Accelerating conflict-intensive applications

Hugo Rito, João Cachopo

INESC-ID
Instituto Superior Técnico, Portugal

April 10, 2012
Overview
Overview

![Graph showing speedup with increasing number of threads for read-dominated access.](image)

- **Motivation**
- **Overview**

Speedup (1 thread):

- # Threads: 1, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48
- Read-dominated
Overview

![Graph showing speedup with varying number of threads for read-dominated workload. The graph peaks at approximately 16 threads and then decreases as the number of threads increases.]
Overview

The diagram illustrates the speedup (compared to 1 thread) for different thread counts. Two categories are shown:

- **Read-dominated**
- **Read-write**

The graph shows that for read-dominated threads, speedup increases with more threads until it plateaus, while for read-write threads, the speedup remains relatively stable. The x-axis represents the number of threads, and the y-axis represents the speedup (1 thread).
Overview

- **Motivation**
- **Overview**

![Graph showing speedup across different thread counts for read-dominated, read-write, and write-dominated scenarios.](image-url)
Overview
STMBench7: Write-dominated + Long-traversals
Problem

Single-threaded

$T_0$
Problem

Single-threaded

T₀

Multi-threaded

T₀

T₁
Problem

Single-threaded

T0

Multi-threaded

T0

T1

Slowdown
Problem

- Optimistic $\rightarrow$ Conflicts $\rightarrow$ Restarts
Problem

- Optimistic $\rightarrow$ Conflicts $\rightarrow$ Restarts

- Pessimistic $\rightarrow$ Limited concurrency $\rightarrow$ !Solution
Transaction execution

```java
@Atomic
public class Paint {
    (...)
    public Color paint() {
        if (       ==       ) {
                 =     ;
        } else {
                 =     ;
        }
        return      +     ;
    }
}
```
Transaction execution

public class Paint {
    (...)
    public Color paint() {
        if (       ==       ) {
                 =     ;
        } else {
                 =     ;
        }
        return      +     ;    
    }

@Atomic

Transaction execution

```java
public class Paint {
    (...)
    public Color paint() {
        if (       ==       ) {
                =     ;
        } else {
                =     ;
        }
        return      +     ;    
    }
}
```
Transaction execution

```java
@Atomic
public class Paint {
    (...)
    public Color paint() {
        if (       ==       ) {
                 =     ;
        } else {
                 =     ;
        }
        return      +     ;
    }
}
```
Transaction execution
Transaction execution

```java
@Atomic
public class Paint{
    (...)
    public Color paint() {
        if (       ==       ) {
                =     ;
        } else {
                =     ;
        }
        return      +     ;    
}
```
Transaction execution

```java
@Atomic
public class Paint {
    (...)
    public Color paint() {
        if (       ==       ) {
                 =     ;
        } else {
                 =     ;
        }
        return      +     ;
    }
}
```
Transaction execution
Transaction execution

```java
public class Paint {
    (...)
    public Color paint() {
        if (       ==       ) {
                 =     ;
        } else {
                 =     ;
        }
        return      +     ;    
    }
}
```
Transaction execution
Transaction execution

```
@Atomic
public class Paint {
    (...)
    public Color paint() {
        if (       ==       ) {
                 =     ;
        } else {
                 =     ;
        }
        return      +     ;
    }
```

Key Observation
Transaction execution

```java
@Atomic
public class Paint {
    (...)
    public Color paint() {
        if (       ==       ) {
                 =     ;
        } else {
                 =     ;
        }
        return      +     ;
    }
}
```
Transaction execution

```java
@Atomic
public class Paint {
    (...)
    public Color paint() {
        if (       ==       ) {
                 =     ;
        } else {
                 =     ;
        }
        return      +     ;    
}
```
Transaction execution

@Atomic
public class Paint {
    (...)
    public Color paint() {
        if (       ==       ) {
                 =     ;
        } else {
                 =     ;
        }
        return      +     ;
    }
}
Transaction execution

public class Paint {
    (...)
    public Color paint() {
        if (       ==       ) {
                 =     ;
        } else {
                 =     ;
        }
        return      +     ;    
}

@Atomic
Transaction execution

```
@Atomic
public class Paint {
    (...)
    public Color paint() {
        if (       ==       ) {
                 =     ;
        } else {
                 =     ;
        }
        return      +     ;    
    }
}
```
Transaction execution

```java
@Atomic
public class Paint {
    (...)
    public Color paint() {
        if (       ==       ) {
                 =     ;
        } else {
                 =     ;
        }
        return      +     ;    
}
```
Transaction execution

```java
@Atomic
public class Paint {
    (...)
    public Color paint() {
        if (       ==       ) {
                 =     ;
        } else {
                 =     ;
        }
        return      +     ;    
    }
}
```
Similar (re)executions

