Fast, guaranteed-accurate sums of many floating-point numbers

Yong-Kang Zhu and Wayne B. Hayes
Department of Computer Science
University of California, Irvine
{yongkanz,wayne}@ics.uci.edu

Outline

1. Introduction
2. Related work
3. Algorithms
4. Results
5. Conclusions

Introduction

Example: (using IEEE 754 standard for binary floating-point arithmetic)

$$a = 1000.13$$

$$b = 0.23$$

$$n = 2^{10} = 1024$$

$$s = a + n \cdot b$$

Program 1

```
double s1=a;
for ( int i=1; i<=n; i++ )
          s1+=b;
```

Program 2

```
double s2=a;
s2+=n*b;
```

s1 = 1.2356500000000185 E 3
s2 = 1.2356500000000001 E 3

Mantissa of s1 = 1.4E99999999EB
Mantissa of s2 = 1.4E999999999A

Related work

- Recursive summation
  - ORS (Ordinary Recursive summation)
  - Recursive summation with orderings (increasing, decreasing, PSum)
  - Two other methods: Pairwise, Insertion
- Compensated summation and its variation
- Using high-precision accumulators
  - Demmel and Hida 2003
- Distillation algorithms
  - Anderson 1999
  - Ogita, et al., 2005
  - Zhu, et al., 2005
  - Rump, et al., 2006

Related work - A comparison

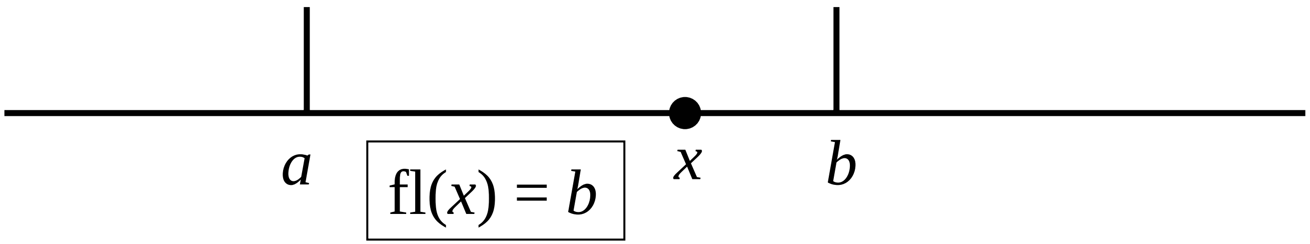
- *No one method is uniformly more accurate than the other.*
  (refer to recursive and compensated summations) – Higham 1993
- No high-precision accumulators in typical computers.
- Distillation algorithms achieve a higher accuracy:

methods		speed	accuracy
Anderson	Modified Deflation	slow	observed errors
Ogita, et al.	SumK (k = 3)	fast	n, R
Rump, et al.	AccSum	fast	n
Zhu, et al.	Zhu05	medium	guaranteed < 1ulp

R is the condition number: $\sum_{i=1}^{n} |x_i| \;/\; \left|\sum_{i=1}^{n} x_i\right|$

Algorithms

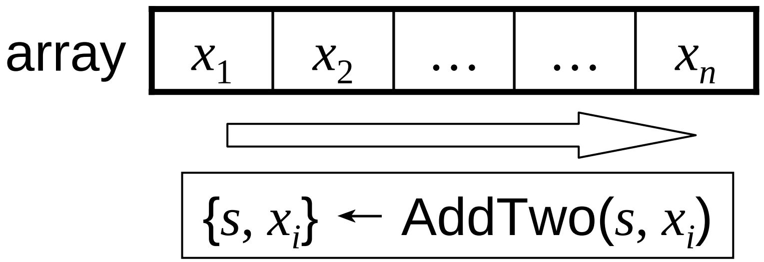
- Error-free addition: $\{s, e\} \leftarrow \mathrm{AddTwo}(a, b)$
  - $s + e = a + b$
  - $s = \mathrm{fl}(a + b)$
  - fl() : standard floating-point operation
- Assume fl() is *correctly rounded* (round-to-nearest)



