Expanding the High Performance Embedded Computing Tool Chest - Mixing Java and C

Nazario Irizarry, Jr.

MITRE Corporation
202 Burlington Rd.
Bedford, MA, 01730

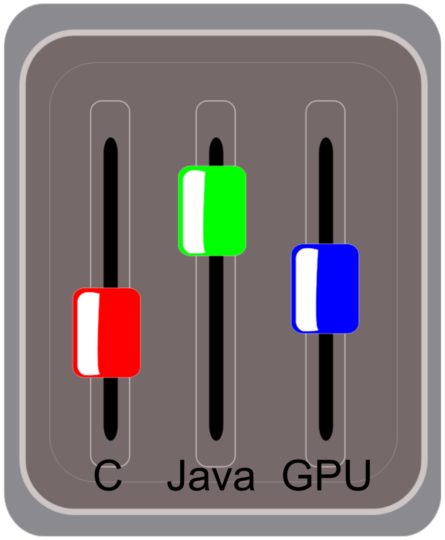
Presented at IEEE High Performance Extreme Computing Conference (HPEC `13), Waltham, MA

Outline

- **Goals & motivation**
- **Performance findings**
  - CPU experiments
  - GPU experiment
- **Helpful frameworks**

Motivation & Goals

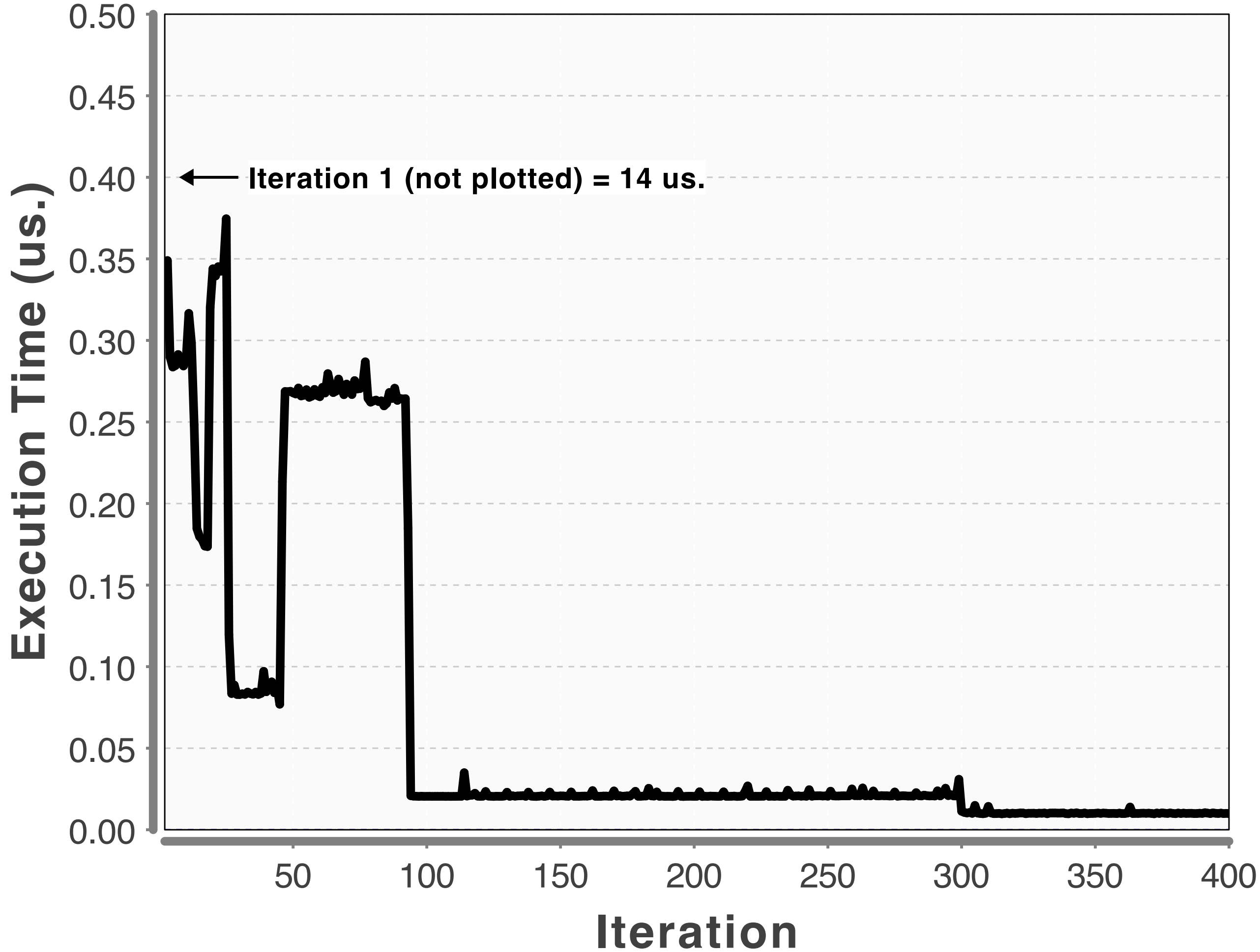
- **C is fast but development is meticulous and time consuming**
- **Java is not as fast but development is more expeditious**
- **Java frameworks are very capable**
  - Logging, dynamic scheduling, CPU load balancing, elastic grid resizing, automatic data serialization, automatic work failover, etc.
- **Want to build hybrid apps that leverage the strengths of both**
  - Need to understand when Java 7 will and won’t perform
  - Need to understand the cost of interactions between Java and C
  - Need to understand how effectively Java can utilize GPUs



