Do C and Java Programs Scale Differently on Hardware Transactional Memory?

Rei Odaira (IBM Research - Tokyo)
Jose G. Castanos
(IBM Research – T. J. Watson Research Center)
Takuya Nakaike (IBM Research - Tokyo)

Hardware Transactional Memory (HTM) Coming into the Market

IBM

Blue Gene/Q
2012

Intel

4th Generation
Core Processor
2013

IBM

zEC12
2012

IBM

POWER8

Many Programming Languages to Support TM

- Transactional langugage constructs
  - Being discussed for C++
- TM intrinsic support
  - GNU C/C++ compiler
  - IBM XL C/C++ compiler
- Software TM (STM) support
  - DSTM2 for Java
  - TinySTM for C/C++
  - Haskell, Closure, etc.

Our Goal

- Which programming language to choose?
- More specifically,
  - Focus on C and Java.
  - Do they scale differently on HTM?
  - If yes, what are the reasons?

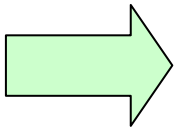
Our Methodology

- Measured C and Java STAMP benchmarks.
  - Widely used TM benchmark suite.
- Experimented on IBM zEC12's HTM.

Transactional Memory

- At programming time
  - Enclose critical sections with transaction begin and end directives.

```
lock();
a->count++;
unlock();
```

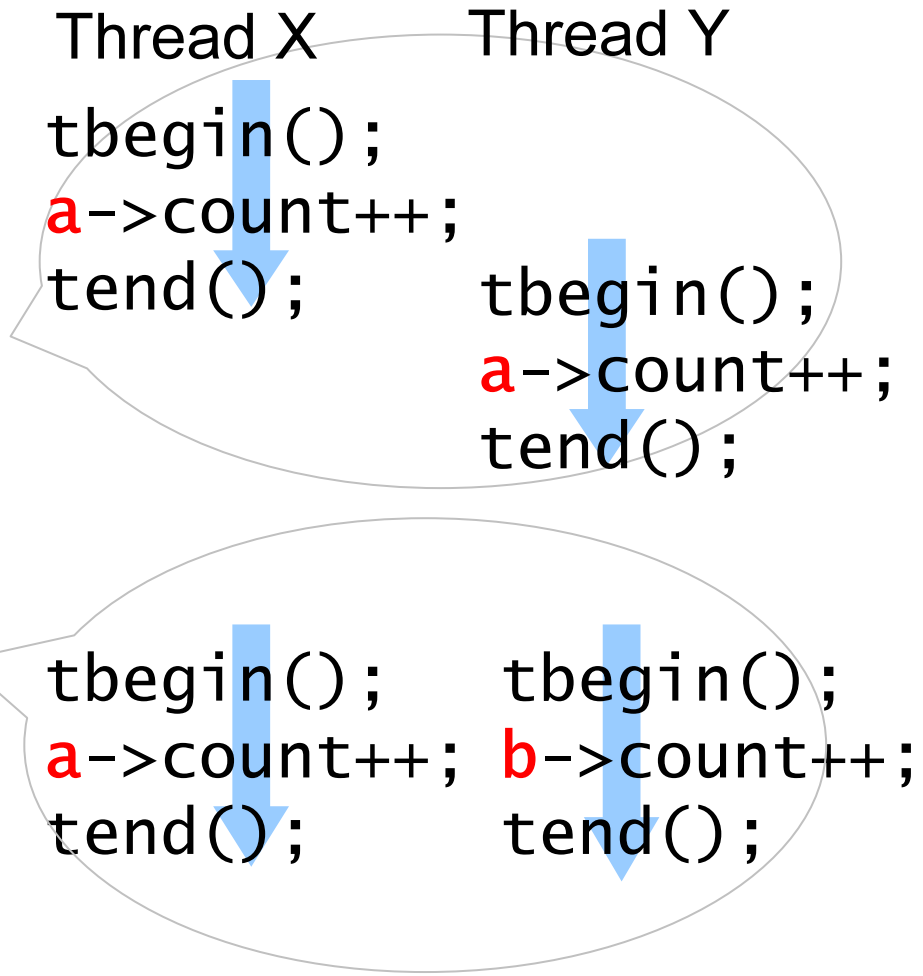


```
tbegin();
a->count++;
tend();
```

Transactional Memory

- At execution time
  - A transaction observed as one step by other threads.
  - Multiple transactions executed in parallel as long as no memory conflict.

→ Higher parallelism than locks.



HTM in IBM zEC12

```
TBEGIN
if (cc!=0)
  goto abort handler
...
...
TEND
```

- Instruction set
  - TBEGIN: Begin a transaction
  - TEND: End a transaction
  - TABORT, NTSTG, etc.
- Micro-architecture
  - Hold read set in L1 and L2 caches (~1MB)
  - Hold write set in L1 cache and store buffer (8KB)
  - Conflict detection using cache coherence protocol
  - 256-byte cache line
- Roll back to immediately after TBEGIN when:
  - Conflict, footprint overflow, etc.

STAMP Benchmarks [Minh et al., 2008]

- Most widely used benchmark suite for TM
- Written in C

bayes	Learns structure of a Bayesian network
genome	Performs gene sequencing
intruder	Detects network intrusion
kmeans-high	Implements K-means clustering
kmeans-low	
labyrinth	Routes paths in maze
ssca2	Creates efficient graph representation
vacation-high	Emulates travel reservation system
vacation-low	
yada	Refines a Delaunay mesh

Java STAMP Benchmarks

- Ported from C STAMP by UC Irvine.
  - Dialect of Java language used.

