Evaluation of AMD's Advanced Synchronization Facility Within a Complete Transactional Memory Stack

Dave Christie
Jae-Woong Chung
Stephan Diestelhorst
Michael Hohmuth
Martin Pohlack

Advanced Micro Devices, Inc.

Christof Fetzer
Martin Nowack
Torvald Riegel

Technische Universität Dresden

Pascal Felber
Patrick Marlier
Etienne Riviere

Université de Neuchâtel

Transactional Memory (TM)

- Multi-core everywhere, need parallel software
- Often, parallel threads need to synchronize over shared memory

- Current synchronization mechanisms (locks, ...) not really suitable
  - Every programmer and every program is affected → too difficult
  - Locks: deadlocks, relies on conventions, not composable

Transactional Memory (TM)

- Multi-core everywhere, need parallel software
- Often, parallel threads need to synchronize over shared memory

- Current synchronization mechanisms (locks, ...) not really suitable
  - Every programmer and every program is affected → too difficult
  - Locks: deadlocks, relies on conventions, not composable

- **TM: programmer declares, generic TM runtime system implements**
  - C++: `__transaction { x = map.remove(key); x.refCount--; }`
  - Compiler transforms code so that it uses TM for memory accesses
  - TM runtime is a software/hardware/hybrid implementation (STM/HTM/HyTM)

Our contribution: Realistic HTM in a realistic TM stack

- What can we expect from first-generation hardware TM (HTM) support?
- Properties of current systems shape first-gen HTM support!

Our contribution:
Realistic HTM in a realistic TM stack

- What can we expect from first-generation hardware TM (HTM) support?
- Properties of current systems shape first-gen HTM support!

- Realistic HW support: AMD's **Advanced Synchronization Facility** (ASF)
  - x86_64 extension for lock-free programming and TM
  - Designed to be feasible to implement in high-volume microprocessors

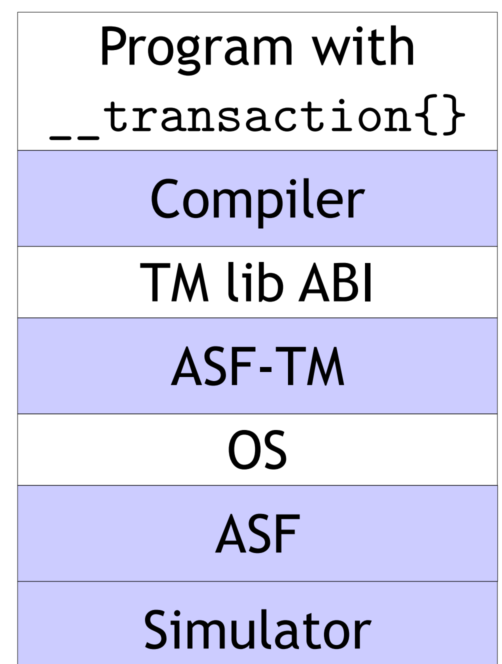
Our contribution:
Realistic HTM in a realistic TM stack

- What can we expect from first-generation hardware TM (HTM) support?
- Properties of current systems shape first-gen HTM support!

- Realistic HW support: AMD's **Advanced Synchronization Facility** (ASF)
  - x86_64 extension for lock-free programming and TM
  - Designed to be feasible to implement in high-volume microprocessors

- Realistic **TM-aware system stack**:
  - C/C++ transaction statements (`__transaction{}`)
  - Dresden TM compiler, based on gcc + LLVM
  - Generic TM library interface (ABI)
  - TM library implemented using ASF

- Evaluation in near-cycle-**accurate simulator**
  - Models x86 with ASF at a high level of detail



Advanced Synchronization Facility (ASF)

- Proposal: Not announced for future products
- ASF provides **Speculative Regions** (SRs)
  - Similar to transactions: SPECULATE, COMMIT
  - Speculative (LOCK MOV) and nonspeculative loads/stores allowed (selective annotation)

```
DCAS:
 MOV R8, RAX
 MOV R9, RBX
retry:
 SPECULATE
 JNZ retry
 MOV RCX, 1
 LOCK MOV R10, [mem1]
 LOCK MOV RBX, [mem2]
 CMP R8, R10
 JNZ out
 CMP R9, RBX
 JNZ out
 LOCK MOV [mem1], RDI
 LOCK MOV [mem2], RSI
 XOR RCX, RCX
out:
 COMMIT
 MOV RAX, R10
```

Advanced Synchronization Facility (ASF)

- Proposal: Not announced for future products
- ASF provides **Speculative Regions** (SRs)
  - Similar to transactions: SPECULATE, COMMIT
  - Speculative (LOCK MOV) and nonspeculative loads/stores allowed (selective annotation)
  - Speculative access
    → ASF monitors cacheline (R/W, W/W conflicts)
  - SR aborts on conflicts, exceeded capacity, far jumps, disallowed instructions

```
DCAS:
 MOV R8, RAX
 MOV R9, RBX
retry:
 SPECULATE
 JNZ retry
 MOV RCX, 1
 LOCK MOV R10, [mem1]
 LOCK MOV RBX, [mem2]
 CMP R8, R10
 JNZ out
 CMP R9, RBX
 JNZ out
 LOCK MOV [mem1], RDI
 LOCK MOV [mem2], RSI
 XOR RCX, RCX
out:
 COMMIT
 MOV RAX, R10
```

Advanced Synchronization Facility (ASF)

- Proposal: Not announced for future products

- ASF provides **Speculative Regions** (SRs)
  - Similar to transactions: SPECULATE, COMMIT
  - Speculative (LOCK MOV) and nonspeculative loads/stores allowed (selective annotation)
  - Speculative access → ASF monitors cacheline (R/W, W/W conflicts)
  - SR aborts on conflicts, exceeded capacity, far jumps, disallowed instructions
  - Simple guarantees:
    - Minimal capacity
    - SR will eventually commit (unless contention / exceeded capacity / far jumps /disallowed)

```
DCAS:
 MOV R8, RAX
 MOV R9, RBX
retry:
 SPECULATE
 JNZ retry
 MOV RCX, 1
 LOCK MOV R10, [mem1]
 LOCK MOV RBX, [mem2]
 CMP R8, R10
 JNZ out
 CMP R9, RBX
 JNZ out
 LOCK MOV [mem1], RDI
 LOCK MOV [mem2], RSI
 XOR RCX, RCX
out:
 COMMIT
 MOV RAX, R10
```

