

## Incrementally Parallelizing Database Transactions with Thread-Level Speculation

Todd C. Mowry  
Carnegie Mellon University

*(in collaboration with Chris Colohan,  
J. Gregory Steffan, and Anastasia Ailamaki)*

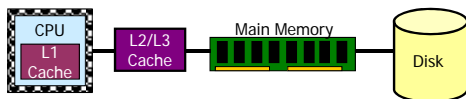
## Twofold Speedup on a Quad-Core with 1 Month of Programmer Effort: A Case Study with BerkeleyDB

Todd C. Mowry  
Carnegie Mellon University

*(in collaboration with Chris Colohan,  
J. Gregory Steffan, and Anastasia Ailamaki)*

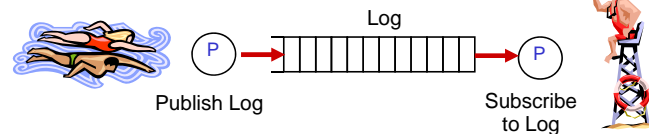
## What Have I Worked On in the Past?

- Automatically extracting **thread-level parallelism**
- Smarter caching** to better utilize deep memory hierarchies
  - SRAM to DRAM; DRAM to disk; local disk to remote web server
- Redesigning core **database algorithms & data structures**
  - to exploit modern processor architectures



## What Am I Working on Now?

- Log-Based Architectures Project**
  - Motivation:** detect (& fix?) software correctness problems in real time
  - Approach:** logging mechanism allows cores to monitor other cores




- Claytronics Project**



## Today's Talk

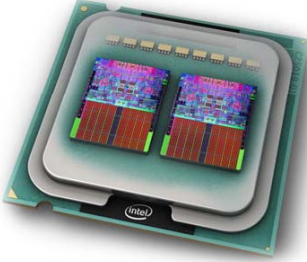
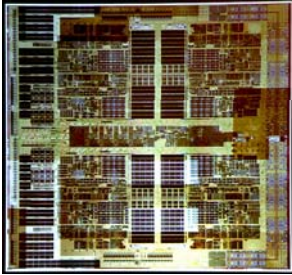
- Chris Colohan's Ph.D. thesis work



Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

5

## Multicore is Here

Intel's Core 2 Quad      AMD's Quad-Core Opteron ("Barcelona")

- Quad-cores are now common
  - 8, 16, 32... cores expected in the future
- Great for throughput, but what about latency?

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

6

## Exploiting Multicore

One view:

- Don't worry: everyone will write parallel software from now on
  - and it will all speed up nicely

Rebuttal:

- Writing parallel software is difficult
- Getting large speedups is also difficult
- What about legacy codes?

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

7

## Exploiting Multicore

Another view:

- Don't worry: the compiler will automatically parallelize everything
  - and it will all speed up nicely

Rebuttal:

- Beyond regular matrix-based codes, compilers really struggle with this
- Ambiguous dependences are a stumbling block

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

8

## The Stampede Project @ CMU

Idea:

- Using novel hardware & compiler support, allow the compiler to *optimistically* create parallel threads
  - "Thread-Level Speculation" (TLS)
- Rollback and recover if speculation fails

Our early work:

- Automatically parallelize SPEC Integer benchmarks
  - Resulted in speedups of roughly 20-35%

This work:

