Accurate floating point summation*

James Demmel† Yozo Hida‡

May 8, 2002

Abstract

We present and analyze several simple algorithms for accurately summing n floating point numbers $S = \sum_{i=1}^{n} s_i$, independent of how much cancellation occurs in the sum. Let f be the number of significant bits in the s_i. We assume a register is available with $F > f$ significant bits. Then assuming that (1) $n \leq \lfloor 2^{F-f}/(1-2^{-f}) \rfloor + 1$, (2) rounding is to nearest, (3) no overflow occurs, and (4) all underflow is gradual, then simply summing the s_i in decreasing order of magnitude yields S rounded to within just over 1.5 units in its last place. If $S = 0$, then it is computed exactly. If we increase n slightly to $\lfloor 2^{F-f}/(1-2^{-f}) \rfloor + 3$ then all accuracy can be lost. This result extends work of Priest and others who considered double precision only ($F \geq 2f$). We apply this result to the floating point formats in the (proposed revision of the) IEEE floating point standard. For example, a dot product of IEEE single precision vectors $\sum_{i=1}^{n} x_i \cdot y_i$ computed using double precision and sorting is guaranteed correct to nearly 1.5 ulps as long as $n \leq 33$. If double extended is used n can be as large as 65537. We also show how sorting may be mostly avoided while retaining accuracy.

*Computer Science Division Technical Report UCB//CSD-02-1180, University of California, Berkeley, 94720. This research was supported in part by LLNL Memorandum Agreement No. B504962 under the Department of Energy Contract No. W-7405-ENG-48 and DOE Grant No. DE-FG03-94ER25219, the National Science Foundation under Grant No. ASC-9813362, NSF Cooperative Agreement No. ACI-9619020, NSF Infrastructure Grant No. EIA-9802069, the National Science Foundation Graduate Research Fellowship, and by a gift from Intel. The information presented here does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

†Computer Science Division and Mathematics Department, University of California, Berkeley, CA 94720 (demmel@cs.berkeley.edu).

‡Computer Science Division, University of California, Berkeley, CA 94720 (yozo@cs.berkeley.edu).

Contents

1 Introduction

We present and analyze several simple algorithms for accurately summing n floating point numbers. A good history of these algorithms may be found in chapter 4 in Higham's book [6, 5] and in Priest's dissertation [18, 17].

Suppose we are given n floating point numbers $s_1, \ldots, s_n$, which we may assume without loss of generality are finite and nonzero. Our goal is to compute $S = \sum_{i=1}^{n} s_i$ accurately, with (nearly) all correct bits no matter how extreme the cancellation is, i.e. no matter how much smaller $|S|$ is than $\sum_{i=1}^{n} |s_i|$. We can of course do this by carrying enough extra precision, but the practical question we address is how little extra precision we need to do so efficiently.

Here is our notation. We write $s_i = \pm 2^{e_i} m_i$, where $e_i \geq e_{\rm MIN}$ is the integer exponent and the significand m_i satisfies $0 \leq m_i < 2$. We say that s_i is *normalized* if $1 \leq m_i < 2$ or $s_i = 0$ (which is represented by $m_i = 0$ and $e_i = e_{\rm MIN}$), *unnormalized* if $0 < m_i < 1$, and *subnormal* if $0 < m_i < 1$ and $e_i = e_{\rm MIN}$. We write $e_i = {\rm EXP}(s_i)$. Let f be the number of significant bits (including any hidden bit [1]) in the m_i. One *unit in the last place (ulp)* in the f-bit floating point number x is ${\rm ulp}(x) \equiv 2^{{\rm EXP}(x)-f+1}$.

If $\hat{x} \neq 0$ is an f-bit floating point approximation of the real number x, we will measure the relative error in $\hat{x}$ in two ways, by $|\hat{x} - x|/|\hat{x}|$ or by $|\hat{x} - x|/{\rm ulp}(\hat{x})$ (the relative error measured in ulps). Note that an error of 2^{f-1} ulps or more means that x and $\hat{x}$ need not even have the same sign, and that just rounding x to f bits can cause an error of 0.5 ulps. Section 8.9 further discusses these two relative error measures.

Suppose we have an extra precise floating point register with $F > f$ significant bits. The following is our first algorithm:

Algorithm 1. *Accurate floating point summation by sorting the input.*

1. *Sort the* s_i *so that* ${\rm EXP}(s_1) \geq {\rm EXP}(s_2) \geq \cdots \geq {\rm EXP}(s_n)$
   ... the input data may be normalized, unnormalized, or subnormal.
2. $SUM \leftarrow 0$. *... SUM has F significant bits.*
3. **for** $i \leftarrow 1$ **to** n
4. $\quad SUM \leftarrow SUM + s_i$
5. **end**
6. $sum \leftarrow {\rm round}(SUM)$ *... round SUM back to f bits.*

Note that the sorting by exponents may be accomplished by sorting the s_i by decreasing magnitude: $|s_1| \geq |s_2| \geq \cdots \geq |s_n|$. This would make the algorithm run in time $O(n \log n)$. Alternatively, one could do a bucket or radix sort on the (much shorter) exponent fields of the s_i, reducing the running time to $O(n)$. The next theorem describes our results.

Theorem 1. *Suppose that*

1. $f \geq 2$, *i.e. we have at least two bits of precision in the data,*
2. $F \geq f + 1$, *i.e. we have at least one extra bit of precision in the extra precise floating point accumulator,*
3. *the exponent range for the* F*-bit format is at least as wide as for the* f*-bit format,*

4. *rounding is to nearest (ties may be broken arbitrarily),*

5. *underflow, if it occurs, is gradual, and*

6. *no overflow occurs.*

Let $\bar{n}$ *be defined by*

$$\bar{n} = 1 + \left\lfloor \frac{2^{F-f}}{1-2^{-f}} \right\rfloor = 1 + 2^{F-f} + 2^{F-2f} + \cdots + 2^{F \bmod f}$$

and r *be defined by*

$$r = f - (F \bmod f) = (\lfloor F/f \rfloor + 1)f - F$$

so that r *is the unique integer in the range* $1 \le r \le f$ *such that* $r + F \equiv 0 \pmod{f}$.

Then exactly one of the following four statements describes the maximum relative error in the value of sum computed by Algorithm 1:

1. *If* $n \le \bar{n}$, *then the error is at most* $\frac{1}{1-2^{1-f}} + \frac{1}{2}$ *ulps. For typical values of* f *this is just over* 1.5 *ulps.*

2. *If* $n = \bar{n} + 1$, $F \ge 2f$, *and* s_2 *is normalized, then the error is at most* $\max\left\{2.5,\ \frac{1.5}{1-2^{1-f}} + \frac{1}{2}\right\}$ *ulps. (By* s_2 *we mean after sorting in the first step of Algorithm 1.)*

3. *If* $n = \bar{n}+1$ *and either* $F < 2f$ *or* s_2 *is unnormalized, then the error is at most* $\max\left\{3.5,\ 2^r + \frac{1}{2}\right\}$ *ulps.*

4. *If* $n \ge \bar{n} + 2$, *then the relative error can be larger or equal to 1 (no relative accuracy). For example, the computed sum may be zero when the true sum is not.*

All the above error bounds are nearly attainable. If $n \le \bar{n}+1$ *(cases 1, 2, and 3 above), the computed sum is positive, zero, or negative exactly when* S *is positive, zero, or negative, respectively.*

If $F < 2f$, *then sorting the* s_i*'s more finely by magnitude instead of just the exponents (so that* $|s_1| \ge |s_2| \ge \cdots \ge |s_n|$*) does not change the above results.*

Assumptions 1 through 5 of Theorem 1 are satisfied by the formats of the (proposed revision of the) IEEE binary floating point standard [1, 2, 7]. Assumption 6 depends of course on the data $s_1, \ldots, s_n$ as well.

This theorem completely characterizes the maximum attainable error from Algorithm 1. It is noteworthy that the worst case error deteriorates from nearly perfect (approximately 1.5 ulps) to complete loss of accuracy (computing $sum = 0$ when the exact sum is nonzero) just by increasing n by 2, from $\bar{n}$ to $\bar{n} + 2$.

The cost of Algorithm 1 is dominated by sorting. We also present Algorithms 2, 3 and 4 that reduce the cost of sorting at the expense of lowering the largest value of n for which they guarantee a small error. Depending on n, f, F, and the relative costs of memory accesses, arithmetic and extracting a bit field from a floating point number, any of Algorithms 1, 2, 3 or 4 may be fastest. This is discussed further in section 5.

The main contribution of this paper is our complete analysis of Algorithm 1, and more generally our algorithms for exploiting any amount of extra precision $F > f$. In contrast, prior work (see section 6) addresses just the case of at least double precision $F \ge 2f$.

A related accurate summation problem is *distillation* [17, 3], where the goal is to compute the *exact* floating point sum S represented as the sum of as few f-bit quantities as possible: $S = \sum_{i=1}^{n} s_i = \sum_{i=1}^{m} t_i$ where m is as small as possible. The idea is that each f-bit quantity t_i represents a subset of the bits in S. Early versions of this algorithm (see Algorithm 5 in section 4) cost $O(n^2)$, and Priest gave an $O(n \log n)$ algorithm. In section 4 we give an $O(n)$ algorithm for distillation of at most $n \leq 2^{F-f}$ quantities that runs in time $O(n)$.

The rest of the paper is organized as follows. Sections 2 and 3 present and analyze Algorithms 2, 3 and 4. Their analyses are simple consequences of Theorem 1, whose complicated proof we postpone until later. Section 4 discusses distillation. Section 5 gives examples comparing Algorithms 1, 2, 3 and 4 for various values of f and F in the proposed IEEE floating point standard. It also considers the computation of dot products and distillation. Section 6 discusses related work. Section 7 presents the results of numerical testing of Algorithm 1. Section 8 proves Theorem 1. Section 9 draws conclusions and states open problems.

2 Accurate summation by partially sorting the input

The cost of Algorithm 1 is dominated by sorting the input s_i by their exponent fields. These fields are typically short (e.g. 8 bits for IEEE floating point single precision numbers) so one could consider radix or bucket sorting.

Our next algorithm reduces the cost of sorting the exponents further to just radix or bucket sorting on the leading exponent bits. But as a consequence, fewer numbers can be added accurately.

We let $\mathrm{EXP}_b(s)$ denote the exponent of s rounded down to the nearest multiple of 2^b; if the exponent is stored as an unsigned integer (as in IEEE floating point arithmetic) then $\mathrm{EXP}_b(s)$ is obtained by zeroing out (or just ignoring) the trailing b bits of $\mathrm{EXP}(s)$.

Algorithm 2. *Accurate floating point summation by partially sorting the input.*

1. *Sort the* s_i *so that* $\mathrm{EXP}_b(s_1) \geq \mathrm{EXP}_b(s_2) \geq \cdots \geq \mathrm{EXP}_b(s_n)$
   ... the input data may be normalized, unnormalized, or subnormal
2. $SUM \leftarrow 0$ *... SUM has F significant bits*
3. **for** $i \leftarrow 1$ **to** n
4. $\quad SUM \leftarrow SUM + s_i$
5. **end**
6. $sum \leftarrow \mathrm{round}(SUM)$ *... round SUM back to f bits*

Theorem 2. *Let*

$$f' = f + 2^b - 1,$$

and

$$\bar{n}' = 1 + \left\lfloor \frac{2^{F-f'}}{1 - 2^{-f'}} \right\rfloor.$$

Then the following statement holds:

1. *If* $n \leq \bar{n}'$, *then the error in the sum computed by Algorithm 2 is bounded by* $\frac{1}{1-2^{1-f'}} + \frac{1}{2}$ *ulps.*

2. *If* $n = \bar{n}'+1$*, then the error in the sum computed by Algorithm 2 is bounded by* $\max\left\{3.5,\ 2^r + \frac{1}{2}\right\}$ *ulps, where* $r = (\lfloor F/f' \rfloor + 1)f' - F$ *is the unique integer in the range* $1 \le r \le f'$ *such that* $r + F \equiv 0 \pmod{f'}$.

Note that these error bounds are the same as statements 1 and 3 of Theorem 1 if we replace f *by* f'.

In other words, ignoring the trailing b exponent bits when sorting lowers the maximum value of n for which a small (about 1.5 ulp) error is guaranteed by a factor of about 2^{2^b-1}. Only the bounds in Theorem 1 are valid; we do not know whether they are attainable in the case of Theorem 2.

Proof. The proof is a simple consequence of Theorem 1. As we will see in the proof of Theorem 1 (Section 8), the only part of the proof of Theorem 1 that assumes that s_2 is normalized is in Section 8.2, where the bound in the case $n = \bar{n} + 1$, $F \ge 2f$ is tightened. We do not need this part of Theorem 1 for Theorem 2.

Suppose the s_i have their exponents sorted as required by Algorithm 2. Then consider an f-bit floating point number s_i as an $(f+2^b-1)$-bit *unnormalized* floating point number, by adjusting the exponent down to the nearest multiple of 2^b and shifting the fraction at most $2^b - 1$ bit positions. These unnormalized numbers have sorted exponents as required by Algorithm 1. □

3 Accurate summation without sorting the input

We give third and fourth accurate summation algorithms that completely eliminate the need to sort the inputs s_i. Instead, we will have a fixed array of F-bit accumulators $A_0, \ldots, A_{N-1}$ each of which will accumulate an exact sum of a subset of the s_i. Then these A_j will be added using Algorithm 1. Depending on f and F, the number N of these accumulators may be much less than n, greatly lowering the cost of sorting. Each s_i will only be read from memory once. This algorithm will require accessing the exponent field of each s_i to decide to which accumulator A_j to add it.

We assume for simplicity that the exponent field of the s_i is an unsigned integer, as in IEEE floating point arithmetic [1]. In other words the true $\mathrm{EXP}(s)$ is gotten by taking the stored exponent bits of s, interpreting them as a nonnegative integer, and subtracting a constant. For example, this constant is 127 for IEEE single precision and 1023 for IEEE double precision.

Suppose that there are E bits in the stored exponent and f bits in the fraction of each s_i.

Algorithm 3. *Accurate summation without sorting the input (version 1)*

1. *Choose parameter e (number of leading exponent bits of each* s_i *to extract) according to Theorem 3 below.*
2. *Initialize* $A_k \leftarrow 0$ *for* $k = 0, 1, \ldots, N-1$*, where* $N = 2^e$.
3. **for** $i \leftarrow 1$ **to** n
4. $\quad j \leftarrow$ *leading e bits of the stored exponent of* s_i
5. $\quad A_j \leftarrow A_j + s_i$
6. **end**
7. *Add the* A_j *in order of decreasing exponent fields, yielding an* F*-bit* SUM
8. $sum \leftarrow \mathrm{round}(SUM)$ *... round* SUM *back to* f *bits*

By letting j be the leading exponent bits of s_i, Algorithm 3 divides the exponent range of the s_i into groups of powers of two. Obviously one could divide the exponent range into groups that are not powers of 2, but this would require an integer division rather than a simple bit extraction to compute j, so we do not consider this more expensive possibility (see section 6 on related work).

