
ORACLE®

Scalable Transactions on NUMA systems

Trevor Brown

Joint work with Alex Kogan, Yossi Lev, Victor Luchangco

ORACLE

Large scale systems with HTM

- Hardware transactional memory (HTM) is starting to become a reality
- Intel 4-core processor with HTM in 2013
- New Intel 18-core processor with HTM (Q3-2014)
  - Can be configured with two sockets and 36-cores
  - First commodity processor that supports multiple sockets and HTM

Research questions

- Do existing HTM-based algorithms scale with more cores?
  - If not, how can we achieve scaling on larger HTM systems?
- How do NUMA effects impact transactions?

Our study

- Try to answer the research questions using transactional lock elision (TLE) as an example

Background

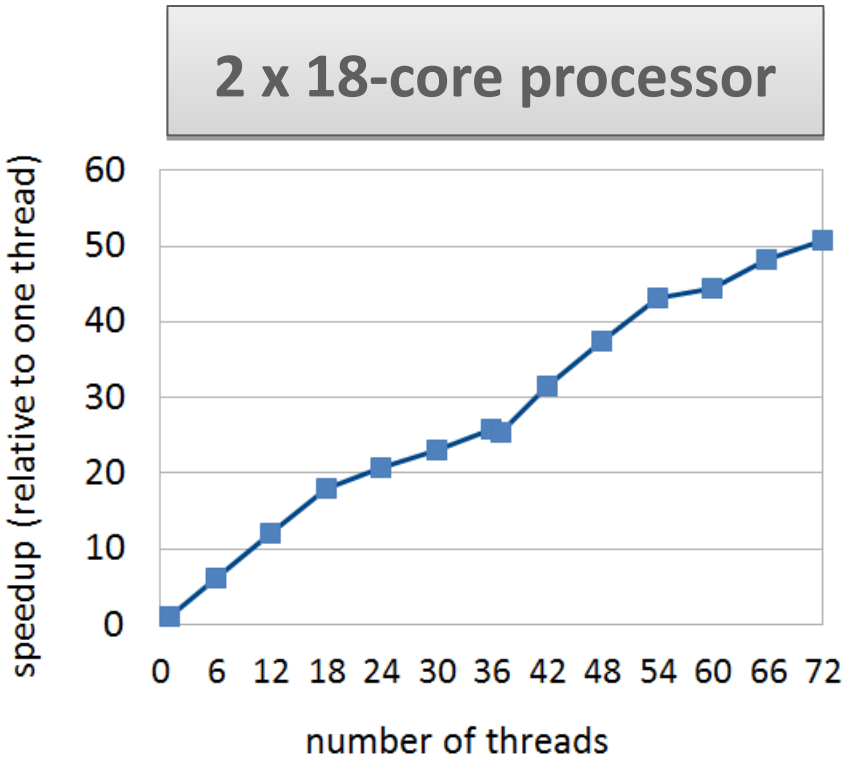
- Non-uniform memory architecture (NUMA)
  - Very different costs for a processor to access its own cache(s), a different processor's cache(s), or main memory
- Hardware transactional memory (HTM)
  - Perform arbitrary code blocks atomically
  - Best effort – can fail for any reason

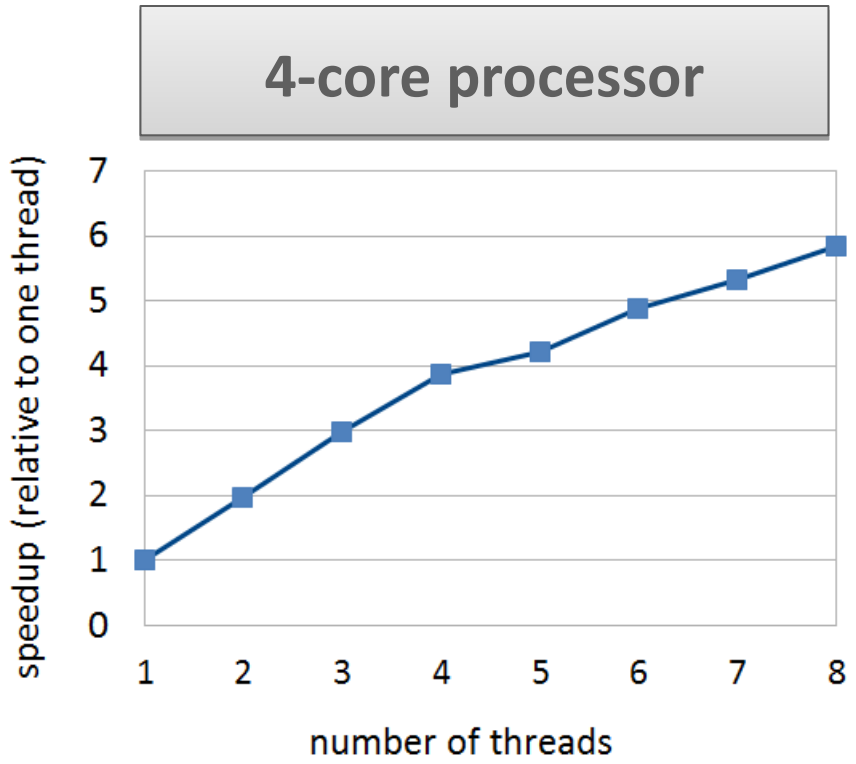
Transactional lock elision (TLE)

- Start with a critical section protected by a single lock
- Replace lock()/unlock() with begin/end transaction
- Invocations of the critical section are executed in parallel, and the HTM system aborts conflicting transactions
- If a process aborts too many times, it falls back to acquiring a lock (instead of starting a transaction)

