並列ソート算法の分類

ワイズマン・インスティテュート応用数学科
D. Bitton

ウィスコンシン大学計算機科学科
D. J. DeWitt

オハイオ州立大学計算機情報科学科
D. K. Hsiao, J. Menon

訳　野下浩平

細　目　次

† ACM Computing Surveys, Vol. 16, No. 3, pp. 287-318.
"A Taxonomy of Parallel Sorting"

配列とファイルに対する各種のソート算法を調べ，並列ソート算法の分類を試みる．初期の頃のソート用ネットワークから共有メモリ型算法や VLSI ソータに至るまで，並列ソーティングに関する研究がどのように進展してきたかを調べる．

ソート用ネットワークとしては，奇偶マージと双単調マージの基本的な並列マージ法を2つ説明する．これらのマージ法から発展したソート算法で並列計算機のためのものについて議論する．この並列計算機は，完全シャフルや網目状のものなどのように（疎に結合した）ネットワークを通じて通信する処理装置からなっている．ネットワーク型ソート算法に続いて，どのようにして総当りソート方式から高速算法が作られるかを説明する．この方式は，共有メモリ型のモデルを用いて，最初キーの順位を求め，次にその順位に従ってキーを並べ換えるものである．

並列算法の時間計算量と計算機アーキテクチャの実現可能性の両方に関係するいくつかの基準に従って，算法を評価する．ネットワーク型ソート算法は，通信方式が好都合である上，非適応型のものなので，具体化するのに向いているということを示す．特に，この型の算法は，容易に一般化できて，並列性に制約をつけても，大きいソート問題が解けるブロックソート算法に適用することができる．また，ディスク装置に調整を加えたものや知的な磁気バブルメモリを用いて，大容量ファイルを並列的にソートする問題を考える．並列ソート算法の研究方向として，VLSI によるソートが将来性があり盛んになることを結論で触れる．

Categories and Subject Descriptors: B.3[**Hardware**]: Memory Structures; B.4 [**Hardware**]: Input/Output and Data Communications; B.7.1[**Integrated Circuits**]: Types and Design Styles; F.2.2[**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms: Algorithms

Additional Key Words and Phrases: Block sorting, bubble memory, external sorting, hardware sorters, internal sorting, limited parallelism, merging, parallel sorting, sorting networks

はじめに

計算機用語として，ソート（整列，sorting）とは，数の列を上昇順または下降順に並べ換えるプロセスのことであると定義される．コンパイラやエディタなどのプログラムでは，（たとえば，要素の探索や追加のために）記号の列や表を参照する場合，記憶領域内で列や表をソートしておくことにより，その速度を上げたり，あるいは算法の単純化をはかるということがよく行なわれる．ソートは，実用的に重要であり，理論的にも面白く，従来，特にランダムアクセス記憶領域でのソートのための算法（内部ソート算法，internal sorting algorithm）が盛んに研究されてきた．このように，まず逐次的なソート算法が調べられてきたが，その後，並列処理の登場により，並列的なソート算法の研究がきわめて活発になってきた．逐次的な算法としては，n個の数に対して高々$O(n \log n)$回の比較しか必要としないものが数多く知られている．この比較の回数は，理論的な下界でもある［Knuth 1973］．逐次的な内部ソート算法については，これらの時間計算量の他に，いろいろな性質も調べられている．特に，時間と記憶領域（入力のデータの分以外に用いるもの）の大きさの間に成り立つ関係，安定性（同じ大きさの値の順序が入力の順序通りになっていること），値の分布への依存性（特に，最も運の良い場合や悪い場合の計算量）などがソート算法の評価に考慮されてきた．

最近の10年間では，ソート算法は，新しく並列処理の側面からも研究されている．並列計算のモデルがいくつか考えられており，それぞれの流儀で記憶場所の繋がり具合や複数の処理装置の参照方法が定義されている．並列ソートということを明確に述べるには，まず，並列処理装置に対して‘ソートされた列’とはどういうことであるかを定義する必要がある．複数の処理装置が1つの記憶装置を共有している場合には，並列処理装置の連続した記憶場所というものは，逐次処理装置の場合と同様である．それで，逐次的な場合のように，ソート算法の時間計算量は，（複数の処理装置で並列的に実行される）比較とデータの転送の回数で計ることができる．一方，処理装置が記憶装置を共有せず，ネットワークの接続線を通じて通信を行なう場合には，ソートということを定義するのに，処理装置の順序とそれぞれの記憶場所の順序に関する約束が必要になる．このような並列処理装置を用いる場合には，ソート算法の時間計算量は，並列的比較回数およびネットワークで隣り合う処理装置間のデータの交換回数によって定義される．

共有の記憶領域をもつ並列計算モデル，すなわち共有メモリ型の並列計算モデルは，ソートにおける本質的な並列性を研究するための道具として用いられてきている．並列ソート算法に関する最初の頃の結果は，ソート用ネットワークに関するものであったが［Batcher 1968］，もっと速い並列ソート算法は，共有メモリ型処理装置の理論的モデルに対するものである［Hirschberg 1978; Preparata 1978］．この共有メモリ型の計算に関する一連の研究によって，並列ソートについても，$O(\log n)$時間計算量をもつ並列ソート算法がいくつか開発されている．共有メモリ型ソート算法とは，n個の数が与えられて，1つの大きな記憶領域を共有するn個以上の処理装置を用いてソートするものであるといえるが，ここで，記憶場所の参照にはいろいろな自由度が考えられる．たとえば，同時に読出しができるか，あるいは同時に書き込むのにどう制限を付けるかというものである．並列ソート算法は，たいてい純理論的なものとして研究されてきたが，ようやく最近になって，並列性に制約を設けたり，あるいは，VLSI に関連して，（チップ面積として表わされる）ハードウェア量と時間計算量の関係を調べるというように，その実現可能性が注目されはじめている．

一般に，ソートは，単に記憶領域内の数を並べ換えるというだけでなく，もっと広くデータ処理に用いるものとしてその重要さが強調されることも多い．そのような場合には，ソート算法は，大容量記憶装置内のレコードのファイルを並べ換えるものとして用いられる．レコードは，キー（key）の値に従って並べられる．キーは，ひとまとまりのデータの場合もあるし，いくつかのデータを繋いだ長いものである場合もある．ファイルはソートされて，（電話帳のように）整った形式で外部に出力されたり，あるいはデータベースでの複雑な操作の中間結果として扱われる［Bitton と DeWitt 1983; Selinger 他 1979］．このようなファイルのソートは，記憶領域の大きさからみて，主記憶領域内ではできないので，外部ソート算法（external sorting algorithm）が必要になる．外部ソートは，マージ（併合，merge）を繰り返して行なわれるのが普通である［Knuth 1973, 5.4節］．高速のディスク装置を大容量記憶装置として用いる場合でも，外部ソートでは，入出力が実行時間の主要部分になっている†．

巨大なファイルを高速にソートすることの必要性は明らかであるが，並列処理を使った外部ソート算法の研究はあまりされていない．並列外部ソートの研究が相対的にみて少ないのは［Bitton-Friedland 1982; Even 1974］，そのような方法を記憶装置の特性に合わせる必要があるということによるものと思われる．

計算機の技術が進んで，たとえば，知的メモリや連想メモリが使えるようになれば，他の操作を行なうための道具としてソート算法を用いることが減ったり，あるいはなくなったりするかもしれない．たとえば，探索を簡単にするためにソートするということは，連想記憶を用いれば，不要になるといえるかもしれない．しかしながら，連想メモリは，広く使われるようになるには，まだ値段が高すぎる．特に，データの量が巨大な場合には，一層そうである．ソートがデータを並べるということにのみ用いる場合には，ソート時間を減らす唯一の道は，高速の並列ソートの方法を開発することである．それは，おそらく，ソートの機能が大容量記憶装置の中に組み込まれるというものであろう［Chen 他 1978; Chung 他 1980］．

この論文では，内部と外部のソート算法の両方をふくめて，並列ソート算法の分類を行なおう．並列ソートの研究が，最初の頃のソート用ネットワークから共有メモリ型モデルの算法や VLSI ソータまでどのように進んできたかを調べよう．そして，時間的な能率やアーキテクチャに対する条件をふくむいろいろな基準によって，多岐にわたる並列ソート算法を分類していこう．この論文の目的は，並列ソート算法に関する研究成果に対する基本的な知識と統一的な見方を与えることである．もちろん，従来提案されてきた計算モデルを詳しく紹介し，算法の計算量を深く解析するのは，一編の論文でできることではなかろう．それで，計算量に関する議論は最少限にとどめ，並列的な比較回数の上界に関する主な結果を示すことだけにする．そして，並列ソートに関する理論的な問題（たとえば Borodin と Hopcroft［1982］，Shiloach と Vishkin［1981］，Valiant［1975］で詳しく

† IBM システムにおいて，あらゆる入出力のための時間のうち，OS/VS Sort/Merge プログラムは，その25%程度使っているものと見積られている［Bryant 1980］．

扱われているようなもの）というより，むしろ，現在あるいは近い将来の技術で実現できる並列ソート算法に関する問題を主として取り上げる．

本論文の構成は，次のとおりである．第1節では，高速の逐次的なソート算法で並列化できるものを示す．しかし，このやり方では，単純であるがあまり速い並列算法は作れない．第2節では，ネットワークによるソート算法を取り扱う．特に，Batcher の双単調（bitonic）法を実現するネットワークをいくつか詳しく述べる．第3節では，非常に高速のソート算法に関する一連の研究結果を紹介する．それらは，共有メモリ型モデルの並列マージ算法［Gavril 1975; Valiant 1975］や共有メモリ型モデルのソート算法［Hirschberg 1978; Preparata 1978］である．第4節では，並列性に制限をつけたものを調べる．ここでは，p個の処理装置で$M \cdot p$個の要素をソートするブロックソートの方法を説明する．そして，ブロックソート算法を導き出す2通りの考え方を示す．第5節では，巨大ファイルを並列的にソートする問題を取り上げる．第4節までの並列ソートに対するたいていの結果は，ソートする配列全体が主記憶領域に納まる内部ソートの方法に対して適用されるものであり，外部ソートのための並列算法は，今後の研究課題であるということを指摘する．第6節は，最近提案されているソート専用機械の設計について調べる．第7節では，この論文をまとめるとともに，今後の研究方向を示す．

1. 逐次ソート算法の並列化

並列計算によれば，単位時間に2回以上の比較を行なうことができる．並列計算のある種のモデルでは，（特にソート用ネットワークでは，）単位時間におのおののキーが他のキー1つと比較するものと仮定する．ここで，並列性が利用できるのは，キーの対を複数個同時に比較できる点である．他のモデルとしては，1つのキーと数多くのキーを同時に比較するというものもある．たとえば，Muller と Preparata［1975］では，$(n-1)$個の処理装置を用いて，単位時間に1つのキーを他の$(n-1)$個のキーと比較している．

また，並列計算によれば，多くのキーを同時に転送できる．並列的な比較のステップの後に，処理装置は，データを交換する．この交換ステップで行なわれる並列的な操作は，処理装置の接続方式とか（共有メモリ方式の場合には）記憶の競合とかによって制約を受ける．

単一の処理装置の記憶領域内での比較と転送のステップと同様にして，並列処理装置では，キーの対複数個に対する並列的な比較・交換を行なうので，並列ソート算法の能率の評価は，この比較・交換の回数で行なうのが自然である．並列ソート算法による速度向上の効果は，最適の逐次ソート算法での比較・転送回数に対するその並列ソート算法の比較・交換回数の割合で計ることができる．

比較に基づいてソートする逐次算法では，n個の要素のソートに$O(n \log n)$回の比較回数が必要にして十分であるので［Knuth 1973, p. 183］，n個の処理装置を用いて，速度向上を最もうまく行なうと，$O(\log n)$回の並列的比較まで減らせることが期待される†．しかしながら，この上界は，$O(n \log n)$時間の逐次算法でよく知られているものを並列化しても達成できそうにない．これらの算法には，逐次的な制約を緩める余地がなさそうである．たとえば，複線マージ法（two-way merge sort）［Knuth 1973, p. 160］を考えてみよう．この算法は，$\log n$回の繰返しからできている．各繰返しごとに，（前の繰返しまでで作られた）ソートされている列の対がもっと長い列にマージされる．はじめのほうでは，処理装置を数多く用いて，数多くの対を並列的に併合できる．しかし，後のほうでは，並列性をよく利用できる方法がありそうにない．特に，最終回の繰返しは，$n/2$個の要素でなる列の対をマージするが，これは，$n-1$回の比較が必要となる逐次的な操作となる．

一方，$O(n^2)$回の比較を要する直接的なソート算法では，並列化が簡単であろう．しかし，このやり方では，$O(n)$個の処理装置を用いても，せいぜい$O(n)$時間の並列算法しか作れない．それは，単位時間にn回しか比較できないので，全実行時間は$O(n^2)$を$O(n)$にしかできないからである．この種の並列化の例としては，普通のバブル法（bubble sort）の並列版で奇偶転置法（odd-even transposition sort）とよばれるものがある（1.1節）．

また，高速の逐次算法を部分的に並列化して，$O(n)$の並列算法を作ることができる．その例としては，木選択法（tree selection sort）を調整して，木の同一レベルでの比較を並列的に行なうようにしたものである．この並列木選択法は，1.2節で説明する．この並列算法は，データベース用 Tree Machine で用いられている［Bentley と Kung 1979］．

† （訳注）本論文を通して，O-記法は，必ずしも上界のみを表わすように用いられていない．$O(f(n))$という書き方は，'$f(n)$に比例する'という程度の意味で解釈するものであろう．

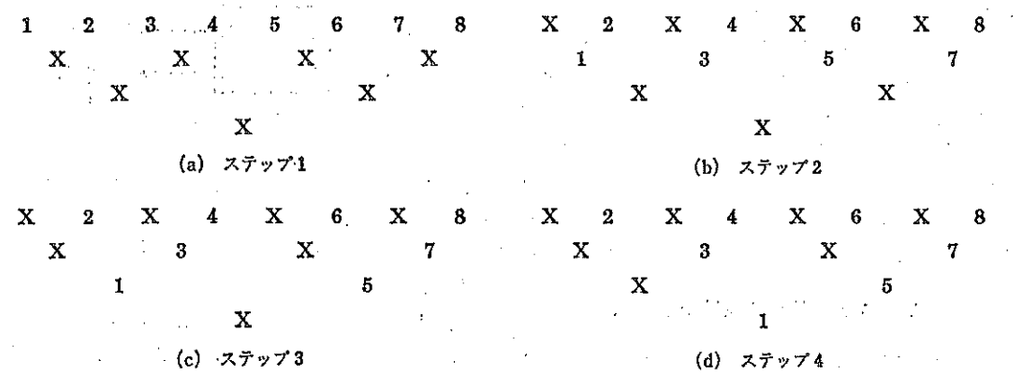


図1　並列木選択法

1.1 奇偶転置ソート法

逐次的"バブル法"は，隣り合う要素を比較して交換することを繰り返すものである．配列$(x_1, x_2, \cdots, x_n)$をソートするには，$(x_1, x_2), (x_2, x_3), \cdots, (x_{n-1}, x_n)$に対して$(n-1)$回の比較・交換を行なう．これによって，配列の右端に最大値がくる．この第1ステップにより，x_nを除外することができ，比較・交換による同様の'バブル'列を要素1つ分小さい列$(x_1, x_2, \cdots, x_{n-1})$に適用する．こうして，$(n-1)$回の繰返しにより，全体がソートされる．

