Cache Coherent Framework for RISC-V Many-core Systems

Zexin Fu, Mingzi Wang, Yihai Zhang, Zhangxi Tan

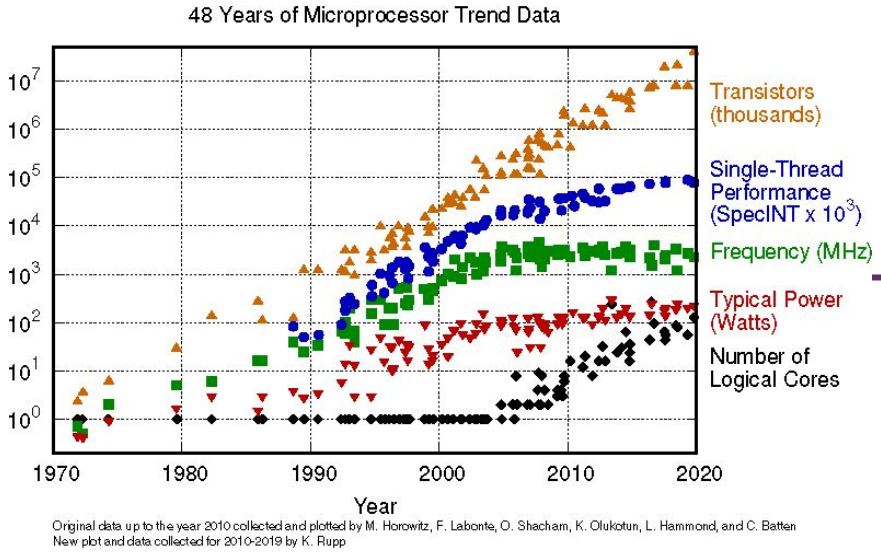
RISC-V International Open-Source Laboratory

Tsinghua University

Contents

- Motivation
- System architecture
  - Coherence protocol design
  - Coherence controller design
  - Network-on-Chip design
- Verification methodology
- Evaluation
- Conclusion and future work

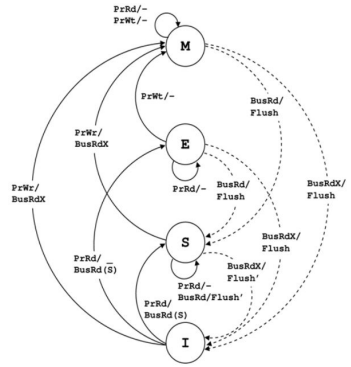
Motivation



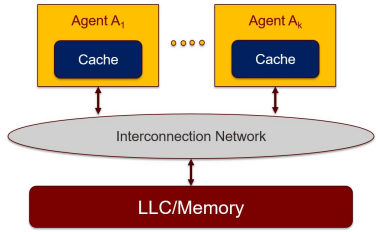
Rapidly growing core count

New challenges

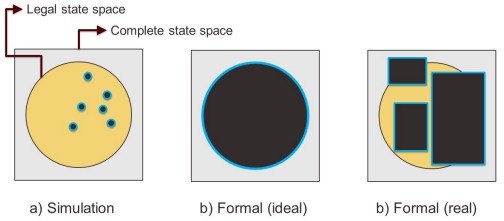
1. **Cache coherence maintenance**



2. **Interconnect scalability**



3. **Large system verification**



- Frequency ⇨
- Single-threaded performance ⇨
- Core count ⇗

Motivation

Challenges

1. **Cache coherence maintenance**
2. **Interconnect scalability**
3. **Large system verification**