Theorem 3. *Algorithm 3 computes sum with an error of about* 1.5 *ulps provided*

$$n \leq 2^{F-f-2^{E-e}-e+1}.$$

This upper bound on n is maximized by the choice $e = E$ or $e = E-1$, yielding

$$n \leq 2^{F-f-E}.$$

Proof. The condition that each A_j be computed exactly means adding up to n numbers with exponents differing by as much as $2^{E-e} - 1$. This means we can think of the s_i as fixed point numbers with $f + 2^{E-e} - 1$ bits. Adding n of them exactly would take an additional $\lceil \log_2 n \rceil$ bits, leading to the inequality

$$\lceil \log_2 n \rceil + f + 2^{E-e} - 1 \leq F \quad . \tag{1}$$

Now we apply Theorem 1 to the array $A_0, ..., A_{N-1}$. Each accumulator has at most $f' = \lceil \log_2 n \rceil + f + 2^{E-e} - 1$ nonzero bits, and thus we can sort and accurately add as many as $N = 2^e$ F-bit words if

$$N \leq \left\lfloor \frac{2^\delta}{1 - 2^{f'}} \right\rfloor \qquad \text{where } \delta = F - f'. \tag{2}$$

This inequality is satisfied if $e \leq \delta$. Inequality (1) is equivalent to the weaker inequality $0 \leq \delta$, so the single inequality that we must choose e in the range $0 \leq e \leq E$ to satisfy is

$$\lceil \log_2 n \rceil \leq F - f - 2^{E-e} - e + 1 \tag{3}$$

yielding the first claim of the theorem.

Next we ask what is the largest number n of s_i that we can add accurately using this method for any choice of e? In other words, we want to maximize the right hand side of (3) over $0 \leq e \leq E$. By inspection we see that $e = E$ or $e = E - 1$ both maximize the expression, yielding

$$\lceil \log_2 n \rceil \leq F - f - E \tag{4}$$

which is the second claim of the theorem. □

The choice $e = E - 1$ is preferable to $e = E$ because there are half as many A_j to sort and add at the end. On the other hand, it may be faster to extract the whole exponent ($e = E$) than just the leading $e = E - 1$ bits.

This analysis of Algorithm 3 is pessimistic because it is unlikely that all $N = 2^e$ accumulators A_j will be nonzero, for this would require the s_i to be distributed all the way from the underflow to the overflow threshold. Nonetheless, we now consider a variation on Algorithm 3 that further increases the maximum number of s_i whose sum we can guarantee to compute accurately.

Algorithm 4 will be very similar to Algorithm 3, except it will split each F-bit accumulator into at most $\lceil F/f \rceil$ f-bit quantities, and then adds these quantities in sorted order.

Algorithm 4. *Accurate summation without sorting the input (version 2)*

1. *Choose parameter e (number of leading exponent bits of each s_i to extract)*
   according to Theorem 4 below.
2. *Initialize $A_k \leftarrow 0$ for $k = 0, 1, \ldots, N-1$, where $N = 2^e$.*
3. **for** $i \leftarrow 1$ **to** n
4. $\quad j \leftarrow$ *leading e bits of the stored exponent of* s_i
5. $\quad A_j \leftarrow A_j + s_i$
6. **end**
7. *... Break all the A_j into f-bit quantities t_k with the same sum*
8. $k = 0$
9. **for all** $A_j \neq 0$
10. $\quad$ **while** $A_j \neq 0$
11. $\quad\quad k = k + 1$
12. $\quad\quad t_k = \text{round}(A_j)$ *... round A_j back to f-bits*
13. $\quad\quad A_j = A_j - t_k$
14. $\quad$ **end**
15. **end**
16. *Add t_1 through t_k using Algorithm 1, 2, 3 or 4(recursively)*
17. *... to terminate, Algorithm 4 must eventually call Algorithm 1, 2, or 3 in the line above*

Theorem 4. *Algorithm 4 computes sum with an error of about* 1.5 *ulps provided*

$$n \leq 2^{F-f-2^{E-e}+1}$$

where

$$e \leq \min(E, F - f - \lceil \log_2 F/f \rceil) \ .$$

The upper bound on n is maximized with the choice of $e = \min(E, F - f - \lceil \log_2 F/f \rceil)$.

Proof. The analysis is analogous to that of Theorem 3. We still get inequality (1) but replace inequality (2) by

$$k \leq \left\lceil \frac{F}{f} \right\rceil N = \left\lceil \frac{F}{f} \right\rceil 2^e \leq 2^{F-f} < \bar{n} = 1 + \left\lfloor \frac{2^{F-f}}{1 - 2^{-f}} \right\rfloor \tag{5}$$

or

$$e \leq F - f - \log_2 \lceil F/f \rceil \ .$$

Since $\lceil \log_2 F/f \rceil \geq \log_2 \lceil F/f \rceil$ and $e \leq E$ as well, we get the statement of the theorem. It is easy to see that maximizing e maximizes the bound on n. □

Given a value of n, we can choose e to minimize the work in Algorithm 3 by finding the smallest e in the range $0 \leq e \leq E$ satisfying the first inequality of Theorem 3, since $N = 2^e$ is the number of accumulators A_j to sort and add at the end. Similarly, we can choose e to minimize the work in Algorithm 4. We consider these possibilities in the section comparing Algorithms 1 through 4.

4 Distillation in $O(n)$ time

We consider the problem of distillation [3, 18], or computing the exact sum $S = \sum_{i=1}^{n} s_i$ of n f-bit quantities represented as the sum $S = \sum_{i=1}^{m} d_i$ of as few f-bit quantities as possible. The goal is for the d_i to contain pairwise disjoint subsets of all the bits of S, but in practice we settle for $|d_i| \geq 2^f |d_{i-1}|$, so that d_m is within 1 ulp of S. (Note that the d_i's do not have to have the same sign.)

Distillation is accomplished most simply by copying the $s()$ array to the $d()$ array and repeatedly sweeping through the $d()$ array and replacing each pair (d_i, d_{i+1}) by $(d_i + d_{i+1} - \hat{d}, \hat{d})$, where $\hat{d}$ is the sum $d_i + d_{i+1}$ rounded to f bits.

Algorithm 5. *Accurate floating point summation via distillation in $O(n^2)$ time [3, 18]. (The exact sum is $\sum_{i=1}^{n} d_i$, but leading d_i that are equal to zero should be discarded, so that m is the number of nonzero d_i.)*

```
1.  repeat
2.      for i ← 1 to n − 1
3.        if |d_i| > |d_{i+1}| then
4.          d̂ ← d_i + d_{i+1}
5.          d_i ← (d_i − d̂) + d_{i+1}
6.          d_{i+1} ← d̂
7.        else
8.          d̂ ← d_i + d_{i+1}
9.          d_i ← (d_{i+1} − d̂) + d_i
10.         d_{i+1} ← d̂
11.       end if
12.     end for
13. until |d_i| ≤ 2^{−F} |d_{i+1}|  ∀ i
```

In the worst case this simple procedure does $\Theta(n^2)$ work, although an $O(n \log n)$ version is given by Priest [18]. To understand the complexity, one can think of Algorithm 5 in the worst case doing bubble sort on the d_i, whereas Priest's algorithm does merge sort.

Both Algorithm 5 and Priest's version assume only an f-bit floating point format is available. Furthermore, there is no limit on the value of n for which Algorithm 5 or Priest's version works correctly. If however we (1) assume an F-bit format is available, and (2) limit n depending on F, f and E as in Algorithm 4, then we can perform distillation in $O(n)$ time, without sorting. The idea is simply that lines 1 through 15 of Algorithm 4 replace the input array $s_1, ..., s_n$ by the (generally shorter) array $t_1, ..., t_k$ with the same sum, and $k \leq \lceil F/f \rceil 2^e \leq \lceil F/f \rceil 2^E$. Note that $\lceil F/f \rceil 2^E$ depends only on the floating point formats, not n. Thus, we could apply either Algorithm 5 or Priest's version to $t_1, ..., t_k$ to get the final distilled sum in $O(n)$ time. In practice, depending on n, F, f, and E, it might pay to call Algorithm 4 recursively to further compress the array $t_1, ..., t_k$, or in fact the algorithm might not work at all, in the case when $k > n$ (in which case no progress is made). We discuss these possibilities further in the next section.

5 Comparing Algorithms 1, 2, 3 and 4

We consider various values of f and F arising from computations in the (proposed revision of the) IEEE floating point standard [1, 2, 7]. Recall that the numbers of significant bits (including hidden ones) in single (S), double (D), extended (E), and quad (Q) formats are 24, 53, 64 and 113, respectively. The number of bits in their exponent fields are 8, 11, 15 and 15, respectively.

5.1 Accurate Summation

We begin by comparing the largest values of n for which we can guarantee an accurate sum (good to about 1.5 ulps) when the s_i are stored with f bits, and SUM and any A_j are stored in $F > f$ bits. These values of n are labeled $n_{1.5\text{ ulps}}$ in the tables below. In the case of Algorithm 1, Theorem 1 tells us $n_{1.5\text{ ulps}} = \bar{n}$. When $n_{1.5\text{ ulps}}$ is very large, we approximate it by the nearest powers of 2 and 10 (the power of 2 shown is actually a lower bound on $n_{1.5\text{ ulps}}$).

Since Algorithms 2, 3 and 4 have parameters b and e to choose, that reduce their costs while also reducing the maximum value of n for which they guarantee accuracy, we also show $n_{1.5\text{ ulps}}$ for various choices of b and e. For Algorithm 2, we actually show $E - b$, where E is the number of exponent bits in s_i, because the dominant cost of Algorithm 2 is radix or bucket sorting on $E - b$ bits. Theorem 2 tell us that the value of $n_{1.5\text{ ulps}}$ for Algorithm 2 is at least 2^{F-f-2^b+1}. Only values of b for which $n_{1.5\text{ ulps}}$ exceeds 2 are shown, since otherwise the algorithm is not useful.

Theorem 3 tells us that the value of $n_{1.5\text{ ulps}}$ for Algorithm 3 is $2^{F-f-2^{E-e}-e+1}$. Only values of e for which $n_{1.5\text{ ulps}}$ exceeds 2 are shown, since otherwise the algorithm is not useful. $N = 2^e$ is the number of F-bit accumulators A_j that need to be sorted and added, and is also shown. Only values of e for which N is less than $n_{1.5\text{ ulps}}$ are shown, since otherwise the algorithm is not useful (because it requires more work than Algorithm 1).

Theorem 4 tells us that the value of $n_{1.5\text{ ulps}}$ for Algorithm 4 is $2^{F-f-2^{E-e}+1}$ where $e \leq \min(E, F - f - \lceil \log_2 F/f \rceil)$. Both e and the maximum number $M = \lceil F/f \rceil 2^e \geq k$ of f-bit quantities $t_1, ..., t_k$ that are generated (to be added by Algorithms 1, 2, 3, or 4) are shown. Only values of e for which $n_{1.5\text{ ulps}}$ exceeds M are shown, since otherwise the algorithm is not useful (because it requires more work than Algorithm 1). Similarly, only values of e for which the number of items to sort is smaller than Algorithm 3 is shown (for the same value of n), since otherwise Algorithm 4 requires more work than Algorithm 3.

Note that for all combinations except s_i of type D and SUM of type E, it is possible to sum tens of millions of numbers accurately while sorting only tens, hundreds or perhaps thousands of numbers. Only with the aforementioned D/E combination are we limited to summing just $2^{11} + 1 = 2049$ numbers accurately. In particular, Algorithm 3 does not apply to the D/E combination (because n is limited to $n \leq 2^{F-f-E} = 2^{64-53-11} = 1$), and neither does Algorithm 4 (because it reduces the problem of summing $2^{F-f-2^{E-e}+1} = 2^{12-2^{11-e}}$ numbers to the problem of summing $\lceil F/f \rceil 2^e = 2^{e+1}$ numbers, which is greater than $2^{12-2^{11-e}}$ for $0 \leq e \leq \min(E, F - f - \lceil \log_2 F/f \rceil) = 10$).

Note that Table 1 also suggests how to choose the cheapest algorithm to use for any particular n, by indicating which one does the least sorting. (Which algorithm is really cheapest for a particular value of n will of course depend on programming and architectural details.) For example, consider the case where s_i of type S and SUM is of type D:

Table 1: Limit of lengths of accurate sums for Algorithms 1, 2, 3, 4.

		Algorithm 1	Algorithm 2		Algorithm 3			Algorithm 4		
s_i	SUM	$n_{1.5\text{ ulps}}$	$E-b$	$n_{1.5\text{ ulps}}$	e	N	$n_{1.5\text{ ulps}}$	e	M	$n_{1.5\text{ ulps}}$
S	D	$\approx 2^{29} \approx 10^9$	7	$\approx 2^{28} \approx 10^8$	7	128	$2^{21} \approx 10^6$	8	768	$2^{29} \approx 10^9$
			6	$\approx 2^{26} \approx 10^8$	6	64	$2^{20} \approx 10^6$	7	384	$2^{28} \approx 10^8$
			5	$\approx 2^{22} \approx 10^6$	5	32	$2^{17} = 131072$	6	192	$2^{26} \approx 10^8$
			4	$2^{14}+1 = 16385$	4	16	$2^{10} = 1024$	5	96	$2^{22} \approx 10^6$
S	E	$\approx 2^{40} \approx 10^{12}$	7	$\approx 2^{39} \approx 10^{12}$	7	128	$2^{32} \approx 10^9$	8	768	$2^{40} \approx 10^{12}$
			6	$\approx 2^{37} \approx 10^{11}$	6	64	$2^{31} \approx 10^9$	7	384	$2^{39} \approx 10^{12}$
			5	$\approx 2^{33} \approx 10^{10}$	5	32	$2^{28} \approx 10^8$	6	192	$2^{37} \approx 10^{11}$
			4	$\approx 2^{25} \approx 10^7$	4	16	$2^{21} \approx 10^6$	5	96	$2^{33} \approx 10^{10}$
			3	$2^9+1 = 513$	3	8	$2^6 = 64$			
S	Q	$\approx 2^{89} \approx 10^{27}$	7	$\approx 2^{88} \approx 10^{26}$	7	128	$2^{81} \approx 10^{24}$	8	1280	$2^{89} \approx 10^{27}$
			6	$\approx 2^{86} \approx 10^{26}$	6	64	$2^{80} \approx 10^{24}$	7	640	$2^{88} \approx 10^{26}$
			5	$\approx 2^{82} \approx 10^{24}$	5	32	$2^{77} \approx 10^{23}$	6	320	$2^{86} \approx 10^{26}$
			4	$\approx 2^{74} \approx 10^{22}$	4	16	$2^{70} \approx 10^{21}$	5	160	$2^{82} \approx 10^{24}$
			3	$\approx 2^{58} \approx 10^{17}$	3	8	$2^{55} \approx 10^{16}$			
			2	$\approx 2^{26} \approx 10^8$	2	4	$2^{24} \approx 10^7$			
D	E	$2^{11}+1 = 2049$	10	$2^{10}+1 = 1025$						
			9	$2^8+1 = 257$						
			8	$2^4+1 = 17$						
D	Q	$\approx 2^{60} \approx 10^{18}$	10	$\approx 2^{59} \approx 10^{18}$	10	1024	$2^{49} \approx 10^{15}$	11	6144	$2^{60} \approx 10^{18}$
			9	$\approx 2^{57} \approx 10^{17}$	9	512	$2^{48} \approx 10^{14}$	10	3072	$2^{59} \approx 10^{18}$
			8	$\approx 2^{53} \approx 10^{16}$	8	256	$2^{45} \approx 10^{13}$	9	1536	$2^{57} \approx 10^{17}$
			7	$\approx 2^{45} \approx 10^{13}$	7	128	$2^{38} \approx 10^{11}$	8	768	$2^{53} \approx 10^{16}$
			6	$\approx 2^{29} \approx 10^9$	6	64	$2^{23} \approx 10^7$			
E	Q	$\approx 2^{49} \approx 10^{15}$	14	$\approx 2^{48} \approx 10^{14}$	14	16384	$2^{34} \approx 10^{10}$	15	65536	$2^{49} \approx 10^{15}$
			13	$\approx 2^{46} \approx 10^{14}$	13	8192	$2^{33} \approx 10^{10}$	14	32768	$2^{48} \approx 10^{14}$
			12	$\approx 2^{42} \approx 10^{12}$	12	4096	$2^{30} \approx 10^9$	13	16384	$2^{46} \approx 10^{14}$
			11	$\approx 2^{34} \approx 10^{10}$	11	2048	$2^{23} \approx 10^7$	12	8192	$2^{42} \approx 10^{12}$
			10	$\approx 2^{18} \approx 10^5$				11	4096	$2^{34} \approx 10^{10}$

1. For $n \le 15$, Algorithm 1 or perhaps Algorithm 2 with $b = 4$ may be fastest (requiring sorting at most $n \le 15$ numbers).

