

ハードウェア・ソートによるデータベース 基本演算の高速化について

BASIC OPERATIONS OF THE DATA BASE BASED ON THE HARDWARE SORTING

有澤 博

土肥 康孝

Hiroshi ARISAWA

Yasunori DOHI

横浜国立大学工学部

Yokohama National University

1. まえがき.

現在, 広汎な分野で使われるようになった計算機システムの利用形態は, 大きく分けて,
① 単発的に発生する数値計算的な仕事を処理,
② 外部システムの制御と直接データ処理,
③ 大規模データを恒常的に管理し, 問合せ,
更新に応じる, の3種に分類できる.

今後, ②のリアルタイム制御と, ③のデータベース・マネジメントは電子技術の飛躍的發展と対応して, ますます多くの分野に進出するであろう. ここで, ②はハードウェアも含めた総合的なシステムとしてとらえられることが多いのに対し, ③では多様な実務上の要求に対して, 長い間かなり現実的なレベルで, ソフトウェア的な対応がとられていたにすぎなかった. 最近になって, 個別的なシステムを統合化し, 新しい理論モデルを形成しようとする試みが活発になり, その一方で, モデルとその基本理念に即したハードウェアで効率よく実現させようという試み——データベース・マシンが考えられる^{1)~3)}. データベース・マシンを特徴づけるものは, ④ハードウェアを意識した記憶モデルと, ②データ集合演算の並列処理である. しかし今までのところ, データベース・マシンに必要な機能と, それを実現させるためのアーキテクチャとの関係について, 上記2点のもとに十分検討されているとは言えない.

本報告では, まず2章でこれらの検討と問題点の整理を行う. つづいて3章で有用な機能のひとつであるハードウェア・ソートについて,

特に並列性の点から議論する. 4章では, 実際にデータベースの基本演算子といくつかとり上げ, ハードウェア・ソートによる演算の実現法を紹介する.

2. データベース・マシンのアーキテクチャと問題点.

本章ではデータベース・マシンのアーキテクチャと, それを実際に構成する上で特に考慮しなければならない点について, いくつかの側面から検討する.

2.1. データベース・マシンの位置づけ

データベース管理システム(DBMS)が担当する作業は, ①データ操作言語(DML)で書かれた指令を, 現実のデータベースに対するデータ処理の手順に変換する部分と, ②データ処理をハードウェアの特性や, システム資源の利用状況を考慮して, 最適効率で実行する部分とに分けられる. 本稿では②の機能を, そのために設計された特別のハードウェアで, 効率よく実行させるものをデータベース・マシンとよぶ^{注1)}.

まず, ②のデータ処理の手順を図2.1のように一般化して考えよう. ある情報の検索を行う

注1) ①の機能を中心に考え, ミニコン付きのディスクのような形のものもデータベース・マシンとよぶことがある³⁾.

場合、はじめに、すべての情報を含むマスター・スペースの必要なファイルからデータ集合ととり出し(1)、システム・バッファ上で検索の対象になりうるデータを抽出して集合と小さくしたり、集合と集合と結びつけたりする(2)。そして結果を利用者のワーク・スペースに送り(3)、最終的なデータ操作を行って、出力する(4)。更新の場合は、はばこの反対の経路をとる(1, 2, 5, 6, 7)。

さて、データベース・マシンを構成するとき特に重要な分水嶺になるのは、マスター・スペースをデータベース・マシン内にもつかどうかである。いま、マシン内におくものを便宜的に入出力装置型、外部にある汎用計算機(GPC)のディスクなどを利用するものをチャネル型とよぶことにする。入出力装置型のデータベース・マシンは、メモリ自体をデータベース処理向きに特殊な機能をもたせることもでき、システム・バッファと機能要素を多数用いて並列処理も行いやすい。たとえばトロント大学のRAP⁴⁾は、メモリはディスクで、各トラック毎にセルとヘッドがある。メモリとしては磁気バブルやCCDの利用も考えられ、新しい機能メモリの提案もされている⁵⁾。しかし言うまでもなく、このようなメモリは主記憶要素子に比べ安いものの、従来のディスクに比べ高価であり、データベース本体を特殊メモリ上におくのは、将来の技術動向を考えてもなお疑問であろう。

