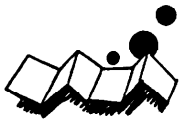
解　説

2. 専用 VLSI プロセッサの具体例



2.1 VLSI ソートプロセッサ†

喜連川 優†† 楊 維康†† 鈴木 慎司††

1. はじめに

ビジネスアプリケーションにおいてソート処理はCPU 全処理量の 30-40% を占めるといわれており，ソート処理の高速化に対する要望は強い．特に関係データベース操作，インデックス生成，レポート生成などにおいて，ソーティングは不可欠な処理といえよう．近年，半導体技術の進歩により並列ハードウェアを活用した高速大容量ソータの研究が活発に進められてきた[AKL 85]．本稿では LSI による実現を意図したハードウェアソートアルゴリズムを解説するとともに，実装されたソータを取りあげ，そのアーキテクチャについて解説する．

2. ハードウェアソートアルゴリズム

一般的なソートアルゴリズムを評価する際，平均実行時間，最悪時の実行時間，必要とするメモリの容量などに注目するが[KNU 73]，ハードウェアソートアルゴリズムの場合は，通常，ソート時間，所要のハードウェア量と遅延時間で評価する[TAN 83], [BIT 84]．

大容量データをソートする場合，そのデータは通常ディスクなどの二次記憶装置に格納されており，データの入出力はシリアルに行われるため，N をレコード数とする時，$0(N)$ 時間ソータの研究が特に盛んに行われてきた．バッチャのネットワークソーティング[BAT68]など，$0(N)$ を下回る時間でのアルゴリズムも知られているが，上記のデータの入出力時間を考慮すると，現実的な意味はそれほど大きいとはいえない．

実用的な高速大容量ソータを実現するためには，それを構成するためのハードウェア量が，実装上重要なパラメータとなる．すなわち，データの移動が局所的であり，処理の基本となる回路エレメントが単純で，かつその数が少ないことなどの特徴が要求される．また，回路エレメント間の接続が簡単で，大容量化への拡張が容易であることが望ましい．

ソータの遅延時間とは，ソート対象データの入力完了後，ソート結果の出力が始まるまでの最大値であり，$0(N)$, $0(\log N)$ 等さまざまなアルゴリズムがある．また，大容量データをソートする場合，データ入出力がソート時間の大半を占めるため，ソート結果の出力と重畳して，次にソートするデータストリームを入力する．すなわち複数データストリームのパイプライン処理ができるか否かも，ソータの特性を評価するときの重要なポイントである．

本稿では，$0(N)$ 時間でソートを実行可能なソータに絞って解説する．

2.1 ハードウェアソートアルゴリズムの分類

解説対象のソータをその構成によって，次の3種類に分類する．

① $0(\log N)$ 個の比較回路エレメントを用いるソータ

② $0(N)$ 個の比較回路エレメントを用いるソータ

③ テーブルルックアップ機構を利用するソータ

以上の①，②のソータは比較ベースのソート手法に基づいたものであり，③は非比較ベースのソート手法を用いている．

本節では，この3種類のソートアルゴリズムについて，各グループごとに，基本となるソートの手法，ソートエレメントの接続形態などを論じる．

2.2 $0(\log N)$ 比較器のソートアルゴリズム

$0(\log N)$ 個の比較器を用いるソータとしては，パイプラインマージソータ，パイプラインヒープソータが開発されている．いずれも，ソートに必要な時間は $0(N)$ であり，時間・ハードウェア量の積は $0(N \log N)$ で，最適といえる．

2.2.1 パイプラインマージソートアルゴリズム

[EVE 74], [TOD 78], [KIT 83]

マージソートの原理を示すマージ木を図-1(a)に示す．マージ木の最下段では偶数番目と奇数番目のレ

† VLSI Sort Processor by Masaru KITSUREGAWA, Weikang YANG and Shinji SUZUKI (Institute of Industrial Science, University of Tokyo).

†† 東京大学生産技術研究所

コードが比較され，大きい（あるいは小さい）ほうを前にして2レコードからなるストリングが上段へ出力される．マージ木の各段では，下位のノードから送られる二つのストリングをマージし2倍の長さのストリングを生成して出力する．このような各段のマージ操作により，最終段から出力されるストリングは入力レコード列を昇順(降順)にソートしたものになる．N個のレコードのマージ木の深さは $\log N$ であり，$\log N$ 個の比較器を用いてマージ操作をパイプライン的に並列実行し，$0(N)$ の時間でソートを完了させることができる．

パイプラインマージソータは**図-1**(b)のようにソートエレメント(SE)が1次元アレイ上に配列され，各段のソートエレメント SE_i が2本の 2^{i-1} レコードからなるストリングのマージを行う．また，各 SE には，マージ対象の2本のストリング中の第1ストリングを格納するために，2^{i-1} レコード分のメモリが必要である．

ソーティングのプロセスを**図**-1(c)に示す．ソータの入出力を含めてのソート時間は $2N+\log_2 N-1$ であり，遅延は $\log_2 N-1$，すなわちパイプラインセグメント数となる．

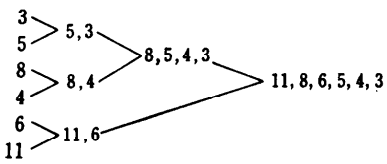
2.2.2 パイプラインヒープソートアルゴリズム [TAN 80]

