Optimistic Synchronization

…and the Natural Degree of Parallelism of Concurrent Applications

Prof. P. Felber

Pascal.Felber@unine.ch

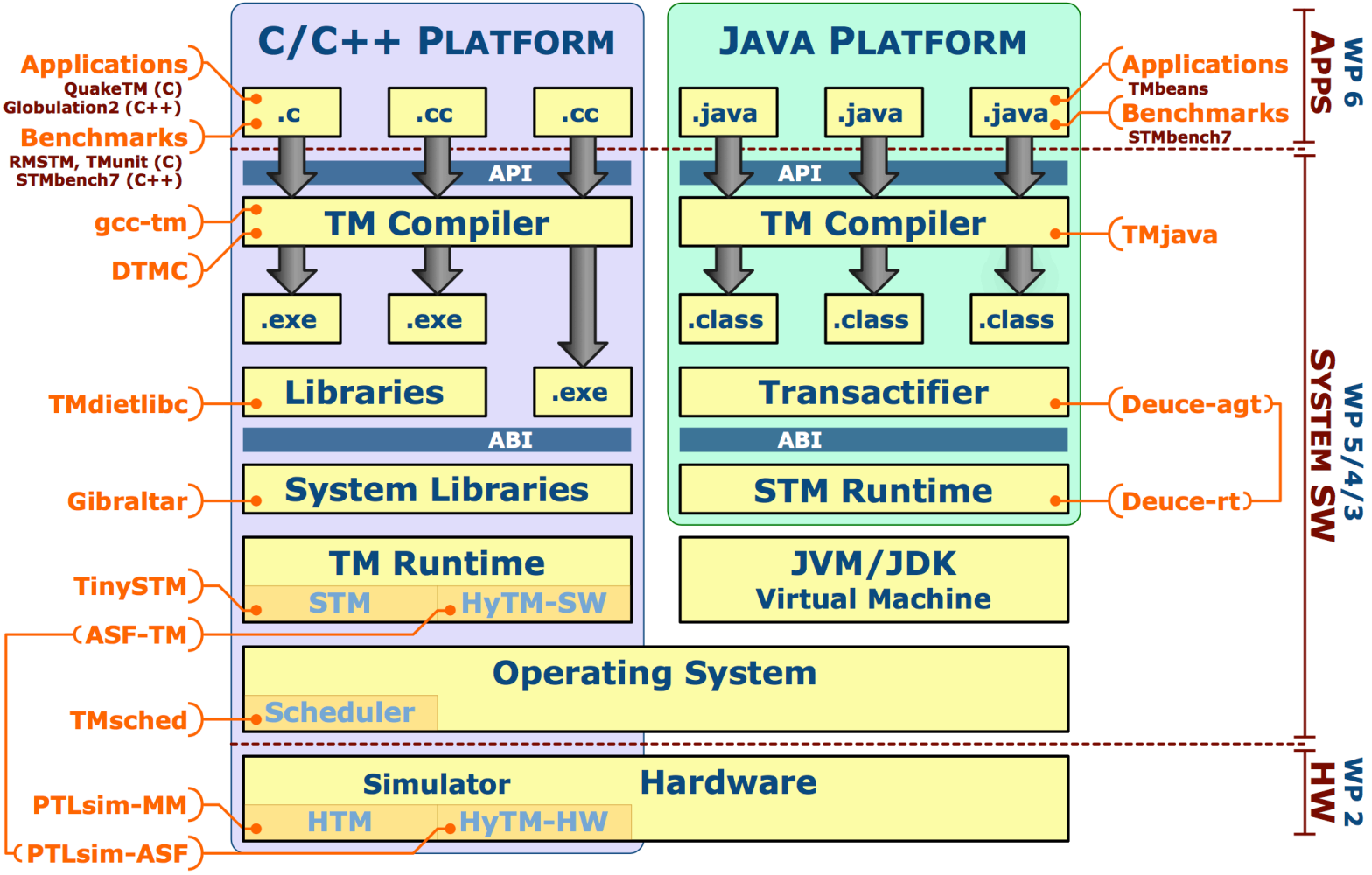
http://iiun.unine.ch/

Institut d'informatique
Université de Neuchâtel

Based on the MSc thesis of J. Schenker

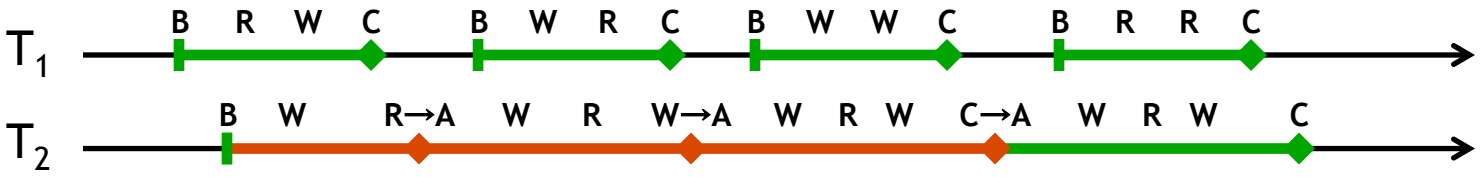
VELOX Stack Overview

Concurrent Programming for Multi-core Architectures



TM in a Nutshell

- Multi-cores are here, the “free ride” is over
  - Concurrent programming necessary, hard to get right
- TM can simplify concurrent programming
  - Sequence of instructions executed atomically
    - **BEGIN** … **READ** / **WRITE** … **COMMIT**
  - Alleviates problems of locks: both safe and scalable
  - Optimistic CC: upon conflict, rollback & restart

