

フィード・フォワード計算機の処理能力と計算アルゴリズム

PROCESSING POWER OF THE FEED-FORWARD MACHINE AND ITS ALGORITHM

富沢 方之⁺ 山田 秀和⁺ 吉岡 良雄⁺⁺ 中村 維男⁺ 重井 芳治⁺
Masayuki TOMISAWA Hidekazu YAMADA Yoshio YOSHIOKA Tadao NAKAMURA Yoshiharu SHIGEI

⁺ 東北大学 工学部

⁺⁺ 岩手大学 工学部

⁺ Faculty of Engineering, Tohoku University

⁺⁺ Faculty of Engineering, Iwate University

1. はじめに

近年、計算速度の向上を目指して、並列処理型計算機の様々なアーキテクチャが提案されている。その中の1つとして、著者らによって提案されたフィード・フォワード計算機(FFM)^{(1)~(5)}がある。

従来の計算機は、命令語のフェッチ及びそのデコードを行ってから、1つの演算回路の機能を切り替えて、計算を実行していた。これに対してFFMは、命令語もフェッチすると、デコードを待たずに複数の演算回路ですべての種類の演算を開始し、デコード結果によってその演算結果を選択する。これにより、制御と実行をある程度独立に行い、処理速度を上昇させ並列演算を実行することができる。

本稿では、FFMの並列性に着目し、その並列処理能力及びその計算アルゴリズムについて述べる。

2. FFMの概念と特徴

FFMの基本的動作概念は、「命令語をフェッチしてくると、デコードを待たずにすべての種類の演算を開始し、デコード結果によってその演算結果を選択する。」というものである。(図1参照)。これを実現するために、プロセッサ内には、機能を固定した複数の演算回路と、各演算回路に対して数ステップ以前までのすべての演算結果を格納しておくレジスタ群を持って

いる。(図2参照)。一対のオペランドに対して、これらの演算回路が同時に動作し、かつそれぞれの演算結果を各レジスタ群に格納することによって、ある程度の並列処理が可能となる。例えば、 $A+B$ 、 $A-B$ の計算等は1ステップで実行できる。

このように、FFMは従来のノイマン型計算機にくらべて処理速度の向上が期待できるが、それは、主に次のようなFFMの特徴による。

- 1). 同一オペランドに対して複数の異なる演算があれば、1ステップで並列に実行できる。

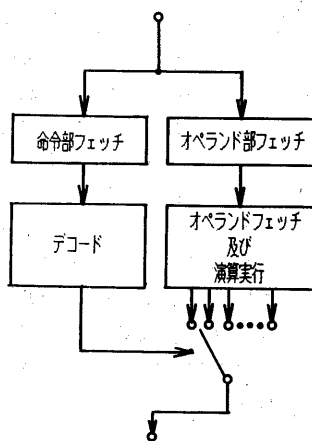


図1. FFMの動作概念

- 2). 各ALUに付加されている大量のレジスタ群によって、メモリ回避命令(ストア命令)が減少する。
- 3). デコード時間が節約される。

本論文では、この特徴の1)に関して議論する。まず、データ・フローグラフによりFFMの並列性を評価する。次に、実際の数値計算をFFMで実行する場合のアルゴリズム、及びその計算量について考察する。

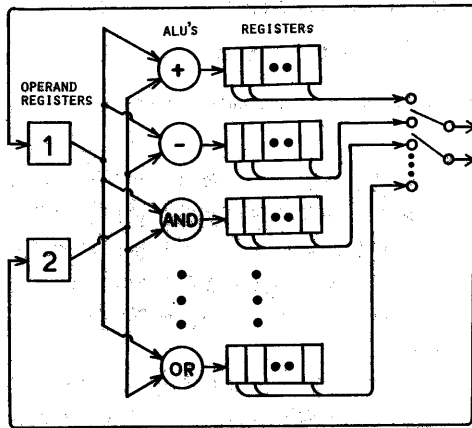


図2. FFMの構成

3. FFMの並列性の評価

3.1 理想計算機

Schwartz (1980) は、プロセッサ・アレイの性能評価のために、パラコンピュータの概念を導入した。⁽⁹⁾ このコンピュータは、処理エレメントを無限個持ち、これらのそれぞれがルーチング・ディレイやメモリ競合なしに、共通メモリを並列にアクセスできる。これに習い、プログラム中のいかなる並列性をも最大限に生かすことができ、従っていかなるプログラムも最短ステップ数で実行できるような理想的計算機(以下これを理想計算機と呼ぶ)を仮定する。以下、FFMの実行ステップ数を従来の計算機と理想計算機の実行ステップ数と比較し、並列性を評価していく。

