2007－SLDM－131 (2)
2007／10／26

3入力演算器を搭載した2Dアレイプロセッサの提案

佐藤 由香† 野本 裕輔† 宮崎 敏明† Stanislav G. Sedukhin†

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

あらまし 行列演算は多くの有用なアルゴリズムに頻繁に必要とされ，その高速化は重要である．本稿では，行列演算に向いた新しいアレイプロセッサを提案する．各プロセッシングエレメントは，3入力1出力の演算器を持ち，積和演算のみならず，大小比較演算やブール演算を含む複合演算を1ステップで実行する．本機構により，各種のグラフ理論に基づく問題を一般化したAPP (Algebraic Path Problem) を効率的に解くことができる．本論文では，行列同士の演算をまとめた後，提案されたアーキテクチャの概要を述べ，FPGA によるプロトタイプの実装結果を示す．

キーワード アレイプロセッサ，APP，FPGA，FOU

2D Array Processor Featuring Ternary Input Fused Operations

Yuka SATO† Yuusuke NOMOTO† Toshiaki MIYAZAKI† and Stanislav G. SEDUKHIN†

† Graduate School of Computer Science and Engineering, The University of Aizu

Tsuruga, Ikki-machi, Aizuwakamatsu-shi, Fukushima, 965-8580 Japan

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

Abstract The speedup of matrix-matrix operations is very important for many applications. We propose a new array processor architecture suitable for matrix-matrix operations. Each processing element has a functional unit performing three input operations such as fused min/max and Boolean operations, and an ordinary fused multiply-add operation. Using these mechanisms, we can commonly solve the algebraic path problem (APP), which is a feature of many useful applications. In this paper, after summarizing the matrix-matrix operations, the array processor architecture is proposed. Then, we show the implementation using an FPGA with some experimental results.

Keyword Array Processor, APP, FPGA, FOU

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ことが知られている．それらのアルゴリズムは，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，小規模な命令セットを繰り返す形で実現されていることが多く，その並列化は行列を用いて表現す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]．近年，Intel Itanium2，IBM PowerPC G4，Sony/Toshiba/IBM CELL processor などの商用プロセッサでは，fused multiply-add や fused multiply-accumulation (**fma**)と呼ばれる特別な演算器が，通常の演算に用いるALU (Arithmetic Logic Unit)とは別に用意され，行列演算の高速処理に貢献している．たとえば，スカラ **fma** として以下のような演算を行なうものがある．

fma(a, b, c): $d = \pm a\ (\times)\ b\ (\pm)\ c$,
ただし，a, b, c, d はスカラデータ

本演算は1つの三項演算で，かつ分割することができないアトミック演算である．**fma** は有用な演算器だが，行列データを扱うためには，**fma** 演算を何度も繰り返す必要がある．その繰り返しを減らす一構成として，2次元アレイプロセッサが考えられる．アレイを構成する各PE (Processing Element)に上記 **fma** 演算を搭載した場合，下記に示す行列 **FMA** 演算が可能となる．

FMA(A, B, C): $D = \pm A\ (\times)\ B\ (\pm)\ C$,
ただし， A, B, C, D 正方行列

ここで，行列 **FMA** は，LINPACK[3]で核となるBLAS (Basic Linear Algebra Subprogram) レベル3の主要部分である．LINPACK は，多くの複雑な科学計算や工学技術のために広く用いられている．DFT (Discrete Fourier Transform) や DCT (Discrete Cosine Transform) のような，広く知られている多次元離散変換も，行列同士の乗算の組み合わせとして表現できる．本稿では，*"RapidMatriX"* (Rapid Matrix eXtension)と名付けた行列 **FMA** 演算を高速化するためのアレイプロセッサのアーキテクチャを提案する．本アーキテクチャは，比較的単純な PE を行列状に接続する意味では，従来のアレイプロセッサと同様であるが，各PEがFOU (Fused Operation Unit)とよぶ特別な3入力演算器持つところに最大の特徴である．FOU は，積和演算など，従来のスカラ **fma** 演算だけでなく，min/max などを含む多くの3入力1出力の演算が可能である．これにより，同一行列構造，かつ同一データ操作で，異なる問題を解くことを可能にする．

我々は，*RapidMatriX* アーキテクチャの初期実装手段として，FPGA を用いることにした．FPGA は近年HPC (High-Performance Computing)の分野でも注目され，FPGA を再構成可能なアクセレータやコプロセッ

