

Using Software Transactional Memory In Interrupt-Driven Systems

Michael J. Schultz

Department of Mathematics, Statistics, and Computer Science
Marquette University

Thesis Defense

Introduction

Thesis Statement

Software transactional memory can be used in interrupt-driven device drivers as a method to automate and provide fine-grained synchronization between the upper and lower halves.

Operating Systems

- *operating system*—low-level software on a computer.
- *application programming interface*—set of functions a programmer can use.
- *device driver*—deals with high-level API and low-level hardware.
- *interrupt*—external signal to processor.
- *jitter*—variations in delay of interrupt handling.

Jitter Avoidance

Don't disable interrupts!

```
global integer our_var = 1

function increment(void)
    our_var++
```

```
.data
    .globl  our_var
our_var: 0x0001

.text
increment:
    lw     s0, 0x4c(zero)
    addiu s0, s0, 1
    sw     s0, 0x4c(zero)
    j     ra
```

Jitter Avoidance

Don't disable interrupts!

```
global integer our_var = 1

function increment(void)
    our_var++
```

```
.data
    .globl  our_var
our_var: 0x0001

.text
increment:
    lw     s0, 0x4c(zero)
    addiu s0, s0, 1
    sw     s0, 0x4c(zero)
    j     ra
```

Jitter Avoidance

Don't disable interrupts!

```
global integer our_var = 1

function increment(void)
    our_var++
```

```
.data
    .globl  our_var
our_var: 0x0001

.text
increment:
    lw     s0, 0x4c(zero)
    addiu s0, s0, 1
    sw     s0, 0x4c(zero)
    j      ra
```

Related Work

Transactional Memory

- Transactions began in database community.
- Brought to systems as concurrency control for multiprocessors.
- Hardware TM
 - Suffer from physical memory boundaries.
- Software TM
 - Blocking vs. non-blocking implementations.
 - Blocking STM allows progress guarantees.
- Hybrid TM
 - HTM is expensive and restrictive.
 - STM needs more space and time to compute.
 - Hybrid TM takes advantages from both (speed and reliability).
 - Also takes disadvantages.






Related Work

Operating Systems

- Lock-free kernels: Synthesis and Cache.
- Use CAS and DCAS opcodes for shared data structures.
- TxLinux uses HTM and *cxspinlocks*.

Related Work

For Further Reading...

-  Massalin and Pu. “A lock-free multiprocessor OS kernel.” Tech. Rep. CUUCS-005-91, Columbia University.
-  Gray and Reuter. *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, First Edition.
-  Herlihy and Moss. “Transactional memory: Architectural support for lock-free data structures.” *ISCA '93*.
-  Rossbach et al. “TxLinux: using and managing hardware transactional memory in an operating system.” *SOSP '07*.
-  Ni et al. “Design and Implementation of Transactional Constructs for C/C++.” *OOPSLA '08*.

Major Contributions

- Modernized port of the Xinu kernel for IA-32.
- Method for integrating STM library into kernel.
- Embedded Xinu augmented with STM:
 - “Transactional Xinu.”
- Evaluation of interrupt-driven device drivers in Transactional Xinu.

Outline

- 1 Design and Implementation
 - Critical Sections
 - Transactional System
- 2 Performance Results
 - Measurements
 - Testing Methodology
- 3 Summary and Future Work
 - Summary
 - Future Work

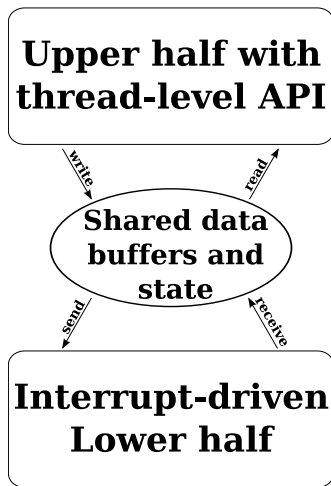
Outline

- 1 Design and Implementation
 - Critical Sections
 - Transactional System
- 2 Performance Results
 - Measurements
 - Testing Methodology
- 3 Summary and Future Work
 - Summary
 - Future Work

Outline

- 1 Design and Implementation
 - Critical Sections
 - Transactional System
- 2 Performance Results
 - Measurements
 - Testing Methodology
- 3 Summary and Future Work
 - Summary
 - Future Work

Device Driver Structure



Critical Sections

A *critical section* is a piece of code that accesses shared data or resources in the system and must not be accessed by two or more processes simultaneously.

