社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－SLDM－133（20）
2008／1／17

FIFOを用いて実現するアレイプロセッサのための効率的なデータ入出力機構の提案

野本 裕輔† 佐藤 由香† 宮崎 敏明†

†会津大学大学院コンピュータ理工学研究科
〒965-8580 福島県会津若松市一箕町鶴賀

E-mail: †{m5101209, m5111110, miyazaki}@u-aizu.ac.jp

あらまし 信号処理や画像処理を行なうアレイプロセッサは，扱うデータ量が多いため，効率的にデータの入出力を行なう必要がある．本稿では，FIFOを用いることでデータの入出力を効率的に行なう機構を提案する．入力行列と途中結果を保持するために複数本のFIFOを用意する．それらをタイミングよく切り替え，アレイプロセッサへのデータ転送を行なうことで，アレイプロセッサのボトルネックである外部メモリとの通信をなくし，入出力効率を上げることができる．

キーワード アレイプロセッサ，入出力制御

An effective data I/O mechanism utilizing FIFOs for an array processor

Yuusuke NOMOTO† Yuka SATO† and Toshiaki MIYAZAKI†

† Graduate School of Computer Science and Engineering, The University of Aizu
Tsuruga, Ikki-machi, Aizuwakamatsu-shi, Fukushima, 965-8580 Japan
E-mail: †{m5101209, m5111110, miyazaki}@u-aizu.ac.jp

Abstract Data I/O management is very important to handle huge data effectively in signal and image processing with array processors. In this paper, we propose an effective data I/O architecture utilizing FIFOs tuned to an array processor. Several FIFOs are provided to store initial matrix data and partially processed results. By switching the FIFOs promptly and transferring data from (to) them to (from) the array processor, we can eliminate the data transfer itself between the external memory and the array processor, which often becomes a performance bottle-neck of array processors, and improve the I/O throughputs.

Keyword Array Processor, I/O controller

1. はじめに

データレベルの並列化は，科学計算，信号処理，メディアを扱うアルゴリズムに広く内在している．それらアルゴリズムは，グリッド，ピクセル，ビデオフレーム，音声サンプルなど，連続データに対し，小規模な命令のセットを繰り返す形で実現されていることが多い．また，その並列化は行列で表現されるのが一般的である．よって，行列演算を効率的に扱うことは非常に重要であり，歴史的に見ても，行列演算の為の様々なアルゴリズムや実装手法が提案されてきた[1][2]．また，それら行列演算を行なうアレイプロセッサの１つとして，シストリックアレイ[3]が広く知られている．シストリックアレイは，比較的単純な Processing Element (PE)を，線形状，格子状，六角形状に接続し，並列的に行列演算を行なう．各 PE 同士は単純かつ規則正しく接続されているため，高い集積度，演算処理の単純化が期待できる．

一方，科学計算，信号処理，メディアを扱う演算処理では，扱うデータ量が膨大となる．そのため，データの入出力に要するコストが，処理全体の性能に大きな影響を与えている．よって，アレイプロセッサの処理性能上げることと同時に，データの入出力に要するコストを減らすことも重要となる．

現在，我々は２次元格子状のアレイプロセッサの開発を行なっている[5]．開発しているアレイプロセッサでは，PE に特別な３入力１出力の Fused Operation Unit (FOU)を持つことで，様々な Algebraic Path Problem (APP)を効率良く解くことができる．しかし，PE 内のレジスタファイルに解く問題の全てのデータを抱え込む必要があり，PE 内の演算部と記憶部のバランスが悪い．特に小規模の PE アレイを実現しようとすると，１つの PE が持つべきレジスタファイルが肥大化する．それを防ぐために，外部メモリを設け，途中結果を保存することにより，処理自体は実現可能となるが，データ入出力に大きな時間を要してしまい，アーキテクチャ本来の良さを完全に引き出すことが出来ない．

