ETH zürich

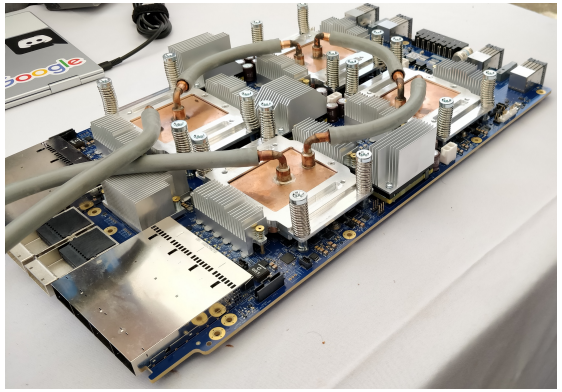
NTX: A 260 Gflop/sW Streaming Accelerator for Oblivious Floating-Point Algorithms in 22nm FD-SOI

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

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

Introduction: The Specialization Challenge

- Recent surge of Machine Learning cloud workloads:
  - **3x** growth of ML compute workload in 12 months at Facebook [1]
  - Huawei DaVinci Max **cloud training platform** [2]
  - Compute spent on training **doubles every 3.5 months** (Intel Nervana NNP-T) [3]
  - Wafer-Scale Deep Learning at Cerebras [4]
  - Gaudi AI training processor by Habana Labs [5]



- Key challenge: Training algorithms change
  - Rise of sparsity in DNNs
  - Rise of novel number formats (e.g. bfloat)
- Avoid **overspecialization**!
- GPUs are successful because the remain flexible (general purpose):
  - Reduction of **von Neumann bottleneck** thanks to SIMT
  - **Memory latency tolerance** thanks to heavy multithreading

[1] “Zion: Facebook Next-Generation Large Memory Training Platform”, Facebook, HotChips 2019
[2] “DaVinci: A Scalable Architecture for Neural Network Computing”, Huawei, HotChips 2019
[3] “Deep Learning Training at Scale”, Intel, HotChips 2019
[4] “Wafer-Scale Deep Learning”, Cerebras Systems, HotChips 2019
[5] Habana Labs, HotChips 2019

Introduction: Target Workload

- Key points for designing new systems:

- **Maintain flexibility** (fast moving algorithms)
- **Extreme energy efficiency**

- Our approach: design an architecture for a large class of problems!
- **Data-Oblivious Programs**
  - Control flow does not depend on data
  - Large number of algorithms fall into this category
  - Prominently includes DNN training

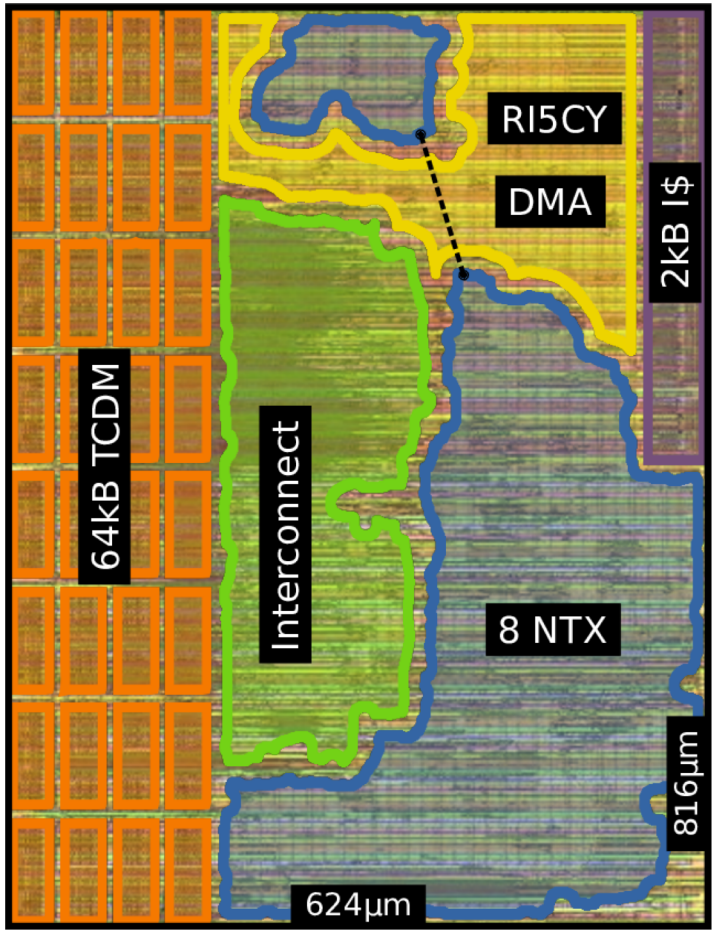
- Enter the Network Training Accelerator

Data-Oblivious Program Examples:

- ✓ Reductions and Scans
- ✓ Stencils
- ✓ Linear Algebra
  - ✓ Matrix Multiplication
  - ✓ Tridiagonal Solve
  - ✓ Cholesky Factorization
  - ✓ LU decomposition (almost oblivious)
- ✓ Deep Learning (Convolution, ReLU)
- ✓ FFT
- ✓ Graph Algorithms
  - ✓ Breadth-first Search
  - ✓ Single-source Shortest Path
  - ✓ Connected Components
- ✓ Sorting Networks
  - ✓ Bitonic Sort

Introduction: NTX at a Glance

- "Network Training Accelerator"
  - 32 bit float streaming co-processor (IEEE 754 compatible)
  - Custom 300 bit "wide-inside" Fused Multiply-Accumulate
  - 1.7x lower RMSE than conventional FPU
- Manufactured in Globalfoundries 22FDX
  - 1 RISC-V core ("RI5CY") and DMA
  - 8 NTX co-processors
  - 64 kB L1 scratchpad memory (comparable to 48 kB in V100)
- 0.5 mm^2, 1.25 GHz worst-case, 166 mW, 0.8 V

