Evolving Java's Floating Point Support:

The Good, the Bad, and the Ugly

Joseph D. Darcy

formerly of the University of California, Berkeley
now at Sun Microsystems

The Value of Diversity

Panic! UNIX System Crash Dump Analysis
by Chris Drake and Kimberly Brown

"For complete technical information about the [Floating-point Status Register] and the contents of some of its fields, refer to both the SPARC Version 8 Specification and the ANSI/IEEE Standard 754-1985,

preferably on nights when you can't sleep."

Outline

Background on IEEE 754

Programming Languages and Floating Point

- Base conversion
- Supporting different floating point systems (IEEE 754, VAX, Cray, etc.)
- Implementation shortcomings

Languages and Extensions

- Borneo
- Sun's PEJFPS
- Java Grande Numerics
- JVM 1.2

Background: Basic IEEE 754

An IEEE 754 format is specified by two numbers, K and N.

- K+1 exponent bits, N significand bits
- Together K and N determine the size of the format

Finite values representable in a format are given by allowing two parameters k and n to range over a set of values in the formula

$$\text{finite value} = 2^{k+1-N} \cdot n$$

- exponent: $1\text{-}2^K < k < 2^K$
- significand: $\text{-}2^N < n < 2^N$
- subnormals in IEEE 754 allow gradual underflow (no break in range of n around zero)
- also IEEE 754 special values
  - NaN (Not a Number)
  - ±infinity

More IEEE 754 features

- The standard defines a number of formats of different sizes

	Total bits, $N+(K+1)$	bits for N	bits for $K+1$.	E_{max}	E_{min}
`single`	32	24	8	127	-126
`double`	64	53	11	1023	-1022
`double extended`	≥ 79	≥ 64	≥ 15	≥ 16383	≥ -16382

- Defined operations, +, -, *, /, √, conversions, comparison
- Result computed as if an infinitely precise intermediate result is calculated, then rounded to the destination format according to the current dynamic rounding mode
  - default mode is round to nearest even (no statistical bias, prevents numerical drift in certain loops)
  - round to +∞ and round to -∞ (used for finding error bounds, e.g. interval arithmetic)
  - round to 0 (convert floating point to integer, error analysis)
- Rounding mode usually stored in FPU control register

IEEE 754 Exceptional Conditions

- Default exception handling policy can be overridden by the programmer
- Two mechanisms to deal with exceptional conditions
  - Default mode (required), set corresponding sticky flag and continue with (possibly) special value (used by Java *without* the flags)
  - Trapping mode (optional), execute a user-defined subroutine, trap handlers can be set independently for all five conditions (maps to language exceptions in Borneo)
- Sticky flags in FPU status register, trapping status in FPU control register
- Five exceptional conditions and default responses
  - **Invalid** ($\sqrt{-1.0}$, $\infty - \infty$, ∞*0.0, 0.0/0.0, ∞/∞, etc.)**:** continue with a NaN.
    Invalid is also signaled when some comparison operators have a NaN operand
  - **Overflow:** continue with $\pm\infty$ or $\pm MAX\_VALUE$ (depending on rounding mode)
  - **Divide by Zero:** continue with $\pm\infty$
  - **Underflow:** continue with a subnormal or 0.0
  - **Inexact** (due to rounding, usually ignored)**:** ordinary value

The Good: Base Conversion in Java

- Base conversion is usually omitted from language specs; Java requires correctly rounded decimal to binary and binary to decimal conversion (actually available in both directions since JDK 1.1)
- incorrect binary ⇔ decimal conversion creates inconsistencies
  - runtime vs. compile time differences (documented in IBM FORTRAN II)
  - hard to trust delivered answers, obscures correctness
- despite multiple publications in widely-read conferences [1990 PLDI], problem persists
  - Microsoft math: Windows 3.1 calculator `2.01-2.00=0.00` !)
  - bug in a MS C library function, not attributable to hardware roundoff

