Accurate Floating Point Product

Stef Graillat

LIP6/PEQUAN - Université Pierre et Marie Curie (Paris 6)

REC'08, Third International Worskhop on Reliable Engineering Computing
Savannah, Georgia, USA, February 20-22, 2008

Motivations

- Determinant of a triangle matrix

$$T = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1n} \\ & t_{22} & & t_{2n} \\ & & \ddots & \vdots \\ & & & t_{nn} \end{bmatrix}.$$

$$\det(T) = \prod_{i=1}^{n} t_{ii}.$$

- Evaluation of a polynomial when represented by the root product form $p(x) = a_n \prod_{i=1}^{n}(x - x_i)$

What are Error-Free Transformations (EFT) ?

Assume floating point arithmetic adhering IEEE 754 with rounding to nearest with rounding unit $\mathbf{u}$ (no underflow nor overflow)

Error free transformations are properties and algorithms to compute the generated elementary rounding errors,

$$a, b \text{ entries } \in \mathbb{F}, \quad a \circ b = \mathrm{fl}(a \circ b) + e, \text{ with } e \in \mathbb{F}$$

Key tools for accurate computation

- fixed length expansions libraries : double-double (Briggs, Bailey, Hida, Li), quad-double (Bailey, Hida, Li)
- arbitrary length expansions libraries : Priest, Shewchuk
- compensated algorithms (Kahan, Priest, Ogita-Rump-Oishi, Graillat-Langlois-Louvet)

EFT for the summation

$$x = \mathrm{fl}(a \pm b) \quad \Rightarrow \quad a \pm b = x + y \quad \text{with } y \in \mathbb{F},$$

Algorithms of Dekker (1971) and Knuth (1974)

Algorithm 1 (EFT of the sum of 2 floating point numbers with $|a| \geq |b|$)

function $[x, y] = \texttt{FastTwoSum}(a, b)$
$x = \mathrm{fl}(a + b)$
$y = \mathrm{fl}((a - x) + b)$

Algorithm 2 (EFT of the sum of 2 floating point numbers)

function $[x, y] = \texttt{TwoSum}(a, b)$
$x = \mathrm{fl}(a + b)$
$z = \mathrm{fl}(x - a)$
$y = \mathrm{fl}((a - (x - z)) + (b - z))$

EFT for the product (1/3)

$$x = \mathrm{fl}(a \cdot b) \quad \Rightarrow \quad a \cdot b = x + y \quad \text{with } y \in \mathbb{F},$$

Algorithm `TwoProduct` by Veltkamp and Dekker (1971)

$$a = x + y \quad \text{and} \quad x \text{ and } y \text{ non overlapping with } |y| \leq |x|.$$

Algorithm 3 (Error-free split of a floating point number into two parts)

function $[x, y] = \mathtt{Split}(a, b)$
  factor $= \mathrm{fl}(2^s + 1)$ % $\mathbf{u} = 2^{-p}$, $s = \lceil p/2 \rceil$
  $c = \mathrm{fl}(\mathtt{factor} \cdot a)$
  $x = \mathrm{fl}(c - (c - a))$
  $y = \mathrm{fl}(a - x)$

EFT for the product (2/3)

Algorithm 4 (EFT of the product of 2 floating point numbers)

function $[x, y] = \texttt{TwoProduct}(a, b)$
$\quad x = \mathrm{fl}(a \cdot b)$
$\quad [a_1, a_2] = \texttt{Split}(a)$
$\quad [b_1, b_2] = \texttt{Split}(b)$
$\quad y = \mathrm{fl}(a_2 \cdot b_2 - (((x - a_1 \cdot b_1) - a_2 \cdot b_1) - a_1 \cdot b_2))$

EFT for the product (3/3)

Given $a, b, c \in \mathbb{F}$,

- $\texttt{FMA}(a, b, c)$ is the nearest floating point number $a \cdot b + c \in \mathbb{F}$

Algorithm 5 (EFT of the product of 2 floating point numbers)

function $[x, y] = \texttt{TwoProductFMA}(a, b)$
  $x = \mathrm{fl}(a \cdot b)$
  $y = \texttt{FMA}(a, b, -x)$

The FMA is available for example on PowerPC, Itanium, Cell processors.