Our Solutions

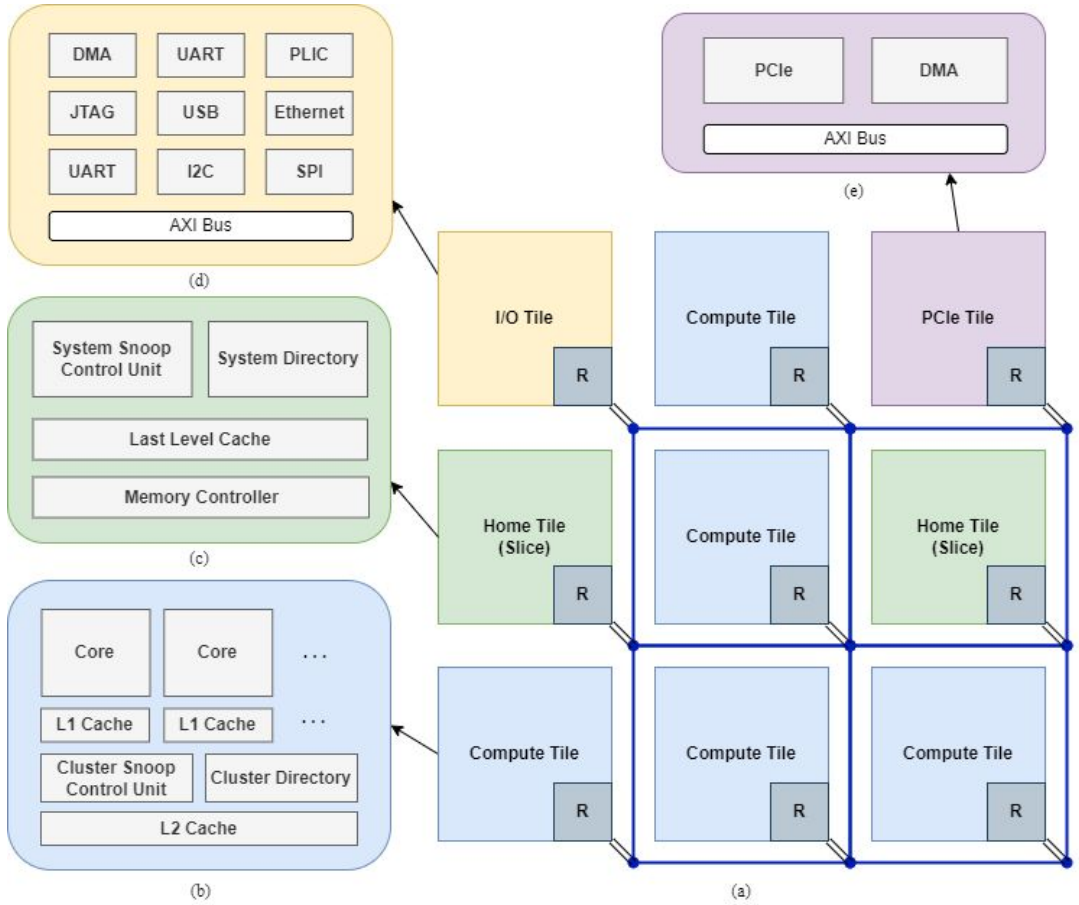
1. **Verified cache coherent fabric**
2. **Scalable Network-on-Chip IP**
3. **Generated verification models combining simulation & formal**

System architecture

- Coherence protocol design
- Coherence controller design
- Network-on-Chip(NoC) design

System architecture

- **Features**
  - Tile-based flexible design
  - Scalable package-switching NoC
  - Two-level coherence control to reduce traffic
- **Tiles**
  - Compute Tile
  - Home Tile
  - I/O Tile
  - PCIE Tile