逐次的奇偶転置法［Knuth 1973］は，上述の基本的なバブル法の変形である．これは，$n/2$回の比較でなるステップをn回繰り返す．そして，奇数回目と偶数回目のステップは，おのおの次のように行なう．

奇数回目のステップでは，奇数番目の要素がおのおの右隣りの要素と比較する．つまり$(x_1, x_2), (x_3, x_4), \cdots$という比較を行なう．偶数回目のステップでは，$(x_2, x_3), (x_4, x_5), \cdots$というように偶数番目の要素がおのおの右隣りと比較する．ソートが終了するまでには，奇偶交互にnステップを繰り返す［Knuth 1973, p. 65］．

この算法は，直接並列化できる［Baudet と Stevenson 1978］．1列に接続した処理装置をn個用意して，それらを$P_1, P_2, \cdots, P_n$とよぶ．処理装置は両方向に接続して，P_iがP_{i-1}とP_{i+1}の両方と通信できるようにする．最初，要素x_iをP_iにおく$(i=1, 2, \cdots, n)$．列$(x_1, \cdots, x_n)$を並列的にソートするには，奇数番目のステップにおいて，$P_1, P_3, P_5, \cdots$を働かせて，逐次的な奇偶転置法の奇数番目の操作を並列的に行ない，偶数番目のステップにおいては，$P_2, P_4, P_6, \cdots$に対して同様に偶数番目の操作を行なえばよい．

1回の比較・交換には，2回のデータの転送が必要である．たとえば，第1ステップでは，x_2がP_1に転送されて，x_1と比較される．ここで，$x_1 > x_2$であれば，x_1がP_2に転送される．$x_1 < x_2$であれば，x_2がP_2に戻される．こうして，並列奇偶転置法では，n個の処理装置でn個の数をソートするのに，n回の（並列的）比較と$2n$回の転送が必要である．

1.2 木選択ソート法の並列版

逐次的木選択法では，$(2n-1)$個の節点をもつ2分木のデータ構造を用いる．この木は，n個の葉をもち，最初は，各葉に入力の数を1つずついれておく．ソートのやり方は，n個の数の最小値を選択し，次に，残りの$n-1$個の数の最小値を選択し，以下同様，というものである．

2分木構造を用いて，兄弟の2つの数を比較して，小さいほうの数をその親に送るという操作を繰り返すことによって最小値を求めることができる（図1をみよ）．2分木の同じレベルにあるものをすべて並列的に比較するのが並列的木選択法である［Bentley と Kung 1979］．

処理装置を$(2n-1)$個用意して，n個の葉の節点と$(n-1)$個の内部節点おのおのに1つずつ処理装置を割り当て，2分木の形に接続する．葉の処理装置からはじめて，$\log_2 n$回の並列的比較と転送のステップを繰り返せば，最小値を根の処理装置まで送ることができる．各ステップにおいて，親が2つの子から要素を受け取り，比較し，小さい要素を残して，大きい要素を戻す．最小値が根に到着すると，それを外へ書き出す．一般に，空の（つまり要素を親に送った）処理装置は，空でない子からデータを受け取り，2つのデータを受け取った場合に小さいほうを選択するというように働く．各ステップごとに，上昇順にみて順次要素が根に到着するので，ソ

ートは，$O(n)$ 時間で終了する．

奇偶転置法も木選択法も簡単な並列算法であり，おのおの対応する逐次算法の各段階で行なう比較の列を並列的に行なうという考え方で作られている．どちらも，任意の n 個の数に対して，$O(n)$ 個の処理装置を用いて，$O(n)$ 回の比較でソートする．後でみることであるが，逐次的なソート算法を並列化したものよりも，ソートの並列的性質を直接利用して作った並列算法のほうが速い．

2. ネットワーク型ソート算法

多入力多出力のスイッチ・ネットワークの設計というハードウェアの問題が，並列ソート算法を生むもとになったということはちょっと面白い．並列ソートに関する最初の頃の結果は，ソート用ネットワークに関する文献にでている [Batcher 1968; Van Voorhis 1971]．その頃以後，接続の仕方に応じていろいろなネットワークが提案されており，それによってソートを高速に行なうことが詳しく研究されている．2.1 節では，奇偶ネットワークと双単調ネットワークを説明する．2.2 節では，SIMD(単一命令多データ制御式) 計算機での並列ソート算法が双単調ネットワークより作り出せることを示す．特に，網目状に接続された処理装置に対する双単調算法を 2 種類説明する [Nassimi と Sahni 1979; Thompson と Kung 1977]．他の型のネットワークも同様に重要である．特に，超立体格子的な接続をもつものは [Pease 1977; Preparata と Vuillemin 1979]，数値計算問題だけでなく，ソートにもうまく応用できる．双単調マージに基づくソートは，この種のネットワークにおいて経路の選択法を定めることで実現できる．これらのネットワークについて深く検討することは，この論文の範囲をこえているが，経路の選択法を定めるための基本的なマージの方法は詳しく説明しよう．そして，Batcher のネットワークでのソート時間に関する下界 $O(\log^2 n)$ を導くことにする．

2.1 ソート用ネットワーク

ソート用ネットワークは，もともと高速で値段の安いスイッチ・ネットワークとして考え出された．n 入力のソート用ネットワークでは，$(1, 2, \cdots, n)$ の順列の並べ換えを行なうので，それを多入力多出力のスイッチ・ネットワークとして用いることができる [Batcher 1968]．比較器 (comparator) から構成されるネットワークによって逐次的なソート算法を実現しようとすると，比較器を直列的に繋ぐことになり，したがって時間遅れも大きくなる．それで，高速のソート用ネットワークは，比較を並列的に行なう比較器モジュールから構成しなければならず，その設計のためには，並列的なソート算法が必要となる．

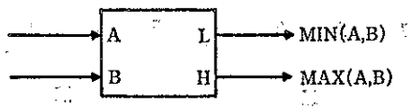


図2　比較交換器

並列ソートに関する結果の中で，Batcher [1968] のものは，最初の頃のものである．Batcher は，2 つの方法を考えたが，いずれも n 個のキーのソートに $O(n\log^2 n)$ 個の比較器を使い，$O(\log^2 n)$ 時間ですませるものである．図 2 に示すように，比較器は，2 つの入力線 A，B をもち，出力線 L から小さいほうの値，出力線 H から大きいほうの値をそれぞれ出力するようになっている．逐次的な比較器は，A，B から最上位ビットから順に入力するものであり，少数の論理ゲートで実現できる．並列的な比較器は，いくつかのビットを同時に入力するもので，速くなるが，複雑な構成になる．奇偶法と双単調法とよばれる Batcher の算法はいずれも，マージを繰り返すという考え方によっている．繰返し方を詳しくいうと，2^k 個の入力列に対して，長さ $2, 4, 8, \cdots, 2^k$ のソートされた列を順次作っていくというものである．

2.1.1 奇偶マージ方式

奇偶マージ (odd-even merge) 法における繰返し方は，図 3 に示している．あらかじめソートされた列 2 つ $(a_1, a_2, \cdots, a_n)$, $(b_1, b_2, \cdots, b_n)$ から次のような 2 つの列を作る．おのおのの列から奇数番目のものでなる列と偶数番目のものでなる列を取り出し，それぞれマージする．すなわち，奇列というもの $(c_1, c_2, \cdots)$ は，奇数番目の列 $(a_1, a_3, \cdots)$ と $(b_1, b_3, \cdots)$ をマージしたものであり，偶列というもの $(d_1, d_2, \cdots)$ は，偶数番目の列 $(a_2, a_4, \cdots)$ と $(b_2, b_4, \cdots)$ をマージしたものである．次に，奇列 $(c_1, c_2, \cdots)$ と偶列 $(d_1, d_2, \cdots)$ を次のような比較・交換によりマージして，列 $(e_1, e_2, \cdots, e_{2n})$ を作る．

$$e_1 = c_1$$
$$e_{2i} = \min(c_{i+1}, d_i)$$
$$e_{2i+1} = \max(c_{i+1}, d_i) \qquad i = 1, 2, \cdots$$
$$e_{2n} = d_n$$

こうしてできた列は，実際にソートされた列になっている (証明は Knuth [1973, pp. 224, 225] をみよ)．この奇偶マージの考え方を用いて，2^k 個の数をソートするには，2^{k-1} 個の (長さ 1 の対用) マージ・ネットワーク (比較・交換器)，その後に，2^{k-2} 個の (長さ 2 の対用) のもの，その後に 2^{k-3} 個の (長さ 4 の対用) のものという具合に並べる．長さ 2^{i+1} の列の対用マージ・ネットワークには，長さ 2^i の列の対用のものより比較・交換が 1 回余分に必要であるので，入力は，高々 $1+2+3+\cdots+k = k(k+1)/2$ 個の比較器を通過するだけであることがわかる．このことより，2^k 個の数のソートには，$k(k+1)/2$ 回の比較・交換が実行されることになる．しかしながら，このネットワークでは，比較器が $(k^2-k+4)\,2^{k-2}-1$ 個必要である [Batcher 1968]．

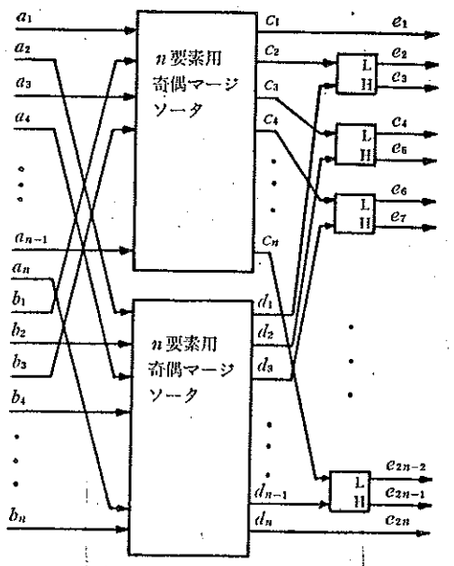


図3　奇偶マージの繰返し方

2.1.2 双単調マージ法

双単調法は，繰返し方が異なっている (図 4)．双単調列 (bitonic sequence) とは，一方が上昇順で他方が下降順の 2 つの列を繋いでできる列である．さらに，この列を巡回シフトしたものも双単調列という．双単調法の繰返し方は，次の考え方に基づいている：双単調列は，比較・交換を 1 段行なえば，2 つの双単調列に分割することができる．いま，$(a_1, a_2, \cdots, a_{2n})$ が $a_1 \le a_2 \le \cdots \le a_n, a_{n+1} \ge a_{n+2} \ge \cdots \ge a_{2n}$ である双単調列とすれば†，次の 2 つの列

$$\min(a_1, a_{n+1}), \min(a_2, a_{n+2}), \cdots$$
$$\max(a_1, a_{n+1}), \max(a_2, a_{n+2}), \cdots$$

はいずれも双単調列になっている．さらに，前者の列は，もとの列の小さいほうの n 個，後者のほうは大きいほうの n 個でなっている．それで，双単調列をソートするには，その列の半分の長さの双単調列をおのおの別々にソートすればよいことがわかる．

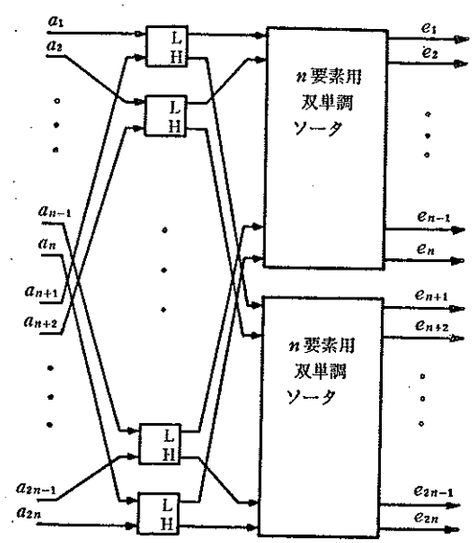


図4　双単調マージの繰返し方

この双単調法の繰返し方に従って，2^k 個の数をソートするには，ソートとマージを繰り返し，だんだん大きい列を作り，最後に 2^k 個の数でなる双単調列を作り，この双単調列を分割して，下半分の部分列と上半分の部分列を作る．双単調列を繰り返し作り上げる部分には，(双単調列の下降順の部分のために，出力線を入れ換え，下降順に数の対を出力する) ような比較器が必要である．図 5 には，8 入力の双単調ネットワークを示している．一般に，2^k 個の数に対する双単調法では，$k(k+1)/2$ ステップかかり，各ステップに 2^{k-1} 個の比較器を使う．

最初の双単調ソータが発表された後，同様のソートの考え方であるが，$n/2$ 個しか比較器を使わないもので完全シャフル (perfect shuffle) の繋ぎ方によるものが示された [Stone 1971]．いま入力の値に 2 進数のラベルをつければ，双単調ソートのどの段階でも比較器に入るキーの対は，そのラベルが 2 進表現で 1 ビットだけ異なっていることがわかる．さらに，ネットワークは $\log n$ 個の段階からなり，第 i 段階は，i 個のステップでなり，各ステップでは，(巡回シフトによってできる) 最下位ビットの異なる入力が比較されることがわかる．このような双単調ソータの規則性からわかることは，隣り合う段

