データベースマシン実験システム

An Experimental Database Machine System

牧野武則，箱崎勝也，水摩正行，梅村護，日吉茂樹，渡辺正信
T. Makino, K. Hakozaki, M. Mizuma, M. Umemura, S. Hiyoshi, M. Watanabe
日本電気（株）　中央研究所
Central Reserch Laboratries, Nippon Electric Co. Ltd.

1. はじめに

データベースシステムに対する需要の増大とともに，大量データを統合的に管理するデータベース管理システムの高速化が望まれている。いわゆる汎用計算機にとって，非数値データの処理効率は必ずしもよいとはいえず，大量のハードウェアを投入して得られた演算能力は，データベース処理の性能向上には，それ程の効果はない。このような背景をもとに，汎用計算機からデータベース処理機能を切り離し，専用サブシステムにより効率よく実行することで，システム全体の性能向上を計る方式が研究されている。

データベースマシンには，すでに多くの解説論文（例えば[1]）により紹介されているように，いくつかのアプローチがある。これらのアーキテクチャは，変形はあるにしても，検索アルゴリズムにより，RAP[2]のような並列フルスキャン，通常のデータベース処理で採用されているインデックスやリンクによる検索，およびそれらの中間形としてみることができるDBC[3]のような部分並列スキャンに分けることができる。それぞれ特徴を有しており，データベースマシンの適用領域によりその効果が検討される。RAPのような擬似連想メモリによるデータ検索は魅力的ではあるが，近い将来，高速で安価な連想メモリが得られる保障がないこと，定型処理については，インデックスやリンクによるデータ検索が依然として強力であることから，商用を意識したデータベースマシンでは，インデックスやリンクによるデータ検索をサポートするアーキテクチャが有用であると考えられる。

実験システムは，本データベースマシンの概念である汎用データベースサブシステム(GDS)[4]のミニコンピュータによる実現を主体とし，汎用システムをホスト計算機として結合した，主記憶共有型の機能分散多重プロセッサ構成である。当初，文献[4]で紹介したように，基本モジュールをハードウェアで実現することを考えていたが，実験システムでは，GDS全体はソフトウェアとファームウェアにより実現している。しかしながら，この実験システムで得られた性能データにより，ソフト／ファーム／ハードトレードオフに関する情報が得られるよう設計されている。

ホスト計算機との結合は，性能を支配する大きな問題である。この実験では，ホストとGDS間のデータ／コマンド転送時間を最少にするため，主記憶を共有するようにしている。一方GDSの起動，終了の通知は，ホスト計算機のOSの要求をさけるため，IOインタフェイスを使用し，ホストからは，GDSがIO装置にみえる方式をとっている。

ここでは，まずDBMで実現しているデータベースサブシステムGDSの簡単な説明をし，次に実験システムの構成について述べる。そして実験システムで得られた性能評価データをもとに，DBMの有効性と，さらに高性能化を追求した場合のDBMの性能を，DBMアーキテクチャとともに議論する。

2. 汎用データベースサブシステム（GDS）

DBMがいろいろなタイプのデータベースシステム（CODASYLやRDB）を共通に支援できるように，汎用性の高いデータベースサブシステムインタフェイス（GDI）が設定された。GDIの論理機能は，GDSによってサポートされる。Fig.1に，DBM（ここでは，GDPと呼んでいる）を有するデータ管理システムの構成を示す。

[GDI]

GDIは，汎用の高レベル機能インタフェイスである。基本的には，CODASYL型のDML（データ操作言語）に対応する単一レコードインタフェイスであり，RDBを意識した機能が強化されている。付録Aに，GDIコマンドのうち，データ操作に関するコマンドの一覧を示す。

GDIコマンドの機能を説明するために，Fig.2に，簡単な問い合せ（query）に対するGDIコマンドの展開例を示す。展開されたシーケンスの流れは，まず，GDSからトランザクションIDを受け取り，次に，エリアの使用権を得る。そして，Fig.2aで示す問い合せの処理に入る。この問い合せ言語はINGRESのQUEL[5]に準じている。仮りにSALに2次キイインデックス（image）が設定されているとすると，まず，アクセスパスを確立し（003），imageを介し直接検索を行ない，同時に第2の条件であるDEPTに対する比較も行う。この第2の条件も満足すれば，ターゲットリストで指定されるフィールド値を，MMに転送する。次のNEXTコマンドは，前のコマンドでポイントされたアクセスパス上の位置から，第2の条件を満足する次のレコードを検索し，存在すればフィールド値を，存在しなければ"Null"を転送する．一連の処理が終了すれば，使用したアクセスパス，エリアをリリースし，トランザクションを終了させる。

```
E   range   EMP
retrieve    E.NAME , E.AGE
where       E.SAL > 100000
            E.DEPT = TOY
```