ヒープソートは，ヒープ構成フェイズとデータ引き出しフェイズから構成される．

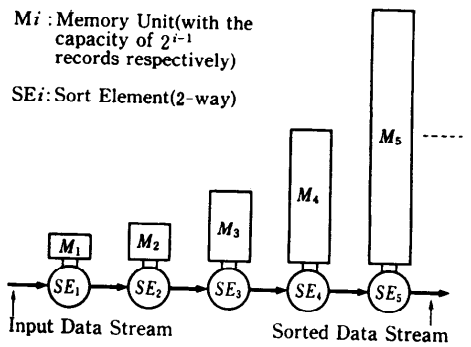
初期のヒープ木は完全2分木で，その根ノードからBreadth First 順に1から始まる番号が付けられている．ヒープ構成フェイズでは，i 番目のレコードが入力されると，番号 i のノードにつながるパスに沿って，ヒーププロパティを保持するようにデータの置換が行われる．たとえば，**図-2**(a)に示す5個のレコードが入力された段階でのヒープ木に，6個目のレコードが入力されるときには，パス (1→3→6) 上のデータが再配置され，**図-2**(b)のようなヒープ木が得られる．

このようなパスに沿ったデータの再配置を行うには，全レコードをアクセスする必要はなく，入力データをヒープ木の根から落とし込み，おのおのレベルのノードにおいて，上段のノードから降りてきたレコードと当該レベルのレコードを比較し，値の大きいほうをパスに沿って下段へ落とすようにすればよい．

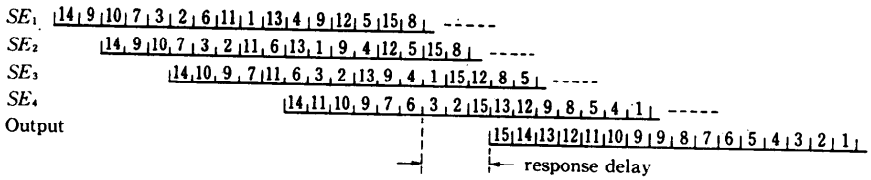
このようにして，すべてのレコードを入力し終わると，ヒーププロパティをもった二分木（ヒープ木）が構成される．そのヒープ木からソート列を得るデータ引き出しフェイズでは，ヒープ構成フェイズと逆の操作を行う．まず，根ノードのレコードを出力する．空になったノード（ホールと呼ぶ）に，その左と右の子ノードのうち小さいほうを送り込むと，ホールが一段下のほうに落ちる．この動作を繰り返し，ホールが最下段のノードまで伝播してゆくのと同時に，木の再構成



(a) マージ木の例



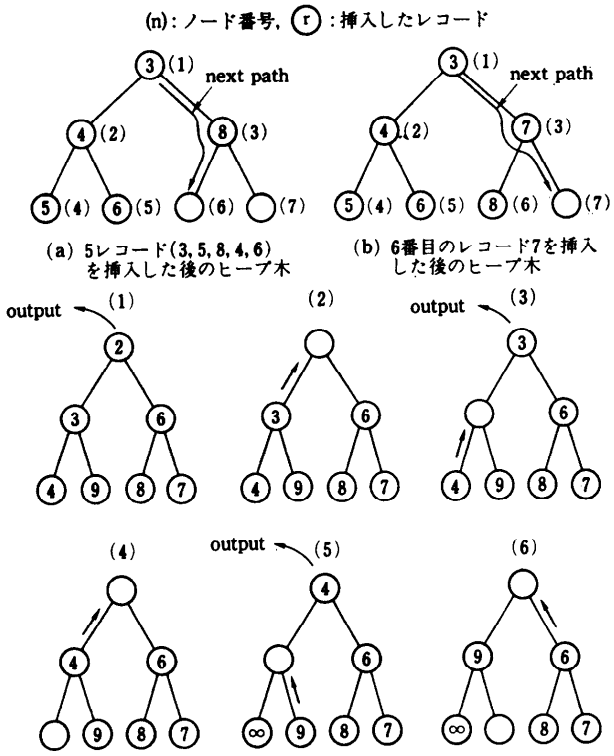
(b) パイプラインマージソータの構成



necessary sort element……$\log_2 N$
response delay ……$\log_2 N-1$
total sort time ……$2N+\log_2 N-1$

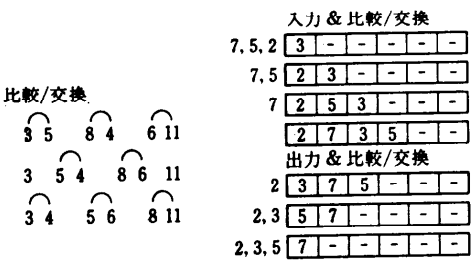
(c) パイプライン動作の様子

図-1 パイプラインマージソータ



(a) 5レコード(3, 5, 8, 4, 6)を挿入した後のヒープ木
(b) 6番目のレコード7を挿入した後のヒープ木
(c) データ出力フェーズにおけるホールの伝播の様子
(◯: ホール, ∞: システムの最大値)