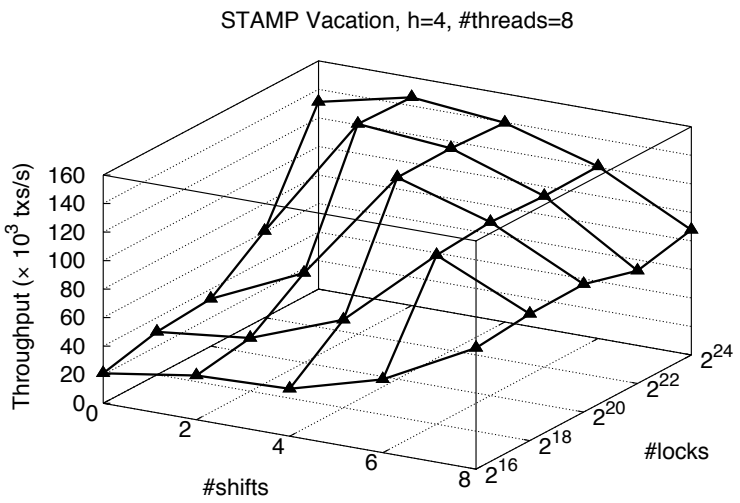


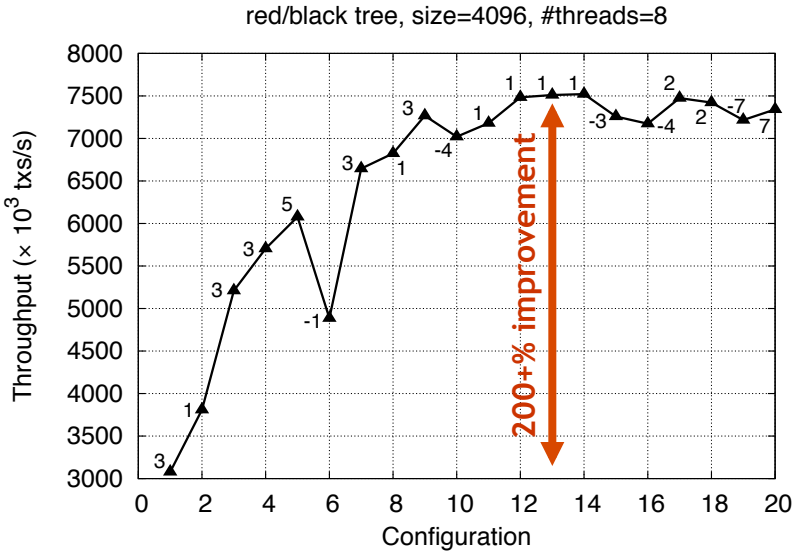
What Affects TM Performance?

- TM is not good for all applications
- There should be **some conflicts**...
  (otherwise no synchronization necessary)
- ...but **not too many**...
  (otherwise pessimistic CC is better)
- ...with **not-too-long** transactions...
  (to keep the cost of aborts reasonable)
- ...involving data **not known statically**
  (otherwise no simpler than locks)

TM Designs and Adaptivity

- Performance of TM depends on workload
  - Obstruction-free vs. lock-based, object- vs. word-based, visible vs. invisible reads, encounter- vs. commit-time locking, write-through vs. write-back, implementation parameters…





Another Dimension

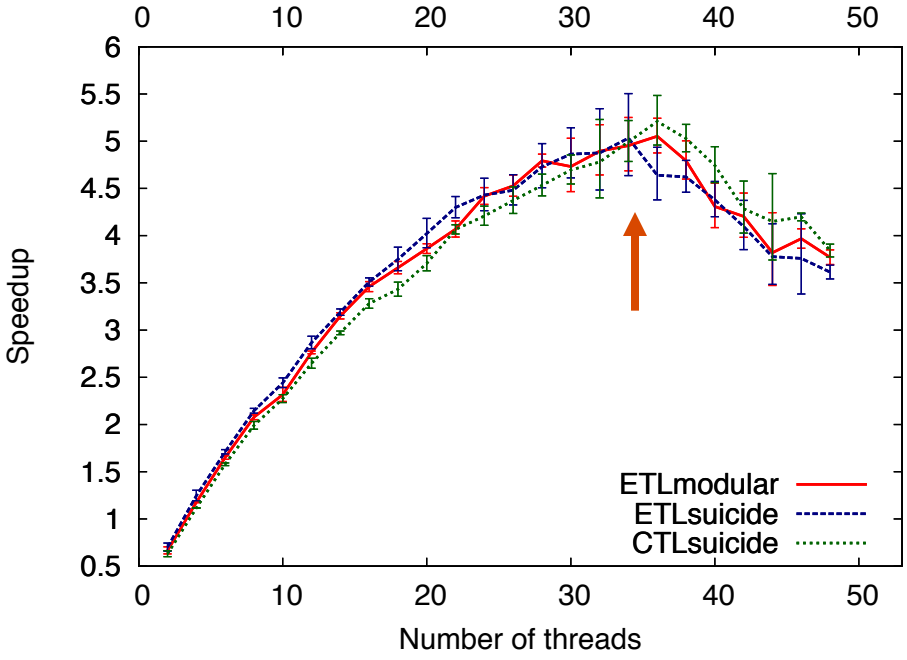
- Most existing TM benchmarks have a variable number of threads
  - Should we have as many threads as cores?
  - More cores = more processing power…
    = more concurrency = more potential conflicts…
    ¿=? less performance
  - Is there an “optimal” number of threads for a given workload (and a given TM algorithm/ configuration)?

