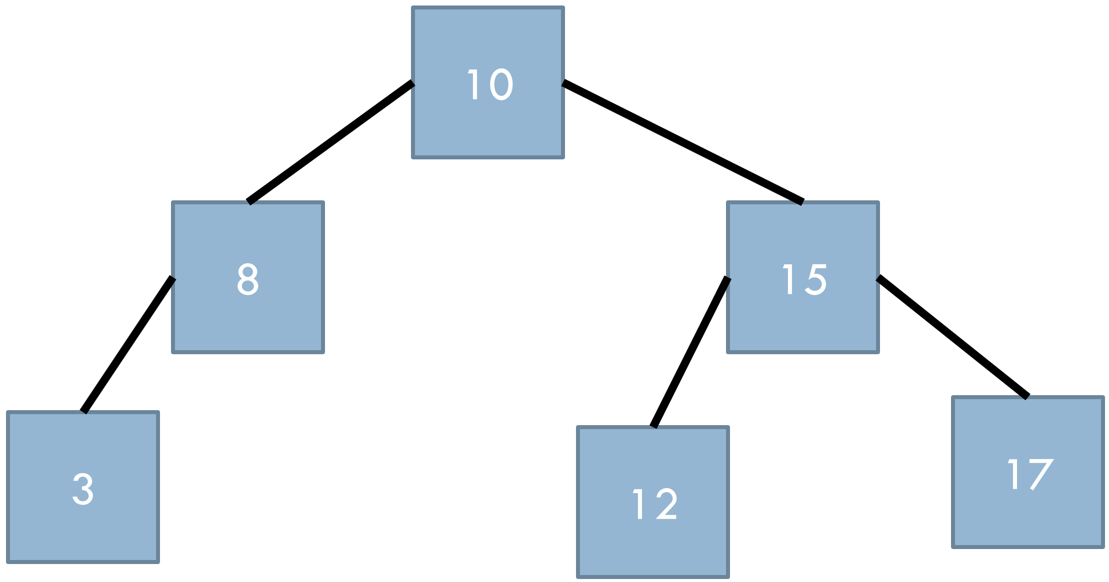
TOWARDS A SOFTWARE TRANSACTIONAL MEMORY FOR GRAPHICS PROCESSORS

Daniel Cederman, Muhammad Tayyab Chaudhry, Philippas Tsigas

Introduction

Software Transactional Memory

- We want to locate an element in a binary balanced tree
- The problem is, some other process is rebalancing it



Software Transactional Memory

STMs provides a construct that guarantees that the enclosed code will be executed atomically

```
atomic
{
  find position in tree
  insert element
  rebalance if necessary
}
```

Software Transactional Memory

- One lock
  - No concurrency
  - Busy waiting
  - Convoying
- Multiple locks
  - Better concurrency
  - Difficult
  - Static analysis

Software Transactional Memory

- Dynamic locks
  - Locks are assigned to words, objects, … and are acquired when data at these locations are read and/or written to
  - Could be acquired directly or at the end of transaction
  - In case of conflict - abort
    - Keep log of reads/writes
    - Keep undo log
- Dynamic locks with helping
  - Removes the need for busy waiting

Software Transactional Memory

- Efficiency is an issue
- Might get better with hardware support

- How does it fare on graphics processors?

Graphics Processors

- Many-core
- SIMD Instructions
  - Single Instruction Multiple Data
- Small or no cache
- High memory bandwidth
- Thousands of threads

CUDA

- Programming platform for NVIDIA graphics processors
- C/C++ based language extended to support executing functions on the graphics processors instead of CPU

CUDA

- Small processor-local memory
- 8-word SIMD instruction
- Coalesced memory access
  - Multiple memory accesses merged into one larger
- No stack – functions inlined

Implementations

Two STMs

- Blocking STM
  - Simpler, and potentially more efficient, if locks are held only for a very short time
  - No recursion needed
- Non-blocking STM
  - T. Harris and K. Fraser "Language support for lightweight transactions", OOPSLA 2003
  - One transaction will always be successful
  - Protected against poor scheduling
  - No busy waiting

