Celerity: An Open Source RISC-V Tiered Accelerator Fabric

Tutu Ajayi‡, Khalid Al-Hawaj†, Aporva Amarnath‡, Steve Dai†, Scott Davidson*, Paul Gao*, Gai Liu†, Atieh Lotfi*, Julian Puscar*, Anuj Rao*, Austin Rovinski‡, Loai Salem*, Ningxiao Sun*, Christopher Torng†, Luis Vega*, Bandhav Veluri*, Xiaoyang Wang*, Shaolin Xie*, Chun Zhao*, Ritchie Zhao†,

Christopher Batten†, Ronald G. Dreslinski‡, Ian Galton*, Rajesh K. Gupta*, Patrick P. Mercier*, Mani Srivastava§, Michael B. Taylor*, Zhiru Zhang†

* University of California, San Diego

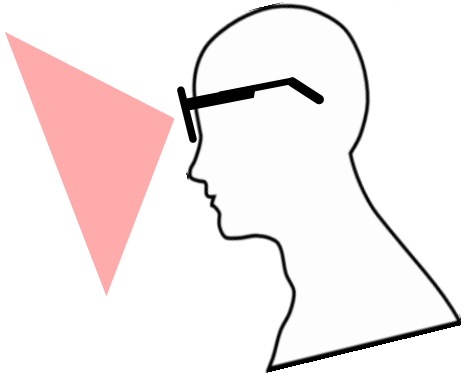
† Cornell University

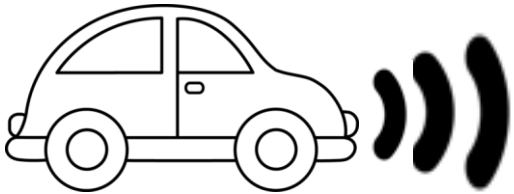
‡ University of Michigan

§ University of California, Los Angeles

High-Performance Embedded Computing

- Embedded workloads are *abundant* and *evolving*
  - Video decoding on mobile devices
    - Increasing bitrates, new emerging codecs
  - Machine learning (speech recognition, text prediction, …)
    - Algorithm changes for better accuracy and energy performance
  - Wearable and mobile augmented reality
    - Still new, rapidly changing models and algorithms
  - Real-time computer vision for autonomous vehicles
    - Faster decision making, better image recognition
- We are in the post-Dennard scaling era
  - Cost of energy > Cost of area
- How do we attain extreme energy-efficiency while also maintaining flexibility for evolving workloads?





http://clipartfan.com/wp-content/uploads/2017/03/car-clipart-black-and-white-car-black-and-white-images.png
http://www.clker.com/cliparts/9/t/V/w/x/j/head-outline-md.png

Celerity: Chip Overview

- TSMC 16nm FFC
- 25mm^2 die area (5mm x 5mm)
- ~385 million transistors
- 511 RISC-V cores
  - 5 Linux-capable “Rocket Cores”
  - 496-core mesh tiled array “Manycore”
  - 10-core mesh tiled array “Manycore” (low voltage)
- 1 Binarized Neural Network Specialized Accelerator
- On-chip synthesizable PLLs and DC/DC LDO
  - Developed in-house
- 3 Clock domains
  - 400 MHz – DDR I/O
  - 625 MHz – Rocket core + Specialized accelerator
  - 1.05 GHz – Manycore array
- 672-pin flip chip BGA package
- 9-months from PDK access to tape-out

UCSD

CORNELL UNIVERSITY FOUNDED A.D. 1865

UNIVERSITY OF MICHIGAN

Ucla

Celerity Overview

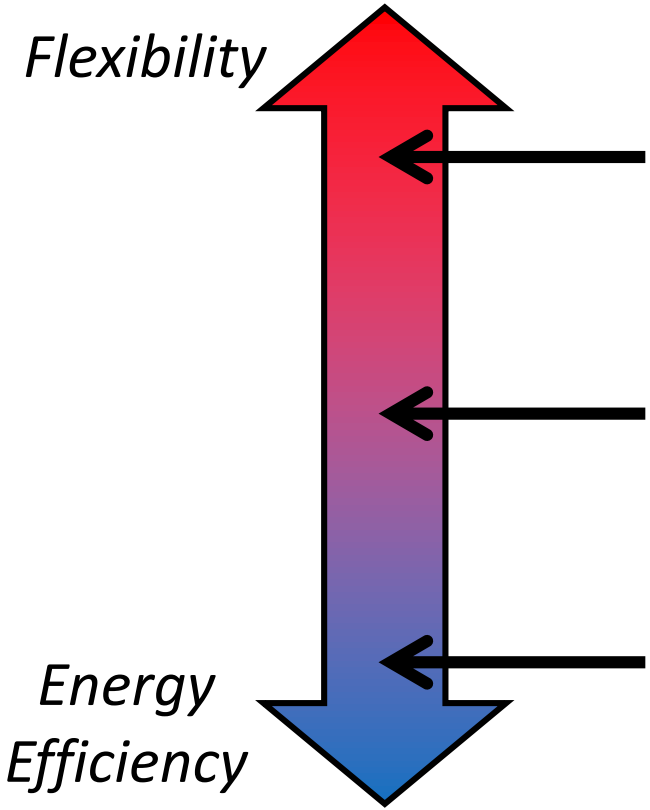
Tiered Accelerator Fabric

Case Study: Mapping Flexible Image Recognition to a Tiered Accelerator Fabric

Meeting Aggressive Time Schedule

Conclusion

Decomposition of Embedded Workloads



- General-purpose computation
- Operating systems, I/O, etc.

- Flexible and energy-efficient
- Exploits coarse- and fine-grain parallelism

- Fixed-function
- Extremely strict energy efficiency requirements

Tiered Accelerator Fabric

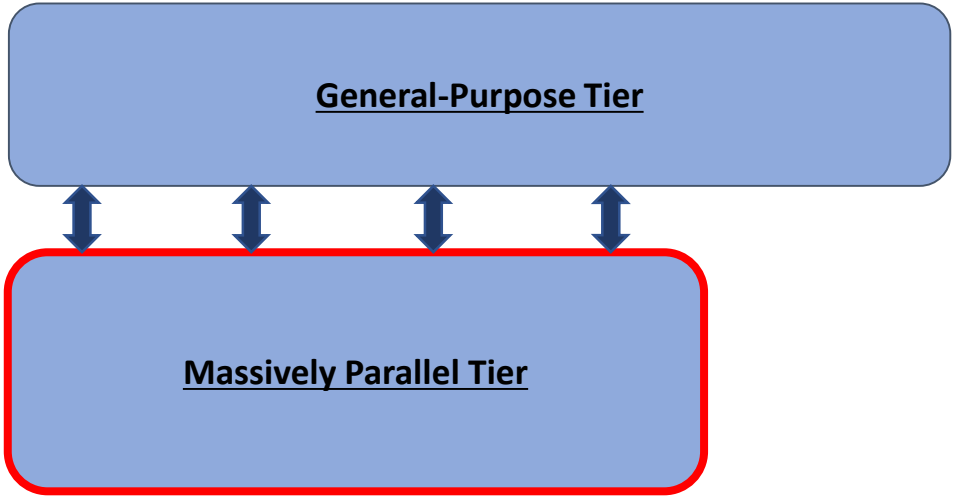
An architectural template that maps embedded workloads onto distinct tiers to ***maximize energy efficiency*** while ***maintaining flexibility***.

Tiered Accelerator Fabric

General-purpose computation, control flow and memory management

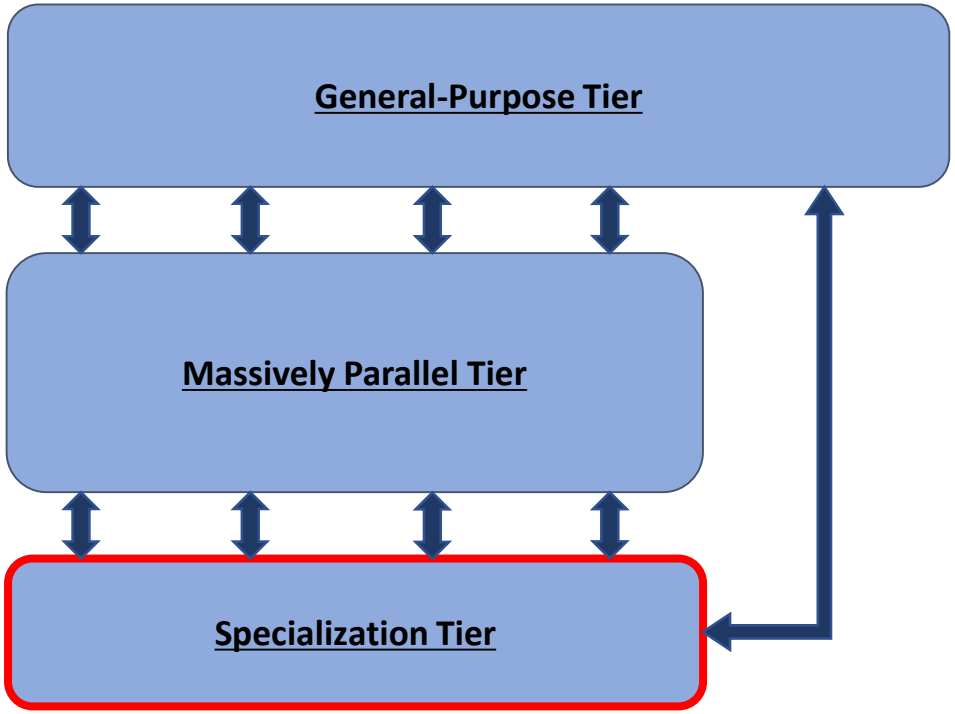
Tiered Accelerator Fabric

Flexible exploitation of coarse and fine grain parallelism

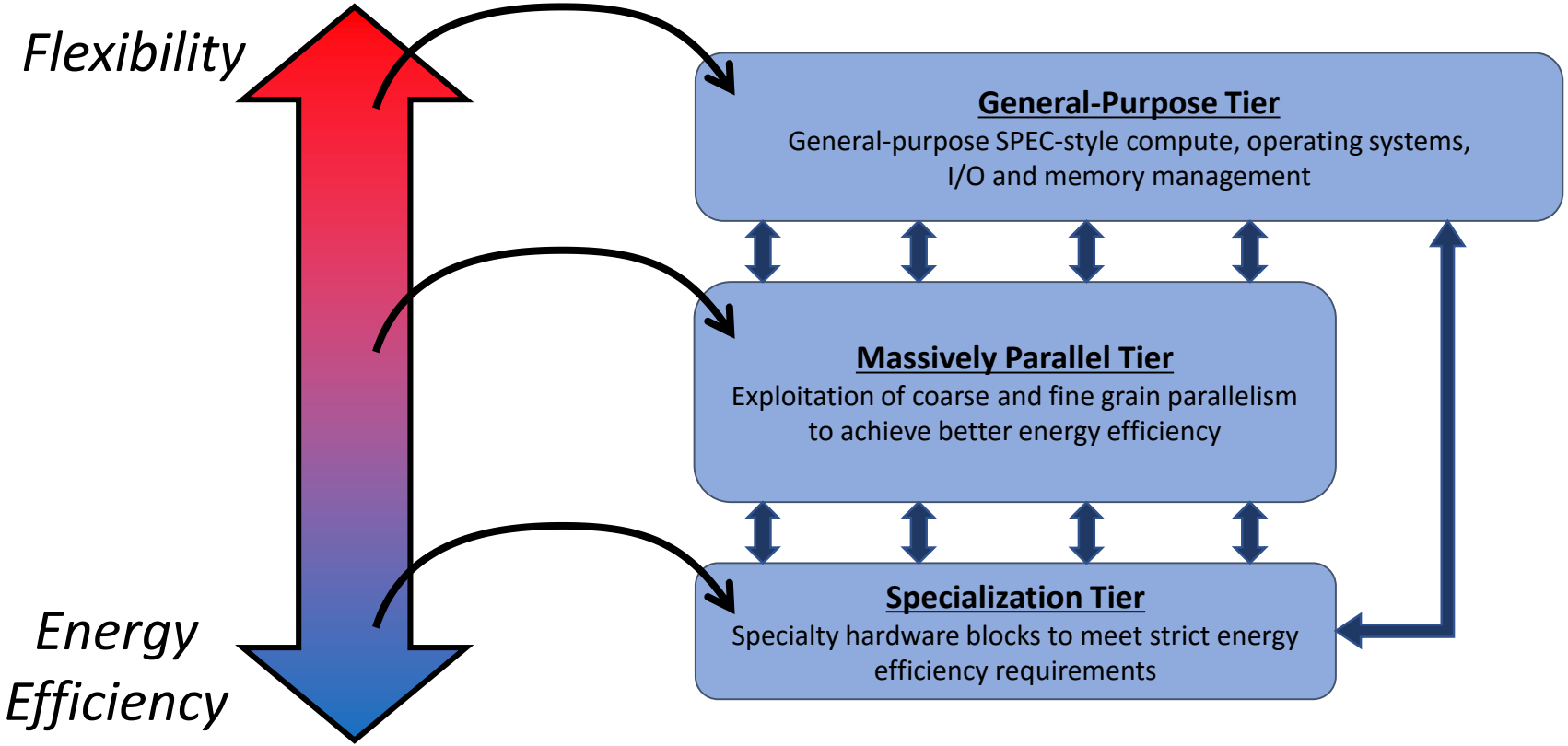


Tiered Accelerator Fabric

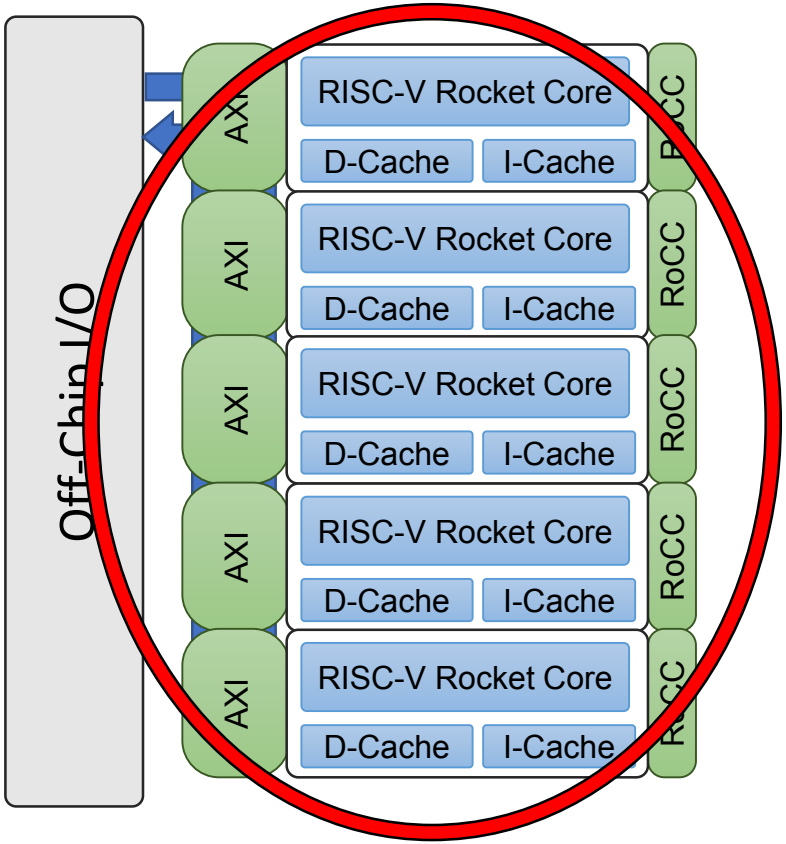
Fixed-function specialized accelerators for energy efficiency requirements