Critical Sections

Properties

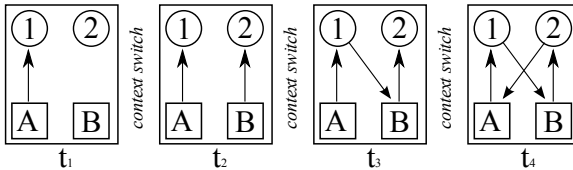
- *atomically*—perform indivisible operations “instantaneously.”
- *consistent*—no illegal system state will exist.
- *isolation*—no thread will see intermediate state.
- *durable*—no reversion after completion.

*These are known as the ACID properties.
We are only interested in the A, C, and I properties.*

Critical Sections

Problems

- Deadlock

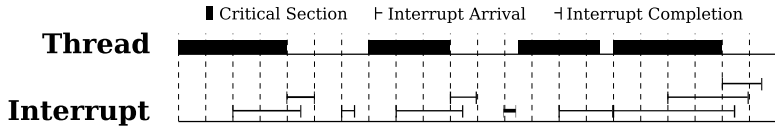


- Priority Inversion

Critical Sections

Jitter

- Variations in the amount of time the system takes to respond to incoming interrupts.



Critical Sections

Transactional Memory

- STM library assures that the ACI properties are followed.
- Compiler and library automatically handle rollbacks and commits.

Critical Sections

Interrupt Handling

- Interrupts are implicitly given highest priority in system.
- STM can be set up to give interrupt-driven transaction highest priority.
- User-level thread will not prevent interrupt from entering.
- I/O operations are difficult because they cannot be taken back.

Embedded Xinu

The Beginning

- Xinu was developed 25 years ago by Doug Comer.
- Provide a easy-to-understand O/S for teaching and research.
- Under 20,000 lines of code, but still a rich experimentation platform.
- lightweight thread model, shared memory space, preemptive multitasking priority scheduler, synchronization primitives, IPC, and device drivers.

Transactional Xinu

- Built on top of Embedded Xinu.
- Provides needed components for transactions.
 - POSIX-thread library.
 - Thread-local storage.
 - Intel's STM library.

POSIX-thread library

- Keep the Xinu model: lightweight.
- `pthread_key_create`
- `pthread_setspecific`
- `pthread_create`
- `pthread_join`

Interrupt-local Storage

- Thread-local storage gives private memory to every thread.
- Interrupts push state onto stack, does not update GS register.
- Interrupt-local storage switches GS context for interrupts.

Transactional Library

- Several modes of operation: optimistic, pessimistic, serial, and obstinate.
- Optimistic and pessimistic are “normal” modes.
- Serial mode works with legacy and irrevocable operations.
- Obstinate allows for “stubborn” transactions.

Transactional Library

Single Global Lock Atomicity

- Creates an equivalence between global lock code and atomic code.
- STM library actually uses SGLA for serialized transactions.

```
__tm_atomic {          wait (global_lock);  
    Statements;  →    Statements;  
}                    signal (global_lock);
```

Transactional Library

Obstinate Mode

- Library lets an obstinate transaction beat all conflicting transactions.
- Provides an interface allowing the programmer to declare an obstinate transaction.
- Transactional Xinu uses this in interrupt handlers.

Transactional Library

Versioning and Logging

- Read versioning tracks the version number of a variable.
- Writing will obtain a lock and increment the version number.
- Undo logging saves original values to private memory.

