コンピュータビジョンと
イメージメディア 106－26
（1997．7．25）

一次元プロセッサアレイに基づくリアルタイム画像処理システムの開発環境

許　昭倫[†]　佐藤　完[††]
岡崎信一郎[†]　藤田善弘[†]

実世界を対象とする画像処理のアルゴリズム開発は、多量のシーンを対象としたアルゴリズム検証に加え、多数のパラメータチューニングを行う必要がある場合が多い。これには十分な計算パワーと、効率のよいプログラミング環境を兼ね備えた画像処理マシンが不可欠である。本稿では、これらの要件を満たすように開発された、ピーク性能10GOPSのPCIバス用1ボードLPA（一次元SIMD型プロセッサアレイ）をターゲットマシンとする、1）ソースコードの行単位の実行解析および試行錯誤的なパラメタチューニングに適したGUIを有する高級言語ベースの開発環境、および2）画素更新波の考え方に基づくLPA向け並列アルゴリズムの設計手法、について述べる。

A Programming Environment for Realtime Image Processing based on Linear Processor Array Architectures

Sholin Kyo,[†] Kan Sato,[††] Shin-ichiro Okazaki[†]
and Yoshihiro Fujita[†]

The development of real world image processing and pattern recognition algorithms require not only to perform experiments on a large amount of real world scenes, but also various parameter tuning. These requirements lead to the need of a really high performance computer system equipped with an efficient programming environment. In this paper, we describe a high level programming environment possessing a GUI suitable for parameter tuning and source code performance analysis, for a 10 GOPS one board PCI bus LPA (Linear Processor Array) machine. Together, we also describe a new way of designing parallel LPA algorithms based on pixel updating waves.

1. はじめに

画像処理では、膨大な量の（画素）データが処理対象である。特に実世界を対象とした場合、そのデータ量はさらに増大する。それに加え、画像処理アルゴリズムの開発は、多数のパラメータチューニングや、アルゴリズム自体の微調整を多く必要とする。このように膨大なデータ量に対し、多数回の試行を繰り返すことは多大な計算パワーを要することから、これまで多くの画像処理アルゴリズムは、その設計・開発の段階において、多数の画像データを対象にアルゴリズムの動作検証を満足に行えない場合が多かった。

上記の問題点を解決するためには、優れた開発環境を備えた高速画像処理エンジンの存在が不可欠である。既存の高速画像処理エンジンの内、高速性に加え、実世界での使用に求められるコンパクト性を共に備える画像処理アーキテクチャの一つが、一次元SIMD型プロセッサアレイ（Linear Processor Array: LPA）[1)2)]である。しかし残念ながら、既存のLPAに対しこれまで実用レベルのプログラム開発環境が提供された例はなかった。そうした中で、我々は高集積型LPAであるIMAP-VISION[5)6)]をターゲットマシンとする、真に実用レベルのプログラム開発環境の構築を目指し、これまで高級言語の設計[7)]、最適化に重点を置いたそのコンパイラの開発[8)]、そしてソースコードの行単位の実行解析および試行錯誤的なパラメタチューニングに適したGUIインタフェースを有する高級言語デバッガを開発してきた。本稿では、このIMAP-VISION上で実現された、高級言語ベースのプログラミング環境について述べると同時に、実際に既存の画像処理アルゴリズムをLPA向けに並列化する手法を示す。

2章では、IMAP-VISIONについて簡単に紹介した後、LPA用にC言語を拡張した高級言語1DC(One Dimensional C)について、その言語仕様、IMAP-VISION向け1DC言語処理系、そして1DCプログラムの起動およびビデオ画像処理時の画像入出力方法について述べる。3章では、1DCのプログラミング環境、

† 日本電気株式会社
NEC Corportaion
†† NEC情報システムズ
NEC Informatec Systems, Ltd.

特に1DCソースデバッガについて、そのビデオ画像処理時でのデバグやチューニングに有効な、大域変数へのスライダーバー割り付け機能、そして処理時間の計測機能等について述べる。4章では画像処理のカテゴリ毎に対応した、画素更新波の考え方に基づくLPA向け並列化手法と、その1DCテンプレートプログラムを与えると同時に、既存画像処理アルゴリズムに対する幾つかの並列化例も示す。最後に5章でまとめを与える。

2. IMAP-VISIONとLPA用C言語1DC

2.1 1ボードリアルタイム画像処理システムIMAP-VISION

ここではLPAを次のように定義する。すなわち全PEを制御する一つのコントローラを有し、PE数は画像の列数に等しく、隣り合う各PEは同じように隣合う画像の各列の処理を担当する(PE数をPENOとすると、処理対象の画像サイズは$NROW \times PENO$)。コントローラからは、全PEへのデータブロードキャスト、全PEのステータス値の集計、各PEのローカルメモリへの逐次アクセス等が可能であり、PE間では、少なくとも各PEと自分の左右のPEとの間、そして最右端PEと最左端PEとの間に、データ転送経路が存在する(図1)。