† 上昇列と下降列が同じ長さに限らないほうがよいということが査読者によって指摘されたが，双単調マージを理解するのがやさしいので，この仮定をおいておく．

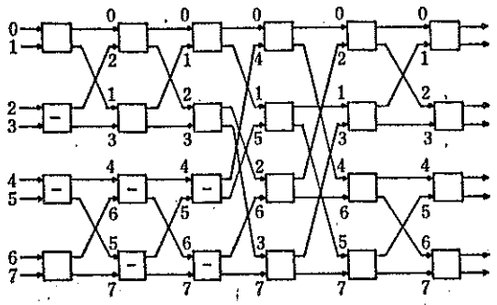


図5　Batcher の8入力用双単調法
（負号のついた箱は出力線を
入れ換える比較器を示す）

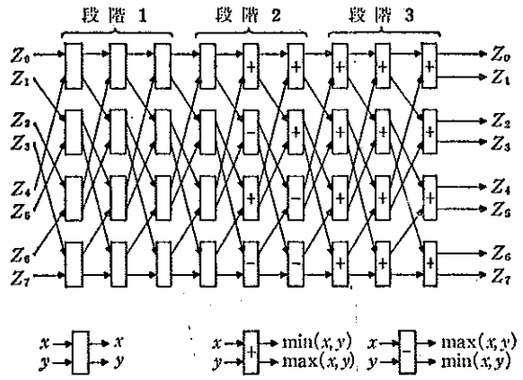


図6　Stone の双単調法（負号のついた箱は出力線
を入れ換える比較器を示す）

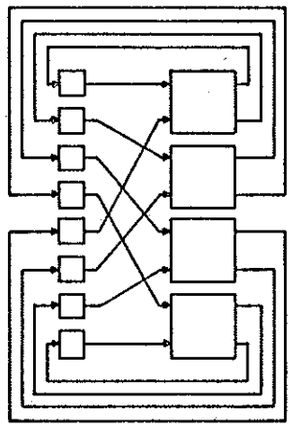


図7　Stone による双単調法のアーキテクチャ

にある比較器どうしの繋ぎ方を同じ構造にすることができそうである．Stone は，完全シャフルの繋ぎ方がネットワークのすべての段階で適用できるということを示した．入力線の‘シャフル’は，（トランプのシャフルに似たもので，）その2進数表現を左へ巡回シフトすることに等しい．シャフルを2回すれば，おのおのの2進表現が2ビット巡回シフトする．それで，比較・交換の各ステップの前に，双単調法で必要な回数だけシャフルすることにより，入力線を並べることができる．この考え方を実現するネットワークで8入力に対するものを図6に示す．一般に，$n=2^k$ 個の入力線に対して，この型の双単調ソータは，$(n/2)$ 個の比較器を $(\log n)^2$ 段に配置して，全体として，$(n/2)(\log n)^2$ 個の比較器を使う．ネットワークは，$\log n$ 段階でなっていて，各段階は，$\log n$ ステップでなる．各ステップでは，出力線がシャフルされて，次の段の比較器に入る．第 i 段階のはじめの $(\log n)-i$ ステップでは，比較器は入力の交換に使われず，入力のシャフルにのみ使われる．

このような多段階のネットワークとは異なる方式として，双単調法を循環式のネットワークとして実現するものがある．これによれば，比較器の数を大幅に減らすことができる．たとえば，図7に示すように，シフトレジスタとシャフルの接続でなる段階が1つのものとして，双単調ソータが実現できる．双単調ソート法の第 i 段階では，i 回の比較・交換がいるので，Batcher 法では，

$$1+2+\cdots+\log n=\log n\,(\log n+1)/2$$

段の並列的比較・交換が必要であったが，Stone の双単調ソータでは，全体で $(\log n)^2$ 段必要である．これは，（比較しないで，）単に入力線のシャフルのための余分のステップが必要であるからである．こうして，逐次ソートでは $O(n\log n)$ の計算量であったものが，並列ソートでは $O(n/\log n)$ 倍の速度向上が得られたことになる．それで，この算法によって，n 個の処理装置で n 個の数をソートするのに，$O(n)$ 時間かかっていたものよりはるかに速くできることになった．

Siegel によれば，双単調法は，他の種類のネットワークでも $O(\log^2 n)$ 時間できる［Siegel 1977］．Siegel の考えたものには，超立体格子や Plus-Minus 2^i ネットワークなどがある．ここでのポイントは，双単調法で必要なデータの交換がこのようなネットワークでも実現できるという点である．（実際に，完全シャフルは，超立方格子を模倣するものとみることができる．）Siegel は，ネットワークの繋ぎ方のある一般的なクラスに対して，シャフルを模倣するには，$O(\log^2 n)$ 時間かかり，そしてソートもこの時間内で実行できるということを示した．最後に，融通性の高い CCC (cube-connected-cycle) に触れておく．これは，超立方格子を能率的に模倣するネットワークであるが，処理装置当り3本の通信線しか必要としない［Preparata と Vuillemin 1979］．CCC を用いれば，双単調法も奇偶マージ法も $O(\log^2 n)$ 時間で実行できる．

2.2 SIMD 機械のソート算法

ソート用ネットワークの特徴は，非適応性である．すなわち，比較する順序は最初から固定されており，途中の比較の結果に依存しない．言い換えれば，2つのキー R_i, R_j が比較されるとすると，以後の R_i に対する比較は，$R_i<R_j$ の場合も $R_i>R_j$ の場合もまったく同一である．この非適応性より，ネットワーク型のソート算法は，SIMD 機械でうまく実現できる．SIMD 機械（単一命令流れ・多データ流れ）とは，1つの制御装置があり，その他に，それぞれの専用メモリをもつ処理装置のいくつかがネットワークで接続されているシステムである．処理装置は完全に同期的に働く．制御装置により，命令が伝達されて，すべての処理装置（で活動中のもの）により同時に実行される．（マスクを用いて，命令サイクルの間，処理装置のうちいくつかを休止させておくこともできる．）ネットワークによるソート算法で実行される比較と転送の列は，ソートの開始時点で決定されるので，中央制御装置は，各ステップで相応しい比較・交換命令を各処理装置に伝達することによって，算法の実行をつかさどることができる．

2.2.1 アレイ処理装置によるソート

SIMD 機械では，ソートすることは，処理装置の専用メモリの間でデータを並べ換えることと考えることもできる．特に，網目状に接続された処理装置それぞれが専用メモリの中に1つの数を貯えておくものとすると，ソートは，隣り合う処理装置に貯えた数を交換して，最終的に，処理装置のある順序に合わせるように並べるプロセスとみることができる．$n\times n$ 個の処理装置が網目状に接続されている場合，たとえば，行優先あるいは列優先というような一定のやり方で番号をふることができる．このやり方は，配列の添字を定めるのに普通よく行なわれるものである．Thompson と Kung［1977］によって，双単調法が網目状接続のシステムに適用されている．これには，3つの番号のふり方が考えられている：行優先のもの，ジグザグ行優先のもの，シャフル行優先のもの．これらの例は，図8に示されている．いま，n^2 個のキーが任意に与えられていて，それらが各処理装置にちょうど1つずつばらまかれているものとすると，ソートは，i 番目の処理装置に小さいものから i 番目のものを移すということになる $(i=1,2,\cdots,n^2)$．ソート用ネットワークと同様に，並列性の使い方は，数の対いくつかを同時に比較することであるが，どの時点でも，どの数も別の数1つと比較されるだけである．データの転送は同時に行なえるが，すべて同一の方向に限る．つまり，すべての処理装置は，転送レジスタの内容を上下左右のいずれかの方向に同時に転送する．このような計算モデルは SIMD であるといえる．というのは，各時刻で，1つの命令（比較または転送）のみが伝達されて，その命令に従って，処理装置が一斉に同時に実行するからである．この方式でソート問題を解くための計算量は，比較回数と単位距離で測った経路決定のステップによって定められる．この節では以下，経路決定の単位距離を転送の単位にとる．このモデルにおいて，入力を並べ換えるどんな算法でも少なくとも $4(n-1)$ 回の転送が必要である．なぜならば，アレイ処理装置では最も遠い2つの隅にある処理装置の間で数を交換しなければいけない場合があるからである．

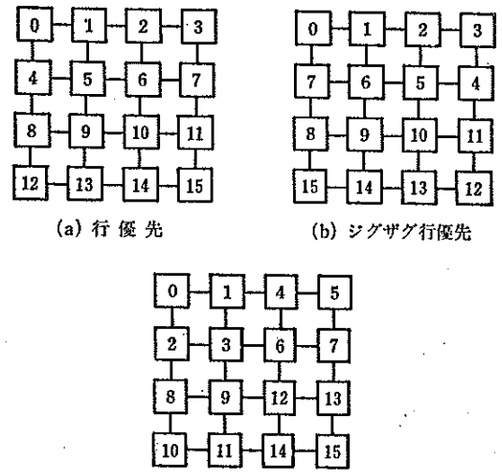


図8　アレイ処理装置の順序付け方式

奇偶法も双単調法もここでの並列計算モデルに適用されている．これらによれば，$O(n)$ 回の比較と転送でソートが実行できる［Thompson と Kung 1977］．前者の算法では，2次元配列の奇偶マージを用いて，ジグザグ行優先の番号付けでキーを並べる．後者の算法では，双単調法を用いて，シャフル行優先の番号付けでキーを並べる．同様の計算量をもつ3つ目の算法で行優先の番号

付けのものは，後で発表されている[Nassimi と Sahni 1979]．この算法も双単調法を適用したもので，繰返し方に2次元配列のマージを用いている．最後に，2次元の奇偶マージの改良版が最近提案されている[Kumar と Hirschberg 1983]．このマージのやり方をもとにして，2次元配列が行優先の順序に $O(n)$ 時間でソートできる．これは，前に述べた算法に比べて比例定数がもっと小さくなっている．

2.3 まとめ

この節では，奇偶マージ法と双単調法というよく知られているソート用ネットワークを2種類調べ，ソート用ネットワークの考え方が同期的な並列ソートのいろいろな方式に拡張できることを示した．ハードウェア量についていくらかは考えたが，主に，ネットワークの計算量として，実行時間と処理装置の個数を取り上げた．つまり，ネットワークで使われるソートの方式を評価するのに，基本的には，比較・交換の回数を用いており，ソート算法の計算量として，ネットワークの接続の複雑さは特に系統的には取り扱わなかった．ネットワークの接続方式を包括的に解析することは，ここでの守備範囲をこえている．この話題については，文献が豊富にあるので，関心をもたれた読者のためにそのいくつかを挙げておく[Feng 1981; Nassimi と Sahni 1982; Pease 1977; Preparata と Vuillemin 1979; Siegel 1977, 1979; Thompson 1980]．

ごく最近までは，最も良い計算量のもつソート用ネットワークは，$O(n\log^2 n)$ 個の比較器を使い，$O(\log^2 n)$ 時間のものであった．本稿でも，双単調法によれば，n 個の数のソートが $n/2$ 個の処理装置で $O(\log^2 n)$ 時間でできることを示した．第3節では，もっと速い並列ソート算法を開発する目的で，ネットワークの比較器よりももっと融通のきく並列計算モデル——共有メモリ型モデル——を調べる．しかしながら，最近の理論的な結果[Ajtai 他 1983]によって，ネットワーク型のソート算法への関心が再び高まっている．この結果によれば，n 個の数のソートが $O(\log n)$ 回の比較ですむ．残念ながら，この算法は，具体的に実現するのに向いていない．これはあるグラフの複雑な構成に基づいており，(比較回数の下界に対する) 比例定数が救いがたく大きい．

3. 共有メモリ型ソート算法

ネットワーク型算法で $O(\log^2 n)$ 時間の上界を得たので，当時，研究者は，次に理論的な下界 $O(\log n)$ にどこまで近づけるかを調べはじめた．この節では，$O(\log n)$ 回の比較で n 個の数をソートする算法をいくつか説明する．これらの算法では，共有メモリをもつ並列計算モデルを仮定する．

ネットワーク型の算法は，要素の対に対する比較・交換に基づいているが，一般的にいって，共有メモリ型算法では，要素の順位を計算する総当り (enumeration) 方式によっている．ソーティングの仕方は，まず並列的に要素それぞれの順位を計算して，次に，その順位に従って，各要素を望みの場所に転送するというものである．ネットワーク型算法では，個々の処理装置がすぐ隣りのものと連絡をとって局所的に転送経路を決定しているが，共有メモリ型算法では，どの時点においても，どの処理装置も全体で共有しているメモリのどの場所も参照できる．第2節で示したように，ネットワーク型算法では，接続に疎なものを用いており，接続の形の差異しかなかった．共有メモリ型算法では，読出しと書込みの競合を許すか否か，あるいはどのような競合を許すかといった計算モデルの設定によって差異がでてくる[Borodin と Hopcroft 1982]．明らかに，共有メモリ型モデルのほうがより強力である．しかしながら，現在のところ，このモデルは，主として理論的な関心から調べられているにすぎない．ネットワーク型モデルは，現在あるいは近い将来の技術で実現可能なものである．

この節では以下，$O(n^2)$ 個の処理装置を用いて，総当りによるソート法でネットワーク型のものを変形したものを説明する[Muller と Preparata 1975]．その後で，(第2節での奇偶マージや双単調マージのような) 非適応型のネットワーク型の算法より速い並列マージ法を2つ調べ，そして，それらに総当りの考え方を組み合わせたソート算法をみていこう[Preparata 1978]．これら総当りソート算法の他に，並列バケツソート算法も説明する[Hirschberg 1978]．

3.1 ソート用ネットワークの変形版

Muller と Preparata[1975]は，ソートに要する比較回数を減らすのに，$O(n)$ 以上の並列性を持ち込むことによって，これまでとは種類が違う比較器を用いるネットワークの変形版を発表した(図9)．これらの比較器には，2つの入力線と1つの出力線がある．入力線AとBに2つの数を受け取り，出力線には，A＜Bなら0，A＞Bなら1を出力する．n 要素のソートには，全部で $n(n-1)$ 個の比較器を用いて，要素それぞれが他のすべての要素と同時に比較する．次に，この比較器からの出力ビットを並列カウンタに送り，各要素の順位を $\log n$ ステップで計算する．これには，各要素と他の $(n-1)$ 個の要素の比較で求められた1の個数を勘定してやればよい．最後に，二又の枝分れスイッチを $(\log n)+1$ 段二分木状に並べてできるスイッチ・ネットワークを用いて，順位が i の要素を i 番目の端子に転送する．スイッチ・ネットワークの2分木1つ当り $(2n-1)$ 個のスイッチが必要であり，各要素ごとにこの2分木を用意することになる．この2分木を通して要素を転送する時間は，$\log n$ であり，これが算法全体の計算量を決定する．ハードウェア量を増やし，$O(n^2)$ 個の処理装置を用いているが，$O(\log n)$ 時間でソートできることになった．

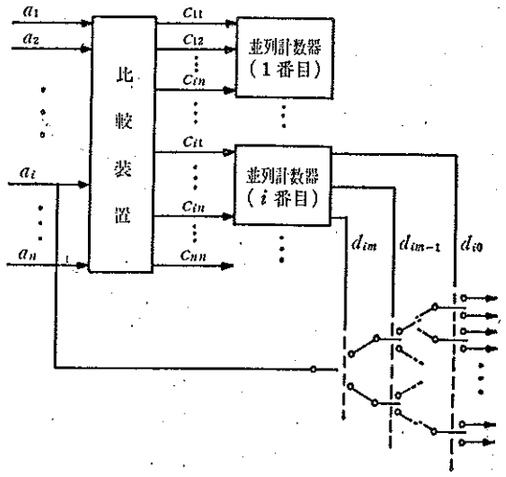


図9　Muller-Preparata のソート用ネットワーク[1975]