3.2 FFMの並列性

図3(a)のような例を考える。この式に単一代入規則を適用し、2項演算に展開すると図3(b)のようになる。さらに、このプログラムのデータ・フローグラフを書くと、図3(c)のようになる。このプログラムを理想計算機で実行すると、5ステップで実行できる。(これを5パラステップと呼ぶことにする)。FFMで実行する場合、図3(c)の□の部分を1ステップで実行できるため、8ステップで実行できる。また、従来の計算機で実行すると、ノード数に等しいステップ数も必要とし、11ステップかかる。

ここで注目すべきことは、データ・フローグラフ中に、図3(c)の太線で描かれたような、

$$y = \frac{(a+b)*\{(a-b)+(c-d)\}}{\{(e-f)-(a-b)*(c-d)\}*\{(a-b)-(c-d)\}}$$

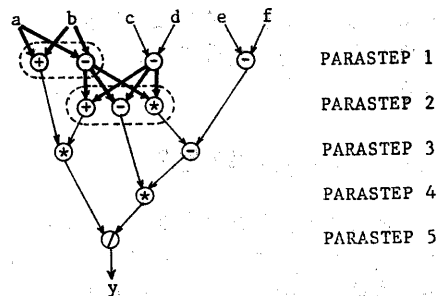
(a). SAMPLE EXPRESSION

```

t1 = a+b
t2 = a-b
t3 = c-d
t4 = t2+t3
t5 = t1*t4
t6 = e-f
t7 = t2*t3
t8 = t6-t7
t9 = t2-t3
t10 = t8*t9
y = t5/t10

```

(b). 式の展開



(c). データ・フローグラフ

図3. 式の中の並列性

ノード間の結線パターン（同一一対の入カデータに対して、複数の異なる種類の演算を行う場合に相当）が生じると、FFMにおいては1ステップで処理することができることである。

3.3 FFMの実行ステップ数の評価

本節では、データ・フローグラフのノードを与え、そのノード間結線のすべての組合せを考え、その中に上述のFFM向き結線パターンが、どの程度生じるかを調べ、FFMの実行ステップ数の期待値を求める。

図4 (a)のようにデータ・フローグラフのノードのみが与えられたものとする。パラステップ i に着目し、 i の各ノードが、1パラステップ前のノードとのみ結線されるものとする。結線の仕方の組合せは、図4 (b)に示すように27通り生じる。FFMにおいては、図4 (b)の $\cdots$ で囲まれた複数のノードを1ステップで実行することができる。従って、図4 (a)のようなノード・パターンを与えられた時の第 i パラステップの全ノードをFFMによって実行する場合のステップ数の期待値 $E(M_1, M_2)$ (ただし、 M_1, M_2 はそれぞれ、第 $i-1$ パラステップ、第 i パラステップのノード数)は、次のようになる。

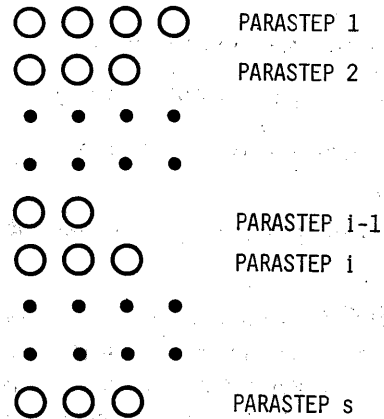
$$E(2, 3) = 1 \cdot \frac{3}{27} + 2 \cdot \frac{18}{27} + 3 \cdot \frac{6}{27} = 2.11$$

一般のデータ・フローグラフでは、あるパラステップ i に着目した時、1パラステップ前のノードとのみ結線されるとは限らず、それ以前のノードと結びつくような場合もかなり見受けられる。しかし、概してパラステップが i から1つ前、2つ前、3つ前、 $\cdots$ と離れていくにつれて、 i の各ノードとの結線が少なくなる傾向がある。従って、ここでは i の各ノードは、1パラステップ前の各ノードとのみ結線されるものと仮定して解析を行う。

