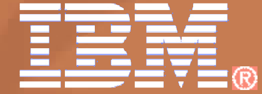


Thread-Level Speculation on Off-the-Shelf Hardware Transactional Memory

Rei Odaira
Takuya Nakaike

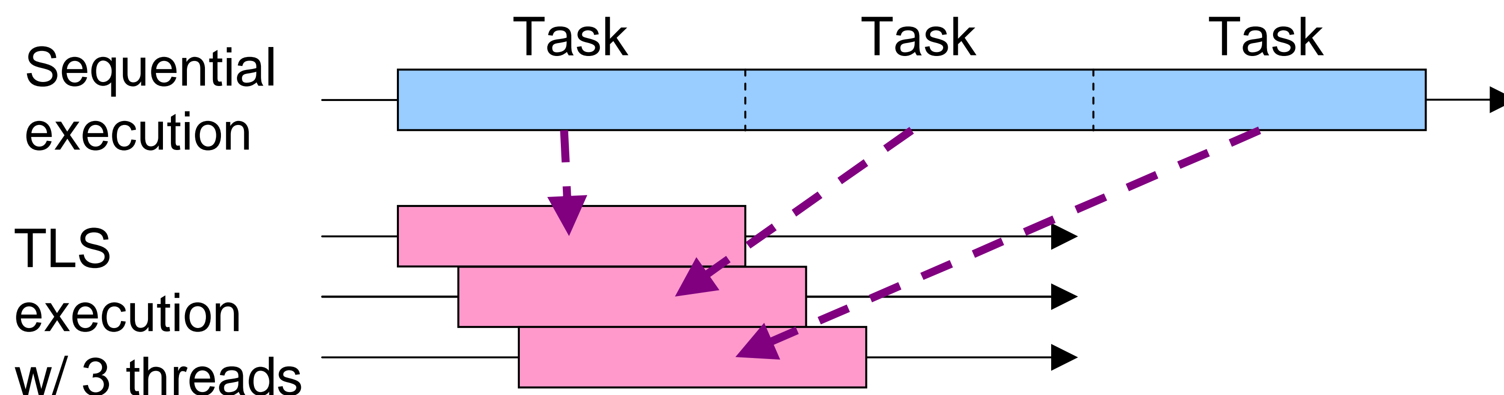
IBM Research – Tokyo

© 2014 IBM Corporation

Thread-Level Speculation (TLS) [Franklin et al.,'92]

or Speculative Multithreading (SpMT)

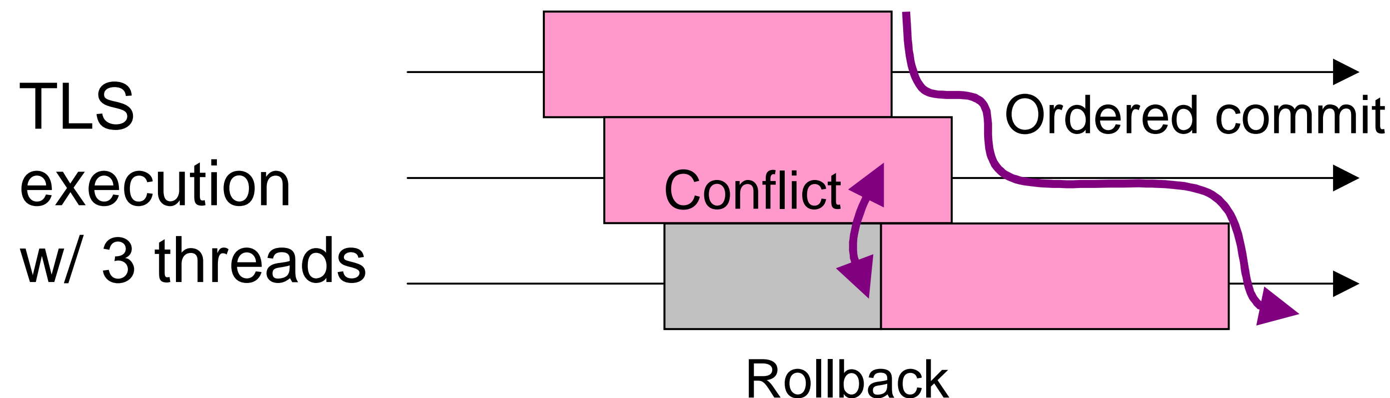
- Speculatively parallelize a sequential program into a multithreaded program.
- What is parallelization?
  - To find data-independent tasks from a program.
- Why speculation?
  - Because a compiler cannot detect every data dependence.



© 2014 IBM Corporation

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



© 2014 IBM Corporation

Hardware Transactional Memory (HTM) Coming into the Market

Blue Gene/Q

zEC12

POWER8

4th Generation Core Processor (Haswell)

- HTM supports…
  - Conflict detection among transactions
  - Execution rollback at a conflict
- HTM satisfies 2/3 of the runtime requirements for TLS!
  - Task = transaction

© 2014 IBM Corporation

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.

© 2014 IBM Corporation

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

© 2014 IBM Corporation

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?

© 2014 IBM Corporation

Transactional Memory

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

- 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.

