XJ

Closed, Open Nested and Boosted Atomic Actions for Java

Keith Chapman
Purdue U

Tony Hosking
ANU/Data61, Purdue U

Eliot Moss
UMass

Overview

- XJ Language
  - Support for flat and closed/open/boosted nested transactions
- Implementation
- Results
- Support for HTM
- Current status

closedatomic class

```
public closedatomic class ClosedMap<K, V> implements Map<K, V> {
  private final Map<K, V> map;

  public ClosedMap(Map<K, V> map) { this.map = map; }

  // Implemented methods of the Map interface
  public V put(K key, V val) {
    return map.put(key, val);
  }

  public int size() {
    return map.size();
  }

  ⋮
}
```

closedatomic methods / blocks

```
public closedatomic Object putIfAbsent(K key, V val) {
    if (!containsKey(key))
        return put(key, val);
    else
        return get(key);
}
```

```
public Object putIfAbsent(K key, V val) {
    closedatomic {
        if (!containsKey(key))
            return put(key, val);
        else
            return get(key);
    }
}
```

Abstract locks

- Locking framework consists of
  - lock object: represents an abstract lock
  - lock space: defines which lock shapes can be requested
  - lock shape: indicates *what* is being locked
  - lock mode: indicates *how* it is being accessed

Example lock shapes

- Consider an openatomic class OrderedSet<T> implementing java.util.SortedSet<T>
- Several suitable lock shapes,
  - point(x): locking single “point” objects,
  - GT(x): locking upward “rays” starting at x
  - LT(x): locking downward “rays” starting at x
  - Range(x,y): locking ranges defined on values x and y

Example lock modes

```
public enum SXLockModes implements LockMode<SXLockModes> {
  SHARED {
    public boolean conflictsWith(SXLockModes other)
    { return other != SHARED; }
  },
  EXCLUSIVE {
    public boolean conflictsWith(SXLockModes other)
    { return true; }
  }
}

public enum PinChangeLockModes implements LockMode<PinChangeLockModes> {
  PIN {
    public boolean conflictsWith(PinChangeLockModes other)
    { return other != PIN; }
  },
  CHANGE {
    public boolean conflictsWith(PinChangeLockModes other)
    { return other != CHANGE; }
  }
}
```

openatomic class

```
public openatomic class OpenMap<K, V> implements Map<K, V> {
  private final Map<K, V> map;
  private final LockSpace<SXLockModes, K>
    keySpace = new PointSpace<SXLockModes, K>();
  private final LockSpace<PinChangeLockModes, OpenMap<K, V>>
    mapSpace = new SinglePointSpace<PinChangeLockModes, OpenMap<K, V>>();

  public OpenMap(Map<K, V> map) { this.map = map; }

  // Implemented methods of the Map interface in next slide

  ⋮
}
```

openatomic class

```
public openatomic class OpenMap<K, V> implements Map<K, V> {
  private final Map<K, V> map;
  private final LockSpace<SXLockModes, K>
    keySpace = new PointSpace<SXLockModes, K>();
  private final LockSpace<PinChangeLockModes, OpenMap<K, V>>
    mapSpace = new SinglePointSpace<PinChangeLockModes, OpenMap<K, V>>();

  public OpenMap(Map<K, V> map) { this.map = map; }

  // Implemented methods of the Map interface in next slide

  ⋮
}
```

openatomic methods

```
⋮
public openatomic V put(K key, V val)
  [V result = null]
  locking
    (keySpace : point(key) : SXLockModes.EXCLUSIVE),
    (mapSpace : get() : PinChangeLockModes.CHANGE)
{ return result = map.put(key, val); }
onabort
{ if (result == null) map.remove(key); else map.put(key, result); }

public openatomic V get(K key)
  locking
    (keySpace : point(key) : SXLockModes.SHARED)
{ return map.get(key, val); }

public openatomic int size()
  locking (mapSpace : get() : PinChangeLockModes.PIN)
{ return map.size(); }
⋮
```

openatomic methods

```
⋮
public openatomic V put(K key, V val)
  [V result = null]
  locking
    (keySpace : point(key) : SXLockModes.EXCLUSIVE),
    (mapSpace : get() : PinChangeLockModes.CHANGE)
{ return result = map.put(key, val); }
onabort
{ if (result == null) map.remove(key); else map.put(key, result); }

public openatomic V get(K key)
  locking
    (keySpace : point(key) : SXLockModes.SHARED)
{ return map.get(key, val); }

public openatomic int size()
  locking (mapSpace : get() : PinChangeLockModes.PIN)
{ return map.size(); }
⋮
```

