ReproBLAS: Much Summation! So Reproducible!

Peter Ahrens, Hong Diep Nguyen, James Demmel

July 31, 2015

Contents

1 Introduction

2 Notation and Background

Let $\mathbb{R}, \mathbb{F}$, and $\mathbb{Z}$ denote the sets of real numbers, floating-point numbers, and integers respectively.

For all $r \in \mathbb{R}$, let $r\mathbb{Z}$ denote the set of all multiples of r, $\{rz | z \in \mathbb{Z}\}$.

For all $r \in \mathbb{R}$, let $\lfloor r \rfloor$ be the minimum element $z \in \mathbb{Z}$ such that $z \geq r$.

For all $r \in \mathbb{R}$, let $\lceil r \rceil$ be the maximum element $z \in \mathbb{Z}$ such that $z \leq r$.

We define the function $\mathcal{R}_\infty(r, e), r \in R, e \in \mathbb{Z}$ as

$$\mathcal{R}_\infty(r, e) = \begin{cases} \lfloor r/2^e + 1/2 \rfloor \, 2^e \text{ if } r \geq 0 \\ \\ \lceil r/2^e - 1/2 \rceil \, 2^e \text{ otherwise} \end{cases} \tag{2.1}$$

$\mathcal{R}_\infty(r, e)$ rounds r to the nearest multiple of 2^e, breaking ties away from 0. A property of such rounding is shown in (2.2)

$$|r - \mathcal{R}_\infty(r, e)| \leq 2^{e-1} \tag{2.2}$$

Assume that floating point arithmetic complies with the IEEE 754-2008 standard [2] in some "to nearest" rounding mode (no specific tie breaking behaviour is required) and that underflow occurs gradually, although methods to handle abrupt underflow will be considered in Section 4.1.3.

Let $f = sm2^e \in \mathbb{F}$ be a floating-point number represented in IEEE 754-2008 format [2] where $s \in \{1, -1\}$ is the **sign**, $e_{\max} \geq e \geq e_{\min}$ is the **exponent** ($\exp(f)$ is defined to be e), p is the **precision**, and $m = m_0.m_1 m_2...m_{p-1}$ where $m_0, ..., m_{p-1} \in \{0, 1\}$ is the **significand** of f. f is said to be **normalized** if $m_0 = 1$ and $e \geq e_{\min}$, **unnormalized** if $m_0 = 0$, and **denormalized** if $m_0 = 0$ and $e = e_{\min}$. $f = 0$ if all $m_k = 0$ and $e = e_{\min}$.

$r \in \mathbb{R}$ is **representable** as a floating point number if there exists $f \in \mathbb{F}$ such that $r = f$ as real numbers.

For all $r \in \mathbb{R}$, $e \in \mathbb{Z}$ such that $e_{\min} - p < e$ and $|r| < 2 \cdot 2^{e_{\max}}$, if $r \in 2^e \mathbb{Z}$ and $r \leq 2^{e+p}$ then r is representable.

For all $f_0, f_1 \in \mathbb{F}$, $\text{fl}(f_0 \text{ op } f_1)$ denotes the evaluated result of the expression $(f_0 \text{ op } f_1)$ in floating point arithmetic. If $(f_0 \text{ op } f_1)$ is representable, then $\text{fl}(f_0 \text{ op } f_1) = (f_0 \text{ op } f_1)$.

Machine epsilon, ϵ, the difference between 1 and the greatest floating point number smaller than 1, is defined as $\epsilon = 2^{-p}$.

The unit in the last place of $f \in \mathbb{F}$, $\mathrm{ulp}(f)$, is the spacing between two consecutive floating point numbers of the same exponent as f. If f is normalized, $\mathrm{ulp}(f) = 2^{\exp(f)-p+1} = 2\epsilon 2^{\exp(f)}$.

The unit in the first place of $f \in F$, $\mathrm{ufp}(f)$, is the value of the first significant bit of f. If f is normalized, $\mathrm{ufp}(f) = 2^{\exp(f)}$.

As ReproBLAS is written in C, `float` and `double` refer to the floating point types specified in the 1989 C standard [3] and we assume that they correspond to the `binary-32` and `binary-64` types in the IEEE 754-2008 floating point standard [2].

All indices start at 0 in correspondence with the actual ReproBLAS implementation.

3 Binning

We achieve reproducible summation of floating point numbers through binning. Each number is split into several components corresponding to predefined ranges, then the components corresponding to each range are accumulated separately. We begin in Section 3.1 by explaining the particular set of ranges (referred to as bins) used. Section 3.2 develops original mathematical theory to describe the components (referred to as slices) corresponding to each range. We develop this theory to concisely describe and prove correctness of algorithms throughout the paper (especially Algorithms 5.3 and 5.4).

3.1 Bins

We start by dividing the exponent range $(e_{\min} - p, ..., e_{\max} + 1]$ into **bins** $(a_i, b_i]$ of **width** W according to (3.1), (3.2), and (3.3). Such a range is used so that the largest and smallest (denormalized) floating point numbers may be approximated.

$$i_{\max} = \lfloor (e_{\max} - e_{\min} + p - 1)/W \rfloor - 1 \tag{3.1}$$

$$a_i = e_{\max} + 1 - (i + 1)W, \ 0 \le i \le i_{\max} \tag{3.2}$$

$$b_i = a_i + W \tag{3.3}$$

We say the bin $(a_{i_0}, b_{i_0}]$ is **greater** than the bin $(a_{i_1}, b_{i_1}]$ if $a_{i_0} > a_{i_1}$ (which is equivalent to both $b_{i_0} > b_{i_1}$ and $i_0 < i_1$).

We say the bin $(a_{i_0}, b_{i_0}]$ is **lesser** than the bin $(a_{i_1}, b_{i_1}]$ if $a_{i_0} < a_{i_1}$ (which is equivalent to both $b_{i_0} < b_{i_1}$ and $i_0 > i_1$).

We use $i \leq i_{\max} = \lfloor (e_{\max} - e_{\min} + p - 1)/W \rfloor - 1$ to ensure that $a_i > e_{\min} - p + 1$ as discussed in Section 4.1.2. This means that the greatest bin is

$$(e_{\max} + 1 - W, e_{\max} + 1] \tag{3.4}$$

and the least bin is

$$(e_{\min}-p+2+((e_{\max}-e_{\min}+p-1) \mod W), e_{\min}-p+2+W+((e_{\max}-e_{\min}+p-1) \mod W)] \tag{3.5}$$

Section 4.1.2 explains why the range

$$(e_{\min} - p, e_{\min} - p + 2 + ((e_{\max} - e_{\min} + p - 1) \mod W)]$$

is ignored.

As discussed in [1], we must assume

$$W < p - 2 \tag{3.6}$$

As discussed in Section 4.1.1, we must also assume

$$2W > p + 1 \tag{3.7}$$

ReproBLAS uses both `float` and `double` floating point types. The chosen division of exponent ranges for both types is shown in Figure 3.1.

Floating-Point Type	`float`	`double`
$e_{\max}$	127	1023
$e_{\min}$	-126	-1022
p	24	53
$e_{\min} - p$	-140	-1075
W	13	40
$i_{\max}$	19	51
$(a_0, b_0]$	$(115, 128]$	$(984, 1024]$
$(a_{i_{\max}}, b_{i_{\max}}]$	$(-132, -119]$	$(-1056, -1016]$

Figure 3.1: ReproBLAS Binning Scheme

3.2 Slices

Throughout the text we will refer to the **slice** of some $x \in \mathbb{F}$ in the bin $(a_i, b_i]$. x can be split into several slices, each slice corresponding to a bin $(a_i, b_i]$ and expressible as the (possibly negated) sum of a subset of $\{2^e, e \in (a_i, b_i]\}$, such that the sum of the slices provides a good approximation of x. Specifically, the slice of $x \in \mathbb{F}$ in the bin $(a_i, b_i]$ is defined recursively as $d(x, i)$ in (3.8). We must define $d(x, i)$ recursively because it is not a simple bitwise extraction.

$$d(x, i) = \mathcal{R}_\infty(x - \sum_{k=0}^{i-1} d(x, k), a_i + 1) \tag{3.8}$$

We make three initial observations on the definition of $d(x, i)$. First, we note that $d(x, i)$ is well defined recursively on i with base case $d(x, 0) = \mathcal{R}_\infty(x, a_0 + 1)$.

Next, notice that $d(x, i) \in 2^{a_i + 1}\mathbb{Z}$.

Finally, it is possible that $d(x, 0)$ may be too large to represent as a floating point number. Overflow of this type is accounted for in Section 4.1.1.

Lemmas 3.1 and 3.2 follow from the definition of $d(x, i)$.

Lemma 3.1. For all $i \in \{0, ..., i_{\max}\}$ and $x \in \mathbb{F}$ such that $|x| < 2^{a_i}$, $d(x, i) = 0$

Proof. We show the claim by induction on i.

In the base case, we have $|x| < 2^{a_0}$ and $d(x,0) = \mathcal{R}_\infty(x, a_0 + 1)$. By the triangle inequality and (2.2),

$$|\mathcal{R}_\infty(x, a_0 + 1)| \leq |x| + |x - \mathcal{R}_\infty(x, a_0 + 1)| < 2^{a_0+1}$$

Since $\mathcal{R}_\infty(x, a_0 + 1) \in 2^{a_0+1}\mathbb{Z}$, we then have

$$d(x, 0) = \mathcal{R}_\infty(x, a_0 + 1) = 0$$

In the inductive step, we have $|x| < 2^{a_{i+1}} \leq ... \leq 2^{a_0}$ by (3.2) and by induction $d(x, i) = ... = d(x, 0) = 0$. Thus,

$$d(x, i+1) = \mathcal{R}_\infty(x - \sum_{k=0}^{i} d(x,k), a_{i+1} + 1) = \mathcal{R}_\infty(x, a_{i+1} + 1)$$

Again by the triangle inequality and (2.2),

$$|\mathcal{R}_\infty(x, a_{i+1} + 1)| \leq |x| + |x - \mathcal{R}_\infty(x, a_{i+1} + 1)| < 2^{a_{i+1}+1}$$

Since $\mathcal{R}_\infty(x, a_{i+1} + 1) \in 2^{a_{i+1}+1}\mathbb{Z}$, we then have

$$d(x, i+1) = \mathcal{R}_\infty(x, a_{i+1} + 1) = 0$$

$\square$

Lemma 3.2. For all $i \in \{0, ..., i_{\max}\}$ and $x \in \mathbb{F}$ such that $|x| < 2^{b_i}$, $d(x, i) = \mathcal{R}_\infty(x, a_i + 1)$.

Proof. The claim is a simple consequence of Lemma 3.1.

By (3.2) and (3.3), $|x| < 2^{b_i} = 2^{a_{i-1}} \leq ... \leq 2^{a_0}$. Therefore Lemma 3.1 implies $d(x, 0) = ... = d(x, i - 1) = 0$ and we have

$$d(x, i) = \mathcal{R}_\infty(x - \sum_{k=0}^{i-1} d(x,k), a_i + 1) = \mathcal{R}_\infty(x, a_i + 1)$$

$\square$

Lemma 3.1, Lemma 3.2, and (3.8) can be combined to yield an equivalent definition of $d(x, i)$ for all $i \in \{0, ..., i_{\max}\}$ and $x \in \mathbb{F}$.

$$d(x, i) = \begin{cases} 0 \text{ if } |x| < 2^{a_i} \\ \mathcal{R}_\infty(x, a_i + 1) \text{ if } 2^{a_i} \le |x| < 2^{b_i} \\ \mathcal{R}_\infty(x - \sum_{k=0}^{i-1} d(x, k), a_i + 1) \text{ if } 2^{b_i} \le |x| \end{cases} \tag{3.9}$$

Theorem 3.3 shows that sum of the slices of $x \in \mathbb{F}$ provides a good approximation of x.

Theorem 3.3. For all $i \in \{0, ..., i_{\max}\}$ and $x \in \mathbb{F}$, $|x - \sum_{k=0}^{i} d(x, k)| \le 2^{a_i}$.

Proof. We apply (2.2) and (3.9)

$$\left| x - \sum_{k=0}^{i} d(x, k) \right| = \left| (x - \sum_{k=0}^{i-1} d(x, k)) - d(x, i) \right|$$

$$= \left| (x - \sum_{k=0}^{i-1} d(x, k)) - \mathcal{R}_\infty(x - \sum_{k=0}^{i-1} d(x, k), a_i + 1) \right| \le 2^{a_i}$$

$\square$

Theorem 3.4 shows a bound on $d(x, i)$.

