



CLoF: A Compositional Lock Framework for Multi-level NUMA Systems

Rafael Chehab



Antonio Paolillo



Diogo Behrens



Ming Fu



Hermann Härtig



Haibo Chen



October 29, 2021

Concurrency is everywhere

Modern operating systems,
databases & applications resort to
multi-core concurrency to
achieve high performance.



MariaDB



PostgreSQL

Concurrency is everywhere

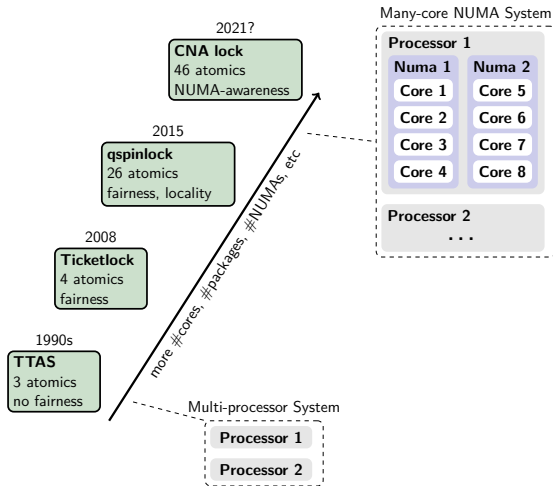
Modern operating systems, databases & applications resort to **multi-core concurrency** to achieve high performance.



MariaDB

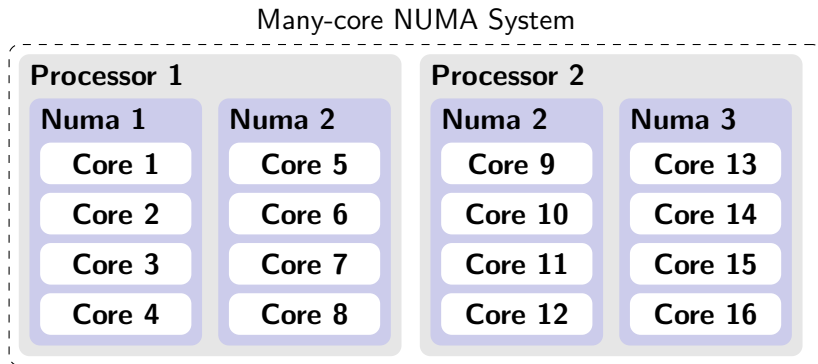


Linux spinlock evolution



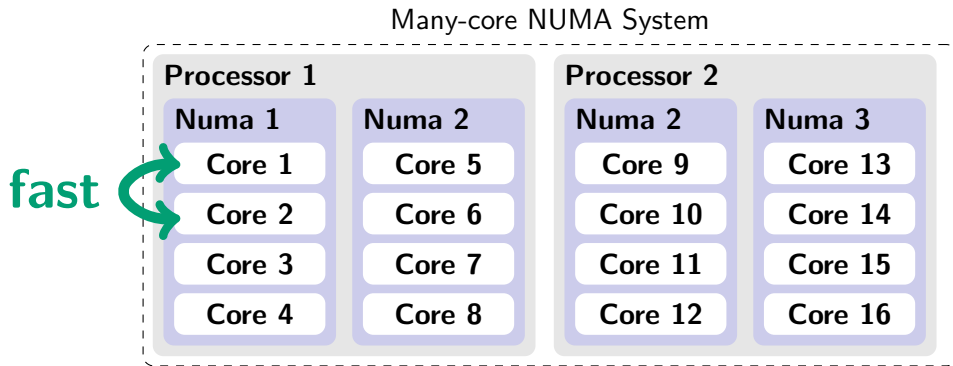
Challenge: exploiting the deep hierarchy of large NUMA machines

Core distance affects shared-memory communication performance



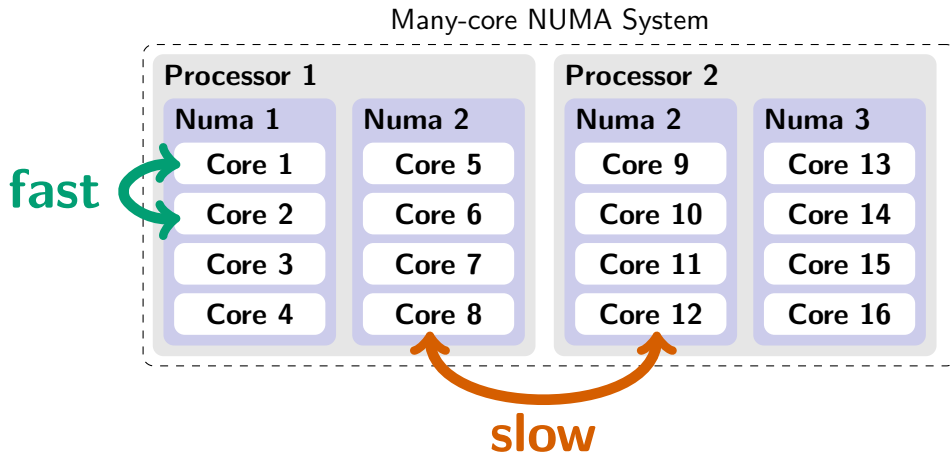
Challenge: exploiting the deep hierarchy of large NUMA machines

