Reducing Memory Buffering Overhead in Software Thread-Level Speculation

Zhen Cao and Clark Verbrugge
McGill University
Introduction

- Traditional parallel programming paradigms
  - OpenMP, pthread and OpenCL
  - difficult to debug and optimize
- Speculative parallelism
  - promising for automatic parallelization
  - guarantee correctness
  - apply to irregular applications
Thread-Level Speculation (TLS)

- launched at fork point, execute from join point
- Buffering reads and writes
Software Thread-Level Speculation

- Advantage
  - apply to existing, commodity hardware
  - greater and more flexible resource limits
  - larger parallelism granularity
- Disadvantage
  - significant buffering overhead
  - long validation/commit (V/C) time
Software TLS buffering

- Lazy version management
  - SableSpMT, SpLSC, Lector and MUTLS
- Eager version management (in-place update)
  - SpLIP and MiniTLS
Lazy version management

- serial commit phase may be a scalability bottleneck
Eager version management

- no serial commit phase, higher scalability
- expensive rollback overhead
- rollback for all RAW, WAR and WAW dependencies
Contribution (1)

- Improve lazy version management
  - per-thread page-table memory buffering
  - parallelize the TLS system itself
    - Parallelized V/C
    - SIMD acceleration
- Improve eager version management
  - shared address-owner memory buffering
  - enable speculation at any-granularity
    - constant space overhead
    - rare buffering conflicts, no buffering overflow
Contribution (2)

- Integrate lazy and eager version management
  - identify independent/dependent variables
    - low overhead and fully dynamic
  - apply eager/lazy version management buffering for independent/dependent variables

- Optimize readonly data
  - identify readonly variables
  - not buffer/validate readonly variables
Page-Table Memory Buffering

- fully support mixed data types
- efficient K-way hash mapping
- enable both coarse-grain and fine-grain parallelism

<table>
<thead>
<tr>
<th>memory address</th>
<th>high bits</th>
<th>hash bits</th>
<th>buffer bits</th>
<th>page bits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>N</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>per-thread memory buffer</th>
<th>mark array</th>
<th>buffer array</th>
<th>page table</th>
<th>accessed pages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N+2</td>
<td>N+2</td>
<td>N+2</td>
<td>N+2</td>
</tr>
</tbody>
</table>

N+2 represents the number of bits required for the respective component.
Parallelized V/C

- validate/commit different buffer pages in parallel
  - improve scalability
  - enabled only if pages are more than a threshold
- CPU resource
  - Current implementation uses dedicated CPUs
  - not important problem for many-core machines
  - could use computation cores as V/C cores
    - set higher priority to V/C threads to reduce critical path delay
SIMD Acceleration

- validation/commit within a page can be vectorized by SIMD instructions such as SSE2, SSE4 and AVX2

| mark array page | 16/32B | 16/32B | ... | 16/32B |
| buffer array page | 16/32B | 16/32B | ... | 16/32B |

4KB
Shared Address-Owner Buffering

- eager version management
- shared by all threads
- at most one history version is buffered
Previous eager version management buffering

- buffer multiple versions of the same written data
- more complex dependency tracking
 Readonly-Page Optimization

- each entry of page type table is marked RO/not-RO
- map the address of a read to the entry in page type table
- not buffering if the page is marked RO

![Diagram showing memory address, high bits, global buffer bits, and page bits with N and 12 values.](image-url)
Readonly-Page Optimization

• maintain the pages of each variable
• reset all pages to RO before entering a speculative region
• if a thread writes a page marked RO
  • all speculative threads are rolled back
  • mark *all* pages of the written variable as not-RO
  • can quickly find all RO and not-RO pages after entering the speculative region
• pointer-based data structures
  • ineffective if different addresses as different variables
  • each memory allocation instruction as a single variable
Buffering Integration

- mark variable page types
  - readonly
  - independent (readonly or threads access disjoint memory)
  - dependent (different threads write the same address)
Experiments

