Accuracy Versus Time

A Case Study with Summation Algorithms

Laurent Thévenoux
with Phillipe Langlois and Matthieu Martel

Équipe de Recherche DALI
Université de Perpignan Via Domitia
52 Avenue Paul Alduy
66860 Perpignan Cedex

5th EVA-Flo Project Meeting

Canet, France, May 20-21, 2010

Introduction
Context and Objectives

Context

- Transformation tool to improve automatically the quality of computations: accuracy and time
- Validation of software

Goal

To improve the accuracy of floating-point computations

Working on Summation Algorithms: Motivation

- Simple but significant problems in our application scope
- Lot of research on this subject
- Many algorithms to improve accuracy

Introduction
Context and Objectives

- Accuracy and time do not cohabit *well*[Demmel]
- How can we improve the accuracy of numerical algorithms if we relax slightly the performance constraints?

Example

For example, let us consider the sum:

$$s = \sum_{i=1}^{N} a_i, \text{ with } a_i = \frac{1}{2^i}, \; 1 \leq i \leq N$$

Introduction
Context and Objectives

- Accuracy and time do not cohabit *well*[Demmel]
- How can we improve the accuracy of numerical algorithms if we relax slightly the performance constraints?

Example

Two extreme algorithms compute s

$$s_1 := \Big(\big((a_1 + a_2) + a_3\big) + \ldots a_{N-1} \Big) + a_N$$

s_1 is computed in linear time

Introduction
Context and Objectives

- Accuracy and time do not cohabit *well*[Demmel]
- How can we improve the accuracy of numerical algorithms if we relax slightly the performance constraints?

Example

and, assuming $N = 2^k$,

$$s_2 := \Big(\big((a_1 + a_2) + (a_3 + a_4) \big) + \ldots + (a_{\frac{N}{2}-1} + a_{\frac{N}{2}}) \Big) +$$

$$\Big(\big((a_{\frac{N}{2}+1} + a_{\frac{N}{2}+2}) + (a_{\frac{N}{2}+3} + a_{\frac{N}{2}+4}) \big) + \ldots + (a_{N-1} + a_N) \Big)$$

s_2 is computed in logarithmic time

Introduction
Context and Objectives

- Accuracy and time do not cohabit *well*[Demmel]
- How can we improve the accuracy of numerical algorithms if we relax slightly the performance constraints?

Example

However, in double precision, we have, for $N = 10$:

$$s = 0.9990234375 \quad s_1 = 0.9990234375 \quad s_2 = 0.99609375$$

and it happens that s_1 is more precise than s_2

Outline of the talk: our approach

- Background material
- Performing an exhaustive study: generating all the equivalent expressions of an expression using associativity and commutativity
- Computing the worst errors which may arise during their evaluation (for different datasets, using intervals)
- Comparing errors and parallelism level to find the best ratio between time and accuracy
- Extend results to bounded parallelism and compensated algorithms

Our Main Conclusion

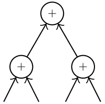
Our main conclusion is that relaxing very slightly the time constraints by choosing algorithms whose critical path are a bit longer than the optimal makes it possible to strongly optimize the accuracy.

Outline of the talk

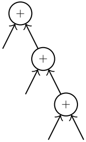
- Background material
- Performing an exhaustive study: generating all the equivalent expressions of an expression using associativity and commutativity
- Computing the worst errors which may arise during their evaluation (for different datasets, using intervals)
- Comparing errors and parallelism level to find the best ratio between time and accuracy
- Extend results to bounded parallelism and compensated algorithms

Summation Algorithms

Two Extreme Algorithms for Parallelism



$O(\log(n))$



$O(n)$

- Algorithm in $O(n)$ is the extreme sequential algorithm. It computes a sum in n operations successively summing the $n+1$ FP numbers
- Pairwise summation algorithm is the most parallel algorithm. It computes a sum in $O(\log(n))$ successive stages

Merging Parallelism and Accuracy

Mixing these algorithms gives many algorithms of parallelism levels between those two extreme ones

Summation Algorithms (2)

Improve Accuracy of One Computed Sum

Extreme Parallel Algorithms do not have the same worst case error bound

To improve accuracy of one computed sum:

- sort the terms according to their characteristics (increasingly, decreasingly, negative, positive sort, etc.)
- inserting methods
- use compensated or EFT (Error-Free Transformation) algorithms
- etc.

