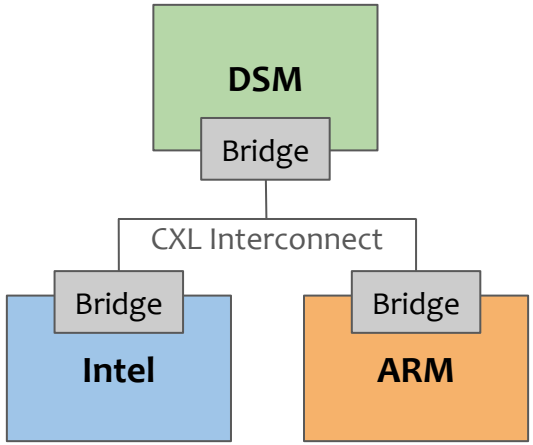
Formal Verification of Heterogeneous Cache Coherence Protocols for CXL

Julian Pritzi
Advisor: Anatole Lefort
Chair of Computer Systems
https://dse.in.tum.de/

16.04.2024 – 16.10.2024

Motivation: Heterogeneous Cache Coherence

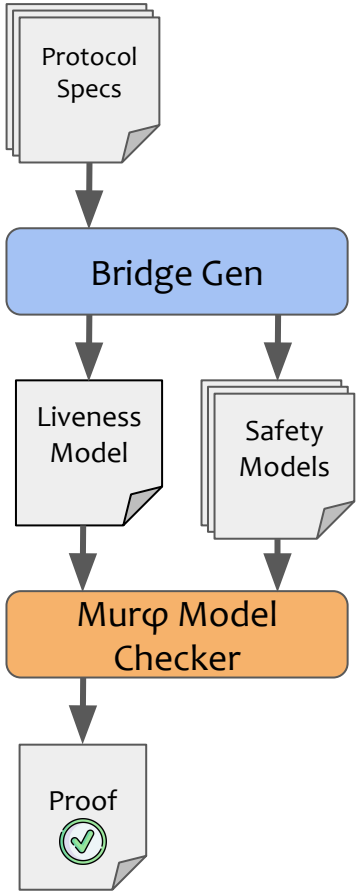
- Disaggregation & scaling ⇒ CXL (Compute Express Link)
- Heterogenous CXL Setups
  - Different memory consistency
- How to ensure cache coherence?
  - Interface between cache coherence protocols
  - Hardware components (bridges)



How to verify the correctness of the heterogeneous cache coherence protocols (HCCP)?

State of the art: Model Checking

- Model protocol and relevant system components
- Exhaustively explore all potential states
  - Using tools such as Murφ, Rumur, etc.
- Check the desired properties for each state
  - Safety: adherence to Memory Consistency Model
  - Liveness: deadlock freedom

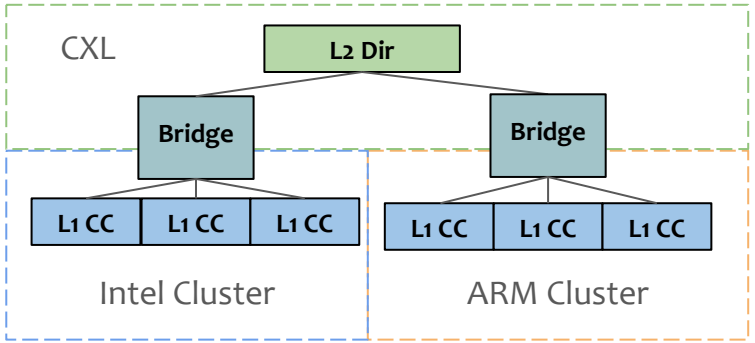


workflow of HeteroGen[1]

[1]Oswald, Nicolai, et al. "HeteroGen: Automatic synthesis of heterogeneous cache coherence protocols." HPCA 2022

Limitations of the state of the art

- Large system, with many relevant components.
- State space grows exponentially.
  ⇒ directly affects required resources.
- HieraGen exceeds **256GB** of RAM[1],
  with just **one** L1 cluster.



- HieraGen artifacts[2] modified to model **two** L1 clusters exceed **2TB** of RAM.

