Experiences Using the RISC-V Ecosystem to Design an Accelerator-Centric SoC in TSMC 16nm

Tutu Ajayi[2], Khalid Al-Hawaj[1], Aporva Amarnath[2], Steve Dai[1],
Scott Davidson[4], Paul Gao[4], Gai Liu[1], Anuj Rao[4],
Austin Rovinski[2], Ningxiao Sun[4], **Christopher Torng**[1], Luis Vega[4],
Bandhav Veluri[4], **Shaolin Xie**[4], Chun Zhao[4] Ritchie Zhao[1],

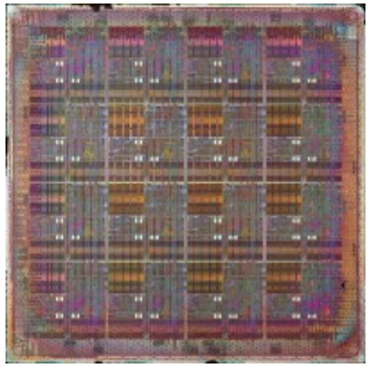
Christopher Batten[1], Ronald G. Dreslinski[2],
Rajesh K. Gupta[3], Michael B. Taylor[4], Zhiru Zhang[1]

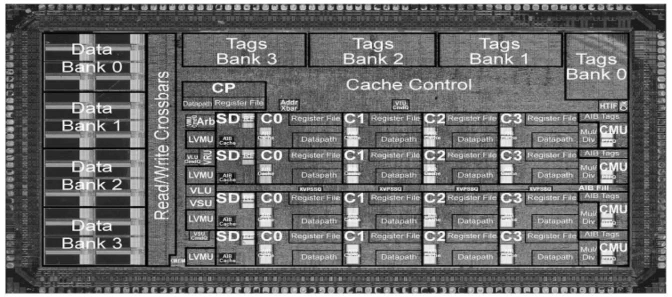
[1] Cornell University
[2] University of Michigan
[3] University of California, San Diego
[4] Bespoke Silicon Group, (U. Washington/ UC San Diego)

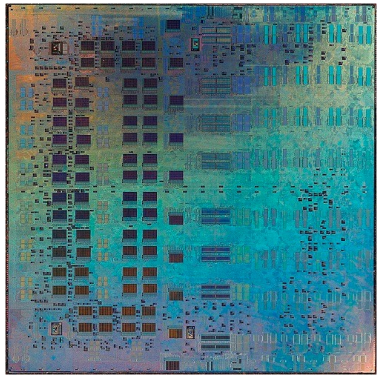
MICRO-50

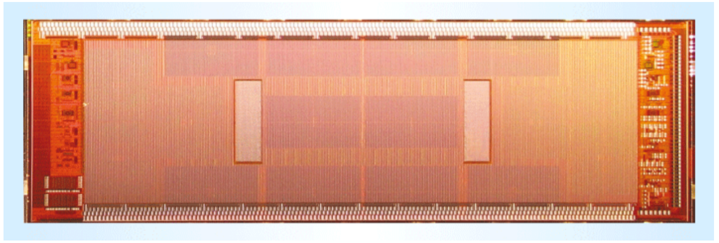
October 14, 2017

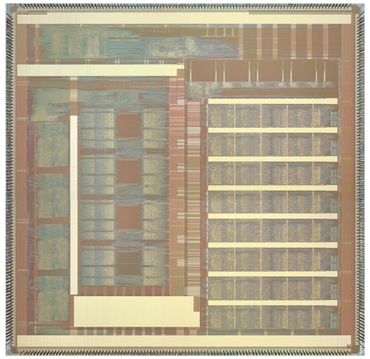
Computer Architecture Research Prototyping

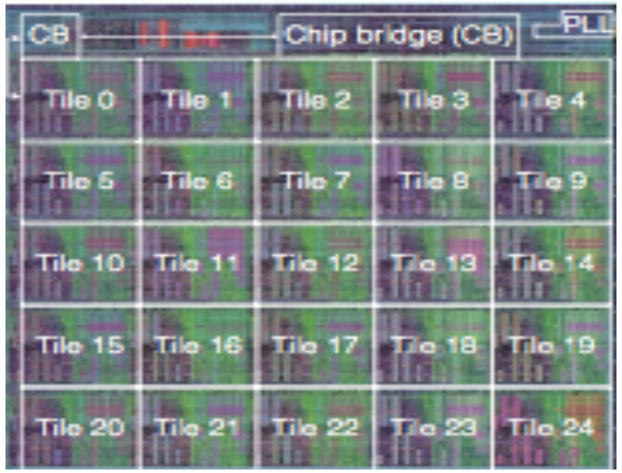












Prototyping is important to *complement* the results of simulation-based research

Many benefits to prototyping :

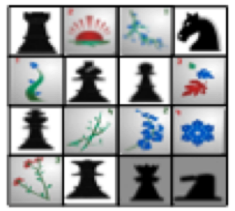
- Validating assumptions
- Validating design methodologies
- Measuring real system-level performance and energy efficiency
- Creating platforms for software research
- Building credibility with industry
- Building intuition for physical design
- Pedagogical benefits
- Building real things is fun!

The Continuing Need for Building Prototypes

The Four Horsemen of the Coming Dark Silicon Apocalypse

"Shrink"

"Dim"



"Specialize"

"Magic"

The rise of the dark silicon era [1], in which an increasing fraction of silicon must remain unpowered, is motivating an increasing trend towards accelerator-centric architectures.

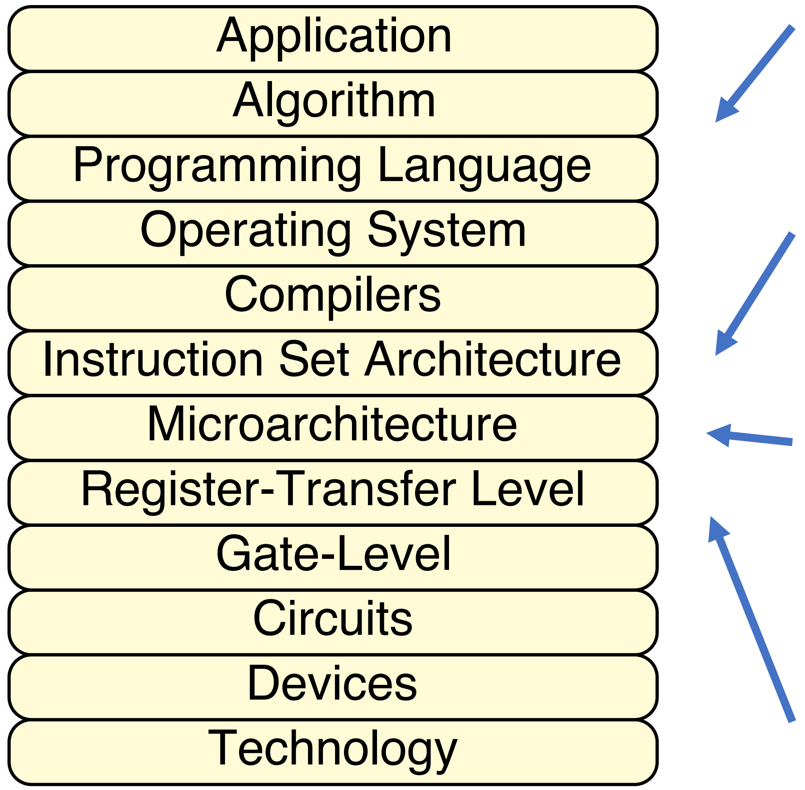
Specialization research requires:

- New ***simulation-based*** evaluation methodologies based on accelerators [2]
- New ***prototyping*** methodologies for rapidly building accelerator-centric prototypes

Unfortunately, building research prototypes can be tremendously challenging.

[1] M. Taylor. "**Is Dark Silicon Useful? Harnessing the Four Horsemen of the Coming Dark Silicon Apocalypse**," In Design Automation Conference, 2012.
[2] Y. Shao, et al. "**Aladdin: A Pre-RTL, Power-Performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures**", ISCA 2014

Prototyping with the RISC-V Software/Hardware Ecosystem



Software Toolchain

- A complete, off-the-shelf software stack (e.g., binutils, GCC, newlib/glibc, Linux kernel & distros) for both embedded and general-purpose

Architecture

- RISC-V ISA specification designed to be both modular and extensible, with a small base ISA and optional extensions

Microarchitecture

- On-chip network specifications and implementations (NASTI, TileLink)
- RISC-V processor implementations for both in-order (Berkeley Rocket) and out-of-order (Berkeley BOOM) cores

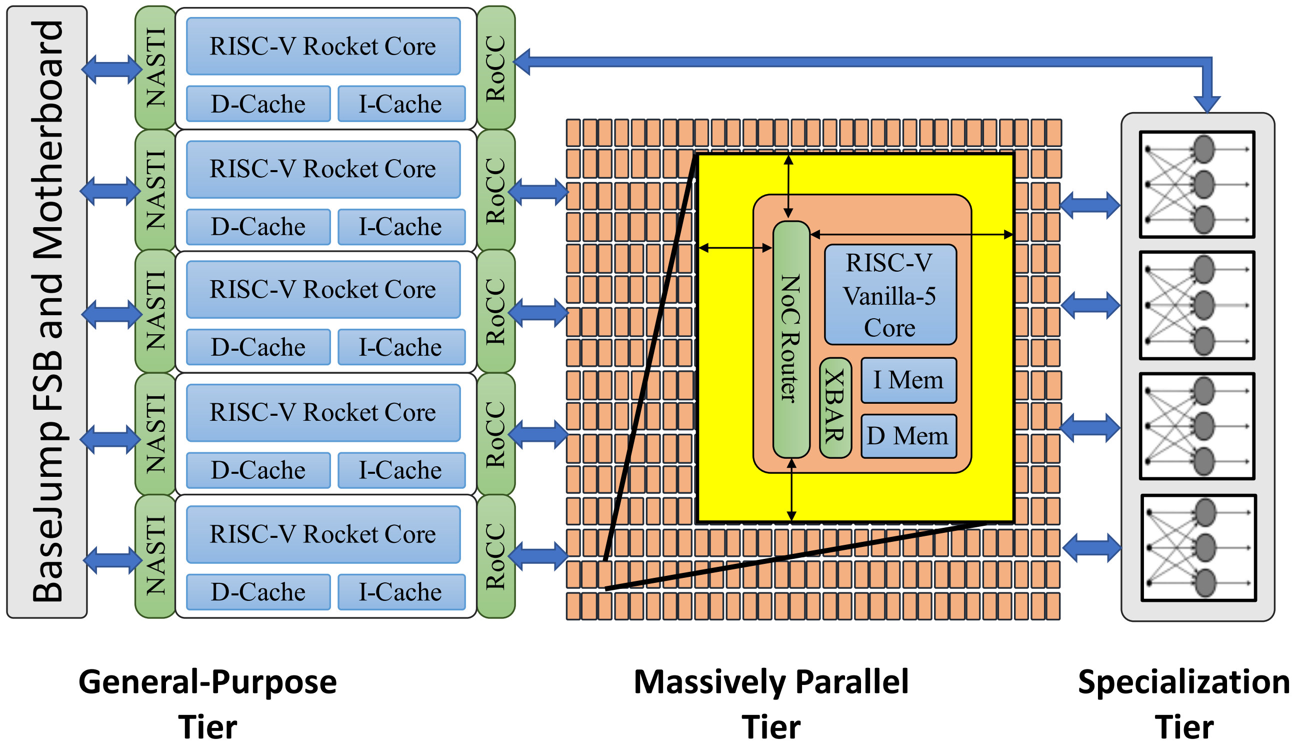
Physical Design

- Previous spins of chips for reference

Testing

- Standard core verification test suites + Turn-key FPGA gateware

The Celerity System-on-Chip



Celerity, an accelerator-centric SoC with a tiered accelerator fabric that targets highly performant and energy-efficient embedded systems

Funded by the DARPA CRAFT program, *"Circuit Realization At Faster Timescales"*

The goal was to develop new methodologies to design chips more quickly

We leveraged the *RISC-V software/hardware ecosystem* as we built Celerity, and we believe it was instrumental in enabling a team of **20 graduate students** to tape out a complex SoC in **only 9 months**

