



HAL
open science

Comparison between binary and decimal floating-point numbers

Nicolas Brisebarre, Christoph Lauter, Marc Mezzarobba, Jean-Michel Muller

► **To cite this version:**

Nicolas Brisebarre, Christoph Lauter, Marc Mezzarobba, Jean-Michel Muller. Comparison between binary and decimal floating-point numbers. 2014. hal-01021928v1

HAL Id: hal-01021928

<https://hal.science/hal-01021928v1>

Preprint submitted on 9 Jul 2014 (v1), last revised 25 Jun 2015 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Comparison between binary and decimal floating-point numbers

Nicolas Brisebarre, Christoph Lauter, Marc Mezzarobba, and Jean-Michel Muller

Abstract—We introduce an algorithm to compare a binary floating-point (FP) number and a decimal FP number, assuming the “binary encoding” of the decimal formats is used, and with a special emphasis on the basic interchange formats specified by the IEEE 754-2008 standard for FP arithmetic. It is a two-step algorithm: a first pass, based on the exponents only, quickly eliminates most cases, then, when the first pass does not suffice, a more accurate second pass is performed. We provide an implementation of several variants of our algorithm, and compare them.

1 INTRODUCTION

The IEEE 754-2008 Standard for Floating-Point Arithmetic [5] specifies binary (radix-2) and decimal (radix-10) floating-point number formats for a variety of precisions. The so-called “basic interchange formats” are presented in Table 1.

The Standard neither requires nor forbids comparisons between floating-point numbers of different radices (it states that *floating-point data represented in different formats shall be comparable as long as the operands’ formats have the same radix*). However, such “mixed-radix” comparisons may offer several advantages. It is not infrequent to read decimal data from a database and to have to compare it to some binary floating-point number. The comparison may be inaccurate if the decimal number is preliminarily converted to binary, or, respectively, if the binary number is first converted to decimal.

Consider for instance the following C code:

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

The standardization of decimal floating-point arithmetic in C [6] is still at draft stage, and compilers supporting decimal floating-point arithmetic handle code sequences such as the previous one at their discretion and often in an unsatisfactory way. As it occurs, Intel’s icc 12.1.3 translates this sequence into a conversion from binary to

	binary32	binary64	binary128
precision (bits)	24	53	113
e_{\min}	-126	-1022	-16382
e_{\max}	+127	+1023	+16383

	decimal64	decimal128
precision (digits)	16	34
e_{\min}	-383	-6143
e_{\max}	+384	+6144

TABLE 1
The basic binary and decimal interchange formats specified by the IEEE 754-2008 Standard.

decimal followed by a decimal comparison. The compiler emits no warning that the boolean result might not be the expected one because of rounding.

This kind of strategy may lead to inconsistencies. Consider such a “naïve” approach built as follows: when comparing a binary floating-point number x_2 of format \mathcal{F}_2 , and a decimal floating-point number x_{10} of format \mathcal{F}_{10} , we first convert x_{10} to the binary format \mathcal{F}_2 (that is, we replace it by the \mathcal{F}_2 number nearest x_{10}), and then we perform the comparison in binary. Denote the comparison operators so defined as \odot , \otimes , \oslash , and \ominus . Consider the following variables (all exactly represented in their respective formats):

- $x = 3602879701896397/2^{55}$, declared as a binary64 number;
- $y = 13421773/2^{27}$, declared as a binary32 number;
- $z = 1/10$, declared as a decimal64 number.

Then it holds that $x \odot y$, but also $y \otimes z$ and $z \otimes x$. Such an inconsistent result might for instance suffice to prevent a sorting program from terminating.

Remark that there exists a longer intermediate format both inputs x_2 and x_{10} could be converted to such that the naïve method yields a correct answer. However, that method requires a correctly rounded radix conversion, which is an intrinsically more expensive operation than

- N. Brisebarre and J.-M. Muller are with CNRS, Laboratoire LIP, ENS Lyon, INRIA, Université Claude Bernard Lyon 1, Lyon, France. E-mail: nicolas.brisebarre@ens-lyon.fr, jean-michel.muller@ens-lyon.fr
- C. Lauter is with Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France. E-mail: christoph.lauter@lip6.fr
- This work was initiated while M. Mezzarobba was with the Research Institute for Symbolic Computation, Johannes Kepler University, Linz, Austria. M. Mezzarobba is now with CNRS, UMR 7606, LIP6, F-75005, Paris, France; Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France. E-mail: marc@mezzarobba.net
- This work was partly supported by the TaMaDi project of the French Agence Nationale de la Recherche, and by the Austria Science Fund (FWF) grants P22748-N18 and Y464-N18.

the comparison itself.

In the following, we describe algorithms to perform such “exact” comparisons between binary numbers in any of the basic binary interchange formats and decimal numbers in any of the basic decimal interchange formats.

It is natural to require that mixed-radix comparisons not signal any floating-point exception, except in situations that parallel exceptional conditions specified in the Standard [5, Section 5.11] for regular comparisons. For this reason, we avoid floating-point operations that might signal exceptions, and mostly use integer arithmetic to implement our comparisons.

2 SETTING AND OUTLINE

We consider a binary format of precision p_2 , minimum exponent e_2^{\min} and maximum exponent e_2^{\max} , and a decimal format of precision p_{10} , minimum exponent e_{10}^{\min} and maximum exponent e_{10}^{\max} . We want to compare a binary floating-point number x_2 and a decimal floating-point number x_{10} .

Without loss of generality we assume $x_2 > 0$ and $x_{10} > 0$ (when x_2 and x_{10} are negative, the problem reduces to comparing $-x_2$ and $-x_{10}$, and when they have different signs the comparison is straightforward). The floating-point representations of x_2 and x_{10} are

$$\begin{aligned} x_2 &= M_2 \cdot 2^{e_2 - p_2 + 1}, \\ x_{10} &= M_{10} \cdot 10^{e_{10} - p_{10} + 1}, \end{aligned}$$

where M_2, M_{10}, e_2 and e_{10} are integers that satisfy:

$$\begin{aligned} e_2^{\min} - p_2 + 1 &\leq e_2 \leq e_2^{\max}, \\ e_{10}^{\min} &\leq e_{10} \leq e_{10}^{\max}, \\ 2^{p_2 - 1} &\leq M_2 \leq 2^{p_2} - 1, \\ 1 &\leq M_{10} \leq 10^{p_{10}} - 1 \end{aligned} \quad (1)$$

with $p_2, p_{10} \geq 1$. (The choice of lower bound for e_2 makes the condition $2^{p_2 - 1} \leq M_2$ hold even for subnormal binary numbers.)

We assume that the so-called *binary encoding* [5], [8] of IEEE 754-2008 is used for the decimal format, so that the integer M_{10} is easily accessible in binary. Denote by

$$p'_{10} = \lceil p_{10} \log_2 10 \rceil,$$

the number of bits that are necessary for representing the decimal significands in binary.

When x_2 and x_{10} have significantly different orders of magnitude, examining their exponents will suffice to compare them. Hence, we first perform a simple exponent-based test (Section 3). When this does not suffice, a second step compares the significands multiplied by suitable powers of 2 and 5. We compute “worst cases” that determine how accurate the second step needs to be (Section 4), and show how to implement it efficiently (Section 5). Section 6 is an aside describing a simpler algorithm that can be used if we only want to decide whether x_2 and x_{10} are equal. Finally, Section 7 discusses our implementation of the comparison algorithms and presents experimental results.

3 FIRST STEP: ELIMINATING THE “SIMPLE CASES” BY EXAMINING THE EXPONENTS

3.1 Normalization

As we have

$$1 \leq M_{10} \leq 10^{p_{10}} - 1,$$

there exists a unique $\nu \in \{0, 1, 2, \dots, p'_{10} - 1\}$ such that

$$2^{p'_{10} - 1} \leq 2^\nu M_{10} \leq 2^{p'_{10}} - 1.$$

Our initial problem of comparing x_2 and x_{10} reduces to comparing $M_2 \cdot 2^{e_2 - p_2 + 1 + \nu}$ and $(2^\nu M_{10}) \cdot 10^{e_{10} - p_{10} + 1}$.

The fact that we “normalize” the decimal significand M_{10} by a binary shift between two consecutive powers of two is of course questionable; M_{10} could also be normalized into the range $10^{p_{10} - 1} \leq 10^t \cdot M_{10} \leq 10^{p_{10}} - 1$. However, as hardware support for decimal floating-point arithmetic is not widespread, a decimal normalization would require a loop, provoking pipeline stalls, whereas the proposed binary normalization can exploit an existing hardware leading-zero counter with straight-line code.

