

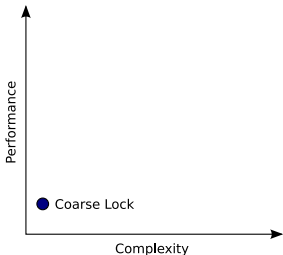
Hardware acceleration for lock-free data structures and software-transactional memory

Stephan Diestelhorst Michael Hohmuth

Advanced Micro Devices
Operating Systems Research Center

EPHAM Workshop, April 6, 2008

Software Approaches to Synchronisation

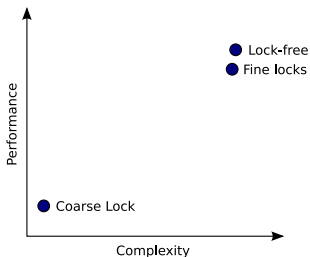


- Single, coarse lock.
- Fine-grained locks.
- Lock-free algorithms.
- Software Transactional Memory (STM).

Advanced Synchronisation Facility

Hardware support for more **flexible** lock-free primitives, an experimental feature by Advanced Micro Devices (AMD).
No intention to release that in silicon.

Software Approaches to Synchronisation

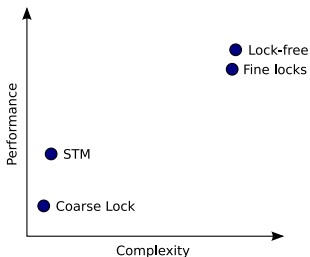


- Single, coarse lock.
- Fine-grained locks.
- Lock-free algorithms.
- Software Transactional Memory (STM).

Advanced Synchronisation Facility

Hardware support for more **flexible** lock-free primitives, an experimental feature by Advanced Micro Devices (AMD).
No intention to release that in silicon.

Software Approaches to Synchronisation

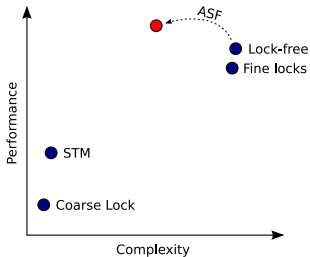


- Single, coarse lock.
- Fine-grained locks.
- Lock-free algorithms.
- Software Transactional Memory (STM).

Advanced Synchronisation Facility

Hardware support for more **flexible** lock-free primitives, an experimental feature by Advanced Micro Devices (AMD).
No intention to release that in silicon.

Software Approaches to Synchronisation

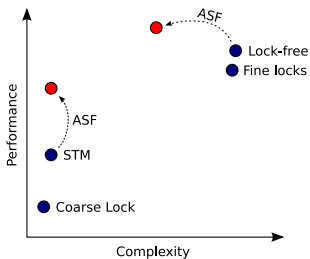


- Single, coarse lock.
- Fine-grained locks.
- Lock-free algorithms.
- Software Transactional Memory (STM).

Advanced Synchronisation Facility

Hardware support for more **flexible** lock-free primitives, an experimental feature by Advanced Micro Devices (AMD).
No intention to release that in silicon.

Software Approaches to Synchronisation



- Single, coarse lock.
- Fine-grained locks.
- Lock-free algorithms.
- Software Transactional Memory (STM).

Advanced Synchronisation Facility

Hardware support for more **flexible** lock-free primitives, an experimental feature by Advanced Micro Devices (AMD).
No intention to release that in silicon.

Outline

- 1 Introduction
 - Motivation
 - Advanced Synchronization Facility
- 2 Acceleration with ASF
 - Lock-Free Linked Lists
 - TinySTM
- 3 Evaluation
 - Setup
 - Accelerated Lock-Free Data Structures
 - Accelerated TinySTM

Outline

- 1 Introduction
 - Motivation
 - Advanced Synchronization Facility
- 2 Acceleration with ASF
 - Lock-Free Linked Lists
 - TinySTM
- 3 Evaluation
 - Setup
 - Accelerated Lock-Free Data Structures
 - Accelerated TinySTM

Need for ASF

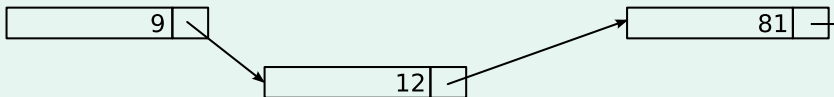
- Atomic CAS only on **single** location.
- Requests for DCAS, DCADS, ...
- More flexibility!

Example: Linked-List Removal

Need for ASF

- Atomic CAS only on **single** location.
- Requests for DCAS, DCADS, ...
- More flexibility!

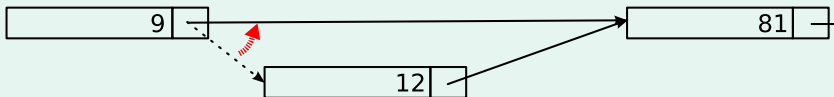
Example: Linked-List Removal



Need for ASF

- Atomic CAS only on **single** location.
- Requests for DCAS, DCADS, ...
- More flexibility!