表 1. Algebraic Path Problem の一覧

S	$\oplus$	$\otimes$	$(*)$	$\bar{0}$	$\bar{1}$	Problem
R	+	×	$a^* = 1/(1-a)$	0	1	Matrix Inversion $(I_n - A)^{-1}$
{0,1}	∨	∧	$a^* = 1$	0	1	Transitive Closure
$R \cup \{+\infty\}$	min	+	$a^* = 0$	$+\infty$	0	All-pairs Shortest Paths
$R \cup \{-\infty\}$	max	+	$a^* = 0$	$-\infty$	0	All-pairs Longest Paths
$R \cup \{+\infty\}$	max	min	$a^* = \infty$	0	$+\infty$	All-pairs Maximum Capacity Paths
$R_{[0,1]}$	max	×	$a^* = 1$	0	1	All-pairs Maximum Reliability Paths
$R \cup \{+\infty\}$	min	max	$a^* = 0$	$+\infty$	0	Minimum Cost Spanning Tree

```
1  for pass = 1 : 3
2    for step = 0 : b -1
3      for all [0 ≤ i, j, k < b] & [(k-j-i) mod b = step]
4      begin
5        c_out ← c_in ⊕ a_in ⊗ a'_in;
6        case(rs):
7        (11): a_out ← c_out ; a'_out ← c_out ;   // i=j=k black element
8        (01): a_out ← c_out ; a'_out ← a'_in ;   // i≠j=k red element
9        (00): a_out ← a_in ; a'_out ← a'_in ;    // (i≠k)&(j≠k) white element
10       (10): a_out ← a_in ; a'_out ← c_out ;    // i=k ≠j blue element
11     end
```

図 1. APP を解く手順

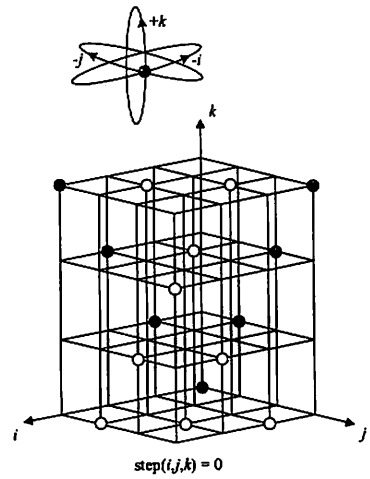

図 2. APP を解くための 3 次元トロイダル・インデックス空間

上記を踏まえ，本稿では，FIFO を用いて実現した効率的なデータ入出力機構を提案する．提案するデータ入出力機構を "*QueueDoRegister*" (external queues realizing pseudo register file)と呼ぶ．データ入出力部に FIFO を置くことは一般的によく行なわれるが，それらはデータのバッファリングや入出力のタイミング調整が主目的である．それに対し，我々が提案する機構はアレイ制御と入出力制御を同時に行なうことを目的としている．本稿では，提案データ入出力機構を我々が提案しているアレイプロセッサに対して適用する場合について議論を進めるが，拡張性や汎用性の面から，提案データ入出力機構は，従来のシストリックアレイを始め，様々なアレイプロセッサに適応できると考える．

本稿の構成は以下の通りである．2 章で，我々が解法対象とする Algebraic Path Problem (APP)について述べた後，提案機構の APP への適用について議論する．3 章で，*QueuDoRegister* アーキテクチャの紹介を行ない，4 章でシミュレーション上での動作検証について議論する．最後に，結論と今後の課題を 5 章で述べる．

2. Algebraic path problem

提案する機構を紹介する前に，開発中のアレイプロセッサの計算対象である Algebraic Path Problem (APP) について簡単に説明する．APP とは，行列の転置，transitive closure, all-pairs shortest paths, minimum cost spanning tree など，よく知られている多くの行列やグラフ問題を解くためのいくつかの解決手法を 1 つに統一した枠組みが APP である．それ故，もし APP を解くための，共通で効果的なメカニズムが提供されれば，実世界に存在する多くの有益な問題を解くことができる．

