データベースシステム　94−30
（１９９３．７．２３）

ハードウェア進歩のデータベース研究への影響

上林弥彦
高倉弘喜
京都大学工学部
目木信太郎
岡山県立大学情報工学部

本稿ではデータベース研究に関して、メモリ技術やＶＬＳＩ技術を反映するとどのような研究の可能性があるかについて検討している。データベースファイルの研究では、主記憶は乱アクセス、２次記憶は高速順アクセスを利用するのが普通である。しかし、最近開発されている高速記憶素子は、内部にキャッシュを持っており、順アクセスによる高速化が可能となる。逆に、将来ディスクを置き換える可能性があるといわれているフラッシュメモリでは、従来のファイル構成で考えられていた大量の順アクセスの利用やデータの近接配置の利点の利用という面が失われてしまう。

従来のデータベースのハードウェアによる高速化はデータベース処理の高速化が中心であった。主記憶データベース等ではバックアップのオーバヘッドは無視できない。さらに一貫性の保持といった操作の効率化も特に重要といえる。最近ではシステム内で自由に論理関数を変更できるＦＰＧＡが実用化しており、一貫性の管理にこのようなハードウェアを使って効率化ができる可能性がある。ダイナミックマイクロプログラムよりもっと低いレベルでのハードウェアの変更の能力は、柔軟さの実現がソフトウェア的方法だけでなくハードウェア的にも可能であることを示している。

データベースの並列処理については多くの研究があるが、並列の特殊な場合であるパイプラインについてはあまり研究がなされていない。並列処理時の並行処理が困難な問題を含んでいることから、並行処理との親和性が高く、ワークステーション向きのＶＬＳＩが開発されつつある、パイプライン処理をもっと見直すべきであろう。

Effects of Recent VLSI Technology to Database Research

Yahiko Kambayashi
Hiroki Takakura
Faculty of Engineering, Kyoto University
Shintaro Meki
Faculty of Computer Science, Okayama Prefecture University

This paper discusses on new research topics utilizing recent advanced hardware technologies. Historically in database file organization, utilization of sequential access and clustering of data are two important techniques. Flash memory, which is said to replace conventional disks, can sequentially access only small amount of data, compared with disks, and data clustering will not contribute to improve system performance. On the other hand, recently developed high-speed RAM contains cache memory which contributes to speed-up sequential access. Most research on improvement of database performance using hardware was to develop hardware for relational database operations. Recently FPGA whose function can be modified during the system utilization has been developed. We may be able to improve system performance by using FPGAs to realize bottle-neck parts of the database software. Such techniques can be applied especially to active and real-time database systems. Pipe-line processing is a special case of parallel processing and recently pipe-line processors for workstations have been developed. Although pipe-line processing is rather restrictive, it can be combined with concurrency control mechanisms rather easily. Optimization for pipe-line processing is also simpler than that for parallel/distributed systems.

1．まえがき

研究における問題点の一つとして生物の進化と同じ定方向性のある点をあげることができる。ある利点のある生物がその利点のために他の生物より有利になり、その利点を生かす方向に進化し、最終的にはその利点が欠点となってしまうことが起こり得る。研究面では、ある時期の技術や状況に基づいて、抽象化やモデル化が行なわれそのモデル化に基づいた研究が行なわれるが、モデル自体が実際の方向とは異なる方向に進んだり、実際面が大きく変化して、モデルと実際の乖離といったことが生じる。本稿ではデータベース研究に関して、メモリ技術やＶＬＳＩ技術を反映するとどのような研究の可能性があるかについて検討している。

データベースファイルの研究では、主記憶は乱アクセス、２次記憶は高速順アクセスを利用するのが普通である。しかし、最近開発されている高速記憶素子は、高速化のために内部にキャッシュを持っておりキャシュサイズ以内のデータ集合のアクセスが高速となる。これは主記憶であっても順アクセスを利用することを考える必要のあることを意味する。逆に、将来ディスクを置き換える可能性があるといわれているフラッシュメモリでは、読み出しと書き込みで対称性を持っておらず、ディスクほどの大きなデータの順アクセスは考えなくてよい。すなわちフラッシュメモリでは、従来のファイル構成で考えられていた順アクセスの利用やデータの近接配置の利点の利用という面が失われてしまう。このようなことを考えると従来型のファイル構成は大きく変わる可能性がある。