Note: *faithfully rounded* $\begin{cases} \mathrm{fl}(x) = \text{either } a \text{ or } b, \text{ if } x \neq a \text{ and } x \neq b \\ \mathrm{fl}(x) = a, \text{ if } x = a \\ \mathrm{fl}(x) = b, \text{ if } x = b \end{cases}$

Algorithms (Cont'd)

- FastSum is based on AddTwo

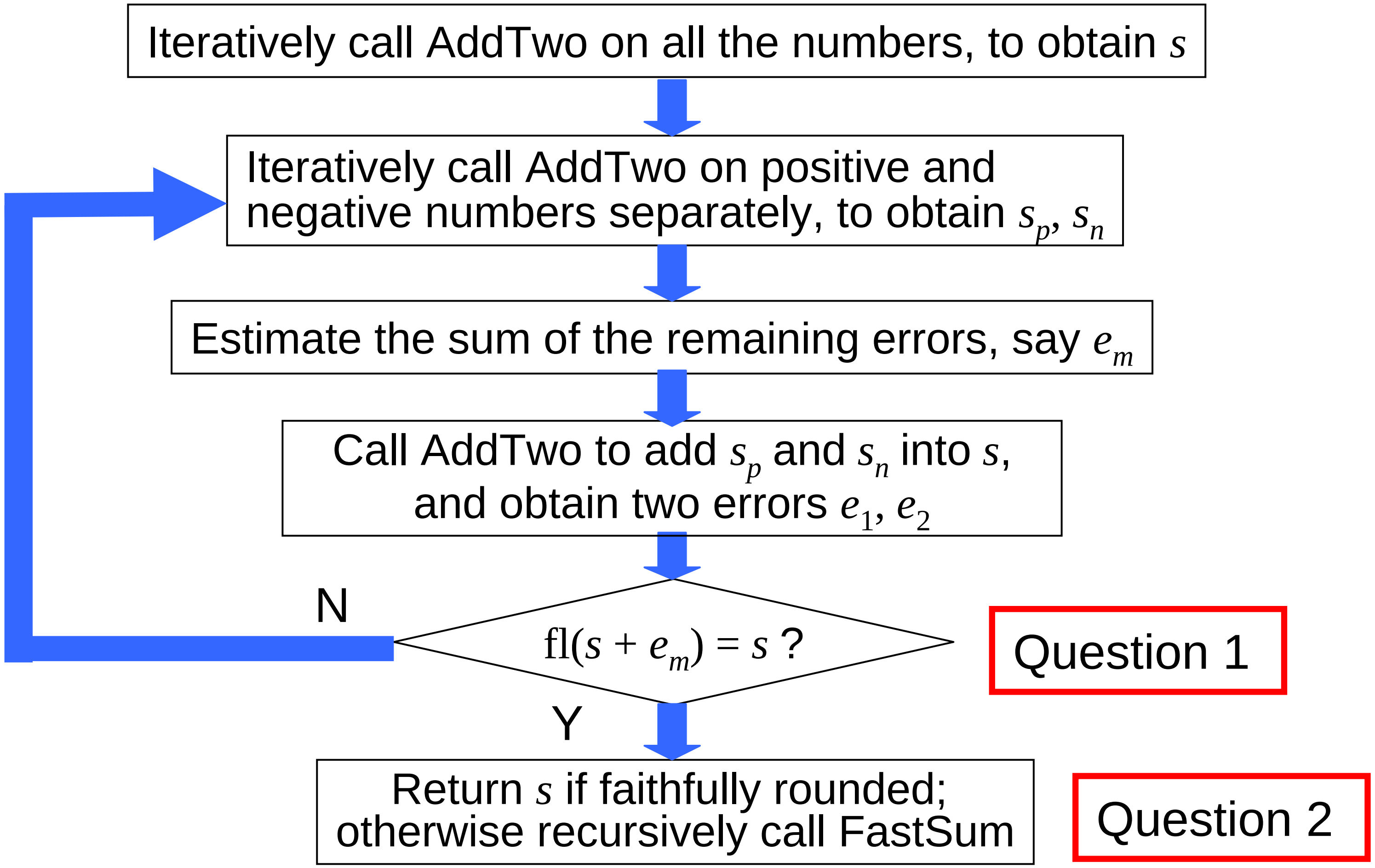


iteratively call AddTwo

  - $s = \mathrm{fl}(x_1+\mathrm{fl}(x_2+(\ldots\mathrm{fl}(x_{n\text{-}1}+x_n)\ldots)))$
  - The errors are redistributed into the original array
  - No significant digits are discarded

