計算機アーキテクチャ 119－17
（1996. 8. 28）

一次元プロセッサアレイ用データ並列言語1DCによる画像処理アルゴリズムの記述とそのコンパイラ

許　昭倫[†]　佐藤　完[††]

一次元SIMD型プロセッサアレイ(LPA)は、その低レベル画像処理にマッチした構成から、多くの画像処理システムのアーキテクチャに採用されてきているが、プログラム開発にはこれまで実用的な高級言語環境が提供されていない。本論文では、まずLPA向きに設計されたデータ並列言語1DC (One Dimensional C)、および高集積型LPAであるIMAP向けに実際に開発された、1DCを中心としたプログラム開発環境について述べる。次に、画像処理アルゴリズムの各処理形式に対し、行単位、列単位、行方向シストリック、斜方向シストリック、そして仮想スタックの5つのLPA向き並列化方式を指針とした、1DCによるLPA向き並列化記述手法を提案する。

Efficient Implementation of Image Processing Algorithms on Linear Processor Arrays using the Data Parallel Language 1DC and its Compiler

Sholin Kyo[†] and Kan Sato[††]

SIMD linear processor arrays (LPAs) have received a great deal of interest as a suitable parallel architecture for low-level image processing. However, few of them possess a high level programming environment support. In this paper, we first presented a design of a data parallel language for the description of linear array based image processing algorithms, and its programming environment including an optimizing compiler for IMAP, a highly integrated LPA. Then we proposed parallel implementation of algorithms on LPAs, for each of the low-level image operation categories, based on line-wise, row-wise, line-systolic, slant-systolic, and pesudo-stack methodologies under the use of 1DC.

1. はじめに

一次元SIMD型プロセッサアレイ(Linear Processor Array: LPA)は、その低レベル画像処理にマッチした構成から、多くの画像処理システムのアーキテクチャに採用されてきている[1)2)3)4)]。しかし、それらは殆んどの場合アセンブリ言語の使用が前提となっており、これまで実用的な高級言語環境が提供された例はない。それに対し、我々は仮想的なLPAをターゲットとしたデータ並列言語1DC (One Dimensional C)を設計し、かつLPA型の画像処理システムの一つであるIMAP[5)6)]向けにそのコンパイラを開発してきた。

LPA向け画像処理アルゴリズムは、LPAが持つ一次元的な並列性を最大限に利用できるものでなければならない。そうしたアルゴリズムを開発するには、画像処理アルゴリズムの処理形式に応じた、LPA向き並列化への明確な指針をが必要である。これまで個別のアルゴリズム用にSIMD的並列化手法が提案された例[12)13)14)]はあるが、LPAを対象とした一般的な並列化への指針は提案されていない。本論文では1DCの言語仕様、IMAP向けに開発された1DCコンパイラについて述べた後、画像処理アルゴリズムをその処理形式から複数のカテゴリに分け、そのそれぞれにおける典型的な処理に対し、1DCによるLPA向き並列化記述を与える。そしてそれらから、行単位、列単位、行方向シストリック、斜方向シストリック、そして仮想スタックという、5つのLPA向き並列化方式を導く。

以下2章では、1DCの言語仕様について述べる。3章では、1DCコンパイラを中心としたIMAPのプログラム開発環境について述べる。4章では、画像処理アルゴリズムの処理形式からみた、各カテゴリでの典型的な処理に対する1DC記述を与えると共に、5つのLPA向き並列化方式を抽出する。最後に5章では、LPA向き並列画像処理アルゴリズム開発への指針についてまとめる。

† NEC 情報メディア研究所
Information Technology Research Laboratories, NEC Corportaion
†† NEC 情報システムズ
NEC Informatec Systems, Ltd.

