Database Workload

+ Low throughput (0.8 IPC on an 8-wide superscalar. 1/4 of SPEC)
+ Naturally threaded (and widely used) application
- Already high cache miss rates on a single-threaded machine (destructive interference could be a problem)

Key question:

Can SMT's simultaneous multi-thread instruction issue hide the latencies from additional misses in this already memory-intensive databases?

Workload & Methodology

On-line transactions processing (OLTP)

- updating account balances for a bank
- 16 server threads

Decision support systems (DSS)

- data warehousing, data mining
- answer business questions

Trace-driven simulation

- ATOM-generated instruction traces of Oracle 7.3.2
- 8-context, 8-wide SMT simulator

OLTP Characterization

Memory behavior for OLTP (1 context, 16 server processes)

Memory region	L1 cache miss rate (128KB caches)	Memory footprint
Instruction text	13.7%	556 KB
Program global area (PGA) (private)	7.4%	1.3 MB
Buffer cache	6.8%	9.3 MB
Metadata	12.9%	26.5 MB

- **High miss rates**
- **Large memory footprints**

Shared Resources are Critical

Almost all SMT hardware resources are **dynamically shared** by all executing threads

Benefit: better hardware resource utilization

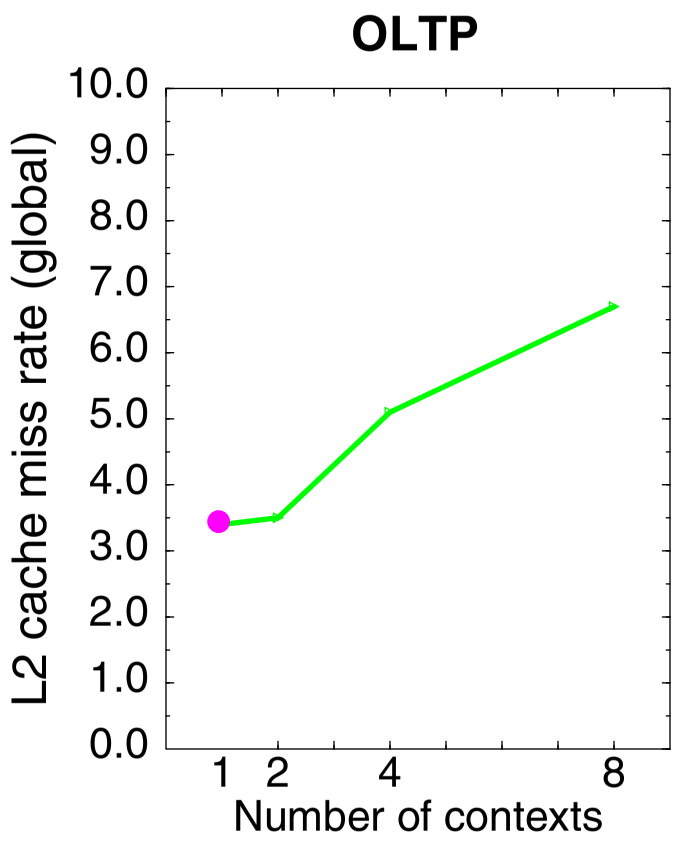
- SMT outperformed superscalar by 2-3X, single-chip MPs by 52%