Theorem 3.4. For all $i \in \{0, ..., i_{\max}\}$ and $x \in \mathbb{F}$, $|d(x, i)| \le 2^{b_i}$.

Proof. First, we show that $|x - \sum_{k=0}^{i-1} d(x, k)| \le 2^{b_i}$.

If $i = 0$, then we have

$$\left| x - \sum_{k=0}^{i-1} d(x, k) \right| = |x| < 2 \cdot 2^{e_{\max}} < 2^{b_0}$$

Otherwise, we can apply (3.2) and (3.3) to Theorem 3.3 to get

$$\left| x - \sum_{k=0}^{i-1} d(x, k) \right| \le 2^{a_{i-1}} = 2^{b_i}$$

As $2^{b_i} \in 2^{a_i+1}\mathbb{Z}$, (3.8) can be used

$$|d(x, i)| = \left| \mathcal{R}_\infty(x - \sum_{k=0}^{i-1} d(x, k), a_i + 1) \right| \le 2^{b_i}$$

$\square$

4   The Indexed Type

The **indexed type** is used to represent the intermediate result of accumulation using Algorithms 6 and 7 in [1]. An indexed type Y is a data structure composed of several accumulator data structures $Y_0, ..., Y_{K-1}$. An indexed type with K accumulators is referred to as a K-**fold** indexed type. Due to their low accuracy, 1-fold indexed types are not considered.

Let Y be the indexed type corresponding to the reproducibly computed sum of $x_0, ..., x_{n-1} \in \mathbb{F}$. Each accumulator Y_j is a data structure used to accumulate the slices of input in the bin $(a_{I+j}, b_{I+j}]$ where I is the **index** of Y and $j \geq 0$ as discussed below. The **width** of an indexed type is equal to the width of its bins, W. Recall the assumptions (3.6) and (3.7) made on the value of W.

The accumulators in an indexed type correspond to contiguous bins in decreasing order. The index of Y is defined as the least $I \in \mathbb{Z}$ such that $2^{b_I+1} > \max(|x_0|, ..., |x_n|)$ (equivalently, the least $I \in \mathbb{Z}$ such that $b_I > \max(e_0, ..., e_n)$). If Y has index I, then $Y_j, j \in \{0, ..., K-1\}$ accumulates slices of input in the bin $(a_{I+j}, b_{I+j}]$. If I is so large that $I + K > i_{\max}$, then the extra $I + K - i_{\min}$ accumulators are unused.

Section 4.1 elaborates on the specific fields that make up the indexed type and the values they represent. Sections 4.1.1, 4.1.2, 4.1.3, and 4.1.4 contain original extensions of the indexed type to handle overflow, underflow, and exceptional values.

4.1   Primary and Carry

As discussed in [1], indexed types are represented using floating point numbers to minimize traffic between floating point and integer arithmetic units.

In the ReproBLAS library, if an indexed type is used to sum `doubles`, then it is composed entirely of `doubles` and likewise for `floats`. ReproBLAS supports complex types as pairs of real and imaginary components (stored contiguously in memory). If an indexed type is used to sum complex `doubles` or `floats`, then it is composed of pairs (real part, imaginary part) of `doubles` or `floats` respectively. The decision to keep the real and imaginary components together (as opposed to keeping separate indexed types for real and imaginary parts of the sum) was motivated by a desire to process accumulators simultaneously with vectorized (SIMD) instructions.

The accumulators Y_j of an indexed type Y are each implemented using

two underlying floating point fields. The **primary** field Y_{j_P} is used during accumulation, while the **carry** field Y_{j_C} holds overflow from the primary field. Because primary fields are frequently accessed sequentially, the primary fields and carry fields are each stored contiguously in separate arrays. The notation for the primary field Y_{j_P} and carry field Y_{j_C} corresponds to the "S_k" and "C_k" of Algorithm 6 in [1].

The numerical value $\mathcal{Y}_{j_P}$ represented by data stored in the primary field Y_{j_P} is an offset from $1.5\epsilon^{-1}2^{a_{I+j}}$ (corresponding to "$M_{[i]}$" at the beginning of Section IV.A. in [1]), where I is the index of Y, as shown in (4.1) below.

$$\mathcal{Y}_{j_P} = Y_{j_P} - 1.5\epsilon^{-1}2^{a_{I+j}} \tag{4.1}$$

This simplifies the process of extracting the slices of input in bins $(a_{I+j}, b_{I+j}]$. It will be shown in Theorem 5.1 that if we represent each primary value $\mathcal{Y}_{j_P}$ as in (4.1) and keep Y_{j_P} within the range $(\epsilon^{-1}2^{a_{I+j}}, 2\epsilon^{-1}2^{a_{I+j}})$, then Algorithm 5.3 extracts the slices of x in bins $(a_I, b_I], ..., (a_{I+K-1}, b_{I+K-1}]$ and adds them to $Y_{0_P}, ..., Y_{K-1_P}$ without error (and hence reproducibly) for all $x \in \mathbb{F}$, where $|x| < 2^{b_I}$.

Because $d(x, I+j) = 0$ for bins with $x < 2^{a_{I+j}}$, the values in the greatest K nonzero accumulators can be computed reproducibly by computing the values in the greatest K accumulators needed for the largest x seen so far. Upon encountering an $x \geq 2^{b_I}$, the accumulators can then be shifted towards index 0 as necessary. [1] contains further discussions of the reproducibility of the algorithm.

In order to keep the primary fields in the necessary range while the slices are accumulated and to keep the representation of Y_j unique, Y_{j_P} is routinely renormalized to the range $[1.5\epsilon^{-1}2^{a_{I+j}}, 1.75\epsilon^{-1}2^{a_{I+j}})$. Section 5.4 contains a discussion of how often such renormalizations are necessary. To renormalize, Y_{j_P} is incremented or decremented by $0.25\epsilon^{-1}2^{a_{I+j}}$ when necessary, leaving the carry field Y_{j_C} to record the number of such adjustments. The numerical value $\mathcal{Y}_{j_C}$ represented by data stored in the carry field Y_{j_C} of an indexed type Y of index I is expressed in (4.2)

$$\mathcal{Y}_{j_C} = (0.25\epsilon^{-1}2^{a_{I+j}})Y_{j_C} \tag{4.2}$$

Combining (4.1) and (4.2), we get that the value $\mathcal{Y}_j$ of the accumulator Y_j of an indexed type Y of index I is

$$\mathcal{Y}_j = \mathcal{Y}_{j_P} + \mathcal{Y}_{j_C} = (Y_{j_P} - 1.5\epsilon^{-1}2^{a_{I+j}}) + (0.25\epsilon^{-1}2^{a_{I+j}})Y_{j_C} \tag{4.3}$$

Therefore, using (4.3), the numerical value $\mathcal{Y}$ represented by data stored in a K-fold indexed type Y of index I (the sum of Y's accumulators) is

$$\mathcal{Y} = \sum_{j=0}^{K-1} \mathcal{Y}_j = \sum_{j=0}^{K-1}((Y_{jP} - 1.5\epsilon^{-1}2^{a_{I+j}}) + (0.25\epsilon^{-1}2^{a_{I+j}})Y_{jC}) \qquad (4.4)$$

It is worth noting here that because the primary field Y_{0P} is stored with an exponent of $a_I + p$, it is unnecessary to store the index of an indexed type explicitly. The index can be determined by simply examining the exponent of Y_{0P}, as all a_I are distinct and the mapping between the exponent of Y_{0P} and the index of Y is bijective.

4.1.1   Overflow

If an indexed type Y has index 0 and the width is W, then the value in the primary field Y_{0P} is stored as an offset from $1.5\epsilon^{-1}2^{e_{\max}+1-W}$. However, $1.5\epsilon^{-1}2^{e_{\max}+1-W} > 2^{e_{\max}+1+(p-W)} > 2 \cdot 2^{e_{\max}}$ since $W < p - 2$ [1], so it is out of the range of the floating-point system and not representable. Before discussing the solution to this overflow problem, take note of Theorem 4.1.

Theorem 4.1. If $2W > p + 1$, then for any indexed type Y of index I and any Y_{jP} such that $I + j \geq 1$, $|Y_{jP}| < 2^{e_{\max}}$.

Proof. $a_1 = e_{\max} + 1 - 2W$ by definition, therefore $a_1 < e_{\max} - p$ and since all quantities are integers, $a_1 \leq e_{\max} - p - 1$. As $a_0, a_1, ...$ is a positive decreasing sequence, $a_{I+j} \leq e_{\max} - p - 1$ since $I + j \geq 1$.

Y_{jP} is kept within the range $(\epsilon^{-1}2^{a_{I+j}}, 2\epsilon^{-1}2^{a_{I+j}})$, therefore

$$\left|Y_{jP}\right| < 2\epsilon^{-1}2^{a_{I+j}} \leq 2^{1+p}2^{e_{\max}-1-p} = 2^{e_{\max}}$$

$\square$

By Theorem 4.1, if $2W > p + 1$ then the only primary field that could possibly overflow is a primary field corresponding to bin 0, and all other primary fields have exponent less than $e_{\max}$. Therefore, we impose $2W > p+1$ and express the value of the primary field corresponding to bin 0 as a scaled offset from $1.5 \cdot 2^{e_{\max}}$. Note that this preserves uniqueness of the exponent of the primary field corresponding to bin 0 because no other primary field has an exponent of $e_{\max}$. The value $\mathcal{Y}_{0P}$ stored in the primary field Y_{0P} of an indexed type Y of index 0 is expressed in (4.5).

$$\mathcal{Y}_{0P} = 2^{p-W+1}(Y_{0P} - 1.5 \cdot 2^{e_{\max}}) \qquad (4.5)$$

4.1.2 Gradual Underflow

Here we consider the effects of gradual underflow on algorithms described in [1] and how the indexed type allows these algorithms to work correctly.

Algorithms 5.4 and 5.5 require that the primary fields Y_{jP} are normalized to work correctly. Theorem 4.2 shows that the primary fields should always be normalized.

Theorem 4.2. For any primary field Y_{jP} of an indexed type Y of index I where $Y_{jP} \in (\epsilon^{-1} 2^{a_{I+j}}, 2\epsilon^{-1} 2^{a_{I+j}})$ ($Y_{0P} \in (2^{e_{\max}}, 2 \cdot 2^{e_{\max}})$ if Y has index 0), Y_{jP} is normalized.

Proof. The least bin is

$$(e_{\min}-p+2+((e_{\max}-e_{\min}+p-1) \mod W), e_{\min}-p+2+W+((e_{\max}-e_{\min}+p-1) \mod W)]$$

Because $Y_{jP} \in (\epsilon^{-1} 2^{a_{I+j}}, 2\epsilon^{-1} 2^{a_{I+j}})$ and

$$a_{I+j} \geq e_{\min} - p + 2 + ((e_{\max} - e_{\min} + p - 1) \mod W) \geq e_{\min} - p + 2$$

we have $\exp(Y_{jP}) = a_{I+j} + p > 2^{e_{\min}+1}$ so Y_{jP} is normalized. $\qquad \square$

Algorithm 5.4 relies on setting the last bit of intermediate results before adding them to Y_{jP} in order to fix the direction of the rounding mode. However, if r is the quantity to be added to Y_{jP}, $\text{ulp}(r)$ must be less than rounding error when added to Y_{jP}. Mathematically, $\text{ulp}(r) < \frac{1}{2}\text{ulp}(Y_{jP})$. This is why we must enforce $a_{i_{\max}} \geq e_{\min} - p + 2$

Although the bins do not extend all the way to $e_{\min} - p$, it can still be shown that the sum of the slices of some $x \in \mathbb{F}$ offer a good approximation of x.

Since $W < p - 2$,

$$a_{i_{\max}} = e_{\min} - p + 2 + ((e_{\max} - e_{\min} + p - 1) \mod W) \leq e_{\min} - 1$$

As a consequence, we can use Theorem 3.3 to say that for any $x \in \mathbb{F}$,

$$\left| x - \sum_{i=0}^{i_{\max}} d(x, i) \right| \leq 2^{e_{\min}-1} \tag{4.6}$$

This means that we can approximate x using the sum of its slices to within $2^{e_{\min}-1}$.

