ETG-22-37, ETT-22-37

自然指数関数の分散型積和演算アルゴリズムの研究

Hemthavy Xaybandith*, 片山 翔吾, 桑名 杏奈, 小林 春夫 （群馬大学）

Distributed Arithmetic for Exponential Function

Xaybandith Hemthavy*, Shogo Katayama, Anna Kuwana, Haruo Kobayashi (Gunma University)

キーワード：分散型積和演算, 指数関数, テイラー展開, デジタル信号処理

(Distributed Arithmetic, Exponential Function, Taylor-Series Expansion, Digital Signal Processing)

1. はじめに

集積回路技術の進展に伴い、科学技術計算機から携帯機器まで広くデジタル浮動小数点演算が普及している。そこで用いられる様々な関数 $\exp(x)$, $\log(x)$, $\sqrt{x}$, $1/x$ 等の高速・高精度・小規模回路・低消費電力で計算できることが求められる。

筆者らはこれらの浮動小数点演算にテイラー展開を用いて乗算と加算で計算するアルゴリズムを研究してきた [1-3]。乗算器は加算器に比べて回路規模が大きく長い遅延(latency) であるので、できるだけ使用したくない。そこで積和演算を、乗算器を用いずにメモリと加算器だけで並列ビット演算をすることで小規模回路にて高速計算を行える分散型積和演算 [4-9]を適用することを検討した。

特に自然現象(アナログ)を表現する際に有用なネイピア数のべき乗、自然指数関数の演算回路を検討した。デジタル浮動小数点を扱い、e^xをテイラー展開して計算する際に、高精度を求めて項数を増やすほど乗算の回数が増えて、回路規模が大きくなってしまう[1]。乗算器が加算器に比べて回路規模が大きいにもかかわらず乗算器を多く搭載しなければならないためである。本論文は分散型積和演算アルゴリズムを用いることで乗算の回数を減らし、乗算器の代わりに加算器を増やして回路を構成できることを確認できたので報告する。

2. 加算器と乗算器

入力が N ビットの 2 入力加算器は全加算器（Full Adder）の N 個の 1 次元配列になる。一方、入力が N ビットの乗算器の直接的な実現回路は$N \times N$個の全加算器の 2 次元配列になる。乗算器は加算器より回路規模が大きくなり遅延(latency) も長い。(図 1) デジタルシステムではできるだけ乗算器を使用しないで実現したい。

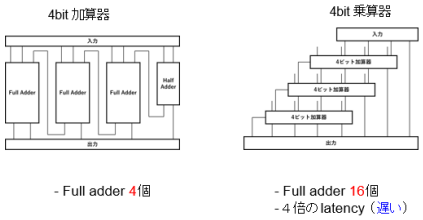


図 1. デジタル加算器と乗算器

3. 分散型積和演算

3-1. モデル

ここで検討する自然指数関数を$y = e^x$とする。xを 2 進数表記して$x = M * 2^E$と表現する。Mは浮動小数点表現での仮数部(Mantissa)であり、$1 \leq M < 2$とする。Eは指数部(Exponent)である。(図 2) したがって、e^xは$e^M * e^{2^E}$と 2 つの数の乗算と表現でき、それぞれをy_1とy_2と定義する。

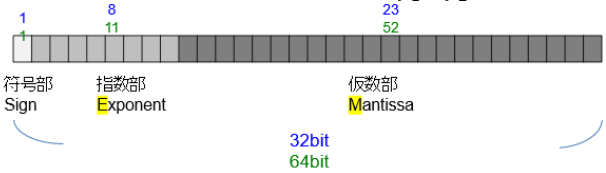


図 2. 64-bit, 32-bit 浮動小数点フォーマット

それぞれにテイラー展開を行った後に分散型積和演算を適用して計算した結果をまとめ、自然指数関数e^xの近似計算結果を得ることを検討する。

3-2. 分散型積和演算の方法

テイラー展開を 6 次まで行った場合を考える。

$$e^x = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6 \quad (1)$$

(1)式を分散型積和演算(並列ビットシリアル演算)で計算する。[4-9]

6 次までのテイラー展開の式においてそれぞれxからx^6までを 2 進数に変換して最も桁数の多い 6 次に桁数をそろえるように式(2)のように行列を作る。

$$\begin{pmatrix} x \\ x^2 \\ x^3 \\ x^4 \\ x^5 \\ x^6 \end{pmatrix} = \begin{pmatrix} \square & \square & \square. & \square & \square & \square & \square \\ \square & \square & \square. & \square & \square & \square & \square \\ \square & \square & \square. & \square & \square & \square & \square \\ \square & \square & \square. & \square & \square & \square & \square \\ \square & \square & \square. & \square & \square & \square & \square \\ \square & \square & \square. & \square & \square & \square & \square \end{pmatrix} \quad (2)$$

