可変構造多重処理データベースマシンの構成

ARCHITECTURE
OF
DYNAMICALLY RECONFIGURABLE AND HIGHLLY PARALLEL DATABASE MACHINE

喜連川 優　　鈴木 重信　　赤松 宏恒
M．KITSUREGAWA　　S．SUZUKI　　H．AKAMATSU

田中 英彦　　元岡 達
H．TANAKA　　T．MOTO-OKA

東京大学　工学部
UNIVERSITY OF TOKYO　　FACULTY OF ENGINEERING

0. はじめに

近年、データベースの普及に伴い、ソフトウエア サポートによるDBMSの性能の限界が次第に明らかとなり、データベース管理の諸機能をハードウェアでサポートする事を試みたデータベースマシンの研究が各所でなされるようになった。(1)では、データベースマシンの観点から、DBMSの *Standards* を検討しておりその存在は不可欠となる様である。

データベースマシンは

1) セルラロジックタイプ
　RAP(4,5)　CASSM
　RARES　DIRECT(6)
　EDC(7)
2) ハッシュビットを用いるもの
　CAFS　LEECH
3) 連想プロセッサを用いるもの
　SETF，(8)
4) ソートを主体とするもの
　(9)，(10)
5) 分散データベース技法を用いるもの
　MUFFIN(11)

その他、最近では、ベクトルプロセッサによるもの(2)や、プロセッサ間をより疎に結合した分散型マシン DIALOG(3) 等、種々のタイプが提案されている。

ここでは 1) のタイプをより発展させた、"可変構造多重処理データベースマシン"及びそれによる、リレーショナルデータベースのサポートについて報告する。

1. 設計指針

1) 潜在的並列性の抽出

セルラロジックタイプのデータベースマシンでは 当該メモリにアクセス出来るのは、そのメモリに付加されたプロセッサだけであり、従って、処理対象となるリレーションの占めるセル数と等しい並列度までしか物理的に抽出し得ない構成となっている。Query 自体には、論理的に高次の並列性が潜在するにもかかわらず、ハードウェアアーキテクチャによってこれを制限していることになる。従って、任意のプロセッサが、任意のメモリにアクセス出来る構成、或は、1つのメモリに対して任意台数のプロセッサがアクセス可能な環境を作る事が望ましい。

2) プロセッサの有効利用

セルラロジックタイプのデータベースマシンでは、自分のメモリに有効データがない場合、プロセッサは完全にアイドル状態になる。データベース容量の増加に伴いセル数が増すと、プロセッサ機能の高級化とともにこのコストは無視できなくなり、プロセッサ、メモリのより柔軟な結合関係を実現することが望まれる。

3) マルチユーザ環境への適合性

データベースは多くのユーザによって同時に使用されるものであり、その処理は、SIMDマシンによってシリアライズする事は好

ましくない。多くのQueryに対し、各々に適したプロセッシングパワーを動的に割当て、並列に処理し得る様な、MIMDマシンアーキテクチャを採るべきである。

4) 磁気バブルの採用

近年、研究開発の盛んな電子ディスクの1つである磁気バブルを、メモリデバイスとして採用し、その柔軟な制御構造、及び、高アクセスタイム、不揮発性等の特徴に関して、データベースマシンへの適用性について検討する。

5) コンカレンシ、インテグティ等への配慮

データ操作とともに、DBMSとして重要な機能であるインテグリティ・チエック、及び、コンカレンシーレベルの向上を、実現しやすい構成とする。

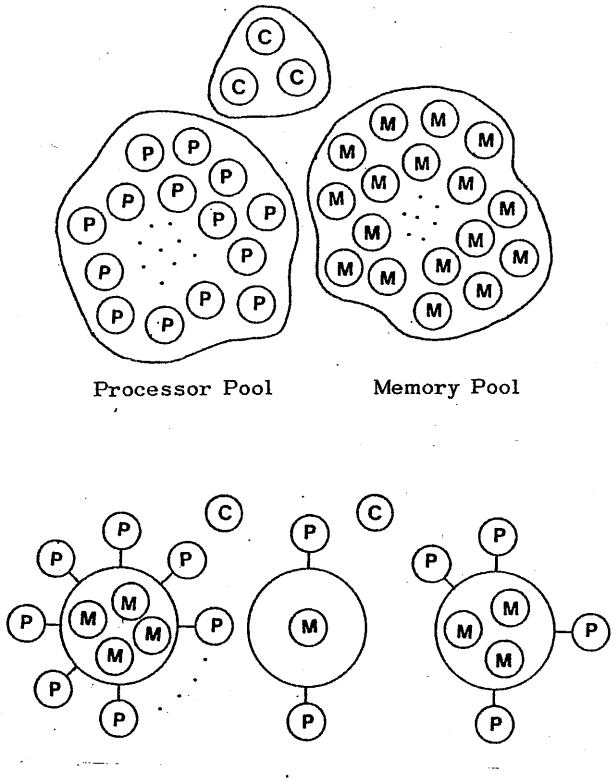


Fig.1 Dynamically Reconfigurable and Highly Parallel Data Base Machine

2. データ操作部のアーキテクチャ

2.0 基本構造

セルラロジックタイプでは、Fig. 2に示される如く、プロセッサとメモリの対応が固定的である。この為、前節1),2),3)の如き問題が生じてきた。本マシンのデータ操作部は、メモリモジュール、プロセッシングモジュール、及びコントローラから成り、各々が相互に任意の結合関係を生成できるアーキテクチャとなっている。即ち、Fig. 1に示される如く、その処理内容に応じ適当な台数のメモリモジュール群に対して適当な数のプロセッシングモジュール群が結合出来る。この駆動はコントローラによってなされる。当該処理に対してプロセッシングパワーを動的に結合、或は、変更する事が可能な可変構造性を有している。

