# What Scalable Programs Need from Transactional Memory

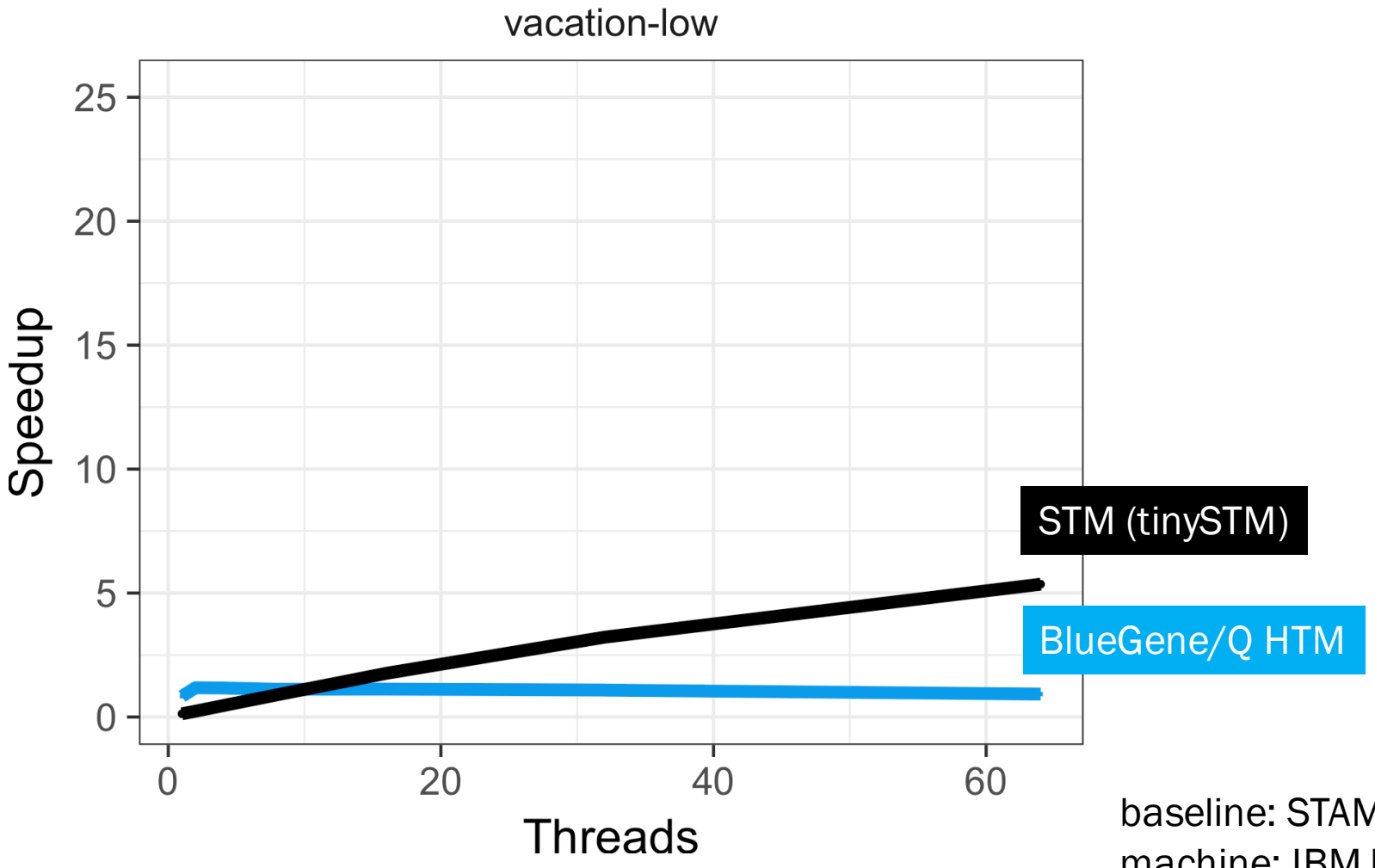
Donald Nguyen
d.nguyen@synthace.com

Keshav Pingali
pingali@cs.utexas.edu

# Transactional Memory

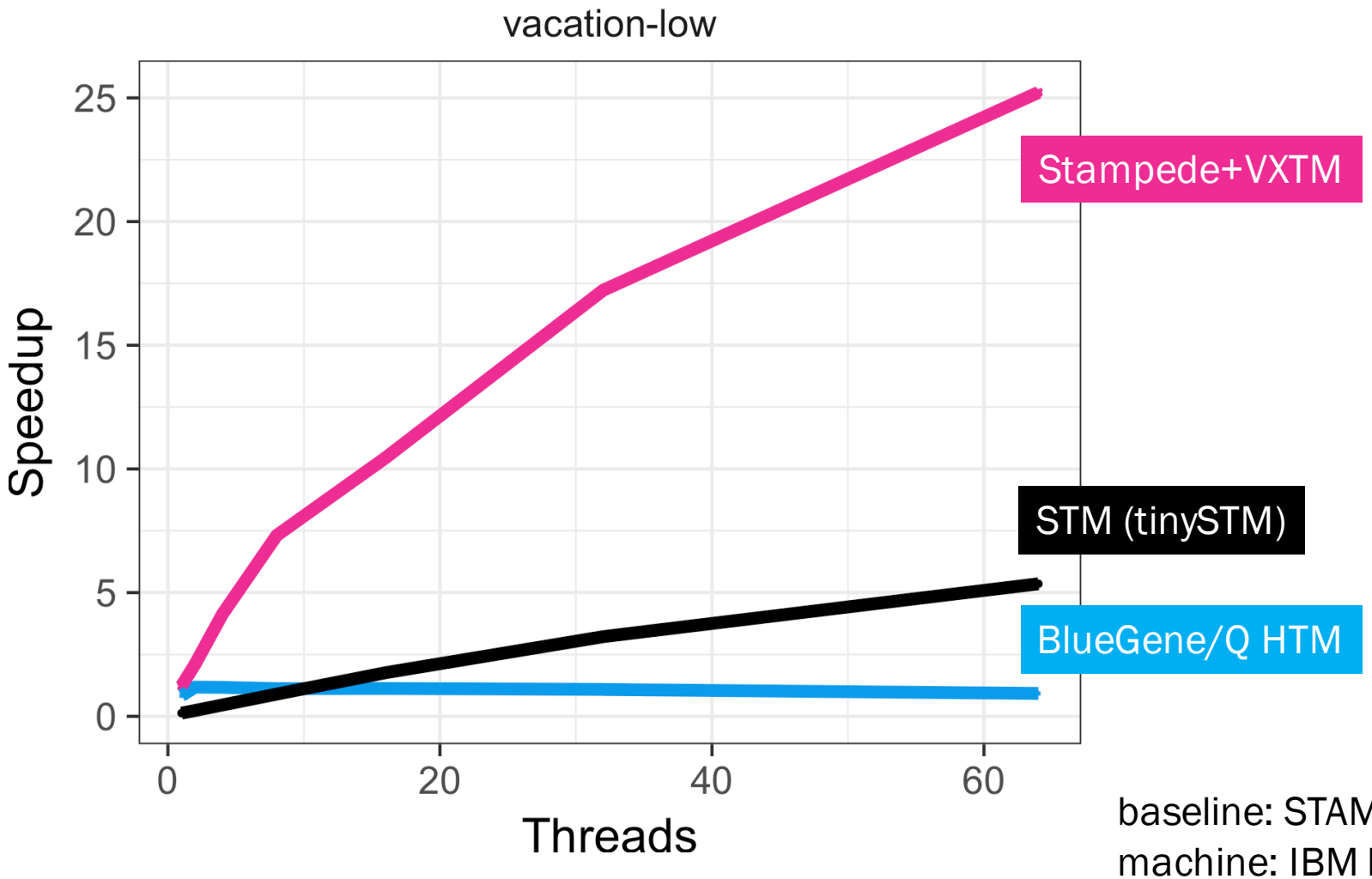
- Atomic and isolated execution for arbitrary code by tracking memory locations
  - In software (STM) and now in hardware (HTM)
- Studies typically use microbenchmarks or STAMP suite
- Topic of active research for a decade

# The Performance to Date




baseline: STAMP (t=1)
machine: IBM Blue Gene/Q

# The Performance to Date




baseline: STAMP (t=1)
machine: IBM Blue Gene/Q

# Why?

Conceptually, most STAMP benchmarks should be moderately scalable

- String matching
- Clustering
- Packet filtering
- Graph construction
- Accessing a relational database
- Triangular mesh refinement