3.2 Comparing the Exponents

Define

$$\begin{cases} m = M_2, \\ h = e_2 - e_{10} + \nu + p_{10} - p'_{10} + 1, \\ n = M_{10} \cdot 2^\nu, \\ g = e_{10} - p_{10} + 1, \\ w = p'_{10} - p_2 - 1 \end{cases} \quad (2)$$

so that

$$\begin{cases} 2^\nu x_2 = m \cdot 2^{h+g+w}, \\ 2^\nu x_{10} = n \cdot 10^g. \end{cases}$$

Our comparison problem becomes:

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

We have

$$\begin{aligned} m_{\min} = 2^{p_2 - 1} &\leq m \leq m_{\max} = 2^{p_2} - 1, \\ n_{\min} = 2^{p'_{10} - 1} &\leq n \leq n_{\max} = 2^{p'_{10}} - 1. \end{aligned} \quad (3)$$

It is clear that $m_{\min} \cdot 2^{h+w} > n_{\max} \cdot 5^g$ implies $x_2 > x_{10}$, while $m_{\max} \cdot 2^{h+w} < n_{\min} \cdot 5^g$ implies $x_2 < x_{10}$. This gives

$$\begin{aligned} (1 - 2^{-p'_{10}}) \cdot 5^g < 2^{h-2} &\Rightarrow x_{10} < x_2, \\ (1 - 2^{-p_2}) \cdot 2^h < 5^g &\Rightarrow x_2 < x_{10}. \end{aligned} \quad (4)$$

In order to compare x_2 and x_{10} based on these implications, define

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

Proposition 1. *We have*

$$\begin{aligned} g < \varphi(h) &\Rightarrow x_2 > x_{10}, \\ g > \varphi(h) &\Rightarrow x_2 < x_{10}. \end{aligned}$$

	b32/d64	b32/d128	b64/d64	b64/d128	b128/d64	b128/d128
p_2	24	24	53	53	113	113
p_{10}	16	34	16	34	16	34
p'_{10}	54	113	54	113	54	113
w	29	88	0	59	-60	-1
$h_{\min}^{(1)}, h_{\max}^{(1)}$	-570, 526	-6371, 6304	-1495, 1422	-7296, 7200	-16915, 16782	-22716, 22560
s_{\min}	18	23	19	23	27	27
datatype	int32	int64	int32	int64	int64	int64

TABLE 2
The various parameters involved in Step 1 of the comparison algorithm.

Proof: If $g < \varphi(h)$ then $g \leq \varphi(h) - 1$, hence $g \leq h \log_5 2 - 1$. This implies that $5^g \leq (1/5) \cdot 2^h < 2^h / (4 \cdot (1 - 2^{-p'_{10}}))$, therefore, from (4), $x_{10} < x_2$. If $g > \varphi(h)$ then $g \geq \varphi(h) + 1$, hence $g > h \log_5 2$, so that $5^g > 2^h > (1 - 2^{-p_2}) \cdot 2^h$. This implies, from (4), $x_2 < x_{10}$. \square

Now consider the range of h : by (2), h lies between

$$h_{\min}^{(1)} = (e_2^{\min} - p_2 + 1) - e_{10}^{\max} + p_{10} - p'_{10} + 1$$

and

$$h_{\max}^{(1)} = e_2^{\max} - e_{10}^{\min} + p_{10}.$$

That range is given in Table 2 for the basic IEEE formats. Knowing that range, it is easy to implement φ as follows.

Proposition 2. Denote by $\lfloor \cdot \rfloor$ the nearest integer function. For large enough $s \in \mathbb{N}$, the function defined by

$$\hat{\varphi}(h) = \lfloor L \cdot h \cdot 2^{-s} \rfloor, \quad \text{with } L = \lfloor 2^s \log_5 2 \rfloor,$$

satisfies $\varphi(h) = \hat{\varphi}(h)$ for all h in the range $[h_{\min}^{(1)}, h_{\max}^{(1)}]$.

Proposition 2 is an immediate consequence of the irrationality of $\log_5 2$. For known, moderate values of $h_{\min}^{(1)}$ and $h_{\max}^{(1)}$, the optimal choice s_{\min} of s is easy to find and small. For instance, if the binary format is binary64 and the decimal format is decimal64, then $s_{\min} = 19$.

Table 2 gives the value of s_{\min} for the basic IEEE formats. The product $L \cdot h$, for h in the indicated range and $s = s_{\min}$, can be computed exactly in (signed or unsigned) integer arithmetic, with the indicated data type. Computing $\lfloor \xi \cdot 2^{-\beta} \rfloor$ of course reduces to a right-shift by β bits.

Propositions 1 and 2 yield to the following algorithm.

Algorithm 1. First, exponent-based step

- 1 compute $h = e_2 - e_{10} + \nu + p_{10} - p'_{10} + 1$ and $g = e_{10} - p_{10} + 1$;
- 2 with the appropriate value of s , compute $\varphi(h) = \lfloor L \cdot h \cdot 2^{-s} \rfloor$ using integer arithmetic;
- 3 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.

Note that, when x_{10} admits multiple distinct representations in the precision- p_{10} decimal format (i.e., when its cohort is non-trivial [5]), the success of the first step may depend on the specific representation passed as

input. For instance, assume that the binary and decimal formats are binary64 and decimal64, respectively. Both $A = \{M_{10} = 10^{15}, e_{10} = 0\}$ and $B = \{M_{10} = 1, e_{10} = 15\}$ are valid representations of the integer 1. Assume we are trying to compare $x_{10} = 1$ to $x_2 = 2$. Using representation A , we have $\nu = 4$, $h = -32$, and $\varphi(h) = -14 > g = -15$, hence the test from Algorithm 1 shows that $x_{10} < x_2$. In contrast, if x_{10} is given in the form B , we get $\nu = 53$, $\varphi(h) = \varphi(2) = 0 = g$, and the test is inconclusive.

3.3 How Often is the First Step Enough?

We may quantify the quality of this first filter as follows. We say that Algorithm 1 *succeeds* if it answers “ $x_2 > x_{10}$ ” or “ $x_2 < x_{10}$ ” without proceeding to the second step, and *fails* otherwise. Let X_2 and X_{10} denote the sets of *representations* of positive, finite numbers in the binary, resp. decimal formats of interest. (In the case of X_2 , each number has a single representation.) Assuming zeros, infinities, and NaNs have been handled before, the input of Algorithm 1 may be thought of as a pair $(\xi_2, \xi_{10}) \in X_2 \times X_{10}$.

Proposition 3. The proportion of input pairs $(\xi_2, \xi_{10}) \in X_2 \times X_{10}$ for which Algorithm 1 fails is bounded by

$$\min \left\{ \frac{1}{e_{10}^{\max} - e_{10}^{\min} + 1}, \frac{4}{e_2^{\max} - e_2^{\min} + 1} \right\},$$

assuming $p_2 \geq 3$.

Proof: The first step fails if and only if $\varphi(h) = g$. Write $h = e_2 + t$, that is, $t = p_{10} - p'_{10} + 1 - e_{10} + \nu$. Then $\varphi(h) = g$ rewrites as

$$\lfloor (e_2 + t) \log_5 2 \rfloor = g,$$

which implies

$$\begin{aligned} -t + g \log_2 5 &\leq e_2 < -t + (g + 1) \log_2 5 \\ &< -t + g \log_2 5 + 2.4. \end{aligned}$$

The value of ν is determined by M_{10} , so that t and $g = e_{10} - p_{10} + 1$ depend on ξ_{10} only. Thus, for any given $\xi_{10} \in X_{10}$, there can be at most 3 values of e_2 for which $\varphi(h) = g$.

A similar argument shows that for given e_2 and M_{10} , there exist at most one value of e_{10} such that $\varphi(h) = g$.

Let X_2^{norm} and X_2^{sub} be the subsets of X_2 consisting of normal and subnormal numbers respectively. Let $r_i = e_i^{\text{max}} - e_i^{\text{min}} + 1$. The number N_{norm} of pairs $(\xi_2, \xi_{10}) \in X_2^{\text{norm}} \times X_{10}$ such that $\varphi(h) = g$ satisfies

$$\begin{aligned} N_{\text{norm}} &\leq \#\{M_{10} : \xi_{10} \in X_{10}\} \cdot \#\{M_2 : \xi_2 \in X_2^{\text{norm}}\} \\ &\quad \cdot \#\{(e_2, e_{10}) : \xi_2 \in X_2^{\text{norm}} \wedge \varphi(h) = g\} \\ &\leq (10^{p_{10}} - 1) \cdot 2^{p_2 - 1} \cdot \min\{r_2, 3r_{10}\}. \end{aligned}$$

For subnormal x_2 , we have the bound

$$\begin{aligned} N_{\text{sub}} &:= \#\{(\xi_2, \xi_{10}) \in X_2^{\text{sub}} \times X_{10} : \varphi(h) = g\} \\ &\leq \#\{M_{10}\} \cdot \#\{X_2^{\text{sub}}\} \\ &= (10^{p_{10}} - 1) \cdot (2^{p_2 - 1} - 1). \end{aligned}$$

The total number of elements of $X_2 \times X_{10}$ for which the first step fails is bounded by $N_{\text{norm}} + N_{\text{sub}}$. This is to be compared with

$$\begin{aligned} \#X_2 &= r_2 \cdot 2^{p_2 - 1} + (p_2 - 1) \cdot (2^{p_2 - 1} - 1) \\ &\geq (r_2 + 1) \cdot 2^{p_2 - 1}, \\ \#X_{10} &= r_{10} \cdot (10^{p_{10}} - 1). \end{aligned}$$

We obtain

$$\frac{N_{\text{norm}} + N_{\text{sub}}}{\#(X_2 \times X_{10})} \leq \min\left\{\frac{1}{r_{10}}, \frac{4}{r_2}\right\}. \quad \square$$

These are rough estimates. One way to get tighter bounds for specific formats is simply to count, for each value of ν , the pairs (e_2, e_{10}) such that $\varphi(h) = g$. For instance, in the case of comparison between binary64 and decimal64 floating-point numbers, one can check that the failure rate in the sense of Proposition 3 is less than 0.1%. As a matter of course, pairs (ξ_2, ξ_{10}) will almost never be equidistributed in practice. Hence the previous estimate should not be interpreted as a *probability* of success of Step 1. It seems more realistic to assume that a well-written numerical algorithm will mostly perform comparisons between numbers which are suspected to be close to each other. For instance, in an iterative algorithm where comparisons are used as part of a convergence test, it is to be expected that most comparisons need to proceed to the second step. Conversely, there are scenarios, e.g., checking for out-of-range data, where the first step should be enough.