Thread X

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

Thread Y

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

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


© 2014 IBM Corporation

HTM

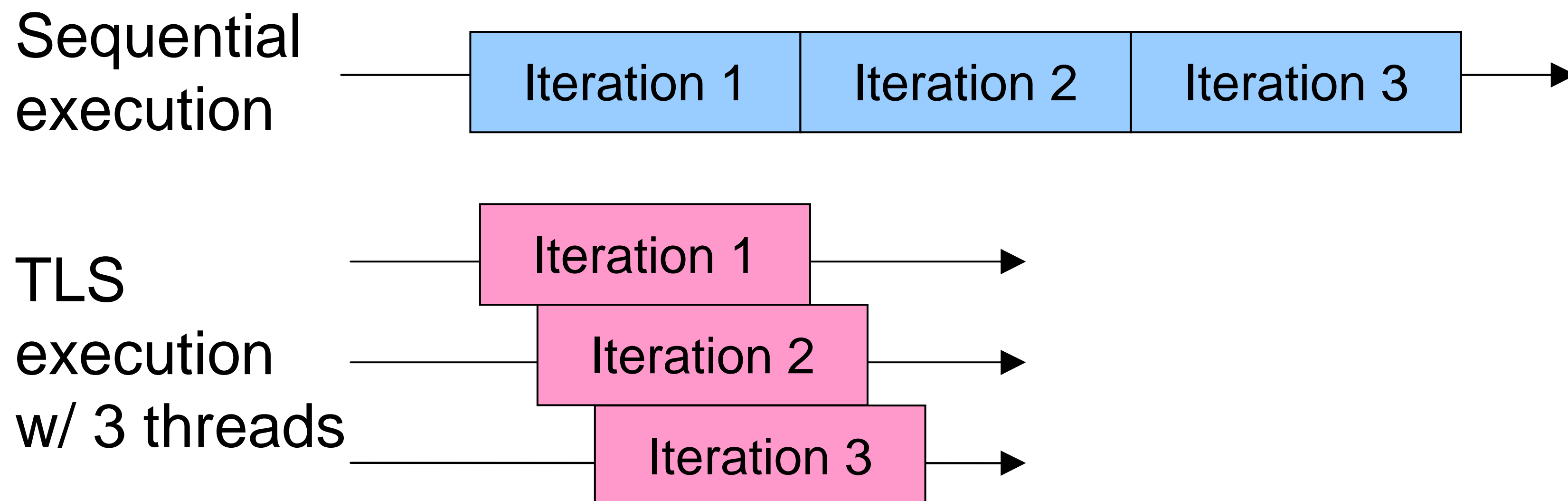
- Instruction set (Intel TSX)
  - XBEGIN: Begin a transaction
  - XEND: End a transaction
  - XABORT, etc.
- Micro-architecture
  - Read and write sets held in CPU caches
  - Conflict detection using CPU cache coherence protocol → Conflict detection by cache line granularity
  - Rollback by discarding 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:
...
```


© 2014 IBM Corporation

TLS for Loops

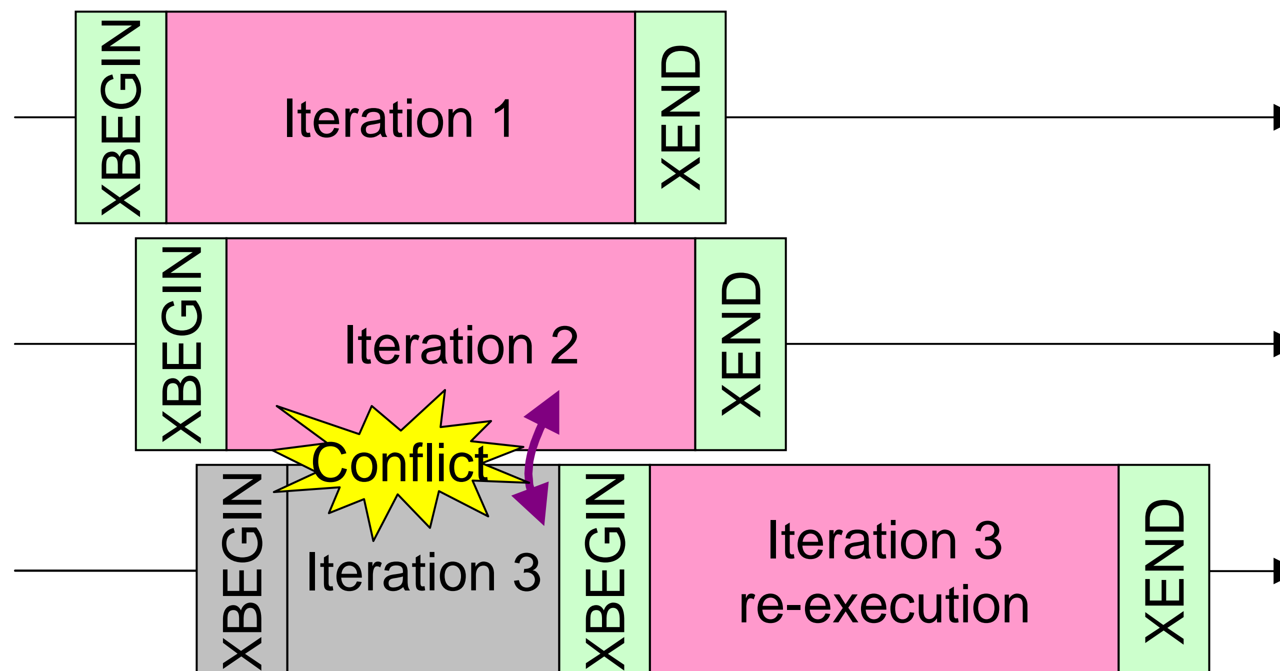