従来のデータベースの高速化はデータベース処理の高速化であった。主記憶データベース等ではバックアップのオーバヘッドは無視できない。バックアップや回復処理、さらに一貫性の保持といった操作の効率化も特に重要といえる。最近ではシステム内で自由に論理関数を変更できるＦＰＧＡが実用化しており、ダイナミックマイクロプログラムよりもっと低いレベルでのハードウェアの変更の能力は柔軟さの実現が可能となってきた。ソフトウェア的方法だけでなくハードウェア的にも可変システムを作り得るので、動的な最適化などにも有用であろう。

データベースの並列処理については多くの研究があるが、並列の特殊な場合であるパイプラインについてはあまり研究がなされていない。並列処理時の並行処理が困難な問題を含んでいることから、並行処理との親和性の高いパイプライン処理をもっと見直すべきであろう。

本稿では以上の様な背景の元で、データベース技術に影響を与えると考えられる最近のハードウェア技術を展望し、いくつかの可能性のある研究課題を紹介する。

2．ハードウェアの利用可能性

関係データベースは、効率の低いことと基本演算が単純であるためにハードウェア化に適していたことから、研究の初期からデータベースマシンの研究が行なわれてきた。しかし、この１０年位はＶＬＳＩの進歩が盛んで、ある時点でのハードウェア技術を用いて設計製作しても、完成時には汎用ＣＰＵの速度に及ばないということや、ソフトウェアの互換性の問題があり市場性は必ずしもなかった。

データベースにおける処理には単なるデータベース演算に限らず、次のようなものがある。

Ａ．データベース演算
Ｂ．最適化処理
Ｃ．一貫性制約処理
Ｄ．並行処理
Ｅ．通信関係
Ｆ．バックアップおよび回復

従来ハードウェア化の主眼はデータベース演算の効率化にあったが総合的な効率化のためには他の部分でもハードウェア化が有効な部分もあるはずである。

［データベース演算］

アルゴリズムのハードウェア化はハードウェアアルゴリズムとして知られており、乗算回路等実用化しているものも多い。データベースでの１つの特色は、データ量が大幅に変化するという点である。６４ビットの乗算機で扱われる数字はせいぜい１６～６４ビットであるが、データベースでは１個から１００万個といった変化が起こり得る。このため次のことも重要である。

１）ハードウェア化は高速化のためであるので，できる限り大きなデータ量が扱えることが望ましい。しかしそのために大きなハードウェアを用意すると、ソフトウェアと異なりコストがかかる上にデータ量の少ないときは遊んでしまう。

２）ハードウェア化した場合にその上限を越えるデータ処理が必要となるため、ソフトウェア的手法と組合せが可能でなければならない。

１）についてはデータ量に対してハードウェア量（メモリは除く）がｎのオーダではなく$\sqrt{n}$とか$\log n$のオーダでないと実用的とはいえない。また、処理時間は回路のサイズで決まるのではなく、データのサイズで決まるのが望ましい。このことは、シストリックアレイ型の処理はデータベース向きでないといえる（回路を通る時間だけの処理を常に必要とする）。

２）については、データをハードウェアで扱える上

限量で分割し、部分的な処理を行った後にソフトウェア的方法で統合する。データベース演算の中で、選択、射影は部分的な処理のみでできるし、集約演算も単純である。ソートや結合は多くのソフトウェア的な処理を必要とする。

大量データのソータとして一般的に使われるToddのソータはハードウェア量log nの条件を満足している。しかし、処理時間が回路サイズとなる。これを改良した田中のソータは処理時間がデータサイズとなる（係数の2がきくので、データ量が多い場合がふつうなら有利とはいえないが）。Toddの考えに基ずくソータが、東大、東芝、三菱等で実用化しているのはこのような良い性質があるためである。

データベース応用の高度化に伴って、扱うデータの種類も多様化し、演算の種類も大幅に増加してきている。これらの演算の高速化については従来画像処理ハードウェア等が知られているが、例えば、オブジェクト指向データベース処理の高速化については研究が行なわれていない。

［最適化処理］

一般に計算が困難であると知られているＮＰ完全のクラスになることが多い。このような問題では扱うデータ量が少なくてもソフトウェア的方法では最適化を扱うには時間がかかりすぎてしまう。しかし近似解と最適解ではディスクアクセス回数等が違うために、最適解を求めるのにいくら時間がかかっても実際の実行時間が短くなる方が、全体としての処理時間は大幅に減少することができる。このため、時間をかけても最適解を求めることが望まれる。

