関係データベース演算のベクトル計算機向きアルゴリズムの評価

5P−8

目木信太郎　梅田憲　上林弥彦

京都大学

1．はじめに

近年のデータベースの応用分野の拡大のために、高速なデータベース管理システム(DBMS)に対する要求は年々大きくなっている。コストに見合ったDBMSを開発するための比較的うまくいっているハードウェア的アプローチとして二つの方法がある。大容量の主記憶を備えた計算機を用いることと、マイクロプロセッサを用いて並列データベースシステムを構築することである。

一方で、近年の大規模数値計算の必要性に応えるべくスーパコンピュータとしてベクトル計算機が開発され、様々な分野の数値計算に利用されている。ベクトル計算機はベクトルデータに対する演算をパイプライン演算器で高速に実行することによって高い最大性能を実現しており、大規模数値計算において大量データを扱う必要から大容量の主記憶も備えている。したがって、ベクトル計算機上の主記憶データベースは高速なデータベースシステムとなることが期待できる。

筆者らは既に、ベクトル計算機の特徴を生かしたベクトル計算機向きのデータベース演算のアルゴリズムと、それをベクトル計算機上に実現して測定したベンチマーク結果を示した[3]。しかし、作成したプログラムはベクトル計算機向きに最適化したものであり、汎用機上での実行には必ずしも適していなかった。そこで、汎用機向きのプログラムを今回新たに作成して、より正確な評価を行った。

2．ベクトル計算機の概要

ベクトル計算機はベクトルデータ(配列)に対する演算をパイプライン演算器で高速に実行する計算機のことである。現在のベクトル計算機は浮動小数点数だけでなく、整数、論理型データなどもベクトル処理の対象としており、単純な四則演算だけでなく論理演算やシフト操作などもベクトル処理される。また、マスク演算機能を備えているので、IF文によって制御される演算もベクトル処理できる。

パイプライン演算器では複数のベクトルデータに対する演算がオーバラップして実行されるので、各演算が並列に実行できるものでなければベクトル計算機上での実行には適さない。総和演算、最大値・最小値を求める演算、ベクトル収集操作などは並列性がないので本来はベクトル処理できないが、現在のベクトル計算機ではこれらのマクロ命令を実行する特別のハードウェアが用意されているのでこれらもベクトル処理される。

主記憶上のデータに対しては連続アクセス、等間隔アクセス、リストベクトルアクセス(ベクトルデータを指標とする間接アクセス)の3種類の方法でアクセスでき、これらもパイプラインでベクトル処理される。

また、ベクトル計算機の性能を十分に引き出すためにはベクトル処理の比率(ベクトル化率)を高くする、ベクトル長(ループの回転数)を長くする、等の点に注意をしてアルゴリズムの設計及びプログラミングを行う必要がある。

3．データ構造

3.1 関係のデータ構造

データベースシステムはデータ量の変化に対応できるものでなければならないので、一つの関係もいくつかのブロックに分割して記憶しておく必要がある。残念ながら、現在のところ自動ベクトル化コンパイラはFortranのものだけが提供されているので、Fortranでこのシステムを記述する必要がある。Fortranでは配列だけがデータ構造として許されているので、データは3次元の配列に記憶することになり、三つの添字はそれぞれ、ブロックの通し番号、ブロック内での組の位置、属性を示す。

3.2 中間結果のデータ構造

主記憶には、ランダムアクセスが可能である、アクセスが非常に高速であるという特徴があるので、質問処理の途中で組を逐一移動したり複写したりすることは、ポインタを使って間接的に組にアクセスするよりも計算量が多いと考えられる。そこでこのシステムでは、質問処理の中間結果はTID(Tuple IDentifier、組識別子)を使って表現することにした。前節の考察から、ブロックの通し番号を示す添字と、ブロック内での組の位置を示す添字の対をTIDとして用いればよいことがわかる。

