計算機アーキテクチャ 125－7
（1997. 8. 20）

ビジョンチップシステムのためのソフトウェア開発環境の構築

松内 良介 村田 達也 石井 抱 石川 正俊
東京大学大学院工学系研究科計数工学専攻

近年，画素毎に光センサと汎用デジタル処理要素を直結したものを1チップ上に集積化した汎用ビジョンチップの開発が注目されており，従来にはない高速リアルタイムビジョンとして様々な応用分野においてその実用化が期待されている．本稿では，このようなビジョンチップのシステムアーキテクチャについて論じた上で，システムとして汎用ビジョンチップを実用化するための重要な課題の一つであるプログラミングのためのソフトウェア環境について議論する．また実際に，ビジョンチップアーキテクチャに基づいたデータ型により定義されたC言語ライクなプログラミング言語 SPE-C (Sensory Processing Elements-C) を提案し，その設計仕様について述べ,.さらにコンパイラを実装し，コンパイル結果を示すことにより動作を確認する．

Programming Language SPE-C for General Purpose Vision Chip System

Ryosuke Matsuuchi, Tatsuya Murata, Idaku Ishii and Masatoshi Ishikawa
Graduated School of Mathematical Engineering and Information Physics, University of Tokyo

A general purpose vision chip on which a photo detector and a processing element is directly connected at each pixel is widely noticed and is hoped for a new powerful system of high speed real-time vision in various application systems. In this paper we argue a new concept for a vision chip architecture, especially on software environment for programming which is inevitable for making a vision chip practicable as a system. In fact we design a programming language called SPE-C (Sensory Processing Elements-C) for a general purpose vision chip system which adopts some abstract data types based on data flow in the system, and implements its compiler for our developing system to show that it works correctly.

1. はじめに

近年，光センサと汎用演算器を画素ごとに直結したものを1チップ上に集積化したビジョンチップの考えが，ビデオフレームレート (約30 frame/sec) を大幅に超える高速リアルタイムビジョンとして注目されている [1]．このような汎用ビジョンチップを様々な応用分野でビジョンシステムとして実用化するためには，効率の良いプログラミング開発環境が重要な問題となるが，現段階ではそのソフトウェア環境については議論されていない．

そこで本稿では，システムとしてのビジョンチップについて論じた上で，それを実現するために必要とされるソフトウェア環境について議論する．また実際に，データの流れを新たなデータ型の定義により記述するビジョンチップ用プログラミング言語を提案・設計し，そのためのコンパイラを実装した結果を示す．

2. ビジョンチップシステム

2.1 ビジョンチップの考え

光センサと演算器を一体化し，高速に初期視覚処理を行う並列演算処理チップであるビジョンチップの研究は，アナログ回路により実現することから始まった [2, 3]．これらはいずれもビデオレートを超える速度での初期視覚処理を可能としたが，アナログ回路による固定回路であったため，用途ごとに回路を設計しなおす必要があった．

小室らはこれに対し，汎用ビジョンチップアーキテクチャ S^3PE (Simple and Smart Sensory Processing Element) を提案し [4]，様々なアルゴリズムが同一センサ上で高速に動作可能であることを確認し，実際にチップ試作を行っている．S^3PE は，プログラマブルな並列処理システムを1チップ上に集積し，命令コード体系を簡潔化するために，1) SIMD 型制御，2) 4近傍接続，3) ビットシリアル演算，4) フラットな

レジスタ (24bit), 5) 機能レジスタの採用, 6) 演算回路の単純化 (AND, OR, XOR, ADD) を行っている.

2.2 システムとしてのアーキテクチャ

汎用ビジョンチップデバイスの開発を行うと同時に, その有効性を示すために, 実際の応用を念頭においたシステムを構築することが重要となる. このような例として, 中坊らは汎用ビジョンチップアーキテクチャを用いた 1ms ターゲットトラッキングシステムを構築し, 従来にはない高速なビジュアルフィードバックレートでのロボット制御を実現した [5]. このような開発を行った結果, 他のシステムとの協調動作を考慮し, 並列演算部からの並列出力を用いた特徴量計算回路 [6] などの周辺回路と組み合わせて動作させることが必要とされ, デバイスとしてだけではなく, システムとしてのビジョンチップを考えることが重要となってきた.