Constraints for ASF's design

- **Realities** in the development of high-volume microprocessors
  - Costs of chip area and verification
  - Only incremental changes feasible
  - Existing CPUs are complex: out-of-order execution, ...
  - HTM touches many sensitive areas
  - All corner cases have to be handled
  - Backward/forward compatibility (code, architecture)

Constraints for ASF's design

- **Realities** in the development of high-volume microprocessors
  - Costs of chip area and verification
  - Only incremental changes feasible
  - Existing CPUs are complex: out-of-order execution, ...
  - HTM touches many sensitive areas
  - All corner cases have to be handled
  - Backward/forward compatibility (code, architecture)

- High-level **ASF design constraints**
  - No change to cache-coherence protocol
  - No transaction virtualization
  - Don't change behaviour of nonspeculative code (e.g., loads/stores)
  - Keep instruction set additions small
  - Keep cost of first-generation TM extensions small
  - Enable further use cases (minimal capacity for lock-free programming)

ASF Implementations

- Dedicated storage: Locked Line Buffer (LLB)
- Augmented cache: Speculative load/store bits for cachelines
- Different capacity limitations:
  - LLB size vs. cache size/associativity

- Our study:
  - LLB, optionally use L1 cache for loads
    → LLB-8, LLB-256
    → LLB-8 w/ L1, LLB-256 w/ L1

- Providing ASF's guarantees is nontrivial
  - Capacity: mispredicted branches leading to additional loads
  - Progress: Pagefaults abort SRs, but OS should see pagefaults

Dresden TM Compiler (DTMC)

Source code	Transformed to use TM ABI	Binary after LTO
`extern long cntr;`	`extern long cntr;`	`; mem1 for cntr`
`void increment() {`	`void increment() {`	
`__transaction {`	`_ITM_beginTransaction(...);`	`SPECULATE`
		`JNZ handle_abort`
`cntr = cntr + 5;`	`long l_cntr = _ITM_R8(&cntr);`	`LOCK MOV RCX, [mem1]`
	`l_cntr = l_cntr + 5;`	`ADD RCX, 5`
	`_ITM_W8(&cntr, l_cntr);`	`LOCK MOV [mem1], RCX`
`}}`	`_ITM_commitTransaction(); }`	`COMMIT`

- Compiler instruments only accesses to shared memory
  - Exploits ASF's selective annotation (no capacity wasted for stack)
- Generic TM ABI important
  - Allows cross-vendor compatibility + dynamic linking
- Compiler uses link-time optimization (LTO)
  - Can do whole-program analysis/transformation/optimization

ASF-TM

- TM runtime library
  - Uses either ASF or simple software fallback (serial execution)

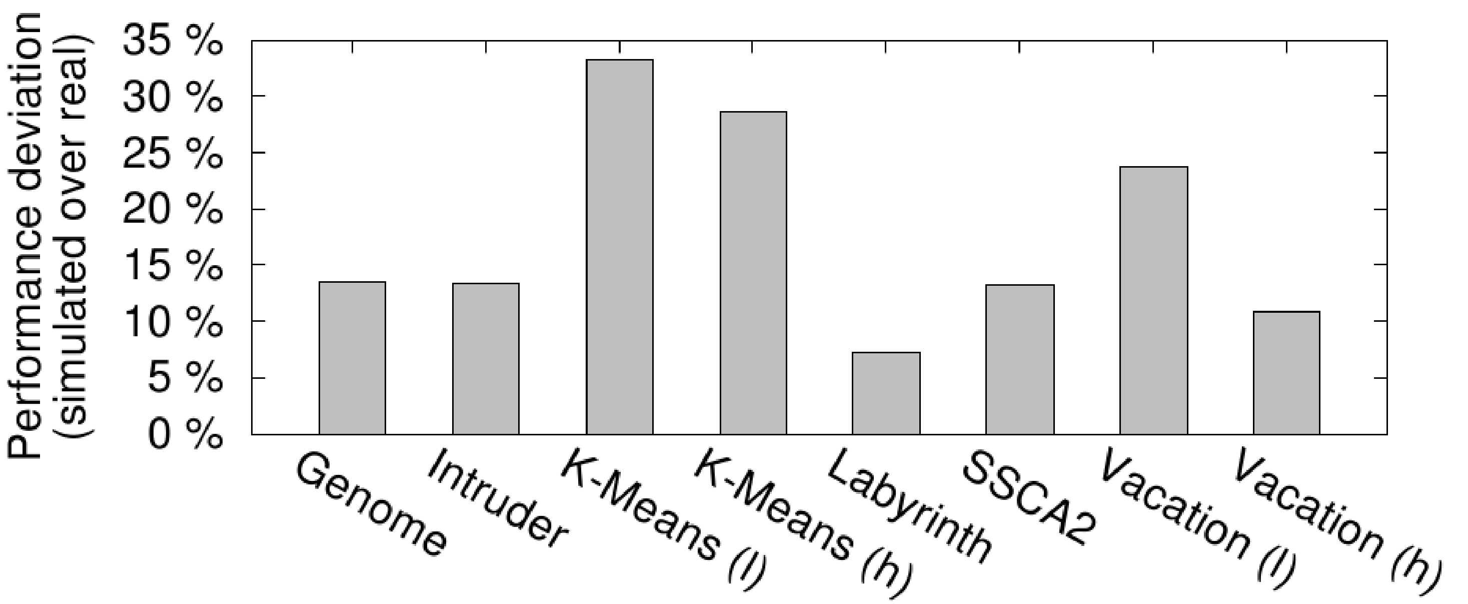
- Some TM functions need software aids
  - Begin: ASF SPECULATE + software setjmp + support for nesting, serial
  - Commit: ASF COMMIT + support for nesting, serial
  - Load/store functions: just use ASF's speculative accesses

