光アレイロジックを用いた並列演算法に関する研究

Parallel Computing Methods Based on Optical Array Logic

1992 年

岩 田 昌 也

Masaya Iwata

目次

緒論

近年，電子に代わる情報媒体として光が注目されている．これは，光が超並列・超高速信号処理の可能性を秘めているからである．例えば，レンズの結像作用を用いることにより，レンズ１枚で画像を別の平面に伝送することができる．このことは，画像を構成する膨大な２次元データが超並列・超高速に伝送されたことを意味する．このような光の特長を有効に利用した超並列・超高速情報処理システムとして，光コンピュータの研究が活発に行われている [1-6].

近年の情報化社会の要望に答えるべく，電子コンピュータの技術はこの 30 年で急速に進歩し，その技術の粋を集めた超高速コンピュータであるスーパーコンピュータが開発されるに至っている [7, 8]．しかし，電子コンピュータの性能が向上するにつれて，その能力向上を制約する本質的な要因がクローズアップされてきている．それらは，CPU からメモリへの逐次アクセスによる信号伝送速度の限界（フォン・ノイマンボトルネック），信号通信帯域・バンド幅の限界，論理ゲートに到達する信号の時間遅れによる誤動作（クロックスキュー），素子の熱の問題，配線の増大に伴う配線領域や素子の端子数の制約，などである．これらは，電子計算機の基本構成方式や，情報媒体として電子を用いていることに起因している [4].

情報媒体として光を用いると，上記の問題を一掃できる可能性がある．情報媒体としての光は，以下の特長を持つ [4].

１）データの超並列伝送・処理が可能.

２）超高速性（3×10^8 m / s），超広帯域性（可視光は約 100 THz）.

３）配線が不要であり，３次元自由空間信号接続が可能.

４）信号相互間で非干渉であり，交差接続が可能.

５）信号相互間をいくら近接させても無誘導である.

これらの特長を有効利用して，大容量情報の並列処理を行いうるコンピュータの開発が，光コンピュータ研究の目標である.

一方，電子コンピュータの分野においても，最近，多数のプロセッサにより並列処理を行う並列コンピュータが注目され，様々の開発研究が行われている [9-16]．並列コンピュータは，並列処理データに対する命令の与え方により，SIMD (Single Instruction stream Multiple Data stream) と MIMD (Multiple Instruction stream Multiple Data stream) の方式に分類できる．SIMD 方式では，全プロセッサが同一命令を同時に実行する．MIMD 方式では，各プロセッサが独立に異なった命令を実行する．両方式とも様々なシステム構成法が提案されている．残念ながら，現状ではこれらの並列コンピュータを効率よく駆動させるための有効なプログラミング技法は確立されていない.

並列コンピュータでは，従来のコンピュータのアルゴリズムとは異なる並列アルゴリズムに基づいたプログラムが必要となる．すなわち，処理対象から並列性を抽出し，その並列性をできるだけ損なわないように各プロセッサを効率よく動作させる必要がある．しかし，従来の逐次処理プログラムとはアルゴリズムの考え方が大きく異なるため，並

列プログラムの作成は困難である．また，並列コンピュータはまだ開発段階にあるため，並列プログラミング環境が整備されていない．これらの原因により，まだ統一的な並列演算法は確立されていないと言える [13, 17].

現在，並列光コンピュータの研究は，新しい光デバイス開発，並列演算原理，並列演算法，システム構成法，システム試作など多方面から進められている．その中でも，並列光演算原理の研究は，並列光コンピュータの構成法を決定する上で非常に重要である．現在までに，並列光演算原理として，光アレイロジック [18]，記号置換 [19]，二値画像代数 [20]，画像論理代数 [21]，などが提案されている．これらは全て処理データを２値離散画像で表現し，画像のシフトと重ね合わせを基本処理として並列光演算を実現する．このとき，画像上の全データに対する SIMD 方式の並列処理が行われる．これらの並列光演算原理を土台として光コンピュータのシステム構成法や並列演算法の開発を行うことができる．特に，光コンピュータの並列演算法を開発し，有効な応用分野を特定することは，システム構築の指針となるだけに，急務である．

現在までに，提案された上記の並列光演算原理を用いて，整数演算，２値画像処理，濃淡画像処理，並列処理マシンの仮想的実現などの並列演算法が開発されている [1, 3, 19, 21-29]．しかし，これらの演算法は，データを直接２値画像で容易に表現でき，演算も比較的簡単なものばかりである．光コンピュータが有する超並列処理の可能性を示すためには，さらに多岐にわたる複雑な処理に対する並列演算法の開発が必要である．

本研究の目的は，大阪大学工学部応用物理学科一岡研究室において開発された並列光演算原理である光アレイロジックを用いて，並列光コンピュータの新たな並列演算法の開発と評価を行い，光コンピュータによる超並列処理の可能性を明らかにすることである．光アレイロジックの特徴として，超並列処理が可能，光学的に実現可能，並列プログラミングが可能などの点が挙げられる．さらに，(1) 光アレイロジックには任意の並列近傍画素間論理演算をプログラミングできる，(2) 専用のプログラミング言語が開発されているため，並列プログラミング環境が整っている，という特長を有している．現在までに，光アレイロジックを用いて簡単な画像処理，数値処理，並列処理マシンの仮想的実現などの並列演算法が開発されている [25, 27-32]．しかし，光アレイロジックが有する汎用超並列処理の可能性を明確にするためには，さらに多岐にわたる応用分野において，並列演算法を開発しなければならない．すなわち，並列プログラミング環境を十分に利用して，より複雑な処理の並列演算法を開発する必要がある．光アレイロジックの超並列処理可能性と有効な応用分野を明らかにすることは，未だ構成法が確立していない並列光コンピュータの構成法の明確化と応用分野の具体化に大きく寄与すると考えられる．

本研究では，光アレイロジックを用いて新たな並列演算法の開発・評価を行い，並列光コンピュータの有効な応用分野を明らかにする．さらに，これらの並列演算法の光学的実現を容易にするため，２次元仮想記憶機構を新たに考案する．また，光アレイロジックによる処理効率を向上させるため，専用光演算モジュールを用いた新しいシステムの構成法を考案する．

以下に本論文の構成と内容を示す．

第１章では，光アレイロジックによる並列プログラミングの概要と，そのプログラム記述法を述べる．そして，専用プログラミング言語であるOALLについて概説し，基本的な並列プログラミング技法を述べる．さらに，本研究で拡張した光アレイロジックの機能を述べる．

第２章では，光アレイロジックの並列性を知識ベースの探索に有効利用した並列推論機構の実現方法を二種類考案する．さらに，推論機構の機能を拡張してエキスパートシステムの処理を実現する方法を検討する．そして，これらの処理の処理効率を評価する．

第３章では，並列光演算システムへの応用を目指して，光アレイロジックの並列性をトークンの並列転送に有効利用したデータフロー型処理の実現方法を検討し，処理効率を評価する．

第４章では，光アレイロジックの並列性を大容量データ処理に有効利用する手法として，データベース処理を検討し，処理効率を評価する．また，画素パターンの移動・複写処理の専用光演算モジュール化により，ハードウェア資源に対する要求が軽減できることを示す．

第５章では，並列光演算システムにおいて，小規模のハードウェアで大画素数の画像処理を実現する方法として，２次元仮想記憶機構を考案する．各種並列光演算原理の処理手順を共通に記述するために，相関演算モデルを導入し，そのモデルを用いて２次元仮想記憶機構の処理手順・処理効率を検討する．

第６章では，今までに開発された光アレイロジックプログラムの処理特性を解析する．その結果に基づき，光アレイロジックによる処理効率を向上させるシステムである，専用光演算モジュールを用いた並列光演算システムの構成法を提案する．

最後に，本研究の研究成果について総括し，今後の研究課題について述べる．

第1章　光アレイロジックによる並列プログラミング

1.1　緒言

並列光コンピュータの研究においては，システムの構成法とプログラミング法を決定する並列光演算原理の研究が重要である．これまでに，並列光演算原理として，光アレイロジック [18]，記号置換 [19]，二値画像代数 [20]，画像論理代数 [21] などが報告され，それぞれの演算原理に対して，システム構成法，システム試作，並列演算技法などの研究が行われている．

光アレイロジックは，２枚の２値画像に対し任意の近傍画素間論理演算を並列に実行する並列光演算原理である．この特長として，

１）画像上のデータに対する超並列処理が可能，

２）並列プログラミングが可能，

３）光学的実現が可能，

が挙げられる．特に，２）は本研究において重要な意味を持つ．光アレイロジックは論理代数を基本としており，既存の情報処理技術との親和性が高い並列プログラミングが可能である．さらに，光アレイロジックでは専用並列プログラミング言語が開発されている．したがって，光アレイロジックは他の並列光演算原理に見られない並列プログラミング環境を有する．光アレイロジックによる並列演算技法の開発は，光コンピュータの研究において，システムの開発指針を得るうえで不可欠である．

本章では，本研究で扱う並列演算原理である光アレイロジックの概念とその並列プログラミング法について述べる．まず，１.２節では光アレイロジックの概念，処理手順，光学的実現方法を概説する．１.３節では光アレイロジックのプログラム記述法について述べ，１.４節では専用プログラミング言語である OALL を概説する．１.５節では光アレイロジックの基本的なプログラミング技法を述べ，１.６節では本研究で新たに拡張した光アレイロジックの機能について述べる．

1.2　光アレイロジック

1.2.1　光アレイロジックの概念

光アレイロジックは，２枚の２値画像に対し，任意の近傍画素間論理演算を実行する技術である．近傍画素間論理演算の論理式は一般的に次のように書ける．

$$c_{i,j} = \sum_{k=1}^{K} F_k(a_{i,j}, b_{i,j}) \qquad (i, j = 1, 2, \cdots\cdots, N), \tag{1-1}$$

$$F_k(a_{i,j}, b_{i,j}) = \prod_{m=-L}^{L} \prod_{n=-L}^{L} f_{m,n;k}(a_{i+m,j+n}, b_{i+m,j+n}). \tag{1-2}$$

ここで，$a_{i,j}$, $b_{i,j}$, $c_{i,j}$ は，それぞれ入力画像 A, B と出力画像 C の座標 (i, j) における画素値を表す論理変数である．$f_{m,n;k}(a_{i+m,j+n}, b_{i+m,j+n})$ は，座標 (i, j) の画素を中心とする近傍領

域内の局所座標 (m, n) にある対応画素対に対する2変数2値論理関数を表す．Σ, Π はそれぞれ論理和と論理積の演算子を表す．$F_k(a_{i,j}, b_{i,j})$ は，論理変数の積からなる項であり，積項と呼ばれる．L, K はそれぞれ演算対象となる近傍画素の大きさ，積項数を表す．

光アレイロジックでは上式の演算を Fig.1.1(a) の処理手順で実行する．2枚の2値画像に対し，符号化，相関演算，サンプリング，反転，並列論理和の手順を行い，演算結果として1枚の2値画像を得る．演算内容は，相関演算で用いられる演算カーネルのパターンにより記述する．この演算カーネルは，各積項ごとに1枚ずつ作成し，相関演算から反転までの手順は，各積項を表す演算カーネルそれぞれについて行う．

次に，各手順について説明する．まず，2枚の2値入力画像を，1枚の符号化画像に変換する．符号化では，2枚の画像を，それぞれ対応する位置にある画素対ごとに，符号化ルールによって変換する（Fig.1.1(b)）．符号化画像に対して，各積項の演算内容を表す演算カーネルとの間で相関演算を行う（Fig.1.1(c)）．相関演算結果に対して，縦横1画素おきに空間的サンプリングを行う（Fig.1.1(d)）．サンプリングする画素の位置は，1画素が符号化された2×2画素のセルのうち，左上の画素である．サンプリング後の画像の各画素を縦横2倍の大きさに拡大し，各画素値を反転する（Fig.1.1(e)）．以上の手順により，1積項の演算結果を得る．

一般的な論理式は，積項の和の形で記述できる．したがって，任意の並列近傍画素間論理演算では，各積項を表す演算カーネルに対してそれぞれ上記手順の処理を行い，得られた演算結果の並列論理和をとることにより，最終的な結果を得る（Fig.1.1(f)）．

ここで，本論文で用いられる用語のうち，狭義で用いられるものや，紛らわしい用語について定義しておく．

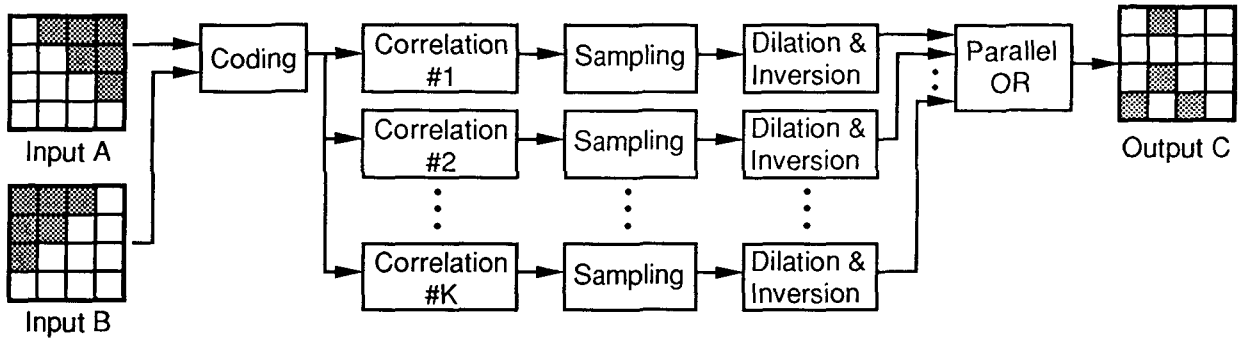
画像とは，特に断らない限り，光アレイロジックの処理対象である2値離散画像を指す．

符号化は，本論文では2種類の意味がある．一つは光アレイロジックの処理手順における符号化，もう一つはデータを2値画像に変換するという意味での符号化である．前者の意味は，前述の光アレイロジック処理手順の説明のみで使用する．よって，本論文では以降符号化を後者の意味のみで使用する．

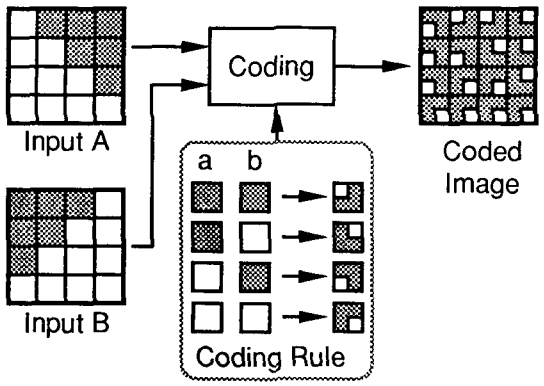
1.2.2　特徴

光アレイロジックの第一の特徴は，処理の超並列性である．光アレイロジックでは，画像上に配置された2次元データ配列に対して，並列に演算を実行する．例えば，1000×1000 画素の2値画像を扱うことにより，10^6 ビットのデータを並列に処理できる．その演算形式は，画像上の全データに対して同一の命令を実行する SIMD (Single Instruction stream Multiple Data stream) 形式の並列処理である．

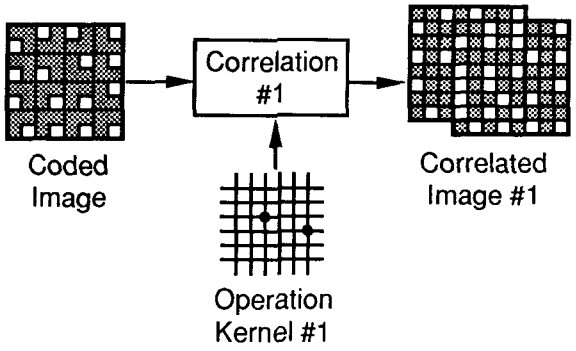
次に，光アレイロジックは，並列プログラミング可能であるという大きな特徴を持つ．光アレイロジックでは，求める演算の論理式の積項ごとに作成した演算カーネルにより，任意の近傍画素間演算を実現する．演算カーネルと，その演算対象となる2枚の画像の組を実行順に並べると，並列プログラムを記述できる．光アレイロジックでは，演算カーネルを直観的にわかりやすく記述するカーネル式が導入され，さらに専用プログラム



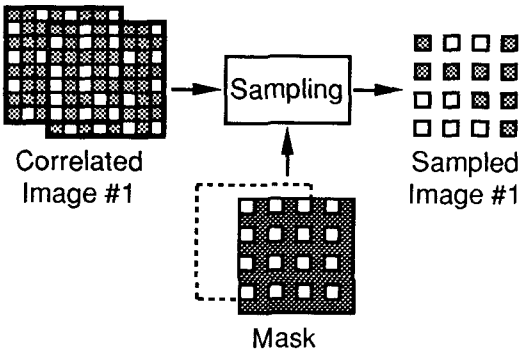
(a)　ブロック図



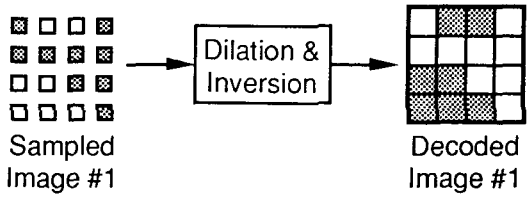
(b)　符号化



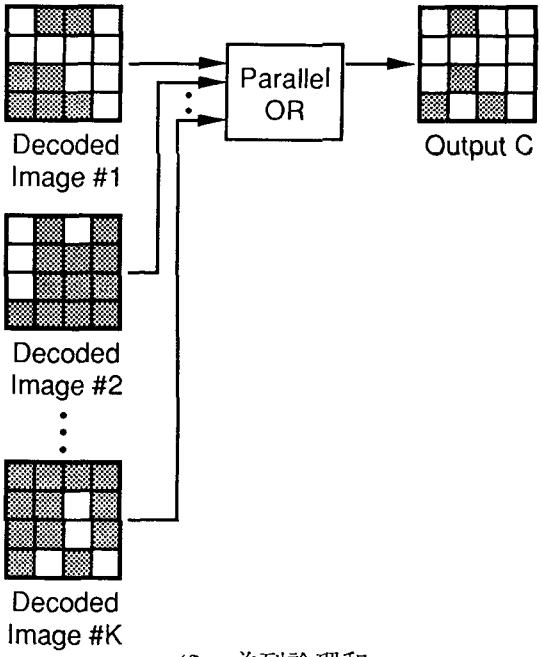
(c)　相関演算



(d)　サンプリング



(e)　画素拡張と反転



(f)　並列論理和

Fig.1.1　光アレイロジック処理手順

言語としてOALLが開発されている．これらを用いると並列アルゴリズムを明確かつ詳細に記述できる．現在，並列アルゴリズムはまだ確立されておらず，光アレイロジックはその研究・開発のための基本原理として非常に有効である．本研究では，この特徴を利用して並列演算技法の開発を行っている．

さらに，光アレイロジックは光学的に実現可能である特徴を持つ．光アレイロジックの処理はディジタル相関演算を基本とする．これは画像のシフトと重ね合わせからなり，多重投影光学系（Fig.1.2）[33]，離散相関光学系 [34] などを用いて容易に実現できる．

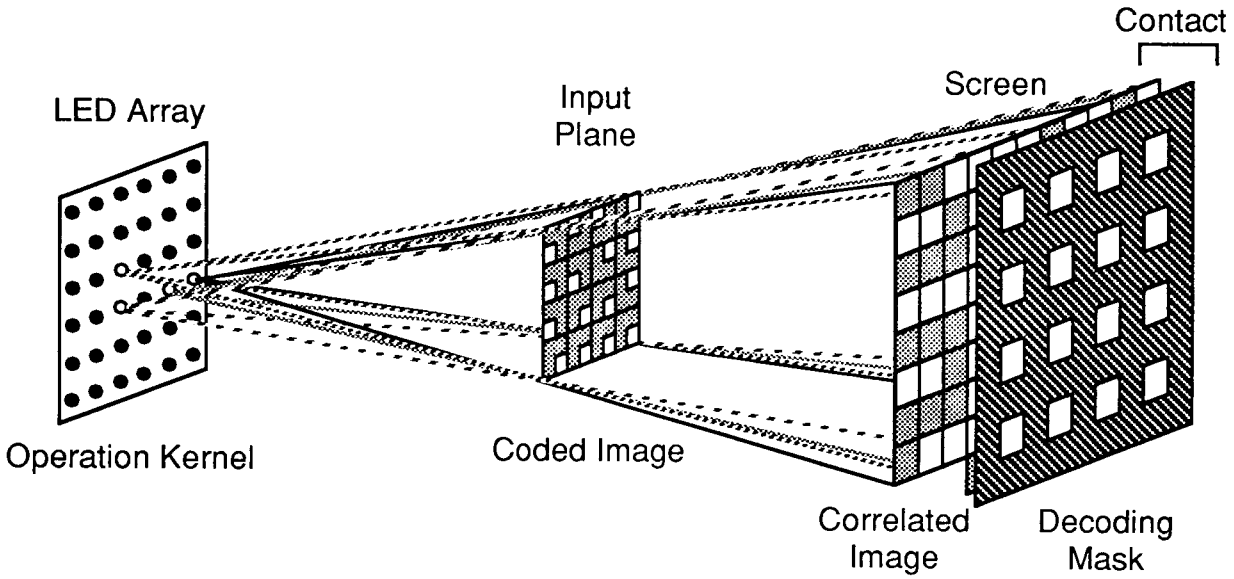


Fig.1.2　多重投影相関光学系

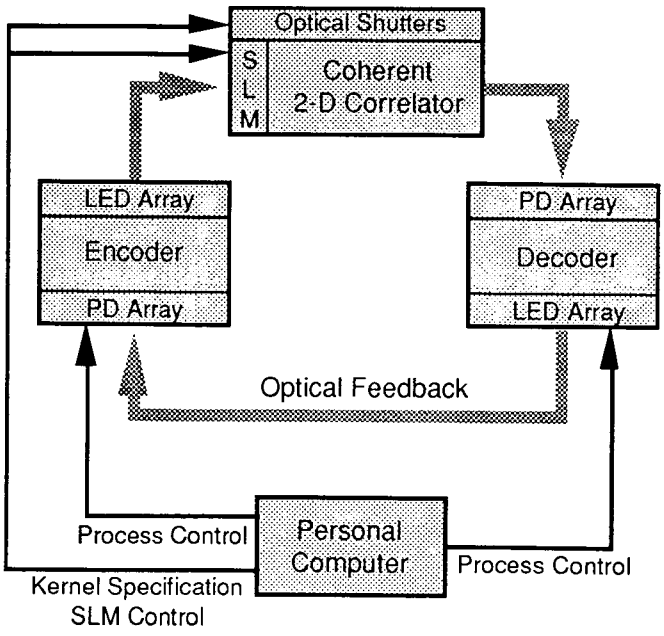


Fig.1.3　光・電子複合型 OPALS ブロック図

光アレイロジックを光学的に実現する汎用光演算システムとしてOPALS (Optical Parallel Array Logic System) が提案され，試作が行われている [31, 35-38]．Fig.1.3に，光・電子複合型OPALSのブロック図を示す．本研究の主な目的は，OPALS駆動用アルゴリズム（ソフトウェア）の開発と考えることができる．

1.3　プログラム記述法

1.3.1　カーネル式の表記法

演算カーネルは，近傍画素間演算の論理式と明確に対応している．そこで，演算カーネルは，論理演算の記号表現を用いて直観的に理解しやすい形で記述できる．この表現法をカーネル式 [25] と呼ぶ．カーネル式は，並列プログラムをわかりやすく表現するために非常に重要である．以下では，カーネル式の表記法について説明する．

カーネル式の主な構成要素は，以下のようなものである．

・積項ブロック（PTB: Product Term Block）

一つの演算カーネルは1積項の演算を指定する．そこで，1演算カーネルに対応する演算ブロックを導入する．この演算ブロックを積項ブロック，またはPTBと呼ぶ．

・演算記号

PTBの構成要素は，2変数2値論理関数を表す2文字の記号である．記号の意味をTable 1.1に示す．記号の左，右の文字はそれぞれ入力画像 A, Bに対する演算を指定する．2変数に対する論理積と論理和の両方を表現するため，2種類の文字集合が使われる．すなわち，'1,' '0,' '.' は論理積で，'P,' 'N' は論理和で使用する．ただし，'UU,' 'EE,' 'DD' は特別な記号である．

Table 1.1　カーネル式表現用記号

Symbol	Function	Kernel Unit	Symbol	Function	Kernel Unit
. .	1		PP	$a+b$	
NN	$\bar{a}+\bar{b}$		UU	$a \oplus b$	
NP	$\bar{a}+b$		.1	b	
0.	$\bar{a}$		01	$\bar{a}b$	
PN	$a+\bar{b}$		1.	a	
.0	$\bar{b}$		10	$a\bar{b}$	
EE	$\overline{a \oplus b}$		11	ab	
00	$\bar{a}\bar{b}$		DD	0	

・近傍画素へのマッピングと原点マーカ

求める近傍画素間論理演算の演算対象となるデータは，PTB内の記号の位置によって指定する．すなわち，一つのPTBは直接近傍画素領域に対応する．PTBの原点は下線で示す．PTBが一つの記号しか持たない場合，またはPTBの左上隅が原点に相当する

場合は，下線を省略できる．近傍画素領域の座標軸は，X, Y 方向がそれぞれ下，右方向に対応するものとする．

カーネル式は対応画素間演算を表す演算記号を近傍画素の空間的位置に対応するように2次元配列の形に書き表したものである．カーネル式は一般的に次のように書ける．

$$\sum_{k=1}^{K}\begin{bmatrix} S_{-L,-L;k} & \cdots & S_{-L,L;k} \\ \vdots & \underline{S_{0,0;k}} & \vdots \\ S_{L,-L;k} & \cdots & S_{L,L;k} \end{bmatrix}. \tag{1-3}$$

ここで，$S_{m,n;k}$ は (1-1) 式の $f_{m,n;k}(a_{i+m,j+n}, b_{i+m,j+n})$ を Table 1.1 にしたがって演算記号で表したものである．角カッコ [ ] で囲まれた部分が一つの PTB を表し，(1-1) 式の1積項 $F_k(a_{i,j}, b_{i,j})$ に対応する．Σ は PTB の和を表す．すなわち，(1-3) 式は (1-1) 式と等価である．

例として，次のカーネル式を考える．

$$\begin{bmatrix} .. & .. & .. \\ 10 & \underline{0.} & .. \\ .. & .. & .. \end{bmatrix} + \begin{bmatrix} .. & .. & .. \\ .. & \underline{..} & 1. \\ .. & .. & .0 \end{bmatrix}. \tag{1-4}$$

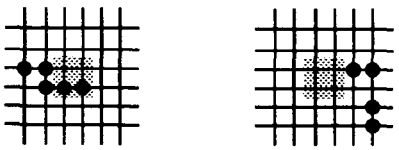
PTB が表す近傍画素領域の原点を，画像の座標 (i, j) で表すと，この式は，次の論理式で書ける．

$$c_{i,j} = a_{i,j-1}\bar{b}_{i,j-1}\bar{a}_{i,j} + a_{i,j+1}\bar{b}_{i+1,j+1}. \tag{1-5}$$

(1-4), (1-5) 式は，Table 1.1 に示すように，PTB 内の各演算記号と対応画素間演算が1対1に対応しているため，互いに容易に変換できる．

光アレイロジックの処理手順により演算を実行する場合は，カーネル式を演算カーネルに変換する必要がある．これは，Table 1.1 を用いて PTB の各記号ごとに，カーネルユニットと呼ばれる2×2の格子パターンに変換することにより，簡単に実現できる．例として，(1-4) 式の演算カーネルを Fig.1.4 に示す．この図では，原点のカーネルユニットを網かけで示している．

以上のように，近傍画素間演算の記述法は3種類ある．一般にプログラミングには，これらの中で演算内容を一番理解しやすい記述法である，カーネル式を用いる．本論文



(a) (1-4) 式第1項　　(b) (1-4) 式第2項

Fig.1.4　演算カーネル例

では，以降演算の記述にはカーネル式を用いる．

１．３．２　カーネル式の拡張

カーネル式は，前項の記述形式だけを用いても基本的な演算は記述できるが，より高度なプログラミングに使用するためには，不十分な点がある．カーネル式は，論理式と１対１で対応しているため，論理式と同様の式変形が行えれば，プログラム開発において便利になる．そこで，カーネル式に対していくつかの拡張を行う．

・オフセット指定

求める演算の対象となるデータが，近傍画素領域の原点から離れている場合がある．その場合に表記を簡単にするため，PTB の右下に PTB 全体のオフセット量を指定する添字を導入する．この添字は，下線が引かれた位置の座標を示すと考えることができる．オフセット指定と PTB 全体のオフセット量の関係は，次のように書ける．

$$\begin{bmatrix} S_{-L+m,\,-L+n} & \cdots & S_{-L+m,\,L+n} \\ \vdots & \underline{S_{m,\,n}} & \vdots \\ S_{L+m,\,-L+n} & \cdots & S_{L+m,\,L+n} \end{bmatrix}_{m,\,n} = \begin{bmatrix} S_{-L,\,-L} & \cdots & S_{-L,\,L} \\ \vdots & \underline{S_{0,\,0}} & \vdots \\ S_{L,\,-L} & \cdots & S_{L,\,L} \end{bmatrix}. \tag{1-6}$$

なお，オフセット値が記されていない場合は，オフセット値 (0, 0) が省略されているものとみなす．

この記法により，カーネル式の簡単化が行える．例えば，(1-4) 式はオフセット記法を用いて，次のように書ける．

$$[10\,\underline{0.}] + \begin{bmatrix} 1. \\ .0 \end{bmatrix}_{0,\,1}. \tag{1-7}$$

すなわち，記号 '.' は特に指定する必要がないので，この記法を用いて省略することができる．また，(1-7) 式第２項のように，オフセット指定位置がカッコ内の左上隅であるか，あるいはカッコ内の演算記号が一つである場合は，下線を省略できる．

・PTBの積

カーネル式において，論理式と同様の式変形を可能にするために，PTB の積を導入する．PTB の積は，各 PTB の対応する位置にある記号が表す関数の論理積として定義する．一般的には次のように書ける．

$$\begin{bmatrix} S^1_{-L,\,-L} & \cdots & S^1_{-L,\,L} \\ \vdots & \underline{S^1_{0,\,0}} & \vdots \\ S^1_{L,\,-L} & \cdots & S^1_{L,\,L} \end{bmatrix} \begin{bmatrix} S^2_{-L,\,-L} & \cdots & S^2_{-L,\,L} \\ \vdots & \underline{S^2_{0,\,0}} & \vdots \\ S^2_{L,\,-L} & \cdots & S^2_{L,\,L} \end{bmatrix} = \begin{bmatrix} S^1_{-L,\,-L}S^2_{-L,\,-L} & \cdots & S^1_{-L,\,L}S^2_{-L,\,L} \\ \vdots & \underline{S^1_{0,\,0}S^2_{0,\,0}} & \vdots \\ S^1_{L,\,-L}S^2_{L,\,-L} & \cdots & S^1_{L,\,L}S^2_{L,\,L} \end{bmatrix}. \tag{1-8}$$

PTB 内の各記号の積は，記号ごとに Table 1.1 を用いて論理式に変換して積を行い，その結果を記号に戻すことにより求められる．カーネル多項式の場合は，分配則により展開した後，PTB の積を求める．PTB の積を用いたカーネル式の簡単化の例を示す．

$$\begin{aligned}[1.]([.0\ \ .1]+[.0\ \ .0\ \ .1]) &= [1.][.0\ \ .1]+[1.][.0\ \ .0\ \ .1] \\ &= [10\ \ .1]+[10\ \ .0\ .1]\ .\end{aligned} \tag{1-9}$$

通常，カーネル式では，入力画像 A, B に対する対応画素間演算を一つの演算記号で表現するが，上式の左辺は，各演算記号を，それぞれの画像に対する演算に分離して記述した形になっている．プログラミング時には，この記述の方がわかりやすい場合がある．このように，PTB の積を用いると，多様な式表現が可能となるため，よりわかりやすい表現方法で式の記述が可能になる．

・PTB の否定

PTB の否定は，論理式の否定と同様にド・モルガンの定理にしたがって行う．すなわち，PTB 内の各記号は積の関係にあるため，PTB の否定は，各記号の否定の和になる．一般的には，次のように書ける．

$$\overline{\begin{bmatrix} S_{-L,-L} & \cdots & S_{-L,L} \\ \vdots & \underline{S_{0,0}} & \vdots \\ S_{L,-L} & \cdots & S_{L,L} \end{bmatrix}} = \overline{[S_{-L,-L}]}_{-L,-L} + \bullet\bullet\bullet\bullet + \overline{[S_{L,L}]}_{L,L}\ . \tag{1-10}$$

PTB 内の各記号の否定は，記号ごとに Table 1.1 を用いて論理式に変換して否定を行い，その結果を記号に戻すことにより求められる．カーネル多項式の場合は，PTB の和の否定をド・モルガンの定理により PTB の否定の積に直した後，各 PTB の否定を行う．カーネル式の否定の例を次に示す．

$$\begin{aligned}\overline{[1.\ \ .0]+[1.]_{0,1}} &= \overline{[1.\ \ .0]}\,\overline{[1.]}_{0,1} \\ &= (\overline{[1.]}+\overline{[.0]}_{0,1})\overline{[1.]}_{0,1} \\ &= ([0.]+[.1]_{0,1})[0.]_{0,1} \\ &= [0.\ \ 0.]+[01]_{0,1}\ .\end{aligned} \tag{1-11}$$

以上のようなカーネル式の拡張により，論理式と同等の式変形が行えるため，カーネル式を多様な形で表現できる．

1.4　OALL

1.4.1　概要

光アレイロジックで並列プログラムを記述する場合，カーネル式による演算内容以外に，演算対象画像や演算実行順序の制御を，明確かつ詳細に記述する必要がある．そのために，光アレイロジック専用のプログラム言語として，OALL (Optical Array Logic

Language) が開発されている．

OALLは，カーネル式とそれによる演算実行の記述を基本とし，既存のプログラム言語を参考にしてデータの型，文の構造，文の制御などの表現を整えたものである．OALLは，JISキーボードからコンピュータに入力することを想定し，すべてJISキーボードから入力可能な文字で記述できる．

このプログラム言語の特長として，並列光演算に適したデータ型を持つことが挙げられる．データ型には，整数を扱うvar型の他に，画像を扱うimage型，カーネル式を扱うkernel型がある．image型の変数を用いると，多量のデータを表す画像を1変数で表すことができる．また，その画像上の全データは並列に処理されるため，多量のデータに対する並列処理を非常にコンパクトに記述できる．

文としては，光アレイロジックの演算実行を指定するexec文が中心となる．exec文の実行順序や，カーネル式の記述を補助する制御文として，if文，loop文，for文があり，複雑な構造を持つアルゴリズムも簡潔に記述できる．

1.4.2　文法

OALLのプログラム構成は，次のようになる．

```
program <プログラム名>;
    <宣言文>
    ......
    <実行文>
    ......
end <プログラム名>;
```

OALLプログラムは，主に宣言文と実行文からなる．それらを補助する文として，コメント文，制御文などがある．以下，プログラムの各構成要素ごとに文法の概説をする．

・予約語

OALLの予約語は，以下の通りである．

condition	exit	kernel	to
do	for	loop	var
else	if	null	
end	image	program	
exec	imout	then	

・定数

定数は，以下の3種類がある．

１）整数定数

数字の並びによる構成される整数定数は，10進数である．

２）カーネル定数

カーネル定数は，カーネル式と等価である．ただし，キーボードからの文字入力の便宜上，カーネル式の各 PTB は次のように表記する．

|<PTB１行目>||<PTB２行目>|・・・・・|<PTBｋ行目>|@(m,n)

これは，次のように PTB１行ごとに改行することにより，PTB を近傍画素領域の配置に対応したわかりやすい表記にすることができる．

```
|<PTB１行目>|
|<PTB２行目>|
・・・・・・・・・
|<PTBｋ行目>|@(m,n)
```

PTB の各行は，Table 1.1 の記号と，それらの区切りを示す空白または _（下線記号）により構成される．_ は，その直後の記号に下線が引かれていることを表す．定数の最後尾の @(m,n) は，オフセット指定記号であり，オフセット値が(m,n)であることを表す．ただし，オフセット値が(0,0)である場合は，オフセット指定の記号を省略できる．また，_ を省略した場合には，第１行左端の位置が局所座標(m,n)であることを表す．例として，(1-7) 式の表記を示す．

```
|10_0.| + |1.|
           |.0|@(0,1)
```

特殊なカーネル定数として，内容を持たないことを示す定数 null がある．これは，kernel型変数（後述）の内容をクリアする場合などに用いる．

３）画像定数

画像定数は，次のように表記する．

/<画像１行目>//<画像２行目>/・・・・・/<画像ｎ行目>/@(m,n)

これは，次のように画像データ１行ごとに改行することにより，画像の２次元配列に対応した，わかりやすい表記にすることができる．

```
/<画像１行目>/
/<画像２行目>/
・・・・・・・・・
/<画像ｎ行目>/@(m,n)
```

各行の画像データは，画素値を表す０と１で構成される数字並びである．定数の最後尾の@(m,n)は，第１行左端の画素の座標指定である．ただし，m＝0, n＝0 である場合は，@(m,n)を省略できる．例として，Fig.1.1 の Input A の表記を示す．

```
/1000/
/1100/
/1110/
/1111/
```

・コメント文

コメント文は，/* で始まり，*/ で終わる．コメント文は入れ子にできる．

・変数宣言文

OALL で扱える定数は３種類ある．これらを扱う変数を利用する場合は，型に応じた変数の宣言を行う．変数宣言は，次のように表記する．

<型> <変数名> [, <変数名>, ・・・] [= <初期化データ>];

型：var, kernel, image

var, kernel, image 型は，それぞれ整数定数，カーネル定数，画像定数を扱う．初期化データは，image型のみで使用でき，画像定数のファイル名で指定する．なお，[] で囲まれた表記は省略可能であることを表す．

・代入文

代入文は，右辺の式を評価し，左辺の式に代入する．このとき，両辺のデータの型は一致していなければならない．

・演算子

整数定数とvar型変数に対しては，四則演算が許されており，それぞれ演算子 +, -, *, / を用いて表す．演算結果は整数定数となる．画像定数は，image型変数への代入のみが許される．カーネル定数とkernel型変数に対しては，カーネル式と等価な演算，すなわち和，積，否定が許されており，それぞれ演算子 +, *, ~ を用いて表す．演算子の優先順位は ~, *, + の順で高い．

・演算実行文

光アレイロジックの演算の実行は，次の形式で記述する．

```
<image型変数3> = exec(<image型変数1>,<image型変数2>,<kernel型変数>);
```

<image型変数1>，<image型変数2>をそれぞれ入力画像A, Bとし，<kernel型変数>のカーネル式を用いて演算が行われる．演算結果は<image型変数3>に代入される．

・制御文

プログラムの流れを制御する文として，if文，for文，loop文，exit文がある．

if文は，次の2種類の記述形式がある．

```
if <条件文> then <実行文1> end;
if <条件文> then <実行文1> else <実行文2> end;
```

どちらの場合も，<条件文>が評価され，結果が真の場合，<実行文1>が実行される．2番目の文の場合は，<条件文>が偽の場合，<実行文2>が実行される．ただし，<条件文>は，var型変数と整数定数間，または二つのvar型変数間の関係を表す等式または不等式である．

for文は，次の形式で記述する．

```
for <var型変数> = <整数定数1> to <整数定数2> do
    <実行文>
    ......
end;
```

この文は，<var型変数>の値を<整数定数1>の値から一つずつ増加させながら，<整数定数2>の値を越えるまで<実行文>の並びを反復実行させる．

loop文は，次の形式で記述する．

```
loop
    <実行文>
    ......
end;
```

この文は，loopとendで挟まれた<実行文>を反復実行させる．この反復実行を終了させるには，exit文を用いる．反復処理実行中にexit文が実行されると，loopと対応するendの次の文に処理が移る．

・画像ファイル出力文

指定された画像定数のファイルを出力するimout文は，次の形式で記述する．

```
imout <ファイル名> <image型変数> ;
```

<image型変数>で指定された画像定数のファイルを作成し，指定されたファイル名で出力する．

・モード

画像の大きさは有限であるため，近傍画素間演算実行時に，画像の外側の状態が未定義であると不都合が生じる場合がある．そこで，この問題を避けるため，画像の外側が0または1の画素で満たされていると仮定する．このとき画素値の違いはモードとして区別され，0の場合を標準モード，1の場合を反転モードと呼ぶ．光アレイロジックプログラムでは，特に指定のない限り標準モードを仮定する．反転モードは，そのモードで実行するカーネル式を< >の記号で囲むことにより指定する．

1.5　プログラミング技法

光アレイロジックの並列性を有効に利用して効率的な処理を行うためには，従来のコンピュータとは異なる考え方でプログラミングする必要である．そこで本節では，プログラミング技法として，光アレイロジックのプログラミングでよく使用される有用な演算のプログラミング方法を述べる．

1.5.1　基本的パターン操作

(a)　シフト演算

シフト演算は，画素パターンを同一画像内で移動させる処理である．画素パターンを(x, y)方向に(m, n)だけシフトさせる演算のカーネル式は，次のように書ける．

$$[1.]_{-m,-n} \quad \text{(for Input A)}, \tag{1-12}$$

$$[.1]_{-m,-n} \quad \text{(for Input B)}. \tag{1-13}$$

(b)　テンプレートマッチング

テンプレートマッチングは，テンプレートと呼ばれる特定の画素パターンが存在する位置を検出する処理である．光アレイロジックでは，テンプレートを演算カーネルにより指定する方法と，入力画像で指定する方法がある．

まず，テンプレートを演算カーネルにより指定する方法を説明する．テンプレートにおいて，画素値1を持つ画素の局所座標の集合をtemp1，画素値0を持つ画素の局所座標の集合をtemp0とすると，入力画像A, Bに対するテンプレートマッチングはそれぞれ次のようなカーネル式で実行できる．

$$\prod_{(m,\,n)\in \mathrm{temp1}} [1.]_{m,\,n} \prod_{(p,\,q)\in \mathrm{temp0}} [0.]_{p,\,q} \qquad \text{(for Input A)}, \tag{1-14}$$

$$\prod_{(m,\,n)\in \mathrm{temp1}} [.1]_{m,\,n} \prod_{(p,\,q)\in \mathrm{temp0}} [.0]_{p,\,q} \qquad \text{(for Input B)}. \tag{1-15}$$

Fig.1.5 に，５画素からなる横長のテンプレートを入力画像 A から検出するテンプレートマッチングの例を示す．テンプレートの存在する位置の左端の画素に画素値１が得られる．

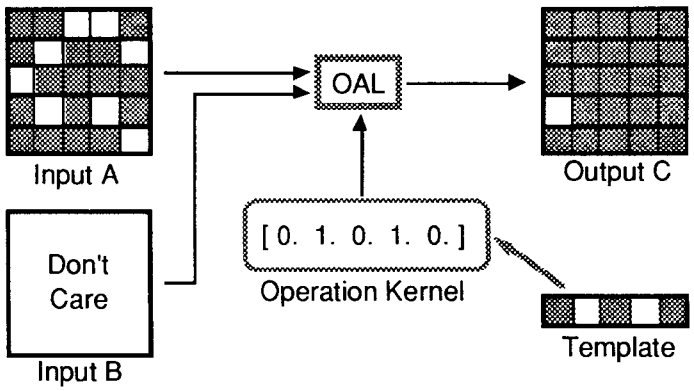


Fig.1.5　テンプレートを演算カーネルから指定するテンプレートマッチングの例

次に，テンプレートを入力画像により指定する方法を説明する．入力画像 A に対するテンプレートマッチングを考える．まず，入力画像 B 上で，マッチングの対象となる領域にテンプレートを複製して配置する．そして，テンプレートの領域と，入力画像 A 上の対応する画素領域との排他的論理和を行えば，並列テンプレートマッチングを実現できる．テンプレートの領域を表す局所座標の集合を temp とすると，このテンプレートマッチングは次式により実行できる．

$$\prod_{(m,\,n)\in \mathrm{temp}} [\mathrm{EE}]_{m,\,n}. \tag{1-16}$$

例として，５画素からなる横長のテンプレートを入力画像 A から検出するテンプレートマッチングを Fig.1.6 に示す．テンプレートが入力画像 B 上に配置されている．横長の５画素の領域に対して排他的論理和を行うと，テンプレートが存在する位置の左端の画素に画素値１が得られる．

以上の２方法のうち，前者は，入力画像を１枚しか使用せず，画像内の任意の位置にあるパターンを検出できる利点がある．しかし，テンプレートを指定する演算カーネルをあらかじめ作成しなければならないため，演算実行前にテンプレートがわかっている必要がある．後者は，２枚の入力画像を必要とするが，マッチングの対象となる領域を画像内で限定でき，領域ごとに異なるテンプレートを指定することができる．また，演

算実行前には演算カーネルからテンプレートのサイズのみを指定すればよいため，前者より柔軟な処理が行える．

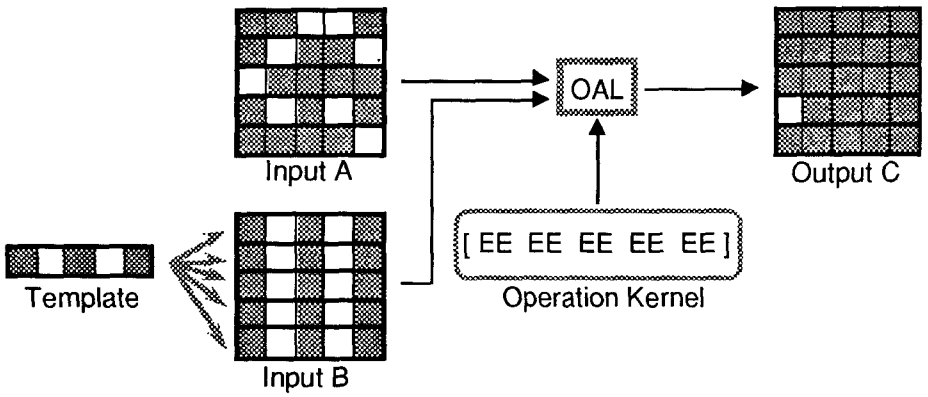


Fig.1.6　テンプレートを画像から指定するテンプレートマッチングの例

(c)　パターン展開

パターン展開 (pattern expansion) は，画素パターンをその近傍領域に複数組複写する技術である．これは，画像上の画素値1の画素から，特定のパターンを生成する技術ともみなせる．複写する位置の局所座標の集合を temp とすると，パターン展開のカーネル式は次のように書ける．

$$\sum_{(m,\,n)\in \mathrm{temp}} [1.]_{-m,\,-n} \qquad \text{(for Input A)}, \tag{1-17}$$

$$\sum_{(m,\,n)\in \mathrm{temp}} [.1]_{-m,\,-n} \qquad \text{(for Input B)}. \tag{1-18}$$

例として，入力画像 A 内の画素値 1 を持つ画素を 4 個右横に複写し，出力画像 C の対応する位置に横長の 5 画素のパターンを生成するパターン展開を Fig.1.7 に示す．

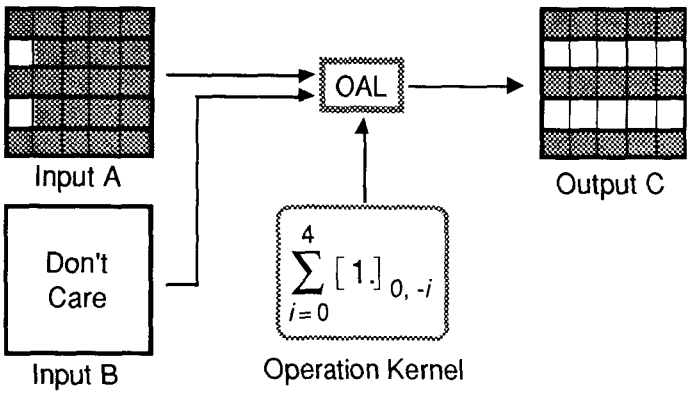


Fig.1.7　パターン展開の例

1.5.2　負論理

負論理とは，論理値0のときに画素値1，論理値1のときに画素値0とする論理である．光アレイロジックでは，特定の演算を負論理で実行し，その結果を反転して結果を得ることにより，演算を簡単化できる場合がある．この場合，目的とする演算のカーネル式の否定を用いて演算を行い，演算結果を反転させることにより正しい結果を得る．この方法を用いると，カーネル多項式の否定をとることにより積項数すなわち相関演算回数を減らすことができる．例えば，Fig.1.7 のパターン展開のカーネル式の否定をとると，ド・モルガンの定理より，

$$\overline{\sum_{i=0}^{4}[1.]_{0,-i}} = \prod_{i=0}^{4}[0.]_{0,-i}, \tag{1-19}$$

となる．そこで，この式を用いて演算を行い，演算結果を反転することによりパターン展開を実行できる．このとき，5回必要であった相関演算回数が1回になり，処理効率が向上する．このように，負論理はパターン展開において非常に有効である．

ただし，負論理による演算の簡単化が有効な場合は，カーネル式の各 PTB 内の記号が少なく，否定をとっても PTB があまり増加しない場合に限られる．

1.5.3　スペースバリアント演算とパターン論理

光アレイロジックは SIMD 方式の並列処理であるため，全画素に対するスペースインバリアント演算である．しかし，この演算形式が有効な場合は，2値画像処理などの全画素に対し同一命令を実行する処理に限られ，数値処理など画素ごとに処理が異なる場合には向いていない．そこで，光アレイロジックにおけるスペースバリアント演算が必要になる．

光アレイロジックでは2枚の入力面のうち，1枚をデータ用に，もう1枚をデータの属性用に用いることにより，スペースバリアント演算が実現できる．一つのデータとそのデータ属性を，二つの画像の対応する位置にそれぞれ配置し，このデータと属性の組に対して光アレイロジックの演算を実行する．すると，データの属性によって各データで異なった処理を行うことができる．このとき，データ用画像をデータプレーン，データの属性用画像をアトリビュートプレーンと呼ぶ．データ，データの属性を表す画素パターンはそれぞれデータパターン，アトリビュートパターンと呼ばれる．また，このように，処理物体をデータパターンとアトリビュートパターンの組で表すことにより，SIMD 方式の演算を用いて MIMD 方式の演算を実行する方法をパターン論理という [28].

スペースバリアント演算の例として，条件付きテンプレートマッチングを説明する．条件付きテンプレートマッチングとは，検索の対象となる画像と，検索すべき場所を指定する画像とを検査し，両方の画像上のパターンに適合した画素を検出する処理である．例えば Fig.1.8 に示すように，入力画像 B 上にあるパターンに対してテンプレートマッチングを行う場合，入力画像 A 上のパターンにより検索範囲を限定することができる．

この例では，入力画像Ａ上で左端の画素値が１である３行のパターンのある領域のみが検索対象となる．このとき，入力画像Ａがアトリビュートプレーン，入力画像Ｂがデータプレーンである．

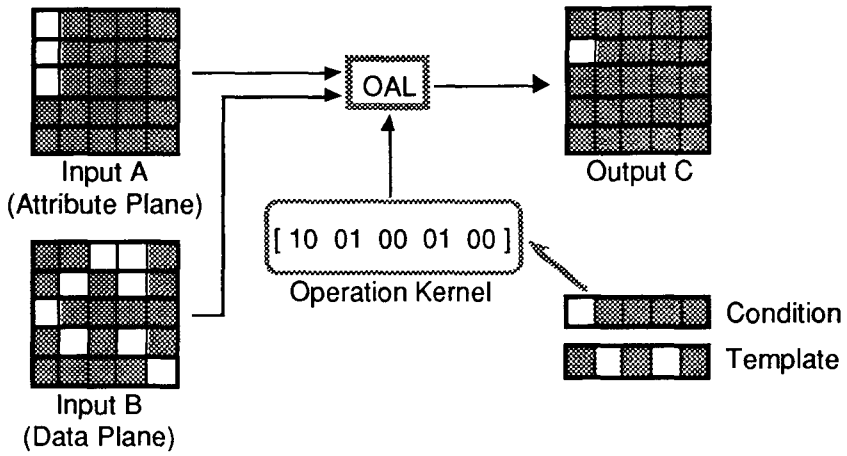


Fig.1.8　条件付きテンプレートマッチングの例

１．５．４　トークン伝播

本研究において，新しいプログラミング技法として，トークン伝播と呼ぶ方法を開発した．これは，トークンと呼ぶ，何らかの情報を表現する画素パターンを２次元画像上に複数個配置し，各トークンを独立に画像内で並列転送することにより，スペースバリアントな処理を実現するものである．この手法では，パターン展開とテンプレートマッチングを使用する．Fig.1.9 に，トークン伝播の手順を示す．まず，伝播させるトークンをパターン展開を用いて特定の領域に複写する．複写されたトークンのうち，条件画像と呼ばれる画像と一致したものをテンプレートマッチングにより選択する．結果として，最初のトークンを，与えられた条件を満たす場所へ転送することができる．この過程は，SIMD 方式の処理で実行できるため，光アレイロジックにおいて有効である．

トークン伝播では，パターンを展開する領域を制限することにより，トークンの転送

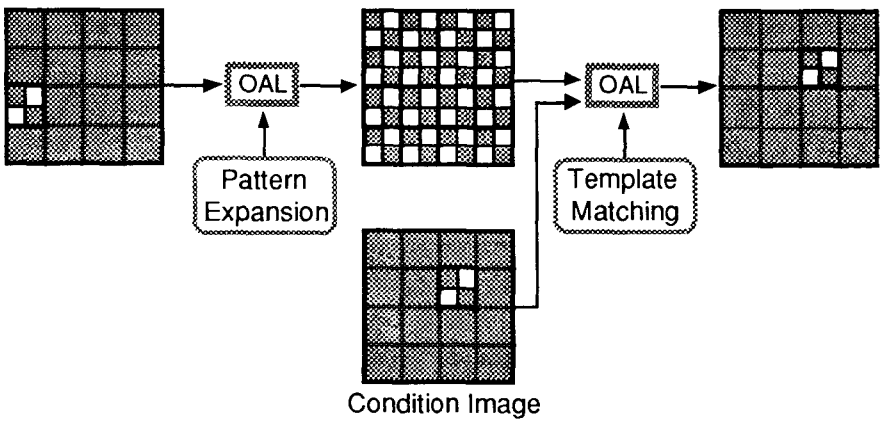


Fig.1.9　トークン伝播手順

先を制限できる．Fig.1.10 に示すように，トークンを横方向のみに展開すると，縦に配列した複数のトークンを独立に横方向に伝播させることができる．さらに，条件画像に適当なパターンをセットすると，各トークンの動きを独立に制御できる．また，それぞれ転送方向が異なる多重トークン伝播を数回行うことにより，各トークンを画像内の任意の位置へ転送することが可能になる．この技術により複数の演算過程を並列に制御できる．

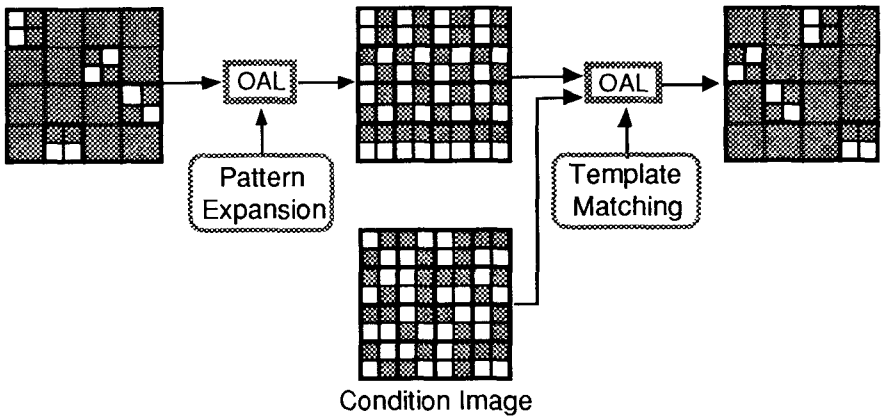


Fig.1.10　多重トークン伝播手順

1.6　光アレイロジックの機能の拡張

光アレイロジックをあらゆる処理に応用する場合，従来の光アレイロジック演算のみでは実行不可能な処理がある．本研究では，それらの処理を可能にするために，状態変数と，画像／カーネル変換と呼ぶ新機能を光アレイロジックに導入した．以下にそれらの新機能について述べる．

1.6.1　状態変数

状態変数は，演算結果画像の状態を示す変数であり，マイクロプロセッサの状態フラグに相当する．これは，光アレイロジックが苦手とする，ある領域内で画素値1の画素数を数える処理を補助する役目をする．状態変数の種類には，Number と Zero がある．状態変数 Number は，画像中の画素値1の画素数を返す．状態変数 Zero は，画像中の全ての画素の値が0ならば1を，そうでなければ0を返す．状態変数を用いることにより，演算結果の画像状態を容易に参照でき，効率のよい処理を実行できる．例えば，ある演算により全画素値が0である画像が得られたら処理を終了するようにプログラムを作成しておくと，状態変数 Zero を参照することにより処理終了の判定ができる．状態変数は，光アレイロジックの演算により求めることはできるが，6.3節で述べるように，簡単な専用光学系で効率よく求められる．

状態変数の導入に伴い，OALL 文法に次の項目を付加する．

・状態変数

状態変数は，光アレイロジックの演算結果の状態を表す変数で，マイクロプロセッサ

における状態フラグに相当する．状態変数は，次の形式で記述する．

```
condition ( <状態変数名> );
```

状態変数名：Number, Zero

状態変数は，最後に実行されたexec文の結果画像の状態を，整数定数の形でcondition文に返す．状態変数 Number は，画像中の画素値1の画素数を返す．状態変数 Zero は，画像中の全ての画素の値が0ならば1を，そうでなければ0を返す．

1.6.2　画像／カーネル変換

画像／カーネル変換は，画素パターンを演算カーネルの格子点パターンに変換するものであり，本研究で新たに追加した機能である．画像／カーネル変換は，画像と演算カーネルという，二つの異なるデータ形式間の変換を可能にするものである．これにより，データをプログラムとして使用することが可能になる．この機能は，プログラム実行前に演算カーネルを決定することができない場合に有効である．その例としては，演算カーネルでテンプレートを指定するテンプレートマッチング（1.5.1項）を実行したいが，プログラム実行前にテンプレートのパターンを決定できない場合（2.3.3項）などが考えられる．

画像／カーネル変換の導入に伴い，以下の項目を OALL 文法に追加する．

・型変換文

カーネル定数は光アレイロジックの処理手順実行時に Table 1.1 に基づいて格子パターンに変換される．したがって，画像定数とカーネル定数は，どちらも2値の2次元配列パターンで構成されると考えられる．そこで，この2定数間の型変換を許す．型変換文は，次の形式で記述する．

```
<kernel型変数> = kernel (<image型変数1>, <image型変数2>);
```

<image型変数1> が指定する画像定数の画素パターンをそのまま演算カーネルのパターンに変換したカーネル定数を生成し，<kernel型変数> に代入する．原点のカーネルユニットの左上に相当する画素の位置は，<image型変数2> 上の対応画素に画素値1をセットすることにより指定する．ただし，<image型変数2> 上に画素値1の画素が二つ以上存在する場合は，正しい変換は保証されない．

1.7　結言

本章では，本研究における並列演算パラダイムである光アレイロジックによる並列プログラミングについて概説した．光アレイロジックの概念，処理手順，光学的実現方法を述べ，プログラム記述法と専用言語である OALL について概説した．さらに，光ア

レイロジックの基本的な並列プログラミング技法を，本研究で新たに開発したものを含めて述べた．また，光アレイロジックでより柔軟な処理を行うために，本研究で新たに拡張した光アレイロジックの機能を述べた．本節にまとめたプログラミング技法は，本研究で開発した並列演算技法の基本となるものである．ここで示したような，光アレイロジックの並列プログラミング能力と記述言語の充実度は，他の並列光演算原理には見られない．したがって，光アレイロジックは光演算に限らず，未だ確立されていない並列演算技法の研究にも有用であると考えられる．

第2章　光アレイロジックによる推論機構

2.1　緒言

人工知能 (AI: Artificial Intelligence)は人間の知的行為を代行するコンピュータシステムである [39, 40]. AIでは推論が主な処理となる. 推論は, 知識ベースから特定のデータを探索する処理であり, 通常大容量データを扱わなければならない. エキスパートシステム [41] のような, 典型的なAIシステムでは, この推論処理を司る推論機構がシステム全体の性能を決定する. したがって, AI問題においては, 大容量データに対する効率のよい探索法の開発が非常に重要である. これには並列処理が有効であり, 並列処理マシンによる高速探索法の研究が活発に行われている [10, 42].

