数値シミュレーションのための データベースアーキテクチャ[†*]

牧　野　武　則[††]

大規模な数値シミュレーションでは，大容量でかつ高速な外部メモリを必要とする．ここでは，数値シミュレーションを対象にした，高速データ転送と，柔軟なデータアクセスを可能にする，データベースアーキテクチャを提案する．このアーキテクチャは，線形置換と呼ぶ，$b+d\cdot i\rightarrow i \pmod m$ なる置換をベースに組み立てられ，大規模な並列メモリシステムを実現できる．ここで，b, d は定数，m は素数として選ばれるメモリモジュール数である．通常の差分解法は，線形置換により効果的にサポートされ，一方，ランダムシミュレーションについては，線形置換を用いた並列ソートによりサポートされる．

1. まえがき

ベクトルプロセッサに代表される超高速科学計算システムの登場により，大規模な数値計算が多分野で行われつつある．数値計算の大規模化は，計算速度の向上だけでなく，計算速度の向上に見合う I/O の性能向上を必要とする[1]．また，I/O 処理に伴う，プログラムコードの繁雑さの回避も望まれている．

I/O の高速化を目的として，商用のベクトルプロセッサでは，半導体メモリを使用した外部メモリを提供している．現在提供されている外部メモリシステムの多くは，基本的には，ディスクデバイスと同様，ブロックランダムアクセスのアーキテクチャが採用されている．この理由として，既存のプログラムの書き換え，大幅なオペレーティングシステムの変更を避けることがあげられる．しかし，ブロックランダムアクセスデバイスの使用は，プログラムコーディングを繁雑にするだけでなく，ブロックへのデータの格納順に適合しないデータ参照については十分な性能を引き出すことは困難である．

ブロックランダムアクセスデバイスのもとで，ブロックのフェッチ回数を最少にする研究もなされており[2]，このようなデバイスを効果的に使用する一つの方向を示している．しかし，固定したブロックという物理的な制約は避けられない．

最近のメモリ素子の大容量化は著しく，十数 G，数十 G バイトのメモリシステムの実現も可能になってきた．こうしたメモリシステムは，本来，ランダムアクセス可能であり，この特徴を生かしたアーキテクチャが考えられる．

この論文では，大規模な数値シミュレーション分野を対象にし，高速のデータ転送と柔軟なデータアクセスを目的とした，データベースアーキテクチャを提案する．このアーキテクチャは，高速処理のため，並列メモリシステムとデータ置換ネットワークから構成される簡素な構造をもち，大規模化が可能である．

はじめに，ここで提案するアーキテクチャの概要を説明し，目的と効果について述べる．つぎに，いくつかの代表的応用プログラムを例にとり，データベースと計算システムとの間での，データ置換のための演算について述べる．そして，それらの演算で基本となる線形置換（定義は後述）と並列ソートの方式について説明する．最後に，データベースシステムの構成について述べる．

2. アーキテクチャの概要

大規模な科学計算では，大きなメモリ領域を必要とするプログラムが少なくない．たとえば，3次元の偏微分方程式を差分で解く際，128^3 グリッドについては 2^{27} 語程度，数 G バイト，を必要とし，また，分子軌道法によるクロロフィルの解析では 100G バイトが必要と言われている[3]．このような大規模なプログラムは他の分野でも多く報告されている．

計算システムに入りきらないデータセットについては，外部メモリに格納され，計算の進行に合わせて，必要なデータのサブセットが，計算システムとの間で交換される．この交換を実現する方法として，外部メモリを，ファイル装置として見せるか，仮想メモリの

---

† Database Architecture for Numerical Simulation by TAKENORI MAKINO (C&C Systems Research Laboratories, NEC Corporation).
†† 日本電気（株）C&C システム研究所
* 本研究は工業技術院大型プロジェクト「科学技術用高速計算システムの研究開発」の一環として行われた大容量高速記憶装置の検討の一部である．

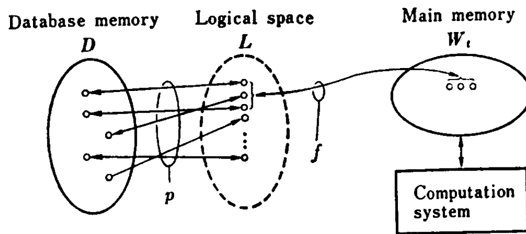

図 1 データ交換の概念図
Fig. 1 Conceptual outline of data exchange between database and main memory.
p: Permutation, f: Partial mapping

バックエンドデバイスとして見せることが考えられる.

しかし，科学計算では，後述するように，規則的なデータアクセスが期待され，また，どの時点で，どのデータサブセットを要求するかを前もって知ることができるケースが多いことから，オンデマンドにデータを要求する仮想メモリ方式や，ユーザが陽に I/O を指定するファイル方式よりも，外部メモリにインテリジェンスをもったデータベース方式のほうが有利である.