Our contributions:

- ✓ Ported to standard Java language.
- ✓ Modified to use HTM intrinsics for Java.
  - ✓ com.ibm.htm.HTM.begin(), HTM.end(), etc.
  - ✓ Intrinsics converted to HTM instructions by our just-in-time (JIT) compiler

Comparing C and Java STAMP Benchmarks

C / sequencer_run() in genome

```
TM_BEGIN();
status =
  TMTABLE_INSERT(startHashToConstructEntryTables[j],
                 (ulong_t)startHash,
                 (void*)constructEntryPtr );
TM_END();
```

Java / Sequencer.run() in genome

```
AtomicRegion.begin();
try {
  check =
  startHashToConstructEntryTables[newj]
    .table_insert(startHash, constructEntryPtr);
} finally {
  AtomicRegion.end();
}
```

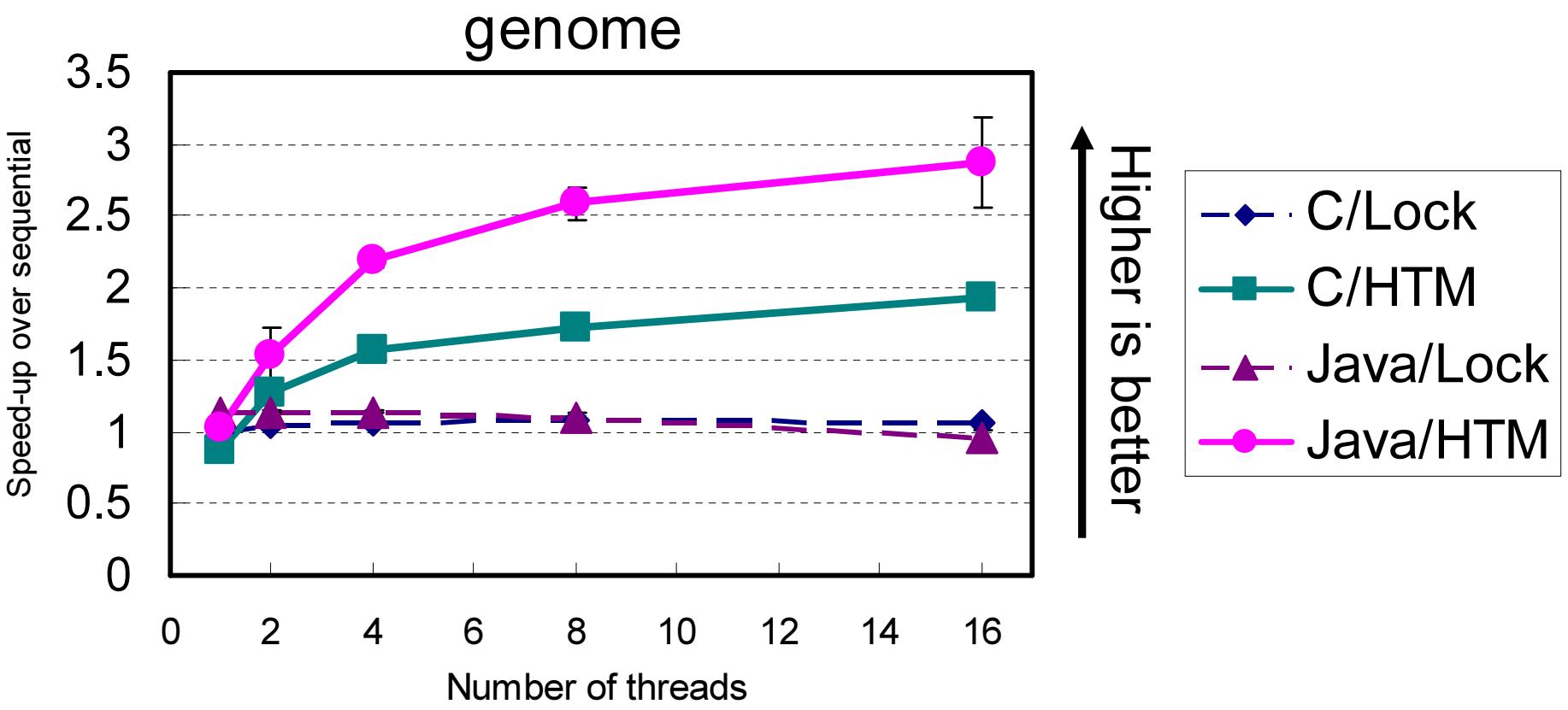
Experimental Settings and Environment

- Default runtime options used for STAMP
  - Large data set
- C
  - 64 bits, -O3, IBM XL C/C++ compiler for z/OS
- Java
  - 64-bit IBM J9/TR
  - 4-GB Java heap with mark-and-sweep GC
  - Iterated for 2 minutes and measured the second half.
- Environment
  - z/OS 1.13 with UNIX System Services
  - 16-core 5.5-GHz zEC12 with 6 GB memory