Fig. 2a Example of Query

```
001     START-TRANS  return( trans-id )
002     USE-AREA( area-id, usage-mode,
          lock-mode, trans-id )
003     INITIATE-PATH( area-id, path-id,
          record-id )  return( cur-id )
004     GET-IMAGE-DIRECT( cur-id, key-value,
          key-condition )  return( rad )
005     GET-IMAGE-NEXT  return( rad )
006     RELEASE-PATH( cur-id )
007     RELEASE-AREA( area-id, trans-id )
008     END-TRANS( trans-id )
```

Fig. 2b GDI Command Sequence

このように，GDIコマンドは，問い合せの処理を効果的に実行することができ，RDBタイプのデータベースシステムをサポートすることができる。一方，CODASYLタイプのデータベースシステムの支援は，文献[7]で詳細に述べられる。

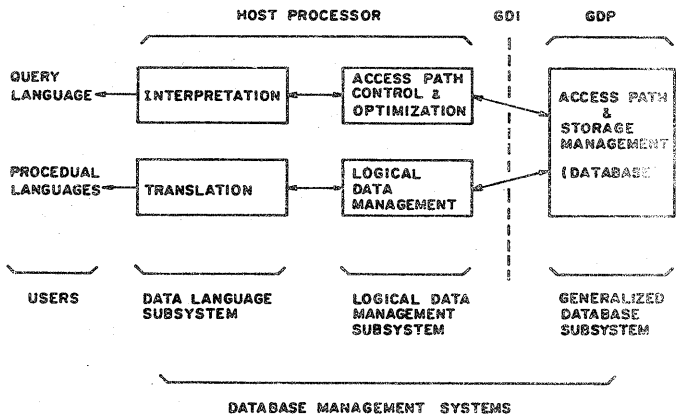

Fig. 1 Conceptual Database Management System Structure

[GDS]

GDSはデータベース管理システムの実行管理部に相当し，言語プロセッサやサービスユーティリティは含んでいない。

GDSの主な機能は，GDIコマンドの実行，動的な物理カタログの管理，トランザクションの実行管理，レコードのロック等の同時制御，データやトランザクションのリカバリを実行するとともに，データベース記憶空間の管理，制御を行なう。

以下，GDSの論理仕様を簡単にまとめる。設計概念は文献[4]に述べている。

論理空間

論理空間の表現は（area#, page#, line#）であり，それぞれ，2バイト，3バイト，1バイトで表わされる。areaは複数のrecord class（relation）と，2次アクセスパス（imageとlink），およびそれらのカタログ（物理スキーマ）とリカバリのためのログ情報，ページテーブルを含む。

pageは，固定長レコード用と可変長レコード用の2種があり，サイズは4kバイト*である。また，1つのページは複数のrecord classで共有されない。これらの制限により，処理効率の向上と，ページ管理を容易にしている。ページヘッダには，論理的な前ページと次ページのページ名が格納されており，あるrecord classのスキャンを可能にしている。

格納構造

データベースのオブジェクトは，次の3つの格納構造のうち1つに従って格納される。

- キイシーケンシャル構造
- キイランダマイズド構造
- エントリシーケンシャル構造

record classは，その処理される特徴によって，いずれかの格納構造を選ぶことができる。エントリシーケンシャル構造では，データの生成順の他，レコードアドレスを指定した近傍格納によって，データのクラスタリングを行うこともできる。area内でのページのアロケーションは，動的に行われるが，キイランダマイズド構造の場合は，初期ロード時に連続したページ名をもつページをリザーブしておかなければならない。

アクセスパス

SYSTEM-R [6] と同じように，imageとlinkと呼ぶ，2種のアクセスパスをもつ。imageは，2次インデックスを意味し，B*-tree [8] で維持される。一方，linkは，CODASYLのsetと同じであり，1つの親レコードと複数の子レコードを結ぶものである。

imageは値による検索に使用され，linkはCODASYLのsetに沿う検索，および，RDBでの候補キイによるjoin演算のサポートに使用される。

トランザクションの管理

GDSへの処理要求は，トランザクション単位に管理される。まず，START-TRANSACTIONが出されるとGDSは，その時点でユニークなtrans-idを返す。以降，このtrans-idでロックの管理，IO待ち，トランザクションバックアウト情報の収集・管理が行われる。END-TRANSACTIONを受けとると，GDSは，そのトランザクションに関するロックの解除，制御ブロックの解放を行なう。