今，重み付き有向グラフ $G=(V,E,w)$ を考える．ここで，$V=\{0,1,...,n-1\}$ は n 個の頂点の集合，$E \subseteq V \times V$ は辺の集合，$w: E \to S$ は集合 S から得られる値を用いた辺の重み関数である．本関数は，加算 $\oplus: S \times S \to S$，乗算 $\oplus: S \times S \to S$，" closure" と呼ぶ単項演算 $(*): S \to S$ と共に，パス代数 (path algebra) $\langle S, \oplus, \otimes, (*), \bar{0}, \bar{1} \rangle$ を形成する．ここで，定数 $\bar{0}$ と $\bar{1}$ は S に含まれる．このパス代数は閉じた半環 (closed semiring)である．

パス p は，$0 \le t$ かつ $(v_{i-1}, v_i) \in E$ を満たす頂点 $(v_0, v_1, ..., v_{i-1}, v_t)$ の列である．パス p の重みは下記のように定義される：

$$w(p) = w_1 \otimes w_2 \otimes ... \otimes w_t,$$

ただし，w_i は辺 (v_{i-1}, v_i) の重み

APP は，頂点 (i,j) の各ペア間の全ての有効パスの重みの合計を決定することと定義できる．もし，$P(i,j)$ が i から j への全ての有効パスの集合であるならば，APP は，その値を下式のように決定できる．

$$d_{ij} = \oplus \{ w(p) : p \in P(i,j) \}$$

APP についての詳細な議論は本稿の趣旨ではなく，より詳しい議論は，文献[6][7][8]などへ譲る．

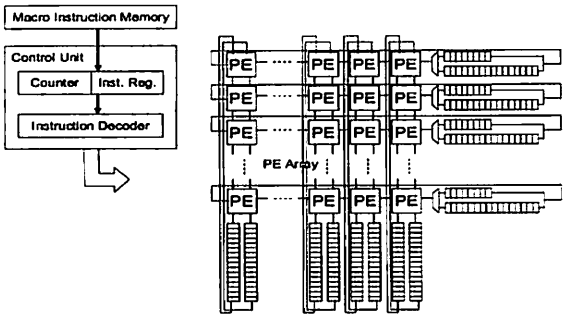

図 3. *QueuDoRegister* アーキテクチャの概観

ここで，$\oplus$，$\otimes$，$(*)$，$\bar{0}$，$\bar{1}$を具体的な演算や値に置き換えた時，解き得る問題を表 1 にまとめる．表 1 は，演算の種別や定数を変更するだけで，同一のアルゴリズムを用いて多くの問題が解けることを示している．言い換えると， APP を解くためのハードウェアが提供できれば，それを多くのアプリケーションへ応用できることを意味している．ハードウェア実装の観点から，これは大きな利点となる．

APP は行列の形で表せる．重み付きグラフ $G=(V,E,w)$ は， $n\times n$ 行列 $A=[a_{ij}]$ を用い，$(i,j)\in E$ のときは $a_{ij}=w(i,j)$，その他の場合は $a_{ij}=0$ として表現できる．本行列表現では，$1\le k\le n$ の範囲で，行列 $A^{(k)}=[a_{ij}{}^{(k)}]$ の連続した計算として APP 問題が解ける．ここで， $a_{ij}{}^{(k)}$ は中間の頂点 v $(1\le v\le k)$を含んだ頂点 i から j までの全ての有効なパスの重みである．最初は $A^{(0)}=A$ で，それ以降は $A^{*}=A^{(n)}$ である．$A^{*}=[a_{ij}{}^{*}]$ は，もし $(i,j)\in P(i,j)$ ならば $a_{ij}{}^{*}=d_{ij}$，その他は $a_{ij}{}^{*}=\bar{0}$ という条件を満たす $n\times n$ 行列である．ここで，与えられた行列の大きさ $n\times n$ が，実際のアレイプロセッサのサイズ $b\times b$ より大きい場合，APP を解くためにブロック化された行列演算が必要となる．APP のためのブロック化行列演算の詳細については文献[2][9][11][12]などを参照されたい．