[1]Oswald, Nicolai, et al. "HieraGen: Automated generation of concurrent, hierarchical cache coherence protocols." ISCA 2020
[2]https://github.com/Errare-humanum-est/HieraGen

Research Question

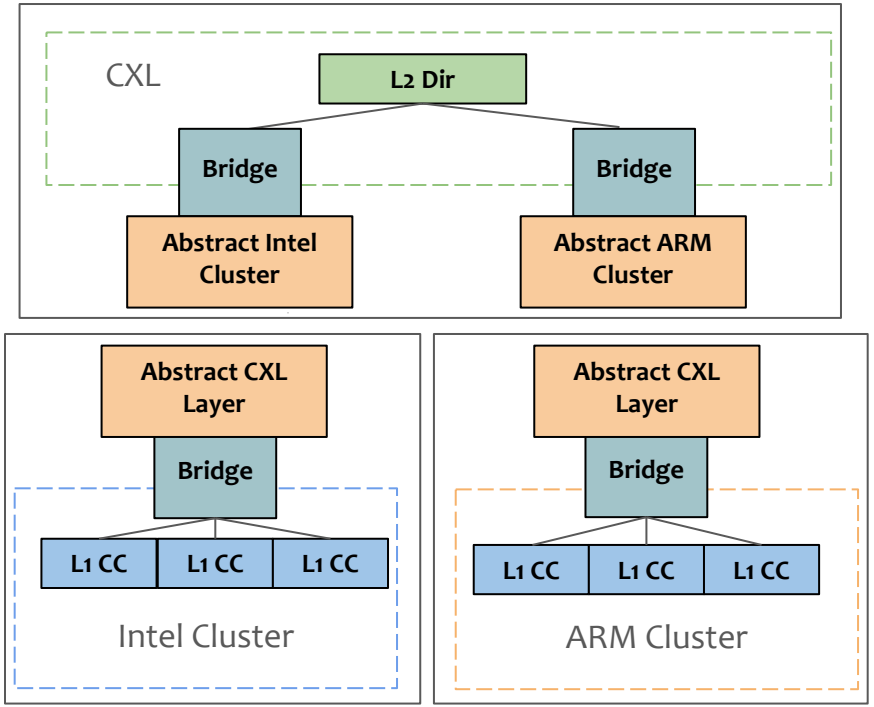
How can we **reduce the resources** required for verifying heterogeneous cache coherence protocols using **model checking**?

Solution: Apply **compositional** model checking

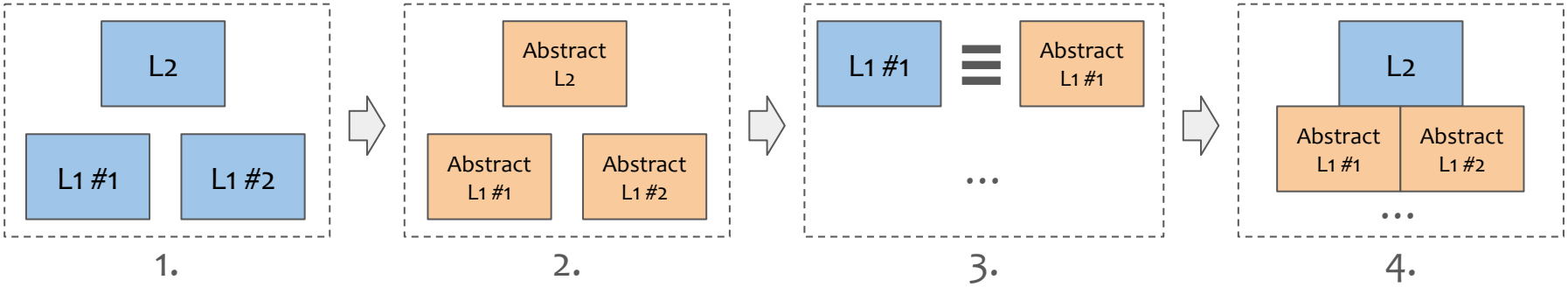
Compositional model checking

- Decompose the full system model.
- Verify desired properties for each partial model.
- **Challenge:** Behaviour of partial models depends on interaction with the remaining system.

  ⇒ Mock remaining system using abstractions with smaller state space



Overview: Compositional verification for general HCCP



1. Decompose system into individual clusters.
2. Generate abstractions with less state space, for each cluster.
3. Use an equivalence check to justify substituting a full cluster with it's abstraction.
4. Verify each cluster by replacing others with their abstractions.