Celerity: Chip Overview

http://www.opencelerity.org

- TSMC 16nm FFC
- 25 mm^2 die area (5mm x 5mm)
- ~385 million transistors
- 511 RISC-V cores
  - 5 Linux-capable RV64G Berkeley Rocket cores
  - 496-core RV32IM mesh tiled array “manycore”
  - 10-core RV32IM mesh tiled array (low voltage)
- 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

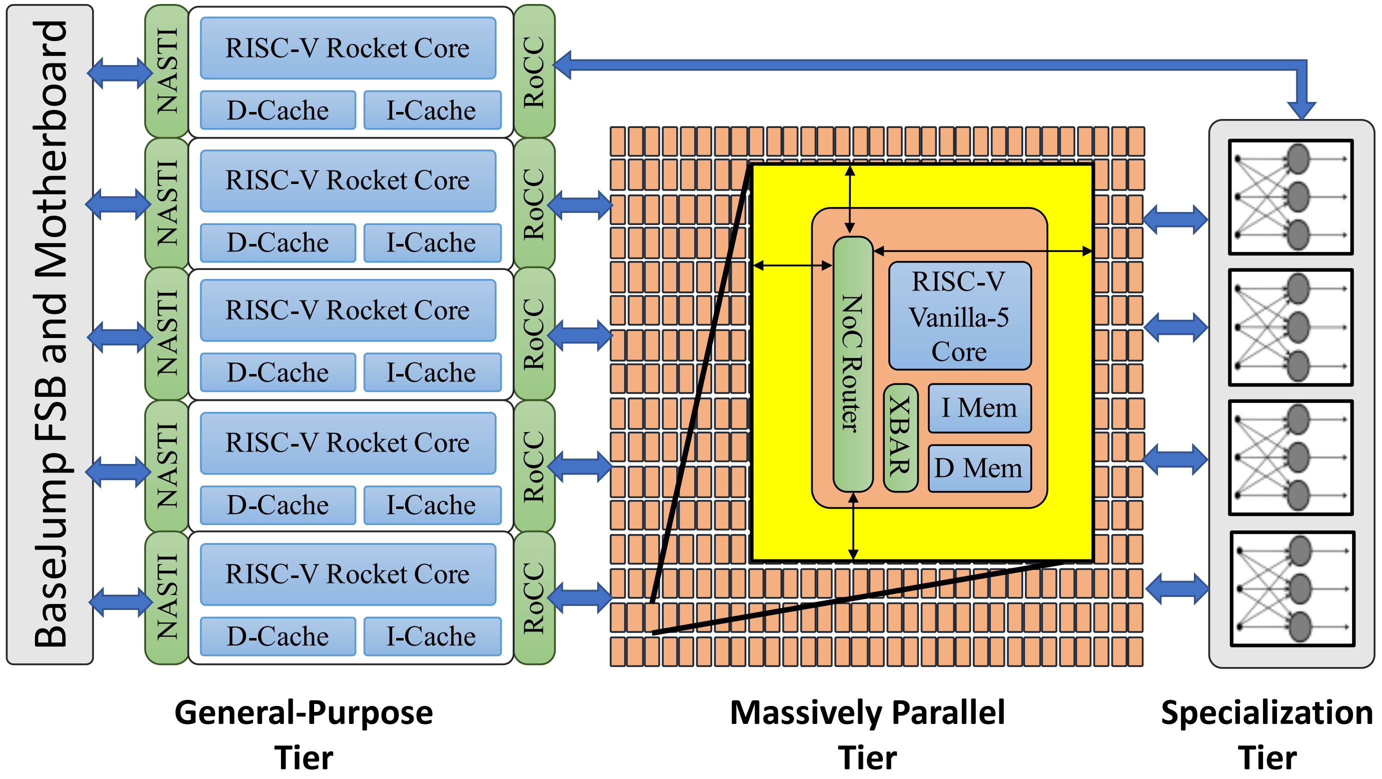
Cornell University

UNIVERSITY OF MICHIGAN

UC San Diego

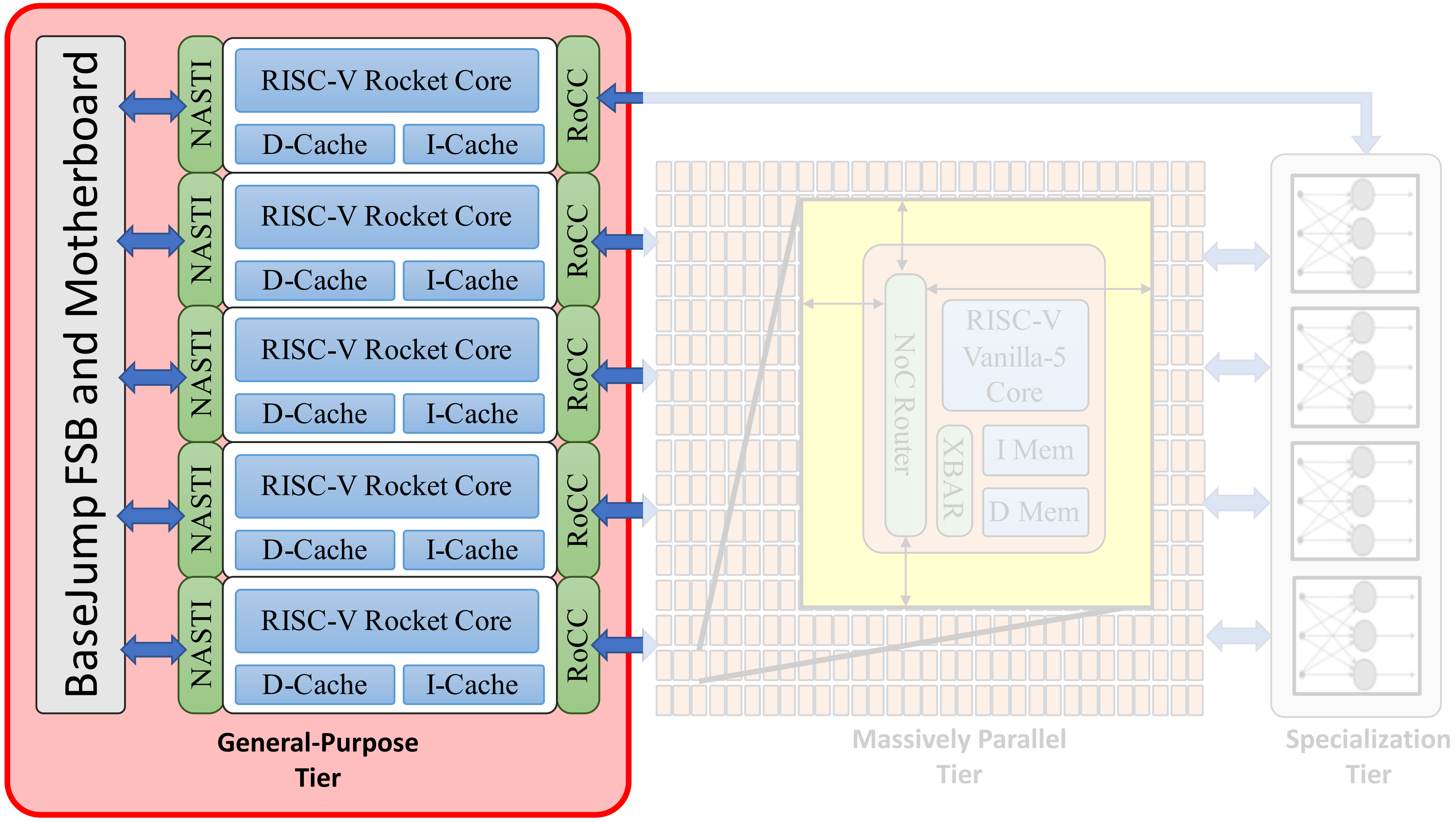
UNIVERSITY of WASHINGTON

Agenda



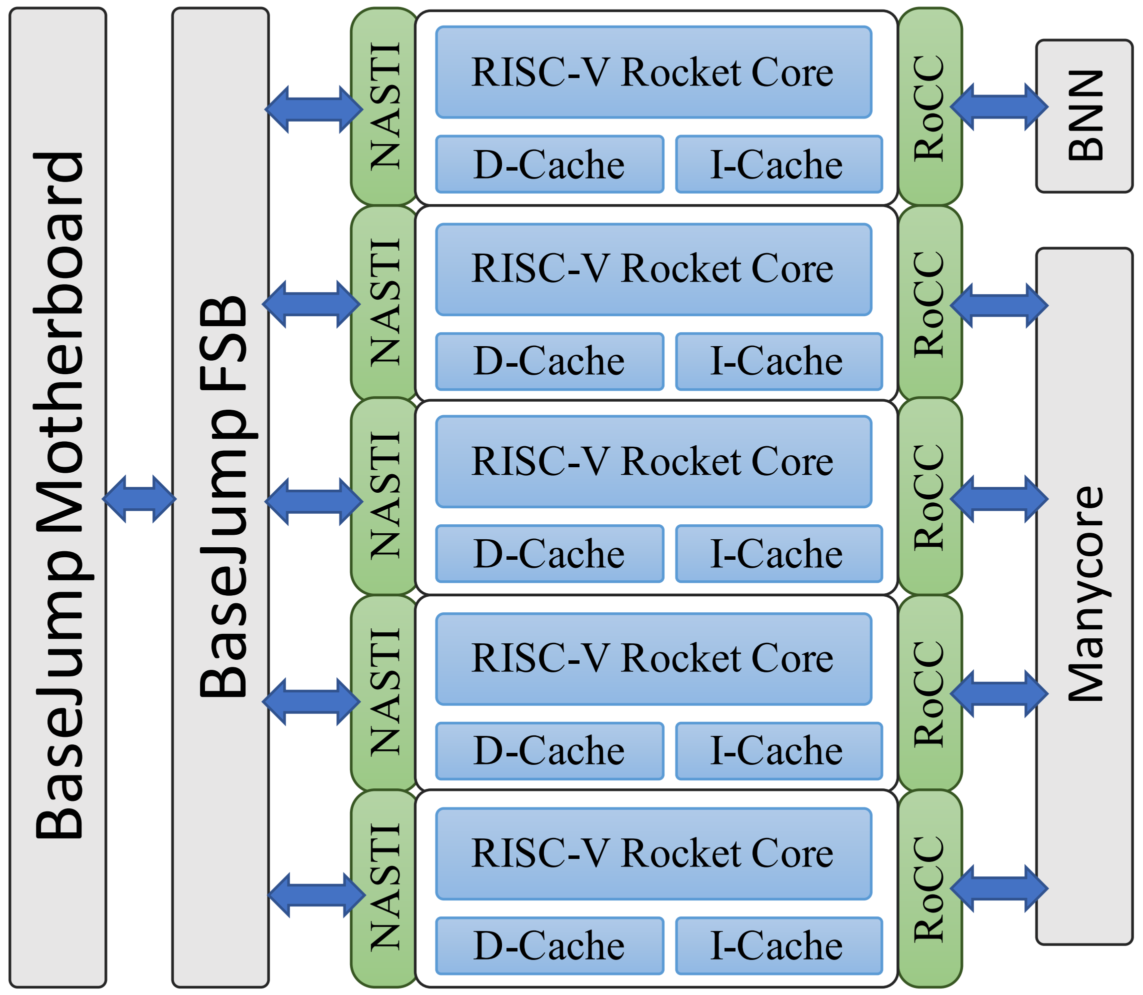
- Introduction
- For each Tier:
  - What did we build?
  - How did we build it?
  - RISC-V Ecosystem Successes
  - RISC-V Ecosystem Challenges
- Conclusion

