CNCC

CNCC
China National
Computer Congress

后摩尔时代通过验证提升性能的工业实践

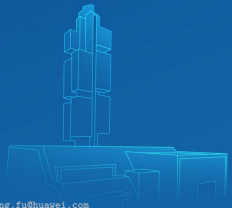
Practical Verification in Practice

Enabling Performance on Modern Hardware after Moore's Law

付明

Huawei Dresden Research Center, Germany

Huawei Central Software Institute, China



ming.fu@huawei.com

How to survive in the industry as a formal verification person?



The Verification Monks Tale

Once upon a time, **a great king** asked the **three verification monks**:
"How to **be sure** that my **folks follow all tax rules**?"



The Verification Monks Tale

Once upon a time, **a great king** asked the **three verification monks**:
"How to **be sure** that my **folks follow all tax rules**?"



Academicus

- **Simplifies the problem**
  → abstracts the kingdom with a model
- **Applies precise verification**
  → to prove payment correctness
- Shows: 100% taxes are payed

The Verification Monks Tale

Once upon a time, **a great king** asked the **three verification monks**:
"How to **be sure** that my **folks follow all tax rules**?"



Academicus

- **Simplifies the problem**
  → abstracts the kingdom with a model
- **Applies precise verification**
  → to prove payment correctness
- Shows: 100% taxes are payed
- **King does not like models!**

The Verification Monks Tale

Once upon a time, **a great king** asked the **three verification monks**:
"How to **be sure** that my **folks follow all tax rules**?"



Academicus

- **Simplifies the problem**
  → abstracts the kingdom with a model
- **Applies precise verification**
  → to prove payment correctness
- Shows: 100% taxes are payed
- **King does not like models!**



Practicus

- **Considers the real kingdom**
  → with all details and tax rules
- **Simplifies verification method**
  → incomplete, false positives
- Shows: 3% tax-collection loss

The Verification Monks Tale

Once upon a time, **a great king** asked the **three verification monks**:
"How to **be sure** that my **folks follow all tax rules**?"



Academicus

- **Simplifies the problem**
  → abstracts the kingdom with a model
- **Applies precise verification**
  → to prove payment correctness
- Shows: 100% taxes are payed
- **King does not like models!**



Practicus

- **Considers the real kingdom**
  → with all details and tax rules
- **Simplifies verification method**
  → incomplete, false positives
- Shows: 3% tax-collection loss
- **King is unhappy!**
  His tax collectors already find 2%!

ming.fu@huawei.com

The Verification Monks Tale

Once upon a time, **a great king** asked the **three verification monks**:
"How to **be sure** that my **folks follow all tax rules**?"



Academicus

- **Simplifies the problem**
  → abstracts the kingdom with a model
- **Applies precise verification**
  → to prove payment correctness
- Shows: 100% taxes are payed
- **King does not like models!**



Practicus

- **Considers the real kingdom**
  → with all details and tax rules
- **Simplifies verification method**
  → incomplete, false positives
- Shows: 3% tax-collection loss
- **King is unhappy!**
  His tax collectors already find 2%!



Cleverus

- **Proposes better tax rules**
  → more efficient, higher return
- **Adapts theory, creates tool**
  → not perfect, but easy to use
- Shows: 20% tax-collection increase
  → with high confidence that folk pays taxes

The Verification Monks Tale

Once upon a time, **a great king** asked the **three verification monks**:
"How to **be sure** that my **folks follow all tax rules**?"



Academicus

- **Simplifies the problem**
  → abstracts the kingdom with a model
- **Applies precise verification**
  → to prove payment correctness
- Shows: 100% taxes are payed
- **King does not like models!**



Practicus

- **Considers the real kingdom**
  → with all details and tax rules
- **Simplifies verification method**
  → incomplete, false positives
- Shows: 3% tax-collection loss
- **King is unhappy!**
  His tax collectors already find 2%!



Cleverus

- **Proposes better tax rules**
  → more efficient, higher return
- **Adapts theory, creates tool**
  → not perfect, but easy to use
- Shows: 20% tax-collection increase
  → with high confidence that folk pays taxes
- **King is extremely happy!**
  → better tax rules, more money
  → a tool for tax collectors

The Verification Monks Tale

Last month, **a great boss** asked the **three verification experts**:
"How to **be sure** that my **product is correct**?"



Academicus

Focus on theory

- **Simplifies the problem**
  → abstracts the program with a model
- **Applies precise verification**
  → to prove bug freedom
- Shows: 100% bug free
- **Boss does not like models!**



Practicus

Focus on problem

- **Considers the real program**
  → with all details
- **Simplifies verification method**
  → incomplete, false positives
- Shows: 3 bugs per MLoC
- **Boss is unhappy!**
  His developers already find 2!



Cleverus

Focus on business

- **Proposes better algorithm**
  → more efficient, higher performance
- **Adapts theory, creates tool**
  → not perfect, but easy to use
- Shows: 20% higher throughput
  → with high confidence that code is correct
- **Boss is extremely happy!**
  → better performance, more money
  → a tool for developers

Agenda

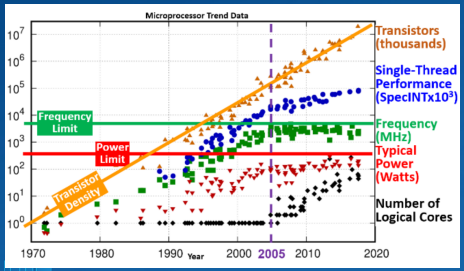
CNCC China National Computer Congress

- **Challenges in post Moore's Law Era**
- Enabling Performance with Practical Verification
- Practical Verification in Practice
  VSync, CNA, CLoF and BBQ
- Closing thoughts and outlook

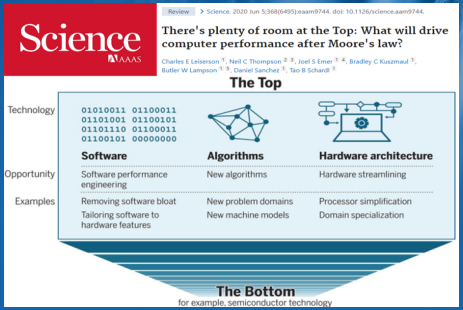


Free lunch is over

We are in a post Moore's Law era



[1]



[1] https://education.dellemc.com/content/dam/dell-emc/documents/en-us/2019KS_Yellin-Saving_The_Future_of_Moores_Law.pdf

Concurrency is everywhere

Operating systems, databases and server applications resort to **multicore concurrency** to achieve high performance.









Concurrency is everywhere

Operating systems, databases and server applications resort to **multicore concurrency** to achieve high performance.









. . . and brings problems!

- **Notoriously hard to get right**
  - Non-deterministic thread interleaving

Concurrency is everywhere

Operating systems, databases and server applications resort to **multicore concurrency** to achieve high performance.









. . . and brings problems!

- **Notoriously hard to get right**
  - Non-deterministic thread interleaving
- **Traditional testing isn't sufficient**
  - Testing is **unlikely** to trigger subtle concurrency bugs
  - Production is **likely** to trigger them, → many instances, long periods
- **Reproducing concurrency bugs is hard**

Concurrency is everywhere

Operating systems, databases and server applications resort to **multicore concurrency** to achieve high performance.









. . . and brings problems!

- **Notoriously hard to get right**
  - Non-deterministic thread interleaving
- **Traditional testing isn't sufficient**
  - Testing is **unlikely** to trigger subtle concurrency bugs
  - Production is **likely** to trigger them, → many instances, long periods
- **Reproducing concurrency bugs is hard**
- **Scalability isn't perfect!** $\rightsquigarrow 1+1<2$

Modern hardware makes matters worse

Relaxed memory models (RMMs)

- Modern architectures becoming popular
  eg, Arm, RISC-V



Modern hardware makes matters worse

Relaxed memory models (RMMs)

- Modern architectures becoming popular
  eg, Arm, RISC-V



Huawei Unveils Industry's Highest-Performance ARM-based CPU

Bringing Global Computing Power to Next Level

Jan 07, 2019

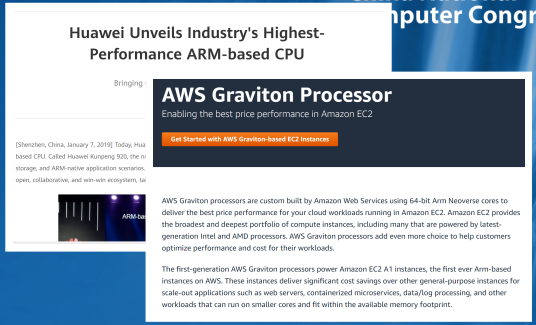
[Shenzhen, China, January 7, 2019] Today, Huawei announced the industry's highest-performance Advanced RISC Machine (ARM)-based CPU. Called Huawei Kunpeng 920, the new CPU is designed to boost the development of computing in big data, distributed storage, and ARM-native application scenarios. Huawei will join with industry players to advance the ARM industry and foster an open, collaborative, and win-win ecosystem, taking computing performance to new heights.



Modern hardware makes matters worse

Relaxed memory models (RMMs)

- Modern architectures becoming popular
  eg, Arm, RISC-V



Huawei Unveils Industry's Highest-Performance ARM-based CPU

Bringing

[Shenzhen, China, January 7, 2019] Today, Hua
based CPU. Called Huawei Kunpeng 920, the n
storage, and ARM-native application scenarios.
open, collaborative, and win-win ecosystem, tai

ARM-ba

AWS Graviton Processor

Enabling the best price performance in Amazon EC2

Get Started with AWS Graviton-based EC2 Instances

AWS Graviton processors are custom built by Amazon Web Services using 64-bit Arm Neoverse cores to deliver the best price performance for your cloud workloads running in Amazon EC2. Amazon EC2 provides the broadest and deepest portfolio of compute instances, including many that are powered by latest-generation Intel and AMD processors. AWS Graviton processors add even more choice to help customers optimize performance and cost for their workloads.

The first-generation AWS Graviton processors power Amazon EC2 A1 instances, the first ever Arm-based instances on AWS. These instances deliver significant cost savings over other general-purpose instances for scale-out applications such as web servers, containerized microservices, data/log processing, and other workloads that can run on smaller cores and fit within the available memory footprint.

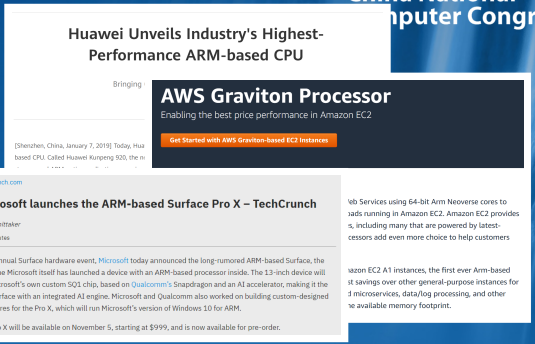


Modern hardware makes matters worse

Relaxed memory models (RMMs)

- Modern architectures becoming popular
  eg, Arm, RISC-V





Huawei Unveils Industry's Highest-Performance ARM-based CPU

Bringing

[Shenzhen, China, January 7, 2019] Today, Hua
based CPU. Called Huawei Kunpeng 920, the n

AWS Graviton Processor

Enabling the best price performance in Amazon EC2

Get Started with AWS Graviton-based EC2 Instances

techcrunch.com

Microsoft launches the ARM-based Surface Pro X – TechCrunch

Zack Whittaker

3-3 minutes

At its annual Surface hardware event, Microsoft today announced the long-rumored ARM-based Surface, the first time Microsoft itself has launched a device with an ARM-based processor inside. The 13-inch device will use Microsoft's own custom SQ1 chip, based on Qualcomm's Snapdragon and an AI accelerator, making it the first Surface with an integrated AI engine. Microsoft and Qualcomm also worked on building custom-designed GPU cores for the Pro X, which will run Microsoft's version of Windows 10 for ARM.