表 1. Algebraic Path Problems

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	+	$a^* = 0$	+∞	0	All-pairs Shortest Paths
$R \cup \{-\infty\}$	max	+	$a^* = 0$	-∞	0	All-pairs Longest Paths
$R \cup \{+\infty\}$	max	min	$a^* = \infty$	0	+∞	All-pairs Maximum Capacity Paths
$R_{[0,1]}$	max	×	$a^* = 1$	0	1	All-pairs Maximum Reliability Paths
$R \cup \{+\infty\}$	min	max	$a^* = 0$	+∞	0	Minimum Cost Spanning Tree

サとして使用する商用 HPC マシンも存在する[4][5][6]. *RapidMatriX* は行列操作に関して高い柔軟性を持つ一方，ホストプロセッサが周辺機器やデータの入出力を扱う必要がある．上述した商用マシンは，ホスト-FPGA インタフェースを実現しており，実装が比較的容易であることが，初期プラットフォームとして FPGA を選んだ理由である．

本稿の構成は以下の通りである．2 章で，APP (Algebraic Path Problem)について述べた後，提案アーキテクチャの APP への適用について議論する．3 章で，*RapidMatriX* アーキテクチャの紹介を行ない，4 章で FPGA 実装に関して議論する．5 章で，評価結果と幾つかの実装に対する試みを示す．最後に，結論と今後の課題を 6 章で述べる．

2. Algebraic path problem

提案するアーキテクチャを紹介する前に，我々が目標の 1 つとする APP を示し，我々のアレイプロセッサでどのように APP を解くかを示す．行列の転置，transitive closure, all-pairs shortest paths, minimum cost spanning tree など，よく知られた行列問題やグラフ問題を解くための幾つかの解決手法を 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できれば，実世界に存在する多くの有益な問題を解くことが可能となる．

