Comparison between binary64 and decimal64 floating-point numbers

Nicolas Brisebarre Christoph Lauter Marc Mezzarobba
Jean-Michel Muller

ARITH 21, Austin, TX

Decimal and Binary FP numbers

- IEEE 754-2008 Std for FP Arithmetic: several binary and decimal formats
  - Binary format: (s, E, m) representing $(-1)^s \cdot 2^E \cdot m$
  - Decimal format: (s, E, m) representing $(-1)^s \cdot 10^E \cdot m$

Decimal and Binary FP numbers

- IEEE 754-2008 Std for FP Arithmetic: several binary and decimal formats
  - Binary format: (s, E, m) representing $(-1)^s \cdot 2^E \cdot m$
  - Decimal format: (s, E, m) representing $(-1)^s \cdot 10^E \cdot m$

- The Standard *does not* require (yet does not *forbid*) that a binary and a decimal number be comparable

Comparisons between Decimal and Binary Numbers

- Such comparisons may nevertheless offer several advantages
  - Databases may contain both decimal and binary data
  - Binary data might need to be forced to stay in decimal bounds

- *Conversion* before comparison *is not an option*
  - small inaccuracies
  - possibly non-terminating programs
  - conversions are actually more expensive

Decimal and Binary FP numbers

Consider

```
double x = ...;
_Decimal64 y = ...;
if (x <= y) {
    ...
}
```

- This code gets translated by Intel's icc 12.1.3 into
  - a conversion from binary to decimal
  - followed by a decimal comparison
- No warning that the boolean result might not be the expected one because of rounding

Comparison to decimal constants

Assume that x is the binary64 number

$$\frac{3602879701896397}{2^{55}} = 0.1000000000000000055511151231\cdots$$

- $x > 0.1$
- x is the binary64 number nearest 0.1
- x is less than the binary32 number nearest 0.1

Comparison to decimal constants

Assume that x is the binary64 number

$$\frac{3602879701896397}{2^{55}} = 0.1000000000000000055511151231\cdots$$

- $x > 0.1$
- x is the binary64 number nearest 0.1
- x is less than the binary32 number nearest 0.1

- The test `x > 0.1D` should return `true`

→ We get `false` instead if `0.1D` is converted to binary64

Comparison to decimal constants

Assume that x is the binary64 number

$$\frac{3602879701896397}{2^{55}} = 0.1000000000000000055511151231\cdots$$

- $x > 0.1$
- x is the binary64 number nearest 0.1
- x is less than the binary32 number nearest 0.1

- The test `x > 0.1D` should return `true`

→ We get `false` instead if `0.1D` is converted to binary64

- The test `x < 0.1D` should return `false`

→ We get `true` instead if `0.1D` is converted to binary32

Problem statement

We aim at introducing an algorithm for comparing a binary64

$$x_2 = M_2 \cdot 2^{e_2 - 52}$$

to a decimal64

$$x_{10} = M_{10} \cdot 10^{e_{10} - 15}$$

- M_2 and M_{10} are integers, with $|M_2| < 2^{53}$ and $|M_{10}| < 10^{16}$
- e_2 and e_{10} are the FP exponents of x_2 and x_{10}

Problem statement (cont'd)

- The algorithm is to set the flags foreseen for comparisons:
  - Invalid must be set for NaNs according to IEEE 754-2008
  - Inexact is not to be touched
    $\rightarrow$ algorithm using integer arithmetic only

Problem statement (cont'd)

- The algorithm is to set the flags foreseen for comparisons:
  - Invalid must be set for NaNs according to IEEE 754-2008
  - Inexact is not to be touched
    $\rightarrow$ algorithm using integer arithmetic only

- We restrict ourselves to binary64 and decimal64
- We assume that the *binary encoding* is used for the decimal64
  $\rightarrow$ M_{10} is easily accessible in binary.

