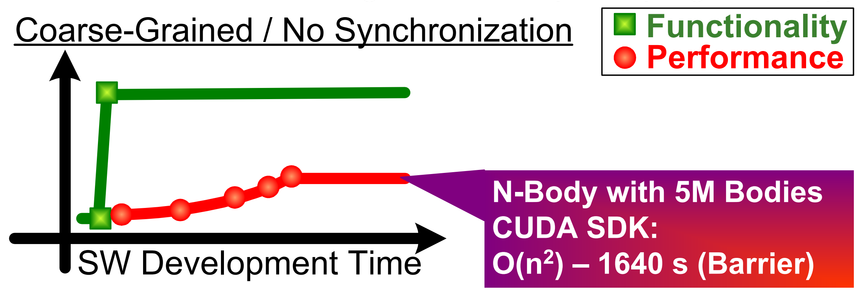
KILO Transactional Memory for GPU

Wilson W. L. Fung* wwlfung@ece.ubc.ca
Inderpeet Singh* isingh@ece.ubc.ca
Andrew Brownsword andrew@brownsword.ca
Tor M. Aamodt* aamodt@ece.ubc.ca
*University of British Columbia, Canada

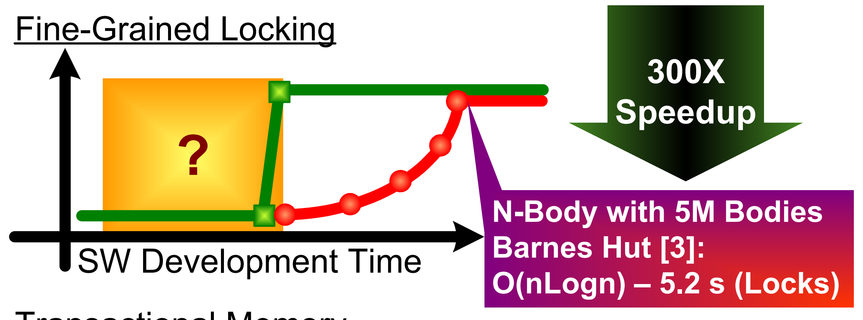
Transactional Memory on GPU?

Motivation: Exploit irregular parallelism on GPUs
- Enable more work-efficient algorithms + Improve programmer productivity

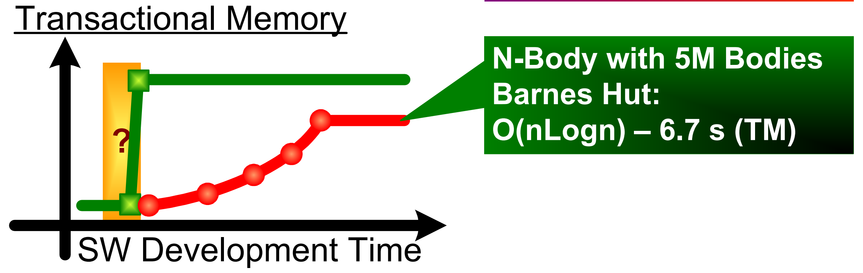
Coarse-Grained / No Synchronization



Fine-Grained Locking



Transactional Memory



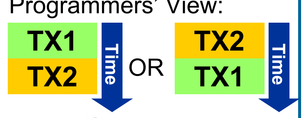
Transactional Memory [1]

- Programmer specifies atomic code blocks called transactions
  - Reduce correctness burden on programmer

```
Lock Version:                    TM Version:
Lock(A[s], A[t]);                atomic {
  A[s] -= A[t] / 2;                A[s] -= A[t] / 2;
Unlock(A[s], A[t]);              }
```

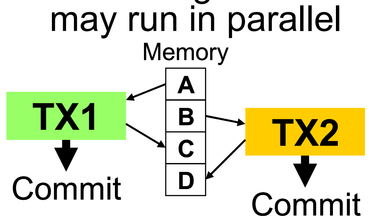
Potential Deadlock!