［一貫性制約処理］

一貫性は常に満足されることを求められるため、近似的な方法は用いることができない。データベース応用が広がるにつれてデータ間の意味制約としても種々のものが考えられ、この効率化は大きな問題となってきつつある。一貫性と関連する能動データベースでは、条件を満足するとどのような動作集合が引き起こされるのかとか、動作の連鎖の矛盾の検出、可能な動作のうちどれか一つの選択といった点に問題が起こる。また、ある条件が満足されたらどのようになるかを試験的に調べてみる必要のあることが生じる。このような場合にも高速の処理を必要とする。

［並行処理］

並行処理は一般に複数の資源があるときに、それらを全てうまく使うという場合に必要である。この資源の中に非常に速度が速いものがあるときは、並行処理制御も高速でなければならない。このようなハードウェア資源を用いる場合に、並行処理の効率化は非常に重要である。ホットスポットデータの並行処理も高速化が必要である。並行処理のハードウェア化やロールバックの連鎖、デッドロック検出の効率化等の問題がある。

［通信］通信関係では、一般の計算に比べてデータベースではデータの通信量が非常に多いので、通信コストを減少させることが非常に重要である。この問題は最適化処理になるが、例えばデータ圧縮等の効率化などはハードウェア利用になる。圧縮されたデータの上でもデータベース演算を適用できるなどということも必要であるために、圧縮するための符号はどのようなものでもいいというわけではない。

［バックアップおよび回復］ある種のソフトウェアの応用では非常に大きなオーバヘッドを要する場合がある。このためにハードウェア的な方法を併用するということも重要である。

3. 最近のメモリテクノロジー

近年の半導体技術の進歩にはめざましいものがあり、従来の半導体素子とは性質が大きく異なるものが次々と開発されている。ここでは、それらのうちメモリ素子について述べる。

A. 高速DRAM

従来のキャッシュ用の高速SRAMはビット当たりのコストが通常のメモリに比べ遥かに高かったため、大容量キャッシュを持つシステムはほとんど存在しなかった。これに対して、近年、高速DRAMと呼ばれるキャッシュ内臓型DRAMが開発された。現在、Rambus、Synchronous DRAM、Cach DRAM、Enhanced DRAMの4種類の高速DRAMがそれぞれの優劣を競いあっている。構造的にはどの製品もほとんど同じで、1チップのメモリセルをいくつかのバンクに分け、それぞれのバンクでセンスアンプか内蔵高速SRAMをキャッシュとして利用する。それぞれのバンクを交互にアクセスすることにより、アドレスが連続していればキャッシュのサイズを超える大容量のアクセスでも見かけ上ミ

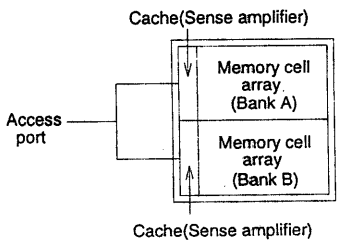

図 1: 高速 DRAM の構成

アクセスサイクル	キャッシュヒット時	2~10 ナノ秒
	ミスヒット時	30~100 ナノ秒
キャッシュ容量 (1 メガバイト当たり)		512 バイト ~4 キロバイト
チップのコスト		通常の DRAM とほぼ同じ

表 1: 高速 DRAM の仕様

スヒットが存在しないため高速アクセスを行うことができる。高速DRAMの仕様を表 1 に示す。

このように高速DRAMを利用すると、安価な大容量主記憶と大容量キャッシュを持つデータベースシステムを実現することが比較的容易に可能になる。

これまで主記憶は乱アクセスでも順アクセスでもアクセスに要する時間は同じであったが、高速DRAMを主記憶に利用する場合はできるだけ順アクセスをするようなデータ構造が必要になると考えられる。

