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

1. Accuracy of the classic dot product
2. How can we obtain more accuracy?
3. Practical efficiency
4. Conclusion
Notations

- IEEE-754 floating point arithmetic + FMA:
  - $\mathbb{F}$ denotes the set of the floating point numbers,
  - $u$ is the working precision:
    - e.g. $u = 2^{-53} \approx 10^{-16}$ in IEEE-754 double precision.

- Floating point Fused Multiply and Add (FMA):
  - given $a$, $b$ and $c$ in $\mathbb{F}$, $\text{FMA}(a, b, c)$ 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
  \[
  \text{cond}(x^T y) = 2 \frac{|x|^{T} |y|}{|x^T y|}, \quad \text{for} \quad x^T y \neq 0.
  \]
Accuracy of the classic dot product

- We consider dot products without/with FMA:

  **Algo. (Classic Dot)**

  \[
  \text{function } \hat{s} = \text{Dot}(x, y) \\
  \hat{s} = x_1 \otimes y_1 \\
  \text{for } i = 2 : n \\
  \hat{s} = \hat{s} \oplus x_i \otimes y_i
  \]

  **Algo. (Dot with FMA)**

  \[
  \text{function } \hat{s} = \text{DotFMA}(x, y) \\
  \hat{s} = x_1 \otimes y_1 \\
  \text{for } i = 2 : n \\
  \hat{s} = \text{FMA}(x_i, y_i, \hat{s})
  \]

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

  \[
  \frac{|\hat{s} - x^T y|}{|x^T y|} \leq \frac{1}{2} \gamma_n \text{cond}(x^T y).
  \]

  \[\approx nu\]
Practical accuracy

- **FMA** only slightly improves the actual accuracy:

![Graph showing relative forward error vs. condition number for classical dot product with and without FMA.](image)

- 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 ($u^2$) and quad-double ($u^4$) (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.
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 $\hat{c}$ of the global error $c$ thanks to Error Free Transformations (EFT).

Then a **compensated result** $\bar{r}$ is provided correcting the computed $x^T y$ as follows,

$$\bar{r} = \text{computed}(x^T y) \oplus \hat{c}.$$
Error free transformations (EFT)

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Formula</th>
<th>Cost</th>
<th>Author/Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>$(x, y) = 2\text{Sum}(a, b)$</td>
<td>6 flops</td>
<td>Knuth (74)</td>
</tr>
<tr>
<td></td>
<td>such that $a + b = x + y$ and $x = a \oplus b$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplication</td>
<td>$(x, y) = 2\text{Prod}(a, b)$</td>
<td>17 flops</td>
<td>Veltkamp Dekker (71)</td>
</tr>
<tr>
<td></td>
<td>such that $a \times b = x + y$ and $x = a \otimes b$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Indeed $y = a \times b - x = \text{FMA}(a, b, -x)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FMA</td>
<td>$(x, y, z) = 3\text{FMA}(a, b, c)$</td>
<td>17 flops</td>
<td>Boldo Muller (05)</td>
</tr>
<tr>
<td></td>
<td>such that $x = \text{FMA}(a, b, c)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>and $a \times b + c = x + y + z$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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)

```latex
\text{function } \tilde{r} = \text{CompDot} (x, y)\\
[\hat{s}, \hat{c}] = \text{2ProdFMA} (x_1, y_1)\\
\text{for } i = 2 : n\\
[\hat{p}, \pi] = \text{2ProdFMA} (x_i, y_i)\\
[\hat{s}, \sigma] = \text{2Sum} (\hat{s}, \hat{p})\\
\hat{c} = \hat{c} \oplus (\pi \oplus \sigma)\\
\text{end}\\
\tilde{r} = \hat{s} \oplus \hat{c}
```

Algo. (Compensated DotFMA)

```latex
\text{function } \tilde{r} = \text{CompDotFMA} (x, y)\\
[\hat{s}, \hat{c}] = \text{2ProdFMA} (x_1, y_1)\\
\text{for } i = 2 : n\\
[\hat{s}, \alpha, \beta] = \text{3FMA} (x_i, y_i, \hat{s})\\
\hat{c} = \hat{c} \oplus (\alpha \oplus \beta)\\
\text{end}\\
\tilde{r} = \hat{s} \oplus \hat{c}
```
Worst case accuracy

The relative accuracy of the compensated result now verifies:

\[
\frac{|\bar{r} - x^Ty|}{|x^Ty|} \leq \begin{cases} 
    u + \frac{1}{2} \gamma_n^2 \text{ cond}(x^Ty), & \text{with \ CompDot,} \\
    \approx n^2 u^2 
\end{cases}
\]

\[
\begin{cases} 
    u + \frac{1}{2} \gamma_{n+1} u \text{ cond}(x^Ty), & \text{with \ CompDotFMA.} \\
    \approx (n+1) u^2 
\end{cases}
\]

CompDot and CompDotFMA are both as accurate as classic dot product computed in doubled working precision $u^2$. 
Accuracy of the result $\lesssim u + \text{condition number} \times u^2$. 

Compensated dot product algorithms (n=100, 720 samples)
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 measure here the running time overcost of **CompDot**, **CompDotFMA** and **DotXBLAS** compared to **DotFMA**.

<table>
<thead>
<tr>
<th>$n$</th>
<th>DotFMA</th>
<th>CompDot</th>
<th>CompDotFMA</th>
<th>DotXBLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1.0</td>
<td>1.63</td>
<td>2.61</td>
<td>9.87</td>
</tr>
<tr>
<td>100</td>
<td>1.0</td>
<td>1.35</td>
<td>2.43</td>
<td>9.65</td>
</tr>
<tr>
<td>1000</td>
<td>1.0</td>
<td>1.26</td>
<td>2.6</td>
<td>10.86</td>
</tr>
<tr>
<td>10000</td>
<td>1.0</td>
<td>1.25</td>
<td>2.62</td>
<td>10.97</td>
</tr>
<tr>
<td>100000</td>
<td>1.0</td>
<td>1.25</td>
<td>2.35</td>
<td>9.8</td>
</tr>
</tbody>
</table>

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.
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.