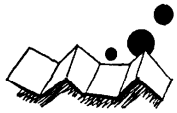
解　説



サーチ，ソート・ハードウェアと記号処理への応用†

田　中　　譲††

1. まえがき

データベース処理や，テキスト処理の高速化のニーズと共に，**サーチ**や**ソート**を高速処理するハードウェアに対する関心が高まりつつある．一方，VLSI 技術の発展は，VLSI 時代到来の感を与え，VLSI 化に適したサーチやソートのハードウェア・アルゴリズムの研究に拍車をかけている．そもそも，サーチやソートは，四則演算と同様に基本的な処理であり，データ処理の自動化の歴史の初期にはソート・マシンも存在していたのである．この解説では，サーチとソートを並列処理により高速に処理する種々のハードウェア・アルゴリズムを解説し，記号処理，特にデータベース処理への応用を概説する．

2. サーチ・ハードウェア

2.1 サーチ・ハードウェアの分類

n 個の固定長データの順序集合 $\{K_i | 0 \leq i \leq n-1\}$ に対し，キー k と，任意の関係演算子 θ を与えられたとき，添字の部分集合 $\{j | K_j \theta k\}$ を求める機能をサーチ機能と呼ぶ．データの順序集合を**テーブル**と呼び T で表し，k を**サーチキー**と呼ぶ．添字集合はサーチ結果と呼んで $S_\theta(k, T)$ で表す．この場合，T 中のデータが前もって昇順にソートされているか否かによって，**整列テーブル**のサーチと，**非整列テーブル**のサーチの2つの場合に分けられる．T がソートされていることを T* で示すことにする．

整列テーブルのサーチでは，$S_\theta(k, T^*)$ を求める代わりに，

$$\mathrm{mins}(k, T^*) = \begin{cases} 0 & \text{if } k \leq K_0, \\ (K_{j-1} < k \leq K_j \text{ となる}) j & \text{otherwise} \end{cases}$$

$$\mathrm{maxs}(k, T^*) = \begin{cases} n-1 & \text{if } K_{n-1} \leq k, \\ (K_j \leq k < K_{j+1} \text{ となる}) j & \text{otherwise,} \end{cases}$$

$$\mathrm{flag}(k, T^*) = \begin{cases} 1 & K_j = k \text{ となる } j \text{ が存在する場合} \\ 0 & \text{otherwise,} \end{cases}$$

と定義される **mins**, **maxs**, **flag** を求める機能を持ってもよい．

同一のテーブル T に対し，多数のサーチキー $k_0, k_1, \cdots, k_{m-1}$ を与え，

$$S_\theta(k_0, T),\ S_\theta(k_1, T), \cdots, S_\theta(k_{m-1}, T)$$

を求める処理を**一括サーチ**と呼ぶ．

サーチ・ハードウェアは，ハードウェアの持つ照合器の個数 h，サーチキー k を入力してから，k についてのサーチ結果を得るまでの最大遅れ時間 d, m が充分に大きい一括サーチの全処理時間の最大値を m で割ったスループット時間 t によって分類することができる．h, t, d が n に関する関数 $h(n), t(n), d(n)$ となるハードウェアを $(O(h(n)), O(t(n)), O(d(n)))$ 型と分類する．

$(f_1(n), f_2(n), f_3(n))$ 型のサーチ・ハードウェアが B 個あれば，テーブル T をほぼ等しい大きさの B 個のテーブルに分け，$(B \cdot f_1(n/B), f_2(n/B), f_3(n/B))$ 型のハードウェアを構成することができる．これを**バンク分け並列処理**という．サーチでは，どの型のハードウェアにもバンク分け並列処理を適用できるので，この種の並列処理は以下の解説から省く．

