Berkeley Architecture Research

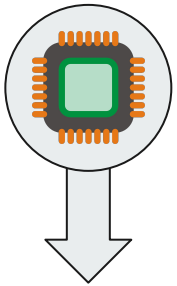
SLICE

ReRoCC: An Open-source Framework for Virtualized and Disaggregated RISC-V Accelerators

Jerry Zhao, Seah Kim,
Krste Asanovic, Borivoje Nikolic, Yakun Sophia Shao
<seah|jzh>@berkeley.edu

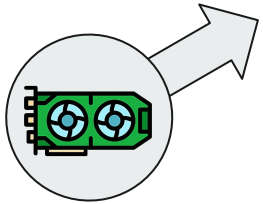
Berkeley
UNIVERSITY OF CALIFORNIA

Trends in Modern SoCs

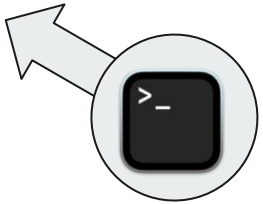


More cores
- End of single-thread performance scaling
- Many-core SoCs to extract TLP

How to scalably architect many-accelerator SoCs?

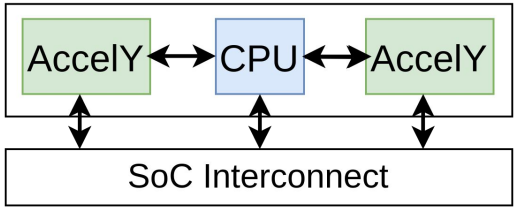


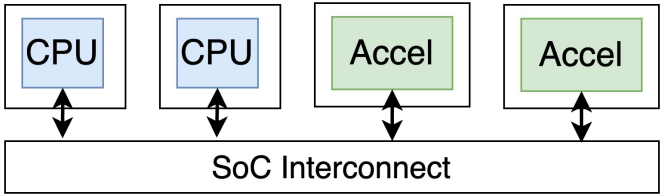
More accelerators
- More compute-bound workloads require acceleration



More applications
- Software stacks grow in complexity
- Graphics/multimedia/AI are pervasive

Existing Accelerator Integration Methodologies





CPU-integrated:

- Appears as custom ISA extensions to host software
- Simplifies software stacks, accelerators appear as architectural extensions in the instruction stream

Memory-mapped devices:

- Physically disaggregated over interconnect
- SW/HW features for managing these devices: interrupts/IOMMU/mmap

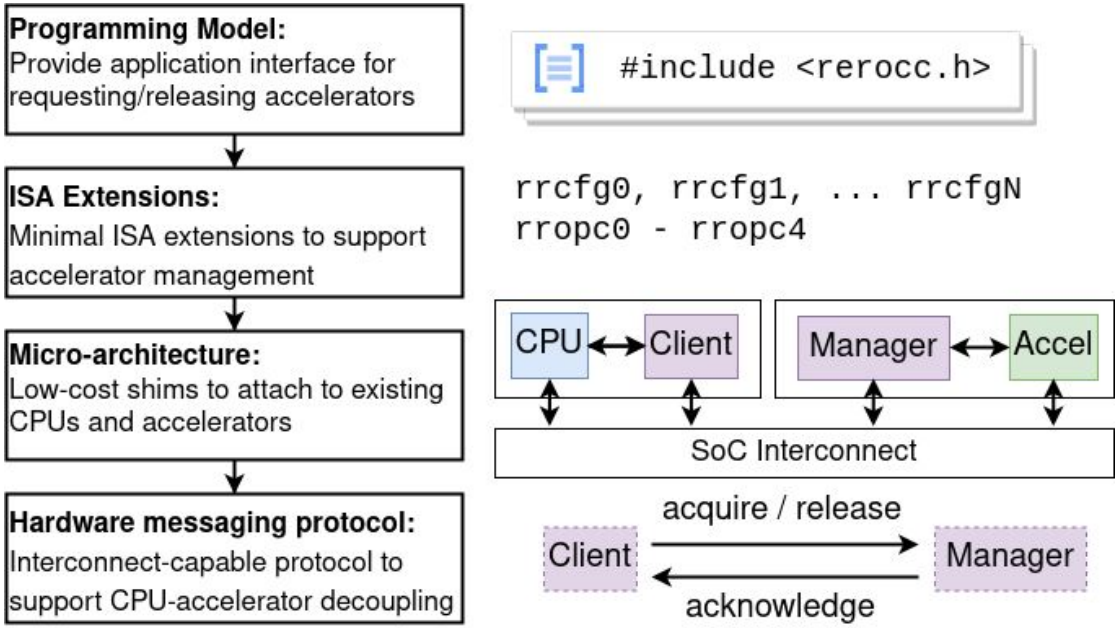
Can we develop a new accelerator integration methodology that preserves the advantages of both approaches?

