RC23203 (W0405-038) May 7, 2004
Mathematics

IBM Research Report

On Intermediate Precision Required for Correctly-Rounding Decimal-to-Binary Floating-Point Conversion

Michel Hack
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

On Intermediate Precision Required for Correctly-Rounding Decimal-to-Binary Floating-Point Conversion.

Michel Hack

IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598, USA

Abstract

The algorithms developed ten years ago in preparation for IBM's support of IEEE Floating-Point on its mainframe S/390 processors use an overly conservative intermediate precision to guarantee correctly-rounded results across the entire exponent range. Here we study the minimal requirement for both bounded and unbounded precision on the decimal side (converting to machine precision on the binary side). An interesting new theorem on Continued Fraction expansions is offered, as well as an open problem on the growth of partial quotients for ratios of powers of two and five.

Key words: Floating-Point conversion, Continued Fractions

1 Introduction

Floating-Point conversion involves transforming the product of a fixed-point number and a power of one base (e.g. 10) into the product of another fixed-point number and a power of a different base (e.g. 2), preserving the value of this product as much as possible, subject to constraints on the result format — in particular, the precision of the resulting fixed-point number.

A correctly-rounding conversion produces the one and only result that satisfies a given rounding rule. For example, for IEEE round-to-nearest, it must produce *the* nearest representable result (in the target precision) when there is only one, or the one with a low-order bit of 0 when the infinitely-precise result is an exact midpoint between two representable numbers. The difficulty

Email address: hack@watson.ibm.com (Michel Hack).

of correct rounding is due to the fact that the infinitely-precise result can be very close to the rounding threshold. We call such numbers *difficult* numbers. Continued Fraction expansions can be used to bound how close this can be without being exactly equal to the threshold.

The arguments presented in this paper apply to binary-to-decimal conversion as well as to decimal-to-binary conversion. The former is conceptually a trifle easier, since every binary floating-point number has an exact decimal representation (with a finite though perhaps large number of digits); this is due to the fact that 2 is a factor of 10. Here we shall focus on decimal-to-binary conversion.

Two cases arise in practice, depending on whether input precision is bounded or not. Some programming languages or environments accept only as many input digits as correspond to the target machine precision; others (e.g. Java, IBM's assemblers and run-time conversion routines on z/Series) accept (and honour) as many digits as are presented. One very important class of bounded-precision conversions is the class of machine-format-to-machine-format conversions resulting from the upcoming Revised IEEE 754 standard for Floating-Point that defines both binary and decimal floating-point formats.

2 Prior Work

This paper is a follow-up to [1] which includes a section on Floating-Point conversion, which gives the historical background.

Briefly, there were several unpublished efforts dating back to 1977, as well as David Matula's seminal paper [8] on the precision required for reversible conversions in 1968. Jerry Coonen's Thesis [3] led to the IEEE 754 requirements for extended formats, so correctly-rounding conversion could be performed economically with compatible precision in the middle of the exponent range, and with tight error bounds across the entire exponent range, which is all the 1985 IEEE standard requires. (The IEEE 754 standard is undergoing revision right now (Spring 2004), and a correct-rounding requirement across the entire exponent range is under consideration.)

There were two papers at ACM SIGPLAN 1990 ([2] and [10]): the methods therein use conventional conversion methods but track the conversion error, and redo the conversion using a different, full-precision, method when the rounding threshold is dangerously nearby. Also from that period is Gordon Slishman's method [9] which exploits the limited exponent range of IBM Hexadecimal Floating-Point for a table-driven approach, which picks carefully chosen multipliers so as to avoid dangerous rounding thresholds.

Much later, Kenton Hanson [4] described a method for finding the most *difficult* numbers for each format, under the assumption of compatible decimal precision, by a suitably limited search. The method resembles Continued Fraction expansion, but this is not mentioned.

All methods described above deal with a small set of fixed binary formats, and compatible bounded decimal precision.

The conversion method in [1] is generic in that it (a) can handle any combination of significand precision and exponent range, and (b) imposes no constraints at all on the decimal format (unlimited precision and exponent range, subject only to storage constraints). It uses integer bignum arithmetic, where numbers are represented as arrays of machine-word (32-bit) digits in a large base, 10^9 for *bigdecimal* or 2^{32} for *bigbinary*, as appropriate: when division by 5 is involved, bigdecimal is used, so as to avoid nonterminating fractions. For a given target precision and exponent range there is a maximum-length intermediate bignum, and the workspace requirement reflects this. The actual computation is carried out using the larger precision of source [1] and target, plus one or two guard bigdigits. Truncating arithmetic is used, and a threshold is computed for the truncation error. When the result is closer to the rounding threshold than this error threshold, the conversion is repeated with full precision; this would happen only for *difficult* numbers. The article mentions that this is too conservative, and hints at the solution described in this paper.

3 Continued Fractions

Continued Fraction theory is the theory of choice when it comes to studying close rational approximations to a real number. Rational approximations to a rational number are useful when the denominator in the approximation has many fewer digits than the denominator in the original (reduced) fraction, and yet the difference between the approximation and the original is very small. The hallmark of a Continued Fraction approximation is that it is always a *best* approximation, in the sense that any rational approximation with a smaller denominator necessarily leads to a larger approximation error.