System architecture

Coherence protocol design

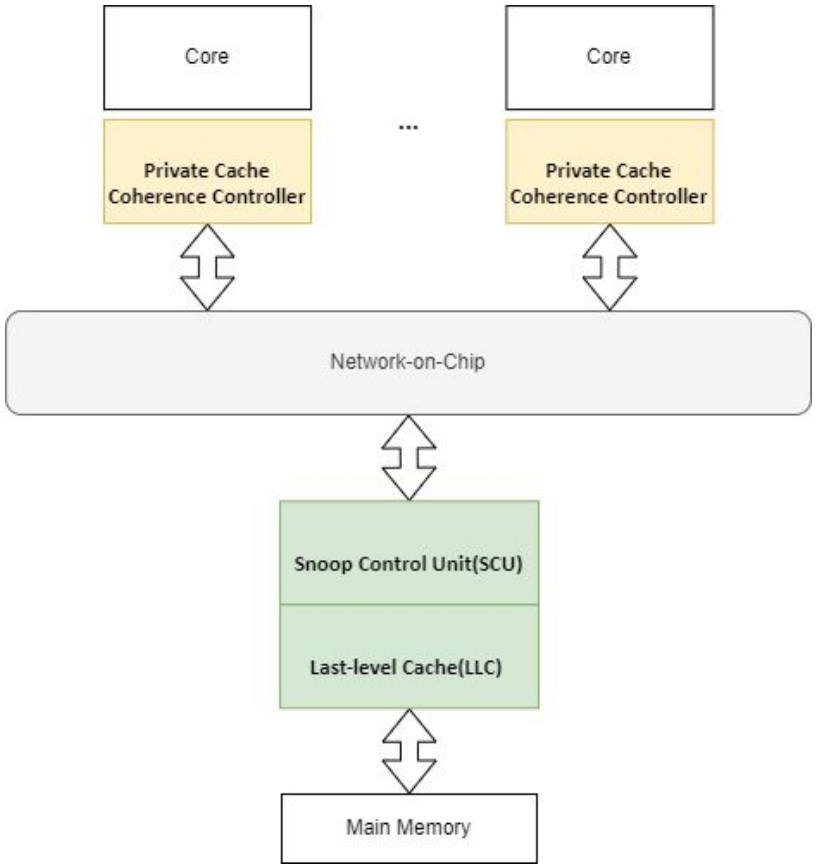
- **Features**
  - **MESI protocol**
    - Optimize for sequence of load then store requests
  - **Sparse directory-based** cache line tracking mechanism
    - Less directory overhead while low coherence traffic
  - **Five Coherence message**
    - Each form a virtual network to avoid deadlock

Five virtual networks

Channels	Can be blocked?	Description
Request	Yes	Request from private cache to SCU
Evict	Yes	Evict request from private to SCU
Response	No	Response message, both direction
Snoop	No	Snoop request from SCU to private cache
Data	No	Data message, both direction

