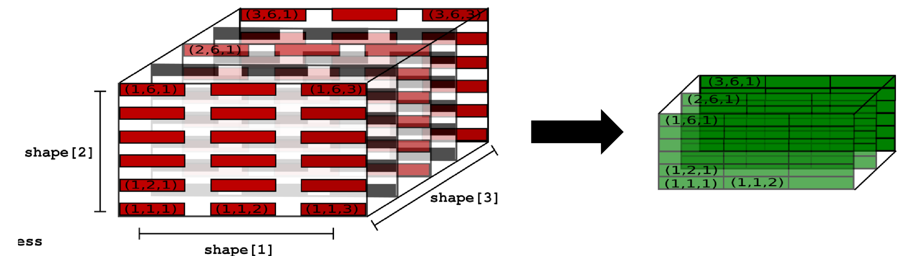


Future Work

- Transprecision Computing
  - Save precious DRAM bandwidth
  - Custom number formats
    - Use float8, float16
    - Logarithmic numbers?
  - On-the-fly data type conversion in DMA



- Tensor DMA
  - Often bottlenecked by 2D DMA
  - Processor needs to issue many small transfers
  - 4D/5D DMA could transfer large tensors independently



- Automated Mapping of Kernels
  - Starting from Compute Graph, e.g. TensorFlow

Future Work

- NTX beyond Machine Learning
  - Sparse matrices/vectors
  - General stencil operations
  - Operations under Winograd/FFT transform (sparsity!)
- HBM (High Bandwidth Memory)
  - Consumes less power than HMC
  - Gaining traction with GPUs
  - Multiple vendors; HMC only produced by one company
  - Direct access to DRAM via usual DDR interface
- Other Memory Substrates
  - Non-volatile memory?