Continued Fraction theory is covered in many books on Number Theory, e.g. the classic Hardy&Wright [5], but one of the most readable and complete expositions is Khinchin [6]. The basic idea is to get a sequence of successfully better rational approximations to a given real number (say positive, for the sake of easier exposition). The crudest approximation is the integral part; the

[1] up to the maximum needed for exact decimal representation, given the effective exponent

left-over (if any) is the fractional part. The inverse of this left-over is therefore greater than one: it has an integral part and a (new) fractional part. This process can be repeated until there is no left-over (which will happen if, and only if, the original value is rational). The sequence of integral parts obtained in this way is the sequence of *partial quotients* of the expansion. The sequence of progressively better approximations (where the left-over is ignored) is called the sequence of *partial convergents*. The term "Continued Fraction" comes from the appearance of the expression of the original value x in terms of its partial quotients a_i:

$$x = a_0 + 1/(a_1 + 1/(a_2 + ...))$$

Because of the sequence of inversions, the approximations alternately underestimate and overestimate the original value. There is a simple recurrence relation that ties a partial convergent to the two preceding partial convergents and the partial quotient of the same index i. The $i$eth partial convergent is P_i/Q_i in lowest terms, and the recurrence relation only generates further ratios in lowest terms:

$$\begin{aligned} P_i &= a_i P_{i-1} + P_{i-2} \\ Q_i &= a_i Q_{i-1} + Q_{i-2} \end{aligned}$$

The recurrence is started with dummy values:

$$\begin{aligned} P_{-1} &= 1 \quad P_{-2} = 0 \\ Q_{-1} &= 0 \quad Q_{-2} = 1 \end{aligned}$$

so that $P_0/Q_0 = a_0$ is the zeroeth approximation, the integral part of x. All partial quotients except possibly a_0 are positive integers.

An interesting property of immediately adjacent partial convergents (which in fact implies that the recurrence always generates partial convergents in lowest terms) is:

$$P_{i+1}Q_i - Q_{i+1}P_i = (-1)^i$$

Because of the alternating approximations, we get the following picture of three successive partial convergents and the original value (or its mirror image, depending on the parity of the index i):

$$\frac{P_i}{Q_i} < \frac{P_{i+2}}{Q_{i+2}} \leq x < \frac{P_{i+1}}{Q_{i+1}}$$

(We assume that x has at least three successive approximations; the last one might be exact.) From this we see that the difference between two successive partial convergents (on opposite sides of x) provides an upper bound on the approximation error at that point, and that the difference between two partial convergents on the same side of x provides a lower bound on the approximation error due to the earlier partial convergent. If we work this out, we get:

$$\frac{1}{Q_i(Q_i + Q_{i+1})} < |x - \frac{P_i}{Q_i}| \leq \frac{1}{Q_i Q_{i+1}}$$

Taking into account that $Q_{i-1} < Q_i$ and using the recurrence relation for Q_{i+1} we get:

$$\frac{1}{(a_{i+1} + 2)Q_i^2} < |x - \frac{P_i}{Q_i}| < \frac{1}{a_{i+1}Q_i^2}$$

This is the bounding formula that expresses lower and upper bounds on the approximation error of partial convergent P_i/Q_i in terms of the *next* partial quotient a_{i+1}. It is interesting to note that although all accounts of Continued Fractions mention the upper bound, very few ([6] being one of them) mention the lower bound. The lower bound is the one of interest in this paper.

Another useful fact (see [5] or [6]) is that if $|x - P/Q| < 1/(2Q^2)$, then P/Q is some Partial Convergent of the Continued Fraction expansion of x. In other words, rational approximations that are *quadratically very close* (in terms of denominator size) are always Partial Convergents.

4 Decimal-to-Binary conversion with fixed source and target precision

A binary floating-point number (BFP) in a certain format has a precision of p bits and an exponent range from $Emin$ to $Emax$; it can express finite numbers from $2^{(Emin+1-p)}$ to $(2^p - 1)2^{(Emax+1-p)}$. The usual representation is as a fixed-point number (with one bit before the radix point) times two to the power of an exponent in the given range; for our purposes it is easier to express it as an integer significand B in the range 2^{p-1} to $2^p - 1$ times a power of two in an appropriately shifted range. We shall ignore denormal (recently renamed *subnormal*) numbers here — they complicate details of rounding and underflow handling, but don't substantially affect issues of required precision, except to the extent that the applicable exponent range has to include them. (Later we will justify this assumption.)

A decimal floating-point number (DFP) similarly consists of a q-digit *significand* and a decimal exponent whose range may exceed that of the target binary format; in free-form decimal string input the exponent might even be unbounded; it might also be implied by the position of a given decimal point. Regardless of input format, a crude precomputation can weed out decimal inputs that would lead to obvious binary overflow or underflow, and here we assume that the effective decimal exponent is implicitly bounded by the target binary format. True zero (all given significand digits zero) can also be disposed of at this point, and the sign can be ignored (it has to be remembered for insertion in the target of course, and it affects the details of directed rounding, but it has no effect on intermediate precision requirements.) For the moment, we also assume that the number of decimal digits is bounded a-priori, so that we only have to address q-digit normalised decimal significands D with an appropriate exponent d and a value of

$$D \times 10^d \quad \text{where} \quad 10^{q-1} \leq D < 10^q$$

The nearest representable BFP number in the target format would be

$$B \times 2^b \quad \text{where } 2^{p-1} \leq B < 2^p \qquad \text{(B integral)}$$

$$\text{and:} \quad 10^d D = 2^b(B + 1/2 + F) \quad \text{where} \quad |F| < 1/2 \quad \text{(round-to-nearest)}$$

$$\text{or:} \quad 10^d D = 2^b(B + F) \quad \text{where} \quad |F| < 1/2 \quad \text{(directed rounding)}$$