- implemented in the MUTLS software TLS system and llvm-3.2
- 4x16-core AMD Opteron 6274 machine with 64 GB memory
- 48 cores for TLS and 8 cores for parallelized V/C

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mandelbrot</td>
<td>stand-alone</td>
</tr>
<tr>
<td>MolecularDynamics</td>
<td>stand-alone</td>
</tr>
<tr>
<td>RayTracing</td>
<td>stand-alone</td>
</tr>
<tr>
<td>SparseMatMul</td>
<td>SciMark</td>
</tr>
<tr>
<td>lavaMD</td>
<td>Rodinia</td>
</tr>
<tr>
<td>streamcluster</td>
<td>Rodinia</td>
</tr>
<tr>
<td>bwaves</td>
<td>SPEC CPU 2006</td>
</tr>
<tr>
<td>lbm</td>
<td>SPEC CPU 2006</td>
</tr>
</tbody>
</table>
Speedup (compute intensive)

- mutls
- simd
- simd-omp
- simd-omp-ro
- simd-omp-eager
- simd-omp-ro-eager
- theoretically optimal

Mandelbrot

Molecular Dynamics
Speedup (memory-bus intensive)

- mutls
- simd
- simd-omp
- simd-omp-ro
- simd-omp-eager
- simd-omp-ro-eager
- theoretically optimal

Graphs showing speedup for streamcluster and lbm with different configurations.
Speedup (cache intensive)

- **mutls**
- **simd**
- **simd-omp**
- **simd-omp-ro**
- **simd-omp-eager**
- **simd-omp-ro-eager**
- **theoretically optimal**

- **SparseMatMul**
- **bwaves**

Graphs showing speedup for mutls, simd, simd-omp, simd-omp-ro, simd-omp-eager, simd-omp-ro-eager, and theoretically optimal for SparseMatMul and bwaves.
Speedup (cache intensive)

- mutls
- simd
- simd-omp
- simd-omp-ro
- simd-omp-eager
- simd-omp-ro-eager
- theoretically optimal

RayTracing

lavaMD
Critical Path V/C Time Ratio (compute intensive)

- mutls
- simd
- simd-omp
- simd-omp-ro
- simd-omp-eager
- simd-omp-ro-eager
- theoretically optimal

Graphs showing the ratio for Mandelbrot and MolecularDynamics.
Critical Path V/C Time Ratio (memory-bus Intensive)

- mutls
- simd
- simd-omp
- simd-omp-ro
- simd-omp-eager
- simd-omp-ro-eager
- theoretically optimal
Critical Path V/C Time Ratio (cache Intensive)

- mutls
- simd
- simd-omp-eager
- simd-omp-ro-eager
- theoretically optimal

For SparseMatMul:
- 0% 5% 10% 15% 20% 25%
- 2 4 8 16 32 48

For bwaves:
- 0% 10% 20% 30% 40% 50% 60% 70% 80%
- 2 4 8 16 32 48
Critical Path V/C Time Ratio (cache Intensive)

- **mutls**
- **simd**
- **simd-omp**
- **simd-omp-ro**
- **simd-omp-eager**
- **simd-omp-ro-eager**
- **theoretically optimal**

**RayTracing**

- 0.0%
- 0.5%
- 1.0%
- 1.5%
- 2.0%
- 2.5%
- 3.0%

**lavaMD**

- 0.0%
- 0.2%
- 0.4%
- 0.6%
- 0.8%
- 1.0%
- 1.2%
- 1.4%
- 1.6%
- 1.8%
Future Work

- buffering integration mechanism
  - maintaining different buffering integration data for different fork points
  - reduce rollback time for programs iterating different speculative regions such as bwaves and lbm
- more accurately identify independent and readonly variables
  - Profilers and/or feedback logs
- hardware acceleration
  - GPU/APU
  - Hardware Transactional Memory (HTM)
Thank You

- Q & A