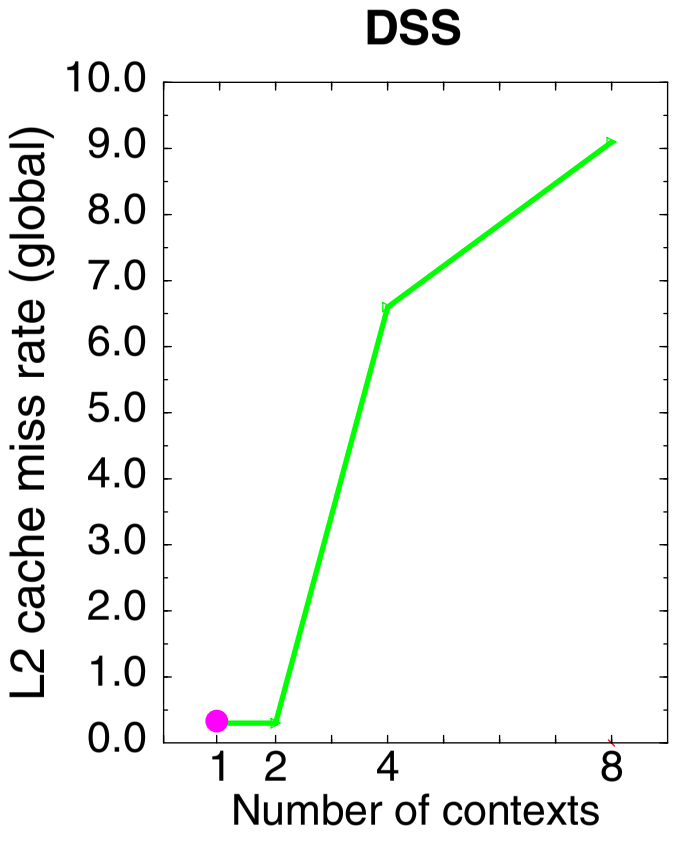
Cost: potential inter-thread contention

- shared hardware data structures contain the working set of **multiple** threads
- increase in some type of conflict

Managing the shared resources effectively may avoid the conflicts

L2 Cache Interference





- Interthread conflict misses

Critical Working Sets

Total foot print is too big, but how much of it do we really need?

