Fast Reproducible Floating-Point Summation

James Demmel
Mathematics Department and CS Division
University of California at Berkeley
Berkeley, CA 94720
demmel@eecs.berkeley.edu

Hong Diep Nguyen
EECS Department
University of California at Berkeley
Berkeley, CA 94720
hdnguyen@eecs.berkeley.edu

***Abstract*—Reproducibility, i.e. getting the bitwise identical floating point results from multiple runs of the same program, is a property that many users depend on either for debugging or correctness checking in many codes [1]. However, the combination of dynamic scheduling of parallel computing resources, and floating point nonassociativity, make attaining reproducibility a challenge even for simple reduction operations like computing the sum of a vector of numbers in parallel. We propose a technique for floating point summation that is reproducible independent of the order of summation. Our technique uses Rump's algorithm for error-free vector transformation [2], and is much more efficient than using (possibly very) high precision arithmetic. Our algorithm trades off efficiency and accuracy: we reproducibly attain reasonably accurate results (with an absolute error bound $c \cdot n^2 \cdot macheps \cdot max|v_i|$ for a small constant c) with just 2n + O(1) floating-point operations, and quite accurate results (with an absolute error bound $c \cdot n^3 \cdot macheps^2 \cdot max|v_i|$ with 5n + O(1) floating point operations, both with just two reduction operations. Higher accuracies are also possible by increasing the number of error-free transformations. As long as the same rounding mode is used, results computed by the proposed algorithms are reproducible for any run on any platform.**

***Keywords*-reproducibility; summation; floating-point; rounding error; parallelism;**

I. Introduction

The non-reproducibility of floating-point computation arises because the associative law is no longer guaranteed in floating-point arithmetic. Therefore results of floating-point sums or products depend on the order of computation. In this paper we are interested in obtaining reproducibility for floating-point summation. Let $v = [v_1, \ldots, v_n]$ be a vector of n floating-point numbers. A conventional way to compute the sum of all elements of v is given by Algorithm 1.

In Algorithm 1, the $\texttt{Reduce}(s, \texttt{SUM})$ operation aggregates all the partial sums from all the processors using the $\texttt{SUM}$ operation to produce the final sum. When the processor count differs from run to run then the length of each local vector as well as the number of partial sums are different too. It means the order of evaluation of the global sum is different. In addition, there can also be various ways to implement the reduction such as a flat tree or binary tree, which can also lead to different orders of evaluation.

Techniques have been proposed to address the problem of

Algorithm 1 Conventional parallel sum

Require: V is the global vector of size N, v is the local vector of length n_p on the p-th processor, $\sum n_p = N$
  $s = 0$
  for $i = 1$ to n_p **do** ▷ Local sum
    $s = s + v_i$
  end for
  $\texttt{Reduce}(s, \texttt{SUM})$ ▷ Aggregate global sum
Ensure: s is the sum of all elements in V

non-associativity of floating point computation for massively multi-threaded system [1] which is based on a deterministic parallel tree scheme, as well as for GPUs [3] which is based on extra precision. At extreme scale, a naive use of these techniques can lead to either dramatic increase of computing time (extra precision) or substantial communication overhead (deterministic tree scheme). Using an arbitrarily high precision can certainly help to compute a very precise result (at a high overhead of computing time of course). Nevertheless, when failing to compute the exact summation one cannot always avoid the Table-Maker's Dilemma, where a very small error in evaluation can lead to different rounded value. That may only happen with a very small probability but can lead to non-deterministic result.

Our goal is to compute a rigorously reproducible sum of floating-point numbers regardless of the order of evaluation as well as the *ill-conditioning* of the problem. The main idea is to "pre-round" the input floating-point numbers to some common base so that their sum can be computed accurately without any rounding error. The same technique is found in [4], [2] to compute an accurate floating-point summation.

It should be noted that reproducibility is not equivalent to accuracy. On one hand, computing a highly accurate result does not guarantee reproducibility, i.e. computed results might be close to the exact result but not identical. On the other hand, a reproducible result might be far away from the exact result. Our proposed technique trades off between accuracy and efficiency depending of the required accuracy as well as the availability of rounding modes. If only rounding to nearest even mode is available then one can choose to use a fast but less accurate algorithm which costs $4n + \mathcal{O}(1)$

operations with an error bound $c \cdot n^2 \cdot macheps \cdot \max |v_i|$ or a slower but more accurate algorithm which costs $8n + \mathcal{O}(1)$ operations with an error bound $c \cdot n^3 \cdot macheps^2 \cdot \max |v_i|$. Otherwise if a directed rounding mode is available then one can use a faster algorithm which costs $2n + \mathcal{O}(1)$ operations with an absolute error bound $c \cdot n^2 \cdot macheps \cdot \max |v_i|$, or a more accurate algorithm which costs $5n + \mathcal{O}(1)$ operations with an absolute error bound $c \cdot n^3 \cdot macheps^2 \cdot \max |v_i|$. All algorithms require only 2 reduction operations.

The paper is organized as follows. Section II presents some notation and numerical analysis that will be used throughout the paper. Section III presents the error-free vector transformation algorithms proposed by Rump [4], [2]. Section IV presents our new algorithms to compute the reproducible sum. Section V contains some numerical experiments to check the reproducibility as well as the accuracy of the proposed algorithm. Section VI contains conclusions.

II. Notation and background

Denote by $\mathbb{R}, \mathbb{F}, \mathbb{Z}$ the set of real numbers, floating-point numbers and integers respectively.

In this paper we assume that the floating-point arithmetic in use complies with the IEEE 754-2008 standard with five rounding modes: rounding to nearest even, rounding to nearest with tie-breaking away from 0, rounding toward 0, rounding toward $+\infty$, and rounding toward $-\infty$. To distinguish from the two rounding to nearest modes, the last 3 rounding modes are called directed roundings.

Let $f = s \times 2^e \times m \in \mathbb{F}$ be a floating-point number represented in *IEEE 754* format, where $s = \pm 1$ is the sign, $e_{MAX} \geq e \geq e_{MIN}$ is the exponent (usually refered as $\texttt{exponent}(f)$), p is the precision, and $m = m_0.m_1m_2 \ldots m_{p-1}, m_i \in \{0, 1\}$, is the significand of f. f is said to be normalized if $m_0 = 1$ and $e > e_{MIN}$. $f = 0$ if all $m_i = 0$ and $e = e_{MIN}$. $f \neq 0$ is said to be subnormal if $m_0 = 0$ and $e = e_{MIN}$.

The unit in the last place, denoted by $\texttt{ulp}(f)$, represents the spacing between two consecutive floating-point numbers of the same exponent e: $\texttt{ulp}(f) = 2^e \times 2^{1-p} = 2^{e-(p-1)}$.

Machine epsilon ϵ is the spacing between 1 and the next smaller floating-point number: $\epsilon = 2^{-p}$.

The unit roundoff $\mathbf{u}$ is the upper bound on the relative error due to rounding. $\mathbf{u} = \epsilon$ for rounding to nearest, and $\mathbf{u} = 2\epsilon$ for directed rounding.

$\texttt{fl}(\cdot)$ denotes the evaluated result of an expression in floating-point arithmetic.

For any normalized floating-point number $f \in \mathbb{F}$, and $r \in \mathbb{R}$ we have the following properties:

(a) $|r - \texttt{fl}(r)| < \texttt{ulp}(\texttt{fl}(r)) < 2\epsilon|\texttt{fl}(r)|$,
(b) $|r - \texttt{fl}(r)| \leq \mathbf{u} \times |\texttt{fl}(r)|$ and $|r - \texttt{fl}(r)| \leq \mathbf{u} \times |r|$,
(c) $\frac{1}{2}\epsilon^{-1}\texttt{ulp}(f) \leq |f| < \epsilon^{-1}\texttt{ulp}(f)$,

Another way to represent f is $f = M * \texttt{ulp}(f)$, $M \in \mathbb{Z}$. $2^{p-1} \leq |M| < 2^p$ for normalized numbers and $0 < |M| < 2^{p-1}$ for subnormal numbers. Moreover, for any $n \in \mathbb{Z}$ and $|n| < 2^p = \epsilon^{-1}$ then $n * \texttt{ulp}(f) \in \mathbb{F}$. Let $x \in \mathbb{R}$ be a real number, denote $x\mathbb{Z} = \{n * x \mid n \in \mathbb{Z}\}$. If $a, b \in x\mathbb{Z}$ then $a \pm b \in x\mathbb{Z}$, letting us deduce the following lemmas.