チャネル型のデータベース・マシンのブロック図を図2.2に示した。この場合は、どうしても大量のデータ転送が必要になり、転送速度と、必要なデータのみを通過させるフィルタ的な機能が重要な問題になるであろう。チャネル型のデータベース・マシンの構想は、今のところ筆者らが考えているもの^{6), 7)}の他はないが、データベース・マシンを実現性という点から考える限り、何らかの形でチャネル型の形式をとらざるを得ないであろう。(RAPでも、チャネル型マシンとして考えようとする論文⁸⁾がでている。)

2.2. 並列処理のアーキテクチャ

データベースの処理が、本質的に集合に対する、あるいは集合間の演算である以上、並列処理は効率向上のための有効な手段である。また、

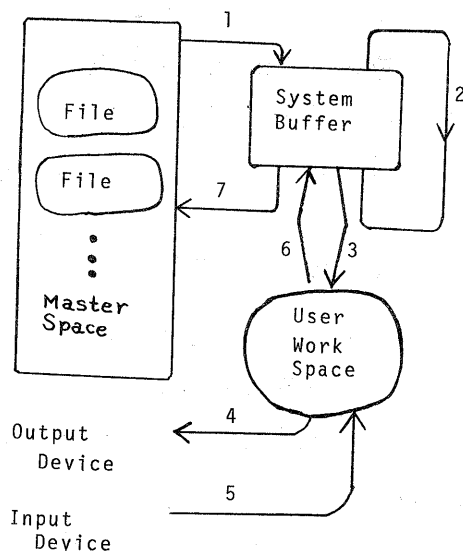


Fig. 2.1 Data Flow in DBMS

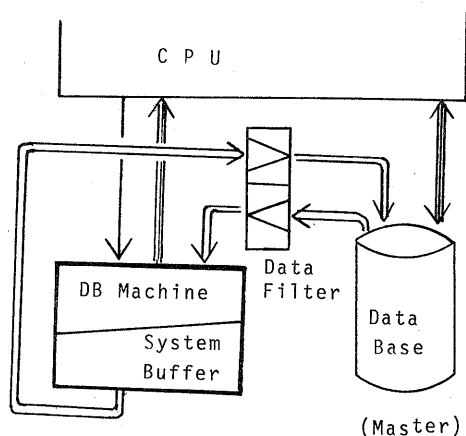


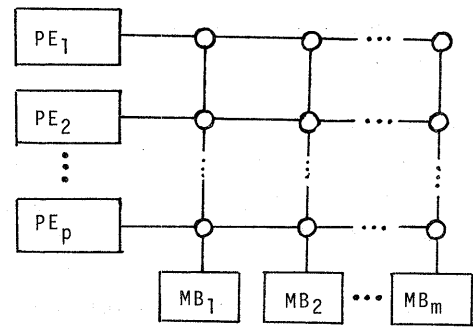
Fig. 2.2 Block Diagram of DB Machine

最近の技術的進歩から、多数のプロセッサによる並列処理方式は十分に実現性がある⁹⁾。ここでは図2.2のマシン形態をとるとして、DBマシンとシステム・バッファ上で、並列処理をどのように実現させるかを考える。

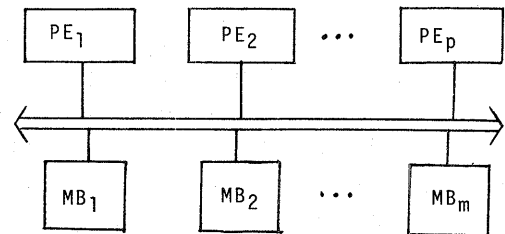
ところで、あるひとつの仕事(JOB)を複数のプロセッサで並列処理(parallel processing)させるには、つぎの2つの形態が考えられる。第1は、複数のプロセッサ間で共有するメモリをもち、各プロセッサが共有メモリ上にあるデータあるいは処理プログラムの一部を、任意に占有したり解放したりする、いわゆる多重処理(multi-processing)の形態である。第2は、各プロセッサが独自のメモリをもち、1まとまりの処理に必要なデータと処理プログラムをとり込んで、各プロセッサ毎に全体の仕事の1部を請け負う形の処理形態、いわゆる分散処理(distributed processing)である^{注1)}。

