

18

EC 81-35

Hash と Sort による 関係代数マシン

喜連川 優・鈴木 重信
田中英彦・元岡 達
(東 大)

1981年10月29日

社団法人 電子通信学会

HashとSortによる関係代数マシン

Relational Algebra Machine Based on Hash and Sort

喜連川 優

M. Kitsuregawa

田中 英彦

H. Tanaka

東京大学

University of Tokyo

鈴木 重信

S. Suzuki

元岡 達

T. Moto-oka

工学部

Faculty of Engineering

1. はじめに

データベースマシンの出発点は "Logic per Track" ⁽²⁾ にあると考えられる。これはRAPI⁽⁴⁾やCASSMに見られる如くディスクのヘッドに簡単なロジックを付加し、単純な条件検索を実現するものであり、後に記憶媒体がCCDや磁気バブルなどに置きかえられ、メモリとプロセッサのペアを基本単位とする "Cellular Logic Type DBM" ⁽¹⁾ としてとらえられる様になった。この方式は SELECTION や UPDATE など1タプル内で処理が完了する比較的負荷の軽い操作に対しては従来のマシンに比べて大きな性能向上をもたらすとともに、メインメモリとディスクとの間の von Neumann Bottleneck の解消を図り、Filter Processorとしての位置を固めたい⁽³⁾。しかしながら、JOINやPROJECTIONなど処理負荷の重いオペレーション、2つのリレーションにまたがるオペレーションに対して直接適用しようとしても、その効果は薄く、Stream Orientedな全数サーチ処理手法の適応限界が明らか⁽⁵⁾にされた。即ち、これまで開発されてきた多くのマシンは基本的には "Filter Processor" でありリレシヨナルモデルのサポートに際し、不可欠なJOINやPROJECTIONに対しては十分な性能を有しているとは言い難い。いくつかのマシン上でのJOINの処理方式を検討すると以下の如くなる。

• Cellular Logic Type DBMではJOINの処理負荷は両リレーションの Cardinality の積に比例し、それをセル数台のプロセッサにより並列

処理する事になる。従って処理時間Tは

$$T \propto \frac{N \cdot M}{n \cdot k} \quad \text{と表わされる。}$$

M, N: 2つのリレーションの Cardinality

n: プロセッサ数=セル数

k: プロセッサ1台当りの比較器の数

RAPなどがこの範囲に入る。既に我々もこのタイプのDBMに改良を加え、プロセッサとメモリ間の結合関係を1対1の固定化されたものからより柔軟なものにした "可変構造多重処理データベースマシン" の開発を行なって来た。このマシンでは上記のnが固定ではなく処理負荷に応じて動的にプロセッサが駆動される点に長がある。^(12,14)

• RELACSでは連想プロセッサ(STARAN like)を用い、上記のkが大変大きくなったとみなす事ができる。

• KUNGらの Systolic Array⁽⁶⁾ではVLSIの利用により多数のコンパレータをパイプライン的に駆動する事によりページ内ではO(1)時間でJOINが実現できるが、これはビットマップを生成するだけで実際の結合操作は伴わない。ページレベルでの制⁽⁷⁾御方式は明らかでない。

• DIRECTでもページレベルでO(nxm) (n, m: 2つのリレーションの占めるページ数)を要し、プロセッサ台数を充分に増やし max(m, n) 台用いる事でO(min(m, n))時間、即ち、小さい方のリレーションサイズに比例した時間で処理を試みている。又ページ内の処理は、ページロード、ソートによるJOIN、ページストアの3ステップからなり、汎用μPにより

実現している。⁽⁸⁾

• CASFの Hash Bit Array方式は後に再び触れるが、JOIN候補の篩い落としには効果があるものの、あくまで前処理であり、実際の結合操作はホストで行う必要がある。

以上の如く、現実的な大きさのリレーションに関して関係代数演算の $O(N)$ 化は充分に実現されているとは言えない。尚、最近(9)等国内マシンですぐれた方向も存在する。

本稿では Hash と Sort を用いた $O(N)$ の関係代数マシンについて、その処理方式、基本構成について述べる事とする。

2. Hash の適用

Hash は一般データベースでのアクセス法としては、しばしば用いられ、ロードファクタが小さい時、略1回で目的とするレコードを得ることが出来、最も高速な手法と考えられている。この様なメモリ媒体上のスタティックなデータ管理手法としての利用とは別に、データベースマシンでは以下の2つの適用技法が考えられる。⁽⁸⁾

1. CASFに見られる篩い落とし技法
2. クラスタリング手法

JOINの処理手法について考えると、1.ではバケットの数を多くして結合アトリビュートに Hash を施し、両リレーション間でのシノニムと結合可能性のあるタプルとして以後の処理対

象とする手法である。これにより JOIN のとれる可能性のないタプルは篩い落とされ、両リレーションのサイズは減少するため、処理負荷が Cardinality の積で効く単純な処理方式では、その効果は大きいと考えられる。しかしながら、これはあくまでも前処理であり、Explicit Join に於けるタプルの結合操作は後にホストで実行する必要がある。

一方2.の手法は対象とするリレーションのタプル数自体の減少を目的とするのではなく、実際の結合処理に於ける負荷を小さくしようとする手法である。即ち、JOIN は最も単純に処理すれば $O(N \cdot M)$ 時間が必要であるが、JOIN アトリビュートに関して両リレーションに Hash を施し、全体を S 個のバケットに分割したとすると、異なる Hash ID をもつバケット同志は JOIN がとれる可能性がない為、同一バケット ID 同志の処理に限定できる事になる。すなわち

$$N = \sum_{i=1}^S n_i, \quad M = \sum_{i=1}^S m_i$$

とすると、所要時間 T は

$$T \propto \sum_{i=1}^S n_i \cdot m_i$$

ですむ事になる。図1に示される如く、Hash のクラスタリング効果により、大きな処理負荷の減少がもたらされる事がわかる。我々はこの2の手法を基本に関係代数演算の高速化を試みた。尚、1., 2.の手法は互いに独立であり、2.で実

