Concurrent GC
Leveraging Transactional Memory

Phil McGachey  Purdue University
Ali-Reza Adl-Tabatabai  Intel
Richard L. Hudson  Intel
Vijay Menon  Intel
Bratin Saha  Intel
Tatiana Shpeisman  Intel
Introduction

- Multi-cored desktops now standard
- Look to simplify concurrent programming
- Two emergent technologies
  - Transactional memory
  - Concurrent garbage collection
- We exploit the overlap in mechanisms
Our Goals

- Leverage transactions infrastructure for GC
- Target desktop applications
  - Games, multimedia, VOIP
- Aim to keep 90% of pauses under 1ms
  - Competitive with modern real-time GCs
Copying GC
Copying GC

Java Heap
Copying GC

Java Heap
Copying GC

Java Heap
Copying GC

Java Heap
Copying GC

Java Heap
Copying GC

Java Heap

From Space  To Space
Copying GC

Java Heap

From Space          To Space

Thread Stacks
Copying GC

Java Heap

From Space

To Space

Thread Stacks
Copying GC

Java Heap

From Space — To Space

Thread Stacks
Copying GC

Java Heap

From Space

To Space

Thread Stacks
Copying GC

Java Heap

From Space

To Space

Thread Stacks
Copying GC

Java Heap

From Space

To Space

Thread Stacks
Multi-Cored GC

- Most current desktop GCs “Stop the World”
- Becomes unfeasible with many cores
  - Overhead of pausing the threads
  - Bottlenecks in the GC code
  - Large heaps increase GC time
- Concurrent GC requires synchronization
Concurrent Copying

GC Thread

Application Thread
Concurrent Copying

GC Thread

Begin Copy

Application Thread
Concurrent Copying

GC Thread
Begin Copy
Copy Field A

Application Thread
Concurrent Copying

GC Thread
- Begin Copy
- Copy Field A
- Copy Field B

Application Thread
Concurrent Copying

GC Thread
- Begin Copy
- Copy Field A
- Copy Field B
- Copy Field C

Application Thread
- Write to Field A
Concurrent Copying

GC Thread
- Begin Copy
- Copy Field A
- Copy Field B
- Copy Field C
- Write Forwarding Ptr

Application Thread
- Write to Field A
Concurrent Copying

GC Thread
- Begin Copy
- Copy Field A
- Copy Field B
- Copy Field C
- Write Forwarding Ptr

Application Thread
- Write to Field A
- Read from Field A
Concurrent copying must be atomic
Implementing Atomicity
Implementing Atomicity

Object A
Implementing Atomicity

Object A

| Version Number |
Implementing Atomicity

Object A

Version Number

Tx1: Read
Implementing Atomicity

Object A

Version Number

Tx2: Write
Implementing Atomicity

Object A

Tx2 Lock

Tx2: Write
Implementing Atomicity

Object A

Tx2 Lock

Tx2: Write

Tx3: Read
Implementing Atomicity

Object A

Tx2 Lock

Tx2: Write
Implementing Atomicity

Object A

Tx2 Lock

Tx2: Validate
Implementing Atomicity

Object A

<table>
<thead>
<tr>
<th>Version Number ++</th>
</tr>
</thead>
</table>

Tx2: Commit
Implementing Atomicity

Object A

Tx1: Validate

| Version Number ++ |
Implementing Atomicity

Object A

Tx1: Abort

| Version Number ++ |
Transactional Integrity

- Two forms of transactional memory
- Weak atomicity
  - Programmer marks transactional accesses
  - Atomic / non-atomic conflicts possible
- Strong atomicity
  - System ensures that all accesses are atomic
- We build upon strong atomicity
Transactional Copying

- Concurrent copying must be atomic
- Wrap each copy operation in a transaction?
  - No lost updates due to strong atomicity
  - Each copy operation incurs overhead
- Perform all copies as one large transaction?
  - Increased probability of conflicts
- We leverage the underlying infrastructure
Synergy Part 1

- Shared mechanisms
- GC must observe modifications to objects
- TM must detect conflicts
- We leverage the overlap
- Treat “to-space” objects as speculative
The GC Algorithm

- Don’t stop the world
- Threads paused one at a time
- Minimize work during each pause
- Work split over three phases
- Copy a portion of the heap per GC cycle
- Use TM infrastructure for copying
Pauses

- Phase changes
  - All threads must be aware of phase change
  - Pause threads to synchronize memory
- Mark and Flip phases
  - Pause each thread to scan stack
  - Pause to ensure all references processed
Transactional Copying

GC Thread

Store version #

Application Thread
Transactional Copying

GC Thread

- Store version #
- Copy Field A

Application Thread
Transactional Copying

GC Thread
- Store version #
- Copy Field A
- Copy Field B

Application Thread
Transactional Copying

GC Thread
- Store version #
- Copy Field A
- Copy Field B
- Copy Field C

Application Thread
- Write to Field A
  - Increment Version #
Transactional Copying

GC Thread
- Store version #
- Copy Field A
- Copy Field B
- Copy Field C
- Compare version #

Application Thread
- Write to Field A
Transactional Copying

GC Thread
- Store version #
- Copy Field A
- Copy Field B
- Copy Field C
- Compare version #

Application Thread
- Write to Field A
- Read from Field A
Synergy Part 2

- Both systems use read and write barriers
- Code inserted around memory accesses
- Strong atomicity barriers:
  - Log transactional reads and writes
- Concurrent GC barriers
  - Prevents writes of unmarked pointers
  - Follow forwarding pointers
Experiments

