Software Transactional Memory for GPU Architectures

Yunlong Xu[1] Rui Wang[2] Nilanjan Goswami[3]

Tao Li[3] Lan Gao[2] Depei Qian[1, 2]

[1] *Xi'an Jiaotong University, China*

[2] *Beihang University, China*

[3] *University of Florida, USA*

CGO, Orlando, USA
February 17, 2014

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

Lock schemes on GPUs	Pitfalls due to SIMT
Spinlock	Deadlock
Serialization within warp	Low utilization
Diverging on failure	Livelock

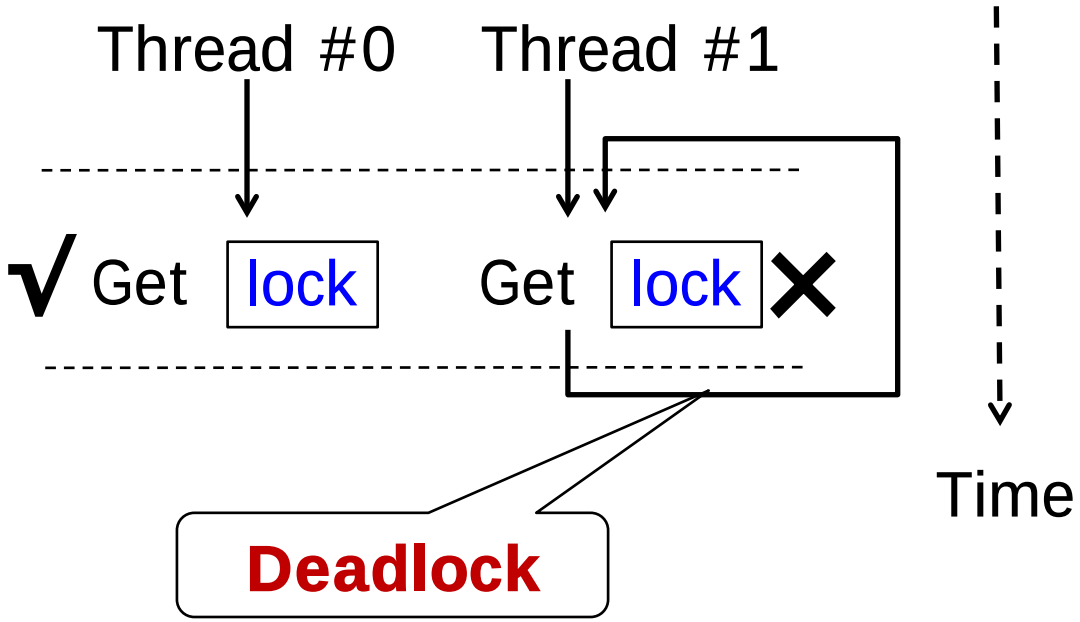
Background: GPU Locks

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

```
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



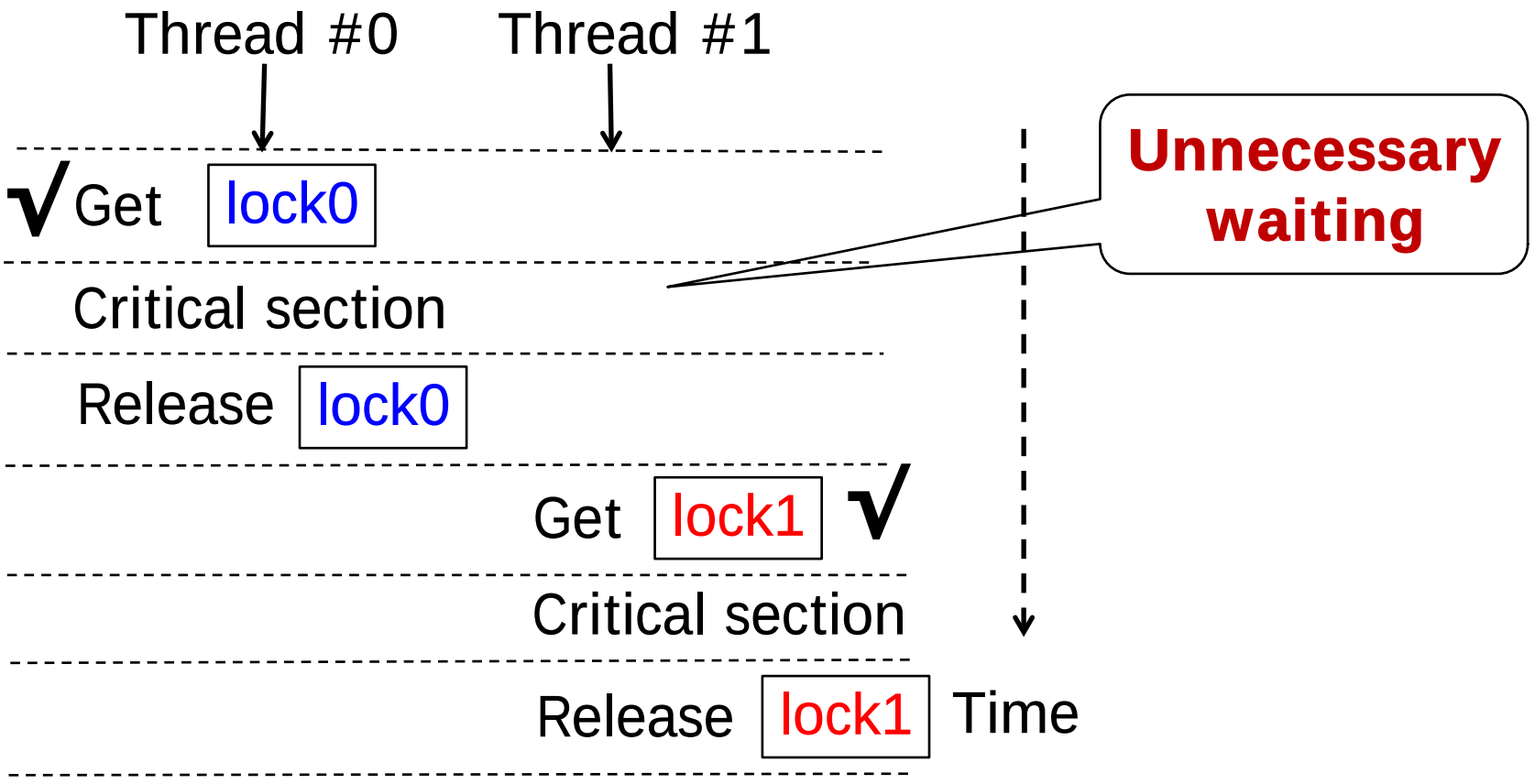
Background: GPU Locks

- GPU lock scheme #2: serialization within warp
  - Pitfall due to SIMT: low hardware utilization

