Choosing a twice more accurate dot product implementation

Stef Graillat, Philippe Langlois, Nicolas Louvet

Choosing a Twice More Accurate Dot Product Implementation

Stef Graillat, Philippe Langlois and Nicolas Louvet
Laboratoire LP2A, Université de Perpignan Via Domitia,
52, avenue Paul Alduy, F-66860 Perpignan Cedex, France
{graillat, langlois, nicolas.louvet}@univ-perp.fr

Abstract

The fused multiply and add (`FMA`) operation computes a floating point multiplication followed by an addition or a subtraction as a single floating point operation. Intel IA-64, IBM RS/6000 and PowerPC architectures implement this `FMA` operation. The aim of this talk is to study how the `FMA` improves the computation of dot product with classical and compensated algorithms. The latters double the accuracy of the former at the same working precision. Six algorithms are considered. We present associated theoretical error bounds. Numerical experiments illustrate the actual efficiency in terms of accuracy and running time. We show that the `FMA` does not improve in a significant way the accuracy of the result whereas it increases significantly the actual speed of the algorithms.

The fused multiply and add (`FMA`) operation computes a floating point multiplication followed by an addition or a subtraction as a single floating point operation. This means that only one final rounding (to the working precision) error is generated by a `FMA` whereas two occur in the classical implementation of $x \times y + z$. Intel IA-64, IBM RS/6000 and PowerPC architectures implement this `FMA` operation. On the Itanium processor, the `FMA` operation enables a multiplication and an addition to be performed in the same number of cycles than one multiplication or one addition [4].

The `FMA` operation seems to be advantageous for both speed and accuracy. Indeed, it approximately halves the number of rounding errors in many numerical algorithms. This is the case for example within the computation of a dot product of two n-vectors where just n rounding errors occur instead of $2n-1$ without `FMA`.

Moreover, it is well known that `FMA` yields an efficient computation of the rounding error generated by a floating point product. Such rounding error computation at the current working precision is a key task when implementing multi-precision libraries as double-double or quad-double ones [1] or even when designing compensated algorithms. Compensated algorithms implement inner computation of the rounding errors generated by the original algorithms and so provide more accurate results; [6, 5] are examples of compensated summation and dot product. The latter reference recently proved that these compensated implementations double the accuracy of the classical algorithm still running in the current working precision.

Here we study how the `FMA` can improve the computation of dot products in terms of accuracy and running time.

First, we consider the classical dot product computed at the working precision with or without `FMA`. We report the theoretical error analysis (worst case bounds) and some experimental results to show that the use of `FMA` only slightly improve the accuracy of the computed dot product, even if the number of rounding errors is halved.

Nevertheless, the accuracy provided by the classical dot product may not be sufficient when applied to ill conditioned dot products. Such cases appear for instance when computing residuals for ill conditioned linear systems. So we also consider accurate dot products whose computed result is as accurate as if computed in twice the working precision. Here we consider the classical dot product performed with double-double computation as it can be found in the XBLAS library [3] and the compensated dot product from [5] where the `FMA` is used to compute the rounding error generated by each product. We also present a new compensated dot product using a recent algorithm by Boldo and Muller [2] that computes the exact result of a `FMA` operation as the unevaluated sum of three floating point values. We present theoretical error bounds to prove that all these algorithms provide results as accurate as if computed in twice the working precision. Then we compare these implementations in terms of practical computing time to identify the best choice to double the computing precision. Our experimental results show that

the compensated algorithms run both considerably faster than the one with double-double computation. Moreover, the most efficient approach to benefit from the availability of a `FMA` seems to be to compensate the rounding error generated by each (classical) product without using the `FMA` operation in the inner loop of the dot product.

Algorithm	Brief description
`Dot`	Dot product without `FMA`
`DotFMA`	Dot product with `FMA`
`Dot2FMA`	Compensated dot product with `FMA`
`DotThreeFMA`	Compensated `DotFMA` with `FMA`
`DotXBLASFMA`	XBLAS dot product with `FMA`

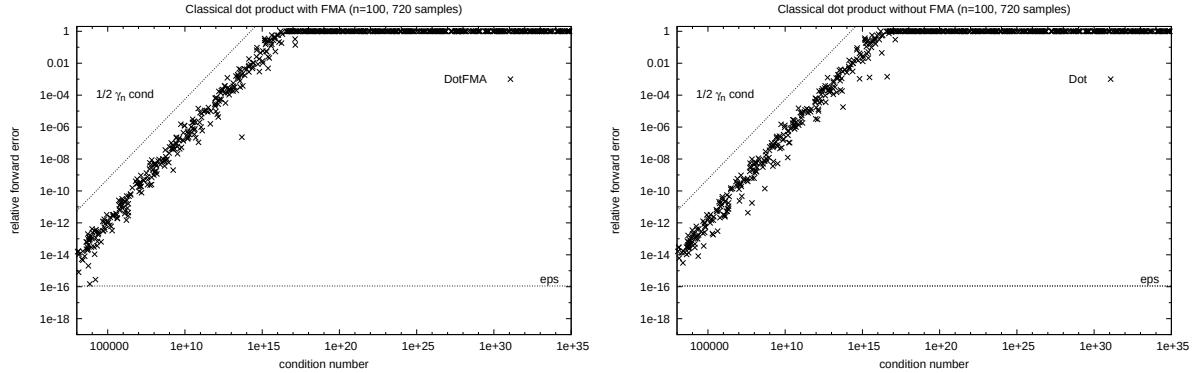

Figure 1: Classical dot product with and without `FMA`: the `FMA` does not improve the actual accuracy.

	n	`DotFMA`	`Dot2FMA`	`DotThreeFMA`	`DotXBLASFMA`
Theoretical		1	10	19	22
Measured	50	1.0	1.4	2.3	8.24
	100	1.0	1.29	2.37	8.98
	1000	1.0	1.24	2.63	10.46
	10000	1.0	1.25	2.63	10.5
	100000	1.0	1.07	1.76	6.27

Table 1: Measured computing times (Itanium 2, 1600 MHz, Intel C++ Compiler v9.0). Using the `FMA` to compensate the multiplication rounding error (`Dot2FMA`) is the best choice to double the accuracy.

References

[1] David H. Bailey. A Fortran-90 double-double library. Available at URL = `http://www.nersc.gov/~dhb/mpdist/mpdist.html`, 2001.

[2] Sylvie Boldo and Jean-Michel Muller. Some functions computable with a fused mac. In IEEE, editor, *IEEE Symposium on Computer Arithmetic ARITH'17*, Cape Cod, Massachusetts, USA, June 2005.

[3] Xiaoye S. Li, James W. Demmel, David H. Bailey, Greg Henry, Yozo Hida, Jimmy Iskandar, William Kahan, Suh Y. Kang, Anil Kapur, Michael C. Martin, Brandon J. Thompson, Teresa Tung, and Daniel J. Yoo. Design, implementation and testing of extended and mixed precision BLAS. *ACM Trans. Math. Softw.*, 28(2):152–205, 2002.

[4] Peter Markstein. *IA-64 and elementary functions. Speed and precision.* Hewlett-Packard Professional Books. Prentice-Hall PTR, 2000.

[5] Takeshi Ogita, Siegfried M. Rump, and Shin'ichi Oishi. Accurate sum and dot product. *SIAM J. Sci. Comput.*, 26(6):1955–1988, 2005.

[6] Michèle Pichat. Correction d'une somme en arithmétique à virgule flottante. *Numer. Math.*, 19:400–406, 1972.