同時処理制御

areaの使用権は，START-TRANSACTIONのすぐあとに出されるUSE-AREAコマンドの実行時に調べられる。使用モードは，検索／更新，ロックモードは共有／プロテクト／排他であり，トランザクションはFIFOの順にuse area queueにつながれ，使用権が確立するとready queueに送られ，実行に入る。

一方，実行中のレコードロックは排他ロックのみであり，必要な時点でレコードをロックすることができる。ロックされているオブジェクトをロックしようとすれば，lock wait queueにつながれるが，この時，このwaitでdead lockになるか否かが調べられる。

このロック機構の詳細は，文献[9]に報告している。

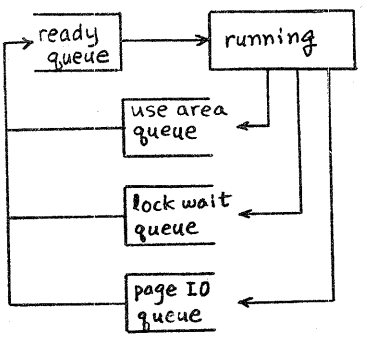

Fig. 3 Transaction state

*すべてのページを4kバイトにしているが，最大ページ内エントリ数が256のため，エントリ長が16以下の場合，空スペースが多くなる。従って，ページサイズを固定にするならば，1or2kバイトにすべきであった。

3. 実験システム

実験システムは，DBMの有効性を確認・検証することを主な目的としてあり，この具体的な目的は次の項目を含む。

- GDS基本仕様の実現性の確認
- 専用プロセッサの性能予測データの収集
- 専用システムにおける技術的課題の明確化

以上の目的に沿って実験システムの開発を行った。この実験システムは，かならずしも最良の構成をとっているわけではなく，少なくとも上の目的を達成するため，次の方針を取っている。

- 手持ちのホスト，ミニコンピュータを使用し，必要な改造を加える。
- GDSを機能別に切り分け，機能モジュールのハード/ファーム/ソフトのトレードオフが議論できるようにする。
- 基本モジュールはファームウェア化し，ファームウェアの効果を把握する。
- DBIOについては別途評価する。
- プロセッサ間通信は，ホストOSの改造をさけるため，PSI（IOインタフェイス）を使用する。
- GDIを介して，ホストでADBSやRDBを実現し，実験できるようにする。
- 実験システムで収集できないデータ（IO関係）は，実システムの測定や解析モデルにより補足する。

Fig. 4 は実験システムの構成を示す。ホスト計算機は，ACOS-400（汎用中型システム）を使用し，DBMは，バリアン社のミニコンピュータV-76を中核に，ホストとの結合機構を付加して実現している。

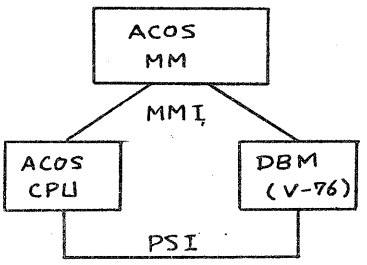

Fig. 4 Experimental system

3.1 データベースマシン（DBM）

DBMの中核であるミニコンピュータV-76の性能仕様は次のとおりである。

- メインメモリ：サイクル 660nsec
  容量 2バンク（32Kワード/バンク）
- WCS：サイクル 195nsec
  容量 2Kワード（64ビット/ワード）
- 命令実行時間
  1ワード命令：1320nsec
  2ワード命令：1980nsec
  マイクロプログラム：195nsec

IOデバイスは，DB格納とプログラム格納のため，5Mバイトのディスクを持つ。

WCSはマイクロプログラムで実現されるGDS基本モジュール等，ファームウェアが格納される。実際に使用している容量は，基本モジュールとDBMシステム制御を合せ，2Kステップである。

メインメモリは，DBバッファと，ソフトウェアで実現されたGDS機能モジュールが格納される。片方のバンクには，ソフトウェアと制御テーブル類が，他方のバンクはバッファとして使用される。

3.2 結合機構

ホストであるACOSシステムとDBMは，MMI（メインメモリインタフェイス）を介して，ACOSのメインメモリを共有する。さらに，DBMは，PSIを介して，ACOSシステムのチャンネルに結合されている。