そこで, 村田らはシステムとしてのビジョンチップという観点から制御アーキテクチャの新たな設計を行い, そのためのビジョンチップコントローラの開発を行った [7]. 図 1に, 制御アーキテクチャの概要を示す. この制御アーキテクチャでは, 1) プログラムメモリの設置, 2) システムとしての演算能力の独立性, 3) 階層型制御, 4) 分散モジュール化によるソフトウェア制御 を実現する. これにより, 上位プロセッサと, 入力部, 並列演算部, 出力部からなるビジョンチップデバイス間の I/O 速度の限界, およびビジョンチップデバイス単体での処理の限界などの制約を, 新たに専用コントローラを設置することにより, ビジョンチップデバイスの持つ能力を最大限に引き出すことができる.

新たに設計したコントローラには, プログラムメモリ (並列演算部およびコントローラ自身のプログラムを格納), レジスタ, ALU が含まれる. コントローラだけで独立した演算処理, およびプログラムの分岐が可能であり, コントローラ／入力部／並列演算部／出力部においてプログラムをループさせることができるようになっている.

2.3 ソフトウェア環境の重要性

以上のようにして, ビジョンチップシステムのハードウェアが, 入力部, 並列演算部のみならず, 出力部 (特徴量計算回路), コントローラ, 上位プロセッサといったように, 多種多様なモジュールから構成されるようになってきた. このような複雑化してきたハードウェアをプログラミングするには, より多くの分野でより多くの人が簡単に使うことを可能とする, 抽象度の高いプログラミング言語が不可欠である. そこで, 以降では我々が設計・開発したビジョンチップ用プログラミング言語 SPE-C について述べ, そのコンパイラ環境開発を実際のプログラムを例に説明する.

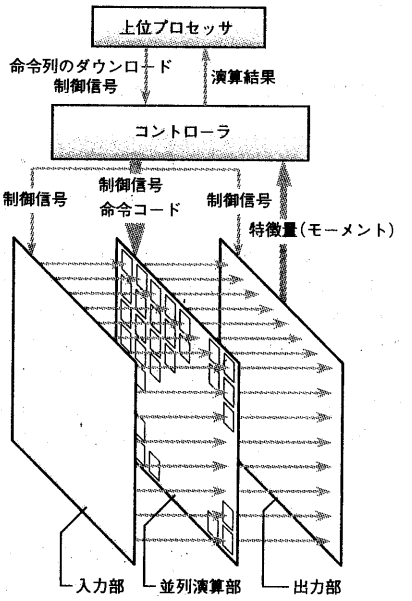

図 1 ビジョンチップシステムのアーキテクチャ

3. プログラミング言語 SPE-C

提案するビジョンチップ用プログラミング言語 SPE-C は, ビジョンチップシステムのうち, 並列演算部, 入力部, 出力部 (特徴量計算回路), コントローラ, という 4 つの部分の動作を統一的に記述するための高級言語である.

3.1 設計方針

SPE-C のデータ型は, 各ハードウェアモジュールに直接対応するものではなく, それらの間で取り交わされる情報の種類に対応したデータ型となっている. ハードウェア各部の連携動作を, こうして用意したデータ型どうしの演算として定義し直すことで, SPE-C は, ハードウェア各部の連携動作の詳細をプログラマから隠蔽する.

SPE-C の基礎となっているデータ型は, parallel(N) 型, shared(N) 型の 2 つである (図 2). 記憶領域で言えば, parallel(N) 型は並列演算部の全 PE (Processing Element) のレジスタの中の長さ N ビットの領域に対応し, shared(N) 型はコントローラのレジスタの中の長さ N ビットの領域に対応している. SPE-C において宣言された変数は, このような記憶領域に対応づけられる. parallel(N) 型の演算は並列演算部の ALU により行われ, shared(N) 型の演算はコントローラの ALU により行われる.