際の処理をほどこす前に、Joinability Filter Processor として1.を適用することによって、作業領域を大きく節約することが出来る。この様に、1., 2.両手法の融合も可能である。

又、以上は JOIN に関して述べたが PROJECTION の処理については、1.の手法を利用する場合、シノニムを全て重複タプルと見做す事は必ずしも出来ず、少々問題が残るのに対し、2.の手法では、異なるバケット間での重複は有り得ず、JOIN と全く同様に適用することが出来る。

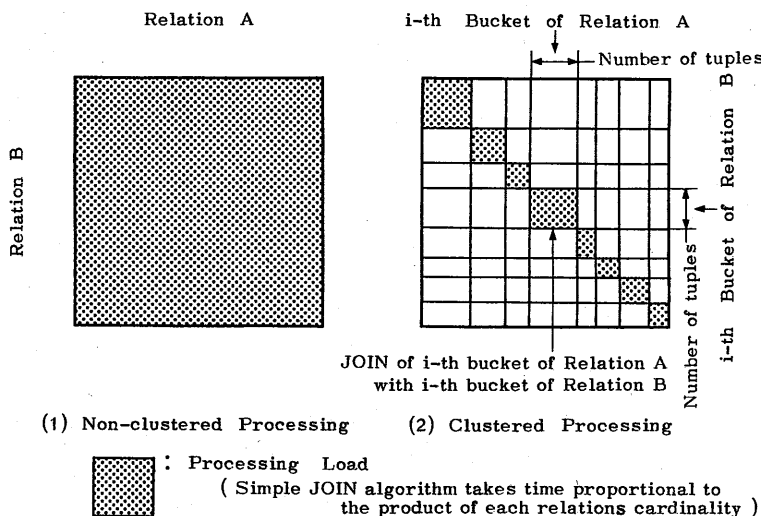


Fig. 1. Clustering Effects By Hashing In JOIN Operation

3. Hash技法の並列処理化とその問題点

前節ではHashを適用し、リレーションを互いに独立なバケットに分割する事により、JOIN、PROJECTIONなどに対して処理負荷の大きな減少が実現されるという事実が示された。ここではこの方式をいかにデータベースマシンとして実現するかについて考える事にする。一台のプロセッサでシリアル処理する事も可能であるが多数のプロセッサを用い各バケットを並列処理することにより一層高速化が期待出来る。以下、Hashをもとにした関係代数マシンを構成するにあたっての問題点をいくつか列挙し、次節でそれらに対して我々の採った方策を説明する。⁽⁵⁾

1) バンクパラレルリズムの反映

処理の対象とするリレーションが複数のメモリバンクに格納されている場合には、それが1つのバンクに入れられている場合に比べバンク数倍の高速化を図る。

データベース処理ではデータが主体であり、その流れにそって処理が施される。この際、たとえ $O(N)$ の処理負荷であろうとも、一本の長いデータ流とするのではなく、それを短い複数本のデータ流とする事により、バンク数だけの高速化が可能である事が望ましい。即ち、 $O(N)$ の N をリレーションのCardinalityではなくページサイズにする事が好ましい。これにより作業ページ空間に入りきるリレーションの操作に関しては、その大きさに依存しない高速関係代数処理が可能となる。

2) バケット内 $O(m)$ 処理

Hashを用い、 S つのリレーションを互いに独立な S 個のバケットに分割する事により、 S^2 個ではなく S 個のバケット処理に軽減出来た。即ち、バケットレベルで $O(S)$ の処理負荷を実現したが、更に1つのバケット内の処理も、そのサイズを m とすると $O(m)$ 時間で処理を完了させる事が望まれる。これにより全体として高速な関係代数処理が実現できることになる。

3) プロセッサへのバケット分配技法

バケットは互いに独立であり、従って、基本的には1つのバケットを1つのプロセッサに割り当て、各々並列に処理する事で高速化が期待出来る。この際プロセッサが当該バケットを構成するデータと効率よく収集出来る様にする必要がある。又、或程度以上の大きさをもつリレーションの処理と

考えると、全てのバケットを並列に処理する事は困難であり、駆動プロセッサ台数を越えるバケット処理に対しては、シリアルに行なう必要がある。この時 $O(m)$ を実現するためには、複数台のメモリバンクはデータ流の乱れを最小限とし、円滑なバケット切替と実現する必要がある。

4) 分散不均一性への対処

Hashを利用する場合、その性質上、Hash後のデータの分布は必ずしも均一にならず、バケットの大きさに偏りが生じてしまう。更に、プロセッササイズに収まらず、バケットオーバーフローを生ずる事もある。この分布不均一性はHashを採用する場合には避け難い特長であり、一方Hardware Architectureから見れば、処理対象の大きさの変動は、リソースの使用効率及び性能と下げ要因となる従って、この問題を効率よく処理し、又オーバーフローの如き例外事象にも対処できる必要がある。