Isn’t Java Slow?

- **Java startup is slow**
  - Disk I/O
  - Class lookup & validation
  - Static initialization
- **Java Hotspot optimizes during runtime**
  - Dynamically switch from interpreted to compiled
  - Aggressive method in-lining

Effect of Java Warming on Execution Time

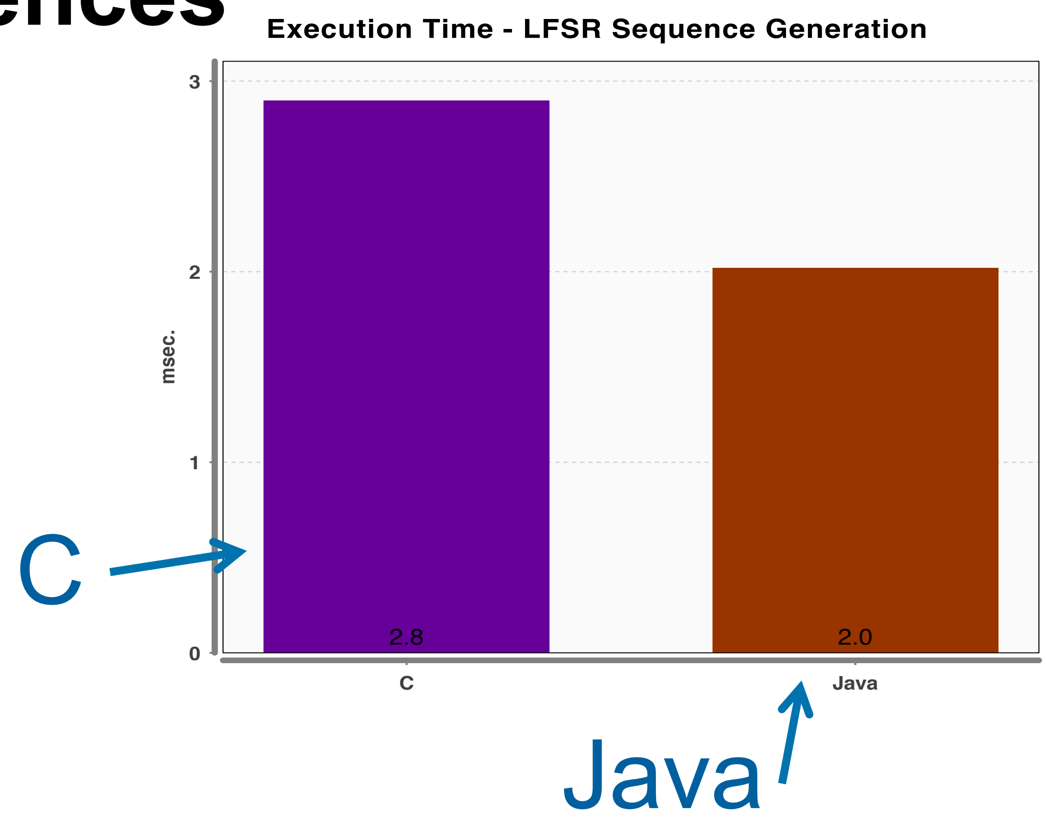


Test Environment

- **Hardware**
  - Intel Core 2 Quad Q9650 @ 2.99GHz
  - 4 GB RAM
  - Four core, not hyper-threaded, SSE4.1
- **Linux 2.6.32**
- **GCC 4.4.6**
  - -O3 optimization
- **Oracle Java 7**
  - Standard edition, 1.7.0_02