```
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
```

Background: GPU Locks

- GPU lock scheme #2: serialization within warp
  - Pitfall due to SIMT: low hardware utilization



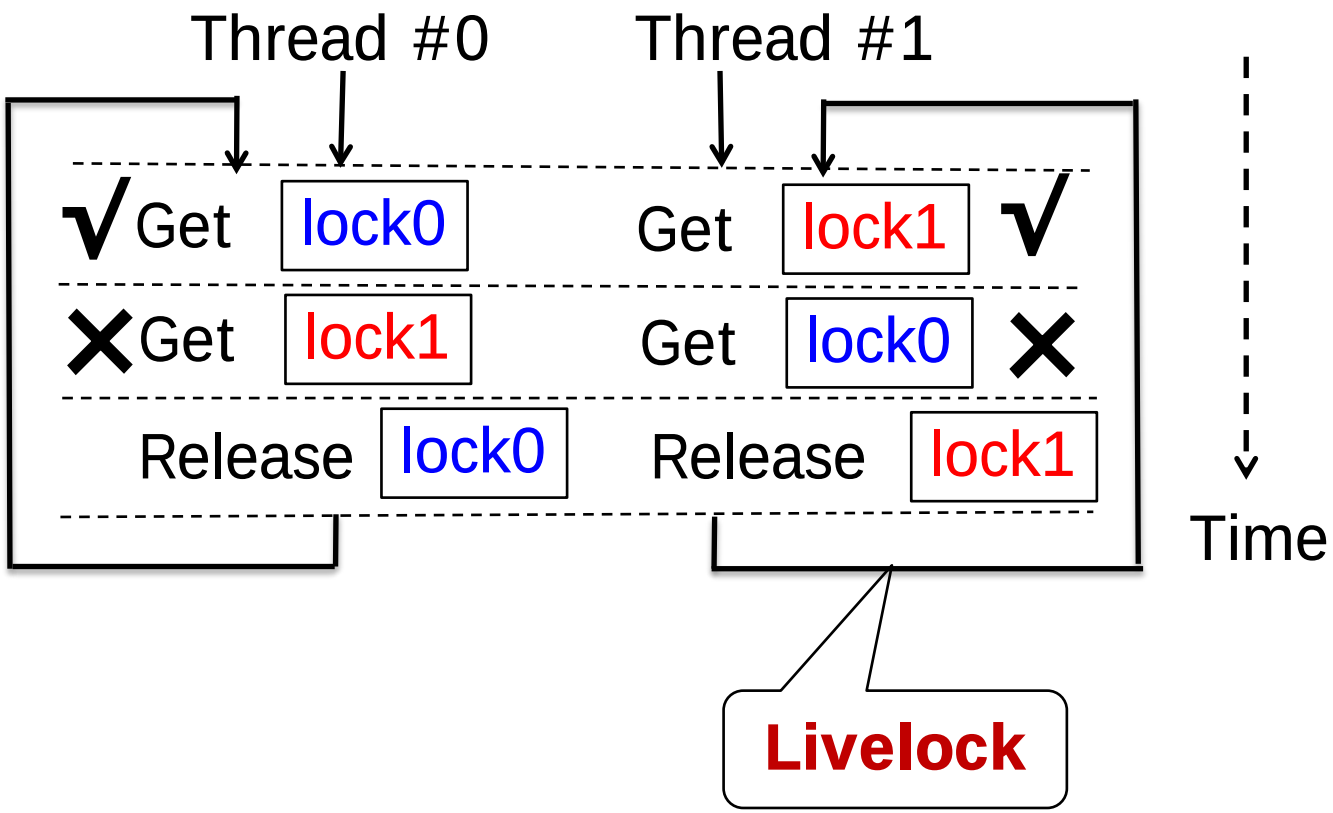
Background: GPU Locks

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

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

Background: GPU Locks

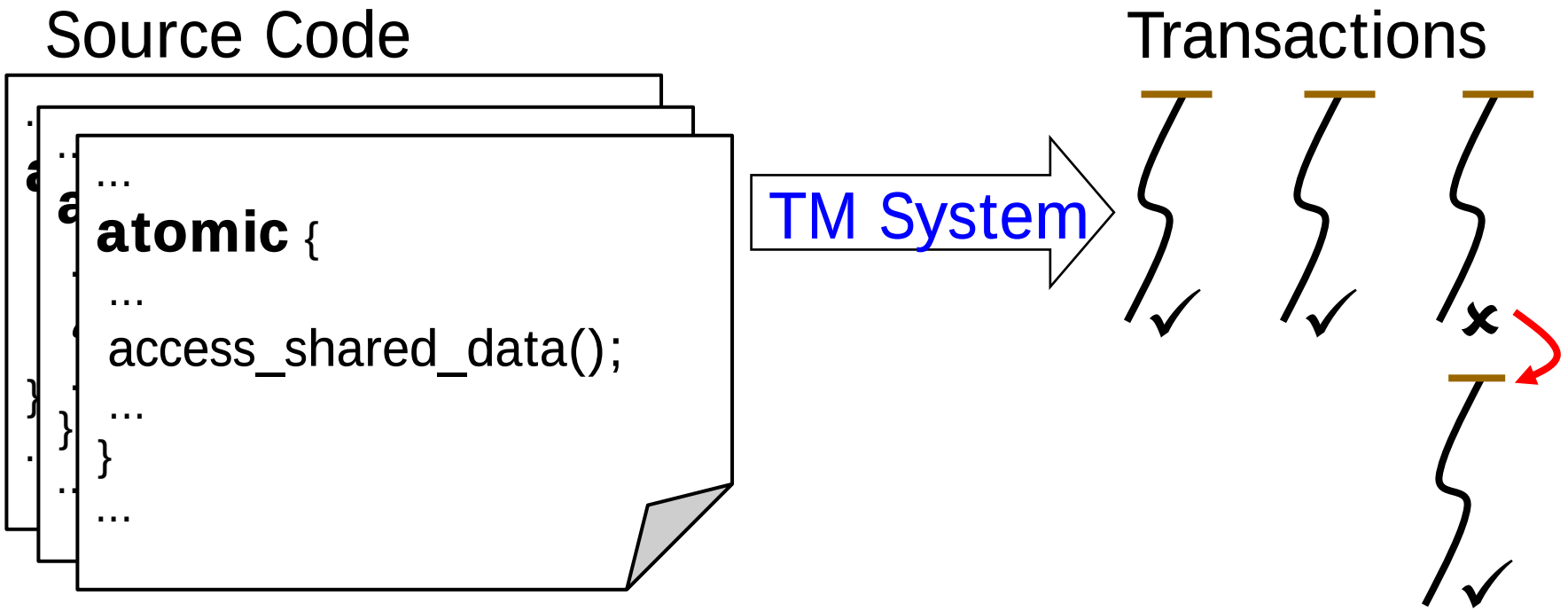
- GPU lock scheme #3: diverging on failure
  - Pitfall due to SIMT: livelock



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

```
__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);
}
```

GPU-STM Code Example

```
__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

GPU-STM Algorithm

- A high-level view of GPU-STM algorithm

```
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)
}
```

GPU-STM Algorithm

- GPU-STM highlight #1: locking algorithm

```
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)
}
```

GPU-STM Algorithm

- GPU-STM highlight #2: conflict detection algorithm

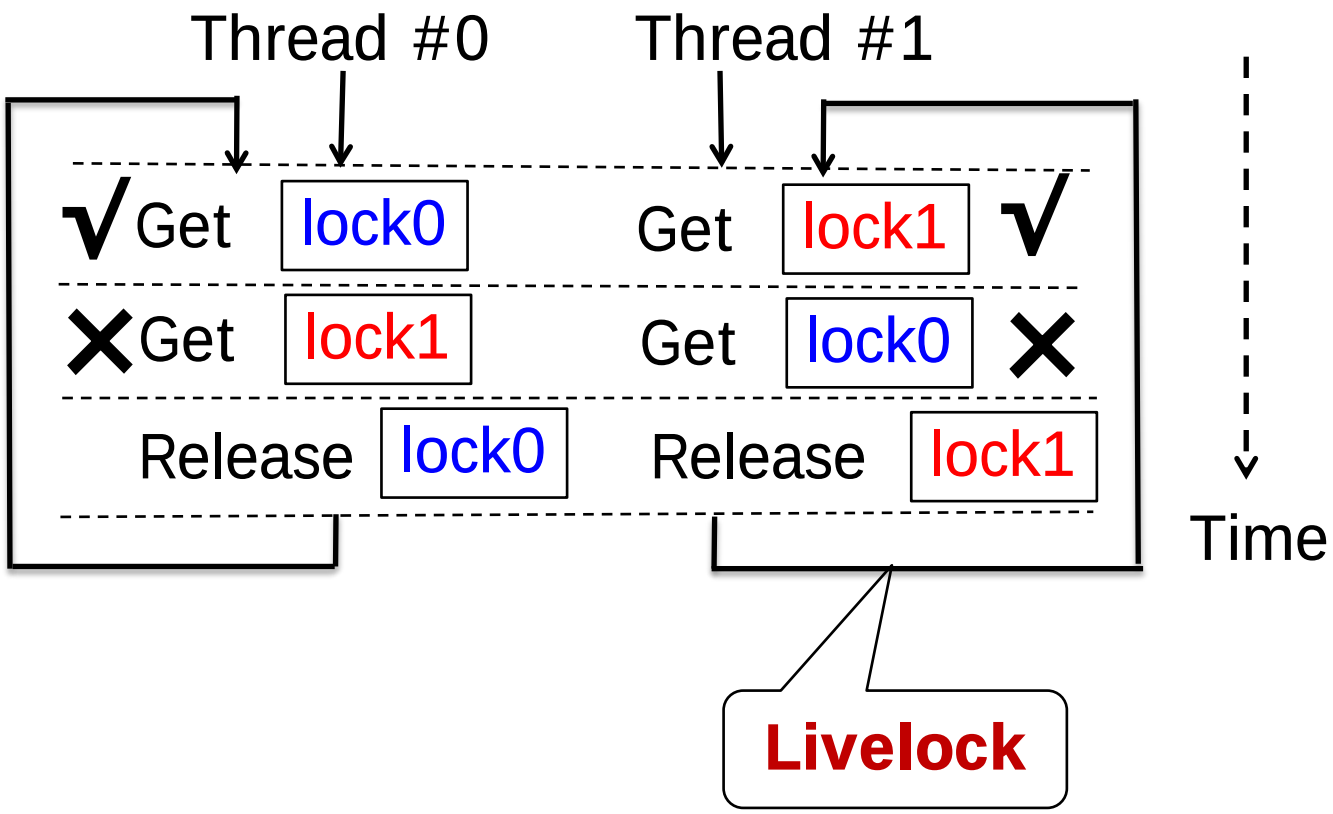
```
Val TXRead(Addr addr){
  if write_set.find(addr)
    return write_set(addr)
  val = *addr
  validation()
  lock_log.insert(hash(addr))
  return val
}

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)
}
```