Core distance affects shared-memory communication performance



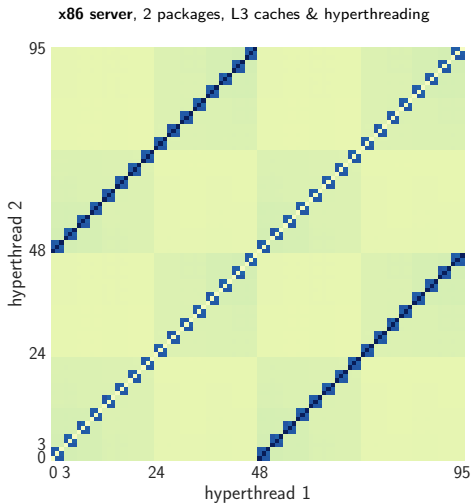
Challenge: exploiting the deep hierarchy of large NUMA machines

Core distance affects shared-memory communication performance



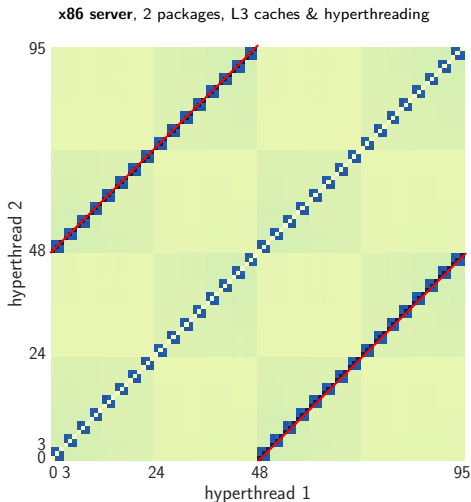
Challenge: exploiting the deep hierarchy of large NUMA machines

Discovering the hierarchy with a pair of threads incrementing a shared counter



Challenge: exploiting the deep hierarchy of large NUMA machines

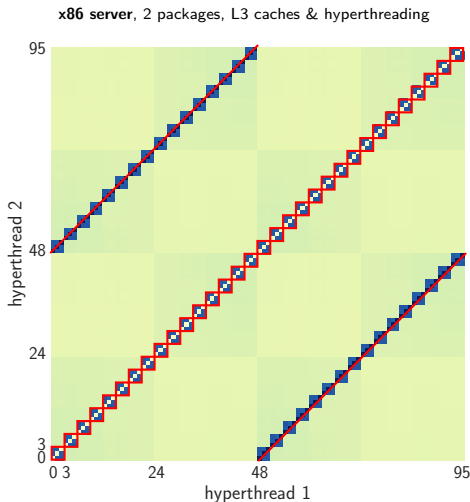
Discovering the hierarchy with a pair of threads incrementing a shared counter



- ▶ The full hierarchy
 - ▶ hyperthreading – secondary diagonals

Challenge: exploiting the deep hierarchy of large NUMA machines

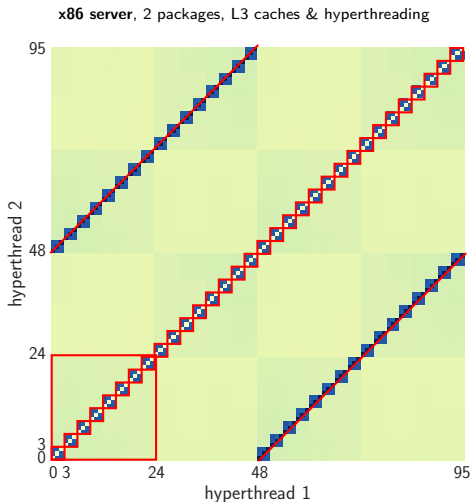
Discovering the hierarchy with a pair of threads incrementing a shared counter



- ▶ The full hierarchy
 - ▶ hyperthreading – secondary diagonals
 - ▶ L3 cache partitions – 3×3 squares

Challenge: exploiting the deep hierarchy of large NUMA machines

Discovering the hierarchy with a pair of threads incrementing a shared counter

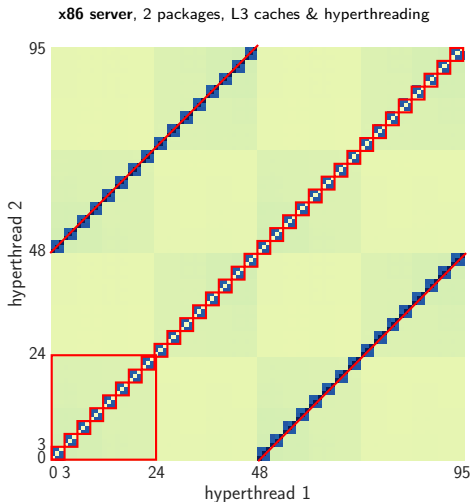


► The full hierarchy

- hyperthreading – secondary diagonals
- L3 cache partitions – 3×3 squares
- NUMA nodes / packages – 24×24 squares

Challenge: exploiting the deep hierarchy of large NUMA machines

HMCS performs better when it sees the deep hierarchy

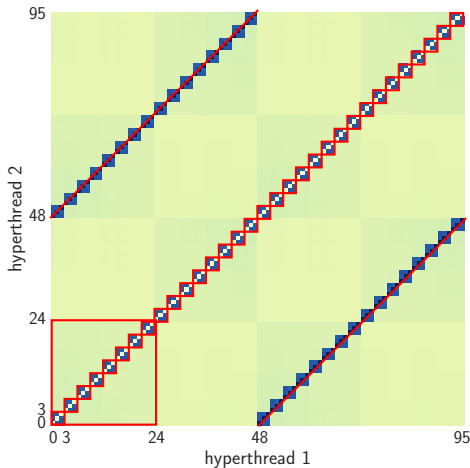


- ▶ The full hierarchy
 - ▶ hyperthreading – secondary diagonals
 - ▶ L3 cache partitions – 3×3 squares
 - ▶ NUMA nodes / packages – 24×24 squares
- ▶ HMCS lock can exploit the hierarchy
 - ▶ HMCS<2>, HMCS<3>: partial hierarchy
 - ▶ HMCS<4>: **full hierarchy**