2. For $16 \le n \le 1024$, Algorithm 3 with $e = 4$ may be fastest (requiring sorting at most 16 numbers).

3. For $1025 = 2^{10}+1 \le n \le 2^{22}$, Algorithm 4 with $e = 5$ may be fastest (generating at most 96 f-bit numbers which can then be added using Algorithm 3 with $e = 4$, requiring sorting of at most 16 numbers).

4. For $2^{22}+1 \le n \le 2^{26}$, Algorithm 4 with $e = 6$ may be fastest (generating at most 192 f-bit numbers which can then be added using Algorithm 3 with $e = 4$, requiring sorting of at most 16 numbers).

5. For $2^{26}+1 \le n \le 2^{28}$, Algorithm 4 with $e = 7$ may be fastest (generating at most 384 f-bit

numbers which can then be added using Algorithm 3 with $e = 4$, requiring sorting of at most 16 numbers).

6. For $2^{28} + 1 \le n \le 2^{29}$, Algorithm 4 with $e = 8$ may be fastest (generating at most 768 f-bit numbers which can then be added using Algorithm 3 with $e = 4$, requiring sorting of at most 16 numbers).

7. For $2^{29} + 1 \le n \le \bar{n} = 2^{29} + 2^5 + 1$, only Algorithm 1 is guaranteed accurate (requiring sorting all n numbers).

In contrast, when the type of s_i is D and the type of SUM is E, the best we can do is using Algorithm 1 or Algorithm 2 and sorting all $n \le 2049$ inputs.

We note that the Intel Itanium architecture [8] has 126 assignable floating point registers of type E, which would make Algorithm 3 for s_i of type S quite efficient for very large n. In contrast, the earlier Pentium architecture [9] with only 8 such registers would not support Algorithm 3 efficiently for $n \ge 64$.

5.2 Accurate Dot Products

Next, we consider the computation of inner products $\sum_{i=1}^{n} x_i \cdot y_i$ using Algorithms 1, 2, 3 or 4. We assume x_i and y_i are f-bit numbers, and that we have one or more F-bit accumulators available. We can perform this dot product accurately in one of two ways. The first way is to assume $F > 2f$, compute $s_i = x_i \cdot y_i$ exactly in F bits, and then sum the $2f$-bit numbers s_i using Algorithm 1, 2, 3 or 4. (Note that s_i may also have an exponent field one bit wider than that of x_i or y_i, so E is one larger.) The values of $n_{1.5\text{ ulps}}$ are shown for this method in Table 2. As before, we omit versions of algorithms that are manifestly inferior to other versions, or that simply do not work (as with D/E in Table 1).

Table 2: Limit of lengths of accurate dot products for Algorithms 1, 2, 3, 4.

		Algorithm 1	Algorithm 2		Algorithm 3			Algorithm 4		
x_i, y_i	SUM	$n_{1.5\text{ ulps}}$	$E-b$	$n_{1.5\text{ ulps}}$	e	N	$n_{1.5\text{ ulps}}$	e	M	$n_{1.5\text{ ulps}}$
S	D	$2^5+1=33$	8	$2^4+1=17$						
			7	$2^2+1=5$						
S	E	$2^{16}+1=65537$	8	$2^{15}+1=32769$				9	1024	$2^{16}=65536$
			7	$2^{13}+1=8193$				8	512	$2^{15}=32768$
			6	$2^9+1=513$				7	256	$2^{13}=8192$
S	Q	$\approx 2^{65} \approx 10^{19}$	8	$\approx 2^{64} \approx 10^{19}$	8	256	$\approx 2^{56} \approx 10^{17}$	9	1536	$2^{65} \approx 10^{19}$
			7	$\approx 2^{62} \approx 10^{18}$	7	128	$\approx 2^{55} \approx 10^{16}$	8	768	$2^{64} \approx 10^{19}$
			6	$\approx 2^{58} \approx 10^{17}$	6	64	$\approx 2^{52} \approx 10^{15}$	7	384	$2^{62} \approx 10^{18}$
			5	$\approx 2^{50} \approx 10^{15}$	5	32	$\approx 2^{45} \approx 10^{13}$	6	192	$2^{58} \approx 10^{17}$
			4	$\approx 2^{34} \approx 10^{10}$	4	16	$\approx 2^{30} \approx 10^{9}$			
			3	$2^2+1=5$						
D	Q	$2^7+1=129$	8	$2^6+1=65$						
			7	$2^4+1=17$						

Alternatively, we can convert the dot product into a sum $\sum_{i=1}^{2n} s_i$ of twice as many f-bit numbers by using well-known techniques for breaking the exact $2f$-bit product $x_i \cdot y_i = h_i + t_i$ into the sum of

two f-bit numbers (heads and tails) containing the leading and trailing bits of the product, [17, 18]. The most efficient was to get h_i and t_i is to use the fused-multiply-add instruction available on the Itanium [8]; the two instructions $h_i = x_i \cdot y_i$, $t_i = x_i \cdot y_i - h_i$ with the latter implemented using fused-multiply-add do the trick. This lets us compute accurate dot products for n up to half the values of $n_{1.5\text{ ulps}}$ shown in Table 1. This is the best way to deal with long dot products when the type of x_i and y_i are S and the type of SUM is D or E, or when x_i and y_i are in higher precision.

5.3 Distillation

We may example the data for Algorithm 4 in Table 1 to see how well distillation works. For example, consider the case where s_i of type S and SUM is of type D:

1. For $n \leq 96$, apply Priest's distillation algorithm.
2. For $96 < n \leq 2^{22}$, apply the first 15 lines of Algorithm 4 with $e = 5$ to reduce the problem to distilling at most 96 numbers. Then, apply Priest's distillation algorithm to distill these 96 numbers.
3. For $2^{22} < n \leq 2^{26}$, apply the first 15 lines of Algorithm 4 with $e = 6$ to reduce the problem to distilling at most 192 numbers. Then apply the first 15 lines of Algorithm 4 with $e = 5$ to reduce the problem to distilling at most 96 numbers. Finally, apply Priest's distillation algorithm to distill these 96 numbers.
4. For $2^{26} < n \leq 2^{28}$, apply the first 15 lines of Algorithm 4 with $e = 7$ to reduce the problem to distilling at most 384 numbers. Then apply the first 15 lines of Algorithm 4 with $e = 5$ to reduce the problem to distilling at most 96 numbers. Finally, apply Priest's distillation algorithm to distill these 96 numbers.
5. For $2^{28} < n \leq 2^{29}$, apply the first 15 lines of Algorithm 4 with $e = 8$ to reduce the problem to distilling at most 768 numbers. Then apply the first 15 lines of Algorithm 4 with $e = 5$ to reduce the problem to distilling at most 96 numbers. Finally, apply Priest's distillation algorithm to distill these 96 numbers.

In contrast, for the D/E case, only the original distillation algorithms apply.

6 Related Work

There is a long history of these sorts of algorithms discussed in [5, 6, 17, 18]. Important contributors include Bohlender [3], Dekker [4], Kahan [10], Knuth [11], Leuprecht/Oberaigner [12], Linnainmaa [13], Malcolm [14], Møller [15], Pichat [16], Priest [17, 18], Ross [19] and Wolfe [20].

Priest considers the two most similar algorithms to Algorithm 1: double precision summation [18, p. 62] and doubly compensated summation [18, p. 64], both of which correspond to Algorithm 1 when $F \geq 2f$. Double precision summation assumes two formats are available, and doubly compensated summation uses only one precision, but does 10 add/subtract operations in the inner loop instead of a single addition, in order to simulate double precision.

The main contribution of our analysis of Algorithm 1 is to analyze the general case $F > f$, for example to understand what can be done with the formats in the (proposed revision of the) IEEE

floating point standard [1, 2, 7], when F may not satisfy $F \geq 2f$. In particular, our contribution is the unimprovable bound on the number n of numbers that can be accurately summed, as a function of F and f.

We also slightly improve Priest's result [18, p. 62] by increasing the maximum value of n for which summation is guaranteed accurate over 8-fold from 2^{F-f-3} to $\bar{n} > 2^{F-f}$. This was not a significant restriction on the utility of Priest's result, since 2^{F-f-3} was already quite large when for practical values of F and f (e.g. $F - f - 3 = 53 - 24 - 3 = 26$ for IEEE single and double precisions). Our error bound and Priest's are essentially the same, about 1 ulp before the final rounding of SUM to get sum.

Malcolm [14] presents an algorithm analogous to our Algorithm 3, where he breaks up each f-bit number s_i into q words each with f/q nonzero bits, sums the resulting $q \cdot n$ words into f-bit accumulators exactly, and then sums the accumulators in (roughly) decreasing order. Malcolm uses an integer division by a possibly-non-power-of-2 to extract and scale the exponent; in contrast we restrict ourselves to the much cheaper operation of extracting a bit field from the exponent. As above, our contribution is to exploit the availability of any extra precision $F > f$ to lower the cost.

Bohlender [3] and Priest [18] both discuss distillation algorithms, of complexity $O(n^2)$ and $O(n \log n)$, respectively, for any values of n. For n bounded by at most 2^{F-f}, our distillation algorithm based on Algorithm 4 runs in time $O(n)$.

7 Numerical Testing

We confirmed the bound on n in Theorem 1 via extensive numerical testing where we generated "random" data $s_1, \ldots, s_n$ with massive cancellation, and confirmed the correctness of the computed sum. Our testing also showed that random tests are very unlikely to reveal the limits in the cases $n = \bar{n} + 1$ and $n \geq \bar{n} + 2$, unless they are constructed with the revelation of these limits in mind. In other words, adding numbers in decreasing magnitude order is likely to give accurate answers for n much larger than the $\bar{n}$ limit of Theorem 1.

Here is how we generated "random" examples. We did all computation in IEEE single precision, i.e. with $F = 24$. We generated $f < 24$ bit data by taking single precision numbers and zeroing out the trailing $24 - f$ bits.

We used the simple distillation Algorithm 5 from section 4 to compute the *exact* sum of a set of single precision numbers. This simple procedure is adequate for our testing purposes.

In this section, we assume $F < 2f$, so that $\bar{n} = 1 + 2^{F-f}$. Our initial sets of test cases were determined by 5 parameters n_a, n_s, e, f, and f_s as follows:

Algorithm 6. *Generating random numbers whose sum is tiny.*

1. *Generate n_a random numbers with f-bit fractions (stored in 24-bit numbers), with random exponents spanning the range from $-e/2$ to $e/2$. (If $n_a = 1$, then e must be zero.)*

2. *Compute the exact sum of these n_a numbers by distillation.*

3. *Choose n_s numbers to cancel this exact sum as much as possible. Each number is chosen as follows:*

3.1. *Take the correct sum of the numbers chosen so far as computed by distillation, truncate it to f_s bits, fill in the remaining $f - f_s$ bits randomly, and negate the result. This number cancels the leading f_s bits of the sum so far.*

3.2. *Adjoin the number just selected to the list so far, and update the correct sum by distillation again.*

Thus the total number of numbers generated is $n = n_a + n_s$, unless the loop in step 3 terminates prematurely because the overall sums cancels to zero exactly, in which case $n < n_a + n_s$. The total cancellation is about $C \approx \min(e + f, n_s \cdot f_s)$ bits, depending on whether cancellation to zero occurs. Statistics on the actual cancellation C are shown in the tables below. When C exceeds 24, as it does in almost all cases shown, the usual error analysis of straightforward summation without sorting says that we can expect no relative accuracy.

The answer is computed two ways: using Algorithm 1, and using the straightforward algorithm where the numbers are added in the order generated, without sorting. Error statistics (max and median) are given for both algorithms in the tables below. Error for Algorithm 1 is denoted "Err" and error for the straightforward algorithm is denoted "SErr".

Error is measured as follows. When the true sum is zero, the error is measured as the number of nonzero bits that failed to cancel in the computed sum, which is the number of bits in the range from the trailing fraction bit of any s_i to the leading bit of the computed sum. For example if the smallest trailing fraction bit is 2^{-120} and the leading bit of the computed sum is 2^{-25}, then Err is $-25 - (-120) + 1 = 96$. Thus Theorem 1 says that when $n \le 2^{F-f} + 1$, the maximum Err should be $\log_2 0 = -\infty$.

When the true sum is nonzero, the error is measured as base-2 logarithm of the error in units of ulps in the computed sum, where the leading word of the distilled sum is taken as the exact sum. In other words, the error is roughly the number of incorrect bits. If the computed sum and the exact sum agree exactly, then Err $= \log_2 0 = -\infty$. If the computed sum is zero, we say one ulp is 2^{-149}, the smallest positive subnormal number. Thus Theorem 1 says that when $n \le 2^{F-f} + 1$, the maximum Err should be a little over 0 (1 ulp) at most (we do not do the final rounding to f bits).

The initial parameters sets chosen for testing were as follows, all of which satisfy the condition $n \le \bar{n}$. One million random tests were generated for each parameter set.