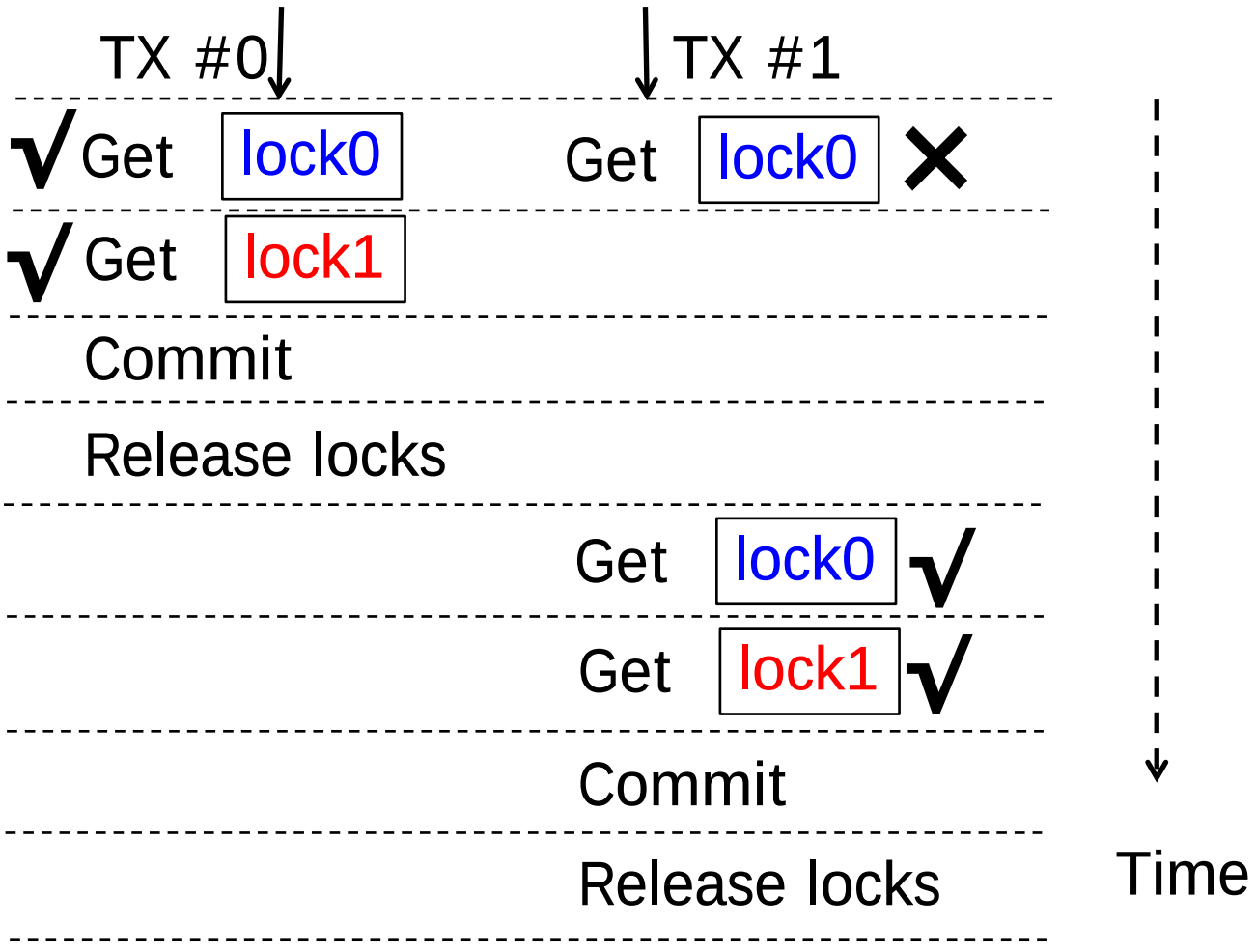
GPU-STM Algorithm: Locking

- GPU lock scheme #3: diverging on failure



GPU-STM Algorithm: Locking

- Solution: sorting before acquiring locks



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
}

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)
}
```

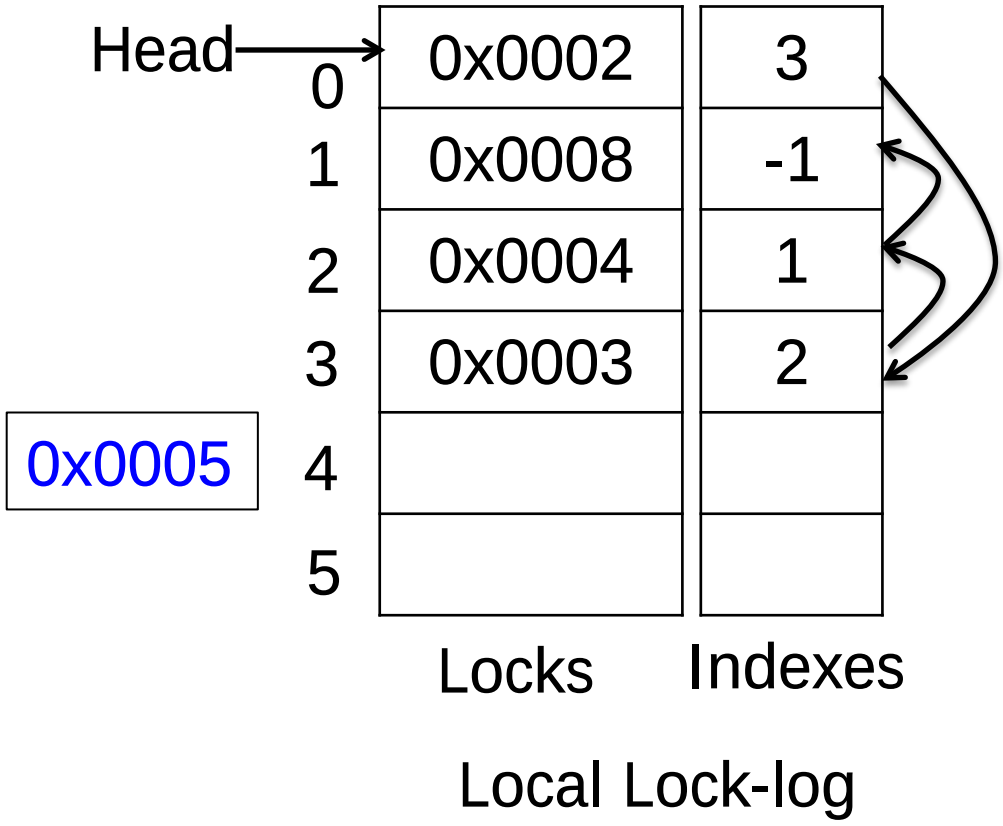
GPU-STM Algorithm: Locking

- Local lock-log

GPU-STM Algorithm: Locking

- Encounter-time lock-sorting

For each incoming lock:

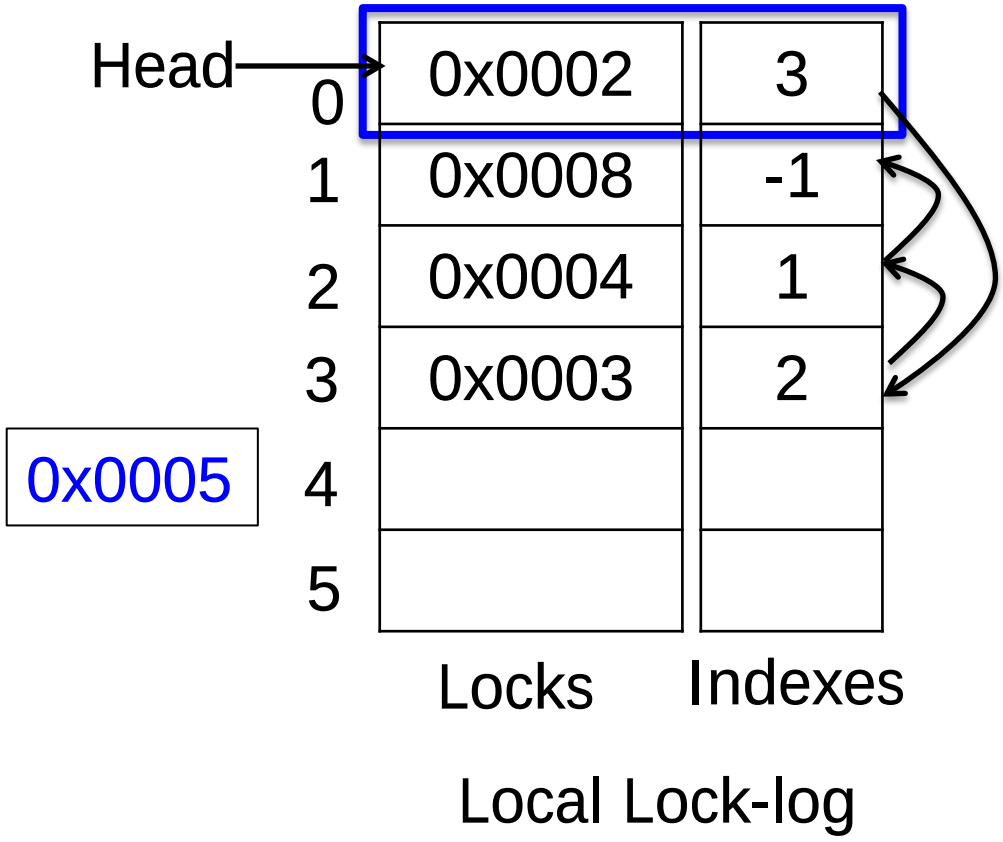


GPU-STM Algorithm: Locking

