Thread-Level Speculation on Off-the-Shelf Hardware Transactional Memory

Rei Odaira
Takuya Nakaike
IBM Research – Tokyo
Thread-Level Speculation (TLS) [Franklin et al., ’92] or Speculative Multithreading (SpMT)

- Speculatively parallelize a sequential program into a multithreaded program.

- What is parallelization?
  - To find data-independent tasks from a program.

- Why speculation?
  - Because a compiler cannot detect every data dependence.
Runtime Requirements for TLS

- With TLS:
  - Compiler finds *probably* data-independent tasks.
  - Runtime guarantees data independence among tasks.

- (Minimum) runtime requirements for TLS
  - Data dependence (= conflict) detection among tasks
  - Execution rollback at a conflict
  - Ordered commit of tasks
Hardware Transactional Memory (HTM) Coming into the Market

- HTM supports…
  - Conflict detection among transactions
  - Execution rollback at a conflict

- HTM satisfies 2/3 of the runtime requirements for TLS!
  - Task = transaction

Blue Gene/Q  
zEC12  
POWER8  
4th Generation Core Processor (Haswell)
Our Goal

- How well can TLS improve the performance on real HTM hardware?
  - Used Intel 4th Generation Core Processor (Intel TSX).
  - Manually modified and measured SPEC CPU2006.
Our True Goal

- How *poorly* can TLS improve the performance on real HTM hardware?

- Because proposed TLS systems had advanced hardware support.
  - E.g. ordered transactions, data forwarding, etc.

- Blue Gene/Q is the only real system supporting advanced hardware for TLS.
  - Ordered transactions
Our True Goal

- How *poorly* can TLS improve the performance on real HTM hardware?
- What kind of hardware support should be implemented next in the off-the-shelf HTM?
Transactional Memory

- At programming/compile time
  - Enclose critical sections with transaction begin/end operations.

- At execution time
  - Memory operations within a transaction observed as one step by other threads.
  - Multiple transactions executed in parallel as long as their memory operations do not conflict.

```c
xbegin();
a->count++;xend();

Thread X
xbegin();
a->count++;xend();   
Thread Y
xbegin();
a->count++;xend();

xbegin();
a->count++;xend();

xbegin();

b->count++;xend();
```
HTM

- Instruction set (Intel TSX)
  - XBEGIN: Begin a transaction
  - XEND: End a transaction
  - XABORT, etc.

- Micro-architecture
  - Read and write sets held in CPU caches
  - Conflict detection using CPU cache coherence protocol
    → Conflict detection by cache line granularity
  - Rollback by discarding write set and restoring registers

- Abort reasons:
  - Read set and write set conflict
  - Read set and write set overflow
  - External interruptions, etc.

```
XBEGIN abort_handler
  ...
XEND

abort_handler:
  ...
```
TLS for Loops

- We focus on frequently executed loops.
  - Task = iteration(s) = transaction

- Why not parallelize function calls?
  - Difficult to implement TLS for function calls on HTM. (Refer to the paper for the details.)

Sequential execution

TLS execution w/ 3 threads
TLS on HTM

- Enclose each iteration with XBEGIN and XEND.
- Re-execute iteration in case of abort.
Ordered Transactions

- Must commit in the same order as sequential execution.
  - Because data independence can be guaranteed only after all of the preceding iterations have committed.
Ordered Transactions by Software

- Hardware support by proposed TLS systems
  - Wait until the preceding iterations commit.

- Software implementation by checking commit order
  - Use a global variable to indicate the next iteration to commit.
  - Abort if cannot commit.

Iteration 1

Iteration 2

Can commit?

Can commit?

Iteration 3

Can commit?

Iteration 3 re-execution

Can commit?
Ordered Transactions by Software

- Hardware support by proposed TLS systems
  - Wait until the preceding iterations commit.

- Software implementation by checking commit order
  - Use a global variable to indicate the next iteration to commit.
  - Abort if cannot commit.

