超並列画像プロセッサのためのビットレベルコンパイラ

小室 孝[†] 鏡 慎吾[††]
石川 正俊[†] 片山 善夫[†††]

超並列画像プロセッサの一種であるビジョンチップのプログラミングを容易にするために開発したビットレベルコンパイラについて報告する．本コンパイラは，(1) コード生成効率，(2) モジュールの再利用性，(3) ユーザ利便性の向上を目標に設計されており，C ベースの文法，並列演算と逐次演算の自動振り分け，任意のビット長に対する一般記法，ビットレベルコード最適化などを特徴とする．

A Bit-level Compiler for Massively Parallel Image Processors

Takashi Komuro,[†] Shingo Kagami,[††] Masatoshi Ishikawa[†]
and Yoshio Katayama[†††]

We report on a bit-level compiler which was developed for easy programming of a vision chip, that is a kind of massively parallel image processor. The compiler is designed aiming improvements of (1) code generation efficiency, (2) reusability of modules, (3) user convenience, and features C-based grammer, auto division of parallel and sequential processing, general notation of arbitrary bit-length, and bit-level code optimization.

1. はじめに

画像による機械装置の自動化技術は，いわゆるマシンビジョンとして，工業製品の製造加工・検査などで古くから用いられてきた．さらに近年では，コンピュータの性能向上と画像認識技術の進歩により，機械が自ら認識・判断・行動するといった，機械の自律化・知能化に画像が利用されるようになってきている．

このような目的には，リアルタイムの画像認識が必須となるが，特に自動車や家電などの民生品においては，高速演算と小型・低コストを両立させる必要があり，専用プロセッサが採用されることもある．東芝は，車載用の画像認識 LSI “Visconti” [1] を開発しており，実際の自動車製品にも組み込まれている．これは，同社で開発したコンフィギュラブルなプロセッサコア MeP に VLIW コプロセッサ拡張を行ったものを 3 つ搭載したものであり，個々の MeP コアは，64 ビットの SIMD 演算命令（最大 8 並列）を同時に 3 つまで実行できる．

一方，アプリケーションによっては，より速い演算が必要とされる場合がある．そこで，並列度を大幅に高めた超並列プロセッサを画像処理に利用するアプローチも存在する．NEC は，一次元アレイ状に接続された 256 個の 8 ビット PE が SIMD 型の並列処理を行う IMAP-VISION ボード[2] を開発した．これは画像の列ごとに 1 つのプロセッサが対応し，画像処理を行単位で並列に実行できる．

さらに並列度を高めて，画像の画素ごとに 1 つの PE を対応させることもできる．このような構成は古くから研究されてきたが[3]，近年では半導体集積化技術の進歩により，ワンチップに搭載することも可能となった[4),5]．

これらの超並列処理を用いたアプローチは，実装できる画像処理はある程度限定されるが，高速演算という点で有利である．したがって，画像処理の柔軟性よりも，高速性が強く求められるような場合に威力を発揮する．

例として，イメージセンサの画素ごとに PE を搭載したビジョンチップ[6] があげられる．これは，超並列処理による演算の高速性に加え，画像の転送を不要にすることで，1000 フレーム毎秒の高速視覚処理を実

† 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†† 東北大学大学院情報科学研究科
Graduate School of Information Sciences, Tohoku University
††† 株式会社 PFU
PFU Limited

現するものである．本ビジョンチップは，ロボットの高速制御などへの応用が期待されている．

上記ビジョンチップを含め，画素レベルの並列度を持つ超並列画像プロセッサでは，個々の PE にそれほど大きな回路を設置できないため，演算をビットシリアルに行っているものが多い．また，個々の PE が持つ記憶素子（レジスタやメモリ）の容量はあまり大きくない．超並列画像プロセッサのソフトウェア開発環境を作成するうえでは，超並列演算への対応に加えてそのことに留意する必要がある．

本論文ではその一実施例として，上記ビジョンチップのためのビットシリアル超並列演算に対応したコンパイラを開発したので報告する．

2. 関 連 研 究

2.1 ビジョンチップ

本論文で紹介するコンパイラは，以下に示すビジョンチップ[7] とそれを用いた実時間視覚処理システム[8] を対象にしている．

図 1 にビジョンチップの構成を示す．処理要素（PE）が格子状に並べられ，それぞれ光検出器（PD）が備え付けられている．PD からの光強度信号はそのまま対応する PE に送られる．各 PE は上下左右の近傍 PE と接続されており，1 ビット幅でデータの受け渡しを行う．行および列ごとに共通の 1 ビット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．演算処理の最終結果は右端 PE に接続された列累積加算器を経由してスカラー量で出力される．

PE はビットシリアルの ALU とビット単位でアクセス可能な 24 ビットのメモリから構成されており，外部のコントローラから制御される．すなわち，すべての PE は同時に同一の命令を実行する．PD からの入力や列/行に共通の入力はメモリ空間にマップされており，メモリアクセスによって取り込める．PD と PE は 1 対 1 に対応しているので，撮像された画像のデータは直接 PE アレイに渡されることになる．画像の一部だけを処理したい場合は，PE の演算を用いてマスク処理を行う．近傍 PE とのデータの受け渡しはラッチを介して行われ，PE ごとに近傍入力元を選択できるようになっている．ラッチをイネーブルにした状態で近傍演算を行うことで，隣接 PE の ALU が連結され，総和などのグローバル演算も効率的に実行することができる．

このビジョンチップを制御するコントローラの構成を**図 2** に示す．これは命令コードに基づいてビジョンチップの制御信号を生成する SIMD パイプラインを RISC 型のマイクロプロセッサに統合したものである．この構成をとることにより，プログラム制御や整数演算命令もこのコントローラ上で行うことができる．コントローラとビジョンチップの間のデータの受け渡しは，コントローラ上のデータメモリにマップされた I/O を通じて行う．