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 $\otimes: S \times S \to S$，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 $(v_0, v_1, ..., v_{t-1}, v_t)$ 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，

```
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

より詳しい議論は，文献[17][18]などへ譲る．

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エアが提供できれば，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．ハードウエア実装の観点から，これは大きな利点である．

APP は行列を用いて表現できる．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 $a_{ij} = \bar{0}$ 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 $a_{ij}^* = 0$ 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[7][14][15][16]などを参照されたい．一般に，ブロック化されたパス代数のアルゴリズムは，Canon アルゴリズム[19]と類似のアルゴリズムを用い，$b \times b$ の 2 次元トーラス状のアレイプロセッサを用いて b ステップで解くことができる．

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 $C = C \oplus A \otimes A'$ 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．これは，図 2 で示した 3 次元トロイダル・インデックス空間で，各頂点での計算と 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

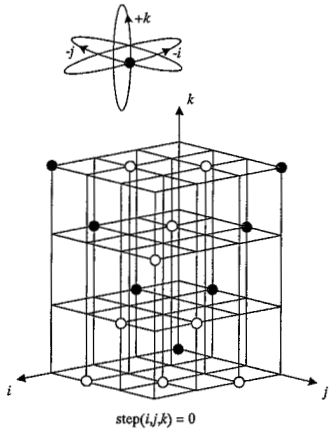


図 2. APP を解くための 3 次元トロイダル・インデックス空間

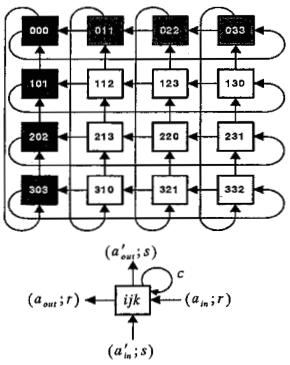


図 3. 図 2 を k 軸方向へ射影して得られた 2 次元アレイプロセッサ.

図上には表示していない．図 1 の手順に於いて，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．図 2 のインデックス空間に於いて，要素 A，A'，Cの初期値は時間ステップ関数 $step(i,j,k):=(k-j-i)\bmod b$ で求まる頂点に予め配置される．ここで，$i,j,k\in[0,b-1]$である．インデックスに関連付けられたデータ要素は，3 次元インデックス空間をステップ毎に移動する．図 2 でマークされている頂点は，$step(i,j,k)=0$で移動すべき要素を示し，$a_{ik}=a_{in}$，$a'_{kj}=a'_{in}$，$c_{ij}=c_{in}=\bar{0}$ に初期化される．各 $step(i,j,k)=0,1,\ldots,b-1$に於いて，各頂点 $(i,j,k)^T$ は，スカラ **fma** 演算の後，隣接頂点へデータを同期して転送する．各軸方向にトーラス接続しているため，実際のデータ移動は回転移動となる．この"演算と回転"が b ステップ繰り返された後，時間ステップ関数は

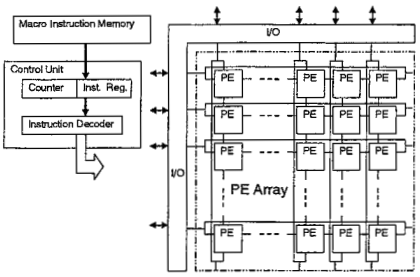


図 4. RapidMatriX アーキテクチャの概要

$step(i,j,k)=0$に再び戻り，更新された A，A'，C'は最初のインデックスと同じ頂点に位置していることになる．すなわち，図 1 の手順に於いて 2 回目と 3 回目のループ処理が，そのまま実行可能となる．

基本的に，この "演算と回転" という動作は，SIMD (Single Instruction Multiple-Data)型の 1 つの命令によって実現することができる．しかし，図 1 に示した手続きを実行するためには，別の機構が必要となる．ここでは，文献[13]と同様に論理的属性 r と s を，行列 A と A'の各要素にそれぞれ付加する．r と sの値は，行列の対角要素に対して $r=s=1$とし，その他の要素では $r=s=0$とする．処理中，本論理的属性は，図 1 に示したように，3 次元のインデックス空間中で動作する各頂点 $(i,j,k)^T$で異なるデータを隣接に送る制御をするために用いる．言い換えると，各頂点から隣接頂点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，これらの論理的属性の導入により，演算の各ステップで各頂点の状態をチェックするための特別な処理が不要となる．

図 1 で示した APP を解くための手順では，明らかに **fma** 演算が高速化のためのカギとなっている．また，図 1 から分かるように APP アルゴリズムを完全に解くには，3 回のパス（ループ）が必要であり，$3b$ステップの "演算と回転" を行なう必要がある．各ステップでb^2の "演算と回転" が行なわれるので，全体では $3b^3$ステップの処理が，APP アルゴリズムを解くために必要となる．図 2 で示したインデックス空間は，1 つの軸に着目し，$b\times b$の 2 次元アレイプロセッサ上に射影可能である．図 3 は，k 軸方向に投影して得られる 2 次元アレイプロセッサの例である．本射影では，行列 Cの (i,j)番目の要素は，対応する (i,j)番目の PE に格納され更新される．本稿で議論するアレイプロセッサは，図 3 を基本として考えることにする．勿論，本アレイプロセッサで，上述した APP だけでなく，行列同士の乗算のような，一般的な行列演算を処理することは容易である．