Celerity: General-Purpose Tier



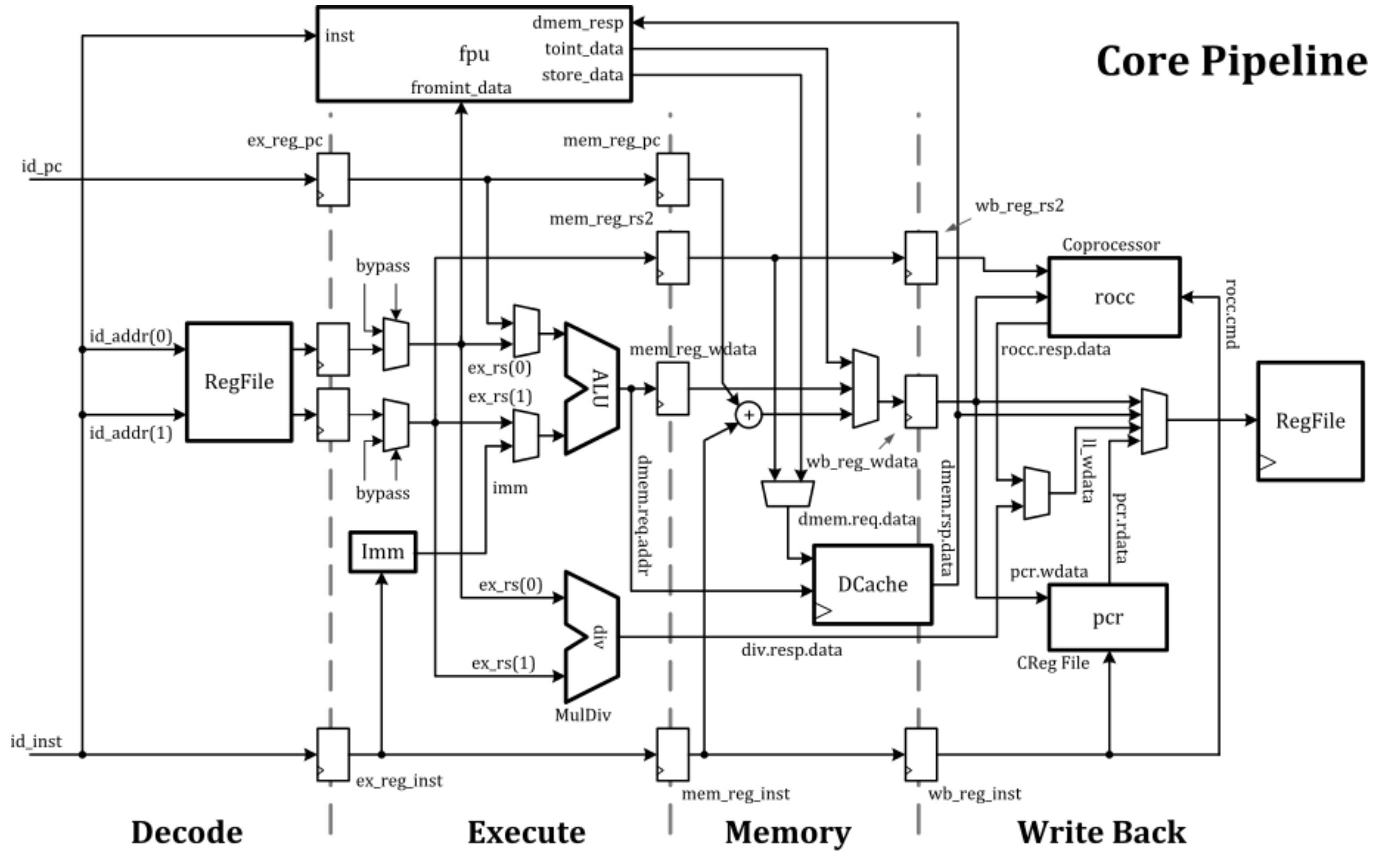
General-Purpose Tier Overview

- 5 Berkeley Rocket Cores (RV64G)
- Workload
  - General-purpose compute
  - Operating system (e.g. Linux & TCP/IP Stack)
  - Interrupt and Exception handling
  - Program dispatch and control flow
- Interface
  - Interface to off-chip I/O and other peripherals
  - 4 Cores connect to the manycore array
  - 1 Core interfaces with the BNN
- Memory
  - Each core executes independently within its own address space
  - Memory management for all tiers



Berkeley Rocket Cores

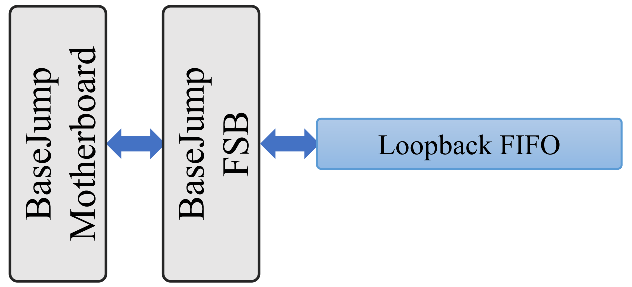
- 5 Berkeley Rocket Cores (https://github.com/freechipsproject/rocket-chip)
  - Generated from Chisel
  - RV64G ISA
  - 5-stage, in-order, scalar processor
  - Double-precision floating point
  - I-Cache: 16KB 4-way assoc.
  - D-Cache: 16KB 4-way assoc.
- Physical Implementation
  - 625 MHz (Critical path in FSB)
  - 0.19 mm^2 per core



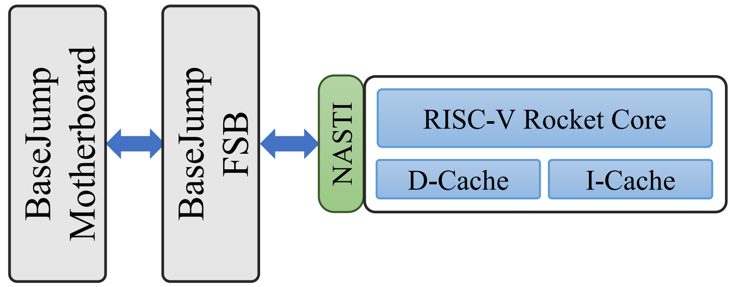
http://www.lowrisc.org/docs/tagged-memory-v0.1/rocket-core/

Design Iterations

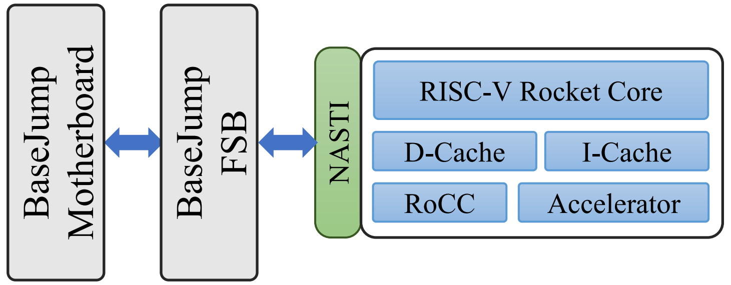
1. Loopback
Baseline design to validate FSB and Northbridge



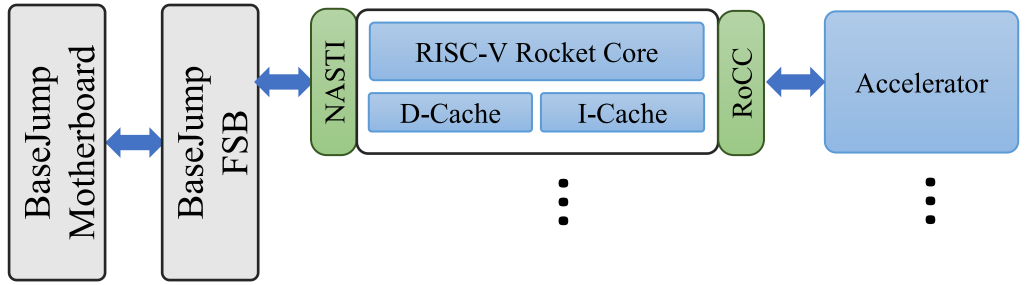
2. Alpaca
Implemented NASTI bridge and connected rocket core



3. Bison
Implemented accelerator connected through Blackboxed RoCC

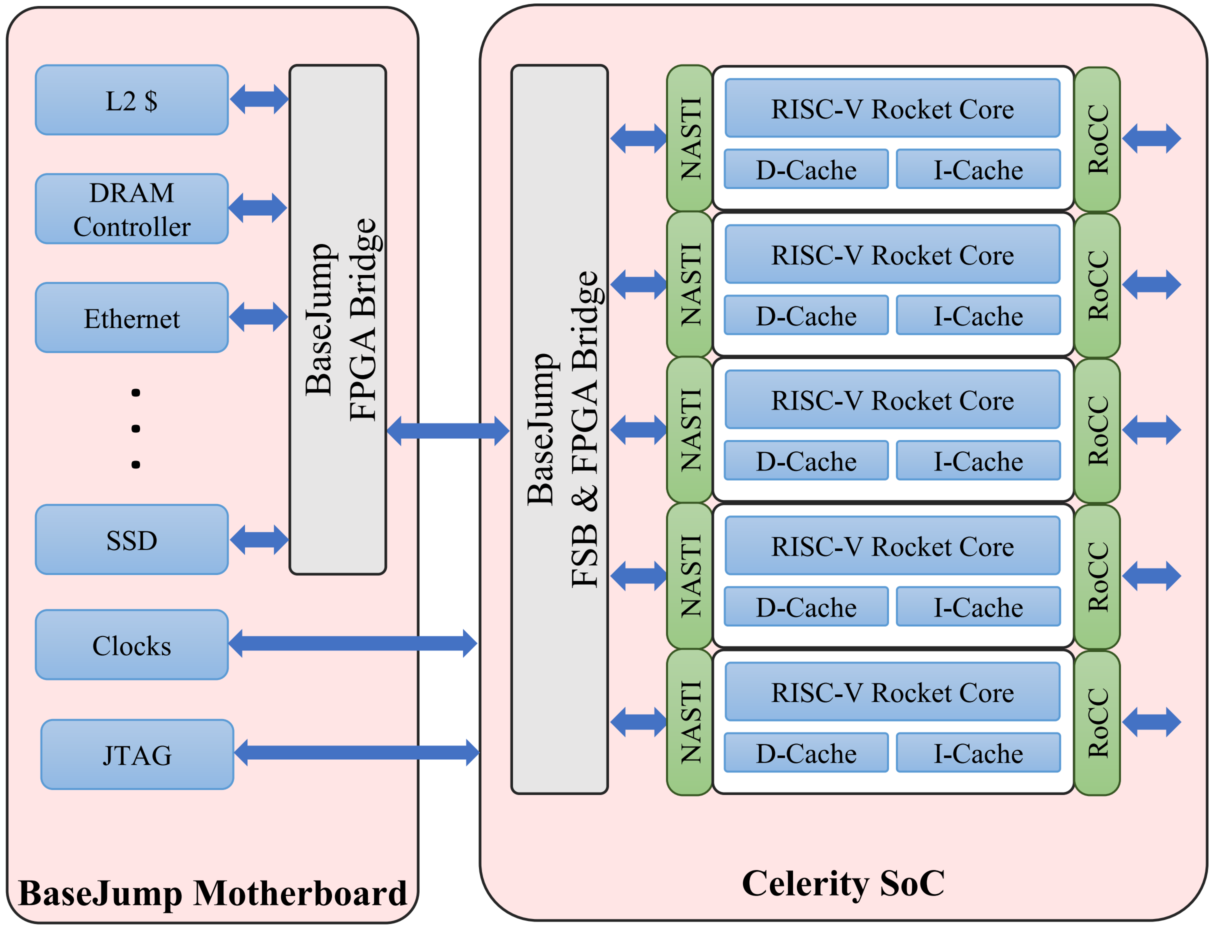


4. Coyote
Modularized RoCC interface to accelerator



Off-Chip Interface and Northbridge

- Open-source BaseJump IP Library
  - http://bjump.org
- Front Side bus
  - BaseJump Communication Link
  - High Speed (DDR) Source-Synchronous Communication Interface
- Packaging
  - Modified BaseJump BGA Package and I/O Ring
- Validation
  - BaseJump Super Trouble PCB (Daughter Card)
  - BaseJump Motherboard (ZedBoard)