表-1 にサーチ・ハードウェアの分類を示す．

表-1　サーチ・ハードウェアの分類

h	d	t	整列／非整列	例
1	n	n	非	逐次サーチ
1	$\log n$	$\log n$	整	2分サーチ
l	n	$\frac{n}{l}$	非	多重キー逐次サーチ
l	$n+l$	$\frac{n}{l}+1$	非	パイプライン・多重キー逐次サーチ
$\log n$	$\log n$	1	整	パイプライン・一括2分サーチ[1]
n	n	1	非	パイプライン・一括逐次サーチ
n	1	1	非	連想メモリ[2]

h：照合器の個数
d：単一サーチキーの処理時間
t：一括サーチのスループット時間

† Searching and Sorting Hardwares and Their Applications to Symbolic Manipulations by Yuzuru TANAKA (Faculty of Engineering, Hokkaido University).

†† 北海道大学工学部

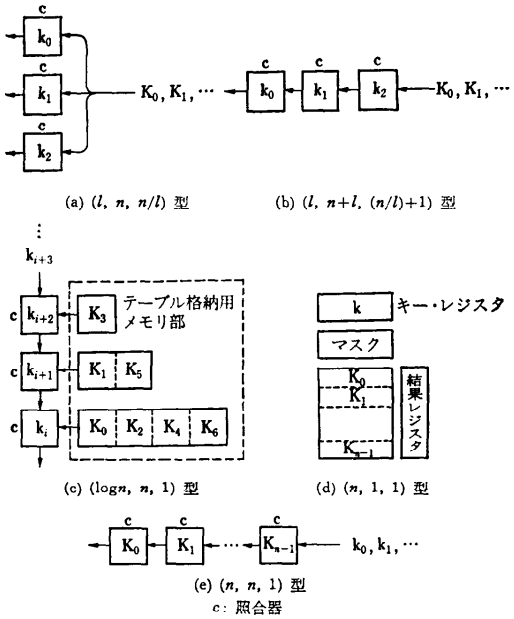


図-1　サーチ・ハードウェアの構成概容

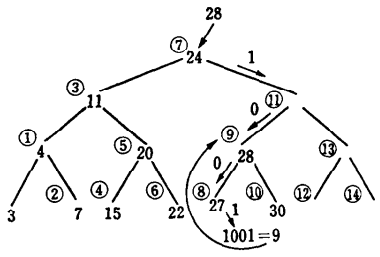


図-2　SEE でのサーチ過程

2.2 アルゴリズムの概容

表-1 において，$h\times t$ が n となっているものは，**逐次サーチ**を，$\log n$ となっているものは **2 分サーチ**を行う．複数個の照合器を持つものについて，ハードウェア構成を図-1 に示す．

パイプライン 2 分サーチは**サーチエンジン (SEE)** のハードウェア・アルゴリズムとして提案された方法で，一括サーチの高速処理に適している[1]。SEE では整列テーブルを図-2 に示す木構造のノードに ⓪, ①, … の順に格納する．SEE は図-1(c)のように，図-2 の木の各レベルに対応して独立なメモリと専用の照合器を持つ．サーチキー 28 に対するサーチは図-2 のように進む．28 が見つかった時はフラグを立てて左下へ進み，サーチを続ける．サーチが右下へ進むことを 1 で，左下を 0 で表すと，上から順にこの値をとり，左から右へ並べた 2 進数は，サーチが次に進むべき 1 レベル下のノードのレベル内での番地になっており，最終結果の 1001 (=9) は mins (28, T*)=9 であることを示す．SEE には maxs (k, T*) を求めるように指示することもでき，2 台の SEE を用いることにより，両者を同時に求めることができる．SEE を用いた一括サーチでは，各レベルに独立に在るメモリと照合器を用い，レベルごとに別々のサーチキーを比較処理することによって，パイプライン処理が行われる．

2.3 各種素子技術との親和性

$(1, n, n)$ 型，**$(l, n, n/l)$ 型**，**$(l, n+l, (n/l)+1)$ 型**はディスク装置，CCD，磁気バブル素子等の**回転メモリ**との親和性がよく，通常はバンク分け並列処理と併用して用いられる．

