

# CXL-Bridges

Generating Heterogeneous Simulations with gem5

Simon Dittrich

Advisor: Dr. Anatole Lefort

Chair of Computer Systems

<https://dse.in.tum.de/>



15.3.2024 – 20.10.2024

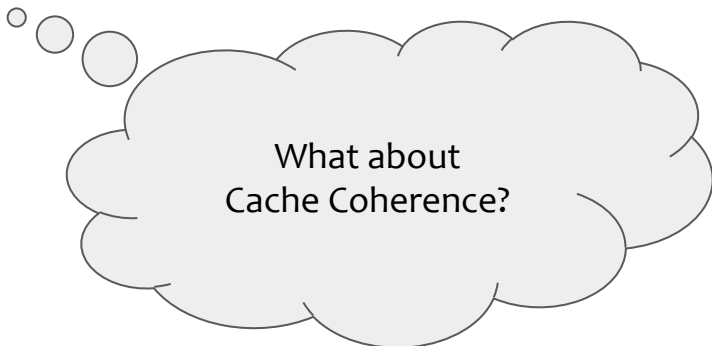
# Background: What is CXL?

- Open high-speed interconnect standard
- Initially released in 2019
- Primarily developed by Intel
- Aimed at high performance data center use



# Background: Why is CXL interesting?

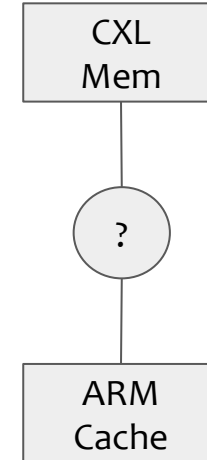
- Enables memory pooling across racks
  - Improved memory utilization
  - Reduced hardware costs
- Shared memory communication between nodes
  - Databases could return direct pointers to data
  - Directly shared state
  - etc.



What about  
Cache Coherence?

# Motivation: Cache Coherence in CXL

- CXL.mem defines MESI as the “wire-protocol”
- What about non-MESI systems?
  - ARM Processors (MOESI)
  - GPUs
- Translate between protocols with dedicated bridges
  - What is the performance impact of this translation?
  - Does it depend on the bridged protocol?
- Let's simulate various bridges and find out



# Background: Cache simulation with gem5

- gem5 includes the Ruby memory system to simulate various cache architectures.
- Ruby cache controllers are written in a DSL (SLICC)
  - Fairly low-level (*essentially syntax sugar over C++*)
  - Contains the entire state machine incl. transient states
  - All possible transitions must be defined
  - Essentially a black box that receives and sends messages
- Ruby controllers are compiled directly into gem5
  - Exposed the same way as C++ components

```
transition(E, Inv, I) {  
    // don't send data  
    forward_eviction_to_cpu;  
    fi_sendInvAck;  
    l_popRequestQueue;  
}  
  
transition(E, Fwd_GETX, I) {  
    forward_eviction_to_cpu;  
    d_sendDataToRequestor;  
    l_popRequestQueue;  
}  
  
transition(E, {Fwd_GETS, Fwd_GET_INSTR}, S) {  
    d_sendDataToRequestor;  
    d2_sendDataToL2;  
    l_popRequestQueue;  
}
```

```
for cpu in cluster:  
    l1cache = L1CacheA_Controller()  
    l1cache.cacheMemory = RubyCache(size = '16kB')
```