図1に，ここで提案するデータベースアーキテクチャにおける計算システムとデータベースの間のデータ交換を，概念的に示す．データセットは，データベースのメモリ上では，ある順序に従った物理構造で格納される．この構造を $\boldsymbol{D}$ と表す．一方，計算システムでのプログラムは，計算の進行上必要なデータベースのサブセット $\boldsymbol{W}$ をデータベースに要求する．この構造 $\boldsymbol{W}$ は，構造 $\boldsymbol{D}$ の部分写像であり，ここでは，この部分写像を二つの演算に分けて考える．この演算は，構造 $\boldsymbol{D}$ から構造 $\boldsymbol{L}$ への置換演算 p と，構造 $\boldsymbol{L}$ からサブセットを取り出し，構造 $\boldsymbol{W}$ にマッピングする部分写像演算 f である．ここで，演算 f には，置換演算は含まれないものとする．すなわち，データベースメモリから計算システムのメモリへの写像は，次のように表される.

$$\boldsymbol{W}=f(p\boldsymbol{D}).$$

構造 $\boldsymbol{W}$ は，ある時点での計算に必要なデータのサブセットであり，ここでは，データウィンドウあるいは簡単にウィンドウと呼ぶ．このウィンドウは，通常，Do ループにおいて，ループのインデックスを変数とし動かされ，時点 t におけるウィンドウを $\boldsymbol{W}_t$ と書く.

データウィンドウの動きを調べるため，簡単な行列 A_{ij} について考える．構造 $\boldsymbol{D}$ は，ベクトル $\{A_{00}, A_{01}, A_{02}, \cdots\}$ で与えられるとする．この行列について行方向の要素について計算が行われるとき，構造 $\boldsymbol{L}$ は $\{A_{00}, A_{01}, A_{02}, \cdots\}$ であり，ベクトル $\boldsymbol{L}$ の要素を L_i とすると，$\boldsymbol{W}_t=L_i|i=k_1, \cdots, k_2\}$ である．k_1 と k_2 は t により与えられる．一方，列方向の要素について計算が行われるとき，構造 $\boldsymbol{L}$ は，ベクトル $\{A_{00}, A_{10}, A_{20}, \cdots\}$ であり，$\boldsymbol{W}_t=\{L_i|i=k_1, \cdots, k_2\}$.

すべてのデータセットの構造 $\boldsymbol{D}$ は，ある順序で順序づけられる．ここでは，データの高速転送のため，並列メモリシステムを採用するため，データセットは，すべてベクトルとして格納されるとする．例では，構造 $\boldsymbol{D}, \boldsymbol{L}$ ともベクトルであり，$\boldsymbol{L}$ は $\boldsymbol{D}$ を置換して得られ $\boldsymbol{W}_t$ は，順序ベクトルである構造 $\boldsymbol{L}$ の部分順序ベクトルである.

基本的には，以上のように，データベースメモリから，計算システムへのデータ転送がなされる．また，計算システムのメモリからデータベースメモリへのデータの転送も，同様にしてなされる．すなわち，データウィンドウ $\boldsymbol{W}_t$ について，時点 t で定義されるレンジのベクトルについて，置換 p の逆置換 p^{-1} を行い，データベースメモリに書き込まれる.

データベースで用意すべきデータセットの置換演算 p は，すべての置換が可能なように設定されるのが理想だが，実現上のハードウェアや演算時間の制限のため，いくつかの演算に限定する．科学計算のように，応用を限定できる場合，演算の限定が可能であり，次の章では，いくつかの応用プログラムについて，用意すべき置換演算を議論する.

部分写像のための演算 f は，計算システムのメモリ上での，計算に許された領域のサイズに依存する．また計算システムのアーキテクチャ，とくにメモリシステムの構造にも依存する．計算システムの構造が，2次元アレイならば，演算 f は，構造 $\boldsymbol{L}$ のサブセットを2次元配置に写像することになる.

ここでは，議論を具体的にするため，計算システムとして，ベクトルプロセッサを想定する．また，前述したように，データベースのメモリ構造は，並列処理を行うため，並列メモリシステム，つまりベクトル構造とする．すなわち，ここで述べた，置換や部分写像の演算は，ベクトルに対する演算に限定される.

3. 応用プログラムの分析

ここでは，代表的な応用プログラムをいくつか例にとり，計算の構造とデータの構造から，計算システム

とデータベースとの間でのデータ交換を調べ，データセットの置換に必要な演算について述べる．

応用プログラムとして，偏微分方程式の差分解法と，プラズマ粒子コードにおけるパーティクルプッシュ部分の，二つのコードを取り上げる．これらのコードは，いずれも，3次元モデルについては，非常に大きなメモリ空間を必要とする．

まず，偏微分方程式の陽解法について述べる．簡単化した陽解法のプログラムの一部を，以下に示す．

```
Do 10 i=1, n1
Do 10 j=1, n2
U*(i, j)=F{U(i, j), U(i±1, j), U(i, j±1),
        U(i-1, j-1)}
10 Continue
```

ここで，U^*, U は，それぞれ新旧のグリッド変数，F は，陽解法の計算における関数を表す．

このプログラムで，最内側のループでは，入力として，$\{U(i,\ j), U(i\pm 1, j), U(i, j\pm 1), U(i-1, j-1)\}$ を要求し，$U^*(i, j)$ を出力する．これらが，この計算の最少のウィンドウであり，ウィンドウの構造も示す．