このとき行列の要素が 1 であるところのみ、それぞれ(1)

式より行に対応する係数の、列ごとの和を計算する。例としてxの行であれば 1 を、x^2の行であれば1/2が対応する係数である。この和を左の列より、現在の列の和を 2 倍して次の列の和を加算する作業を最後の列まで行い、小数点以下の桁数nの分、$\frac{1}{2^n}$をかけることで(1)式の両辺から 1 を引いた近似計算結果が得られる。

この分散型積和演算は次の式(3)の様に表せる。

$$e^x - 1 = \left(2 * \cdots \left(2 * \left(\begin{matrix} 2 * (2 * \square) \\ +\square \end{matrix} \right) \right) \right) * \frac{1}{2^n} \qquad (3)$$
$$\cdots + \square$$

両辺に 1 を足すことで最終的なe^xの答えを得る。

3-3. 分散型積和演算による乗算回数の削減

上記のように 6 次までのテイラー展開の式において、直接の計算では乗算を行う個所はx^2からx^6を得るまでの 5 回とそれらと係数との積 5 回の合計 10 回である。分散型積和演算ではx^2からx^6までを同様に 5 回の乗算で求めた後、加算器とメモリを用いた並列ビットシリアル演算で計算できるため、乗算の回数は 5 回である。これはテイラー展開するxの次数を増やしても同様に係数との積の分を削減できるため、常に乗算の回数は通常の直接的な計算の半分となる。

なお、(3)式での 2 や$\frac{1}{2^n}$の積は回路においてビットシフトで実現できるため、乗算回数に含めないで考える。

3-4. 実行例

10 進で$x = 1.5$は 2 進では$x = 1.1$となり、e^xは$e^{1.1}$となる。6 次までテイラー展開すると次のようになる。

$$e^{1.1} = 1 + 1.1 + \frac{1}{2}1.1^2 + \frac{1}{6}1.1^3 + \frac{1}{24}1.1^4 + \frac{1}{120}1.1^5 + \frac{1}{720}1.1^6$$

1.1 のべき乗を計算し、(2)式の行列を作ると次のようになる。

$$\begin{pmatrix} 1.1 \\ 1.1^2 \\ 1.1^3 \\ 1.1^4 \\ 1.1^5 \\ 1.1^6 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1. & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0. & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1. & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1. & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1. & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1. & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

分散型積和演算を適用すると(4)式が得られる。

$$e^{1.1} - 1 = \left(2 * \left(2 * \left(2 * \left(2 * \left(2 * \left(2 * \left(2 * \frac{1}{720} \right) + \left(\frac{1}{120} + \frac{1}{24} \right) \right) + \left(\frac{1}{720} + \frac{1}{120} + \frac{1}{6} + \frac{1}{2} \right) \right) + \left(\frac{1}{720} + \frac{1}{120} + \frac{1}{24} + \frac{1}{6} + 1 \right) \right) + \left(\frac{1}{120} + 1 \right) \right) + \left(\frac{1}{720} + \frac{1}{6} + \frac{1}{2} \right) \right) + \left(\frac{1}{720} + \frac{1}{6} \right) \right) + \left(\frac{1}{120} + \frac{1}{24} \right) \right) + \frac{1}{120} \right) + \frac{1}{720} \right) * \left(\frac{1}{2} \right)^6 \qquad (*)$$

この式を計算すると、$e^{1.1(2\text{ 進})} = 4.477439063(10\text{ 進})$ が得られ、元の$e^{1.5(10\text{ 進})}$の電卓での計算結果、4.48168907と近い値が得られることが確認できた。(図 4)

4. 項数の増加と精度向上

分散型積和演算アルゴリズムを用いて計算するテイラー展開の式の項数の増加に伴う計算結果の精度の向上について検証する。ここでは先ほど計算した$e^{1.5}$について検証していく。精度をAccとして以下のように算出する。

$$Acc = \frac{(\textit{分散型積和演算アルゴリズム計算結果})}{(\textit{電卓計算結果})} \times 100\ (\%)$$