⇒ “Natural degree of parallelism” of the workload

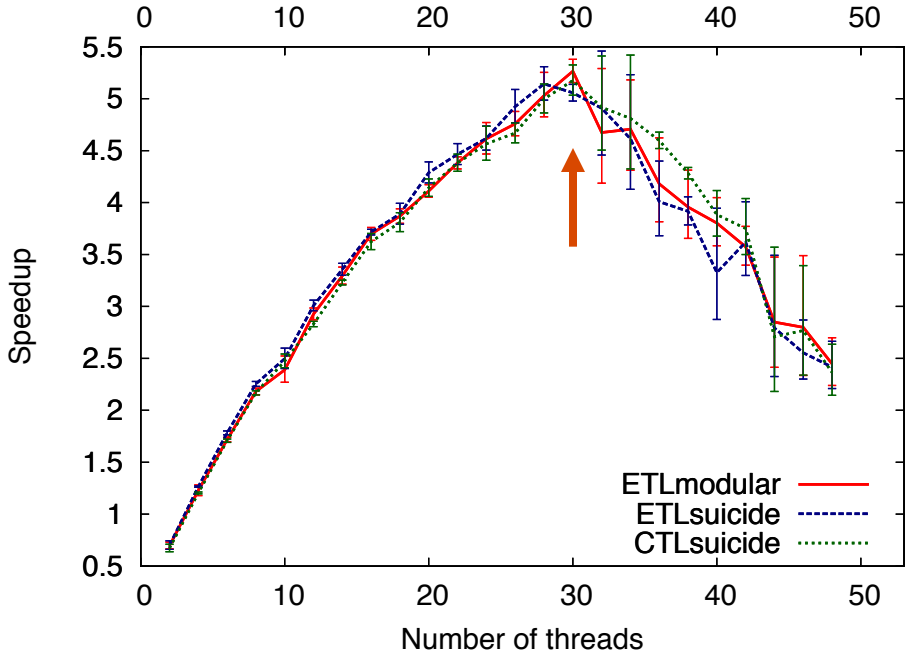
Natural Degree of Parallelism

STAMP

vacation-high



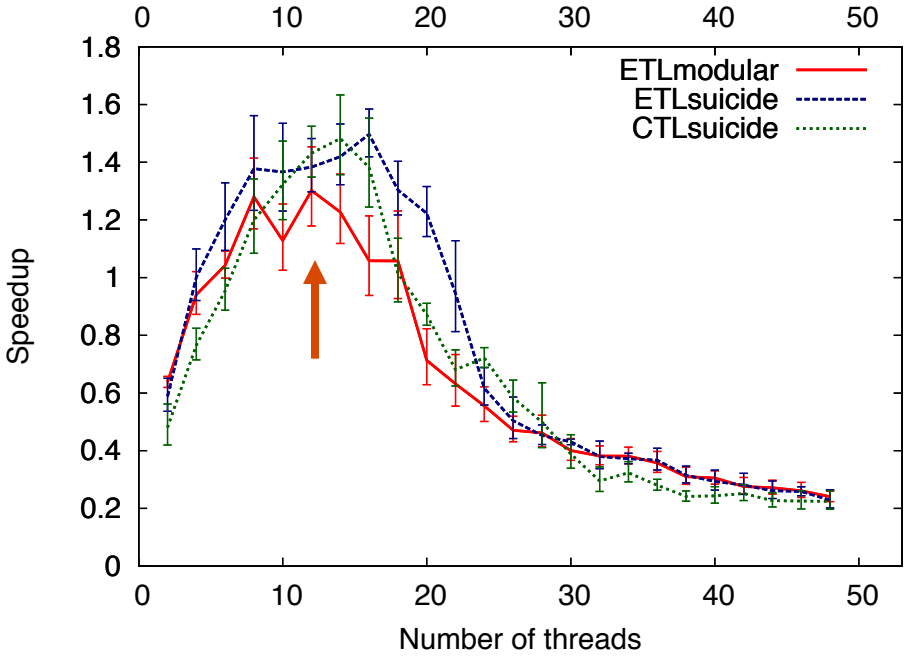
vacation-low



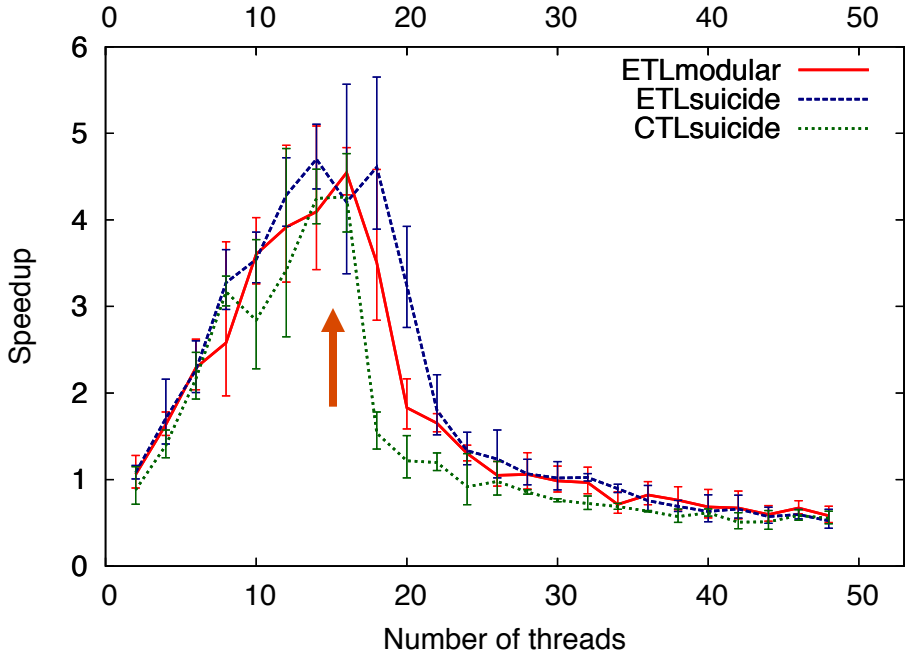
Natural Degree of Parallelism

STAMP

kmeans-high



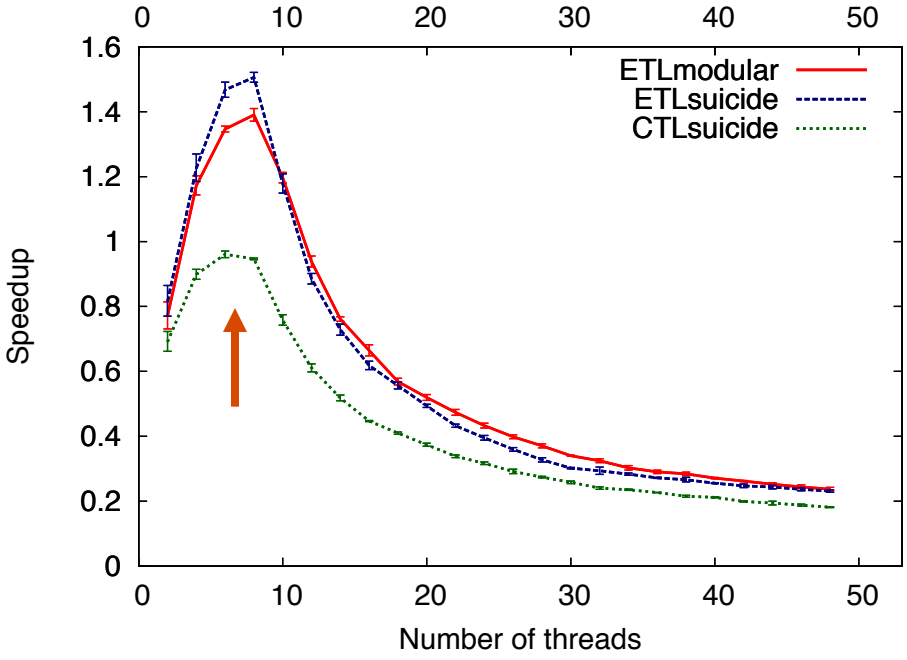
kmeans-low



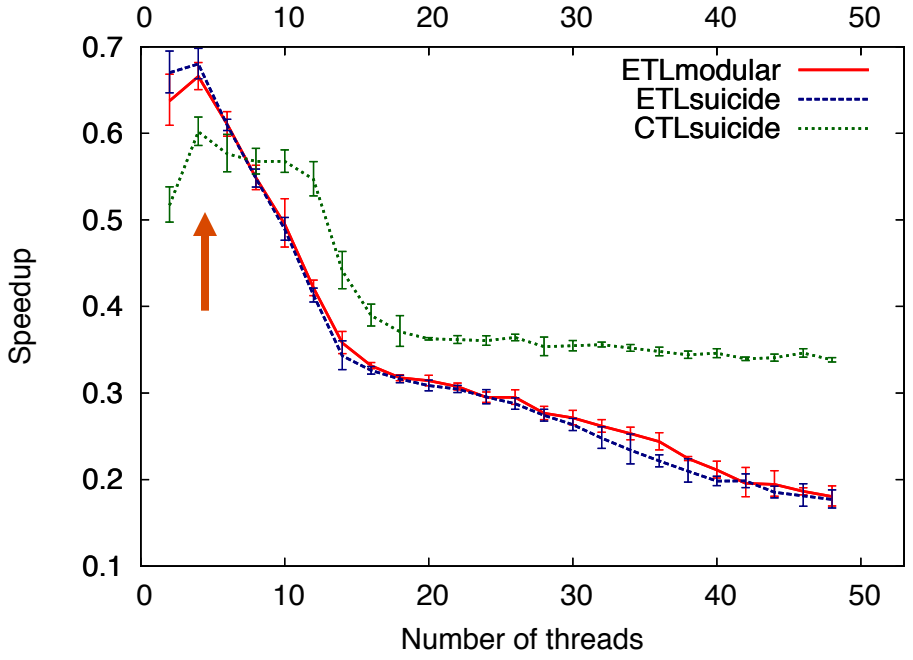
Natural Degree of Parallelism

STAMP

intruder



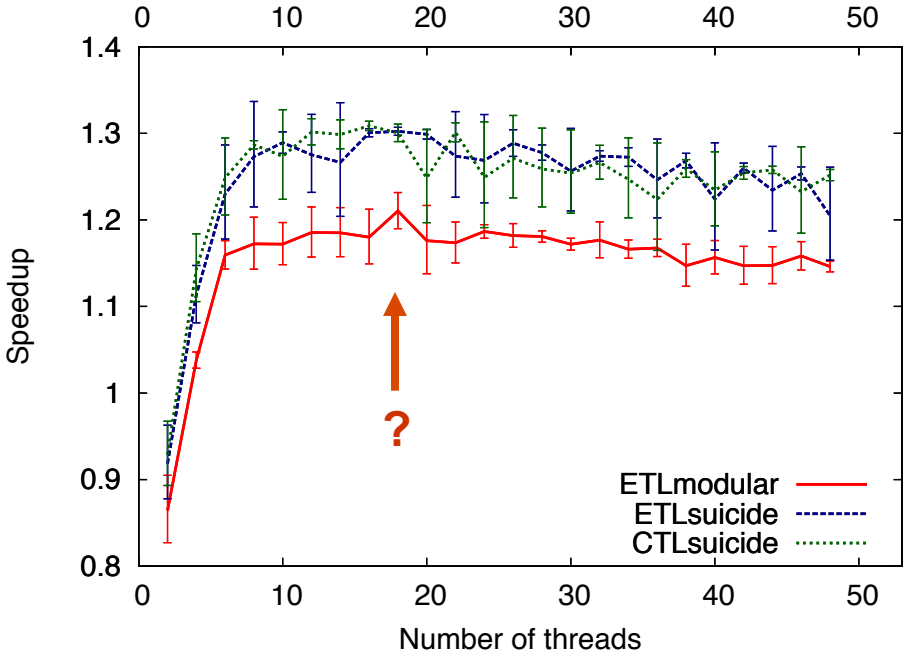
yada