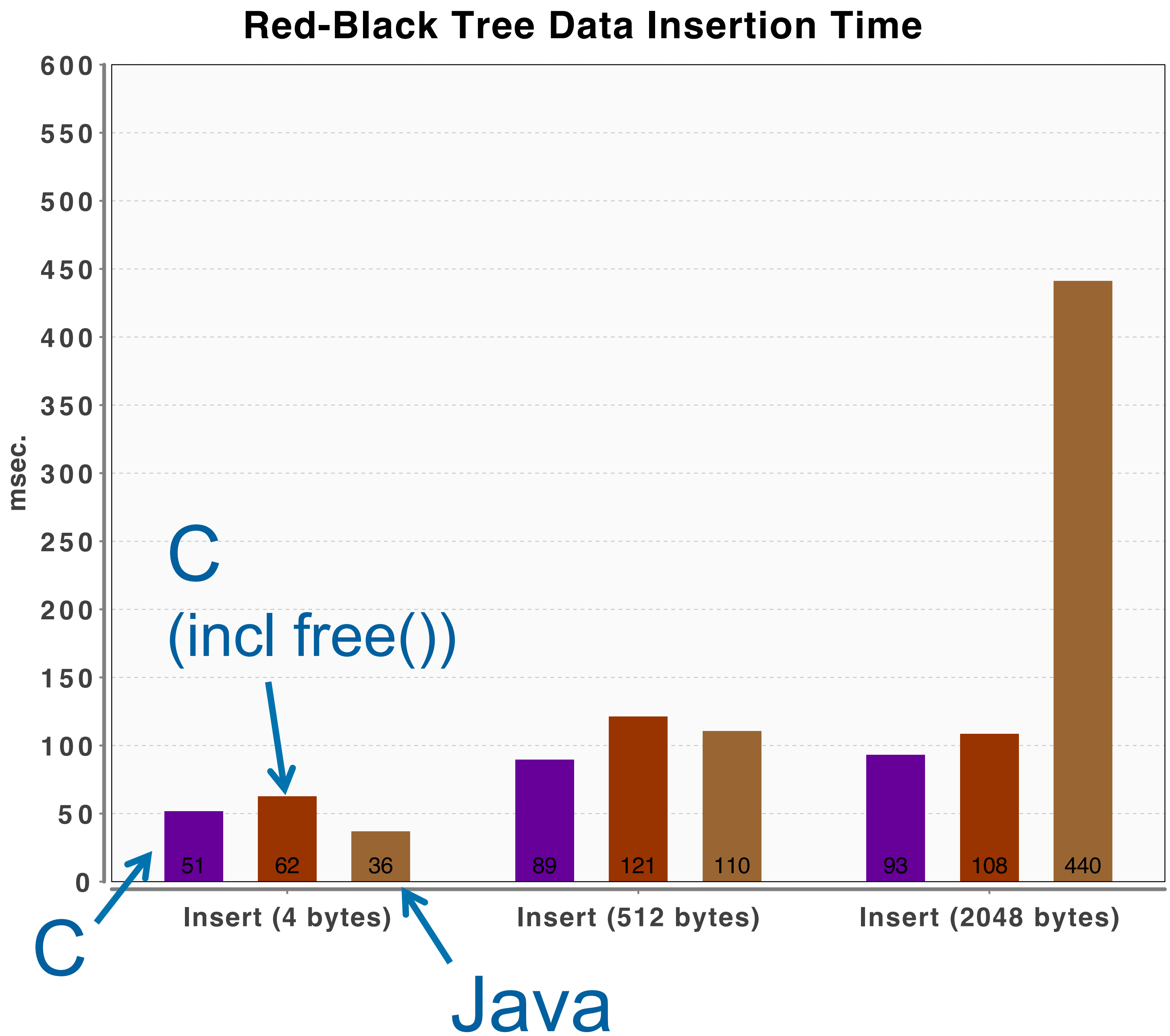
Test: Bit-Twiddling

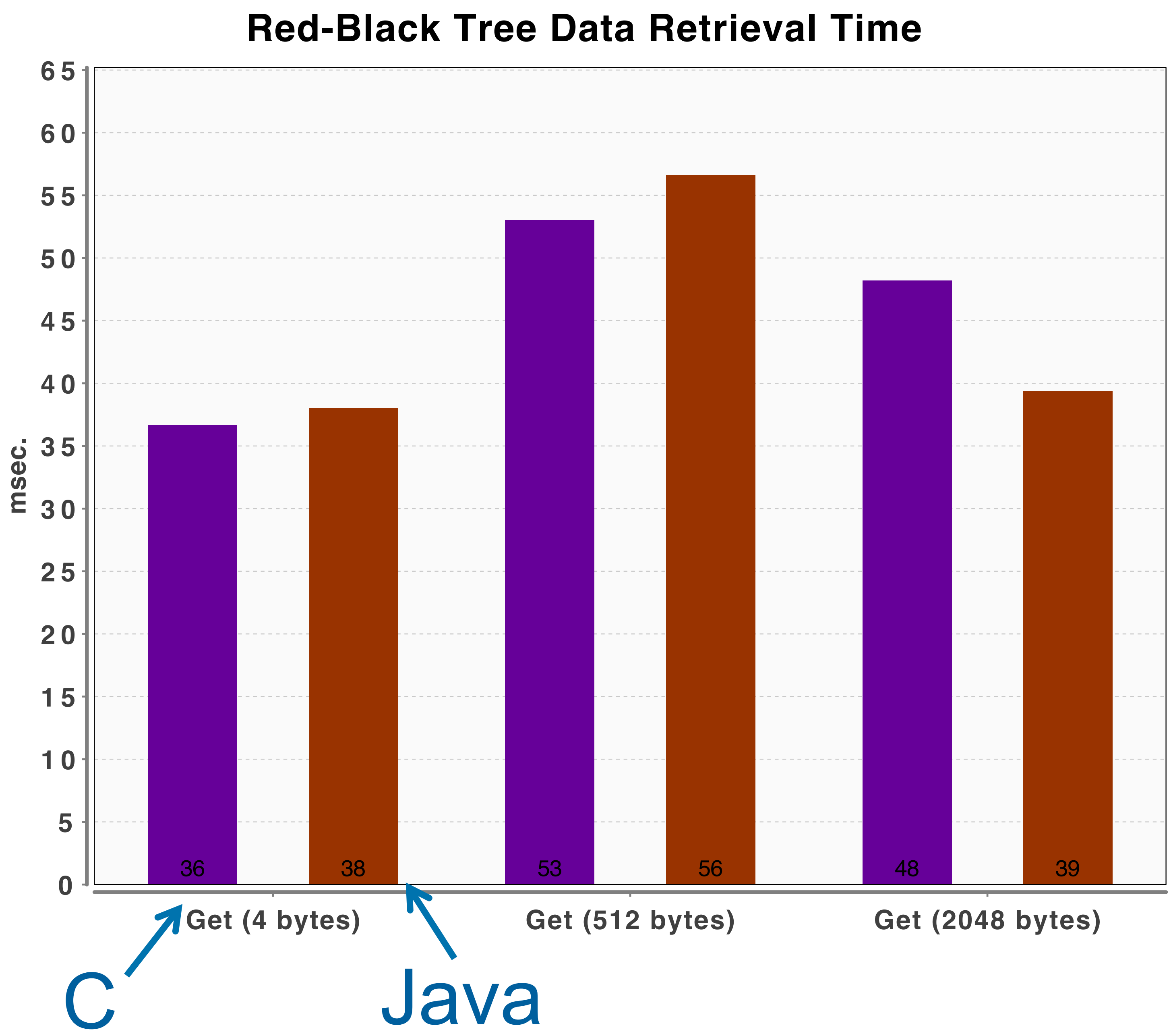
- **Workload consists of generating multiple linear feedback shift register sequences**



Test: Structure Building

- **Red-black tree key-value insertion and retrieval**
- **Exercises**
  - Memory allocation
  - Integer key comparison
  - Conditional code execution
  - Reference manipulation