Summary

Theorem 1

Let $a, b \in \mathbb{F}$ *and let* $x, y \in \mathbb{F}$ *such that* $[x, y] = \texttt{TwoSum}(a, b)$. *Then,*

$$a + b = x + y, \quad x = \mathrm{fl}(a + b), \quad |y| \leq \mathbf{u}|x|, \quad |y| \leq \mathbf{u}|a + b|.$$

The algorithm `TwoSum` *requires* 6 *flops.*
Let $a, b \in \mathbb{F}$ *and let* $x, y \in \mathbb{F}$ *such that* $[x, y] = \texttt{TwoProduct}(a, b)$. *Then,*

$$a \cdot b = x + y, \quad x = \mathrm{fl}(a \cdot b), \quad |y| \leq \mathbf{u}|x|, \quad |y| \leq \mathbf{u}|a \cdot b|,$$

The algorithm `TwoProduct` *requires* 17 *flops.*

Classic method for computing product

The classic method for evaluating a product of n numbers $a = (a_1, a_2, \dots, a_n)$

$$p = \prod_{i=1}^{n} a_i$$

is the following algorithm.

Algorithm 6 (Product evaluation)

function $\texttt{res} = \texttt{Prod}(a)$
  $p_1 = a_1$
  for $i = 2 : n$
    $p_i = \mathrm{fl}(p_{i-1} \cdot a_i)$ % rounding error π_i
  end
  $\texttt{res} = p_n$

This algorithm requires $n - 1$ flops

Error analysis

$$\gamma_n := \frac{n\mathbf{u}}{1 - n\mathbf{u}} \quad \text{for } n \in \mathbb{N}.$$

A forward error bound is

$$|a_1 a_2 \cdots a_n - \mathtt{res}| \leq \gamma_{n-1} |a_1 a_2 \cdots a_n|$$

A validated error bound is

$$|a_1 a_2 \cdots a_n - \mathtt{res}| \leq \mathrm{fl}\left(\frac{\gamma_{n-1}|\mathtt{res}|}{1 - 2\mathbf{u}}\right)$$

Compensated method for computing product

Algorithm 7 (Product evaluation with a compensated scheme)

function $\mathtt{res} = \mathtt{CompProd}(a)$
  $p_1 = a_1$
  $e_1 = 0$
  for $i = 2 : n$
    $[p_i, \pi_i] = \mathtt{TwoProduct}(p_{i-1}, a_i)$
    $e_i = \mathrm{fl}(e_{i-1} a_i + \pi_i)$
  end
  $\mathtt{res} = \mathrm{fl}(p_n + e_n)$

This algorithm requires $19n - 18$ flops

Compensated method for computing product

Algorithm 8 (Product evaluation with a compensated scheme with `TwoProductFMA` and `FMA`)

function $\texttt{res} = \texttt{CompProdFMA}(a)$

$\quad p_1 = a_1$

$\quad e_1 = 0$

$\quad$ for $i = 2 : n$

$\qquad [p_i, \pi_i] = \texttt{TwoProductFMA}(p_{i-1}, a_i)$

$\qquad e_i = \texttt{FMA}(e_{i-1}, a_i, \pi_i)$

$\quad$ end

$\quad \texttt{res} = \mathrm{fl}(p_n + e_n)$

This algorithm requires $3n - 2$ flops

Error analysis

Theorem 2

Suppose Algorithm `CompProd` *is applied to floating point number* $a_i \in \mathbb{F}$, $1 \leq i \leq n$, *and set* $p = \prod_{i=1}^{n} a_i$. *Then,*

$$|\mathtt{res} - p| \leq \mathbf{u}|p| + \gamma_n \gamma_{2n} |p|$$

Condition number of the product evaluation :

$$\mathrm{cond}(a) = \lim_{\varepsilon \to 0} \sup \left\{ \frac{|(a_1 + \Delta a_1) \cdots (a_n + \Delta a_n) - a_1 \cdots a_n|}{\varepsilon |a_1 a_2 \cdots a_n|} : |\Delta a_i| \leq \varepsilon |a_i| \right\}$$