4. 実装方式

前節で述べた問題点に対し、我々の採った手法について解説する。

1) バンクパラレルリズムに関して

対象リレーションが、後に m 台のメモリバンク(メモリモジュール or ディスクモジュール)に格納されているとすると、 m 本のデータ流に対して $2 \cdot m$ 台のプロセッサを駆動する事によりモジュール間のパイプラインを形成し、各データ流を並列に処理する。これにより m 倍の処理の高速化即ちバンクパラレルリズムを実現する。(6節2)参照)

2) バケット内 $O(m)$ 処理について

プロセッサは割り付けられたバケットの処理を $O(m)$ 時間で完了する事が望まれる。一方、関係代数演算ではJOIN或はPROJECTION外リビュートに関してソートされていると、 $O(m)$ 時間で処理出来る事は明らかである。そこで、ここではプロセッサにソート機能をもたせる事とした。従って問題は、 $O(m)$ のバケット処理から $O(m)$ のソータに還元された事になる。我々は既にソータの開発と行なってきているが、これは入力データ流に遅れることなく同期したソータを行なう $O(N)$ (N : レコード数)ソータである。アルゴリズム的にはマージソータを基本とし $O(N \log N)$ のソータ負荷を一次元状に結合された $\log N$ 台のプロセッシングユニットを用い、パイプライン処理

する事により、 $O(N)$ 時間のソートを実現している。本ソートをプロセッサに組み込む事により、バケット内処理と $O(N)$ 時間で完了出来る。

3) バケット分配技法に関して

一般に、駆動プロセッサ台数に比べて、Hashによって生成されたバケットの数の方が多いため、並列処理と上まわる分については、バケットシリアルに処理を進めて行く必要がある。一方、全体として $O(N)$ 時間で処理と完了する事が目的であり、この為にはメモリデバイスがバケットシリアルにデータを途切れる事なく出力する必要がある。一般に、1つのバケットを構成するタプルがメモリデバイス内で物理的に連続して配置されているとは期待出来ない為、何らかの工夫が必要になる。即ち、メモリ内にランダムに存在するバケット内のデータを連続して出力する機能が要求される。ディスクによってこれを実現する事は困難であり、逆にRAMを用いれば明らかに可能であるが、現実のリレーション全体をステージングする場合フィルタープロセッサを連立としてもかなり大容量となり、メモリ階層上、ディスクとRAMの中間領域に存するデバイスを用いる事がより自然である。我々はこれらの条件を満たすメモリ媒体として磁気バブルメモリを採用した。バブルは現在、容量的には4Mbit/chipが可能であり、又、コンティギュアディスクを用いる事により、一層の進歩が期待出来る。価格的には、製造工程がRAMと比べて簡素化されている為市場開拓により将来かなり低いコストが見込まれる。更にその不揮発性はデータベースサポートを実現する上では有効な要因と言える。我々は改良Major/minor方式のバブルチップを利用する事により効率よいバケットシリアル出力を可能とし、この問題を解決した。

4) 分散不均一性に関して

Hash関数によるクラスタリングの問題点として次の又点をあげる事が出来る。第一は従来同様の問題であり、Hashによって生成されたバケットサイズに関して、その偏りが生じてしまい必ずしも均一にならないという事実である。但し、我々の場合にはこれが直接メモリ使用効率に反映される事はない。第二は、本マシンではHashを施すデータ流が1本ではなく、1)で述べた如くバンクパラレルリズムを実現する為複数のデータ流に対して同時にHashが施されると

いう点である。この2つの問題が重なり、より複雑になっている。我々はこの問題に対してHashの段階でモジュール間バケット平坦化処理を行ないマルチストリーム間の偏りをなくし、更に、バケットサイズの偏りに対しては、その後バケットサイズチューニングを行ないバケットを複数集めて、プロセッサのメモリ容量(プロセッササイズ)に合わせる様に統合し、プロセッサの利用効率を高めるとともに、モジュール間パイプラインの擾乱を抑えている。バケット数を多くしてHashを施す為、オーバーフローは殆ど生じないが、(一般のHashの様にlocationとHash値と対応させる方式ではなく、クラスタリングを目的としている為、予め定まったバケットサイズが存在するわけではない。メモリは単に当該レコードデータにHash値と連結して記憶するに過ぎない。ここでのオーバーフローとは、同一Hash値を有するタプルがプロセッサのメモリ容量を越えた場合をいう。)この場合には、複数のプロセッシングモジュールがリングバス上の1つのチャネルを介し、動的に結合して、協調処理を行なう事により解決できる。

3. システム構成

本DBMの構成を図2に示す。データ操作部は図3に表わされる如く、4つのモジュール、プロセッシングモジュール(PM)、メモリモジュール(MM)、ディスクモジュール(DM)、コントロールモジュール(CM)、及び、これら多数のモジュールを結合する時分割多重チャネル方式のリングバスからなる。図中、下のステージングリングを用いてDMが