ASF-TM

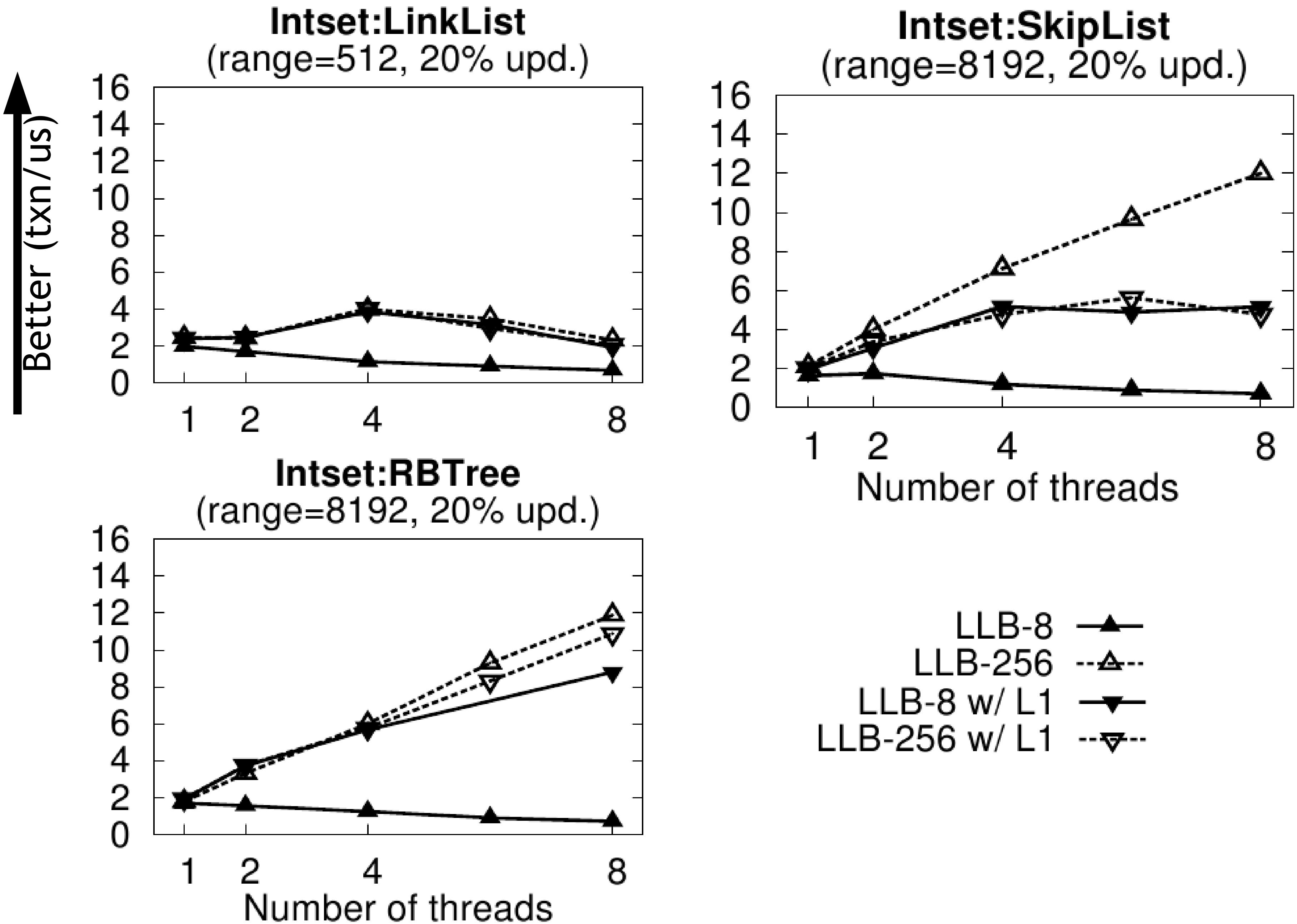
- TM runtime library
  - Uses either ASF or simple software fallback (serial execution)
- Some TM functions need software aids
  - Begin: ASF SPECULATE + software setjmp + support for nesting, serial
  - Commit: ASF COMMIT + support for nesting, serial
  - Load/store functions: just use ASF's speculative accesses
- Calling libraries: Memory allocator as an example
  - Asynchronous aborts of SRs: Can't use malloc/free/... as is even though they're thread-safe
  - Currently we use custom pre-allocation
  - We could as well let the compiler instrument malloc with ASF-TM

