データベース・システム 73－6
オペレーティング・システム 44－6
（１９８９．９．８）

データベースオペレーティングシステム μOPT－Rにおける演算処理の高速化

水谷 高康　　國枝 和雄　　大久保 英嗣　　津田 孝夫

京都大学工学部情報工学科

あらまし　本稿では、パーソナルコンピュータ上に実現したデータベースオペレーティングシステムμOPT-Rにおいて用いられている演算処理の高速化のための手法について述べる。μOPT-Rでは、(1)データベース構造として階層転置型ファイルを採用することによって個々の演算に不必要なデータを識別し、処理対象となるデータ量を削減する、(2)ストリーム指向型演算処理方式によって処理の多重度を上げ、I/O処理時間とCPU処理時間を可能な限りオーバラップさせる、という２つの観点から高速な演算処理を実現している。

A High Speed Processing Scheme for Relational Operations in Database Operating System μOPT-R

Takayasu MIZUTANI, Kazuo KUNIEDA, Eiji OKUBO and Takao TSUDA

Department of Information Science, Faculty of Engineering, Kyoto University

Abstract　In this paper, we describe a high speed processing scheme for relational operations in database operating system μOPT-R which is implemented on personal computers. μOPT-R achieves a high speed query processing by adopting following two approaches: (1)reducing the amount of processing data by using Hierarchical Transposed File as database structure, (2)enhancement of the degree of multi-programming by using stream-oriented processing scheme, and overrapping I/O processing with CPU processing as much as possible.

１．はじめに

これまで大型の計算機システムの上に構築されることが多かったデータベース管理システム（以下DBMSと記す）は、ハードウェア技術の向上やパーソナルコンピュータの普及によって、パーソナルコンピュータ上に構築されるものが増加してきている。しかし、それらのDBMSは汎用オペレーティングシステム（以下OSと記す）の上に構築されていることが多い。このような汎用OSは様々な場面に対応できる能力を備えていることが必要であるので、必ずしもデータベース処理に最適な環境を与えているとは言い難い。また、現在実現されているパーソナルDBMSは処理速度などの点においてまだまだ問題がある。

我々は、データベース処理に最適な環境を提供することによってシステム全体として処理効率を向上させるという考え方に基づき、パーソナルな使用を目的としたデータベース専用OS μOPT-R を実現し、より優れたパーソナルDBMSを構築することを考えている。本稿では、特にμOPT-Rにおける演算処理の高速化のための手法について述べる。

２．μＯＰＴ－Ｒの特徴

本章では、μOPT-Rがデータベース専用OSとして備えている特徴的な機能について要約する。

(1)階層転置型ファイル

通常、関係データベースシステムでは、データをタプル単位に格納しているが、μOPT-Rでは、属性単位に１つのファイルを割り当てる階層転置型ファイル[1]を採用している。このため、演算が必要とする属性データ、しかも各属性に付加された副次索引によって演算の条件を満たす可能性のあるデータブロックだけにアクセスを行うことが可能となり、演算対象となるデータすなわち主記憶に読み込むデータを最小限におさえている。

(2)マルチタスク機能

パーソナルコンピュータの２次記憶はフロッピーディスクが主流となっており、そのアクセス速度が問題となる。演算処理の際、データの読み込みや結果の書き出しなど、I/O処理のためにCPUが待ち状態となる場面がしばしば見られる。こうしたI/O処理を行っている間にCPUを有効に使用することができれば、システムの効率は飛躍的に向上すると考えられる。そこで、CPU処理とI/O処理のオーバラップを可能とするために、μOPT-Rではマルチタスク機能をサポートしている。