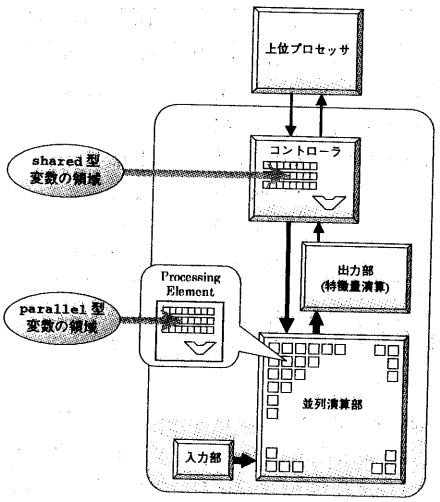

図 2 parallel 型 / shared 型の着想

これらのデータ型を用いると，入力部，出力部 (特徴量計算回路) の働きが，データ型どうしの代入演算や型変換として定義できる．そこで，入力部，出力部 (特徴量演算回路) の振る舞いを抽象化して定義したオブジェクトを「システムオブジェクト」として言語に用意し，入出力の動作は，システムオブジェクトと parallel / shared 型変数の間での代入演算として定義する．

データ型以外の構文 (式，制御構造，関数) については，ほぼ ANSI C 言語に沿って定義した．このため，SPE-C プログラムの概観は C 言語とほぼ変わらない．例えば，SPE-C プログラムの実行は，C 言語と同様に main 関数から開始される．

3.2 parallel 型変数

parallel(N) 型変数は，並列演算部において並列に存在する PE のレジスタのうち，長さ N ビットの領域に対応する変数である．この変数に対する演算はそのまま，全 PE においてインストラクションを SIMD 方式で実行することを意味する．parallel(N) 型の変数を一つ宣言すると，それがスコープ内にある間は，並列演算部の各 PE のレジスタ領域がそれぞれ N ビットずつ消費されている．

例えば，

```
parallel(3) a;
```

と宣言された変数 a は，3 ビットの長さを持つ．この 3 ビットがどのように PE レジスタ上に確保されるかはプログラマには見えない．

こうして宣言された (PE のレジスタ上に確保された)parallel 型変数に対して

```
a = 3;
```

のような代入を行えば，並列演算部のすべての PE のレジスタの値が同時に変化する．この場合，長さ 3 ビットの parallel 型変数に十進数の 6 (二進数で 110) を代入しているので，対応する各 PE のレジスタ領域の値は一斉に 110 になる．

さらに，

```
parallel(3) a,b,c;
a = b + c;
```

のような PE レジスタ内での演算 (これがすべての PE において同時に起こる) も記述でき，

```
parallel(5) a;
a = (a*4+a.right+a.left+a.up+a.down)/8;
```

などのような式を記述することにより，画像フィルタを容易に表現できる．

parallel 型の演算式には，C 言語と同様の四則演算子，ビット演算子が利用できる．

3.3 shared 型変数

shared(N) 型変数とは，コントローラのレジスタのうち，長さ N ビットの領域に対応する変数である．この変数に対する演算はそのまま，コントローラにおいて演算命令を実行することを意味する．shared(N) 型の変数を一つ宣言すると，それがスコープ内にある間はコントローラのレジスタ領域が N ビット消費されている．

parallel 型変数とは違い，shared 型変数は並列に存在するのではなく，1 個の shared 型変数はコントローラのレジスタ上の 1 つの領域にだけ対応している．この点で，SPE-C の shared 型変数は，一般に利用されている C 言語や FORTRAN などの変数に似ており，多くの人に馴染みやすいものである．

shared 型変数の宣言は例えば次のように行う．

```
shared(4) a;
```

この宣言により，コントローラレジスタ上に 4 ビットの領域が確保され，識別子 a で参照可能となる．こうして宣言された shared 型変数に対しては，C 言語の int 型と同様に演算を行える．

3.4 システムオブジェクト vchip

SPE-C においては，並列演算部，入力部，出力部，コントローラといった各部の間での入出力は，parallel 型 / shared 型変数とシステムオブジェクトとの間での代入演算によって記述する．

システムオブジェクトは，SPE-C において予約語 vchip として定義されており，このオブジェクトの個別のプロパティを式の中に記述することによって入出力演算を記述する．用意した vchip オブジェクトのプロパティは表 1 のようなものである．