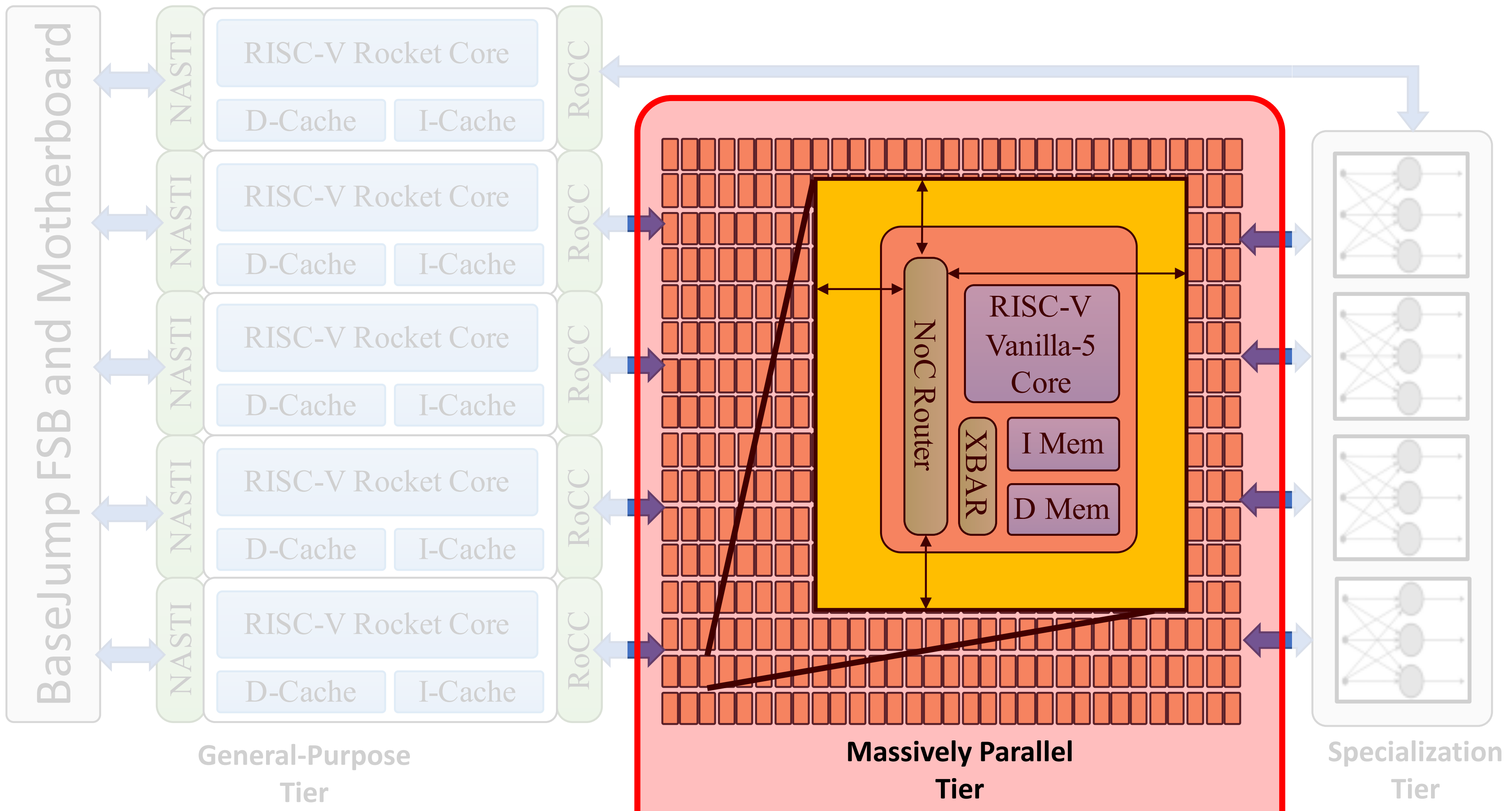
RISC-V Successes

- Berkeley Rocket Cores
  - Very quickly generated validated designs
  - Vibrant ecosystem to provide feedback and support
  - Test and Validation infrastructure
  - Software and Toolchain support
- Flexible memory system and peripheral I/O support
  - Easy integration with BaseJump IP Library
- Balances extensibility and software support

RISC-V Lessons Learned

- Component stability, compatibility and versioning
- Chisel adoption
- RTL simulationissues
  - Deciphering Chisel generated RTL
  - Register initialization and X-Pessimism

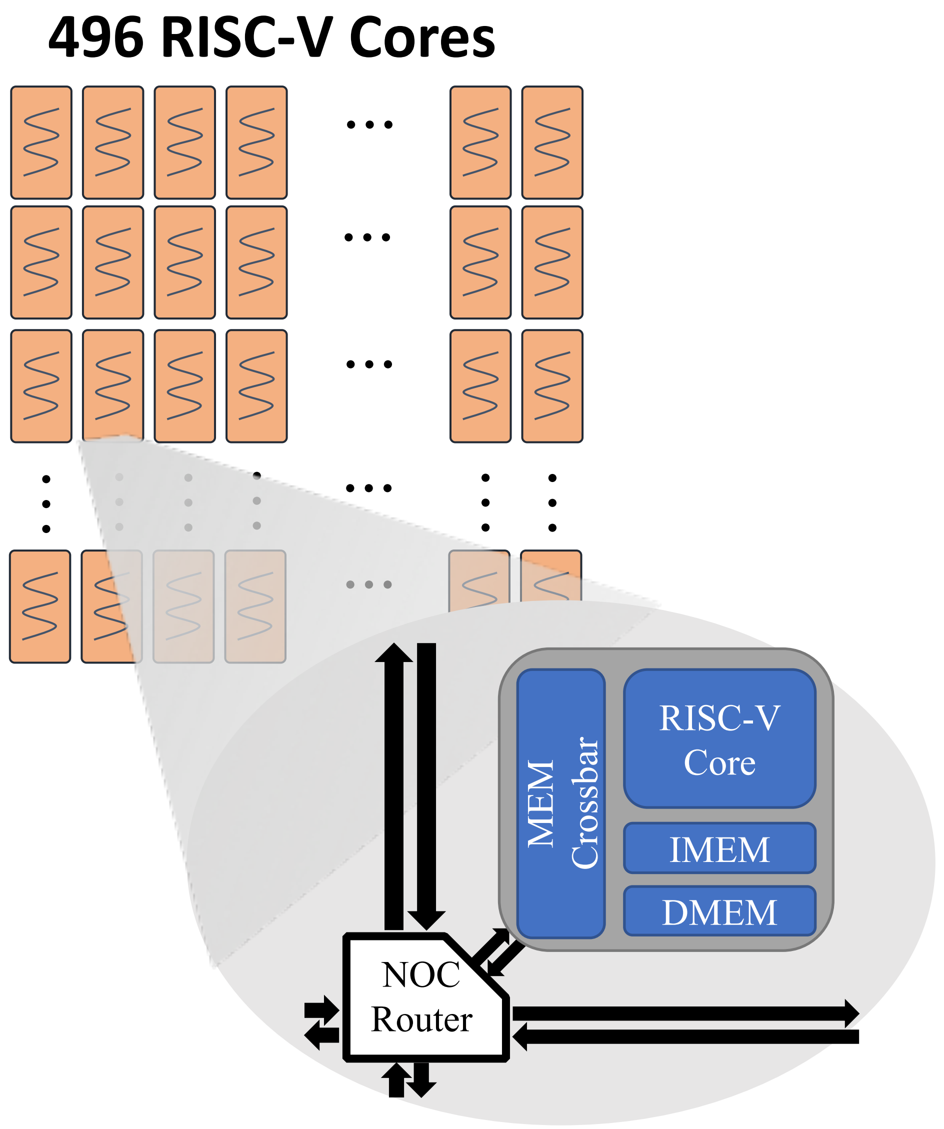
Celerity: Massively Parallel Tier



http://bjump.org/manycore

Developed by Taylor's Bespoke Silicon Group @ UW

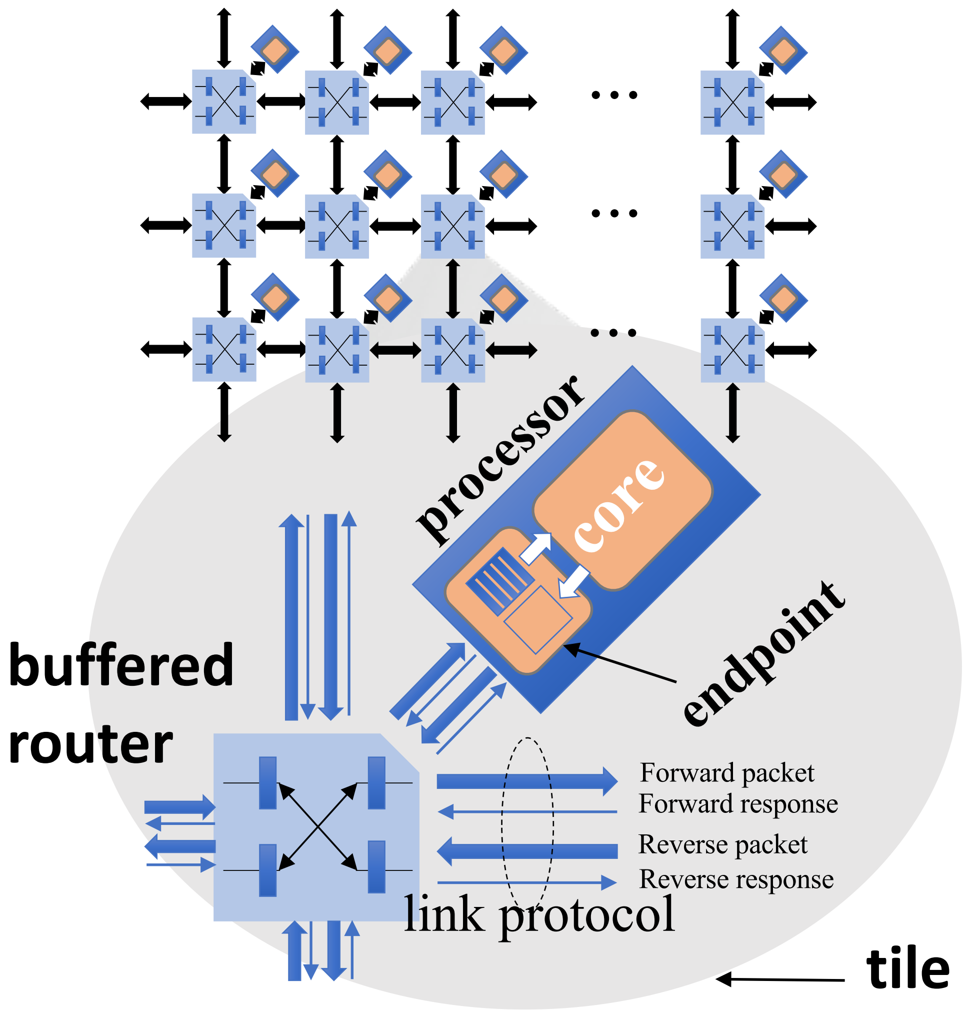
The tiled architecture



The Vanilla core: Simple but efficient to run C code without any toolchain modification