4．関係データベース演算のアルゴリズム

4.1 スカラアルゴリズム

4.1.1 射影

この演算は、1つの関係の中の幾つかの属性を消去するものであるが、質問言語のSQLでは二つのタイプの射影質問が提供されている。一つは結果の関係の組がユニークであることを保証するものであり、もう一つは結果の関係の組は必ずしもユニークではないものである。後者の射影質問の場合、そのアルゴリズムは自明であるので、ここでは前者の射影質問で、結果の関係の組がユニークであることを保証するために、重複する組を除く処理について考える。

重複する組を除く方法として、すべての組をソートする方法とハッシュを用いる方法の二つが考えられるが、ハッシュを用いる方法の方が計算量が少ないと考えられるので、この方法を採用することにする。しかし、異なる組のハッシュ値が衝突を起こすことがあるので、単にハッシュするだけでは重複する組を完全に同定することはできない。そこで、ハッシュを行って関係を分割した後で、各グループ毎に重複する組が含まれていないかどうかチェックする必要がある。このとき、組をソートしてから隣接する組同士で比較を行う方法と、総当りで比較を行う方法がある。実験から、ハッシュによって分割された一つのグループに含まれる組の数が100個程度ならば、総当りで比較を行う方法の方が高速であることがわかったので、ここでは後者の方法を採用した。また、ハッシュ値が衝突を起こした場合には、chainingによって処理した。

4.1.2 結合

結合演算の主なアルゴリズムとして、入れ子操作法、ソートマージ法、ハッシュによる方法がある。このうち、ハッシュによる方法の計算量は、基本的には関係の組の数に対して線形であるので、計算量の観点からは最良の方法と考えられる。そこで、結合演算の場合にも射影演算の場合と同様ハッシュを用いることにする。ハッシュを行って二つの関係をそれぞれ分割した後、同じハッシュ値のグループ毎に結合を行うが、このとき、ソートマージによる方法と、総当りで結合する方法の二つが考えられる。射影演算の場合と同様、一つのグループの組の数が100個程度ならば、総当りで結合する方が高速であることが実験からわかったので、ここでは後者の方法で実行している。

4.2 ベクトル化アルゴリズム

4.2.1 選択

ベクトル計算機はベクトル収集操作の機能を備えており、図

Evaluation of the Vectorized Algorithms for Relational Database Operations

Shintaro MEKI, Ken UMEDA, Yahiko KAMBAYASHI
KYOTO Univ.

1はこの機能をFortranのプログラムで利用する方法を示している。図からわかるように、この操作は選択演算そのものなので、選択演算はベクトル計算機のベクトル収集命令によって、容易にベクトル処理をすることができる。しかも、ベクトル収集操作を行うこのプログラムはスカラ処理を行う場合にも最適なものとなるので、いずれの場合においてもまったく同じようにして組の選択は実行されることになる。しかし、選択した組を入力の関係から出力の関係へ複写する方法が、スカラ処理の場合とベクトル処理の場合とで異なる。つまり、スカラ処理では、関係のデータを行方向にアクセスして、一つの組のデータを一度にまとめて複写するが、ベクトル処理ではベクトル長を長くする必要があるので、関係のデータを列方向にアクセスして複写するようにする。こうすると、ベクトル長が関係の組の数に等しくなって、ベクトル処理の効率が向上する。関係のデータを列方向にアクセスするためには、ブロックの通し番号を示す添字と、ブロック内での組の位置を示す添字の二つを変化させることになるので、アドレス計算が複雑になるが、ベクトル計算機は強力なパイプライン演算器を備えているので、これはあまり問題にはならない。

```
      K=0
      DO 10 I=1,N
         IF (A(I).GT.0) THEN
            K=K+1
            B(K)=A(I)
         ENDIF
10 CONTINUE
```

