2L−11

プログラミング言語P3Lにおける永続オブジェクト参照方式とその評価

鈴木 慎司　喜連川 優　高木 幹雄
(東京大学 生産技術研究所)

1 はじめに

オブジェクト指向データベース(OODB)として分類される商用データベースが5つを越えるベンダから供給され、利用者も徐々に増加するにつれ、ベンチマークテストや、実際のアプリケーションの実行結果から、ある種の分野におけるOODBの関係データベース(RDB)に対する優位性が確認されるようになった。

この優位性の原因の一つに、データの仮想空間内における効率的なキャッシングがあげられる。RDBにおいては、I/Oの効率を最適化することが重要視され、データが主記憶に読まれた後、プログラミング言語からアクセスされる際の効率についてあまり考慮されてこなかった。[SMIT 90]

しかし、単にデータの検索をするだけでなく、そのデータベース中のデータに対して繰り返し多くの演算やナビゲーションを行なう応用においてはプログラム言語から主記憶中のデータへのアクセスを最適化することが重要である。逆の見方をすれば、バックエンドにRDBを用いた場合でもデータのキャッシュ方式に変更を加えることができれば、それらのアプリケーションにおける性能向上がはかれる可能性もある。[STON 90]

本稿では、RDBと埋込SQLを用いたシステムと本研究室で実装した永続性を備えたプログラミング言語P3Lのキャシュ方式の違いを述べ、次にP3Lで採用した「先行ポインタ検査方式」をPS-Algol [COCK 84]に見られるポインタ検査方式と比較して得られた評価結果を報告する。

P3Lは、データベースプログラミング言語として使用することを目的とした、並列パーシステントプログラミング言語であり、C言語のスーパーセットとなっている。[SUZU 90b]

2 RDB+埋込SQLにおけるデータ参照方式

この方式においては、データはRDBの管理領域に置かれ、プログラムは必要に応じ欲しいデータをデータベースに要求したり、データの変更を通知する。主記憶上のキャッシュはRDBにより管理される。(図1)

従って一時的には、埋込SQL文内に現われる変数にデータの値がコピーされるが、その値をキャッシュし、次に必要になった時に使うという思想はなく、データが必要になる度に、RDBへの呼びだしとRDBのキャッシュからのコピーが行なわれる。

もともとRDBが対応しようとした領域の応用においては、この方式でも効率上の問題はそれほど生じなかった。なぜなら、処理の大部分は、RDB内部でおこなわれる必要なデータを選択する操作であり、結果をプログラム中に取り込むのに時間は無視できるからである。しかし、CAD/CAMのように同じデータを何度も参照し、その間で演算やナビゲーションをおこなう応用においては、このオーバヘッドが顕著にあらわれてくると思われる。

3 P3Lにおけるデータ参照方式

P3Lを含むいくつかの永続的プログラミング言語(PPL)やOODBでは、キャッシュについて別の考え方をしている。RDB + 埋込SQLとの対比で考えると、P3Lではデータのキャッシュはプログラム言語の変数領域において行なわれる[1]、といえる(図2)。参照されない変数のために領域を割り当てるのは無駄であったり、仮想記憶の制限によって不可能であったりするので、変数が最初に参照された時に主記憶を割り当て、そこへデータベースからデータを移す。その結果、トランザクション中では、一度データベースから取り出されたデータは対応する変数領域にキャッシュされているためにRDBへのリクエストやRDBからのデータコピーの必要がない。