Coherence fabric

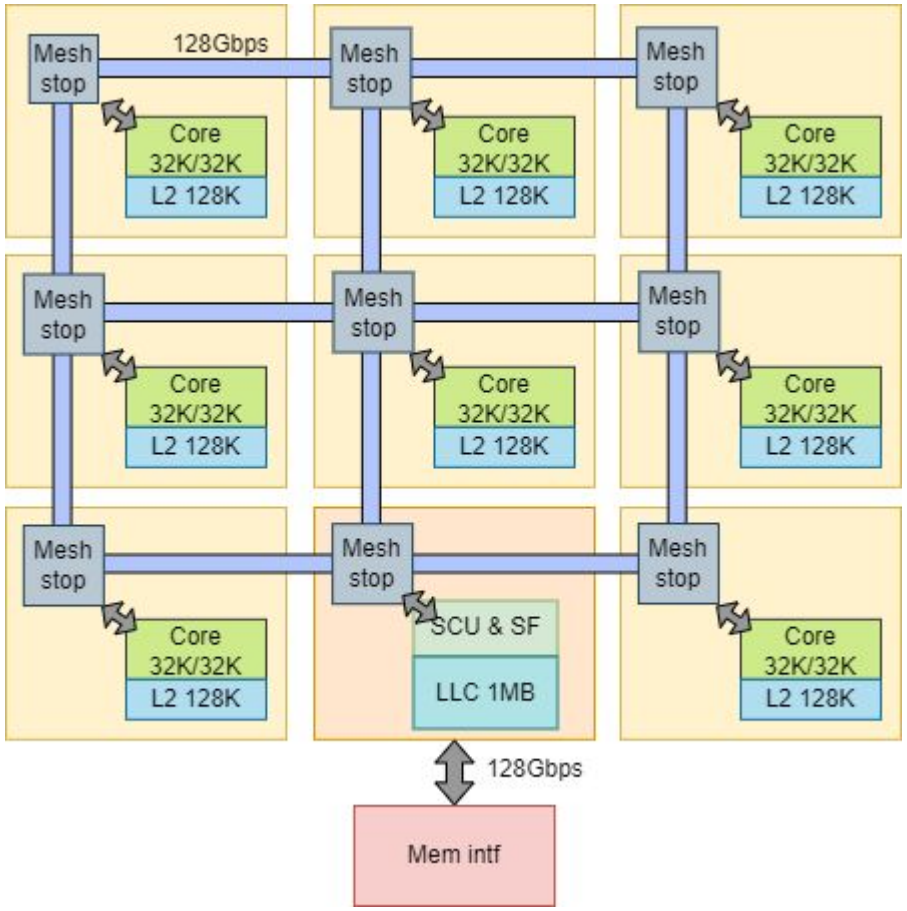
- **Private Cache Coherence Controller**
  - Private cache
  - Snoop request handling logic
  - Coherent fabric interface with cores
- **Snoop Control Unit(SCU) & LLC Controller**
  - Directory
  - Coherence control logic
  - Last-level Cache
  - Coherent fabric interface with the main memory



Coherence controllers

Network-on-Chip(NoC) design

- Features
  - **2D Mesh topology**
    - High scalability
    - Easy to layout on-chip
  - **Virtual Channel (VC) flow control**
    - Multiple virtual channels time-division multiplex physical channels to **improve link utilization**.
    - Flits from different messages can be put in the same VC to **improve buffer utilization**.
  - **Quality of Service (QoS) support**
    - Priority based on the QoS value of packages
    - Extra real-time VC channel for I/O and real-time applications

