ON A TAPERED FLOATING POINT SYSTEM*

Aqil M. Azmi+ and Fabrizio Lombardi++

++Texas A&M University
Department of Comp. Science
College Station, Tx 77843
+University of Colorado
Department of ECE, Boulder

Abstract

Morris [Mor71] suggested adding an extra field to that of the fixed floating point system, thus exponents can be stored more efficiently. The exponents are stored in the smallest possible space, passing the extra bits to the mantissa. The extra field is used to monitor the current length of the exponent. The gain in precision and/or exponent range outweights the overhead of the extra field and the processing speed. Unfortunately Morris' paper lacked the implementation detail and the comparison with existing systems. In this paper we provide implementation details, error analysis and some future research ideas. Simulation results are provided for comparison purposes.

1 Introduction

A floating point system can be defined by a 3-tuple (r, q, p) where r is the radix of the exponent, q and p are respectively the exponent's and mantissa's length in bits. Any new floating point system addresses one or more of the following issues:

- *Precision:* This measure of accuracy in the calculation. A simple, but not always practical solution is to increase the size of the mantissa field.
- *Range:* To be able to access larger and larger numbers. The easiest solution is to increase the size of the exponent field, or to use higher radix for the exponent.

Morris [Mor71] suggested a "tapered" system, where there is a tradeoff between the exponent range and the precision of the floating point. Unfortunately Morris didn't discuss implementation issues nor gave them an analysis for it. The reason behind a "tapered" system can be attributed to

1. Most applications which require the greatest accuracy of representation, are likely not to generate numbers of extremely large or extremely small magnitude [Mor71].
2. Exponents are certainly not equally distributed. They mostly follow the Gaussian distribution [Ham70].

In [Mat81], the extension of this method for avoiding overflow/underflow is presented. This approach is very elegant, but impractical due to the complex process involved in conversion from one format to another (when the lengths of the exponent fields are different).

In [Olv87A] and [Olv87B] , the concept of level-index arithmetic is introduced. The closure property of this system which prevents overflow/underflow, is fully characterized. It is proven that a gradual erosion of relative precision occurs. An implementation based on partial table loop-up is proposed.

In this work, we further investigate Morris' suggested system. The implementation is discussed, the performance analysis compared with that of a fixed floating point is also given.

2 The Implementation of Tapered System

In a fixed floating point system, exponents occupy the same space regardless of their value, which can be otherwise used to get higher precision [Mor71]. The *TFP* floating point system takes advantage of this drawback. The *TFP* system attempts to efficiently distribute the storage between the exponent and the mantissa.

Let S denote the sign field; E and M denote the exponent and the mantissa fields. The *TFP* is a three field $< G, S, W >$ of fixed length floating point system with the exponent and mantissa combined into a single field $W =< E, M >$. This is due to their varying lengths, however the combined length is fixed. The G-field † keeps track of the current length of the exponent. Kornerup [Ko83] uses a similar k-field to indicate the position of the slash in the floating-slash arithmetic unit.

A *TFP* system can be characterized by a 3-tuple (r, g, w), where r is the radix of the exponent, g is the length of the G-field in bits, w is the length of W-field (Figure 1).

A notation is devised to differentiate between the 3-tuples of a fixed floating point with that of *TFP*, or to describe the size of the exponent (mantissa) field of a *TFP* system at a certain instant. The notation consists of subscripting the parenthesis of the 3-tuples either by f or t for fixed *TFP*, respectively. The tuple $(2,3,29)_t$ means that a *TFP* system‡ is considered. This system has radix $r = 2$, length of the G-field $g = 3$ bits, while the length of W-field is 29 bits. Similarly the tuple $(2,8,24)_f$

*This research supported in part by grants from NATO and AT&T.

†Following Morris'notation.

means the radix $r = 2$, $q = 8$ and $p = 24$ bits, these are the length of the exponent and mantissa respectively.

The simplest *TFP* arithmetic processor can be thought of as a fixed floating point processor with a pre- and post-processing stages (Figure 2). Let two inputs A and B be given in *TFP* format. The pre-processing stage (unpack operation) converts them to fixed floating point format $\hat{A}$ and $\hat{B}$. The fixed floating point format output $\hat{C}$ has to undergo a pack operation to convert back to a *TFP* format.

For a given $(r, g, w)_t$ *TFP* system, the fixed floating point processor should be able to handle all possible numbers, such as exponents of length 2^g bits and simultaneously $w - 1$ bits mantissa. This implies a fixed floating point processor of 2^g bits for the exponent, and at least $w - 1$ bits in the manitssa. The unpack operation transforms a *TFP* input $A(r, q_A, w - q_A)_f$ to $\hat{A}(r, 2^g, w - 1)_f, q_A - 1$ zeros are appended to the lower order mantissa bits. The pack operation retransforms the result from $\hat{C}(r, 2_g, w - 1)_f$ to $C(r, q_c, w - q_c)_f$, where the $q_c - 1$ lower order mantissa bits will not be used.