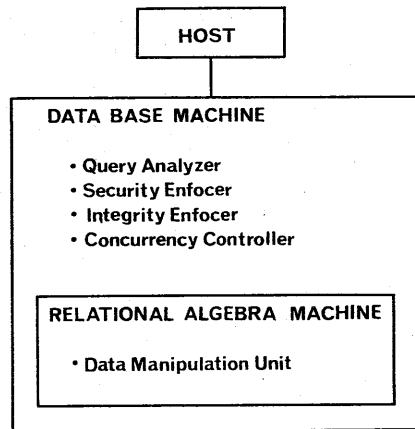


Fig.2 Abstract View Of Data Base Machine

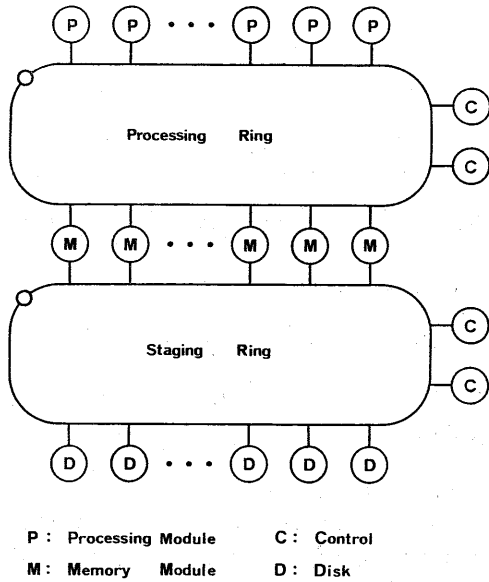


Fig. 3. Global Architecture Of Data Manipulation Unit

らMMにリレーションデータをステージングし、その後上のプロセッシングリングを用いて、PMがMM上のデータに処理を施す。CMは両方のリングに存在し、全体の実行制御を司る。又これらの上位には Query Analyzer, Security Enforcer, Integrity Enforcer等が存在し、CMには、関係代数演算Treeに展開された Queryが渡される。

1) プロセッシングモジュール

本モジュールは Control Unit, Sorting Unit, Tuple Manipulation Unit, Hash Unit, Ring Bus Interface Unitなどからなる。(図4)

- Control UnitはCMからのコマンドを解釈

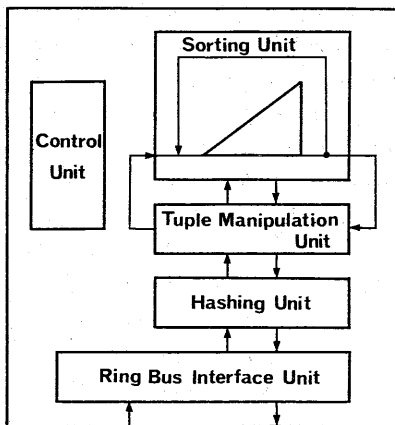


Fig. 4 Processing Module Organization

し、ソース及びリザルトリレーションに対応するポインタ番号、Hashアトリビュートや Qualification Predicate に関する情報の管理を行なう。又PMの全体的な振舞いを制御すると共に、各unitの初期化、駆動を司る。

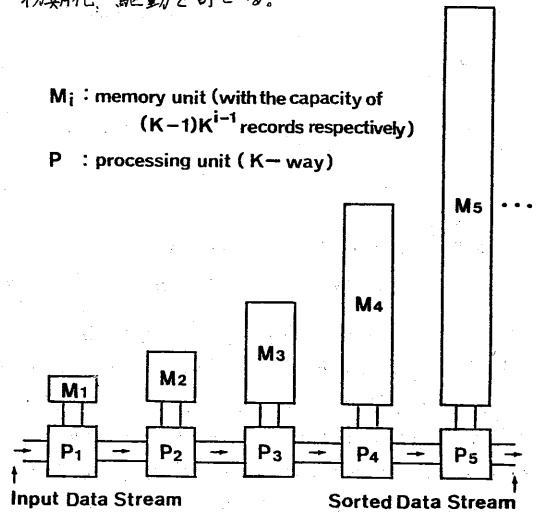


Fig. 5 Global Architecture Of Stream Driven Sorter

Sorting Unitは入力データ流に遅れる事なく同期したソートを行ない、処理と入力をオーバーラップさせる事によって、最後のレコード入力後、僅かの遅れと共にソートされたレコード出力が得られる構成となっている。ソートアルゴリズムはマージソートのパイプライン化である。今、 $N (=K^L)$ のレコードのソートを行なうものとする。K-way マージを行なうプロセッサを i 台用意し、 i 番目のプロセッサには $(K-1) \times K^{i-1}$ レコード分のメモリを付加する。K=2とした時の全体像を図5に示す。レコード流は、シリアルに左端のプロセッサに入力される。処理は基本的にはマージであり、 i 番目のプロセッサは $i-1$ 番目のプロセッサから送られて来る K^{i-1} のレコードからなるソートされたストリングをK本マージして、 K^i レコードからなる1本の長いストリングを生成し、 $i+1$ 番目のプロセッサへ出力する。図6の如く、各プロセッサはマージすべきK(K=2)本のストリングに関して i 番目のストリングの最初のレコードが到着した時点でマージを開始できる。図から明らかになく、 $\log_k N$ 台のプロセッサにより $O(N)$ ソータが実現できる事がわかる。処理の詳細については、文献[3,10]を参照されたい。