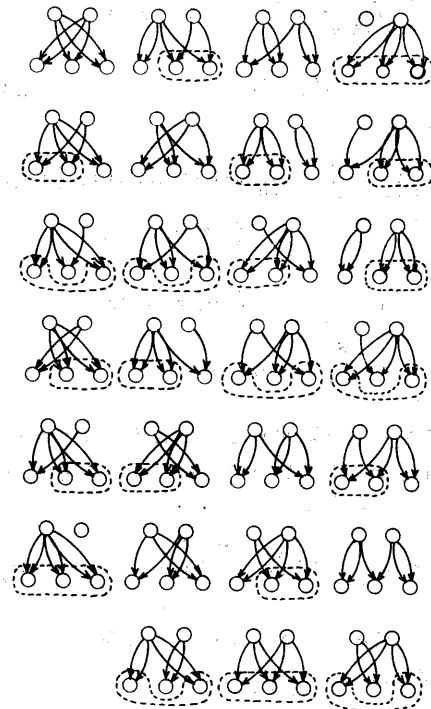
パラステップ1, 2, $\cdots$, S におけるノード数が、 $n_1, n_2, \cdots, n_S$ と与えられたものとする。このプログラムをFFMで実行する場合の実行ステップ数の期待値は、次式のようになる。

$$E_{\text{全}} = \sum_{i=1}^S E(n_{i-1}, n_i) \quad (1)$$

ただし、 n_0 は初期入カデータ数とする。



(a) . 与えられたノード・パタンの例



(b) . 結線の組合せ

図4 . ノード・パターンと結線の仕方

次に、 $E(M_1, M_2)$ の一般式を求める。 M_2 が、次式のように分割されるものとする。

$$\begin{aligned} M_2 &= (m_1 + m_1 + \dots + m_1) + (m_2 + m_2 + \dots + m_2) \\ &+ \dots + (m_n + m_n + \dots + m_n) \\ &= m_1 a_1 + m_2 a_2 + \dots + m_n a_n. \end{aligned} \quad (2)$$

ここで

$$m_1 \geq m_2 \geq \dots \geq m_n \geq 1, \quad a_i, m_i: \text{整数}$$

M_2 の分割要素 $m_1, m_1, \dots, m_1, m_2, m_2, \dots, m_2, \dots, m_n, m_n, \dots, m_n$ の各々が、FFMにおいては1ステップで実行できるものとする。 M_2 個の全ノードもFFMは、 $\sum_{i=1}^n a_i$ ステップで実行することができる。 M_1 と M_2 の結合パターンの中に、このような M_2 の分割 α が生じる確率 P_α は、次のようになる。

$$P_\alpha = \frac{\prod_{1 \leq j \leq \sum_{k=1}^n a_k} (m_k C_2 + M_1 - j + 1)}{(M_2 C_2 + M_1)^{M_2}} \prod_{1 \leq i \leq n} a_i \quad (3)$$

ここで

$$Q_i = \begin{cases} \frac{\prod_{1 \leq j \leq a_i} (M_2 - \sum_{k=1}^{i-1} a_k m_k - (j-1)m_i) C_2}{(m_i > 1 \text{ のとき})} \\ 1 & (m_i = 1 \text{ のとき}) \end{cases} \quad (4)$$

$$\text{ただし、} \sum_{k=1}^n a_k m_k = 0$$

M_2 のすべての分割の仕方（もれなく重複なく）について P_α を求めることにより、 $E(M_1, M_2)$ は次のように求まる。

$$E(M_1, M_2) = \sum_{\alpha} P_\alpha \sum_{i=1}^{n(\alpha)} a_i(\alpha) \quad (5)$$

ただし、分割 α は M_2 のすべての分割の中で

$$\sum_{i=1}^n a_i \leq (M_1 C_2 + M_1)$$

が成立するもののみを採用する。

また、 M_2 の分割は、図5に示すアルゴリズムを M_2 に適用することによって (ELEMENT(M_2, M_2))、すべての分割を重複なく求めることができる。