The pack processor (Figure 3) transforms the output of the fixed floating point processor $\hat{C}(r, 2^g, w - 1)_f$ to $C(r, q_c, w - q_c)_f$ in the *TFP* format.

The pack processor includes the rounding mechanism. The packing operation must be performed after the normalization process and prior to rounding. This is due to the dispossession of the lower $q_c - 1$ mantissa bits during the packing. The dispossessed bits will be used in the rounding. The q_c exponents bits must have the least number of bits to accommodate the exponent and achieve the best possible precision. Any non-zero integral number α will require at least $\lfloor 1 + \log_2 \mid \alpha \mid \rfloor + 1$ bits to be representable uniquely. The extra bit is needed in order for the positive and negative numbers to be unique. The transformation process must be also as simple as possible.

Exponent	$e_c{}^a$	$e_c{}^b$	γ^c
0	100.....0	1	000
-1	011.....1	0	000
1	100.....01	11	001
-2	011.....10	00	001
2	100....010	110	010
-3	011....101	001	010
3	100....011	111	010
-4	011....100	000	010
4	100...0100	1100	011
-5	011....011	0011	011

[a]biased exponent of the fixed floating point $\hat{C}$.
[b]biased exponent of the TFP number C.
[c]contents of the G-field, $g = 3$ is assumed.

Table 1: The pack operation table.

Based on the model in which smaller exponents in the absolute term are used more frequently than larger ones, e.g the numbers $10^0, 10^1$ are more frequently used than 10^{20} or 10^{-20}, Huffman coding technique assigns codes of length inversely proportional to the frequency of occurrence [Hu52]. This is a common technique for encoding information. A simple and elegant systematic exponent packing algorithm similar to Huffman coding principles had been found as shown in Table 1. Numbers around 1.0 will have the highest precision. Figure 4 shows an example of packing.

Let γ be the contents of the G-field. The *TFP* exponent e_c is biased by 2^γ. The actual exponent value therefore is $e_c - 2^\gamma$. This can be viewed as a viewed as a variable bias exponent.

Definition 1. The packing operation is a mapping *pack* : $N \rightarrow N \times N$

$$\tilde{e} \xrightarrow{\text{pack}} (\gamma, e), \tag{1}$$

where N is the set of non-negetive integers, $\tilde{e}$ is the biased exponent in the fixed floating point format. γ and e are the contents of the G-field and the biased exponent in the *TFP* format respectively.

γ and e are computed as follows. Let q be the number of bits in the exponent of the fixed floating point. $\tilde{e}$ is biased by 2^{q-1}. Hence,

$$\tilde{e} \xrightarrow{\text{pack}} (\gamma, \tilde{e} - 2^{q-1} + 2^\gamma), \tag{2}$$

such that

$$\gamma = \begin{cases} 0, & \text{if the unbiased exponent is zero} \\ \lfloor 1 + \log_2 |\tilde{e} - 2^{q-1}| \rfloor - \xi, & \text{otherwise,} \end{cases} \tag{3}$$

where,

$$\xi = \begin{cases} 1, & \text{if the unbiased exponent} = -2^i, i \geq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Lemma 1. The pack operation is unique.

Proof: It is required to show that for two exponents $\tilde{e}_1$ and $\tilde{e}_2, \tilde{e}_1 \neq \tilde{e}_2$ in the fixed floating point format, and the transformation $\tilde{e} \xrightarrow{\text{pack}} (\gamma, e)$, results in $(\gamma_1, e_1) \neq (\gamma_2, e_2)$. This can easily be shown by contradiction. Assume $(\gamma_1, e_1) = (\gamma_2, e_2)$, then using the previous Definition

$$\begin{aligned} e_1 - e_2 &= \tilde{e}_1 - \tilde{e}_2 + 2^{\gamma 1} - 2^{\gamma 2} \\ &= 0, \end{aligned} \tag{5}$$

which implies $\tilde{e}_1 = \tilde{e}_2$, a contradiction. □

The packing operation can be described algorithmically as follows:
let $COUNT$ be a g-bit counter, and let SR be a shift register of $(2^g + w - 1)$ bits long. SR_1 is the leftmost bit.

Step 1: Load the shift register SR with the exponent concatenated with the mantissa of $\hat{C}$.

Step 2: Set all the bits of the $COUNT$ to 1.

Step 3:

While $SR_1 \neq SR_2$ and $COUNT \neq 0$
{
- shift left the contents of SR, such that SR_1 is

[‡]This system fits into 32 bits, as we take the advantage of the implicit bit whenever radix $r = 2$ is used.

unchanged,
$SR_i \leftarrow SR_{i+1}$ for all $i > 1$.
• decrement $COUNT$.
}

Step 4: Load $COUNT$ into the G-field of C.

