Can Software Transactional Memory make Parallel Programs Simple and Safe?

Ketil Malde

Institute of Marine Research

February 11, 2013
Outline

Parallelism

Concurrency Challenges

Transactional Memory

Genome Scaffolding

Results

Conclusions
Outline

Parallelism

Concurrency Challenges

Transactional Memory

Genome Scaffolding

Results

Conclusions
Increasing number of cores

CPU cores per chip by model family

- Magny-Cours
- Beckton
- Istanbul
- Dunnington
- Barcelona
- Kentsfield
- K8
- Conroe

2013: Octo-core phones?
Outline

Parallelism

Concurrency Challenges

Transactional Memory

Genome Scaffolding

Results

Conclusions
Bank account example

`deposit(acct, val) =
   x <- readAcct(acct)
   y = x+val
   writeAcct(acct, y)`
Bank account example

A: $100

\[
\text{deposit}(A, 10) = \\
x \leftarrow \text{readAcct}(A) \\
y = x + 10 \\
\text{writeAcct}(A, y)
\]
Bank account example

\[
A: \quad \$100
\]

\[
\text{deposit}( A, 10 ) = \\
x \leftarrow \text{readAcct}( A ) \\
y = x + 10 \\
\text{writeAcct}( A, y ) \\
\]

\[x = 100\]
Bank account example

\[
\text{deposit}( \text{A} \ , 10 ) = \\
x \leftarrow \text{readAcct}( \text{A} ) \\
y = x + 10 \\
\text{writeAcct}( \text{A} \ , y)
\]

\[
x=100 \quad y=110
\]
Banc account example

\[ \text{deposit}( A , 10 ) = \]
\[ x \leftarrow \text{readAcct}( A ) \]
\[ y = x + 10 \]
\[ \text{writeAcct}( A , y ) \]

\[ x = 100 \quad y = 110 \]
Race condition!

A: $???$

\[
\text{deposit}(A, 10) = \begin{align*}
x & \leftarrow \text{readAcct}(A) \\
y & = x + 10 \\
\text{writeAcct}(A, y)
\end{align*}
\]

\[
\begin{align*}
x & = 100 \\
y & = 110
\end{align*}
\]

\[
\text{deposit}(A, 20) = \begin{align*}
x & \leftarrow \text{readAcct}(A) \\
y & = x + 20 \\
\text{writeAcct}(A, y)
\end{align*}
\]

\[
\begin{align*}
x & = 100 \\
y & = 120
\end{align*}
\]
Solution: Locking

\[
\text{deposit}( A, 10 ) = \\
\text{take\_lock}(A) \\
x <- \text{readAcct}(A) \\
y = x + 10 \\
\text{writeAcct}(A, y) \\
\text{release\_lock}(A)
\]

\[
\text{deposit}( A, 20 ) = \\
\text{take\_lock}(A) \\
x <- \text{readAcct}(A) \\
y = x + 20 \\
\text{writeAcct}(A, y) \\
\text{release\_lock}(A)
\]
Solution: Locking?

It is difficult to get right.

- Need to protect all shared data
- Deadlocks
- Livelocks
- *Locks prevent composability*
Composability

\[
\text{transfer}(\text{src}, \text{dst}, \text{amount}) = \\
\text{withdraw}(\text{src}, \text{amount}) \\
\text{deposit}(\text{dst}, \text{amount})
\]
Composability

\[
\text{transfer}(\text{src}, \text{dst}, \text{amount}) = \\
\text{withdraw}(\text{src}, \text{amount}) \\
\text{deposit}(\text{dst}, \text{amount}) \quad \text{Inconsistent state!}
\]
Composability

\[
\text{transfer}(\text{src}, \text{dst}, \text{amount}) = \\
\text{take\_lock}(\text{src}) \\
\text{take\_lock}(\text{dst}) \\
\text{withdraw}(\text{src}, \text{amount}) \\
\text{deposit}(\text{dst}, \text{amount}) \\
\text{release\_lock}(\text{dst}) \\
\text{release\_lock}(\text{src})
\]
Outline