第1の形態では、1つの仕事を合計いくつの単位(プロセス)に分解するかは、あまり考慮する必要がない。すなわちプロセッサとプロセスは動的に任意に組合せることができるために、プロセスの個数は問題にならない。またこの意味で、仕事に対する各プロセッサの寄与は均等であると言える。これに反して第2の形態では各プロセッサに割当てられたプロセスは、始めから終りまでそのプロセッサ上で実行されるのが普通なので、処理効率向上のために、データの配分や処理内容を各プロセッサに対して均質にし、負荷をそろえる工夫が必要である。

このように、一般的には分散処理よりも多重処理の形態の方が有利であるが、プロセッサの数が多くなると多重処理には問題がある。すなわち、メモリ共有型並列処理の最大のネックは、メモリ・アクセスの衝突が起ることである。このため通常はメモリをいくつかのメモリ・バンク(MB)に分けておき、各プロセッサ(PE)に対してマトリックス状のスイッチを経由して割当てる(図2.3(a))か、共通バス線と各PEが時分割に利用する(図2.3(b))方法とする。しかし、前者では p 台のPEと m 台のMBに対して $p \times m$ 個のバス・スイッチが必要であり、これは高価で制御も複雑である上にPEやMBの追加が困難である。一方後者は共通バス線を1



(a) Matrix Switch



(b) Common Bus

Fig. 2.3 Shared Memory

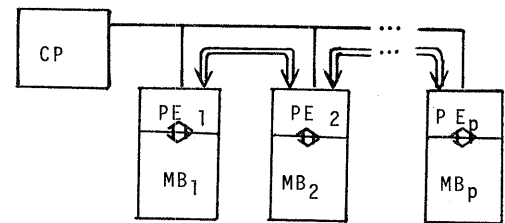


Fig. 2.4 Distributed Memory

台のPEあたり $1/p$ の時間しか使えず、データのやりとりが多いデータベース処理には不向きである。データベース・マシンで並列処理の利点を生かすためには、比較的簡単なプロセッサでよいが数は100以上は必要と思われる、多重処理方式では問題が多い。

図2.4に分散処理の1形態を示した。ここでは各PEに一定の大きさのMBを管理させている。ただし各PEは隣接したPEとデータのやりとりはできるものとする。また全体の制御用に別のプロセッサ(CP)があり、各PEに同時に命令を与えることができる。この方式は、多重処理に比べて制約が大きいが、後章で示すように、十分並列処理の特長を生かせる。

注1) 第2の形態で、各プロセッサが同じ処理プログラムを同期して実行する、という制限をつけたものがILLIAC-IVにみられるようなレイ・プロセッサである。ここでは、もっと広く、むしろ緊密に結合されたコンピュータ・ネットワークに近いものを考えている。

2.3. データベース・マシン向きの データ・モデル

データベース・マシンは、論理的に定義された集合演算を、ハードウェアによって高速に実行させることが目的であるから、データの物理構造と論理構造が近いものであることが望ましい。またハードウェアで並列的にアクセスするというためから、レコード内構造 (intra record) のように可変長で複雑な構造や、陽に定義されたポインタによる逐次的なアクセス手段もできるだけ避けた。

このような点と、多くのデータ・モデルの基本的性質を含んでいるという点から、E. F. Codd のリレーショナル・データ・モデル¹⁰⁾が有用である。しかし Codd モデルにはいくつかの問題点もあり¹¹⁾、またデータの依存性 (dependency) をはっきりさせることと、レコードの長さを小さくするために、ドメイン単位に「たて割り」の形でデータ蓄積するのがよい¹²⁾。ここでは、図 2.5. に簡単に例示するととめる。

