Note on the Veltkamp/Dekker Algorithms with Directed Roundings

Paul Zimmermann
Université de Lorraine, CNRS, Inria, LORIA

February 27, 2024

Abstract

The Veltkamp/Dekker algorithms are very useful for double-double arithmetic, when no fused-multiply-add is available in hardware. Their analysis is well-known for rounding to nearest-even [4, 1, 3]. We study how they behave with directed roundings in radix 2.

1 Veltkamp's Splitting

We recall Veltkamp's algorithm in radix 2, using the description from [4, Algorithm 4.9], where the floating-point constant $C = 2^s + 1$ is precomputed, and $\circ(\cdot)$ denotes a rounding:

Algorithm 1 Split(x, s)

Input: a p-bit floating-point number x, an integer s, $2 \le s \le p-2$
Output: floating-point numbers x_h, x_ℓ such that $x = x_h + x_\ell$

1: $C = 2^s + 1$
2: $\gamma = \circ(Cx)$
3: $\delta = \circ(x - \gamma)$
4: $x_h = \circ(\gamma + \delta)$
5: $x_\ell = \circ(x - x_h)$

In [4], it is said that Dekker [2] proves this algorithm in radix 2 (for rounding to nearest-even), with the implicit assumption that neither overflow nor underflow occurs, and that Boldo [1] shows that for any radix and any precision p, provided that Cx does not overflow, the algorithm works (again for rounding to nearest-even). We consider in this note the case where $\circ(\cdot)$ is a directed rounding from IEEE 754: towards $-\infty$, towards $+\infty$, or towards zero. We further assume the *same rounding* is used in all steps of Veltkamp's algorithm.

Theorem 1 *When $\circ(\cdot)$ is a directed rounding, assuming no overflow, the outputs x_h, x_ℓ from Veltkamp's algorithm satisfy $x = x_h + x_\ell$, x_h fits in $p - s$ bits, and x_ℓ fits in s bits.*

Proof: Let us first assume that $\mathrm{ulp}(x)$ is greater or equal to the smallest positive normal number, say y. Since $|\gamma| \ge |x|$, it is easy to see that all numbers $\gamma, \delta, x_h, x_\ell$ are integer multiples of y, thus there is no underflow in the algorithm. We mimic the proof of [4]. The rounding towards zero satisfies $\circ(-e) = -\circ(e)$ for any expression e, thus it is easy to see that the values of $\gamma, \delta, x_h, x_\ell$ obtained by Veltkamp's algorithm for x and $-x$ are opposite. Similarly, if we run Veltkamp's

algorithm once for x and rounding towards $+\infty$, and then for $-x$ and rounding towards $-\infty$, the values obtained for x and x' are opposite. We can thus restrict to $x > 0$ (the case $x = 0$ being trivial). If we multiply x by 2^k for an integer k, all variables in the algorithm are scaled by 2^k, thus we can assume $1 \le x < 2$. If $x = 1$, then $\gamma = C = 2^s + 1$, $\delta = -2^s$, $x_h = 1$, $x_\ell = 0$, thus the statement of the theorem holds. We can thus assume $1 < x < 2$, which yields $1 + 2^{-p+1} \le x \le 2 - 2^{-p+1}$ since x is a p-bit floating-point number.

Computation of γ. $Cx = (2^s + 1)x$ implies $2^s + 1 < Cx < 2^{s+2}$, therefore $2^{s-p+1} \le \mathrm{ulp}(Cx) \le 2^{s-p+2}$. This gives $\gamma = (2^s + 1)x + \varepsilon_1$ with $|\varepsilon_1| < 2^{s-p+2}$, and γ is a multiple of 2^{s-p+1}.

Computation of δ. We have $\delta = x - \gamma + \varepsilon_2$ with ε_2 the rounding error in the computation of δ. This yields $\delta = -2^s x - \varepsilon_1 + \varepsilon_2$.