Lemma 1. *Let $f, g \in \mathbb{F}$. If $|g| \geq |f|$ then $g \in \texttt{ulp}(f)\mathbb{Z}$.*

Lemma 2. *Let $f, x, y \in \mathbb{F}$ where $x, y \in \texttt{ulp}(f)\mathbb{Z}$. If $|x + y| < \epsilon^{-1}\texttt{ulp}(f)$ then $x + y \in \mathbb{F}$, i.e. $x + y$ can be computed exactly.*

Let $x, y \in \mathbb{F}$. If there is no overflow or underflow in evaluating $\texttt{fl}(x \circ y)$, $\circ \in \{+, -, \times, /\}$, then

$$\texttt{fl}(x \circ y) = (x \circ y)(1 + \delta), \quad |\delta| \leq \mathbf{u} \tag{1}$$

which is the standard model for numerical analysis of an algorithm. For the following theorem, as well as for other numerical analysis throughout the paper, we assume that there is no overflow or underflow during execution. The effect of overflow and underflow will be discussed later, at the end of Section IV.

Theorem 1 (Error bound of conventional sum)**.** *[5, Section 4.2] Let s be result computed by Algorithm 1. Then*

$$|s - \sum_{i=1}^{n} v_i| < (n-1) \cdot \mathbf{u} \cdot \sum_{i=1}^{n} |v_i| + \mathcal{O}(\mathbf{u}^2). \tag{2}$$

Theorem 1 provides the error bound of the conventional sum. The goal of this paper is to provide techniques to compute a reproducible sum which is almost of the same accuracy as the conventional sum, i.e. having an error bound of the same order as (2).

III. Error-free vector transformation

In this section, we summarize the error-free vector transformations proposed by Rump[4], [2], which consist of transforming a vector into a sum of high order parts of all elements and a vector of trailing parts of all elements. Basically, each element is split into a high order part and a low order part using Algorithm 2, which is based on the `FastTwoSum` algorithm by Dekker [6].

Algorithm 2 Extract scalar: $[S, q, r] = \texttt{FastTwoSum}(M, x)$

Require: $M, x \in \mathbb{F}, M \geq |x|$
1: $S = \texttt{fl}(M + x)$
2: $q = \texttt{fl}(S - M)$
3: $r = \texttt{fl}(x - q)$

Algorithm 2 is called an error-free transformation because it computes the sum of $S = \texttt{fl}(M + x)$ together with its rounding error $r = (M + x) - \texttt{fl}(M + x)$. Nevertheless, beside the requirement of sorting absolute values $|M| \geq |x|$, it must also be performed in round to nearest mode.

Theorem 2 (Algorithm 2 in rounding to nearest)**.** *Let $S, q, r \in \mathbb{F}$ be the results computed by Algorithm 2. Then*

$q = S - M$, $x = q + r, M + x = S + r$, $q \in \frac{1}{2}\mathtt{ulp}(M)\mathbb{Z}$, *and* $|r| \le \frac{1}{2}\mathtt{ulp}(S) \le \mathtt{ulp}(M) \le 2\mathbf{u}M$.

Proof: The proof of this theorem can be found in [7, Lemma 2.6] by replacing $\epsilon\mathtt{ufp}(S)$ with $\frac{1}{2}\mathtt{ulp}(S)$, where $\mathtt{ufp}$ denotes the unit in the first place: $\mathtt{ufp}(S) = 2^{\mathtt{exponent}(S)}$. ∎

In directed rounding, Algorithm 2 is no longer an error-free transformation. For example in rounding mode toward 0, if $M = 1$ and $x = -2^{-110}$ then $S = \mathtt{fl}(M+x) = 1-2^{-53}$, $q = \mathtt{fl}(S - M) = -2^{-53}$, $r = \mathtt{fl}(x - q) = 2^{-53}$, so $q+r = 0 \neq x$. Theorem 3 provides some other requirements for Algorithm 2 to be error-free in directed rounding.

Theorem 3 (Algorithm 2 in directed rounding)**.** *Let* $x, M \in \mathbb{F}$ *which satisfy* $M \ge |x|$. *Let* $S, q, r \in \mathbb{F}$ *be results computed by Algorithm 2. Then*

(a) $q = S - M, q \in \frac{1}{2}\mathtt{ulp}(M)\mathbb{Z}$, *and* $|r| \le \mathtt{ulp}(S) \le 2\mathtt{ulp}(M) \le 2\mathbf{u}M$,
(b) $|x - (q + r)| < 2\mathbf{u} \cdot \mathtt{ulp}(M) \le 2\mathbf{u}^2 \cdot M$,
(c) *if* $|x| \ge 4\mathtt{ulp}(M)$ *then* $x = q + r$ *and* $M + x = S + r$.

Proof: Since $M \ge |x|$, then $0 \le M + x \le 2M$. So $0 \le S \le 2M$ and $\mathtt{ulp}(S) \le 2\mathtt{ulp}(M) \le 2\mathbf{u}M$. We distinguish two cases: $-M \le x \le -\frac{1}{2}M$ and $-\frac{1}{2}M < x \le M$:

(i) $-M \le x \le -\frac{1}{2}M$. According to Sterbenz's theorem [5, Theorem 2.5] $S = \mathtt{fl}(M + x) = M + x$. Therefore $q = x$, and $r = 0$. All the properties are verified.

(ii) $-\frac{1}{2}M < x \le M$. This implies $\frac{1}{2}M \le S \le 2M$. Therefore $q = \mathtt{fl}(S - M) = S - M$. $S \ge \frac{1}{2}M$ also implies $S \in \frac{1}{2}\mathtt{ulp}(M)\mathbb{Z}$. So $q = S - M \in \frac{1}{2}\mathtt{ulp}(M)\mathbb{Z}$. Let $e = x - q = (M+x) - S = (M+x) - \mathtt{fl}(M+x)$. Then $|e| < \mathtt{ulp}(S)$ and $r = \mathtt{fl}(e)$. That implies $|r| \le \mathtt{ulp}(S)$. Therefore $|x - (q+r)| = |\mathtt{fl}(e) - e| < \mathbf{u} \cdot |e| < \mathbf{u} \cdot \mathtt{ulp}(S) \cdot \le 2\mathbf{u} \cdot \mathtt{ulp}(M) \le 2\mathbf{u}^2 \cdot M$.
In addition, if $|x| \ge 4\mathtt{ulp}(M)$ then $|x| \ge 2\mathtt{ulp}(S) > 2|e|$. Therefore $q = x + e$ is of the same sign as x and $\frac{1}{2}x < |q| < 2|x|$. That implies $r = \mathtt{fl}(x - q) = x - q$, so $x = q + r$ and $S + r = (M + q) + (x - q) = M + x$.

∎

If M is large enough, we only keep some leading bits of all the elements so that their sum can be computed accurately. This leads to Algorithm 3 by Rump [7].

Algorithm 3 requires $4n + \mathcal{O}(1)$ floating-point operations (FLOPs). In Algorithm 3, if all operations are performed in round to nearest even mode, then each element v_i is transformed exactly into a high order part q_i and a trailing part r_i. According to [4, p.5 Theorem 3.2], in rounding to nearest mode if $M = 2^k, k \in \mathbb{Z}$ and $M \ge \mathtt{fl}(\sum_{i=1}^{n} |v_i|/(1 - 2n \cdot \epsilon))$ then $T = \sum_{i=1}^{n} q_i$ i.e. the sum of all the high order parts can be computed accurately. Moreover $T + \sum_{i=1}^{n} r_i = \sum_{i=1}^{n} v_i$. Therefore Algorithm 3 is also called an error-free vector transformation.

In [2] Rump made an observation that M does not need to be the same for each iteration. Instead we can update the value of M after each iteration to absorb the value q_i so that

Algorithm 3 Error-free vector transformation: $[T, r] = \mathtt{ExtractVector}(M, v)$

Require: v is a vector of n floating-point numbers, $M = 2^k, k \in \mathbb{Z}$ and $M \ge \sum_i |v_i|/(1 - 2 * n * \mathbf{u})$. All operations are performed in rounding to nearest even mode.

```
T = 0
for i = 1 to n do
    q_i = fl(fl(M + v_i) − M)
    r_i = fl(v_i − q_i)
    T = fl(T + q_i)
end for
```

Ensure: T is the exact sum of high order parts q_i, r is the vector of low order parts r_i.

we do not have to explicitly sum the high order parts q_i. This leads to an improved algorithm called $\mathtt{ExtractVectorNew}$.

Algorithm 4 Error-free vector transformation: $[T, r] = \mathtt{ExtractVectorNew}(M, v)$

Require: v is a vector of n floating-point numbers, $M \ge 2 \cdot \mathtt{fl}(\sum_i |v_i|/(1 - (3n + 1)\mathbf{u}))$. All operations are performed in rounding to nearest even mode.

```
M_0 = M
for i = 1 to n do
    M_i = fl(M_{i−1} + v_i)
    q_i = fl(M_i − M_{i−1})
    r_i = fl(v_i − q_i)
end for
T = fl(M_n − M_0)
```

Ensure: T is the exact sum of high order parts q_i, r is the vector of low order parts r_i.

