UNDERSTANDING TRANSACTIONAL MEMORY PERFORMANCE

Donald E. Porter and Emmett Witchel
Multicore is here

This laptop
2 Intel cores

Intel Single-chip Cloud Computer
48 cores

Tilera Tile GX
100 cores

- Only concurrent applications will perform better on new hardware
Concurrent programming is hard

- Locks are the state of the art
  - Correctness problems: deadlock, priority inversion, etc.
  - Scaling performance requires more complexity
- Transactional memory makes correctness easy
  - Trade correctness problems for performance problems
  - Key challenge: performance tuning transactions
- This work:
  - Develops a TM performance model and tool
  - Systems integration challenges for TM
Simple microbenchmark

lock();
if(rand() < threshold)
    shared_var = new_value;
unlock();

xbegin();
if(rand() < threshold)
    shared_var = new_value;
xend();

- **Intuition:**
  - Transactions execute optimistically
  - TM should scale at low contention threshold
  - Locks always execute serially
Ideal TM performance

- Performance win at low contention
- Higher contention degrades gracefully

```c
xbegin();
if(rand() < threshold)
    shared_var = new_value;
xBend();
```

Lower is better
Ideal, not real data
Actual performance under contention

Lower is better

Actual data

Comparable performance at modest contention

40% worse at 100% contention

```c
xbegin();
if(rand() < threshold)
    shared_var = new_value;
xend();
```
First attempt at microbenchmark

Execution Time (s) vs. Probability of Conflict (%)

- Locks 32 CPUs
- TM 32 CPUs

Lower is better

Approximate data

```c
xbegin();
if(rand() < threshold)
    shared_var = new_value;
xend();
```
Subtle sources of contention

```c
if(a < threshold)
    shared_var = new_value;
eax = shared_var;
if(edx < threshold)
    eax = new_value;
shared_var = eax;
```

- Compiler optimization to avoid branches
- Optimization causes 100% restart rate
- Can’t identify problem with source inspection + reason
Developers need TM tuning tools

- Transactional memory can perform pathologically
  - Contention
  - Poor integration with system components
  - HTM “best effort” not good enough

- Causes can be subtle and counterintuitive

- Syncchar: Model that predicts TM performance
  - Predicts poor performance \(\rightarrow\) remove contention
  - Predicts good performance + poor performance \(\rightarrow\) system issue
This talk

- Motivating example
- **Syncchar performance model**
- Experiences with transactional memory
  - Performance tuning case study
  - System integration challenges
The Syncchar model

- Approximate transaction performance model
- Intuition: scalability limited by serialized length of critical regions
- Introduce two key metrics for critical regions:
  - **Data Independence**: Likelihood executions do not conflict
  - **Conflict Density**: How many threads must execute serially to resolve a conflict
- Model inputs: samples critical region executions
  - Memory accesses and execution times
Data independence ($I_n$)

- Expected number of non-conflicting, concurrent executions of a critical region. Formally:
  \[ I_n = n - |C_n| \]
  \[ n = \text{thread count} \]
  \[ C_n = \text{set of conflicting critical region executions} \]

- Linear speedup when all critical regions are data independent ($I_n = n$)
  - Example: thread-private data structures

- Serialized execution when ($I_n = 0$)
  - Example: concurrent updates to a shared variable
Example:

- Same data independence (0)
- Different serialization
Intuition: Low density

High density

How many threads must be serialized to eliminate a conflict?