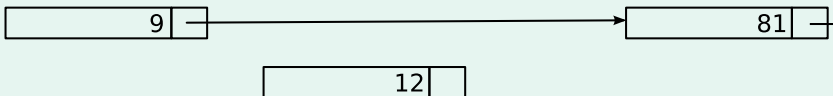
Example: Linked-List Removal



Need for ASF

- Atomic CAS only on **single** location.
- Requests for DCAS, DCADS, ...
- More flexibility!

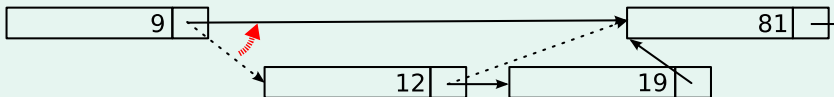
Example: Linked-List Removal



Need for ASF

- Atomic CAS only on **single** location.
- Requests for DCAS, DCADS, ...
- More flexibility!

Example: Linked-List Removal



Need for ASF

- Atomic CAS only on **single** location.
- Requests for DCAS, DCADS, ...
- More flexibility!

Example: Linked-List Removal



Overview ASF

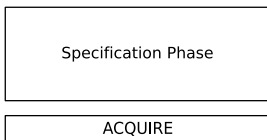
- 1 Specify locations.
- 2 Acquire ownership.
- 3 Modify locations in critical section.
- 4 Make changes available with `commit`.

Overview ASF

Specification Phase

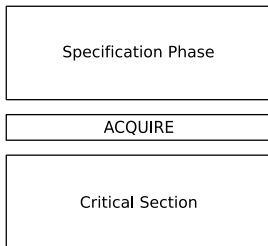
- 1 **Specify** locations.
- 2 **Acquire** ownership.
- 3 Modify locations in **critical section**.
- 4 Make changes available with **commit**.

Overview ASF



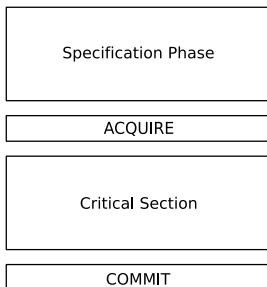
- 1 Specify locations.
- 2 Acquire ownership.
- 3 Modify locations in critical section.
- 4 Make changes available with commit.

Overview ASF



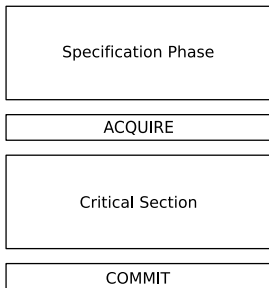
- 1 Specify locations.
- 2 Acquire ownership.
- 3 Modify locations in critical section.
- 4 Make changes available with `commit`.

Overview ASF



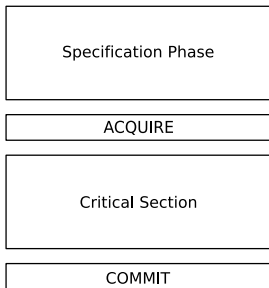
- 1 Specify locations.
- 2 Acquire ownership.
- 3 Modify locations in **critical section**.
- 4 Make changes available with **commit**.

Atomicity of ASF



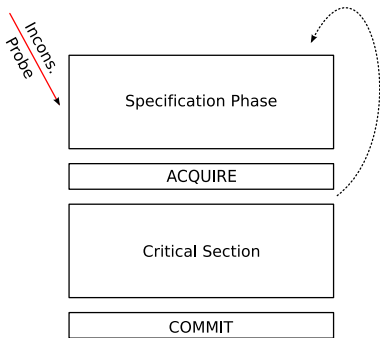
- Ensure that nobody observes **inconsistent** state.
- Use **cache coherence** mechanism to detect observation attempts.
- Incoming inconsistent probe in specification phase:
 - ▶ No speculative updates yet.
 - ▶ ACQUIRE will fail.
 - ▶ Application controls restart.
- Incoming inconsistent probe in critical section:
 - ▶ **Undo** changes. (→ make backup copy)
 - ▶ **Re-execute** ACQUIRE.

Atomicity of ASF



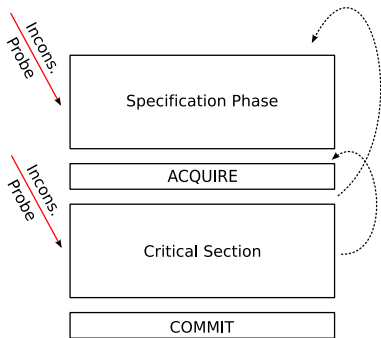
- Ensure that nobody observes **inconsistent** state.
- Use **cache coherence** mechanism to detect observation attempts.
- Incoming inconsistent probe in specification phase:
 - ▶ No speculative updates yet.
 - ▶ ACQUIRE will fail.
 - ▶ Application controls restart.
- Incoming inconsistent probe in critical section:
 - ▶ Undo changes. (→ make backup copy)
 - ▶ Re-execute ACQUIRE.