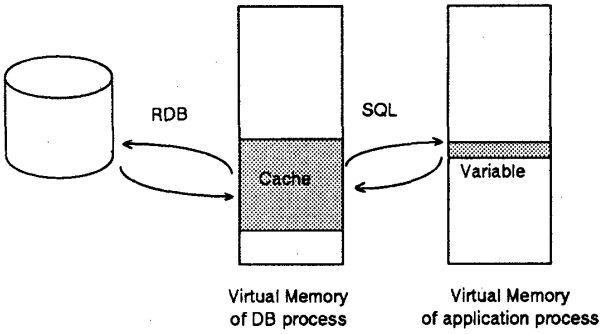

Fig.1 Data access in RDB + Embedded SQL

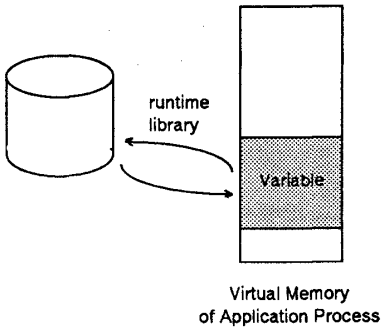

Fig.2 Data access in P3L

そこで、如何にして変数への最初のアクセスを検出するかが問題になる。RDB + 埋込SQLの場合には、この問題(キャッシュ中に参照したいデータが無い場合は、ディスクから取り出す)はRDBによって処理されていた。しかし、P3Lのようなキャッシュ方式を取る場合には、この問題はプログラミング言語の処理系によって対処されなければならない。しかも、データアクセスの効率を犠牲にしないことが重要である。そのための手法をオブジェクト参照方式と呼ぶことにする。[SUZU 90a]

4 複数空間モデル(MSM)とポインタ検査

ここでは、多くの手法のうち、PS-Algolにおいて開発され、P3Lでも基本方式として用いた複数空間モデルについて考察、評価する。

この方式の基本は、主記憶上のオブジェクトを指すものと二次記憶上に置かれているオブジェクトを指すものの2種類のポインタを用意し、その2つを値で区別することである。オブジェクトを参照するために使用したポインタが後者であれば、参照されているオブジェクトを仮想空間中に読み込む。

[0] An evaluation of the method for accessing persistent objects in programming language P3L
Shinji Suzuki, Masaru Kitsuregawa, Mikio Takagi
Institute of Industrial Science,the University of Tokyo

[1] このキャッシュとは別にページバッファが存在する場合もある

同時にポインタの値は書き換えられ、ポインタの種類が変更される。以後、有効な仮想アドレスを指しているという意味で、前者を有効なポインタと呼ぶ。

PS-Algolでは、上記の検査を、ポインタを用いた全ての参照に対して行なうことにより、orthogonal persistence[2]を実現している。

しかし、全てのポインタ参照に際して検査を行なうコストは低くなく、いくつかのPPLにおいては、複数のポインタ型を用意し、永続オブジェクトへのポインタと一時オブジェクトのポインタを言語的に区別することで、一時オブジェクトのアクセスの場合だけでも高速にポインタ参照が可能となることを狙っている。しかし、このアプローチでは、プログラムの記述が困難になるだけではなく、キャッシュされたオブジェクトへのアクセスを高速化しないという点で、本質的な解決になっていない。

そこで、MSMを元に、1) コンパイラによる最適化、2) 先行ポインタ検査方式を適応することにより、どの程度の速度向上が得られるかを評価した。

4.1 冗長検査除去による最適化

MSMの基本思想は、ポインタ参照が行なわれるごとにポインタを検査することであるが、同一のオブジェクトを連続的にアクセスする場合、つまりpを構造体 struct foo { int x, y, z;} へのポインタとして

i=p->x; j=p->y; k=p->z;

というプログラムを実行する場合には、最初のxのアクセスの前に一度検査を行なえば良い。今回の最適化は、関数内の情報のみでpの値が不変であると判断できる部分に対してのみ行なった。また、ループ不変要素移動に似た最適化、つまりループ内で不変のポインタを用いた参照がある場合には、ループの外側でポインタの検査をする、という最適化は行なっていない。

4.2 先行ポインタ検査方式

ポインタの値は、他のポインタの値、および静的データのアドレスを基に生成される。従って、プログラムが実行を開始した時に、全てのポインタ(レジスタ中、スタック、静的エリア上)の値が有効であれば、以後メモリから読み出されるポインタの値が有効である限り、プログラムがオブジェクト参照に使用する全てのポインタは有効となる。

