ReproBLAS: Much Summation! So Reproducible!

Peter Ahrens, Hong Diep Nguyen, James Demmel

September 28, 2015

Contents

1 Introduction

The algorithms for summation presented in [1] were shown to enjoy the following properties:

1. They compute a reproducible sum independent of the order of the summands, how they are assigned to processors, or how they are aligned in memory.

2. They make only basic assumptions about the underlying arithmetic (A subset of IEEE Standard 754-2008).

3. They scale well as a performance-optimized, non-reproducible implementation, as n (number of summands) and p (number of processors) grow.

4. The user can choose the desired accuracy of the result. In particular, getting a reproducible result with about the same accurace as the performance optimized algorithm is only be a small constant times slower, but higher accuracy is possible too.

Our goal is to modify the algorithms in [1] so that in addition to the above properties, they enjoy the following properties:

1. The algorithms can be applied in any binary IEEE 754-2008 floating point format, and any valid input in such a format can be summed using the algorithms. The algorithms must be able to sum numbers close to zero, numbers close to overflow, and exceptional values.

2. They must be expressed as basic operations that can be applied in several applications. They must be able to be built into a user-friendly, performant library.

2 Notation and Background

Let $\mathbb{R}$ and $\mathbb{Z}$ denote the sets of real 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 $\lceil r \rceil$ be the minimum element $z \in \mathbb{Z}$ such that $z \geq r$.

For all $r \in \mathbb{R}$, let $\lfloor r \rfloor$ 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. Properties of such rounding are shown in (2.2)

$$\begin{aligned} \left| r - \mathcal{R}_\infty(r,e) \right| &\leq 2^{e-1} \\ \mathcal{R}_\infty(r,e) &= 0 \text{ if } r < 2^{e-1}. \end{aligned} \tag{2.2}$$

Let $\mathbb{F}$ be the set of all floating-point numbers $f = sm2^e$ represented in some binary IEEE 754-2008 format [2] where $s \in \{1,-1\}$ is the **sign**, $e \in \mathbb{Z}$, $e_{\text{max}} \geq e \geq e_{\text{min}}$ is the **exponent** ($\exp(f)$ is defined to be e), $p \in \mathbb{Z}, p > 1$ is the **precision**, and $m = m_0.m_1m_2...m_{p-1}$ where $m_0, ..., m_{p-1} \in \{0,1\}$ is the **significand** of f. Assume that the representation of f is made unique using the "hidden bit" convention, so that f is represented using the smallest exponent possible. (In memory, the bit m_0 is not stored and assumed to be 1 unless the exponent field contains a special value signaling both that $e = e_{\text{min}}$ and $m_0 = 0$). f is said to be **normalized** if $m_0 = 1$ and $e \geq e_{\text{min}}$, **unnormalized** if $m_0 = 0$, and **denormalized** if $m_0 = 0$ and $e = e_{\text{min}}$. $f = 0$ if all $m_j = 0$ and $e = e_{\text{min}}$.

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.

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

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

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

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 $\mathrm{fl}(f_0 \text{ op } f_1) = (f_0 \text{ op } f_1)$. If rounding is "to nearest," then we have that $|\mathrm{fl}(f_0 \text{ op } f_1) - (f_0 \text{ op } f_1)| \leq 0.5\mathrm{ulp}(\mathrm{fl}(f_0 \text{ op } f_1))$.

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 exponent 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 mathematical theory to describe the components (referred to as slices) corresponding to each bin. 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.

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

$$a_i = e_{\max} + 1 - (i+1)W \text{ for } 0 \leq i \leq i_{\max} \tag{3.2}$$

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

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 **less** 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,

$(a_0, b_0]$, is

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

and the least bin, $(a_{i_{\max}}, b_{i_{\max}}]$, is

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

Section 4.1.2 explains why the bottom of the exponent range

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

is ignored.

As discussed in [1], and explained again in Section 5.3, 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.

$$\begin{aligned} d(x,0) &= \mathcal{R}_\infty(x, a_0+1) \\ d(x,i) &= \mathcal{R}_\infty\big(x - \sum_{j=0}^{i-1} d(x,j), a_i+1\big) \text{ for } i > 0. \end{aligned} \tag{3.8}$$

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, $|x| < 2^{a_0}$, by (2.2) we have $d(x,0) = \mathcal{R}_\infty(x, a_0+1) = 0$.

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

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

Again, since $x < 2^{a_{i+1}}$, by (2.2) we have $d(x,i+1) = \mathcal{R}_\infty(x, a_{i+1}+1) = 0$. □

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

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

□

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 $x \in \mathbb{F}$.

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

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 $x \in \mathbb{F}$, $|x - \sum\limits_{j=0}^{i} d(x,j)| \le 2^{a_i}$.

Proof. We apply (2.2) and (3.9)

$$\begin{aligned} \big|x - \sum_{j=0}^{i} d(x,j)\big| &= \Big|\big(x - \sum_{j=0}^{i-1} d(x,j)\big) - d(x,i)\Big| \\ &= \Big|\big(x - \sum_{j=0}^{i-1} d(x,j)\big) - \mathcal{R}_\infty\big(x - \sum_{j=0}^{i-1} d(x,j), a_i + 1\big)\Big| \le 2^{a_i} \end{aligned}$$

□

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 $|x - \sum\limits_{j=0}^{i-1} d(x,j)| \le 2^{b_i}$.

If $i = 0$, then we have

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

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

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

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

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

□

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 $x_0, ..., x_{n-1} \in \mathbb{F}$. Y is referred to as the **indexed sum** of $x_0, ..., x_{n-1}$.

Each accumulator Y_k is a data structure used to accumulate the slices of input in the bin $(a_{I+k}, b_{I+k}]$ where I is the **index** of Y and $k \geq 0$ 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 $b_I > \max(e_0, ..., e_n)$ where $e_i = \exp(x_i)$). If Y has index I, then $Y_k, k \in \{0, ..., K-1\}$ accumulates slices of input in the bin $(a_{I+k}, b_{I+k}]$. 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 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_k of an indexed type Y are each implemented using two underlying floating point fields. The **primary** field Y_{kP} is used during accumulation, while the **carry** field Y_{kC} 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_{kP} and carry field Y_{kC} corresponds to the "S_j" and "C_j" of Algorithm 6 in [1].

The numerical value $\mathcal{Y}_{kP}$ represented by data stored in the primary field Y_{kP} is an offset from $1.5\epsilon^{-1}2^{a_{I+k}}$ (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.

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

This simplifies the process of extracting the slices of input in bins $(a_{I+k}, b_{I+k}]$. It will be shown in Theorem 5.1 in Section 5.2 that if we represent each primary value $\mathcal{Y}_{kP}$ as in (4.1) and keep Y_{kP} within the range $(\epsilon^{-1}2^{a_{I+k}}, 2\epsilon^{-1}2^{a_{I+k}})$, then Algorithm 5.3 in Section 5.2 extracts the slices of x in bins $(a_I, b_I], ..., (a_{I+K-1}, b_{I+K-1}]$ and adds them to $Y_{0P}, ..., Y_{K-1P}$ without error (and hence reproducibly) for all $x \in \mathbb{F}$, where $|x| < 2^{b_I}$.

Because $d(x, I+k) = 0$ for bins with $|x| < 2^{a_{I+k}}$, 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. Since the maximum absolute value operation is always reproducible, so is the index of the greatest accumulator.