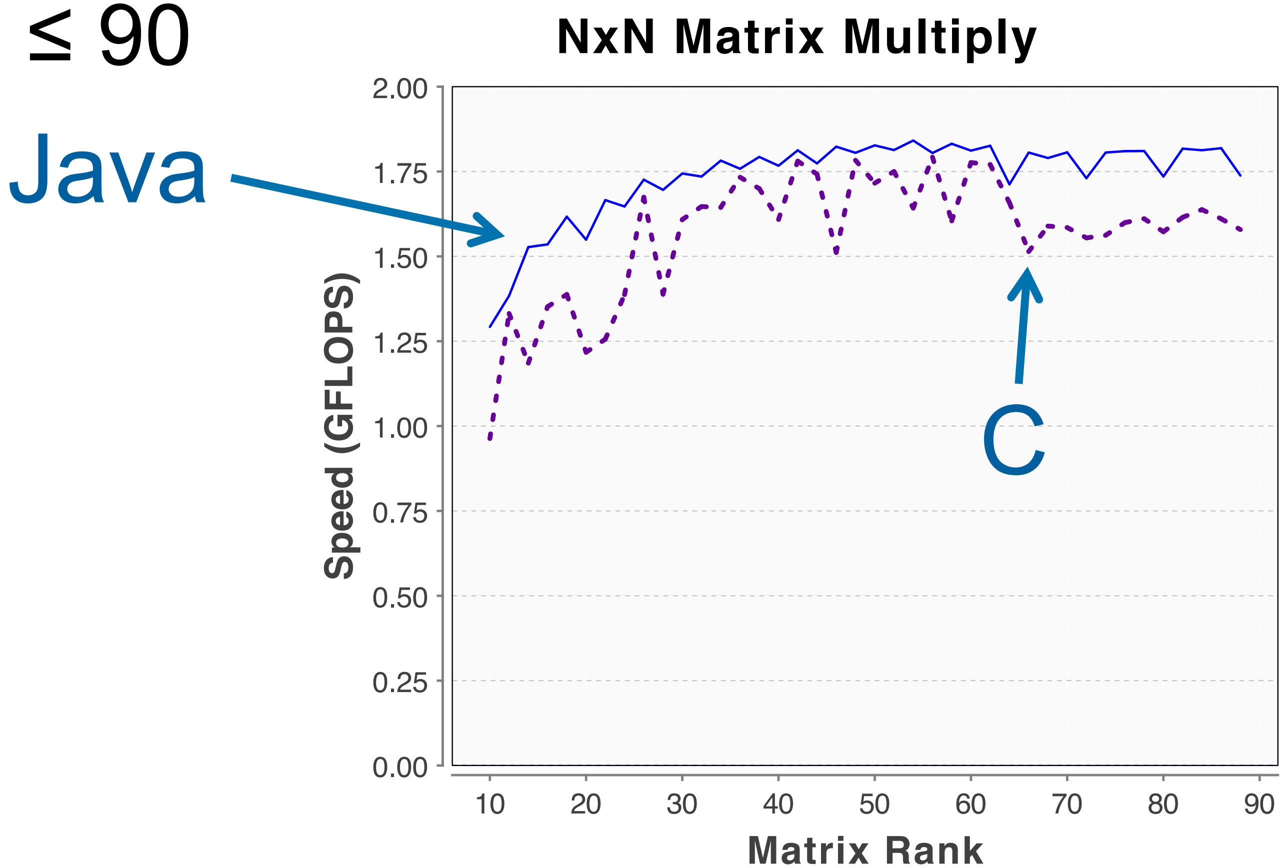

Red-Black Tree Data Insertion Time
msec.
600
550
500
450
400
350
300
250
200
150
100
50
0
51
62
36
89
121
110
93
108
440
Insert (4 bytes)
Insert (512 bytes)
Insert (2048 bytes)
C
(incl free())
C
Java



Red-Black Tree Data Retrieval Time
msec.
65
60
55
50
45
40
35
30
25
20
15
10
5
0
36
38
53
56
48
39
Get (4 bytes)
Get (512 bytes)
Get (2048 bytes)
C
Java