The Good: IEEE 754 required in Java

Java mandates IEEE 754 `float` and `double` formats

- Jettisons limitations from legacy of compatibility with Cray, VAX, and IBM 370 arithmetic (c.f. C9X)
- Simplified library development
- Disciplined exact reproducibility feasible
- Greatly enhances *predictability*

The Bad: Theory and Practice

- Expression evaluation rules
  - K&R C vs. ANSI C, Java chose ANSI C
  - widest available policy protects against unrecognized numerical instabilities
- *Exactly* reproducible floating point often not needed
- Java vendors, including Sun, release non-conforming environments
  - faulty decimal to binary conversion in JDK 1.0.2
  - exponent limit complications on the x86
  - transcendental function library

The Bad: Subsetting IEEE 754

- Must use IEEE 754 formats *but* required features are disallowed
  - Directed rounding modes and floating point exceptions are explicitly forbidden (JLS §4.2.4)
  - Sticky flags are omitted
- Floating point types
  - Only `float` and `double` are supported
  - Java lacks a name for the beneficial `double extended` format (some C compilers map `long double` to `double extended`)
- Can't use simpler, faster, more understandable and maintainable robust algorithms, e.g.
  "Faster Numerical Algorithms via Exception Handling," Demmel and Li, IEEE Transactions on Computers, vol. 43, no. 8, August 1994, pp. 983-992
- Denies features helpful to ordinary programmers

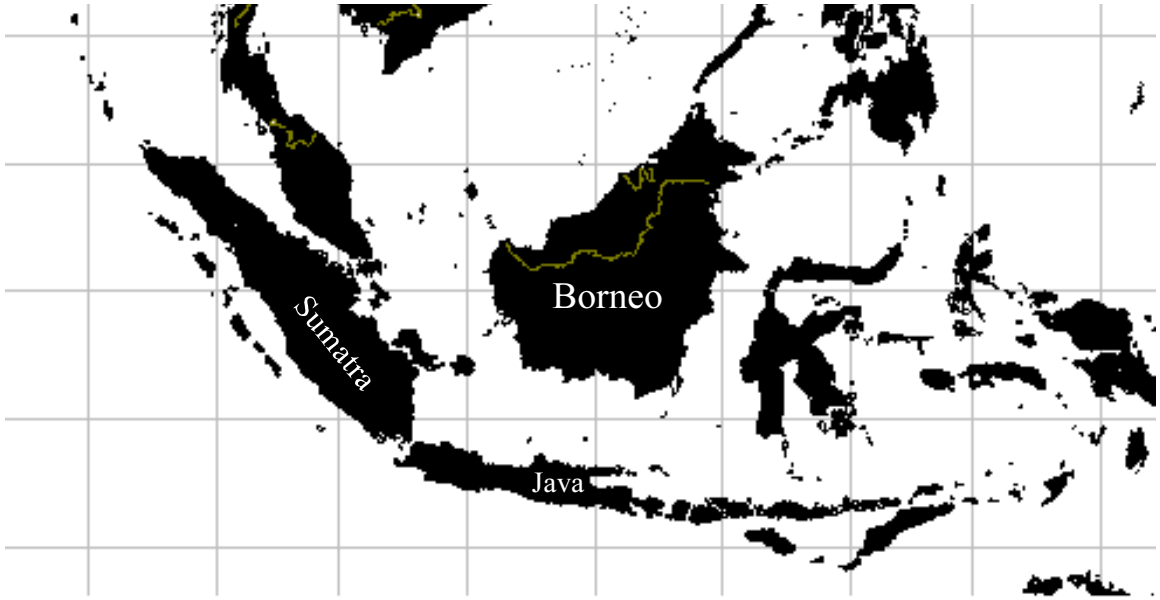
Detecting Numerical Sensitivities

Two formulae to calculate the area of a triangle, one unstable (but commonly found), one stable (less widely known)

