

ビジョンチップのための動的再構成可能な SIMD プロセッサ

小室 孝[†] 鏡 慎吾[†] 石川 正俊[†]

A Dynamically Reconfigurable SIMD Processor for a Vision Chip

Takashi KOMURO[†], Shingo KAGAMI[†], and Masatoshi ISHIKAWA[†]

あらまし 従来の画像処理用 SIMD プロセッサは、その並列性により主に初期視覚処理に対して強い威力を発揮するが、より高度な処理を行おうとした場合に、非局所演算が不得手であることや、画素数と汎用性の間にトレードオフが存在することなどの問題があった。本論文では、PE 同士を結合させる機能をもたせ、ハードウェアを動的に再構成可能にした新しいビジョンチップのアーキテクチャとアルゴリズム例を示す。また、同アーキテクチャに基づき、 $0.35\ \mu\text{m}$ CMOS プロセスを用いて 64×64 画素を集積したチップを試作したので報告する。

キーワード ビジョンチップ, 再構成可能, SIMD, 超並列処理, 高速画像処理

1. ま え が き

画像処理を専門に行うプロセッサは古くから開発されており、その多くは何らかの並列処理機構を備えていた。中でも並列度が数万にのぼる超並列プロセッサでは、少数または一つの画素ごとに処理要素 (PE) を割り当て、それらに同時に同一の命令を実行させる SIMD 型の制御を行うことで、ある種の画像処理を効率良く実行できることがわかっている。このようなプロセッサはかつてはスーパーコンピュータクラスの大規模なものであったが、近年、半導体集積化技術の進歩や、画像処理を手軽に扱いたいというニーズの高まりから、これをワンチップ化する動きが進んでいる。

しかし、このような画像処理プロセッサの問題点として、並列処理により画像処理が高速になっても、画像データの入力や出力に時間がかかってしまうため、実現できるフレームレートは低く抑えられるということがあった。それに対し筆者らは、イメージセンサの画素ごとに PE を取り付けることで、高フレームレートの画像処理を実現するビジョンチップの開発を行っている。

ビジョンチップにおいても、従来の画像処理用 SIMD プロセッサで採用されている構造はある程度有効であり、特に初期視覚処理と呼ばれる画像の前処理には、

強い威力を発揮する。その一方で、非局所的な演算が不得手であることや、画素数と汎用性の間にトレードオフが存在することなどの問題があった。

そこで今回設計したアーキテクチャでは、PE 同士を結合させる機能をもたせ、ハードウェアを動的に再構成可能にすることで、これらの問題を克服することに成功した。本論文では、設計したアーキテクチャとアルゴリズム、試作チップについて述べる。

2. 研究の背景

2.1 従来の研究例

イメージセンサの画素ごとに PE を付加したビジョンチップの研究はこれまで多く行われてきたが、それらの多くは PE の回路として集積化の容易な特定用途のアナログ回路を用いている [1]。アナログ回路はデジタル回路に比べ、少ない回路素子で実現されるため、高集積化には向いているが、デジタル回路に比べ回路の機能がある程度固定されてしまうという欠点がある。そこで石川は、ALU とレジスタからなる汎用の PE を採用したビジョンチップの提案を行い、そのためのコンパクトな PE のアーキテクチャを設計した。また、スケールアップモデルとして $64 \times 64 = 4096$ 個の PE をもつ SPE-4k システムを開発した [2]。このシステムを用いて様々な高速視覚のアルゴリズム開発や応用デモが行われ、システムの有効性が確認されている。この成果をもとに、小室らは高集積化に向けたアーキテクチャ S^3PE を設計し [3]、 64×64 画素の

[†] 東京大学大学院情報理工学系研究科, 東京都
Graduate School of Information Science and Technology,
The University of Tokyo, Tokyo, 113-8656 Japan

ワンチップ化を果たした [4] .

また、スウェーデンの Linköping 大学では光センサと AD コンバータ、プロセッサをワンチップ化したビジョンチップの研究が行われ、IVP 社より製品化された [5] . MAPP2200 と呼ばれるこのセンサは、 256×256 画素のイメージセンサの列ごとに AD コンバータと汎用の PE が取り付けられている . 光シートの投影によるレンジ像からの 3 次元計測に用いられる . 更に同研究グループは、画素単位で並列演算を行うビジョンチップ Near-Sensor Image Processing (NSIP) の研究も行っており、 32×32 画素のチップが開発されている [6] . 同チップは各 PE が 8 ビットのメモリをもち、ワイアード NAND 論理を基本とした演算を行う . フランスの Bernard らは Programmable Artificial Retina と呼ばれる汎用の PE を採用したビジョンチップを設計しており、 65×76 画素 (後に 128×128 画素) のチップが開発されている [7] . 同チップは 3 ビットのメモリをもち、主としてバイナリデータに対し、各種の論理演算を実行する .

一方、並列プロセッサの分野においても、これまでは主に科学技術計算などを想定した大規模なものが中心であったが、近年ロボットビジョンなどのリアルタイム応用を想定した小型化のアプローチが出てきている . NEC は 256 PE が SIMD 型の並列処理を行う 1 ボード画像処理システム IMAP-VISION を開発した [8] . これは 1 列分の画素を一つの PE で処理するものであるが、1 画素を一つの PE が処理する画素単位のものでは、Gealow らによる Pixel-Parallel Image Processor があり、 $64 \times 64 = 4096$ 個の PE がワンチップに収められている [9] . 同種のアプローチとして、Gayles らによる MGAP2 があり、49152 個の PE が 32 チップ構成で実現されている [10] .

2.2 従来の SIMD プロセッサの問題点

ビジョンチップを設計する上で、従来の画像処理用 SIMD プロセッサの構造は、大いに取り入れるべき部分も多いが、いくつかの問題もある .