Parallelism
Concurrency Challenges
Transactional Memory
Genome Scaffolding
Results
Conclusions
Software Transactional Memory

\[
\text{deposit}(A, 10) =
\begin{align*}
&x \leftarrow \text{readAcct}(A) \\
y = x + 10 \\
&\text{writeAcct}(A, y)
\end{align*}
\]

\[
\text{deposit}(A, 20) =
\begin{align*}
&x \leftarrow \text{readAcct}(A) \\
y = x + 20 \\
&\text{writeAcct}(A, y)
\end{align*}
\]
Software Transactional Memory

\[
\text{deposit}(A, 10) = \\
\text{BEGIN} \\
x \leftarrow \text{readAcct}(A) \\
y = x + 10 \\
\text{writeAcct}(A, y) \\
\text{COMMIT}
\]

\[
\text{deposit}(A, 20) = \\
\text{BEGIN} \\
x \leftarrow \text{readAcct}(A) \\
y = x + 20 \\
\text{writeAcct}(A, y) \\
\text{COMMIT}
\]
Software Transactional Memory

$\text{deposit}(\ A, 10\ ) =$
\begin{align*}
&\text{BEGIN} \\
&x \leftarrow \text{readAcct}(\ A\ ) \\
y = x + 10 \\
&\text{writeAcct}(\ A, y) \\
&\text{COMMIT} \\
x = 100 \quad y = 110
\end{align*}

$\text{deposit}(\ A, 20\ ) =$
\begin{align*}
&\text{BEGIN} \\
&x \leftarrow \text{readAcct}(\ A\ ) \\
y = x + 20 \\
&\text{writeAcct}(\ A, y) \\
&\text{COMMIT} \\
x = 100 \quad y = 120
\end{align*}
Software Transactional Memory

deposit(A, 10) =
BEGIN
x <- readAcct(A)
y = x + 10
writeAcct(A, y)
COMMIT

x=100 y=110

deposit(A, 20) =
BEGIN
x <- readAcct(A)
y = x + 20
writeAcct(A, y)
COMMIT

x=100 y=120

A: $100
$110
Software Transactional Memory

deposit( A, 10 ) =
BEGIN
x <- readAcct( A )
y = x + 10
writeAcct( A, y )
COMMIT
x=100 y=110

retry

deposit( A, 20 ) =
BEGIN
x <- readAcct( A )
y = x + 20
writeAcct( A, y )
COMMIT
x=100 y=120
STM Performance Impact

- Logging data access
- Retrying transaction
Outline

Parallelism

Concurrency Challenges

Transactional Memory

Genome Scaffolding

Results

Conclusions
Scaffolding

Given a set of *contigs*
And a set of *links*
Order and orient the contigs, satisfying as many links as possible.

A graph problem:

- Contigs are nodes
- Link information make edges
Scaffolding

Given a set of *contigs*
And a set of *links*
Order and orient the contigs,
satisfying as many links as possible.

A graph problem:

- Contigs are nodes
- Link information make edges
The scaffolding graph
Identify edge

![Diagram showing network with nodes A and B connected by an edge.](attachment:image.png)
Update graph
▶ Changes are local
▶ ..but unpredictable

→ Ideal for transactions?
Outline

Parallelism
Concurrency Challenges
Transaction Memory
Genome Scaffolding
Results
Conclusions
Simple Implementation

Join contigs A and B iff:

B is the contig with most links from A
and
A is the contig with most links from B
Transactional Overhead

![Transactional Overhead Graph](image)

- Array
- STM-S
- STM-1

Scaffolding time (seconds)

0 50 100 150 200 250 300 350 400

Array STM-S STM-1
Speedup

![Graph showing speedup vs threads]

- Y-axis: Scaffolding speedup
- X-axis: Threads

1. Speedup 1
2. Speedup 2
3. Speedup 4
4. Speedup 8
5. Speedup 16

---

Software
Transactional Memory
Ketil Malde

Parallelism
Concurrency Challenges
Transactional Memory
Genome Scaffolding
Results
Conclusions
Caveats

Performance will be hurt by retried transactions.

Affected by:

- Number of threads/size of graph
- Size of subgraph affected by each transaction
Outline

Parallelism
Concurrency Challenges
Transactional Memory
Genome Scaffolding
Results
Conclusions
Conclusions

CPU time is precious — don’t waste it on single-thread applications!

Programmer time is even more precious — don’t waste it on primitive concurrency primitives!
Conclusions

**CPU time is precious** — don’t waste it on single-thread applications!

**Programmer time is even more precious** — don’t waste it on primitive concurrency primitives!
Conclusions

**CPU time is precious** — don’t waste it on single-thread applications!

**Programmer time is even more precious** — don’t waste it on primitive concurrency primitives!
The End

Thanks