Step 2: Abstraction generation

- **Goals:**
  - Utilize less state space
  - Behave equivalently from outside perspective

- Generated automatically using one or more reduction methods:
  - hiding internal message communication
  - predefined order for components
  - adding atomicity for caches/network

- Derived from operation of the bridge:
  - HieraGen bridges should behave like a cache controller on the higher level

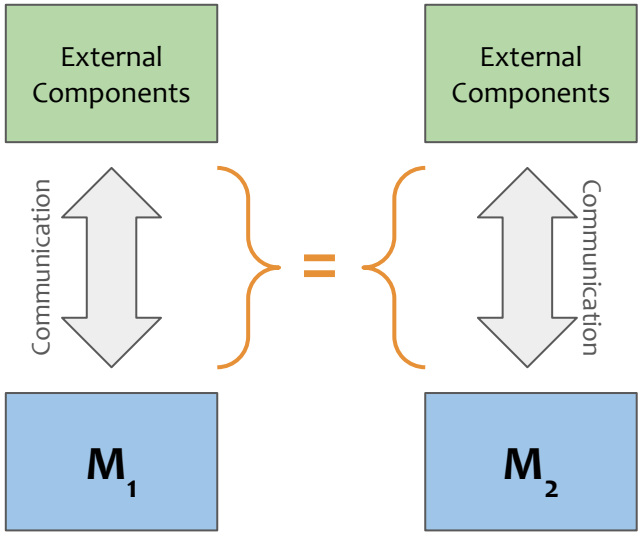
⇒ Use abstraction which **succeeds the equivalence check** & has the **largest state space reduction**

Step 3: Equivalence Check (1/2)

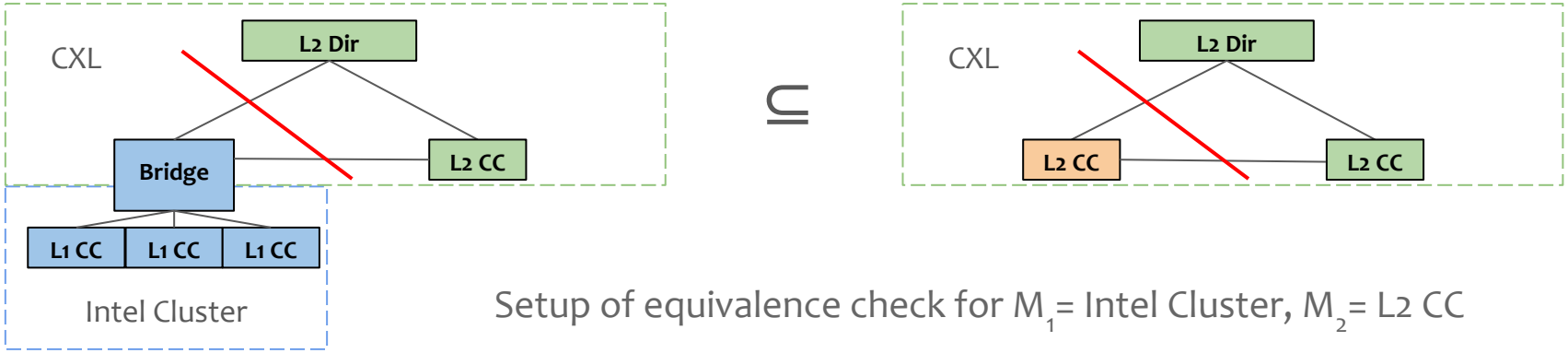
- **Goal:** verify equivalence in behaviour of any two models (M_1, M_2)

- Split into two checks:
  - M_1 can reproduce all behaviours of M_2
  - M_2 can reproduce all behaviours of M_1

- Each check is verified using a model containing:
  - both M_1 and M_2
  - the external components they interface with
  - a comparator for their communication
  - logic for searching a reproduction path



Step 3: Equivalence Check (2/2)