B. フラッシュメモリ

従来のEEPROMでは、データの書き換えに際し、システムから取り外し、専用の書き込み・消去装置を用いる必要があったが、EEPROMの一種であるフラッシュメモリではシステムに実装したまま書き込みおよび消去が可能である。フラッシュメモリにはNOR型とNAND型の2種類があるが、ここでは大容量の観点から考えてNAND型について述べる。従来のEEPROMは複雑な工程で製造されていたため高密度化や低価格化が困難であった。しかしフラッシュメモリの場合は、DRAMと同じ工程で製造することができ、さらに同じ面積のDRAMの4倍の容量が実現できる。このため、フラッシュメモリのビット当たりのコストはかなり低く、今後ディスクに代わる記憶媒体として期待されている。

フラッシュメモリは不揮発のメモリセルとレジスタで構成される。データのアクセスはレジスタを介して行われる。レジスタの読み出し開始アドレスは任意に指定できるが、以降のデータは順アクセスで読み出さ

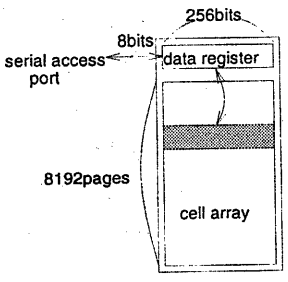

図 2: フラッシュメモリの構成

レジスタサイズ		256、512 バイト
ブロックサイズ		4 キロバイト
アクセスサイクル		60 ~ 100 ナノ秒
データ転送速度	読み出し時	10 ~ 15 マイクロ秒
	書き込み時	35 ~ 40 マイクロ秒
消去時間		6 ~ 10 ミリ秒
消去回数の制限		10 万~ 100 万回

表 2: フラッシュメモリの仕様

なければならない。既にデータが書き込まれているブロックに書き込む場合は、書き込みの前にブロックを消去する必要がある。現在のフラッシュメモリの仕様を表 2 に示す。

このようにフラッシュメモリでは、消去が必要となる最悪の場合、読み出しに対して書き込みにかかる時間が100倍以上となる点がディスクなどの磁気記憶と異なる、

C. 不揮発DRAM

通常のDRAM内でメモリセルを構成するコンデンサは時間経過に伴い自然放電により電荷を失うため定期的にリフレッシュで再充電し、読み出しはコンデンサの電荷を抜き取るため次のアクセスの前にプリチャージで電荷を蓄えなおす必要があった。コンデンサの充填剤として利用されている誘電体に電圧を加えると電界の向きに従って分極を生じる。さらに強誘電体では、電圧を加えなくなった後も分極したままの状態を保つことができる。この性質を利用して、最近コンデンサに強誘電体を用いた不揮発DRAMが開発された。不揮発DRAMでは誘電体の分極状態でデータを記憶することによりリフレッシュ不要かつ電源の供給なしにデータを保持することができる。

しかし強誘電体が分極に因って劣化するため、書込・読みだし回数の合計で寿命が決まってしまう。すなわち、アクセスの度にプリチャージするため、読みだし回数まで影響する。不揮発DRAMの寿命をのばすため、強誘電体薄膜と半導体薄膜を重ね合わせ、半導体薄膜の抵抗値が分極の状態により変化することでデータを読み出す方式が開発されつつある。この方式ではデータ読み出し後も分極状態に変化がないため、プリチャージの必要がなくなり、強誘電体の寿命は書き込み回数にのみで決められることとなる。このような不揮発DRAMはSRAMとまったく同様に利用できる。現在の不揮発DRAMの仕様を表 3 に示す。

将来的には、アクセスサイクルは10ナノ秒程度、書

アクセスサイクル	20 ~ 200 ナノ秒
書き換え回数	$10^8 \sim 10^{13}$ 回
容量	4 ~ 64 キロビット

表 3: 不揮発 DRAM の仕様

き換え回数は10の15乗回、容量もDRAM並になると考えられる。

4.半導体２次記憶の条件

ファイル構成や、２次記憶を考慮したデータベースアルゴリズムでは、ディスクの持つ次のような性質が利用されてきた。

（１）データは大量であるため、基本的にディスクに蓄えられ、必要に応じて主記憶に読み出す。

（２）２次記憶では、同じデータ量を読み出すのに順アクセスの方が乱アクセスを何度も繰り返すよりもはるかに高速である。また、ヘッドがトラック間を移動する場合、その距離の総和が短い方が時間がかからない。

