Synchronization Strategies on Many-Core SMT Systems

Agustín Navarro-Torres[1], Maria Carpen-Amarie[2], Jesús Alastruey-Benedé[1], Pablo Ibáñez-Marín[1]

[1]Universidad de Zaragoza
{**agusnt**,jalastru,imarin}@unizar.es

[2]Zürich Research Center, Huawei Technologies Switzerland
maria.carpen.amarie@huawei.com

October 12, 2021

Introduction

- The number of native threads can reach hundreds
- Increasing complexity of synchronization mechanism
- Several synchronization mechanisms

Contributions

- Extensive scalability analysis
- Hardware Transactional Memory easy-to-use case
- Evaluation of Simultaneos Multithreading impact on Hardware Transactional Memory
- Transactional Aware Replacement policy for cache

Synchronization Strategies

- Clasical Strategies: Locks and Lock-Free
- Hardware Transactional Memory

Synchronization Strategies

- **Clasical Strategies: Locks and Lock-Free**
  - Control access to data region
  - Requires in-depth knowledge
  - Common programming errors
- Hardware Transactional Memory

Synchronization Strategies

- Clasical Strategies: Locks and Lock-Free
- **Hardware Transactional Memory**
  - Section of code as a transaction
  - A transaction are executed atomically
  - Easy-of-use
  - Optimistic execution: higher parallelism