べき等半環(idempotent semiring) $a=a\oplus a$ に対して，$b\times b$ の大きさの APP は，$b\times b$ の行列演算 $C=C\oplus A$ の特別な場合として表現でき，図 1 の手順で並列計算できる．図 1 において，入力データは，$b\times b$ の重み付き隣接行列 $A=[a_{ik}]=A\oplus I$ である．ここで，I は単位行列，$A'=[a'_{kj}]$ は行列 A のコピー，行列 $C=[c_{ij}]$ は $\bar{0}$ を初期値とする行列である．これは，図 2 で示した 3 次元トロイダル・インデックス空間上で，各頂点における計算と 3 方向へのデータ転送に対応する．図 2 において，端点に位置する頂点は反対側の頂点に接続されているが，簡単のために図上には表示していない．

我々のアレイプロセッサは，同一時刻には全ての PE が同じ命令を実行するという，いわゆる SIMD (Single Instruction Multiple-Data)型の制御方式を基本としている．しかし，図 1 に示した手続きを実行するためには，各 PE が隣接 PE へ転送するデータを個別に変更する必要がある．ここでは，文献[10]と同様な手法を用いて本問題に対処する．すなわち，1 ビットの論理属性を行列 A と A' の各要素に付加する．ここで，行列 A の各要素に付加する論理属性を r とし，行列 A' の各要素に付加する論理属性を s とする．r と s の値は，行列の対角要素へは $r=s=1$ とし，その他の要素へは $r=s=0$ とする．処理中，論理属性は，図 1 に示したように，異なるデータを隣接に送るための制御に用いられる．言い換えると，各 PE から隣接 PE へ出力されるデータは，r と s の値の組み合わせによって自動的に決定される．すなわち，これらの論理属性の導入により，各演算ステップで，それぞれの PE の状態を個別にチェックするための特別な処理が不要となる．

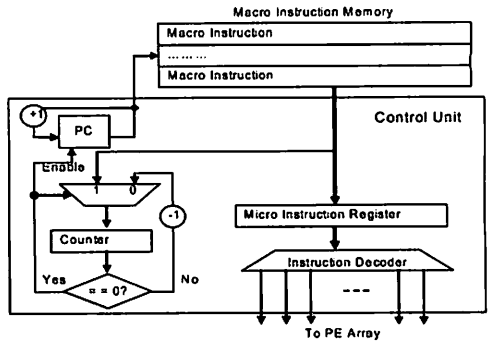

図 4. 制御部の構造

3. アーキテクチャ

APP は，我々が提案した 2 次元アレイプロセッサを用いて効率的に解くことができる[5]．しかし，前述したように，解くべき問題に比べてアレイサイズが小さいと，各 PE のレジスタファイルサイズを大きくしなければならなかった．ここでは，APP 計算アルゴリズム上，レジスタファイルに格納すべきデータを PE アレイの周囲に設けた FIFO に格納することにより，本来の APP 計算アルゴリズムを変更することなく，限られた PE アレイサイズで，大規模な APP を解くためのデータ入出力機構 *QueuDoRegister* アーキテクチャを提案する．*QueuDoRegister* アーキテクチャの全体構成を図 3 に示す．*QueuDoRegister* アーキテクチャは，主に PE アレイ部，その周辺に設けた FIFO，および制御部からなる．PE アレイ部は，FIFO をはさんで，2 次元のトーラスの構造となっている．すなわち，全ての隣接 PE 間および隣接する PE と FIFO は垂直方向と水平方向に接続され，さらに上端，左端に位置する PE は反対端の FIFO と接続している．PE アレイ部の右側に用意した 2 本の FIFO に入力行列 A を，PE アレイ部の下側に用意した 2 本の FIFO にそれぞれ入力行列 B，