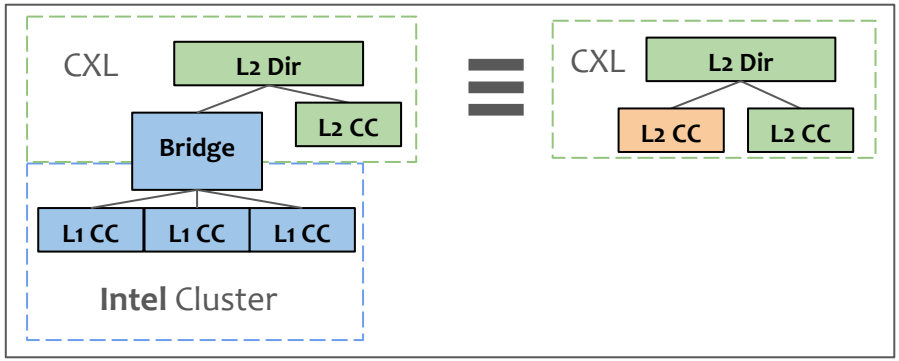


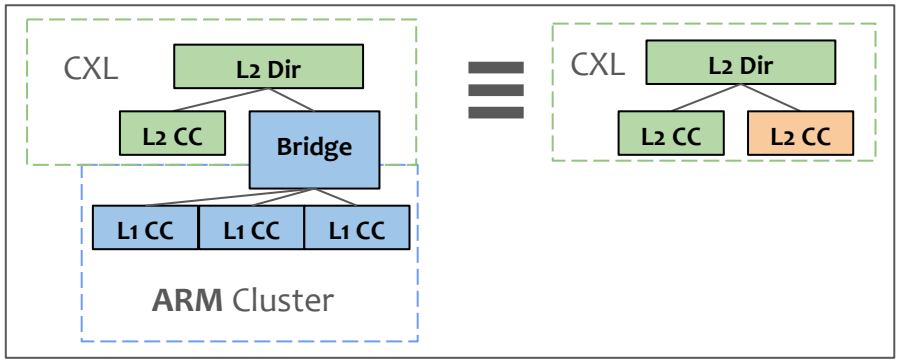
Setup of equivalence check for M_1= Intel Cluster, M_2= L2 CC

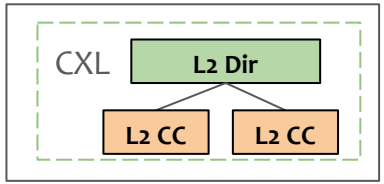
1. Explore left hand side (LHS) and monitor the L2 queues of the **bridge**
2. Queue changes cause a switch to explore the right hand side (RHS)
3. Explore the RHS until the **L2 CC** can reproduce the queue state
4. If the queue state is equal switch exploration back to LHS (step 1)

The check succeeds if there is always **some** way to switch state exploration back to the LHS.

Step 4: Substitution by abstractions







If **both** the Intel and ARM cluster are equivalent to an L2 CC

⇒ The L2 cluster with two L2 CC has the same behaviour.

Allows verifying properties for a simpler system.

Evaluation: Problems of our baseline

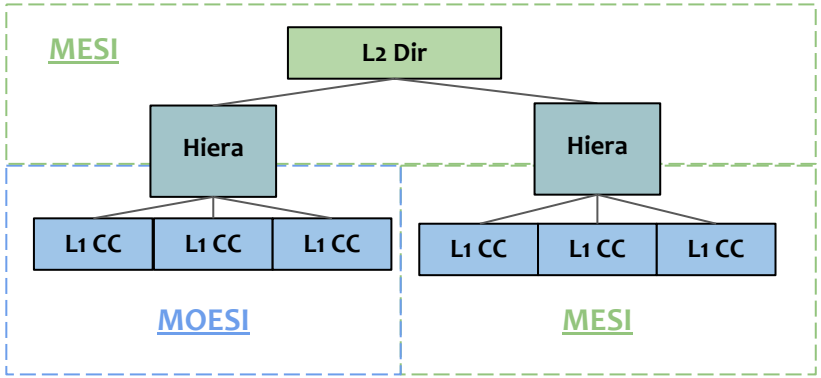
- The artifacts of the HieraGen paper have two problems:

- Deadlock:
  - Bridge generated by HieraGen deadlocks when having two or more L1 clusters
  - Unhandled request stalls the Bridge in certain states
  - **Fix for evaluation:** Allow bridge to change to state where it can handle request

- Optimization:
  - Models do not clean up unused variables ⇒ state space explosion
  - **Fix for evaluation:** Perform rudimentary clean up
  - The optimized version decreases required RAM by more than 99.86% for some models