表 1 システムオブジェクト vchip に用意されているプロパティ

プロパティ	意味	型[1]	Read/Write[1]
vchip.output(N)	出力部へ送る値 (N ビット)	parallel 型	Write のみ
vchip.out.moment0	出力部の出力(0 次モーメント)	shared 型	Read のみ
vchip.out.moment1x	出力部の出力(x 方向の 1 次モーメント)	shared 型	Read のみ
vchip.out.moment1y	出力部の出力(y 方向の 1 次モーメント)	shared 型	Read のみ
vchip.out.gx	x 方向の重心	shared 型	Read のみ
vchip.out.gy	y 方向の重心	shared 型	Read のみ
vchip.input(N)	入力部から得られる値 (N ビット)	parallel 型	Read のみ

[1] vchip オブジェクトの各プロパティの parallel 型, shared 型のビット数はハードウェア仕様に依存する。
[2] Read/Write : ここでは、代入式の左辺 (代入先) に用いることのできるプロパティを「Write」、その他の右辺式 (評価されて値が決定される式) に用いることのできるプロパティを「Read」と分類している。現在のところ、左辺式にも右辺式にも用いることのできる vchip プロパティは存在しない。

3.4.1 parallel 型変数と入出力

入力部から並列演算部へセンサ情報を並列に取り込むには, 次のように記述する.

```
parallel(1) a;
a = vchip.input(1);
```

ここで, vchip.input はシステムオブジェクト vchip のプロパティである. この例では引数が 1 になっているが, これは「ビジョンチップの入力部から 1 ビットの値を得る」ということを意味している.

vchip.input(1) の (式としての) 型は parallel(1) である. この代入文により, 入力部から得られた値が, PE のレジスタの, 変数 a に対応する領域に各画素それぞれについて格納される.

同様にして, 並列演算部から出力部へレジスタ値を転送するには, 次のように記述する.

```
parallel(1) a;
vchip.output(1) = a;
```

システムオブジェクト vchip のプロパティ output に対して代入を行なうことにより, 並列出力を行なうことができる (vchip.output(1) の型は parallel(1)). この結果, 出力部により特徴量が計算され, これ以後 vchip.out.X プロパティにより画像の特徴量を shared 型の値として取り出すことができるようになる.

3.4.2 shared 型変数と入出力

shared 型変数に画像の特徴量を得るには, 次のように記述する.

```
shared(8) a;
a = vchip.out.moment0;
```

この代入式により, vchip.output(1) に代入された parallel 型変数から計算された特徴量を, shared 型変数として取り出すことができる. 他の特徴量についても, 表 1 のとおりに用意されているが, 利用方法はこれと同様である.

3.5 parallel 型と shared 型の相互演算

以上のような parallel(N) 型, shared(N) 型, システムオブジェクト vchip の各プロパティの関係を整理すると, ビジョンチップシステムにおけるハードウェアモジュールどうしの連携動作は, 多くの場合にこの 2 つの型の間の代入演算, あるいは型変換演算として抽象化して把握できることがわかる (図 3).

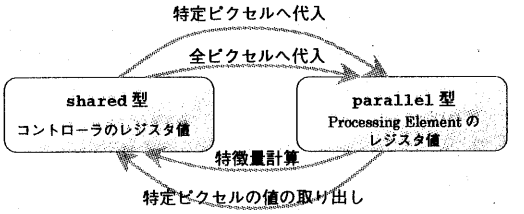

図 3 shared 型と parallel 型の相互演算

3.5.1 parallel 型から shared 型へ

例えば, 特徴量計算は, parallel 型から shared 型への型変換演算の一種である. 定型的な記述は次のように考えられる.

```
parallel(1) a;
shared(4) b;
vchip.output(1) = a;
b = vchip.out.moment0;
```

これは, 長さ 1 ビットの PE レジスタ領域に対応する parallel 型変数 a の, 各画素における値の総和を, shared 型変数 b に代入する演算である. いっぽう, 特定画素の PE のレジスタ値を shared 型変数に代入するという演算もある. これは SPE-C ではたとえば次のように記述する.

```
parallel(1) a;
shared(1) b;
b = a.pixelAt(3,6);
```

これは, parallel 型変数 a に対応する PE レジスタのビット領域のうち, 画素位置 (3,6) の PE における値をとりだして shared 型変数 b に代入するという演算である.