Mapping Workloads onto Tiers

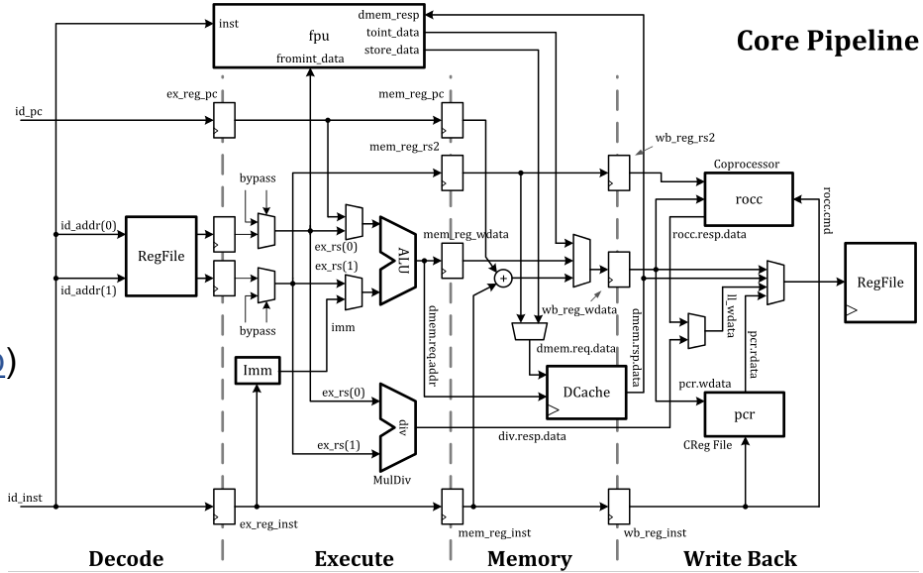


Celerity: General-Purpose Tier

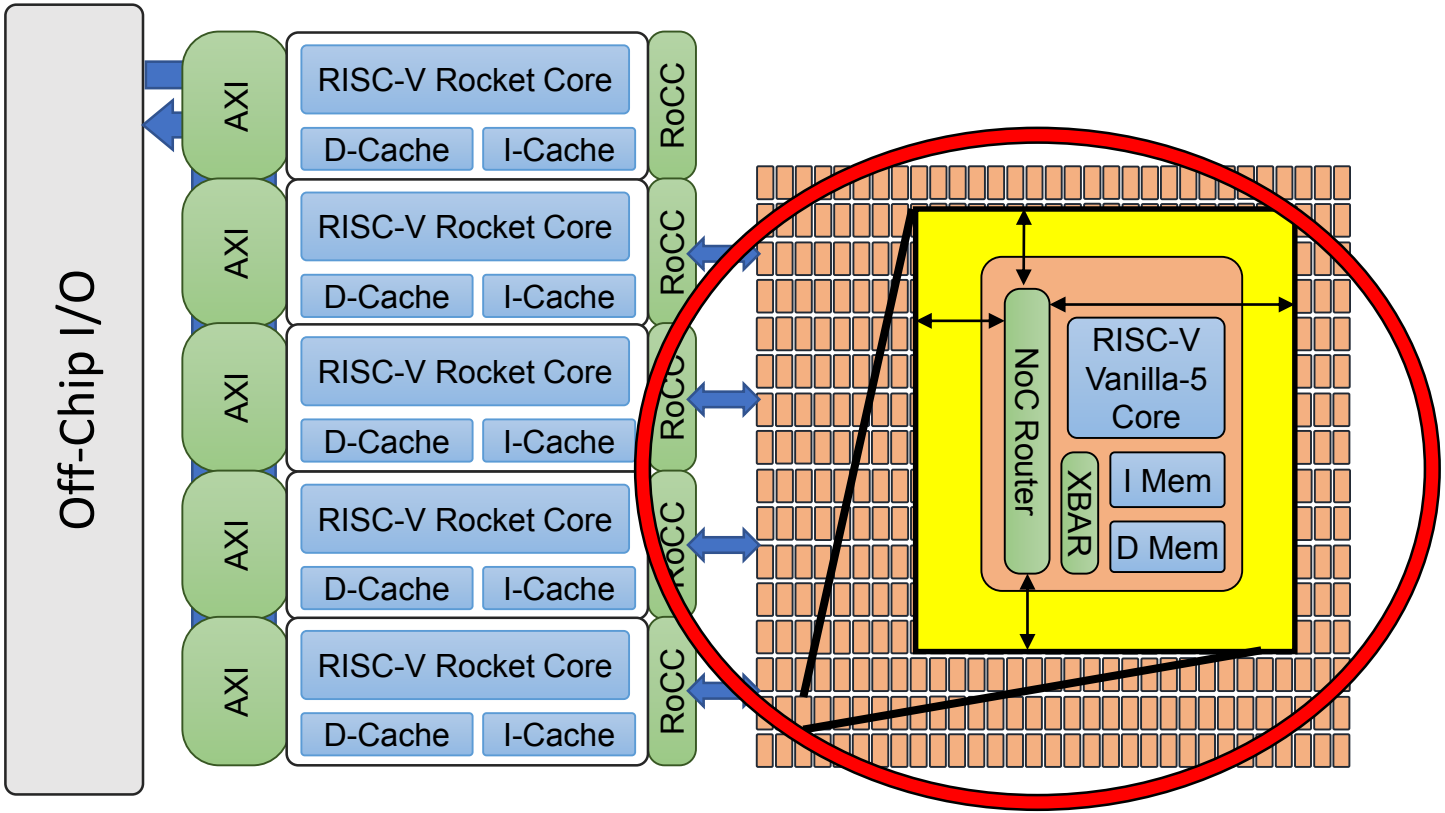


General-Purpose Tier: RISC-V Rocket Cores

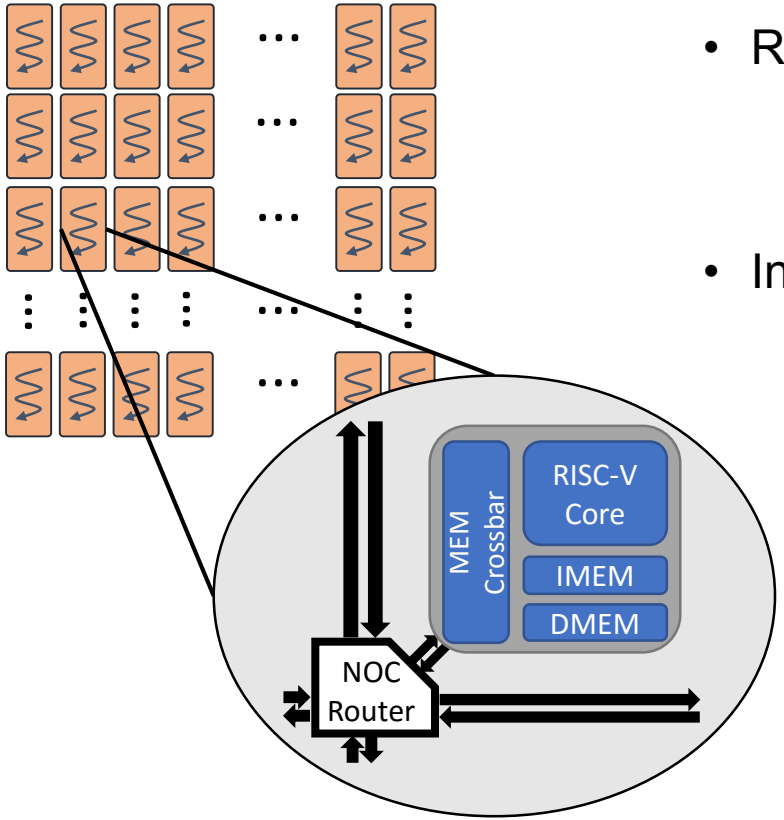
- Role of the General-Purpose Tier
  - General-purpose SPEC-style compute
  - Exception handling
  - Operating system (e.g. TCP/IP Stack)
  - Cached memory hierarchy for all tiers
- In *Celerity*
  - 5 Rocket Cores, generated from Chisel (https://github.com/freechipsproject/rocket-chip)
    - 5-stage, in-order, scalar processor
    - Double-precision floating point
    - I-Cache: 16KB 4-way assoc.
    - D-Cache: 16KB 4-way assoc.
    - RV64G ISA
  - 0.97 mm^2 per Rocket core @ 625 MHz



Celerity: Massively Parallel Tier



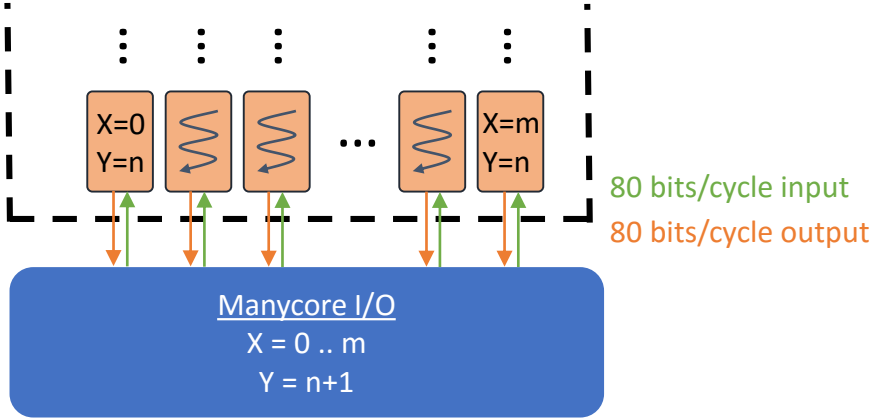
Massively Parallel Tier: Manycore Array

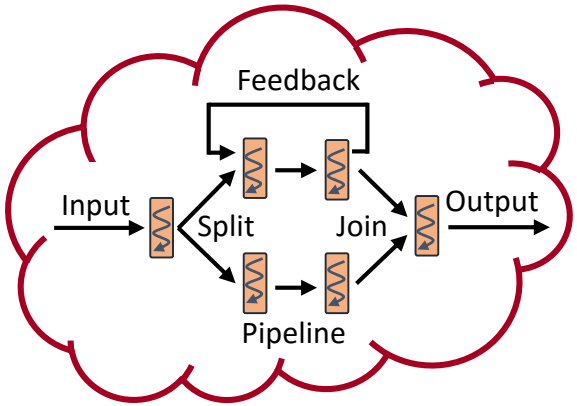


- Role of the Massively Parallel Tier
  - Flexibility and improved energy efficiency over the general-purpose tier by massively exploiting parallelism
- In *Celerity*
  - 496 low power RISC-V Vanilla-5 cores
    - 5-stage, in-order, scalar cores
      - Fully distributed memory model
      - 4KB instruction memory per tile
      - 4KB data memory per tile
    - RV32IM ISA
    - 16x31 tiled mesh array
    - Open source!
  - 80 Gbps full duplex links between each adjacent tile
  - 0.024mm^2 per tile @ 1.05 GHz

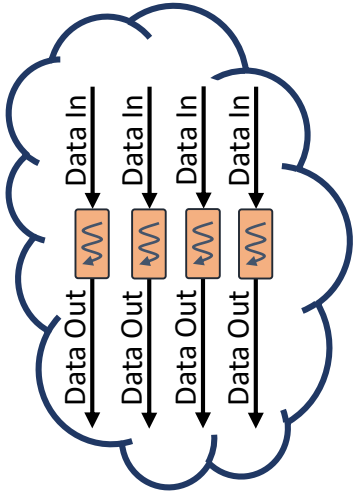
Manycore Array (Cont.)

- XY-dimension network-on-chip (NoC)
  - Unlimited deadlock-free communication
  - Manycore I/O uses same network
- Remote store programming model
  - Word writes into other tile's data memory
  - MIMD programming model
    - *Fine-grain parallelism through high-speed communication between tiles*
- Token-Queue architectural primitive
  - Reserves buffer space in remote core
  - Ensures buffer is filled before accessed
  - Tight producer-consumer synchronization
  - Streaming programming model
    - *Producer-consumer parallelism*