4 SECOND STEP: A CLOSER LOOK AT THE SIGNIFICANDS

4.1 Problem Statement

In the following we assume that $g = \varphi(h)$, i.e.,

$$e_{10} - p_{10} + 1 = \lfloor (e_2 - e_{10} + \nu + p_{10} - p'_{10} + 1) \log_5(2) \rfloor.$$

(Otherwise, the first step already allowed us to compare x_2 and x_{10} .)

Define a function

$$f(h) = \frac{5^{\varphi(h)}}{2^{h+w}}.$$

We have

$$\begin{cases} f(h) \cdot n > m \Rightarrow x_{10} > x_2, \\ f(h) \cdot n < m \Rightarrow x_{10} < x_2, \\ f(h) \cdot n = m \Rightarrow x_{10} = x_2. \end{cases} \quad (5)$$

The second test consists in performing this comparison, with $f(h) \cdot n$ replaced by an accurate enough approximation.

In order to ensure that an approximate test is indeed equivalent to (5), we need a lower bound η on the minimum nonzero value of

$$d_h(m, n) = \left| \frac{5^{\varphi(h)}}{2^{h+w}} - \frac{m}{n} \right| \quad (6)$$

that may appear at this stage. We want η to be as tight as possible in order to avoid unduly costly computations when approximating $f(h) \cdot n$. The search space is constrained by the following observations.

Proposition 4. *The equality $g = \varphi(h)$ where g and h are defined by (2) can hold only if*

$$\frac{e_2^{\text{min}} - p_2 - p'_{10} + 3}{1 + \log_5 2} \leq h < \frac{e_2^{\text{max}} + 2}{1 + \log_5 2}. \quad (7)$$

Additionally, n satisfies the following properties:

- 1) if $n \geq 10^{p_{10}}$, then n is even;
- 2) if $\nu' = h + \varphi(h) - e_2^{\text{max}} + p'_{10} - 2$ is nonnegative (which holds for large enough h), then $2^{\nu'}$ divides n .

Proof: From (2), assuming $g = \varphi(h)$, we get

$$e_2 = h + g + p'_{10} - \nu - 2,$$

so that (since $e_2^{\text{min}} - p_2 + 1 \leq e_2 \leq e_2^{\text{max}}$)

$$e_2^{\text{min}} - p_2 + 1 \leq h + \varphi(h) - \nu + p'_{10} - 2 \leq e_2^{\text{max}},$$

hence

$$e_2^{\text{min}} - p_2 + \nu - p'_{10} + 3 \leq (1 + \log_5 2)h < e_2^{\text{max}} + \nu - p'_{10} + 3,$$

and (7) follows because $0 \leq \nu \leq p'_{10} - 1$.

The binary normalization of M_{10} , yielding n , implies that $2^\nu \mid n$. If $n \geq 10^{p_{10}}$, then, by (1) and (2), it follows that $\nu \geq 1$ and n is even. As $h + \varphi(h) - \nu \leq e_2^{\text{max}} - p'_{10} + 2$, we also have $\nu \geq \nu'$. Since φ is increasing, there exists h_0 such that $\nu' \geq 0$ for $h \geq h_0$. \square

Table 3 gives the range (7) for h with respect to the comparisons between basic IEEE formats.

4.2 Required Worst-Case Accuracy

Let us now deal with the problem of computing η , considering $d_h(m, n)$ under the constraints given by Equation (3) and Proposition 4. A similar problem was considered by Cornea et al. [3] in the context of correctly rounded binary to decimal conversions. Their techniques yield worst and bad cases for the approximation problem we consider. We will take advantage of them in Section 7 to test our algorithms on many cases when x_2 and x_{10} are very close. Here, we favor a different approach that is

	b32/d64	b32/d128	b64/d64	b64/d128	b128/d64	b128/d128
$h_{\min}^{(2)}, h_{\max}^{(2)}$	-140, 90	-181, 90	-787, 716	-828, 716	-11565, 11452	-11606, 11452
$g_{\min}^{(2)}, g_{\max}^{(2)}$	-61, 38	-78, 38	-339, 308	-357, 308	-4981, 4932	-4999, 4932
# g	100	117	648	666	9914	9932
h_0	54	12	680	639	11416	11375

TABLE 3

Range of h for which we may have $g = \varphi(h)$, and size of the table used in the “direct method” of Section 5.1.

less computationally intensive and mathematically simpler, making the results easier to check either manually or with a proof-assistant.

Problem 1. Find the smallest nonzero value of

$$d_h(m, n) = \left| \frac{5^{\varphi(h)}}{2^{h+w}} - \frac{m}{n} \right|$$

subject to the constraints

$$\begin{cases} 2^{p_2-1} \leq m \leq 2^{p_2} - 1, \\ 2^{p'_{10}-1} \leq n \leq 2^{p'_{10}} - 1, \\ h_{\min}^{(2)} \leq h \leq h_{\max}^{(2)}, \\ n \text{ is even if } n \geq 10^{p'_{10}}, \\ \text{if } h \geq h_0, \text{ then } 2^{\nu'} \mid n \end{cases}$$

where

$$\begin{aligned} h_{\min}^{(2)} &= \left\lceil \frac{e_2^{\min} - p_2 - p'_{10} + 3}{1 + \log_5 2} \right\rceil, \\ h_{\max}^{(2)} &= \left\lfloor \frac{e_2^{\max} + 2}{1 + \log_5 2} \right\rfloor, \\ h_0 &= \left\lfloor \frac{e_2^{\max} - p'_{10} + 3}{1 + \log_5 2} \right\rfloor, \\ \nu' &= h + \varphi(h) - e_2^{\max} + p'_{10} - 2. \end{aligned}$$

We recall how such a problem can be solved using the classical theory of continued fractions [4], [7], [9].

Given $\alpha \in \mathbb{Q}$, build two finite sequences $(a_i)_{0 \leq i \leq n}$ and $(r_i)_{0 \leq i \leq n}$ by setting $r_0 = \alpha$ and

$$\begin{cases} a_i = \lfloor r_i \rfloor, \\ r_{i+1} = 1/(r_i - a_i) \text{ if } a_i \neq r_i. \end{cases}$$

For all $0 \leq i \leq n$, the rational number

$$\frac{p_i}{q_i} = a_0 + \frac{1}{a_1 + \frac{1}{\ddots + \frac{1}{a_i}}}$$

is called the i th convergent of the continued fraction expansion of α . Observe that the process defining the a_i essentially is the Euclidean algorithm. If we assume $p_{-1} = 1$, $q_{-1} = 0$, $p_0 = a_0$, $q_0 = 1$, we have $p_{i+1} = a_{i+1}p_i + p_{i-1}$, $q_{i+1} = a_{i+1}q_i + q_{i-1}$, and $\alpha = p_n/q_n$.

It is a classical result [7, Proposition 19] that any rational number m/n such that

$$\left| \alpha - \frac{m}{n} \right| \leq \frac{1}{2n^2}$$

is a convergent of the continued fraction expansion of α . For basic IEEE formats, it turns out that this classical result is enough to solve Problem 1, as cases when it is not precise enough can be handled in an ad hoc way.

First consider the case $\max(0, -w) \leq h \leq p_2$. We can then write

$$d_h(m, n) = \frac{|5^{\varphi(h)}n - 2^{h+w}m|}{2^{h+w}n}$$

where the numerator and denominator are both integers. If $d_h(m, n) \neq 0$, we have $|5^{\varphi(h)}n - m2^{h+w}| \geq 1$, hence $d_h(m, n) \geq 1/(2^{h+w}n) > 2^{p_2-h-2p'_{10}+1} \geq 2^{-2p'_{10}+1}$. For the range of h where $d_h(m, n)$ might be zero, the nonzero minimum is thus no less than $2^{-2p'_{10}+1}$.

If now $p_2 + 1 \leq h \leq h_{\max}^{(2)}$, then $h + w \geq p'_{10}$, so $5^{\varphi(h)}$ and 2^{h+w} are integers, and $m2^{h+w}$ is divisible by $2^{p'_{10}}$. As $n \leq 2^{p'_{10}} - 1$, we have $d_h(m, n) \neq 0$. To start with, assume that $d_h(m, n) \leq 2^{-2p'_{10}-1}$. Then, the integers m and $n \leq 2^{p'_{10}} - 1$ satisfy

$$\left| \frac{5^{\varphi(h)}}{2^{h+w}} - \frac{m}{n} \right| \leq 2^{-2p'_{10}-1} < \frac{1}{2n^2},$$

and hence m/n is a convergent of the continued fraction expansion of $\alpha = 5^{\varphi(h)}/2^{h+w}$.