MMIは主として，データ/コマンド/パラメータの転送に使用され，それらの転送オーバヘッドを最少にしている。

一方，PSIは，DBMへのトランザクションの起動と，ACOSへの終了通知を行うために用いられる。

Fig. 5 は，PSIを介したDBMの起動，GDIコマンドの実行，終了通知を，プロセス間の通信方式を示している。

ユーザプログラムは，実行すべきGDIコマンド（列）をUCA（通信エリア）に置き，そのアドレスをパラメータとして，Call GDIを出す。call GDIは，IO命令として解釈され，PIOルーチンは，セマフォにP-opを出すとともに，PSIを介して，UCAアド

レスをDBMに送る。DBMは送られたアドレスにより，GDIコマンドを取り込み実行に入る。検索キイ値や検索結果はUWA（ワークエリア）と直接転送する。一連の処理が終了するとDBMは，ACOSにPSIを介して終了通知を送り，この通知を受け取ったPIOルーチンはセマフォに V-op を出し，ユーザプログラムに処理を渡す。

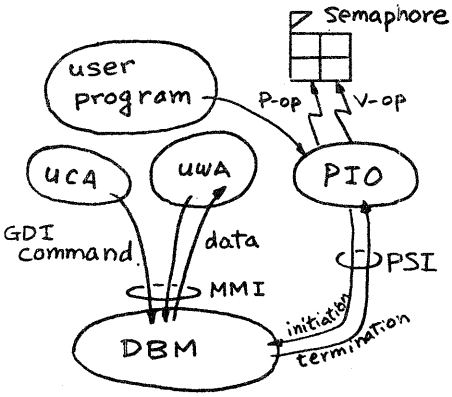

Fig.5 Control & data flow

3.3 機能モジュール

GDSの機能は，設計の容易さと，機能別の性能測定・評価を可能にするため，機能モジュールに分けられる。Fig.6 に機能モジュールの一覧を示す。

機能モジュール間には，プリミティブとそれに付随するパラメータが定義される。定義されたパラメータにより，機能モジュールに対するトランザクション制御ブロック内のパラメータボックスを決める。

GDSの実行は，function call命令により，機能モジュールを渡り歩くことで行われる。function call 命令は，ソフトウェアモジュール間では，return address stack & jump 命令に，ファームウェアモジュールを呼ぶ場合は，WCS call 命令，ハードモジュールを呼ぶ場合はIO命令にコンパイル時点に置き換わる。Fig.7に，機能モジュール間の連絡を示す。GPR-5 には trans-id が置かれ，機能モジュールは常に，GPR-5を参照し，TCB内のパラメータボックスをアクセスする。

```
GDS Function Modules
  Transaction Management
    Transaction Control Module      (TCM)
    Lock Control Module             (LCM)
    Backout Control Module          (BCM)
    Open/Close Control Module       (OCM)
  Logical Data Access
    GDI Processor                   (GDIP)
    Access-path Generator           (APG)
    Data Access Module              (DAM)
    Descriptor Access               (DDA)
  Physical Data Access
    Address Translation Module      (ATM)
    Hash Module                     (HSM)
    Data Manipulation Module        (DXM)
  Page Management
    Page Split/Garbage collection   (PSM)
    Page Allocation/Deallocation    (PAM)
    Page Replacement Module         (PRM)
```

Fig.6 FUNCTION MODULE

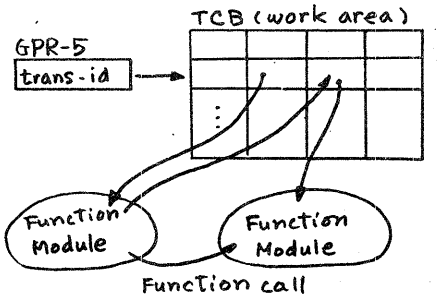

Fig.7 Function module call

次に主な機能モジュールの処理機能と実現形態を述べる。

TCMはトランザクションのディスパッチ制御とホストとの交信（PSI）とを主な機能としてもつ。LCMは，ATMでロック待ちが検出されたとき起動され，ロックマトリックスを維持する。OCMはエリアディスクリプタの準備/解放を行う。

GDIPはGDIコマンドをUCAから取り込み解釈を行う。APGはデータアクセスに必要なアクセスパスの設定を行う。DAMはデータ検索，選択，更新やアクセスパスの更新をデータ格納構造にもとづいて実行する。DDAはデータアクセスに必要なカタログ情報を用意する。