Step 5: Load the contents of the shift register SR into the W-field of C.

Step 6: Perform the rounding operation on the mantissa of C.

Step 7: Copy the sign bit of $\hat{C}$ to C.

The unpack processor (Figure 5) is the preprocessing stage to the fixed floating point processor. An input in the TFP format is transformed to a fixed floating point format. It can be shown that the unpack operation is unique, i.e.,

$$\bar{e} \underset{\text{unpack}}{\overset{\text{pack}}{\longleftrightarrow}} (\gamma, e) \tag{6}$$

Unpacking can be described by an algorithm similar to the packing algorithm.

3 Theoretical Analysis of Tapered Arithmetic

Theoretical analysis of floating point is not limited to error analysis, but it certainly is the main object. Error analysis of floating point operations has proved to be rather complicated.

The exponent range (ER) of a $(2^k, q, p)_f$ fixed floating point of radix $r = 2^k$ is defined as

$$ER(2^k, q, p)_f = 2^{q-1} \times k - 1. \tag{7}$$

This equation appears (incorrectly) as $k \times (2^{q-1} - 1)$ in the original definition [Bro69]. Unfortunately Cody [Cod73] has misquoted (6) and others [Hw79], [Wa82] copied his mistake. For a TFP system,

$$ER(2^k, g, w)_t = 2^{2^g - 1} \times k - 1. \tag{8}$$

To increase the exponent range of a TFP system over that of a fixed floating point of same radix, the length of the G-field g must be greater than $\log_2 q$. The values of ER for some TFP systems of different radices are given in Table 2.

Table 2 ER for different TFP systems

r	g	w	Largest mant. (bits)	Smallest mant. (bits)	ER
2	3	29	28	21	127
2	4	28	27	12	32767
4	3	28	27	20	255
16	3	28	27	20	511

Let X be a real number with exponent x within the system range. The maximum relative representation error ($MRRE$) [Wilk63] over all normalized fractions for a TFP system can be written as

$$\begin{aligned} MRRE(r, g, w) &= \frac{\text{maximum representation error}}{\text{smallest normalized fraction}} \\ &= \frac{\frac{1}{2} \times 2^{-(w-2^g)} r^x}{\frac{1}{r} \times r^x} \\ &= 2^{-(w-2^g+1)} \times r. \end{aligned} \tag{9}$$

From (8), smaller radix would reduce the $MRRE$. And as might be expected the $MRRE$ of a TFP system is worse than that of a fixed floating point system of same size and radix. This is due to the space allotted to the G-field. Table 3 shows the $MRRE$ of some different TFP systems.

Table 3. $MRRE$ for different TFP systems

r	g	w	$MRRE$
2	3	29	2^{-21}
2	4	28	2^{-12}
4	3	28	4×2^{-21}
16	3	28	16×2^{-21}

The average length of the mantissa P_{avg} is obtained by applying the Huffman codes average length equation [Hu52]

$$\begin{aligned} P_{avg}(g, w) &= \sum_i p_i \cdot \text{Prob}(p_i) \\ &= \sum_i p_i \cdot \text{Prob}(q_i), \end{aligned} \tag{10}$$

where p_i and q_i are the mantissa and exponent length corresponding to i in the G-field. For each $i, 0 \leq i < 2^g$ in the G-field, the range R_i of representable exponents is

$$R_i = \begin{cases} [-1, 0], & \text{if } i = 0 \\ [-2^i, -2^{i-1} - 1] \bigcup [2^{i-1}, 2^i - 1], & \text{otherwise.} \end{cases} \tag{11}$$

There is no good model available for the distribution of the exponents. Sweeney [Sw65] analyzed floating point addition. His analysis was based on collecting almost two hundred thousand floating point numbers and finding the average length of the shift during normalization. No model was given. Hamming [Ham70] mentioned that it is reasonable to consider exponents to be normally distributed.

The unbiased exponent $-2^{2^g-1} \leq e < 2^{2^g-1}$ takes on discrete values. Their distribution will be approximated by a normal distribution of zero mean and standard deviation σ, then

$$\text{Prob}(q_i) = \frac{1}{\sqrt{2\pi}\sigma} \int_{R_i} e^{-x^2/2\sigma^2} dx. \tag{12}$$

The following expression has been used to facilitate the integration

$$R_i = \begin{cases} [-1.5, 0.5], & \text{if} i = 0 \\ [-2^i - 0.5, -2^i - 0.5] \bigcup [2^{i-1} - 0.5, 2^i - 0.5], & \text{otherwise.} \end{cases} \tag{13}$$

so that the discrete probabilities will add to one. The contents of the G-field in a TFP system is one less than the exponent's length. Equation (9) can be rewritten as

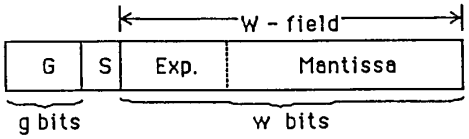

Figure 1. An $(r, g, w)_t$ TFP floating point representation.