図-2 パイプラインヒープソートアルゴリズム



(a) 奇偶交換ソート (b) 奇偶交換ソートの動作例

図-3 奇偶交換ソートのアルゴリズム

が行われ，パイプライン的に根ノードから次のレコードを引き出してゆくことができる．この引き上げとホールの伝播の様子を**図-2**(c) に示す．

ヒープソータは，ヒープの各レベルにプロセッサとデータを格納するためのメモリを設けて構成され，$\log N$ 台のプロセッサで $2N$ ステップでソートできる．

2.3 $O(N)$ 比較器のソートアルゴリズム

$O(N)$ の比較器を用いるハードウェアソータの基本アルゴリズムとしては，奇偶交換ソート (Odd-Even Transposition Sort)，計数ソート (Enumeration Sort) がよく知られている．

2.3.1 奇偶交換ソート

奇偶交換ソートはバブルソートを並列化したものと考えることができ，$O(N)$ 個のセルを用いるソータのほとんどは，このアルゴリズムを採用している．

奇偶交換ソートの動作はきわめて単純で，**図-3**(a)に示すように，$2i$ と $2i+1$ 番目のレコードに対する比較・交換操作と，$2i+1$ と $2i+2$ 番目のレコードに対する同じ操作の組を $\left\lceil \frac{N}{2} \right\rceil$ 回繰り返せばよい．

VLSI ソータを構成する場合には，隣りあった二つのデータと比較器を一つのセルに収めるようにすると，このアルゴリズムは容易に実装できる．また，入出力と比較交換をオーバラップさせ，入力(出力)と同時にセル間の比較と位置交換を行い，ソートアレイ内のレコードを順次シフトするようにして，$2N$ ステップでソートすることができる (**図-3**(b))．

2.3.2 計数ソート[YAS 82]

計数ソートはソート対象のレコード列 $R_1, R_2, \ldots, R_N$ に対し，各レコードの順位 (ランク) を求め，そして，そのランクの順にデータを並べ替えることでソーティングを実現する．i 番目のレコード R_i のランクは次のように計算できる．

```
Ci=0;
for (j=1; j<=N; j++){
    if (Ri>=Rj)Ci=Ci+1;
}
```

このアルゴリズムを並列化し，1次元のセルアレイで実装するために考案されたアルゴリズムが並列計数ソートである．並列計数ソータは**図-4** に示す構造を有し，その動作は次の三つのフェーズからなる．

① 入力，計数獲得 (Count aquisition)
② 再配置 (Rearrangement)
③ 出力 (Output)

入力計数獲得フェーズにおいては，1サイクルごとに各セルの Y レジスタの内容を右にシフトしながら

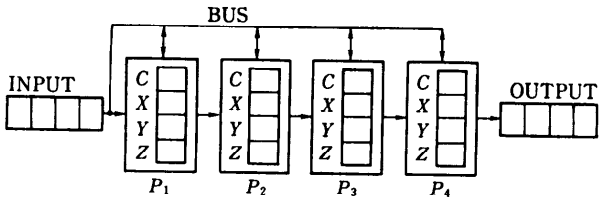


図-4 並列計数ソータのアーキテクチャ

ソート対象のレコードが P_1 の Y レジスタに入力される．i 番目に入力されたレコード R_i が同時に，P_i の X レジスタにも格納され，そして，$X>Y$ の場合には C レジスタがインクリメントされる．このようにして，P_i の X レジスタに格納されたレコード R_i は，入力開始から $N+(i-1)$ ステップ後には，全レコードとの比較が終了し，上記アルゴリズム中の変数 C_i に相当し，P_i の C レジスタ中にレコード R_i のランクが保持される．

データ入力終了後（$N+1$ ステップ目），1番目のレコードのランクが確定しているので，そのランクを Z とすると，P_1 の X レジスタの内容（R_1）を，P_z の Z レジスタに転送する．これが再配置である．そして後続のステップでは $P_2, P_3, \ldots$ の再配置を行う．

こうして $2N$ ステップ後には，$P_1, \ldots, P_N$ の Z レジスタに，ソート済みのレコード列ができる．出力フェーズでは，このソート済みの列を端から順に取り出す．

2.4 テーブルルックアップを利用したソートアルゴリズム

Content to Address のマッピングを利用することで，$0(N)$ の時間でのソートを，1台のプロセッサを使用して実現することができる．ただし，必要なメモリ量はマージソータなどに比べ大きくなる．

ラディックスソートの原理に基づき，レコードのキーの値をアドレスに変換し，得られたアドレスにレコードを書き込む．全レコードがメモリ中に書き込まれたら，メモリを順番に走査し，メモリ中のレコードを拾い出してゆけばソート列が得られる．

このソートアルゴリズムを利用するためには，以下の三つの問題点に対処する必要がある．

① 重複したキーへの対応
② 必要なメモリサイズを限定する手法
③ ソート結果抽出の高速化

以上の①については，衝突したレコードをリストでつなぐなどの方法で解決できる．②については，バケットソートの原理を利用して，キーの1バイトごとにソートを行う方法を用いることができる．③については，特にキーの分布が密でない場合，メモリスキャンを避けるために，プライオリティエンコーダを用いて，データの入っているメモリアドレスを検索する解決法がある．