$(n, 1, 1)$ 型は**連想メモリ**であり，**$(n, n, 1)$ 型**は本質的には **$(l, n+l, (n/l)+1)$ 型**と同じであるが，照合器の個数を $O(n)$ にしたものである．いずれも **VLSI** 指向である．これらの型は，コスト，大容量化に問題がある他，テーブルデータとキーの語長の多倍長化のための接続には $O(n)$ のピン数を必要とする等の難点がある．

$(\log n, \log n, 1)$ 型では，n に比例して増えるのはメモリであり，多倍長化に要すピン数も $O(\log n)$ ですみ，メモリも一緒に 1 チップ化することにより，高速化できると共に，総ピン数も入出力用と多倍長化用のピン数の和に限定でき，**VLSI** 化に向いている．ただ，各レベルのメモリの大きさが，上位レベルから下位レベルにいくに従って倍々になっているため，パターン設計での幾何学的配置にやや難点を持つ．

3. ソート・ハードウェア

3.1 ソート・ハードウェアの分類

n 個の固定長データの順序集合 $\{K_i \mid 0\leq i\leq n-1\}$ に対し，置換 p を

$$K_{p(i)}\leq K_{p(j)} \quad \text{iff} \quad i\leq j \qquad (1)$$

を満足するある置換として，順序列

$$K_{p(0)}, K_{p(1)}, \cdots, K_{p(n-1)} \qquad (2)$$

を得る処理を**ソート**と呼ぶ．これに対して，数列

$$p(0), p(1), \cdots, p(n-1) \qquad (3)$$

を得る処理を**レコード・ソート**と呼ぶことにする．類似の処理として，p の逆置換を p^{-1} として，

$$p^{-1}(0), p^{-1}(1), \cdots, p^{-1}(n-1) \qquad (4)$$

なる数列を求める処理を**順番付け**と呼ぶ．単一プロセ

表-2　ソート・ハードウェアの分類

h	t	d	ソート	順番付け	例
1	n^2	n^2	○		バブルソート
1	$n\log n$	$n\log n$	○		ヒープソート
$\log n$	n	$\log n$	○		パイプライン・マージソート[3]
$\log n$	n	0	○		パイプライン・ヒープソート[1]
n	n	0		○	並列計数ソート[4]
n	n	0	○		パイプライン・バブルソート[5]
n	n	0	○		リバウンド・ソート[6]
n	$\sqrt{n}$	$\sqrt{n}$	○		格子上バイトニック・ソート[7]
n	$\log^2 n$	$\log^2 n$	○		バイトニック・ソート[8]

h：照合器の個数
t：処理時間
d：入力完了後出力開始までの時間遅れ

ッサで処理する場合，(3)から(2)への変換と，(3)と(4)の間の相互変換は，$O(n)$ の領域を用いて $O(n)$ の時間で処理できる．これに対して，(2)から(3)，(4)への変換は，$O(n)$ の領域と $O(n\log n)$ の処理時間を要する．

ソート・ハードウェアは，用いる照合器の個数 h と処理時間の最大値 t，それに，対象データの入力完了後からソート結果を出力し始めるまでの時間遅れの最大値 d の3つの量で分類することができる．h, t, d が n に関する関数 $h(n), t(n), d(n)$ となるハードウェアを，$(O(h(n)), O(t(n)), O(d(n)))$ 型と分類する．**表-2** に分類を示す．

3.2　アルゴリズムの概容

いくつかのアルゴリズムについて解説する．

(A)　パイプライン・マージソート[3]

ソートは，**図-3**(a)に示すように2つの系列のマージを繰り返し，順序よく並んだ部分列の長さを 1, 2, 4, … と倍々に長くしていけばよい．この処理を，(b)図に示す構成によってパイプライン処理する．各段のマージャは，前段から送られてくる2つの部分列の先頭が揃いしだいにマージを行う．

(B)　パイプライン・ヒープリート[1]