最近, 光の並列性を有効利用したデータ探索の手法がいくつか報告されている [43-46]. これらの方法では, 知識データが画像で表され, 推論が行列演算 [44] やマッチド・フィルタリング [45] などの, SIMD方式の並列処理で実行される. これらの処理は, 光システムにより効率よく高いスループットで実行できる. しかしながら, これらの方法は柔軟性に欠けるため, データに対する詳細な命令を実現できない. この欠点を解消するため, 知識データを光電子複合型マルチプロセッサ・システムに直接割り付ける方法が提案されている [46]. この方法では, 推論は光インターコネクションによる処理要素間のマーカ伝播により実行できる. しかしながら, この方法ではマーカ伝播以外の処理に光の並列性が利用されていない.

本研究では, 光アレイロジックの並列性を知識データの探索に有効利用する並列推論機構の実現方法を2種類考案した. さらに, 光アレイロジックによる処理の柔軟性をいかして, 推論機構の機能を拡張することにより, 簡単な並列エキスパートシステムを実現する方法も考案した. 以下, 2.2節では, 推論機構と知識の表現法, エキスパートシステムについて概説する. 2.3, 2.4節では, 光アレイロジックによる推論機構の実現方法を2種類述べる. 2.5節では, 本推論機構の機能を拡張して, エキスパートシステムの処理を実現する方法を述べる. さらに, 2.6節では, これらの手法の処理効率を評価する.

2.2　推論機構

2.2.1　推論機構とは

エキスパートシステムなどの典型的なAIシステムにおいては, 推論とは, さまざまな知識を表現するデータの集まり（知識ベース）を用いて, 与えられた質問に答える処理のことである [39, 40]. 推論機構は, 知識ベースから質問に関係するデータを探索することにより, 質問に対する答を発見するメカニズムである. 人間に近い知的な推論を行うためには, 知識ベースには多量のデータを格納しておかなければならない. したがって, 推論機構では, データの探索を高速に効率よく実行しなければならない.

2.2.2　知識の表現法

AIの研究では，数種の知識表現法が提案されている．例えば，プロダクション・ルール，フレーム，意味ネットワークなどがある[39, 40]．ここでは，光アレイロジックによるSIMD方式の処理に適した知識表現として，意味ネットワークを用いる[43]．意味ネットワークは，ノードとリンクからなる有向グラフで記述される図式的な知識表現である（Fig. 2.1）．ノードは概念を表し，リンクは二つの概念間の関係を表す．意味ネットワークを用いた推論は，あるノードから，それとつながっているノードへリンクをたどることにより実行できる．例えば，“猫は何の一種か”という質問が与えられると，‘猫’ノードから‘動物’ノードへと‘ISA’リンクをたどっていくことにより，答‘動物’が得られる．

意味ネットワークでは，リンクをたどる処理にSIMD方式の並列性が存在する．例えば，“ひげを持つ動物は何か”という質問が与えられた場合は，‘ひげ’ノードから‘HAS’リンクを逆方向にたどると同時に，‘動物’ノードから‘ISA’リンクを逆方向にたどる．両方が行き着いたノード‘猫’が答となる．このように，複雑な質問になるほど，リンクをたどる処理に並列性が出てくる．したがって，エキスパートシステムなどで使用する，膨大な意味ネットワークに対して効率よくリンクをたどる手法に，光アレイロジックの並列性が有効に利用できる．

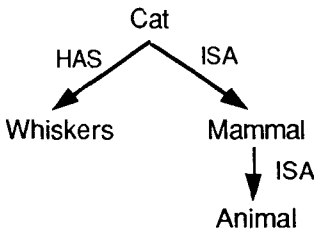


Fig.2.1　意味ネットワーク

2.2.3　エキスパートシステム

エキスパートシステムは，専門家の知識を形式化して知識ベースに蓄えておき，非専門家でもそれらの知識を効率よく利用できるようにしたシステムである．エキスパートシステムでは，知識ベースと推論機構が基本要素となり，人とシステムとの間で質問・回答の対話を行う．エキスパートシステムが専門家により近い回答を行うためには，通常多くの知識データが必要となる．エキスパートシステムの実用化には，膨大な知識ベースに対する効率的な推論機構の処理方法の開発が課題となっている．

本章では，光アレイロジックによる並列エキスパートシステムの実現方法を検討する．エキスパートシステムでは，質問に対して柔軟に対応する必要があるため，推論機構にいくつかの機能を追加することが要求される．

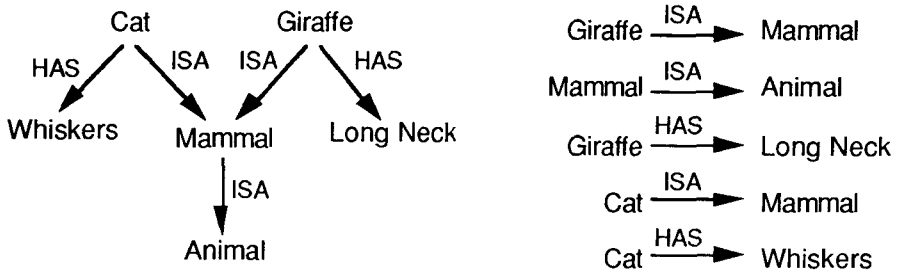
2.3　テンプレートマッチング法

2.3.1　概要

意味ネットワークを知識ベースに格納する場合，Fig.2.2のように，意味ネットワー

クを，１本のリンクからなる最小の意味ネットワークの集合に分割する方法がよく用いられる．このように分割すると，意味ネットワーク全体の形にかかわらず一定のフォーマットでメモリ上に格納できる利点がある．しかし，この方法では，推論機構において一回リンクをたどるごとに，次に続くノードとリンクを知識ベースの全データ中から探索するため，効率のよいデータ探索法が必要である．

テンプレートマッチング法は，分割した意味ネットワークの集合を１枚の画像で表し，光アレイロジックによる並列テンプレートマッチングを用いて，効率のよい探索を行う方法である．



(a) 意味ネットワーク例　　(b) 分割された意味ネットワーク

Fig.2.2　意味ネットワークの分割

2.3.2　知識ベースの符号化法

意味ネットワークの画素パターンへの変換は次の手順で行う．

1）知識ベースの意味ネットワークを，Fig.2.2(b) のように，２ノードとその間を結ぶリンクからなる最小の意味ネットワーク（サブネットワーク）の集合に分割する．

2）各ノード，リンクをそれぞれ画素パターンに変換する．

3）サブネットワークを表す２ノードと１リンクの画素パターンを１行に並べてグループ化し，それらを縦方向に並べていく．

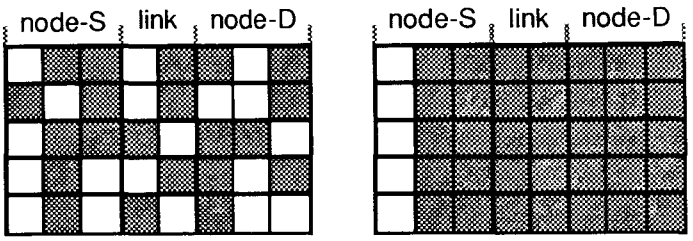
4）これらの画素パターンを２値画像上に配置し，知識ベースのデータプレーンと呼ぶ．

5）データプレーン上の１サブネットワークを表す画素パターンの左端の画素の位置に画素値１をセットした画像を作成し，これを知識ベースのアトリビュートプレーンと呼ぶ（1.5.3項）．

以上の手順により，知識ベースの意味ネットワークは Fig.2.3(a), (b) の２枚の画像で表される．なお，この図で node-S (Source), node-D (Destination) は，それぞれリンクの始端，終端にあるノードを表す．

2.3.3　推論機構の実現方法

知識ベースを Fig.2.3 のように符号化すると，推論は条件つきテンプレートマッチングで実行できる．Fig.2.4 は，符号化された知識ベースを用いた推論の処理の流れを示す．質問は Fig.2.4 に示すように，サブネットワークの終端ノードがない形をしている



(a)　データプレーン　　(b)　アトリビュートプレーン

Fig.2.3　知識ベースの符号化

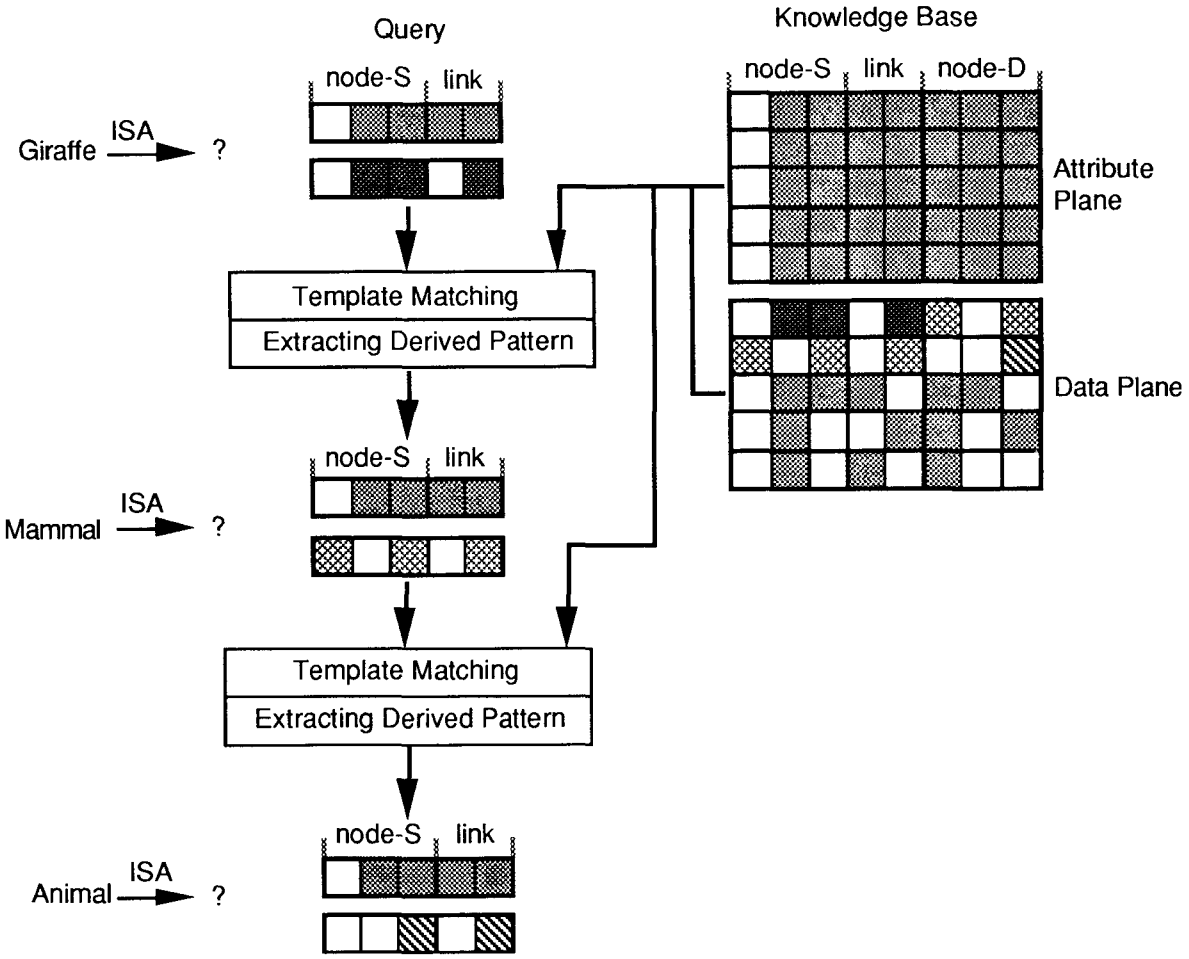


Fig.2.4　テンプレートマッチング法による推論の流れ

ため，終端ノードに対応する画素パターンのみ除いて，知識ベースと同様に符号化する．推論の手順は次のようになる．

1）与えられた質問の符号化パターンをテンプレートとして，知識ベースに対する条件つきテンプレートマッチング（1.5.3項）を行う．

2）マッチングに成功したパターンの右隣にあるノード（質問により導出されたノード）のパターンを出力する．

３）出力したパターンの右隣にリンクのパターンを並べ，次のステップの推論での質問とし，上記処理を繰り返す．

光アレイロジックによる演算技術を用いると，テンプレートマッチングは，１．５．１項で述べたように，テンプレートで設定されたパターンを検出する近傍画素間演算として全画素に対して並列に実行できる．近傍画素間演算を行うためには，演算内容，すなわち特定のテンプレートが演算カーネルの形で表されていなければならない．しかし，上記の処理手順では，推論１ステップが終了するまでに次の推論ステップで用いるテンプレートを決定することができない．したがって，全推論過程で用いられる全てのテンプレートを推論開始前に用意することは不可能である．この問題を避けるため，本手法では，画素パターンを演算カーネルのパターンに変換する機能である，画像／カーネル変換（１．６．２項）を用いる．この機能により，推論におけるテンプレートマッチングを Fig.2.5 に示す手順で実行できる．すなわち，推論１ステップが終了し，次の推論ステップで用いるテンプレートが決定したら，その都度，Fig.2.5 の手順によりテンプレートを検出する演算カーネルを作成する．その演算カーネルで光アレイロジック処理を行うことにより，テンプレートマッチングによる推論を続行することができる．

光アレイロジックによる推論のシミュレーション結果を Fig.2.6 に示す．入力画像は，Fig.2.6(a), (b) に示すように，符号化された知識ベースと質問である．各画像の上部に画像名を示す．まず，与えられた質問をテンプレートとするテンプレートマッチングを行うために，その演算カーネルのパターンを作成する．この過程は光アレイロジックの演

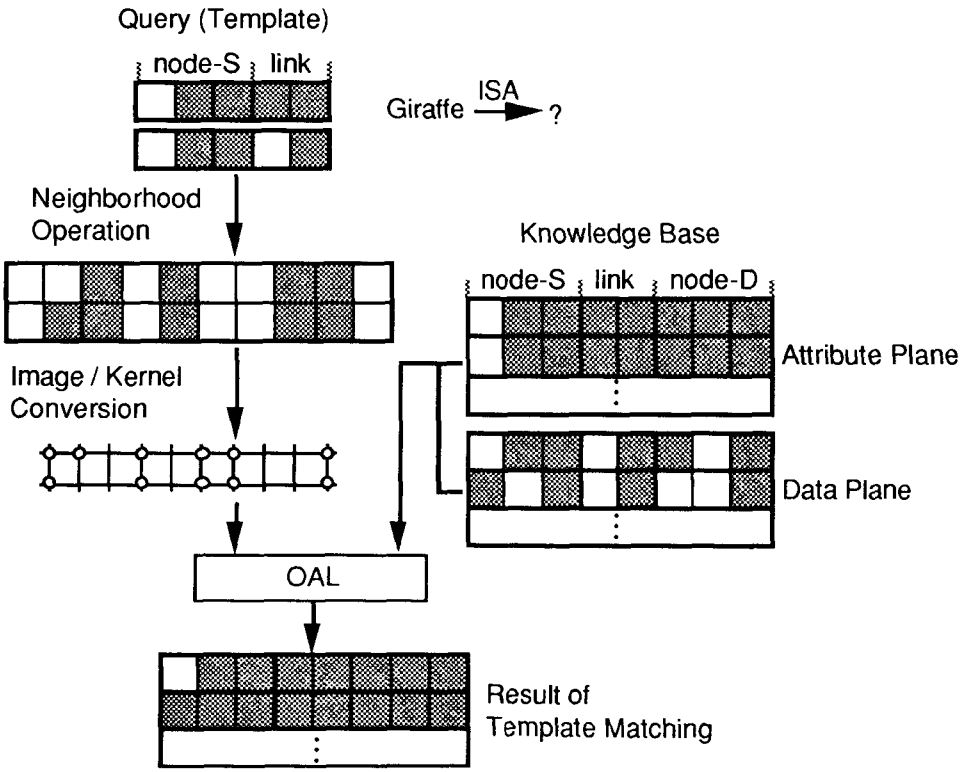
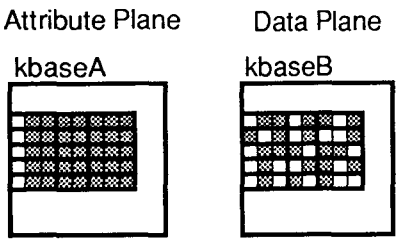
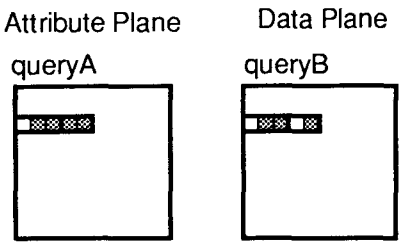


Fig.2.5　画像／カーネル変換を用いた条件つきテンプレートマッチング

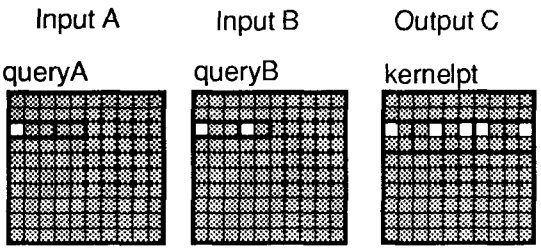
算3ステップで実行できる（Fig.2.6(c) - (e)）．作成した画素パターンから画像／カーネル変換を用いて演算カーネルを作成し，その演算カーネルを用いてテンプレートマッチングを実行する（Fig.2.6(f)）．その結果，マッチングに成功したパターンの右隣のパターン（質問により導出されたノードのパターン）が出力される．次に出力パターンの右隣にリンクのパターンを配置し，次の推論ステップの質問を作成する（Fig.2.6(g)）．1本のリンクをたどる操作である推論1ステップは，上記のように光アレイロジックによる5回の演算ステップと1回の画像／カーネル変換が必要である．最後のステップで出力された画素パターンを次の推論ステップの質問として，上記処理を繰り返すことによ



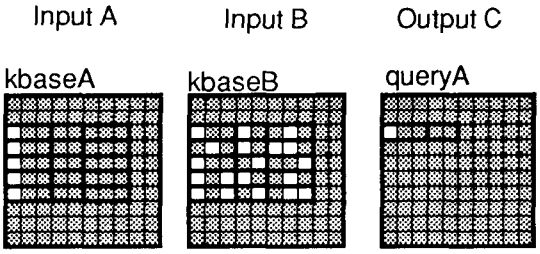
(a)　知識ベース画像



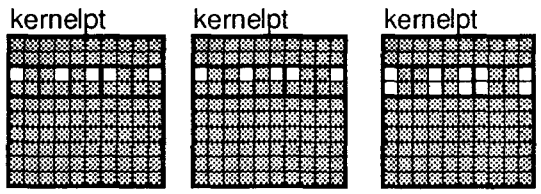
(b)　質問画像



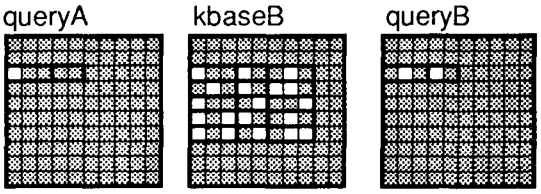
(c)　画素パターンから演算カーネルのパターンへの変換(1)



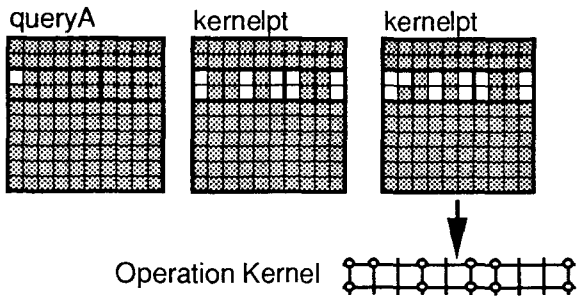
(f)　(e)で作成された演算カーネルによるテンプレートマッチング



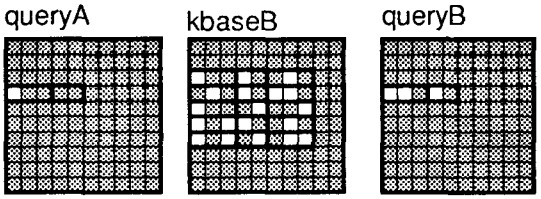
(d)　画素パターンから演算カーネルのパターンへの変換(2)



(g)　推論1ステップで導出されたパターンの出力



(e)　画素パターンから演算カーネルのパターンへの変換(3)



(h)　推論2ステップで導出されたパターンの出力

Fig.2.6　テンプレートマッチング法による推論のシミュレーション結果

り，Fig.2.6(h) のように推論２ステップ目の結果が得られる．本処理のカーネル式は Appendix A．1 に，OALL プログラムは Appendix B．1 に示す．

2.4　トークン伝播法

2.4.1　概要

意味ネットワークにおけるリンクをたどる過程は，マーカ伝播 [43] と呼ばれる方法でも実行できる．マーカは，推論処理を補助するためにリンク上を伝播する信号である．意味ネットワークを用いた推論の並列性は，主にマーカ伝播アルゴリズムにより得られる．例えば，Fig.2.7(a) のような意味ネットワークが知識ベースに蓄えられている場合に，“何という動物がひげを持つか”という質問が Fig.2.7(b) に示すように与えられると，二つのマーカ（‘HAS’リンクに沿って逆方向に伝播するマーカ#1と，‘ISA’リンクに沿って逆方向に伝播するマーカ#2）が用意され，それぞれ‘ひげ’と‘動物’のノードに配置される．これらのマーカは，同時に指定されたリンクに沿って伝播する．両方のマーカが到達したノードが答‘猫’を示す（Fig.2.7(c)）．

マーカ伝播を用いた推論機構の処理は次のようになる．

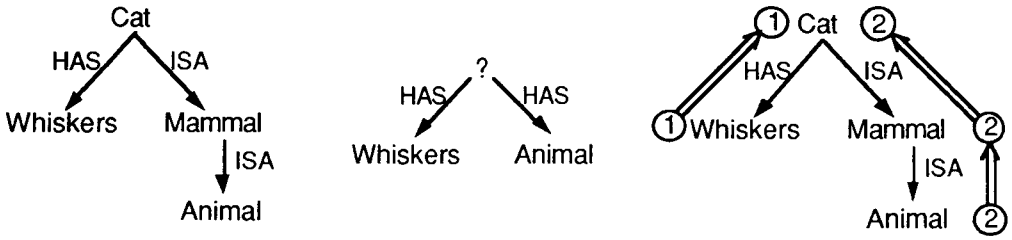
１）与えられた質問に含まれているノードを探索する．

２）探索したノードにマーカを配置する．

３）特定のリンクに沿って，接続されているノードへマーカを伝播させる．

４）全てのマーカが伝播経路の終端に来るまでステップ１）～３）を繰り返す．

上記過程は SIMD 方式の並列処理によりすべてのマーカを並列に処理できる．本節の方法では，マーカ伝播に光アレイロジックのトークン伝播技術を用いる．リンクの方向とタイプを１枚の画像上に画素パターンとして符号化することにより，光アレイロジックの並列処理能力を推論処理に有効利用できる．



(a)　意味ネットワーク　　(b)　質問　　(c)　マーカ伝播による推論

Fig.2.7　意味ネットワークとその推論

2.4.2　知識ベースの符号化法

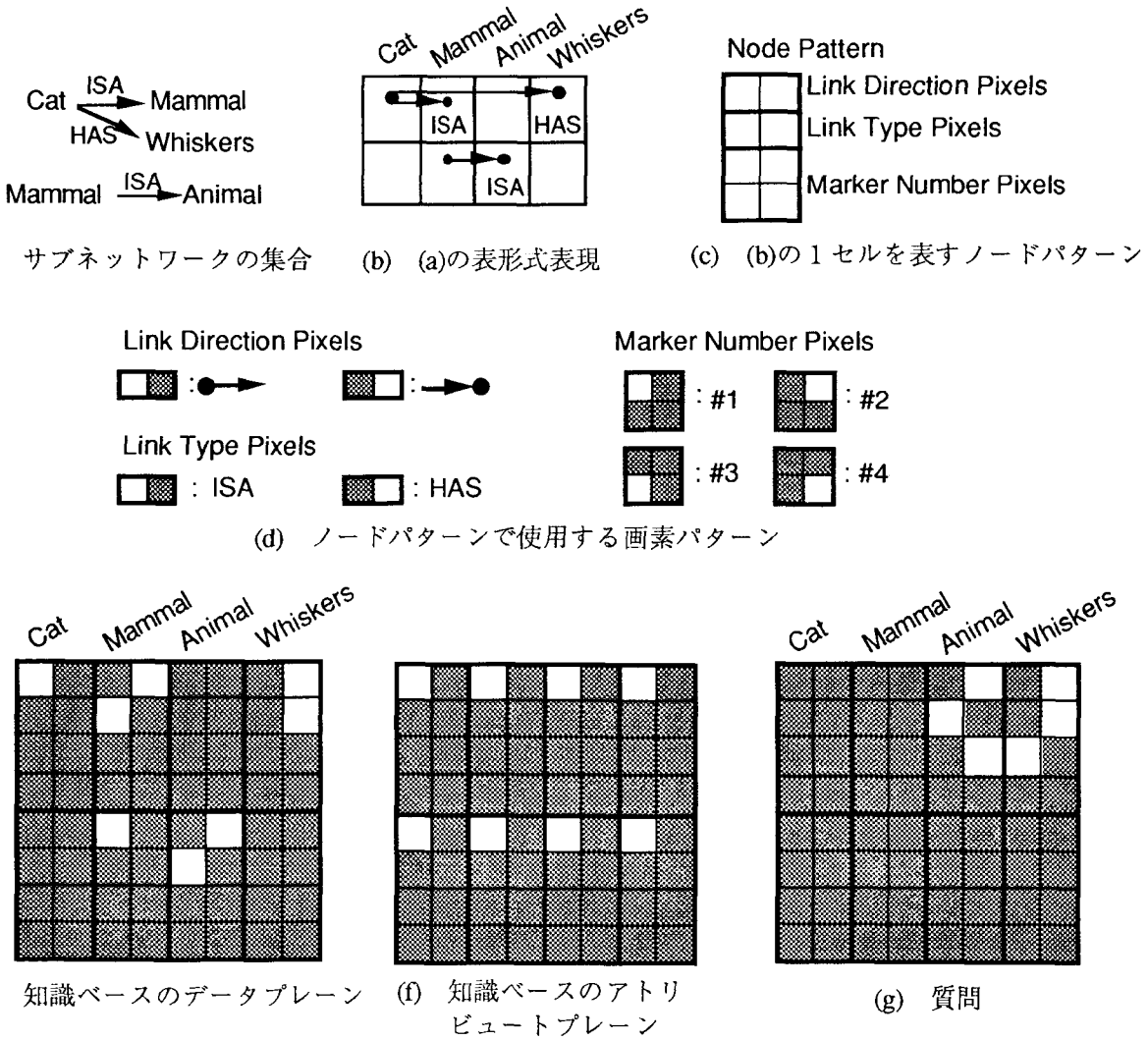
光アレイロジックにより推論機構を実現するためには，意味ネットワークを画像上の画素パターンとして表現しなければならない．そこで，Fig.2.7(a) の意味ネットワークを，Fig. 2.8 に示すように，以下の手順により画素パターンに変換する．

まず，知識ベースの意味ネットワークを，１対多のノード間接続形式を持つサブネッ

トワークに変換する．サブネットワークでは，一つのノードが異なるリンクを通じて複数ノードに接続されることになる（Fig.2.8(a)）．サブネットワークの集合は，表形式で符号化する．表の各行，列にそれぞれサブネットワークとノードを一つずつ割り付ける（Fig.2.8(b)）．

この表形式では，リンクの方向と‘ISA’などのタイプをそれぞれのセル（表の１区画）に格納する．リンクタイプは，リンクの行先のセルにセットする．これらを，ノードパターンと呼ぶ空間的パターンとして符号化し，各セルに配置する（Fig.2.8(c)）．推論でマーカをセットする領域もこのセル内に用意する．Fig.2.8(d) に示すように，それぞれの意味に対して符号化パターンを割り当てる．リンク方向画素の表現では，点がノードを表す．リンクタイプの符号化パターンは，この例では２種類であるが，任意のリンクタイプを割り当てることができる．符号化されたパターンを，Fig.2.8(b) の各ノードに対して配置することにより，意味ネットワークを画像で表現できる（Fig.2.8(e)）．

さらに，それぞれのセルの位置を識別するために，セルの位置の目印となる画素パターンを配置した画像を用意する（Fig.2.8(f)）．この技術は，処理効率を上げるために光



(a)　サブネットワークの集合　(b)　(a)の表形式表現　(c)　(b)の１セルを表すノードパターン

(d)　ノードパターンで使用する画素パターン

(e)　知識ベースのデータプレーン　(f)　知識ベースのアトリビュートプレーン　(g)　質問

Fig.2.8　意味ネットワークで表現された知識ベースの符号化

アレイロジックでしばしば使われる．これらの，目印用の画像と意味ネットワークを表す画像を，それぞれ知識ベースのアトリビュートプレーン，データプレーンと呼ぶ（1.5.3節）．両方の画像に対するテンプレートマッチングにより，知識ベースのアクセスを並列に行うことができる．

質問は，Fig.2.7(b) に示すように，意味ネットワークのサブセットで表される．これは，知識ベースと同一の方法で符号化することができる．すなわち，質問を表形式で表し，空間的パターンに変換する．さらに，質問の各セルにマーカを割り当てる．すなわち，マーカ#1と#2を，セルのマーカ番号画素に配置する（Fig.2.8(g)）．

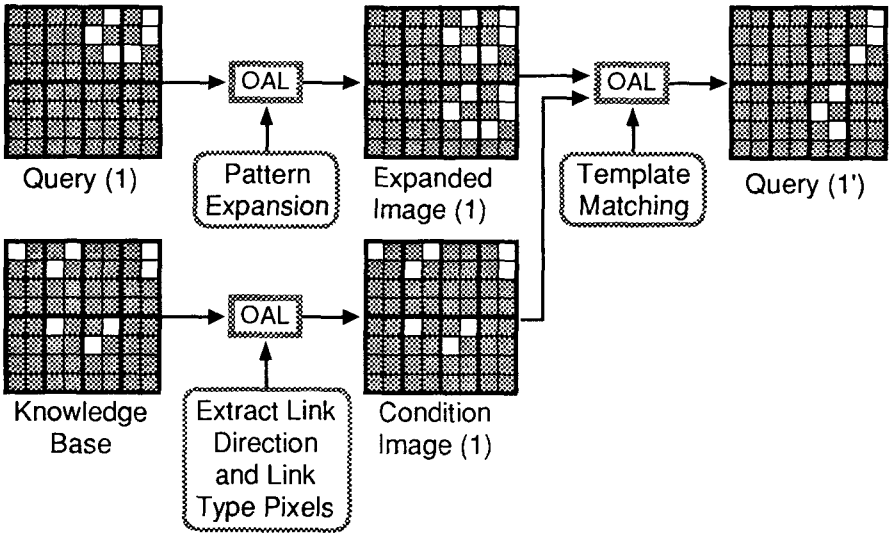
2.4.3　推論機構の実現方法

知識ベースを空間的に符号化すると，推論処理は光アレイロジックによるトークン伝播技術（1.5.4項）を用いて効率よく実行できる．例として，Fig.2.7(b) の問題，すなわち“何という動物がひげを持つか”を光アレイロジックにより解く場合を考える．2.5.1で述べたように，‘ひげ’と‘動物’ノードから伝播するマーカがこの問題で用いられる．本手法では，マーカを画像上のトークンに割り当て，光アレイロジックのトークン伝播技術により処理する．

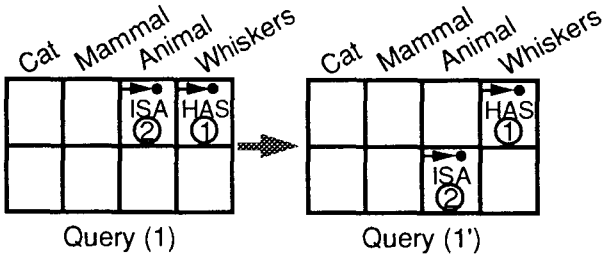
推論を行う基本的な手順は次のようになる．

1）表形式で表現した知識ベースの各行に割り当てたサブネットワークの中から，与えられた質問に対応するものを探索する．
2）探索したサブネットワークのセルにトークンをセットする．
3）トークンを同一行内で特定のリンクに沿って逆方向に伝播させる．
4）ステップ1）～3）を繰り返し，全てのトークンが同一ノード，すなわち同一の列に到達したら，処理を終了する．

Fig.2.9 - 2.11 は，トークン伝播法における主要な処理手順を示す．ステップ1）と2）は，Fig.2.9 に示す縦方向トークン伝播で実行できる．ここでは，知識ベースから‘ISA’と‘HAS’リンクの終端ノードを選択するために，条件画像 Condition Image (1) を作る．この場合，トークンとして4×2画素からなるパターンを1セルに割り当てる．各セルの領域を識別するために，知識ベースのアトリビュートプレーン（Fig.2.8(f)）を使う．ステップ3）の準備として，Fig.2.10 のようにトークン上のリンク方向画素を変更し，転送先となるリンクの始端ノードを探索するために用いる．変更したトークンを横方向に伝播し，Fig.2.11 のようにステップ3）を実行する．転送したトークンは，再びリンクの終端ノードを転送先とするように変更する．得られたトークンは，質問と同一のフォーマットを持つため，上記処理は，Fig.2.12 のように最終結果が得られるまで繰り返すことができる．本手法のカーネル式を Appendix A.2 に，OALL プログラムを Appendix B.2 に示す．

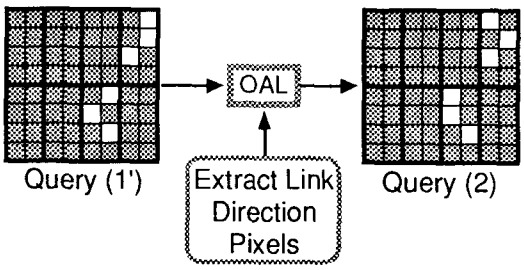


(a)　処理画像の流れ

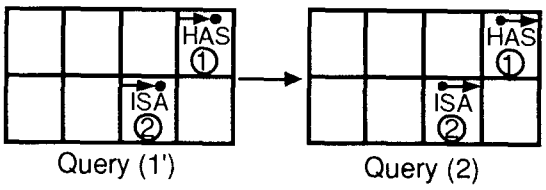


(b)　質問の表形式表現

Fig.2.9　縦方向トークン伝播の処理手順

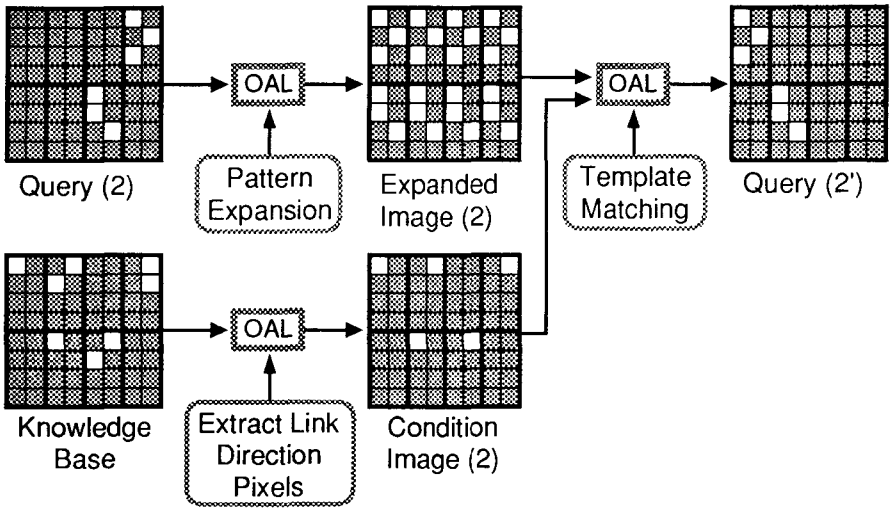


(a)　処理画像の流れ

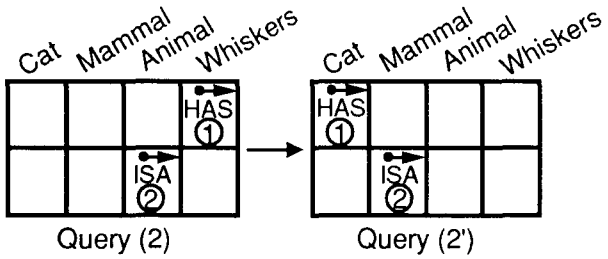


(b)　表形式表現

Fig.2.10　トークンのリンク方向画素の変更

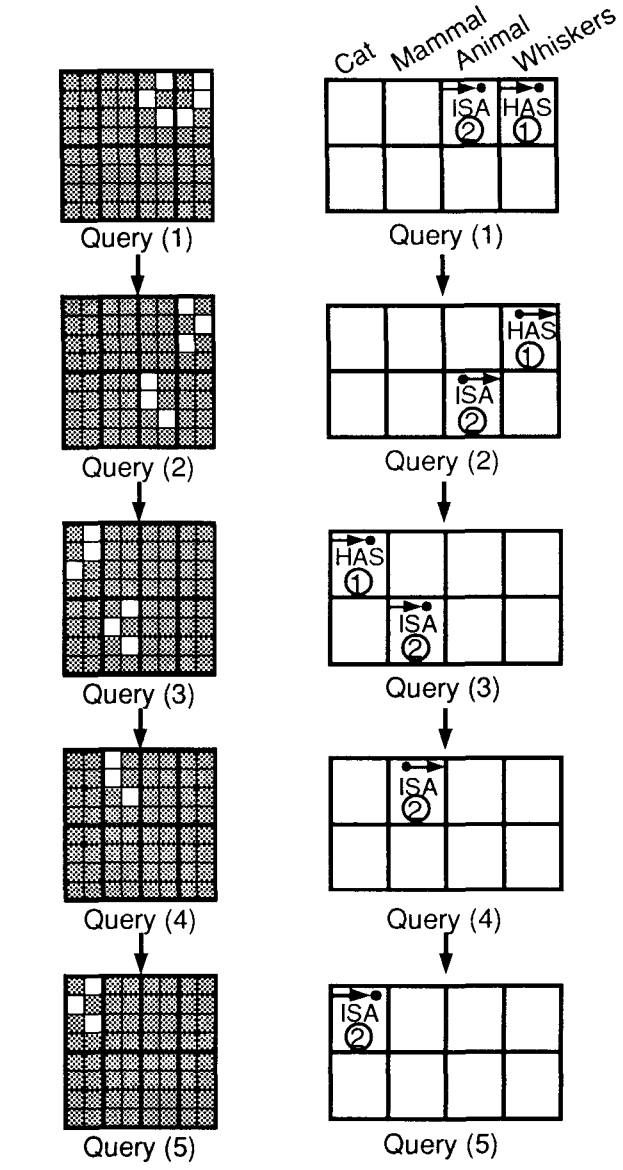


(a)　処理画像の流れ



(b)　質問の表形式表現

Fig.2.11　横方向トークン伝播の処理手順



(a)　処理画像の流れ　　(b)　(a)の表形式表現

Fig.2.12　質問画像上でのトークン伝播の様子

2.5　エキスパートシステム

2.5.1　推論機構の拡張

エキスパートシステムは，特定領域の専門家の知識を知識ベースに格納し，非専門家にその知識を効率よく提供するシステムである．本節では，トークン伝播法による推論機構を，エキスパートシステムに拡張する方法について述べる．

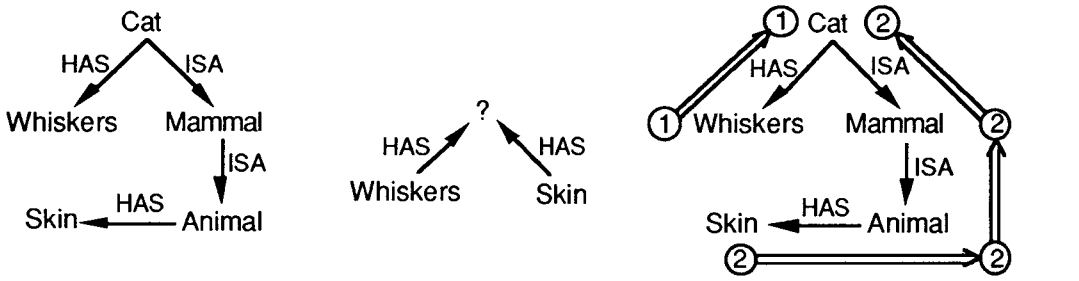
基本的な推論機構をエキスパートシステムに拡張するためには，次の三つの機能，1）属性の継承，2）伝播したマーカの軌跡の検出，3）マーカ伝播終了の検出，が付加される必要がある．

まず，機能1をFig.2.13の例で説明する．今，Fig.2.13(a)に示す意味ネットワークに，“ひげと皮膚を持つものは何か”という質問（Fig.2.13(b)）がされた場合を考える．意味ネットワークで示されているように，‘猫’は‘動物’に含まれる．‘動物’の特徴

は，その中に含まれる‘猫’に受け継がれるため，属性“皮膚を持つ”は‘動物’から‘猫’ノードまで継承されるべきである．この機能は，マーカを特定のリンク（この例では‘HAS’リンク）と‘ISA’リンクの両方に沿ってマーカを伝播させることにより達成される．

機能２を実行するためには，マーカを伝播した全てのノード上に記録する．伝播後，記録された該当するマーカの組を検出することにより，特定の条件を満たすノードを検出することができる．Fig.2.13 の例では，マーカ#1と#2の組を検出することにより，‘猫’ノードが答として得られる．

所望のマーカを持つノードを検出する前に，マーカ伝播の処理を終了しなければならない（機能３）．この機能は，どのマーカも伝播しなくなったことを検出することにより達成できる．これらの機能は，全てのデータに対して一度に実行できるため，光アレイロジックの並列性を有効に利用できる．



(a)　意味ネットワーク　　(b)　質問　　(c)　マーカ伝播を用いた属性の継承

Fig.2.13　意味ネットワークによる属性の継承

2.5.2　エキスパートシステムの実現方法

エキスパートシステムにおける推論の処理手順を Fig. 2.14 に示す．処理の主要部は，2.4 で述べたトークン伝播による推論機構の処理に等しい．そこで，ここではエキスパートシステムのために付加した機能について説明する．

手順を簡単化するため，属性の継承はリンクを逆方向にたどる場合のみ行われるものとする．この場合，質問が示す各ノードにセットしたトークンが，‘ISA’リンク以外の全てのリンクを逆方向にたどり，その後伝播した各トークンが‘ISA’リンクを逆方向にたどる．結果として，属性の継承は“皮膚を持つものは何か”のような質問に対する推論に有効である．ここでは簡単のためにエキスパートシステムの機能を制限したが，光アレイロジックではさらに複雑なプログラムを作成することによって，この機能を拡張できる．

伝播したマーカの軌跡は，質問のマーカ番号画素を知識ベースの対応する画素に複写することにより記録できる．Fig.2.15 に，マーカ記録前と記録後の知識ベースの例を示す．この処理は，論理演算で簡単に実行できる．

どのマーカも伝播しなくなったかどうかを検出するためには，光アレイロジックの拡張機能である状態変数（1.6.1 項）を用いる．同様の機能は，光アレイロジックによ

る論理演算で実現可能だが，状態変数を用いると，専用ハードウェアで処理することにより，処理効率の大幅な向上が期待できる．結果として，終了状態のテストは，状態変数 Zero を参照することにより達成される．

所望のマーカを持つノードの検出は，テンプレートマッチングで実行できる．Fig.2.15 では，各セルの３行目のマーカ番号画素を参照することにより，マーカ#1と#2を両方含むセルが検出される．結果として，左上のセルが検出され，‘猫’が答として導出される．上記の手順により，Fig.2.16(a) に示す知識ベースを用いて，簡単なエキスパートシステムの実現例を示す．エキスパートシステムに，Fig.2.16(b) の“毛が短く，しっぽのない猫は何か”という質問がされた場合を考える．Fig.2.17 は，エキスパートシステムのシミュレーション結果である．Fig.2.17(a) と (b)は，知識ベースのアトリビュートプレーンとデータプレーン，Fig.2.17(c) は質問の画像を示す．Fig.2.17(d) は，マーカ伝播処理の後，マーカが記録された知識ベースのデータプレーンである．

Fig.2.17(e) は，全てのマーカが記録されたノードの検出結果である．このノードが質問の答を示す．画像の端にある縦，横方向の目盛は，それぞれサブネットワークの境界と各ノードを表す．横方向の目盛の数字は，Fig.2.16 のノードの識別番号と対応している．以上により，最終結果として‘マンクス’が得られる．なお，本手法の OALL プログラムを Appendix B.3 に示す．

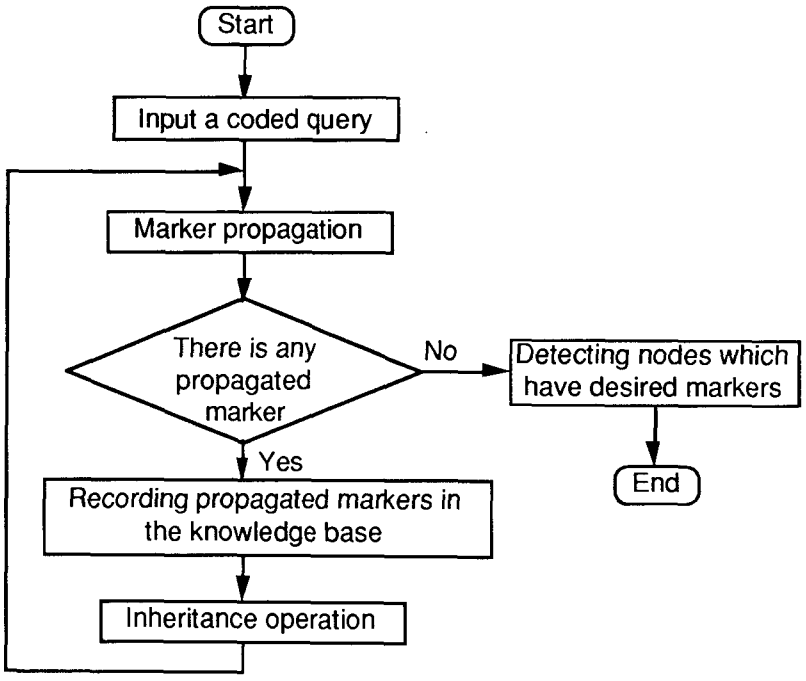
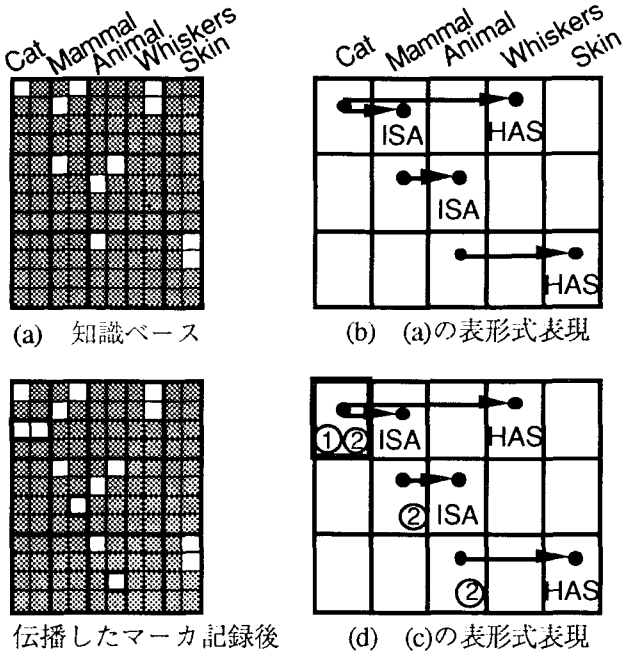