By considering a $p+1$-bit binary significand C we can handle all rounding modes in a common way. Round-to-nearest will then involve an *odd* integer $C = 2(B + 1/2)$, and directed rounding will involve *even* $C = 2B$, with:

$$10^d D = 2^{b-1}(C + 2F) \quad \text{where} \quad |F| < 1/2$$

Here C is an integer in the range $2^p + 1$ to $2^{p+1} - 1$.

In all rounding modes, we need to know the *sign* of F with *absolute* precision, i.e. we need to know whether it is exactly zero, and if not, whether it is positive or negative without any doubt. Other than that, the actual value of F is not important (its absolute value will be less than $1/2$, by definition). The correctly-rounded B can then be obtained from C and the sign of F. (The precise rules are messy, since the rounded result might have one more or fewer bits than the expected p bits, in which case the exponent b would be adjusted as the result is fitted to its box. This could then trigger belated overor underflow. These complications are not the subject of this paper.)

Conceptually, to compute C (and F) we first compute b from d and from the

size constraints on D and C, i.e. the given source and target precisions. In some algorithms (e.g. [1]) this need not be computed explicitly; it "falls out" of the exponent-reduction process used there, at the same time as $C + 2F$ (when a full-precision retry is performed), or an approximation to $C + 2F$ sufficient to determine the sign of F.

The sign of F is simply the high-order bit of the fraction $2F$. It may seem strange to talk about a sign bit *inside* the bit string that represents the positive fixed-point number $C + 2F$, but it is really just a way of describing $C - 2 + (2 - (-2F))$ when $1 > |2F| \geq 1$, to describe how far the exact value is from a rounding threshold — an even integer C for directed rounding, or an odd one for round-to-nearest.

The computation of $C + 2F$ may introduce an error e if it is carried out with less than "infinite" precision (full precision will do, if we use decimal instead of binary). Let us write this computed result as $U + V$, broken up into an integral part U and fractional part V such that $|V| \leq 1/2$ (it does not matter whether we pick $+1/2$ or $-1/2$ for V if that should be the case): $U + V = C + 2F + e$.

For large $|F| < 1/2$, the computed value $2F + e$ could cause the "sign bit" of $2F + e$ to differ from the sign of F, i.e. we could have $|2F + e| \geq 1$. This would be ok however, because as long as $|e| < 1/2$ the two numbers $C + 2F + e$ and $C + 2F$ would still round to the same final result, though by different paths through the rounding-rule maze: crossing *this* threshold would simultaneously change the sign of V and the parity of U, which should have no effect on the final correctly-rounded result.

Numbers whose conversion results in a large $|F|$ may be called *easy* numbers. Their conversion needs only a few extra bits of precision. An implementation can take advantage of this by choosing an initial error threshold $E < 1/4$, and first computing $U + V = C + 2F + e$ using fast but limited-precision arithmetic such that $|e| < E$. Suppose we have $|V| > E$. If V is positive this leads to: $E < V < E + 2F + (C - U)$, hence $0 < 2F + (C - U)$. Since C and U are integers, and $|2F| < 1$, C and U must be equal, and F must be positive like V. If V is negative we get $E < -V < E - 2F - (C - U)$, hence $0 < -2F - (C - U)$. In either case, C equals U and F has the same sign as V: the limited-precision result delivers the correctly-rounded result. The smaller we choose E, the larger the proportion of *easy* numbers.

For small $|F|$, the computed sign of $F + e$ could differ from the true sign of F, and if this were to happen, the computed $U + V = C + 2F + e$ would be on the wrong side of the rounding threshold. This is why we shall need enough intermediate precision to guarantee that $|e| < |2F|$ (unless the computation is exact). The problem here is crossing *this* threshold changes the sign of V without changing U, which would lead to an incorrectly-rounded result.

We will show that, for a given BFP format, there is a lower bound L on non-zero $|2F|$, i.e. L such that:

$$|2F| < L \leq \frac{1}{2} \Longrightarrow F = 0$$

If we then pick an intermediate precision such that we can guarantee $|e| < L$, and deal with *easy* numbers as described above, we have $|V| < E < 1/4$ and $|(U - C) + (V - 2F)| = |e| < L$, with L defined as above. We also know that $|2F| < E + |e| < 2E$, because otherwise V would have passed the *easy number* threshold. Again, $U - C$ must be zero: $|V - 2F| < L$. If $|V| > L$ this implies that V and F must have the same sign. If $|V| \leq L$ the definition of L implies that F must be zero. In either case, we get a correctly-rounded result.