ATMは論理アドレスをバッファ内物理アド

レスに変換するとともに，ページロック／アンロックを行なり，ページフォルトならPRMをロックされているページが参照されるとLCMを呼ぶ。DXMは主としてDAMから呼ばれ，レコード内フィールド値のread/write/compare，B^+-tree pageのバイナリサーチ，レコードのページへのアロケーション／デアロケーションを実行する。

PRMはATMが採用するアドレス変換表の作成と，ページ再配置アルゴリズムを実行する。現在の仕様では，追い出しはLRUによる。PAMはページテーブルの管理を，PSMはB^+ tree pageのスプリット処理を行う。

4. 実験システム性能評価

GDSの機能モジュール，GDIコマンドの実行時間は，ハードウェアモニタにより詳細に測定している。システム性能については，これら測定結果により検討を行なう。

4.1 機能モジュール性能

実験で使用したデータモデルをFig. 8に示す。linkは，A.a_1とB.b_1を結合するものが定義されており，imageはA.a_2に対して設定されている。

測定では，単一トランザクションのもとで行い，IO時間は測定に含まない。

Table Iに，GDIコマンド別の各モジュールの走行時間を示す。GDIP以外のモジュールの走行時間には，ファームウェア化されているATMとDXMの走行時間も含んでいる。

ファームウェア化された機能モジュール単体の性能は次のとおり。

- ATM：アドレス変換，ロックの検査に要する時間は約30μsecである。
- DXM：ストリングデータの転送・比較は1バイト当り1μsec，B^+-treeページ内のバイナリサーチは128エントリの時約80μsec，ページへのレコードのアロケート／デアロケートは固定長の場合47μsec，可変長の場合72μsecである。

これらの走行時間は，ソフトで実現した場合に比べて，4倍から処理によっては10倍近く速くなっている。

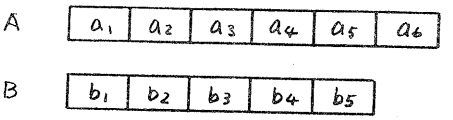

Fig. 8 Data structure

Table I Function module elapsed time

1) initiate-path : 788.4 μsec

GDIP	APG	DDA
42.4	361.6	384.4

2) get-image-direct : 3818 μsec

GDIP	DAM	
	image	f-move
97.0	390.0	3419.0

3) get-image-next : 4825 μsec

GDIP	DAM		
	image	f-move	compare
97.0	775.0	3419.0	534.0

4) get-link-next : 3873 μsec

GDIP	DAM		
	link	f-move	compare
88.0	366.0	3419.0	

5) store : 6184 μsec

GDIP	DDA/DAM			
	allocate	link-mod	image-mod	f-modify
211	435	1975	$108_{+(n)}$	3455

6) update : 1145 μsec

GDIP	DAM	
	check-key	f-modify
154	496	497

7) add-image-entry (internal) : 2040 μ

GDIP	DAM/APG			
	fetch-key	add-image	initiate-p	release-p
207	325	363	1095	50

Table I で示したGDIコマンド実行時間を評価するには，汎用システムで稼動するデータベースシステムと比較することが望ましい。しかし，コマンドレベルの違い，機能が正確に対応しないことなど，同じ環境では比較は難かしい。そこで，仮りに，汎用中型システムであるACOS-400Lで，ADBSのDMLにより似た処理を行わせるとすると，DBMの方が，2倍からコマンドによっては10倍程度速くなっている。

4.2 プロセッサ間通信性能

MMIを介したメインメモリアクセスはACOSとは独立に行われる。転送時間はメモリサイクルスチールに成功したとき，1μsec/4wordであり，ACOSがメモリを参照しているときは，1サイクル遅らされる。

PSIを使用した交信時間は，DBM起動時間とDBMから送られてくる終了通知を処理する時間を含め，平均5msec要している。この時間は，直接のプロセッサ間通信を行えば，10μsec（予想値）ぐらいになるはずである。

4.3 システム性能

DBMを導入した場合のシステム性能は，DBMにオフロードされるデータベース処理がシステム全体の何%になるかが問題になる。

いま，対象システムを中型汎用システムとすると，DBMは2～10倍の性能を示す。ただし，IO処理やプロセッサ間交信時間は無視する。DBMが平均5倍の性能を示すとすると，汎用システムから80%以上，DBMにオフロードしても，DBMがネックにならない。汎用システムから80%オフロードできたとすると，システムスループットは，ホストネック(20%)で決まり，5倍に向上する。