(3)ストリーム指向型方式による関係演算

μOPT-Rでは、関係演算の処理をあるひとまとまりのブロック、すなわちページ単位の処理に分割し、先に述べたマルチタスク機能をさらに活用して、処理の多重度を可能な限り高めている。この機能により、ほとんどの場合I/O処理時間をCPU処理時間の中に埋没させることが可能となる。従って、関係演算を高速に処理することが可能となる。また、将来μOPT-Rを分散環境に対応するように拡張する際には、基本的なアルゴリズムを変更することなくトランザクションを構成する演算を並列に処理することが可能となり、より一層の高速化が期待できる。

(4)ソフトウェアセグメンテーション

パーソナルコンピュータにおいて仮想記憶方式を実現しているシステムは少なく、アプリケーションプログラムはデータが主記憶上に存在するか否かを判定しなければならない。μOPT-Rでは、ソフトウェアセグメンテーションを実現することによってアプリケーションプログラムからこのような問題を除去している。すなわち、アプリケーションプログラムは処理対象となるセグメントの名前だけを意識すればよい。

３．階層転置型ファイルと関係演算[1]

関係演算は属性値に対する条件によって行われることがほとんどであり、必ずしも関係内のすべての属性データを必要とはしない。また、演算の対象となる属性の最小値、最大値を知ることにより、演算の条件を満足するデータ集合をある程度推定することが可能となる。この点において、各属性ごとにアクセスが可能であるμOPT-Rの階層転置型ファイル(HTF:Hierarchical Transposed File)は効率がよいと言える。本章ではμOPT-RのHTF

の構成について説明し、HTFの特性を利用した演算アルゴリズムについて述べる。

3.1 階層転置型ファイル

μOPT-Rではデータベースの論理構造として関係データベースを採用している。関係データベースは論理的には関係と呼ばれる表形式データの集まりである。μOPT-Rではこれに対する物理構造として、従来の、表を行方向に蓄積する方法とは異なり、列方向に蓄積する方法であるHTFを用いている。HTFは属性ごとに1つのファイルを割り当て、属性値によって転置して格納し、さらに副次索引ファイルを付加したものである。HTFは、以下の3種類のファイルによって構成される（図1参照）。

(1)ＰＤＦ (Piecewise Distribution File)

属性の実現値の分布状態を表現したビットマップ形式のファイルである。行が属性の実現値の分布区間、列がタプル識別子(TID)に対応する。このファイルは、TIDをもとにその実現値が格納されているBMF番号を求めるために使用される。

(2)ＢＭＦ (Binary Matrix File)

行が属性の相異なる実現値、列がTIDに対応するビットマップ形式のファイルである。属性値が文字列の場合、行はCTFの該当するエントリ番号になる。これによって属性の値からTIDを求めることが容易になる。

(3)ＣＴＦ (Compressed fully Transposed File)

属性の相異なる実現値を圧縮して格納したファイルであり、BMFに格納されているエントリ番号から該当するものを検索することによって必要な実現値を得るためのものである。このファイルは属性値がBMFに格納できない場合に用いられる。

μOPT-Rでは属性の実現値が極端に長くなる場合を除いてCTFは使用していない。すなわち、基本的には属性値をBMFに格納してPDFとBMFのみでHTFを構成することにした。これはすべての場合にCTFを生成した場合、セグメントのアクセス回数が増え効率の低下を招くと考えたからである。

3.2 データディクショナリ

μOPT-Rではデータベース管理のために各種のカタログを使用している。これらのカタログは、関係マスタカタログ、ドメインマスタカタログ、ビューマスタカタログをルートとした階層構造をなしている。各マスタカタログはシステム初期化時に読み込まれ、主記憶に常駐する。μOPT-Rでは、このデータディクショナリによって演算処理の高速化、データベースの一貫性制御や安全性制御を行っている。

3.3 演算アルゴリズム

従来のDBMSに見られるような水平方向にデータを格納するデータベース構造においては、ある属性にアクセスする場合、各タプルのすべての属性値を読み込む必要がある。しかし、HTFはBMFを単位としたファイル構成をとっているため、演算に必要な属性のみにアクセスすることが可能である。従って、データの読み込みに対するI/O回数を減少させることができる。一般に、関係演算はある属性の実現値に対する条件によって行われることが多いため、HTFによるデータ構造は非常に有効であると考えられる。