Description of Case								True sum = 0					True sum ≠ 0				
Case	n_a	n_s	n	e	f	$F-f$	f_s	% Exs	Med C	Max Err	Max SErr	Med SErr	% Exs	Med C	Max Err	Max SErr	Med SErr
1	1	2	3	0	23	1	18	7	23	$-\infty$	$-\infty$	$-\infty$	93	26	$-\infty$	$-\infty$	$-\infty$
2	1	2	3	0	23	1	22	88	23	$-\infty$	$-\infty$	$-\infty$	12	26	$-\infty$	$-\infty$	$-\infty$
3	3	2	5	17	22	2	18	8	40	$-\infty$	16	12	92	39	$-\infty$	118	22
4	3	2	5	23	22	2	22	83	45	$-\infty$	22	18	17	45	$-\infty$	114	22
5	5	4	9	50	21	3	18	17	72	$-\infty$	53	44	83	78	$-\infty$	21	21
6	6	3	9	50	21	3	21	22	71	$-\infty$	49	43	88	68	$-\infty$	21	21

Here are some comments on these tests. The test cases are grouped by $F - f$, with 2 cases each for $F - f = 1$, 2 and 3. The errors for the true sum equal to zero and nonzero are shown separately (the "% Exs" column shows the percentage of the 10^6 examples in each category). Since $n \le 2^{F-f} + 1$, we expect a small Err, at most a little more than 0, in all cases. In fact all 24 bits of the computed sum from Algorithm 1 were correct in all cases, leading to Err $= \log_2 0 = -\infty$. This was true even though the number of bits C that need to cancel correctly range up to a median of 78, far exceeding single precision.

The error SErr from the straightforward algorithm is also shown. In Cases 1 and 2 (with $n_a = 1$) Algorithm 6 turns out to generate the summands in sorted order, so the Algorithm 1 and the straightforward algorithm behave identically. But in Cases 2 through 6 the error SErr from the straightforward algorithm is much larger than Err as expected. When the true sum is nonzero and SErr $= f$, as it frequently does, this means all bits in the computed sum are wrong. The cases when Max SErr equal 114 or 118 occur when the straightforward sum cancels exactly to zero (so one ulp is 2^{-149}) but the the exact sum is nonzero.

Now we consider cases where n far exceeds the limit $2^{F-f} + 1$ of Theorem 1. Thus we expect larger errors. In the first set below, the random numbers were generated just as before.

Description of Case								True sum = 0					True sum ≠ 0				
								%	Med	Max	Max	Med	%	Med	Max	Max	Med
Case	n_a	n_s	n	e	f	$F-f$	f_s	Exs	C	Err	SErr	SErr	Exs	C	Err	SErr	SErr
7	50	12	62	200	23	1	18	5	224	207	225	198	95	227	23	23	23
8	50	9	59	200	23	1	23	19	223	199	202	196	81	221	30	23	23
9	50	12	62	200	22	2	18	13	223	$-\infty$	224	195	87	227	$-\infty$	22	22
10	50	9	59	200	22	2	22	4	221	$-\infty$	200	194	96	213	$-\infty$	22	22
11	50	12	62	200	21	3	18	28	222	$-\infty$	224	195	72	227	$-\infty$	21	21
12	50	9	59	200	21	3	21	.3	221	$-\infty$	199	193	99.7	205	-2	21	21

In all cases, well over 200 bits of cancellation (Med C) of the 256-bit exponent range were required to get the correct answer. In cases 7 and 8, with $f = 23$, the Max Err is large, with nearly all bits wrong. But in cases 9 through 12, with $f \leq 22$, the error is very small, almost always $-\infty$ (i.e. no difference from the exact result) and at most -2 (i.e. 2^{-2} ulps). This shows that random testing is unlikely to reveal the limits on n in Theorem 1. SErr shows that the straightforward algorithm was nearly always wrong in all its bits.

Finally, we present some results where we modify Algorithm 6 to generate test cases slightly differently to try to increase the error: we make sure that that n_s numbers chosen to cancel the sum of the previously chosen numbers have have magnitude at most $2^{f-F-1} \cdot \max_k |s_k|$. In other words, the numbers are chosen to mimic the example used in the proof of attainability of the error bound in the case $n = \bar{n} + 1$ of Theorem 1, with numbers chosen to slowly cancel away the largest s_k, and small enough to cause rounding. Only 10^4 random examples were run in each case.

Description of Case								True sum = 0					True sum ≠ 0				
								%	Med	Max	Max	Med	%	Med	Max	Max	Med
Case	n_a	n_s	n	e	f	$F-f$	f_s	Exs	C	Err	SErr	SErr	Exs	C	Err	SErr	SErr
13	50	9	59	200	23	1	23	.7	224	190	199	194	99.3	144	23	23	23
14	50	9	59	200	22	2	22	.07	222	$-\infty$	196	194	99.93	48(221)	24	222	22
15	50	9	59	200	21	3	21	.01	221	$-\infty$	195	195	99.99	2(219)	21	221	-2

Now we see the expected failures in Algorithm 1 (large Max Err's) when $f = 22$ and 21, when the true sum is nonzero. The entries "48(221)" and "2(219)" indicate that the median cancellation C is indeed only 48 bits (when $f = 22$) or 2 bits (when $f = 21$), but the maximum is 221 bits (when $f = 22$) or 219 bits (when $f = 21$), i.e. much larger.

8 Proof of Theorem 1

We begin by establishing more basic facts and notation.

Suppose $x = \pm 2^e m$ is a normalized nonzero floating point number, so that $1 \leq m < 2$. Then $e \leq \log_2 |x| < e + 1$ and hence $e = \lfloor \log_2 |x| \rfloor$. For any normalized, unnormalized, or subnormal x we have $|x| < 2^{e+1}$.

We assume without loss of generality that the s_i have already been sorted as described in Algorithm 1. We introduce the following notation so we can talk about intermediate results in the algorithm:

1. $\widehat{SUM}_0 \leftarrow 0$
2. **for** $i \leftarrow 1$ **to** n
3. $\quad\quad \widehat{SUM}_i \leftarrow \widehat{SUM}_{i-1} + s_i$
4. **end**
5. $\widehat{sum} \leftarrow \text{round}(\widehat{SUM}_n)$

We let quantities with hats like $\widehat{SUM}_i$ denote the computed quantity corresponding to the exact value SUM_i.

We let $E_k = \text{EXP}(\widehat{SUM}_k)$ for all k. We let $e_S = \text{EXP}(S)$, where S is the exact sum represented as a floating point number, with as many significant bits as necessary. Let ϵ_i be the roundoff error committed in the F-bit floating point addition $\widehat{SUM}_i = \text{fl}(\widehat{SUM}_{i-1} + s_i)$. In other words $\widehat{SUM}_i = \widehat{SUM}_{i-1} + s_i + \epsilon_i$ exactly.

Let

$$\theta = 1 + \frac{2^{F-f}}{1-2^{-f}} = 1 + 2^{F-f} + 2^{F-2f} + 2^{F-3f} + \cdots,$$

so that

$$\bar{n} = \lfloor \theta \rfloor = 1 + 2^{F-f} + 2^{F-2f} + \cdots + 2^{F-\lfloor F/f \rfloor f}.$$

Note that if $F < 2f$, then $\bar{n} = 1 + 2^{F-f}$. Also note that

$$\theta - \bar{n} = 2^{-r} + 2^{-r-f} + 2^{-r-2f} + \cdots = \frac{2^{-r}}{1-2^{-f}},$$

where $r = (\lfloor F/f \rfloor + 1)f - F$, the unique integer in the range $1 \le r \le f$ such that $r + F \equiv 0 \pmod{f}$. Hence

$$\bar{n} = 1 + \frac{2^{F-f} - 2^{-r}}{1-2^{-f}}. \tag{6}$$

We will sometimes use the bounds

$$\bar{n} = \frac{2^{F-f} + 1 - 2^{-r} - 2^{-f}}{1-2^{-f}} < \frac{2^{F-f}+1}{1-2^{-f}} \tag{7}$$

and

$$\bar{n} - 2 = \frac{2^{F-f} - 1 - 2^{-r} + 2^{-f}}{1-2^{-f}} \le \frac{2^{F-f}-1}{1-2^{-f}}. \tag{8}$$

Recall that 1 ulp of an f-bit number $x = \pm 2^e \cdot m \neq 0$ is 2^{e-f+1}. By writing x as a fixed point bit string, we may speak of the *leading bit* of x being located at position e, and the *trailing bit* of x being located at position $e - f + 1$, the leading bit of 2^{e+1} being one position to the left of the leading bit of 2^e, and so on. We may also speak of the *rightmost nonzero bit* of x, which may lie anywhere from e to $e - f + 1$ inclusive.

We will occasionally use

Fact 1. *The rightmost nonzero bit of $\widehat{SUM}_k$ must lie at or to the left of the trailing bit of s_k. This is because $\widehat{SUM}_k$ is gotten by summing s_1 through s_k, whose trailing bits are at or to the left of the trailing bit of s_k, since the exponents of s_1 through s_k are in decreasing order.*

To proceed with the proof of Theorem 1 we define I so that SUM_I is the last summand that is computed *exactly*. In other words $SUM_i = \widehat{SUM}_i$ for $1 \le i \le I$, and $SUM_{I+1} \neq \widehat{SUM}_{I+1}$. Note that this means $SUM_I \neq 0$, for otherwise $\widehat{SUM}_{I+1} = s_{I+1}$ would be exact. We may also assume $I < n$ for otherwise there would be nothing to prove.

We may also assume that SUM_I is normalized. To see this, let E_{MIN} be the exponent of the smallest positive normalized number, and suppose that SUM_I is subnormal so that it requires at most $F-1$ bits to represent. If $e_{I+1} < E_{\text{MIN}}$, then both SUM_I and s_{I+1} occupy the same $(F-1)$-bit range $E_{\text{MIN}}-1$ to $E_{\text{MIN}}-F+1$, so SUM_{I+1} can be represented exactly in F bits. If $e_I \ge E_{\text{MIN}}$, then by Fact 1, the rightmost nonzero bit of SUM_I must be at or to the left of the trailing bit of s_I, i.e., the bits of SUM_I are all in the range from $E_{\text{MIN}}-1$ to $e_{I+1}-f+1$, inclusive. Thus both s_{I+1} and SUM_I lie in the f-bit range e_{I+1} to $e_{I+1}-f+1$, so the sum SUM_{I+1} is also exactly representable in F bits. Hence in either case the next sum SUM_{I+1} is computed exactly, contradicting the definition of I. Thus we can henceforth assume that SUM_I is normalized, and so $2^{E_I+1} > |SUM_I| \ge 2^{E_I}$.

The strategy of the proof will be to show that $|\widehat{SUM}_n|$ cannot be too much smaller than $|SUM_I|$, and that the rounding errors can also be bounded proportionally to $|SUM_I|$, and use this to establish a relative error bound. For most of the proof, we will be content to establish a bound on the relative error before the final rounding to f bits, i.e. a bound on

$$\frac{|\widehat{SUM}_n - S|}{|\widehat{SUM}_n|}.$$

In Section 8.9, we will translate this bound into a bound on $|\widehat{sum} - S|$.

The proof considers a number of cases. To make the case analysis clear, we consider two independent ways of dividing all the situations.

The first way to divide up the situations considers the relative locations of the trailing bits of SUM_I and s_{I+1}. The following 3 properties clearly divide all possible situation into 3 disjoint sets:

Property 1: The trailing bit of s_{I+1} is located to the right of the trailing bit of SUM_I: $e_{I+1} - f + 1 < E_I - F + 1$.

Property 2: The trailing bit of s_{I+1} is at the same location as the trailing bit of SUM_I: $e_{I+1} - f + 1 = E_I - F + 1$.

Property 3: The trailing bit of s_{I+1} is located to the left of the trailing bit of SUM_I: $e_{I+1} - f + 1 > E_I - F + 1$.

The second way to divide up the situations considers the leftmost location of the leading bits of $\widehat{SUM}_{I+1}$ through $\widehat{SUM}_n$:

Property A: The leftmost leading bit of $\widehat{SUM}_{I+1}$ through $\widehat{SUM}_n$ is at the same location or to the right of the leading bit of SUM_I: $\max_{k>I} E_k \le E_I$.

Property B: The leftmost leading bit of $\widehat{SUM}_{I+1}$ through $\widehat{SUM}_n$ is to the left of the leading bit of SUM_I: $\max_{k>I} E_k > E_I$.

Now we may consider six cases, labeled 1A, 1B, 2A, 2B, 3A and 3B, according to which pair of properties holds. We may also have subcases of these cases depending on the size of n. There may be further subcases depending on when the exponents e_k and E_j further decrease below their initial levels.

We would like to believe a simpler proof exists, but have not managed to find one.

8.1 Case 1A - $n \leq \bar{n} + 1$

Property 1 means $e_{I+1} \leq E_I - F + f - 1$, so let K be the smallest integer in the range $I \leq K \leq n$ such that $e_k \leq E_I - F + f - 2$ for all $k > K$. In other words, e_{I+1} through e_K are all $E_I - F + f - 1$, and e_{K+1} through e_n are all at most $E_I - F + f - 2$. Note that either list, but not both, can be vacuous. Thus we have the bounds

$$|s_k| \leq \begin{cases} 2^{E_I - F + f}(1 - 2^{-f}) & \text{for } I + 1 \leq k \leq K \\ 2^{E_I - F + f - 1}(1 - 2^{-f}) & \text{for } K + 1 \leq k \leq n \end{cases} \tag{9}$$

Property A implies $E_k \leq E_I$ for all $k \geq I$, so let J be the largest integer in the range $I \leq J \leq n$ such that $E_J = E_I$ but $E_j < E_I$ for all $j > J$. In other words $\widehat{SUM}_J$ is the last computed partial sum with the exponent E_I. This enables us to bound 1 ulp on the partial sums:

$$\text{ulp}(\widehat{SUM}_j) \leq \begin{cases} 2^{E_I - F + 1} & \text{for } I \leq j \leq J \\ 2^{E_I - F} & \text{for } J + 1 \leq j \leq n \end{cases} \tag{10}$$

We consider the cases $J \leq K$ and $K < J$ separately.