Muller-Preparata 算法は，並列ソートに総当り方式を使った最初のものである．総当りによるソートの考え方は，他の高速並列ソート算法にも利用されている[Hirschberg 1978; Preparata 1978]．この結果によれば，Muller-Preparata の結果より処理装置の個数が改善されている．$O(\log n)$ 時間のソートに n^2 個処理装置を用いるのでは，理論的な観点からも不十分であろう．完全に並列的に働く高々 $O(n)$ 個の処理装置を用いれば，$O(\log n)$ 時間でソートを行なうことは，理論的には可能であると思われる．

3.2 高速並列マージ算法

並列ソート算法で最適のものを見つけるには，総当りの考え方の他，高速マージ算法も利用できるかもしれない．Valiant[1975]は，比較問題における並列性を研究して，高速マージ算法を発表した．それは，長さ n と m のソートされた列2本をマージするのに，mn 個の処理装置を使って，$2\log(\log n)+O(1)$ 回の比較でできるという再帰的な算法である．双単調マージでは，$\log n$ 回の比較を要することに注意せよ．また，Gavril[1975]は，長さ n と m のソートされた列2本のマージに p 個 ($p\leq n\leq m$) しか処理装置を使わない高速マージ算法を提案した．この算法は，2分挿入法に基づいており，$n=m$ の場合には，わずか $2\log(n+1)+4(n/p)$ 回の比較しか実行されない．

Valiant と Gavril の両方のマージ算法では共有メモリ型計算モデルを仮定している．このモデルでは，すべての処理装置が入力データも途中の結果も同時に参照できる．

3.3 バケツソート法

Hirschberg のバケツソート法 (bucket sort) は，n 個の数のソートが n 個の処理装置を用いて，$O(\log n)$ 時間でできるものである．ただし，入力の数は，$\{0,1,\cdots,m-1\}$ からとられるものとする．この算法では，副作用として，入力列に重複して現われている数がソートの間に除去されてしまう．メモリの競合のことを無視できるとすると，逐次バケツ法を直接的に並列化できよう．すなわち，m 個バケツを用意しておいて，各処理装置に1つずつ入力の数を割り当てれば十分であろう．i 番目の数を受け取る処理装置を P_i とすると，P_i の仕事は，その数に相応しいバケツに番号 i を入れることである．たとえば，P_3 が数5を受け取れば，5番目のバケツに数3を入れる．この単純な解法には，問題がある．つまり，同じバケツにいろいろな i の値を同時に入れようとすると，メモリの衝突が起こってしまう．

メモリの衝突の問題は，メモリ量をうんと増やしてやれば解決できる．いま，大きさ n の配列を m 本用意できるものとしよう．するとメモリの衝突なしに，処理装置はバケツに書込みができる．そして，最後に，m 本の配列をマージすればよい．このマージの操作は，2分探索木で行なうような具合に，処理装置が兄弟関係にある処理装置で活動中のものを探すことによる．P_i と P_j が互のマークをみつけると $(i<j)$，P_i は活動を続け，P_j は休止する (ここで重複した値が除去されることがわかる)．

Hirschberg は，この算法を一般化して，重複した数もソートされた列に残るようにするものを作ったが，この一般化のために，能率は犠牲にしている．この方法では，n 個の数のソートが，$n^{1+1/k}$ 個の処理装置を用い

て，$O(k \log n)$時間でできる（ここでkは任意の整数である）．

共有メモリ型モデルが具体化できないという点に加えて，並列バケツソート法には，$O(mn)$のメモリが必要であるという欠点もある．値の範囲がそれほど大きくないという場合でも，このメモリ量は減らすことが望ましかろう．値の範囲が大きい場合（たとえば，整数の代わりに文字の列である場合），この算法はこのままでは利用できない．

3.4 総当りソート法

ソートされる値の範囲に制限をつけないで，$O(\log n)$時間で走る並列ソート算法で総当りによるものがPreparata [1978]によって示された．キーの集合を部分集合にわけ，各要素に対して，それぞれ対応する集合でカウントする．すると，これらのカウントを合計することでその要素の順位が決定できる．Preparata の算法の最初のものは，Valiant のマージ算法[1975]を利用するもので，n個の数のソートが，$n \log n$ 個の処理装置を用いて，$O(\log n)$時間で実行できる．第2の算法は，Batcher の奇偶マージを利用しており，$n^{1+1/k}$ 個の処理装置で$O(k \log n)$時間のものである．こちらの算法の能率は，Hirschberg のもの (3.3節) に似ているが，メモリの衝突を起こさない点が優れている．Hirschberg のモデルでは，共有メモリから同時に読出しを行なっているが，Preparata のものは行なっていない（これは各キーがどの時点でも1回の比較にしか関与していないことによる）．

3.5 まとめ

最近の共有メモリ型算法は，メモリの衝突を除く改良が加えられているが，依然具体化に適当でない．どのモデルも処理装置の個数が少なくともキーの個数必要であり，きわめて大きいメモリを共有するので，現在や近い将来の技術では実現不可能である．さらに，これらのモデルでは，たとえば，ソートの途中で処理装置を割り当てるための時間といったオーバヘッドが考慮されていない．（なお，計算モデルにこの要因を取り入れることがVishkin[1981]によって試みられている．）

しかしながら，ここでの結果は，理論的にきわめて重要であり，そこで使われた手法は，ある種のソート算法の本質的な並列性を明らかにしている．また，今後の研究によっては，並列計算の共有メモリ型モデルの改良がうまくいくかもしれないし，計算機アーキテクチャの見方から無理のないものになる可能性もある．共有メモリ型モデルに対する研究においていろいろな仮定を分類することが Borodin と Hopcroft[1982] によって試みられている．特に面白いのは，同時読取りを許し，すべて同一の値のときにのみ同時書込みを許すという算法のクラスである[Shiloach と Vishkin 1981]．

4. ブロックソート算法

これまで説明した並列ソート算法はどれについても，ソートするレコードやキーの数と処理装置の数になんら制限をおいていなかった．n個のレコードのソートに$O(n)$個あるいはそれ以上の処理装置を使ってよいという具合である．つまり，これらの算法では，処理装置がいくつでも使えるものと仮定している．

このような仮定は，最初の頃，並列ソート算法の開発が高速スイッチ・ネットワークの実現を目的にしていたので，無理からぬものであった．こうした場合，n個の数をソートするのに，処理装置をn個（あるいは$n/2$個）使う理由としては，次の2つがあげられる．まず，スイッチ・ネットワークでは，個々の処理装置が2入力の比較・交換を行なう簡単なハードウェアであるし，また，処理装置の個数が入力線の本数に比例しており，その個数が極度に増えることはありえない．

しかしながら，汎用のソート算法としては，処理装置の数にある程度制限をおくにしても，ソートされるレコードの数に制限をおかないほうが望ましい．そして，大きな配列のソートに少数の処理装置しか用いないですませる必要がある．一般的にいって，ソートやサーチや数値計算などの問題に対する並列算法の研究では，無制限の並列性を仮定してきた．最近になって，技術的な制限を考慮することと並列算法への理解が深まったことにより，相対的に処理装置が少ない計算機のための算法が開発されはじめている．この動向の好例として，完全シャフルや超立方格子などのネットワークに対する商ネットワーク (quotient network) の研究があげられる[Fishburn と Finkel 1982]．商ネットワークは，制限付きの並列性を能率良く実現するアーキテクチャである．その考え方は，処理装置p個のネットワークによって，サイズが（任意に大きい）nである問題を解くのに，同じ接続の仕方で大きさ$O(n/p)$のネットワークを処理装置それぞれが模倣するというものである．これで，p個の処理装置が併わせて大きさ$O(n)$のネットワークを模倣する．

並列ソートの分野では，制限付き並列性は，最近まで

組織的に研究されてこなかった．今後この方向で研究するための基本的な考え方を次にいくつか提案しよう．

処理装置p個でレコードn個をソートする場合，レコードを処理装置に配分して，各処理装置の専用メモリに$M=\lceil n/p \rceil$ 個のレコードを1ブロックとして貯えるというやり方が考えられる．（最後のブロックにダミーのレコードを追加して個数を調整するものとする．）処理装置には，ネットワークの接続の仕方に応じて番号をふり，$P_1, P_2, \cdots, P_p$ とする．そして，次のように処理装置間でレコードを再配分する：

（1） 各処理装置のメモリにあるブロックは，長さMのソートされた列 S_i となっている

（2） これらの列を並べたもの $S_1, S_2, \cdots, S_p$ は長さnのソートされた列になっている．

たとえば，3個の処理装置では，ソートの前後でキーの配分の仕方が次のようになろう．

	ソート前	ソート後
P_1	2, 7, 3	1, 2, 3
P_2	4, 9, 1	4, 5, 6
P_3	6, 5, 8	7, 8, 9

以上のように処理装置全体に一定の順序を与えることにすると，nがpよりはるかに大きい場合も大きさnの配列の並列ソートという問題がはっきりする．

大きい配列を処理装置の専用メモリに配分してソートする算法としては，ブロックのマージ分割ステップの列として実現するものが考えられる．1回のマージ分割ステップでは，（前のステップで作られる）同じ大きさのソートされたブロック2つをマージして，次に，大きいほうと小さいほうの2つのブロックに分け，（比較交換ステップのようにして，）次の処理装置それぞれに送る．（比較交換ステップでできているソート算法において）比較交換ステップをすべてマージ分割ステップに置き換えればブロックソート算法ができる．この算法で正しくソートできることを確かめるのは容易である．

さて，マージ分割ステップを実行する方法は2通り考えられる．1つは，複線マージ (two-way merge) によるものであり[Baudet と Stevenson 1978]，もう1つは，双単調マージによるものである[Hsiao と Menon 1980]．4.1節と4.2節で，両方の方法を示して，奇偶転置ソート法（1.1節）と双単調ソート法（2.1.2節）にそれぞれ基づいているブロックソート算法を例で説明しよう．これらの方法で作られる並列ブロックソート算法に関する重要な性質としては，ネットワーク型ソート算法と同様に，SIMD機械(2.2節)で実行できることである．

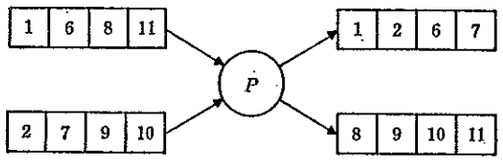


図10　複線マージに基づくマージ分割

4.1 複線マージ分割

複線マージ分割ステップは，大きさMのソートされたブロック2つを複線マージして，できた大きさ$2M$のブロックを半分ずつ2つに分けるものである．この操作は，各処理装置の専用メモリの中で実行できる．処理装置のメモリの内容が複線マージ分割でどう変わるかを図10に示す．長さMのソートされた列2つが専用メモリに納められると，処理装置が並列的にマージ手続きを実行して，出力バッファ$O[1\cdot\cdot 2M]$に結果を入れる．それで，複線マージ分割ステップでは，専用メモリが少なくとも$4M$必要になる．すべての処理装置がマージ手続きを終了すると，出力バッファを分割して，半分ずつ次の処理装置に送る．次の処理装置の番号は，前に調べた比較・交換の算法で定められるものと同じである．

4.1.1 複線マージ分割に基づくブロック奇偶ソート法

最初，p個の処理装置のメモリにはそれぞれ長さMの列が貯えられているとする．この算法は，各メモリの列それぞれをソートする前処理の段階（ステップ0）の後，（ステップ1からpまでの）p段階を経て，ステップ0で作られた列を並列的にマージする．ステップ0では，逐次的な高速ソート法のどれか1つを用いて，専用メモリの列をソートする．たとえば，クイック法 (Quicksort) を用いる．ステップ1からpまでは，奇偶転置ソート法（1.1節）のステップ1からpまでと同様である．奇数番目のステップでは，奇数番目の処理装置が右隣りからソートされたブロックを受け取り，複線マージを行ない，大きいほうのM個のレコードを送り返す．偶数番目のステップについても同様である．この算法は，同期的に実行できるものであり，奇数番目と偶数番目の処理装置が交互に働く．

4.1.2 複線マージ分割に基づくブロック双単調ソート法

Batcher の双単調ソート法を応用すれば，$p/2$台の処

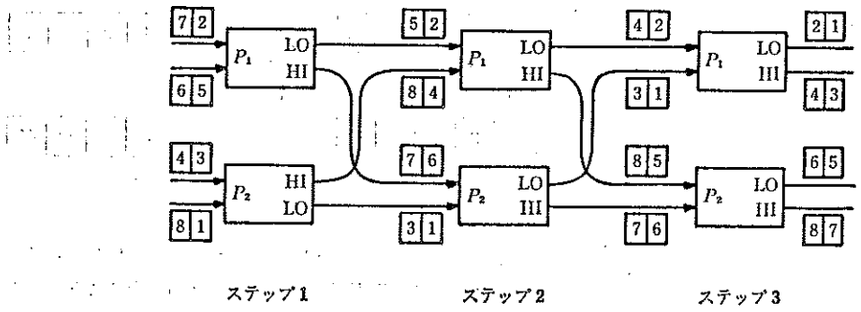


図 11　複線マージ分割に基づくブロック双単調ソート法

理装置によって，$\log^2 p$ 回のシャフルステップと 1/2 $((\log p)+1)(\log p)$ 回の比較・交換ステップで p 個のレコードがソートできる．今の場合，処理装置は，大きさ M のブロック2つに対して複線マージ分割を行なうことができる．そこで，比較交換ステップを複線マージ分割で置き換えれば，ブロックソート算法が出来上がる．この算法では，$p/2$ 個の処理装置によって，$\log^2 p$ 回のシャフルステップと 1/2 $((\log p)+1)(\log p)$ 回のマージ分割ステップによって，$M \cdot p$ 個のレコードをソートすることになる．マージ分割ステップでは，処理装置がそれぞれ(シャフルステップで受け取った)長さ M の列2つに対して複線マージ分割を実行し，長さ M の2つの列に分割する．そして，次のマージ分割ステップのために，処理装置がそれぞれ行先の処理装置に長さ M のソートされた列を送る．算法の実行例を図11に示す．ここで，処理装置は2個で，$M=2$ である．

一般的には，この算法では，$p/2$ 個の処理装置が必要である．ここで，p は2の巾として，$M \cdot p$ 個のレコードのソートには，各処理装置が大きさ $4M$ の専用メモリをもつものとする．