また、前節で述べたように、データベース管理のためのカタログ類は階層構造をなしており、各関係において各々の属性に対して1つずつ属性カタログが作られる。この属性カタログは各BMFに格納されている実現値の最大値および最小値を保有しており、それを調べることによってそのBMFが演

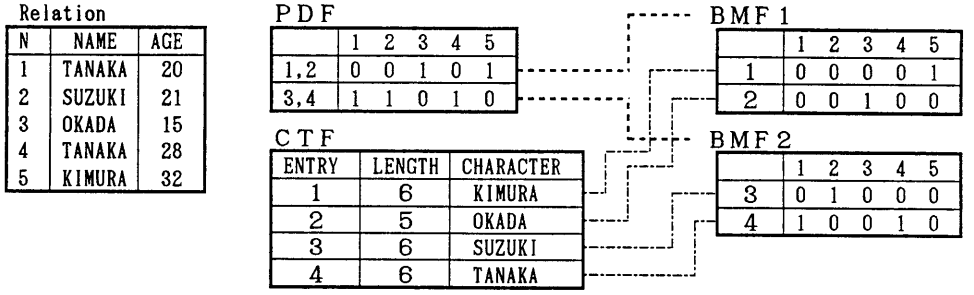

図1．階層転置型ファイル

算に必要か否かをBMFを読み込むことなく知ることができる（図２）。本節ではこのようなHTFの特性を利用した関係演算のアルゴリズムについて選択演算(selection)を例にして説明する。

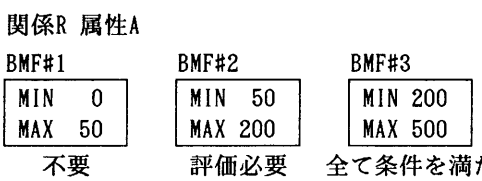

図２．BMFの区間演算

μOPT-Rでは、関係演算は以下の手順で行われる。

1.該当する関係カタログと属性カタログを読み込み、アクセスすべきBMFを求める。
2.該当するBMFを読み込む。
3.演算を実行する。

以下で説明する演算アルゴリズムにおいて使用する記号を次のように定義する。

$M_A(i)$ … 属性Aのi番目のBMFに格納されている実現値の最大値
$m_A(i)$ … 属性Aのi番目のBMFに格納されている実現値の最小値
θ … 比較演算子
C … 定数

μOPT-Rの検索言語では、選択演算は次のような形式をしている。

```
SELECT R.A FROM R WHERE R.B θ C
```

(1)BMFの区間演算

まず、属性Bに関する属性カタログの$M_B(i)$、$m_B(i)$を求める。次に、θ、C、$M_B(i)$、$m_B(i)$によってアクセスすべきBMFを求める。演算の対象となるBMFのBMF番号iの集合は以下のようにして求められる。

θが＝の場合 … $\{i \mid m_B(i) \leqq C \leqq M_B(i)\}$
＞、≧ … $\{i \mid M_B(i) \geqq C\}$
＜、≦ … $\{i \mid m_B(i) \leqq C\}$
≠ … {i｜iはすべてのBMF番号}

条件式の比較演算子θが≠の場合はすべてのBMFにアクセスしなければならない。そこで、θを＝として演算し、当該属性に属するすべてのTIDの集合と演算結果のTIDの集合との差(difference)をとることによって結果を生成する。

(2)演算の実行

まず、比較演算子θが＝の場合を考える。(1)で示した方法によってアクセスすべきBMFのBMF番号を求めた後に、当該BMF番号に対応するセグメントを要求する。そして、BMF内の条件を満たすTIDを中間結果として出力することによって演算が完了する。