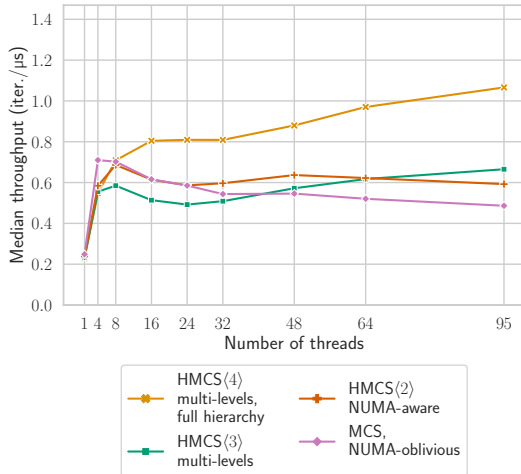
Challenge: exploiting the deep hierarchy of large NUMA machines

HMCS performs better when it sees the deep hierarchy

x86 server, 2 packages, L3 caches & hyperthreading



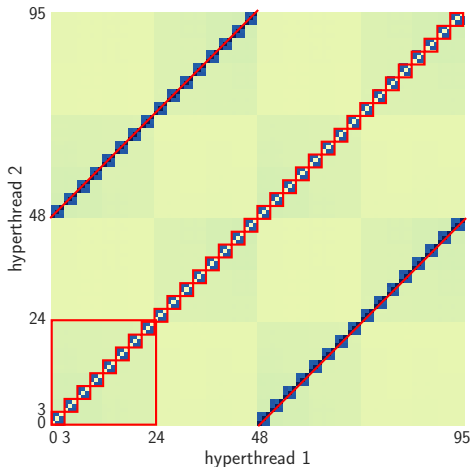
LevelDB readrandom benchmark, x86 server



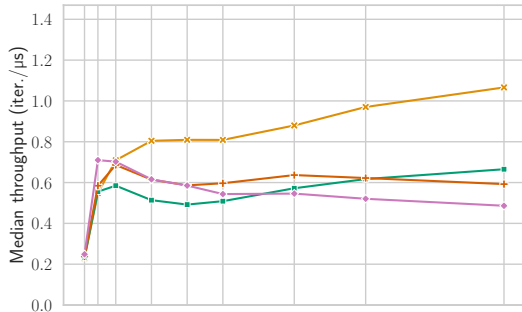
Challenge: exploiting the deep hierarchy of large NUMA machines

HMCS performs better when it sees the deep hierarchy

x86 server, 2 packages, L3 caches & hyperthreading



LevelDB readrandom benchmark, x86 server



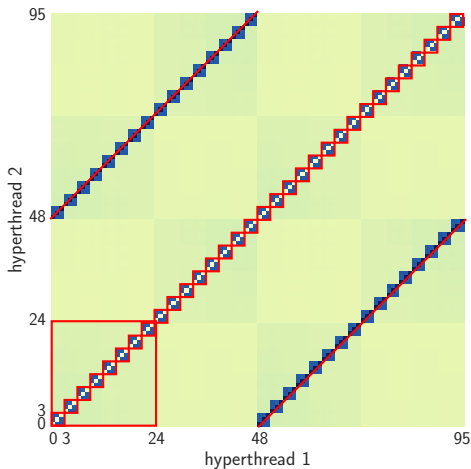
Multi-Level:

Encoding the actual deep hierarchy in a multi-level lock maximizes locality, thus improves performance.

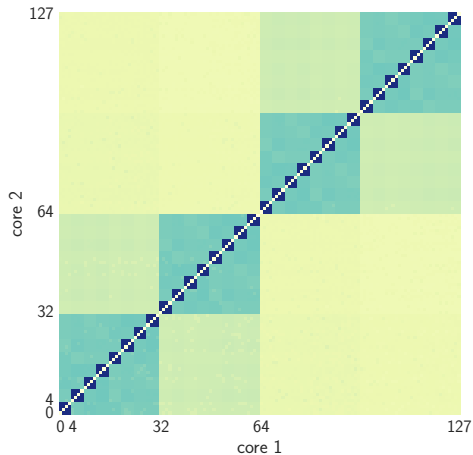
Challenge: exploiting the deep hierarchy of large NUMA machines

Different platforms may have different deep hierarchies

x86 server, 2 packages, L3 caches & hyperthreading

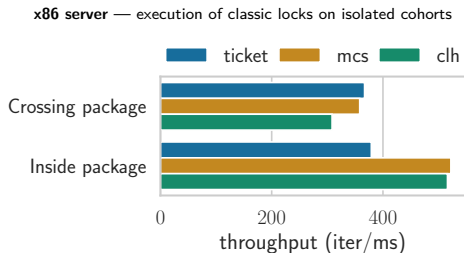


Arm server, 2 packages, 4 NUMA nodes, cache tagging



Challenge: locks perform differently according to scheduling & architecture

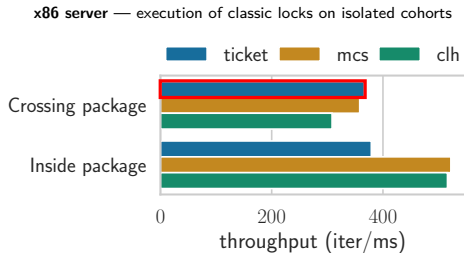
Comparing locks across different cohorts



Different levels may have better performance with different locks.