しかし，CODASYLタイプ(ADBS)では，DBMにオフロードできるのは40%程度であり，ホストには60%の処理が残る。この場合，システムスループットは約1.7倍となり，DBM付加により，70%のパフォーマンス向上が得られよう。

一方，問い合せタイプでは，DBMにオフロードできる割合いは，70%に達し，この場合には，システムスループットは3倍になる。

システム性能/価格比からみると，汎用プロセッサに比べ，ミニコンやデータのプロセッサのコストはかなり低く，仮りに1/5とすると，オフロード率70%時，性能/価格比は3/1.2≒2.7，オフロード率40%時1.7/1.2≒1.4　と予想され，システムレベルで比較しても，DBMの導入効果は大きいといえる。

さらに詳しいDBMの導入効果は，文献[10]に待ち行列網による解析を報告している。

5. 高性能データベースマシン

実験システムで実現したDBMの評価から，専用プロセッサによって，データベース処理に適合したファームウェアモジュールや，システム管理を行うことにより，汎用システムを上まわる性能を実現できることが確認された。

この実験システムは当初考えられていた，高性能を追求した実現形態にはなっていない。ここでは，実験評価において気がついた処理アルゴリズムの変更すべき点，およびハードウェア化の効果について述べる。

Table Iで，GDIコマンド実行時間の大部分は，field-moveで費やされている。例えば，get_image_directの場合，imageを検索し，record addressを得るまでが340μsecに対して，7つのfield値を転送するのに3400μsecかかっている。この原因は，field descriptorをサーチする時間が長いこと，1つのfield値の転送にアドレス変換が入っていること等による。この時間はカタログ内のfield descriptorの格納構造，および，field値の転送方式を改善することで1/5以下にできることがわかっている。

一方，ハードウェア化の効果の大きい機能モジュールは，ATM(アドレス変換モジュール)とDXM(データ操作モジュール)である。現DBMでは，これらはファームウェア化されているが，それでもGDS全体の走行時間の40%以上をこの2つのモジュールの走行時間で占めている。さらに，ATMとDXMの走行時間比をみると，ほぼ1.8対1となっている。モジュールをハードウェア化したときの走行時間は，使用デバイスとハードウェアブロック図，タイムチャートを作成すれば，算出できる。

- ATM
  文献[4]で述べたTLB (table lookaside buffer)に似た構成にする。使用メモリはスタティックMOS RAMとすると1μsec以下で実現できる。したがって，現DBMに比べ，30分の1となる。
- DXM
  データベースバッファを64kビットRAMで実現し，データ中を4バイト以上にする。そして，ストリングのread/write/compare および コード変換，タイプ変換を行うハードウェアを導入する。これらにより，現DBMが1バイト当り1μsecだったストリングデータ処理を4バイト当り，500nsecにすることができる。

ハードウェアだけでなく，ファームウェアの導入効果も大きい。使用したミニコンピュータでは，ソフトに比べ，同じ処理をファームで実現すれば，4～10倍の処理速度の向上が得られる。しかし，使用メモリ量は，平均4倍ぐらい増える。コストパフォーマンスからは，やはり基本機能(DAMの一部とTCM)に制限するべきと思われる。

Fig. 9に，以上の処理見直し，ハードウェア化モジュール，ファームウェア化を行った場合の処理速度の向上を示す。例として，get-image-directの実行時間を示す。(1)は現DBMでの実行時間，(2)は，field-moveを改善した場合，(3)はATMとDXMのハード化と，DAMとGDIPのファームウェア化を行った場合の予測される実行速度を示す。