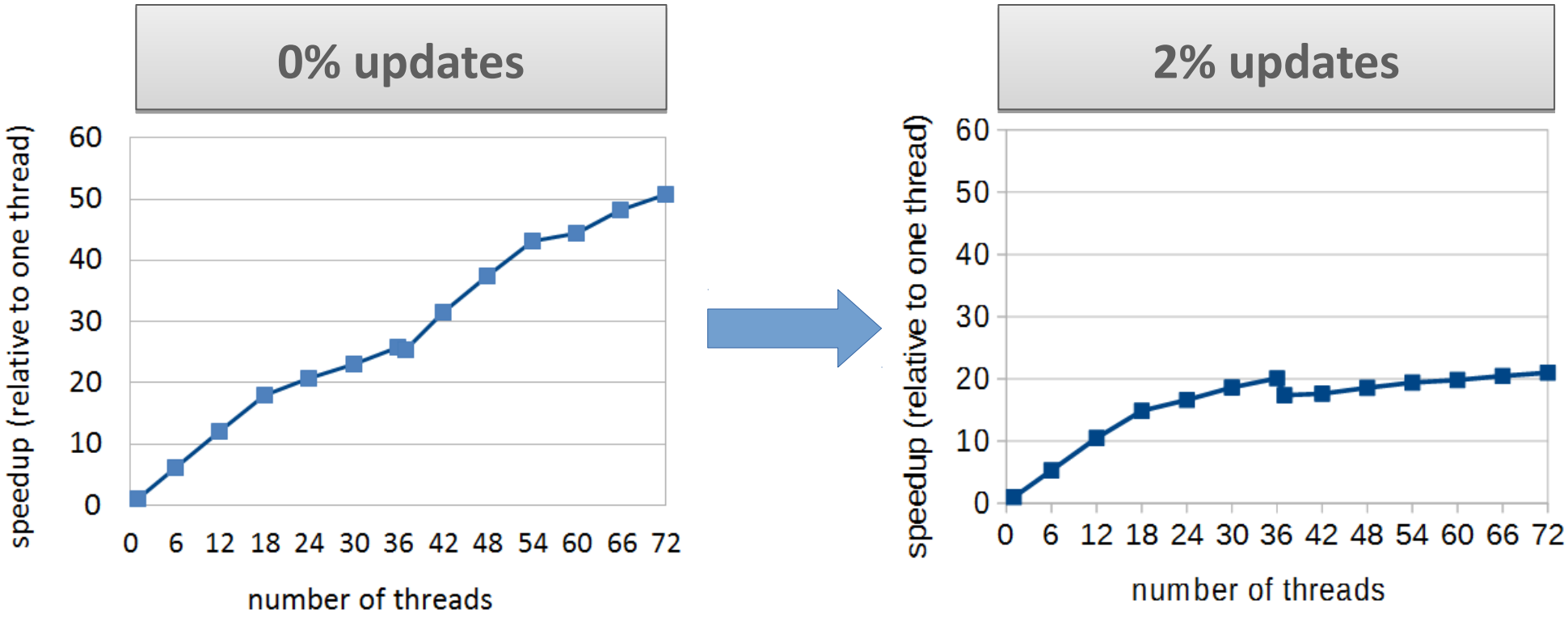
How NUMA affects HTM

TLE AVL tree with read-only workload

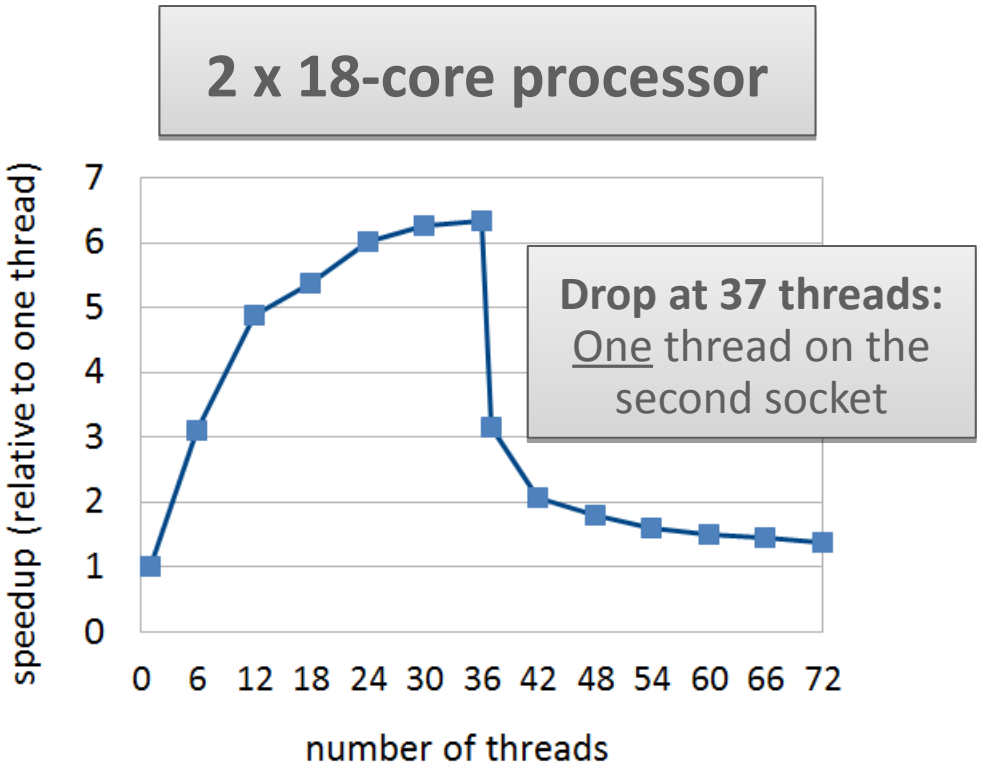


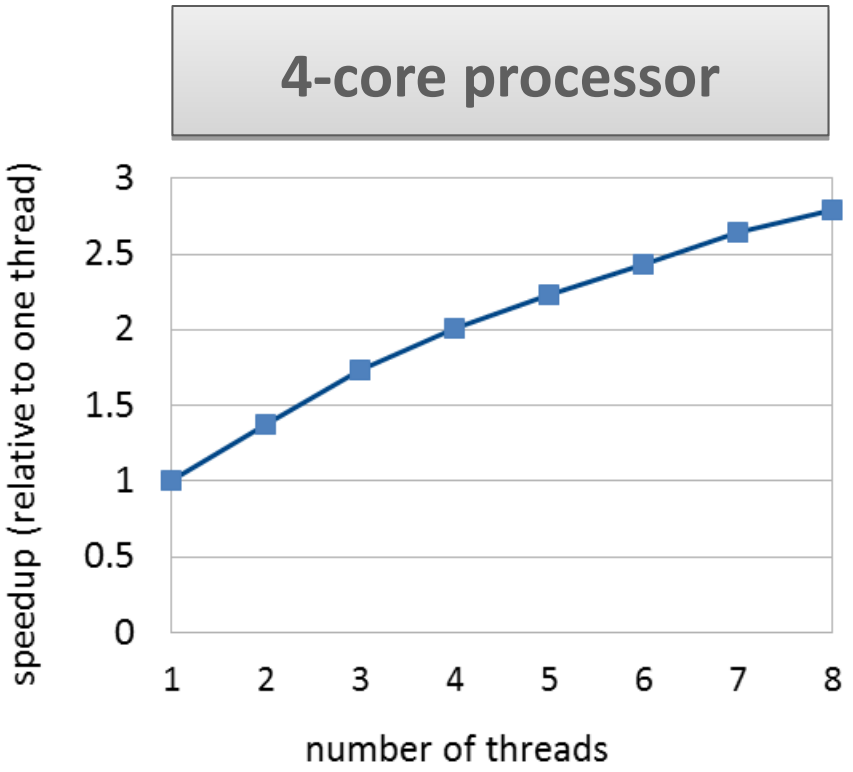


TLE AVL tree with 2% update workload



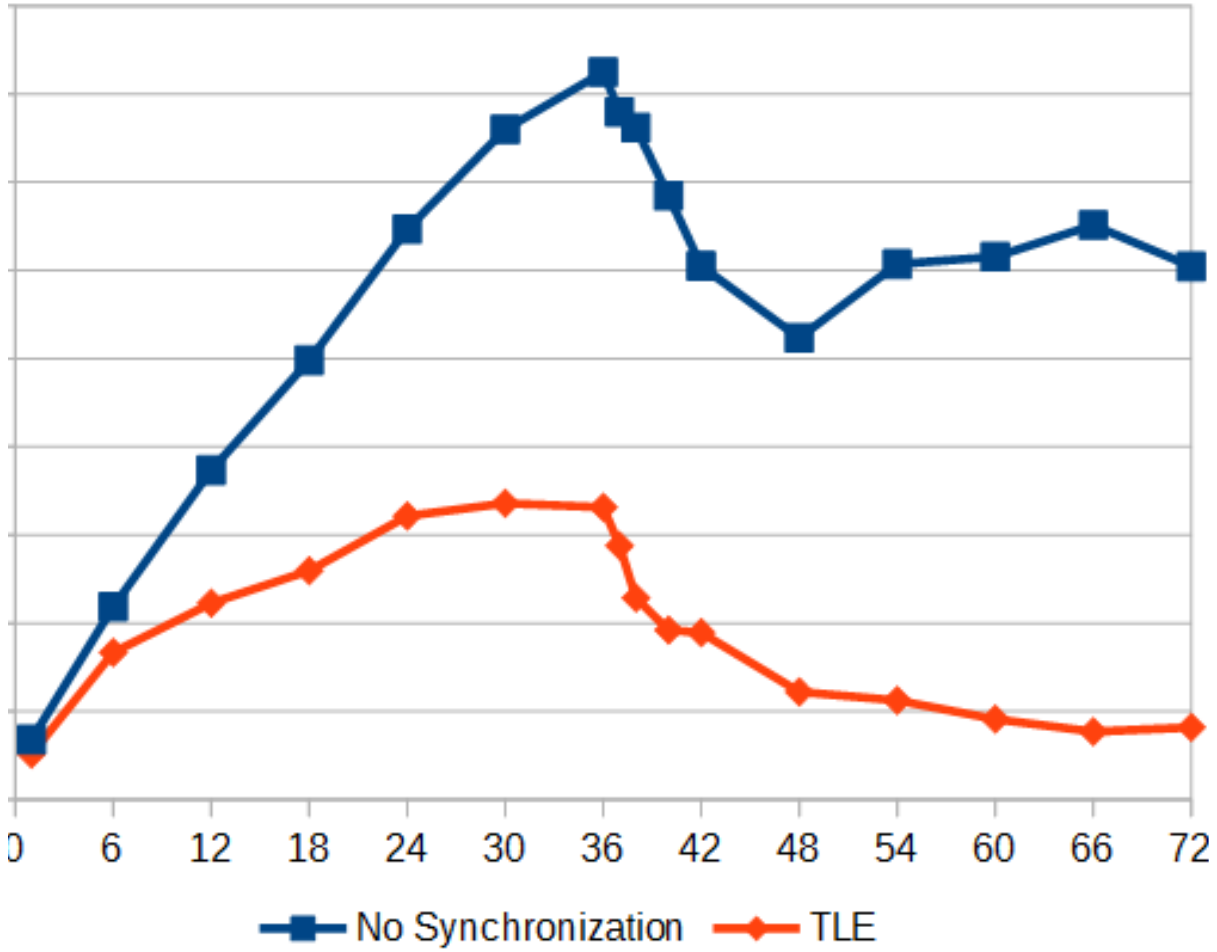
TLE AVL tree with 100% update workload





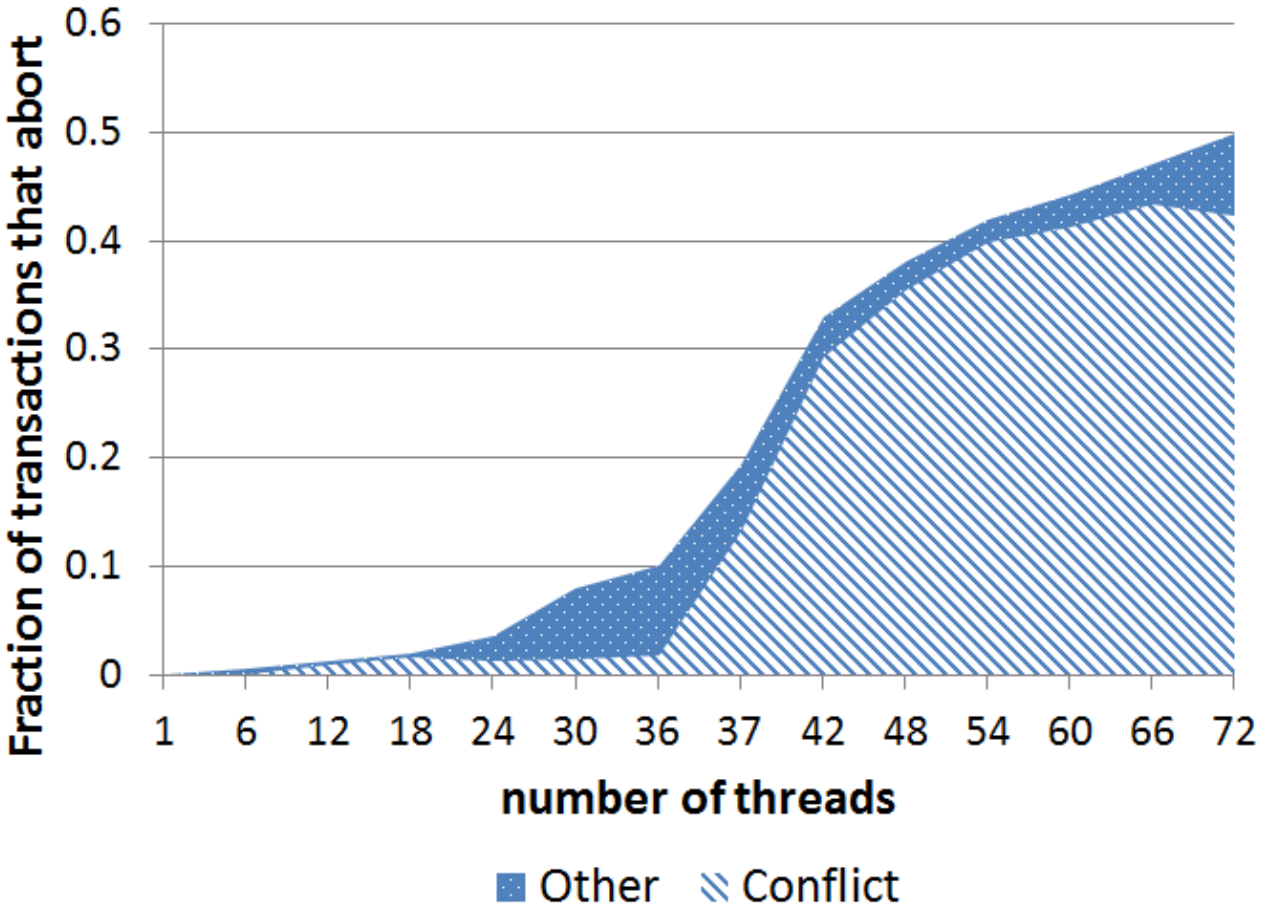
NUMA effects are worse with HTM

- Experiment
  - Each thread repeatedly: performs a search, and then overwrites the key of the last node it reaches
- x-axis: number of threads
- y-axis: operations/second (up is good)



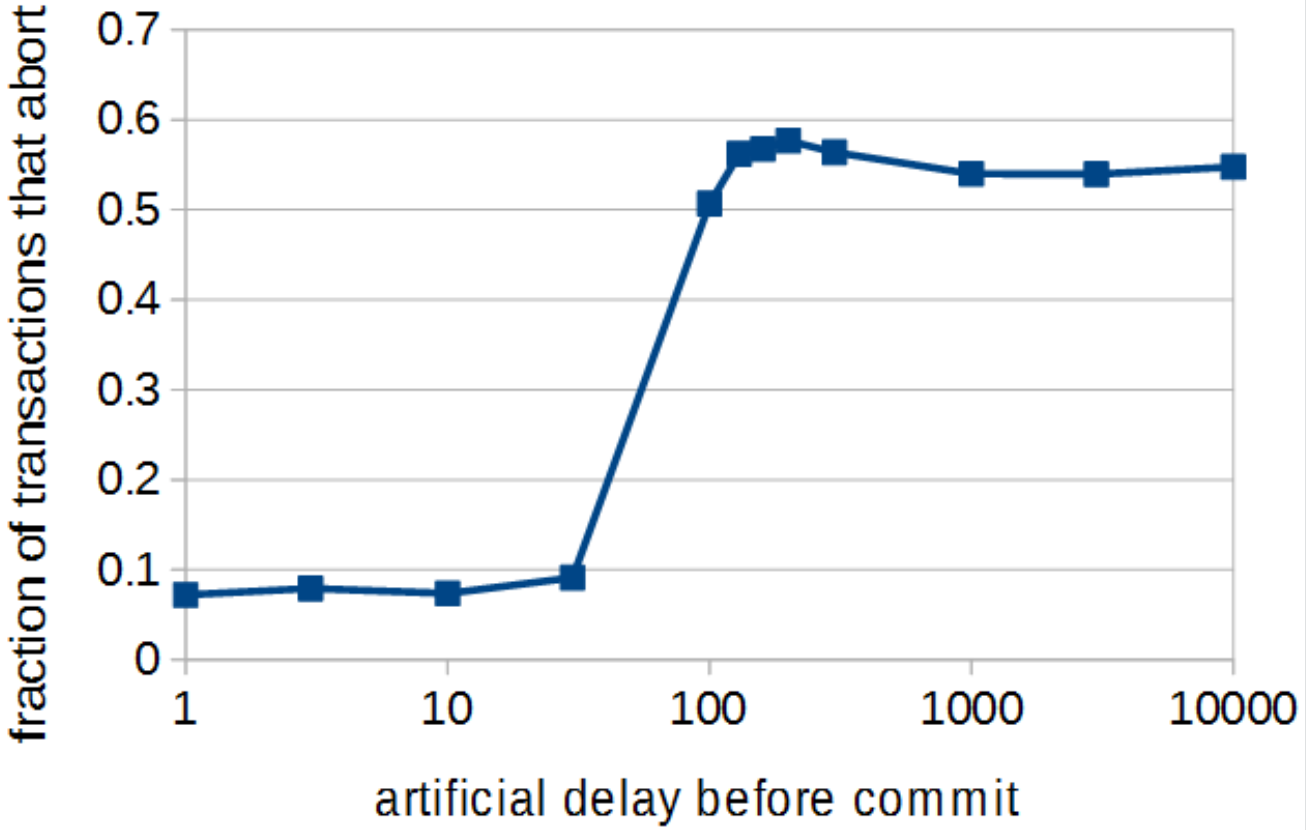
Why NUMA hurts HTM

- Abort rates increase dramatically with threads on a second socket
- Claim: cross-socket cache invalidations lengthen transactions, which increases their windows of contention



Abort rates for longer transactions

- Claim: cross-socket cache invalidations lengthen transactions, which increases their windows of contention
- Experiment: run **single socket**, and add artificial delay just before commit



Dealing with NUMA

Naïve approaches

- Threads back off when their transactions abort
  - Even one thread on the second socket can be too many
- Starve all but one socket
  - Poor performance for workloads that scale on multiple sockets