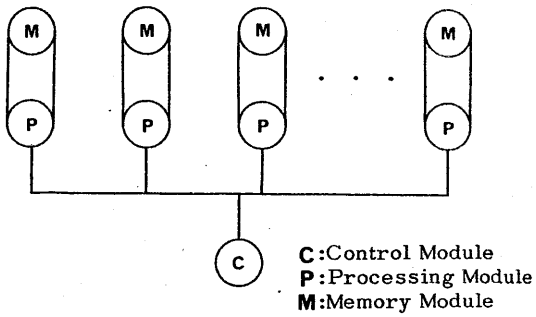


Fig.2 Cellular Logic Type Data Base Machine

2.1 ブロードキャスト方式

任意のプロセッサ、メモリ間結合を生成する事が好ましいが、一般にマルチプロセッサの環境では特定のメモリバンクに多くのプロセッサを結合すると、メモリコンフリクトが生じてしまい必然的に結合し得るプロセッサ数が限られてしまう。これは、個々のプロセッサのメモリに対する要求が異なる為、各々をシリアルに処理せねばならないからである。要求が共通であれば問題はない。一方、データベースの環境では、データの取り出しに際して連想機能を有する事が理想的であるが、そのメモリ容量が膨大である為、連想メモリはコストの点から非現実的となり、バルクメモリとそのフルスキャンによって実現することが考えられてきた。そしてこのフルスキャンという操作では各プロセッサの処理内容は異なろうとも、データは共通であ

り、メモリモジュールからプロセッシングモジュール群へのデータブロードキャストによって実現出来、メモリコンフリクトフリーの環境を生成する事が出来る。この為、任意台数のプロセッシングモジュールを結合する事が可能となり、しかも、結合台数に比例した処理能力の向上が期待できる。

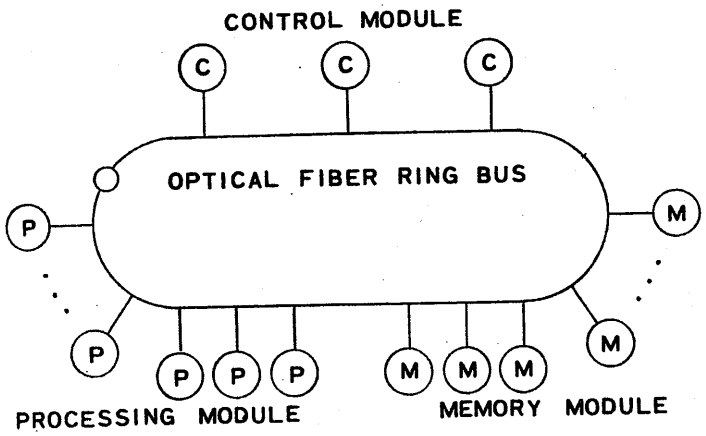


FIG.3 ARCHITECTURE OF DATA MANIPULATION UNIT

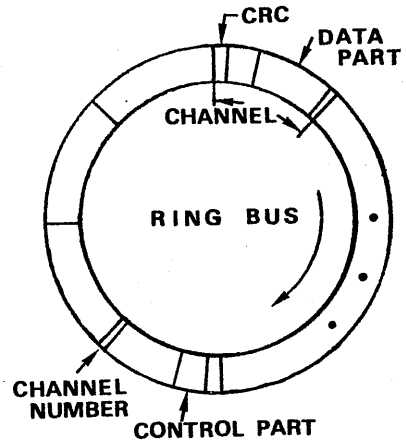


FIG.4 RING BUS DATA FORMAT

2.2 リングバスアーキテクチャ

本マシンのデータ操作部は、Fig.3の如き構成となっている。プロセッサ、メモリ群をほぼ完全に結合できる構成が望ましいが、モジュール数が多い為容易ではない。空間分割型のクロススイッチを用いると結合点数は$O(n^2)$となる為大規模マルチプロセッサの構成には適さない。ここでは、高いデータ転送レートを有するリングバス構成を採用し、結合点数を$O(n)$とした。リングバスは、時分割チャネル多重方式を採用し、対応するメモリモジュールとプロセッシングモジュール群を1つのチャネルを通して結合することが出来る。チャネルはコントローラによって割り付けられ、メモリからプロセッサ群に対してデータのブロードキャストが行なわれる。

チャネル多重方式では、チャネル数を変更する事は困難であるが、プロセッシングモジュール、メモリモジュールの増加は、初期設定値までは自由に行なう事が可能であり、メモリ容量の増大、及び、プロセッシングパワーの調整を比較的容易に行なう事が可能である。

リングバスは、Fig.4に示される如きチャネル形式をとっている。即ち、プロセッサは、メモリから送られてきたデータに対し(処理を施すが、これはバスの転送レートではなく、メモリモジュールからのデータ出力レートによって定まる為、プロセッサはその処理結果であるマークビット情報を当該データパートのチャネルよりもいくつか後のチャネル上コントロールパートに出力する。

3. テンポラリリレーションとマークビットに関する問題とメモリモジュールのインテリジェント化

RAPでは、全ての中間結果をタプルの先頭に設けたいくつかのマークビット(RAP3では16 bit)によって維持している。この方式では、所要メモリはきわめてわずかですむが、当該Queryがその処理を終えるまでずっとそのリレーションをLockしてしまう為、Query間の並列性を抽出する事が出来ず、大変アクセス頻度の高いリレーションの場合に問題になるという欠点がある。この観点から、DIRECTでは、中間結果をテンポラリリレーションとして動的に生成し、マークビットは採用していない。その他、DIRECTは、その方式によって、リレーションの恒久化が容易になり、又出力効率もよくなる為、所要メモリ容量、及び、ページトラフィックの増大は打ち消されるとしている。[6] 要するに、並列処理性とメモリ容量という相反する要求に対して適当な妥協点を見い出す問題に帰着される。ここでは両者の中間策を採用する事にした。即ち、確かにテンポラリリレーションを作る事によって並列性は向上するが

これはほとんどの場合ベースリレーションに限られるはずであり、中間結果を共有する事はほとんどないと考えられる。つまり、中間結果に対する演算を再び別のテンポラリリレーションとして生成する事はメモリ容量も要されるし、さらに時間もかかり大変無駄である。一方、或ベースリレーションへのアクセス頻度が極めて高い時、それに関するQuery群を並列に処理する事は重要である。よってここでは、ベースリレーションに関しては、それに対する演算結果をテンポラリーリレーションとして生成することによって早急に解放し、後の演算はその上にマーキングを施す事によって処理を進める事とする。

本マシンは、プロセッサとメモリを分割したという点では、DIRECTに、又、マークビットを用いるという点では、RAPに似ている点もあるが、ここではメモリモジュール側の機能がよりインテリジェント化されており、上記2つのマシンと大きく異なる。次の二点がメモリモジュールの特徴としてあげられる。