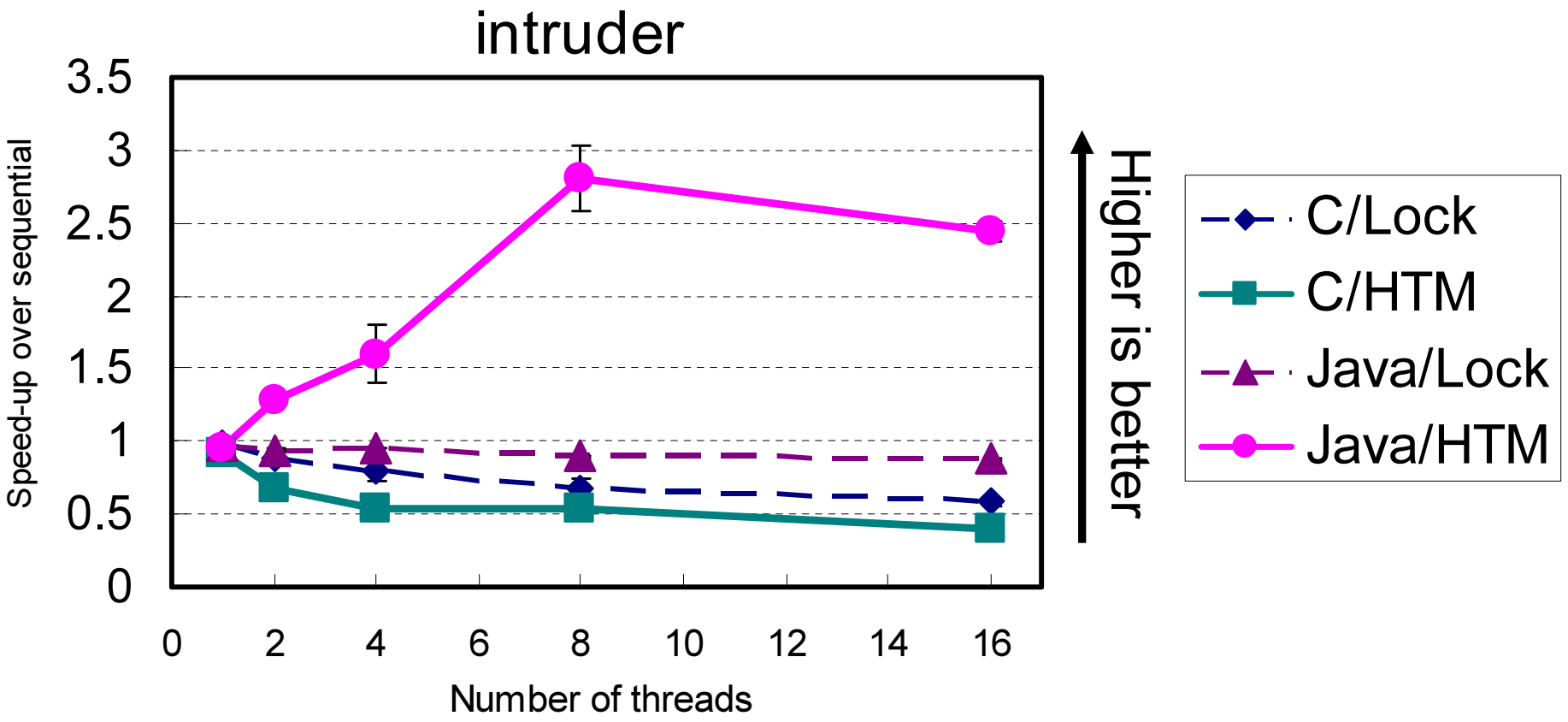
Overview of the Results

- Among 10 benchmarks in STAMP,
- Java scaled better than C in 4 benchmarks.
- C scaled better than Java in 2 benchmarks.
- C and Java scaled similarly in 4 benchmarks.
  - Both scaled well in 1 benchmark.
  - Both did not scale in 3 benchmarks.

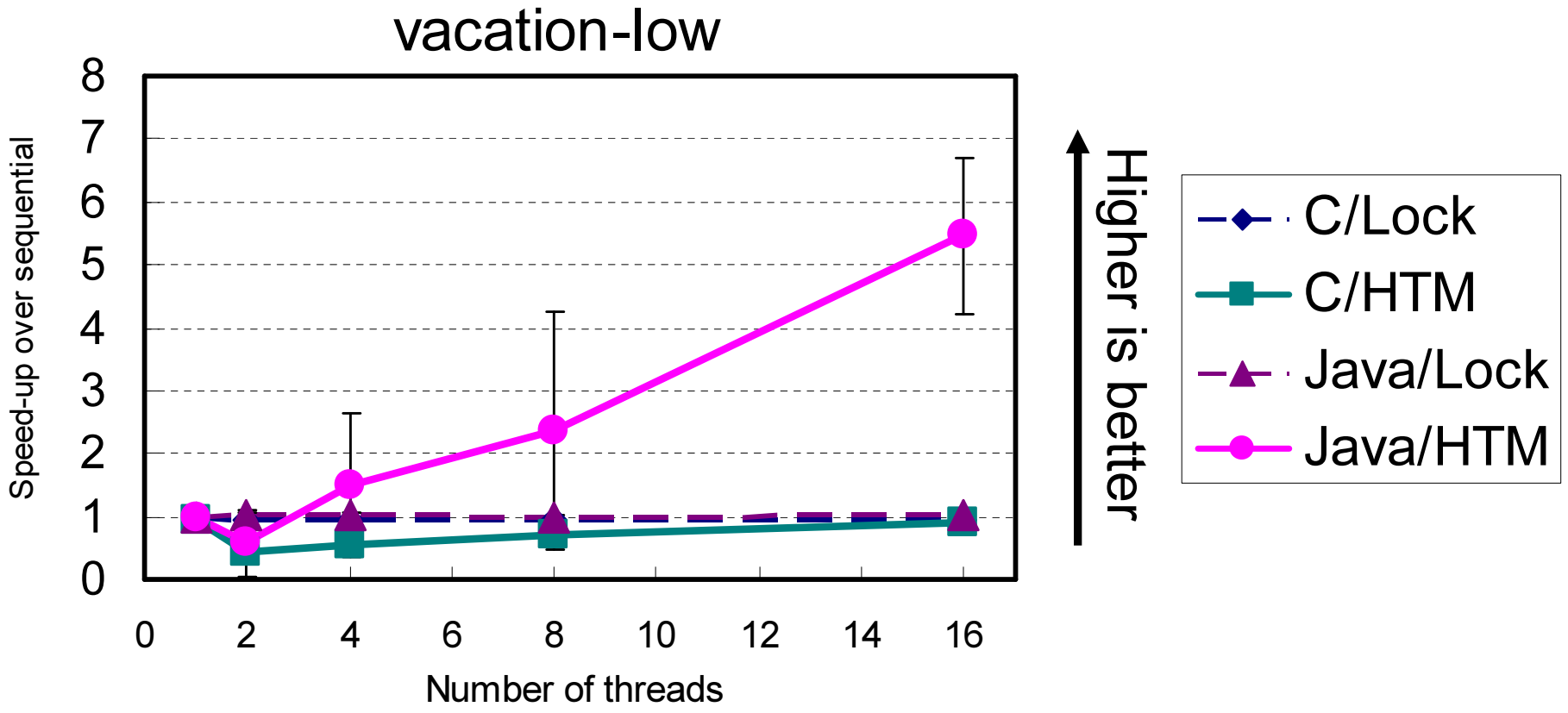
Java Scaled Better than C in 4/10 Benchmarks