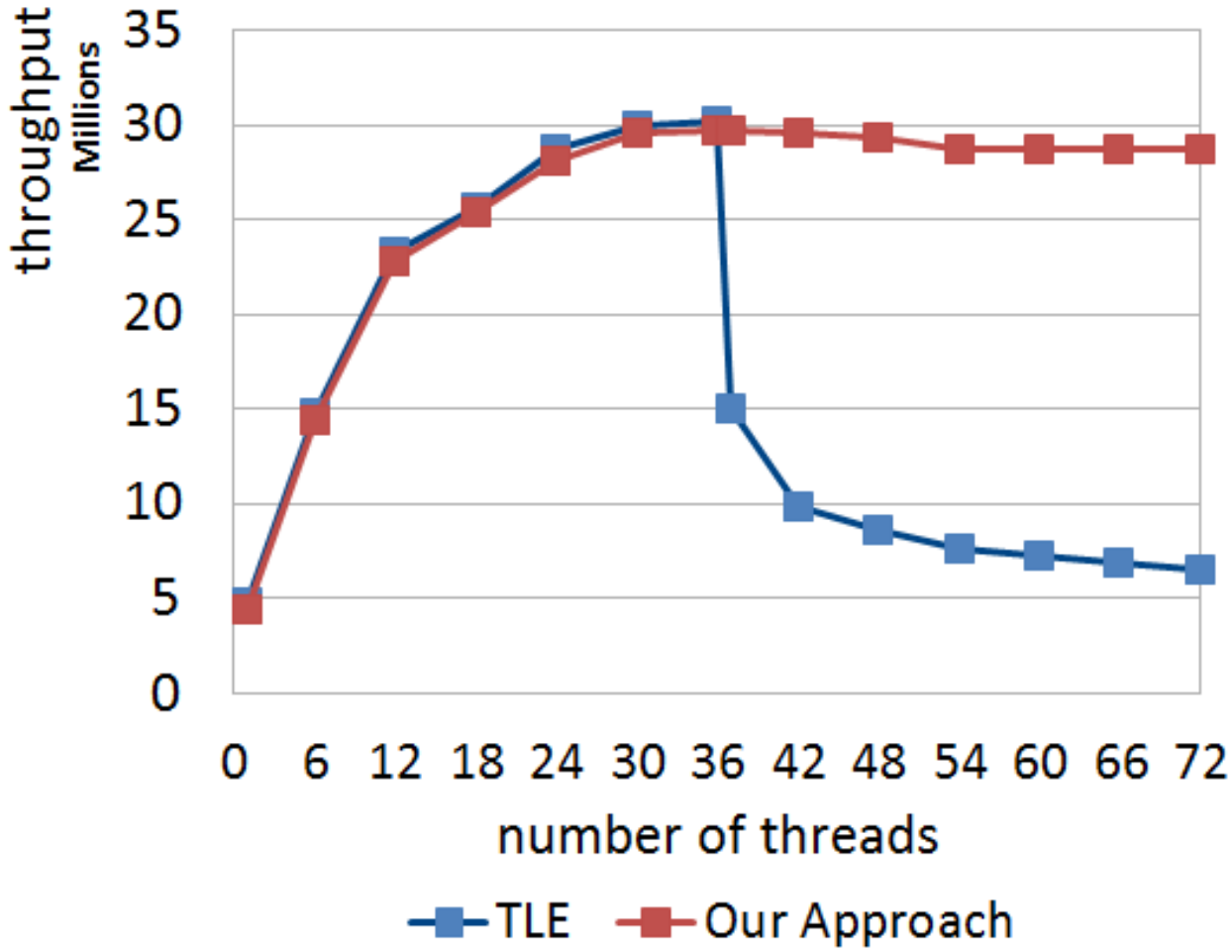
Our approach

- Example: TLE in a system with multiple locks
- Associate a mode with each lock
  - Mode 1: socket 1 only
    - The lock can be acquired only by threads on socket 1 (and threads on other sockets will simply block on lock())
  - Mode 2: socket 2 only
    - The lock can be acquired only by threads on socket 2
  - Mode 3: both sockets
    - The lock can be acquired by threads on either socket

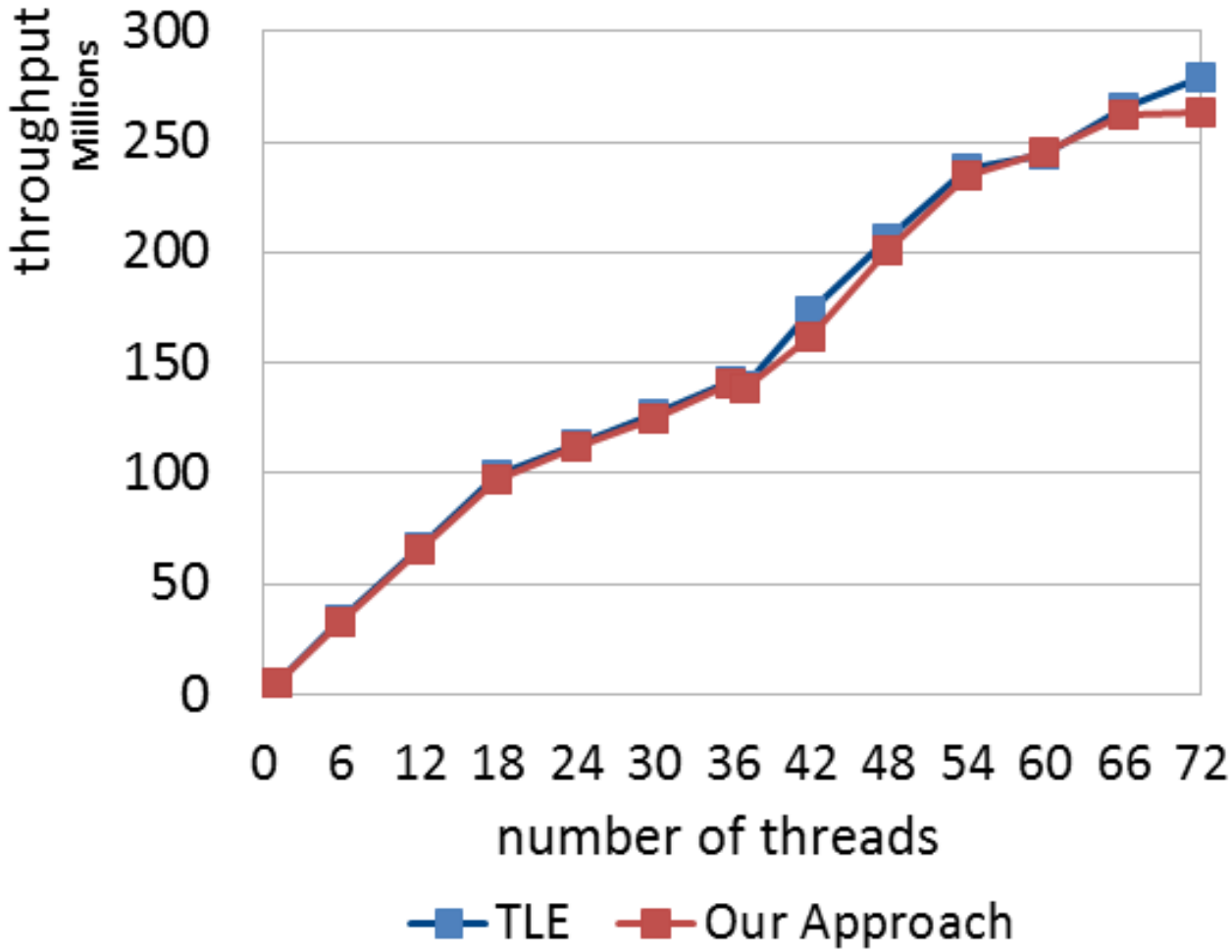
Our approach

- New primitive for programmers – **AdaptToContention**
- AdaptToContention starts a profiling session which will:
  - For each mode x,
    - Set all locks to mode x, and
    - Measure the total number of lock acquisitions for a short, fixed period of time,
  - Then, set each lock to the mode for which it performed best.