Summation Algorithms (3)
More Accuracy with Compensation

- Compensation is a well known and efficient technique to improve accuracy
- It uses EFT (Error-Free Transformation)
- Many kinds of compensation algorithm:
  - VecSum[Rump]: Error-Free Vector transformation of $n+1$ FP Numbers
  - Sum2, SumComp: Compensated Summation of $n+1$ FP Numbers
    - Sum2 applies compensation at the last summation
    - SumComp[Kahan] applies compensation to the next summand before adding it to the previous partial sum

Measuring the Error Terms(1)

Let x be a real number, in FP arithmetic, x is approximated by $\hat{x}$, such that $x = \hat{x} + \epsilon_x$, $\epsilon_x \in \mathbb{R}$

Let us consider the sum $S = x + y$ approximated by $\hat{S} = \hat{x} \oplus \hat{y}$ (where $\oplus$ is a floating-point addition)

We write the difference ϵ_S between S and $\hat{S}$

$$\epsilon_S = S - \hat{S} = \epsilon_x + \epsilon_y + \epsilon_+$$

where ϵ_+ denotes the round-off error introduced by the operation $\hat{x} \oplus \hat{y}$ itself

Measuring the Error Terms(2)

We use interval $\mathbf{x}$, $\mathbf{y}, \ldots$ instead of FP numbers $\hat{x}$, $\hat{y}, \ldots$

- To improve accuracy of any dataset or, at least, of a wide range of datasets
- Necessarily to represent real numbers (as error terms) using rounding modes towards outside
- Finally used in compiler tools[*Fluctuat*]

An interval $\mathbf{x}$ with related interval error $\epsilon_{\mathbf{x}}$ denotes all the floating-point numbers $\hat{x} \in \mathbf{x}$ with a related error $\epsilon_x \in \epsilon_{\mathbf{x}}$. This means that the pair $(\mathbf{x}, \epsilon_{\mathbf{x}})$ represents the set of exact results

Measuring the Error Terms(3)

Let $\mathbf{x}$ and $\mathbf{y}$ be two sets of floating-point numbers with error terms $\epsilon_{\mathbf{x}} \subseteq \mathbb{R}$ and $\epsilon_{\mathbf{y}} \subseteq \mathbb{R}$. We have

$$\mathbf{S} = \mathbf{x} \oplus_I \mathbf{y} \; ; \epsilon_{\mathbf{S}} = \epsilon_{\mathbf{x}} \oplus_O \epsilon_{\mathbf{y}} \oplus_O \epsilon_+$$

where $\oplus_I$ is the sum round to the nearest, $\oplus_O$ is the sum round towards outside and ϵ_+ is the round-off error of $\hat{x} \oplus_I \hat{y}$

Measuring ϵ_+

Let $ulp(x)$ a function which computes the ulp of x and let $S = [\underline{S}, \overline{S}]$ We bound ϵ_+ by the interval $[-u, u]$ with

$$u = \frac{1}{2} \max(ulp(|\underline{S}|), ulp(|\overline{S}|))$$

Outline of the talk

- Background material
- Performing an exhaustive study: generating all the equivalent expressions of an expression using associativity and commutativity
- Computing the worst errors which may arise during their evaluation (for different datasets, using intervals)
- Comparing errors and parallelism level to find the best ratio between time and accuracy
- Extend results to bounded parallelism and compensated algorithms

Generation of Expressions

Generalities

How our tool generates all the re-parsings of an expression

- In case of summation, the combinatorial is huge, this was often studied but no general solution exists
- Our tool finds all the equivalent expressions of an expression but generates only the different equivalent expressions : $a + (b + c) == a + (c + b)$

Terms	All expressions	Different expressions
5	1680	120
10	$1.76432e^{+10}$	$4.66074e^{+07}$
15	$3.4973e^{+18}$	$3.16028e^{+14}$
20	$4.29958e^{+27}$	$1.37333e^{+22}$

Table: Number of terms and expressions.