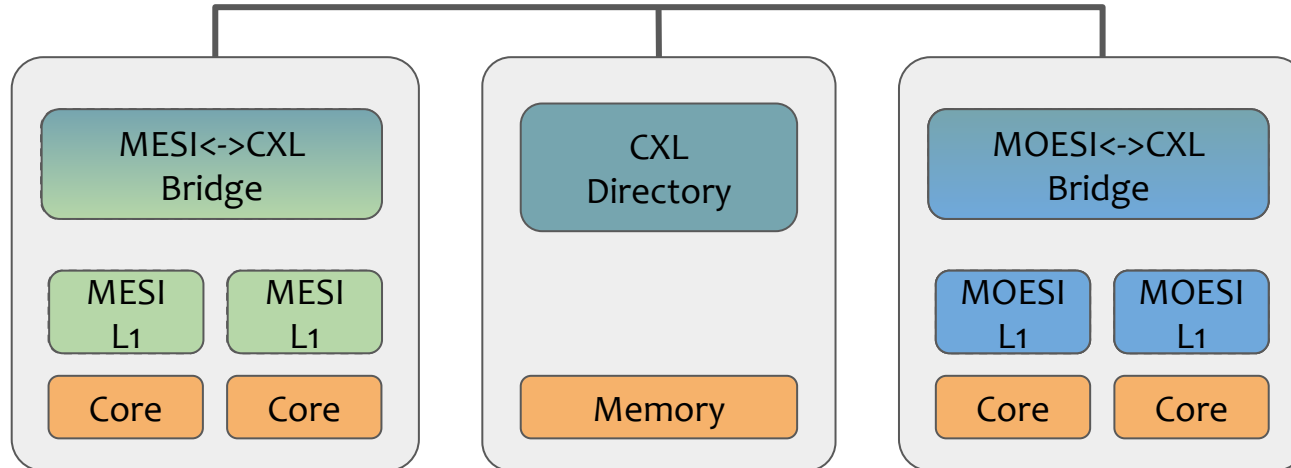
- Challenge 1: Bridge state machine complexity
  - Bridge states are a cross product of source protocol states
  - Example numbers (stable states, transient states, transitions):
    - Original protocols: 4-5, 15, 30
    - Bridge: 12, 50, 220
- Challenge 2: Manually writing SLICC is difficult
  - Controllers (especially Bridges) can get very complicated
  - Dependent on knowledge and experience with SLICC
  - Risk of introducing bugs
  - Risk of implementing protocols inefficiently and skewing results

# Motivation: Simulating Heterogeneous Systems

- Problem: We want to test many different system configurations
  - Many protocol combinations possible
  - Exploring new combinations should be easy
- We want a solution that is independent of specific protocols
- Manually writing SLICC appears to be a poor choice