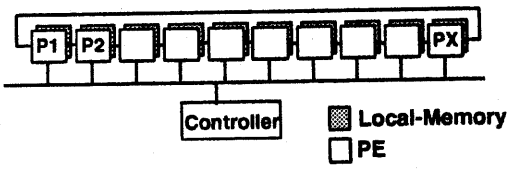

図1 LPAの基本構成

IMAP[3]は、こうしたLPAの基本アーキテクチャのもとで、PEアレイとメモリ間のデータ転送のさらなる高速化を実現するために、メモリとPEアレイをワンチップに集積したLPAアーキテクチャである。IMAPに基づく画像処理システムは、これまでVMEバス版のRVS-1(256PE)[7],RVS-2(512PE)[4]、そして最近ではVME版[5]と共にPCI版[6]のIMAP-VISION(256PE)が開発された。IMAP-VISIONの大きな特徴の一つが、256×256の8bit画像をチップ内に最大4面、それとチップ内と高速に結合された外部メモリ(SDRAM)に最大256面保持できる点にある。この外部メモリの存在により、例えば最大7秒程度のビデオ画像を一時メモリ上に保持できるので、大きなワーク領域を必要とする画像処理にも対応できるようになる。図2にPCI版IMAP-VISIONボード写真を示す。左側に見える8チップが各32PE+32KB(SRAM)を集積したIMAP-VISIONチップ、右側の1チップが全体を制御するコントローラ(16bitのRISCプロセッサ)である。表1にIMAP-VISIONの基本画像処理性能を示す。

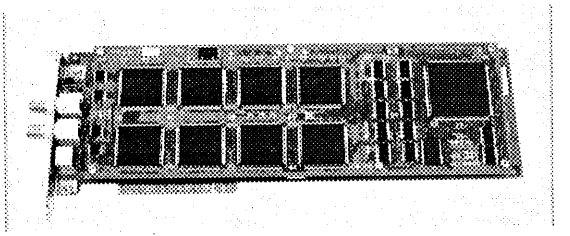

図2 PCI版IMAP-VISIONボード

表1 基本画像処理の性能

処理	処理時間(ms)
画像2値化	0.036
3×3 2値論理マスク	0.045
3×3 ラプラシアンフィルタ	0.186
3×3 コンボリューション	0.65
3×3 メディアンフィルタ	1.07
濃度ヒストグラム	0.082
90度回転	0.51
任意角回転	0.8-2.5
拡大縮小	1.68
細線化(1回周囲を削る時間)	0.36
2値ラベリング	最大19.2
ハフ変換(1340点)	4.3
5x5オプティカルフロー	5.6

2.2 1DCの言語仕様

1DC[7][8]は、上記の基本的なLPAアーキテクチャのもとで、一次元SIMD的な並列動作を明快に記述できるよう、C言語に必要最小限の仕様拡張を行ったLPA用データ並列言語である。C言語からみたその拡張は、大きく以下の3点に分けられる。

- 各PEにそれぞれ実体が存在する変数を指定するための拡張変数宣言子sep
- sepタイプの変数を操作するための拡張演算子:>, :<, :||, :&&, :[, :], :(, :) \item 全PEを複数のグループに分けるための拡張制御文\verbmif...melse, mwhile, mfor, mdo+

拡張変数宣言子sep付きで宣言された変数(sep変数)は各PE上のメモリ領域、それ以外の変数(スカラー変数)は全体を制御するコントローラ上のメモリ領域を割り当てられる。スカラー変数と比べるとsep変数は、その各要素が分散して各PE上に存在する一次元スカラー配列に相当する。

PEを複数のグループに分けるための拡張制御文は、sepタイプの値を返す条件式を評価し、その結果が0のグループとそうでないグループにPE群を2分する。拡張制御文をネストして記述すればさらに細かい分割も可能である。各拡張制御文は、対応するCの制御文の先頭にmを付けたものをその記法としている。同様に、Cの演算子の一部で先頭に:が付くものが拡張演算子である。例えばE_{sep}、E_{sca}をそれぞれsepタイプ、スカラータイプの式とすると、E_{sep}:[E_{sca} :] はE_{sep}

の第 E_{sca} 番目の PE 上でのスカラー値を取り出すことを意味する。:>E_{sep} と :<E_{sep} はそれぞれ左と右に位置する PE 上での E_{sep} のスカラー値への参照を意味する。最後に:&&E_{sep} と :||E_{sep} は、それぞれ E_{sep} の全 PE に渡るスカラー値の論理積と論理和を意味する。図 3に 1DC の拡張演算子と拡張構文の動作を示す。