In order to keep the primary fields in the necessary range while the slices are accumulated and to keep the representation of Y_k unique, Y_{kP} is routinely renormalized to the range $[1.5\epsilon^{-1}2^{a_{I+k}}, 1.75\epsilon^{-1}2^{a_{I+k}})$. As will be shown in Section 5.3, that renormalization is required every 2^{p-W-2} iterations, so 2^{11} in double and 2^9 in single precision. Which means the renormalization introduces a very low overhead to the overall running time. To renormalize,

Y_{kP} is incremented or decremented by $0.25\epsilon^{-1}2^{a_{I+k}}$ when necessary, leaving the carry field Y_{kC} to record the number of such adjustments. The numerical value $\mathcal{Y}_{kC}$ represented by data stored in the carry field Y_{kC} of an indexed type Y of index I is expressed in (4.2)

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

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

$$\mathcal{Y}_k = \mathcal{Y}_{kP} + \mathcal{Y}_{kC} = (Y_{kP} - 1.5\epsilon^{-1}2^{a_{I+k}}) + (0.25\epsilon^{-1}2^{a_{I+k}})Y_{kC} \tag{4.3}$$

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

$$\mathcal{Y} = \sum_{k=0}^{K-1} \mathcal{Y}_k = \sum_{k=0}^{K-1} \big((Y_{kP} - 1.5\epsilon^{-1}2^{a_{I+k}}) + (0.25\epsilon^{-1}2^{a_{I+k}})Y_{kC}\big) \tag{4.4}$$

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. As will be explained in Section 5.1, 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 $W < p-2$, 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_{kP} such that $I + k \geq 1$, $|Y_{kP}| < 2^{e_{\max}}$.

Proof. $a_1 = e_{\max} + 1 - 2W$ by (3.2), therefore $a_1 < e_{\max} - p$ using $2W > p+1$ and since all quantities are integers, $a_1 \leq e_{\max} - p - 1$. If $I + k \geq 1$, $a_{I+k} \leq a_1 \leq e_{\max} - p - 1$ by (3.2).

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

$$|Y_{kP}| < 2\epsilon^{-1}2^{a_{I+k}} \leq 2^{1+p}2^{e_{\max}-1-p} = 2^{e_{\max}}$$

□

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).

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

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 for adding a floating point input to an indexed type in Section 5.2 and Algorithm 5.5 for renormalizing an indexed type in Section 5.3 require that the primary fields Y_{kP} 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_{kP} of an indexed type Y of index I where $Y_{kP} \in (\epsilon^{-1}2^{a_{I+k}}, 2\epsilon^{-1}2^{a_{I+k}})$ ($Y_{0P} \in (2^{e_{\max}}, 2 \cdot 2^{e_{\max}})$ if Y has index 0), Y_{kP} is normalized.

Proof. By (3.5),

$$a_{I+k} \geq a_{i_{\max}} = e_{\min} - p + 2 + \big((e_{\max} - e_{\min} + p - 1) \mod W\big) \geq e_{\min} - p + 2$$

Because $Y_{kP} \in (\epsilon^{-1}2^{a_{I+k}}, 2\epsilon^{-1}2^{a_{I+k}})$ we have $\exp(Y_{kP}) = a_{I+k} + p > e_{\min} + 1$ so Y_{kP} is normalized. □

Algorithm 5.4 relies on setting the last bit of intermediate results before adding them to Y_{kP} in order to fix the direction of the rounding mode. However, if r is the quantity to be added to Y_{kP}, $\text{ulp}(r)$ must be less than rounding error when added to Y_{kP}. Mathematically, we will require $\text{ulp}(r) < 0.5\text{ulp}(Y_{kP})$ in order to prove Theorem 5.2 about the correctness of Algorithm 5.4. This is why we must enforce $a_{i_{\max}} \geq e_{\min} - p + 2$ so that the least significant bit of the least bin is larger than twice the smallest denormalized number.

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

Using $W < p - 2$ and (3.5),

$$\begin{aligned} a_{i_{\max}} &= e_{\min} - p + 2 + \big((e_{\max} - e_{\min} + p - 1) \mod W\big) \\ &\le e_{\min} - p + 2 + (W - 1) \\ &< e_{\min} - p + 2 + (p - 2 - 1) = e_{\min} - 1 \end{aligned}$$

Hence,

$$a_{i_{\max}} \le e_{\min} - 2$$

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

$$\Big|x - \sum_{i=0}^{i_{\max}} d(x,i)\Big| \le 2^{a_{i_{\max}}} \le 2^{e_{\min}-2} \tag{4.6}$$

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

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 most straightforward 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).

Since the Y_{kP} are kept within the range $(\epsilon^{-1}2^{a_{I+k}}, 2\epsilon^{-1}2^{a_{I+k}})$ (where I is the index) and are normalized (by Theorem 4.2), we have

$$Y_{kP} > 2^{e_{\min}}$$

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`s have been added to Y or one or more of both `Inf` and `-Inf` have been added to Y.

As the Y_{kP} 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 5.6.

5 Primitive Operations

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 5.1. Theoretical summaries of algorithms (with some improvements) from [1] are provided in Sections 5.4, 5.2, 5.3, 5.5. Section 5.6 provides an original algorithm (with an improved error bound) to obtain the value represented by an indexed type. Sections 5.7 and 5.8 extend analysis in [1] to the new algorithms.

5.1 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)
2:     return $\lfloor (e_{\max} + p - \exp(Y_{0P}) - W + 1)/W \rfloor$
3: end function
Ensure:
    Returned result $I$ 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. This uses the assumption that $\frac{p+1}{2} < W < p-2$, so $3 < p-W+1 < W$.

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)
2:     return $\lfloor (e_{\max} - \exp(x))/W \rfloor$
3: end function
Ensure:
    The bin $(a_J, b_J]$ satisfies $2^{b_J} > |x| \geq 2^{a_J}$.
```

5.2 Deposit

The deposit operation (here referred to as Algorithm 5.4, which deals with overflow, unlike a simpler version 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.

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).
$|x| < 2^{b_I}$.
$Y_{kP} \in (\epsilon^{-1}2^{a_{I+k}}, 2\epsilon^{-1}2^{a_{I+k}})$ at all times. (The carry fields Y_{kC} will be used to ensure this in Algorithm 5.5)

```
1: function DepositRestricted(K, x, Y)
2:     $r = x$
3:     for $k = 0$ to $(K - 2)$ do
4:         $S = Y_{kP} + (r|1)$
5:         $q = S - Y_{kP}$
6:         $Y_{kP} = S$
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_{kP} by this algorithm is exactly $d(x, I + k)$.

The last bit of r is set to break ties when rounding "to nearest" so that the amount added to Y_{kP} does not depend on the size of Y_{kP} 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_{kP} is exactly $d(x, I + k)$.

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

Note that lines 4-7 correspond to Algorithm 4 of [1]. Therefore, if $\mathrm{ulp}(Y_{kP}) = \mathrm{ulp}(S_k)$ and $\mathrm{ulp}(r_k) < 0.5\mathrm{ulp}(Y_{kP})$, Corollary 3 of [1] applies and we have that $S_k - Y_{kP} \in \mathrm{ulp}(Y_{kP})\mathbb{Z} \in 2^{a_{I+k}+1}\mathbb{Z}$ and that $|r_{k+1}| \leq 0.5\mathrm{ulp}(Y_{kP}) = 2^{a_{I+k}}$.