3.5.2 shared 型から parallel 型へ

たとえば

```
parallel(1) a;
shared(1) b;
b = 1;
a = b;
```

と記述した場合, まずコントローラのレジスタの 1 ビットが 1 になり, それがすべての PE のレジスタ (のうちの 1 ビット) に代入される. これは, shared 型の値を parallel 型 (全画素) に代入するという演算である.

さらに, 特定画素の PE のレジスタに値を設定したいときには,

```
parallel(1) a;
shared(1) b;
b = 1;
a.pixelAt(3,6) = b;
```

のように記述する. ここでは代入式の右辺を shared 型変数にしているが, 定数を含む式の場合も同じである.

3.6 SPE-C 言語の特徴のまとめ

SPE-C によるプログラミング例を図 4に, そのコンパイル結果の一部を図 5に示す. この例は, 入力された画像にぼかしフィルタを 5 回かけた後, その画像の重心の座標をコントローラのレジスタに格納する, という作業を繰り返すプログラムである. 上位プロセッサは, コントローラのレジスタ上の変数 out_data_x, out_data_y を読み出して結果を取り出すことができる. また, 上位プロセッサがコントローラのレジスタ上の変数 break_in の値を 1 に設定することで, この繰り返し作業を中断させることができるようになっている.

この例を見てもわかるように, SPE-C 言語は,

- parallel/shared 型を利用した変数宣言を混在させることで, 入力部, 並列演算部, 出力部 (特徴量計算回路), コントローラの動作を統一的に記述できる
- 各ハードウェアモジュールどうしの入出力は, parallel 型 / shared 型どうしの相互演算として記述するため, 抽象度が高い. また, 多くの関数名を覚える必要もない
- 構文は ANSI C 言語に沿って定義されているため, プログラムコードの見た目の雰囲気が多くの人にとって馴染みやすい

```
parallel(8)  image;      // 256 階調画像
shared(4)  out_data_x;   // 4 ビットの出力
shared(4)  out_data_y;   // 4 ビットの出力
shared(1)  break_in;     // 1 ビットの入力

void diffuse()
{
    shared(4)  i;
    for (i=0; i<5; i++)
    {
        image = (image * 4 +
          image.right + image.left +
          image.up + image.down) / 8;
    }
}

int main()
{
    break_in = 0;
    for(;;)
    {
        image = vchip.input(1) * 32;
        diffuse();
        vchip.output(8) = image;
        out_data_x = vchip.out.gx;
        out_data_y = vchip.out.gy;
        if (break_in)
            break;
    }
}
```

図 4 SPE-C プログラムの例

```
; ALLOC parallel [image]:0,1,2,3,4,5,6,7
; ALLOC shared [out_data_x]:0,1,2,3
; ALLOC shared [out_data_y]:4,5,6,7
; ALLOC shared [break_in]:8
FUNC_diffuse:
; ALLOC shared [i]:9,10,11,12
            MOV     CREG[9],0
            MOV     CREG[10],0
            MOV     CREG[11],0
            MOV     CREG[12],0
START_FOR_1:
            CMP     CREG[12],0
            JNCZ    EXIT_FOR_1
            CMP     CREG[11],1
            JC      CONTINUE_FOR_1
            CMP     CREG[10],0
            JNCZ    EXIT_FOR_1
            CMP     CREG[9],1
            JC      CONTINUE_FOR_1
            JMP     EXIT_FOR_1
CONTINUE_FOR_1:
            ; ALLOC parallel [tmp1]:8,9,10,11,12,13,14,15
            MOV     REG[8],0
            MOV     REG[9],0
                    REG[10],0
            MOV     REG[11],0
            MOV     REG[12],0
            MOV     REG[13],0
            MOV     REG[14],0
```

図 5 コンパイル結果の例

といった, ビジョンチップ用プログラミング言語として大きな意味を持つ特徴を備えている.