3. アーキテクチャ

図 4 に *RapidMatriX* アーキテクチャの概要を示す．*RapidMatriX* は主に PE アレイ部と制御部から構成され，PE アレイ部は 2 次元のトーラス構造となっている．すなわち，全ての隣接 PE は垂直方向と水平方向に接続され，さらに上下，左右の端に位置する PE は反対端

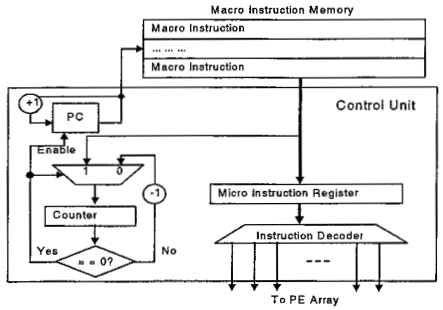


図 5. 制御部の構造

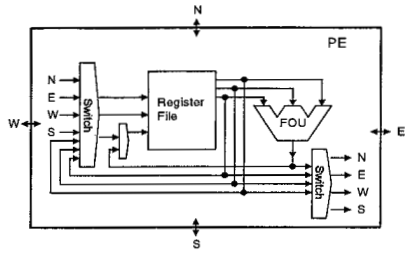


図 6. processing element の概要

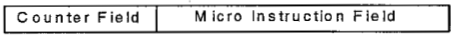


図 7. マクロ命令

の PE と互いに接続している．PE の左端と上端に I/O ポートを設けることによって，左端と上端にある PE へ外部のデータを直接入力することができる．その他の PE へデータを入力するには，隣接ＰＥ内のレジスタ同士を接続しＰＥアレイ部全体でシフトレジスタを構成し，垂直方向または水平方向の隣接接続を用いて，$b \times b$の行列をbステップで PE へ入力する．左端と上端の I/O ポートを同時に使うことで，2 つの$b \times b$行列をbステップでロードすることができる．さらに，即値を扱う"LOAD"命令を用意している．この命令によって，"LOAD"命令の *immediate* フィールドに指定された値を，1 つの PE へ入力することができる．

制御部は，入力された命令に従って，ＰＥアレイ部の制御を行なう．我々のアレイプロセッサは，2 章で示した APP を解くときに論理的属性によって各 PE から隣接 PE へ出力するデータを自動決定することを除いては，基本的に SIMD 型で制御される．言い換えれば，各命令は全ての PE に発行され，各 PE 内のデータパスを同じ方法で制御する．図 5 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，フェッチされた命令の発行すべき回数を管理するための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．この機構は，行列演算を扱うとき，非常に有用である．

デコードされた信号は全ての PE へ直接配られる．これは配線遅延を生み，拡張性を損ねる可能性がある．しかし，各 PE にローカルな制御ロジックを設ける必要がなくなり，行列部全体の回路規模を小さくすることができる．また，我々の目標は，2 次元空間内に配置する PE の数を多くする一方，動作速度を落し，全体の消費電力量を小さく抑えることにある．そのため，たとえ制御信号を配信する距離が長くなったとしても，そのことが致命的になることはないと考え，本方式を採用した．

PE は図 6 に示した通り，3 入力 1 出力の FOU，3 入力 3 出力のレジスタファイル(*reg-file*)，そしていくつかのスイッチから構成されている．レジスタファイルの各出力ポートは，直接 FOU の入力ポートに接続されている．すなわち，レジスタファイルと FOU を互いに接続するのに，スイッチまたはセレクタを必要としない．これは，レジスタファイルから FOU へのパス遅延を短くできるだけでなく，回路規模の削減にも有効である．スイッチを制御することにより，隣接 PE 同士の柔軟な接続を可能とする．FOU は，表 1 に示したスカラ **fma** 演算を行なうことができる．本 FOU の最もユニークな点は，一般的な乗算器と加算器を結合した **fma** 演算に加え，大小比較や論理演算を含む複合演算を扱うことができる点である．FOU の演算種別の決定は，一般の 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. 命令セット

本アーキテクチャでは，マイクロ命令とマクロ命令という 2 つの命令のタイプがある．前者の命令が各 PE を直接制御するために使用され，後者の命令は図 7 に示したようにカウンタ部とマイクロ命令から成る．マイクロ命令は，"Load"，"Move"，"FOU"の 3 つの基本的な操作をサポートしている．それぞれ，即値データの入力，隣接する PE 間のデータ転送，FOU の演算と隣接する PE 間のデータ転送を行なう命令である．すなわち，アセンブリ言語が提供されれば，比較的容易にアレイプロセッサの制御を行なうことができる．さらに複雑なプログラムを作るためには，MATLAB[8] のような行列演算を扱うことを得意とするツールをフロントエンドとしたソフトウエア開発環境の実現を計画している．

4. FPGA 実装

3 章で述べたアーキテクチャの実現可能性を実証するために，プロトタイプを FPGA 上に実装した．使用した FPGA ボードは GiDEL 社製 PROCStar II FPGA board であり，PCI-X インタフェースを介して，ホスト PC 上で動作するソフトウエアとデータ送受ができる．本 FPGA ボードは Altera Stratix II EP2S60 を搭載している．Stratix II EP2S60 は，72 個の 18-bit × 18-bit 組込み乗算器と，"M512"という 329,512-bit の組込み RAM ブロックを有している．実装したプロトタイプは 16-bit 整数値と論理演算をサポートしているが，浮動小数点演算はサポートしていない．FOU の乗算には，FPGA の組込み乗算器を使用した．レジスタファイルを実装するために，図 8 で示した 3 ブロック構造のレジスタファイルを導入し，各ブロックは FPGA 内の"M512" RAM にそれぞれ割り当てた．行列同士の演