2.4. ハードウェア・ソートの有用性

データベースの集合演算をハードウェアで実現させる場合に、もっとも必要な機能として連想記憶 (associative memory) がある。たとえば図 2.5 で、R1 と R2 の θ -join¹³⁾

$$R1[B=B]R2$$

をとる場合を考えると、R1 上の B-domain の各具体値に対して、R2 の B-domain で連想サーチを行う必要がある。(連想サーチを行わず、毎回全データをスキャンするとすれば、この join 演算に必要なデータ参照回数は、

$$(R1 \text{ の tuple 数}) \times (R2 \text{ の tuple 数})$$

となり、実際的でない。)

ところで、連想サーチを行うアーキテクチャとして次の3種が考えられる。

- ① メモリ自体に連想機能をもたせる。
- ② ハッシュ法などのリフト的な連想記憶アルゴリズムをハード化する。
- ③ 着目しているフィールドについて、ハードウェアで高速ソートできるようにする。

①は機能メモリ¹⁴⁾の提案などが以前からあるが、LSI 化したときの集積度やピン数などに問題があり、実際的でないと思われる。②は同

R1 (A B C)			R2 (B D)	
a1	b2	c1	b1	d2
a2	b1	c3	b2	d1
a3	b1	c2	b3	d2
a4	b3	c1		
a5	b2	c2		

Fig. 2.5 Sample Data

じ具体値が多数現れる場合問題があり、また大小関係の join (G.T.-join, L.E. join など) がとれななどの欠点があるが、使いかによっては高い効率を期待できる^{17,18)}。③では、ソートが大量のメモリに対しても、比較的簡単なハードウェアで高速に実行できるように工夫すれば、効率のよい処理ができる。すなわち、ソートされた N 個の要素中をバイナリ・サーチすればよいから、 $\log_2 N$ 回のサーチで必ず終了する。また大小関係の join や具体値の重複にも問題がない。ハードウェア・ソートは、その他、データベース上の他の集合演算や、DBMS の報告書作成機能でも有用である。本稿の次章からは、ハードウェア・ソートとその応用について述べられている。

3. ハードウェア・ソートの方式

本章では、チャネル型のデータベース・マシンで、流水こんでくるデータをソートしながら蓄積するためのアーキテクチャについて述べる。本稿のデータベース・マシンでは、少なくともソートすべき領域 (domain) の全具体値が、マシン上のシステム・バッファに収納できるものとしている。また、いったん蓄積したデータ集合を昇順または降順に高速に掃き出したリ、データ集合の一部を任意に消去したりできる機能も必要である。(消去しても、全体はソートされたままでなければならぬ。)

3.1. 特殊メモリによる方法

流水こんでくるデータをソートするためには、必ずデータの相対的な位置関係の変更を伴う。

この変更が、11かに並列的にうまく処理される
 かが、ソートの効率にかかわってくる。たとえ
 ば、この位置関係がデータの物理的な位置に直
 接対応しているとすると、1群のデータを並行
 して移動する作業が必要である。そのような機
 能を実現するには、Random Access Memory
 とShift Registerの両方を合わせたもの（こ
 こではSRAMとよぶ）を作ればよい。

図3.1にSRAMのブロック図を示した。
 このメモリは外から見ると、アドレス線、デー
 タ線は普通のRAMと同じであるが、オーダーは
 READ, WRITEの他にPUSHとPOPの計4種
 がある。PUSHは指定された番地から高位番地
 の方へ、データを1つつずつシフトする命令、POP
 は反対に指定された番地まで1つつずつ詰める命
 令である。メモリ内の各セル M_i は、与えられた
 アドレスと自分の位置との大小関係が判別でき、
 また隣のセルとの連絡線 T_i, T_{i-1} をもつ。各
 命令と、大小判別の結果、各セルの状態の変化
 を図3.1(b)にまとめてある。(X印の所は変化
 なし。)このメモリは通常のRAMと外部仕様を似
 せて、ビット・スライスに作る事ができるが、
 論理回路でセルを作るとすると、1セル当り筆
 者の試算では70~80ゲートになる。