Programmers' View:

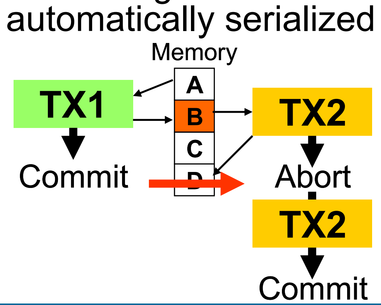


- *Naturally extends* CUDA Programming Model

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. Olszewski, J. Cutler, and J. G. Steffan. JudoSTM: A Dynamic Binary-Rewriting Approach to Software Transactional Memory. *PACT 2007.*
[3] M. Burtscher and K. Pingali. An Efficient CUDA Implementation of the Tree-based Barnes Hut n-Body Algorithm. *Chapter 6 in GPU Computing Gems Emerald Edition, 2011.*
See Our Full Paper for More Details:
W. W. L. Fung, I. Singh, A. Brownsword and T. M. Aamodt. Hardware Transactional Memory for GPU Architectures. *MICRO 2011.* **(Selected for IEEE Micro Top Picks)**

Challenges

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

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

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

Commit Bottleneck: Potentially serializing all transaction
- Needs to 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:

```
A: t = tid.x;
   if (…) {
B:   tx_begin;
C:   x[t%10] = y[t] + 1;
D:   if (s[t])
E:     y[t] = 0;
F:   tx_commit;
G:   z = y[t];
   }
H: w = y[t+1];
```

Implicit loop when abort

@ tx_begin:

	Type	PC	RPC	Active Mask
	N	H	--	1111 1111
	N	B	H	1111 0011
	R	C	--	0000 0000
TOS	T	C	--	1111 0011

Copy Active Mask

@ tx_commit, thread 6 & 7 failed validation:

	Type	PC	RPC	Active Mask
	N	H	--	1111 1111
	N	B	H	1111 0011
TOS	R	C	--	**0000 0011**
	~~T~~	~~F~~	~~--~~	~~0000 0000~~

Copy Active Mask + PC

@ tx_commit, restart Tx for thread 6 & 7:

	Type	PC	RPC	Active Mask
	N	H	--	1111 1111
	N	B	H	1111 0011
	R	C	--	**0000 0000**
TOS	T	C	--	0000 0011

Branch Divergence within Tx:

	Type	PC	RPC	Active Mask
	N	H	--	1111 1111
	N	B	H	1111 0011
	R	C	--	0000 0000
	T	F	--	1111 0011
TOS	N	E	F	0001 0011

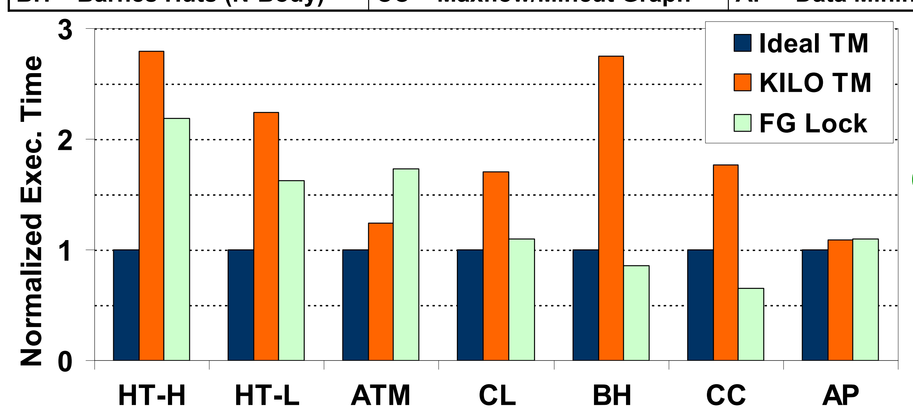
@ tx_commit, all threads with Tx committed:

	Type	PC	RPC	Active Mask
	N	H	--	1111 1111
TOS	N	G	H	1111 0011
	~~R~~	~~C~~		~~0000 0000~~