PROCEDURE ELEMENT(y,i);

begin

if y=0 then RETURN ($a_1, a_{i+1}, \dots, a_{M_2}$)

else if i=1 then

begin

$a_i \leftarrow y;$

RETURN ($a_1, a_{i+1}, \dots, a_{M_2}$)

end

else

for j:=0 to $\lfloor \frac{y}{i} \rfloor$ do

begin

$a_i \leftarrow j;$

ELEMENT ($y-i \cdot a_i, i-1$)

end

end

図5. 分割アルゴリズム

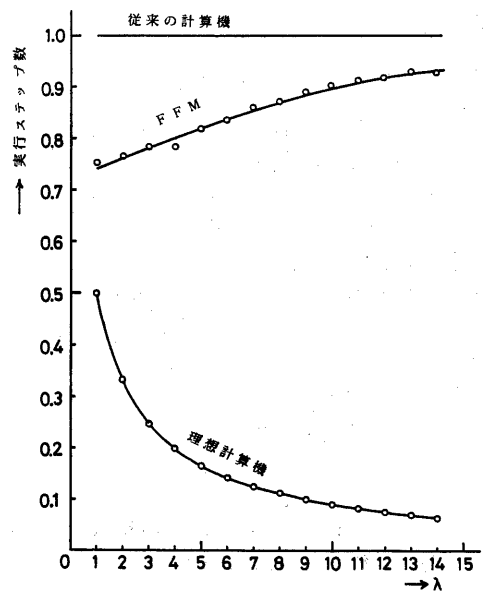


図6. FFM, 従来の計算機, 理想計算機の
実行ステップ数の期待値(相対値)

以上述べたことを用い、データ・フローグラフのノードをパラステップごとに乱数で与え、FFMの実行ステップ数の期待値を求める。
数値例として

- パラステップごとのノード数の分布 (ノード数が k 個の確率):

$$P_k = e^{-\lambda} \cdot \frac{\lambda^{k-1}}{(k-1)!}$$

- 全パラステップ数: $S = 50$

の場合について、求めたものを図6に示す。ただし、結果は従来の計算機の実行ステップ数で正規化してある。これより、FFMは、1パラステップ当りのノード数の平均が小さいほど、並列実行できる割合が高くなり、従来の計算機にくらべ、最大2~3割程度実行ステップ数の減少が期待できることがわかる。

4. FFM向き計算アルゴリズムと計算量

本章では、いくつかの数値計算をFFMで実行する場合の計算量をアルゴリズム・レベルで検討する。演算の種類としては、四則演算を考える。また、本FFMにおいては、加減算と乗除算の処理時間の違いを考慮する。すなわち、オペランドのフェッチとともにすべての演算を

開始するが、デコード結果が加減算だった場合は、加減算の結果が出た時点で、乗除算もうちきり、次の命令へ進むことにする。また、以下では加減算1個の処理時間をA, 乗除算1個の処理時間をMとする。

4.1 べき乗の計算

x を与え、 $x^2, x^3, \dots, x^m$ のすべてを求めるアルゴリズムを考える。 $m=14$ の例を図7に示す。

【アルゴリズム(a)】

図7(a)に示すように、逐次的に計算していく従来の方法であり、 $x^2 \sim x^{14}$ を求めるのに13ステップかかる。一般に、 $x^2 \sim x^m$ を求めるのには、 $m-1$ ステップを必要とする。

【アルゴリズム(b)】

$$\begin{cases} x^i * x^2 = x^{i+2} & (6) \\ x^{i+2} * x = x^{i+3}, x^{i+2} \div x = x^{i+1} & (7) \end{cases}$$

という計算式に従って、計算を進めていく方法である。FFMにおいては、式(7)の2つの計算を1ステップで実行できるため、従来の計算よりも少ないステップ数で済む。一般に、 $x^2 \sim x^m$ をFFMで求める場合の処理ステップ数は、次式のようになる。

$$\begin{aligned} \text{Step 1. } x * x &= x^2 \\ 2. x^2 * x &= x^3 \\ 3. x^3 * x &= x^4 \\ 4. x^4 * x &= x^5 \\ 5. x^5 * x &= x^6 \\ 6. x^6 * x &= x^7 \\ 7. x^7 * x &= x^8 \\ 8. x^8 * x &= x^9 \\ 9. x^9 * x &= x^{10} \\ 10. x^{10} * x &= x^{11} \\ 11. x^{11} * x &= x^{12} \\ 12. x^{12} * x &= x^{13} \\ 13. x^{13} * x &= x^{14} \end{aligned}$$