（３）同時に読み出す可能性が高いデータが、同じトラック上や、近くのトラック上に置けないときは、別のディスクにおいて並列に読み出すことを考える。

データ量がｎの時、これを処理するアルゴリズムは、少なくともＯ（ｎ）かかる（データの読み出しに必要な時間）。主記憶と２次記憶の間の転送を考えると、等価的に主記憶のアクセス速度がｋ倍になると見做してよい。このため、主記憶アルゴリズムでＯ（ｎ'）のアルゴリズムは、２次記憶を考えてもやはりＯ（ｎ'）である。従って、データベースアルゴリズムでは、この係数に相当するｋをいかに小さくするかが問題となる。主記憶のアクセス速度を１００ナノ秒として、ディスクの乱アクセスをヘッドの位置決め時間の１０ミリ秒とすると速度差は１０万倍にも達する。アルゴリズムは、ディスクアクセス回数や、ディスクアームの動く距離の和で評価される。このようなディスクの性質を利用したものに次のものがある。

１）索引構成：Ｂ木などの索引では、一つの節点に多数のポインタが含まれる。これは、順アクセスによって一度に主記憶に取り込めるためである。Ghoshによって導入された一連検索ファイルはよく使われる質問集合に対応するデータ集合が全て一連になるように並べるものであった。上林らが拡張した準一連検索は、あるバッファサイズ内に並べればよいというものであった。これはバッファサイズの内容を読み出すという仮定のもとで同時に必要なデータの間が飛んでいてもよいということで一般化しており、ディスクの順アクセスの高速性を利用しているものである。

２）グループコミット：コミットをまとめることにより、ディスクアクセス回数を大幅に減らし、トランザクション処理の効率化をはかっている。

３）再帰性の計算：アルゴリズム解析では標準的なグラフの到達可能性アルゴリズムなどであるが、ディスクアクセス回数を考えることにとり多くの論文が発表されている。

今後開発される半導体ファイルでは、つぎのような性質がある。

１）アドレス指定を簡単にするため、或る程度のブロック単位でのアクセスが可能である。

２）バッファの利用により、等価的に順アクセスを高速化できる。

３）しかし、近接性などの性質は無く、順アクセスの出来ない所はどこも同じ時間になる。

４）主記憶との速度差が少ない。

５）乱アクセスと順アクセスの間の速度差はディスクほど大きくない。

以上は、ファイル構成とデータベースアルゴリズムについてであったが、もうひとつ重要なのは並行処理である。

ディスクなどを利用するときにＣＰＵが遊ぶのを防ぐために、並行処理によって複数の処理を実現する。主記憶データベースでは、この問題は起こらない。主記憶のみを用いるという仮定では、直列実行の方が応答時間の総和を最小化できる。しかし、通信を行ったり、利用者の応答を待つなど、cpuが遊ぶ要因は存在している。半導体２次記憶では、遊び時間が少なくなるので、並行多重度がディスクよりも少なくて最適となるであろう。アルゴリズム設計では、データの近接性が重要であったが、半導体ディスクディスクでは、その必要はない。データベースアルゴリズムでも、単独実行しか考えていないものが普通で、並行実行の場合は近接性は利用出来ていなかった。例えば、或るトランザクションを実行して或るデータを順アクセスしたもち、別のトランザクションに切り替わると、ヘッドは全く異なる位置に行くため、元のトランザクションに戻った場合、次にアクセスするデータが前にそのトランザクションを実行したときにあったヘッドの位置に近くても効率の改善は不可能である。このようにデータベース並行処理の影響を全く考えないデータベースアルゴリズムの論文が多く採用されているのは、データベースアルゴリズムの研究者が並行処理を全く知らないと言うことであろう。半導体２次記憶では近接性を利用していないので、この問題は生じない。

5.フラッシュメモリによるファイル構成

フラッシュメモリは以下に述べるように従来のような記憶媒体(ディスクやRAM)と大きく異なる性質を持つため、これまでのB木などのデータ構造がそのまま利用できるか検討する必要がある。

フラッシュメモリにおけるレジスタ-メモリセル間のデータ転送時間は、書き込みが読み出しの約4倍と

なっている。また、データ書き換えの場合は消去時間が加わるため100倍以上の時間を必要とする。さらに、読み出しまたは書き込みの最小単位がレジスタであるのに対して、書き換えはレジスタ十数個をまとめたブロック単位である。このようにフラッシュメモリでは読み出しと書き換え(書き込み)がアクセス単位と速度の点で非対称である。