次に、θが＞の場合であるが、(1)の方法では一般に多数のBMFが求められる。それらすべてのBMF中の実現値について条件の評価を行う必要はなく、境界となるBMF（すなわちθが＝の時にアクセスするべきBMF）の場合のみについて評価を行えばよい。それ以外のBMFに格納されている実現値はすべて条件を満たすことになる。つまり、境界となるもの以外のBMFに格納されているTIDは無条件に中間結果に出力すればよい。θが≧、＜、≦の場合も同様である。さらに、他の制約(restriction)、結合(join)などの関係演算についても同様にしてBMFの区間演算を行うことができる。

以上のアルゴリズムによって、主記憶に読み込むデータ量と演算条件の評価回数が削減される。このため、演算処理に要するI/O時間およびCPU時間を大幅に短縮でき、演算処理の高速化を実現することが可能となる。

４．関係演算のストリーム指向型処理

４.１　演算処理の概要

ここでは、μOPT-Rにおける基本的な演算処理機構について述べる。

ユーザの入力したトランザクションはいくつかのデータベースに対する問い合わせによって構成される。これらの問い合わせは、さらに関係演算の系列に分解される。この系列は、２分木を構成し、木の個々のノードが１個の関係演算や論理演算に対応している。μOPT-Rでは、これらの各ノー

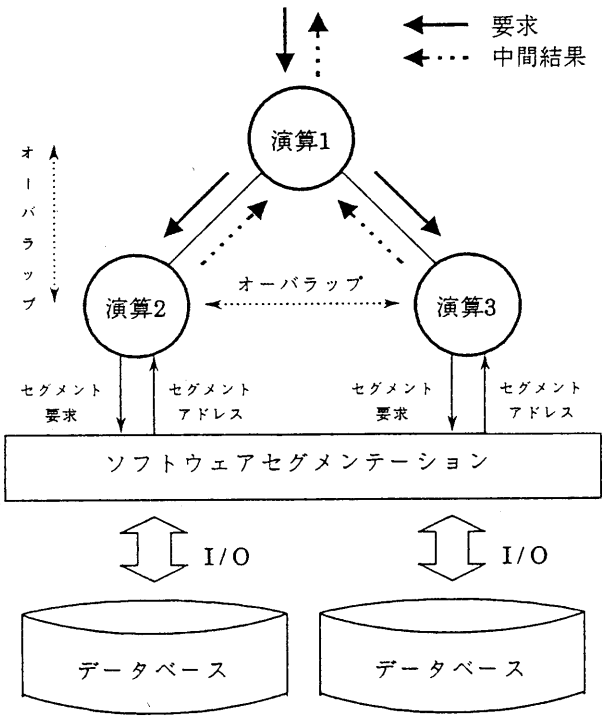

図３．μOPT-Rにおける演算処理

ドに１つのタスクを割り当てて処理を行っている。図３にμOPT-Rにおける演算処理の概略を示す。

各演算は入出力データによって３つに分類される（表１）。

表１．演算の分類

	入力	出力
葉ノード	原関係	中間結果
中間ノード	中間結果	中間結果
根ノード	中間結果	要求の答

葉ノードの演算は直接データベースをアクセスするものであり選択、制約、結合演算がそれにあたる。中間ノードの演算は、演算条件が複数の論理演算子によって結合されている場合に必要なものであり、中間結果の論理和もしくは論理積をとる。根ノードの演算は中間結果を入力としてユーザ要求の答えを求めるものである。μOPT-Rでは、中間結果は指定された条件を満たすタプルのTIDの集合で表されるため、最終的に条件を満たす属性の実現値を取り出す作業が必要となる。このため、演算の最後のフェーズとして根ノードの演算を実行する。

４．２　演算の駆動方式

演算木に分解された関係演算の駆動方式として、逐次駆動型、要求駆動型、データ駆動型の３つが考えられる。

(1)逐次駆動型方式

各ノードの演算の実行を制御フロー順序づけ(control flow ordering)[2]に基づいて行う方式で、実行を開始する前から静的に実行順序が決められている。処理の単位を分割することはない。つまり、全ての入力データに対して演算の実行を終了しなければ、結果のデータを次の演算に渡さない。従ってCPU処理とI/O処理のオーバラップは行われない。