As it is assumed $Y_{kP}, S_k \in (\epsilon^{-1}2^{a_{I+k}}, 2\epsilon^{-1}2^{a_{I+k}})$, we have $\mathrm{ulp}(S_k) = \mathrm{ulp}(Y_{kP})$ for all k.

We show $|r_k| \leq 2^{b_{I+k}} = 2^{a_{I+k-1}}$ for all k inductively. As a base case, $r_0 = x$ (from line 2) so $|r_0| = |x| < 2^{b_I}$. As an inductive step, assume $|r_k| \leq 2^{b_{I+k}}$. We must show $\mathrm{ulp}(r_k) < 0.5\mathrm{ulp}(Y_{kP})$.

By Theorem 4.2 we have that Y_{kP} is normalized and therefore $\mathrm{ulp}(Y_{kP}) = 2^{a_{I+k}+1}$. If r_k is normalized, then because $\mathrm{ulp}(r_k) \leq 2^{1-p}|r_k| \leq 2^{b_{I+k}-(p-1)} = 2^{a_{I+k}+W-(p-1)}$, and $W < p-2$, we have $\mathrm{ulp}(r_k) \leq 2^{a_{I+k}-3} < 0.5\mathrm{ulp}(Y_{kP})$. (This case is considered in [1]). If r_k is denormalized, $\mathrm{ulp}(r_k) = 2^{e_{\min}-p+1}$ since the unit in the last place of a denormalized number is always equal to $2^{e_{\min}-p+1}$. Using (3.5), $\mathrm{ulp}(r_k) = 2^{e_{\min}-p+1} \leq 2^{e_{\min}-p+1+((e_{\max}-e_{\min}+p-1) \mod W)} = 2^{a_{i_{\max}}-1} \leq 2^{a_{I+k}-1} < 0.5\mathrm{ulp}(Y_{kP})$.

Therefore we have $\mathrm{ulp}(r_k) < 0.5\mathrm{ulp}(Y_{kP})$. Thus, Corollary 3 of [1] applies and we have that $|r_{k+1}| \leq 0.5\mathrm{ulp}(Y_{kP}) = 2^{a_{I+k}}$. This completes the induction.

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

If $|r_k - (S_k - Y_{kP})| < 2^{a_{I+k}}$, then $S_k - Y_{kP} = \mathcal{R}_\infty(r_k, a_{I+k}+1)$.

If $|r_k - (S_k - Y_{kP})| = 2^{a_{I+k}}$, then $S_k - Y_{kP} \in \{r_k + 2^{a_{I+k}}, r_k - 2^{a_{I+k}}\}$. As $S_k = \mathrm{fl}(Y_{kP} + (r_k|1))$, we have $|S_k - Y_{kP} - (r_k|1)| \leq 0.5\mathrm{ulp}(S_k) = 2^{a_{I+k}}$. As $\mathrm{ulp}(S_k) = \mathrm{ulp}(Y_{kP}) = 2^{a_{I+k}+1}$, we also have that $r_k \in 2^{a_{I+k}}\mathbb{Z}$ and because $\mathrm{ulp}(r_k) < 2^{a_{I+k}}$, $|(r_k|1) - r_k| > 0$ (with $(r_k|1) - r_k$ taking the same sign as r_k). If $r_k > 0$, then $(S_k - Y_{kP}) = r_k + 2^{a_{I+k}}$ (otherwise we will have $|S_k - Y_{kP} - (r_k|1)| = |r_k - 2^{a_{I+k}} - (r_k|1)| > 2^{a_{I+k}}$). If $r_k < 0$, then $(S_k - Y_{kP}) = r_k - 2^{a_{I+k}}$ (otherwise we will have $|S_k - Y_{kP} - (r_k|1)| = |r_k + 2^{a_{I+k}} - (r_k|1)| > 2^{a_{I+k}}$). Therefore, $S_k - Y_{kP} = \mathcal{R}_\infty(r_k, a_{I+k}+1)$.

We can now show $r_{k+1} = x - \sum\limits_{i=0}^{I+k} d(x,i)$ and $S_k - Y_{kP} = d(x, I+k)$ for all k by induction on k.

In the base case, $S_0 - Y_{0P} = \mathcal{R}_\infty(r_0, a_I+1) = \mathcal{R}_\infty(x, a_I+1)$. As $|x| < 2^{b_I}$, 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), $|x| < 2^{b_I} \leq 2^{a_i}$ for all $i \in \{0, ..., I-1\}$, and therefore by Lemma 3.1, $r_1 = x - \sum\limits_{i=0}^{I} d(x,i)$.

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

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

And by Theorem 3 of [1],

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

□

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, i.e. when overflow is an issue. 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:      $I$ = IIndex(Y)
3:      if I = 0 then
4:          $r = x/2^{p-W+1}$
5:          $S = Y_{0P} + (r|1)$
6:          $q = S - Y_{0P}$
7:          $Y_{0P} = S$
8:          $q = q \cdot 2^{p-W}$
9:          $r = x - q$
10:         $r = r - q$
11:         for $k = 1$ to $(K - 2)$ do
12:             $S = Y_{kP} + (r|1)$
13:             $q = S - Y_{kP}$
14:             $Y_{kP} = S$
15:             $r = r - q$
16:         end for
17:         $Y_{K-1P} = Y_{K-1P} + (r|1)$
18:     else
19:         DepositRestricted(K, x, Y)
20:     end if
21: end function
```

Ensure:

No overflow occurs during the algorithm.
The amount added to Y_{kP} is exactly $d(x, I + k)$ if $I + k \neq 0$.
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 19. 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 5) 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 4.

Since $Y_{0P}, S \in (2^{e_{\max}}, 2 \cdot 2^{e_{\max}})$, we have $\mathrm{ulp}(Y_{0P}) = \mathrm{ulp}(S)$ in line 5. As $|r| < 2^{e_{\max} - p + W}$ and $W < p - 2$, we have $\mathrm{ulp}(r) \leq |r| 2^{1-p} < 2^{e_{\max}+1-2p+W} < 0.5\mathrm{ulp}(Y_{0P})$ in line 5.

Because line 5 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 $|r - (S - Y_{0P})| \leq 0.5\mathrm{ulp}(Y_{0P}) = 2^{a_0 - p + W - 1}$ in line 5.

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

If $|r - (S - Y_{0P})| = 2^{a_0 - p + W - 1}$, then $S - Y_{0P} \in \{r + 2^{a_0 - p + W - 1}, r_k - 2^{a_0 - p + W - 1}\}$. As $S = \mathrm{fl}(Y_{0P} + (r|1))$, we have $|S - Y_{0P} - (r|1)| \leq 0.5\mathrm{ulp}(S) = 2^{a_0 - p + W - 1}$. 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 as 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 $S - Y_{0P} = d(x, 0)/2^{p - W + 1}$.

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

If $|x| \geq 2^{a_0} = 2^{e_{\max}+1-W}$, then there is no underflow in line 4 and $r = x/2^{p - W + 1}$ exactly as we scale by a power of two. Therefore, $S - Y_{0P} = \mathcal{R}_\infty(x/2^{p - W + 1}, a_0 - p + W) = d(x, 0)/2^{p - W + 1}$

At this point, all that remains to be shown is that $r = x - d(x, 0)$ in line 10. In line 6, we have that $q = S - Y_{0P} = d(x, 0)/2^{p - W + 1}$. By Theorem 3.4, $d(x, 0) \leq 2 \cdot 2^{e_{\max}}$. We then have that in line 8, 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 9 and line 10.

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 $x \leq 2^{b_0}$, $d(x, 0)/2 \in \mathrm{ulp}(x)\mathbb{Z}$ and therefore $x - d(x, 0)/2 \in \mathrm{ulp}(x)\mathbb{Z}$. Combined with $|x - d(x, 0)/2| \leq |x|$ this implies that $x - d(x, 0)/2$ is representable, and that $r = x - q$ exactly in line 9.

Again since $d(x, 0)/2, x - d(x, 0)/2 \in \mathrm{ulp}(x)\mathbb{Z}$, $x - d(x, 0) \in \mathrm{ulp}(x)\mathbb{Z}$ and since $|x - d(x, 0)| \leq |x|$, $x - d(x, 0)$ is representable and $r = r - q$ exactly in line 10.

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. $\square$

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 9).

5.3 Renormalize

When depositing values into a K-fold indexed type Y of index I, the assumption is made that $Y_{kP} \in (\epsilon^{-1}2^{a_{I+k}}, 2\epsilon^{-1}2^{a_{I+k}})$ 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 $Y_{kP} \in [1.25\epsilon^{-1}2^{a_{I+k}}, 2\epsilon^{-1}2^{a_{I+k}})$, renormalize Y such that $Y_{kP} \in [1.5\epsilon^{-1}2^{a_{I+k}}, 1.75\epsilon^{-1}2^{a_{I+k}})$.