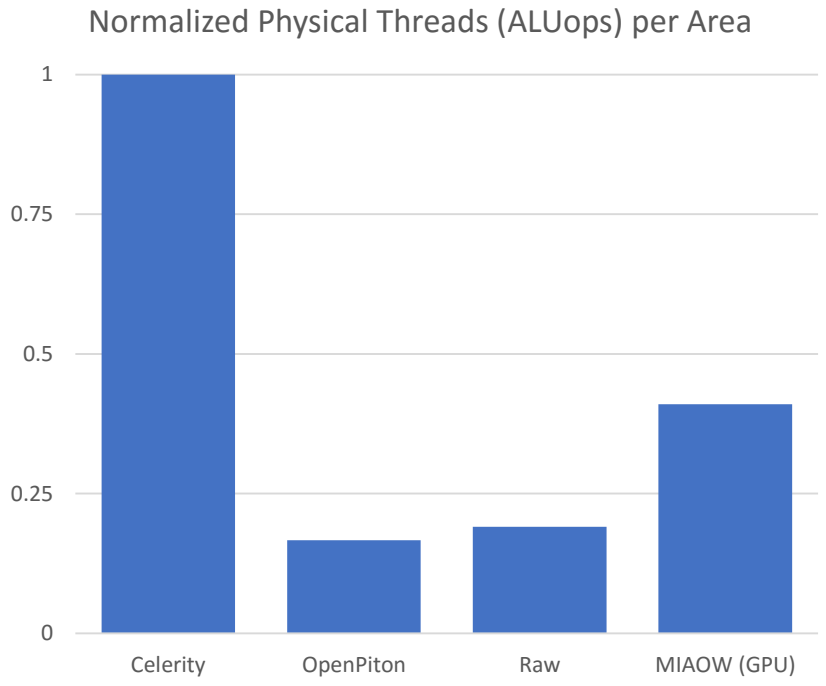
Stream Programming



SPMD Programming

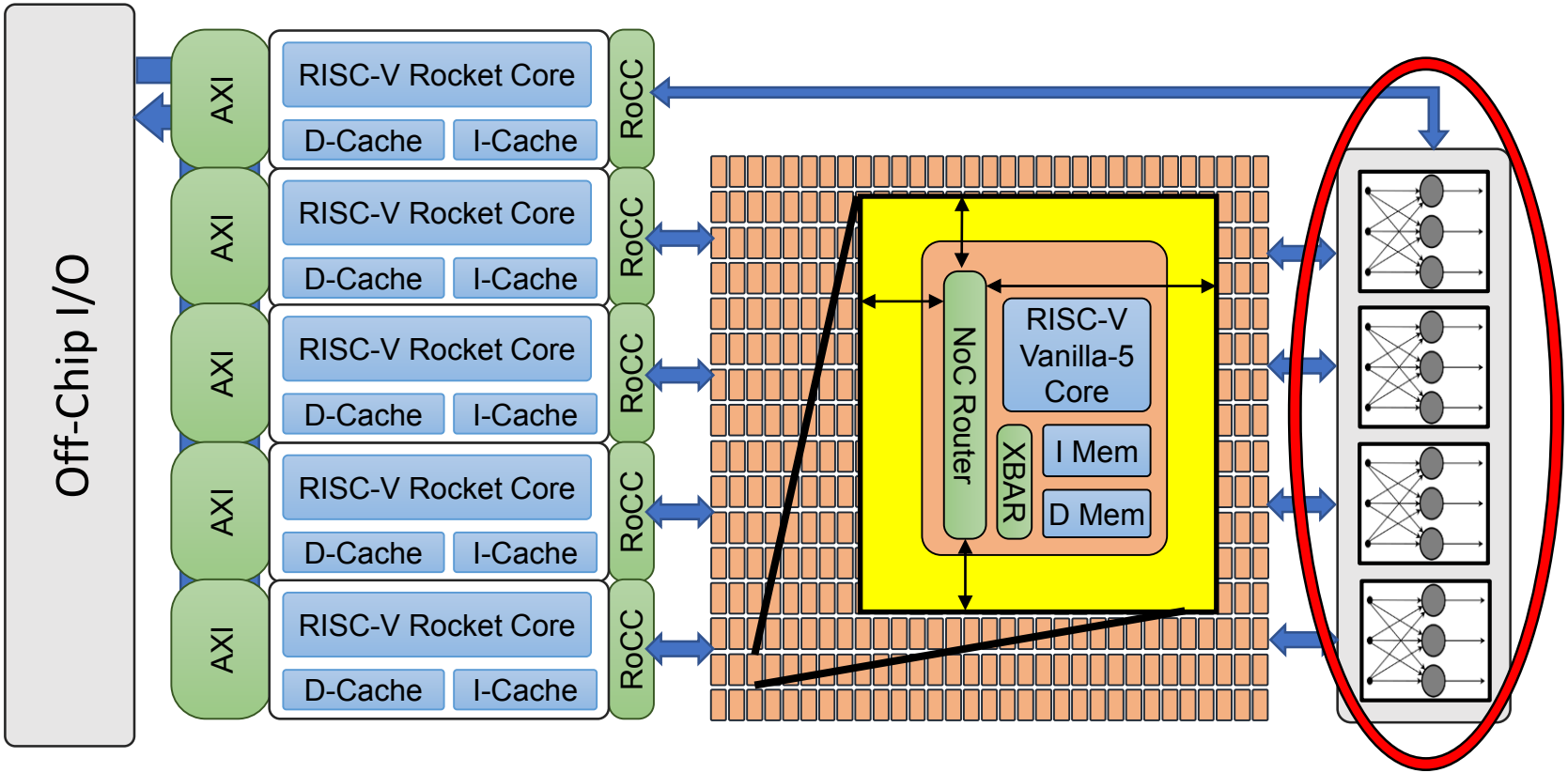
Manycore Array (Cont.)

	Configuration	Normalized Area (32nm)	Area Ratio
Celerity Tile @16nm	D-MEM = 4KB I-MEM = 4KB	0.024 * (32/16)^2 = 0.096 mm^2	1x
OpenPiton Tile @32nm	L1 D-Cache = 8KB L1 I-Cache = 8KB L1.5/L2 Cache = 40KB	1.17 mm^2 [1]	12x
Raw Tile @180nm	L1 D-Cache = 32KB L1 I-SRAM = 96KB	16.0 * (32/180)^2 = 0.506 mm^2	5.25x
MIAOW GPU Compute Unit Lane @32nm	VRF = 256KB SRF = 2KB	15.0 / 16 = 0.938 mm^2 [2]	9.75x



[1] J. Balkind, et al. “OpenPiton : An Open Source Manycore Research Framework,” in *the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016.
[2] R. Balasubramanian, et al. "Enabling GPGPU Low-Level Hardware Explorations with MIAOW: An Open-Source RTL Implementation of a GPGPU," in *ACM Transactions on Architecture and Code Optimization (TACO)*. 12.2 (2015): 21.

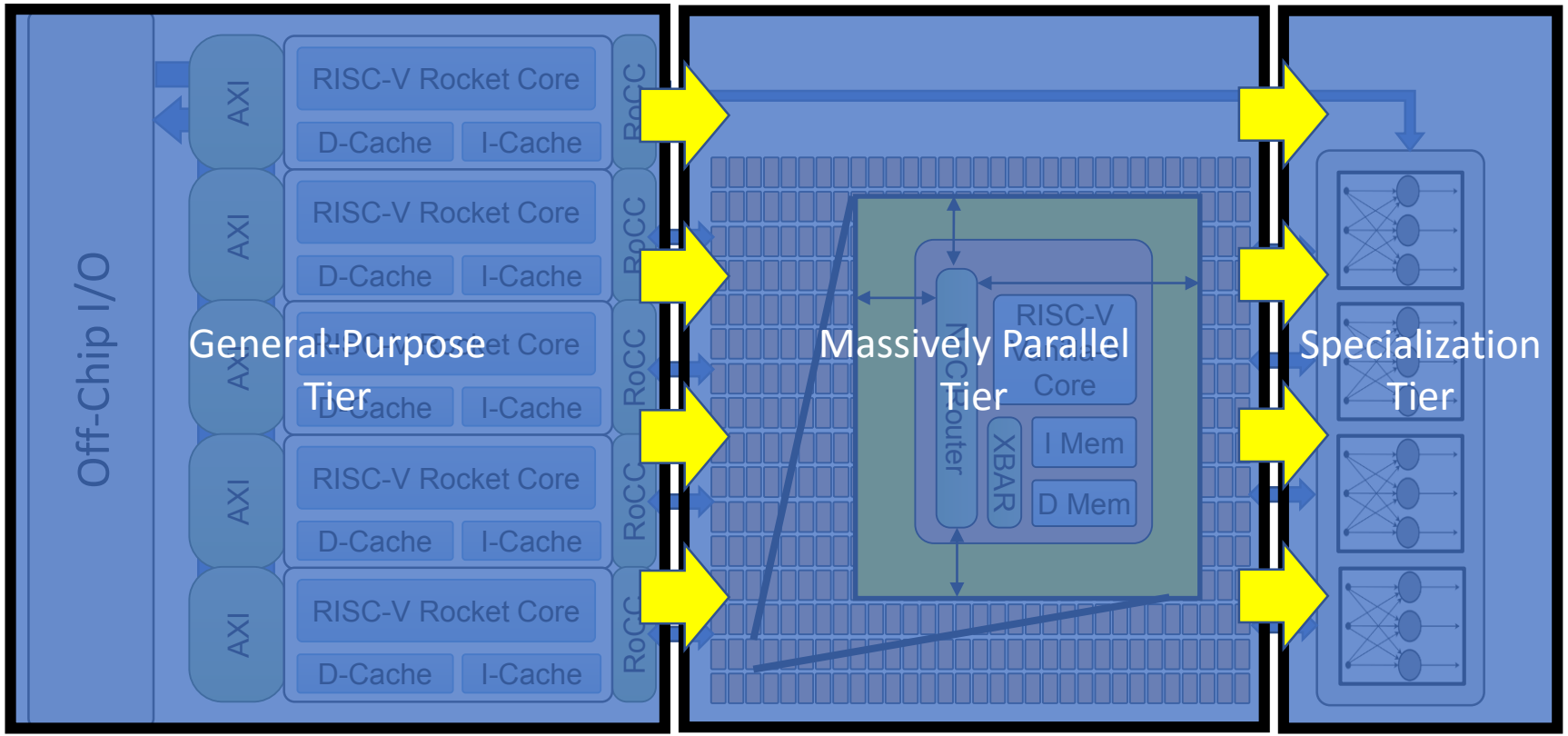
Celerity: Specialization Tier



Specialization Tier: Binarized Neural Network

- Role of the Specialization Tier
  - Achieves high energy efficiency through specialization
- In *Celerity*
  - Binarized Neural Network (BNN)
    - Energy-efficient convolutional neural network implementation
    - 13.4 MB model size with 9 total layers
      - 1 Fixed-point convolutional layer
      - 6 Binary convolutional layers
      - 2 Dense fully connected layers
    - Batch norm calculations done after each layer
  - 0.356 mm^2 @ 625 MHz

Parallel Links Between Tiers



Celerity Overview

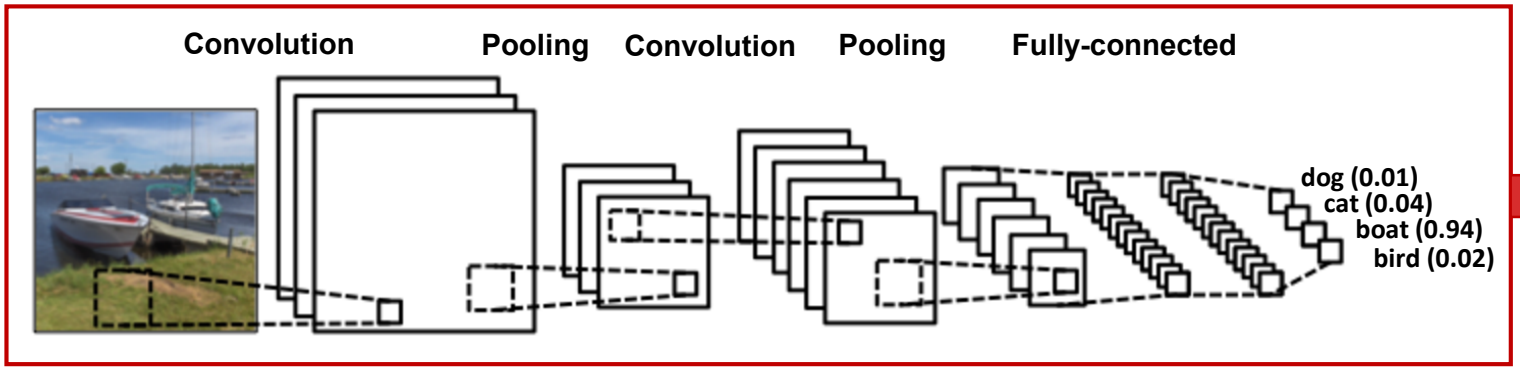
Tiered Accelerator Fabric

Case Study: Mapping Flexible Image Recognition to a Tiered Accelerator Fabric

Meeting Aggressive Time Schedule

Conclusion

Case Study: Mapping Flexible Image Recognition to a Tiered Accelerator Fabric

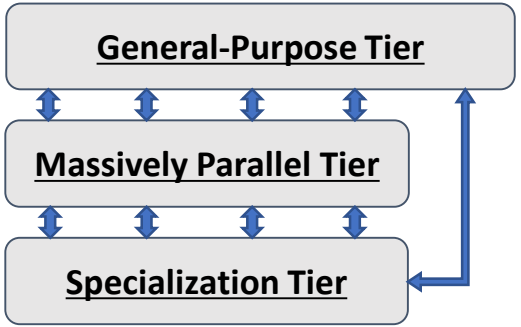


Three steps to map applications to tiered accelerator fabric:

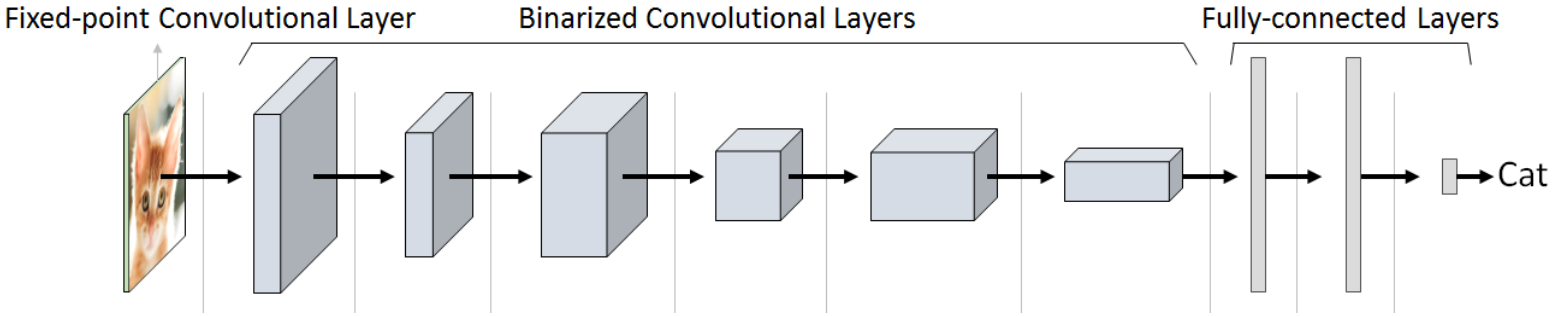
Step 1. Implement the algorithm using the general-purpose tier

Step 2. Accelerate the algorithm using either the massively parallel tier **OR** the specialization tier

Step 3. Improve performance by cooperatively using both the specialization **AND** the massively parallel tier



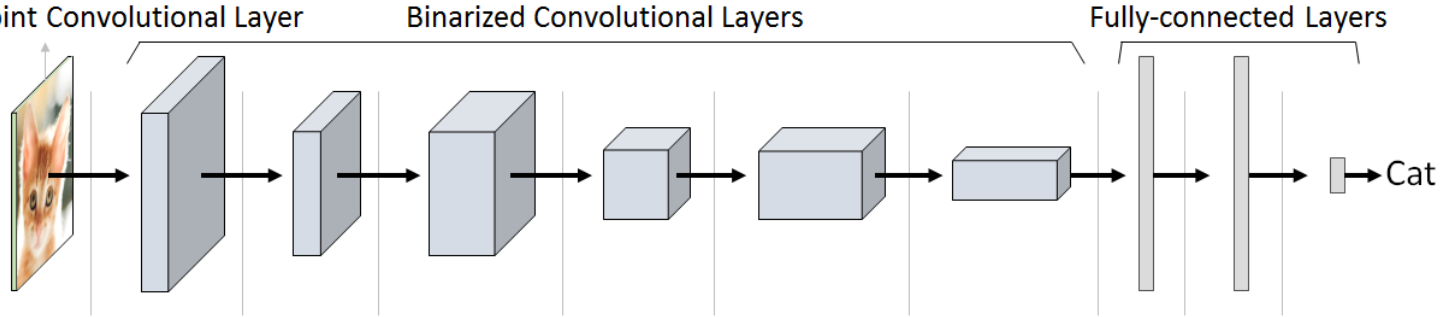
Step 1: Algorithm to Application
Binarized Neural Networks



- Training usually uses floating point, while inference usually uses lower precision weights and activations (often 8-bit or lower) to reduce implementation complexity
- Rastergari et al. [3] and Courbariaux et al. [4] have recently shown single-bit precision weights and activations can achieve an accuracy of 89.8% on CIFAR-10
- Performance target requires ultra-low latency (batch size of one) and high throughput (60 classifications/second)

[3] M. Rastergari, et al. "Xnor-net: Imagenet classification using binary convolutional neural networks," In *European Conference on Computer Vision*, 2016.
[4] M. Courbariaux, et al. "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," arXiv preprint arXiv:1602.02830 (2016).

