Join Processorの提案

佐藤和洋(*)　　金子裕(**)　　中村史朗(*)
（(*) 日立システム開発研究所　(**) 日立デバイス開発センター）

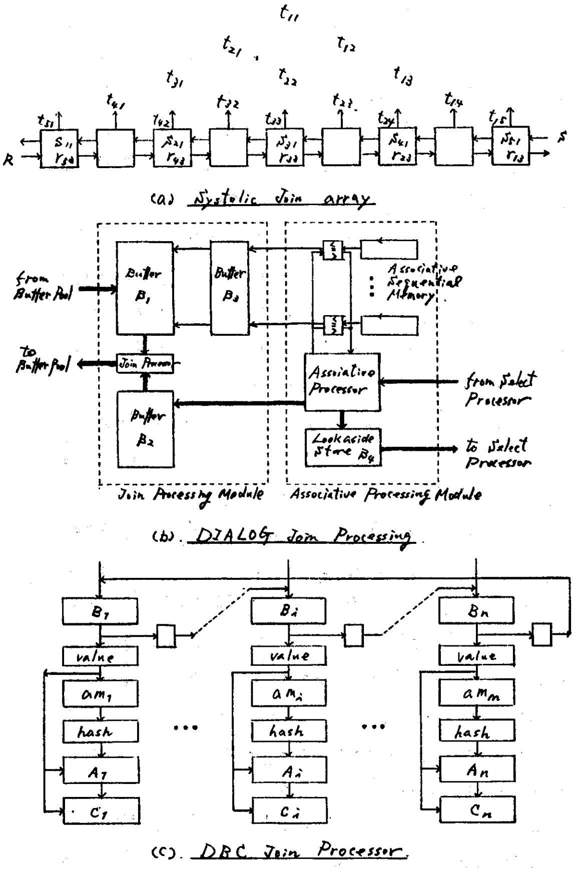
1. はじめに

半導体素子技術の進歩により，メモリ素子及びマイクロプロセッサの高性能，低価格化が進み，これらを利用した機能／負荷分散を実現する複合マイクロプロセッサシステムの提案及び研究開発が活発である。特に，データベースの分野においてはデータベースマシンとして研究が進められている[1,2)]。我々は，上記背景を踏え，リレーショナルデータベースシステムを効率的に実現するための複合マイクロプロセッサ及びデータベース機能専用ハードウェア構成のデータベースシステムについて報告した[4)]。本稿では，上記システムの構成要素の一つであり，リレーショナルデータベースにおけるジョイン演算処理を専門にサポートするJoin Processorについて報告する。

以下，第2章では，既提案Join Processor（ここでは，ジョイン演算処理専用ハードウェア及び，ハードウェアオリエンティドなジョイン演算処理方式をJoin Processorと呼ぶことにする。）をいくつか紹介し，第3章では，本稿で提案するJoin Processorで実現するジョイン演算処理の考え方について述べ，第4章で，提案Join Processorの構成及び動作概要について述べる。

2. 既提案Join Processor

リレーショナルデータベースにおけるジョイン演算は最も高負荷な処理の一つであり，当該処理の高速化のために，ソフトウェア及びハードウェアの両面から多くの研究がなされている[3)]。本章では，簡単に，代表的なJoin Processorについて紹介する。この中で，VLSIでのチップ化を意識しているJoin Processorの構成をFig. 1に示した。Fig. 1の(a)は，Kungらの Systolic Join array processorである[4)]。本方式は，2つのリレーションのジョインカラムを互いに逆方向から，array processorに入力し，Join Processor要素での比較結果をある行列Tの要素として出力し，パイプライン的にジョイン演算処理を行なう。Fig. 1の(b)は



(a) Systolic Join array
(b) DIALOG Join Processing
(c) DBC Join Processor

Fig. 1.