この条件を満たすには、ポインタを使用してメモリ中からポインタを読み出す時に、読み出されたポインタを検査すれば十分である。例を上げると、

struct foo { int x, y, z; struct foo *next; }

という構造体へのポインタpを用いた、

i=p->x; j=p->y; k=p->z; q = p->next;

という参照があった時には、最後についてだけ $p->next$ の値を検査してから、qに代入するという操作を行なえばよい。これによりかなり検査の回数を減らすことができる。

[2]データベースに格納するデータとプログラムで一時的に使用するデータを区別することなくプログラム中で参照・変更できること

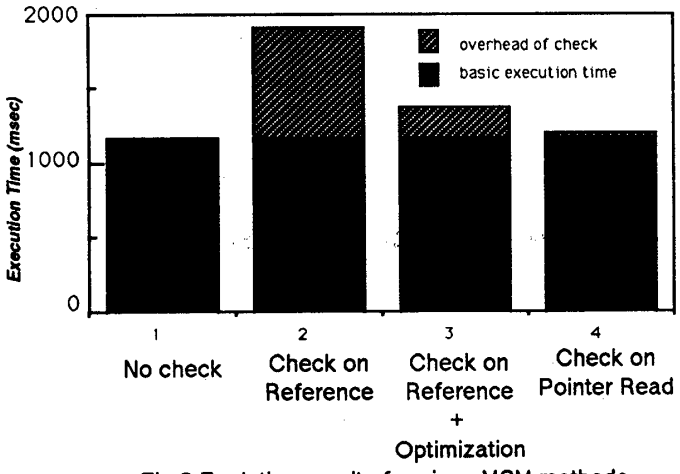

Fig.3 Evalution result of various MSM methods

5 各種MSMのオブジェクト参照方式の評価

評価法としては、Open address hashingを用いて、指定文書内の単語の出現回数を調べるプログラムをC++で作成し、そのプログラムにポインタ検査用のコードを挿入して、実行時間を調べた。ライブラリ関数(strcpy, strcmpなど)も新たに書いて、検査コードを挿入した。ただし、メモリアロケータには手を入れていないので計測範囲でのnewオペレータの使用を回避した。実行時間にはI/O時間は含まない。全単語数は5727個であり、ハッシュテーブルのサイズは2048個とした。

以下の4つのバリエーションについて評価を行なった。結果を図3に示す。

(1) 何の検査もしない
(2) ポインタ使用時の検査
(3) ポインタ使用時の検査 + 共通検査除去
(4) ポインタ読みだし時の検査

6 終りに

今回の評価によって、早期ポインタ検査法の有用性を確認できた。また、簡単な最適化を行なうことによって基本的なMSMの方式でもオーバヘッドが10%程度に押えらえることがわかった。

ここで述べたMSMとは別に、オブジェクト識別子をオブジェクトテーブルへのキーとし、テーブルを介してオブジェクトをアクセスする方式がある。Compacting GCやオブジェクトの書き戻しが容易にできるという利点があるが、オブジェクト参照速度の問題がある。今後、この方式を用いた場合のオーバヘッドを評価してみたい。

参考文献

[SMIT 87] K.E. Smith and S.Zdonik,Intermedia:A Case Study of the Differences Between Relational and Object-Oriented Database systems,OOPSLA 87

[CADF 89] Third-Generation Database System Manifesto, SIGMOD RECORD Vol 19, Number 3

[COCK 84] W.P.Cockshot,M.P.Atkinson and K.J.Chisholm, Persistent Object Management,Software Practice and Experience Vol14,1984

[SUZU 90a] 鈴木, 喜連川, 高木: 永続的空間内の効率的オブジェクトアクセスについて; 情報処理学会平成2年度前期全国大会

[SUZU 90b] 鈴木, 喜連川, 高木: 並列データベースプログラミング言語P3Lの概要; 情報処理学会平成2年度後期全国大会