The Pro X will be available on November 5, starting at $999, and is now available for pre-order.

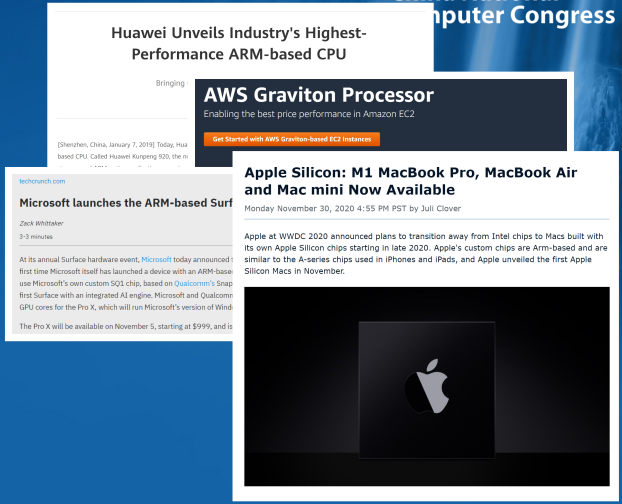
eb Services using 64-bit Arm Neoverse cores to
oads running in Amazon EC2. Amazon EC2 provides
s, including many that are powered by latest-
cessors add even more choice to help customers

azon EC2 A1 instances, the first ever Arm-based
st savings over other general-purpose instances for
l microservices, data/log processing, and other
e available memory footprint.

Modern hardware makes matters worse

Relaxed memory models (RMMs)

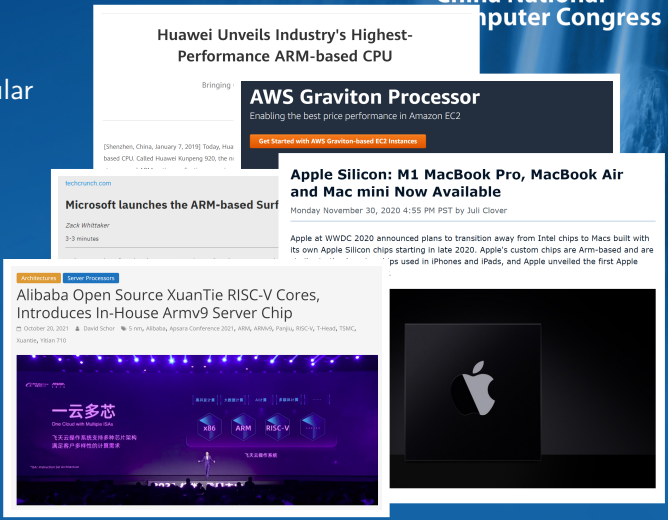
- Modern architectures becoming popular
  eg, Arm, RISC-V



Modern hardware makes matters worse

Relaxed memory models (RMMs)

- Modern architectures becoming popular
  eg, Arm, RISC-V



Modern hardware makes matters worse

Relaxed memory models (RMMs)

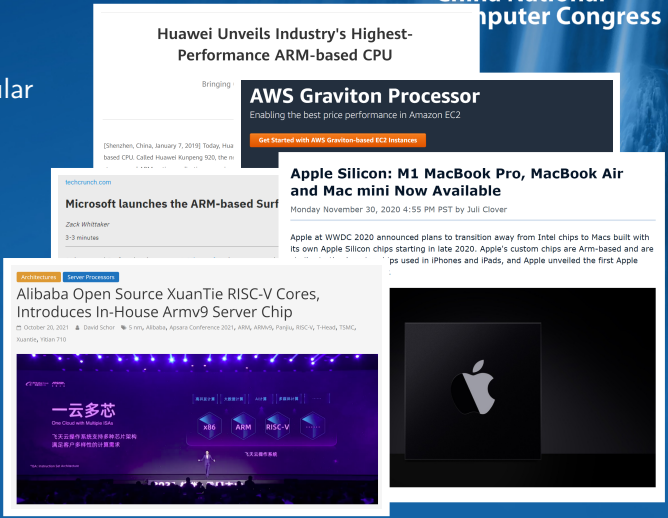
- Modern architectures becoming popular
  eg, Arm, RISC-V
- **Aggressive reorderings** to improve performance
- **Much higher non-determinisim**
- Careful use of **memory barriers**
  (neither too many, nor too few)

Init

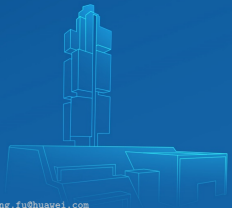
```
data = ctrl = 0;
```

Thread 1	Thread 2
`data = 1;`	`while(!ctrl) {}`
`ctrl = 1;`	`assert(data == 1);` ✗

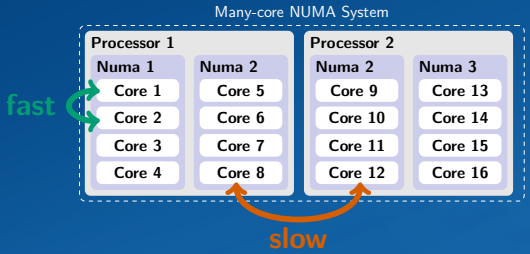
WMM reorder



Modern hardware makes matters worse



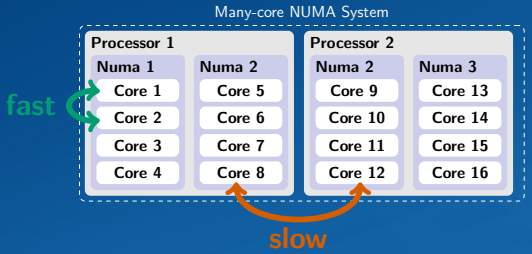
Modern hardware makes matters worse



Deep NUMA hierarchies

- Core **distance affects** shared-memory communication **performance**

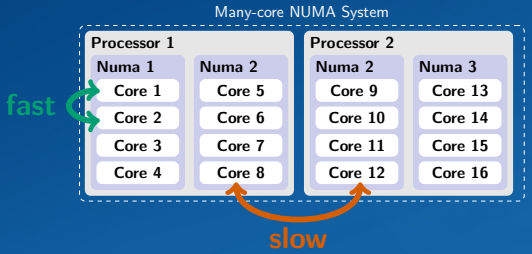
Modern hardware makes matters worse



Deep NUMA hierarchies

- Core **distance affects** shared-memory communication **performance**
- Concurrent algorithms must exploit that!

Modern hardware makes matters worse

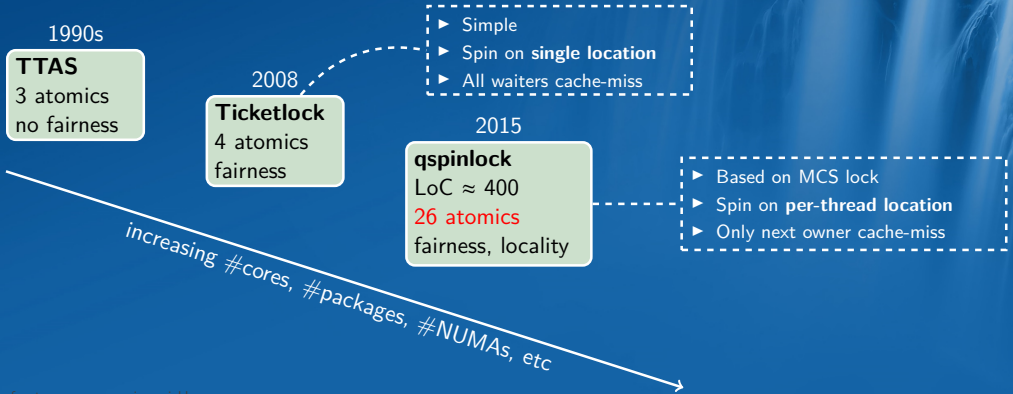


Deep NUMA hierarchies

- Core **distance affects** shared-memory communication **performance**
- Concurrent algorithms must exploit that!
- **Makes code even more complex**

Hardware pushes software evolution

Example of Linux spinlock over the years



"atomics" refers to racy accesses, ie, variables concurrently accessed by multiple cores

Hardware pushes software evolution

Example of Linux spinlock over the years

Version	Date	Correctness
Linux 4.4	2015/09/11	Not verified
Linux 4.5	2015/11/09	RMM bug (fixed in 4.16)
Linux 4.8	2016/06/03	RMM bug (fixed in 4.16)
Linux 4.16	2018/02/13	Not verified
Linux 5.6	2020/01/07	Not verified

1990s

TTAS
3 atomics
no fairness

2008

Ticketlock
4 atomics
fairness

2015

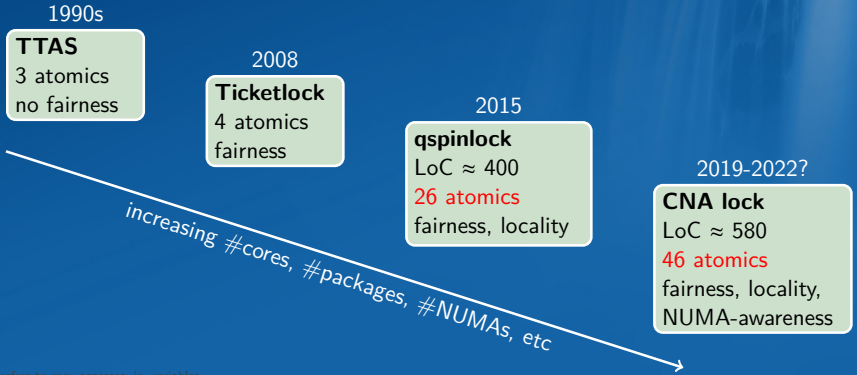
qspinlock
LoC ≈ 400
26 atomics
fairness, locality

increasing #cores, #packages, #NUMAs, etc

"atomics" refers to racy accesses, ie, variables concurrently accessed by multiple cores

Hardware pushes software evolution

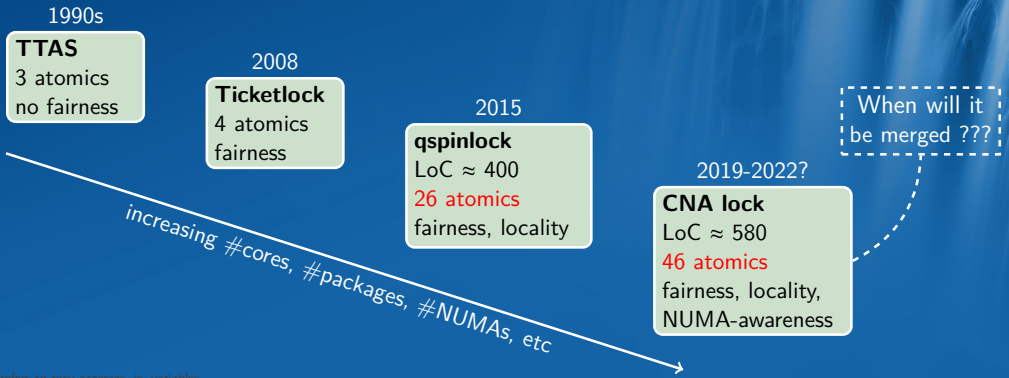
Example of Linux spinlock over the years



"atomics" refers to racy accesses, ie, variables concurrently accessed by multiple cores

Hardware pushes software evolution

Example of Linux spinlock over the years



"atomics" refers to racy accesses, ie, variables concurrently accessed by multiple cores

What can the industry do?