4. SPE-C コンパイル環境

最後に, SPE-C によるプログラミングを, S^{3}PE およびその制御アーキテクチャというハードウェア環境に対し実現可能とするコンパイラ環境について簡単に説明する. SPE-C コンパイル環境の構成は図6のようになっている.

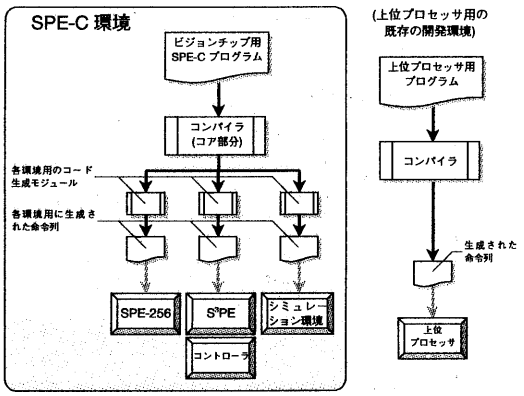

図 6 SPE-C コンパイル環境

この SPE-C コンパイラはコンパイラ自身のモジュール分割 (機種に依存する部分と依存しない部分の分離) により複数のターゲット環境に対してコード生成が行えるようになっており, 頻繁なビジョンチップの仕様変更に容易に可能な構造になっており, 実際にこれまで開発してきた S^{3}PE 及びその前バージョンである SPE システムに対応可能なものであることを確認している.

また, プログラミングの多くは論理的なデバッグを中心としたワークステーション上で行うことが予想され, システムのハードウェアを必要としないシミュレーション環境を一般的な端末上でソフトウェアだけで実現することがプログラミング開発効率を向上させることになる. このような考えのもとに, SPE-C コンパイラは, コード生成モジュールの一つとしてそのシミュレーション環境のためのコード生成を実現可能であり, SPE-C プログラムのアルゴリズムのチェックなどが容易に行える.

コンパイラ本体, および各機種用のコード生成モジュール以外に, アセンブラ, リンカといったツールも実装し (これらは yacc, lex, C++ などを用いて開発された), これらの動作を, SunOS Release 4.1.4 環境, および Windows 95 環境で確認している.

5. おわりに

本論文では, ビジョンチップシステムを実現する上で重要となる, プログラミング言語, およびコンパイラを中心とするソフトウェア開発環境について, 設計および実装を行い, その動作を確認した.

設計したプログラミング言語 SPE-C は, parallel(N) 型, shared(N) 型の 2 つのデータ型を基礎としている. これにより, ビジョンチップシステムにおいて入力部, 並列演算部, 出力部 (特徴量計算回路), コントローラの連携動作を, 型同士の代入演算や型変換演算として抽象化できることを示した.

今後の課題としては, より汎用的・一般的な演算を記述するためのデータ型の拡張, プログラミング言語としての形式的定義の整備, コンパイラとしての様々な最適化の実現, ビジュアルフィードバックシステムのためのループ時間間隔の概念の導入といったものが挙げられる.

参考文献

[1] 石川. 超高速・超並列ワンチップビジョンとその応用. 日本ロボット学会誌, Vol. 13, No. 3, pp. 31–34, 1995.

[2] C. Mead. *Analog VLSI and Neural Systems*. Addison-Wesleys, 1989.

[3] J.L. Wyatt, Jr., D.L. Standley, and W. Yang. The MIT vision chip project: analog VLSI system for fast image acqusition and early vision processing. *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 1330–1335, 1991.

[4] 小室, 鈴木, 石川. 超並列ビジョンチップアーキテクチャ. 信学技報, Vol. CPSY-19, pp. 63–69, 1995.

[5] 中坊, 石井, 石川. 超並列・超高速ビジョンを用いた 1ms ターゲットトラッキングシステ ム. 日本ロボット学会誌, Vol. 15, No. 4, pp. 417–421, 1997.

[6] 坂口, 小室, 石井, 石川. ビジョンチップのためのモーメント出力回路. 第 35 回計測自動御学会学術講演会予稿集, pp. 829–830, 1996.

[7] 村田, 松内, 石井, 石川. ビジョンチップシステムのための制御アーキテクチャ. 日本機械学会 ROBOMEC 講演会論文集, pp. 1089–1092, 1997.