Wahらの提案するデータベースマシンDIALOGにおけるジョイン演算処理方式を示す図である[5]。本方式では，Join Processing Module と Associative Processing Module を用いてジョイン演算処理を行なう。B1になりレーションRのタップルを格納し，Associative Sequential Memory（ビットスライス連想メモリ）には、Rのジョインカラム値を記憶しておき，Select Processor から転送されてくるリレーションSのタップルのジョインカラムと上記連想メモリ内のRのジョインカラムとの比較を Associative Processor で行ない，カラム値の一致したSのタップルをB2に格納し，B3（ビット行列）の対応するビットを"1"にセットする。最終的には，B3のビットをみて Join Processor で実際のジョイン処理を行ない，Buffer Pool に転送する。次の(c)は，Menonらによるデータベースマシン DBC の Join Processor のアーキテクチャである[6]。B_iにリレーションSのタップルを読込み，"value"で必要カラムを抽出し，am_i（連想メモリ）にジョインカラム値を重複排除して記憶し，さらに，ジョインカラム値をhashしメモリA_iの対応するブロックに"value"で抽出したカラム群を格納しておく。この処理後，リレーションRのタップルをB_iに読込み，"value"で必要カラムを抽出し，am_iでジョインカラム値間の比較を行なう。マッチしたらジョインカラム値でhashし，A_i内の対応するブロックを指定して，このブロック内データとの実際の連結処理を行ない，その結果をメモリC_iに格納する。上記処理は，複数プロセッサで並列に処理され，また，リレーションSのタップルは propagation path により各プロセッサに転送される。

以上の他に，Join演算の前処理として hashed bit array[1] を用いる方式も提案されている[7,8]。また，国内では，北大，東大，広大等でJoin演算の高速化方式が研究されている。北大では，多種データベースマシンの提案及び開発が行なわれており，その中で DFDBC[9] は，カーテエンジン及びソートエンジンを用いたJoin演算処理方式が提案されており，DSDBC[10]ではさらに，処理対象データの静的及び動的クラスタリング手法が導入され，動的クラスタリングとセグメント内のソート処理は同時に実行できるようになっており，ジョイン処理の高速化が図られている。また，東大のGRACE[11]では，hashとsortを用いる方式を提案している。まず，Joinカラムに関しhashを施し，ジョイン対象タップルを動的にクラスタリングしメモリブロックに格納する。次に，当該ブロック内データをソートモジュールでソートし，この結果をもとに，対応するクラスタリング毎のジョイン処理がパラレルに実行される。また，広大のSPIRIT-Ⅲ[12]では，hashによらず，ドメイン単位にそのカラム値の分布情報をもとに作成されるグルーピングテーブルを用いるクラスタリング手法によるJoin演算処理の高速化方式が提案されている。グループ化には，連想メモリが用いられており，同一グループに属するタップルはチェインを専用ハードウェアで制御されバッファに保持される。

以上，いくつかのJoin Processorについて紹介した。これらは次のように分類できそうである。すなわち，

(A). 単純なパイプライン処理方式
- Systolic Join Array Processor[4]

(B). 連想メモリによる処理方式
- DIALOG Join Processor[5]
- DBC Join Processor[6]

(C). Hashを利用する処理方式
- LEECH[9]

・ CAFS[8]

(D). 動的クラスタリングを前処理として利用する処理方式

・ DSDBC[10]
・ GRACE[11]
・ SPIRIT-III[12]

この中で、DSDBCはクラスタリングと同時にクラスタ内データのソート処理を行なうことができる。

3. Join 演算処理の考え方

以下、Join は Equi-join を仮定する。

いま、リレーション R_1 と R_2 のジョインを考え、そのジョインカラムを各々 $R_1.C$ 、 $R_2.C$ とする。この時、これらのジョインカラム値の分布により Fig.2 に示すように、ジョインカラム値範囲内によって5つのケースが存在する。

Fig.2 において、ケース(i)の場合は、ジョイン処理は不要である。ケース(ii)の場合は、a または b がジョイン処理範囲で、もとのデータの一部分である。同様に、ケース(iii)及び(iv)では、c が、ケース(v)では d がジョイン処理範囲である。

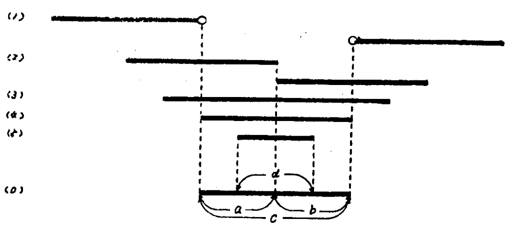
このように、ジョインカラム値の分布を知っておくことにより、ジョイン処理のための比較操作を大幅に減じることが可能である。例えば、ケース(i)の場合は、各ジョインカラム値の最大値/最小値を把握し、これらの間の比較操作により、ジョイン処理不要の判断が可能である。このためには、ジョイン処理の可能性を考慮し、前処理として、ジョインカラム値シーケンスをソートしておくことが重要である。