We compute the convergents with denominators less than $2^{p'_{10}}$ and take the minimum of the $|\alpha - p_i/q_i|$ for which there exists $k \in \mathbb{N}$ such that $m = kp_i$ and $n = kq_i$ satisfy the constraints of Problem 1. Notice that this strategy only yields the actual minimum nonzero of $d_h(m, n)$ if we eventually find, for some h , a convergent with $|\alpha - p_i/q_i| < 2^{-2p'_{10}-1}$. Otherwise, taking $\eta = 2^{-2p'_{10}-1}$ yields a valid (yet not tight) lower bound.

The case $h_{\min}^{(2)} \leq h \leq \min(0, -w)$ is similar. A numerator of $d_h(m, n)$ then is $|2^{-h-w}n - 5^{-\varphi(h)}m|$ and a denominator is $5^{-\varphi(h)}n$. We notice that $5^{-\varphi(h)} \geq 2^{p'_{10}}$ if and only if $h \leq -p'_{10}$. If $-p'_{10} + 1 \leq h \leq \min(0, -w)$ and $d_h(m, n) \neq 0$, we have $|2^{-h-w}n - 5^{-\varphi(h)}m| \geq 1$, hence $d_h(m, n) \geq 1/(5^{-\varphi(h)}n) \geq 2^{-2p'_{10}}$. For $h_{\min}^{(2)} \leq h \leq -p'_{10}$, we use the same continued fraction tools as above.

There remains the case $\min(0, -w) \leq h \leq \max(0, -w)$. If $0 \leq h \leq -w$, a numerator of $d_h(m, n)$ is $|5^{\varphi(h)}2^{-h-w}n - m|$ and a denominator is n . Hence, if $d_h(m, n) \neq 0$, we have $d_h(m, n) \geq 1/n > 2^{p'_{10}}$. If $-w \leq h \leq 0$, a numerator of $d_h(m, n)$ is $|n - 5^{-\varphi(h)}2^{h+w}m|$ and a denominator is $5^{-\varphi(h)}2^{h+w}n \leq 5 \cdot 2^{-h}2^{h+w}n < 5 \cdot 2^{2p'_{10}-p_2-1}$. It follows that if $d_h(m, n) \neq 0$, we have $d_h(m, n) \geq 1/5 \cdot 2^{-2p'_{10}+p_2+1}$.

	h	m	n	$\log_2(\eta^{-1})$
b32/d64	50	10888194	13802425659501406	111.40
b32/d128	-159	11386091	8169119658476861812680212016502305	229.57
b64/d64	-275	4988915232824583	12364820988483254	113.68
b64/d128	-818	5148744585188163	9254355313724266263661769079234135	233.58
b128/d64	2546	7116022508838657793249305056613439	13857400902051554	126.77
b128/d128	10378	7977485665655127446147737154136553	9844227914381600512882010261817769	237.14

TABLE 4
Worst cases for $d_h(m, n)$ and corresponding lower bounds η .

Proposition 5. *The minimum nonzero values of $d_h(m, n)$ under the constraints from Problem 1 for the basic IEEE formats are attained for the parameters given in Table 4.*

5 INEQUALITY TESTING

As already mentioned, the first step of the full comparison algorithm is straightforwardly implementable in 32-bit or 64-bit integer arithmetic. The second step reduces to comparing m and $f(h) \cdot n$, and we have seen in Section 4 that it is enough to know $f(h)$ with a relative accuracy given by the last column of Table 4. We now discuss several ways to perform that comparison. The main difficulty is to efficiently evaluate $f(h) \cdot n$ with just enough accuracy.

5.1 Direct Method

A direct implementation of Equation (5) just replaces $f(h)$ with a sufficiently accurate approximation, read in a table indexed with h . The actual accuracy requirements can be stated as follows. Recall that η denotes a (tight) lower bound on (6).

Proposition 6. *Assume that μ approximates $f(h) \cdot n$ with relative accuracy $\epsilon < \eta/2^{-w+2}$ or better, that is,*

$$|f(h) \cdot n - \mu| \leq \epsilon f(h) \cdot n < \frac{\eta}{2^{-w+2}} f(h) \cdot n. \quad (8)$$

The following implications hold:

$$\begin{cases} \mu > m + \epsilon \cdot 2^{p_2+1} \implies x_{10} > x_2, \\ \mu < m - \epsilon \cdot 2^{p_2+1} \implies x_{10} < x_2, \\ |m - \mu| \leq \epsilon \cdot 2^{p_2+1} \implies x_{10} = x_2. \end{cases} \quad (9)$$

Proof: First notice that $f(h) \leq 2^{-w} = 2^{p_2-p'_{10}+1}$ for all h . Since $n < 2^{p'_{10}}$, condition (8) implies $|\mu - f(h) \cdot n| < 2^{p_2+1}\epsilon$, and hence $|f(h) \cdot n - m| > |\mu - m| - 2^{p_2+1}\epsilon$.

Now consider each of the possible cases that appear in (9). If $|\mu - m| > 2^{p_2+1}\epsilon$, then $f(h) \cdot n - m$ and $\mu - m$ have the same sign. The first two implications from (5) then translate into the corresponding cases from (9).

If finally $|\mu - m| \leq 2^{p_2+1}\epsilon$, then the triangle inequality yields $|f(h) \cdot n - m| \leq 2^{p_2+2}\epsilon$, which implies $|f(h) - m/n| < 2^{p_2+2}\epsilon/n < 2^{p_2+w}\eta/n \leq \eta$. By definition of η , this cannot happen unless $m/n = f(h)$. This accounts for the last case. \square

For instance, in the case of binary64–decimal64 comparisons, it is more than enough to compute the product $f(h) \cdot n$ in 128-bit arithmetic.

Proposition 4 gives the number of values of h to consider in the table of $f(h)$, which grows quite large (23.5 kB for b64/d64 comparisons, 721 kB in the b128/d128 case, assuming a memory alignment on 8-byte boundaries). However, it could possibly be shared with other algorithms such as binary-decimal conversions. Additionally, h is available early in the algorithm flow, so that the memory latency for the table access can be hidden behind the first step.

The number of elements in the table can be reduced at the price of a few additional operations. Several consecutive h map to a same value $g = \varphi(h)$. The corresponding values of $f(h) = 2^{\varphi(h) \cdot \log_2 5^{-h}}$ are binary shifts of each other. Hence, we may store the most significant bits of $f(h)$ as a function of g , and shift the value read off the table by an appropriate amount to recover $f(h)$. The table size goes down by a factor of about $\log_2(5) \approx 2.3$.

More precisely, define $F(g) = 5^{g-2^{-\psi(g)}}$, where

$$\psi(g) = \lfloor g \cdot \log_2 5 \rfloor. \quad (10)$$

We then have $f(h) = F(\varphi(h)) \cdot 2^{-\rho(h)}$ with $\rho(h) = h - \psi(\varphi(h))$. The computation of ψ and ρ is similar to that of φ in Step 1. In addition, we may check that

$$1 \leq F(\varphi(h)) < 2 \quad \text{and} \quad 0 \leq \rho(h) \leq 3$$

for all h . Since $\rho(h)$ is nonnegative, multiplication by $2^{\rho(h)}$ can be implemented with a bitshift, not requiring branches.

We did not implement this size reduction: instead, we concentrated on another size reduction opportunity that we shall describe now.

5.2 Bipartite Table

That second table size reduction method uses a bipartite table. Additionally, it takes advantage of the fact that, for small nonnegative g , the exact value of 5^g takes less space than an accurate enough approximation of $f(h)$ (for instance, 5^g fits on 64 bits for $g \leq 27$). In the case of a typical implementation of comparison between binary64 and decimal64 numbers, the bipartite table uses only 800 bytes of storage.

Most of the overhead in arithmetic operations for the bipartite table method can be hidden on current processors through increased instruction-level parallelism. The reduction in table size also helps decreasing the probability of cache misses.

Recall that we suppose $g = \varphi(h)$, so that g and h lie between bounds $g_{\min}^{(2)}, g_{\max}^{(2)}$, resp. $h_{\min}^{(2)}, h_{\max}^{(2)}$ deduced from Proposition 4. For some integer "splitting factor" $\gamma > 0$ and $\epsilon = \pm 1$, write

$$g = \varphi(h) = \epsilon(\gamma q - r), \quad q = \left\lceil \frac{\epsilon g}{\gamma} \right\rceil, \quad (11)$$

so that

$$f(h) = \left(\frac{5^{\gamma q}}{5^r} \right)^\epsilon \cdot 2^{-h-w}.$$

Typically, γ will be chosen as a power of 2, but other values can occasionally be useful. With these definitions, we have $0 \leq r \leq \gamma - 1$, and q lies between

$$\left\lceil \frac{\epsilon g_{\min}^{(2)}}{\gamma} \right\rceil \quad \text{and} \quad \left\lceil \frac{\epsilon g_{\max}^{(2)}}{\gamma} \right\rceil$$

(the order of the bounds depends on ϵ). The integer 5^r fits on $\psi(r) + 1$ bits, where ψ is the function defined by (10).

Instead of tabulating $f(h)$ directly, we will use two tables: one containing the most significant bits of $5^{\gamma q}$ roughly to the precision dictated by the worst cases computed in Section 4, and the other containing the exact value 5^r for $0 \leq r \leq \gamma - 1$. We denote by λ_1 and λ_2 the respective precisions (entry sizes in bits) of these tables. Based on these ranges and precisions, we assume

$$\lambda_1 > \log_2(\eta^{-1}) - w + 3, \quad (12)$$

$$\lambda_2 \geq \psi(\gamma - 1) + 1. \quad (13)$$

In addition, it is natural to suppose λ_1 larger than or close to λ_2 : otherwise, an algorithm using two approximate tables will likely be more efficient.