いま，入力データのウィンドウに許されたサイズが $m\cdot n_2$ ならば，ウィンドウ w_t の内容は，

$$\{U(i,\ j), U(i\pm 1, j), U(i, j\pm 1), U(i-1, j-1)|\ j=1, \cdots, n_2;\ i=k_1, \cdots, k_2\},$$

である．k_1, k_2 は t の関数であり，それぞれ，$k_1=m\cdot t+1$, $k_2=m(t+1)$ である．ただし，$t=0, 1, 2, \cdots$.
このウィンドウには，要素の重複があるため整理すると，

$$\{U(i, j)|\ j=0, \cdots, n_2+1;\ i=k_1-1, \cdots, k_2+1\},$$

と表せる．実際には，$j=0$, $i=0$ はインデックス値として存在しないが，便宜上，記述している．

陽解法では，新しいグリッドは，前の時点のグリッドの値を用いて計算されるため，各グリッドは独立に計算できる．しかし，陰解法では，スキャンする軸と対応したデータの交換が必要となる．以下に，陰解法である AFI 法[4] に沿って説明する．

AFI 法では，2次元の非線形問題は，

$$\left(I+\frac{\Delta t}{2}\frac{\partial}{\partial x}A^n\right)\left(I+\frac{\Delta t}{2}\frac{\partial}{\partial y}B^n\right)U^{n+1} =\left(I-\frac{\Delta t}{2}\frac{\partial}{\partial y}A^n\right)\left(I-\frac{\Delta t}{2}\frac{\partial}{\partial y}B^n\right)U^n$$

と差分形式は因数分解され，まず x 軸に沿ってスキャンし，軸ごとに連立方程式を解き，ついで y 軸について同様の計算を行う．したがって，軸単位の計算が本算的であり，データベースとのデータの交換は，軸データが単位となる．また，このスキャン軸は，前半と後半で入れ換わる．**図2**に，AFI 法における，転送データセット，ウィンドウの動きを示す．ウィンドウの重なりは，陽解法でのウィンドウの構造で述べたインデックス関数 $(i\pm 1)$ や $(j\pm 1)$ による．

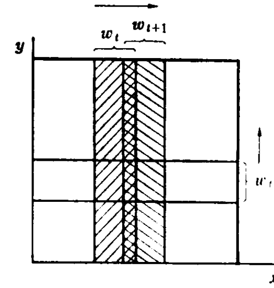

図 2 AFI 法におけるウィンドウの動き
Fig. 2 Data window in implicit method of partial differential equation.

陰解法のようにスキャン軸が入れ替わる例は，数値計算では一般的である．たとえば，多次元 FFT を始め，連立方程式の解法（CG 法やガラス消去法）や一般の行列の計算（行列積）等に現れる．

つぎに，プラズマ粒子輸送コード[5] について述べる．このコードでは，10^9 以上の粒子についてのシミュレーションが望まれており，ギガオーダのメモリ空間を必要とするため，外部メモリに粒子データが置かれることになる．したがって，性能を保証するためにはソートをするなどの工夫が必要なプログラムの例である．ここでは，外部メモリ側でデータセットをソートすることにより，こうしたコードの高速計算が実現できること，およびベクトル計算が都合良く行えることを示す．

図3(a)に，1次元の粒子コードのパーティクルプッシュ部分を示す．このプログラムで，インデックス変数 k は，Do ループ内にあるインデックス変数 ix や ir とは独立なため，このままでは直接参照によるベクトル化はできない．Sinz[6] は，ソートを用いてベクトル化を行う方法を提案している．Sinz のアルゴリズムを使用するように書き直したプログラムの例を図3(b)に示す．このプログラムには，ランニングサム等，通常のベクトルプロセッサに提供されていない演算（枠で囲んだステーメント）が含まれている．これらの演算については，文献 6) に詳細が述べられている．すなわち，図3 b)において，$ex(ix)$ を $ex(ix(k))$ とインデックス k についてのベクトルに $ex(ix)$ を拡張することで，ステートメント 050 のよ

```
010        do 10 k=1, n
020        ix=x(k)
030        vx(k)=vx(k)+ex(ix)+…
040        x(k)=x(k)+vx(k)
050        ir=x(k)
060        rx1=x(k)-float (ir)
070        rh(ir)=rh(ir)+1-rx1
080   10   rh(ir+1)=rh(ir+1)+rx1
```

(a) Original code

```
010        [sort (x(k))]
020        do 10 k=1, n
030        ix(k)=x(k)
040        [ex(k)=ex(ix(k))]
050        vx(k)=vx(k)+ex(k)+…
060   10   x(k)=x(k)+vx(k)
070        [sort (x(k))]
080        do 11 k=1, n
090        ir(k)=x(k)
100   11   rx(k)=x(k)-ir(k)
110        [running sum (rx(k), rs1(ir(k))) ]
120        [running sum (1-rx(k), rs2(ir(k)))]
130        do 12 i=1, nr
140        rh(i)=rh(i)+rs2(i)
150   12   rh(i+1)=rh(i+1)+rs1(i)
```

(b) Parallel representation code

図 3 1次元プラズマ輸送コードのパーティクルプッシュ部

Fig. 3 Particle push part in plasma simulation.