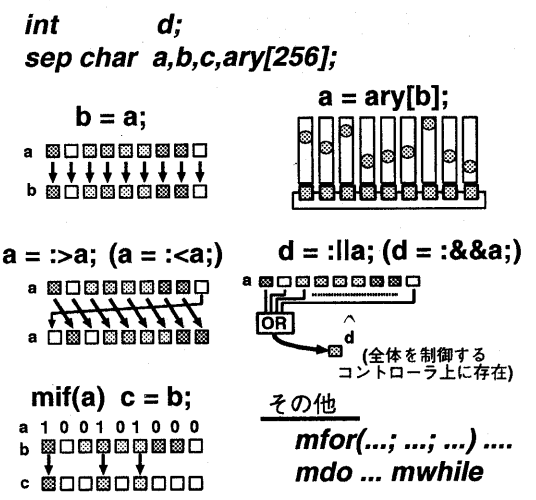

図 3　1DC の拡張演算子 / 拡張構文の動作

2.3　言語処理系

1DC コンパイラはこれまで、IMAP-VISION をターゲットマシンとして開発を行ってきた。その言語処理系を図 4に示す。ccimap は全体で 1dc1(構文解析・中間コード生成)、1dc2(中間コード最適化)、1dc3(PE 群分割用コードの生成および最適化)、1dc4(IMAP-VISION アセンブリコード生成)、そして 1dc5(アセンブリコード最適化)、の 5 つの処理パスからなる。C プログラムに変換するパス (1dcc) も存在し、それにより 1DC プログラムをホスト PC あるいは WS 上で実行することができる。その他リンカ ldimap、GUI ベースの 1DC ソースデバッガ sdbimap、アセンブリレベルデバッガ xdbimap、そして IMAP-VISION を実装しないホスト上でも 1DC プログラムの動作シミュレーションが可能なシミュレータ simimap 等が存在する。1DC ソースデバッガ sdbimap については次章で詳しく述べる。

1DC コンパイラは、1DC の C に対する簡潔な拡張言語仕様と、IMAP-VISION の PE アレイやコントローラが有する RISC 命令セットのもとで、従来からの汎用プロセッサ向け最適化手法が十分に適用可能であった。1DC コンパイラに対する性能評価結果として、表 2に 4 つの簡単な画像処理プログラムに対する (a) 最適なアセンブラプログラム、(b) コンパイラ生成コード、のそれぞれの実行ステップ数の比較結果を示す。効率のよいコード生成が実現されていることがわかる。

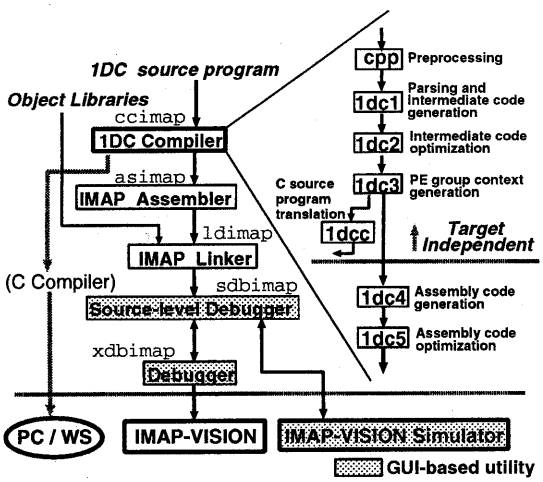

図 4　IMAP-VISION 向け 1DC 言語処理系

表 2　1DC コンパイラの性能評価

ソース名	(a)	(b)	(b)/(a)	(b) の実行時間
二値化	1547	2402	1.55	0.061ms
3x3 平滑化	5600	6430	1.15	0.161ms
ヒストグラム	4039	4872	1.21	0.122ms
90 度回転	20696	23326	1.13	0.583ms

2.4　プログラムの起動と画像入出力の方式

IMAP-VISION システムでは、画像の入出力やプログラムの起動をビデオ回路から送られてくるビデオ信号に同期した割込みプログラムによって行っている。

1DC では通常の C 言語と同様、処理は main という名前の関数からスタートする。ビデオ画像処理と静止画像処理のどちらを行うかの選択は、特定のシステム変数値によって指定する。静止画像処理の場合、main 関数の終わりまで実行したら停止する。ビデオ画像処理の場合では、16.5ms 毎あるいは 33ms 毎 main 関数が起動される。これは main 関数の処理が起動間隔よりも速く終了した場合でも同様であり、逆に前回起動された main 関数の実行がまだ終了していない場合、main 関数は新たに起動されない (図 5参照)。

IMAP-VISION では、画像入出力は、1 行の画像の入力終了に対応するビデオ信号が発生する度に所定の割り込みプログラムが起動される。その割り込みプログラムは、既に IMAP-VISION の 2 本の画像入出力シフトレジスタ内に溜った 1 ～ 2 行の画像を、指定のメモリ領域 (A とする) に格納すると同時に、同じく指定のメモリ領域 (C とする) から 1 ～ 2 行の画像を読み、同シフトレジスタに格納する。それを 33ms 間に 240 回行うことで、main 関数と独立に画像入出力を実現している。

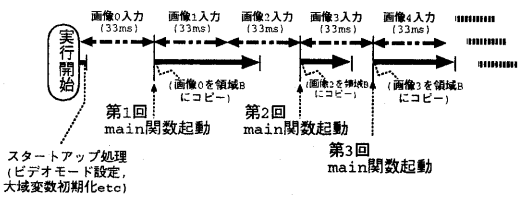

図5 動画像処理時の1DCのmain関数が起動されるタイミング

なおmain関数の第1回目の起動は、1枚目の画像の入力が終わった時点(スタートから33ms後)になされる。通常ユーザはmain関数の先頭において、その時点領域Aにある画像を別の領域(Bとする)にコピーし、main関数内では以降領域B内の画像を対象に処理を行う(図5参照)。なおアセンブリ関数を用いれば、1枚目の画像の入力終了を待たずに処理を始める(例えば各行の画像の入力終了と共に処理を開始する)ことも可能である。