Rounding Mode	**Heron's Formula** $s=((a+b)+c)/2$ $\sqrt{s\cdot(s-a)\cdot(s-b)\cdot(s-c)}$ (unstable if evaluated in `float` precision)	$\frac{\sqrt{(a+(b+c))\cdot(c-(a-b))\cdot(c+(a-b))\cdot(a+(b-c))}}{4}$ (stable with $a \geq b \geq c$)	**Heron's Formula** (stable if `float` data evaluated in `double` precision)
a=12345679, b=12345678, c=1.01233995 > $a-b$			
to nearest	0.00	972730.06	972730.06
to +∞	17459428.00	972730.25	972730.06
to -∞	0.00	972729.88	972730.00
to 0	-0.00	972729.88	972730.00
a=12345679, b=12345679, c=1.01233995 > $a-b$			
to nearest	12345680.00	6249012.00	6249012.00
to +∞	12345680.00	6249013.00	6249012.50
to -∞	0.00	6249011.00	6249012.00
to 0	0.00	6249011.00	6249012.00

- can be used by non-expert programmers to find who to blame
- doesn't require the source code
- anomalies very "spiky," perturbing data can hide the problem
- (technique supported indirectly in Borneo via extra-lingual method call and code generation requirements)

The Better: Borneo 1.0.2†



- Includes all required and recommended IEEE 754 features
- Design constraint: *upwards compatible with Java 1.0*
- Given a Java class *P* compiled to bytecode, another Java class cannot determine whether *P* was compiled under Borneo or Java semantics

†Formerly known as Teak. In the future will be known as Kalimantan.

Borneo Features

Carryover from Java

- correctly rounded base conversion and requiring IEEE 754

Add

- IEEE 754 semantics, sticky flags are an observable side affect
- libraries using flexible operator overloading (interval, etc.)
- primitive floating point type `indigenous` to access the `double extended` floating point format where available
- language declarations
  - `anonymous` *FloatingPointType* to specify "Old C" expression evaluation
  - control IEEE 754 features
    - sticky flags, control flow and method interface
    - rounding modes

Language Control of IEEE 754: Rounding Modes

- compiler handles messy details, better programmer interface
- allows specialized compiler optimizations and improves compiler and programmer ability to reason about the code
- lexically scoped, callee does not inherit from caller (less global state)
- default rounding mode round to nearest
- (language backdoor to support fully dynamic rounding)

```
// Syntactically
// in Borneo
{
  rounding new_rm;
  Some computation...
}
```

```
// Logically
{
  int saved_rm = getRound();
  setRound(new_rm);
  Some computation...
  setRound(saved_rm);
}
```

```
// Correct Implementation
// in Java
{
  int saved_rm = getRound();
  try
    {
      setRound(new_rm);
      Some computation...
    }
  finally
    {
       setRound(saved_rm);
    }
}
```

Java and the x86: What is the issue?

- The x86 most naturally operates on 80-bit `double extended` register values
- What about just setting the rounding precision?
  - the x86 can be made to round to `double` precision or `float` precision *but*
  - only the significand is restricted, *not* the exponent
  - to restrict the exponent, a store to memory is required
  - the store to memory doesn't change the value in the register
  - to continue the computation, the rounded value has to be reloaded from memory
  - excess memory traffic degrades performance
  - works correctly for `float`, but not for `double`

Pure `double` on the x86

- double rounding of `double` subnormals can lead to (literally) a very small difference from pure `double`, $\approx 10^{-324}$
- future subnormal answer is rounding once (to full precision) when first computed, rounded again (to reduced precision) when stored to memory
- two-step rounding of full precision value to a subnormal can differ from direct rounding to a subnormal
- eliminating the double rounding on underflow discrepancy can be very expensive
  - 10X slowdown reported using previously known methods
  - a new technique gives a factor of 2X to 3X slowdown (comparable to current practice that doesn't quite conform to Java semantics)

