Composing Transaction Scalability on Multicore Platforms

Anastasia Ailamaki

Pınar Tözün, Danica Porobic, and Erietta Liarou (EPFL)
Islam Atta and Andreas Moshovos (U Toronto)
“performance” means scalability
increasing HW contexts

TPC-C (64) Payment on Shore-MT
Sun Niagara T2

enemy #1: concurrency control

Linear Scalability
distributed transactions

% Multisite Transactions in Workload

throughput

throughput

Shared-Nothing

Shared-Everything

Best thread-to-core

Worst

enemy #2: communication latency
cache efficiency

At peak throughput
On Shore-MT

Execu?on	
  cycles	
  breakdown

Instruc.on	
  Stalls
Other	
  Stalls
Busy

TPC-C	
  TPC-E

Instruc?ons	
  per	
  cycle

Intel	
  Xeon	
  X5660
Max.

enemy #3: instruction misses
composing OLTP scalability

- alleviate concurrency control
- observe variability in comm latency
- maximize instruction locality
shared-everything OLTP

contention due to unpredictable data accesses
physiological partitioning (PLP)

PLP eliminates 70% of the critical sections
PLP on multicores

Sun Niagara T2
64 HW contexts, in order, 1.4GHz

TATP – GetSubData

4 socket Quad AMD
16 HW contexts, OoO, 2.8GHz

higher benefit as # of HW contexts goes up
ugly duckling -> swan

Sun Niagara T1
Insert-only workload

(using DORA + PLP + Aether)
composing OLTP scalability

• alleviate concurrency control

• observe variability in comm latency

• maximize instruction locality
communication delays

<10 cycles

50 cycles

500 cycles

latency varies wildly
Placement of application threads

Counter microbenchmark
8socket x 10cores

TPC-C Payment
4socket x 6cores

Throughput (Mtps)

Throughput (Ktps)

Unpredictable

40%

47%

39%

thread-to-core assignment matters
impact of sharing data among threads

Counter microbenchmark

8socket x 10cores

Counter per core

Counter per socket

Single counter

10000

1000

100

10

1

18.7x

516.8x

TPC-C Payment – local-only

4socket x 6cores

Throughput (Mtps)

Throughput (Ktps)

Shared nothing

Shared everything

fewer sharers -> better performance
impact of skewed input

4 Islands effectively balance skew and contention
composing OLTP scalability

- alleviate concurrency control
- observe variability in comm latency
- maximize instruction locality
L1-I misses are a significant factor in stall time
concurrent transactions

Database Operations
- Index Probe
- Index Scan
- Update Record
- Delete Record
- Insert Record

Transaction
1. Index Probe (X)
2. Update Record (X)
3. Index Probe (Y)
4. Insert Record (Y)
5. Delete Record (Z)

Threads
- T1
  - Index Probe (X1)
  - Update Record (X1)
  - Index Probe (Y1)
  - Delete Record (Z1)
- T2
  - Index Probe (X2)
  - Update Record (X2)
  - Index Probe (Y2)
  - Insert Record (Y2)
  - Delete Record (Z2)

execute many common instruction blocks
transactions on a single core

Stratified Transaction Execution (STREX)

phase leader

1 T1
2 T1
3 T1
4 T1
5 T2

Traditional

L1I
T1 T2 T3 T1 T2 T1 T3 T1 T2 T3

Miss penalty

Threads

T1
T2
T3

time-multiplexing to reduce instruction misses
transactions on multicores

<table>
<thead>
<tr>
<th>#Cache Fills</th>
<th>CORES</th>
<th>L1I</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T1</td>
<td>T2</td>
</tr>
<tr>
<td>3</td>
<td>T1</td>
<td>T2</td>
</tr>
<tr>
<td>6</td>
<td>T1</td>
<td>T2</td>
</tr>
<tr>
<td>9</td>
<td>T1</td>
<td>T2</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>T3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#Cache Fills</th>
<th>CORES</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T1</td>
<td>T2</td>
<td>T1</td>
</tr>
<tr>
<td></td>
<td>T2</td>
<td>T1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T3</td>
<td>T2</td>
<td>T1</td>
</tr>
<tr>
<td></td>
<td>T1</td>
<td>T2</td>
<td>T3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T3</td>
<td>T2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>T3</td>
</tr>
</tbody>
</table>

exploits aggregate L1-I & instruction overlap
reduced L1-I misses

Traces from Shore-MT

Simulation – 4-way OoO cores
4-way 32KB L1-I & L1-D, 1MB per core L2

Conventional
STREX
SLICC

TPC-C
#Cores
TPC-E

STREX reduces L1-I misses regardless of core count
SLICC is better on high core counts
hybrid: STREX + SLICC

Traces from Shore-MT

Simulation – 4-way OoO cores
32KB L1-I & L1-D, 1MB per core L2

TPC-C

TPC-E

up to 80% better
summary

• multicore = all parallelism methods
  – scalability a complex problem
• alleviate concurrency control
  – eliminate critical sections: DORA, PLP, Aether, etc
• observe variability in comm latency
  – OLTP Islands
• maximize instruction locality
  – chase locality with SLICC + STREX
what’s next?

- data partitioning across Islands
- transaction-aware thread migration
THANK YOU!