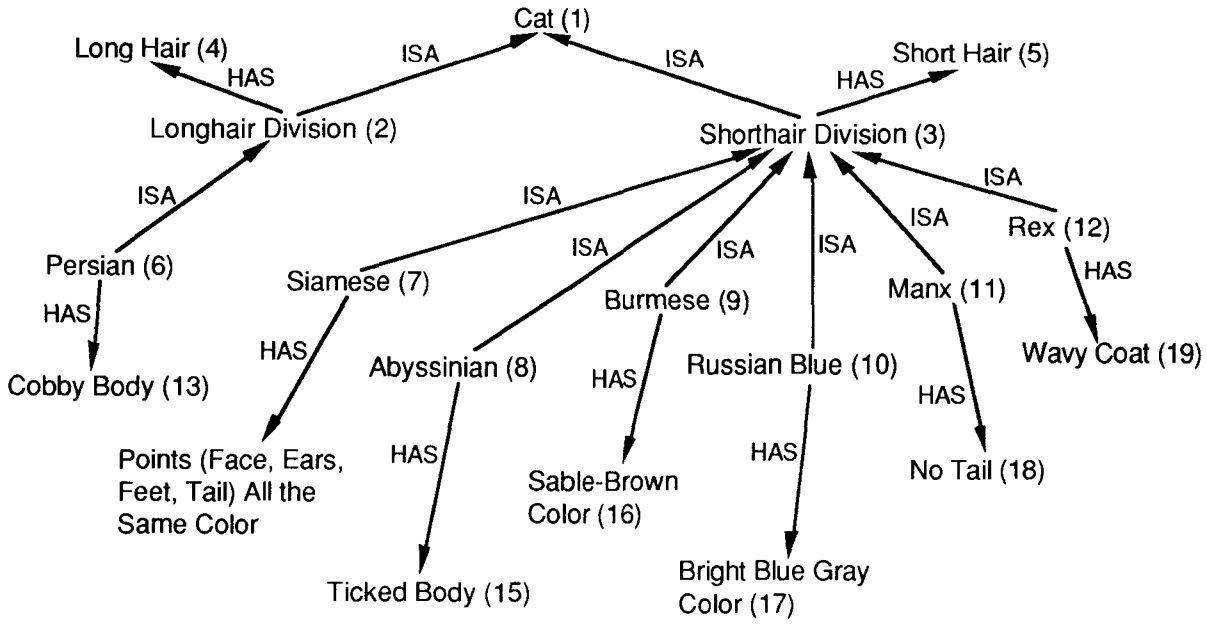
Fig.2.14　マーカ伝播を用いたエキスパートシステムの処理の流れ



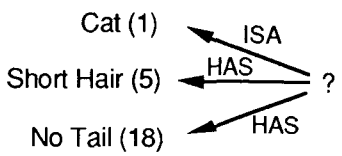
(a)　知識ベース　　(b)　(a)の表形式表現

(c)　伝播したマーカ記録後の知識ベース　　(d)　(c)の表形式表現

Fig.2.15　伝播したマーカの知識ベース上への記録

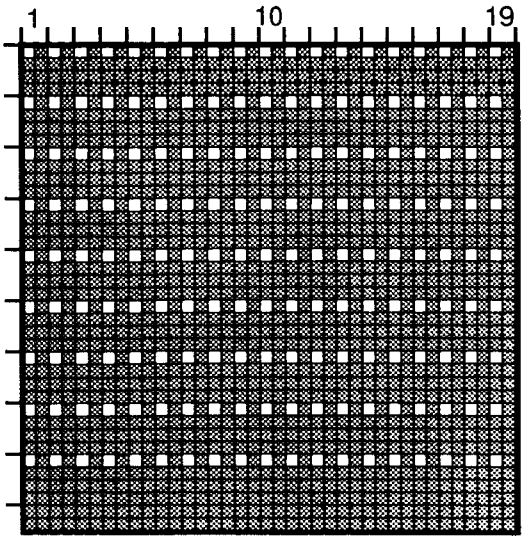


(a)　意味ネットワーク

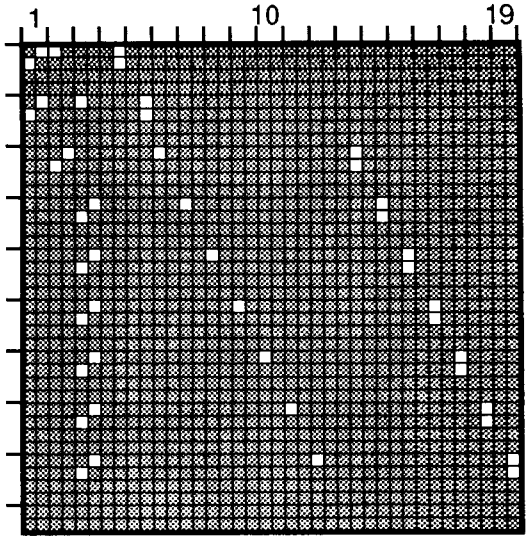


(b)　質問

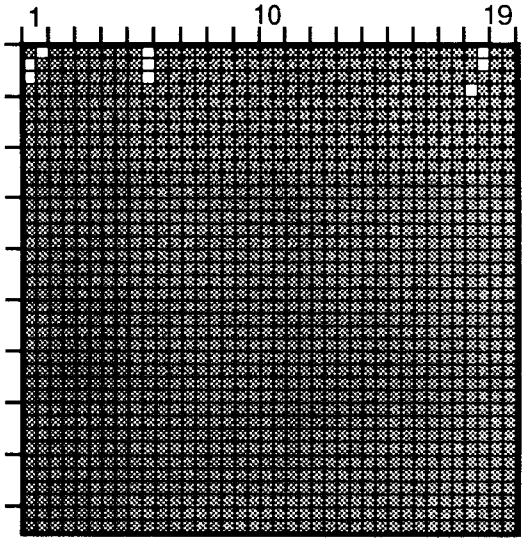
Fig.2.16　エキスパートシステムの意味ネットワーク



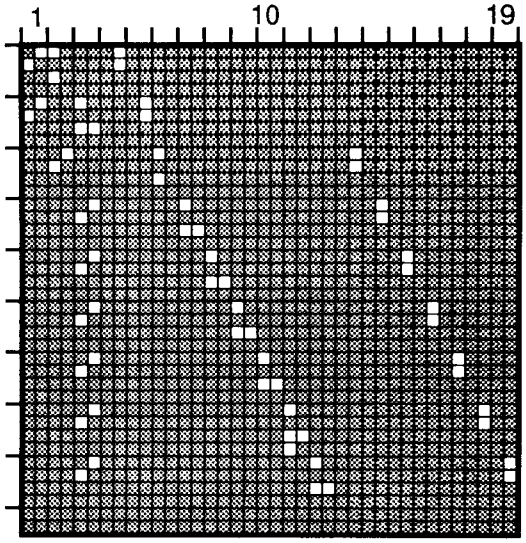
(a)　知識ベースのアトリビュートプレーン



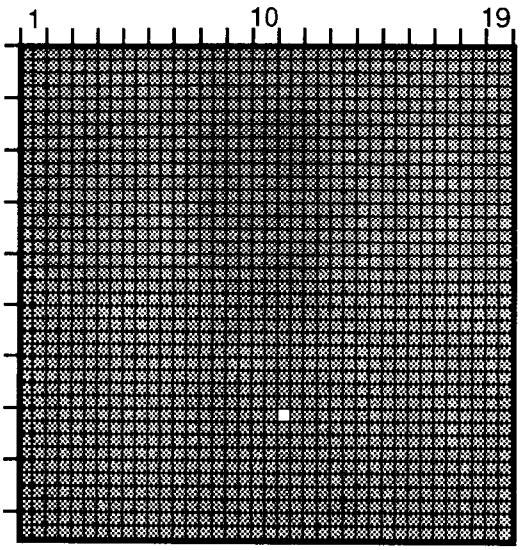
(b)　知識ベースのデータプレーン



(c)　質問



(d)　伝播したマーカが記録された知識ベースのデータプレーン



(e)　全マーカが記録されたノードを検出した結果

Fig.2.17　光アレイロジックによるエキスパートシステムのシミュレーション結果

2.6　処理効率評価

2.6.1　評価結果

OALLで記述されたプログラムの処理効率は，次の四つのパラメータ，1）符号化回数，2）相関回数，3）カーネルサイズ，4）画像サイズ，で評価する．1）と2）は処理速度に関するパラメータ，3）と4）は要求されるハードウェア仕様に関するパラメータである．

符号化回数を Ne，相関回数を Nc とし，それぞれ1回の処理時間を Te，Tc，サンプリングと論理和の処理時間を To とすると，ハードウェアでの処理時間 T は次のようになる．

1．並列相関型

$$T = Ne\,(Te + Tc + To), \tag{2-1}$$

2．逐次相関型

$$T = Ne\,Te + Nc\,(Tc + To). \tag{2-2}$$

ここで，並列相関型とは，光アレイロジック処理手順における積項演算を全て並列に実行するシステム，逐次相関型とは，積項演算を逐次的に実行するシステムである．また，ハードウェアでの処理時間に対して，次のような仮定をする：1）光アレイロジック処理の各手順の処理時間は，カーネル，画像サイズによらず一定とする，2）各手順を実行するための画像の読みだし，転送時間は，各手順の処理時間に含まれるものとする．

Table 2.1は，本章で検討した推論機構，エキスパートシステムの評価結果である．まず，2種類の推論機構の比較を行う．テンプレートマッチング法の推論機構では，処理効率を決定するパラメータである符号化回数，相関回数の値が，意味ネットワークのノード，リンク数や，同時に導出されるノードの数に依存している．しかし，トークン伝播法では，符号化回数，相関回数の値が推論1サイクル当たり一定である．このことは，エキスパートシステムなどで意味ネットワークが膨大になっても，トークン伝播法では推論1サイクル当たり一定の速度で処理できることを意味している．したがって，処理データが膨大になるほどトークン伝播法の方が高速に処理できることがわかる．

カーネルサイズと画像サイズは，テンプレートマッチング法の方がトークン伝播法より小さい．また，テンプレートマッチング法では，サブネットワークを画像内の任意の位置に配置できるため，画像の利用効率がよい．

以上をまとめると，テンプレートマッチング法では，画像の利用効率がよく，トークン伝播法では処理速度において優れていることがわかる．

トークン伝播法の推論機構をエキスパートシステムに拡張した場合の評価結果を推論機構と比べると，符号化回数，相関回数が多少増加したのみで，パラメータの大きな変化はないことがわかる．これより，光アレイロジックでは，推論機構が持つ特徴を損なわずに，効率のよい並列エキスパートシステムが実現できることがわかる．

Table 2.1　推論機構，エキスパートシステムの評価

		Inference Engine		Expert System
		Template Matching Method	Token Propagation Method	Token Propagation Method
Number of Encoding		$< (2S + 4N_{Di} + 4)\, i + 3$	$8\, i + 1$	$17i + 15$
Number of Correlation		$<[2S + \{3(P_N + P_L) + 5\}N_{Di} + P_N + P_L + 2]\, i + 3$	$12\, i + 1$	$21i + 14 + 2P_M P_H$
Kernel Size [/ Kernel Unit]	(x)	3	$(2S - 1)P_V$	$(2S - 1)P_V$
	(y)	$4\,(P_N + P_L) + 1$	$(2N - 1)P_H$	$(2N - 1)P_H$
Image Size [/ Pixel]	(x)	S	SP_V	SP_V
	(y)	$4\,(P_N + P_L)$	NP_H	NP_H

N : number of nodes
S : number of subnetworks
P_V : vertical pixel number of a node pattern
P_H : horizontal pixel number of a node pattern
P_M : vertical pixel number of marker number pixels
P_N : pixel number for a node
P_L : pixel number for a link
N_{Di} : number of nodes derived in the i -th cycle of inference
i : cycle number of inference sequence

ここで，本手法による処理速度の試算を行う．処理を実行するシステムは，Fig.1.3 の光・電子複合型 OPALS (H-OPALS) を想定する．このシステムでは，LED，PD の応答速度がそれぞれ 10^{-8} 秒，10^{-11} 秒のオーダーなのに対し [47]，光シャッタに用いる空間光変調器の応答速度は強誘電性液晶で 10^{-4}～10^{-5} のオーダーである [48]．したがって，システムの処理速度は，相関器で用いる空間光変調器の応答速度により決定される．ここでは，システムのサイクル時間を，相関演算時間 Tc として計算する．

処理時間は，(2-2) 式より次式で表せる．

$$T = Nc\, Tc \quad . \tag{2-3}$$

処理速度 V は，処理 1 サイクルにおける並列処理データ数 Np を用いて次のように書ける．

$$V = \frac{Np}{T} = \frac{Np}{NcTc} \quad . \tag{2-4}$$

この式に，Table 2.1 の相関演算回数を代入することにより，本手法の処理速度は，画像サイズ（画像の 1 辺の画素数）Ni と Tc をパラメータとして次のように書ける．

１．推論機構（テンプレートマッチング法）

$$V > \frac{Ni}{\left\{19Ni + 3Ni\log_2 Ni + \log_2 Ni + 6\right\} Tc} \cdot \tag{2-5}$$

２．推論機構（トークン伝播法）

$$V = \frac{Ni}{36Tc} \cdot \tag{2-6}$$

３．エキスパートシステム

$$V = \frac{Ni}{63Tc} \cdot \tag{2-7}$$

ただし，Ni は画像サイズ（画像の１辺の画素数）である．また，(2-4) 式から (2-5) 式の導出には，$Np = Ni,\ S = Ni,\ P_L = 4,\ N_{Di} = Ni$ を用い，(2-6), (2-7) 式の導出には $Np = Ni / 3$ を用いている．なお，(2-5) 式の不等号は，同時に導出されるノード数により処理速度が変動することを表す．

Fig.2.18 - 2.20 に，それぞれテンプレートマッチング法による推論機構，トークン伝播法による推論機構，エキスパートシステムの処理速度を，画像サイズとシステムのサイクル時間をパラメータとして描いたグラフを示す．ここでの処理速度は，１秒間の推論数で示しており，全データ並列に推論を行った場合の性能である．この場合，テンプレートマッチング法による推論機構の処理速度は，(2-5) 式で示される最小の処理速度となる．

テンプレートマッチング法による推論機構 (Fig.2.18) では，画像サイズが増加して並列度が増すにつれて，処理速度はわずかながら減少する．これは，本手法では，テンプレートマッチングで検出されたパターンを上から逐次的に検査する処理の効率が悪く，並列度が増すと，この回数が増すためである．この逐次的検査をする必要のない，すなわち検出されたパターンが各推論ステップで１個の場合 ($N_{Di} = 1$) の処理速度をグラフ上に示す．グラフより，この方法は同時に推論を行うノードの数が少ない場合に向いていることがわかる．

トークン伝播法による推論機構 (Fig.2.19) では，並列度が増すにつれて処理速度が向上する．最高速クラスの意味ネットワークマシンである SNAP [49] の性能もグラフに示している．グラフにおけるSNAP の横軸の値は，本手法により SNAP と同等の並列度の処理を実現するために必要な画像サイズである．グラフの比較により，例えばシステムのサイクル時間 $Tc = 10^{-6}$ の場合に，画像サイズが約 1000×1000 画素以上で SNAP の性能を上回ることがわかる．エキスパートシステム (Fig.2.20) も同様のことが言える．

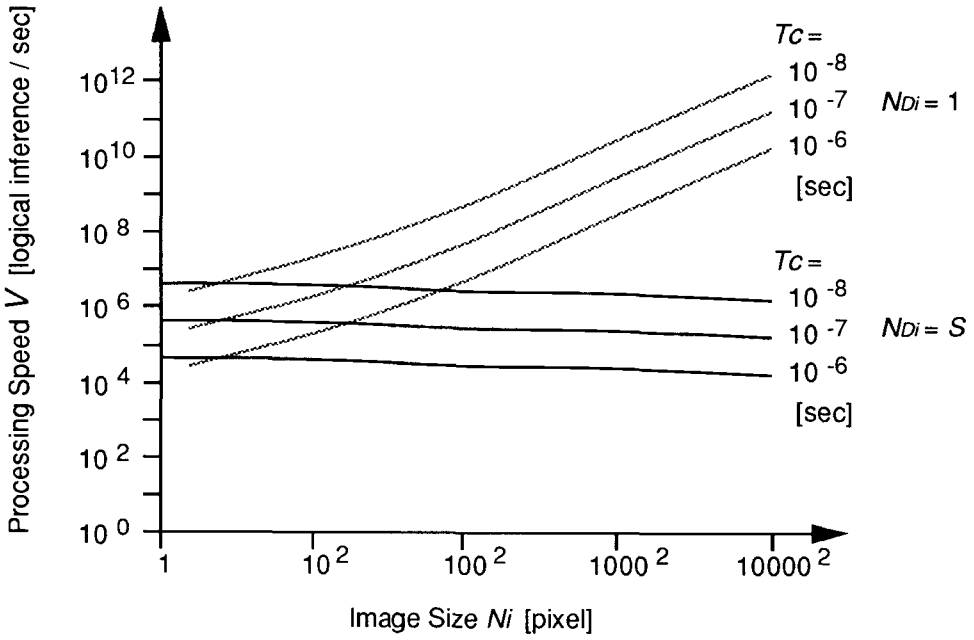


Fig.2.18　テンプレートマッチング法による推論機構の処理速度

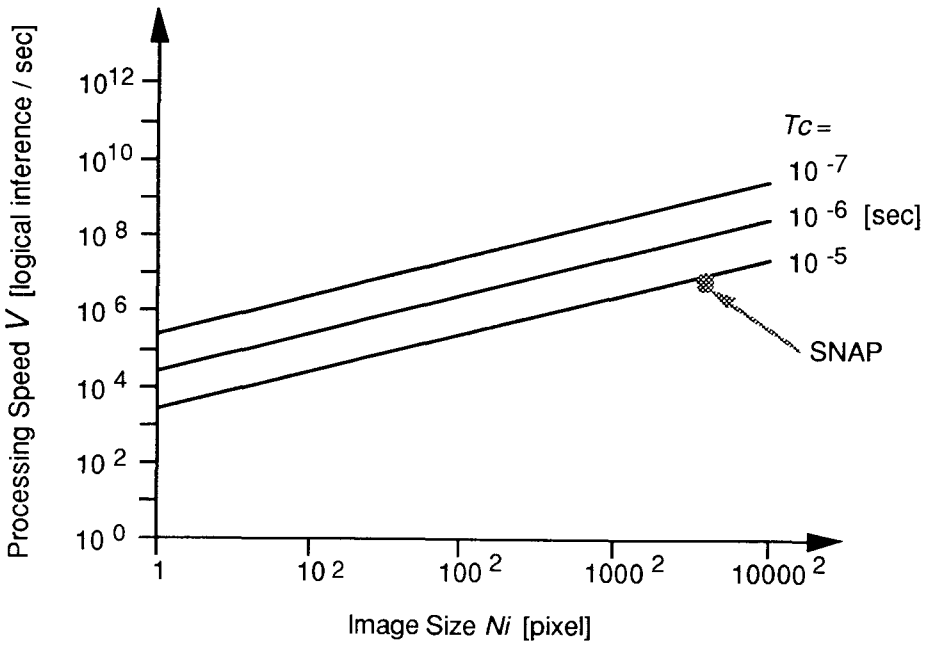


Fig.2.19　トークン伝播法による推論機構の処理速度

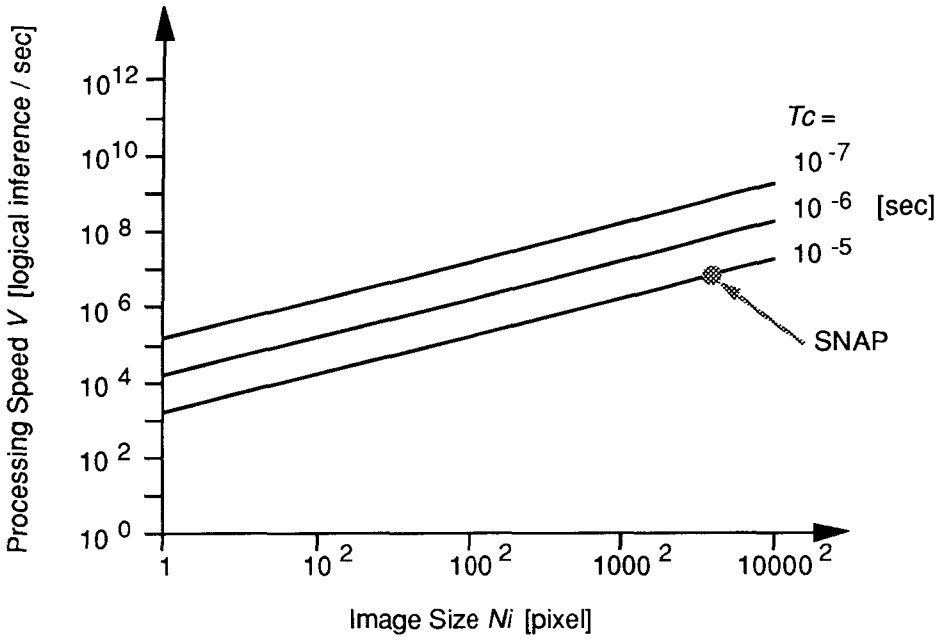


Fig.2.20　エキスパートシステムの処理速度

2.6.2　考察

前節の評価結果より，テンプレートマッチング法ではハードウェア仕様に対する要求がゆるい点で優れ，パターン伝播法では，処理速度において優れていることがわかった．しかし，テンプレートマッチング法では，個々のサブネットワークが独立しているため，画像内の任意の位置に配置できる半面，光アレイロジックによるSIMD方式の処理では，複雑な処理が難しい．一方，トークン伝播法では，配置の自由度がより少なく，多くの画素が必要であるが，複雑な処理に対してもSIMD方式の並列性がいかせる．よって，トークン伝播法のみがエキスパートシステムに拡張可能である．なお，本手法は，エキスパートシステムの主要部の処理を行っただけであり，一つの完成されたシステムにするためには，ユーザインターフェースなどを整える必要がある．しかし，これらの部分は光の並列性を必要としないので，電子コンピュータで十分機能する．

エキスパートシステムでは，通常大容量データを扱わなければならない．標準的なエキスパートシステムでは，少なくとも数千のノードを扱う必要がある．もし5000ノードが必要なら，テンプレートマッチング法では約1000×1000画素の画像と約100×100ユニットのカーネルが必要であり，パターン伝播法とそれを用いたエキスパートシステムでは，約10000×10000画素の画像と，20000×20000ユニットのカーネルが必要となる．現在開発中の光アレイロジックシステムでは，扱える画素数が数十であり，現在最も優れている光学素子の性能を最大限に活用しても，１辺数千画素が限界と考えられる．すると，本手法を光演算システムで実現するためには，システムが一度に処理できる画像より画素数が多い画像を扱う方法の開発が必要となる．その解決策として，画像を分割してページごとに処理する手法である，２次元仮想記憶機構を考案した．この手法については，第５章で述べる．

2.7　結言

本章では，光アレイロジックの並列性を知識ベースの探索に有効利用した推論機構の実現方法としてテンプレートマッチング法とトークン伝播法を考案した．さらに，推論機構の機能を拡張してエキスパートシステムの処理を実現する方法を検討した．これらの処理の光アレイロジックプログラムを作成し，処理効率を評価した．その結果，テンプレートマッチング法では，ハードウェアに対する要求がゆるい面で優れるが，トークン伝播法のほうが，処理速度，機能の拡張性において優れていることを確認した．また，トークン伝播法を用いた並列エキスパートシステムが，推論機構の並列性を損なわずに実行できることを示した．ただし，これらの手法では，通常画素数の多い画像を扱う必要があるため，ハードウェアが扱える画素数より多い画像を扱う方法の開発が必要であるとの知見を得た．

第3章　光アレイロジックによるデータフロー型処理

3.1　緒言

並列コンピュータで効率的な処理を行わせるためには，各プロセッサへ最適な形で処理を分散させる必要がある．しかし，その分散配置は，対象とする問題によって異なる．したがって，汎用並列コンピュータは，プロセッサ構成が可変であることが望ましい[13].

光アレイロジックを用いると，再構成可能な並列プロセッサを仮想的に実現できる[25]．その概念図をFig.3.1に示す．画像上に処理要素のレジスタ領域を多数配置する．各処理要素の演算と処理要素間のデータ転送を，光アレイロジックで並列に実行することにより，多数の処理要素からなる並列マシンを仮想的に実現できる．この手法は，処理要素を画像上の任意の位置に配置できるため，プロセッサの再構成が簡単であるという特徴を持つ．本章では，並列プロセッサの一つとしてデータフローマシンを取り上げ，その処理方式であるデータフロー型処理を光アレイロジックにより実現する方法を検討する．

データフロー型処理[10, 50, 51]は，処理に内在する並列処理構造をそのまま抽出し，高度の並列処理を実現する処理方法である．これは，演算に必要なデータがそろったものから順次実行する非同期的な処理形式をとる．データフロー型処理の手順には，データの転送，被処理データの準備状態の検出，演算の実行において，それぞれ並列性が存在するため，処理効率の向上が期待できる．

本章では，データの転送と被処理データの準備状態の検出が，簡単なSIMD方式の並列処理で実現可能なことに着目して，光アレイロジックによるデータフロー型処理の実現方法を検討した．ただし，各プロセッサにおける演算の実行は一般にMIMD方式の処理であるため，この部分は複数の専用光演算モジュールを用いるものと仮定した．したがって，この方法では，複数の専用モジュールの並列制御を，光アレイロジックによるデータフロー型処理で実現することを目標とする．以下，3.2節でデータフロー型処理を概説し，3.3，3.4節で光アレイロジックを用いる場合の2値画像への符号化法と，処理の実現方法を述べる．3.5節で処理効率を評価し，3.6節でデータフロー型処理を専用光演算モジュール間接続の制御に応用する方法を検討する．

3.2　データフロー型処理

データフロー型処理[10, 50, 51]は，演算に必要なデータがそろった演算を順次実行

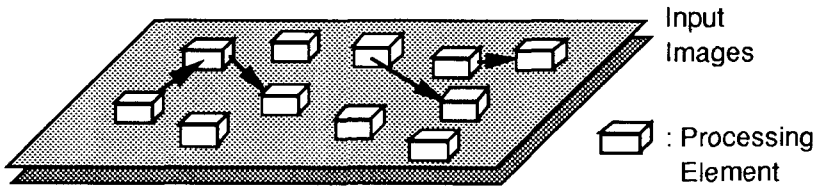


Fig.3.1　光アレイロジック上での仮想マシンの概念

する非同期的な処理形式である．この処理に用いられるプログラムは，データフローグラフと呼ばれる有向グラフで表される．Fig.3.2 にその例を示す．データフローグラフは，演算や制御を示すノードと，ノード間を結ぶアークからなる．データフロー型処理は，データを表すトークンを複数のアーク上で並列に伝播させることにより遂行される．Fig.3.3 にトークン伝播の例を示す．まず，入力アーク a, b からデータであるトークンを入力し，アーク上を伝播させる．そして，必要なトークンがそろったノードの演算を実行する．これをノードの発火という．この例では＋，－のノードにおける演算が並列に実行される．それぞれの演算結果のトークンを出力し，×の演算を実行することにより，所望の演算結果が得られる．このように，データフロー型処理では命令の並列性を自然に抽出できる特徴がある．

データフロー型処理では，1）トークンの伝播，2）必要なトークンがそろったノードの検出，3）演算の実行，に並列性が存在する．特に1）と2）には SIMD 方式の並列性が存在するので，光アレイロジックが有効に利用できる．

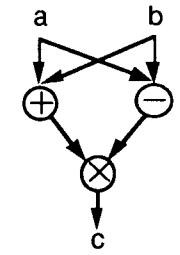


Fig.3.2　データフローグラフ

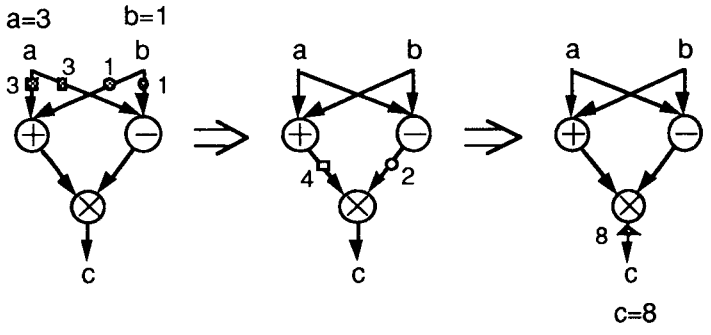
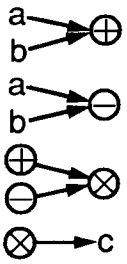


Fig.3.3　トークン伝播例

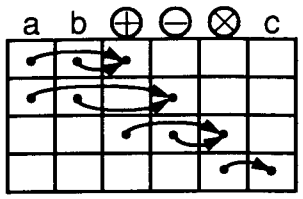
3.3　データフローグラフの符号化法

光アレイロジックでデータフロー型処理を行うために，データフローグラフを2値画像に符号化する．その手順を Fig.3.4 に示す．まず，データフローグラフをサブグラフの集合に分割する．各サブグラフは，Fig.3.4(a) に示すように，複数のノードから1ノードへと接続されている形をとる．次に，サブグラフの集合を Fig.3.4(b) のような表形式に変換する．表の各行と列にそれぞれサブグラフ，ノードを1個ずつ割り当てる．

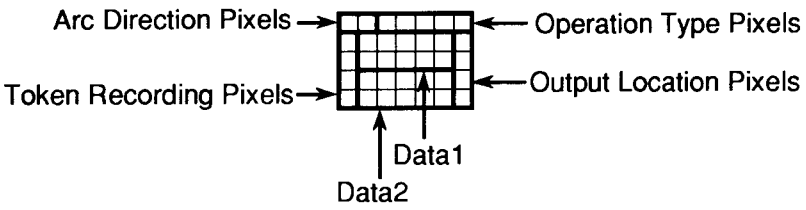
表の各セルを，Fig.3.4(c) に示すように5×7画素からなる画素パターンの集合に変換する．各属性を示す画素パターンの例を Fig.3.4(d) に示す．なお，この例ではノードの演算に MSD 加減乗算 [25] を使用するため，データは MSD 5 ビットで表す．この方法で各セルを画素パターンに変換することにより，データフローグラフを表す2値画素パターンが得られる（Fig.3.4(e)）．この画素パターンを持つ画像を，データフローグラフのデータプレーンと呼ぶ．さらに，このデータプレーン上の各セルの位置を示す目印を持つ画像を作成する（Fig.3.4(f)）．これをデータフローグラフのアトリビュートプレーンと呼ぶ．データフローグラフは，これらの2枚の画像により表現される．



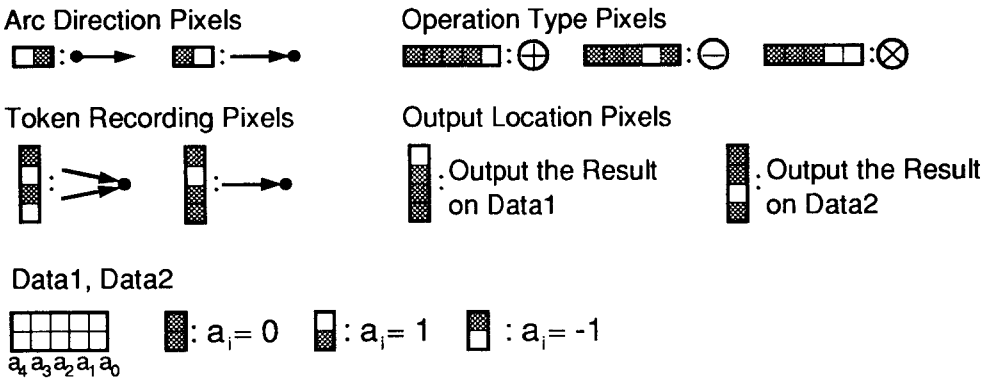
(a)　データフローグラフのサブグラフへの分割



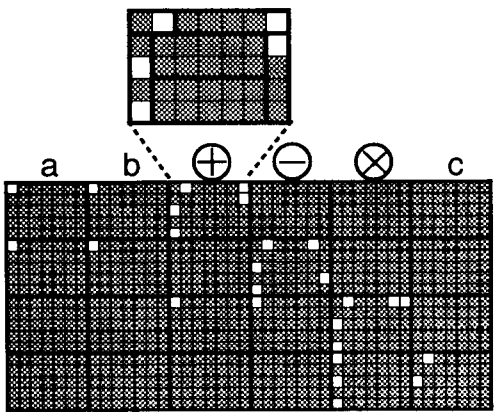
(b)　データフローグラフの表形式による表現



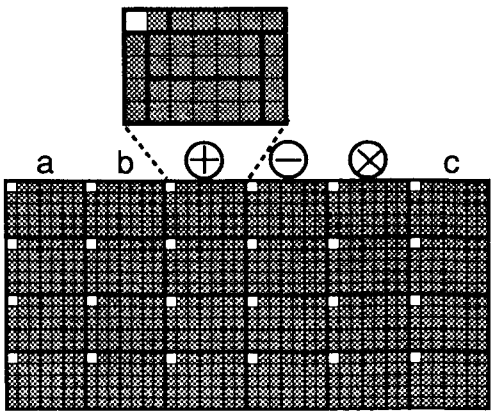
(c)　表の１セルの画素パターンへの符号化法



(d)　表の１セルの画素パターンへの符号化例



(e)　データフローグラフのデータプレーン



(f)　データフローグラフのアトリビュートプレーン

Fig.3.4　データフローグラフの符号化

３.４　実現方法

データフロー型処理は，データフローグラフ上でのトークン伝播により処理が進行する．光アレイロジックでは，２値画像に符号化されたデータフローグラフ上で，トークンを表す２値画素パターンを並列に伝播させることにより効率よく実行できる．トークンの並列伝播には，光アレイロジックによるトークン伝播の手法（１.５.４節）を用いる．

データフロー型処理の基本的な手順は次のようになる．

１）表形式で表現したデータフローグラフの各行に割り当てたサブグラフの中から，入力されたトークンに対応するものを検索する．

２）検索したサブグラフのセルに，トークンをセットする．

３）トークンを同一行内でリンクに沿って伝播させる．

４）必要なトークンがそろった（発火した）セルを検出する．

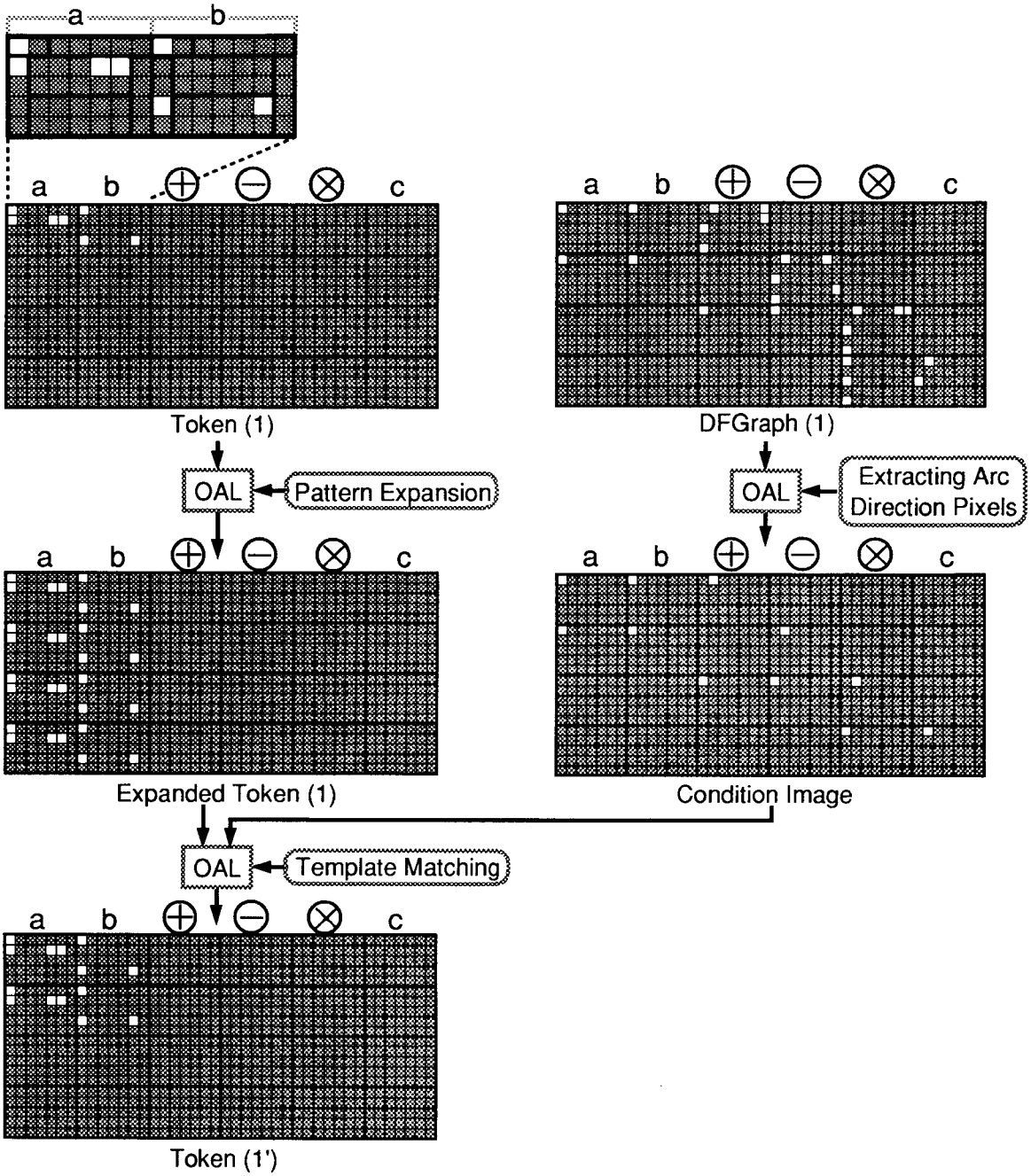
５）検出されたセルで指定された演算を実行する．

６）演算結果をセットしたトークンを，演算を実行したセルに配置する．

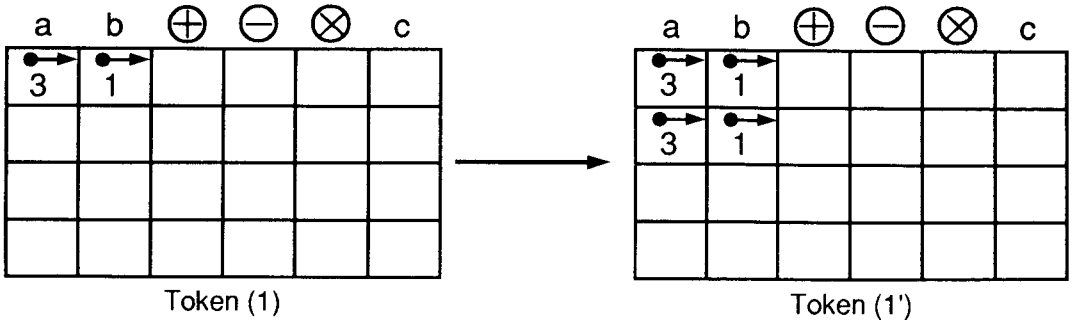
７）ステップ１）～６）の処理を繰り返し，トークンがデータ出力ノードに到達したら，処理を終了する．

Fig.3.5 - 3.13 は，Fig.3.3 の処理の例である．ステップ１）と２）は，Fig.3.5 の縦方向トークン伝播で実行できる．ここで，条件画像 Condition Image は，データフローグラフの画像から，アーク方向を表す画素を抽出したもので，トークンの伝播先を示している．この画像を用いて，画像 Token(1) 上のトークンを並列に伝播させる．ここでは，トークンは５×７画素のパターンである．この結果，アークの始点を持つノード a, b のセルへ縦方向にトークンが伝播し，画像 Token(1') が得られる．

ステップ３）の準備として，アークの終点があるノードを探索するため，トークン上のアーク方向画素を変更する (Fig.3.6)．そして，Fig.3.7 に示すように，ステップ３）を実行する．各トークンが乗っている各サブグラフの行のうち，アークの終点を持つセルを画像 Condition Image から見つけて，横方向トークン伝播を行う．転送されたトークンと，画像 DFGraph(2) との論理和をとり，その結果を画像 DFGraph(2) 上に記録する（Fig.3.8）．ステップ４）では，画像 DFGraph(2) 上のトークン記録画素のテンプレートマッチングにより，必要なトークンがすべてそろった（発火した）ノードを検出する（Fig.3.8）．ステップ５）として，ここでは MSD 加減乗算 [25] を実行する．演算結果は，演算を行ったセルのデータ１の領域に出力させる（Fig.3.9）．ステップ６）では，まず発火したノードに対応するセルに記録されている画像 DFGraph (2) 上のトークンを消去する（Fig.3.10）．次に，演算結果が指定位置にセットした画像 Calculated Result (1) を得る（Fig.3.11）．そして，Fig.3.12 のように，画像 DFGraph (1) から，アーク方向画素とトークン記録画素を抽出し，画像 Token Temp を得る．ただし，アーク方向画素は，矢印の先のノードを探索するように変換する．画像 Token Temp と，画像 Calculated Result (1) との論理和により，新しいトークンを生成する．Fig.3.13 に上記処理における画像上でのトークン伝播の様子を示す．

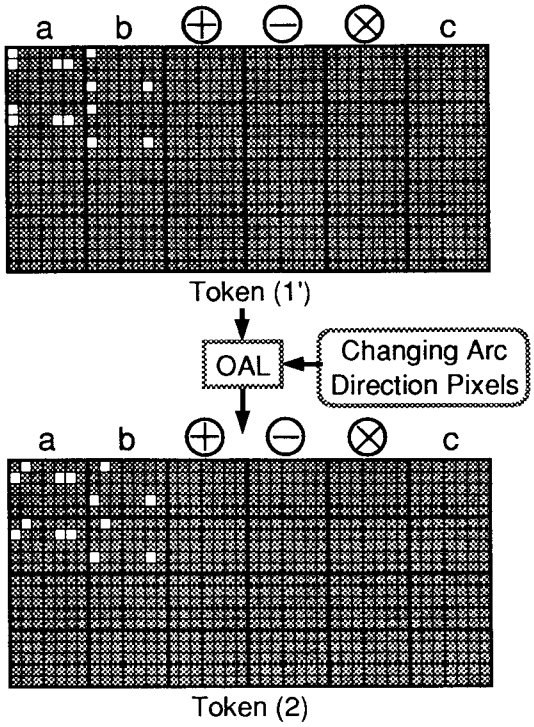


(a)　処理画像の流れ

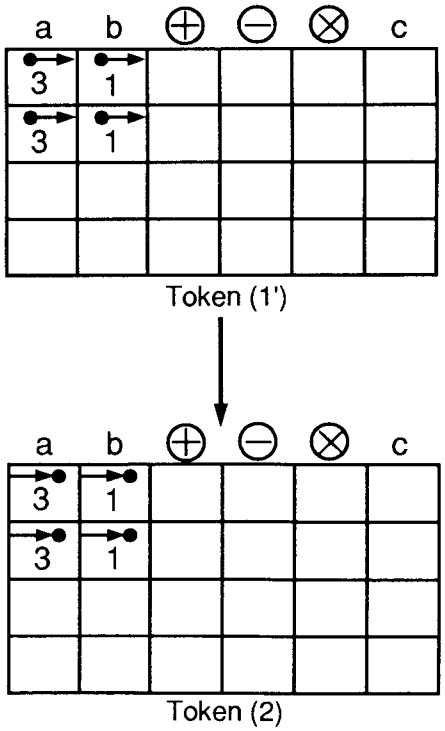


(b)　表形式表現によるトークン画像の変化の様子

Fig.3.5　縦方向トークン伝播の処理手順

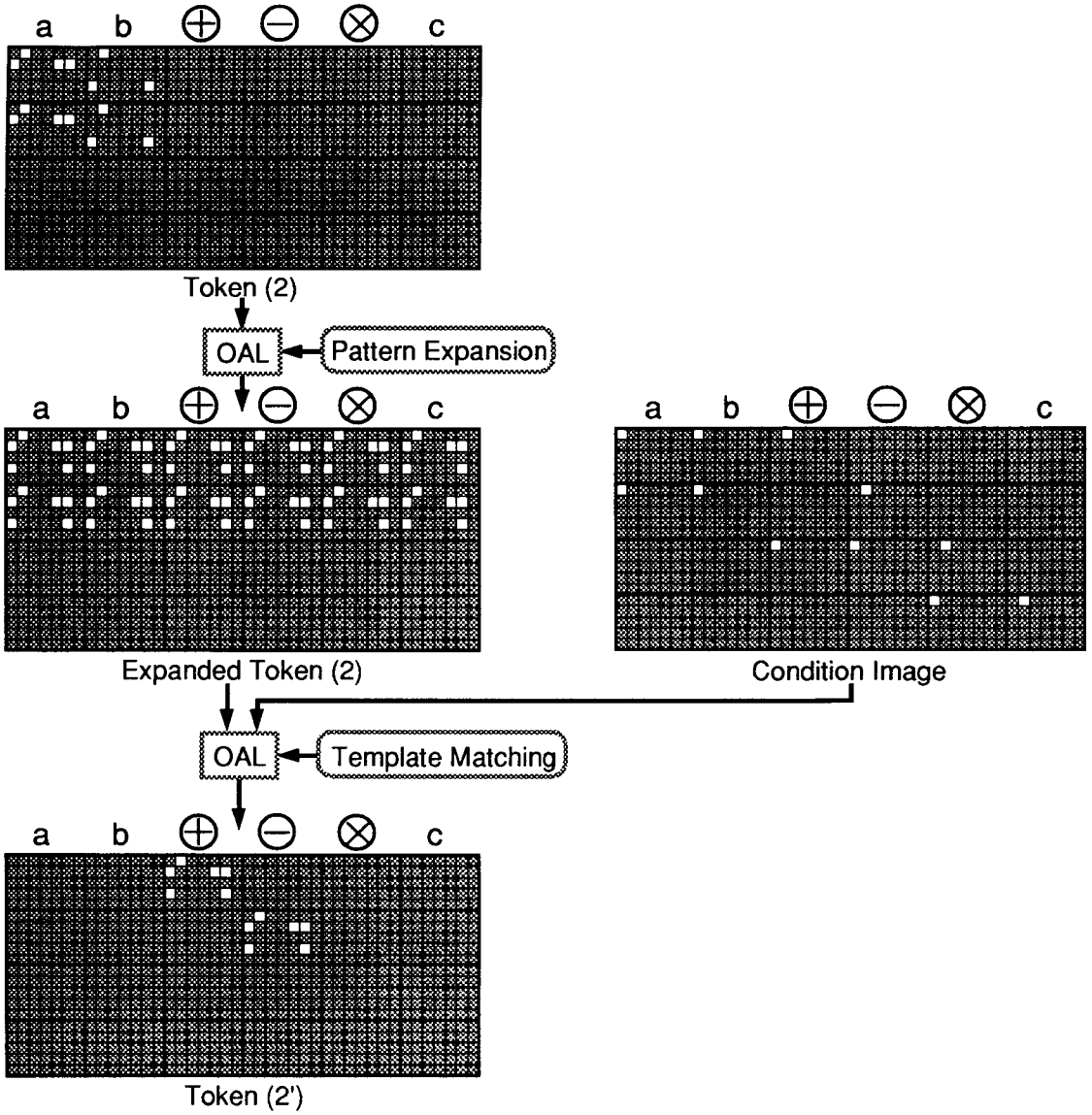


(a)　処理画像の流れ

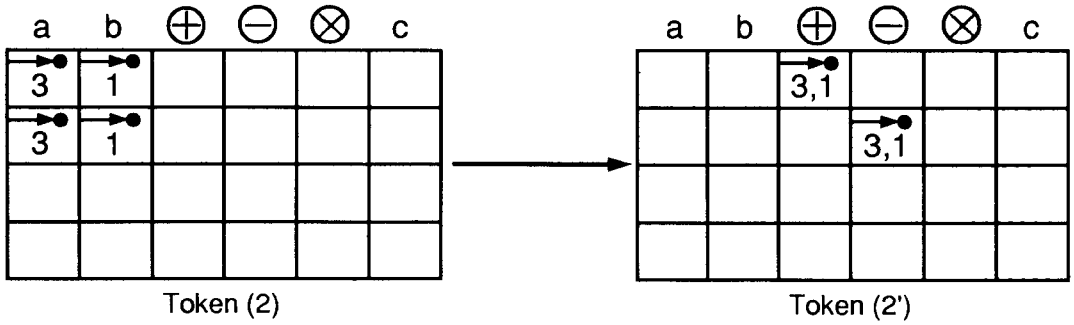


(b)　表形式表現によるトークン画像の
変化の様子

Fig.3.6　トークンアーク方向画素の変更



(a)　処理画像の流れ



(b)　表形式表現によるトークン画像の変化の様子

Fig.3.7　横方向トークン伝播の処理手順

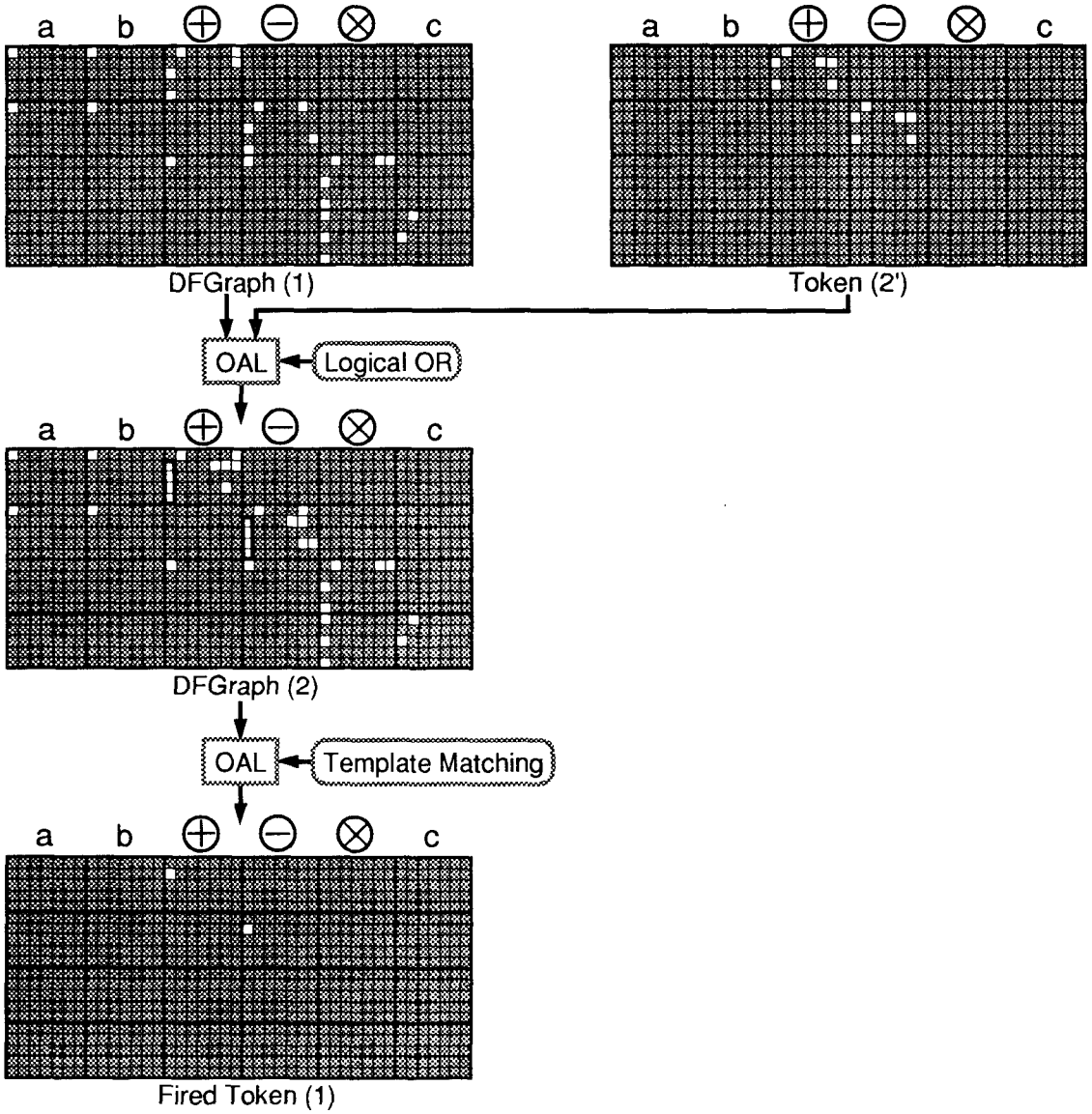
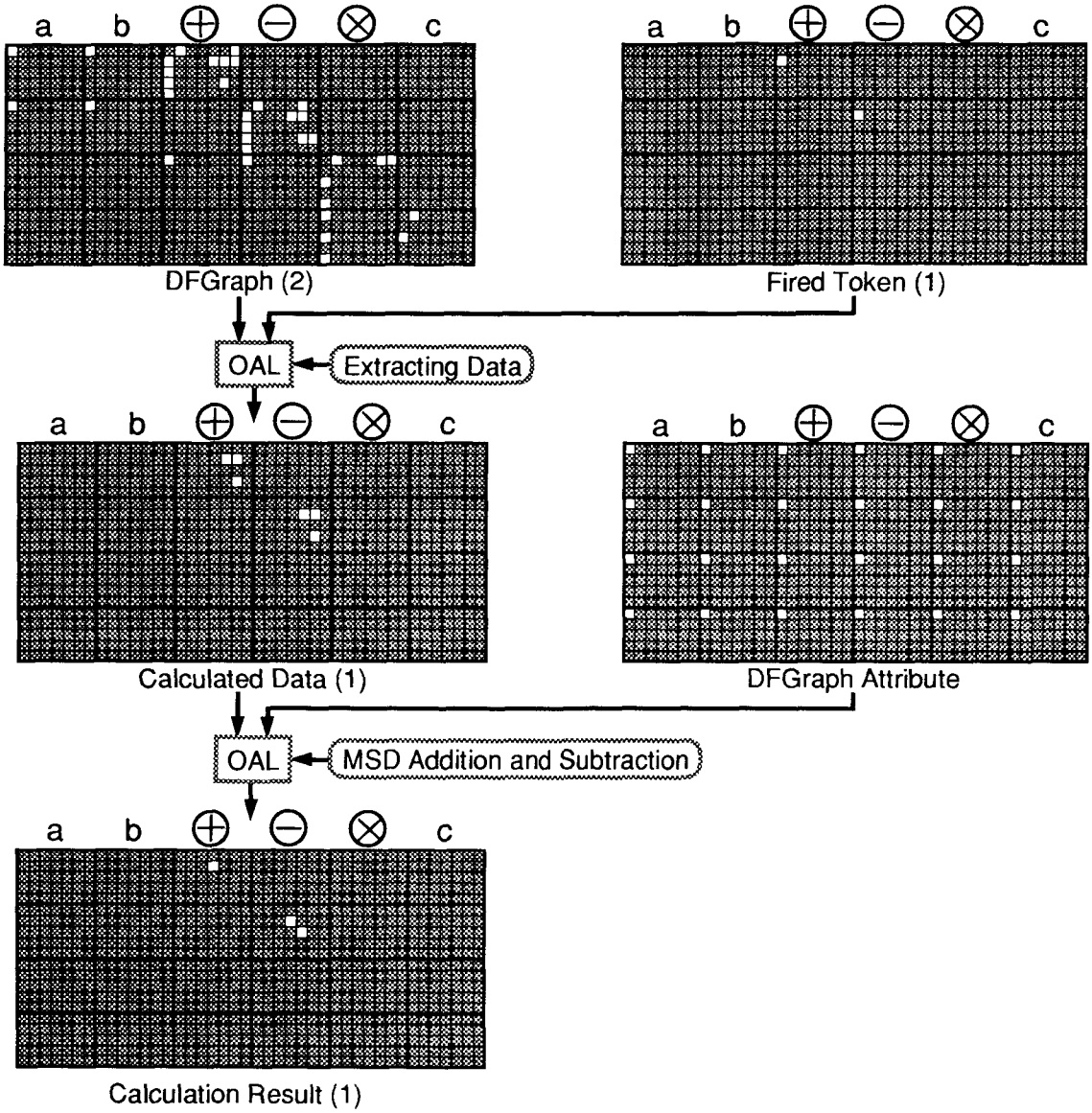
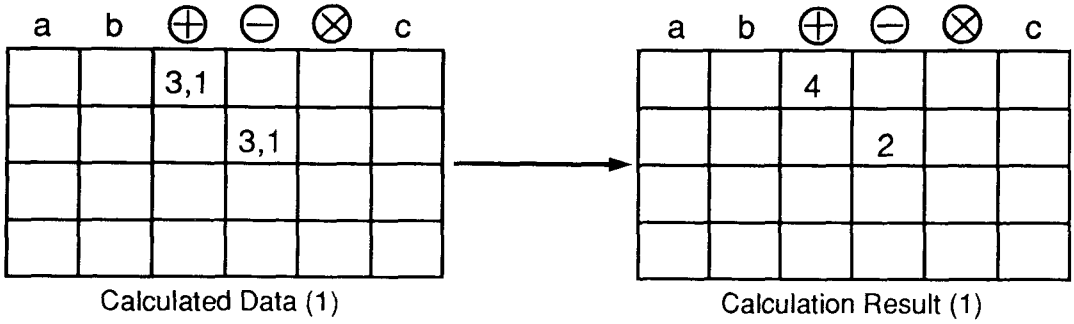