Evaluation: Simulator

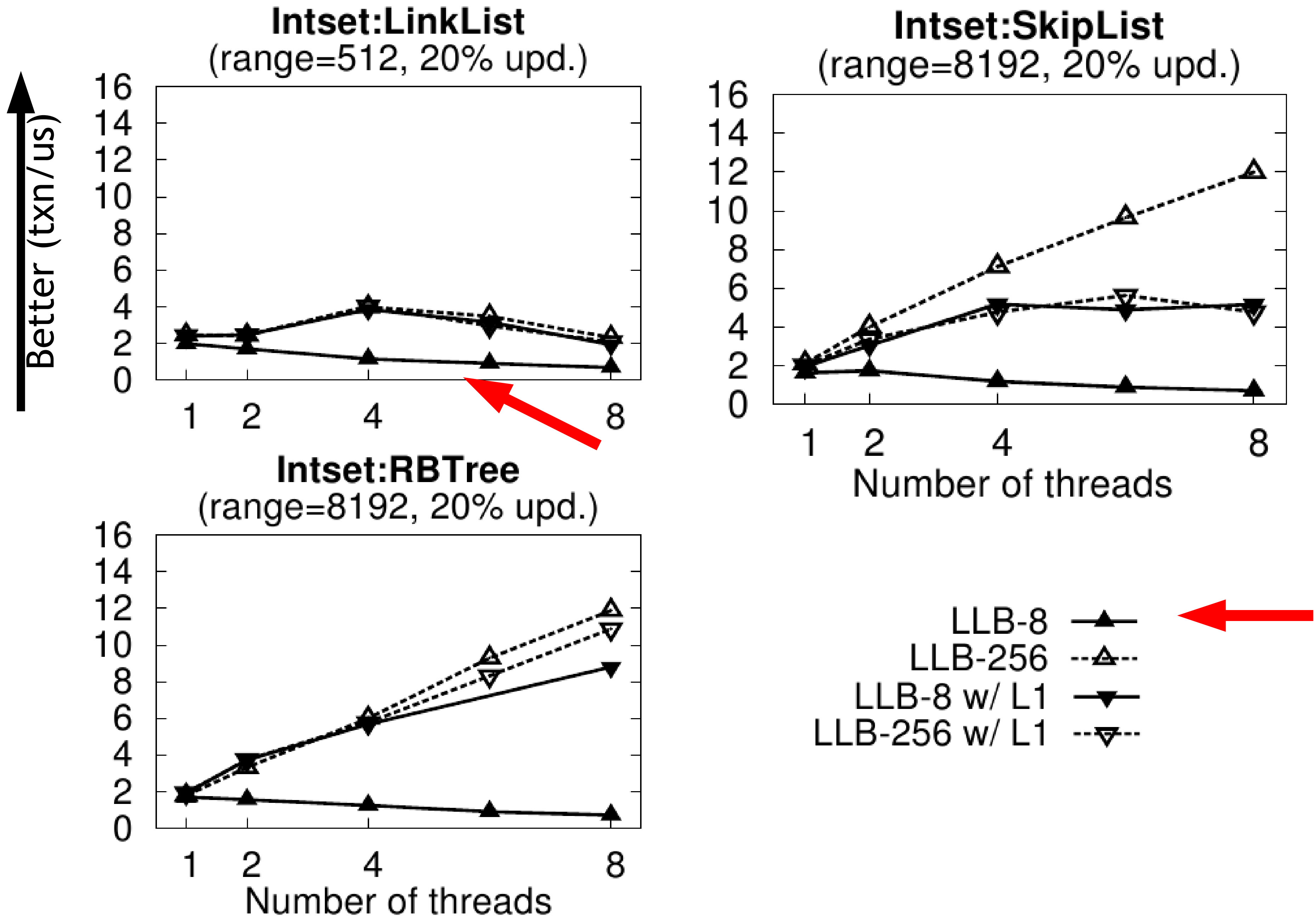
- PTLSim with ASF extensions
  - Highly detailed x86_64 simulation (out-of-order execution, …)
  - Close to AMD and Intel architectures
- Single-socket 2.1GHz 8-core simulated
- All experiments performed in the simulator
- Microbenchmarks, STAMP TM benchmark suite (use `__transaction{}`, compiled with DTMC)