Failure lies in STAMP programming model

# STAMP Programming Model

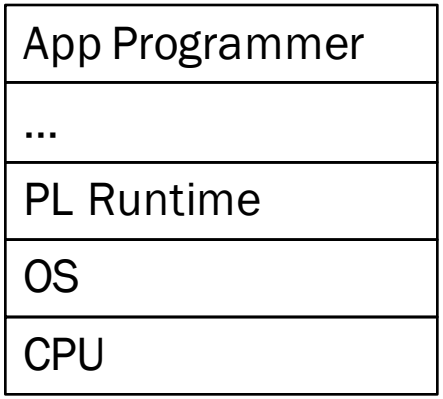
```
thread_begin
  ...
  txn_begin
    ... memory accesses ...
  txn_end
  ...
thread_end
```

# STAMP Programming Model

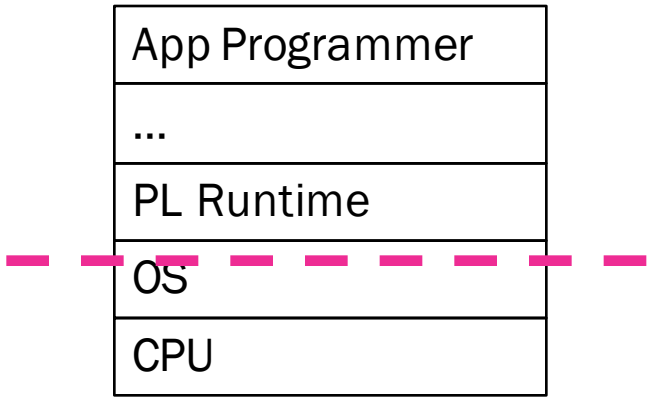
```
thread_begin
  ...
  txn_begin
    ... memory accesses ...
  txn_end
  ...
thread_end
```

- TM protects access to data structures
  - Application data: data structure may not be scalable
  - Worklists: semantics too strict
- Transactions are tied to threads

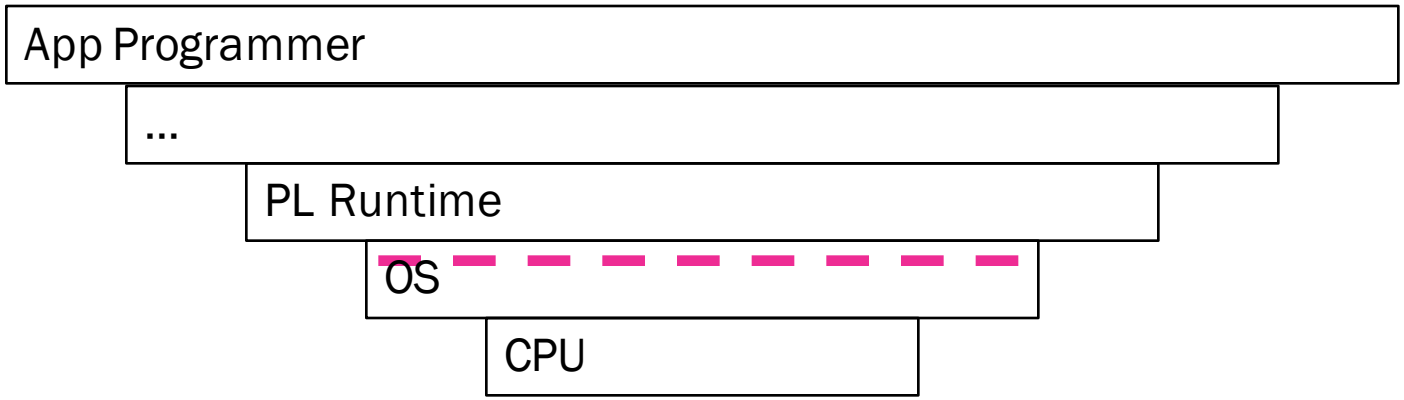
# Scalable Programs and Structured Parallelism