Problem statement (cont'd)

- The algorithm is to set the flags foreseen for comparisons:
  - Invalid must be set for NaNs according to IEEE 754-2008
  - Inexact is not to be touched
    $\rightarrow$ algorithm using integer arithmetic only

- We restrict ourselves to binary64 and decimal64
- We assume that the *binary encoding* is used for the decimal64
  $\rightarrow$ M_{10} is easily accessible in binary.

- Straightforward comparisons if x_2 and x_{10} have different signs or are 0
  $\rightarrow$ we assume that $M_2 > 0$ and $M_{10} > 0$

An easy first step in comparing x_2 and x_{10}

Comparison between $x_2 = M_2 \cdot 2^{e_2 - 52}$ and $x_{10} = M_{10} \cdot 10^{e_{10} - 15}$

Step 1

If x_2 and x_{10} have significantly different orders of magnitude, examining their exponents suffices to compare them

An easy first step in comparing x_2 and x_{10}

Comparison between $x_2 = M_2 \cdot 2^{e_2 - 52}$ and $x_{10} = M_{10} \cdot 10^{e_{10} - 15}$

Step 1

If x_2 and x_{10} have significantly different orders of magnitude, examining their exponents suffices to compare them

$\rightarrow$ First perform an exponent-based test

An easy first step in comparing x_2 and x_{10}

Comparison between $x_2 = M_2 \cdot 2^{e_2 - 52}$ and $x_{10} = M_{10} \cdot 10^{e_{10} - 15}$

Step 1

If x_2 and x_{10} have significantly different orders of magnitude, examining their exponents suffices to compare them

$\rightarrow$ First perform an exponent-based test

$\rightarrow$ Issue: Decimal FP significands are not normalized

Normalizing the significands

- A first idea: normalize M_{10} into one decade
- This would yield $10^{15} \leqslant M'_{10} \leqslant 10^{16} - 1$

Normalizing the significands

- A first idea: normalize M_{10} into one decade
- This would yield $10^{15} \leqslant M'_{10} \leqslant 10^{16} - 1$
- However:
  - The normalization would require a loop
    $\rightarrow$ pipelining issues
  - Several binades fall into one decade: $1 < 2 < 4 < 8 < 10$
    $\rightarrow$ one decimal exponent would match several binary exponents
    $\rightarrow$ inefficient first step

Normalizing the significands

- A first idea: normalize M_{10} into one decade
- This would yield $10^{15} \leqslant M'_{10} \leqslant 10^{16} - 1$
- However:
  - The normalization would require a loop
    $\rightarrow$ pipelining issues
  - Several binades fall into one decade: $1 < 2 < 4 < 8 < 10$
    $\rightarrow$ one decimal exponent would match several binary exponents
    $\rightarrow$ inefficient first step
- Normalization of M_{10} into a *binade* is also possible:
- Since $M_{10} < 10^{16} < 2^{54}$, there exists one integer ν, $0 \leqslant \nu \leqslant 53$ such that

$$2^{53} \leqslant 2^{\nu} M_{10} \leqslant 2^{54} - 1$$

- The problem reduces to comparing $M_2 \cdot 2^{e_2 - 52 + \nu}$ and $2^{\nu} M_{10} \cdot 10^{e_{10} - 15}$

Comparison after normalization

Our comparison problem becomes:

$$\text{Compare } m \cdot 2^h \text{ with } n \cdot 5^g$$

where

$$\begin{aligned} 2^{52} \leqslant m &\leqslant 2^{53} - 1 \\ 2^{53} \leqslant n &\leqslant 2^{54} - 1 \\ -398 \leqslant g &\leqslant 369, \\ -1495 \leqslant h &\leqslant 1422 \end{aligned} \tag{1}$$

Idea:

- $\rightarrow$ m and n both live in a single *binade*
- $\rightarrow$ The "exponents" g and h are *closely related*