次に、Fig.2のケース(iv)をもう少し詳細にみてみると、ジョインカラム値シーケンスは、Fig.3 に示すようになっていると考えられる(重複値がない場合)。ここで、●印は、ジョインカラム値の一致、○印は不一致を示す。

このジョインカラム値シーケンスのジョイン処理において、上に示した(A)では、双方のシーケンスをプロセッシングエレメントにシーケンシャルに入力し比較操作を行ない、(B)では、連想メモリに一方のジョインカラム値群をセットして、他方のジョインカラム値をシーケンシャルに入力し比較操作を行なう。また、(D)では、何らかの方法で、ジョインカラム値シーケンスをクラスタリングし、対応するクラスタ間の比較操作を行なう。

以下、本稿で提案する Join Processor で実現するジョイン演算処理方式について述べる。本方式の特徴は次の通りである。(詳細後述)

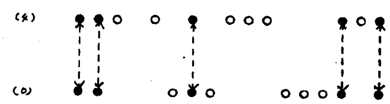
(1) 動的クラスタリングの採用、
(2)、ソートとクラスタリングの同時処理、



(0) : リレーション R_1 のJoinカラム値範囲 ($R_1.C(v)$).
(1)~(5) : 〃 R_2 〃 ($R_2.C(v)$)の
(0)に対するケース

● ケース (i). (0)と(1) (or(1'))の場合 : Join処理不要.
● 〃 (ii). (0)と(2) (or(2'))の場合 : a (or b) がJoin範囲
● 〃 (iii). (0)と(3)の場合 : c がJoin範囲.
● 〃 (iv). (0)と(4) 〃 : 〃
● 〃 (v). (0)と(5) 〃 : d がJoin範囲

Fig. 2



● : Match
○ : Non-Match

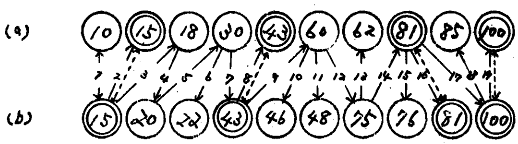
Fig. 3

(3). ソーティングハードウェア内データ(一つのクラスタでもよい)のクラスタ管理

である。ここで、クラスタのクラスタリングとは、Fig.4に示すジョインカラム値シーケンスを一つのソートされたクラスタとみなし、これをさらにソート処理と同様にクラスタリングすることである。

いま、2つのリレーションR_1, R_2のジョインカラム値$R_1.C$, $R_2.C$のソート済みシーケンスをFig.4に示すものとする。このシーケンス間での一致カラム値は15, 43, 81, 100である。このジョインカラム値シーケンスのジョイン処理をMerge-Scanで行なうと、Fig.4に示すように、19回の比較処理が必要である。しかし、次のようにカラム値シーケンスをクラスタリングすることにより、比較回数を減少させることが可能である。Fig.4に示すカラム値シーケンスを、カラム値の大きさにより、10以上20未満をクラスタ(1)、20以上30未満をクラスタ(2)、… 100以上110未満をクラスタ(10)というようにクラスタリングする。このクラスタリングをクラスタ単位10のクラスタリングという。このクラスタを管理するために、Fig.5に示すクラスタビットマップを導入する。ビットマップ内の"1"は、対応するクラスタに属するカラム値が存在することを示し、"0"は存在しないことを示す。これにより、ジョイン処理を要するクラスタは(1), (4), (8), (10)(Fig.5の●印を付したクラスタ)となる。よって、対応するクラスタ間でMerge-Scanを行なえばよい。さらに、クラスタビットマップに、各クラスタ毎のエントリ数を付加した形で、クラスタを管理する。これをJoinカラム値クラスタ管理マップという。Fig.6にこのマップ構成を示す。

以上の管理法により、Fig.4に示したカラム値シーケンス間の比較回数は半分以下に減じることができる。こ



(a) : Joinカラム $R_1.C$ のカラム値シーケンス

(b) : 〃 $R_2.C$ 〃

◎←--→◎ : Matchカラム値

←--- : マージスキャンシーケンス

Fig. 4

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
	●			●				●		●
(a)	1	0	1	1	0	1	0	1	0	1
	10—20	20—30	30—40	40—50	50—60	60—70	70—80	80—90	90—100	100—110
(b)	1	1	0	1	0	0	1	1	0	1