Step 1: Algorithm to Application
Characterizing BNN Execution

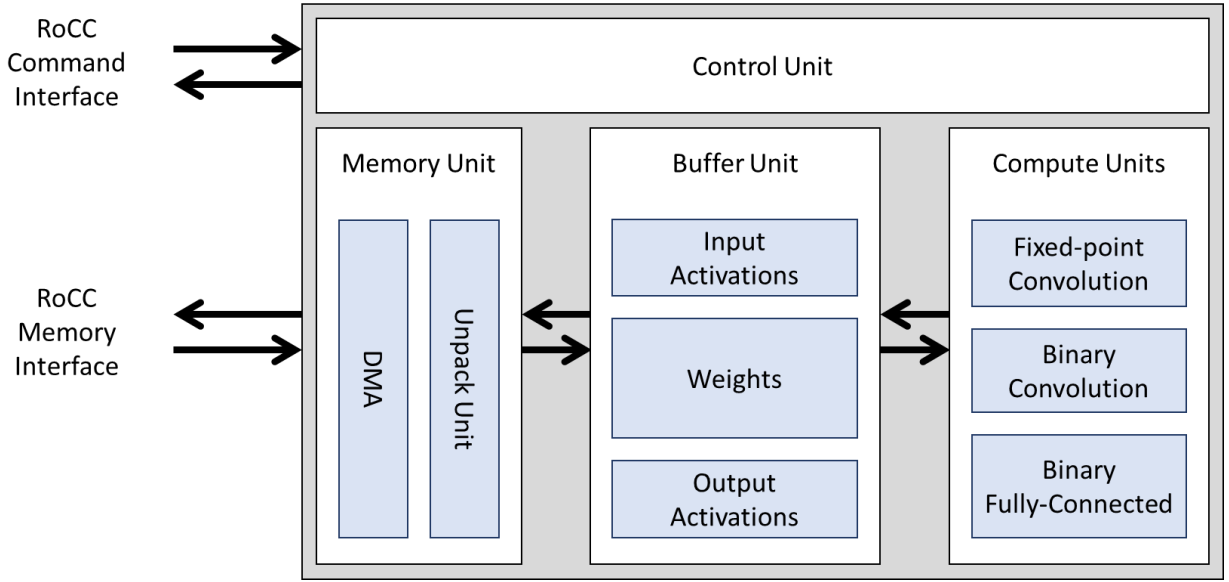


Execution Time	2.15%	23.11%	12.02%	23.98%	12.85%	25.68%	0.2%	0.02%	<0.01%
Weights' Size (Mbits)	0.003	0.14	0.28	0.56	1.13	2.25	8	1	0.01
Input Size (Kbits)	60	128	32	64	16	32	8	1	1

- Using just the general-purpose tier is 200x slower than performance target
- Binarized convolutional layers consume over 97% of dynamic instruction count
- Perfect acceleration of just the binarized convolutional layers is still 5x slower than performance target
- Perfect acceleration of all layers using the massively parallel tier could meet performance target but with significant energy consumption

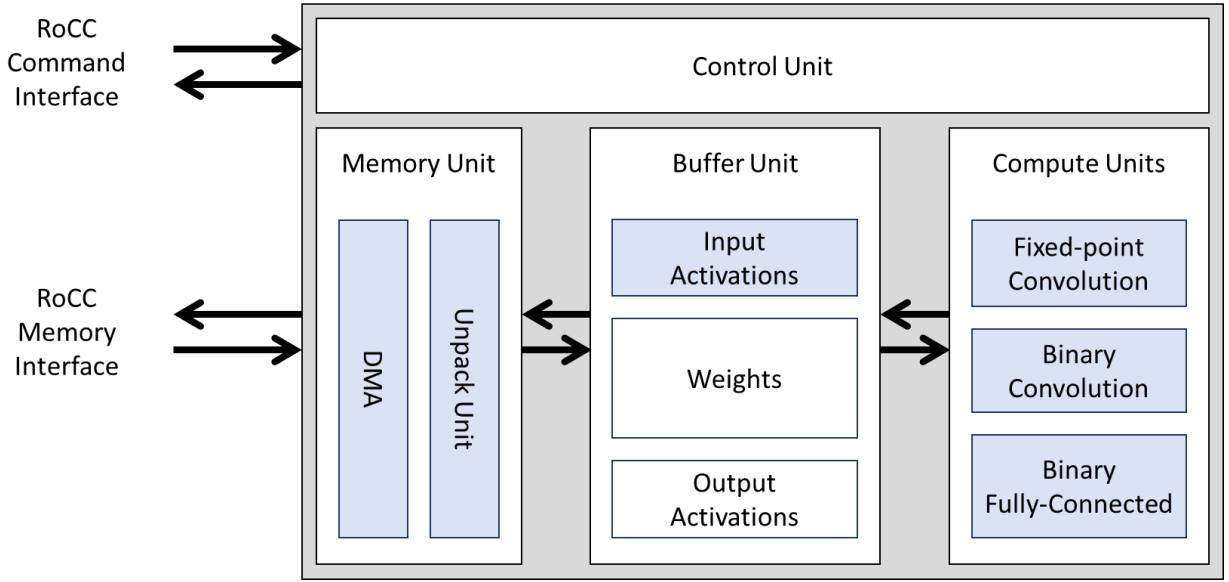
Step 2: Application to Accelerator

BNN Specialized Accelerator



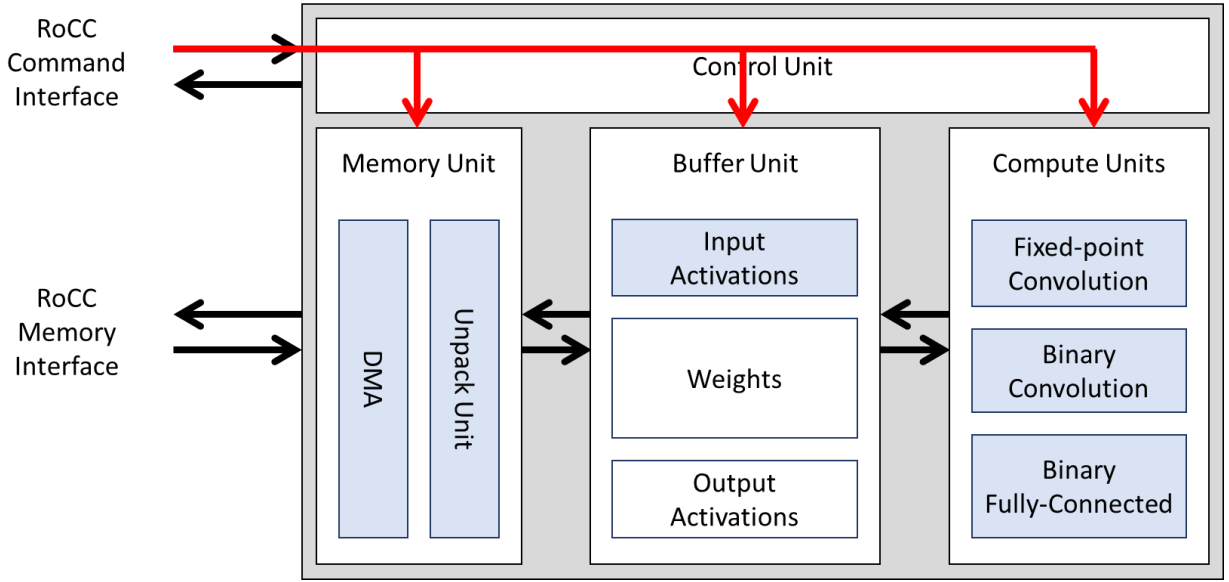
Step 2: Application to Accelerator

BNN Specialized Accelerator



Step 2: Application to Accelerator

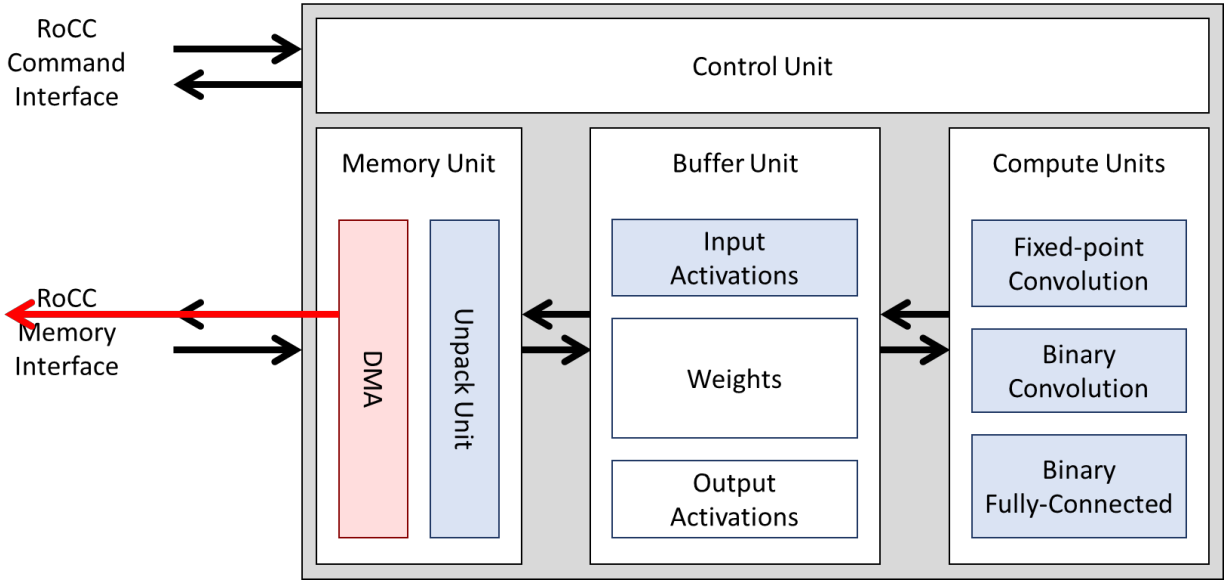
BNN Specialized Accelerator



1. Accelerator is configured to process a layer through RoCC command messages

Step 2: Application to Accelerator

BNN Specialized Accelerator

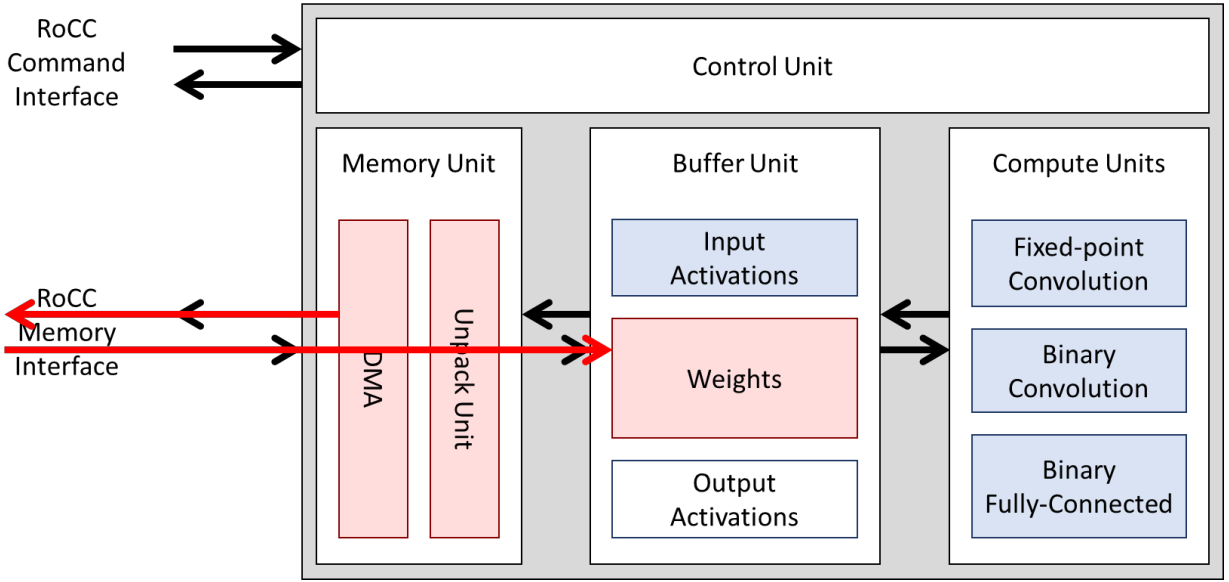


1. Accelerator is configured to process a layer through RoCC command messages

2. Memory Unit starts streaming the weights into the accelerator and unpacking the binarized weights into appropriate buffers