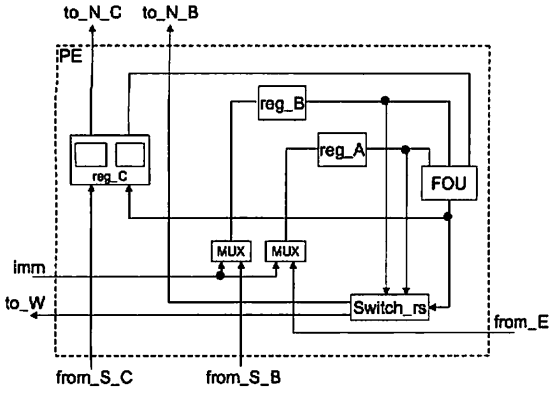

図 5. processing element の構造

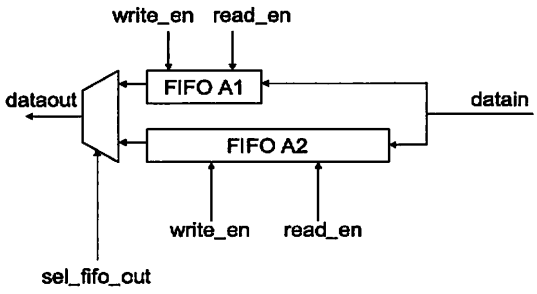

図 6. FIFO_A の構造

C を割り当てる．右側の FIFO を FIFO_A とし，下側の２本の FIFO をそれぞれ FIFO_B，FIFO_C とする．FIFO_A，FIFO_B にバッファリングされたデータは，順次各 FIFO から送り出され，従来のシストリックアレイ動作と同様に，演算と隣接 PE へのデータ転送を交互に行なうことによりデータ処理が実現される．さらに左端または上端に到達したデータは再び各 FIFO にバッファリングされる．この時の FIFO_A の詳細動作は 3.1.章で述べる． FIFO_C は，APP を解くときに用いる $C = A \otimes B \oplus C$ の演算結果を保持するための FIFO である．各 PE 内のレジスタファイルの初期値は，FIFO を通じて順次ロードするか，即値を扱う"LOAD"命令により，*immediate* フィールドに指定した値を，1つまたは全ての PE へ同時にロードすることも可能である．

制御部は，入力された命令に従って，PE アレイ部と各 FIFO の制御を行なう．本アレイプロセッサは，2章で述べた APP を解くときに論理属性 r と s により各 PE から隣接 PE へ転送すべきデータを自動的に決定することを除いて，基本的に SIMD 型で制御される．図4 に制御部のブロック図を示す．命令レジスタとデコーダに加え，フェッチした命令を発行すべき回数を管理するカウンタを設けてある．例えば，同じ命令を連続して 5 回発行したい場合，値 5 がカウンタにセットされる．行列演算では，同一命令を複数回繰り返すことが多く，本機構は非常に有用である．

$$A = \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix}$$

FIFO_A $\cdots A_{00}A_{01}A_{02}A_{10}A_{11}A_{12}A_{20}A_{21}A_{22}$
FIFO_B $\cdots B_{00}B_{10}B_{20}B_{01}B_{11}B_{21}B_{02}B_{12}B_{22}$

図 7. FIFO_A，FIFO_B に入れ込む行列の順番

PE の内部構成を図 5 に示す，3 入力 1 出力の FOU，行列 A，B，C の要素データを格納するレジスタ，幾つかのスイッチから構成されている．３つの入力データを PE 内の FOU で演算し，隣接 PE へ演算結果を転送する点は従来提案したアレイプロセッサと同じだが，データ入出力の効率を上げるため，PE からレジスタファイルを取り除き，外部の FIFO を擬似的なレジスタファイルとしたことにより，従来の PE に比べ回路が簡単化できている．各信号線，モジュールの意味は以下の通りである．名前に from が付いている信号線は隣接 PE もしくは FIFO からの入力であり，to が付いている信号線は隣接 PE もしくは FIFO への出力である．また，N，E，S，W はそれぞれ，North（図上，上側），East（右側），South（下側），West（左側）を意味する．imm は制御部からの即値の入力である．Switch_rs は PE の出力すべきデータを論理属性によって決定するためのスイッチである．行列 A，B に対応するレジスタは１つのデータを保持し，行列 C に対応するレジスタは読み出し２ポート，書き込み２ポート，２ワードを保持できるレジスタからなる．FOU は，表 1 に示したスカラ **fma** 演算を行なうことができる．この FOU の最もユニークな特徴は，一般的な積和演算に加えて，min/max や論理演算を含む複合演算も扱うことができる点である．FOU の演算種別決定は，通常の ALU の制御と同様に，制御部からの機能選択の信号によって行なわれる．