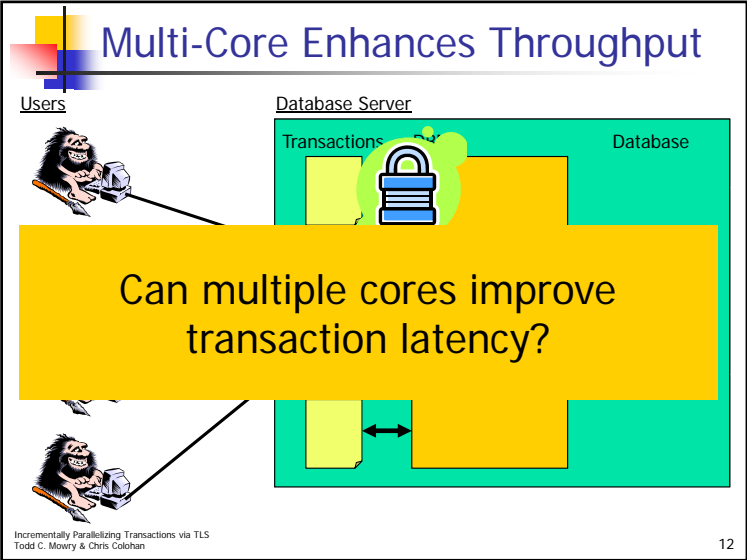
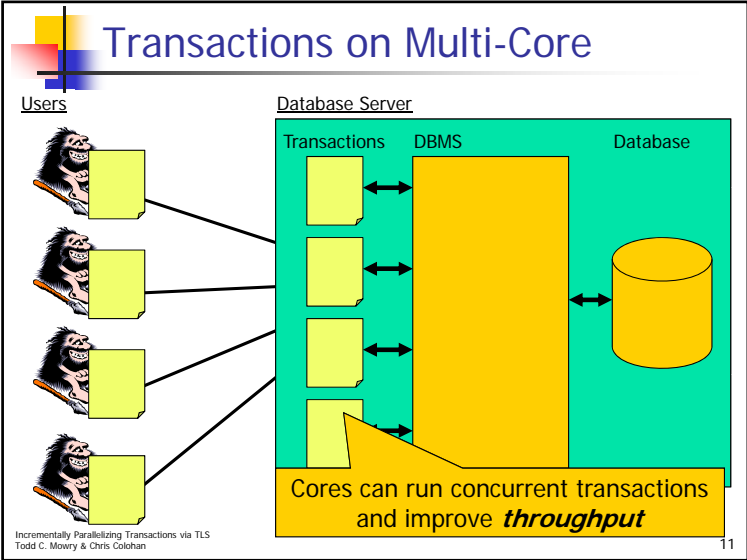
- Focus on *large, legacy code that is hard to parallelize*
- "semi-automatic" approach: the programmer is involved

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

## Case Study: BerkeleyDB

- We chose to parallelize *individual transactions* in BerkeleyDB
- The code was not written to support parallelism
  - Much the opposite: it takes advantage of the fact that there is never concurrency within a given transaction
- Rewriting the code to support intra-transaction parallelism would be extremely painful
  - Problems throughout the 200K lines of code
  - Would probably need to start over again from scratch

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan



## Parallelizing transactions

```

SELECT cust_info FROM customer;
UPDATE district WITH order_id;
INSERT order_id INTO new_order;
foreach(item) {
  GET quantity FROM stock;
  quantity--;
  UPDATE stock WITH quantity;
  INSERT item INTO order_line;
}
        
```

DBMS

- Intra-query parallelism
  - Used for long-running queries (decision support)
  - Does not work for short queries
- Short queries dominate in commercial workloads

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

13

## Parallelizing transactions

```

SELECT cust_info FROM customer;
UPDATE district WITH order_id;
INSERT order_id INTO new_order;
foreach(item) {
  GET quantity FROM stock;
  quantity--;
  UPDATE stock WITH quantity;
  INSERT item INTO order_line;
}
        
```

DBMS

- Intra-transaction parallelism
  - Each thread spans multiple queries
- Hard to add to existing systems!
  - Need to change interface, add latches and locks, worry about correctness of parallel execution...

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

14

## Parallelizing transactions

```

SELECT cust_info FROM customer;
UPDATE district WITH order_id;
INSERT order_id INTO new_order;
foreach(item) {
  GET quantity FROM stock;
  quantity--;
  UPDATE stock WITH quantity;
  INSERT item INTO order_line;
}
        
```

DBMS

### Thread Level Speculation (TLS) makes parallelization easier.

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

15

## Thread Level Speculation (TLS)

Sequential

Parallel

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

16

## Thread Level Speculation (TLS)

The diagram shows two vertical bars representing execution over time. The left bar is labeled 'Sequential' and shows a single vertical bar with four segments labeled `*p=`, `*q=`, `=*p`, and `=*q`. The right bar is labeled 'Parallel' and is divided into 'Epoch 1' and 'Epoch 2'. In Epoch 1, the sequential order is followed. In Epoch 2, the order is violated: `*p=` is executed before `*q=`. A red arrow labeled 'Violation!' points to the `*p=` segment in Epoch 2. Green arrows show the continuation of `*q=` and `=*q` segments. A list of bullet points is on the right.