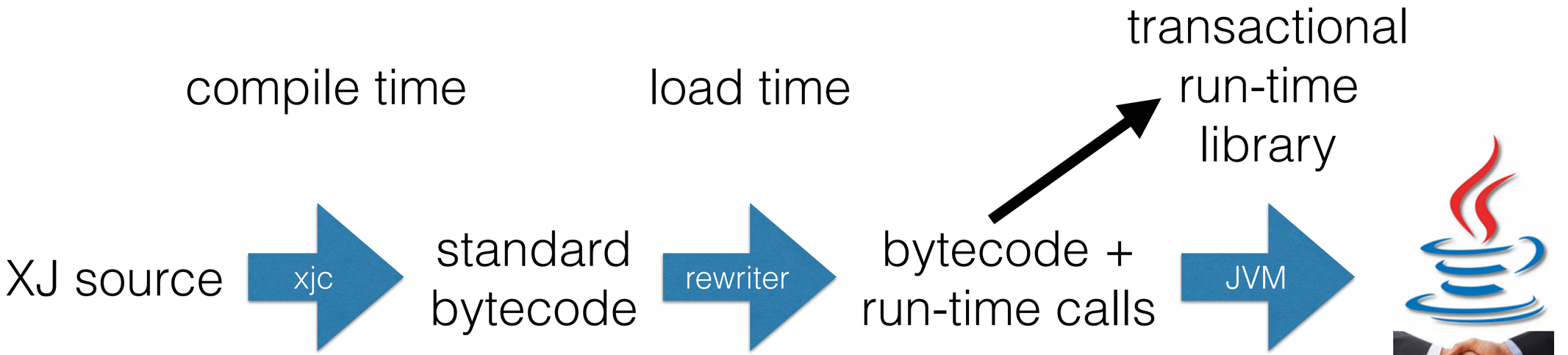
openatomic methods

```
⋮
public openatomic V put(K key, V val)
  [V result = null]
  locking
    (keySpace : point(key) : SXLockModes.EXCLUSIVE),
    (mapSpace : get() : PinChangeLockModes.CHANGE)
{ return result = map.put(key, val); }
onabort
{ if (result == null) map.remove(key); else map.put(key, result); }

public openatomic V get(K key)
  locking
    (keySpace : point(key) : SXLockModes.SHARED)
{ return map.get(key, val); }

public openatomic int size()
  locking (mapSpace : get() : PinChangeLockModes.PIN)
{ return map.size(); }
⋮
```

boostedatomic class

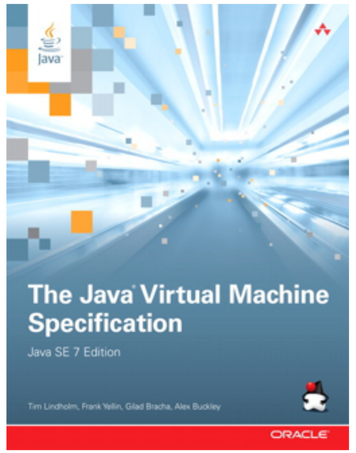
```
public boostedatomic class BoostedMap<K, V> implements ConcurrentHashMap<K, V> {
  private final ConcurrentHashMap<K, V> map;
  private final LockSpace<RWLockModes, K>
    keySpace = new PointSpace<RWLockModes, K>();
  private final LockSpace<SIXLockModes, OpenMap<K, V>>
    mapSpace = new SinglePointSpace<SIXLockModes, OpenMap<K, V>>();

  public BoostedMap(ConcurrentHashMap<K, V> map) { this.map = map; }

  // Implemented methods of the ConcurrentHashMap interface
  public V put(K key, V val) [V result = null] locking …
  public int size() locking …

  ⋮
}
```