In practice, λ_2 will typically be one of the available machine word widths, while, for reasons that should become clear later, λ_1 will be chosen slightly smaller than a word size. For each pair of basic IEEE formats, Tables 6 and 7 provide values of $\epsilon, \gamma, \lambda_1, \lambda_2$ (and other parameters whose definition follows later) suitable for typical processors. The suggested values were chosen by optimizing either the table size or the number of elementary multiplications in the algorithm, with or without the constraint that γ be a power of two.

It will prove convenient to store the table entries left-aligned, in a fashion similar to floating-point significands. Thus, we set

$$\theta_1(q) = 5^{\gamma q} \cdot 2^{\lambda_1 - 1 - \psi(\gamma q)},$$

$$\theta_2(r) = 5^r \cdot 2^{\lambda_2 - 1 - \psi(r)},$$

where the power-of-two factors provide for the desired alignment. One can check that

$$2^{\lambda_1 - 1} \leq \theta_1(q) < 2^{\lambda_1}, \quad 2^{\lambda_2 - 1} \leq \theta_2(r) < 2^{\lambda_2} \quad (14)$$

for all h .

The value $f(h)$ now decomposes as

$$f(h) = \frac{5^g}{2^{h+w}} = \left(\frac{\theta_1(q)}{\theta_2(r)} \right)^\epsilon 2^{\epsilon \cdot (\lambda_2 - \lambda_1) - \sigma(h) - w} \quad (15)$$

where

$$\sigma(h) = h - \epsilon \cdot (\psi(\gamma q) - \psi(r)). \quad (16)$$

Equations (10) and (16) imply that

$$h - g \log_2 5 - 1 < \sigma(h) < h - g \log_2 5 + 1.$$

As we also have $h \log_5 2 - 1 \leq g = \varphi(h) < h \log_5 2$ by definition of φ , it follows that

$$0 \leq \sigma(h) \leq 3. \quad (17)$$

Now let $\tau \geq -1$ be an integer. In practice, we will usually choose $\tau = 0$, sometimes $\tau = -1$, and $\tau > 0$ for highly unusual floating-point formats only. A reader who is not interested in such corner cases may simply assume $\tau = 0$. Define

$$\begin{cases} \Delta = \theta_1(q) \cdot n \cdot 2^{\tau - p'_{10}} \\ \quad - \theta_2(r) \cdot m \cdot 2^{\tau + \lambda_1 - \lambda_2 + \sigma(h) - p_2 - 1}, & \epsilon = +1, \\ \Delta = \theta_1(q) \cdot m \cdot 2^{\tau - p_2 - 1} \\ \quad - \theta_2(r) \cdot n \cdot 2^{\tau + \lambda_1 - \lambda_2 - \sigma(h) - p'_{10}}, & \epsilon = -1. \end{cases}$$

The relations $|x_2| > |x_{10}|$, $|x_2| = |x_{10}|$, and $|x_2| < |x_{10}|$ hold respectively when $\epsilon \Delta < 0$, $\Delta = 0$, and $\epsilon \Delta > 0$.

Proposition 7. *Unless $x_2 = x_{10}$, we have $|\Delta| > 2^{\tau+1}$.*

Proof: Assume $x_2 \neq x_{10}$. For $\epsilon = +1$, and using (15), the definition of Δ rewrites as

$$\Delta = \theta_2(r) n 2^{\tau + \lambda_1 - \lambda_2 + \sigma(h) - p_2 - 1} \left(f(h) - \frac{m}{n} \right).$$

Since $f(h) = m/n$ is equivalent to $x_2 = x_{10}$, we know by Proposition 5 that $|f(h) - m/n| \geq \eta$. Together with the bounds (3), (12), (14), and (17), this implies

$$\begin{aligned} \log_2 |\Delta| &\geq (\lambda_2 - 1) + (p'_{10} - 1) + \tau + \lambda_1 - \lambda_2 - p_2 - 1 \\ &\quad + \log_2 \eta \\ &= (\lambda_1 + \log_2 \eta + w - 3) + \tau + 1 > \tau + 1. \end{aligned}$$

Similarly, for $\epsilon = -1$, we have

$$\Delta = -\theta_1(q) n 2^{\tau - p_2 - 1} \left(f(h) - \frac{m}{n} \right),$$

so that

$$\begin{aligned} \log_2 |\Delta| &\geq (\lambda_1 - 1) + (p'_{10} - 1) + \tau - p_2 - 1 + \log_2 \eta \\ &> \tau + 1 \end{aligned}$$

when $\Delta \neq 0$. \square

As already explained, the values of $\theta_2(r)$ are integers and can be tabulated exactly. In contrast, only an approximation of $\theta_1(q)$ can be stored. We represent these values as $[\theta_1(q)]$, hence replacing Δ by the easy-to-compute

$$\begin{cases} \tilde{\Delta} = \lceil [\theta_1(q)] \cdot n \cdot 2^{\tau - p'_{10}} \rceil \\ \quad - \lfloor \theta_2(r) \cdot m \cdot 2^{\tau + \lambda_1 - \lambda_2 + \sigma(h) - p_2 - 1} \rfloor, & \epsilon = +1, \\ \tilde{\Delta} = \lceil [\theta_1(q)] \cdot m \cdot 2^{\tau - p_2 - 1} \rceil \\ \quad - \lfloor \theta_2(r) \cdot n \cdot 2^{\tau + \lambda_1 - \lambda_2 - \sigma(h) - p'_{10}} \rfloor, & \epsilon = -1. \end{cases}$$

Proposition 7 is in principle enough to compare x_2 with x_{10} , using a criterion similar to that from Proposition 6. Yet additional properties hold that allow for a more efficient final decision step.

Proposition 8. *Assume $g = \varphi(h)$, and let $\tilde{\Delta}$ be the signed integer defined above. For $\tau \geq 0$, the following equivalences hold:*

$$\begin{aligned}\Delta < 0 &\iff \tilde{\Delta} \leq -2^\tau, \\ \Delta = 0 &\iff 0 \leq \tilde{\Delta} \leq 2^\tau, \\ \Delta > 0 &\iff \tilde{\Delta} \geq 2^{\tau+1}.\end{aligned}$$

Furthermore, if $\tau \in \{-1, 0\}$ and

$$\lambda_1 - \lambda_2 + \tau \geq \begin{cases} p_2 + 1, & \epsilon = +1, \\ p'_{10} + 3, & \epsilon = -1, \end{cases} \quad (18)$$

then Δ and $\tilde{\Delta}$ have the same sign (in particular, $\tilde{\Delta} = 0$ if and only if $\Delta = 0$).

Proof: Write the definition of $\tilde{\Delta}$ as

$$\tilde{\Delta} = \lceil [\theta_1(q)]X \rceil - \lfloor \theta_2(r)X' \rfloor$$

where the expressions of X and X' depend on ϵ , and define $\delta_{\text{tbl}}, \delta_{\text{rnd}}, \delta'_{\text{rnd}}$ by

$$\begin{aligned}\lceil \theta_1(q) \rceil &= \theta_1(q) + \delta_{\text{tbl}}, & \lceil [\theta_1(q)]X \rceil &= \lceil \theta_1(q) \rceil X + \delta_{\text{rnd}}, \\ \lfloor \theta_2(r)X' \rfloor &= \theta_2(r)X' + \delta'_{\text{rnd}}.\end{aligned}$$

The δ 's then satisfy

$$0 \leq \delta_{\text{tbl}} < 1, \quad -1 < \delta_{\text{rnd}}, \delta'_{\text{rnd}} \leq 0.$$

For both possible values of ϵ , we have $1 \cdot 0 \leq X \leq 2^\tau$ and hence

$$-1 < \tilde{\Delta} - \Delta = X\delta_{\text{tbl}} + \delta_{\text{rnd}} - \delta'_{\text{rnd}} < 2^\tau + 1.$$

If $\Delta < 0$, Proposition 7 implies that

$$\tilde{\Delta} < -2^{\tau+1} + 2^\tau + 1 = -2^\tau + 1.$$

It follows that $\tilde{\Delta} \leq -\lceil 2^\tau \rceil$ since $\tilde{\Delta}$ is an integer. By the same reasoning, $\Delta > 0$ implies $\tilde{\Delta} \geq \lceil 2^{\tau+1} \rceil$, and $\Delta = 0$ implies $0 \leq \tilde{\Delta} \leq \lceil 2^\tau \rceil$. This proves the first claim.

Now assume that (18) holds. In this case, we have $\delta'_{\text{rnd}} = 0$, so that $-1 < \tilde{\Delta} - \Delta < 2^\tau$. The upper bounds on $\tilde{\Delta}$ become $\tilde{\Delta} \leq -\lceil 2^\tau \rceil - 1$ for $\Delta < 0$ and $\tilde{\Delta} \leq \lceil 2^\tau \rceil - 1$ for $\Delta = 0$. When $-1 \leq \tau \leq 0$, these estimates respectively imply $\tilde{\Delta} < 0$ and $\tilde{\Delta} = 0$, while $\tilde{\Delta} \geq \lceil 2^{\tau+1} \rceil$ implies $\Delta > 0$. The second claim follows. \square

The conclusion $\tilde{\Delta} = 0 = \Delta$ when $x_2 = x_{10}$ for certain parameter choices means that there is no error in the approximate computation of Δ in these cases. Specifically, the tabulation error δ_{tbl} and the rounding error δ_{rnd} cancel out, thanks to our choice to tabulate the ceiling of $\theta_1(q)$ (and not, say, the floor).