Challenge: locks perform differently according to scheduling & architecture

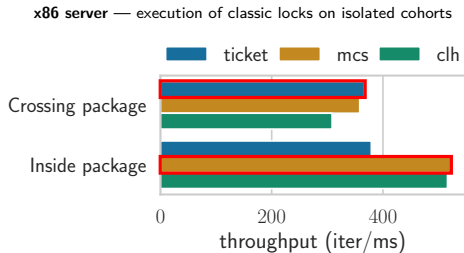
Comparing locks across different cohorts



Different levels may have better performance with different locks.

Challenge: locks perform differently according to scheduling & architecture

Comparing locks across different cohorts

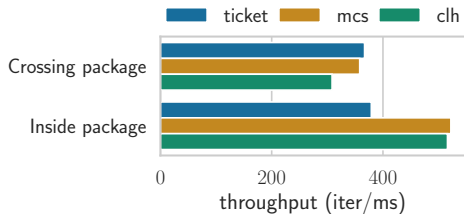


Different levels may have better performance with different locks.

Challenge: locks perform differently according to scheduling & architecture

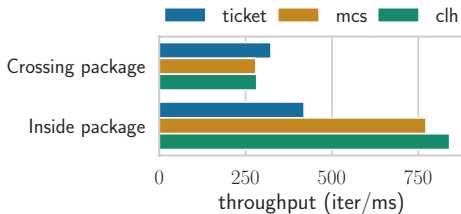
Comparing locks across different architectures

x86 server — execution of classic locks on isolated cohorts



Different levels may have better performance with different locks.

Arm server — execution of classic locks on isolated cohorts

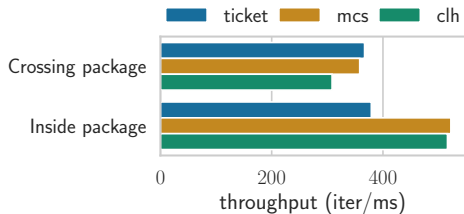


Different architectures/platforms may have better performance with different locks.

Challenge: locks perform differently according to scheduling & architecture

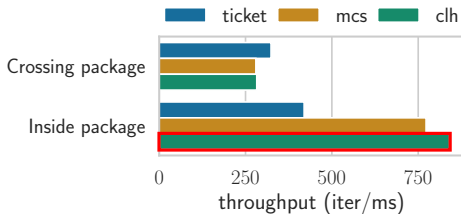
Comparing locks across different architectures

x86 server — execution of classic locks on isolated cohorts



Different levels may have better performance with different locks.

Arm server — execution of classic locks on isolated cohorts

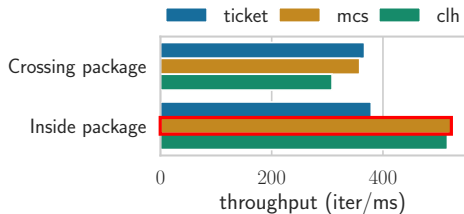


Different architectures/platforms may have better performance with different locks.

Challenge: locks perform differently according to scheduling & architecture

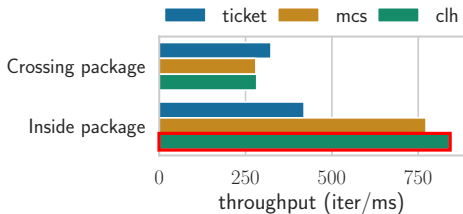
Comparing locks across different architectures

x86 server — execution of classic locks on isolated cohorts



Different levels may have better performance with different locks.

Arm server — execution of classic locks on isolated cohorts



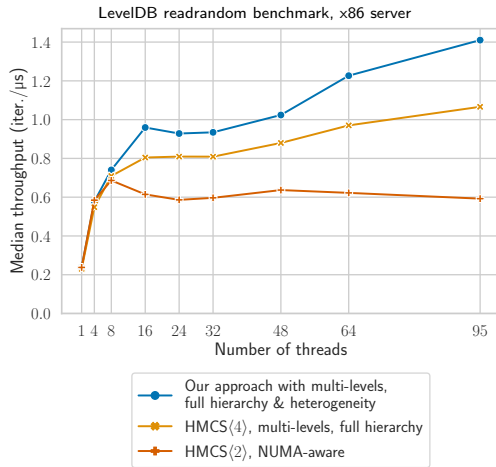
Different architectures/platforms may have better performance with different locks.

Challenge: locks perform differently according to scheduling & architecture

Potential performance benefits of heterogeneity

Heterogeneity:

Locks perform differently according to levels, architectures & platforms.