- We focus on frequently executed loops.
  - Task = iteration(s) = transaction
- Why not parallelize function calls?
  - Difficult to implement TLS for function calls on HTM. (Refer to the paper for the details.)



© 2014 IBM Corporation

TLS on HTM

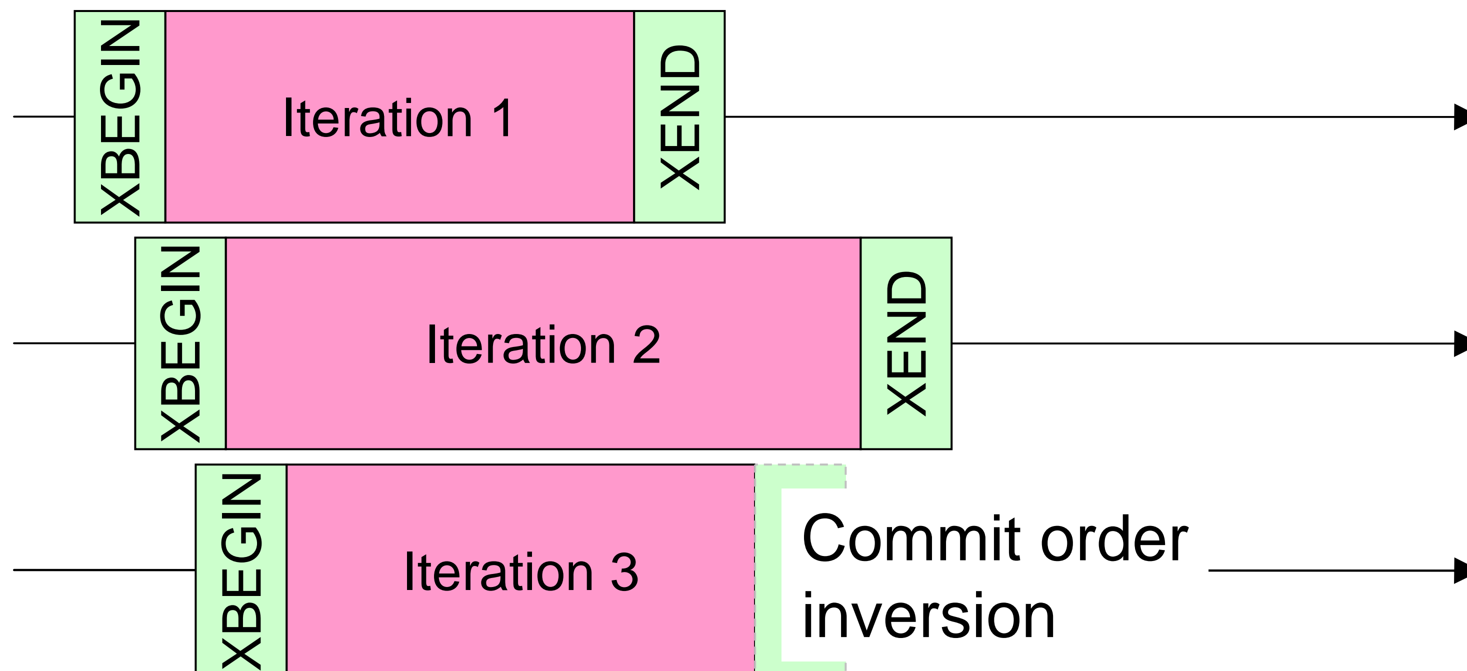
- Enclose each iteration with XBEGIN and XEND.
- Re-execute iteration in case of abort.



© 2014 IBM Corporation

Ordered Transactions

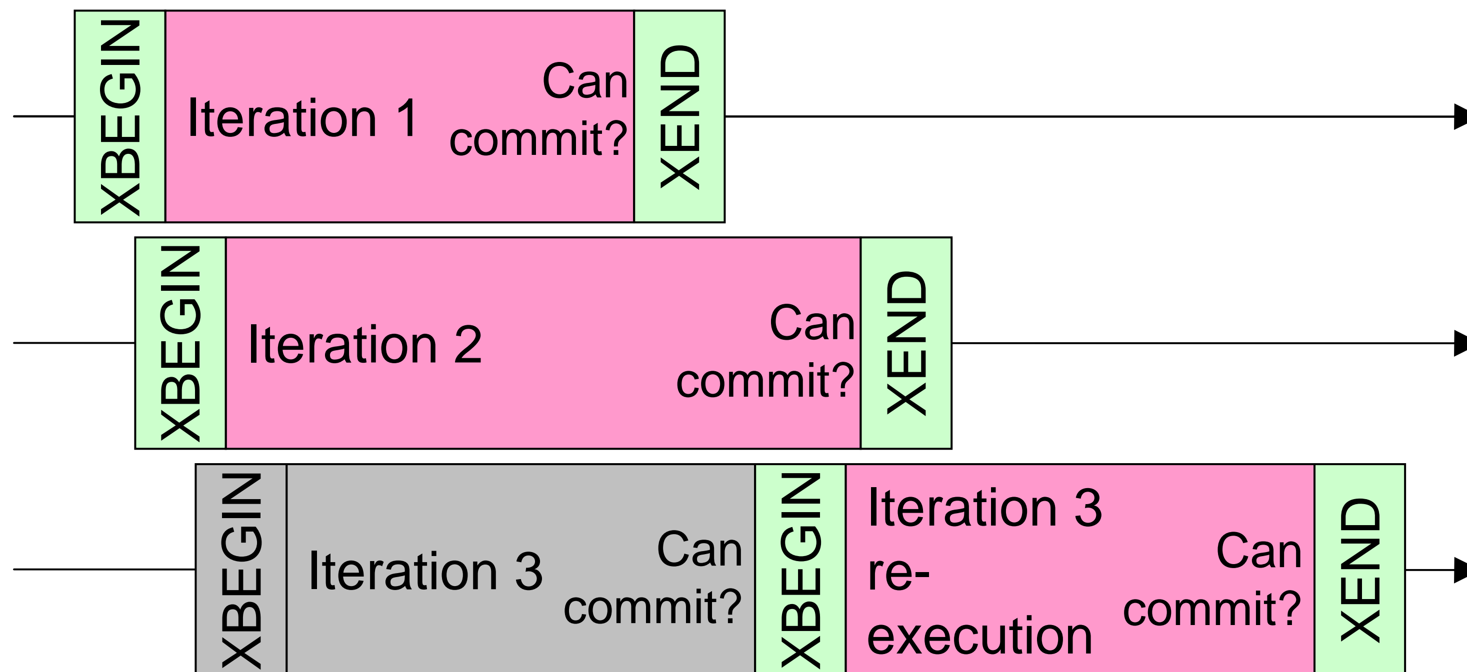
- Must commit in the same order as sequential execution.
  - Because data independence can be guaranteed only after all of the preceding iterations have committed.



© 2014 IBM Corporation

Ordered Transactions by Software