Requirements for Approach

1. **Scalable to many-accelerator systems**
   - Implies physical scalability, distribute accelerators across an interconnect
   - Implies support for accelerator virtualization
2. **Minimize area overhead**
   - Low-cost hardware modules added to existing cores + accelerators
3. **Minimize latency for accelerator management**
   - Bound by physical interconnect latency, not the software stack
4. **Backwards compatiblility**
   - Out-of-the-box support for existing accelerator implementations
   - No modifications to existing accelerator software stacks

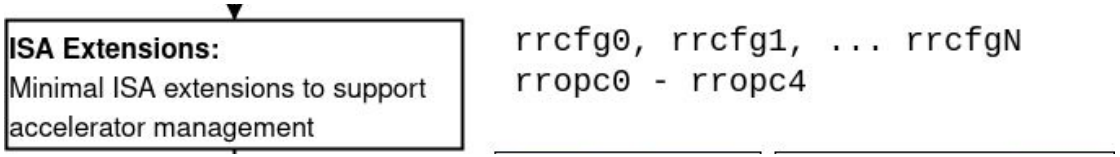
ReRoCC: Remote RoCC Accelerators

A full-stack system enabling disaggregation + virtualization of RoCC (tightly-coupled) accelerators



ReRoCC: Remote RoCC Accelerators

A full-stack system enabling disaggregation + virtualization of RoCC (tightly-coupled) accelerators



ReRoCC Architectural Extensions

System-level:

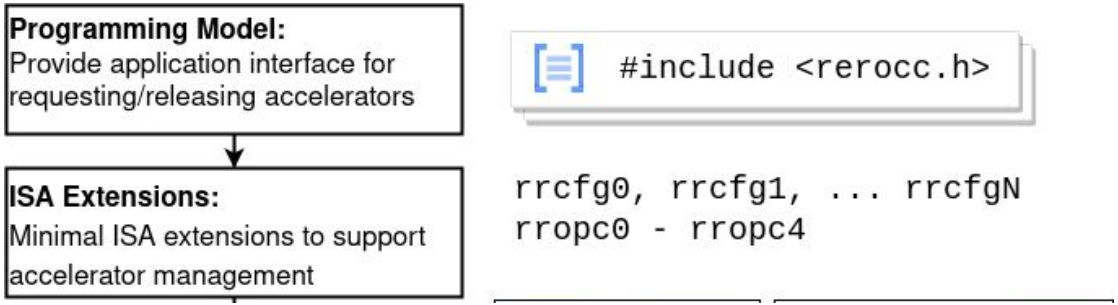
- Assumes a global physical "ID-space" of remotely-attached RoCC accelerators
- Platform should encode the physical "ID-space"

ISA-level:

- CSRs: `rrcfg0 - rrcfgX`
  - Configure which of the system-level physical accelerators are locked to this thread
  - Writes acquire/release shared accelerators
- CSRs: `rropc0 - rropc3`
  - Set which accelerator should receive instructions of this opcode
  - Enables virtualization of the opcode/accelerator space