Solution 1: overprotection

- Spray the code with memory barriers
- Simplify design as much as possible
- Cost: **performance impact**



Solution 2: expert design

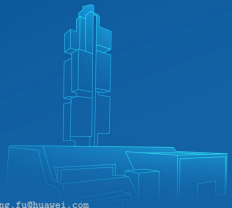
- Hire highly-skilled engineers
- Carefully design and implement
- Cost: **error-prone, low maintainability**

Agenda

- Challenges in post Moore's Law Era
- **Enabling Performance with Practical Verification**
- Practical Verification in Practice
  VSync, CNA, CLoF and BBQ
- Closing Thoughts and Outlook



What is **Practical** Verification?



What is **Practical** Verification?

Formal verification

- Goal: **show correctness**
- Examples:
  - interactive theorem proving
  - model checking

What is **Practical** Verification?

Formal verification

- Goal: **show correctness**
- Examples:
  - interactive theorem proving
  - model checking
- In contrast to testing, **exhaustive:**
  - All possible interleavings
  - All possible reorderings
  - All possible contexts, ...

What is **Practical** Verification?

Formal verification

- Goal: **show correctness**
- Examples:
  - interactive theorem proving
  - model checking
- In contrast to testing, **exhaustive:**
  - All possible interleavings
  - All possible reorderings
  - All possible contexts, ...
- Problems:
  - **High level of expertise** required
  - Often **unrealistic assumptions** (eg, hardware simplifications)

What is **Practical** Verification?

Formal verification

- Goal: **show correctness**
- Examples:
  - interactive theorem proving
  - model checking
- In contrast to testing, **exhaustive:**
  - All possible interleavings
  - All possible reorderings
  - All possible contexts, ...
- Problems:
  - **High level of expertise** required
  - Often **unrealistic assumptions** (eg, hardware simplifications)

Practical verification?

- Goal: **increase confidence**
- Exhaustive as long **as it pays off**
- A concurrency counterpart of testing
- **Ideally any developer** can use it:
  - Push-button, fully-automated
  - No expertise required

What is **Practical** Verification?

Formal verification

- Goal: **show correctness**
- Examples:
  - interactive theorem proving
  - model checking
- In contrast to testing, **exhaustive:**
  - All possible interleavings
  - All possible reorderings
  - All possible contexts, ...
- Problems:
  - **High level of expertise** required
  - Often **unrealistic assumptions** (eg, hardware simplifications)

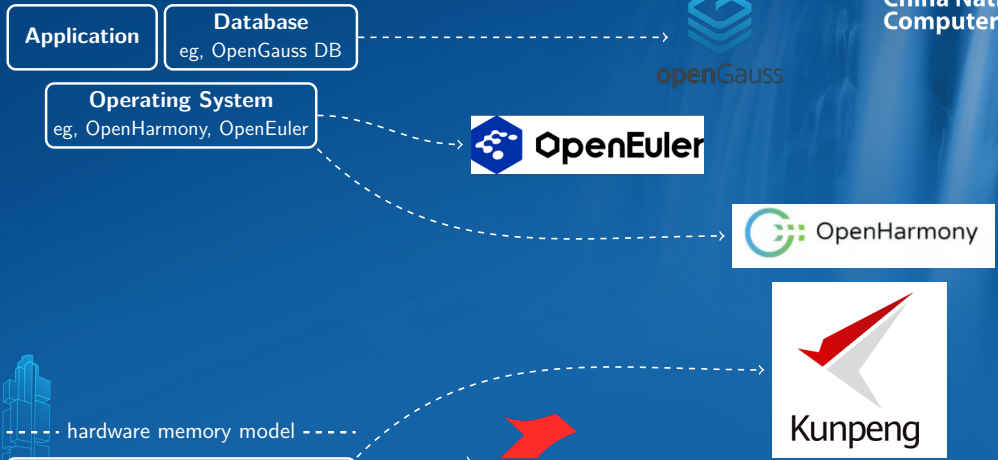
Practical verification?

- Goal: **increase confidence**
- Exhaustive as long **as it pays off**
- A concurrency counterpart of testing
- **Ideally any developer** can use it:
  - Push-button, fully-automated
  - No expertise required
- **In practice, a few experts** must help
  - Defining what to check (properties)
  - Improving tool **scalability**
  - Making methods **more realistic**

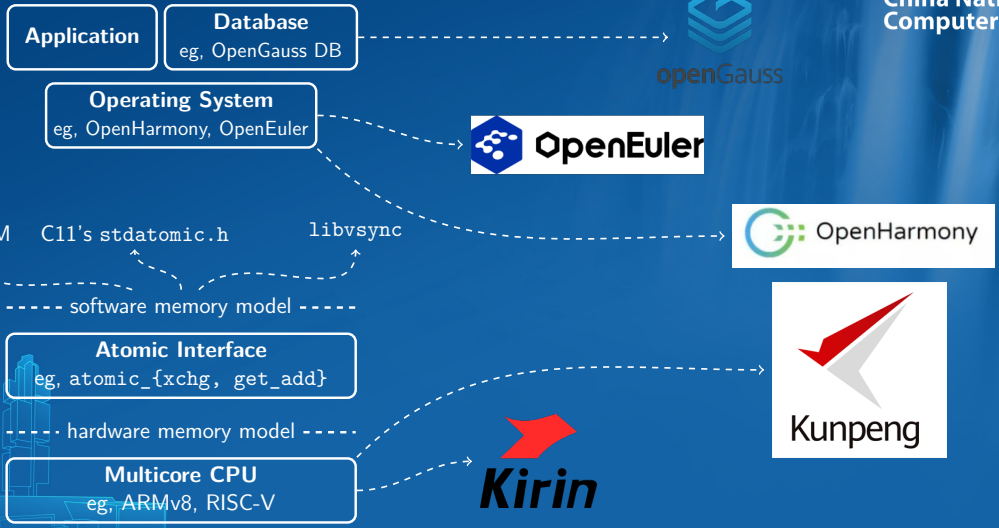
Where do we apply practical verification?



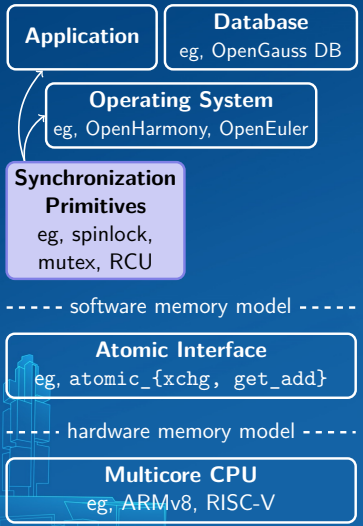
Practical Verification for Concurrency



Practical Verification for Concurrency



Practical Verification for Concurrency



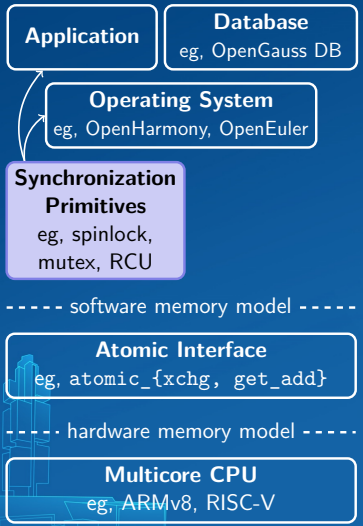
Synchronous with locks

```
run_thread() {
    // some parallel work
    initialize_data();
    do_something_locally();

    // critical section
    lock_acquire(&lock);
    sequential_code();
    lock_release(&lock);

    // some more parallel work
    do_something_locally();
}
```

Practical Verification for Concurrency



Synchronous with locks

```
run_thread() {
    // some parallel work
    initialize_data();
    do_something_locally();

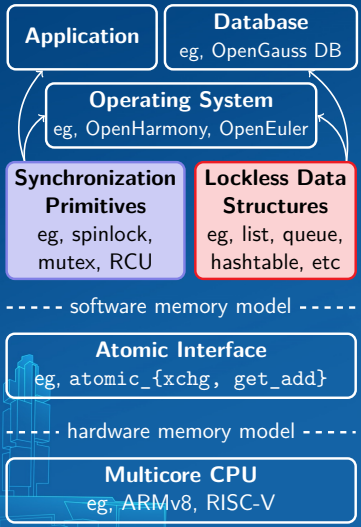
    // critical section
    lock_acquire(&lock);
    sequential_code();
    lock_release(&lock);

    // some more parallel work
    do_something_locally();
}
```

DPDK
DATA PLANE DEVELOPMENT KIT

seL4
Security. Performance. Proof.

Practical Verification for Concurrency



Synchronous with locks

```
run_thread() {
    // some parallel work
    initialize_data();
    do_something_locally();

    // critical section
    lock_acquire(&lock);
    sequential_code();
    lock_release(&lock);

    // some more parallel work
    do_something_locally();
}
```

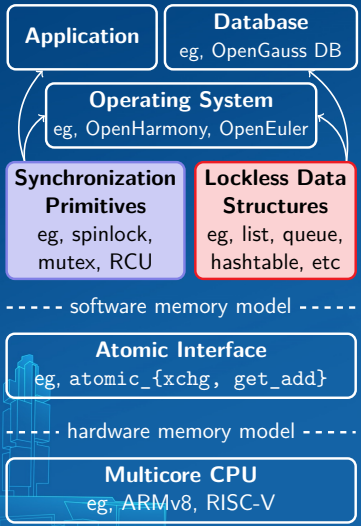
Asynchronous with data structures

```
thread1() {
    do_something_locally();
    queue_enq(q, data);
    do_something_else_locally();
}
thread2() {
    data_t *data;
    data = queue_deq(q);
    if (data != NULL)
        do_something_locally(data);
}
```

DPDK
DATA PLANE DEVELOPMENT KIT

seL4
Security. Performance. Proof.

