KILO Transactional Memory for GPU

Wilson W. L. Fung  Inderpeet Singh  Andrew Brownswor
d*  Tor M. Aamodt
University of British Columbia, Canada  Intel Corp.

**Transaction Memory on GPU?**

- Motivation: Exploit irregular parallelism on GPUs
  - Widen GPU application scope
  - Enable more work-efficient algorithms
  - Expose more parallelism to hardware
  - Requires fine-grained data synchronizations to guard against potential data races

- Synchronization primitives on current GPUs
  - Thread Block Barrier, 32-bit/64-bit Atomic Operations
  - Atomic Ops + Memory Fence = Locks for larger structures
  - Fine-grained locks needed for good parallelism

- Debugging deadlocks with fine-grained locks is hard
  - Consideration of all possible interaction between locks

- Application Developer need to determine which of these states are deadlocks

**Transitional Memory [1]**

- Program specifies atomic code blocks called transactions
- Transaction are executed as if runs in isolation
- Simplify correctness burden on programmer

- Lock Version:
  - \(# Locks\) = \(# Possible Global Lock States\)

- Transaction-Aware SIMT Stack

  - SIMT stack serializes a warp @ CF divergence
  - Special entries for TX to handle transaction aborts:

```cpp
#define _begin() 
#define _commit() 
#define _tcommit() 
```

**Performance Results**

- Modeled KILO TM on GPGPU-Sim v3.0
- Evaluated on enhanced CUDA and OpenCL applications

**Transactions in parallel**

**Conflict Flow Divergence:**
- Transaction aborts may cause a warp to diverge

**Scalable Conflict Detection:**
- 1000s of concurrent transactions
- 1000 x 1000 parallel address-set comparison = too expensive?
- No cache coherency protocol on GPU

**Version Management:**
- Checkpointing register file of 1000s of threads is not cheap
- No caches for write buffering

**Commit Bottleneck:** Potentially serializing all transaction
- Allow non-contradicting transaction to commit in parallel

**KILO Transactional Memory**

- Support 1000s of concurrent transactions on a GPU

- **Key Insights**
  - Self-Validation @ Commit (from RingSTM [2]): Simpler Protocol
  - Check conflict with committed transactions
  - Value-Based Conflict Detection [3]: Committed TX = Memory
  - Eliminate storage problem for 1000s of concurrent TX

- **Self-Validation @ Commit**
  - Value-Based Conflict Detection
  - Most CPU-Based HTM Systems

- **Invalidation @ Commit**
  - BUT, it serialize Commits!
  - TX1 has to finish commit before TX2 start checking memory
  - Use HW to do fast conflict detection among committing TX
  - Non-contradicting TX can validate in parallel

**Other High Level Design Choices**

- Weak Isolation
- Flattened Nested Transaction
- Lazy Version Management
  - Register file rollback done by software – usually not needed
  - Memory writes stored in local memory – backed by L1 cache
- Lazy Conflict Detection and Resolution
  - Simpler protocol and KILO support
- Support Unbounded Transaction - Limit by local memory size

**Detail Design**

- TX Log Unit: Log Generation and Transmission
- Commit Unit: Parallel Validation and Commit Pipeline

**References**