一つは、画像処理の自由度を高めようとする、個々の PE の能力を高める必要があり、その結果回路面積が増大し、ワンチップに集積できる画素数が減ることになるという問題である . このように、汎用性と画素数の間にトレードオフの関係が存在することが、ビジョンチップを設計する上でジレンマとなる .

これに対し、前述の MGAP2 やいくつかの SIMD プロセッサは、コンディションレジスタを利用した PE

の結合機能をもち、結合された複数の PE を一つの大きな PE と見立てることで、PE の粒度を擬似的に変更することができる . したがって、上記トレードオフの問題をある程度解決しているといえる . しかし、結合がレジスタを介したものであるため、演算を行う際、PE の結合段数に比例した繰返し処理が必要となり、結合数が多くなった場合に効率が悪いという欠点を残している .

もう一つの問題は、多くの画像処理用 SIMD プロセッサが採用しているメッシュ結合ネットワークは、非局所的な演算、すなわち遠く離れた画素の情報を利用するような演算が不得手であるということである . メッシュ結合ネットワークは、ある PE が直接通信できるのはその周囲の PE とのみであるため、例えばエッジ検出における空間差分計算のように、ある画素の値がその画素自身とその周辺の画素からのみ決定されるような局所的な演算に対しては非常に強力であるが、非局所的な演算を行うには近傍通信を距離分だけ繰返す必要がある .

これに対し、前述の NSIP で採用されている GLU と呼ばれる回路や、小室らが提案しているモーメント抽出回路 [11] では、PE 間をレジスタを介さずに直接接続することで、近傍通信を繰返すことなく、遠くの画素に情報を伝達することができるようになっていく . 無論、経路となる PE の段数に比例して遅延時間が長くなるため、結局処理速度は距離に依存することになるが、それでもなおこの方式が有効であるのは、繰返し処理のオーバーヘッドがないことに加え、各段の処理時間が純粋な回路遅延のみから決まり、命令あるいは制御信号の供給周期に依存しないことなどによる .

したがって、これらの回路はメッシュ結合ネットワークにおける非局所演算の効率を高めている . しかし、実行できる演算が、画像全体に対する固定した処理に限られているため、一部の演算は効率良く実行できるが、種々のアルゴリズムを実装するには柔軟性が不足している .

3. アーキテクチャ

前節で述べたような問題点を解決し、より柔軟で高度な処理にも対応できるようなビジョンチップを実現するため、新しいアーキテクチャの設計を行った . 以下に詳細を述べる .

3.1 特徴

本アーキテクチャは、ビジョンチップに柔軟な PE

結合機能をもたせ、かつその結合をレジスタを介さないものにする事で、ハードウェアを動的に再構成可能にしているのが特徴である。その結果、例えば以下のようなことが実現されている。

- 画面全体または部分の、総和などのスカラ特徴量が高速に計算できる
- 離れた距離にある PE 同士の通信が高速に行える
- 異なる PE 粒度やネットワーク構造がエミュレートできる

なお、適当な数の PE ごとにレジスタを挟み、パイプライン処理を行うことで更なる効率化が実現できると期待されるが、その場合各段で PE に異なる処理をさせなければならず、ハードウェアが複雑になることから、今回は導入していない。

3.2 ハードウェア構成

全体の構造を図 1 に示す。PE がアレー状に並べられ、それぞれ光検出器 (PD) が備え付けられている。PE 間には上下左右が接続されており、近傍通信を行う。行及び列ごとに共通のグローバルバスが引かれており、シフトレジスタを通じて外部からデータを与えられる。処理結果は右端 PE に接続された列累積加算器を經由してスカラ値として最下位ビットから順にビットシリアルに出力される。

PE の構造を図 2 に示す。1 ビット ALU と 24 ビットのメモリから構成されており、プログラムにより制

御される。プログラム制御は外部の単一のコントローラで行い、すべての PE は一度に同一の命令を実行する。PD や列/行バスからの入力信号、ゼロ信号はメモリ空間にマップされており、入出力専用の命令を使わずに取り出せる。近傍 PE との通信はラッチを介して行われ、メモリ空間上に用意されたコンディションレジスタの内容によって、近傍通信の入力先を上下左右ゼロのいずれかを選択できるようになっている。

表 1 に命令セットを示す。各命令はアレーの外で制御信号にデコードされて各 PE に送られる。命令ごとの実行時間は通常一定であるが、後述の累積演算においては結合 PE の段数によって実行時間が変化する。

PD からの光強度信号の A-D 変換は、コンデンサに蓄えられた電荷が光電流によって放電され、降下する電圧がしきい値を切るまでの時間を計測している。その際、PE をカウンタとして用いることにより、専

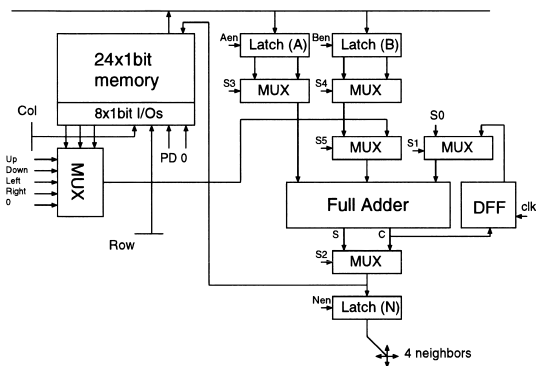


図 2 PE の構成
Fig. 2 Structure of the PE.

表 1 命令セット
Table 1 Instruction set.

