Exploiting Distributed Software
Transactional Memory

Christos Kotselidis
Research Fellow
Advanced Processor Technologies Group
The University of Manchester
Outline

- Transactional Memory
- Distributed Transactional Memory
- DiSTM
  - Architecture
  - Protocols
- Evaluation
- Conclusions
Need for Concurrent Programming

- Multicores are mainstream: new software challenges
  - Exploit parallelism
  - Manage concurrency
- Locks are challenging for safe shared data access
- Problem is \textit{explicit synchronization}
  - Programmer manages shared accesses
  - Correctness: Race conditions, deadlocks, …
  - Performance/complexity: lock granularity (coarse/fine grain)
What is Transactional Memory? (1/2)

- New concurrent programming model, aims to:
  - Simplify programming compared to fine-grain locks
  - Provide similar or better performance than fine-grain locks
- Database transactions adapted for memory accesses
- Growing Research Area
  - 50+ TM implementations (last decade)
  - Software, Hardware, Hybrid Platforms (STM, HTM, HyTM)
  - Intel Haswell RTM, IBM Blue Gene/Q
  - Akka, PGAS languages, etc.
What is Transactional Memory? (2/2)

- Instead of acquiring locks, execute code optimistically
- Resolve detected conflicts
- Commit and publicize the changes
- Atomicity, Consistency, Isolation (ACI)

```
synchronized(this) {
    ...
    x++; 
}
```

```
atomic {
    ...
    x++; 
}
```

Programming with locks

Programming with transactions
TM research

Most TM systems target shared-memory architectures
• Hardware, Software, Hybrid

Concerning distributed computing:
• Partitioned Global Address Space (PGAS) languages (X10, Fortress) contain the atomic construct without currently having any underlying distributed TM system
• Distributed JVM domain for Enterprise Applications (Terracotta) use locks for synchronization
• Transactions have started being used in Distributed Systems (Sinfonia)
STM on CMPs

Thread 1

```
x = a;
x++;
```

Thread 2

```
a++;
```

![Diagram showing Threads, Memory Heap, Transactional Engine, and Transactional Coherence Protocols]
STM on Clusters
Distributed Software Transactional Memory DiSTM – Architecture (1/4)

- Clustered JVMs behaving as a Single System Image
- Modular and Pluggable architecture
- JVM middleware coordinating transactional execution
- Proactive Framework (RMI) for distributed communication
- @distatomic annotated interface denote transactional objects
DiSTM – Architecture (2/4)

- Automatic class re-writing (BCEL) in order to inject the transactional protocol within the objects
- Four distributed transactional coherence protocols
  - TCC, Single Lease, Multiple Leases, Anaconda
- Library of distributed atomic collection classes
  - Arrays, Singleton Objects, HashMaps, Linked Lists
@atomic
public interface AtomicInteger {
    public int getValue();
    public void setValue(int value);
}
DiSTM – Architecture (4/4)

- DiSTM’s single instance overview
DiSTM – Protocols

- DiSTM supports two modes of operation
  - Centralized mode: Data and coherency handled by the master node
    - Three protocols in centralized mode (TCC, Single Lease, Multiple Leases)
    - Two stage validation protocols (eager localValidation(), lazy remoteValidation())
  - Decentralized mode: Fully decentralized operation, data distribution
    - Data are partitioned amongst the nodes
    - Anaconda decentralized protocol
    - Unified validation procedure (lazy localValidation(), lazy remoteValidation())
DiSTM– Centralized Protocols (1/5)

TCC, Single Lease, Multiple Lease

- Data consistency
  - Master Node keeps a guaranteed consistent view of data
  - Worker Nodes keep cached working dataset
  - Upon a transaction’s `commit` the master node eagerly forces the worker nodes to update their working datasets

Commit stage is serialized thus blocking possible parallel commits.
DiSTM – TCC (2/5)

1) remoteValidate()
2) true/false
3) update global data
4) update cached data
DiSTM – Single lease (3/5)

1) acquire lease
2) true/false
3) update release lease
4) update assign lease
DiSTM – Multiple leases (4/5)

1) acquire lease validate

2) true/false

3) update release lease

4) reacquire lease on abort
DiSTM – Anaconda protocol

- Fully decentralized, 3-stage protocol
- Object caching and replication
- Enables parallel commit of transactions
- Library of distributed atomic collection classes
  - Arrays, Singleton Objects, HashMaps, Linked Lists
DiSTM – Anaconda protocol (2/3)

Three stage protocol:
1. Lock Acquisition: Acquire locks of objects
2. Validation: Validate against concurrently running transactions
3. Update Objects: Update objects with new values and Release locks
DiSTM – Anaconda protocol (3/3)
Evaluation

Benchmarks:
- Lee–TM (Classic PCB Routing Benchmark)
- Kmeans (Clustering algorithm)
- Glife–TM (Conway’s Automaton)

Hardware
- 4 nodes x 8 dual core Opterons, Open Suse, Sun HotSpot 1.6, Gigabit Ethernet

Experimental Setup:
- Each node utilizes 1 to 8 threads (* 4 nodes: min=4, max=32)
- We start by one thread per node and continue by incrementing by one
- Comparative evaluation of protocols
- Evaluation against industrial-strength Terracota clustering JVM
LeeTM