うに直接参照でのベクトル計算を可能にする．また，ステートメント 110, 120 で，拡張されたベクトルをランニングサムによりインデックス ir に関するベクトルにすることにより，パーティクルプッシュを行う．また，この処理を行うためソートを行うことにより，それらの演算を効率良く実行できる．さらに，ソートを外部メモリ側で行えば，無駄な I/O を避けることができる．

このように一見ランダムなデータ構造をもつ，ランダムシミュレーションと言われているコードも，ソートを導入すれば，順序ベクトルとして扱うことができる場合がある．たとえば，スパース行列の計算では，インデックスの値によりソートすれば，無駄な I/O を避けることができる．また，分子軌道法の計算[7] においてもソート処理は重要である．

以上の検討からデータベースのもつべき機能として，

- 多次元アレイについて異なった軸によるデータアクセスが柔軟に行えること，
- ランダムなデータ構造をもつシミュレーションに対して強力なソート機構が用意されていること

等があげられる．

ここでの検討は，数値シミュレーションの一部に対するものであり，全体に対する結論を与えるものではない．しかし，少なくとも，偏微分方程式の差分解法やプラズマ粒子輸送コード，およびそれらに類する計算については，以上のようにまとめられる．

4. 線形置換

提案するデータベースアーキテクチャの基本となる線形置換について述べる．この置換および置換ネットワークの構成については，文献 8)，9) に詳しく述べているので，要旨のみを説明する．

線形置換の定義は，

$b+d\cdot i\rightarrow i \pmod{m}$

であり，この置換を，距離 d の線形置換と呼ぶ．ここで，b は，オフセット，m は，後述するが，メモリモジュールの個数である．

ここでは，線形置換により，前章で述べたプログラムにおける置換演算が表されることを述べる．例として，次の3次元アレイを考える．

dimension $A\ (n_1, n_2, n_3)$

このアレイは，データベースのメモリ上では，順序ベクトルとして，次のように格納されるとする．

$A(1,1,1), A(1,1,2), \cdots, A(1,2,1), A(1,2,2), \cdots, A(2,1,1), \cdots, A(n_1, n_2, n_3).$

つぎに，Do ループの中に，その Do ループの制御変数 i の関数として，A のインデックスが現れるときの，距離 (distance)，すなわち，上のベクトルで参照される要素間の距離，はつぎのように与えられる．ただし，k_1, k_2, k_3 は i に依存しないとする．

$A(k_1, k_2, i)$: distance$=1$,

$A(k_1, i, k_3)$: distance$=n_3$,

$A(i, k_2, k_3)$: distance$=n_2\cdot n_3$

$A(k_1, i, i)$: distance$=n_3+1$

$A(k_1, a_2\cdot i, k_3)$: distance$=a_2\cdot n_3$

⋮

一般に，r 次元のデータアレイについて，第 k 番目のインデックス関数が，i に関し，1 次式で表せ，それを，$a_k\cdot i+b_k$ とすれば，

$$(\text{distance})=\sum_{k=1}^{r} a_k \prod_{j=0}^{k-1} n_j$$

ただし，$n_0=1$ である．

このように，変数のインデックス関数が，Do ループの制御変数の一次式で表せる場合，(distance) を与えることができる．

あるデータセットは，データベースのメモリ上ではすべて，連続したアドレスをもつベクトルとして表現される．そして，このベクトルは，メモリモジュールに沿って，要素0はモジュール1，要素1は2，というように格納される．あるデータセットに対するベクトルの先頭アドレスを (base) とし，メモリモジュールの数を m とすると，ペア (b, d) は，

$b=(\text{base}) \bmod m$,

$d=(\text{distance}) \bmod m$

と与えられる．b, d はメモリモジュール番号に対応する．

すなわち，(b, d) のペアで制御される線形置換は，アレイのインデックス関数が，Do ループの制御変数の一次式で表される場合，そのアレイに対するベクトル要素の置換を行うことができる．

線形置換のためのネットワーク[8] は，b に対応して回転置換を行う部分と，d に対応してスキップ置換を行う部分から構成される．このネットワークは，2入力セレクタを単位ゲートとすると，M 要素について，$2M \log_2 M$ のオーダのゲートで構成でき，かつ，制御もペア (b, d) だけによるためきわめて簡素である．したがって，M が 1,000 のオーダ程度の大規模なネットワークを，現在の LSI 技術で，実現することができる．

5. 並列ソート

前述したように，ランダムシミュレーションの実行には，ソート処理が有用である．一般にソート処理はコストがかかり，またソートされるデータセットの全体をなん度も参照する．このため，計算システムでソートを行うより，データセットを保持しているデータベース側でソートするほうが望ましい．