4.1.3　処理装置の同期

M が大きい場合やレコード自身が大きい場合，処理装置間で $M \cdot p$ 個のレコードを転送すると，各処理装置の命令実行速度に比べて桁違いに遅くなる．さらに，データの分布の具合によっては，大きさ M のブロック2つのマージに要する比較回数が異なりうる．それで，複線マージ分割に基づくブロックソート算法を実行するには，処理装置の同期を機械命令レベルで同期をとる SIMD 型のものよりもっと緩くするほうが望ましい．多数の処理装置がそれぞれ独立に働くというモデルで，処理装置間でメッセージを交換することで同期をとるとか，あるいは1つの主処理装置が数千命令ごとに同期をとるとかいったものがこの算法により相応しい．ブロックソート算法の開始時点で，主処理装置がいくつかの処理装置を実行させる．他の計算を行なっているかもしれないので，主処理装置は，使用可能な処理装置をリストに並べておいて，このリストから実行するものを割り当てていく．この主処理装置は，使用可能かどうかを調べるだけでなく，ソートするデータ量に応じて処理装置をうまく割り当てるものとする．

4.2　双単調マージ交換

今2つの処理装置 P_i と P_j にそれぞれ長さ M のソートされたブロックがあるとして，処理装置間でレコードを比較・交換して，小さいほうの M 個のレコードを P_i に，大きいほうの M 個のレコードを P_j に置くことを考える．これは，次のような方法で実行できる：

(1)　P_j は P_i にブロックを送る

(2)　P_i は複線マージ分割を行なう

(3)　P_i は大きいほう半分を P_j に送る

しかしながら，前節で触れたように，複線マージ分割では，1つの処理装置の専用メモリが $4M$ 必要になる．それで，別の方法として，P_j がレコードを1つ送り，P_i からレコードが1つ返ってきてから，次のレコードを送るというものが考えられる．いま，P_i のメモリに上昇順に M 個のレコード $(x_1, x_2, \cdots, x_M)$．P_j のメモリに下降順に M 個のレコード $(y_1, y_2, \cdots, y_M)$ がそれぞれ入っているものとしよう．まず，P_j が y_1 を送り，P_i で x_1 と y_1 を比較する．このうち，小さいほうを P_i に残し，大きいほうを P_j に返す．同様にこの手続きを (x_2, y_2), $\cdots$, (x_M, y_M) に対して繰り返す．この比較交換の列は，双単調マージになっており，P_j には大きいほうの M 個のレコード，P_i には小さいほうの M 個のレコードがそれぞれ残る [Alekseyev 1969; Knuth 1973]．こうして，P_i と P_j それぞれが独立にソートを行なえば，マージ分割の操作は完了する．図12には，$M=5$ に対

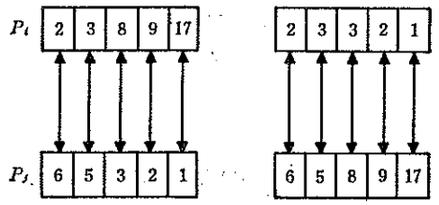


図 12　双単調マージ交換ステップ

する双単調マージ交換の操作の例を示す．ここで，(複線マージ分割と異なり，) データの交換が同期して行なわれることに注意されたい．それで，双単調マージ交換に基づくブロックソート算法は，処理装置間の同期性の高い並列計算機で実現するのに相応しい．

双単調マージ交換は，複線マージ分割よりバッファ領域がはるかに小さくてよい．複線マージ分割は，専用メモリで大きさ M のブロック2つをマージするので，$4M$ のメモリ量を使う．一方，双単調マージ交換は，わずか $M+1$ 個のレコードの分しか必要としない．双単調マージ交換では，各ステップで，(レコードの対の) 比較と転送を並列的に実行できる．つまり，複線マージ分割では，マージをはじめる前に，ブロック全体を転送しておかなければならないのに対して，双単調マージ交換では，レコードの転送時間と処理時間を重ねることができる．

しかしながら，双単調マージ交換の欠点は，交換のステップの後で，処理装置それぞれが独立にソートを行なう必要があることである．このソートには，(ヒープ法 (Heapsort) のように) 余分の記憶場所を使わない算法を用いるべきである．でなければ，交換のために用いる分以上に使ってしまうことになる．なお，双単調交換で作られた列は双単調になることに注意せよ．

4.2.1　双単調マージ交換に基づく ブロック奇偶ソート法

複線マージに基づくブロック奇偶マージと同様に，ここでも，各処理装置のメモリにレコードが M 個納められているとして，各メモリで独立にソートすることではじめる．しかし，ステップ1から p までは異なっている．奇数番目 (偶数番目) のステップでは，奇数番目 (偶数番目) の処理装置が右隣りのものと双単調マージ交換を実行する．図13に，$M=5$, $p=5$ に対する算法の実行例を示す．

4.2.2　双単調マージ交換に基づく ブロック双単調ソート法

2.1.2節で説明した双単調ソート法の Stone によるものを用いて，高速でメモリ量も少ないブロックソート法を作ることができる．いま，p 個の処理装置からなるネットワークを考えよう．ここで，p は2の巾であるとして，処理装置は次の2種類の線で連結されているとする (図14)．

(1)　隣り合う処理装置 $P_0P_1, P_2P_3, \cdots$ をつなぐ両方向の線

(2)　各 P_i のシャフルに用いる一方向のシャフルの線

各処理装置が大きさ $M+1$ の専用メモリをもっていれば，$M \cdot p$ 個のレコードのソートは，専用メモリ内でのソート，隣り合う処理装置間のブロック双単調交換，シャフルの3操作を順番に繰り返すことによって実行できる．シャフル操作では，各処理装置は，メモリに入っているレコードを順次受取り側の処理装置に送るとともに，送り側からのレコードを順次受け取る．図15に，$M=5$, $p=4$ に対する算法の実行例を示す．

5.　並列外部ソート算法

この節では，大きいファイルを並列的にソートすることを考えよう．逐次的なソートで“外部ソート”といわれるものに対応する．これまでに説明したものは，“内部ソート”とよばれる．普通の計算機システムでは，外部ソートは，ファイルが主記憶に納め切れない状況で必要になる．そこで，単一の処理装置では，内部ソートと外部ソートの区別がはっきりするし，算法の評価基準も定めやすかった．しかしながら，並列的な外部ソートという話題は，十分考察されてきたとはいえない．

第4節では，複数の処理装置に配分されているデータをソートする算法をいくつか説明してきた．この場合，データ全体の大きさは，(処理装置の専用メモリを併せた) 全メモリ量以下であるものとしていた．逐次的な内部ソートと同様に考えると，そのような算法は，“並列内部ソート算法”といってもよかろう．

並列ソート算法が“並列外部ソート算法”といわれるのは，処理装置のメモリ全部を併せてもなお納まらないほど大量のデータをソートするような場合である．この言葉使いによれば，“共有メモリ”多数処理装置や“緩く結合された”多数処理装置のような並列アーキテクチャなども含めて取り扱うことができる．

共有メモリ多数処理装置に対しては，データ全体 (とソート算法のための作業用場所) が共有メモリに納まらない場合に，外部ソート算法が必要になる．また，緩く結合された複数処理装置では，データ全体が処理装置の専用メモリに配分し切れない場合に対応する．つまり，

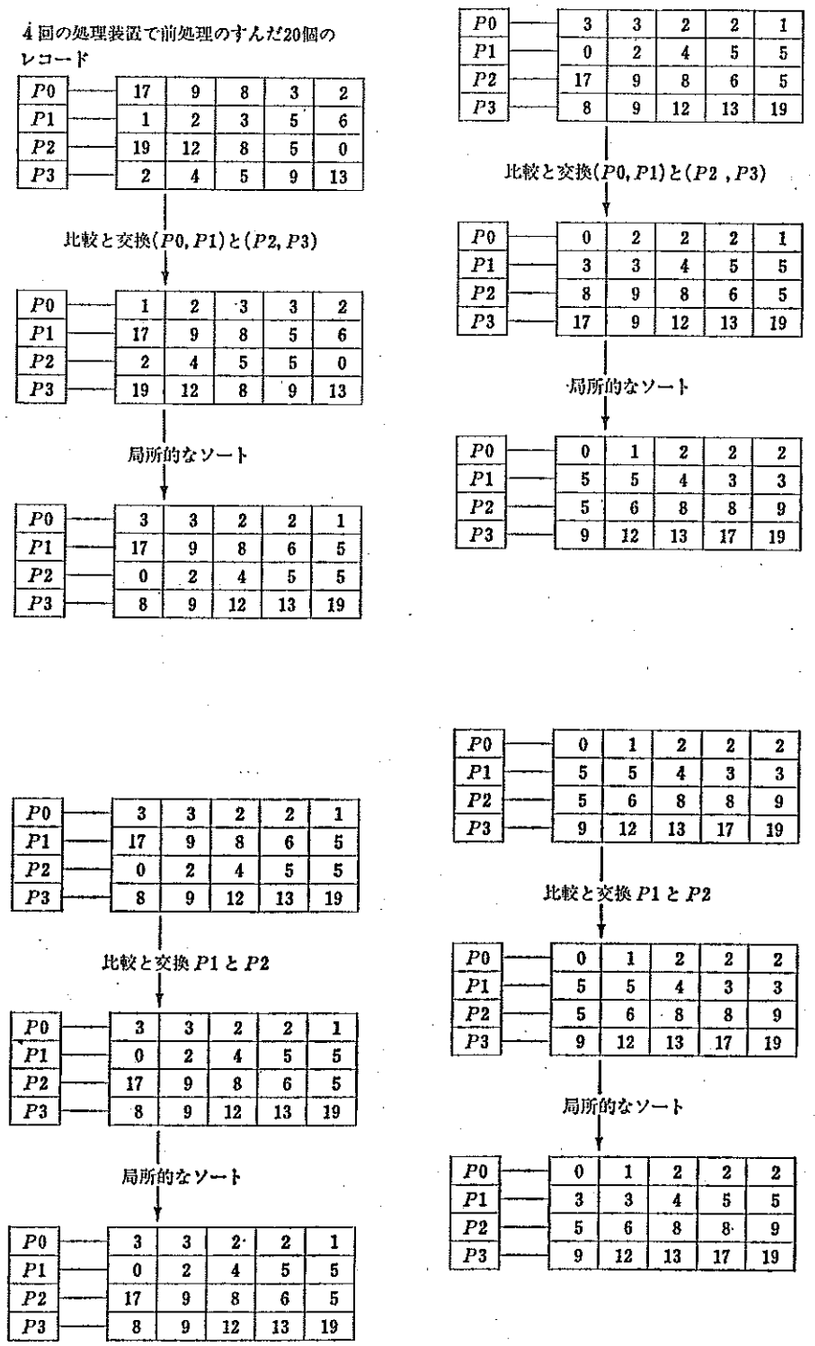


図 13　ブロック奇偶ソート法 (20個のレコード)

複数処理装置は，p個の同一の処理装置をもち，各専用メモリにk個のレコードを納めることができるとして，ソートするデータが$p \cdot k$個のレコードより多い場合である．算法の終了時点では，全ファイルが大容量記憶装置にソートされた順に書き込まれることになる†．

磁気テープに対する並列ソート法に関する結果がまず Even [1974] により発表されている．最近，Bitton [1982]により，可動ヘッドディスクに調整を加えたものに対する並列ソート算法がいくつか提示されている．

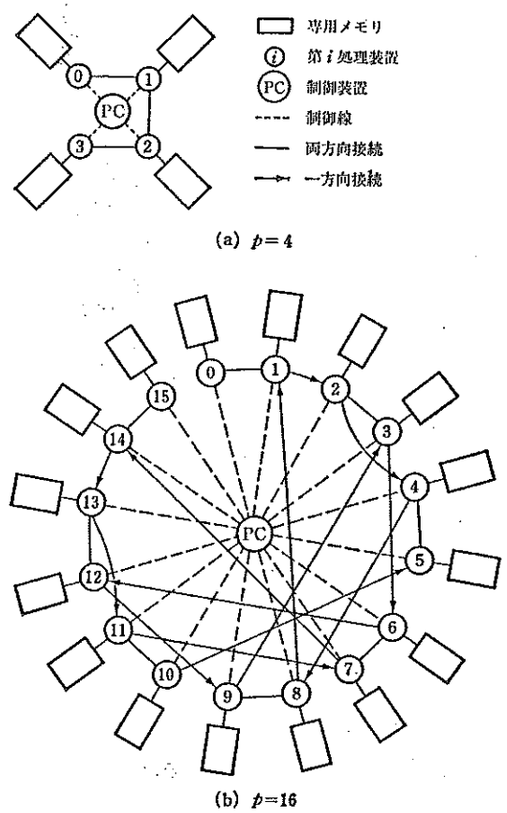


図 14　ブロック双単調ソート法の処理装置の接続

5.1　並列テープソート算法

Even [1974] が考えた問題は，p個の処理装置と$4p$本の磁気テープでn個のレコードをソートすることである（ここで，nはpよりはるかに大きいとする）．内部メモリに関する要請としては，処理装置の専用メモリに3個のレコードが納まるということである．このような仮定の下で，Even の提示した2つの方法は，逐次的な外部複線マージソート算法を並列化したものである．一番目の方法では，処理装置がすべて一斉に働きはじめ，互に独立に，ファイルの分割された部分を取り扱う．二番目の方法では，パイプライン的にソートを実行する．両方法とも簡単に説明できる．

方法1：各処理装置には，4本のテープとn/p個のレコードを割り当てることにして，(逐次的な) 外部ソート法によりそれぞれソートする．この並列的な段階で，p本のソートされた列ができるので，あとは，やはり普通の外部ソート法により，2本ずつ並列的にマージしていけば，最終的に1本のソートされた列を作ることができる．

方法2：基本的な考え方は，各処理装置には逐次的なマージ手続きの各段階を割り当てるというものである．i番目の処理装置は，大きさ2^{i-1}の上昇列の対を大きさ2^iの上昇列にマージすることを担当する．いまnがちょうど2の巾で$\log n$個の処理装置が使えるものとして考えよう．1つの処理装置の出力テープを次の処理装置の入力テープとすることで，高度の並列性が実現できる．ここで，1つの処理装置が出力として，2つの上昇列を作り出すとすぐに，もう1つの処理装置がそれらを読み込んでマージするものとする．処理装置の出力時間と次の処理装置の入力時間を重ね合わせるために，各処理装置は，出力に4本のテープを順繰りに用いる（つまり1本のテープに1つの上昇列を出力する）．

これらの方法をみれば，算法としての見方から，普通の外部複線ソート法に高度の並列性を導入できることがわかる．しかしながら，大容量記憶装置に対するここでの仮定には，技術的な制約が十分考慮されていない．内部ソーティングの共有メモリ型モデルと同様に，並列外部ソーティングのモデルには，(p個の処理装置で$4p$本のテープ装置のモデルのように,) いくらでも同時に入出力できる大容量共有記憶装置が使えるものと仮定しているので，実際の場での実現のためには，あまり良い見通しが得られない．