2. 1DC の言語仕様

1DC[9)10)] は、LPA による一次元 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う形で設計された LPA 用データ並列言語である。1DC の C からみた拡張は大きく分けると以下の 3 点にある。

- PE アレイ内の各 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 `:>`, `:<`, `:||`, `:&&`, `:[`, `:]`, `:(`, `:)`, `:=`
- PE アレイ内の PE を複数のグループに分けるための拡張構文 `mif...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 上に、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に、その実体が存在する。sep 変数は、その各要素が各 PE 上に存在するような一次元の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**拡張構文**は、sep タイプの値を返す条件式を評価し、その結果が 0 の PE とそうでない PE に PE 群を 2 分割する。拡張構文をネストして記述すればさらに細かい分割が可能である。各拡張構文の記法は、その先頭の m を取り除いた場合の C の対応する構文のそれと同じである。

同じように演算子の場合も、表記で先頭に : を有する演算子が**拡張演算子**である。c0,···,cn を定数とすると、:(c0,···,cn :) とは LPA で最左端 PE からみて 0 番目から n 番目までの仮想 PE 上に、それぞれ c0,···,cn の値、第 n+1 番目から n 番目までの仮想 PE 上に 0 値、が存在するような *sep* タイプの定数である。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} 番目の仮想 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。

3. 1DC コンパイラ

1DC コンパイラはこれまで、高集積型 LPA である IMAP[6)] をターゲットマシンとして開発を行ってきた。

1DC による IMAP 向けプログラム開発環境を図 1に示す。コンパイラ全体は図 1に示すように、1dc1(構文解析・中間コード生成)、1dc2(中間コード最適化)、1dc3(PE 群分割用コードの最適化)、1dc4(IMAP アセンブリコード生成)、そして 1dc5(アセンブリレベル最適化)、の 5 つの処理パスからなる。1dc3 まではターゲットマシンに依存しない中間コードに対する処理であるため、ターゲットマシンを変えてもそのまま使用可能である。C プログラムに変換するパス (1dcc) も含まれており、1DC プログラムを PC やワークステーション上で実行することも可能である。開発ツール類の充実化も図っている。X ウィンドウベースのソースレベルデバッガは、ソースプログラムに対する行単位のブレイクポイント設定やトレース、変数値の表示、ヒストリ等の機能のみならず、動画像処理時にフレーム単位でプログラム内の変数値をインターラクティブに調整する機能も有し、快適なリアルタイム画像処理プログラム開発環境を提供する。

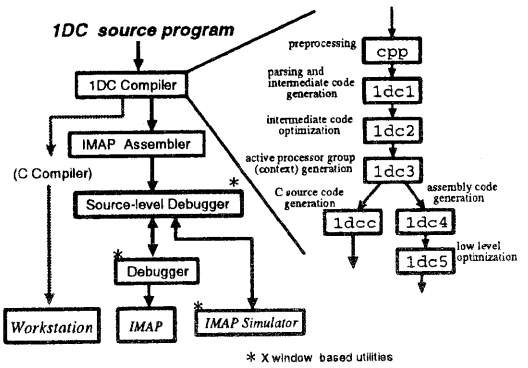


図 1 IMAP 向け 1DC プログラム開発環境

1DC コンパイラに対する性能評価例として、表 1に 4 つのシンプル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。1DC が持つ簡潔な拡張言語仕様と、IMAP の PE アレイ部の RISC ライクな命令セットにより、効率のよいコード生成が実現されていることがわかる。なお現時点では最適化処理 (1dc3,1dc5) の一部が未完成なため、表 1の評価には人手でループアンローリングを施した 1DC のソースコードを使用している。

ソース名	(a)	(b)	(b)/(a)
単純二値化処理	1547	2402	1.55
3x3 平滑化処理	5600	6430	1.15
ヒストグラム処理	4039	4872	1.21
90 度回転処理	20696	23326	1.13

表 1 プロトタイプ 1DC コンパイラの性能評価

4. 一次元 SIMD 的並列画像処理の記述

ここでは低～中レベルの画像処理 (画像を対象とした前処理・特徴抽出処理) をその処理形式に応じて下記 6 つのカテゴリ[8)] (図 2参照)、すなわち

(1) 点処理 (Point Operation: PO)

(2) 局所近傍処理 (Local Neighbourhood Opera-

tion: LNO)
(3) 大局的処理 (Global Operation: GO)
(4) 幾何学的処理 (Geometric Operation: GeO)
(5) 領域処理 (Region Operation: RO)
(6) 再帰的近傍処理 (Recursive Neighbourhood Operation: RNO)

