Improving Average Delay for On-Line Arithmetic Algorithms

Colin D. Walter

Computation Department, U.M.I.S.T.,
PO Box 88, Sackville Street, Manchester M60 1QD, U.K.
e-mail: C.Walter@umist.ac.uk

Abstract. Using normalised input directly for on-line arithmetic does not always lead to minimal on-line delay. Shifting some inputs can reduce the average delay.

Index Terms: Computer arithmetic, multiplication, on-line delay, on-line algorithms, redundant number systems, recurrence relations.

1 Introduction

On-line algorithms consume their inputs and generate output digit serially at the same rate, most significant bit first. So there is a natural, fixed time delay δ between input and output of corresponding digits: the output digit of index j depends solely on the input digits of index $j+\delta$ or less. Such algorithms for multiplication, division and square root have been extensively studied in the literature, usually using a simple addition for the iterative step (see [2], [4], [1]), but sometimes using a fuller, more accurate computation (see [3]), or without all inputs being on-line (see [5]). However, it is invariably the case that only normalised or pseudo-normalised input is allowed (i.e. that in the half-open intervals $[1/2, 1)$ or $[1/4, 1)$), and this rarely fits well with an optimal choice for δ: δ could be reduced by changing the input interval. In practice, it is more convenient to retain the usual input intervals, but shift those inputs for which the reduced value of δ fails. The input conditions necessary for the reduced delay need to be established, and a procedure given for deciding which inputs should be shifted. They are considered here in the case of multiplication where, because of the small value of δ, the detail is at one time more significant and less obscured by the algorithm.

2 The Notation

We start with a definition of the multiplication algorithm. We consider mainly mantissas M of real numbers with redundant radix 2 representations $M \equiv$

$\sum_{t=1}^{\infty} m_t 2^{-t}$, where the digit m_t lies in the signed digit set $\{-1,0,+1\}$. At time $j \geq 0$, the first j digits provide inputs $X[j] \equiv \sum_{t=1}^{j} x_t 2^{-t}$ and $Y[j] \equiv \sum_{t=1}^{j} y_t 2^{-t}$. The output digit $p_{j-\delta}$ of the product P is formed using the scaled *partial remainder*

$$W[j] = 2^j\{X[j]\times Y[j] \;-\; P[j-1-\delta]\}$$

which is calculated using the recurrence relation

$$W[j+1] = 2W[j] \;+\; y_{j+1}X[j] \;+\; x_{j+1}Y[j] \;+\; 2^{-1-j}x_{j+1}y_{j+1} \;-\; 2^{1+\delta}p_{j-\delta}$$

with initial value $W[0] = 0$. The digit $p_{j-\delta}$ is chosen to minimise $W[j]$ if it were included in its definition, but speed is gained by using just the approximation $W[j]'$, which is given by truncating all digits after that of index ε. So, for some fixed integer i, we define

$$p_{j-\delta} = \begin{array}{rl} +1 \text{ if } & 2^\varepsilon W[j]' \geq i \\ 0 \text{ if } & |2^\varepsilon W[j]'| < i \\ -1 \text{ if } & 2^\varepsilon W[j]' \leq -i \end{array}$$

It is fairly straightforward to show by induction on j that

$$B[j] = 2^{1+\delta}-|X[j]|-|Y[j]|-2^{-j}$$

is a bound on the absolute value of $W[j]$ for $j > \delta$ when both

$$\begin{array}{l} 2^{1-\varepsilon}i \qquad + 4 \leq 2^{1+\delta} \text{ and} \\ 2^{1-\varepsilon}(1-i) + 4 \leq 0 \end{array}$$

hold. This is optimally the case for $\delta = 3$, $\varepsilon = -1$ and $i = 2$ or 3.

3 Input Bounds.

Redoing the proof of the algorithm for fixed δ, ε and i enables conditions on the inputs to be derived for which the algorithm will work. For the base case of the induction argument we require $|W[\delta]| \leq |B[\delta]|$. This certainly holds for $|X| < 1$ and $|Y| < 1$ since $|X[t]| \leq 1 - 2^{-t}$ and $|Y[t]| \leq 1 - 2^{-t}$ for all $t \geq 0$.