Until now we have ignored the details of how $C + 2F$ was computed from the given DFP number $D \times 10^d$, with $10^{q-1} \leq D < 10^q$. The exact value in BFP form is $2^{b-p-1}(C + 2F)$ with $2^p \leq C < 2^{p+1}$ and $|2F| < 1$. In fact, having disposed of *easy* numbers already, we can assume $|2F| < 2E$ which is certainly less than $1/2$, and often much smaller than that (we can pick E small enough to ensure that the proportion of numbers requiring more intermediate precision is very low).

Let x be the rational number $10^d/2^{b-1}$ (or $x = 2^{d+1-b}5^d$). Then $C + 2F = xD$, which we rewrite as $x - C/D = 2F/D$ to remind us of the form used to describe Continued Fraction approximations.

The smallest difference occurs when C/D is a partial convergent of the Continued Fraction expansion of x, unless conversion is exact and F is zero. It is of course possible that none of the partial convergents have the right shape: the one with the largest numerator of fewer than $p + 1$ bits may be followed by one with more than $p + 1$ bits. If we look at the recurrence formula, we see that this can happen near a partial quotient of 2 or more. If this is the case, we know that the approximation of x by C/D must be quadratically bad, i.e. $|x - C/D| > 1/(2D^2)$. Otherwise, let i be the index of the partial (or possibly final) convergent that follows one of the right shape (numerator has exactly $p + 1$ bits). Then our lower bound is $|x - C/D| > 1/(a_i + 2)D^2$.

In either case, if k is an upper bound on all relevant partial quotients, we have:

$$|x - \frac{C}{D}| > \frac{1}{(k + 2)D^2}$$

provided that conversion is not exact.

We conclude that there is a suitable lower bound L for non-zero $|2F|$:

$$L = \frac{10^{-q}}{(k+2)}$$

where k is as described above, and q is the decimal source precision.

The extra precision required for intermediate computation is therefore the number of bits in $k+2$ plus $3.3q$ (number of bits in D), plus whatever is needed to cover rounding or truncation errors during the actual computation (there may be several steps), or the approximation error in any inexact constants (e.g. in a table of selected powers of ten in some internal binary floating-point representation). (The constant 3.3 above stands for $\log(10)/\log(2)$, or the base-2 logarithm of 10.)

We're not quite done yet, however, until we've established this bound k on *all relevant partial quotients*.

Because both decimal D and binary C are assumed normalised, the value of C/D has a dynamic range (ratio of largest to smallest possible values) of 20. Each decimal exponent d may therefore lead to up to five possible corresponding binary exponents b, and hence four or five different values of x, whose Continued Fraction expansion is needed until a partial convergent of at least $p+1$ bits in the numerator is reached. Another way to describe it is that for each binary exponent b there are one or two possible values of d.

For any fixed BFP format, the range of b is bounded by the format's exponent range. The total number of Continued Fraction expansions needed is small enough (about 45,000 for 128-bit IEEE Extended Precision) to permit an exhaustive search for the largest applicable partial quotient — and the perhaps surprising answer is that the largest partial quotient for 36-digit decimal input and 128-bit IEEE Extended Precision output is only 13117, a 13-bit number. For IEEE Double Precision and 19-digit decimal input, the largest applicable partial quotient is a puny 580, a 10-bit number. Interestingly, the largest partial quotient for IEEE Double and 17-digit decimal input is 1199, an 11-bit number. This illustrates the fact that the number of significant decimal input digits dominates the required intermediate precision, each digit requiring an extra 3.3 bits: the bound L for 17-digit input is 50 times larger than that for 19-digit input.

To cover the complete range of fixed-precision conversions to a given BFP format, one has to address denormal (subnormal) numbers too: these are numbers used to represent gradual underflow, when the minimal exponent has been reached, and significands of fewer than p bits are generated. This affects the balance between the size of D and the size of C, and leads to p additional

values of x whose Continued Fraction expansion has to be computed (out to progressively smaller partial convergent sizes).

We shall see later that it is in fact sufficient to compute a sparse subset of applicable Continued Fraction expansions. This is important when we consider unbounded decimal input precision, where the number of possible values of x is much larger. It also implies that additional cases to deal with subnormal numbers can in fact be ignored (they cannot lead to a tighter required error bound L).

Ideally, we would like to be able to derive bounds on the applicable partial quotients for any collection of ratios of powers of two and five. It would then be possible to compute a suitable bound from the specification of the BFP format alone (precision and exponent range), without having to resort to an explicit search (sparse or not).

5 Decimal-to-Binary conversion with unbounded source precision

The target precision is still assumed to be that of a given BFP format, but now we impose no constraints at all on the decimal input, and we promise to take *all* given digits into account. Leading and trailing zeros can be discarded but may have to be counted if they affect the effective decimal exponent, which they may depending on the relative position of a given decimal point.

The first observation to make is that the maximum number of decimal digits we will have to compute with is bounded by the target BFP format — any digits beyond that need only be checked to see if they are all zero (in which case they would be discarded, and their effect folded into the effective exponent). The reason for this bound is that any BFP number has an exact decimal representation, and the one with the largest number of significant digits (not counting leading or trailing zeros) corresponds to the number represented by a significand of all-one bits, and the minimum exponent $Emin$. When $Emin < -(p+1)$ (which is the case for all practical BFP formats) this number, $(2^p - 1)2^{Emin-p-1}$, has $p + 1 + .7(-Emin)$ decimal digits (note that $Emin$ is negative). In fact, we need to collect one additional digit, since we need to recognise exact half-way points in the case of round-to-nearest. Any non-zero digits beyond this can be folded into a single extra non-zero "sticky" digit. (The .7 above stands for the base-10 logarithm of 5, i.e. .69897...)