上記ビジョンチップとコントローラ，センサ制御用 DA 変換器，外部インタフェースなどを搭載したシステムボード（VCS-IV）が開発されている．**図 3** に写真を示す．このシステムの上で各種の視覚処理プログラムが動作しているほか，リアルタイム計測などに応

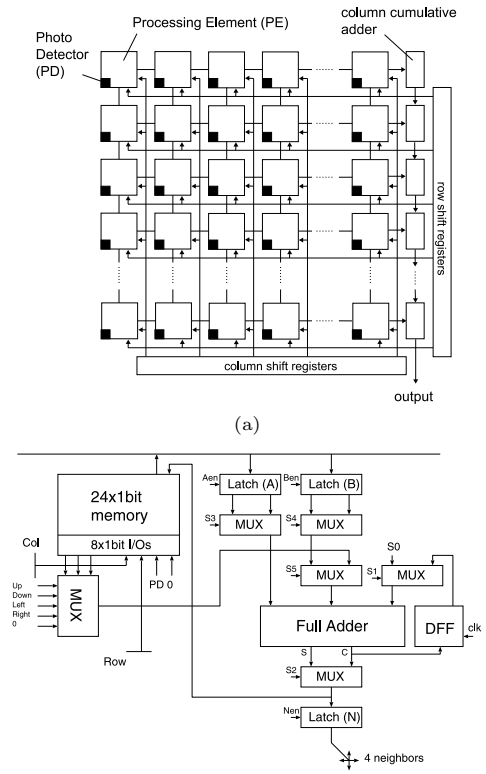


図 1 ビジョンチップの構成：(a) 全体 (b) PE
Fig. 1 Structure of the vision chip: (a) the whole (b) the PE.

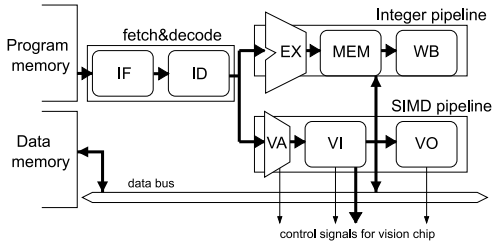


図 2 ビジョンチップコントローラの構成
Fig. 2 Structure of the Vision Chip Controller.

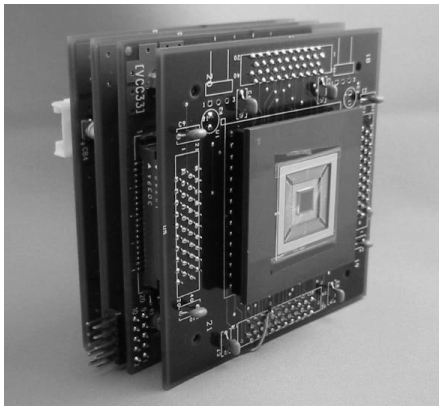


図 3 VCS-IV システムの写真
Fig. 3 Photo of the VCS-IV system.

用されている．

2.2 ビジョンチップ用プログラミング言語 SPE-C

ビジョンチップが普及し，広く使われるようになるためには，高級言語によるプログラム開発環境が必須である．しかし，ビジョンチップとコントローラからなるシステムは，画素並列と逐次の 2 種類の演算が混在しており，通常のプロセッサ向きに設計されたプログラミング言語をそのまま使うことはできない．それに対し，ビジョンチップ用のプログラミング言語として SPE-C が提案されている[9]．

これは，通常の C 言語に新たに parallel というデータ型を付加したものであり，このデータ型に対する演算は画素並列に行われる．これによりユーザは，ハードウェアの構成を意識することなく，最小限の移行コストでビジョンチップのプログラミングができることになる．parallel 型は画素内のメモリを効率的に利用するために，任意のビット長が指定できるようになっている．

この SPE-C に対応したコンパイラも過去に作られており[10),11)]，プログラミングの容易化という当初の目的はある程度達成されている．しかしながら，コードの最適化がまったく行われていないなど，出力されるコードの質という点では十分とはいえず，ビジョンチップの性能をフルに活用できていなかった．また，コードの再利用性が悪く，作成したプログラムをライブラリとして蓄積することが難しかった．

2.3 ビットレベルコンパイラの例

SIMD 型の並列演算を行う超並列プロセッサには，その構造に適したプログラミング言語が多数提案されているが，これらは主に科学技術計算の高速化を目的としており，Fortran などで書かれた逐次プログラムを，データや処理を細かく分割し，多数の PE に割り当てることで並列化するものであった．

一方，画像処理においてはデータや処理は画素並列とそれ以外に大きく二分されるため，逐次プログラムの並列化ではなく，画素並列処理に対応したデータ型を用意することで，より簡潔な形で並列プログラムを記述できる．Herbordt らは，BitPlane，CharPlane，IntPlane などの画素並列のデータ型を持つプログラミング言語 ICL とそのコンパイラを開発しているが[12)]，比較的規模の大きい超並列プロセッサを対象としているため，使用メモリ量を抑えることは特に考えられておらず，任意のビット長のデータを扱えるようにはなっていない．

前述の IMAP VISION ボードにもソフトウェア開発環境として，1DC という列並列のデータ型を持つプログラミング言語に対応したコンパイラが用意されているが，やはり任意のビット長のデータを扱えるようにはなっていない．

任意のビット長のデータを扱い，ビットレベルのコード最適化を行っているビットレベルコンパイラの例はむしろハードウェア記述言語の分野に見られる．Handel-C[13)] は，C を基本とした文法でハードウェアを設計することができる言語で，Celoxica 社より製品化されている．この言語では，たとえば int 5 は 5 ビットの整数を表すなど，データ型のビット長を指定することができるようになっている．ビット長を指定しない場合は，コンパイラが必要なビット長を推定し，それが用いられる．

これらは，超並列画像プロセッサのコンパイラを開発するうえで参考になるが，並列演算と逐次演算の混在や限られたメモリ量など，超並列画像プロセッサ特有の問題もあることから，新たに考慮すべき点も多い．