Fig.3.8　伝播したトークンの記録とトークンがそろったノードの検出の処理手順



(a)　処理画像の流れ



(b)　表形式表現による演算データの変化の様子

Fig.3.9　発火したノードの演算の処理手順

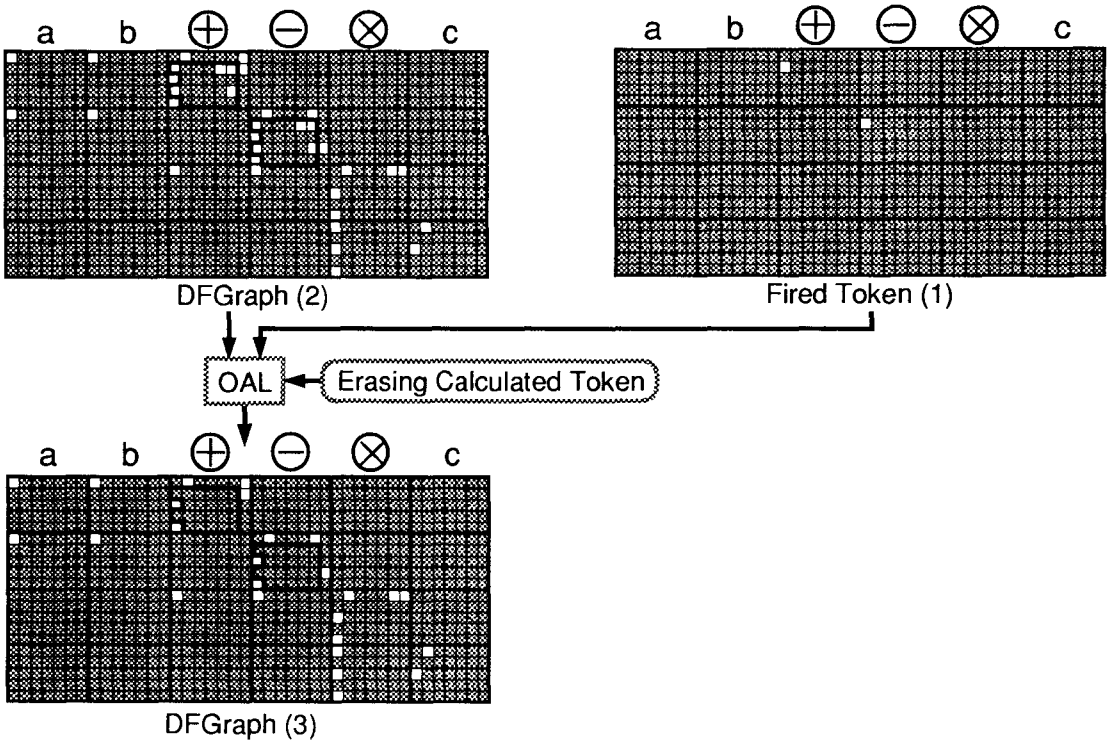


Fig.3.10　発火したノードに記録されているトークンの消去

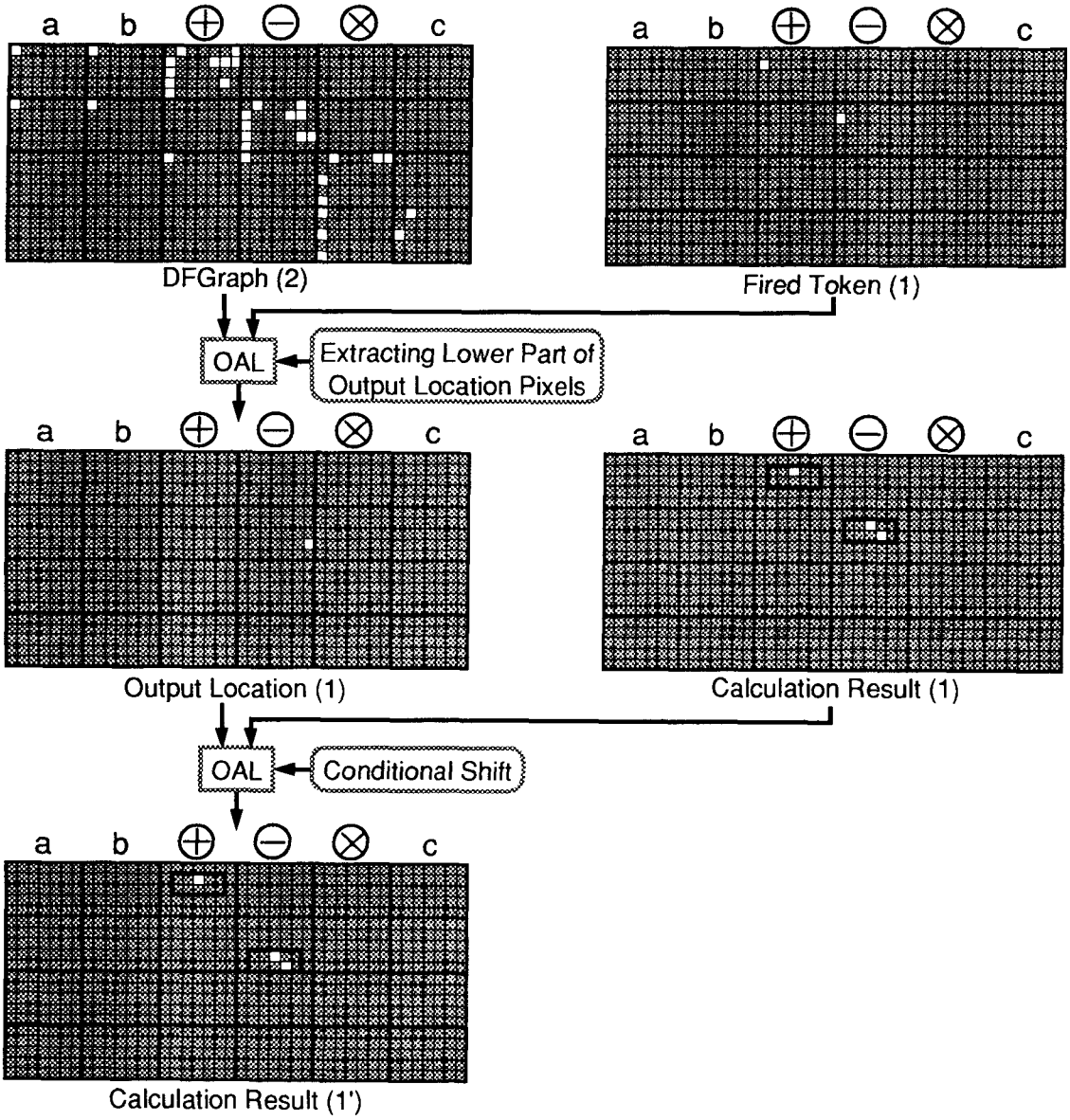
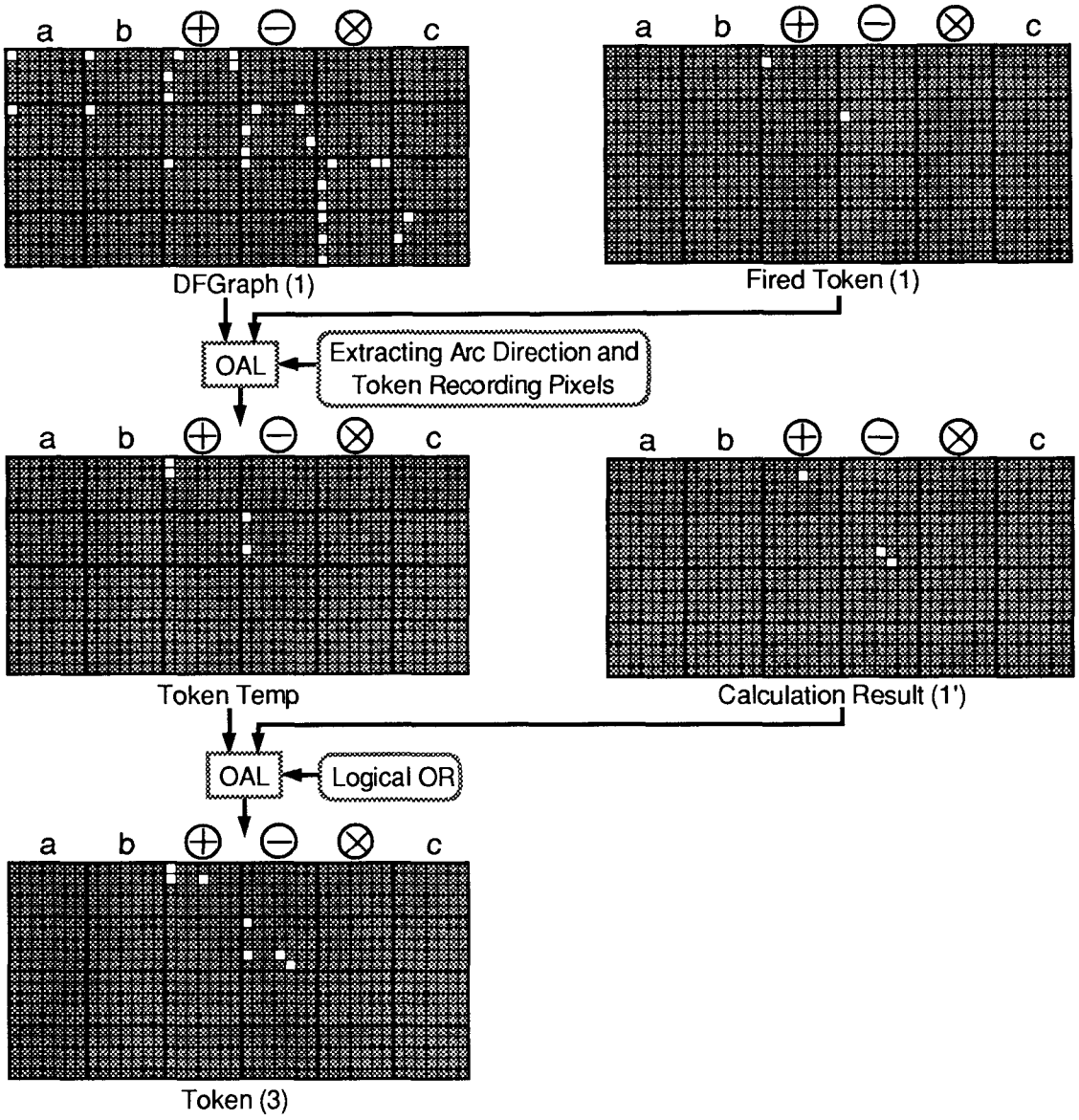
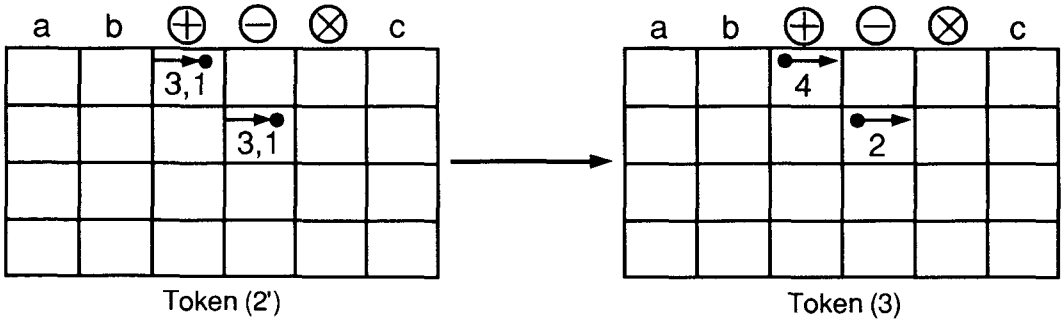


Fig.3.11　演算結果の指定位置へのセットの処理手順



(a)　処理画像の流れ



(b)　表形式表現によるトークン画像の変化の様子

Fig.3.12　新たなトークンの出力の処理手順

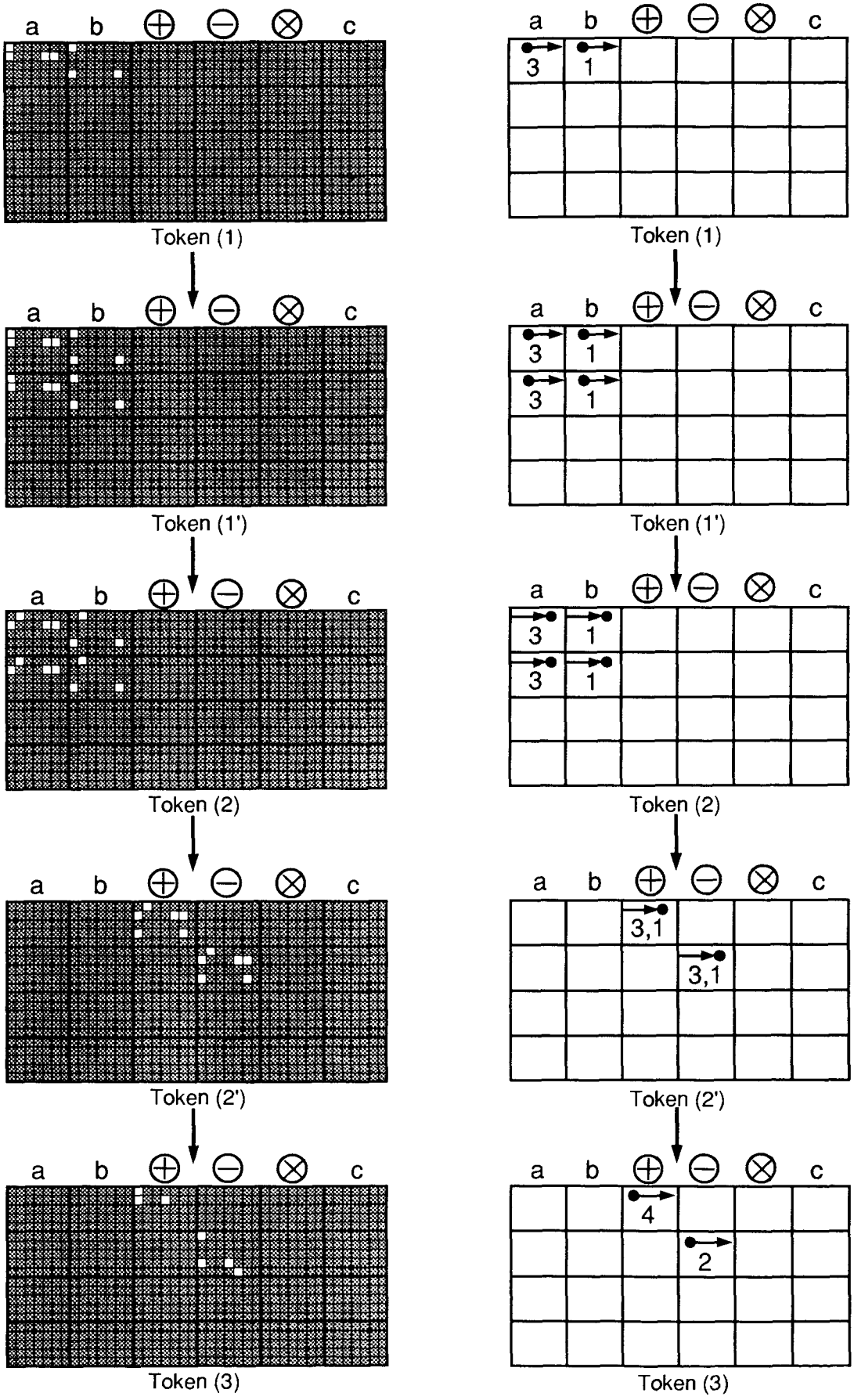
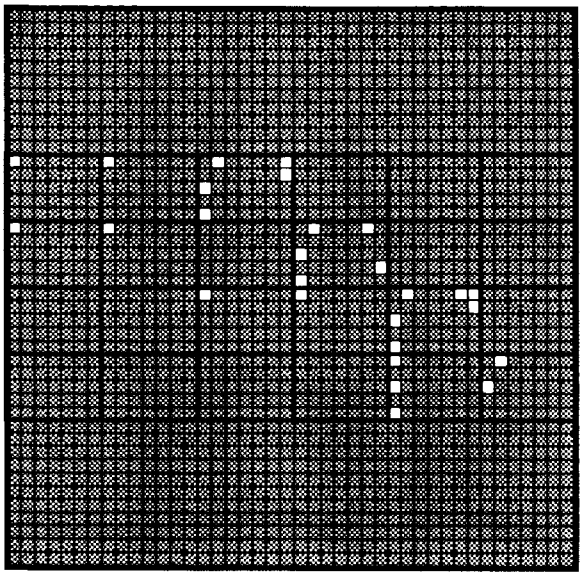
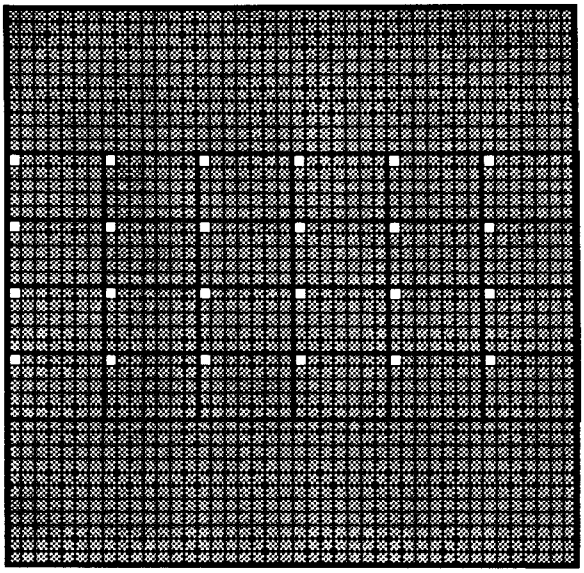


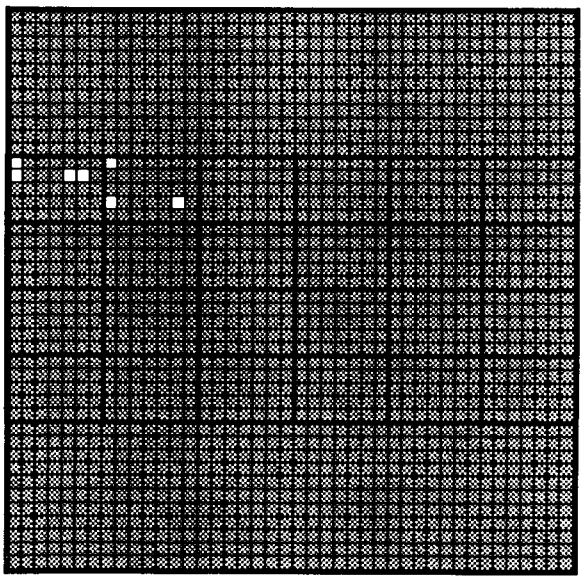
Fig.3.13　トークン伝播の様子



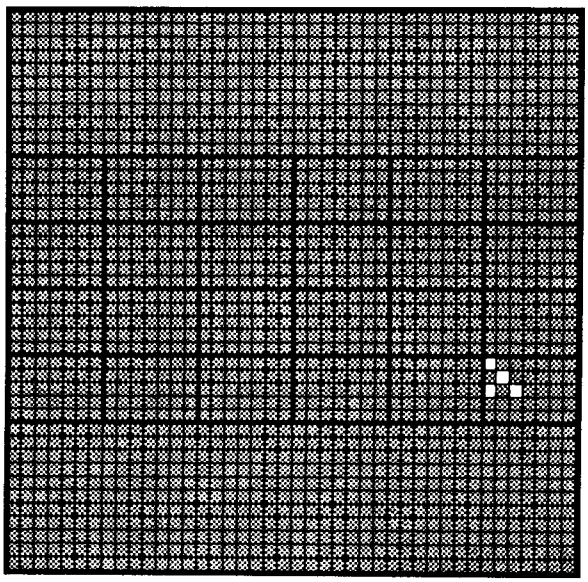
(a)　データフローグラフのデータプレーン



(b)　データフローグラフのアトリビュートプレーン



(c)　トークン



(d)　処理結果

Fig.3.14　データフロー型処理のシミュレーション結果

上記処理を繰り返し行って実行したデータフロー型処理のシミュレーション結果をFig.3.14に示す．Fig.3.14(a), (b)は，それぞれFig.3.2のデータフローグラフのデータプレーン，アトリビュートプレーンを表す．Fig.3.14(c)の入力トークンを表す画素パターンに対し，上記の処理を3サイクル行うと，Fig.3.14(d)の結果画像が得られる．出力ノードには，正しい演算結果8が得られている．Appendix A.3とAppendix B.4にそれぞれ本処理のカーネル式とOALLプログラムを示す．

3.5　処理効率評価

3.5.1　評価結果

光アレイロジックによるデータフロー型処理の処理効率の評価結果をTable 3.1に示

Table 3.1　データフロー型処理の評価

Number of Encoding		$22i$
Number of Correlation		$27i$
Kernel Size [/ Kernel Unit]	(x)	$< 5(2N - 1)$
	(y)	$(B + 2)(2N - 1)$
Image Size [/ Pixel]	(x)	$< 5N$
	(y)	$(B + 2)N$

i : cycle number of data flow processing sequence
N : number of nodes
B : number of bit for a data

す．評価で用いるパラメータは，２．７節のものと同様である．なお，ここでは一般的なデータフロー型処理の評価を行うことを目的としているため，各ノードで実行するMSD加減乗算に関する評価は除いてある．

この表から，本処理では，光アレイロジックプロセッサの処理速度を決定するパラメータである符号化回数と相関回数が，処理１サイクル当たり一定であることがわかる．これより，全トークンが並列に効率よく転送されていることがわかる．ただし，カーネルサイズと画像サイズは，データフローグラフのノード数，データのビット数に依存するため，ハードウェアに対する要求は厳しいものとなる．

ここで，本手法の処理時間を２．６．１項と同じ方法を用いて試算する．(2-4) 式にTable 3.1 の相関演算回数を代入することにより，処理速度 V は次式で表せる．

$$V > \frac{7.4 \times 10^{-3} Ni}{Tc} \quad . \tag{3-1}$$

ただし，Ni は画像サイズ（画像の１辺の画素数）である．また，上式の導出には，$Np > Ni/5$ を用いている．

Fig.3.15 に，画像サイズとシステムのサイクル時間をパラメータとして描いた処理速度のグラフを示す．ここでの処理速度は，１秒間に転送するデータのパケット数で示している．グラフには，データフローマシンの転送速度の例として，電総研の高並列計算機 EM-4 [52] における転送速度を併記している．グラフにおける EM-4 の横軸の値は，本手法により EM-4 と同等の並列度を実現するために必要な画素数である．これらのグラフの比較より，例えばH-OPALS のサイクル時間 $Tc = 10^{-10}$ の場合，画像サイズが約200×200 以上で EM-4 の性能を上回ることがわかる．

3.5.2　考察

データフロー型処理における各ノードの演算は，一般的には非同期的な MIMD 方式の並列処理である．光アレイロジックでは各ノードの演算を MIMD 方式で実行することは

難しく，非同期的な処理は不可能である．また，各ノードでは演算結果は数ビット分しか得られないので，光アレイロジックの並列性を十分利用しているとは言えない．そこで，各ノードに専用光演算モジュールを1台ずつ割り付け，トークンに画像単位でデータをのせることにより，各ノードの演算をMIMD方式で実行し，演算結果を画像上に多数同時に得る方法を検討した．それについては次節で述べる．

光アレイロジックでデータフロー型処理を実行する場合，プログラムをデータフローグラフで書き，それを変換した画像をシステムに入力する手順をとる．すなわち，データフローグラフを表現した画像がプログラムであり，データフロー型処理の光アレイロジックプログラムは，データフローグラフのコンパイラに相当する．したがって，本手法において，データフローグラフは光アレイロジックの上位言語と考えることができる．

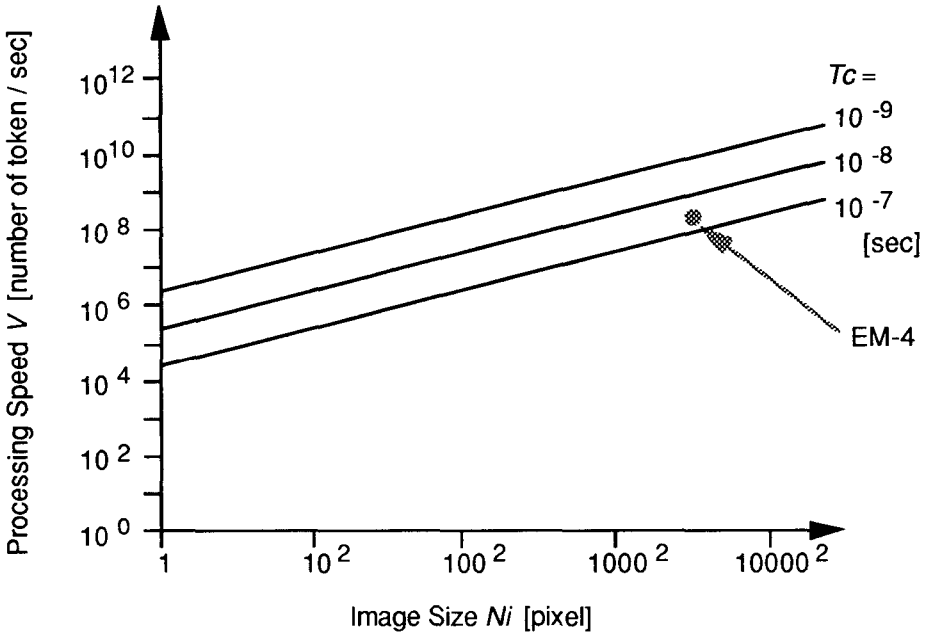
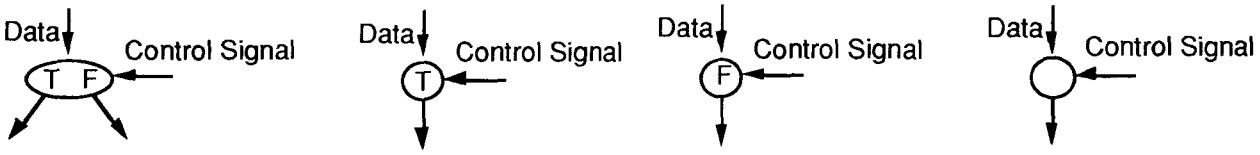


Fig.3.15　データフロー型処理の処理速度

データフローグラフを光アレイロジックの上位言語として使用すると，処理に内在する命令間の並列性を自然に引き出せ，データの画像内配置や演算カーネルを考えずに並列プログラムが作成できる．しかし，この方法では，プログラム量が増加するにつれて，データフローグラフを表す画像が大きくなるため，ハードウェアに対する要求はより厳しくなる．この問題の解決法として，第5章で述べる2次元仮想記憶機構や，第6章で述べる専用光演算モジュールの利用が考えられる．

データフローグラフでは，条件分岐や同期をとる処理を行うために，制御用信号をトークン上に乗せる方法がある．この場合，制御用信号を扱うノードとして，Fig.3.16のようなものが考えられている[10]．ノード内のT, Fは，それぞれ制御用信号の値が1 (True) または0 (False) の場合に，データのトークンが出力されることを表している．これらのノードでは，制御用信号とデータを同様に扱えるので，本手法によりこれらのノードの処理を行うことができる．



(a) TFスイッチ　(b) Tスイッチ　(c) Fスイッチ　(d) ゲート

Fig.3.16　制御用信号を扱うノード

3.6　専用光演算モジュール間接続制御への応用

光アレイロジックによるデータフロー型処理では，各ノードの演算をMIMD方式で実行することができない．また，一度に得られる演算結果はたかだか数ビット長の数値データであるため，光アレイロジックの並列性を十分利用しているとは言えない．そこで，一つの数値データをトークンとするのではなく，複数のデータを含む1画像をトークンとして利用する方法を考案した（Fig.3.17）．この場合，1画像上のデータは各専用光演算モジュールで並列に演算が行われる．各ノードで行われる演算はSIMD方式であるが，システム全体ではMIMD方式の並列処理が実現される．この場合，データフロー型処理を行う光アレイロジックプロセッサは，専用モジュール間画像転送の制御を行うと考えられる．しかし，トークンに画像データを直接乗せると，トークンを表す画素パターンが大きくなる．そこで，画像名を表す画素パターンを利用する．各モジュールは，

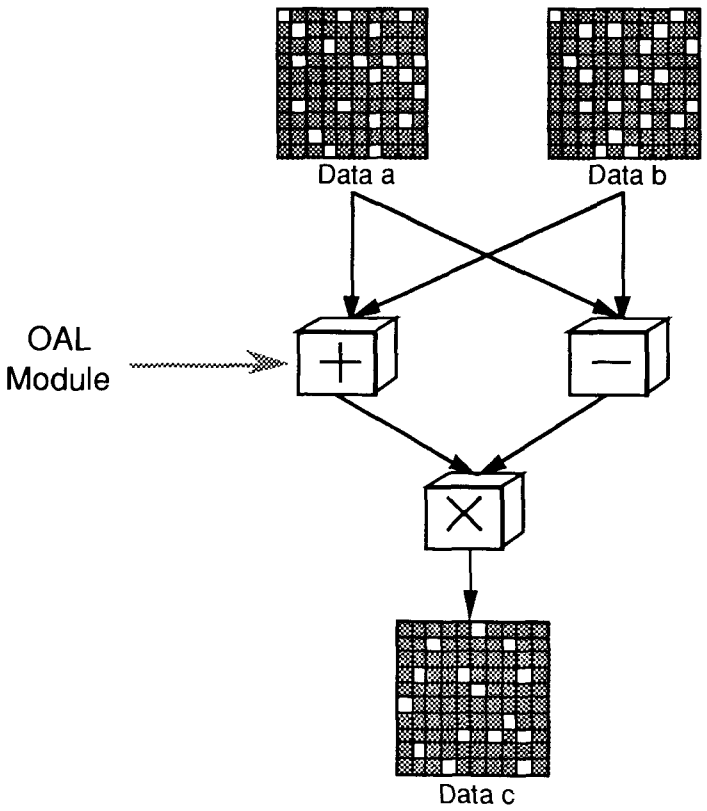


Fig.3.17　専用光演算モジュールを用いたデータフロー型処理

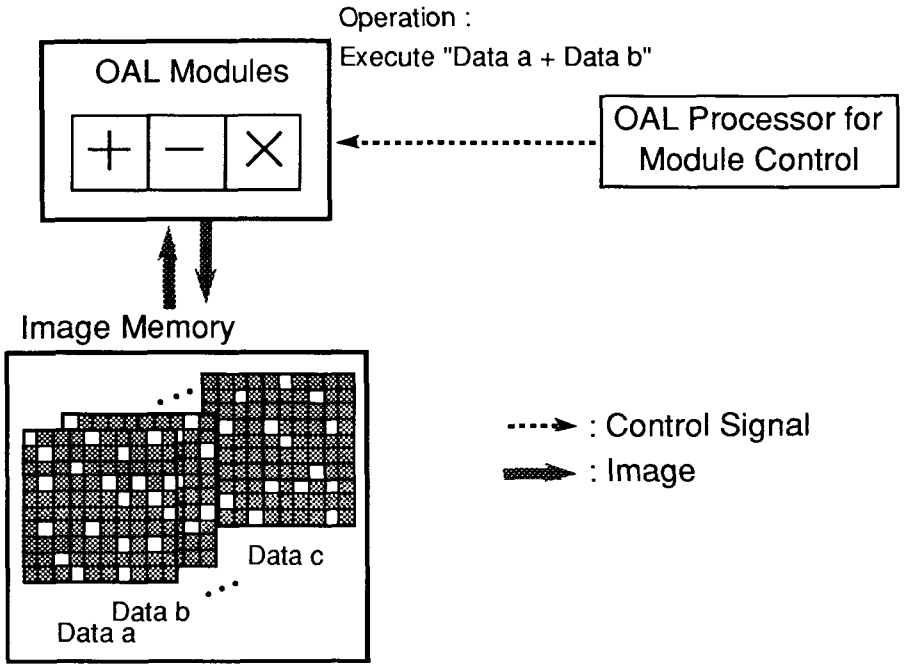


Fig.3.18　専用光演算モジュールを用いたデータフロー型処理システムの構成

演算時にトークンを画像名で受け取り，画像データを画像メモリから読みだす．専用光演算モジュールを用いたデータフロー型処理システムの構成を Fig.3.18 に示す．

しかし，この方法にはいくつかの問題点がある．まず，データフロー型処理の結果から，演算開始命令，トークンに乗っている画像名などを各モジュールに効率よく伝える必要がある．この解決法として，データフロー型処理の出力画像を光検出器で読み取り電気信号に変換し，各モジュールに制御信号を送ることが考えられる．また，各モジュールは，演算時にトークンを画像名で受け取り，画像データを画像メモリから読みだす．各モジュールが同時に演算を行うと，画像メモリへのアクセスが混雑または衝突する問題がある．この解決法としては，データフローグラフの接続に合わせてモジュール間を接続する方法が考えられる．しかし，この場合，効率のよいモジュール間の可変接続法が必要である．これには，画像クロスバスイッチ [21]，バンヤンネットワーク [53] などが利用できる．

また，本研究では，ハードウェアに厳しい要求をする処理を専用光演算モジュールで処理することにより，光アレイロジックプロセッサの処理を軽減する方法も検討した（第6章参照）．データフロー型処理は，この方法において専用光演算モジュール間接続の並列制御法として利用できる．

3.7　結言

本章では，光演算システムへの応用を目指して，光アレイロジックの並列性をトークンの並列転送に有効利用したデータフロー型処理の実現方法を検討した．その結果，光アレイロジックにより，データフローグラフによらず一定の処理速度でトークン転送を

実行できることがわかった．演算ノードの処理をMIMD方式で並列に実行するためには，トークンにデータの代わりに画像名を乗せ，各演算ノードに専用光演算モジュールを割り当てる方法が有望である．しかし，この方法の実現には，データフロー型処理によるトークン伝播結果を効率的に伝える方法と，各モジュールから画像メモリへのアクセスの混雑・衝突を避ける方法の開発が必要である．

第4章　光アレイロジックによるデータベース処理

4.1　緒言

最近のコンピュータの応用では，大容量情報処理の占める割合が大きくなり，その高速処理技術の開発が重要になっている．光アレイロジックは，2次元配列データに対してSIMD方式の演算を行うため，大容量データの一括処理に適している．本章では，光アレイロジックを大容量データ処理に応用するための体系的な手順の検討を行う．その基本的な考え方は，データの並列複製と，複製したデータに対する並列演算である．すなわち，目標となる処理に用いられるデータを複製して，画像上に配置し，SIMD方式の演算で一括して処理する．この手順では，データの複製が重要な演算となるが，これには，光アレイロジックの基本的パターン操作であるパターン展開（1.5.1項）を用いる．

大容量データ処理の応用としては，データベース処理 [10, 54, 55] をとりあげる．データベース処理は，所望のデータを大容量データの中から効率よく取り出す処理である．その処理は並列データ検索と比較が基本となる．どちらの方法も簡単なSIMD方式の演算で効率よく実行できるため，並列光演算に適している [56, 57]．本章では，光アレイロジックによる関係データベース [58] の基本演算についてプログラムを開発し，その処理能力について評価する．

4.2節では，光アレイロジックを大容量データ処理に適用する場合に有用な処理手法を整理する．4.3節では，現在データベース処理の主流となっている関係データベースの基本演算について概説し，その光アレイロジックによる実現方法を述べる．4.4節では，光アレイロジックによるデータベース処理の処理効率を評価し，データサイズとハードウェア資源の関係を求める．その結果に基づき，ハードウェア資源に対する要求を軽減する方法として，特定演算に対する専用光演算モジュール化を検討する．

4.2　大容量データ処理手法

4.2.1　基本方針

SIMD方式の並列処理においては，データを複製してそれらを並列に処理することにより，効率のよい処理が行える．光アレイロジックにこの処理方法を応用する場合は，データを画素パターンに符号化し，光アレイロジックの論理演算を利用して処理する．本章では，データの複製を画像内に多数作成する技術であるパターン展開を用いて，効率の良い大容量データ処理を実現する方法を検討する．この方法では，1.5.1項で述べたテンプレートマッチングとパターン展開の他に，データベース処理でよく用いられる大小比較とソーティングが基本演算と考えられる [55]．そこで，以下にパターン展開を利用して大小比較とソーティングを実現する方法について述べる．

4.2.2　大小比較

二つのデータの大小比較は，1）2データの組で値が異なるビットの検出，2）検出

した最上位ビットの大小比較により実行できる．比較する2データを s ビットの整数 $X = x^{s-1}x^{s-2}\cdots x^0$，$Y = y^{s-1}y^{s-2}\cdots y^0$ とすると，ステップ1，2の論理式は，それぞれ次のようになる．

$$z^i = x^i \oplus y^i \quad (i = 0, 1, \bullet\bullet\bullet, s-1)\ , \tag{4-1}$$

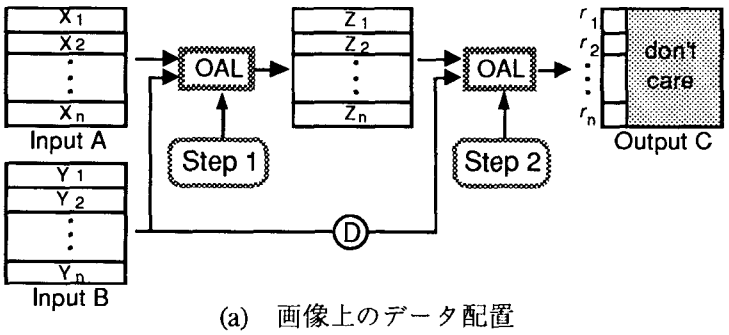
$$r = z^{s-1}y^{s-1} + \sum_{j=0}^{s-2} z^j y^j \left(\prod_{i=j+1}^{s-1} \bar{z}^i \right) . \tag{4-2}$$

ここで，$Z = z^{s-1}z^{s-2}\cdots z^0$ はステップ1の結果を表す s ビットの整数，r はステップ2の結果を表すビットである．(4-2) 式から，$X < Y$ の時 $r = 1$ が得られる．

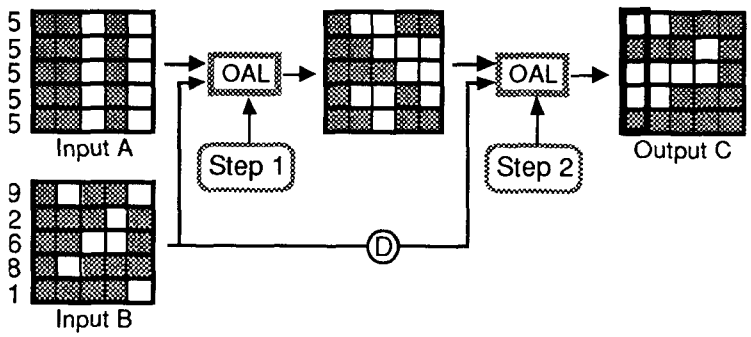
大小比較を光アレイロジックで実現するためには，データの2進数を2値画素パターンで表現し，Fig.4.1(a) のように画像上に配置する．この場合，ステップ1，2のカーネル式はそれぞれ，

$$[UU]\ , \tag{4-3}$$

$$[11] + \sum_{j=1}^{s-1} [11]_{0,j} \prod_{i=0}^{j-1} [0.]_{0,i}\ , \tag{4-4}$$



(a)　画像上のデータ配置



(b)　画像 InputB から5より大きいデータを検出する例

Fig.4.1　大小比較の処理の流れ

となる．この2ステップを光アレイロジックにより実行すると，比較条件を満たす領域の左端の画素 r_k に画素値1が得られる．領域の左端以外の画素は無視する．Fig.4.1(b) に，Y>5 を満たすデータを画像 InputB から検出する例を示す．ここでは，選択条件の5を画像 InputA 上に展開している．以上の手順により，画像 InputB 上の全データに対する並列大小比較が実現できる．

処理内容によっては，1枚の画像上の2データに対する大小比較が必要な場合がある．この演算を行うには，データを配置した画像の他に，比較すべき2データの位置を示す目印を配置した画像を用意する．これらの画像を用いて並列大小比較を行うためには，上記の方法とは少し異なるアルゴリズムを用いる．すなわち，ステップ1，2において，(4-1), (4-2) 式の代わりに以下の式を用いる．

$$p^i = x^i \overline{y^i} \qquad (i = 0, 1, \cdots, s-1), \tag{4-5}$$

$$q^i = \overline{x^i} y^i \qquad (i = 0, 1, \cdots, s-1), \tag{4-6}$$

$$r = \overline{p^{s-1}} q^{s-1} + \sum_{j=0}^{s-2} \overline{p^j} q^j \left(\prod_{i=1}^{s-1} \overline{p^i} q^i \right), \tag{4-7}$$

(4-7) 式の結果として，X < Y の時 $r = 1$ が得られる．

光アレイロジックによる，上式に基づく大小比較の実現方法を Fig.4.2 に示す．比較する2データを画像 InputA 上に隣接させて配置する．比較する2データを示す特定のパターンを画像 InputB 上の対応する位置に配置する．このパターンで画素値1の位置にあるデータと，その1画素下のデータが比較されることになる．ステップ1では，(4-5), (4-6) 式の演算を次のカーネル式で同時に実行する．

$$\begin{bmatrix} \underline{11} \\ 00 \end{bmatrix} + \begin{bmatrix} 01 \\ \underline{10} \end{bmatrix}. \tag{4-8}$$

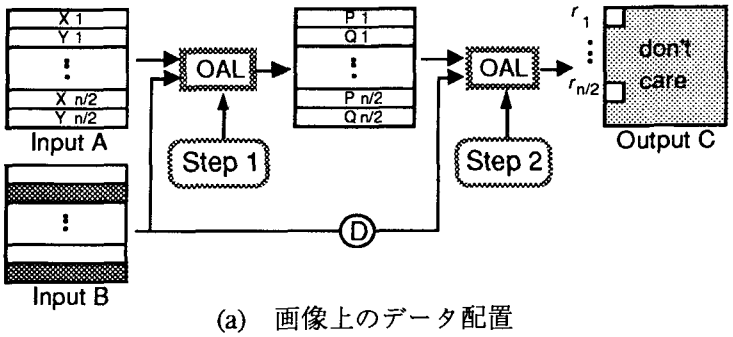
ステップ2のカーネル式は，(4-7) 式より次のようになる．

$$\begin{bmatrix} 01 \\ \underline{10} \end{bmatrix} + \sum_{j=1}^{s-1} \begin{bmatrix} \underline{01} \\ 10 \end{bmatrix}_{0,j} \prod_{i=0}^{j-1} \begin{bmatrix} 01 \\ \underline{00} \end{bmatrix}_{0,i}. \tag{4-9}$$

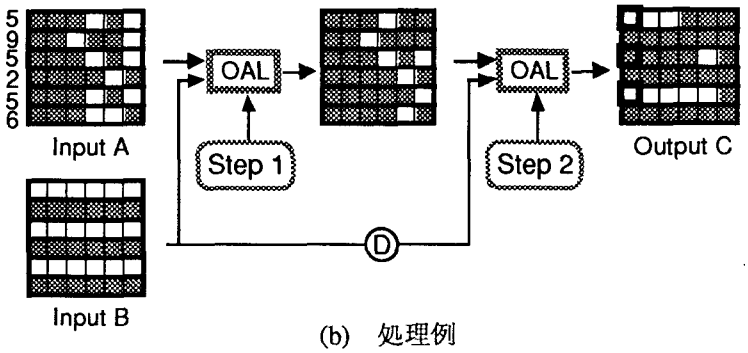
以上の手順により，大小比較の結果が画像 OutputC 上の左端の画素 r_k に得られる．Fig.4.2(b) の処理例のように，X < Y の時 $r_k = 1$ が得られる．

4.2.3　ソーティング

ソーティングは，2データの大小比較と置換処理で構成される．各ステージにおける比較・置換処理は比較的簡単な SIMD 方式の並列処理で実行できるため，光アレイロジ



(a)　画像上のデータ配置



(b)　処理例

Fig.4.2　1枚の画像上のデータに対する大小比較の処理の流れ

ックに適している．比較・置換の方法により種々のソートアルゴリズムが提案されているが [11, 59-61]，ここでは，SIMD 方式の並列処理に適した奇偶置換ソート [60] と奇偶マージソート [61] について検討する．

(a)　奇偶置換ソート

奇偶置換ソートは，Fig.4.3 のように比較・置換を行うデータの組をステージごとにずらせていくアルゴリズムである．データ数 n に対して，ステージ数は最大 n 段必要である．これを光アレイロジックで実行する手順を Fig.4.4 に示す．2データの大小比較は，Fig.4.2 の方法と同様である．すなわち，Fig.4.4(a) のように，ソートする s ビットのデータ $X_i^{(1)}$ $(i = 1, 2, \cdots, n)$ を配置した画像の他に，比較すべき2データの位置を示す目印を配置した画像を用意する．この画像で画素値1の位置にあるデータと，その1画素下のデータが比較される．以下，これらの画像をそれぞれ入力画像 A, B とする．

大小比較におけるステップ1，2はそれぞれ (4-8), (4-9) 式により実行する．その結果，Fig.4.4(a) のようになる．ただし，ステップ1における $X_{2k-1}^{(1)}$ と $X_{2k}^{(1)}$ の比較結果をそ

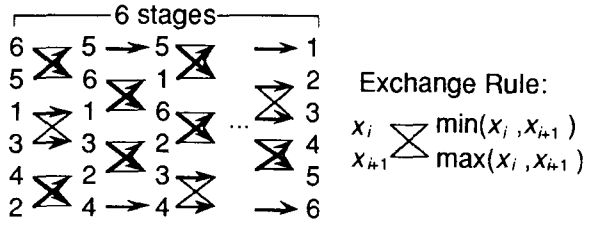
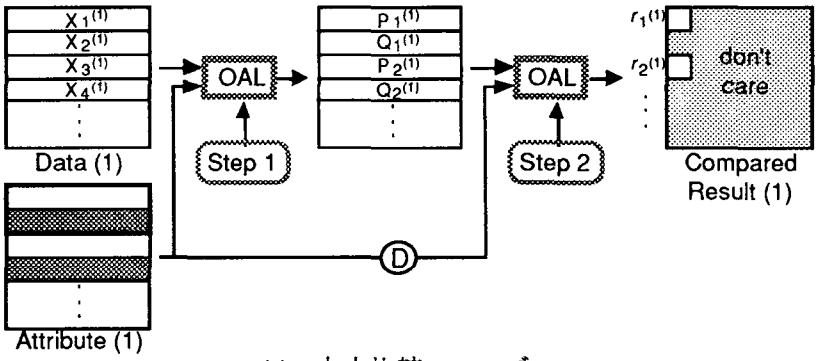


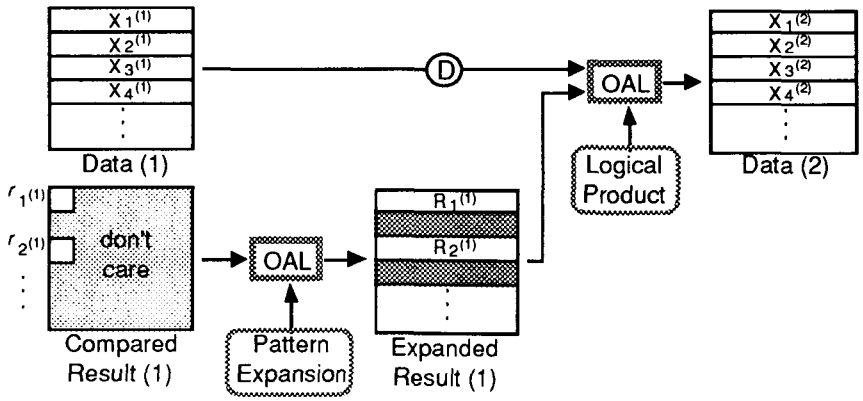
Fig.4.3　奇偶置換ソート

れぞれ $P_k^{(1)}$ と $Q_k^{(1)}$，ステップ2における比較結果を $r_k^{(1)}$ と表記する．また，ソートの第 m ステージで用いる画像を，画像名の後ろに (m) を付加して表記する．入力画像 A, B をそれぞれ画像 Data (1), Attribute (1) とすることにより，比較結果を表す画素 $r_k^{(1)}$ を，画像 Compared Result (1) 上のデータの左端の画素に得る．$r_k^{(1)}$ の画素値は，比較条件を満たす場合は1，それ以外は0となる．画像 Compared Result (1) 上の他の画素は無視する．

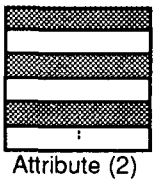
比較結果に応じて，データ $X_i^{(1)}$ を置換する．Fig.4.4(b) は，置換の処理手順を示す．まず，画像 Compared Result (1) の画素 $r_k^{(1)}$ を横方向に展開し，画素パターン $R_k^{(1)}$ を生成する．それらを画像 Expanded Result (1) 上にセットし，置換すべき2データの位置を示す画素パターンとして使用する．画像 Data (1), Expanded Result (1) をそれぞれ入力画像 A, B とすると，置換を行うカーネル式は，条件付きシフト演算



(a)　大小比較フェーズ



(b)　置換フェーズ



(c)　次のステージで使用するデータ位置指定用画像

Fig.4.4　光アレイロジックによる奇偶置換ソートの処理手順

$$\begin{bmatrix}11\\ \underline{.0}\end{bmatrix}+\begin{bmatrix}\underline{.1}\\ 10\end{bmatrix}+\begin{bmatrix}.0\\ \underline{10}\end{bmatrix}+\begin{bmatrix}\underline{10}\\ .0\end{bmatrix}, \tag{4-10}$$

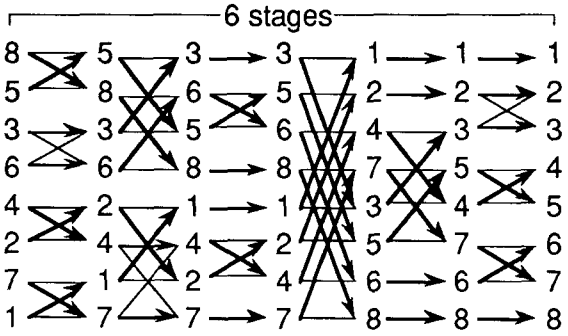
となる．この式を実行すれば，画像 Data (2) 上にソートの1ステージを完了したデータ $X_i^{(2)}$ が得られる．

次のステージでは，Fig.4.3 に示したように，比較するデータの組を一つ下にずらさなければならない．これは，画像 Attribute (1) を1画素下にシフトさせた画像 Attribute (2) により指定できる（Fig.4.4(c)）．次のステージでは，画像 Data (2) と画像 Attribute (2) とを用いて同様の処理を行う．以下，画像 Attribute (1) と Attribute (2) を交互に使用しながら上記処理を n 回繰り返すことにより，並列ソーティングが完了する．

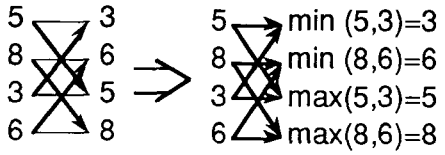
(b)　奇偶マージソート

奇偶マージソートは，Fig.4.5(a) のように，比較・置換するデータの組合せをステージごとに変えることにより，より少ないステージ数（$\log_2 n\,(\log_2 n+1)/2$ 段）でソートを完了できる．しかしこの方法は，比較する2データが隣接してない場合があるため，通常 Fig.4.5(b) のように，あらかじめ比較すべき2データの複写・転送を行い，比較演算によって必要なデータのみを残す手順を用いる．

データの符号化は Fig.4.6 のように行う．データの2進数表現を2値画素パターンで表し，さらにその横に比較時のデータ転送領域としてデータと同じ大きさの画像領域を用意する．このソートでは比較するデータの組合せが複雑で，しかもステージごとに異なるため，アトリビュートプレーンは1ステージごとに4枚ずつ作成する．Fig.4.6 は，第1～3ステージのアトリビュートプレーンを示す．ここでは，データごとに大小比較を行うかどうかなどを指定する．



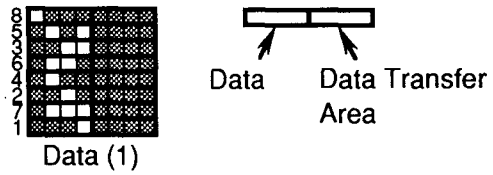
(a)　ソート過程



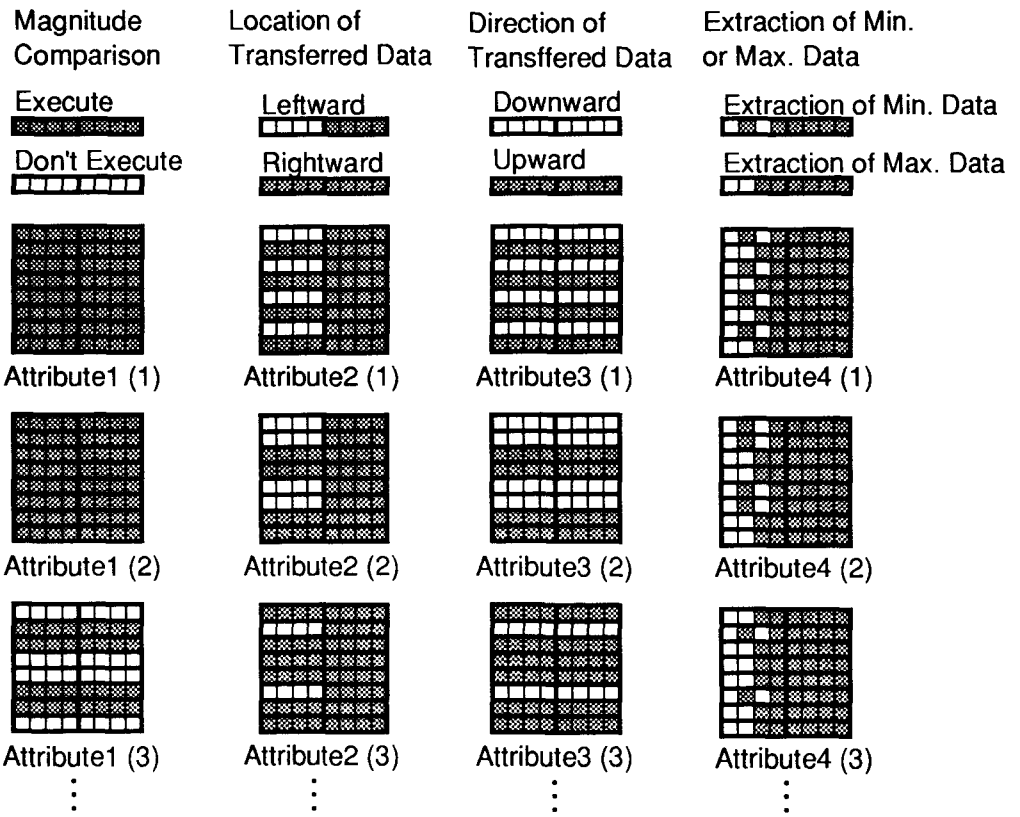
(b)　比較・置換処理の手順

Fig.4.5　奇偶マージソート

Fig.4.7 にソーティングの処理手順を示す．まず Fig.4.7(a) のように，4枚のアトリビュートプレーンを用いてデータ転送を行い，比較すべき2データを横に隣接させたパターンを画像 Data (1') 上にセットする．これは，シフトと論理和の演算で行う．次に，データの大小比較を Fig.4.2 と同様の方法によりに行う．ただし，ステップ2においては，大小比較 $\min(X_{2k-1}^{(1)}, X_{2k}^{(1)})$，$\max(X_{2k-1}^{(1)}, X_{2k}^{(1)})$ の結果をそれぞれ $r_{k\min}^{(1)}, r_{k\max}^{(1)}$ として出力する．その結果を Fig.4.7(b) のようにパターン展開し，置換処理のアトリビュートプレーンを作成する．この画像を参照してどちらか一方のデータを出力することにより，データの置換に相当する処理が実行できる．以下，各ステージで用意したアトリビュートプレーンを用いて，上述の処理を繰り返す．