```
Require:
    $Y_{kP} \in [1.25\epsilon^{-1}2^{a_{I+k}}, 2\epsilon^{-1}2^{a_{I+k}})$
 1: function RENORM(K, Y)
 2:     for $k = 0$ to $K - 1$ do
 3:         if $Y_{kP} < 1.5 \cdot \mathrm{ufp}(Y_{kP})$ then
 4:             $Y_{kP} = Y_{kP} + 0.25 \cdot \mathrm{ufp}(Y_{kP})$
 5:             $Y_{kC} = Y_{kC} - 1$
 6:         end if
 7:         if $Y_{kP} \geq 1.75 \cdot \mathrm{ufp}(Y_{kP})$ then
 8:             $Y_{kP} = Y_{kP} - 0.25 \cdot \mathrm{ufp}(Y_{kP})$
 9:             $Y_{kC} = Y_{kC} + 1$
10:         end if
11:     end for
12: end function
Ensure:
    $Y_{kP} \in [1.5\epsilon^{-1}2^{a_{I+k}}, 1.75\epsilon^{-1}2^{a_{I+k}})$.
    The values $\mathcal{Y}_k$ are unchanged. Recall that by (4.3),
```

$$\mathcal{Y}_k = \mathcal{Y}_{kP} + \mathcal{Y}_{kC} = (Y_{kP} - 1.5\epsilon^{-1}2^{a_{I+k}}) + (0.25\epsilon^{-1}2^{a_{I+k}})Y_{kC}$$

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 $\max |x_j| < 2^{b_I}$. If Y initially satisfies $Y_{kP} \in [1.5\epsilon^{-1}2^{a_{I+k}}, 1.75\epsilon^{-1}2^{a_{I+k}})$ and $n \leq 2^{p-W-2}$, then after all of the deposits, $Y_{kP} \in [1.25\epsilon^{-1}2^{a_{I+k}}, 2\epsilon^{-1}2^{a_{I+k}})$.

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

By Theorem 5.2, the deposit operation extracts and adds the slices of x_j exactly (assuming $Y_{kP} \in (\epsilon^{-1}2^{a_{I+k}}, 2\epsilon^{-1}2^{a_{I+k}})$ at each step, which will be shown),

$$\Big|\sum_{n=0}^{j-1} d(x_j, I+k)\Big| \leq n2^{b_{I+k}} = n2^{W}2^{a_{I+k}}$$

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

$$\begin{aligned} Y_{kP} &\in \big[(1.5\epsilon^{-1} - n2^{W})2^{a_{I+k}}, (1.75\epsilon^{-1} + n2^{W})2^{a_{I+k}}\big) \\ &\in [1.25\epsilon^{-1}2^{a_{I+k}}, 2\epsilon^{-1}2^{a_{I+k}}) \end{aligned}$$

□

If an indexed type Y initially satisfies $Y_{kP} \in [1.5\epsilon^{-1}2^{a_{I+k}}, 1.75\epsilon^{-1}2^{a_{I+k}})$ (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, $Y_{kP} \in [1.25\epsilon^{-1}2^{a_{I+k}}, 2\epsilon^{-1}2^{a_{I+k}})$. Therefore, after another renormalization, the primary fields would once again satisfy $Y_{kP} \in [1.5\epsilon^{-1}2^{a_{I+k}}, 1.75\epsilon^{-1}2^{a_{I+k}})$. The limit of at most 2^{p-W-2} numbers can be accumulated before having to carry out a renormalization also requires that $p - W - 2 > 0$, or

$$W < p - 2. \tag{5.1}$$

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

$$0.25\epsilon^{-1}2^{-W}(\epsilon^{-1} - 1) \approx 2^{2p-W-2} \tag{5.2}$$

floating point numbers. It means 2^{64} in double and 2^{31} in single precision using the values in Table 3.1.

5.4 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:
    $Y_{0P} \in (\epsilon^{-1}2^{a_I}, 2\epsilon^{-1}2^{a_I})$
 1: function UPDATE(K, x, Y)
 2:     $I$ = IINDEX(Y)
 3:     $J$ = INDEX(x)
 4:     if $J < I$ then
 5:         $[Y_{\min(I-J,K)_P}, ..., Y_{K-1_P}] = [Y_{0P}, ..., Y_{K-1-\min(I-J,K)_P}]$
 6:         $[Y_{0P}, ..., Y_{\min(I-J,K)-1_P}] = [1.5\epsilon^{-1}a_J, ..., 1.5\epsilon^{-1}a_{\min(I,K+J)-1}]$
 7:         $[Y_{\min(I-J,K)_C}, ..., Y_{K-1_C}] = [Y_{0C}, ..., Y_{K-1-\min(I-J,K)_C}]$
 8:         $[Y_{0C}, ..., Y_{\min(I-J,K)-1_C}] = [0, ..., 0]$
 9:     end if
10: end function
Ensure:
    The bin $(a_J, b_J]$ satisfies $2^{b_J} > |x| \geq 2^{a_J}$.
    The new index of $Y$, $(\min(I, J))$ is such that $2^{b_{\min(I,J)}} > |x|$.
    Existing accumulators of $Y$ are shifted towards index 0, losing the lesser
    bins.
    New accumulators are shifted into $Y$ with value 0 in $Y_{kC}$ and $1.5\epsilon^{-1}2^{a_{\min(I,J)+k}}$
    in $Y_{kP}$.
```

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

If Y is the K-fold indexed sum of some $x_0, ..., x_{n-1}$ with index I, then the shift described by Algorithm 5.6 produces an indexed type of index $J < I$ with the value represented by accumulator k satisfying

$$\mathcal{Y}_k = \sum_{j=0}^{n-1} d(x_j, J + k)$$

The new accumulators Y_k with $0 \leq k < I - J$ must represent 0 because $x_j < 2^{a_{J+k}} \leq 2^{a_I}$ so $\sum\limits_{j=0}^{n-1} d(x_j, J + k) = 0$ by Lemma 3.1.