に分けた上、1DC を用いて各カテゴリに属する典型的な処理に対する LPA 向き並列化記述の例を与える。

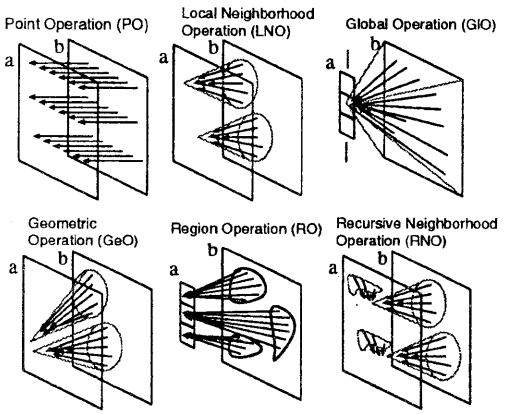


図2 処理形式に基づく画像処理アルゴリズムの分類

以下では仮想 LPA として、全体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、各 PE は画像の各列の処理を担当するものとする。即ち PE 数は PENO、処理対象の画像サイズは $NROW \times NCOL$、かつ NCOL は PENO に等しいとする。またコントローラ -PE 間の結合はコントローラから全 PE へのブロードキャスト、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、各 PE と自分の左右の PE との間、最右端 PE と最左端 PE との間以外に、データ転送経路はないものとする。

4.1 点処理 (PO) と局所近傍処理 (LNO)

PO と LNO は、共に処理画像の各画素を結果画像の各画素へ一対一に並列に変換可能な処理であり、その際 PO は他の画素に依存せずに変換を行うのに対し、LNO は自分を中心とした局所領域内の画素を参照し変換を行う。LPA 上では、両者ともに行方向の並列性を持つため、1DC 記述では行単位で処理を記述し、それを画像の行数回だけ繰り返せばよい。以降これを**行単位方式**と呼ぶ。例として画像反転 (PO) と、各画素に対し

```
 0 -1  0
-1  4 -1
 0 -1  0
```

の 3×3 フィルタをかける処理 (LNO) の 1DC 記述を示す。

```
void revimg(sep unsigned char *src,
            sep unsigned char *dst, int len)
/* src: ソース画像  dst: 結果画像  len: 処理行数 */
{
    while ( --len>=0 )
        *(dst++) = 0xff-*(src++);
}

void laplace(sep unsigned char *src,
             sep unsigned char *dst, int len)
{
 int i;
 sep int p0, p1, p2, p3, p4;

 for ( i=1 ; i<len-1 ; i++ ) {
                    p0 = src[i-1];
   p1 = :>src[i];   p2 = src[i]<<2; p3 = :<src[i];
                    p4 = src[i+1];
   p0 = p2-p0-p1-p3-p4;
   mif (p0>0) dst[i] = p0;
   melse dst[i] = -p0;
 }
}
```

4.2 大局的処理 (GlO)

GlO とは、全画素からデータを集計する統計的な処理を指す。LPA の場合、大局的処理は間接アドレッシング、即ち各 PE が同時に異なるメモリアドレス (画像の各列位置について異なる行位置) のデータへアクセスした上、それらを PE 間の転送経路を利用したシストリックな転送によってデータを集計すればよい。以降では、このような間接アドレッシングを**列単位方式**、シストリックなデータ転送を**行方向シストリック方式**と呼ぶとする。