並列ソートのアルゴリズムは，Batcher のバイトニックソート[10]が知られている．ここでは，このバイトニックソートをベースに，線形置換を使用した並列ソートの方式について述べる．

バイトニックソートは，バイトニック列 $\{a_0, a_1, \cdots, a_{n-1}\}$，ここで，$0 \leq i < n/2$ については $a_i \leq a_{i+1}$, $n/2 \leq i < n-1$ については $a_i \geq a_{i+1}$, を入力とし，ソートされた列を出力する．要素数が8，$n=8$，のときのアルゴリズムを図4に示す．

このアルゴリズムの構造から推察できるよう，要素数が固定されている場合，そのアルゴリズムは，ハードウェアロジックとして実現できるが，任意の要素数

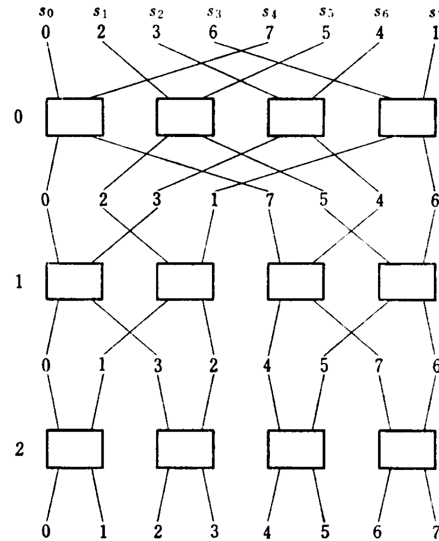

図 4 バイトニックソートアルゴリズム
Fig. 4 Batcher's bitonic sort algorithm for 8 elements.

については，たとえばベクトル化しようとしたとき，FFT と同じように，ベクトルの圧縮といった特別の処理が必要になる．

なお，バイトニックソートの複雑さは，要素数 n について，$O(n \log^2 n)$ であり，ヒープソート等，シリアル処理のソートは $O(n \log n)$ であるのに対し，全体の処理量は多くなるが，手続きが簡単なこと，並列処理が行える点で高速なソートを実現できる．

まず，図4に示した例に沿って線形置換によりバイトニックソートを実現する方法について述べる．アルゴリズムは，3段の並列に配置された互換セル (permutation cell) とその間の相互結合から構成されている．入力されたバイトニック列 $\{0, 2, 3, \cdots, 1\}$ は，このアルゴリズムにより，出力としてソートされた列 $\{0, 1, \cdots, 7\}$ が得られる．

バイトニック列 $\{0, 2, 3, \cdots, 1\}$ はメモリに順序ベクトルとして格納される．このベクトルから，各互換セルに対し，線形置換によりメモリ競合なしに比較ペアを送らなくてはならない．第1段目では，$\{0, 2, 3, 6\}$ と $\{7, 5, 4, 1\}$ のベクトルを，第3段目では $\{0, 3, 4, 7\}$ と $\{1, 2, 5, 6\}$ のベクトルペアを，各互換セルに送ればよい．一方，第2段目については，線形置換でデータ参照するとすると，$\{0, 2\}$ と $\{3, 1\}$ および $\{7, 5\}$ と $\{4, 6\}$ の二つのベクトルペアを作るか，$\{0, 7\}$ と $\{3, 4\}$ および $\{2, 5\}$ と $\{1, 6\}$ の二つのベクトルペアを作る二つの方法が考えられる．前者は第1段目と

```
010      /*N; bitonic sequence length
020      NL=log N;                    /* number of levels
030      do L=1, NL;                  /* level number
040        K=N/2^L;
050        if (L<LM)
060          then do J=1, 2^(L-1);          /* algorithm a)
070                 do all I=2K(J-1)+1, 2K(J-1)+K;
080                   compare & exchange (S(I), S(I+K));
090                 do end;
100                 do end;
110          else do J=1, K;                /* algorithm b)
120                 do all I=J, 2K2^(L-1)+J, 2K;
130                   compare & exchange (S(I), S(I+K));
140                 do end;
150                 do end;
160      do end;
```

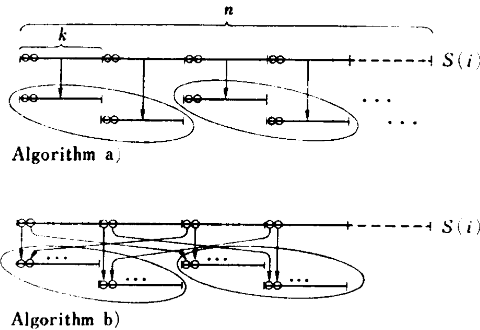

図 5 線形置換によるバイトニックソートアルゴリズム
Fig. 5 Parallel algorithm for bitonic sort using linear permutation.