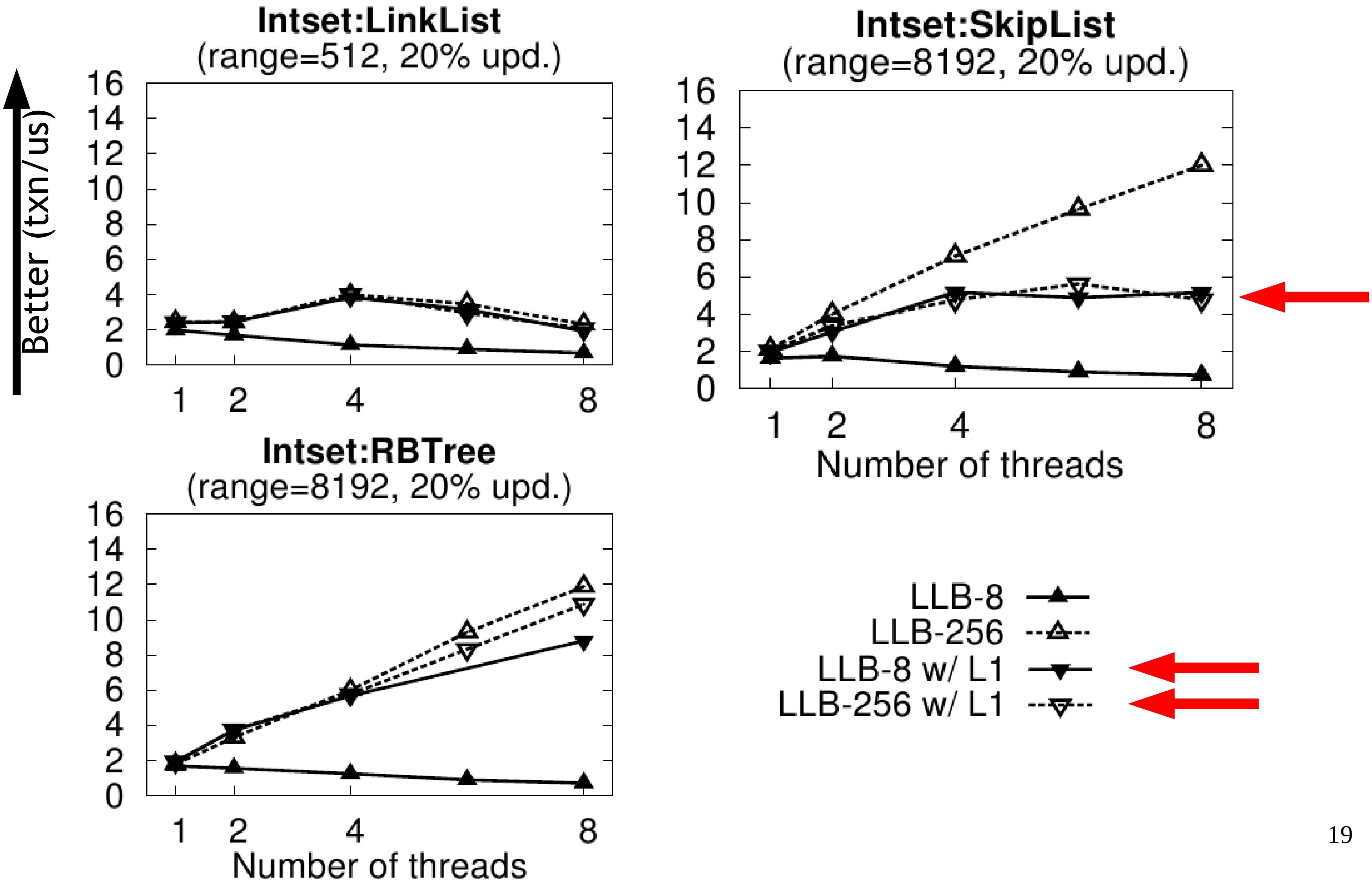
Microbenchmarks: Scalability



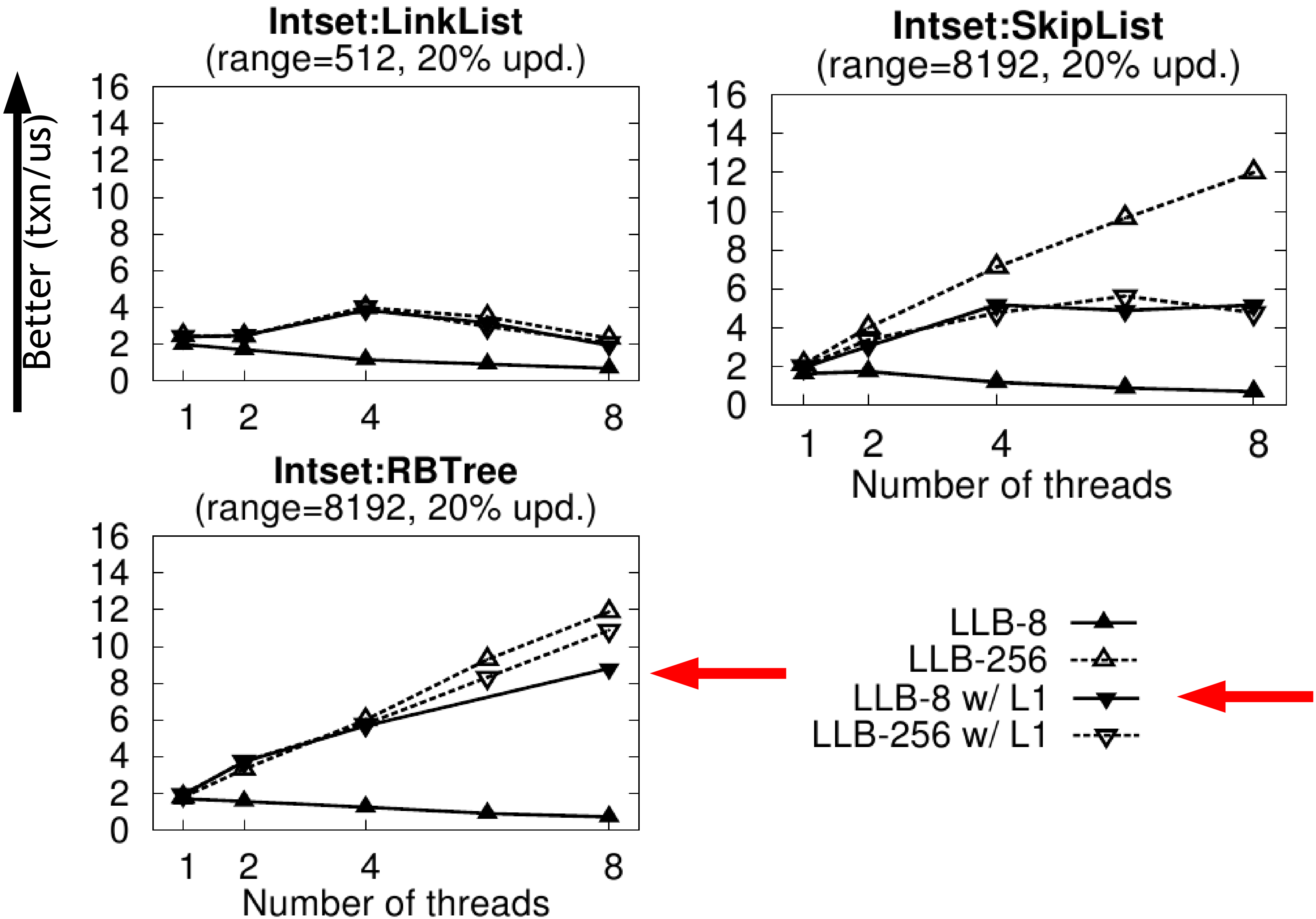
Microbenchmarks: Scalability