Algorithms (Cont'd)

- The basic idea of FastSum



Algorithms (Cont'd)

Question 1: can fl(s+e_m)=s fail to be satisfied?

- Claim: iteratively calling AddTwo will, in finite loops, converge to the following stable state:
  - The array is sorted by increasing magnitude
  - the mantissas of adjacent elements are non-overlapping
  - thus AddTwo does nothing
  - and the sum is constant

Algorithms (Cont'd)

- e_m is calculated by:

$$e_m \leftarrow count \cdot \mathrm{ulp}(\max(|s_p|, |s_n|))$$

number of non-zero errors

ulp (unit in last place)

- If such a stable state arrives, then
  - *count* equals the number of non-overlapping floating-point numbers
  - The maximum of non-overlapping floating-point numbers is limited by the arithmetic

Algorithms (Cont'd)

Compute this maximum:

- $\mathrm{fl}(s+s_p)=s$, $\mathrm{fl}(s+s_n)=s$ $\Longrightarrow$ $\max(|s_p|,|s_n|)<\mathrm{ulp}(s)$
- $\mathrm{fl}(s+e_m)\neq s$ $\Longrightarrow$ $e_m>\mathrm{ulp}(s)$
- Recall: $e_m \leftarrow count \cdot \mathrm{ulp}(\max(|s_p|,|s_n|))$
- Thus, $count \cdot \mathrm{ulp}(\mathrm{ulp}(s))>\mathrm{ulp}(s)$ $\Longrightarrow$ $count > \beta^t$
- But in IEEE754 double,

  exponent = 11 bits, mantissa = 53 bits

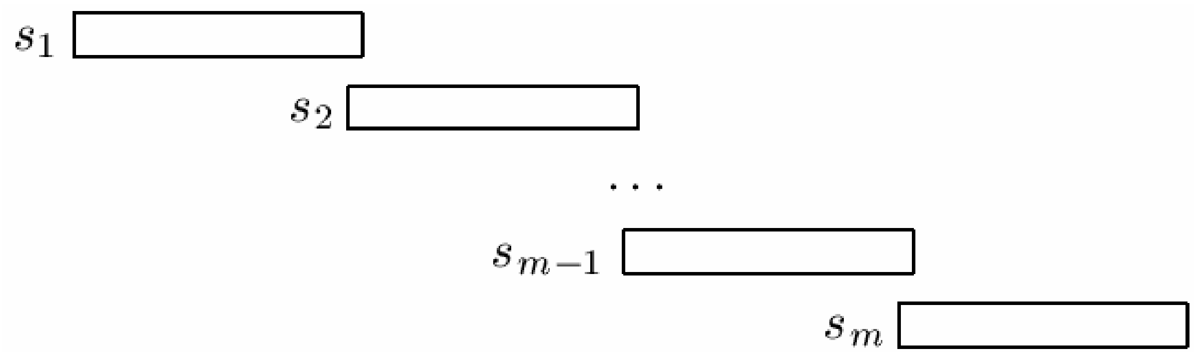
  so, $count < 2^{11}/53 \approx 38 \ll \beta^t = 2^{53}$

$\boxed{\textbf{Answer 1}\text{: } \mathrm{fl}(s+e_m)=s \text{ can be satisfied in finite loops}}$

Algorithms (Cont'd)