Algorithm 4 costs only $3n + \mathcal{O}(1)$ FLOPs, which is 25% faster than Algorithm 3. As suggested by Rump [2, Lemma 4.5], in order for T to be the exact sum of high order parts $\sum_{i=1}^{n} q_i$, the sufficient requirement is: $M \ge 2\sum_{1}^{n} |v_i|/(1 - (3n + 1) \cdot \epsilon)$. Algorithm 4 is also an error-free vector transformation in rounding to nearest even mode.

IV. Reproducible Sum

Using either of the above two error-free vector transformations, we can obtain the exact sum of high order parts of all elements. It means that the summation of these high order parts does not depend on the order of evaluation. Nevertheless, in order to obtain a reproducible sum we need to ensure that these high order parts are reproducible too.

For Algorithm 3, this can be ensured by computing a reproducible M. Rump suggested using $M = 2^{\lceil \log_2(\delta) \rceil}$ where $\delta = \mathtt{fl}((\sum_{i=1}^{n} |v_i|)/(1 - 2n\epsilon)$ since the smaller M is, the more accurate the final sum is. However $\sum_{i=1}^{n} |v_i|$ is itself a sum of floating-point numbers, which in general cannot be computed reproducibly.

A natural choice is to use $m = \max(|v_i|)$, which can always be evaluated reproducibly since floating-point maximum is associative. In addition, in order to meet the requirement $M \geq \mathtt{fl}((\sum_{i=1}^{n}|v_i|)/(1-2n\epsilon)$ we set $M = 2^{\lceil \log_2(\delta) \rceil}$ where $\delta = \mathtt{fl}(n * m/(1-2n\epsilon))$. Note that if we are only interested in obtaining the reproducible sum, then we can ignore the trailing parts as described by Algorithm 5 below.

Algorithm 5 Reproducible Sequential Sum

Require: v is a vector of n floating-point numbers. All operations are performed in rounding to nearest even mode.

```
1: m = max(|v_i|)
2: δ = fl(n * m/(1 − 2nϵ))
3: M = 2^⌈log2(δ)⌉
4: T = 0
5: for i = 1 to n in any order do
6:     q_i = fl(fl(M + v_i) − M)
7:     T = fl(T + q_i)
8: end for
```

Ensure: T is the sum of high order parts

Algorithm 5 is a simple modification of Algorithm 3 and costs $4n + \mathcal{O}(1)$ FLOPs, counting additions and comparisons.

Theorem 4 (Reproducibility of Algorithm 5)**.** *Let v be a vector of n floating-point numbers. The sum of all elements of v computed using Algorithm 5 is always reproducible as long as the same rounding mode is used.*

Proof: Since $m = \max(|v_i|)$ is always reproducible, so is M. This means the computation of q_i (line 6 of Algorithm 5) is reproducible. Moreover, according to [4, Lemma 3.2], the summation of q_i is exact for any order of evaluation, hence the computed result is reproducible, which completes the proof. ∎

Moreover, since the sum of high order parts q_i is exact it can be performed in any order i.e. using any reduction tree. For example sequential summation corresponds to a flat tree, and pair-wise summation [5] corresponds to a binary tree.

Algorithm 4 however cannot be used directly to compute a reproducible sum. Since we make no assumption about the order of evaluation, even if we use a reproducible value for M, the intermediate values M_i are not reproducible. Therefore the extracted high order parts q_i are not reproducible, so their sum is not necessarily reproducible.

For example consider a case where $M = 2^{53}$, $v_1 = -0.75, v_2 = 3.25$. The following table shows the execution of Algorithm 4 in two different evaluation orders.

$S = v_1 + v_2 + \ldots$	$S = v_2 + v_1 + \ldots$
$M_0 = 2^{53}$	$M_0 = 2^{53}$
$M_1 = 2^{53} - 1$	$M_1 = 2^{53} + 4 = M_{(2)}$
$q_1 = -1$	$q_{(2)} = 4 \neq q_2$
$M_2 = 2^{53} + 2$	$M_2 = 2^{53} + 4 = M_{(1)}$
$q_2 = 3$	$q_{(1)} = 0 \neq q_1$

In the above example, the evaluated high order parts of the first two elements in two evaluation orders are different. This is because the terms M_i used for splitting v_i in this example have different units in the last place which leads to different rounding. It means that in order for the high order parts to be computed reproducibly we need to ensure that all the intermediate value M_i have the same unit in the last place, i.e. have the same exponent.

Nevertheless, having the same exponent alone is still not sufficient to extract the high order parts reproducibly. Consider another example where we want to extract the higher order part of $x = 7$ (in rounding to nearest even mode) (i) by $M_1 = 2^{53}$: $M_1' = \mathtt{fl}(M_1 + x) = 2^{53} + 8 \rightarrow q_1 = M_1' - M_1 = 8$ and (ii) by $M_2 = 2^{53} + 2$: $M_2' = \mathtt{fl}(M_2 + x) = 2^{53} + 8 \rightarrow q_2 = M_2' - M_2 = 6 \neq q_1$.

Even though M_1 and M_2 have the same exponent, the evaluated high order parts of x are different due to the effect of the mid-point in rounding to nearest even mode. The following lemma provides a sufficient requirement to obtain a reproducible high order part.

Lemma 3. *Let $x, M1, M2$ be three floating-point numbers which satisfy: $|x| < M1, M2$ and $\mathtt{ulp}(M1) = \mathtt{ulp}(M2) = \mathtt{ulp}(\mathtt{fl}(M1 + x)) = \mathtt{ulp}(\mathtt{fl}(M2 + x)) = u$. If floating-point operations are all performed in one of the following IEEE 754 rounding modes: to nearest with ties away from 0, toward 0, toward $+\infty$, or toward $-\infty$; then:*

$$\mathtt{fl}(\mathtt{fl}(M1 + x) - M1) = \mathtt{fl}(\mathtt{fl}(M2 + x) - M2).$$

Proof: Since $|x| < M1$ and $|x| < M2$, according to Theorem 2 for rounding to nearest mode and Theorem 3 for directed rounding modes we have: $\mathtt{fl}(\mathtt{fl}(M1+x) - M1) = \mathtt{fl}(M1+x) - M1$ and $\mathtt{fl}(\mathtt{fl}(M2+x) - M2) = \mathtt{fl}(M2 + x) - M2$.

Let $\epsilon_1 = (M1 + x) - fl(M1 + x)$ and $\epsilon_2 = (M2 + x) - fl(M2 + x)$. Then $|\epsilon_1| < u$, $|\epsilon_2| < u$ and $\mathtt{fl}(M1 + x) - \mathtt{fl}(M2 + x) = (M1 - M2) + (\epsilon_2 - \epsilon_1)$.

Since $M1, M2, \mathtt{fl}(M1 + x), \mathtt{fl}(M2 + x) \in u\mathbb{Z}$, then $\epsilon_2 - \epsilon_1 \in u\mathbb{Z}$. We distinguish two cases of rounding mode: rounding to nearest and directed roundings.

1) Case of directed roundings. Since $|x| < M1, |x| < M2$, then $M1 + x$ and $M2 + x$ are both positive. Hence ϵ_2 and ϵ_1 must be of the same sign or zero. Therefore $|\epsilon_2 - \epsilon_1| \leq \max(|\epsilon_1|, |\epsilon_2|) < u$. $|\epsilon_2 - \epsilon_1| \in \mathbf{u}\mathbb{Z}$, hence $|\epsilon_2 - \epsilon_1| = 0$. This implies $\mathtt{fl}(M1 + x) - M1 = \mathtt{fl}(M2 + x) - M2$.
2) Case of rounding to nearest, ties away from zero. We have: $|\epsilon_1| \leq \frac{1}{2}\mathtt{ulp}(fl(M1 + x)) = \frac{1}{2}u$ and $|\epsilon_2| \leq$

$\frac{1}{2}\mathtt{ulp}(fl(M2+x)) = \frac{1}{2}u$. If either $|\epsilon_1| < \frac{1}{2}u$ or $|\epsilon_2| < \frac{1}{2}u$ then $|\epsilon_2 - \epsilon_1| < u$, which implies that $\epsilon_2 - \epsilon_1 = 0$. Otherwise if $|\epsilon_1| = |\epsilon_2| = \frac{1}{2}u$, since tie-breaking is away from zero and $M1 + x > 0, M2 + x > 0$ we deduce $\epsilon_1 = \epsilon_2 = -\frac{1}{2}u$, which concludes the proof.

■

In practice, *most* current processors do not implement the IEEE rounding mode to nearest with tie-breaking away from zero. For example, there is no function to set that rounding mode in the C language. Therefore, for the rest of the paper we only consider either rounding mode to nearest (tie-breaking to even) or directed roundings.