5.2　並列ディスクソート算法

磁気ディスクは逐次的な装置でないので，ファイルがソートされているという場合，物理的な順序というものを定義してやる必要がある．ディスクのトラックの中では，レコードが逐次的に貯えられるが，トラックには番号をふる約束がいる．たとえば，隣り合うトラックというものは，ディスク面で相続くものであるという具合である．この約束は，異なる処理装置が別々のディスク面に対応させている場合に適当である．大容量記憶装置のモデルとしては，同一シリンダのトラックに対して並列的に読み書きができるように可動ヘッドディスクを調整

† 大容量記憶装置での物理的順序は，その装置の物理的な性質に応じて定義されるものとする．たとえば，磁気ディスクでは，トラックの番号の約束に従って定義される．

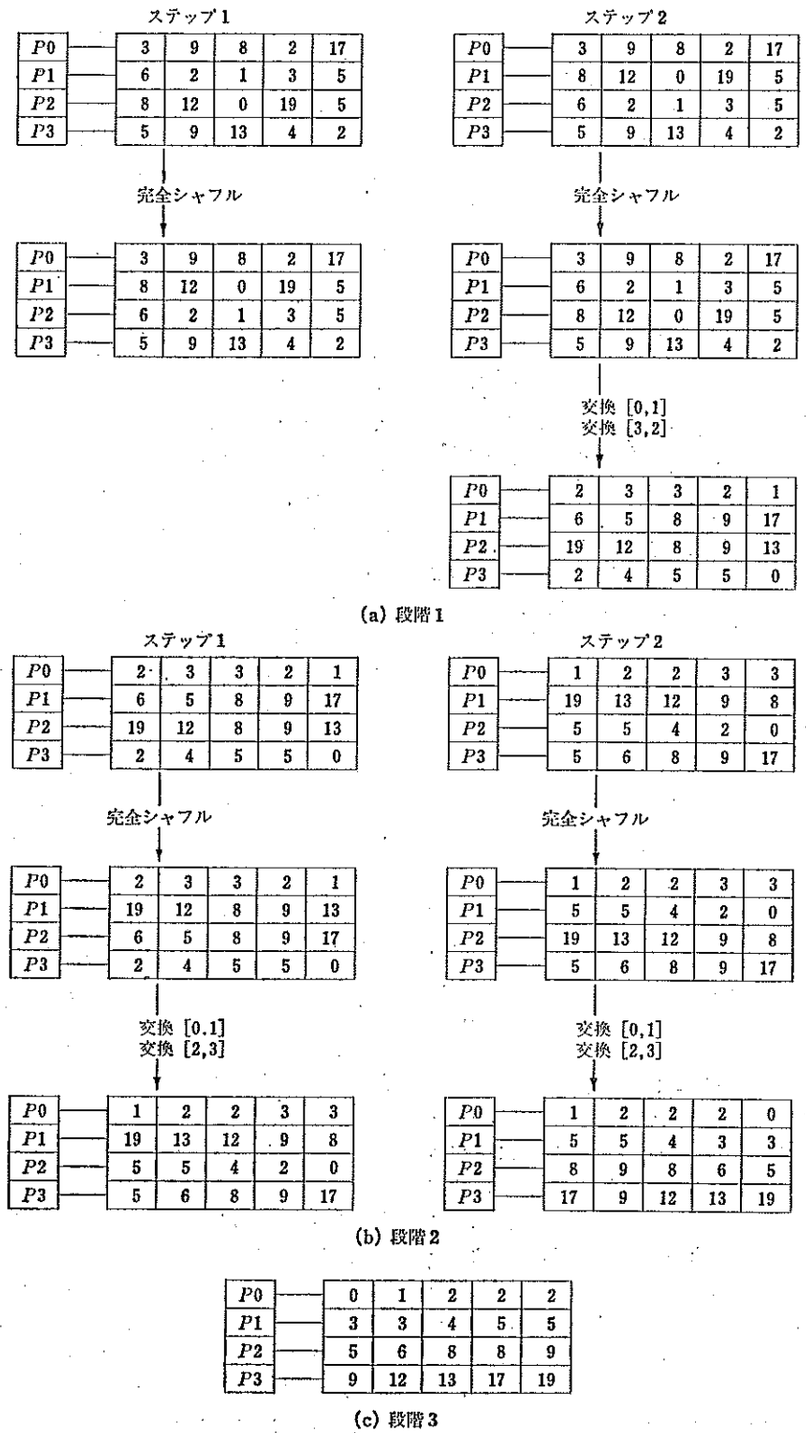


図 15　ブロック双単調ソート法

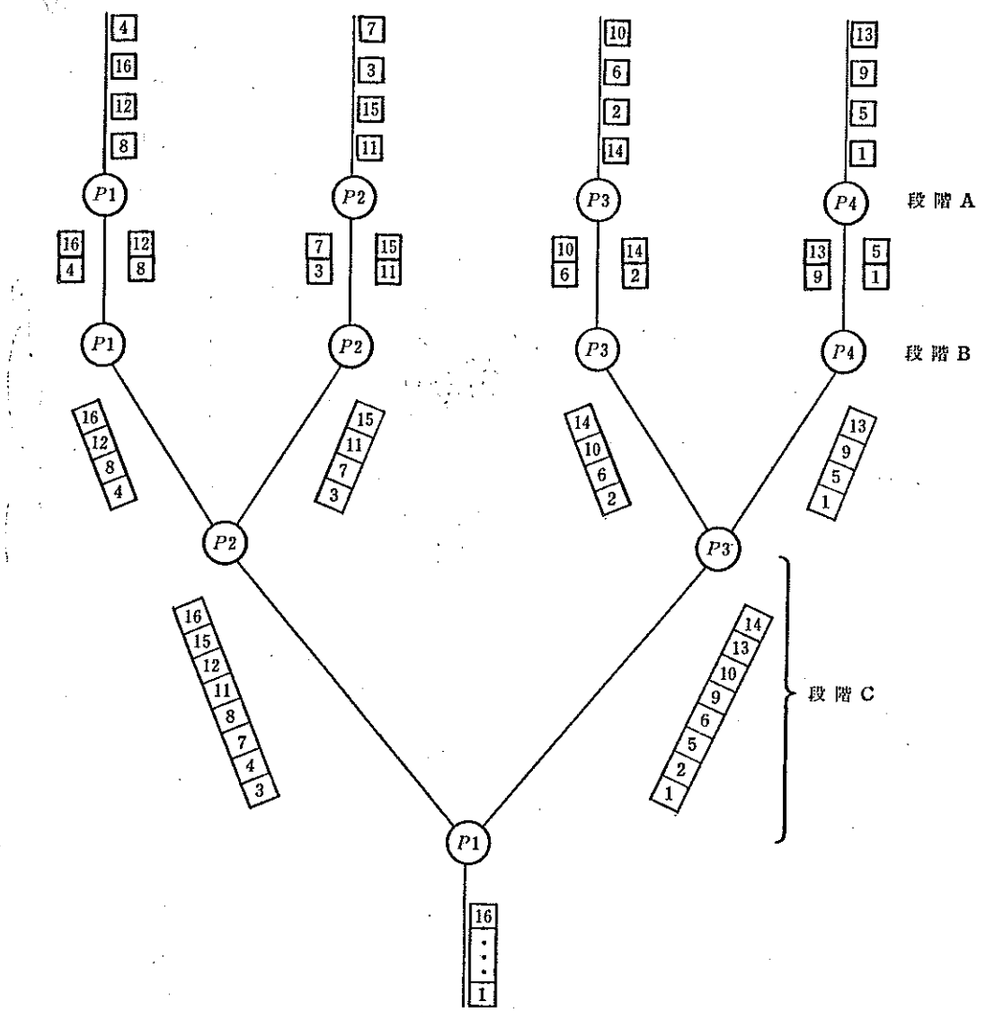


図 16　並列2分木マージソート法

したものがある（図17）．この機能をもつディスクはすでに提案されていたが［Banerjee 他 1978］，実際にもいくつか実現されている．この考え方は，データベース機械の設計ではじめて使われ，データベース研究用として具体化されている（たとえば Leilich 他［1978］）．商業用の並列読出し可能なディスクは，最近，高性能の計算機で使用できるようになっている（たとえば，600 メガバイト，4 トラック並列読出し可能，データ転送速度 4.84メガバイト/秒のものが Cray-1 コンピュータで使える）．それで，並列読出し可能のディスクは，コスト的に有利な普通の可動ヘッドディスクと旧式の固定ヘッドディスクの間でうまく調和をとったものと思われる．

シーク時間を最小にするために，ディスク装置2台を並列的に使うことができる．ソート算法の1段階を実行するのに，一方を読出しに使い，他方を書込みに使う．

Bitton-Friedland［1982］では，並列外部ソート算法とそのためのアーキテクチャが研究されている．そこでは，大容量記憶装置が並列読出し/書込みディスクとして，モデル化されており，最高の性能をもつ算法は，並列2分木マージ法（parallel binary merge）と名づけられた並列外部複線マージソート法である．これは，5.1節の方法1の改良版であり，この方法の第2段階を並列化したものである．

出力の上昇列の個数が 2^k になった場合（$k>1$），2^{k-1} 個の処理装置を用いればマージソート法の次のステップが並列的に実行できる．それで，並列2分木マー

ジ法は，図16に示すように3段階に分けて実行される．この算法は，（方法1の第1段階と同様に）まず段階Aで実行がはじまる．この段階では，上昇列の対を次々とマージして，上昇列の個数が処理装置数の2倍に等しくなるまで続ける．この段階Aでは，処理装置が並列的に働くが，おのおの独立したデータを取り扱っている．並列入出力は，各処理装置に読み側のディスクと書き側のディスクの各面を対応させることにより実現できる．

上昇列の数が$2p$になると，各処理装置は，2つずつ上昇列をマージする．このときマージされる上昇列の長さはおのおの$n/2p$である．これが段階Bである．次の段階Cでは，並列性が2通りのやり方で導入される．まず第一に，2^{k-1}個の処理装置が上昇列の2^{k-1}個の対を並列的にマージするのに用いられる（これは$\log(n/p)$回のマージのステップの後生ずる）．第二に，マージのステップの間，パイプラインの考え方が使われる．i番目のマージのステップは，$(i-1)$番目のステップで2本の出力上昇列のはじめのデータができるとすぐに，i番目のマージステップが開始される（ここで1つのデータは，1つのレコードのこともあるし，ディスクのページ全体のこともある）．

この算法を実行するアーキテクチャとして理想的なのは，図17に示すように，処理装置が2分木状になっているものである．大容量記憶装置は，2つのディスク装置でなり，葉の処理装置はそれぞれ両方の装置の面に対応している．葉の処理装置の他に，根の処理装置も出力ファイルを書き出すためにディスクを参照する．この構成によって，葉の処理装置は，入出力を並列的に実行できるし，実際に入出力を行なう処理装置の個数をほぼ半分に減らすことができる．

5.3　並列外部ソート算法の解析

逐次的な外部ソート算法については，実際の計算機で実際のデータを使って，算法の評価のための実験的研究が数多くなされている．参照時間の効果やデータの分布の影響を解析的なモデルを作って調べるのが複雑すぎる場合，これらの実験的な研究結果は解析的な結果と相補うものであった．並列的な計算機では，入出力装置が複雑なので，外部ソート法を解析的に性能評価することは一層難しくなる．

同時にできる入出力数が処理装置数で押えられるものと仮定すれば，外部ソートを並列的に実行することによる速度向上についてある程度の目安を得ることができる．しかしながら，並列外部ソート算法を解析して十分

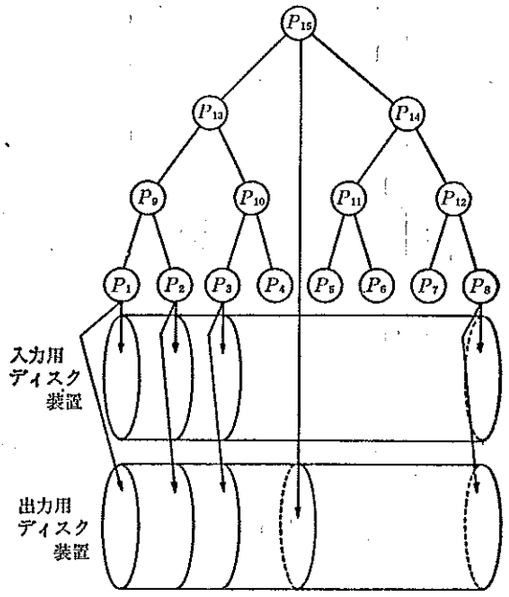


図17　並列2分木マージソート法のアーキテクチャ

な結果を得るには，大容量記憶装置の技術からくる制限も考慮しなければならない．たとえば，5.2節で述べたディスクの調整したものを用いる場合，並列2分木マージ算法の段階Aは，複数の処理装置が同一シリンダ内のトラックを参照できるか否かによって，高度に並列的に実行できたり，ほとんど逐次的にしか実行できなかったりする．

6.　ハードウェア・ソータ

ソートは，頻繁に必要であり，コストが高くつくので，ソート・エンジンを設計して，汎用の中央処理装置からソート機能を分離させようという考え方がある．高能率のソート算法で実行される比較・転送の操作をハードウェアで実現すれば，中央処理装置の負担を大幅に軽減するような低価格で高速な装置になろう．最近，ハードウェア・ソータ（hardware sorter）がいくつか提案されている[Chen 他 1978; Chung 他 1980; Dohi 他 1982; Lee 他 1981; Thompson 1983; Yasuura 他 1982]．また，予備的な評価によれば，ソータは，VLSIでもうすぐ実現できるとされている．ソートには，比較的簡単なロジックしか必要でないことが，この方向が支持される理由になっている．さらに，磁気バブルメモリやCCDのように新しい低価格のシフトレジスタ技術が利用可能になったことが刺激になって，ハードウェア・ソータが設計されている[Chung 他 1980; Lee 他 1981]．

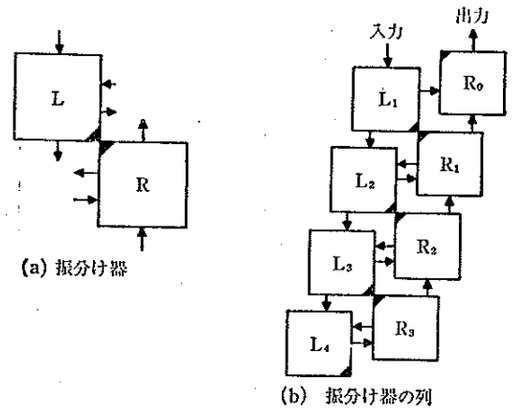
将来技術が進歩すれば，磁気バブルのチップでソート機能をもつ大容量記憶装置が実現できるであろう．この場合，ソート機能は，大容量記憶装置と一体となっていて，別のソート専用機や汎用計算機にファイルを転送することはない．しかし現時点では，技術の進歩によって，ソート機能をもつ知的な大容量記憶装置が具体化できるかどうかを判定するにはまだ時期尚早である．