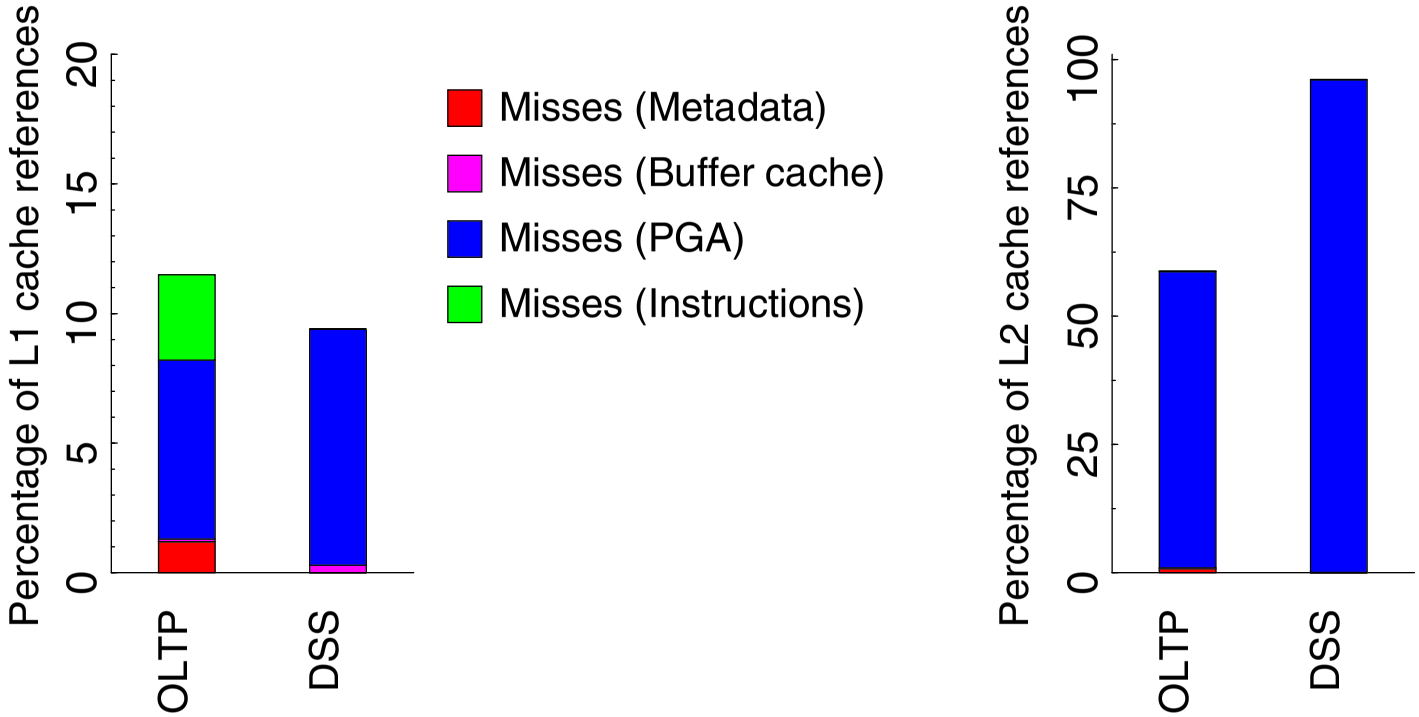
Some data is more important than others

- ***Skewed reference behavior***: a majority of references to a minority of memory blocks
  - for example, 87% of OLTP instruction references are to 31% of the instruction footprint; 98% & 6KB for DSS
  - for example, 41% of OLTP metadata references are to 26KB
- Commercial workloads have ***small critical working sets***

‹ Even with multiple threads, performance-critical working sets might fit in the caches

‹ Multithreading doesn't have to cause more misses

Cache Conflicts

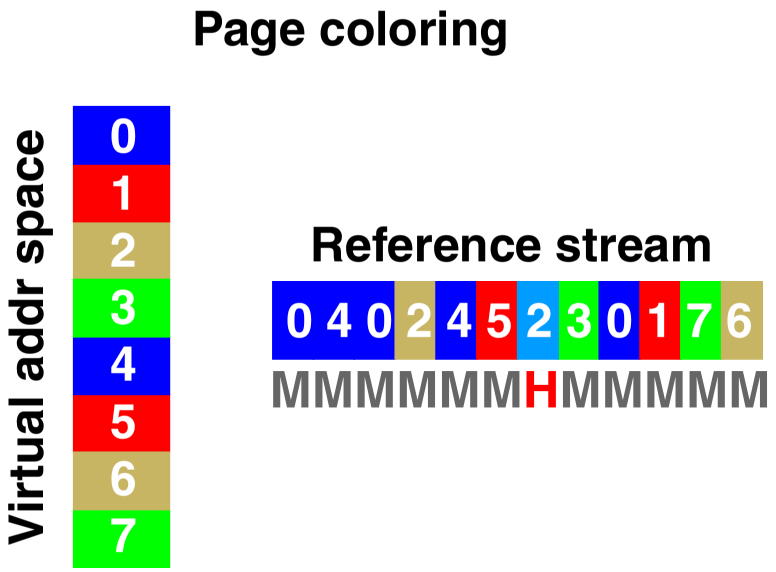


- L1 and L2 misses dominated by PGA references
- Conflict misses can be avoided by **page mapping** and **per-thread address offsetting**