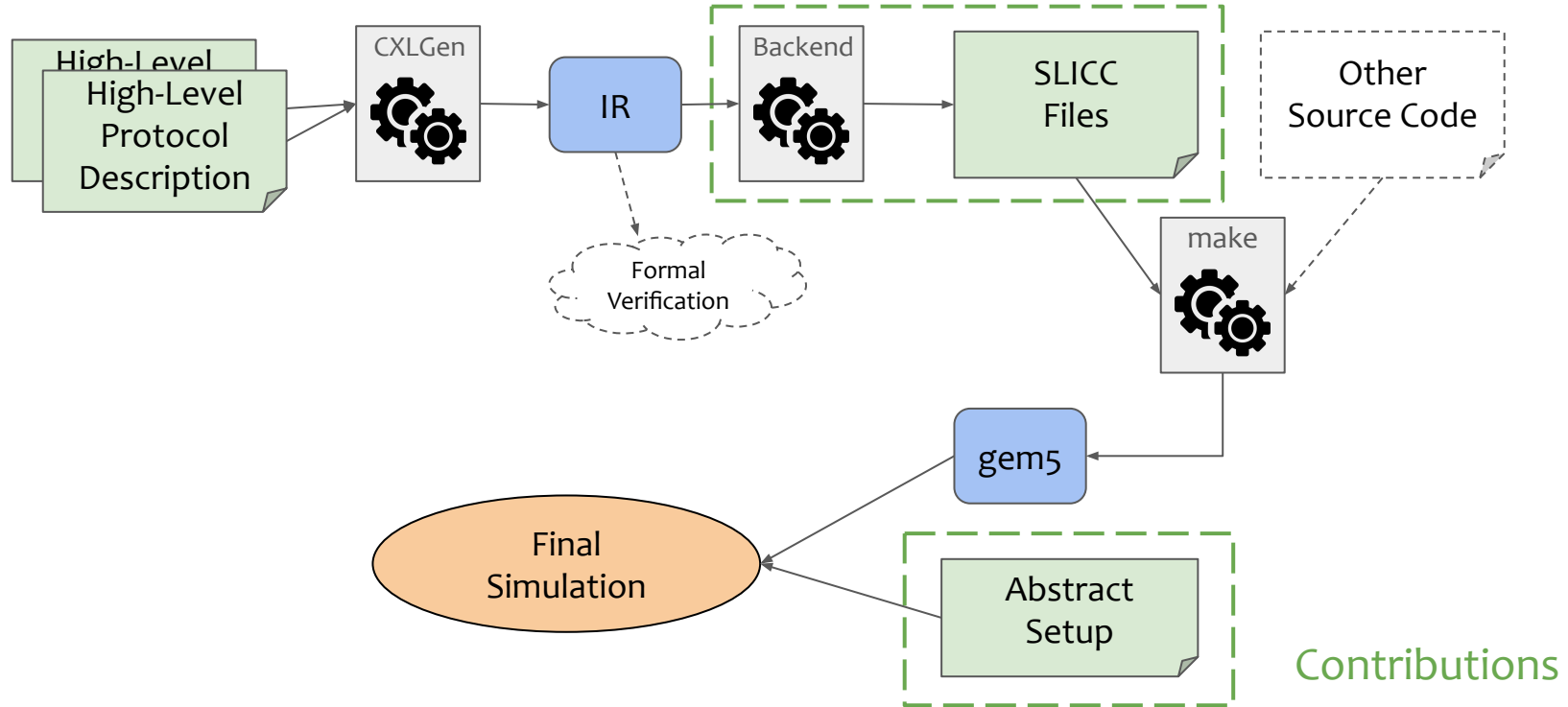
Can we automatically generate the necessary SLICC controllers?

- We setup a generic simulation with abstract cache controllers
  - This includes homogeneous controllers such as L1
- We automatically generate concrete implementation for these abstract controllers and slot them in





# Design: Simulation Pipeline



# Implementation: Controller Generation

- The IR of each protocol primarily contains:
  - Messages
  - States
  - Transitions
  - *actions* (as part of Transitions)
- Some necessary information isn't included

```
machine(MachineType:L1Cache, "MESI cache")
: //...
{
  state_declaration(State, desc="Cache states") {
    I, AccessPermission:Invalid, desc="...";
    M, AccessPermission:Read_Write, desc="...";
    IM, AccessPermission:Busy, desc="..";
    //...
  }

  enumeration(Event, desc="Cache events") {
    Load, desc="Load from processor";
    Store, desc="Store from processor";
    //...
  }

  //...

  action(a_issueRequest, "a", desc="Issue a request") {
    //...
  }

  //...

  transition(I, Store, IM) {
    a_issueRequest; //...
  }

  //...
}
```



# Implementation: Function Access Permission

- Required by gem5 to know when it is safe to modify memory associated with a controller
- Transient state are almost always marked as **Busy**
- Cache states: Look at load/store transitions
- Directory states: Invert permission of equivalent cache state

Cache			Directory		
State	Read	Write	State	Read	Write
M	✓	✓	M	✗	✗
E	✓	✓	E	✗	✗
S	✓	✗	S	✓	✗
I	✗	✗	I	✓	✓

# Implementation: Functional Access Permissions

- Bridges use compound states (e.g:  $S \times M$ )
  - directory-side
  - cache-side
- Calculate permissions for each state individually
- Join their permissions:
  - **Busy:** If either of them is a transient state
  - **ReadWrite:** If cache is *ReadWrite* and directory is *Invalid*
  - **Read:** If cache is at least *Read* and directory is *Read*
  - **Invalid:** Everything else

