ベクトル型データベースプロセッサ IDP†

小島啓二†† 鳥居俊一†† 吉住誠一††

Integrated Database Processor (IDP) は汎用計算機の付加機構として開発された関係データベース処理専用プロセッサである．IDP の特徴は，①大容量主記憶指向，②データベース向き拡張ベクトル・アーキテクチャの二点にある．IDP は検索対象テーブルを主記憶上にベクトル形式で展開し，パイプライン処理して検索を高速化する．IDP の拡張ベクトル演算機構は，従来ベクトル化が困難であった探索，ソート，結合，集合演算等のベクトル化を可能とする．これらの演算は IDP により，通常のスカラ処理に比べ10倍以上高速化されることを実測により確認した．本論文では，IDP の設計思想，アーキテクチャ，実現方式，および性能評価結果について述べる．

1. はじめに

関係データベースは平明な論理構造と強固な数学的基盤を背景に幅広い注目を集めてきた．1970年代の理論研究から80年代初頭の実験システム開発を経て，今日その実用期を迎えつつある．しかしながら，関係データベースは従来の構造型データベースに比べ処理負荷が重く，性能向上が最重要課題となっている．

この課題を専用アーキテクチャにより解決しようとする試み，すなわちデータベースマシンの研究開発が活発に行われてきた[1]．既に商用マシンの開発例もあるが，なおいくつかの基本的課題が残されていることも事実である．そのうちの一つに I/O 隘路の問題がある．実際，CPU の演算速度と磁気ディスクの I/O 速度のギャップは拡大する一方であり，データベースマシンの問い合わせ処理の大半はディスク I/O が占めるといっても過言ではない．

この問題に対し，二つのアプローチが有望視されている[2]．一つは小規模なディスク装置を多数台用意し，これらを並列アクセスして I/O 速度の向上をはかる方式である．この考え方にそったデータベースマシンとしては，GRACE[3]，DBC/1012[4]，GAMMA[5] などがあげられる．

一方，急速な半導体技術の進展を背景に，主記憶装置のような大容量の半導体メモリ上にデータベースの一部を常駐する方式も現実的なものになりつつある[6]．著者らはこの主記憶データベースのアプローチを基本に，さらにスーパコンピュータとして効果をあげているベクトル処理の考え方を導入した新しい関係データベースシステムのアーキテクチャを提案した[7]．本方式は検索対象テーブルをベクトル形式で主記憶上に動的に展開し，これをパイプライン処理して高速な関係演算の実現を目指すものである．

しかしながら従来の数値計算用のベクトル処理装置では，探索演算やソート演算などの関係データベースにおける重要な基本演算のベクトル化が困難であった．われわれはこの問題に対処するため，データベース処理向きの新しいベクトル処理装置，Integrated Database Processor (IDP) を日立製作所の大型汎用計算機である M-680 H の付加機構として開発した．IDP は昭和61年末より出荷が開始されており，ユーザサイトでの使用実績が積まれている．

以下本論文では，IDP の基本思想（第2章），アーキテクチャ（第3章），実現方式（第4章），および性能評価結果（第5章）について論じる．

2. 基本思想

2.1 データベースマシンにおける処理単位問題

ソフトウェアで実現された従来の関係データベースシステムの多くはテーブルの横一行（以下レコードと呼ぶ）を処理の単位（粒度）としている．ディスク上の物理的データ構造についてみると，System R[8] にみられるようにページ内にレコードを単位として格納する形式をとっている．また内部の処理もすべて1時点1レコード (one record at a time) を原則とする．つまり各プログラムモジュールは単一のレコードのみを操作するよう設計されており，システムは1レコードずつ逐次的に処理を進めている．

レコード単位処理方式の利点として以下があげられる．

（1） 数キロバイト程度の主記憶さえあれば任意の

† IDP—A Main Memory Based Vector Database Processor by KEIJI KOJIMA, SHUN'ICHI TORII and SEIICHI YOSHIZUMI (Central Research Laboratory, Hitachi, Ltd.).

†† 日立製作所中央研究所

大きさのデータベースが処理できる．

（2） 少量のレコードを検索する場合，適当なインデクスを用意することによってディスク I/O 回数を削減できる．

（3） レコードをそれへのポインタで管理する形式をとれば，レコードを連続配置する必要がなくなるため，レコードの追加や更新あるいは削除が容易となる．

（4） レコードやページをロック単位とした並列度の高いトランザクション制御が可能となる．

逆に欠点としては以下があげられる．

（1） 問い合わせ処理の中で大量のレコードアクセスが必要な場合には，不連続なディスク I/O が多発し平均ディスクアクセス速度が大幅に低下する．またレコードの型のチェックやモジュール間遷移処理など，レコード数に比例するオーバヘッドのため，CPU 処理時間も増大する．

（2） 論理的には単純なはずのテーブル構造がポインタを多用したチェイン構造で実現されることになり，処理の複雑化を招く．

（3） 専用ハードウェアを用いたパイプライン処理あるいは並列処理の適用が困難である．

すなわちレコード単位処理方式は各トランザクションが比較的少数のレコードを処理する場合に適した方式であるといえる．

一方データベースマシンに目を転じると，その多くはテーブル結合のような大量レコード処理の高速化を目指して設計されており，より大きな粒度の処理単位を採用している．ストリーム[9]はその好例である．実際，いくつかの実験システムではテーブル結合の性能においてレコード単位処理方式の商用システムを大きく上回る結果を報告している．しかしながら例えば単一レコード検索のような負荷の軽い問い合わせの処理ではむしろ不利である．

現実のデータベース環境では，通常は単一レコード検索が行われ，時に大量レコード処理要求が発生するといった形態が多くみられる．したがって1レコードから数十メガレコードに至る検索要求をともに効率良く処理する必要がある．いいかえると，処理単位をどう設定してこの相反する要求に応えるかがデータベースマシンの実用化に向けた大きな課題である．