次に重複のない m 個のデータ $d_0, d_1, \dots, d_{m-1}$
 のソート方法を簡単に示す。

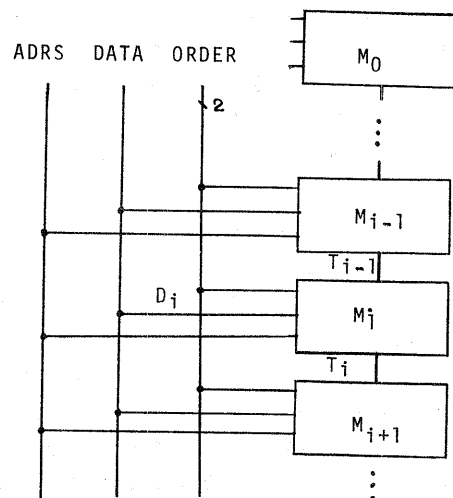
アルゴリズム

1. $M_0 \leftarrow 0$ (最小数), $M_i \leftarrow X$ (最大数).
 ($d_0, \dots, d_{m-1}$ は0とXを含まない)
 2. $d_0 \sim d_{m-1}$ を順次とり出し、各 d_j について
 次の操作を行う。
 - 2-1. M_0 から M_{j+1} までをバイナリ・サーチし
 て、 $M_{k-1} < d_j < M_k$
 となるような k を見つける。(READ)
 - 2-2. 2-1で求めた k を利用して、
 push(k) を行う。
 - 2-3. $M_k \leftarrow d_j$ (WRITE) 以上。
- この方法でソートに要する命令の数は、

$$\sum_{j=1}^m (2 + \log_2 j)$$
 のオーダーである。

また m 個のデータがすでに蓄積されていて、そ
 の中の1データを消去するには

$2 + \log_2 m$
 の命令ですむ。



(a) Block Diagram

	Compare	ADRS	ADRS	ADRS
Order		< i	= i	> i
READ	X		$D_i \leftarrow M_i$	X
WRITE	X		$M_i \leftarrow D_i$	X
PUSH	X		$M_i \leftarrow T_{i-1}$ $T_i \leftarrow M_i$	
POP	X		$M_i \leftarrow T_i$ $T_{i-1} \leftarrow M_i$	

(b) State Diagram

Fig. 3.1 Shifting RAM (SRAM)

このような特殊メモリは、前に述べたように
 高い集積度のものを作るのが困難で、高価なも
 のにつくのが難点であるが、システムの1部に
 使用するには有効であらう。

3.2. 並列プロセッサによる方法

3.1節のSRAMではデータ間の大小関係を、
 そのまま物理的位置に対応させたため、デー
 タの移動を各セル上で並列に行う必要が生じた。
 ここでは、データの移動を減らすため、ポイン
 タを利用することを考える。すなわちデータを
 いくつかブロック化し、ブロック間の大小関
 係はポインタで示すことにする。このような方
 式はソフトウェア上ではよく行われており、

B-tree による方法が効率が低いことが知られて¹⁵⁾、そこで B-tree を原理的には用いることにして、これを 2.2 節で述べた分散記憶の形態を利用してハードウェア的に高速化することを考えた。図 3.2 にこのブロック図を示してある。ここで各メモリ・ブロック (MB) に付属している SRAM は、全体として (横 1 列で) も SRAM になっている。MB は普通の RAM であり、プロセッサ (PE) によって読み書きされ、また SRAM に転送される。このメモリ上で各 MB に対して同じアドレスを与えたときにえられるデータの組 (図で破線で示してある) を B-tree のノードに対応させ、この中ではデータはソートされているようにする。また各データは、ノード間の順序を示すポインタ領域 (P) をもっている (図 3.3 参照)

次に図 3.2 のメモリ上で、3.1 節と同様、流れこんでくるデータをソートしながら蓄積するアルゴリズムの概略を示そう。ここで、 m は MB の個数で奇数である。