実現上，パラメータの受け渡しやハードウェアモジュールの起動オーバヘッドがみこまれるが，この概算には極端に表われないと考えられる。

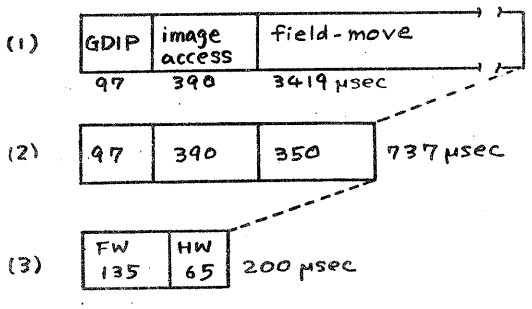

Fig. 9 Performance improvement

DBMの性能向上は，処理の見直しによって5倍程度改善され，ハードウェア化とファームウェア化によって，20倍まで改善されると予測される。

このように，データベースサブシステムの実行部は，ミニコンピュータあるいはそれに付加ハードウェアモジュールを導入すれば現状で，充分な性能向上が計れる。

DBMに要求される機能は，単体としての性能の他に，次の項目が満足される必要がある。

(1) 拡張性と稼動性
  DBMの性能の整合と稼動性を同時に満足する為は，多重化であり，DBMを多重プロセッサ構成にする必要がある。
(2) メモリ階層管理
  データベースバッファの管理方式もバッファサイズの選択とともに重要である。この方式は，実システムの測定結果の解析[11]やページ置換アルゴリズム[12]を検討している。
(3) DBページIO
  メモリ階層の高速化，高信頼性化は，高性能DBMをサポートするために要求される。この両方を解決するため，ペアディスク構想[13]の検討を進めている。
(4) プロセッサ間通信
  現DBMでは，ユーザプロセスとDB処理の同期制御をセマフォを介して行わせるため，PSIを使用している。このオーバヘッドが5msec要しているが，本来，プロセッサ間の直接のインタフェイスによって割り込み制御を行ない，セマフォをインタフェイスとして，プロセス間の制御を行うことにより，高速(数10μsecのオーダ)の通信が行われる。
(5) データリカバリ
  データベースでは，特にホストと独立になるDBMでは，完全なデータリカバリが要求される。この機能は，ディスクについては，ペアディスクシステム[13]，DBバッファについては，SYSTEM-R[6]や前に提案した文献[4]の，旧ページと新ページの両方を維持する方法が有効である。

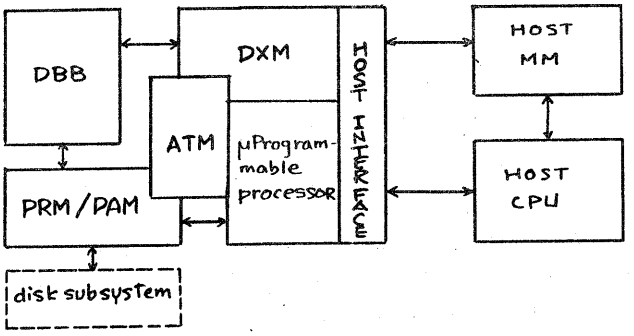

Fig. 10 High performance DBM configuration

Fig. 10 に高性能DBMの予定される構成を示す。DXM, ATMがハードウェア化され, PRMは, ページIOチャンネルとともに独立なプロセッサを構成する。マイクロプログラマブルプロセッサは, 他の機能モジュールを格納するために使用される。ホストインタフェイス部分は, ホストとなる計算機システムに適用するよう設計される。

DBMの高速化にともなって, DBMへのインタフェイスは, GDI (単一レコードインタフェイス) から, Query レベルインタフェイス (集合演算インタフェイス) に上げた方が効果が大きい。これについては, Query 分解, アクセスパス最適化方式, user view の維持方法について検討を進めている。

6. おわりに

汎用データベースサブシステム (GDS) を設定し, その機能をミニコンピュータを主体とするDBMに実現した。実験システムは, 汎用システムの主記憶をこのDBMが共有する機能分散型の多重プロセッサ構成をなしている。

ホストとDBMの論理インタフェイスであるGDIは次の特徴をもっている。

- 単一レコードインタフェイスであり, CODASYL の DML より高いレベルに設定し, Query 処理の有効なサポートを可能にしている。
- アクセスパスの自動更新, ロックの管理はGDSが行なう。
- データベース空間の管理はGDSが行なう。

このインタフェイスにより, ホストからの独立性を確保している。

実験システムの実現は, GDSの基本機能の実現性の確認と, DBMの有効性, 主として性能, を検証することを目的としており, 機能別に詳細に実行時間を測定している。これら測定結果にもとずき, DBMをミニコンピュータの規模で実現したときの性能と, さらに, ハードウェア/ファームウェア化を計った場合の性能について予測した。これらの結果は次のようにまとめられる。

- 現実験システムのDBMで汎用中型システムに比べ, 2～10倍高速である。
- 現DBMの処理を見直すことで, さらに5倍程度の改善が予測される。
- ハードウェアモジュールの導入, 主要モジュールのファームウェア化によって, 現DBMの20倍程度の性能向上が期待される。