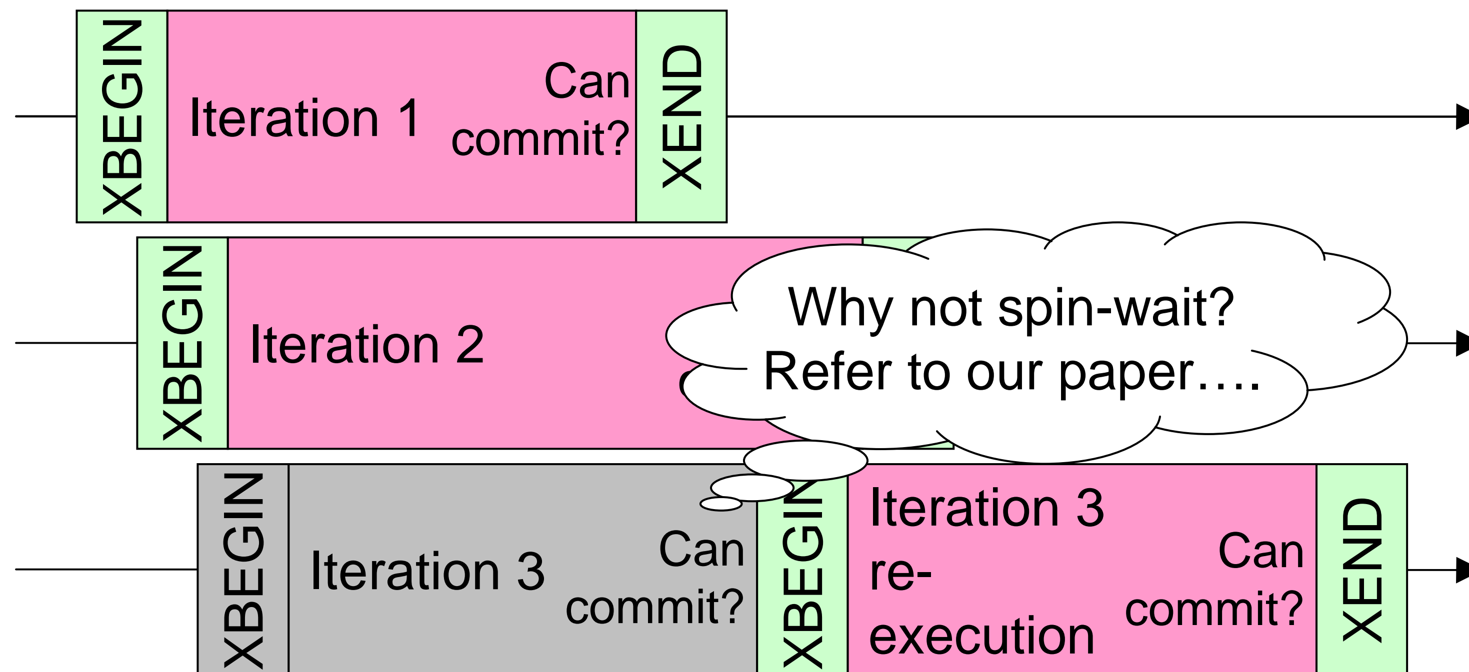
- Hardware support by proposed TLS systems
  - Wait until the preceding iterations commit.
- Software implementation by checking commit order
  - Use a global variable to indicate the next iteration to commit.
  - Abort if cannot commit.



© 2014 IBM Corporation

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.



© 2014 IBM Corporation

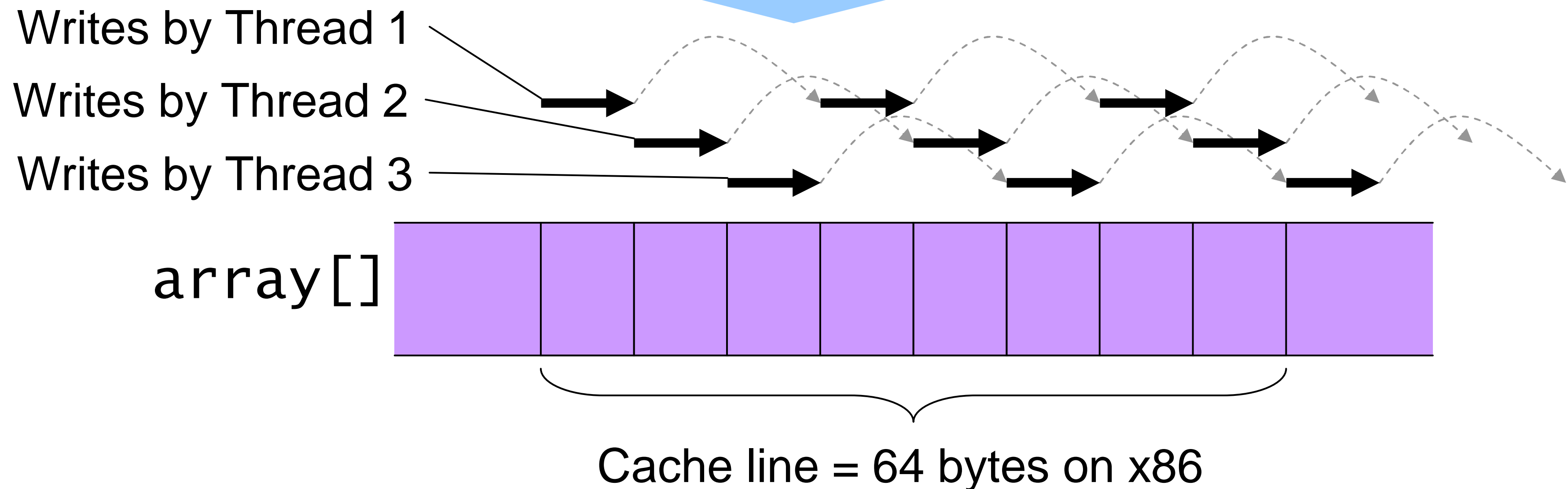
Our Goal

- 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?

© 2014 IBM Corporation

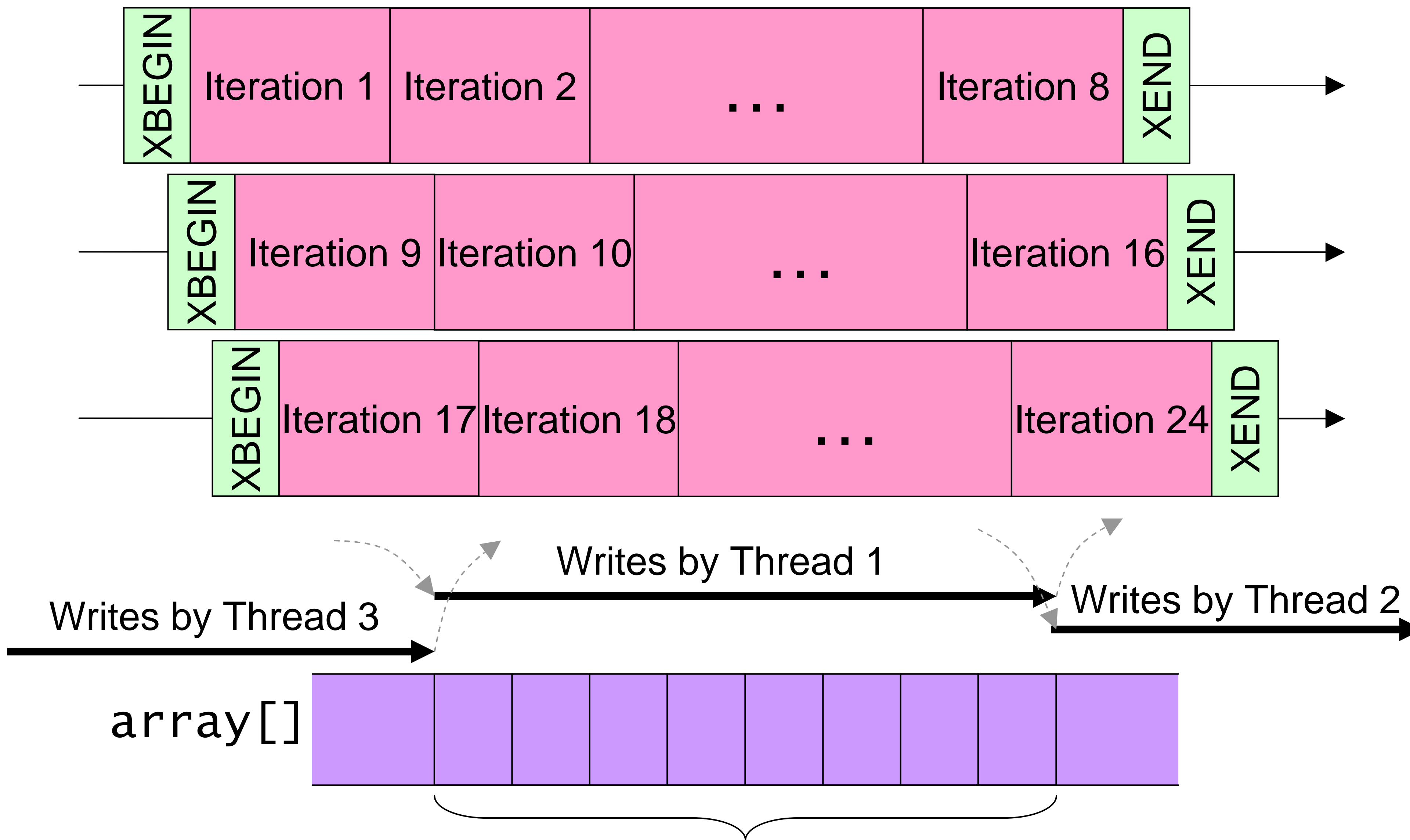
False Sharing due to Cache-Line Granularity Conflict Detection

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



© 2014 IBM Corporation