Atomicity of ASF



- Ensure that nobody observes **inconsistent** state.
- Use **cache coherence** mechanism to detect observation attempts.
- Incoming inconsistent probe in specification phase:
 - ▶ No speculative updates yet.
 - ▶ ACQUIRE will fail.
 - ▶ Application controls restart.
- Incoming inconsistent probe in critical section:
 - ▶ Undo changes. (→ make backup copy)
 - ▶ Re-execute ACQUIRE.

Atomicity of ASF



- Ensure that nobody observes **inconsistent** state.
- Use **cache coherence** mechanism to detect observation attempts.
- Incoming inconsistent probe in specification phase:
 - ▶ No speculative updates yet.
 - ▶ ACQUIRE will fail.
 - ▶ Application controls restart.
- Incoming inconsistent probe in critical section:
 - ▶ **Undo** changes. (→ make backup copy)
 - ▶ **Re-execute** ACQUIRE.

ASF Primitives

LOCK loads

- Start specification phase.
- Specify memory location as part of the speculative working set.

ACQUIRE

- Start critical section.
- Check for consistency of specified locations.
- Store rIP and rSP for rollback.
- Arm the rollback mechanism.

COMMIT

- Conclude critical section.
- Make speculative modifications available in the system.

ASF Primitives

LOCK loads

- Start specification phase.
- Specify memory location as part of the speculative working set.

ACQUIRE

- Start critical section.
- Check for consistency of specified locations.
- Store rIP and rSP for rollback.
- Arm the rollback mechanism.

COMMIT

- Conclude critical section.
- Make speculative modifications available in the system.

ASF Primitives

LOCK loads

- Start specification phase.
- Specify memory location as part of the speculative working set.

ACQUIRE

- Start critical section.
- Check for consistency of specified locations.
- Store rIP and rSP for rollback.
- Arm the rollback mechanism.

COMMIT

- Conclude critical section.
- Make speculative modifications available in the system.

Outline

- 1 Introduction
 - Motivation
 - Advanced Synchronization Facility
- 2 Acceleration with ASF
 - Lock-Free Linked Lists
 - TinySTM
- 3 Evaluation
 - Setup
 - Accelerated Lock-Free Data Structures
 - Accelerated TinySTM

```
int set_remove(set_t *set, int val) {
    ...
retry:
    find(set, val, &prev, &next);
    ...
    contained = 0;
    prev_next = asf_lock_load(&prev->next);
    next_next = asf_lock_load(&next->next);
    next_val = asf_lock_load(&next->val);

    if (!asf_acquire(3)) { /* atomic start */
        /* Could not acquire locations -> Retry */
        goto retry;
    }
    /* check for chaining errors */
    if (prev_next != next) {
        asf_commit();
        goto retry;
    }
    if (next_val == val) {
        contained = 1;
        prev->next = next_next;
        next->next = (node_t*)NULL;
    }
    asf_commit(); /* atomic end */
    ...
    return contained;
}
```

- Easy lock-free removal from linked-list
- Atomically
 - ▶ Swing around pointer.
 - ▶ Monitor next-pointer of doomed element
- Simple adaption for doubly linked lists

Introduction to TinySTM

- Implementation of STM.
- Features:
 - ▶ Word-based with write locks. (Encounter time locking)
 - ▶ Time-based with eager validation.
 - ▶ Direct updates, redo-log. (WT version)
- Interface:
 - ▶ C library.
 - ▶ Start, abort and commit a transaction.
 - ▶ Transactional memory access with `stm_load` and `stm_store`.

Accelerating TinySTM

- General idea: Use ASF to track locations.
- Goal: `call stm_load → lock mov (%rsi), %rax`
- Limitations of ASF:
 - ▶ Limited **capacity**
 - ▶ Specification phase → less **flexibility**

Design Requirements

- Assumptions:

- ▶ Average TX size larger than ASF's capacity.
- ▶ Many read locations, few updates.

→ ASF also for large TXs

→ Integrated functionality

Acceleration Idea

- Use ASF's specification phase to initially track read-set.
- Hit capacity limit: Switch to original TinySTM for **subsequent** accesses.
- Benefits:
 - ▶ Less meta-data to keep.
 - ▶ Smaller validation overhead.

Design Requirements

- Assumptions:

- ▶ Average TX size larger than ASF's capacity.
- ▶ Many read locations, few updates.

→ ASF also for large TXs

→ Integrated functionality

Acceleration Idea

- Use ASF's specification phase to initially track read-set.
- Hit capacity limit: Switch to original TinySTM for **subsequent** accesses.
- Benefits:
 - ▶ Less meta-data to keep.
 - ▶ Smaller validation overhead.