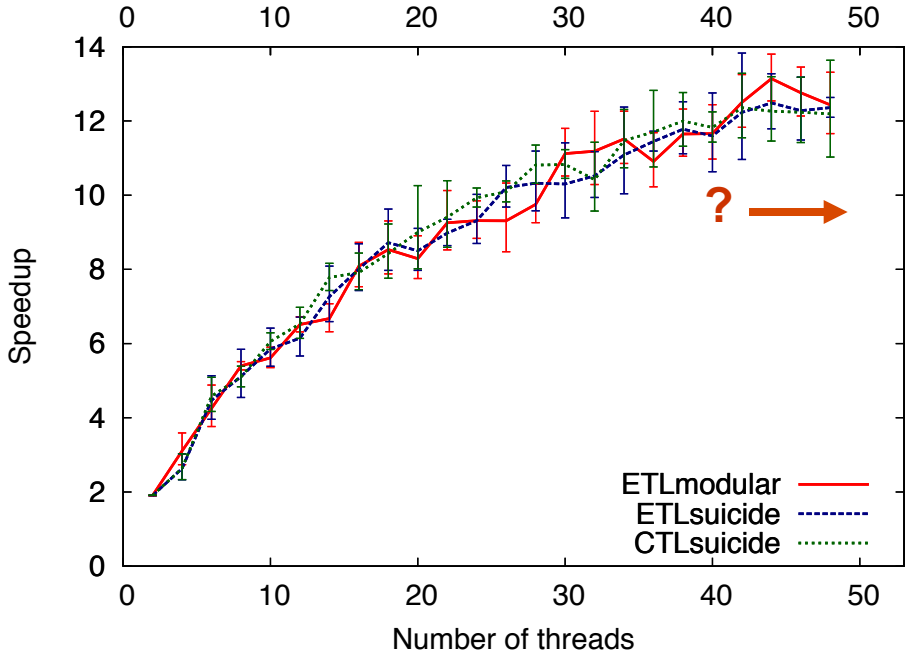
Natural Degree of Parallelism

STAMP

ssca2

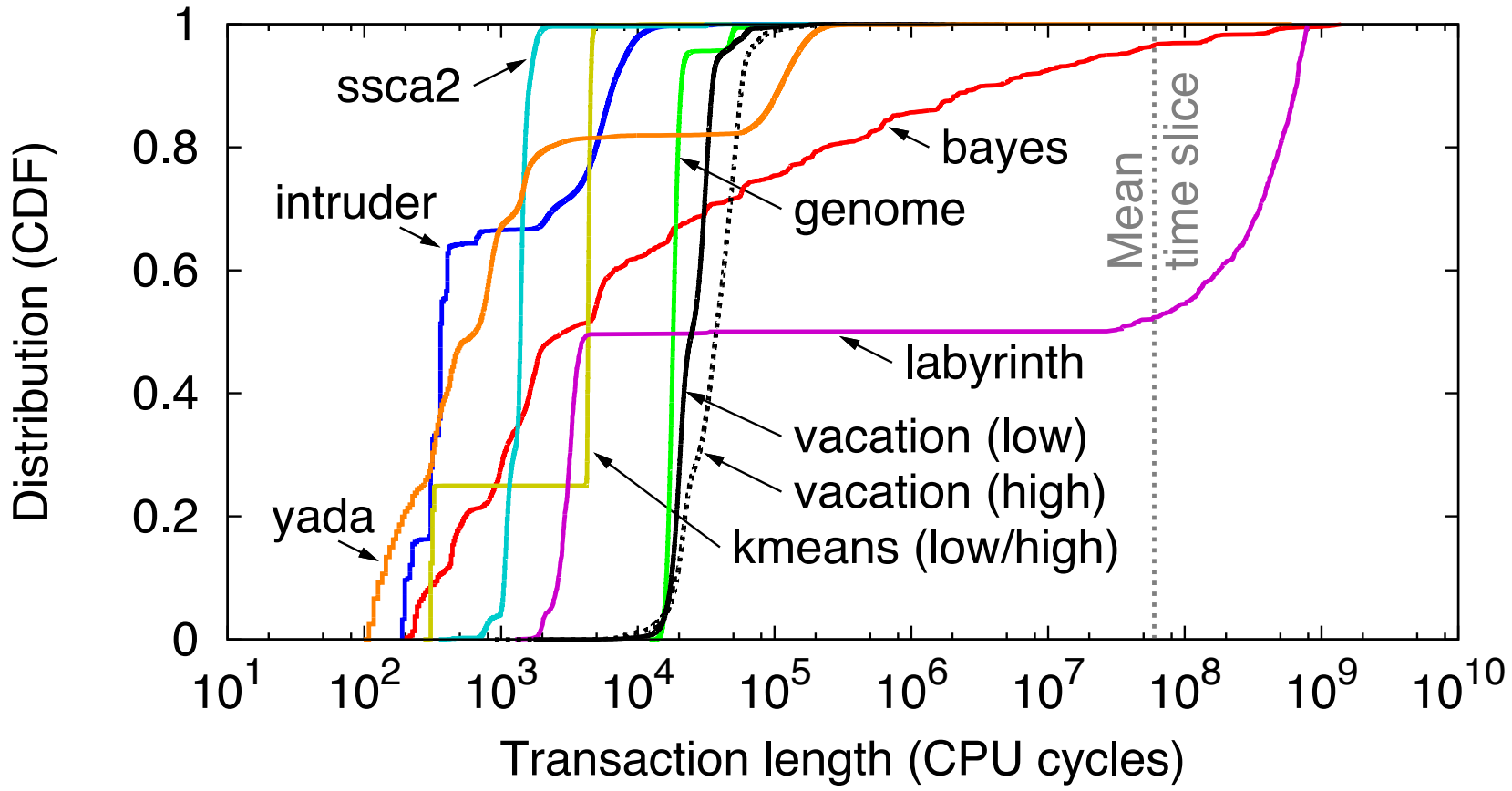


labyrinth



STAMP Workloads

A wide variety of transactions!



Note on Contention Management

- Proper contention management can avoid degradation of throughout for high thread counts
  - E.g., serialize conflicting transactions

… but it does not improve performance…

  - Typically remains flat

… and it wastes resources

  - More threads, same performance!