- Tuple Manipulation UnitはSorting Unitから

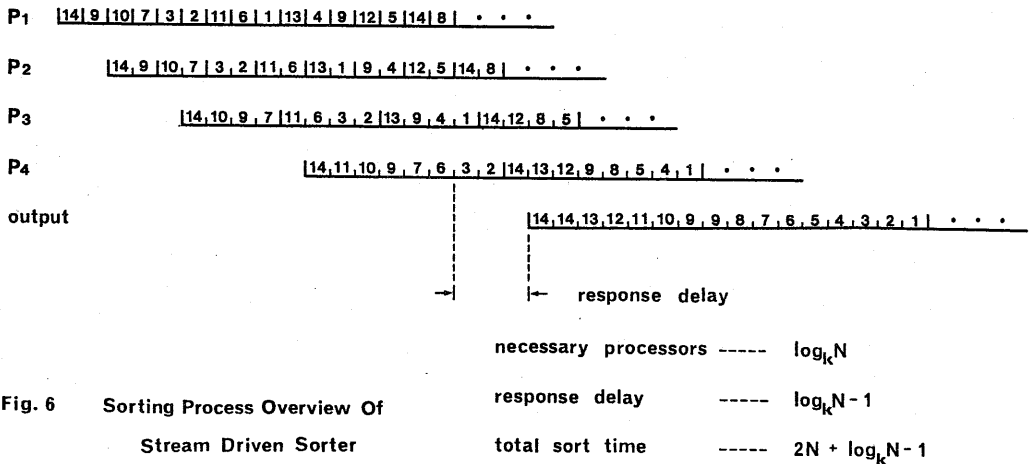


Fig. 6 Sorting Process Overview Of Stream Driven Sorter

のソートされたデータ流に対して関係代数処理を施す。又 JOIN 後の 2 つのリレーションに及ぶ RESTRICTION や複雑な Predicate はここで評価される。更に不要アトリビュートの除去やアトリビュートの並びかえなどタプルの整形処理も行なわれる。

- Hashing Unit は Tuple Manipulation Unit より生成された結果タプルを入力し、次のオペレーションに関するアトリビュートに対して Hash を施す。結果タプルは Hash 値を付加し出力される。

- Ring Bus Interface Unit は リングバス上のチャネルの検出、及びデータの入出カー切を受け持つ。チャネル番号、及び Hash ID のマッチングにより、タプルの取り込み制御を行なう。

2) メモリモジュール⁽¹²⁾

本モジュールは Control Unit, Magnetic Bubble Memory, Mark Bit RAM, Bubble Control Unit, Tuple Control Unit, Bucket Management Unit, 及び Ring Bus

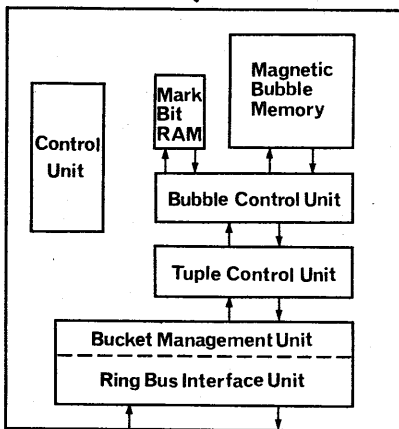


Fig. 7 Memory Module Organization

Interface Unit などからなる。(図 7)

- Control Unit は CM からのコマンドを解釈し、ステージング左^(読書)にはステージングリング上の有効チャネルリストを管理し、DM とのリンクを表現するとともに、プロセッシング左^(読書)にはプロセッシングリング上の割り付けられたチャネルを用い、PM との動的なリンクを制御するなど、MM の全体的な振舞いを制御する。又モジュール内各 unit の初期化、駆動を可とする。

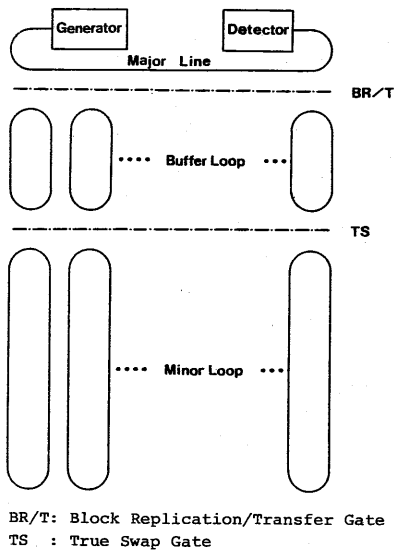
- Magnetic Bubble Memory
バブル、特に M/m 方式のバブルがディスク等のシリアルループタイプメモリに対して優れている点は「1 ビットタイムによる不要データの読みとばし機能」

にあり、これにより次の 2 つの特長が生ずる。

1. アクセスタイムが短い。
2. 実効データ転送レートが高い。

従来のプログラミング環境下で用いる場合にはランダムなアクセス要求がなされる為、とりわけ 1. の特長が重要視されているが、データベース応用を考えると、メモリに対するアクセス要求はアドレス指定による 1 ケー1 ケのレコードではなく、「ある条件を満たすレコードの集合」の如く、

Set Oriented な要求となる。又この際、データの出力順序は問題でない場合が多い。従って、一回毎のアクセスタイムよりも寧ろ、全体としての実効データ転送レートが重要になる。我々は従来の M/m 方式のバブルに対し更に改良を加え、より一層 2. の効果を高める方式を採用した。チップデザインに関する詳細は別の機会に譲るとして、簡単の為最も単純な改良を図 8 に示す。即ち、メインラインとマイナーループの間には小さなバッフルループ



BR/T: Block Replication/Transfer Gate
TS : True Swap Gate

Fig. 8 Modified Major/minor Bubble Chip Organization

を設け、必要なデータをマイナーループからバッファループへ動的に移す事により、レコード間のギャップタイムを殆どなくすることが可能となり、高い実効データ転送レートを実現出来る。一例として、マイナーループ長が 2048 マイナーループ数が 64 のバブルチップに対し、バッファループ長をパラメタとしシミュレーションにより求めたチップ性能を図9に示す。ここで横軸は、ロードファクタ、即ち、出力すべきレコード数の全レコード数に対する割合であり、縦軸は1レコード当りのレコード間ギャップタイムの割合を示す。この図から、わずかの工夫で次に述べるマ