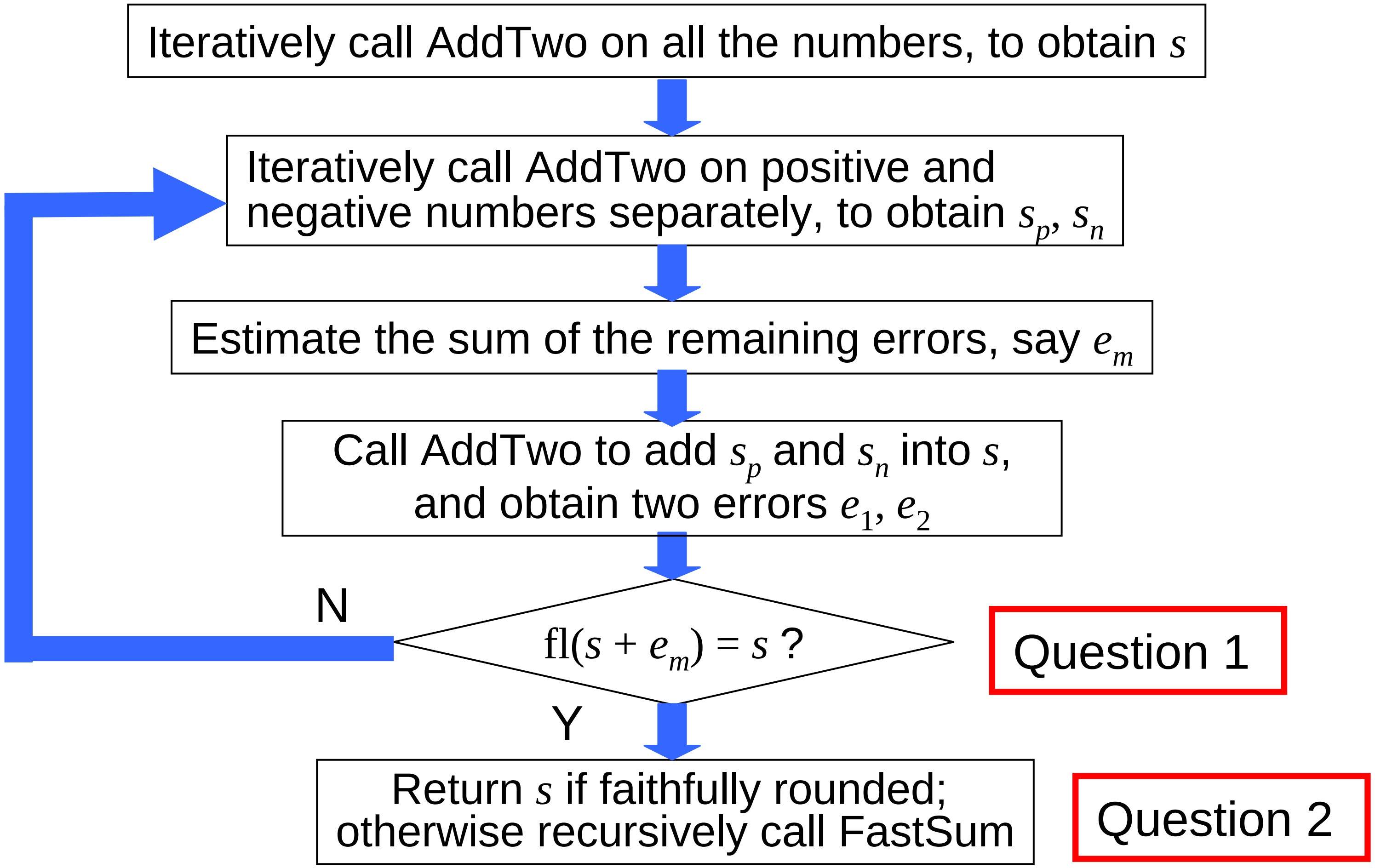
Question 2: Is s faithfully rounded when recursively calling FastSum?