- Encounter-time lock-sorting

For each incoming lock:

- Compare with existing locks

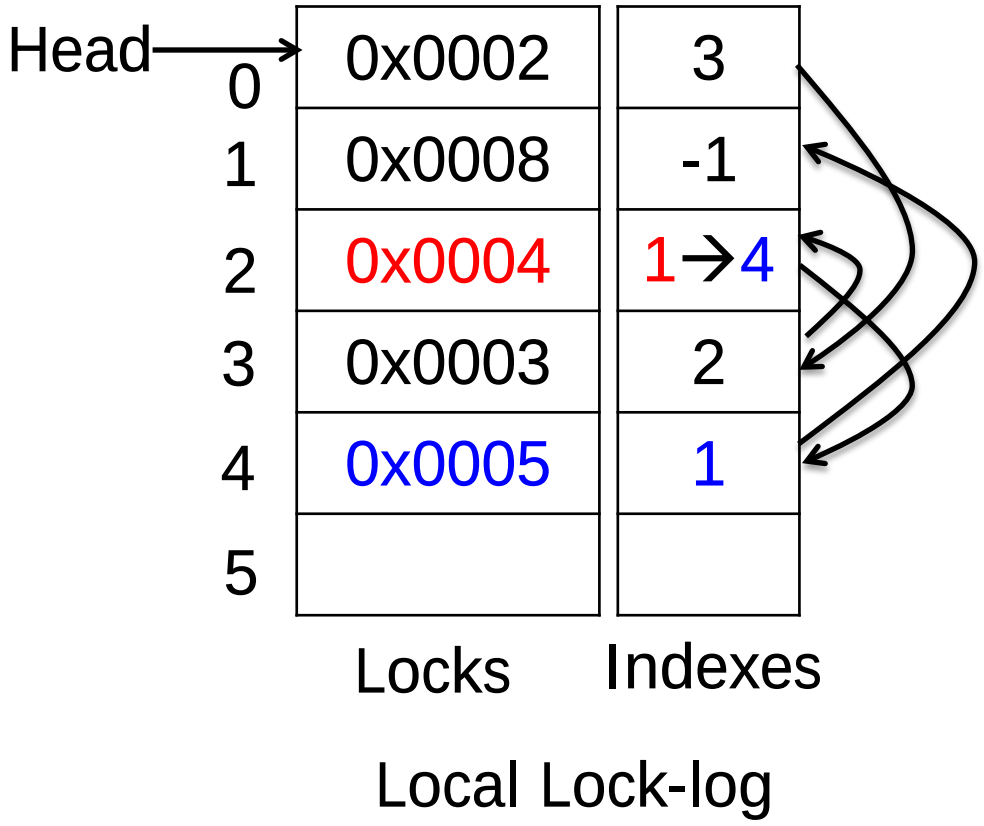


GPU-STM Algorithm: Locking

- Encounter-time lock-sorting

For each incoming lock:

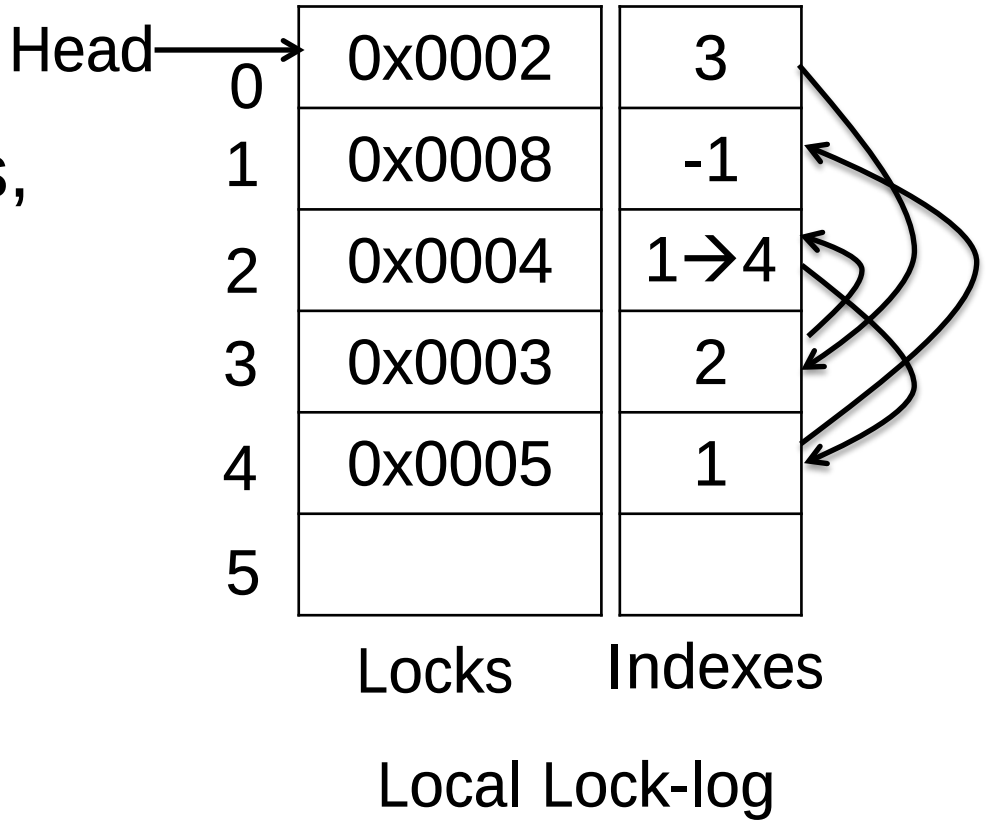
- Compare with existing locks
- Insert into log, and update indexes



GPU-STM Algorithm: Locking

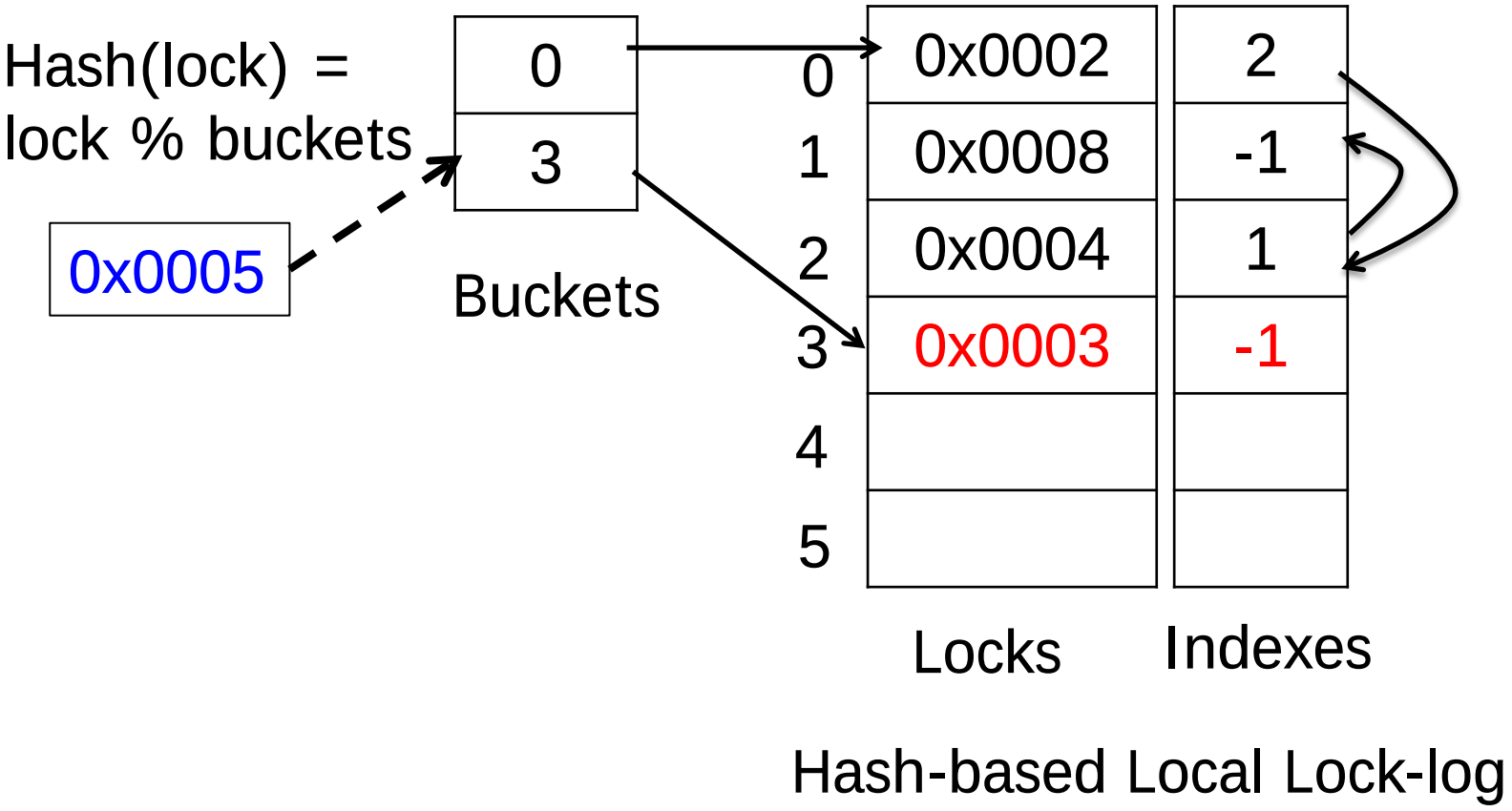
- Encounter-time Lock-sorting

- Average cost:
  $n^2/4 + \Theta(n)$ comparisons,
  n is num of locks.
- If $n = 64$,
  $n^2/4 + \Theta(n) > 1024$.



