KILO Transactional Memory for GPU

Wilson W. L. Fung Inderpeet Singh Andrew Brownsword[1] Tor M. Aamodt

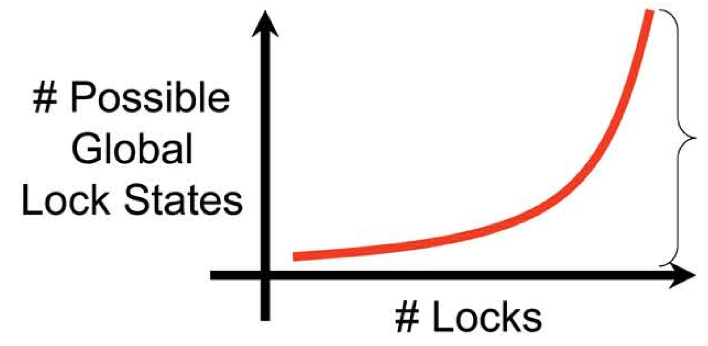
University of British Columbia, Canada [1]Intel Corp.

Transactional 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*
- Require consideration of all possible interaction between locks



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

Transactional Memory [1]
- Program specifies atomic code blocks called transactions
- Transaction are executed as if it runs in isolation
  - Simplify correctness burden on programmer

```
Lock Version:                    TM Version:
s = S[threadIdx.x];              s = S[threadIdx.x];
t = T[threadIdx.x];              t = T[threadIdx.x];
Lock(A[s], A[t]);                __tbegin();
  A[s] -= A[t] / 2;                A[s] -= A[t] / 2;
Unlock(A[s], A[t]);              __tcommit();
```

Potential Deadlock!

- A *natural extension* to CUDA Programming Model
  - Transaction execution sequencing abstracted away:

Non-conflicting transactions may run in parallel

Conflicting transactions automatically serialized

References

[1] M. Herlihy and J. E. B. Moss. Transactional Memory: architectural support for lock-free data structures. *ISCA 1993.*

[2] M. F. Spear, M. M. Michael, and C. von Praun. RingSTM: scalable transactions with a single atomic instruction. *SPAA 2008.*

[3] M. Olszewski, J. Cutler, and J. G. Steffan. JudoSTM: A Dynamic Binary-Rewriting Approach to Software Transactional Memory. *PACT 2007.*

Challenges

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

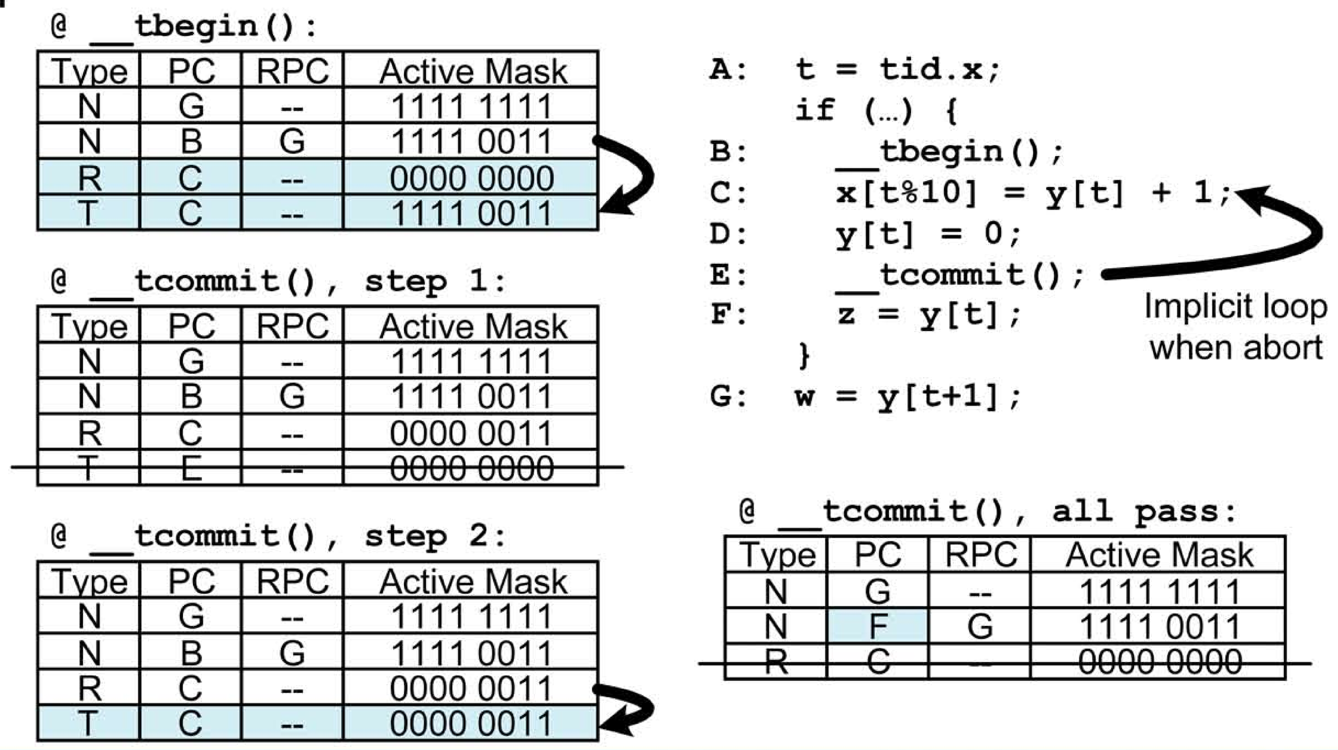
Scalable Conflict Detection: 1000s of concurrent transactions
- 1000 x 1000 parallel address-set comparison – too expensive?
- No cache coherency protocol on GPU