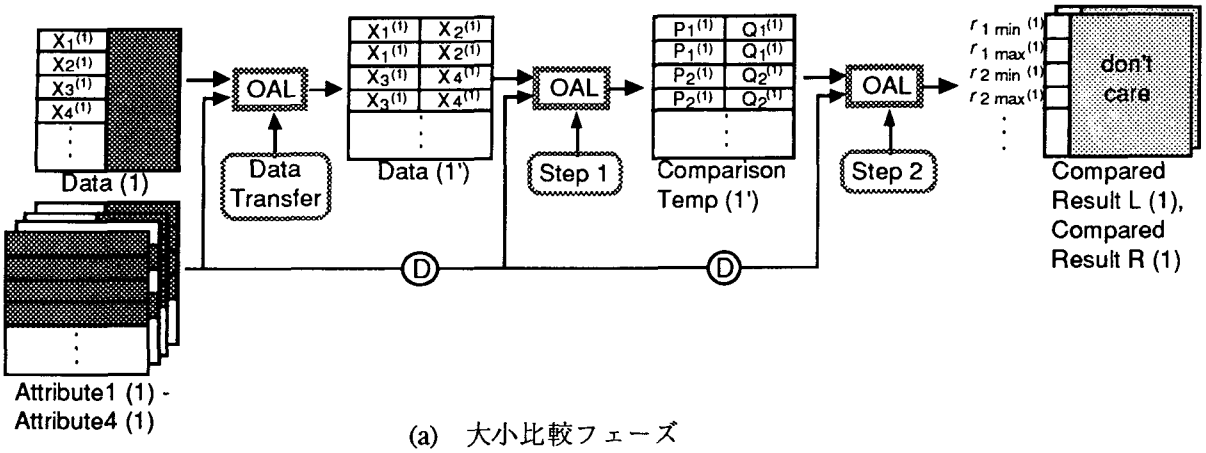


(a)　データプレーン

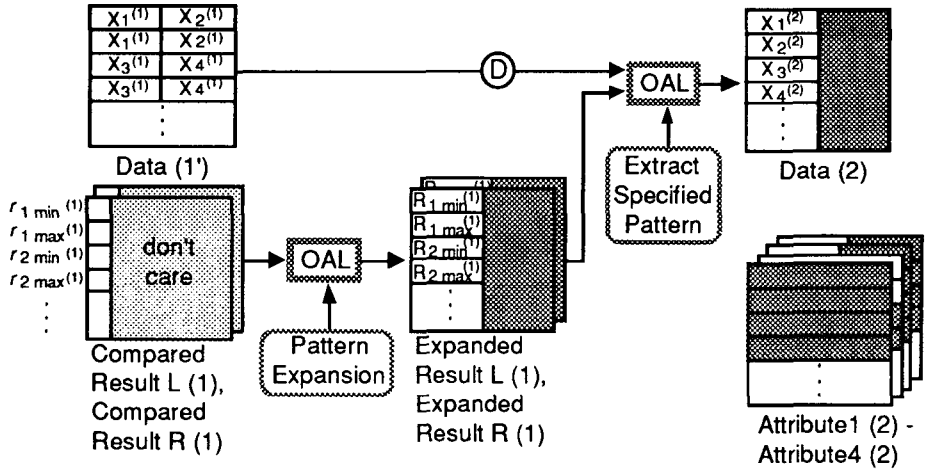


(b)　アトリビュートプレーン

Fig.4.6　光アレイロジックによる奇偶マージソート・データの符号化



(a)　大小比較フェーズ



(b)　置換フェーズ

Fig.4.7　光アレイロジックによる奇偶マージソートの処理手順

4.3　データベース処理

4.3.1　関係モデル

本研究では，データベース処理のなかでも，現在主流となっている関係モデル [58] を対象とする．関係モデルでは，情報は Fig.4.8 に示すような関係と呼ばれる表の集合で表される．表の各行，列はそれぞれタプル (Tuple)，属性 (Attribute) と呼ばれる．関係に対して，Fig.4.8 のように選択 (Selection)，射影 (Projection)，準結合 (Semijoin) という関係演算が定義されている．選択は，全タプルから特定の属性を持つものを選び出す．射影は，指定した属性のみを残し，さらに重複したタプルを除去する．準結合は，二つの関係の中で指定した属性の全タプル同士を比較し，条件を満たすタプルを，一方の関係から選択する．準結合において，二つの関係それぞれから条件を満たすタプルを選択し，条件を満たすタプル同士を連結することにより，二つの関係を連結して一つの関係にまとめることができる．この演算を結合と呼ぶ．しかし，この演算は一般に難しいため，本研究では検討していない．この他，データの整理や分類のために，特定の属性の全タプルに対するソーティングもよく使用される．

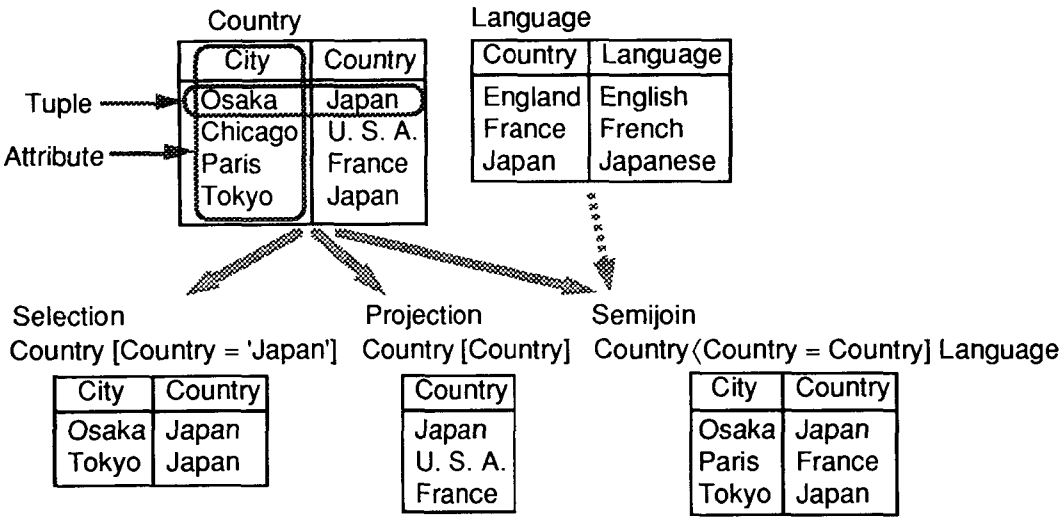


Fig.4.8　関係と関係演算

以上の関係演算を組み合わせることにより，関係の集合から必要な情報を効率的に取り出すことができる．これらの演算は，全タプルに対する検索・比較演算が基本となるため，光アレイロジックによるSIMD方式の並列処理が有効である．

4.3.2　データベース処理の基本演算の実現方法

データベース処理は，画素パターンに対するテンプレートマッチングや大小比較などの簡単な演算で実行できる．ここでは，光アレイロジックによるデータベース処理の基本演算の実現方法を説明する．

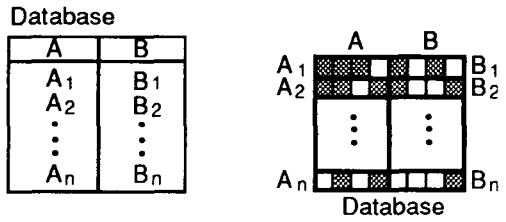
まず，ある関係を1枚の画像で表す．画像への変換は，Fig.4.9に示すように，各データをそれぞれ2値画素パターン化して行う．Fig.4.10は属性Aについての選択演算の処理手順を示す．選択条件を表すパターンCを画像Condition上に置き，列方向に展開する．次に，展開画像とデータベース画像Database上のパターン A_i $(k = 1, 2, \cdots, n)$ との間でデータ比較を行い，比較結果を画像Comparison Resultに出力する．この画像上のパターン R_i $(i = 1, 2, \cdots, n)$ の全画素値は，条件を満たしたタプルでは1，それ以外では0をとる．最後に，この画像と画像Databaseの論理積をとれば所望のタプルを選択できる．

データベース処理の他の基本演算も，同様の手法により光アレイロジックで実現できる．Fig.4.11に属性Aについての射影演算の処理アルゴリズムを示す．射影演算では，指定した属性の一番上のパターンから順に重複したパターンの検索と消去を行う．画像Database (1) 内の，画像Condition Area (1) で指定されたパターンを取り出し，検索対象となる領域に展開し，画像Database (1) とのマッチングで重複したパターンを検出する．画像Matching Result (1) 上のパターン $R_i^{(1)}$ $(i = 1, 2, \cdots, n)$ の全画素値は，$A_i^{(1)}$ が重複したパターンである場合には1，それ以外では0とする．この画像を用いて，重複したパターンを，反転と論理和を用いて画像Database (1) から消去することにより，結果画像Database (2) が得られる．以下，同様の処理を，画像Condition Area (1) の

画素を1画素ずつ下にシフトさせた画像 Condition Area (2), (3), ···, (10) と，画像 Database (2), (3), ···, (10) に対して繰り返すことにより，射影演算の結果が得られる．

準結合演算では，二つの関係を表す2枚の画像のうち，結果を出力しない関係の画像から，指定した属性のパターンを上から順に一つずつ取り出す．それぞれのパターンについて，逐次的にもう一方の関係の指定した属性と選択演算を行い，その結果の論理和をとることにより，準結合演算が実現できる．

上記処理の光アレイロジックプログラムを作成し，実現方法の正当性を確認した．Fig.4.12 に，属性 Country = 'Japan' を条件とする選択演算のシミュレーション結果を示す．まず，Table 4.1 のデータベースを，Fig.4.12(a) のように画素パターンで表す．各画素の左端の数字は，Table 4.1 のタプル識別番号に対応している．画像 Condition に対応する画像は Fig.4.12(b) である．この2枚の画像を用いて，Fig.4.12(c) の選択演算の結果が得られた．得られた画素パターンのタプル識別番号を参照することにより，正しい答が得られていることがわかる．なお，本手法のカーネル式は Appendix A.4－A.6 に，OALL プログラムは Appendix B.5－B.7 に示す．



(a)　関係の例　　(b)　符号化画像

Fig.4.9　関係の符号化

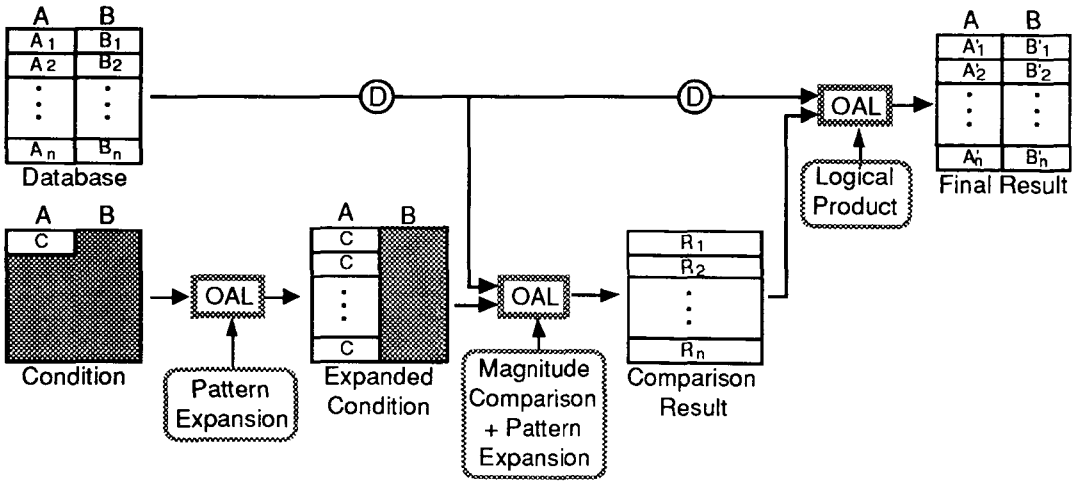


Fig.4.10　光アレイロジックによる選択演算の処理手順

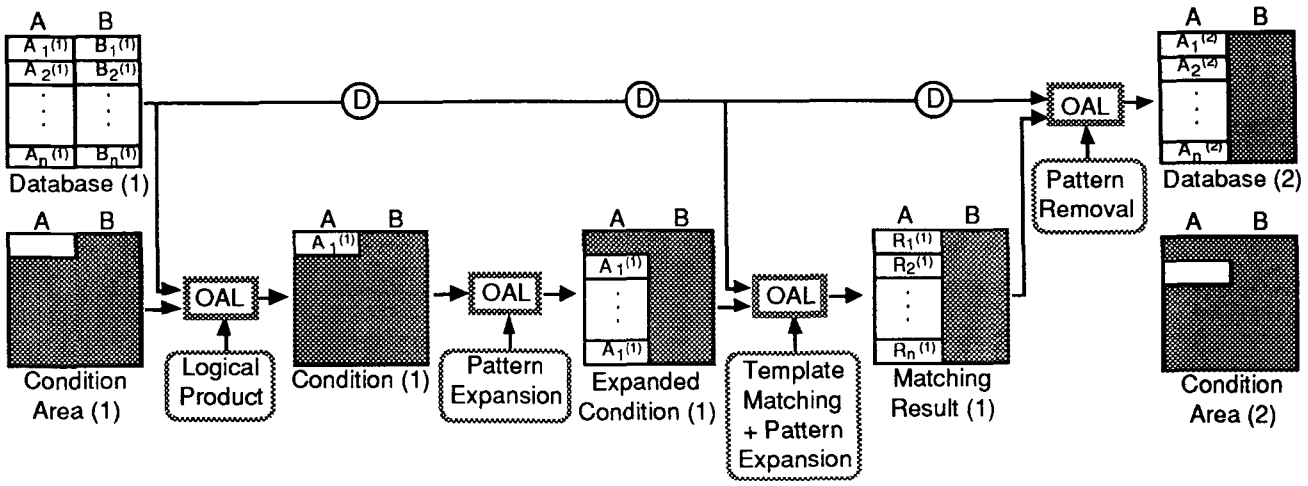
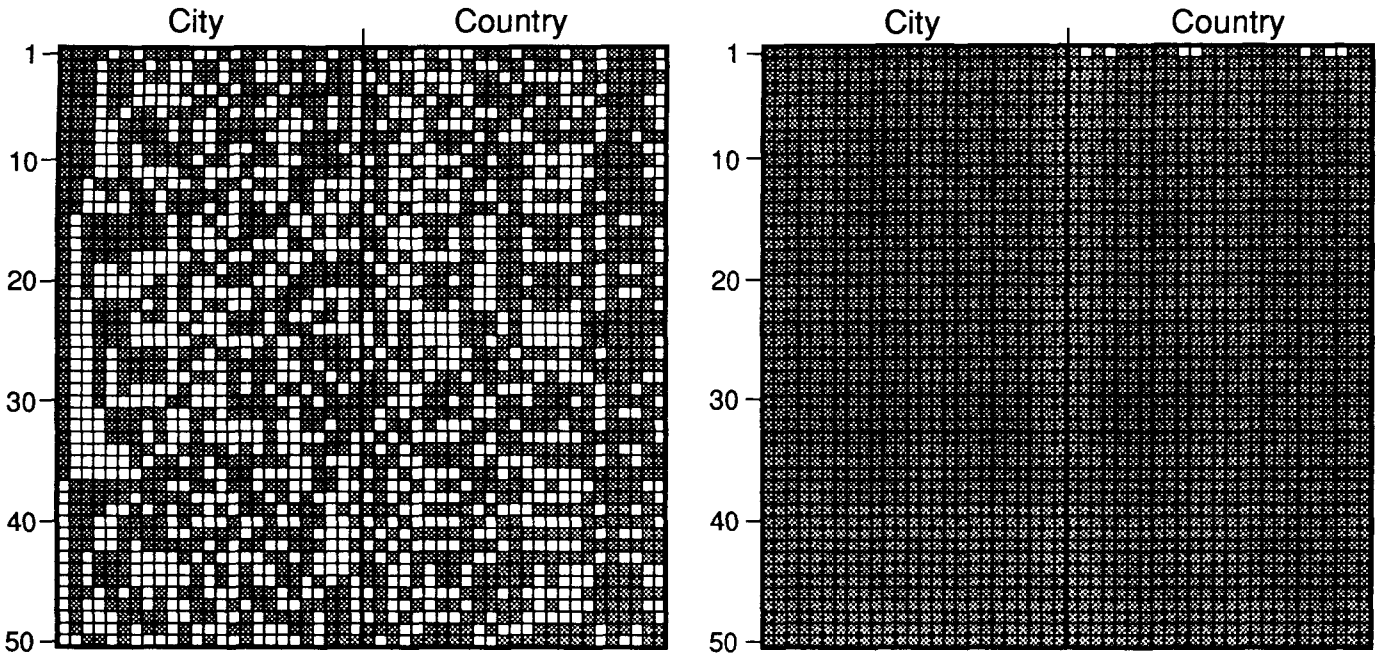


Fig.4.11　光アレイロジックによる射影演算の処理手順

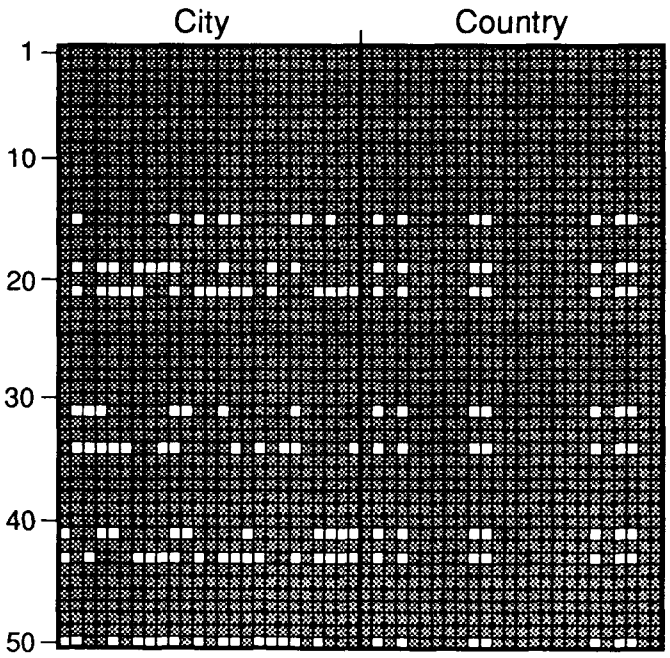
Table 4.1　関係の例

Country

No.	City	Country	No.	City	Country
1	Ahmedabad	India	26	Madras	India
2	Berlin	Germany	27	Manchester	U.K.
3	Birmingham	U.K.	28	Marseille	France
4	Bombay	India	29	Montreal	Canada
5	Bordeaux	France	30	München	Germany
6	Bremen	Germany	31	Nara	Japan
7	Calcutta	India	32	New Delhi	India
8	Calgary	Canada	33	New York	U.S.A.
9	Cherbourg	France	34	Osaka	Japan
10	Chicago	U.S.A.	35	Ottawa	Canada
11	Cognac	France	36	Oxford	U.K.
12	Detroit	U.S.A.	37	Paris	France
13	Frankfort	Germany	38	Phoenix	U.S.A.
14	Greenwich	U.K.	39	Quebec	Canada
15	Hakata	Japan	40	San Francisco	U.S.A.
16	Hamburg	Germany	41	Sapporo	Japan
17	Hannover	Germany	42	Sheffield	U.K.
18	Houston	U.S.A.	43	Tokyo	Japan
19	Kobe	Japan	44	Toulon	France
20	Köln	Germany	45	Toulouse	France
21	Kyoto	Japan	46	Rugby	U.K.
22	Leipzig	Germany	47	Versailles	France
23	Liverpool	U.K.	48	Washington	U.S.A.
24	London	U.K.	49	Winnipeg	Canada
25	Los Angeles	U.S.A.	50	Yokohama	Japan



(a)　関係を表す画像　　(b)　選択条件を表す画像



(c)　選択演算の結果

Fig.4.12　光アレイロジックによる選択演算のシミュレーション結果

4.4　処理効率評価

4.4.1　評価結果

光アレイロジックによるソーティングとデータベース処理基本演算の処理効率を評価した結果をそれぞれ Table 4.2, Table 4.3 に示す．Table 4.2 において，N, B はそれぞれデータ数，データのビット数を表す．S は奇偶マージソートにおけるステージ数を表す．Table 4.3 において，M, N, B はそれぞれ関係データベースの属性数，タプル数，データ

のビット数を表す．評価法は2．6．1項の方法と同様である．

ソーティングの評価より，システムの処理時間を決定するパラメータである符号化回数，相関回数が，それぞれ各ソートにおいてステージ当たり一定，B のみに比例の関係にあることがわかる．逐次処理では，ステージ当たりの処理時間がデータ数に比例するので，光アレイロジックの並列性がソーティングに有効利用されていることがわかる．さらに，二つのソートを比べると，奇偶マージソートのほうが，符号化回数，相関回数が少ないため，処理速度が速いことがわかる．必要なカーネルの大きさ，必要画素数などのハードウェア資源に対する要求は，奇偶マージソートのほうが厳しい．特にデータ転送時のシフト量が大きいため，カーネルサイズに対する要求が厳しい．

データベース処理の評価より，システムの処理時間を決定するパラメータである符号化回数と相関回数は，選択演算では N に対して一定であり，他の演算では N に比例することがわかる．これらを逐次処理で行うと，処理時間がそれぞれ N, N^2 に比例するので，これらの演算が効率よく行われていることがわかる．しかし，必要なカーネルの大きさ，必要画素数などのハードウェア資源に対する要求は，データベースが大容量になるにしたがって厳しくなる．

ここで，本手法の処理速度を試算する．試算方法は，2．6．1項のものと同様である．(2-4) 式に Table 4.2 の相関演算回数を代入することにより，ソーティングの処理速度は次式で表せる．

1．奇偶置換ソート

$$V = \frac{Ni^2}{17(23Ni + 7)Tc} \quad . \tag{4-8}$$

2．奇偶マージソート

$$V = \frac{Ni^2}{1232 \log_2 Ni \ (\log_2 Ni + 1)Tc} \quad . \tag{4-9}$$

ただし，Ni は画像サイズ（画像の1辺の画素数）である．また，(2-4) 式から (4-8) 式の導出には，$Np = N^2 / (B + 1)$, $N = Ni$, $B = 16$ を用いており，(4-9) 式の導出には，$Np = N^2 / 32$, $N = Ni$, $B = 16$ を用いている．同様に，Table 4.3 を参照して，データベース処理の基本演算の処理速度は次式のようになる．

1．選択

$$V \geq \frac{Ni}{26Tc} \quad . \tag{4-10}$$

2．射影

$$V \geq \frac{Ni}{(7Ni + 5)Tc} \quad . \tag{4-11}$$

３．準結合

$$V \geq \frac{Ni}{(25Ni + 7)Tc} \quad . \tag{4-12}$$

４．ソーティング

$$V \geq \frac{Ni}{(25Ni + 10)Tc} \quad . \tag{4-13}$$

Table 4.2　光アレイロジックによるソーティングの評価

		Odd-even Transposition Sort	Odd-even Merge Sort
Number of Encoding		$5N + 4$	$12S$
Number of Correlation		$(B + 7)N + 8$	$(4B + 13)S$
Kernel Size [/ Kernel Unit]	(x)	3	$N + 1$
	(y)	$2B - 1$	$4B - 1$
Image Size [/ Pixel]	(x)	N	N
	(y)	$B + 1$	$2B$

N : number of data
B : number of bit for a data
S : stage number for odd-even merge sort ($= \log_2 N (\log_2 N + 1) / 2$)

Table 4.3　光アレイロジックによるデータベース処理の評価

		Selection	Projection	Semijoin	Sorting
Number of Encoding		≤ 11	$\leq 7N + 5$	$\leq 10N + 7$	$7N + 6$
Number of Correlation		$\leq S + 10$	$\leq 7N + 5$	$\leq (S + 9)N + 7$	$(S + 9)N + 10$
Kernel Size [/ Kernel Unit]	(x)	$2N - 1$	$2N - 1$	$2N - 1$	3
	(y)	$2MS - 1$	$\leq 2MS - S$	$2MS - 1$	$2MS - 1$
Image Size [/ Pixel]	(x)	N	N	N	N
	(y)	MS	MS	MS	MS

M : attribute number
N : tuple number
S : bit number of data

ただし，(2-4) 式から (4-10), (4-11) 式の導出には，$Np = N$, $N = Ni$ を用い，(4-12), (4-13) 式の導出には，$Np = N$, $S = 16$, $N = Ni$ を用いている．(4-10) - (4.13) 式の不等号は，本手法のプログラムがデータに応じて不要な処理を省くため，処理速度が変動することを示している．

Fig.4.13, 4.14 に，それぞれ奇偶置換ソート，奇偶マージソートの処理速度を，Fig.4.15 - 4.18 に，それぞれデータベース処理の選択，射影，準結合，ソーティングの処理速度をグラフで示す．ただし，データベース処理の処理速度は，それぞれ最小値で示している．

Fig.4.13, 4.14 からわかるように，奇偶置換ソート，奇偶マージソートでは，画素数が増加するに連れて処理速度が向上する．このことより，これらの手法が光アレイロジックの並列性を有効利用していることがわかる．なお，電子計算機の処理速度の例として，東京大学で開発されたソータを搭載した商用プロセッサ GREO [62] をグラフ上に併記している．GREO の横軸の値は，本手法により GREO と同等の並列性を実現するために必要な画素数である．このグラフと比較すると，奇偶置換ソート (Fig.4.13) では，H-OPALS のサイクル速度 $Tc = 10^{-7}$ 秒において，約 250 ×250 以上の画素数の画像を扱うと，GREO の処理速度を上回ることがわかる．奇偶マージソート (Fig.4.14) では，H-OPALS のサイクル速度 $Tc = 10^{-6}$ 秒でも，約 1000 ×1000 以上の画素数の画像を扱うと，GREO の処理速度を上回る．奇偶置換ソートと奇偶マージソートのグラフを比べると，奇偶マージソートのグラフほうが傾きが大きいため，画像サイズが大きくなるほど処理速度の増加率が大きいことを示している．

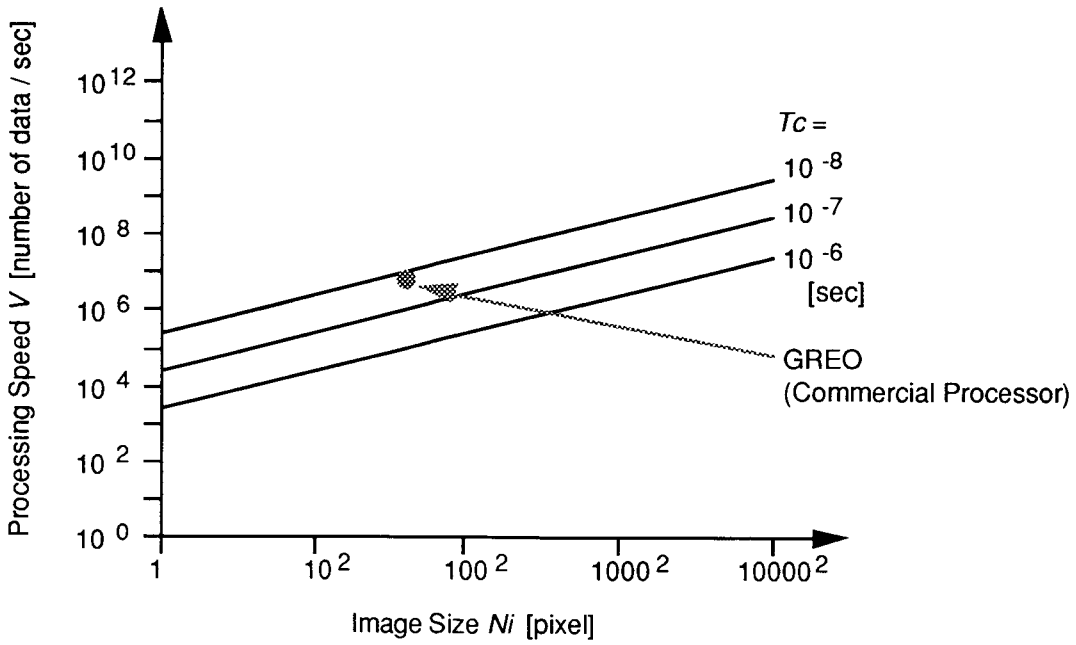


Fig.4.13　光アレイロジックによる奇偶置換ソートの処理速度

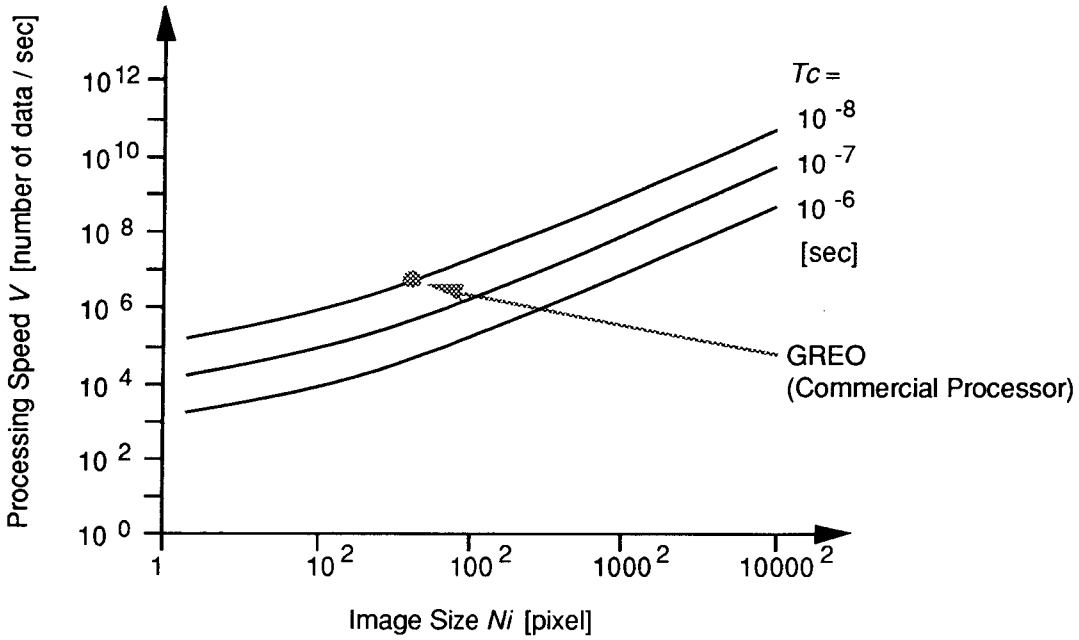


Fig.4.14　光アレイロジックによる奇偶マージソートの処理速度

データベース処理の選択演算 (Fig.4.15) では，現在最高速クラスのデータベースマシンにおける選択演算の処理速度 [63] を併記している．データベースマシンの横軸の値は，本手法により同等の並列性を実現するために必要な画素数である．これより，H-OPALS のサイクル速度 $Tc = 10^{-4}$ 秒で，約 100×100 以上の画素数の画像を扱うと，データベースマシンの処理速度を上回ることがわかる．選択演算は，光アレイロジックに非常に適した処理であると言える．

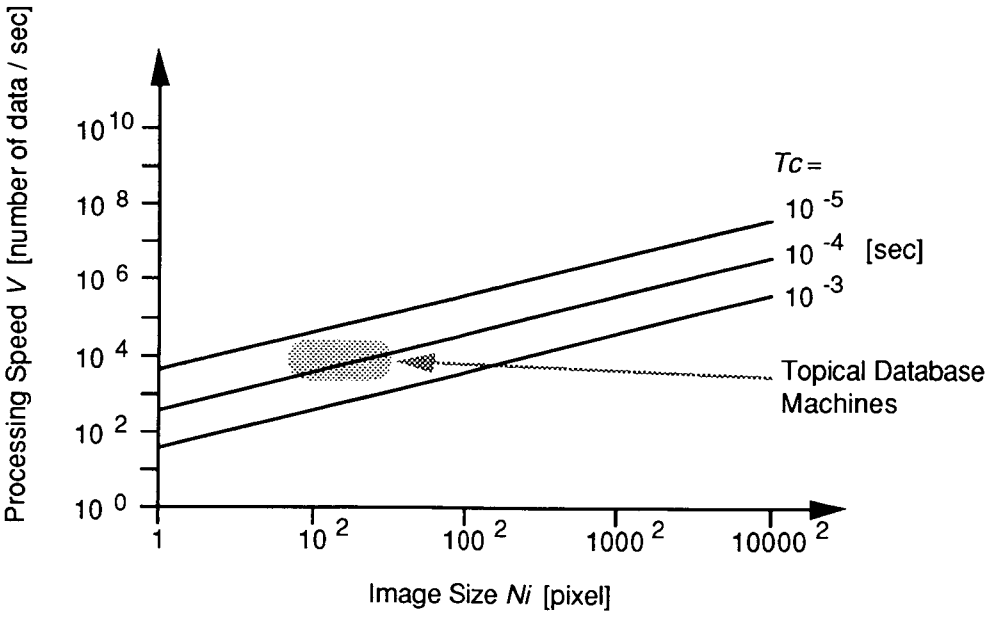


Fig.4.15　光アレイロジックによるデータベース処理（選択演算）の処理速度

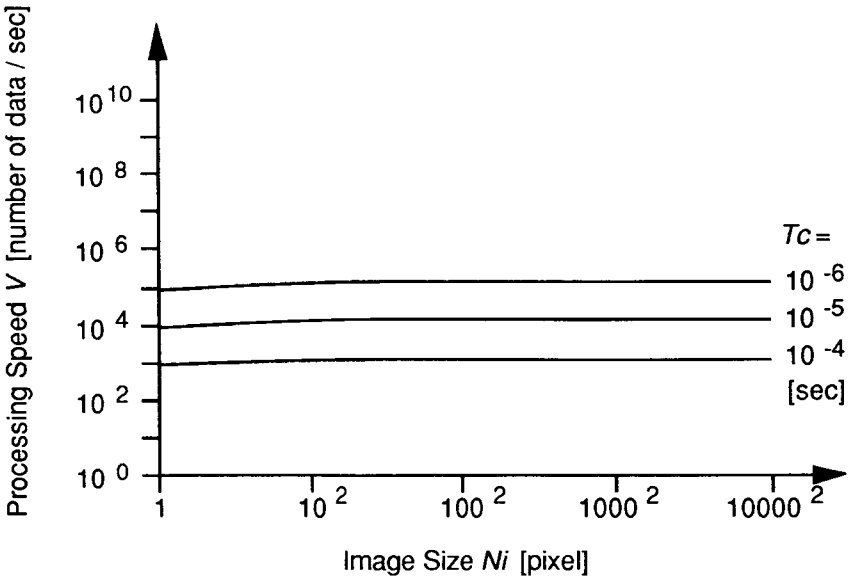


Fig.4.16　光アレイロジックによるデータベース処理（射影演算）の処理速度

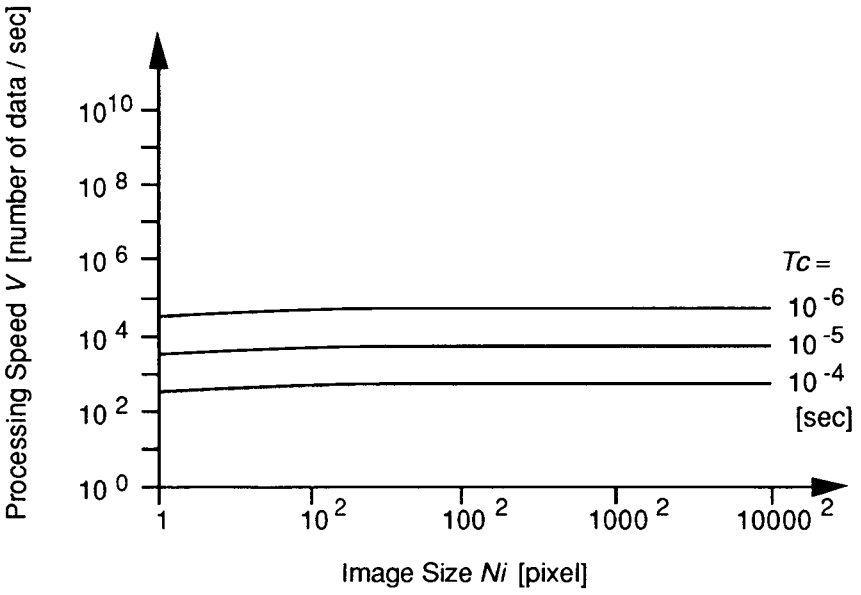


Fig.4.17　光アレイロジックによるデータベース処理（準結合演算）の処理速度

データベース処理の射影，準結合演算では，データを逐次的に調べる必要がある．この処理回数は，タプル数に比例する．しかし，x方向の画素数もタプル数に比例するため，画素数が増加すると逐次処理の回数が増加する．その結果，処理速度はほとんど増加しない．したがって，これらの演算は並列性の有効利用が困難であると言える．

データベース処理としての奇偶置換ソートの処理速度 (Fig.4.18) は，4.2.3項で述べた奇偶置換ソートの処理速度 (Fig.4.13) より遅くなっている．これは，データベース処理としてのソーティングが，単にソーティングのみを行う場合と異なり，画像上のデータ配置がソーティングに最適な形にすることができないためである．

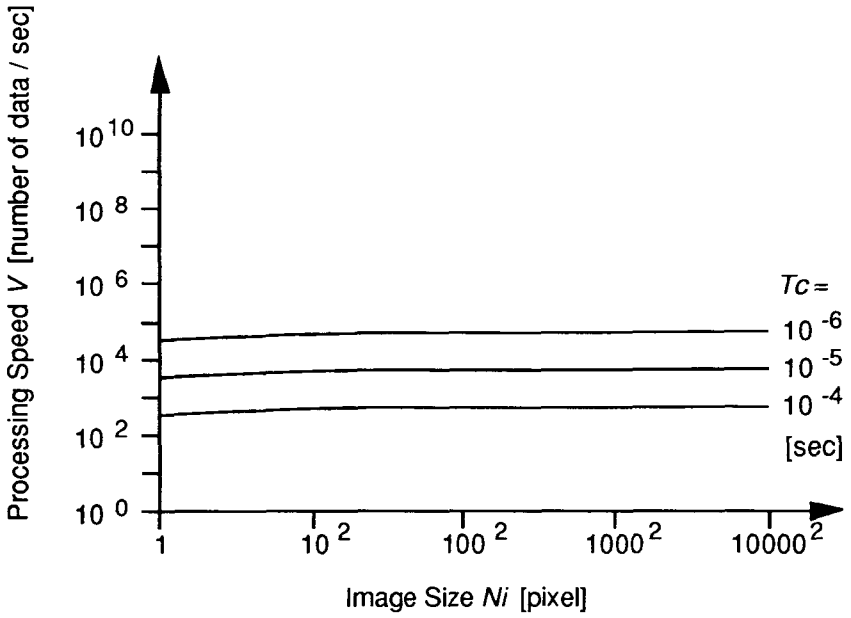


Fig.4.18　光アレイロジックによるデータベース処理（ソーティング）の処理速度

4.4.2　考察

4.4.1項で述べたように，本手法では，データベースが大容量になるにしたがって，必要なカーネルの大きさ，必要画素数などのハードウェア資源に対する要求が厳しくなることが問題となる．この解決法としては，ハードウェア資源への要求が厳しい処理を専用光演算モジュールで処理することにより，演算カーネルの最大シフト量を減少させる方法が考えられる．これは第6章で検討する．

データベース処理の出力画像では，データが画像上に散らばっているため，出力画像を用いてさらに処理を継続する場合は，画像の利用効率が悪くなる．そこで，得られた結果を集めて整列させる処理が必要となる．しかし，この処理は並列処理向きでないため，光アレイロジックでこの処理を行うと，処理効率が悪くなる．したがって，この処理に対して，専用の逐次処理モジュールを開発する必要があろう．

光アレイロジックによるデータベース処理では，全データに対する大小比較を基本とした選択演算が非常に効率的である．しかし，他の基本演算は，光アレイロジックの並列性をいかしきれていない．これらの演算では，データに対する逐次処理をなくせないことから，データベースマシンの研究でも問題になっている．現在はデータの記憶装置からの逐次転送に合わせてパイプライン処理を行う方法が有力と考えられている[10]．したがって，実用的なデータベース処理システムでは，光アレイロジックが特に有効であるデータ検索，大小比較のみを光アレイロジックで行い，他の処理はパイプライン的に行う方法がよいと考えられる．

データベース処理など大容量データ処理を光アレイロジックで行う場合は，2次記憶装置のデータ読み出し速度が重要な問題となる．2次記憶装置としては，画像内のデータを並列に読みだせるもの，例えばホログラフィック光連想メモリなどが有力である[64]．この他に，2次記憶装置，専用光演算モジュールを含めたシステム全体の構成法の検討が重要となる．これについては，第6章で述べる．

4.5　結言

光アレイロジックの大容量データ処理への応用として，データベース処理を検討した．まず，光アレイロジックの並列性を大容量データ処理に有効利用する手法として，パターン展開を利用した大小比較とソーティングの光アレイロジックプログラムを開発した．それらの演算を利用して，データベース処理の関係演算プログラムを開発し，その処理効率，必要なハードウェアについて評価した．その結果，光アレイロジックがデータベース処理に対して有用な技術となり得ることを確認した．実用的なシステムでは，光アレイロジックの並列性が特に有効なデータ検索と大小比較を光アレイロジックで行い，他の処理を電気的に行う方法が有力である．今後，2次記憶装置などシステム全体を考慮した処理方式の開発が重要である．

第５章　並列光演算システム用２次元仮想記憶機構

５.１　緒言

並列光コンピュータの演算能力を十分に引き出すためには，並列光演算原理の有する並列性を最大限に活用しうるソフトウェアとハードウェア構成が必要である．本研究において開発した並列演算技法では，基本的に並列処理可能なデータをできるだけ多く一枚の画像上に配置し，処理の並列性を引き出す考えが基本となっている．したがって，大容量データを一度に処理する必要がある処理分野ほど並列性を活用し得ると言える．しかし，光アレイロジックによる推論機構（第２章）などで，実世界の問題を処理するためには，一辺数千～数万の画素数が要求される．

残念ながら現在の並列光演算システムのハードウェア技術は，まだ原理的な機能の確認レベルにあり，多くの画素を同時に扱うことはできない [65]．また，たとえ将来，大規模並列光デバイスが実現できたとしても，それを利用したシステムは，大きさ，生産性，コスト，処理効率などの点で問題を持つ．これらのことより，並列光演算システムであまり多くの画素を扱うことは困難と考えられる．このようにソフトウェアとハードウェアの間には開発方針にギャップが存在する．

さらに，情報化社会においては，データの大容量化が急速に進んでいるため，将来，光コンピュータのソフトウェアに要求される処理画像の画素数には際限がないことが予想される．その場合，システムがいくら多くの画素数を扱えるようになっても，ソフトウェアの要求は満たせない．この問題に対する解決法は，ソフトウェア，ハードウェアそれぞれの側から考えられる．しかし，ソフトウェア開発の負担を軽減し，効率的な処理を行うためには，ハードウェアから解決するほうが望ましい．すなわち，さまざまな処理画素数を要求するソフトウェアに柔軟に対応し，しかも並列性を最大限に利用して効率のよい処理を行うハードウェアが要望される．

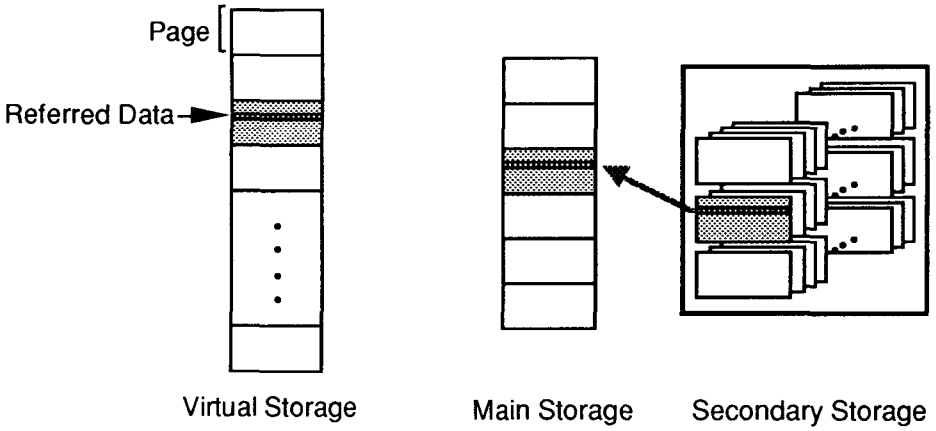
以上のような，ソフトウェア／ハードウェア間ギャップを埋め，さまざまな処理画素数に柔軟に対応するハードウェアの実現法として，処理画像をページに分割して処理する２次元仮想記憶機構を検討した．以下，５.２節では，２次元仮想記憶機構の概念について概説する．５.３節では，並列光演算システムのモデルとして考案した相関演算モデルとその２次元仮想記憶機構への適用について述べる．５.４節では，２次元仮想記憶機構の処理手順とその処理効率を，演算における最大シフト量により分類して検討する．５.５節では，パイプラインページ転送方式を用いた２次元仮想記憶機構の構成法について述べる．

５.２　２次元仮想記憶機構の概念

仮想記憶機構 [10] は，コンピュータにおける処理の自由度を拡張する手法の一つで，プログラマに対しコンピュータ内に実装されているメモリ（主記憶）より十分大きなメモリ空間があるように見せかける技術である．Fig.5.1 に示すように，仮想記憶機構では１次元の仮想メモリ空間をページと呼ばれる小単位に区切り，ページ単位でデータの

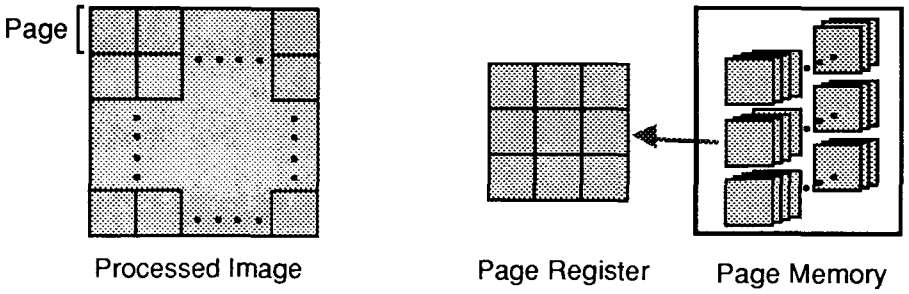
参照を行う．プログラマからは，Fig.5.1(a) のような大きなメモリ空間があるように見えるが，実際には，Fig.5.1(b) のように，２次記憶装置上にメモリの内容が保存され，必要に応じてページごとに主記憶上へ呼び出される．主記憶と２次記憶装置間のデータ移動は自動的に行われ，結果として，大きなメモリ空間を自由に取り扱うことができる．

本研究では，並列ディジタル光演算にページ分割の概念を適用し，処理画像を分割してページ単位で処理する２次元仮想記憶機構を考案した．Fig.5.2 にその概念図を示す．２次元仮想記憶機構は，Fig.5.2(a) に示すように，非常に多くの画素数を持つ画像に対する並列演算を可能にする．実際には，Fig.5.2(b) に示すように，ページメモリから少数ページが逐次的に読み出され，それらに対する並列演算が行われる．従来の仮想記憶機構における仮想記憶，主記憶，２次記憶が，２次元仮想記憶機構における処理画像，ページレジスタ，ページメモリにそれぞれ対応する．なお，この２次元仮想記憶機構を使用するシステムは，並列ディジタル光演算システムで，スペースインバリアントな処理を行うものに限定する．従来の仮想記憶機構は，処理の逐次性により仮想記憶上のデータを局所的に処理するが，２次元仮想記憶機構は，仮想画像全体に対して同一の演算を行う．



(a)　プログラマから見たメモリ空間　　(b)　ハードウェアにおける実現方法

Fig.5.1　仮想記憶機構



(a)　プログラマから見た処理画像　　(b)　ハードウェアにおける実現方法

Fig.5.2　２次元仮想記憶機構

５.３　相関演算モデルと２次元仮想記憶機構への適用

現在までに報告されている並列光演算原理の多くは，並列近傍画素間演算を実現する方法として相関演算を利用している [18-21]．そこで，２次元仮想記憶機構に対する議論を一般化するため，各並列光演算原理の手順を，Fig.5.3 に示す相関演算モデルで記述する．すなわち，入力画像とそれに対する処理を決定するカーネルを考え，それらの間の相関演算として各並列光演算原理を一般化する．このモデルにおいて，カーネルは入力画像の画素間隔に等しい格子上に並べられた点関数の集合パターンである．相関演算の結果，カーネル上のそれぞれのデルタ関数に対応して入力画像がシフトし，出力面で重なり合う．各種並列光演算原理において，カーネルとそれに対応する概念を Table 5.1 に示す．出力画像は，相関画像において，カーネルの原点と入力画像との相関像が出力される領域をクリッピングした２値画像とする．

２次元仮想記憶機構では，画像から分割されたページを単位とする処理が行われる．この場合，相関演算モデルは，ページを入力とし，カーネルとの相関演算結果をページとして出力するものとして適用される．ただし，ページ間の連続性を保つためには，相関出力をページ領域でクリッピングせず，隣接したページへの出力として扱う必要がある．すなわち，Fig.5.4 に示すように，あるページの相関演算結果が隣接ページへも出力されるため，それらを適切に処理しなければならない．特に，並列光演算システムでは，相関演算は並列に実行されるため，出力面での処理方法が演算効率に大きく影響する．

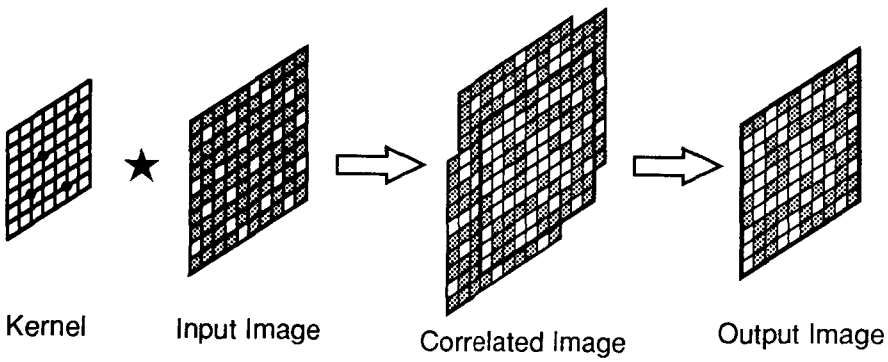


Fig.5.3　相関演算モデル

Table 5.1　各並列光演算原理におけるカーネルに対応する概念

Optical Parallel Computing Principle	Concept Corresponding to Kernel
Optical Array Logic	Operation Kernel
Symbolic Substitution	Search and Substitution Pattern
Binary Image Algebra	Reference Image
Image Logic Algebra	Neighborhood Configuration Pattern

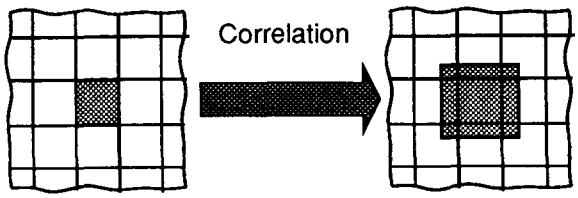
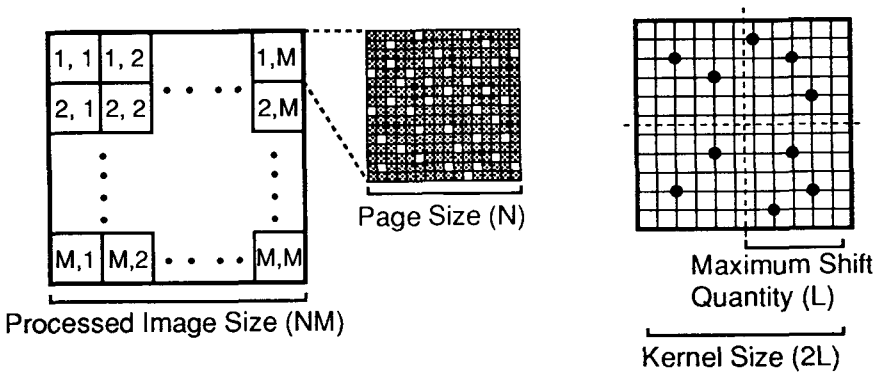


Fig.5.4　２次元仮想記憶機構における相関演算結果の隣接ページへの出力



(a)　処理画像　　(b)　カーネル

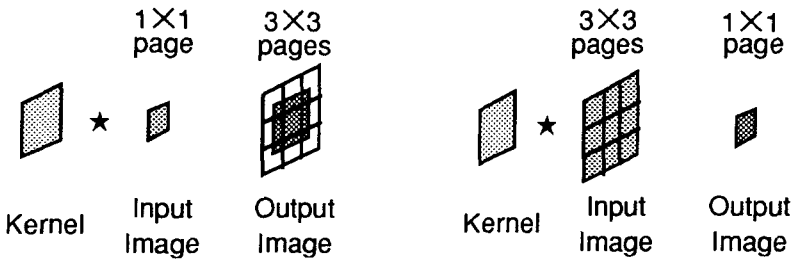
Fig.5.5　画像およびカーネルに関する諸量の関係

以下の考察を簡単化するために，処理対象の画像は $M \times M$ ページ，各ページは $N \times N$ 画素により構成されているものとする．また，カーネルは原点を中心に四方向へ等しく広がっていると仮定する．カーネルの原点から端までの長さは，相関演算により入力画像がシフトされる最大量に等しいため最大シフト量と呼ぶ．シフト量が画像サイズ以上の場合，出力画像領域に入力画像の情報が入らないため，最大シフト量は画像サイズより小さいと仮定する．画像およびカーネルに関する諸量の関係を Fig.5.5 に示す．最大シフト量 L のカーネルのカーネルサイズは $2L \times 2L$ となる．

5.4　２次元仮想記憶機構の処理手順

5.4.1　最大シフト量が画像のページサイズより小さい場合

最大シフト量 L が画像サイズ N より小さい場合 $(0 < L < N)$，１ページの処理結果は３×３ページ内におさまる．したがって，１ページを入力とし，出力面で３×３ページを同時に取り込めばよいことがわかる（Fig.5.6(a)）．しかし，この方法では各ページの演算結果は１回の相関演算だけでは得られず，９回の相関演算の総和を求める必要がある．そこで，目的のページと隣接した３×３ページを入力とし，中心の１ページのみを出力とする方式が有効となる（Fig.5.6(b)）．すなわち，この方法では，隣接ページの情報が光学的に並列に加算され，中央１ページの処理結果が１度に得られる．この場合，全画像の相関演算はページ数に等しい回数，すなわち M^2 回で完了する．



(a)　出力面拡張　　(b)　入力面拡張

Fig.5.6　各ページの境界部分と他のページとの連結方法

5.4.2　最大シフト量が画像のページサイズより大きい場合

最大シフト量がページサイズより大きい場合，１ページの処理結果は３×３ページの領域にはおさまらない．その場合は，カーネルを $2N \times 2N$ のカーネルサイズを持つサブカーネルに分割して処理する．このとき，分割したカーネルが $2N \times 2N$ に満たない場合は拡張を行い，すべてのサブカーネルが等しいカーネルサイズを持つようにする．

カーネル分割の例として，$N \le L < 2N$ の場合を Fig.5.7 に示す．この場合，カーネルは２×２個のサブカーネルに分割される．各サブカーネルの最大シフト量はページサイズ N を越えないため，各サブカーネルによる演算は５．４．１項に示した方法で実行できる．１ページの演算結果は，各サブカーネルに対する演算結果の総和により得られる．したがって，$N \le L < 2N$ では，１ページあたり４回の相関演算が必要となり，全相関回数 C は，ページ数 M を用いて，

$$C = 4M^2, \tag{5-1}$$

となる．

一般には，$(k-1)N \le L \le kN$ に対してサブカーネル数は k^2 となるので，全相関回数 C は $O(k^2M^2)$ で与えられる．C と L の関係を図示すると Fig.5.8 のようになる．これより，２次元仮想記憶機構における相関演算は，最大シフト量 L によって不連続的に変化し，

