Reducing Memory Buffering Overhead in Software Thread-Level Speculation

Zhen Cao  
zhen.cao@mcgill.ca

Clark Verbrugge  
clump@cs.mcgill.ca

McGill
Thread Level Speculation (TLS)

- Basis for automatic parallelization
  - Start from sequential program
  - Optimistically parallelize future execution
    - Safety guaranteed
- Traditionally a hardware technique
- **Software** implementation
  - Large parallel granularity, no special hardware
  - Overhead concerns
Thread Level Speculation (TLS)

- Speculative threads execute in the future
Thread Level Speculation (TLS)

- Past should affect the future

\[
\begin{align*}
    a &= \ldots \\
    x &= \ldots \\
    \ldots &= x
\end{align*}
\]
Thread Level Speculation (TLS)

- Future should not affect the past
Thread Level Speculation (TLS)

- Future should not affect the past
Contents

• Version management
  – Lazy Buffering
    • Optimized design
  – Eager Buffering
    • Optimized design
• Integrating lazy & eager
• (Thread coverage)
• Experiments
Version Management

• Key property for safety
  – Need to avoid RAW, WAR, WAW errors
  – Isolate and/or restore

• Two main flavours: Lazy & Eager
Lazy Buffering

- Lazy version management
  - Non-speculative thread accesses memory
  - Speculative threads buffered
    - Reads for validation
    - Writes for isolation
Lazy Buffering

Non-speculative

Speculative 1

Speculative 2

Memory

R-Buffer

W-Buffer

R-Buffer

W-Buffer
Lazy Buffering

Non-speculative

Speculative 1

Speculative 2

R-Buffer
W-Buffer

read x

save x

validate

commit

x written?

Memory

Read x
Lazy Improvements

- Problem: idleness, due to validation/commit
  - Bigger granularity = large buffers

- Parallelize V/C?
  - Processors idle anyway...

- Coarse and fine-grain parallelization
  - But need to structure buffers to help
Per-Thread Page Tables

- Different pages committed in parallel
  - Partitioned (pages), guaranteed separate
- V/C can be vectorized on a page
  - SIMD acceleration
- Supports mixed data types
- Extra cost
  - More space, hashing
Contents

● Version management
  - Lazy Buffering
    • Optimized design
  - Eager Buffering
    • Optimized design
● Integrating lazy & eager
● (Thread coverage)
● Experiments
Eager Buffering

- Used in SpLIB, MiniTLS, ...
- All speculative, access main memory directly
  - Keep shadow buffer for rollback
  - Track versions for proper restore
Eager Scheme
Eager Scheme
Eager Scheme

Check x
x written by another?

Speculative 0
R/W x

Speculative 1

Speculative 2

Rollback thread

Memory

Load/Store Vector

Buffer

Buffer

Buffer (restore)
Eager Scheme

- Faster (no) commit, but slower rollback
- WAR and WAW dependencies require rollback
Eager Improvements

- Problem: multiple buffers, lots of versions
- Only keep one version?
  - Need to consider thread order
Shared Address-Owner Buffering

- Single shadow buffer for all threads
  - At most one buffered copy of each variable
- Improved space
  - $O(D) \text{ vs } O(D+W)$
    - $D$ data accesses, $W$ number of writes
- Finer or coarser granularity
  - Treat vars as WORD bytes
Shared Address-Owner Buffering

Owner Vector

Non-speculative

Speculative 1

Speculative 2

Buffer

Memory
Shared Address-Owner Buffering

• Each WORD has
  – Owner for dependency tracking
    • One bit for whether written or not
  – Shadow copy for rollback

• Owners ordered
  – In-order forking
    • Global counter ok
  – NS lowest

<table>
<thead>
<tr>
<th>Condition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>t &lt; t₀</td>
<td>?</td>
</tr>
<tr>
<td>t₀ ≤ t ≤ t_max</td>
<td>0</td>
</tr>
<tr>
<td>t₀ ≤ t ≤ t_max</td>
<td>1</td>
</tr>
<tr>
<td>111 ... 111</td>
<td>0</td>
</tr>
</tbody>
</table>
Contents

- Version management
  - Lazy Buffering
    - Optimized design
  - Eager Buffering
    - Optimized design
- Integrating lazy & eager
- (Thread coverage)
- Experiments
Buffering Integration

- Builds on a **readonly** optimization

- Lots of speculative regions have readonly vars
  - Fills up buffer space

- Static is rare; dynamic is not easy to identify
  - Most approaches manual, profile-based

- Just need transitively readonly
  - Within a speculative region
ReadOnly

- Page based; work with larger chunks of mem
  - Heap alloc sites as single vars
- Degrees of readonly-ness
  - Readonly (default on entry)
  - Independent (threads R/W different parts)
  - Dependent (conflicting)
- Rollback reduces degrees
Readonly pages – no buffering required
Independent pages – use **Eager Buffering**
Dependent pages – use **Lazy Buffering**
Contents