3. ビデオ画像処理プログラムのデバッグ環境

IMAP-VISIONでは、1DCおよびその最適化コンパイラccimap、そしてGUI付きのソースデバッガsdbimapを中心とするプログラミング環境を開発した。

sdbimapはそのバックグラウンドタスクとして、xdbimapあるいはsimimapを起動する。sdbimapの画面はcommand-windowとsource-windowの他、ユーザが随時生成するwindowとしては、変数値を常時表示するdisplay-window、コマンド履歴を表示するhistory-window、そして動的変数値調整ツールに対応する各window等からなる。図6にsdbimapで各種windowを開いた時の様子を示す。

sdbimapの特徴は、1)行単位ブレイクポイント設定、2)任意変数名へのマウスクリックによる変数値表示とその常時表示、3)関数単位のディスアセンブル表示、4)メモリ内容を、画像形式(任意倍率)あるいはキャラクタ形式で表示、等のソースデバッガにとしての基本機能の他、5)変数値の動的調整ツール(1次元スライダー、2次元スクロールマップ、択一型ボタン)、6)行/関数単位の処理時間の自動計測、のようなビデオ画像処理用の機能を有する点にある。

実世界を対象とする画像処理アルゴリズムの開発では、試行的なパラメタチューニングや、多数のシーンを相手としたアルゴリズム検証は必須である。sdbimapではそうした繰り返し型の試行に適したフレーム毎に変数値を動的に調整できるGUIとして、択一ボタン、1次元スライダー、2次元スクロールマップ、を用意した(図7参照)。

各ツールと変数名との対応付けは、ソースwindow内に現われた変数名をマウスで選択した上、いずれかのツールに対応するボタンをクリックすることにより行う。それによりビデオ画像処理実行時では、ユーザがス

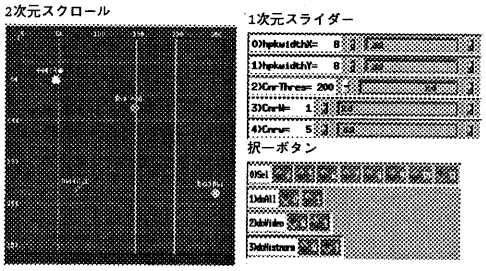

図7 sdbimapで用意された3種類の変数値調整ツール

ライダー等をドラッグする度に、対応する変数に割り付けられているメモリ領域へ、対応する値がその都度書き込まれるようになる。なお、2次元スクロールマップの場合は、1組の変数を指定すると、対応する記号がマップ内に現われる。そしてユーザが記号をドラッグすると、記号のx,y座標値は対応する1組の変数のメモリ領域に書き込まれる。また択一ボタンを使用することで、対応する変数に書き込む値を、限られた数個のものに制限できる。ユーザは、main関数の先頭でこれらの変数を参照するようにプログラムを作成することで、main関数が毎回起動される度に、その時点でのスライダー等の位置に対応する値が、当該起動に対応するフレームの画像の処理パラメータとして使用することができる。

sdbimapのこうした動的パラメータ調整機能とIMAP-VISIONの高い処理性能の組み合わせにより、パラメタの変更に対するアルゴリズムの挙動はビデオ画像上でつぶさに観測できるようになる。すなわち一瞬のうちに多数の画像に対し適用結果が得られる。それらの適用結果を、IMAP-VISIONが有する大容量の外部画像メモリに格納おくことで、ビデオ画像を対象としたアルゴリズム検証を容易に実現できる。図8にsdbimapを利用したビデオ画像処理プログラム開発の様子を示す。