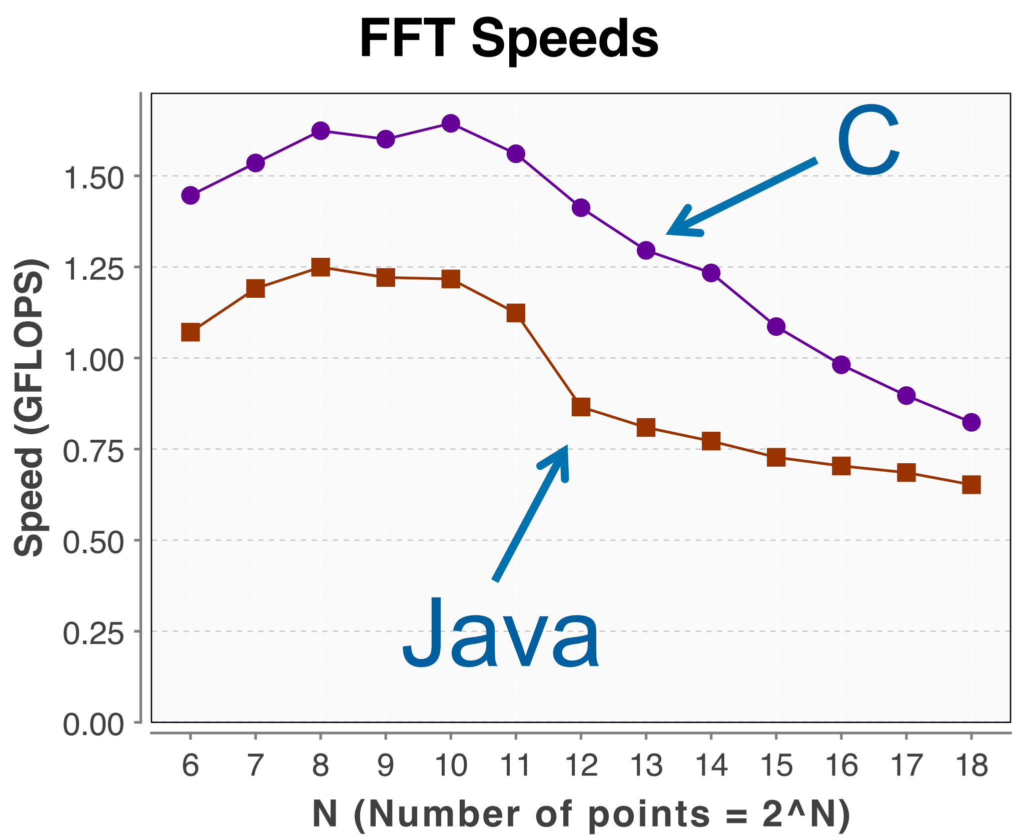
Test: Simple Iteration and Math

- **Matrix multiply**
  - 10 ≤ N ≤ 90



Test: Intricate Iteration and Math

- **Fast Fourier Transform**
  - Non-optimized radix 2 butterflies



Test Design: Impact of Garbage Collector

Test 1

- **0 GC actions/sec**
  - Bit twiddling test
  - Preallocated buffers

Test 2

- **7.5 GC actions/sec.**
  - Red black tree test
  - Lots of memory allocation

Intentionally poorly tuned

- **2 scenarios for each**
  – 2 compute threads on 2 cores
  – 2 compute thread on 3 cores

Test Results: Impact of Garbage Collector

Test 1

- **0 GC actions/sec**

No Impact

Test 2

- **7.5 GC actions/sec.**

15 % Execution Time Impact

- **Key design practices**
  - Preallocate buffers and structures
  - Be smart about String concatenation
  - Keep the GC quiet

Investigation: Accessing Native Memory

- **Java data is “fenced”**
  - Passing reference types (arrays and objects) to C incurs overhead
- **Java Native Interface (JNI) is fastest but most tedious**
- **Alternatives: Java Native Access, BridJ, SWIG, HawtJNI**