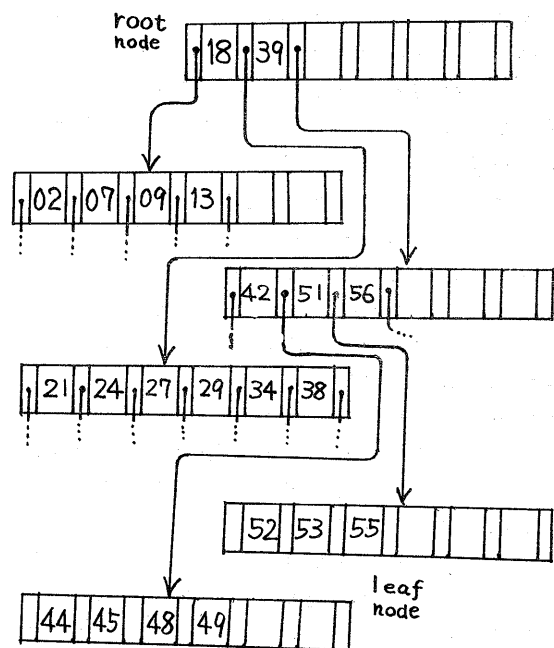


Fig 3.3. B-tree Example

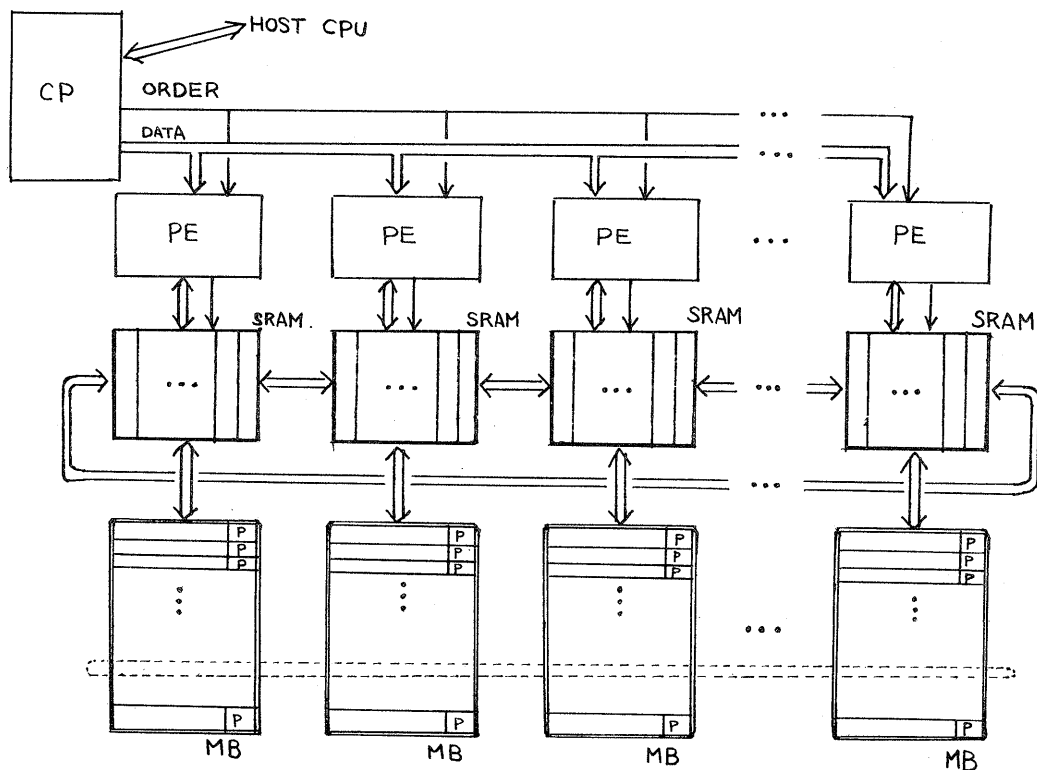


Fig 3.2. Distributed SRAM

アルゴリズム (データdの挿入)