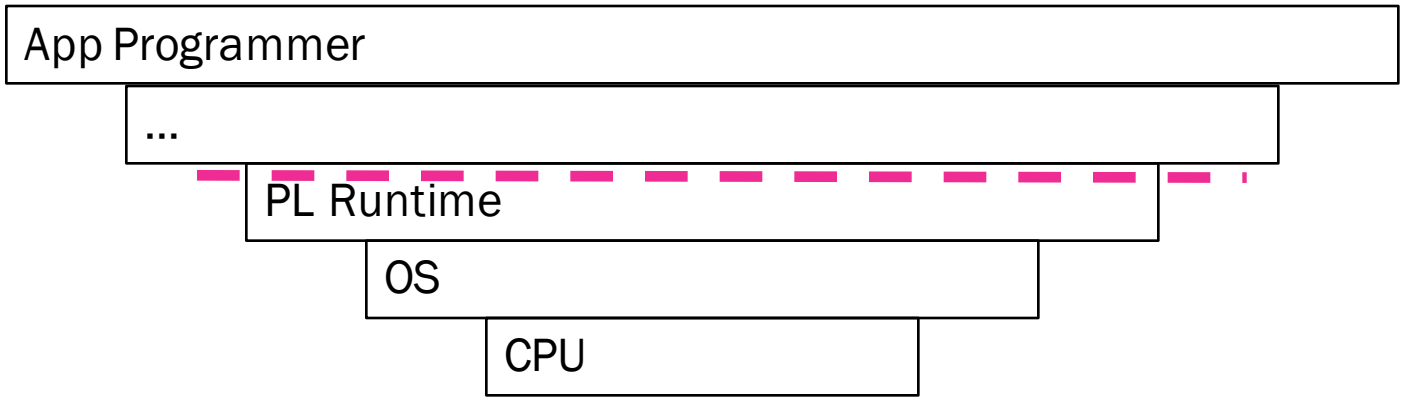
# Scalable Programs and Structured Parallelism



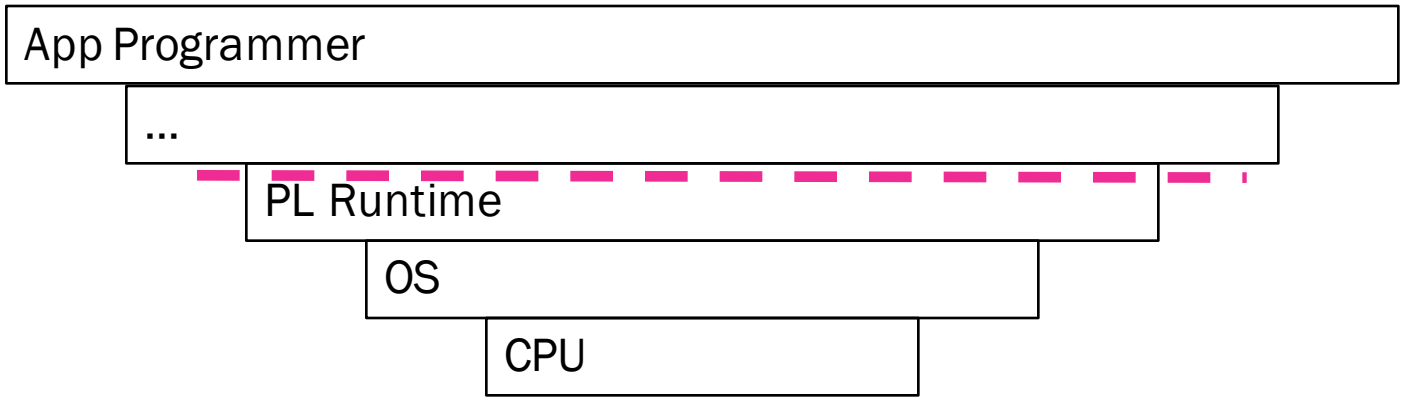
# Scalable Programs and Structured Parallelism



# Scalable Programs and Structured Parallelism



# Scalable Programs and Structured Parallelism




STAMP PM

- Ad-hoc //ism
- Arbitrary data structures

Stampede PM

- Amorphous data //ism [PLDI 2011]
  - Dynamic task generation
  - Tasks may conflict
- Composition of scalable data structures

# Stampede Principles

## Disjoint Access

Transactions disjoint at logical level should be disjoint at physical level.

## Virtualized Transactions

Transactions should be decoupled from threads.

