Software Transactional Memory for GPU Architectures

Yunlong Xu\textsuperscript{1} \quad Rui Wang\textsuperscript{2} \quad Nilanjan Goswami\textsuperscript{3}

Tao Li\textsuperscript{3} \quad Lan Gao\textsuperscript{2} \quad Depei Qian\textsuperscript{1, 2}

\textsuperscript{1} Xi’an Jiaotong University, China
\textsuperscript{2} Beihang University, China
\textsuperscript{3} University of Florida, USA
Motivation

- General-Purpose computing on Graphics Processing Units (GPGPU)
  - High compute throughput and efficiency
  - GPU threads usually operate on independent data

- GPU + applications with data dependencies between threads?
  - Current data synchronization approaches on GPUs
    - Atomic read-modify-write operations
    - Locks constructed by using atomic operations
Background: GPU Locks

- GPU lock-based synchronization is challenging
  - Conventional problems related to locking
  - 1000s of concurrently executing threads
  - SIMT execution paradigm

<table>
<thead>
<tr>
<th>Lock schemes on GPUs</th>
<th>Pitfalls due to SIMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spinlock</td>
<td>Deadlock</td>
</tr>
<tr>
<td>Serialization within warp</td>
<td>Low utilization</td>
</tr>
<tr>
<td>Diverging on failure</td>
<td>Livelock</td>
</tr>
</tbody>
</table>
Background: GPU Locks

- GPU lock scheme #1: spinlock
  - Pitfall due to SIMT: deadlock

```c
repeat locked ← CAS(&lock, 0, 1)
until locked = 0

critical section...
lock ← 0
```
Background: GPU Locks

- GPU lock scheme #1: spinlock
  - Pitfall due to SIMT: deadlock

![Diagram illustrating deadlock between two threads accessing a lock]

Thread #0
- Get lock
Thread #1
- Get lock

Deadlock

Time
GPU lock scheme #2: serialization within warp

- Pitfall due to SIMT: low hardware utilization

```c
for i ← 1 to WARP_SIZE do
    if (threadIdx.x % WARP_SIZE) = i then
        repeat locked ← CAS(&lock, 0, 1)
        until locked = 0
    critical section...
    lock ← 0
```
GPU lock scheme #2: serialization within warp

- Pitfall due to SIMT: low hardware utilization
Background: GPU Locks

- GPU lock scheme #3: diverging on failure
  - Pitfall due to SIMT: livelock

```plaintext
done ← false
while done = false do
    if CAS(&lock, 0, 1) = 0 then
        critical section...
        lock ← 0
        done ← true
```

- GPU lock scheme #3: diverging on failure
  - Pitfall due to SIMT: livelock
Background: GPU Locks

- GPU lock scheme #3: diverging on failure
  - Pitfall due to SIMT: livelock

![Diagram showing lock interactions]

- Thread #0
  - Get lock0
  - Get lock1
  - Release lock0

- Thread #1
  - Get lock1
  - Get lock0
  - Release lock1

Livelock
Talk Agenda

- Motivation
- Background: GPU Locks
- Transactional Memory (TM)
  - GPU-STM: Software TM for GPUs
    - GPU-STM Code Example
    - GPU-STM Algorithm
- Evaluation
- Conclusion
Transactional Memory (TM)
Talk Agenda

- Motivation
- Background: GPU Locks
- Transactional Memory (TM)
- GPU-STM: Software TM for GPUs
  - GPU-STM Code Example
  - GPU-STM Algorithm
- Evaluation
- Conclusion
GPU-STM Code Example

```c
__host__ void host_fun () {
    STM_STARTUP();
    trans_kernel <<<BLOCKS, BLOCK SIZE>>>();
    STM_SHUTDOWN();
}

__global__ void trans_kernel (){
    Warp *warp = STM_NEW_WARP();
    TXBegin(warp);
    ...
    TXRead(&addr1, warp);
    TXWrite(&addr2, val, warp);
    ...
    TXCommit(warp);
    STM_FREE_WARP(warp);
}
```
__host__ void host_fun () {
    STM_STARTUP();
    trans_kernel <<<BLOCKS, BLOCK SIZE>>>();
    STM_SHUTDOWN();
}
__global__ void trans_kernel (){
    Warp *warp = STM_NEW_WARP();
    TXBegin(warp);
    ...
    TXRead(&addr1, warp);
    TXWrite(&addr2, val, warp);
    ...
    TXCommit(warp);
    STM_FREE_WARP(warp);
}
Talk Agenda