表 2. FPGA 実装結果

対象 FPGA は Altera Stratix II EP2S60.

ブロック名	LC 組合せ	LC レジスタ	18x18 乗算器.**	ALUT 合計	%
制御部	515	54	0	607	1.25
PE 行列 (64PEs*+I/O)	35,478	256	64	35,734	73.9
命令メモリ	217	640	0	872	1.8
合計	**36,220**	**950**	**64**	**37,213**	**76.9**
*) 1 PE	509	0	1	567	1.1
インタフェース	248	445	0	695	1.4

**)18x18 mult = 18*18 bit 組み込み乗算器

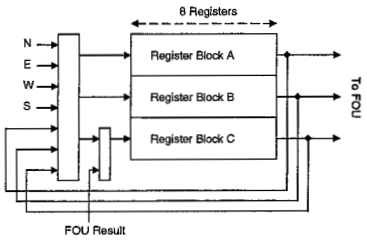


図 8. レジスタファイル実装

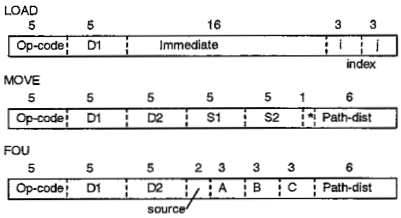


図 9. マイクロ命令の形式. 各命令は 32 ビット長.

算において，1 つの行列内の要素を 1 つのメモリブロックに格納するようにすれば，行列演算処理の間，ブロック間のデータ転送は基本的に必要ない. そのため，速度面で特に問題になることはなく，FPGA 内のレジスタの使用する必要がないため，本レジスタファイル構成を採用することとした．表 2 はプロトタイプを FPGA 上に実現したときの FPGA 利用率を示している. PE アレイ部が FPGA の 77%を占めていることがわかる．また，"interface circuit" は GiDEL ツールが自動生成した PCI インタフェースである．本システムは 36.6MHz で動作しており，ピーク性能値は 4.68GOPS (64 PEs × 2 operations × 36.6MHz)である．今回は，提案アーキテクチャの実現可能性を確認に重点を置いたため，単純な制御シーケンスを採用した. そのため，パイプラインの導入など性能面での改良の余地は充分残っている.

3.2. マクロ命令とマイクロ命令

現在実装しているマクロ命令の命令長は 40 ビット（8 ビット:カウンタ部，32 ビット: マイクロ命令部）である．マクロ命令が制御部に格納されると，カウンタ部の値はカウンタレジスタに，マイクロ命令部は命令レジスタに読み込まれる．図 9 にマイクロ命令の詳細を示す. 現在実装されている命令の数は 16 種類であり，それらは 3 つの形式（"Load", "Move", "FOU"）に分類される．"Load"命令は *immediate* フィールドに書いた値をインデックス i と j で定義された PEij の *D1* レジスタに書き込むことができる．"Move"命令を使うことによって，レジスタに格納されている 2 つのデータを 2 つの隣接 PE のレジスタに同時に転送することができる．"Move"命令の *D1, D2* は隣接 PE から転送されてきたデータを格納するレジスタを示しており，*S1, S2* は PE から隣接 PE へ転送するデータが格納されているレジスタを示している．また，データ転送の方向は *Path-dist* によって指示される．"FOU"命令は計算とデータ転送を行う最も良く使用される命令である．*A, B, C* 部は FOU への入力データのレジスタを指示する．また，*source* 部で転送するデータを指示すると，入力 *A, B, C* のうち 2 つのデータを同時に 2 つの隣接 PE に転送することができる．他の命令部位は"Move"命令で説明したのと同じ意味である．また，FOU 演算は *Op-code* によって定義される．すなわち，異なる FOU 演算はそれぞれ個別の *Op-code* を有する.

マクロ命令はその先頭に 8bit のカウンタ部を付加した形式であり，マイクロ命令それ自身はユーザーレベルのプログラミングでは使われない．よって，我々はマクロアセンブラ言語と専用のマクロアセンブラソフトウエアツールのみを現在提供している.

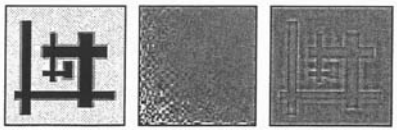
4. 評価

FMA 演算の簡単な例として，DCT と HPF (High Pass Filter) と IDCT (inverse Discrete Cosine Transform) を適用して 2 次元画像のエッジ強調を行った．2 次元の DCT (IDCT)は，下記の様に，通常 3 つの行列の乗算で表すことができる.

$$Y = C \cdot X \cdot C^T \quad (X = C^T \cdot Y \cdot C)$$

ここで C はコサイン係数行列，C^T は C の転置行列である．図 10 は計算結果の例である．画像のサイズは 64 ×64 ピクセルである．今回実装したプロトタイプシステムは 16-bit の整数演算のみサポートしているため，入力画像データと係数行列はオーバフローを避けるためスケーリングした．データの丸めにより計算精度は良くないが，境界強調されていることが確認できる．また，我々が提案したシステムとアルゴリズムを 8 つの頂点を持つグラフに適用し，APP のひとつである All-pairs shortest path problem が正しく解けることを確認した．さらに，交換，置換，転置，複製，約分などの基本的なデータ操作が我々のアレイプロセッサで適切な変換行列との行列演算を行うことで実現できるこ

とも確認した．



(a) Input (b) 2D DCT (c) HPF + 2D IDCT

図 10. 提案されたアレイプロセッサによる境界強調処理結果.

4.1. 浮動小数点 FOU