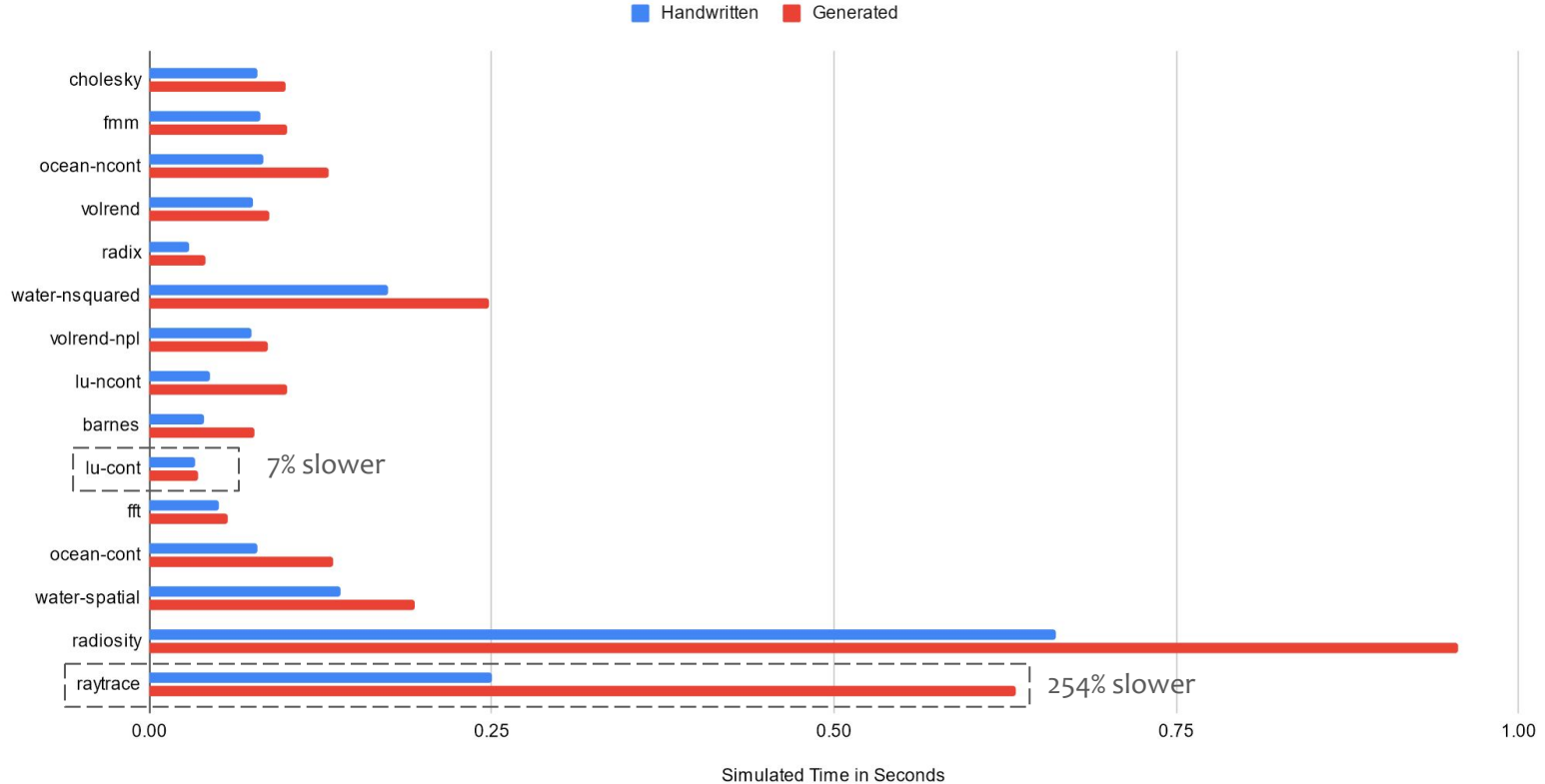
		cache			
		M	E	S	I
directory	M	I			
	E	I			
	S	R	R	R	
	I	RW	RW	R	I