現在，ハードウェア・ソータ，特にVLSIソート用回路は，盛んに研究されている．面積・時間複雑さに関する理論では，VLSIによるソートが非常な関心を集めている[Leiserson 1981; Thompson 1980, 1983]．VLSIソータのチップ面積と時間に関する理論的結果を概観するのは，この論文の範囲外のことである．これらは，計算量理論で確立しつつある研究分野での結果であり，VLSI回路の面積をきちんと定義せずに，あるいは面積×時間^{2}の理論的な関係を説明せずに，ここで紹介することはできない†．

この節では以下，ハードウェア・ソータの設計案をながめていく．ここでは，もともと磁気バブルのために考えられたソータをもっぱら取り上げるが，その理由は，同時入出力量，メモリ，制御線数などのパラメタとソート速度の関係が技術的制約により決定されることをよく示しているからである．特に，反転ソータ（rebound sorter）と昇降ソータ（up-down sorter）を説明する．これらは，奇偶転置法（1.1節）を巧みにパイプライン化したものである．それから，磁気バブルメモリにソート機能を組み込んだ磁気バブルソータをいくつか調べる．

6.1　反転ソータ

一様梯子（uniformladder）[Chen 1976]は，1つのループに1個のレコードを対応させ，レコードN個を貯えるループN個のシフトレジスタである．隣り合っているループに貯えられているレコードは互に交換できるので，この構造は，奇偶転置法（1.1節）をハードウェア化するのにぴったりである．ループ内でビットが1周する時間を周期とよぶことにすると，（梯子に貯えられている）レコードN個は，$(N-1)$個の比較器を用いて，$(N+1)/2$周期でソートできる．



(a) 振分け器　　(b) 振分け器の列

図18　反転ソータ [Chen 他1978© IEEE 1978]

この梯子構造はさらに詳しく調べられて，レコードの入出力を完全に重ね合わせることができる別の新しいソート法が設計された．これが反転ソート法である[Chen 他1978]．基本的な構成要素は，左上セルLと右下セルRをもつ振分け器である（図18(a)）．レコードN個のソータは，$(N-1)$個の振分け器を$(N-1)$個重ね，さらに一番上と下にそれぞれセルを追加したものである（図18(b)）．振分け器の2つのセルには，比較器が1個つけられていて，左上と右下のセルの値2つを比較する．各レコードは，隣り合う2つの振分け器を使って貯えられるが，ソートのためのキー（レコードの先頭部分にあるとする）の大きさは，必ず1つのセルに納まる必要がある．これは，1つの振分け器の中で2つのキーを比較するためである．振分け器の列を通してレコードをパイプライン的に流すことでソートが実行される．レコードは，最上段の左上セルから入り，$2N$ステップ後に，最上段の右上隅のセルからソートされて出ていく．この方法で$N=4$の例を図19に示す．ソータは，交互に判定状態と継続状態をとる．判定状態では，振分け器が左上セルと右下セルのキーを比較して，比較の結果に従って，水平方向（左上キーを右方向，右下キーを左方向）か垂直方向（左上キーを下方向，右下キーを上方向）かに送る．継続状態では，直前の判定状態で定められた方向にレコードを送り出すが，この状態が必要なのは，キーの後にレコード本体を付け加えねばならないからである．すぐわかるように，最初のキーは，Nステップ後（図19ではt_0+8）にソータから送り出され，次のN

† （本論文の書かれた後で発表された）Thompsonの仕事の中で，ヒープ法，パイプライン・マージ法，双単調法，バブル法，総当り法などのソート算法の実現のためのVLSI回路が13種類調べられている．これらのソート算法のおのおのについて，1つあるいは複数の回路の接続方法（一次元配列，網目，二分木，シャフル交換，CCC，木の網目）が考えられていて，面積×時間^{2}に関する評価が与えられている．

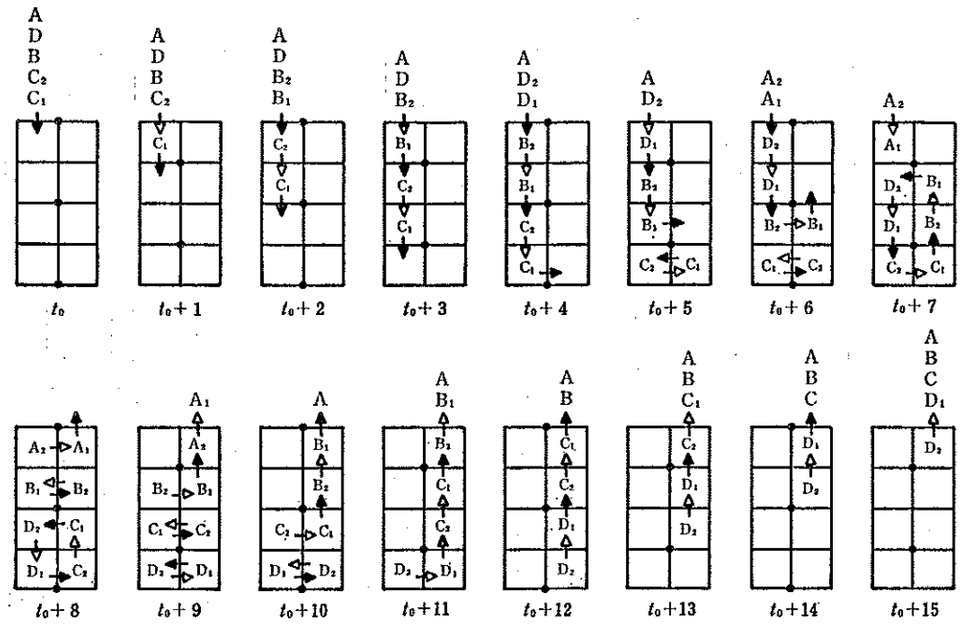


図19　4個のレコードに対する反転ソータの操作 [Chen 他 1978 © IEEE 1978]

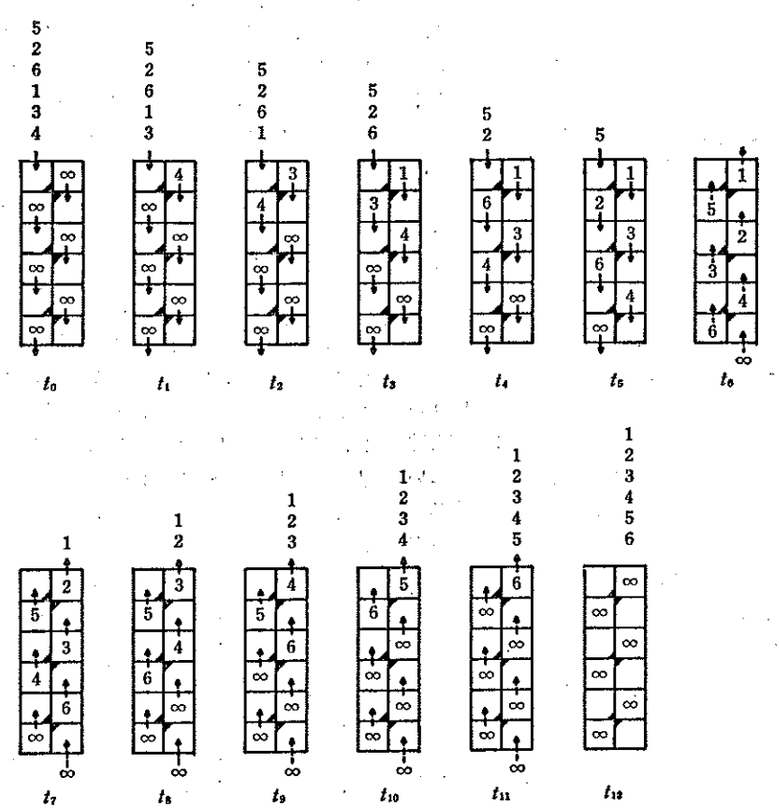


図20　6個のレコードに対する昇降ソータの操作 [Lee 他 1981 © IEEE 1981]

ステップでソートされた列が全部送り出される．

6.2 昇降ソータ

梯子形ソータは大幅に改良できる．それは，基本的な振分け器に比較機能を組み込み，反転ソート方式の代わりに，昇降ソート方式を用いるものである．"比較/振分け磁気バブルソータ" [Lee 他 1981] では，$2N$個のキーをソートするのに，N個の比較/振分けユニットを重ねたものである．いま，レコード全体が1つのセルに納まるものとしよう．(それで1つの比較/振分けユニットに2個のレコードが入る．) 昇降ソート法で $N=3$ の例を図20に示す (比較/振分けユニット3個でレコード6個をソートする)．昇降ソータは2段階に分かれている．まず，下方向の入力段階では，$2N$周期で$2N$個のキーが入力される．入力段階の各周期では，キーが入力されると，すべての1ユニットは，2つのキーの大きいほうを下方のユニットに押し下げる．上方向の出力段階では，それぞれのユニットは，2つのキーの小さいほうを上方のユニットに押し上げる．そして，各周期ごとに1つずつキーが出力される．

昇降ソータでは，反転ソータで必要であった数多くの制御線が不要になる．反転ソータでは，磁気バブル梯子のスイッチを独立に制御するための多数の制御線を必要としたのに対して，昇降ソータでは，2つの段階の最初に，すべての比較/振分けユニットをリセットするための制御線が1本あればよい．それで，比較/振分けユニットをチップ上に作るほうが大ファイル用ソータを実現するのに好都合であろう．

反転ソータも昇降ソータもソート時間と入出力時間を完全に重ね合わせることができるという大変うまい性質をもっている．それで，レコードの入出力を逐次的に行なうという仮定の下では，ファイルのソートが最適なハードウェアとして実現できる．

6.3 磁気バブルメモリによるソート

これまで説明したソータでは，磁気バブルのチップの設計に比較機能を取り入れていた．それで，この種のチップは，ソートに必要な論理操作が実行できる知的メモリでできている．Chung 他 [1980] が提案したソータでは，やはり磁気バブルメモリに比較器を取り入れているが，前の算法では本質的であった入出力機能を取り除いている．メモリ内でソートする磁気バブル回路を設計する動機は，磁気バブルメモリがやがて価格的に有利な大容量記憶装置として使えるという想定からきている．技術の進歩でこの想定が現実的になれば，ファイルのソートには，入出力操作や CPU の介在なしに，メモリ内で実行される比較の結果に応じて，大容量記憶装置内でレコードが並べ換えられることになる．

知的な磁気バブルのモデルが Chung 他[1980]により4種類考えられており，モデルごとに別のソート方式が提案されている．モデルの差異は，磁気バブルのループの大きさとループ間の (比較のための) スイッチの数によっている．そのうち2つのモデルは，バブル法と奇偶転置法をそれぞれ実現するものである．他の2つは，双単調法を実現している．第一のモデルは，大きさ $(n-1)$ と1の2つのループでなるもので，ループの間に1つスイッチがある (図21(a))．これでバブル法を実行する．第二のモデルは，大きさ1のループを一次元的に並べたもので，隣り合うループの間に1つずつスイッチを置いている (図21(b))．$(n-1)$ 個のスイッチは，奇偶転置法(1.1節)に従って並列的に比較を実行する．他の2つのモデルでは (図21(c)，(d)，基本となる考え方として，(モデル3で同一の大きさ，モデル4で異なる大きさの)隣り合うループの間のスイッチを開放して，2つのループを大きいループにまとめることもできるようになっている．双単調ソートの各ステップで，双単調列をふくむ大きいループが作られる．このソータは，速い算法を実現しているので，前のものより高速である．しかしその代わりに，ハードウェアが著しく複雑になる．スイッチが多数必要であり，スイッチ当りの状態数もふえる．それで，現在のチップの密度では実現が難しい．たとえば，バブル法によるソータでは，$O(n^2)$ 回の比較回数でソートするが，わずか1個のスイッチ (3状態) しか必要でない．一方，双単調ソータでは，$O(n \log^2 n)$ 回の比較でソートできるが，$\log n$ 個の複雑なスイッチ (状態数 $3 \log n$) が要る．磁気バブルソータの設計を詳しく考えれば，ソートにおける価格と性能の相反する関係がよくわかる．

6.4 まとめと最近の結果

最近，ハードウェア・ソータが数種類提案されており，それらの実現可能性について予備的な評価が行なわれている．最も見込みのある方法の1つとして，磁気バブルチップに単純なパイプライン式のソート法を実現するものがあげられる．

しかし，その他の設計もいくつか研究されている．ごく最近，VLSI ソータの詳しいレイアウトが提案されている．総当りソート法を実現する大容量のセル構造配列

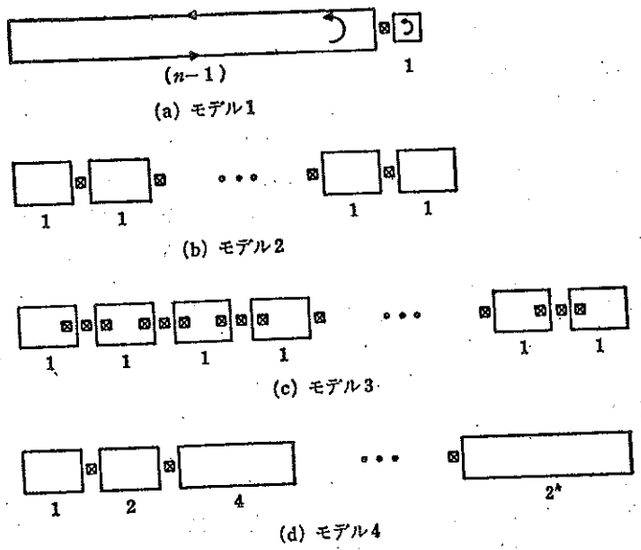


図21　磁気バブルのループ構造 [Chung 他 1980 © IEEE 1980]

が Yasuura 他[1982]により研究されている．Dohi 他[1982]には，圧縮された形のデータをソート/マージできるセルを2分木構造に接続して，強力なソータが作られている．どちらのものも，基本となるセルは，十分簡単なものなので，現在か近い将来の技術で非常に高密度に詰めることができる．

現在提案されているハードウェア・ソータでは，そこに使っているソート算法がソート用ネットワークや共有メモリ型モデルの算法のいずれと比べても，単純で遅い．もっと速い VLSI ソータに対して，理論的な計算量の限界や重要な結果が得られているが[Bilardi と Preparata 1983; Thompson 1983]，高速で大容量の VLSI ソータが実現可能であるかどうかはいぜんわかっていない．しかしこの方向での並列ソート法の研究は，将来性が大きい．ハードウェアの複雑さと時間的能率を結びつけた，うまい VLSI のモデルが定義できれば，並列ソート算法の解析が組織的に研究できる．磁気バブルメモリ装置に対しては，レコードが逐次的に読み書きされると仮定して，入出力時間がソート時間と完全に重ね合わせうることを示した．それで，技術の進歩によって，やがて，うまく設計されたソート専用装置が，価格的に有利なものとして，多くの計算機システムに付加されることになろう．