さらに、ブロックの消去回数に制限があるため、全てのフラッシュメモリの書き換え回数をできるだけ均等にし、ブロックが利用できなくなると別のブロックを利用する方式が必要である。また、ディスクアレイ技術のように連続したデータを複数のフラッシュメモリに分散させることは、データ書き換え時における消去対象のブロック数を急激に増加させる。このため、連続性のあるデータは1つのブロックに記憶させる方が良い。

例えば、一般にデータ構造は木状のインデックスで管理される場合が多い。木の場合、キー値を挿入・削除すると複数の節点を更新するし、さらには木のバランス調整のため大がかりな節点の移動を生じる場合もある。このような木をそのままフラッシュメモリ上で実現すれば、多くの消去操作を必要とすることになる。

しかし、フラッシュメモリは二次記憶用の媒体として開発され、主記憶には従来からのRAMや高速DRAMの方が適している。また、主記憶におけるデータの管理もやはり木による方式が効率的であると考えられている。このため、いくらフラッシュメモリに最適なデータ構造であっても、主記憶のデータ構造にマッピングしにくいものであっては意味がない。このため、主記憶のデータ構造にできるだけ近くて、消去回数が全体的に均等でかつ少なくなるデータ構造を考える必要がある。

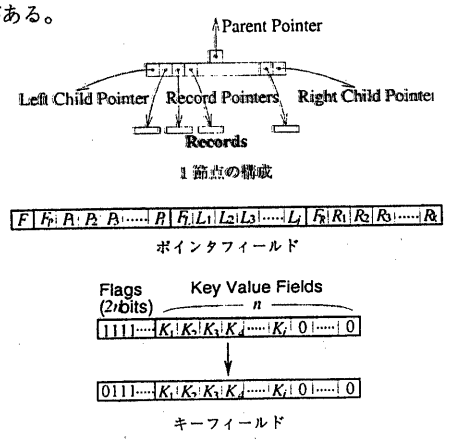

図 4: F 木の構成

消去回数を減らすためには、まずキー値とレコードを分け、節点にはキー値とレコードを指すポインタおよび節点の親と子を指すポインタのみを存在させる。1ブロックは数百バイトの容量を持つため、キー値とポインタの組合せでは数十組が存在し得る。

これだけでは、木に対する操作とデータ更新操作を単に分けただけなので消去回数を大幅に減らすことはできない。このため従来考えられてきたフラッシュメモリの利用法とは異なる上書き操作を導入する。消去直後のメモリセルのデータは1であり、0を書き込むまで1を保ち続ける。一方、一度0を書き込んだメモリセルは1を書き込もうとしても0のまま変化しない。この性質を利用すると次のようなデータ構造が考えられる。

1節点はレジスタ数個分の容量を持つものとする。節点はポインタフィールドとキー値フィールドの2つのフィールドから構成される。ポインタフィールドは図のように4つの部分に分かれポインタフィールドの有効フラグ、親(P)と左(L)右(R)の子のポインタを複数持つ。ポインタフィールド有効フラグとそれぞれのポインタ群の先頭にある有効フラグは次のように利用される。

ポインタフィールド有効フラグ
- 1: ポインタフィールドは使用可
- 0: ポインタフィールドは使用不可

親子ポインタ有効フラグ
- 1…111: ポインタ群は未使用
- 1…110: ポインタ1が有効
- 1…100: ポインタ2が有効
- ……
- 0…000: ポインタi(jまたはk)が有効

キーフィールドの先頭部にはキー値とポインタの組数(n)に対して2nビットがフラグとして割り当てられている。2ビットで1つのキー値の状態を示す。それぞれの状態は以下の通りである。

- 11: 初期状態(何も書き込まれていない)
- 01: キー値は有効
- 00: キー値は無効

各フィールドの初期状態は全てのビットが1であり、木に対する操作は、必要なビットに0を上書きするだけである。また、ポインタフィールドかキー値フィールドのどちらかを使い切ってしまうと、新しく節点を作り直す。同じブロックに存在する節点が全て無効であればブロックを消去する。