According to Lemma 3, if we choose an initial value of M so that all the intermediate values M_i in Algorithm 4 have the same unit in the last place then using an appropriate rounding mode will result in reproducible high order parts and therefore will produce a reproducible sum. Let δ be an upper bound of $\sum_{i=1}^{n} |q_i|$. Then it is easy to see that all the intermediate values of M_i will be bounded by: $M - \delta < M_i < M + \delta$. Therefore if $\mathtt{ulp}(M - \delta) = \mathtt{ulp}(M + \delta)$ then $\mathtt{ulp}(M_i) = \mathtt{ulp}(M)$ for all i. In this case $\frac{1}{2} \cdot \epsilon^{-1} \cdot \mathtt{ulp}(M) \leq M \pm \delta < \epsilon^{-1} \cdot \mathtt{ulp}(M)$, hence $4\delta < \epsilon^{-1} \cdot \mathtt{ulp}(M)$. A natural choice is to use $M = 3 \cdot 2^{\lceil \log_2 \delta \rceil}$. That leads to a new sequential algorithm to compute the reproducible sum of floating-point numbers, which is described in Algorithm 6.

Algorithm 6 Fast Reproducible Sequential Sum

Require: v is a vector of size n. All operations are performed in the same directed rounding mode.

$m = \mathtt{max}_i(|v_i|)$

$\delta = \mathtt{fl}(\mathtt{fl}(n * m)/(1 - 4 * (n+1) * \epsilon))$

$e = \lceil \log_2(\delta) \rceil$

$M = \mathtt{fl}(3 * 2^e)$

$M_0 = M$

for $i = 1$ to n in any sequential order **do**

$\quad M_i = \mathtt{fl}(M_{i-1} + v_i)$

end for

$T = \mathtt{fl}(M_n - M)$

Ensure: T is a reproducible sum of all elements of v.

Theorem 5. *Let T be the result computed by Algorithm 6. Then $T = M_n - M = \sum_{i=1}^{n}(\mathtt{fl}(M + v_i) - M)$, $T \in \mathtt{ulp}(M)\mathbb{Z}$ and $|T| < M$. This means T is reproducible.*

Proof: First, we prove $\mathtt{ulp}(M_i) = \mathtt{ulp}(M)$ for $0 \leq i \leq n$. It is true for $i = 0$. Suppose that $\mathtt{ulp}(M_i) = \mathtt{ulp}(M)$ for all $i = 0 : k, k < n$, we will prove that $\mathtt{ulp}(M_{k+1}) = \mathtt{ulp}(M)$. Let $\epsilon_i = (M_i + v_i) - M_{i+1}$. So $|\epsilon_i| < \mathtt{ulp}(M_{i+1})$ and $M_k - M = \sum_{i=1}^{k}(M_i - M_{i-1}) = \sum_{i=1}^{k}(v_i + \epsilon_i)$. Hence $|(M_k + v_{k+1}) - M| \leq \sum_{i=1}^{k+1}|v_i| + \sum_{i=1}^{k}|\epsilon_i| < (k+1) \cdot m + k \cdot \mathtt{ulp}(M) \leq n \cdot m + n \cdot \mathtt{ulp}(M) - \mathtt{ulp}(M)$.

We have $\delta > (1 - \mathbf{u})^2 \cdot (n \cdot m)/(1 - 4 \cdot (n+1) \cdot \epsilon) > n \cdot m/(1 - 4 \cdot (n+1) \cdot \epsilon)$. Moreover $2^e = 2^{\lceil \log_2 \delta \rceil} \geq \delta$. Therefore $n \cdot m/(1 - 4 \cdot n \cdot \epsilon) < 2^e$, so $n \cdot m < 2^e - 4 \cdot n \cdot \epsilon \cdot 2^e$.

In addition $M = 3 \cdot 2^e$ so $\mathtt{ulp}(M) = 4 \cdot \epsilon \cdot 2^e$. Hence $n \cdot m < 2^e - n \cdot \mathtt{ulp}(M)$, and $|(M_k + v_{k+1}) - M| < 2^e - \mathtt{ulp}(M)$. It is easy to see that $2^{e+1} + \mathtt{ulp}(M) \leq M_{k+1} \leq 2^{e+2} - \mathtt{ulp}(M)$, which implies $\mathtt{ulp}(M_{k+1}) = 4 \cdot \epsilon \cdot 2^e = \mathtt{ulp}(M)$.

By induction, we deduce $\mathtt{ulp}(M_i) = \mathtt{ulp}(M)$ for all $i = 0 : n$. According to Lemma 3 we have $M_{i+1} - M_i = \mathtt{fl}(M_i + v_{i+1}) - M_i = \mathtt{fl}(M + v_{i+1}) - M$. In addition, $\mathtt{ulp}(M_i) = \mathtt{ulp}(M)$ also implies $0 < M/2 < M_i < 2M$, according to Sterbenz's theorem $\mathtt{fl}(M_i - M) = M_i - M$. Which implies $T = \mathtt{fl}(M_n - M) = M_n - M_0 = \sum_{i=1}^{n}(M_i - M_{i-1}) = \sum_{i=1}^{n}(\mathtt{fl}(M + v_i) - M)$.

Since M is reproducible, T is reproducible too. Moreover, $\mathtt{ulp}(M_n) = \mathtt{ulp}(M)$ also implies $T = M_n - M \in \mathtt{ulp}(M)\mathbb{Z}$ and $|T| = |M_n - M| < M$. ■

Note that on one hand the exactness of the summation of high order parts allows the for-loop in Algorithm 6 to be performed in any order. On the other hand, since the value of M is updated after each iteration it limits the use of evaluation tree to a flat tree, i.e. sequential order, only.

Algorithm 6 costs $2n + \mathcal{O}(1)$ FLOPs, counting additions and comparisons. So it is two times faster than Algorithm 5.

Theorem 6. *Let T be the result computed by either Algorithm 5 or Algorithm 6. If $n \cdot \epsilon \ll 1$ ($n \cdot \epsilon < 0.1$ is good enough) then*

$$\left| \sum_{i=1}^{n} v_i - T \right| < \sum_{i=1}^{n} \mathtt{ulp}(M + v_i) < c \cdot n^2 \cdot \epsilon \cdot \max |v_i|. \quad (3)$$

where $c \approx 4$ for Algorithm 5, and $c \approx 8$ for Algorithm 6.

Proof: We have $T = \sum_{i=1}^{n}(\mathtt{fl}(M + v_i) - M)$. Therefore $|\sum_{i=1}^{n} v_i - T| = |\sum_{i=1}^{n}(v_i - (\mathtt{fl}(M + v_i) - M))| \leq \sum_{i=1}^{n}|(M + v_i) - \mathtt{fl}(M + v_i)| < \sum_{i=1}^{n}\mathtt{ulp}(M + v_i)$.

In Algorithm 5, $M = \mathtt{fl}(n * m/(1 - 2 * n * \mathbf{u}) < n \cdot m \cdot (1 + \mathbf{u})^2/(1 - 2 \cdot n \cdot \mathbf{u})$ and $|v_i| < M$. So $|\sum_{i=1}^{n} v_i - T| < \sum_{i=1}^{n}\mathtt{ulp}(M + M) = 2n \cdot \mathtt{ulp}(M) < 4n \cdot \epsilon \cdot M < \frac{4(1+2\epsilon)^2}{1-2n\epsilon} n^2 \cdot \epsilon \cdot \max|v_i| = c_1 \cdot n^2 \cdot \epsilon \cdot \max|v_i|$.

In Algorithm 6, $\mathtt{ulp}(M + v_i) = \mathtt{ulp}(M) = 4\epsilon \cdot 2^e$. It is easy to see that $\delta \leq 2^e < 2\delta$. In addition $\delta = \mathtt{fl}(n * m/(1 - 2 * n * \mathbf{u})) < (1 + \mathbf{u})^2 \cdot n \cdot m/(1 - 2 \cdot n \cdot \mathbf{u})$. Therefore $|\sum_{i=1}^{n} v_i - T| < \sum_{i=1}^{n}\mathtt{ulp}(M) < 8 \cdot n \cdot \epsilon \cdot \delta < \frac{8(1+\mathbf{u})^2}{(1-2n \cdot \mathbf{u})} n^2 \cdot \epsilon \cdot \max|v_i| = c_2 \cdot n^2 \cdot \epsilon \cdot \max|v_i|$.

Therefore, for both algorithms we get an absolute error bound on T: $E_{rep} = c \cdot n^2 \cdot \epsilon \cdot \max|v_i|$, with $c = c_1 \approx 4$ for Algorithm 5 and $c = c_2 \approx 8$ for Algorithm 6. ■

The error bound from Theorem 6 is tight since the error of each algorithm is the exact sum of the rounding errors of n floating-point additions $M + v_i$, each of order $\mathbf{u} \cdot M$, and the value of M is of order $n \cdot \max|v_i|$.