Transaction Coarsening to Avoid False Sharing



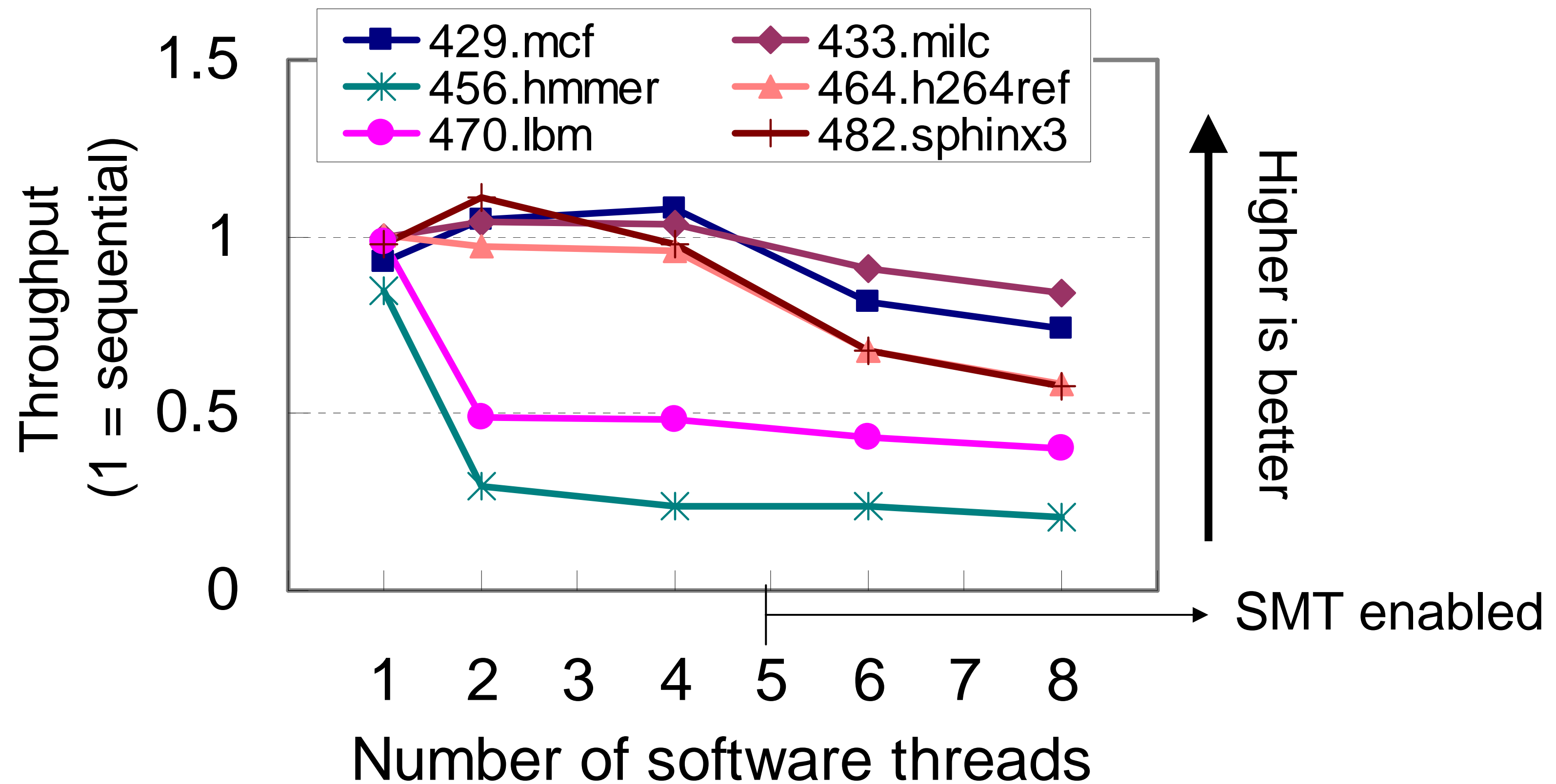
© 2014 IBM Corporation

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

© 2014 IBM Corporation

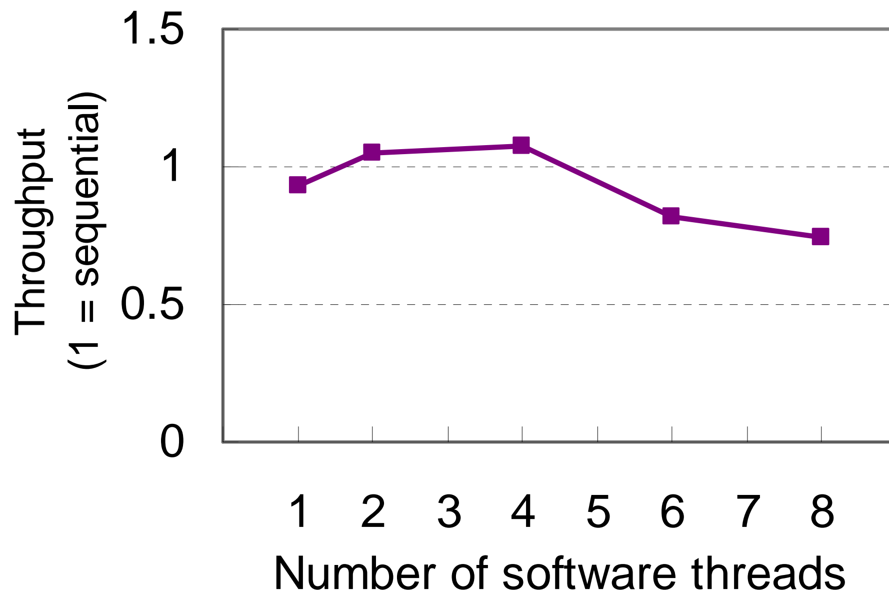
Normalized Throughput Results

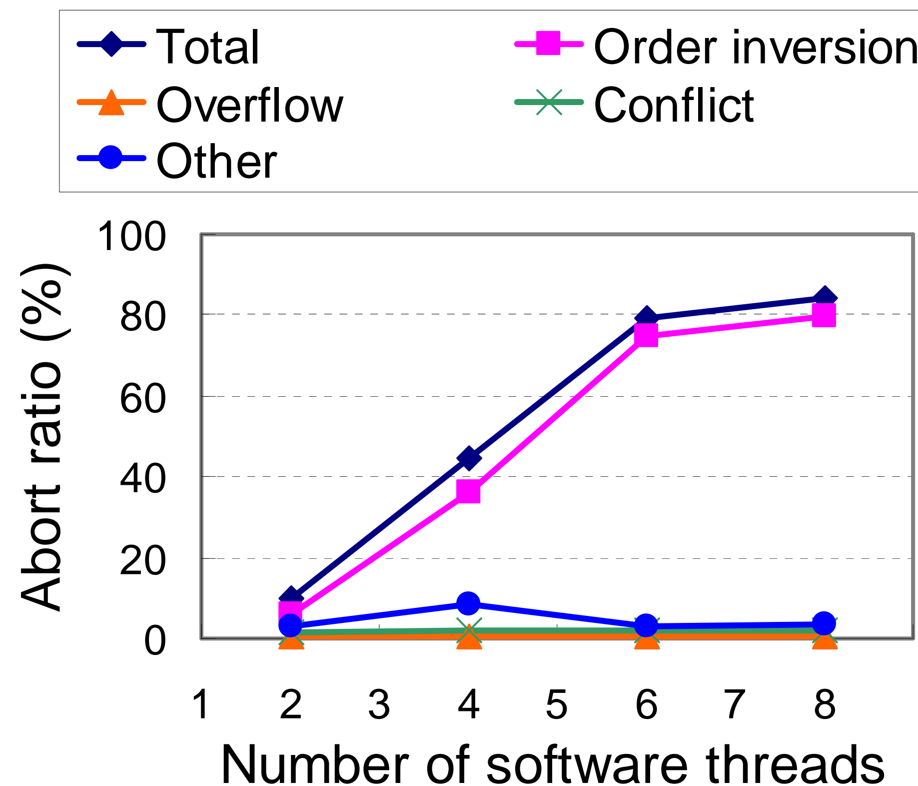


- Up to 11% speedups with 2 or 4 threads.
- But mostly degraded the throughput.

© 2014 IBM Corporation

433.milc

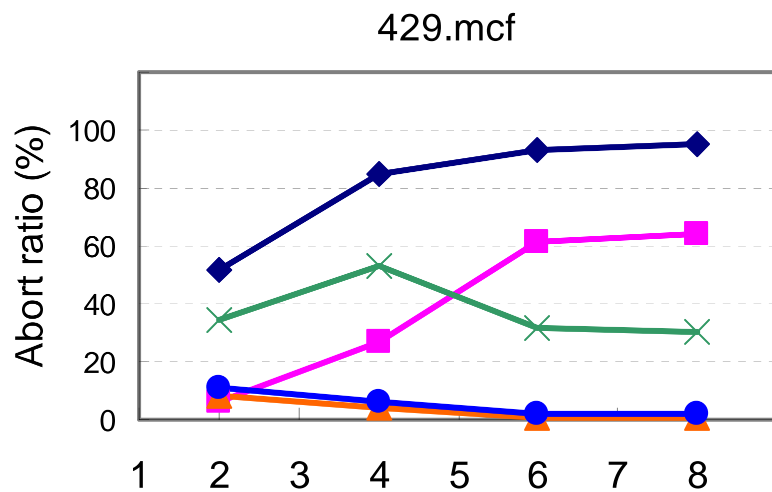


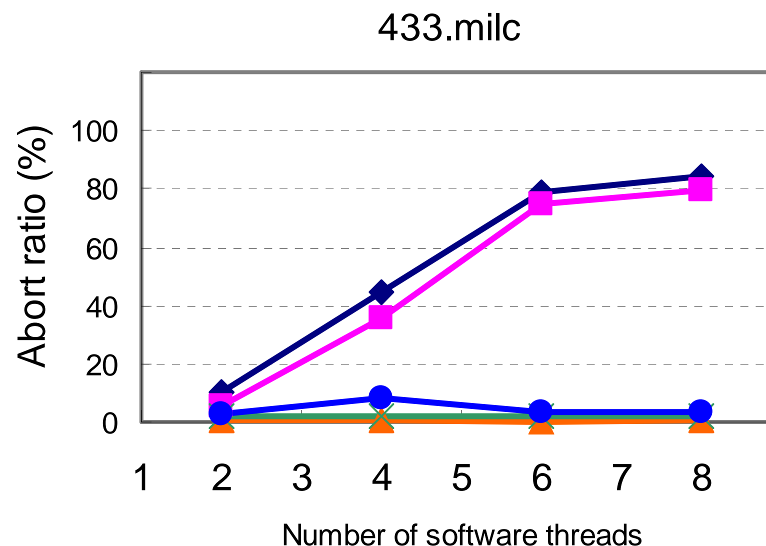


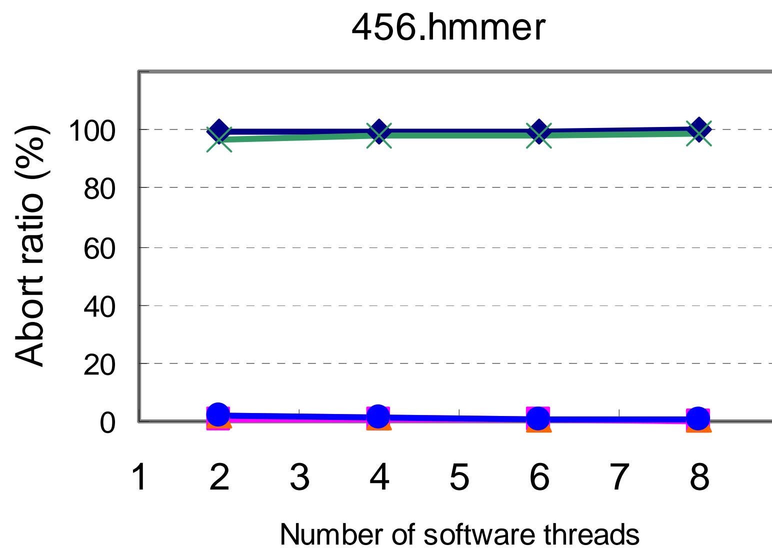
- Parallel program
  - Loop coverage: 23%
- Commit order inversion is a dominant abort reason.
  → Hardware support for ordered transactions will help.

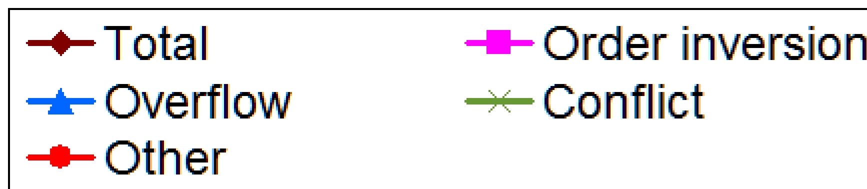