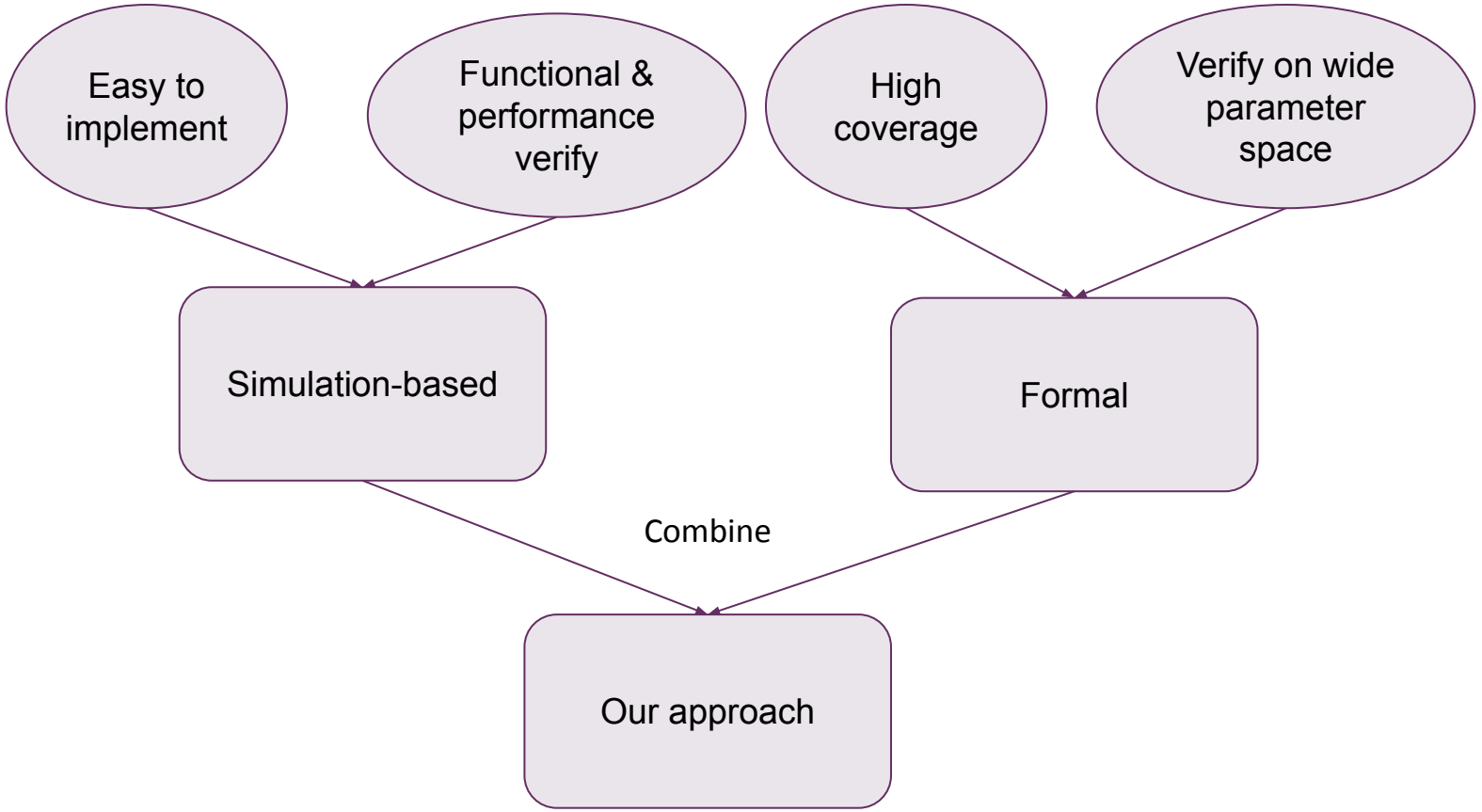


NoC architecture

Verification methodology

- Simulation-based verification
- Formal verification
- Combine both methods

Verification methodology



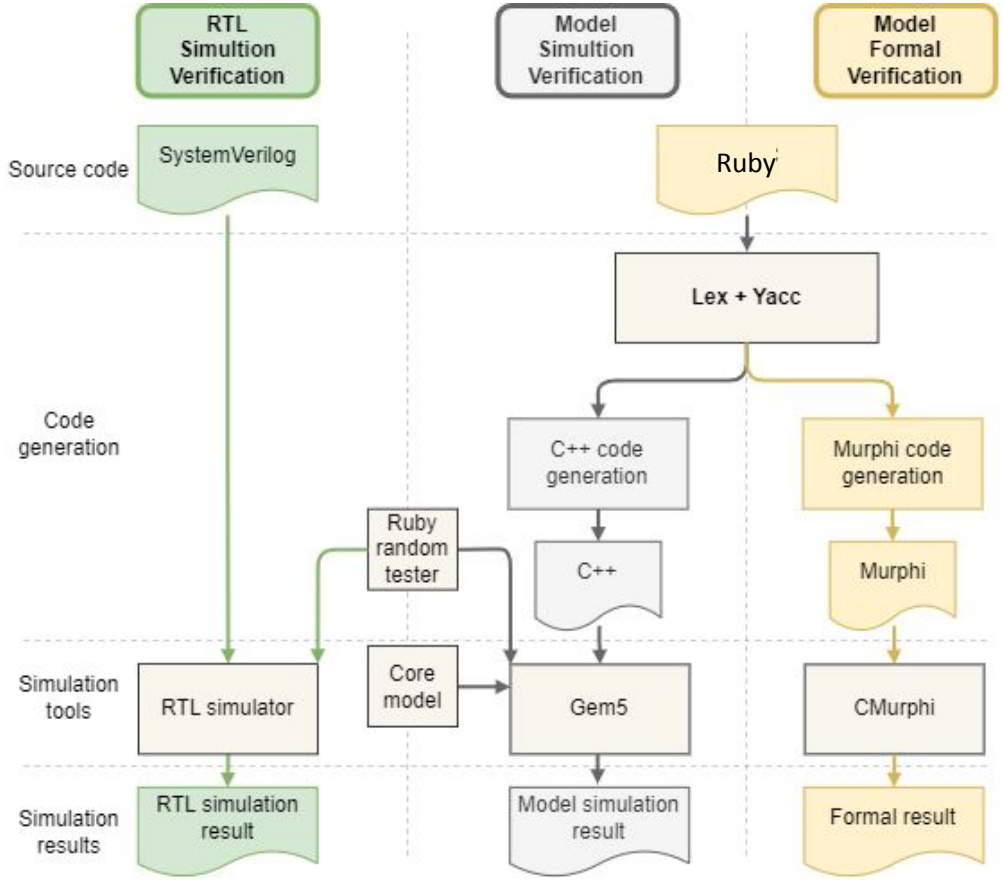
Verification methodology

Simulation

- DUT:
  - **Gem5 ruby** cache coherence model
  - **RTL** implementation
- Tester:
  - Ruby random tester

Formal

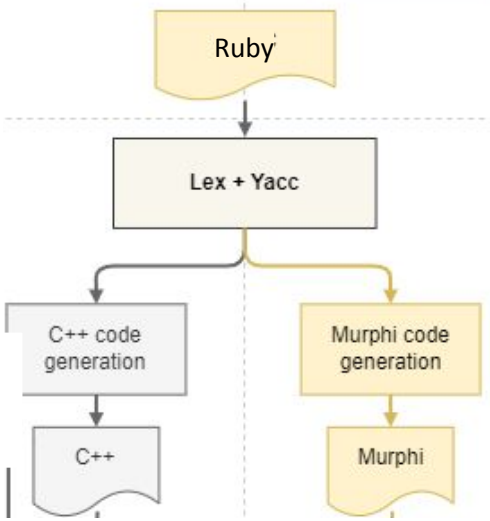
- DUT:
  - **Murphi** generated from **Gem5 ruby model**
- Tester:
  - CMurphi



Combined verification flow

Verification methodology

- Generate Murphi formal model from Ruby simulation model.



Code generating flow

Ruby Code

```
1:  transition (I, Store, IM_AD) {
2:          allocateCacheBlock;
3:          allocateTBE;
4:          sendGetM;
5:          popMandatoryQueue;
6:  }
```

Murphi Code

```
1:  Rule "L1Cache_State_I ==> L1Cache_Event_Store ==>
2:          L1Cache_State_IM_AD"
3:      ( ( isundefined ( p.tbe.m_TBEState ) ?
4:        p.entry.m_CacheState : p.tbe.m_TBEState )
4:        = L1Cache_State_I ) &
6:      ( L1Cache_GetQueueHead ( p.eventqueue ) =
7:        L1Cache_Event_Store )
8:  ==>
9:      p.tbe.m_TBEState := L1Cache_State_IM_AD;
10:     p.entry.m_CacheState := L1Cache_State_IM_AD;
11:     L1Cache_allocateCacheBlock(n);
12:     L1Cache_allocateTBE(n);
13:     L1Cache_sendGetM(n);
14:     L1Cache_popMandatoryQueue(n);
15:     L1Cache_Dequeue(p.eventqueue);
16: endrule;
```