TLE AVL tree with 100% update workload

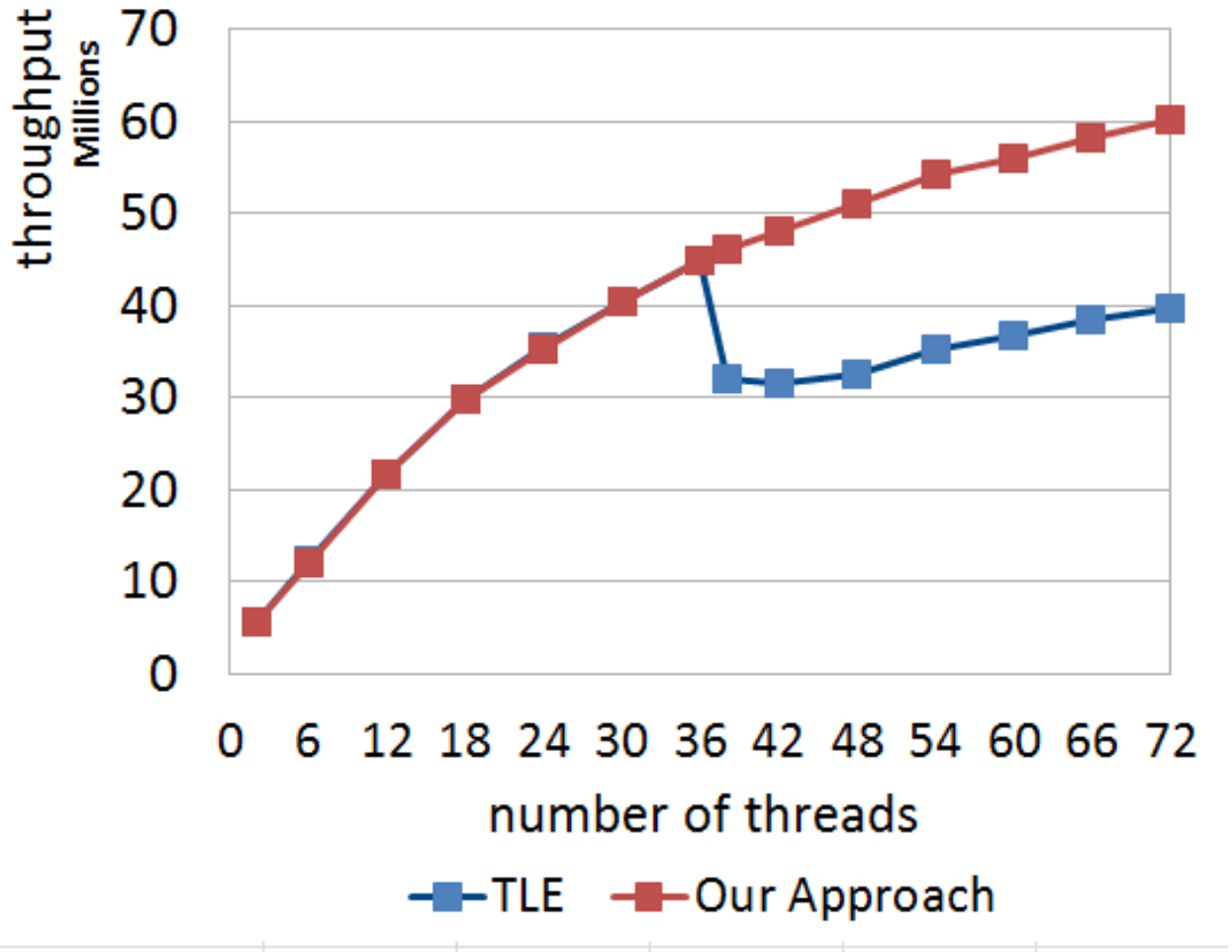


TLE AVL tree with read-only workload



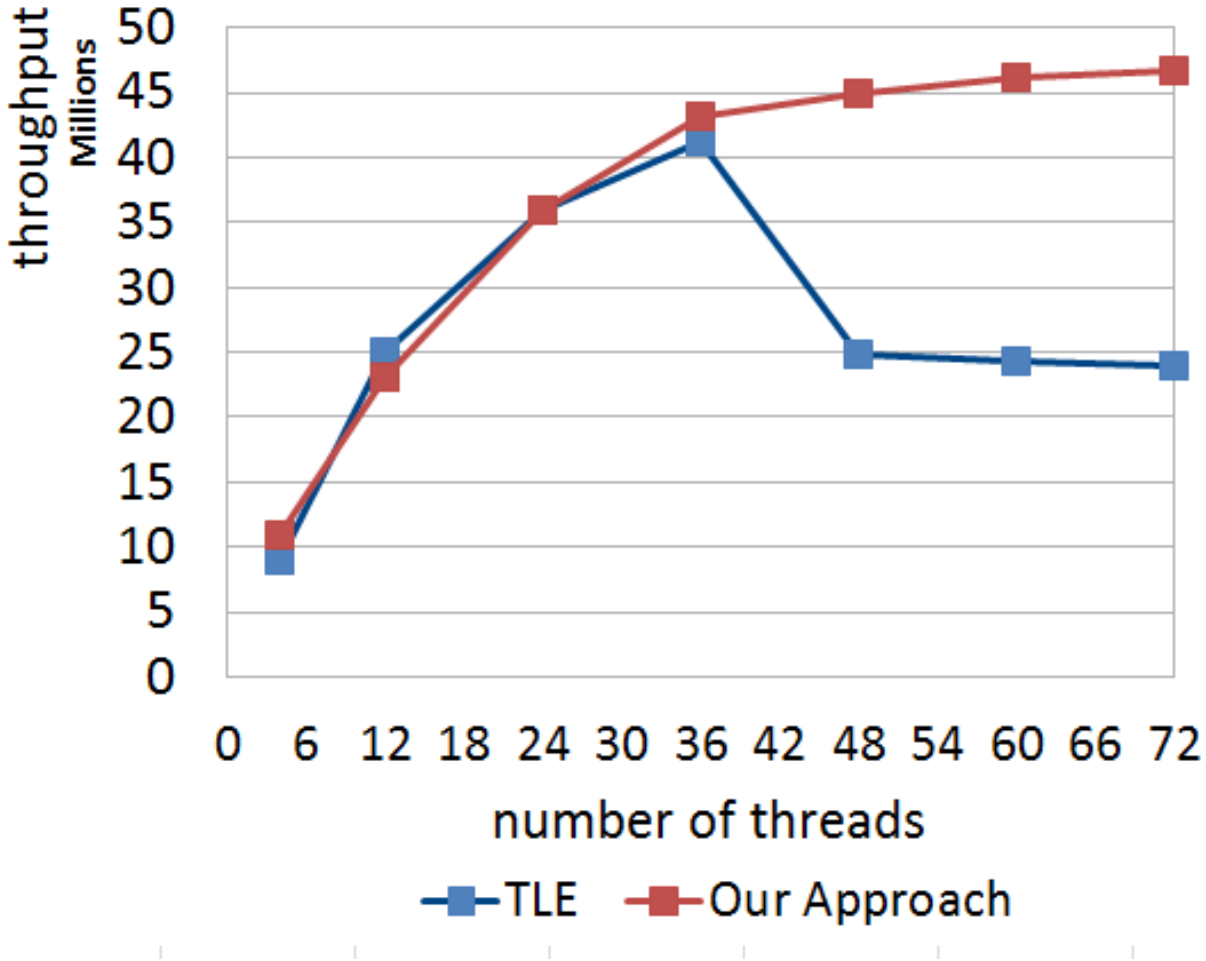
TLE AVL tree with sophisticated workload 1

- Workload:
  - AVL tree #1: small tree, 100% updates
  - AVL tree #2: large tree, 100% searches



TLE AVL tree with sophisticated workload 2

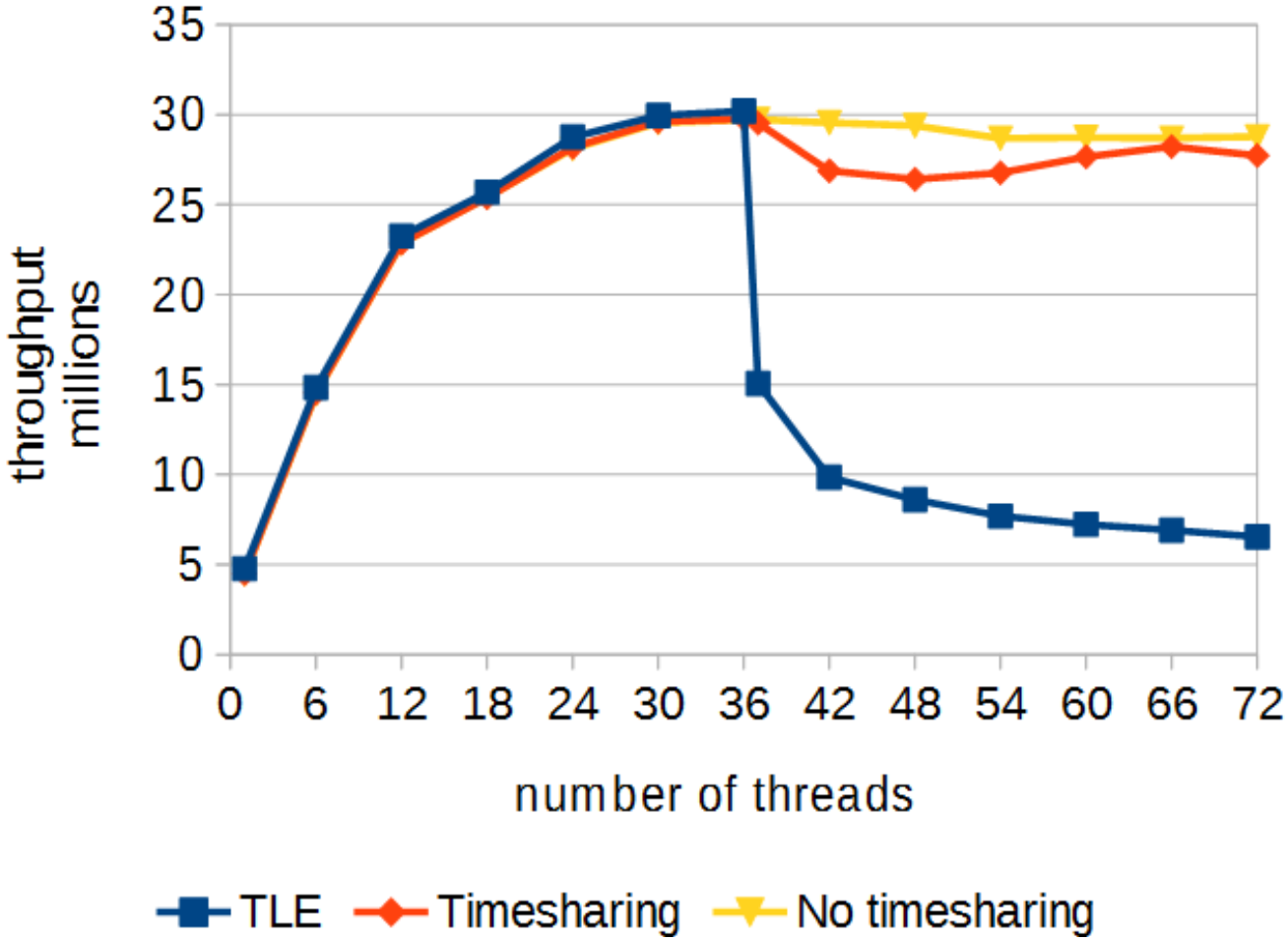
- Workload:
  - AVL tree #1: threads perform operations all over the tree
  - AVL tree #2: threads running on socket 1 operate on the smaller half of the key range (and vice versa for socket 2)