14

It is possible to sum the input in the denormalized range. One simple way the algorithm could be extended to denormalized inputs would be to scale the least bins up, analogously to the way we handled overflow. Due to the relatively low priority for accumulating denormalized values, this method was not implemented in ReproBLAS.

4.1.3 Abrupt Underflow

If underflow is abrupt, several approaches may be taken to modify the given algorithms to ensure reproducibility.

The simplest approach would be to accumulate input in the denormalized range by scaling the smaller inputs up. This has the added advantage of increasing the accuracy of the algorithm. A major disadvantage to this approach is the additional branching cost incurred due to the conditional scaling.

A more efficient way to solve the problem would be to set the least bin to have $a_{i_{\max}} = 2^{e_{\min}}$. This could be accomplished either by keeping the current binning scheme and having the least bin be of a width not necessarily equal to W, or by shifting all other bins to be greater. The disadvantage of shifting the other bins is that it may cause multiple greatest bins to overflow, adding multiple scaling cases. Setting such a least bin would enforce the condition that no underflow occurs since all intermediate sums are either 0 or greater than the underflow threshold. The denormal range would be discarded.

In the case that reproduciblity is desired on heterogeneous machines, where some processors may handle underflow gradually and others abruptly, the approach of setting a least bin is reccomended. The indexed sum using this scheme does not depend on whether or not underflow is handled gradually or abruptly, so the results will be the same regardless of where they are computed.

4.1.4 Exceptions

Indexed types are capable of representing exceptional cases such as `NaN` (Not a Number) and `Inf` (Infinity). An indexed type Y stores its exception status in its first primary field Y_{0P}.

A value of 0 in Y_{0P} indicates that nothing has been added to Y_{0P} yet (Y_{0P} is initialized to 0).

The least bin is

$$(e_{\min}-p+2+((e_{\max}-e_{\min}+p-1) \mod W), e_{\min}-p+2+W+((e_{\max}-e_{\min}+p-1) \mod W)]$$

and since the Y_{jP} are kept within the range $(\epsilon^{-1}2^{a_{I+j}}, 2\epsilon^{-1}2^{a_{I+j}})$ (where I is the index), the least possible Y_{jP} is

$$\epsilon^{-1}2^{e_{\min}-p+2+((e_{\max}-e_{\min}+p-1) \mod W)} > 2^{e_{\min}}$$

Therefore the value of 0 in a primary field is unused in any previously specified context and may be used as a sentinel value. (As the exponent of 0 is distinct from the exponent of normalized values, the bijection between the index of an indexed type Y and the exponent of Y_{0P} is preserved)

A value of `Inf` or `-Inf` in Y_{0P} indicates that one or more `Inf` or `-Inf` (and no other exceptional values) have been added to Y respectively.

A value of `NaN` in Y_{0P} indicates that one or more `NaN` have been added to Y or one or more of both `Inf` and `-Inf` have been added to Y.

As the Y_{jP} are kept finite to store finite values, `Inf`, `-Inf`, and `NaN` are unused in any previously specified context and are valid sentinel values. (As the exponent of `Inf`, `-Inf`, and `NaN` is distinct from the exponent of finite values, the bijection between the index of an indexed type Y and the exponent of Y_{0P} is preserved)

This behavior follows the behavior for exceptional values in IEEE 754-2008 floating point arithmetic. The result of adding some exceptional values using floating-point arithmetic therefore matches the result obtained from indexed summation. As `Inf`, `-Inf`, and `NaN` add associatively, this behavior is reproducible.

It should be noted here that it is possible to achieve a final result of `Inf` or `-Inf` when Y_{0P} is finite. This is due to the fact that the indexed representation can express values outside of the range of the floating point numbers that it is composed with. More specifically, it is possible for the value $\mathcal{Y}$ represented by the indexed type Y to satisfy $|\mathcal{Y}| \geq 2 \cdot 2^{e_{\max}}$. The condition that $\mathcal{Y}$ is not representable is discovered when calculating $\mathcal{Y}$ (converting Y to a floating point number). The methods used to avoid overflow and correctly return the `Inf` or `-Inf` are discussed in Section 5.7.

5   Primitive Operations

5.1   Section Summary

Here we reorganize algorithms in [1] into a user-friendly set of primitive operations on an indexed type. Two simple original algorithms relating to the index of an indexed type are given in Section 5.2. Theoretical summaries of algorithms (with some improvements) from [1] are provided in Sections 5.5, 5.3, 5.4, 5.6. Section 5.7 provides an original algorithm (with an improved error bound) to obtain the value represented by an indexed type. Sections 5.8 and 5.9 extend analysis in [1] to the new algorithms.

5.2   Index

When operating on indexed types it is sometimes necessary to compute their index. Algorithm 5.1 yields the index of an indexed type in constant time.

Algorithm 5.1. Given an indexed type Y, calculate its index I
Require:
   $Y_{0P} \in (\epsilon^{-1} 2^{a_I}, 2\epsilon^{-1} 2^{a_I})$
 1: **function** IINDEX(Y, I)
 2:     $I = \lfloor (e_{\max} + p - \exp(Y_{0P}) - W + 1)/W \rfloor$
 3: **end function**
Ensure:
   I is the index of Y.

Note that the floor function is necessary in Algorithm 5.1 to account for the case of Y with index 0, which has $\exp(Y_{0P}) = 2^{e_{\max}}$ as discussed in Section 4.1.1.

Another useful operation is, given some $x \in \mathbb{F}$, to find the unique bin $(a_J, b_J]$ such that $2^{b_J} > |x| \geq 2^{a_J}$. Algorithm 5.2 yields such a J in constant time.

Algorithm 5.2. Given $x \in \mathbb{F}$, calculate J such that $2^{b_J} > |x| \geq 2^{a_J}$.
 1: **function** INDEX(x, J)
 2:     $J = \lfloor (e_{\max} - \exp(x))/W \rfloor$
 3: **end function**
Ensure:
   The bin $(a_J, b_J]$ satisfies $2^{b_J} > |x| \geq 2^{a_J}$.

5.3 Deposit

The deposit operation (here referred to as Algorithm 5.4, a simpler version is described in the "Extract K first bins" Section (lines 18-20) of Algorithm 6 in [1]) is used to extract the slices of a floating point number and add them to the appropriate accumulators of an indexed type.

The indexed type Y that results from depositing the floating point values $x_0, ..., x_{n-1}$ into an empty indexed type is referred to as the **indexed sum** of $x_0, ..., x_{n-1}$. Algorithm 5.3 deposits floating point numbers in the case that there is no overflow (the indexed type has an index greater than 0).

Algorithm 5.3. Extract slices of $x \in \mathbb{F}$, where $|x| < 2^{b_I}$, in bins $(a_I, b_I], ..., (a_{I+K-1}, b_{I+K-1}]$ and add to indexed type Y. Here, $(r|1)$ represents the result of setting the last bit of the significand (m_{p-1}) of floating point number r to 1. This is a restatement of lines 18-20 of Algorithm 6 in [1].

Require:

No overflow occurs.

Operations are performed in some "to nearest" rounding mode (no specific tie breaking behavior is required).

$|x| < 2^{b_I}$.

$Y_{jP} \in (\epsilon^{-1} 2^{a_{I+j}}, 2\epsilon^{-1} 2^{a_{I+j}})$ at all times. (The carry fields Y_{jC} will be used to ensure this in Algorithm 5.5)

1: **function** DepositRestricted(K, x, Y)
2:     $r = x$
3:     **for** $j = 0$ to $(K-2)$ **do**
4:         $S = Y_{jP} + (r|1)$
5:         $q = S - Y_{jP}$
6:         $Y_{jP} = S$
7:         $r = r - q$
8:     **end for**
9:     $Y_{K-1P} = Y_{K-1P} + (r|1)$
10: **end function**

Ensure:

The amount added to Y_{jP} by this algorithm is exactly $d(x, I + j)$.

The last bit of r is set to break ties when rounding "to nearest" so that the amount added to Y_{jP} does not depend on the size of Y_{jP} so far. The

following theorem proves the "Ensure" claim at the end of Algorithm 5.3.

Theorem 5.1. Let Y be an K-fold indexed type of index I. Assume that we run Algorithm 5.3 on Y and some $x \in \mathbb{F}$, $|x| < 2^{b_I}$. If all requirements of the algorithm are satisfied, then the amount added to Y_{jP} is exactly $d(x, I+j)$.

Proof. Throughout the proof, assume that the phrase "for all j" means "for all $j \in \{0, ..., K-1\}$." Assume also that r_j and S_j refer to the value of r and S after executing line 4 in the jth iteration of the loop. Finally, assume Y_{jP} to refers to the initial value of Y_{jP} and S_j refers to the final value of Y_{jP}. Therefore, $S_j - Y_{jP}$ is the amount added to Y_{jP}.

Note that lines 4-7 correspond to Algorithm 4 of [1]. Therefore, if $\mathrm{ulp}(Y_{jP}) = \mathrm{ulp}(S_j)$ and $\mathrm{ulp}(r_j) < \frac{1}{2}\mathrm{ulp}(Y_{jP})$, Corollary 3 of [1] applies and we have that $S_j - Y_{jP} \in \mathrm{ulp}(Y_{jP})\mathbb{Z} \in 2^{a_{I+j}+1}\mathbb{Z}$ and that $|r_{j+1}| \leq \frac{1}{2}\mathrm{ulp}(Y_{jP}) = 2^{a_{I+j}}$.

As $Y_{jP}, S_j \in (\epsilon^{-1}2^{a_{I+j}}, 2\epsilon^{-1}2^{a_{I+j}})$, we have $\mathrm{ulp}(S_j) = \mathrm{ulp}(Y_{jP})$ for all j.

We show $|r_j| \leq 2^{b_{I+j}} = 2^{a_{I+j}-1}$ for all j inductively. As a base case, $r_0 = x$ (from line 2) so $|r_0| = |x| < 2^{b_I}$. As an inductive step, assume $|r_j| \leq 2^{b_{I+j}}$. By Theorem 4.2 we have that Y_{jP} is normalized and therefore $\mathrm{ulp}(Y_{jP}) = 2^{a_{I+j}+1}$ If r_j is normalized, then because $\mathrm{ulp}(r_j) \leq 2^{1-p}|r_j| \leq 2^{b_{I+j}-(p-1)} = 2^{a_{I+j}+W-(p-1)}$, and $W < p - 2$, we have $\mathrm{ulp}(r_j) < \frac{1}{2}\mathrm{ulp}(Y_{jP})$. (This case is considered in [1]).

If r_j is denormalized, $\mathrm{ulp}(r_j) = 2^{e_{\min}-p+1}$ since the unit in the last place of a denormalized number is always equal to $2^{e_{\min}-p+1}$. Therefore $\mathrm{ulp}(r_j) = 2^{e_{\min}-p+1} \leq 2^{e_{\min}-p+1+((e_{\max}-e_{\min}+p-1)\ \mathrm{mod}\ W)} = 2^{a_{i_{\max}}-1} \leq 2^{a_{I+j}-1} < \frac{1}{2}\mathrm{ulp}(Y_{jP})$.

Therefore we have $\mathrm{ulp}(r_j) < \frac{1}{2}\mathrm{ulp}(Y_{jP})$. Thus, Corollary 3 of [1] applies and we have that $|r_{j+1}| \leq \frac{1}{2}\mathrm{ulp}(Y_{jP}) = 2^{a_{I+j}}$.

Next, we show $S_j - Y_{jP} = \mathcal{R}_\infty(r_j, a_{I+j} + 1)$. As Corollary 3 of [1] applies for all j, then $S_j - Y_{jP} \in 2^{a_{I+j}+1}\mathbb{Z}$. By Theorem 3 of [1], $r_{j+1} = r_j - (S_j - Y_{jP})$. Since $|r_{j+1}| \leq 2^{a_{I+j}}$, we consider two cases.

If $|r_j - (S_j - Y_{jP})| < 2^{a_{I+j}}$, then $S_j - Y_{jP} = \mathcal{R}_\infty(r_j, a_{I+j} + 1)$.