ソートエンジン (SOE) のハードウェア・アルゴリズムとして提案された．データ構造として**図-4**(a)に示すようなヒープ木を用いる．ヒープ木では根から各葉に至る道上のデータが昇順に並ぶ．SOE の構成は図-1 (c) の SEE の構成と類似しており，ヒープ木の各レベルに対応して独立なメモリと照合器を持つ．

データの入力と共に，ヒープ木は上位レベルから下位レベルへと成長し，レベル内では左から右へと成長する．初期状態のヒープ木の各ノードには全ビットが1のデータが格納されているものとする．図では∞で示している．次のデータが入力されると，∞から何等かのデータ値に変わるノードのことを次ノードと呼ぶ．図-4 (a) まで成長したヒープ木にデータを入力するには，根から次ノードに至る道上のデータをレベルごとに読み出し，これに新しいデータを挿入し，読み出した場所に再び書き込めばよい (**図-4** (b))．結果もまたヒープ木である．

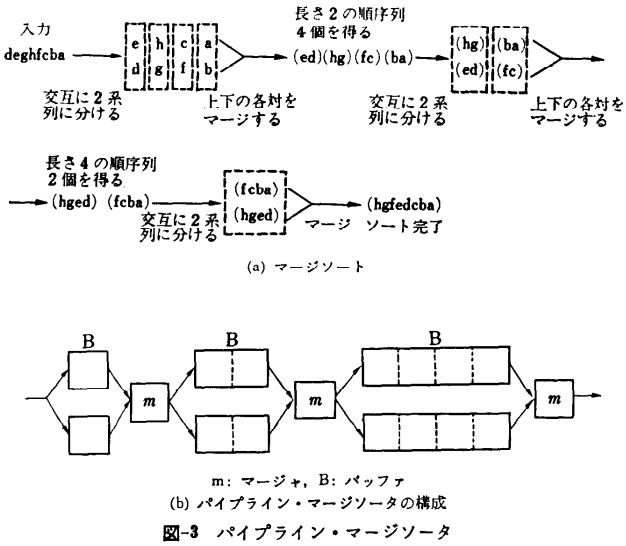


図-3　パイプライン・マージソータ

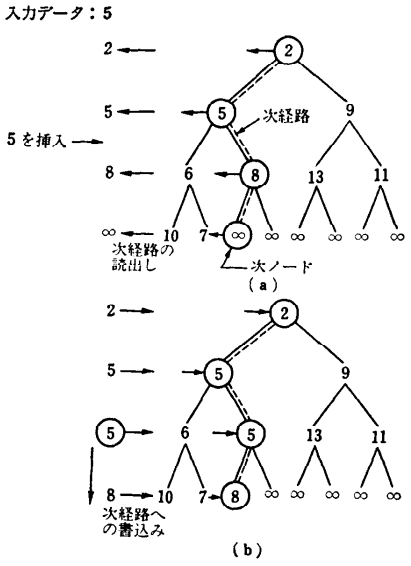


図-4　SOE のヒープ木構築法

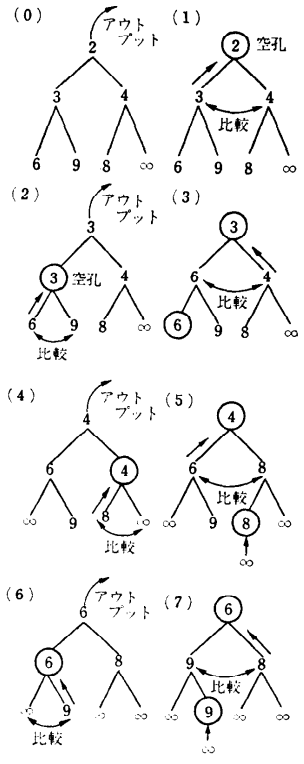


図-5 SOE のヒープ木からの出力法

ヒープ木からの昇順データの出力は 図-5 のように進められる．最小データは根にあるから，まず根の値が出力され，根に空孔が生じる．このように，あるレベルで空孔が生じると，次のレベルではこの空孔の左下と右下のノードのデータがメモリから読み出され，小さい方のデータが上のレベルへ送られ，選ばれた方のノードに空孔が生じたことが記憶される．上のレベルでは送られてきたデータが空孔の位置に書き込まれる．以上の動作を各レベルで並列に行うことによって，図-5 のようなパイプライン処理が実現できる．最下位レベルで空孔が生じた場合には∞が書き込まれる．これにより，出力終了時には，ヒープ木は初期状態に戻っている．

(C) バイトニック・ソート[8]

順序列 $K_0, K_1, \cdots, K_{n-1}$ が