© 2014 IBM Corporation

Abort Statistics (1/2)





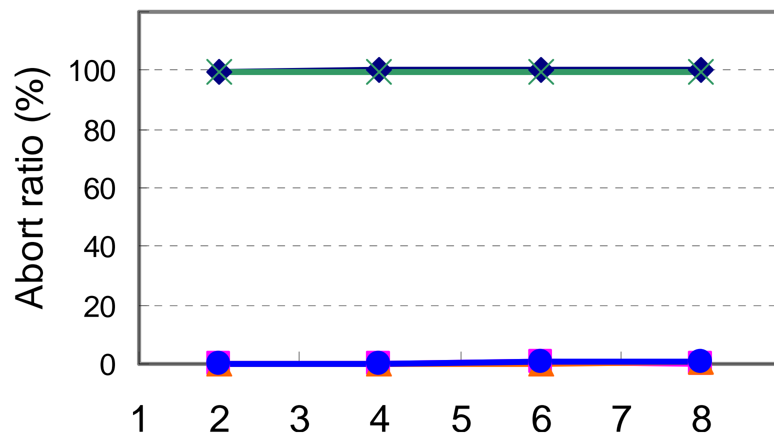


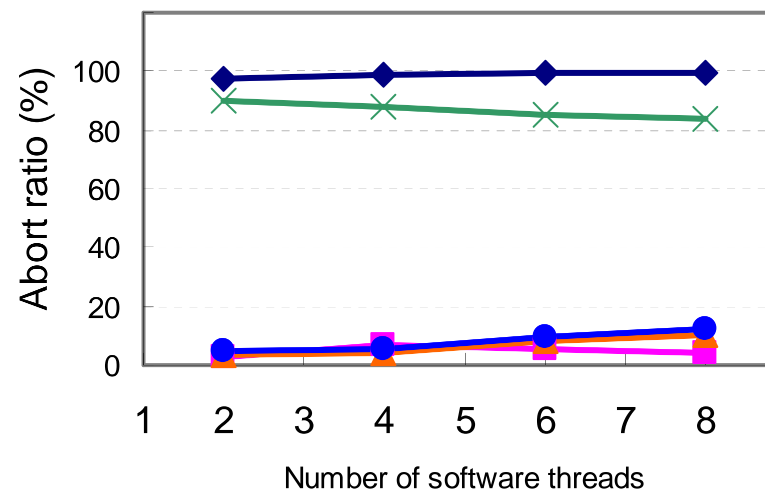


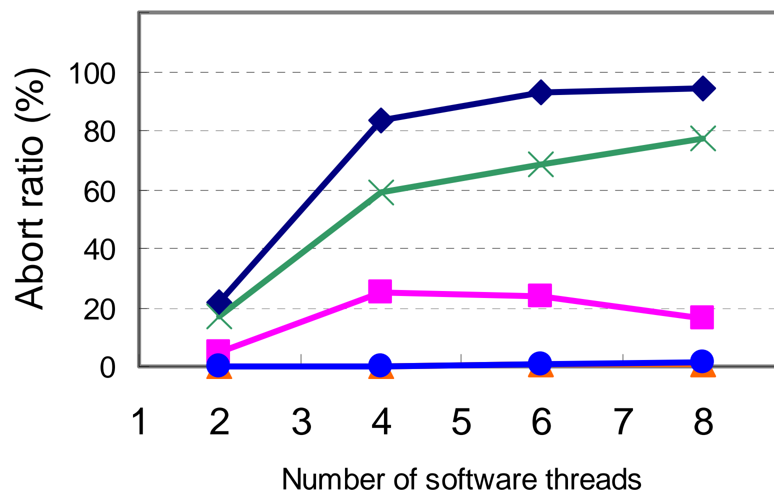
- Conflicts were a dominant abort reason in all of the benchmarks except 433.milc.

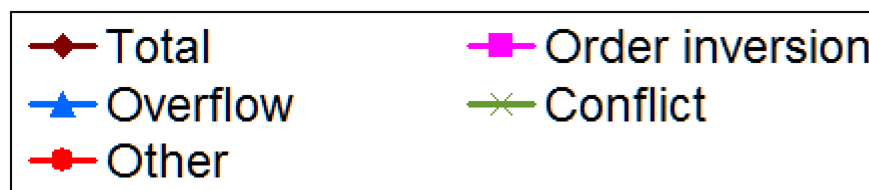
© 2014 IBM Corporation

Abort Statistics (2/2)









- Conflicts were a dominant abort reason in all of the benchmarks except 433.milc.

© 2014 IBM Corporation

Reasons for Conflicts and Possible Hardware Support

Benchmark	Conflict reason	Possible hardware support
429.mcf	RAW dependence	Data forwarding
433.milc	No	
456.hmmer	RAW dependence	Data forwarding
464.h264ref	WAR dependence	Multi-version cache