- Use *epochs*
- Detect violations
- Restart to recover
- Buffer state
- Worst case:
  - Sequential
- Best case:
  - Fully parallel

**Data dependences limit performance.**

Todd C. Mowry & Chris Colohan

## TLS in Database Systems

The diagram compares 'Non-Database TLS' and 'TLS in Database Systems'. 'Non-Database TLS' shows a single vertical bar with a red arrow indicating a violation. 'TLS in Database Systems' shows multiple vertical bars representing concurrent transactions. A yellow callout box points to the database system diagram with the text: 'Large epochs: • More dependences • Must tolerate • More state • Bigger buffers'. A yellow callout box points to the concurrent transactions with the text: 'Concurrent transactions'.

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

## Violations as a Feedback Signal

The diagram shows a feedback loop. On the left, a 'Sequential' bar is shown. On the right, a 'Parallel' bar shows a 'Violation!' where `*p=` is executed before `*q=`. A thought bubble from a cartoon character says 'Must...Make...Faster'. A computer monitor displays a list of memory addresses: `0x0FD8`, `0xFD20`, `0xFC0`, and `0xFC18`. Green arrows indicate the flow from the violation to the feedback signal and then to the performance goal.

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

## Violations as a Feedback Signal

The diagram shows a feedback loop. On the left, a 'Sequential' bar is shown. On the right, a 'Parallel' bar shows a 'Violation!' where `*p=` is executed before `*q=`. A third bar on the right shows the sequential order being restored after the violation. Red arrows indicate the flow from the violation to the feedback signal and then to the performance goal.

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

## Eliminating Violations

0x0FD8 →  
 0x0FD20  
 0x0FC0 →  
 0x0FC18

**All-or-nothing execution makes optimization harder**

Parallel      Eliminate \*p Dep.

Optimization may make slower?

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan      21

## Tolerating Violations: Sub-threads

Time ↓

Eliminate \*p Dep.      Sub-threads

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan      22

## Sub-threads

- Periodic checkpoints of a speculative thread
- Makes TLS work well with:
  - Large speculative threads
  - Unpredictable frequent dependences

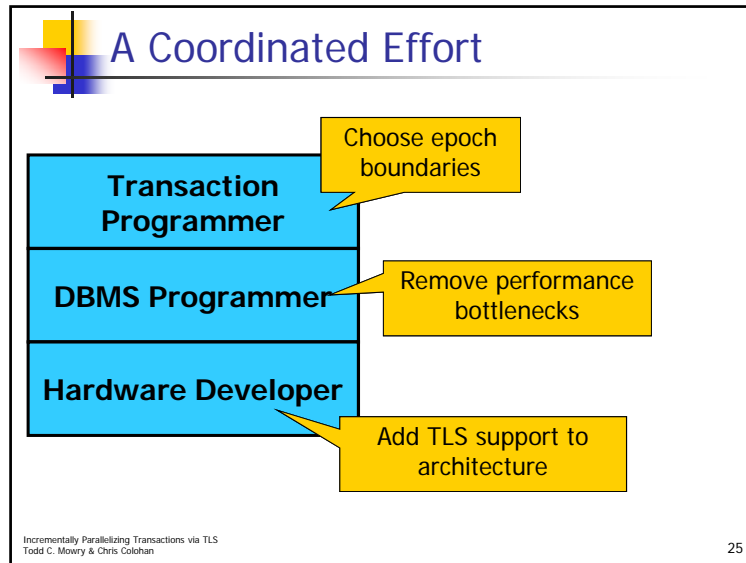
**Speed up database transaction response time by a factor of 1.9 to 2.9.**

Todd C. Mowry & Chris Colohan      23

## A Coordinated Effort

Transactions (TPC-C)  
 DBMS (BerkeleyDB)  
 Hardware (Simulated machine)

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan      24



- ## What's New
- Intra-transaction parallelism
    - Without changing the transactions
    - With minor changes to the DBMS
    - Without having to worry about locking
    - Without introducing concurrency bugs
    - With good performance
  - Halve transaction latency on four cores
- Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan
- 26

- ## Outline
- Modifying the DBMS to exploit TLS
    - Dividing transactions into epochs
    - Removing bottlenecks in the DBMS
  - Results
  - Conclusions
- |                        |
|------------------------|
| Transaction Programmer |
| DBMS Programmer        |
| Architect              |
- Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan
- 27