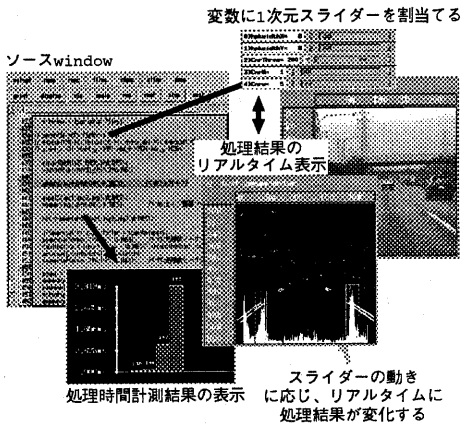

図8 sdbimapを利用したビデオ画像処理プログラムの開発

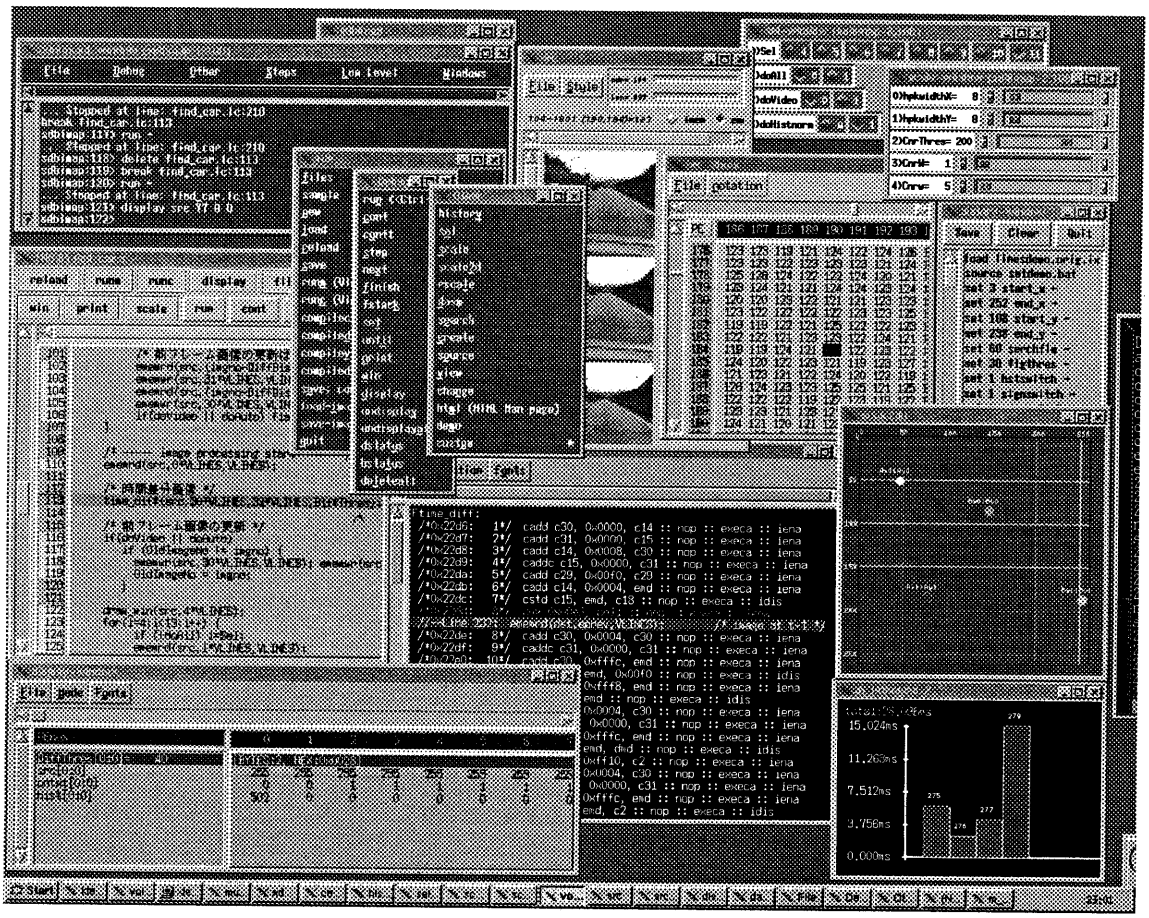

図6 sdbimap が生成する各種 window

動画像処理を実現する際のもう一つのポイントが、処理の高速化のためのアルゴリズムのチューニングである。sdbimap では、関数名をマウスでクリックした後所定のボタンをクリックすれば、当該関数が消費した動的な処理時間の計測値が得られる。同様に任意箇所のソース行間に挟まれたコードの処理時間も行番号を指定することで容易に得られる。この処理時間解析機能は、処理全体のボトルネックとなる箇所の検出に有効であり、ユーザはプログラムのそうした箇所を再チェックすることで、処理全体のパフォーマンスを効率よく高めることができる。

4. LPA 向き並列処理の記述

LPA の処理性能を最大限に引き出すためには、一次元的に接続された各 PE を、SIMD 動作の枠組み中で積極的に利用するような並列アルゴリズムが必要である。以下では、画像処理をその処理の形式に応じて幾つかのカテゴリ[10] に分類した上、各カテゴリの処理に適したタイプの画素更新波 (Pixel Updating Wave: PUW) を生成することにより並列化を行う方式について述べる。

PUW に基づく LPA 向き並列化方式とは、画像の各列を格納した PE アレイのローカルメモリ集合からなる画像メモリという 2 次元平面上に、PUW の 1 次元のウェーブフロント (WF) を必要に応じ自在に進行させ、そして WF 上に位置する各画素を同時に処理対象とすることで、処理の並列性を得ようとするものである。LPA の各 PE が有するアドレッシングの自律性 (各 PE が異なるメモリアドレスをアクセスできる性質) をフルに利用した並列化手法といえる。図 9に画像処理の各カテゴリと、それらを並列化する際に用いられる、水平型、折返し型、傾斜型、そして自律型の 4 種類 PUW のとの対応付けを示す。