クラスタビットマップ(クラスタ単位:10)

● : Join対象クラスタ

(a) : Joinカラム $R_1.C$

(b) : 〃 $R_2.C$

(1)~(10) : クラスタ番号

Fig. 5

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
(a)	1	0	1	1	0	1	0	1	0	1
(b)	3	0	1	1	0	2	0	2	0	1
	10—20	20—30	30—40	40—50	50—60	60—70	70—80	80—90	90—100	100—
(a)	1	1	0	1	0	0	1	1	0	1
(b)	1	2	0	3	0	0	2	1	0	1

(1)~(10) : クラスタ番号

(a) : クラスタビットマップ

(b) : エントリカウンタ

Joinカラム値クラスタ管理マップ(クラスタ単位:10)

Fig. 6.

の効果は、クラスタ単位をうまく設定することにより向上させることができる。また、上記では、クラスタ単位を一定にしたが、これをクラスタ内で種々設定できれば、さらに効果を発揮するものと思われる。

以上、ここで提案するJoin Processorで実現するジョイン処理方式の基本的な考え方を示した。すなわち、ジョインカラム値シーケンスをソートし、そのソート結果をあるクラスタ単位にクラスタリングしておき、対応するクラスタリング毎のMerge-Scanでジョイン処理を行なうものである。提案Join Processorは、ジョインカラム値シーケンスのソートとそのソート結果のクラスタリングを同時に実行することができる。次章ではJoin Processorの実現方法について述べる。

4. Join Processorの概要

本章では、まず、ハードウェアオリエンテッドなデータベースシステム内でのJoin Processorの位置付けについて述べ、次に、Join Processorの構成について述べる。

(A). Join Processorの位置付け

我々は、リレーショナルデータベースシステムを効率的に実現するための複合マイクロプロセッサ及びデータベース機能処理専用ハードウェア構成のデータベースシステムの構想について報告した。本稿では、この詳細には触れない。Fig. 7. にそのシステム構成の概要を示す。この中で、データベースアクセス部に存在する〈SFPs〉(Special Function Processors)には次のものがあり、専用ハードウェアで実現される。

(1). Selection & Restriction & Pseudo-Projection Module.[15,16]

(2) Sort Module.

(3). Join Module (Join Processorに対応)., etc

すなわち、Join Processorは〈SFP〉の一つである。データベースアクセス部では、〈BS〉(Buffer Storage)あるいは〈SS〉(Secondary Storage)からのデータ転送ストリームに同期した処理が行なわれる。本システムでは、問合せを関係代数演算列に展開して処理するが、各演算処理毎に得られる結果データに対

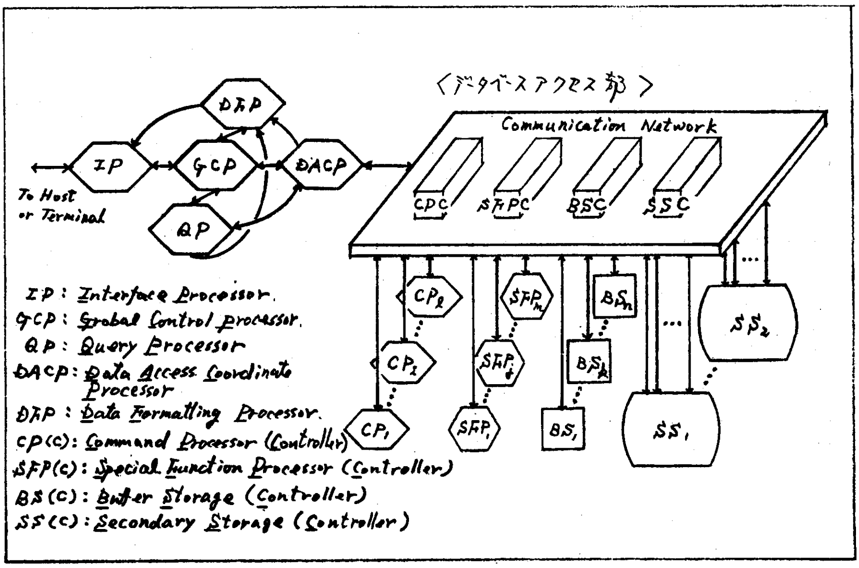


Fig. 7.