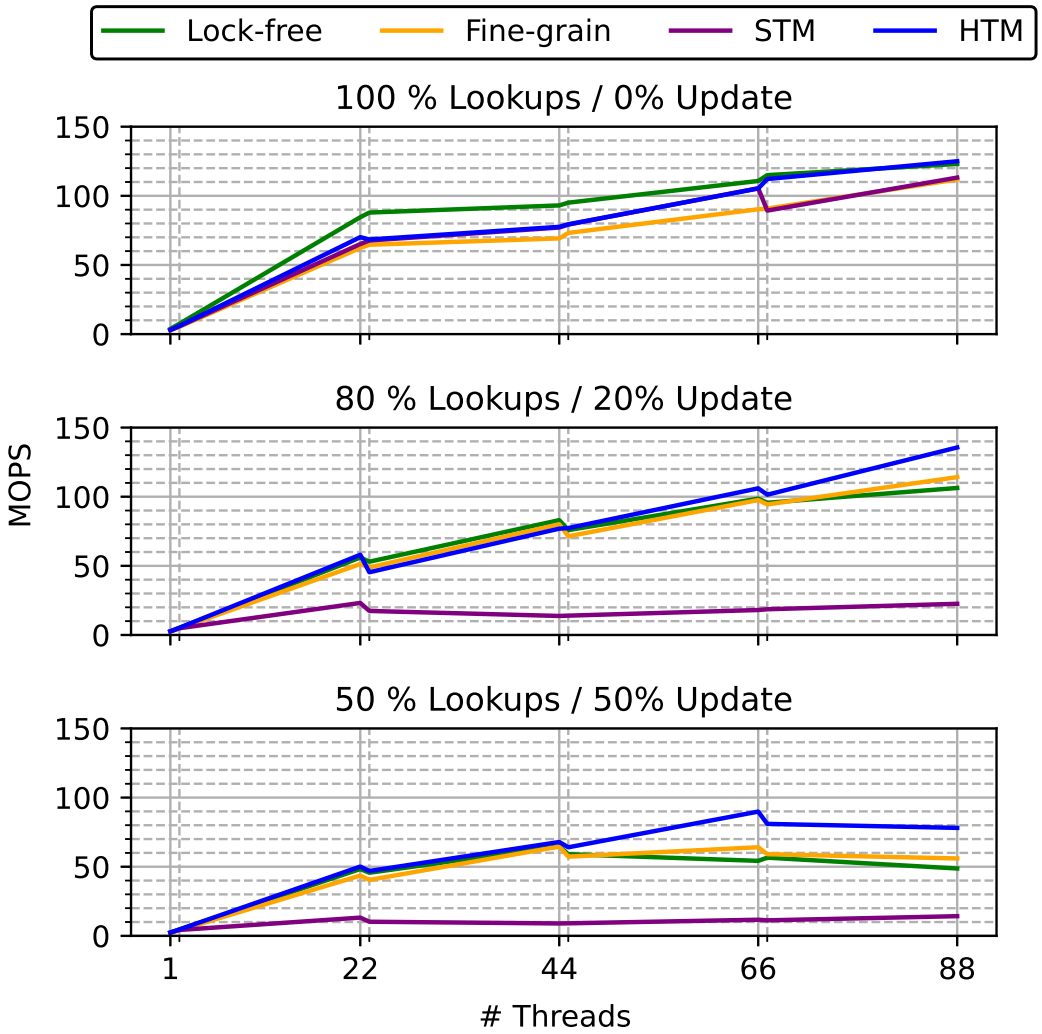
Experimental Setup

Processor	2×Intel Xeon Gold 6238T
Threads	2×44 (88)
Cores	2×22 (44)
Family	Cascade Lake
OS	Ubuntu 20.04
Kernel	5.4.0
NUMA policy	Default
Governor	Performance
Compiler	GCC 9.3.0

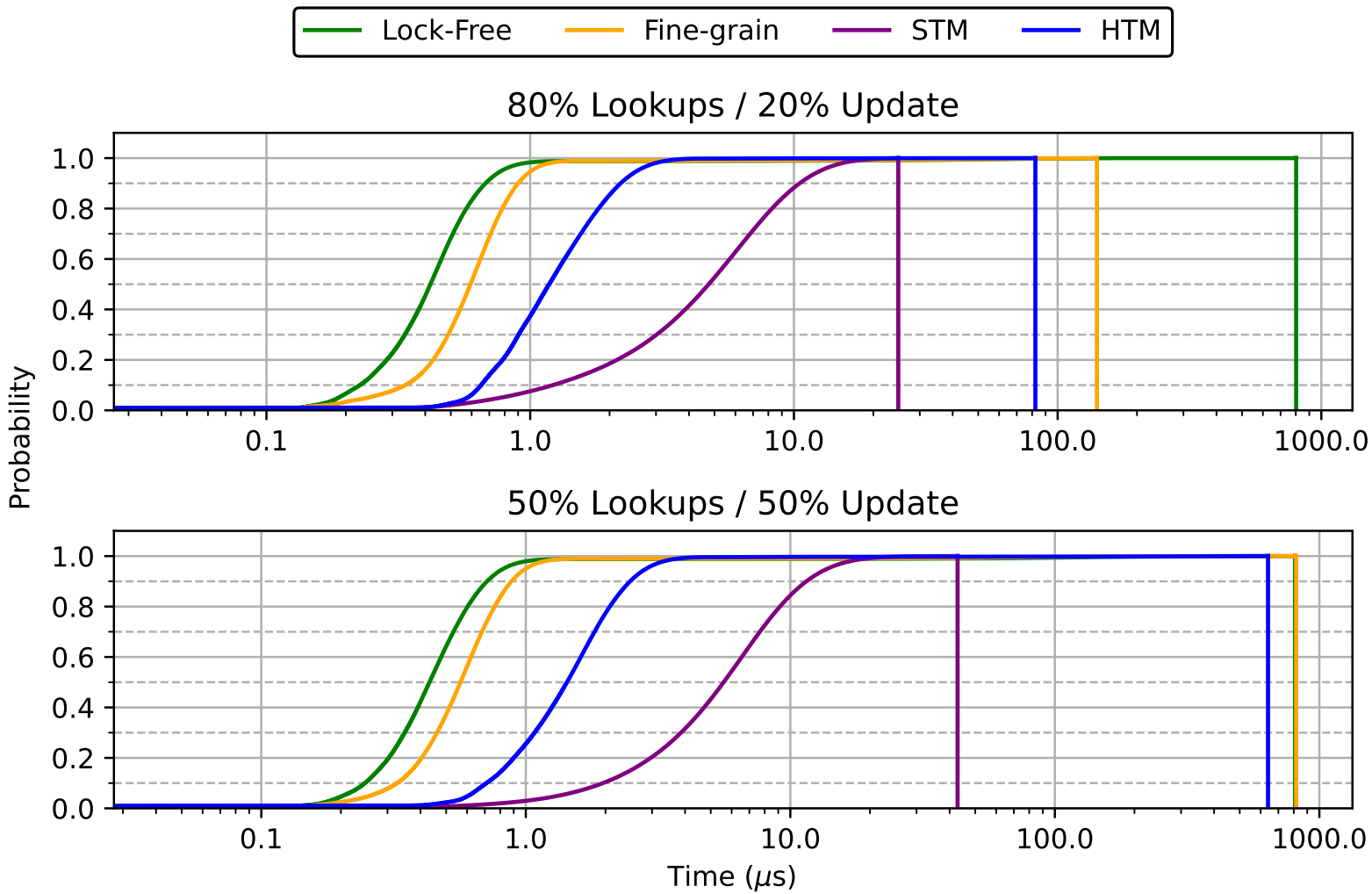
Scalability of Syncronization Strategies