The number of needed decimal digits can be determined dynamically after the effective exponent is known; the formula above represents the worst case for a given BFP format. This observation can significantly cut down the computational effort for numbers with middle-of-the-road exponents. When b is posi-

tive, the maximum needed q is simply $1+.3(b+p)$; otherwise it is $p+1-.7(b+p)$ (again, .3 and .7 are simply shorthand for the decimal logarithm of 2 respectively 5).

The analysis of the preceding section still applies, only now we have a variable (though still bounded, as described above) decimal input precision q, and, more importantly, the relative sizes of the decimal significand D and the binary significand B (or C) can vary wildly. This affects the range of values of x whose Continued Fractions have to be explored. It is clear that we have to compute with at least q digits (roughly $3.3q$ bits), because a given number that differs from an exact rounding threshold by one in the least significant digit must be rounded to the correct side of that threshold. This requirement was of course already implicit in the formula for the error threshold $L = 10^{-q}/(k+2)$ given in the preceding section.

The number of values of x that need to be checked to establish bounds for fixed-precision conversion grows linearly as a function of the binary exponent range, but for unbounded decimal input precision it grows quadratically due to the double contribution of exponent range and size range for the ratios x of powers of two and five to be explored. For 128-bit IEEE extended precision the binary exponent range is 2^{15} and the maximum number of decimal digits q is 11550, leading to about 100 million combinations (there will be a small amount of overlap, i.e. some ratios x may result from two different combinations of d and q): for each value of b there are one or two values of $d+q$, with q varying with more or less discipline, as suggested below.

If we only wanted to know the maximum precision we would ever need, it would be sufficient to explore all possible values of x for the largest possible q — e.g. the $q = 11550$ mentioned above. That would however be gross overkill to use for any reasonable input: this is why we need the two-dimensional search. (In the fixed-precision case we don't mind computing with the precision needed for, say, 36 decimal digits, when the given number has only 6 significant digits (and 30 trailing zeros in the normalised D), and that is why a one-dimensional search across the exponent range suffices.)

What we can do however is increase the granularity of q. For example, the bignum approach of [1] packages decimal digits in groups of 9, so q will always be a multiple of 9. This cuts down the search by a factor of 9.

We can also make the two-dimensional searchscape triangular instead of square, by using a maximum for q that depends on b. This cuts the area to be searched roughly in half — but it is still quadratic in terms of the exponent range.

So far we've only addressed the number of points to be explored. The total cost of exploration is higher, because of the increasing cost of Continued Fraction expansion as the exponents grow. Luckily the cost of computing a limited

Continued Fraction expansion (up to a partial convergent of a fixed size) increases only linearly with the number of digits of the powers or two and five; the cost of a full expansion would be quadratic. This raises the total cost by one order (quadratic for bounded precision, cubic for unbounded precision, as a function of the exponent range).

This is why the next section is important.

6 The hunt for large partial quotients of ratios of powers

We are looking for the largest partial quotient for partial convergents of the right size, i.e. with a $p+1$-bit numerator among a collection of successive ratios of powers of two and five. Is there a way to get the same information from a sparse subset of these ratios? It turns out there is, because the partial quotients of the expansion of mx or of x/m are related to those of the expansion of x. Hurwitz published a recurrence relation in 1891 (reference from exercises in Knuth [7]) for the case of doubling (or halving), in which the size of the largest partial quotient does not change by more than a factor of roughly two ($(2a+1)/a$ to be precise). This rule holds in general for integral factors m as we shall show (though we won't give a neat recurrence relation to compute all partial quotients of mx from those of x, as Hurwitz did for the case $m = 2$). We are in fact interested only in partial quotients near partial convergents of a given size, but the rule holds in that case too.

Suppose we have a partial convergent P/Q for x, where P is of the right size, and the following partial quotient is a, and let us consider partial convergents for $y = mx$ for integer m. From the approximation bounds for Continued Fractions we have:

$$\frac{1}{(a+2)Q^2} < |x - \frac{P}{Q}| < \frac{1}{aQ^2}$$

Let us multiply through by m and replace mx with y. There are several cases, depending on the Greatest Common Divisor g of m and Q. Let $g = \gcd(m, Q)$ and let $f = m/g$ be the cofactor of g in m.

Case (m1): m divides Q, i.e. $f = 1$. We get (keeping the approximation in reduced form):

$$\frac{m}{(a+2)Q^2} < |y - \frac{P}{(Q/m)}| < \frac{1}{ma(Q/m)^2}$$

In this case, $P/(Q/m)$ clearly satisfies the condition for being a partial conver-

gent of y (since $ma \geq 2$), and the numerator has not changed, so it is still of the right size. We don't know yet what the corresponding partial quotient is; we do know that it is at least ma. That's because the approximation derived here is good enough to qualify $P/(Q/m)$ as a partial quotient, but the approximation might actually be better than $1/ma(Q/m)^2$. Let c be the actual partial quotient; then we have:

$$\frac{m}{(a+2)Q^2} < |y - \frac{P}{(Q/m)}| < \frac{1}{c(Q/m)^2} \leq \frac{1}{ma(Q/m)^2}$$

From this it is easy to extract the bounds $m(a+2) > c \geq ma$. When a is large relative to m (the only case of interest), this shows that the growth of the partial quotient is roughly a factor of m.

