STEPS Towards Cache-Resident Transaction Processing

Stavros Harizopoulos

joint work with Anastassia Ailamaki

VLDB 2004 Carnegie Mellon
OLTP workloads on modern CPUs

- L1-I stalls account for 25-40% of execution time
- Instruction caches cannot grow
  - We need a solution for instruction cache-residency
Steps for cache-resident code

- Eliminate misses for a group of Xactions
  - Xactions are assigned to threads
  - Multiplex execution at fine granularity
  - Reuse instructions in L1-I cache

👉 STEPS: Synchronized Transactions through Explicit Processor Scheduling
Fewer misses & misspred. branches

- Up to 1.4 speedup
- Eliminate 96% of L1-I misses for each add’l thread
- Eliminate 64% of mispredicted branches
Outline

• Background & related work
• Basic implementation of Steps
• Microbenchmarks
  - AthlonXP, SimFlex simulator
• Applying Steps to OLTP workloads
• TPC-C results
  - Shore on AthlonXP
Background

- Caches trade size for lookup speed
- L1-I misses are expensive

Example: 2-way set associative L1-I

<table>
<thead>
<tr>
<th>capacity misses</th>
<th>for loop {</th>
<th>F3</th>
<th>B ( ) {</th>
<th>B1 }</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (?) call B</td>
<td>F4 }</td>
<td></td>
<td>F1</td>
<td>F2</td>
</tr>
<tr>
<td>F1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>cache block</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

L1-I cache

L1-D cache

L2 cache

data

code
Background

- Caches trade size for lookup speed
- L1-I misses are expensive

Example: 2-way set associative L1-I

<table>
<thead>
<tr>
<th>capacity misses</th>
<th>conflict misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>for loop {</td>
<td>B ( ) {</td>
</tr>
<tr>
<td>if (?) call B</td>
<td>B1 }</td>
</tr>
<tr>
<td>F1</td>
<td>F3</td>
</tr>
<tr>
<td>F2</td>
<td>cache block</td>
</tr>
<tr>
<td></td>
<td>F4 }</td>
</tr>
</tbody>
</table>

+ larger cache size
+ higher associativity

slower access to L1-I cache
slower CPU clock

Databases
@Carnegie Mellon
Related work

• Database & Architecture papers:
  - DB workloads are increasingly non I/O-bound
  - L2/L3 data misses, L1-I misses
  - ORACLE OLTP code working set 560KB

• Hardware & compiler approaches
  - Increase block size, add stream buffer [asplos98]
  - Call graph prefetching (for DSS) [tocs03]
  - Code layout optimizations [isca01] [..]
Related work: within the DBMS

• Data-cache misses (mostly DSS)
  - Cache aware page layout, B-trees, join algorithms
  - Active area [..]

• Instruction-cache misses in DSS
  - Batch processing of tuples [icde01][sigmod04]

• Instruction-cache misses in OLTP
  Challenging!
Outline

• Related work
• Basic implementation of Steps
• Microbenchmarks
• Applying Steps to OLTP workloads
• TPC-C results
Steps overview

• DBMS assign Xactions to threads
• Xactions consist of few basic operators
  – Index select, scan, update, insert, delete, commit
• Steps groups threads per Op
• Within each Op reuse instructions
  ➤ I-cache aware context-switching
I-cache aware context-switching

**BEFORE**

- Thread 1:
  - Instruction cache:
    - select( )
    - Miss
    - s1
    - M
    - s2
    - M
    - s3
    - M
    - s4
    - M
    - s5
    - M
    - s6
    - M
    - s7
    - M

- Thread 2:
  - Instruction cache:
    - Miss
    - s1
    - M
    - s2
    - M
    - s3
    - M
    - s4
    - M
    - s5
    - M
    - s6
    - M
    - s7
    - M

**AFTER**

- Thread 1:
  - Instruction cache:
    - Miss
    - s1
    - M
    - s2
    - M
    - s3
    - M
    - s4
    - M
    - s5
    - M
    - s6
    - M
    - s7
    - M