- 与えられたデータdに対して, dにもっとも近い前後の値を含むノード (dを挿入すべきノード) を次の手順でさがす.
 - 1-1. ルート (root) ノードをSRAMにロードする. (CPから各PEに, ルートのアドレスを与え, SRAM上へデータを移動.)
 - 1-2. dと, ロードされたデータの大小判定を各PE上で行い, 次にロードすべきノードのアドレスを決定する.
もしもSRAM上のノードが最末端 (leaf) であれば 1. のおわり. 2へ.
 - 1-3. 1-2 で決った新しいロード・アドレスのノードをSRAM上にロードする. 1-2へ.
- 1で得られたノードにdを挿入する.
 - 2-1. SRAM上のデータ列中にdを挿入する. (3.1節のアルゴリズム)
挿入後のデータの総数がmより小さければ, そのまま元の場所にノードを戻して終り.
挿入後のデータ総数がmになったときは 2-2へ.
 - 2-2. ノードの後半分 ($m/2$ 個) のデータを新しいノード・アドレスを作って, そこへ移す. (同じMB内の, もとは別の番地への格納になる) 2-3へ.
 - 2-3. 元のノードの中央値 ($m/2 + 1$ 個目) のデータを, 1段階上のノードへ挿入するため, この値を新たにdとし, 元ノードをポイントして1段階上のノードを新ノードとしてSRAMにロードする. 2-1へ.

以上

上のアルゴリズムはB-treeの場合とほぼ同じであるが, ノードの開始点になるMBが2ヶ所できるため, SRAMの全体はサイクリックにしておく必要がある. また2-2のステップで作られる新しいノードの生成や, ノードとアドレスの管理はCPに行わせなければならぬ.

図3.2のアーキテクチャは, データの消去に関して, やはりB-treeの手法で行える.

上のアーキテクチャでは, 長さmのノードの内部での大小判定や移動は, m個のPEによって常にm重に並列処理されている. いまN個のデータが蓄積されているときに, あるデータの検索, あるいは新ノードの生成を伴わない追加を行うときの命令の数は

$$(2 + \log_2 m) \cdot \log m N$$

のオーダーである. また確保された記憶領域中の有効な領域量は平均 75%, データの追加時に新ノードの生成処理が必要になる確率は

$$2/m-1 \text{ である.}$$

なお上のアーキテクチャではPEに少し複雑な処理を負わせることにより, ノード中の各データ長を可変にできるし, leafノード中のポインタ・エリアの削除など, ソフト的なB-treeより高い機能をもたせ, また記憶領域を有効に使う工夫も可能である.

最後に本稿ではMBとしてRAMを想定してきたが, 図3.2のアーキテクチャは, 同時アクセスという性質上, 各MBをCCD, 磁気バブルなどのシフト形式のメモリにおきかえても, なお有効であることと付記しておく.

4. ハードウェア・ソートによるデータベース演算の実現

本章では前章までに述べたハードウェア・ソートの機能を利用して, データベースの基本的な演算がどのように実現されるかについて述べる. 本章で述べるデータベースは, 2章のドメイン単位蓄積方式を想定しており, またデータベース・マシンのアーキテクチャとして, 3章のハードウェア・ソートにもとづくチャネル型の並列処理マシンを考えている. (Coddモデルは, ドメイン単位やその他, 任意の形でデータを蓄積できるし, そのようにしても記憶領域のむだは少ない¹⁶⁾.)

ここではCoddの提案した集合演算¹³⁾を考えよう.

• Projection

関係Rはdomain $D_1, \dots, D_m$ から構成されているとする. projection

$$R[D_{r_1}, \dots, D_{r_k}]$$

をとるには, tuple $(d_{r_1}, \dots, d_{r_k})$ を順次のように蓄積すればよい.

- $(d_{r_1}, \dots, d_{r_k})$ がすでに蓄積されているか検索する. もしすでにあれば, 何もしないで次のtupleをよむ.

- まだ蓄積されていないければ, ソートした

がら3.2節のアルゴリズムで蓄積する。

Projectionの結果の集合はデータベース・マシン上に残ったものである。

Join

関係 R, S はそれぞれ domain $D_1, \dots, D_m, E_1, \dots, E_n$ から構成されているとする。

R, S の θ -join

$$R [D_r \theta E_s] S$$

をとるには次のようにすればよい。

1. R, S のうち tuple 数の少ない方 (仮に R とする) について, join の対象になる domain (D_r) の具体値を, 重複を許してソートしながら蓄積する。

このとき, 具体値と共に TID (tuple identifier, すなわちここでは R 中の tuple へのポインタ) をデータと一っしょに格納する。

2. もう一方の関係 (S) の tuple を順次データベース・マシンに送りこみ, 各 tuple の join のフィールド (E_s) と, 1. で蓄積した (D_r) の具体値との間で, θ で結びつくものを検索し, (S) の tuple と (D_r) の TID の組を返す。