XJ system architecture




```
public openatomic V put(K key, V val)
    [V result = null]
    locking
        (keySpace : point(key) :
        SXLockModes.EXCLUSIVE),
        (mapSpace : get() :
        PinChangeLockModes.CHANGE)
{ return result = map.put(key, val); }
onabort
{ if (result == null) map.remove(key);
else map.put(key, result); }
```

```
00101001000100100010010001001000
00100101001010100101000010100010
00100001000010100101000010010001
00100010000101001000101110010100
00101001010101000100000100000101
00010001011100100100000100101000
01000101000100101010100101001000
01101010010000101010100100000010
```



Runtime Library

- Contains important runtime classes
  - TxnObject: adds transaction semantics for (almost) all objects
    - superclass of (almost) all transactional object classes
    - provides the lock/version word
  - TxnArrayWrapper: similarly for arrays
  - TxnDescriptor: transaction context (log, locks, txn id, …)
- Support for locking/logging/undoing
- Collection of pre-defined abstract lock libraries

Benchmarks

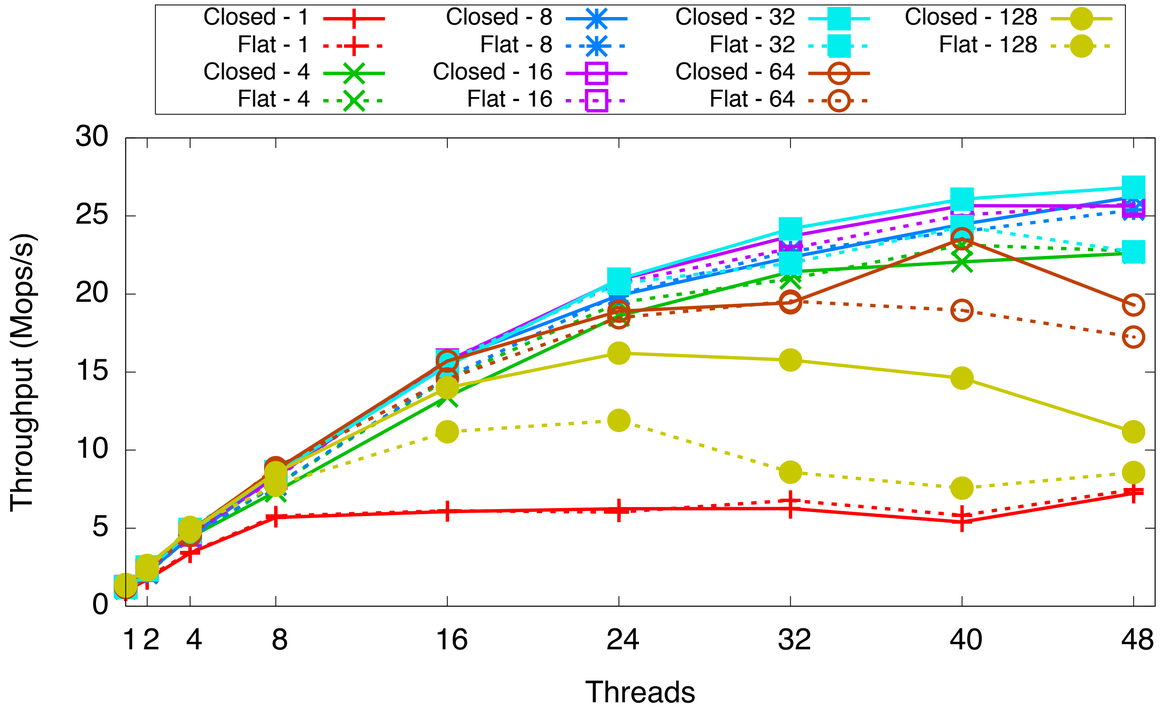
- We use synchrobench
  - A micro-benchmark suite to evaluate synchronization techniques of various data structures
- Added ability to run multiple operations within a single transaction (group size)
- Included XJ versions of the benchmarks
  - TransactionalFriendlyTreeSet
- 48-way, Intel Xeon E5- 2690 v3 machine with 2 sockets of 12 hyperthreaded cores, running at 2.60GHz