1. Actually $0 \leq X \leq 2^{\tau-1}$ for $\epsilon = -1$: this dissymmetry is related to our choice of always expressing the lower bound on Δ in terms of η rather than using a lower bound on $f(h)^{-1} - n/m$ when $\epsilon = -1$.

We now describe in some detail the algorithm resulting from Proposition 8, in the case $\epsilon = +1$. In particular, we will determine integer datatypes on which all steps can be performed without overflow, and arrange the powers of two in the expression of $\tilde{\Delta}$ in such a way that computing the floor functions comes down to dropping the least significant words of multiple-word integers.

Let $W : \mathbb{N} \rightarrow \mathbb{N}$ denote a function satisfying $W(k) \geq k$ for all k . In practice, $W(k)$ will typically be the width of the smallest ‘‘machine word’’ that holds at least k bits. We further require that (for $\epsilon = +1$):

$$W(p'_{10}) - p'_{10} \geq 1, \quad (19)$$

$$W(\lambda_1) - \lambda_1 \geq W(\lambda_2) - \lambda_2 + \tau + 3, \quad (20)$$

$$W(p_2) - p_2 + W(\lambda_2) - \lambda_2 \geq W(\lambda_1) - \lambda_1 - \tau + 1. \quad (21)$$

These assumptions are all very mild in view of the IEEE-754-2008 Standard. Indeed, for all IEEE interchange formats (including binary16 and formats wider than 64 bits), the space taken up by the exponent and sign leaves us with at least 5 bits of headroom if we choose for $W(p_2)$, resp. $W(p'_{10})$ the word width of the format. Even if we force $W(\lambda_2) = \lambda_2$ and $\tau = 0$, this always allows for a choice of λ_1, λ_2 and $W(\lambda_2)$ satisfying (12), (13) and (19)–(21).

Denote $x^+ = \max(x, 0)$ and $x^- = \max(-x, 0)$, so that $x = x^+ - x^-$.

Algorithm 2. Step 2, second method, $\epsilon = +1$.

- 1 Compute q and r as defined in (11), with $\epsilon = +1$. Compute $\sigma(h)$ using Proposition 2.
- 2 Read $\lceil \theta_1(q) \rceil$ from the table, storing it on a $W(\lambda_1)$ -bit (multiple-)word.
- 3 Compute

$$A = \lceil (\lceil \theta_1(q) \rceil \cdot 2^{\tau^+}) \cdot (n2^{W(p'_{10})-p'_{10}-\tau^-}) \cdot 2^{-W(p'_{10})} \rceil$$

by (constant) left shifts followed by a $W(\lambda_1)$ -by- $W(p'_{10})$ -bit multiplication, dropping the least significant word(s) of the result that correspond to $W(p'_{10})$ bits.

- 4 Compute

$$B = \lfloor \theta_2(r) \cdot (m2^{\sigma(h)+\omega}) \cdot 2^{W(\lambda_1)-W(\lambda_2)-W(p_2)} \rfloor$$

where the constant ω is given by

$$\omega = W(p_2) - p_2 + W(\lambda_2) - \lambda_2 - W(\lambda_1) + \lambda_1 + \tau - 1$$

in a similar way, using a $W(\lambda_2)$ -by- $W(p_2)$ -bit multiplication.

- 5 Compute the difference $\tilde{\Delta} = A - B$ as a $W(\lambda_1)$ -bit signed integer.
- 6 Compute $\Delta' = \lfloor \tilde{\Delta} 2^{-\tau-1} \rfloor 2^{\tau+1}$ by masking the $\tau + 1$ least significant bits of $\tilde{\Delta}$.
- 7 Return

$$\begin{cases} \text{‘‘}x_{10} < x_2\text{’’} & \text{if } \Delta' < 0, \\ \text{‘‘}x_{10} = x_2\text{’’} & \text{if } \Delta' = 0, \\ \text{‘‘}x_{10} > x_2\text{’’} & \text{if } \Delta' > 0. \end{cases}$$

Proposition 9. When either $\tau \geq 0$ or $\tau = -1$ and (18) holds, Algorithm 2 correctly decides the ordering between x_2 and x_{10} .

Proof: Hypotheses (19) and (21), combined with the inequalities $\sigma(h) \geq 0$ and $\tau \geq -1$, imply that the shifts in Steps 3 and 4 are left shifts. They can be performed without overflow on unsigned words of respective widths $W(\lambda_1)$, $W(p'_{10})$ and $W(p_2)$ since $W(\lambda_1) - \lambda_1 \geq \tau + 1$ and $W(p_2) - p_2 \geq \omega + 3$, both by (20). The same inequality implies that the (positive) integers A and B both fit on $W(\lambda_1) - 1$ bits, so that their difference can be computed by a subtraction on $W(\lambda_1)$ bits. The quantity $A - B$ computed in Step 5 agrees with the previous definition of $\tilde{\Delta}$, and Δ' has the same sign as $\tilde{\Delta}$, hence Proposition 8 implies that the relation returned in Step 7 is correct. \square

When $\tau \in \{-1, 0\}$ and (18) holds, no low-order bits are dropped in Step 4, and Step 6 can be omitted (that is, replaced by $\Delta' := \tilde{\Delta}$). Also observe that a version of Step 7 returning $-1, 0$ or 1 according to the comparison result can be implemented without branching, using bitwise operations on the individual words which make up the representation of Δ' in two's complement encoding.

The algorithm for $\epsilon = -1$ is completely similar, except that (19)–(21) become

$$W(p_2) - p_2 \geq 2, \quad (19')$$

$$W(\lambda_1) - \lambda_1 \geq W(\lambda_2) - \lambda_2 + \tau + 1, \quad (20')$$

$$W(p'_{10}) - p'_{10} + W(\lambda_2) - \lambda_2 \geq W(\lambda_1) - \lambda_1 - \tau + 3, \quad (21')$$

and A and B are computed as

$$A = \lfloor ([\theta_1(q)] \cdot 2^{\tau^+}) \cdot (m2^{W(p_2)-p_2-\tau^- - 1}) \cdot 2^{-W(p_2)} \rfloor$$

$$B = \lfloor [\theta_2(r) \cdot (n2^{\omega-\sigma(h)}) \cdot 2^{W(\lambda_1)-W(\lambda_2)-W(p'_{10})}] \rfloor,$$

with

$$\omega = W(p'_{10}) - p'_{10} + W(\lambda_2) - \lambda_2 - W(\lambda_1) + \lambda_1 + \tau,$$

respectively using a $W(\lambda_1)$ -by- $W(p_2)$ -bit and a $W(\lambda_2)$ -by- $W(p'_{10})$ multiplication.

Tables 6 and 7 suggest parameter choices for comparisons in basic IEEE formats. (We only claim that these are reasonable choices that satisfy all conditions, not that they are optimal for any well-defined metric.) Observe that, though it has the same overall structure, the algorithm presented in [2] is not strictly speaking a special case of Algorithm 2, as the definition of the bipartite table (cf. (11)) is slightly different.

6 AN EQUALITY TEST

Besides deciding the two possible inequalities, the direct and bipartite methods described in the previous sections are able to determine cases when x_2 is equal to x_{10} . However, when it comes to solely determine such equality cases additional properties lead to a simpler and faster algorithm.

The condition $x_2 = x_{10}$ is equivalent to

$$m \cdot 2^{h+w} = n \cdot 5^g, \quad (22)$$

and thus implies certain divisibility relations between m , n , $5^{\pm g}$ and $2^{\pm h \pm w}$. We now describe an algorithm that takes advantage of these relations to decide (22) using only binary shifts and multiplications on no more than $\max(p'_{10} + 2, p_2 + 1)$ bits.

Proposition 10. Assume $x_2 = x_{10}$. Then, we have

$$g_{\min}^{\text{eq}} \leq g \leq g_{\max}^{\text{eq}}, \quad -p'_{10} + 1 \leq h \leq p_2.$$

where

$$g_{\min}^{\text{eq}} = -\lfloor p_{10}(1 + \log_5 2) \rfloor, \quad g_{\max}^{\text{eq}} = \lfloor p_2 \log_5 2 \rfloor.$$

Additionally,

- if $g \geq 0$, then 5^g divides m , otherwise 5^{-g} divides M_{10} (and n);
- if $h \geq -w$, then 2^{h+w} divides n , otherwise $2^{-(h+w)}$ divides m .

Proof: First observe that g must be equal to $\varphi(h)$ by Proposition 1, so $g \geq 0$ if and only if $h \geq 0$. The divisibility properties follow immediately from the coprimality of 2 and 5.

When $g \geq 0$, they imply $5^g \leq m < 2^{p_2} - 1$, whence $g < p_2 \log_5 2$. Additionally, $2^{-h-w}n$ is an integer. Since $h \geq 0$, it follows that $2^h \leq 2^{-w}n < 2^{p'_{10}-w}$ and hence $h \leq p'_{10} - w - 1 = p_2$.