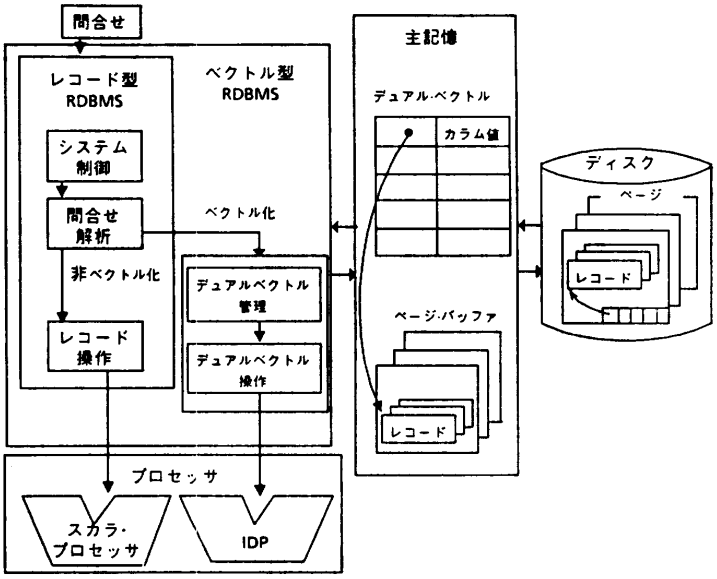


図 1 ベクトル型 RDBMS のシステム構成
Fig. 1 RDBMS based on dynamic vectorization strategy.

われわれはこの問題の解決に向けたアプローチとして，関係データベースの動的ベクトル化方式を考案した．次節ではこの方式について詳しく述べる．

2.2 関係データベースの動的ベクトル化方式

図1は動的ベクトル化方式に基づく関係データベースシステムの構成を示している．本方式の特徴は次のようにまとめられる．

（1） ディスク上のデータ格納形式は従来のレコード単位処理方式と同様とする．

（2） 主記憶上のデータは2通りの形式とする．一つはページバッファ，すなわちディスク上のデータページ群の写しである．もう一つはテーブルの縦一欄（以下カラムと呼ぶ）を後述のデュアルベクトルと呼ぶ形式で配列したものである．図1に示されるようにデュアルベクトルの各要素は二つの部分から成っており，この例では前半部にレコードへのポインタが，後半部にはカラムの値がそれぞれ格納されている．

（3） システムは検索要求を受け取ると，まず処理単位としてレコードとカラムのどちらが有利かを判定する．レコードが選ばれた場合，検索要求は従来のレコード単位処理方式で処理される．カラムが選ばれた時には，必要なカラムのデュアルベクトルが動的に生成されて IDP によりパイプライン処理される．

以下では具体的な検索要求の処理手順を例として動的ベクトル化方式について説明する．検索要求としては SQL[10] で記述された次のようなテーブル結合を考

える.

```
SELECT
  PARTNAME, CODEA,
  MAKER, PRICE, TEL
    FROM A, B
    WHERE
    A. CODEA=B. CODEB
```

この問い合わせは**図2**に示した手順で処理される.

(1) デュアルベクトル生成

まず,従来のデータ構造を動的にベクトル形式に変換する.具体的には以下の手順をとる.

(a) A, B の各テーブルについて,その中のレコード識別子 (RID) を後半部とするデュアルベクトルを作成する.RID はそのレコードが格納されているページの番号とページ内の何番目のレコードであるかを示すスロット番号とのペアである.デュアルベクトルの前半部はこの時点では空であり,後で代入される.

(b) 各レコードに対してその主記憶アドレスをデュアルベクトルの前半部に格納する.

(c) 得られた主記憶アドレスを用いてテーブル A, B 中の各レコードからその結合カラムの値を取り出し,別のデュアルベクトルの後半部に格納する.前半部には,通番を格納する.

(2) ベクトルソート

前記(1)(c)で作成したデュアルベクトルをその後半部についてソートする.

(3) ベクトル結合

ソートされたベクトルをマージしながら結合する.結果として後半部(カラム値)が等しいデュアルベクトル要素の前半部(通番)の対が得られる.

(4) 出　　力

最終的な結合結果を,(3)で得られた通番の対を使って作成し応答する.結合すべきレコードの主記憶アドレスはアドレスベクトルを通番をインデクスとして参照することにより知ることができる.

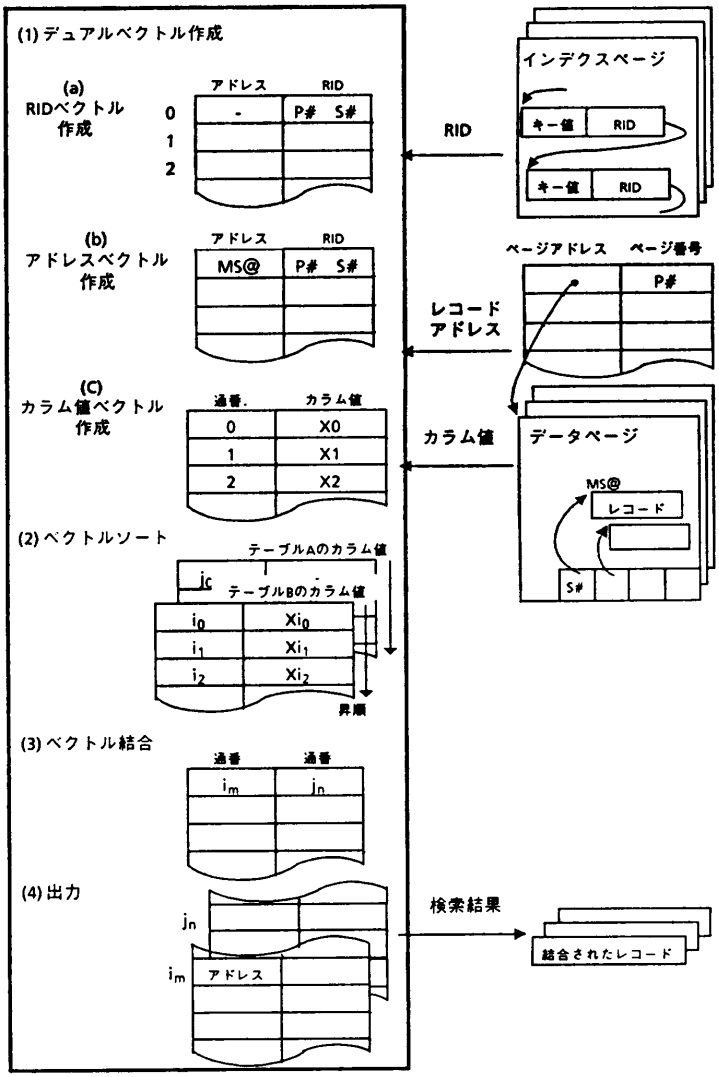


図 2 結合処理のベクトル化
Fig. 2 Vectorized join process.