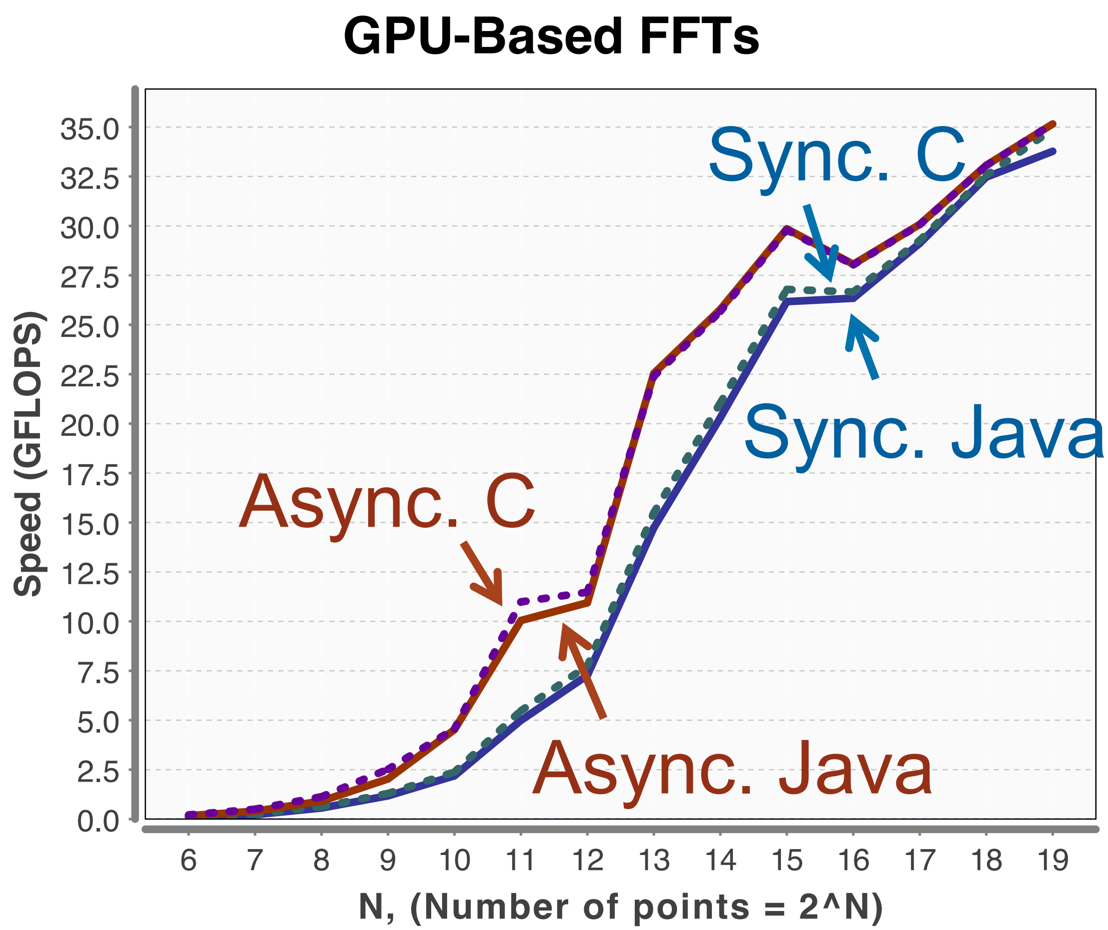
Comparing 3 Bridging Alternatives

Call Arguments	JNI	SWIG	BridJ
4 int args	11 ns.	17 ns.	200 ns.
2 read-only Java arrays (double, length 200)	13 ms.	190 ms.	110 ms.
Pointers to 2 native arrays reference by Java proxies	N/A	58 ns.	10 ns.

- **BridJ and SWIG provide good performance**
- **BridJ is easier to use**
  - Uses .h files directly
  - Includes ability to manage C memory from Java
- **See paper and final MITRE report for more details and test cases**

Test: Utilizing GPUs (1 of 2)

- **FFT's of varying sizes invoked with both synchronous and asynchronous kernel invocation**
- **Apple's optimized FFT's for OpenCL was ported to Java**
  - BridJ for native data management
  - JavaCL for OpenCL access
  - Both are subprojects of "nativelibs4java" at Sourceforge



- **Java kept the GPU's as busy as C did**
  - Kernel invocation from Java was 2 ms longer than from C (9 ms)

Summary

- **Java 7 (w/ Hotspot) can perform well after warming**
  - Code loops with regular indexes do well
  - Structure allocation and manipulation of moderate sized structures
  - Boolean integer operations
  - Conditional logic
- **Well designed code can incur negligible GC impact**
- **Hybrid performance will depend on:**
  - Granularity of interactions between the Java and C code
  - Whether data is created natively or in Java
- **Frameworks exist to help glue the hybrid Java/C app**
- **GPUs can be effectively managed and utilized if the data is native**
- **A detailed final research report will be available**
  - Parallelism; CPython, Jython, PyPy, Scala; Java grid-computing frameworks

Frameworks and Links

- **BridJ –Java to C binding**
  – http://code.google.com/p/bridj/
- **JavaCL – Java tier over OpenCL**
  – http://code.google.com/p/javacl/
- **SWIG – Multiple languages to C binding**
  – http://www.swig.org/
- **CShimToJava – C to Java binding**
  – http://cshimtojava.sourceforge.net
- **Benchmarks used in this study**
  – http://jcompbmarks.sourceforge.net