Evaluation of Blue Gene/Q Hardware Support for Transactional Memory

Amy Wang, Matthew Gaudet\textsuperscript{1,3}
Christopher Barton, Raul Silvera

Peng Wu, Martin Ohmacht
Maged M. Michael

IBM Toronto Lab
IBM Research

Jose Nelson Amaral
University of Alberta\textsuperscript{3}
Edmonton, AB, Canada
Transactional Memory (TM)

- Critical sections

```plaintext
critical section
{
    a[b[i]] += c[i];
}
```

- Different implementations
  - Locks
  - Transactional memory

- Transactional Memory in a Nutshell
  - Optimistically grant everyone access to a critical section
  - But continuously monitor reads and writes of all concurrent transactions
  - Upon detection of memory conflicts or irreversible actions, abort the offending transactions

The Curve of Innovation in TM

- 1990 Herlihy & Moss
- 2000 ~ 2010 boom in (S)TM research
- Emergence of commercial HTMs (Azul, Rock, BG/Q, Z, Haswell)
- Understand the system
- Evaluating existing applications
Overview of BG/Q HTM

- Basic architecture
  - Each compute chip has 16 in-order cores, each w/ 4 hardware threads
  - L1 cache is local to each core
  - L2 cache is the point of coherence, shared by 16 cores

- BG/Q TM architecture
  - Book-keeping for speculative states is done in L2
    - Up to 15 ways of each directory set can be used for speculative data
    - Up to 30MB speculative space
  - Conflict detection granularity
    - Write: 8B granularity
    - Read: 8B~>64B granularity
  - Each transaction applies for a SPEC-ID, managed by hardware
  - No hardware support for register check pointing

- The short- and long-running TM modes
  - short-running mode: L1 is bypassed during transactional execution
  - long-running mode: L1 is flushed when entering a transaction
From Best-effort HTM to TM Programming Model

Critical section
#pragma tm_atomic {
   // any code
}

Translating to lower-level API
- register check-pointing
- syntactical translation

Runtime support for HTM
- Software setup/config for HTM
- Sandboxing and detection of HTM failure
- Ensure forward progress

Causes of HTM abort
- Transactional conflicts
- Capacity overflow and way-conflict
- Jail mode violation

Programming interface

XL compiler
TM runtime
Kernel

Best-effort HTM
Evaluation of BG/Q TM

1. Identify factors that impact the performance of BG/Q HTM

2. Evaluate the STAMP benchmarks
   - Single-thread performance (over sequential code)
   - Scalability (over sequential code)
   - Compare against lock and STM
Single-thread TM Overhead

- On average 17% single-thread TM overhead over sequential code
  - additional L1 misses (short vs. long)
  - TM enter/exit overhead
  - Capacity overflow
Single-thread BG/Q TM vs TinySTM

- Single-thread overhead of BG/Q TM is significantly less than that of STM
- STM version of STAMP is manually instrumented and heavily privatized

avg. STM tranx in bayes has <30 words
Effective HTM (Genome and Vacation)
ssca2

- **Lock**
  - scale up to 4 threads
  - Lock granularity: 16% of parallel region is critical section (1-thread)

- **BG/Q TM**
  - 0 abort ratio, but only scale to 4 threads
  - running out of spec-IDs due to frequent short transactions

- **STM**
  - scales to 64 threads
  - because of 0 abort ratio and low STM overhead
kmeans

- **Lock**
  - Scales to 16 threads
  - Lock granularity: 5% of parallel region is spent in critical section (1-thread)

- **BG/Q TM** scales to 8 threads

- **STM** scales to 32 threads
  - But single-thread overhead is significant

### Chart: kmeans_high

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Serialization ratio (# threads)</th>
<th>Abort ratio (# threads)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>kmeans_low</td>
<td>0 %</td>
<td>0 %</td>
</tr>
<tr>
<td>kmeans_high</td>
<td>0 %</td>
<td>0 %</td>
</tr>
</tbody>
</table>

High-abort ratio results in high serialization ratio
Intruder

- Lock: scale up to 4 threads (1.5X speedups)
  - Lock granularity: 66% of parallel region is critical section (1-thread)

- BG/Q TM: scales to 8 threads, but drops sharply after 8

- STM: scales to 16 threads
  - read-/write-set: < 30 words

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Serialization ratio (# threads)</th>
</tr>
</thead>
<tbody>
<tr>
<td>intruder</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Abort ratio (# threads)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>0%</td>
</tr>
</tbody>
</table>
Scalability Map of Lock, HTM, STM

- Lock
- STM w/ privatization
- Capacity overflow (STM w/ privatization scales)
- Effective HTM
- HTM better at low thread-count (High serialization at high thread-count)
- Mem conflicts (lock scales)
- Kmeans
- Spec-ID bottleneck (STM scales)
- Ssca
- Intruder
- STM w/ privatization
- STM/HTM