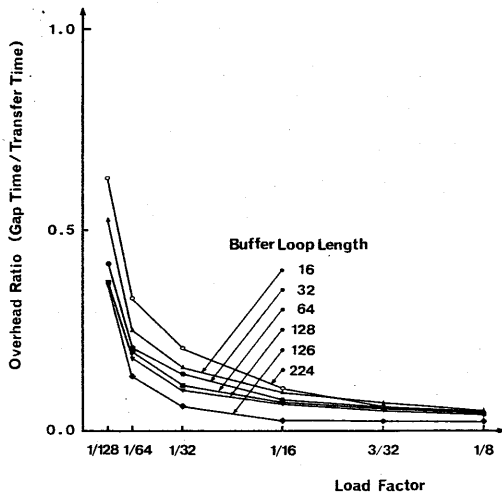


Fig. 9 Overhead Ratio Of Modified Major/minor Bubble Chip With Buffer Loops

ークビットを用いた連想読出し機能により、高い実効転送レートを実現出来る事がわかる。

- Mark Bit RAM / Bubble Control Unit
マークビットはバブルの回転と同期しており、マイナーループ及びバッファループレコードに対応して1つのエンタリーが設けられている。ステージフェイズ又はプロセッシングフェイズのリザルトリレーションでは Mark Bit RAM に当該タプルの Hash 値 (バケット ID) が記録される。そして、プロセッシングフェイズのソースリレーションでは、バケットシリアルにデータを入力するが、この際、Mark Bit を参照しながらバブルを制御する。Mark Bit は処理に応じ、随時適当な意味を持たせる事が出来る。Mark Bit の操作並びにバブルの制御は全て Bubble Control Unit (BCU) が行なう。

- Tuple Control Unit は BCU からのタプルデータの整形及び BCU の駆動、即ち、R/W モードの設定読み出すべきタプルの Mark Bit 情報の設定、バケット切替制御などを行なう。バケットシリアルなタプル送込に際し、バケット切替の契機を生成するが、この契機はループによって異なる。マイナーループ側はバッファループ側より速く、マイナーループ内に存する当該バケットの全てのタプルをバッファループに吸い上げた後は次のバケット処理に移ってよい。尚、図9のデータは最初のバケットに対するものであり、バケットの連続出力を行なうと後続のバケットに対しては、更にギャップタイムが少くなる。Tuple Control Unit は、以上の如く、BCU に比べ、より上位レベルの制御を行なう。

- Bucket Management Unit はバケットを構成するタプルの管理を行なう。DM又はPMからのデータ入力時には、バケット毎にいくつのタプルを入力したかを管理しており、モジュール間バケット平坦化処理を行なう。又、入力完了後、これをもとにバケットサイズチューニングを施す。

3) ディスクモジュール (図10)

本モジュールは Control Unit, Disk Unit, Filter Processor, Hashing Unit, Ring Bus Interface Unit がある。

- Control Unit は CM からのコマンドを解釈し、接続チャネル番号、Hash アトリビュート、Qualification Predicate などの情報と管理し、DM の全体的な振舞いを制御するとともに各部の適切な初期化、駆動を司る。又、ディスク内のページ分割されたリレーションの管理、及びディレクトリ管理を行なう。

- Disk Unit

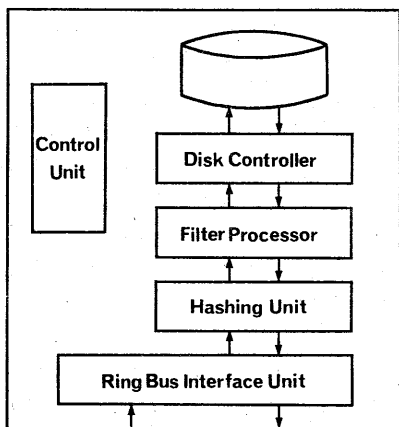


Fig.10 Disk Module Organization

本マシンでは、2次記憶メモリとして、ディスクを採用し、これにより大容量データベースのサポートを可能にしている。MSSを接続する事により、更に大きな容量を維持する事も可能である。

- Filter ProcessorはDisk Unitからのデータ流に対し、Selection等の簡単な条件検索が施され、条件を満たすデータのみをHashing Unitに出力する。

4) コントロールモジュール

Query実行に際する各モジュールの振舞は次節で詳しく述べるが、CMは関係代数処理に先立ち、PM、MM、DM等に適切なコマンドを発行し、駆動する。これらのコマンドは、各モジュールに属するControl Unitが解釈・実行する。又、当該Queryに対するResource Scheduling即ち、モジュール台数、及びチャネルの割付け等と共にQueryの実行管理、上位システムとのやり取り等を行なう。

6. システム動作

本節では、関係代数演算子の処理過程を説明する事により、前節までに述べた基本機能部位の位置付け及び処理方式等をより明確にする。

リレーシジョンの処理は大きく次の2つに分ける事が出来る。

1. ステージングフェイズ…… ディスク(DM)に入っているデータの必要な部分をバブルメモリの空間(MM)へ移す段階
2. プロセッシングフェイズ…… MMに移されたデータに対してPMが処理を施す段階

1) ステージングフェイズ

リレーシジョンは複数のDMにまたがってページ単位に格納されており、当該リレーシジョンのMMへのステージングはページパラレルに行なわれる。DMでは、Filter Processorにより、Selection, Restriction等の簡単な検索が行なわれ、条件を満たすタプルのみがステージングされる。これは従来のRAPやCAFSなどで実現されている機能と同種のものである。こうして選択されたタプルはHashing Unitに送られ、最初のオペレーション(JOIN, PROJECTION, DIVISION etc.)に関するアトリビュートに対してHashが施され、当該タプルにバケットIDを付加して、ステージングリング上に送られる。