A.) タプルリダクション

ディスクやシリアルループのバブルを用いた場合には、必然的にレコードを連続出力せねばならないのに対し、メイジャ、マイナループ方式のバブルでは不要タプルを1bitタイムで読みとばすことが可能である。本マシンではマーキング用半導体RAMをバブルとは別に持ち、マークの施されたタプルのみを選んで送出する事が出来る。RAPでは先頭のデリートビットを見て、当該タプルを処理する必要があるかをチェックしており、もし1だと、そのタプルが出力される間、プロセッサはアイドルとなる。一方DIRECTでは、メモリモジュールは単にデータを垂れ流しているだけである為、いかなる場合も全メモリページを読み出さねばならない。即ち、リレーションに対して処理をすすめていくと有効タプルが僅かしか入っていないメモリモジュールが増えてくるのが普通であるが、これらに対して演算を施そうとすると、実際にはわずかな量のタプルを読む為に、多くの不要な意味のないデータを取り込まねばならず、大きなパフォーマンスの低下が生ずる事は明らかであり、しかもこの様な状態はしばしば生ずるものと考えられる。DIRECTではこの読み込みに関する無駄をなくす為に「COMPRESS」という操作によって、散在するリレーションの破片を1ヶ所にまとめるという事を行なっている。ところがこのオペレーションは、それ自体を並列に処理すると又フラグメンテーションを生じてしまう為、それ程並列度を上げるわけにはいかず、時間のかかるオペレーションとなってしまう。そこで、COMPRESSを行なうのに最適なプロセッサ数を計算するといった複雑な問題が生じてくるが、本マシンの場合には、この様な問題は一切生じない。

尚RAP3ではバブルを使う様であるが、これは単なるシリアルループタイプのものを用いそのStart-Stop制御性のみを利用せんという点で本マシンとは異なる。

B.) ドメイン選択機能

不必要なタプルを送出しないばかりでなく、さらに不要なアトリビュートも送出しない。即ち、必要なタプルの必要なアトリビュートだけを連続的に送出する機能をメモリモジュールにもたせている。データベースマシンでは一般にデータの出力レートに密着した形で処理を行なうという考えが根底にある。従って、バブルチップから一旦、完全な形のタプルを読み出し、それを整形して必要な部分だけを取り出して送出するのでは、実質的なメモリのデータ出力レートは何ら向上しない。ここでは、バブルチップのレベルから一定のレートで必要なアトリビュートだけを取り出せる機能を考えている。これによって、大きなパフォーマンスの向上が期待できる。バブルチップについては、6節で再び述べる。

4 関係代数の処理方式

データ操作部の言語インタフェースは、Sets Orientedな関係代数とする。即ち、コントローラから、プロセッシングモジュールに対して与えられるコマンドは、関係代数のオペレータ単位である。サポートするオペレータとして以下のものがあげられる。

- RESTRICTION
- JOIN(EXPLICIT)
- IMPLICIT JOIN

・PROJECTION
・DIVISION
・DIFFERENCE
・UNION
・INTERSECTION
・UPDATE
・INSERT
・AGGREGATE OPERATION

ここで、先に述べた如く、メモリモジュールはドメイン選択機能がある為、単なるアトリビュート抽出のプロジェクションは、メモリ側によってなされ、必要な部分だけがプロセッシングモジュールに送られる。上に記したPROJECTIONは、重複除去を意味する。

1. RESTRICTION

RESTRICT(R, g) = R[g]
= {r : r∈R∧g(r) = true}
R : Relation　r : Tuple Variable on R
g(r) : g(r[1], r[2], ⋯, r[m])　検索条件

処理方法

プロセッサは割り当てられたチャネル(当該メモリが対応する)から送られてくるQualification Attributeに対して検索を行ない、マッチするとマーキング情報をメモリへ送る。メモリはそれを受けとるとマークビットをセットし、当該タプルを記憶する。当該メモリの全タプルの検索が終わった時点でそのモジュールに対する処理が終了する。(Fig.5)

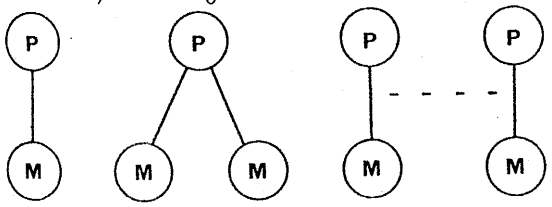


FIG.5 PROCESSOR ALLOCATION IN RESTRICTION

当該リレーションがベースリレーションでアクセス頻度が高い場合には、別モジュールにテンポラリリレーションを生成する。RESTRICTIONの場合、メモリモジュールの台数分までの並列性しか抽出する事が出来ない。

2. JOIN (Explicit)

CODDによればθ-JoinはリレーションR_i, R_jに対するドメインリストをA, Bとすると

$$R_i[A\theta B]R_j = \{rs : r\in R_i \wedge s\in R_j \wedge (r[A]\,\theta\, s[B])\}$$
$$= \{rs : r\in R_i \wedge s\in R_j \wedge (r[a_1]\,\theta\, s[b_1]) \wedge \cdots \wedge (r[a_k]\,\theta\, s[b_k])\}$$
$$\theta = \{=, \neq, <, \leq, >, \geq\}$$

と定義される。本マシンでは条件部はより一般的に任意のPredicateが書けるものとする。JOINの後にRESTRICTIONがある場合、最適化ルーチンが一方のリレーションのみのアトリビュートに関するRESTRICTIONをJOINの前に行なう事によってその負荷を軽減する事はよく知られている所である。この結果、残るRESTRICTIONは両リレーションのアトリビュートを含む条件式から成り、これは拡張JOINの条件部に取り込む事が出来る。

又、定義によれば生成タプルは両リレーションのタプルを連結したものであるが、不要アトリビュートを含む長いタプルは出力競合を招く為所要の整形を施すものとする。

従って　f = f(r, s) なる真理値関数、及び
T = (t[1] = g_1, t[2] = g_2, ⋯, t[l] = g_l)
g_a = g_a(r, s) = g_a(r[1], ⋯, r[n], s[1], ⋯, s[m])
出力関数リストに対し、本マシン上でのRiとRjのEXPLICIT JOINは