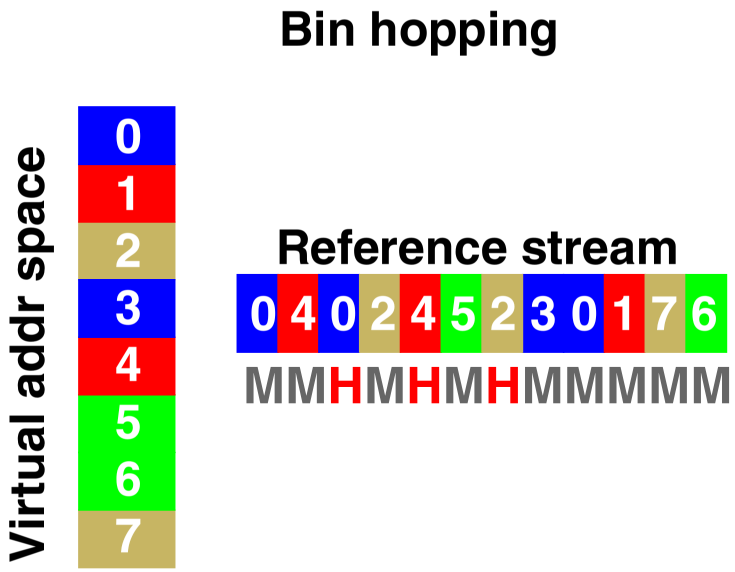
Page Mapping Alternatives

Map virtual pages to physical page frame

- example reference stream (virtual pages): 0 4 0 2 4 5 2 3 0 1 7 6

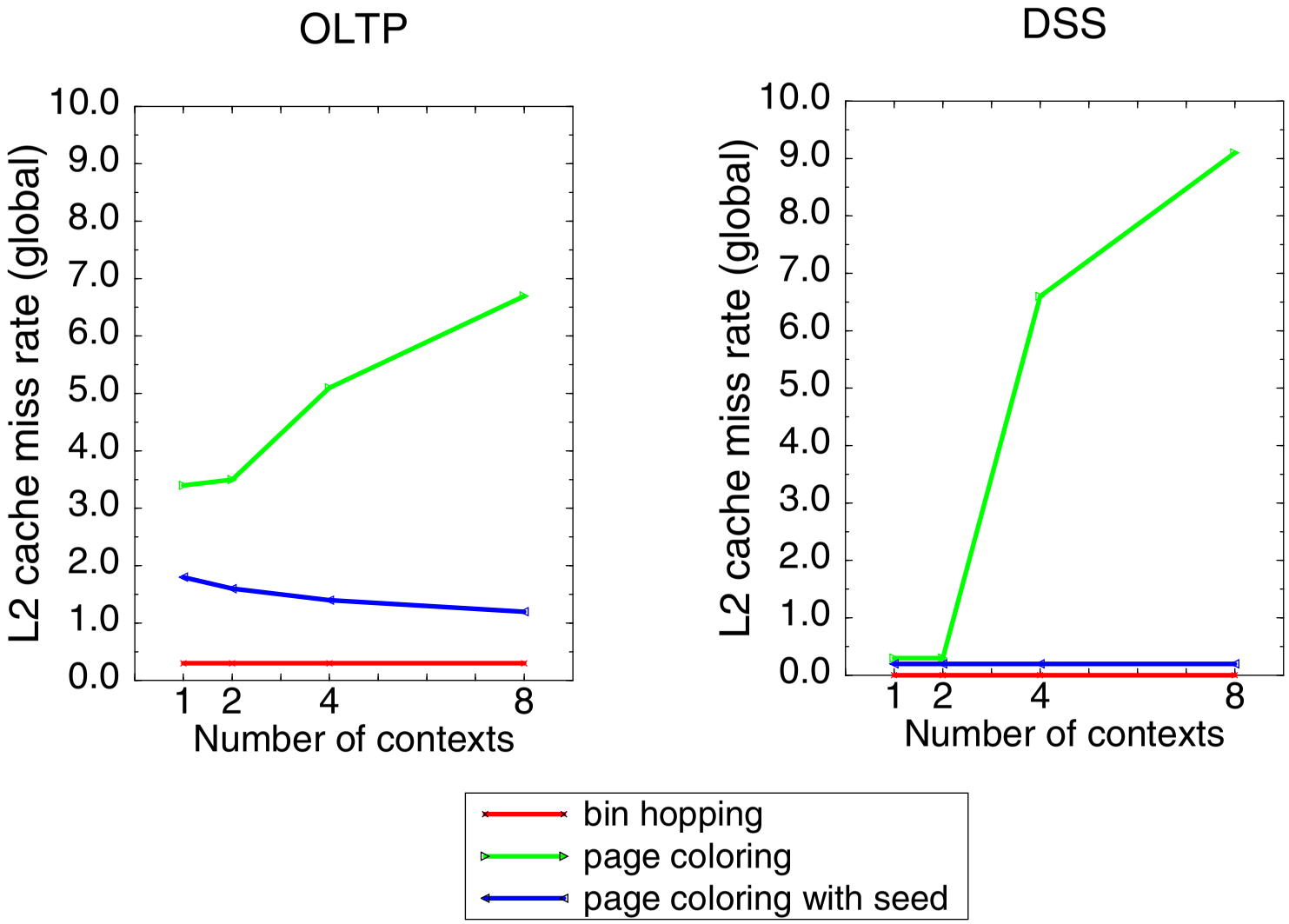


- Page coloring exploits spatial locality

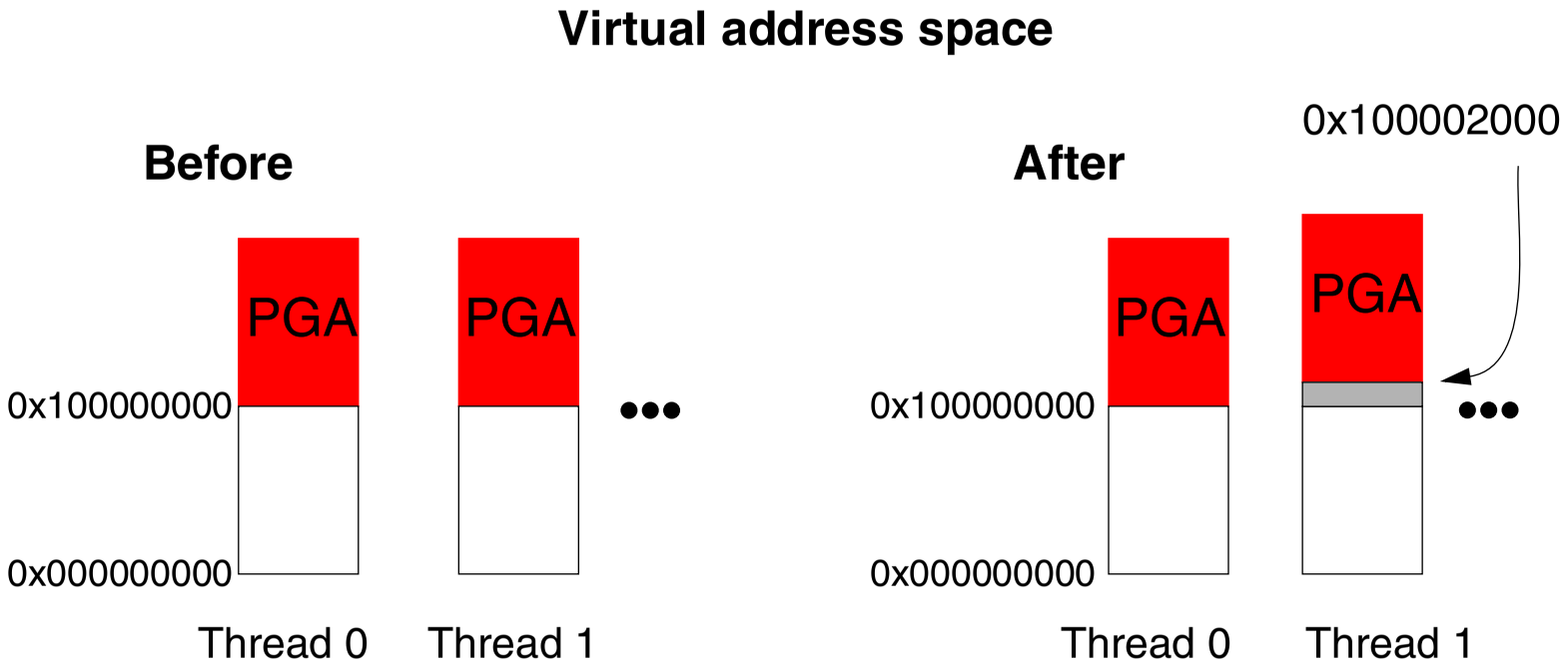