7.　結論と研究課題

この10年間，並列ソートは集中的に研究されてきた．現在，並列ソート算法が数多く知られているし，新しい算法も開発されている．それらは，ネットワーク型から共有メモリ型や VLSI チップまでさまざまなものである．並列ソートは，理論家にもシステム設計者にも数多くの研究課題を提供してきた．理論的な見方からは，主な課題は，ソートやマージの本質的な並列性を考慮して，時間に関する理論的下界を達成するような算法を設計することである．つまり，その算法は，n 個の数のソートが，$O(n)$ 個処理装置をもつ並列計算機の上で，$O(\log n)$ 時間でできるものであろう．実際的な見方からは，現在または近い将来の技術でシステム設計に使えるかどうかを調べ，入出力時間も並列ソートのコストに取り入れて考慮することである．

これまで提案されてきた算法は実にさまざまなものであるが，大きく分けてネットワーク型，共有メモリ型，並列ファイルソートの3種類に分類できる．第一のものは，非適応型のマージの繰返しに基づく算法である．基本的な並列マージ算法2つ（2.1節で説明した奇偶マージと双単調マージ）は，当初ソート用ネットワークのために提案されたものであるが，後でもっと一般的な並列計算モデル（疎に結合したネットワークの通信線を通して処理装置がデータを同期的に交換するもの）に適用されている．特に，双単調ソート法は，網目状に接続された処理装置やシャフル，超立方格子，CCCなどのネットワークなどに応用されている．

第二の種類の算法では，ネットワーク型算法よりメモリの参照がもっと融通の効く方式が必要になる．この算法では，処理装置が1つの大きなメモリ装置に対して読み書きを行なうという共有メモリのモデルを仮定している．ここで，メモリの読み書きには，制限がさまざまな形で導入されている．大体において，共有メモリ型の並列ソート算法は，ネットワーク型のものより高速であるが，ハードウェアの見方からは，実現がはるかに難しい．表1には，ネットワーク型と共有メモリ型の算法の主なものについて，処理装置数と計算時間の漸近的な上界を簡単にまとめている．（ここで，計算時間は，算法で実行される並列的な比較回数で計っている）．

第三の種類としては，大規模な問題に対して並列性を限定したソート算法を取り扱った．まず，処理装置数に比例する要素数のデータに対するブロックソート算法を調べた．ここで，その比例定数は，処理装置のメモリの大きさによって定まる．次に，大容量記憶装置のファイルをソートするための並列外部ソート算法を紹介した．

これら3種類のソート算法の他に，6節では，ハードウェア・ソータの設計をいくつか説明した．これまで提案されているハードウェア・ソータは，処理部分が疎に接続されて固定されているものとしている．このソータによる並列ソート算法は，高度に同期的であり，大抵，ネットワーク型と分類した算法（特に双単調ソート算法）から作られたものである．ハードウェア・ソータには新しい考え方の算法が導入されていないが，その設計には，並列ソーティング研究の新しく重要な方向が示されつつある．そのような方向の1つとして，磁気バブルのような新しい記憶装置技術の特徴を生かした算法の開発があげられる．もう1つは，VLSI ハードウェアの複雑さとコストのモデルを設定して，並列ソート算法を評価するというものである．

この概説論文からすぐに得られる結論を述べよう．並列ソートの研究は大概，ソート算法の理論的な計算時間を改良する方法の発見に集中されており，他の側面（たとえば技術的な制約やデータへの依存性）が比較的なおざりにされてきた．典型的なものとしては，並列性や記憶容量になんの制限もない仮想的な計算機で漸近的に最小時間をもつ算法の開発である．現在では，ネットワーク型でも共有メモリ型でもソートの計算量についてはかなり良く理解されていると思われる．並列ソートの計算量について残されている研究課題は，主として，チップ面積と時間を結びつける VLSI 計算量に関する新しいモデルに関するものである[Thompson 1983]．

この10年間は，主として並列ソートの計算量を理論的に研究してきたので，今後は，現在あるいは近い将来の技術を前提として，並列ソートの具体化の可能性に関する側面が組織的に研究されることになると思われる．並列ソートが実際上重要なのは，最初の頃の並列ソート算法が，ハードウェアの問題を解くためのものであったことをみればよくわかる．その問題とは，逐次的なソートより小さい遅延時間で，n 本の入力線のすべての順列を作るスイッチ・ネットワークを設計することである．数多くの高速並列ソート算法が知られているので，これらが並列計算の現実的なモデルに適用できるかどうか調べることは，重要であると思われる．特に，今後一層研究しなければならないのは，制約付きの並列性（問題の大きさに対する処理装置数に関する制約を取り除く），部分的な情報の伝達（同じ記憶場所から同時読出しをやめる），メモリの競合の解決などに関連するものである．

もう1つの重要な問題は，これまで並列ソート算法を評価するのに用いてきた基準を再考することである．明らかに，通信，入出力コスト，ハードウェアの複雑さをコストの包括的なモデルに取り入れるべきである．それによって，並列処理装置のさまざまなアーキテクチャを広く包括できる一般的なモデルが作られることになる．特に，入力データを処理装置のメモリに読み込むための初期設定コストは，従来大概の場合，並列ソートの研究で無視されてきた．逐次的な内部ソート算法ではこれを無視しても差支えないが，並列処理では，状況がまったく別である．処理装置が1台の場合，入力データは，逐次的にメモリに読み込まれる．並列処理では，複数の処理装置が同時に読み書きできる可能性がある．たとえば，ILLIAC-IV 計算機では，64個の処理装置すべてが並列的に入出力をするために，固定ヘッドのディスクが使われた．しかし，はるかに多くの処理装置が使われる場合，そのごく一部しか並列的に入出力を実行できない．それで，並列内部ソートに対しては，算法の評価にデータを読み書きするためのコストを考慮すべきである．特に，データをメモリに読み込むのに $O(n)$ 時間かかる場

表1　並列ソート算法の処理装置数と計算時間

算法	処理装置数	計算時間
奇偶転置	n	$O(n)$
Batcher の双単調	$O(n \log^2 n)$	$O(\log^2 n)$
Stone の双単調	$n/2$	$O(\log^2 n)$
網目状接続双単調	n	$O(\sqrt{n})$
Muller-Preparata	n^2	$O(\log n)$
Hirschberg(1)	n	$O(\log n)$
Hirschberg(2)	$n^{1+1/k}$	$O(k \log n)$
Preparata(1)	$n \log n$	$O(\log n)$
Preparata(2)	$n^{1+1/k}$	$O(k \log n)$
Ajtai 他	$n \log n$	$O(\log n)$

合，$O(\log n)$ 時間しかかからない並列ソート算法を使っても意味がなかろう．入出力のコストをモデルに取り入れることは，大容量ファイルを並列的にソートする問題に対して一層本質的なものになる．データベース・システムにおけるファイルのソートは，重要なので，今後この方向でさらに研究されていくことが確実である．

参考文献

AJTAI, M., KOMLÓS, J., AND SZEMERÉDI, E. 1983. An $O(n \log n)$ sorting network. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing* (Boston, Apr. 25–27). ACM, New York, pp. 1–9.

ALEKSEYEV, V. E. 1969. On certain algorithms for sorting with minimal memory. *Kibernetica 5*, 5.

BANERJEE, J., BAUM, R. I., AND HSIAO, D. K. 1978. Concepts and capabilities of a database computer. *ACM Trans. Database Syst. 3*, 4 (Dec.), 347–384.

BATCHER, K. E. 1968. Sorting networks and their applications. In *Proceedings of the 1968 Spring Joint Computer Conference* (Atlantic City, N.J., Apr. 30–May 2), vol. 32. AFIPS Press, Reston, Va., pp. 307–314.

BAUDET, G., AND STEVENSON, D. 1978. Optimal sorting algorithms for parallel computers. *IEEE Trans. Comput. C-27*, 1 (Jan.).

BENTLEY, J. L., AND KUNG, H. T. 1979. A tree machine for searching problems. In *Proceedings of the 1979 International Conference on Parallel Processing* (Aug.).

BILARDI, G., AND PREPARATA, F. P. 1983. A minimum area VLSI architecture for $O(\log n)$ time sorting. TR-1006, Computer Science Department, Univ. of Illinois at Urbana-Champaign (Nov.).

BITTON, D., AND DEWITT, D. J. 1983. Duplicate record elimination in large data files. *ACM Trans. Database Syst. 8*, 2 (June), 255–265.

BITTON-FRIEDLAND, D. 1982. Design, analysis and implementation of parallel external sorting algorithms. Ph.D. dissertation, TR464, Computer Science Department, Univ. of Wisconsin, Madison (Jan.).

BORODIN, A., AND HOPCROFT, J. E. 1982. Routing, merging and sorting on parallel models of computation. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing* (San Francisco, Calif., May 5–7). ACM, New York, pp. 338–344.

BRYANT, R. 1980. External sorting in a layered storage architecture. Lecture. IBM Research Center, Yorktown Heights, N.Y.

CHEN, T. C., LUM, V. Y., AND TUNG, C. 1978. The rebound sorter: An efficient sort engine for large files. In *Proceedings of the 4th International Conference on Very Large Data Bases* (West Berlin, FRG, Sept. 13–15). IEEE, New York, 312–318.

CHUNG, K., LUCCIO, F., AND WONG, C. K. 1980. On the complexity of sorting in magnetic bubble memory systems. *IEEE Trans. Comput. C-29* (July).

DOHI, Y., SUZUKI, A., AND MATSUI, N. 1982. Hardware sorter and its application to data base machine. In *Proceedings of the 9th Conference on Computer Architecture* (Austin, Tex., Apr. 26–29). IEEE, New York, pp. 218–225.

EVEN, S. 1974. Parallelism in tape-sorting. *Commun. ACM 17*, 4 (Apr.), 202–204.

FENG, T.-Y. 1981. A survey of interconnection networks. *Computer 14*, 12 (Dec.).

FISHBURN, J. P., AND FINKEL, R. A. 1982. Quotient networks. *IEEE Trans. Comput. C-31*, 4 (Apr.).

GAVRIL, F. 1975. Merging with parallel processors. *Commun. ACM 18*, 10 (Oct.), 588–591.

HIRSCHBERG, D. S. 1978. Fast parallel sorting algorithms. *Commun. ACM 21*, 8 (Aug.), 657–666.

HSIAO, D. C., AND MENON, M. J. 1980. Parallel record-sorting methods for hardware realization. Tech. Rep. OSU-CISRC-TR-80-7, Computer and Science Information Dept., Ohio State Univ., Columbus, Ohio (July).

KNUTH, D. E. 1973. Sorting and searching. In *The Art of Computer Programming*, vol. 3. Addison-Wesley, Reading, Mass.

KUMAR, M., AND HIRSCHBERG, D. S. 1983. An efficient implementation of Batcher's odd-even merge algorithm and its application in parallel sorting schemes. *IEEE Trans. Comput. C-32* (Mar.).

LEE, D. T., CHANG, H., AND WONG, K. 1981. An on-chip compare/steer bubble sorter. *IEEE Trans. Comput. C-30* (June).

LEILICH, H. O., STIEGE, G., AND ZEIDLER, H. C. 1978. A search processor for database management systems. In *Proceedings of the 4th Conference on Very Large Databases* (West Berlin, FRG, Sept. 13–15). IEEE, New York, pp. 280–287.

LEISERSON, C. E. 1981. Area-efficient VLSI computation. Ph.D. dissertation. Tech. Rep. CMU-CS-82-108, Computer Science Dept.; Carnegie-Mellon Univ., Pittsburgh, Pa. (Oct.).

MULLER, D. E., AND PREPARATA, F. P. 1975. Bounds to complexities of networks for sorting and for switching. *J. ACM 22*, 2 (Apr.), 195–201.

NASSIMI, D., AND SAHNI, S. 1979. Bitonic sort on a mesh connected parallel computer. *IEEE Trans. Comput. C-27*, 1 (Jan.).

NASSIMI, D., AND SAHNI, S. 1982. Parallel algorithms to set up the Benes permutation network. *IEEE Trans. Comput. C-31*, 2 (Feb.).

PEASE, M. C. 1977. The indirect binary n-cube microprocessor array. *IEEE Trans. Comput. C-26*, 5 (May).

PREPARATA, F. P. 1978. New parallel sorting schemes. *IEEE Trans. Comput. C-27*, 7 (July).

PREPARATA, F. P., AND VUILLEMIN, J. 1979. The cube-connected-cycles. In *Proceedings of the 20th Symposium on Foundations of Computer Science.*

SELINGER, P. G., ASTRAHAN, M. M., CHAMBERLIN, D. D., LORIE, R. A., AND PRICE, T. G. 1979. Access path selection in a relational database system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Boston, Mass., May 30–June 1). ACM, New York, pp. 23–34.

SHILOACH, Y., AND VISHKIN, U. 1981. Finding the maximum, merging and sorting in a parallel computation model. *J. Algorithms 2*, 1 (Mar.).

SIEGEL, H. J. 1977. The universality of various types of SIMD machine interconnection networks. In *Proceedings of the 4th Annual Symposium on Computer Architecture* (Silver Spring, Md., Mar. 23–25). ACM SIGARCH/IEEE-CS, New York.

SIEGEL, H. J. 1979. Interconnection networks for SIMD machines. *IEEE Comput. 12*, 6 (June).

STONE, H. S. 1971. Parallel processing with the perfect shuffle. *IEEE Trans. Comput. C-20*, 2 (Feb.).

THOMPSON, C. D. 1980. A complexity theory for VLSI. Ph.D. dissertation, Tech. Rep. CMU-CS-80-140, Computer Science Dept., Carnegie-Mellon Univ., (Aug.).

THOMPSON, C. D. 1983. The VLSI complexity of sorting. *IEEE Trans. Comput. C-32*, 12 (Dec.).

THOMPSON, C. D., AND KUNG, H. T. 1977. Sorting on a mesh-connected parallel computer. *Commun. ACM 20*, 4 (Apr.), 263–271.

VALIANT, L. G. 1975. Parallelism in comparison problems. *SIAM J. Comput. 3*, 4 (Sept.).

VAN VOORHIS, D. C. 1971. On sorting networks. Ph.D. dissertation, Computer Science Dept., Stanford Univ., Stanford, Calif.

VISHKIN U. 1981. Synchronized parallel computation. Ph.D. dissertation, Computer Science Dept., Technion—Israel Institute of Technology, Haifa, Israel.

YASUURA, H., TAKAGI, N., AND YAJIMA, S. 1982. The parallel enumeration sorting scheme for VLSI. *IEEE Trans. Comput. C-31*, 12 (Dec.).