命令	説明
read (addr, f _{ae} , f _{ben})	メモリからデータを読み込み、ALU 入力ラッチ (A and/or B) に格納する。
opw (op, addr, f _{clka})	演算の種類を選択し、結果をメモリに書き込む。必要ならキャリーを更新する。
opn (op, f _{clka})	演算の種類を選択し、結果を近傍出力ラッチに書き込む。必要ならキャリーを更新する。
nset (f _{nen})	近傍出力ラッチをイネーブル/ディセーブルにする。
pdset (f _{pdr})	PD のリセットスイッチをオン/オフする。
pulse (f _{clkb} , f _{clkc})	列累積加算器または列/行シフトレジスタのクロック線にパルスを与える。

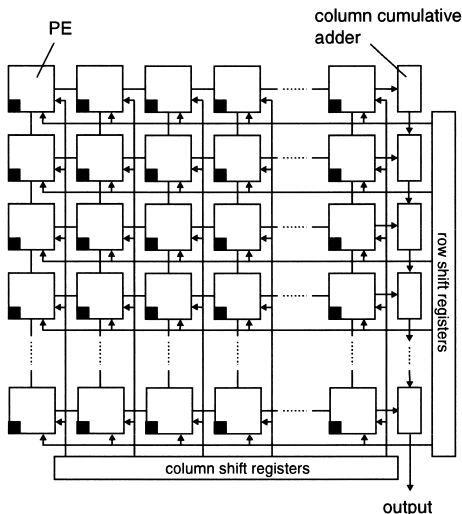


図 1 全体の構成
Fig. 1 Structure of the whole chip.

用回路を付加することなく A-D 変換を実現している。

3.3 演算機能

本アーキテクチャにおいて、単一の PE における演算は以下のような手順で実行される。

(1) 二つのデータをメモリから読み出し、ALU 入力ラッチ (A, B) にそれぞれに格納する。

(2) 演算の種類を選択し、結果をメモリに格納する。必要ならキャリーレジスタを更新する。

前回の演算で入力ラッチに格納した値と同じ値を用いる場合、読みみを省略できる。

PE 間通信は、演算結果を近傍出力ラッチ (N) に格納し、次の演算で ALU の入力的一方を近傍入力に設定することで行われる。近傍通信の入力元は、コンディションレジスタの内容によって PE ごとに選択できる。

これに加え、本アーキテクチャでは、ALU の入力的一方を近傍に設定した上で近傍出力ラッチ (N) をイネーブルにすることで、隣接する二つの PE を連結することができる。これをある範囲にわたって連続的に行うことで、一群の PE を結合することができる。手順としては、始点になる PE の近傍入力先を零に設定し、そこから一筆書きで終点まで連結していく。図 3 に結合の例を示す。

結合された PE の中では、累積演算を行うことができる。近傍出力ラッチをイネーブルにすることで近傍出力を筒抜けにし、それを通じて ALU の出力を隣接 PE の ALU の入力に直接つなぐことで、ALU を多段にし、累積演算を実現する。具体的には以下のような手順で行う。

(1) データをメモリから読み出し、ラッチ (A) に格納する。

(2) 演算の種類を選択する。この際、ALU の入

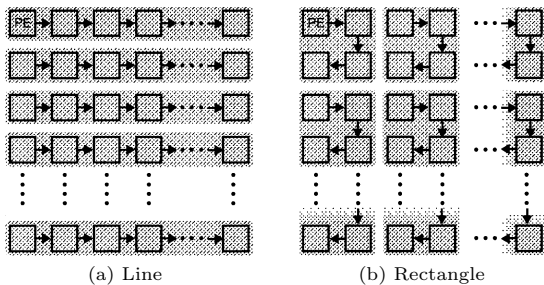


図 3 PE 結合の例
Fig. 3 Example of the PE connection.

力的一方を近傍に設定する。

(3) 近傍出力ラッチをイネーブルにする (図 4)。

(4) 結果をメモリに格納し、必要ならキャリーレジスタを更新する。

図 4 に累積演算の例を示す。演算の種類を論理和にすれば累積論理和が、加算にすれば累積加算が、キャリーにすれば多ビット加算が実行される。多ビット加算を行う場合、事前にキャリーレジスタにオペランドの一方 (足される値) を格納しておく必要がある。終点 PE からは、累積論理和を実行した場合は総論理和、累積加算を実行した場合は総和が出力される。

このように結合された PE は、一つの大きな PE とみなすことができる。例えば、 n 個の PE を結合した場合、1 ビット ALU が縦続接続され、 n ビット ALU として振る舞うことになる。この n ビット ALU は、先の累積演算を用いて n ビットデータ同士の加減算を一度に計算できる。このときにかかる遅延時間は、キャリー先読みなどをしない単純なリプルキャリアダのそれにデータがラッチやマルチプレクサを通過するのにかかる遅延分を加えたものと同等になる。

また、メモリ素子も 1 ビットメモリを 24 個もつ PE を n 個結合することで、 n ビットが 1 ワードのメモリが 24 個使用可能になる。

したがって、結合する PE の数を大きくすることで、結合された PE の演算能力を単一の PE の場合に比べ高めることが可能となる。すなわち可変粒度の並列処理が可能となる。

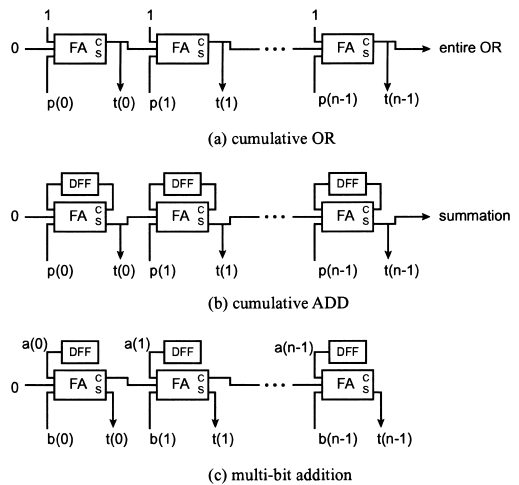


図 4 累積演算の例
Fig. 4 Example of the cumulative operation.