$$K_0 \leq K_1 \leq \cdots \leq K_i \geq K_{i+1} \geq \cdots \geq K_{n-1}$$

のとき，**双調**であるという．図-6 (a) のような素子を用いることにより，$n=2m$ の双調列を (b) 図のようなネットワークでソートされた順序に並べ換えることができる．n が 2 の冪乗であれば，このネットワークは (a) 図の素子を縦に $n/2$ 個並べたものを $(\lg^2 n + \log n)/2$ 段横に並べて適当に結合することによって構成できる．この各段をすべて初段に重ね合わせ，基本素子を $n/2$ 個ですませたものが $(n, \log^2 n, \log^2 n)$ 型の**バイトニック・ソータ**である．

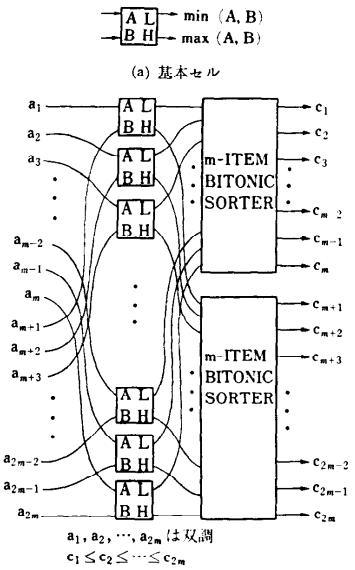


図-6 バイトニック・ソータ

この素子間の結合は複雑すぎるので，性能を少し落とし，結合を格子状の結合に制限したものが $(n, \sqrt{n}, \sqrt{n})$ 型の**格子上バイトニック・ソータ**である[7]．

リバウンド・ソータと**パイプライン・バブルソータ**の解説は 9) に譲る．

3.3 シミュレータ

レコード・ソートを行うためには，各 K_i と i を $K_i \circ i$ と連接して 1 つのデータとして扱い，これをソートして結果の添字部分を取り出すことで可能であるが，ソータの語長が充分であるか，語長の多倍長化の機能が必要である．

多倍長化接続と類似の接続を行う**シミュレータ**の概念を解説する．ソート・ハードウェアは一般に 図-7 の上部のような構造を持つが，図の下部に示されるように，これから照合器を取り除いたハードウェアを元のソータのシミュレータと呼ぶ．これらを図のように接続し，ソータには $\{K_i\}$ を，シミュレータには $\{i\}$ を入力することにより，レコード・ソートをシミュレータの出力として得ることができる．(f_1, f_2, f_3) 型のシミュレータ接続には $O(f_1)$ の結合線とピンが必要である．