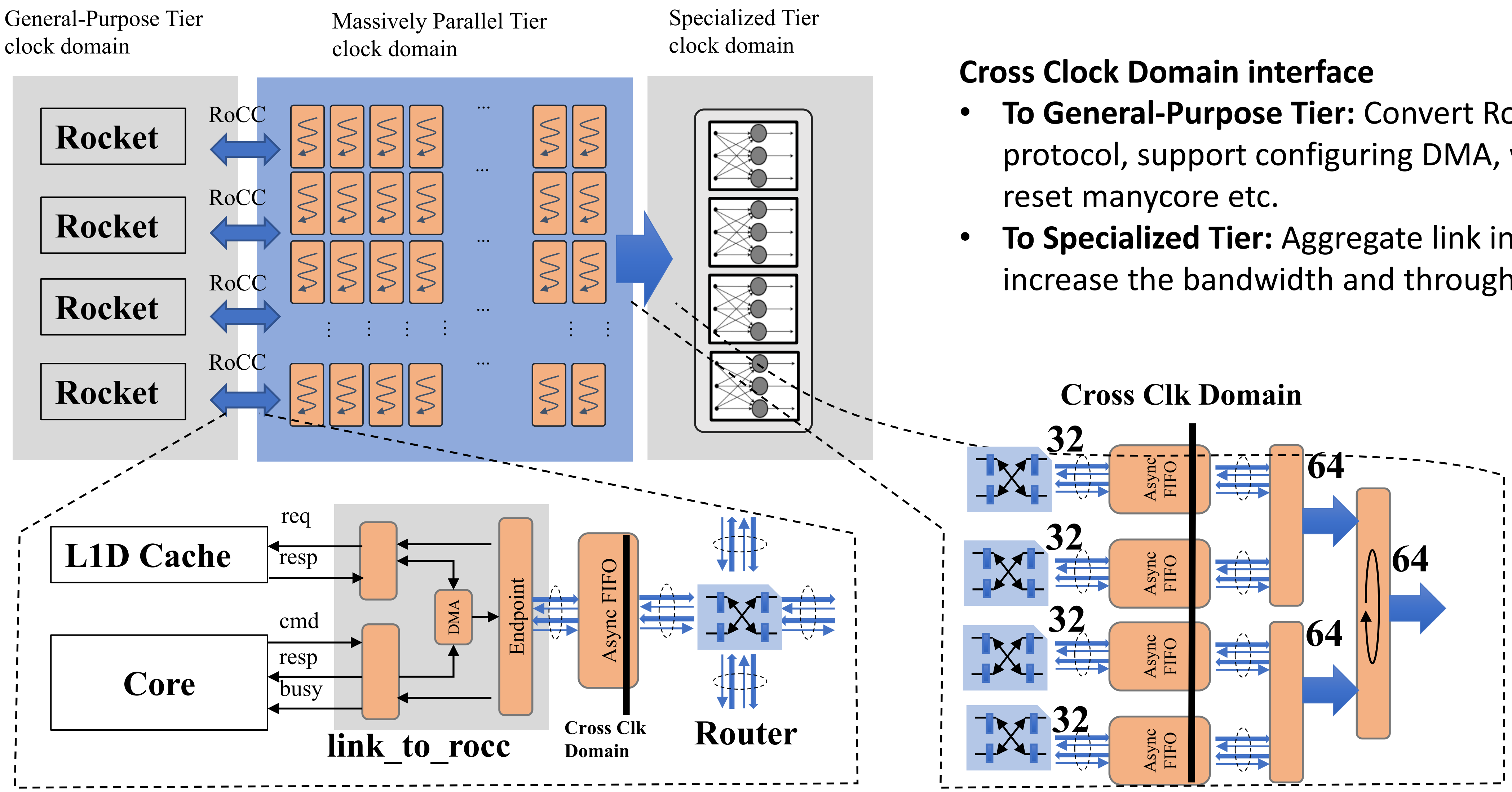
- ISA: RV32IM
- Pipeline: 5-stage, fully forwarded, in-order, single issue
- Scratchpad memory: 4KB for I Mem, 4KB for D Mem
- Second Tape-out of this tiled architecture (10-core)

Mesh Network



- **Link Protocol:** Forward/Reverse paths, parameterizable address/data bits
- **Credit-Based:** Each packet will be acknowledged with response
- **Flow control**: Endpoint controls the number of the outstanding packet.
- **Router:** simple XY-dimension routing, buffered

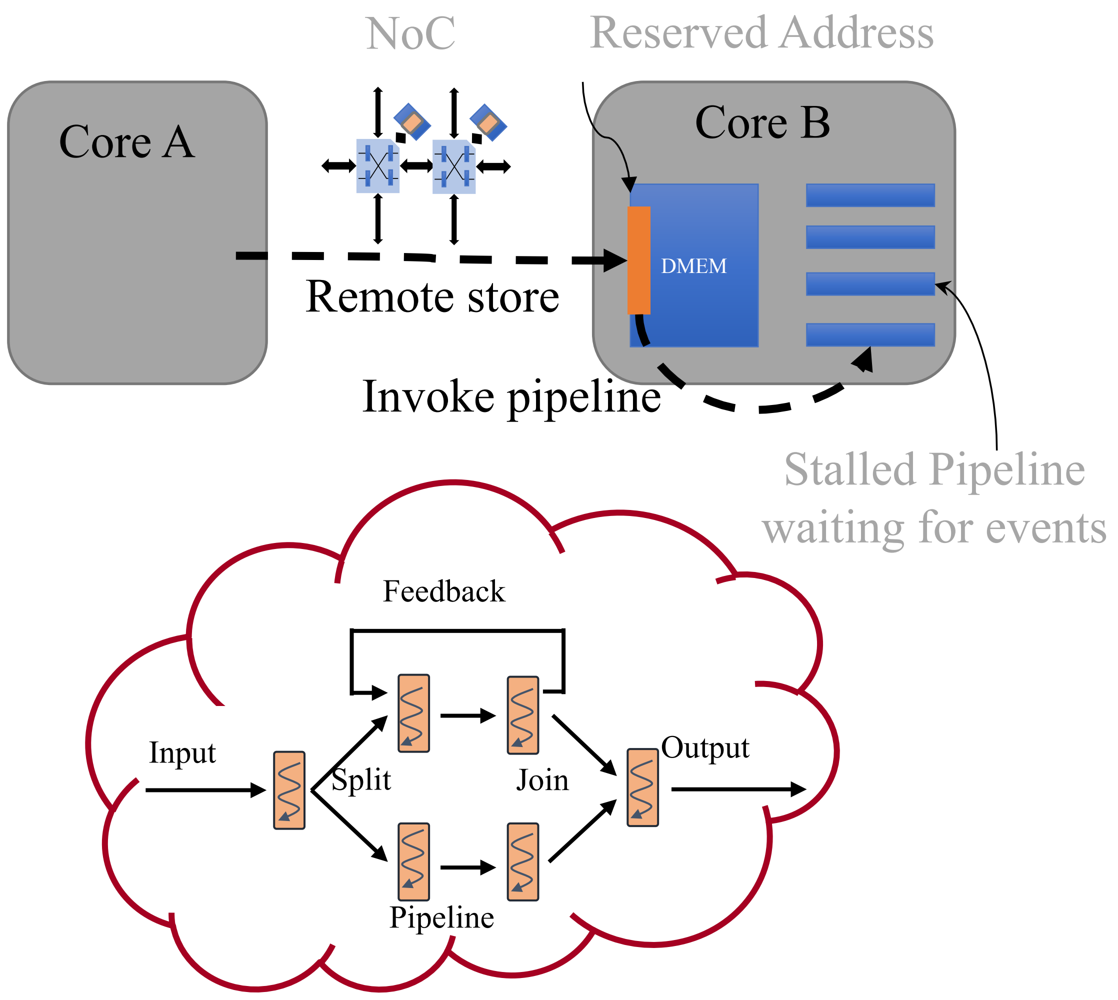
Manycore Links to General-Purpose and Specialized Tier



Cross Clock Domain interface

- **To General-Purpose Tier:** Convert RoCC to link protocol, support configuring DMA, write and reset manycore etc.
- **To Specialized Tier:** Aggregate link interface to increase the bandwidth and throughput

Programming Model



Producer-consumer Programming

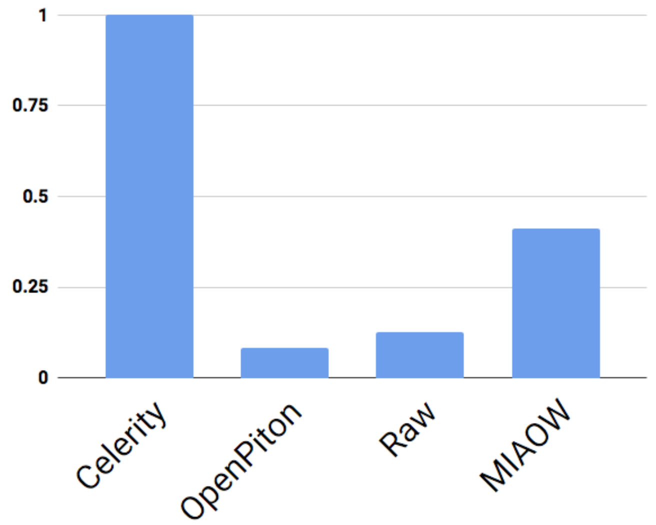
Producer-consumer programming model:

extended instructions for efficient inter-tile synchronization

- **Load Reserved (lr.w):** load value and set the reservation address
- **Load-on-broken-reservation (lr.lbr):** stall if the reserved address didn't written by other cores
- **Consumer:** wait on <address, value>
- **Benefits:** No polling, no interrupt, fast response, stalled pipeline can save power

Thread Density Comparison

- **Timing:** 1.05 GHz @ 16 nm
- **Area:** 0.024 mm^2 @ 16 nm
- **Si Utilization Ratio:** 90%



Normalized Physical Threads (ALUops) per Area

	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 = 16KB L1.5/L2 Cache = 72KB	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

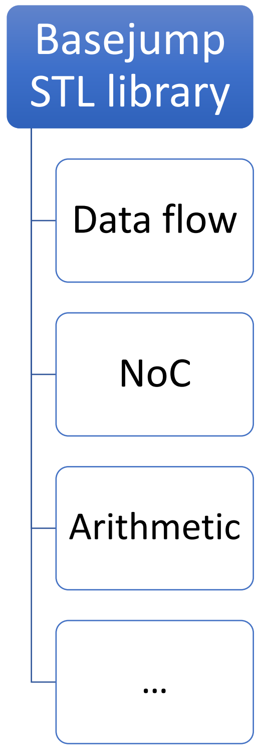
[1] J. Balkind, et al. "OpenPiton : An Open Source Manycore Research Framework," in *the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016.
[2] R. Balasubramanian, et al. "Enabling GPGPU Low-Level Hardware Explorations with MIAOW: An Open-Source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

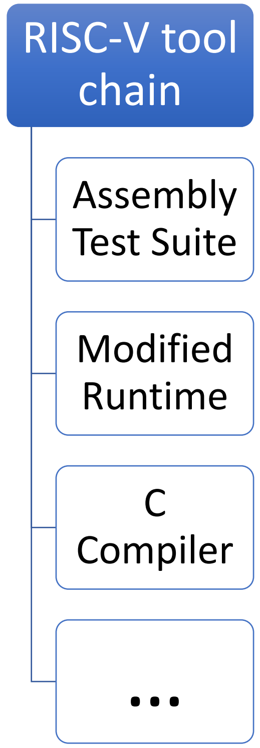
How did we build the massively parallel tier?

Design

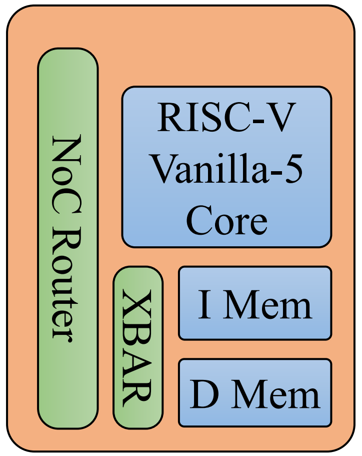
Testing

Hierarchical Flow

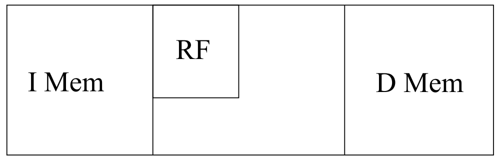




In house



One tile



Hard-macro

Floorplan

RISC-V Ecosystem Successes

- Modular ISA
  - Flexible for both complex cores (i.e. Rocket) and simple cores (i.e. Vanilla)
- Extensible RoCC interface
  - 4 customizable instructions: we used one
- Comprehensive assembly test suite(434 test cases)
- Off-the-shelf toolchain

Building up the RISC-V Ecosystem

We provide an efficient RV-32IM implementation in System Verilog.