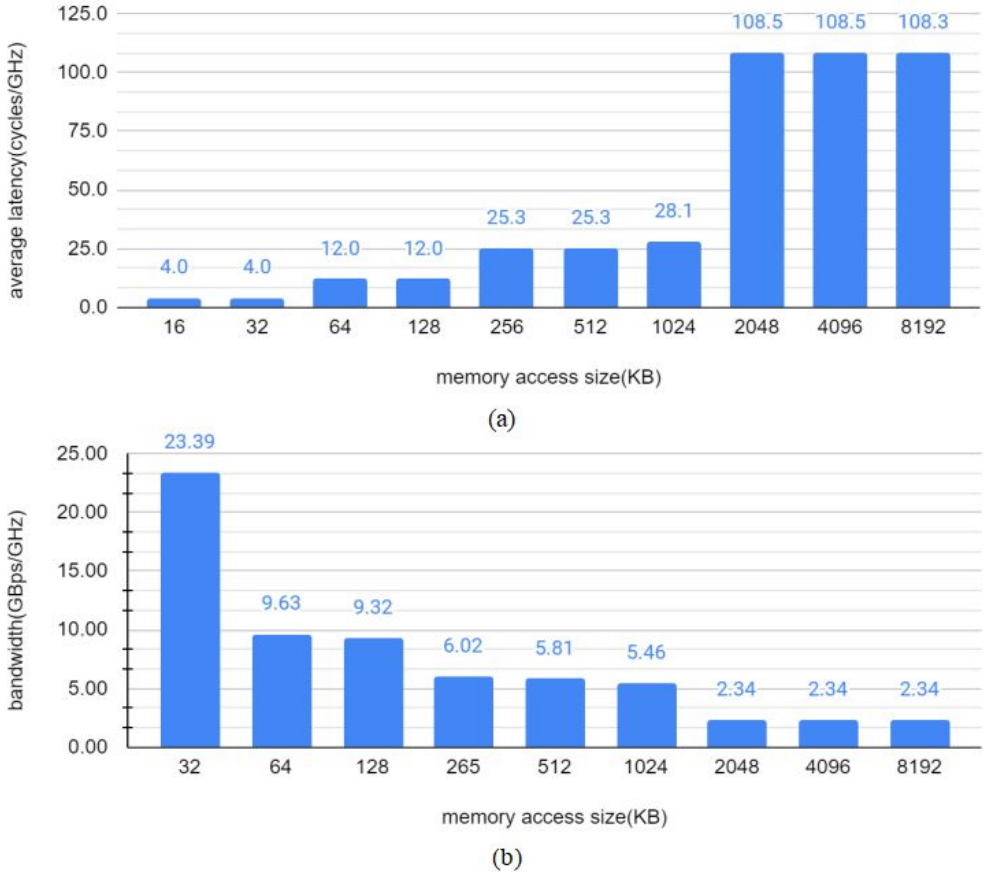
Example of generating Murphi code from Ruby code

Evaluation

- Cache Performance
- Formal verification
- ASIC Prototype

Cache Performance

- Benchmark: Lmbench
- Configuration
  - Core: ~Arm A75
    - Max throughput: 2 load and 1 store per cycle
  - Cache
    - L1 I/D: 32KB, 4 ways
    - L2: 128KB, 4 ways
    - LLC: 1MB, 8 ways