A standard computation yields

$$\mathrm{cond}(a) = n$$

Numerical experiments

Laptop with a Pentium M processor at 1.73GHz with `gcc` version 4.0.2.

Tab.: Measured computing times with `Prod` normalised to 1.0

n	`Prod`	`CompProd`
100	1.0	3.5
500	1.0	4.4
1000	1.0	5.0
10000	1.0	4.9
100000	1.0	5.5

The theoretical ratio for `CompProd` is 19.

Compensated algorithms are generally faster than the theoretical performances

$\rightarrow$ due to a better instruction-level parallelism.

Validated error bound

Lemma 1

Suppose Algorithm `CompProd` *is applied to floating point numbers* $a_i \in \mathbb{F}$, $1 \leq i \leq n$ *and set* $p = \prod_{i=1}^{n} a_i$. *Then, the absolute forward error affecting the product is bounded according to*

$$|\mathtt{res} - p| \leq \mathrm{fl}\left(\left(\mathbf{u}|\mathtt{res}| + \frac{\gamma_n \gamma_{2n} |a_1 a_2 \cdots a_n|}{1 - (n+3)\mathbf{u}}\right) / \left(1 - 2\mathbf{u}\right)\right).$$

Faithful rounding (1/3)

Floating point predecessor and successor of a real number r satisfying $\min\{f : f \in \mathbb{R}\} < r < \max\{f : f \in \mathbb{F}\}$:

$$\mathrm{pred}(r) := \max\{f \in \mathbb{F} : f < r\} \quad \text{and} \quad \mathrm{succ}(r) := \min\{f \in \mathbb{F} : r < f\}.$$

Definition 1

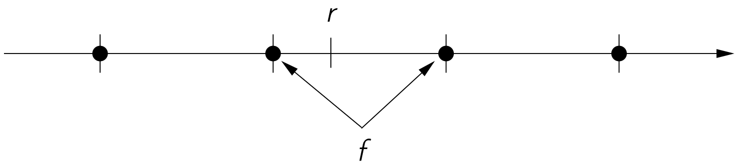
A floating point number $f \in \mathbb{F}$ is called a faithful rounding of a real number $r \in \mathbb{R}$ if

$$\mathrm{pred}(f) < r < \mathrm{succ}(f).$$

We denote this by $f \in \square(r)$. For $r \in \mathbb{F}$, this implies that $f = r$.

Faithful rounding means that the computed result is equal to the exact result if the latter is a floating point number and otherwise is one of the two adjacent floating point numbers of the exact result.

Faithful rounding (2/3)



Lemma 2 (Rump, Ogita and Oishi, 2005)

Let $r, \delta \in \mathbb{R}$ *and* $\widetilde{r} := \mathrm{fl}(r)$. *Suppose that* $2|\delta| < \mathbf{u}|\widetilde{r}|$. *Then* $\widetilde{r} \in \Box(r+\delta)$, *that means* $\widetilde{r}$ *is a faithful rounding of* $r+\delta$.