Practical Verification for Concurrency



Synchronous with locks

```
run_thread() {
    // some parallel work
    initialize_data();
    do_something_locally();

    // critical section
    lock_acquire(&lock);
    sequential_code();
    lock_release(&lock);

    // some more parallel work
    do_something_locally();
}
```

Asynchronous with data structures

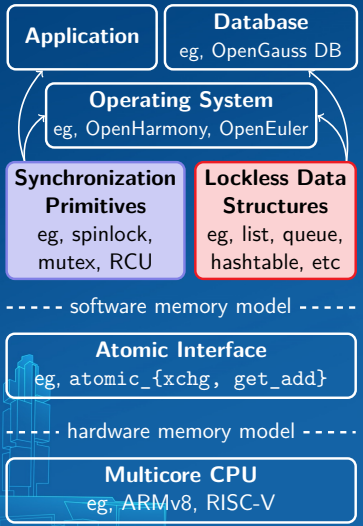
```
thread1() {
    do_something_locally();
    queue_enq(q, data);
    do_something_else_locally();
}
thread2() {
    data_t *data;
    data = queue_deq(q);
    if (data != NULL)
        do_something_locally(data);
}
```

DPDK DATA PLANE DEVELOPMENT KIT

seL4 Security. Performance. Proof.

MariaDB

Practical Verification for Concurrency



Synchronous with locks

```
run_thread() {
    // some parallel work
    initialize_data();
    do_something_locally();

    // critical section
    lock_acquire(&lock);
    sequential_code();
    lock_release(&lock);

    // some more parallel work
    do_something_locally();
}
```

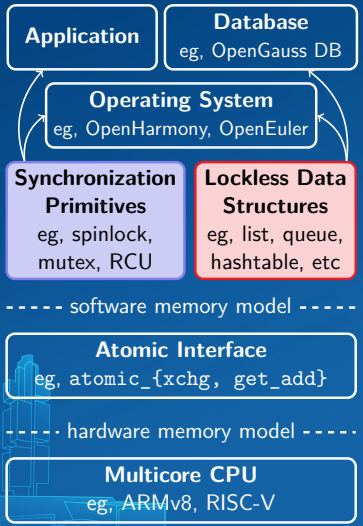
Asynchronous with data structures

```
thread1() {
    do_something_locally();
    queue_enq(q, data);
    do_something_else_locally();
}
thread2() {
    data_t *data;
    data = queue_deq(q);
    if (data != NULL)
        do_something_locally(data);
}
```

Adhoc synchronization

```
if (atomic_add(counter) > MAX)
    do_something(data);
```

Practical Verification for Concurrency



Synchronous with locks

```
run_thread() {
    // some parallel work
    initialize_data();
    do_something_locally();

    // critical section
    lock_acquire(&lock);
    sequential_code();
    lock_release(&lock);

    // some more parallel work
    do_something_locally();
}
```

Asynchronous with data structures

```
thread1() {
    do_something_locally();
    queue_enq(q, data);
    do_something_else_locally();
}
thread2() {
    data_t *data;
    data = queue_deq(q);
    if (data != NULL)
        do_something_locally(data);
}
```

Adhoc synchronization

```
if (atomic_read(counter) > MAX)
    do_something(data);
```

Practical Verification for Concurrency

- **Sequential code**
  - It can be tested! Good news!
  - Even the critical section code
- **Concurrent code**
  - Cannot be easily tested
  - Barrier placement is hard
  - Hardware-awareness is hard
  - Scope of practical verification!

Synchronous with locks

```
run_thread() {
    // some parallel work
    initialize_data();
    do_something_locally();

    // critical section
    lock_acquire(&lock);
    sequential_code();
    lock_release(&lock);

    // some more parallel work
    do_something_locally();
}
```

Asynchronous with data structures

```
thread1() {
    do_something_locally();
    queue_enq(q, data);
    do_something_else_locally();
}
thread2() {
    data_t *data;
    data = queue_deq(q);
    if (data != NULL)
        do_something_locally(data);
}
```

Adhoc synchronization

```
if (atomic_read(counter) > MAX)
    do_something(data);
```



Agenda

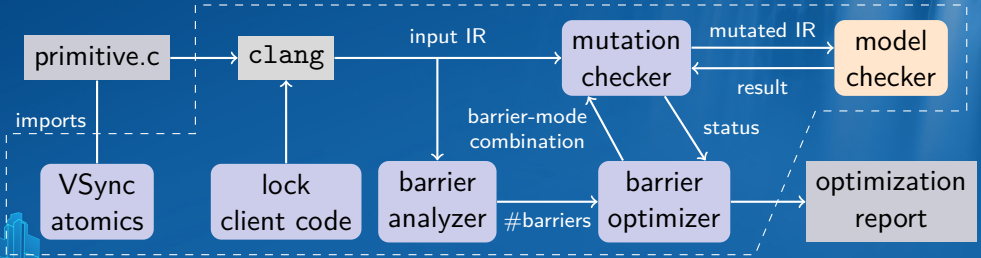
- Challenges in post Moore's Law Era
- Enabling Performance with Practical Verification
- **Practical Verification in Practice**
  VSync, CNA, CLoF and BBQ
- Closing Thoughts and Outlook



VSync

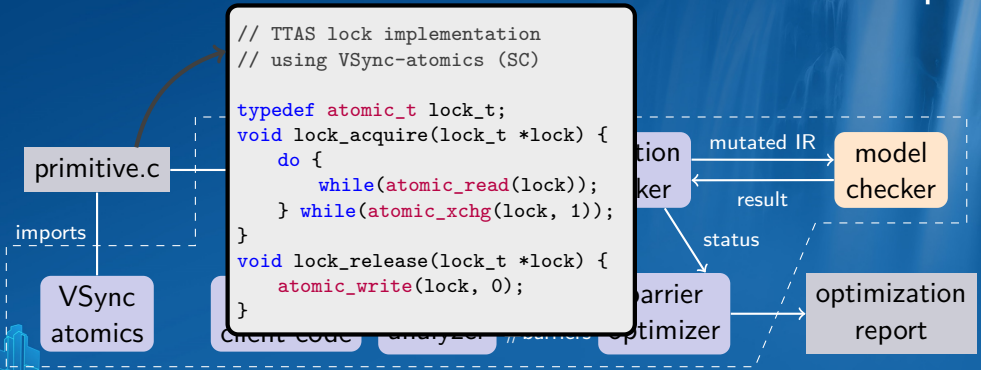
Push-button Verification and Optimization on WMMs

- VSync: https://dl.acm.org/doi/abs/10.1145/3445814.3446748 — **Best paper** @ ASPLOS'21
  by Oberhauser, Chehab, Behrens, Fu, Paolillo, Oberhauser, Bhat, Wen, Chen, Kim, Vafeiadis
- Making relaxed memory models fair: https://dl.acm.org/doi/abs/10.1145/3485475 — **Best paper** @ OOPSLA'21
  by Lahav, Namakonov, Oberhauser, Podkopaev, Vafeiadis



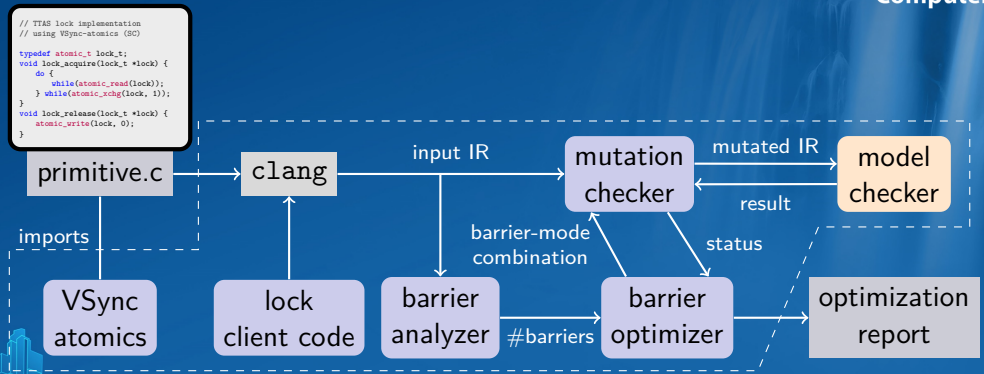
VSync

Push-button Verification and Optimization on WMMs



VSync

Push-button Verification and Optimization on WMMs



VSync

Push-button Verification and Optimization on WMMs


```
// TTAS lock implementation
// using VSync-atomics (SC)

typedef atomic_t lock_t;
void lock_acquire(lock_t *lock) {
    do {
        while(atomic_read(lock));
    } while(atomic_xchg(lock, 1));
}
void lock_release(lock_t *lock) {
    atomic_write(lock, 0);
}
```

primitive.c

cl

imports

VSync-atomics	RISC-V	ARMv8	x86
atomic_xchg	amoswap.w.aq.rl.sc	LDAXR;STLXR	XCHG
atomic_xchg_rel	amoswap.w.rl	LDXR;STLXR	XCHG
atomic_xchg_acq	amoswap.w.aq	LDAXR;STXR	XCHG
atomic_xchg_rlx	amoswap.w	LDXR;STXR	XCHG
atomic_read	fence [rw,rw]; lw; fence [r,rw]	LDAR	MOV
atomic_read_acq	lw; fence [r,rw]	LDAR	MOV
atomic_read_rlx	lw	LDR	MOV
atomic_write	fence [rw,w]; sw; fence [rw,rw]	STLR	MOV;MFENCE
atomic_write_rel	fence [rw,w]; sw	STLR	MOV
atomic_write_rlx	sw	STR	MOV
atomic_fence	fence [rw,rw]	DMB ISH	MFENCE
atomic_fence_acq	fence [r,rw]	DMB ISHLD	NOP
atomic_fence_rel	fence [rw,w]	DMB ISH	NOP
atomic_fence_rlx	nop	NOP	NOP