一方MMはDMから送られてきたデータをバブルメモリに入力するが、この際当該バケットを構成するタプルがMM群にわたって出来る均等に分配される様、Bucket Management Unitによりモジュール間バケット平担化処理がなされる。MMは、当該バケットに対し、いくつのタプルデータを入力したかを常に管理している。これは、プロセッシングフェイズのバケット出力時にも利用される。又、タプルの入力時には、タプルデータはバブルに貯えられ、バケットIDはバブルと同期したMark Bit RAM上に格納される。データ入力時にはバブル内のデータクラスリングを行なっているゆとりはない為、チップ内でランダムに分散される事になる。本フェイズでは以上の如く処理対象のリレーシジョンの中で必要な部分に対して、Hashを施しつつバブル空間上にステージングする。

2) プロセッシングフェイズ

ステージングフェイズ時にステージングリングに結合していたメモリモジュールはプロセッシングリングに結合し実際の関係代数処理に取りかかる。MMはMark Bitに記録されているバケットIDをもとに、バケットシリアルにタプルデータを出力する。この際、バブルに付加されたバツアループの効果により、略、連続した効率の良いデータ出力を実現する。同時に、バケットをプロセッササイズに結合するバケットサイズチューニングがモジュール間にわたって行なわれる。小さなバケットはプロセッシングモジュールの利用効率が低く、又、バブルメモリに於けるバケット切替オーバーヘッドも問題になる為、他のバケットと統合される。バケット管理により、当該バケットのタプルを全て出力すると次の

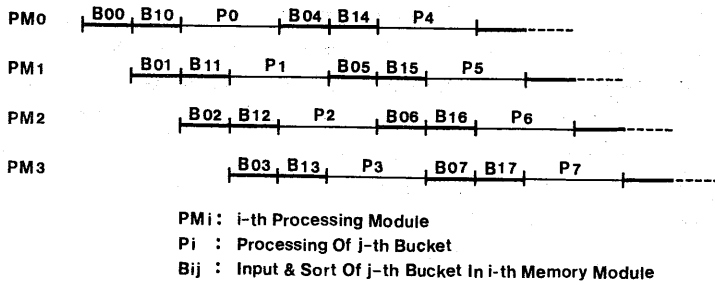


Fig.11 Pipeline Overview Of Bucket Processing

バケット出力にとりかかる。

PMはMMから出力されるバケットに対し関係代数処理を施す。あるバケットに割り当てられたPMは当該バケットを構成するタプルデータを複数のMMから順次収集する。即ち、まずMM₀にリンクし、MM₀の持つバケットiのデータを全て取り込むと、次にMM₁にリンクし・・・この様にしてPMは各MMを訪ねてバケットデータを取り込むが、その際テーブルをO(N)ソータに供給する事によって入力に遅れる事なくソーティングを行なう。全てのMMとリンクし終わった後、ソータは当該バケットを構成するタプルデータをソート順に出力可能となり、ソートされたデータ流に対し、Tuple Manipulation Unitが関係代数処理を施す。この様に、PMの処理はMMからのデータ収集、及びそれと重畳されたソートフェイズと収集後ソートデータ流に対する関係代数処理のフェイズの二つから成り立っている。これらのフェイズはバケットサイズをnとすると各々O(n)時間で終了する為、全体として略、nに比例した時間が必要となる。依って、m台のMMに対しm台のPMを駆動する事によりパイプライン処理が可能となる。MM2台、PM4台の時の理想的な処理の様子を図11に示す。実際にはバケットサイズを完全に平坦化出来るわけではなく、少々の偏りは残存するし、又直積バッファオーバーフロー

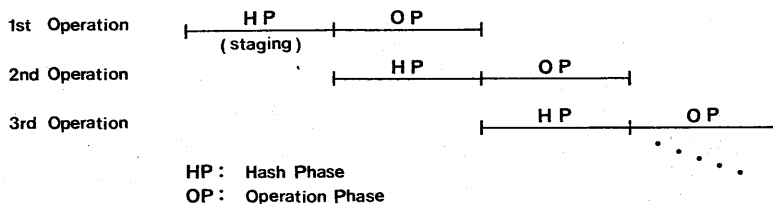


Fig.12 Operator Level Pipeline Processing (Overlapped Processing Of Hash Phase & Operation Phase)

による処理の遅れの可能性もある為、パイプラインは幾分乱れたものになると考えられる。処理を終えたPMは再び次のバケット処理に取りかかり、全てのバケットと多数のPMにより順次パイプライン的に処理して行く。尚、処理の途中で結果テーブルが生成されると、現在処理中のオペレーションの次のオペレーションのアトリビュートに関してHashを施し、リザルトリレーションを格納するMMに送出する。

以上の如く「ステージング」フェイズで「ベースリレーション」をMM上に移した後、「プロセッシング」フェイズで処理を施す。一般に問合せ処理は複数オペレーションのシーケンスで表現されるが、処理結果を「テンポラリリレーション」として、順次MM上に生成する事により、以後「プロセッシング」フェイズを繰返しつつ処理を進めて行く。この際次節で述べるオペレータレベルのパイプラインが形成される。

7. オペレータレベルパイプライン

本マシンでは複数のデータ流に対するパラレルプロセッシングに加え、次の様な種々のレベルでのパイプラインプロセッシングが行なわれている。

- 1 ... ソータの $\log N$ プロセッシングユニットによるマージソートのパイプライン処理
- 2 ... 複数のPMによるバケットパイプライン処理
- 3 ... オペレータレベルパイプライン