Design Requirements

- Assumptions:

- ▶ Average TX size larger than ASF's capacity.
- ▶ Many read locations, few updates.

→ ASF also for large TXs

→ Integrated functionality

Acceleration Idea

- Use ASF's specification phase to initially track read-set.
- Hit capacity limit: Switch to original TinySTM for **subsequent** accesses.
- Benefits:
 - ▶ Less meta-data to keep.
 - ▶ Smaller validation overhead.

Steps of Transactional Load

Steps for `stm_load`

- 1 Check lock.
- 2 Check timestamp vs. TX's validity range.
- 3 Read value.
- 4 Re-read lock, ensure it is still free.
- 5 Add meta-data (location, version).

ASF vs. TinySTM, Revised

- TinySTM updates memory in-place.
- ASF detects these accesses.

→ No need to check locks in accelerated `stm_load`.

What if...

- the modifying TX aborts, after TX with accelerated load has committed?
- the accelerated TX propagates updated value to another location and commits?

ASF vs. TinySTM, Revised

- TinySTM updates memory in-place.
 - ASF detects these accesses.
- No need to check locks in accelerated `stm_load`.

What if...

- the modifying TX aborts, after TX with accelerated load has committed?
- the accelerated TX propagates updated value to another location and commits?

ASF vs. TinySTM, Revised

- TinySTM updates memory in-place.
 - ASF detects these accesses.
- No need to check locks in accelerated `stm_load`.

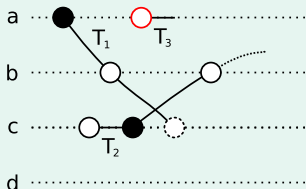
What if...

- the modifying TX aborts, after TX with accelerated load has committed?
- the accelerated TX propagates updated value to another location and commits?

ASF vs. TinySTM, Revised

- TinySTM updates memory in-place.
 - ASF detects these accesses.
- No need to check locks in accelerated `stm_load`.

What if...

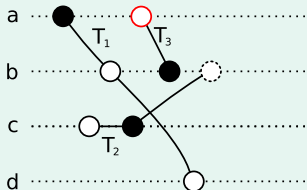


- the modifying TX aborts, after TX with accelerated load has committed?
- the accelerated TX propagates updated value to another location and commits?

ASF vs. TinySTM, Revised

- TinySTM updates memory in-place.
 - ASF detects these accesses.
- No need to check locks in accelerated `stm_load`.

What if...



- the modifying TX aborts, after TX with accelerated load has committed?
- the accelerated TX propagates updated value to another location and committs?

The Need for Overhead

- Check that location is not locked.
- Validate ASF's read-set in specification phase.
- Add `VALIDATE ASF` instruction.
- Check lock and validity in accelerated `stm_load`.
- Validate ASF's read-set in `stm_validate` with `VALIDATE`.

Steps for accelerated `stm_load`

- 1 Check capacity.
- 2 Read the location with `LOCK MOV`.
- 3 Read the lock.
- 4 `VALIDATE` the read location.

The Need for Overhead

- Check that location is not locked.
- Validate ASF's read-set in specification phase.
- Add `VALIDATE ASF` instruction.
- Check lock and validity in accelerated `stm_load`.
- Validate ASF's read-set in `stm_validate` with `VALIDATE`.

Steps for accelerated `stm_load`

- 1 Check capacity.
- 2 Read the location with `LOCK MOV`.
- 3 Read the lock.
- 4 `VALIDATE` the read location.

Putting it all Together

```
stm_load(addr) {
    ...
    res = asf_lock_load(addr);
    if (locked(addr)) stm_abort();
    if (!asf_validate()) stm_abort();
}

stm_validate() {
    ...
    if (!asf_validate()) stm_abort();
    ...
}

stm_commit() {
    ...
    if (!asf_validate()) stm_abort();
    // if (!last_acquire(N)) stm_abort();
    asf_commit();
    ...
}

stm_start() {
    ...
}

stm_store(addr, val) {
    ...
}
```

Putting it all Together

```
stm_load(addr) {
    ...
    res = asf_lock_load(addr);
    if (locked(addr)) stm_abort();
    if (!asf_validate()) stm_abort();
}

stm_validate() {
    ...
    if (!asf_validate()) stm_abort();
    ...
}

stm_commit() {
    ...
    if (!asf_validate()) stm_abort();
    // if (!asf_acquire(N)) stm_abort();
    asf_commit();
    ...
}

stm_start() {
    ...
}

stm_store(addr, val) {
    ...
}
```

Putting it all Together

```
stm_load(addr) {
    ...
    res = asf_lock_load(addr);
    if (locked(addr)) stm_abort();
    if (!asf_validate()) stm_abort();
}

stm_validate() {
    ...
    if (!asf_validate()) stm_abort();
    ...
}

stm_commit() {
    ...
    if (!asf_validate()) stm_abort();
    // if (!asf_acquire(N)) stm_abort();
    asf_commit();
    ...
}

stm_start() {
    ...
}

stm_store(addr, val) {
    ...
}
```