以下では、各種の PUW、およびそれらを記述するための 1DC テンプレートプログラムを示す。また、幾つかの既存画像処理をそれらによって並列化した例と、

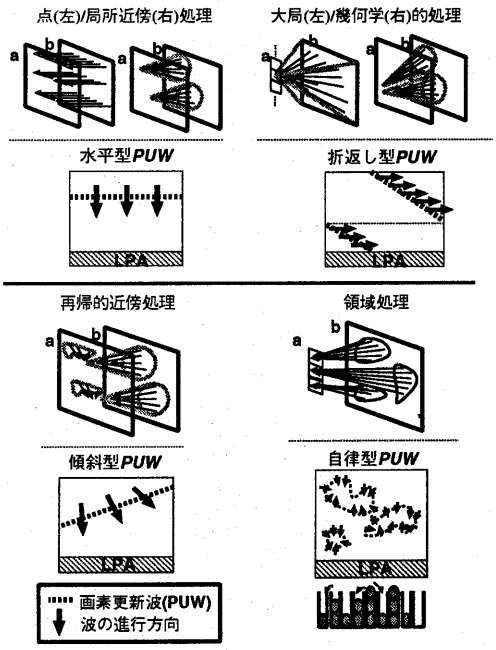

図9 画像処理の各カテゴリとそれらをLPA上で並列化する各種PUWとの対応付け

IMAP-VISION上でのそのパフォーマンスも合わせて示す。

4.1 水平型PUWによる並列化

PEアレイと平行なWFを有するPUW(水平型PUW)は、あらゆるPUWの基本形である。水平型PUWのWFで画像を一掃する(全画素を一回ずつ訪れる)のにかかる反復数は常にNROW(画像の行数)に等しく、LPAにとってはもっとも効率のよい並列処理方法の一つである。

水平型PUWを生成する1DCプログラムのテンプレートを以下に示す。WFの位置は全PEで同一であるため、それを指定する変数はスカラー変数cを用い、WFの進行はfor文でcを順次増分させることにより記述し、そしてWF位置の画素に対する処理は、for文の本体によって指定する。

```
#define uchar unsigned char

void Row-wise-PUW(src,lines)
     sep uchar src[];
     int lines;
{
 int c;
 for(c=0;c<lines;c++) {/* 行数回だけWFを進行 */
    update(src+c);      /* WF位置の情報を更新 */
 }
}
```

4.2 折返し型PUWによる並列化

折返し型PUWでは、最右端と最左端のPEが互いに隣接関係にあることを利用したWFを生成する。そのWFで画像を一掃するのにかかる反復数は常にPENO(PE数)に等しく、水平型PUW同様、LPAにとってもっとも効率のよい並列処理方法の一つである。

折返し型PUWの1DC記述では、WFを進める方向により、1DCの拡張演算子:<(右から左への場合)、:>(同左から右への場合)のいずれかを使用する。またWF位置は各PEで異なるため、その指定にはsepタイプの変数を用いる。右から左へ進むWFを生成する1DCプログラムのテンプレートを以下に示す。但しPENUMはPE番号(最左端から順に0,···,255)を表すsepタイプの定数、PENOはPE数(256)を表すスカラー定数とする。

```
void wraping-PUW(src)
    sep uchar src[];
{
 int i;
 sep uchar s=PENUM;  /* WF位置初期化 */

 for(i=0;i<PENO;i++){/* PE数回WFを進行 */
    update(src+s);/* WF位置の情報を更新 */
    s = :<s;       /* WF位置更新 */
 }
}
```

折返し型PUWは、特に濃度ヒストグラムあるいは、領域毎の面積/重心/周囲長等のような統計量計算、あるいは画像回転等のような幾何学的処理の並列化に有効である。例えば画像内のラベル付けされた各領域の面積を求めるには、水平型PUW、折返し型PUWを一回ずつ発行することで、ラベルNの領域の面積がN番目のPE上に求まる。図10にその処理のシーケンスを示す。また、同様の処理で濃度ヒストグラムを求めた場合の処理例(1DC使用:処理時間0.1ms)を図??に示す。

4.3 傾斜型PUWによる並列化

画像処理アルゴリズムにおいて、各画素が自分の右上、上、左上、左の画素が処理済みになって始めて処理対象となり得るような場合(ラスタスキャン順に画像を処理した場合はそのようになる)、隣接画素同士で処理対象となるタイミングにディレイが生じる。そこで、このディレイの方向に応じて上方または下方、またディレイの度合に応じた角度だけ傾斜したWFを発生する必要がある。

傾斜型PUWの1DC記述では、x方向(PEが並ぶ方向)とy方向(画像の列方向)の両方向へのWFの進み具合を制御するための1組のsep変数(sxとsy)を用いる。sxは反復毎に(左から右へ)一画素ずつ値を伝搬させ、syは各画素が制約を受ける近傍の大きさをMとすると、M-1反復おきに増分させる。そしてWFは、sxの到達範囲内のPE上のsyによって表現する。例えば、各画素が制約を受ける近傍が自分の右上、上、左上、左の画素である場合のMは2となる。なお、このWFで画像を一掃するのにかかる反復数は常にM×(画像行数-1)+PENOに等しい。以下にこの傾斜型PUWを生成する1DCプログラムのテンプレートを示す。