Transactional Library

Versioning Example

Reader Code

```
global integer our_var
local integer my_var

atomic
  if ( our_var = 1 )
    my_var = our_var + 1
  else
    my_var = 1
```

$\{\} \rightarrow \{our_var : 2\}$

Writer Code

```
global integer our_var

atomic
  our_var = our_var + 1
```

$\{\} \rightarrow \{our_var : 2\} \rightarrow \{our_var : 3\}$

Transactional Library

Two Phase Locking

- 2PL for contention manager.
- Maps every contended memory location to a unique lock.
- Mapping performed at runtime, takes ~ 4 MB of memory.
- Transactional Xinu minimizes size of atomic sections.

Transactional Device Drivers

- Wrap upper half critical sections in `tm_atomic`.
- *Wrap lower half shared data in* `tm_atomic`.
- Every function call and incidental function call must be re-instrumented.

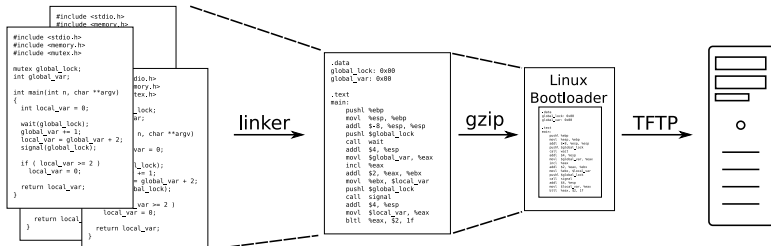
Code Size Differences

Kernel image	Non-STM	STM	Increase	%
raw	365,628	595,251	229,623	62.80%
stripped	351,048	540,504	189,456	53.97%
excluding STM library (170,869 bytes)				
raw	365,628	424,382	58,754	16.07%
stripped	351,048	369,635	18,587	5.29%

Transactional System

- Transactional Xinu was built to run on real hardware, available today.
- Built on mid-model Pentium 4, 3.0 GHz processor (“Northwood”).
- No BIOS calls or Linux code used behind-the-scenes.

Loading the Kernel



Measurements

- On-chip: read timestamp counter (`readtsc`).
- In-system: timer counters (thread time, monotonic time).
- External: min, avg, max, mdev of round-trip time.

Ping Testing

- Packet enters machine and raises interrupt (`RX_TSC`).
- Upper half call transfers incoming data (`READ_TSC`).
- Upper half call transfers outgoing data (`WRITE_TSC`).
- Final call in lower half of driver (`TX_TSC`).

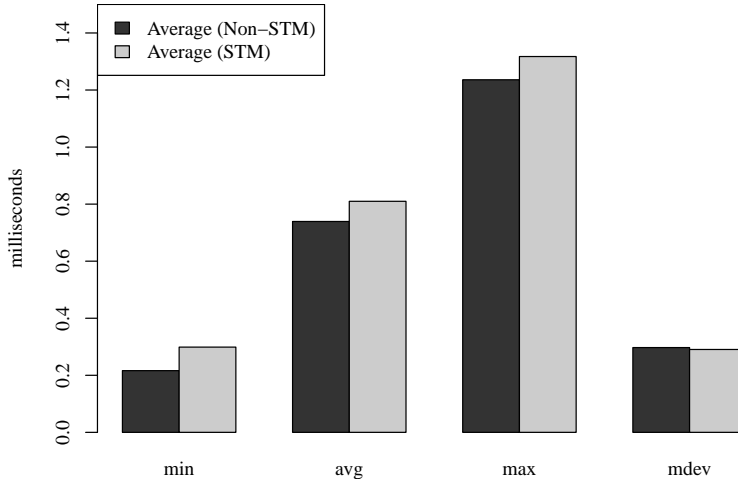
Ping Testing

Other notes

- Data gathering occurs at runtime, stores in memory until needed.
- Timer interrupt fires every $1/10$ of a millisecond, lightweight.