The Ugly: PEJFPS

Proposal for Extension of Java™ Floating Point Semantics, Revision 1, May 1998

formerly found at

`http://java.sun.com/feedback/fp.html`

Summary of PEJFPS

- Goals:
  - allowed some access to extended precision where it is supported by hardware
  - ameliorated Java's floating point performance implications on the x86
- Some `float` and `double` values could be stored as and operated on as extended format floating point values.
- Method qualifiers `widefp` and `strictfp` controlled contexts where extended formats could and could not be used.
- The virtual machine decided when and whether to use extended formats.
- Existing Java source code and `class` files were subject to the revised semantics

A change in philosophy

Java allows application developers to write a program once and then be able to run it everywhere on the Internet.

Except for timing dependencies or other non-determinisms and given sufficient time and sufficient memory space, a Java program should compute the same result on all machines and in all implementations.
—Preface to The Java™ Language Specification

- For both intrinsic and practical reasons, Java code does not live up to it "write once, run anywhere" slogan
- But, Java is much more *predictable* than other contemporary languages. The sizes of the types are given, expression evaluation order is specified, etc.
- PEJFPS would have removed Java's predictability for floating point

Compiler Latitude

Under PEJFPS, the compiler decided to use or not use extended precision at its discretion. From PEJFPS,

Section 5.1.8, Format Conversion
Within an FP-wide expression, format conversion *allows an implementation, at its option, to perform any of the following operations on a value:*

- float → float extended ***and*** float extended → float
- double → double extended ***and*** double extended → double

Conclusion:

- extended formats could be used *inconsistently* at the compiler's whim

Do cry over spilt registers

Will breaking an expression into pieces change the value computed?

```
// widefp context
double a, t1, t2;
a = BigExpression1 * BigExpression2;
// will common subexpression elimination happen?
t1 = BigExpression1;
t2 = BigExpression2;
if(a == t1 * t2)
   ...
```

- Faster to register spill 64 bit `double` values instead of 80 bit `double extended` values (lower latency instruction and less memory traffic)
- For no good reason breaks referential transparency present in Java 1.0

Everything old is new again

- Sun III compilers used extended precision for anonymous values but had no language type corresponding to `double extended`

Those who cannot remember the past are condemned to repeat it.
– George Santayana
The Life of Reason, vol. 1, Reason in Common Sense

- Lack of a language type caused problems since the register's `doubled extended` value of an expression assigned to a `double` variable could be used in place of the rounded `double` value stored in the variable
- Analogous problems exists on some FORTRAN and C compilers for the x86, recently discussed in the `comp.compilers` newsgroup

The Better: Java Grande

- What is Java Grande? (see `http://www.javagrande.org`)
  - Industrial, academic, and research community effort to advise on how to evolve Java to be a suitable environment for "grande" computation
  - Numerics and Concurrency working groups
- "Improving Java for Numerical Computation"
  - Serves the interests of non-exert programmers better than Java
  - Describes reasonable and desirable floating point semantics
    - `strictfp` (Java 1.0)
    - default, allow some extended exponent range on the x86
    - `associativefp`, allow the compiler to rewrite floating point code as if the floating point operations were associative (used to generate fast platform-specific matrix multiply routine)
  - Identifies issues related to "lightweight classes" and operator overloading
  - Provides detailed examples of code generation on the x86

Faster pure `double` on the x86

- A refinement of existing store-reload technique from Roger Golliver of Intel
- store-reload works for addition and subtraction
- for multiplication
  - scale down one of the operands by $2^{(E_{max}\,\texttt{double extended} - E_{max}\,\texttt{double})}$
  - perform the operation
  - rescale product up to proper range
  - perform a store-reload to enforce proper overflow threshold
- analogous idiom for division
- exact emulation marginally more expensive than plain store-reload
- slowdown factor of 2 to 4 instead of 10 (includes exception optimization)
- No special testing is needed to handle ±0.0, infinities, and NaN
- IEEE sticky flags are set properly and the technique works under dynamic rounding modes