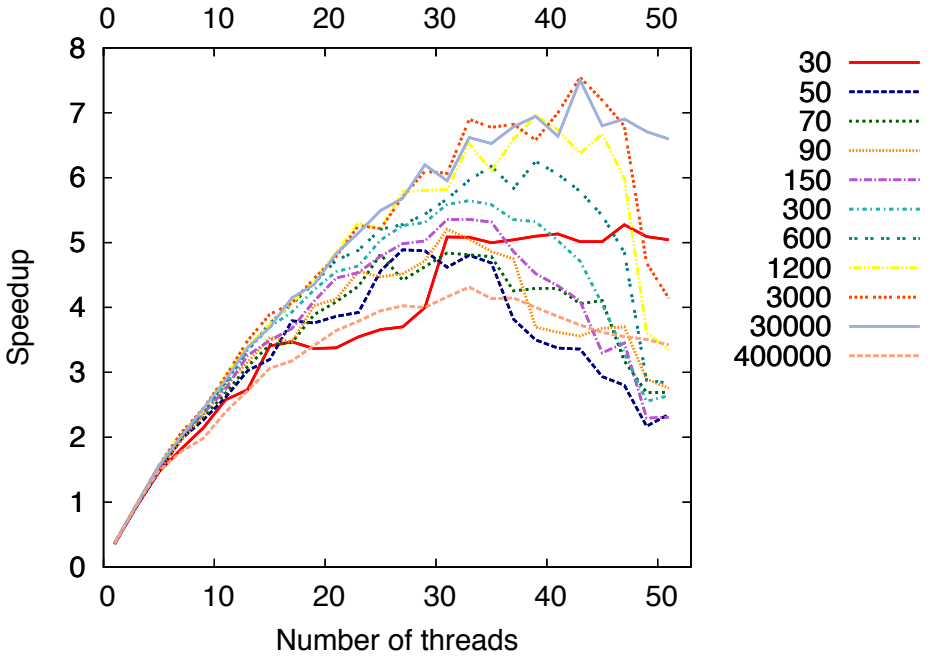
Adapting Concurrency

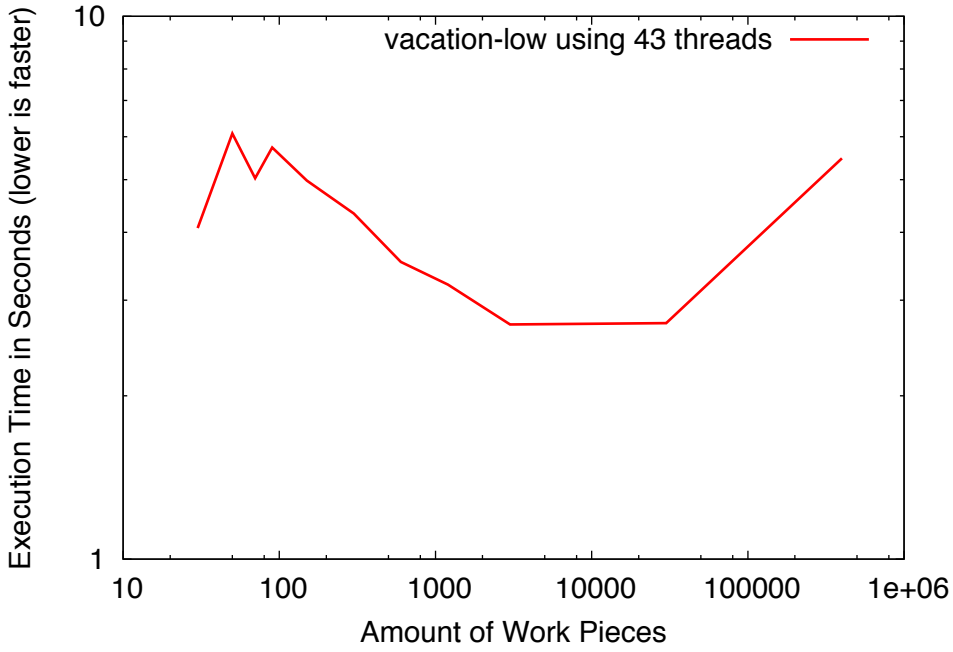
- Goal: dynamic adaptation of the degree of parallelism in TM applications
- Requires support for variable number of threads
- Refactoring of application/benchmark code
  - Split work into small pieces
  - Assign pieces to tasks
  - Use thread pool to execute tasks
- Introduction of a new “main” loop
  - 3 phases: measurement, decision making, adaptation
  - Threads run tasks that process pieces