上の方法で, TID で指定された R の tuple と S の tuple を結合した集合が本当に必要ならば, マスター・スペース上で, この情報をもとに併合する (merge) しかない。しかし, 通常は 1 つのファイル (S) からもう一方 (R) へ直接参照できればよいので, 上記演算で十分である。

Division

Join と同じ前提で division

$$R [D_r \div E_s] S$$

をとるには次のようにすればよい。

1. R の tuple の, D_r 以外の domain の具体値の組と, TID と, 具体値の組でソートしながら蓄積する。

2. 1 の集合と具体値の組が共通なものごと 1 つの部分集合を作り, この部分集合ごと 1 マスター側 (GPC) へ掃き出す。

3. S の tuple の, E_s domain の具体値をソートしながら蓄積する。

4. 2 でえられた部分集合ごと 1 データベース

・マシンに D_r の具体値を送りこみ, S の具体値を全部読んでみるかどうか調べる。尽していれば, その部分集合の共通な (D_r 以外の) 具体値の組が, 求める division の要素である。

この他, restriction は, チャネル型のデータベース・マシンである以上, 問題がないし, Union, Intersection も, ハードウェア・ソートができれば, そのまま処理できる。

5. むすべ

本稿では, ハードウェア・ソートを基礎におくデータベース演算子の実現およびデータベース・マシンの形成について, 筆者らの DB マシンの構想紹介も兼ねて, できるだけ一般性のある議論をしてきた。データベース・マシンの計画は, 現在電子技術総合研究所や北海道大学でもすすめておられ¹⁷⁾ 今後活発な進展が期待できる。

本報告を終えるにあたり, 日頃有益な示唆をいただいた電子技術総合研究所の面野博二部長, 植村俊亮主任研究官はじめ, 大型工業技術研究開発連絡会議データベース・マシンWG の諸氏に感謝の意を表します。

参考文献

- 1) 植村: "CODASYL 方式のデータベース・システム", 情報処理 Vol. 17, No. 10, 1976.
- 2) 穂鷹, 他: "データベースの関係形式", 同上.
- 3) 関野, 他: "データベース・マシン", 同上.
- 4) E. A. Ozkaran, et. al.: "RAP - An Associative Processor for Data Base Management." AFIPS (NCC), Vol. 44, 1975
- 5) M. Edelberg, et. al.: "Intelligent Memory", AFIPS (NCC), Vol. 45, 1976.
- 6) 有澤: "データベース・処理の専用マシン化について" 情報処理学会 17 回全国大会, 1976.

- 7) 有澤・土肥: "データベース・マシン向きのデータ処理方式." (投稿中)
- 8) S. Schuster, et. al: "A Virtual Memory System for a Relational Associative Processor." AFIPS(NCC) Vol. 45, 1976.
- 9) 飯塚, 他 "C. mmp マルチ・ミニ・プロセッサについて." 情報処理, Vol. 15, No. 7, 1974.
- 10) E. F. Codd: "A Relational Model of Data for Large Shared Data Banks", Comm. of the ACM, Vol. 13, No. 6, 1970.
- 11) 有澤: "リレーショナル・データベース上の領域間の決定関係についての考察" 情報処理, Vol. 17, No. 11, 1976.
- 12) 有澤・土肥: "リレーショナル・データベース・マシンについての考察" 昭52年電子通信学会総合全国大会.
- 13) E. F. Codd: "Relational Completeness of Data Base Sublanguages" Courant Computer Science Symposia 6, 1971.
- 14) P. L. Gardner: "Functional memory and its microprogramming implications" IEEE Trans. C-20, No. 7, 1971.
- 15) D. Knuth: "The Art of Computer Programming" Vol. 3, pp 473- Addison-Wesley
- 16) P. A. Alsberg: "Space and Time Saving through Large Data Base Compression and Dynamic Restructuring." Proc. IEEE, Vol. 63, No. 8, 1975.
- 17) 本研究報告と同時に発表される。