Java Scaled Better than C in 4/10 Benchmarks

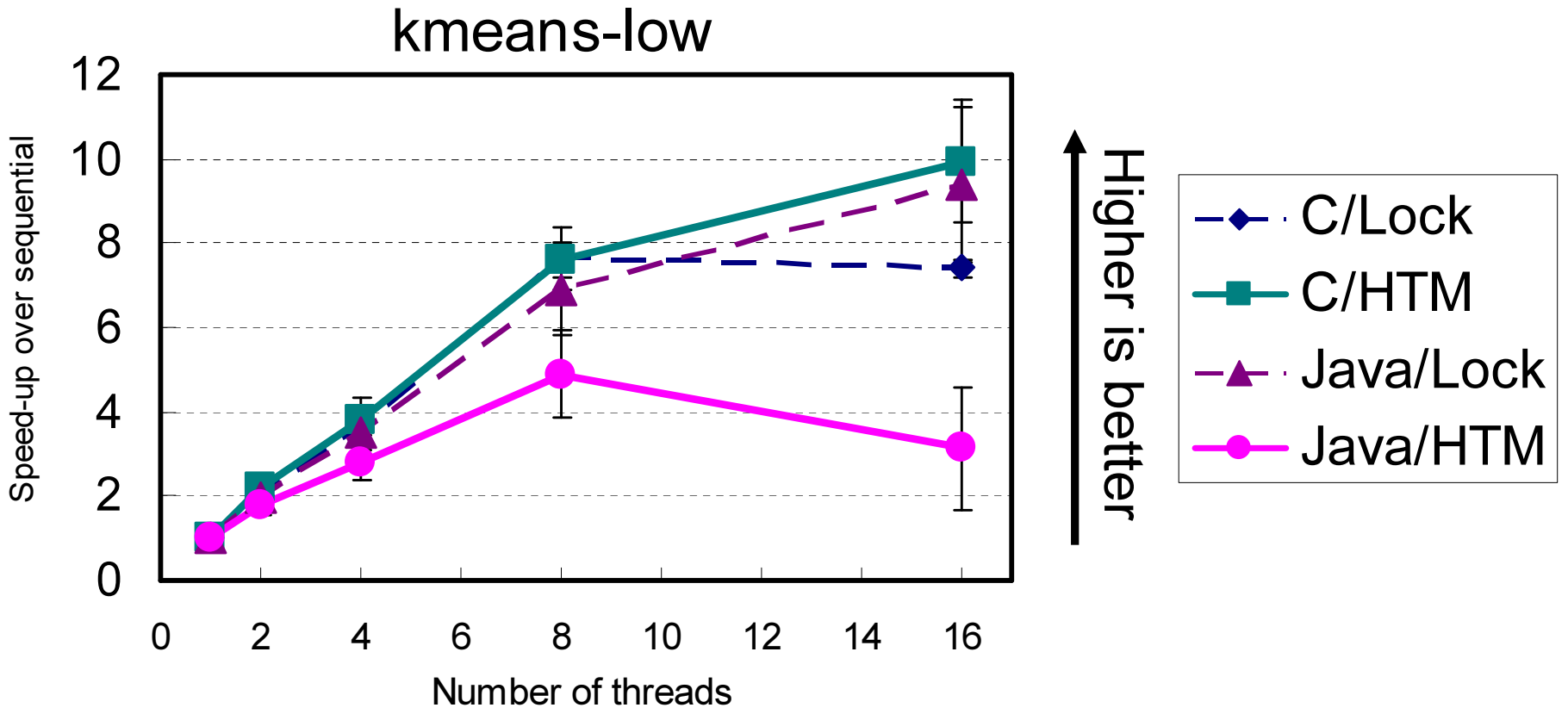


Java Scaled Better than C in 4/10 Benchmarks



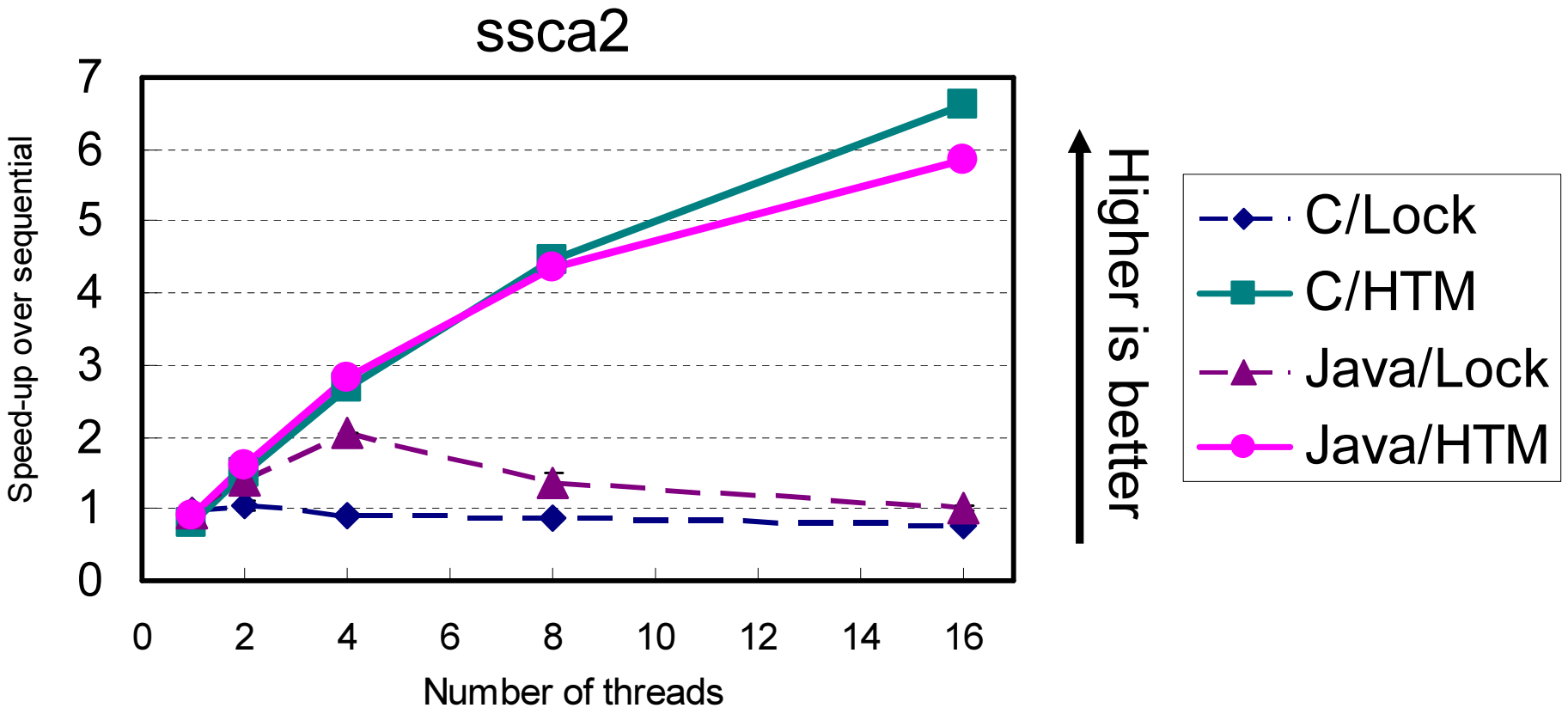
Similar in vacation-high

C Scaled Better than Java in 2/10 Benchmarks

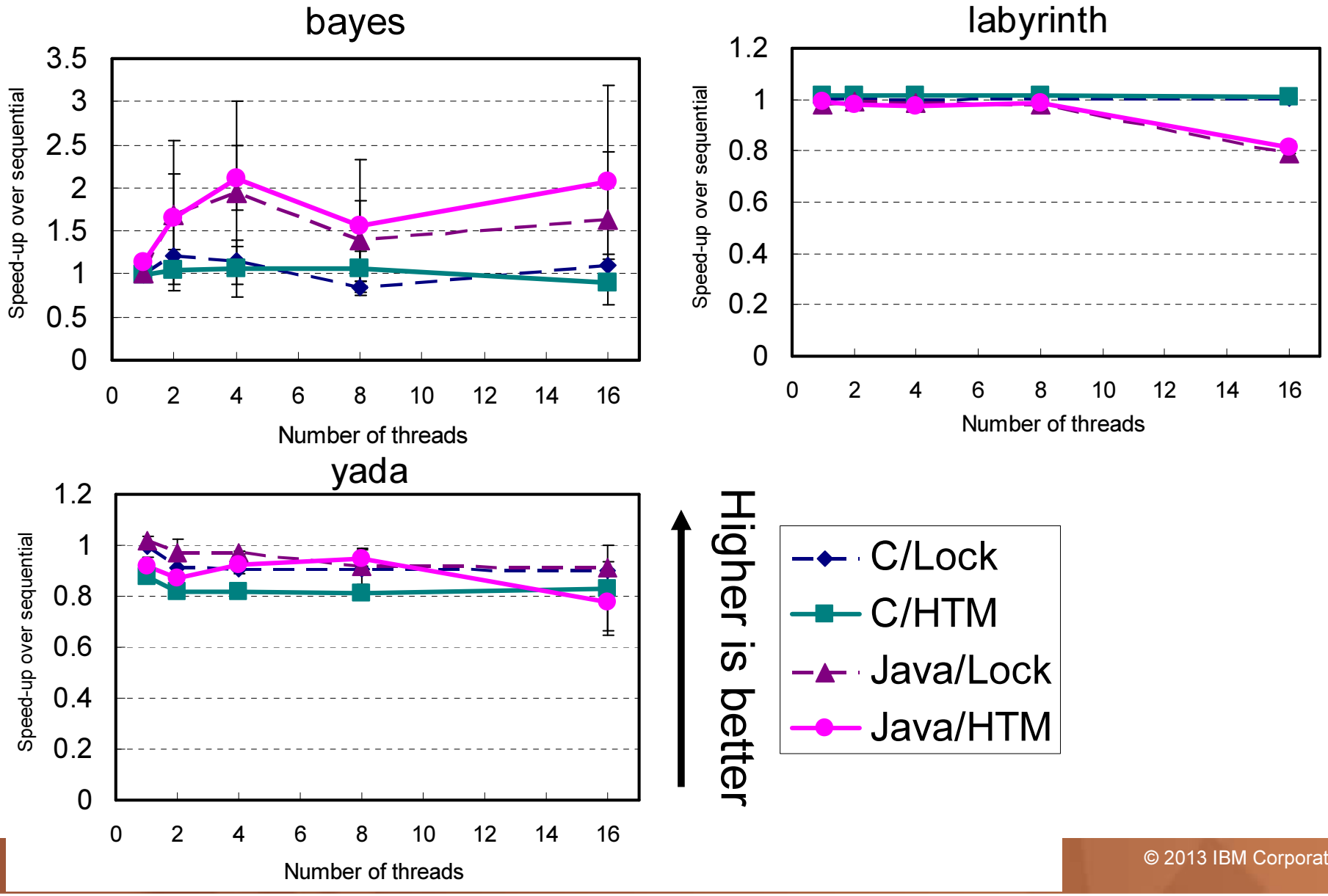


Similar in kmeans-high

C and Java Scaled Similarly in 4/10 Benchmarks