# Disjoint Access

Replacing data structures with scalable alternatives

| | | |
|---|---|---|
| red-black tree | => | hashtable |
| hashtable | => | reduction over hashtables |
| dynamic buffer | => | per-task buffer |

# Disjoint Access

Replacing data structures with scalable alternatives

| | | |
|---|---|---|
| red-black tree | => | hashtable |
| hashtable | => | reduction over hashtables |
| dynamic buffer | => | per-task buffer |

Using scheduling-specific data structures

| | | |
|---|---|---|
| linked-list | => | workstealing scheduler |
| counter | => | workstealing scheduler |
| dynamic array | => | workstealing scheduler |

# Virtualized Transactions

Schedule transactions as tasks

- Execute on any thread in any order
- Exploit runtime information to increase throughput
  - E.g., serialize conflicting tasks
- Use high-performance scheduling data structures

Code changes can be automated

- E.g., compiler support for anonymous functions

# XTM

If most transactions access disjoint data, is exclusive conflict detection sufficient?

XTM Implementation

- CAS owner field on transactional access
- Add to list of owned objects
- Reset field on transaction commit or abort

# Stampede

Stampede: STAMP transformed to satisfy scalability principles

- Code 88% same as STAMP

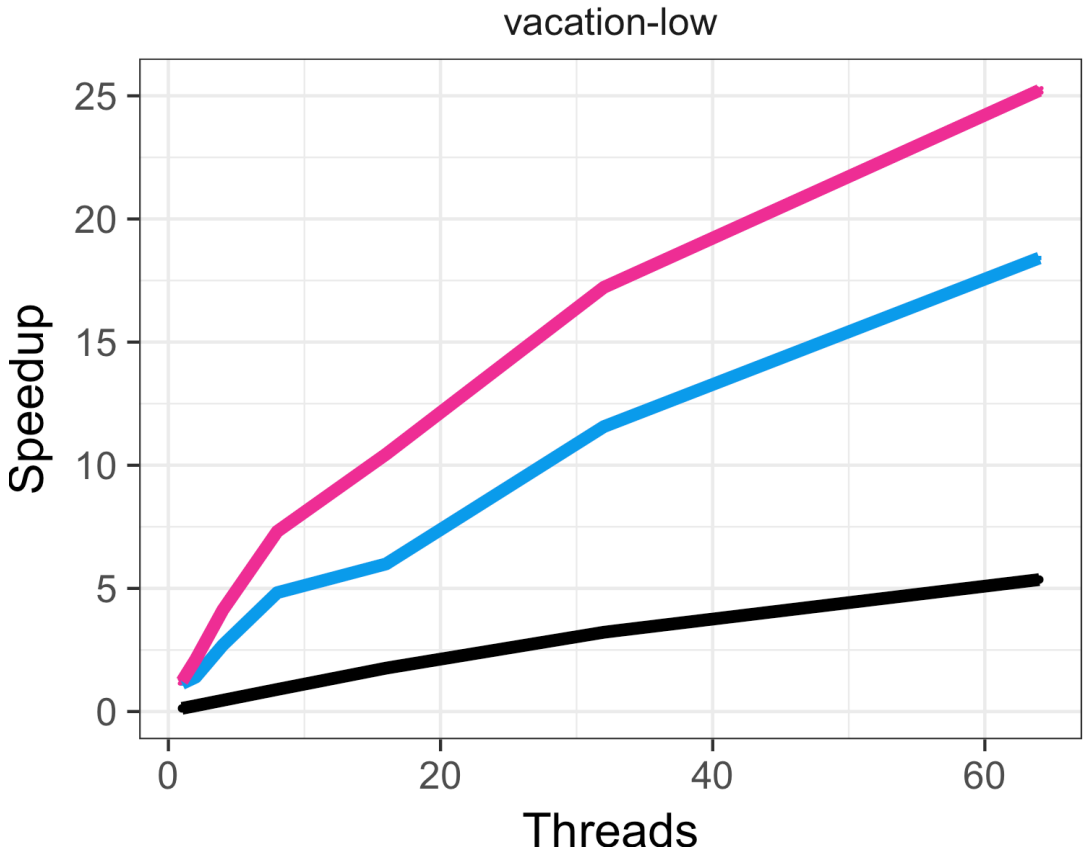
# Stampede

**Stampede:** STAMP transformed to satisfy scalability principles

- Code 88% same as STAMP

| Variant | Code | Conflict Detection | Reschedule Transaction? |
|---|---|---|---|
| STAMP+STM | STAMP | tinySTM | No |
| Stampede+VTM | Stampede | tinySTM | Yes |
| Stampede+VXTM | Stampede | XTM | Yes |