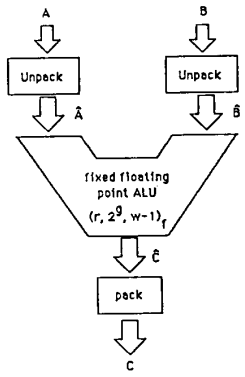

Figure 2. TFP arithmetic processor.
A, B, and C are in TFP format. $\hat{A}, \hat{B}$, and $\hat{C}$ are in fixed format.

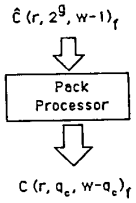

Figure 3. The pack processor

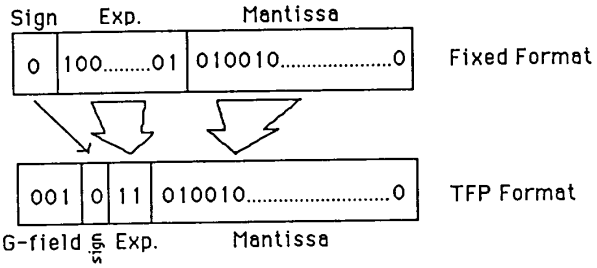

Figure 4. An example of packing a fixed floating point.

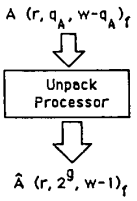

Figure 5. The unpack processor.

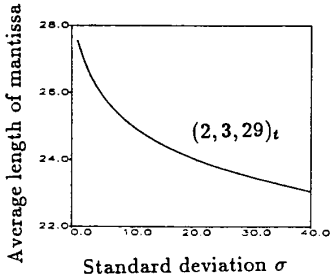

Figure 6. Average length of the mantissa P_{avg} as a function of standard deviation σ.

$$P_{\text{avg}}(g,w) = \frac{1}{\sqrt{2\pi}\sigma} \sum_{i=0}^{2^g-1} (w-i-1) \cdot \int_{R_i} e^{-x^2/2\sigma^2} dx. \quad (14)$$

Figure 6 shows P_{avg} for different values of the standard deviation σ.

As a consequence of the exponential factor, the interval between one floating point number and the next, is not the same over the whole range of representable numbers. For normalized mantissas within the range $1/r \leq x < 1$, Hamming [Ham70] demonstrated that the reciprocal distribution approaches the form

$$\text{Prob}(x) = \frac{1}{x \times \ln r} \quad (15)$$

Let p be the number of bits in the mantissa. Then, the mantissa bits subdivide the range from $1/r$ to 1 into intervals of size 2^{-p}. Numbers lying within one of these intervals must be represented by the nearest boundary value. The maximum value of the representation error is thus $\frac{1}{2} \times 2^{-p}$. The average value of the error in an interval is $\frac{1}{4} \times 2^{-p} = 2^{-(p+2)}$. If x is the value of the mantissa being represented, then the relative error may be taken as [Sc85]

$$Q(x) = \frac{2^{-p}}{4x} \quad (16)$$

Assume the reciprocal distribution of the mantissa. More mantissas will be having small values, that is, large relative representation errors. The Average Relative Representation Error *(ARRE)* [McK67] is defined as

$$\begin{aligned} ARRE(r,q,p)_f &= \int_{1/r}^{1} \text{Prob}(x) \times Q(x) dx \quad (17) \\ &= \frac{(r-1)}{4 \ln r} \times 2^{-p}. \end{aligned}$$

The smaller *ARRE* is, the better it is. For the *TFP* system substitute P_{avg} in place of p, so

$$ARRE(r,q,w)_t = \frac{r-1}{4 \ln r} \times 2^{-P_{\text{avg}}}. \quad (18)$$

The values of *ARRE* over different σ's for some *TFP* systems are given in Table 4.

Table 4. *ARRE* for different *TFP* systems ($g = 3$)

		Average Relative Representation Error			
r	w	$\sigma = 15$	$\sigma = 20$	$\sigma = 25$	$\sigma = 30$
2	29	$.26 \times 2^{-24}$	$.35 \times 2^{24}$	$.22 \times 2^{-23}$	$.26 \times 2^{-23}$
4	28	$.39 \times 2^{-23}$	$.52 \times 2^{-23}$	$.32 \times 2^{-22}$	$.39 \times 2^{-22}$
16	28	$.98 \times 2^{-23}$	$.13 \times 2^{-22}$	$.81 \times 2^{-22}$	$.97 \times 2^{-22}$

4 Applications

Simulation was used to evaluate some functions, or to solve a system of equations. Since it is not always possible to have an integral number of decimal digits in improvement/disimprovement, it seems more natural to use a model that indicates relative improvement. Further detail about simulation software can be found in [Az87].