C and Java Scaled Similarly in 4/10 Benchmarks



Why Did C Sometimes Scale Worse than Java?

- Because of the conflicts at malloc().
- Java
  - Objects allocated from thread-local heaps.
- C
  - Global data manipulated in z/OS malloc().

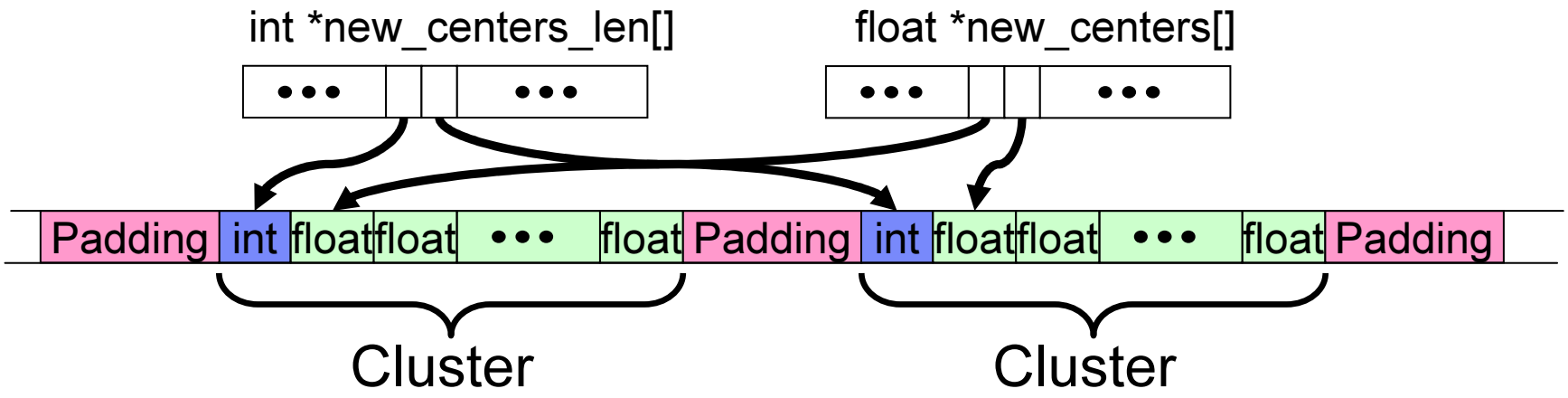
  → Used a simple thread-local allocator attached with STAMP. (free() not supported).