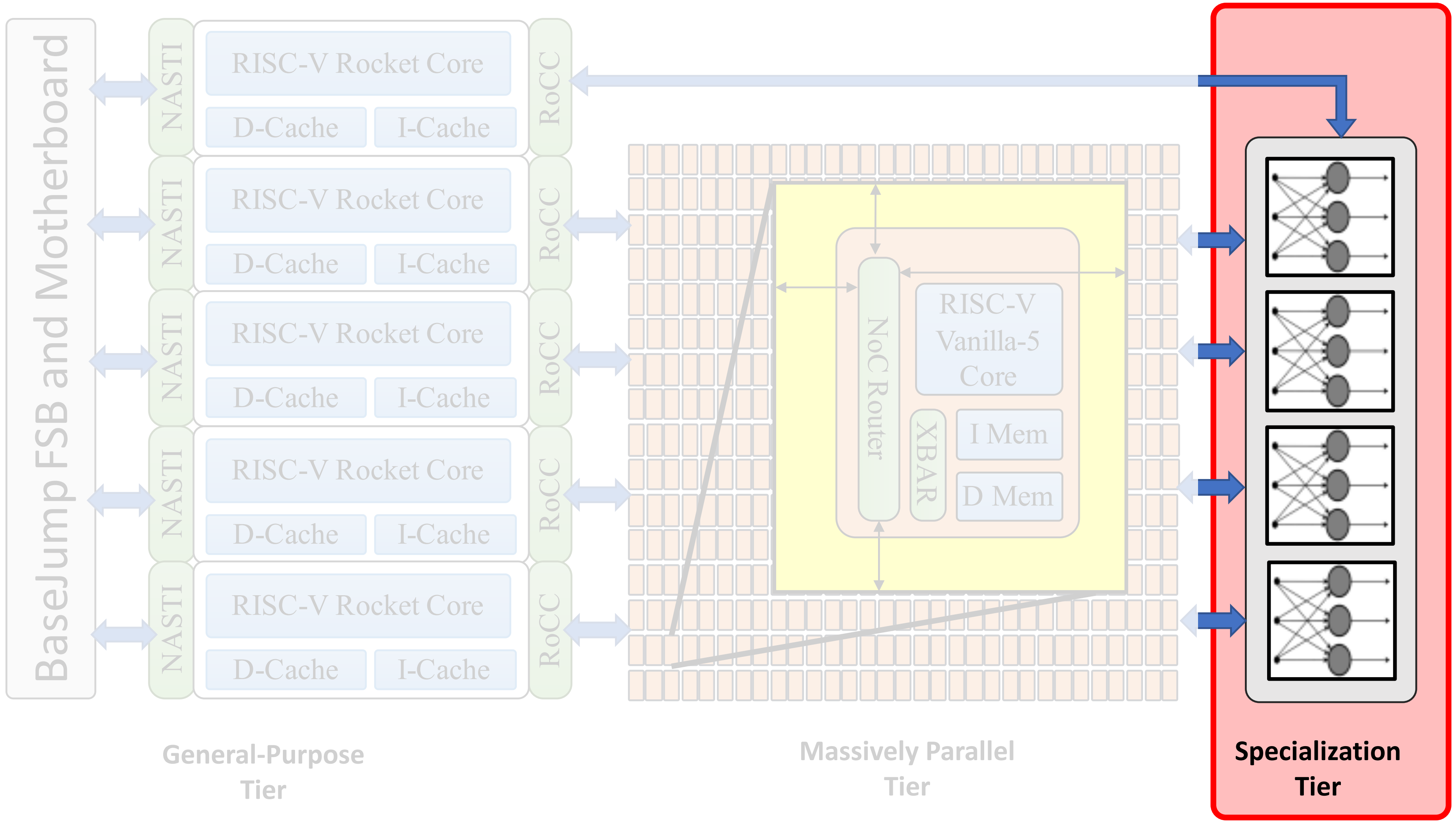
We consolidated Information about RoCC that was scattered across the internet.

With Celerity

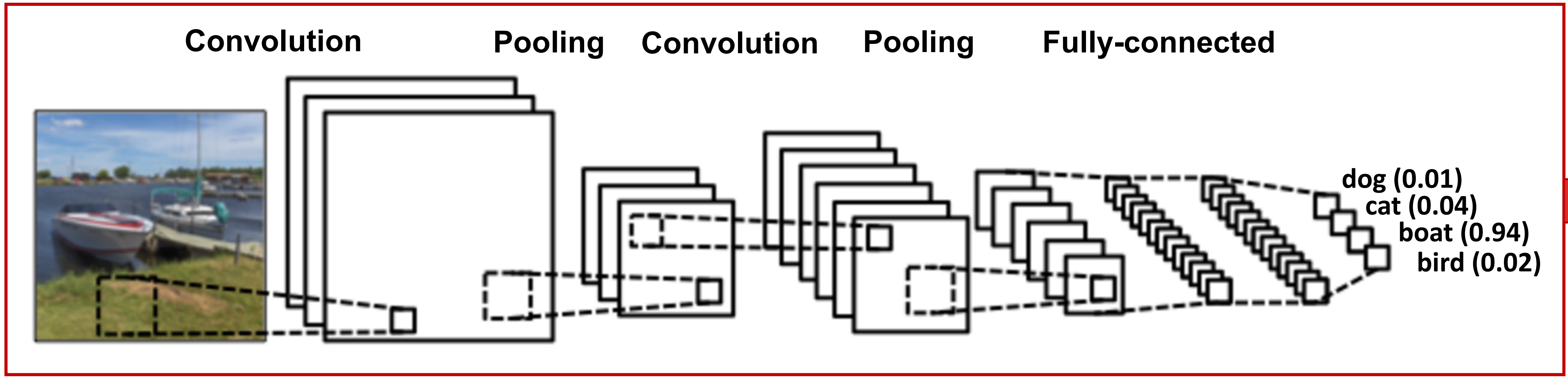
- Efficient open source core
- Based on Systemverilog
- Silicon proven

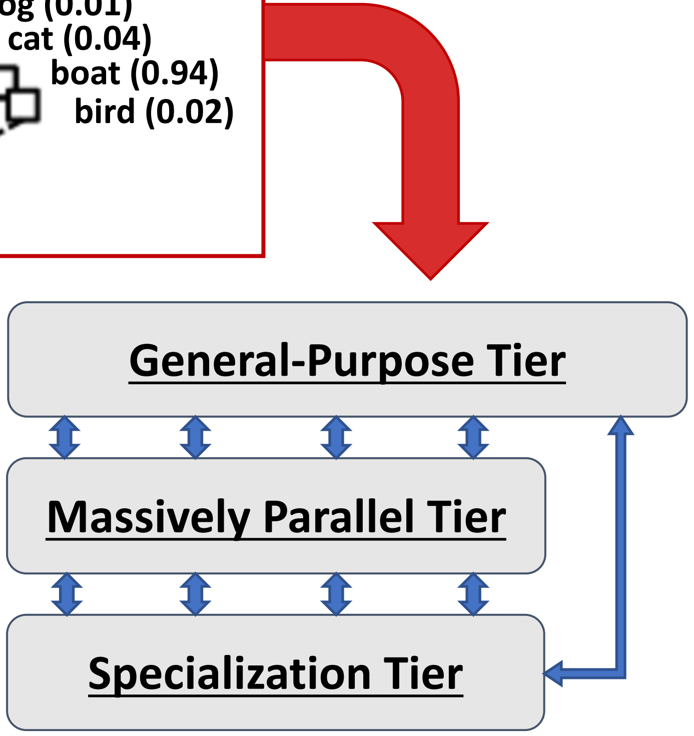
- Public RoCC document V.2 [bjump.org/rocc_doc]
- Exported RoCC interface on top level

Celerity: Specialization Tier



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





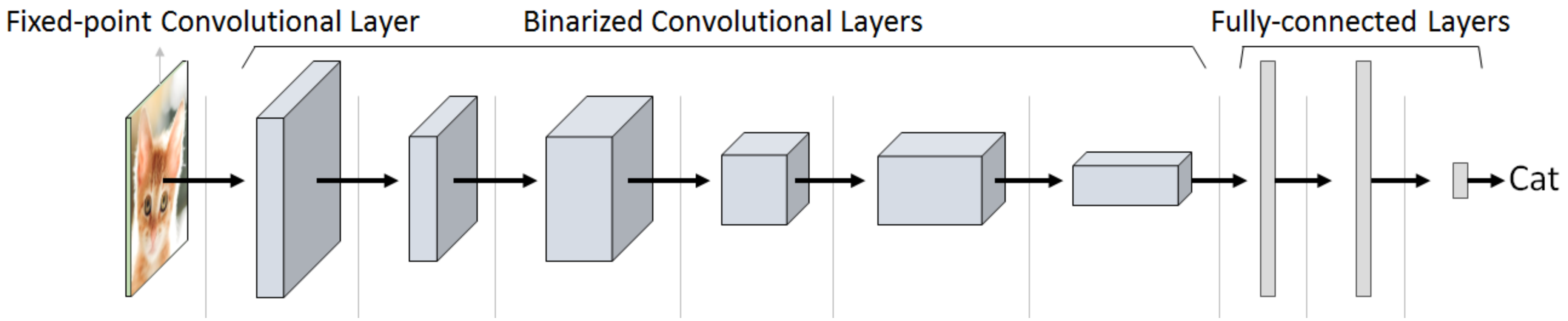
Three steps to map applications to tiered accelerator fabric:

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

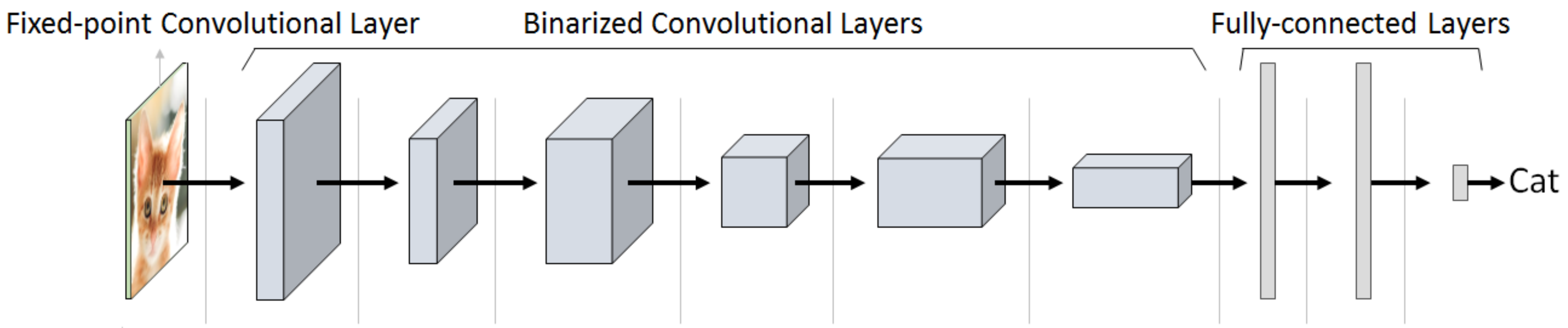
Step 1: Algorithm to Application
Binarized Neural Networks


Fixed-point Convolutional Layer
Binarized Convolutional Layers
Fully-connected Layers
Cat

- 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).