	WAW dependence (false sharing by prefetching)	(Fix in prefetcher)
470.lbm	WAW dependence (false sharing)	Word-level conflict detection
482.sphinx3	WAW dependence (false sharing by prefetching)	(Fix in prefetcher)

© 2014 IBM Corporation

Examples of Read-After-Write Data Dependence

429.mcf

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

456.hmmer

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

- Hardware support already proposed in TLS literatures.
  - Data forwarding.

© 2014 IBM Corporation

Example of Write-After-Read Data Dependence

464.h264ref

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

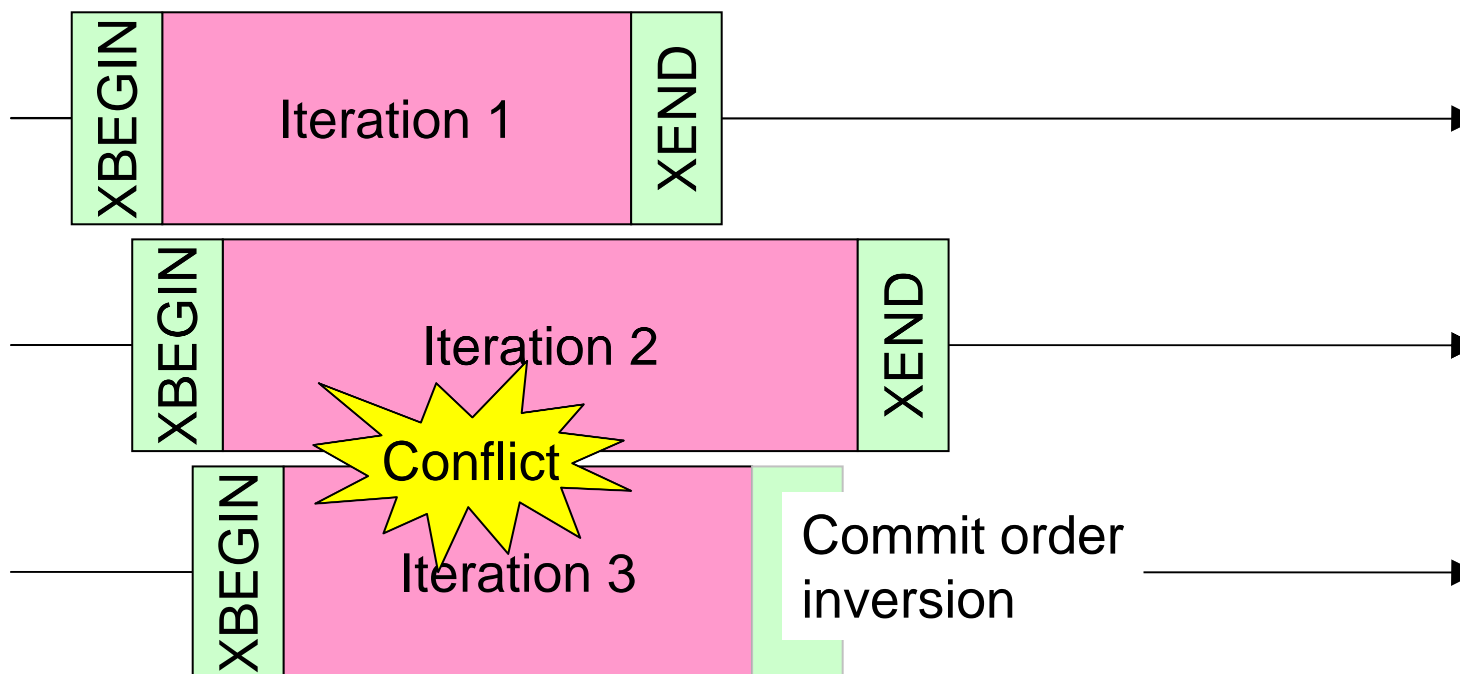
```
static DATA line[N];