Challenge: verifying complex locks

We cannot model check the full hierarchy on WMMs

- ▶ Showing lock correctness is challenging — especially on different memory models

Challenge: verifying complex locks

We cannot model check the full hierarchy on WMMs

- ▶ Showing lock correctness is challenging — especially on different memory models
- ▶ **Weak Memory Models (WMMs)** allow reorderings for optimization
 - ▶ Order of operations in concurrent code cannot be compromised
 - ▶ Barriers must be used carefully to guarantee correct & efficient code

Challenge: verifying complex locks

We cannot model check the full hierarchy on WMMs

- ▶ Showing lock correctness is challenging — especially on different memory models
- ▶ **Weak Memory Models (WMMs)** allow reorderings for optimization
 - ▶ Order of operations in concurrent code cannot be compromised
 - ▶ Barriers must be used carefully to guarantee correct & efficient code
- ▶ Model checkers – e.g., GenMC – can verify correctness of simple locks. . .
But too slow for a large multi-level lock

Challenge: verifying complex locks

We cannot model check the full hierarchy on WMMs

- ▶ Showing lock correctness is challenging — especially on different memory models
- ▶ **Weak Memory Models (WMMs)** allow reorderings for optimization
 - ▶ Order of operations in concurrent code cannot be compromised
 - ▶ Barriers must be used carefully to guarantee correct & efficient code
- ▶ Model checkers – e.g., GenMC – can verify correctness of simple locks. . .
But too slow for a large multi-level lock

Correctness on WMMs:

Lock correctness is critical but verifying it is very expensive.

Our contributions: CLoF

A Compositional Lock Framework for Multi-level NUMA Systems

We propose CLoF, a framework to generate locks:

Our contributions: CLoF

A Compositional Lock Framework for Multi-level NUMA Systems

We propose CLoF, a framework to generate locks:

- ▶ that support an arbitrary hierarchy with *multiple levels*;

Our contributions: CLoF

A Compositional Lock Framework for Multi-level NUMA Systems

We propose CLoF, a framework to generate locks:

- ▶ that support an arbitrary hierarchy with *multiple levels*;
- ▶ in each level, the lock implementation may be *different*, levels are heterogeneous;

Our contributions: CLoF

A Compositional Lock Framework for Multi-level NUMA Systems

We propose CLoF, a framework to generate locks:

- ▶ that support an arbitrary hierarchy with *multiple levels*;
- ▶ in each level, the lock implementation may be *different*, levels are heterogeneous;
- ▶ the locks are *correct-by-construction* on Weak Memory Models.

Our contributions: CLoF

A Compositional Lock Framework for Multi-level NUMA Systems

We propose CLoF, a framework to generate locks:

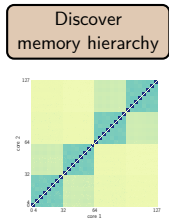
- ▶ that support an arbitrary hierarchy with *multiple levels*;
- ▶ in each level, the lock implementation may be *different*, levels are heterogeneous;
- ▶ the locks are *correct-by-construction* on Weak Memory Models.

NUMA-aware locks		Correctness on WMMs	Heterogeneity	Multi-Level
lock cohorting	PPoPP'12	✗	✓	✗
HMCS	PPoPP'15	✗ ¹	✗	✓
CNA lock	EuroSys'19	✗	✗	✗
ShflLock	SOSP'19	✗	✗	✗
CLoF	SOSP'21	✓	✓	✓

¹Insufficient barriers, fixed in Oberhauser *et al.*, Verifying and Optimizing the HMCS Lock for Arm Servers, NETYS'2021.