If now $g < 0$, then 5^{-g} divides $m = 2^\nu M_{10}$ and hence divides M_{10} , while 2^{-h} divides $2^w m$. Since $M_{10} < 10^{p_{10}}$ and $2^w m < 2^{p_2+w}$, we have $-g < p_{10} \log_5 10$ and $-h < p_{10} \log_2 10 < p'_{10}$. \square

These properties translate into Algorithm 3 below. Recall that $x^+ = \max(x, 0)$ and $x^- = x^+ - x$. (Branchless algorithms exist to compute x^+ and x^- [1].) Let $p = \max(p'_{10} + 2, p_2 + 1)$.

Algorithm 3. Equality test.

- 1 If not $g_{\min}^{\text{eq}} \leq g \leq g_{\max}^{\text{eq}}$ then return false.
- 2 Set $u = (w + h)^-$ and $v = (w + h)^+$.
- 3 Using right shifts, compute

$$m_1 = \lfloor 2^{-u} m \rfloor, \quad n_1 = \lfloor 2^{-v} n \rfloor.$$

- 4 Using left shifts, compute $m_2 = 2^u m_1$ and $n_2 = 2^v n_1$.
- 5 Read 5^{g^+} and 5^{g^-} from a table (or compute them).
- 6 Compute $m_3 = 5^{g^-} m_1$ with a $p'_{10} \times p_2 \rightarrow p$ bit multiplication (meaning that the result can be anything if the product does not fit on p bits).
- 7 Compute $n_3 = 5^{g^+} n_1$ with a $p_2 \times p'_{10} \rightarrow p$ bit multiplication.
- 8 Return $(m_2 = m) \wedge (n_2 = n) \wedge (g = \varphi(h)) \wedge (n_3 = m_3)$.

As a matter of course, the algorithm can return false as soon as any of the conditions from Step 8 is known unsatisfied.

Proposition 11. Algorithm 3 correctly decides whether $x_2 = x_{10}$.

Proof: First observe that, if any of the conditions $g = \varphi(h)$, $g_{\min}^{\text{eq}} \leq g \leq g_{\max}^{\text{eq}}$, $m_2 = m$ and $n_2 = n$ is violated, then $x_2 \neq x_{10}$ by Propositions 1 and 10. Indeed,

$x_2 = x_{10}$ implies $m_2 = m$ and $n_2 = n$ due to the divisibility conditions of Proposition 10. In all these cases, the algorithm correctly returns false. In particular, no table access occurs unless g lies between g_{\min}^{eq} and g_{\max}^{eq} .

Now assume that these four conditions hold. In particular, we have $m_1 = 2^{-u}m$ as $m_2 = m$, and similarly $n_1 = 2^{-v}n$. The bounds on g imply that 5^{g^+} fits on p_2 bits and 5^{g^-} fits on p'_{10} bits. If g and $w + h$ are both nonnegative, then $m_3 = m$, and we have

$$5^{g^+} n_1 = 5^{\varphi(h)} 2^{-(w+h)} n < 2^{-w+p'_{10}} = 2^{p_2+1} \leq 2^p.$$

By the same reasoning, if $g, w + h \leq 0$, then $n_3 = n$ and

$$5^{-g} 2^{w+h} m < 5 \cdot 2^{p'_{10}-1} < 2^p.$$

If now $g \geq 0$ with $h \leq -w$, then $m_3 = m_1$ and

$$5^g n < 2^{h+p'_{10}} \leq 2^{-w+p'_{10}} = 2^{p_2+1} \leq 2^p.$$

Similarly, for $g \leq 0$, $-h \leq w$, we have

$$5^{-g} m < 5 \cdot 2^{-h+p_2} \leq 5 \cdot 2^{w+p_2} \leq 2^p.$$

In all four cases,

$$m_3 = 5^{g^-} 2^{-u} m \quad \text{and} \quad n_3 = 5^{g^+} 2^{-v} n$$

are computed without overflow. Therefore

$$\frac{m_3}{n_3} = \frac{m \cdot 2^{(h+w)^+ - (h+w)^-}}{n \cdot 5^{g^+ - g^-}} = \frac{m \cdot 2^{h+w}}{n \cdot 5^g}$$

and $x_2 = x_{10}$ if and only if $m_3 = n_3$, by (22). \square

7 EXPERIMENTAL RESULTS

Both the method based on direct tabulation of $f(h)$ presented in Section 5.1 and the bipartite table method presented in Section 5.2 have been implemented and thoroughly tested in the case of binary64–decimal64 comparisons. Our implementation is available on this article’s page at

<http://hal.archives-ouvertes.fr/>

along with tables of worst and bad cases for the approximation problem from Section 4.

To the extent possible when using decimal floating-point numbers, we gave priority to portability over performance in the main source code. It is written in plain C, using the decimal floating-point type support offered by the compiler we are using, `gcc` 4.9.0 [10]. No particular effort at optimization was made. For instance, multiple-word operations are implemented in portable C, with no use of hardware operations such as $64 \times 64 \rightarrow 128$ bit multiplication except for that automatically inserted by the compiler. Nevertheless, we also provide an alternative implementation of the bipartite table method featuring some manual optimization. This optimized version is written using `gcc` extensions and compiler intrinsics besides plain C.

Testing was done using test vectors that extensively cover all floating-point input classes, such as normal

numbers, subnormals, Not-A-Numbers, zeros, infinities, for both the binary input x_2 and the decimal input x_{10} . The test vectors also include worst cases (i.e., inputs reaching the minimal value of $d_h(m, n)$) for each admissible value of h , as well as many next to worst cases computed using the method described in the Appendix of [3]. The rationale behind exercising not only the worst-case for each h but also other bad cases is that an implementation might work for the worst-case input just by chance, whereas chances decrease rapidly for a larger number of difficult test cases. The test vectors were finally completed with a large set of random inputs that fully exercise all possible values of the normalization parameter ν (cf. Section 3) as well as both the first step succeeding or the second step being necessary.

Both implemented methods have been compared for performance to the naïve comparison method where one of the inputs is converted to the radix of the other input. These experimental results are reported in Table 5. In the last two rows, “easy” cases are cases where the first step of our algorithm succeeds, while “hard” cases refer to those for which running the second step is necessary.

The code is executed on a system equipped with a quad-core Intel Core i5-3320M processor clocked at 2.6 GHz, running Linux 3.14 in 64 bit mode. The comparison functions are compiled at optimization level 3, with the `-march=native -msse4` flags set. The timing measurements were done using the Read-Time-Step-Counter instruction after pipeline serialization. The serialization and function call overhead was subtracted off after timing an empty function. Measurements were taken once the caches were preheated by previous comparison function calls, the results of which were discarded. The indicated numbers are in cycles and given for the minimum, maximum and (except in cases involving special inputs) average value that was observed. Cycle counts larger than about 370 were discarded, and the corresponding tests run again, in order not to account for delays likely due to exceptional external events.

As can be seen from Table 5, our implementations of both the Direct Method presented in Section 5.1 and the Bipartite Table Method described in Section 5.2 outperform both naïve methods in most cases.

The (unoptimized) Direct Method is slightly faster than the (unoptimized) Bipartite Table Method. This is mainly due to the more computationally expensive second step of the Bipartite Table Method. The difference in performance probably does not justify the use of an about 30 times larger table for the direct method. We do not have experimental data for the Direct Method using the table size reduction technique through tabulation of $F(g)$ (cf. Section 5.1).

8 CONCLUSION

Though not foreseen in the IEEE 754-2008 Standard, exact comparisons between floating-point formats of different radices would enrich the current floating-point

	Naïve method converting x_{10} to binary64 (<i>incorrect</i>) min/avg/max	Naïve method converting x_2 to decimal64 (<i>incorrect</i>) min/avg/max	Direct method described in Section 5.1 min/avg/max	Bipartite table method described in Section 5.2 min/avg/max	Bipartite table method (optimized implementation) min/avg/max
Special cases (± 0 , NaNs, Inf)	22/-/150	36/-/193	15/-/94	30/-/116	21/-/81
x_2 , x_{10} of opposite sign	41/107/186	60/144/196	17/37/82	33/55/105	27/38/83
x_2 normal, same sign, “easy” cases	34/107/186	65/150/197	30/64/128	46/83/151	28/50/104
x_2 subnormal, same sign, “easy” cases	42/106/172	108/142/193	158/169/195	172/183/211	30/35/80
x_2 normal, same sign, “hard” cases	54/84/176	54/100/196	45/63/143	67/86/192	36/48/126
x_2 subnormal, same sign, “hard” cases	73/105/173	96/125/196	171/183/195	196/208/213	43/57/102

TABLE 5
Timings (in cycles) for both presented methods and for two naïve methods.

environment and enhance the safety and provability of numerical software.

This paper has investigated the feasibility of such comparisons. A simple test has been presented that eliminates most of the comparison inputs. For the remaining cases, two algorithms were proposed, a more direct method and a technique based on a bipartite table. For instance, in the case of binary64–decimal64, the bipartite table uses only 800 bytes of table space. It is still feasible in the case of binary128–decimal128 comparisons.

Both methods have been proven, implemented (in the case of binary64–decimal64 comparisons) and thoroughly tested. They outperform the naïve comparison technique consisting in conversion of one of the inputs to the respectively other format. Furthermore, they always return a correct answer, which is not the case of the naïve technique.