# Implementation: Other Issues

- IR is not made for made exclusively for SLICC
  - Some operations are not supported by SLICC
  - Some information necessary for SLICC is missing
- SLICC requires the programmer to deal with many, low-level details
  - Correctly sleeping when when no progress can be made
  - Correctly waking up when progress is possible again
  - Evicting old cache lines if no more space is available
  - Allocating and freeing temporary data structures

- Are our generated protocols competitive with existing protocols?
  - We compare against handwritten protocols included with gem5
  - Homogenous protocols only
  - MESI + MOESI
- We use the Splash-4 benchmark suite

# Evaluation

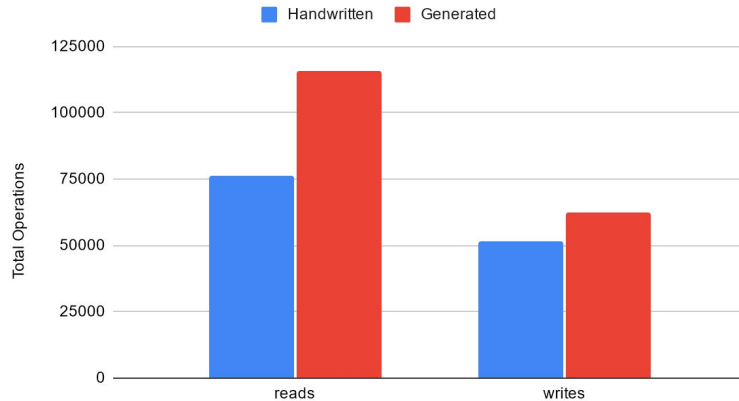


Protocol: **MESI**

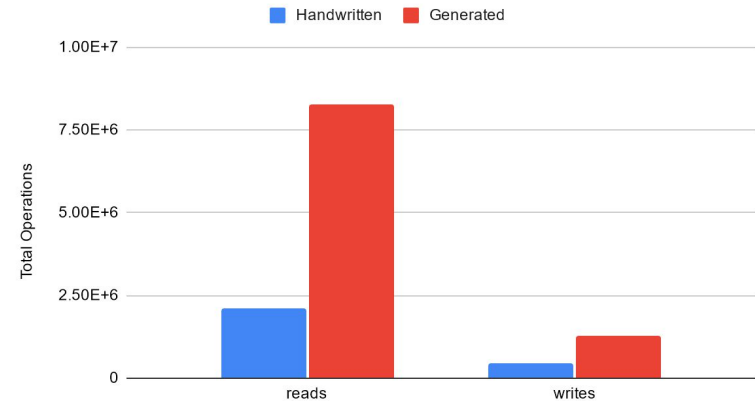
*MOESI yields similar results*

- Comparison not entirely fair
  - Missing instruction caches
  - Missing shared cache
- Can we verify this assumption?
  - Look at the difference in memory operations

lu-ncont



raytrace





## The creation of *bridge* simulations should be automated

- High complexity of *bridge* state machines
- Manually writing such a controller for gem5 is already difficult
- We requires multiple such controllers with consistent code quality

## We automate the creation of *bridge* simulations using the following steps

- We are generating *bridges* from high level descriptions
- We automatically output all required gem5 controllers
- We slot these generated controllers into a prepared simulation setup

**Try it out!**

Generator: <https://github.com/Gingerbreadz/CXLGen>

Abstract Setup: <https://github.com/sidit77/cxlgen-gem5>

# Implementation: Abstract Setup

- Python script executed by gem5 at startup
- Simulate all nodes as single system
  - Model CXL using interconnect primitives with high latency
- Get all controller to talk with each other in the correct order
  - Problem: gem5 typically assigns each controller an address range
- Main Challenge: Very little documentation available
  - A lot of trial-and-error