@Atomic
Similar (re)executions

@Atomic @Atomic

[Diagram showing two similar execution flows with a red 'X' indicating a failure or issue]
Similar (re)executions

`@Atomic`  `@Atomic`
Green STM
Green STM
Green STM
Green STM

Recycle
(faster reexecutions)
Approach

Green STM

Reuse
(faster executions)

Recycle
(faster reexecutions)
Green STM

- **Reduce** (overheads)
- **Reuse** (faster executions)
- **Recycle** (faster reexecutions)
Memoization

also known as function caching
Memoization

- Transparent
Memoization

- Transparent

- Benefits from reexecutions
Memoization

- Transparent
- Benefits from reexecutions
- Free
Memoization

- Transparent
- Benefits from reexecutions
- Free (almost)
Automatic Transaction-Oriented Memoization
STM Bench7: Read-dominated
STM
STM + Memo = Green STM
STM + Memo = Green STM

@Atomic  @Atomic
STM + Memo = Green STM

@Atomic  @Atomic

[Diagram showing the concept of caching and atomic operations]
Per-transaction memo-cache

<table>
<thead>
<tr>
<th>Key</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method(_x)</td>
<td>Arg(_1), \ldots, Arg(_n)</td>
</tr>
<tr>
<td>Method(_y)</td>
<td>Arg(_1), \ldots, Arg(_n)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Per-transaction memo-cache

<table>
<thead>
<tr>
<th>Key</th>
<th>Read-set</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method$_x$ $Arg_1,...,Arg_n$</td>
<td><img src="image" alt="Read-set" /></td>
<td><img src="image" alt="Result" /></td>
</tr>
<tr>
<td>Method$_y$ $Arg_1,...,Arg_n$</td>
<td><img src="image" alt="Read-set" /></td>
<td><img src="image" alt="Result" /></td>
</tr>
<tr>
<td>...</td>
<td><img src="image" alt="Read-set" /></td>
<td><img src="image" alt="Result" /></td>
</tr>
</tbody>
</table>
Per-transaction memo-cache

<table>
<thead>
<tr>
<th>Key</th>
<th>Read-set</th>
<th>Result</th>
<th>Write-set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method(_x)</td>
<td>Arg(_1), ..., Arg(_n)</td>
<td>![Red Hexagon]</td>
<td>![Blue Hexagon]</td>
</tr>
<tr>
<td>Method(_y)</td>
<td>Arg(_1), ..., Arg(_n)</td>
<td>![Red Hexagon]</td>
<td>![Black, Grey, Orange, Red Hexagons]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>![Red Hexagon]</td>
<td>![Black, Grey, Orange, Red Hexagons]</td>
</tr>
</tbody>
</table>

Method\(_x\) and Method\(_y\) are methods with arguments Arg\(_1\), ..., Arg\(_n\). The read-sets and write-sets are shown for each method, with the result indicated as a red hexagon.
Green STM implementation

- **Reduce** (overheads)
- **Reuse** (faster executions)
- **Recycle** (faster reexecutions)
Green STM implementation

Per-transaction memo-cache

Reduce
(overheads)

Reuse
(faster executions)
Green STM implementation

Per-transaction memo-cache

Reduce (overheads)

Central memo-cache
Green STM implementation

Per-transaction memo-cache

Sibling thread

Central memo-cache
Green STM implementation

**Per-transaction memo-cache**

Sibling thread

Central memo-cache
Experimental setup

- STMBench7 benchmark
- 4xAMD Opteron 6168 (48 cores)
- 1 → 48 threads
- Total 1200 ops
**STMBench7: Write-dominated + Long-traversals**

![Graph showing speedup for different thread counts for JVSTM and Memo](chart.png)

- **X-axis**: Number of threads
- **Y-axis**: Speedup (1 thread)
- **Legend**:
  - Red squares: JVSTM
  - Green diamonds: Memo

The graph illustrates the speedup for JVSTM and Memo across different thread counts, highlighting the performance characteristics under write-dominated + long-traversals scenarios.
Conclusions

- STMs collect lots of information
- Accelerate similar executions
- Memoization
Thank you,