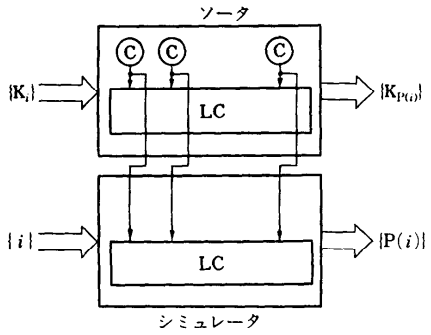


図-7 ソータとシミュレータの接続

3.4 各種素子技術との親和性

パイプライン・マージソータは **CCD**[3] や **RAM**[10] を使用するものが提案されている．**リバウンド・ソータ**は特殊な**磁気バブル素子**による実現が提案されている．複数個の照合器を持つその他のハードウェアは **VLSI** を指向したものが多くなりつつある．VLSI 化を考えると，$O(n)$ の照合器を持つものは，語長の拡大や，シミュレータ接続に $O(n)$ のピン数を必要とし，n を大きくできない．さらにまた，処理時間の短い**バイトニック・ソータ**や**格子上バイトニック・ソータ**では，データの入出力が処理の前後に独立に行われなければならないので，データが逐次入出力される場合には総合的な所要時間は $O(n)$ となって処理の高速性を活かすことができない．

これに対し，**(log *n*, *n*, log *n*) 型**や**(log *n*, *n*, 0) 型**は，ピン数も多くなく，逐次入出力を処理と重畳できるという利点がある．

4. サーチ・ソート・ハードウェアと記号処理

4.1 ストリーム処理

$(1, n, n)$型のサーチ・ハードウェアは，サーチキーの入った照合器にテーブル・データを逐次転送することで処理を行うことができる．同様に $(l, n, n/l)$ 型や $(l, n+l, (n/l)+1)$ 型のサーチ・ハードウェアでは，一度に l 個の異なるサーチキーを照合部に格納し，テーブル・データの逐次転送に同期して処理を行うことができる．このような処理を**オン・ザ・フライ (on-the-fly)** の処理と呼んでいる．一方，このようなハードウェアでは，一括サーチをサーチキーの逐次転送に同期して処理することはできない．

$(\log n, \log n, 1)$ 型や，$(n, 1, 1)$ 型，$(n, n, 1)$ 型では，サーチキーの逐次転送に同期して一括サーチを行うことができる．ただし，$(n, 1, 1)$ 型，$(n, n, 1)$ 型では，各サーチキーに対し，結果 $S_\theta(k, T)$ が n ビットとなり，この出力を転送に同期して取り出すことは実用上困難である．

ソート・ハードウェアでは，$(\log n, n, 0)$ 型，$(\log n, n, \log n)$ 型，$(n, n, 0)$ 型が，データ集合の逐次入力

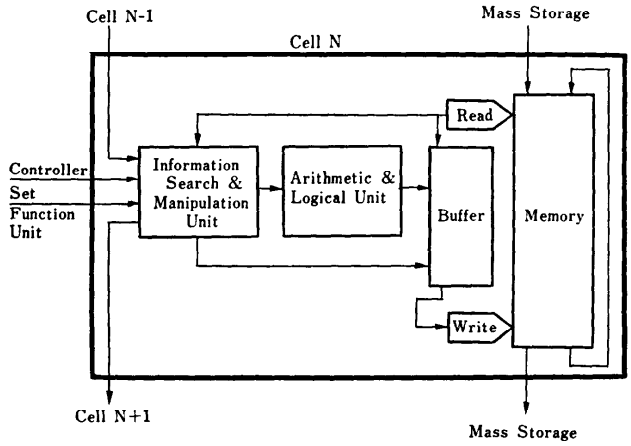


図-8 RAP のセル・アーキテクチャ

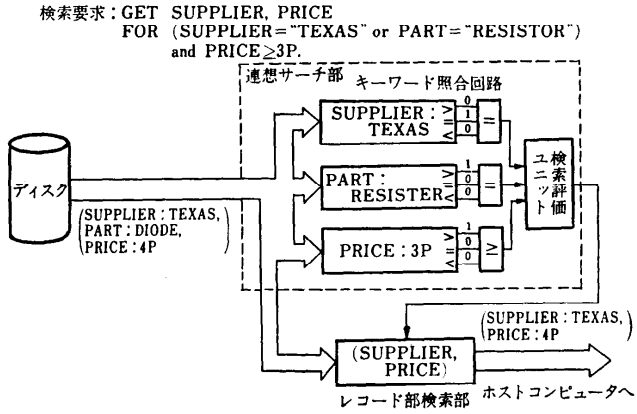


図-9 CAFS の基本アーキテクチャ

と，ソート結果の逐次出力に同期してソート処理を行うことができる．特に，$(\log n, n, 0)$ 型と $(n, n, 0)$ 型は逐次入出力に要する時間以外に処理時間を必要としない．