![Graph showing time (seconds) vs. number of threads for different concurrency control methods.](image-url)
LeeTM

LeeTM–ER

Aborts per commit

- TCC
- SL
- ML
- ANA

4 threads
8 threads
12 threads
16 threads
20 threads
24 threads
28 threads
32 threads
KMeans

- Anaconda High
- Anaconda Low
- TCC Low
- Serialization Lease Low
- Multiple Leases Low
- Terracotta
KMeans-Low

The bar chart represents the number of aborts per commit for different thread counts for TCC, SL, ML, and ANA. The x-axis represents the applications, and the y-axis represents the number of aborts per commit. The chart includes data for 4, 8, 12, 16, 20, 24, 28, and 32 threads.
GLife

![Graph showing the performance of GLife with different thread counts and configurations. The graph compares time (in seconds) for different numbers of threads (0 to 32). The graph includes lines for Anaconda, Terracotta Coarse, and Terracotta Medium configurations.]
# Categorization

<table>
<thead>
<tr>
<th>Transaction Length</th>
<th>Contention</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>Small</td>
<td>Anaconda (Glite)</td>
</tr>
<tr>
<td></td>
<td>Terracotta (Vacation)</td>
</tr>
<tr>
<td>Large</td>
<td>Anaconda (LeeTM-ER)</td>
</tr>
</tbody>
</table>
Bottlenecks – Future Work

- Network Optimizations
  - Immediate Services
  - Java Fast Sockets
- Garbage Collection
  - Tuning
  - Distributed GC
- Transactional Protocols
  - Multi-versioning (D2STM)
- Integration with Enterprise Servers
  - Real-time workloads
Conclusions

- JVM Clustering with Software TM
- Study of Distributed TM protocols
  - Centralized – TCC, SL, ML
  - Decentralized – Anaconda
- Performance influenced by:
  - Transaction abort rate
  - Computational intensity of applications
- Different winning protocol depending on workload
- Evaluation against state-of-the-art commercial lock based solution
Further Contributions

- Intel: Hardware/Software CPU Codesign
Further Contributions

Features

• CISC→ VLIW
• Dynamic Binary Translation and Optimizations
  • Load Hoisting, Code Versioning
• Targeting better power/performance
• Real time path profiling and optimizations
• Aggressive speculation and fail recovery
Further Contributions

Oracle

- Truffle/Graal (One VM to rule them all)
  - Abstract Syntax Tree Interpreter + Dynamic Compilation on top of the HotSpot VM
- Multi-language VM
  - JavaScript, Python, Ruby, R, etc.
- Compiler/Garbage Collection optimizations
  - Write Barrier Elision, Compressed Pointers
Research Opportunities within the APT Group

Christos Kotselidis
Research Fellow
Advanced Processor Technologies Group
The University of Manchester
History

- World’s first stored program computer (The Baby)
- Invention of virtual memory (Atlas)
- Manchester Dataflow Computer
- 2008 1st place in RAE
Advanced Processor Technologies Group

- Led by ICL Professor Steve Furber
  - Designer of the ARM processors (BBC Micro, Acorn)

- Diverse research agenda
  - Spinnaker (one of the few academic institutions fabricating chips)
  - Computer Architecture
  - Systems
  - Compilers and Managed Runtimes
**Advanced Processor Technologies Group**

- **Major Spinoffs**
  - ICL Goldrush database server
  - Amulet processors (Low power)
  - Transitive (Rosseta software, acquired by IBM)
  - Silistix (Network-on-Chip)

- **Career opportunities**
  - ARM, Oracle, Intel, Google, Imagination Technologies, etc.
Advanced Processor Technologies Group

- Current Projects
  - SpiNNaker: A universal Spiking Neural Network Architecture
  - Teraflux: Research in Many-core (Software and Hardware) following Data Driven Task model
  - AXLE: Big Data Analytics Acceleration
  - AnyScale Apps

Further info at: http://apt.cs.man.ac.uk/
Advanced Processor Technologies Group

New Initiatives

- **Pamela**: A Panoramic Approach to the Many-CorE LAndscape
  
  We focus on hardware/software codesign for heterogeneous many-core systems for computer vision and data-centers with emphasis on novel virtualization techniques.

- **DOM**: Delaying and Overcoming Microprocessor Errors
  
  We focus on hardware/software codesign using virtualization (Managed Runtime Environments) for delaying and detecting microprocessor errors.
Advanced Processor Technologies Group

Funding

- Industrial Funding Positions by ARM (3 years): Deadline ASAP
- Center of Doctoral Training—CDT Positions (4 years): Deadline: As early as possible

http://www.cs.manchester.ac.uk/phd/programmes/cdt/
http://www.mdc.manchester.ac.uk/funding/pdsaward/
http://www.cs.manchester.ac.uk/study/postgraduate-research/programmes/phd/funding/school-studentships/
http://www.cs.manchester.ac.uk/phd/

Contacts:
Mikel Lujan (mikel.lujan@manchester.ac.uk)
Christos Kotselidis (christos.kotselidis@manchester.ac.uk)