(2)要求駆動型方式

この方式においては、演算の実行順序はデータフロー順序づけ(data flow ordering)[2]によって決定される。すなわち、各ノードは入力データがそろうと実行可能になる。ただし、要求が伝わっていないノードは入力データがそろっても処理は開始されない。あくまでも演算の駆動は要求が到着することによって行われる。この方式の処理においては、最初に演算木の根に該当するノードが要求を出し、それが下位ノードへと次々に伝播していく。そして、要求が葉ノードに到着すると演算を開始し、出力データが上位ノードへ次々と渡されていく。

(3)データ駆動型方式

要求駆動型方式と同様に、演算の実行順序はデータフロー順序づけに基づいており、入力データがそろった段階で処理が開始される。ただし、要求が存在しないという点において先の要求駆動型方式とは異なっている。すなわち、演算を駆動するのは入力データということになる。当然、この場合(2)で述べたような要求を伝播することによるオーバヘッドは存在しない。

４．３　ストリーム指向型処理

従来の演算処理方式では、問い合わせを構成する演算系列を１つ１つ実行し、演算木の上位ノー

ドに該当する演算は、それより下位のノードの演算がすべて終了するまで実行を開始することはできなかった。これでは演算がI/O要求を発行して、演算対象のデータが主記憶に読み込まれている間は、CPUは処理を進めることができず、高速な演算処理は望めない。したがって、問い合わせを構成する各演算タスク群中の並列（あるいは並行）実行可能性を検出し、CPU処理とI/O処理をオーバラップさせて、全体の処理時間を短縮することが考えられる。

CPU処理とI/O処理のオーバラップによる効果を十分に引き出すためには、この並行実行可能性を最大限に検出することが必要である。μOPT-Rでは演算木レベルにおいて以下の２点に関してこの可能性を検出している。

(1)水平方向の並列性（兄弟ノード間の並列性）

問い合わせを構成する各演算間にはデータの受渡しによる実行順序に関する依存が存在する。μOPT-Rでは要求駆動型方式で演算タスクを駆動する場合、その依存関係を表した隣接行列を用いて並行実行可能部分を検出している[3]。これによって検出された並行実行可能なタスク群を一斉に起動することにより複数のタスクが実行可能状態になる。そして、データ入力等でI/O要求を発行したタスクは待ち状態になり、実行可能状態にある他のタスクが走行することになる。これにより、CPU処理とI/O処理のオーバラップが実現される。また、データ駆動型方式においては、この並行実行可能性は暗黙のうちに検出される。

(2)垂直方向の並列性（親子ノード間の並列性）

ストリーム指向型処理方式によって検出されるのがこの並列性である。ストリーム指向型処理方式は、要求駆動型もしくはデータ駆動型方式によって各演算ノードが処理を進め、さらに個々の演算処理の単位を１ページごとに分割したものである。これにより、入力となる中間結果を計算する演算ノードが結果を１ページ出力した時点で、上位ノードの演算の実行を行うことが可能となる。μOPT-Rでは、関係を構成するタプルのタプル識別子がストリームの各データ要素に対応し、その系列がストリームとなる。

以上の２種類の並列性の検出により、多種多様なオーバラップを実現することが可能となり、CPUのアイドル時間を可能な限り短くすることができる。したがって、パーソナルDBMSにおける最大の問題点であるI/O処理時間をCPU処理時間の中に埋め込む可能性が高くなり、関係演算の高速処理を実現することができる。また、演算結果はその演算がまだ終了していなくても演算木の上位ノードの方へ次々に渡されていくので、システムの応答時間は大幅に短縮される。