## Case Study: New Order (TPC-C)

```

GET cust_info FROM customer;
UPDATE district WITH order_id;
INSERT order_id INTO new_order;
foreach(item) {
  GET quantity FROM stock
  WHERE i_id=item;
  UPDATE stock WITH quantity-1
  WHERE i_id=item;
  INSERT item INTO order_line;
}

```

*78% of transaction execution time*

- Only dependence is the **quantity** field
  - Very unlikely to occur (1/100,000)

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

28

## Case Study: New Order (TPC-C)

```

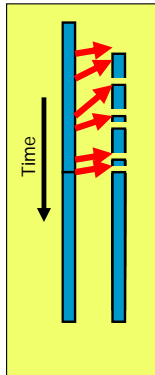
GET cust_info FROM customer;
UPDATE district WITH order_id;
INSERT order_id INTO new_order;
foreach(item) {
  GET quantity FROM stock
  WHERE i_id=item;
  UPDATE stock WITH quantity-1
  WHERE i_id=item;
  INSERT item INTO
}
GET cust_info FROM customer;
UPDATE district WITH order_id;
INSERT order_id INTO new_order;
TLS_foreach(item) {
  GET quantity FROM stock
  WHERE i_id=item;
  UPDATE stock WITH quantity-1
  WHERE i_id=item;
  INSERT item INTO order_line;
}
    
```

## Outline

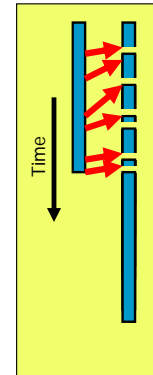
- Modifying the DBMS to exploit TLS
  - Dividing transactions into epochs
  - Removing bottlenecks in the DBMS
- Results
- Conclusions

Transaction Programmer
DBMS Programmer
Architect

## Dependencies in DBMS



## Dependencies in DBMS

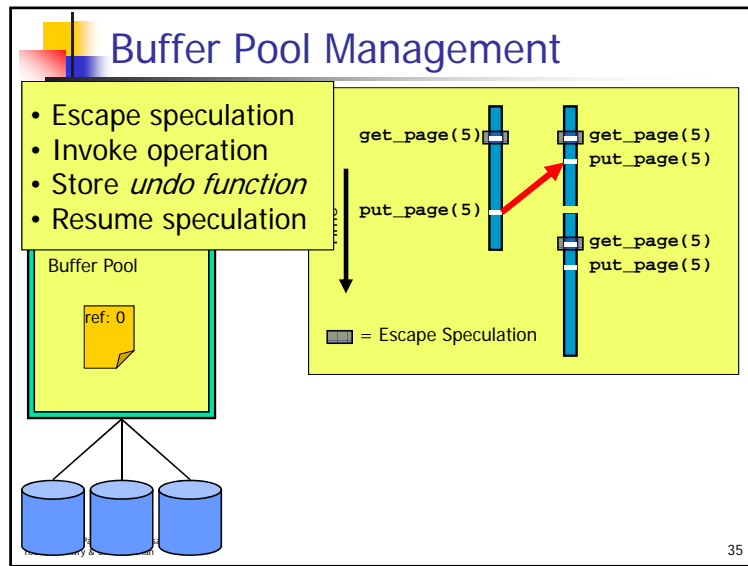
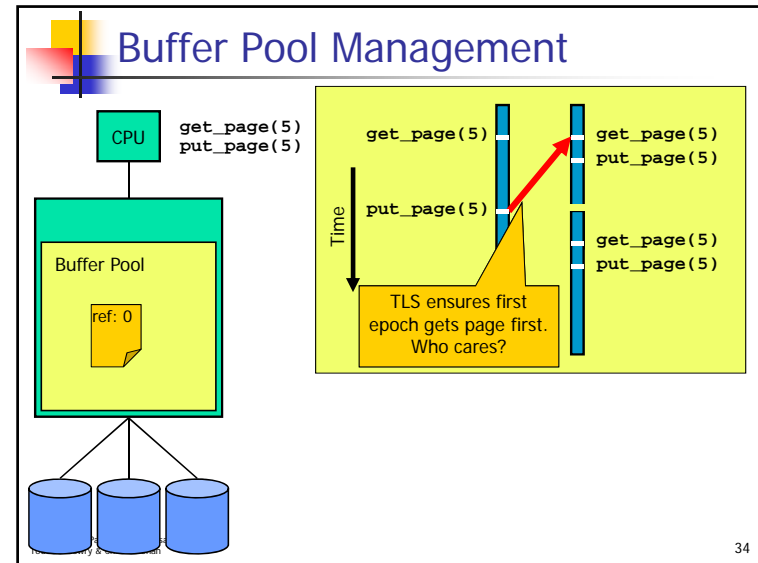
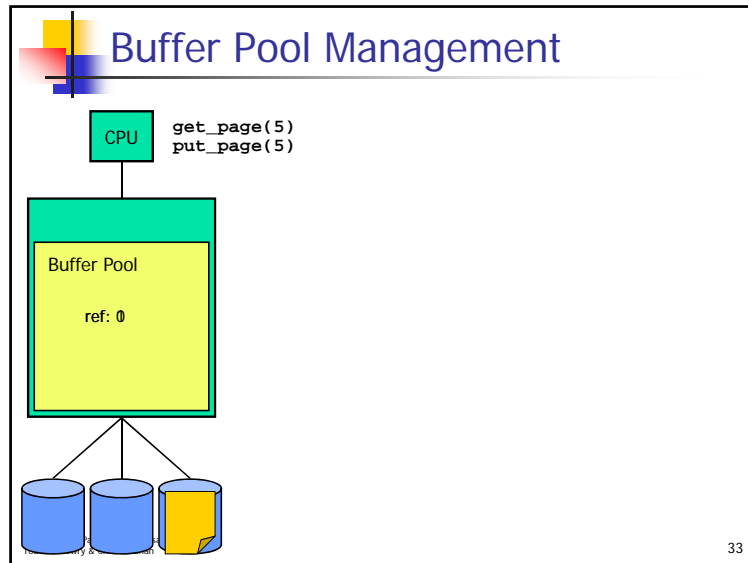