Relating the exponents g and h

Compare $m \cdot 2^h$ with $n \cdot 5^g$

Define two functions φ and ψ:

$$\varphi(h) = \lfloor h \cdot \log_5 2 \rfloor, \qquad \psi(g) = \lfloor g \cdot \log_2 5 \rfloor \tag{2}$$

The function φ makes the first comparison step:

Property 1

We have

$$g < \varphi(h) \Rightarrow x_2 > x_{10}$$
$$g > \varphi(h) \Rightarrow x_2 < x_{10}$$

Computation of $\varphi(h)$ and $\psi(g)$ is easy

$\varphi(h) = \lfloor h \cdot \log_5 2 \rfloor$ and $\psi(g)$ involve transcendental constants

Computation of $\varphi(h)$ and $\psi(g)$ is easy

$\varphi(h) = \lfloor h \cdot \log_5 2 \rfloor$ and $\psi(g)$ involve transcendental constants

Property 2

Let be

$$L_\varphi = \lfloor 2^{19} \log_5 2 \rceil = 225799,$$
$$L_\psi = \lfloor 2^{12} \log_2 5 \rceil = 9511.$$

For all h in the range (1)*, we have* $\varphi(h) = \lfloor h \cdot L_\varphi \cdot 2^{-19} \rfloor$.
Similarly,

- $\psi(g) = \lfloor L_\psi \cdot g \cdot 2^{-12} \rfloor$ *for* $|g| \leqslant 204$
- $\psi(16q) = \lfloor L_\psi \cdot q \cdot 2^{-8} \rfloor$ *for* $|q| \leqslant 32$.

- Products $L_\varphi \cdot h$ and $L_\psi \cdot g$: exact 32-bit integer arithmetic
- Computing $\lfloor \xi \cdot 2^{-\beta} \rfloor$ reduces to a right-shift by β bits

Algorithm for the first, exponent-based, step

Algorithm 1

First step

- *Compute* $h = \nu + e_2 - e_{10} - 37$ *and* $g = e_{10} - 15$*;*
- *Compute* $\varphi(h) = \lfloor L_\varphi \cdot h \cdot 2^{-19} \rfloor$ *using 32-bit integer arithmetic;*
- *If* $g < \varphi(h)$ *then return* "$x_2 > x_{10}$"
  else if $g > \varphi(h)$ *then return* "$x_2 < x_{10}$"
  else perform the second step.

$\rightarrow$ This first test eliminates 99.9% of the inputs
(shown under mild assumptions)

Second step: a closer look at the significands

We're comparing $m \cdot 2^h$ to $n \cdot 5^g$

If the first step does not succeed, we can assume that

$$g = \varphi(h) = \lfloor h \cdot \log_5 2 \rfloor$$

Second step: a closer look at the significands

We're comparing $m \cdot 2^h$ to $n \cdot 5^g$

If the first step does not succeed, we can assume that

$$g = \varphi(h) = \lfloor h \cdot \log_5 2 \rfloor$$

Define

$$f(h) = 5^{\varphi(h)} \cdot 2^{-h} = 2^{\lfloor h \log_5 2 \rfloor \cdot \log_2 5 - h}$$

We have

$$\begin{cases} f(h) \cdot n > m \Rightarrow x_{10} > x_2 \\ f(h) \cdot n < m \Rightarrow x_{10} < x_2 \\ f(h) \cdot n = m \Rightarrow x_{10} = x_2 \end{cases} \tag{3}$$

Second test: perform that comparison of $f(h) \cdot n$ and m

Second step: a closer look at the significands

We're comparing $m \cdot 2^h$ to $n \cdot 5^g$

If the first step does not succeed, we can assume that

$$g = \varphi(h) = \lfloor h \cdot \log_5 2 \rfloor$$

Define

$$f(h) = 5^{\varphi(h)} \cdot 2^{-h} = 2^{\lfloor h \log_5 2 \rfloor \cdot \log_2 5 - h}$$

We have

$$\begin{cases} f(h) \cdot n > m \Rightarrow x_{10} > x_2 \\ f(h) \cdot n < m \Rightarrow x_{10} < x_2 \\ f(h) \cdot n = m \Rightarrow x_{10} = x_2 \end{cases} \tag{3}$$

Second test: perform that comparison of $f(h) \cdot n$ and m

Issue: $f(h)$ needs to be replaced with an approximation

How accurately must $f(h)$ be evaluated?