Faithful rounding (3/3)

Let $\texttt{res} = \texttt{CompProd}(p)$

Lemma 3

If $n < \frac{\sqrt{1-\mathbf{u}}}{\sqrt{2}\sqrt{2+\mathbf{u}}+2\sqrt{(1-\mathbf{u})\mathbf{u}}}\mathbf{u}^{-1/2}$ *then* $\texttt{res}$ *is a faithful rounding of* p*.*

If $n < \alpha\mathbf{u}^{-1/2}$ where $\alpha \approx 1/2$ then the result is faithfully rounded

In double precision where $\mathbf{u} = 2^{-53}$, if $n < 2^{25} \approx 5 \cdot 10^7$, we get a faithfully rounded result

Validated error bound and faithful rounding

If

$$\mathrm{fl}\left(2\frac{\gamma_n\gamma_{2n}|a_1a_2\cdots a_n|}{1-(n+3)\mathbf{u}}\right) < \mathrm{fl}(\mathbf{u}|\mathtt{res}|)$$

then we got a faitfully rounded result. This makes it possible to check *a posteriori* if the result is faithfully rounded.

Exponentiation - a linear algorithm

Algorithm 9 (Power evaluation with a compensated scheme)

function $\texttt{res} = \texttt{CompLinPower}(x, n)$

$p_1 = x$

$e_1 = 0$

for $i = 2 : n$

$[p_i, \pi_i] = \texttt{TwoProduct}(p_{i-1}, x)$

$e_i = \mathrm{fl}(e_{i-1}x + \pi_i)$

end

$\texttt{res} = \mathrm{fl}(p_n + e_n)$

Complexity : $\mathcal{O}(n)$

In double precision where $\mathbf{u} = 2^{-53}$, if $n < 2^{25} \approx 5 \cdot 10^7$, we get a faithfully rounded result

A double-double library

Algorithm 10 (Multiplication of two double-double numbers)

function $[r_h, r_l] = \texttt{prod\_dd\_dd}(a_h, a_l, b_h, b_l)$
  $[t_1, t_2] = \texttt{TwoProduct}(a_h, b_h)$
  $t_3 = \mathrm{fl}(((a_h \cdot b_l) + (a_l \cdot b_h)) + t_2)$
  $[r_h, r_l] = \texttt{TwoProduct}(t_1, t_3)$

Algorithm 11 (Multiplication of double-double number by a double number)

function $[r_h, r_l] = \texttt{prod\_dd\_d}(a, b_h, b_l)$
  $[t_1, t_2] = \texttt{TwoProduct}(a, b_h)$
  $t_3 = \mathrm{fl}((a \cdot b_l) + t_2)$
  $[r_h, r_l] = \texttt{TwoProduct}(t_1, t_3)$

A double-double library

Theorem 3 (Lauter 2005)

Let be $a_h + a_l$ and $b_h + b_l$ the double-double arguments of Algorithm `prod_dd_dd`*. Then the returned values r_h and r_l satisfy*

$$r_h + r_l = ((a_h + a_l) \cdot (b_h + b_l))(1 + \varepsilon)$$

where ε is bounded as follows : $|\varepsilon| \leq 16\mathbf{u}^2$. Furthermore, we have $|r_l| \leq \mathbf{u}|r_h|$.

A logarithmic algorithm

Algorithm 12 (Power evaluation with a compensated scheme)

```
function res = CompLogPower(x, n)          % n = (n_t n_{t-1} ··· n_1 n_0)_2
   [h, l] = [1, 0]
   for i = t : -1 : 0
      [h, l] = prod_dd_dd(h, l, h, l)
      if n_i = 1
          [h, l] = prod_dd_d(x, h, l)
      end
   end
   res = fl(h + l)
```

Complexity : $\mathcal{O}(\log n)$

A logarithmic algorithm

Theorem 4

The two values h and l returned by Algorithm `CompLogPower` *satisfy*

$$h + l = x^n(1+\varepsilon)$$

with

$$(1-16\mathbf{u}^2)^{n-1} \le 1+\varepsilon \le (1+16\mathbf{u}^2)^{n-1}.$$

For example, in double precision where $\mathbf{u} = 2^{-53}$, if $n < 2^{49} \approx 5 \cdot 10^{14}$, then we get a faithfully rounded result.

Thank you for your attention