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 $1.5\epsilon^{-1}a_j$ for all valid $j \in Z$ are stored in a precomputed array.

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

5.5 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 $x_n, ..., x_{n+m-1} \in F$.
    $Y_{kP} \in [1.5\epsilon^{-1}2^{a_{I+k}}, 1.75\epsilon^{-1}2^{a_{I+k}})$
    $Z_{kP} \in [1.5\epsilon^{-1}2^{a_{J+k}}, 1.75\epsilon^{-1}2^{a_{J+k}})$
 1: function REDUCE(K, Y, Z)
 2:     $I$ = IINDEX(Y)
 3:     $J$ = IINDEX(Z)
 4:     if $J < I$ then
 5:         $R = Z$
 6:         REDUCE(K, $R$, $Y$)
 7:         $Y = R$
 8:     end if
 9:     $k = J - I$
10:     while $k < K$ do
11:         $Y_{kP} = Y_{kP} + Z_{k+I-J_P} - 1.5\epsilon^{-1}2^{a_{I+k}}$
12:         $Y_{kC} = Y_{kC} + Z_{k+I-J_C}$
13:         $k = k + 1$
14:     end while
15:     RENORM(K, $Y$)
16: end function
Ensure:
    $Y$ is set to the indexed sum of $x_0, ..., x_{n+m-1}$.
    $Y_{kP} \in [1.5\epsilon^{-1}2^{a_{\min(I,J)+k}}, 1.75\epsilon^{-1}2^{a_{\min(I,J)+k}})$
```

5.6 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 $2^{b_I} > |x| \geq 2^{a_I}$ $Y_{kP} \in [1.5\epsilon^{-1}2^{a_{I+k}}, 1.75\epsilon^{-1}2^{a_{I+k}})$

We must renormalize the indexed type because $d(x, I+k)$ could be negative, meaning $Y_{kP} < 1.5\epsilon^{-1}2^{a_{I+k}}$ 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 (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 (using (5.30))

$$|\overline{\mathcal{Y}} - \sum_{j=0}^{n-1} x_j| \leq n\big(2^{W(1-K)} + (2K-1)\epsilon\big) \max |x_j| \tag{5.3}$$

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

$$|\overline{\mathcal{Y}} - \sum_{j=0}^{n-1} x_j| \leq n2^{W(1-K)} \max |x_j| + 7\epsilon|\sum_{j=0}^{n-1} x_j| \tag{5.4}$$

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])

$$|S_n - \sum_{j=0}^{n-1} x_j| \leq n\epsilon \sum_{j=0}^{n-1} |x_j| \leq n^2 \epsilon \max |x_j| \tag{5.5}$$

Figure 5.1 compares these three approximate error bounds.

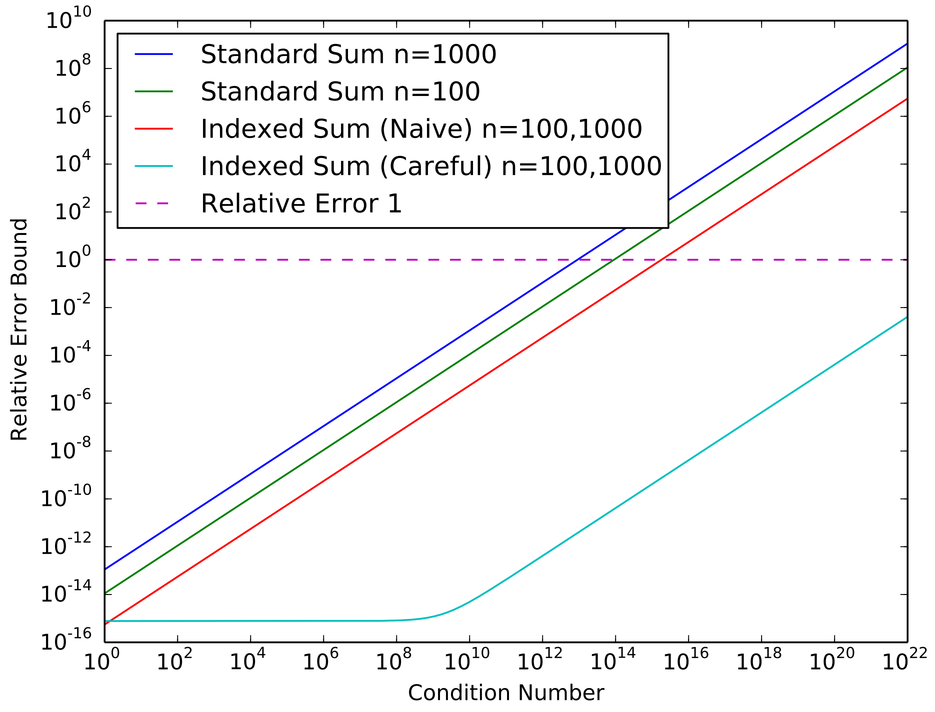

Figure 5.1: Relative error bounds in calculating $|\sum_{j=0}^{n-1} x_j|$ for different condition numbers (which we define as $\frac{n \cdot \max|x_j|)}{|\sum_{j=0}^{n-1} x_j|}$) of the sum. It is assumed that we sum using `double`, $K = 3$, and $W = 40$. "Indexed Summation (Careful)" corresponds to (5.4). "Indexed Summation (Naive)" corresponds to (5.3). "Standard Summation" corresponds to (5.5) 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 $\mathcal{Y}_{kP}$ in a primary field Y_{kP} is represented by an offset from $1.5\epsilon^{-1}2^{a_{I+k}}$ and $Y_{kP} \in (\epsilon^{-1}2^{a_{I+k}}, 2\epsilon^{-1}2^{a_{I+k}})$, $\mathcal{Y}_{kP}$ can be expressed exactly using an unnormalized floating point number $\mathcal{Y}'_{Pk}$ with an exponent of $a_{I+k}+p-1$. As each carry field Y_{kC} is a count of renormalization adjustments later scaled by $0.25\epsilon^{-1}2^{a_{I+k}}$, $\mathcal{Y}_{kC}$ can be expressed exactly using an unnormalized floating point number $\mathcal{Y}'_{kC}$ with an exponent of $a_{I+k} + 2p - 3$.

First, we have $\exp(\mathcal{Y}'_{kP}) > \exp(\mathcal{Y}'_{k+1P})$ and $\exp(\mathcal{Y}'_{kC}) > \exp(\mathcal{Y}'_{k+1C})$ because $a_{I+k} > a_{I+k+1}$.

Next, note that

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

and

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

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

Finally, note that

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

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

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

$$\begin{aligned}\exp(\mathcal{Y}'_{0C}) > \exp(\mathcal{Y}'_{1C}) &> \exp(\mathcal{Y}'_{0P}) > \exp(\mathcal{Y}'_{2C}) > \exp(\mathcal{Y}'_{1P}) > ...\\ ... > \exp(\mathcal{Y}'_{kC}) &> \exp(\mathcal{Y}'_{k-1P}) > \exp(\mathcal{Y}'_{k+1C}) > \exp(\mathcal{Y}'_{kP}) > ...\\ ... > \exp(\mathcal{Y}'_{K-2C}) &> \exp(\mathcal{Y}'_{K-3P}) > \exp(\mathcal{Y}'_{K-1C}) > \exp(\mathcal{Y}'_{K-2P}) > \exp(\mathcal{Y}'_{K-1P})\end{aligned}$$

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

$$\exp(\gamma'_0) > ... > \exp(\gamma'_{2K-1}) \tag{5.6}$$

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

It should be noted that if γ_j is a primary field, then either γ_{j+1} or γ_{j+2} is a primary field. If γ_j is a carry field, then either γ_{j+1} or γ_{j+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 $j \in \{0, ..., 2K - 1\}$

$$\exp(\gamma'_j) \geq \exp(\gamma'_{j+2}) + W \geq \exp(\gamma'_{j+2}) + \left\lceil\frac{p+1}{2}\right\rceil \tag{5.7}$$

It should be noted that the $\mathcal{Y}'_{kP}$ and the $\mathcal{Y}'_{kC}$ can be expressed exactly using floating point types of the same precision as Y_{kP} and Y_{kC} (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 $|\gamma'_j| \leq 2^{e_{\max}-W+2p-1-j}$. The absolute value represented by an indexed type can therefore be bounded by

$$\sum_{j=0}^{2K-1} |\gamma_j| < \sum_{j=0}^{2K-1} 2^{e_{\max}-W+2p-1-j} < \sum_{j=0}^{\infty} 2^{e_{\max}-W+2p-1-j} = 2^{e_{\max}-W+2p} \quad (5.8)$$

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 $k = 1$ to $K - 1$ do
4:         $z = z + \mathcal{Y}_{kC}$
5:         $z = z + \mathcal{Y}_{k-1P}$
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_j 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 $\mathcal{Y}_{kP}$ or $\mathcal{Y}_{kC}$ with $|\mathcal{Y}_{kP}| < 0.5 \cdot 2^{-\rho} 2^{e_{\max}}$ or $|\mathcal{Y}_{kC}| < 0.5 \cdot 2^{-\rho} 2^{e_{\max}}$ (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 $\exp(\gamma_j') < e_{\max} - \rho - 1$, 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_j' 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:     $k = 1$
3:     while $k \leq 2K$ and $\exp(\gamma_k) \geq e_{\max} - \rho - 1$ do