ReRoCC: Remote RoCC Accelerators

A full-stack system enabling disaggregation + virtualization of RoCC (tightly-coupled) accelerators



ReRoCC Programming Model

Procedure:

1. Software should attempt to acquire an accelerator from the system

Code:

```
do {
  csr_write(CSR_RRCFG0, RRCFG_ACQ | 0x0);
} while (csr_read(CSR_RRCFG0) & !RRCFG_ACQ);
```

- Accelerator acquisition can happen in user-space
- Latency is just interconnect latency to query the accelerator for availability

- Platforms can implement multiple homogeneous accelerators
- User threads can query to request any, or multiple available accelerators
- Threads which fail to acquire can `sleep()` to deschedule themselves

ReRoCC Programming Model

Procedure:

1. Software should attempt to acquire an accelerator from the system

2. Software should map a local opcode to the accelerator

Code:

```
do {
  csr_write(CSR_RRCFG0, RRCFG_ACQ | 0x0);
} while (csr_read(CSR_RRCFG0) & !RRCFG_ACQ);

csr_write(CSR_OPC0, 0x0);
```

- Fast, low-latency, just writing some CSR on the core
- Supporting more acquired accelerators than opcodes lets us get around limited available opcode space

ReRoCC Programming Model

Procedure:

1. Software should attempt to acquire an accelerator from the system
2. Software should map a local opcode to the accelerator
3. After acquisition, accelerator appears to be architecturally part of the host thread

Code:

```
do {
  csr_write(CSR_RRCFG0, RRCFG_ACQ | 0x0);
} while (csr_read(CSR_RRCFG0) & !RRCFG_ACQ);

csr_write(CSR_OPC0, 0x0);

libaccel_execute_task();
```

- After acquisition, accelerator is part of host thread
- Unmodified accelerator kernels/libraries can be executed

ReRoCC Programming Model

Procedure:

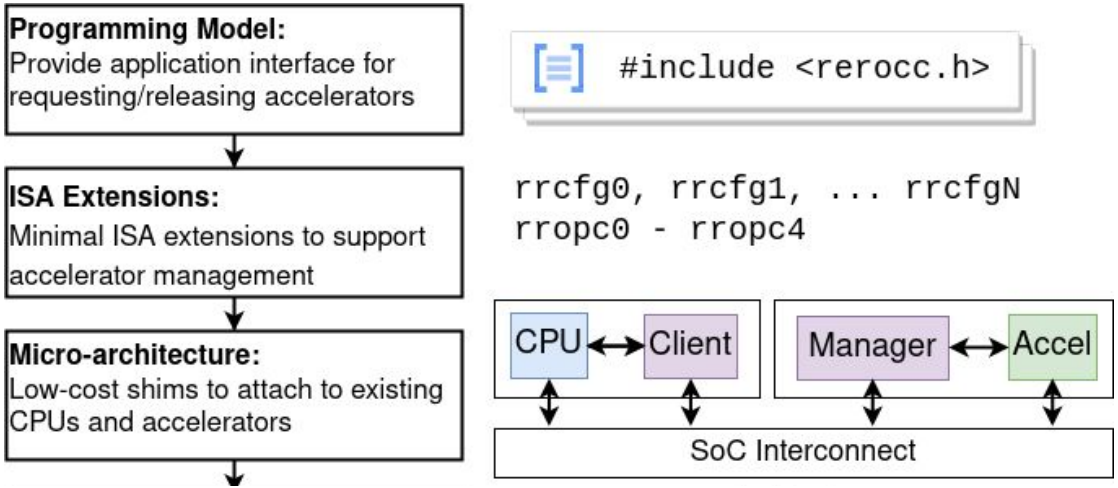
1. Software should attempt to acquire an accelerator from the system
2. Software should map a local opcode to the accelerator
3. After acquisition, accelerator appears to be architecturally part of the host thread
4. Software should release the accelerator to the system after completion