- How accurately must $f(h) = 5^{\varphi(h)} \cdot 2^{-h}$ be evaluated in

$$\begin{cases} f(h) \cdot n > m \Rightarrow x_{10} > x_2, \\ f(h) \cdot n < m \Rightarrow x_{10} < x_2, \\ f(h) \cdot n = m \Rightarrow x_{10} = x_2. \end{cases} \tag{3}$$

  to ensure that the approximate test is equivalent to (3)?
- Find a lower bound η on the minimum nonzero value of

$$\left| \frac{m}{n} - f(h) \right| = \left| \frac{m}{n} - \frac{5^{\varphi(h)}}{2^h} \right|$$

  that may appear at this stage
- We want η to be as tight as possible in order to avoid unduly costly computations when approximating $f(h) \cdot n$

Minimum nonzero value of $\left|\frac{m}{n} - \frac{5^{\varphi(h)}}{2^h}\right|$?

The search space is constrained by the following observations:

- First, remember that

$$2^{52} \leqslant m \leqslant 2^{53} - 1,$$
$$2^{53} \leqslant n \leqslant 2^{54} - 1$$

- Second, we have

Lemma 1

The equality $g = \varphi(h)$ can hold only if $-787 \leqslant h \leqslant 716$.

Additionally,

- *if $n \geqslant 10^{16}$, then n is necessarily even;*
- *if $h \geqslant 680$, then $\nu' = h + \varphi(h) - 971 > 0$ and $2^{\nu'}$ divides n.*

Minimum nonzero value of $d_h(m,n) = \left|\frac{m}{n} - \frac{5^{\varphi(h)}}{2^h}\right|$

Minimum nonzero value of $d_h(m,n) = \left|\frac{m}{n} - \alpha\right|$ with $\alpha = \frac{5^{\varphi(h)}}{2^h}$

- A classical approximation problem
- Continued fractions are the tool to be readily employed
- Taking into account all the constraints needs some care

Minimum nonzero value of $d_h(m,n) = \left|\frac{m}{n} - \frac{5^{\varphi(h)}}{2^h}\right|$

Minimum nonzero value of $d_h(m,n) = \left|\frac{m}{n} - \alpha\right|$ with $\alpha = \frac{5^{\varphi(h)}}{2^h}$

- A classical approximation problem
- Continued fractions are the tool to be readily employed
- Taking into account all the constraints needs some care

Theorem 2

The minimum nonzero value of $d_h(m,n)$ under the given constraints is

$$6.0485\ldots \times 10^{-35} < 2^{-113.67},$$

attained for

$$h = -275, \quad \frac{m}{n} = \frac{4988915232824583}{12364820988483254}.$$

We may thus take $\eta = 2^{-113.7}$

Implementing the comparison

Second step: efficiently evaluate $f(h) \cdot n$ with just enough accuracy

Implementing the comparison

Second step: efficiently evaluate $f(h) \cdot n$ with just enough accuracy

Theorem 3

Assume that μ approximates $f(h) \cdot n$ with relative accuracy $\epsilon < \eta/4$ or better, that is,

$$|f(h) \cdot n - \mu| \leqslant \epsilon f(h) \cdot n < \frac{\eta}{4} f(h) \cdot n. \tag{4}$$

The following implications hold:

$$\begin{cases} \mu > m + \epsilon \cdot 2^{54} \Longrightarrow x_{10} > x_2, \\ \mu < m - \epsilon \cdot 2^{54} \Longrightarrow x_{10} < x_2, \\ |m - \mu| \leqslant \epsilon \cdot 2^{54} \Longrightarrow x_{10} = x_2. \end{cases} \tag{5}$$