して、シーケンスNo.（S#とする）を付して管理する。従って，システム内のデータ流は（S#，データ本体）の形式である。S#は，システム内でユニークに設定される。

いま，次のSQL文[13]の処理を考える。

```
SELECT  R1.C1 , R2.C2
FROM    R1 , R2
WHERE   R1.C2 = R2.C3
  AND   R1.C3 = 'A'
  AND   R2.C1 = 'B'
  AND   R2.C4 < R2.C5
```

各カラムにはインデクス等特別なアクセスパスはないものとする。最適アクセスストラテジーとして，Selection及びRestriction，次にJoin，Projectionの処理手順が選択されたとすると，本システムでは，次のように実行される。まず，Selection & Restriction & Pseudo-Projection処理用〈SRP〉を用い，$R_1.C_3$ = 'A' の条件チェックと $R_1.C_1$ 及び $R_1.C_2$ のカラム切出し，$R_2.C_1$ = 'B' AND $R_2.C_4$ < $R_2.C_5$ の条件チェックと $R_2.C_2$ 及び $R_2.C_3$ のカラム切出しを，SDからの R_1，R_2 転送に同期して同時に実行する。この時，結果データには，上述したようにS#が付加され，〈BS〉あるいは次の処理モジュールに転送される。今の場合は，（S#1，$R_1.C_1$）がある〈BS〉に，（S#2，$R_2.C_2$）がまたある〈BS〉に転送格納され，〈S#1，$R_1.C_2$〉及び（S#2，$R_2.C_3$）は〈SRP〉の一つJoin Moduleに転送される。ここで，クラスタ単位は，データベースの初期作成時情報及びアクセスパターン（条件タイプ，条件に使用されるカラム等の比較）の統計情報に基づいて，アクセスストラテジー設定時に選択され，クラスタ制御部にセットされている。以上の下で，（S#1，$R_1.C_2$），（S#2，$R_2.C_3$）は各々，$R_1.C_2$，$R_2.C_3$ でソートされると同時に設定したクラスタ単位に基づい