Microbenchmarks: Scalability

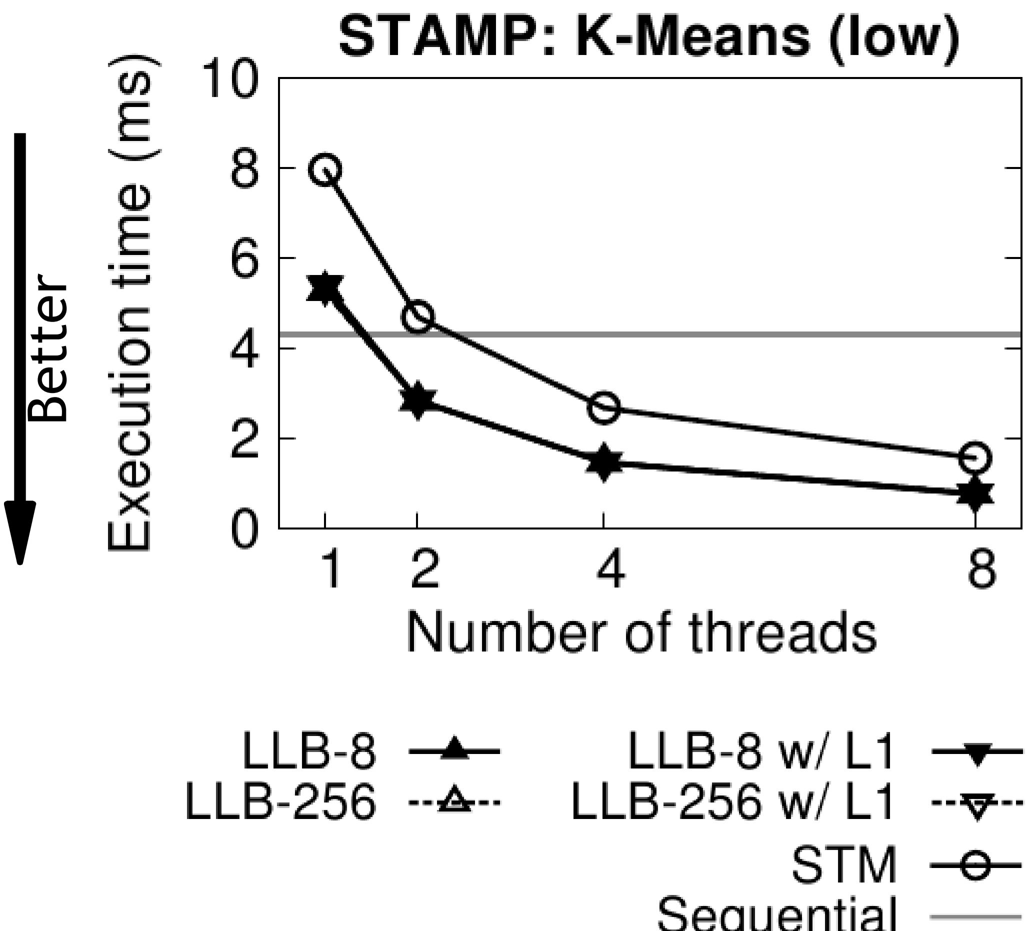


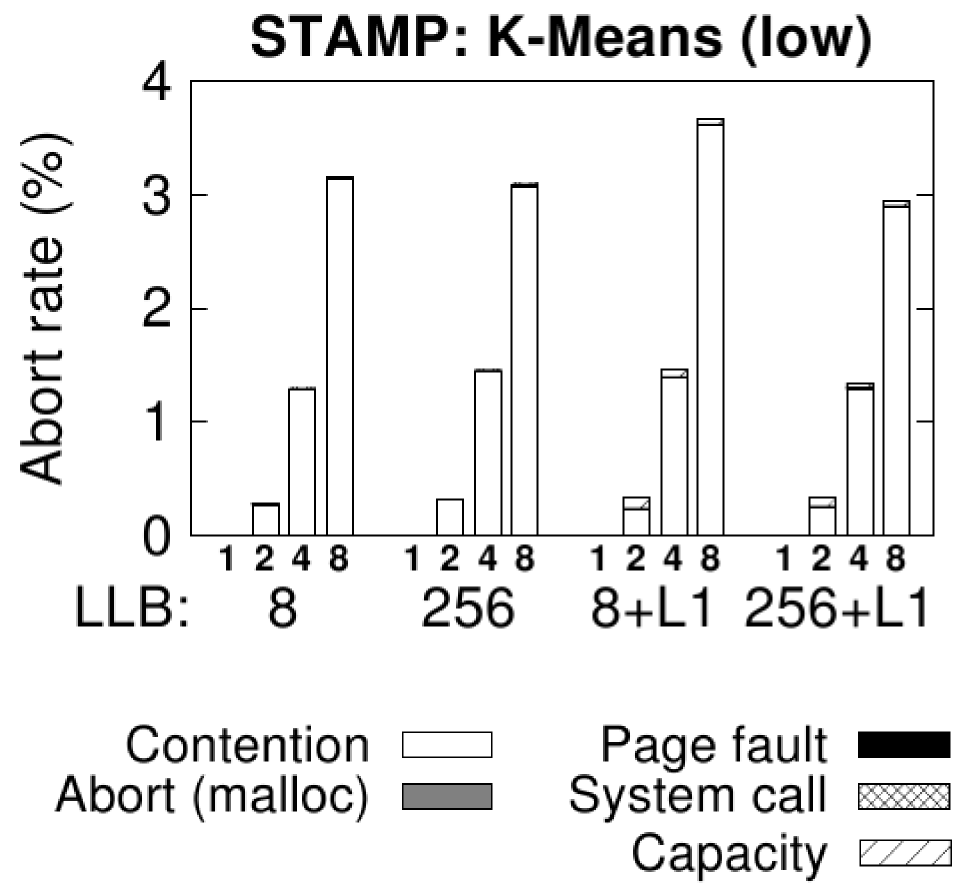
Microbenchmarks: Scalability