A direct table-based method

- A first implementation:
  - Compute an approximation $\mu \approx f(h) \cdot n$
  - Theorems 2 and 3: 128 bit arithmetic is enough
  - $f(h)$ comes from a table, directly indexed with h
    each table entry contains the 128 high-order bits of $f(h)$

A direct table-based method

- A first implementation:
  - Compute an approximation $\mu \approx f(h) \cdot n$
  - Theorems 2 and 3: 128 bit arithmetic is enough
  - $f(h)$ comes from a table, directly indexed with h
    each table entry contains the 128 high-order bits of $f(h)$
- Advantages of a direct method:
  - h is available in an early stage of the algorithm
    $\rightarrow$ memory latency of table fetch for $f(h)$ can be hidden
  - The algorithm boils down to a table read and
    one 128×64 bit multiplication

A direct table-based method

- A first implementation:
  - Compute an approximation $\mu \approx f(h) \cdot n$
  - Theorems 2 and 3: 128 bit arithmetic is enough
  - $f(h)$ comes from a table, directly indexed with h
    each table entry contains the 128 high-order bits of $f(h)$
- Advantages of a direct method:
  - h is available in an early stage of the algorithm
    $\rightarrow$ memory latency of table fetch for $f(h)$ can be hidden
  - The algorithm boils down to a table read and
    one 128×64 bit multiplication
- Disadvantages:
  - Table size is large: 23.5 KB
  - Direct application of Theorem 5 requires complex branching

Reducing the table size

- The table is large and contains some redundancies

Reducing the table size

- The table is large and contains some redundancies
- Transform the problem

$$f(h) \cdot n \overset{?}{<} m$$

Reducing the table size

- The table is large and contains some redundancies
- Transform the problem

$$2^{-h} \cdot 5^{\varphi(h)} \cdot n \overset{?}{<} m$$

Reducing the table size

- The table is large and contains some redundancies
- Transform the problem

$$2^{-h} \cdot 5^{g} \cdot n \overset{?}{<} m$$

where $g = \varphi(h)$

Reducing the table size

- The table is large and contains some redundancies
- Transform the problem to use 2 tables

$$2^{-h} \cdot 5^{16q-r} \cdot n \quad \overset{?}{<} \quad m$$

where $g = \varphi(h)$, $q = \lfloor \frac{g}{16} + 1 \rfloor$

Reducing the table size

- The table is large and contains some redundancies
- Transform the problem to use 2 tables

$$2^{-h} \cdot 5^{16q} \cdot n \quad \overset{?}{<} \quad 5^{r} \cdot m$$

where $g = \varphi(h)$, $q = \lfloor \frac{g}{16} + 1 \rfloor$

Reducing the table size

- The table is large and contains some redundancies
- Transform the problem to use 2 tables

$$5^{16q} \cdot n \quad \stackrel{?}{<} \quad 5^r \cdot m \cdot 2^h$$

where $g = \varphi(h)$, $q = \lfloor \frac{g}{16} + 1 \rfloor$

Reducing the table size

- The table is large and contains some redundancies
- Transform the problem to use 2 tables

$$2^{-\psi(16q)+127} \cdot 5^{16q} \cdot n \cdot 2^{-64} \overset{?}{<} 2^{-\psi(r)+63} \cdot 5^{r} \cdot m \cdot 2^{\sigma}$$

where $g = \varphi(h)$, $q = \lfloor \frac{g}{16} + 1 \rfloor$, $\psi(g) = \lfloor g \log_2 5 \rfloor$