The CLoF workflow

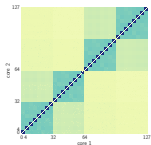
A user's perspective



The CLoF workflow

A user's perspective

Discover
memory hierarchy

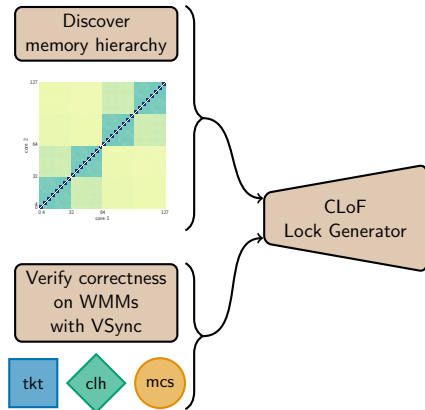


Verify correctness
on WMMs
with VSync



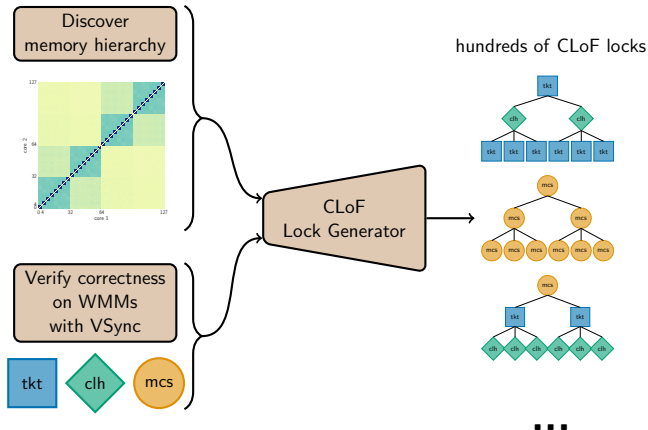
The CLoF workflow

A user's perspective



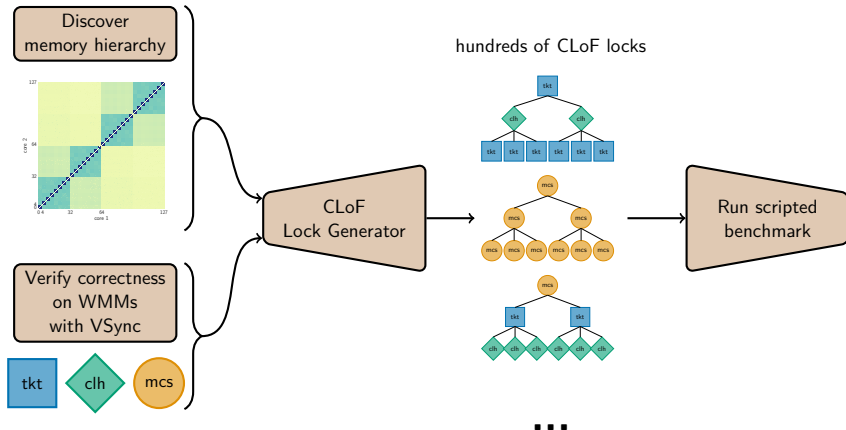
The CLoF workflow

A user's perspective



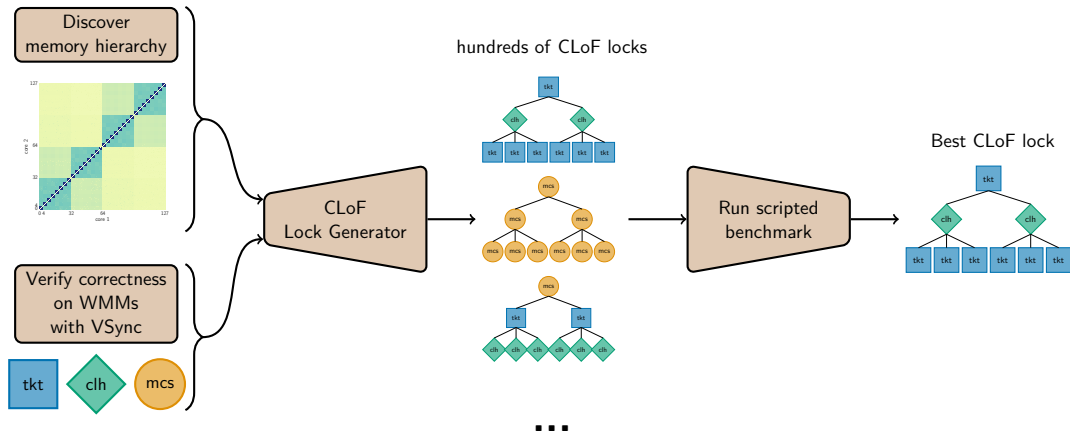
The CLoF workflow

A user's perspective



The CLoF workflow

A user's perspective



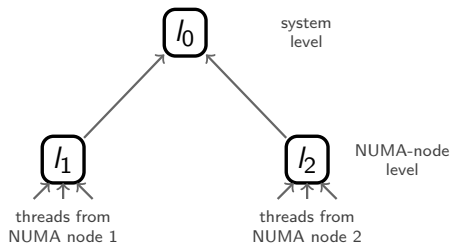
Composing CLoF locks

Two NUMA-node example



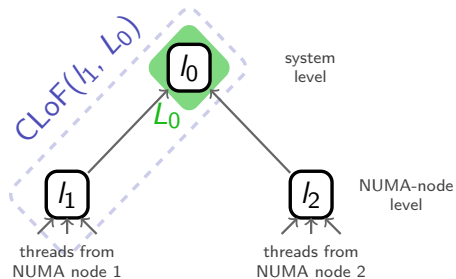
Composing CLoF locks

Two NUMA-node example



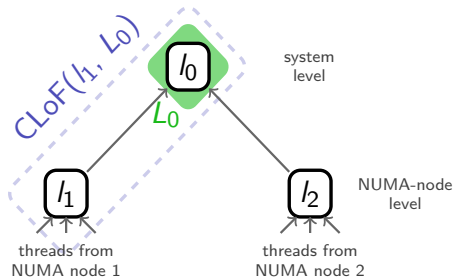
Composing CLoF locks

Two NUMA-node example



Composing CLoF locks

Two NUMA-node example



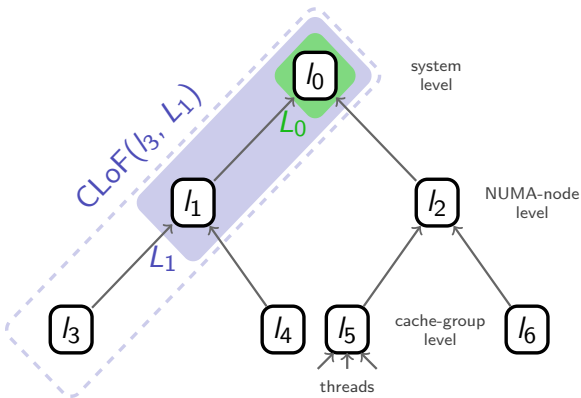
Simplified algorithm

```
CLoF( $I, L$ )::acquire =  
  acquire  $I$ ;  
  if ( $\neg$ already has  $L$ )  
    acquire  $L$ ;
```

```
CLoF( $I, L$ )::release =  
  if (others won't starve)  
    release  $I$ ;  
  else  
    release  $L$ ;  
    release  $I$ ;
```

Composing CLoF locks

Two NUMA-node example



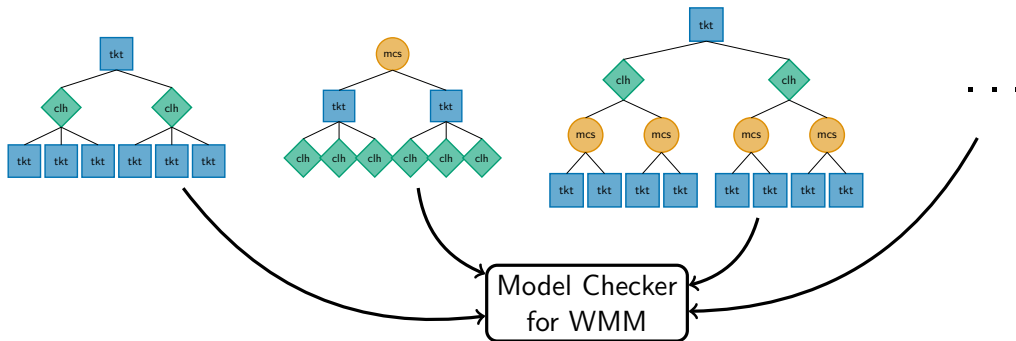
Simplified algorithm

```
CLoF( $l$ ,  $L$ )::acquire =  
  acquire  $l$ ;  
  if ( $\neg$ already has  $L$ )  
    acquire  $L$ ;
```

```
CLoF( $l$ ,  $L$ )::release =  
  if (others won't starve)  
    release  $l$ ;  
  else  
    release  $L$ ;  
    release  $l$ ;
```