LPA 上でのもっとも典型的な GlO が濃度ヒストグラム計算である。その実現法を紹介すると、まず各 PE が自らの担当する 1 列の画像データに対するヒストグラム結果をメモリ内の一つずつずれた位置 (図 3参照) に格納する。次に、それらをメモリから順番に取り出しては、隣接 PE に転送しながら累積加算していく。それにより、最終的に画像内の各画素値の累積数が、対応する PE 番号を持つ PE 上に求まる。これらの処理の 1DC 記述を以下に示す。また図 3は NCOL が 256 の場合に、画素値 0 の画素の数の計算がどのように最左端の PE に集計されていくかを表現したものである。なお→は右隣 PE へのデータ転送、↓は加算を表す。

```
sep unsigned int histogram(sep unsigned char src[],
                           sep unsigned char hst[])
{
   /* PENUM とは :( 1,2,...,255 :) の既定定数 */
   sep unsigned char *p, penum=PENUM;
   sep unsigned int result=0;
   int i;

   /* Y 方向のヒストグラム (列単位方式) */
   for(p = src, i=0; i<NROW; i++)
         hst[(*(p++) - penum)&255]++;
   /* X 方向での集計 (行方向シストリック方式) */
   for(p = hst, i=0; i<NCOL; i++)
         result = :<(*(p++) + result);
   return(result);
}
```

4.3 幾何学的処理 (GeO)

GeO とは、画像の各画素の位置を変える処理を指す。それは、多くの場合 Y 方向 (画像の列方向) および X 方

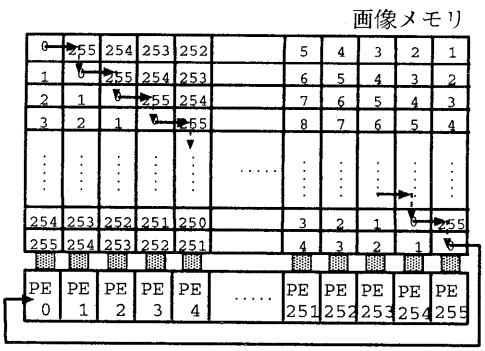


図3 ヒストグラム計算のアルゴリズム

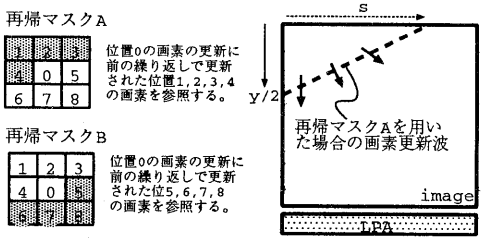
向(画像の行方向、LPAにとってはPE方向)への画素の並列シフトを組み合わせることにより実現できる。そこでLPA上ではY方向の画素の並列シフトは**列単位方式**、X方向の画素の並列シフトは**行方向シストリック方式**によって実現すればよい。画像をY方向、X方向、Y方向の順に適量シフトすることにより逆時計回りの90度回転を実現するアルゴリズム[12]を、LPA上に実現する場合の1DC記述を以下に示す。

```
void rotate90(sep unsigned char src[],
              sep unsigned char tbl[])
{
   int          i;

   /* 1回目のY方向シフト */
   for(i=0;i<NROW;i++)
       tbl[(i+255-PENUM+1)&255]= src[i];

   /* 1回目のX方向シフト */
   for(i=0;i<NROW;i++)
       tbl[i] = tbl[i] :< (PEN0-i);

   /* 2回目のY方向シフト */
   for(i=0;i<NROW;i++)
       src[i]= tbl[(i+PENUM+1)&255];
}
```

4.4 再帰的近傍処理(RNO)

RNOでは、中心画素の値を更新する際に、既に更新された近傍の画素値を参照するという、逐次性の強い処理であり、通常は画像の各画素をスタスキャン順に処理することにより実現される。ここでは、それをLPA上で効率よく実行するための方式として、**斜方向シストリック方式**を提案する。

RNOにおける画素同士の依存関係は、通常図4(a)のような再帰マスクの形で記述できる。斜方向シストリック方式とは、この再帰マスクが課す制約を満足するために、LPA上の各PEをシストリックに、すなわちあるタイムディレイのもとで次々と起動するように制御する方式である。その結果、タイムディレイの値に応じた角度を持った斜方向への画素更新波が得られる(図4(b)参照)。タイムディレイの大きさは再帰マスクのサイズに依存し、サイズが(2M+1)×(2M+1)の再帰マスクの場合、タイムディレイはMとなる。本方式を用いることでN×Nの画像に対し、MがNより充分に小さい場合、RNOはN個のPEを有するLPAのもとでO(N)の時間で処理される。