3. VLSI ソートプロセッサの具体例

前章の基本となるソートアルゴリズムの解説につづき，本章では，VLSI によるハードウェアソータの実装方式について解説する．ハードウェアソータについては盛んに研究が進められ，現在では，商用のプロセッサも登場している．

3.1 $0(\log N)$ 個の比較器を使用するソータ

$0(\log N)$ 個の比較器を利用するソータとしては，パイプラインヒープソータ，パイプラインマージソータなどが開発されている．これらのソータの特長として，

① 比較器としてのソートエレメント（SE）が比較的複雑になるが，その必要な数が少ないため，ソータ全体のハードウェア量がきわめて少なくて済むこと．

② いわゆる logic in memory のアプローチと異なり，メモリとソートエレメントと分離されているため，最新の半導体メモリ技術を利用できること．

などがあげられる．近年，半導体メモリの容量増加は著しい進歩を遂げてきた．本ソータでは，容量に対して所要のソートエレメント台数は容量の対数に比例するため，ソータの大容量化がメモリと少数のソートエレメントの追加で容易に実現できる．

パイプラインヒープソータに比べ，パイプラインマージソータは遅延時間は $\log_2 N-1$ であるが，複数のデータストリームをパイプラインに処理することが可能である．

3.1.1 パイプラインマージソータ

パイプラインマージソータは，東京大学で改良された後[KIT 83]，第五世代コンピュータプロジェクトにおける DELTA マシンの構成要素として ICOT において構築された[IWA 87]．また最近，日立よりデータベースベクトルプロセッサとして，二つのベクトルをマージするハードウェアが開発され，IDP の名称で商用化された．さらに，三菱電機が東大で開発されたソータを基にして，19 段のソータをオフィスコンピュータの付加プロセッサとして実用化している．

a） 東大・三菱ソータ[KIT 83], [KIT 87], [KIT 89]

東京大学で開発されたパイプラインマージソータ全

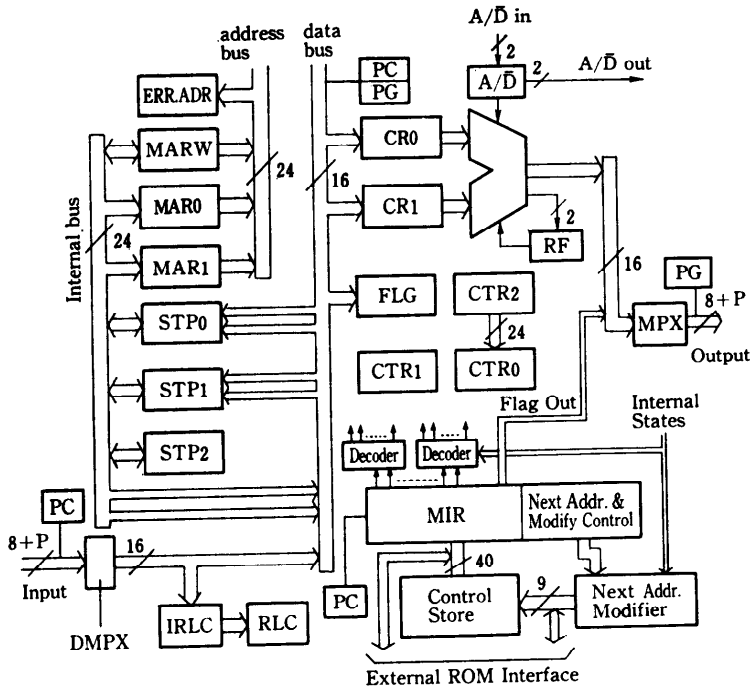


図-5 東大パイプラインマージソータのソートエレメントの構成

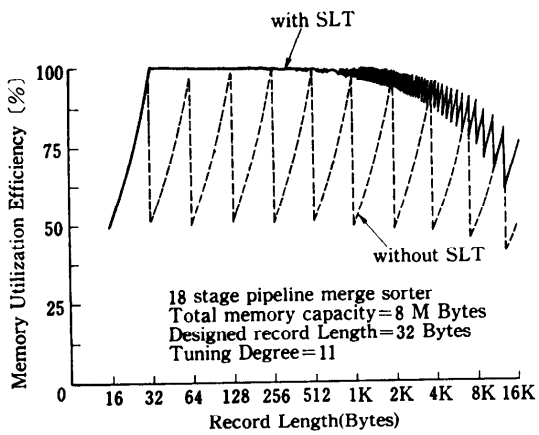


図-6 SLT 機能によるメモリ利用効率の最適化

体の構成は，図-1(b) に示すように，n 個のソートエレメント (SE) を一次元に結合し，各段には 2^{i-1} レコード分のメモリを付加する．SE の構成を図-5 に示す．各種の柔軟性を実現するために必要な制御情報は，2 ワードのフラグとして入力データの各レコードの先頭に付加される．その領域はメモリ内ではポインタ領域として利用され，メモリに格納されたレコードはリスト構造によって管理されている．SE 内の各レジスタの操作及び内部状態遷移の制御は水平型マイクロプログラムによって行われる．