Code:

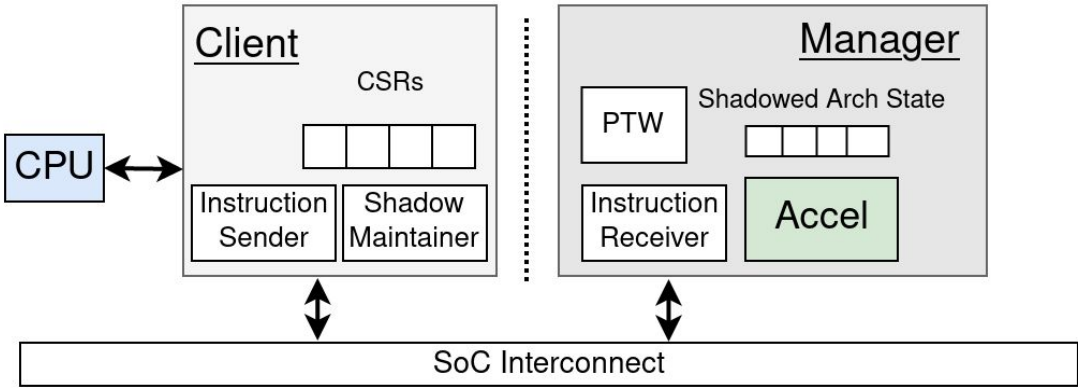
```
do {
  csr_write(CSR_RRCFG0, RRCFG_ACQ | 0x0);
} while (csr_read(CSR_RRCFG0) & !RRCFG_ACQ);

csr_write(CSR_OPC0, 0x0);

libaccel_execute_task();

csr_write(CSR_CFG0, 0x0);
```

ReRoCC: Remote RoCC Accelerators

A full-stack system enabling disaggregation + virtualization of RoCC (tightly-coupled) accelerators



ReRoCC Hardware Components



Client:

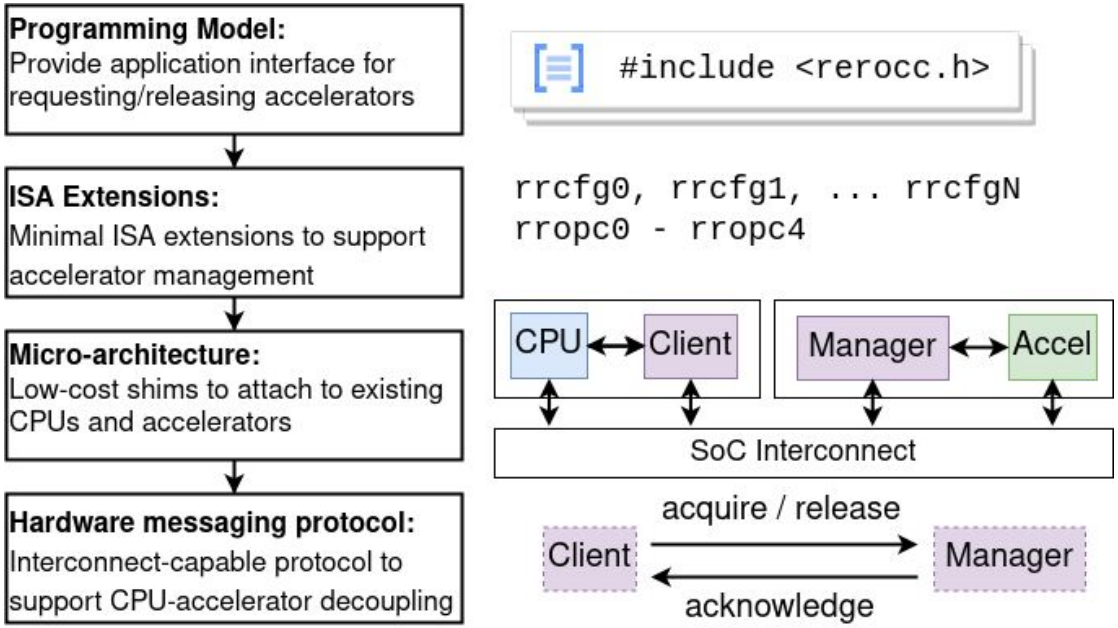
- Attaches to CPUs via RoCC
- Sends instructions to acquired accelerators
- Maintains shadows arch. state on the manager