JOIN(Ri, Rj, T(g); f) = {T : r∈Ri∧s∈Rj∧f(r, s) = true}

と定義される。

• 処理方式

2つのリレーションのJOINを実行する際一方を、ソースリレーション、他方をターゲットリレーション、結果をリザルトリレーションと呼ぶ事にする。これらのリレーションは幾つかのメモリモジュールにより構成される。プロセッサは先ずソースリレーションにリンクし、一定数のタプルを取り込む。その後ターゲットリレーションにリンクし、両リレーションのタプルを結合アトリビュートに対してマッチングを行ない、条件が真の時新しい生成タプルをリザルトリレーションへ出力する。以上の動作を多くのプロセッサが並列に処理する事によって、高速にJOINを実行できる。出力競合が問題にならない場合には、駆動プロセッサに比例した処理速度の向上が期待できる。(Fig 6)

• ソースリレーションに於けるタプルフェッチの相互排他制御

プロセッサ群は先ず、ソースリレーションから任意に一定個のタプルを取り込むが、この時

同一タプルが複数のプロセッサによって取り込まれると結果に重複を生じ、又、無駄な処理を行なう事になる。従って、ソースリレーションのタプルフェッチに関しては相互排他制御が行なわれねばならず、当該リレーションを形成するメモリがこれを実行する。メモリは当該チャネル上データパートにタプルを出力後、コントロールパートに相互排他制御用ビットを設け、これを1にしてプロセッサ群にブロードキャストする。位置的に最も優位なプロセッサがこれを0にする事で、当該タプルを獲得したとみなす。又、これが0で来ると、あきらめて他のタプルを求める。メモリにとっては1のままで戻ってくると受けとられなかった事を示し、再送出の対象とする。一方、0の場合にはタプルリダクションを行ない当該オペレーション中再び出力する事はない。メモリはこれをマークビットで管理する。

• 生成タプルの出力

プロセッサはターゲットリレーションを検索中、条件が満たされると結果タプルを作り、リザルトリレーションを形成するメモリのいずれかに出力する。プロセッサ台数に比べ出力モジュール数は一般に少ない為、多くのプロセッサにより空チャネルを奪い合って出力する事になる。出力競合が予想される為、プロセッサは或程度の出力バッファを用意する必要がある。hit ratioが高い場合、バッファオーバフローの可能性があるが、これに対しては、ターゲットリレーションの有効タプル数は一定であるため、バッファが空くまで、読みとばしたタプル数を憶えておく事で再処理可能となる。又出力モジュールが一杯になった場合、新しいモジュールの割付はコントローラによってなされる。

3. IMPLICIT JOIN

Implicit Join はJOINの特殊形であり結果タプルを作るのに必要なアトリビュートが何れか一方のリレーションにのみ存する場合をさす。Join では結果タプルの出力の為に最終的には出力モジュールのバンド幅によって処理速度の上限が押えられるが、Implicit Join ではマーキングによって結果を表わす事が出来る。

ターゲットリレーションのタプルは一旦マッチすると以後そのタプルを検索対象とする必要はなく、メモリモジュールは、タプルリダクションを行なう。プロセッサは処理開始時の有効タプル数分だけ検索すればよく、動的にタプルが減少するため、処理が進むにつれて検索時間が少なくてすむ様になる。又 メモリはデータを一定順に出力する。

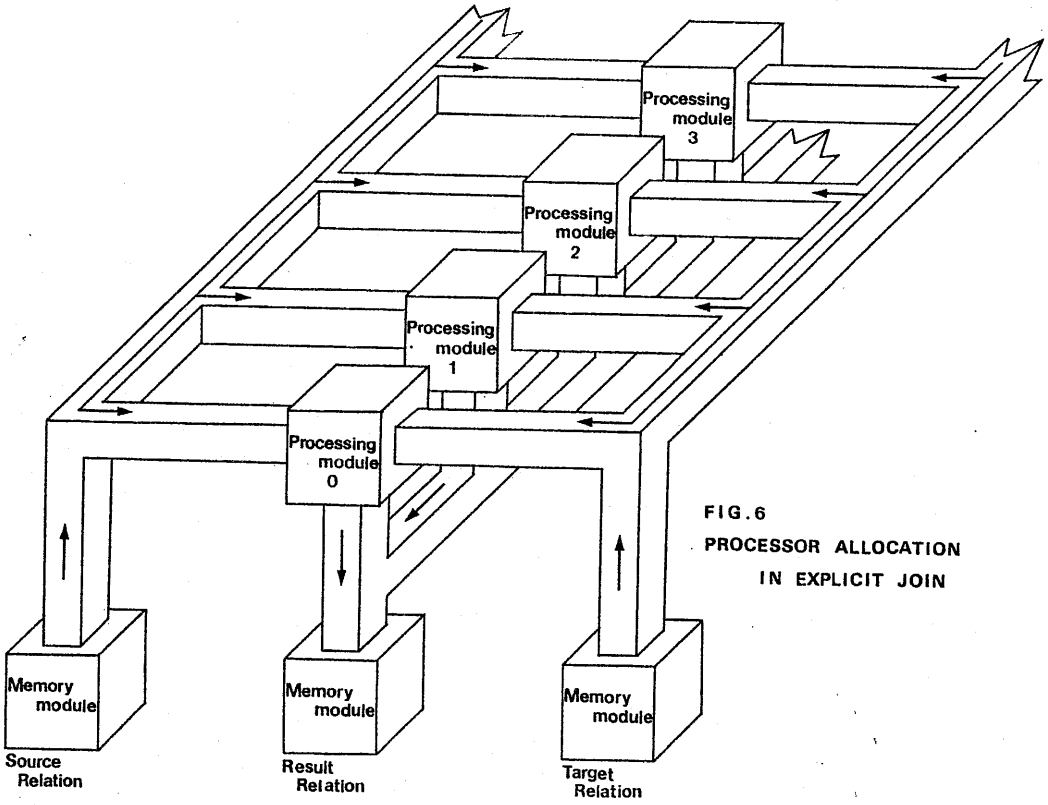


FIG.6
PROCESSOR ALLOCATION IN EXPLICIT JOIN

5. プロセッサ群制御

コントローラは、関係代数の系列として表わされたQueryに対してメモリモジュール、プロセッシングモジュールを駆動し、その実行を監視する。プロセッサ群の割付けは関係代数のオペレータレベルとした。

数多くのプロセッサを管理するに当たって、個々のプロセッサを意識していたのではその制御出来る台数に限りがある。従ってここでは効率よく制御する為に、プロセッサを群として扱うことにした。コントローラは各オペレーションに対しプロセスIDを付加し、それに対して駆動したプロセッサの台数のみを管理する。プロセッサを割付ける時も数だけを意識し、当該Queryの優先度、及び、当該オペレーションの性質から適当と考えられる台数分のプロセッサ要求を発行し、そのプロセスに割当てる。