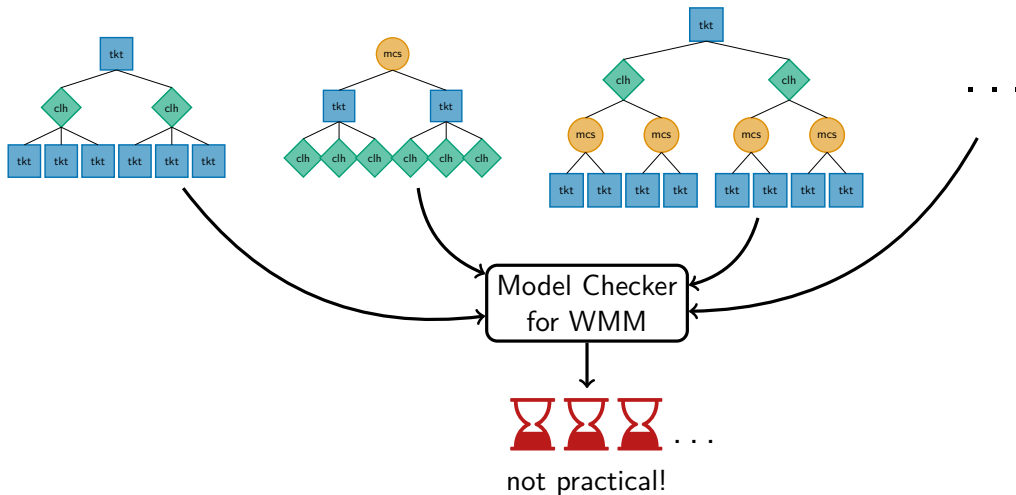
CLoF correctness

Can we model check the CLoF locks?



CLoF correctness

Can we model check the CLoF locks?



