Transactional Memory Support for Scalable and Transparent Parallelization of Multiplayer Games

Daniel Lupei, Bogdan Simion
Don Pinto, Mihai Burcea
Matthew Misler, William Krick
Cristiana Amza

University of Toronto
Multiplayer games

- More than 100k concurrent players

Game server is the bottleneck
State-of-the-art

• Previous parallelizations of Quake
  – Lock-based [Abdelkhalek et. al ‘04] shows that false sharing is a challenge
  – Zyulkyarov et. al ’09
  – Gajinov et. al ’09
Game interactions

Game map

Action bounding box
Collision detection

Game map

Action bounding box
Conflicting player actions

Game map

T1

T2

Need for synchronization
Player actions

Compound action:
- move, charge
  weapon and shoot

Requirement:
consistency and atomicity
of whole game action
Conservative locking

Conservatively acquire all locks at beginning of action

Problem 1: Unnecessarily long conflict duration
Conservative locking

Conservative estimate of impact range at beginning of action

Problem 2: Unnecessarily high number of locked objects

Estimated impact radius
Fine-grained locking alternative?

GAME ACTION

- Lock 1
- Subaction 1
- Unlock 1
- Lock 2
- Subaction 2
- Unlock 2
- Lock 3
- Subaction 3
- Unlock 3

Problem:
- No atomicity for whole action

Not possible!
Fine-grained locking alternative?

Not possible!

Problem:
- Deadlocks

GAME ACTION

Lock 1
Subaction 1

Lock 2
Subaction 2

Lock 3
Subaction 3

Unlock 1, 2, 3
Software Transactional Memory

- Alternative parallelization paradigm
  - Implement game actions as transactions
  - Track accesses to shared and private data
  - Conflict detection and resolution

- Automatic *consistency and atomicity*
  - Transaction commits if no conflict
  - Transaction rolls back if conflict occurs
STM - Synchronization

BEGIN Transaction

Subaction 1

Subaction 2

Subaction 3

COMMIT Transaction

Problems solved:
- Deadlocks
- Atomicity
Handled automatically
STM - Synchronization

Estimated impact radius
STM - Synchronization

Collision detection optimized:

- split action into subactions
- perform collision detection gradually for each subaction
Transactional Memory vs. Locks

• Advantages of STM
  – Simpler programming task
  – Transparently ensures correct execution (deadlock problems and atomicity)

• Disadvantages
  – Software (STM) access tracking overheads

Never before shown to be competitive with lock synchronization for real applications
Contributions

• Case study of parallelization for games
  – synthetic version of Quake (SynQuake)
• We compare 2 approaches:
  – lock-based and STM parallelization
• We showcase the first application where
  STM outperforms locks 😊
Outline

• Application environment: SynQuake game
  – Data structures, server architecture
• Parallelization issues
  – False sharing
  – Load balancing (true sharing)
• Experimental results
Environment: SynQuake game

• Same as Quake:
  – Gameplay
    • entities
    • interactions
  – Data structures
  – Server design
Environment: SynQuake game

- Different from Quake
  - 2D maps
  - World physics

- Facilitates workload generation
  - Game map
  - Bots
  - Quests
Game map representation

- Fast retrieval of game objects
- Quake spatial data structure: **Areanode Tree**
Areanode tree

Game map

Areanode tree

Root node
Areanode tree
Areanode tree
Areanode tree

Game map

Areanode tree
Server frame

Spawn threads

Parallelization: request processing

Client requests

Barrier

Receive & Process Requests

Barrier

Admin (single thread)

Barrier

Form & Send Replies

Barrier

Client updates
Outline

• Application environment: SynQuake game
• Parallelization issues
  – False sharing
  – Load balancing
• Experimental results
Action bounding box

Move range

Shoot range
False sharing

- Action bounding box with TM
- Action bounding box with locks
- Move range
- Shoot range
Synchronization algorithm: Locks

Areanode tree

Top-view of world

Overlapping regions

Leaf locking: True Sharing
Synchronization algorithm: Locks

Object lists

Non-Overlapping regions

Parent node locking: False Sharing
Outline

• Application environment: SynQuake game
• Parallelization issues
  – False sharing
  – Load balancing
• Experimental results
Load balancing tradeoff

Cross-border conflicts (true sharing) => synchronization

- Good load distribution
- High synchronization
- Bad load distribution
- Low synchronization
Locality-aware load balancing

• Dynamically detect player hotspots and adjust workload assignments

• Compromise between load balancing and reducing synchronization
Dynamic locality-aware LB

Game map

Graph representation
Dynamic locality-aware LB

Game map

Graph representation
Experimental results

• Test scenarios: 1 – 8 quests, short/long range actions

• Performance comparison
  – Locks vs. STM scaling and performance
  – Influence of load balancing on scaling

• In the paper:
  – Varying the access tracking granularity for STM
Quest scenario: high contention

\[ X \]

\[ Y \]

\[ \times - \text{Quest 1} \]
Quest scenario: medium contention

- Quest 1
- Quest 2
- Quest 3
- Quest 4
STM scales better in all 3 contention scenarios
Processing times

Medium contention
Baseline load balancing policies

**Round-robin**

**Spread**

- **Thread 1**: +
- **Thread 2**: *
- **Thread 3**: -
- **Thread 4**: x
Load balancing

**Locks**
- locality-aware
- spread
- round robin

**STM**
- locality-aware
- spread
- round robin
Conclusions

• First application where STM outperforms locks:
  – Overall performance of STM is better at 2, 4, 8 threads in all scenarios

• STM eliminates false sharing through on-the-fly collision detection
  – Unlocks the potential of using locality-aware load balancing to reduce true sharing
SynQuake vs. Quake

- SynQuake - thorough evaluation of tradeoffs
- Quake
  - More complex graphics
  - More world physics computation
- More physics computation → STM overhead becomes negligible
- Performance results expected to hold for complex 3D games
Thank you!
STM: access tracking granularity
Processing times
Load balancing
- low false sharing -
LibTM

- LibTM: goal of providing high flexibility
  - Concurrency control
  - Access tracking granularity

- Widespread reliability problems among existing TM systems available at the time
  - e.g. Memory management limitations
  - Dragojevic ’08 – “Dividing transactional memories by zero” – DSTM2, RSTM, TL2, TinySTM
LibTM statistics

- Locality-aware load balancing
- Over 2 million transactions

<table>
<thead>
<tr>
<th>Contention Level</th>
<th>No. threads</th>
<th>Abort rate</th>
<th>Write ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>1</td>
<td>0%</td>
<td>22.74%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1%</td>
<td>22.70%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2%</td>
<td>22.65%</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>6%</td>
<td>22.56%</td>
</tr>
<tr>
<td>Medium</td>
<td>1</td>
<td>0%</td>
<td>4.65%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1%</td>
<td>4.80%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2%</td>
<td>4.76%</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>3%</td>
<td>4.77%</td>
</tr>
<tr>
<td>High</td>
<td>1</td>
<td>0%</td>
<td>0.62%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1%</td>
<td>0.62%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2%</td>
<td>0.63%</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2%</td>
<td>0.63%</td>
</tr>
</tbody>
</table>