4:         $z = z + (\gamma_k / 2^{2p-W-2})$
5:         $k = k + 1$
6:     end while
7:     $z = z \cdot 2^{2p-W-2}$
8:     while $k \leq 2K$ do
9:         $z = z + \gamma_k$
10:        $k = k + 1$
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.

5.7 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. $f_j = f'_j$ for all $j \in \{0, ..., n-1\}$
2. $\exp(f'_0) > ... > \exp(f'_{n-1})$
3. $\exp(f'_j) \geq \exp(f'_{j+2}) + \lceil \frac{p+1}{2} \rceil$ for all $j \in \{0, ..., n-3\}$

Let $S_0 = \overline{S_0} = f_0$, $S_j = S_{j-1} + f_j$, and $\overline{S_j} = \text{fl}(\overline{S_{j-1}} + f_j)$ so that $S_{n-1} = \sum\limits_{j=0}^{n-1} f_j$. Then we have

$$\left|S_{n-1} - \overline{S_{n-1}}\right| < \frac{7\epsilon}{1 - 6\sqrt{\epsilon}} |S_{n-1}| \approx 7\epsilon |S_{n-1}|$$

Proof. Throughout the proof, let $f_j = 0$ if $j > n-1$ 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 $\exp(f'_0) > ... > \exp(f'_m)$, $f_0, ..., f_m \in \text{ulp}(f'_m)\mathbb{Z}$. Thus, $S_m \in \text{ulp}(f'_m)\mathbb{Z}$.

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 $S_m \in \text{ulp}(f'_m)\mathbb{Z}$, 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

$$|S_m| > 2 \cdot 2^{\exp(f'_m)} \tag{5.9}$$

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

$$|f_{m+1}| < 2 \cdot 2^{\exp(f'_m - 1)} = 2^{\exp(f'_m)} \tag{5.10}$$

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

$$|\sum_{j=m+2}^{n-1} f_j| \leq \sum_{j=m+2}^{n-1} |f_j| < \sum_{j=m+2}^{n-1} 2 \cdot 2^{\exp(f'_j)} \leq \sum_{j=m+2}^{n-1} 2 \cdot 2^{\exp(f'_m) - \lceil \frac{p+1}{2} \rceil - (m+2-j)}$$
$$< \sum_{j=0}^{\infty} \left(2\sqrt{\epsilon}\right) 2^{\exp(f'_m) - j} = \left(4\sqrt{\epsilon}\right) 2^{\exp(f'_m)} \tag{5.11}$$

We can combine (5.10) and (5.11) to obtain

$$|\sum_{j=m+1}^{n-1} f_j| \leq \sum_{j=m+1}^{n-1} |f_j| < 2^{\exp f'_m} + \left(4\sqrt{\epsilon}\right) 2^{\exp(f'_m)} = \left(1 + 4\sqrt{\epsilon}\right) 2^{\exp(f'_m)} \tag{5.12}$$

By (5.9) and (5.12),

$$|S_{n-1}| = |\sum_{j=0}^{n-1} f_j| \geq |\sum_{j=0}^{m} f_j| - |\sum_{j=m+1}^{n-1} f_j| = |S_m| - |\sum_{j=m+1}^{n-1} f_j|$$
$$\geq 2 \cdot 2^{\exp(f'_m)} - \left(1 + 4\sqrt{\epsilon}\right) 2^{\exp(f'_m)} = \left(1 - 4\sqrt{\epsilon}\right) 2^{\exp(f'_m)} \tag{5.13}$$

By (5.13) and (5.11),

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

By (5.13) and (5.12),

$$|\sum_{j=m+1}^{n-1} f_j| \leq \sum_{j=m+1}^{n-1} |f_j| \leq \left(1 + 4\sqrt{\epsilon}\right) 2^{\exp(f'_m)} \leq \frac{1 + 4\sqrt{\epsilon}}{1 - 4\sqrt{\epsilon}} |\sum_{j=0}^{n-1} f_j| \tag{5.15}$$

And by (5.13) and (5.15),

$$|S_m| \leq |\sum_{j=0}^{n-1} f_j| + |\sum_{j=m+1}^{n-1} f_j| \leq \left(1 + \frac{1 + 4\sqrt{\epsilon}}{1 - 4\sqrt{\epsilon}}\right) |\sum_{j=0}^{n-1} f_j| \leq \frac{2}{1 - 4\sqrt{\epsilon}} |\sum_{j=0}^{n-1} f_j| \tag{5.16}$$

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

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

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

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

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

Therefore

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

Using the Triangle Inequality we have

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

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

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

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

and by (5.16) and (5.14),

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

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

$$< \frac{6\epsilon}{1-6\sqrt{\epsilon}}|S_{n-1}| \qquad (5.17)$$

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,

$$\exp(f'_m) \geq \exp(f'_{m+5}) + p + 2 \tag{5.18}$$

Because $\exp(f'_0) > ... > \exp(f'_{n-1})$, (5.18) yields

$$|\sum_{j=m+5}^{n-1} f_j| \leq \sum_{j=m+5}^{n-1} |f_j| < \sum_{j=m+5}^{n-1} 2 \cdot 2^{\exp(f'_m)-p-2-(j-(m+5))} < \sum_{j=0}^{\infty} 2^{\exp(f'_m)-p-1-j} = \epsilon 2^{\exp(f'_m)} \tag{5.19}$$

Using (5.13) and (5.19),

$$|\sum_{j=m+5}^{n-1} f_j| \leq \sum_{j=m+5}^{n-1} |f_j| < \frac{\epsilon}{1-4\sqrt{\epsilon}} |S_{n-1}| \tag{5.20}$$

By (5.17) and (5.20)

$$\begin{aligned} \left|S_{n-1} - \overline{S_{m+4}}\right| &\leq |S_{n-1} - S_{m+4}| + \left|S_{m+4} - \overline{S_{m+4}}\right| \\ &\leq |\sum_{j=m+4}^{n-1} f_j| + \frac{6\epsilon}{1-6\sqrt{\epsilon}} |S_{n-1}| \\ &\leq \frac{\epsilon}{1-4\sqrt{\epsilon}} |S_{n-1}| + \frac{6\epsilon}{1-6\sqrt{\epsilon}} |S_{n-1}| \\ &< \frac{7\epsilon}{1-6\sqrt{\epsilon}} |S_{n-1}|. \end{aligned} \tag{5.21}$$

When combined with (5.13) gives

$$\begin{aligned} \left|\overline{S_{m+4}}\right| &\geq \left(1 - \frac{7\epsilon}{1-6\sqrt{\epsilon}}\right) |S_{n-1}| \\ &> \left(1 - \frac{7\epsilon}{1-6\sqrt{\epsilon}}\right) \left(1 - 4\sqrt{\epsilon}\right) 2^{\exp(f'_m)} \\ &> \left(1 - 4\sqrt{\epsilon} - \frac{7\epsilon\left(1-4\sqrt{\epsilon}\right)}{1-6\sqrt{\epsilon}}\right) 2^{\exp(f'_m)} \end{aligned}$$

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

$$\left|\overline{S_{m+4}}\right| > 2^{\exp(f'_m)-1} \tag{5.22}$$

Using (5.18), for all $j \geq m+5$ we have

$$|f_j| < 2 \cdot 2^{\exp(f'_j)} \leq 2 \cdot 2^{\exp(f_m)-p-2} = \epsilon \cdot 2^{\exp(f'_m)-1} \tag{5.23}$$

And by (5.23) and (5.22), all additions after f_{m+4} have no effect and we have $\overline{S_{n-1}} = \overline{S_{m+4}}$. This, together with (5.21), implies

$$\left|S_{n-1} - \overline{S_{n-1}}\right| \leq \frac{7\epsilon}{1-6\sqrt{\epsilon}}|S_{n-1}|$$

The proof is complete. □

Consider the K-fold indexed sum Y of floating point numbers $x_0, \ldots, x_{n-1}$. We denote the true sum $\sum_{j=0}^{n-1} x_j$ 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 5.6 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.