ALGORITHM (a)

$$\begin{aligned} \text{Step 1. } x * x &= x^2 \\ 2. x^2 * x^2 &= x^4 \\ 3. x^4 * x &= x^5, x^4 \div x = x^3 \\ 4. x^5 * x^2 &= x^7 \\ 5. x^7 * x &= x^8, x^7 \div x = x^6 \\ 6. x^8 * x^2 &= x^{10} \\ 7. x^{10} * x &= x^{11}, x^{10} \div x = x^9 \\ 8. x^{11} * x^2 &= x^{13} \\ 9. x^{13} * x &= x^{14}, x^{13} \div x = x^{12} \end{aligned}$$

ALGORITHM (b)

$$\begin{aligned} \text{Step 1. } x * x &= x^2 \\ 2. x^2 * x^2 &= x^4 \\ 3. x^4 * x &= x^5, x^4 \div x = x^3 \\ 4. x^5 * x^5 &= x^{10} \\ 5. x^{10} * x &= x^{11}, x^{10} \div x = x^9 \\ 6. x^{10} * x^2 &= x^{12}, x^{10} \div x^2 = x^8 \\ 7. x^{10} * x^3 &= x^{13}, x^{10} \div x^3 = x^7 \\ 8. x^{10} * x^4 &= x^{14}, x^{10} \div x^4 = x^6 \end{aligned}$$

ALGORITHM (c)

図7. Three algorithms for computing $x^2, x^3, \dots, x^{14}$

$$2 \lfloor \frac{m}{3} \rfloor + 1. \quad (8)$$

このように、アルゴリズム (b) に従えば、FFMは、従来の計算機のほぼ $\frac{2}{3}$ のステップ数で $x^2 \sim x^m$ を求めることができる。

【アルゴリズム (c)】

図9は図7の計算の流れを示したものである。ただし、 $x^n \oplus x^m = x^p$ ($\oplus$ は、 $\times$ または $\div$)において、 $n \geq m$ の場合、少なくとも x^n から x^p を求めることができるという意味で、 $n \rightarrow p$ と表示する。図9の(c)に注目すると x^{10} を中心として、両側に計算が広がっていることがわかる。このような点(同図の x^{10} や x^4 のような点)を噴火点と呼ぶことにすると、FFMにおいては、噴火点を中心として対称の位置にある2つの計算を1ステップで実行することができるので、実行ステップ数は減少する。

一般に $x^2 \sim x^m$ を求める場合、どこに噴火点をとるかが、問題となる。これは、図8に示す手続き ERUPT(m) によって求める。

アルゴリズム (c) を使い、 $x^2 \sim x^m$ を FFM で求める場合の処理ステップ数は

$$\left\lfloor \frac{m + \lfloor \log_3(\frac{2}{3}m - 1) \rfloor + 2}{2} \right\rfloor \quad (9)$$

となり、 $\frac{m}{2}$ ステップ強で実行できるが、 $\frac{m}{2}$ 以下

PROCEDURE ERUPT(m);

begin

if $m \leq 1$ then

begin

$x = \lfloor \frac{m-2}{3} \rfloor$;

噴火点 = $m-x$;

$m-2x \sim m$ の範囲で噴火;

$m-2x-1 \rightarrow m-x$;

ERUPT($m-2x-1$)