Generation of Expressions
Structures

1 algorithm (expression) is represented by 1 binary tree: nodes are sums and leaves are values

Recursively:

- An expression is composed of one term at least: $n \geq 1$
- A leaf x has only one representation, it is a tree of one term represented like this: 

  x

  Then the number of structures for one term trivially reduces to one

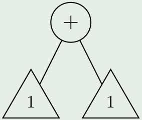
Generation of Expressions
Structures

1 algorithm (expression) is represented by 1 binary tree: nodes are sums and leaves are values

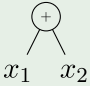
Recursively:

- Expression $x_1 + x_2$ is a tree of two terms 

It has the following structural representation:



With two terms we can create only one tree:



Generation of Expressions
Structures

1 algorithm (expression) is represented by 1 binary tree:
nodes are sums and leaves are values

Recursively:

- Recursively, we apply the same rules
- For a tree of n terms, we generate all the different structural trees for all the possible combinations of sub-trees, i.e. for all $i \in [1, n-1]$, two sub-trees with, respectively, i and $(n-i)$ terms
- Because summation is commutative, it is sufficient to generate these $(i; n-i)$-sub-trees for all $i \in [1, \lfloor \frac{n}{2} \rfloor]$

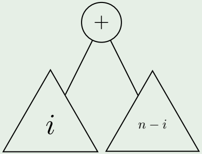
Generation of Expressions
Structures

1 algorithm (expression) is represented by 1 binary tree:
nodes are sums and leaves are values

Recursively:

This is represented as it follows:

$\forall i \in [1, \lfloor \frac{n}{2} \rfloor],$



Generation of Expressions

Structures

1 algorithm (expression) is represented by 1 binary tree: nodes are sums and leaves are values

Recursively:

- So, for n terms, we generate the following numbers of structurally different trees,

$$S_{truct}(1) = S_{truct}(2) = 1,$$

$$S_{truct}(n) = \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} S_{truct}(n-i) \cdot S_{truct}(i)$$

Generation of Expressions
Permutations

- To generate only different permutations, the leaves are related to the tree structure
- There is a restriction on permutation, for example, we do not wish to have the following two permutations: $a + (d + (b + c))$ and $a + ((c + b) + d)$
- In order to generate all the permutations, we use a similar method as described for the generation of structures

Generation of Expressions
Permutations

- Firstly, we know that for an expression of one term, we generate only one permutation. $P_{erm}(1) = 1$
- Using our permutation restriction, it is sufficient to generate one permutation for an expression of two terms; so again $P_{erm}(2) = 1$
- Permutations is related to the tree structure and we count it with the following recursive relation:

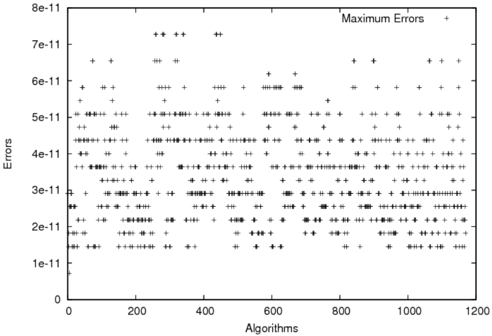
$$P_{erm}(1) = P_{erm}(2) = 1$$

$$P_{erm}(n) = \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} C_n^i \cdot P_{erm}(n-i) \cdot P_{erm}(i)$$

Outline of the talk

- Background material
- Performing an exhaustive study: generating all the equivalent expressions of an expression using associativity and commutativity
- Computing the worst errors which may arise during their evaluation (for different datasets, using intervals)
- Comparing errors and parallelism level to find the best ratio between time and accuracy
- Extend results to bounded parallelism and compensated algorithms

Numerical Accuracy
of Non Time-Optimal-Algorithms



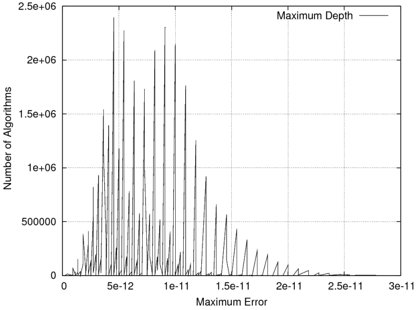
Maximum errors among each algorithms for a sumation of six terms.

- uniformly distributed
- belong to a small number of stages

Numerical Accuracy
of Non Time-Optimal-Algorithms

Error repartition when summing ten terms.