- Two widely use data structure: **Hash Table** & Binary Search Tree
- High performance implementations
- Three diffent Workloads:
  - 100% Lookups operations
  - 80% Lookups and 20% updates operations
  - 50% Lookups and 50% updates operations

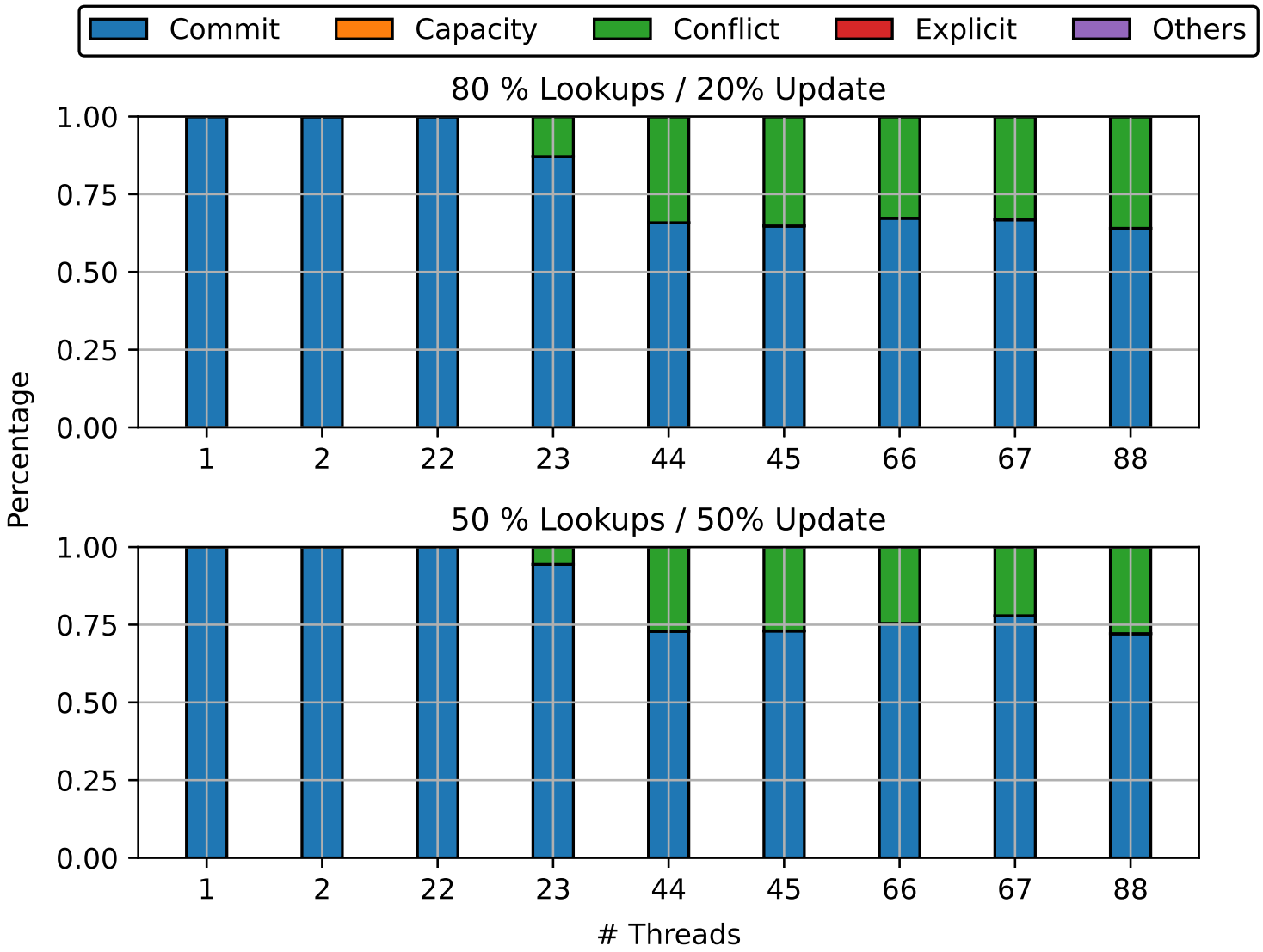
Scalability of Syncronization Strategies: Hash Table Througput