Evaluation: Setup

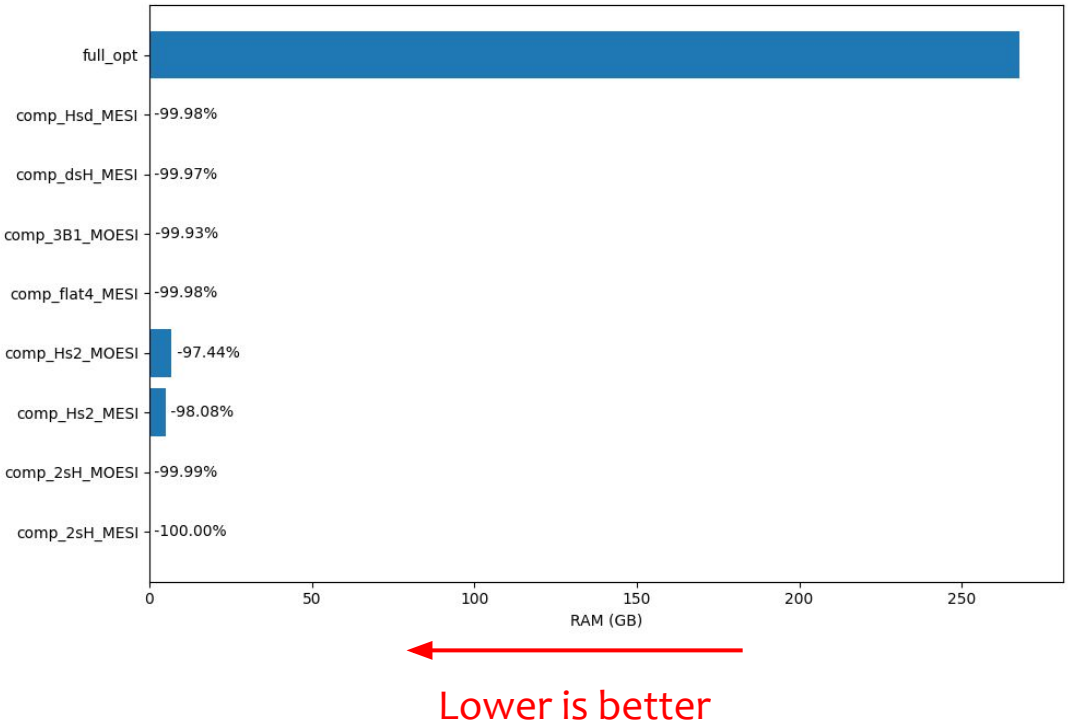
- Verify liveness for a target system with:
  - Two L1 clusters (MOESI & MESI)
  - Three cache controllers per L1
  - Using the fixed HieraGen bridge



- Baseline:
  - optimized full system HieraGen model
  - verification using Murφ

- Compositional Setup:
  - included all models & equivalence checks
  - verification using **Rumur** for equivalence checks and else Murφ

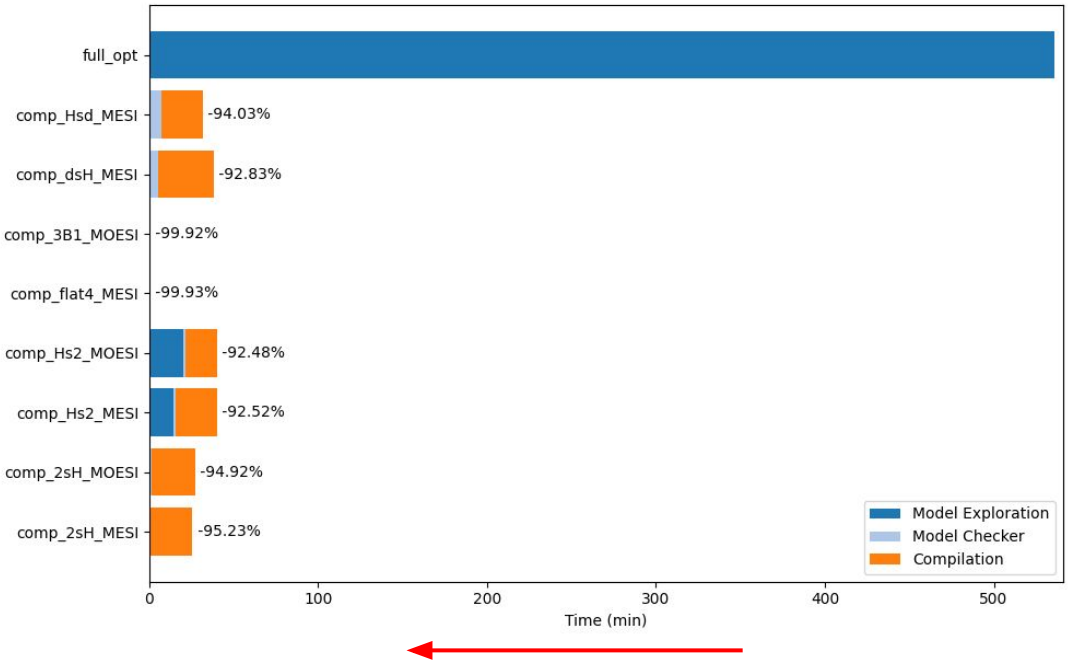
Evaluation: RAM usage for liveness verification



Baseline (*full_opt*) requires **268 GB** to verify the full optimized model.

All compositional checks (*comp_**) require at most **7 GB** individually.