- Recall e_1 and e_2
- It is possible that $\mathrm{fl}(s+e_m)=s$, but $\mathrm{fl}(s\pm e_m+e_1+e_2)\neq s$
- In this case, name the current s s_1, and recursively call FastSum on the remaining numbers to obtain s_2, etc.
- Note that s_i and s_{i+1} overlap by at most 1 digit
- $s = s_1 + s_2 + \ldots + s_m$, m is finite, and s_m is faithfully rounded



Algorithms (Cont'd)

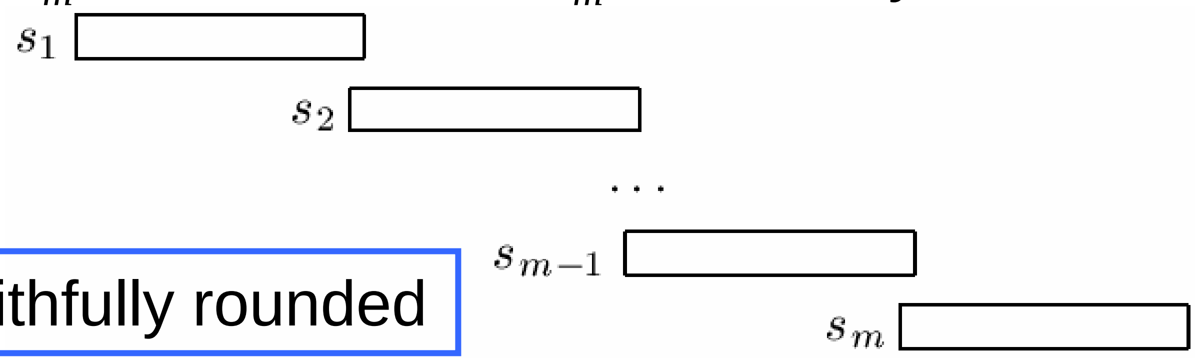
- The basic idea of FastSum



Algorithms (Cont'd)

Question 2: Is s faithfully rounded when recursively calling FastSum?

- Recall e_1 and e_2
- It is possible that $\mathrm{fl}(s+e_m)=s$, but $\mathrm{fl}(s \pm e_m+e_1+e_2) \neq s$
- In this case, name the current s s_1, and recursively call FastSum on the remaining numbers to obtain s_2, etc.
- Note that s_i and s_{i+1} overlap by at most 1 digit
- $s = s_1 + s_2 + \ldots + s_m$, m is finite, and s_m is faithfully rounded



Answer 2: s is faithfully rounded

Results

- Running time for 4 algorithms

  Three ways to generate the original data

  - Data No.1
    - well-conditioned
    - condition number $R = 1$, i.e., all positive or negative
  - Data No.2
    - ill-conditioned
    - subtract the mean from each summand
  - Data No.3
    - ill-conditioned data
    - condition number $R = +\infty$
    - generate pairs of equal numbers with opposite signs (the final sum is exactly zero)

Results (cont'd)

- Running time for 4 algorithms

Data No.1	2	4	6	8	10 ($\times 10^6$)
Sum3	3.9	4.5	6.8	6.0	5.5
Zhu05	11.7	11.6	17.1	16.0	14.9
AccSum	4.0	3.5	4.9	4.7	4.3
FastSum	4.0	4.5	6.3	5.9	5.5
Data No.2	**2**	**4**	**6**	**8**	**10 ($\times 10^6$)**
Sum3	5.2	4.0	6.3	5.7	5.6
Zhu05	15.7	14.1	21.9	18.9	16.6
AccSum	6.3	7.0	9.8	8.7	9.5
FastSum	4.1	4.5	6.8	5.9	5.8
Data No.3	**2**	**4**	**6**	**8**	**10 ($\times 10^6$)**
Sum3	4.2	8.8	6.6	6.0	7.2
Zhu05	15.6	29.3	22.7	20.3	24.7
AccSum	6.3	13.7	10.5	9.0	11.4
FastSum	5.2	10.7	8.1	7.3	8.8

Results (cont'd)

