ICNAAM 2006

Minisymposium on Self-validating methods and applications

Choosing a Twice More Accurate Dot Product Implementation

Stef Graillat

Université Paris 6

http://www-anp.lip6.fr/~graillat

Joint work with Philippe Langlois and Nicolas Louvet (Université de Perpignan)

Motivation: best way to use FMA for accurate dot products?

- IEEE-754 floating point arithmetic, with rounding to the nearest.
- no undeflow nor overflow

Outline

Notations

- IEEE-754 floating point arithmetic + FMA:
  - $\mathbb{F}$ denotes the set of the floating point numbers,
  - $\mathbf{u}$ is the working precision:
    e.g. $\mathbf{u} = 2^{-53} \approx 10^{-16}$ in IEEE-754 double precision.
- Floating point Fused Multiply and Add (FMA):
  - given a, b and c in $\mathbb{F}$, $\mathrm{FMA}(a, b, c)$ equals $a \times b + c$ rounded to the nearest floating point value.
  - only one rounding error for two arithmetic operations!
- Available on Intel IA-64, IBM RS/6000, PowerPC, Cell.
- $x = (x_1, \ldots, x_n)^T$ and $y = (y_1, \ldots, y_n)^T$ belong to $\mathbb{F}^{n\times 1}$.
- The condition number for the computation of $x^T y$ is

$$\mathrm{cond}(x^T y) = 2\frac{|x|^T|y|}{|x^T y|}, \quad \text{for} \quad x^T y \neq 0.$$

Accuracy of the classic dot product

- We consider dot products without/with FMA:

Algo. (Classic Dot)

function $\widehat{s} = \mathbf{Dot}(x, y)$
  $\widehat{s} = x_1 \otimes y_1$
  for $i = 2 : n$
    $\widehat{s} = \widehat{s} \oplus x_i \otimes y_i$

Algo. (Dot with FMA)

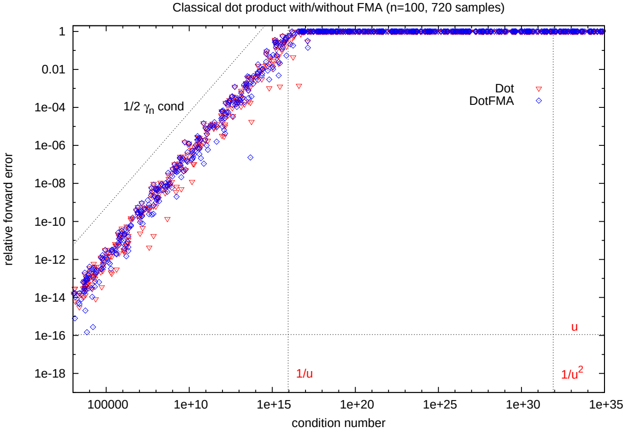
function $\widehat{s} = \mathbf{DotFMA}(x, y)$
  $\widehat{s} = x_1 \otimes y_1$
  for $i = 2 : n$
    $\widehat{s} = \mathit{FMA}(x_i, y_i, \widehat{s})$

- **Worst case accuracy:** FMA does not improve the accuracy of computed dot product since **Dot** and **DotFMA** both verifies

$$\frac{|\widehat{s} - x^T y|}{|x^T y|} \leq \frac{1}{2} \underbrace{\gamma_n}_{\approx n\mathbf{u}} \mathrm{cond}(x^T y).$$

Practical accuracy

- FMA only slightly improves the actual accuracy:



- Not accurate enough when applied to ill-conditioned dot products *e.g.* when computing residuals for ill-conditioned linear systems.
- **Question:** How can we obtain more accurate dot products?

How can we obtain more accuracy?

1. More bits:
   - Extended internal precision (80 bits register on x86)
   - Arbitrary precision libraries (MP, MPFR, Arprec/MPFUN...)
   - A reference in scientific computing: fixed length expansions libraries, such as double-double ($\mathbf{u}^2$) and quad-double ($\mathbf{u}^4$) (Berkeley)
2. Compensated algorithms:
   - Algorithms that correct the generated rounding errors.
   - Many examples: Kahan's compensated summation (65), Priest's doubly compensated summation (92), Ogita-Rump-Oishi (SISC 05)...
   - The rounding errors are computed thanks to error free transformations.

Principle of the compensated algorithms

- The forward error in the floating point evaluation of $x^T y$ is

$$c = x^T y - \text{computed}(x^T y).$$

- The main idea is to compute an approximate $\widehat{c}$ of the global error c thanks to **Error Free Transformations** (EFT).
- Then a **compensated result** $\overline{r}$ is provided correcting the computed $x^T y$ as follows,

$$\overline{r} = \text{computed}(x^T y) \oplus \widehat{c}.$$

Error free transformations (EFT)

EFT are properties and algorithms to compute the rounding errors at the current working precision.

$+$	$(x, y) = \mathbf{2Sum}(a, b)$ such that $a + b = x + y$ and $x = a \oplus b$	6 flops	Knuth (74)
$\times$	$(x, y) = \mathbf{2Prod}(a, b)$ such that $a \times b = x + y$ and $x = a \otimes b$	17 flops	Veltkamp Dekker (71)
$\times$	$(x, y) = \mathbf{2ProdFMA}(a, b)$ such that $a \times b = x + y$ and $x = a \otimes b$ Indeed $y = a \times b - x = \text{FMA}(a, b, -x)$	2 flops	
FMA	$(x, y, z) = \mathbf{3FMA}(a, b, c)$ such that $x = \text{FMA}(a, b, c)$ and $a \times b + c = x + y + z$	17 flops	Boldo Muller (05)

Compensated dot products

- From **Dot** and **DotFMA**, we derive two compensated algorithms using EFT:
  - **CompDot**: correcting $+$ and $\times$ in **Dot** with **2Sum** and **2ProdFMA** (see Ogita-Rump-Oishi (SISC 05)).
  - **CompDotFMA**: correcting FMA in **DotFMA** with **3FMA**.