結合された PE 群の中では累積論理和によるブロードキャストでデータの受け渡しが行えるため、これを用いて離れた距離にある PE 同士の通信を効率良く実行できる。また、結合を動的に切り換えることで、ピラミッドアーキテクチャなどのメッシュ結合以外のネットワーク構造がエミュレートできる。

3.4 演算効率の評価

これまで述べてきたように、本アーキテクチャでは、レジスタを介さない PE 結合により、総和計算やブロードキャストなどの非局所演算が効率良く実行できるようになっている。ここでは、演算効率を定量的に比較することで、単純なメッシュ結合アーキテクチャに対する本アーキテクチャの優位性を示す。

例として、長さが N である 1 次元区間上の PE 内データの総和を計算することを考える。演算の方法は、先に示したビットシリアルな累積加算によるものとする。また、演算量は N に比例する部分だけを考慮し、前後の処理は含めないことにする。

単純なメッシュ結合アーキテクチャでは、総和の各桁を計算するのに N 回の繰返し処理が必要となるので、 N に比例した命令数がかかることになる。ここでは最小で見積もって N 命令かかるとし、命令の供給周期を T_i とすると、全体の実行時間は NT_i となる。

それに対し、本アーキテクチャでは総和の各けたを繰返し処理なしで計算することができる。ただし、 N 個の PE をデータが通過するため、データが PE を 1 個通過するのにかかる遅延時間を t_d とすると、全体の実行時間は Nt_d となる。厳密には、命令を送るタイミングの時間分解能には制限があるので、その精度で切り上げられることになる。

通常 T_i は t_d より大きく設定されるため、その分全体の実行時間は後者の方が短くなる。特にビジョンチップの場合、並列性の効果により、局所演算に関しては非常に少ない命令数で実行できるため、命令の供給周期は必ずしもそれほど速くなくてもよい。むしろ、命令の供給周期を大きくすることで、命令の供給にかかわる部分の回路設計を楽にする上、消費電力を抑える効果もある。後に示す試作チップでは、そのような設計を行っており、 T_i を t_d の見積もりよりずっと大きく設定している。

しかし、 T_i を t_d にできるだけ近づけた場合でも、PE ごとの遅延のばらつきを多数の法則により吸収できることから、その分 t_d の方が T_i より小さくなる。また、現実には t_d は ns オーダの数字となるため、そ

の周期で数千～数万の PE に同時に命令を供給することは技術的に相当困難であると予想される。

ここでは総和に関して評価を行ったが、ブロードキャストや多ビット加算など他の累積演算を用いた処理に対しても同様に評価を行うことができる。

4. アルゴリズム

前章で示したアーキテクチャに実装可能なアルゴリズムの例と、実行時間の見積りやメモリ使用量などの具体的な数字を以下に示す。

4.1 初期視覚処理

従来の画像処理用 SIMD プロセッサ同様、本アーキテクチャでも採用されているメッシュ結合ネットワークのハードウェア構成は、初期視覚処理と呼ばれる画像の前処理に効果的である。

表 2 に、各種の初期視覚処理アルゴリズムを実装した場合の命令数、実行時間、使用メモリ量を示す。命令ごとの実行時間 T は命令の供給周期 T_i に依存するが、ここでは $T = T_i = 100$ ns とする。また、チップ外の処理は含まない。これらのアルゴリズムは局所演算のみで構成されているため、命令数は画素数に依存しない。

4.2 スカラ特徴量計算

ビジョンチップの処理結果を画像として出力したのではセンサとプロセッサを一体化した意味がない。したがって、チップ内で画面全体からモーメントなどの特徴量を計算し、スカラ量として出力するのが望ましい。モーメントを求めるには総和計算が必須であるの

表 2 初期視覚処理アルゴリズムの実装例
Table 2 Sample implementation of the early visual processing.

アルゴリズム	命令数	時間	メモリ
画像取得* (6 ビット階調)	892	89 μ s	6
膨張 (バイナリー)	20	2.0 μ s	2
収縮 (バイナリー)	20	2.0 μ s	2
エッジ検出 (バイナリー)	31	3.1 μ s	3
平滑化 (バイナリー)	47	4.7 μ s	7
細線化 (バイナリー)	69	6.9 μ s	4
対象追跡 (バイナリー)	34	3.4 μ s	4
エッジ検出 (6 ビット階調)	193	19 μ s	19
平滑化 (6 ビット階調)	154	15 μ s	19
コンポリューションフィルタ (3×3 , 6 ビット階調)	2469	250 μ s	24
ポワソン方程式** (6 ビット階調)	144	14 μ s	21

* ウェイトなし

** この処理を何回か繰り返す

で、これを効率的に行うことがパフォーマンスの向上につながる。

本アーキテクチャで画面全体の総和を計算する方法として、全 PE を直列に結合して累積演算を行うことも可能ではあるが、結合段数が多すぎて演算効率が悪い。また、演算結果をチップ外で利用するには、外部出力のインタフェースが必要である。そこで、画面全体の総和を計算する場合に限り特別に、アレー右側に用意された列累積加算器を利用する。手順としては、まず PE を行方向に結合し、累積演算を行うことで行方向の総和を計算し、更にアレー右側の列累積加算器で列方向の総和を計算する。

ここで画素数を $N \times N (N = 2^n)$ 、各データのビット長を a とすると、行方向の総和を求めるには、 N 個の a ビットデータの総和を計算することとなり、これは最大 $(n+a)$ ビットの値をもち得るため、 $(n+a)$ けた分の総和演算が必要となる。

更に列方向の総和は、この $(n+a)$ ビットの各行総和を $N = 2^n$ 個合算することになり、これは最大 $(2n+a)$ ビットの値をもち得るため、 $(2n+a)$ けた分の総和演算が必要となる。

行方向の総和演算と列方向の総和演算は同時に実行できるため、結局 $(2n+a)$ けた分の総和演算が必要となる。各けたの総和演算に要する時間は先に示したとおり、PE の結合段数に比例する。これに、メモリからのデータの取出しや演算結果のメモリへの格納など、PE の結合段数によらない処理が加わる。

表 3 に、スカラー特徴量計算の例として画像全体の面積と重心を計算するアルゴリズムを実装した場合の命令数と実行時間を示す。画像はバイナリーとする。(A) は通常命令の数、(B) は結合 PE に対する命令の数である。このほか、列/行のグローバルバスに値をセットする処理も必要となるが、PE の処理ではないのでここでは考慮しない。

表 3 スカラー特徴量計算アルゴリズムの実装例
Table 3 Sample implementation of scalar feature extraction.