同じアルゴリズムによるもので，後者は，第3段目と同じである．

要素 N の場合について，**図5**に，線形置換によるバイトニックソートのアルゴリズムを示す．ここで，レベルが小さい場合，アルゴリズムa）を，大きい場合，アルゴリズムb)を使用し，変数 LM により，その選択が行われる．

いま，バイトニック列 $\{a_0, a_1, \cdots, a_{n-1}\}$ が与えられたとき，$L=1$ では，$\{a_0, \cdots, a_{n/2-1}\}$ と $\{a_{n/2}, \cdots, a_{n-1}\}$ が，互換セルに送られる．この互換セルの数 p が $n/2$ よりも小さければ，数回に分けて互換が行われる．$L=2$ では，$\{a_0, \cdots, a_{n/4-1}\}$ と $\{a_{n/4}, \cdots, a_{n/2-1}\}$ が互換セルに送られ，つぎに $\{a_{n/2}, \cdots, a_{3n/4-1}\}$ と $\{a_{3n/4}, \cdots, a_{n-1}\}$ が互換セルに送られる．このときベクトルペアのベクトル長は $n/4$ である．L が大きくなると $n/2^L$ であり，L とともにベクトル長が短くなる．ある L に達すると，アルゴリズムb）を選び，ベクトルペアとして $\{a_0, a_{2k}, a_{4k}, \cdots, a_{2hk}\}$ と $\{a_k, a_{3k}, \cdots, a_{(2h-1)k}\}$ および $\{a_1, a_{2k+1}, a_{4k+1}, \cdots, a_{2hk+1}\}$ と $\{a_{k+1}, \cdots, a_{(2h-1)k+1}\}, \cdots$ を作成するようにする．ここで k は，$k=n/2^{L-1}$ であり，$h=2^{L-1}$ である．したがって，L が大きくなるとベクトル長 k は大きくなる．

アルゴリズムa）とb）で，ベクトル長が等しくなるのは，$L=\log_2 n/2$ であり，LM は $\log_2 n/2$ として選べばよい．

このアルゴリズムが最も効率良く働くには，各段で作成されるベクトルペアのベクトル長が，並列に配置された互換セルの数よりも長いことが必要である．いま互換セルの数を 2^q とすると，要素数 n は 2^{2q} 以上の場合である．つまり，互換セルの数を1,024とすると要素数は1メガ個以上の場合である．ここで検討しているデータベースでは，本来，巨大なデータセットを対象にしているため，要素数が 10^6 以上で最もよい効率を示すことは不自然ではない．無論，それ以下の要素数では，要素当りのソート時間が長くなるが，極端に長くなるわけではない．

以上，バイトニックソートのアルゴリズムについて説明した．ソートは，このバイトニックソートを使用して行われる．まず，与えられた順序付けされていない要素列，要素数 n, k について，まず長さ2のバイトニック列を作成し，つぎに長さ4，長さ8,… を作成し，最後に長さ n のバイトニック列についてバイトニックソートを行うことで全体のソートが達成される．

ここでは，バイトニックソートが線形置換により実行することができ，十分に大きな要素数について効率良くソートを行えることとを示した．

6. システムの構成

図6に数値シミュレーションを対象としたデータベースシステムの構成を示す．このシステムは，m 個の並列メモリモジュールと，並列メモリと計算システムとの間およびバイトニックソータとの間でのベクトル置換を行う二つの線形置換ネットワークおよびそれらを制御する制御プロセッサから構成される．

並列メモリモジュールの個数 m は，メモリ競合を避けるため，素数から選ばれる．この個数は，計算システムとの間のデータ転送率により与えられるが，要求されるデータ転送率を1G 語/秒とすると，使用するメモリデバイスのサイクル時間が数百ナノ秒ならば，数百〜1,000 程度を想定する必要がある．

このデータベースメモリからのベクトルの読み出しは，回転置換 R とスキップ置換 S から構成される線形置換ネットワークを介して行われ，書き込みは，ス

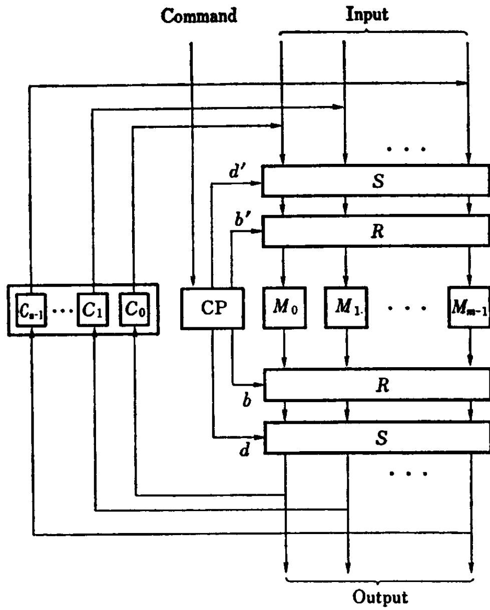

図 6 データベースシステムの構成
Fig. 6 Database system configuration.
CP: Control processor, M_i: Memory module, C_i: Permutation cell for bitonic sort, R, R': Rotation permutation network, S, S': Skip permutation network.