Let I be the index of Y, which is also the index of $\max|x_j|$, we have $2^{b_I} > \max|x_j| \geq 2^{a_I}$. Therefore for all $i < I$ the slice of x_j in bin i is $d(x_j, i) = 0$. The index of the smallest bin of Y is $I + K - 1$. According to Theorem 3.3, we have

$$\begin{aligned}|x_j - \sum_{j=I}^{I+K-1} d(x,i)| = |x_j - \sum_{j=0}^{I+K-1}| &\leq 2^{a_{I+K-1}} = 2^{a_I-(K-1)W} \\ &\leq 2^{W(1-K)} \max|x_j|.\end{aligned}$$

Since the summation in each bin Y_i is exact, we have

$$\begin{aligned}|T - \mathcal{Y}| = |\sum_{j=0}^{n-1} x_j - \sum_{i=I}^{I+K-1}\sum_{j=0}^{n-1} d(x_j,i)| &= |\sum_{j=0}^{n-1}(x_j - \sum_{i=I}^{I+K-1} d(x_j,i)| \\ &\leq n2^{W(1-K)} \max|x_j|.\end{aligned} \tag{5.24}$$

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

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

By (5.6) and (5.7), Theorem 5.4 applies to yield

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

By the triangle inequality

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

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

$$\begin{aligned}
\bigl|T - \overline{\mathcal{Y}}\bigr| &\leq |T - \mathcal{Y}| + \bigl|\mathcal{Y} - \overline{\mathcal{Y}}\bigr| \\
&< n \cdot \max\bigl(2^{W(1-K)} \max|x_j|, 2^{e_{\min}-1}\bigr) + \frac{7\epsilon}{1 - 6\sqrt{\epsilon}}|\mathcal{Y}| \\
&< n \cdot \max\bigl(2^{W(1-K)} \max|x_j|, 2^{e_{\min}-1}\bigr) \\
&\quad + \frac{7\epsilon}{1 - 6\sqrt{\epsilon}}\Bigl(n \cdot \max\bigl(2^{W(1-K)-1} \max|x_j|, 2^{e_{\min}-1}\bigr) + |T|\Bigr) \\
&< \left(1 + \frac{7\epsilon}{1 - 6\sqrt{\epsilon}}\right)\Bigl(n \cdot \max\bigl(2^{W(1-K)} \max|x_j|, 2^{e_{\min}-1}\bigr)\Bigr) + \frac{7\epsilon}{1 - 6\sqrt{\epsilon}}|T|
\end{aligned} \tag{5.26}$$

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

$$\begin{aligned}
\bigl|T - \overline{\mathcal{Y}}\bigr| &< \left(1 + \frac{7\epsilon}{1 - 6\sqrt{\epsilon}}\right)\Bigl(n \cdot \max\bigl(2^{W(1-K)} \max|x_j|, 2^{e_{\min}-1}\bigr)\Bigr) + \frac{7\epsilon}{1 - 6\sqrt{\epsilon}}|T| \\
&\approx n2^{W(1-K)} \max|x_j| + 7\epsilon|T|
\end{aligned} \tag{5.27}$$

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,

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

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

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

After simplification,

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

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

$$\begin{aligned}\left|T - \overline{\mathcal{Y}}\right| &\le |T - \mathcal{Y}| + \left|\mathcal{Y} - \overline{\mathcal{Y}}\right| \\ &< n \cdot \max\left(2^{W(1-K)} \max|x_j|, 2^{e_{\min}-1}\right) + \frac{7\epsilon}{1-6\sqrt{\epsilon}}|\mathcal{Y}| \\ &< n \cdot \max\left(2^{W(1-K)} \max|x_j|, 2^{e_{\min}-1}\right) + \frac{7\epsilon}{1-6\sqrt{\epsilon}}\left(\frac{1-6\sqrt{\epsilon}}{1-6\sqrt{\epsilon}-7\epsilon}\,|\overline{\mathcal{Y}}|\right) \\ &< n \cdot \max\left(2^{W(1-K)} \max|x_j|, 2^{e_{\min}-1}\right) + \frac{7\epsilon}{1-6\sqrt{\epsilon}-7\epsilon}\,\left|\overline{\mathcal{Y}}\right| \qquad (5.28)\end{aligned}$$

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

$$\begin{aligned}|T - \overline{\mathcal{Y}}| &< n \cdot \max\left(2^{W(1-K)} \max|x_j|, 2^{e_{\min}-1}\right) + \frac{7\epsilon}{1-6\sqrt{\epsilon}-7\epsilon}\,\left|\overline{\mathcal{Y}}\right| \\ &\approx n2^{W(1-K)} \max|x_j| + 7\epsilon\left|\overline{\mathcal{Y}}\right| \qquad (5.29)\end{aligned}$$

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

$$n \cdot \max\left(2^{W(1-K)} \max|x_j|, 2^{e_{\min}-1}\right) + \left(\frac{(2K-1)\epsilon}{1-(2K-1)\epsilon}\right)\sum_{0}^{2K-1}|\gamma_j|$$

which is approximately bounded by

$$n \cdot \max|x_j|\left(2^{W(1-K)} + (2K-1)\epsilon\right) \qquad (5.30)$$

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

5.8 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_{kP} are not in excess of $2^{b_{I+k}}$, a maximum of $0.25\epsilon^{-1}2^{-W}$ elements may be deposited into Y_{kP} between renormalizations, as discussed in section 5.3. 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 (5.2), 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 Composite Operations

Although several reproducible summation algorithms can be built from the set of primitive operations defined in section 5, it may be unclear how these operations can be composed to form such algorithms. What follows are the specifications for common summation-based algorithms in terms of the primitive operations. To obtain a general completely reproducible algorithm for summation that is, one must design for reproducibility under both data permutation and reduction tree shape. Section 6.1 details a general reproducible summation algorithm that is independent of input data ordering, providing analysis of its correctness and runtime. Section **??** shows how the algorithm in Section **??** may be extended to arbitrary reduction tree shapes, and gives another proof of correctness. Sections 6.2, **??**, **??**, and **??** explain how to obtain reasonably performant reproducible versions of representative operations from the BLAS. Implementation details of these routines are discussed in Section **??**.

6.1 Sum

Algorithm 6.1 is an indexed summation algorithm that produces the indexed sum of a vector of floating point numbers $x_0, ..., x_{n-1} \in \mathbb{F}$. The indexed sum of a list of numbers is independent of the ordering of the list. Algorithm 6.1 uses only one indexed type to hold the intermediate result of the recursive summation, and the vast majority of time in the algorithm is spent in the deposit routine.

Algorithm 6.1. Return the K-fold indexed sum of $x_0, ..., x_{n-1}$. This is similar to Algorithm 6 in [1], but requires no restrictions on the inputs $x_0, ..., x_{n-1}$.