Putting it all Together

```
stm_load(addr) {
    ...
    res = asf_lock_load(addr);
    if (locked(addr)) stm_abort();
    if (!asf_validate()) stm_abort();
}

stm_validate() {
    ...
    if (!asf_validate()) stm_abort();
    ...
}

stm_commit() {
    ...
    if (!asf_validate()) stm_abort();
    // if (!asf_acquire(N)) stm_abort();
    asf_commit();
    ...
}

stm_start() {
    ...
}

stm_store(addr, val) {
    ...
}
```

Outline

- 1 Introduction
 - Motivation
 - Advanced Synchronization Facility
- 2 Acceleration with ASF
 - Lock-Free Linked Lists
 - TinySTM
- 3 Evaluation
 - Setup
 - Accelerated Lock-Free Data Structures
 - Accelerated TinySTM

Evaluation Setup

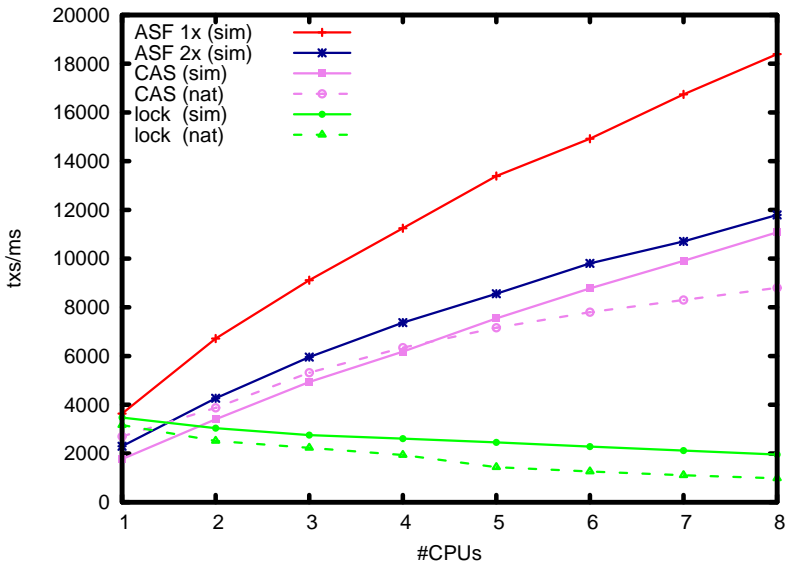
● Benchmark

- ▶ Integer set, interface: add(), remove(), contains()
- ▶ Data structures:
 - ★ Sorted linked list
 - ★ Red-black tree
- ▶ Implementations:
 - ★ Lock-free using CAS (T. Harris)
 - ★ Lock-free using ASF
 - ★ Global lock
 - ★ Transactified, vanilla TinySTM
 - ★ Transactified, ASF-accelerated TinySTM

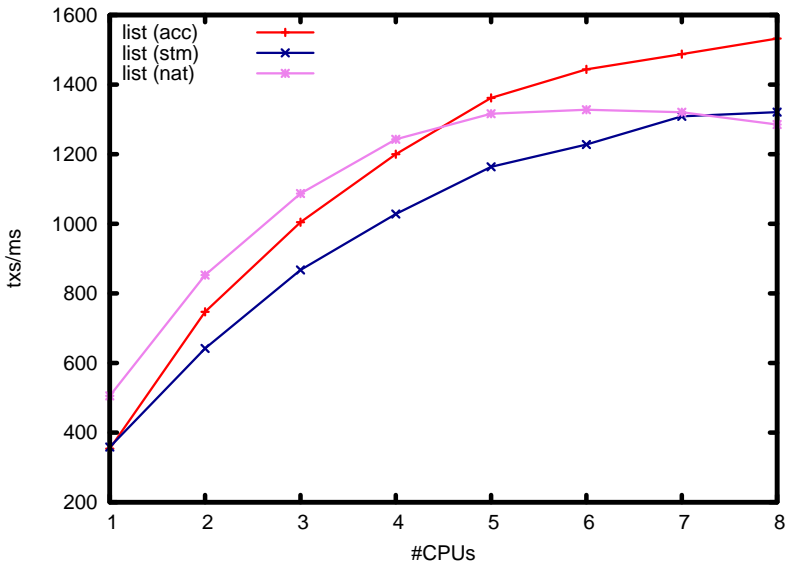
● Environment

- ▶ Native: 2x AMD Quad-Core Opteron “Barcelona”
- ▶ Simulator: 8 core custom PTLsim model