Evaluation: Time required for liveness verification

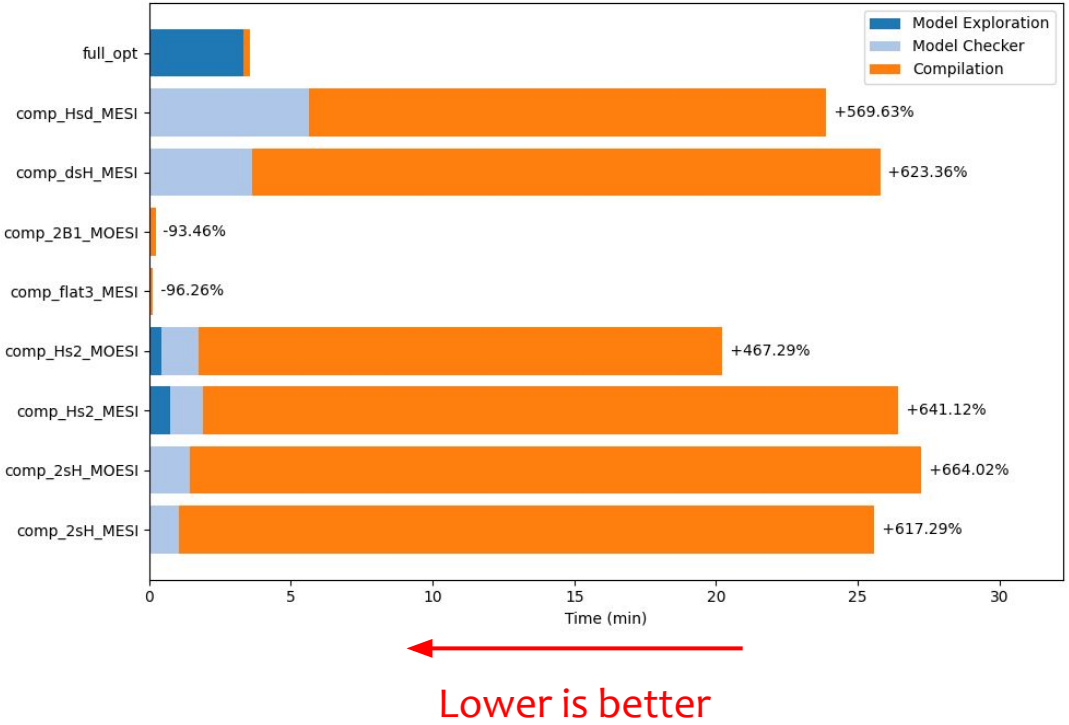


Lower is better

Baseline (*full_opt*) requires **8 h 54 min** to verify the full optimized model.

All compositional checks (*comp_**) require at most **41 min** individually.

Evaluation: Time required for liveness verification



Results when using a target system with **only two CC** in each L1 cluster.

⇒ It is no longer faster to perform a compositional verification.

Conclusion

- Verifying large cache coherence systems using conventional model checking is infeasible on modern machines.

- Compositional model checking:
  - allows to reduce the required RAM to analyze our target system by **more than 97%.**
  - decreases total time required to verify larger systems
  - verification time overhead for verifying smaller systems

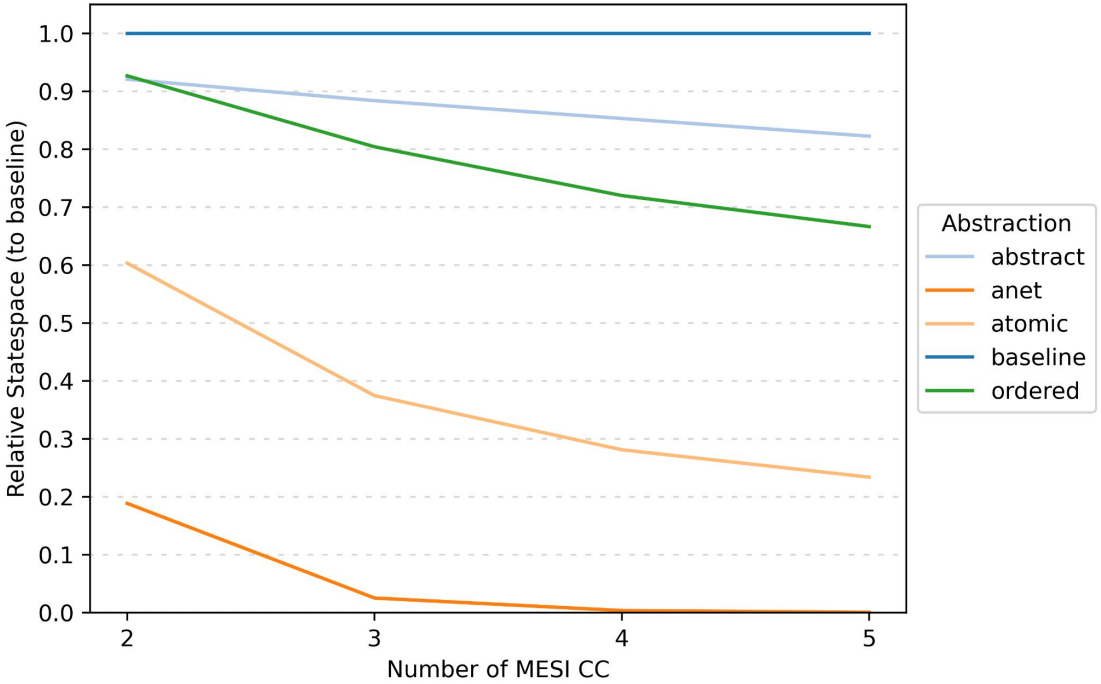
Compositional model checking allows to verify larger cache coherence systems on modern machines.