3. 設計方針と言語仕様

上記問題の解決のため，新たにビジョンチップ用コンパイラの仕様設計を行った．設計にあたっては，以下の項目の向上を目標に掲げた．

(1) コード生成効率
(2) モジュールの再利用性
(3) ユーザ利便性

(1) に対してはビジョンチップの演算構造に合った最適化が必要である．(2) については，ビットシリアル演算では，同じ演算でもビット長が異なると別のコードとなってしまい，再利用性が悪いという問題があった．たとえば，従来の SPE-C 対応のコンパイラでは，parallel 型変数のビット長を宣言時に決定する必要が

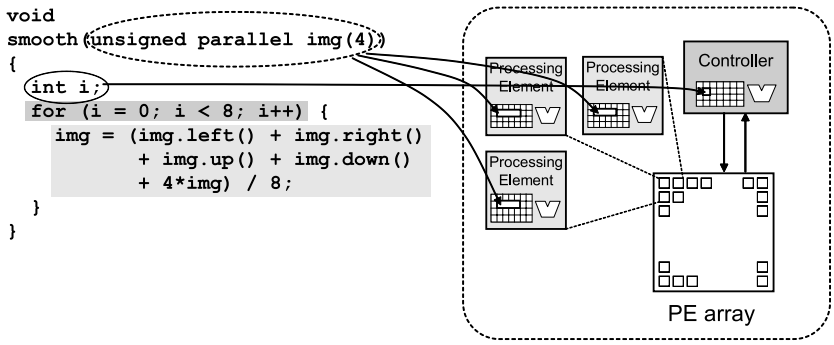


図 4 parallel 型の導入
Fig. 4 Adoption of parallel-type.

あり，ライブラリ関数はビット長ごとに用意する必要があった．したがって，任意のビット長を扱う一般的な記法が必要である．(3) に対しては，既存のプログラミング言語からのスムーズな拡張が必要である．

これらの目標を達成するため，前述の SPE-C をベースに言語から再設計を行った．SPE-C と同様に，C ベースの文法となっているが，ビジョンチップ向けの拡張として以下のような特徴を持たせている．なお，詳細な仕様は付録に掲載する．

3.1 parallel 型の導入

通常の C 言語に新しく parallel 型を導入し，ビジョンチップで行う超並列処理とコントローラで行う逐次処理を同時に記述できるようにした．

コンパイラはソースコードに記述された演算のうち，画素並列に行うべきものと逐次で行うべきもの，すなわちビジョンチップ本体で行うべきものとコントローラで行うべきものを自動的に判別し，それぞれに適切なコードを出力する．

parallel 型変数は，入力画像や演算中の中間画像を格納するものであり，画像のすべての画素の画素値で構成される．ビジョンチップでは，parallel 型変数に対する演算は，画素ごとに対応する PE が処理を行う．すべての PE は同時に同一の命令を実行するため，画像全体に対して画素単位で共通の演算が実行される．

parallel 型は，型名または変数名の後ろに括弧で数字を入れることにより，任意のビット長が指定できる．これにより，効率的なコード生成が可能となり，貴重なメモリ資源を有効利用できるようになっている．

関数は，parallel 型を引数や返り値にとることもできる．限られたメモリを無駄にしないように，parallel 型変数の実引数は自動的に参照渡しとなる．

図 4 に説明図を示す．図には，4 ビットグレースケールの画像に対し，平滑化のアルゴリズムを適用する関数の例が示されている．仮引数である img に画像データが格納されており，すべての画素で隣接画素の値との重み付き平均を計算し，img を更新する処理をループで 8 回繰り返している．left/right/up/down は隣接画素の式の値を返す組み込み関数である．組み込み関数の詳細は 3.5 節で述べる．img には parallel 型変数の実引数が参照渡しされるため，処理結果はそのまま実引数に反映される．ループの制御はコントローラで実行される命令に，画像に対する演算はビジョンチップ本体で実行される命令に展開される．

3.2 不定長 parallel 型

parallel 型の変数のビット長は宣言時にプログラマが指定するようになっているが，宣言時にビット長を指定しない不定長 parallel 型も利用できるようにした．不定長 parallel 型の変数は，コンパイラがデータフロー解析によりビット長を推定し，その値が適用される．

3.3 不完全 parallel 型

関数の引数や返り値にビット長を指定しない parallel 型を記述することができる．これを不完全 parallel 型と呼ぶ．これにより，任意のビット長を扱う関数を統一的に記述することが可能となる．

ビットシリアル演算では，同じ演算でもビット長が異なると別のコードとなってしまうので，通常のサブルーチンコールによる実装では，とりうるビット長ごとにコードを用意しておかなければならない．

そこで，関数に inline 修飾詞をつけることで，インライン展開されるようにした．これにより，関数の呼び出しごとにコードが作成されるため，異なるビット長で別のコードが生成されても問題は起こらなくなる．

なお，関数内の処理が仮引数を 1 ビットずつ取り出しながら行われる場合は，非インライン関数でも不完全パラレル型仮引数をとることができる．これには，後述の組み込み関数 getbit(s)/setbit(s) を使用する．

3.4 パラレル条件文

本コンパイラは条件式が parallel 型の if 文をサポートしている．これは，PE ごとに条件の成否を判定し，条件が成り立つ PE のみが実行部に書かれた内容を実行するというものである．

通常，if 文は条件分岐を含むコードに展開されるが，パラレル条件文では PE ごとに条件の成否が異なるため，展開されるコードは条件分岐を使わず演算の組合せで構成される．そのため，実行部に記述できるのは関数呼び出しをともなわない式文および if 文のみに限定される．また，parallel 型以外の変数への代入は原理的に不可能なので許されない．