STAMP Benchmark Suite: KMeans

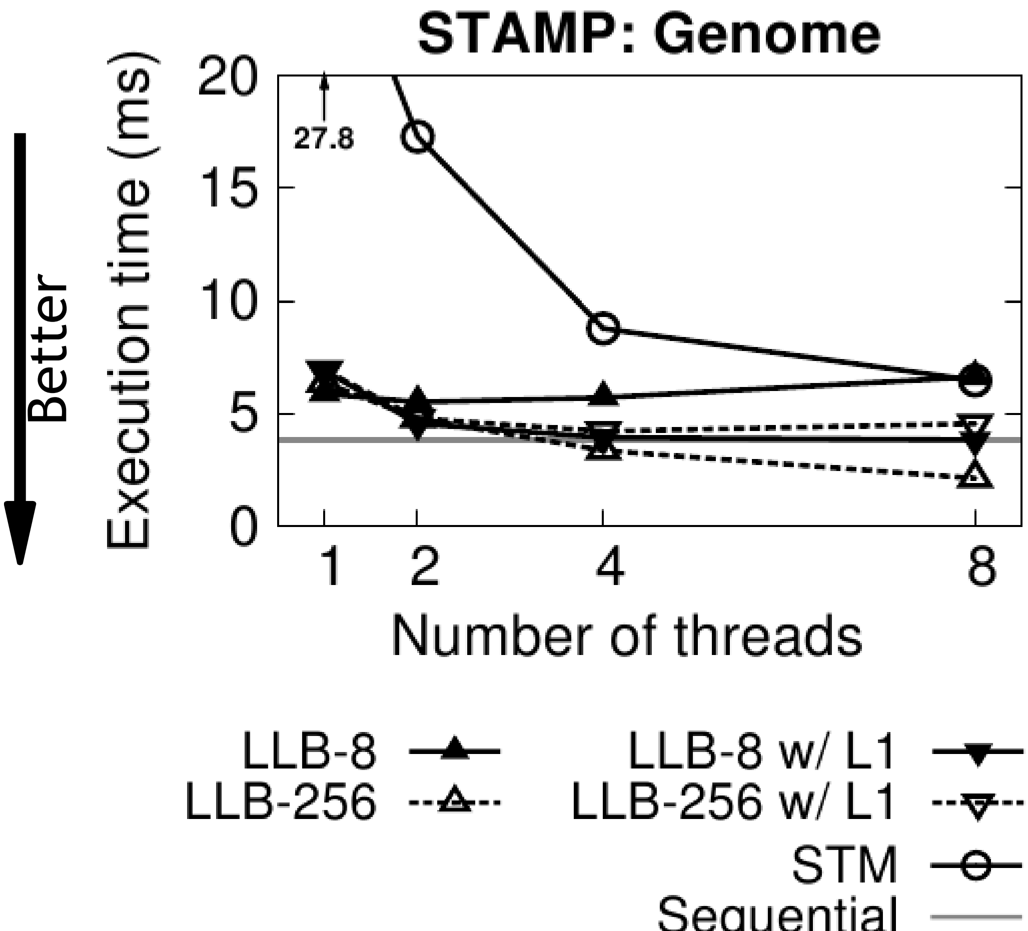
- Scales well, very few aborts
- STM is pretty good compared to sequential