Step 2: Application to Accelerator

BNN Specialized Accelerator

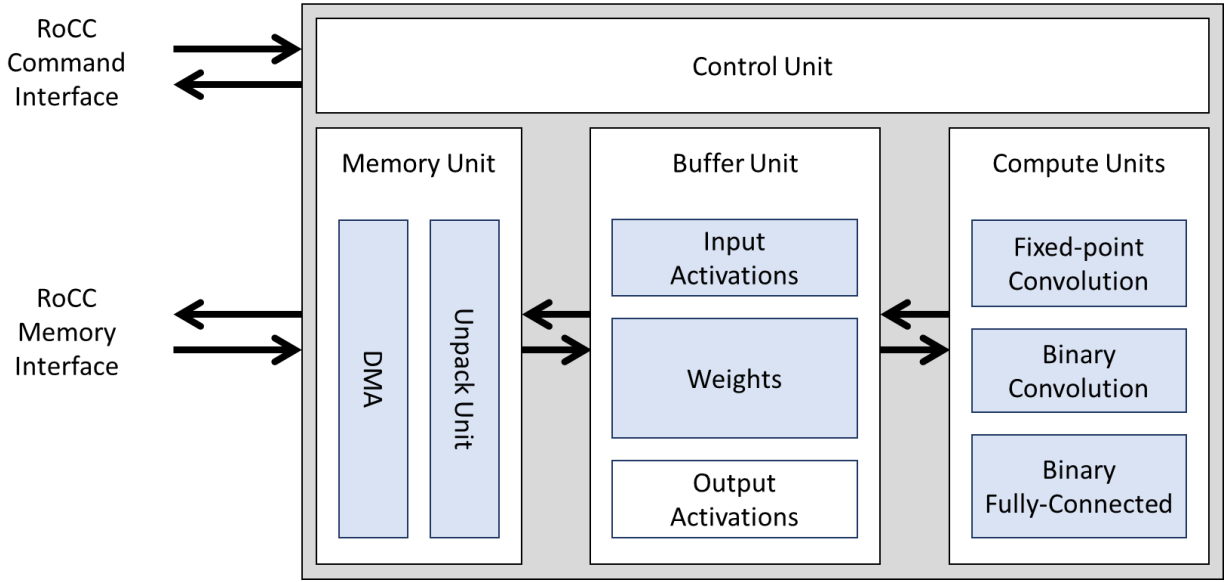


1. Accelerator is configured to process a layer through RoCC command messages

2. Memory Unit starts streaming the weights into the accelerator and unpacking the binarized weights into appropriate buffers

Step 2: Application to Accelerator

BNN Specialized Accelerator

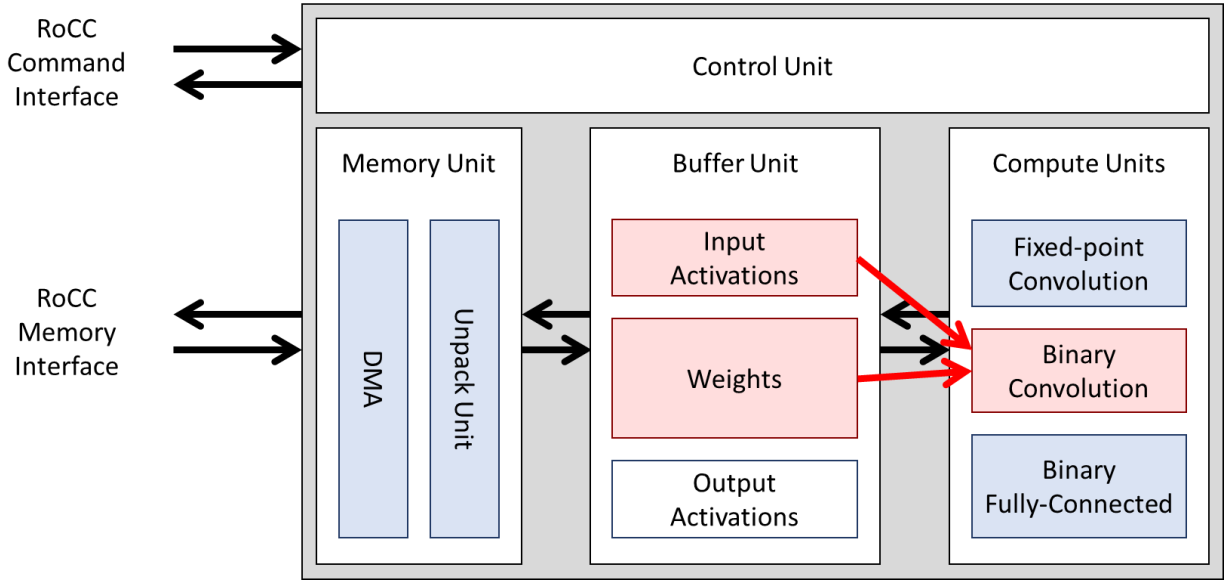


1. Accelerator is configured to process a layer through RoCC command messages

2. Memory Unit starts streaming the weights into the accelerator and unpacking the binarized weights into appropriate buffers

Step 2: Application to Accelerator

BNN Specialized Accelerator



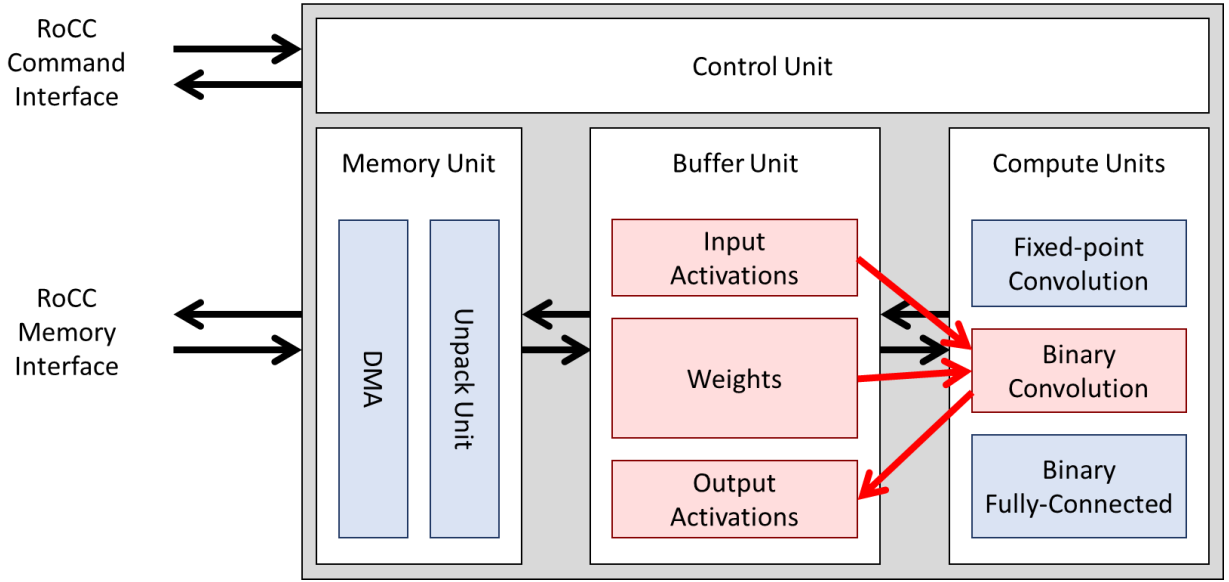
1. Accelerator is configured to process a layer through RoCC command messages

2. Memory Unit starts streaming the weights into the accelerator and unpacking the binarized weights into appropriate buffers

3. Binary convolution compute unit processes input activations and weights to produce output activations

Step 2: Application to Accelerator

BNN Specialized Accelerator



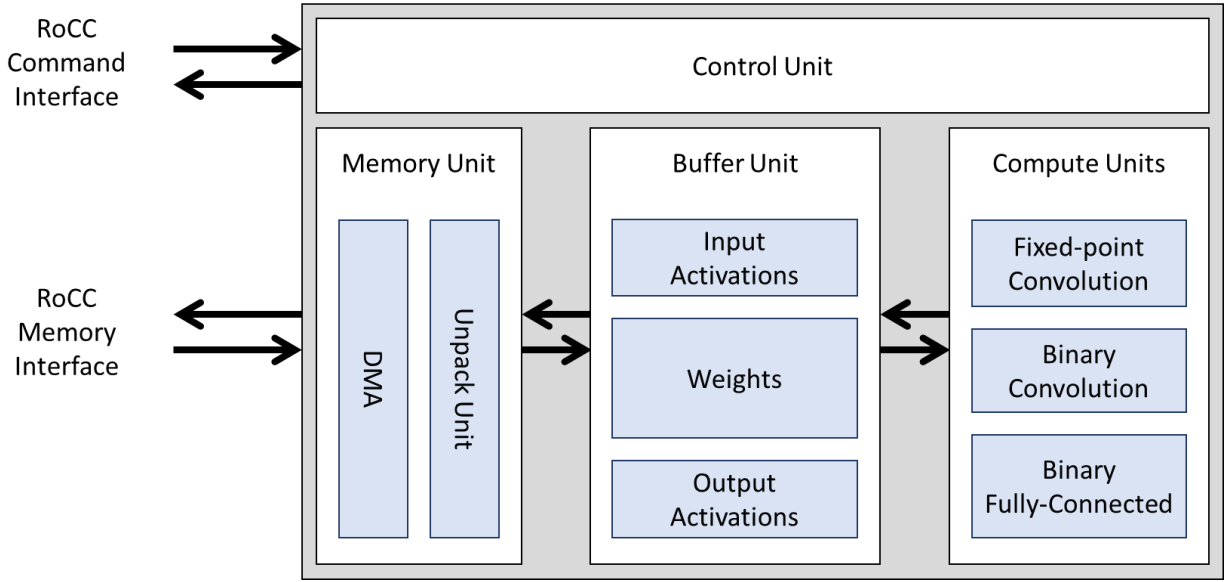
1. Accelerator is configured to process a layer through RoCC command messages

2. Memory Unit starts streaming the weights into the accelerator and unpacking the binarized weights into appropriate buffers

3. Binary convolution compute unit processes input activations and weights to produce output activations

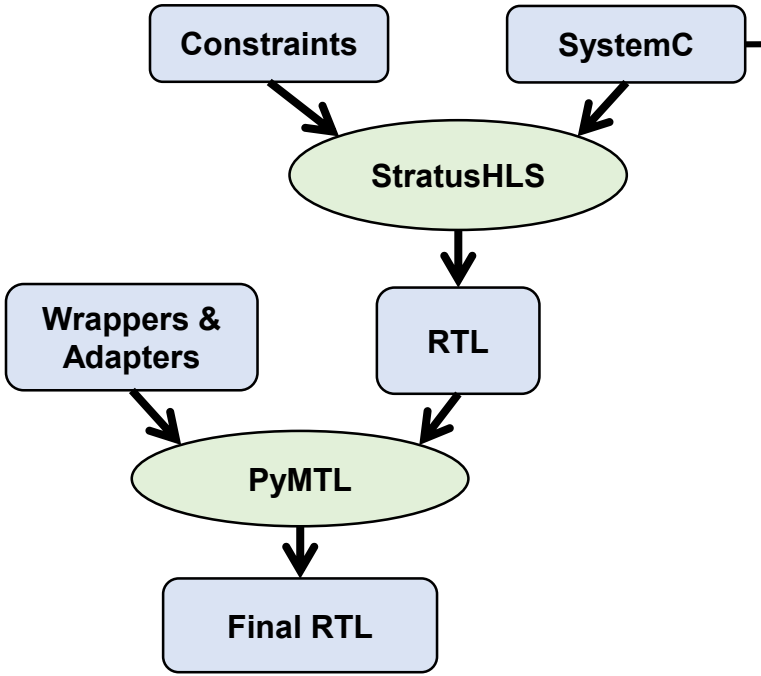
Step 2: Application to Accelerator

BNN Specialized Accelerator



1. Accelerator is configured to process a layer through RoCC command messages
2. Memory Unit starts streaming the weights into the accelerator and unpacking the binarized weights into appropriate buffers
3. Binary convolution compute unit processes input activations and weights to produce output activations

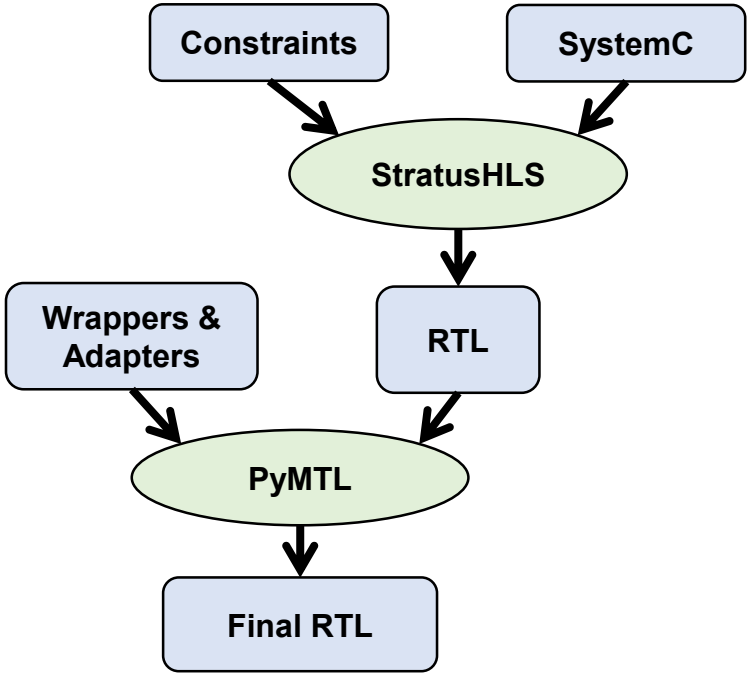
Step 2: Application to Accelerator
Design Methodology




```
void bnn::dma_req() {
 while( 1 ) {
  DmaMsg msg = dma_req.get();

  for ( int i = 0; i < msg.len; i++ ) {
   HLS_PIPELINE_LOOP( HARD_STALL, 1 );

   int     req_type = 0;
   word_t data = 0;
   addr_t addr = msg.base + i*8;

   if ( type == DMA_TYPE_WRITE ) {
    data = msg.data;
    req_type = MemReqMsg::WRITE;
   } else {
    req_type = MemReqMsg::READ;
   }

   memreq.put(MemReqMsg(req_type,addr,data));
  }

  dma_resp.put(DMA_REQ_DONE);
 }
}
```

