Speculative Synchronization

José F. Martínez
Department of Computer Science
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu/martinez
Problem 1: Conservative Parallelization

- No parallelization unless 100% safe:
  - Hard-to-analyze access patterns
    - Subscripted array subscripts
    - Pointer accesses
  - Corner cases in mostly parallel codes

```
for(i=0; i<n; i++) {
  ... = A[B[i]] ...
  ...
  A[C[i]] = ...
}
```
Problem 2: Conservative Synchronization

- Synchronization in parallel codes conservative:
  - Hard-to-analyze access patterns
  - Corner cases in mostly race-free codes
  - Aggressive sync not affordable
    - Too time consuming
    - Too complicated

```c
parallel {
    for(i=0;i<n;i++)
        b[pid][i]=f(a[pid-1][i],a[pid+1][i]);
    barrier();
    for(i=0;i<n;i++)
        a[pid][i]=f(b[pid-1][i],b[pid+1][i]);
}
```
Technology: Speculative Parallelization*

- Execute speculatively in parallel hard-to-analyze codes
  - Assume no dependences and execute in parallel
  - Track memory accesses, detect violations
  - Squash and restart offending threads
  - Keep safe thread (earliest) at all times

```c
for(i=0;i<n;i++){
    ... = A[B[i]] ... 
    ...
    A[C[i]] = ...
}
```

* a.k.a. Thread-Level Speculation (TLS)
Outline

- Speculative Synchronization
  - Introduction
  - Implementation
  - Related Work
    - Herlihy & Moss (DEC/UMass)
    - Stone et al (IBM)
    - Rajwar & Goodman (Wisconsin)
Synchronization often Conservative

- Barriers, locks, flags widely used
  - Parallelizing compilers: mostly full barriers
  - Programmers: M4 macros, OpenMP directives
- Often placed conservatively
  - Hard-to-analyze memory access patterns
  - Corner cases in mostly race-free codes
  - Aggressive sync not affordable
    - Too time-consuming
    - Too complicated
Proposal: Speculative Synchronization

- **Idea:** off-load synchronizing ops from processor
- **Apply TLS to speculate past active barriers, locks, flags**
  - Detect conflicts, roll back offending threads
  - Use caches to store speculative state
- **Maintain 1 or more safe threads → forward progress**
  - Lock: owner
  - Flag: producer
  - Barrier: lagging threads
- **Speculative threads execute past sync points**
Important Features

- Concurrency possible even if conflicts
  - All in-order safe-to-spec conflicts tolerated
- No order among spec threads → simpler HW
  - No MDT
- No programming effort
  - Retargetted macros/ directives
- Can coexist with conventional sync at run-time
Speculative Barrier
Speculative Barrier

BARRIER

Safe
Speculative
Speculative Barrier

A

BARRIER

C

Safe
Speculative
Speculative Barrier

SAFE

SPECULATIVE
Speculative Lock

- ACQUIRE
- RELEASE

Safe
Speculative
Speculative Lock

<table>
<thead>
<tr>
<th>ACQUIRE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safe</td>
</tr>
<tr>
<td>Speculative</td>
</tr>
</tbody>
</table>

| RELEASE |
Speculative Lock

\[\text{ACQUIRE}\]

\[\text{RELEASE}\]

- Safe
- Speculative
Speculative Lock

ACQUIRE

SAFE

Speculative

RELEASE
Speculative Lock

ACQUIRE

RELEASE

Safe
Speculative
Speculative Synchronization Unit

- Extends cache controller
- Simple hardware:
  - 1 spec bit/line
  - Some control logic
Speculative Lock Request

- **Processor side:**
  - Program SSU for speculative lock
  - Checkpoint register file
- **SSU side:**
  - Initiate T&T&S loop on lock variable
- **Use caches as speculative buffer (like TLS)**
  - Set Speculative bit in lines accessed speculatively
Lock Acquire

- SSU acquires lock (T&S successful)
  - Clears all Speculative bits → one-shot commit
  - Becomes idle
- Release (store) later by processor
Release while Speculative

- Processor issues release, SSU still active
  - SSU intercepts release (store) by processor
  - SSU toggles Release bit – “already done”
- When lock becomes available later
  - SSU:
    - **Does not** perform T&S
    - Clears all Speculative bits → one-shot commit
Memory Access Conflict

- External coherence actions
  - Request to safe line: service normally
  - Request to spec line: squash thread
    - Invalidate lines marked Speculative+Dirty → one-shot squash
    - Roll back & restart at sync point
- Safe threads never squashed → forward progress
- All safe-to-spec in-order dependences tolerated
Speculative Flags and Barriers

- Flag spin: Test only – no T&S
  - Handle like “Release while Speculative” case

- Barrier: leverage flag spin support
  - Update thread counter
  - If not last one, spin on flag speculatively
## Retargetted M4 Macros