動的ベクトル化方式の長所として以下があげられる.

(1) 問い合わせの性質に応じて,最適な処理単位を選択できる.

(2) 従来の商用関係データベースシステムからみると,ベクトル化されたデータベースアクセスは新たな高速アクセスパスとなる.したがって商用システムがもつ豊富な機能はそのままにして高速化がはかれる.

(3) 全データベースを最初からベクトル形式で主記憶上に常駐する静的ベクトル化方式と比べ,主記憶の利用効率が高い.

(4) 二次記憶中を含めテーブルをすべてカラム単位に格納する方式では,各カラムの値を結果のレコー

ドへと組み上げる処理の負荷が重い．動的ベクトル化方式ではレコードとカラム値ベクトルの両方をもっているのでこの処理が不要となる．

動的ベクトル化方式の問題点はいうまでもなくベクトル生成処理のオーバヘッドである．これについては第5章で定量的に評価する．

3. IDP のアーキテクチャ

3.1 従来のベクトル計算機の問題点

大規模数値計算専用と考えられてきたベクトル計算機は，記号処理の高速化にも高い潜在能力を有していることが認識されつつある．例えば Prolog などの論理型言語で記述された記号処理プログラムに対しても，ベクトル計算機による高速実行の可能性が示されている[11]．

しかしながらデータベース処理においては，従来のベクトル計算機がもつ機能（算術・論理演算，比較演算，マスク付き演算，圧縮・伸張演算，および間接アドレッシングなど）だけでは十分な高速化が達成できない．ソート処理はその好例であり，処理の複雑さが $O(n \log n)$ であるような実用的アルゴリズムに対しては良いベクトル化手法が知られていない[12]．

アーキテクチャ的にみると，従来のベクトル計算機の問題点は以下の点に集約される．

（1） 従来のベクトル演算では，演算されるベクトル要素対は演算の開始前に静的に指定される．例えばベクトル A と B の加算 (for i=0 to N−1 do $C(i) := A(i) + B(i)$) を例にとると，演算される要素はその要素番号が等しいもの同士に限られている．しかしながら記号処理において重要な演算の多くは，この制約下では実現できない．データベース処理で多用されるマージ演算もその好例である．マージ演算では要素 $A(i)$ と $B(j)$ の比較結果に応じて次に $A(i+1)$ と $B(j)$，あるいは $A(i)$ と $B(j+1)$ との比較が行われる．すなわち演算対象が演算の進行に応じて動的に変化する．

（2） 従来のベクトル計算機では，ベクトル要素のデータ型は，整数，浮動小数点数，論理データ，インデクス等のアトミックなものに限られている．このためタグ付きデータや Lisp の cons セルが備えているような，柔軟なデータ構造表現力に欠ける．

3.2 デュアルベクトルアーキテクチャ

本節では，IDP の演算形式と命令体系について述べる．IDP はデータベース処理向きに拡張されたベクトル計算機である．前節において指摘した従来の問題点に対しては以下の解決をはかった．

（1） IDP 命令では演算対象となるベクトル要素は動的に決定される．したがって演算要素を示すベクトル要素カウンタは，各ベクトルオペランド対応に設けられ独立に更新される．

（2） 基本データ型としてデュアルベクトル形式を新設した．第2章で述べたように，デュアルベクトルの各要素はフロント部とリア部の二つの部分から成っている．フロント部とリア部にはそれぞれ異なった型のデータを格納できる．前章の図2は，フロント部に要素番号，リア部にはキー値を格納してソートする等の，デュアルベクトルの典型的使用例のいくつかを示している．

デュアルベクトルは任意の2項関係を表現でき，関係データベース処理のベクトル化に有効なデータ構造である．デュアルベクトル構造は Lisp の cons セルをベクトル形式に配列したものともみることができ，より一般の記号処理のベクトル化においても有効と考えられる．例えば記号処理用 SIMD 型計算機であるコネクションマシン[13]も，Xector（ゼクタ）と呼ばれるデュアルベクトルと類似性の高い基本データ構造を採用している．

IDP の命令体系は従来の数値処理用ベクトル計算機のもつ命令群に，デュアルベクトル系の命令群を加えて構成されている．**表1**に新設したデュアルベクトル系命令の一覧を示す．機能的には，ソート，結合，逐次探索，および集合演算を行う命令群を用意している．各命令と関係演算との対応は以下のとおりである．

（1） 選択，制限，ソート，結合

これらの関係操作については直接に対応する命令を用意している．すなわち逐次探索命令が選択ならびに制限に，ソート命令がソートに，また結合命令が結合にそれぞれ対応している．

（2） 重複排除

逐次探索命令を用いて，射影演算などにおける中心的な処理である重複排除を行うことができる．例えば**図3**に示すように，ソート済みのカラム値ベクトル X の重複排除を行うには，X と X の開始アドレスを1要素分ずらしたベクトル Y とを対象に，異なるリア部をもつ要素対を逐次探索命令により探索すれば良い．この逐次探索命令では，リア部が異なるキー値をもつデュアルベクトル要素のフロント部の対のみがベ

表 1 IDP 拡張ベクトル命令
Table 1 IDP extended vector instructions.