# Results




Stampede+VXTM

(XTM and rescheduling txns)

Stampede+VTM

(tinySTM and rescheduling txns)

STAMP+STM

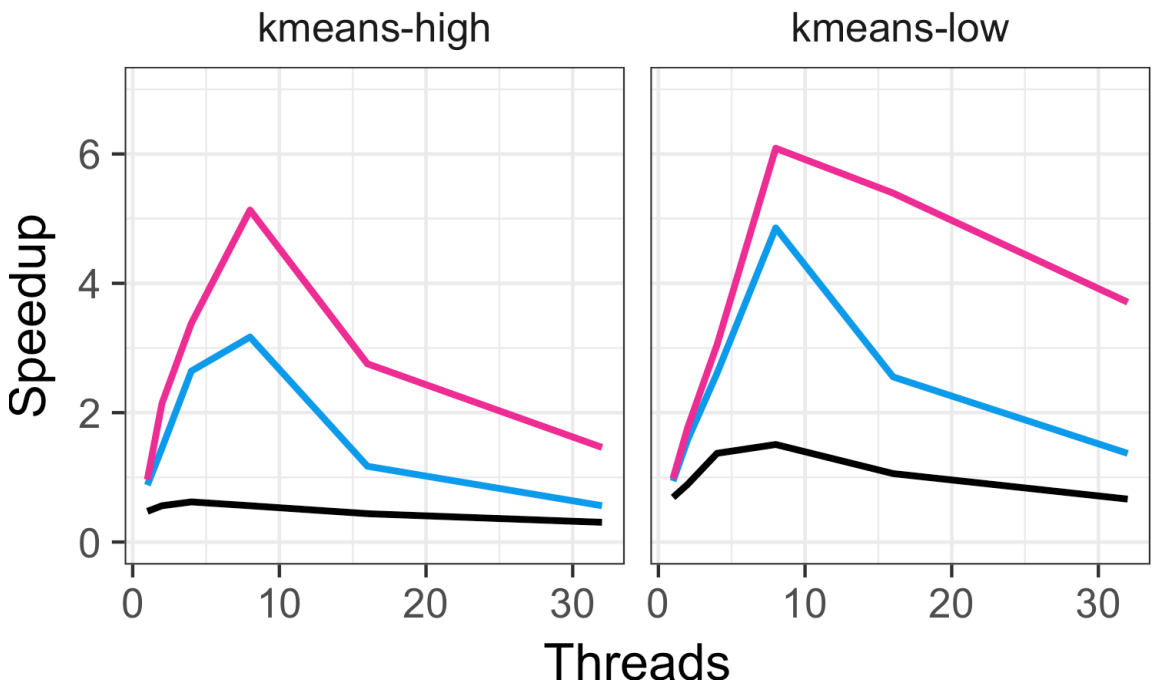
baseline: STAMP (t=1)
machine: Blue Gene/Q