8.1.1 Case $J \leq K$

In this case, we have $1 \leq I \leq J \leq K \leq n$. The additions of s_{I+1} through s_J, resulting in $\widehat{SUM}_{I+1}$ through $\widehat{SUM}_J$, can yield a maximum roundoff error of half an ulp in each of $\widehat{SUM}_{I+1}$ through $\widehat{SUM}_J$, which is at most $2^{E_I - F}$ each. If $K \geq J + 1$, then addition of s_{J+1} causes *no roundoff*, since $\widehat{SUM}_{J+1}$ is computed by exact cancellation. Additions of s_{J+2} through s_K to the partial sums $\widehat{SUM}_{J+1}$ through $\widehat{SUM}_{K-1}$, resulting in the partial sums $\widehat{SUM}_{J+2}$ through $\widehat{SUM}_K$, also causes *no roundoff*, since all the numbers involved occupies the same F-bit range. Finally, the additions of s_{K+1} through s_n can cause roundoff errors at most $2^{E_I - F - 1}$ each. Thus we have the roundoff error bounds

$$|\epsilon_i| \leq \begin{cases} 2^{E_I - F} & \text{for } I + 1 \leq i \leq J \\ 0 & \text{for } J + 1 \leq i \leq K \\ 2^{E_I - F - 1} & \text{for } K + 1 \leq i \leq n \end{cases} \tag{11}$$

Thus we can bound the total roundoff error

$$\begin{aligned} |\widehat{SUM}_n - S| &\leq \sum_{i=I+1}^{n} |\epsilon_i| \\ &\leq (J - I)2^{E_I - F} + (n - K)2^{E_I - F - 1} \\ &= (2J - 2I + n - K)2^{E_I - F - 1} \\ &= 2^{E_I} N_{1\text{A},\, J \leq K}(I, J, K, n), \end{aligned} \tag{12}$$

where

$$N_{1\mathrm{A},\ J\le K}(I,J,K,n) = (2J - 2I + n - K)2^{-F-1}.$$

We now bound $|\widehat{SUM}_n|$ from below by noting that $|\widehat{SUM}_J| \ge 2^{E_I}$ and using the triangle inequality:

$$\begin{aligned}
|\widehat{SUM}_n| &= |\widehat{SUM}_J + (s_{J+1} + \cdots + s_n) + (\epsilon_{J+1} + \cdots + \epsilon_n)| \\
&\ge |\widehat{SUM}_J| - \sum_{i=J+1}^{n} |s_i| - \sum_{i=J+1}^{n} |\epsilon_i| \\
&\ge 2^{E_I} - (K-J)2^{E_I+f-F}(1-2^{-f}) - (n-K)2^{E_I+f-F-1}(1-2^{-f}) - (n-K)2^{E_I-F-1} \\
&= 2^{E_I}\left[1 - (K-2J+n)2^{f-F-1}(1-2^{-f}) - (n-K)2^{-F-1}\right] \\
&= 2^{E_I} D_{1\mathrm{A},\ J\le K}(J,K,n), \qquad (13)
\end{aligned}$$

where

$$D_{1\mathrm{A},\ J\le K}(J,K,n) = 1 - (K-2J+n)2^{f-F-1}(1-2^{-f}) - (n-K)2^{-F-1}.$$

The relative error is then bounded by

$$\frac{|\widehat{SUM}_n - S|}{|\widehat{SUM}_n|} \le \frac{N_{1\mathrm{A},\ J\le K}(I,J,K,n)}{D_{1\mathrm{A},\ J\le K}(J,K,n)} \equiv RE_{1\mathrm{A},\ J\le K}(I,J,K,n). \qquad (14)$$

Note that $I = J < K$ cannot occur since means that $E_{I+1} < E_I - 1$ and $\widehat{SUM}_{I+1}$ is computed without roundoff by exact cancellation, contradicting our choice of I. Hence we must have either $I = J = K$ or $I < J \le K$, and the worst case relative error is bounded by the maximum of $RE_{1\mathrm{A},\ J\le K}(I,J,K,n)$ over the domain $U = \{(I,J,K) \mid 1 \le I = J = K \le n \text{ or } 1 \le I < J \le K \le n\}$:

$$\frac{|\widehat{SUM}_n - S|}{|\widehat{SUM}_n|} \le \max_{(I,J,K)\in U} RE_{1\mathrm{A},\ J\le K}(I,J,K,n).$$

We consider the two cases $I = J = K$ and $I < J \le K$ separately.

8.1.1.1 Case $I = J = K$. We first note that the denominator $D_{1\mathrm{A},\ J\le K}(I,I,n)$ becomes

$$D_{1\mathrm{A},\ J\le K}(I,I,n) = 1 - (n-I)2^{f-F-1}.$$

Since $(n-I) \le \bar{n}$, we can use bound (7) to get

$$D_{1\mathrm{A},\ J\le K}(I,I,n) \ge 1 - \bar{n}2^{f-F-1} > 1 - \frac{2^{-1} + 2^{f-F-1}}{1-2^{-f}} \ge 1 - \frac{2^{-1}+2^{-2}}{1-2^{-2}} = 0.$$

Thus $n \le \bar{n} + 1$ implies that the denominator is positive.
If $(n-I) \le \bar{n} - 1$ (implied by $n \le \bar{n}$), then

$$\begin{aligned}
RE_{1\mathrm{A},\ J\le K}(I,I,I,n) &\le \frac{(\bar{n}-1)2^{-F-1}}{1-(\bar{n}-1)2^{f-F-1}} \\
&= \frac{2^{-1-f} - 2^{-F-1-r}}{(1-2^{-f}) - (2^{-1} - 2^{f-F-r-1})}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{2^{-f}(1-2^{f-F-r})}{1-2^{1-f}+2^{f-F-r}} \\
&< \frac{2^{-f}}{1-2^{1-f}}. \qquad (15)
\end{aligned}
$$

If $(n-I)=\bar{n}$ (implying $n=\bar{n}+1$), then

$$
\begin{aligned}
RE_{1\mathrm{A},\,J\le K}(I,I,I,n) &\le \frac{\bar{n}2^{-F-1}}{1-\bar{n}2^{f-F-1}} \\
&= \frac{2^{-1-f}+(1-2^{-f}-2^{-r})2^{-F-1}}{(1-2^{-f})-2^{-1}-(1-2^{-f}-2^{-r})2^{f-F-1}} \\
&= \frac{2^{-f}\left[2^{-1}+(1-2^{-f}-2^{-r})2^{f-F-1}\right]}{(1-2^{-f})-2^{-1}-(1-2^{-f}-2^{-r})2^{f-F-1}} \\
&= \frac{2^{-f}\left[1+(1-2^{-f}-2^{-r})2^{f-F}\right]}{1-2^{1-f}-(1-2^{-f}-2^{-r})2^{f-F}}. \qquad (16)
\end{aligned}
$$

To bound the last line in the above inequality, we consider the cases $F-f=1$ and $F-f\ge 2$ separately. If $F-f=1$, then $r=f-1$, and so

$$
\begin{aligned}
RE_{1\mathrm{A},\,J\le K}(I,J,K,n) &\le 2^{-f}\left[\frac{1+(1-2^{-f}-2^{-r})2^{f-F}}{1-2^{1-f}-(1-2^{-f}-2^{-r})2^{f-F}}\right] \\
&= 2^{-f}\left[\frac{1+(1-2^{-f}-2^{1-f})2^{-1}}{1-2^{1-f}-(1-2^{-f}-2^{1-f})2^{-1}}\right] \\
&= 2^{-f}\left[\frac{3(1-2^{-f})}{1-2^{-f}}\right] \\
&= 3\cdot 2^{-f}. \qquad (17)
\end{aligned}
$$

If $F-f\ge 2$, then

$$
\begin{aligned}
RE_{1\mathrm{A},\,J\le K}(I,J,K,n) &\le 2^{-f}\left[\frac{1+(1-2^{-f}-2^{-r})2^{f-F}}{1-2^{1-f}-(1-2^{-f}-2^{-r})2^{f-F}}\right] \\
&\le 2^{-f}\left[\frac{1+(1-2^{1-f})2^{-2}}{1-2^{1-f}-(1-2^{1-f})2^{-2}}\right] \\
&= 2^{-f}\frac{1}{3}\left[1+\frac{4}{1-2^{1-f}}\right] \\
&\le 3\cdot 2^{-f}. \qquad (18)
\end{aligned}
$$

Hence in either case, $RE_{1\mathrm{A},\,J\le K}(I,J,K,n)\le 3\cdot 2^{-f}$.

8.1.1.2 Case $I<J\le K$. We maximize $RE_{1\mathrm{A},\,J\le K}$ as follows. First, we need to confirm that the denominator $D_{1\mathrm{A},\,J\le K}(I,J,K,n)$ remains positive over the range of parameters, so that $RE_{1\mathrm{A},\,J\le K}(I,J,K,n)$ is bounded. Then we compute the derivatives of $RE_{1\mathrm{A},\,J\le K}(I,J,K,n)$ with respect to J and K in order to find the maximum.

We first consider minimizing $D_{1\mathrm{A},\ J\le K}(I,J,K,n)$. We see easily that $D_{1\mathrm{A},\ J\le K}$ is an affine function of J, K and n:

$$D_{1\mathrm{A},\ J\le K}(J,K,n) = 1 - K(2^{f-F-1}(1-2^{1-f})) + J(2^{f-F}(1-2^{-f})) - n2^{f-F-1}.$$

Thus the minimum occurs when K is maximized and J is minimized, i.e., $J = I+1$ and $K = n$. At this point, we have

$$D_{1\mathrm{A},\ J\le K}(I,I+1,n,n) = 1 - (n-I-1)2^{f-F}(1-2^{-f}).$$

This is positive if and only if

$$n - I - 1 < \frac{2^{F-f}}{1-2^{-f}} = \theta - 1.$$

Since $n \le \bar{n}+1$ certainly implies that $n - I - 1 < \theta - 1$, the denominator remains positive.

We now proceed to the maximization of $RE_{1\mathrm{A},\ J\le K}(I,J,K,n)$. We note that the derivatives of $RE_{1\mathrm{A},\ J\le K}$ with respect to J and K are given by

$$\frac{\partial}{\partial J} RE_{1\mathrm{A},\ J\le K} = \frac{1}{2^F D^2_{1\mathrm{A},\ J\le K}} \left[1 - (n-I)2^{f-F}(1-2^{-f}) - (n-K)2^{-F-1}\right] \tag{19}$$

and

$$\frac{\partial}{\partial K} RE_{1\mathrm{A},\ J\le K} = \frac{-1}{2^{F+1} D^2_{1\mathrm{A},\ J\le K}} \left[1 - (n-I)2^{f-F}(1-2^{-f}) + (J-I)2^{-F}\right]. \tag{20}$$

We consider the cases $(n-I) \le \bar{n}-1$ and $(n-I) = \bar{n}$ separately.

Case $(n-I) \le \bar{n}-1$ (implied by $n \le \bar{n}$). We note that

$$1 - (n-I)2^{f-F}(1-2^{-f}) \ge 1 - (\bar{n}-1)2^{f-F}(1-2^{-f}) > 0,$$

so that $\frac{\partial}{\partial K} RE_{1\mathrm{A},\ J\le K} < 0$. Thus maximal $RE_{1\mathrm{A},\ J\le K}$ occurs when $J = K$. To determine J, we set $K = J$ and compute the derivative

$$\frac{\partial}{\partial J}\left[RE_{1\mathrm{A},\ J\le K}(I,J,J,n)\right] = \frac{1}{2^{F+1} D^2_{1\mathrm{A},\ J\le K}} \left[1 - (n-I)2^{f-F}\right].$$

Note that the sign of this derivative does not depend on J, so we can compute the value of $RE_{1\mathrm{A},\ J\le K}$ at the endpoints $J = n$ and $J = I+1$, depending on the sign of $\frac{\partial}{\partial J} RE_{1\mathrm{A},\ J\le K}(I,J,J,n)$. If $\frac{\partial}{\partial J} RE_{1\mathrm{A},\ J\le K}(I,J,J,n) \ge 0$, then

$$\begin{aligned} RE_{1\mathrm{A},\ J\le K}(I,J,K,n) &\le RE_{1\mathrm{A},\ J\le K}(I,n,n,n) \\ &= (n-I)2^{-F} \le (\bar{n}-1)2^{-F} \\ &= \frac{2^{-f}(1-2^{f-F-r})}{1-2^{-f}} \\ &\le \frac{2^{-f}}{1-2^{1-f}}. \end{aligned}$$

If $\frac{\partial}{\partial J} RE_{1\mathrm{A},\ J\le K}(I,J,J,n) < 0$, then

$$\begin{aligned} RE_{1\mathrm{A},\ J\le K}(I,J,K,n) &\le RE_{1\mathrm{A},\ J\le K}(I,I+1,I+1,n) \\ &\le RE_{1\mathrm{A},\ J\le K}(I,I,I,n) \\ &\le \frac{2^{-f}}{1-2^{1-f}}, \end{aligned}$$

where the last inequality comes from (15).

Case $(n-I) = \bar{n}$ (implies $n = \bar{n}+1$). Note that $(n-I) = \bar{n} > \theta - 1 = 2^{F-f}/(1-2^{-f})$, so

$$\begin{aligned} \frac{\partial}{\partial J} RE_{1\mathrm{A},\ J\le K} &= \frac{1}{2^F D^2_{1\mathrm{A},\ J\le K}} \left[1 - (n-I)2^{f-F}(1-2^{-f}) - (n-K)2^{-F-1}\right] \\ &< \frac{-(n-K)2^{-F-1}}{2^F D^2_{1\mathrm{A},\ J\le K}} \le 0, \end{aligned} \tag{21}$$

and so the maximum occurs when $J = I+1$. Thus given $J = I+1$, we have

$$\begin{aligned} \frac{\partial}{\partial K} RE_{1\mathrm{A},\ J\le K}\Big|_{J=I+1} &= \frac{-1}{2^{F+1} D^2_{1\mathrm{A},\ J\le K}} \left[1 - (n-I)2^{f-F}(1-2^{-f}) + 2^{-F}\right] \\ &= \frac{-1}{2^{F+1} D^2_{1\mathrm{A},\ J\le K}} \left[1 - (2^{F-f} + 1 - 2^{-r} - 2^{-f})2^{f-F} + 2^{-F}\right] \\ &= \frac{2^{f-F}(1-2^{-r}-2^{1-f})}{2^{F+1} D^2_{1\mathrm{A},\ J\le K}} \ge 0. \end{aligned} \tag{22}$$

Thus we want to maximize K, so we take $K = n$. Thus the maximum occurs when $J = I+1$ and $K = n$, giving

$$\begin{aligned} RE_{1\mathrm{A},\ J\le K}(I,I+1,n,n) &= \frac{2^{-F}}{1-(n-I-1)2^{f-F}(1-2^{-f})} \\ &= \frac{2^{-F}}{1-(2^{F-f}-2^{-r})2^{f-F}} \\ &= \frac{2^{-F}}{2^{f-F-r}} = 2^{r-f}. \end{aligned} \tag{23}$$

Note that if $F < 2f$, then $r = 2f - F$, so $RE_{1\mathrm{A},\ J\le K} \le 2^{2f-F} \cdot 2^{-f} < \frac{1}{2}$. Section 8.2 shows that this bound can be tightened if $F \ge 2f$ and s_2 is normalized.