Reducing the table size

- The table is large and contains some redundancies
- Transform the problem to use 2 tables

$$\theta_1(q) \cdot n \cdot 2^{-64} \quad \stackrel{?}{<} \quad \theta_2(r) \cdot m \cdot 2^{\sigma(h)}$$

  where $g = \varphi(h)$, $q = \lfloor \frac{g}{16} + 1 \rfloor$, $\psi(g) = \lfloor g \log_2 5 \rfloor$
- Two new tables

$$\begin{aligned} \theta_1(q) &= 2^{-\psi(16q)+127} \cdot 5^{16q} \\ \theta_2(r) &= 2^{-\psi(r)+63} \cdot 5^{r} \end{aligned}$$

- An easy-to-compute shift amount $\sigma(h) = \psi(r) - \psi(16q) + h$

A bipartite table approach

- Two new tables

$$\begin{aligned} \theta_1(q) &= 2^{-\psi(16q)+127} \cdot 5^{16q} \\ \theta_2(r) &= 2^{-\psi(r)+63} \cdot 5^{r} \end{aligned}$$

- These tables are really small:

$$-20 \leqslant q \leqslant 21$$

$$1 \leqslant r \leqslant 16$$

  $\rightarrow$ Both tables hold on 800 bytes of memory

- The table for $\theta_2(r)$ can be stored exactly as $5^r \leqslant 5^{16} < 2^{38}$

Compensating rounding errors

- The comparison problem reduces to computing the sign of

$$\Delta = \theta_1(q) \cdot n \cdot 2^{-64} - \theta_2(r) \cdot m \cdot 2^{\sigma(h)}$$

- The second table, $\theta_2(r)$ is exactly stored with 64 bit entries $\rightarrow \theta_2(r) \cdot m \cdot 2^{\sigma(h)}$ can be computed exactly on 128 bits

Compensating rounding errors

- The comparison problem reduces to computing the sign of

$$\Delta = \theta_1(q) \cdot n \cdot 2^{-64} - \theta_2(r) \cdot m \cdot 2^{\sigma(h)}$$

- The second table, $\theta_2(r)$ is exactly stored with 64 bit entries
  $\rightarrow \theta_2(r) \cdot m \cdot 2^{\sigma(h)}$ can be computed exactly on 128 bits
- The first table, $\theta_1(q)$ cannot be stored exactly
  $\rightarrow$ It contains the 128 high order bits of 5^{16q}
- On the other hand, the exact product $\theta_1(q) \cdot n$ is on 192 bits
  $\rightarrow$ rounding to 128 bits is necessary

Compensating rounding errors

- The comparison problem reduces to computing the sign of

$$\Delta = \theta_1(q) \cdot n \cdot 2^{-64} - \theta_2(r) \cdot m \cdot 2^{\sigma(h)}$$

- The second table, $\theta_2(r)$ is exactly stored with 64 bit entries
  $\rightarrow \theta_2(r) \cdot m \cdot 2^{\sigma(h)}$ can be computed exactly on 128 bits
- The first table, $\theta_1(q)$ cannot be stored exactly
  $\rightarrow$ It contains the 128 high order bits of 5^{16q}
- On the other hand, the exact product $\theta_1(q) \cdot n$ is on 192 bits
  $\rightarrow$ rounding to 128 bits is necessary
- When replacing Δ with

$$\tilde{\Delta} = \lfloor \lceil \theta_1(q) \rceil \cdot n \cdot 2^{-64} \rfloor - \theta_2(r) \cdot m \cdot 2^{\sigma(h)}$$

  both rounding errors compensate each other when $\Delta = 0$
- The sign of the easy-to-compute $\tilde{\Delta}$ yields the correct answer

The bipartite table-based second step

Algorithm 2

Step 2 (bipartite table-based)