# Results

| | Median Speedup |
|---|---|
| STAMP+STM | 8.0X |
| Stampede+VXTM | 17.7X |

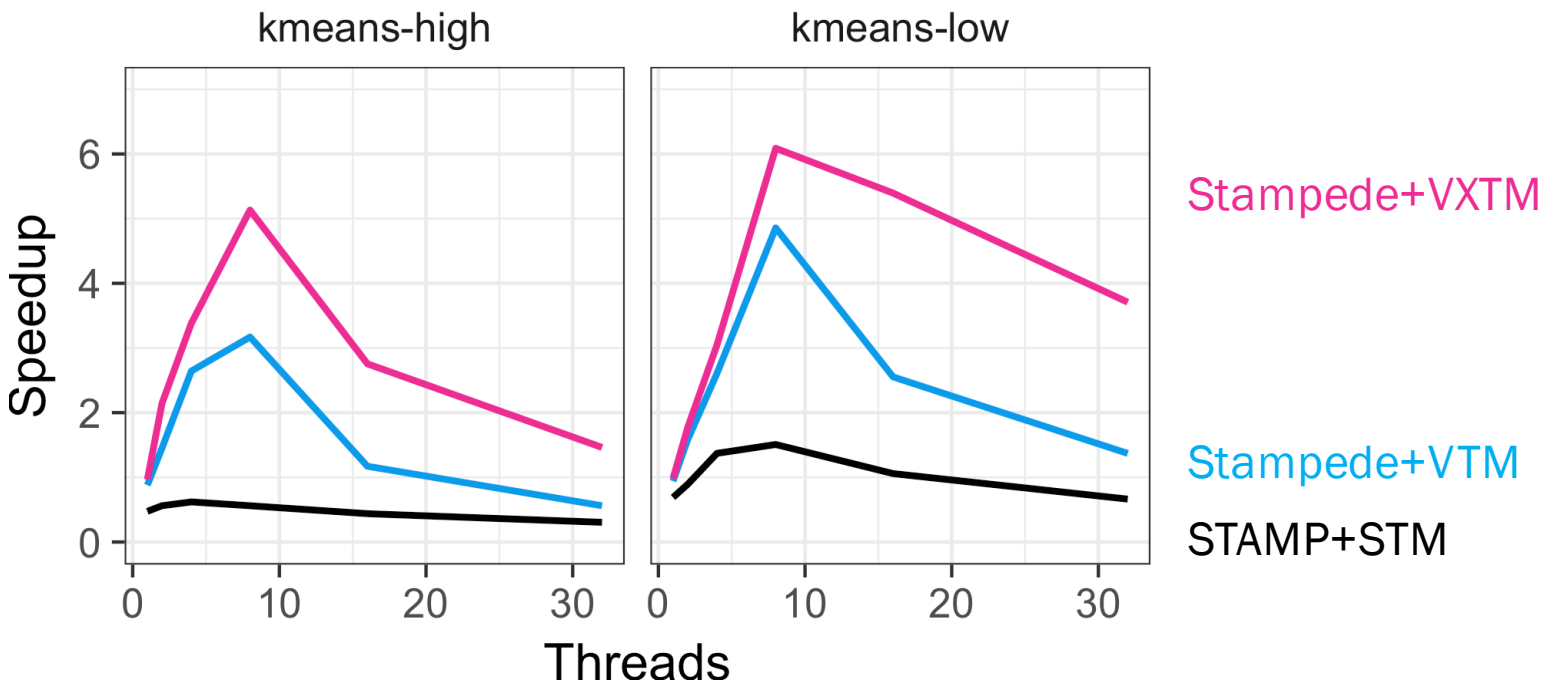
baseline: STAMP (t=1)
machine: Blue Gene/Q