Differences

- Blocking
  - Transactions that fail to acquire a lock are aborted
    - Avoids deadlocks
  - A set of locks are shared between objects
    - Provides a middle ground between having just one lock and having one for each object
- Non-blocking
  - Transactions that fail to acquire a lock can help the other transaction commit or abort it
    - Guarantees that one transaction can make progress
  - Each object has its own lock

Common Features

- Object based
  - Coalesced reads and writes are encouraged
- Updates are kept local until commit time
  - Avoids the problem of handling an inconsistent view of the memory
- The memory is only locked at commit time
  - An optimistic approach. Could delay the time taken to discover conflicts

Common Features

- Minimal use of processor local memory
  - Better left to the main application
- SIMD instruction used where possible
  - Mostly used to coalesce reads and writes

Experiments

Contention levels

- We performed the experiments using different contention levels
- One with zero wait time between transactions
- And one with around 500ms of work randomly distributed between transactions

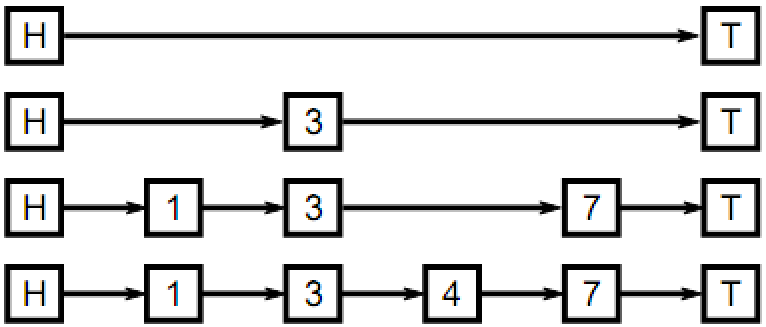
```
while(…)
{
   wait(rand()%max)
   do_operation()
}
```