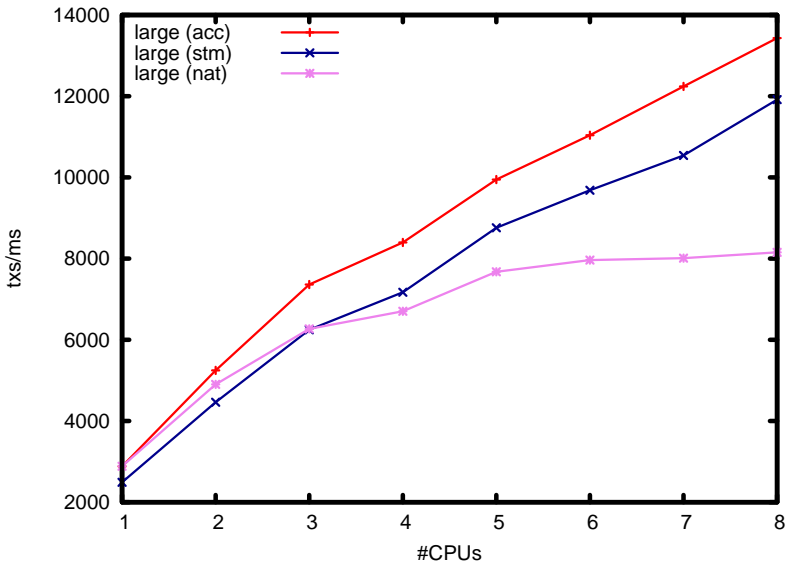
Plain Sorted Linked List



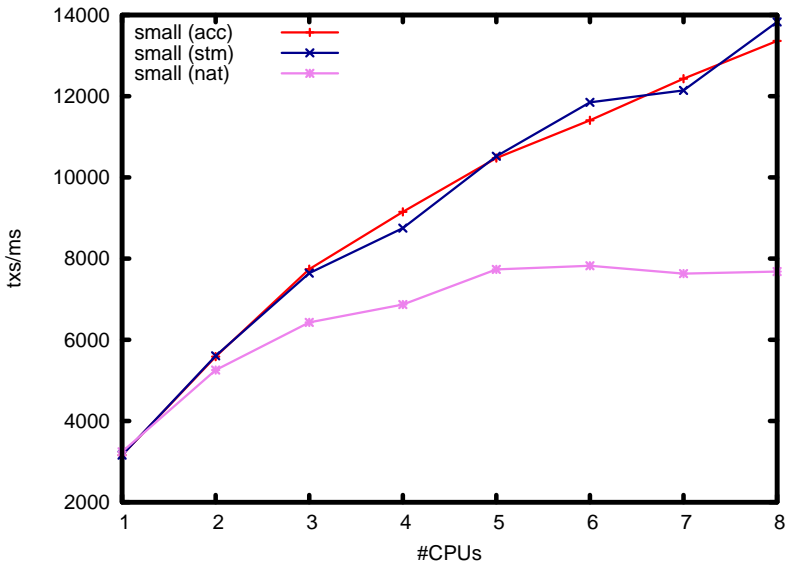
Transactified Linked List



Transactified Red-Black Tree (256 entries)



Transactified Red-Black Tree (128 entries)



Summary

- ASF **accelerates** TinySTM and lock-free data-structures.
- It is available in a **cycle-accurate** simulator.

- Outlook
 - ▶ More experiments with acceleration.
 - ▶ Detailed run-time breakdown of TinySTM.
 - ▶ Clean up, improve and release modifications to PTLsim.

Outline

4

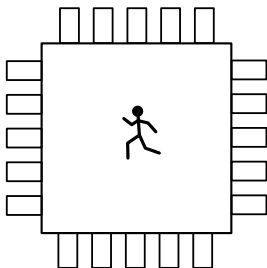
Backup

- Motivation Multi-Core
- Contributions
- Implementation - Problems and Optimisation
- Simulator Precision

The Problem of Synchronisation

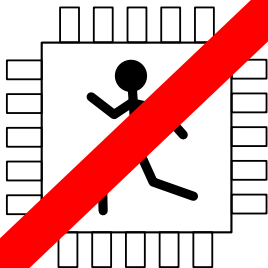
- No faster single cores.
- Increase core count per chip / system.
- Problem:
 - ▶ Split problem and solution—parallelise
 - ▶ Need for synchronisation.

The Problem of Synchronisation



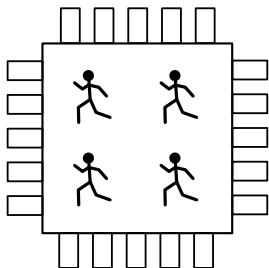
- No faster single cores.
- Increase core count per chip / system.
- Problem:
 - ▶ Split problem and solution—parallelise
 - ▶ Need for synchronisation.

The Problem of Synchronisation



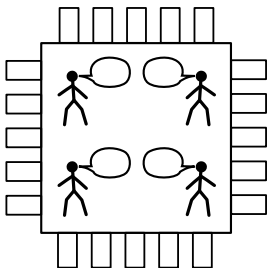
- No faster single cores.
- Increase core count per chip / system.
- Problem:
 - ▶ Split problem and solution—parallelise
 - ▶ Need for synchronisation.