コントローラはオペレーションに対してきめの細かい動的な割付けを行なうことが出来る。即ちオペレーション実行中に優先度の高いQuery.が投入されると、コントローラは実行中のプロセスから当該プロセスへプロセッサ群移動を指示する事が出来、それと同時に、もとのプロセスも処理能力を落として続行することが可能である。マークビットの退避を行なえば、プリエンプションも行なうことが出来る。

6. メモリモジュールの構成

RAP2, DIRECTではCCDを用いているが、不揮発性、将来性、さらにその制御の柔軟性からEDC同様バブルを用いる事にした。

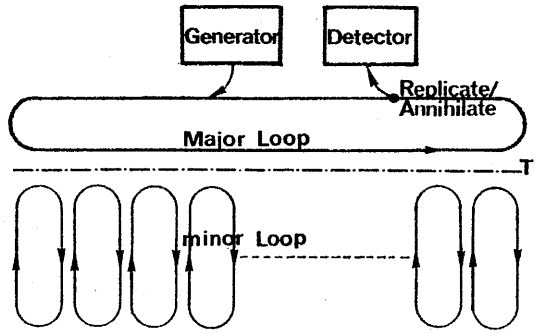
6.1 バブルチップの構成 (12)

バブルメモリはその原理からパーマロイバーパタンタイプ, コンティギュアスディスクタイプ、ラティスファイル等に分けられるが、市販、実用化されているのはオーのタイプだけであり、これはさらに、次の如く分類される。

A) シリアルループ方式
B) メイジャ・マイナ方式
  a) メイジャループマイナループ方式
  b) メイジャラインマイナループ方式
    I) ブロックリプリケーション方式
    II) スワップ方式

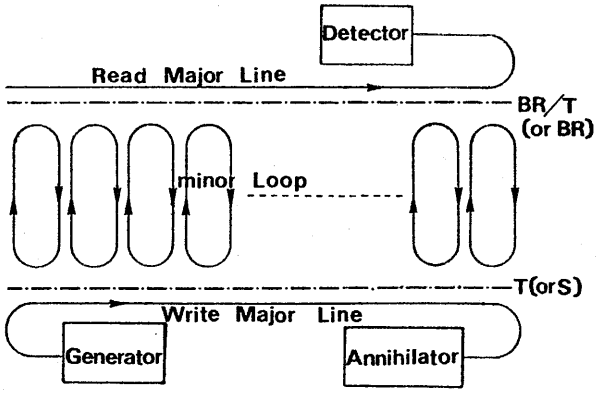
A) はディスクの1トラックイメージを生成するものであり非常にシンプルな構成となっている。一方、B) の構成ではこれをより改良しブロック単位のアクセスを可能にしている。a) ではトランスファゲートと1つのリプリケータを用いる為マイナループからの破壊読み出しとなってしまい、全数サーチによるデータベース処理ではサイクルタイムが重要である為この方式は適さない(Fig.7)。一方 b)-I ではマイナループからのブロックリプリケーションを行なうことでサイクルタイムとアクセスタイムを等しくし大きく改善される。しかしこの方式も書き込む場合には、まず当該位置のバブルを取り除く必要がある。b)-II のスワップ方式では読み出しと書き込みの2つを完全に独立動作させることができる。(Fig. 8)

以上の如くバブルチップには現在種々の改良が加えられている段階である。又バブルの他の研究分野としてバブル間相互作用を用いたバブルロジックがあり、これによってロジックイン



T ; TRANSFER GATE

FIG.7 SINGLE REPLICATION TRANSFER (CONVENTIONAL M/m) BUBBLE



BR ; Block Replication Gate
S ; Swap Gate

FIG.8 BLOCK REPLICATION TRANSFER (or BLOCK REPLICATION SWAP) BUBBLE MEMORY

メモリを構成しようとする　もあるが高速動作は近い将来実現性が薄い為ここでは考えない。

ディスク等の単純なシリアルループ構成の回転型メモリを用いた場合には、処理を進めるにつれて、或はデータの削除が行なわれるに際し当該モジュール中に不要タプルが多くなってくると考えられるにもかかわらず全てのタプルを読み出さねばならず、プロセッサのアイドルタイムが増大する。一方メイジャマイナ方式のバブルを用いた方式では不要なタプルは1ビットタイムで読みとばす事が可能であり必要なタプルだけをかなり連続した形で取り出せる。さらにメイジャラインとマイナループの間にバッファ（アイドラ）を置く事により当該タプルの読み出し中に次のタプルを取り込む事が出来、従ってほとんど連続したタプル出力が期待できる。(Fig.10)

しかし、この方式を採用したとしても長いタプルの内で必要なアトリビュートがわずかである場合には、大変無駄になる。そこでここではドメインセレクティブバブルを採用し、必要なタプルの必要なアトリビュートだけを連続して取り出せる様にする。（Fig 9）分割されたリプリケーションゲートを適当に制御する事によって不要なアトリビュートを除き必要な部分のみが連結された短いタプルとして取り出す事が出来る。バブルでは記憶密度を上げる為に径を小さくするが、逆にこれによって検出用ストレッチャが大きな場所を占める為、一般に1チップに多くのディテクタを置く事は出来ない。よってディテクタまでの距離が等しくなる様にバッファ量を調節する必要がある。又更新に伴う書き込みについてはフルタプルの形で行なう。これは一般にインテグリティチェックの為 Change Set を生成することに依る。現在この様な機能を備えたバブルは存在しない為、半導体RAMによってこれをシミュレートしている。

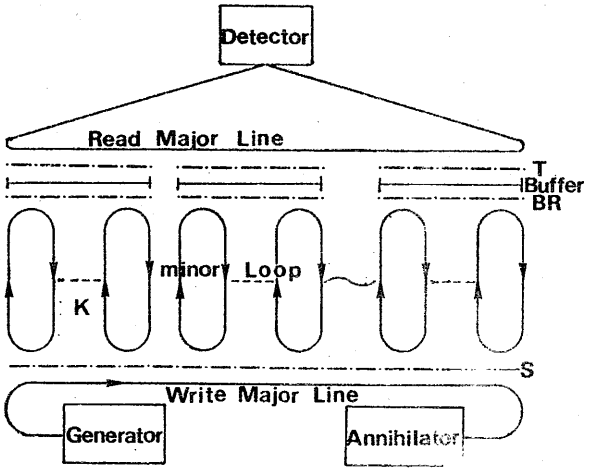


FIG. 9 BLOCK REPLICATION — SWAP BU... WITH BUFFER AND DOMAIN SELE... TRANSFER MECHANISM

6.2 リレーションの格納

本マシンでは、メモリモジュールがインテリジェント化されており、1モジュール内に格納されるリレーションの数に制限はない。各々マークビットで管理される。又タプルはマイナーループと垂直な方向に貯えられドメイン選択機構が有効である様、各アトリビュートは固定長とする。