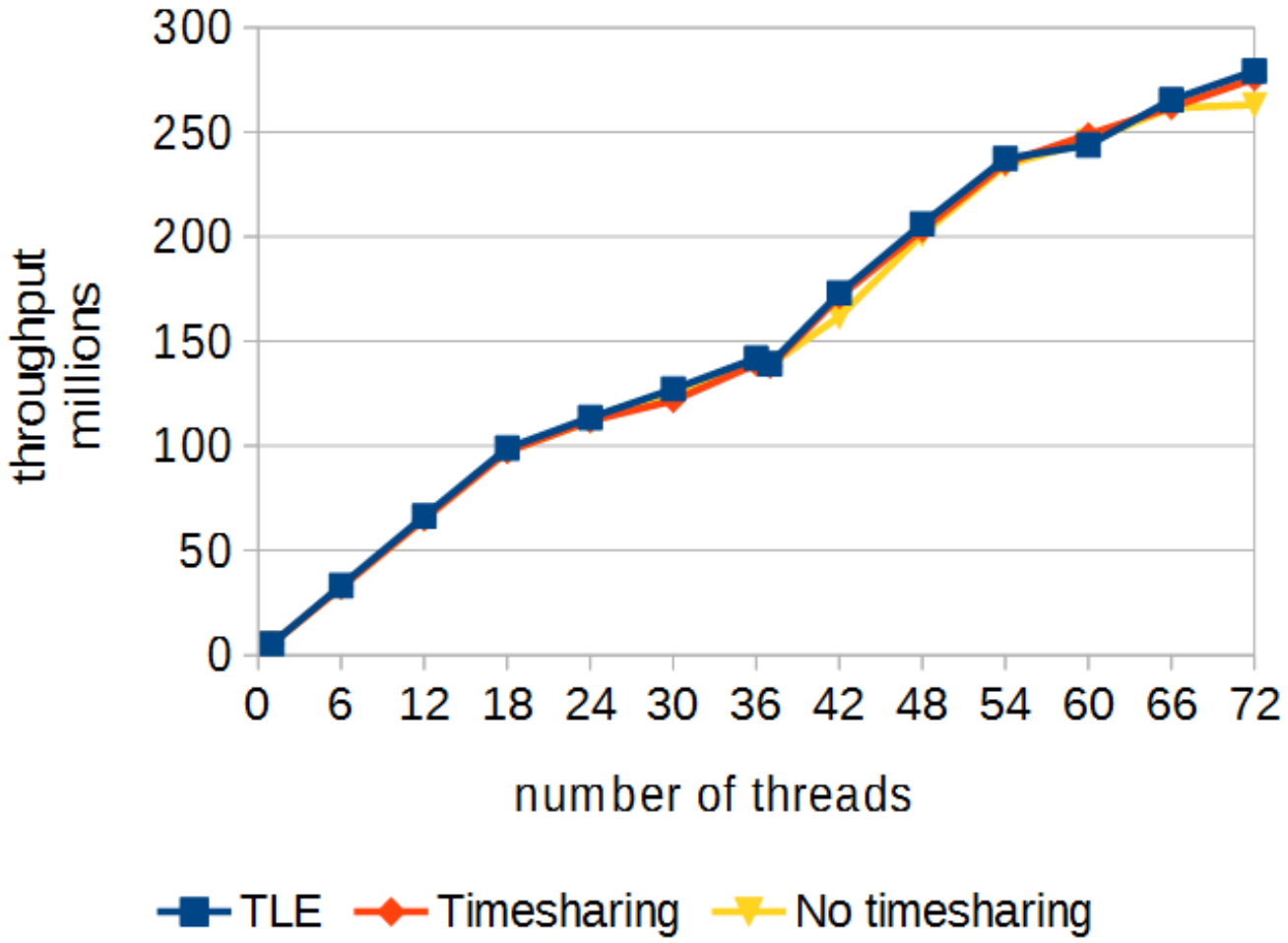
Fairness

- For some applications, it is not a good idea to starve threads
- Solution: time sharing to allow starved threads to run
- Each lock remembers the best mode, and the second best mode, and regularly alternates between these two modes, giving more time to the faster mode
- The amount of time given to a mode is a function of the performance of the lock for each mode