3.1. FIFO 長

FIFO_A の詳細な構造を図 6 に示す．２本の FIFO，セレクタで構成される．write_en は FIFO への書き込み許可，read_en は FIFO からの読み出し許可のための制御信号である．また，sel_fifo_out は FIFO_A の出力を決定するためのセレクト信号である．

上側の FIFO_A1 は，入力正方行列が，$n \times n$ とすると，n 個のデータを保持できる長さを持ち，下側の FIFO_A2 は解く問題の大きさ n^2 / b 個のデータを保持

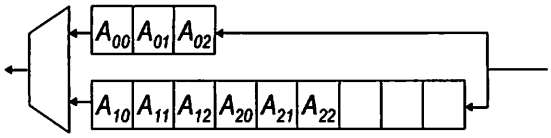

図 8. $A_{00}A_{01}A_{02}$ と $B_{00}B_{10}B_{20}$ の演算後の FIFO_A の状態

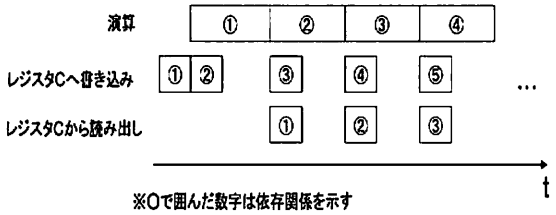

図 9. 演算，レジスタ C への書き込み，レジスタ C からの読み出しの時間的依存関係

できる長さを持つ．ただし，PE アレイ部が b x b 個の PE で構成される場合である．例えば 4 x 4 の大きさの PE アレイ部に 12 x 12 の大きさの問題を解かせる場合，最低必要な FIFO_A1 の長さは 12，FIFO_A2 の長さは 36 となる．また，FIFO_B および FIFO_C の長さは，FIFO_A2 と同じく n^2/b の長さが必要になるので，36 となる．

3.2. 処理の流れ

提案アーキテクチャでの処理の流れを，4 x 4PE アレイ部に 12 x 12 の 2 つの正方行列の乗算を行なう場合を例として説明する．

まず，各 FIFO に入力データを入れ込む．解く問題の大きさが PE アレイ部の大きさより大きいため，入力する行列を部分行列に分けて FIFO に入れ込む必要がある．FIFO_A，FIFO_B に入れ込む部分行列の順番を図 7 に示す．A_{XX} は 4 x 4 の部分行列であり．また，FIFO_A に関しては下側の FIFO_A2 に全てのデータを入れ込み，FIFO_C は演算結果を保持するために用いるため，初期値として 0 を入れ込む．

各 FIFO へのデータの入れ込みが完了後，演算を開始する．初めに，FIFO_A2 から $A_{00}A_{01}A_{02}$，FIFO_B から $B_{00}B_{10}B_{20}$ を取り出し，シストリックアレイと同様に，演算と隣接 PE へのデータ転送（トーラス接続しているので実際派アレイプロセッサ上のデータの回転移動となる．）を交互に繰り返す演算（以後，"演算と回転"と呼ぶ）を行なう．その結果，部分行列 C_{00} が求まる．そして，$B_{00}B_{10}B_{20}$ をそのまま FIFO_B にバッファリングし，$A_{00}A_{01}A_{02}$ を FIFO_A2 ではなく FIFO_A1 にバッファリングする．この時の状態を図 8 に示す．