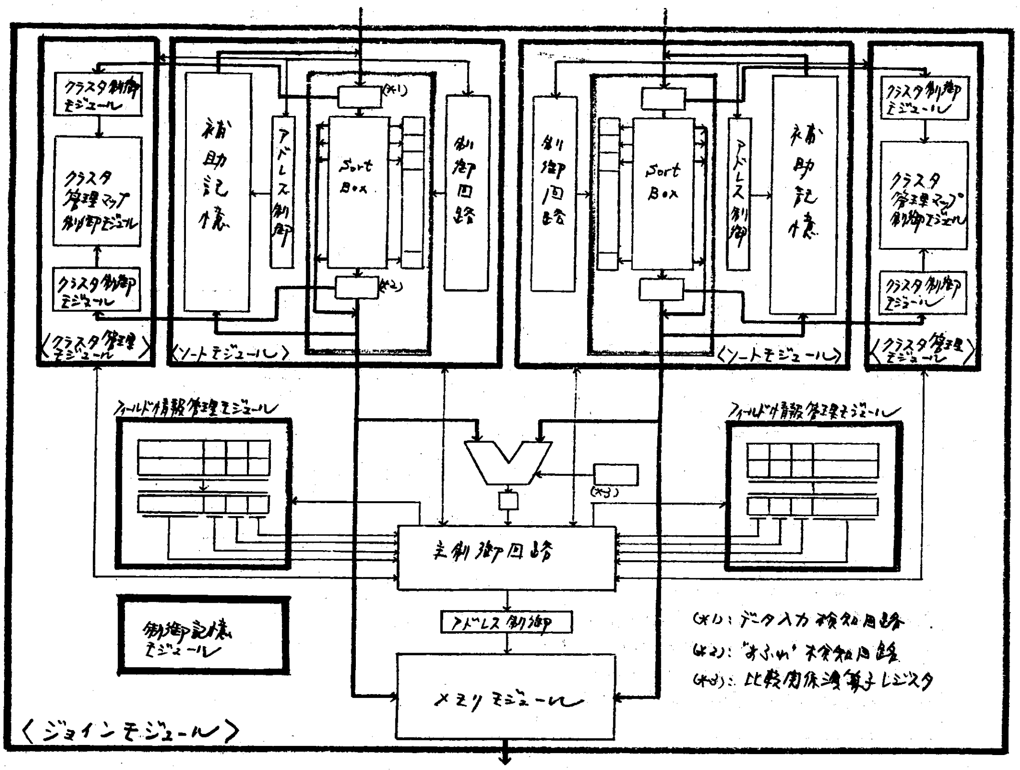


Fig. 8

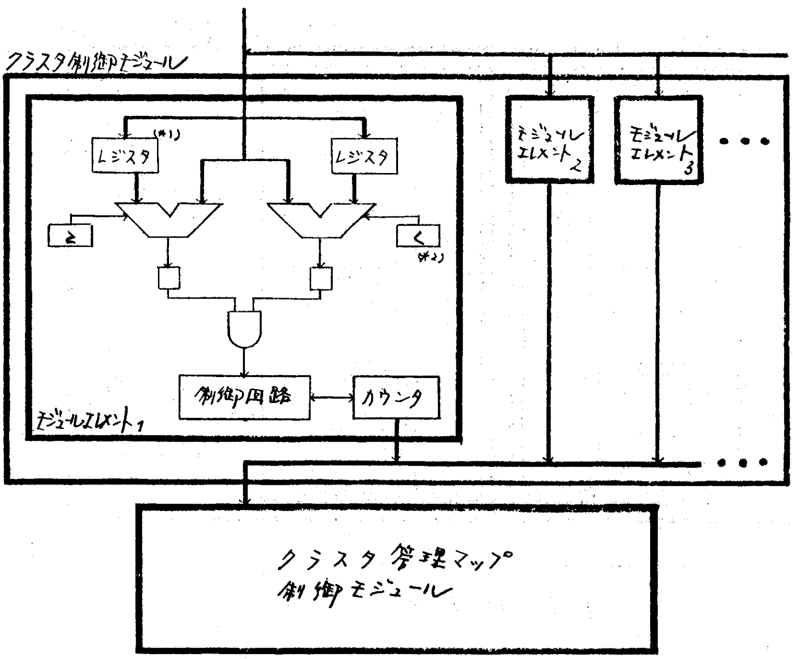
てクラスタリングされ，前述したように，ジョイン処理を要するクラスタ分の比較演算を行ない，〈S#1，S#2〉対をメモリモジュールあるいは他の処理プロセッサ（〈SFP〉及び〈CP〉）に転送し，これに基づいて，〈BS〉に格納した$R_1.C_1$及び$R_2.C_2$のデータを読み出し，連結する。次にこの結果に対し，重複排除（〈SFP〉を用いる）を施し，〈BS〉に格納する。全ての処理が終了したことを対応する〈CP〉に通知し，〈BS〉から結果データをデータベースアクセス部からシステムコントロール部に転送して処理を終える。

以上が，Join Processorの位置付けであり，関係処理概要である。次に，これらを実現するために提案するJoin Processorの構成について示す。

(B) Join Processorの構成

Fig. 8にJoin Processor（Join Module）の構成を示す。Join Processorは，ソートモジュールとクラスタ管理モジュール，フィールド情報管理モジュール，メモリモジュール[5,16]，及びこれらを制御するための回路，制御記憶から構成されている。Fig. 8には，2対のソートモジュールとクラスタ管理モジュールが示されているが，複数のソートモジュールの構成も可能である。ここでは，Fig. 8の構成に基づく処理動作の概要について述べる。

ジョインカラム値シーケンスの転送に同期して，データ入力検知回路に入力された（S#，ジョインカラム値）は，クラスタ管理モジュール及びSort Boxに転送される。Sort Box内はProcessing ElementのOne-dimentional structureである。この中では，データ転送に同期して，パイプライン的にソート処理が行なわれる。Sort Boxでは，このProcessing Element数だけのデータしかソートできず，"あふれ"たデータは，補助記憶に格納する。"あふれ"が生じた場合は，"あふれ"検知回路によりその制御情報をクラスタ管理モジュールに転送する。全ての対象データがソートモジュールに入力し終わると，



(*1)：レジスタには、クラスタ単位に基づいて必要なデータ値がセットされる。本モジュールは、連想メモリを用いても実現できる。

(*2)：比較演算子レジスタ

Fig. 9.