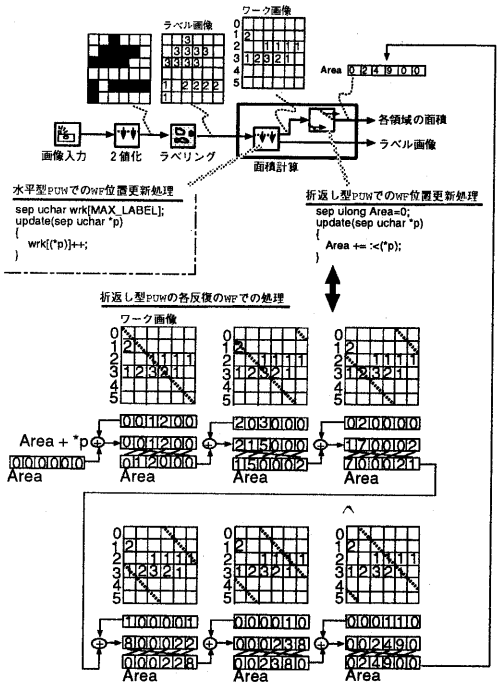

図10　折返し型 PUW に基づく並列化例

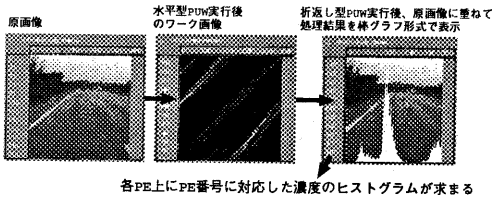

図11　水平型 PUW →折返し型 PUW による濃度ヒストグラム計算の処理例

```
void slant-PUW(src,lines)
    sep uchar src[];
    int lines;
{
 int i,cnt=0;
 sep uchar sx=0,sy=0;

 for(sy=0,i=0;i<M*(lines-1)+PEN0;i++){
    sx:[0:]=1;
    mif(sx & (cnt%M==0)){  /* sx到着チェック */
       update(src+sy); /* WF位置の情報を更新 */
       sy++;           /* WF位置更新 */
    }
    cnt++;
    sx = :>sx;          /* sxの伝搬 */
 }
}
```

例えば、画像の左上から右下に一回、右下から左上に一回の、計2パスのスキャンにより2値画像の距離変換を実現する既存の処理[9]の、LPA向け並列化は、傾斜型 PUW で2回画像を一掃することにより実現される。テスト画像に対する処理例を図12に示す(1DC 使用: 処理時間 8ms)。

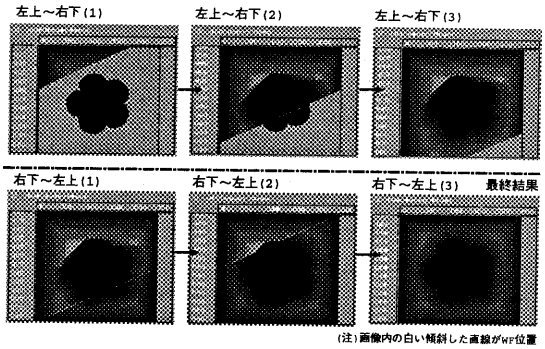

図12　2回の傾斜型 PUW 発行による距離変換処理の実現例

4.4　自律型 PUW による並列化

画像処理アルゴリズムにおいて、例えば輪郭追跡等のように、処理の進行状態が画像内容に左右される場合では、PUW を自律的に画像内で進行させることで、処理の並列化は自然な形で達成される。自律型 PUW では、LPA の各 PE(のローカルメモリ) 上にソフトウェアスタックを設ける。まず初期 WF 位置 (あるいはタネ画素位置) 位置を検出し (seed detection)、それを当該画素の処理を担当する PE のスタックにプッシュしておく。次に、空でないスタックがあるかどうかを調べ (stack check)、あればそこからポップ (pop) された画素位置の集合をカレント WF とした上、WF 位置の画素の情報更新 (update) を行うと同時に、WF の周りで、次回以降 WF となることが可能な画素位置があるかどうかを調べ、あればその位置情報を、PE のスタックにプッシュ (push) する。これを全 PE のスタックが空になるまで繰り返す (図13)。

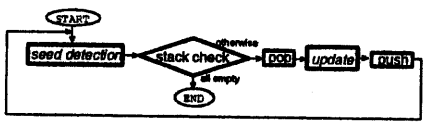

図13　自律型 PUW による処理の流れ

自律型 PUW を生成する 1DC プログラムのテンプレートを以下に示す。まず、一度水平型 PUW を用いてタネ画素を検出しスタックにプッシュしておく。次に、mwhile 文等により、図13のループを実現する。反復毎の WF 位置は、反復の度にスタックからポップされる画素位置で構成される。

```
void autonomous-PUW(src,stack,lines)
    sep uchar src[],stack[];
    int lines;
```

```
{
 int i;
 sep uchar sp=0,r,x;

 for(i=0;i<lines;i++) /* 水平型PUW: タネ画素検出 */
    mif(isSeed(src+i))
       stack[sp++]=i;      /* スタックにプッシュ */

 mwhile(sp) {           /* 自律型PUWのメインループ */
    x = stack[--sp];        /* ポップ:WF位置更新 */
    r = update(src+x);      /* WF位置の情報を更新 */
    prop(src+x,r); /* WF候補位置を検出→プッシュ */
 }
}
```