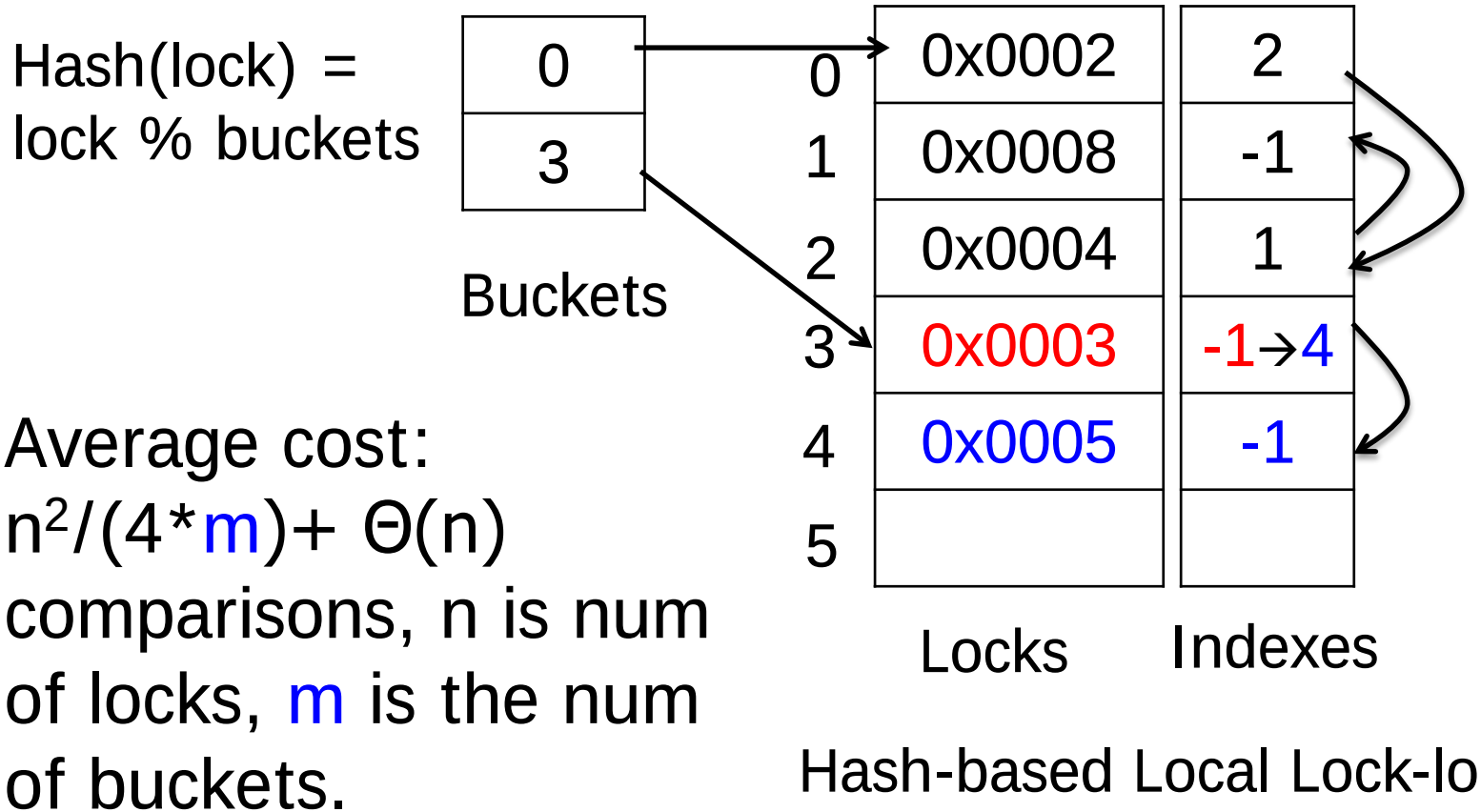
GPU-STM Algorithm: Locking

- Hash-table based encounter-time lock-sorting



GPU-STM Algorithm: Locking

- Hash-table based encounter-time lock-sorting



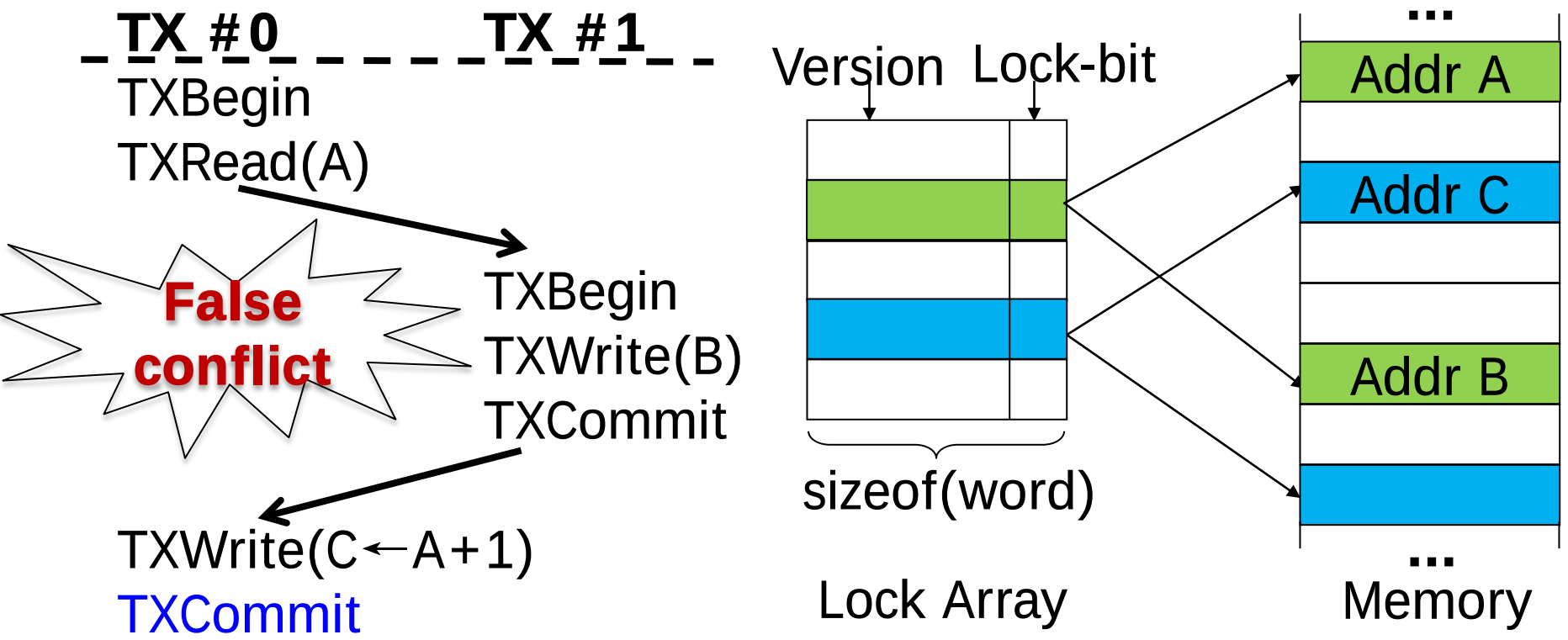
- Average cost: $n^2/(4*m)+ \Theta(n)$ comparisons, n is num of locks, m is the num of buckets.

GPU-STM Algorithm : Conflict Detection

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

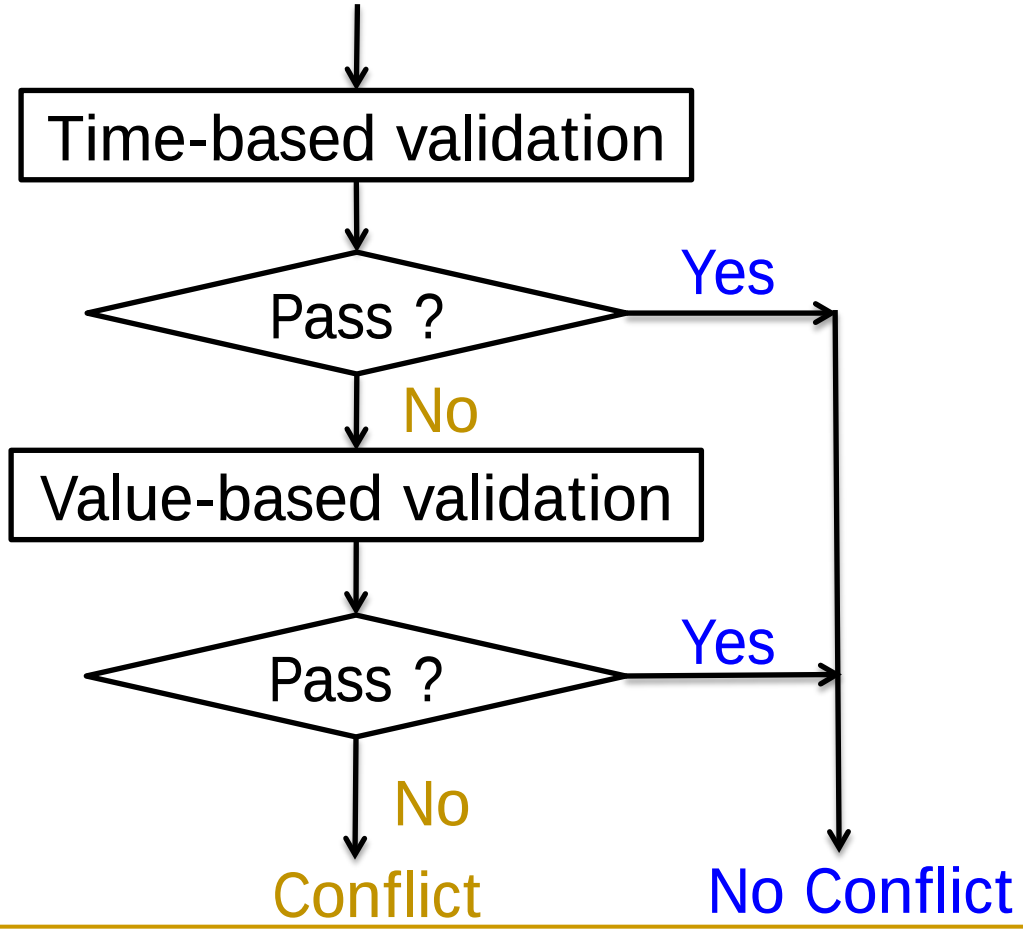
GPU-STM Algorithm : Conflict Detection

- 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