ソートするデータのパラメータ（レコード長，レコード数）に対する柔軟性を重視して実現した点が他に例のない本ソータの最大の特徴といえる．特にレコード長は通常，ハードウェアの実装で決められた資源に制限されがちで，決められたレコード長と異なるデータファイルのソートは著しく効率が低下する．これに対処するため，本ソータでは String Length Tuning (SLT) アルゴリズムを実装している．SLT は入力レコード長に応じて，各ソートエレメントが出力する部分列の長さを調整しソータ全体のメモリ利用効率のし，最適化を行う．図-6 からパイプラインマージソータにおいて SLT 機能があるとき，メモリ利用効率が大幅に改良されることがよく分かる．

本ソータのソートエレメントは東京大学において SSI, MSI により試作された後，三菱電機により LSI 化された．その性能など諸元を表-1 に示す．19 個のソート LSI を 2 枚の基板上に実装し（図-7 は 15 個のソート LSI を実装した基板の写真である），オフィスコンピュータの付加プロセッサ GREO として商用化されている[AND 89]．

b) IDP[KOJ 87]

IDP は日立製作所の大型メインフレーム M 680 H のオプションとして開発された非数値演算用ベクトルプロセッサである．高速ベクトルマージユニットを中心とする IDP は 2-WAY マージをベースに関係データベース処理の高速化を図っている．ソートは 2 本のベクトルのマージ操作命令を $\log_2 N$ 回繰り返し使用することにより実現される．

3.1.2 パイプラインヒープソータ

パイプラインヒープソータは北海道大学で TTL. MSI を用いて試作された[TAN 83]．ヒープソートを並列に行うために，ヒープ木の各段に一つの比較器を配置し，n 個の比較器で 2^n-1 個のレコードをソートできる．

表-1 東大・三菱ソート LSI の性能諸元

ソート速度	最大 8 MBytes/sec.
ソートデータ量	最大 64 MBytes
ソートデータ数	最大 200万個 ($=2^{21}$ 個)
ソート可能なレコード長	2 B-64 KB まで2単位に可変．ただし同一ソートデータ集合内では固定
キー数	制限なし
キー長	制限なし
データタイプ	ビットパターンの単純比較（レコードの再上位ビットからの比較）のみ
昇降順	1バイトごとに指定可能
LSI プロセス	CMOS 1.3 μ, Al 2層 VTM ゲートアレイ（三菱電機製）
ゲート数	12,000 ゲート+12.8 Kbit ROM
パッケージ	160 pin, Quad Flat Package
使用マスタ	M 6003 X シリーズ（三菱電機）
Basic Cell 数	ROM 27,960 Logic 35,949 ($=11,983$ Gates) Total 63,909

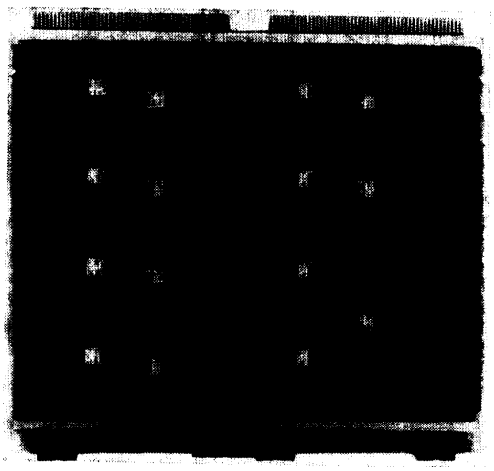
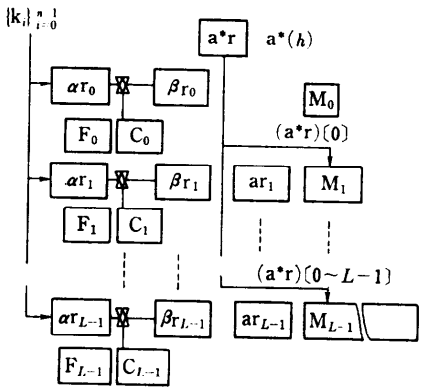


図-7 15個のソート LSI を実装した基板

図-8 に，ソートエンジン (SOE) の構成を示す．ヒープ構成フェーズでは，次に入力するレコードがバスKを用いて，ヒープ木の各段のプロセッサ P_i にブロードキャストされ，αr_i に格納される．これと同時に，再配置を行うパスを指定するためのフラグも，バスで各プロセッサにブロードキャストされる．P_i はこのパスフラグによりローカルメモリのアドレスを生成して，そのパス上のノードに相当するレコードを βr_i にロードし，αr_i の内容と比較し，入力レコードのパス上での挿入位置を決定する．挿入位置以下のプロセッサ P_j は，上段のプロセッサの βr_{j-1} レジスタの内容を読み込み，メモリに格納する．



$\alpha r_i, \beta r_i$: registers, F_i: even/odd flag, M_i: memory, ar_i: address register, a*r: address generator. (a*r) [0～l]: the leftmost $l+1$ bits of the register a*r.