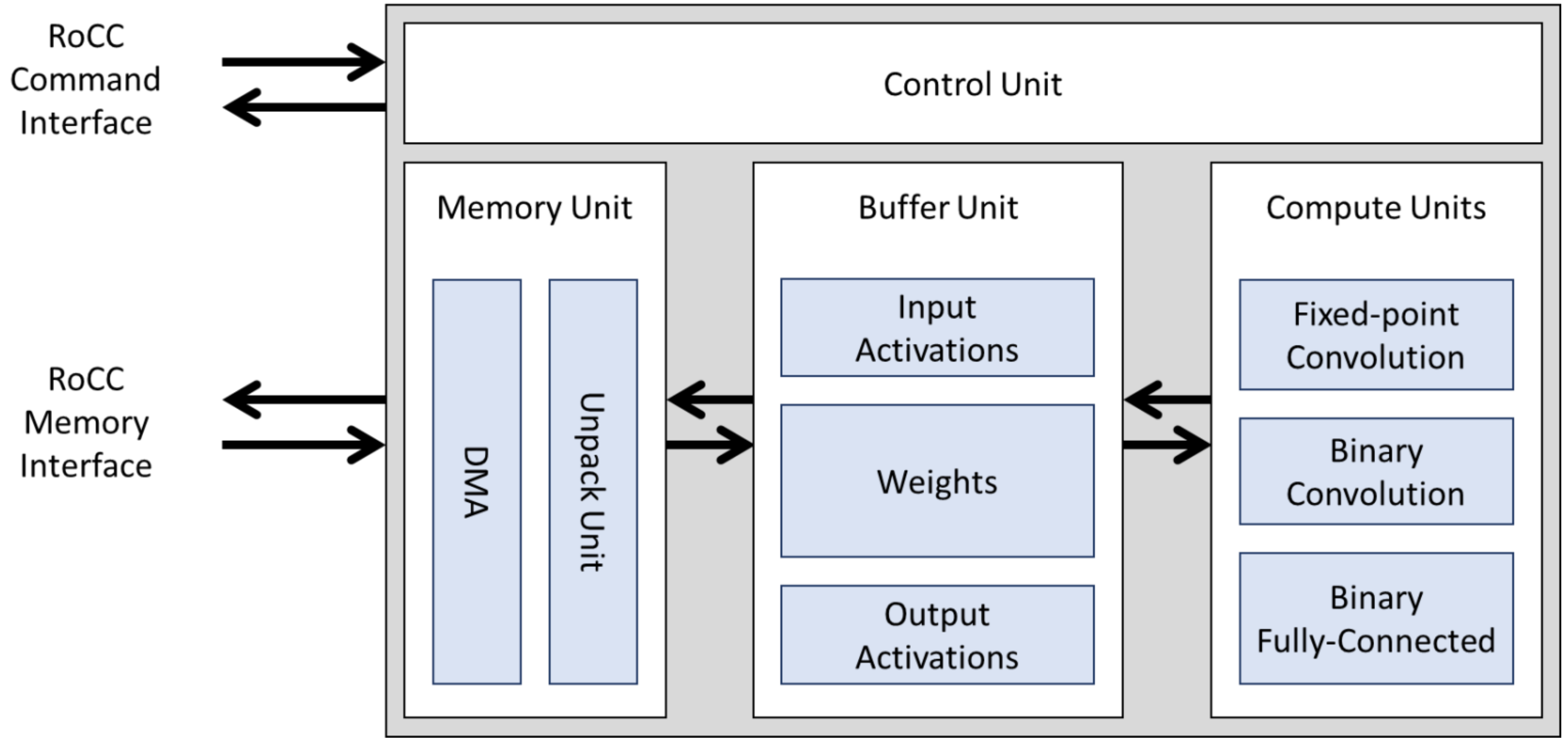
Step 1: Algorithm to Application
Characterizing BNN Execution


Fixed-point Convolutional Layer
Binarized Convolutional Layers
Fully-connected Layers
Cat

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

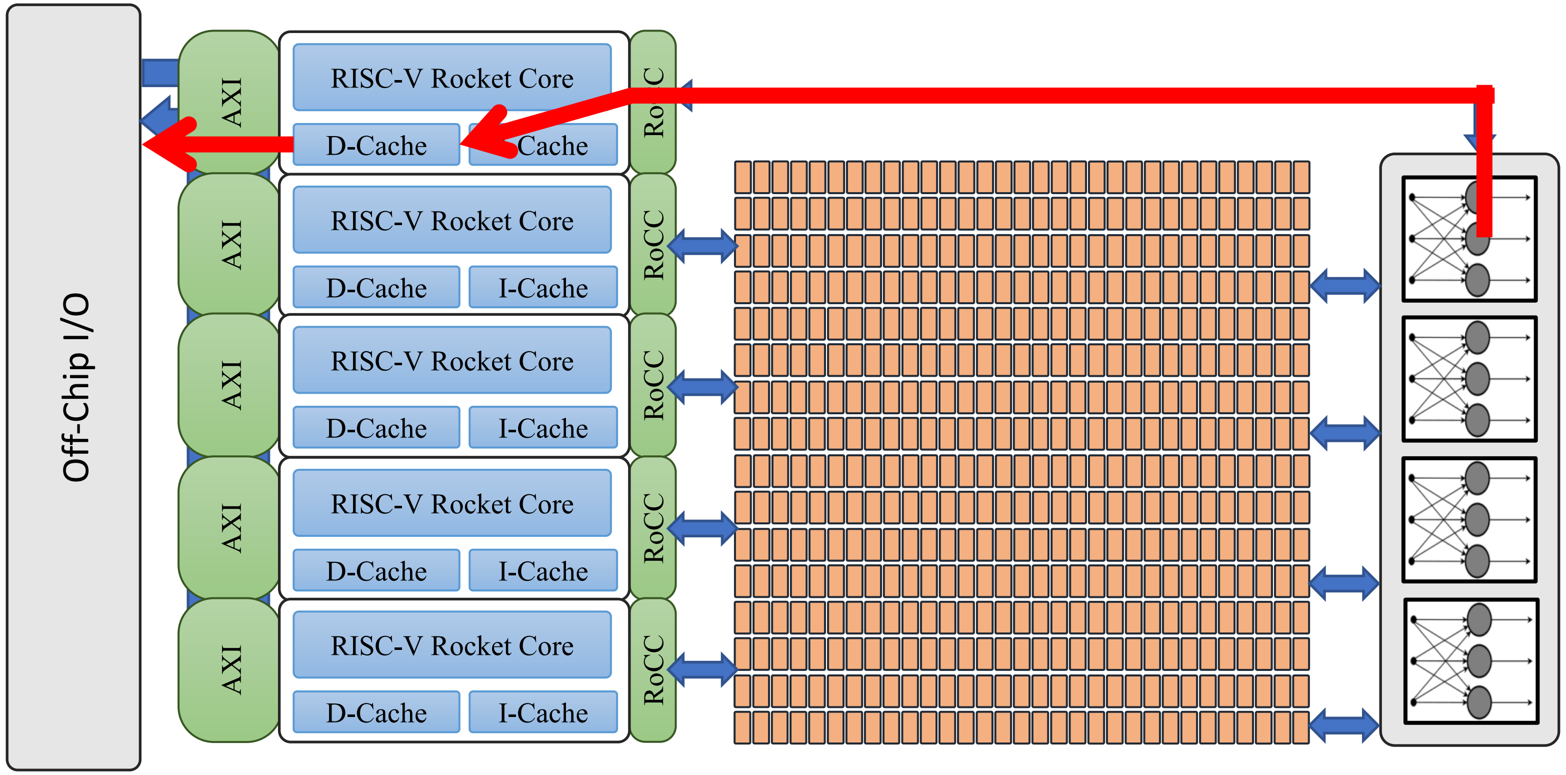
- Using just the general-purpose tier would be 200x slower than the performance target (60 classifications / sec)
- Binarized convolutional layers consume over 97% of dynamic instruction count
- Perfect acceleration of just the binarized convolutional layers is still 5x slower than performance target
- 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



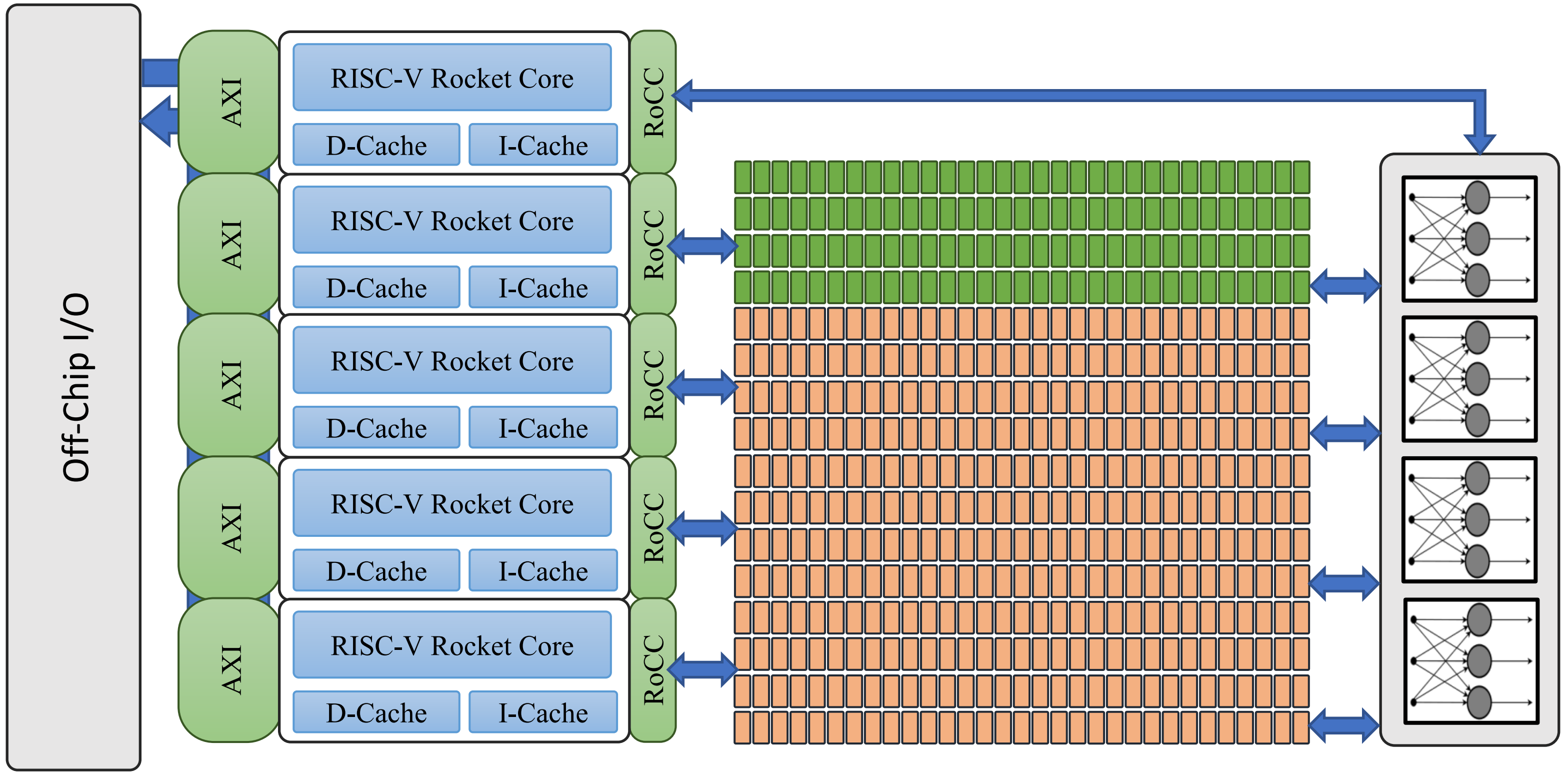
1. Accelerator is configured to process a layer through RoCC command messages
2. Memory Unit starts streaming the weights into the accelerator and unpacking the binarized weights into appropriate buffers
3. Binary convolution compute unit processes input activations and weights to produce output activations

Step 2: Application to Accelerator
General-Purpose Tier for Weight Storage



- The BNN specialized accelerator can use one of the Rocket cores' caches to load every layer's weights