(a)再帰マスクの例　(b)斜方向画素更新波の生成

図4 再帰マスクの例とそれによる斜方向の画素更新波の生成

サイズが3×3の再帰マスクの場合の画像内の各画素の処理される順番を図5に示す。こうした画素の処理順番をとることにより、サイズ3×3の再帰マスクが課する制約が満たされることは、例えば図5の左側で、太線で囲まれた、繰返し5(円内に5と表記)に処理される画素を中心とする、3×3近傍内の各画素が処理される繰返し数が、全て5以下であることからもわかる。

例として、図4(a)の再帰マスクA,Bを用いて、画像をそれぞれ左上から右下(forward)へ、右下から左上(backward)へ全面スキャンして距離変換☆を行う処理[7]の、斜方向シストリック方式に基づく1DCの記述を以下に示す。関数**RScan**では、2つのsep変数s,yを用いて斜方向の画素更新波の形成を実現している。sは、起動開始シグナルをLPA上の端(forward時は最左端、backward時は最右端)のPEから反対側の端(forward時は最右端、backward時は最左端)のPEまで伝搬させるための制御変数である。sが既に到着したPEでは、繰り返し毎に、yを増分(forward時)あるいは減分(backward時)させ、また再帰マスクのサイズが3×3からタイムディレイは1であり、したがって起動開始シグナル到着済みのPEでは、yが偶数になる度に、画素に対する距離変換関数**dt**に第y行にある画素を更新させるように動作する。

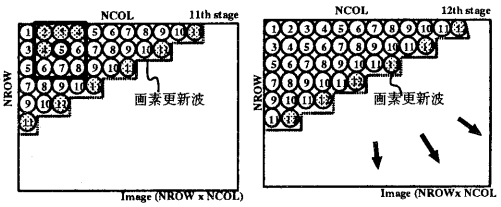


図5 サイズが3×3の再帰マスクの場合の各画素が処理される順番

☆ 2値画像の各前景画素位置に、それへのもっとも近い背景画素への距離を書き込む処理

```
#define D 3  /* 対角画素への距離値加算 */
#define S 2  /* 隣り画素への距離値加算 */
#define min(a,b)  (((a)>(b))? (b) : (a))

sep unsigned char dt(in,y,dir)
sep unsigned char *in,y;
char dir;
{
 sep unsigned char a,b,c,d,e;

 e=in[y];
 if (dir){
   a=:>in[:<y-1]; b=in[y-1]; c=:<in[:>y-1];
   d=:>in[:<y];   } else  {  d=:<in[:>y];
   c=:>in[:<y+1]; b=in[y+1]; a=:<in[:>y+1];
 }
 /* 最小値選択 */
 return min(min(min(e,a+D),min(b+S,c+D)),d+S);
}

void RScan(in, dir, xsz, ysz)
sep   unsigned char *in;
int   dir, xsz, ysz;
{
    int      i,start,inc_or_dec;
    sep int  y,y2,s=0;

    inc_or_dec = (dir) ? 1 : -1;
    start = (dir) ? 1 : xsz;
    s:[start:] = 1;
    y = start<<1;

    for(i=0; i< 2*(ysz-1)+xsz; i++) {
       mif (s) {
          y2 = (y>>1);
          mif (!(y&1) && y2<=ysz && y2>0)
               in[y2] = dt(in,y2,dir);
          y += inc_or_dec;
       }
       if (i < xsz-1) {
           s = (dir) ? :>s : :<s;
           s:[start:] = 1;
       }
    }
}

void distance_trans_scan(sep unsigned char *src)
{
 /* src に処理対象２値画像が格納されているとする */
 RScan(src,1,NROW-2,NROW-2);/* forward スキャン */
 RScan(src,0,NROW-2,NROW-2);/* backward スキャン */
}
```

4.5 領域処理 (RO)