To have a comparison of the error bound provided by Theorem 6 with the error bound E_{con} for the conventional non-reproducible sum let's consider two extreme cases:

(i) In the best case, where all the elements of v are almost of the same magnitude and of the same sign, then $\sum_{i=1}^{n} |v_i| \approx n \cdot \max|v_i|$. Therefore $E_{rep} \approx cE_{con}$.

(ii) In the worst case where one element of v is much larger in magnitude than the others then $\sum_{i=1}^{n} |v_i| \approx \max|v_i|$. Therefore $E_{rep} \approx cnE_{con}$.

In general we have $cE_{con} < E_{rep} < cnE_{con}$. In practice, while E_{rep} is tight, the measured error of the conventional sum can be much smaller than E_{con}. Therefore, as will be seen in Section V, the measured errors of conventional sum are much smaller than the measured errors of Algorithms 5 and 6. Since we clipped the trailing parts and only summed the high order parts of input elements, in order to improve the accuracy of the reproducible summation, we need to repeat the phase of extracting high order parts on the trailing bits of the input elements.

K-fold reproducible summation

For Algorithm 5, if all the operations are performed in rounding to nearest even, then the low order parts of all elements can be computed accurately. So we can repeat the high order extraction phase k times until all the significant bits of all elements are taken into account, or until obtaining the expected accuracy, for example as if computed in K-fold precision or with a faithful rounding scheme [4].

For Algorithm 6, however, since we exclude the use of rounding to nearest even, normally a directed rounding mode must be used. According to Theorem 3, we cannot accurately split input elements into high order parts and trailing parts since Algorithm 2 is not an error-free transformation with directed rounding.

Lemma 4. *Let $T \in \mathbb{F}$, $r \in \mathbb{F}^n$ be the results computed by Algorithm 3 or Algorithm 4 with directed rounding. Then*

$$|(T + \sum_{i=1}^{n} r_i) - \sum_{i=1}^{n} v_i| < n \cdot 2\mathbf{u}^2 \cdot M. \tag{4}$$

Proof: For both algorithms $T = \sum_{i=1}^{n} q_i = \sum_{i=1}^{n} (\mathtt{fl}(M + v_i) - M)$. So $(T + \sum_{i=1}^{n} r_i) - \sum_{i=1}^{n} v_i = \sum_{i=1}^{n} (v_i - (q_i + r_i))$. Moreover, according to Theorem 3: $|v_i - (q_i + r_i)| < 2\mathbf{u}^2 \cdot M$, which concludes the proof. ∎

Therefore the error bound of Algorithm 6 cannot be lower than $2n \cdot \mathbf{u}^2 \cdot M$. Nevertheless, our goal is to compute a reproducible sum which is almost as accurate as the normal sum, so this lower bound is acceptable provided that $n\epsilon < 1$.

For the sake of unambiguity, let `ExtractVectorNew2` be an implementation of Algorithm 4 in which all operations are performed in directed rounding mode.

Another remark is that for each reproducible extraction phase, we need to obtain the maximum absolute value of all elements which would be costly especially for parallel computing since it requires another reduction operation. Instead, according to Theorem 3, the absolute value of the trailing part is always bounded by $\mathtt{ulp}(M + x)$, so we can use that value to avoid having to recompute $\max|v_i|$. Note that for Algorithm 5, M is a power of 2, so $\mathtt{ulp}(M + v_i) \leq \mathtt{ulp}(M) = 2 \cdot \epsilon \cdot M$ for all $i = 1 : n$. Meanwhile for Algorithm 6 we have $\mathtt{ulp}(M_i + v_i) = \mathtt{ulp}(M) = \frac{4}{3} \cdot \epsilon \cdot M$.

Algorithm 7 describes the k-fold reproducible sum based on Algorithm 5 and Algorithm 8 describes the k-fold reproducible sum which is based on Algorithm 6. Both algorithms consist of splitting input elements into chunks of adjacent bits which are of the same order of magnitude, and can be summed accurately.

Algorithm 7 Accurate Sequential K-fold Reproducible Sum

Require: v is a vector of size n. All operations are performed in the same rounding to nearest mode.

$m = \mathtt{max}_i(v_i)$
$\delta_1 = \mathtt{fl}(n * m/(1 - 2 * n * \epsilon))$
$M_1 = 2^{\lceil log_2(\delta_1) \rceil}$
for $f = 1$ to $k - 1$ **do**
  $[T_f, v] = \mathtt{ExtractVector}(M_f, v)$
  $\delta_{f+1} = \mathtt{fl}(n * (2 * \epsilon * M_f)/(1 - 2 * n * \epsilon))$
  $M_{f+1} = 2^{\lceil log_2(\delta_{f+1}) \rceil}$
end for
$T_k = 0$
for $i = 1$ to n in any order **do**
  $q_i = \mathtt{fl}(\mathtt{fl}(M_k + v_i) - M_k)$
  $T_k = \mathtt{fl}(T_k + q_i)$
end for
$T = 0$
for $i = 1$ to k **do**
  $T = \mathtt{fl}(T + T_i)$
end for

Ensure: T is a reproducible sum of all elements of v.

Algorithm 7 costs $4kn + \mathcal{O}(1)$ FLOPs. And Algorithm 8 costs only $(3k - 1)n + \mathcal{O}(1)$ FLOPs. Note that in both algorithms we can pre-compute all the $M_{i=1:k}$ to pipeline the k extracting phases in order to improve the performance.

Lemma 5. *Let T be the result computed by either Algorithm 7 or Algorithm 8. If $n \cdot \epsilon \ll 1$ ($n \cdot \epsilon < 0.1$ is good enough) then:*

(a) $M_i = c \cdot \theta^i \cdot n^i \cdot \mathbf{u}^{i-1} \cdot m$, where $\theta \in [2, 4]$, $c = 1$ for Algorithm 7, $c = \frac{3}{4}$ for Algorithm 8,

(b) $|T_i| < |M_i|$

(c) $|T - \sum_{i=1}^{k} T_i| < k \cdot \mathbf{u} \cdot |T|$.

Proof: $4n \ll \epsilon^{-1}$, so for both algorithms we have $\delta_1 \approx n \cdot m$. In addition, $\delta_i \leq 2^{\lceil \log_2(\delta_i) \rceil} < 2 \cdot \delta_i$ for $i = 1 : k$.

For Algorithm 7: $\delta_i \approx 2 \cdot \epsilon \cdot n \cdot M_{i-1}$ for $i = 2 : k$. Since $M_i = 2^{\lceil \log_2(\delta_i) \rceil}$, then $\delta_i \leq M_i < 2 \cdot \delta_i$. Therefore $M_i = \theta \cdot n \cdot \epsilon \cdot M_{i-1}$ for $i = 2 : k$, and $M_1 = \theta \cdot n \cdot m$ where $\theta \in [2, 4]$. By induction we deduce $M_i = \theta^i \cdot n^i \cdot \epsilon^{i-1} \cdot m$.

For Algorithm 8: $\delta_i \approx \frac{4}{3} \cdot \epsilon \cdot n \cdot M_{i-1}$ for $i = 2 : k$. Since $M_i = 3 * 2^{\lceil \log_2(\delta_i) \rceil}$, then $3 \cdot \delta_i \leq M_i < 6 \cdot \delta_i$. Moreover for

Algorithm 8 Fast Sequential K-fold Reproducible Sum

Require: v is a vector of size n. All operations are performed in the same directed rounding mode.

$m = \mathtt{max}_i(v_i)$
$\delta_1 = \mathtt{fl}(n * m/(1 - 4 * (n+1) * \epsilon))$
$M_1 = 3 * 2^{\lceil log_2(\delta_1) \rceil}$
for $f = 1$ to $k-1$ **do**
  $[T_f, v] = \mathtt{ExtractVectorNew2}(M_f, v)$
  $\delta_{f+1} = \mathtt{fl}(n * (4 * \epsilon * M_f/3)/(1 - 4 * (n+1) * \epsilon))$
  $M_{f+1} = 3 * 2^{\lceil log_2(\delta_{f+1}) \rceil}$
end for
$M = M_k$
for $i = 1$ to n in any sequential order **do**
  $M = \mathtt{fl}(M + v_i)$
end for
$T_k = \mathtt{fl}(M - M_k)$
$T = 0$
for $i = 1$ to k **do**
  $T = \mathtt{fl}(T + T_i)$
end for

Ensure: T is a reproducible sum of all elements of v.

directed rounding $\mathbf{u} = 2 \cdot \epsilon$, therefore $M_i = \theta \cdot n \cdot \mathbf{u} \cdot M_{i-1}$ for $i = 2 : k$, and $M_1 = \frac{3}{4}\theta \cdot n \cdot m$. By induction we deduce $M_i = \frac{3}{4}\theta^i \cdot n^i \cdot \mathbf{u}^{i-1} \cdot m$. That concludes item (a).