Talk Agenda

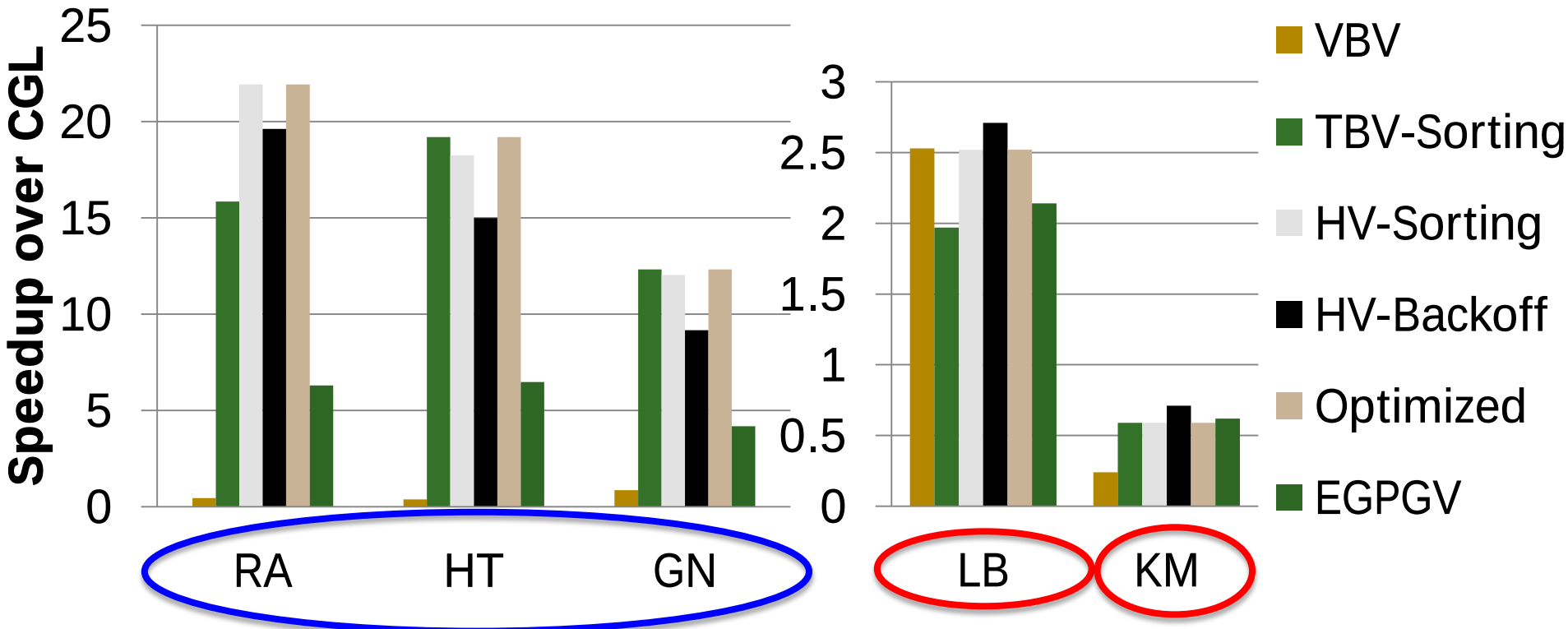
- Motivation
- Background: GPU Locks
- Transactional Memory (TM)
- GPU-STM: Software TM for GPUs
  - GPU-STM Code Example
  - GPU-STM Algorithm
- Evaluation
- Conclusion

Evaluation

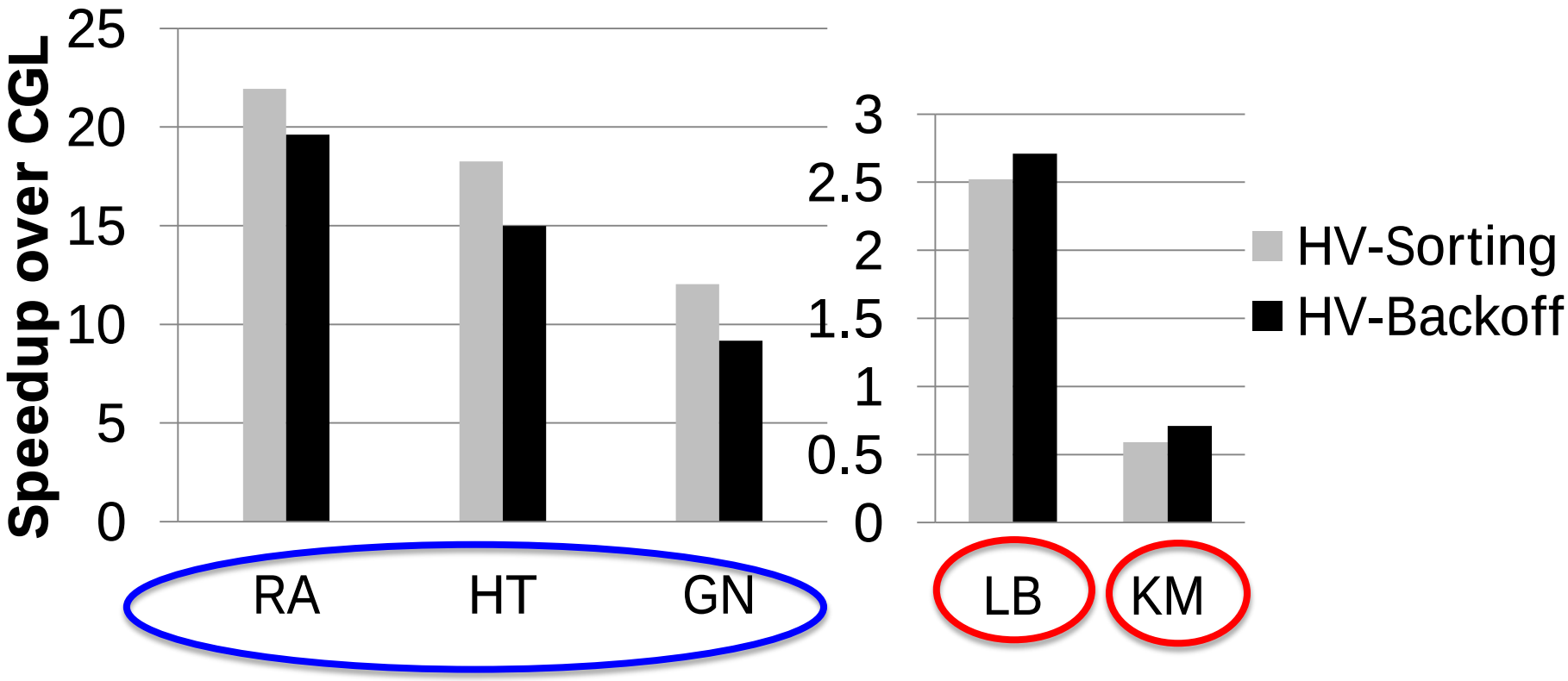
- Implement GPU-STM on top of CUDA runtime
- Run GPU-STM on a NVIDIA C2070 Fermi GPU
- Benchmarks
  - 3 STAMP benchmarks + 3 micro-benchmarks

Name	Shared Data	RD/TX	WR/TX	TX/Kernel
RA	8M	16	16	1M
HT	256K	8	8	1M
EB	1M-64M	32	32	1M
GN	16K/1M	1	1	4M/1M
LB	1.75M	352	352	512
KM	2K	32	32	64K