Two new definitions are introduced. The definition will simplify the task of comparing different systems. Given the number x and the function f, let

$$\begin{array}{lcll} y & = & f(x) & \text{exact or most precise value} \\ y_i & = & [f(x)]_i \;\; f(x) & \text{evaluated using method } i \\ y_j & = & [f(x)]_j \;\; f(x) & \text{valuated using method } j. \end{array}$$

Definition 2. The Relative Decimal Improvement (*RDI*) of calculating $f(x)$ using method i over the method j is given by

$$RDI(y_i, y_j) = -\log_{10}\left[\frac{\epsilon(y,y_i)}{\epsilon(y,y_j)}\right], \quad (19)$$

where $\epsilon(y,\hat{y})$ is the relative error of $\hat{y}$ with respect to y.

As the relative error $\epsilon(y,\hat{y}) = |y-\hat{y}|/|y|$, the *RDI* can be rewritten as

$$RDI(y_i, y_j = -\log_{10}\left|\frac{y-y_i}{y-y_j}\right|. \quad (20)$$

As an example, consider the precise value of some calculation to be $y = 0.5432$. Also assume that $y_1 = 0.5429$ and $y_2 = 0.5439$ to be values corresponding to same calculation using methods one and two. By inspection y_1 has only two correct decimal digits, while y_2 has three. $RDI(y_1,y_2) = -\log_{10}\frac{0.0002}{0.0007} = 0.368$. This can be thought as y_1 having an improvement of 0.37 decimal digits over y_2 with respect to the correct value y. *RDI* can be summarized as

$$RDI(y_i, y_j) =$$

$$\begin{cases} > 0, & y_i \text{is better than} y_j \text{compared to the precise value} \\ = 0, & \text{no improvement} \\ < 0, & y_i \text{is not as good as} y_j \text{compared to precise value.} \end{cases} \quad (21)$$

The larger *RDI* is, the higher improvement is, and vice versa.

In the case when vectors are involved, *RDI* of individual vector elements is not expected to be the same. The following definition will be useful in this case

Definition 3. The Vector Relative Decimal Improvement (*VRDI*) in calculating the vector $\bar{x} = (x_1, x_2, \ldots, x_n)$ using method i over that calculated using method j is

$$\begin{aligned} VRDI([\bar{x}]_i, [\bar{x}]_j) &= \text{average of} RDI \text{for all the vector elements} \quad (22) \\ &= \frac{1}{n} \sum_k RDI([x_k]_i, [x_k]_j). \end{aligned}$$

As with vector norms, this is not the only possible definition for *VRDI*. We could have defined it as $\max_k RDI([x_k]_i, [x_k]_j)$. The given definition is preferred, for example, let $(2.0, -0.5, -1.8)$ be the set of *RDI*'s of calculating a vector. The average of *RDI*'s is -0.1, while the maximum of *RDI*'s is 2.0.

A brief comparison of calculating some functions using different precisions follows. The *TFP* precision is based on simulating a $(2,3,29)_t$ *TFP* system.

Sine & Cosine: Can be calculated by using the Maclaurin

expansion [Th79]. Figure 7 (Figure 8) shows the $RDI([\sin\theta]_t, [\sin\theta]_s)$ [‡] $(RDI([\cos\theta]_t, [\cos\theta]_s))$ corresponding to angle $\theta = 1, 2, \ldots, 60$ degrees.

The RDI of $\sin\theta$ is positive and it improves as θ increases. Overall $\cos\theta$ outperforms $\sin\theta$ in the range $\theta \in [0, 60]$. This is not a coincidence as $0.5 < \cos\theta \leq 1.0$ for $\theta < 60$, and *TFP* performs best for numbers around one.

Exponential: This can be calculated using again the Maclaurin expansion of e^x [Th79]. The $RDI([e^x]_t, [e^x]_s)$ is positive (Figure 9) and as x reaches a certain value, RDI starts declining. For $x = 0, RDI$ is zero as expected since no fractions are involved and $e^o = 1$. The e^x performs best in the region $x \in [-1, 0]$. This again can be attributed to $0.367 \leq e^x \leq 1.0$ for $x \in [-1, 0]$. Range reduction technique can be employed to map any x to $\acute{x} \in [-1, 0]$. This will guarantee a better approximation of e^x.