<table>
<thead>
<tr>
<th>Conventional Macros (Existing)</th>
<th>Speculative Macros (Proposed)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LOCK</strong>(<code>{</code>&lt;br&gt;lock($1);<code>}</code>)</td>
<td><strong>SS_LOCK</strong>(<code>{</code>&lt;br&gt;if(!issu_lock(&amp;$1))&lt;br&gt;LOCK($1)<code>}</code>)</td>
</tr>
<tr>
<td><strong>UNLOCK</strong>(<code>{</code>&lt;br&gt;unlock($1);<code>}</code>)</td>
<td><strong>SS_UNLOCK</strong>(<code>{</code>&lt;br&gt;UNLOCK($1)<code>}</code>)</td>
</tr>
<tr>
<td><strong>WAIT</strong>(<code>{</code>&lt;br&gt;while($1 != $2);<code>}</code>)</td>
<td><strong>SS_WAIT</strong>(<code>{</code>&lt;br&gt;if(!issu_wait(&amp;$1,$2))&lt;br&gt;WAIT($1,$2)<code>}</code>)</td>
</tr>
<tr>
<td><strong>BARRIER</strong>(<code>{</code>&lt;br&gt;$1.1f[PID] = !$1.1f[PID];&lt;br&gt;LOCK($1.lock)&lt;br&gt;$1.c++;&lt;br&gt;if($1.c == NUMPROC) <code>{&lt;br&gt;$1.f = $1.1f[PID];&lt;br&gt;UNLOCK($1.lock)&lt;br&gt;</code>}&lt;br&gt;else <code>{&lt;br&gt;UNLOCK($1.lock)&lt;br&gt;WAIT($1.1f,$1.1f[PID])&lt;br&gt;</code>}<code>&lt;br&gt;}&lt;br}</code></td>
<td><strong>SS_BARRIER</strong>(<code>{</code>&lt;br&gt;$1.1f[PID] = !$1.1f[PID];&lt;br&gt;<strong>SS_SYNC</strong>&lt;br&gt;LOCK($1.lock)&lt;br&gt;$1.c++;&lt;br&gt;if($1.c == NUMPROC) <code>{&lt;br&gt;$1.f = $1.1f[PID];&lt;br&gt;UNLOCK($1.lock)&lt;br&gt;</code>}&lt;br&gt;else <code>{&lt;br&gt;UNLOCK($1.lock)&lt;br&gt;**SS_WAIT**($1.1f,$1.1f[PID])&lt;br&gt;</code>}<code>&lt;br&gt;</code>}`</td>
</tr>
</tbody>
</table>
Related Work

- Herlihy & Moss (1993), Rajwar & Goodman (2001)
  - Lock-free → may save lock overhead if successful
  - Only for critical sections → no barrier, flag support
  - Execute all threads speculatively → slowdown possible

  - Lock-free, only for critical sections
  - Spec buffer limited to reservation registers
  - Oklahoma update costly in space and time
  - False sharing requires exponential back-off
Experimental Setup

- CC-NUMA, 4-issue dynamic superscalar processors
  - Flat configuration
  - Node: single processor
    - SSU, private L1+L2
  - System: 16-64 procs
Applications

- Mix of parallel codes
- Parallelization:
  - Compiler [16p] (applu)
  - Annotated [16p] (mst, bisort)
  - Hand [64p] (ocean, 2×barnes)
Speculative Sync: Summary of Results

- Promising results for such simple hardware
  - Average sync time reduction: 40%
  - Execution time improves up to 15%, avg 7.5%
Sync Time Reduction

Large reduction: 40%  
Room for improvement
Execution Time Reduction

Across-the-board reduction
What We Learned

- Speculative Synchronization very effective
  - Promising speedups
  - TLS’s forward progress guarantee
    - Critical path not affected
    - Speculative buffer overflow simply stalls
  - Simple hardware
  - No programming effort
  - Room for improvement
    - WAR, WAW dependences
    - False sharing
Overall Conclusions

- TLS very promising emerging technology
- Contributions to TLS
  - Build hierarchical scalable TLS
    • Use “commodity” spec CMPs as building blocks
  - Improve conservatively sync’d parallel codes
- TLS’s safe-thread mechanism effective
Optimistic Concurrency Control

- Target critical sections
- Concurrent access assuming no dependences
- Write specialized code
- All threads execute optimistically
- Check for conflicts at the end
- Discard changes and re-execute if conflicts found
  - Serialization, chance of slow-down if conflicts repetitive

- Example: Transactional Memory
Support for Multiple Locks

Speculative thread T1 hits an acquire for loc1

Loc1 is free:
- Acquire loc1 speculatively
  - T1 becomes non-speculative
  - T2 attempts to acquire loc1
  - T1 is squashed

Loc1 is not free:
- Spin on local copy of loc1
  - T1 is still speculative
  - T1 is not speculative
  - T1 competes for loc1

Owner releases

* Unless T1 is already out of loc1’s critical section
Projects in TLS

- Multiscalar/SVC (Wisconsin)
- Superthreaded (Minnesota)
- Hydra (Stanford)
- Speculative MP (UPC)
- TLDS (CMU)
- LRPD, Zhang, MDT (Illinois)
- DMT (Intel)
- MAJC (Sun)