一方、ストリーム指向型方式では各演算ノード内に大量の中間データを残すことになる。これは、２項関係演算がインナーリレーションの全ページとアウターリレーションの全ページを付き合わせなければならないためである。つまり、どちらか一方の演算が完了していない場合、もう一方の入力データを捨てることができないことになる。マルチプロセッサシステムにおいては、この問題を解決する方法としてインナーリレーションの再計算によるアルゴリズムが見いだされている[4]。しかし、この方法をシングルプロセッサシステムであるμOPT-Rに用いるとインナーリレーションの再計算によるオーバヘッドが膨大なものになってしまう。μOPT-Rではソフトウェアセグメンテーションを実現しているため、中間結果がメモリに入りきらない場合はそのセグメントが自動的に２次記憶へページアウトされる。従って、中間データを格納する場合のメモリ不足は発生し得ない。この場合I/O回数は増えるが、インナーリレーションを再計算するよりは短い時間で処理できる。

５．性能評価

以上述べてきたμOPT-RをNEC製のPC-9801 VM21（CPUはV30、クロックは10MHz）上に実現し、性能測定を行った。時間測定はμOPT-Rのタイマ機能を用いて行った（10ミリ秒単位）。

５．１　検索操作

データベースに対する問い合わせのうち、最も頻繁に行われるのが検索操作である。それゆえ、

検索操作の処理速度はシステムの性能を左右する重要な項目である。なかでも、結合演算(join)は多量のCPU処理を必要とするため、負荷のかかる演算である。

ここでは選択演算(selection)と結合演算(join)およびそれらを組み合わせた演算系列についての実験を行った（表２）。比較のために、microrim社のR:BASE5000において同様の実験を行った結果も挙げておく。なお、R:BASE5000での実験は作業領域をRAMディスク上に確保して行った。

実験に用いた演算系列は以下の４つである。

①selection
②join
③join(join,join)
④join(join(join,join),join(join,join))

いずれも、データベースのサイズは属性数が２つ、タプル数は400のものを使用した。

測定結果を見ると、選択演算については、R:BASE5000の処理速度よりもわずかに速いが、オーダ的には変わらない。しかし、結合演算およびそれを複合した演算の結果は5倍から15倍の処理速度となっていることがわかる。

５.２　変更操作

変更操作については、１タプルの挿入、削除、更新の実験を、データベースのタプル数を変えて行った（表３）。

３章で述べたように、μOPT-Rで採用しているデータベース構造HTFは、その性質上、検索条件の評価など属性単位のアクセスには多大な効果を発揮する。しかし、タプルの挿入、削除などの変更操作に見られるタプル単位のアクセスでは、通常のデータベース構成法に比べ処理時間が増大するものと思われる。測定結果を見ると、やはり処理速度は落ちるが、それでも数秒程度のオーダである。また、このようなタプル単位の操作は、対象とする関係のサイズに影響されないことがわかる。変更操作は頻繁には行われないことを考えると、この程度のオーバヘッドはシステムの性能にはあまり影響はないものと思われる。

表２．検索操作の実行時間

	(a)μOPT-R	(b)R:BASE5000	(b)/(a)
①	0.300	0.641	2.14
②	0.760	4.380	5.76
③	8.160	120.410	14.76
④	22.300	353.700	15.86

(単位:sec)

表３．変更操作の実行時間

関係サイズ	100	500	1000
insert	1.960	2.960*	1.980
delete	1.620	1.880	1.700
update	2.000	3.600*	2.420

属性数は５　(単位:sec)
＊ … BMFの分割を行った。

表４．応答時間

	逐次駆動型	要求駆動型
① 結果２ページ	30.980	3.460
② 結果４ページ	162.220	12.360

(単位:sec)

５.３　応答時間

システムの性能評価を行なう場合の重要な項目の１つに、応答時間(response time)がある。表４は、その応答時間を逐次駆動型方式と要求駆動型方式で測定したものである。実験は①、②共に５.１節で示した④の構造の演算系列を用いて行った。ただし、結果のサイズが①は２ページ、②は４ページとなるように設定している。表４を見ると、要求駆動型方式の応答時間は、逐次型駆動方式のそれの10%以下となっている。これはまさにストリーム指向型処理の利点である。