図1：ベクトル収集操作

4.2.2 射影

スカラ処理の場合と同様、ベクトル処理においてもハッシュによる方法を用いることにする。

[アルゴリズム1：重複する組の除去]

step1 いくつかの属性を消去した関係の組の集合をTとする。またT´=U=U´=φとする。

step2 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値の計算を行い、ハッシュテーブルにすべ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う(並列書き込み)。

step3 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

step4 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、ある一つの属性の値について比較を行う。もし異なっていれば、T´=T´-{t´}、U´=U´∪{t´}とする。この操作をすべての属性について行う。最後まで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。

step5 U´=φならば終了。そうでなければ、T=U´、T´=U´=φとしてstep2へ戻る。

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。

4.2.3 結合

結合演算の場合にも、ベクトル計算機向きに修正したハッシュによる方法を採用することにする。

[アルゴリズム2：結合演算]

step1 関係RとSの組の集合をそれぞれT、Uとする。またT´=U´=φとする。

step2 ハッシュテーブルを初期化してから、Tのすべての要素をハッシュテーブルに登録する。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う(並列書き込み)。

step3 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。ハッシュテーブルに登録されていない要素tが存在する場合、そのことを示す記しを対応するハッシュテーブルのエントリに付け、T´=T´∪{t}とする。

step4 Uのすべての要素uについてこのハッシュテーブルを引いて結合する。そのとき、対応するハッシュテーブルのエントリにstep3で付けた記しがついている場合、U´=U´∪{u}とする。

step5 U´=φならば終了。そうでなければT´=T´、U=U´、T´=U´=φとしてstep2へ戻る。

5．性能評価

4章で示したアルゴリズムを京大大型計算機センターのFACOM VP-2600上に実現し、Wisconsin Benchmark[2]で用いられている組数10万の関係を用いて実験を行った結果を表1に示す。この実験では、選択質問として選択率10%で1万個の組を選択するものを、射影質問としては重複する組を除いた後、4000個の組が残るようなものを、結合質問は組数1万の関係と結合させて、1万個の組ができるようなものを実行させた。また、集約質問としては最小値を求める演算を採用した。

表のS版S実行とV版S実行は、それぞれスカラアルゴリズムとベクトル化アルゴリズムをベクトル計算機のパイプライン演算器を使わずに実行した場合であり、V版V実行は、ベクトル化アルゴリズムをパイプライン演算器を使って実行した場合である。また、加速率はS版S実行の場合のCPU時間をV版V実行の場合のCPU時間で割ったものであり、この値が大きいほどベクトル計算機の性能を十分に引き出しているといえる。表からわかるように、我々の提案するベクトル化アルゴリズムにはおよそ100〜200%のオーバヘッドが発生しているが、それでもなお十数倍以上の加速率を達成しており、ベクトル計算機の性能を十分に引き出していると考えられる。

表1　実行結果

	CPU時間(ms)			加速率
	S版S実行	V版S実行	V版V実行	
選択	107	287	6	17.8
射影	729	2280	52	14.1
結合	290	616	15	19.3
集約	27	27	2	13.5

6．おわりに

ベクトル計算機向きのベクトル化アルゴリズムと、汎用計算機向きのスカラアルゴリズムをベクトル計算機上に実現し、ベクトル化アルゴリズムのより正確な評価を行った。その結果、我々の提案したベクトル化アルゴリズムには、ベクトル化のためのオーバヘッドがあるが、それを考慮に入れてもなお十数倍の加速率が得られることがわかった。

謝辞　種々御議論頂いた上林研究室の皆様に感謝致します。

参考文献

[1] 越智裕之他:ベクトル計算機向き共有二分決定グラフ処理法,情報処理学会第42回全国大会,(1991-3).

[2] David J.DeWitt:The Wisconsin Benchmark:Past,Present, and Future,The Benchmark Handbook,pp.119-165(Morgan Kaufmann,1991).

[3] 目木,上林:ベクトル計算機による主記憶データベースの実現,情報処理学会第44回全国大会,(1992-3).