- Bin hopping exploits temporal locality

Page Mapping Results

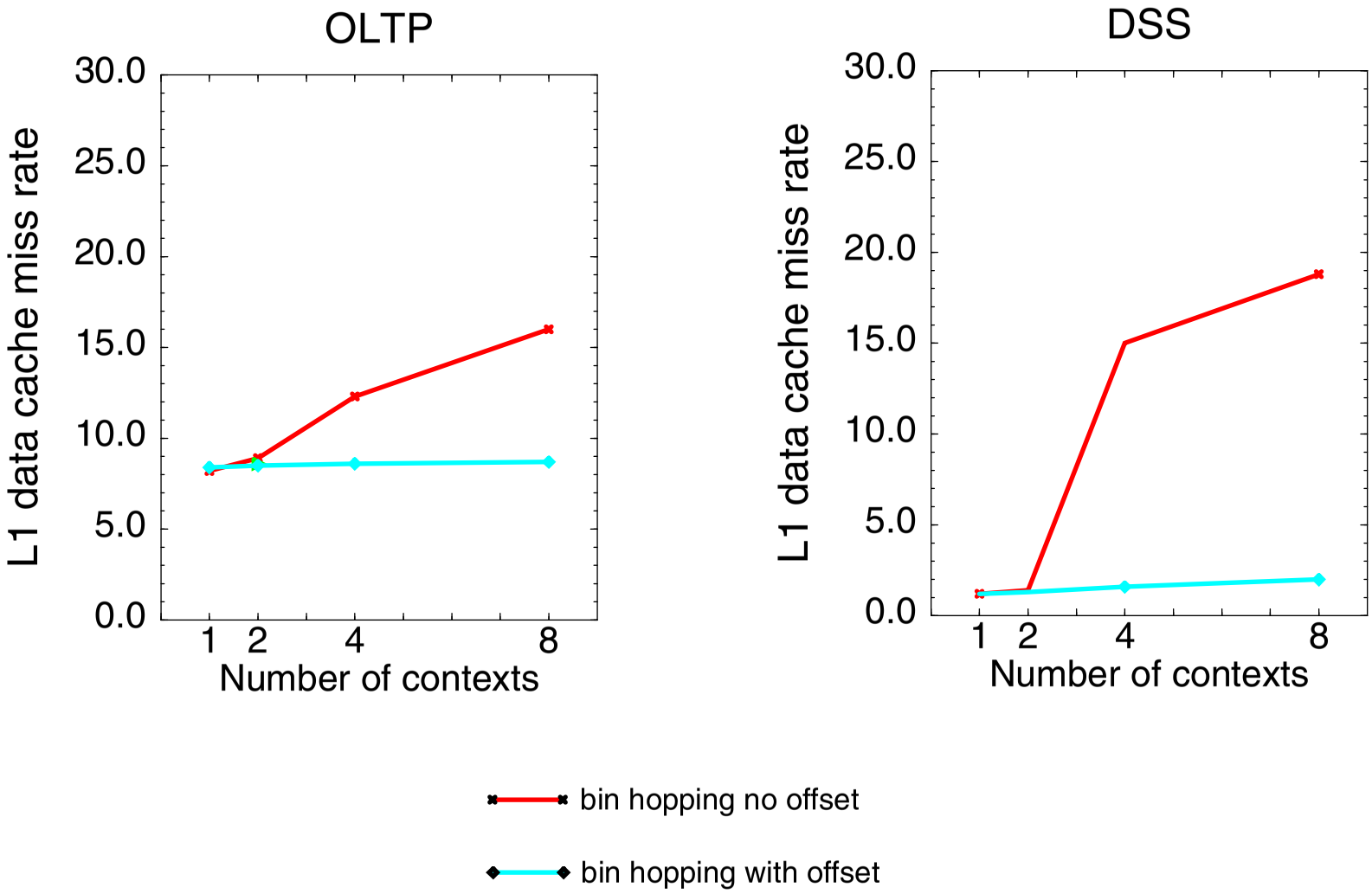


Application-level Offsetting



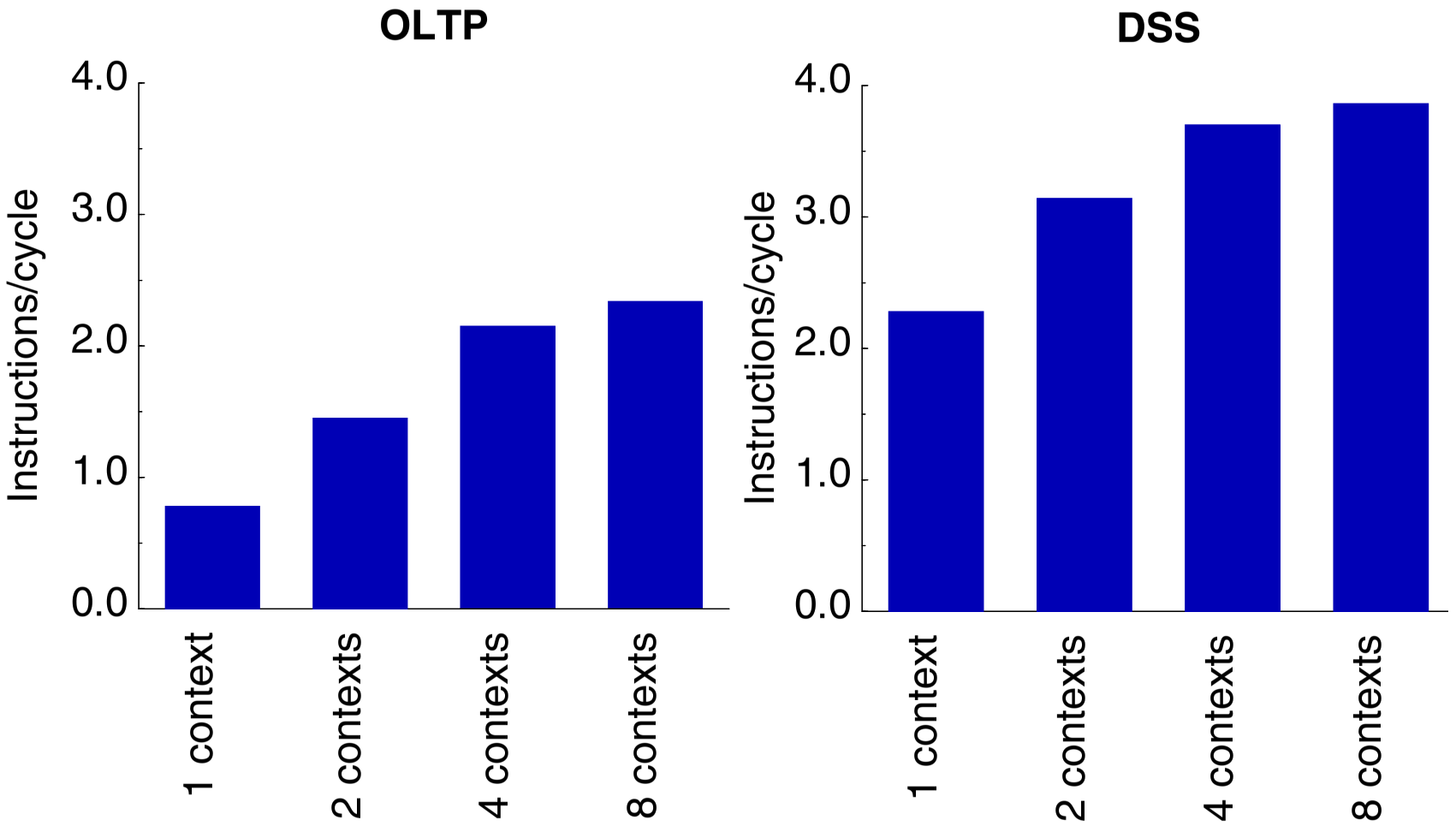
- Base of each thread's PGA is at the same virtual address
- Causes inter-thread conflicts in (virtually-indexed) L1 cache
- Address offsets can avoid interference (thread id * 8KB)

Offsetting Results



SMT Performance

with bin hopping, application offsetting, L1 I-cache thread-sharing



Summary

SMT gets good speedups on commercial database workloads

- Limits inter-thread data interference to superscalar levels
  - virtual-to-physical page mapping policies
  - address offsetting for thread-private data
- Exploits inter-thread instruction sharing
  (35% reduction in instruction cache miss rates)

SMT gets good speedups on commercial database workloads

- 3x for debit/credit systems, 1.5 for big search problems