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.

© 2014 IBM Corporation

Conflicts Precede Commit Order Inversion

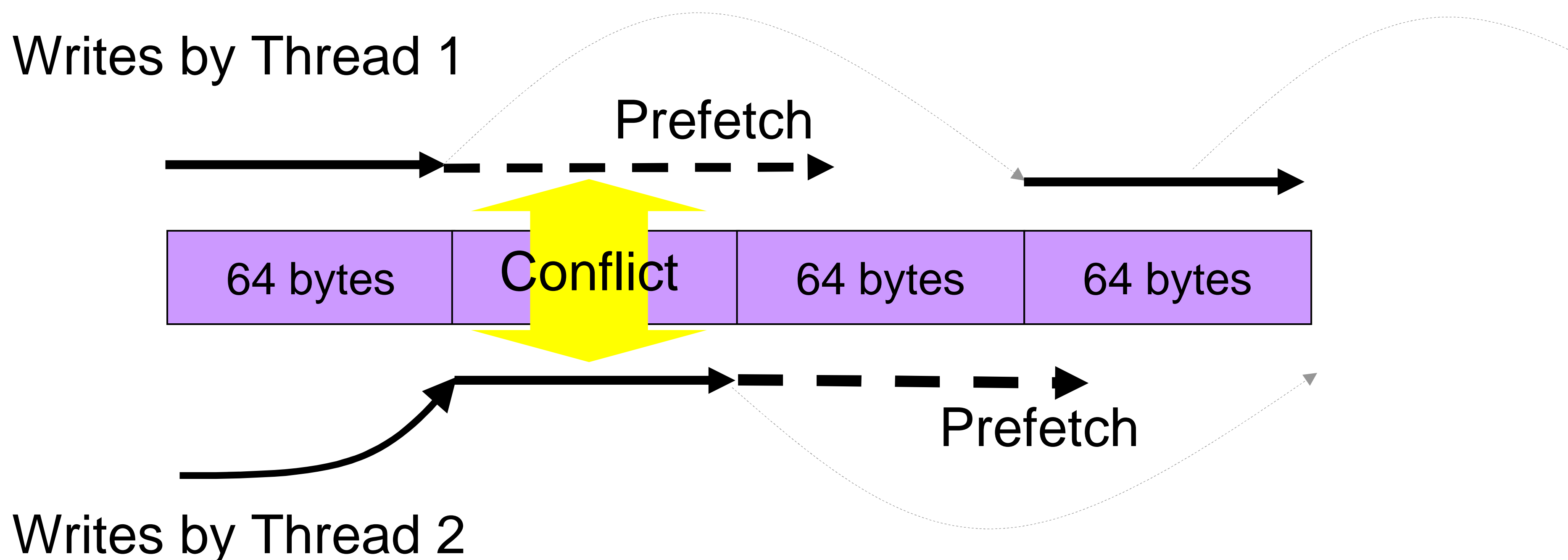
- Commit order matters only when most of the transactions reach the committing points.
- With data dependence, most of the transactions cannot run to the end.



© 2014 IBM Corporation

Conflicts due to Prefetching

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



© 2014 IBM Corporation

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!)

© 2014 IBM Corporation