3.5 組み込み関数

ハードウェア固有の機能や，基本演算の組合せでは効率良く記述することができない演算を組み込み関数という形で提供している．

たとえば，隣接画素の式の値を返す up/down/left/right，parallel 型の式の符号や属性を返す sign/length/issigned，parallel 型の式の特定のビット値を返す getbit/getbits，parallel 型変数の特定のビットに値をセットする setbit/setbits，絶対値や平方根を計算する pabs/psqrt などが用意されている．

組み込み関数では，left(a) のような通常の関数呼び出しの記法のほか，a.left() というメソッド呼び出し型の記法もサポートされている．これにより，a.left().up() といった記述が可能となり，可読性が増すことになる．

4. ビジョンチップ用コンパイラ vcc

上述の仕様に基づいて，コンパイラの開発を行った．開発したコンパイラは，前述のビジョンチップシステムを対象にしているが，ビジョンチップに限らず，類似の構造を持つ超並列画像プロセッサにも流用可能である．

4.1 基 本 構 成

以下に開発したコンパイラの構成要素とその役割を示す．

コンパイラ制御部

コンパイラ各部の起動，中間コードファイルの作成などを行う．

プリプロセッサ

ソースプログラム中の前処理指令を処理する．これは通常の C 言語用のものを用いる．

構文解析部

プリプロセッサで前処理されたソースプログラムの構文解析を行い，中間コード I に変換し，出力する．

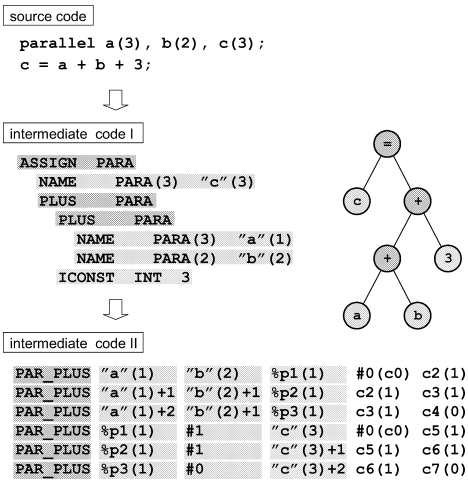


図 5 中間コードの生成例
Fig. 5 Example of generated intermediate codes.

ビットシリアル化部

中間コード I を読み込み，並列演算部分をビジョンチップに合わせて 1 ビット単位で表現した中間コード II に変換し，出力する．

コード生成部

中間コード II を読み込み，変数割付けを行い，アセンブリ言語で記述されたオブジェクトコードを生成する．

アセンブラ

アセンブリ言語を再配置可能バイナリファイルに変換する．

リンカ

再配置可能プログラムを実行形式プログラムに変換する．

図 5 に中間コードの生成例を示す．

中間コード I では，演算は変数単位で記述されており，解析・変更が容易な式木表現となっている．中間コード II では，演算はビット単位で記述されているが，変数はまだ抽象化されており，具体的なハードウェアリソースに割り当てられていない．ビットシリアルの加減算に対応するため，コードにはキャリの情報も付加されている．中間コード I，II の説明は付録に掲載する．

4.2 最 適 化

最適化はコンパイラが出力するコードの質を決める重要な処理である．ビジョンチップが行っているビットシリアル演算に対応するため，本コンパイラでは，最適化をビットレベルで行っている．以下で具体的な

手法について説明する．

4.2.1 ビットシリアル化前の最適化

ビットシリアル化前の中間コード I に対しては，式木の最適化とデータフロー解析による不定長 parallel 型変数のビット長決定を行っている．

式木の最適化では，具体的には以下の最適化を行っている．

- 定数畳み込み
- 冗長演算削除
- 可換演算子の組み変え
- 複合代入演算の “単純代入 + 二項演算” による置き換え
- “単純代入 + 二項演算” の複合代入演算による置き換え

可換演算子の組変えは，たとえば (a+b)+(c+d) → a+b+c+d のように演算の順序を変えることで，テンポラリで使用する PE 内メモリの量（以下，テンポラリメモリと呼ぶ）を抑えているといったものである．変更前の式では，演算の途中で (a + b) と (c + d) の両方を保持する必要があるのに対し，変更後の式では，そのときの加算結果のみを保持すればよい．PE 内のメモリリソースが限られているビジョンチップにおいて，このようなテンポラリメモリの削減は，実装可能なアルゴリズムの幅を増やせるという点で重要である．

このほか，ビット長が短いオペランドを前にして，ビットレベルでの演算回数が少なくなるようにしたり，加減算の場合に unsigned parallel(1) 型のオペランドを適当に分散させて，キャリを利用した演算（後述）を利用できるようにしたりしている．

“単純代入 + 二項演算” の複合代入演算による置き換えは，不定長 parallel 型変数のビット長決定を行う際に解析が容易になることから，このような変形を行っている．

逆に，複合代入演算の “単純代入 + 二項演算” による置き換えは，a += b+c のような演算を，後の最適化においてテンポラリメモリを使わない形（a += b, a += c）に変形するための準備として，不定長型のビット長決定の後に行う．

演算結果のビット長は，演算結果のとりうる値の範囲を追跡することで求めている[14]．**図 6** に適用例を示す．図中の conventional は，値の追跡を行わず演算ごとにオペランドのビット長から演算結果のビット長を計算した場合であり，従来の SPE-C 対応のコンパイラはこのようにしてビット長を決定していた．proposed は，本コンパイラで採用した値の追跡を行った場合である．

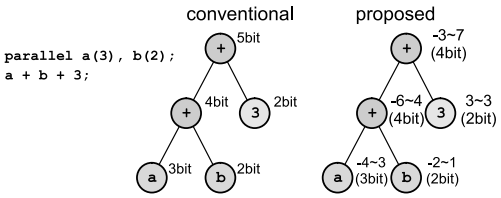


図 6 最大・最小値の追跡による演算ビット長の削減例
Fig. 6 Example of bit-length reduction by max-min tracking.