Now assume $W[j] \leq B[j]$ where $j \geq \delta$. For convenience, suppose $X > 0$, $Y > 0$ and $\varepsilon < \delta$. The induction step for $p_{j-\delta} = 1$ is easy to verify because the bounds $B[j]$ are chosen precisely so that it does work. So suppose $p_{j-\delta} = 0$. As $W[j]$ has at most j digits after the point, $W[j]+2^{-j} \leq W[j]'+2^{-\varepsilon} \leq 2^{-\varepsilon}i$. Hence, from the definition of $W[j+1]$,

$$W[j+1] \leq 2^{1-\varepsilon}i - 2^{1-j} + X[j+1] + Y[j+1] - 2^{-1-j}$$

which is at most $B[j+1]$ when

$$2^{1-\varepsilon}i + 2X + 2Y \leq 2^{1+\delta}$$

Now suppose $p_{j-\delta} = -1$. Then, $W[j]{+}2^{-j} \le W[j]'{+}2^{-\varepsilon} \le 2^{-\varepsilon}(1{-}i)$, and so,

$$W[j{+}1] \le 2^{1-\varepsilon}(1{-}i) - 2^{1-j} + X[j{+}1] + Y[j{+}1] - 2^{-1-j} + 2^{1+\delta}$$

This is at most $B[j{+}1]$ if

$$2^{1-\varepsilon}(1{-}i) + 2X + 2Y \le 0$$

Combining these two inequalities, we obtain convergence of the algorithm when

$$|X| + |Y| \le min\{2^{-\varepsilon}(i{-}1), 2^{\delta}{-}2^{-\varepsilon}i\}$$

Clearly, as $|X|{+}|Y| < 2$, this holds for the original choice of parameters.

4 Fractional Delays

An improved delay is obtained by the choice $\delta = 2$, $\varepsilon = 1$ and $i = 4$ or 5, for inputs satisfying $|X|{+}|Y| < 3/2$. If the multiplier receives normalised input, then over-large inputs must be shifted down until they are small enough to satisfy the initial conditions. A fractional average delay results which is an improvement on the earlier $\delta = 3$. The cost of this is i) not all output shows the same time delay, ii) the initial bound checking, and iii) the extra digit used in W' for computing the output digit.

Suppose the two inputs are uniformly and independently distributed over the interval $[1/2, 1)$. Half the inputs satisfy the bound and theoretically for them we could have just 2 cycles delay between input and output. For the other inputs, the larger input must be divided by 2 before the multiplication, resulting in 3 cycles delay. This apparently yields an improved average delay of 2.5 cycles. However, the exact test to see if the bound is met requires the examination of arbitrarily many digits, and so is not feasible since it must be completed before the output digits start to be generated. Thus any required action needs to be based on the first δ input digits. Already $x_1 = y_1 = 1$ because of the assumptions on the input range. Without looking at further digits, only when $x_2 = y_2 = 0$ is the input bound now guaranteed. Otherwise, just in case, either input, say X, must be halved to guarantee the condition holds. This can be done without changing the already processed first digit by inserting $\bar{1}$ for x_2 and delaying the actual x_2 and subsequent digits by one cycle. In consequence the output is generated one cycle later, i.e. as if δ were 3. Three quarters of all inputs are delayed in this way by one cycle. But, overall, an improved average delay δ of 2.75 is thus obtained.

5 Conclusion.

We have shown how to reduce the average delay for on-line algorithms at minor hardware cost, illustrating the ideas using the example of multiplication.

References

[1] Milos D. Ercegovac, "An On-Line Square Rooting Algorithm", *Proc. 4th IEEE Symposium on Computer Arithmetic*, Santa Monica, CA, 1978, pp. 183-189.

[2] Milos D. Ercegovac & Tomas Lang, "On-Line Arithmetic: A Design Methodology and Applications in Digital Signal Processing", *VLSI Signal Processing III*, R. W. Brodersen, H. S. Moscovitz eds., IEEE Press, New York, 1988, pp. 252-263.

[3] H. J. Sips & H. X. Lin, " A New Model for On-Line Arithmetic with an Application to the Reciprocal Calculation", *J. Parallel & Distrib. Comp.* vol. **8**, 1990, pp. 218-230.

[4] Kishor S. Trivedi & Milos D. Ercegovac, "On-Line Algorithms for Division and Multiplication", *IEEE Trans. Comput.*, vol. **C-26**, No 7, July 1977, pp. 681-687.

[5] J. H. Zurawski & J. B. Gosling, "Design of a high-speed square root, multiply and divide unit", *IEEE Trans. Comput.*, vol. **C-36**, 1987, pp. 13-23.