Step 3: Assisting Accelerators
Massively Parallel Tier for Weight Storage



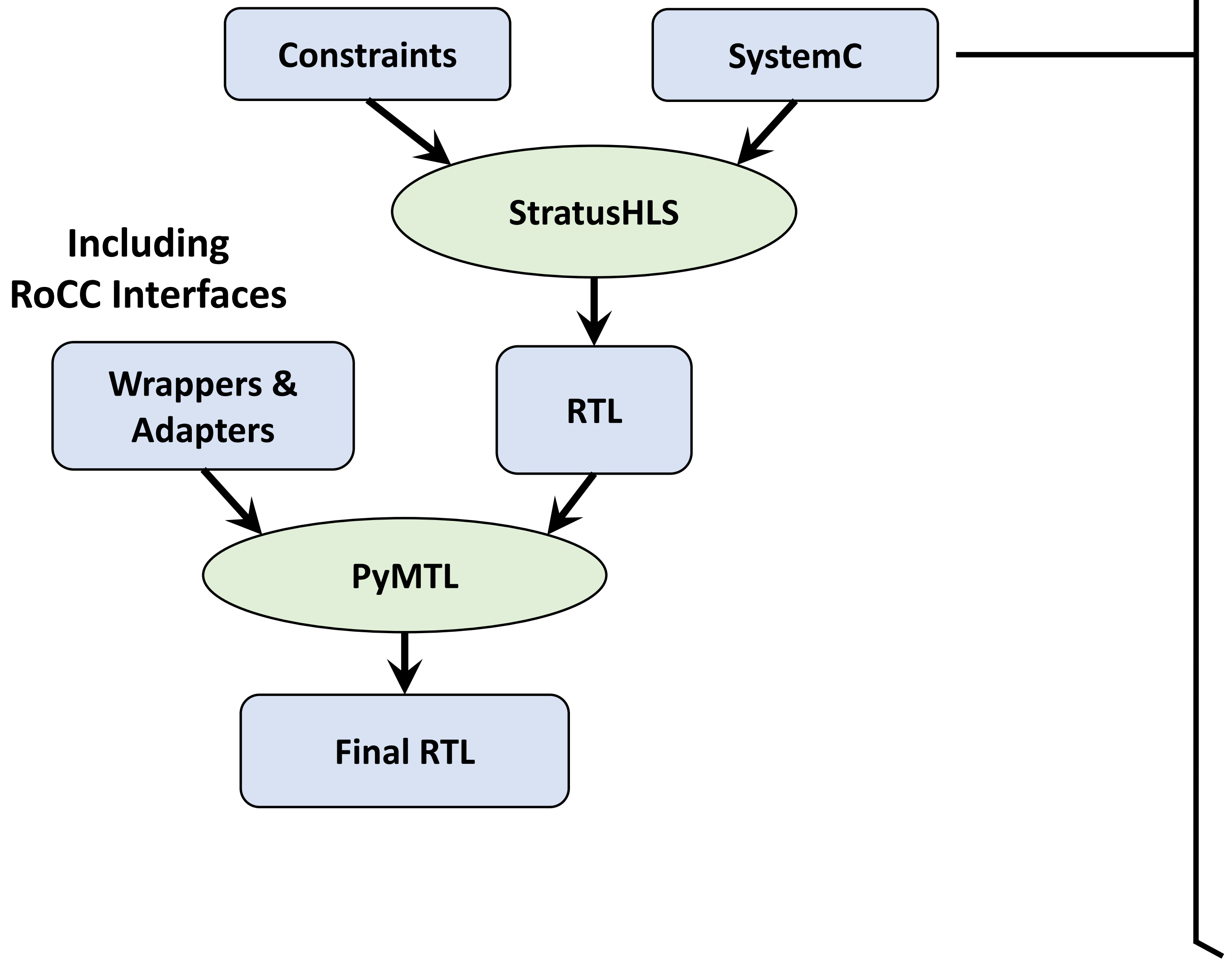
- The BNN specialized accelerator can use one of the Rocket cores' caches to load every layer's weights
- Each core in the massively parallel tier executes a remote-load-store program to orchestrate sending weights to the specialization tier via a hardware FIFO

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	20	3.3
Power (Watts)	0.2 – 0.5	0.2 – 0.5	0.5 – 2.0
Improvement in Perf / Power	1x	~200x	~400x

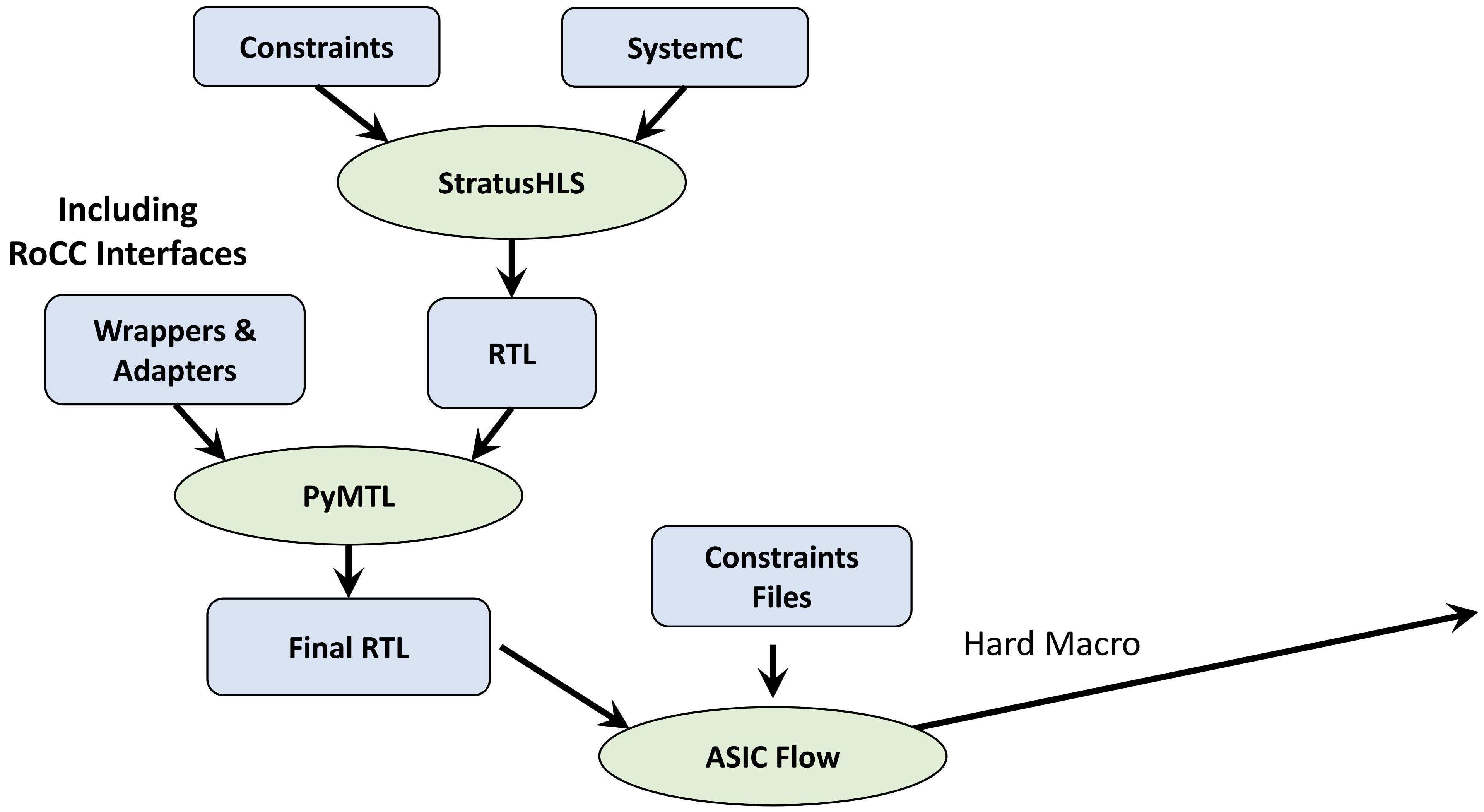
General-Purpose Tier	Software implementation assuming ideal performance estimated with an optimistic one instruction per cycle
Specialization Tier	Full-system RTL simulation of the BNN specialized accelerator running with a frequency of 625 MHz
Specialization + Massively Parallel Tiers	Full-system RTL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

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);
 }
}
```

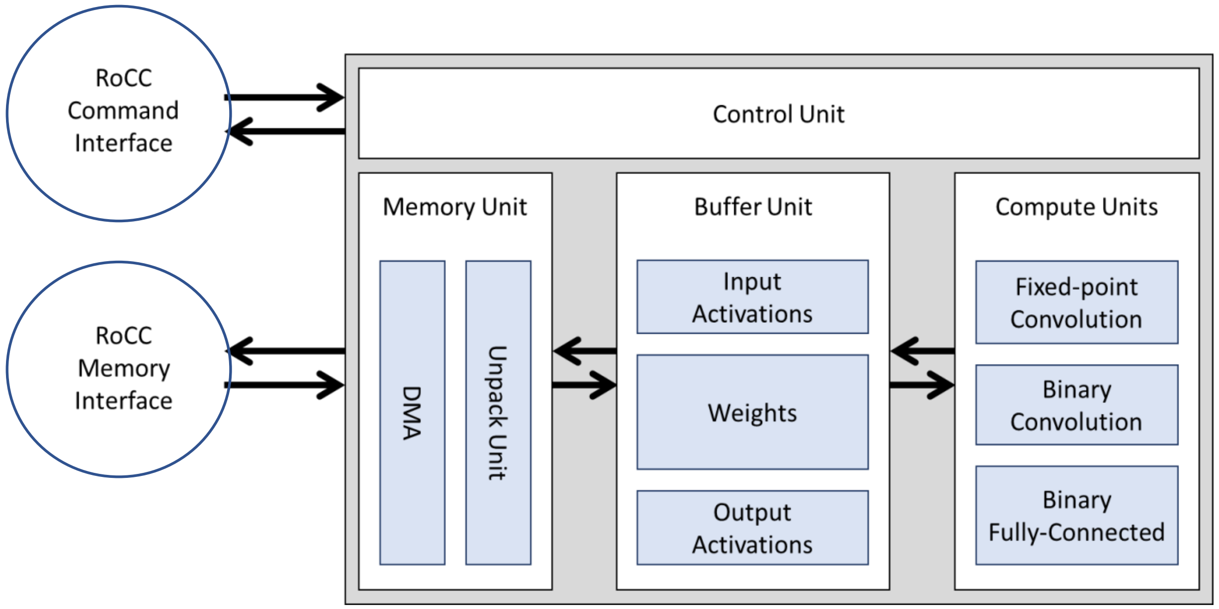
Design Methodology



RISC-V Ecosystem Successes and Challenges

Successes

- The RoCC command and memory interface were both significant successes. We connected the accelerator with **no changes to RV64G core**, just as we did for the manycore array in the massively parallel tier.

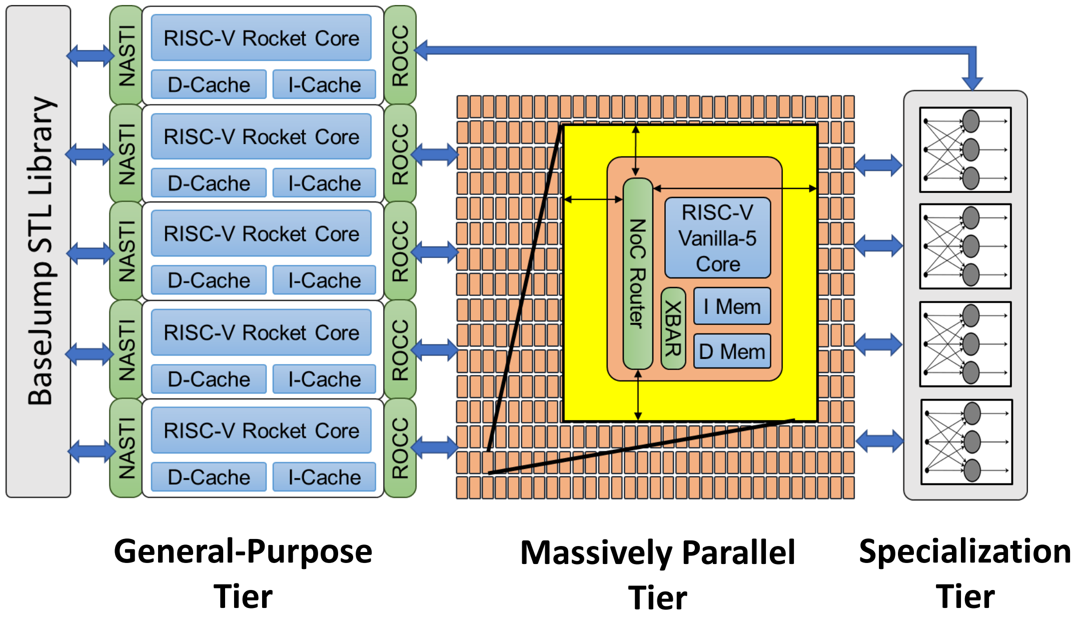


Challenges

- Small challenge in the RoCC accelerator interface at the specific commit we chose to use
  - Memory management unit in RV64G used only physical addresses
  - We did a small workaround to give us virtual addresses as well
  - This challenge has already been fixed upstream

The Celerity System-on-Chip

http://www.opencelerity.org



Celerity, an accelerator-centric SoC with a tiered accelerator fabric that targets highly performant and energy-efficient embedded systems

Celerity's goal was to develop new methodologies to design chips more quickly

We believe the **RISC-V software/hardware ecosystem** was instrumental in enabling a team of **20 graduate students** to tape out a complex SoC in **only 9 months**

Acknowledgements*: DARPA, under the CRAFT program*

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

We thank the many contributors to the open-source RISC-V software and hardware ecosystem with special thanks to U.C. Berkeley for forming the RISC-V ecosystem