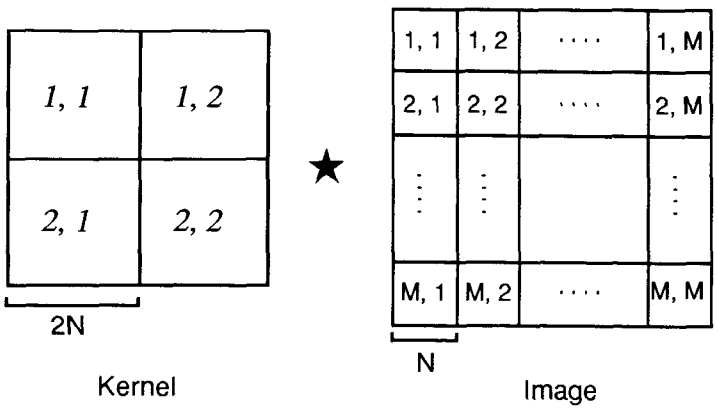


Fig.5.7　N ≤ L < 2N の場合のカーネル分割例

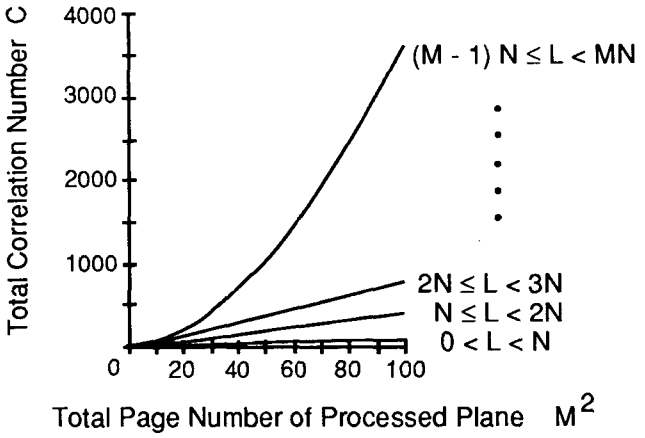


Fig.5.8　全相関回数と全ページ数，最大シフト量との関係

Table 5.2　光アレイロジックによる処理とその最大シフト量

Application	Required Shift Size
Binary Image Processing	
(Edge Detection, Thinning)	3
Gray Image Processing	
(Digital Filtering)	$(M+1)(B+1)/2$
Numerical Processing	
Ripple-Carry Addition and Subtraction	3
Ripple-Carry Multiplication	$2B-1$
MSD Addition and Subtraction	7
MSD Multiplication	$4B-3$
Maze Solution	3
Turing Machine	5
Systoric Array Processor	$4B+5$
Inference Engine (1000 Nodes)	
Template Matching Method	～ 60
Token Propagation Method	～ 4000
Expert System	～ 4000
Database Management	
Selection	$2N-1$
Projection	$2N-1$
Semijoin	$2N-1$
Sorting	$2B-1$

M : image size,　N : data number,　B : bit number of data

L が N の整数倍を越えるごとに処理効率が大きく低下することがわかる．したがって，２次元仮想記憶機構設計におけるページサイズ決定の過程では，処理に必要となる最大シフト量 L を下回らない範囲でページ画素数を最小にすることが一つの規範となる．Table 5.2 は，光アレイロジックにおけるいくつかの処理と最大シフト量をまとめたものである [25, 27-32]．この表より，各応用に対する２次元仮想記憶機構の適切なページ画

素数が決定できる．

5.5　パイプラインページ転送方式を用いた２次元仮想記憶機構

２次元仮想記憶機構を実現するためには，入力画像の３×３ページのデータ転送を効率的に行う必要がある．そこで，ページデータ転送に並列光インターコネクションを利用する２次元仮想記憶機構の構成法を考案した．Fig.5.9 に示すように，このシステムは，ページ転送系と相関光学系を組み合わせた構成を持ち，ページ転送系で相関光学系に対する入力ページを順次そろえていく．

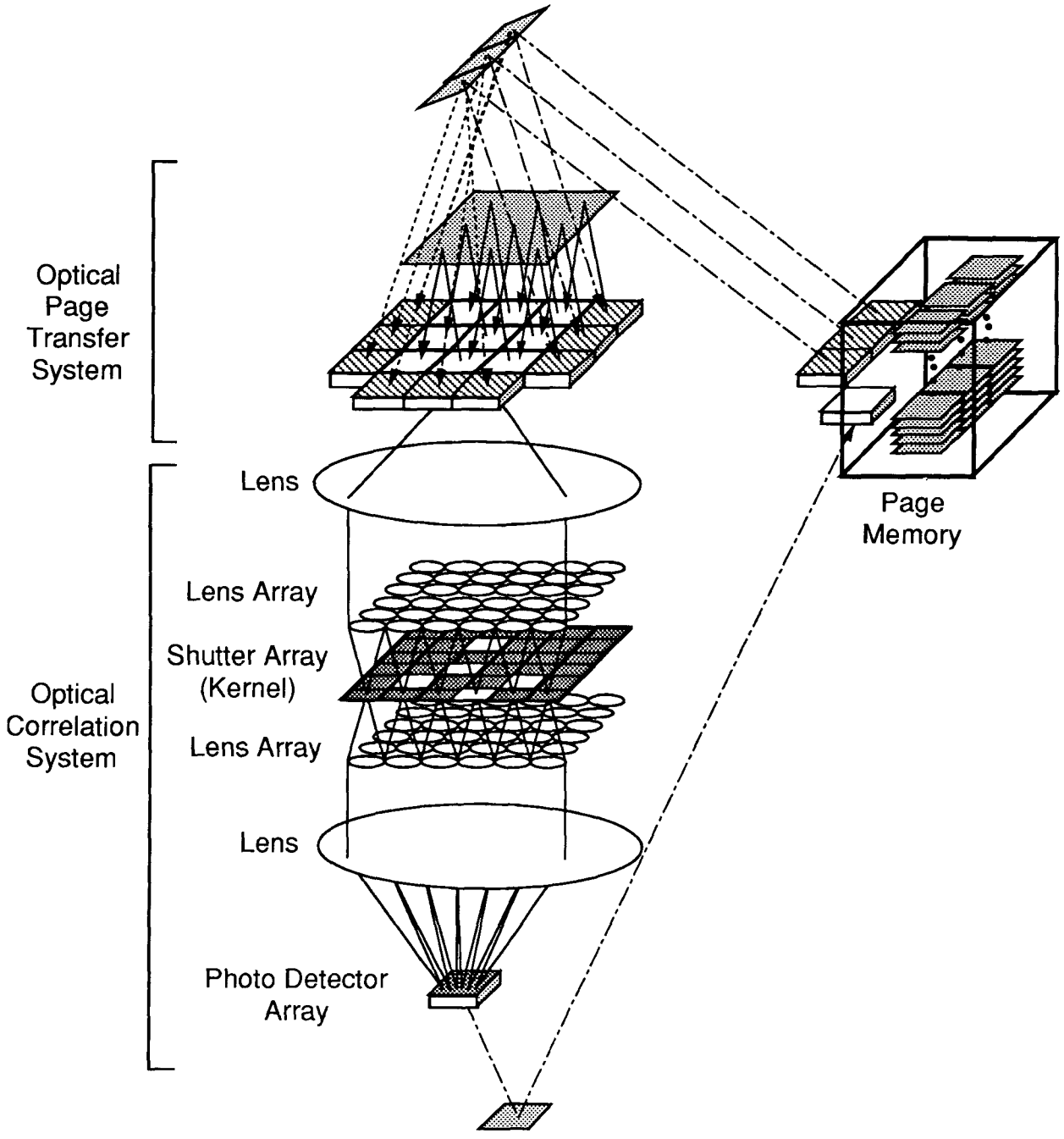


Fig.5.9　パイプラインページ転送方式を用いた２次元仮想記憶機構

本システムは次の手順により動作する．まず，ページメモリから３ページのデータを選び，それらを同時にページ転送系の入力ポート（斜線部）に転送する．入力ポートは３ケ所にあるが，ページデータの転送方向により１つのポートを選ぶ．入力ポートに送られたページデータはLEDアレイなどの発光素子により光信号に変換した上で，ページ転送光学系を通して中央の３×３ページの相関光学系入力部分に転送する．Fig.5.10はページ転送光学系におけるページ転送の様子を示す．中央の３×３ページが相関演算の入力データとして用いられる．図に示すように，パイプライン的にページ転送することにより，M^2+2 回のページ転送で全画像に対する処理を完了できる．カーネル分割による処理の場合は，各サブカーネルごとに上記処理を行い，サブカーネルの演算結果の総和をページメモリ上で求めればよい．

ページ転送用素子には，ページ転送系側に受光素子と発光素子，相関光学系側に発光素子，内部に信号ラッチ回路を有するOEICが利用できる．Fig.5.11(a)にその構成例を示す．１画素分のOEICは，受光素子の信号をラッチへの入力とし，その値を保持する．ページ転送命令により，保持していた値を２つの発光素子に送る．全OEICがこの動作を同時に行うことにより，並列ページ転送が実現できる．受光・発光素子には，それぞれフォトディテクタと発光ダイオードが利用できる．このOEICは，画素ごとに独立に処理を行い，構造が簡単なため，LSIによる集積化に向いている．ページ転送光学系には，レンズアレイを用いた結像系が利用できる．Fig.5.11(b)にその断面図を示す．相関光学系には，Fig.5.9に示した離散相関光学系[34]が利用できる．その他，ホログラムによるコヒーレント相関光学系[66]なども利用できる．

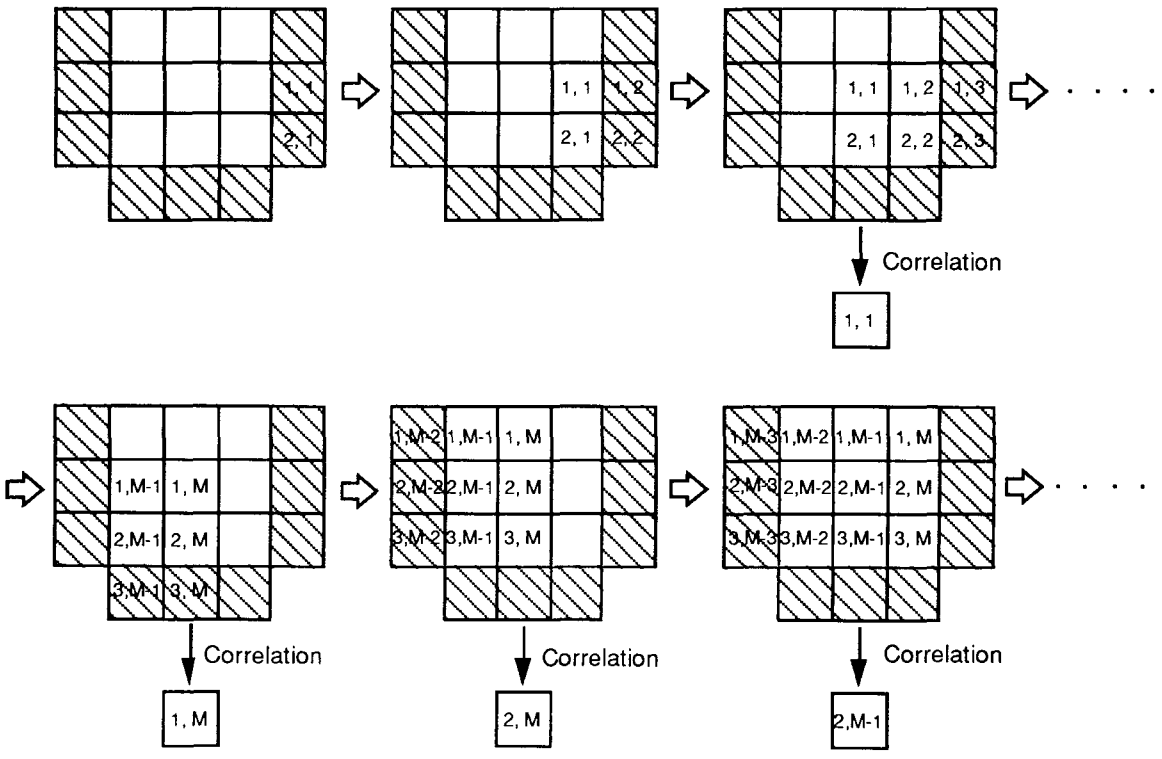
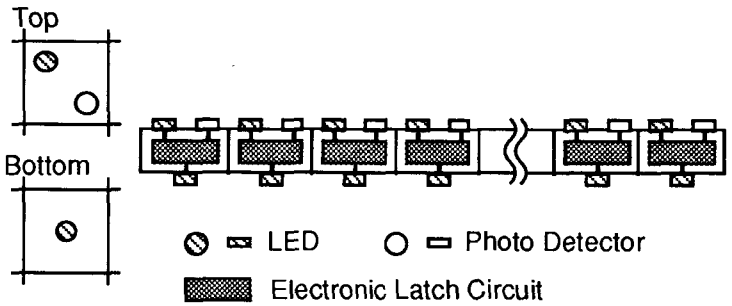


Fig.5.10　ページ転送光学系におけるページ転送の様子

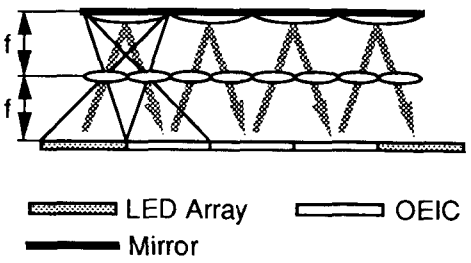
ページメモリとしては，ページデータを光学的に並列転送できるものが望ましい．しかし，現在の光デバイスでは，その実現は困難である．現時点で考えられるページメモリ用装置としては，光磁気ディスクなどの書き換え可能な光ディスクが考えられる．ページデータは，ディスクから逐次的にページ転送系の入力部に転送される．現在光磁気ディスクの転送速度は 10^7 bit / s 程度なので [67]，100×100 画素のページならば，３ページ分を同時に転送しても転送速度は 10^3 pages / s 程度となる．光処理部分の処理速度は，シャッタアレイの応答速度（強誘電性液晶で数十 μ s [65]）と考えられるので，ページ転送部分がシステム処理速度のボトルネックになる．この問題を解決するためには，10^{10} bit / s 程度の転送速度を持つ書き換え可能な光ディスクが要求される．このボトルネックを解消する将来のページメモリとしては，ページホログラムなどページデータを光信号として並列に読み出せるホログラフィックメモリが考えられる [57]．しかし，この実用化には，ホログラフィ技術の進歩が必要である．

5.6　結言

並列光演算システムにおいて，小規模のハードウェアで大画素数の画像に対する処理を実現する方法として，２次元仮想記憶機構を提案した．まず，各種並列光演算原理の処理手順を共通に記述するために，相関演算モデルを導入し，そのモデルを用いて２次元仮想記憶機構の処理手順・処理効率を検討した．その結果，相関演算系において入力側を３×３ページに拡張してページごとに処理を行うと，処理効率を高められることがわかった．また，処理に要求される最大シフト量よりページサイズを大きくした場合に処理効率が最大になるため，これをページサイズ決定の基準とすればよいことを確認し



(a)　ページ転送用OEIC



(b)　ページ転送光学系の断面図

Fig.5.11　ページ転送光学系

た．さらに，2次元仮想記憶機構を実現するシステムについて検討し，パイプラインページ転送が有効に利用できることを示した．

第6章　専用光演算モジュールを用いた並列光演算システム

6.1　緒言

前章までに，光アレイロジックを用いた推論機構，データフロー型処理，データベース処理について述べてきた．しかし，これらは画素数の多い画像やシフト量の大きい演算カーネルを必要とし，現在のハードウェアでは実現困難な処理が多い．この解決法として，本章では，ハードウェアに厳しい要求をする処理を専用光演算モジュール化することにより，従来光アレイロジックプロセッサ1台にかかっていた負荷を分散させることを考える．数種の専用光演算モジュールと光アレイロジックプロセッサでシステムを構成することにより，効率よく光アレイロジックの演算系を実現できることを示す．本章では，今までに開発された光アレイロジックプログラムを解析し，使用回数の多い演算，またはシステムに厳しい要求を課する演算の種類を調べる．そして，各演算を専用光演算モジュール化した場合の処理効率を評価し，従来方法の処理効率と比較する．それらの結果に基づき，専用光演算モジュールを用いた並列光演算システムの構成法を考察する．

6.2　光アレイロジックプログラムの解析

光アレイロジックは，並列プログラムが作成可能であるという特徴を持つ．この特徴をいかして，今までに，画像処理 [27, 29, 31]，数値処理 [25, 28, 31] 等の OALL プログラムが開発されている．本研究においても AI 処理（第2章），データフロー型処理（第3章），データベース処理（第4章）の OALL プログラムを開発した．そこで，光アレイロジックによる処理の傾向を調べるために，これらのプログラムの演算内容を解析し，使用回数の多い演算，またはシステムに厳しい要求をする演算の種類を調査した．

その結果から，パターン展開，シフト演算，論理積，論理和の4種類の演算の出現回数が比較的多く，これらの演算専用モジュールの使用が効果的であることがわかった．光アレイロジックによる推論機構（テンプレートマッチング法，トークン伝播法）とそれを用いたエキスパートシステム，データフロー型処理，ソーティング，データベース処理について，専用光演算モジュールを使用しない場合と使用した場合の処理効率を比較した結果を Table 6.1 - 6.4 に示す．専用光演算モジュールを使用しない場合には，光アレイロジックプロセッサ1台ですべての処理を行うものとした．専用光演算モジュールを使用する場合には，光アレイロジックプロセッサと専用光演算モジュールを併用するものと仮定した．これら以外に，画像処理 [27, 29, 31]，数値処理 [25, 28, 31] についても同様の解析を行ったが，モジュール使用の効果は現われなかった．

専用光演算モジュール使用時の利点として，カーネルサイズ（演算カーネルの一辺に必要なカーネルユニット数）の減少があげられる．パターン展開とシフト演算は通常大きなサイズのカーネルを必要とするため，ハードウェア化が困難である．問題となるこれらの演算を専用光演算モジュール化すれば，光アレイロジックプロセッサではより小さなカーネルサイズの処理だけを行えばよくなる．

Table 6.1　光アレイロジックによる推論機構（テンプレートマッチング法）の評価

		Without Modules	With Modules
Number of Encoding		$< (2S + 4N_{Di} + 4)\, i + 3$	$< (4N_{Di} + 2)\, i + 3$
Number of Correlation		$<[2S + \{3(P_N + P_L) + 5\}N_{Di} + P_N + P_L + 2]\, i + 3$	$< [\{3(P_N + P_L) + 5\}N_D + P_N + P_L]\, i + 3$
Kernel Size [/ Kernel Unit]	(x)	3	3
	(y)	$4\,(P_N + P_L) + 1$	$4\,(P_N + P_L) + 1$
Image Size [/ Pixel]	(x)	S	S
	(y)	$4\,(P_N + P_L)$	$3\,(P_N + P_L)$
Number of Shift Operation		———	$< S\, i$
Number of Logical AND		———	$< S\, i$
Number of Logical OR		———	i

S : number of subnetworks
i : cycle number of inference sequence
P_N : pixel number for a node
P_L : pixel number for a link
N_{Di} : number of nodes derived in the i -th cycle of inference

Table 6.2　光アレイロジックによる推論機構（トークン伝播法）とそれを用いたエキスパートシステムの評価

		Inference Engine		Expert System	
		Without Modules	With Modules	Without Modules	With Modules
Number of Encoding		$8i + 1$	$5i + 1$	$17i + 15$	$4i + 8$
Number of Correlation		$12i + 1$	$9i + 1$	$21i + 14 + 2MH$	$8i + 7 + 2MH$
Kernel Size [/ Kernel Unit]	(x)	$(2S - 1)V$	3	$(2S - 1)V$	$2(S - 1)V - 2L - 1$
	(y)	$(2N - 1)H$	3	$(2N - 1)H$	$2(N - 1)H + 1$
Image Size [/ Pixel]	(x)	SV	SV	SV	SV
	(y)	NH	NH	NH	NH
Number of Pattern Expansion		——	$2i$	——	$5i + 5$
Number of Logical AND		——	i	——	$5i + 1$
Number of Logical OR		——	0	——	$3i + 1$

N : number of nodes
S : number of subnetworks
V : vertical pixel number of a node pattern
H : horizontal pixel number of a node pattern
M : vertical pixel number of marker number pixels
i : cycle number of inference sequence
L : pixel number for a link

Table 6.3　光アレイロジックによるデータフロー型処理の評価

		Without Modules	With Modules
Number of Encoding		$22i$	$10i$
Number of Correlation		$27i$	$15i$
Kernel Size [/ Kernel Unit]	(x)	$< 5(2N - 1)$	9
	(y)	$(B + 2)(2N - 1)$	$2B + 3$
Image Size [/ Pixel]	(x)	$< 5N$	$< 5N$
	(y)	$(B + 2)N$	$(B + 2)N$
Number of Pattern Expansion		——	$6i$
Number of Logical AND		——	$3i$
Number of Logical OR		——	$3i$

i : cycle number of data flow processing sequence
N : number of nodes
B : number of bit for a data

Table 6.4　光アレイロジックによるソーティングの評価

		Odd-even Transposition Sort		Odd-even Merge Sort	
		Without Modules	With Modules	Without Modules	With Modules
Number of Encoding		$5N + 4$	$3N + 4$	$12S$	$7S$
Number of Correlation		$(B + 7)N + 8$	$(B + 5)N + 8$	$(4B + 13)S$	$(4B + 8)S$
Kernel Size [/ Kernel Unit]	(x)	3	3	$N + 1$	$N + 1$
	(y)	$2B - 1$	$2B - 1$	$4B - 1$	$4B - 1$
Image Size [/ Pixel]	(x)	N	N	N	N
	(y)	$B + 1$	$B + 1$	$2B$	$2B$
Number of Pattern Expansion		——	1	——	$2S$
Number of Logical AND		——	N	——	$2S$
Number of Logical OR		——	0	——	S

N : number of data
B : number of bit for a data
S : stage number for odd-even merge sort ($= \log_2 N (\log_2 N + 1) / 2$)

Table 6.5　光アレイロジックによるデータベース処理の評価

		Selection		Projection	
		Without Modules	With Modules	Without Modules	With Modules
Number of Encoding		≤ 11	≤ 2	$\leq 7N+5$	$\leq 2N$
Number of Correlation		$\leq S+10$	$\leq S+1$	$\leq 7N+5$	$\leq 2N$
Kernel Size [/ Kernel Unit]	(x)	$2N-1$	1	$2N-1$	1
	(y)	$2MS-1$	$2S-1$	$\leq 2MS-S$	$2S-1$
Image Size [/ Pixel]	(x)	N	N	N	N
	(y)	MS	MS	MS	MS
Number of Pattern Expansion		——	5	——	$\leq 2N+3$
Number of Shift Operation		——	1	——	$\leq N+1$
Number of Logical AND		——	≤ 3	——	$\leq 2N+1$
Number of Logical OR		——	0	——	0

		Semijoin		Sorting	
		Without Modules	With Modules	Without Modules	With Modules
Number of Encoding		$\leq 10N+7$	$\leq 3N$	$7N+6$	$5N+2$
Number of Correlation		$\leq (S+9)N+7$	$\leq (S+1)N$	$(S+9)N+10$	$(S+7)N+6$
Kernel Size [/ Kernel Unit]	(x)	$2N-1$	1	3	3
	(y)	$2MS-1$	$2S-1$	$2MS-1$	$2S+1$
Image Size [/ Pixel]	(x)	N	N	N	N
	(y)	MS	MS	MS	MS
Number of Pattern Expansion		——	$\leq 2N+4$	——	$N+2$
Number of Shift Operation		——	$\leq 2N+3$	——	1
Number of Logical AND		——	$\leq 3N+1$	——	1
Number of Logical OR		——	$\leq N$	——	3

M : number of attribute
N : number of tuple
S : bit number of data

また，専用モジュール使用の利点として符号化回数，相関演算回数の減少も重要である．これらの回数は，専用光演算モジュールの使用回数分だけ減少する．専用モジュールの演算は符号化を必要とせず，それぞれ最適なハードウェアで高速に処理できる．符号化と相関演算に要する時間が長い場合は，専用モジュール使用による高速化が特に有効である．さらに，各モジュールでは画像が符号化されないため，同一の光学素子を使用しても，光アレイロジックプロセッサに比べて4倍の画素数を処理できる利点も有する．

6.3　専用光演算モジュールの種類

前節で解析した結果，パターン展開，シフト演算，論理和，論理積の4種類の演算の専用モジュールの使用が効果的であることがわかった．本章では，光アレイロジック処理手順では実現が困難な処理を含めて，専用光演算モジュールの種類と，各モジュールの具体的な実現法を考える．

1）パターン展開モジュール

パターン展開は，データを処理画像上に多数複製する技術であり，通常大きなカーネルサイズを必要とする．この処理は，単純な画素パターンの移動・複写で実現できるため，Fig.6.1 のような，レンズを用いた簡単な光学系で実現可能である．パターン展開モジュールを用いると，光アレイロジックプロセッサに必要なカーネルサイズを大幅に減少させることができる．

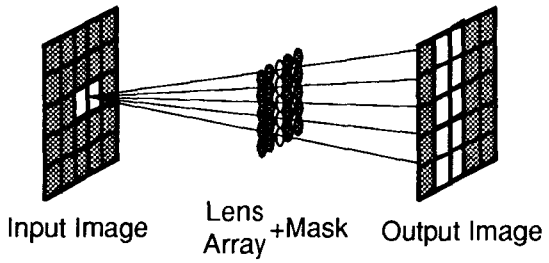


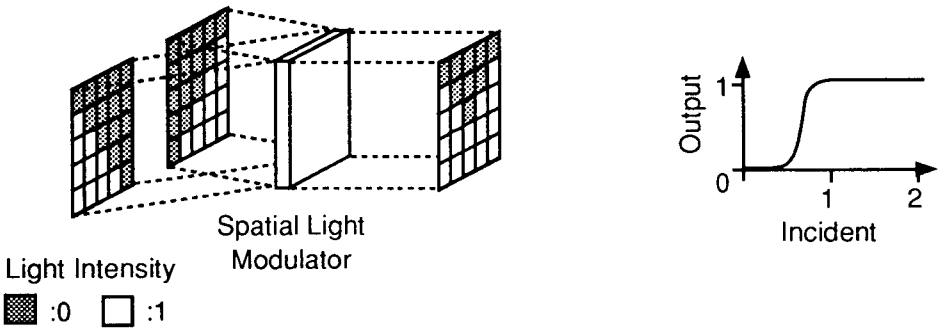
Fig.6.1　パターン展開専用光演算モジュールの光学系

2）シフト演算モジュール

シフト演算は，画素パターンの転送に使われる場合が多い．したがって，光アレイロジックによるシフト演算では，大きなカーネルサイズが必要な場合が多い．これは，プリズム，ミラーなどを用いた簡単な光学系で実現できる．すなわち，シフト演算モジュールを用いることにより，光アレイロジックプロセッサに必要なカーネルサイズを大幅に減少させることができる．

3）論理和モジュール，論理積モジュール

対応画素間の論理和は，Fig.6.2 に示すように，2枚の入力画像をしきい値素子上に投影して重ね合わせ，入力光強度1以上の画素を光強度1で出力すれば簡単に実現でき



(a)　光学系　　(b)　空間光変調器の入出力特性

Fig.6.2　空間光変調器を用いた並列論理和

る．同様に，対応画素間の論理積も，Fig.6.2 において入力光強度２以上で出力光強度が１になる入出力特性を持つしきい値素子により，簡単に実現できる．したがって，論理和と論理積は，専用光論理ゲートを用いることで，高速な実行が期待できる．

４）状態変数モジュール

状態変数 Number と Zero は，演算結果の画像の状態を表す変数である．変数 Number は，画像中の画素値１の画素数を表す．変数 Zero は，画像中の全画素値が０かどうかを表す（１．６．１項）．状態変数の値を光アレイロジックの演算で求めるためには，状態変数の出力に対して考えられる全ての画素パターンを演算カーネルで指定しなければならず，非常に効率が悪くなる．しかし，これらの値は，Fig.6.3 に示すように，レンズを利用して全画素を一点に集光し，その光強度を検出することにより簡単に求められる．

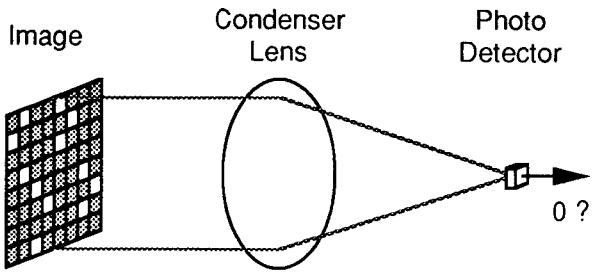


Fig.6.3　状態変数光学系

５）カーネルパターン生成モジュール

カーネルパターン生成モジュールは，推論機構（テンプレートマッチング法）において使用される（Fig.2.5）．このモジュールは，２枚の入力画像から見つけ出したい画素パターンの組を入力とし，両方のパターンが存在する場所の左上隅画素を１にする演算カーネルのパターンを，画素パターンとして出力する．光アレイロジックでこの演算を行うと，検出したい画素パターンの画素数 N に対し，$(2N + 1)$ 回の相関演算が必要とな

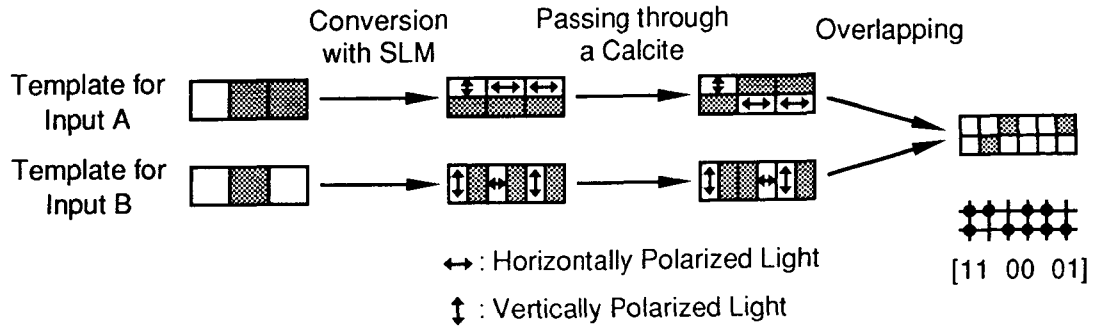


Fig.6.4　カーネルパターン生成モジュールの光学的実現方法

り，処理効率が悪い．しかし，Fig.6.4 のように直線偏光の方位としてテンプレートパターンを符号化し，複屈折結晶によるシフト現象を利用することにより高速にカーネルパターンが生成できる．

6.4　専用光演算モジュールを用いた処理例

専用光演算モジュールを用いて処理を行う場合のシステムの動作解析を行った．具体的には，代表的な処理についてモジュール間接続パターンを記述した．その結果を Fig.6.6 - 6.11 に示す．図中の記号は Fig.6.5 に示す．このシステムには次の仮定をおく．

1）各モジュールの入出力は２値画像とする．パターン展開，シフト演算，相関演算の各モジュールとレジスタは１入力１出力であり，論理積，符号化，カーネルパターン生成の各モジュールは２入力１出力である．論理和モジュールは，多入力１出力である．

2）状態変数モジュールは，１枚の入力画像に対し，１整数を出力する特殊なモジュールである．

3）各モジュール，画像メモリのファンアウト数は無制限である．

4）画像記憶装置として，レジスタとメモリを考える．レジスタは，メモリよりアクセス速度が速いとし，ここでは画像を一時的に保持する目的で使用する．また，メモリの内容は画像名を用いてアクセスできる．

5）任意のモジュール間で接続可能である．

6）全モジュールは同一クロックで動作する．

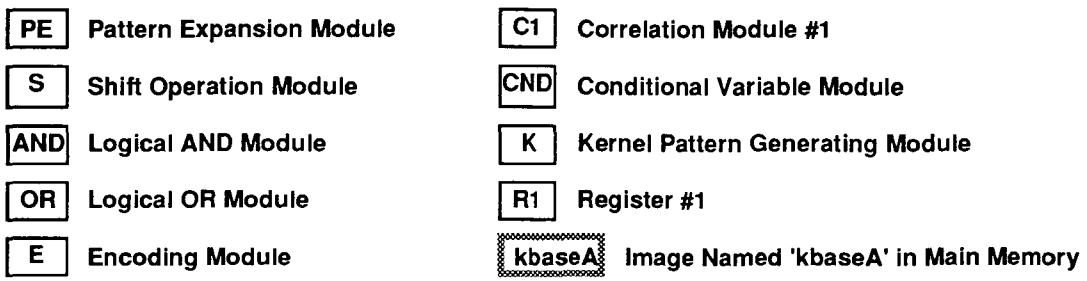


Fig.6.5　モジュール間接続パターン記述用記号

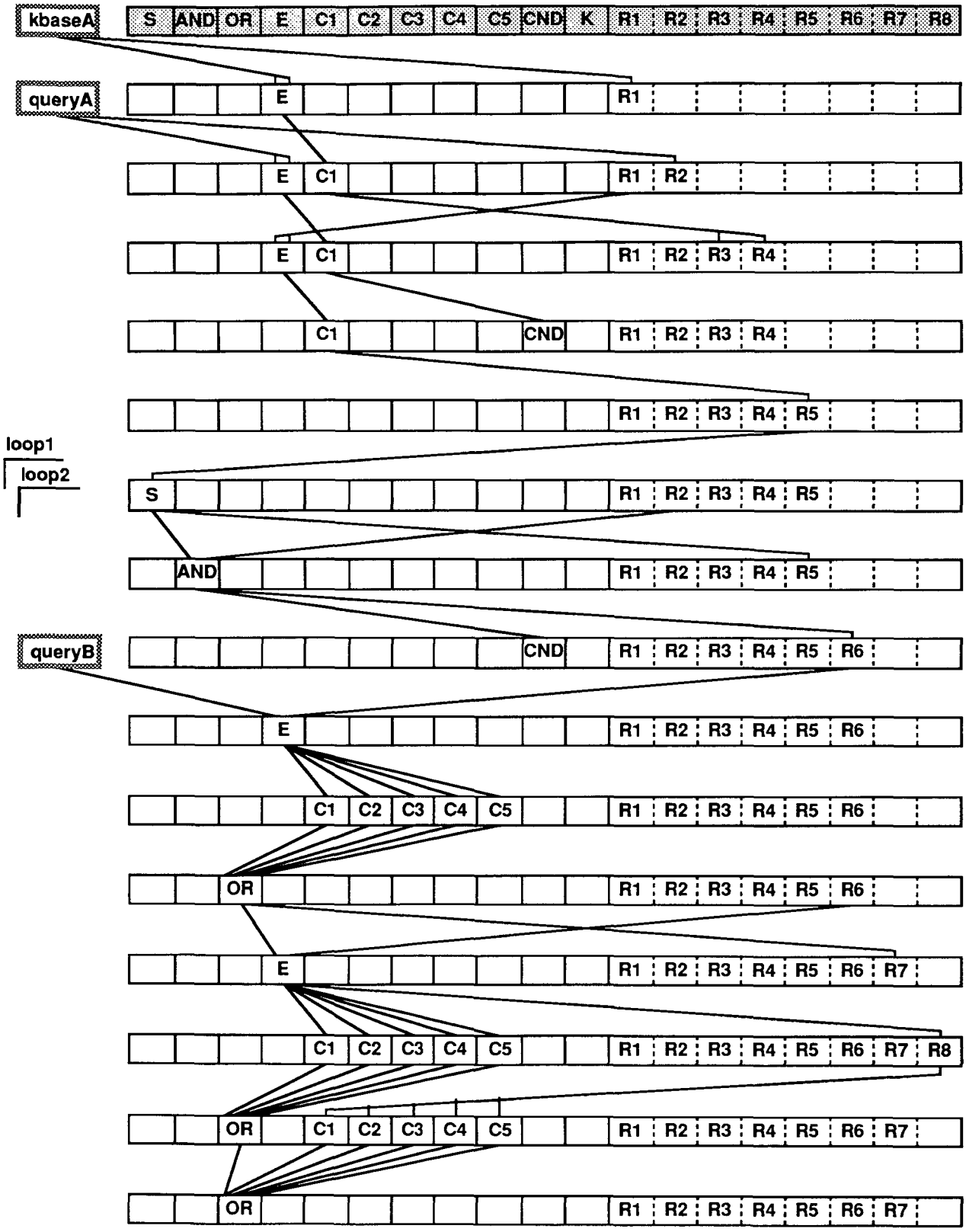


Fig.6.6　推論機構（テンプレートマッチング法）のモジュール間接続パターン（１）

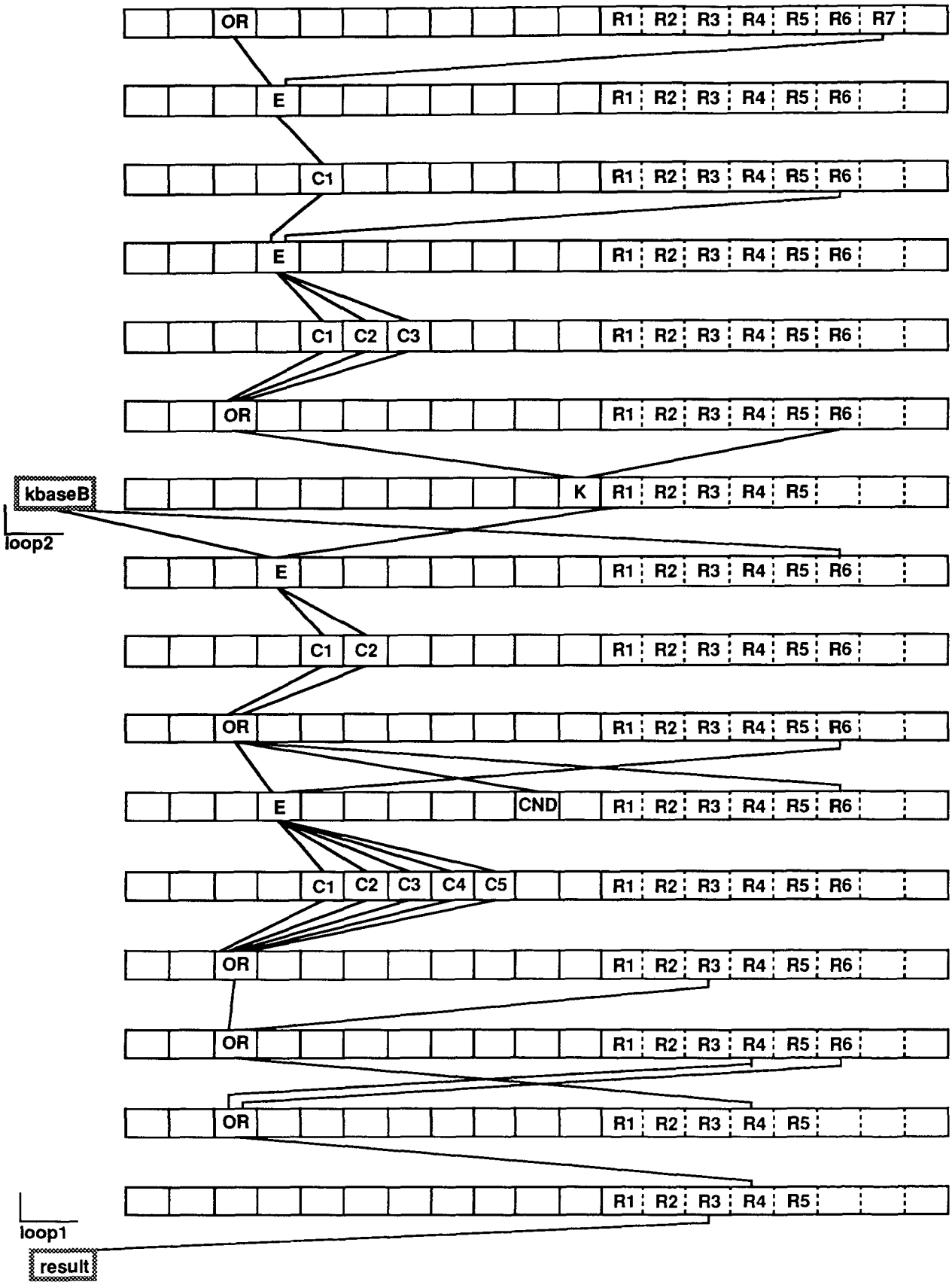


Fig.6.7　推論機構（テンプレートマッチング法）のモジュール間接続パターン（2）

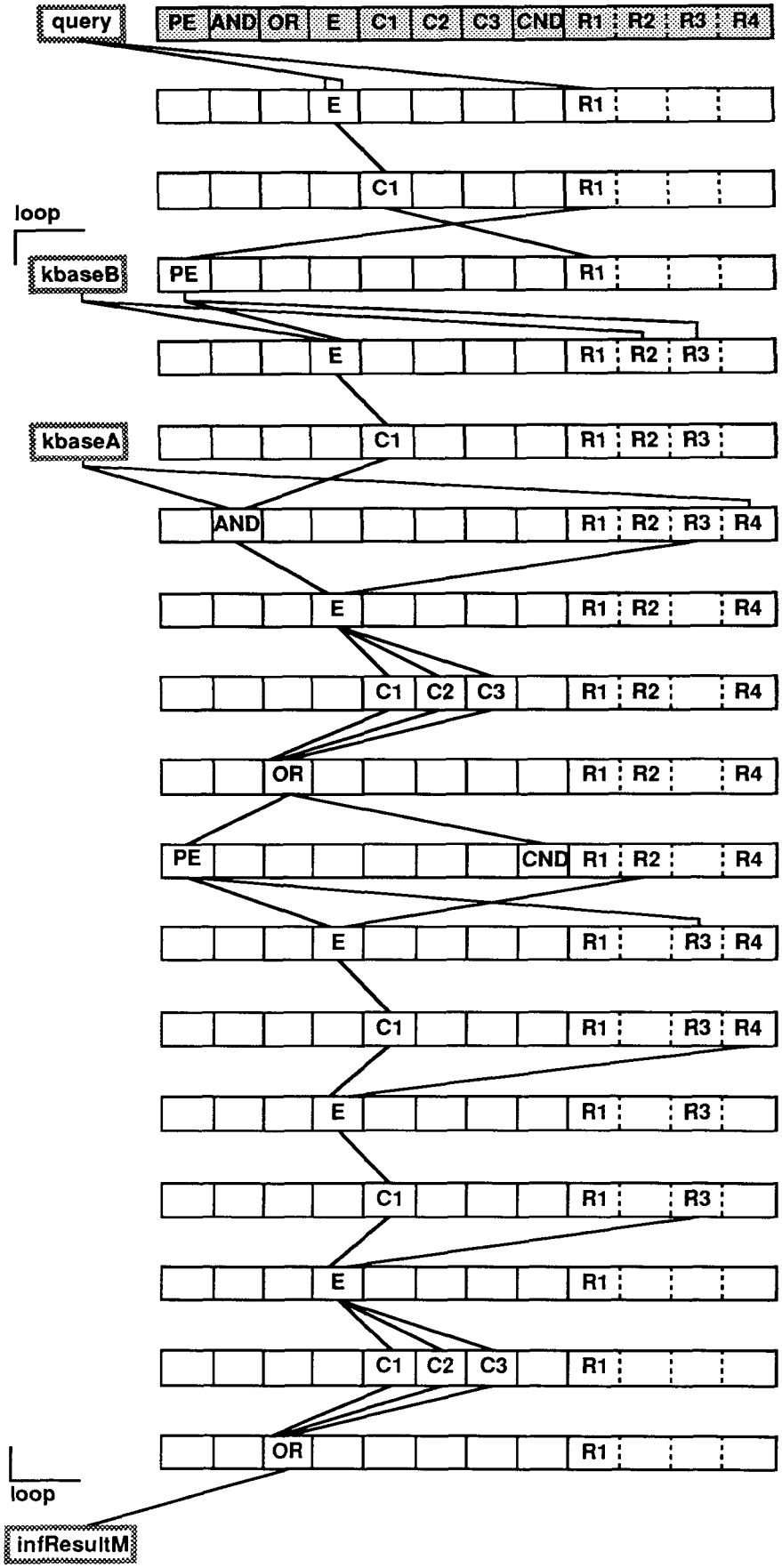


Fig.6.8　推論機構（トークン伝播法）のモジュール間接続パターン

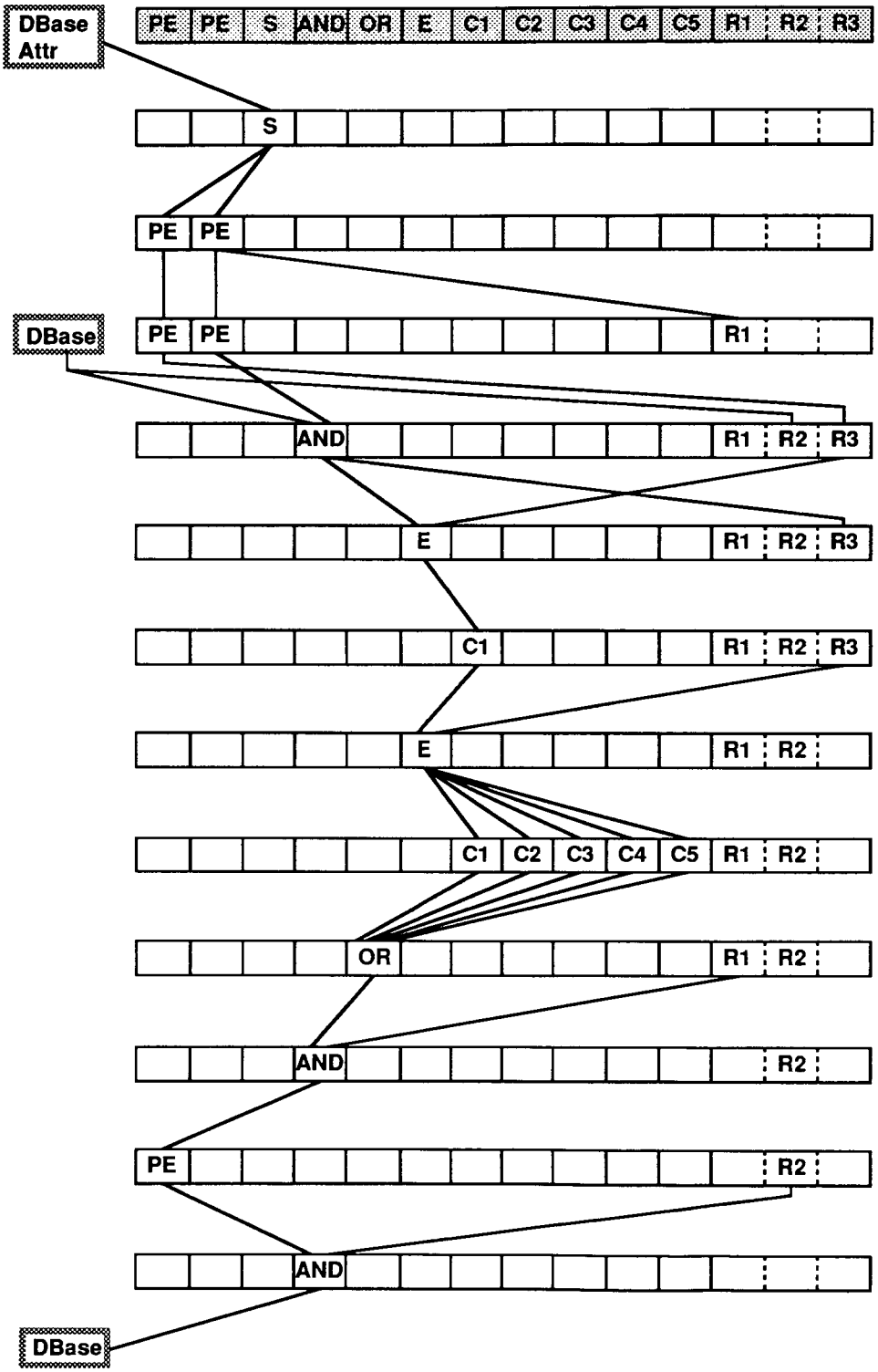


Fig.6.9　データベース処理（選択演算）のモジュール間接続パターン

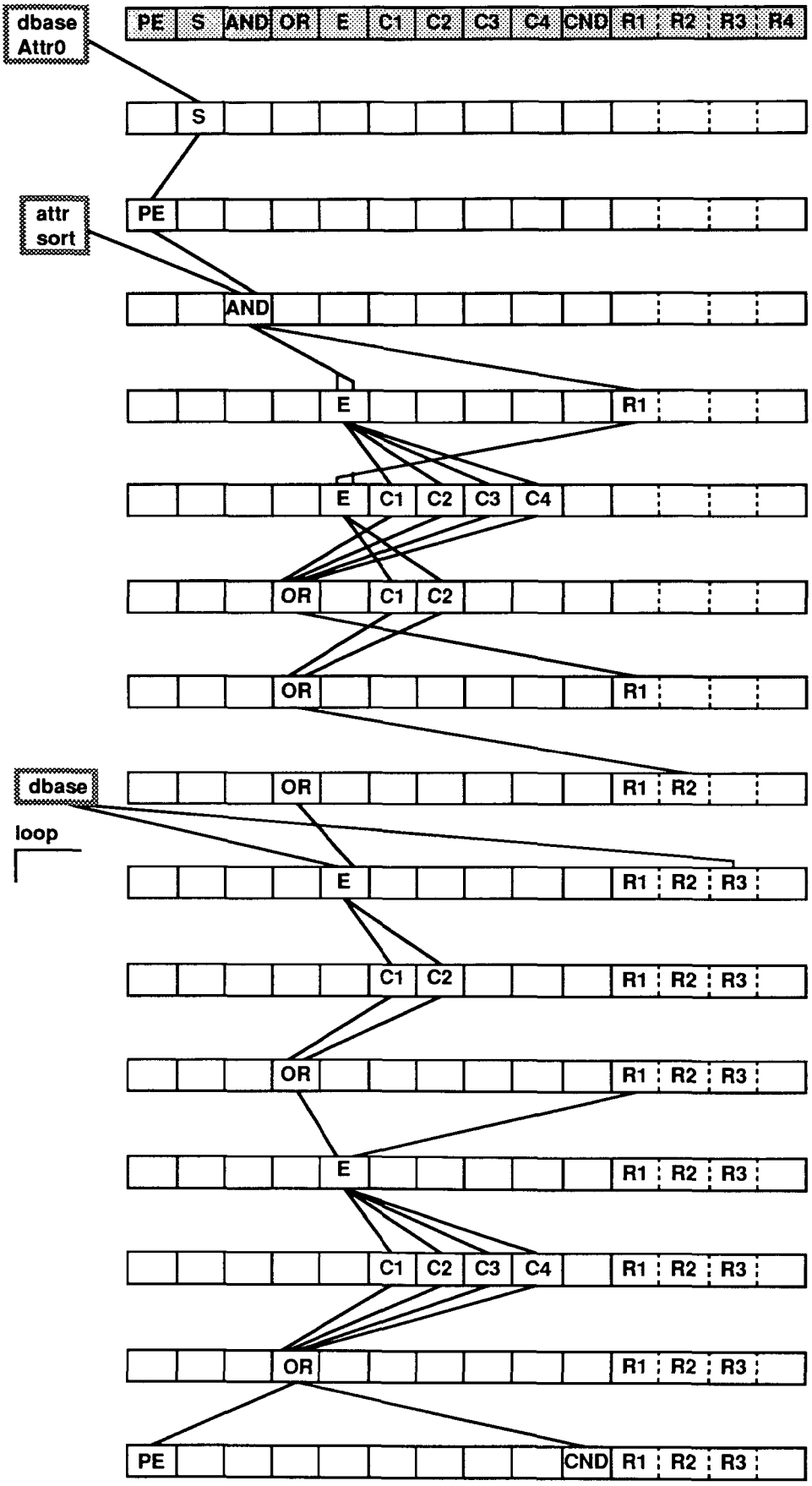


Fig.6.10　データベース処理（ソート演算）のモジュール間接続パターン（１）

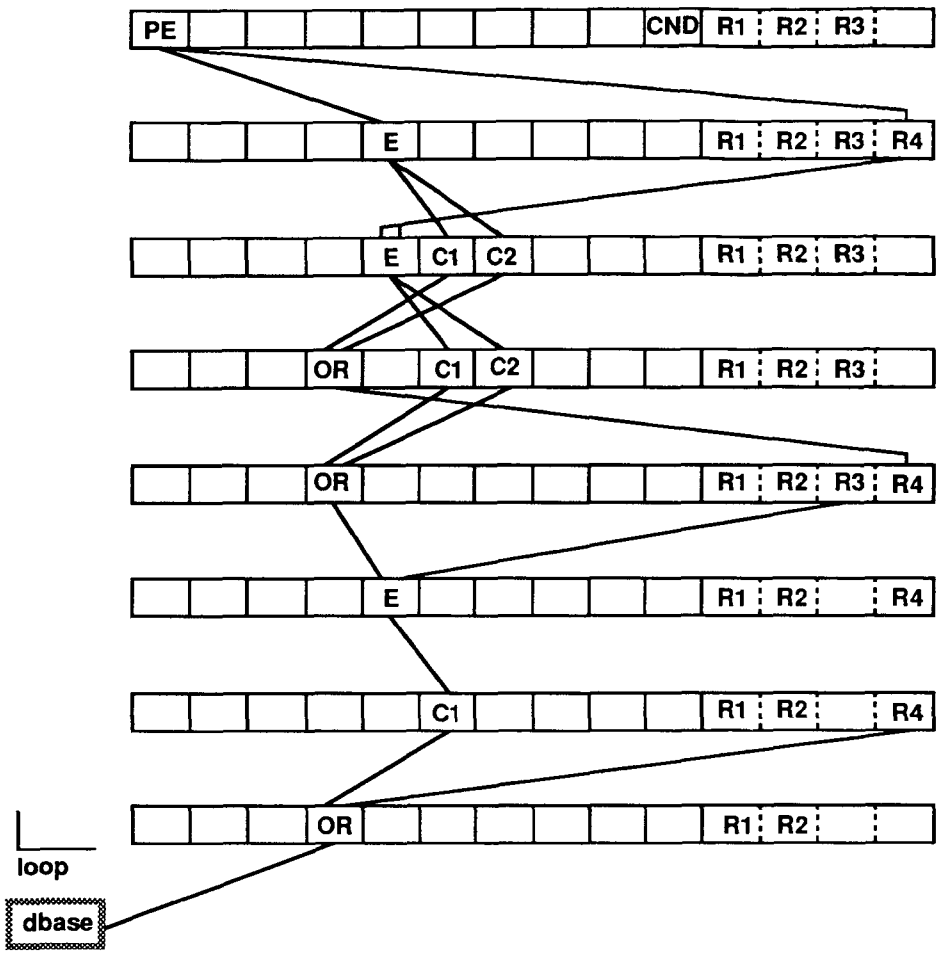


Fig.6.11　データベース処理（ソート演算）のモジュール間接続パターン（2）

Fig.6.6 - 6.11 は，モジュール間接続の時間的推移を表している．各図の1行目ではモジュールの種類とその配置を示し，2行目以降では記号の書かれているモジュールが処理を行っていることを表す．ただし，レジスタでは，記号が書かれている間，最後に書き込まれた画像が保持されていることを表す．‘loop’のかっこで囲まれた部分は，指定回数だけ繰り返す処理の部分である．各モジュールとも，演算命令など制御信号の流れは省略している．

6.5　専用光演算モジュールを用いた並列光演算システム

前節までに，光アレイロジック処理に有効な専用光演算モジュールを検討し，それらのモジュールを利用した場合のモジュール間接続パターンを調べた．本節では，以上の結果をもとに，専用光演算モジュールを用いて効率的な処理を行う並列光演算システムの構成法を考察する．

6.5.1　画像伝送ネットワークによるモジュール間接続方式

専用光演算モジュールを用いたシステム構成では，各モジュール間の接続方法が重要である．専用光演算モジュールを用いた並列光演算システムの構成としては，Fig.6.12 に示すように，各モジュール・メモリ間を画像伝送ネットワークで接続する方法が考え

られる．ただし，符号化－相関演算－並列論理和の各モジュールは，光アレイロジックで処理を行う限り交替はできない．そこで，この3モジュール間は固定接続とし，これらのモジュールの組を光アレイロジックモジュールと呼ぶ．この画像伝送ネットワークを光学的に実現する方法としては，画像クロスバスイッチ (OPIX) [68] や，サニャックインバータを用いたバンヤンネットワーク [53] が利用できる．