8.1.2 Case $K < J$.

In this case we have $I \le K < J \le n$. As before addition of s_{I+1} through s_J can yield a roundoff error of at most $2^{E_I - F}$ each. Addition of s_{J+1} through s_n can cause roundoff error of at most $2^{E_I - F - 1}$ each. Thus we have the bounds

$$|\epsilon_i| \le \begin{cases} 2^{E_I - F} & \text{for } I+1 \le i \le J \\ 2^{E_I - F - 1} & \text{for } J+1 \le i \le n. \end{cases}$$

Thus the total roundoff error is bounded by

$$\begin{aligned} |\widehat{SUM}_n - S| &\leq (J-I)2^{E_I-F} + (n-J)2^{E_I-F-1} \\ &= 2^{E_I}(J-2I+n)2^{-F-1} \\ &= 2^{E_I} N_{1\mathrm{A},\, K<J}(I,J,n), \end{aligned} \tag{24}$$

where $N_{1\mathrm{A},\, K<J}(I,J,n) = (J-2I+n)2^{-F-1}$. We can bound $|\widehat{SUM}_n|$ from below:

$$\begin{aligned} |\widehat{SUM}_n| &= |\widehat{SUM}_J + (s_{J+1} + \cdots + s_n) + (\epsilon_{J+1} + \cdots + \epsilon_n)| \\ &\geq |\widehat{SUM}_J| - \sum_{i=J+1}^{n} |s_i| - \sum_{i=J+1}^{n} |\epsilon_i| \\ &\geq 2^{E_I} - (n-J)2^{E_I+f-F-1}(1-2^{-f}) - (n-J)2^{E_I-F-1} \\ &= 2^{E_I}\left[1 - (n-J)2^{f-F-1}\right] \\ &= 2^{E_I} D_{1\mathrm{A},\, K<J}(I,J,n), \end{aligned} \tag{25}$$

where

$$D_{1\mathrm{A},\, K<J}(I,J,n) = 1 - (n-J)2^{f-F-1}.$$

Now we want to maximize

$$RE_{1\mathrm{A},\, K<J}(I,J,n) = \frac{N_{1\mathrm{A},\, K<J}(I,J,n)}{D_{1\mathrm{A},\, K<J}(I,J,n)}$$

over $1 \leq I \leq K < J < n$. But notice that

$$RE_{1\mathrm{A},\, K<J}(I,J,n) = RE_{1\mathrm{A},\, J\leq K}(I,J,J,n).$$

The domain we are maximizing, $I < J \leq n$, is a subset of the domain we are maximizing in section 8.1.1.2, so the same bounds holds on $RE_{1\mathrm{A},\, K<J}$.

8.2 Case 1A - $n = \bar{n} + 1$, $F \geq 2f$

If $F \geq 2f$, we can further tighten the error bound for the case $n = \bar{n} + 1$ using the fact that SUM_1 cannot have full span of F bits. A part of this section (section 8.2.2) assumes that s_2 is normalized, and thus the error bound shown in this section (claim 2 of Theorem 1) does not apply to Theorem 2.

The only places where the error bound is larger than claim 2 of Theorem 1 are bounds (17), (18), and (23). Of these, bound (17) does not apply, since we are assuming $F \geq 2f$, which implies $F - f \geq 2$. Bound (18) can be dealt with ease by considering the cases $F - f = 2$ and $F - f \geq 3$ separately. If $F - f = 2$, then $f = 2$, and thus

$$\begin{aligned} RE_{1\mathrm{A},\, J\leq K}(I,J,K,n) &\leq 2^{-f}\frac{1}{3}\left[1 + \frac{4}{1-2^{1-f}}\right] \\ &= 2^{-f}\frac{1.5}{1-2^{1-f}}. \end{aligned}$$

If $F - f \geq 3$, then

$$\begin{aligned} RE_{1\text{A},\ J\leq K}(I,J,K,n) &\leq 2^{-f}\left[\frac{1+(1-2^{-f}-2^{-r})2^{f-F}}{1-2^{1-f}-(1-2^{-f}-2^{-r})2^{f-F}}\right] \\ &\leq 2^{-f}\left[\frac{1+(1-2^{1-f})2^{-3}}{1-2^{1-f}-(1-2^{1-f})2^{-3}}\right] \\ &= 2^{-f}\frac{1}{7}\left[\frac{9-2^{1-f}}{1-2^{1-f}}\right] \\ &< 2^{-f}\frac{1.5}{1-2^{1-f}}. \end{aligned}$$

Hence we now focus on tightening bound (23). Note that this inequality can be reached via section 8.1.2 as well. In addition, both of these sections can be reached via 8.3. Thus assume $I < J$ as in sections 8.1.1.2 and 8.1.2, and also assume $(n - I) = \bar{n}$ (which implies $I = 1$ and $n = \bar{n} + 1$).

If $J > I + 1 = 2$, then $\frac{\partial}{\partial J} RE_{1\text{A},\ J\leq K}$ shown in (19) is still negative (as in (21)), so maximal $RE_{1\text{A},\ J\leq K}$ occurs when $J = I + 2$. Given $J = I + 2$, we evaluate $\frac{\partial}{\partial K} RE_{1\text{A},\ J\leq K}$:

$$\frac{\partial}{\partial K} RE_{1\text{A},\ J\leq K}\Big|_{J=I+2} = \frac{2^{f-F}(1-2^{-r}-3\cdot 2^{-f})}{2^{F+1}D^2_{1\text{A},\ J\leq K}}.$$

Note that the above expression is negative if $r = 1$ and $f = 2$, and non-negative otherwise. Thus if $r = 1$ and $f = 2$, then $\bar{n} = 1 + (2^F - 2)/3$, and so

$$\begin{aligned} RE_{1\text{A},\ J\leq K}(I,J,K,n) &\leq RE_{1\text{A},\ J\leq K}(I,I+2,I+2,n) \\ &= \frac{(n-I+2)2^{-F-1}}{1-(n-I-2)2^{f-F-1}} \\ &= 2^{-f}\left[\frac{(\bar{n}+2)2^{1-F}}{1-(\bar{n}-2)2^{1-F}}\right] \\ &= 2^{-f}\left[\frac{2+14\cdot 2^{-F}}{1+10\cdot 2^{-F}}\right] \\ &< 2\cdot 2^{-f}. \end{aligned}$$

Otherwise $\frac{\partial}{\partial K} RE_{1\text{A},\ J\leq K}$ remains non-negative, and

$$\begin{aligned} RE_{1\text{A},\ J\leq K}(I,J,K,n) &\leq RE_{1\text{A},\ J\leq K}(I,I+2,n,n) \\ &= \frac{2\cdot 2^{-F}}{1-(n-I-2)2^{f-F}(1-2^{-f})} \\ &= \frac{2\cdot 2^{-F}}{1-(\bar{n}-2)2^{f-F}(1-2^{-f})} \\ &= \frac{2\cdot 2^{-f}}{1+2^{-r}-2^{-f}} \\ &\leq 2\cdot 2^{-f}. \end{aligned}$$

The remaining case is when $J = I + 1 = 2$. We split this case into two subcases, depending on whether $K = I$ or $K > I$. (The former case can arise from section 8.1.2.)

8.2.1 Case $K = I = 1$

This case arises from section 8.1.2, so we consider the relative error bound $RE_{1\mathrm{A},\ K<J}(I, J, n)$. Now note that

$$\frac{\partial}{\partial J} RE_{1\mathrm{A},\ K<J} = \frac{1}{2^{F+1} D^2_{1\mathrm{A},\ K<J}} \left[1 - (n - I)2^{f-F}\right],$$

which is negative when $n = \bar{n} + 1$. Hence the bound is maximized when $J = I + 1$, and thus (using (8)),

$$\begin{aligned}
RE_{1\mathrm{A},\ K<J}(I, J, n) &\leq RE_{1\mathrm{A},\ K<J}(I, I + 1, n) \\
&= \frac{(n - I + 1)2^{-F-1}}{1 - (n - I - 1)2^{f-F-1}} \\
&= \frac{(\bar{n} - 2)2^{-F-1} + 2^{-F}}{1 - (\bar{n} - 2)2^{f-F-1}} \\
&\leq 2^{-f} \left[\frac{1 - 2^{f-F} + 2^{f-F+1} - 2^{1-F}}{1 - 2^{1-f} + 2^{f-F}}\right] \\
&< 2^{-f} \left[\frac{1 + 2^{1+f-F}}{1 - 2^{1-f}}\right] \\
&\leq 2^{-f} \left[\frac{1.5}{1 - 2^{1-f}}\right].
\end{aligned}$$

8.2.2 Case $K > I = 1$

This subsection is the only part of the proof that assumes that s_2 is normalized.

We will show that

$$|\widehat{SUM}_2| \geq 2^{E_1}(1 + 2^{f-F-1}),$$

so that the bound on the denominator $|\widehat{SUM}_n|$ can be tightened (remember that we are assuming that $I = 1$ and $J = 2$).

Since $SUM_1 = s_1$, SUM_1 must be an f-bit number. Thus either $|SUM_1| = 2^{E_1}$, or $|SUM_1| \geq 2^{E_1}(1 + 2^{1-f})$. In the former case, SUM_1 and s_2 must have the same signs (since $E_1 = E_2$), so

$$|\widehat{SUM}_2| \geq 2^{E_1} + 2^{E_1-F+f-1} = 2^{E_1}(1 + 2^{f-F-1}).$$

In the latter case,

$$\begin{aligned}
|\widehat{SUM}_2| &\geq |SUM_1| - |s_2| \\
&\geq 2^{E_1}(1 + 2^{1-f}) - 2^{E_1-F+f} \\
&= 2^{E_1}(1 + 2^{-f} + 2^{-f} - 2^{f-F}) \\
&\geq 2^{E_1}(1 + 2^{-f}) \\
&> 2^{E_1}(1 + 2^{f-F-1}),
\end{aligned}$$

assuming $F \geq 2f$.

Thus the denominator in the relative error bound is slightly larger:

$$\begin{aligned}
|\widehat{SUM}_n| &\geq 2^{E_1}(1+2^{f-F-1}) - (K-J)2^{E_1+f-F}(1-2^{-f}) - \\
&\quad (n-K)2^{E_1+f-F-1}(1-2^{-f}) - (n-K)2^{E_1-F-1} \\
&= 2^{E_1}\left[1+2^{f-F-1} - (K-4+n)2^{f-F-1}(1-2^{-f}) - (n-K)2^{-F-1}\right] \\
&= 2^{E_1}\left[D_{1\mathrm{A},\, J\leq K}(2,K,n) + 2^{f-F-1}\right] \\
&\equiv 2^{E_1} D_{1\mathrm{A},\, J\leq K,\, F\geq 2f}(K,n)
\end{aligned}$$

The total roundoff error is still

$$|\widehat{SUM}_n - S| \leq 2^{E_1} N_{1\mathrm{A},\, J\leq K}(1,2,K,n) = 2^{E_1}(n-K+2)2^{-F-1}.$$

Thus we have the relative error bound

$$\frac{|\widehat{SUM}_n - S|}{|\widehat{SUM}_n|} \leq \frac{N_{1\mathrm{A},\, J\leq K}(1,2,K,n)}{D_{1\mathrm{A},\, J\leq K,\, F\geq 2f}(K,n)} \equiv RE_{1\mathrm{A},\, J\leq K,\, F\geq 2f}(K,n).$$

We can now compute the derivative

$$\frac{\partial}{\partial K} RE_{1\mathrm{A},\, J\leq K,\, F\geq 2f} = \frac{-1}{2^{F+1} D^2_{1\mathrm{A},\, J\leq K,\, F\geq 2f}} \left[1 + 3\cdot 2^{f-F-1} - 2^{f-F}(1-2^{-f})n\right].$$

Since the sign of the derivative does not depend on K, we can just evaluate $RE_{1\mathrm{A},\, J\leq K,\, F\geq 2f}$ at the two endpoints $K = J = 2$ and $K = n$. Thus

$$RE_{1\mathrm{A},\, J\leq K,\, F\geq 2f}(n,n) = \frac{2^{-F}}{1+2^{f-F-1} - (\bar{n}-1)2^{f-F}(1-2^{-f})} = \frac{2\cdot 2^{-f}}{1+2^{1-r}} < 2\cdot 2^{-f}$$

and

$$\begin{aligned}
RE_{1\mathrm{A},\, J\leq K,\, F\geq 2f}(2,n) &= \frac{n2^{-F-1}}{1+2^{f-F-1} - (n-2)2^{f-F-1}} \\
&= 2^{-f}\left[\frac{(\bar{n}-1)2^{f-F-1} + 2^{f-F}}{1-(\bar{n}-2)2^{f-F-1}}\right] \\
&\leq 2^{-f}\left[\frac{1+2^{1+f-F} - 2^{1-F}}{1-2^{1-f}+2^{f-F}}\right] \\
&\leq 2^{-f}\cdot 2\left[\frac{1+2^{1+f-F} - 2^{1-F}}{1+2^{1+f-F}}\right] \\
&< 2^{-f}\cdot 2
\end{aligned}$$

Hence in either case the relative error is bounded by $2\cdot 2^{-f}$.