(b) is deduced from Theorem 5 and [4, Theorem 3.2].

To prove item (c), we analyze the loop to compute T. Let $T^1 = T_1, T^{i+1} = \mathtt{fl}(T^i + T_{i+1}), i = 1 : k-1$, then $T^k = T$. Let $e_i = T^{i+1} - (T^i + T_{i+1})$ then $|e_i| < \mathbf{u} \cdot |T^i + T_{i+1}|$ and $T - \sum_{i=1}^{k} T_i = \sum_{i=1}^{k-1} e_i$. If all $e_i = 0$ then $T = \sum_{i=1}^{k} T_i$, the proof is trivial. Otherwise, let $s < k$ be the index where $e_s \neq 0$ and $e_{i<s} = 0$. Therefore $T^s = \sum_{i=1}^{s} T_i$.

For Algorithm 8, according to Theorem 5: $T_i \in \mathtt{ulp}(M_i)\mathbb{Z}$, which also implies $T_i \in \mathtt{ulp}(M_{j|j \geq i})\mathbb{Z}$. It is easy to deduce that $T^i \in \mathtt{ulp}(M_{j|j \geq i})\mathbb{Z}$. Therefore $e_s = T^{s+1} - (T^s + T_{s+1}) \in \mathtt{ulp}(M_{s+1})\mathbb{Z}$. According to Lemma 2 if $|T^s + T_{s+1}| < \epsilon^{-1}\mathtt{ulp}(M_{s+1})$ then $T^{s+1} = \mathtt{fl}(T^s + T_{s+1})$. So $|T^s + T_{s+1}| \geq \epsilon^{-1}\mathtt{ulp}(M_{s+1}) > M_{s+1}$.

For Algorithm 7, according to [4, Theorem 3.2]: $T_i \in \epsilon \cdot M_i\mathbb{Z}$. Since $M_{i+1} \ll M_i$, then $T_i \in \epsilon \cdot M_{j|j \geq i}\mathbb{Z}$. It is easy to deduce that $T^i \in \epsilon \cdot M_{j|j \geq i}\mathbb{Z}$. Therefore $e_s = T^{s+1} - (T^s + T_{s+1}) \in \epsilon \cdot M_{s+1}\mathbb{Z}$. In addition, if $|T^s + T_{s+1}| < M_{s+1}$ then $T^{s+1} = \mathtt{fl}(T^s + T_{s+1})$. Therefore $|T^s + T_{s+1}| \geq M_{s+1}$.

Since $M_i = \theta \cdot n \cdot \mathbf{u} \cdot M_{i-1}$, it is easy to see that $\sum_{i=s+2}^{k} M_i < \frac{\theta \cdot n \cdot \mathbf{u}}{1 - \theta \cdot n \cdot \mathbf{u}} M_{s+1}$. In addition $|T_i| < M_i$, so $\sum_{i=s+2}^{k} |T_i| < \frac{\theta \cdot n \cdot \mathbf{u}}{1 - \theta \cdot n \cdot \mathbf{u}} |T^s + T_{s+1}| = \frac{\theta \cdot n \cdot \mathbf{u}}{1 - \theta \cdot n \cdot \mathbf{u}} |\sum_{i=1}^{s+1} T_i|$. Which implies $|\sum_{i=1}^{k} T_i| > (1 - \frac{\theta \cdot n \cdot \mathbf{u}}{1 - \theta \cdot n \cdot \mathbf{u}}) \cdot |\sum_{i=1}^{s+1} T_i|$.

We also have $T = \mathtt{fl}(T^{s+1} + T_{s+2} + \ldots + T_k)$, so $|T^{s+1} + \sum_{i=s+2}^{k} T_i - T| < (k-s-1) \cdot \mathbf{u} \cdot (|T^{s+1}| + \sum_{i=s+2}^{k} |T_i|) + \mathcal{O}(\mathbf{u}^2) < \frac{(k-s-1)}{1 - \theta \cdot n \cdot \mathbf{u}} \cdot \mathbf{u} \cdot |T^{s+1}| + \mathcal{O}(\mathbf{u}^2)$.

Therefore $|\sum_{i=1}^{k} T_i - T| = |T^s + T_{s+1} + \sum_{i=s+2}^{k} T_i - T| < |T^s + T_{s+1} - T^{s+1}| + |T^{s+1} + \sum_{i=s+2}^{k} T_i - T| < \mathbf{u} \cdot |T^{s+1}| + \frac{(k-s-1)}{1-\theta \cdot n \cdot \mathbf{u}} \cdot \mathbf{u} \cdot |T^{s+1}| + \mathcal{O}(\mathbf{u}^2) < \frac{(k-s)}{1-\theta \cdot n \cdot \mathbf{u}} \cdot \mathbf{u} \cdot |T^{s+1}| + \mathcal{O}(\mathbf{u}^2) < \frac{(k-s)}{1-\theta \cdot n \cdot \mathbf{u}} \cdot \mathbf{u} \cdot (1+\mathbf{u}) \cdot |T^s + T_{s+1}| + \mathcal{O}(\mathbf{u}^2)$. Moreover $|T^s + T_{s+1}| = |\sum_{i=1}^{s+1} T_i| < (1 - \frac{\theta \cdot n \cdot \mathbf{u}}{1 - \theta \cdot n \cdot \mathbf{u}})^{-1} \cdot |\sum_{i=1}^{k} T_i|$. Hence $|\sum_{i=1}^{k} T_i - T| < \frac{(k-s)(1+\mathbf{u})}{1 - 2\theta \cdot n \cdot \mathbf{u}} \cdot \mathbf{u} \cdot |\sum_{i=1}^{k} T_i| + \mathcal{O}(\mathbf{u}^2)$.

With the assumption $4n \ll \epsilon^{-1}$ and $s \geq 1$ we can deduce item (c) of the lemma, which completes the proof. ∎

Theorem 7. *Let T be the result computed by Algorithm 8. If $n\epsilon \ll 1$ ($n \cdot \epsilon < 0.1$ is good enough) then*

$$|T - \sum_{i=1}^{n} v_i| < (k + c_1 \cdot n \cdot \mathbf{u} + c_2 \cdot n^k \cdot \mathbf{u}^{k-1}) \cdot n \cdot \mathbf{u} \cdot \max |v_i|. \quad (5)$$

Proof: Let S_i be the exact sum of all elements of v after the i-th step of extraction. $S_0 = S$ is the exact sum of the original input vector. According to Lemma 4, we have: $|T_i + S_i - S_{i-1}| < 2n \cdot \mathbf{u}^2 \cdot M_i$, $1 \leq i < k$. According to Theorem 6 $|S_{i-1} - T_i| < n \cdot \mathtt{ulp}(M_i) < \frac{2}{3} n \cdot \mathbf{u} \cdot M_i$. Therefore $\sum_{i=1}^{k} T_i - S = \sum_{i=1}^{k-1}(T_i + S_i - S_{i-1}) + (T_k - S_{k-1})$, which implies $|\sum_{i=1}^{k} T_i - S| < \sum_{i=1}^{k-1} 2n \cdot \mathbf{u}^2 \cdot M_i + \frac{2}{3} n \cdot \mathbf{u} \cdot M_k$. According to Lemma 5: $M_i = \theta \cdot n \cdot \mathbf{u} \cdot M_{i-1} = \frac{3}{4} \cdot \theta^i \cdot n^i \cdot \mathbf{u}^{i-1} \cdot m$, where $\theta \in [2, 4]$. So $|\sum_{i=1}^{k} T_i - S| < \frac{2n \cdot \mathbf{u}^2}{1 - \theta \cdot n \cdot \mathbf{u}} \cdot M_1 + \frac{1}{2} \cdot \theta^k \cdot n^{k+1} \cdot \mathbf{u}^k \cdot m$.

We have $E_{rep.k} = |T - \sum_{i=1}^{n} v_i| < |T - \sum_{i=1}^{k} T_i| + |\sum_{i=1}^{k} T_i - S| < k \cdot \mathbf{u} \cdot |\sum_{i=1}^{k} T_i| + |\sum_{i=1}^{k} T_i - S| < k \cdot \mathbf{u} \cdot (|S| + |\sum_{i=1}^{k} T_i - S|) + |\sum_{i=1}^{k} T_i - S| < k \cdot \mathbf{u} \cdot |S| + (1 + k \cdot \mathbf{u}) \cdot |\sum_{i=1}^{k} T_i - S|$.