次に，FIFO_A1 から $A_{00}A_{01}A_{02}$，FIFO_B から

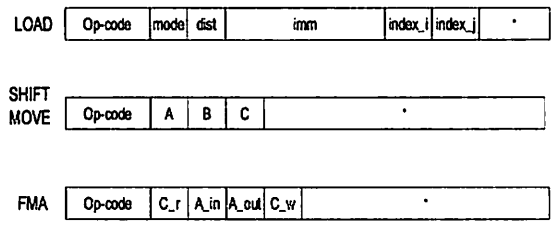

図 10. マイクロ命令

$B_{01}B_{11}B_{21}$ を取り出し，"演算と回転"を行ない，部分行列 C_{01} を求める．この場合も $B_{01}B_{11}B_{21}$ を FIFO_B にバッファリングし，$A_{00}A_{01}A_{02}$ を FIFO_A1 にバッファする．FIFO_A にバッファリングされた状態は図 8 に示したものと同じである．

同様に，FIFO_A1 から $A_{00}A_{01}A_{02}$，FIFO_B から $B_{02}B_{12}B_{22}$ を取り出し演算を行なうが，この時 $A_{00}A_{01}A_{02}$ は FIFO_A2 にバッファリングされる．つまり，今までの一連の演算の流れは，FIFO_B にバッファリングされたデータを 1 周させる間，$A_{00}A_{01}A_{02}$ を 3 周させ，その後 FIFO_A2 にバッファリングしている．これは，行列 A の n/b 個の部分行列を n/b 回，回転させるということである．

上述した一連の処理全体を，さらに 2 回繰り返すことで，全ての部分行列の積を求めることができる．つまり，上述した n/b 回の"演算と回転"を伴う一連の処理を計 3 回繰り返すことで全ての演算処理を終了することができる．このように一度データを FIFO に溜め込んでしまえば，外部メモリとデータのやりとりをせずに，全ての演算処理を終了させることができるため，本提案機構はデータの入出力の負荷を減らすことに貢献する．演算終了後，FIFO_A，FIFO_B にバッファリングされているデータは初期状態の配置と等しい．これは，APP を解く際に役立つ．例えば All-pairs Shortest Path を求める場合，乗算を求める際に行なった処理全体と同様な処理を，3 回繰り返す必要がある．1 回目の処理が終わった時点でデータが初期状態の配置に戻っていれば，2 回目の演算をすぐに始めることができるため，同一処理を何回も繰り返す演算に対して有利となる．

次に，PE 内のレジスタ C に関する処理を示す．1 つのデータだけ保持するレジスタ A，B に対して，レジスタ C は 2 つのデータを保持することができ，また読み出し書き込みともに 2 つポートある．これは，演算，レジスタ C の初期化，レジスタ C からの演算結果の読み出しを全て並列に行なうためである．図 9 に演算，レジスタ C への書き込み，レジスタ C からの読み出しの時間的依存関係を示す．レジスタ C の 2 つのレジスタをそれぞれ C1，C2 としたとき，図に示した奇数番号では C1 を使い，偶数番号では C2 を使用する．

まずレジスタCへの書き込みの①のとき，C1の初期化を行なう．初期化が終わった後，C1を使ってすぐに演算を開始する．それと同時にC2を次の演算に使用するデータを書き込む．①の演算が終わった後すぐに，C2のデータを使って演算を開始する．それと同時にC1を次の演算で使用するデータを書き込み，同時に演算結果が押し出されFIFOに書き戻される．この操作を繰り返すことで，データCの入出力に要する時間を演算時間に隠すことができ，全体の処理時間の向上につなげることができる．

4. 動作検証

提案したアーキテクチャが正しく動作することを確認するために，Verilog-HDLを用いてRTL記述し，動作検証を行なった．検証には，Cadence社のVerilog-XLシミュレータを使用した．ここでは暫定的に，図10に示すマイクロ命令を使用した．LOAD命令は，即値をPE内のレジスタに書き込む命令である．1つのPEに書き込むか全てのPEに書き込むかを“mode”フィールドで決定し，“dist”フィールドはA，B，Cのどのレジスタに書き込むかを指定する．“imm”フィールドは即値である．また，書き込むPEのインデックスを指定するために“index_i”，“index_j”フィールドを設けた．SHIFT，MOVE命令はそれぞれ，データをFIFOに入れ込む命令，FIFOからPEアレイ部にデータを読み出す命令である．“A”，“B”，“C”フィールドはFIFOの指定に用いる．FMA命令はFOUを使った演算を行なう命令である．“C_r”，“C_w”はそれぞれ，FIFO_Cからの読み出し許可，FIFO_Cへの書き込み許可のためのフィールドである．また，“A_in”，“A_out”はそれぞれ，FIFO_Aのどちらの FIFOへ書き込むかを指定するフィールド，FIFO_Aのどちらの FIFOから読み出すかを指定するフィールドである．