***Dependencies serialize execution!***

Performance tuning:

- Profile execution
- Remove *bottleneck* dependence
- Repeat





## get\_page() wrapper

```

page_t *get_page_wrapper(pageid_t id) {
    static tls_mutex mut;
    page_t *ret;

    tls_escape_speculation();
    check_get_arguments(id);
    tls_acquire_mutex(&mut);

    ret = get_page(id);
    tls_release_mutex(&mut);
    tls_on_violation(put, ret);
    tls_resume_speculation();

    return ret;
}

```

→ Wraps `get_page()`

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

36

## get\_page() wrapper

```

page_t *get_page_wrapper(pageid_t id) {
    static tls_mutex mut;
    page_t *ret;

    tls_escape_speculation();
    check_get_arguments(id);
    tls_acquire_mutex(&mut);

    ret = get_page(id);

    tls_release_mutex(&mut);
    tls_on_violation(put, ret);
    tls_resume_speculation();

    return ret;
}

```

→ No violations while calling `get_page()`

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

37

## get\_page() wrapper

```

page_t *get_page_wrapper(pageid_t id) {
    static tls_mutex mut;
    page_t *ret;

    tls_escape_speculation();
    check_get_arguments(id);
    tls_acquire_mutex(&mut);

    ret = get_page(id);

    tls_release_mutex(&mut);
    tls_on_violation(put, ret);
    tls_resume_speculation();

    return ret;
}

```

→ May get bad input data from speculative thread!

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

38

## get\_page() wrapper

```

page_t *get_page_wrapper(pageid_t id) {
    static tls_mutex mut;
    page_t *ret;

    tls_escape_speculation();
    check_get_arguments(id);
    tls_acquire_mutex(&mut);

    ret = get_page(id);

    tls_release_mutex(&mut);
    tls_on_violation(put, ret);
    tls_resume_speculation();

    return ret;
}

```

→ Only one epoch per transaction at a time

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

39

## get\_page() wrapper

```

page_t *get_page_wrapper(pageid_t id) {
    static tls_mutex mut;
    page_t *ret;

    tls_escape_speculation();
    check_get_arguments(id);
    tls_acquire_mutex(&mut);

    ret = get_page(id);

    tls_release_mutex(&mut);
    tls_on_violation(put, ret);
    tls_resume_speculation();

    return ret;
}