Simultaneous Linear Equations: Solving simultaneous linear equations, $\mathbf{A}\bar{x} = \mathbf{b}$ for $\bar{x}$ will give another look at the performance of the *TFP* system. Of special interests are those systems where **A** is an ill-conditioned matrix. An ill-conditioned problem is where the exact solution is extremely sensitive to "small" perturbations in the data. In solving such problems, therefore, the introduction of rounding errors can be disastrous. The LU factorization of an ill-conditioned matrix yields diagonal elements of extreme magnitudes that could cause over/underflow. This is a good test for *TFP* performance at extreme magnitudes.

The Hilbert matrix **A** of order n [At83] is defined as

$$[A_{ij}]_{n\times n} = \left[\frac{1}{i+j-1}\right]_{1\leq i,j\leq n} \tag{23}$$

and it is well known to be notoriously ill-conditioned. Let the solution of the Hilbert linear system given by $\bar{x}^T = (x_1, x_2, \ldots, x_n)$, as the RDI varies for each $x_i, VRDI$ will be used to get an overall view. Figure 10 shows the $VRDI([\bar{x}]_t, [\bar{x}]_s)$ of the solution of Hilbert system versus the order of the Hilbert matrix. Except for a Hilbert matrix of order 17, *TFP* performed better than single precision. No explanation was found for the abnormal behaviour of the *TFP* for Hilbert matrix of order 17.

5 Discussion

From the error analysis and the modeling results, some improvements in the precision have been accomplished over the fixed floating point system. The improvement is highly application dependent.

A 32 bit word can store numbers in $(2, 3, 29)_t$ *TFP* format, or $(2, 8, 24)_f$ fixed floating point format. The $ARRE$ of the $(2, 8, 24)_f$ is 0.36×2^{-24}. This is almost the same as that of the $(2, 3, 29)_t$ system when $\sigma = 20$ (see Table 4). Higher precision would be achieved if the exponents used are normally distributed with standard deviation $\sigma < 20$. The $(2, 4, 28)_t$ *TFP* system, also fits into a 32 bit word and is able to maintain at least 12 bits in the mantissa (around $3 \sim 4$ decimal digits) while simultaneously holding numbers as large (small) as $10^{9864}(10^{-9864})$.

However, is there any gain in giving up mantissa bits to the G-field as compared to giving it up to normalization of the fixed floating point of radices $r > 2$? Consider a fixed and a *TFP* floating point system both of which are 32 bits long. The fixed floating point is the $(2^n, q, 31-q)_f$ system with radix $r = 2^n$, for $n > 1$, while the $(2, g, 32-g)_t$ is a *TFP* system of binary radix.

If the exponent range ER of the *TFP* system is to be at least as large as that of fixed floating point, then equating exponent ranges this gives

$$2^g = q + \log_2 n \tag{24}$$

hence, the length of the G-field is

$$g = \lceil \log_2(q + \log_2 n) \rceil. \tag{25}$$

Let $\acute{q}_{min}$, (where $\acute{q}_{min} \leq 2^g$) correspond to the smallest length of the exponent of the *TFP* system that achieves the exponent range at least as good as that of the fixed floating point, therefore

$$\begin{aligned} \acute{q}_{min} &= \lceil q + \log_2 n \rceil \\ &= q + \lceil \log_2 n \rceil. \end{aligned} \tag{26}$$

For example, if the ER of the $(8, 8, 23)_f$ fixed floating point is to be matched, then the length of the G-field $g = 4$ bits, though we can have exponents up to 16 bits long, $\acute{q}_{min} = 10$ bits are sufficient to achieve the exponent range.

To compare the maximum relative representation error $MRRE$ of both systems, given that ER is to be matched, we will use the ratio ρ as the ratio of the $MRRE$ of the *TFP* system to that of the fixed floating point. The $MRRE$ of the *TFP* system is based on the length of the exponent not to exceed $\acute{q}_{min}$ bits, this gives

$$\begin{aligned} \rho &= \frac{MRRE\text{of}(2, \acute{q}_{min}, 32-g-\acute{q}_{min})_f TFP\text{system}}{MRRE\text{of}(2^n, q, 31-q)_f\text{fixed floating point}} \\ &= 2^{g+(\acute{q}_{min}-q)-n} \\ &= 2^{g+\lceil \log_2 n \rceil - n}. \end{aligned} \tag{27}$$

From equation (28) it is clear that $\rho > 1$ means that the $MRRE$ of the *TFP* system is inferior to that of fixed floating point, and the larger the worse it is. Table 5. tabulates ρ for different *TFP* systems that matches the exponent range of the corresponding fixed floating point. Note that in our case the $MRRE$ of the *TFP* system is always inferior than that of a fixed floating point.