Step 2: Application to Accelerator

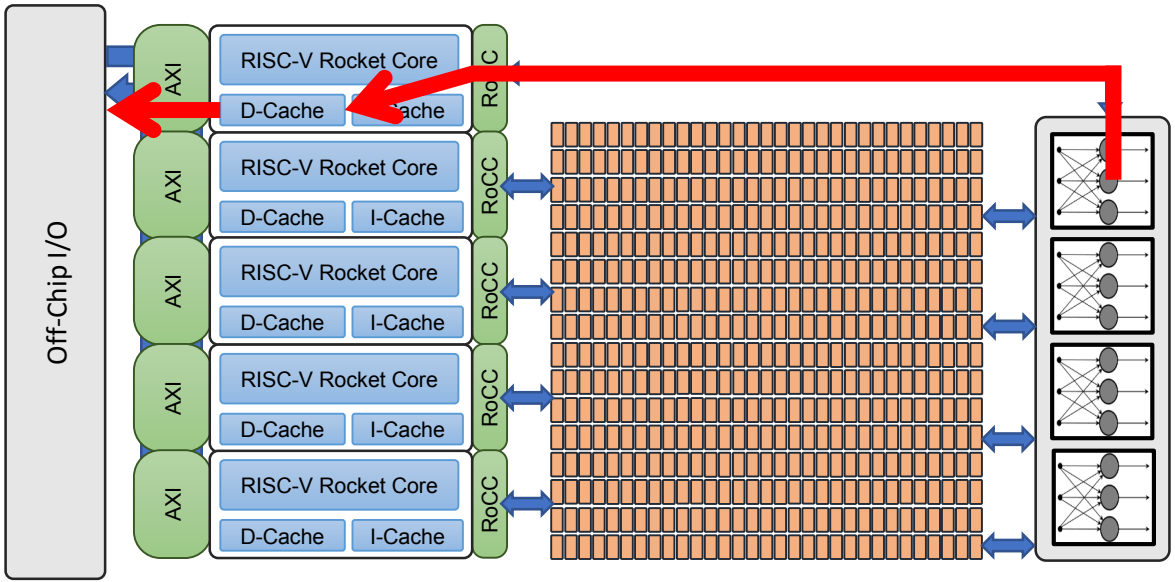
Design Methodology



- HLS enabled quick implementation of an accelerator for an emerging algorithm
  - Algorithm to initial accelerator in weeks
  - Rapid design-space exploration
- HLS greatly simplified timing closure
  - Improved clock frequency by 43% in few days
  - Easily mitigated long paths at the interfaces with latency insensitive interfaces and pipeline register insertion
- HLS tools are still evolving
  - Six weeks to debug tool bug with data-dependent access to multi-dimensional arrays

Step 2: Application to Accelerator

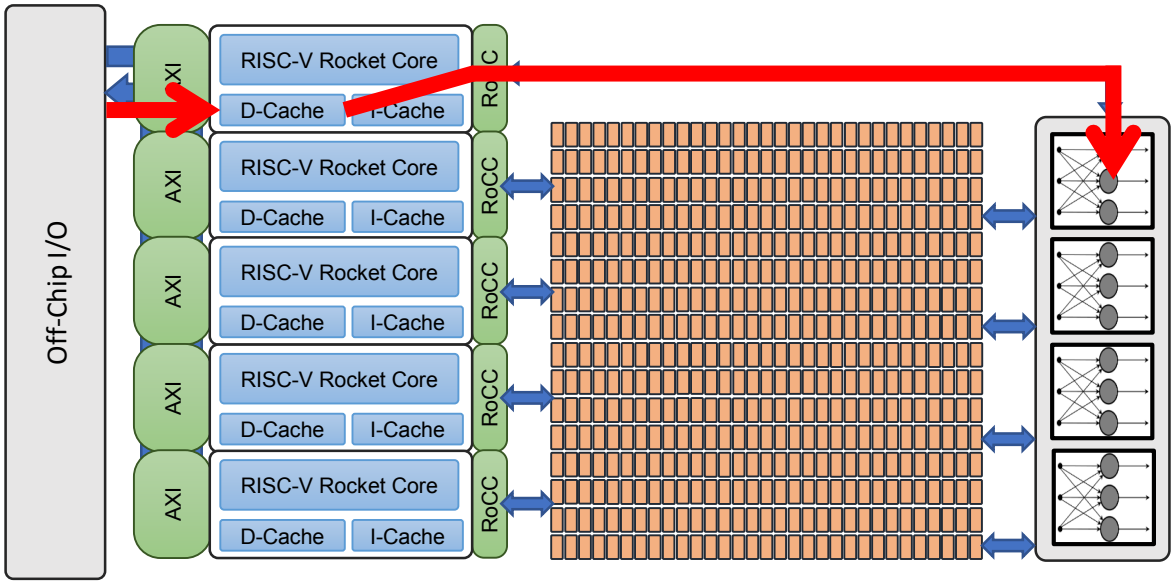
General-Purpose Tier for Weight Storage



- The BNN specialized accelerator can use one of the Rocket cores' caches to load every layer's weights; but, it is inefficient due to off-chip traffic

Step 2: Application to Accelerator

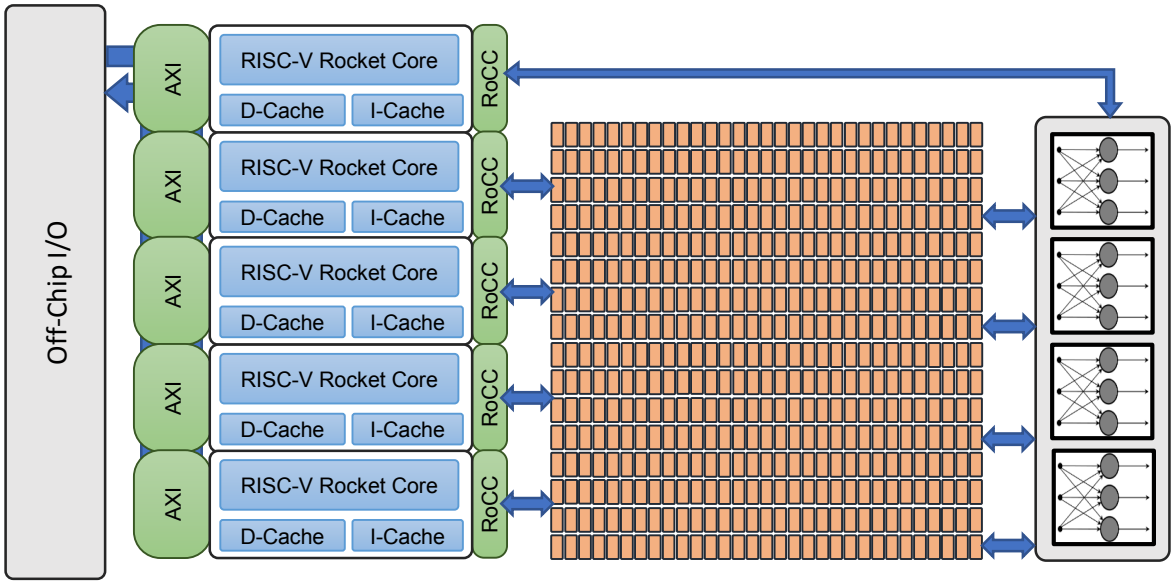
General-Purpose Tier for Weight Storage



- The BNN specialized accelerator can use one of the Rocket cores' caches to load every layer's weights; but, it is inefficient due to off-chip traffic

Step 2: Application to Accelerator

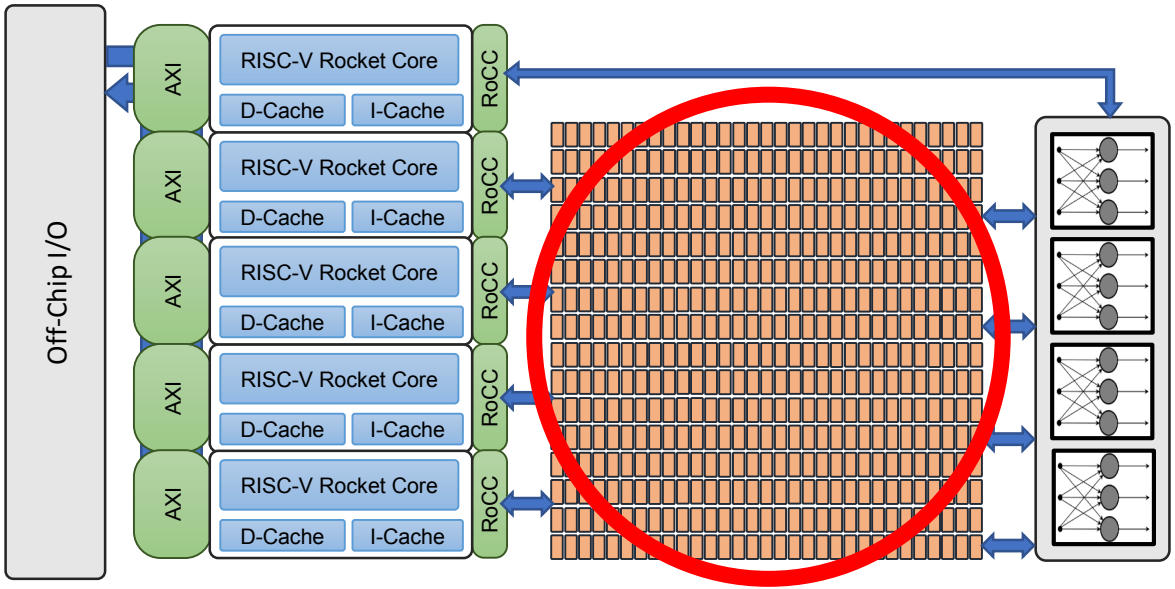
General-Purpose Tier for Weight Storage



- The BNN specialized accelerator can use one of the Rocket cores' caches to load every layer's weights; but, it is inefficient due to off-chip traffic
- A large L2 or more storage in the BNN specialized accelerator could improve performance

Step 3: Assisting Accelerators

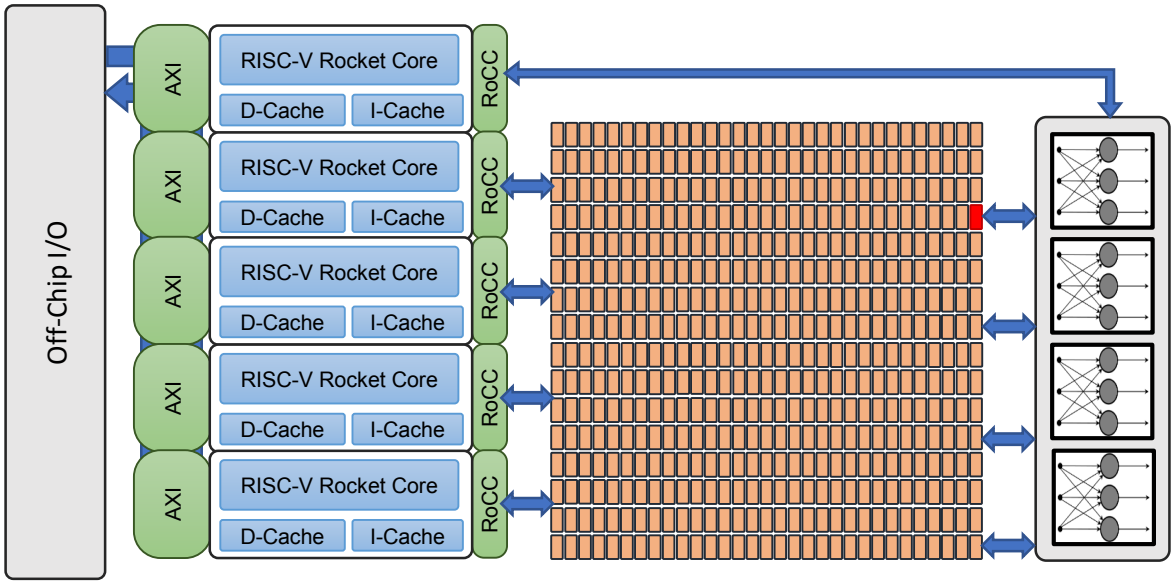
General-Purpose Tier for Weight Storage



- The BNN specialized accelerator can use one of the Rocket cores' caches to load every layer's weights; but, it is inefficient due to off-chip traffic
- A large L2 or more storage in the BNN specialized accelerator could improve performance
- Instead, weights can be stored in the massively parallel tier

Step 3: Assisting Accelerators

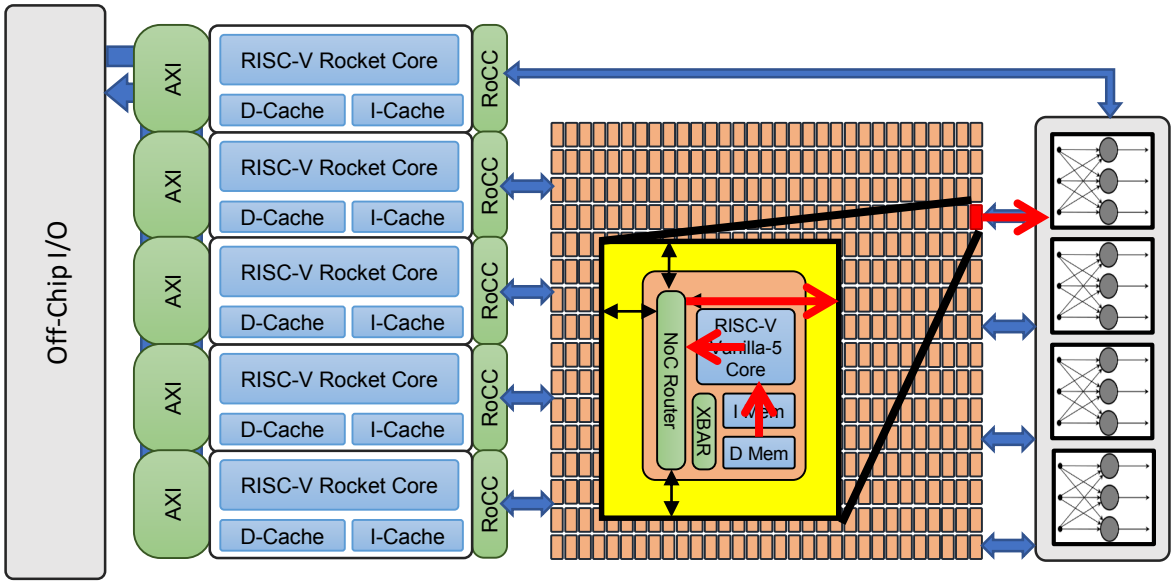
Massively Parallel Tier for Weight Storage



- The BNN specialized accelerator can use one of the Rocket cores' caches to load every layer's weights; but, it is inefficient due to off-chip traffic
- A large L2 or more storage in the BNN specialized accelerator could improve performance
- Instead, weights can be stored in the massively parallel tier
- Each core in the massively parallel tier executes a remote-load-store program to orchestrate sending weights to the specialization tier via a hardware FIFO