縦軸にAcc、横軸にテイラー展開した最終項のxの次数をとったグラフを以下の図 3 に示す。

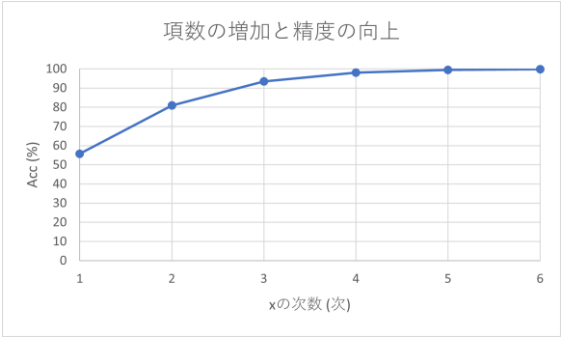


図 3. 6 次までの項数の増加と精度の向上

以上より、分散型積和演算アルゴリズムを適用するテイラー展開の式の項数を増やすほど精度は 100%に収束することが分かる。

5. 回路実装にあたって

(2)式ではxからx^6までの 6 行の行列としたために回路実装の際に、6 ワード以上のメモリが必要となる。そこで行列をそれぞれ分割して 2 行ずつまたは 3 行ずつの行列として順次計算してまとめる回路構成でメモリを減らし、回路規模のさらなる小型化ができる。テイラー展開の次数が大きい場合により有効である。(図 5)

6. 他関数への応用

分散型積和演算アルゴリズムは以下の式で記述されるように、xのべき乗と係数との積の形にテイラー展開できる関数であれば他のどの関数にも適用可能である。

$$f(x) = ax + bx^2 + cx^3 + dx^4 + ex^5 + \cdots$$

例に、$\frac{1}{\sqrt{x}}$の計算に分散型積和演算アルゴリズムを適用してみる。まず$\frac{1}{\sqrt{x}}$をaまわりでテイラー展開すると

$$\frac{1}{\sqrt{x}} = \frac{1}{\sqrt{a}}\left\{1 - \frac{x-a}{2a} + \frac{3(x-a)^2}{8a^2} - \frac{5(x-a)^3}{16a^3} + \frac{35(x-a)^4}{128a^4} - \cdots\right\}$$

となるので、$a = 1$としてテイラー展開し、さらに式変形すると以下のようにまとめられる。

$$\frac{1}{\sqrt{1+x}} = 1 + (\frac{3}{8}x^2 + \frac{35}{128}x^4 + \cdots) - (\frac{1}{2}x + \frac{5}{16}x^3 + \cdots)$$

上式より 1 と正の項と負の項の和にまとめることができるので正の項と負の項についてそれぞれ分散型積和演算アルゴリズムを適用して計算することで$\frac{1}{\sqrt{x}}$の計算結果が求まる。

ここで$\frac{1}{\sqrt{1.5}}$の計算を行うことを考えると、上式より

$$\frac{1}{\sqrt{(1+0.5)}} = 1 + (\frac{3}{8}x^2 + \frac{35}{128}x^4 + \frac{231}{1024}x^6) - (\frac{1}{2}x + \frac{5}{16}x^3 + \frac{63}{256}x^5)$$

とテイラー展開できる。正の項の和をy_1、負の項の和をy_2とすると、y_1のxの次数は 2、4、6 であり、y_2のxの次数は 1、3、5 であるのでそれぞれまとめてべき乗を 2 進数に変換して行列にまとめると以下のようになる。

$$\begin{pmatrix} x^2 \\ x^4 \\ x^6 \end{pmatrix} = \begin{pmatrix} 0 & . & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & . & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & . & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} x \\ x^3 \\ x^5 \end{pmatrix} = \begin{pmatrix} 0 & . & 1 & 0 & 0 & 0 & 0 \\ 0 & . & 0 & 0 & 1 & 0 & 0 \\ 0 & . & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

これらに分散型積和演算アルゴリズムを適用してそれぞれy_1、y_2を計算すると以下のように求まる。

$$y_1 = \begin{pmatrix} 2 \times \begin{pmatrix} 2 \times \begin{pmatrix} 2 \times \begin{pmatrix} 2 \times \frac{3}{8} \\ +0 \end{pmatrix} \\ +\frac{35}{128} \end{pmatrix} \\ +0 \end{pmatrix} \\ +\frac{231}{1024} \end{pmatrix} \times (\frac{1}{2})^6 = 0.114364624$$

$$y_2 = \begin{pmatrix} 2 \times \begin{pmatrix} 2 \times \begin{pmatrix} 2 \times \begin{pmatrix} 2 \times \frac{1}{2} \\ +0 \end{pmatrix} \\ +\frac{5}{16} \end{pmatrix} \\ +0 \end{pmatrix} \\ +\frac{63}{256} \end{pmatrix} \times (\frac{1}{2})^5 = 0.2967529297$$