Synchrobench

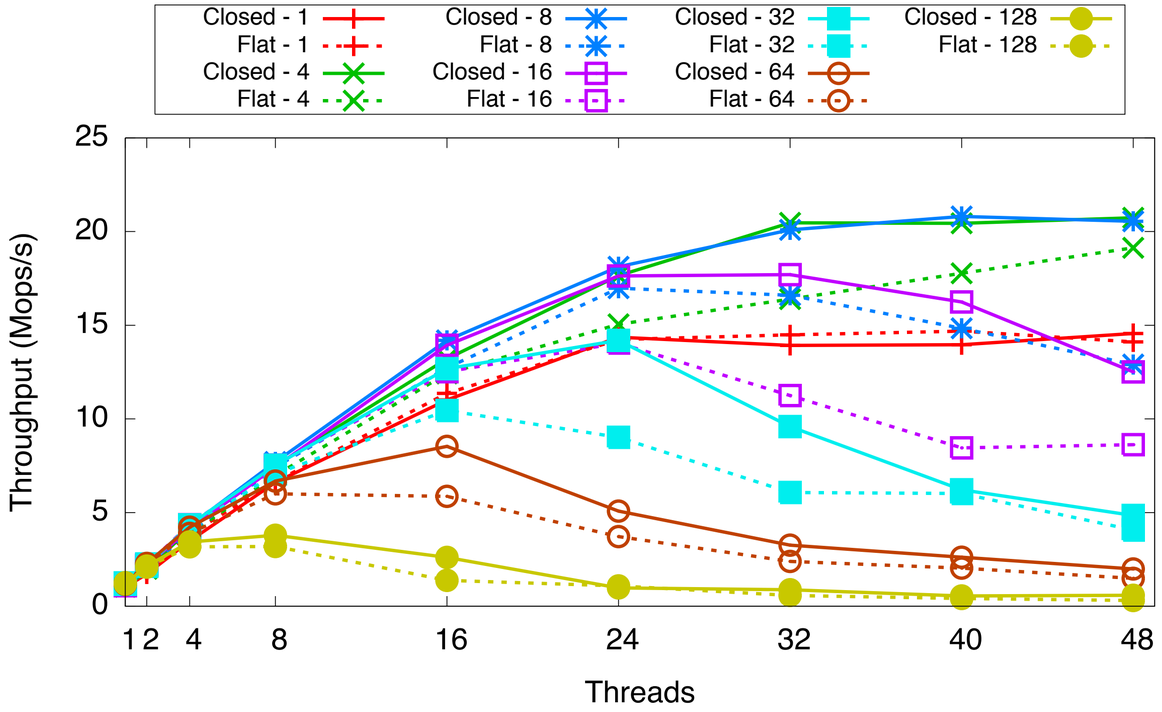
[Gramoli, USyd]

- i = 64K: initial number of elements added to the data structure
- r = 128K: range of possible keys used in the data structure
- u = 0|5|50: % of operations that are updates
- n = 5: number of iterations of the benchmark
- t = 1|2|4|8|16|24|32|40|48: number of application threads
- W = 5: warm up time in seconds
- d = 5000: duration of a single iteration of the benchmark in milliseconds
- g = 1|4|8|16|32|64|128: number of operations to group per transaction