With $|S| \leq n \cdot \max |v_i|$, we can deduce (5). ∎

A more practical error bound can be obtained by using the estimated sum and intermediate values M_i.

$$E_{rep.k} < k \cdot \mathbf{u} \cdot |T| + \frac{2n}{1 - 4n \cdot \mathbf{u}} \cdot \mathbf{u}^2 \cdot M_1 + \frac{2}{3} n \cdot \mathbf{u} \cdot M_k. \quad (6)$$

In order to obtain the same accuracy as the conventional non-reproducible sum we want $E_{con} \approx E_{rep.k}$. This means $n \cdot \mathbf{u} \cdot \sum_{i=1}^{n} |v_i| \approx (k + c \cdot n^k \cdot \mathbf{u}^{k-1}) \cdot n \cdot \mathbf{u} \cdot \max |v_i|$. Hence $\sum_{i=1}^{n} |v_i| \approx (k + c \cdot n^k \cdot \mathbf{u}^{k-1}) \cdot \max |v_i|$.

Therefore if $n^k \cdot \epsilon^{k-1} \ll 1$ or $n \ll \epsilon^{-(k-1)/k}$ then $E_{rep.k} \lessapprox E_{con}$. For example, with $k = 2$ phases of extracting high orders, if the number of elements n is much smaller than $\epsilon^{-1/2}$ then the error bound of Algorithm 8 is of the same order as the conventional sum. The practical accuracy of proposed algorithms will be presented in Section V.

Algorithm 9 is the parallelized version of Algorithm 8. Since the summation of each corresponding chunk of bits of all elements is accurate, i.e. independent of the evaluation order, it is easy to prove that all the intermediate values T_i in Algorithm 9 are reproducible, so is the final sum T. T is always reproducible even if the input vector is distributed differently across different number of processors. It should be noted that Algorithm 9 uses only two reductions: one for computing and broadcasting the global maximum absolute value of input elements, and one for aggregating all the partial sums. In other words, the cost of Algorithm 9 in

Algorithm 9 Parallel K-fold Reproducible sum

Require: v is the local vector of length n_p of processor p. N is the length of the global vector. All operations are performed in the same directed rounding mode.

allocate a vector R of k floating-point numbers
$m = \mathtt{max}_i(|v_i|)$ ▷ local max
$m = \mathtt{AllReduce}(m, \mathtt{MAX})$ ▷ global max
$\delta_1 = \mathtt{fl}(N * m/(1 - 4 * (N + 1) * \epsilon))$
$M_1 = 3 * 2^{\lceil log_2(\delta) \rceil}$
for $f = 1$ to $k - 1$ **do**
  $[T_f, v] = \mathtt{ExtractVectorNew2}(M_f, v)$
  $\delta_f = \mathtt{fl}(N * (4 * \epsilon * M_f/3)/(1 - 4 * (N + 1) * \epsilon))$
  $M_{f+1} = 3 * 2^{\lceil log_2(\delta_{f+1}) \rceil}$
end for
$M = M_k$
for $i = 1$ to n_p in any sequential order **do**
  $M = \mathtt{fl}(M + v_i)$
end for
$T_k = \mathtt{fl}(M - M_k)$
$[T_1, \ldots, T_k] = \mathtt{Reduce}([T_1, \ldots, T_k], \mathtt{SUM})$
if my rank is 0 **then** ▷ Final summation
  $T = 0$
  for $i = 1$ to k **do**
    $T = \mathtt{fl}(T + T_i)$
  end for
end if

Ensure: T is a reproducible sum of all elements of V.

terms of the number of reductions is constant regardless of the number of extraction phases. All the other algorithms presented above can also be parallelized very similarly.

Overflow and underflow

Up to this point, we made the assumption of no overflow and underflow to carry out rounding error analysis of algorithms. We will now discuss the effect of overflow and underflow on the reproducibility of the proposed algorithms.

For Algorithm 5, if there is no overflow in the evaluation of M, i.e. $M < 2^{e_{MAX}}$, since $M = 2^k, k \in \mathbb{Z}$, then $M \leq 2^{e_{MAX}-1}$. Moreover $|v_i| < M$ for all $i = 1 : n$ and $\sum_{i=1}^n |q_i| < M$, it is easy to deduce that there is no overflow in evaluating $q_i = \mathtt{fl}(\mathtt{fl}(M + x_i) - M)$ as well as $T = \sum_{i=1}^n q_i$. According to [7], if denormalized floating-point arithmetic is used then Algorithm 3 is an error-free transformation even in the presence of underflow. It means that Algorihm 5 is also reproducible in the presence of underflow. Otherwise the sum of high order parts q_i is not guaranteed to be reproducible if flush-to-zero is used in case of underflow. We can however avoid underflow in summing q_i by ensuring that $M \geq \epsilon^{-1} \cdot \eta$ where η is the underflow threshold. Since according to Theorem 2 we have $q_i \in \frac{1}{2}\mathtt{ulp}(M)$, so $q_i \in \eta\mathbb{Z}$, which suffices to conclude that there will be no underflow in evaluating $T = \sum_{i=1}^n q_i$.

For Algorithm 6, if there is no overflow or underflow in the evaluation of M then from the proof of Theorem 5 we can see that $\mathtt{ulp}(M_i) = \mathtt{ulp}(M)$ for all $i = 1 : n$. Hence there is no overflow or underflow in evaluating M_i. It also means that there is no overflow in evaluating $T = \mathtt{fl}(M_n - M)$. Nevertheless, there can still be underflow in evaluating T, but it does not affect the reproducibility of T since both M_n and M are reproducible. Otherwise we can use the same technique for Algorithm 5 to avoid underflow.

If there is overflow in the evaluation of M then we can prematurely exit the algorithms. Note that an overflowed M does not mean that the true sum overflows since $M > n \cdot \max |v_i| > |\sum_{i=1}^n v_i|$. We can however avoid overflow once we know the value of n and m by scaling the input numbers so that $M < 2^{e_{MAX}}$. That will of course lead to overhead in the computing time.

In case of underflow, i.e. $M < \eta$, since $|\sum_{i=1}^n v_i| < M < \eta$ we can simply return 0 as the computed result of both algorithms.

The same techniques can be used to handle overflow and underflow for Algorithm 7, Algorithm 6, and Algorithm 8.

V. Experimental results

In this section, we present some experimental results in Matlab to check the reproducibility as well as the accuracy of the proposed algorithms in double precision, i.e. $\epsilon = 2^{-53}$. In order to set the rounding mode, we use the `setround` function provided in the INTLAB library [8]. In order to check reproducibility, the non-determinism of evaluation order is simulated as follows:

- each generated input vector is split into blocks of b summands, with $b \in \mathbb{N}$ a power of 2 and varying from 32 up to the size of vector,
- each block is summed sequentially,
- the final result is computed by evaluating either a binary or flat reduction tree whose leaves are the partial sums sorted in random order.

A test is only considered reproducible if all the computed results for varying reduction configurations described above are bit-wise identical. For all of the tests below, computed results from Algorithm 5, Algorithm 6 and Algorithm 8 are always reproducible. In contrast for the normal sum, computed results are always non-reproducible. Each "Normal sum" in the plot is taken to be the sum computed sequentially from v_1 through v_n using Matlab's `sum` function. The differences between the maximum and minimum results for normal sum for different reduction configurations are always bigger than the errors of the normal sum plotted in the graphs.

Figure 1 and Figure 2 show the accuracy of the proposed algorithms as well as of the conventional sum for vector sizes of 10^3 and 10^6. The x-axis of both figures represents $log_2(\mathtt{fl}(|\sum_i v_i|)/|\mathtt{fl}(\sum_i v_i)|)$, which corresponds to the condition number of the input vector. The y-axis represents the relative error in Figure 1 and the absolute error in

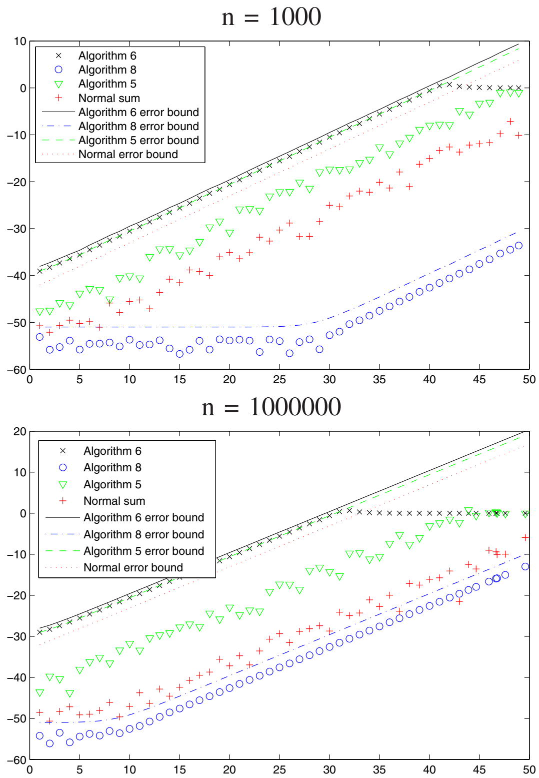

Figure 1. Accuracy of reproducible sum: log2(relative error)