  → Should use efficient malloc(), like TCMalloc.

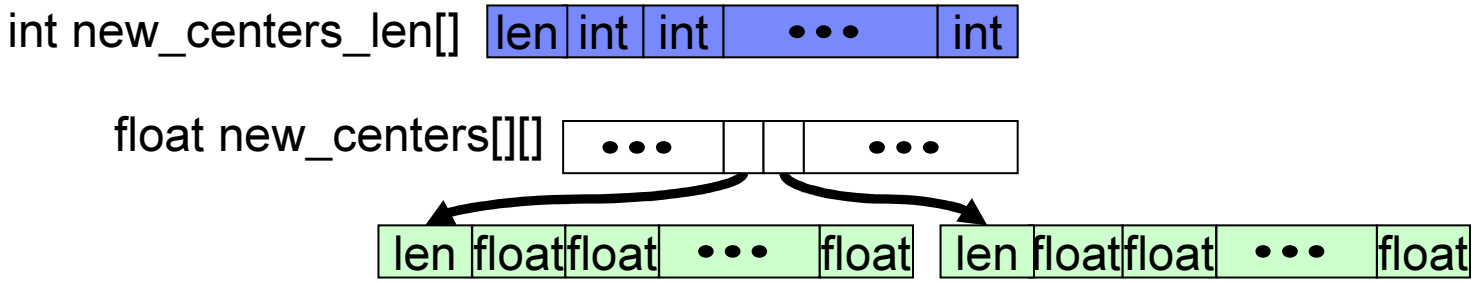
Why Did Java Sometimes Scale Worse than C?

- Because of the lack of padding.

C version of kmeans



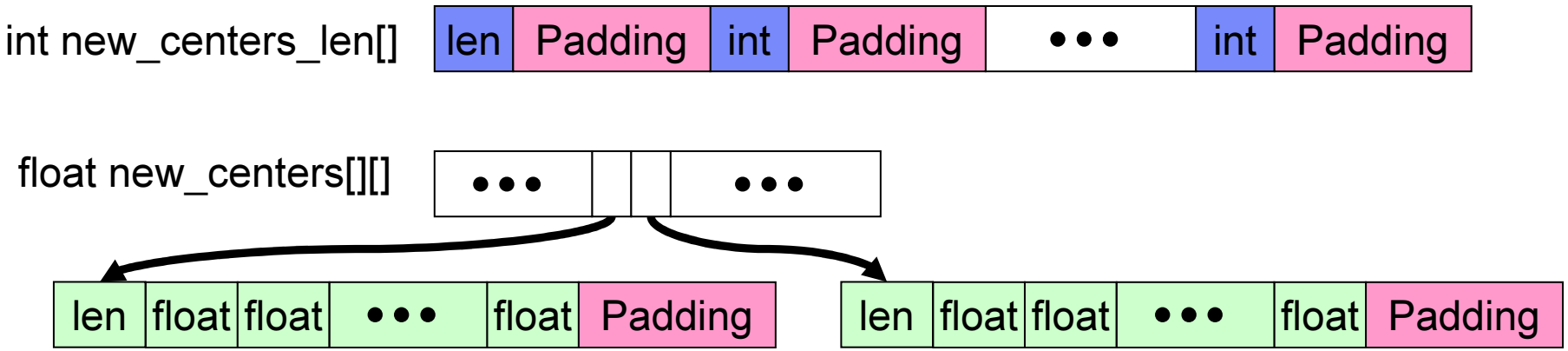
Java version of kmeans



Padding by Rewriting Java Source Code

- Not memory efficient
- CPU and JVM implementation dependent

Modified Java version of kmeans



Better Solutions

- Structs in Java
  - Being proposed.
- Automatic mechanism in Java VM
  - Collocate objects accessed in the same transaction.
  - Separate objects updated in different transactions.

Why Did Java Sometimes Scale Worse than C?

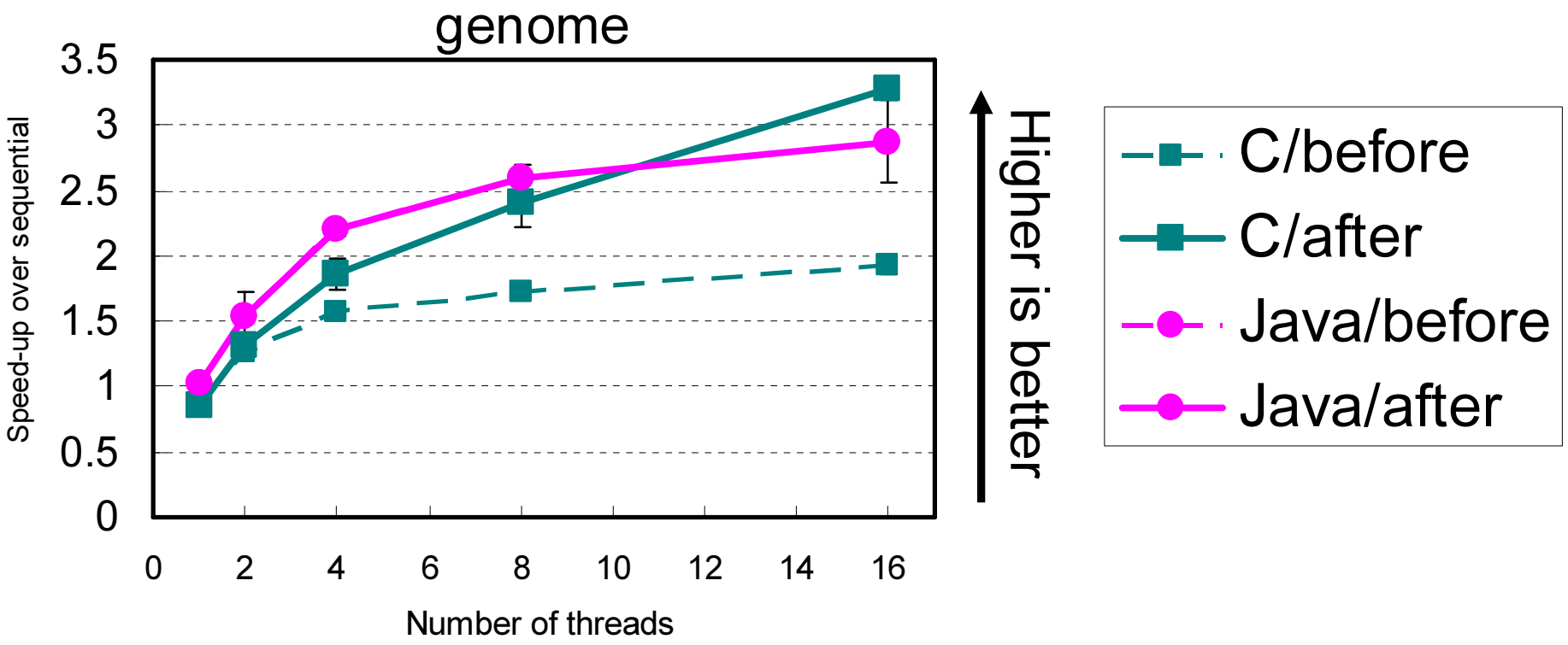
- Java VM service invoked during transactions.
- In the vacation benchmark, profiling code executed in transactions.
  - Profiling needed for JIT-compiler optimizations.
  - Global data updated by the profiling code.