Case (m2): m does not divide Q. We get (still in reduced form, by exposing the cofactor $f > 1$ — possibly $f = m$, e.g. when m is prime):

$$\frac{m}{(a+2)Q^2} < |y - \frac{fP}{(Q/g)}| < \frac{f}{ga(Q/g)^2}$$

This leads to an approximation whose numerator is too large, even when $m = 2$, because if P had $p+1$ bits, fP will have at least $p+2$ bits. Instead, we should look for an approximation R/S of the right size ($2^p \leq R < 2^{p+1}$) which was not derived by the numerator-preserving transformation (m1) described above. If there is such an approximation, it could have been derived from an earlier partial convergent P/Q of x by transformation (m2), with $R = fP$ and $S = Q/g$. Let c be the partial quotient that follows the partial convergent R/S of y described in this case (noting that $Q = gS$):

$$\frac{f}{g(a+2)S^2} < |y - \frac{R}{S}| < \frac{1}{cS^2} \leq \frac{f}{gaS^2}$$

This leads to the bounds $g(a+2)/f > c \geq ga/f$ for the partial quotients.

The relative size of a and c depends on the balance between the GCD and its cofactor in m; this factor ranges from $1/m$ to m. The greatest increase from a to c occurs for case (m1) described above; in the other cases, a is an *early* partial quotient of x, and the increase is less than a factor of $m/2$ (roughly), and a decrease of up to a factor of m is possible. If we are tracking the development of large partial quotients and plan to skip some expansions, we have to keep track of early partial quotients too, not just those that correspond to a partial convergent of the right size.

Case (m3): R/S is a partial convergent of $y = mx$ that was not derived by either of the two transformations (m1) or (m2). It must be that R has no factor

in common with m (not case (m2), and R/mS is not a partial convergent of x (not case (m1)). The second fact implies that $|x - R/mS| > 1/2(mS)^2$. From this we conclude that $m/2(mS)^2 < 1/cS^2$, or: $c < 2m$. In other words, case (m3) can't happen for a *large* partial quotient c.

Now let's consider division of x by an integer m. The situation is pretty much the same as for multiplication, if we reverse the roles of numerator and denominator. It is not quite the same however because we are focusing on the *numerators* of a certain size, $p+1$ bits. Let $z = x/m$ now. What matters here are common factors of m and the numerator P, so let $g = \gcd(m, P)$ and, as before, let $f = m/g$ be the cofactor of g in m.

We still start with:

$$\frac{1}{(a+2)Q^2} < |x - \frac{P}{Q}| < \frac{1}{aQ^2}$$

Case (d1): No factor of m divides P (i.e. $g = 1$ and $f = m$). We get an approximation P/mQ for z with an unchanged numerator, hence of the right size:

$$\frac{m}{(a+2)(mQ)^2} < |z - \frac{P}{mQ)}| < \frac{m}{a(mQ)^2}$$

This will be a partial convergent provided that $a > 2m$ — and we shall assume this, being interested only in large partial quotients. In that case we get the following bounds on the corresponding partial quotient q: $(a+2)/m > c \geq a/m$. (It shrinks by a factor of at most m.) This case is just the reverse of case (m1) above.

Case (d2): $\gcd(m, P) > 1$. We get an approximation with a smaller numerator P/g which could be an early partial convergent of z, depending on the relative size of g and its cofactor $f < m$:

$$\frac{1}{m(a+2)Q^2} < |z - \frac{(P/g)}{fQ}| < \frac{f}{ga(fQ)^2}$$

Again, when a is large (at any rate, larger than $2m$ or even just larger than $2f/g$), $(P/g)/fQ$ will be a partial convergent of z, and its partial quotient c would be bounded by a factor in the range $1/m$ to m relative to a or $a+2$.

An appropriate-size partial convergent R/S for z in the case where the numerator-preserving transformation (d1) does not apply can only be an instance of (d2) which derives R/S from a *late* partial convergent of x, i.e. one with a numerator gR of more than $p+1$ bits. That's because there is no parallel to

case (m3), which does not have a partial convergent at the other end of the transformation. What's important in this case is that c may be derived from a *smaller* a for that late partial convergent, which means that if we are tracking the development of large partial quotients and plan to skip some expansions, we have to run the earlier expansions a few steps beyond p+1-bit numerators.

If we forget for a moment that we are focussing on convergents of a particular size, and just look at the fate of the largest partial quotient in the Continued Fraction expansions of x and either mx or x/m, we get:

Theorem 1 *If a is the largest partial quotient in the Continued Fraction expansion of x, and m is an integer, then the largest partial quotient c in the expansion of either mx or x/m is bounded by $a/m \leq c < (a+2)m$.*

7 Exploring the range

Recall the definition of the relevant parameters:

The target BFP format has a precision of p bits and an effective exponent range $Emin - (p-1) < b < Emax - (p-1)$ (the exponent range is usually quoted for the normalised exponent, hence the offset of $p-1$).