The Problem of Synchronisation



- No faster single cores.
- Increase core count per chip / system.
- Problem:
 - ▶ Split problem and solution—**parallelise**
 - ▶ Need for **synchronisation**.

The Problem of Synchronisation



- No faster single cores.
- Increase core count per chip / system.
- Problem:
 - ▶ Split problem and solution—**parallelise**
 - ▶ Need for **synchronisation**.

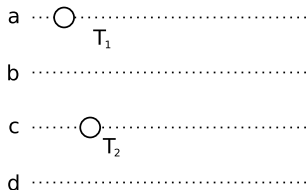
Contributions

- Advanced Synchronization Facility:
 - ▶ Implementation in timing-accurate simulator PTLsim.
 - ▶ Proof of coherence.
 - ▶ Ordering constraints for correct out-of-order execution.
 - ▶ Fast and simple lock-free linked lists.
 - ▶ Acceleration of TinySTM.
- PTLsim:
 - ▶ Improved core, cache and interconnect models.
 - ▶ Bug-fixing and tuning.
 - ▶ Accuracy analysis.
- Evaluation

Contributions: PTLsim

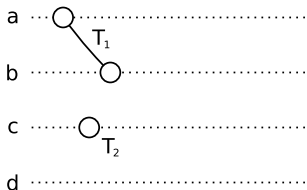
Feature	Original Model	Improved Model	AMD Barcelona
Core Model	single-core, SMT	SMP	multi-core, ccNUMA
Interconnect Latency Bandwidth	none	simplified zero infinite	Hyper-Transport topology dependent limited
L1 D-Cache L3 Cache Hierarchy TLB HW Prefetcher	physically indexed single inclusive L1 none	virtually indexed private to core inclusive L1 & L2 preliminary support	virtually indexed shared in package exclusive, adaptive L3 L1 & L2 adaptive prefetchers
Atomic RMW Memory Consistency	internal locks immediate visibility	internal locks immediate with cache co-herency	two-phase, optimistic processor consistency with MOESI cache coherency

Understanding the Transactional Load



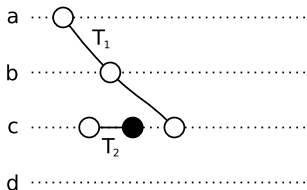
- **Invisible** read: Readers do not enqueue / lock.
 - (Re-)validation by reading TXs:
 - ▶ Only if read version “too new”.
 - ▶ Extend TX's **validity interval**.
 - ▶ Check all read locations for original version.
- Store **meta-data** for each location.
- Eager validation with reduced overhead.

Understanding the Transactional Load



- **Invisible** read: Readers do not enqueue / lock.
 - (Re-)validation by reading TXs:
 - ▶ Only if read version “too new”.
 - ▶ Extend TX's **validity interval**.
 - ▶ Check all read locations for original version.
- Store **meta-data** for each location.
- Eager validation with reduced overhead.

Understanding the Transactional Load

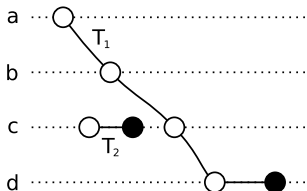


- **Invisible** read: Readers do not enqueue / lock.
- (Re-)validation by reading TXs:
 - ▶ Only if read version “too new”.
 - ▶ Extend TX's **validity interval**.
 - ▶ Check all read locations for original version.

→ Store **meta-data** for each location.

→ Eager validation with reduced overhead.

Understanding the Transactional Load

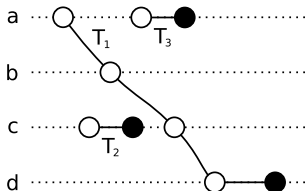


- **Invisible** read: Readers do not enqueue / lock.
- (Re-)validation by reading TXs:
 - ▶ Only if read version “too new”.
 - ▶ Extend TX’s **validity interval**.
 - ▶ Check all read locations for original version.

→ Store **meta-data** for each location.

→ Eager validation with reduced overhead.

Understanding the Transactional Load

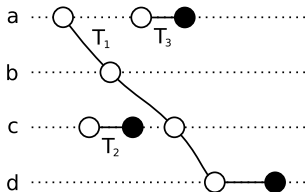


- **Invisible** read: Readers do not enqueue / lock.
- (Re-)validation by reading TXs:
 - ▶ Only if read version “too new”.
 - ▶ Extend TX’s **validity interval**.
 - ▶ Check all read locations for original version.

→ Store **meta-data** for each location.

→ Eager validation with reduced overhead.

Understanding the Transactional Load



- **Invisible** read: Readers do not enqueue / lock.
 - (Re-)validation by reading TXs:
 - ▶ Only if read version “too new”.
 - ▶ Extend TX’s **validity interval**.
 - ▶ Check all read locations for original version.
- Store **meta-data** for each location.
- Eager validation with reduced overhead.

