Java in space?

Filip Pizlo, Luke Ziarek, Ethan Blanton, Jan Vitek

Fiji Systems LLC
Java is a *modern, portable, simple, safe, and widely-used* language.
But, so far, Java has seen limited use in safety-critical applications.
Fiji Systems LLC is trying to make Java flightworthy.
Why even consider Java?
Why even consider Java?

- Top defects in existing embedded code: *
  - Buffer overflow and underflow
  - Null object dereference
  - Uninitialized variable
  - Inappropriate cast
  - Division by zero
  - Memory leaks

Source: “Diagnosing Medical Device Software Defects”
Medical DeviceLink, May 2009
Why even consider Java?

- Top defects in existing embedded code: *
  - Buffer overflow and underflow: impossible in Java
  - Null object dereference: checked in Java
  - Uninitialized variable: impossible in Java
  - Inappropriate cast: checked in Java
  - Division by zero: checked in Java
  - Memory leaks: GC

Source: “Diagnosing Medical Device Software Defects”
Medical DeviceLink, May 2009
Java’s safety also gives you fine-grained space partitioning and isolation for free.
Java’s safety also gives you fine-grained space partitioning and isolation for free.

Standard language taught in universities around the world.
- Java’s safety also gives you fine-grained **space partitioning** and isolation for free.

- **Standard language** taught in universities around the world.

- Java is known to scale to **800+ cores**; rich library support for parallel/concurrent programming.
- Java’s safety also gives you fine-grained space partitioning and isolation for free.

- **Standard language** taught in universities around the world.

- Java is known to scale to 800+ cores; rich library support for parallel/concurrent programming.

- Concurrency in Java is fast. Java locks are at least 50% faster than POSIX locks on Linux.
- Java’s safety also gives you fine-grained space partitioning and isolation for free.
- **Standard language** taught in universities around the world.
- Java is known to scale to 800+ cores; rich library support for parallel/concurrent programming.
- Concurrency in Java is fast. Java locks are at least 50% faster than POSIX locks on Linux.
- Rich tool support (IDEs, bug finders, verification tools, modeling tools, etc.)
- Java’s safety also gives you fine-grained **space partitioning** and isolation for free.

- **Standard language** taught in universities around the world.

- Java is known to scale to **800+ cores**; rich library support for parallel/concurrent programming.

- Concurrency in Java is **fast**. Java locks are at least **50% faster** than POSIX locks on Linux.

- Rich **tool support** (IDEs, bug finders, verification tools, modeling tools, etc.)

- Java supports safe **dynamic loading**.
Summary of Java’s benefits
Summary of Java’s benefits

- A subset of program errors go away.
Summary of Java’s benefits

- A subset of program errors go away.
- For the remaining program errors:
Summary of Java’s benefits

* A subset of program errors go away.

* For the remaining program errors:
  * Java programs are more **analyzable**, making model-checking easier.
Summary of Java’s benefits

- A subset of program errors go away.
- For the remaining program errors:
  - Java programs are more analyzable, making model-checking easier.
  - Java programs are more testable, thanks to extensive tool support for unit testing.
Summary of Java’s benefits

- A subset of program errors go away.

- For the remaining program errors:
  - Java programs are more analyzable, making model-checking easier.
  - Java programs are more testable, thanks to extensive tool support for unit testing.
  - Java programs are more readable, making code audits simpler.
Summary of Java’s benefits

- A subset of program errors go away.
- For the remaining program errors:
  - Java programs are more **analyzable**, making model-checking easier.
  - Java programs are more **testable**, thanks to extensive tool support for unit testing.
  - Java programs are more **readable**, making code audits simpler.
- Java brings a plethora of modern language features that make software engineering more cost-effective.
Java for Real Time

- 1998 - NIST prepared a set of requirements for RT Java
- 1999 - JSR-001, first draft specification
- 2002 - Real Time Specification for Java (RTSJ) accepted
- 2005 - SUN and IBM release RTSJ implementations
- 2006 - JSR-302 Safety Critical Java started
- 2008 - development of Fiji VM SCJ begins.
Fiji VM
High-performance Real-time Java
Fiji VM