Step 3: Assisting Accelerators

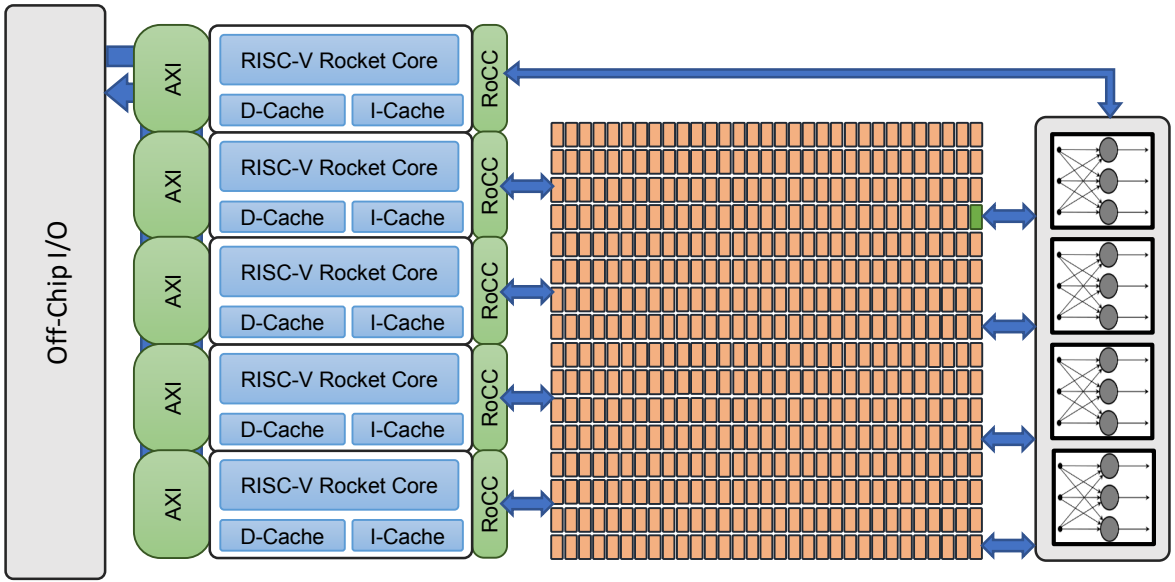
Massively Parallel Tier for Weight Storage



- The BNN specialized accelerator can use one of the Rocket cores' caches to load every layer's weights; but, it is inefficient due to off-chip traffic
- A large L2 or more storage in the BNN specialized accelerator could improve performance
- Instead, weights can be stored in the massively parallel tier
- Each core in the massively parallel tier executes a remote-load-store program to orchestrate sending weights to the specialization tier via a hardware FIFO

Step 3: Assisting Accelerators

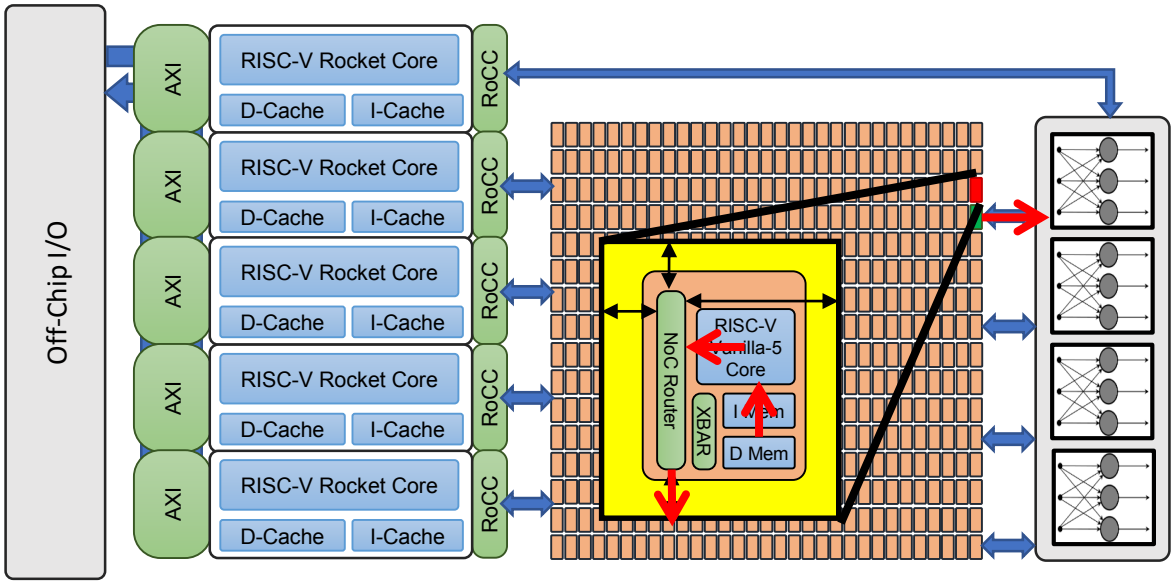
Massively Parallel Tier for Weight Storage



- The BNN specialized accelerator can use one of the Rocket cores' caches to load every layer's weights; but, it is inefficient due to off-chip traffic
- A large L2 or more storage in the BNN specialized accelerator could improve performance
- Instead, weights can be stored in the massively parallel tier
- Each core in the massively parallel tier executes a remote-load-store program to orchestrate sending weights to the specialization tier via a hardware FIFO

Step 3: Assisting Accelerators

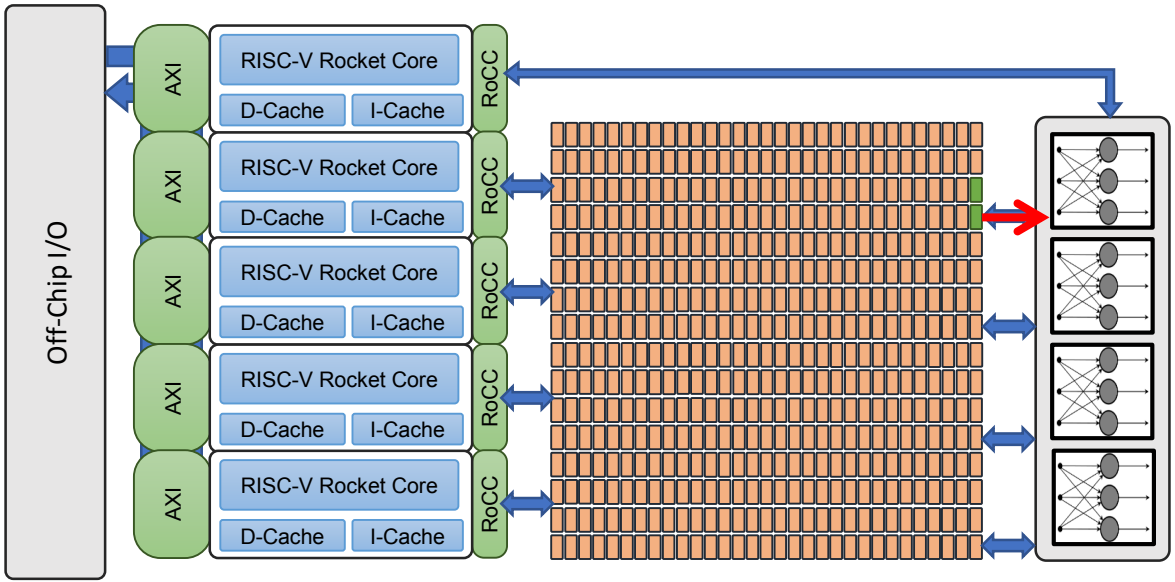
Massively Parallel Tier for Weight Storage



- The BNN specialized accelerator can use one of the Rocket cores' caches to load every layer's weights; but, it is inefficient due to off-chip traffic
- A large L2 or more storage in the BNN specialized accelerator could improve performance
- Instead, weights can be stored in the massively parallel tier
- Each core in the massively parallel tier executes a remote-load-store program to orchestrate sending weights to the specialization tier via a hardware FIFO

Step 3: Assisting Accelerators

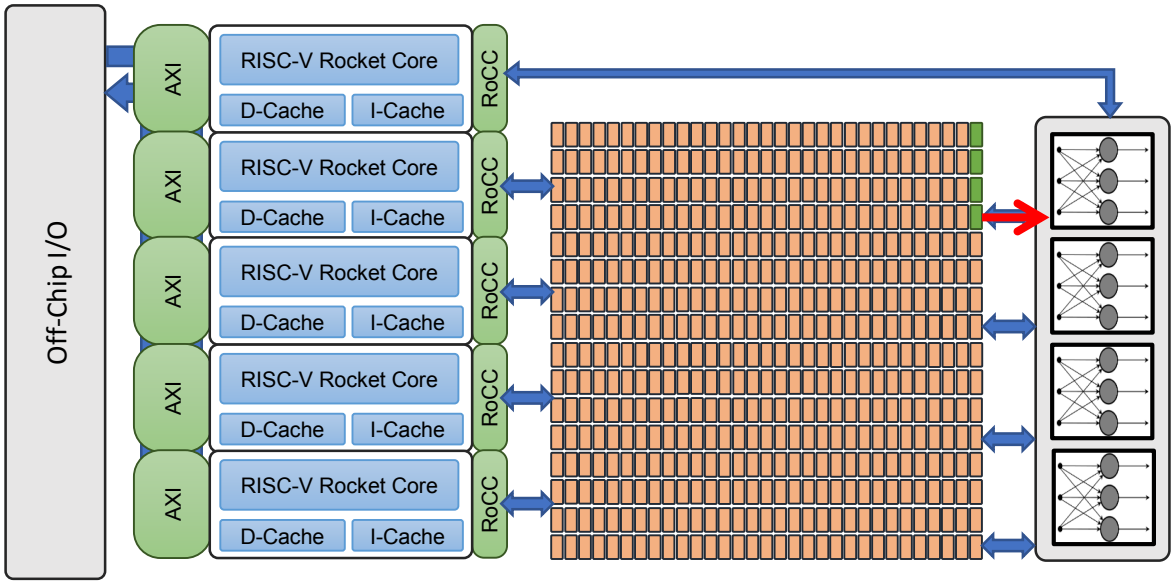
Massively Parallel Tier for Weight Storage



- The BNN specialized accelerator can use one of the Rocket cores' caches to load every layer's weights; but, it is inefficient due to off-chip traffic
- A large L2 or more storage in the BNN specialized accelerator could improve performance
- Instead, weights can be stored in the massively parallel tier
- Each core in the massively parallel tier executes a remote-load-store program to orchestrate sending weights to the specialization tier via a hardware FIFO

Step 3: Assisting Accelerators

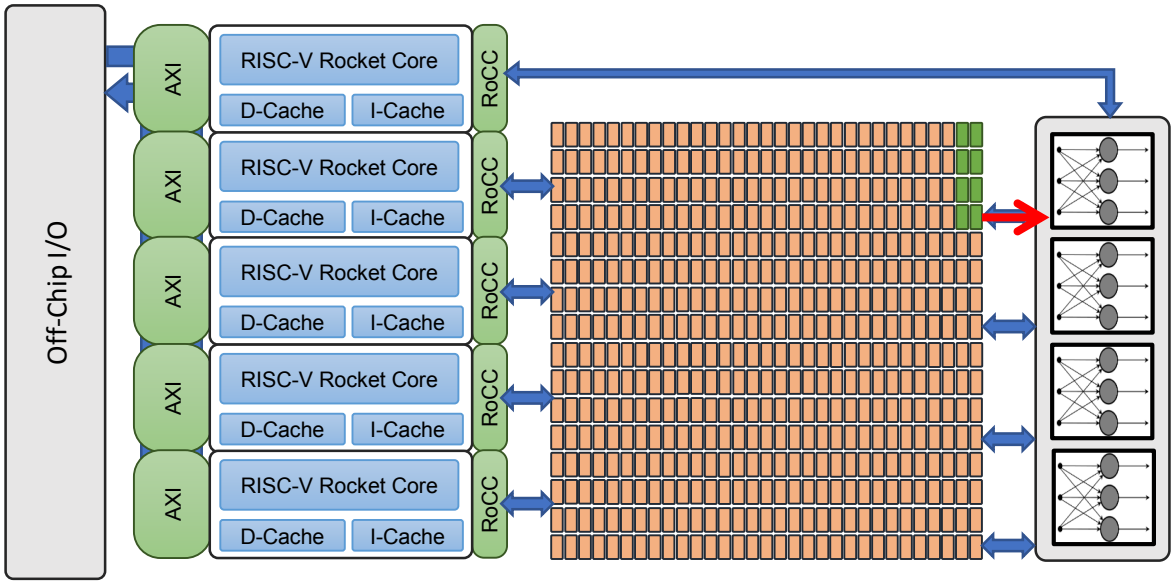
Massively Parallel Tier for Weight Storage



- The BNN specialized accelerator can use one of the Rocket cores' caches to load every layer's weights; but, it is inefficient due to off-chip traffic
- A large L2 or more storage in the BNN specialized accelerator could improve performance
- Instead, weights can be stored in the massively parallel tier
- Each core in the massively parallel tier executes a remote-load-store program to orchestrate sending weights to the specialization tier via a hardware FIFO

Step 3: Assisting Accelerators

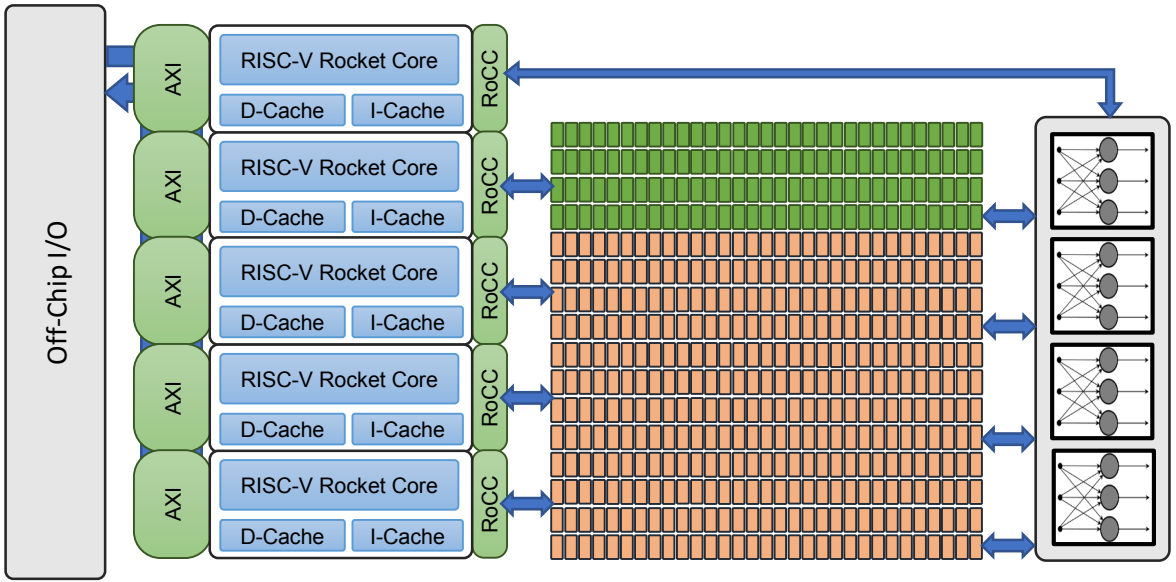
Massively Parallel Tier for Weight Storage