RO とは、画像内の個々の領域に対し同一の処理を施し、それぞれ独立した結果を得るような処理を指す。ここでは、LPA 上で RO を効率よく (領域並列的に) 実現するため手法として、**仮想スタック方式**を提案する。仮想スタック方式では、まず最初にタネとなる画素 (タネ画素) の検出を行い、それへのポインタを PE 上の仮想的なスタックエリアにプッシュしておく。次に、スタックから画素へのポインタをポップし、それが指す画素に対し画素処理を行い、かつ常にその周りで新たに処理可能となる画素の検出を行い、検出された処理可能画素へのポインタを、その処理を担当することになる PE 上のスタックにプッシュする。これを、全 PE のスタックが空になるまで繰り返す。こうした処理手順により、処理する必要のある画素のみが処理されていくため、例えば２値画像で背景画素 (すなわち処理する必要のない画素) の多い画像や、線分追跡タイプの処理 (例えば輪郭追跡処理) の場合、特に効果を発揮する。

以下では前節で扱った距離変換を、２値画像の前景領域を対象とした RO と見なし、その仮想スタック方式による実現例を 1DC を用いて示す。forward 時/backward 時のタネ画素検出処理部では、それぞれ LNO により最初から図 4(a) の再帰マスク A,B が示す依存関係を満たす画素を検出する☆。検出されたタネ画素へのポインタは、当該画素をローカルメモリに持つ PE のスタック**stack** に投入される。スタック処理部では、スタック内に画素へのポインタが存在する PE だけが、それをポップして画素に対する距離変換関数dt に渡して処理をしてもらい、全 PE のスタックが空になれば処理は終了する。

```
sep unsigned char src[NROW],tmp[NROW],stack[NROW],ss;
extern sep unsigned char pack8(sep unsigned char *a);
sep unsigned char mtbl1[32] = {
255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
};
sep unsigned char mtbl2[32] = {
 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,
 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
};
#define Poped      0
#define Finished   1
#define Pushed     2
#define tlb(tbl,addr) \
 (idx = pack8(addr),((tbl[idx>>3]>>(idx&7))&1))

void pnbh4(sep unsigned char x, char f)
{
 sep unsigned char a,b,c,d,e=tmp[x];

 if (f){
     a= :>tmp[:<x-1];b= tmp[x-1];c= :<tmp[:>x-1];
     d= :>tmp[:<x];   } else {   d= :<tmp[:>x];
     a= :>tmp[:<x+1];b= tmp[x+1];c= :<tmp[:>x+1];
 }
 mif ((e==Poped) && (a & b & c & d)==Finished)){
     stack[ss++]=x; tmp[x] = Pushed;
 }
}

void stack_proc_dt(src,ltb,f)
sep unsigned char *src, *ltb;
char f;
{
 int i;
 sep unsigned char x,idx;

   /* --- タネ画素検出処理部 -- */
   for(i=0; i<NROW; i++) {
      mif (src[i]) tmp[i] = Poped;
      melse        tmp[i] = Finished;
      mif (src[i] && tlb(ltb,&src[i])){
         stack[ss++]=i;
         tmp[i]=Pushed;
      }
   }
   /* -- スタック処理部 -- */
```

☆ これの検出には、mtbl1,mtbl2 という、それぞれ再帰マスク A,B に対応するビットパックされたルックアップテーブルを使用している。またpack8 は引数のアドレスが指す画素の 3 × 3 近傍の 8 画素を、1 バイトにパックする処理を行う組込み関数

```
  while (:||ss) {
    mif (ss) {
       x=stack[--ss]; stack[ss]=0;
       src[x]= dt(src,x,!f);
       tmp[x]= Finished;
    }
    if (f==1) {
     pnbh4(:<x-1,0);pnbh4(x-1,0);pnbh4(:>x-1,0);
     pnbh4(:<x,0); } else  {     pnbh4(:>x,1);
     pnbh4(:<x+1,1);pnbh4(x+1,1);pnbh4(:>x+1,1);
    }
  }
}

void distance_trans_stack(sep unsigned char *src)
{
 stack_proc_dt(src,mtbl1,0); /* forward 方向  */
 stack_proc_dt(src,mtbl2,1); /* backward 方向 */
}
```