図-8 北大パイプラインヒープソータの構成

データ出力フェーズでは，P_0 からレコードを出力する．P_0 以下のプロセッサは上段のプロセッサのパスレジスタ ar_i を参照してアドレスを生成し，その二つの子ノードのレコードの比較を行うが，ホールが発生したプロセッサは次段のプロセッサの比較結果によって1個のレコードを読み込み，ホールを次段に伝播する．

パイプラインヒープソータの利点は，データ入力終了後，遅れなしにデータの出力が開始できること，ソータ容量よりも少ないデータに対しても，データの個数に比例した時間でソートできることなどである．データ容量 4095 語，語長 16 bit，処理速度 250 K 語／秒の試作機が構築された．また，ソートするデータの語長を拡張するために，ビットスライスアーキテクチャが提案されている．

また，アルゴリズムがやや異なるが，パイプラインヒープソータの出力モードと似たような動作で，大容量ファイルのマージソートを行うソータが，土肥らによって提案された[DOH 84]．

3.2 $O(N)$ 個の比較器を使用するソータ

この種の VLSI ソータでは $O(N)$ 個の比較器とメモリを用意し，メモリと比較器を一体として一つのセルを構成し，そのセルを線型に結合しソータを形成する．そこで，この種のアプローチは，広い意味で Logic in Memory 方式といえる．

半導体の集積率が低い時代には，この方法はまったく現実性を欠いたものであったが，近年の急速な集積

度の向上によって，本方式による多くのソータが設計されている．

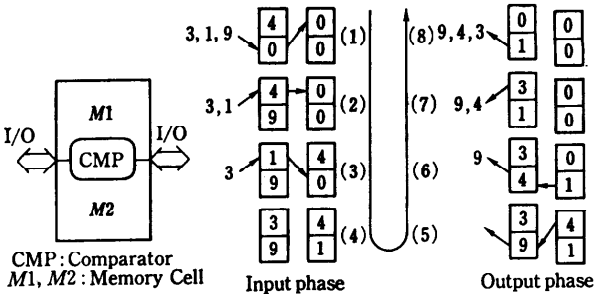
2.3 で述べたように，この種のソータの基礎アルゴリズムとしては，奇偶交換ソート，計数ソートがある．特に奇偶交換ソートに関しては，いくつもの実装例がある．奇偶交換ソートではグローバルコミュニケーションの必要がなく，セルの構成も簡潔で VLSI 化に特に適したアルゴリズムであると考えられる．また複数のセルを集積したチップをカスケード接続することにより，容易に大容量化できるという利点もある．

3.2.1 奇偶交換ソートを基本とするソータ

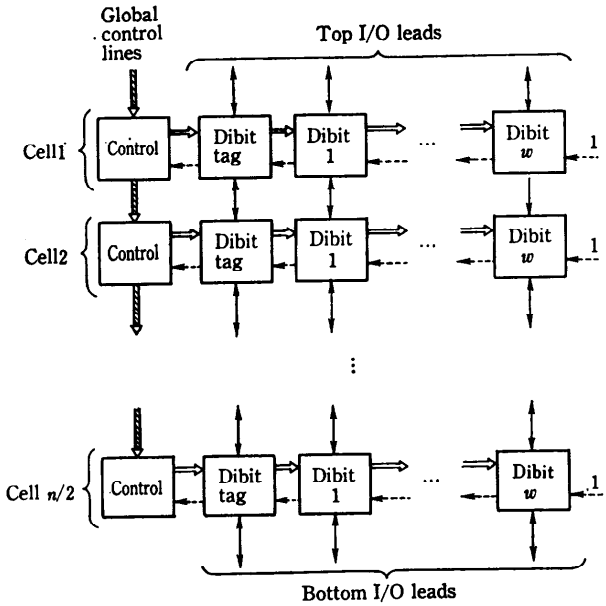
代表的ソータとして Zero Time Sorter[MIR 83]，NTT ソータ[SAT 89]，RESST[CAR 82] があげられる．この三つのソータは，すべて入出力とパイプライン化された奇偶交換ソートを基本とし，実装の方法もほぼ同様である．

ソータの基本ユニットは図-9(a) に示すように，比較器と二つのメモリを含んだセルであり，その動作は二つのフェーズからなる．まず比較フェーズでは，メモリ $M1$ と $M2$ 中のレコードを比較し，次のフェーズで入出力の対象となるメモリを決定する．転送フェーズでは，比較フェーズで決定されたメモリのデータを出力ポートに出力するのと同時に，入力ポートから送られるデータを同メモリに入力する．

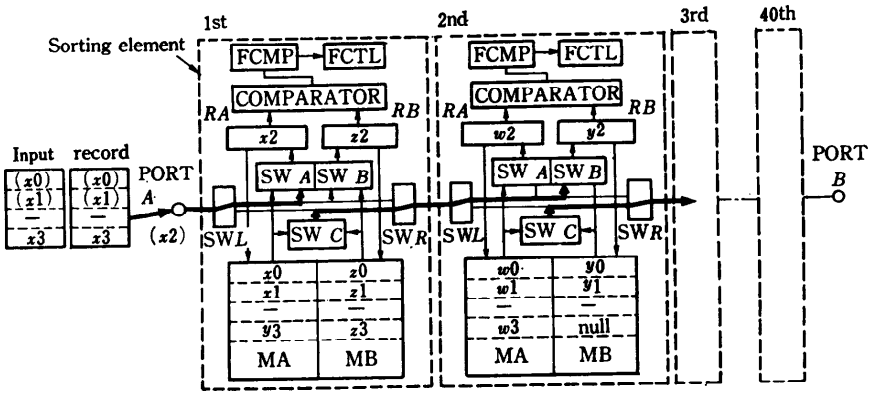
ポートからの入力（入力が終了したら出力）と上記の動作を交互に行うことにより，出力時にはソート済みの列を得ることができる．ソート動作例を図-9(b) に示す．以下では Zero Time Sorter,