- very few small of very large errors is small
- most of the algorithms present an average accuracy between small and large errors
- find the best accurate (as well as the worst one) algorithm is difficult



Numerical Accuracy
of Non Time-Optimal-Algorithms

Using different levels of parallelism

We observe that the most parallel one does not allow us to compute the most accurate results

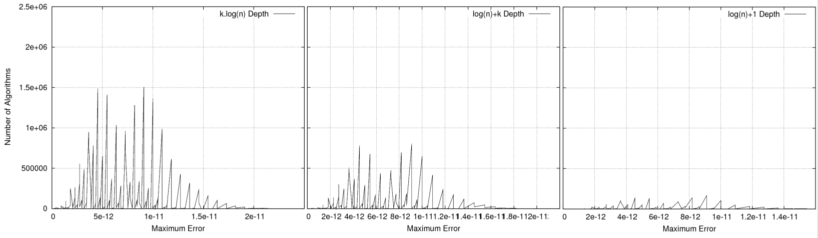
Parallelism	Best Error	Percent
no parallelism	$2.273e^{-13}$	0.006
$\lfloor \log(n) \rfloor + 1$	$4.547e^{-13}$	0.007
$\lfloor \log(n) \rfloor + k$	$2.273e^{-13}$	0.006
$k \times \lfloor \log(n) \rfloor$	$2.273e^{-13}$	0.007

Table: Error value and average on level parallelism.

Numerical Accuracy
of Non Time-Optimal-Algorithms

The more the level of parallelism is, the harder it is to find the most accurate algorithms among all of them

Error repartition with three different degrees of parallelism.



Numerical Accuracy (2)
Larger Experiments

- To study a more representative sets of data
- Using various kinds of values chosen as well-known error-prone problems, i.e. ill conditioned sets of summands
  - condition number for computing $s = \sum_{i=1}^{N} x_i$, is

$$cond(s) = \frac{\sum_{i=1}^{N} |(x_i)|}{|s|}$$

  - suffering of absorption and cancellation
- Using 9 different datasets to generate different type of absorptions and cancellations
- Using interval data, more precisely, small variation around scalar values

Numerical Accuracy (3)

Larger Experiments - Datasets

- D1. Positive sign, 20% of LV among SV
- D2. Negative sign, 20% of LV among SV
- D3. Positive sign, 20% of LV among SV and MV
- D4. Negative sign, 20% of LV among SV and MV
- D5. Both signs, 20% of ill-conditioned LV among SV
- D6. Both signs, few SM, MV and ill-conditioned LV
- D7. Both signs, few SM, ill-conditioned LV and MV
- D8. Both signs, few SM, LV and ill-conditioned MV
- D9. Data representative of values sent by a sensor

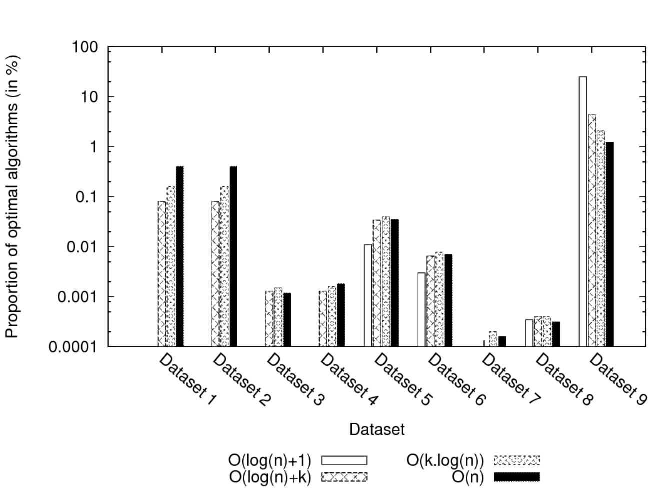
SV = Small value = 10^{-16}, LV = large value = 10^{16} and MV = medium value = 1

(justified in double precision IEEE-754 arithmetic)

Numerical Accuracy (4)

Proportion of the Optimal Algorithms

Proportion of optimal algorithms (average on 10 datasets).