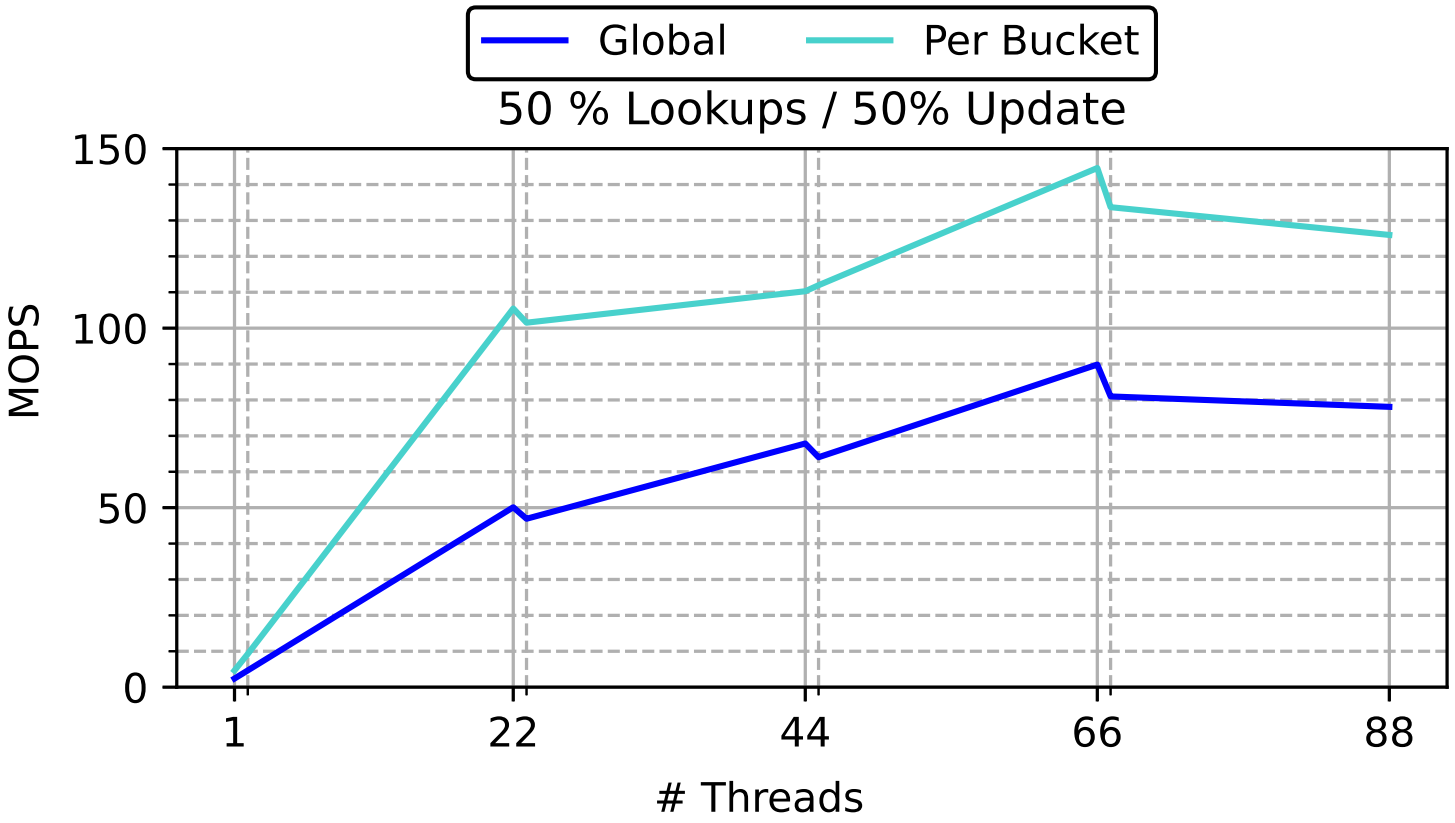
Scalability of Syncronization Strategies: Hash Table Latency