(a) 奇偶交換ソータの基本ユニット　(b) 奇偶交換ソータの基本動作例



(c) IBM Zero Time Sorter の構成



(d) NTT ソータの構成

図-9 $O(N)$ 比較器ソータの実装例

NTT ソータについて解説する.

a) IBM Zero Time Sorter

IBM の Miranker らによって提案された Zero Time Sorter の構成を図-9(c) に示す. 一つのソートエレメントはビット幅の数だけの Dibit セルから構成される. Dibit セルは, 2 bit のメモリセルと 1 bit の比較器からなる. 比較器の出力は下位ビットから上位ビットへとカスケード接続され, Control 回路で処理され, 転送フェーズで転送されるビットの選択のために使用される.

b) NTT ソータ

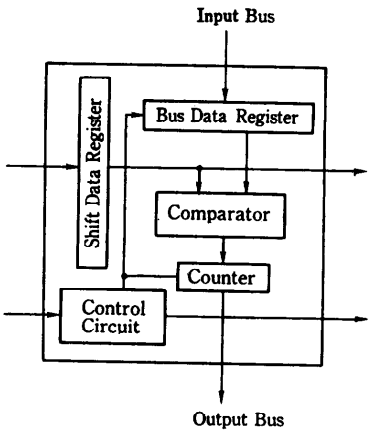
NTT ソータの構成を図-9(d) に示す. ソートエレメントは, バイト幅の比較器と 16 バイトのメモリから構成されている. 本ソータにおいては, バイト単位でセル間の転送を行い, 同時に転送されたデータと, もう一方のメモリユニット中のデータの比較を行うことで, 前述の比較フェーズと転送フェーズがオーバラップして実行される. そして, レコード長を拡張するために隣接するセル間にレコードを分割して格納し, ソートするための機能が付加されている. また Zero Time Sorter と同様に, 右側のポート B からもデータの入力が可能であり, ポート A からデータ出力をしている間に, ポート B から次のソートデータを入力できる.

このソータは, 1 チップに 40 セルを集積した LSI 化が行われており, マルチウェイマージソータのビルディングブロックとして, データベースプロセッサ RINDA において使用されている[INO 89].

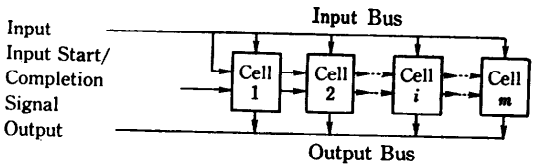
3.2.2 並列計数ソータ[YAS 82]

京都大学の安浦らによって提案された並列計数ソータのセル構造とその接続法を図-10(a), (b) に示す. Bus Data Register (BDR) は, 入力キー列中の一つのキーを保持するためのレジスタで, アルゴリズムの解説で述べた X レジスタに相当する. Shift Data Register (SDR) は左隣のセルからシフトされてくるデータを 1 クロックの間保持し, 右隣のセルに送り出す (Y レジスタに相当).

データが SDR に保持されている間に比較器 (Comparator) によって, BDR と SDR が比較され, 結果によってカウンタ (Counter, C レジスタに相当) がインクリメントされる. 制御回路は, 左隣のセルから入力開始信号を受けると, 入力バス (Input Bus) よりデータを BDR に取り込み, カウンタをリセットする. また, 入力終了信号を受け取ると, カウンタの値



(a) 計数ソータのセル構造



(b) 計数ソータの構成

図-10 京大並列計数ソータの実装

を Output Bus へ出力する. 参考文献 [YAS 82] では, レジスタ長を 32 bit, カウンタ長を 16 ビットとして, nMOS 技術による集積化について考察している.

3.3 テーブルルックアップを利用したソータ

ここでは, その他のアルゴリズムを用いたソータとして, SFU-DB[LEE 87] について解説する.

SFU-DB はフロリダ大学で開発されている, ソートとソートを基にしたデータベース操作のための専用プロセッサである. SFU-DB は MSB First のラディックスソートを基本とするアルゴリズムを採用した, ARM (Automatic Retrieval Memory) と呼ばれる専用ハードウェアにより, 単一プロセッサで $0(N)$ 時間のソートを実現している.

処理はバイト単位で行い, ソートキーの最上位バイトから行う. ソートキーの値によって, そのレコードへのポインタを 256 個のエントリを 1 ページとする ARM の対応番地に格納する. 衝突はフラグで示され, 別のメモリにリスト構造により格納される. すべてのレコードの処理が終了すると, ARM から順番にポインタを抽出する. これはプライオリティエンコーダでエントリのないメモリ番地へのアクセスを避けている. ソートキーが複数バイトの場合, 前のページはスタックに入れて, 重複した属性に対して ARM の別のページで次の上位バイトについて同様にソートする.