Sort Boxには、昇順あるいは降順のソート結果が、補助記憶には、その"あふれ"が格納されている。

ところで、Sort Boxへの入力及びSort Boxからの"あふれ"に応じて、クラスタ管理モジュールに制御がわたるが、これに対して、各モジュールは次のように動作する。まず、クラスタ制御モジュールに制御がわたる。このモジュールの構成をFig. 9に示す。レジスタには、設定されたクラスタ単位に基づいて必要な値をセットしておく。ソートモジュールからの転送データはクラスタ制御モジュール内のモジュールエレメント（クラスタ決定のための処理要素）にブロードキャストされ、ANDゲート出力が論理"1"のモジュールエレメントのカウンタがインクリメント（Sort Boxへのデータ入力時）、あるいは、デクリメント（Sort Boxからのデータ出力時）されて、クラスタ内のエントリ数が保持される。この情報に基づいて、クラスタ管理ビットマップメモリ内の対応する部分を更新する。Fig. 10にクラスタ管理モジュールの構成及びクラスタ管理マップ制御モジュールの構成を示す。クラスタ管理マップ制御モジュールは、クラスタ管理マップメモリ（クラスタビットマップ及びクラスタエントリ格納エリアからなる）とこれを制御する制御モジュールからなる。

全ての対象データに対するクラスタリングが終了すると、主制御回路に制御がわたり、ジョイン処理における比較演算が2つのSort Box内データに対して実施される。Fig. 8に示すように、比較データの情報はフィールド情報管理モジュールに保持されており、データ比較及びメモリモジュールへデータ格納の情報を提供する。まず、主制御回路は、2つのクラスタ管理モジュール内の対応するクラスタビットマップメモリアドレスにサーチ及びマッチングを行ない、マッチ（互いに"1"の場合）しない場合は、次のクラスタビットマップをチェックすると共に、比較演算すべきSort Box内のエントリア

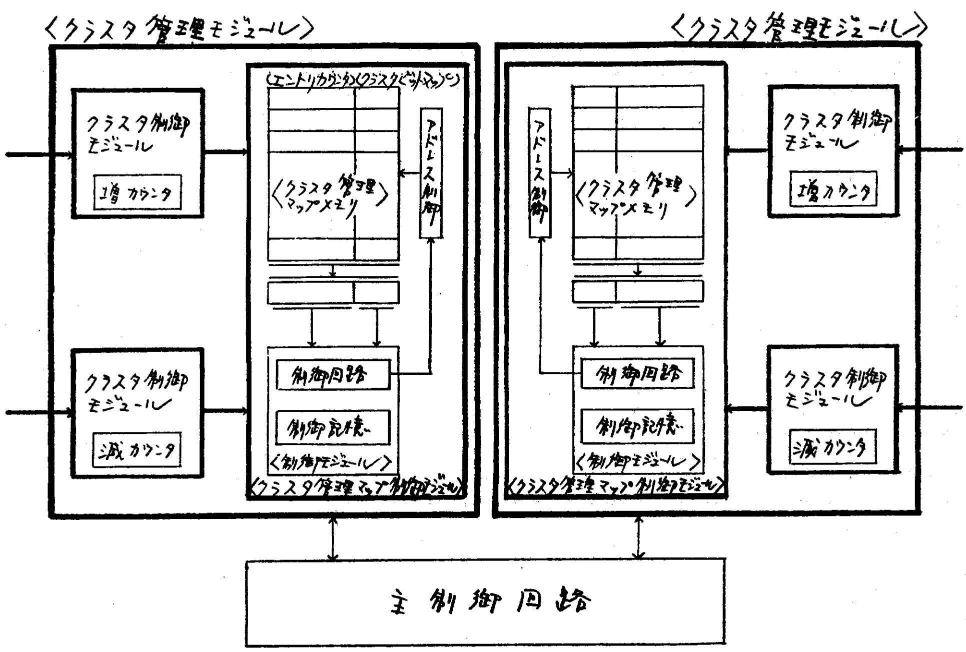


Fig. 10

ドレスを生成しておく。これにより、Sort Box内データの不要データをスキップしてアクセス可能となる。また、マッチした場合は、上記同様に、クラスタ内エントリ数を考慮して、Sort Box内の対応データのエントリアドレスを生成して、Sort Box内から必要データを同期させて読出し、比較演算を行なう。すなわち、クラスタ間では、制御記憶モジュールのファームウェアによりハード的にMerge Scanを制御する。比較演算の結果、一致した場合は、前述したシーケンスNoの対がメモリモジュールに記憶される。以上の動作をSort Box内の全てのデータに対して実施する。メモリモジュールに格納される情報は逐一他〈SP〉から他〈CP〉に転送され実際の結合処理が、上記処理と並行して実行することも可能である。