If $|r_j - (S_j - Y_{jP})| = 2^{a_{I+j}}$, then $S_j - Y_{jP} \in \{r_j + 2^{a_{I+j}}, r_j - 2^{a_{I+j}}\}$. As $S_j = \mathrm{fl}(Y_{jP} + (r_j|1))$, we have $|S_j - Y_{jP} - (r_j|1)| \leq \frac{1}{2}\mathrm{ulp}(S_j) = 2^{a_{I+j}}$. As $\mathrm{ulp}(S_j) = \mathrm{ulp}(Y_{jP}) = 2^{a_{I+j}+1}$, we also have that $r_j \in 2^{a_{I+j}}\mathbb{Z}$ and because $\mathrm{ulp}(r_j) < 2^{a_{I+j}}$, $|(r_j|1)-r_j| > 0$ (with $(r_j|1)-r_j$ taking the same sign of r_j). If $r_j > 0$, then $(S_j - Y_{jP}) = r_j + 2^{a_{I+j}}$ (otherwise we will have $|S_j - Y_{jP} - (r_j|1)| > |r_j - 2^{a_{I+j}} - (r_j|1)| > 2^{a_{I+j}}$). If $r_j < 0$, then $(S_j - Y_{jP}) = r_j - 2^{a_{I+j}}$ (otherwise we will have $|S_j - Y_{jP} - (r_j|1)| > |r_j + 2^{a_{I+j}} - (r_j|1)| > 2^{a_{I+j}}$). Therefore, $S_j - Y_{jP} = \mathcal{R}_\infty(r_j, a_{I+j} + 1)$.

We can now show $r_{j+1} = x - \sum_{i=0}^{I+j} d(x, i)$ and $S_j - Y_{jP} = d(x, I + j)$ for all j by induction on j.

In the base case, $S_0 - Y_{0P} = \mathcal{R}_\infty(r_0, a_I + 1) = \mathcal{R}_\infty(x, a_I + 1)$. As $|x| < b_I$, Lemma 3.2 implies $S_0 - Y_{0P} = d(x, I)$. By Theorem 3 of [1], $r_1 = r_0 - (S_0 - Y_{0P}) = x - d(x, I)$. By assumption and (3.2) and (3.3), $|x| < b_I \le a_i$ for all $i \in \{0, ..., I - 1\}$, and therefore by Lemma 3.1, $r_1 = x - \sum_{i=0}^{I} d(x, i)$.

In the inductive step, assume $r_{j+1} = x - \sum_{i=0}^{I+j} d(x, i)$. Then by definition,

$$S_{j+1} - Y_{j+1P} = \mathcal{R}_\infty(r_{j+1}, a_{I+j+1} + 1) = \mathcal{R}_\infty(x - \sum_{i=0}^{I+j} d(x, i), a_{I+j+1} + 1) = d(x, I+j+1)$$

And by Theorem 3 of [1],

$$r_{j+2} = r_{j+1} - (S_{j+1} - Y_{j+1P}) = (x - \sum_{i=0}^{I+j} d(x, i)) - d(x, I+j+1) = x - \sum_{i=0}^{I+j+1} d(x, i)$$

$\square$

Of course, what remains to be seen is how we can extract and add the components of a floating point number to an indexed type Y of index 0. Algorithm 5.4 shows the adaptation of Algorithm 5.3 for indexed types of

index 0.

Algorithm 5.4. Extract components of $x \in \mathbb{F}$, where $|x| < 2^{b_I}$, in bins $(a_I, b_I], ..., (a_{I+K-1}, b_{I+K-1}]$ and add to indexed type Y of index I. Here, $(r|1)$ represents the result of setting the last bit of the significand (m_{p-1}) of floating-point r to 1.

Require:

All requirements (except for the absence of overflow, which we will ensure) from Algorithm 5.3 except that Y_{0P} must now be kept within the range $(2^{e_{\max}}, 2 \cdot 2^{e_{\max}})$ if Y has index 0.

1: **function** DEPOSIT(K, x, Y)
2:     **if** Y has index 0 **then**
3:         $r = x/2^{p-W+1}$
4:         $S = Y_{0P} + (r|1)$
5:         $q = S - Y_{0P}$
6:         $Y_{0P} = S$
7:         $q = q \cdot 2^{p-W}$
8:         $r = x - q$
9:         $r = r - q$
10:        **for** $j = 1$ to $(K-2)$ **do**
11:           $S = Y_{jP} + (r|1)$
12:           $q = S - Y_{jP}$
13:           $Y_{jP} = S$
14:           $r = r - q$
15:        **end for**
16:        $Y_{K-1P} = Y_{K-1P} + (r|1)$
17:     **else**
18:         DEPOSITRESTRICTED(K, x, Y)
19:     **end if**
20: **end function**

Ensure:

No overflow occurs during the algorithm.
The amount added to Y_{jP} is exactly $d(x, I + j)$ if $I + j \neq 0$.
The amount added to Y_{0P} is exactly $d(x, 0)/2^{p-W+1}$ if $I = 0$.

Theorem 5.2 shows that Algorithm 5.4 enjoys the necessary properties.

Theorem 5.2. Let Y be a K-fold indexed type of index I. Assume that we run Algorithm 5.4 on Y and some $x \in \mathbb{F}$, $|x| < 2^{b_I}$. If all requirements of the algorithm are satisfied, then the "Ensure" claim of Algorithm 5.4 holds.

Proof. We can break the proof into two cases based on the index I of Y.

If $I \neq 0$, then we execute line 18. By (3.3), $|x| < 2^{b_I} \leq 2^{e_{\max}+1-W}$. By Theorem 4.1, all quantities are obviously well below the overflow threshold and Theorem 5.1 applies and we are done.

If $I = 0$, more work is needed.

We begin by showing that the amount added to Y_{0P} ($S - Y_{0P}$ in line 4) is equal to $\mathcal{R}_\infty(r, a_0 - p + W)$.

As $|x| < 2^{b_0} = 2 \cdot 2^{e_{\max}}$ and we scale by a power of two, we have $|r| < 2^{b_0 - p + W - 1} = 2^{e_{\max} - p + W}$ in line 3.

Since $Y_{0P}, S \in (2^{e_{\max}}, 2 \cdot 2^{e_{\max}})$, we have $\mathrm{ulp}(Y_{0P}) = \mathrm{ulp}(S)$ in line 4. As $|r| < 2^{e_{\max} - p + W}$ and $W < p - 2$, we have $\mathrm{ulp}(r) < \frac{1}{2}\mathrm{ulp}(Y_{0P})$ in line 4.

Because line 4 corresponds to lines 1-2 of Algorithm 4 of [1], Corollary 3 of [1] applies and we have that $S - Y_{0P} \in \mathrm{ulp}(Y_{0P})\mathbb{Z} = 2^{a_0 - p + W}\mathbb{Z}$ and $|r - (S - Y_{0P})| \leq \frac{1}{2}\mathrm{ulp}(Y_{0P}) = 2^{a_0 - p + W - 1}$ in line 4.

If $|r - (S - Y_{0P})| < 2^{a_0 - p + W - 1}$, then $S - Y_{jP} = \mathcal{R}_\infty(r, a_0 - p + W)$.

If $|r - (S - Y_{0P})| = 2^{a_0 - p + W - 1}$, then $S - Y_{0P} \in \{r + 2^{a_0 - p + W - 1}, r_j - 2^{a_0 - p + W - 1}\}$. As $S = \mathrm{fl}(Y_{0P} + (r|1))$, we have $|S - Y_{0P} - (r|1)| \leq \frac{1}{2}\mathrm{ulp}(S) = 2^{a_0 - p + W - 1}$. As $\mathrm{ulp}(S) = \mathrm{ulp}(Y_{0P}) = 2^{a_0 - p + W}$, we also have that $r \in 2^{a_0 - p + W - 1}\mathbb{Z}$ and because $\mathrm{ulp}(r) < 2^{a_0 - p + W - 1}$, $|(r|1) - r| > 0$ (with $(r|1) - r$ taking the same sign of r). If $r > 0$, then $(S - Y_{0P}) = r + 2^{a_0 - p + W - 1}$ (otherwise we will have $|S - Y_{0P} - (r|1)| > |r - 2^{a_0 - p + W - 1} - (r|1)| > 2^{a_0 - p + W - 1}$). If $r < 0$, then $(S - Y_{0P}) = r - 2^{a_0 - p + W}$ (otherwise we will have $|S - Y_{0P} - (r|1)| > |r + 2^{a_0 - p + W - 1} - (r|1)| > 2^{a_0 - p + W - 1}$). Therefore, $S - Y_{0P} = \mathcal{R}_\infty(r, a_0 - p + W)$.

Next, we show that $Y_{0P} - S = d(x, 0)/2^{p - W + 1}$.

We divide into two cases based on the size of x.

If $|x| < 2^{a_0} = 2^{e_{\max}+1-W}$, then we have $|r| < 2^{a_0 - p + W - 1}$ in line 3 regardless of whether or not there is underflow in the division as we scale by a power of two. Therefore, since $|r| < 2^{a_0 - p + W - 1}$, $Y_{0P} - S = \mathcal{R}_\infty(r, a_0 - p + W) = 0 = d(x, 0)/2^{p - W - 1}$ by Lemma 3.1.

If $|x| \geq 2^{a_0} = 2^{e_{\max}+1-W}$, then there is no underflow in line 3 and $r = x/2^{p - W + 1}$ exactly as we scale by a power of two. Therefore, $Y_{0P} - S = \mathcal{R}_\infty(x/2^{p - W + 1}, a_0 - p + W) = d(x, 0)/2^{p - W + 1}$

At this point, all that remains to be shown is that $r = x - d(x, 0)$ in line 9. In line 5, we have that $q = Y_{0P} - S = d(x, 0)/2^{p - W + 1}$. By Theorem 3.4, $d(x, 0) \leq 2 \cdot 2^{e_{\max}}$. We then have that in line 7, since $q = (d(x, 0)/2^{p - W + 1})2^{p - W} \leq 2^{e_{\max}}$ there is no overflow and as we scale by a power of two, $q = d(x, 0)/2$ exactly. Again we divide into two cases based on the size of x.

If $|x| < 2^{a_0}$, we have $d(x, 0) = 0$ by Lemma 3.1 and therefore $r = x - d(x, 0) = x$ exactly in both line 8 and line 9.

If $|x| \geq 2^{a_0}$, we have $|x - d(x, 0)| \leq 2^{a_0}$ by Theorem 3.3. Therefore, we have $|x| \geq |x - d(x, 0)|$.

If $x > 0$, we have

$$x \geq x - d(x, 0)/2 \geq x - d(x, 0) \geq -x$$

If $x < 0$, we have

$$-x \geq x - d(x, 0) \geq x - d(x, 0)/2 \geq x$$

In either case we have $|x - d(x, 0)/2| \leq |x|$.

Since $d(x, 0)/2 = \mathcal{R}_\infty(x, a_0 + 1)/2 \in 2^{a_0}\mathbb{Z} \in 2\epsilon 2^{b_0}$ (As $W < p - 2$) and $x \leq 2^{b_0}$, $d(x, 0)/2 \in \text{ulp}(x)\mathbb{Z}$ and therefore $x - d(x, 0)/2 \in \text{ulp}(x)\mathbb{Z}$. Combined with $|x - d(x, 0)/2| \leq |x|$ this implies that $r = x - d(x, 0)/2$ exactly in line 8.

Again since $d(x, 0)/2, x - d(x, 0)/2 \in \text{ulp}(x)\mathbb{Z}$, $x - d(x, 0) \in \text{ulp}(x)\mathbb{Z}$ and since $|x - d(x, 0)| \leq |x|$, $r = x - d(x, 0)$ exactly in line 9.

At this point, since $r = x - d(x, 0)$ and $|r| \leq 2^{a_0}$, no more overflow can occur in the algorithm and since the algorithm at this point is identical to Algorithm 5.3, the proof of Theorem 5.1 applies. □

Modifying Algorithm 5.4 to correctly handle exceptional values is easy to implement. At the beginning of Algorithm 5.4, we may simply check x and Y_{0P} for the exceptional values `Inf`, `-Inf`, and `NaN`. If any one of x or Y_{0P} is indeed exceptional, we add x to the (possibly finite) Y_{0P}. Otherwise, we deposit the finite value normally.

Note that an explicit check for exceptional values is necessary at some point in the summation, as Algorithm 5.4 cannot correctly sum an `Inf` or `-Inf` (it changes the `Inf` to a `NaN` when setting the last bit of r). Although checking for exceptional values explicitly is expensive, the cost can be reduced if several values are to be summed in the same method (for example, using the methods discussed in Section 8).