Ping Testing

Ping 1000 Millisecond Interval

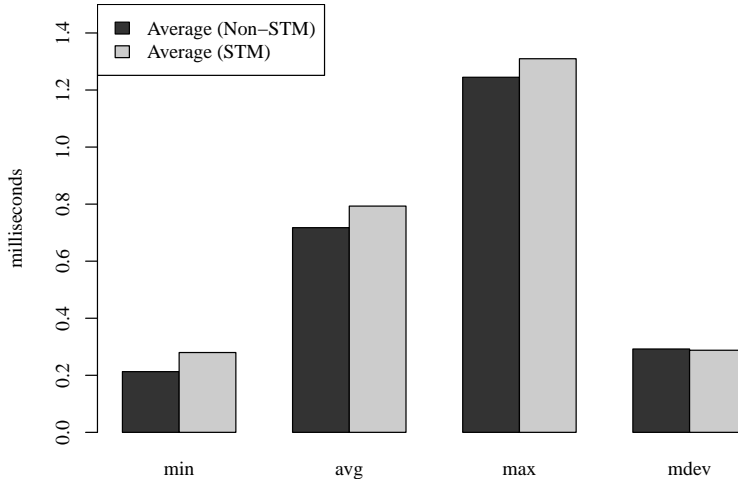


measure



Ping Testing

Ping 500 Millisecond Interval

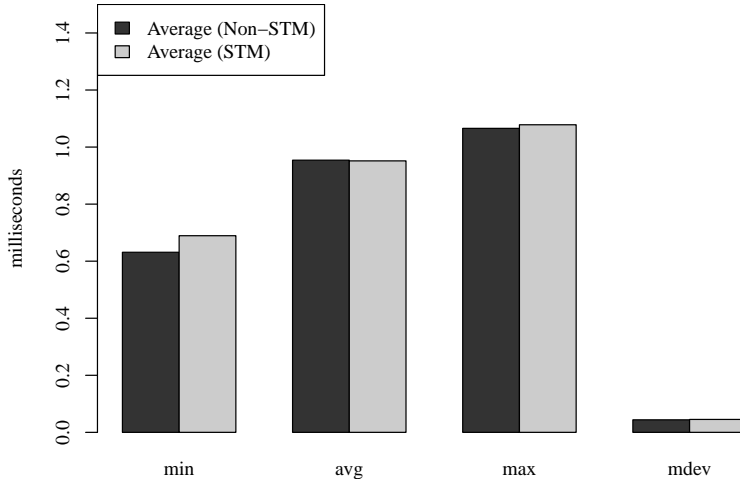


measure



Ping Testing

Ping Flood (0 Millisecond Interval)



measure



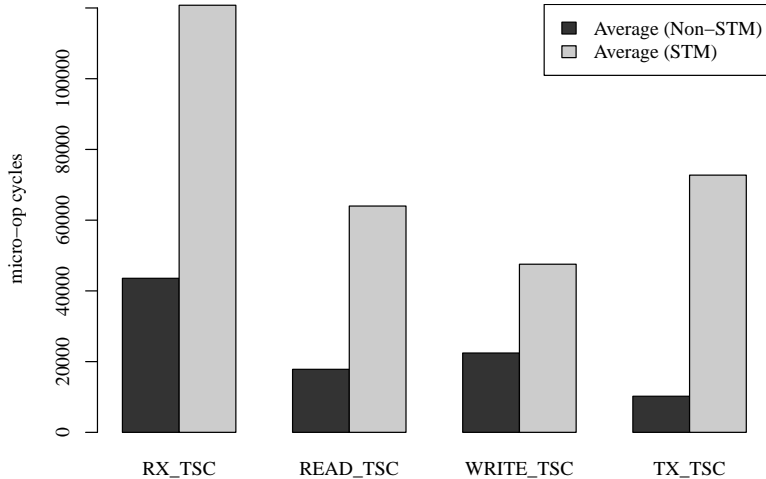
Ping Testing

Timestamp Counter

- Machine specific
- 100,000 micro-operations takes about 33 microseconds.