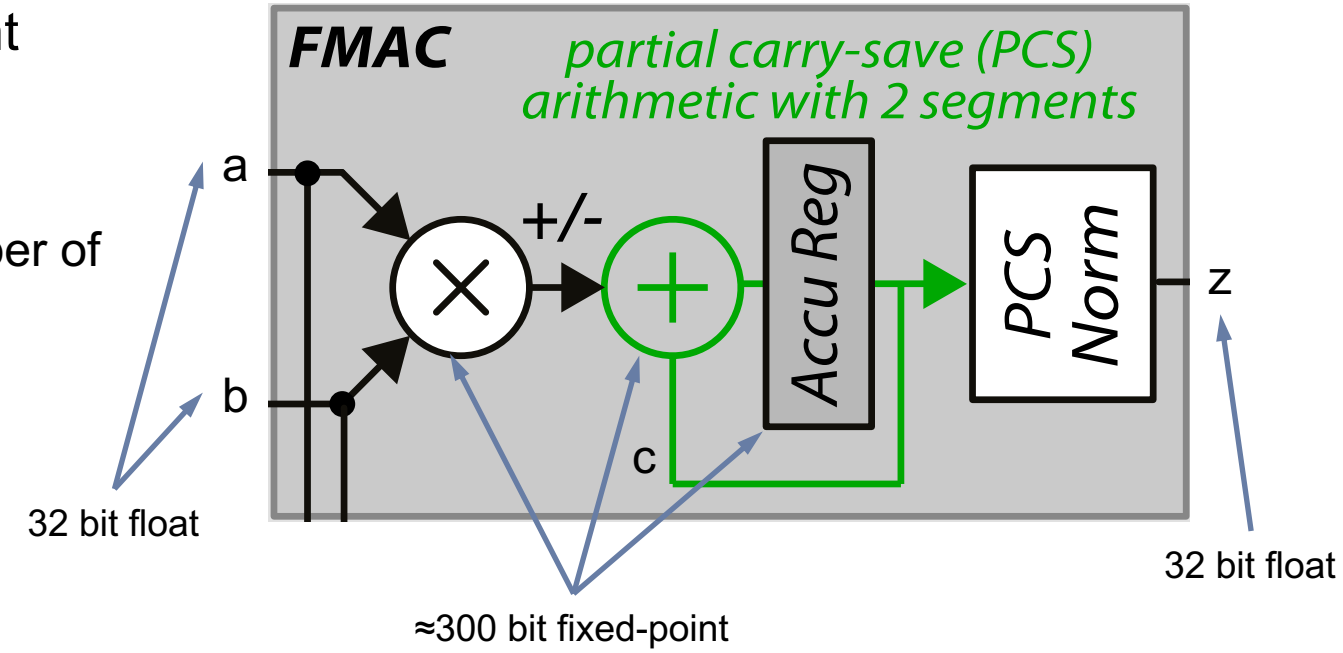


Key ideas to increase hardware efficiency:

- Reduction of von Neumann bottleneck (**load/store elision** through streaming)
- Latency hiding through DMA-based **double-buffering**

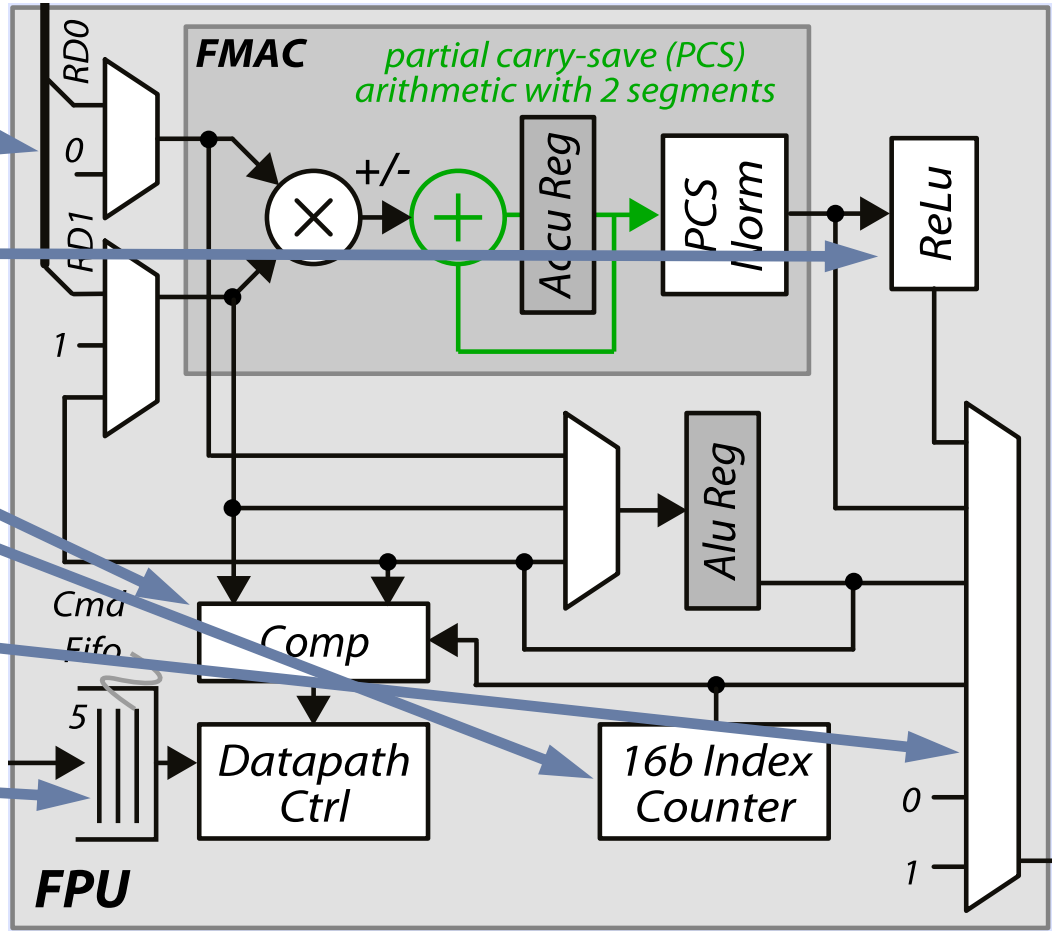
Architecture: FMAC

- Main data path is a **single-cycle partial carry save FMA**
- Expansion of float operands to fixed-point
- Multiplication and addition in fixed-point
  - **Single-cycle**
  - Tuneable performance by increasing number of partial sums
- Conversion to float after accumulation
  - Partial sums are accumulated
  - Conversion from fixed-point to float
- Heavily pipelined