Outline of the talk

- Background material
- Performing an exhaustive study: generating all the equivalent expressions of an expression using associativity and commutativity
- Computing the worst errors which may arise during their evaluation (for different datasets, using intervals)
- Comparing errors and parallelism level to find the best ratio between time and accuracy
- Extend results to bounded parallelism and compensated algorithms

Further Examples
Compensated Summation

To improve the accuracy of expression E, we compute an expression E_{cmp} which could be generated by a compiler

To illustrate this, we present an example with a summation of five terms $(((a+b)+c)+d)+e$:

$$
\begin{gathered}
a = -9.5212224350e^{-18} \\
b = -2.4091577979e^{-17} \\
c = 3.6620086288e^{+03} \\
d = -4.9241247828e^{+16} \\
e = 1.4245601293e^{+04}
\end{gathered}
$$

Further Examples

Compensated Summation

The maximal accuracy which can be obtained is given by the algorithm $(((a+b)+c)+e)+d$. It generates the absolute error $\Delta = 4.00000000000020472513$. We observe that this algorithm is Algorithm Sum with increase order

The maximal accuracy given by the maximal level of parallelism is obtained by the algorithm $((a+c)+(b+e))+d$. In this case, the absolute error is

$$\delta_{nocomp} = 4.000000000000029558578$$

Further Examples
Compensated Summation

When applying compensation on this algorithm, we obtain the following more accurate algorithm:

$$(f + (g + (h + i))) + (d + ((b + e) + (a + c))),$$

with:

$$\begin{gathered} f = C(a, c) = -9.5212224350000e^{-18} \\ g = C(b, e) = -2.4091577978999e^{-17} \\ h = C(f, g) = -1.8189894035458e^{-12} \\ i = C(h, d) = 3.6099218000017 \end{gathered}$$

It appears that this algorithm found with the application of compensation is actually the Sum2 algorithm

Further Examples
Compensated Summation

Now we measure the improved absolute error

$$\delta_{comp} = 4.00000000000000008881$$

$$\delta_{nocomp} = 4.0000000000029558578$$

$$\Delta = 4.0000000000020472513$$

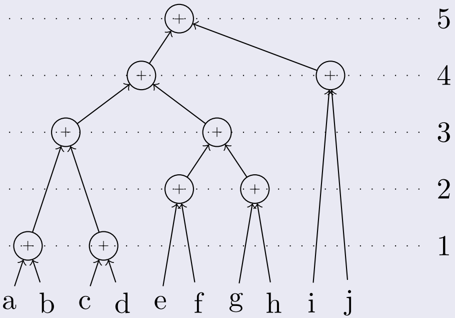
These results illustrates that we can automatically find algorithms existing in the bibliography and that the transformation improves the accuracy

Further Examples

Bounded Parallelism

In processor architectures parallelism is bounded, so it is possible to execute an algorithm less parallel in the same execution time as the fastest one

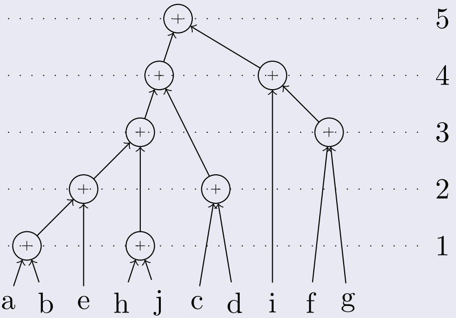
$\lfloor \log(n) \rfloor + 1$ algorithm does not provide the maximum accuracy



Further Examples
Bounded Parallelism

In processor architectures parallelism is bounded, so it is possible to execute an algorithm less parallel in the same execution time as the fastest one

Algorithm in $\lfloor \log(n) \rfloor + k$ provide the maximum accuracy



Conclusion

- First steps towards the development of a tool that aims at automatically improving the accuracy of numerical expressions in floating-point arithmetic
- Algorithms described in bibliography can be automatically generated
- Trade-Off between time and accuracy is reasonable in practice
- *Relaxing very slightly the time constraints by choosing algorithms whose critical paths are a bit longer than the optimal makes it possible to strongly optimize the accuracy*

Perspectives

- Increase the complexity of the case study : including more and different operations
- Solve the problem of the combinatorics of possible transformation
- How to develop significant datasets corresponding to any data interval provided by the user of the expression to transform
- Certified an accurate transformation with a certification tool (static analysis, abstract interpretation)