機能概要	適用処理
ソート X(i).r : y(j).r ≦ : Z(k)←X(i); i←i+1; k←k+1 X(i).r : y(j).r > : Z(k)←y(j); j←j+1; k←k+1	集合和 ソート
結合 i←i+1 X(i).r : y(j).r < : Z(k).f←X(i).f; Z(k).r←y(j).f; i←i+1; j←j+1; k←k+1 X(i).r : y(j).r = : 同上 X(i).r : y(j).r > : j←j+1	集合積 結合
集合差 j←j+1 X(i).r : y(j).r > : j←j+1 X(i).r : y(j).r < : Z(k)←X(i); i←i+1; k←k+1 X(i).r : y(j).r = : i←i+1	集合差
逐次探索 X(i).r : y(j).r yes : Z(k).f←X(i).f; Z(k).r←y(j).f; i←i+1; j←j+1; k←k+1 X(i).r : y(j).r no : i←i+1; j←j+1	単純選択 統計演算 重複排除

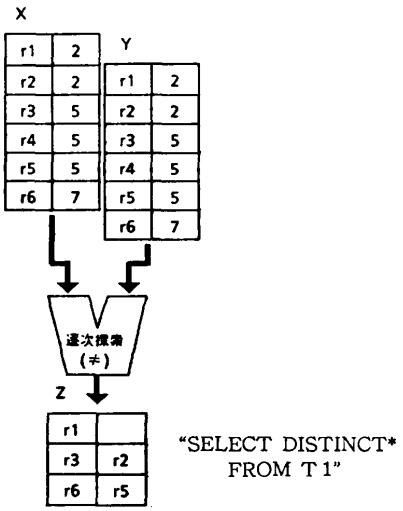


図 3 IDP 命令による重複排除
Fig. 3 Duplication removal.

クトル Z に出力される．したがって Z のフロント部には X の中で等しいキー値をもつ要素群 ($r1$, $r2$), ($r3$, $r4$, $r5$), ($r6$) の先頭要素 $r1$, $r3$, $r6$ が得られ，逆にリア部には最後の要素である $r2$, $r5$ が得られる．

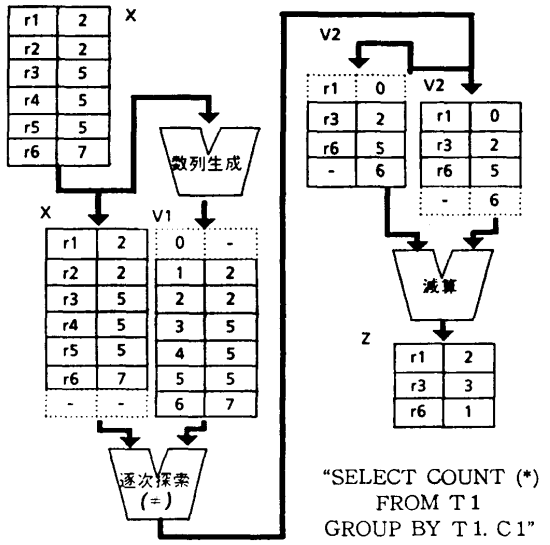


図 4 統計演算のベクトル化
Fig. 4 Aggregation.

(3) 統 計 演 算

SQL における GROUP BY 指定のような統計的な処理も，重複排除と同様の手法で図 4 に示すようにベクトル化できる．ここでは逐次探索命令に加えて二つの従来型のベクトル命令を使用している．一つは数列生成用の命令 (Vector Element Increment) である．この命令の機能は (for $i=0$ to $N-1$ do $X(i+1) := X(i)+A(i)$) であり，$X(1)$ を 0, $A(i)$ を 1 とすれば X に 0 から始まる整数列が作業用デュアルベクトル $V1$ のフロント部に得られる．もう一つはベクトル要素間の差を求める減算命令 (Vector Elementwise Subtract) である．減算命令により結果ベクトル Z のリア部に重複要素の個数が得られる．

第 2 章で説明したように，上述のデュアルベクトル操作の前処理として，カラム値ベクトルの生成が行われる．したがって問い合わせ処理全体を効率良くベクトル化するには，関係の動的ベクトル化処理そのもののベクトル化が必要となる．新設したデュアルベクトル命令はこの目的にもかなうものである．図 5 に IDP によるアドレスベクトル作成処理の概要を示す．この例はデータページ番号のリスト Vr を受け取ってその主記憶上アドレスの一覧 Vr' を得る，一種のアドレス変換処理である．主記憶にあるページバッファ上にロードされているデータページのアドレスはページ番号順にソートして，デュアルベクトル形式のページバッファディレクトリ Vd に格納しておく．変換は以下の手順で進められる．

(1) IDP の集合差命令により，Vr と Vd の集合差 Vu を作成する．Vu は変換を要求されたページのうち，まだページバッファにロードされていないページの番号のリストである．

(2) スカラ命令を使って，未ロードページをロードし，その主記憶アドレス Vu のフロント部に格納して Vu' を作成する．

(3) Vd と Vu' をソート命令によりマージしてページバッファディレクトリを Vd' に更新する．ここではソート命令は集合和を計算するのに使っている．

(4) Vr と Vd' の集合積を結合命令を用いて計算し，変換結果 Vr' を得る．

4. IDP のハードウェア

4.1 ハードウェア設計方針

データベースマシンはホストマシンとの結合形態から，スタンドアロン型，バックエンド型，および統合型の3種に大別される．スタンドアロン型は通信制御等を含めてすべての処理を独立して行うマシンである．スタンドアロン型のマシンにおいては，設計の自由度が大きいので，高速化のために最適な構成を追求することができる．反面，設計コストは高く，すべてのデータベース機能を実現するのは決して容易ではない．バックエンド型のマシンは，チャネルやネットワークを介してホストマシンと通信しながらデータベース処理を行う形態をとる．バックエンド型のデータベースマシンの主要な問題点はホストとの通信オーバヘッドにある．通信オーバヘッドを削減するには，ホストとのインタフェースの水準を上げてコマンドやデータのやり取りを減らす必要がある．しかしながらインタフェースを高水準化するほど，バックエンドマシンに必要な機能は多くなる．すなわちスタンドアロン型に近くなり，その性能価格比を損なう要因となる．