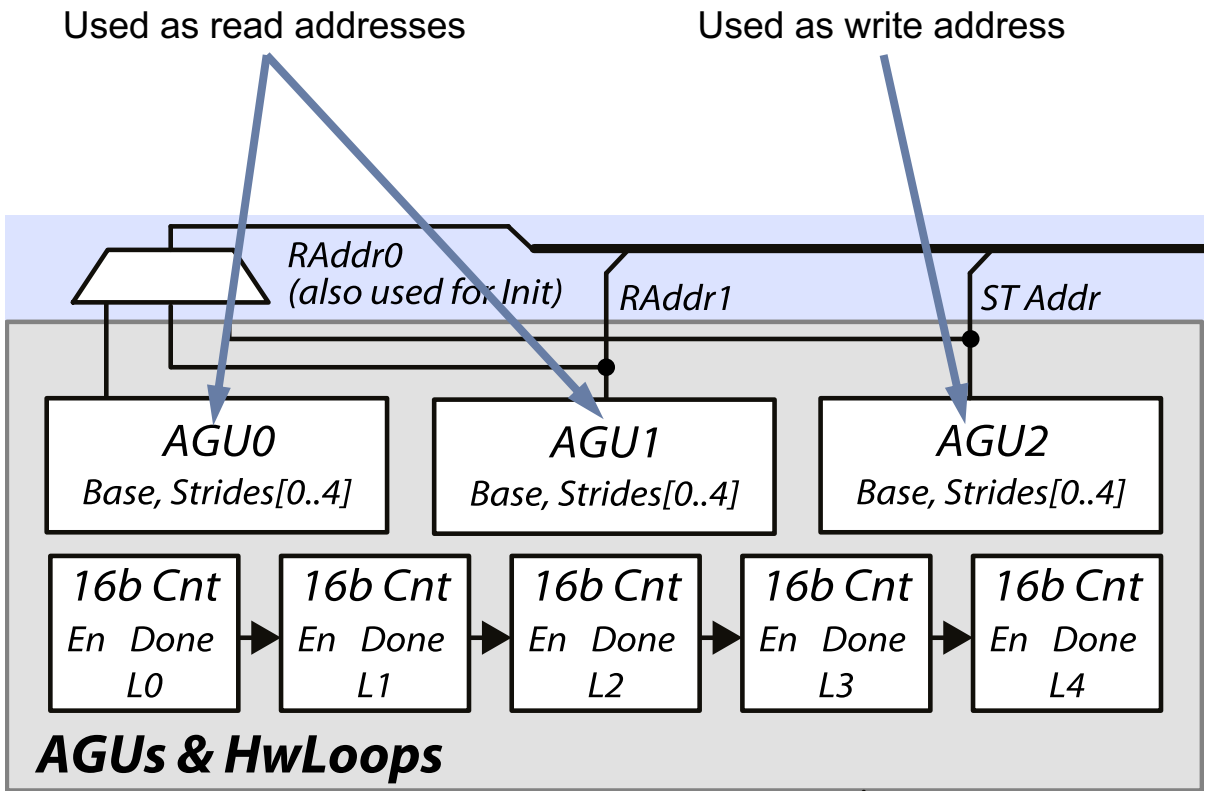
Architecture: Data Path

- FMA operands arrive as **memory streams**
  - Maskable to 0/1 to disable add/mul
- Optional **ReLU** on FMA result
- Comparator for finding **max/min**
- Index counter for finding **argmax/argmin**
  - Enables maxpool derivatives
- Output can be **masked** to 0/1
  - Enables ReLU derivatives
- **Fire-and-forget datapath**
  - Command pushed into FIFO
  - Consumes fixed number of input items
  - Produces fixed number of output items