6.3 メモリモジュール

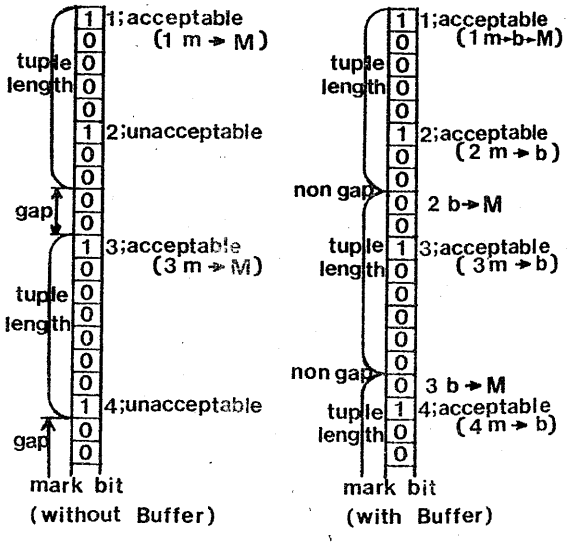
メモリモジュールはバブル上にリレーションを貯え、管理するとともに以下の如き機能を有する。

1. バブルチップの制御
2. マークビットの管理
3. タプルデータの整形とリングバスへのデータ伝送制御
4. 初期化、実行管理　その他」

ここではこれらの機能について試作検討する為、4つのユニットを用いてメモリモジュールを構成した。各々は現在μP上のソフトウェアで実現されている。(Fig. 11)

a. BSU（Bubble Simulation Unit）

前節で述べたバブルの機能をμPと半導体RAMを用いてシミュレートしており、バッファは一段である。読み出し時には、m→b, b→M



m ; minor loop　b ; buffer　M ; Major Line

FIG. 10 EFFECTS OF BUFFER IN M/m BUBBLE

の信号によってマイナループからバッファへ、バッファからメイジャラインへのバブル移動を行なう。タプルはディテクタに到着するとパイトシリアルにTCUへ出力される。逆に書き込みオペレーション時には、TCUから入力されこれがスワップゲートを用いてマイナループに書き込まれる。

b. BTCU (Bubble Timing Control Unit)

バブルに対する制御信号を発生する事が主な役目であり、マークビットを監視しながらゲートのON/OFFを制御する。マークビットはマイナループ長だけ設けられ、マイナループ上の当該位置が有効か無効かなどタプルに関する種々の状態を示すのに用いられる。オペレーションによっては一定順の読み出しが要求され、この場合にはマークビットチェインが形成される。BTCUはBSUと同期したバブルクロックが供給されており、これによってマイナループの当該位置とマークビットの対応関係を維持している。又、オペレーション開始に先だちTCUよりコマンドが与えられ、マークビットの初期化を行なう。

読み出しオペレーションの場合、BTCUはマークビットが1で且つバッファが空の時BSUに対してm→bの信号を送り、又メイジャラインが空になると、バッファが空でない時b→Mの信号を送出する。さらにタプルがディテクタに達するとTCUへMoutの信号を送り、タプル出力を知らせる。一方、書き込みオペレーションの場合には、TCUから書き込み要求(writable)が来ると適当な空領域を見出し、遅延を考慮に入れTCUへMinを送りタプル出力を促す。以上の如くBTCUはバブルからの出力データに直接作用する事はなく、一切TCUに任せられている。又プロセッサからの返答によりTCUからマークビットの更新要求がくるとそれを反映させる。

C. TCU (Tuple Control Unit)

以下に示す如く、バブル制御以外の重要な機能がTCUでサポートされる。第一にタプルの入出力処理を行なう。BTCUからMout信号があるとBSUからのタプルを入力し、適当な制御情報を付加する等タプルを整形した後、リングバス上の伝送制御手順(Broadcast Hand Shaking, 3-Phase Hand Shaking (16))のもとで当該チャネルに出力する。書き込み時にはMinでBSUへデータを出力する。第二に、オペレーション実行中プロセッサ群からの制御情報を解釈し、BTCUに対してマーキングの指定を出す。第三に、コントローラからのコマンドを受けそれを解釈し、BTCUを駆動する。即ち、読み出しか書き込みか、読み出しである時はいかなるマークビット条件のタプルを読み出すのか、どのアトリビュートを選択的に読み出すのか、又一回スキャンなのか一定順反復読み出しか等の指令を与える。第四にオペレーション実行中の監視を行なう。読み出し時にはバブル内の有効タプル数を、書き込み時には空き領域の数を管理し、オペレーションの終了を判定するとコントローラに報告する。又、タプルの出力エラーなどでTCU内のバッファ領域があふれそうになった場合等は、バブルクロックを止める事によって、その出力制御を行なう。

d. サービスプロセッサ

メモリモジュールとしては、上記3つのユニットから構成されるが、現在その機能の明確化及び動作の確認の目的から、回線制御以後のリングバス、コントローラ、プロセッシングモジュールを1つのμPによって代用している。サービスプロセッサはメモリモジュール開発の為に設けたもので、他のモジュールのメモリ空間を全てアクセス出来る様に構成されている。