Why not spin-wait? Refer to our paper....
Our Goal

- How *poorly* can TLS improve the performance on real HTM hardware?
- What kind of hardware support should be implemented next in the off-the-shelf HTM?
  - Will hardware support for ordered transactions really help?
False Sharing due to Cache-Line Granularity Conflict Detection

double array[];
...
for (int i = ...; i < ...; i++) {
    ...
    array[i] = ...;
    ...
}

Writes by Thread 1
Writes by Thread 2
Writes by Thread 3
array[]

Cache line = 64 bytes on x86

TLS
Transaction Coarsening to Avoid False Sharing

Iteration 1
Iteration 2
... Iteration 8

Iteration 9 Iteration 10 ... Iteration 16

Iteration 17 Iteration 18 ... Iteration 24

Writes by Thread 1
Writes by Thread 2

Writes by Thread 3

array[]
Benchmarks and Methodology

- **SPEC CPU2006**
  - 6 benchmarks showing more than 1.5-fold speedups with 4 threads in a previous TLS study [Packirisamy et al., 2009]
  - 429.mcf, 433.milc, 456.hmmer, 464.h264ref, 470.lbm, and 482.sphinx3

- Manually modified frequently executed loops.
  - Inserted XBEGIN, XEND, and commit order checks.
  - Transformed a target loop into a doubly-nested loop for transaction coarsening

- **Experimental environment**
  - Core i7-4770 processor (4 cores, 2-way SMT)
  - 4-GB memory
  - Linux 2.6.32-431 / GCC 4.9.0
Normalized Throughput Results

- Up to 11% speedups with 2 or 4 threads.
- But mostly degraded the throughput.
Commit order inversion is a dominant abort reason. → Hardware support for ordered transactions will help.
Conflicts were a dominant abort reason in all of the benchmarks except 433.milc.
Abort Statistics (2/2)

Conflicts were a dominant abort reason in all of the benchmarks except 433.milc.
## Reasons for Conflicts and Possible Hardware Support

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Conflict reason</th>
<th>Possible hardware support</th>
</tr>
</thead>
<tbody>
<tr>
<td>429.mcf</td>
<td>RAW dependence</td>
<td>Data forwarding</td>
</tr>
<tr>
<td>433.milc</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>456.hmmer</td>
<td>RAW dependence</td>
<td>Data forwarding</td>
</tr>
<tr>
<td>464.h264ref</td>
<td>WAR dependence</td>
<td>Multi-version cache</td>
</tr>
<tr>
<td></td>
<td>WAW dependence (false sharing by prefetching)</td>
<td>(Fix in prefetcher)</td>
</tr>
<tr>
<td>470.ibm</td>
<td>WAW dependence (false sharing)</td>
<td>Word-level conflict detection</td>
</tr>
<tr>
<td>482.sphinx3</td>
<td>WAW dependence (false sharing by prefetching)</td>
<td>(Fix in prefetcher)</td>
</tr>
</tbody>
</table>
Examples of Read-After-Write Data Dependence

429.mcf

```c
static int size;
static DATA array[N];
func() {
    ...
    for (...) {
        ...
        if (...) {
            size++;
            array[size]->field = ...;
        }
    }
    ...
}
```

456.hmmer

```c
for (k = 1; k <= M; k++) {
    ...
    dc[k] = dc[k-1] + ...;
    ...
}
```

- Hardware support already proposed in TLS literatures.
  - Data forwarding.
Example of Write-After-Read Data Dependence

464.h264ref

for (...) {
    ...
    line = func();
    ...
    = line[0];
    ...
}

static DATA line[N];

DATA *func() {
    ...
    line[0] = ...;
    ...
    return line;
}

- Difficult to analyze by a compiler.
  - WAR dependence across different functions in different source files.

- Multi-version caches needed.
Conflicts Precede Commit Order Inversion

- Commit order matters only when most of the transactions reach the committing points.
- With data dependence, most of the transactions cannot run to the end.
Conflicts due to Prefetching

- Even with transaction coarsening, conflicts still happened.
  - 464.h264ref and 482.sphinx3.
- Prefetched adjacent cache lines caused conflicts.

Writes by Thread 1

| 64 bytes | Conflict | 64 bytes | 64 bytes |

Prefetch

Writes by Thread 2

Prefetch
Conclusion

- How well can TLS improve the performance on real HTM hardware?
  - Up to 11% speedups with 4 threads in SPEC CPU2006 on 4th Generation Core Processor.
  - But degraded throughput in most cases.

- What kind of hardware support should be implemented next in the off-the-shelf HTM?
  - Hardware support for ordered transactions will help in parallel programs.
  - However, many programs contain data dependence. Not only ordered transactions, but also other hardware facilities to avoid conflicts should be implemented.
  - (Intel should fix the adjacent cache line prefetcher!)