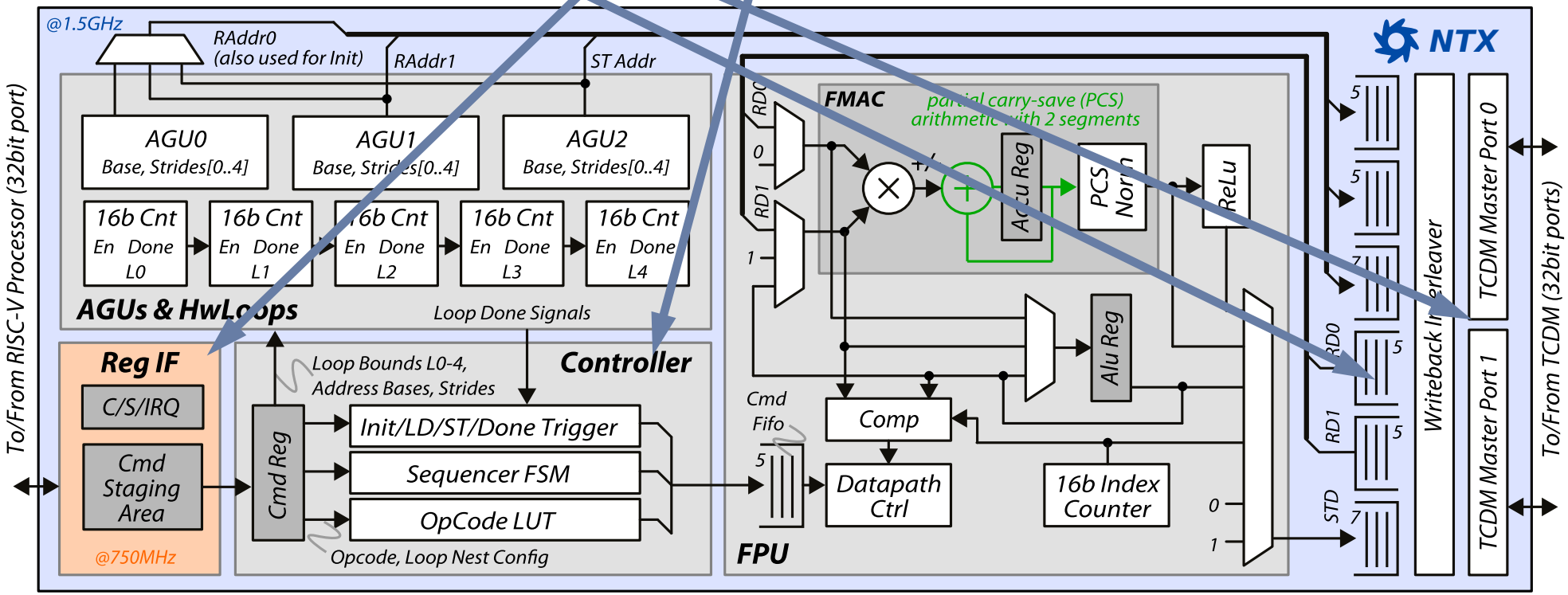
Architecture: Address Generation

- 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



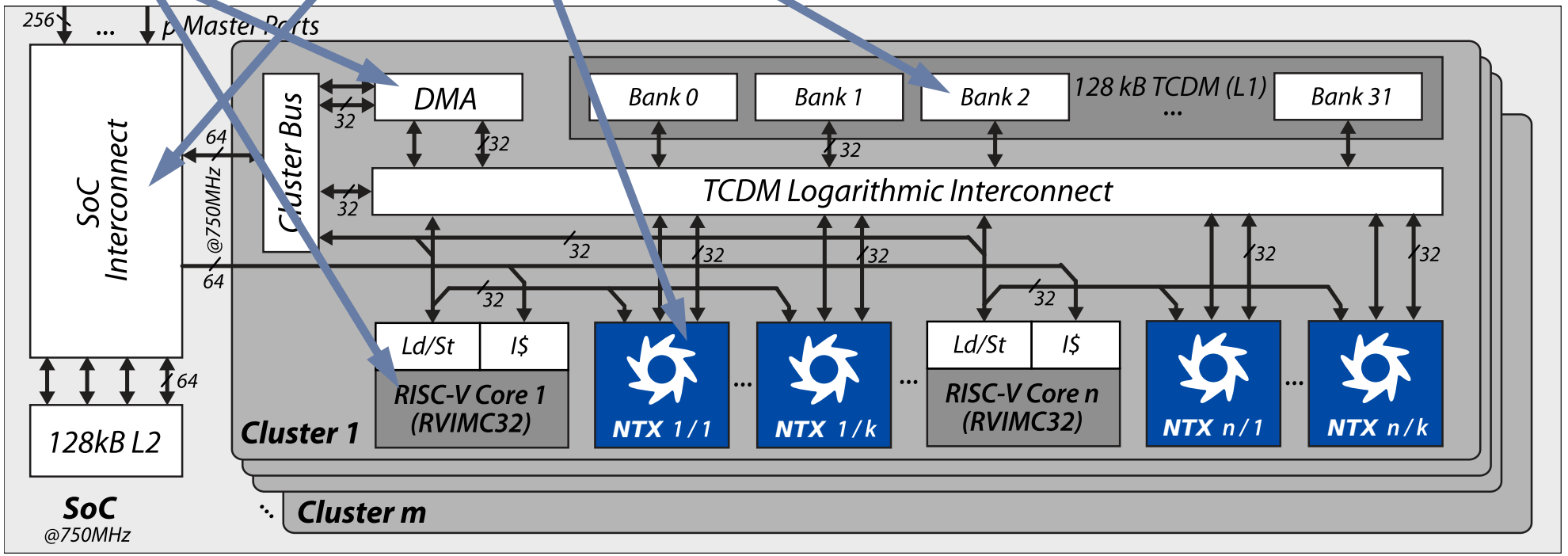
Architecture: Coprocessor

- 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)



Programming: Von Neumann Bottleneck

- NTX helps alleviate the von Neumann bottleneck
  - No explicit load/store instructions
  - No explicit address calculation instructions
- Simple example: Dot product over 1024 elements
- With single RV32IF:
  - **6146** instructions executed
- With single NTX (plus RV32I):
  - **10** instructions executed
  - **1024 idle cycles** while NTX executes (can be used)
- NTX reduces instruction bandwidth by **>500x**
  - Even more when using more nested loops
- NTX amortizes single instruction stream over 8 FPUs
  - Data/Inst. bandwidth ratio of **16** (worst case, usually higher)

Single RV32IF Core:

```
Setup      li      t0, 1024
Hot Loop   flw     ft0, 0(a0)
           flw     ft1, 0(a1)
           fmadd   ft2, ft0, ft1, ft2
           addi    t0, t0, -1
           addi    a0, a0, 4
           addi    a1, a1, 4
           bgtz    t0, -6
Writeback  fsw     ft2, 0(a2)
```

Single NTX:

```
Setup   sw  a0, NTX_AGU0_PTR
        sw  a1, NTX_AGU1_PTR
        sw  a2, NTX_AGU2_PTR
        li  t1, 1024
        sw  t1, NTX_BOUND_L0
        li  t1, 4
        sw  t1, NTX_AGU0_S0
        sw  t1, NTX_AGU1_S0
        li  t1, NTX_MAC_CMD
        sw  t1, NTX_CMD
Idle    wfi
```