Lmbench cache benchmark result:
(a) latency; (b)bandwidth

Formal verification

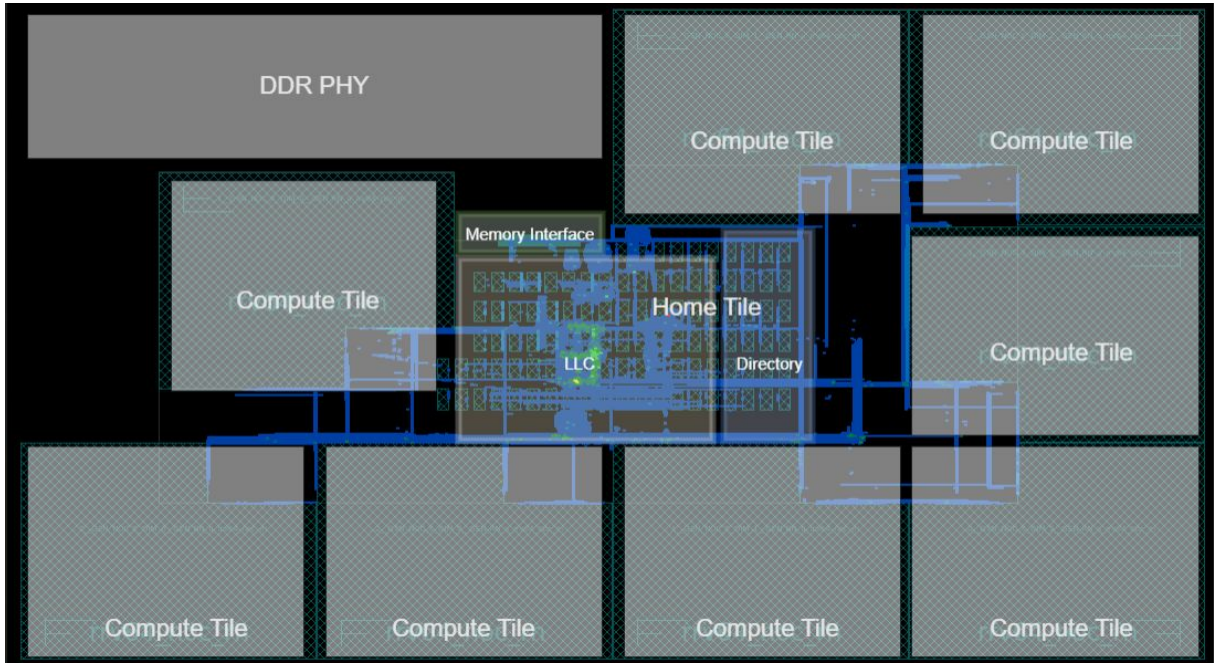
- Generated Murphi code from Gem5 Ruby SLICC code
- **Found deadlock in a Ruby model in gem5**
  - The model is design for a point-to-point ordered interconnect, when it shift to a unordered NoC, it runs into deadlock
- **Discovery:**
  - Formal verification is effective for finding deadlocks.
  - When the number of cores exceeds three, the verification time will be prohibitively long.

Murphi formal verification results

Protocol	Core number	State number	Time	Result
MI	1	121	8.46s	No error
	2	276111	64.85s	No error
	3	123768664	61783.27s	No error
MSI	1	5596	8.89s	No error
	2	22741	10.71s	Deadlocked
	3	99504	21.63s	Deadlocked

ASIC Prototype

- Testchip tapes out at 6nm process
  - Test cache coherence system with NoC
  - Verify NoC backend implementation
- System configuration
  - 1 **Home Tile**
    - with 512 KB LLC and 100 KB system directory
  - 8 **Compute Tile**
    - 32 KB L1 instruction cache, 32 KB L1 data cache
    - 128 KB L2 cache.
  - 3x3 **Mesh NoC**
- Backend implementation
  - Target frequency: 1GHz
  - Process corner: 0.85V@TT
  - Total area: 4.52 mm^2, each
    - Compute Tile: 0.51 mm^2
    - Home Tile: 0.46 mm^2



ASIC prototype layout

Experience

1. The performance overhead of coherence need to be reduced.

   => including cut down the critical path of coherence transactions, filter more coherence traffic at near core side and design more efficient coherence protocol.

2. Formal verification is powerful but has state explosion problem

   => Exploring ways to accelerate formal tools, including multi-threading and hardware acceleration

3. Back-end implementability is important in Network-on-Chip design

   => High bandwidth designs may strain routing resources

Conclusion and future work

This work presents

- A framework of a many-core system consisting of four kinds of tiles;
- A 2D-mesh-based Network-on-Chip;
- A verification process combining simulation and formalization from model to RTL.

Future work

- Exploring coherence system and cache design;
- Improve NoC with more real workloads;
- Large-scale system-level emulation;
- Accelerate formal verification process;
- Try and improve Open EDA flow