距離変換処理のように、RNO としても RO としても処理を記述可能な場合、並列化方式の選択は、領域サイズに依存する[11]。領域サイズが小さい場合、斜方向シストリック方式では背景画素に対する(無駄な)処理、領域サイズが大きい場合、仮想スタック方式では画素の処理可能性検出処理、がオーバーヘッドとなるため、一般に領域サイズが大きい場合は斜方向シストリック方式、小さい場合は仮想スタック方式がより効率的と考えられる。但し、前者の方式は画素間の依存関係が静的なものしか扱えず、動的な画素間依存関係を持つ RO の場合は、常に仮想スタック方式を用いる必要がある。

5. ま と め

図6に、これまでに述べた LPA 向け画像処理アルゴリズムの並列化方式をまとめる。一次元 SIMD という制限の中で、各 PE に許された自律性をフルに利用することにより、大多数の画像処理アルゴリズムに対し効率的な LPA 向き並列化の手法を見つけることが可能である。また 1DC は、そうした LPA 向き並列アルゴリズムの記述に適した高級言語であるといえる。

今後は、IMAP システムおよび 1DC コンパイラの商用化等に向け、研究開発を進めていく予定である。

謝辞 本研究を進めるにあたり、御指導頂いた天満統括部長、岡崎課長、そして藤田主任に感謝いたします。

参 考 文 献

1) T.J.Fountain, *The CLIP7A Image Processor*,IEEE Trans. on PatternAnalysis and Machine Intelligence, 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) A. Astrom and R.Forchheimer, *MAPP2200 Smart Vision Sensor: Programmability and Adaptivity*, MVA'92, pp.17-20(1992).
4) J.Childers, *SVP: Serial Video Processor*, IEEE 1990 Custom Integrated Circuits Conference, 17.3 (1990).
5) Fujita et al, *IMAP: Integrated Memory Array Processor —Toward a GIPS Order SIMD Processing LSI—*,IEICE Trans. Electron.,Col.E76-C,No.7,pp.1144-1150(1993).
6) 藤田 他,10GIPS IMAP-VISON ボード — ハードウェア —, 信学春季全大, (1996).
7) G. Borgefors: *Distance Transformations in Digital Images*, Computer Vision, Graphics, and Image Processing, Vol.34, pp.344–371, 1988.
8) 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.
9) 許 他: メモリ型画像処理プロセッサ IMAP とその応用, 信学技報, CPSY93-50,pp.39-46,1994.
10) 許 他, メモリ型プロセッサによる動画像処理システム RVS-2 — 基本ソフトウェア —, 第 49 回情処全大,5R-04 (1994).
11) 許: 1 次元プロセッサアレイによる逐次型画像処理アルゴリズムの並列化について, 第 53 回情処全大,4N-07,(1996).
12) A.Tanaka et.al, *A Rotation Method for Raster Image Using Skew Transformation*, Proc. IEEE Conf. on Computer Vision and Pattern Recognition, Miami, pp.272-277 (1986).
13) D.R.Helman, *Efficient Image Processing Algorithms on the Scan Line Array Processor*, Technical Report, University of Maryland (1993).
14) M.Herbordt et al, *A Computational Framework and SIMD Algorithms for Low-level Support of Intermediate Level Vision Processing*, Technical Report of Univ. of Massachusetts at Amherst (1991).

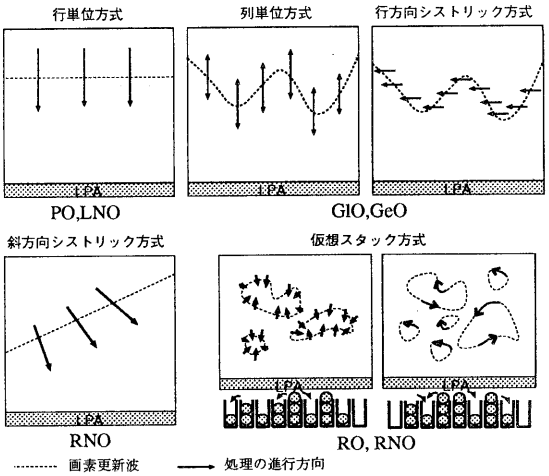


図6 LPA 向け画像処理アルゴリズムの並列化方式