- SPEC JVM98
- SPECjbb2000
- Atomicjbb
- AtomicTSP, AtomicOO7
AtomicTSP
All workloads
Conclusion

- Identified overlap between GC and TM
- Leverage TM to safely copy concurrently
- Focus on pause times
  - Aim to keep 90% < 1ms
  - Result: 98.8% < 1ms, 96.9% < 0.5ms
## Outliers

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>&lt; 1ms</th>
<th>1..10 ms</th>
<th>10...100 ms</th>
<th>&gt; 100 ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>201_compress</td>
<td>100.0%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>202_jess</td>
<td>100.0%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>209_db</td>
<td>100.0%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>213_javac</td>
<td>99.43%</td>
<td>0.57%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>222_mpegaudio</td>
<td>100.0%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>227_mtrt</td>
<td>100.0%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>_228_jack</td>
<td>100.0%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>SPECjbb</td>
<td>99.72%</td>
<td>0.14%</td>
<td>0.14%</td>
<td>0.00%</td>
</tr>
<tr>
<td>AtomicOO7</td>
<td>100.0%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>AtomicTSP</td>
<td>100.0%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Atomicjbb</td>
<td>85.00%</td>
<td>12.50%</td>
<td>2.14%</td>
<td>0.36%</td>
</tr>
<tr>
<td>Total</td>
<td>98.92%</td>
<td>0.85%</td>
<td>0.21%</td>
<td>0.02%</td>
</tr>
<tr>
<td>Target</td>
<td>≥ 90%</td>
<td>≤ 9%</td>
<td>≤ 0.9%</td>
<td>≤ 0.1%</td>
</tr>
</tbody>
</table>
## Pauses per GC

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Mark Phase</th>
<th>Flip Phase</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>201_compress</td>
<td>2.0</td>
<td>2.0</td>
<td>4.0</td>
</tr>
<tr>
<td>202_jess</td>
<td>2.6</td>
<td>2.0</td>
<td>4.6</td>
</tr>
<tr>
<td>209_db</td>
<td>2.0</td>
<td>2.0</td>
<td>4.0</td>
</tr>
<tr>
<td>213_javac</td>
<td>2.7</td>
<td>2.0</td>
<td>4.7</td>
</tr>
<tr>
<td>222_mpegaudio</td>
<td>2.0</td>
<td>2.0</td>
<td>4.0</td>
</tr>
<tr>
<td>227_mtrt</td>
<td>3.7</td>
<td>2.9</td>
<td>6.6</td>
</tr>
<tr>
<td>_228_jack</td>
<td>2.1</td>
<td>2.0</td>
<td>4.1</td>
</tr>
<tr>
<td>SPECjbb</td>
<td>5.7</td>
<td>2.7</td>
<td>8.4</td>
</tr>
<tr>
<td>AtomicOO7</td>
<td>3.6</td>
<td>2.0</td>
<td>5.6</td>
</tr>
<tr>
<td>AtomicTSP</td>
<td>2.0</td>
<td>2.0</td>
<td>4.0</td>
</tr>
<tr>
<td>Atomicjbb</td>
<td>4.0</td>
<td>2.0</td>
<td>6.0</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>2.9</strong></td>
<td><strong>2.1</strong></td>
<td><strong>5.1</strong></td>
</tr>
</tbody>
</table>
## Time In Each Stage

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Mark Phase</th>
<th>Copy Phase</th>
<th>Flip Phase</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>201_compress</td>
<td>0.19%</td>
<td>0.08%</td>
<td>0.26%</td>
<td>0.53%</td>
</tr>
<tr>
<td>202_jess</td>
<td>0.91%</td>
<td>0.37%</td>
<td>1.18%</td>
<td>2.45%</td>
</tr>
<tr>
<td>209_db</td>
<td>0.43%</td>
<td>0.14%</td>
<td>0.44%</td>
<td>1.00%</td>
</tr>
<tr>
<td>213_javac</td>
<td>0.82%</td>
<td>0.17%</td>
<td>0.82%</td>
<td>1.81%</td>
</tr>
<tr>
<td>222_mpegaudio</td>
<td>0.22%</td>
<td>0.08%</td>
<td>0.27%</td>
<td>0.57%</td>
</tr>
<tr>
<td>227_mtrt</td>
<td>1.81%</td>
<td>0.61%</td>
<td>2.02%</td>
<td>4.44%</td>
</tr>
<tr>
<td>_228_jack</td>
<td>0.77%</td>
<td>0.36%</td>
<td>0.58%</td>
<td>1.71%</td>
</tr>
<tr>
<td>SPECjbb</td>
<td>1.27%</td>
<td>0.27%</td>
<td>1.02%</td>
<td>2.56%</td>
</tr>
<tr>
<td>AtomicOO7</td>
<td>0.04%</td>
<td>0.01%</td>
<td>0.03%</td>
<td>0.08%</td>
</tr>
<tr>
<td>AtomicTSP</td>
<td>0.51%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.51%</td>
</tr>
<tr>
<td>Atomicjbb</td>
<td>1.37%</td>
<td>0.52%</td>
<td>2.39%</td>
<td>4.28%</td>
</tr>
<tr>
<td>Average</td>
<td>0.76%</td>
<td>0.24%</td>
<td>0.82%</td>
<td>1.81%</td>
</tr>
</tbody>
</table>