equisition/GraphicServices

Dept. No. 95669

5-28

Date Required	Account
ASAP please	126-6353-41

Signing Authority: M.H. van Emden — M. H. van Emden

Room	Phone	Delivery
ster Science M&C 5100B	3402	☐ Mail ☐ Via Stores ☐ Pick-up ☐ Other

1. Please complete unshaded areas on form as applicable. (4 part no carbon required).
2. Distribute copies as follows: White, Canary and Pink—Printing, Arts Library or applicable Copy Centre Goldenrod—**Retain.**
3. On completion of order, pink copy will be returned with printed material. Canary copy will be costed and returned to requisitioner, **Retain as a record of your charges.**
4. Please direct enquiries, quoting requisition number, to Printing/Graphic Services, Extension 3451.

rements
s/Repro's ☐ Xerox

Number of Pages: 11

Number of Copies: 30 (Please note remarks below.)

ook ☐ Cover ☐ Bristol ☐ Supplied

☐ 8½ x 14 ☐ 11 x 17

Ink
☐ Other ☐ Black

Numbering
☐ 2 Sides to

ations
r Stitching ☐ 3 Ring ☐ Tape ☐ Plastic Ring ☐ Perforating

Cutting
Finished Size

staples

with covers.

ithout covers; only corner stitching.

Cost: Time/Materials	Fun.	Prod.Un.	Prod. Cl.	Opr. No.	Mins.	Total
	1					
Camera	2					
Correcting & Masking Negatives	3					
Platemaking	4					
Printing	5					
Bindery	6					
Sub. Total Time						
Sub. Total Materials						
Prov. Tax						
Total						

Size

Plates Qty Size & Type

Size

Plastic Rings Qty Size

Graphic Services May 68 482-2

AN ALGORITHM FOR

BALANCED FLOATING-POINT ADDITION

by

M.H. van Emden and Helio de M. Silva
Department of Computer Science
University of Waterloo
Waterloo, Ontario

CS-78-28
June 1978

AN ALGORITHM FOR BALANCED FLOATING-POINT ADDITION

by

M.H. van Emden and Helio de M. Silva *)
University of Waterloo

ABSTRACT

The associativity of exact addition of n floating-point numbers leaves a certain amount of freedom in the choice of which partial sums to evaluate. We represent this choice as an "addition tree" and derive an error bound that is proportional to its path length, thereby generalizing a result of P. Linz. We present an algorithm which adds n floating-point numbers in such a way that the error bound is close to its theoretically attainable minimum. Apart from an additive constant, the time required by the algorithm is proportional to n; the required space is proportional to log n. We provide informal, yet rigorous, proofs of the correctness and of the claimed performance characteristics of the algorithm.

Keywords and Phrases: error bounds, floating-point addition, path length, balanced trees

CR categories: 5.11, 5.24, 5.25

---

*) on leave of absence from the Universidade Federal da Paraiba

1. Introduction

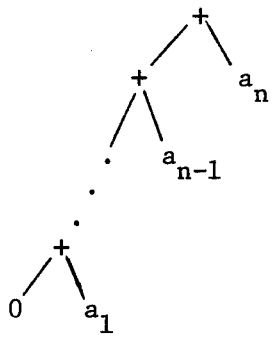
Because of rounding errors, floating-point addition is not quite an associative binary operation. Therefore it matters in which order the additions are performed which are required to obtain the sum of n numbers. The expression

$$S_n = a_1 + a_2 + \ldots + a_n$$

does not convey this information and therefore we consider instead an addition tree. For example, if S_n is obtained by

$$S_0 = 0$$
$$S_i = a_i + S_{i-1} \qquad \text{for } i = 1,\ldots,n \qquad \ldots(1)$$

then the addition tree for S_n is



We will call this "linear addition". In general, we recursively define an addition tree for a sequence of numbers

$$a_1, a_2, \ldots, a_n \qquad \text{with } n \geq 1$$

to be either a node containing a single number or to consist of a left subtree and a right subtree, both of which are addition trees. We are interested in addition trees which nearly minimize a certain bound for the rounding error. We express the bound in terms of the path length of the addition tree.

The path length p of an addition tree is the sum of the distances from the root to each of the leaves. More precisely, p can be defined recursively as follows. If the addition tree consists of a single node, then $p = 0$. Otherwise $p = n_l + n_r + p_l + p_r$, where $n_l\{n_r\}$ is the number of leaves in its left {right} subtree and $p_l\{p_r\}$ is the path length of its left {right} subtree.

A bound for the error ε in a single addition of two numbers a_i with floating-point representation $f_i \times 2^{e_i}$, where $.5 \leq f_i < 1$, and where $i = 1,2$, is

$$|\varepsilon| \leq t(2^{e_1} + 2^{e_2})$$

In this expression $t = 2^{-q}$, where q is the number of bits to which the exact sum is truncated.

When the sequence $a_1,\ldots,a_n$ is summed, a bound for the round-off error ε is given by

$$|\varepsilon| \leq 2atp \qquad \ldots(2)$$

where a is such that $|a_i| \leq a$, $i = 1,\ldots,n$, and p is the path length of the addition tree according to which the sequences is summed.