VSync atomics

lock client code

barrier analyzer

#barriers

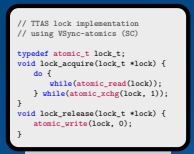
barrier optimizer

optimization report

VSync

Push-button Verification and Optimization on WMMs

CNCC China National Computer Congress



primitive.c

clang

input IR

mutation checker

mutated IR

model checker

result

imports

barrier-mode combination

status

VSync atomics

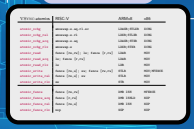
lock client code

barrier analyzer

#barriers

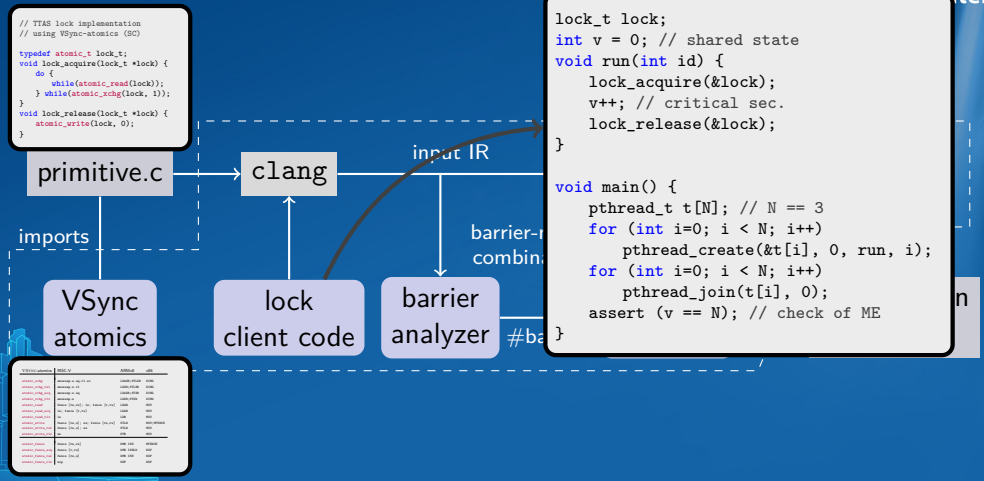
barrier optimizer

optimization report



VSync

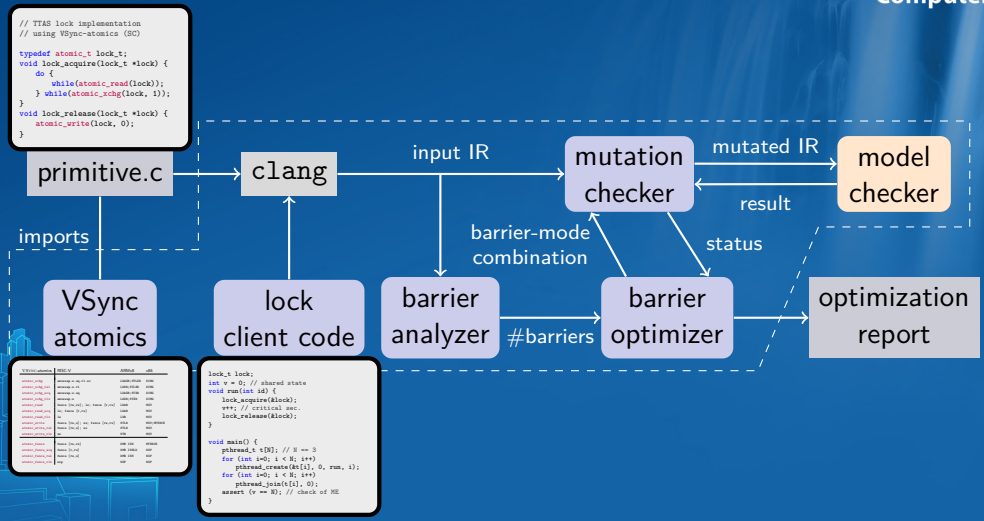
Push-button Verification and Optimization on WMMs



VSync

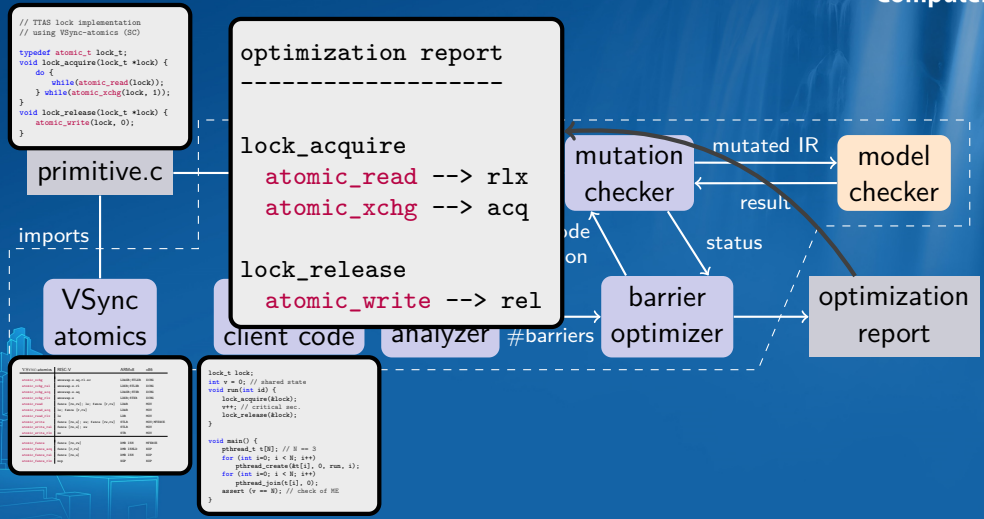
Push-button Verification and Optimization on WMMs

CNCC China National Computer Congress



VSync

Push-button Verification and Optimization on WMMs

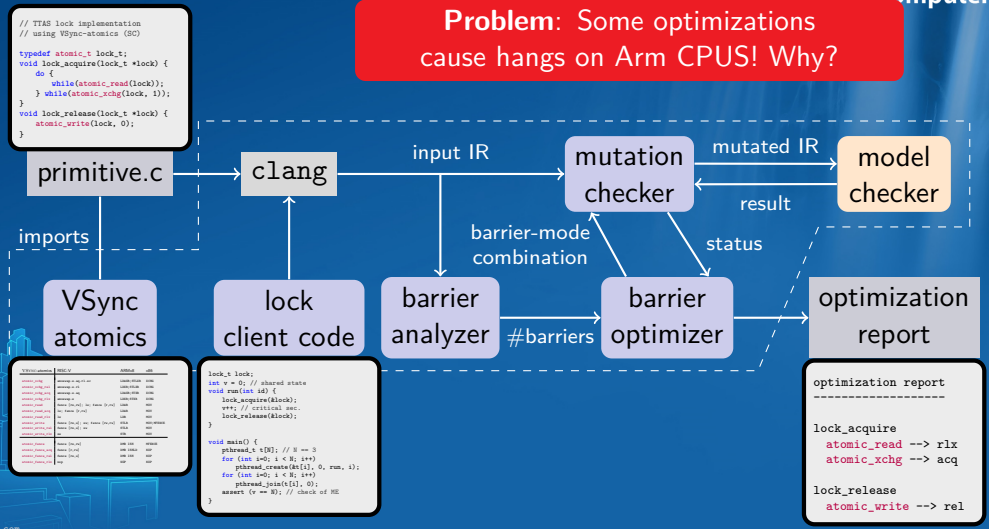


VSync

Push-button Verification and Optimization on WMMs

China National Computer Congress

Problem: Some optimizations cause hangs on Arm CPUS! Why?

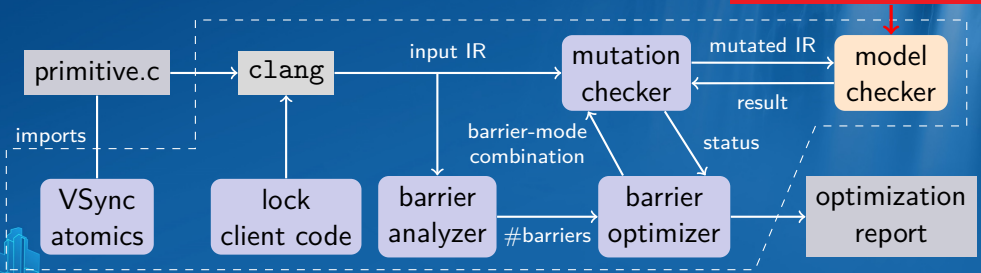


VSync

Push-button Verification and Optimization on WMMs

- VSync: https://dl.acm.org/doi/abs/10.1145/3445814.3446748 — **Best paper** @ ASPLOS'21
  by Oberhauser, Chehab, Behrens, Fu, Paolillo, Oberhauser, Bhat, Wen, Chen, Kim, Vafeiadis
- Making relaxed memory models fair: https://dl.acm.org/doi/abs/10.1145/3485475 — **Best paper** @ OOPSLA'21
  by Lahav, Namakonov, Oberhauser, Podkopaev, Vafeiadis

Await Model Checking



No scalable model checker for WMMs was capable of checking termination of spinloops!



Daniel Vetter @danvet · Aug 2

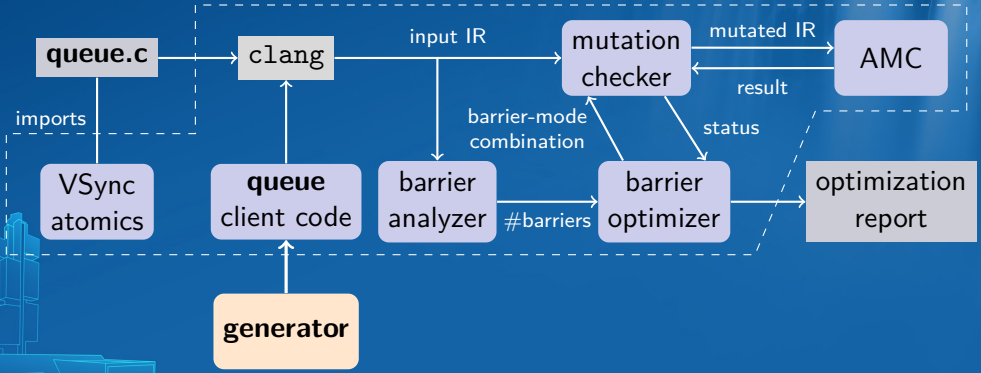
never invent your own locking primitives. the kernel's locks are built and tuned by people who've done nothing else their entire career, you wont beat them except in bug count, and that by a lot.

for more of me being opinionated in locking engineering
blog.ffwll.ch/2022/07/lockin...

4 18 125

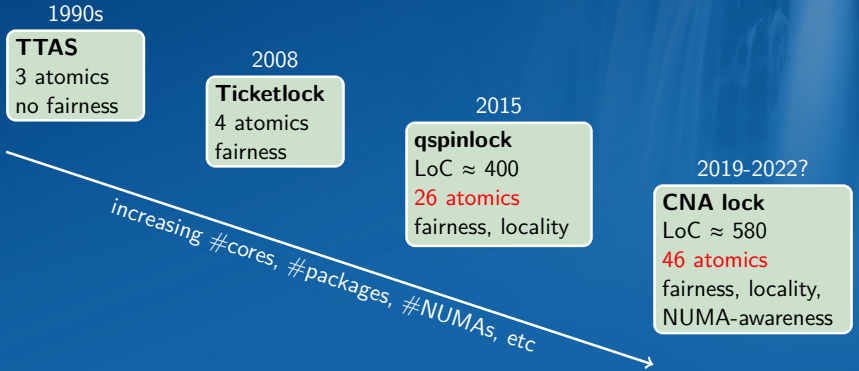
VSync

Current work: extend VSync to data structures
eg, queues, lists, stacks



Verifying Linux qspinlock on LKMM

LKMM = Linux Kernel Memory Model



"atomics" refers to racy accesses, ie, variables concurrently accessed by multiple cores

Verifying CNA-based Linux qspinlock on LKMM

Technical report @ arXiv Jul'22 — by Paolillo, Ponce de León, Haas, Behrens, Chehab, Fu, Meyer

https://arxiv.org/abs/2111.15240

- **Goal**:
  - Support merge of CNA patch (Oracle)
- **Verification:**
  - Linux Kernel Memory Model (LKMM)
  - Armv8 memory model
  - Power memory model
- **Verification time:**
  - With 4 threads, from 12h to 15h



Verifying CNA-based Linux qspinlock on LKMM

Technical report @ arXiv Jul'22 — by Paolillo, Ponce de León, Haas, Behrens, Chehab, Fu, Meyer

https://arxiv.org/abs/2111.15240

- **Goal**:
  - Support merge of CNA patch (Oracle)
- **Verification:**
  - Linux Kernel Memory Model (LKMM)
  - Armv8 memory model
  - Power memory model
- **Verification time:**
  - With 4 threads, from 12h to 15h