Importance of Number of Pieces

- Too many pieces ⇒ extra overhead
- Too few pieces ⇒ less even load balancing

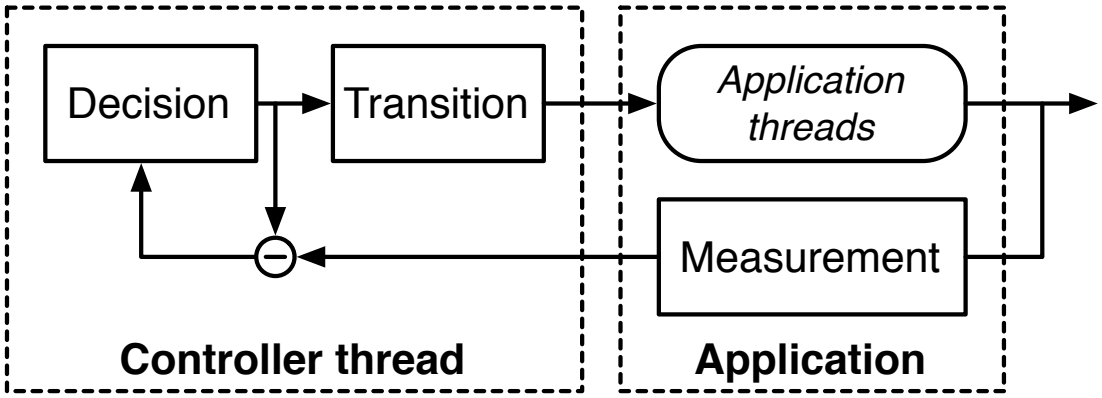
vacation-low





Thread Management

- Exploration-based scaling
  - Measure performance of application threads
    - Period duration adjusted according to throughout
  - Explore neighboring/random configurations
    - Adjust the number of threads
  - Upon improvement continue, otherwise revert



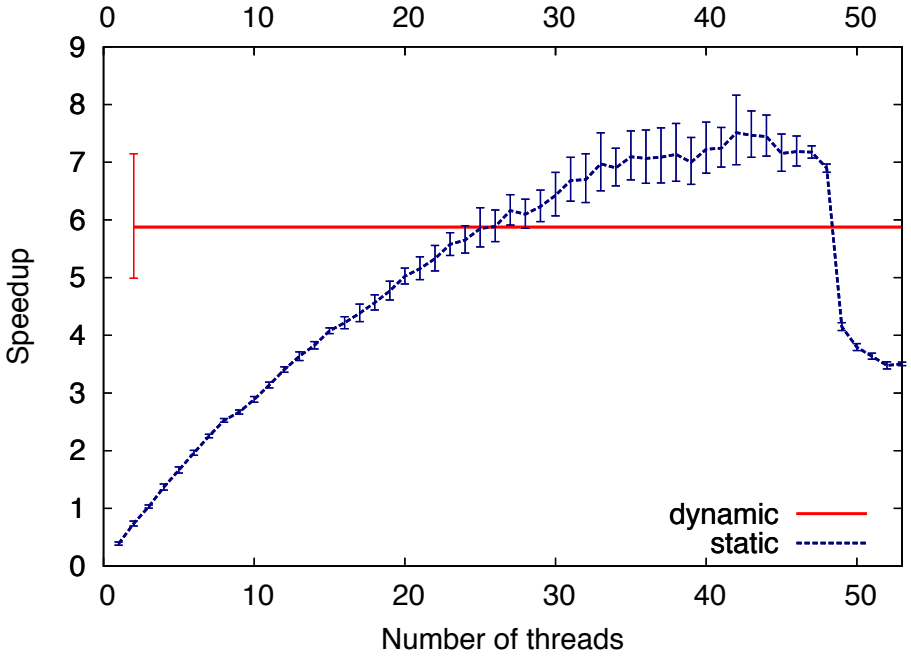
Hill Climbing Algorithm

- Simple hill-climbing algorithm
  - Increase #threads
    - If **performance improves**: keep on adding threads
    - If **performance degrades**: revert #threads
  - Decrease #threads
    - If **performance improves**: keep on removing threads
    - If **performance degrades**: revert #threads
  - Periodically choose random #threads
    - If **performance improves**: keep #threads
    - If **performance degrades**: revert #threads
  - Step sizes: linear or logarithmic