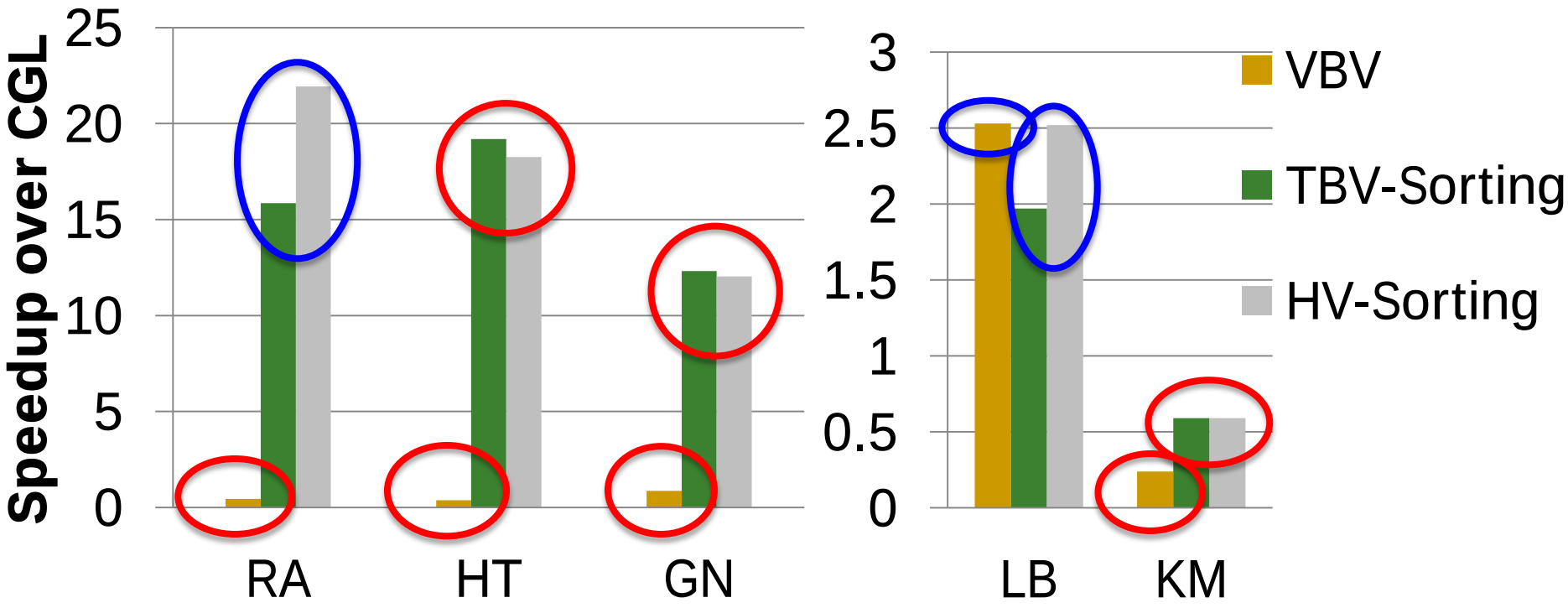
Performance Results



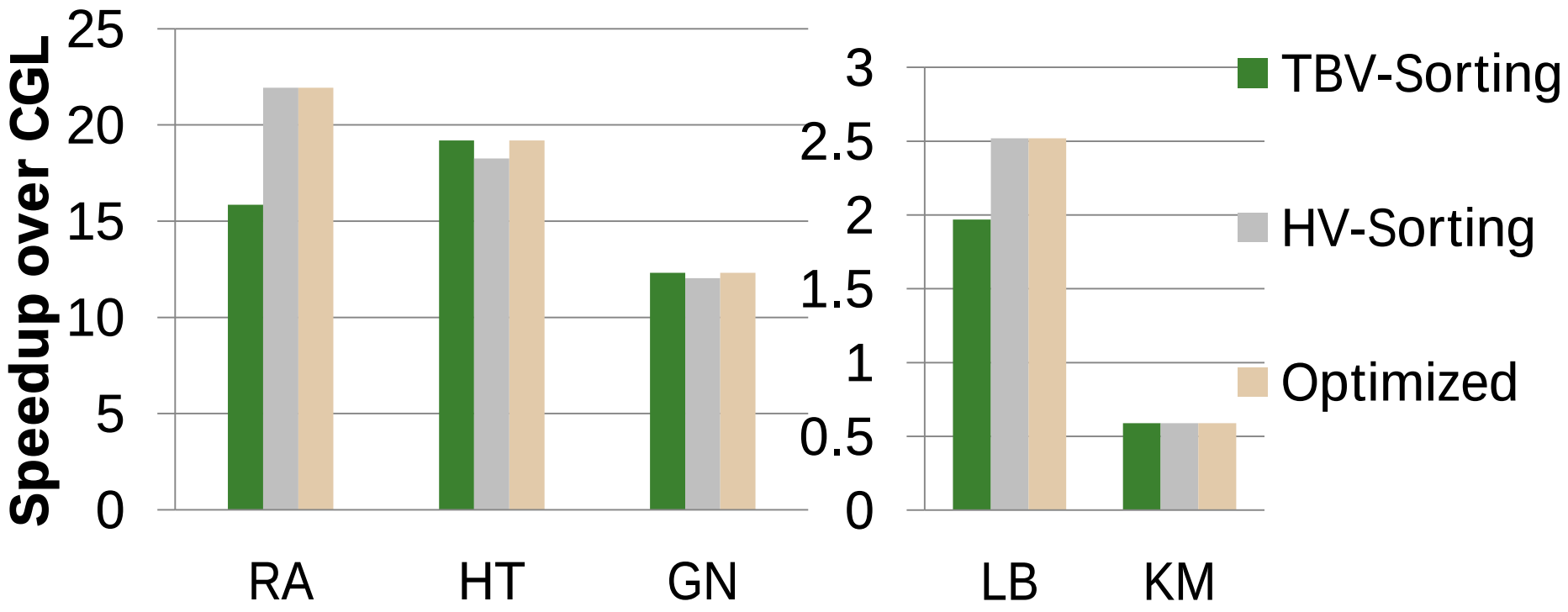
Performance Results



Performance Results



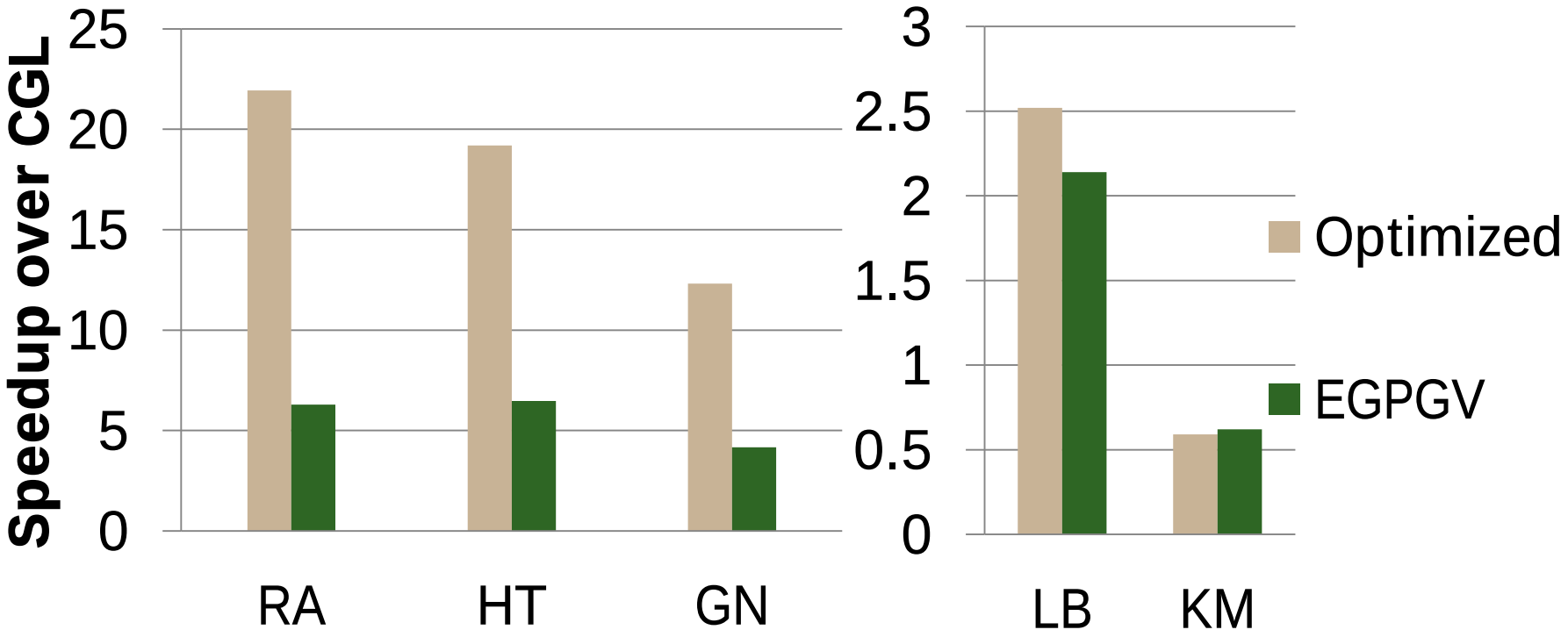
Performance Results



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

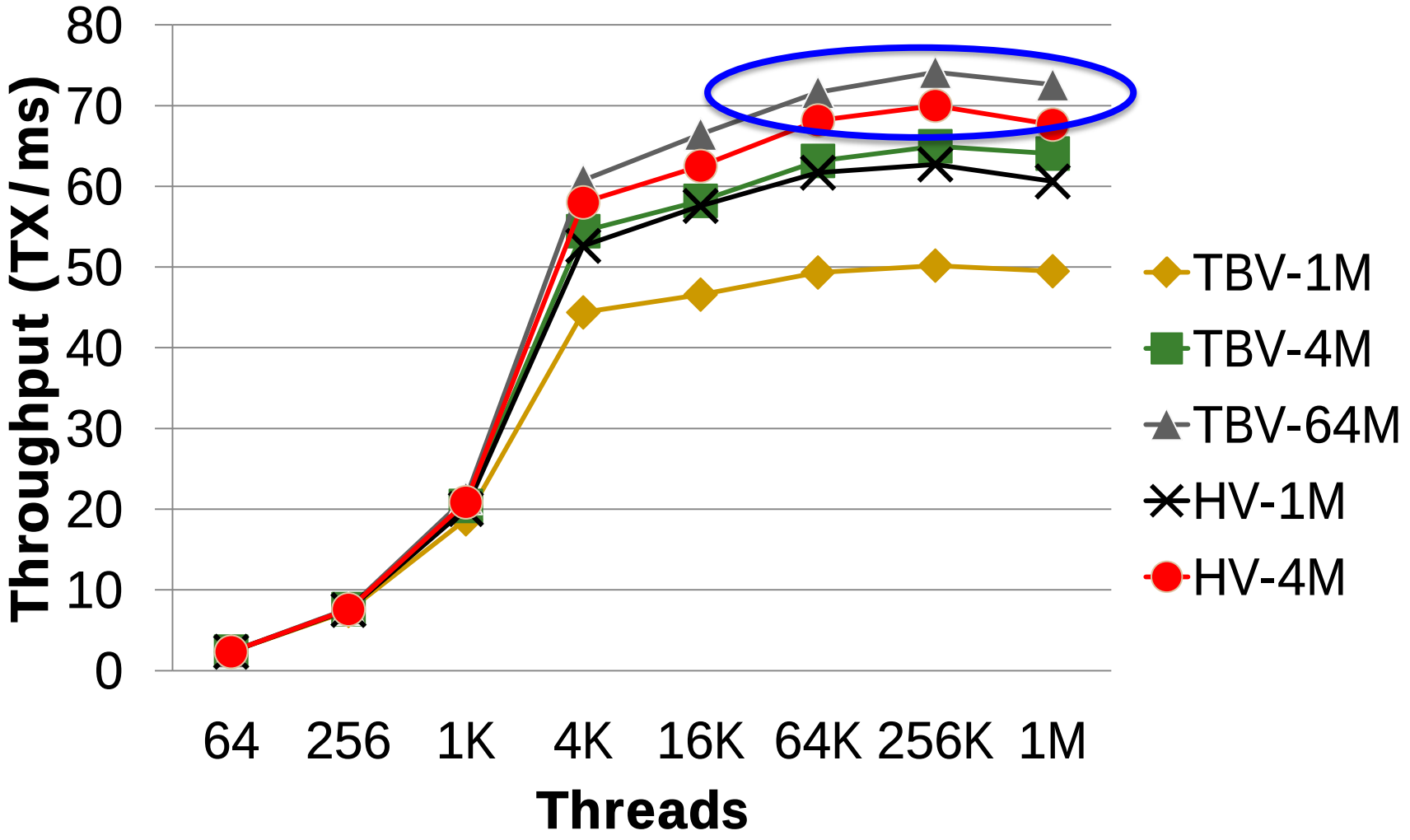
- When amount of shared data < amount of locks, TBV
- Otherwise, HV

Performance Results



EGPGV STM [Cederman et al. EGPGV'10]

Hierarchical Validation vs. Time-based Validation



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[1] Rui Wang[2] Nilanjan Goswami[3]