- **Linux qspinlock and CNA:**
  - **correct on Armv8**
  - correct on Power



Verifying CNA-based Linux qspinlock on LKMM

Technical report @ arXiv Jul'22 — by Paolillo, Ponce de León, Haas, Behrens, Chehab, Fu, Meyer

https://arxiv.org/abs/2111.15240

CNCC China National Computer Congress

- **Goal**:
  - Support merge of CNA patch (Oracle)
- **Verification:**
  - Linux Kernel Memory Model (LKMM)
  - Armv8 memory model
  - Power memory model
- **Verification time:**
  - With 4 threads, from 12h to 15h

- **Linux qspinlock and CNA:**
  - **correct on Armv8**
  - correct on Power
  - **incorrect on LKMM**!
- **Lessons learned**
  - Was the model checker broken? No!
  - LKMM mismatches reality!
  - Our fix patch was merged:

    https://lkml.org/lkml/2022/11/14/694



Verifying CNA-based Linux qspinlock on LKMM

Technical report @ arXiv Jul'22 — by Paolillo, Ponce de León, Haas, Behrens, Chehab, Fu, Meyer

https://arxiv.org/abs/2111.15240

CNCC China National Computer Congress

- **Goal**:
  - Support merge of CNA patch (Oracle)
- **Verification:**
  - Linux Kernel Memory Model (LKMM)
  - Armv8 memory model
  - Power memory model
- **Verification time:**
  - With 4 threads, from 12h to 15h

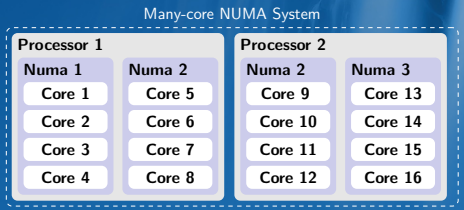


- **Linux qspinlock and CNA:**
  - **correct on Armv8**
  - correct on Power
  - **incorrect on LKMM**!
- **Lessons learned**
  - Was the model checker broken? No!
  - LKMM mismatches reality!
  - Our fix patch was merged:
    https://lkml.org/lkml/2022/11/14/694
  - Having 2 model checkers is helpful!

https://github.com/hernanponcedeleon/Dat3M
https://github.com/MPI-SWS/genmc

What if too complex to model check?

Modern lock-design challenges

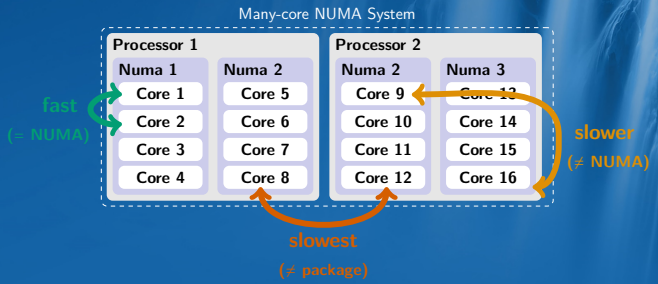




What if too complex to model check?

Modern lock-design challenges

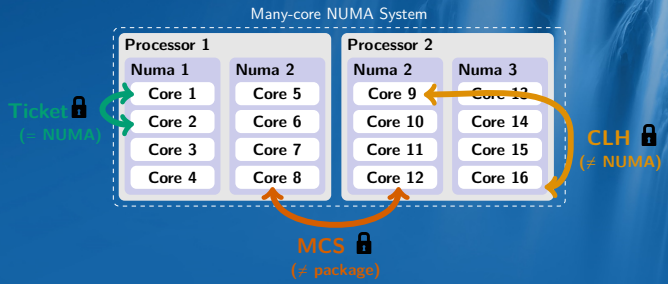
- **Deep NUMA hierarchies**
  eg, packages, NUMA nodes, L3 cache partitions



What if too complex to model check?

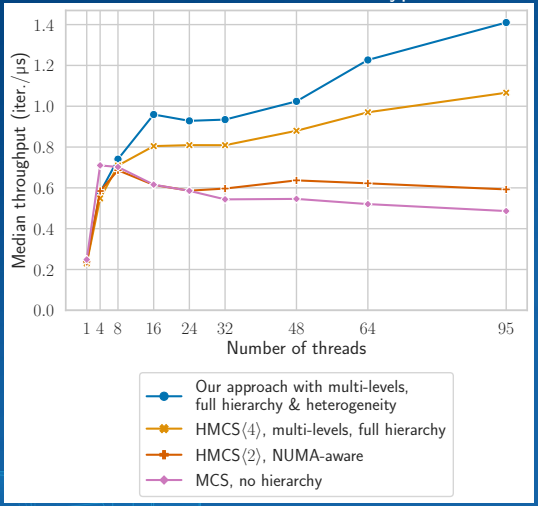
Modern lock-design challenges

- **Deep NUMA hierarchies**
  eg, packages, NUMA nodes, L3 cache partitions
- **Lock heterogenity**
  Different locks perform better in different contexts, eg, cores with shared cache or not



What if too complex to model check?

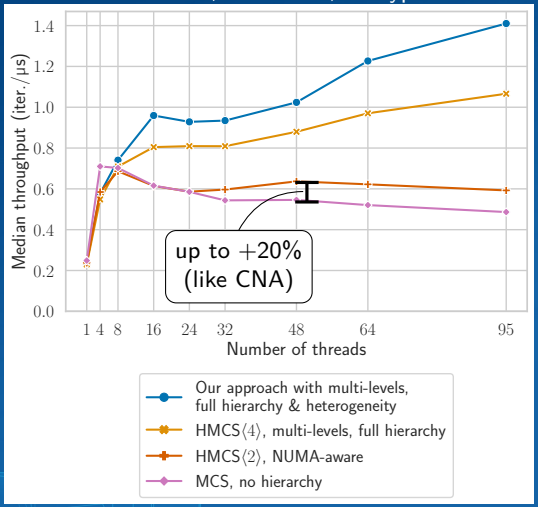
LevelDB benchmark, x86 server, 96 hyperthreads



What are the potential improvements?

What if too complex to model check?

LevelDB benchmark, x86 server, 96 hyperthreads

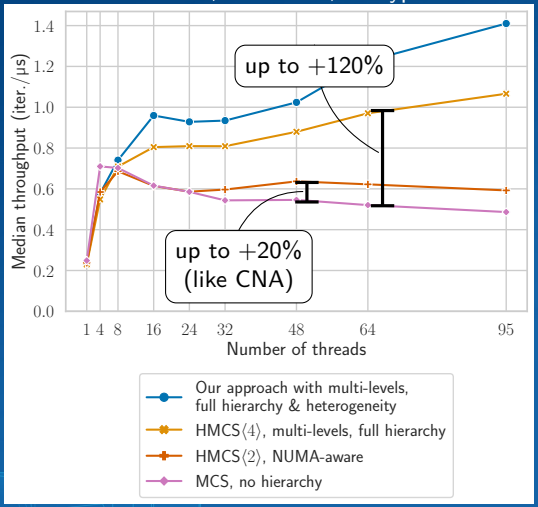


What are the potential improvements?

- 2-level is good

What if too complex to model check?

LevelDB benchmark, x86 server, 96 hyperthreads

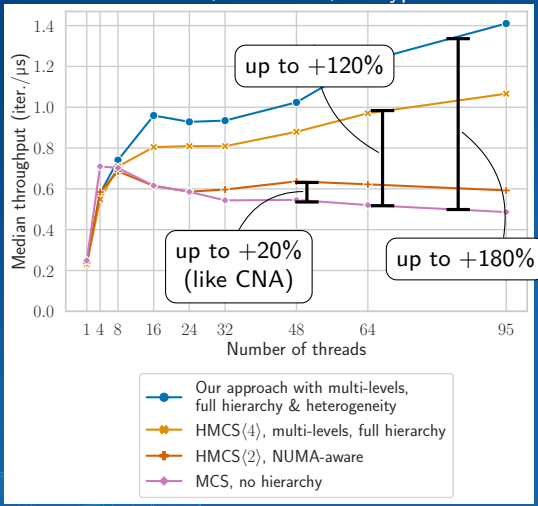


What are the potential improvements?

- 2-level is good
- Multi-level is great

What if too complex to model check?

LevelDB benchmark, x86 server, 96 hyperthreads

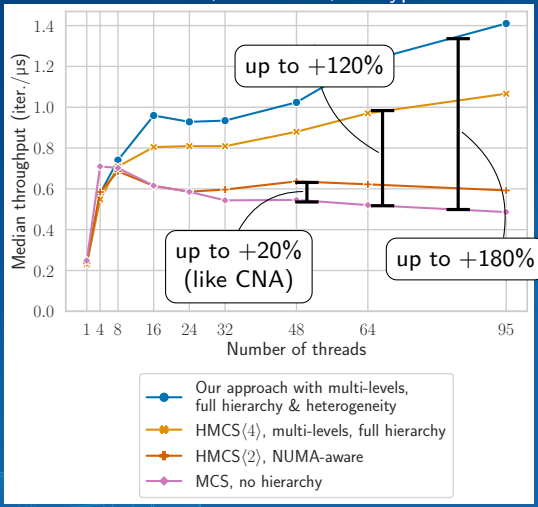


What are the potential improvements?

- 2-level is good
- Multi-level is great
- Multi-level+heterogeneity is awesome!

What if too complex to model check?

LevelDB benchmark, x86 server, 96 hyperthreads



What are the potential improvements?

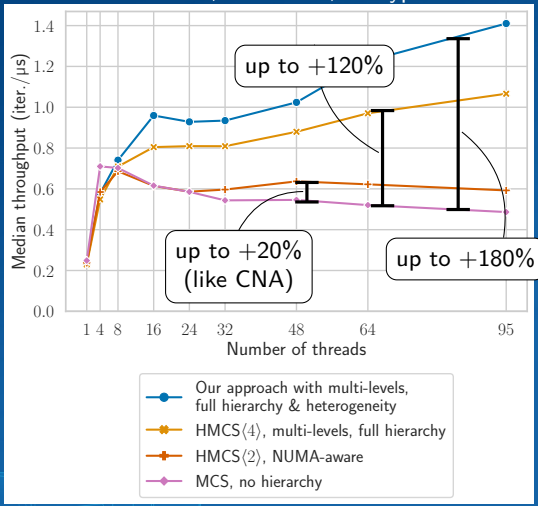
- 2-level is good
- Multi-level is great
- Multi-level+heterogeneity is awesome!

But model checkers can't handle that

- Example multi-level HMCS:

What if too complex to model check?

LevelDB benchmark, x86 server, 96 hyperthreads



What are the potential improvements?

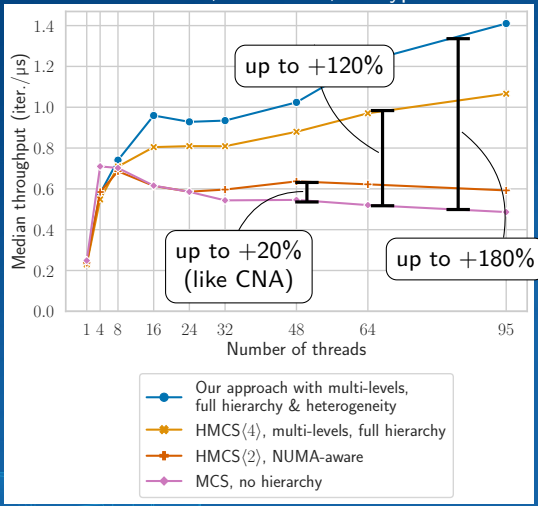
- 2-level is good
- Multi-level is great
- Multi-level+heterogeneity is awesome!