Manager:

- Attaches to existing RoCC accelerators
- Implements PTW/TLB
- Maintains shadow copy of thread architectural state

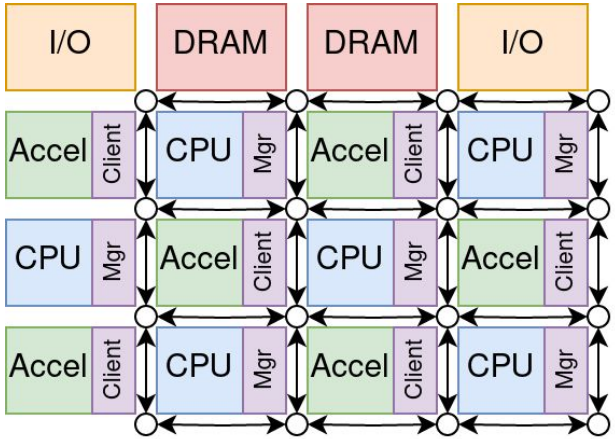
ReRoCC: Remote RoCC Accelerators

A full-stack system enabling disaggregation + virtualization of RoCC (tightly-coupled) accelerators



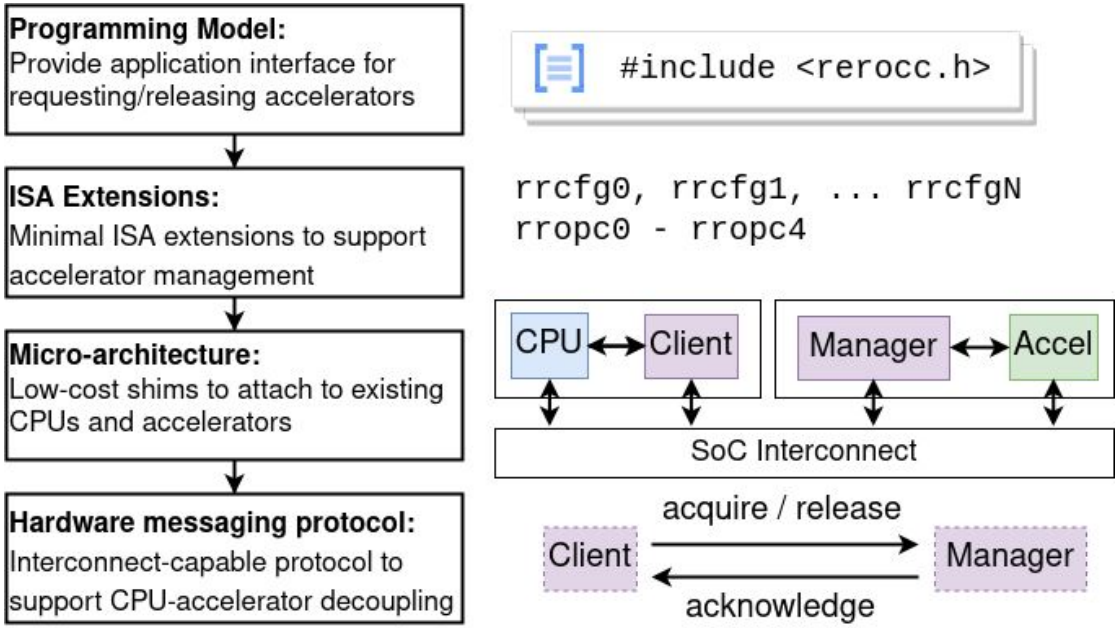
ReRoCC Messaging Protocol

- Non-blocking, two-channel (req/resp) FIFO protocol for client-manager communication
- Minimal set of flows
  - Acquire->AcquireResp : Attempt to acquire an accelerator for the client
  - Inst->InstAck : Send instructions, performs flow control
  - Update : Update shadowed architectural state
  - Release : Relinquish an accelerator
- Interconnect ready - adapters to support overlay on top of example NoC implementation (Constellation)



ReRoCC: Remote RoCC Accelerators

A full-stack system enabling disaggregation + virtualization of RoCC (tightly-coupled) accelerators