浮動小数点演算の FPGA 実装法は，過去多く提案されており [10][11]，幾つかの IP コアは実際に使用可能である[12]．しかしながら，それらは加算や乗算などの基本的な演算しかサポートしておらず，提案したアレイプロセッサで必要となる **fma** 演算は実現されていない．よって，我々は IEEE 754 に準拠した単精度浮動小数点版の FOU を独自に設計した．FOU の仕様を表 3 にまとめる．乗算には Wallace Tree 構造を用い，FPGA 内の組込み乗算器は使用していない．よって，本回路構成は，まだ最適化の余地がある．本結果から，今日入手可能な大規模 FPGA を用いれば，*RapidMatrixX* アーキテクチャの浮動小数点版が実装可能であると考える．

6. まとめ

PE が 3 入力 1 出力の演算器をもつアレイプロセッサ・アーキテクチャを提案した．本アレイプロセッサは，Algebraic Path Problem を含む行列演算やデジタル信号処理などに広く適用可能である．16-bit 8x8 PE アレイを持つプロトタイプシステムを FPGA に実装し，実現可能性を評価した．本プロトタイプでは，大規模な問題が解けないものの，行列演算を効率的に解けることを確認した．今後の課題は，浮動小数点演算への対応と高速化のための制御部の改善である．

表 3. 浮動小数点版 FOU の仕様

Format	IEEE 754 standard-compliant 32-bit floating (unbiased rounding)
# of lines in Verilo-HDL	6k lines
Speed	200 MHz
# of pipelined stages	15
Mapped results (Xilinx Virtex-4 XC4LX200-11)	3561 slices (3 % utilization) used Xilinx ISE9.1
# of Gates (NAND2 equivalent)	31 KG (12KG combinational logic, 19 KG FF for pipeline) Used Synopsys Design Compiler Y-2006.06for linux

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] ---, "LAPACK – Linear Algebra PACKage," http://www.netlib.org/lapack/

[4] N. Moore, A. Conti, M. Leeser, and L. S. King, "Vforce: An Extensible Framework for Reconfigurable Supercomputing," IEEE Computer, pp.39-49, March 2007.

[5] Cray, "Cary XD1 Datasheet and XT4 Datasheet," www.cray.com

[6] SGI, "SGI RASC RC100 Blade Datasheet," www.sgi.com

[7] 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.

[8] GiDEL Ltd., www.gidel.com/PROCBoards.htm

[9] The MathWorks Inc., www.mathworks.com

[10] L. Zhuo, G. R. Morris, and V. K. Prasanna, "Designing Scalable FPGA-based Reduction Circuits Using Pipelined Floating-point Cores," Proc. Reconfigurable Architecture Workshop (RAW05), April 2005.

[11] O. Callanan, D. Gregg, A. Nisbet, and M. Peardon, "High Performance Scientific Computing Using FPGAs with IEEE Floating Point and Logarithmic Arithmetic for Lattice QCD," Proc. FPL 2006, August 2006.

[12] Xilinx, "Floating-point Operator V1.0," Datasheet DS335, July 2005.

[13] 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.

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

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

[16] G. Venkatarman, S. Sahni, and S. Mukhopadhyaya, "A blocked all-pairs shortest-paths algorithm," ACM Journal of Experimental Algorithms, Vol. 8, p. 2.2 , 2003.

[17] 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.

[18] D.J. Lehmann, "Algebraic structures for transitive closure," Theoretical Computer Science, Vol. 4, pp. 59–764, 1977.

[19] G. Rote, "Path problems in graphs," Springer Computing Supplementum, Vol. 7, pp. 155-189, 1990.

[20] L.E. Cannon, "A cellular computer to implement. the Kalman filter algorithm," Ph.D. dissertation, Montana State Univ., Bozeman, MT, 1969.