But model checkers can't handle that

- Example multi-level HMCS:
  - 2 levels: 2s
  - 3 levels: 10s
  - 4 levels: timeout after 24h

What if too complex to model check?

LevelDB benchmark, x86 server, 96 hyperthreads



What are the potential improvements?

- 2-level is good
- Multi-level is great
- Multi-level+heterogeneity is awesome!

But model checkers can't handle that

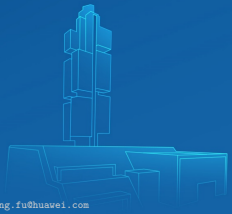
- Example multi-level HMCS:
  - 2 levels: 2s
  - 3 levels: 10s
  - 4 levels: timeout after 24h
- **Enter CLoF**!

CLoF: Compositional Lock Framework

Published @ SOSP'21 — by Chehab, Paolillo, Behrens, Fu, Chen, Härtig
https://dl.acm.org/doi/abs/10.1145/3477132.3483557

Base Step

Induction Step



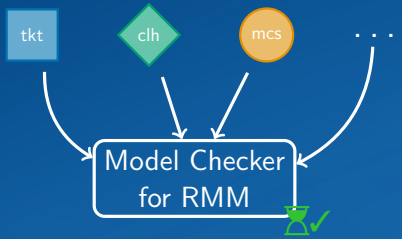
CLoF: Compositional Lock Framework

Published @ SOSP'21 — by Chehab, Paolillo, Behrens, Fu, Chen, Härtig

https://dl.acm.org/doi/abs/10.1145/3477132.3483557

China National Computer Congress

Base Step



As in VSync paper

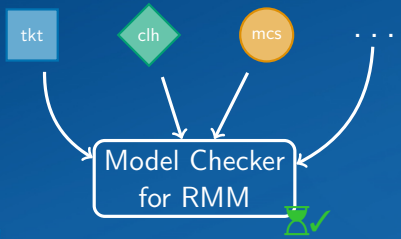
Induction Step

CLoF: Compositional Lock Framework

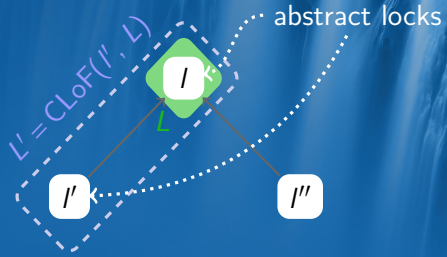
Published @ SOSP'21 — by Chehab, Paolillo, Behrens, Fu, Chen, Härtig
https://dl.acm.org/doi/abs/10.1145/3477132.3483557

CNCC China National Computer Congress

Base Step



Induction Step

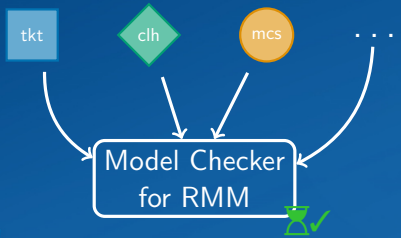


CLoF: Compositional Lock Framework

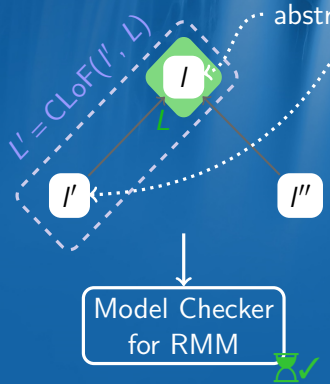
Published @ SOSP'21 — by Chehab, Paolillo, Behrens, Fu, Chen, Härtig
https://dl.acm.org/doi/abs/10.1145/3477132.3483557

CNCC China National Computer Congress

Base Step



Induction Step

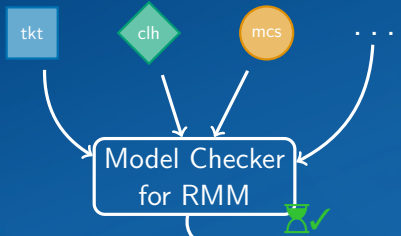


CLoF: Compositional Lock Framework

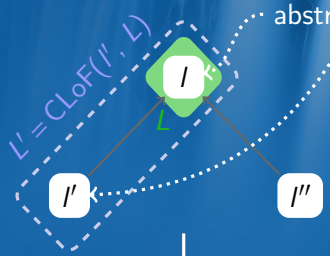
Published @ SOSP'21 — by Chehab, Paolillo, Behrens, Fu, Chen, Härtig
https://dl.acm.org/doi/abs/10.1145/3477132.3483557

CNCC China National Computer Congress

Base Step



Induction Step

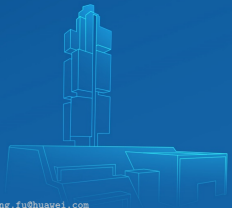


Model Checker for RMM

Inductive correctness for CLoF locks of any depth and basic lock

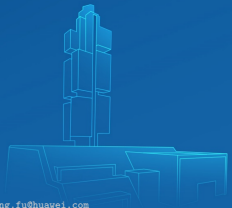
ming.fu@huawei.com

What about concurrent data structures?



What about concurrent data structures?

Ringbuffers are pervasive



What about concurrent data structures?

Ringbuffers are pervasive

















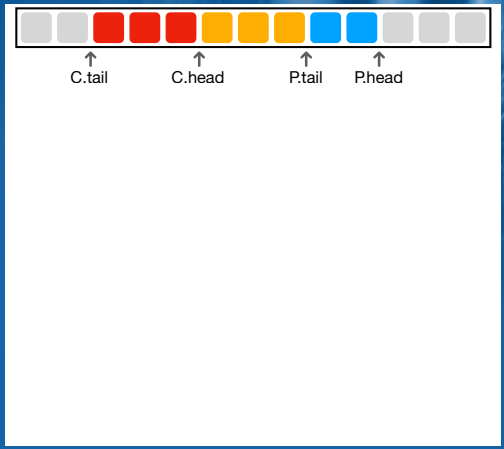




BBQ: Block-based Bounded Queue

USENIX ATC'22 — by Wang, Behrens, Fu, Oberhauser, Oberhauser, Lei, Chen, Härtig, Chen

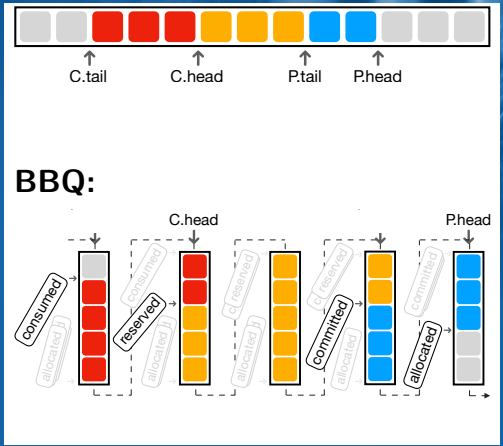
- **Conventional designs:**
  - most **favor simplicity**
  - performance ~ interference enq/deq



BBQ: Block-based Bounded Queue

USENIX ATC'22 — by Wang, Behrens, Fu, Oberhauser, Oberhauser, Lei, Chen, Härtig, Chen

- **Conventional designs:**
  - most **favor simplicity**
  - performance ~ interference enq/deq
- **Block Approach:**
  - Split ringbuffer and metadata in blocks
  - Drastically reduce enq/deq interference
  - 1.5x to 50x higher throughput
  - **Verified with VSync**
  - BBQ is the **fastest queue in the world**



ming.fu@huawei.com

BBQ Learnings

The cost of increased performance

China National Computer Congress

DPDK-like algorithm

```
1  enqueue(data){
2  again:
3    ph = LOAD(P.head);
4    pn = ph + 1;
5    if (pn > LOAD(C.tail) + SZ)
6      return FULL;
7    if (!CAS(P.head, ph, pn))
8      goto again;
9    entry[pn % SZ] = data;
10   while(LOAD(P.tail) != ph);
11   STORE(P.tail, pn);
12   return OK;
13 }
```

```
14 dequeue(){
15 again:
16   ch = LOAD(C.head);
17   cn = ch + 1;
18   if (cn > LOAD(P.tail))
19     return EMPTY;
20 if (!CAS(C.head, ch, cn))
21   goto again;
22 data = entry[cn % SZ];
23 while(LOAD(C.tail) != ch);
24 STORE(C.tail, cn);
25 return data;
26 }
```



≈ 10 atomics

BBQ Learnings

The cost of increased performance

China National Computer Congress

Part of BBQ

DPDK-like algorithm

```
enqueue(data){
again:
  ph = LOAD(P.head);
  pn = ph + 1;
  if (pn > LOAD(C.tail) + SZ)
    return FULL;
  if (!CAS(P.head, ph, pn))
    goto again;
  entry[pn % SZ] = data;
  while(LOAD(P.tail) != ph);
  STORE(P.tail, pn);
  return OK;
}
```

```
dequeue(){
again:
  ch = LOAD(C.head);
  cn = ch + 1;
  if (cn > LOAD(P.tail))
    return EMPTY;
  if (!CAS(C.head, ch, cn))
    goto again;
  data = entry[cn % SZ];
  while(LOAD(C.tail) != ch);
  STORE(C.tail, cn);
  return data;
}
```