5.4 Renormalize

When depositing values into a K-fold indexed type Y of index I, the assumption is made that $Y_{jP} \in (\epsilon^{-1}2^{a_{I+j}}, 2\epsilon^{-1}2^{a_{I+j}})$ throughout the routine.

To enforce this condition, the indexed type must be renormalized at least every 2^{p-w-2} deposit operations, as will be shown in Theorem 5.3. The renormalization procedure is shown in Algorithm 5.5.

Algorithm 5.5. For a K-fold indexed type Y of index I, assuming $Y_{jP} \in [1.25\epsilon^{-1}2^{a_{I}+j}, 2\epsilon^{-1}2^{a_{I}+j})$, renormalize Y such that $Y_{jP} \in [1.5\epsilon^{-1}2^{a_{I}+j}, 1.75\epsilon^{-1}2^{a_{I}+j})$.

Require:

   $Y_{jP} \in [1.25\epsilon^{-1}2^{a_{I}+j}, 2\epsilon^{-1}2^{a_{I}+j})$

 1: **function** RENORM(K, Y)
 2:     **for** $j = 0$ to $K - 1$ **do**
 3:         **if** $Y_{jP} < 1.5 \cdot \mathrm{ufp}(Y_{jP})$ **then**
 4:             $Y_{jP} = Y_{jP} + 0.25 \cdot \mathrm{ufp}(Y_{jP})$
 5:             $Y_{jC} = Y_{jC} - 1$
 6:         **end if**
 7:         **if** $Y_{jP} \geq 1.75 \cdot \mathrm{ufp}(Y_{jP})$ **then**
 8:             $Y_{jP} = Y_{jP} - 0.25 \cdot \mathrm{ufp}(Y_{jP})$
 9:             $Y_{jC} = Y_{jC} + 1$
10:         **end if**
11:     **end for**
12: **end function**

Ensure:

   $Y_{jP} \in [1.5\epsilon^{-1}2^{a_{I}+j}, 1.75\epsilon^{-1}2^{a_{I}+j})$.
   The values $\mathcal{Y}_j$ are unchanged. Recall that by (4.3),

$$\mathcal{Y}_j = \mathcal{Y}_{jP} + \mathcal{Y}_{jC} = (Y_{jP} - 1.5\epsilon^{-1}2^{a_{I}+j}) + (0.25\epsilon^{-1}2^{a_{I}+j})Y_{jC}$$

The renormalization operation is described in the "Carry-bit Propagation" Section (lines 18-29) of Algorithm 6 in [1], although it has been slightly modified so as not to include an extraneous case.

To show the reasoning behind the assumptions in Algorithm 5.5, we prove Theorem 5.3.

Theorem 5.3. Assume $x_0, x_1, ...x_{n-1} \in \mathbb{F}$ are successively deposited (using Algorithm 5.4) in a K-fold indexed type Y of index I where $\max |x_k| < 2^{b_I}$. If Y initially satisfies $Y_{jP} \in [1.5\epsilon^{-1}2^{a_{I}+j}, 1.75\epsilon^{-1}2^{a_{I}+j})$ and $n \leq 2^{p-W-2}$, then after all of the deposits, $Y_{jP} \in [1.25\epsilon^{-1}2^{a_{I}+j}, 2\epsilon^{-1}2^{a_{I}+j})$.

Proof. First, note that $|d(x_k, I+j)| \leq 2^{b_{I}+j}$ by Theorem 3.4, where $d(x_k, I+j)$ is the amount added to Y_{jP} on iteration j.

By Theorem 5.2, the deposit operation extracts and adds the slices of x_k exactly (assuming $Y_{jP} \in (\epsilon^{-1}2^{a_I+j}, 2\epsilon^{-1}2^{a_I+j})$ at each step, which will be shown),

$$|\sum_{n=0}^{k-1} d(x_k, I+j)| \leq n2^{b_I+j} = n2^W 2^{a_I+j}$$

If $n \leq 2^{p-W-2}$, then after the nth deposit

$$Y_{jP} \in [(1.5\epsilon^{-1} - n2^W)2^{a_I+j}, (1.75\epsilon^{-1} + n2^W)2^{a_I+j})$$

$$\in [1.25\epsilon^{-1}2^{a_I+j}, 2\epsilon^{-1}2^{a_I+j})$$

$\square$

If an indexed type Y initially satisfies $Y_{jP} \in [1.5\epsilon^{-1}2^{a_I+j}, 1.75\epsilon^{-1}2^{a_I+j})$ (such a condition is satisfied upon initialization of an empty Y) and we deposit at most 2^{p-W-2} floating point numbers into it, then Theorem 5.3 shows that after all of the deposits, $Y_{jP} \in [1.25\epsilon^{-1}2^{a_I+j}, 2\epsilon^{-1}2^{a_I+j})$.

Therefore, after another renormalization, the primary fields would once again satisfy

$$Y_{jP} \in [1.5\epsilon^{-1}2^{a_I+j}, 1.75\epsilon^{-1}2^{a_I+j})$$

As Y_{jC} must be able to record additions of absolute value 1 without error, Y_{jC} must stay in the range $(-\epsilon^{-1}, \epsilon^{-1})$. As each renormalization results in addition not in excess absolute value of 1 to Y_{jC}, a maximum of $(\epsilon^{-1} - 1)$ renormalizations may be performed, meaning that an indexed type is capable of representing the sum of at least

$$0.25\epsilon^{-1}2^{-W}(\epsilon^{-1} - 1) \approx 2^{2p-W-2} \tag{5.1}$$

floating point numbers.

5.5 Update

As noted in Algorithm 5.4, when adding $x \in \mathbb{F}$ to a K-fold indexed type Y of index I, we make the assumption that $|x| < 2^{b_I}$. However, this is not always the case and sometimes it is necessary to adjust the index of Y. This

adjustment is called an **update**. The process of updating Y to the necessary index is summarized succinctly in Algorithm 5.6.

Algorithm 5.6. Update K-fold indexed type Y of index I to have an index suitable to deposit x.

Require:

$\quad Y_{0P} \in (\epsilon^{-1} 2^{a_I}, 2\epsilon^{-1} 2^{a_I})$

1: **function** UPDATE(K, x, Y)
2: $\quad$ IINDEX(Y, I)
3: $\quad$ INDEX(x, J)
4: $\quad$ **if** $J < I$ **then**
5: $\quad\quad$ $[Y_{\min(I-J,K)\,P}, ..., Y_{K-1\,P}] = [Y_{0P}, ..., Y_{K-1-\min(I-J,K)\,P}]$
6: $\quad\quad$ $[Y_{0P}, ..., Y_{\min(I-J,K)-1\,P}] = [1.5\epsilon^{-1} a_J, ..., 1.5\epsilon^{-1} a_{\min(I,K+J)-1}]$
7: $\quad\quad$ $[Y_{\min(I-J,K)\,C}, ..., Y_{K-1\,C}] = [Y_{0C}, ..., Y_{K-1-\min(I-J,K)\,C}]$
8: $\quad\quad$ $[Y_{0C}, ..., Y_{\min(I-J,K)-1\,C}] = [0, ..., 0]$
9: $\quad$ **end if**
10: **end function**

Ensure:

$\quad$ The bin $(a_J, b_J]$ satisfies $2^{b_J} > |x| \geq 2^{a_J}$.

$\quad$ The new index of Y, $(\min(I, J))$ is such that $2^{b_{\min(I,J)}} > |x|$.

$\quad$ Existing accumulators of Y are shifted towards index 0, losing the lesser bins.

$\quad$ New accumulators are shifted into Y with value 0 in Y_{jC} and $1.5\epsilon^{-1} 2^{a_{\min(I,J)+j}}$ in Y_{jP}.

The update operation is described in the "Update" Section (lines 7-14) of Algorithm 6 in [1].

It should be noted that if Y_{0P} is 0, then the update is performed as if $I + K < J$. If Y_{0P} is `Inf`, `-Inf`, or `NaN`, then Y is not modified by an update.

To speed up this operation, the factors $1.5\epsilon^{-1} a_k$ for all valid $k \in Z$ are stored in a precomputed array.

It should be noted that if J is such that $J + K > i_{\max}$, then $Y_{i_{\max}-J+1\,P}, ..., Y_{J+K\,P}$ are set to $1.5\epsilon^{-1} 2^{a_{i_{\max}}}$ and the values in these accumulators are ignored.

5.6 Reduce

An operation to produce the sum of two indexed types is necessary to perform a reduction. For completeness we include the algorithm here, although apart

from the simplified renormalization algorithm, it is equivalent to Algorithm 7 in [1].

Algorithm 5.7. Given a K-fold indexed type Y of index I and a K-fold indexed type Z of index J, add Z to Y.

Ensure:

Y is the indexed sum of some $x_0, ..., x_{n-1} \in F$.

Z is the indexed sum of some $x_n, ..., x_{n+m-1} \in F$.

$Y_{jP}, Z_{jP} \in [1.5\epsilon^{-1}2^{a_{J+j}}, 1.75\epsilon^{-1}2^{a_{J+j}})$

1: **function** REDUCE(K, Y, Z)
2:     IINDEX(Y, I)
3:     IINDEX(Z, J)
4:     **if** $J < I$ **then**
5:         $R = Z$
6:         REDUCE(K, R, Y)
7:         $Y = R$
8:     **end if**
9:     $j = J - I$
10:     **while** $j < K$ **do**
11:         $Y_{jP} = Y_{jP} + Z_{j+I-J\,P} - 1.5\epsilon^{-1}2^{a_{I+j}}$
12:         $Y_{jC} = Y_{jC} + Z_{j+I-J\,C}$
13:         $j = j + 1$
14:     **end while**
15:     RENORM(K, Y)
16: **end function**

Ensure:

Y is set to the indexed sum of $x_0, ..., x_{n+m-1}$.

$Y_{jP} \in [1.5\epsilon^{-1}2^{a_{\min(I,J)+j}}, 1.75\epsilon^{-1}2^{a_{\min(I,J)+j}})$

5.7 Convert

It is necessary to provide the ability to convert between indexed and floating point representations of a number.

Converting a floating point number to an indexed type should produce, for transparency and reproducibility, the indexed sum of the single floating

point number. The procedure is very simply summarized by Algorithm 5.8

Algorithm 5.8. Convert floating point x to a K-fold indexed type Y.

1: **function** CONVERTFLOATTOINDEXED(K, x, Y)
2:     $Y = 0$
3:     UPDATE(K, x, Y)
4:     DEPOSIT(K, x, Y)
5:     RENORM(K, Y)
6: **end function**

Ensure: Y is the indexed sum of x. The index I of Y is such that $2^{b_I} > |x| \geq 2^{a_I}$ $Y_{jP} \in [1.5\epsilon^{-1}2^{a_{I+j}}, 1.75\epsilon^{-1}2^{a_{I+j}})$

We must renormalize the indexed type because $d(x, I + j)$ could be negative, meaning $Y_{jP} < 1.5\epsilon^{-1}2^{a_{I+j}}$ after execution of line 4.

Converting an indexed type to a single floating point number is more difficult. There are two sources of error in the final floating point sum produced by ReproBLAS. The first is from the creation of the indexed sum (analyzed in [1]). The second is from the conversion from indexed sum to floating point number (evaluation of (4.4)). Because [1] guarantees that all fields in the indexed type are reproducible, as long as the fields are operated upon deterministically, any method to evaluate (4.4) accurately and without unnecessary overflow is suitable.

Assume that Y is the indexed sum of some $x_0, ..., x_{n-1} \in \mathbb{F}$. If we simply evaluate equation (4.4) and apply the standard summation error bound given by [4], the error in the final answer (the computed floating point approximation of $\mathcal{Y}$) is only bounded by about (using (5.30))

$$n(2^{W(1-K)-1} + (2K - 1)\epsilon) \max |x_k| \tag{5.2}$$

However, if we sum the fields in order of decreasing "unnormalized" exponent, the error is bounded by about (using (5.27))

$$n2^{W(1-K)-1} \max |x_k| + 7\epsilon \left| \sum_{k=0}^{n-1} x_k \right| \tag{5.3}$$

When adding the fields it is not necessary to examine the values in the fields or sort them explicitly. Their "unnormalized" exponent does not depend on their values, and their "unnormalized" exponents have a predetermined order. There is therefore little difference in computational cost

between the two methods (they both require the same number of additions, namely $2K$, using the same precision), but showing that the fields have a certain ordering and that the stronger bound applies requires extensive analysis.