検証は行列の乗算，All-pairs shortest path problem，2次元画像のDCTを対象に行なった．入力とする行列データ，グラフ，画像はいくつかの大きさのものを用意し，ソフトウェアでシミュレーションを行なった結果と比較を行なった．その結果，我々が提案したアーキテクチャは，問題の大きさに依存せず正しく動作することを確認した．また，データをFIFOに入れ込む時とFIFOからデータを取り出す時以外は，外部メモリとのデータ通信を行なっていないことから，入力されたデータを効率よく使いまわすことができている．

5. まとめ

本稿では，FIFOを用いて実現した効率的なデータ入出力機構を備えたアレイプロセッサを提案した．また，提案した機構を我々が現在開発しているアレイプロセッサに組み込んだものをRTL記述し，シミュレーションによって，正しく動作することを確認した．提案するアーキテクチャが．今後，提案アーキテクチャをFPGAに実装し，さらに詳細な評価を行なっていく予定である．

謝　辞

本研究を進めるにあたり，ご指導，議論いただいた会津大学 Stanislav G. Sedukhin 教授，大塚学氏，佐藤崇信氏に心より感謝いたします．

文　献

[1] S. Y. Kung, “VLSI Array Processors,” Prentice Hall, 1988

[2] G. Fox, S. Otto, and A. Hey, “Matrix Algorithms on a Hypercube in Matrix Multiplication,” Parallel Computing, Vol. 4, pp.17–31, 1987.

[3] H. T. Kung and C. E. Leiserson, “Systolic Arrays for VLSI,” in Introduction to VLSI Systems, C. A. Mead and L. A. Conway. Reading, MA:Addison-Wesley, 1980, Sec. 8.3.

[4] D. I. Moldovan and J. A. B. Fortes: “Partitioning and mapping algorithms into fixed size systolic arrays”, IEEE Trans. on Computers C-35 (1986) 1, pp.1-12

[5] Y. Sato, Y. Nomoto, T. Miyazaki, and S. G. Sedukhin, “2D Array Processor Featuring Ternary Input Fused Operations,” IPSJ SIG Technical Reports, 2007-SLDM-131, pp.7-12, 2007

[6] M. Penner, J.-S. Park, and V.K. Prasanna, Optimizing graph algorithms for improved cache performance,” IEEE Trans. on Parallel and Distributed Systems, Vol. 15, pp.769–782, 2004.

[7] D.J. Lehmann, “Algebraic structures for transitive closure,” Theoretical Computer Science, Vol. 4, pp.59–76, 1977.

[8] G. Rote, "A systolic array algorithm for the algebraic path problem (Shortest Paths; Matrix Inversion)" *Computing*, 34 (1985) pp.191–219

[9] F. J. Nunez and M. Valero, “A Block Algorithm for the Algebraic Path Problem and its Execution on a Systolic Array,” Proc. the International Conference on Systolic Arrays, pp.265-274, 1988.

[10] L.J. Guibas, H.T. Kung, and C.D. Thompson, “Direct VLSI implementation of combinatorial algorithms,” Proc. Conference on VLSI: Architecture, Design, Fabrication, CalTech, pp.509-525, Jan. 1979.

[11] C.H. Sequin, “Double twisted torus networks for VLSI processor arrays,” Proc. the 8th Annual Symposium on Computer Architecture, Minneapolis, Minnesota, USA, 1981.

[12] G. Griem and L. Oliker, “Transitive closure on the Imagine stream processor,” Proc. the 5th Workshop on Media and Stream Processors, 2003.