4. おわりに

本稿では，ハードウェアソートアルゴリズム，ならびにその VLSI による実装例について述べた．ソートはもっとも基本的なデータ処理の一つであり，その高速化はソートを利用する多くのアプリケーションの性能向上に寄与することから，現在でも活発に研究が行われている．特に最近ハードウェアによるソートの支援が活発になされており，本稿で紹介したように，日立，NTT，三菱のデータベースプロセッサはすべて専用ソータを搭載している．また，AMD より Content Addressable Data Manager (Am 95 C 85[AMD 85]) なるソートチップも発表されており，今後，ハードウェアソータの実用化はますます進むものと思われる．

参考文献

[AKL 85] Akl, Selim, G.: Parallel Sorting Algorithm, Academic Press, Inc. (1985).

[AMD 85] Advanced Micro Devices: Am 95 C 85 ADVANCED INFORMATION.

[AND 89] 安藤隆朗他：リレーショナルデータベースプロセッサ GREO の構成，信学技報，Vol. 89, No. 335 (1989).

[BAT 68] Batcher, K. E.: Sorting Networks and Their Applications, Proceedings of the 1968 Spring Joint Computer Conference (Atlantic City, N. J., Apr. 30-May 2) Vol. 32, AFIPS Press Reston, Va. (1968).

[BIT 84] Bitton, D., DeWitt, D. J., Hsiao, D. K. and Menon, J.: A Taxonomy of Parallel Sorting, ACM Computing Surveys, Vol. 16, No. 3 (1984).

[CAR 82] Carey, M. J., Hansen, P. M. and Thompson, C. D.: RESST: A VLSI Implementation of a Record-Sorting Stack, Tech. Report of UCB, No. UCB/CSD 82/102.

[DOH 84] 土肥康孝：大容量ファイルを整列するシストリック・ソータ，電子通信学会論文誌 '84/3 Vol. J 67-D, No. 3 (1984).

[EVE 74] Even, S.: Parallelism in Tape Sorting, Commum. ACM 17(4) (1974).

[INO 89] Inoue, U., Hayami, H., Fukuoka, H. and Suzuki, K.: RINDA—A Ralational Database Processor for Non-Indexed Queries, Proc. Int. Symp. Database Systems for Advanced Applications (1989).

[IWA 87] 岩田和秀他：関係データベース処理エンジンのソータの試作と評価，情報処理学会論文誌，Vol. 28, No. 7 (1987).

[KIT 83] 喜連川優他：パイプラインマージソータの構成，電子通信学会論文誌 '83/3 Vol. J 66-D, No. 3 (1983).

[KIT 85] Kitsuregawa, M. et al.: Memory Management Algorithms in Pipeline Merge Sorter, Database Machines, Fourth International Workshop, Springer Verlag (1985).

[KIT 87] Kitsuregawa, M., Yang, W., Suzuki, T. and Takagi, M.: Design and Implementation of High Speed Pipeline Merge Sorter with Run Length Tuning Mechanism, Database Machines and Knowledge Base Machines, Kluwer Academic Publishers, Kitsuregawa Ed. (1988).

[KIT 89] Kitsuregawa, M., Yang, W. et al.: Implementation of LSI Sort Chip for Bimodal Sort Memory, Proceedings of the IFIP TC 10/WG 10.5 Int. Conf. on VLSI. Munich, 16-18 August (1989).

[KNU 73] Knuth, D. E.: Sorting and Searching. The Art of Computer Programming, Vol. 3, Addison-Wesley, Reading, Mass. (1973).

[KOJ 87] Kojima, K., Torii, S. and Yoshizumi, S.: IDP—A Main Storage Based Vector Database Processor, Database Machines and Knowledge Base Machines, Kluwer Academic Publischers, Kitsuregawa Ed. (1988).

[LEE 87] Lee, C., Su, S. Y. W. and Lam, H.: Algorithms for Sorting and Sort-Based Database Operations Using a Special-Function Unit, Database Machines and Knowledge Base Machines, Kluwer Academic Publishers, Kitsuregawa Ed. (1988).

[MIR 83] Miranker, G., Tang, L. and Wong, C. K.: A "Zero-Time" VLSI Sorter, IBM J. RES DEVELOP, Vol. 27, No. 2 (1983).

[SAT 89] Satoh, T., Takeda, H. and Tsuda, N.: A Compact Multiway Merge Sorter using VLSI Linerarray Comparators, Proceedings of 3rd Int. Conf., FODO 1989, Foundation of Data Organizations and Algorithms, Springer-Verlag (1989).

[TAN 80] Tanaka, Y., Nozaka, Y. and Masuyama, A.: Pipeline Searching and Sorting Modules as Components of a Data Flow Database Computer, Information Processing 80, North Holland Publishing Company, S. H. Lavington Ed. (1980).

[TAN 83] 田中 譲：データベース処理や文書処理を高速化するサーチ／ソート・ハードウェアの動向，日経エレクトロニクス 1983. 8. 1.

[TAN 84] Tanaka, Y.: Bit-Sliced VLSI Algorithms for Search and Sort, Proceedings of the 10th Int. Conf. on Very Large Data Bases (1984).

[TOD 78] Todd, S.: Algorithm and Hardware for a Merge Sort Using Multiple Processors, IBM J. R & D, 22, 5 (1978).

[YAS 82] 安浦，高木：並列計数法による高速ソーティング回路；電気通信学会論文誌，'82/2, Vol. J 65-D, No. 2 (1982).

(平成元年2月2日受付)