end

end

図8. 噴火点を求めるアルゴリズム

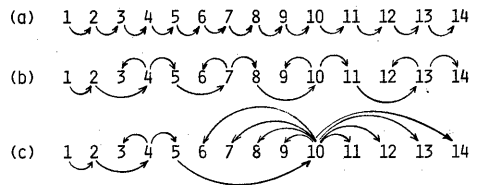


図9. 図7の計算の流れ

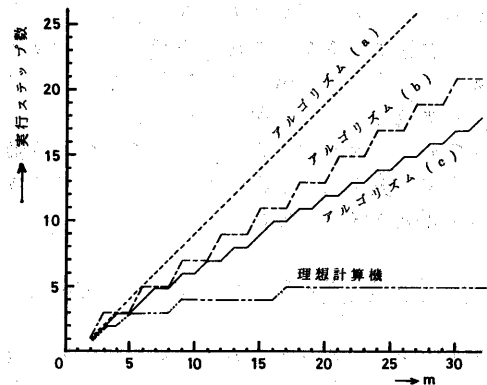


図10. $x^2 \sim x^m$ の計算ステップ数

にはなりえない。

図10は、アルゴリズム (a), (b), (c) に従い、 $x^2 \sim x^m$ を FFM で計算した場合の計算ステップ数である。

なお、 $x^2 \sim x^m$ を理想計算で計算する場合の実行ステップ数は、

$$\lceil \log_2 m \rceil. \quad (10)$$

となる。

4.2 複素数乗除算

図11に、複素数乗算 $(a+bi) \cdot (c+di) = (ac-bd) + (ad+bc)i$ を計算する2つのアルゴリズムを示す。アルゴリズム (a) は、従来の計算方法であり、従来の計算機、FFMともに $4M + 2A$ で計算することができる。(図中で太線で示された計算量は、FFMで実行した場合を表し、() 内は、従来の計算機で実行した場合を表す)。

一方、アルゴリズム (b) を用いると、同じ複

素数計算をするのにもかかわらず、乗算数を1個減らすことができる。さらに、FFMにおいては、S6, S7を1ステップで計算できるため加減算数を1個減少させることができる。従って、アルゴリズム(b)は、FFM向きのアルゴリズムと言うことができる。

同様なアルゴリズムが、複素数除算の場合も存在し、これを図12に示す。

4.3 数値計算例と実行ステップ数

表1にいくつかの数値計算例をFFMで計算する場合の実行ステップ数を示す。表中で[]内は、従来の計算機によって、そのアルゴリズムを実行する場合のステップ数である。ただし表中、要素が複素数のFFM向きアルゴリズムは、図11(b)の方法を複素数乗算に使用したものである。

このように、FFMにおいては、べき乗計算や複素数演算等を使用するような数値計算分野において、従来の計算機よりも少ないステップ数で処理を行うことができる。(ステップ数が減少した箇所も太い文字で示す)。

S1 = a*c
S2 = b*d
S3 = a*d
S4 = b*c
S5 = S1-S2 = ac-bd
S6 = S3+S4 = ad+bc

S1 = a+b
S2 = c+d
S3 = S1*S2 = ac+ad+bc+bd
S4 = a*c
S5 = b*d
S6 = S4-S5 = ac-bd
S7 = S4+S5 = ac+bd
S8 = S3-S7 = ad+bc

4M + 2A
(4M + 2A)

ALGORITHM(a)

図11.(a+bi)*(c+di) = (ac-bd)+(ad+bc)i

S1 = a*c
S2 = b*d
S3 = a*d
S4 = b*c
S5 = c*c
S6 = d*d
S7 = S5+S6 = c²+d²
S8 = S1+S2 = ac+bd
S9 = S8/S7
= (ac+bd)/(c²+d²)
S10 = S4-S3 = bc-ad
S11 = S10/S7
= (bc-ad)/(c²+d²)

8M + 3A
(8M + 3A)

ALGORITHM(a)

3M + 4A
(3M + 5A)

ALGORITHM(b)

S1 = a+b
S2 = c+d
S3 = S1*S2 = ac+ad+bc+bd
S4 = c*c
S5 = d*d
S6 = S4+S5 = c²+d²
S7 = a*d
S8 = b*c
S9 = S8-S7 = bc-ad
S10 = S9/S6
= (bc-ad)/(c²+d²)
S11 = S8+S7 = bc+ad
S12 = S3-S11 = ac+bd
S13 = S12/S6
= (ac+bd)/(c²+d²)

7M + 5A
(7M + 6A)

ALGORITHM(b)

図12.(a+bi)/(c+di) = (ac+bd)/(c²+d²) + ((bc-ad)/(c²+d²))i

表1. 各種計算のFFMによる実行ステップ数

([]内は、従来の計算機によるステップ数)