この検討は, データベースシステムの実行管理部における比較であり, IOや, プロセッサ間通信は含めていない。この問題も独立に検討を行なっている。

この実験システムの評価から, 理にかなった規模, コストで高性能をもつDBMが実現できることを確認した。例えば, ミニコンピュータの規模で, 数Mipsの汎用システムに相当する性能を, ハードウェア化, ファームウェア化を企ることで, 数10 Mipsの汎用システムに相当する性能を得ることが期待される。

高性能DBMの導入は, 現在, CPUネックといわれている照合処理や query by example (QBE) を強力にサポートするとともに, さらに高度の機能をもつだろう知能データベースシステムへ性能の面から道を拓くと思われる。

謝辞

このデータベースマシン実験システムの開発において, コンピュータ技術本部, 基本ソフトウェア開発本部の北村部長, 登家部長, 関野主

任をはじめ多くの方々の暖かい支援を受けた。コンピュータシステム研究部，林澤部長には本開発に対し数々の便宜をはかって頂いた。(株)日本システムアプリケーション富林氏，小林氏にはソフトウェアの開発の多くを担当して頂いた。応用システム研究部，藤田氏にはPSI結合部を開発して頂いた。ここに，感謝の意を表わしたい。

[参考文献]

[1] 関野，植村： データベースマシン，情報処理，Vol. 17, No. 10, pp.940-946 (1976)
[2] E.A. Ozkarahan, et al.: RAP: An Associative Processor for Data Base Management", Proc. AFIPS NCC, Vol. 44, pp. 379-387 (1975)
[3] J. Banerjee, D. K. Hsiao: DBC - a database computer for large databases, IEEE Trans. Comput., C-28, No. 6, pp. 414 (1979)
[4] K. Hakozaki, et al: A conceptual design of generalized database subsystem, Proc. 3rd Int. Conf. on VLDB, pp. 246-253 (1977)
[5] G. Held, et al: INGRES - A relational data base system, Proc. AFIPS NCC, 1975, pp 409-416
[6] M.M. Astrahan, et al: SYSTEM-R: Relational Approach to Data Base Management, ACM Trans. on Databases, Vol. 1, No. 2, pp.97-137 (1976)
[7] 水摩，他：データベースマシンによるCODASYL型DBMSの実現と評価，電子計算機研究会
[8] 牧野：B-tree型インデックスにおけるページ内候用率，情報処理学会論文誌 20,5 (1979)
[9] 水摩，他：データベースマシンの同時制御，電子計算機研究会，EC 78-45 (1978)
[10] 日吉，他：データベースマシンを導入したシステム性能解析，EC78-4 (1978)
[11] 植村：データベースマシンにおけるバッファ管理方式の考察，EC78-44 (1978)
[12] 牧野：データベースシステムにおけるページ置換アルゴリズムについて，EC 79-29 (1979)
[13] 渡辺，他：ページディスク構想について，電子通信学会昭和55年度総合全国大会

Appendex A GDIコマンド（データ操作コマンド）一覧

```
GDI COMMAND LIST

 (1) RETRIEVE COMMAND
   1) Direct Access
      GET-PRIM-DIRECT (cur_id,key_value,field_list,predicate)
      GET-IMAGE-DIRECT (cur_id,key_value,field_list,predicate)
   2) Relative Access
      GET-PRIM-NEXT/PRIOR (cur_id,field_list,predicate)
      GET-IMAGE-FIRST/LAST (cur_id,field_list,predicate)
      GET-IMAGE-NEXT/PRIOR (cur_id,field_list,predicate)
      GET-LINK-FIRST/LAST (cur_id,field_list,predicate)
      GET-LINK-NEXT/PRIOR (cur_id,field_list,predicate)
      GET-LINK-OWNER (cur_id,field_list,predicate)
      GET-SCAN-FIRST/LAST (cur_id,field_list,predicate)
      GET-SCAN-NEXT/PRIOR (cur_id,field_list,predicate)
      GET-CURRENT (cur_id,field_list)
 (2) UPDATE COMMAND
      STORE (cur_id [,key_value],field_value_list [,near rec_add])
      UPDATE (cur_id,field_list,field_value_list)
      DELETE (cur_id)
      CONNECT (cur_id,rec_add)
      DISCONNECT (cur_id)
 (3) ACCESS PATH CONTROL COMMAND
      INITIATE-PATH (area_id,path_id [,LOCK] [,PREFETCH])
      RELEASE-PATH (cur_id)
```