Tao Li[3] Lan Gao[2] Depei Qian[1, 2]

[1] *Xi'an Jiaotong University, China*

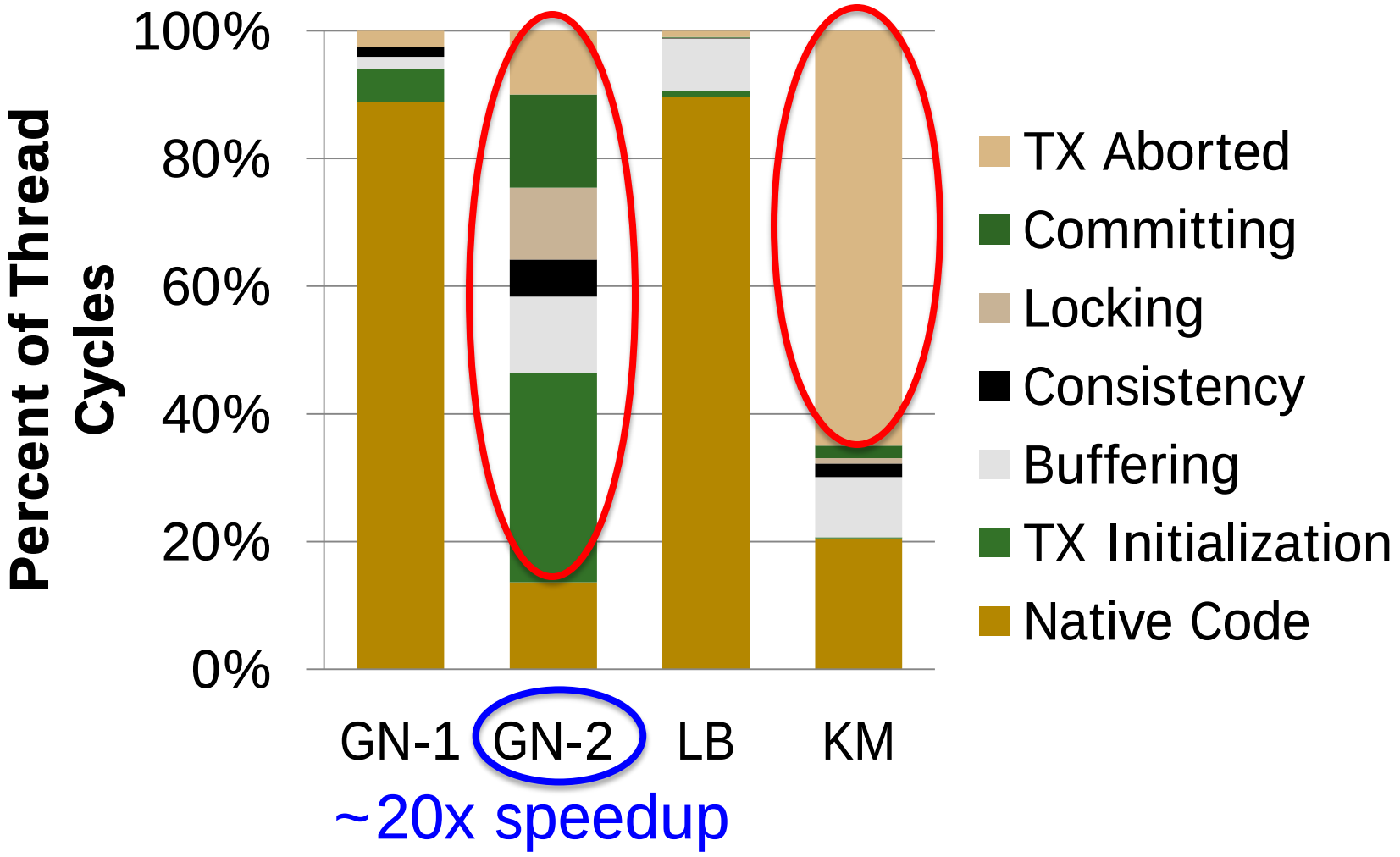
[2] *Beihang University, China*

[3] *University of Florida, USA*

Launch Configurations of Workloads

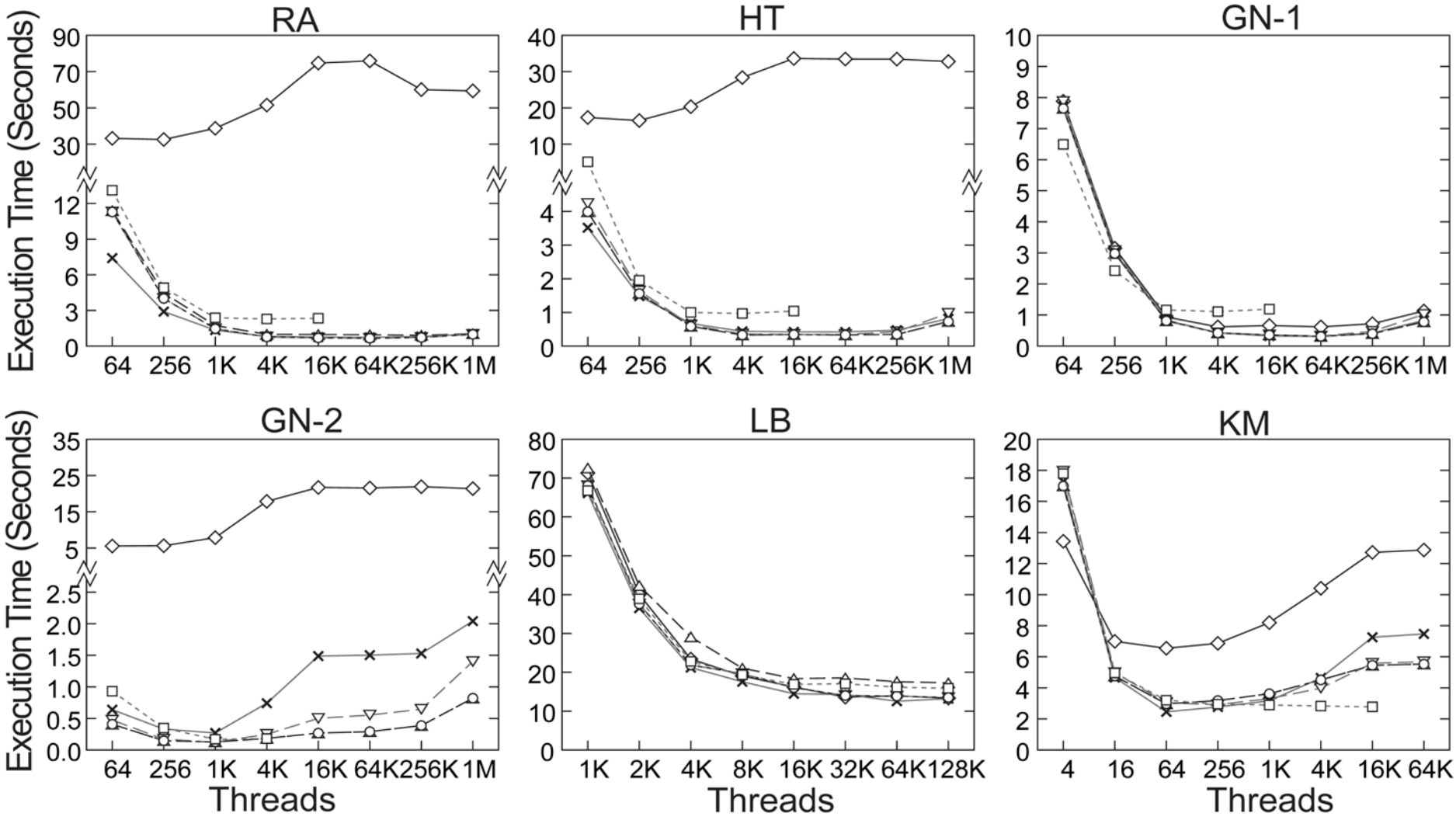
	RA	HT	GN-1, GN-2	LB	KM
Thread-blocks	256	256	256, 16	512	64
Threads per Block	256	256	256, 64	256	4

Execution Time Breakdown



Tradeoff: STM overhead vs. scalability enabled

Scalability Results



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]
  - ...