This may be proved by induction. For an addition tree consisting of a single node (2) is immediate: $p = 0$ and there is no addition and no rounding

error. For an addition tree consisting of a left subtree with sum S_1 and path length p_1 and a right subtree with sum S_r and path length p_r, the rounding error is bounded as in

$$|\varepsilon| \leq 2atp_1 + 2atp_r + t2^{e_1} + t2^{e_r}$$

where e_1 is the exponent of S_1 and e_r the exponent of S_r. Let n_1 be the number of leaves of the left subtree; then $2^{e_1} \leq 2an_1$ and similarly for the right subtree. Using this we obtain

$$|\varepsilon| \leq 2at\ (p_1 + p_r + n_1 + n_r) = 2atp$$

which completes the proof by induction of (2).

The bound in (2) suggests that one should add in such a way that for the resulting addition the path length p is minimized. It is well-known that $p \geq n \log_2 n$. The lower bound is achieved when the addition tree is a complete addition tree, which is one where each leaf has an equal distance k from the root, $k = 0,1,2,\ldots$. Such an addition tree has 2^k leaves and a path length of $k2^k$.

We call balanced addition any algorithm for which the addition tree has path length $\leq n \lceil \log_2 n \rceil$. That is, the path length, though not necessarily minimal, is nearly so. Note that the algorithm (1) has an addition tree with a path length of about $1/2\ n^2$. Relative to (1) balanced addition achieves as improvement in the error bound a factor of about $\frac{n}{2 \log_2 n}$.

Linz (1970) has described a form of balanced addition and derived for complete addition trees the bound obtained by substituting $p = n \log_2 n$ into (2). Our analysis is more general because it applies to any addition tree. Moreover, we show that in this method the bound for the rounding error is a

"path length phenomenon" and that, therefore, the improvement of balanced addition over linear addition is due to the same phenomenon as the improvement of binary search over linear search and of quicksort over insertion sort.

2. The algorithm

Linz (1970) mentioned as disadvantages of balanced addition "... it is more difficult to program than the standard method, and it is difficult to use unless all numbers are available at the start of the summation ...". But balanced addition is desirable precisely in those cases where the length of the sequence to be summed is so large that it is out of the question to have the entire sequence simultaneously available and typically the length of the sequence is not even known in advance.

In figure 1 we present a PASCAL program for a demonstrably correct linear-time algorithm for balanced addition, which does not require the length of the sequence to be summed to be known in advance. We will prove the algorithm correct by an informal version of Floyd's well-known method. First we establish some definitions.

The binary representation of a nonnegative integer n is a sequence of binary digits (value 0 or 1). For $n = 0$ the binary representation is empty (not, as conventionally, the sequence consisting of a single 0). For $n > 0$ it is $d_0,\ldots,d_k$ such that $n = \Sigma_{j=0}^{k} d_j 2^j$ and $d_k = 1$. The partial sums of n are $n_j = \Sigma_{i=0}^{j} d_i 2^i$ for $j = -1,\ldots,k$.

The part of the sequence of numbers that is already read by the program is $A = a_1,\ldots,a_n$. In order to obtain ultimately a balanced sum the program maintains balanced sums of the following subsequences of A:

$$A_j = a_{n-n_j+1},\ldots,a_{n-n_{j-1}} \qquad j = 0,\ldots,k$$

```
function bsum: real;
        var x: real; j,k: integer; d: array[0..80] of 0..1
         ;  S: array[0..80] of real;
begin d[0]:=0; k:=-1
   ;  while ~eof do
      begin {U} read(x); j:=0
        ;    while d[j]=1 do
             begin {U1} d[j]:=0; x:=x+S[j]; S[j]:=0; j:=j+1 {V1}
             end
        ;    d[j]:=1; S[j]:=x
             {V}
        ;    if j=k+1 then begin k:=j; d[k+1]:=0 end
      end
   ;  {W} x:=0
   ;  for j:=0 to k do if d[j]=1 then x:=x+S[j]
   ;  bsum:=x
      {X}
end
```

Figure 1

We claim that the assertion P, where

$$P \equiv P_1 \ \& \ P_2 \ \& \ P_3 \ \& \ P_4$$

holds whenever execution passes the checkpoints {U} or {V} in the program of figure 1, where

P_1 is: "d[0],...,d[k] is the binary representation of n, the length of the part of the sequence read so far"

P_2 is: "d[k+1] = 0"

P_3 is: "S[j] is the sum of A_j for j = 0,...,k"

P_4 is: "the addition tree for each S[j], such that d[j] = 1, is the complete addition tree of 2^j numbers"

It is clear that P holds the first time checkpoint {U} is reached. Next, assuming that P holds at {U} we have to ascertain that P holds when checkpoint {V} is reached the first time after. Let us first consider P_1, P_2, and P_3. The code between {U} and {V} reads an additional number, so that some of the A_j change. The code then makes the corresponding changes in the corresponding S[j]. These changes are determined by how the binary representation of n is updated to the one for n+1.