IDP はその命令からもわかるように，統合型のデータベースマシンである．統合型のマシンは，ホストマシンの CPU に直接付加してその性能を向上させるアクセラレータである．IDP を統合型のマシンとしたのは次の二つの理由からである．

(1) 通信オーバヘッドがほとんどない．

(2) ホストマシンとしてベクトルプロセッサを選ぶことにより少ない設計コストでの実装が可能である．

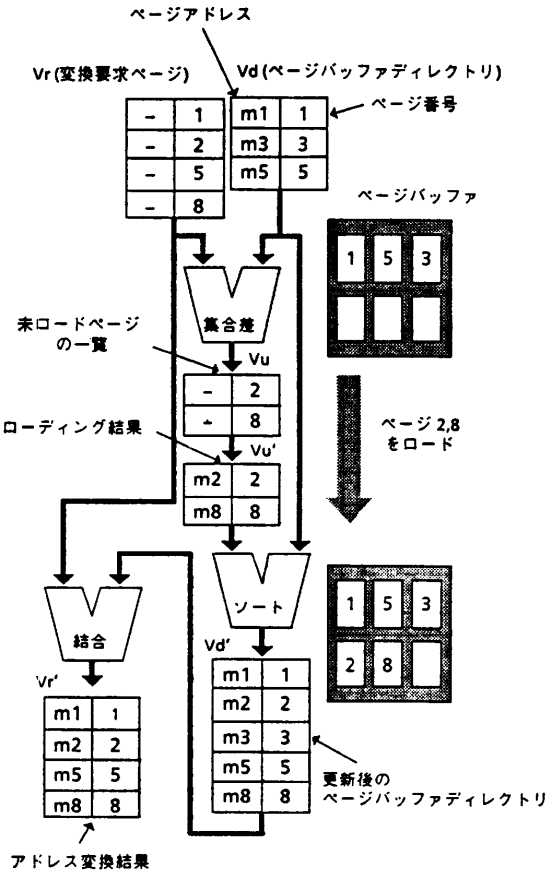
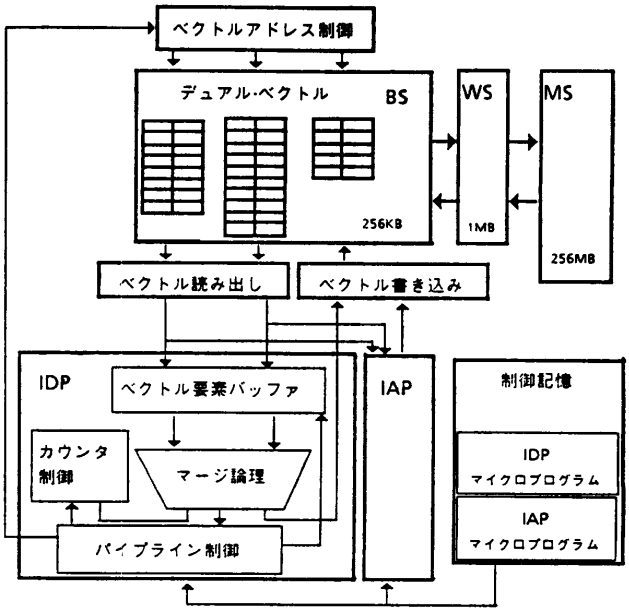


図 5 アドレスベクトル作成処理
Fig. 5 Vectorized address translation process.



BS：バッファ記憶，WS：ワーク記憶，MS：主記憶
図 6 M-680 H/IDP のハードウェア構成
Fig. 6 Construction of M-680 H/IDP.

4.2 ハードウェア構成

IDP は日立製作所の大型汎用計算機，M-680 H[14] をそのホストマシンとしている．M-680 H には，IAP (Integrated Array Processor)[15] と呼ばれる数値計算用のベクトルプロセッサが付加機構として用意されている．関係データベース処理のベクトル化にあたって，従来型のベクトル命令が必要な際には，この IAP を使用している．例えば前章図 4 中の数列生成や減算は IAP のもつベクトル命令である．IDP は IAP の存在を前提として設計されている．IDP と IAP はベクトル処理に必要な基本論理の多くを共用しているので，IDP の付加はコスト面での大きな上昇を伴わない．実際，IDP の論理規模は M-680 H 本体の約 5 % 程度である．

図 6 に IDP を付加した M-680 H のハードウェア構成を示す．M-680 H の記憶階層は，バッファ記憶 (BS)，ワーク記憶 (WS)，および主記憶 (MS) の 3 階層から成っている．IDP と IAP によるベクトル要素アクセスは高速なキャッシュである BS に対して行われる．IDP または IAP の動作は制御記憶 (CS) に格納されたファームウェアにより制御される．

物理的には IDP は約 70 LSI を搭載可能な基板 1 枚から成る．各 LSI は ECL (Emitter Coupled Logic) で，約 2 キロゲートの容量を有する．IDP とホストマシンは同一のハードウェア実装技術を用いている．