8.3 Case 1B - $n \leq \bar{n} + 1$

This case differs from case 1A since property B implies $\max_{k>I} E_k > E_I$. We now show that $\max_{k>I} E_k = E_I + 1$. Suppose not, and let M be the smallest index in the range $I < M \leq n$ such that $E_M = E_I + 2$. Then additions of s_{I+1} through s_{M-1} causes roundoff of at most $2^{E_I - F + 1}$, while the addition of s_M may cause a roundoff of at most $2^{E_I - F + 2}$. Then $(M - I) \leq \bar{n}$, so

$$\begin{aligned}
|\widehat{SUM}_M| &\leq |\widehat{SUM}_I| + \sum_{j=I+1}^{M} |s_j| + \sum_{j=I+1}^{M} |\epsilon_j| \\
&\leq 2^{E_I+1}(1 - 2^{-F}) + (M - I)2^{E_I+f-F}(1 - 2^{-f}) + (M - I)2^{E_I-F+1} + 2^{E_I-F+1} \\
&= 2^{E_I+1}\left[1 - 2^{-F} + (M - I)2^{f-F-1}(1 - 2^{-f}) + (M - I)2^{-F} + 2^{-F}\right] \\
&= 2^{E_I+1}\left[1 + (M - I)2^{f-F-1}(1 + 2^{-f})\right] \\
&\leq 2^{E_I+1}\left[1 + \frac{2^{F-f} + 1 - 2^{-f} - 2^{-r}}{1 - 2^{-f}} 2^{f-F-1}(1 + 2^{-f})\right].
\end{aligned}$$

To bound the last line, we consider $F - f = 1$ and $F - f \geq 2$ separately. If $F - f = 1$, then $r = f - 1$, so

$$|\widehat{SUM}_M| \leq 2^{E_I+1}\left[1 + \frac{3 - 3 \cdot 2^{-f}}{1 - 2^{-f}} \frac{5}{16}\right] = 2^{E_I+1}\left[1 + \frac{15}{16}\right] < 2^{E_I+2}.$$

If $F - f \geq 2$, then

$$\begin{aligned}
|\widehat{SUM}_M| &\leq 2^{E_I+1}\left[1 + \frac{1 + 2^{f-F} - 2^{1-F}}{1 - 2^{-f}} 2^{-1}(1 + 2^{-f})\right] \\
&\leq 2^{E_I+1}\left[1 + \frac{1 + 2^{-2} - 2^{-1-f}}{1 - 2^{-f}} \cdot \frac{5}{8}\right] \\
&= 2^{E_I+1}\left[1 + \frac{5 - 2 \cdot 2^{-f}}{1 - 2^{-f}} \cdot \frac{5}{32}\right] \\
&\leq 2^{E_I+1}\left[1 + \frac{15}{16}\right] \\
&< 2^{E_I+2}.
\end{aligned}$$

Hence we see that the maximum exponent E_M is $E_I + 1$. We define K as in case 1A, but define J slightly differently as follows, using Property B. Let J be the largest integer in the range $I \leq J \leq n$ such that $E_J = E_I + 1$ but $E_j \leq E_I$ for all $j > J$. This means that $J > I$, and

$$\text{ulp}(E_i) \leq \begin{cases} 2^{E_I-F+2} & \text{for } I + 1 \leq i \leq J \\ 2^{E_I-F+1} & \text{for } J + 1 \leq i \leq n \end{cases}$$

Since roundoff in computing $\widehat{SUM}_i$ is at most half an ulp of $\widehat{SUM}_i$, we see that

$$|\epsilon_i| \leq \begin{cases} 2^{E_I-F+1} & \text{for } I + 1 \leq i \leq J \\ 2^{E_I-F} & \text{for } J + 1 \leq i \leq n \end{cases}$$

Thus the total roundoff is bounded by

$$|\widehat{SUM}_n - S| \le (J-I)2^{E_I-F+1} + (n-J)2^{E_I-F} = (J-2I+n)2^{E_I-F} = 2^{E_I+1}N_{1\mathrm{B}}(I,J,n),$$

where $N_{1\mathrm{B}}(I,J,n) = (J-2I+n)2^{-F-1}$. We can bound $|\widehat{SUM}_n|$ below as follows

$$\begin{aligned}
|\widehat{SUM}_n| &= \left|\widehat{SUM}_J + \sum_{j=J+1}^{n}(s_j+\epsilon_j)\right| \\
&\ge |\widehat{SUM}_J| - \sum_{j=J+1}^{n}|s_j| - \sum_{j=J+1}^{n}|\epsilon_j| \\
&\ge 2^{E_I+1} - (n-J)2^{E_I-F+f}(1-2^{-f}) - (n-J)2^{E_I-F} \\
&= 2^{E_I+1}\left[1-(n-J)2^{f-F-1}\right] \\
&= 2^{E_I+1}D_{1\mathrm{B}}(J,n)
\end{aligned}$$

where $D_{1\mathrm{B}}(J,n) = 1-(n-J)2^{f-F-1}$. We immediately see that $n \le \bar{n}+1$ implies $|\widehat{SUM}_n| > 0$.

Thus we have

$$RE_{1\mathrm{B}}(I,J,n) = \frac{N_{1\mathrm{B}}(I,J,n)}{D_{1\mathrm{B}}(J,n)} = \frac{(J-2I+n)2^{-F-1}}{1-(n-J)2^{f-F-1}}.$$

Here note that

$$RE_{1\mathrm{B}}(I,J,n) = RE_{1\mathrm{A},\ \mathrm{K<J}}(I,J,n),$$

so the same analysis in Section 8.1.2 applies.

8.4 Case 2A

We will show that case 2A cannot occur by showing that $\widehat{SUM}_{I+1}$ would be computed exactly. The nonzero bits of $\widehat{SUM}_I$ lie between positions E_I through $E_I - F + 1$, inclusive. Property 2 says that the trailing bit of s_{I+1} is in the same location as the trailing bit of $\widehat{SUM}_I$. Since the exponent of $\widehat{SUM}_{I+1}$ does not increase (by property A), the nonzero bits of $\widehat{SUM}_I$, $\widehat{SUM}_{I+1}$, and s_{I+1} are all in same range of F bits from E_I to $E_I - F + 1$. Thus $\widehat{SUM}_{I+1} = \widehat{SUM}_I + s_{I+1}$ is computed exactly.

8.5 Case 2B

Property 2 means that $e_{I+1} = E_I - F + f$, so we can define K to be the smallest integer in the range $I \le K \le n$ such that $e_k \le E_I - F + f - 1$ for $k > K$. In other words, e_{I+1} through e_K are all equal to $E_I + f - F$ and e_{K+1} through e_n are at most $E_I + f - F - 1$. This implies that

$$|s_k| \le \begin{cases} 2^{E_I+f-F+1}(1-2^{-f}) & \text{for } I+1 \le k \le K \\ 2^{E_I+f-F}(1-2^{-f}) & \text{for } K+1 \le k \le n \end{cases}$$

Note that this definition of K is almost the same as in Case 1A (see (9)), except that all exponents are increased by 1.

Property B says that $\max_{k>I} E_k > E_I$. We first show that $E_k \le E_I + 2$ for all k, $I < k \le n$. Suppose not, and let $M > I$ be the smallest index in the range $I < M \le n$ such that $E_M = E_I + 3$. Then

$$\begin{aligned}
|\widehat{SUM}_M| &\le |\widehat{SUM}_I| + \sum_{i=I+1}^{M} |s_i| + \sum_{i=I+1}^{M} |\epsilon_i| \\
&\le 2^{E_I+1}(1-2^{-F}) + (M-I)2^{E_I+f-F+1}(1-2^{-f}) + (M-I)2^{E_I-F+2} + 2^{E_I-F+2} \\
&= 2^{E_I+1}\left[1 + 2^{-F} + (M-I)2^{f-F}(1+2^{-f})\right] \\
&\le 2^{E_I+1}\left[1 + 2^{-F} + \frac{1+2^{f-F}}{1-2^{-f}}(1+2^{-f})\right] \\
&\le 2^{E_I+1}\left[1 + 1/8 + \frac{1+1/2}{1-1/4}(5/4)\right] \\
&= 2^{E_I+1}\frac{29}{8} < 2^{E_I+3}
\end{aligned}$$

Hence the maximum exponent that can occur is $E_I + 2$.

Now let M be the smallest integer (in the range $I < M \le n$) such that E_M is the maximum exponent. Then there are two cases to consider: $E_M = E_I + 1$ and $E_M = E_I + 2$.

8.5.1 $E_M = E_I + 1$

Since $\max_{k>I} E_k = E_I + 1$, so we can let J be the largest index in the range $I \le J \le n$ such that $E_J = E_I + 1$ but $E_j \le E_I$ for any $j > J$. This implies that 1 ulp in $\widehat{SUM}_{I+1}$ through $\widehat{SUM}_J$ is at most 2^{E_I-F+2}, and that 1 ulp in $\widehat{SUM}_{J+1}$ through $\widehat{SUM}_n$ is at most 2^{E_I-F+1}. Note that $J > I$, since E_{I+1} must equal $E_I + 1$, for otherwise there would be no roundoff when computing $\widehat{SUM}_{I+1}$, contradicting the definition of I. This definition of J and associated bounds are analogous to those of Case 1A, except that all exponents are increased by 1. Hence the definition of J and K are both identical as in Case 1A except for the increased exponent, which cancels out of the final relative error bound (14). Thus analogous analysis as in Case 1A applies, except that of section 8.2.2 (which deals with the case $F \ge 2f$, s_2 normalized, $I = 1$, $J = 2$, $K > I$).

We now show that this exceptional case (where $F \ge 2f$, s_2 normalized, $I = 1$, $J = 2$, $K > I$) cannot happen. Note that $\widehat{SUM}_1$ must be an f-bit number, so $|\widehat{SUM}_1| \le 2^{E_1+1}(1-2^{-f})$. Since $E_2 = E_1 + 1$, we must have $|s_2 + \epsilon_2| \ge 2^{E_1+1-f}$. If $|s_2| < 2^{E_1+1-f}$, then $|SUM_2| < 2^{E_1+1}$, so we must have $|\epsilon_2| \le \frac{1}{2}\text{ulp}(SUM_2) = 2^{E_1-F}$. In this case,

$$\begin{aligned}
|\widehat{SUM}_2| &= |SUM_1 + s_2 + \epsilon_2| \\
&\le 2^{E_1+1}(1-2^{-f}) + 2^{E_1+1-f}(1-2^{-f}) + 2^{E_1-F} \\
&= 2^{E_1+1}(1 - 2^{-2f} + 2^{-1-F}) \\
&< 2^{E_1+1},
\end{aligned}$$

which contradicts the assumption that $E_2 = E_1 + 1$. Hence we must have $|s_2| \ge 2^{E_1+1-f}$. By property B we also have $|s_2| < 2^{E_1+f-F+1}$, so

$$2^{E_1+1-f} \le |s_2| < 2^{E_1+f-F+1},$$

which implies $E_1 + 1 - f < E_1 + f - F + 1$, or $F < 2f$, again a contradiction. Hence the case in section 8.2.2 cannot occur.

8.5.2 $E_M = E_I + 2$

We must have

$$\begin{aligned} 2^{E_I+2} &\leq |\widehat{SUM}_M| \\ &\leq |\widehat{SUM}_I| + \sum_{i=I+1}^{M} |s_i| + \sum_{i=I+1}^{M} |\epsilon_i| \\ &\leq 2^{E_I+1}(1-2^{-F}) + (M-I)2^{E_I+f-F+1}(1-2^{-f}) + (M-I)2^{E_I-F+1} + 2^{E_I-F+1} \\ &= 2^{E_I+1}\left[1 + (M-I)2^{f-F}\right] \end{aligned}$$

Hence we must have $(M-I) \geq 2^{F-f}$. Note that if $M - I = 2^{F-f}$, then the last addition just barely rounds up to 2^{E_I+2}, so the roundoff error (in the last addition) is at most 2^{E_I-F+1}, not 2^{E_I-F+2}. This implies that $E_M = E_I$, so this case cannot occur. Thus we must have $M \geq 2^{F-f} + 2$.

The total roundoff error is bounded by

$$\begin{aligned} |S - \widehat{SUM}_n| &\leq (M-I-1)2^{E_I-F+1} + (n-M+1)2^{E_I-F+2} \\ &= 2^{E_I+1}(2n-M-I+1)2^{-F}. \end{aligned}$$

We now consider the cases $F < 2f$ and $F \geq 2f$ separately.

8.5.2.1 Case $F < 2f$.

In this case, we must have $M = n = 2^{F-f} + 2$, so

$$|S - \widehat{SUM}_n| \leq 2^{E_I+1}(2^{F-f}+2)2^{-F} = 2^{E_I+1}(2^{-f} + 2^{1-F}).$$

Since $M = n$, we have $|\widehat{SUM}_n| = |\widehat{SUM}_M| \geq 2^{E_I+2}$. Thus the relative error is bounded by

$$\frac{|S - \widehat{SUM}_n|}{|\widehat{SUM}_n|} \leq 2^{-1-f} + 2^{-F} \leq 2^{-f}.$$

8.5.2.2 Case $F \geq 2f$.

We first bound the denominator $|\widehat{SUM}_n|$:

$$\begin{aligned} |\widehat{SUM}_n| &\geq |\widehat{SUM}_M| - \sum_{i=M+1}^{n} |s_i| - \sum_{i=M+1}^{n} |\epsilon_i| \\ &\geq 2^{E_I+2} - (n-M)2^{E_I+f-F+1}(1-2^{-f}) - (n-M)2^{E_I-F+2} \\ &= 2^{E_I+1}\left[2 - (n-M)2^{f-F}(1+2^{-f})\right] \\ &\geq 2^{E_I+1}\left[2 - (\bar{n}-1-2^{F-f})2^{f-F}(1+2^{-f})\right] \\ &> 2^{E_I+1}\left[2 - \frac{2^{-f}}{1-2^{-f}}(1+2^{-f})\right] \\ &\geq 2^{E_I+1}\left[\frac{19}{12}\right]. \end{aligned}$$

Now if $n \le \bar{n}$, then

$$\begin{aligned} |S - \widehat{SUM}_n| &\le 2^{E_I+1}(2n - M - I + 1)2^{-F} \\ &\le 2^{E_I+1}(2\bar{n} - 2^{F-f} - 2)2^{-F} \\ &\le 2^{E_I+1}\left[\frac{2^{F-f} + 2^{F-2f}}{1 - 2^{-f}}\right]2^{-F} \\ &= 2^{E_I+1}2^{-f}\left[\frac{1 + 2^{-f}}{1 - 2^{-f}}\right] \\ &\le 2^{E_I+1}\frac{2^{-f}}{1 - 2^{-f}}\frac{5}{4} \end{aligned}$$

This gives the bound on the relative error

$$\frac{|S - \widehat{SUM}_n|}{|\widehat{SUM}_n|} \le \frac{15}{19}\frac{2^{-f}}{1 - 2^{-f}} \le \frac{2^{-f}}{1 - 2^{1-f}}.$$

Now consider when $n = \bar{n} + 1$. In this case,

$$\begin{aligned} |S - \widehat{SUM}_n| &\le 2^{E_I+1}(2n - M - I + 1)2^{-F} \\ &\le 2^{E_I+1}(2\bar{n} - 2^{F-f})2^{-F} \\ &< 2^{E_I+1}\left[\frac{2^{-f} + 2^{1-F} + 2^{-2f}}{1 - 2^{-f}}\right] \\ &= 2^{E_I+1}2^{-f}\left[\frac{1 + 2^{1+f-F} + 2^{-f}}{1 - 2^{-f}}\right] \\ &\le 2^{E_I+1}2^{-f}\frac{9/4}{1 - 2^{-f}}. \end{aligned}$$

So this gives the bound on the relative error when $n = \bar{n} + 1$:

$$\frac{|S - \widehat{SUM}_n|}{|\widehat{SUM}_n|} \le \frac{27}{19}\frac{2^{-f}}{1 - 2^{-f}} < 2^{-f}\left[\frac{1.5}{1 - 2^{1-f}}\right].$$

8.6 Cases 3A and 3B

Finally we consider the easy cases 3A and 3B, and argue that they cannot occur, because $\widehat{SUM}_{I+1} = SUM_{I+1}$ would in fact be computed exactly.

Fact 1 means that $e_{I+1} - f + 1 \le E_I$ since $SUM_I \neq 0$, and in fact the nonzero bits of SUM_I lie between E_I and $e_{I+1} - f + 1$ inclusive. Since the nonzero bits of s_{I+1} lie between e_{I+1} and $e_{I+1} - f + 1$ inclusive, we conclude that all the nonzero bits of SUM_I and s_{I+1} lie between $\max(E_I, e_{I+1})$ and $e_{I+1} - f + 1$, a maximum number of nonzero bits equal to

$$\max(E_I, e_{I+1}) - (e_{I+1} - f + 1) + 1 = \max(E_I - e_{I+1} + f, f) \le \max(F - 1, f) = F - 1.$$

Adding (or subtracting) bits that occupy at most $F - 1$ bit positions yields a sum (or difference) that occupies at most F bit positions, which can be represented exactly in an F-bit register. So $\widehat{SUM}_{I+1} = SUM_{I+1} = SUM_I + s_{I+1}$ is computed exactly.