Performance: Memory Accesses

- Compared to NVIDIA Volta GPU [1]:
  - Register file in GPU holds registers and thread-local data
  - Each register read/write is an SRAM access
  - Register and data accesses compete for SRAM

Volta Assembly	**NTX Pseudocode**

`LDS R2, [R0]`
`LDS R3, [R1]`
`FFMA R4, R2, R3, R2`
	`FMAC accu, [AGU0], [AGU1]`

2 mem. acc. (“[…]”) 8 reg. acc.	2 mem. acc. (“[…]”) 0 reg. acc. (+ addr. calc for free)
= 10 SRAM hits total	**= 2 SRAM hits total**

- Our latency hiding technique is not more expensive than in GPUs

1 Volta SM	8 NTX cl.
64 FPUs	64 FPUs
256 kB RF 128 kB L0 Cache	512 kB TCDM
32-2048 threads	8 threads

[1] Volta Architecture Whitepaper, NVIDIA

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();
}
```

Configure loop bounds once for the entire kernel

Start reading input data

Point NTX at the address of the input data

Wait for the input data to be loaded (overlaps with previous NTX computation)

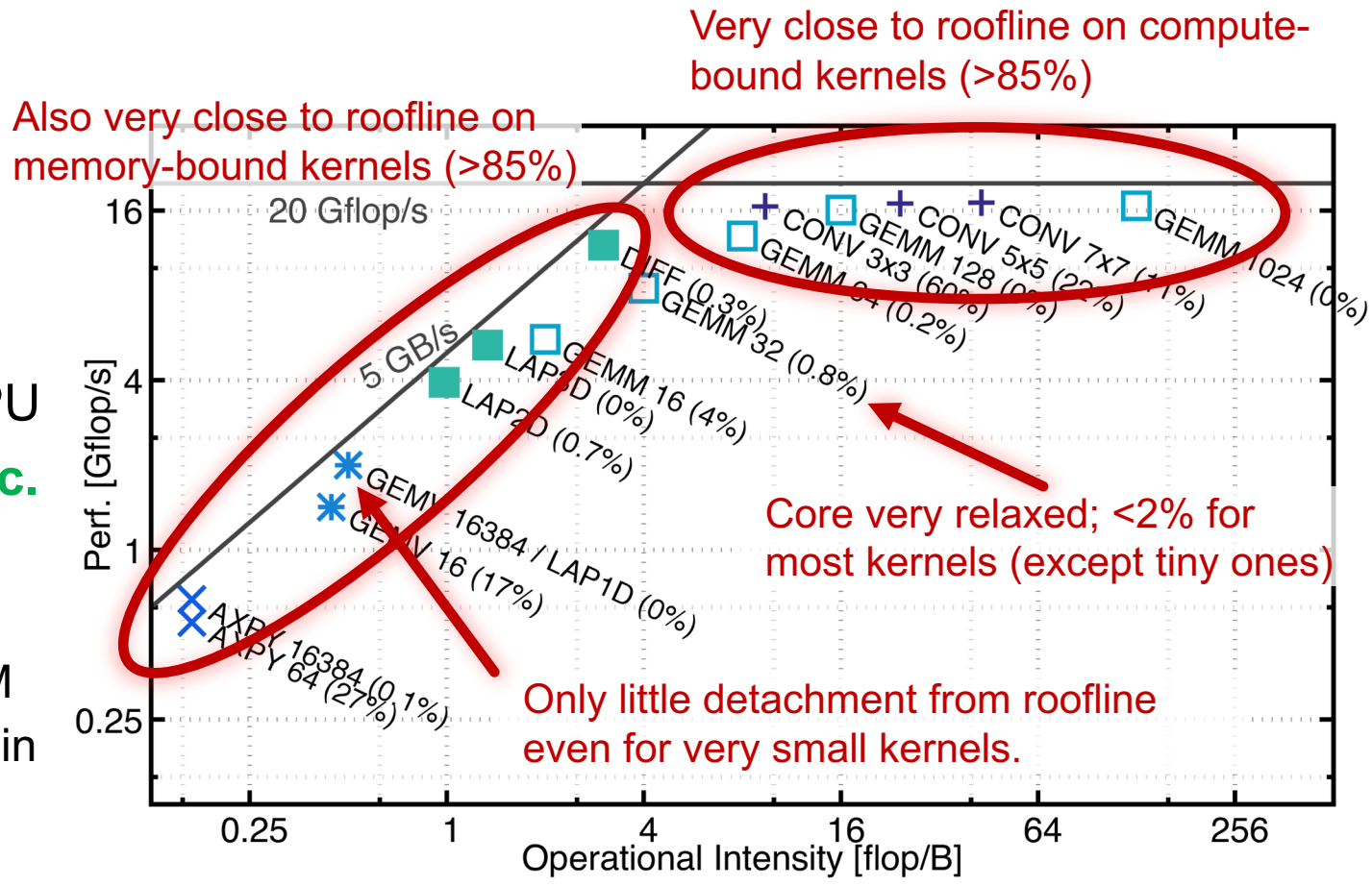
Start next computation

Wait for computation to complete

Start writing back output data

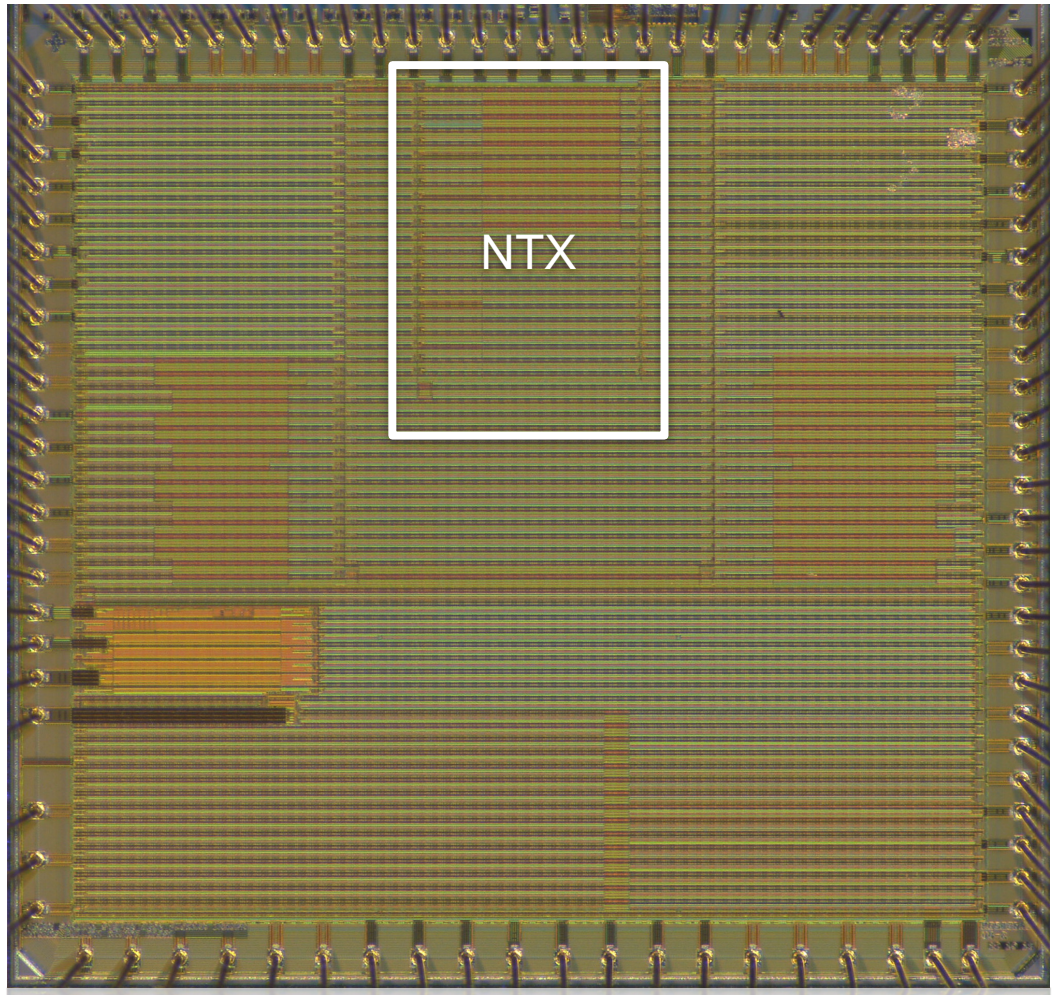
Results: Roofline

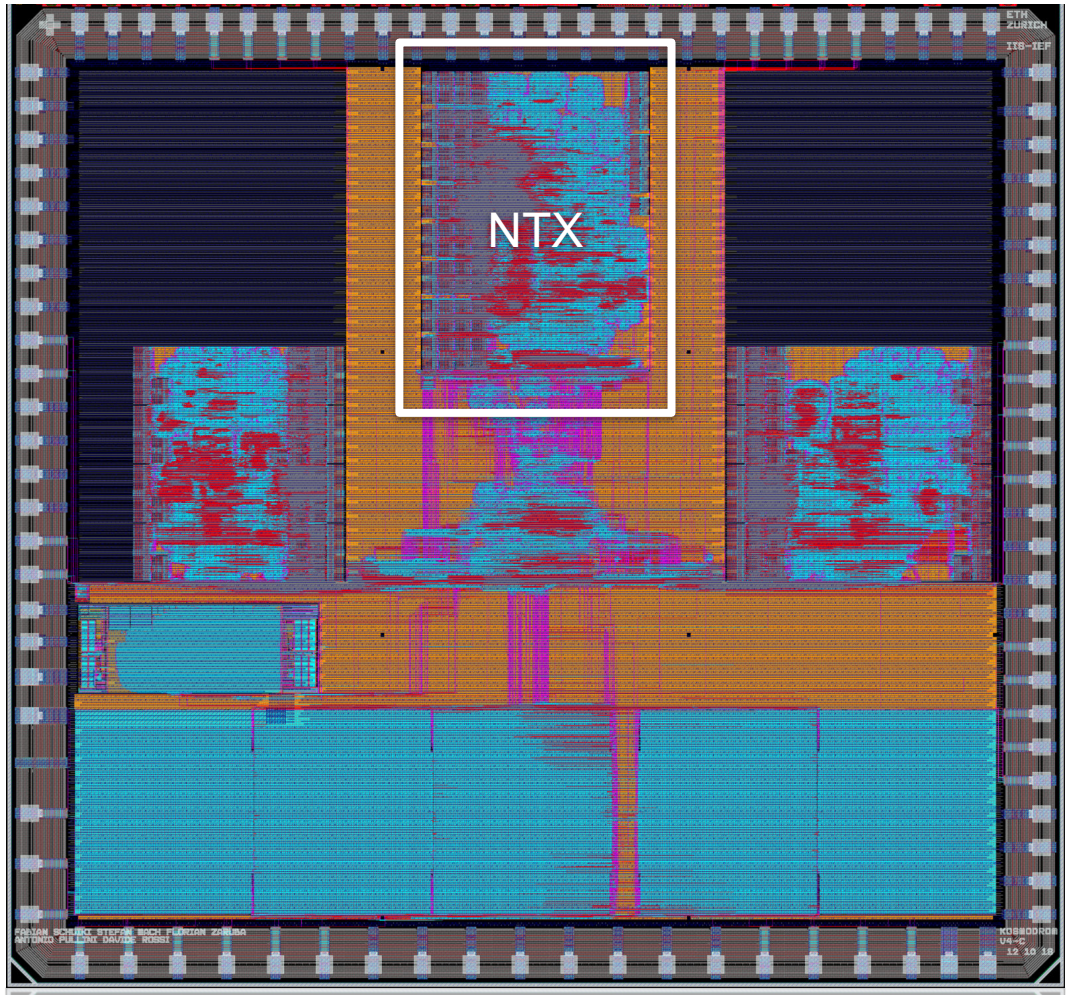
- Alleviates **von Neumann bottleneck**:
  - No explicit loads, stores, address calculation
- DMA transfers data in background
- NTX independent over **>100000 cycles**
- Strong **>85% utilization** of DMA and FPU
- Effectively issues **32 flops**, **20 mem. acc.** per cycle (16 local & 4 global, 32 bit)
- Covers wide range of oblivious kernels:
  - **Linear Algebra:** e.g AXPY, GEMV, GEMM
  - **Stencils:** e.g. Diffusion, Discrete Laplace in 1D/2D/3D
  - **Machine Learning:** e.g. Convolution, ReLU, FC



Results: Manufactured Chip in 22FDX

see asic.ethz.ch

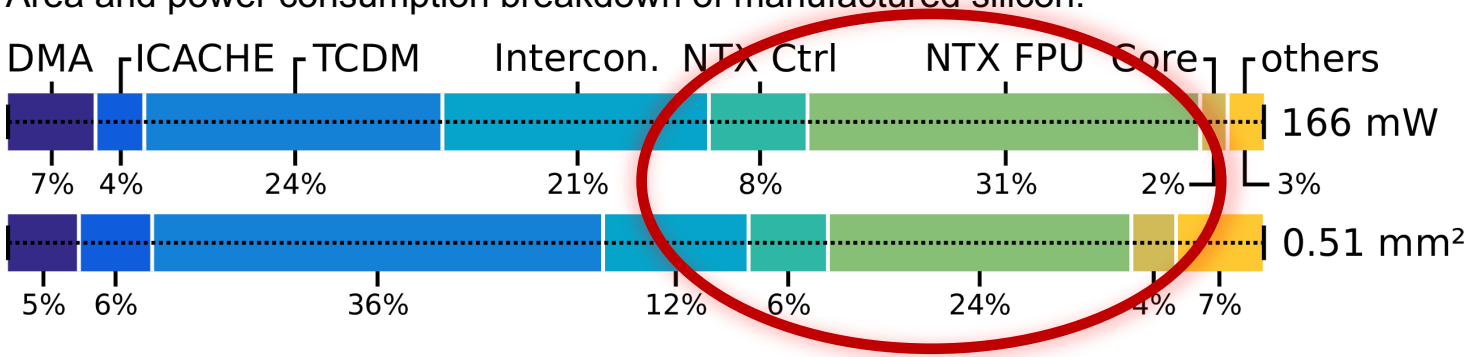




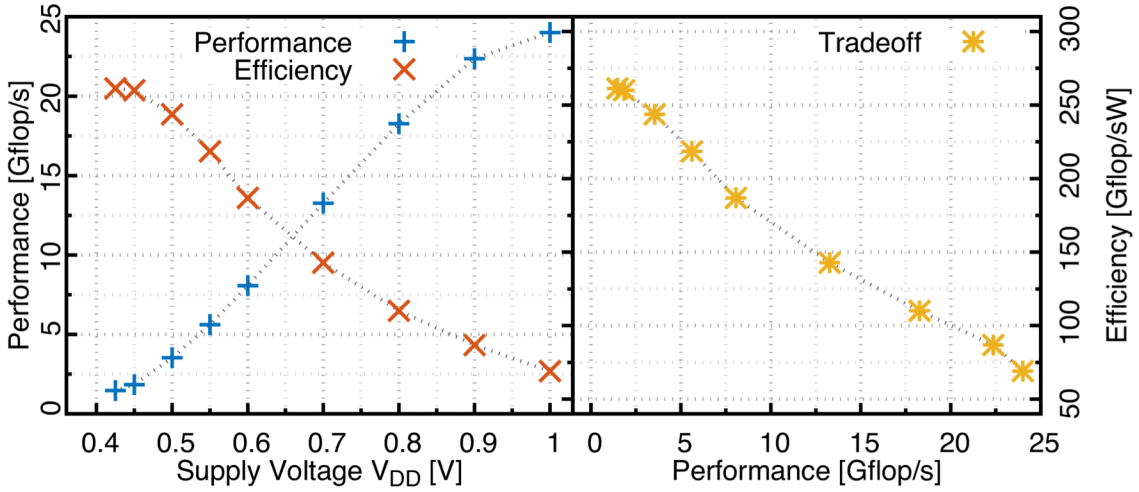
Results: Silicon Measurements

- Key benefit: **>30% of area is FPU**
- Yields high area efficiency:
  - **47 Gflop/s·mm²**
- High efficiency mode at 0.425 V:
  - **260 Gflop/sW**, 1.5 Gflop/s
- High performance mode at 1.0V:
  - **24 Gflop/s**, 70 Gflop/sW
- Wide range of operating voltage:
  - Logic: **0.425 V to 1.0 V** and above
  - SRAMs: **0.55 V to 1.0V** and above
- Dynamically set operating point for workload
  - Almost linear trade-off between perf./efficiency
  - Compensate for PVT variation via body-bias
  - Performance boost up to 1.6x

Area and power consumption breakdown of manufactured silicon:

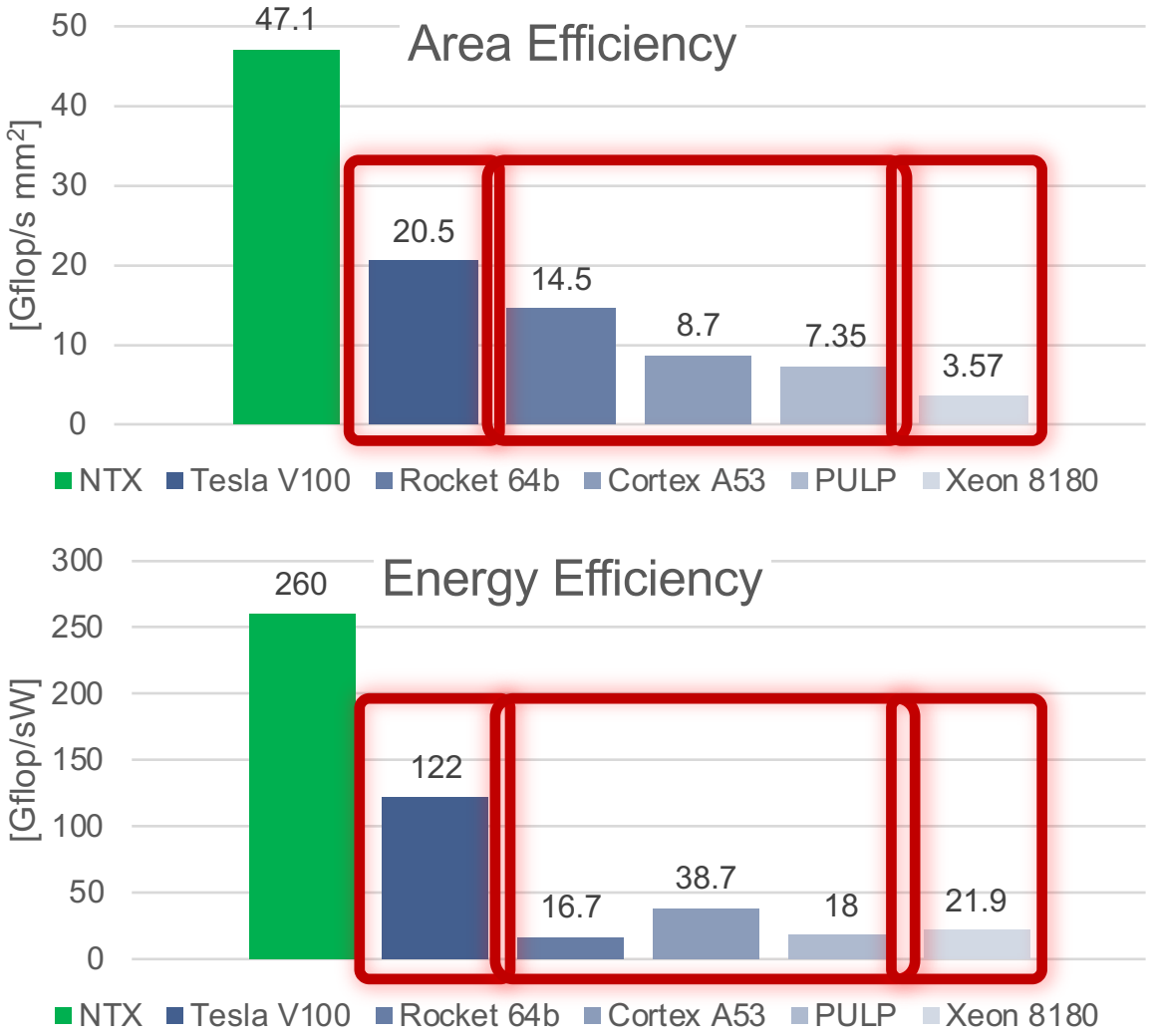


Performance and efficiency measured on manufactured silicon:



Results: Comparison with Other Work

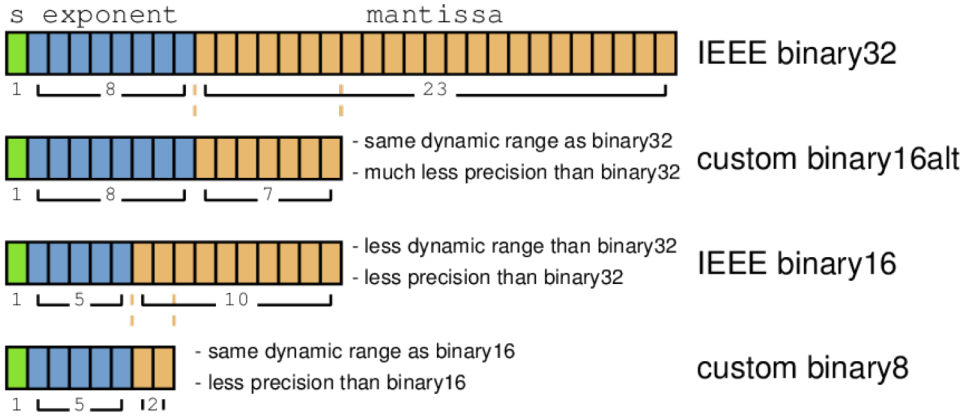
- NTX is highly competitive with SoA
- Compared to equivalent SM in Volta V100 (our estimates):
  - **2.1x** energy efficiency gain
  - **2.3x** area efficiency gain (node-compensated)
- Compared to PULP cluster, ARM Cortex A53, and a 64 bit Rocket core:
  - **6.7x to 15.6x** energy efficiency gain
  - **3.2x to 6.4x** area efficiency gain
- Compared to a 28-core dual-AVX-512 Intel Xeon 8180 CPU:
  - **11.9x** energy efficiency gain
  - **13.2x** area efficiency gain



Future Work

- Address Generator Extension
  - NTX's address generator applicable to more kernels
  - FFTs, linear algebra decompositions/factorizations
  - Searches? Sorting? Graphs?
- Bring streaming to RISC-V cores
- Transprecision Computing
  - Reduced-precision training is around the corner [1]
  - Save precious DRAM bandwidth
  - Custom number formats
    - Use float8, float16
    - Logarithmic numbers?
  - On-the-fly data type conversion in DMA
- Automated Mapping of Kernels
  - Starting from Compute Graph, e.g. TensorFlow

PRECOMP
Open Transprecision Computing



[1] Coleman, Cody, et al. "Analysis of dawnbench, a time-to-accuracy machine learning performance benchmark." ACM SIGOPS Operating Systems Review 53.1 (2019): 14-25.

Thanks!

—

Questions?