図 6 は IDP のパイプラインによるベクトル処理の流れを示している．単一のベクトル要素に注目するとその処理は以下の順で行われる．

(1) ベクトル要素のフェッチ要求を発行するとともにフェッチアドレスを更新する．

(2) IDP へのベクトル要素到着を待つ．

(3) ベクトル要素を比較し，結果に応じてベクトル要素カウンタを更新する．

(4) 結果のストア要求を発行するとともに，ストアアドレスを更新する．

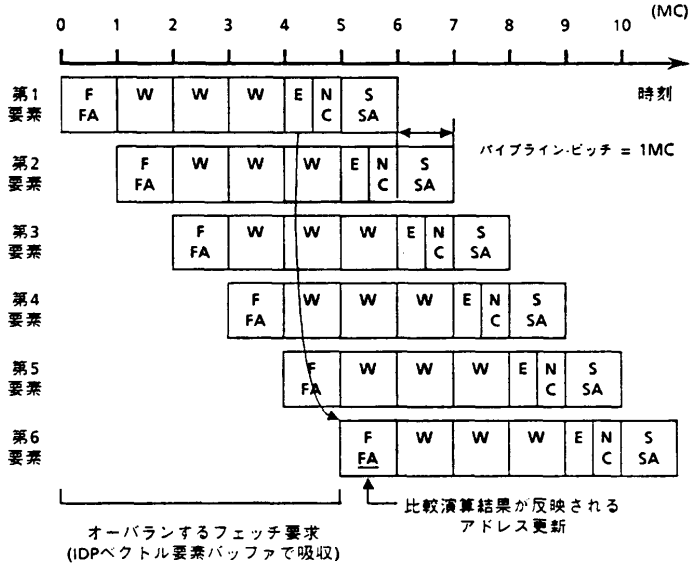
各要素の処理は 1 マシンサイクルごとに開始されるので，キャッシュのミスヒットなどのパイプライン擾乱がない限り，IDP は 1 サイクルピッチで演算結果を生成する．IDP 命令の機能を，IDP を使わずに通常のスカラ命令で実現した場合，その最内側ループは約 20 命令から成る．したがって，IDP を用いればピークで 20 倍以上の加速率が得られる．

IDP のベクトル要素に対するアクセス制御は従来のベクトル命令に比べてより複雑である．その理由は既に述べたように，IDP 命令では，直前のベクトル要素に対する比較結果に応じて次にアクセスされるベクトル要素が決まるからである．この性質から次のベクトル要素のアクセスは前要素の演算結果が定まってから行うのが自然である．しかしながら効率の良いパイプライン化を実現するには，あるベクトル要素に対する演算結果が確定する以前に，次のベクトル要素のアクセスを開始する必要がある．**図 7** が示すように 1 サイクルピッチのパイプライン処理では，本来第 2 要素のアクセスに反映されるべき第 1 要素の演算結果を第 6 要素にしか反映できない．IDP ではこの制御遅れにより誤って IDP に到着するベクトル要素を専用のベクトル要素バッファに退避しておくことでこの問題を解決している．

5. 性能評価

5.1 命令レベル評価

本節では IDP の命令レベルの性能実測結果について述べる．その目的は前章で述べた IDP のパイプラ



F：フェッチ要求，FA：フェッチアドレス更新，W：ベクトル要素到着待ち，N：IDP ベクトル要素バッファからのデータ取り出し，E：要素の比較演算，C：ベクトルカウンタ更新，S：ストア要求，SA：ストアアドレス更新

図 7 IDP のパイプライン制御
Fig. 7 IDP pipelining method.

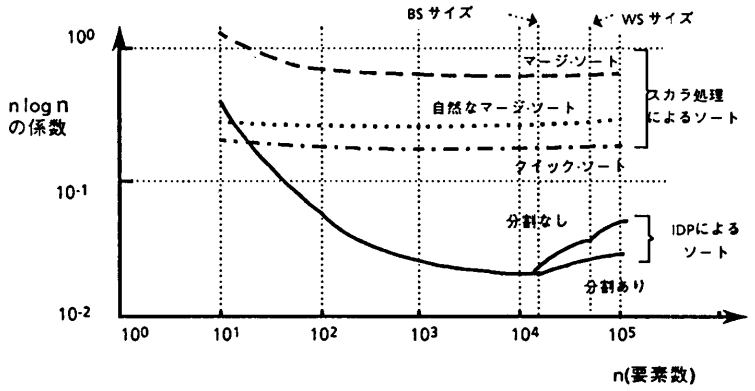


図 8 IDP の命令レベル性能（ソート処理）
Fig. 8 IDP instruction level performance (Sort).

イン化方式の効果の確認にある．命令としてはソート命令を例にとり，種々のアルゴリズムを採用した通常のスカラプログラムとの性能比較を行った．

図8はその結果を示すグラフである．スカラ命令によるソートのアルゴリズムとしては，単純マージソート，自然マージソート，およびクイックソート[16]を用いた．図8のグラフの横軸にはソート対象データの個数 n をとり，縦軸には各ソートアルゴリズムのオーダである $n \log n$ の係数をとっている．ソート対象データは，乱数的に発生させた4バイトのキー値を含む8バイトデータの列である．図7から明らかなように，データ数が十分に多い場合，IDP によるソートはソフトウェアによるソートプログラムに比べて約 10 倍高速である．IDP の性能曲線はデータ数 10^4 を超えたあたりで二つに分かれている．上の線はデータ量がホストマシンのバッファ記憶あるいはワーク記憶に入りきらなくなって，ヒット率が低下しパイプラインが乱れることによる性能低下を示している．下の線は入力データをバッファ記憶やワーク記憶に入りきるように分割して処理した場合の性能を示している．この結果から大規模なテーブルをベクトル処理する際には，キャッシュに入る程度の大きさにテーブルを分割して処理するのが有効であるといえる．

5.2 コマンドレベル性能評価

SQL のコマンドレベルでの IDP の性能を評価するにあたって，性能測定用のデータベースとして二つのテーブル T1, T2 を用意した．各テーブルは 6,000 レコードを格納し，各レコードの長さは 200 バイトである．次の 10 個の検索コマンドから成る連続するデータベースアクセスをベンチマークとして使用した．

（1） 結合処理2件（インデクスのない環境での T1 と T2 の結合．結合結果は 100 レコードである．）

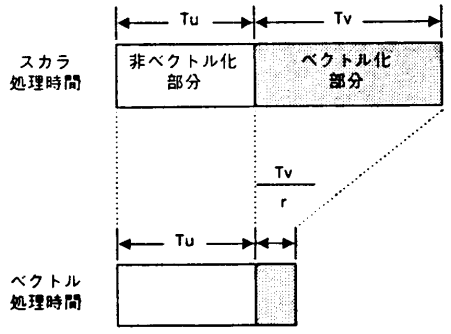
（2） 全件検索1件（インデクスのない環境で，T1 の全レコードを走査して 100 レコードを出力する．）