よって、計算結果をまとめると分散型積和演算では

$$\frac{1}{\sqrt{1.5}} = 1 + 0.114364624 - 0.2967529297 = 0.8176116943$$

となる。電卓計算では0.8164965809であったことから$\frac{1}{\sqrt{x}}$のテイラー展開の計算にも分散型積和演算アルゴリズムは有効であることが分かる。同様に他の関数にも応用することが可能であることが推測できる。

7. まとめ

本論文では、特に自然指数関数の演算回路の小型化を実現するアルゴリズムについて検討した。今後は他の関数の場合でも効率的な演算回路構成の検討を行う。

文　　献

(1) J. Wei, A. Kuwana, H. Kobayashi, K. Kubo, “Divide and Conquer: Floating-Point Exponential Calculation Based on Taylor-Series Expansion”, IEEE 14th International Conference on ASIC (Oct. 2021)

(2) J. Wei, A. Kuwana, H. Kobayashi, K. Kubo, "IEEE754 Binary32 Floating-Point Logarithmic Algorithms based on Taylor-Series Expansion with Mantissa Region Conversion and Division" IEICE Trans. Fundamentals, Vol.E105-A, No.7 (Jul. 2022.)

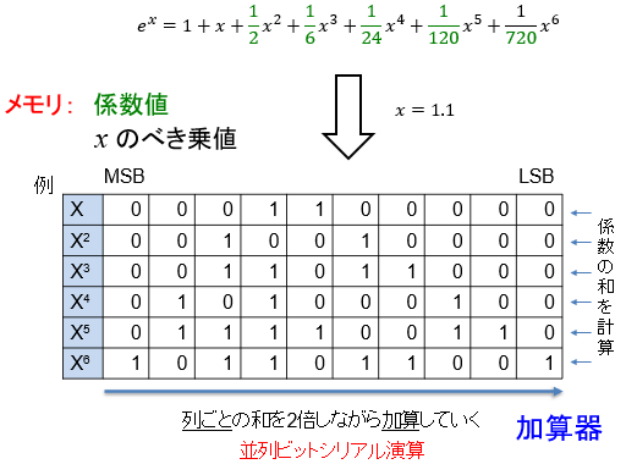
(3) J. Wei, A. Kuwana, H. Kobayashi, K. Kubo, Y. Tanaka, "Floating-Point Inverse Square Root Algorithm Based on Taylor-Series Expansion", IEEE Transactions on Circuits and Systems II: Express Briefs, Vol. 68, Issue 7, pp. 2640-2644 (July 2021).

(4) J. Wei, A. Kuwana, H. Kobayashi, K. Kubo, Y. Tanaka, “Floating-Point Inverse Square Root Algorithm Based on Taylor-Series Expansion”, IEEE Trans. Circuits and Systems Ⅱ:Express Briefs, Vol. 68, No. 7, July 2021

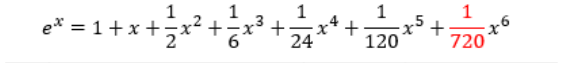
(5) M. T. Khan, R. A. Shaik, “High-Performance VLSI Architecture of DLMS Adaptive Filter for Fast-Convergence and Low-MSE”, IEEE Trans. Circuits and Systems Ⅱ, 10.1109/TCSⅡ.2022.3141687.

(6) E. Özalevli, W. Huang, P. E. Hasler, D. V. Anderson, “A Reconfigurable Mixed-Signal VLSI Implementation of Distributed Arithmetic Used for Finite Impulse Response Filtering”, IEEE Trans. Circuits and Systems- I : Vol.55, No. 2, pp. 510-521 (March 2008).

(7) Ali M. Al-Haj, “Fast Discrete Wavelet Transformation Using FPGAs and Distributed Arithmetic”, International Journal of Applied Science and Engineering, pp. 160-171 (Jan. 2003).
(8) S. Badave, A. Bhalchandra, “Critical Path Reduction of Distributed Arithmetic Based FIR Filter”, International Journal of Advanced Computer Science and Applications, Vol. 7, No. 3, pp. 71-77 (2016).
(9) R. Bala, S. Aktar, “Fast Fourier Transformation Realization with Distributed Arithmetic”, International Journal of Computer Applications, Vol. 102, No. 15, pp. 22-25 (Sept. 2014).
(10) R. Mehra, Ginne, “FPGA Based Gaussian Pulse Shaping Filter Using Distributed Arithmetic Algorithm”, International Journal of Scientific & Engineering Research Vol. 4, Issue 8, pp. 711-715 (Aug. 2013).