Table 5. The ratio of ρ of the $MRRE$ of a *TFP* system to that of fixed floating point, both share the size & the ER

fixed FP	corres. TFP	ρ
$(4, 7, 24)_f$	$(2, 3, 29)_t$	2^2
$(8, 8, 23)_f$	$(2, 4, 28)_t$	2^3
$(16, 7, 24)_f$	$(2, 4, 28)_t$	2^2

If the average relative representation error $ARRE$ of the *TFP* system should be as least as good as that of fixed float-

[‡] Subscripts t and s denote tapered and single precision respectively.

ing point, given that ER is to be matched. Then, we have the following condition for the P_{avg}

$$P_{\text{avg}} = 31 + \log_2(\frac{n}{2^n - 1}) - q. \quad (28)$$

In the *TFP* system, the P_{avg} depends on the standard deviation σ of the normally distributed exponents. Table 6. shows the standard deviation needed so that the ER and the $ARRE$ of both systems match. For example, the $ARRE$ of the $(2,3,29)_t$ *TFP* system is same as that of $(4,7,24)_f$ fixed floating point if $\sigma \simeq 31$, and if $\sigma < 31$ we can expect a better $ARRE$ for the *TFP* system, and worse if $\sigma > 31$.

Table 6. The standard deviation σ, such that the $ARRE$ of both systems is the same, given that ER is matched.

fixed FP	corres. TFP	Std dev σ
$(4,7,24)_f$	$(2,3,29)_t$	31
$(8,8,23)_f$	$(2,4,28)_t$	49
$(16,7,24)_f$	$(2,4,28)_t$	39

From Tables 5. and 6. we conclude that given a *TFP* system that matches the exponent range of a fixed floating point, it is possible that the *TFP* will match the $ARRE$ of a fixed floating point provided that the exponents are distributed in such a way so that P_{avg} satisfies equation (27), but in no way the *TFP* will match the $MRRE$ of any fixed floating point.

Let q be the length of the exponent corresponding to γ in the G-field. We can think of the relation between γ and q as a mapping $\Phi : \gamma \rightarrow q$. In this work we tackled the *TFP* system where the contents of the G-field is one less than the length of the exponent, that is $\Phi : \gamma \rightarrow \gamma + 1$. In fact there exist a whole family of *TFP* systems depending on the mapping Φ. There is no reason to restrict ourselves to only linear mapping. Nonlinear mapping is also applicable. For example, if we used the mapping $\Phi : \gamma \rightarrow 2^\gamma$, then we can still have exponents up to 8 bits long with only two bits allocated to the G-field. Smaller G-field means better $MRRE$, but the effect on the $ARRE$ needs further research.

It turns out that for any non-binary fixed floating point system there exists a binary *TFP* system that can match the ER and the $MRRE$ and have a better $ARRE$. For example, the $(2,2,30)_t$ *TFP* system can match the ER and $MRRE$ of the $(16,7,24)_f$ fixed floating point system, provided that the largest exponent of the *TFP* system is 9 bits long. A possible scheme for the case is $\Phi : \gamma \rightarrow 2\gamma + 3$, another valid mapping is $\Phi : \gamma \rightarrow 2^\gamma + 1$. The choice of the mapping scheme that has the same upper bound on the exponent length and that occupies the same space for the G-field, needs further investigation.

6 Conclusions

The main reasons behind the TFP can be summarized as follows:

- In a fixed floating point, exponents occupy the same space regardless of their value, which could have been used to get higher precision [Mor71].
- Most applications that require the greatest accuracy are not likely to generate numbers of extreme magnitude.
- Exponents mostly follow the Gaussian distribution [Ham70].

The overhead in the *TFP* system is in the introduction of an extra field that does the housekeeping job, and some additional hardware in the arithmetic unit. Furthermore the pack and unpack operation will affect the speed of performing arithmetic operations. Though the original system suggested by Morris' has an inferior $MRRE$ when compared to a fixed floating point, it has a somewhat promising $ARRE$. Nevertheless there are *TFP* systems that use a different mapping scheme that could match the $MRRE$ and the ER of a fixed floating point of radix $r > 2$, and in this case the *TFP* will have a superior $ARRE$.

The proposed method is a very simple extension of [Mor71]; however its practicality both for VLSI implementation and precision characteristics, makes it viable when compared with elegant but more complex approaches [Mat81], [Olv87B].

References

[At83] L.V. Atkinson, *et al*, "An Introduction to Numerical Methods with Pascal," *Addison-Wesley*, 1983.

[Az87] A.M. Azmi, "A Floating Point System with Variable Length Exponent," *Masters Thesis, Univ. of Colorado*, May 1987.

[Bro69] W.S. Brown, *et al*, "The Choice of Base," *Comm. ACM*, Vol. 12, Oct. 1969, pp. 560-561.