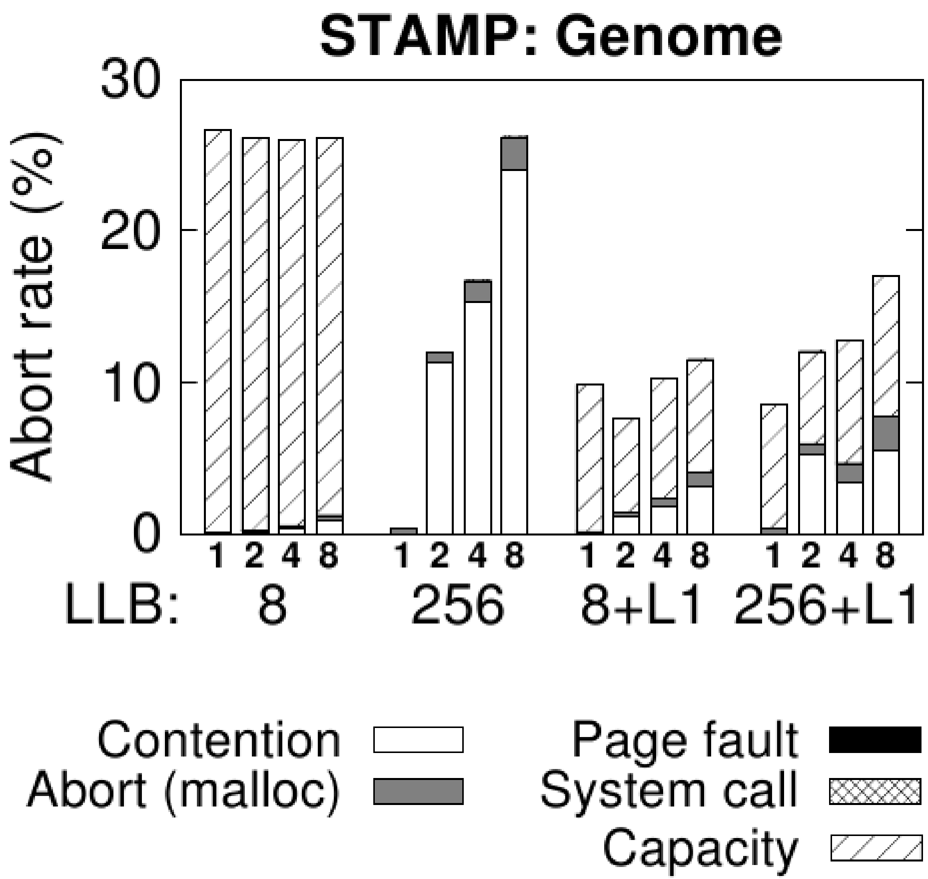




STAMP Benchmark Suite: Genome

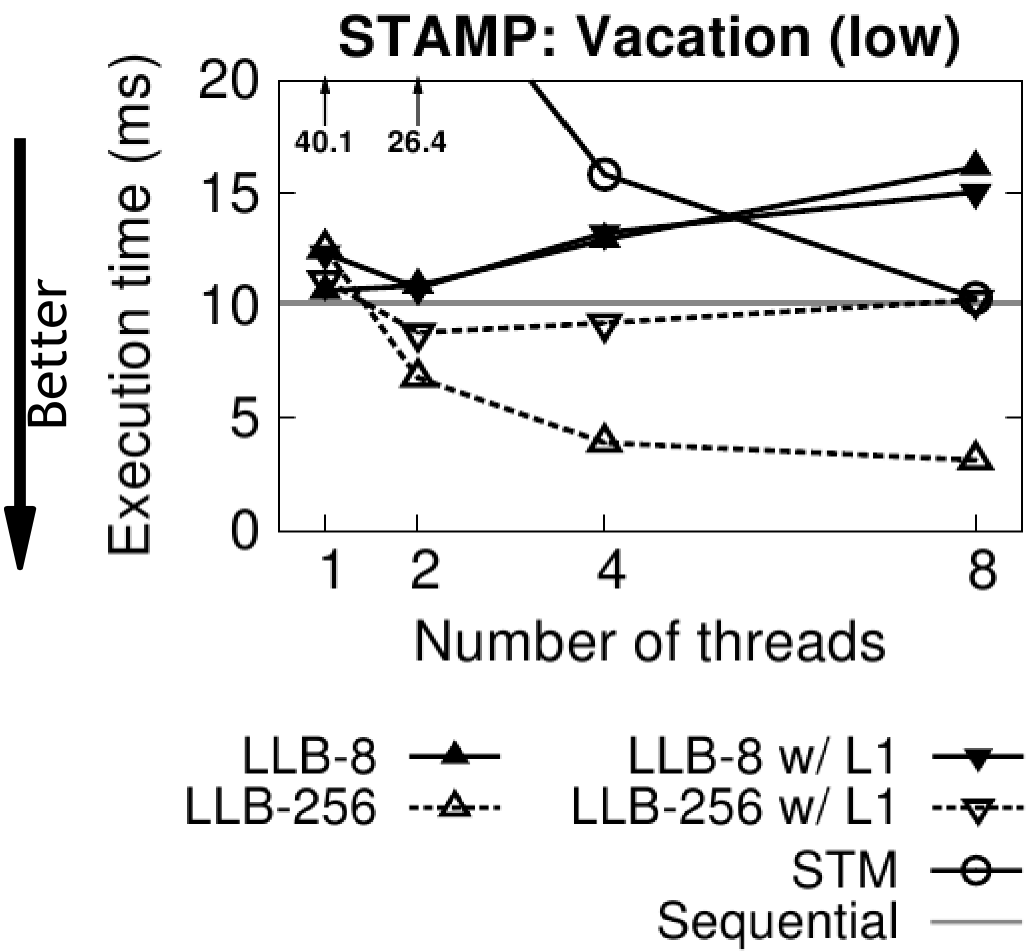
- STM overhead is much larger
- LLB-8 has not enough capacity
- LLB-8/256+L1 also have capacity aborts (cache associativity)

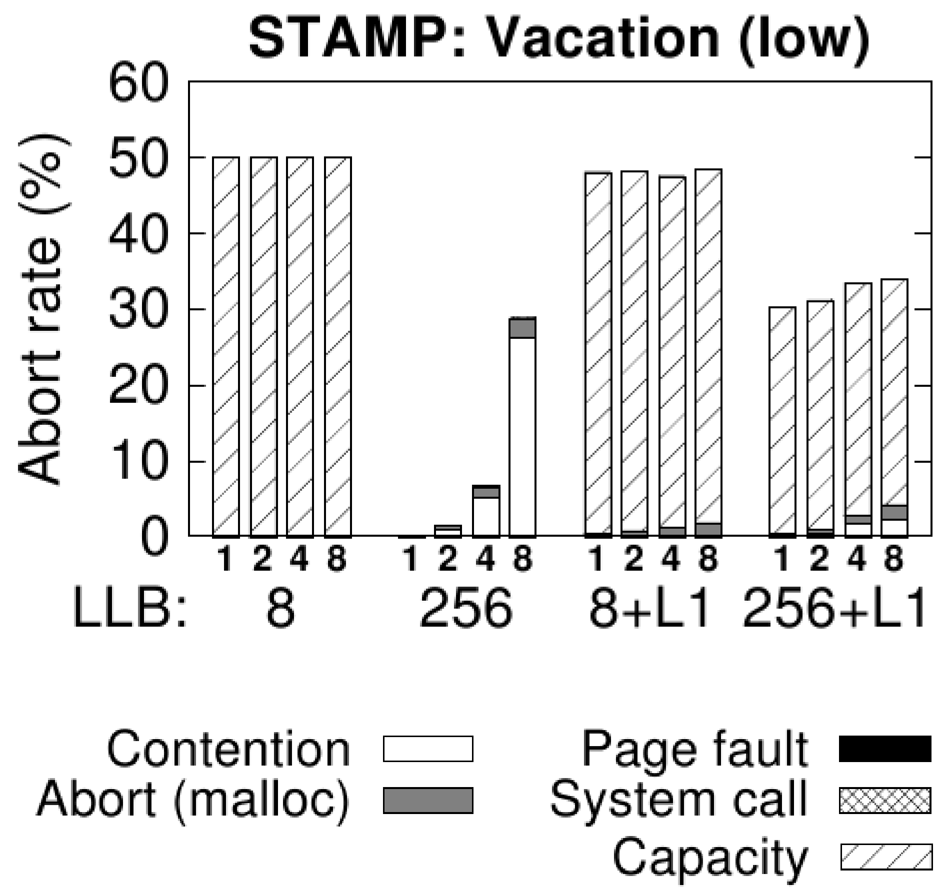




STAMP Benchmark Suite: Vacation

- LLB-8(+L1): more than 8 modified cachelines per transaction
- LLB-256+L1 scalability limited by cache associativity





Conclusions

- ASF is a proposal by industry for realistic first-gen HW TM support
- Often sufficient to get good TM performance
- Lots of systems work on higher layers (TM library, compiler, ...)
- Full-stack TM research necessary to build ready-to-use TM systems
- Open source releases: PTLSim-ASF, Dresden TM Compiler

http://amd64.org/research/multi-and-manycore-systems.html
http://tm.inf.tu-dresden.de
http://tmware.org
http://velox-project.eu