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-conflicting transaction to commit in parallel

Transaction-Aware SIMT Stack

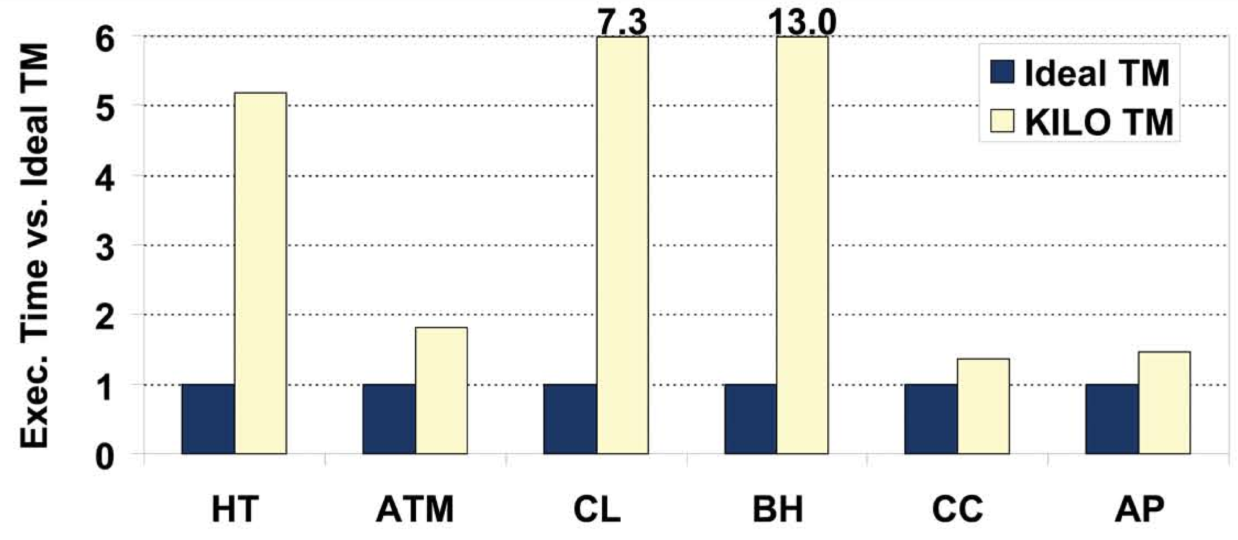
- SIMT stack automatically serializes a warp @ CF divergence
- Special entries for TM to handle transaction aborts:



Performance Results

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

HT – Hash Table Construction	ATM – Bank Transactions	CL – Cloth Simulation
BH – Barnes Huts (N-Body)	CC – Maxflow/Mincut Graph	AP – Data Mining