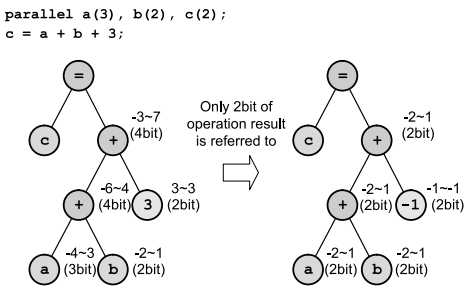


図 7 参照ビット長による演算ビット長の削減例
Fig. 7 Example of bit-length reduction by max-min tracking.

算術演算，シフト演算では，各オペランドの最大値と最小値が分かれば，演算結果の最大値と最小値を求めることが可能である．ビット論理演算では各オペランドの最大値と最小値を正確に求めることはできず，広めに見積もった範囲を示すことになる．

このような方法は，ハードウェア合成の分野ですでに用いられているが[15]，本コンパイラでは，オペランドの一方が定数の場合はそれを考慮したアルゴリズムを適用してさらに範囲を絞っているなどの違いがある．

また，前述の組み込み関数においても他の演算と同様に最大値・最小値が計算される．

上記方法に加え，演算結果のうち実際に参照されるビットを調べることで，最終的なビット長を決定している．**図 7** に適用例を示す．図中の矢印の左が適用前，右が適用後のビット長の計算結果であり，演算結果のうち下記 2 ビットしか参照されないため，途中の不要なビットの演算が削除されている．

不定長 parallel 型変数のビット長は，次のようにして決定している．

(1) 不定長 parallel 型変数の値の範囲を $[-\infty, \infty]$ とする．
(2) データフロー解析を行い，変数のとりうる最大・最小値の伝播を行う．
(3) 代入される値と参照ビット長の小さい方をビット長とする．
(4) 不定長 parallel 型変数のビット長が変化しなく

なるまで，(2)〜(3) を繰り返す．

収束するまで繰り返しているのは，不定長 parallel 型変数に代入している式に，他の不定長 parallel 型変数が含まれている場合などでは，1 回ですべての不定長 parallel 型変数のビット長が決定できないからである．ループ中に不定長 parallel 型変数の相互代入がある場合などでは，上記アルゴリズムを適用しても，ビット長が決定できないため，エラーとなる．

4.2.2 ビットシリアル化時の最適化

ビットシリアル化時には，以下のような演算子固有の最適化を行っている．演算やテンポラリ変数のビット幅は，前述の最大・最小値の追跡によって決定される．

- 乗算のビットシリアル化では，ビット長が多い方を被乗数とすることによって，命令数を削減する．
- 加減算のキャリの初期値を利用して a＋b＋c（c は unsigned parallel(1) 型）などの計算を 1 回の加減算で実行する．
- 積和および二乗を自動検出し，効率の良いコードを出力する．

4.2.3 ビットシリアル化後の最適化

ビットシリアル化後の中間コード II に対しては，無効ブロック削除などのブロック単位の最適化のほか，ビットレベルの最適化を行っている．最適化をビット単位で行うことにより，変数単位で最適化を行う通常のコンパイラに比べ，使用メモリやコードの削減をより細かく行うことができる．

ビットレベルの最適化は大きく以下の種類に分類できる．以下，中間コード II の各行のデータの組をタプルと呼ぶ．

- タプル単体の最適化
- 2 タプルの最適化
- 複写伝播
- 不要タプル削除
- タプル融合

タプル単体または 2 タプルの最適化は，特定の条件を満たすタプルを，他の最適化が容易な形に変換するものである．たとえば，変数を定数に変換したり，加算を論理演算に変換したりすることで，不要タプル削除やタプル融合の条件を満たしやすくしている．また，加減算におけるキャリ値を考慮することで，オペランドの入れ替えをさまざまな形で行い，他のタプル単体または 2 タプルの最適化の条件を満たすようにしている．

タプル融合は，tmp ← a⊕b，c ← tmp を c ← a⊕b に，また，tmp ← a⊕b，d ← tmp⊗c を d ← a⊕b，d ← d⊗c に変形することで，コードやテンポラリメモリを削減する（⊕，⊗ は演算子）．

```
unsigned parallel x(4), y(1);
y = x >= 5;

PAR_PLUS   "x"(1)    #0       %p1(0)    #1(c1)  c2(1)
PAR_PLUS   "x"(1)+1  #1       %p2(0)    c2(1)   c3(1)
PAR_PLUS   "x"(1)+2  #0       %p3(0)    c3(1)   c4(1)
PAR_PLUS   "x"(1)+3  #1       %p4(0)    c4(1)   c5(1)
PAR_PLUS   #0        #1       %p5(1)    c5(1)   c6(0)
PAR_ASSIGN ~(%p5(1))          "y"(2)

                  ⇩

PAR_OR     "x"(1)+1  "x"(1)    "y"(2)
PAR_AND    "x"(1)+2  "y"(2)    "y"(2)
PAR_OR     "x"(1)+3  "y"(2)    "y"(2)
```

図 8 ビットレベル最適化の適用例
Fig. 8 Example of bit-level optimization.

2 タプルの最適化やタプル融合は，タプルの移動をともなうものを含む．

図 8 にビットレベル最適化の適用例をあげる．

最適化において，タプル単体の最適化により定数との加算は論理演算に変換され，タプル融合によりテンポラリメモリが削除されている．その結果，コードサイズ，テンポラリメモリともに大幅に削減されている．

4.2.4 変数割付け時の最適化

自動変数およびテンポラリのレジスタ/メモリへの割付けは，中間コード II からアセンブリコードを生成する際に行っている．parallel 型自動変数を構成する各ビットは，PE 内のメモリに割り付けられるが，必ずしも連続したアドレスに割り付けられるわけではなく，ビット単位で個別に割り付けられる．ただし，実引数として使用される場合や getbit(s)/setbit(s) で使われている場合は，変数単位で連続したアドレスに割り付けられる．これは，実引数は参照渡しされるため，また getbit(s)/setbit(s) ではビット位置からアドレスを直接計算しているため，ビット並びが連続している必要があるからである．