6.パイプライン処理

近年、ワークステーション用のパイプライン演算回

路等が開発されておりパイプラインの利用は重要である。

並列処理とその特殊な場合であるパイプライン処理は共に並列性を生かしたものであるが、パイプライン処理は並列処理とは異なる利点をもっている。

（1）パイプライン処理では処理時間がほぼ同じであることが要求される。並列処理では全く処理時間の異なる操作を並列に扱うことができる。

（2）パイプライン処理ではデータの処理順が固定しており、同じ操作をやり直した場合でも処理の順序は同じである。

（2）はパイプライン処理の特色となっており、このことから

（2－1）処理の再現性が高い。例えば、並列処理の場合は、例え同じ操作であっても処理をするたびに処理順が変化してしまう。このため、処理中のログを正確に取っておかなければ回復処理が困難である。パイプライン処理の場合は、どのデータまで進行したかが分かれば処理は簡単に再現できる。したがって、ログのオーバヘッドが小さい

（2－2）処理の複合化が容易である。これは、相続く処理が同じデータアクセス順で適用されるため、これらをまとめて、データ一度の読み出しで処理を複合化できることがあり、効率化がはかれる。

（2－3）並行処理制御が容易である。同じデータ集合に対して後で開始した処理が前の処理を追い抜くことはないので、トランザクションの直列順を保つのが容易である。

データベース演算として、射影後に重複する組を取り除く非冗長な射影を例に取ってみる。

アルゴリズム：重複する組の除去

step 1．射影によりいくつかの属性を消去した関係の組の集合をTとし、T'＝U＝U'＝ϕとする。

step 2．ハッシュテーブルを初期化してから、各要素の属性の値からハッシュの計算を行ない、ハッシュテーブルに全ての要素を登録する。その際ハッシュの衝突は無視して、ハッシュテーブルに重ね書きを行なう。

step 3．このハッシュテーブルを引いてTの各要素が実際に登録されているかどうか調べる。要素tがハッシュテーブルに登録されていれば、U＝U∪｛t｝とし、もしハッシュテーブルに登録されていなければ、T'＝T'∪｛t｝とする。

step 4．T'の要素t'に対して対応するハッシュテーブルのエントリに登録されている要素と比較を行う。もし異なっていれば、T'＝T'－｛t'｝、U'＝U'∪｛t'｝とする。

T'に残った要素はハッシュテーブルに登録されている要素と重複している要素なので取り除く。

step 5．U'＝ϕならば終了。そうでなければ、T＝U'、T'＝U'＝ϕとして、step 2へもどる。

ユニークな組と、重複する組を代表する組は、このアルゴリズムを実行した後、集合Uの要素として得られる。step 2からstep 5の過程は、ほぼ全過程がパイプライン処理できる。

実はこの方式は、並列処理のアルゴリズムでも実行可能である。しかし、並列処理では重ね書きの順度は異なってしまうので、結果として得られる組は処理を実行する度に異なってしまう。それに対して、パイプライン処理を行った場合には常に同じものが得られる。

ある関係に対して、属性値Aを読む操作、書く操作、走査を、それぞれ、R(A)、W(A)、S(A)と略記することにする。その他の操作はP_iで示す。上記のアルゴリズムでは、ハッシュする属性値のREADのあとに、この操作固有の操作が続くので、R(A) P_hと表現出来る。属性Aの値が或る条件を満足する組を見つけて、属性Bの値に10を加えるという更新操作は、S(A) R(B) P_* W(B) となる。走査および読み出しには可換性があり、同じ属性に対する走査や読みだしをまとめて、複合処理とできるので、一般のパイプラインで扱っていないデータベース向きの最適化が可能である。

7.自由度の高い論理回路の利用

最近、利用中に動的に論理関数を変えることができるＦＰＧＡが製品化され、ソフトウェアと同程度の自由度を持つハードウェアが入手可能となった。このようなハードウェアをソフトウェアのボトルネックとなる部分に採用するとシステムの効率を大幅に向上させうる可能性がある。

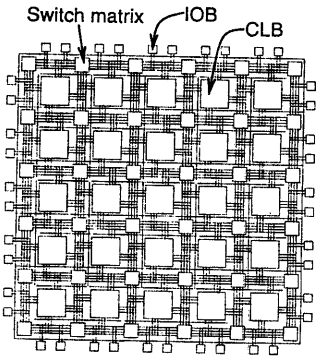

図 3: LCA の構成

A. FPGA(Feild Programmable Gate Array)

PLD(Programable Logic Device)を利用することにより、ある程度の任意の論理回路を実現できるため、コンピュータシステムのチップ数を減らすことができる。これまでシステムを小型化するためにPLDがよく利用されてきた。しかし1つのPLDでは10変数程度までの簡単な順序回路や同期回路しか実現できなかったため、さらに大規模の回路を扱えるFPGAが開発された。FPGAには従来のPLDを複数含むようなCPLD、SRAMで内部スイッチや論理回路の状態を記憶するSRAM FPGA、高電圧を加えて端子間をショーとさせることにより論理回路を実現するAntifuseFPGAの3種類がある。ここではそのうちの1つであるSRAM FPGAについて述べる。これは回路の動作中にSRAMの内容を変えることにより、論理回路を自由に変化させることができる。