To further motivate the new conversion algorithm, we compare both the above error bounds for indexed summation to an approximated standard error bound obtained through standard (recursive) summation of $x_0, ..., x_{n-1}$ in some arbitrary order (given by [4])

$$n\epsilon \sum_{k=0}^{n-1} |x_k| \leq n^2 \epsilon \max |x_k| \tag{5.4}$$
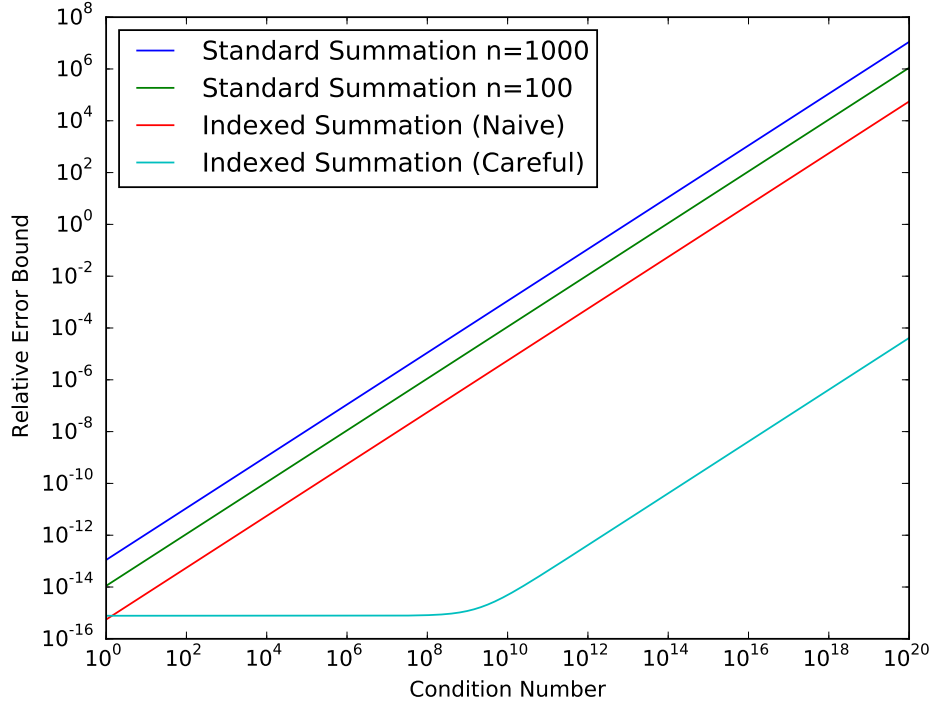
Figure 5.1 compares these three approximate error bounds.

Figure 5.1: Relative error bounds in calculating $|\sum\limits_{k=0}^{n-1} x_k|$ for different condition numbers (which we define as $\frac{n \cdot \max |x_k|}{|\sum\limits_{k=0}^{n-1} x_k|}$) of the sum. It is assumed that we sum using `double`, $K = 3$, and $W = 40$. "Indexed Summation (Careful)" corresponds to (5.3). "Indexed Summation (Naive)" corresponds to (5.2). "Standard Summation" corresponds to (5.4) and due to a dependence on n multiple error bounds are shown.

Consider a K-fold indexed type Y of index I. As each value $\mathcal{Y}_{jP}$ in a primary field Y_{jP} is represented by an offset from $1.5\epsilon^{-1}2^{a_{I+j}}$ and $Y_{jP} \in (\epsilon^{-1}2^{a_{I+j}}, 2\epsilon^{-1}2^{a_{I+j}})$, $\mathcal{Y}_{jP}$ can be expressed exactly using an unnormalized floating point number $\mathcal{Y}'_{Pj}$ with an exponent of $a_{I+j}+p-1$. As each carry field Y_{jC} is a count of renormalization adjustments later scaled by $0.25\epsilon^{-1}2^{a_{I+j}}$, $\mathcal{Y}_{jC}$ can be expressed exactly using an unnormalized floating point number $\mathcal{Y}'_{jC}$ with an exponent of $a_{I+j} + 2p - 3$.

For all $k < j$, $\exp(\mathcal{Y}'_{kP}) > \exp(\mathcal{Y}'_{jP})$ and $\exp(\mathcal{Y}'_{kC}) > \exp(\mathcal{Y}'_{jC})$ because $a_{I+k} > a_{I+j}$.

Note that

$$\exp(\mathcal{Y}'_{jC}) = a_{I+j} + 2p - 3$$

and

$$\exp(\mathcal{Y}'_{j-1P}) = a_{I+j-1} + p - 1 = a_{I+j} + W + p - 1$$

Therefore $\exp(\mathcal{Y}'_{jC}) > \exp(\mathcal{Y}'_{j-1P})$ using $W < p - 2$.

Note that

$$\exp(\mathcal{Y}'_{j-2P}) = a_{I+j-1} + p - 1 = a_{I+j} + 2W + p - 1$$

Therefore $\exp(\mathcal{Y}'_{jC}) < \exp(\mathcal{Y}'_{j-2P})$ using $2W > p + 1$.

Combining the above inequalities, we see that the exponents of all the $\mathcal{Y}'_{jP}$ and $\mathcal{Y}'_{jC}$ are distinct and can be sorted as follows:

$$\exp(\mathcal{Y}'_{0C}) > \exp(\mathcal{Y}'_{1C}) > \exp(\mathcal{Y}'_{0P}) > \exp(\mathcal{Y}'_{2C}) > \exp(\mathcal{Y}'_{1P}) > \dots$$

$$\dots > \exp(\mathcal{Y}'_{jC}) > \exp(\mathcal{Y}'_{j-1P}) > \exp(\mathcal{Y}'_{j+1C}) > \exp(\mathcal{Y}'_{jP}) > \dots$$

$$\dots > \exp(\mathcal{Y}'_{K-2C}) > \exp(\mathcal{Y}'_{K-3P}) > \exp(\mathcal{Y}'_{K-1C}) > \exp(\mathcal{Y}'_{K-2P}) > \exp(\mathcal{Y}'_{K-1P})$$

These unnormalized floating point numbers may, for convenience of notation, be referred to in decreasing order of unnormalized exponent as $\gamma'_0, ..., \gamma'_{2K-1}$.

We have just shown that

$$\exp(\gamma'_0) > \dots > \exp(\gamma'_{2K-1}) \tag{5.5}$$

γ_k denotes the normalized representations of the γ'_k, and it should be noted that $\gamma_k = \gamma'_k$ as real numbers and that $\exp(\gamma_k) \leq \exp(\gamma'_k)$.

It should be noted that if γ_k is a primary field, then either γ_{k+1} or γ_{k+2} is a primary field. If γ_k is a carry field, then either γ_{k+1} or γ_{k+2} is a carry field (with the exception of γ_{2K-3}, but in this case we have $\exp(\gamma_{2K-3}) = a_{I+K-1} + 2p - 3 \geq a_{I+K-1} + p + \lceil \frac{p+1}{2} \rceil - 1 = \exp(\gamma_{2K-1}) + \lceil \frac{p+1}{2} \rceil)$. Therefore, as $2W > p + 1$, for all $k \in \{0, ..., 2K - 1\}$

$$\exp(\gamma'_k) \geq \exp(\gamma'_{k+2}) + W \geq \exp(\gamma'_{k+2}) + \left\lceil \frac{p+1}{2} \right\rceil \tag{5.6}$$

It should be noted that the $\mathcal{Y}'_{jP}$ and the $\mathcal{Y}'_{jC}$ can be expressed exactly using floating point types of the same precision as Y_{jP} and Y_{jC} (except in the case of overflow, in which a scaled version may be obtained), and such exact floating point representations can be obtained using (4.1) and (4.2).

Now that we know how to obtain sorted, possibly scaled, fields in order of decreasing unnormalized exponent, we explain how to sum them while avoiding overflow. We will refer to the floating point type that we use to hold the sum during computation as the **intermediate** floating point type. Such a type must have at least as much precision and exponent range as the original floating point type.

Notice that $|\gamma'_0| = |\mathcal{Y}'_{0C}| < 2 \cdot 2^{\exp(\mathcal{Y}'_{0C})} = 2 \cdot 2^{e_{\max}+1-W+2p-3}$ and $\exp(\gamma'_0) >$ ... $> \exp(\gamma'_{2K-1})$. Therefore $|\gamma'_k| \leq 2^{e_{\max}-W+2p-1-k}$. The absolute value represented by an indexed type can therefore be bounded by

$$\sum_{k=0}^{2K-1} |\gamma_k| < \sum_{k=0}^{2K-1} 2^{e_{\max}-W+2p-1-k} < \sum_{k=0}^{\infty} 2^{e_{\max}-W+2p-1-k} = 2^{e_{\max}-W+2p} \quad (5.7)$$

If the intermediate floating point type has a maximum exponent greater than or equal to $e_{\max} - W + 2p - 1$, then no special cases to guard against overflow are needed.

Algorithm 5.9 represents a conversion routine in such a case.

Algorithm 5.9. Convert K-fold indexed type Y of index I to floating point x. Here, z is a floating point type with at least the original precision and maximum exponent $E_{\max}$ greater than $e_{\max} - W + 2p$

1: **function** CONVERTINDEXEDTOFLOAT(K, x, Y)
2:    $z = \mathcal{Y}_{0C}$
3:    **for** $j = 1$ to $K - 1$ **do**
4:        $z = z + \mathcal{Y}_{jC}$
5:        $z = z + \mathcal{Y}_{j-1P}$
6:    **end for**
7:    $z = z + \mathcal{Y}_{K-1P}$
8:    $x = z$
9: **end function**

Note that an overflow situation in Algorithm 5.9 is reproducible as the fields in Y are reproducible. z is deterministically computed from the fields

of Y, and the condition that z overflows when being converted back to the original floating point type in line 8 is reproducible.

If an intermediate floating point type with exponent greater than or equal to $e_{\max} - W + 2p - 1$ is not available, the γ_k must be scaled down by some factor during addition and the sum scaled back up when subsequent additions can no longer effect an overflow situation.

If the scaled sum is to overflow, then its unscaled value will be greater than or equal to $2 \cdot 2^{e_{\max}}$ and it will overflow regardless of the values of any $\mathcal{Y}_{jP}$ or $\mathcal{Y}_{jC}$ with $|\mathcal{Y}_{jP}| < 0.5 \cdot 2^{-\rho} 2^{e_{\max}}$ or $|\mathcal{Y}_{jC}| < 0.5 \cdot 2^{-\rho} 2^{e_{\max}}$ (where ρ is the intermediate floating point type's precision). If the floating point sum has exponent greater than or equal to $e_{\max}$ these numbers are not large enough to have any effect when added to the sum. If the sum has exponent less than $e_{\max}$, then additions of these numbers cannot cause the exponent of the sum to exceed $e_{\max}$ for similar reasons.

As the maximum absolute value of the true sum is strictly smaller than $2^{e_{\max} - W + 2p}$, a sufficient scaling factor is $2^{2p - W - 2}$, meaning that the maximum absolute value of the true scaled sum is strictly smaller $2 \cdot 2^{e_{\max} - 1}$ (and since it will be shown later that the computed sum is accurate to within a small factor of the true sum, the computed sum will stay strictly smaller than $2 \cdot 2^{e_{\max}}$ and will not overflow.)

When $\exp(\gamma_k') < e_{\max} - \rho - 1$, the sum may be scaled back up and the remaining numbers added without scaling. Notice that no overflow can occur during addition in this algorithm. If an overflow is to occur, it will happen only when scaling back up. As the fields in the indexed type are reproducible, such an overflow condition is reproducible.

If the sum is not going to overflow, then the smaller y_k' must be added as unscaled numbers to avoid underflow.

Algorithm 5.10 represents a conversion routine in such a case.