Sort Box内データの処理が終了すると、主制御回路は、双方のクラスタ管理モジュール内のクラスタビットマップを読み出し、パターンマッチングを行ない、対応するクラスタが互いに"1"の場合を除いて全てのクラスタビットマップの値を"0"にリセットし、再びクラスタ管理マップメモリに書き込む。(処理済みのエントリはクリアする。)この後、補助記憶から、Sort Boxへのデータ入力を行ない、前述のように処理を行なう。但し、クラスタビットマップが"0"に属するデータは、クラスタ管理モジュールからソートモジュールにフィードバックしてそのデータを無効とする。これによりソート及び比較演算の対象となるデータ数を減じることができる。補助記憶内の全てのデータに対して上記処理を行ない終了する。

5. おわりに

以上、ジョイン演算処理方式を提案し、これを実現するためのJoin Processorの構成について報告した。しかし、いくつか未検討の部分もあり、また、実現方式においても、他の方法を考えられる部分もある。例えば、クラスタ管理モジュールはかなり容量の大きくない連想メモリを利用して実現できるであり、また、補助記憶内のデータ管理に関しては、エリア内をクラスタ毎に分割しておく方法、あるいはクラスタ単位のリスト構造で管理する方法もある。後者に関しては、データ転送に同期してリスト構造が管理できれば、前述した方法より有効と思われる。

また、クラスタ単位の設定に関しては、初期データベース作成時のカラム値分布情報、アクセスパターン情報等を利用することを述べたが、これらの統計情報は多量化することが予想される。この管理方法は未検討である。本方式においては、このクラスタ単位をいかに正確に設定するかが非常に重要である。

ところで、本稿では、情報の係数として、数値データのみを扱ったが、他のデータタイプは符号化[2]で対処できると思われる。また、2つのSort Moduleによりジョイン演算実現方式について述べたが、これを多重化することは容易である。高多重度により高性能は期待できるが、実現規模(物量、価格)とのトレードオフを考慮する必要がある。

今後、未検討部分の検討及び性能評価を通して、詳細検討を行なう予定である。

<参考文献>

1) 田中:データベースマシン, 情報処理, Vol.23, No.10, pp.939-947(1982)
2) 植村,前川:データベースマシン, 情報処理学会(1980).

3). Tong, F. et al : Performance analysis of database join processors, ATIPS, vol. 51, pp 627-638 (1982)
4). Kung, H.T. et al : Systolic (VLSI) Arrays for Relational Database Operations, ACM SIGMOD, pp 105-116 (1980).
5). Wah, B.W. et al : DIALOG - A distributed processor organization for database machine, ATIPS, vol. 49, pp 243-253. (1980).
6). Menon, M.J. et al : Design and Analysis of a Relational Join Operation for VLSI, Proc. 7th VLDB, pp. 44-55 (1981)
7). McGregor, D.R. et al : High Performance for Database Systems, 2nd VLDB, pp. 103-116 (1976)
8). Babb, E : Implementing a Relational Database by means of specialized hardware, ACM TODS, vol. 4, No. 1, pp. 1-27 (1979)
9). Tanaka, Y. et al : Pipeline Searching and Sorting Modules as Components of a Data Flow Data Base Computer, IFIP 80, pp. 427-432 (1980).
10). Tanaka, Y. et al : A Data Stream Database Machine with Large Capacity, Proc. Int'l Workshop on Database Machines (1982)
11). 喜連川 他 : Hash & Sort による関係代数マシン, 信学技報, EC 81-35 (1981)
12). 上林 他 : 関係代数マシンSPRINT-IIにおける2ストリーム関係代数演算オペレータTuple Stream Filtersの実現方式, 信学技報, EC81-66 (1981).
13). Date, C.J. : An Introduction to Data Base Systems, 2nd Ed., p. 236, Addison Wesley (1977)
14). 佐藤 他 : マイクロプロセッサ及び専用ハードウェア構成のデータベースマシンの構想, 情報処理学会24回全国大会, 4G-12, pp. 581-582 (1982).
15). 佐藤 他 : データベース処理のための専用サテライトプロセッサの提案, 情報処理学会22回全国大会, 3L-6, pp. 499-500. (1981).
16). 佐藤 他 : データベース処理のための専用サテライトプロセッサ, 情報処理学会論文誌, Vol. 23, No. 4, pp. 349-357 (1982).