アルゴリズム	命令数		時間
	(A)	(B)	
面積 (64 × 64 画素)	60	14	7.4 μ s
面積 (128 × 128 画素)	68	16	10 μ s
面積 (256 × 256 画素)	76	18	13 μ s
重心 (64 × 64 画素)	344	52	40 μ s
重心 (128 × 128 画素)	398	60	52 μ s
重心 (256 × 256 画素)	452	68	66 μ s

(A) の命令ごとの実行時間 T_A は、先と同様 $T_A = T_i = 100$ ns とするが、(B) の命令ごとの実行時間は T_B は結合段数 n と PE ごとの遅延時間 t_d に依存し、ここでは $T_B = nt_d = 0.5n$ ns (100 ns 未満は切上げ) とする。なお、列累積加算器の遅延時間は、PE の遅延時間と同じとして計算する。

一方、単純なメッシュ結合のアーキテクチャを想定した場合、累積演算は段数分の通常演算に置き換える必要があるため、(B) の 1 命令は (A) の n 命令分に換算され、例えば 256×256 画素のバイナリー画像の重心を求めるのに 3.5 ms もかかる計算になる。

上で示した処理は、画面全体の面積と重心を求めるものであるが、対象追跡のアルゴリズムと組み合わせ、領域をマスクして計算することで、対象の面積・重心のみを求めることが可能である。対象が複数ある場合でも、最初にラベリングを行ったのち、対象ごとにトラッキング・特徴量計算することで、各対象の面積・重心が求められる [12]。

また、バイナリーエッジ検出と総和計算を組み合わせることで周辺長の計算を行うなど、面積・重心以外のスカラー特徴量を計算することもできる。例えば、対象の形状をモデル近似表現したときのパラメータ算出を行いたい場合、パラメータを変化させながらモデル画像をビジョンチップ内に生成し、対象画像との SAD (差分の絶対値の総和) を計算し、その値が最小となるパラメータを探索することで実現できる [13]。この場合、繰返し演算が必要となるのが欠点であるが、高フレームレート動画を仮定すると、フレーム間の画像の変動が少なくなることから、モデルの種類によってはパラメータの変動も少ないと仮定できる場合があり、その場合は前フレームの情報を利用して計算量を大幅に減らすことも可能である [14]。

4.3 ローカル特徴量フィードバック

結合された PE の中で、累積演算を用いて総和などのスカラー量を計算し、その結果をブロードキャストを用いて結合 PE に戻すことで、結合 PE 内でのローカルな特徴量のフィードバックが実現される。これらの一連の処理はビットシリアルに実現可能であるため、作業用のメモリを消費せずに済む。図 5 に例を示す。

4×4 の PE が直列に結合されており、バイナリー画像 (図の上段に黒で表示されている) が与えられているとする。まず、結合 PE の中で画像の総和の LSB を計算し、終点 PE のメモリに格納する (図ではハッチングで表示されている)。次に終点 PE に格納され

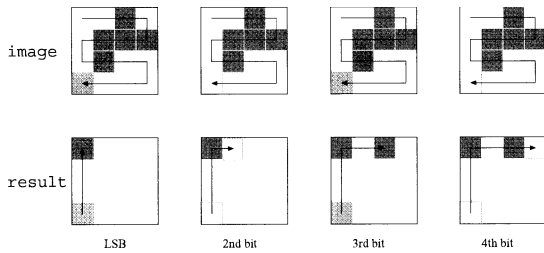


図 5 ローカル特徴量フィードバック
Fig. 5 Local feature value feedback.

た総和の LSB をブロック内にブロードキャストし、左上の PE のメモリに格納する (図の下段に黒で表示されている)。これを LSB からビットごとに繰り返すことで、総和値全体をビットシリアルにフィードバックする。このとき、フィードバックした総和値はビットごとに異なる位置の PE に格納する。このように、複数の PE を、あるときは空間を表すために、あるときはビット列を表すために用いており、メモリを自由に無駄なく使うことができる。

このローカル特徴量フィードバックを用いたアルゴリズムの例として、並列ブロックマッチングが挙げられる。2 枚の画像のうち、一方の区分けされた各部分が、もう一方の画像においてどちらにずれているかを探索するアルゴリズムである。1 フレーム前の入力とのマッチングを取ることで、オプティカルフローを計算し、動き計測やモーションステレオを実現するなどの応用が考えられる。

アルゴリズムの手順は、以下ようになる。

- (1) 一方の画像を探索領域の中で適当に並行移動させる。
- (2) ブロック単位でもう一方の画像との SAD を計算する
- (3) SAD がこれまでの最小値より小さい場合、最小値とインデックスを更新する。
- (4) 探索領域をすべて移動し終わるまで(1)~(3)を繰り返す

通常 SAD のような演算は、総和計算に時間がかかり、何度も繰り返し演算するのは難しいが、本チップではブロック内総和が効率良く計算できるため、無理なく実行できる。SAD、最小値、インデックスを結合 PE のメモリに分散してもたせることができるため、メモリ領域を圧迫せずに済んでいる。また、SAD の比較などブロック単位で行われる演算は、PE 結合に

表 4 並列ブロックマッチングアルゴリズムの実装例
Table 4 Sample implementation of parallel block matching.