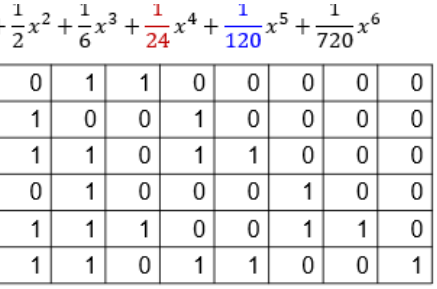
(a) 分散型積和演算器



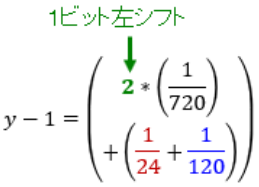
X	0	0	0	1	1	0	0	0	0	0
X²	0	0	1	0	0	1	0	0	0	0
X³	0	0	1	1	0	1	1	0	0	0
X⁴	0	1	0	1	0	0	0	1	0	0
X⁵	0	1	1	1	1	0	0	1	1	0
X⁶	1	0	1	1	0	1	1	0	0	1

$$y - 1 = \frac{1}{720}$$

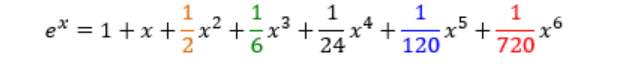
(b) ステップ 1 (MSB の計算)



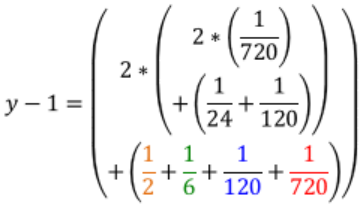
X	0	0	0	1	1	0	0	0	0	0
X²	0	0	1	0	0	1	0	0	0	0
X³	0	0	1	1	0	1	1	0	0	0
X⁴	0	1	0	1	0	0	0	1	0	0
X⁵	0	1	1	1	1	0	0	1	1	0
X⁶	1	0	1	1	0	1	1	0	0	1



(c) ステップ 2 (MSB-1 の計算)



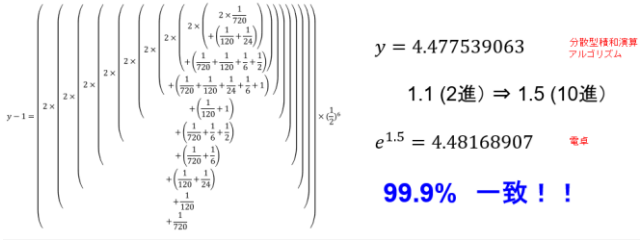
X	0	0	0	1	1	0	0	0	0	0
X²	0	0	1	0	0	1	0	0	0	0
X³	0	0	1	1	0	1	1	0	0	0
X⁴	0	1	0	1	0	0	0	1	0	0
X⁵	0	1	1	1	1	0	0	1	1	0
X⁶	1	0	1	1	0	1	1	0	0	1



(d) ステップ 3 (MSB-2 の計算)

$$e^x = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6$$

X	0	0	0	1	1	0	0	0	0	0
X^2	0	0	1	0	0	1	0	0	0	0
X^3	0	0	1	1	0	1	1	0	0	0
X^4	0	1	0	1	0	0	0	1	0	0
X^5	0	1	1	1	1	0	0	1	1	0
X^6	1	0	1	1	0	1	1	0	0	1



(e) 最終結果

図 4 分散型積和演算の各ステップ

X	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
X^2	0	0	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
X^3	0	1	0	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0
X^4	0	1	1	1	0	0	1	0	0	1	1	0	0	0	1	0	0	0	0	0
X^5	1	0	0	1	1	1	0	1	0	1	0	0	0	1	1	0	1	1	0	0
X^6	1	1	0	1	1	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0

$$e^x = 1 + \left(x + \frac{1}{2}x^2\right) + \left(\frac{1}{6}x^3 + \frac{1}{24}x^4\right) + \left(\frac{1}{120}x^5 + \frac{1}{720}x^6\right)$$

(a) 2 項まとめての演算

X	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
X^2	0	0	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
X^3	0	1	0	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0
X^4	0	1	1	1	0	0	1	0	0	1	1	0	0	0	1	0	0	0	0	0
X^5	1	0	0	1	1	1	0	1	0	1	0	0	0	1	1	0	1	1	0	0
X^6	1	1	0	1	1	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0

$$e^x = 1 + \left(x + \frac{1}{2}x^2 + \frac{1}{6}x^3\right) + \left(\frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6\right)$$

(b) 3 項まとめての演算

図 5 分散型積和演算の回路規模削減のための改良