- Motivation
- Background: GPU Locks
- Transactional Memory (TM)
- GPU-STM: Software TM for GPUs
  - GPU-STM Code Example
  - GPU-STM Algorithm
- Evaluation
- Conclusion
A high-level view of GPU-STM algorithm

```c
Val TXRead(Addr addr){
    if write_set.find(addr)
        return write_set(addr)
    val = *addr
    validation()
    lock_log.insert(hash(addr))
    return val
}

void TXWrite(Addr addr, Val val){
    write_set.update(addr, val)
    lock_log.insert(hash(addr))
}

void TXCommit () {
    loop:
    if !get_locks(lock_log)
        goto loop
    validation()
    update_memory(write_set)
    release_locks(lock_log)
}
```
GPU-STM Algorithm

- GPU-STM highlight #1: locking algorithm

```c
Val TXRead(Addr addr) {
    if write_set.find(addr)
        return write_set(addr)
    val = *addr
    validation()
    lock_log.insert(hash(addr))
    return val
}

TXWrite(Addr addr, Val val) {
    write_set.update(addr, val)
    lock_log.insert(hash(addr))
}

void TXCommit() {
    loop:
        if !get_locks(lock_log)
            goto loop
        validation()
        update_memory(write_set)
        release_locks(lock_log)
}
```
Val TXRead(Addr addr){
    if write_set.find(addr)
        return write_set(addr)
    val = *addr
    validation()
    lock_log.insert(hash(addr))
    return val
}

void TXWrite(Addr addr, Val val){
    write_set.update(addr, val)
    lock_log.insert(hash(addr))
}

TXWrite(Addr addr, Val val){
    write_set.update(addr, val)
    lock_log.insert(hash(addr))
}

void TXCommit () {
    loop:
        if !get_locks(lock_log)
            goto loop
        validation()
        update_memory(write_set)
        release_locks(lock_log)
    }
GPU-STM Algorithm: Locking

- GPU lock scheme #3: diverging on failure

![Diagram showing locking process with threads and locks](image)

**Livelock**
GPU-STM Algorithm: Locking

- Solution: sorting before acquiring locks

```
TX #0
√ Get lock0
√ Get lock1
Commit
Release locks

TX #1
Get lock0 ✗
Get lock0
Get lock1 ✓
Commit
Release locks
```

Time
GPU-STM Algorithm: Locking

- **Encounter-time lock-sorting**

Val `TXRead(Addr addr)`{
  
  if write_set.find(addr)
  
  return write_set(addr)
  
  val = *addr
  
  validation()

  lock_log.insert(hash(addr))

  return val

}

void `TXWrite(Addr addr, Val val)`{
  
  write_set.update(addr, val)

  lock_log.insert(hash(addr))

}

void `TXCommit ()` {
  
  loop:

  if !get_locks(lock_log)

    goto loop

  validation()

  update_memory(write_set)

  release_locks(lock_log)

}
**GPU-STM Algorithm: Locking**

- **Local lock-log**

<table>
<thead>
<tr>
<th>Locks</th>
<th>Indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0002</td>
<td>3</td>
</tr>
<tr>
<td>0x0008</td>
<td>-1</td>
</tr>
<tr>
<td>0x0004</td>
<td>1</td>
</tr>
<tr>
<td>0x0003</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of local lock-log](image)
GPU-STM Algorithm: Locking

- Encounter-time lock-sorting

For each incoming lock:

<table>
<thead>
<tr>
<th>Head</th>
<th>Lock</th>
<th>Indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x0002</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0x0008</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>0x0004</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0x0003</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local Lock-log

0x0005
GPU-STM Algorithm: Locking

- Encounter-time lock-sorting

For each incoming lock:

- Compare with existing locks

<table>
<thead>
<tr>
<th>Index</th>
<th>Locks</th>
<th>Indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x0002</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0x0008</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>0x0004</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0x0003</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local Lock-log
**GPU-STM Algorithm: Locking**

- **Encounter-time lock-sorting**

For each incoming lock:
- Compare with existing locks
- Insert into log, and update indexes

```
<table>
<thead>
<tr>
<th>Head</th>
<th>Locks</th>
<th>Indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x0002</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0x0008</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>0x0004</td>
<td>1→4</td>
</tr>
<tr>
<td>3</td>
<td>0x0003</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0x0005</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Local Lock-log
GPU-STM Algorithm: Locking

- Encounter-time Lock-sorting

- Average cost: $\frac{n^2}{4} + \Theta(n)$ comparisons, $n$ is num of locks.

- If $n = 64$, $\frac{n^2}{4} + \Theta(n) > 1024$. 

<table>
<thead>
<tr>
<th>Head</th>
<th>Locks</th>
<th>Indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x0002</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0x0008</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>0x0004</td>
<td>1→4</td>
</tr>
<tr>
<td>3</td>
<td>0x0003</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0x0005</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Local Lock-log
GPU-STM Algorithm: Locking

- **Hash-table based encounter-time lock-sorting**

Hash-based Local Lock-log

Hash(lock) = lock % buckets

Hash(lock) = 0x0005

<table>
<thead>
<tr>
<th>Buckets</th>
<th>Locks</th>
<th>Indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x0002</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0x0008</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>0x0004</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0x0003</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
GPU-STM Algorithm: Locking

- **Hash-table based** encounter-time lock-sorting

Hash(lock) = lock % buckets

<table>
<thead>
<tr>
<th>Buckets</th>
<th>Locks</th>
<th>Indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x0002</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0x0008</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>0x0004</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0x0003</td>
<td>-1→4</td>
</tr>
<tr>
<td>4</td>
<td>0x0005</td>
<td>-1</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average cost: \( \frac{n^2}{4m} + \Theta(n) \) comparisons, \( n \) is num of locks, \( m \) is the num of buckets.
GPU-STM Algorithm : Conflict Detection

- Hierarchical validation
  - Time-based validation + value-based validation
GPU-STM Algorithm: Conflict Detection

- Hierarchical validation
  - Time-based validation → false conflict
    → hardware utilization loss due to SIMT
GPU-STM Algorithm: Conflict Detection

- Hierarchical validation
  - Time-based validation + value-based validation

```
Time-based validation

Pass?

Yes

No

Value-based validation

Pass?