We start with a q-digit normalised integral decimal significand D with an appropriate exponent d and a value of

$$D \times 10^d \quad \text{where} \quad 10^{q-1} \leq D < 10^q$$

This is converted to a matching $p+1$-bit integer C such that:

$$10^d D = 2^{b-1}(C + 2F) \quad \text{where} \quad |F| < 1/2$$

Here C is an integer in the range $2^p + 1$ to $2^{p+1} - 1$.

We are interested in all possible values of the conversion ratio x for the given BFP format (precision and exponent range), and for either a fixed decimal precision q, or one bounded only by the BFP format as described in section 5. We have:

$$x = \frac{(C + 2F)}{D} = \frac{5^d}{2^{b-1-d}}$$

The normalisation constraints on C and D imply:

$$\lfloor b\log(2)\rfloor \le d+q \le 1+\lceil b\log(2)\rceil$$

The bound on q implied by a given binary exponent b (see section 5) is:

$$q \le \begin{cases} 1+\lceil (b+p)\log(2)\rceil & \text{when } b \ge 0 \\ p+1-\lfloor (b+p)\log(5))\rfloor & \text{otherwise} \end{cases}$$

The formula for negative b overestimates by one when $-p \le b \le 0$; this is not worth correcting for. When q is thus constrained by b, the bounds for $d+q$ imply that b and d will have the same sign, except when q is within $p\log(2)$ of its maximum for positive b, in which case x will turn out to be the inverse of an integer, and won't have any partial quotients other than the first, which won't matter. Except for this, x will be a ratio of a power of 5 divided by a power of 2 when b is positive, or a ratio of a power of 2 divided by a power of 5 when b is negative or zero. For each value of b, q can range from small (e.g. the minimal fixed precision) to the maximum that depends on b (when we consider unbounded decimal source precision). This bounds the search space.

Typically we would run through an arithmetic sequence of exponents, taking advantage of the growth constraints on partial quotients to bound the ratio of any missed large partial quotient to the largest one found in the sparse run. For example, each value of $d+q$ corresponds to three or four possible adjacent values of b: by picking only one, we might miss a large partial quotient due the skipped values — but it could not be off my more than a factor of 4 (two factors of 2) from the closest largest partial quotient on either side in the search sequence. It turns out to be easier to pick d and q as the independent variables, and deduce one of the corresponding values of b. In practice, q can be varied in relatively large steps (e.g. 9 or 16) without loss of information (as mentioned in section 5), so additional sparseness can be introduced by spacing the values of d, which affect powers of 5. Skipping three out of four would lead to another factor of 25 in the uncertainty of the value of the largest partial quotient missed relative to those encountered during the search, for a total uncertainty of a factor of 100 (7 bits of possibly over-estimated needed precision) and a factor of 16 reduction in search space. If we're willing to overestimate the needed precision by 16 bits, we could skip 9 out of 10 powers of five.

In fact, we don't need to overestimate at all. Large partial quotients are rare, so whenever we encounter a new large partial quotient whose size is 16 bits less than the truly largest partial quotient encountered in a refined search (less or no skipping of powers), we perform a new refined search of the skipped

interval. The refinement could be progressive or all the way at once; the choice depends on the coarseness of the main run. (For large negative exponents sparse exploration becomes essential.)

Here is a practical search strategy. Let $\alpha = \log(5)/\log(2)$ and $\beta = 1/\log(2)$ — the values are approximately 2.3 and 3.3 — and let us rewrite the expression of x in terms of d and q. The multiplications by α or β are assumed to be truncating to integral values. We don't mind being off-by-one here: in a sparse search it doesn't matter, and in a refined search we can add extra endpoints to make sure we don't miss anything. We get:

$$x = \begin{cases} \frac{5^d}{2^{\alpha d - \beta q}} & \text{for positive } d \\ \\ \frac{2^{\alpha(-d)+\beta q}}{5^{-d}} & \text{for negative } d \end{cases}$$

Positive d runs from small to $(Emax - p)/\beta$ (for a given BFP format), and for each d, q runs from small up to d.

Negative d runs from -1 to $Emin - p/\beta$, and q runs from small up to $-d$ (as q increases, so does b (for fixed d), until b ceases to be negative, at which point q will have reached $-d$). Note that this range is much larger than that for positive d — eleven times larger actually (the range of both d and q is larger by a factor of $1/\beta$, whose square is 11.03). If we include the effect of larger exponents, the computation cost for extreme negative exponents becomes quite expensive.

Note that $2^{\lfloor \alpha n \rfloor}$ is the largest power of 2 that is less than 5^n, and can be derived trivially when both are represented in binary (or bigbinary). This can then be shifted by the q-dependent term to provide the starting point for a Continued Fraction expansion by Euclid's method. The size of the remainder can be monitored to determine when the expansion has collected enough partial quotients to pass the desired size of partial convergent, without actually having to compute the partial convergents. We need to explore enough beyond numerator size $p + 1$ bits to see those late partial quotients that may affect the partial quotients of interest, as described in section 6.