（3） インデクス検索7件（インデクスを用いて T1 から 100 件を選択する．）

一般にベクトル処理による高速化においては，図9に示すような関係が成立する．図9の上段の帯グラフは，汎用計算機のスカラ命令による実行時間を，また下段はベクトル計算機のベクトル命令による実行時間をそれぞれ表している．スカラ命令による実行時間はさらにベクトル化不可能な部分 Tu と，ベクトル化可能な部分 Tv とに分けられる．ベクトル化可能な部分 Tv はベクトル計算機のパイプライン処理ハードウェアにより高速化される．パイプライン処理ハードウェアの加速率を r とすれば，ベクトル計算機による実行時間は Tu と Tv/r の和となる．全体としての性能向上率は次式で表される．

性能向上率 $=(Tu+Tv)/(Tu+Tv/r)$

したがってベクトル化により高い性能向上率を得るためには，加速率を高めるとともにベクトル化可能な部分を多くすることが必要である．このベクトル化可能な部分の比率 $Tv/(Tu+Tv)$ は重要な因子であり，ベクトル化率と呼ばれる．本性能評価では Tv の測定は IDP および IAP のソフトウェアシミュレータを作成して行った．すなわちシミュレータを使ったベクトル型関係データベース管理システム（RDBMS）に



$$\text{性能向上率} = \frac{Tu+Tv}{Tu+Tv/r}$$

（r=加速率）

図 9 ベクトル処理による高速化効果
Fig. 9 Performance improvement by vector processing.

おける SQL コマンド処理時間 ($Tu+Tv$) の中で，シミュレータのみの走行時間を測定しこれを Tv としてベクトル化率を算出している．

前記ベンチマークによる性能実測結果を**図 10** の四つのグラフにより示す．最初のグラフは，10 コマンド全体の処理に要する CPU 時間を，また 2 番目から 4 番目のグラフは各検索コマンドごとの CPU 処理時間を示している．各グラフにおいては，レコード単位処理方式の従来型の商用 RDBMS での処理時間，動的ベクトル化方式を採用したベクトル型 RDBMS を IDP のシミュレータを使って動作させた場合の処理時間，およびベクトル型 RDBMS を IDP ハードウェアを使って動作させた場合の三つが比較されている．IDP のソフトウェアシミュレータを用いての測定も行ったのは，前述のベクトル化率の測定と，動的ベクトル化方式における処理粒度拡大効果とを把握するためである．

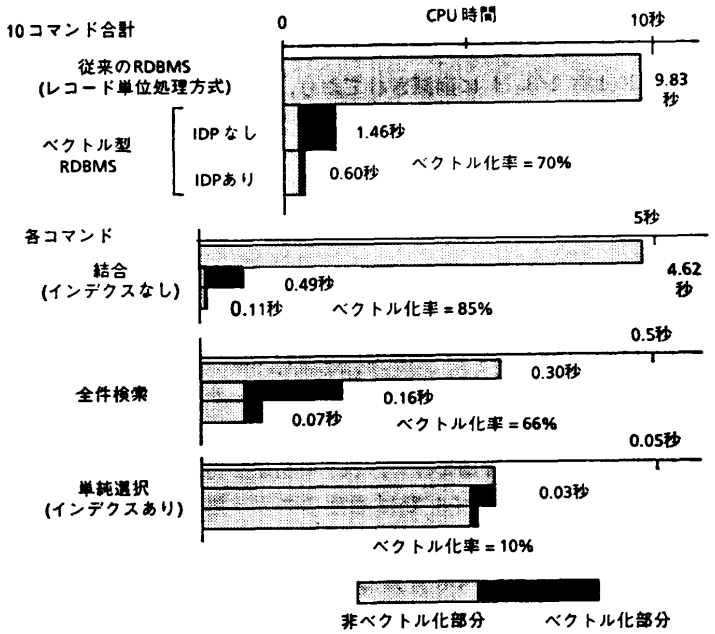


図 10 IDP のコマンドレベル性能評価
Fig. 10 RDB command level evaluation.

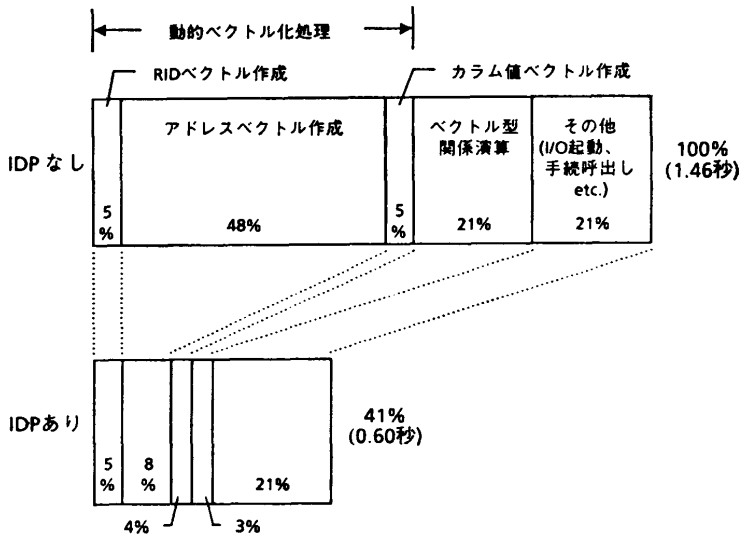


図 11 ベクトル型関係データベースの処理内訳
Fig. 11 Breakdown of vectorized query processing.

なお主記憶上のデータベースバッファ（ページバッファ）のサイズは従来型とベクトル型で等しく，各テーブルおよびインデクスが入りきる容量とした．したがって本ベンチマークにおいては最初の検索時に必要なテーブルおよびインデクスがすべて二次記憶からデータベースバッファ上にロードされ，以後常駐化（キャッシング）されることになる．すなわちキャッシング効果については従来型とベクトル型の間で差が生じない環境で測定した．

図 10 が示すように，このベンチマークでは処理粒度拡大のソフトウェア的効果は大きく，IDP ハードウェアの効果と合わせベクトル型 RDBMS は従来の RDBMS に比べ CPU 時間で約 15 倍の高速化を達成している．特に結合処理はベクトル化率も 85% と高く，性能改善の度合が著しい．反面，ベクトル型 RDBMS はインデクス検索に対しては効果が少ない．この原因は，インデクス検索では対象レコードがインデクスによりあらかじめ絞り込まれるので粒度拡大効果がほとんどないためと，ベクトル化率が 10% 程度と低いのが原因である．ベクトル化率が低い理由はインデクスが B-tree 形式で，その探索のベクトル化が困難なためである．ベクトル化率を向上させるにはよりベクトル化に適したインデクス構造の採用を検討する必要がある．