● Version management
  – Lazy Buffering
    • Optimized design
  – Eager Buffering
    • Optimized design
● Integrating lazy & eager
● (Thread coverage)
● Experiments
# Experiments

<table>
<thead>
<tr>
<th></th>
<th>Language</th>
<th>Benchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td>lavaMD</td>
<td>C</td>
<td>Rodinia</td>
</tr>
<tr>
<td>streamcluster</td>
<td>C++</td>
<td>Rodinia</td>
</tr>
<tr>
<td>kmeans</td>
<td>C</td>
<td>Rodinia</td>
</tr>
<tr>
<td>srad</td>
<td>C</td>
<td>Rodinia</td>
</tr>
<tr>
<td>cfd</td>
<td>C++</td>
<td>Rodinia</td>
</tr>
<tr>
<td>sparsematmul</td>
<td>C</td>
<td>SciMark</td>
</tr>
<tr>
<td>smallpt</td>
<td>C++</td>
<td>smallpt</td>
</tr>
<tr>
<td>bwaves</td>
<td>Fortran</td>
<td>SPEC CPU2006</td>
</tr>
<tr>
<td>fft</td>
<td>C</td>
<td>MUTLS</td>
</tr>
</tbody>
</table>

AMD Opteron 6274 (4x16 cores, 64GB memory)

Parallel V/C cores: 0-7
Experiments

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mutls</td>
<td>Plain lazy</td>
</tr>
<tr>
<td>simd</td>
<td>Lazy with SIMD V/C</td>
</tr>
<tr>
<td>simd-pvc</td>
<td>Add parallel V/C</td>
</tr>
<tr>
<td>simd-ro</td>
<td>Lazy with SIMD and Readonly</td>
</tr>
<tr>
<td>simd-pvc-ro</td>
<td>Add parallel V/C</td>
</tr>
</tbody>
</table>
Experiments

- **eager-nolazy**: Eager only
- **simd-eager**: Eager, fallback to lazy SIMD V/C for dependent
- **simd-eager-ro**: Readonly as well
- **simd-pvc-ro**: As previous slide
Experiments

Nb: Scaled to OpenMP (manual)
Conclusions & Future Work

- **Software TLS**
  - Feasible, but a significant engineering effort
    - Different benchmarks need different optimizations

- **Adaptivity**
  - Effective, better tuning would help

- **Hardware (transactional) help?**
  - Small buffers!
  - Hard to enforce sequential commit
Thank You

Questions

Zhen Cao
zhen.cao@mail.mcgill.ca
http://www.sable.mcgill.ca/~zcao7/mutls/

Clark Verbrugge
clump@cs.mcgill.ca
Adaptive Selection

- Buffering integration defaults to eager
- But costly to non-speculative thread
  - Small V/C, lazy is faster
- Adaptive heuristics, based on profiling
  - Start with lazy (more robust to rollback)
Adaptive Selection

- Compute at commit:
  - m: # memory accesses
  - $T_w$: work time
  - $T_v$: validation/commit time
  - C: overhead on var access by speculative thread (20)
  - K: delay on thread for eager case (8)

- Estimate (future) lazy:eager time over L iters:
  - Lazy speedup $S = 1+(n-1)*(T_w-C*m)/T_w$
  - Lazy = $L*T_w/S+L*T_v$
  - Eager = $L*T_w*K/n$
Contents

● Version management
  – Lazy Buffering
    • Optimized design
  – Eager Buffering
    • Optimized design
● Integrating lazy & eager
● Thread coverage
● Experiments
Thread Coverage

- Usually \#threads \leq \#CPUs
- Speculative threads are slower
  - Waiting to join wastes resources
- Generate more speculative tasks than CPUs
  - For in-order forking, just 1 is sufficient
  - (Generally at most one pending task per thread)
Thread Coverage

CPU0
T0
(NS)

CPU1
T1

CPU2
T2

CPU3
T3

{idle

Commit
Thread Coverage

CPU0
T0
(NS)

CPU1
T1

CPU2
T2

CPU3
T3

CPU0
T4

Commit

---

T0
T1
T2
T3
T4

---

NS

---
Thread Stopping

- Rollback in integration can reduce coverage
  - Rollback all speculative threads
  - Speculation continued when next fork point reached
    - No speculation until then
- Stop direct child(ren)
  - Rollback indirect children
- Reset buffering
  - Restart stopped child(ren)
Thread Stopping

Input:
- CPU0
  - T0
    - (NS)

Output:
- CPU1
  - T1
- CPU2
  - T2
- CPU3
  - T3

Rollback

Next fork pt
Thread Stopping

CPU0
T0 (NS) stop restart

CPU1
T1

CPU2
T2

CPU3
T3

Rollback
Experiments

Nb: Scaled to simd-eager-ro

- **heuristics**: Adaptive buffering heuristics (on simd-eager-ro)
- **lazy-ro**: Compare heuristics to lazy version
- **nostopping**: Disable nostopping (on simd-eager-ro)