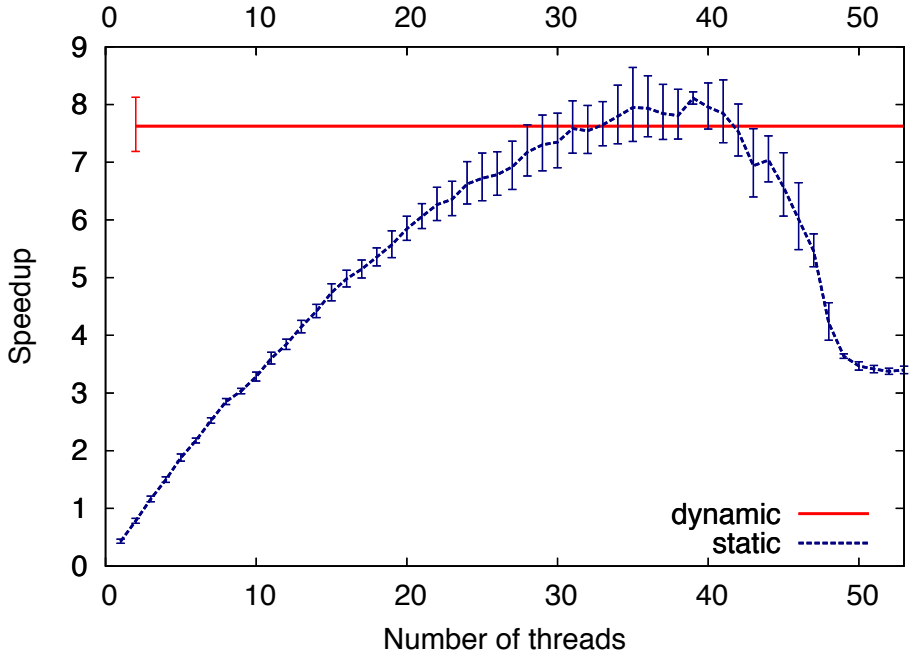
Exploration-Based Scaling

STAMP

vacation-high

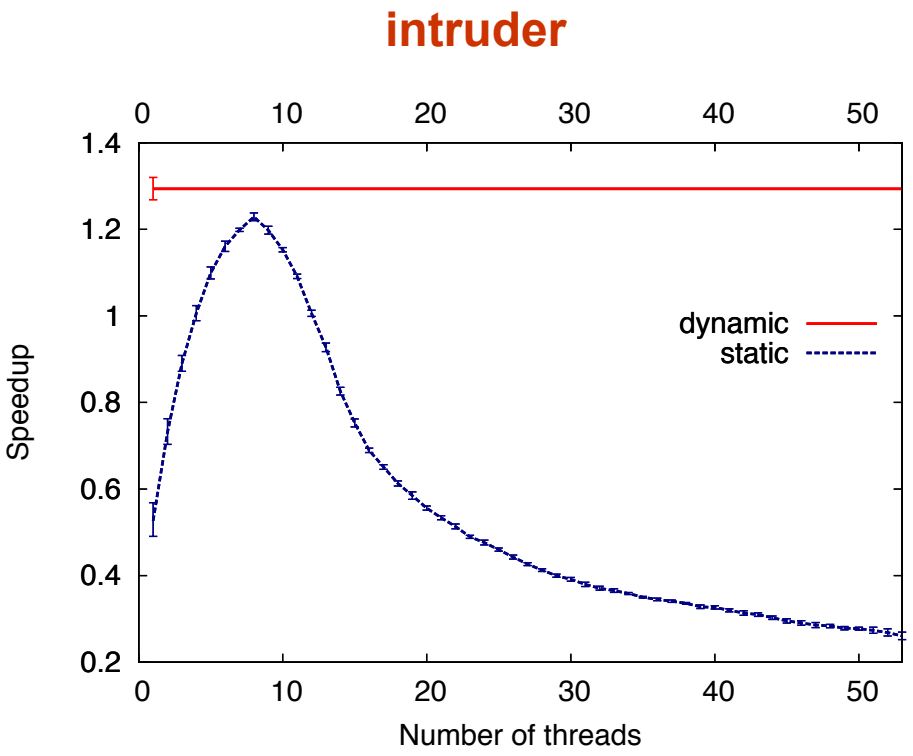


vacation-low



Exploration-Based Scaling

STAMP



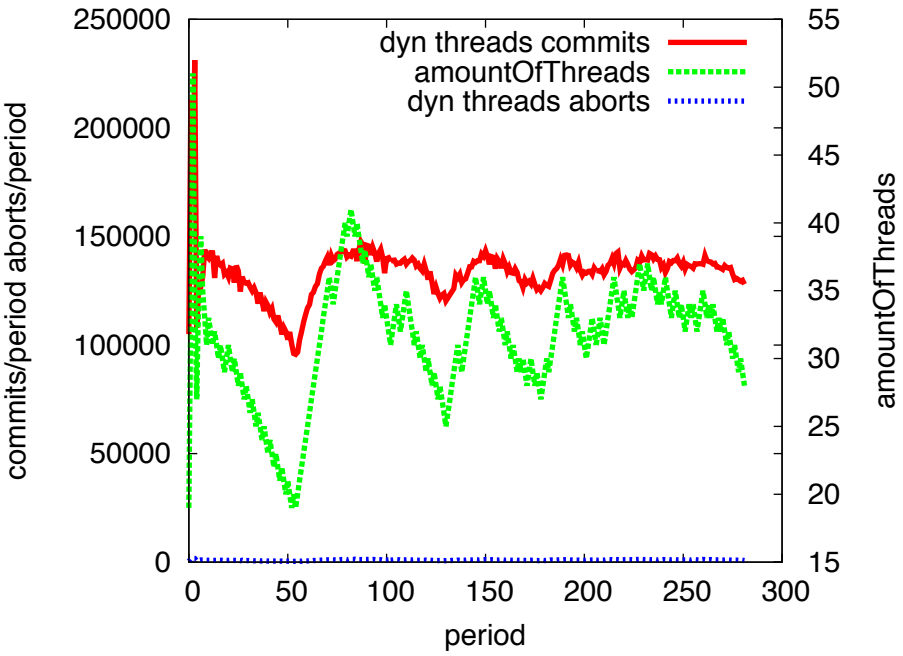
Performance is better than with any static number of threads!

How can that be?

Exploration-Based Scaling

STAMP

vacation-low

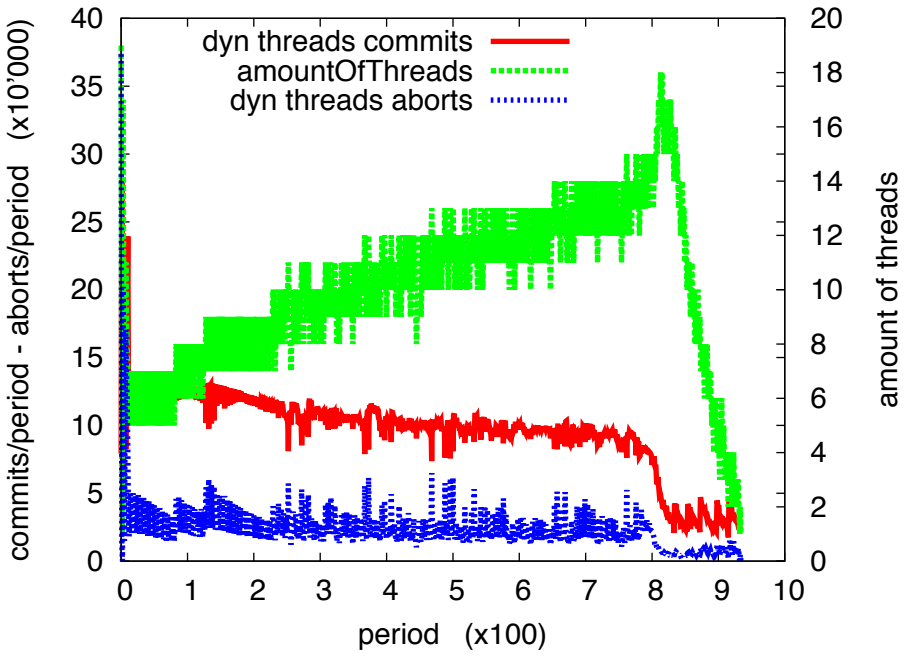


Usually we slowly converge toward an optimal value

Exploration-Based Scaling

STAMP

intruder



With intruder, the number of threads continuously evolves (no optimal value?)