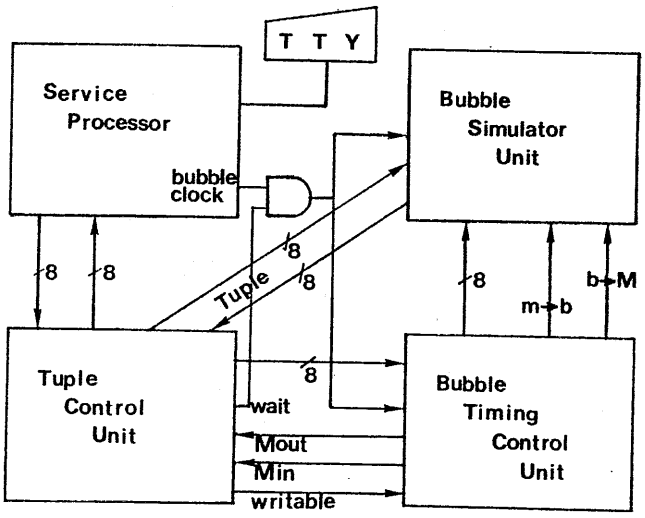


FIG.11 MEMORY MODULE ORGANIZATION

7. インテグリティと更新処理

インテグリティのサポートとは、データベース内の全てのデータを常に正しい値に維持する事を指す。正しい値とは言っても目的の達成される程度には限度があり、個々のデータに対して完全に正しいかどうかチエックする事は原理的に不可能である。即ち、そのデータが尤もらしいかどうか(Plausibility)をチエックする事しか出来ない。Integrity Enforcer の役割はデータを不正な変更から防ぐ事にある。(13)

インテグリティは、およそ以下の如く分類される。

i) Domain Constraints
ii) Relation Constraints
a) Tuple Assertion - Set Assertion
{ Aggregate Assertion / Functional Assertion / Set Relationship Assertion
b) State Assertion - Transition Assertion
c) Immediate Assertion - Deferred Assertion

- データベースマシン上での更新処理方式

インテグリティは上述の如く分類されるが、処理方式の観点からは次の様になる。Immediate 及び Deferred Assertion はインテグリティチエックの時点による分類であり、チエックの仕方に影響を与えるものでない。又 State 及び Transition Assertion は更新の変化に着目するか状態に着目するかの違いであり、この処理もマシン上では区別する必要はない。Tuple Assertion と Set Assertion のみ処理方式が異なってくる。即ち、Tuple Assertion では当該タプルを更新する時点で、それがインテグリティを犯すかどうか判るのに対し、Set Assertion ではタプル間の関係を含む為、一般に当該リレーションに関する更新を全て処理し終えてはじめてリレーション中の他のタプルとの関連で正しいかどうか決定出来るからである。Set Assertion には最大最小平均などを含む Aggregate Assertion, 関数従属性に関する Functional Assertion などがあげられる。

- 1リレーション上の Tuple Assertion を有する更新処理

更新は一般に2つのステップからなる。第一ステップで更新すべきタプルにマークを施す。第二ステップでメモリはフルタプルを送出し、プロセッサはそれを更新し返送する。更新対象は一般に複数のタプルから成り、全ての更新タプルに対して Assertion が満たされて始めてその更新が許される。従って更新を行なうには、

i) 予め全ての更新タプルに対して Assertion が成立するかどうか調べ、正しければその後実際に更新し、駄目な場合は何もしない。
ii) Assertion Check と同時に更新を直接行ない満たさないタプルが出てくると今までの処理を元通りにする。

の2通りがあるが、処理の高速性からここでは Change Set を用いて ii) を実現する。即ち更新と Assertion チエックは並行して行なうが、新しいタプルは元のタプルとは独立に別の場所へ生成し、マークビットにより管理する。この為 Assertion が満たされない場合、マークビットの処理丈で高速にバックアウトする事が出来る。Assertion が成立しないタプルが見つかった場合、その時点で処理を中止するか或は一応全てチエックするか等の指定はユーザによって指定されるものであり、コントローラはこれによって制御する。又正常終了した場合にはメモリはコントローラの指示により更新対象タプルを除き生成タプルを有効タプルとするマークビットの操作を施す。以上の操作によって更新操作が終了する。

- 複数リレーション上の Tuple Assertion を有する更新処理

タプル内のアトリビュートの関係を示す Assertion であるが、他のリレーションのタプルを参照する場合をさす。即ちJOINした生成リレーション上での Tuple Assertion などがこれに相当するが、この際必ずしもJOINによりリレーションを作った後チエックをしなくとも直接行なう事が出来る場合も多い。その時、IMPLICIT JOINと同様の手続きにより、多くのプロセッサを駆動し、チエックする事が可能となる。

- Set Assertion を有する更新処理

Set Assertion の場合には、一旦無条件に更新を行ない、その後 Assertion をチエックする。もし正しくないと、その時点でマークビット操作によりバックアウトされる。チエックの間メモリは更新後のイメージを示す必要があり、更新前のイメージと共に出力する場合には、区別する為の制御情報をプロセッサに与える必要が

ある。

例えば、関数従属性に関するAssertionのチェックでは、多くのプロセッサを駆動し、PROJECTIONに似た処理を行なう事で高速にチェックする事が出来る。

• Deferred Assertionとトランズアクション

トランズアクションとは、一単位の更新操作であり、その前後ではデータベースは無矛盾であるが、途中では必ずしも保証されていない。この様なトランズアクションに対しては一連の更新を実行した後にAssertionのチェックを行なう必要があり、これがDeferred Assertionである。更新用マークビット列が1つだけの場合には、バックアウトによって最初の状態にしか復帰できないが、これを複数ヶ設ける事によって適当な状態にまで復帰する事が可能となる。尚、これらのバックアウトはRAM上のビット操作のみで終了する為オーバヘッドはわずかである。

以上の如くマシン上での処理の観点からは

Immediate	Tuple	Assertion	IT
Immediate	Set	Assertion	IS
Deferred	Tuple	Assertion	DT
Deferred	Set	Assertion	DS

に分けられ、{IT, IS}, DT, DSの順でチェックされる。

• Integrity Enforcer

各リレーションの各アトリビュートに対して、それが更新された場合影響するAssertionの集合を管理しており、更新要求に対してこれらを引き出してくる。各Assertionは3つの部分、即ち満たすべき条件、及びそれをチェックすべき時点、そしてもし満たされなかった場合に採るべき行動から成っている。又Assertion自体のインテグリティ及びその更新などをサポートする。