Backoff

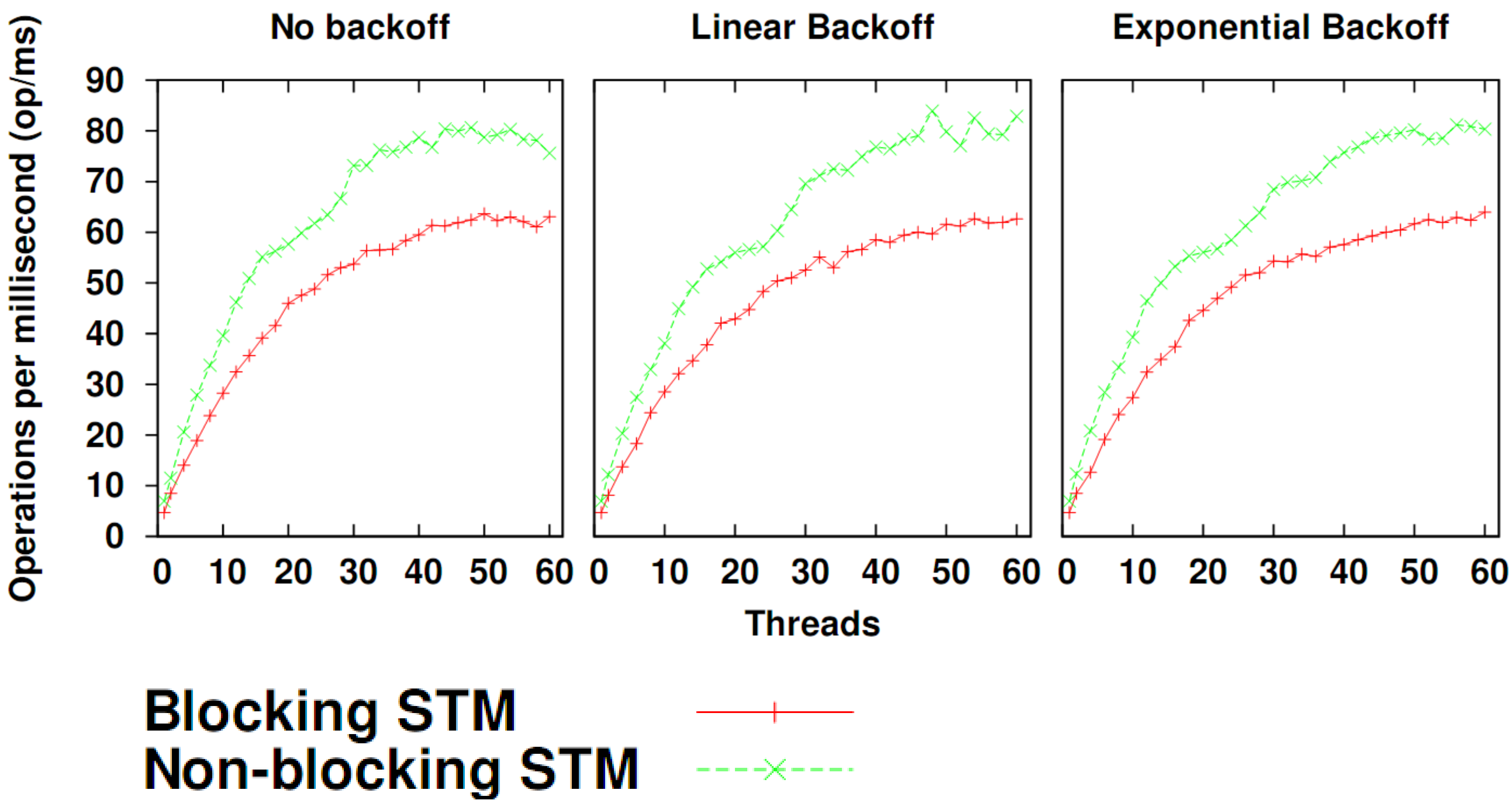
- Lowers contention by waiting before aborted transactions are tried again
- Increases the probability that at least one transaction is successful
- Different types
  - None/static
  - Linear
  - Exponential