Scalability of Syncronization Strategies: Hash Table Events



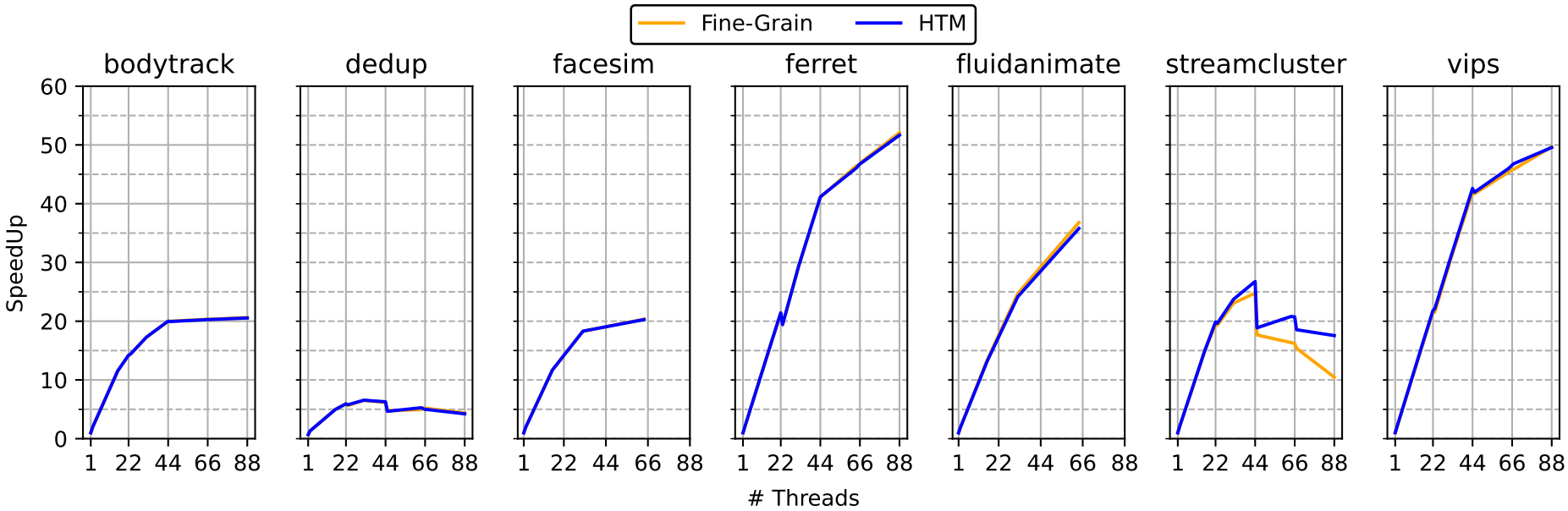
Scalability of Syncronization Strategies: Fallback Discussion



Case Study: HTM easy-of-use

- PARSEC 3.0 with locks and HTM
- HTM through GLIBC lock-elision
- Seven benchmarks ($>$ 100 locks)