アルゴリズム	命令数		時間
	(A)	(B)	
ブロックサイズ 8×8			
探索領域 7×7	23820	1174	2.5 ms
探索領域 5×5	12204	598	1.3 ms
探索領域 3×3	4460	214	470 μ s
ブロックサイズ 4×4			
探索領域 7×7	20374	978	2.1 ms
探索領域 5×5	10438	498	1.1 ms
探索領域 3×3	3814	178	400 μ s

よる多ビット加算を用いているため、ビットシリアルに計算するより効率が高まっている。

表 4 に、並列ブロックマッチングを実装した場合の命令数、実行時間を示す。命令数の定義、各命令にかかる実行時間は表 3 と同様である。画像は 4 ビット階調とする。使用メモリ量は、ブロックサイズ、探索領域に関係なく 23 である。なお、先のスカラー特徴量計算の場合と同様、単純なメッシュ結合のアーキテクチャでは (B) の 1 命令は (A) の n 命令分に換算される。

この例のように、ローカル特徴量フィードバックは計算された特徴量をビジョンチップ内で使用することを前提としている。したがって、特徴量を外部で使いたい場合は、前述の画像全体に対するスカラー特徴量計算を用いることになる。

4.4 列並列処理による座標変換

従来の画像処理用 SIMD プロセッサの中には MAPP2200 や IMAP-VISION のように、列に一つだけ PE を設置した列並列のものが存在する。このタイプのプロセッサは、列並列にすることで完全並列に比べ、1 列分の繰返し処理が必要となるので速度は落ちるが、その分個々の PE の演算能力やメモリ量を高めることを目指していると思われる。また、列並列プロセッサでは、列ごとに任意の画素にアクセスできるため、座標変換を含むアルゴリズムにおいて威力を発揮する。

本アーキテクチャでは、PE 結合機能を用いて PE を列方向に一列につなげることで、演算能力やメモリ量を高められるほか、ブロードキャストを用いて同じ列内の PE 同士が通信することができる。これにより、列並列プロセッサと同等の機能を擬似的に実現することができる。

ただし、先に示したように、離れた距離にある PE 同士の通信には距離に比例した遅延がかかるため、性

能的には列並列プロセッサと全く同等ということではできない。しかし、画素並列、列並列、行並列などの異なる構造を動的に切り換えられるという利点がある。座標変換系アルゴリズムの実装例として、90度回転のアルゴリズムの手順を図6に示す。

対角線を介して列ごとに列を行に置き換えていく。同一列内及び同一行内でのデータの移動は、累積演算によるブロードキャストを用いて行う。

拡大・縮小・平行移動も、まず列ごとに行方向に画素値をコピーする処理を行い、次に行ごとに列方向に同処理を行うことで実現される。

表5に、これらの座標変換系アルゴリズムを実装した場合の命令数と実行時間を示す。ステップ数の定義、各ステップにかかる実行時間は表3と同様である。画像はバイナリーとする。

これまでの例と同様、単純なメッシュ結合のアーキテクチャでは(B)の1命令は(A)の n 命令に換算される。一方、列並列のアーキテクチャを想定した場合、列方向の処理に限り、結合段数に関係なく(B)の1命令は(A)の1命令に置き換えられる。例えば90度回転は、(B)の命令の半分が列並列の処理であるため(A)の1命令に置き換えられ、もう半分が列並列の処理でないため(A)の n 命令分に換算される。その結果、 256×256 画素の場合、90度回転のアルゴリズム

は14msかかることになる。

4.5 結合の自己生成

これまで述べたPEの結合の形はある程度規則的であり、結合の指定は外から列/行共通バスを通じて行っていた。それに対し、画像処理の結果をもとにPEの結合を自己生成することもできる。

例えば、対象を囲む長方形にPEを結合し、その中でローカル特徴量フィードバックを行うことで、従来対象ごとに順番に行わなければならなかった処理を同時に行うといったようなことができる。ただし、各長方形がオーバーラップした場合、複数の対象が同一のものとなってしまうので、そのような状況が起こらない環境で用いるか、オーバーラップした部分だけ繰り返し処理を行うなどの工夫が必要となる。具体的には、例えば各長方形内で高次局所自己相関特徴を用いて孤立対象の個数を計算し[15]、個数が2個以上と判定された領域はまとめて画像全体でラベリングしたのち、対象ごとに特徴量計算を行うといった方法が考えられる。

結合の自己生成そのものの処理は通常演算で行うため、単純なメッシュ結合アーキテクチャとの性能の違いはないが、結合されたPEの中では累積演算を行うことになるため、そこではこれまでの処理と同様に性能向上が期待される。

5. 試作チップ

本アーキテクチャをもとに試作チップの設計を行った。0.35 μm CMOS DLP/TLM プロセス、エリアサイズ $5.4 \text{ mm} \times 5.4 \text{ mm}$ の中に 64×64 画素のPEアレーと列累積加算器、行/列シフトレジスタが搭載されている。命令を制御信号に変換するデコーダは、今回は外付けとした。各PEの面積は $67.4 \mu\text{m} \times 67.4 \mu\text{m}$ であり、高集積化が達成されている。 256×256 画素が約1.8cm角のチップに乗る計算であり、画像処理デバイスとして標準的な画素数に迫っている。

動作速度に関しては、デコーダが外付けであるため、制御周期がその速度に制限されており、現時点では12.5nsで動作させている。これは今後大いに改善の余地があり、デコーダがチップ内に搭載されれば、画素数にもよるが、数ns程度まで向上できると期待される。なお、1命令は8ステップの制御信号に変換される。したがって、命令の供給周期は100nsである。

また、累積演算におけるPE当りの遅延時間の測定値は2.02nsであった。この数字はアルゴリズム評価

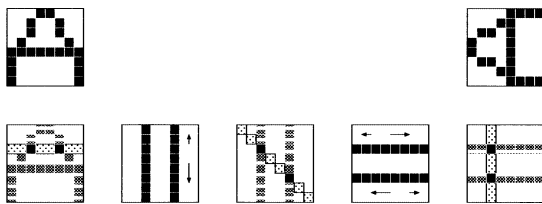


図6 90度回転のアルゴリズム
Fig. 6 90 degree rotation algorithm.