→ Profiling disabled for certain methods.

- Abort-prone JVM services:
  - JIT compiler, code patching, etc.

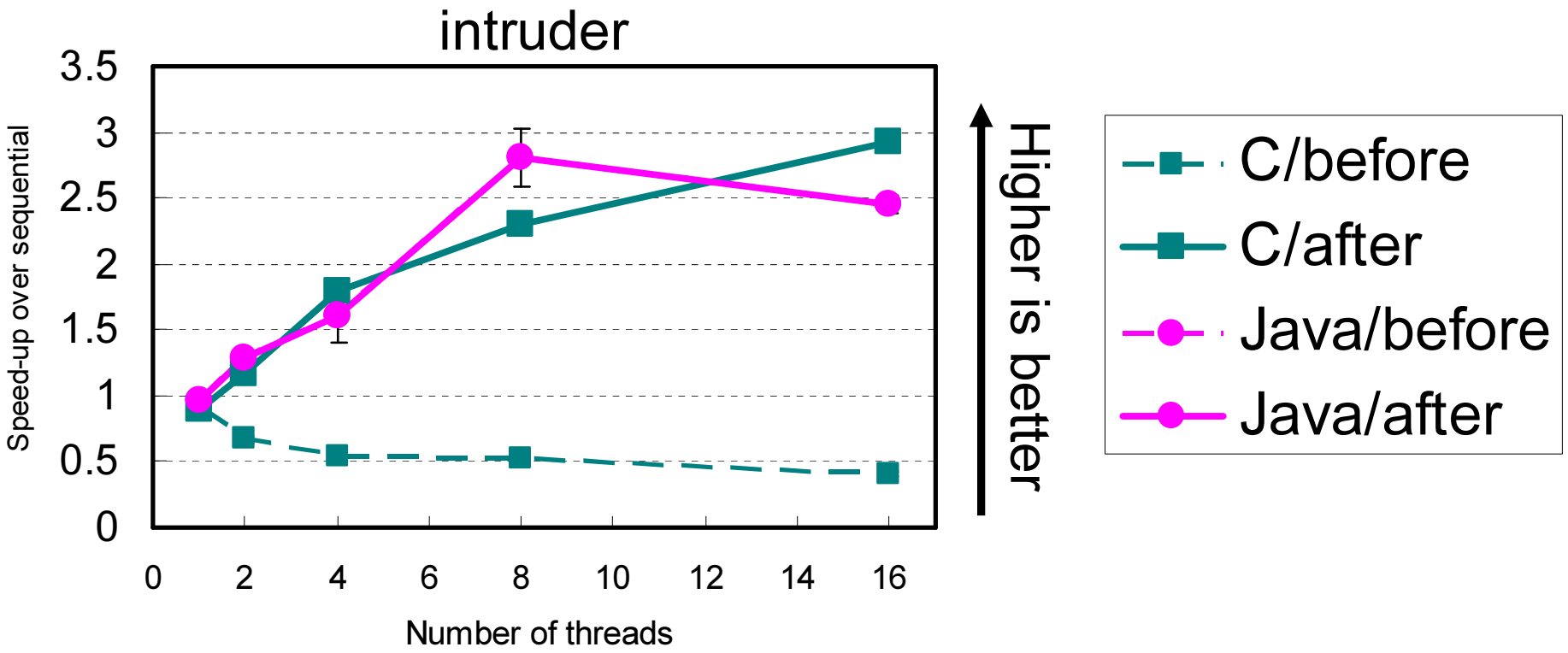
→ HTM-aware JVM services needed.