We have $x - \gamma = -2^s x - \varepsilon_1$, thus $|x - \gamma| < 2^s(2 - 2^{-p+1}) + 2^{s-p+2} = 2^{s+1} + 2^{s-p+1}$. But for $|x - \gamma|$ to exceed 2^{s+1}, we need $\varepsilon_1 > 0$, thus rounding towards $+\infty$; in that case since $-2^{s+1} - 2^{s-p+1} < x - \gamma < -2^{s+1}$, $x - \gamma$ is rounded to -2^{s+1}, thus in all cases $|\delta|$ lies in the binade $[2^s, 2^{s+1}]$. This implies that δ is a multiple of 2^{s-p+1}. From Sterbenz's lemma, since $-\gamma$ and δ are within a factor of 2 from each other, $\gamma + \delta$ is computed exactly: $x_h = \gamma + \delta = x + \varepsilon_2$. Since both γ and δ are multiples of 2^{s-p+1}, so is x_h. We also have $|\varepsilon_2| < 2^{s-p+1}$: either $|x - \gamma|$ lies in the binade $[2^s, 2^{s+1}]$, and it follows from the fact that $\varepsilon_2 < \mathrm{ulp}(x - \gamma)$, otherwise we have seen that $-2^{s+1} - 2^{s-p+1} < x - \gamma < -2^{s+1}$, and that $x - \gamma$ is rounded to -2^{s+1}, thus $|\varepsilon_2| < 2^{s-p+1}$ too. From $x < 2$ and $x_h = x - \varepsilon_2$, one deduces $x_h < 2 + 2^{s-p+1}$, but the only multiple of 2^{s-p+1} between 2 and $2 + 2^{s-p+1}$ is 2, so $x_h \le 2$.

Computation of x_ℓ. By Sterbenz's lemma, $x - x_h$ is computed exactly, thus $x_\ell = x - x_h = \varepsilon_2$. As a consequence, $|x_\ell| = |\varepsilon_2|$ is less than 2^{s-p+1}. Moreover x_ℓ is a multiple of 2^{-p+1}, since x and x_h are mutiples of 2^{-p+1}.

In summary, we have $x = x_h + x_\ell$ with:

- $x_h \le 2$ and x_h multiple of 2^{s-p+1} imply that x_h fits in $p - s$ bits;
- $|x_\ell| < 2^{s-p+1}$ and x_ℓ multiple of 2^{-p+1} imply that x_ℓ fits in s bits.

Now assume that $\mathrm{ulp}(x)$ is smaller than the smallest positive normal number y. Letting $2^e \le |x| < 2^{e+1}$, then by scaling the bounds of the above proof we find the bounds $|\varepsilon_1| < 2^{e+s-p+2}$ and $|\varepsilon_2| < 2^{e+s-p+1}$. If $2^{e+s-p+1}$ is greater or equal to the smallest positive subnormal number α, these bounds hold, and the theorem too. Now assume $2^{e+s-p+2} \le \alpha$: this means that $|x| < 2^{e+1} \le 2^{p-s-1}\alpha$. It follows $(2^s + 1)|x| < (2^s + 1)2^{p-s-1}\alpha < 2^p\alpha$, thus γ is computed exactly, $\delta = -2^s x$, $x_h = x$, $x_\ell = 0$, and the statement holds. ∎

Note that the statement from Theorem 1 does not hold when different roundings are used for the computation of γ and δ. For example with $p = 4$, $s = 2$, $x = 15/8$, if γ is rounded upwards we get $\gamma = 10$, then if δ is rounded downwards we get $\delta = -9$, which yields $x_h = 1$ and $x_\ell = 7/8$, which does not fit in 2 bits.

If overflow occurs in the computation of $\gamma = \circ(Cx)$, the error bound $\gamma = (2^s + 1)x + \varepsilon_1$ with $|\varepsilon_1| < 2^{e+s-p+2}$ no longer holds. Thus the statement Theorem 1 is the best possible.

Algorithm 2 Dekker's product

Input: x, y two p-bit floating-point numbers

Output: floating-point numbers r_1, r_2 such that $x \cdot y = r_1 + r_2$

1: $s = \lceil p/2 \rceil$
2: $(x_h, x_\ell) = Split(x, s)$
3: $(y_h, y_\ell) = Split(y, s)$
4: $r_1 = \circ(xy)$
5: $t_1 = \circ(-r_1 + \circ(x_h y_h))$
6: $t_2 = \circ(t_1 + \circ(x_h y_\ell))$
7: $t_3 = \circ(t_2 + \circ(x_\ell y_h))$
8: $r_2 = \circ(t_3 + \circ(x_\ell y_\ell))$

2 Dekker's Product

Again, we use the algorithm from [4, Algorithm 4.10]:

Theorem 2 *When $\circ(\cdot)$ is a directed rounding and p is even, assuming no overflow nor underflow, the outputs r_1, r_2 from Dekker's algorithm satisfy $xy = r_1 + r_2$.*

Proof: Again, we mimic the proof from [4]. From Theorem 1, we know that x_h, y_h fit into $p - s$ bits, that x_ℓ, y_ℓ fit into s bits, $x = x_h + x_\ell$, and $y = y_h + y_\ell$. Since p is even, $s = p - s = p/2$, and all four products $x_h y_h$, $x_h y_\ell$, $x_\ell y_h$, $x_\ell y_\ell$ fit into p bits, thus $t_1 = \circ(-r_1 + x_h y_h)$, $t_2 = \circ(t_1 + x_h y_\ell)$, $t_3 = \circ(t_2 + x_\ell y_h)$, $r_2 = \circ(t_3 + x_\ell y_\ell)$. Without loss of generality, we can assume $1 \le x, y < 2$. We have seen in the proof of Theorem 1 that $x_h, y_h < 2$, and that $|x - x_h|, |y - y_h| \le 2^{s-p+1}$. From

$$xy - x_h y_h = (x - x_h)y + (y - y_h)x_h,$$

we deduce

$$|xy - x_h y_h| \le 2^{s-p+3}.$$

Since we also have

$$|xy - r_1| \le \mathrm{ulp}(xy) \le 2^{-p+2},$$

we get

$$|r_1 - x_h y_h| \le 2^{-p+2} + 2^{s-p+3}.$$

This shows that r_1 and $\circ(x_h y_h) = x_h y_h$ are very close, so Sterbenz's lemma can be applied to their subtraction. As a consequence,

$$t_1 = -r_1 + x_h y_h.$$

Now, $xy - x_h y_h = x_h y_\ell + x_\ell y_h + x_\ell y_\ell$, so

$$\begin{aligned} |t_1 + x_h y_\ell| &= |-r_1 + x_h y_h + x_h y_\ell| \\ &= |-r_1 + xy + (x_h y_h + x_h y_\ell - xy)| \\ &\le |-r_1 + xy| + |x_\ell (y_h + y_\ell)| \\ &< 2^{-p+2} + (2^{s-p+1} - 2^{-p+1}) \cdot 2 \le 2^{s-p+2}, \end{aligned}$$

where we have used the fact that $|x_\ell| < 2^{s-p+1}$ and x_ℓ is a multiple of 2^{-p+1}, thus $|x_\ell| \le 2^{s-p+1} - 2^{-p+1}$, and $|y_h + y_\ell| = |y| < 2$. Since x_h is a multiple of 2^{s-p+1} and y_ℓ is a multiple of 2^{-p+1}, $x_h y_\ell$ is

a multiple of 2^{s-2p+2}. This implies that $t_1 +_h y_\ell$ is a multiple of 2^{s-2p+2}. This and $|t_1 +_h y_\ell| < 2^{s-p+2}$ imply that $t_1 +_h y_\ell$ is exactly representable. Therefore,

$$t_2 = -r_1 + x_h y_h + x_h y_\ell.$$

Now, $t_2 + x_\ell y_h = (-r_1 + xy) - x_\ell y_\ell$; therefore,

$$\begin{aligned} |t_2 + x_\ell y_h| &\leq |-r_1 + xy| + |x_\ell y_\ell| \\ &\leq 2^{-p+2} + 2^{2s-2p}. \end{aligned}$$

Since $2s \leq p+1$,

$$|t_2 + x_\ell y_h| \leq 2^{-p+2} + 2^{-p+1} \leq 2^{-p+3}.$$

This and the fact that $t_2 + x_\ell y_h$ is a multiple of $u := 2^{s-2p+2}$ imply that $|t_2 + x_\ell y_h| \leq 2^{p+1-s} \cdot u$, thus $t_2 + x_\ell y_h$ is exactly representable; therefore,

$$t_3 = t_2 + x_\ell y_h = -r_1 + x_h y_h + x_h y_\ell + x_\ell y_h.$$

Finally, $|t_3 + x_\ell y_\ell| = |-r_1 + xy| \leq 2^{-p+2}$, and it is a multiple of 2^{-2p+2}, thus $t_3 + x_\ell y_\ell$ is exactly computed. Therefore,

$$r_2 = xy - r_1.$$

■

For p odd, the statement of Theorem 2 does not hold. Take for example $p = 3$, $x = y = 5/4$, and rounding towards zero. We get $x_h = y_h = 2$, $x_\ell = y_\ell = -3/4$. It follows $r_1 = 3/2$, $t_1 = 5/2$, $t_2 = 1$, $t_3 = -1/2$, $r_2 = 0$. The issue is that since x_ℓ and y_ℓ both fit into 2 bits, their product fits into 4 bits and not 3, thus $x_\ell y_\ell$ is not exact.