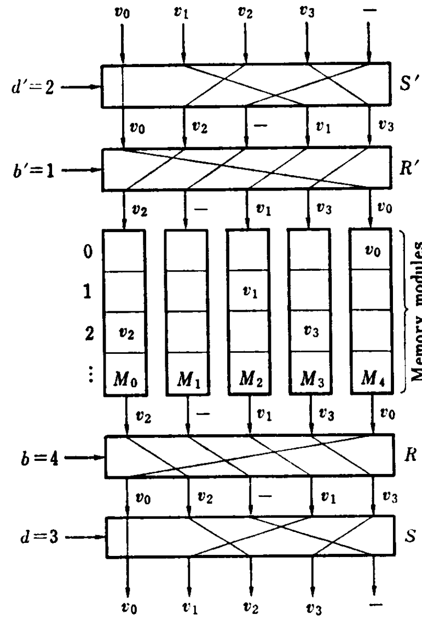

図 7 線形置換によるベクトル要素の置換
Fig. 7 Vector permutation using linear permutation network.

キップ置換 S' と回転置換 R' から構成される線形逆置換ネットワークにより行われる．ともに，ペア (b, d) により制御され，R と R' は b に，S と S' は d に対応して転送パスが設定される．

制御プロセッサは，command (base, distance, length) の形式で，入出力命令を受け取り，write ならば，まず，(base, distance) から，

$b=(\text{base}) \bmod m,$

$d=(\text{distance}) \bmod m$

が計算され，つぎに，逆置換を行うために，

$b'=m-b \pmod m,$

$d \cdot d'=1 \pmod m$

から，(b', d') を得，R' と S' を設定する．

例として，$m=5$ とし，図7に示すようにベクトル $\{v_i\}$ が格納されているとする．いま，(base)=4, (distance)=3 の要点を書き込むとする．すなわち，$\{v_0, v_1, v_2, v_3, -\}$．$b=4$, $d=3$ から，$b'=1$, $d'=2$ である．まず，スキップ置換 S により，$\{v_0, v_2, -, v_1, v_3\}$ が得られ，つぎに回転置換 R' により $\{v_2, -, v_1, v_3, v_0\}$ と並べ換えられる．v_0, v_1, v_2, v_3 は，それぞれ，モジュール 4, 2, 0, 3 に入っており，所定の置換が行える．

書き込む場合，まず要素のモジュール内アドレス文を計算する．すなわち，第 i 番目の要素のアドレスは，

$\lfloor((\text{base})+(\text{distance})*i)/m\rfloor.$

ここで，$\lfloor x \rfloor$ は x より小さく最大の整数を表す．このようにして計算されたアドレスと書き込み指令，および書き込むべきデータ要素は，ベクトルとして，線形逆置換ネットワークに送られ，所定の置換が行われたあと，各メモリモジュールに送られ，書き込まれる．

読み出す場合は，まず，書き込みの場合と同様に，線形逆置換ネットワークを介して，各アドレスと読み出しの指令を，各メモリモジュールに送り，データを読み出す．読み出されたデータはベクトルとして，線形置換ネットワークに送られる．この線形置換ネットワークは，制御プロセッサにより送られているパラメータ (b, d) により，置換パスが設定されている．そのパスにより，ベクトルデータは置換され，計算システムに送られる．

前の例を使って，読み出しの場合を説明する．まず書き込みの場合と同様にして，読み出し指令と各要素のモジュール内アドレスが，S' と R' を介し，各メモリモジュールに送られ，要素値が得られる．図7のように，$\{v_2, -, v_1, v_3, v_0\}$．このベクトルは，$R$ によ

り，$b=4$ に従い，回転置換され，$\{v_0, v_2, -, v_1, v_3\}$ が得られ，さらに，S により，$d=3$ に従い，スキップ置換され，$\{v_0, v_1, v_2, v_3, -\}$ と順序ベクトルが得られる．

以上のようにして，線形置換が行われる．なお，メモリモジュールの数 m と d が互いに素であれば，メモリ競合が生じないことが保証されている[9]ので，線形置換ネットワークやメモリモジュールに，メモリ競合を解決するための特別の機構は必要がなく，簡素に実現できるだけでなく，制御も簡単となる．

ソータは，メモリモジュールの数 m よりも小さい2のべき乗個の互換セルから構成される．前章で説明したように，並列の互換セルに対して，メモリモジュールからベクトルとして，比較ペアを送る．このベクトルの読み出し，書き込みは (b, d) で与えられ，要素間の距離 d は，2のべき乗である．したがって，線形置換ネットワークにより，メモリ競合なしに，各互換セルにデータが供給される．メモリからのベクトルの読み出し，書き込みおよび各互換セルの制御は，5章で述べたアルゴリズムに従い，制御プロセッサが実行する．なお，各互換セルには，比較・交換すべきベクトル要素のペアを格納する二つのバッファをもつ．

互換セルの数を p とし，要素数を n とすると，並列ソートの実行時間は，$n \log_2 n \cdot t/p$ である．ここで t は，メモリからの要素の読み出し書き込み時間であり，0.5 μsec のオーダとする．$p=1,024$ の場合，10^6 要素について 0.2 秒，10^9 要素については，450 秒程度の実行時間でソートできる能力をもっている．