- The BNN specialized accelerator can use one of the Rocket cores' caches to load every layer's weights; but, it is inefficient due to off-chip traffic
- A large L2 or more storage in the BNN specialized accelerator could improve performance
- Instead, weights can be stored in the massively parallel tier
- Each core in the massively parallel tier executes a remote-load-store program to orchestrate sending weights to the specialization tier via a hardware FIFO

Step 3: Assisting Accelerators

Massively Parallel Tier for Weight Storage



- The BNN specialized accelerator can use one of the Rocket cores' caches to load every layer's weights; but, it is inefficient due to off-chip traffic
- A large L2 or more storage in the BNN specialized accelerator could improve performance
- Instead, weights can be stored in the massively parallel tier
- Each core in the massively parallel tier executes a remote-load-store program to orchestrate sending weights to the specialization tier via a hardware FIFO

Performance Benefits of Cooperatively Using the Massively Parallel and the Specialization Tiers

	General-Purpose Tier	Specialization Tier	Specialization + Massively Parallel Tiers
Runtime per Image (ms)	4,024	5.8	3.3
Speedup	1x	~700x	~1,220x

General-Purpose Tier	Software implementation assuming ideal performance estimated with an optimistic one instruction per cycle
Specialization Tier	Full-system RTL simulation of the BNN specialized accelerator running with a frequency of 625 MHz
Specialization + Massively Parallel Tiers	Full-system RTL simulation of the BNN specialized accelerator with the weights being streamed from the manycore

Celerity Overview

Tiered Accelerator Fabric

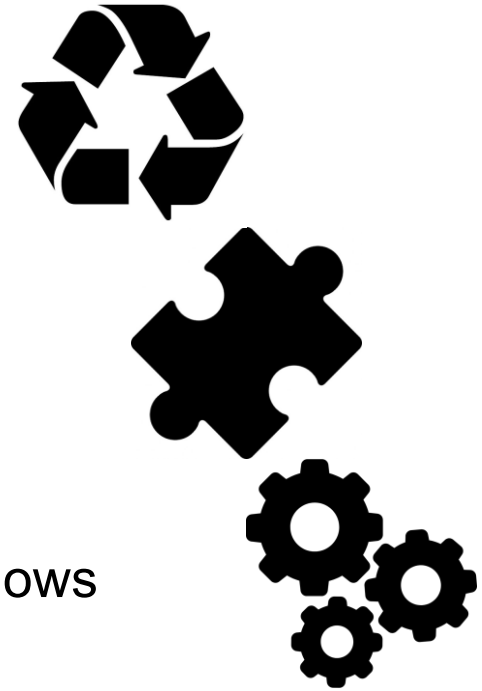
Case Study: Mapping Flexible Image Recognition to a Tiered Accelerator Fabric

Meeting Aggressive Time Schedule

Conclusion

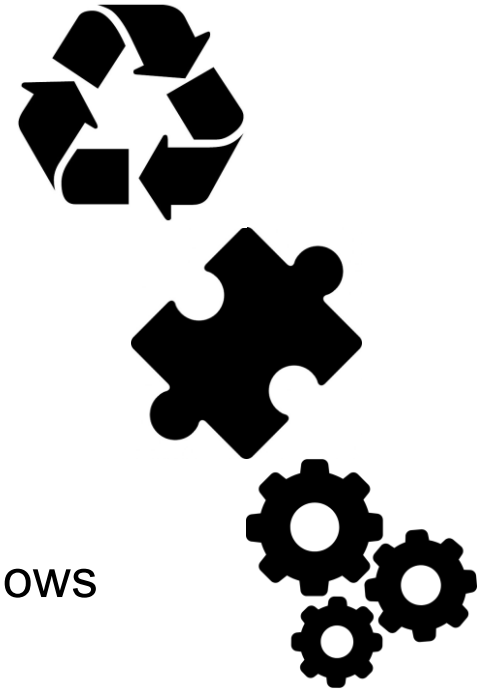
How to make a complex SoC?

- Reuse
  - Open-source and third-party IP
  - Extensible and parameterizable designs
- Modularize
  - Agile design and development
  - Early interface specification
- Automate
  - Abstracted implementation and testing flows
  - Highly automated design



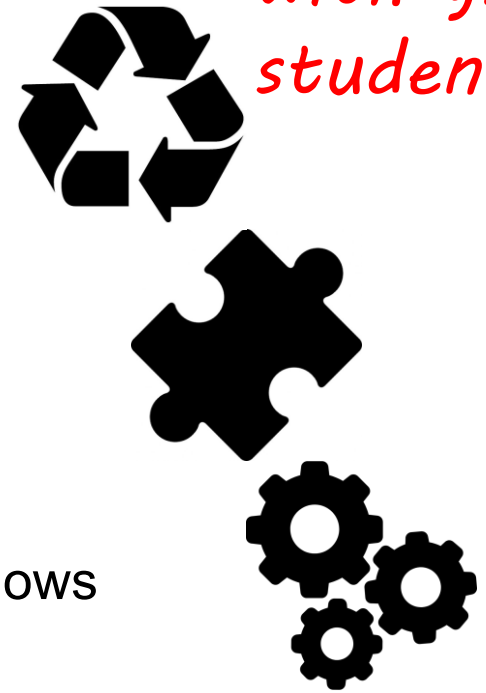
How to make a complex SoC? *in 9 months*

- Reuse
  - Open-source and third-party IP
  - Extensible and parameterizable designs
- Modularize
  - Agile design and development
  - Early interface specification
- Automate
  - Abstracted implementation and testing flows
  - Highly automated design



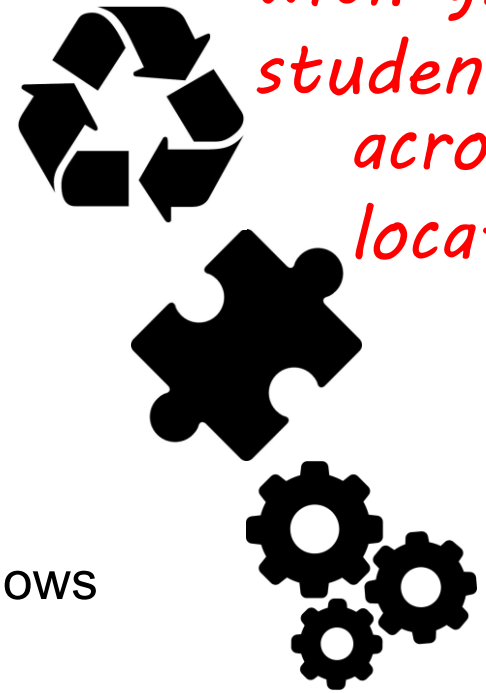
How to make a complex SoC? *in 9 months with grad students*

- Reuse
  - Open-source and third-party IP
  - Extensible and parameterizable designs
- Modularize
  - Agile design and development
  - Early interface specification
- Automate
  - Abstracted implementation and testing flows
  - Highly automated design



How to make a complex SoC? in 9 months with grad students across 4 locations

- Reuse
  - Open-source and third-party IP
  - Extensible and parameterizable designs
- Modularize
  - Agile design and development
  - Early interface specification
- Automate
  - Abstracted implementation and testing flows
  - Highly automated design

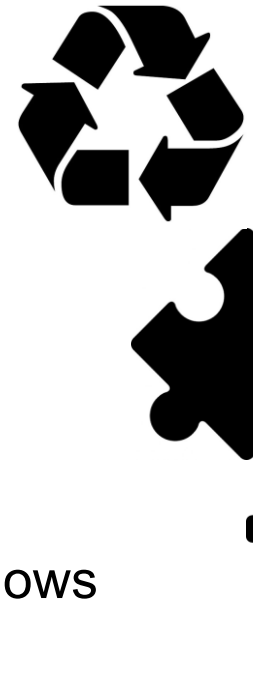


How to make a complex SoC? in 9 months with grad students across 4 locations in 16nm

- Reuse
  - Open-source and third-party IP
  - Extensible and parameterizable designs
- Modularize
  - Agile design and development
  - Early interface specification
- Automate
  - Abstracted implementation and testing flows
  - Highly automated design

How to make a complex SoC? in 9 months with grad students across 4 locations in 16nm with $1.3M

- Reuse
  - Open-source and third-party IP
  - Extensible and parameterizable designs
- Modularize
  - Agile design and development
  - Early interface specification
- Automate
  - Abstracted implementation and testing flows
  - Highly automated design



Reuse

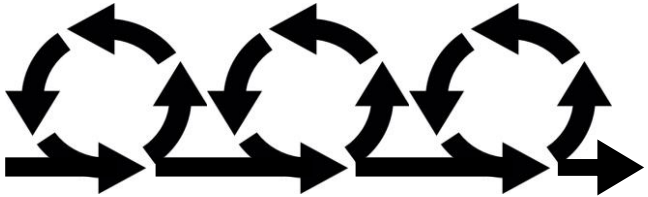
- Basejump: Open-source polymorphic HW components
  - **Design libraries**: BSG IP Cores, BGA Package, I/O Pad Ring
  - **Test infrastructure**: Double Trouble PCB, Real Trouble PCB
  - Available at bjump.org
- RISC-V: Open-source ISA
  - **Rocket core**: high performance RV64G in-order core
  - **Vanilla-V**: high efficiency RV32IM in-order core
- RoCC: Open-source on-chip interconnect
  - Common interface to connect all 3 compute tiers
- Extensible designs
  - **BSG Manycore**: fully parameterized RTL and APR scripts
- Third Party IP
  - ARM Standard Cells, I/O cells, RF/SRAM generators

RISC-V

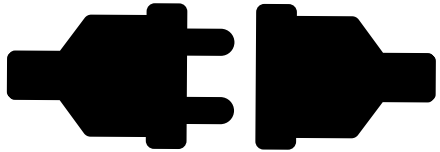
ARM ARTISAN
Physical IP

Modularize

- Agile design
  - Hierarchical design to reduce tool time
  - Optimize designs at the component level
  - Black-box designs for use across teams
  - SCRUM-like task management
  - Sprinting to “tape-ins”



- Establish interfaces early
  - Establish design interfaces early (RoCC, Basejump)
  - Use latency-insensitive interfaces to remove cross-module timing dependencies
  - Identify specific deliverables between different teams (esp. analog→digital)

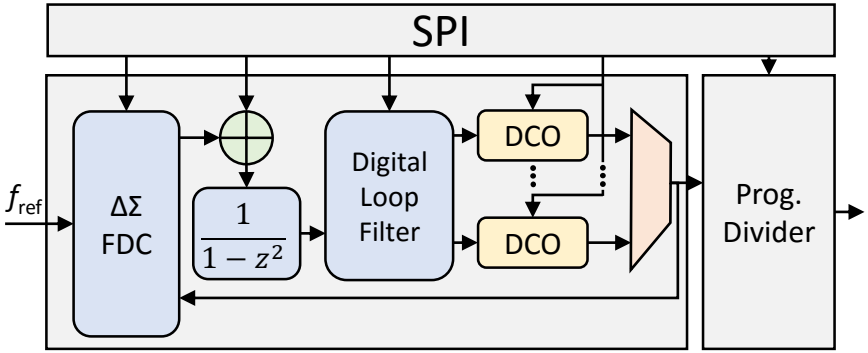


Automate

- Abstract implementation and testing flows
  - Develop implementation flow adaptable to arbitrary designs
  - Use validated IP components to focus only on integration testing
  - Use high-level testing abstractions to speed up test development (PyMTL)
- Automate design using tools
  - Use High-Level Synthesis to speed up design-space exploration and implementation
  - Use digital design flow to create traditionally analog components

Synthesizable PLL

- Reuse
  - Interfaces and some components reused from previous designs
- Modularize
  - Controlled via SPI-like interface
  - Isolated voltage domain for all 3 PLLs to remove power rail noise
- Automate
  - Fully synthesized using digital standard cells
  - Manual placement of ring oscillators, auto-placement of other logic
  - Very easy to create additional DCOs that cover additional frequency ranges

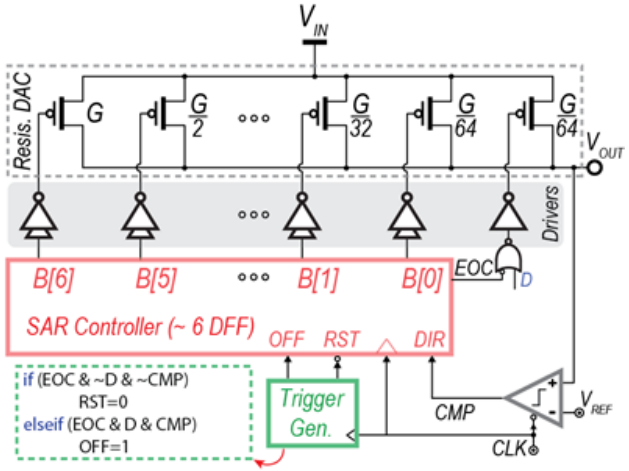


Area	0.0059 mm^2
Frequency range*	20 - 3000 MHz
Frequency step*	2%
Period jitter*	2.5 ps

* Collected via SPICE on extracted netlist

Synthesizable LDO

- Reuse
  - Taped out and tested in 65nm [5], waiting on 16nm results
- Automate
  - Fully synthesized controller
  - Custom power switching transistors
  - Post-silicon tunable
- Compared to conventional N-bit digital LDOs:
  - $2^N/N$ times smaller
  - $2^N/N$ times faster
  - 2^N times lower power
  - $2^{2N}/N$ better FoM



Controller Area	< 0.0023 mm^2
Decap Area	< 0.0741 mm^2
Voltage Range	0.45 – 0.85 V
Peak Efficiency	> 99.8 %

[5] L. Salem et al. “20.3 A 100nA-to-2mA successive-approximation digital LDO with PD compensation and sub-LSB duty control achieving a 15.1 ns response time at 0.5 V,” In *International Solid-State Circuits Conference (ISSCC)*, 2017.

Celerity Overview

Tiered Accelerator Fabric

Case Study: Mapping Flexible Image Recognition to a Tiered Accelerator Fabric

Meeting Aggressive Time Schedule

Conclusion

Conclusion

- Tiered accelerator fabric: an architectural template for embedded workloads that enable performance gains and energy savings without sacrificing programmability

- Celerity: a case study for accelerating low-latency, flexible image recognition using a binarized neural network that illustrates the potential for tiered accelerator fabrics

- Reuse, modularization, and automation enabled an academic-only group to tape out a 16nm ASIC with 511 RISC-V cores and a specialized binarized neural network accelerator in only 9 months

Acknowledgements

This work was funded by DARPA under the
Circuit Realization At Faster Timescales (CRAFT) program

Special thanks to Dr. Linton Salmon for program support and coordination