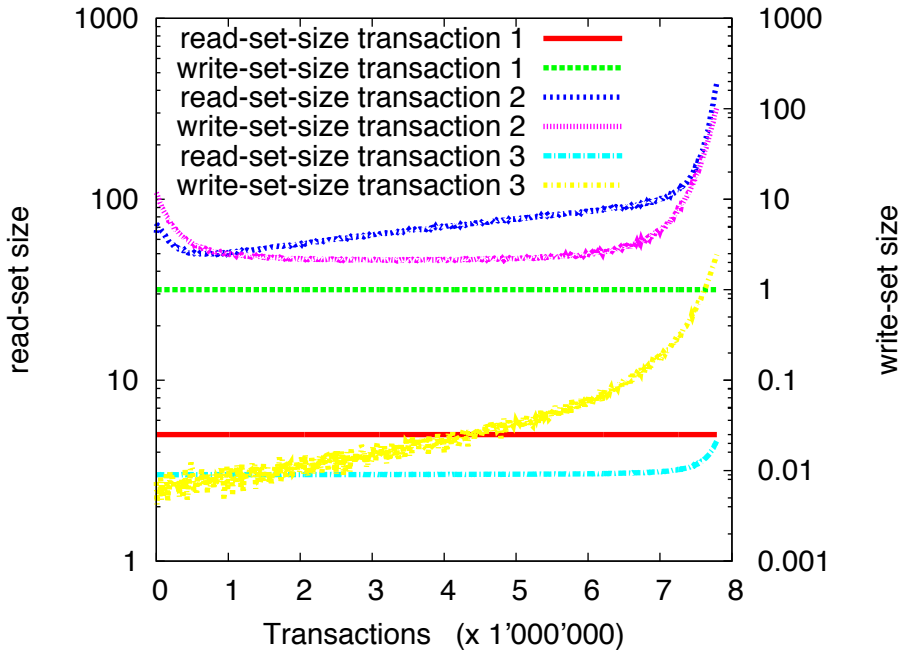
Exploration-Based Scaling

- The intruder benchmark models an intrusion detection system
  - Processes fragments of messages
- Three types of transactions
  - T1 – Get network packet
  - T2 – Find position in data structure
    - If message complete, search for intrusions
  - T3 – Store information about detected intrusions

Exploration-Based Scaling

STAMP

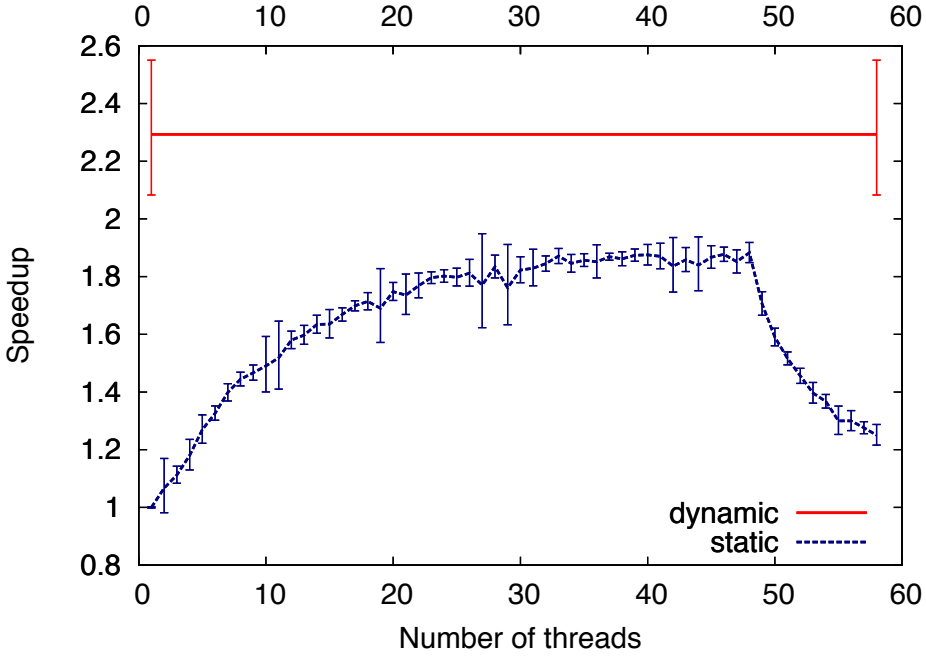
intruder

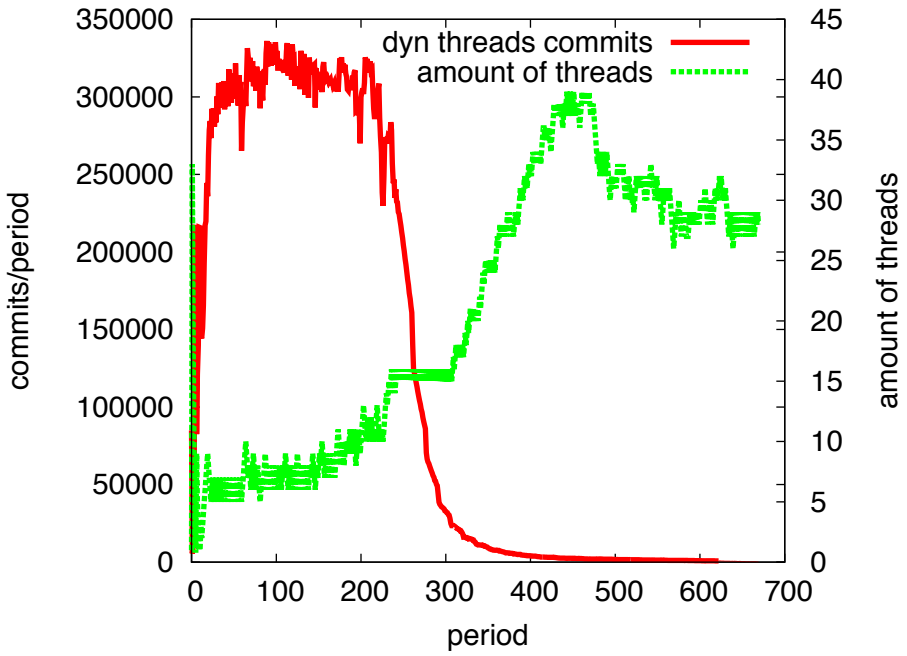


- 2nd/3rd transactions grow
- Cost of 2nd transaction dominates
- More conflicts near the end of the execution (larger data structures)

Exploration-Based Scaling

Synthetic benchmark





Dynamic Workloads

- Some workloads have dynamically-changing properties
  - Varying transaction lengths (e.g., as data structures get populated)
  - More conflicts at certain times
  - Different mixes of transactions
- The optimal degree of parallelism is not always a constant value
- Likely to happen in real-world applications (?)

Exploration vs. Modeling

- Alternative: predictive models to forecast the impact of adding/removing threads
  - Good for distributed TM (exploration does not scale due to cost of adding nodes)
  - Accurate models hard to get right
- Idea: combine both approaches
  - Online exploration can improve model's accuracy
  - Analytical model can improve scalability of online technique
  - Joint work with D. Didona & P. Romano

Summary

- Performance depends on workload
  - Driven mainly by conflicts
  - Long transactions, large write sets, etc.
- There is no one-size-fits-all TM
  - Many dimensions for adaptation
  - Degree of concurrency particularly important
- Some workloads evolve at runtime
  - Dynamic adaptation techniques can help TM adjust to changing workloads
- One more knob: optimistic vs. pessimistic?