Skip-list

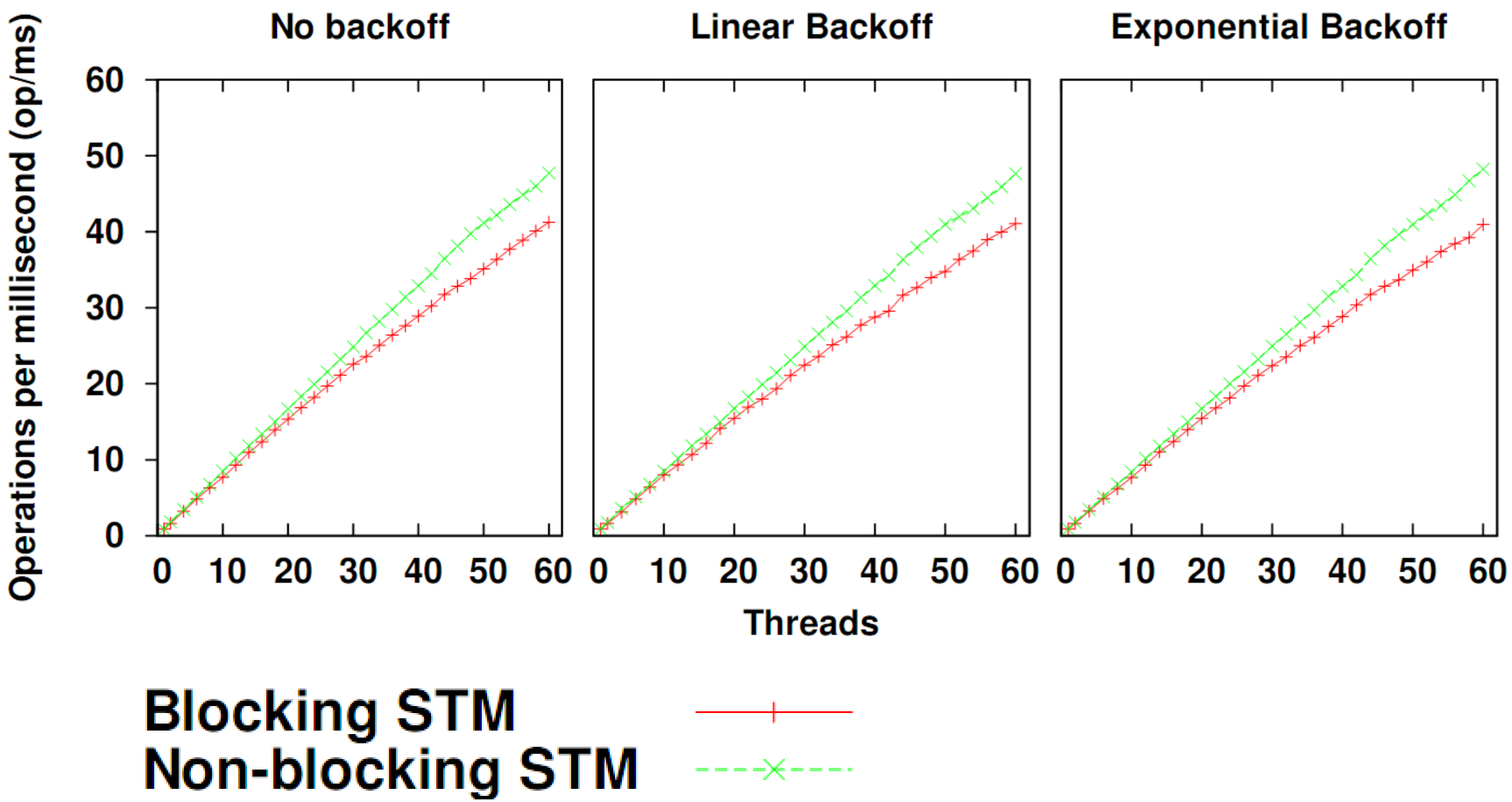
- GTX 280 – 30 multiprocessors
- 1-60 threads
- Even distribution of inserts/lookups/removes