In order to justify that P_4 also holds when {V} is reached, it suffices to ascertain that x has a complete addition tree of 2^j numbers whenever S[j]: = x is executed. We show that x has a complete addition tree with 2^j numbers whenever execution reaches $\{U_1\}$ or $\{V_1\}$. It is obviously true the first time (since reaching {U}) execution reaches $\{U_1\}$. Let us assume it holds at $\{U_1\}$. At $\{U_1\}$ both x and S[j] have complete addition trees with 2^j numbers each. An addition tree with these as subtrees is complete and has 2^{j+1} numbers. Therefore at $\{V_1\}$ x has a complete addition tree of 2^j numbers.

At {W} we assert P in conjunction with eof. This implies that S[0] + ... + S[k] is the sum of all numbers in the file. We now prove that at {X} bsum contains the sum of all n numbers in the file and that its addition tree has path length $\leq n \lceil \log_2 n \rceil$. We distinguish two cases. In the first case, $n = 2^k$ and bsum = S[k]. The addition tree for bsum is the complete one for S[k] with path length $k2^k = n \log_2 n \leq n \lceil \log_2 n \rceil$.

In the other case $2^k < n < 2^{k+1}$; hence $\lceil \log_2 n \rceil = k+1$. Let $j_0, j_1, \ldots, j_m$ be all values of j, in increasing order, for which d[j] = 1. Note that $j_m = k$. Between {W} and {X} a sequence of values of x is computed according to

$$x_0 = S[j_0]$$
$$x_i = x_{i-1} + S[j_{i-1}] \qquad i = 1,\ldots,m$$

We prove by induction on i that the addition tree for x_i has path length $\leq (j_i+1)n_{j_i}$. For $i = 0$ the path length is the one of the complete addition tree for $S[j_0]$, which is $j_0 2^{j_0} = j_0 n_{j_0}$.

The addition tree for x_{i+1} has as left subtree the addition tree for x_i. It has n_{j_i} numbers in it and, according to the induction hypothesis, its path length $\leq (j_i+1)n_{j_i}$. The right subtree of the addition tree for x_{i+1} is the addition tree for $\mathbf{S}_{j_{i+1}}$ which is complete, has $2^{j_{i+1}}$ numbers in it and has path length $j_{i+1}2^{j_{i+1}}$. We therefore have for the path length p_{i+1} of the addition tree for x_{i+1}:

$$p_{i+1} \leq n_{j_i} + (j_i+1)n_{j_i} + 2^{j_{i+1}} + j_{i+1}2^{j_{i+1}}$$
$$p_{i+1} \leq n_{j_i} + (j_i+1)n_{j_i} + (j_{i+1}+1)2^{j_{i+1}}$$
$$p_{i+1} \leq (j_{i+1}+1)(2^{j_{i+1}} + n_{j_i}) = (j_{i+1}+1)n_{j_{i+1}}$$

which was to be proved.

The path length of the addition tree for x_m (which becomes the value of bsum) is $\leq (j_m+1)n_{j_m} = (k+1)n_k = (k+1)n = n\lceil\log_2 n\rceil$.

3. Analysis of the efficiency of the algorithm

The algorithm has two nested loops. The outer loop is executed n times, where n is the length of the sequence summed. At the beginning of each activation of the inner loop, d holds successively the binary representations of $0, 1,\ldots, n-1$. Let $f(i)$ be the largest integer such that

$d[0] = d[1] = \ldots = d[f(i)] = 1$ when d holds the binary representation of i. Therefore $d[f(i)+1] = 0$, unless the binary representation of i contains only ones. In the i-th execution of the outer loop, the inner loop is executed $f(i)$ times. The total number of times the inner loop is executed is $\Sigma_{i=0}^{n-1} f(i)$.

It is easy to verify that $\Sigma_{i=1}^{n} f(i) \leq n$ with equality iff n is one less than a power of 2. That is, the number of times the inner loop is executed, averaged over executions of the outer loop, is at most 1. Therefore our algorithm requires for the balanced addition of n floating-point numbers an execution time proportional to n.

4. Concluding Remarks

Linz (1970) has obtained experimental results comparing linear with balanced addition of $n = 2048$ numbers randomly distributed between 0 and 1. The improvement in the error bound of balanced addition over linear addition is about $n/(2 \log_2 n)$ which is about 100 for this value of n. Linz observed an improvement in actual error by a factor of about 140. For $n = 2^{16}$ which is perfectly feasible with our algorithm, the factor $n/(2 \log_2 n)$ is already about 2000.

It is out of the question, because (as far as we know) computationally prohibitive, to obtain always an addition tree with minimum path length, which can, anyway, never be less than $n \log_2 n$. It is therefore quite satisfying to be able to add in linear time in such a way that the addition tree has a path length bounded by $n\lceil \log_2 n \rceil$ which is not far from the minimum for large n.

5. References

P. Linz (1970): Accurate floating-point summation.
Comm. ACM 13, pp. 361-362.

6. Acknowledgements

We gratefully acknowledge partial support from the Canadian National Research Council.