６．分散環境における演算処理方式

これまでの議論はすべてシングルプロセッサシステムとしてのものである。しかし、最近はパーソナルコンピュータをLANで結合したシステムも見られるようになり、信頼性など様々な局面から分散DBMSの必要性が高まっている。そこで、現在我々はμOPT-Rを分散環境に対応させるべくシステムを拡張するプロジェクトを進めている。

分散システムにおいては、演算を真の意味で並

列実行させることができるため、ストリーム指向型処理方式の効果をより高めることが可能である。μOPT-Rでは、演算木の個々のノードに１つのタスクを割り当て、演算間のデータの受渡しにタスク間通信を使用しているため、分散環境でもそのまま対応することができる。

ただし、演算対象データが各サイトに分散して格納されている場合には、通信コストを下げるためのアルゴリズムを考えることが必要となる。この問題に対してはsemi-joinを用いたものが知られている[5][6]が、μOPT-RではHTFを利用したより有効なアルゴリズムが考えられる。

semi-joinでは、通信データ量を削減するために対象となる属性データを射影したものを相手サイトに送信する。μOPT-Rでは、そのかわりに対象属性の属性カタログを送信し、相手サイトでBMFの区間演算を実行する。これによって演算条件を満たさないBMFを切り捨てることができ、通信データ量は大幅に削減される。

７．おわりに

本稿では、μOPT-Rにおいて演算処理の高速化に用いられている手法について述べた。μOPT-Rでは、２つのアプローチから演算処理の高速化を実現している。１つは処理対象となるデータ量の削減であり、もう１つはCPU処理とI/O処理のオーバラップである。前者に対応するものが階層転置型ファイルとそれを利用した属性データファイルの区間演算であり、後者に対応するものがマルチタスク機能およびストリーム指向型演算処理方式である。

ストリーム指向型処理は、関係演算の実行対象をデータのストリームとしてとらえることによって、より細かいレベルでCPU処理時間とI/O処理時間をオーバラップさせることを目的としている。μOPT-Rでは、ソフトウェアセグメンテーションによる仮想記憶方式を採用しているので、演算の中間結果の蓄積によるメモリオーバフローという問題に対処することが可能である。

実験の結果、演算処理速度としては既存のデータベースシステムの数倍から数十倍程度の高速化が達成できた。ストリーム指向型処理方式は、I/O処理速度とCPU処理速度の格差が非常に大きいパーソナルコンピュータでは有効な演算処理方法といえよう。

なお、本研究は、一部文部省科学研究費（課題番号01580025）による。

（参考文献）

[1]大久保英嗣，津田孝夫：階層転置型ファイルに基づく関係操作アルゴリズム，情報処理学会論文誌，Vol.26，No.1，130-138 (1985).

[2]J.A.Sharp:Data Flow Computing, Ellis Horwood Ltd. (1985).

[3]大久保英嗣，津田孝夫：データベースオペレーティングシステムシステムＯＰＴ－Ｒのタスク管理とトランザクションのスケジューリング技法，情報処理学会論文誌，Vol.25，No.4，544-551 (1984).

[4]清木康，長谷川隆三，雨宮真人：先行・遅延評価を用いた関係演算処理方式，情報処理学会論文誌，Vol.26，No.4，685-695 (1985).

[5]C.T.Yu: “Distributed Database Query Processing” in Won Kim et al.: “Query Processing in Database Systems” , Springer-Verlag Berlin, 48-61 (1985).

[6]Patrick Valduriez: “SEMI-JOIN ALGORITHMS FOR DISTRIBUTED DATABASE MACHINES” in H.-J.Schneider: “DISTRIBUTED DATA BASES” , North-Holland Amsterdam, 23-36 (1982).