Algo. (Compensated Dot)

function $\bar{r} = \mathbf{CompDot}(x, y)$
  $[\widehat{s}, \widehat{c}] = \mathbf{2ProdFMA}(x_1, y_1)$
  for $i = 2 : n$
    $[\widehat{p}, \pi] = \mathbf{2ProdFMA}(x_i, y_i)$
    $[\widehat{s}, \sigma] = \mathbf{2Sum}(\widehat{s}, \widehat{p})$
    $\widehat{c} = \widehat{c} \oplus (\pi \oplus \sigma)$
  end
  $\bar{r} = \widehat{s} \oplus \widehat{c}$

Algo. (Compensated DotFMA)

function $\bar{r} = \mathbf{CompDotFMA}(x, y)$
  $[\widehat{s}, \widehat{c}] = \mathbf{2ProdFMA}(x_1, y_1)$
  for $i = 2 : n$
    $[\widehat{s}, \alpha, \beta] = \mathbf{3FMA}(x_i, y_i, \widehat{s})$
    $\widehat{c} = \widehat{c} \oplus (\alpha \oplus \beta)$
  end
  $\bar{r} = \widehat{s} \oplus \widehat{c}$

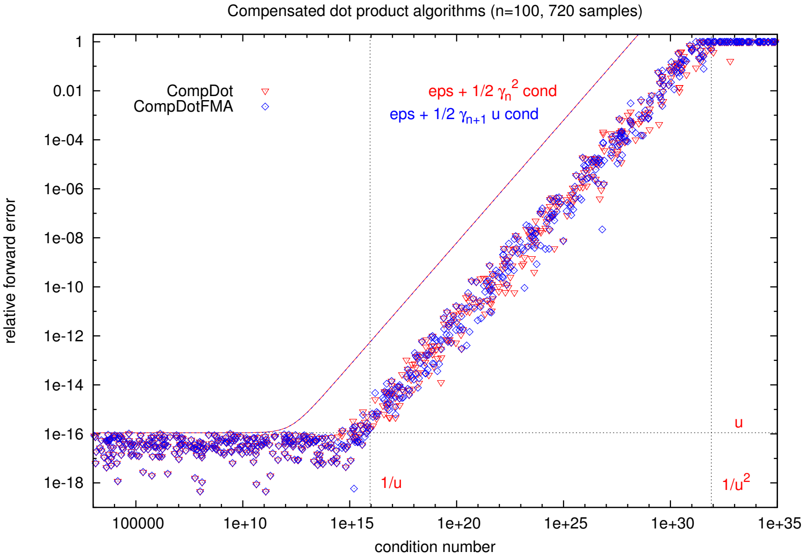
Worst case accuracy

- The relative accuracy of the compensated result now verifies:

$$\frac{|\,\overline{r} - x^T y|}{|x^T y|} \leq \begin{cases} \mathbf{u} + \frac{1}{2} \underbrace{\gamma_n^2}_{\approx n^2 \mathbf{u}^2} \mathrm{cond}(x^T y), & \text{with} \quad \textbf{CompDot}, \\ \\ \mathbf{u} + \frac{1}{2} \underbrace{\gamma_{n+1} \mathbf{u}}_{\approx (n+1) \mathbf{u}^2} \mathrm{cond}(x^T y), & \text{with} \quad \textbf{CompDotFMA}. \end{cases}$$

- **CompDot** and **CompDotFMA** are both as accurate as classic dot product computed in doubled working precision $\mathbf{u}^2$.

Accuracy of the result $\lesssim \mathbf{u} +$ condition number $\times\, \mathbf{u}^2$.



XBLAS dot product

- **XBLAS** = BLAS + Bailey's double-doubles = eXtended and mixed precision BLAS.
- A double-double number = unevaluated sum of two IEEE-754 double precision numbers = at least 106 significand bits.
- **DotXBLAS** = Classic dot product (**Dot**) + double-doubles.
- **DotXBLAS** also benefits from the availability of FMA.

What is the running time overcost?

- We mesure here the running time overcost of **CompDot**, **CompDotFMA** and **DotXBLAS** compared to **DotFMA**.

n	**DotFMA**	**CompDot**	**CompDotFMA**	**DotXBLAS**
50	1.0	1.63	2.61	9.87
100	1.0	1.35	2.43	9.65
1000	1.0	1.26	2.6	10.86
10000	1.0	1.25	2.62	10.97
100000	1.0	1.25	2.35	9.8

Measured computing times on Intel Itanium 2
(1.6 GHz, ICC v9.0, IEEE-754 double precision)

- **Observations:**
  1. **CompDot** and **CompDotFMA** run both faster than **DotXBLAS**,
  2. **CompDot** is the most efficient alternative to **DotXBLAS**.

Conclusion (1/2)

- FMA only slightly improves the accuracy of the classic dot product.
- Nevertheless FMA is useful for designing accurate algorithms: **CompDot** and **CompDotFMA** are very efficient for doubling the working precision.
- In particular **CompDot** is about 6 times faster than XBLAS algorithm **DotXBLAS** in our experiments.

Conclusion (2/2)

- FMA is useful to compute the error in the multiplication.
- Revision of IEEE 754 should include **tailadd** and **tailmultiply**. FMA makes it possible to compute **tailmultiply** efficiently.
- Similar results with the Compensated Horner Scheme.