PE 内メモリのアドレッシングは，ビジョンチップコントローラが持つレジスタ値をベースアドレスとし，それにオフセットを加えたアドレスがアクセスされる．その機構を用いて PE 内メモリにスタックを構成し，自動変数およびテンポラリを保存している．

変数の生死解析の結果をもとに，空きアドレスを小さい方から探索して割り付けている．外部変数/仮引数に対しても同様に生死解析を行い，空いている場合はその領域に自動変数/テンポラリを，スタック領域より優先的に割り付ける．

5. 性能評価

開発したコンパイラの性能評価のため，いくつかの

画像処理アルゴリズムを実装したプログラムを作成し，コンパイルを行った．比較対象として，コンパイラで最適化を行わなかった場合の結果と，手動でアセンブリコードを作成した場合の結果も示す．後者における最適化はアルゴリズムが可読できる程度にとどめるとする．これは，標準的な開発者が手動で最適化できる限界とほぼ一致する．

実装した画像処理は，以下のとおりである．

平滑化（sm）

6 ビットグレースケールの画像に対し，以下の式で表される平滑化のアルゴリズムを適用する．

$$g(x,y) = \{4f(x,y) + f(x-1,y) + f(x+1,y) + f(x,y-1) + f(x,y+1)\}/8$$

トラッキング（tr）

バイナリ画像に対し，以下の式で表される 1 フレーム分のトラッキングのアルゴリズム[16] を適用する．

$$W_{k+1}(x,y) = g_k(x,y) \cup g_k(x+1,y) \cup g_k(x-1,y) \cup g_k(x,y+1) \cup g_k(x,y-1)$$

$$g_{k+1}(x,y) = W_{k+1}(x,y) \cap f_{k+1}(x,y)$$

モーメント計算（mo）

バイナリ画像に対し，以下の式で表される画像特徴量であるモーメントを 0 次から 2 次まで計算するアルゴリズム[17] を適用する．

$$m_{pq} = \sum_{x=1}^{N} \sum_{y=1}^{N} x^p y^q f(x,y)$$

ラベリング（la）

バイナリ画像に対し，二分探索によるラベリングのアルゴリズム[18] を 1 ラベル分適用する．

図形描画（dr）

与えられたパラメータに基づき，楕円を描画するアルゴリズム[17] を適用する．

距離計算（di）

与えられた座標 (x_o, y_o) からの距離を画素ごとに以下の式で計算する．

$$d = \sqrt{(x-x_o)^2 + (y-y_o)^2}$$

それぞれ生成されたコードに含まれる命令数を**図 9** に，実行サイクル数を**図 10** に，テンポラリで使用した PE 内のメモリ量を**図 11** にグラフで示す．命令数および実行サイクル数のグラフの横軸は，最適化を行わなかった場合の命令数で正規化している．グラフの横の数字は実際の命令数/実行サイクル数である．手動との比較を容易にするため，関数はすべてインライン展開するものとした．コンパイラで最適化を行わない場合，変数割付けは生死解析を行わずに，単純に出現している間はその変数は生きていると見なして割り付けている．

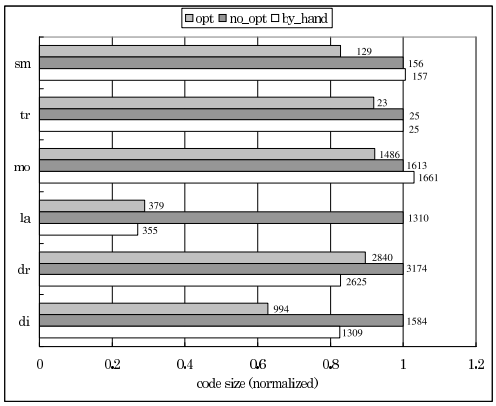


図 9 命令数の比較
Fig. 9 Code size comparison.

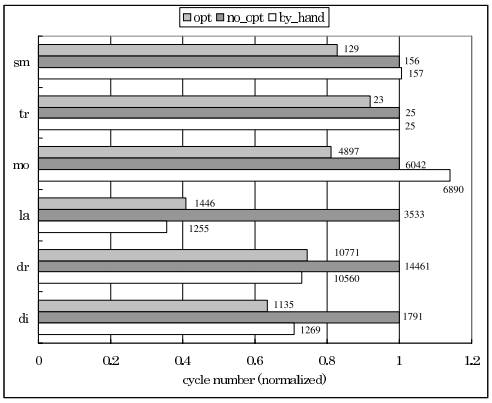


図 10 実行サイクル数の比較
Fig. 10 Cycle number comparison.

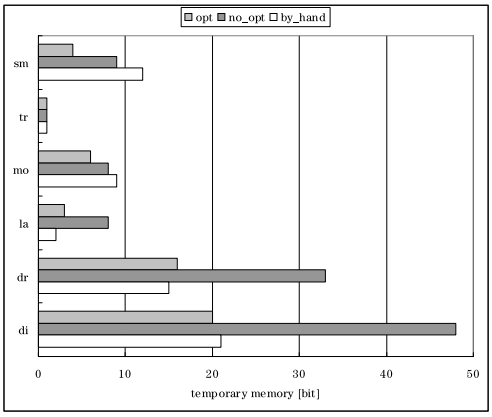


図 11 テンポラリメモリ量の比較
Fig. 11 Temporary memory comparison.

