Digging parallelism out of a highly-conflicting workload

Nuno Diegues    João Cachopo

nmld, joao.cachopo@ist.utl.pt

INESC-ID/Technical University of Lisbon

April 10, 2012
Problem
Problem

What is going wrong?
Problem

\begin{itemize}
  \item P_1: A
  \item P_2: B
  \item P_3: C
  \item P_4: D
\end{itemize}
Problem
Optimistic assumption defrauded
Explore parallelism within transactions
Solution
Solution
Solution
Outline

1. Parallel nesting
Outline

1. Parallel nesting
2. Transaction scheduling
Naive parallel nesting
Naive parallel nesting

```
A
write set
x: 5
B C
```
Naive parallel nesting

```
write set
x: 5
A

write set
y: 42
B

write set
y: 42
C
```

Diagram:

```
   A
  /|
 / |\write set
x: 5
B --
    |
    |
    |
   C
```
Naive parallel nesting
Naive parallel nesting

read x = ?
Naive parallel nesting

read x = ?

write set
x: 5

write set
y: 42

write set
Naive parallel nesting

write set
x: 5
write set
y: 42
read x = ?
Naive parallel nesting

read x = ?
Naive parallel nesting

read x = 5    read-after-write!
Naive parallel nesting

Write set
- A: x: 5
- B: y: 42
- E
- F

Read z = ?

Read operation worst-case: $O(d)$
Naive parallel nesting

\[ A \quad E \quad F \]
write set
\[ x: 5 \]
\[ B \quad C \]
write set
\[ y: 42 \]
read \( z = ? \)

Read operation worst-case: \( O(d) \)
## Read-after-writes in STMbench7, Lee-TM and Vacation

<table>
<thead>
<tr>
<th>test</th>
<th>reads ($\times 10^3$)</th>
<th>raws ($\times 10^3$)</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>bench7-r-notrav</td>
<td>8000</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>bench7-rw-notrav</td>
<td>9000</td>
<td>34</td>
<td>0</td>
</tr>
<tr>
<td>bench7-w-notrav</td>
<td>5000</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>bench7-r</td>
<td>83000</td>
<td>45000</td>
<td>54</td>
</tr>
<tr>
<td>bench7-rw</td>
<td>109000</td>
<td>65000</td>
<td>59</td>
</tr>
<tr>
<td>bench7-w</td>
<td>127000</td>
<td>71000</td>
<td>56</td>
</tr>
<tr>
<td>lee-mainboard</td>
<td>507000</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>lee-memboard</td>
<td>281000</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>lee-sparselong</td>
<td>67000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lee-sparseshort</td>
<td>1000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>vac-reservations</td>
<td>84000</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>vac-deletions</td>
<td>33000</td>
<td>600</td>
<td>1.8</td>
</tr>
</tbody>
</table>
Shared write-set per nesting tree

- **A** (write set: x: 5)
- **B** (write set: y: 42)
- **C**
- **E**
- **F**
Shared write-set per nesting tree
Shared write-set per nesting tree

Shared write-set per nesting tree

Profiling of read operations
Profiling of \textbf{write} operations
Writes in-place
Inplace: What was improved?

- Cheaper writes
- Reads still independent of depth
- Reduced commit-time:
  \[ O(\#children) \ll O(size(writeSet) + size(readSet)) \]
Profiling of read operations
Profiling of write operations
Profiling of commit operations
Evaluation

speedup

# threads: top-level (nested)

0 2 4 6 8 10 12 14
1(1) 1(2) 1(4) 1(8) 1(16) 2(16) 3(16)
Evaluation

1. Vacation from STAMP
2. STMBench7
3. Lee-TM
Evaluation

1. Vacation from STAMP
2. STMBench7
Evaluation

1. Vacation from STAMP
2. STMBench7
Vacation - **high** contention

![Graph showing speedup vs number of threads for nonest.](image)

- From 4.6 to 12.8 speedup
- Nonest line graph
Vacation - high contention

From 4.6 to 12.8 speedup
Vacation - **high** contention

From 4.6 to 12.8 speedup
Vacation - low contention
Vacation - low contention

Overhead is visible
Vacation - **low** contention

Overhead is visible
Evaluation

1. Vacation from STAMP
2. STMBench7
Parallel nesting in the wild

![Graph showing speedup vs. number of threads for different operations](image)

- r-top
- rw-top
- w-top

Speedup vs. # threads
Parallel nesting in the wild

![Graph showing speedup with different thread numbers for r-top, rw-top, w-top, r-par-nest, rw-par-nest, and w-par-nest.]
Parallel nesting in the wild

Missing one of the requirements in the motivation
Transaction Scheduling
Scheduling results - no nesting

Read-dominated: 1.22 to 1.37
Read-write: 0.87 to 1.19
Write-dominated: 0.84 to 1.21
Time spent computing by each thread

Side effect: processors are idle (rather than computing conflicting transactions)
STM Benchmark 7 - read-dominated

Speedup

# threads: top-level (nested)

Speedup

# threads: top-level (nested)
STMBench7 - **read**-dominated

![Graph showing speedup with varying numbers of threads](image)

The graph illustrates the speedup of STMBench7 with different thread counts, highlighting the impact of read-dominated operations.
STMBench7 - read-dominated

<table>
<thead>
<tr>
<th>Speedup</th>
<th># threads: top-level (nested)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>1(1)</td>
</tr>
<tr>
<td>1</td>
<td>1(2)</td>
</tr>
<tr>
<td>1.2</td>
<td>1(3)</td>
</tr>
<tr>
<td>1.4</td>
<td>2(3)</td>
</tr>
<tr>
<td>1.6</td>
<td>4(3)</td>
</tr>
<tr>
<td>1.8</td>
<td>8(3)</td>
</tr>
<tr>
<td>2</td>
<td>16(3)</td>
</tr>
</tbody>
</table>

Graph showing speedup for different thread counts and thread scheduling strategies.

Missing one of the requirements in the motivation.
STM Bench7 - read-dominated

From 1.22 to 2.39 speedup.
From 1.22 to 2.39 speedup.
STMBench7 - write-dominated

![Graph showing speedup vs. number of threads for top-level (nested)]

- Missing one of the requirements in the motivation
STM Bench 7 - write-dominated

Speedup vs. number of threads (top-level and nested):

- `top` (solid line)
- `top-schd` (dashed line)

The graph shows the speedup for different numbers of threads (1 to 16) for both top-level and nested execution. The speedup is calculated as the ratio of execution time with one thread to the execution time with multiple threads. The graph illustrates how the speedup changes as the number of threads increases, demonstrating the efficiency of the write-dominated benchmarks under varying thread counts.
STMBench7 - write-dominated

![Graph showing speedup by number of threads for top, top-schd, and par-nest.](image)

*Missing one of the requirements in the motivation*
STMBench7 - write-dominated

From 0.84 to 1.84 speedup.
From 0.84 to 1.84 speedup.
Thank you

Questions?
Embarrassingly parallelizable transaction

- More than one thread in the same transaction
- No validation required
- Accesses performed as if it was a top-level transaction
Lee-TM - mainboard

120% increase in performance