自律型PUWの利用によりLPA上で効率的に並列化可能な処理には、ラベリング、細線化、輪郭/線分追跡、領域拡張(region growing)、そして動的輪郭(snakes etc)等、多くのものが存在する。図14に、一個のタネ画素から、8近傍に位置する画素の濃度が自分のそれとの違いが所定しきい値(この場合5に設定)以内ならば自分のラベルをそこに伝搬する、いわゆる単純領域拡張処理の場合の、処理途中における各PEでのスタック長の増減の様子を示す(1DC使用: 処理時間4ms)。

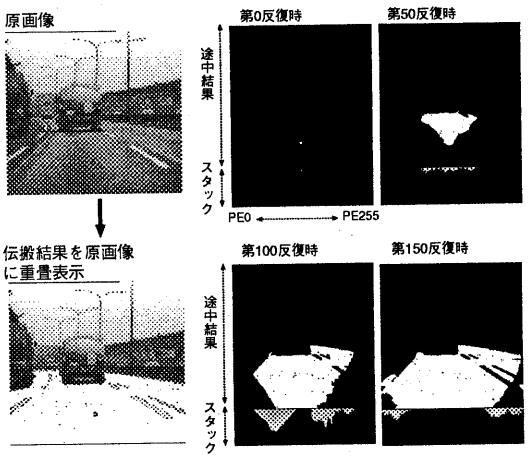

図14 自律型PUWによる単純領域拡張処理の並列化例

5. おわりに

実世界画像を対象とする動画像処理のアルゴリズム開発では、多数の画像データを用いたアルゴリズム検証に加え、現状では依然として多数のパラメータによるチューニングを必要とする場合が多い。これには十分な計算パワーと、効率のよいプログラミング環境を兼ね備えた画像処理マシンが不可欠である。本稿では、そうした計算パワーを持つマシンの一つであるIMAP-VISION向けに開発された高級言語に基づくGUIベースのプログラミング環境、そして既存の画像処理アルゴリズムをLPA向けに並列化するためのPUW(画素更新波)に基づく並列化手法について述べた。

高速でかつプログラム開発効率に優れた画像処理エンジンは、より高度な画像処理方式を産み出すツールとして不可欠である。しかし、これまでこの二つの要件を兼ね備えたものは筆者らの知る限り存在しなかった。高級言語の最適化コンパイラやビデオ画像処理に向くGUIを有するソースデバッガを備えるIMAP-VISIONは、そうした条件を満たす数少ない画像処理システムであり、本システムが今後、様々なリアルタイムビデオ画像処理応用の研究開発に役立つと考えられる。

謝辞 本研究を進めるにあたり、御指導頂いた天満センター長に感謝いたします。

参考文献

1) T.J.Fountain: The CLIP7A Image Processor,IEEE Trans. on PAMI, Vol.10, No.3. pp.310-319(1988).
2) Chin et al: The Princeton Engine: A Real-Time Video System Simulator, IEEE Trans. on Consumer Electronics, Vol.34, No.2, pp.285-297 (1988).
3) 藤田善弘, 山下信行, 木村亨, 中村和之, 岡崎信一郎: メモリ集積型SIMDプロセッサIMAP, 信学論D-I, Vol.J78-D-I, No.2, pp.82–90, Feb., 1995.
4) S.Okazaki, Y.Fujita, N.Yamashita: A Compact Real-Time Vision System using Integrated Memory Array Processor Architecture, IEEE Transaction on Circuits and Systems for Video Technology, pp.446–452, Oct., 1995.
5) Y.Fujita, N.Yamashita, S.Okazaki: IMAP-VISION: An SIMD Processor with High-Speed On-Chip Memory and Large Capacity Exrernal Memory, Machine Vision and Applications (MVA'96), Nov., 1996, Tokyo.
6) 岡崎信一郎, 藤田善弘, 許昭倫, 山下信行: 1ボード超高速動画像処理システムIMAP-VISION, 第3回画像センシングシンポジウム講演論文集, pp.201-206,1997,Tokyo.
7) 許昭倫, 岡崎信一郎, 藤田善弘, 山下信行: メモリ型画像処理プロセッサとその応用, 電子情報通信学会コンピュータシステム研究会報告, CPSY-93-50, (g1) pp.39–46, Jan., 1994, 東京.
8) S.Kyo, K.Sato: Efficient Implementation of Image Processing Algorithms on Linear Processor Arrays using the Data Parallel Language 1DC, Machine Vision and Applications (MVA'96), Nov., 1996, Tokyo.
9) G. Borgefors: *Distance Transformations in Digital Images*, CVGIP,Vol.34, pp.344–371, 1988.
10) P. P. Jonker: *Architectures for Multidimensional Low- and Intermidiate Level Image Processing*, Proc. of IAPR Workshop on Machine Vision Applications (MVA), pp.307–316, 1990.