図 11 にベクトル型 RDBMS の CPU 処理の内訳を示す．IDP のソフトウェアシミュレータを使った場合では，全処理の 58% を前処理であるデュアルベ

クトル作成処理に要しており，そのオーバヘッドは大きい．IDP を使った場合，デュアルベクトル作成オーバヘッドは約 1/3．4 に削減されており，第3章で述べたデュアルベクトル作成処理自体のベクトル化は有効な手法といえる．

6. おわりに

関係データベースの動的ベクトル化方式に基づく，大容量主記憶向けのデータベースプロセッサ IDP (Integrated Database Processor) について述べた．IDP は従来の数値計算用のベクトルプロセッサを記号処理向きに拡張したアーキテクチャを有する．IDP の試作ならびに性能評価を行い，以下の結果を得た．

（1） IDP は命令レベルでは通常のスカラ処理に比べ約10倍高速である．

（2） テーブル結合検索を含む負荷の重いベンチマークでは，IDP を用いたベクトル型関係データベース管理システムは従来のレコード単位処理方式のシステムに比べ，約15倍の性能改善を達成した．ベクトル化率としては約70%を実測した．

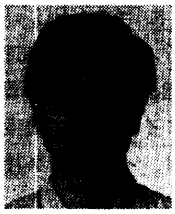
これらの結果から関係データベースの動的ベクトル化方式と IDP アーキテクチャの有効性を確認した．

謝辞 本研究を御支援頂いた日立製作所中央研究所の堀越彌所長に，また IDP の試作に御協力頂いた日立製作所神奈川工場の河辺峻主任技師と日立製作所ソフトウェア工場の高橋政美主任技師をはじめとする多くの方々に深く感謝致します．

参考文献

1) Hsiao, D. K.: Data Base Computers, *Advances in Computers*, Vol. 19 (1983).
2) Boral, H. and Dewitt, D. J.: Database Machines: An Idea Whose Time Has Passed ?, in *Database Machines* (Leilich, H. and Missikoff, M. eds.), Springer-Verlag (1983).
3) Kitsuregawa, M., Tanaka, H. and Moto-Oka, T.: Architecture and Performance of the Relational Algebra Machine GRACE, *Proceedings of International Conference on Parallel Processing*, August, pp. 241-250 (1984).
4) Neches, P. M.: Hardware Support for Advanced Data Management Systems, *IEEE Comput.*, Vol. 17, No. 11, pp. 29-40 (1984).
5) Dewitt, D. J., Gerber, R. H., Graefe, G., Heytens, M. L., Kumar, K. B. and Muralikrishna, M.: GAMMA—A High Performance Dataflow Database Machine, *Proceedings of Twelfth International Conference on Very Large Data Bases*, August, pp. 228-237 (1986).
6) Dewitt, D. J., Katz, R., Olken, F., Shapiro, D., Stonebraker, M. and Wood, D.: Implementation Techniques for Main Memory Database Systems, *Proceedings of the 1984 SIGMOD Conference*, June, pp. 1-8 (1984).
7) Torii, S., Kojima, K., Yoshizumi, S., Sakata, A., Takamoto, Y., Kawabe, S., Takahashi, M. and Ishizuka, T.: A Relational Database System Architecture Based on a Vector Processing Method, *Proceedings of the Third International Conference on Data Engineering*, February, pp. 182-189 (1987).
8) Date, C. J.: *An Introduction to Database Systems*, 3rd ed., pp. 171-180, Addison-Wesley (1981).
9) Tanaka, Y.: A Data Stream Database Machine with Large Capacity, in *Advanced Database Machine Architectures* (Hsiao, D. K. ed.), pp. 168-202, Prentice-Hall (1983).
10) Date, C. J.: *An Introduction to Database Systems*, 4th ed., pp. 127-153, Addison-Wesley (1986).
11) 金田 泰，小島啓二，菅谷正弘：ベクトル計算機のための探索問題の計算法「並列バックトラック計算法」，情報処理学会論文誌，Vol. 29, No. 10, pp. 985-994 (1987).
12) 石浦菜岐佐，高木直史，矢島脩三：ベクトル計算機上でのソーティング，情報処理学会論文誌，Vol. 29, No. 4, pp. 378-385 (1988).
13) Hills, W. D.: *The Connection Machine*, The MIT Press (1985).
14) Wada, K., Nagashima, S. and Odaka, T.: Design for a High Performance Large-Scale General Purpose Computer, The HITACHI M-680 H Processor, *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers*, October, pp. 481-484 (1985).
15) 堀越 彌，梅谷征雄：汎用計算機のための内蔵ベクトル演算方式，情報処理学会論文誌，Vol. 24, No. 2, pp. 191-199 (1983).
16) Knuth, D. E.: Searching and Sorting, *The Art of Computer Programming*, Vol. 3, Addison-Wesley (1973).

（平成元年5月22日受付）
（平成元年9月12日採録）



小島 啓二（正会員）

昭和31年生．昭和55年京都大学理学部卒業．昭和57年同大学院修士課程修了．同年(株)日立製作所中央研究所入所．以来データベースマシン，ユーザインタフェースの研究に従事．ソフトウェア科学会会員．

鳥居 俊一（正会員）

昭和24年生．昭和46年東京大学工学部計数工学科卒業．昭和48年同大学院修士課程修了．同年(株)日立製作所中央研究所入社．以来，大型計算機，スーパコンピュータおよびデータベース高速化の研究に従事．現在同所第8部主任研究員．

吉住 誠一（正会員）

昭和43年東京大学理学部数学科卒業．昭和46年同大学院修士課程修了．同年(株)日立製作所中央研究所入所．以来，OS，性能評価，コンピュータアーキテクチャの研究に従事．現在同所企画室主任研究員．ソフトウェア科学会，IEEE，ACM 各会員．