```
function Sum(K, [$x_0, ..., x_{n-1}$])
    $Y = 0$
    $block = 0$
    $j = 0$
    while $j < n$ do
        $m = \min(n, j + 0.25\epsilon^{-1}2^{-W})$
        Update(K, max([$|x_j|, ..., |x_{m-1}|$]), Y)
        while $j < m$ do
            Deposit(K, $x_j$, Y)
            $j = j + 1$
        end while
        Renorm(K, Y)
    end while
    return $Y$
end function
```

Ensure:

Y is the unique indexed sum of $x_0, ..., x_{n-1}$. This is equivalent to the following three statements.

The index I of Y is the greatest integer such that $\max(|x_j|) < 2^{b_I}$.

$Y_{kP} \in [1.5\epsilon^{-1}2^{a_{I+k}}, 1.75\epsilon^{-1}2^{a_{I+k}})$ unless $I + k = 0$, in which case $Y_{0P} \in (2^{e_{\max}}, 2 \cdot 2^{e_{\max}})$

$\mathcal{Y}_k = \sum_{j=0}^{n-1} d(x_j, I + k)$

In less specific terms, what this algorithm ensures is that the fields of Y represent the unique indexed sum of Y. If a floating point result is desired, it may be obtained from Y using Algorithm 5.10

Theorem 6.1. Assume that we have run Algorithm 6.1 on n floating point numbers $x_0, ..., x_{n-1} \in \mathbb{F}$ with some K. If all requirements of the algorithm are satisfied, then the "Ensure" claim at the end of the algorithm holds.

Proof. We show inductively that after each execution of line 12, Y is the indexed sum of $x_0, ..., x_{j-1}$.

In the first iteration of the loop on line 5, Y is set to 0, meaning all of its fields are set to zero. In subsequent iterations of the loop, since we have at line 7 that Y is the indexed sum of $x_0, ..., x_{j-1}$, we know that $Y_{kP} \in [1.5\epsilon^{-1}2^{a_{I+k}}, 1.75\epsilon^{-1}2^{a_{I+k}})$ unless $I + k = 0$, in which case $Y_{0P} \in (2^{e_{\max}}, 2 \cdot 2^{e_{\max}})$.

In either case, the "Require" clause of Algorithm 5.6 is satisfied. Therefore, after executing line 7, the following statements hold:

1. The index of Y is I where I is the greatest integer such that $\max(|x_0|, ..., |x_{m-1}|) < 2^{b_I}$
2. $\mathcal{Y}_k = d(x_0, I + k) + ... + d(x_{j-1}, I + k)$
3. $Y_{kP} \in [1.5\epsilon^{-1}2^{a_{I+k}}, 1.75\epsilon^{-1}2^{a_{I+k}})$ unless $I + k = 0$, in which case $Y_{0P} \in (2^{e_{\max}}, 2 \cdot 2^{e_{\max}})$

Between lines **??** and **??**, at most $0.25\epsilon^{-1}2^{-W}$ calls to **??** are made, and by Theorem **??**, and the above initial properties, the requirements of Algorithm 5.4 are met at each call. Therefore, after line **??**, we have that $\mathcal{Y}_k = d(x_0, I + k) + ... + d(x_{j-1}, I + k)$.

Again by Theorem **??**, be the time line 12 is executed, the requirements of Algorithm 5.5 are met. Therefore, after execution of line 12, we have that $Y_{kP} \in [1.5\epsilon^{-1}2^{a_{I+k}}, 1.75\epsilon^{-1}2^{a_{I+k}})$ unless $I + k = 0$, in which case $Y_{0P} \in (2^{e_{\max}}, 2 \cdot 2^{e_{\max}})$. Therefore, Y is the indexed sum of $x_0, ..., x_{j-1}$, completing the induction. □

6.2 Euclidean Norm

7 Interface

7.1 Section Summary

7.2 Fold

7.3 Complex Types

7.4 Naming Conventions

7.5 Build System

8 indexed.h

8.1 Section Summary

8.2 Types

8.3 Functions

8.3.1 xixadd

8.3.2 xixiadd

8.3.3 xscale

8.3.4 xixiaddsq

9 idxdBLAS.h

9.1 Section Summary

9.2 Functions

9.2.1 xixdot

9.2.2 xixnrm2

9.2.3 xixgemv

9.2.4 xixgemm

9.3 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.

10 repBLAS.h

10.1 Section Summary

11 idxdMPI.h

11.1 Section Summary

11.2 Types

11.3 Functions

11.3.1 XIXIREDUCE

12 Applications

12.1 Section Summary

12.2 Parallel Reproducible Dot Product

12.3 Parallel Reproducible Vector Norm

12.4 Parallel Reproducible Matrix-Vector Multiply

12.5 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.