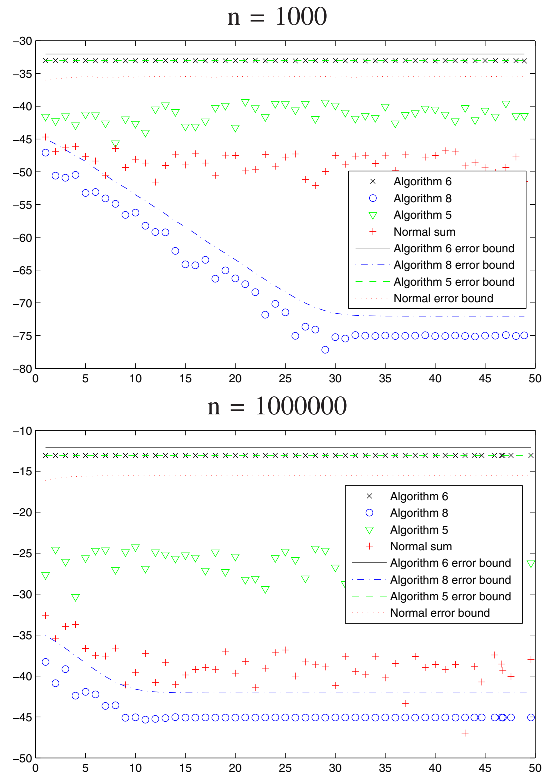

Figure 2. Accuracy of reproducible sum: log2(absolute error)

Figure 2, both in a logarithmic scale, so down is good. Together with the measured errors, both figures also show the error bounds of each algorithm. The error bound of the normal sum is provided by (2). The error bound of Algorithm 5 and Algorithm 6 is provided by (3), and of Algorithm 8 is provided by (6).

First, as can be seen from Figure 1 and Figure 2, numerical error bounds for Algorithm 6 and Algorithm 8 are tight, i.e. the factor between measured errors and the corresponding error bounds of each algorithm are close to 1. Nevertheless, the error bound of Algorithm 5 is less tight, even though Algorithm 5's error bound is almost of the same order as the error bound for Algorithm 6 (differing by a factor of less than 4). Since both Algorithm 5 and Algorithm 6 compute the exact sums of extracted high order parts, the only difference between the two algorithms that can explain the difference in measured errors is the rounding mode used by each algorithm, namely rounding to nearest even mode in Algorithm 5 and directed rounding in Algorithm 6. To see the effect of rounding mode on the accuracy of these two algorithms, let us consider an example where all the input elements are of the same sign. On one hand, using rounding to nearest mode ensures that the magnitude of errors are always smallest among all possible rounding modes. On the other hand, in directed rounding all the errors of the additions $\mathtt{fl}(M + v_i)$ are of the same sign, which means the sum of rounding errors is of the same magnitude as the sum of absolute errors. Meanwhile in rounding to nearest mode, the signs of rounding errors can be either positive or negative, hence their sum is smaller than the sum of absolute errors in magnitude. For example, if the rounding errors are distributed following the uniform distribution around 0, then the ultimate computing error is a lot smaller in magnitude than the sum of absolute errors as well as the error bound.

Second, we can also see that the normal error bounds are loose. Even though the normal error bounds are almost of the same order as those from reproducible sums Algorithm 5 and Algorithm 6, the measured errors of the normal sum are much smaller than those of Algorithm 5 and Algorithm 6. Since both these two reproducible algorithms ignore the trailing bits, as explained in the previous section that can be improved by increasing the number of extraction phases. As can be seen from these figures, results computed by Algorithm 8 with two passes of extracting high order parts are much more accurate than results computed by Algorithm 5 and Algorithm 6, and are most of the time more accurate than results computed by the conventional algorithm. It is worth noting that, in comparison with Algorithm 5, Algorithm 8 for $k = 2$ is only 25% slower but provides much more accurate results. It can also be seen that the gap in accuracy between Algorithm 8 ($k = 2$) and the conventional algorithm is much smaller for $n = 10^6$ than for $n = 10^3$. That means the accuracy of Algorithm 8 depends on the size n of the vector (with n^{k+1} in the error bound) more than the dependence of the accuracy of the normal sum on the size of vector (with n in the error bound).

Therefore for a given input vector of size n, one will need to choose the appropriate algorithm with the right number of extraction phases in order to obtain reproducible results of a required accuracy. The above experimental results show that for practical use, where the number of floating-point numbers is not too high, i.e. $n \lessapprox 10^6$ in double precision, then using Algorithm 8 with two extraction steps is appropriate to obtain reproducible results which are almost always more accurate than the conventional sum.

VI. Conclusions

In this paper, we presented a technique to obtain reproducibility of floating-point summation, which means obtaining bit-wise identical results for multiple runs on the same input data. The proposed techniques rely solely on floating-point arithmetic and do not require any kind of conversion from floating-point numbers to fixed-point numbers or to integers. It also does not require scaling input data. As long as all the processors use the same rounding mode, the reproducibility is always guaranteed regardless of the ill-conditioning of input data, independent of the number of processors, assignment of data to processors, data alignment, or SIMDization by the compiler (all of which can affect execution order), and is not affected by the Table Maker's Dilemma.

The proposed techniques also introduce some trade offs between efficiency and accuracy. Depending on the need of each specific application, one can choose to increase the accuracy by sacrificing the efficiency and vice versa. For example when accuracy is not of prime importance then we can use Algorithm 6 which uses only two times more FLOPs than the conventional sum. Nevertheless, when accuracy is needed then Algorithm 8 can be used with more extraction steps.

In parallel computing, Algorithm 9 uses two reduction operations and does $\mathcal{O}(n)$ arithmetic operations, so depending on the problem size and the computing environment the execution time may be dominated either by computing time or reduction time. For example in MapReduce architectures, the execution time may be totally dominated by the reduction, so Algorithm 9 is only two times slower than the conventional sum regardless of the number of extraction phases. If we want to obtain reproducible results with just one reduction, i.e. with almost the same execution time as a conventional sum, then to use the proposed algorithms we need to set M to the highest value as possible, i.e. the overflow threshold, and use enough extraction phases to cover all the possible magnitudes of floating-point numbers, i.e. down to the underflow threshold. In this case one could also consider using an exact summation algorithm, for example [9].

A similar approach can be applied to other problems to obtain reproducibility. For example a reproducible floating-point product $\Pi_{i=1}^{n} v_i$ can be computed by taking the sum of logarithms of $|v_i|$ and the one-bit xor reduction to get the correct sign bit of the product. We can also apply the "pre-round" technique to obtain reproducibility for parallel prefix-sum [10] (since all partial sums can be accurately computed), as well as for parallel prefix-product by means of logarithmic sum.

References

[1] O. Villa, D. Chavarria-Miranda, V. Gurumoorthi, A. Marquez, and S. Krishnamoorthy, "Effects of floating-point non-associativity on numerical computations on massively multi-threaded systems," in *Cray User Group meeting, CUG*, 2009.

[2] S. M. Rump, T. Ogita, and S. Oishi, "Fast high precision summation," *Nonlinear Theory and Its Applications, IEICE*, vol. 1, no. 1, pp. 2–24, 2010.

[3] M. Taufer, O. Padron, P. Saponaro, and S. Patel, "Improving numerical reproducibility and stability in large-scale numerical simulation on GPUs," in *IPDPS*, 2010, pp. 1–9.

[4] S. M. Rump, "Ultimately fast accurate summation," *SIAM Journal on Scientific Computing (SISC)*, vol. 31, no. 5, pp. 3466–3502, 2009.

[5] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1996.

[6] T. J. Dekker, "A floating-point technique for extending the available precision," *Numerische Methematik*, vol. 18, no. 3, pp. 224–242, 1971.

[7] S. M. Rump, T. Ogita, and S. Oishi, "Accurate Floating-Point Summation Part I: Faithful Rounding," *SIAM J. Sci. Comput.*, vol. 31, no. 1, pp. 189–224, 2008.

[8] S. Rump, "INTLAB - INTerval LABoratory," in *Developments in Reliable Computing*, T. Csendes, Ed. Dordrecht: Kluwer Academic Publishers, 1999, pp. 77–104, http://www.ti3.tu-harburg.de/rump/.

[9] S. Siegel and J. Wolff von Gudenberg, "A long accumulator like a carry-save adder," *Computing*, vol. 94, no. 2-4, pp. 203–213, Mar. 2012. [Online]. Available: http://dx.doi.org/10.1007/s00607-011-0164-x

[10] G. E. Blelloch, "Prefix sums and their applications," School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-90-190, Nov. 1990.