- Compiles Java to fast machine code
  - no interpretation! competitive with C code!
- Small footprint (~100KB)
- Strong real-time support
  - Real Time Garbage Collection and scoped memory
- Can write device drivers in Java
- Multi-core ready
- Portable (x86, PPC, SPARC, ARM, others)
- Runs on RTEMS, Linux, others
Fiji VM is a compiler

- Fiji VM uses Java byte code as input (versions 1.0 through 7 supported)
- Fiji VM produces executables.
- Your code is never interpreted.
- Result: small footprint, fast and predictable execution.
Fiji VM is a compiler

- Fiji VM uses Java byte code as input (versions 1.0 through 7 supported)
- Fiji VM produces executables.
- Your code is never interpreted.
- Result: small footprint, fast and predictable execution.
Fiji VM is a compiler

- Fiji VM uses Java byte code as input (*versions 1.0 through 7 supported*)
- Fiji VM produces executables.
- Your code is never interpreted.
- Result: small footprint, fast and predictable execution.
Fiji VM real-time support

- Striving for Safety-critical Java (SCJ) compliance, targeting DO-178B and DO-178C level A certification.
- Every operation has predictable timing.
- Memory usage is predictable.
  - both garbage collection and SCJ scoped memory
Fiji VM Garbage Collection

- Garbage collection allows safe, *fragmentation-free* dynamic memory allocation.
- Garbage collector runs as a separate thread.
- The scheduling of this thread can be *programatically* controlled.
- We provide WCET and schedulability formulas that describe how much CPU time this thread should get.
- Application threads see no interference from the GC.
This isn’t the Java you’re used to.
This isn’t the Java you’re used to.

Fiji VM is 83% faster.
Predictability Benchmark

- We use the CDx benchmark to measure performance and predictability of Fiji VM.
  - see http://adam.lille.inria.fr/soleil/rcd/
- CDx has a single periodic task that performs aircraft collision detections.
- CDx includes both C and Java versions.
Predictability

*CDx benchmark “HARD” configuration.
Predictability

Standard JVM

>100ms pauses!

*CDx benchmark “HARD” configuration.
*CDx benchmark “HARD” configuration.*
Predictability

Standard JVM

Fiji VM

>100ms pauses!

no pauses!

*CDx benchmark “HARD” configuration.
Predictability

Standard JVM

Fiji VM

*CDx benchmark “HARD” configuration.

>100ms pauses!

no pauses!

completely predictable execution!
Against C code on RTEMS/LEON3
Against C code on RTEMS/LEON3

C code
Against C code on RTEMS/LEON3

Java, no GC

C code
Against C code on RTEMS/LEON3
Against C code on RTEMS/LEON3

Java, with GC

Java, no GC

C code

C worst case is 160 ms
Against C code on RTEMS/LEON3

Java, with GC

Java, no GC worst case is 226 ms

C code

Java no GC worst case is 226 ms
Against C code on RTEMS/LEON3

Java with GC worst case is 233 ms

Java, with GC

Java, no GC

C code

Java with GC worst case is 233 ms

Friday, November 6, 2009
Java's worst-case is 45% worse than C's.
Against C code on RTEMS/LEON3

GC’s effect on predictability: Worst-case goes up by 3%
Correlation: C vs. Java

Fiji VM Java iteration execution time

Fiji code iteration execution time
Correlation: C vs. Java

Thread preemption latency

C code iteration execution time vs. Fiji VM Java iteration execution time
Correlation: C vs. Java

Fiji VM Java iteration execution time

C code iteration execution time

Thread latency

preemption

GC's only effect on predictability is due to the fact that otherwise, there are no other threads running in CDx.
Performance summary

- Fiji VM is currently the **fastest** Java implementation.
- Performance within 45% of C code.
- Garbage collection does not interfere with program execution.
  - but GC does increase load on scheduler by introducing one additional thread.
Integration

Fiji VM

Rapita

JML

Eclipse IDE
Summary

- Fiji VM allows Java code to run quickly and predictably.
- Safety Critical Java promises a more cost-effective development process for safety-critical systems.

http://www.fiji-systems.com/
Fiji VM Code Statistics

- Size of runtime:
  - 17,000 lines of C code
  - 11,000 lines of Java code
  - *this includes code for all configurations.*

- Size of compiler:
  - 3,000 lines of Ruby code
  - 44,000 lines of Java code