Backup

Backup: Abstraction Comparison

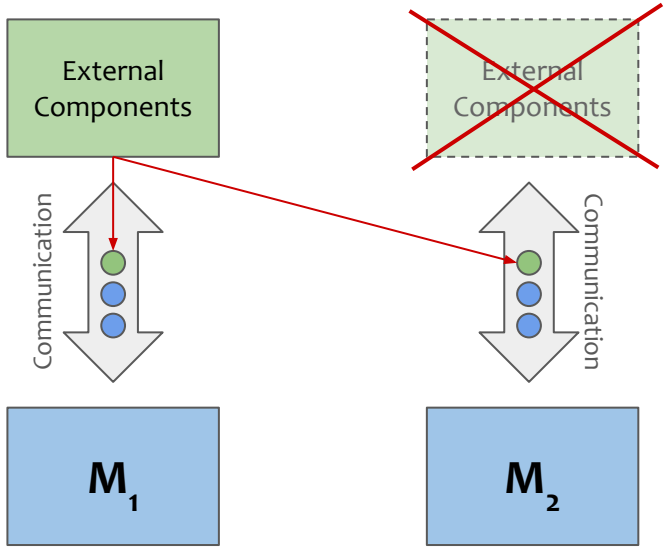
- Reduction increase with larger systems

  ⇒ likely different asymptotic growth



Backup: Equivalence Check Optimizations (1/2)

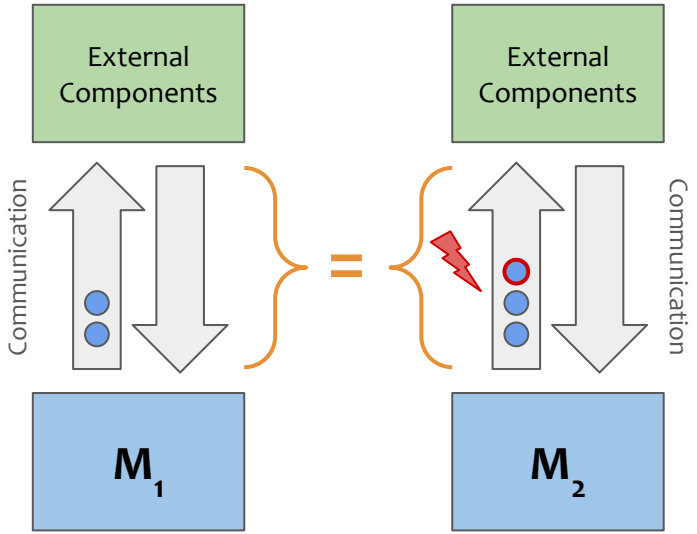
- Redundant external components:
  - Only model external components once
  - Replicate operations of external component to both LHS and RHS
  - External components only operate on equivalent queues, else the RHS search is active



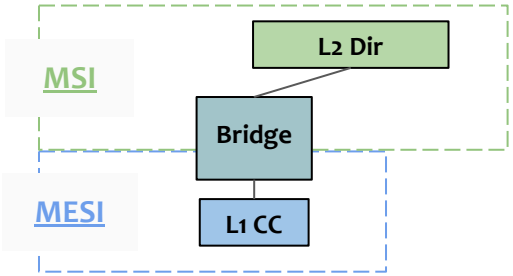
20

Backup: Equivalence Check Optimizations (2/2)

- No RHS queue rollback
  - Network queues are directional ⇒during RHS exploration it can be determined If a queue is only read or written to
  - Restrict RHS exploration once queue is in an unfixable state
  - Restricting RHS is valid as we only search for one replicating trace



Backup: HieraGen Deadlock



1. L1 CC → Hiera: getS
2. Hiera → L2 Dir: getS
3. L2 Dir → Hiera: getS_Ack
4. Hiera → L1 CC: getS_Ack
5. L1 CC → Hiera: putS
6. Hiera → L1 CC: putAck
7. L1 CC → Hiera: getS