Open-sourced at github.com/ucb-bar/rerocc

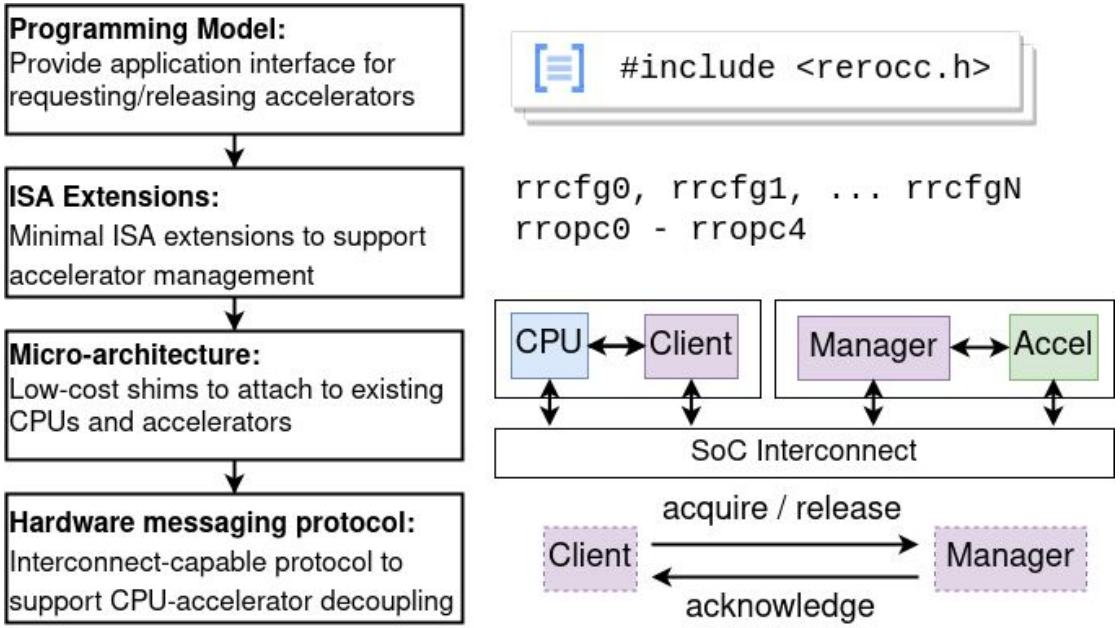
ReRoCC Directions

Common open-source platform for exploring new ideas

- Provides an implementation of existing proposals for virtualized accelerators
- Enables easy evaluation of tradeoffs between tightly-coupled/disaggregated accelerator architectures
- Lots of design parameters to explore in the Client/Manager/Interconnect implementations
- Compatible with existing open-source Chipyard-based ecosystem

ReRoCC: Remote RoCC Accelerators

A full-stack system enabling disaggregation + virtualization of RoCC (tightly-coupled) accelerators



Open-sourced at github.com/ucb-bar/rerocc

Acknowledgements

Research was partially funded by NSF CCRI Award #2016662 and partially by SLICE Lab industrial sponsors and affiliates. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.