8. コンカレンシーレベルの向上

複数のQueryが同一のデータに対し並行して処理を行なう場合、その処理内容は個々には正しくとも、即ちインテグリティを犯すものではなくとも共有データに対して並列に処理をするという事から必ずしも正しい結果は保証されない。これは一貫性の問題となる。一方データベースマシンは互いに干渉がないと判断されたQuery群に対して並列処理出来る環境を提供すべきである。(14)

i) Retrieval - Retrieval Concurrency

理論的には何ら制約はなく並列に処理して問題ない。本マシンではタプルリダクション及びドメイン選択機能をメモリ側に付加している為、必ずしも同時にQueryを処理する事が好ましいとは限らず、順次処理した方が良い場合も多い。完全に同時に処理しなくともテンポラリリレーションを生成する為ベースリレーションは早急に開放されるからである。しかし更にアクセス頻度の高いリレーションに対しては、フルタプルで送出する事によって同時処理可能な環境を生成出来る。この時メモリは常に一定数のフルタプルをくり返し送っている丈である。

ii) Retrieval - Update Concurrency

更新中のデータをRetrieval Queryが参照すると一貫性が保証されない可能性がある。しかし、この種の競合は従来のオブジェクトロックによって解決出来る。即ち検索Queryがその対象をロックする事により更新操作に先行する順に直列化し得るからである。さて、データベースマシン上では更新動作に於いてChange Setを作るのは1スキャンの検索と同程度の負荷ですみ、一般のソフトウェアデータベースの場合と比べるとインデックス等の更新が全く不要である為、かなり高速化され、それ自体のオペレーションコストは大きな問題ではない。しかし、厳重なAssertion Checkを課す様な場合や、又トランズアクション処理の場合等は当該リレーションを長時間ロックする事になり好ましくない。これらの問題に対しては、検索Queryに更新前のリレーションイメージを、一方更新中のQueryには更新後のイメージを提供する事によって並行処理する事が可能である。メモリモジュールはフルタプルに加えて当該タプルの状態を送出すればよい。又更新が正常終了し、当該リレーションにChange Setを取り込む際には検索Queryが終了しているべきであり、この取り込みに関しては、幾つかの一貫性レベルを設定する事が出来る。

iii) Update - Update Concurrency

同一リレーションに対する2つの更新要求の同時実行可能性については、従来のオブジェクトロックは適用出来ず、実在しないタプル(ファントム)に対してもロックするプレディケイトロックが要求される。更新要求が互いに干渉しない、即ち2つのプレディケイトが重なりを

持たない事が判明すると、これらの更新はマシン上で同時処理可能である。尚、本マシンではアトリビュートレベルの同時更新をサポートする事は困難であり、タプルレベルのGranularityを仮定する。マシン上で同時更新を行なう事は可能であるが、どのタプルはどの更新操作の対象になったかを維持する必要があり、処理数には上限がある。

9. むすび

本マシンの特徴として次の様な点をあげる事が出来る。

a) プロセッサ、メモリの結合を柔軟にし、データベース処理に存する潜在的並列性を充分に取り出す事が出来、それと共にプロセッサの有効利用化を図れる。

b) プロセッシングパワーを動的に割り当てる事が可能であり、きめの細い制御を実現する事が出来る。

c) 多くの独立したQueryを並行して処理する事が出来、マルチユーザの環境に適している。

d) JOIN, DIVISION, PROJECTIONなど、比較的処理負荷の重いオペレーションに対しては、メモリセル数とは無関係に多くのプロセッサを駆動する事が出来、高速化を図れる。

e) ブロードキャストシステムによってメモリコンフリクトフリーの環境を生成しており、処理速度は駆動プロセッサ数に略比例して向上する。

f) 記憶媒体として改良されたメイジャマイナ方式のバブルメモリを用い、不要データ出力をメモリチップレベルから押える事によってより一層性能の向上が期待できる。

データ操作部のアーキテクチャとしては、リングバス結合マルチプロセッサシステムを採用しており、いずれかのモジュールに障害が生じても他のモジュール群だけで再構成可能である。又回線制御部に関しては、伝送エラーや一時的同期はずれによる障害に対して回復可能な伝送制御手順を用いる事が出来る。(16)

本マシンではホストにつながるバックエンドコントローラによってQueryが受け取られるとSecurity Enforcer, Integrity Enforcer, 及び、Concurrency Controllerを通った後、Demand Createrに送られる。ここで、関係代数に変換されたQueryを既に述べたデータ操作部のコントローラに割り付け、当該処理が開始される。又メモリモジュール数は有効チャネル数より多く、処理と並行してマスメモリからのステージングを行なう事が出来る。

参考文献

1) S.W.Y.SU et al "Data Base Machines and Some Issues on DBMS Standards" Proc. AFIPS NCC P191, 1980

2) W.Frank King "Relational Data Base Systems : Where We Stand Today" Proc. IFIP 80 P369 1980

3) B.W. WAH et al "DIALOG - A distributed processor organization for database machine" Proc. AFIPS NCC P213, 1980

4) K.Oflazer et al "RAP.3 A Multi-microprocessor Cell Architecture for the RAP Data Base Machine" Proc of Int. Workshop on High Level Language Computer Architecture, P108, 1980

5) E.A.Ozkarahan "Hardware and Software Systems Research on the RAP Data Base Machine" Proc. of IE³ Int. conf. on Circuits and Computers, 1980

6) D.J.Dewitt "Query Execution in DIRECT" Proc of ACM SIGMOD 1979 P13, 1979

7) S. Uemura et al "The Design and Implementation of a Magnetic-Bubble Data Base Machine" PROC. IFIP 80, 1980

8) P. Bruce Berra "The Role of Associative Array Processors in Data Base Machine Architecture" IE³ Computer Vol.12 No3 P33, 1979

9) S. Todd, "Algorithm and Hardware for a merge-sort using multiprocessors" IBM Jof R&D vol22 NO.5 p509 1978

10) Y. Tanaka "Pipeline Searching and Sorting Modules as Components of a Data Flow Database Computer" PROC. IFIP 80, 1980

11) M. Stonebraker "MUFFIN: A Distributed Data Base Machine" Proc. of 1st Int. conf. on Distributed Computing Systems, P259, 1979

12) Hsu Chang "Magnetic Bubble Memory Technology" Electrical Engeneering and Electronics /6 Marcel Dekker 1978

13) K.P. Eswaran "Functional Specifications of Subsystem for Data Base Integrity" Proc of VLDB 1975

14) R. Bayer "On the Integrity of Data Bases and Resource Locking" Lecture Notes in Computer Science vol 39 P339 1975

15) 喜連川他 "可変構造多重処理データベースマシンの構成" 情報処理学21回全国大会 2F-4, 1980

16) 喜連川他 "多重チャネルリングバスに於けるブロードキャスト伝送制御手順" 情報処理学分散処理研究会 5-2 1980

0節で述べたマシンを含めてデータベースマシンに関する文献は次に詳しい。

17) 植村・前川著 "データベースマシン" 情報処理叢書1 情報処理学会 1980