Algorithm 5.10. Convert a K-fold indexed type Y of index I to floating point x. Here, z is a floating point number with precision $\rho > p$

```
 1: function CONVERTINDEXEDTOFLOAT(K, x, Y)
 2:     j = 1
 3:     while j ≤ 2K and exp(γ_j) ≥ e_max − ρ − 1 do
 4:         z = z + (γ_j/2^{2p−W−2})
 5:         j = j + 1
 6:     end while
 7:     z = z · 2^{2p−W−2}
 8:     while j ≤ 2K do
 9:         z = z + γ_j
10:         j = j + 1
11:     end while
12:     x = z
13: end function
```

If an indexed type is composed of `float`, then `double` provides sufficient precision and exponent to use as an intermediate type and Algorithm 5.9 may be used to convert to a floating point number. However, if an indexed type is composed of `double`, many machines may not have any higher precision available. We therefore perform the sum using `double` as an intermediate type. As this does not extend the exponent range we must use Algorithm 5.10 for the conversion.

5.8 Error Bound

We first state and prove Theorem 5.4, as it is critical in the error analysis of the algorithm. It should be noted that Theorem 5.4 is similar to that of Theorem 1 from [5], but requires less intermediate precision by exploiting additional structure of the input data.

It is possible that future implementors may make modifications to the indexed type (adding multiple carry fields, changing the binning scheme, etc.) such that the summation of its fields cannot be reordered to satisfy the assumptions of Theorem 5.4. In such an event, [5] provides more general ways to sum the fields while still maintaining accuracy.

Theorem 5.4. Given n floating point numbers $f_0, ..., f_{n-1}$ for which there exists (possibly unnormalized) floating point numbers $f'_0, ..., f'_{n-1}$ of the same precision such that

1. $f_k = f'_k$ for all $k \in \{0, ..., n-1\}$

2. $\exp(f'_0) > ... > \exp(f'_{n-1})$

3. $\exp(f'_k) \geq \exp(f'_{k+2}) + \left\lceil \frac{p+1}{2} \right\rceil$ for all $k \in \{0, ..., n-3\}$

Let $S_0 = \overline{S_0} = f_0$, $S_k = S_{k-1} + f_k$, and $\overline{S_k} = \mathrm{fl}(\overline{S_{k-1}} + f_k)$ so that $S_{n-1} = \sum_{k=0}^{n-1} f_k$. Then we have

$$\left| S_{n-1} - \overline{S_{n-1}} \right| < \frac{7\epsilon}{1 - 6\sqrt{\epsilon}} \left| \sum_{k=0}^{n-1} f_k \right| \approx 7\epsilon \left| \sum_{k=0}^{n-1} f_k \right|$$

Proof. Throughout the proof, let $f_k = 0$ if $k > n-1$ so that $S_\infty = S_{n-1}$ and $\overline{S_\infty} = \overline{S_{n-1}}$.

Let m be the location of the first error such that $S_{m-1} = \overline{S_{m-1}}$ and $S_m \neq \overline{S_m}$.

If no such m exists then the computed sum is exact $(S_{n-1} = \overline{S_{n-1}})$ and we are done.

If such an m exists, then because $\exp(f'_0) > ... > \exp(f'_m)$, $f_0, ..., f_m \in \mathrm{ulp}(f'_m)\mathbb{Z}$. Thus, $S_m \in \mathrm{ulp}(f'_m)\mathbb{Z}$.

We now show $|S_m| > 2 \cdot 2^{\exp(f'_m)}$. Assume for contradiction that $|S_m| \leq 2 \cdot 2^{\exp(f'_m)}$. Because $S_m \in \mathrm{ulp}(f'_m)\mathbb{Z}$, this would imply that S_m is representable as a floating point number, a contradiction as $\overline{S_m} \neq S_m$. Therefore, we have

$$|S_m| > 2 \cdot 2^{\exp(f'_m)} \tag{5.8}$$

Because $\exp(f'_m) > \exp(f'_{m+1})$,

$$|f_{m+1}| < 2 \cdot 2^{\exp(f'_m - 1)} = 2^{\exp(f'_m)} \tag{5.9}$$

Because $\exp(f'_m) \geq \exp(f'_{m+2}) + \left\lceil \frac{p+1}{2} \right\rceil$ and $\exp(f'_0) > ... > \exp(f'_{n-1})$,

$$\left| \sum_{k=m+2}^{n-1} f_k \right| \leq \sum_{k=m+2}^{n-1} |f_k| < \sum_{k=m+2}^{n-1} 2 \cdot 2^{\exp(f'_k)} \leq \sum_{k=m+2}^{n-1} 2 \cdot 2^{\exp(f'_m) - \left\lceil \frac{p+1}{2} \right\rceil - (m+2-k)}$$

$$< \sum_{k=0}^{\infty} (2\sqrt{\epsilon}) 2^{\exp(f'_m) - k} = (4\sqrt{\epsilon}) 2^{\exp(f'_m)} \tag{5.10}$$

We can combine (5.9) and (5.10) to obtain

$$\left| \sum_{k=m+1}^{n-1} f_k \right| \leq \sum_{k=m+1}^{n-1} |f_k| < 2^{\exp f'_m} + (4\sqrt{\epsilon}) 2^{\exp(f'_m)} = (1 + 4\sqrt{\epsilon}) 2^{\exp(f'_m)} \tag{5.11}$$

By (5.8) and (5.11),

$$|S_{n-1}| = \left| \sum_{k=0}^{n-1} f_k \right| \geq \left| \sum_{k=0}^{m} f_k \right| - \left| \sum_{k=m+1}^{n-1} f_k \right| = |S_m| - \left| \sum_{k=m+1}^{n-1} f_k \right|$$

$$\geq 2 \cdot 2^{\exp(f'_m)} - (1 + 4\sqrt{\epsilon}) 2^{\exp(f'_m)} = (1 - 4\sqrt{\epsilon}) 2^{\exp(f'_m)} \tag{5.12}$$

By (5.12) and (5.10),

$$\left| \sum_{k=m+2}^{n-1} f_k \right| < (4\sqrt{\epsilon}) 2^{\exp(f'_m)} \leq \frac{4\sqrt{\epsilon}}{1 - 4\sqrt{\epsilon}} \left| \sum_{k=0}^{n-1} f_k \right| \tag{5.13}$$

By (5.12) and (5.11),

$$\left| \sum_{k=m+1}^{n-1} f_k \right| \leq \sum_{k=m+1}^{n-1} |f_k| \leq (1 + 4\sqrt{\epsilon}) 2^{\exp(f'_m)} \leq \frac{1 + 4\sqrt{\epsilon}}{1 - 4\sqrt{\epsilon}} \left| \sum_{k=0}^{n-1} f_k \right| \tag{5.14}$$

And by (5.12) and (5.14),

$$|S_m| \leq \left| \sum_{k=0}^{n-1} f_k \right| + \left| \sum_{k=m+1}^{n-1} f_k \right| \leq (1 + \frac{1 + 4\sqrt{\epsilon}}{1 - 4\sqrt{\epsilon}}) \left| \sum_{k=0}^{n-1} f_k \right| \leq \frac{2}{1 - 4\sqrt{\epsilon}} \left| \sum_{k=0}^{n-1} f_k \right|$$
$$\tag{5.15}$$

By definition, $\overline{S_{m+4}}$ is the computed sum of $\overline{S_m}, f_{m+1}, \ldots, f_{m+4}$ using the standard recursive summation technique. According to [4, Equation 1.2, 2.4]

$$\left| \overline{S_m} + \sum_{k=m+1}^{m+4} f_k - \overline{S_{m+4}} \right| \le \frac{4\epsilon}{1-4\epsilon} \left| \overline{S_m} + f_{m+1} \right| + \frac{3\epsilon}{1-3\epsilon} \sum_{k=m+2}^{m+4} |f_k|$$

$$\le \frac{4\epsilon}{1-4\epsilon} \left(\left| \overline{S_m} - S_m \right| + |S_m + f_{m+1}| \right) + \frac{3\epsilon}{1-3\epsilon} \sum_{k=m+2}^{n-1} |f_k|.$$

Since $S_{n-1} = S_m + f_{m+1} + \sum_{k=m+2}^{n-1} f_k$, we have

$$|S_m + f_{m+1}| = \left| S_{n-1} - \sum_{k=m+2}^{n-1} f_k \right| \le |S_{n-1}| + \sum_{k=m+2}^{n-1} |f_k|$$

Therefore

$$\left| \overline{S_m} + \sum_{k=m+1}^{m+4} f_k - \overline{S_{m+4}} \right| \le \frac{4\epsilon}{1-4\epsilon} \left| S_m - \overline{S_m} \right| + \frac{4\epsilon}{1-4\epsilon} |S_{n-1}| + \frac{7\epsilon}{1-4\epsilon} \sum_{k=m+2}^{n-1} |f_k|.$$

Using the Triangle Inequality we have

$$\left| S_{m+4} - \overline{S_{m+4}} \right| = \left| S_m + \sum_{k=m+1}^{m+4} f_k - \overline{S_{m+4}} \right| \le \left| S_m - \overline{S_m} \right| + \left| \overline{S_m} + \sum_{k=m+1}^{m+4} f_k - \overline{S_{m+4}} \right|$$

$$\le \left(1 + \frac{4\epsilon}{1-4\epsilon} \right) \left| S_m - \overline{S_m} \right| + \frac{4\epsilon}{1-4\epsilon} |S_{n-1}| + \frac{7\epsilon}{1-4\epsilon} \sum_{k=m+2}^{n-1} |f_k|$$

$$\le \frac{1}{1-4\epsilon} \epsilon |S_m| + \frac{4\epsilon}{1-4\epsilon} |S_{n-1}| + \frac{7\epsilon}{1-4\epsilon} \sum_{k=m+2}^{n-1} |f_k|$$

$$\le \frac{\epsilon}{1-4\epsilon} \left(|S_m| + 4 |S_{n-1}| + 7 \sum_{k=m+2}^{n-1} |f_k| \right).$$

and by (5.15) and (5.13),

$$\left| S_{m+4} - \overline{S_{m+4}} \right| \le \frac{\epsilon}{1-4\epsilon} \left(\frac{2}{1-4\sqrt{\epsilon}} |S_{n-1}| + 4 |S_{n-1}| + 7 \frac{4\sqrt{\epsilon}}{1-4\sqrt{\epsilon}} |S_{n-1}| \right)$$

$$\le \frac{\epsilon}{1-4\epsilon} \frac{6+12\sqrt{\epsilon}}{1-4\sqrt{\epsilon}} |S_{n-1}| = \frac{6\epsilon}{(1-2\sqrt{\epsilon})(1-4\sqrt{\epsilon})} |S_{n-1}|$$

$$< \frac{6\epsilon}{1-6\sqrt{\epsilon}} |S_{n-1}| \tag{5.16}$$

Notice that

$$\exp(f'_m) \geq \exp(f'_{m+2}) + \left\lceil \frac{p+1}{2} \right\rceil \geq \exp(f'_{m+4}) + 2\left\lceil \frac{p+1}{2} \right\rceil > \exp(f'_{m+5}) + 2\left\lceil \frac{p+1}{2} \right\rceil$$

Therefore,

$$\exp(f'_m) \geq \exp(f'_{m+5}) + p + 2 \tag{5.17}$$

Because $\exp(f'_0) > ... > \exp(f'_{n-1})$, (5.17) yields

$$\left| \sum_{k=m+5}^{n-1} f_k \right| \leq \sum_{k=m+5}^{n-1} |f_k| < \sum_{k=m+5}^{n-1} 2 \cdot 2^{\exp(f'_m) - p - 2 - (k-(m+5))} < \sum_{k=0}^{\infty} 2^{\exp(f'_m) - p - 1 - k} = \epsilon 2^{\exp(f'_m)}$$
$$\tag{5.18}$$

Using (5.12) and (5.18),

$$\left| \sum_{k=m+5}^{n-1} f_k \right| \leq \sum_{k=m+5}^{n-1} |f_k| < \frac{\epsilon}{1 - 4\sqrt{\epsilon}} \left| \sum_{k=0}^{n-1} f_k \right| \tag{5.19}$$