Skip-List – High Contention

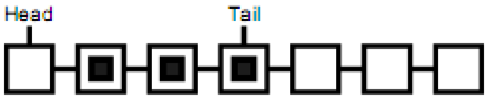


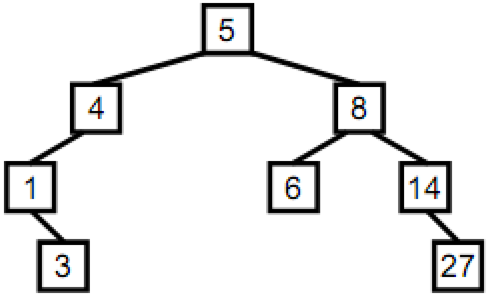
Skip-List – Low Contention

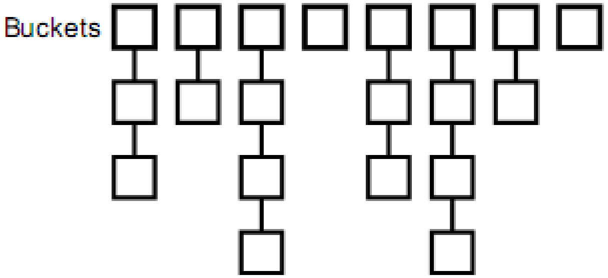


Experiments

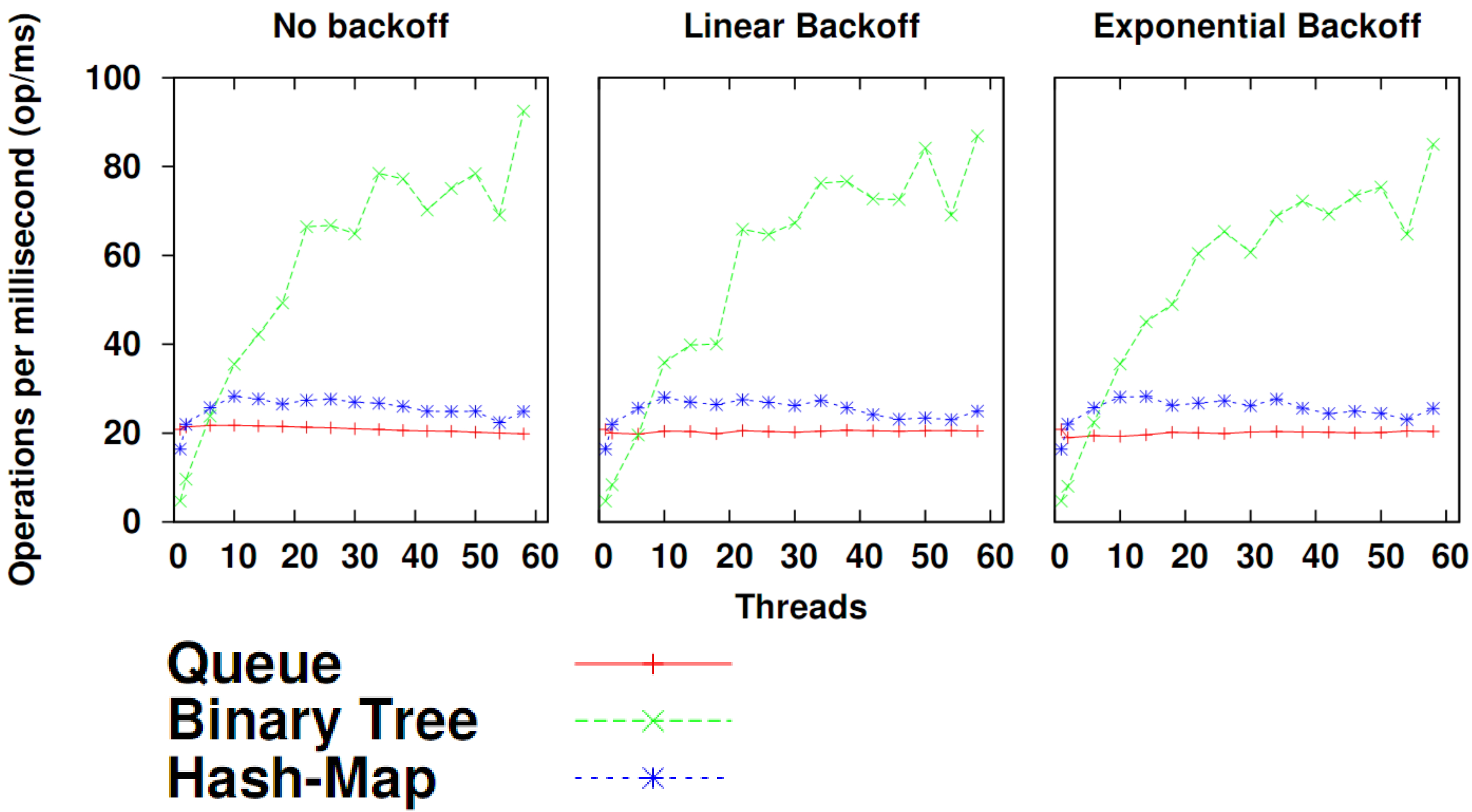
- Queue
- Binary Tree
- Hash-map



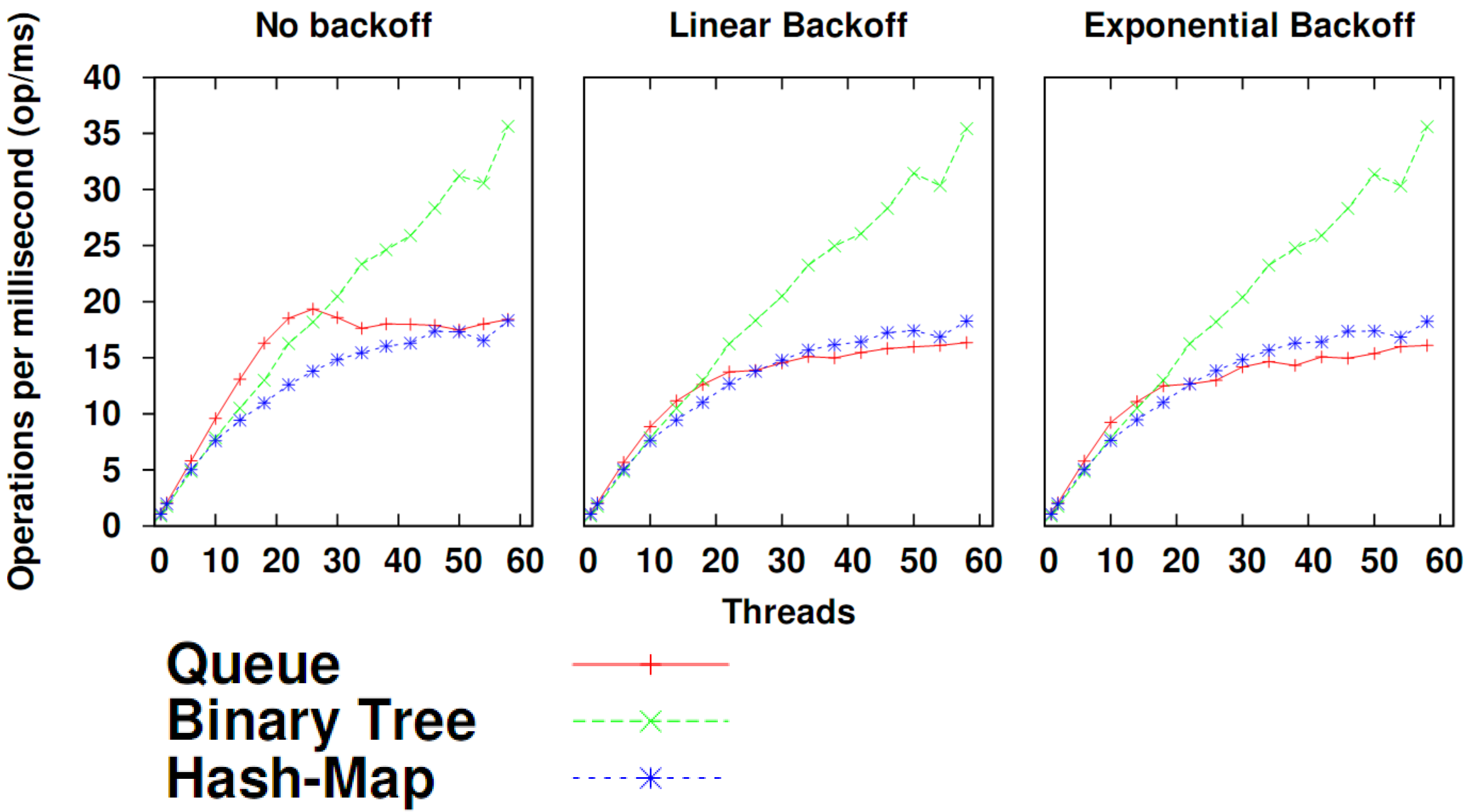