この結果から，本コンパイラで行っている最適化が特にテンポラリメモリ量の削減に高い効果があることが示された．従来の SPE-C 対応のコンパイラでは最適化が行われていなかったことを考えると，本コンパイラによって出力コードの質が大幅に高められたといえる．また，手動でコードを作成した場合に比べても遜色ない質のコードが生成されており，自動化による労力の削減を考えれば，開発環境として導入する価値が十分にあるコンパイラであるといえる．

6. むすび

画素レベルの並列度を持ち，ビットシリアルに演算を行う超並列画像プロセッサに向いたビットレベルコンパイラの提案と開発事例の紹介を行った．ビットシリアル演算と超並列処理の組合せは，低階調のグレースケール画像や 2 値化後のバイナリ画像を頻繁に扱うことが多いマシンビジョン・ロボットビジョンにおいて，特に高い計算効率を発揮する．本コンパイラの開発が，ビジョンチップや超並列画像プロセッサの実用化を促進することを期待する．

参考文献

1) Tanabe, J., Taniguchi, Y., Miyamori, T., Miyamoto, Y., Takeda, H., Tarui, M., Nakayama, H., Takeda, N., Maeda, K. and Matsui, M.: Visconti: Multi-VLIW Image Recognition Processor based on Configurable Processor, *Proc. 25th IEEE Custom Integrated Circuits Conference* (*CICC2003*), pp.185–188 (2003).
2) Fujita, Y., Kyo, S., Yamashita, N. and Okazaki, S.: A 10 GIPS SIMD processor for PC-based real time vision applications — architecture, algorithm implementation and language support, *Proc. 4th IEEE International Workshop on Computer Architecture for Machine Perception* (*CAMP'97*), pp.22–32 (1997).
3) Maresca, M., Lavin, M.A. and Li, H.: Parallel Architectures for Vision, *Proc. IEEE*, Vol.76, No.8, pp.970–981 (1988).
4) Gayles, E.S., Kellihr, T.P., Owens, R.M. and Irwin, M.J.: The design of the MGAP-2: A micro-grained massively parallel array, *IEEE Trans. Very Large Scale Integration* (*VLSI*) *Systems*, Vol.8, No.6, pp.709–716 (2000).
5) Gealow, J.C. and Sodini, C.G.: A pixel-parallel image processor using logic pitch-matched to dynamic memory, *IEEE Journal of Solid-State Circuits*, Vol.34, No.6, pp.831–839 (1999).
6) 石川正俊，小室　孝：ディジタルビジョンチップとその応用，電子情報通信学会論文誌 C，Vol.J84-C, No.6, pp.451–461 (2001).
7) 小室　孝，鏡　慎吾，石川正俊：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，電子情報通信学会論文誌 D-II，Vol.J86-D-II, No.11, pp.1575–1585 (2003).
8) 鏡　慎吾，小室　孝，渡辺義浩，石川正俊：ビジョンチップを用いた実時間視覚処理システム VCS-IV，電子情報通信学会論文誌 D-I，Vol.J88-D-I, No.2, pp.134–142 (2005).
9) 松内良介，村田達也，石井　抱，石川正俊：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，情報処理学会研究報告，Vol.ARC-125, No.7, pp.37–42 (1997).
10) 鏡　慎吾，小室　孝，中坊嘉宏，石井　抱，石川正俊：ビジョンチップ評価システムとソフトウェア開発環境，第 19 回日本ロボット学会学術講演会予稿集，pp.387–388 (2001).
11) 向坂直久，豊田晴義，水野誠一郎，中坊嘉宏，石川正俊：超高速インテリジェントビジョンシステム—2 眼化による 3D 計測，第 8 回画像センシングシンポジウム講演論文集，B-29, pp.173–178 (2002).
12) Herbordt, M., Burrill, J. and Weems, C.: Making a Dataparallel Language Portable for Massively Parallel Array Computers, *Proc. 4th IEEE Int. Workshop on Computer Architecture for Machine Perception* (*CAMP'97*), pp.160–169 (1997).
13) Page, I.: Constructing hardware-software systems from a single description, *Journal of VLSI Signal Processing*, Vol.12, No.1, pp.87–107 (1996).
14) 山野高将，小室　孝，鏡　慎吾，石川正俊：ビジョンチップコンパイラのビットレベル最適化手法，情報科学技術フォーラム 2003 一般公演論文集第 1 分冊，pp.177–178 (2003).
15) Ogawa, O., Takagi, K., Itoh, Y., Kimura, S. and Watanabe, K.: Hardware Synthesis from C Programs with Estimations of Bit Length of Variables, *IEICE Trans. Fundamentals*, Vol.E82-A, No.11, pp.2338–2346 (1999).
16) 石井　抱，石川正俊：1 ms ビジュアルフィードバックシステムのための高速対象追跡アルゴリズム，日本ロボット学会誌，Vol.17, No.2, pp.195–201 (1999).
17) Komuro, T., Senjo, Y., Sogen, K., Kagami, S. and Ishikawa, M.: Real-time Shape Recognition Using a Pixel-parallel Processor, *Journal of Robotics and Mechatronics*, Vol.17, No.4, pp.410–419 (2005).
18) 渡辺義浩，小室　孝，鏡　慎吾，石川正俊：ビ

ジョンチップのためのマルチターゲットトラッキングとその応用，電子情報通信学会論文誌 D-II, Vol.J86-D-II, No.10, pp.1411–1419 (2003).

付　　録

A.1 文法仕様

本コンパイラは**表 1** に示す文法に対応している．基本的に C 言語に parallel のデータ型を追加したものであるが，浮動小数点数や文字（列），構造体などはサポートしていない．

parallel 型は，型名または変数名の後ろに括弧で数字を入れることにより，任意のビット長が指定できる．ビット長を指定しない parallel 型も記述できる．

parallel 型の変数は，自動変数，静的変数，外部変数のいずれの記憶クラスもとることができる．

算術演算，ビット演算，シフト演算の演算結果の型は，オペランドに parallel 型の式が含まれていれば parallel 型に，そうでなければ int 型になる．結果が parallel 型の場合，符号の有無とビット長は，値を壊さないことを原則とし，できるだけビット長が小さくなるように定める．