以上に述べたように，線形置換とソート処理が実行される．まだ議論していない課題として，計算システムのメモリアロケーション，読み出し，書き込みデータセットの無矛盾性の保証等，重要な問題が残っている．この問題については，この論文では直接扱わず，プログラムの問題，すなわちプログラム側で責任をもつものとする．とくにクリティカルな場合は，read/write window に伴う非同期処理であり，差分解法では，時刻 n で計算されたデータセットは，時刻 $n+1$ の計算が始まる前にデータベースに書き戻されていなければならない．

7. まとめ

大規模な数値シミュレーションにおけるプログラムの構造を分析することにより，それらのプログラムの実行を効果的にサポートするデータベースアーキテクチャを提案した．

偏微分方程式の差分解法や一般の連立方程式，FFT では，ベクトルやアレイデータを対象とし，アレイデータを異なった軸でスキャンしたり，等間隔離れた要素並びを参照する．また，プラズマ粒子輸送コードでは，ソートにより粒子を並べ換えることで，並列計算を可能にすることができる．

このデータベースアーキテクチャは，次の線形置換と呼ぶ置換により組み立てられる．

$$b+d\cdot i \rightarrow i \pmod{m}$$

ここで，b, d は定数である．この置換はベースアドレス b と，要素間距離 d のベクトルを順序ベクトルに並べ換える．

データベースメモリには，すべてのデータセットはベクトルとして格納され，異なった軸でのアレイのスキャンは，要素間距離 d の要素並びの参照となり，線形置換によりサポートされる．また，ソート処理は，線形置換を用いたバイトニックソートで並列に実行でき，プラズマ粒子輸送コードのようなランダムシミュレーションを効果的にサポートできる．

このように，線形置換をベースにしたデータベースアーキテクチャによって，数値計算の広い分野の応用プログラムに適合した，計算システムとデータベースメモリとの間のデータ変換を実現でき，I/O に係るプログラミングの繁雑さの回避も期待できる．

また，線形置換を行うネットワークおよび並列ソートを実行するバイトニックソートも簡単なハードウェア構成，制御で実現できるため大規模化が可能であり，科学計算システムの高速化に伴う I/O ネックの解消に一つの方向を与える．

残された課題として，データベースとしての諸機能の実現方法，プログラムとの具体的なインタフェースの設定等がある．また，数値シミュレーションの他の多くの応用プログラムについて，このアーキテクチャの有用性を検証することも必要である．データベースの機能として，リレーショナルデータベースで定義されているような論理演算や，データベース管理機構がある．これらは，数値計算においても重要であり，興味あるテーマである．また，プログラムとのインタフェースについては，データベースシステムが用意する機能とともに，プログラム言語との関係を考慮し設定される．

ここで提案したデータベースアーキテクチャは，単

に数値計算における I/O のボトルネックの解消だけでなく，新プログラム言語を含む将来の数値計算システムアーキテクチャの構築への道を示すものである．

謝辞　最後に，本研究の機会を与えていただいた，当研究所の三上徹所長代理，千葉成美部長，そして活発な議論をいただいた富士通(株)の三浦謙一，(株)日立の梅谷征雄両氏を始めとするスーパーコンピュータ開発組合の皆様に，また丁寧な査読をいただいた査読者に謝意を表します．

参考文献

1) 津田，巽：メモリの階層性とベクトル計算機の実効性能，情報処理学会論文誌，Vol. 25, No. 1, pp. 37-45 (1984).
2) Tsuda, T., Sato, T. and Tatsumi, T.: Minimizing Page Fetches for Permuting Information in Two-Level Storage, Part 1. Generalization of the Floyd Model, *J. Inf. Process.*, Vol. 6, No. 2, pp. 74-77 (1983).
3) 柏木：スーパーコンピュータとその分子科学への応用，分子科学研究所電子計算機センター，分子研研究会報告書 (1982).
4) Beam, R. M. and Warming, R. F.: An Implicit Factored Scheme for the Compressive Navier-Stokes Equations, *AIAA J.*, No. 16, pp. 393-405 (1978).
5) 阿部：プラズマ数値シミュレータに関するワークショップ報告，核融合研究別冊，Vol. 44/その3, pp. 63-71，名大プラズマ研 (1980).
6) Sinz, K.: Optimal Use of a Vector Processor, *COMPCON 80*, Spring, pp. 277-281 (1980).
7) Takada, T. and Sasaki, F.: Improved Transformation Algorithm of Two-Electron Integrals from Atomic Orbital Basis to a Symmetry Orbital Basis, *Int. J. Quantum Chemistry*, Vol. XVIII, pp. 1157-1163 (1980).
8) 牧野：並列プロセッサシステムにおける d-順序ベクトルに対するメモリープロセッサ結合，情報処理学会論文誌，Vol. 23, No. 3, pp. 251-259 (1982).
9) Makino, T.: A Rotation and Skip Permutation Network for Parallel Processor Systems, *NEC R & D*, No. 68, pp. 101-110 (1983).
10) Batcher, K. E.: Sorting Networks and Their Applications, SJCC, pp. 308-314 (1968).

(昭和59年8月24日受付)
(昭和60年2月21日採録)