	計算式	要素が実数		要素が複素数	
		従来のアルゴリズム	FFM向きアルゴリズム	従来のアルゴリズム	FFM向きアルゴリズム
FFT バタフライ演算	$X + W^k Y = X$ $X - W^k Y = X$			4M + 4A [4M + 6A]	3M + 6A [3M + 9A]
ベクトル内積	$a = (a_1, a_2, \dots, a_n)$ $b = (b_1, b_2, \dots, b_n)$ $a \cdot b = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$	nM + (n-1)A [nM + (n-1)A]	無し	4nM + (4n-2)A [4nM + (4n-2)A]	3nM + (6n-2)A [3nM + (7n-2)A]
行列積	$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{bmatrix}$ $= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + \dots + a_{1n}b_{n1} & \dots & a_{11}b_{1n} + a_{12}b_{2n} + \dots + a_{1n}b_{nn} \\ \vdots & \ddots & \vdots \\ a_{m1}b_{11} + a_{m2}b_{21} + \dots + a_{mn}b_{n1} & \dots & a_{m1}b_{1n} + a_{m2}b_{2n} + \dots + a_{mn}b_{nn} \end{bmatrix}$	n ³ M + (n ³ -n ²)A [n ³ M + (n ³ -n ²)A]	無し	4n ³ M + (4n ³ -2n ²)A [4n ³ M + (4n ³ -2n ²)A]	3n ³ M + (6n ³ -2n ²)A [3n ³ M + (7n ³ -2n ²)A]
たたみ込み	$\begin{matrix} a_1 b_1 \\ a_1 b_2 + a_2 b_1 \\ \vdots \\ a_1 b_n + a_2 b_{n-1} + \dots + a_n b_1 \\ \vdots \\ a_1 b_n + a_2 b_{n-1} + \dots + a_n b_1 \\ \vdots \\ a_n b_n \end{matrix}$	n ² M + (n-1) ² A [n ² M + (n-1) ² A]	無し	4n ² M + (4n ² -4n+2)A [4n ² M + (4n ² -4n+2)A]	3n ² M + (6n ² -4n+2)A [3n ² M + (7n ² -4n+2)A]
多項式の計算	$a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n$	(2n-1)M + nA [(2n-1)M + nA]	$\{n+1(n+1 \log_2(2n/3-1)+2)/2\}M + nA$ [(2n-1)M + nA]	(8n-4)M + (6n-2)A [(8n-4)M + (6n-2)A]	(6n-3)M + (10n-4)A [(6n-3)M + (12n-5)A]

5. 結 言

プログラム中の並列性の中で、FFMで実行できるものがどの程度あるかを、並列性を乱数で与えてノード間の結線の組合せを調べることにより評価した。その結果、与えられた並列性があまり高くない場合に、FFM向きの計算が出現しやすく、数値例では最大2~3割程度実行ステップ数の減少が期待できることがわかった。

また、FFM向き計算アルゴリズムを検討したところ、べき乗の計算、複素数乗除算等で、従来の計算機よりも有利なアルゴリズムがあることがわかった。

< 参考文献 >

- (1). 重井, 長谷川, 中村: "Feed Forward 計算機の提案", 昭和55, 情報処理学会第21回全国大会, 2J-5.
- (2). 金井, 長谷川, 中村, 重井: "フィード・フォワード計算機の構造", 昭和55, 信学技報, EC 80-34.
- (3). 山田, 吉岡, 中村, 重井: "フィード・フォワード計算機の構成とその性能評価", 情報理論とその応用研究会, G-2 (昭和56).
- (4). 山田, 吉岡, 中村, 重井: "フィード・フォワード型汎用計算機", 信学論(D), J66-D, 8, pp. 985-992 (昭和58-08).
- (5). 富沢, 山田, 吉岡, 中村, 重井: "フィード・フォワード計算機における複素数計算の実行について", 昭和58, 信学情報・システム部門全大 557.
- (6). A.V Aho, J.E.Hopcroft and J.D.Ullman: "アルゴリズムの設計と解析II", 野崎昭弘, 野下浩平共訳, サイエンス社 (昭和52).
- (7). 坂村 健, 石川千秋: "VLSI コンピュータ・アーキテクチャ", 情報処理, vol.24, No.2, pp.156-175, Feb. 1983.
- (8). C.Berge: "組合せ論の基礎", 野崎昭弘訳, サイエンス社, 昭和48.
- (9). R.W.Hockney and C.R.Jesshope: "Parallel Computers", Adam Hilger Ltd, Bristol, 1981.
- (10). Edmund A.Lamagna: "Fast Computer Algebra", IEEE Computer, vol.15, No.9, pp.43-56, Sept.1982.
- (11). Kung, H.T.: "Putting Inner Loops in VLSI", Lecture Notes for the 16th IBM Computer Science Symposium, pp.55-56, Oct.1982.