These techniques permit the determination of the largest relevant partial quotient even for wide-exponent range extended BFP target formats for correctly-rounding conversion across the entire exponent range, and for unbounded source precision. The total search space of 100 Million for IEEE Extended Precision (our first gross estimate in section 5) is now reduced to maybe 100 Thousand. Of course, a simple formula for a bound on the largest relevant partial quotient would be even nicer...

8 A formula for a bound on partial quotients?

Preliminary exploration of the range for IEEE Extended Precision has not uncovered any very large partial quotients. One refinement around a partial quotient of about 1.6 Million found a partial quotient of over 500 Million (29 bits), but nothing larger yet. (This particular big partial quotient occurred for a partial convergent of the "wrong" size; the search was not actually computing partial convergents, so we can only find upper bounds on partial quotients in the explored exponent ranges.) An extensive search of complete Continued Fraction expansions of near-unity ratios of powers of two and five (carried out in 1999, before we knew to look only for partial convergents of the "right" size) eventually found one big 35-bit partial quotient for $5^{28888}/2^{67078}$. Why aren't there any bigger ones? It turns out that the frequency of large partial quotients closely matches that predicted by Gauss – but for our purposes we need a *proof* that no partial quotient above a certain bound exists (for partial convergents of the "right" size). By the way, that large partial quotient is quite a champ: no other 35-bit partial quotient was found until $5^{65125}/2^{151214}$ was reached, and it was smaller. (At that point the search was abandoned.)

It would have been nice to discover a rule similar to that of Theorem 1 for x^2 in terms the largest partial quotient for x, because that might have led to a growth rule compatible with observations. Such a rule cannot exist for squaring in general, however, because of the following counterexample: If a/b is a partial convergent of $\sqrt{2}$, then the expansion of $(a/b)^2$ has a partial quotient larger than $b^2/\sqrt{2}$ (hence arbitrarily large), yet the partial quotients of the expansion of a/b are all equal to 2. And that's the problem: all general rules about the maximum size of partial quotients bound them to not much less than the size of the original numerator or denominator n of the rational number being expanded. That's a far cry from the logarithmic growth that we are looking for! (A very crude fit suggests a maximum partial quotient proportional to the product of the exponents of two and five in $5^a/2^b$.)

9 Conclusion

Using Continued Fraction expansions of a set of ratios of powers of two and five we can derive tight bounds on the intermediate precision required to perform correctly-rounding floating-point conversion: it is the sum of three components: the number of bits in the target format, the number of bits in the source format, and the number of bits in the largest partial quotient that follows a partial convergent of the "right" size among those Continued Fraction expansions. (This is in addition to the small number of bits needed to cover computational loss, e.g. when multiple truncating or rounding multiplications

are performed.)

When both source and target precision are fixed, the set of ratios to be expanded grows linearly with the target exponent range, and is small enough to permit a simple exhaustive search, in the case of the IEEE 754 standard formats: the extra number of bits (3rd component of the sum mentioned above) is 11 for 19-digit Double Precision and 13 for 36-digit Extended Precision.

When the source precision is unbounded, the set of ratios to be explored grows quadratically (and the computation cost cubically) with the target exponent range, and becomes unreasonably large. A (possibly) new Theorem on the growth of partial quotients under multiplication (or division) by an integer permits sparse exploration of the set of ratios in order to make the cost manageable.

We are still looking for a useful formula to bound the largest partial quotient of a ratio of powers of two and five, so as to avoid the need for an expensive search (sparse or not).

10 Acknowledgements

I wish to thank Alan Stern, then at the Rowland Institute in Cambridge, Massachusetts, for a spirited e-mail exchange during the spring of 2000 following the publication of [1]. He pointed out that the hints at improvement mentioned in that article were still too conservative.

References

[1] Abbott et al., "Architecture and software support in IBM S/390 Parallel Enterprise Servers for IEEE Floating-Point arithmetic" *IBM Journal of Research and Development*, vol. 43, 733–741, Nov 1999.

[2] W. D. Clinger, "How to Read Floating Point Numbers Accurately," *Proc. ACM SIGPLAN 90 Conference on Programming Design and Implementation*, 1990.

[3] J. T. Coonen, "Contributions to a Proposed Standard for Binary Floating-point Arithmetic," *Ph.D. Thesis, University of California, Berkeley*, 1984.

[4] Kenton Hanson, "Economical Correctly Rounded Binary Decimal Conversions," published on the web with a given date of 1997-12-19, now inaccessible.

[5] G. H. Hardy and E. M. Wright, "An Introduction to the Theory of Numbers," Oxford Science Publications (*Oxford University Press*), 5th ed. 1979.

[6] A. Ya. Khinchin, "Continued Fractions", Moskow 1935; English translation in *University of Chicago Press*, 1964.

[7] D. Knuth, "The Art of Computer Programming", vol. 2, *Addison-Wesley*, 1969.

[8] D. Matula, "The Base Conversion Theorem,", *Proc. Amer. Math Soc., vol. 19, no. 3*, 1968.

[9] G. Slishman, "Fast and Perfectly Rounding Decimal/Hexadecimal Conversions." *IBM Research Report RC 15683*, 1990.

[10] G. L. Steele and J. L. White, "How to Print Floating Point Numbers Accurately," *Proc. ACM SIGPLAN 90 Conference on Programming Design and Implementation*, 1990.