1から3の順序でよりハイレベルになっており、ここでは特に3.について述べる。

Hashによるクラスタリング効果を利用する場合、中途半端な状態から処理を開始する事は難しく、あらかじめ全体をいくつかのクラスタに完全に分離する必要がある。

従って1つのオペレーションの処理はHashによる前処理(Hash Phase)と、その後の関係代数についての真処理(Operation Phase)の二つに分離される。しかし、これ等を単にシリアルに実行するのではなく、以下に述べる如く処理を互いに重

疊化させ性能を向上させる事が出来る。

プロセッシングフェイズでは、PMはMMからのバケットに対し、関係代数処理を施すが、この際結果タプルの生成時に次のオペレーションに関するアトリビュートについて Hash を施し、リザルトリレーションを構成するMM群へ送出する。従って、当該オペレーションの関係代数処理と次オペレーションに対する Hash の処理とが重畳されている事になり、事実上、Hash に関する時間的オーバーヘッドは消失する。図12に示される如く Operation Phase を連続させられる事がわかる。又、一番最初は、ステージ動作に対して Hash 処理が重畳化されていると見做せる。この様に本マシンでは、多数の JOIN, PROJECTION を含む Query に対し、オペレータレベルのパイプラインにより、Hash のオーバーヘッドをなくし、高速処理を可能としている。

8. むすび

Hash と Sort による $O(n)$ 関係代数マシンについて、その基本構成を述べた。本マシンにより、実際的な大きさのリレーションの JOIN, PROJECTION 等と $O(n)$ (n : メモリページサイズ) 時間で実現できる事が示された。

Filter Processor は従来通りデータ源でその絞り込みを行なう上で重要な役割を果たすが、バブルメモリ・フィルタプロセッサペアの単なる多重化 (Cellular Type) だけでは集合演算の高速化は困難と考えられる。それに対し、ここではメモリバンクに固定されたデータ流を操作するのではなく、メモリバンクから別のバンクへデータを流す過程で、PM MM の各種機能により、関係代数演算等の処理を施すとともにデータ流の変換を行なう方式となっている。以下本マシンの特長を簡単にまとめる。

1. JOIN, PROJECTION など処理負荷の重いオペレーションを $O(n)$ 時間で処理する事が出来る。
2. バンクパラレルリズムの効果により、バブル空間 (MMs) に入るリレーションの操作については処理時間は大きさに依存せず、略、メモリページサイズで済む一定時間内に処理出来る。
3. バブルメモリの採用により大きな作業空間を提供するとともに、そのチップの改良により効率のよいバケットシリアル出力機能をもたせている。
4. $O(n)$ ソートによりバケット内高速処理を可能にしている。
5. テイストを 2 次記憶として採用することにより、大容量データベースのサポートを可能にしている。
6. Hash 処理と関係代数処理の重畳効果によ

り、Hash の時間的オーバーヘッドをなくしている。尚、バケット平坦化、バケットサイズ調整、オーバーフロー処理、パイプラインへの影響、リングバス上での手順など、バケット操作に関する諸問題の詳細は別の機会に発表を譲る。又バブルチップの構成に関する検討も別稿に譲るが、図8に示される程度の改良は実現可能である。⁽¹⁾

◀ 謝 辞 ▶

磁気バブルメモリに関して、日頃御世話になっている 藤原正三氏、高橋恒介氏 (NEC) に感謝致します。

◀ 参 考 文 献 ▶

1. S.W.Y.Su: Cellular Logic Devices, IE³ COMPUTER vol.12, No.3, pp.11-25, March, 1979
2. S.A.Schuster et al: RAP-An Associative Processor for Data Base Management, Proc. AFIPS NCC, Vol.44, 1975
3. E.A.Ozkarahan et al: Performance Evaluation of a Relational Associative Processor, ACM TODS Vol.2, No.2, 1977
4. S.W.Y.Su: CASSM: a Cellular System for Very Large Data bases, Proc. Int. Conf. on VLDB, 1975
5. E.J.Oliver et al: RELACS-A Relational Associative Computer System, 5th Workshop on Computer Architecture for Numerical Processing, 1980
6. H.T.Kung et al: Systolic (VLSI) Arrays for Relational Data Base Operation Proc. ACM SIGMOD, 1980
7. D.J.DeNitt: A Multi-processor Organization for Supporting Relational Database Management Systems, IE³ Trans on Comput, Vol. C-25, No.6, 1977
8. E. Babb: Implementing a Relational Data Base by means of Specialized Hardware, ACM TODS vol.4, No.1, 1979
9. Y. Tanaka et al: Pipelined Searching and Sorting Modules as Components of a Data Flow Data Base Computer, Proc. IFIP 80, 1980
10. Hsu-Chang: Magnetic Bubble Memory Technology, Electrical Engineering and Electronics / 6 Marcel Dekker, 1978
11. H. Kohara et al: True Swap Gate Design for on Chip Chassis Organization Megabit Bubble Memory, INTERMAG 81
12. 喜連川他 "可変構造多重処理データベースマシンの構成" 信学技報, EC 80-51, 1980
13. 喜連川他 "可変構造多重処理データベースマシンにおけるソートモジュール" 信学技報, EC 81-15, 1981
14. 喜連川他 "多重バブルバスに於けるポトキャスト伝送制御手順" 情報処理分散処理研究会 5-2, 1980
15. 喜連川他 "可変構造多重処理データベースマシンに於ける Hash の適用" 情報処理第3回全国大会 4F-5, 1981
16. 鈴木喜連川他 " に於ける問合せ処理試" 同 4F-6
17. 伏見喜連川他 " に於けるソートユニット" 同 4F-7
18. 雄城喜連川他 " に於けるバブルチップ" 同 4F-8