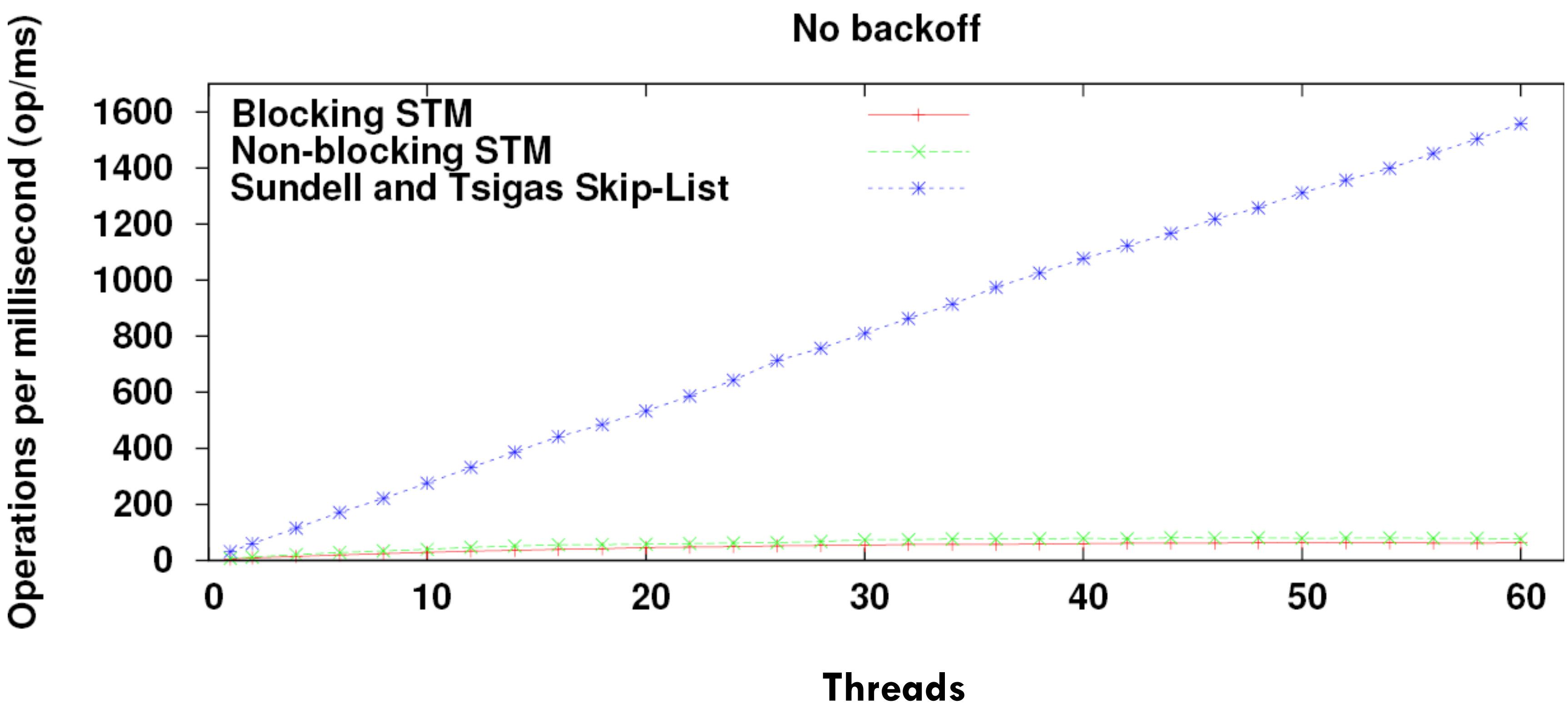
Results - High Contention



Results - Low Contention



Lock-free Skip-List



Conclusion

- Software Transactional Memory has attracted the interest of many researchers over the recent years
- We have tested a blocking and a non-blocking STM on a graphics processor. This is, to the best of our knowledge, the first time this has been done
- The performance behavior was comparable to results from conventional processors
- We now have a basis to build on for further analysis

Thank you!

For more information:

http://www.cs.chalmers.se/~dcs