Performance: 59% of FG-Locking

- 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



128X Speedup over CG-Locks

0.5% HW Area Overhead

KILO Transactional Memory

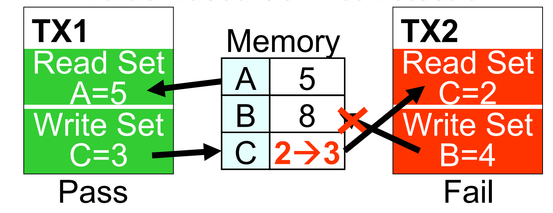
Support 1000s of concurrent transactions on a GPU
Support Unbounded Transaction ← Limited by local memory size

Key Ideas

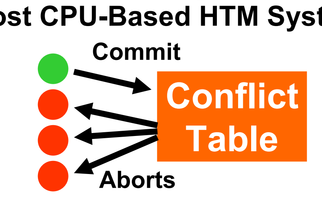
Only Detect *Existence* of Conflict (Not Identity)
- Check conflict with *committed* transactions

Value-Based Conflict Detection [2]: Committed TX in Memory
- Eliminate storage problem for 1000s of concurrent TX

Self-Validation @ Commit
Value-Based Conflict Detection



Invalidation @ Commit
(Most CPU-Based HTM Systems)



- BUT, it *serialize Commits*!
  - TX1 has to finish commit before TX2 start checking memory

Speculative Validation: Split Conflict Detection into Two Parts
- Recently Committed TX in Parallel ← Check Value in Memory
- Concurrently Committing TX in Commit Order
  - Fast but Approximate HW: *Recency Bloom Filter*
  - Serialize Detected Conflicting TX Commits
  - **Conflict Rare → Good Commit Parallelism**

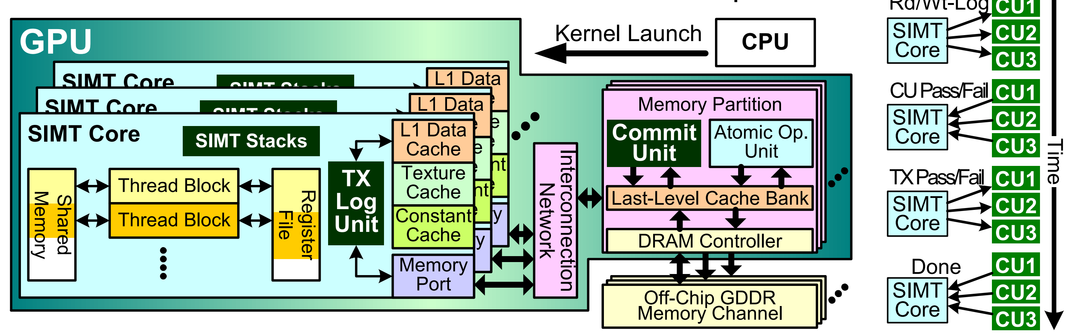
Other Design Choices

- Minimalistic: Weak Isolation + Flattened Nested Transactions
- Software Register Rollback – Rarely Needed
- Linear Memory Write Logs in Local Memory – Rarely Searched

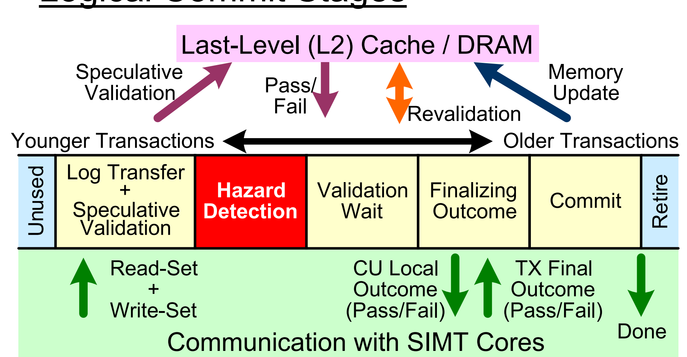
HW Design Detail

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

Commit Traffic



Logical Commit Stages



Commit Unit