比較演算，論理演算の演算結果は，オペランドに parallel 型が含まれていれば 1 ビットの unsigned parallel 型に，そうでなければ int 型になる．

sizeof 演算子は，int 型や配列に対しては通常の C と同じ働きをするが，parallel 型変数に対してはビット長を定数として返す．

ポインタ，配列は int 型に対してのみ用意されており，parallel 型のポインタ，配列は存在しない．したがって，アドレス演算子&，間接演算子 *，添字演算子 [] は，parallel 型には適用できない．

代入演算は，左辺の型に合わせて型変換される．int 型から parallel 型への変換は可能だが，parallel 型から int 型への変換は原理的に不可能なので，禁止されている．変換後の型のビット長が元の型のビット長より小さい場合，あふれた上位ビットは切り捨てられる．逆に変換後の型のビット長が元の型のビット長より大きい場合，余った上位ビットは元の型が符号付きか符号なしかによって符号拡張またはゼロ拡張される．

while 文や for 文，switch 文などの括弧内の式は，int 型でなくてはならない．ただし，if 文に限ってのみ parallel 型の条件式もサポートしている．

関数は，parallel 型を引数や返り値にとることもできる．parallel 型変数の実引数は自動的に参照渡しとなる．

表 1 コンパイラが対応する文法
Table 1 Target syntax of the compiler.

型	整数型　int signed int unsigned int 並列型　parallel signed parallel unsigned parallel 派生型　配列型 ポインタ型 関数型
型修飾子	volatile const inline
記憶域指定子	extern static auto register
定数/リテラル	整数定数（2/8/10/16 進数定数）
演算子	[] () ++ −− & * + − ~ ! sizeof キャスト * / % + − << >> < > <= >= == != & ^ | && || ? : = *= /= %= += −= <<= >>= &= ^= |= ,
文	式文 if 文 switch 文 while 文 do-while 文 for 文 goto 文 continue 文 break 文 return 文 asm 文
組込関数	up down left right length issigned sign getbit setbit など
外部宣言	関数定義 外部オブジェクト定義 宣言

A.2 中間言語 I の説明

中間コード I は，

- 変数や関数の定義
- 外部名の定義
- ブロックの範囲
- 式木
- 分岐

などを表す部分からなる．以下では，式木に関してのみ述べる．

各行は，OP コード，オペランドの順に記述される．オペランドの有無や個数は，OP コードに依存する．OP コードには，

- 二項演算子
- 単項演算子
- 末端節

などの種類がある．式木の表現は前置記法になっており，二項演算子の場合，演算子中間コードに続いて，被演算子中間コードが生成される．

二項演算子には，PLUS，ASSIGN などがあり，それぞれ加算，代入を表す．単項演算子にはマイナスを表す UMINUS などがある．

末端節には，変数を表す NAME や定数を表す ICONST などがある．NAME 末端節は，型オペランドと名前参照オペランドをとり，名前参照オペランドは変数名を表す．ICONST 末端節は，型オペランドと数値オペランドをとり，数値オペランドは値を表す．

型オペランドは，INT，PARA などがあり，それぞれ int 型，parallel 型を表す．PARA の後ろの括弧の中の数値は，ビット長が指定された parallel 型を表す．“” で囲まれた記号は変数名を表し，その後の括弧の中の数値は変数番号で，名前が同じ変数が 2 つ以上存在する場合に区別できるようにしている．

A.3 中間言語 II の説明

中間コード II は，

- 変数や関数の定義
- 外部名の定義
- テーブル
- ブロックの範囲
- 演算処理

などを表す部分からなる．以下では，演算処理のうち並列に行われる部分に関してのみ述べる．

各行は，順に演算の種類，二項演算のオペランド 2 つ，演算結果の格納先，キャリ情報 2 つ（参照キャリ，定義キャリ）を表している．

演算の種類には PAR_PLUS，PAR_AND，PAR_OR，PAR_ASSIGN などがあり，それぞれビットシリアルの加算，論理積演算，論理和演算，単純代入を表す．

#0，#1 はそれぞれ 0，1 の定数，“” で囲まれた記号は変数名を表し，$+i$ を付けることで LSB（Least Significant Bit）から（$i+1$）ビット目を示す．

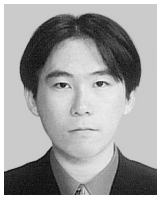
%pi はビットシリアル時に生成される 1 ビットのテンポラリメモリで，ci はキャリレジスタに格納されている 1 ビット値である．番号が同じものは，同じデータを表す．

変数の後ろの括弧の中の数字は変数番号で，名前が同じ変数が 2 つ以上存在する場合に区別できるようにしている．テンポラリメモリ，キャリ値の後ろの括弧の中の数字は参照回数であり，最適化に利用する．~ 記号は補数反転を表す．

(平成 19 年 1 月 22 日受付)
(平成 19 年 4 月 24 日採録)

小室　孝

平成 8 年東京大学工学部計数工学科卒業．平成 13 年同大学大学院博士課程修了．科学技術振興事業団研究員，東京大学助手を経て，現在，東京大学大学院情報理工学系研究科システム情報学専攻講師．ビジョンチップ，超並列画像プロセッサに関する研究に従事．博士（工学）．



鏡　慎吾

平成 10 年東京大学工学部計数工学科卒業．平成 15 年同大学大学院博士課程修了．科学技術振興事業団研究員，東京大学助手，東北大学講師を経て，現在，東北大学大学院情報科学研究科システム情報科学専攻准教授．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（正会員）

昭和 52 年東京大学工学部計数工学科卒業．昭和 54 年同大学大学院修士課程修了．同年通産省工業技術院製品科学研究所に入所．平成元年東京大学工学部計数工学科助教授．現在，同大学大学院情報理工学系研究科創造情報学専攻教授．超並列・超高速ビジョン，センサフュージョン，メタパーセプション等に関する研究に従事．工学博士．

片山　善夫

昭和 55 年広島大学工学部電子工学科卒業．同年（株）パナファコム（現，株式会社 PFU）入社．コンパイラ，多言語処理，PC クラスタの研究・開発に従事．