By (5.12), (5.16), and (5.18)

$$\left| \overline{S_{m+4}} \right| \geq |S_{m+4}| - \left| S_{m+4} - \overline{S_m + 4} \right| \geq \left(\left| \sum_{k=0}^{n-1} f_k \right| - \left| \sum_{k=m+5}^{n-1} f_k \right| \right) - \left| S_{m+4} - \overline{S_m + 4} \right|$$

$$= \left(\left| \sum_{k=0}^{n-1} f_k \right| - \left| S_{m+4} - \overline{S_m + 4} \right| \right) - \left| \sum_{k=m+5}^{n-1} f_k \right|$$

$$> (1 - \frac{6\epsilon}{1 - 6\sqrt{\epsilon}}) \left| \sum_{k=0}^{n-1} f_k \right| - \left| \sum_{k=m+5}^{n-1} f_k \right|$$

$$> (1 - \frac{6\epsilon}{1 - 6\sqrt{\epsilon}})(1 - 4\sqrt{\epsilon})2^{\exp(f'_m)} - \epsilon 2^{\exp(f'_m)}$$

$$= (1 - 4\sqrt{\epsilon} - \frac{6\epsilon(1 - 4\sqrt{\epsilon})}{1 - 2\sqrt{\epsilon}} - \epsilon)2^{\exp(f'_m)} \tag{5.20}$$

which, assuming $\epsilon << 1$, can be simplified to

$$\left| \overline{S_{m+4}} \right| > 2^{\exp(f'_m) - 1} \tag{5.21}$$

Using (5.17), for all $k \geq m + 5$ we have

$$|f_k| < 2 \cdot 2^{\exp(f'_k)} \leq 2 \cdot 2^{\exp(f_m)-p-2} = \epsilon \cdot 2^{\exp(f'_m)-1} \tag{5.22}$$

And by (5.22) and (5.21), all additions after f_{m+4} have no effect and we have

$$\overline{S_{m+4}} = \overline{S_{n-1}} \tag{5.23}$$

With (5.23), (5.19) and (5.16)

$$\left| S_{n-1} - \overline{S_{n-1}} \right| = \left| S_{n-1} - \overline{S_{m+4}} \right| \leq \left| S_{m+4} - \overline{S_{m+4}} \right| + \left| \sum_{k=m+5}^{n-1} f_k \right|$$

$$< \frac{6\epsilon}{1 - 6\sqrt{\epsilon}} \left| \sum_{k=0}^{n-1} f_k \right| + \frac{\epsilon}{1 - 4\sqrt{\epsilon}} \left| \sum_{k=0}^{n-1} f_k \right|$$

$$< \frac{7\epsilon}{1 - 6\sqrt{\epsilon}} \left| \sum_{k=0}^{n-1} f_k \right|$$

$$\square$$

Consider the K-fold indexed sum Y of floating point numbers $x_0, ..., x_{n-1}$. We denote the true sum $\sum_{k=0}^{n-1} x_k$ by T, the true value of the indexed sum as obtained using (4.4) by $\mathcal{Y}$, and the floating point approximation of $\mathcal{Y}$ obtained using an appropriate algorithm from Section 5.7 by $\overline{\mathcal{Y}}$.

[1] discusses the absolute error $|T - \mathcal{Y}|$ but does not give a method to construct $\overline{\mathcal{Y}}$ and therefore no error bound ($|T - \overline{\mathcal{Y}}|$) on the final floating point answer was given. Here we extend the error bound of [1] all the way to the final return value of the algorithm.

It has been shown in [1] that

$$|T - \mathcal{Y}| < n 2^{W(1-K)-1} \max |x_k| \tag{5.24}$$

However, this bound does not consider underflow. By (4.6), a small modification yields a bound that considers underflow

$$|T - \mathcal{Y}| < n \cdot \max(2^{W(1-K)-1} \max |x_k|, 2^{e_{\min}-1}) \tag{5.25}$$

By (5.5) and (5.6), Theorem 5.4 applies to yield

$$|\mathcal{Y} - \overline{\mathcal{Y}}| < \frac{7\epsilon}{1 - 6\sqrt{\epsilon}} |\mathcal{Y}|$$

By the triangle inequality

$$|\mathcal{Y}| \leq |T| + |T - \mathcal{Y}| < n \cdot \max(2^{W(1-K)-1} \max |x_k|, 2^{e_{\min}-1}) + |T|$$

The above results can be used to obtain (5.26), the absolute error of the floating point approximation of an indexed sum $|T - \overline{\mathcal{Y}}|$.

$$|T - \overline{\mathcal{Y}}| \leq |T - \mathcal{Y}| + |\mathcal{Y} - \overline{\mathcal{Y}}|$$

$$< n \cdot \max(2^{W(1-K)-1} \max |x_k|, 2^{e_{\min}-1}) + \frac{7\epsilon}{1 - 6\sqrt{\epsilon}} |\mathcal{Y}|$$

$$< n \cdot \max(2^{W(1-K)-1} \max |x_k|, 2^{e_{\min}-1})$$
$$+ \frac{7\epsilon}{1 - 6\sqrt{\epsilon}}(n \cdot \max(2^{W(1-K)-1} \max |x_k|, 2^{e_{\min}-1}) + |T|)$$

$$< (1 + \frac{7\epsilon}{1 - 6\sqrt{\epsilon}})(n \cdot \max(2^{W(1-K)-1} \max |x_k|, 2^{e_{\min}-1})) + \frac{7\epsilon}{1 - 6\sqrt{\epsilon}} |T| \quad (5.26)$$

Equation (5.26) can be approximated as (5.27).

$$|T - \overline{\mathcal{Y}}| < (1 + \frac{7\epsilon}{1 - 6\sqrt{\epsilon}})(n \cdot \max(2^{W(1-K)-1} \max |x_k|, 2^{e_{\min}-1})) + \frac{7\epsilon}{1 - 6\sqrt{\epsilon}} |T|$$

$$\approx n 2^{W(1-K)-1} \max |x_k| + 7\epsilon |T| \quad (5.27)$$

A perhaps more useful mathematical construction is the error expressed relative to the result $\overline{\mathcal{Y}}$, and not the theoretical sum T. Again by the triangle inequality,

$$|\mathcal{Y}| \leq |\overline{\mathcal{Y}}| + |\mathcal{Y} - \overline{\mathcal{Y}}|$$

Applying the bound on $|\mathcal{Y} - \overline{\mathcal{Y}}|$ yields

$$|\mathcal{Y}| < |\overline{\mathcal{Y}}| + \frac{7\epsilon}{1 - 6\sqrt{\epsilon}}|\mathcal{Y}|$$

After simplification,

$$|\mathcal{Y}| < \left(\frac{1}{1 - \frac{7\epsilon}{1 - 6\sqrt{\epsilon}}}\right)|\overline{\mathcal{Y}}|$$

$$< \frac{1 - 6\sqrt{\epsilon}}{1 - 6\sqrt{\epsilon} - 7\epsilon}|\overline{\mathcal{Y}}|$$

The above results can be used to obtain (5.28), the absolute error of the floating point approximation of an indexed sum $|T - \overline{\mathcal{Y}}|$.

$$\left|T - \overline{\mathcal{Y}}\right| \leq |T - \mathcal{Y}| + \left|\mathcal{Y} - \overline{\mathcal{Y}}\right|$$

$$< n \cdot \max(2^{W(1-K)-1}\max|x_k|, 2^{e_{\min}-1}) + \frac{7\epsilon}{1 - 6\sqrt{\epsilon}}|\mathcal{Y}|$$

$$< n \cdot \max(2^{W(1-K)-1}\max|x_k|, 2^{e_{\min}-1}) + \frac{7\epsilon}{1 - 6\sqrt{\epsilon}}\frac{1 - 6\sqrt{\epsilon}}{1 - 6\sqrt{\epsilon} - 7\epsilon}|\overline{\mathcal{Y}}|$$

$$< n \cdot \max(2^{W(1-K)-1}\max|x_k|, 2^{e_{\min}-1}) + \frac{7\epsilon}{1 - 6\sqrt{\epsilon} - 7\epsilon}|\overline{\mathcal{Y}}| \qquad (5.28)$$

Equation (5.28) can be approximated as (5.29).

$$\left|T - \overline{\mathcal{Y}}\right| < n \cdot \max(2^{W(1-K)-1}\max|x_k|, 2^{e_{\min}-1}) + \frac{7\epsilon}{1 - 6\sqrt{\epsilon} - 7\epsilon}|\overline{\mathcal{Y}}|$$

$$\approx n2^{W(1-K)-1}\max|x_k| + 7\epsilon\left|\overline{\mathcal{Y}}\right| \qquad (5.29)$$

We can compare (5.27) to the error bound obtained if the accumulator fields were summed without extra precision. In this case, only the standard summation bound from [4] would apply and the absolute error would be bounded by

$$n \cdot \max(2^{W(1-K)-1} \max|x_k|, 2^{e_{\min}-1}) + (\frac{(2K-1)\epsilon}{1-(2K-1)\epsilon}) \sum_0^{2K-1} |\gamma_k|$$

$$\leq n \cdot \max(2^{W(1-K)-1} \max|x_k|, 2^{e_{\min}-1}) + (\frac{(2K-1)\epsilon}{1-(2K-1)\epsilon})n \cdot \max|x_k|$$

$$\approx n \cdot \max|x_k| \, (2^{W(1-K)-1} + (2K-1)\epsilon) \qquad (5.30)$$

Which is not as tight a bound as (5.27), and grows linearly as the user increases K in an attempt to increase accuracy.

5.9 Limits

As discussed previously, for a K-fold indexed type the minimum K accepted by ReproBLAS is 2. The maximum useful K is $\lfloor(e_{\max} - e_{\min} + p - 1)/W\rfloor$, as this covers all of the bins.

As discussed in [1], $W < p-2$. As discussed in section 4.1.1, $2W > p+1$.

ReproBLAS uses the values $W = 40$ for indexed `double` and $W = 13$ for indexed `float`. W is available as the `XIWIDTH` macro.

As discussed in section 4.1.2, the input is rounded at best to the nearest $2^{e_{\min}-1}$

As absolute value of individual quantities added to Y_{jP} are not in excess of 2^{b_I+j}, a maximum of $0.25\epsilon^{-1}2^{-W}$ elements may be deposited into Y_{jP} between renormalizations, as discussed in section 5.4. For indexed `double` this number is 2^{11}, whereas for indexed `float` this number is 2^9. This number is supplied programmatically using the `XIENDURANCE` macro.

By (5.1), an indexed sum is capable of representing the sum of at least $0.25\epsilon^{-1}2^{-W}(\epsilon^{-1} - 1) \approx 2^{2p-W-2}$ floating point numbers. For indexed `double` this number is almost 2^{64}, whereas for indexed `float` this number is almost 2^{33}. This number is supplied programmatically using the `XICAPACITY` macro.

The indexed types provided by ReproBLAS will, when used correctly, avoid intermediate overflow.

6 Interface

6.1 Section Summary

6.2 Fold

6.3 Complex Types

6.4 Naming Conventions

6.5 Build System

7 indexed.h

7.1 Section Summary

7.2 Types

7.3 Functions

7.3.1 xixadd

7.3.2 xixiadd

7.3.3 xscale

7.3.4 xixiaddsq

8 idxdBLAS.h

8.1 Section Summary

8.2 Functions

8.2.1 xixdot

8.2.2 xixnrm2

8.2.3 xixgemv

8.2.4 xixgemm

8.3 Optimization

Due to the proportion of time spent in the deposit routine, optimization of the deposit routine was prioritized. In the event that multiple x need to be

added to Y, the deposit routine can be vectorized by accumulating the x in multiple copies of Y. The number of copies of Y to make, c, and the number of unrolled loop iterations, u are tuning parameters.

9   repBLAS.h

9.1   Section Summary

10   idxdMPI.h

10.1   Section Summary

10.2   Types

10.3   Functions

10.3.1   XIXIREDUCE

11   Applications

11.1   Section Summary

11.2   Parallel Reproducible Dot Product

11.3   Parallel Reproducible Vector Norm

11.4   Parallel Reproducible Matrix-Vector Multiply

11.5   Parallel Reproducible Matrix-Matrix Multiply

References

[1] Demmel, James, and Hong Diep Nguyen. Parallel Reproducible Summation. IEEE Transactions on Computers, 2014.

[2] IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2008 , vol., no., pp.1,70, Aug. 29 2008

[3] ANSI/ISO 9899-1990 American National Standard for Programming Languages - C, section 6.1.2.5

[4] Higham, Nicholas J. The accuracy of floating point summation. SIAM Journal on Scientific Computing 14, no. 4 (1993): 783-799.

[5] Demmel, James W., and Yozo Hida. Accurate floating point summation. Computer Science Division, University of California, 2002.