≈ 10 atomics

increased complexity

```
<Head, Block> BBQ<T>::get_phead_and_block(){
  ph = LOAD(phead);
  return {ph, blocks[ph.idx]};
}
state BBQ<T>::allocate_entry(Block blk){
  if (LOAD(blk.allocated).off >= BLOCK_SIZE)
    return BLOCK_DONE;
  old = FAA(blk.allocated, 1).off;
  if (old >= BLOCK_SIZE)
    return BLOCK_DONE;
  return ALLOCATED{EntryDesc(.block=blk, .offset=old)};
}
void BBQ<T>::commit_entry(EntryDesc e, T data){
  e.block.entries[e.offset] = data;
  ADD(e.block.committed, 1);
}
state BBQ<T>::advance_phead(Head ph) {
  nblk = blocks[(ph.idx + 1) % BLOCK_NUM];
  cons = LOAD(nblk.consumed);
  if (cons.vsn < ph.vsn ||
      (cons.vsn == ph.vsn && cons.off != BLOCK_SIZE)) {
    reserved = LOAD(nblk.reserved);
    if (reserved.off == cons.off) return NO_ENTRY;
    else return NOT_AVAILABLE;
  }
  cmtd = LOAD(nblk.committed);
  if (cmtd.vsn == ph.vsn && cmtd.off != BLOCK_SIZE)
    return NOT_AVAILABLE;
  MAX(nblk.committed, Cursor{.vsn=ph.vsn + 1});
  MAX(nblk.allocated, Cursor{.vsn=ph.vsn + 1});
  MAX(phead, ph + 1);
  return SUCCESS;
}
class BBQ<T> {
  shared<Head> phead, chead;
  Block<T>[] blocks;
}
class Block<T> {
  shared<Cursor> allocated, committed;
  shared<Cursor> reserved, consumed;
  T[] entries;
}
class EntryDesc {
  Block block; Offset offset; Version version; }
```

```
<Head, Block> BBQ<T>::get_chead_and_block(){
  ch = LOAD(chead);
  return {ch, blocks[ch.idx]};
}
state BBQ<T>::reserve_entry(Block blk){
again:
  reserved = LOAD(blk.reserved);
  if (reserved.off < BLOCK_SIZE) {
    committed = LOAD(blk.committed);
    if (reserved.off == committed.off)
      return NO_ENTRY;
    if (committed.off != BLOCK_SIZE){
      allocated = LOAD(blk.allocated);
      if (allocated.off != committed.off)
        return NOT_AVAILABLE;
    }
    if (MAX(blk.reserved, reserved + 1) == reserved)
      return RESERVED{EntryDesc{.block=blk,
        .offset=reserved.off, .version=reserved.vsn}};
    else goto again;
  }
  return BLOCK_DONE{reserved.vsn};
}
T BBQ<T>::consume_entry(EntryDesc e){
  data = e.block.entries[e.offset];
  ADD(e.block.consumed, 1);
  allocated = LOAD(e.block.allocated);
  if (allocated.vsn != e.version) return NULL;
  return data;
}
bool BBQ<T>::advance_chead(Head ch, Version vsn){
  nblk = blocks[(ch.idx + 1) % BLOCK_NUM];
  committed = LOAD(nblk.committed);
  if (committed.vsn != ch.vsn + 1)
    return false;
  MAX(nblk.consumed, Cursor{.vsn=ch.vsn + 1});
  MAX(nblk.reserved, Cursor{.vsn=ch.vsn + 1});
  if (committed.vsn < vsn + (ch.idx == 0))
    return false;
  MAX(nblk.reserved, Cursor{.vsn=committed.vsn});
  MAX(chead, ch + 1);
  return true;
}
```

retry-new mode | drop-old mode

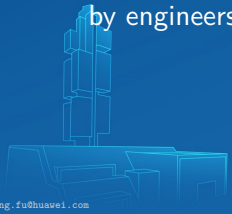
≈ 20 atomics

BBQ Learnings

How to ensure correctness?

- **Long stress testing**
  by engineers
- **Identification of corner cases**
  by experts and engineers
- **Model checking of corner cases**
  by engineers with expert support

- **Only a few corner cases necessary**
  queue full/empty, FIFO, wrap-around
- **3 bugs found model checking them**
  Not found while stress testing
- **Reproducible on real hardware**
  Test cases were built in retrospect



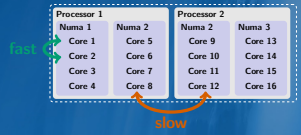
Agenda

- Challenges in post Moore's Law Era
- Enabling Performance with Practical Verification
- Practical Verification in Practice
  VSync, CNA, CLoF and BBQ
- **Closing Thoughts and Outlook**

Enabling Performance with Practical Verification

Modern hardware features

- Relaxed Memory Models, eg, Arm, RISC-V
- Deep NUMA hierarchies



modern hardware

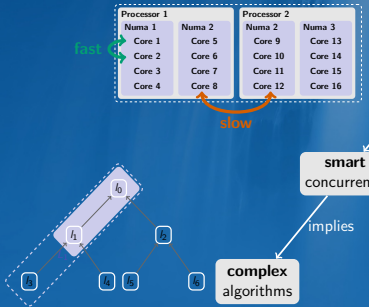
Enabling Performance with Practical Verification

Modern hardware features

- Relaxed Memory Models, eg, Arm, RISC-V
- Deep NUMA hierarchies

Consequences to concurrency

- Must be **smarter** to boost performance



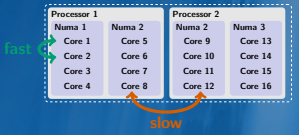
Enabling Performance with Practical Verification

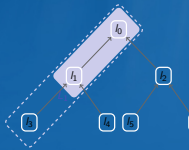
Modern hardware features

- Relaxed Memory Models, eg, Arm, RISC-V
- Deep NUMA hierarchies

Consequences to concurrency

- Must be **smarter** to boost performance
- But **complexity gets out of control!**





modern hardware

requires

smart concurrency

implies

complex algorithms

requires

super expert developer team

Really?





Enabling Performance with Practical Verification

Modern hardware features

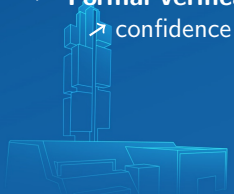
- Relaxed Memory Models, eg, Arm, RISC-V
- Deep NUMA hierarchies

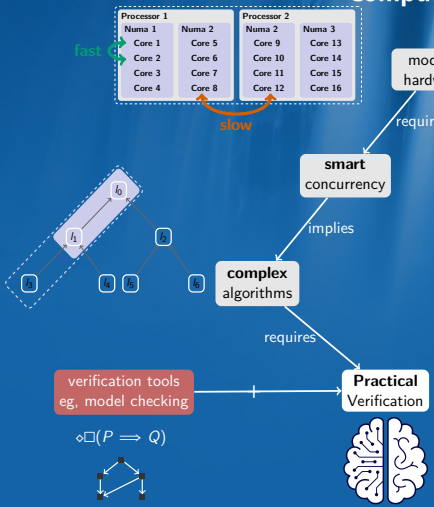
Consequences to concurrency

- Must be **smarter** to boost performance
- But **complexity gets out of control!**

Practical verification

- **Formal verification tools**
  - ↗ confidence on code correctness





Enabling Performance with Practical Verification

CNCC China National Computer Congress

Modern hardware features

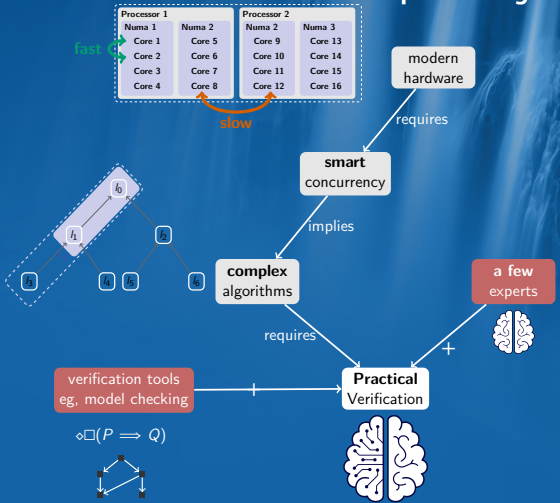
- Relaxed Memory Models, eg, Arm, RISC-V
- Deep NUMA hierarchies

Consequences to concurrency

- Must be **smarter** to boost performance
- But **complexity gets out of control!**

Practical verification

- **Formal verification tools**
  - ↗ confidence on code correctness
- **A few experts**
  - ↗ scalability and coverage of tools



Clash of Communities

	Industry	System	PL
Practicality	✓✓	✓	✗ (ideal model)
Scalability	✓✓	✓	✗
Automation	✓✓	+/-	✗
Performance	✓✓	✓	+/- (complexity)
Theory	🤓 (nobody cares)	+/-	✓✓ (correctness theorem)
Formalization	🤓 (nobody cares)	+/-	✓✓ (math)
Soundness	🤓 (nobody cares)	+/-	✓✓ (Coq proofs)
Finding bugs	✓✓	✓	✓✓ (bug free)
Pay as you go	✓✓	+/-	+/-

AtoMig: Atomic Migrating Huge Codebases

To appear @ ASPLOS'23 — Beck, Bhat, Stricevic, Chen, Behrens, Vafeiadis, Fu, Chen, Härtig

CNCC China National Computer Congress

- **The x86-to-Arm Porting Problem:**
  - Legacy systems **lack barriers** on Arm!
  - Naively protect all accesses?
  - Use special hardware (eg, Apple M1)?
  - Low-level binary porting? (Lasagne)
- **Our Approach**:
  - Look for **patterns** in compile time
  - Add barriers just to those patterns
  - Unsound but **lightweight and practical**!
- **Great results**:
  - Large systems, eg, Postgresql, MariaDB
  - Finds and fixes bugs (see MariaDB bug)
  - Little end-to-end overhead ($< 1\%$)
  - More patterns in the paper!

Pattern: Volatiles

Most volatile variables/accesses in C99 indicate **atomic**. Example:

```
volatile int flag;
...
flag = 1; // must be atomic!
```

Pattern: Spinloops

Loop condition variable **independent of body**:

```
while (!flag) {} // must be atomic!
```

Pattern: Sticky atomics

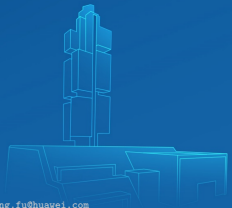
If **atomic once** then **always atomic**, eg,

```
prev = XCHG(&tail, value);
...
assert (tail != NULL); // atomic!
```

ming.fu@huawei.com

How to survive **in the industry** as a formal verification person?

Accept that verification is a tool to achieve the **business goals**;
Verification is the means, not the ends.



Research publications at Huawei DRC

CNCC China National Computer Congress

Hardware-tailored concurrency

- **CLoF: A Compositional Lock Framework for Multi-level NUMA Systems**,
  at SOSP'21 — Chehab, Paolillo, Behrens, Fu, Chen, Härtig
  https://dl.acm.org/doi/10.1145/3477132.3483557
- **Verifying and Optimizing the HMCS Lock for Arm Servers**
  at NETYS'21 — by Oberhauser, Oberhauser, Paolillo, Behrens, Fu, Vafeiadis
  https://link.springer.com/chapter/10.1007/978-3-030-91014-3_17
- **BBQ: A Block-based Bounded Queue for Exchanging Data and Profiling**
  at USENIX ATC'21 — by Wang, Behrens, Fu, Oberhauser, Oberhauser, Lei, Chen, Härtig, Chen
  https://www.usenix.org/conference/atc22/presentation/wang-jiawei
- **Verifying and Optimizing Compact NUMA-Aware Locks on Weak Memory Models**,
  arXiv Jul'22 — by Paolillo, Ponce de León, Haas, Behrens, Chehab, Fu, Meyer
  https://arxiv.org/abs/2111.15240

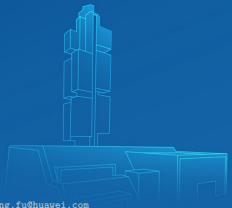
Practical verification

- **VSync: Push-button Verification and Optimization for Synchronization Primitives**,
  at ASPLOS'21 — Oberhauser, Chehab, Behrens, Fu, Paolillo, Oberhauser, Bhat, Wen, Chen, Kim, Vafeiadis
  https://dl.acm.org/doi/10.1145/3445814.3446748
- **AtoMig: Automatically Migrating Millions Lines of Code from TSO to WMM**,
  to appear at ASPLOS'23 — Beck, Bhat, Stričević, Chen, Behrens, Fu, Vafeiadis, Chen, Härtig

CNCC

CNCC
China National
Computer Congress

Thanks



ming.fu@huawei.com