8.7 Sign of S when $n \leq \bar{n} + 1$

Note that if the relative error

$$\frac{|\widehat{SUM}_n - S|}{|\widehat{SUM}_n|} \tag{26}$$

is less than one, then the sign of S is exactly that of $\widehat{SUM}_n$, i.e., S is positive, zero, and negative exactly when $\widehat{SUM}_n$ is positive, zero, and negative, respectively. We also note that rounding to f bits to obtain $\widehat{sum}$ does not affect the sign, and does not round to zero unless $\widehat{SUM}_n$ is zero (this follows from Fact 1). Thus if the relative error (26) is less than one then $\widehat{sum}$ is positive, zero, or negative exactly when S is positive, zero, or negative, respectively.

If $n \leq \bar{n} + 1$, there is only one place in the proof where the relative error reaches one, namely in (23), section 8.1.1.2, under case when $(n - I) = \bar{n}$ (which implies $n = \bar{n} + 1$ and $I = 1$). Note that relation (23) can be reached via sections 8.1.2, 8.3, and 8.5.1 as well.

We now show that in this case the rounding errors are made toward zero, so that the true sum S has the same sign as that of the computed sum $\widehat{sum}$. We can assume $r = f$ since otherwise the bound (23) remains less than one. This means that the derivative $\frac{\partial}{\partial K} RE_{1\mathrm{A},\, J \leq K}$ shown in (22) is strictly positive, so the maximum relative error of 1 only occurs when $J = I + 1$ and $K = n$. This fact precludes the cases arising from sections 8.1.2 and 8.3, since in those cases $J = K$ is assumed.

We now consider the remaining case ($r = f$, $J = I + 1 = 2$, $K = n = \bar{n} + 1$) arising from sections 8.1.1.2 and 8.5.1. Note that $r = f$ implies F is an integer multiple of f. To achieve a relative error of one, we must have $|\widehat{SUM}_2| = 2^{E_I}$, since otherwise the denominator $D_{1\mathrm{A},\, J \leq K}$ will be larger which makes it impossible to attain the maximum relative error of 1. Note that the only roundoff error is ϵ_2 (see (11)). There are two cases to consider, depending on whether the addition $\widehat{SUM}_1 + s_2$ caused a roundoff towards or away from zero. If it was away from zero (SUM_2 rounds up to the next power of 2), then the roundoff error committed is at most $2^{E_I - F - 1}$ instead of $2^{E_I - F}$. This reduces the numerator $N_{1\mathrm{A},\, J \leq K}$, making the relative error smaller than 1. If the roundoff was towards zero, then the true sum is located away from zero relative to the computed sum $\widehat{SUM}_n$. This means that S and $\widehat{SUM}_n$ will have the same sign and both nonzero.

Hence in all cases $n \leq \bar{n}+1$ implies that S and $\widehat{sum}$ have exactly the same sign: $\widehat{sum}$ is positive, zero, or negative exactly when S is positive, zero, or negative, respectively. In particular, we have seen that if a rounding error occurs, then S is bounded away from zero. Taking the contrapositive, if $S = 0$, then no rounding error occurs, which implies that $S = \widehat{sum} = 0$ is computed exactly.

8.8 Attainability of the error bounds

We present examples to show near attainability of error bounds in statements 3 and 4 of the theorem.

8.8.1 Case $n = \bar{n} + 1$, $F < 2f$

If $F < 2f$, then the bound in claim 3 is achieved by the following example:

$$\begin{aligned} s_1 &= 1 + 2^{f-F} \\ s_2 &= -2^{f-F}(1 - 2^{-f}) \\ s_3 &= -2^{f-F}(1 - 2^{-f}) \end{aligned}$$

$$\begin{aligned} &\vdots \\ s_{\bar{n}+1} &= -2^{f-F}(1-2^{-f}) \end{aligned}$$

The true sum is $S = 2^{-f} + 2^{-F}$. The exact value of $SUM_2 = s_1 + s_2 = 1 + 2^{-F}$ requires $F+1$ bits to represent exactly, and so it rounds down to $\widehat{SUM}_2 = 1$. All subsequent additions are exact, yielding $\widehat{SUM}_n = 2^{-f}$. This gives the relative error $2^{f-F} = 2^{2f-F} \cdot 2^{-f}$, which is at most $1/2$ (when $F = f+1$) but can be much greater than 1 ulp whenever $2f \gg F$, i.e. when F is much less than double precision. Note that this example works only if $F < 2f$, since otherwise the first number s_1 cannot be represented in f bits.

8.8.2 Case $n = \bar{n}+1$, $F \geq 2f$ and s_2 unnormalized

Following example shows that if s_2 is not normalized, then the bound in claim 2 does not apply. Instead the weaker bound in claim 3 is achieved:

$$\begin{aligned} s_1 &= 2^{e_{\text{MIN}}+F-f+1} \\ s_2 &= 2^{e_{\text{MIN}}-f+1} \\ s_3 &= -2^{e_{\text{MIN}}+1}(1-2^{-f}) \\ &\vdots \\ s_{\bar{n}+1} &= -2^{e_{\text{MIN}}+1}(1-2^{-f}) \end{aligned}$$

Note that s_2 is subnormal, so $\text{EXP}(s_2) = \text{EXP}(s_3) = \cdots = \text{EXP}(s_{\bar{n}+1}) = e_{\text{MIN}}$ (since e_{MIN} is the smallest exponent). Hence the s_i's are sorted by exponent, but not by magnitude. The true sum is $2^{e_{\text{MIN}}+1}(2^{-f} + 2^{-r})$. The exact value of $SUM_2 = s_1 + s_2 = 2^{e_{\text{MIN}}+F-f+1}(1 + 2^{-F})$ requires $F+1$ bits, so it rounds down to $2^{e_{\text{MIN}}+F-f+1}$. All subsequent additions are exact, yielding $\widehat{SUM}_n = 2^{e_{\text{MIN}}+1} \cdot 2^{-r}$. This gives a relative error of 2^{r-f}, which can be as large as 1.

8.8.3 Case $n = \bar{n}+2$

Consider the following example, corresponding to case 1B (with $I = 1$, $J = 2$, and $K = n$).

$$\begin{aligned} s_1 &= 1 \\ s_2 &= 2^{f-F-1} + 2^{-F} \\ s_3 &= -2^{f-F}(1-2^{-f}) \\ s_4 &= -2^{f-F}(1-2^{-f}) \\ &\vdots \\ s_{\bar{n}+1} &= -2^{f-F}(1-2^{-f}) \\ s_{\bar{n}+2} &= -2^{f-F-1} - 2^{f-F-r} \end{aligned}$$

Note that the exponents of s_k are sorted as long as $r \neq 1$ and all terms are f-bit numbers. Note that the exact result is $S = 2^{-F}$. However, a roundoff error occurs when s_2 is added, since $SUM_2 = 1 + 2^{f-F-1} + 2^{-F}$ needs $F+1$ bits to store. Hence $\widehat{SUM}_2$ is rounded down to

$1 + 2^{f-F-1}$. The rest of the computation proceeds without any roundoff error, and the computed sum is $\widehat{SUM}_n = 0$. Thus we have an example where relative error is infinite.

If $r = 1$, we can obtain an example where relative error is 1, by using $s_{\bar{n}+2} = -2^{f-F}(1 - 2^{-f})$ in the above example.

If $F < 2f$, we have another example (corresponding to case 1A with $I = 1$, $J = 2$, $K = n$):

$$\begin{aligned} s_1 &= 1 + 2^{f-F} \\ s_2 &= -2^{f-F}(1 - 2^{-f}) \\ s_3 &= -2^{f-F}(1 - 2^{-f}) \\ &\vdots \\ s_{\bar{n}+1} &= -2^{f-F}(1 - 2^{-f}) \\ s_{\bar{n}+2} &= -2^{-f} \end{aligned}$$

If $F < 2f$, then all the numbers are f bit long, and they are sorted by decreasing magnitude. Again, the computed result is $\widehat{sum} = 0$, while the true sum is $S = 2^{-F}$. This example shows that sorting more finely by magnitude instead of just exponent does not improve the bound on the number of terms that can be added accurately.

8.9 Bounding the error in ulps in $\widehat{sum}$

The relative error bound derived in all cases have the form

$$\frac{|\widehat{SUM}_n - S|}{|\widehat{SUM}_n|} \le \delta 2^{-f}, \tag{27}$$

where

$$\delta = \begin{cases} \frac{1}{1-2^{1-f}} & \text{for } n \le \bar{n} \\ \max\left\{2, \frac{1.5}{1-2^{1-f}}\right\} & \text{for } n = \bar{n}+1, F \ge 2f, \text{ and } s_2 \text{ normalized.} \\ \max\left\{3,\ 2^r\right\} & \text{for } n = \bar{n}+1. \end{cases}$$

Here we convert this to a bound on $|\widehat{sum} - S|$ equal to $\delta + 0.5$ ulps in $\widehat{sum}$. Bound (27) means

$$|\widehat{SUM}_n - S| \le \delta 2^{-f} |\widehat{SUM}_n| < \delta 2^{E_n - f + 1}.$$

Thus if $E_n \ge e_{\text{MIN}}$ (so that $\widehat{sum}$ does not underflow), then

$$\begin{aligned} |\widehat{sum} - S| &\le |\widehat{SUM}_n - S| + |\widehat{sum} - \widehat{SUM}_n| \\ &< \delta 2^{E_n - f + 1} + 2^{E_n - f} \\ &\le (\delta + 0.5)\ \text{ulp}(\widehat{sum}). \end{aligned}$$

Notice the last bound still holds even when $\widehat{sum}$ rounds up to the next power of 2.

If $E_n < e_{\text{MIN}}$, then rounding of $\widehat{SUM}_n$ to $\widehat{sum}$ does not cause any roundoff error (since by Fact 1 all bits of $\widehat{SUM}_n$ will be in the $f-1$ bit range $e_{\text{MIN}} - 1$ to $E_{\text{MIN}} - f + 1$). Hence the same final bound apply:

$$\begin{aligned} |\widehat{sum} - S| &\le |\widehat{SUM}_n - S| \\ &\le \delta 2^{E_n - f + 1} \\ &< (\delta + 0.5)\ \text{ulp}(\widehat{sum}). \end{aligned}$$

Hence the true sum S is at most $(\delta + 0.5)$ ulps off from computed sum $\widehat{sum}$. This completes the proof of Theorem 1.

9 Conclusions

We have described and analyzed four very simple algorithms for computing the accurate sum of n floating point numbers $S = \sum_{i=1}^{n} s_i$ with f fraction bits each. We exploit the availability of one or more $F > f$ bit accumulators to be as efficient as possible. The maximum value of n for which high accuracy is guaranteed (about 1.5 ulps) depends on f, F and (for Algorithms 2, 3 and 4) the number of exponent bits in the s_i. These maximum values of n are tabulated for the formats available in the IEEE floating point standard. Any of the four algorithms might be cheapest, depending on the relative costs of sorting, bit manipulation, and arithmetic.

Our analysis of Algorithm 1 is tight, in the sense that we know precisely how the maximum attainable error depends on n, f and F. However we do not know if our analyses of Algorithms 2, 3 or 4 are tight, in the sense that our values of n guaranteeing high accuracy are as large as possible. We also do not know if n is as large as possible when any of the algorithms are used to compute dot products. But we do not believe that n could be more than 2 times larger than we have shown.

A natural extension to Theorem 1 is to consider cases with radix other than two. We expect similar results, but exact error bounds are not known. It would also be of interest to analyze the case where rounding in the F-bit format is less accurate than in IEEE round-to-nearest. For example, Priest's analysis of doubly compensated summation [18, p. 64] yields a value of $\bar{n}$ over 8 times smaller than our analysis; what happens with other styles of multiple precision arithmetic?

Acknowledgements

We thank Doug Priest and Nick Higham for commenting on an early two-page proof of a weaker result.

References

[1] ANSI/IEEE, New York. *IEEE Standard for Binary Floating Point Arithmetic*, Std 754-1985 edition, 1985.

[2] ANSI/IEEE, New York. *IEEE Standard for Radix Independent Floating Point Arithmetic*, Std 854-1987 edition, 1987.

[3] G. Bohlender. Floating point computation of functions with maximum accuracy. *IEEE Trans. Comput.*, C-26:621–632, 1977.

[4] T. Dekker. A floating point technique for extending the available precision. *Num. Math.*, 18:224–242, 1971.

[5] N. J. Higham. The accuracy of floating point summation. *SIAM J. Sci. Comput.*, 14(4):783–799, 1993.

[6] N. J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, Philadelphia, PA, 1996.

[7] Proposed Revision of the IEEE Floating Point Arithmetic Standard 754. http://grouper.ieee.org/groups/754, 2001.

[8] Intel Corporation. Intel Itanium Architecture Software Developer's Manual, Volume 1. Intel Corporation, 2002. http://developer.intel.com/design/itanium/manuals

[9] Intel Corporation. IA-32 Intel Architecture Software Developer's Manual, Volume 1. Intel Corporation, 2002. http://developer.intel.com/design/pentium/manuals

[10] W. Kahan. Doubled-precision IEEE Standard 754 floating point arithmetic. manuscript, 1987.

[11] D. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, Reading, MA, 1969.

[12] H. Leuprecht and W. Oberaigner. Parallel algorithms for the rounding exact summation of floating point numbers. *Computing*, 28:89–104, 1982.

[13] S. Linnainmaa. Software for doubled-precision floating point computations. *ACM Trans. Math. Soft.*, 7:272–283, 1981.

[14] M. Malcolm. On accurate floating-point summation. *Comm. ACM*, 14(11):731–736, 1971.

[15] O. Møller. Quasi double precision in floating-point arithmetic. *BIT*, 5:37–50, 1965.

[16] M. Pichat. Correction d'une somme en arithmetique à virgule flottante. *Numer. Math.*, 19:400–406, 1972.

[17] D. Priest. Algorithms for arbitrary precision floating point arithmetic. In P. Kornerup and D. Matula, editors, *Proceedings of the 10th Symposium on Computer Arithmetic*, pages 132–145, Grenoble, France, June 26-28 1991. IEEE Computer Society Press.

[18] D. Priest. *On properties of floating point arithmetics: Numerical stability and the cost of accurate computations.* PhD thesis, University of California at Berkeley, 1992. ftp.icsi.berkeley.edu/pub/theory/priest-thesis.ps.Z.

[19] D. R. Ross. Reducing truncation errors using cascading accumulators. *Comm. ACM*, 8(1):32–33, 1965.

[20] J. M. Wolfe. Reducing truncation errors by programming. *Comm. ACM*, 7(6):355–356, 1964.