CLoF correctness

Combining induction argument with model checking

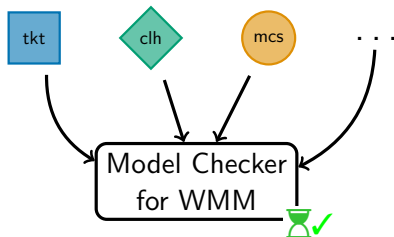
Base Step

Induction Step

CLoF correctness

Combining induction argument with model checking

Base Step

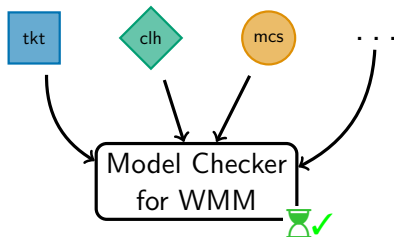


Induction Step

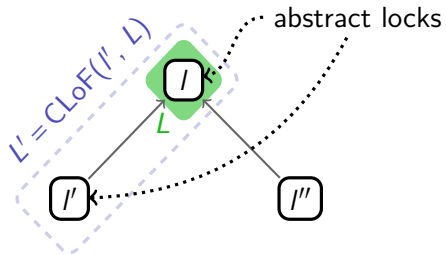
CLoF correctness

Combining induction argument with model checking

Base Step



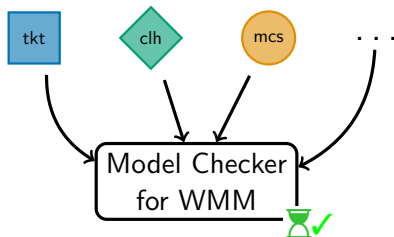
Induction Step



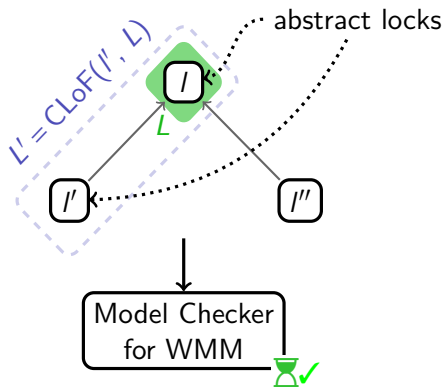
CLoF correctness

Combining induction argument with model checking

Base Step



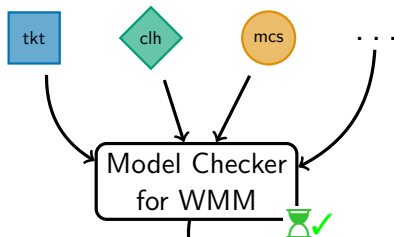
Induction Step



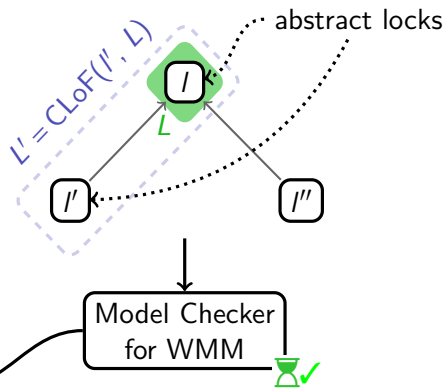
CLoF correctness

Combining induction argument with model checking

Base Step



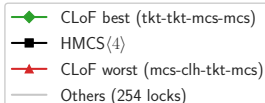
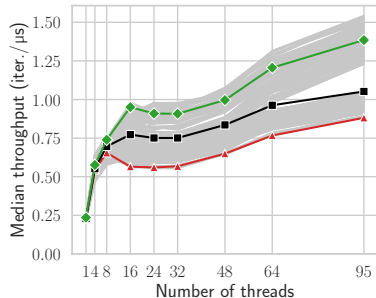
Induction Step



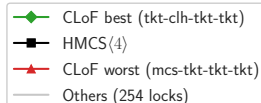
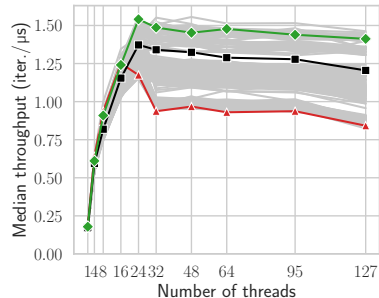
Select the best CLoF $\langle 4 \rangle$ lock

Generating/evaluating 256 CLoF locks and selecting the best

x86 server — LevelDB readrandom benchmark

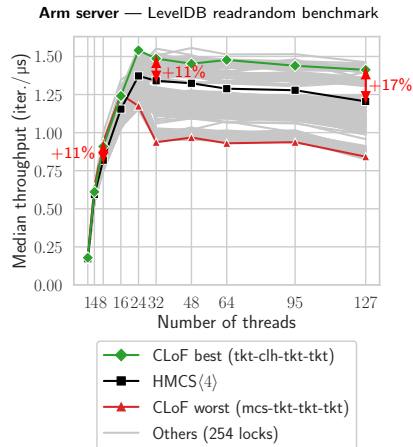
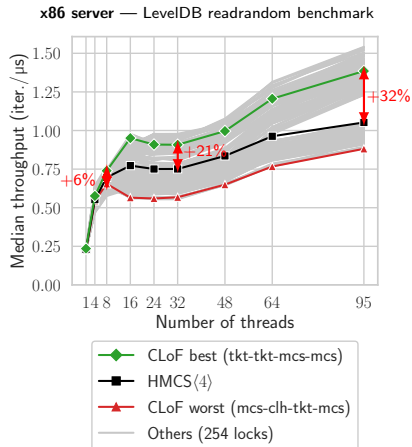


Arm server — LevelDB readrandom benchmark



Select the best CLoF $\langle 4 \rangle$ lock

Generating/evaluating 256 CLoF locks and selecting the best



Conclusion and future work

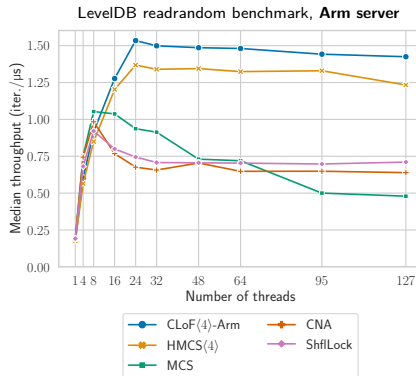
- ▶ **CLoF locks**

- ▶ fully leverage deep hierarchies and heterogeneity for good performance gains
- ▶ are correct-by-construction on Weak Memory Models

Conclusion and future work

► CLoF locks

- fully leverage deep hierarchies and heterogeneity for good performance gains
- are correct-by-construction on Weak Memory Models



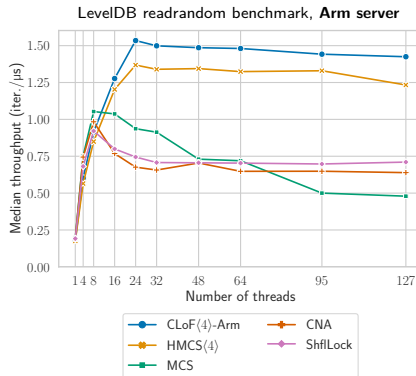
Conclusion and future work

► CLoF locks

- fully leverage deep hierarchies and heterogeneity for good performance gains
- are correct-by-construction on Weak Memory Models

► Don't miss the details!

- tuning points
- platform-specific optimizations
- analysis of lock combinations
- ...



Conclusion and future work

► CLoF locks

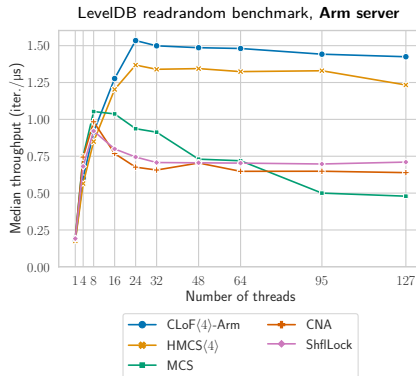
- fully leverage deep hierarchies and heterogeneity for good performance gains
- are correct-by-construction on Weak Memory Models

► Don't miss the details!

- tuning points
- platform-specific optimizations
- analysis of lock combinations
- ...

► Future work

- CLoF in the Linux kernel
- big.LITTLE platforms



Thank you

`antonio.paolillo@huawei.com`