Yes

No

Conflict

No Conflict
```
Talk Agenda

- Motivation
- Background: GPU Locks
- Transactional Memory (TM)
- GPU-STM: Software TM for GPUs
  - GPU-STM Code Example
  - GPU-STM Algorithm
- Evaluation
- Conclusion
Evaluation

- Implement GPU-STM on top of CUDA runtime
- Run GPU-STM on a NVIDIA C2070 Fermi GPU
- Benchmarks
  - 3 STAMP benchmarks + 3 micro-benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th>Shared Data</th>
<th>RD/TX</th>
<th>WR/TX</th>
<th>TX/Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA</td>
<td>8M</td>
<td>16</td>
<td>16</td>
<td>1M</td>
</tr>
<tr>
<td>HT</td>
<td>256K</td>
<td>8</td>
<td>8</td>
<td>1M</td>
</tr>
<tr>
<td>EB</td>
<td>1M-64M</td>
<td>32</td>
<td>32</td>
<td>1M</td>
</tr>
<tr>
<td>GN</td>
<td>16K/1M</td>
<td>1</td>
<td>1</td>
<td>4M/1M</td>
</tr>
<tr>
<td>LB</td>
<td>1.75M</td>
<td>352</td>
<td>352</td>
<td>512</td>
</tr>
<tr>
<td>KM</td>
<td>2K</td>
<td>32</td>
<td>32</td>
<td>64K</td>
</tr>
</tbody>
</table>
Performance Results

![Graph showing performance results for different methods and benchmarks.](image)

- **Methods**:
  - RA
  - HT
  - GN
  - LB
  - KM

- **Benchmarks**:
  - VBV
  - TBV-Sorting
  - HV-Sorting
  - HV-Backoff
  - Optimized
  - EGPGV

- **X-axis**: Benchmarks
- **Y-axis**: Speedup over CGL

The graph illustrates the speedup of various methods compared to the baseline (CGL) across different benchmarks.
Performance Results
Performance Results

![Graph showing speedup over CGL for different algorithms]

- RA
- HT
- GN
- LB
- KM

- VBV
- TBV-Sorting
- HV-Sorting
Performance Results

STM-Optimized: STM-HV-Sorting + STM-TBV-Sorting

- When amount of shared data < amount of locks, TBV
- Otherwise, HV
Performance Results

EGPGV STM [Cederman et al. EGPGV’10]
Hierarchical Validation vs. Time-based Validation

Throughput (TX/ ms)

Threads

Amount of shared data: 256MB
Talk Agenda

- Motivation
- Background: GPU Locks
- Transactional Memory (TM)
- GPU-STM: Software TM for GPUs
  - GPU-STM Code Example
  - GPU-STM Algorithm
- Evaluation
- Conclusion
Conclusion

- Lock-based synchronization on GPUs is challenging

- GPU-STM, a Software TM for GPUs
  - Enables simplified data synchronizations on GPUs
  - Scales to 1000s of TXs
  - Ensures livelock-freedom
  - Runs on commercially available GPUs and runtime
  - Outperforms GPU coarse-grain locks by up to 20x
Software Transactional Memory for GPU Architectures

Yunlong Xu\textsuperscript{1} \quad Rui Wang\textsuperscript{2} \quad Nilanjan Goswami\textsuperscript{3}

Tao Li\textsuperscript{3} \quad Lan Gao\textsuperscript{2} \quad Depei Qian\textsuperscript{1, 2}

\textsuperscript{1} Xi’an Jiaotong University, China
\textsuperscript{2} Beihang University, China
\textsuperscript{3} University of Florida, USA
Launch Configurations of Workloads

<table>
<thead>
<tr>
<th></th>
<th>RA</th>
<th>HT</th>
<th>GN-1, GN-2</th>
<th>LB</th>
<th>KM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread-blocks</td>
<td>256</td>
<td>256</td>
<td>256, 16</td>
<td>512</td>
<td>64</td>
</tr>
<tr>
<td>Threads per Block</td>
<td>256</td>
<td>256</td>
<td>256, 64</td>
<td>256</td>
<td>4</td>
</tr>
</tbody>
</table>
 Execution Time Breakdown

Percent of Thread Cycles

<table>
<thead>
<tr>
<th>GN-1</th>
<th>GN-2</th>
<th>LB</th>
<th>KM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>100%</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

- TX Aborted
- Committing
- Locking
- Consistency
- Buffering
- TX Initialization
- Native Code

~20x speedup

Tradeoff: STM overhead vs. scalability enabled
Scalability Results

RA

Execution Time (Seconds)

Threads

64 256 1K 4K 16K 64K 256K 1M

HT

Execution Time (Seconds)

Threads

64 256 1K 4K 16K 64K 256K 1M

GN-1

Execution Time (Seconds)

Threads

64 256 1K 4K 16K 64K 256K 1M

GN-2

Execution Time (Seconds)

Threads

64 256 1K 4K 16K 64K 256K 1M

LB

Execution Time (Seconds)

Threads

1K 2K 4K 8K 16K 32K 64K 128K

KM

Execution Time (Seconds)

Threads

4 16 64 256 1K 4K 16K 64K

STM-VBV
STM-TBV-Sorting
STM-HV-Backoff
STM-HV-Sorting
STM-Optimized
STM-EGPGV

45
Related Work

- TMs for GPUs
  - A STM for GPUs [Cederman et al. EGPGV’10]
  - KILO TM, a HTM for GPUs [Fung et al. MICRO’11, MICRO’13]

- STMs for CPUs
  - JudoSTM [Olszewski et al. PACT’07]
  - NORec STM [Dalessandro et al. PPoPP’10]
  - TL2 STM [Dice et al. DISC’06]
  - ...