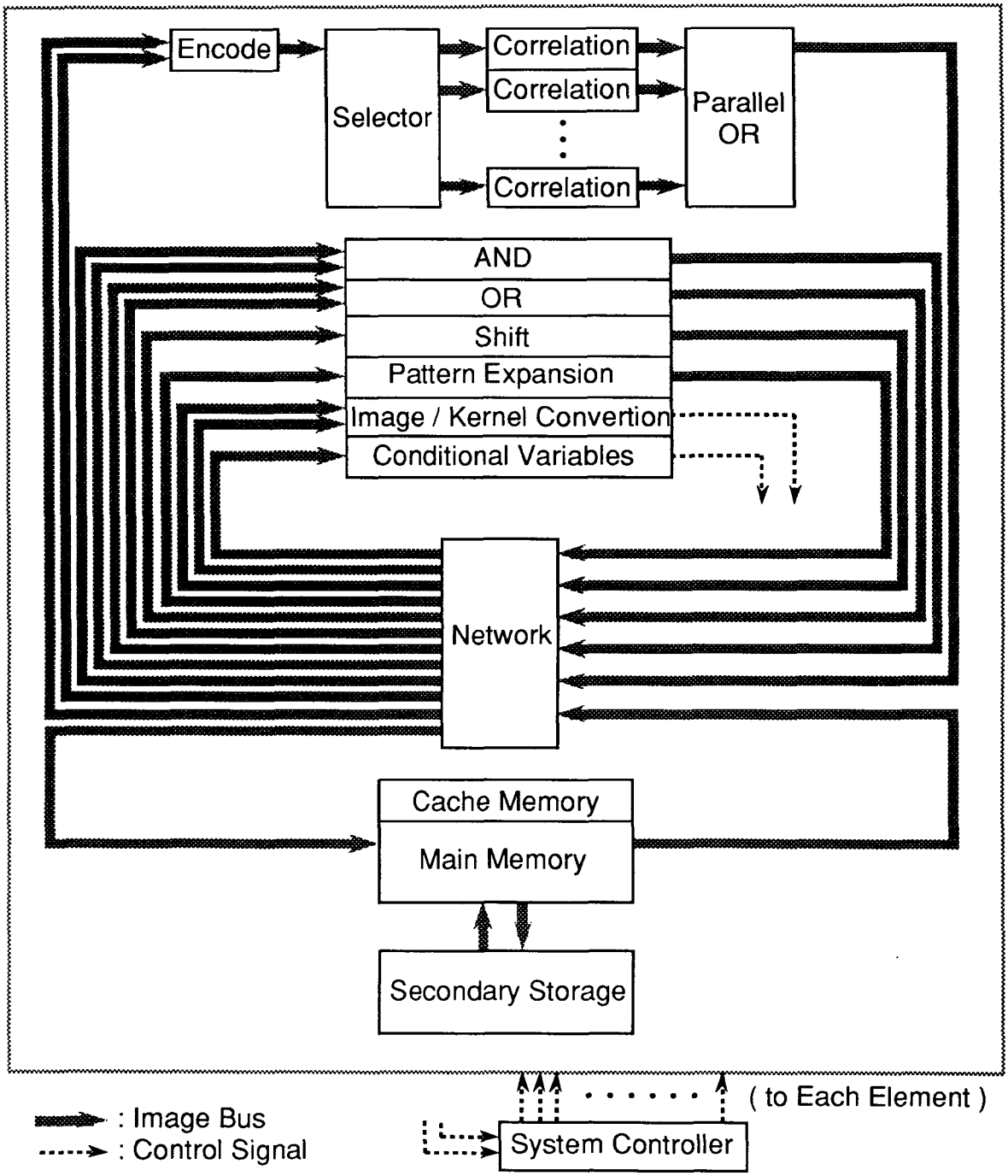


Fig.6.12　画像伝送ネットワークによるモジュール間接続方式を用いた並列光演算システムのブロック図

次に，各構成要素について説明する．論理和，論理積，シフト演算，パターン展開，状態変数の各モジュールは，6.3節で述べたものである．並列論理和モジュールは，多数枚の入力画像に対し対応画素間論理和を行う．これは，2入力論理和の光学系で実現できる．画像／カーネル変換モジュールは，6.3節のカーネル生成モジュールの出力を演算カーネルの電気信号で出力する．また，状態変数モジュールも，その結果を電気信号で出力する．これらの電気信号はシステムコントローラに送られる．システムコントローラは，OALLプログラム（1.4節）に基づいてシステムの各構成要素を制御する．これには電子コンピュータを使用し，制御信号は各構成要素に電気的に送られる．

メモリは，そのアクセス速度に応じてキャッシュメモリとメインメモリの2種類を用意する．キャッシュメモリは，6.4節におけるレジスタに相当する．これは，メインメモリに比べてアクセス速度が速いが，記憶容量は少ない．また，キャッシュメモリはアドレスでアクセスするが，メインメモリは画像名でアクセスできるとする．具体的には，キャッシュメモリは画像一時記憶素子としてSLMを数個用意し，メインメモリには光アドレス可能メモリ[21]などを使用する方式が考えられる．メインメモリには最低1プログラムで使用する画像20〜30枚程度を記憶できる容量があればよいと考えられる．SLM，光メモリと入出力画像バス間の接続には，偏光ビームスプリッタを用いた光バスネットワーク[69, 70]が利用できる．さらに，大容量画像記憶装置として2次記憶装置を考える．これは，画像データをビット列として逐次的にアクセスするものでよく，光磁気ディスク[67]などが考えられる．

この構成法では，各モジュール間の接続で必ずネットワークを通らなければならないため，ネットワーク接続パターンの変更速度が処理のボトルネックとなることが考えられる．また，画像サイズや入出力ポート数が増加するにつれて，ネットワークの光学的実現は一般的に難しくなる．さらに，6.4節より，光アレイロジック処理関係のモジュール以外は並列に接続する必要がほとんどなく，全モジュール間で一様に接続効率が良い必要はないと言える．このことは，処理対象の性質が任意接続可能なネットワークの特長を有効利用しておらず，モジュール間接続の簡略化が可能であることを示している．

6.5.2　共有画像バスによるモジュール間接続方式

以上の考察を踏まえ，各モジュール間をネットワークで接続する代わりに，共有画像バスで接続するシステム構成法を考案した．ブロック図をFig.6.13に示す．まず，メモリから2本の画像バスを出力する．画像バスは，メモリから読み出した画像を光信号として伝送し，各モジュールに分配する．光信号の分配による光量の低下を防ぐため，モジュール数はできるだけ少なくする．モジュールとしては，光アレイロジック処理関係のモジュール，論理積／論理和，シフト演算／パターン展開，画像／カーネル変換の各モジュールを用意する．各モジュールから1本の画像バスを出力し，それらを1本の画像バスにまとめ，メモリと状態変数モジュールに入力する．このように共有バス方式にすると，ネットワーク接続パターンの切り替えが不要になり，より高速な画像伝送が可能になる．

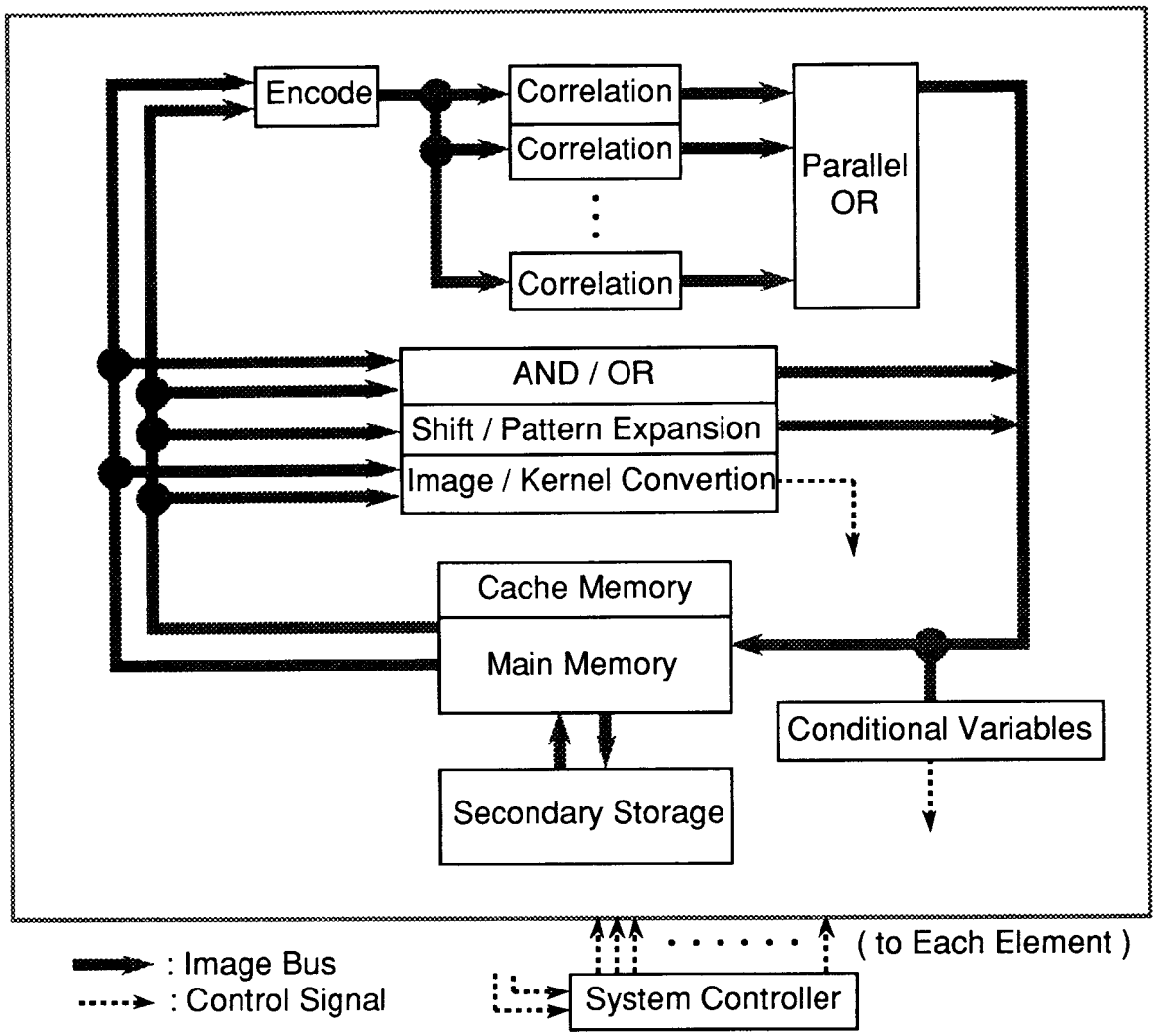


Fig.6.13　共有画像バスによるモジュール間接続方式を用いた並列光演算システムのブロック図

次に，各構成要素について説明する．画像バスの分岐部・結合部は，ビームスプリッタで光信号の分配・結合を行う．論理積／論理和モジュールとシフト演算／パターン展開モジュールでは，それぞれ一方の演算を選択できる．論理積／論理和モジュールは，しきい値素子のしきい値の変更により，演算を切り替えることができる．また，シフト演算はパターン展開に含まれる演算なので，パターン展開モジュールを用いてシフト演算も実行できる．他のモジュールはFig.6.12と同様である．

6.6　考察

Fig.6.13のシステムで問題となるのは，共有バス方式に起因するバス衝突である [10, 68]．すなわち，数個のモジュールが同時にメモリをアクセスすることはできない．しかし，6.4節の結果より，モジュールが並列に駆動する処理は，光アレイロジック処理手順以外ではほとんどないことがわかる．これは，光アレイロジックプログラムがSIMD型の処理用に書かれており，命令の並列性を考慮して書かれていないためである．したがって，相関演算モジュールが並列駆動可能ならば，ほとんどバス衝突は起こらないと考えられる．光アレイロジックモジュールは，相関演算モジュールを並列に駆動で

き，さらに符号化－相関演算－並列論理和の各モジュールが直列に接続されているため，パイプライン方式の並列処理 [10] が可能となる．

6.4節の結果より，各モジュール間接続の出現頻度にはある傾向が見られる．Table 6.6 は，モジュール間接続パターンにおけるモジュール間接続回数をまとめたものである．この表は，縦の項目から横の項目のモジュールへの接続出現回数を示している．この表より，光アレイロジックモジュール内の接続がとび抜けて多いことがわかる．次いで，レジスタ・メモリから各モジュール，各モジュールからレジスタへの転送が多い．また，各モジュール間を，レジスタ・メモリを介さずに直接接続する処理も多い．一方，各モジュールからメモリへの転送は，処理終了時に結果を転送する場合に現われる．Fig.6.13 のシステムは，光アレイロジックモジュール内を固定接続にし，レジスタ・メモリと各モジュール間を直接接続しているため，接続頻度の高い接続を効率良く行える．問題点としては，直接各モジュール間を接続したい場合でも，レジスタ・メモリを介さなければいけないことがある．これに対しては，メモリ部に画像の通り抜け（スルー）の伝送路を作り，高速に接続できるようにすればよい．

光信号に対し共有バス方式をとる場合，各モジュールへの信号のファンアウトにより信号の光量が低下する問題が生じる．本システムでは，符号化－相関演算モジュール間で，信号のファンアウト数が最大となる．そのファンアウト数は相関演算モジュール数に等しく，具体的には5程度を想定している．

Table 6.6　モジュール間接続パターンにおけるモジュール間接続回数

↗	S	PE	AND	OR	E	C	POR	CND	K	R	M
S		3	1							1	
PE		2	3		3					5	1
AND		1			4			1		3	1
OR			1		1					2	
E						57				1	
C			1		4		53	1		4	
POR		1		1	5		1	3	1	5	1
CND											
K											
R	1	1	3	3	19	5	1		1		1
M	2		3		10					8	

PE : Pattern Expansion Module
S : Shift Operation Module
AND : Logical AND Module
OR : Logical OR Module
E : Encoding Module
C : Correlation Module
POR : Parallel Logical OR Module
CND : Conditional Variable Module
K : Kernel Pattern Generating Module
R : Register
M : Main Memory

6.2節で検討したように，光アレイロジック処理において専用光演算モジュールを利用すると，ハードウェアに対する要求を軽減することができる．しかし，この方法はカーネルサイズの縮小に対して効果があるのみで，画像サイズを縮小させることはできない．画像サイズの縮小には，2次元仮想記憶機構（第5章）が利用できる．特に，光アレイロジックモジュールでは，符号化画像を扱う必要上，他の専用モジュールの4倍の画素数を必要とするため，2次元仮想記憶機構の組み込みが特に有効である．2次元仮想記憶機構により実行不可能な大きさの画像を扱える反面，処理が複雑になり，ステップ数が増加する問題が新たに生じる．

6.7　結言

本章では，専用光演算モジュールを用いた並列光演算システムについて考察した．まず，今までに開発された光アレイロジックプログラムを解析し，使用回数の多い演算，またはシステムに厳しい要求をする演算の種類を調べた．次に，各演算それぞれを専用光演算モジュールで実行した場合の処理効率を評価し，従来の方法の処理効率と比較した．それらの結果に基づき，専用光演算モジュールを効率よく使用する並列光演算システムのアーキテクチャを考案した．本システムにより，従来の光アレイロジックによる光演算システムであるOPALSに比べ，ハードウェア資源に対する要求が緩くなる．さらに，相関演算モジュールに2次元仮想記憶機構を組み込むことにより，画素数の多い画像の処理が可能となる．

総括

本論文では，並列光演算原理である光アレイロジックを用いて，新たな応用分野における並列演算法の開発・評価を行った．さらに，並列光コンピュータにおけるハードウェアへの負荷を軽減する方法として，２次元仮想記憶機構と，専用光演算モジュールを用いた並列光演算システムの構成法を考案した．以下，本研究により得られた知見および成果を各章ごとに総括し，今後の研究課題を考察する．

第１章では，本研究における並列演算パラダイムである，光アレイロジックによる並列プログラミングについて概説した．まず，光アレイロジックの概念，処理手順，光学的実現方法を述べた．次に，プログラム記述法と，専用プログラミング言語であるOALLについて概説した．また，光アレイロジックの基本的な並列プログラミング技法を述べ，さらに，本研究において効率のよい処理を行うために，光アレイロジックに状態変数と画像／カーネル変換という手法を新たに導入した．

第２章では，光アレイロジックの並列性を知識ベースの探索に有効利用した並列推論機構の実現方法として，テンプレートマッチング法とトークン伝播法の二種類を考案した．さらに，推論機構の機能を拡張してエキスパートシステムの処理を実現する方法を検討した．そして，これらの処理の光アレイロジックプログラムを作成し，処理効率を評価した．その結果，テンプレートマッチング法では，ハードウェアに対する要求がゆるい面で優れるが，トークン伝播法のほうが，処理速度と機能の拡張性において優れていることを確認した．また，トークン伝播法を用いた並列エキスパートシステムが，推論機構の並列性を損なわずに実行できることを示した．

第３章では，光演算システムへの応用を目指して，光アレイロジックの並列性をトークンの並列転送に利用したデータフロー型処理を検討した．その結果，光アレイロジックにより，データフローグラフの大きさによらず一定の処理速度でトークン転送を実行できることを確認した．また，演算ノードの処理をMIMD方式で並列に実行するためには，データの代わりに画像名をトークンに乗せ，各演算ノードに専用光演算モジュールを割り当てる方法が有望であるとの知見を得た．

第４章では，光アレイロジックの並列性を大容量データ処理に有効利用する手法として，データベース処理を検討した．大容量データ処理の基本演算として，パターン展開を利用した大小比較とソーティングの演算プログラムを開発した．それらの演算を利用してデータベース処理の関係演算プログラムを開発し，その処理効率，必要なハードウェアについて評価した．その結果，光アレイロジックがデータベース処理に対して有用な技術となり得ることを確認した．ただし，実用的なデータベース処理システムでは，光アレイロジックが特に有効であるデータ検索，大小比較のみを光アレイロジックで行い，他の処理を電気的に行う方法が有望であるとの結論を得た．

第５章では，並列光演算システムにおいて，小規模のハードウェアで大画素数の画像に対する処理を実現する方法として，２次元仮想記憶機構を考案した．各種並列光演算原理の処理手順を共通に記述するために，相関演算モデルを導入し，そのモデルを用いて２次元仮想記憶機構の処理手順・処理効率を検討した．その結果，１）相関演算系に

おいて入力側を３×３ページに拡張してページごとに処理を行い，２）処理に要求される最大シフト量よりページサイズを大きくすることで処理効率が高められることを確認した．さらに，２次元仮想記憶機構を実現するシステムについて検討し，パイプラインページ転送が有効に利用できることを示した．

第６章では，光アレイロジックによる処理の効率を向上させるために，専用光演算モジュールを用いた並列光演算システムの構成法を考察した．既開発の光アレイロジックプログラムを解析し，使用回数の多い演算，またはシステムに厳しい要求をする演算の種類を調べた．次に，それらの演算をそれぞれ専用光演算モジュールで実行した場合の処理効率を評価し，従来の方法の処理効率と比較した．以上の結果を踏まえて，専用光演算モジュールを効率的に使用する並列光演算システムの構成法を考案した．このシステムにより，従来の光アレイロジックによる光演算システムに比べ，ハードウェアに対する負荷が少なくなる．さらに，相関演算モジュールに２次元仮想記憶機構を組み込むことにより，画素数の多い画像の処理が可能となるとの知見を得た．

本研究では，今までに光アレイロジックが適用されていなかった応用分野における並列演算法を新たに開発した．これらと，既開発の数値処理，画像処理，並列マシンの仮想的実現に対する並列演算法を加えると，並列処理の有効利用が考えられる応用分野をひととおり検討したことになる．これらの分野において光アレイロジックの有効性が確認されたため，光アレイロジックは汎用性に優れていると言える．その理由としては，３．１で述べたように，光アレイロジックでは再構成可能なプロセッサを仮想的に実現できることが考えられる．すなわち，処理の性質に最適な形で仮想プロセッサを画像上にマッピングできるため，処理内容に最適な専用マシンを自由に構成できる．ただし，各仮想プロセッサは光アレイロジック処理により駆動されるため，SIMD 方式でかつ細粒度の処理しか実行できない．したがって，この形式に適合しない処理には，光アレイロジックは有効でない．

光アレイロジックは汎用性に優れるが，特に適用性の高い応用分野は画像処理である．これは，光アレイロジックが直接画像を処理するため，当然と考えられる．次に適用性の高いのが，本研究で行った，推論機構，データベース処理，データフロー型処理におけるトークン転送などの記号処理である．光アレイロジックは２値画素パターンの操作を基本とする処理である．したがって，記号を扱う処理は，記号を２値画素パターンに変換することで，比較的簡単に光アレイロジックに適合できる．数値処理は，整数演算などの簡単な処理ならば効率的に行えるが，浮動小数点演算など，複雑な処理になると，光アレイロジックにおける細粒度・SIMD 方式の処理には向いていない．以上より，光アレイロジックの有効な応用分野としては，画像処理，記号処理，簡単な数値処理が有望と考えられる．

本研究では，並列演算法開発に最も適した並列光演算原理として光アレイロジックを取り上げた．しかし，光アレイロジック選択の正当性を示すためには，他の並列光演算原理との比較が必要である．この点に関しては，並列光演算原理として活発に研究が行われている記号置換と，光アレイロジックそれぞれで開発された並列演算法間の比較に

よる評価が行われている．並列光演算法間の比較には，各並列光演算原理のプログラムを共通に記述できる言語である POPLAR (Parallel Optical Processing LAnguage for Research) [71] が使用されている．光アレイロジックと記号置換それぞれで開発されたプログラムを POPLAR で共通に記述し，比較した結果，より複雑なパターン操作を必要とする並列処理において，光アレイロジックのほうが処理効率がよいことが確認されている [72]．したがって，あらゆる応用分野において並列光演算の可能性を調べるためには，複雑な処理においても並列性を有効に発揮できる光アレイロジックが適していると言える．

並列光演算の第一の特長はその並列性にある．したがって，処理可能な画素数が多いほど並列性を利用して高速に処理することができる．本研究で考案した手法は，光の並列性をできる限り有効に利用することを基本とするため，処理画像は多くの画素数を必要とする．その結果，システムには多くの画素数が扱える光学素子が必要となり，光学的に実現する際の問題点となる．この問題を解決するため，第５，６章で行ったハードウェアに対する要求軽減の手法が有用となる．ただし，光の並列性を有効に利用するためには，光アレイロジック処理手順にしたがって画像を完全並列に処理できることが望ましい．したがって，多くの画像を一度に処理できる光アレイロジックシステムの開発は重要である．

処理画像が多くの画素数を必要とする問題に対する解決法として，処理データの画像への符号化方法の工夫による方法が考えられる．本研究におけるデータの符号化法では，データを表す画素パターンを一枚の画像上に配置して表す．この方法をビット展開型符号化法と呼ぶ．一方，データを表す画素パターンを，１～数画素ずつ，数枚の画像に分割して配置する方法がある．これをビットスライス型符号化法と呼ぶ．ビット展開型は，光の並列性を最大限に利用できる反面，処理画像に多くの画素数を必要とする．これに対し，ビットスライス型は，データが数枚の画像にわたっているため，処理の並列性は低下するが，処理画像に必要な画素数は小さくなる．すなわち，ビットスライス型は，ビット展開型における処理空間を小さくし，その分，処理時間を多く取る方法である．本研究では，ビット展開型符号化法を扱った．しかし，この場合処理画像の画素数が多いことが問題となるため，今後ビットスライス型符号化法の有効性を検討する必要がある．

残念ながら，実用的な光コンピュータはまだ開発されていない．その理由として，多くの画素数を扱える高速な光学素子が開発されていないことが挙げられる．しかし，その背景には，技術的な問題もさることながら，光コンピュータの有効な応用分野が明確でなく，光学素子やハードウェア開発の目標が定まりにくいことにも原因があると考えられる．すなわち，光コンピュータの研究では，まず並列演算法の開発により，光コンピュータの応用分野，将来像を明確にすることが必要である．本研究は，画素数の多い画像を扱うことにより，エキスパートシステムやデータフロー型処理，データベース処理といった複雑な処理も，光の並列性を有効利用して効率よく実行できることを確認し，処理に必要なハードウェア資源を具体的に示した点で意義がある．並列演算法の研究をさらに進めることにより，光学素子やハードウェア開発の指針が明確になり，光コンピュータの実現がより速くなることを期待する．

謝辞

本研究は，大阪大学工学部応用物理学科において，一岡芳樹教授の御指導の下に行ったものです．終わりに臨み，終始懇切丁寧な御指導と御助言を頂きました一岡芳樹教授に深く感謝の意を表しますと共に厚く御礼申し上げます．

本学工学部樹下行三教授ならびに本学産業科学研究所豊田順一教授には，本論文作成にあたり細部にわたって御検討頂き，貴重な御意見を頂きました．ここに深く感謝致します．

本学工学部伊東一良助教授には，常に有益な御指導，御指摘を頂きました．心から御礼申し上げます．

本学工学部講師谷田純博士には，本研究を遂行するにあたり終始親切な御指導，御助言を頂き，また本論文を細部にわたり御検討いただきました．ここに深く感謝致します．

本学工学部助手井上卓氏には，日頃から親切な御指導，御指摘を頂きました．深く御礼申し上げます．

一岡研究室卒業生の福井将樹氏（現日本電信電話株式会社）には，本研究で用いましたワークステーション用ソフトウェアに関して多大なる御協力を頂きました．ここに厚く御礼申し上げます．

一岡研究室の谷口正樹氏，長谷川玲氏には，本研究遂行に当たり常に有益な御助言，御討論を頂き，また本論文作成にあたり貴重な御指摘を頂きました．ここに深く感謝致します．

本論文作成にあたり，細部にわたり御検討，御助言を頂きました一岡研究室の宮崎大介氏，津村徳道氏，河野努氏に厚く御礼申し上げます．

本研究の遂行にあたり，日本育英会より奨学金を，日本学術振興会より研究奨励金を，文部省より科学研究費補助金を受けました．ここに深く感謝致します．

最後に，本研究遂行にわたり，終始御協力と御援助を頂きました研究室内外の方々に深く感謝致します．

Appendix A　並列演算法のカーネル式

A.1　推論機構（テンプレートマッチング法）

1. 画素パターンから演算カーネルのパターンへの変換 (1)（Fig.2.6(c)）

$$\sum_{i=0}^{PN+PL-1} \left([1.]_{0,\,-2i}[.1]_{0,\,-i} + [1.]_{0,\,-2i-1}[.0]_{0,\,-i-1} \right). \tag{A-1}$$

2. 画素パターンから演算カーネルのパターンへの変換 (2)（Fig.2.6(d)）

$$[1.] + [1.]_{-1,\,0}. \tag{A-2}$$

3. 画素パターンから演算カーネルのパターンへの変換 (3)（Fig.2.6(e)）

$$[1.] + [1.]_{0,\,-1} + [.1]. \tag{A-3}$$

4. テンプレートマッチング（Fig.2.6(f)）
   3.で作成された演算カーネルにより演算を行う.

5. 推論１ステップで導出されたパターンの出力（Fig.2.6(g), (h)）

$$\sum_{i=0}^{PN-1} [1.]_{0,\,-i}[.1]_{0,\,PN+PL-1} + \sum_{i=0}^{PL-1} [1.]_{-PN-i}[.1]. \tag{A-4}$$

PN：１ノードを表す画素パターンの画素数
PL：１リンクを表す画素パターンの画素数

A.2　推論機構（トークン伝播法）

1. 縦方向トークン伝播（Fig.2.9）
   1）縦方向パターン展開（入力画像 A: Query (1), 入力画像 B: don't care, 出力画像 C: Expanded Image (1)）

$$\prod_{i=-S+1}^{S-1} [1.]_{4i,\,0}\,;\ [0.]. \tag{A-5}$$

   2）知識ベース画像から条件画像を作成（入力画像 A: Knowledge Base, 入力画像 B: Attribute Plane of Knowledge Base (Fig.2.8(f)), 出力画像 C: Condition Image (1)）

$$[11]+[.1\ \underline{10}]+\begin{bmatrix}\underline{.1}\\ 10\end{bmatrix}+\begin{bmatrix}.1\ ..\\ ..\ \underline{10}\end{bmatrix}. \tag{A-6}$$

3）テンプレートマッチング（入力画像 A: Expanded Image (1), 入力画像 B: Condition Image (1), 出力画像 C: Query (1')）

$$\sum_{i=0}^{3}\left(\begin{bmatrix}\underline{00} & 11\\ EE & EE\end{bmatrix}_{0,-i}+\begin{bmatrix}00 & \underline{11}\\ EE & EE\end{bmatrix}_{0,-i}\right). \tag{A-7}$$

2．トークン上のリンク方向画素の変更（Fig.2.10）（入力画像 A: Query (1'), 入力画像 B: don't care, 出力画像 C: Query (2)）

$$[\underline{11}\,00]. \tag{A-8}$$

3．横方向トークン伝播（Fig.2.11）

1）横方向パターン展開（入力画像 A: Query (2), 入力画像 B: don't care, 出力画像 C: Expanded Image (2)）

$$\prod_{i=-N+1}^{N-1}[0.]_{0,2i};\ [0.]. \tag{A-9}$$

2）知識ベース画像から条件画像を作成（入力画像 A: Knowledge Base, 入力画像 B: Attribute Plane of Knowledge Base, 出力画像 C: Condition Image (2)）

$$[11]+[.1\ \underline{10}]. \tag{A-10}$$

3）テンプレートマッチング（入力画像 A: Expanded Image (2), 入力画像 B: Condition Image (2), 出力画像 C: Query (2')）

$$\sum_{i=0}^{3}\left([\underline{11}\,00]_{0,-i}+[11\,\underline{00}]_{0,-i}\right). \tag{A-11}$$

4．トークンのリンク方向画素の変更（入力画像 A: Query (1'), 入力画像 B: don't care, 出力画像 C: Query (3)）

$$[01\,\underline{10}]. \tag{A-12}$$

S：サブネットワーク数，N：ノード数

';'：この記号の前後の命令を連続して行う．

A.3　データフロー型処理

1．縦方向トークン伝播（Fig.3.5）

1）縦方向パターン展開（入力画像 A: Token (1), 入力画像 B: don't care, 出力画像 C: Expanded Token (1)）

$$\prod_{i=-S+1}^{S-1} [0.]_{5i,\,0}\,;\ [0.]. \tag{A-13}$$

2）アーク方向画素の出力（入力画像 A: DFGraph (1), 入力画像 B: DFGraph Attribute (Fig.3.4(e)), 出力画像 C: Condition Image）

$$[\underline{11}\,00] + [01\,\underline{10}]. \tag{A-14}$$

3）テンプレートマッチング（入力画像 A: Expanded Token (1), 入力画像 B: Condition Image, 出力画像 C: Token (1')）

$$[\underline{11}\,00] + \sum_{i=1}^{4}\sum_{j=0}^{B+1} [\underline{11}\,00]_{-i,\,-j}[1.]. \tag{A-15}$$

2．トークンのアーク方向画素の変更（Fig.3.6）（入力画像 A: Token (1'), 入力画像 B: DFGraph Attribute, 出力画像 C: Token (2)）

$$[11\,\underline{00}] + \sum_{i=1}^{4}\sum_{j=0}^{B+1} [1.][.1]_{-i,\,-j}. \tag{A-16}$$

3．横方向トークン伝播（Fig.3.7）

1）横方向パターン展開（入力画像 A: Token (2), 入力画像 B: don't care, 出力画像 C: Expanded Token (2)）

$$\prod_{i=-N+1}^{N-1} [0.]_{0,\,7i}\,;\ [0.]. \tag{A-17}$$

2）テンプレートマッチング（入力画像 A: Expanded Token (2), 入力画像 B: Condition Image, 出力画像 C: Token (2')）

$$[00\,\underline{11}] + \sum_{i=1}^{4}\sum_{j=0}^{B+1} [\underline{00}\,11]_{-i,\,-j}[1.]. \tag{A-18}$$

4．伝播したトークンの記録と発火したノードの検出（Fig.3.8）

1）論理和（入力画像 A: DFGraph (1), 入力画像 B: Token (2'), 出力画像 C: DFGraph (2)）

$$[\mathrm{PP}]. \tag{A-19}$$

2）テンプレートマッチング（入力画像 A: DFGraph (2), 入力画像 B: DFGraph Attribute, 出力画像 C: Fired Token (1)）

$$\begin{bmatrix} .1 \\ \underline{1.} \\ 1. \\ 1. \\ 1. \end{bmatrix} + \begin{bmatrix} .1 \\ \underline{1.} \\ 1. \\ 0. \\ 0. \end{bmatrix}. \tag{A-20}$$

5．発火したノードの演算（Fig.3.9）

1）演算データの出力（入力画像 A: DFGraph (2), 入力画像 B: Fired Token (1), 出力画像 C: Calculated Data (1)）

$$\sum_{i=0}^{4} \sum_{j=1}^{B} [1.][.1]_{-i,\,-j}. \tag{A-21}$$

2）MSD 加減算

省略（Appendix B．4 を参照）

6．発火したノードに記録されているトークンの消去（Fig.3.10）（入力画像 A: DFGraph (2), 入力画像 B: Fired Token (1), 出力画像 C: DFGraph (3)）

$$\sum_{i=0}^{3} [1.]\left([.1]_{-i,\,0} + [.1]_{-i,\,-B-1}\right) + \sum_{j=0}^{B+1} [1.][.1]_{1,\,-j}. \tag{A-22}$$

7．演算結果の指定位置へのセット（Fig.3.11）

1）出力位置画素の下半分を出力（入力画像 A: DFGraph (2), 入力画像 B: Fired Token (1), 出力画像 C: Output Location (1)）

$$\sum_{j=2}^{3} [1.][.1]_{-i,\,-B-1}. \tag{A-23}$$

2）条件つきシフト（入力画像 A: Output Location (1), 入力画像 B: Calculation Result (1), 出力画像 C: Calculation Result (1')）

$$\sum_{i=0}^{1}\sum_{j=2}^{3}\begin{bmatrix}1.\\0.\end{bmatrix}_{-i,\,-j}[.1]_{-2,\,0}+\begin{bmatrix}0.\\0.\end{bmatrix}_{-i+2,\,j}[.1]. \tag{A-24}$$

8．新たなトークンの出力（Fig.3.12）

1）アーク方向画素とトークン記録画素の出力（入力画像 A: DFGraph (1)，入力画像 B: Fired Token (1)，出力画像 C: Token Temp）

$$[1.]\left([.1]_{1,\,0}+[.1]_{1,\,-1}+\sum_{i=0}^{3}[.1]_{-i,\,0}\right). \tag{A-25}$$

2）論理和（入力画像 A: Token Temp，入力画像 B: Calculation Result (1')，出力画像 C: Token (3)）

$$[PP]. \tag{A-26}$$

S：サブネットワーク数，B：データのビット数，N：ノード数

';'：この記号の前後の命令を連続して行う．

A.4　奇偶マージソート

1．大小比較（Fig.4.7(a)）

1）データ転送

A. 大小比較を行わないデータの消去（入力画像 A: Data (1)，入力画像 B: Attribute 1 (1)，出力画像 C: Data Temp 1）

$$[01]. \tag{A-27}$$

B. データプレーンのデータ転送領域（Fig.4.5 (a)）にデータを転送（入力画像 A: Data Temp 1，入力画像 B: Attribute 3 (1)，出力画像 C: Data Temp 2）

$$[11]+[01]_{0,\,-B}. \tag{A-28}$$

C. データを上または下に転送（入力画像 A: Data Temp 2，入力画像 B: Attribute 2 (1)，出力画像 C: Data Temp 3）

$$[11]_{-S,\,0}+[10]_{S,\,0}. \tag{A-29}$$

D. 論理和（入力画像 A: Data Temp 2, 入力画像 B: Data Temp 3, 出力画像 C: Data (1')）

$$[PP]_{\cdot} \tag{A-30}$$

2）大小比較ステップ1（入力画像 A: Data (1'), 入力画像 B: Attribute 5 (Fig.A.1), 出力画像 C: Comparison Temp）

$$[11][00]_{0,\,B} + [10]_{0,\,-B}[01]_{\cdot} \tag{A-31}$$

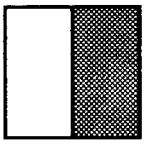


Fig. A.1　大小比較ステップ1のアトリビュートプレーン

3）大小比較ステップ2

A. 比較する2データの左側を出力する行を選択（入力画像 A: Attribute 4 (1), 入力画像 B: Comparison Temp, 出力画像 C: Compared Result L (1)）

$$\sum_{i=0}^{B-1}\left([1.\,1.][.1]_{0,\,i}\prod_{j=0}^{i-1}[.1]_{0,\,j}\prod_{k=B}^{B+i}[00]_{0,\,k}+[1.\,0.\,1.]\prod_{j=0}^{i}[.0]_{0,\,j}[01]_{0,\,B+i}\prod_{k=B}^{B+i-1}[00]_{0,\,k}\right) \tag{A-32}$$

B. 比較する2データの右側を出力する行を選択（入力画像 A: Attribute 4 (1), 入力画像 B: Comparison Temp, 出力画像 C: Compared Result R (1)）

$$\sum_{i=0}^{B-1}\left([1.\,1.]\prod_{j=0}^{i}[.0]_{0,\,j}[01]_{0,\,B+i}\prod_{k=B}^{B+i-1}[00]_{0,\,k}+[1.\,0.\,1.][.1]_{0,\,i}\prod_{j=0}^{i-1}[.0]_{0,\,j}\prod_{k=B}^{B+i}[00]_{0,\,k}\right) \tag{A-33}$$

2. 置換（Fig.4.7 (b)）

1）パターン展開

（入力画像 A: Compared Result L (1), 入力画像 B: don't care, 出力画像 C: Expanded Result L (1)）

（入力画像 A: Compared Result R (1), 入力画像 B: don't care, 出力画像 C: Expanded Result R (1)）

$$\sum_{i=0}^{B-1} [0.]_{0,-i} ; [0.] . \tag{A-34}$$

2）指定されたデータの出力

A. 比較した2データの左側を出力（入力画像 A: Data (1'), 入力画像 B: Expanded Result L (1), 出力画像 C: Left Data (1)）

$$[11] . \tag{A-35}$$

B. 比較した2データの右側を出力（入力画像 A: Data (1'), 入力画像 B: Expanded Result R (1), 出力画像 C: Right Data (1)）

$$[1.][.1]_{0,B} . \tag{A-36}$$

C. ソート結果の出力（入力画像 A: Left Data (1), 入力画像 B: Right Data (1), 出力画像 C: Data (2)）

$$[PP] . \tag{A-37}$$

S：サブネットワーク数，B：データのビット数，N：ノード数

' ; '：この記号の前後の命令を連続して行う．

A.5　データベース処理（選択演算）

1．パターン展開（Fig.4.10）（入力画像 A: Condition, 入力画像 B: don't care, 出力画像 C: Expanded Condition）

$$\prod_{i=0}^{N-1} [0.]_{i,0} ; [0.] . \tag{A-38}$$

2．大小比較＋パターン展開（Fig.4.10）

1）大小比較ステップ1（入力画像 A: Database, 入力画像 B: Expanded Condition, 出力画像 C: Comparison Temp 1）

$$[UU] . \tag{A-39}$$

2）大小比較ステップ2

A. ステップ1で検出した最上位ビットを出力（入力画像 A: Attribute (Fig.A.2), 入

力画像 B: Comparison Temp 1，出力画像 C: Comparison Temp 2）

$$\sum_{i=0}^{S-1}[1.][.1]_{0,i}\prod_{j=0}^{i-1}[.0]_{0,j}. \tag{A-40}$$

B. 検出した最上位ビットの大小比較（入力画像 A: Database，入力画像 B: Comparison Temp 2，出力画像 C: Comparison Result）

・入力画像 A のデータが小さい場合を検出

$$[01]. \tag{A-41}$$

・入力画像 A のデータが大きい場合を検出

$$[11]. \tag{A-42}$$

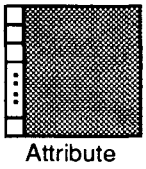


Fig.A.2　データベース処理のアトリビュートプレーン

3）パターン展開（入力画像 A: Comparison Result，入力画像 B: don't care，出力画像 C: Comparison Result）

$$\prod_{i=0}^{MS}[0.]_{0,-i};[0.]. \tag{A-43}$$

3. 論理和（Fig.4.10）（入力画像 A: Database，入力画像 B: Comparison Result，出力画像 C: Final Result）

$$[11]. \tag{A-44}$$

M：属性数，N：タプル数，S：データのビット数

';'：この記号の前後の命令を連続して行う.

A.6　データベース処理（射影演算）

1. 論理和（Fig.4.11）（入力画像 A: Database (1)，入力画像 B: Condition Area (1)，出力画像 C: Condition (1)）

$$[11]. \tag{A-45}$$

2．パターン展開（Fig.4.11）（入力画像 A: Condition (1), 入力画像 B: don't care, 出力画像 C: Expanded Condition (1)）

$$\prod_{i=1}^{N-1} [0.]_{i,0}[0.]_{-i,0}\,;[0.]. \tag{A-46}$$

3．テンプレートマッチング＋パターン展開（Fig.4.11）

1）テンプレートマッチング・ステップ1（入力画像 A: Database (1), 入力画像 B: Expanded Condition (1), 出力画像 C: Match Area (1)）

$$\prod_{i=0}^{S-1} [EE]_{0,i}. \tag{A-47}$$

2）テンプレートマッチング・ステップ2（入力画像 A: Attribute (Fig.A.2), 入力画像 B: Match Area (1), 出力画像 C: Match Area (1')）

$$[11]. \tag{A-48}$$

3）パターン展開（入力画像 A: Match Area (1'), 入力画像 B: don't care, 出力画像 C: Matching Result (1)）

$$\prod_{i=0}^{S-1} [0.]_{0,i}\,;[0.]. \tag{A-49}$$

4．パターン消去（入力画像 A: Database (1), 入力画像 B: Matching Result (1), 出力画像 C: Database (2)）

$$[01]. \tag{A-50}$$

N：タプル数，S：データのビット数

';'：この記号の前後の命令を連続して行う．

Appendix B　並列演算法の OALL プログラム

B.1　推論機構（テンプレートマッチング法）

```
/*
        Inference with semantic network (Template matching method)
        ver. 1.0  1989-09-08  M.Iwata
        ver. 2.0  1989-09-16  M.Iwata
        ver. 3.0  1989-10-09  M.Iwata
        ver. 3.1  1990-02-08  M.Iwata
*/

program matchsn3;

kernel query,query2,templet,makekp1,makekp2,makekp3,mdetect,down,and,detectN;
kernel or,a,clrImg;
image  kbaseA = ../../image/matchsn/matchsn3-jjap.attr;
image  kbaseB = ../../image/matchsn/matchsn3-jjap.data;
image  queryA = ../../image/matchsn/matchsn3-jjapq.attr;
image  queryB = ../../image/matchsn/matchsn3-jjapq.data;
image  kqueryB,kernelpt,detect,detectR,detectT,result,resultA;
var    inf,node,link,matchbit,i,j,onpixel,offimg,pterm;

inf = 2;
node = 2;      /* node = node number - 1 */
link = 1;      /* link = link number - 1 */
matchbit = node + link + 1;

/* make kernel patterns */
for j = 0 to matchbit do
       makekp1 = makekp1
           + |1.|@(0,-2*j) * |.1|@(0,-j)
           + |1.|@(0,-2*j-1) * |.0|@(0,-j-1);
end;

makekp2 = <| .0|
           |_.0|>;

makekp3 = |1.| + |1._..| + |.1|;

/* set templet of the next inference */
for i = 0 to node do
       templet = templet + |1.|@(0,-i) * |.1|@(0,matchbit+1);
end;
for i = 0 to link do
       templet = templet + |1.|@(0,-node-i-1) * |.1|;
end;

/* detect node */
for i = 0 to matchbit do
        detectN = detectN + |1.|@(0,-i) * |.1|;
end;

/* detect the top of the rules in knowledge base */
mdetect = |_.0'
          | .1|;

/* down 1 pixels */
down = | .1|
       |_..|;

/* logical and */
and = |11|;

/* logical or */
or = |PP|;

/* output plane A */
a = |1..;

i = 1;

/* clear image */
clrImg = |DD|;
result = exec(kbaseA,kbaseA,clrImg);
```

```
resultA = result;

/* count number of query */
queryA = exec(queryA,queryA,a);
onpixel = condition(Number);

detect = exec(kbaseA,kbaseA,mdetect);
loop
        if i > inf then
            exit;
        else
            i = i + 1;
        end;

        /* convert image to kernel pattern */

        pterm = 1;
        detectT = detect;
        query = null;
        if onpixel > 1 then
            loop
               detectT = exec(detectT,detectT,down);
               detectR = exec(queryA,detectT,and);
               offimg = condition(Zero);
               if offimg = 0 then
                   kqueryB = exec(detectR,queryB,detectN);
                   kernelpt = exec(detectR,kqueryB,makekp1);
                   kernelpt =!exec(kernelpt,kernelpt,makekp2);
                   kernelpt = exec(detectR,kernelpt,makekp3);
                   query2 = kernel(detectR,kernelpt);
                   query = query + query2;
                   if pterm = onpixel then
                      exit;
                   else
                      pterm = pterm + 1;
                   end;
                end;
            end;
        else
            kernelpt  = exec(queryA,queryB,makekp1);
            kernelpt =!exec(kernelpt,kernelpt,makekp2);
            kernelpt = exec(queryA,kernelpt,makekp3);
            query = kernel(queryA,kernelpt);
        end;

        /* pattern matching */

        queryA = exec(kbaseA,kbaseB,query);

        onpixel = condition(Number);
        if onpixel = 0 then
            exit;
        end;

        queryB = exec(queryA,kbaseB,templet);
end;

imout matchsn3.result result;
imout matchsn3.data.result queryB;

end matchsn3;
```

B.2　推論機構（トークン伝播法）

```
/*
    Inference on a semantic network with pattern expansion
       ver.1.0  1989-11-21  Masaya Iwata
       ver.1.1  1989-11-22  Masaya Iwata
       ver.1.2  1990-01-10  Masaya Iwata
*/

program expandsn;

kernel expandH,expandV,detectRule,detectNode,selectRule,selectNode,clrImg;
kernel detectRule2,detectNode2,or;
image  knowledgeBaseAttr = ../../image/expandsn/expandsn-sr.attr;
```

Appendix B　並列演算法の OALL プログラム

```
image   knowledgeBase = ../../image/expandsn/expandsn-sr.data;
image   query = ../../image/expandsn/expandsn-srq.data;
image   infStart,infResult,infResult0,infResultM,expandedImage;
var     offImg,infStartNum,infNum,maxInfNum,rule,mrule,bit;
var     node,mnode,i,ibit,i2;

maxInfNum = 5; /* maximum inference number */
rule = 3;      /* rule number */
rule = rule - 1;
mrule = -rule;
node =  6;     /* node number */
node = node -1;
mnode = -node;
bit = 2;       /* bit number of a node */

/* expand a pixel horizontally */
expandH = <|0.|@(0,mnode*bit)>;
mnode = mnode + 1;
for i = mnode to node do
    ibit = i * bit;
    expandH = expandH * <|0.|@(0,ibit)>;
end;

/* expand a pixel vertically */
expandV = <|0.|@(mrule*2,0)>;
mrule = mrule + 1;
for i = mrule to rule do
    i2 = i * 2;
    expandV = expandV * <|0.|@(i2,0)>;
end;

/* detect rules that answer queries */
detectRule = |_11 00|
             |  .0 .0|;

/* detect nodes that are derived */
detectNode = |00_11|
             |EE EE|;

/* select data with attribute plane */
selectRule = |11|;

/* select node with attribute plane */
selectNode = |10_01|;

/* detect rules that answer queries (after selection) */
detectRule2 = |  11 00|
              |_10 .0| + |11 00|
                         |.0_10| + |.1_..|;

/* detect nodes that are derived (after selection)  */
detectNode2 = |  00 11|
              |_10 .0|  + |00 11|
                          |.0_10| + |.. .1|;

/* logical or */
or = |PP|;

/* clear image */
clrImg = |DD|;

infNum = 1;

/* make null image */
infResult = exec(query,query,clrImg);
infResultM = infResult;

/* inference loop */
loop
    /* detect rules that answer queries */
    expandedImage = exec(query,query,expandV);
    infStart = exec(expandedImage,knowledgeBase,detectRule);
    infStart   exec(knowledgeBaseAttr,infStart,selectRule);
    infStart = exec(expandedImage,infStart,detectRule2);
    offImg = condition(Zero);
    if offImg = 1 then
        exit;
    end;
```

```
    /* detect derived nodes */
    expandedImage =!exec(infStart,infStart,expandH);
    infResult0 = infResult;
    infResult = exec(expandedImage,knowledgeBase,detectNode);
    infResult = exec(knowledgeBaseAttr,infResult,selectNode);
    infResult = exec(expandedImage,infResult,detectNode2);
    offImg = condition(Zero);
    if offImg = 1 then
       infResult = infResult0;
        exit;
    end;
    query = infResult;
    infResultM = exec(infResultM,infResult,or);

    infNum = infNum + 1;
    if infNum > maxInfNum then
       exit;
    end;
end;

imout expandsn.result infResultM;
imout expandsn.data.result infResult;

end expandsn;
```

B.3　エキスパートシステム

```
/*
    Inference on a semantic network with pattern expansion
       ver.1.0  1989-11-21  Masaya Iwata
       ver.1.1  1989-11-22  Masaya Iwata
       ver.2.0  1989-12-18  Masaya Iwata
                 :append reverse propagation of markers
                 :append marker bits
       ver.2.1  1989-12-19  Masaya Iwata
       ver.2.2  1989-12-27  Masaya Iwata
       ver.2.3  1990-01-27  Masaya Iwata
                 :support inheritance
       ver.2.4  1990-07-12  Masaya Iwata
                 :support only backward inheritance
       ver.2.5  1990-07-19  Masaya Iwata
       ver.2.51 1990-08-23  Masaya Iwata
*/

program expandsnm4;

kernel expandH,expandV,detectNode,selectNode,clrImg;
kernel setNode,setMarker,or,makeMArea,makeLMArea,makeNode,setLM,detectNode2;
kernel detectAns,detectAns2,detectAns3,gatherMarkerA,gatherMarkerB;
kernel makekp1,makekp2,detectRN,setLink,makeAllMArea,detectRIsa;
kernel detectRNIsa1,detectRNIsa2,deleteNIsa,makeDeleteLink,makeNLArea;
image  knowledgeBaseAttr = ../../image/expandsn/expandsnmes-ao.attr;
image  knowledgeBase = ../../image/expandsn/expandsnmes-ao.data;
image  query0 = ../../image/expandsn/expandsnmesq-ao.data;
image  query,infStart,infResult,infResult0,expandedImage,newMarker,infDirect;
image  allMarkerA,allMarkerB,infStartLM,infResultLM,infStartNode,infResultNode;
image  isa,nIsa,expandedImageQ,deleteLink,derivedNode,infResultPoint,infResultM;
image  allMArea;
var    offImg,infStartNum,infNum,maxInfNum,rule,mrule,bitH,bitV,ibitH,ibitV;
var    bitLV,bitMV,bitMVm1,bitHm1,node,mnode,bitHT,bitVT,i,j,j2;
var    inheritance,bitLVm1;

/* variables settings */
maxInfNum = 5; /* maximum inference number */
rule = 9;      /* rule number */
rule = rule - 1;
mrule = -rule;
node =  19;    /* node number */
node = node -1;
mnode = -node;
bitH = 2;      /* horizontal bit number of a node */
bitLV = 1;     /* vertical bit number of a link */
bitMV = 2;     /* vertical bit number of marker bits */
bitV = bitLV + bitMV + 1;    /* vertical bit number of a node */
inheritance = 1;      /* support inheritance = 1, else = 0 */
```

```
/* expand a pixel horizontally */
expandH = <|0.|@(0,mnode*bitH)>;
mnode = mnode + 1;
for i = mnode to node do
    ibitH = i * bitH;
    expandH = expandH * <|0.|@(0,ibitH)>;
end;

/* expand a pixel vertically */
expandV = <|0.|@(mrule*bitV,0)>;
mrule = mrule + 1;
for i = mrule to rule do
    ibitV = i * bitV;
    expandV = expandV * <|0.|@(ibitV,0)>;
end;

/* logical or */
or = |PP|;

/* detect nodes that are derived */
detectNode = |11 00|;
detectNode2 = |00 11|;
for i = 1 to bitLV do
    detectNode = detectNode * |.0 .0|@(i,0);
    detectNode2 = detectNode2 * |EE EE|@(i,0);
end;
detectNode = detectNode + detectNode2;

/* select node with attribute plane */
selectNode = |11|;

/* set link and marker */
setLM = |11|;

/* make node pattern */
makeNode = |11_00| + |10 01|;

/* set nodes that are derived (after selection)  */
setNode = |PP|;

/* make marker area of selected nodes */
makeMArea = <|0. 0.|@(-bitLV-1,-1)>;
if bitMV > 1 then
    for i = 2 to bitMV do
        for j = 1 to bitH do
            makeMArea = makeMArea * <|0.|@(-bitLV-i,1-j)>;
        end;
    end;
end;

/* set marker */
setMarker = |11|;

/* make link and marker area of selected nodes */
makeLMArea = makeMArea;
for i = 1 to bitLV do
    for j = 1 to bitH do
       makeLMArea = makeLMArea * <|0.|@(-i,1-j)>;
    end;
end;

/* detect reverse isa link */
detectRIsa = |1. .1.
             |.1 .0|;
if bitLV > 1 then
    for i = 2 to bitLV do
       detectRIsa = detectRIsa * |.0 .0|@(i,0);
    end;
end;

/* detect reverse non-isa link */
detectRNIsa1 = |0. .1|;
detectRNIsa2 = |11|@(-1,0);


/* make area of deleted non-isa link */
makeDeleteLink =  <'.0 @(0,-1)>;
```

```
if bitLV > 1 then
    for i = 1 to bitLVm1 do
        for j = 0 to bitHm1 do
            makeDeleteLink = makeDeleteLink * <|.0|@(-i,-j)>;
        end;
    end;
end;

/* delete non-isa link */
deleteNIsa = |01|;

/* set isa link to non-isa link */
setLink = |1.|+ |.1|;

/* gather all markers */
bitVT = bitV * rule;
bitHT = bitH * node;
gatherMarkerA = |1.|@(bitVT-bitLV-1,0)
              * |1.|@(-bitLV-1,bitHT)
              * |1.|@(-bitLV-1,0);

gatherMarkerB = |11|;

/* make marker area of selected nodes */
makeAllMArea = <|0. 0.|@(0,-1)>;
if bitMV > 1 then
    for i = 2 to bitMV do
        for j = 1 to bitH do
            makeAllMArea = makeAllMArea * <|0.|@(1-i,1-j)>;
        end;
    end;
end;

/* make a image of a kernel pattern */
bitMVm1 = bitMV - 1;
for i = 0 to bitMVm1 do
    for j = 0 to bitHm1 do
        makekp1 = makekp1
                + |1.|@(-2*i,-2*j) * |.1|@(-i,-j)
                + |1.|@(-2*i,-2*j-1) * |.0|@(-i,-j-1);
    end;
end;

makekp2 = <| .0|
          |_.0|>;

/* detect answer with attribute plane */
detectAns2 = |1.| * |.1|@(bitLV+1,0);

/* make node and link area */
makeNLArea = <|0. 0.|@(0,-1)>;
for i = 1 to bitLV do
    makeNLArea = makeNLArea * <|0. 0.|@(-i,-1)>;
end;

/* set answer to the result image */
detectAns3 = |11|;

/* make null image */
clrImg = |DD|;
infResult = exec(query,query,clrImg);

query = query0;
infNum = 1;

/* inference loop */
loop
    /* detect rules that answer queries */
    expandedImage =!exec(query,query,expandV);
    infStart = exec(expandedImage,knowledgeBase,detectNode);
    infStart = exec(knowledgeBaseAttr,infStart,selectNode);
    offImg = condition(Zero);
    if offImg = 1 then
        exit;
    end;
    infStartLM =!exec(infStart,infStart,makeLMArea);
    infStartNode = exec(infStart,expandedImage,makeNode);
    infStart = exec(infStartLM,expandedImage,setLM);
```

```
    infStart = exec(infStart,infStartNode,setNode);

    /* detect derived nodes */
    infResult0 = infResult;
    expandedImage =!exec(infStart,infStart,expandH);
    infResult = exec(expandedImage,knowledgeBase,detectNode);
    infResultPoint = exec(knowledgeBaseAttr,infResult,selectNode);
    offImg = condition(Zero);
    if offImg = 1 then
       infResult = infResult0;
        exit;
    end;
    infResultLM =!exec(infResultPoint,infResultPoint,makeLMArea);
    infResultNode = exec(infResultPoint,expandedImage,makeNode);
    infResult = exec(infResultLM,expandedImage,setLM);
    infResult = exec(infResult,infResultNode,setNode);
    infResultM =!exec(infResultPoint,infResultPoint,makeMArea);
    newMarker = exec(infResultM,infResult,setMarker);
    knowledgeBase = exec(newMarker,knowledgeBase,or);
    query = infResult;

    /* inheritance operation */
    if inheritance = 1 then
       /* Reverse inheritance */
        if infNum = 1 then
           isa = exec(knowledgeBaseAttr,query,detectRIsa);
           nIsa = exec(isa,query,detectRNIsa1);
           nIsa = exec(knowledgeBaseAttr,nIsa,detectRNIsa2);
           deleteLink =!exec(nIsa,nIsa,makeDeleteLink);
           query = exec(deleteLink,query,deleteNIsa);
           query = exec(nIsa,query,setLink);
        end;
    end;

    infNum = infNum + 1;
    if infNum > maxInfNum then
       exit;
    end;
end;

/* derive answers by detecting all markers */
allMarkerA = exec(knowledgeBaseAttr,knowledgeBaseAttr,gatherMarkerA);
imout expandsnm4.data.1 allMarkerA;
allMArea =!exec(allMarkerA,allMarkerA,makeAllMArea);
expandedImage =!exec(query0,query0,expandH);
allMarkerB = exec(allMArea,expandedImage,gatherMarkerB);
allMarkerB = exec(allMarkerA,allMarkerB,makekp1);
allMarkerB =!exec(allMarkerB,allMarkerB,makekp2);
detectAns = kernel(allMarkerA,allMarkerB);
infResultPoint = exec(knowledgeBaseAttr,knowledgeBase,detectAns);
infResultPoint = exec(knowledgeBaseAttr,infResultPoint,detectAns2);
expandedImage =!exec(infResultPoint,infResultPoint,makeNLArea);
infResult = exec(expandedImage,knowledgeBase,detectAns3);

imout expandsnm4.data.result infResult;

end expandsnm4;
```