表5 座標変換系アルゴリズムの実装例
Table 5 Sample implementation of axis transformation algorithms.

アルゴリズム	命令数		時間
	(A)	(B)	
90度回転			
(64 × 64 画素)	3033	256	330 μs
(128 × 128 画素)	5991	512	650 μs
(256 × 256 画素)	11893	1024	1.4 ms
拡大・縮小・平行移動			
(64 × 64 画素)	3854	128	400 μs
(128 × 128 画素)	7694	256	800 μs
(256 × 256 画素)	15374	512	1.6 ms

時に仮定した 0.5 ns より 4 倍以上も大きい。回路やプロセスの改善によって高速化することは十分可能である。

画素のレイアウトを図 7 に、チップ写真を図 8 に示す。

本チップを用いて各種の画像処理を行った。本来ビジョンチップでは、スカラ特徴量のみを出力するのがあるべき使い方だが、ここでは動作確認のため画像を出力している。

図 9 に初期視覚処理及びスカラ特徴量計算を行った

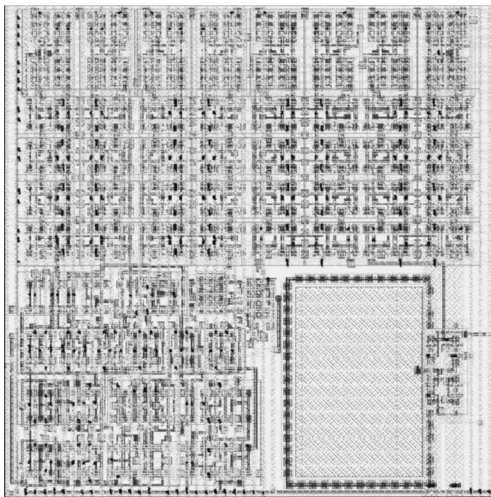


図 7 試作チップの画素レイアウト
Fig. 7 Pixel layout of the prototype chip.

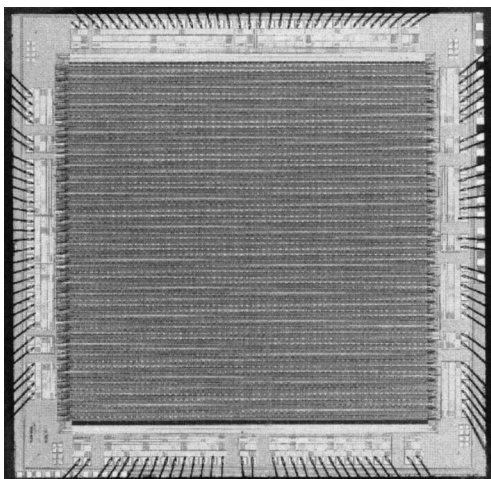


図 8 試作チップの写真
Fig. 8 Photo of the prototype chip.

ときの出力例を示す。入力画像には照明を当てた腕時計を用いた。(a) が原画像、(b) が原画像を 2 値化してエッジ検出を行ったもの、(c) が原画像を 2 値化して重心を計算し、表示したものである。チップ外の処理を含めたトータルの実行時間は、画像取得に 1.0 ms、2 値化に 2.2 μ s、エッジ検出に 3.1 μ s、重心計算に 110 μ s、重心描画に 14 μ s がかかっている。画像取得の際、ウェイトを入れてセンサの露光時間を調整している。重心計算にアルゴリズム評価時の見積りより時間がかかっているのは、主に座標値の供給などチップ外の処理が多いためである。

図 10 はランダムドットを印刷したシートでマスクした透過光に対し、並列ブロックマッチングを用いてオプティカルフローを計算した結果である。ブロックサイズは 8×8 、探索領域は 5×5 である。シートを前後左右に平行移動させると、それに合わせて移動ベクトルが一斉に同じ方向を向くのがわかる。ただし、画面の端が切れているため、その付近は正しく計算されない。チップ外の処理を含めたトータルの実行時間は、マッチング処理に 1.3 ms、移動ベクトル描画に 22 μ s がかかっている。

図 11 の (a) は、文字を印刷したシートでマスクした透過光に対し、90 度回転のアルゴリズムを実行した結果である。同様に (b) は、拡大を行った例である。倍

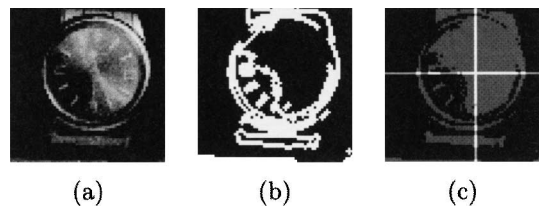


図 9 初期視覚処理/スカラ特徴量計算の出力例
Fig. 9 Sample output of early visual processing and scalar feature extraction.

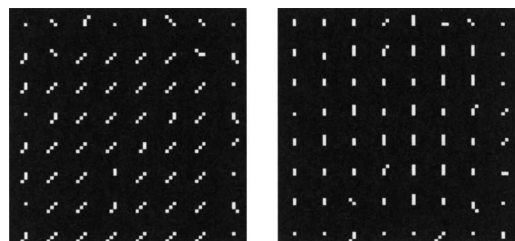


図 10 並列ブロックマッチングの出力例
Fig. 10 Sample output of parallel block matching.