SRAM FPGAの代表的なものにXilinx社のLCAがある。LCAは数千から数万のゲート数を1つのチップで実現することができる。LCA内部には小型のPLDと同等なコンフィギャブル・ロジック・ブロック(CLB)が行列状に存在し、チップの周囲に外部I/Oを実現するインプット・アウトプット・ブロック(IOB)が存在する。それぞれのCLBの回りには縦横にバスが存在し、バスの交差部でスイッチマトリックスによって縦と横のバスの接続・非接続を行なう。CLB、IOB、スイッチマトリックスそれぞれの制御は内蔵されているSRAMに書き込んで行なう。このため電源を入れる度にSRAMに書き込む必要がある。将来的にはSRAMをフラッシュメモリや不揮発DRAMに置き換えることで不揮発化できると考えられる。

B. BDD

BDDは大規模な真理値をソフトウェア的に圧縮して記述する技術である。FPGAである程度の大きさの論理回路はソフトウェアと同じ位柔軟に扱えるようになった。より大きな論理関数を扱う場合、併用するソフトウェア的方式として有望である。

論理関数を利用できるものとしては、一貫性制約の保持、能動データベースのＥＣＡ機構（例えば、ペトリネットである程度シミュレートできるので、オートマトンを実現する必要がある）、質問処理の最適化などである。

8.むすび

本稿では、新しいメモリ、パイプライン、ＦＰＧＡといったハードウェアとデータベース研究との関係について検討した。ディスクが半導体に置き換わるとデータベースファイル構成がかなり変わる可能性のあることを指摘した。一般の並列処理では並行処理性との両立が困難であるが（これは直列可能性という直列性を要求するため）、パイプラインはその問題が少なく並行処理も考えた並列処理として有望である。また、自由どの高い論理回路であるＦＰＧＡは、ソフトウェアの高速化に対して有望で、データベース応用の高速化に伴って生じる効率低下を助けるのに有効と思われる。これらの新しいハードウェア利用に関する研究を怠って従来型のハードウェア上でのデータベースの研究を続けてゆくのは楽ではあるが研究全体が古典となってしまう可能性がある。

文献

1 R.E.Bryant : "Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams," ACM Computing Survey, Vol.24 No.3 September 1992, pp.293-318.

2 Y. Kambayashi : "A Database Machine Based on the Data Distribution Approach," Proceedings of the AFIPS National Computer Conference, Vol.53, pp.613-625, July 1984.

3 Y. Kambayashi, S. P. Ghosh : "Query Processing Using the Consecutive Retrieval Property," in Query Processing in Database Systems, pp. 217-233, Springer-Verlag, 1985.

4 喜連川優："データベースマシン," 情報処理, Vol. 28, No. 1, pp.56-67, 1987.

5 M.H. Eich : "Main Memory Database Research Directions," Proc. 6th International Workshop, IWDM`89, 1989, pp.251-268.

6 I. Kojima, Y. Kambayashi : "Hash-Based File Organization Utilizing Large Capacity Main Memory," Proceedings of the International Conference on Foundations of Data Organization, pp.41-50, May 1985.
Also in Foundations of Data Organization, S. P. Ghosh, Y. Kambayashi, K.Tanaka (Eds.), Plenum Publishing Corp., October 1987.

7 H. Takakura, Y. Kambahashi : "A Design of a Transparent Backup System Using a Main Memory Database, " Proc. of 3rd International Symposium on Database Systems for Adbanced Applications, 1993, pp.178-185.

8 H. Takakura, Y. Kambahashi : "Continuous Backup Systems Utilizing Flash Memory," Proc. of 9th International Conference on Data Engineering, 1993, pp.439-446.

9 R.L.Ukeiley : "Field Programmable Gate Arrays," PTR prentice-Hall, 1993.