Since the algorithmic problems of exact binary to decimal comparison and correct rounding of binary to decimal conversions (and vice versa) are similar, future investigations should also consider the possible reuse of tables for both problems. Finally, one should mention that considerable additional work is required in order to enable mixed-radix comparisons in the case when the decimal floating-point number is stored in dense-packed-decimal representation.

REFERENCES

- [1] Jörg Arndt, *Matters computational. ideas, algorithms, source code*, Springer, 2011.
- [2] Nicolas Brisebarre, Christoph Lauter, Marc Mezzarobba, and Jean-Michel Muller, *Comparison between binary64 and decimal64 floating-point numbers*, ARITH 21 (Austin, Texas, USA) (Alberto Nannarelli, Peter-Michael Seidel, and Ping Tak Peter Tang, eds.), IEEE Computer Society, 2013, pp. 145–152.
- [3] M. Cornea, J. Harrison, C. Anderson, P. T. P. Tang, E. Schneider, and E. Gvozdev, *A software implementation of the IEEE 754R decimal floating-point arithmetic using the binary encoding format*, IEEE Transactions on Computers **58** (2009), no. 2, 148–162.
- [4] G. H. Hardy and E. M. Wright, *An introduction to the theory of numbers*, Oxford University Press, London, 1979.
- [5] IEEE Computer Society, *IEEE standard for floating-point arithmetic*, IEEE Standard 754-2008, August 2008, available at <http://ieeexplore.ieee.org/servlet/opac?punumber=4610933>.
- [6] ISO/IEC JTC 1/SC 22/WG 14, *Extension for the programming language C to support decimal floating-point arithmetic*, Proposed Draft Technical Report, May 2008.
- [7] A. Ya. Khinchin, *Continued fractions*, Dover, New York, 1997.
- [8] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of floating-point arithmetic*, Birkhäuser, Boston, 2010.
- [9] O. Perron, *Die Lehre von den Kettenbrüchen*, 3rd ed., Teubner, Stuttgart, 1954–57.
- [10] The GNU project, *GNU compiler collection*, 1987–2014, available at <http://gcc.gnu.org/>.

	b32/d64	b32/d128	b64/d64	b64/d128	b128/d64	b128/d128
$W(p_2)$	32	32	64	64	128	128
$W(p'_{10})$	64	128	64	128	64	128
$\gamma = 2^k$, optimizing multiplication count						
ϵ	-1	-1	+1	-1	+1	+1
γ	8	8	8	8	8	8
range of q	-4, 8	-4, 10	-42, 39	-38, 45	-622, 617	-624, 617
min λ_1 satisfying (12)	86	145	117	178	190	242
min λ_2 satisfying (13)	17	17	17	17	17	17
chosen $\lambda_1, W(\lambda_1)$	95, 96	159, 160	125, 128	191, 192	190, 192	253, 256
chosen $\lambda_2, W(\lambda_2)$	32, 32	32, 32	32, 32	32, 32	32, 32	32, 32
τ	0	0	0	0	-1	0
Step 6 of Algo. 2 can be omitted	!	!	!	!	!	!
total table size in bytes	188	332	1344	2048	29792	39776
32-bit muls	5	9	10	16	16	36
$\gamma = 2^k$, optimizing table size						
ϵ	-1	-1	+1	-1	-1	+1
γ	8	16	32	32	64	64
range of q	-4, 8	-2, 5	-10, 10	-9, 12	-77, 78	-78, 78
min λ_1 satisfying (12)	86	145	117	178	190	242
min λ_2 satisfying (13)	17	35	72	72	147	147
chosen $\lambda_1, W(\lambda_1)$	95, 96	159, 160	125, 128	191, 192	191, 192	253, 256
chosen $\lambda_2, W(\lambda_2)$	32, 32	64, 64	96, 96	96, 96	160, 160	160, 160
τ	0	0	0	0	0	0
Step 6 of Algo. 2 can be omitted	!	%	%	%	%	%
total table size in bytes	188	288	720	912	5024	6304
32-bit muls	5	13	14	24	34	52
any γ , optimizing multiplication count						
ϵ	-1	-1	+1	-1	+1	+1
γ	13	13	14	14	14	14
range of q	-2, 5	-2, 6	-24, 22	-22, 26	-355, 353	-357, 353
min λ_1 satisfying (12)	86	145	117	178	190	242
min λ_2 satisfying (13)	28	28	31	31	31	31
chosen $\lambda_1, W(\lambda_1)$	95, 96	159, 160	125, 128	191, 192	190, 192	253, 256
chosen $\lambda_2, W(\lambda_2)$	32, 32	32, 32	32, 32	32, 32	32, 32	32, 32
τ	0	0	0	0	-1	0
Step 6 of Algo. 2 can be omitted	!	!	!	!	!	!
total table size in bytes	148	232	808	1232	17072	22808
32-bit muls	5	9	10	16	16	36
any γ , optimizing table size						
ϵ	-1	-1	+1	+1	-1	+1
γ	13	13	28	28	69	83
range of q	-2, 5	-2, 6	-12, 11	-12, 11	-71, 73	-60, 60
min λ_1 satisfying (12)	86	145	117	178	190	242
min λ_2 satisfying (13)	28	28	63	63	158	191
chosen $\lambda_1, W(\lambda_1)$	95, 96	159, 160	125, 128	189, 192	191, 192	253, 256
chosen $\lambda_2, W(\lambda_2)$	32, 32	32, 32	64, 64	64, 64	160, 160	192, 192
τ	0	0	0	0	0	0
Step 6 of Algo. 2 can be omitted	!	!	!	!	%	%
total table size in bytes	148	232	608	800	4860	5864
32-bit muls	5	9	12	28	34	56

TABLE 6
Suggested parameters for Algorithm 2 on a 32-bit machine.

	b32/d64	b32/d128	b64/d64	b64/d128	b128/d64	b128/d128
$W(p_2)$	64	64	64	64	128	128
$W(p'_{10})$	64	128	64	128	64	128
$\gamma = 2^k$, optimizing multiplication count						
ϵ	+1	-1	+1	-1	+1	+1
γ	16	16	16	16	16	16
range of q	-3, 3	-2, 5	-21, 20	-19, 23	-311, 309	-312, 309
min λ_1 satisfying (12)	86	145	117	178	190	242
min λ_2 satisfying (13)	35	35	35	35	35	35
chosen $\lambda_1, W(\lambda_1)$	125, 128	191, 192	125, 128	191, 192	190, 192	253, 256
chosen $\lambda_2, W(\lambda_2)$	64, 64	64, 64	64, 64	64, 64	64, 64	64, 64
τ	0	0	0	0	-1	0
Step 6 of Algo. 2 can be omitted	!	!	!	!	!	!
total table size in bytes	240	320	800	1160	15032	20032
64-bit mults	3	5	3	5	5	10
$\gamma = 2^k$, optimizing table size						
ϵ	+1	-1	+1	-1	-1	+1
γ	16	16	16	32	64	64
range of q	-3, 3	-2, 5	-21, 20	-9, 12	-77, 78	-78, 78
min λ_1 satisfying (12)	86	145	117	178	190	242
min λ_2 satisfying (13)	35	35	35	72	147	147
chosen $\lambda_1, W(\lambda_1)$	125, 128	191, 192	125, 128	191, 192	191, 192	253, 256
chosen $\lambda_2, W(\lambda_2)$	64, 64	64, 64	64, 64	128, 128	192, 192	192, 192
τ	0	0	0	0	0	0
Step 6 of Algo. 2 can be omitted	!	!	!	%	%	%
total table size in bytes	240	320	800	1040	5280	6560
64-bit mults	3	5	3	7	9	14
any γ , optimizing multiplication count						
ϵ	+1	-1	+1	-1	+1	+1
γ	13	20	28	28	28	28
range of q	-4, 3	-1, 4	-12, 11	-11, 13	-177, 177	-178, 177
min λ_1 satisfying (12)	86	145	117	178	190	242
min λ_2 satisfying (13)	28	45	63	63	63	63
chosen $\lambda_1, W(\lambda_1)$	125, 128	191, 192	125, 128	191, 192	190, 192	253, 256
chosen $\lambda_2, W(\lambda_2)$	64, 64	64, 64	64, 64	64, 64	64, 64	64, 64
τ	0	0	0	0	-1	0
Step 6 of Algo. 2 can be omitted	!	!	!	!	!	!
total table size in bytes	232	304	608	824	8744	11616
64-bit mults	3	5	3	5	5	10
any γ , optimizing table size						
ϵ	+1	-1	+1	+1	-1	+1
γ	13	20	28	28	82	83
range of q	-4, 3	-1, 4	-12, 11	-12, 11	-60, 61	-60, 60
min λ_1 satisfying (12)	86	145	117	178	190	242
min λ_2 satisfying (13)	28	45	63	63	189	191
chosen $\lambda_1, W(\lambda_1)$	125, 128	191, 192	125, 128	189, 192	191, 192	253, 256
chosen $\lambda_2, W(\lambda_2)$	64, 64	64, 64	64, 64	64, 64	192, 192	192, 192
τ	0	0	0	0	0	0
Step 6 of Algo. 2 can be omitted	!	!	!	!	%	%
total table size in bytes	232	304	608	800	4896	5864
64-bit mults	3	5	3	7	9	14

TABLE 7
Suggested parameters for Algorithm 2 on a 64-bit machine.