Ping Testing

TSC 1000 Millisecond Interval

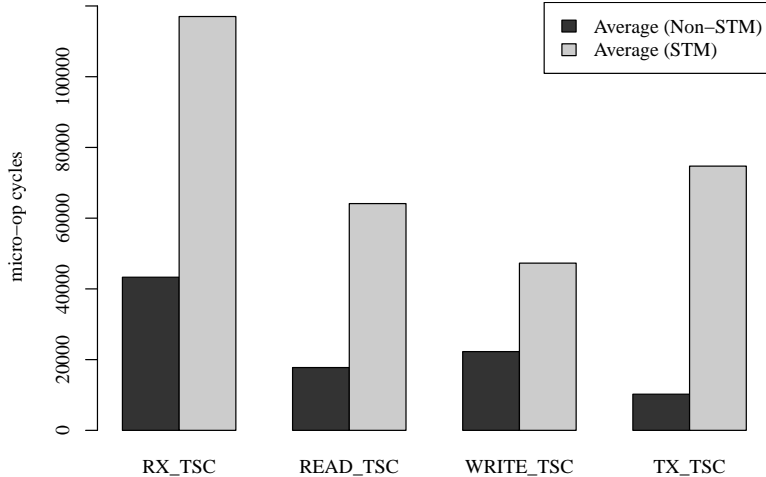


critical region



Ping Testing

TSC 500 Millisecond Interval

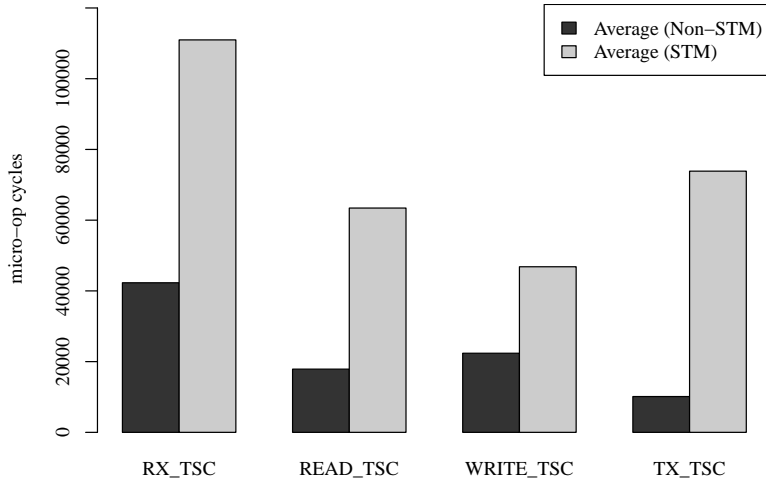


critical region



Ping Testing

TSC Flood (0 Millisecond Interval)



critical region



Timing Testing

Methodology

- Measuring jitter is difficult.
- Lightweight timestamp counter at beginning of handler.
- Packet generator, sending precisely timed packets.

Timing Testing

Results

	Average	Minimum	Maximum	Std. Dev.
Non-STM	30552911.53	30525638	30582946	13847.91
STM	30553291.98	30534954	30576856	13223.17
Difference	380.45	316	-6090	-624.74

Summary

- HTM systems are hard to build, use STM instead.
- Initial results suggest that TM might be able to reduce jitter.
- Transactional Xinu is built to use Intel's STM library and compiler.
 - Adds POSIX library and interrupt-local storage.
 - Specially instrumented interrupt handlers (`tm_atomic` and `tm_callable`)
- Experimentation shows software overhead is minimal in some scenarios.
 - Many other components still disable interrupts.

Future Work

- More tightly integrate STM library to Embedded kernel (scheduler, interrupts, etc.)
- Test scaling to multi-core systems (what TM was designed for).
- Test scaling with multiple network interfaces (ZigBee, WiFi, Bluetooth, GigE).
- Analyze real-time properties, if interrupts can always be received what happens (“interrupt overload”).

`mschul@mscs.mu.edu`

`http://www.mscs.mu.edu/~mschul/`