C vs. Java after Modifications



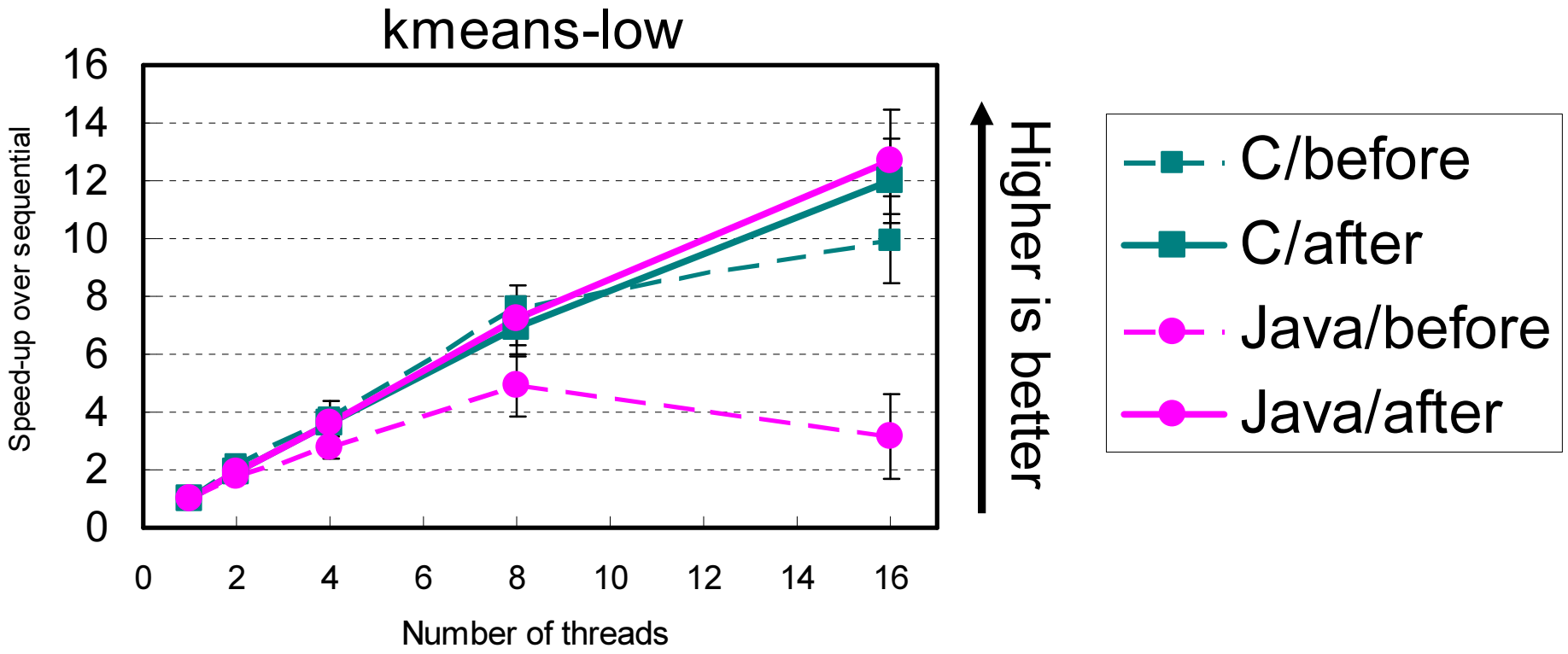
C: thread-local malloc()

C vs. Java after Modifications



C: thread-local malloc()

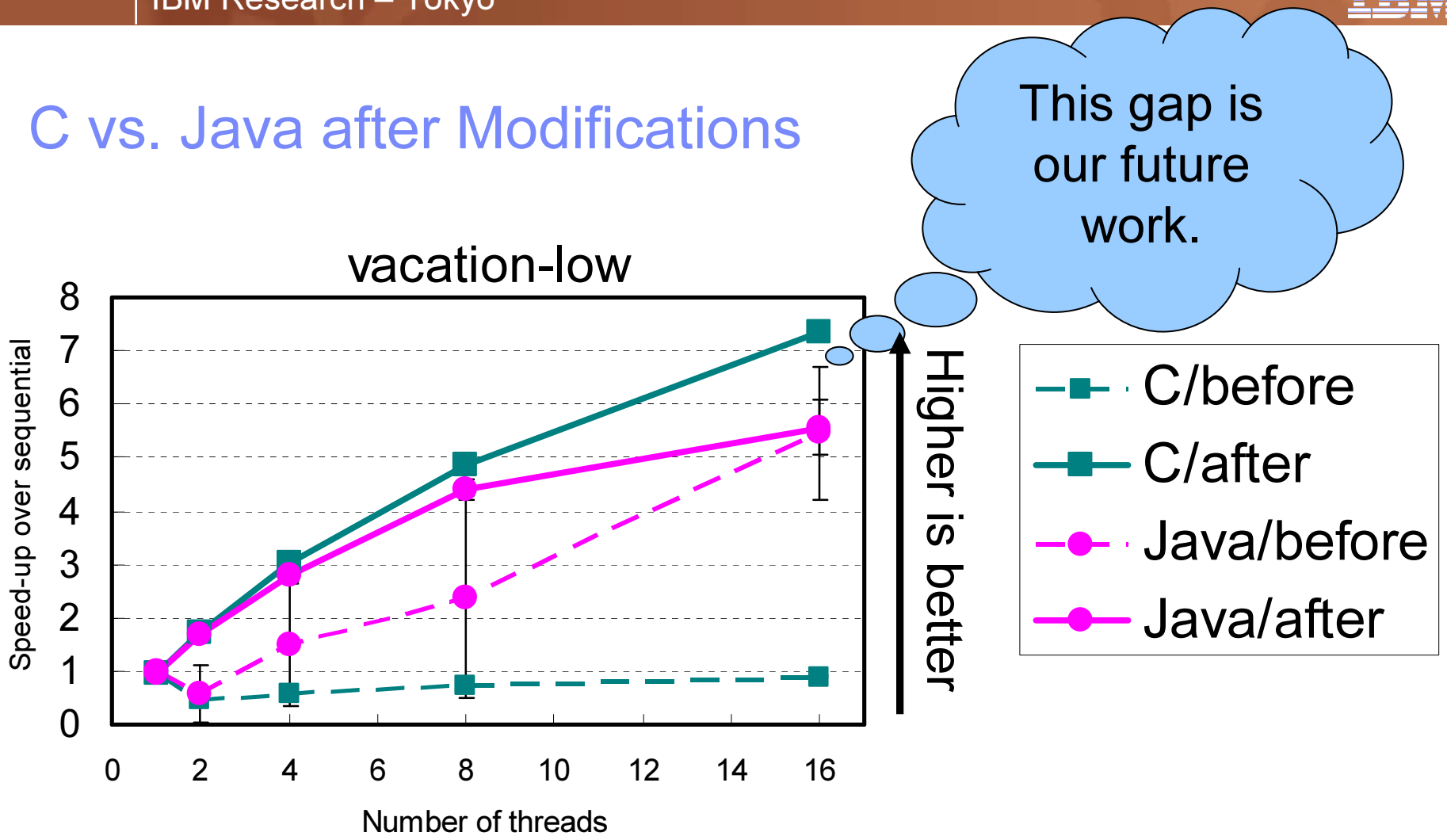
C vs. Java after Modifications



C: thread-local malloc()

Java: padding

C vs. Java after Modifications



C: thread-local malloc()
Java: profiling disabled

Conclusion

- Do C and Java scale differently on HTM?

→ Yes

- But after appropriate optimizations …

→ Scale similarly

- HTM-aware system software really needed.
  - Thread-local malloc()/free()
  - Automatic collocation and padding in Java VM
  - Transaction-aware Java VM services

backup

Absolute Performance