# Contention




baseline: STAMP (t=1)
machine: Westmere

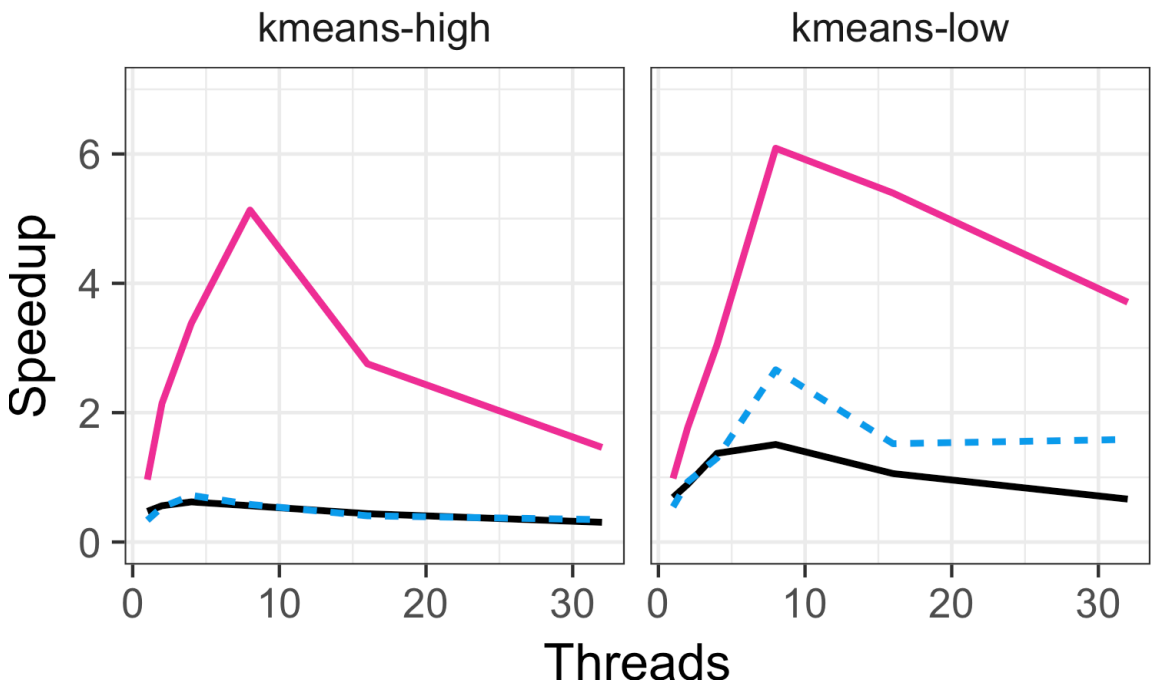
# Contention




- Under high contention, performance still suffers

baseline: STAMP (t=1)
machine: Westmere

# XTM



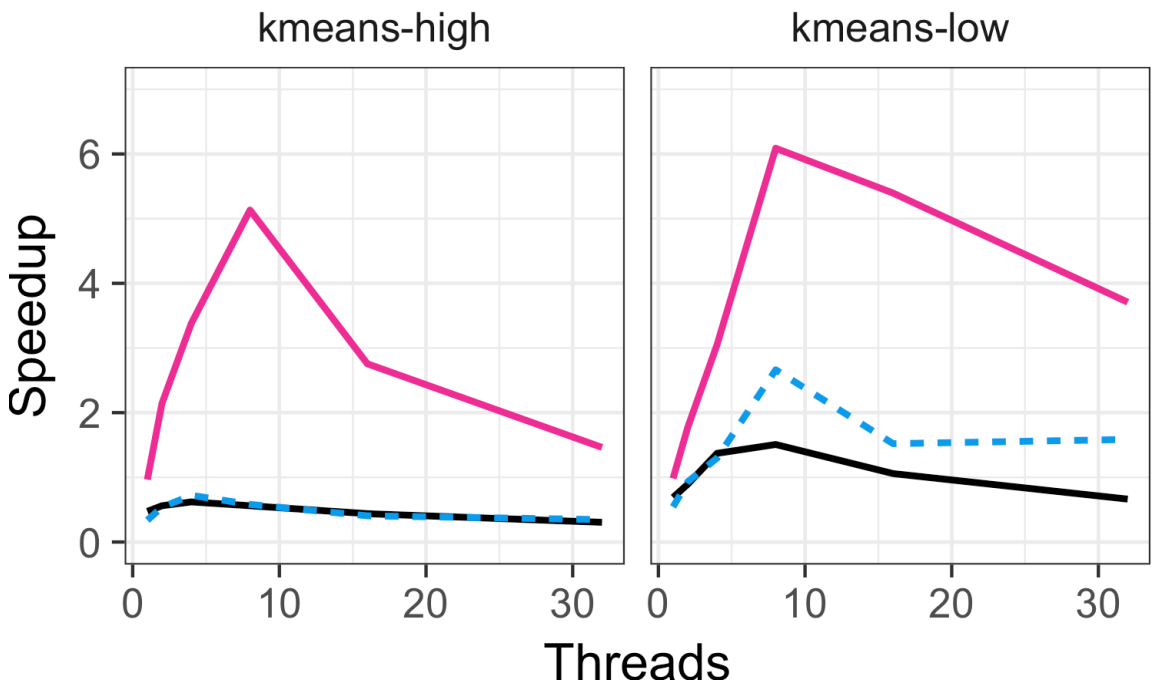

Stampede+VXTM

STAMP+XTM

(STAMP with XTM)

STAMP+STM

# XTM




Stampede+VXTM

STAMP+XTM

(STAMP with XTM)

STAMP+STM

XTM alone is not enough

# Conclusion

Stampede+VXTM gives 4X improvement over STAMP+STM/HTM

- Use scalable data structures
  - Reduce conflicts wherever possible
- Exploit scheduling optimizations
  - Do useful work even when conflicts occur
- Simplify TM to VXTM

Is it time to rethink TM interface?

- Division of labor between application, runtime and hardware
- RISC versus CISC

http://iss.ices.utexas.edu/?p=projects/stampede