このように，処理と逐次転送を重畳させた処理方式を**ストリーム処理**と呼んでいる[1]．

4.2 データベース・マシンへの応用

記号処理への応用の一例として，**データベース処理**への応用例を示す．**図-8** は **RAP**[11] の基本セルで，図中メモリとあるのは，CCD が想定されている．サーチ部には $(l, n+l, (n/l)+1)$ 型のサーチ・ハードウェアが用いられ，CCD の回転に合わせてサーチが処理される．これにより，関係データベースにおけるセレクションの処理を行う．**図-9** は **CAFS**[12] の基本アーキテクチャを示したもので，$(l, n, n/l)$ 型のサーチ・ハードウェアが用いられている．最近の研究の方向は，**ジョイン演算**の高速化と，**大容量化**にあり，ストリーム処理型のソータを用いて，ジョインを $O(n)$ 時間で行うデータベース・マシンが提案されている[1),10]．ジョイン演算用の VLSI アルゴリズムも提案されているが，本質的には $(\log n, n, 0)$，$(\log n, n, \log n)$，$(n, n, 0)$，(n, n, n) 型のソータを用いたものである．

5. まとめ

1960 年代のサーチ・ソート・ハードウェアの研究の興味は**並列処理化**それ自体にあったといえる．1970 年代になると，**データベース処理**や**テキスト処理**の高速化にサーチやソートの専用ハードウェアを用いる試みが盛んになったが，前半は，実際的な見地から照合器の個数の少ないものが研究された．それが後半になって，**VLSI 技術**の発展に刺激されて，$O(\log n)$ や $O(n)$ 個の照合器を持つハードウェアの研究が盛んになってきている．今後 100 万素子，数 100 万素子の VLSI が実現可能になったとき，サーチやソートのような一般性を持った**ヘビー・オペレーション**の 1 チップ化が重要な意味を持ってくるであろう．

参考文献

1) Tanaka, Y. et al.: Pipeline Searching and Sorting Modules as Components of a Data Flow Database Computer, IFIP Congress '80, (1980).
2) Foster, C. C.: Content Addressable Parallel Processors, Van Nostrand Reinhold Comp. (1976).
3) Todd, S.: Algorithm and Hardware for a Merge Sort Using Multiple Processors, IBM J. R & D, Vol. 22, No. 5 (1978).
4) 安浦寛人他：並列計数法による高速ソーティング回路の設計，信学技報 (AL), (2月 1981).
5) Kung, H. T.: The Structure of Parallel Algorithms, in "Advances in Computers Vol. 19" Academic Press (1980).
6) Chen, T. C. et al.: The Rebound Sorter: An efficient Sort Engine for Large Files, Proc. 4th VLDB (1978).
7) Nassim, D. et al.: Bitonic Sort on a Mesh-Connected Parallel Computer, IEEE Trans. Computers, Vol. C-27, No. 1 (1979).
8) K. E. Batcher: Sorting Networks and their Applications, Proc. SJCC (1968).
9) 田中　譲：データベース：マシンの現状と未来展望，bit (3月 1982).
10) 喜連川優他：Hash と Sort による関係代数マシン，信学技法 EC 81-35 (1981).
11) Ozkarahan, E. A.: RAP- An Associative Processor for Data Base Management, Proc. AFIPS, Vol. 44 (1975).
12) Babb, E.: Implementing a Relational Database by Means of Specialized Hardware, ACM TODS, Vol. 4, No. 1 (1979).

(昭和 57 年 4 月 20 日受付)