Hiera is stuck in **I_x_S** state and can not handle the second getS:

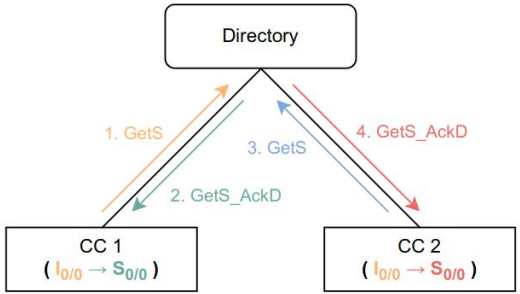
The only ways for the Bridge to leave the I_x_S state:

1. when receiving: Inv
2. On evict
3. On store

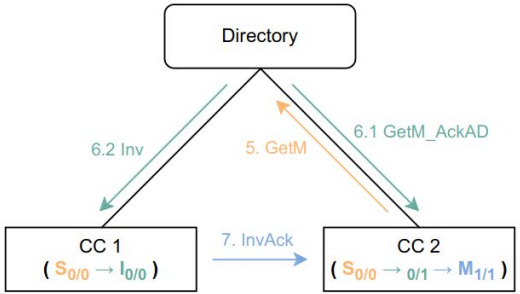
This deadlock does not exist with an L2 CC, as it can send an Inv to the bridge

⇒ Only caught if full system is taken into account

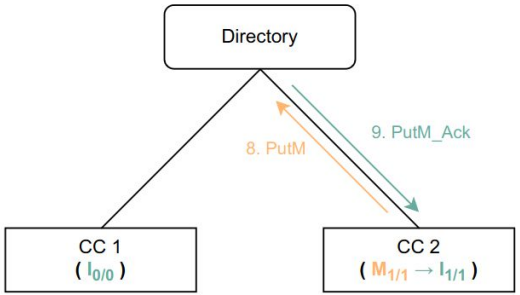
Backup: HieraGen Optimizations



(a) Both caches request and receive shared access to a cache line.

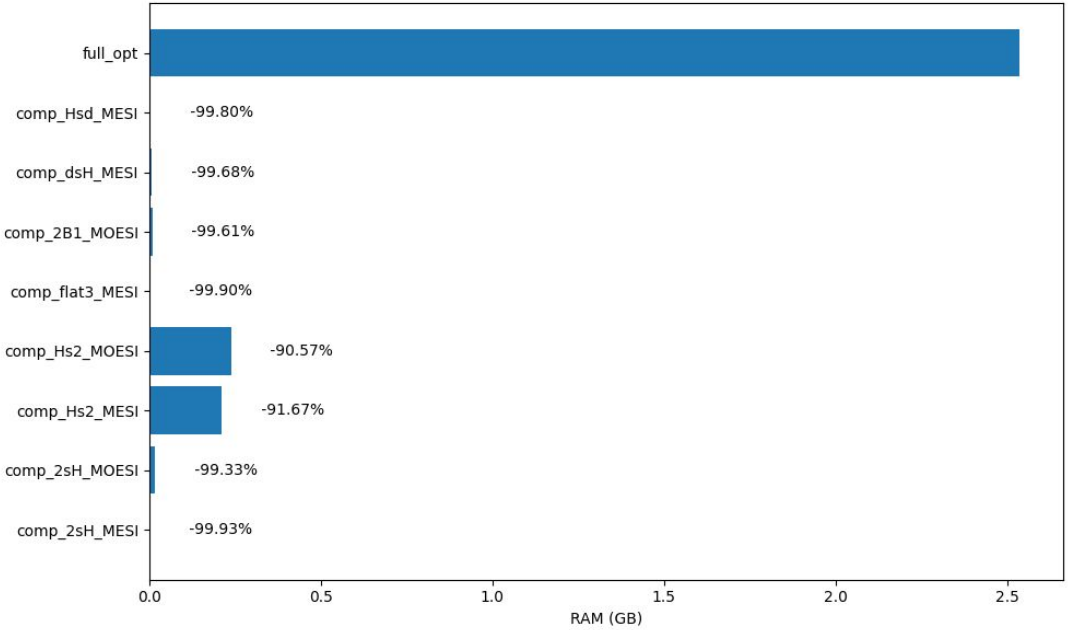


(b) The second cache upgrades to write access, invalidating the other cache.



(c) After evicting the cache line, the internal state of the second cache still retains the acknowledgement information.

Backup: RAM Usage for liveness verification



Results when using a target system with **only two CC** in each L1 cluster.

⇒ Similar but smaller improvements