- Thread 2:
  - Instruction cache:
    - Hit
    - s1
    - H
    - s2
    - H
    - s3
    - H
    - s4
    - H
    - s5
    - H
    - s6
    - H
    - s7
    - H

**Notes:**
- Code fits in I-cache
- Context-switch point (CTX)
- CPU executes code
- CPU performs context-switch (CTX)
Basic implementation on Shore

• Assume (for now)
  - Threads interested in same Op
  - Uninterrupted flow (no locks, I/O)

- Fast, small, compatible CTX code
  - 76 bytes, bypass (for now) full CTX

- Add CTX calls throughout Op source code
  - Use hardware counters (PAPI) on sample Op
Outline

• Related work
• Basic implementation of Steps
• Microbenchmarks
• Applying Steps to OLTP workloads
• TPC-C results
Microbenchmark setup

• All experiments on *index fetch*, in-memory index
  - 45KB footprint
• Fast CTX for both *Steps / Shore*, warm cache

<table>
<thead>
<tr>
<th></th>
<th>L1 I + D cache size</th>
<th>64KB + 64KB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>associativity</td>
<td>2-way</td>
</tr>
<tr>
<td></td>
<td>block size</td>
<td>64 bytes</td>
</tr>
<tr>
<td><strong>L2 cache size</strong></td>
<td></td>
<td>256KB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>vary all cache parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Simulated IA-32</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SimFlex</strong></td>
<td></td>
</tr>
</tbody>
</table>

**AMD AthlonXP**
Steps eliminates 92-96% of misses for add’l threads
• All misses are conflict misses (cache is 64KB)
L1-I misses & speedup

- Steps achieves max performance for 6-10 threads
- No need for larger thread groups
- **Steps** achieves max performance for 6-10 threads
- No need for larger thread groups
Steps outperforms Shore even on smaller caches (PIII)
- 62-64% fewer mispred. branches on both CPUs
SimFlex: L1-I misses 10 threads 64b cache block

- **Steps** eliminates all capacity misses (16, 32KB caches)
- Up to 89% overall miss reduction (upper limit is 90%)
Outline

- Related work
- Basic implementation of *Steps*
- Microbenchmarks
- Applying *Steps* to OLTP workloads
- TPC-C results
Design goals

• High concurrency on similar Ops
  - Cover full spectrum of Ops

• Correctness & low overhead for:
  - Locks, latches, mutexes
  - Disk I/O
  - Exceptions (abort & roll back)
  - Housekeeping (detect deadlock, buffer pool)
Overview

1. Thin wrappers per Op to sync Xactions
   - Form *Execution Teams* per Op
   - Flexible definition of Op

2. Best-effort within execution teams
   - Fast CTX through fixed scheduling
   - Threads leave team on exceptions

3. Repair thread structures at exceptions
   - Modify only thread package
System design

• Threads go astray on exceptions
• Regroup at next Op
• Can have execution teams per database table
Outline

• Related work
• Basic implementation of Steps
• Microbenchmarks
• Applying Steps to OLTP workloads
• TPC-C results
Experimentation setup

- **Shore/Steps**: AthlonXP, 2GB RAM, 2 disks
- **Shore locking**
  - Hierarchy: record, page, table, DB
  - Protocol: 2-phase
- **TPC-C**: Wholesale parts supplier
  - 10-30 Warehouses, 100-300 users
- Increased concurrency though
  - Zero think time TPC-C workload
  - In-memory database, lazy commits
Steps outperforms Shore
- 1.4 speedup, 65% fewer L1-I misses
- 48% fewer mispredicted branches
- For 10 Warehouses: 15 ready threads, 7 threads / team
• Xaction mix reduces average team size (4.3 in 10W)
• Still, *Steps* has 56% fewer L1-I misses (out of 77% max)
Summary of results

- *Steps* can handle full OLTP workloads
- Significant improvements in TPC-C
  - 65% fewer L1-I misses
  - 48% fewer mispredicted branches
- Room for improvement
  - *Steps* was not tuned for TPC-C
  - *Shore’s* code yields low concurrency

*Steps* minimizes both capacity & conflict misses without increasing I-cache size / associativity
Thank you