Lemma 1 *When $\circ(\cdot)$ is a directed rounding and p is odd, assuming no overflow nor underflow, the outputs r_1, r_2 from Dekker's algorithm satisfy $xy = r_1 + r_2 + \varepsilon$ with $|\varepsilon| \leq 2^{e_x+e_y-2p}$, where $2^{e_x-1} \leq |x| < 2^{e_x}$ and $2^{e_y-1} \leq |y| < 2^{e_y}$ (assuming neither x nor y is zero).*

Proof: All the reasoning of the proof of Theorem 2 holds, except that $x_\ell y_\ell$ is not exactly representable. Let $t_4 = \circ(x_\ell y_\ell)$. Since $|x_\ell y_\ell| < 2^{2s-2p+2}$, $t_4 = x_\ell y_\ell + \varepsilon$ with $|\varepsilon| < \mathrm{ulp}(x_\ell y_\ell) \leq 2^{2s-3p+2}$. Moreover, since p is odd, $2s = p+1$, thus $|\varepsilon| < 2^{-2p+3}$. We thus have $r_2 = \circ(-r_1 + xy + \varepsilon)$. In fact, since $x_\ell y_\ell$ is a multiple of 2^{-2p+2}, so is ε, and ε can only take three values: 0 and $\pm 2^{-2p+2}$. We thus have $|-r_1 + xy + \varepsilon| \leq |-r_1 + xy| + 2^{-2p+2}$. Since $|-r_1 + xy| < \mathrm{ulp}(xy) \leq 2^{-p+2}$ and $|-r_1 + xy|$ is a multiple of 2^{-2p+2}, we have $|-r_1 + xy| \leq 2^{-p+2} - 2^{-2p+2}$. It follows $|-r_1 + xy + \varepsilon| \leq 2^{-p+2}$, and as in the case p even, $-r_1 + xy + \varepsilon$ is exactly representable, thus $r_2 = -r_1 + xy + \varepsilon$. ■

The example above with $p = 3$ has $|\varepsilon| = 2^{-2p+2}$, which shows the bound from Lemma 1 is optimal. Overflow can occur in the computation of $r_1 = \circ(xy)$, in which case the error bounds no longer hold; in particular if $|xy|$ exceeds twice the largest representable number Ω, there is no way we can write $xy = r_1 + r_2$ with $|r_1|, |r_2| \leq \Omega$. Underflow can occur in the computation of $\circ(x_\ell y_\ell)$, in which case the error bounds no longer hold; in particular if the exact product xy is not an integer multiple of the smallest positive subnormal α, there is no way we can write $xy = r_1 + r_2$ with both r_1, r_2 multiples of α. The statement of Theorem 2 is thus the best possible.

References

[1] Boldo, S. Pitfalls of a full floating-point proof: Example on the formal proof of the Veltkamp/Dekker algorithms. In *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings* (2006), U. Furbach and N. Shankar, Eds., vol. 4130 of *Lecture Notes in Computer Science*, Springer, pp. 52–66.

[2] Dekker, T. J. A floating-point technique for extending the available precision. *Numerische Mathematik 18*, 3 (1971), 224–242.

[3] Jeannerod, C.-P., Muller, J.-M., and Zimmermann, P. On various ways to split a floating-point number. In *ARITH 2018 - 25th IEEE Symposium on Computer Arithmetic* (Amherst (MA), United States, June 2018), IEEE, pp. 1–8.

[4] Muller, J.-M., Brunie, N., de Dinechin, F., Jeannerod, C.-P., Joldes, M., Lefèvre, V., Melquiond, G., Revol, N., and Torres, S. *Handbook of Floating-point Arithmetic (2nd edition)*. Birkhäuser Basel, July 2018.