[Cod73] W.J. Cody, Jr., "Static and Dynamic Numerical Characteristics of Floating Point Arithmetic," *IEEE Trans. Comput.*, Vol. C-22, Jun. 1973, pp. 598-601.

[Ham70] R.W. Hamming, "On the Distribution of Numbers," *Bell Syst. Tech. J.*, Vol. 49, Oct. 1970, pp. 1609-1625.

[Hu52] D.A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," *Proc. of IRE*, Vol. 40, Sep. 1952, pp. 1098-1101.

[Hw79] K. Hwang, "Computer Arithmetic: Principles, Architecture, and Design," *John Wiley*, 1979.

[Ko83] P. Kornerup, *et al*, "Finite Precision Rational Arithmetic: An Arithmetic Unit," *IEEE Trans. Comput.*, Vol. C-32, April 1983, pp. 378-387.

[Mat81] S. Matsui and M. Iri, "An Overflow/Underflow-Free Floating Point Representation of Numbers," *J. of Inf. Proc.*, Vol. 4, No. 3, 1981, pp. 123-133.

[McK67] W.M. McKeeman, "Representation Error for Real Numbers in Binary Computer Arithmetic," *IEEE Trans. Electron. Comput.*, Vol. EC-16, Oct. 1967, pp. 682-683.

[Mor71] R. Morris, "Tapered Floating Point: A New Floating-Point Representation," *IEEE Trans. on Comput.*, Vol. TC-20, Dec. 1971, pp. 1578-1579.

[Olv87A] F.W.J. Olver, "A Closed Computer Arithmetic," *Proc. IEEE Arith. Symp.*, 1987, pp. 139-143.

[Olv87B] F.W.J. Olver and P.R. Turner, "Implementation of Level-Index Arithmetic Using Partial Table Look-Up,"

Proc. IEEE Arith. Symp., 1987, pp. 144-147.

[Sc85] N.R. Scott, "Computer Number Systems & Arithmetic," *Prentice Hall*, 1985.

[Sw65] D.W. Sweeney, "An Analysis of Floating-Point Addition," *IBM Syst. J.* Vol. 4, 1965, pp. 31-42.

[Th79] G.B. Thomas, Jr., "Calculus and Analytical Geometry," *Addison-Wesley*, 4th edition, 1977.

[Wa82] S. Waser, *et al* "Introduction to Arithmetic for Digital Systems Designers," *CBS College Publishing*, 1982.

[Wilk63] J.H. Wilkinson, "Rounding Errors in Algebraic Process," N.J., *Prentice-Hall*, 1963.

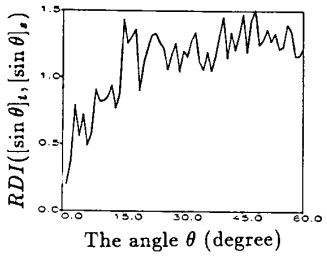

Figure 7. $RDI\,([\sin\theta]_t, [\sin\theta]_s)$, $w = 29$ bits.

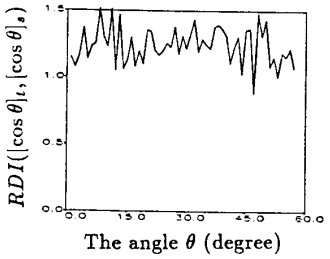

Figure 8. $RDI\,([\cos\theta]_t, [\cos\theta]_s)$, $w = 29$ bits.

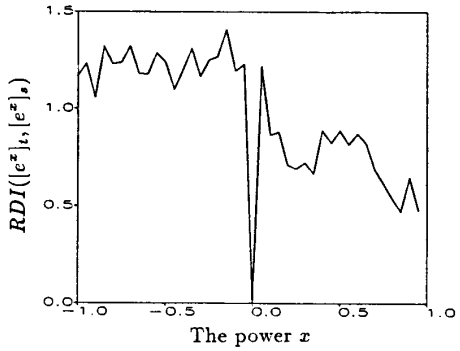

Figure 9. $RDI\,([e^x]_t, [e^x]_s)$, $w = 29$ bits sampled with step size $x = .05$.

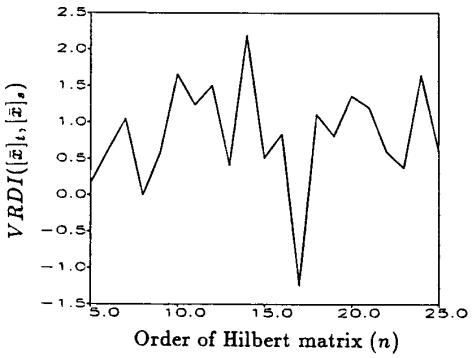

Figure 10. $VRDI\,([\bar{x}]_t, [\bar{x}]_s)$ of Hilbert linear system of order n.