```

→ How to undo `get_page()`

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

40

## get\_page() wrapper

```

page_t *get_page_wrapper(
static tls_mutex mutex;
page_t *ret;

tls_escape_speculat:
check_get_arguments:
tls_acquire_mutex(&

ret = get_page(id);

tls_release_mutex(&
tls_on_violation(pu
tls_resume_speculat:

return ret;
}

```

- **Isolated**
  - Undoing this operation does not cause cascading aborts
- **Undoable**
  - Easy way to return system to initial state
- **Can also be used for:**
  - Cursor management
  - malloc()

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

41

## Buffer Pool Management

CPU: get\_page(5), put\_page(5)

Buffer Pool: ref: 0

Time ↓

get\_page(5) [Escape Speculation]

put\_page(5)

get\_page(5) [Escape Speculation]

put\_page(5)

get\_page(5) [Escape Speculation]

put\_page(5)

Not undoable!

█ = Escape Speculation

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

42

## Buffer Pool Management

CPU: get\_page(5), put\_page(5)

Buffer Pool: ref: 0

Time ↓

get\_page(5) [Escape Speculation]

put\_page(5)

get\_page(5) [Escape Speculation]

put\_page(5)

█ = Escape Speculation

- **Delay put\_page until end of epoch**
  - Avoid dependence

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

43

## Removing Bottleneck Dependences

We introduce three techniques:

- **Delay operations** until non-speculative
  - Mutex and lock *acquire* and *release*
  - Buffer pool, memory, and cursor *release*
  - Log sequence number assignment
- **Escape speculation**
  - Buffer pool, memory, and cursor *allocation*
- **Traditional parallelization**
  - Memory allocation, cursor pool, error checks, false sharing

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

44

## Outline

- Modifying the DBMS to exploit TLS
  - Dividing transactions into epochs
  - Removing bottlenecks in the DBMS
- Results
- Conclusions

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

45

## Experimental Setup

- Detailed simulation
  - Superscalar, out-of-order, 128 entry reorder buffer
  - Memory hierarchy modeled in detail
- TPC-C transactions on BerkeleyDB
  - In-core database
  - Single user
  - Single warehouse
  - Measure interval of 100 transactions
  - Measuring *latency* not throughput

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

46

## Optimizing the DBMS: New Order

Time (normalized)

Legend: Idle CPU, Violated, Cache Miss, Busy

Annotations: 26% improvement, Other CPUs not helping, Cache misses increase, Can't optimize much more

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

47

## Optimizing the DBMS: New Order

Time (normalized)

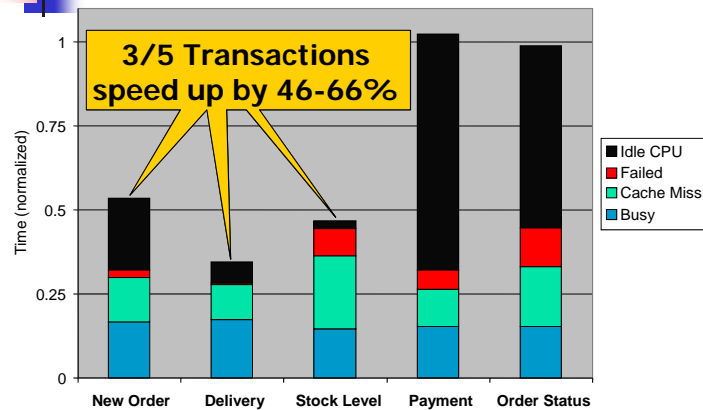
Legend: Idle CPU, Violated, Cache Miss, Busy

This process took Chris 30 days and <1200 lines of code.

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

48

## Other TPC-C Transactions



Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

49

## Conclusions

- A new form of parallelism for databases
  - Tool for attacking transaction **latency**
- Intra-transaction parallelism
  - Without major changes to DBMS
- TLS can be applied to more than transactions
  
- Halve transaction latency by using 4 CPUs

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

50

## Final Thoughts

- We achieved respectable speedups:
  - On a large piece of software that was written without parallelism in mind
  - With roughly a month of (non-expert) programmer effort
- To do this, we need TLS support plus:
  - Feedback on which instruction pairs cause dependence violations
  - Sub-thread support to minimize cost of failed speculation
- There is hope for large dusty-deck codes!!!

Incrementally Parallelizing Transactions via TLS  
Todd C. Mowry & Chris Colohan

51