TLE AVL tree with 100% update workload

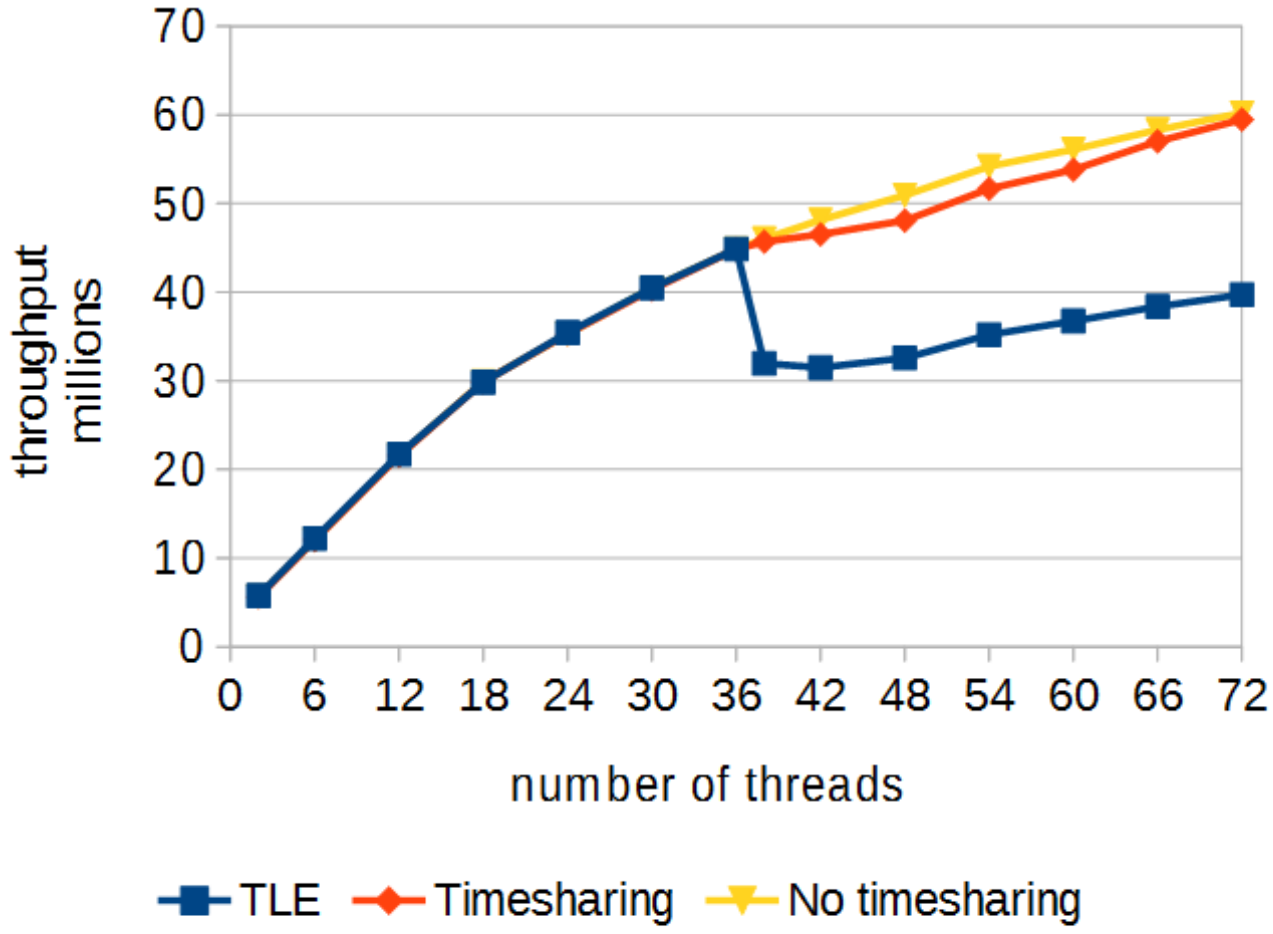


TLE AVL tree with read-only workload



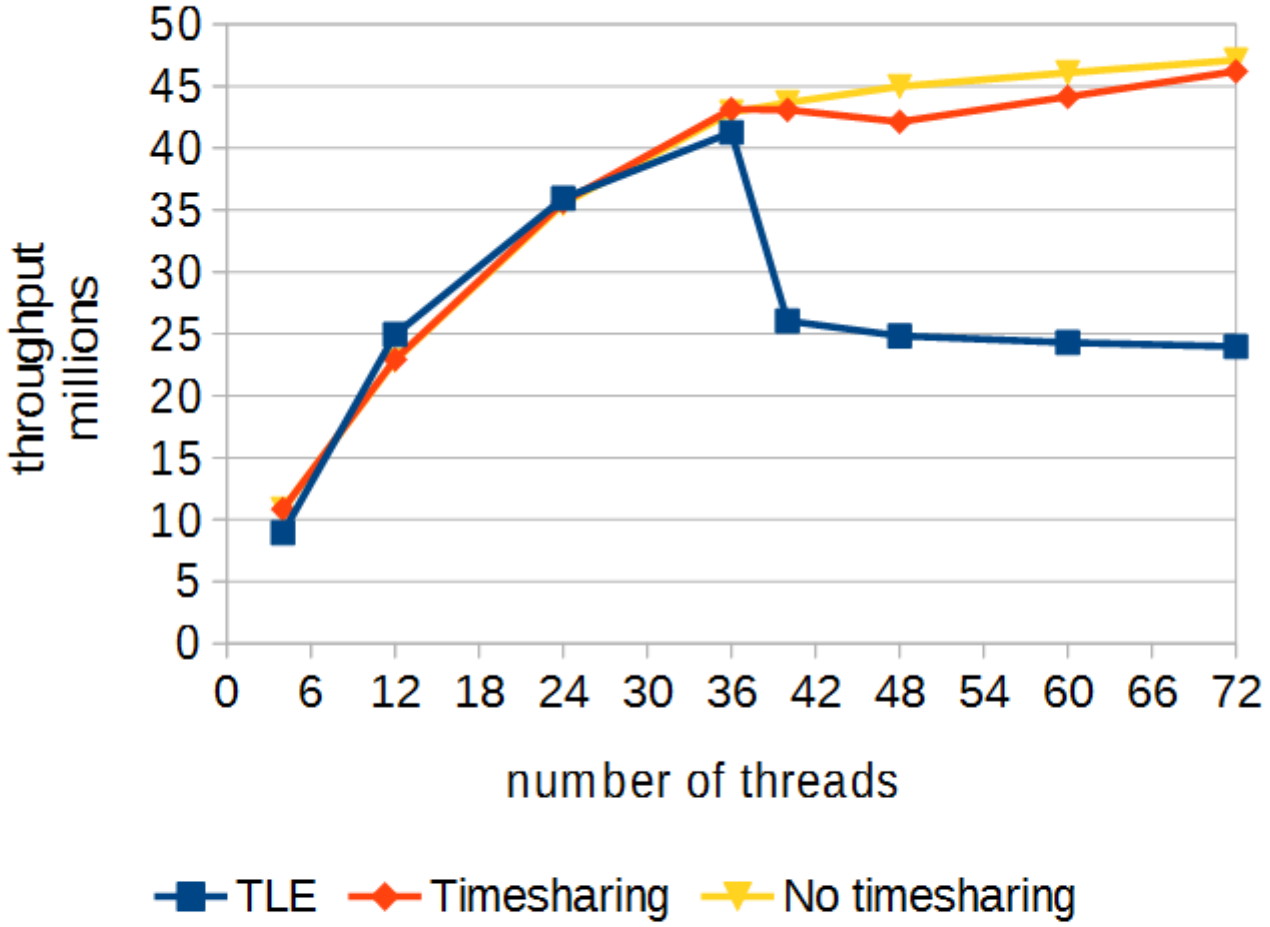
TLE AVL tree with sophisticated workload 1

- Workload:
  - AVL tree #1: small tree, 100% updates
  - AVL tree #2: large tree, 100% searches



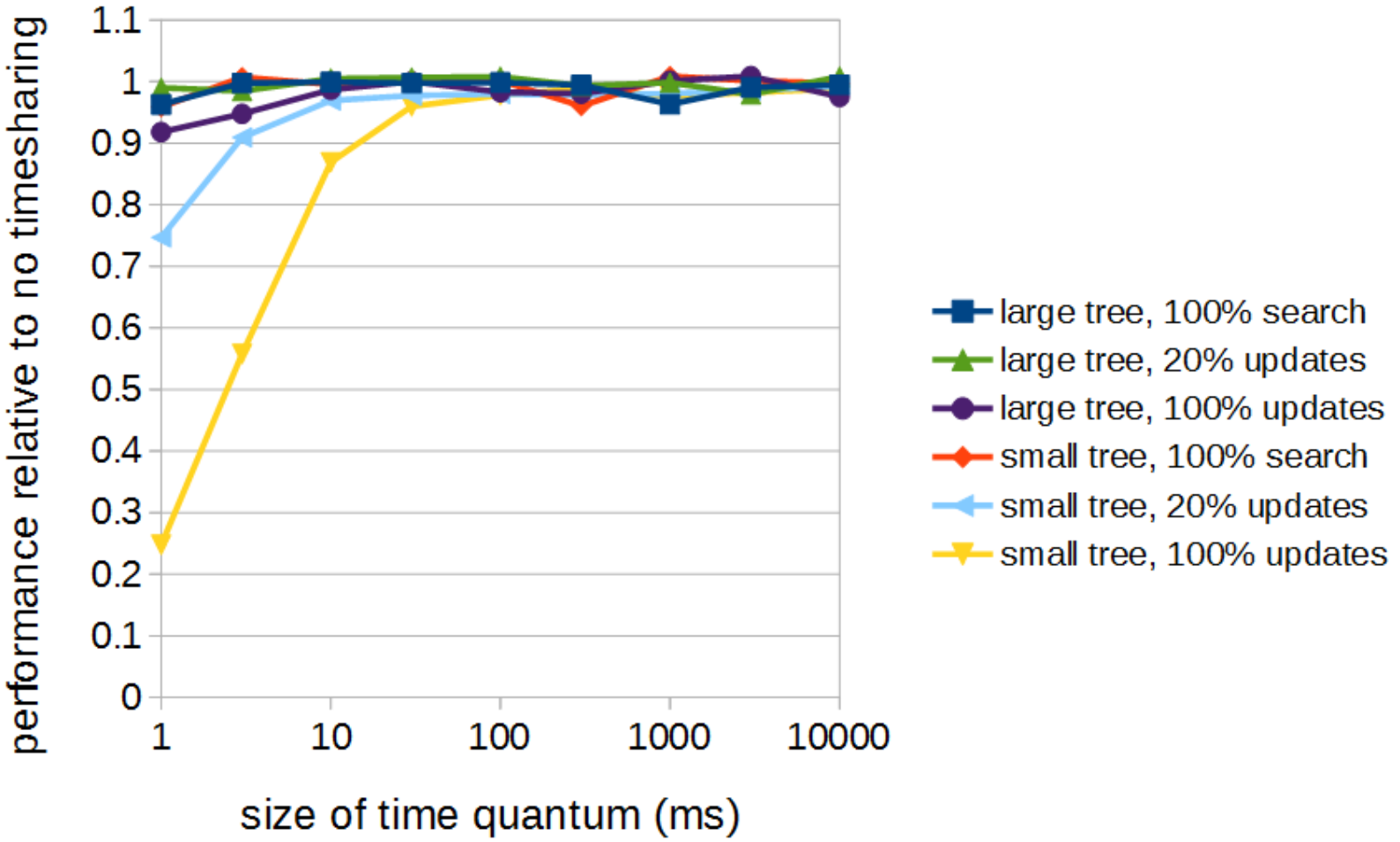
TLE AVL tree with sophisticated workload 2

- Workload:
  - AVL tree #1: threads perform operations all over the tree
  - AVL tree #2: threads running on socket 1 operate on the smaller half of the key range (and vice versa for socket 2)



Overhead of providing fairness

- Divide the execution into quanta and run two lock modes in each quantum
- Very little overhead for 30ms+ quantum
- A smaller quantum is feasible except under very high contention