Similar to dependence density introduced by von Praun et al. [PPoPP ‘07]
Syncchar metrics in STAMP

- Conflict Density
- Data Independence

Higher is better
Predicting execution time

- Speedup limited by conflict density
- Amdahl’s law: Transaction speedup limited to time executing transactions concurrently

\[
\text{Execution Time} = \left( \frac{cs\_cycles}{\max(n, D_n)} \right) + \text{other}
\]

- \( cs\_cycles \) = time executing a critical region
- \( other \) = remaining execution time
- \( D_n \) = Conflict density
Syncchar tool

- Implemented as Simics machine simulator module
- Samples lock-based application behavior
- Predicts TM performance

Features:

- Identifies contention “hot spot” addresses
- Sorts by time spent in critical region
- Identifies potential asymmetric conflicts between transactions and non-transactional threads
Syncchar validation: microbenchmark

- Tracks trends, does not model pathologies
- Balances accuracy with generality

Lower is better
Syncchar validation: STAMP

- Coarse predictions track scaling trend
- Mean error 25%
- Additional benchmarks in paper
Syncchar summary

- Model: data independence and conflict density
  - Both contribute to transactional speedup
- Syncchar tool predicts scaling trends
  - Predicts poor performance \(\rightarrow\) remove contention
  - Predicts good performance + poor performance \(\rightarrow\) system issue
- Distinguishing high contention from system issues is key step in performance tuning
This talk

- Motivating example
- Syncchar performance model
- Experiences with transactional memory
  - Performance tuning case study
  - System integration challenges
TxLinux case study

- TxLinux – modifies Linux synchronization primitives to use hardware transactions [SOSP 2007]

16 CPUs – graph taken from SOSP talk

Lower is better
Bonnie++ pathology

- Simple execution profiling indicated ext3 file system journaling code was the culprit
- Code inspection yielded no clear culprit
- What information missing?
  - What variable causing the contention
  - What other code is contending with the transaction
- Syncchar tool showed:
  - Contended variable
  - High probability (88-92%) of asymmetric conflict
Bonnie++ pathology, explained

- False asymmetric conflicts for unrelated bits
- Tuned by moving state lock to dedicated cache line

```c
lock(buffer->state);
...
xbegin();
...
assert(locked(buffer->state));
...
xend();
...
unlock(buffer->state);
```

```c
struct
bufferhead
{
    ...
    bit state;
    bit dirty;
    bit free;
    ...
};
```
Tuned performance – 16 CPUs

- Tuned performance strictly dominates TxLinux

Lower is better

- Tuned performance strictly dominates TxLinux
This talk

- Motivating example
- Syncchar performance model
- Experiences with transactional memory
  - Performance tuning case study
  - System integration challenges
    - Compiler (motivation)
    - Architecture
    - Operating system
Some best effort HTM designs cannot handle TLB misses
- Sun Rock

What percent of STAMP txns would abort for TLB misses?
- 2% for kmeans
- 50-100%

How many times will these transactions restart?
- 3 (ssca2)
- 908 (bayes)

Practical HTM designs must handle TLB misses
Input size

- Simulation studies need scaled inputs
  - Simulating 1 second takes hours to weeks
- STAMP comes with parameters for real and simulated environments
Input size

Simulator inputs too small to amortize costs of scheduling threads
System calls – memory allocation

Thread 1

xbegin();
malloc();
xend();

Common case behavior:
Rollback of transaction rolls back heap bookkeeping
System calls – memory allocation

Uncommon case behavior:
Allocator adds pages to heap
Rolls back bookkeeping, leaking pages

Pathological memory leaks in STAMP genome and labyrinth benchmark
System integration issues

- Developers need tools to identify these subtle issues
  - Indicated by poor performance despite good predictions from Syncchar
- Pain for early adopters, important for designers
- System call support evolving in OS community
  - xCalls [Volos et al. – Eurosys 2009]
    - Userspace compensation built on transactional pause
  - TxOS [Porter et al. – SOSP 2009]
    - Kernel support for transactional system calls
Related work

- **TM performance models**
  - von Praun et al. [PPoPP ’07] – Dependence density

- **HTM conflict behavior**
  - Bobba et al. [ISCA 2007]
  - Ramadan et al. [MICRO 2008]
  - Pant and Byrd [ICS 2009]
  - Shiriraman and Dwarkadas [ICS 2009]
Conclusion

- Developers need tools for tuning TM performance
- Syncchar provides practical techniques
- Identified system integration challenges for TM

Code available at:
http://syncchar.code.csres.utexas.edu
porterde@cs.utexas.edu
Backup slides