B.4　データフロー型処理

```
/*
    Dataflow machine for numerical operations
       ver.1.0  1990-06-18  Masaya Iwata
*/

program dflow;

kernel expandH,expandV,detectRule,detectNode,selectRule,selectNode,clrImg;
kernel detectRule2,detectNode2,detectToken,detectAnswer,detectMlt;
kernel or,and,clrSubNum,makeAddAttr,makeSubNumArea1,makeSubNumArea2,shiftD;
kernel makeMltAttr,makeMltArea,makeFlowStart,makeOutArea,clrOutArea;
kernel detectOut1,detectOut2,setOut,setToken;
kernel add1,add2,add3,ID,DshiftA,ULshiftA,RshiftB;
kernel multi1,multi1d,multi1d1,multi1d2,multi1d3,multi1d4;
kernel multi2k,multi2kd,multi2kd1,multi2kd2,multi2kd3,multi2kd4;
image  fgraphA = ../../image/dflow.attr;
```

```
image   dfgraph = ../../image/dflow.data;
image   token = ../../image/dflowq.data;
image   conditionImage,flowResult,expandedImg,nullImg,dfgraph0,firedToken,multi;
image   calcStart,calcStartArea,calcStartNum,subNumArea,subNum,addA,calcWork;
image   outArea,lowerOut,output,regA,regB,regB2,answer;
var     offImg,conditionImageNum,flowNum,maxFlowNum,rule,mrule,bitV,bitH,k;
var     node,mnode,i,j,ibitH,ibitV,numBit,numBitm1,numBitp1;

maxFlowNum = 3;       /* maximum flowerence number */
rule = 4;      /* rule number */
rule = rule - 1;
mrule = -rule;
node =  6;     /* node number */
node = node -1;
mnode = -node;
numBit = 5;    /* bit number of a number */
bitV = 5;      /* vertical bit number of a node */
bitH = numBit + 2;    /* holozontal bit number of a node */

/* logical or */
or = |PP|;

/* logical and */
and = |11|;

/* expand a pixel horizontally */
expandH = <|0.|@(0,mnode*bitH)>;
mnode = mnode + 1;
for i = mnode to node do
    ibitH = i * bitH;
    expandH = expandH * <|0.|@(0,ibitH)>;
end;

/* expand a pixel vertically */
expandV = <|0.|@(mrule*bitV,0)>;
mrule = mrule + 1;
for i = mrule to rule do
    ibitV = i * bitV;
    expandV = expandV * <|0.|@(ibitV,0)>;
end;

/* detect rules that answer queries */
detectRule = and;

/* detect nodes that are derived */
detectNode = |00_11|;

/* select data with attribute plane */
selectRule = |11|;

/* select node with attribute plane */
selectNode = |10_01|;

/* detect rules that answer queries (after selection) */
detectRule2 =  <|0.|@(0,-1)>;
for i = 1 to 4 do
    for j = 0 to numBit do
       detectRule2 = detectRule2 * <|0.|@(-i,-j)>;
    end;
end;

/* detect nodes that are derived (after selection)  */
detectNode2 =  <| 0.|
               | 0.|
               | 0.|
               |_0.|@(-1,1)>;
numBitm1 = numBit - 1;
for j = 0 to numBitm1 do
    for i = 1 to 4 do
       detectNode2 = detectNode2 * <|0.|@(-i,-j)>;
    end;
end;

/* make conditionImage pattern */
makeFlowStart = and + |01_10 0.|;
```

```
/* detect token */
detectToken = |1.|
              |.1|
              |.1|
              |.1|
              |.1| + |1.|
                    |.2|
                    |.1|
                    |.0|
                    |.0|;

/* detect answer */
numBitp1 = numBit + 1;
detectAnswer = |1. .. .0|;
for j = 3 to numBitp1 do
    detectAnswer = detectAnswer * |.0|@(0,j);
end;

/* detect multiplication operation */
detectMlt = |1.| * |.1 .1 |@(0,numBit);

/* make an attribute plane of plus operation */
makeAddAttr =  <| .0|
               |_.0|@(-3,0)>;
for j = 1 to numBitm1 do
    for i = 3 to 4 do
       makeAddAttr = makeAddAttr * <|.0|@(-i,-j)>;
    end;
end;

/* make an attribute plane of multiplication operation */
makeMltAttr = |1.|@(-3,-numBit);

/* make subtract number area */
makeSubNumArea1 = |1. .. .0 .0 .0 .1 .0|@(-3,-1);

makeSubNumArea2 = <| 0.|
                  |_0.|>;
for j = 1 to numBitm1 do
    for i = 0 to 1 do
       makeSubNumArea2 = makeSubNumArea2 * <|0.|@(-i,-j)>;
    end;
end;

/* clear subtraction number */
clrSubNum = |01|;

/* shift down */
shiftD = |.1| + |1.|@(-1,0);

/* make output area */
makeOutArea =  <| ..|
               | 0.|
               | ..|
               |_0.|@(-1,0)>;
for j = 1 to numBit do
    for i = 1 to 4 do
       makeOutArea = makeOutArea * <|0.|@(-i,-j)>;
    end;
end;


/* clear output area */
clrOutArea = clrSubNum;

/* detect output bit */
detectOut1 = |1.|@(-3,-numBit-1) * |.1|@(0,0);

detectOut2 = <| 0.|
             |_0.|>;
for j = 1 to numBit do
    detectOut2 = detectOut2 * <| 0.|
                              |_0.|@(0,j)>;
end;
```

```
/* set output number to dfgraph */
setOut = | .1|
         | ..|
         |_1.| + |_.1|
                 | ..|
                 | 0.|;

/* set token and '1' */
setToken = |1.|@(-1,0) * |.1|@(0,numBit+1)
         + |1.|@(-3,0) * |.1|@(0,numBit+1) + |1.|;
/*
       MSD addition (kakizaki version)
*/
add1 =  | 01|
        |_00|
        | 10|
        | 10| + | 00|
                |_00|
                | 11|
                | 10| + |_00|
                        | 01|
                        | 10|
                        | 10| + |_00|
                                | 00|
                                | 10|
                                | 11| +
        | .. 01|
        | .. 00|
        |_.. 1.|
        | .. 10| + | .. 0.|
                    | .. 00|
                    |_.. 11|
                    | .. 10| + | .. 00|
                                | .. 01|
                                | .. 10|
                                |_.. 1.| + | .. 00|
                                            | .. 0.|
                                            | .. 10|
                                            |_.. 11|;
add2 =  |_01|
        | 00|
        | 10|
        | 10| + |_00|
                | 00|
                | 11|
                | 10| + | 00|
                        |_01|
                        | 10|
                        | 10| + | 00|
                                |_00|
                                | 10|
                                | 11| + | .. 01|
                                          | .. 00|
                                          |_.. 11|
                                          | .. 10| + | .. 00|
                                                      | .. 01|
                                                      | .. 10|
                                                      |_.. 11|;
add3 =  |_01|
        | 00|
        | 10|
        | 10| + |_00|
                | 00|
                | 11|
                | 10| + | 00|
                        |_01|
                        | 10|
                        | 10| + | 00|
                                |_00|
                                | 10|
                                | 11|;

/* MSD 3 bit multiplication */

/* step 1 */
multi1 = <|NN|
          |NN|@(1,0)>;
for j = 1 to numBitm1 do
```

```
    multi1 = multi1 * <|NN|
                      |NN|@(1,j)>;
end;
multi1 = multi1 + <|.0|>;

/* step 1' */
multi1d1 = <| .0|
           |_..|
           | .0|>;
multi1d2 = <|.0|@(1,0)> * <|NN|@(2,0)>;
multi1d3 = <|.0|@(-1,0)> * <|NN|@(3,0)>;
multi1d4 = <|NN|
           |NN|@(2,0)>;
for j = 1 to numBitm1 do
    multi1d2 = multi1d2 * <|NN|@(2,j)>;
    multi1d3 = multi1d3 * <|NN|@(3,j)>;
    multi1d4 = multi1d4 * <|NN|
                          |NN|@(2,j)>;
end;
multi1d = multi1d1 + multi1d2 + multi1d3 + multi1d4;

/* shift down */
DshiftA = | 1.|
          |_..|;

/* shift up and left */
ULshiftA = |_.. ..|
           | .. 1.|;


/* clear image */
clrImg = |DD|;

flowNum = 1;

/* make null image */
nullImg = exec(token,token,clrImg);

/* remain graphB image */
dfgraph0 = dfgraph;

/* inference loop */
loop
    /* detect arcs that is connected to other nodes */
    expandedImg =!exec(token,token,expandV);
    conditionImage = exec(expandedImg,dfgraph,detectRule);
    conditionImage = exec(fgraphA,conditionImage,selectRule);
    conditionImage =!exec(conditionImage,conditionImage,detectRule2);
    token = exec(conditionImage,expandedImg,makeFlowStart);

    /* detect connected nodes */
    expandedImg =!exec(token,token,expandH);
    calcStart = exec(expandedImg,dfgraph,detectNode);
    calcStart = exec(fgraphA,calcStart,selectNode);
    calcStartArea =!exec(calcStart,calcStart,detectNode2);
    calcStartNum = exec(calcStartArea,expandedImg,and);
    dfgraph = exec(calcStartNum,dfgraph,or);

    /* detect where all token has reached */
    firedToken = exec(fgraphA,dfgraph,detectToken);

    /* detect answer */
    answer = exec(firedToken,dfgraph,detectAnswer);
    offImg = condition(Zero);
    if offImg = 0 then
       exit;
    end;

    /* calculation branch */
    multi = exec(firedToken,dfgraph,detectMlt);
    offImg = condition(Zero);

    if offImg = 1 then
       /* calculate addition or subtraction */

       /* make negative number for subtraction */
       addA =!exec(calcStart,calcStart,makeAddAttr);
       subNumArea = exec(fgraphA,dfgraph,makeSubNumArea1);
```

```
    subNumArea =!exec(subNumArea,subNumArea,makeSubNumArea2);
    subNum = exec(subNumArea,dfgraph,and);
    dfgraph = exec(subNumArea,dfgraph,clrSubNum);
    dfgraph = exec(subNum,dfgraph,shiftD);

    /* MSD addition */
    calcWork = exec(addA, dfgraph, add1);
    calcWork = exec(addA, calcWork, add2);
    calcWork = exec(addA, calcWork, add3);

else
    /* MSD multiplication */

    /* A * B(bit0) */
    regA = exec(multi,multi,makeMltAttr);
    regB =!exec(regA,dfgraph,multi1);
    regA = exec(regA,regB,DshiftA);
    regB2 =!exec(regA,dfgraph,multi1d);
    regB = exec(regB,regB2,or);

    addA = !exec(multi,multi,makeAddAttr);

    /* A * B(bitk) */
    for k = 1 to numBitm1 do

        /* step 2k */
        multi2k = <|NN|
                   |NN|@(-1,0)>;
        for j = 1 to numBitm1 do
            multi2k = multi2k * <|NN|
                                 |NN|@(-1,j)>;
        end;
        multi2k = multi2k + <|.0|@(-2,k)>;

        /* step 2k' */
        multi2kd1 = <|.0|@(-1,k) * |.0|@(-3,k)>;
        multi2kd2 = <|.0|@(-1,k) * |NN|>;
        multi2kd3 = <|.0|@(-3,k) * |NN|@(1,0)>;
        multi2kd4 = <|NN|
                     |NN|>;
        for j = 1 to numBitm1 do
           multi2kd2 = multi2kd2 * <|NN|@(0,j)>;
            multi2kd3 = multi2kd3 * <|NN|@(1,j)>;
            multi2kd4 = multi2kd4 * <|NN|
                                     |NN|@(0,j)>;
        end;
        multi2kd = multi2kd1 + multi2kd2 + multi2kd3 + multi2kd4;

        regA = exec(regA,regB,ULshiftA);
        regB2 =!exec(regA,dfgraph,multi2k);
        regB = exec(regB,regB2,or);
        regA = exec(regA,regB,DshiftA);
        regB2 =!exec(regA,dfgraph,multi2kd);
        regB = exec(regB,regB2,or);
        regB = exec(addA,regB,add1);
        regB = exec(addA,regB,add2);
        regB = exec(addA,regB,add3);

    end;

    calcWork = regB;

 end;

/* set output number to dfgraph */
outArea =!exec(firedToken,firedToken,makeOutArea);
dfgraph = exec(outArea,dfgraph,clrOutArea);
lowerOut = exec(fgraphA,dfgraph,detectOut1);
lowerOut =!exec(lowerOut,lowerOut,detectOut2);
output = exec(lowerOut,calcWork,setOut);
dfgraph = exec(output,dfgraph,or);

/* set output data to token */
token = exec(firedToken,dfgraph,setToken);
token = exec(output,token,or);

/* count flow number */
flowNum = flowNum + 1;
```

```
    if flowNum > maxFlowNum then
        exit;
    end;
end;

imout dflow.data.result dfgraph;

end dflow;
```

B.5　奇偶置換ソート

```
/*
    Odd-Even Transposition Sort (4bit)
        ver.2.0  1991-06-22  Masaya Iwata
        ver.2.1  1991-06-24  Masaya Iwata
        ver.3.0  1991-06-25  Masaya Iwata
*/

program oesort3;

kernel or,and,detect10,makeAttribute1,makeAttribute2,makeAttribute1-2;
kernel makeMinus,upIsMax,exchange;
kernel makeExchangeArea,makeChangeArea;
kernel exNonChangeData;
image  attr = ../../image/sort/oesort3.attr;
image  data = ../../image/sort/oesort3.data;
image  minusData,exchangeData,comparedResult,endData;
image  attribute1,attribute2,attribute1-2,attribute2-2,changeArea;
image  nonChangeData,expandedResult;
var    n,bit,bitm1,i,sortNum,changed,changed2;

n = 10;        /* number of data (= step number) */
bit = 4;       /* bit number of data */

sortNum = 1;

/* logical and */
and = |11|;

/* logical or */
or = |PP|;

/* detect 10 or 01 */
detect10 = |11|
           |00| + | 10|
                  |_01|;

/* detect where numbers of odd places are maximum
       (changed for the paper)
*/
upIsMax = |_11|
          | 00| + |_10 11|
                  | 00 00| + |_10 10 11|
                            | 00 00 00| + |_10 10 10 11|
                                        | 00 00 00 00|;

/* make exchange area */
bitm1 = bit - 1;
makeExchangeArea = <| NN|>;
for i = 1 to bitm1 do
    makeExchangeArea = makeExchangeArea * <| NN|@(0,-i)>;
end;

/* exchange numbers of odd and even places */
exchange = |11|@(-1,0) + |1.| * |.1|@(1,0);

/* make changed area */
makeChangeArea = |1.|@(-1,0) + |1.|;

/* extract non-changed data */
exNonChangeData = |01|;
```

```
/* make attribute1 */
makeAttribute1 = |1. 1.|
                 |0. 0.| + |1._1.|
                           |0. 0.| + |1. 1.|
                                     |1. 0.| + |1._1.|
                                               |1. 0.|;

/* make attribute2 */
makeAttribute2 = | 1. 1.|
                 |_0. 0.| + |1. 1.|
                            |0._0.|;

/* make attribute1-2, attribute2-2 */
makeAttribute1-2 = |0._1.|;

/* make attribute1 */
attribute1 = exec(attr,attr,makeAttribute1);

/* make attribute2 */
attribute2 = exec(attr,attr,makeAttribute2);

/* make attribute1-2 */
attribute1-2 = exec(attribute1,attribute1,makeAttribute1-2);

/* make attrbute2-2 */
attribute2-2 = exec(attribute2,attribute2,makeAttribute1-2);

/* main loop */
loop
    /* odd-even sort */
    minusData = exec(attribute1,data,detect10);
    comparedResult = exec(attribute1,minusData,upIsMax);
    comparedResult = exec(attribute1-2,comparedResult,and);
    changed = condition(Number);    /* detect number of changed data */
    expandedResult =!exec(attribute1,comparedResult,makeExchangeArea);
    exchangeData = exec(expandedResult,data,exchange);
    changeArea = exec(expandedResult,expandedResult,makeChangeArea);
    nonChangeData = exec(changeArea,data,exNonChangeData);
    data = exec(exchangeData,nonChangeData,or);
    sortNum = sortNum + 1;
    if sortNum > n then
       exit;
    end;

    /* even-odd sort */
    minusData = exec(attribute2,data,detect10);
    comparedResult = exec(attribute2,minusData,upIsMax);
    comparedResult = exec(attribute2-2,comparedResult,and);
    changed2 = condition(Number);   /* detect number of changed data */
    changed = changed * changed2;
    if changed = 0 then
       exit;
    end;
    expandedResult =!exec(attribute2,comparedResult,makeExchangeArea);
    exchangeData = exec(expandedResult,data,exchange);
    changeArea = exec(expandedResult,expandedResult,makeChangeArea);
    nonChangeData = exec(changeArea,data,exNonChangeData);
    data = exec(exchangeData,nonChangeData,or);
    sortNum = sortNum + 1;
    if sortNum > n then
       exit;
    end;

end;

imout oesort.data.f data;

end oesort3;
```

B.6　奇偶マージソート

```
/*
    Odd-Even Merge Sort (4bit)
       ver.2.0  1991-06-22  Masaya Iwata
       ver.3.0  1991-06-27  Masaya Iwata
*/
```

Appendix B　並列演算法の OALL プログラム

```
program oemsort3;

kernel or,and,outLeftData,outRightData;
kernel makeArea,shiftLeft,setRightData;
kernel expandData,detect10,shiftBanyan,shiftBanyanDel;
image  attrMinMax = ../../image/sort/oemsort3.attrMM;
image  data = ../../image/sort/oemsort3.data;
image  attrAorI,attrLorR,attrNonShift,attrShift,comparisonTemp;
image  comparedResultL,expandedResultL,comparedResultR,expandedResultR;
image  leftData,rightData,shiftData;
var    n,nm1,bit,bitm1,bitx2,bitx2m1,i,shiftNo,sortStep,step;

n = 1;         /* number of data group */
step = 6;      /* number of step ( =log N(log N+1)/2 ) */
               /*                    2     2    */
bit = 4;       /* bit number of data */

/* logical or */
or = |PP|;

/* logical and */
and = |11|;

/* detect 10 and 01 */
detect10 = |11| * |00|@(0,bit) + |10|@(0,-bit) * |01|;

/* expand data to left and right */
expandData = |0.|;
nm1 = n - 1;
bitx2 = bit * 2;
for i = 0 to nm1 do
    expandData = expandData * <|0.|@(0,-bitx2*i)>;
end;

/* delete non-shifted data in banyan */
shiftBanyanDel = |01|;

/* select upper data */
outLeftData = |11 1.| * |00|@(0,bit)
            + |10 11| * |00 00|@(0,bit)
            + |10 10 01| * |00 00 00|@(0,bit)
            + |10 10 00 01| * |00 00 00 00|@(0,bit)
            + |10 0. 1.| * |01|@(0,bit)
            + |10 00 1.| * |00 01|@(0,bit)
            + |10 00 10| * |00 00 01|@(0,bit)
            + |10 00 10 00| * |00 00 00 01|@(0,bit);

/* select righter data */
outRightData = |10 1.| * |01|@(0,bit)
             + |10 10| * |00 01|@(0,bit)
             + |10 10 00| * |00 00 01|@(0,bit)
             + |10 10 00 00| * |00 00 00 01|@(0,bit)
             + |11 0. 1.| * |00|@(0,bit)
             + |10 01 1.| * |00 00|@(0,bit)
             + |10 00 11| * |00 00 00|@(0,bit)
             + |10 00 10 01| * |00 00 00 00|@(0,bit);

/* make area of output data (1x4 pixel) */
bitm1 = bit - 1;
makeArea = <| 0.|>;
for i = 1 to bitm1 do
    makeArea = makeArea * <| 0.|@(0,-i)>;
end;

/* shift to left area */
shiftLeft = |1.| * |1|@(0,bit);

/* set Right Data */
setRightData = |11| + |01|@(0,-bit);

shiftNo = 1;
sortStep = 1;

/* read attribute planes of the first step */
attrAorI = ../../image/sort/oemsort3.attrAI-1;
attrLorR = ../../image/sort/oemsort3.attrLR-1;
attrNonShift = ../../image/sort/oemsort3.attrNS-1;
```

```
attrShift = ../../image/sort/oemsort3.attrS-1;

/* main loop */
loop
    /* set data to either left or right */
    data = exec(attrLorR,data,setRightData);
    /* shift data */
    shiftBanyan = |11|@(-shiftNo,0) + |01|@(shiftNo,0);
    shiftData = exec(attrNonShift,data,shiftBanyanDel);
    shiftData = exec(attrShift,shiftData,shiftBanyan);
    data = exec(data,shiftData,or);

    /* detect 10 or 01 */
    comparisonTemp = exec(attrMinMax,data,detect10);

    /* set output data on the left area according to a or i */
    comparedResultL = exec(attrAorI,comparisonTemp,outLeftData);
    comparedResultR = exec(attrAorI,comparisonTemp,outRightData);
    expandedResultL =!exec(comparedResultL,comparedResultL,makeArea);
    expandedResultR =!exec(comparedResultR,comparedResultR,makeArea);
    leftData = exec(expandedResultL,data,and);
    rightData = exec(expandedResultR,data,shiftLeft);
    data = exec(leftData,rightData,or);

    /* count sort step and set shiftNo */
    sortStep = sortStep + 1;
    if sortStep > step then
       exit;
    end;
    if sortStep = 2 then
       shiftNo = 2;
       attrAorI = ../../image/sort/oemsort3.attrAI-2;
       attrLorR = ../../image/sort/oemsort3.attrLR-2;
       attrNonShift = ../../image/sort/oemsort3.attrNS-2;
       attrShift = ../../image/sort/oemsort3.attrS-2;
    end;
    if sortStep = 3 then
       shiftNo = 1;
       attrAorI = ../../image/sort/oemsort3.attrAI-3;
       attrLorR = ../../image/sort/oemsort3.attrLR-3;
       attrNonShift = ../../image/sort/oemsort3.attrNS-3;
       attrShift = ../../image/sort/oemsort3.attrS-3;
    end;
    if sortStep = 4 then
       shiftNo = 4;
       attrAorI = ../../image/sort/oemsort3.attrAI-4;
       attrLorR = ../../image/sort/oemsort3.attrLR-4;
       attrNonShift = ../../image/sort/oemsort3.attrNS-4;
       attrShift = ../../image/sort/oemsort3.attrS-4;
    end;
    if sortStep = 5 then
       shiftNo = 2;
       attrAorI = ../../image/sort/oemsort3.attrAI-5;
       attrLorR = ../../image/sort/oemsort3.attrLR-5;
       attrNonShift = ../../image/sort/oemsort3.attrNS-5;
       attrShift = ../../image/sort/oemsort3.attrS-5;
    end;
    if sortStep = 6 then
       shiftNo = 1;
       attrAorI = ../../image/sort/oemsort3.attrAI-6;
       attrLorR = ../../image/sort/oemsort3.attrLR-6;
       attrNonShift = ../../image/sort/oemsort3.attrNS-6;
       attrShift = ../../image/sort/oemsort3.attrS-6;
    end;

end;

imout oemsort.data.f data;

end oemsort3;
```

B.7　データベース処理

```
/*
    Relational Data Base Operation
      with Odd-Even Transposition Sort ver.3.0 (4bit)
       ver.1.0  1991-08-08  Masaya Iwata
       ver.2.0  1991-08-21  Masaya Iwata
       ver.2.1  1991-10-28  Masaya Iwata
       ver.3.0  1991-11-15  Masaya Iwata
       ver.3.1  1991-11-16  Masaya Iwata
       ver.3.2  1992-02-06  Masaya Iwata

       operations : selection, restriction, projection, semijoin, sort.
*/

program rdbase3;

kernel or,and,delData,detect10,makeAttrOdd,makeAttrEven,makeMinus;
kernel upIsMax,exchange,makeExchangeArea,makeChangeArea,exNonChangeData,expandH;
kernel detect10c,detectFIsMax,shiftD,shiftU,expandV,expandV2,expandHE,expandHA;
kernel makeCondArea,makeDataOp,match,makeAttrD,expandHNE,expandV2m1;
kernel detectFIsMin,MSBofBit1,delOverlap,makeFullExcArea,makeAttr,makeAttr2;
kernel shiftOpAttrNumR,shiftOpAttrNumSJ;
image  database = ../../image/rdbase/rdbase-paper50.data;
image  dbaseAttr0 = ../../image/rdbase/rdbase-paper50.attr;
image  minusData,exchangeData,exchangeArea,endData,attrOdd,attrEven,dataArea,dbase2;
image  changeArea,nonChangeData,opImg,delArea,attrD,expandedCondition,attrSort;
image  expandedData,finalResult,comparisonResult,comparedData,conditionImage;
image  selData,attrSelD,addData,dataAreaAttr,dataOp,conditionArea,matchingResult;
image  dbaseAttr,dbaseAttr2;
var    tupleNum,bit,bitm1,mbitm1,bitH,bitHm1,mbitHm1,bitHmbit,attrNumm1,mattrNumm1;
var    i,sortNum,changed,changed2,attrNum,mtupleNumm1,opNum,eqNum,mtupleNump1;
var    dataEnd,tupleNumm1,compareNum,noData,noMatch,opAttrNum,opAttrNum2,shiftAttrNum;

tupleNum = 50; /* number of data (= step number of sorting) */
bit = 25;      /* bit number of data (available bit number is (bit-1)) */
attrNum = 2;   /* attribute number (= row number of relation) */
bitH = bit * attrNum; /* horizontal bit number */

opNum = 1;     /* operation number */
               /* 1:selection, 2:restriction, 3:projection, 4:semijoin, 5:sort */
opAttrNum = 1; /* begins from left at 0 */
opAttrNum2 = 1;       /* used to direct the 2nd attribute */
eqNum = 1;     /* inequality number .. =:1, <:2, >:3 */

/* make data used in the operation from the attribute plane */
makeDataOp = <|.. .. 0. 0. .. 0. .. .. .. .. .. .. .. .. 0. 0. .. .. .. .. .. 0. .. 0.
.|@(0,-bit+1)>;
        /* make '01010 00001 10000 00001 01100' */

/* make dbaseAttr */
makeAttr = |1.|@(0,-bit*opAttrNum);

/* make dbaseAttr2 */
makeAttr2 = |1.|@(0,-bit*opAttrNum2);

/* shift data according to opAttrNum and opAttrNum2 (for semijoin) */
shiftAttrNum = opAttrNum2 - opAttrNum;
shiftOpAttrNumSJ = |1.|@(0,shiftAttrNum*bit);

/* shift data according to opAttrNum and opAttrNum2 (for restriction) */
shiftAttrNum = opAttrNum - opAttrNum2;
shiftOpAttrNumR = |1.|@(0,shiftAttrNum*bit);

/* logical or */
or = |PP|;

/* logical and */
and = |11|;

/* delete data */
delData = |01|;

/* detect 10 or 01 */
detect10 = |11|
           |00| + | 10|
                  |_01|;
```

```
/* detect where numbers of odd places are maximum (product term number: bit) */
upIsMax = |0._11|
          |0. 00| + |0._10 11|
                    |0. 00 00| + |0._10 10 11|
                              |0. 00 00 00| + |0._10 10 10 11|
                                        |0. 00 00 00 00| + |0._10 10 10 10 11|
                                                  |0. 00 00 00 00 00|;

/* detect 10 or 01 in corresponding pixels */
detect10c = |UU|;

/* detect where numbers on 1st images are minimum */
detectFIsMin = |01| + |.0 01| + |.0 .0 01| + |.0 .0 .0 01| + |.0 .0 .0 .0 01|;

/* detect where numbers on 1st images are maximum */
detectFIsMax = |11| + |.0 11| + |.0 .0 11| + |.0 .0 .0 11| + |.0 .0 .0 .0 11|;

/* detect MSB in the bits of value 1 */
MSBofBit1 = |11|;

/* make exchange area */
bitm1 = bit - 1;
makeExchangeArea = <|NN|>;
for i = 1 to bitm1 do
    makeExchangeArea = makeExchangeArea * <|NN|@(0,-i)>;
end;

/* exchange numbers of odd and even places */
exchange = |11|@(-1,0) + |1.| * |.1|@(1,0);

/* make changed area */
makeChangeArea = |1.|@(-1,0) + |1.|;

/* make attrOdd */
makeAttrOdd = |1. 1.|
              |0. 0.| + |1._1.|
                        |0. 0.| + |1. 1.|
                                  |1. 0.| + |1._1.|
                                            |1. 0.|;

/* make attrEven */
makeAttrEven = | 1. 1.|
               |_0. 0.| + |1. 1.|
                          |0._0.|;

/* shift 1 pixel down */
shiftD = |1.|@(-1,0);

/* shift 1 pixel up */
shiftU = |1.|@(1,0);

/* expand data vertically (to 2 pixel up and down) */
mtupleNumm1 = -tupleNum - 1;
expandV = <|0.|>;
for i = mtupleNumm1 to -1 do
    expandV = expandV * <|0.|@(i,0)>;
end;

/* expand data vertically (to up and down) */
tupleNumm1 = tupleNum - 1;
expandV2 = expandV;
for i = 1 to tupleNumm1 do
    expandV2 = expandV2 * <|0.|@(i,0)>;
end;

/* expand data vertically (to up and down except origin) */
expandV2m1 = <|0.|@(1,0)>;
for i = 2 to tupleNumm1 do
    expandV2m1 = expandV2m1 * <|0.|@(i,0)>;
end;
for i = mtupleNumm1 to -1 do
    expandV2m1 = expandV2m1 * <|0.|@(i,0)>;
end;

/* make conditionArea */
makeCondArea = <|0.|>;
mbitm1 = -bit + 1;
for i = mbitm1 to -1 do
```

```
        makeCondArea = makeCondArea * <|0.|@(0,i)>;
end;

/* matching of 2 data */
match = |EE|;
for i = 1 to bitm1 do
    match = match * |EE|@(0,i);
end;

/* expand horizontally (in 1 attribute) */
expandH = <|0.|>;
for i = mbitm1 to -1 do
    expandH = expandH * <|0.|@(0,i)>;
end;

/* expand horizontally (to all attribute) */
attrNumm1 = attrNum - 1;
mattrNumm1 = -attrNumm1;
expandHA = <|0.|>;
for i = mattrNumm1 to attrNumm1 do
    expandHA = expandHA * <|0.|@(0,i*bit)>;
end;

/* expand horizontally (in '=' operation) */
bitHmbit = bitH - bit;
mbitHm1 = -bitH + 1;
expandHE = <|0.|>;
for i = 1 to bitHmbit do
    expandHE = expandHE * <|0.|@(0,i)>;
end;
for i = mbitHm1 to -1 do
    expandHE = expandHE * <|0.|@(0,i)>;
end;

/* expand horizontally (in '<' or '>' operation) */
bitHm1 = bitH - 1;
expandHNE = <|0.|>;
for i = 1 to bitHm1 do
    expandHNE = expandHNE * <|0.|@(0,i)>;
end;
for i = mbitHm1 to -1 do
    expandHNE = expandHNE * <|0.|@(0,i)>;
end;

/* make attrD */
mtupleNump1 = -tupleNum - 1;
makeAttrD = <|0.|>;
for i = mtupleNump1 to -1 do
    makeAttrD = makeAttrD * <|0.|@(i,0)>;
end;

/* delete overlapping data in projection operation */
delOverlap = |EE|;
for i = 1 to bitm1 do
    delOverlap = delOverlap * |EE|@(0,i);
end;

/* make full-exchange area in sorting operation */
makeFullExcArea = expandHE;

/* make dbaseAttr */
dbaseAttr = exec(dbaseAttr0,dbaseAttr0,makeAttr);

/* make attrD */
attrD =!exec(dbaseAttr,dbaseAttr,makeAttrD);

/* selection operation */
if opNum = 1 then
    expandedCondition =!exec(attrD,attrD,makeCondArea);
    dataOp =!exec(dbaseAttr,dbaseAttr,makeDataOp);
    expandedData =!exec(dataOp,dataOp,expandV);
    /* '=' operation */
    if eqNum = 1 then
        comparisonResult = exec(expandedData,database,match);
        comparisonResult = exec(attrD,comparisonResult,and);
        comparisonResult =!exec(comparisonResult,comparisonResult,expandHE);
        finalResult = exec(comparisonResult,database,and);
    end;
```

```
    /* '<' operation */
    if eqNum = 2 then
       comparedData = exec(expandedCondition,database,and);
       comparisonResult = exec(comparedData,expandedData,detect10c);
       comparisonResult = exec(comparedData,comparisonResult,detectFIsMin);
       comparisonResult = exec(attrD,comparisonResult,MSBofBit1);
       comparisonResult =!exec(comparisonResult,comparisonResult,expandHNE);
       finalResult = exec(comparisonResult,database,and);
    end;
    /* '>' operation */
    if eqNum = 3 then
       comparedData = exec(expandedCondition,database,and);
       comparisonResult = exec(comparedData,expandedData,detect10c);
       comparisonResult = exec(comparedData,comparisonResult,detectFIsMax);
       comparisonResult = exec(attrD,comparisonResult,MSBofBit1);
       comparisonResult =!exec(comparisonResult,comparisonResult,expandHNE);
       finalResult = exec(comparisonResult,database,and);
    end;
end;

/* restriction operation */
if opNum = 2 then
    dbaseAttr2 = exec(dbaseAttr0,dbaseAttr0,makeAttr2);
    dataArea =!exec(dbaseAttr,dbaseAttr,expandV);
    dataArea =!exec(dataArea,dataArea,expandH);
    comparedData = exec(dataArea,database,and);
    comparedData = exec(comparedData,comparedData,shiftOpAttrNumR);
    dataAreaAttr =!exec(dbaseAttr2,dbaseAttr2,expandV);
    conditionImage = database;
   /* '=' operation */
    if eqNum = 1 then
       comparisonResult = exec(comparedData,conditionImage,match);
       comparisonResult = exec(dataAreaAttr,comparisonResult,and);
       comparisonResult =!exec(comparisonResult,comparisonResult,expandHE);
       finalResult = exec(comparisonResult,database,and);
    end;
    /* '<' operation */
    if eqNum = 2 then
       comparisonResult = exec(comparedData,conditionImage,detect10c);
       comparisonResult = exec(comparedData,comparisonResult,detectFIsMin);
       comparisonResult = exec(dataAreaAttr,comparisonResult,MSBofBit1);
       comparisonResult =!exec(comparisonResult,comparisonResult,expandHNE);
       finalResult = exec(comparisonResult,database,and);
    end;
    /* '>' operation */
    if eqNum = 3 then
       comparisonResult = exec(comparedData,conditionImage,detect10c);
       comparisonResult = exec(comparedData,comparisonResult,detectFIsMax);
       comparisonResult = exec(dataAreaAttr,comparisonResult,MSBofBit1);
       comparisonResult =!exec(comparisonResult,comparisonResult,expandHNE);
       finalResult = exec(comparisonResult,database,and);
    end;
end;

/* projection operation */
if opNum = 3 then
    compareNum = 1;
    /* make attribute planes */
    expandedCondition =!exec(attrD,attrD,makeCondArea);
    conditionArea =!exec(dbaseAttr,dbaseAttr,makeCondArea);
    /* cut unused data */
    comparedData = exec(expandedCondition,database,and);
    loop
       conditionImage = exec(conditionArea,comparedData,and);
       noData = condition(Zero);
       if noData = 0 then
          expandedCondition =!exec(conditionImage,conditionImage,expandV2m1);
          comparisonResult = exec(comparedData,expandedCondition,match);
          comparisonResult = exec(comparisonResult,attrD,and);
          noMatch = condition(Zero);
          if noMatch = 0 then
             matchingResult =!exec(comparisonResult,comparisonResult,makeCondArea);
             comparedData = exec(matchingResult,comparedData,delData);
          end;
       end;
       compareNum = compareNum + 1;
       if compareNum > tupleNum then
          exit;
```

```
        end;
        conditionArea = exec(conditionArea,conditionArea,shiftD);
    end;
    finalResult = comparedData;
end;

/* semijoin operation */
if opNum = 4 then
    /* read another database image */
    dbase2 = ../../image/rdbase/rdbase2-sj.data;
    dbaseAttr2 = exec(dbaseAttr0,dbaseAttr0,makeAttr2);
    dataAreaAttr =!exec(dbaseAttr,dbaseAttr,expandV);
    dataArea =!exec(dataAreaAttr,dataAreaAttr,expandH);
    comparedData = exec(dataArea,database,and);
    attrSelD =!exec(dbaseAttr2,dbaseAttr2,expandH);
    /* join operation */
    /* finalResult = database; */
    loop
        selData = exec(dbase2,attrSelD,and);
        selData = exec(selData,selData,shiftOpAttrNumSJ);
        conditionImage =!exec(selData,selData,expandV2);
        dataEnd = condition(Zero);
        if dataEnd = 1 then
            exit;
        end;
        /* '=' operation */
        if eqNum = 1 then
            comparisonResult = exec(comparedData,conditionImage,match);
            comparisonResult = exec(dataAreaAttr,comparisonResult,and);
            noMatch = condition(Zero);
            if noMatch = 0 then
                comparisonResult =!exec(comparisonResult,comparisonResult,expandHE);
                /* semijoin operation */
                addData = exec(comparisonResult,database,and);
                finalResult = exec(addData,finalResult,or);
            end;
        end;
        /* '<' operation */
        if eqNum = 2 then
            comparisonResult = exec(comparedData,conditionImage,detect10c);
            comparisonResult = exec(comparedData,comparisonResult,detectFIsMin);
            comparisonResult = exec(dataAreaAttr,comparisonResult,MSBofBit1);
            noMatch = condition(Zero);
            if noMatch = 0 then
                comparisonResult =!exec(comparisonResult,comparisonResult,expandHNE);
                /* semijoin operation */
                addData = exec(comparisonResult,database,and);
                finalResult = exec(addData,finalResult,or);
            end;
        end;
        /* '>' operation */
        if eqNum = 3 then
            comparisonResult = exec(comparedData,conditionImage,detect10c);
            comparisonResult = exec(comparedData,comparisonResult,detectFIsMax);
            comparisonResult = exec(dataAreaAttr,comparisonResult,MSBofBit1);
            noMatch = condition(Zero);
            if noMatch = 0 then
                comparisonResult =!exec(comparisonResult,comparisonResult,expandHNE);
                /* semijoin operation */
                addData = exec(comparisonResult,database,and);
                finalResult = exec(addData,finalResult,or);
            end;
        end;
        attrSelD = exec(attrSelD,attrSelD,shiftD);
    end;
end;

/* sorting operation */
if opNum = 5 then
    sortNum = 1;
    attrSort = ../../image/rdbase/rdbase-sort.attr;
    /* make attribute planes */
    expandedCondition =!exec(attrD,attrD,makeCondArea);
    attrSort = exec(expandedCondition,attrSort,and);
    attrOdd = exec(attrSort,attrSort,makeAttrOdd);
    attrEven = exec(attrSort,attrSort,makeAttrEven);
    /* cut unused data */
    comparedData = database;
```

```
    /* sorting operation */
    loop
       /* odd-even sort */
       minusData = exec(attrOdd,comparedData,detect10);
       exchangeArea = exec(attrOdd,minusData,upIsMax);
       changed = condition(Number); /* detect number of changed data */
       exchangeArea =!exec(exchangeArea,exchangeArea,makeFullExcArea);
       exchangeData = exec(exchangeArea,comparedData,exchange);
       changeArea = exec(exchangeArea,exchangeArea,makeChangeArea);
       nonChangeData = exec(changeArea,comparedData,delData);
       comparedData = exec(exchangeData,nonChangeData,or);
       sortNum = sortNum + 1;
       if sortNum > tupleNum then
           exit;
       end;
       /* even-odd sort */
       minusData = exec(attrEven,comparedData,detect10);
       exchangeArea = exec(attrEven,minusData,upIsMax);
       changed2 = condition(Number);  /* detect number of changed data */
       changed = changed * changed2;
       if changed = 0 then
           exit;
       end;
       exchangeArea =!exec(exchangeArea,exchangeArea,makeFullExcArea);
       exchangeData = exec(exchangeArea,comparedData,exchange);
       changeArea = exec(exchangeArea,exchangeArea,makeChangeArea);
       nonChangeData = exec(changeArea,comparedData,delData);
       comparedData = exec(exchangeData,nonChangeData,or);
       sortNum = sortNum + 1;
       if sortNum > tupleNum then
           exit;
       end;
    end;
    finalResult = comparedData;
end;

imout rdbase.data.f finalResult;

end rdbase3;
```

参考文献

[1] D. G. Feitelson, *Optical Computing. A Survey for Computer Scientists* (MIT Press, Cambridge, MA, 1988).

[2] R. Arrathoon, ed., *Optical Computing: Digital and Symbolic* (Marcel Dekker, Inc., New York, 1989).

[3] A. D. McAulay, *Optical Computing Architectures: the Application of Optical Concepts to Next Generation Computers* (John Wiley & Sons, New York, 1991).

[4] 辻内順平, 一岡芳樹, 峯本　工, *光情報処理* (オーム社, 東京, 1989).

[5] 谷田　純, "光コンピューター開発研究の現状", 光学 **20**, 632-641 (1991).

[6] B. S. Wherrett and F. A. P. Tooley, ed., *Optical Computing* (Scottish Universities Summer School in Physics, Edingurgh, Scotland, 1989).

[7] 日本物理学会編, *スーパーコンピュータ* (培風館, 東京, 1985).

[8] E. コーコラン, "スーパーコンピューター", 日経サイエンス **21**, 80-93 (1991).

[9] "実用化の第1歩を踏み出した大規模並列処理", 日経コンピュータ 1991. 9. 9, No. 262, 60-83 (1991).

[10] 雨宮真人, 田中　譲, *コンピュータアーキテクチャ* (オーム社, 東京, 1988).

[11] 梅尾博司, *超並列計算機アーキテクチャとそのアルゴリズム* (共立出版, 東京, 1991).

[12] 飯塚　肇, 田中英彦, *ソフトウェア指向アーキテクチャ* (オーム社, 東京, 1985).

[13] 馬場敬信, "超並列マシンへの道", 情報処理 **32**, 348-364 (1991).

[14] 小池誠彦, "超並列マシン", 情報処理 **28**, 94-105 (1987).

[15] 小柳　滋, 田辺　昇, "超並列マシンの実現技術", 情報処理 **32**, 365-376 (1991).

[16] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing* (McGraw-Hill, New York, 1984).

[17] 安浦寛人, "並列計算機と並列計算モデル", 情報処理 **33**, 1024-1032 (1992).

[18] J. Tanida and Y. Ichioka, "A Paradigm for Digital Optical Computing Based on Coded Pattern Processing," *Opt. Comput.* **1**, 113-128 (1990).

[19] K. Brenner, A. Huang, and N. Streibl, "Digital Optical Computing with Symbolic Substitution," *Appl. Opt.* **25**, 3054-3060 (1986).

[20] K. Huang, B. K. Jenkins, and A.A. Sawchuk, "Image Algebra Representation of Parallel Optical Bimary Arithmetic," *Appl. Opt.* **28**, 1263-1278 (1989).

[21] M. Fukui and K. Kitayama, "Image Logic Algebra and its Optical Implementation," *Appl. Opt.* **31**, 581-591 (1992).

[22] R. P. Bocker, B. L. Drake, M. E. Lasher, and T. B. Henderson, "Modefied Signed-Digit Addition and Subtraction Using Optical Symbolic Substitution," *Appl. Opt.* **25**, 2456-2457 (1986).

[23] I. Cramb and C. Upstill, "An Optoelectronic Binary Image Algebra Architecture," *Opt. Comput. Process.* **1**, 115-126 (1991).

[24] M. Fukui and K. Kitayama, "Application of Image-logic Algebra: Wire Routing and Numerical Data Processing," *Appl. Opt.* **31**, 4645-4656 (1992).

[25] M. Fukui, J. Tanida, and Y. Ichioka, "Flexible-Structured Computation Based on Optical Array Logic," *Appl. Opt.* **29**, 1604-1609 (1990).

[26] K. Huang and A. Louri, "Optical Multiplication and Division Using Modified-Signed-Digit Symbolic Substitution," *Appl. Opt.* **28**, 364-372 (1989).

[27] S. Kakizaki, J. Tanida, and Y. Ichioka, "Gray-Image Processing Using Optical Array Logic," *Appl. Opt.* **31**, 1093-1102 (1992).

[28] J. Tanida, M. Fukui, and Y. Ichioka, "Programming of Optical Array Logic. 2: Numerical Data Processing Based on Pattern Logic," *Appl. Opt.* **27**, 2931-2939 (1988).

[29] J. Tanida and Y. Ichioka, "Programming of Optical Array Logic. 1: Image Data Processing," *Appl. Opt.* **27**, 2926-2930 (1988).

[30] M. Iwata, J. Tanida, and Y. Ichioka, "Inference Engine Using Optical Array Logic," *Jpn. J. Appl. Phys.* **29**, L1259-L1261 (1990).

[31] J. Tanida, J. Nakagawa, E. Yagyu, M. Fukui, and Y. Ichioka, "Experimental Verification of Parallel Processing on a Hybrid Optical Parallel Array Logic System," *Appl. Opt.* **29**, 2510-2521 (1990).

[32] M. Iwata, J. Tanida, and Y. Ichioka, "Inference Engine for Expert System by Using Optical Array Logic," *Appl. Opt.* **31**, (1992).

[33] J. Tanida and Y. Ichioka, "Optical Logic Array Processor Using Shadowgrams," *J. Opt. Soc. Am.* **73**, 800-809 (1983).

[34] J. Tanida and Y. Ichioka, "Discrete Correlators Using Multiple Imaging for Digital Optical Computing," *Opt. Lett.* **16**, 599-601 (1991).

[35] D. Miyazaki, J. Tanida, and Y. Ichioka, "Construction of Modularized OPALS Using Optoelectronic devices," *Jpn. J. Appl. Phys.* **29**, L1550-L1552 (1990).

[36] J. Tanida and Y. Ichioka, "OPALS: Optical Parallel Array Logic System," *Appl. Opt.* **25**, 1565-1570 (1986).

[37] D. Miyazaki, J. Tanida, and Y. Ichioka, "Reflective Correlator for Optoelectronic Integration of Hybrid Optical Parallel Array Logic System," *Optik* **89**, 101-106 (1992).

[38] 宮崎大介, 柿崎　順, 谷田　純, 一岡芳樹, "光・電子複合型並列光アレーロジックシステム：H-OPALS 構成用集積型相関器", 信学論 **J75-C-I**, 296-304 (1992).

[39] 豊田順一, *人工知能* (昭晃堂, 東京, 1988).

[40] N. J. Nilsson, *Principles of Artificial Intelligence* (Tioga Publiching Co., Palo Alto, 1980).

[41] D. Nebendahl, ed., *Expert Systems.* (John Wiley & Sons Ltd., Chichester, 1988).

[42] 北野宏明, "超並列人工知能", 人工知能学会誌 **7**, 244-261 (1992).

[43] S. E. Fahlman, NETL: *A System for Representing and Using Real-World Knowledge* (MIT Press, Cambridge, MA, 1979).

[44] G. Eichmann and H. J. Caulfield, "Optical Learning (Inference) Machines," *Appl. Opt.* **24**, 2051-2054 (1985).

[45] C. Warde and J. Kottas, "Hybrid Optical Inference Machines: Architectural Considerations," *Appl. Opt.* **25**, 940-947 (1986).

[46] F. Kiamilev and S. Esener, *Implementation of NETL Knowledge-Base System with Programmable Opto-Electronic Multiprocessor Architecture* (Optical Society of America, 1989).

[47] 末田　正, *光エレクトロニクス* (昭晃堂, 東京, 1985).

[48] 松本正一, 角田市良, *液晶の基礎と応用* (工業調査会, 東京, 1991).

[49] D. Moldovan, W. Lee, and C. Lin, "SNAP: A Marker-Propagation Architecture for Knowledge Processing`," *IEEE Trans. Parallel Distributed Sys.* **3**, 397-410 (1992).

[50] 島田俊夫, "データフローマシン", 情報処理 **28**, 85-93 (1987).

[51] J.B. Dennis, "First Version of a Data Flow Procedure Language," in *Lecture Notes in Computer Science* 362-376 (Springer Verlag, 1974).

[52] 児玉祐悦, 坂井修一, 山口喜教, "高並列計算機 EM-4 とその並列性評価", 信学論 **J75-D-I**, 607-614 (1992).

[53] D. Miyazaki, J. Tanida, and Y. Ichioka, "Optical Implementation of the Banyan Network Using a Sagnac Inverter with a Patterned Mirror," *Opt. Commun.* **93**, 283-288 (1992).

[54] 喜連川優, 伏見信也, "データベースマシン", 情報処理 **28**, 56-67 (1987).

[55] 田中　譲, "データベース・マシンの現状と未来展望", bit **14**, 556-568 (1982).

[56] P. B. Berra, K. Brenner, W. T. Cathy, H. J. Caulfield, S. H. Lee, and H. Szu, "Optical Database/Knowledgebase machines," *Appl. Opt.* **29**, 195-205 (1990).

[57] P. B. Berra, A. Ghafoor, P. A. Mitkas, S. J. Marcinkowski, and M. Guizani, "The Impact of Optics on Data and Knowledge Base System," *IEEE Trans. Knowledge and Data Eng.* **1**, 111-132 (1989).

[58] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Comm. ACM* **13**, 377-387 (1970).

[59] S. G. Akl, *Parallel Sorting Algorithms* (Academic Press, London, 1985).

[60] G. Baudet and D. Stevenson, "Optimal Sorting Algorithms for Parallel Computers," *IEEE Trans. Comput.* **C-27**, 84-87 (1978).

[61] K. E. Batcher, "Sorting Networks and Their Applications," *AFIPS Spring Joint Computing Conference* **32**, 307-314 (1968).

[62] 喜連川優, 楊　維康, 鈴木慎司, "VLSIソートプロセッサ", 情報処理 **31**, 457-465 (1990).

[63] 喜連川優, 中野美由紀, "機能ディスクシステム：関係データベース処理とその性能評価", 電子情報通信学会論文誌 D-I **J74-D-I**, 496-507 (1991).

[64] Y. Owechko, G. J. Dunning, E. Marom, and B. H. Soffer, "Holographic Associative Memory with Nonlinearities in the Correlation Domain," *Appl. Opt.* **26**, 1900-1910 (1987).

[65] 秋山浩二, "光コンピュータを構成する機能デバイス," 光学 **20**, 650-656 (1991).

[66] J.W. Goodman, *Introduction to Fourier Optics* (McGraw-Hill, New York, 1968).

[67] R. P. Freese, "Optical Disks Become Erasable," *IEEE Spectrum* **25**, 41-45 (1988).

[68] M. Fukui and K. Kitayama, "Design Consideration of the Optical Image Crossbar Switch," *Appl. Opt.* **31**, 5542-5547 (1992).

[69] A. W. Lohmann, "What Classical Optics Can Do for the Digital Optical Computers," *Appl. Opt.* **25**, 1543-1549 (1986).

[70] A. W. Lohmann, "Optical Bus Network," *Optik* **74**, 30-35 (1986).

[71] 福井将樹, 岩田昌也, 並河和秀, 谷田　純, 北山研一, 一岡芳樹, "並列光演算記述言語の検討", 信学技報 **92**, 37-42 (1992).

[72] 並河和秀, "並列光演算言語: POL による並列演算技術の比較", 大阪大学工学部応用物理学科特別研究報告 (1992).

著者発表論文

1. M. Iwata, J. Tanida, and Y. Ichioka, "Inference Engine Using Optical Array Logic," *Jpn. J. Appl. Phys.* **29**, 7, L1259 - L1261 (1990).

2. M. Iwata, J. Tanida, and Y. Ichioka, "Inference Engine Using Optical Array Logic," *Conference Record of 1990 International Topical Meeting on optical Computing; Proc. Soc. Photo-Opt. Instrum. Eng.* **1359**, 57-58 (1990).

3. 岩田昌也, 谷田 純, 一岡芳樹, "光アレイロジックを用いたソーティングの検討", 第22回画像工学コンファレンス論文集, pp. 215-218 (1991).

4. 岩田昌也, 谷田 純, 一岡芳樹, "光アレイロジックによるデータベース処理", 並列処理シンポジウム JSPP '92 論文集, pp. 383-390 (1992).

5. M. Iwata, J. Tanida, and Y. Ichioka, "Inference Engine for Expert System by Using Optical Array Logic," *Appl. Opt.* **31**, 5604-5613 (1992).

6. M. Iwata, J. Tanida, and Y. Ichioka, "Database Management Using Optical Array Logic," *Appl. Opt.* (in press).

7. 岩田昌也, 谷田 純, 一岡芳樹, "並列ディジタル光演算システム用2次元仮想記憶機構", 光学 (submitted).