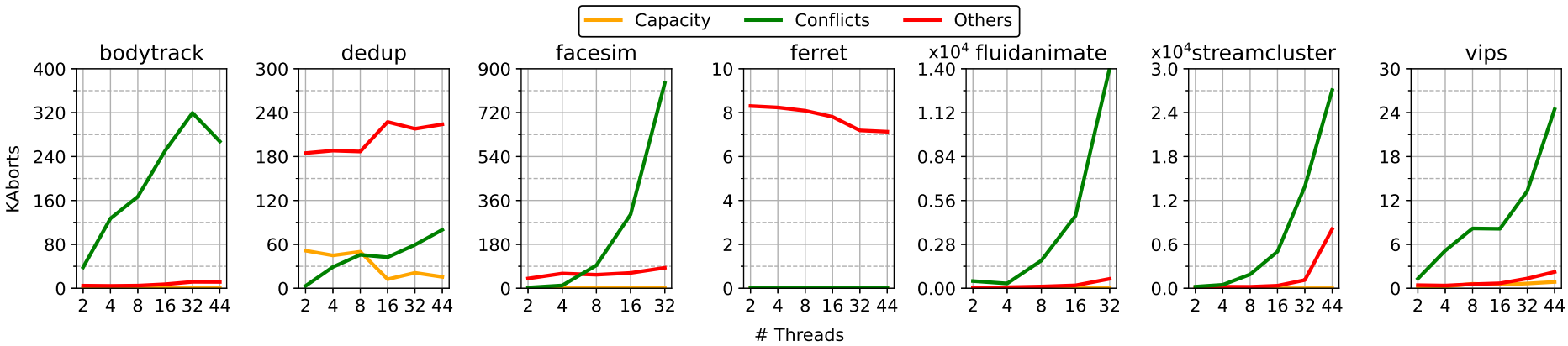
Case Study: HTM easy-of-use



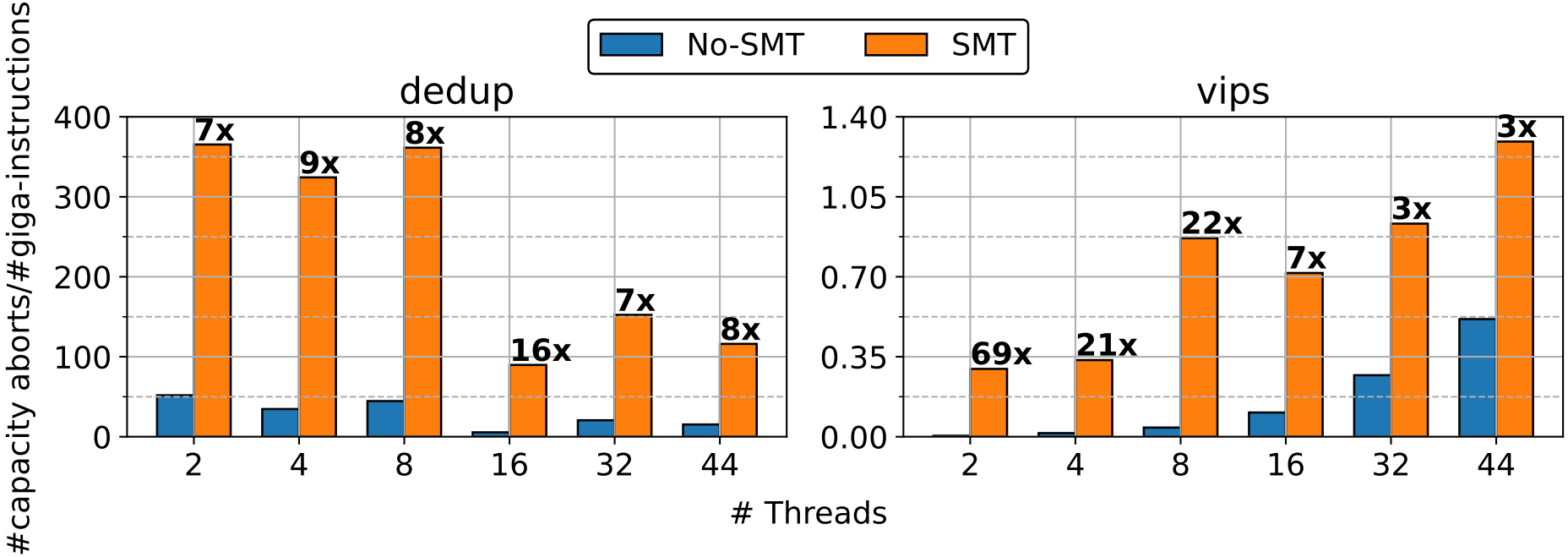
SMT Impact on HTM Capacity Aborts

- HTM implementation use cache hirearchy to track read/write set
- Quantify the SMT impact on real-world applications

SMT Impact on HTM Capacity Aborts



SMT Impact on HTM Capacity Aborts



Transaction-Aware Replacement Algorithm

- Mechanism to mitigate the negative effects of SMT
- Mechanism build on top of the previous replacement algorithm
- Protect cache lines involved in a transaction
- Our mechanism decrease by 16 the capacity aborts
- 2% of performance improve in transactional thread
- 1% of performance loss in the non-transactional thread

Thank you for your attention! Any questions?