- *Compute* q, r, $\sigma(h) = \psi(r) - \psi(16q) + h$ *as defined above.*
- *Read the 128 bits of* $\lceil \theta_1(q) \rceil$ *as two 64 bit words.*
- *Compute the 128 high-order bits* α *of* $\lceil \theta_1(q) \rceil \cdot (n2^8)$ *with one* 128×64*-bit multiplication, dropping the 64 low order bits.*
- *Compute* $\beta = \theta_2(r) \cdot (m2^{8+\sigma(h)})$ *with one full* 64×64*-bit multiplication, keeping all* 128 *bits.*
- *Compute the signed difference* $\tilde{\Delta} = \alpha - \beta$.
- *Return*

$$\begin{cases} \text{“}x_2 > x_{10}\text{”} & \text{if } \tilde{\Delta} < 0, \\ \text{“}x_2 = x_{10}\text{”} & \text{if } \tilde{\Delta} = 0, \\ \text{“}x_2 < x_{10}\text{”} & \text{if } \tilde{\Delta} > 0. \end{cases}$$

Experimental results

- The "direct method" and the "bipartite table method" have been implemented and thoroughly tested
- Plain C, with the decimal FP type support from `gcc` or `icc`
- We gave priority to portability over performance

Test vectors:

- Extensively cover all FP input classes (normal numbers, subnormals, NaNs, ± 0, $\pm\infty$), for both x_2 and x_{10}
- Exercise the worst-cases (m, n) for the critical value $d_h(m, n) = \left|5^{\varphi(h)}/2^h - m/n\right|$ for each admissible value of h
- Large set of random inputs that fully exercise all possible values of the normalization $2^\nu M_{10}$ for the decimal significand

Performance results

Performance results:

	Naïve method converting x_2 to decimal64 *(incorrect)* min/avg/max	Direct method min/avg/max	Bipartite table method min/avg/max	Bipartite table method (optimized implementation) min/avg/max
Special cases (± 0, NaNs, Inf)	14/–/254	8/–/114	7/–/132	7/–/112
x_2 subnormal, x_{10} of same sign	87/137/297	173/199/307	196/219/298	23/54/158
x_2 normal, x_{10} of opposite sign	50/144/278	5/32/118	5/30/121	4/38/122
x_2 normal, same sign, "easy" cases	84/156/294	15/47/144	15/46/122	6/47/122
x_2 normal, same sign, "hard" cases	32/95/283	34/58/212	52/70/226	19/44/198

on an Intel Core i7 M620 @ 2.7GHz with 64bit Linux 3.2.0-2 using gcc 4.6.3-1 with `-O3 -march=native`

Conclusion

- Exact comparisons between binary and decimal FP formats:
  - an enrichment to the current FP environment
  - an enhancement for the safety and provability of FP software

Conclusion

- Exact comparisons between binary and decimal FP formats:
  - an enrichment to the current FP environment
  - an enhancement for the safety and provability of FP software
- Algorithms for mixed-radix comparisons are efficient:
  - A fast exponent-based test eliminates most comparison inputs
  - Remaining cases:
    - Use of approximations just accurate enough
    - A direct method and a bipartite table technique

Conclusion

- Exact comparisons between binary and decimal FP formats:
  - an enrichment to the current FP environment
  - an enhancement for the safety and provability of FP software
- Algorithms for mixed-radix comparisons are efficient:
  - A fast exponent-based test eliminates most comparison inputs
  - Remaining cases:
    - Use of approximations just accurate enough
    - A direct method and a bipartite table technique
- Our implementation:
  - Both methods proven and thoroughly tested
  - Just 800 bytes of table space for the bipartite table method
  - Performance higher than for the naive comparison technique

Conclusion

- Exact comparisons between binary and decimal FP formats:
  - an enrichment to the current FP environment
  - an enhancement for the safety and provability of FP software
- Algorithms for mixed-radix comparisons are efficient:
  - A fast exponent-based test eliminates most comparison inputs
  - Remaining cases:
    - Use of approximations just accurate enough
    - A direct method and a bipartite table technique
- Our implementation:
  - Both methods proven and thoroughly tested
  - Just 800 bytes of table space for the bipartite table method
  - Performance higher than for the naive comparison technique
- Future work:
  - other common IEEE 754-2008 binary and decimal formats
  - reuse of tables for comparison and conversion

Q/A

Thank you for your attention!

Questions?