- Why choosing Dekker's algorithm for AddTwo

	additions	branches
Dekker's	3	1
Knuth's	6	0

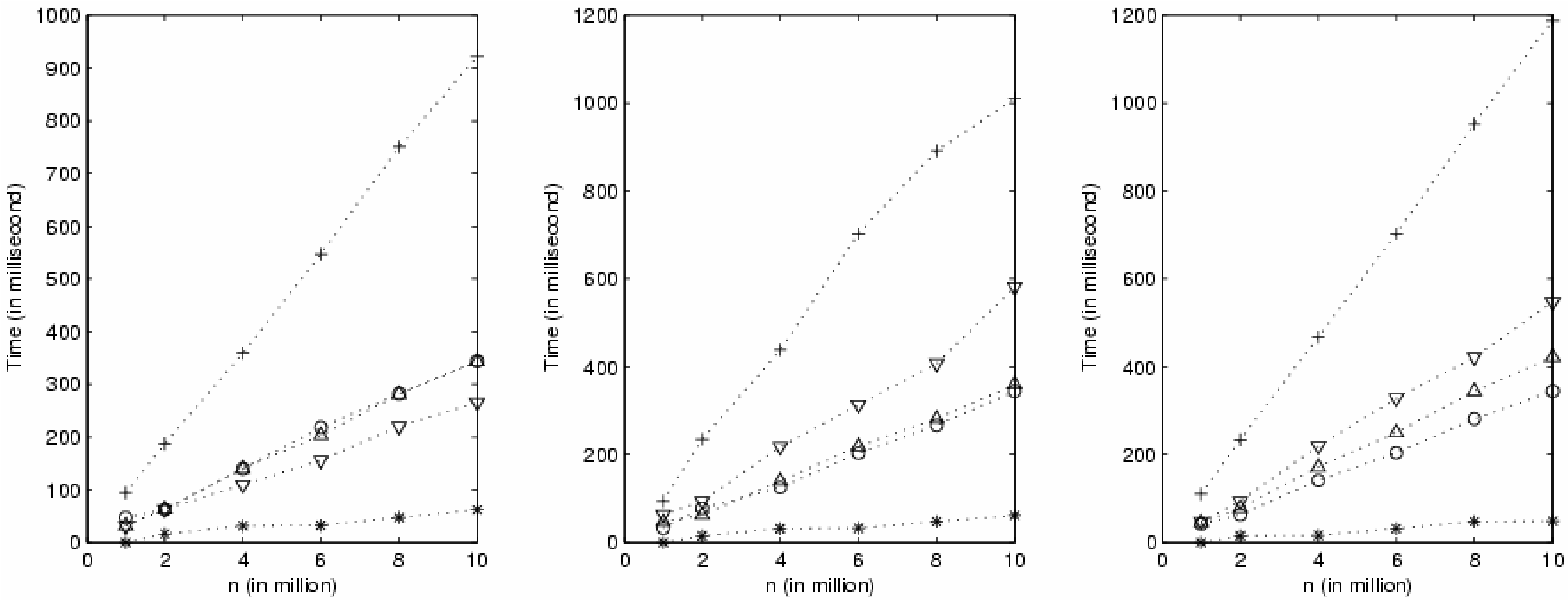
  – Will branches slow down the speed significantly?
- Numerical test

	with function calls	without function calls
Using Dekker's algorithm	437	375
Using Knuth's algorithm	469	391

Results (cont'd)

- Running time of FastSum is linear with *n*

n	times	running time
10,000,000	1	410 ms
1,000	10,000	≈410 ms



(a) (b) (c)

Results (cont'd)

- In our test, for Data No. 3: R = +∞, exact sum = 0
  - Zhu05, AccSum, FastSum can always generate correct results, if no overflow occurs
  - Sum3 fails when $\triangle E > 90$
  - $\triangle E > 2000$, AccSum produces an overflow
  - FastSum requires at most 48 loops when $\triangle E$ is big, although it is not realistic that $\triangle E > t$

Conclusions

- FastSum is as fast as the existing algorithms
- FastSum can guarantee the accuracy,
  independent of both n and the condition number R
- For floating-point arithmetic other than IEEE754, FastSum works as long as an effective AddTwo exists
- The running time is linear with n, if generating the original data with the same attribute
- In our environment, branches are not important