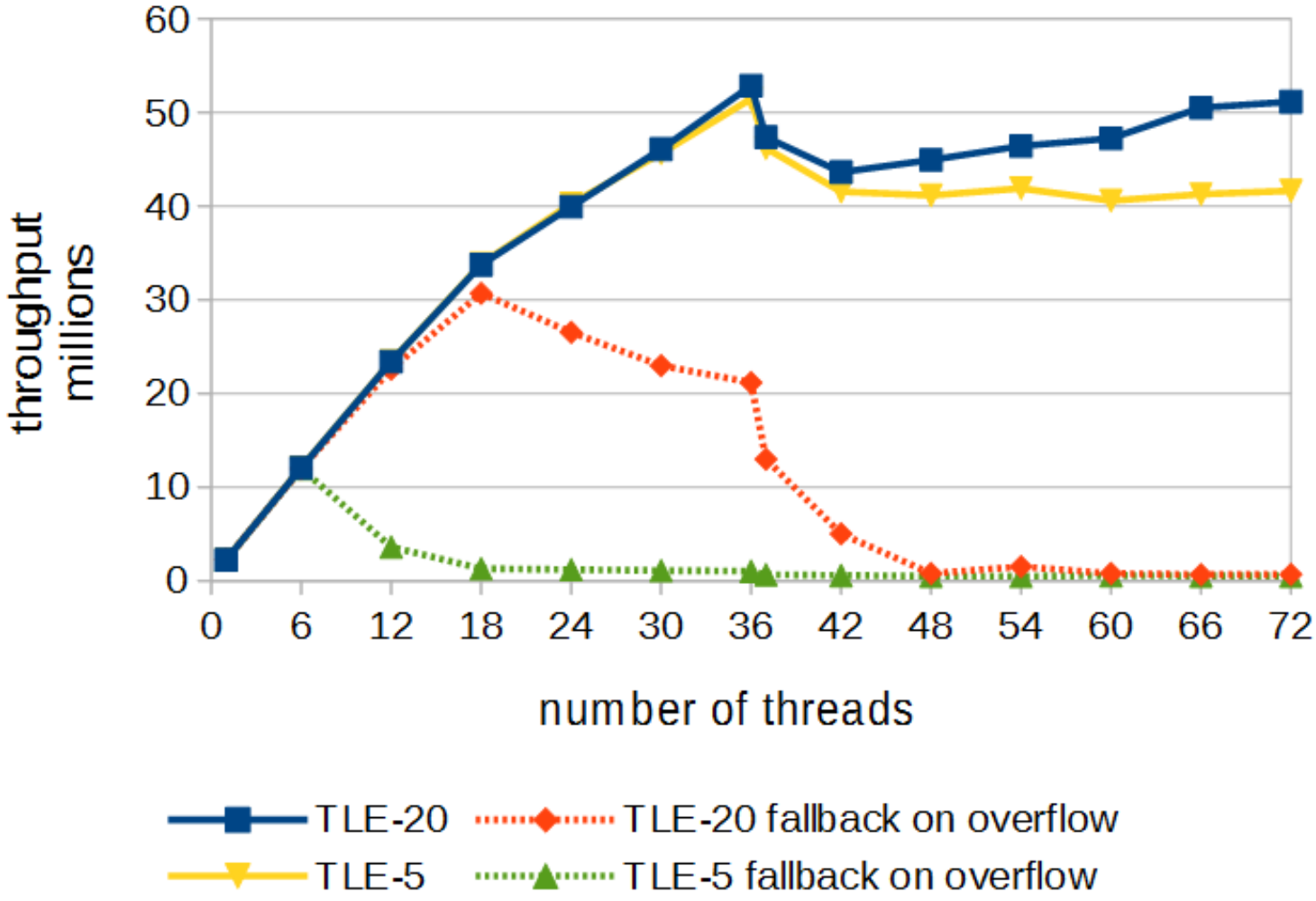
Some details to think about

- Profiling
  - When should profiling be done? (Manually? Periodically?)
  - How long should a profiling session run?
- Fairness
  - When do we switch modes?
  - How much time should be devoted to each mode?
- What if there are more than two sockets?

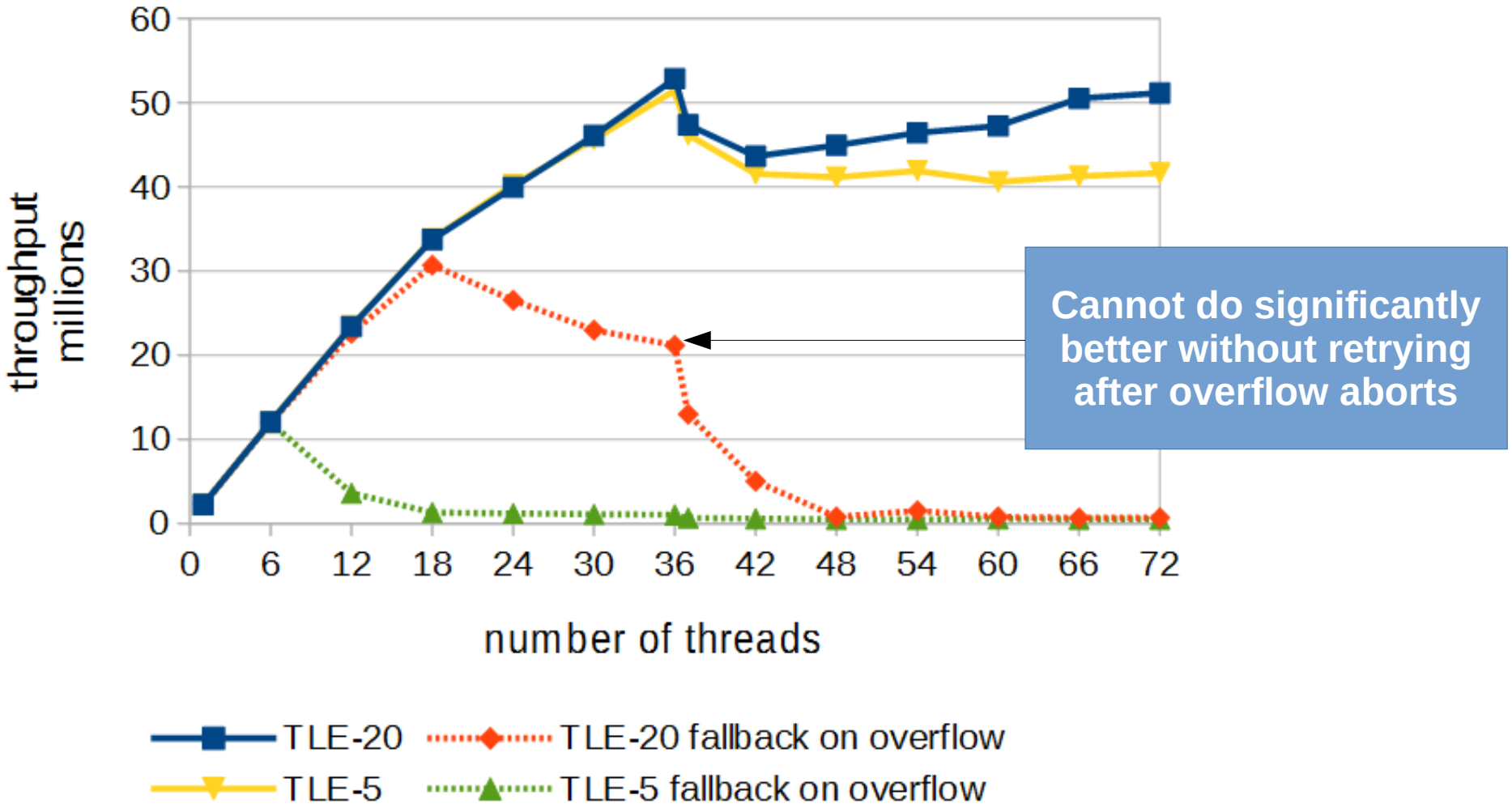
Retrying after aborts

- Many algorithms designed for small HTM systems have poor policies for retrying transactions, e.g.,
  - Retrying transactions at most **5** times before executing the fallback path
  - Going straight to the fallback path when a transaction experiences a capacity/overflow abort
- These policies do not scale on a large HTM system
  - Larger numbers of retries are often necessary, and the impact of taking a global lock on the fallback path is much greater
  - Capacity/overflow aborts are common, and are often resolved by retrying

Retrying after aborts



Retrying after aborts



Another approach: delegation

- Assumption: each operation has a socket on which it should execute. If executed there, it will mostly access data that is local to that socket.
- Idea: send each operation to the socket where it should execute
- Developed interesting algorithms which yield 2x throughput per unit of time spent executing delegated operations, but high overhead
  - Overhead would be smaller for longer operations, but longer operations are more likely to abort…
  - Limited success (~15% improvement) for some workloads by grouping simple operations together in larger transactions and delegating the large transactions

Conclusion

- Found significant differences between NUMA and non-NUMA machines with HTM
  - NUMA has an especially large impact on HTM-based algorithms
  - Cross-socket communication causes high abort rates (even with a single thread on the second socket)
- Presented an extension to TLE for throttling sockets on a per-lock basis
  - Achieves the full performance of two sockets for workloads that scale
  - Avoids performance degradation for workloads that do not scale
- Developed delegation algorithms that reduce L3 cache misses and abort rates. Future work: find workloads that can benefit, despite overhead.

ORACLE


ORACLE®