Closed vs Flat

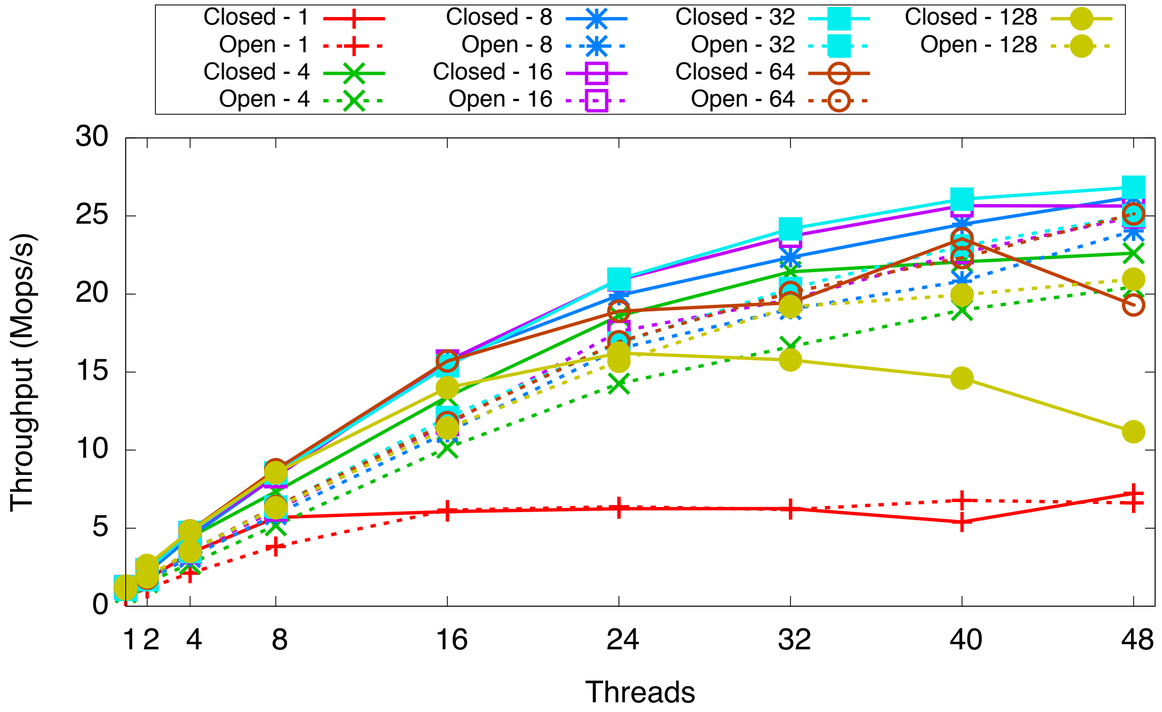


5% Updates

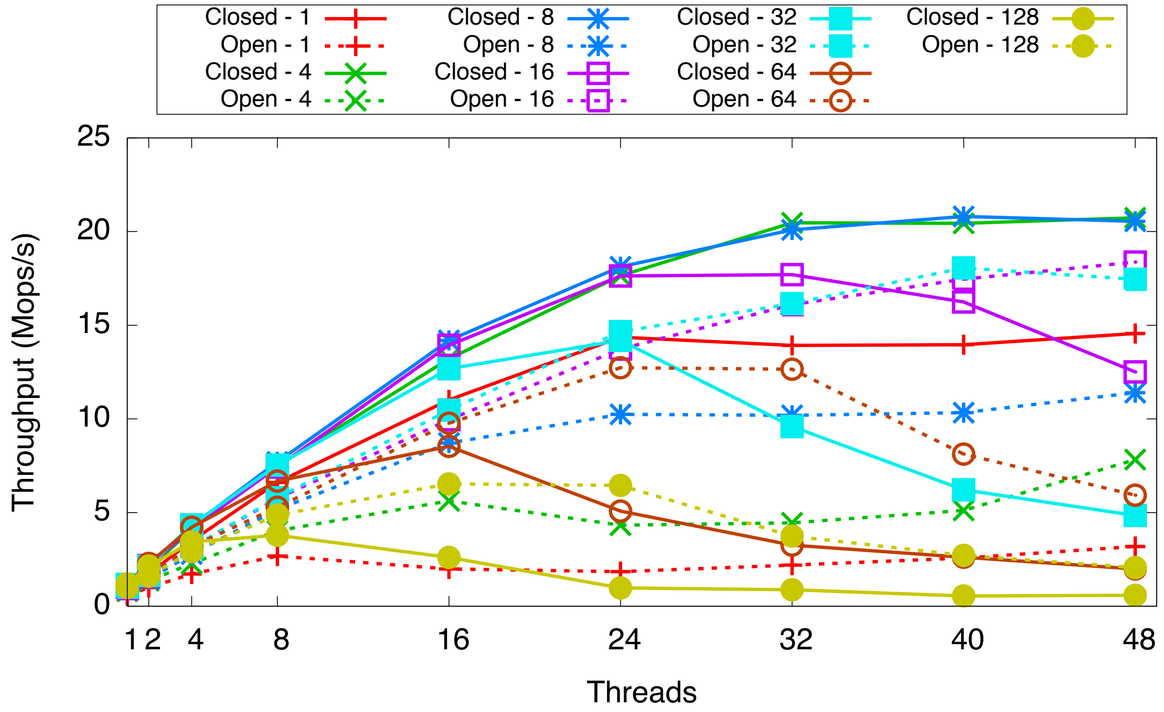


50% Updates

Closed vs Open

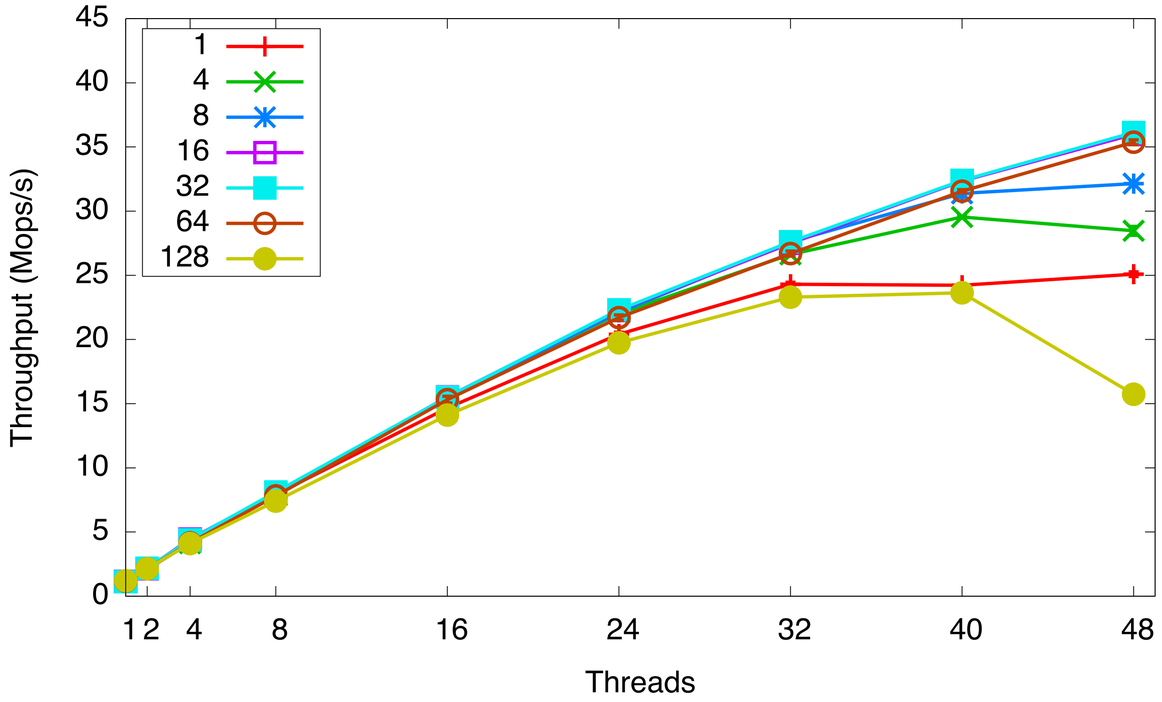


5% Updates

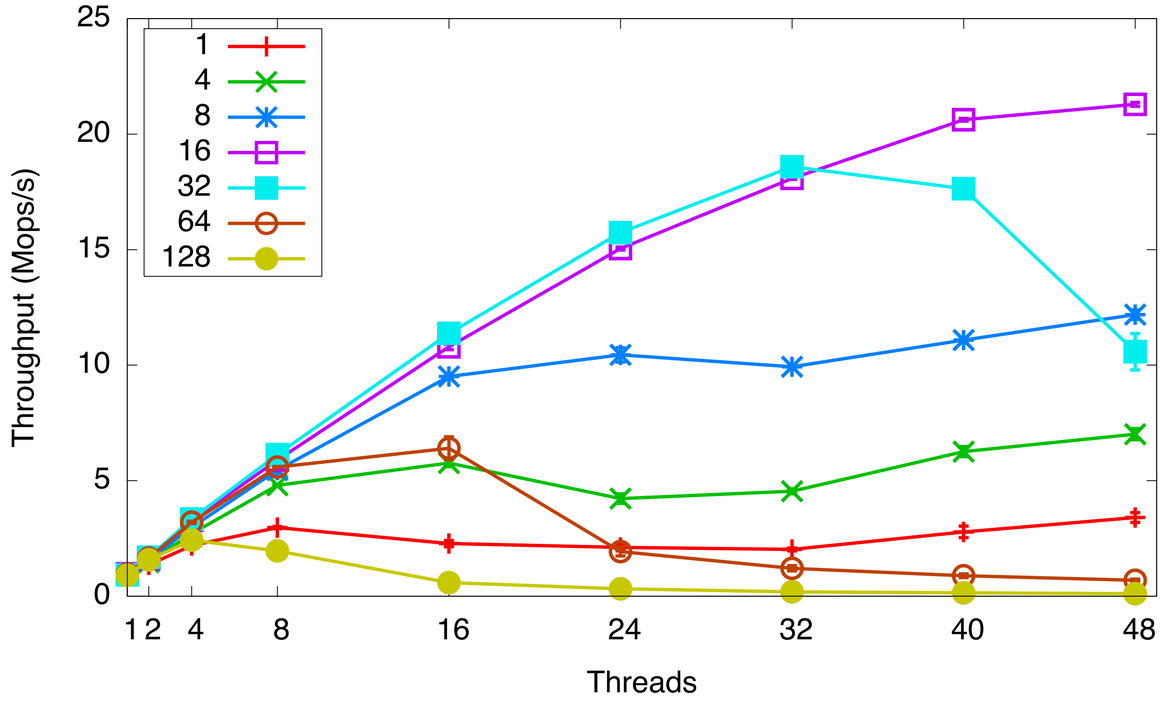


50% Updates

Boosting



5% Updates



50% Updates

What about HTM?

- Server class processors with support for TSX are available
  - “Haswell-EX” Xeon E7 v3 processors
- Can be used to run transactions with best-effort hardware assist

Intel TSX

- XBEGIN to begin a transaction
  - Specifies a fallback code path in the event of an abort, status returned in the EAX register
- XEND to end a transaction
- XABORT to explicitly abort a running transaction

EAX register bit position	Meaning
0	Set if abort caused by XABORT instruction
1	If set, the transaction may succeed on a retry
2	Set if another logical processor conflicted with a memory address
3	Set if an internal buffer overflowed
4	Set if debug breakpoint was hit
5	Set if an abort occurred during execution of a nested transaction

Support for TSX

- Modified OpenJDK to support Intel TSX instructions
- Extend sun.misc.Unsafe with native methods
  - beginHWTxn, endHWTxn, abortHWTxn
- Implemented in Hotspot C1/C2 compilers via compiler intrinsics
  - Generates inlined assembly code
- Fun and games to get compiled HTM code
  - Profiling interpreter causes HTM to abort, reverting profile stats

HTM in XJ

- Create STM and HTM versions of all methods
  - With corresponding openForRead and openForWrite methods
- Create a top level method that routes to STM/HTM version
  - First try HTM version
  - On abort, retry HTM or fall back to STM depending on return code

Preliminary HTM Results

- Still needs tuning
- Potential for great throughput
  - 3–4 × better throughput than STM when running single threaded
- Some challenges when dealing with abstract locks

Questions?