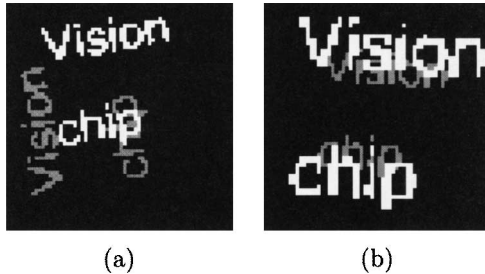


図 11 座標変換系アルゴリズムの出力例

Fig. 11 Sample output of axis transformation algorithms.

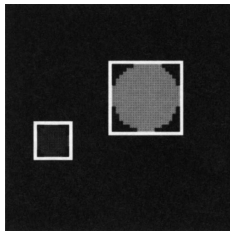


図 12 結合の自己生成の出力例

Fig. 12 Sample output of self-configuration of connections.

率は 1.5 倍とし、画像の中央を中心に拡大した。チップ外の処理を含めたトータルの実行時間は、90 度回転に $520 \mu\text{s}$ 、拡大に $510 \mu\text{s}$ かかっている。

図 12 は、入力画像内の対象を囲む長方形を作成し、それをもとに PE の結合を自己生成し、各対象の面積を計算した例である。面積は明るさで表されており、面積が大きいほど明るくなっている。チップ外の処理を含めたトータルの実行時間は、長方形の更新に $74 \mu\text{s}$ 、結合の自己生成と面積計算に $170 \mu\text{s}$ かかっている。

6. む す び

PE 同士を結合し、ハードウェアを動的に再構成することで、柔軟で高度な画像処理を実現する新しいビジョンチップのプロセッサアーキテクチャを示した。これにより、ビジョンチップをはじめとするリアルタイム画像処理の応用範囲が格段に広がると期待される。

文 献

- [1] A. Moini, VISION CHIPS, Kluwer Academic Publishers, 2000.
- [2] M. Ishikawa, A. Morita, and N. Takayanagi, "High speed vision system using massively parallel processing," Proc. Int. Conf. on Intelligent Robots and Sys-

tems, pp.373-377, 1992.

- [3] 小室 孝, 鈴木伸介, 石井 抱, 石川正俊, "汎用プロセッシングエレメントを用いた超並列・超高速ビジョンチップの設計;" 信学論 (D-I), vol.J81-D-I, no.2, pp.70-76, Feb. 1998.
- [4] M. Ishikawa and T. Komuro, "Digital vision chips and high-speed vision systems," Proc. 2001 Symposium on VLSI Circuits, pp.1-4, 2001.
- [5] 小森有二, "3 次元計測用高速演算チップとカメラシステム;" O plus E, no.202, pp.93-99, 1996.
- [6] R.Forchheimer and A. Åström, "Near-sensor image processing: A new paradigm," IEEE Trans. Image Process., vol.3, no.6, pp.736-746, 1994.
- [7] T.M. Bernard, B.Y. Zavidovique, and F.J. Devos, "A programmable artificial retina," IEEE J. Solid-State Circuits, vol.28, no.7, pp.789-797, 1993.
- [8] Y. Fujita, S. Kyo, N. Yamashita, and S. Okazaki, "A 10 GIPS SIMD processor for PC-based real-time vision applications—Architecture, algorithm implementation and language support," Proc. 4th IEEE Int. Workshop on Computer Architecture for Machine Perception, pp.22-32, 1997.
- [9] J.C. Gealow, F.P. Herrmann, L.T. Hsu, and C.S. Sodini, "System design for pixel-parallel image processing," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.4, no.1, pp.32-41, 1996.
- [10] E.S. Gayles, T.P. Kelliher, R.M. Owens, and M.J. Irwin, "The design of MGAP2: A micro-grained massively parallel array," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.8, no.6, pp.709-716, 2000.
- [11] 小室 孝, 石井 抱, 中坊嘉宏, 石川正俊, "デジタルビジョンチップのためのモーメント抽出アーキテクチャ;" 信学技報, PRMU99-51, 1999.
- [12] 渡辺義浩, 小室 孝, 鏡 慎吾, 石川正俊, "ビジョンチップを用いた分割領域のラベリングと回転計測への応用;" 日本ロボット学会創立 20 周年記念学術講演会講演論文集, 3A14, 2002.
- [13] 石井 抱, 石川正俊, "超並列・超高速ビジョンのためのマッチングアルゴリズム;" 信学技報, PRU-95-70, 1995.
- [14] 石井 抱, 石川正俊, "高速ビジョンのための直線抽出法;" 信学論 (D-II), vol.J81-D-II, no.8, pp.1920-1926, Aug. 1998.
- [15] 山本健吉, 石井 抱, "高次自己相関ビジョンチップのアーキテクチャ設計;" 信学技報, ICD2002-18, 2002.

(平成 15 年 1 月 29 日受付, 4 月 23 日再受付)



小室 孝

平 8 東大・工・計数卒・平 10 同大学院
修士課程了。平 13 同大学院博士課程了。現
在，同大学院情報理工学系研究科システム
情報学専攻助手。ビジョンチップ，並列プロ
セッサに関する研究に従事。博士（工学）。



鏡 慎吾

平 10 東大・工・計数卒・平 12 同大
学院修士課程了。平 15 同大学院博士課程
了。現在，同大学院情報理工学系研究科シ
ステム情報学専攻助手。実時間センサ情報
処理アーキテクチャ・システムの研究に従
事。博士（工学）。



石川 正俊（正員）

昭 52 東大・工・計数卒。昭 54 同大
学院修士課程了。同年通産省工業技術院製
品科学研究所に入所。平元東大・工・計数
助教授。現在同大学院情報理工学系研究
科システム情報学専攻教授。超並列・超高
速ビジョン，センサフュージョン，光コン
ピューティング等に関する研究に従事。工博。