Improve Capacity Utilisation

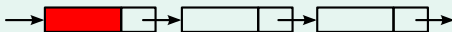
- ASF monitors 8 full cache-lines of 64 bytes each.
- Detect words in same cache-line.
- Only one entry in ASF read-set.
- Increase capacity by recording **last** address.

Example: Traversing Linked-List

Improve Capacity Utilisation

- ASF monitors 8 full cache-lines of 64 bytes each.
 - Detect words in same cache-line.
- Only one entry in ASF read-set.
- Increase capacity by recording **last** address.

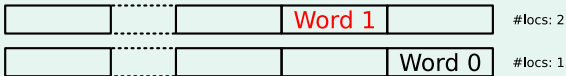
Example: Traversing Linked-List



Improve Capacity Utilisation

- ASF monitors 8 full cache-lines of 64 bytes each.
 - Detect words in same cache-line.
- Only one entry in ASF read-set.
- Increase capacity by recording **last** address.

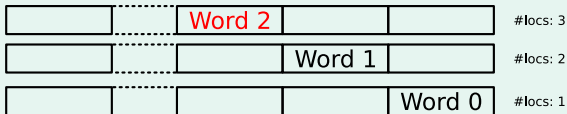
Example: Traversing Linked-List



Improve Capacity Utilisation

- ASF monitors 8 full cache-lines of 64 bytes each.
 - Detect words in same cache-line.
- Only one entry in ASF read-set.
- Increase capacity by recording **last** address.

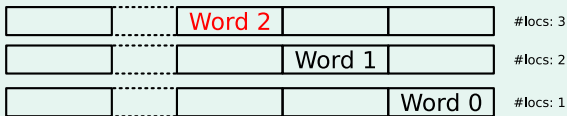
Example: Traversing Linked-List



Improve Capacity Utilisation

- ASF monitors 8 full cache-lines of 64 bytes each.
 - Detect words in same cache-line.
- Only one entry in ASF read-set.
- Increase capacity by recording **last** address.

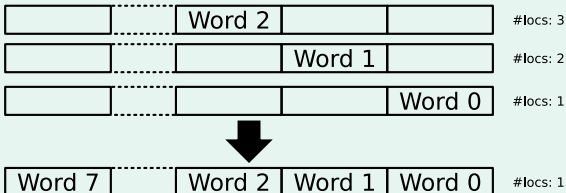
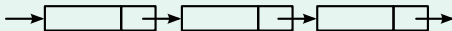
Example: Traversing Linked-List



Improve Capacity Utilisation

- ASF monitors 8 full cache-lines of 64 bytes each.
 - Detect words in same cache-line.
- Only one entry in ASF read-set.
- Increase capacity by recording **last** address.

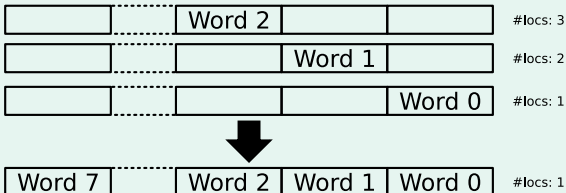
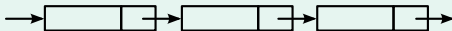
Example: Traversing Linked-List



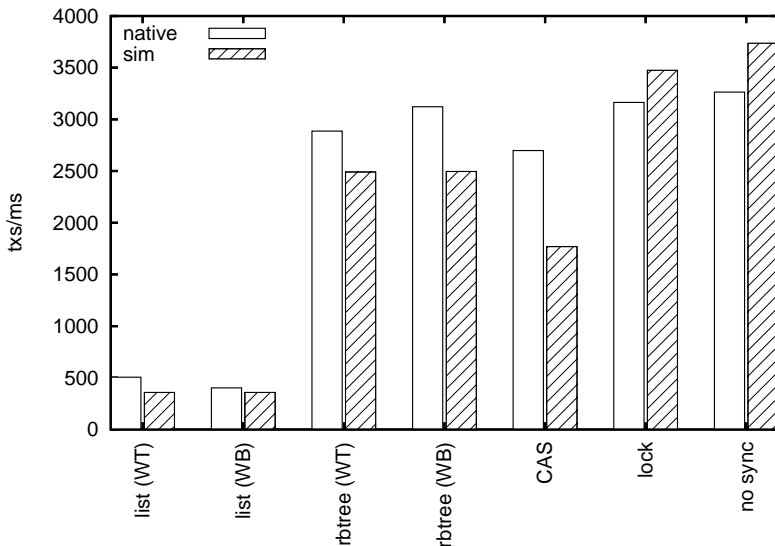
Improve Capacity Utilisation

- ASF monitors 8 full cache-lines of 64 bytes each.
 - Detect words in same cache-line.
- Only one entry in ASF read-set.
- Increase capacity by recording **last** address.

Example: Traversing Linked-List



Simulator Precision - Single Core



Simulator Precision - Multi Core