The Good: Borneo + Java Grande

- Borneo covers adding IEEE 754 support as well as operator overloading and lightweight classes
- Java Grande describes practical code generation techniques for the x86 and outlines different floating point semantics
- Together, Borneo and Java Grande provide a greatly enriched numerical programming environment

The not too bad: JVM 1.2

- The floating point aspects of the JVM specification are being updated
- methods in `class` files can use “strict” or “default” floating point semantics
- in default mode, floating point values on the operand stack may have extended *exponents*
- does not preclude adding more comprehensive IEEE 754 support in the future
- to be completely predictable, need guidelines for compiling Java to JVM

Variable elimination

- Will `b` be represented as a variable in the `class` file?

```
double a;
...
{
  double b;
  b = 2.0  * a;
  a = 0.25 * b;
}
```

  - the computation involving `b` could be done entirely on the operand stack
  - the operand stack can use extended exponent range while variables cannot ∴ different answers can result
  - unknown range much less harmful than unknown precision

Shaping the Future of Java and JVM

- IEEE 754 capabilities were designed for a mass market
- Java is striving for a mass market
- IEEE 754 would benefit programmers
- Sun should add IEEE 754 support to Java
  - adding IEEE 754 features need not compromise other aspects of Java
  - need to get sticky flag and rounding mode support in the VM and IEEE 754 semantics into the language
- Support will not be added without the insistence and guidance of the programming community
- Disaster was averted, to help build something better, get involved!

Acknowledgments

Borneo was designed with the help of William Kahan and Alex Aiken.

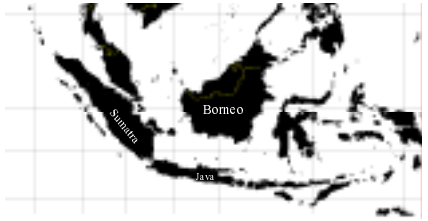
Students in the spring 1997 offering of UCB CS 279 also helped design Borneo and write the early specification. Borneo has benefited from the feedback of various readers included Gregory Tarsy's floating point group at Sun and several Berkeley CS graduate students.

References and Related Work

- Jerome Coonen, “A Note On Java Numerics,” January 25, 1997, Numeric Interest Mailing List, `http://www.validgh.com/`.
- Jerome Coonen, “A Proposal for RealJava, Draft 1.0,” July 4 1997, Numeric Interest Mailing List.
- C9X, `http://www.dkuug.dk/JTC1/SC22/WG14/`.
- J. Demmel and X. Li, “Faster Numerical Algorithms via Exception Handling,” IEEE Transactions on Computers, vol. 43, no. 8, August 1994, pp. 983-992.
- Roger A. Golliver, “First-implementation artifacts in Java™”.
- James Gosling, *The Evolution of Numerical Computing in Java*, `http://java.sun.com/people/jag/FP.html`.
- “Making Java Work for High-End Computing,” Java Grande Forum, `http://www.javagrande.org/`, also `http://math.nist.gov/javanumerics/`.
- Tim Lindholm and Frank Yellin, *The Java™ Virtual Machine Specification*, Addison-Wesley, 1996.
- “Proposal for Extension of Java™ Floating Point Semantics, Revision 1,” May 1998, stale URL `http://java.sun.com/feedback/fp.html`.

Self-promotion

For more information on Borneo:
`http://www.cs.berkeley.edu/~darcy/Borneo`



For a discussion of Java's floating point support:
How Java's Floating-Point Hurts Everyone Everywhere
Professor William Kahan and Joseph D. Darcy
`http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf`

For a critique of PEJFPS:
`http://www.cs.berkeley.edu/~darcy/Research/jgrande.ps.gz`
Lively discussions of PEJFPS are archived at
`http://www.validgh.com/java`

For these slides:
`http://www.cs.berkeley.edu/~darcy/Research/cascon.ps.gz`

Comments? email: `darcy@cs.berkeley.edu`