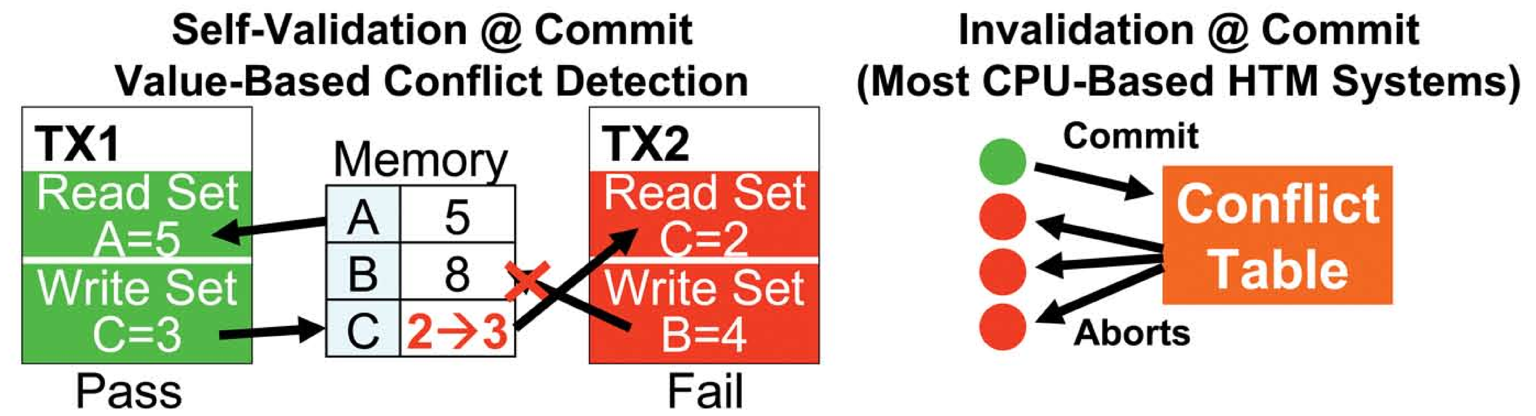


- Avg. 20% perf. of an ideal TM – instant validation and commit

KILO Transactional Memory

Support 1000s of concurrent transactions on a GPU

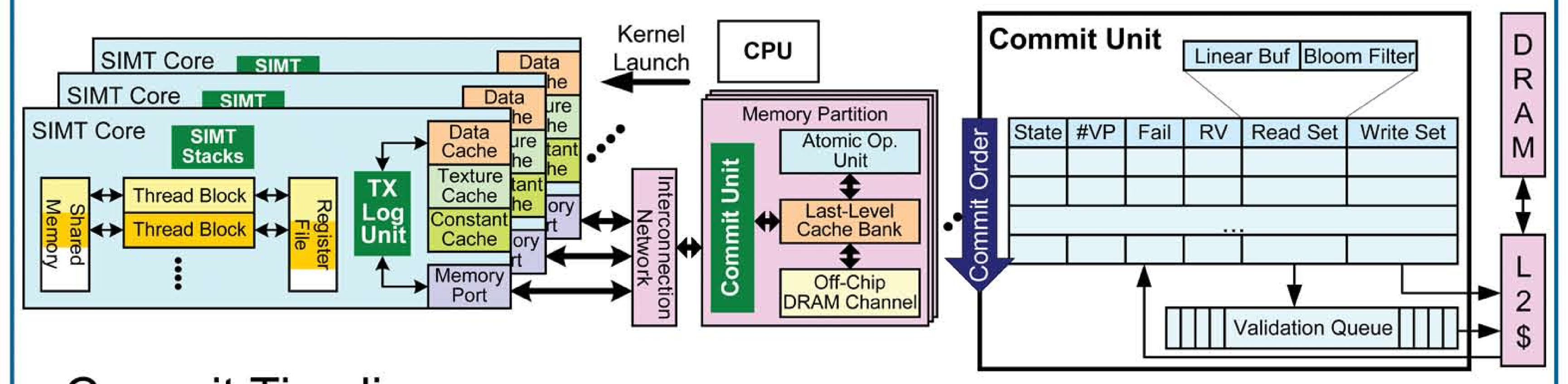
Key Insights
- Self-Validation @ Commit (from RingSTM [2]): Simpler Protocol
  - Check conflict with *committed* transactions
- Value-Based Conflict Detection [3]: Committed TX = Memory
  - Eliminate storage problem for 1000s of concurrent TX



- 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-conflicting 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 SIMT 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



Commit Timeline