Granularity = critical section/parallel region

- Fine-grained: < 1/16
- Medium-grained: < 1/4
- Coarse-grained: > 1/4
Main Findings of BG/Q TM Evaluation

- Surprise factors in BG/Q TM performance
  - short- vs. long-running mode
  - cache locality penalty
  - SPEC-ID contention bottleneck
  - capacity overflow/way-conflicts are insignificant for most STAMP benchmarks
  - retry policies matter

- Single-thread BG/Q TM overhead is much smaller than STM’s, but still noticeable compared to lock’s

- When to use locks, BGQ TM, and STM?
  - Sweet spot of BG/Q TM are medium to long transactions where most memory conflicts are transient (e.g., vacation and genome)
  - When locks scale well, use locks (e.g., kmeans)
  - STM v/ heavy privatization may provide scalability for very long transactions where HTM cannot (e.g., labyrinth); STM may also benefit very short transactions (e.g., scca)

- The simplicity of BG/Q TM programming model is a clear win (over STM)
Conclusion & Future Work

- Understanding the performance behavior of a real HTM system is a non-trivial task, and an essential first step to any future exploitation on a real HTM

- Our methodology to understand the performance of HTM system
  1. Understand critical section characteristics of your TM applications
     • relative critical section size: whether big lock can scale well
     • absolute critical section size: indicator of transactional footprint
  
  2. Evaluate single-thread TM performance overhead
     • measure all aspects of system performance to identify performance factors (CPI, instruction path length, HPM events, TM-specific statistics)

  3. Evaluate TM scalability
     • measure abort ratio and serialization ratio
     • compare against lock and (if feasible) STM implementations
Cache Hierarchy Configurations

- Each compute chip with 16 cores for user applications
  - A2 In-order processor core
  - Each core runs 4 way SMT threads
- Specification on the cache structure
  - L1 is local to the core
  - L2 is the point of coherence, shared by 16 cores

<table>
<thead>
<tr>
<th></th>
<th>Total size</th>
<th>Cache line size</th>
<th>Associativity</th>
<th>Private to the core</th>
<th>Latency on load</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>16k</td>
<td>64 byte</td>
<td>8-way set associative</td>
<td>Y</td>
<td>6 cycles</td>
</tr>
<tr>
<td>L1P</td>
<td>4k</td>
<td>N/A</td>
<td>N/A</td>
<td>Y</td>
<td>30 cycles</td>
</tr>
<tr>
<td>L2</td>
<td>32 M</td>
<td>128 byte</td>
<td>16-way set associative</td>
<td>N</td>
<td>60 cycles</td>
</tr>
</tbody>
</table>
Capacity Overflow

- Bayes (1-thread): 3% tranx are aborted, all due to capacity overflow
- Labyrinth (1-thread): 50% tranx are aborted, all due to capacity overflow
Labyrinth uses very long transactions (>100M cycles/per transaction)
  – always cause capacity overflow
  – No scaling on HTM

The STM version is heavily privatized through manual instrumentation
  – Read/write-set: <200 word
BG/Q HTM on yada

- Critical section characteristics:
  - Relative size: 99% of parallel region
  - Absolute size: 60K cycles

- Lock implementation
  - no scalability
  - explanation: relative critical section size is 99%

- STM: scales to 16 threads
  - read-/write-sets (<50 words)

- BG/Q TM
  - scales to 8 threads, but deteriorate sharply after 8 threads
  - explanation: sharp increase of serialization ratio beyond 8 threads

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Serialization ratio (# threads)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>yada</td>
<td>0 %</td>
</tr>
</tbody>
</table>
BG/Q HTM on bayes

- Critical section characteristics:
  - Relative size: 99% of parallel region
  - Absolute size: 120M cycles

- Lock implementation
  - No scaling except for thread 4

- STM: scales to 64 threads
  - Read/write-set (<30 words)

- BG/Q TM
  - Scales to 64 threads, upto 4X speedups
  - Explanation: significant serialization

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Serialization ratio (# threads)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>bayes</td>
<td>3 %</td>
</tr>
</tbody>
</table>
Performance Considerations of HTM Implementation Details

TM Execution Overhead

- **Loss of cache locality**
  - Long-running mode: L1 cache is flushed upon entering transaction
  - Short-running mode: L1 cache is bypassed during transaction execution

- **Spec-ID allocation contention**
  - Each transaction must apply for a spec-ID first
  - There is only 128 spec-ID, only recycled periodically

- **Transaction enter and exit overhead**
  - Save/restore of register contexts
  - TM runtime overhead

Source of HTM abort

- **Transactional conflicts**
  - Conflict detection granularity 8 ~ >64 bytes

- **Capacity overflow and way-conflict**
  - Default 10/16 speculative ways in a set guaranteed without eviction.
  - Can be triggered by exhausting the way limitation in a set

- **Jail mode violation (JMV)**
  - Irrevocable action detected by the kernel, e.g., write to device I/O address space