1G－2

# 永続プログラミングシステムP3Lのoo1-lookupベンチマークを用いた性能評価

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

P3Lは発見時のポインタ書き換え、およびフォールト時におけるOIDの代理OIDへの書換えを採用した永続C/C++プログラミング環境である[1]。MMUを用いてフォールト検査を実施するTexas Persistent Store[2]に対して、P3Lはコンパイラが予約検査用の命令列を挿入する。[3]では、LWS (Load,Work,Save)のアプリケーションモデル[4]におけるWorkフェーズに相当するベンチマークで比較を行ない、永続ストア内のオブジェクト間に参照の局所性が存在しない場合には、予約検査命令が付加されているにも関わらず、P3L処理系がより高い実行性能を示す場合があることを報告した。

MMUによるアクセス保護を利用してページ単位でのポインタ書き換えを行なう方式は、ソフトウェアにより参照単位でポインタを書き換える方式に比べ、Loadフェイズに関して不利であるとの指摘がある。本論文では、P3LのLoadフェーズにおける性能評価および、その指摘の検証をoo1-lookupベンチマーク[5]によって試みる。

## 2 P3L, Texas PSの方式比較

一般に、ポインタ書き換えを、よりeagerに行なうことによってポインタが書換え済みであるかどうかの検査(予約検査)の頻度の減少をはかることが出来る。しかし、eagerな書き換えは、CPUサイクルが不要な書き換えのために浪費される、不必要なOIDと仮想アドレスのマッピングを保持するために記憶領域が圧迫される、という潜在的な問題を抱えている。

P3Lではポインタの発見時書き換えを採用しているが、フォールト時にもOIDから代理OIDへの書き換えを行なっているので、ポインタの書き換えタイミングに関して、フォールト時書き換えに対する優位性はあまり高くない。

しかしながら、フォールト時書き換えを利用する場合にはMMUのアクセス保護の機能を用いることになるので、読み込み・ポインタ書き換えをページ単位で行うのが普通である。この点で、Texas PSはオブジェクト単位での読み込み・書き換えを行なうP3Lに比べ、積極的(aggressive)にポインタの書き換えを行なっている。ただし、この議論は扱うオブジェクトのサイズがページサイズよりも小さいと仮定している。

Texas PSの書き換えに対する積極性は、メモリ管理の方式によっても押し上げられている。1つのページには、同一サイズのブロックしか割り当てられないので、サイズの異なる2つのオブジェクトは必ず別のページに割り当てられる。このことは、ポインタで関係付けられた、サイズの異なる2つのオブジェクトがアクセスされた場合には、必ず2つのページ中の参照が書き換えられることを意味する。Loadフェイズに関するその他の問題点としては、オペレーティングシステムがアクセス違反をアプリケーションに通知するための機構のオーバヘッドがあげられる。

一方、ページというある程度まとまった単位での書き換えを行なうことが可能なため、1参照当たりの書き換えに必要な命令数を少なくできるという利点がある。必然的にフォールティングもページ単位となり、オブジェクトが小さく、クラスタリングが良好であるならば、OIDと仮想メモリアドレスの変換に必要なテーブルのエントリ数を少なくできるし、変換自体もハッシュ法を使って効率的に実行できる。

## 3 ベンチマークパラメータ

今回使用したデータベースは10万個の部品からなり、スワップ領域をもたないローカルディスク上に格納されている。ベンチマークの実行はSparcStation10を利用し、主記憶に60M程度の空き領域のある環境で行なった。Lookupベンチマークでは、1000個の部品番号を生成し、各々の番号で示される部品のx, y座標、および部品の型番号を引数として空の関数を呼び出す処理に要する時間を計測する。

## 4 結果

図1に、データベースが全くキャッシュされていない場合の1個目から100個目までの処理結果を示す。初期のフォールトに際しては、Texas PSがより多くの処理時間を要していることが解る。ただし1個のターゲットをアクセスするためには、ハッシュテーブルの衝突により、複数のオブジェクトが読み込まれることに注意が必要である。

P3Lが1個目の処理に限って多くの時間を要しているのは、16384個のエントリから構成されるハッシュテーブルを読み込み、それぞれのエントリを代理OIDで書き換えているためである。100個目以降の様子をみるため

---

Evaluation of a Persistent Programming System P3L through oo1-lookup Benchmark
S.Suzuki, M.Kitsuregawa, M.Takagi
Institute of Industrial Science, University of Tokyo

に、グラフの全体を図2に示す。連続する5個をまとめて平均をとっている。いずれも処理が進むにつれ、処理時間が短くなっている。P3Lの場合、その主原因はアクセス対象の部品を含むディスクページがキャッシュされていくことであり、一方Texas PSの場合には、部品がアプリケーションの仮想記憶内に取り込まれて行くことが主原因だと推察される。後者の場合にはポインタ書き換えは全く実行されない。これがTexas PSの処理時間のグラフの立ち下がりが早くなっている原因であると予想される。

さらにI/Oの影響をできる限り排除して、ポインタ書き換えに必要なCPUコストを比較するために、データページが既にキャッシュされている状態でベンチマークを実行した。具体的には、1度目の実行の直後に2度目の実行を行なった。その結果が図3である。最初から、必要なデータページは全てキャッシュされているのでP3Lの場合には、処理時間の減少傾向は見られない。(1000回を越え、さらに継続すれば、ゆっくりながら減少はする。)一方、Texas PSの場合には、アクセス保護違反の発生回数、書き換えターゲットへのOIDの割り当てが必要となる回数が、積極的な書き換えによって急減するために、処理時間の減少傾向が観察される。

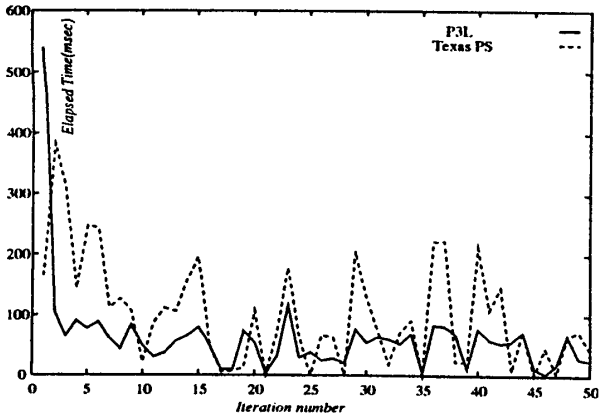


図 1: oo1-Lookup (1st-50th), clean cache

## 5 結論

図3に見られるように、P3Lでは参照単位のポインタ書き換えを、ソフトウェアで実現しているためにLoadフェーズでの処理時間を低減できることが確認できた。しかし、Load対象のオブジェクトの割合がデータベース中の全オブジェクト数に対してある程度大きい場合には、単一参照の書き換えに必要な平均命令パス長が長くなるという問題がある。今後は、ソフトウェアによる発見時書き換えにページ毎のリロケーション、という実装の評価を行なってみたい。

また、図2, 図3より、アクセス保護例外を補足するオーバヘッドは比較的大きい可能性があるものの、I/Oのコストに比べれば無視できる程度のものであることも確認できた。

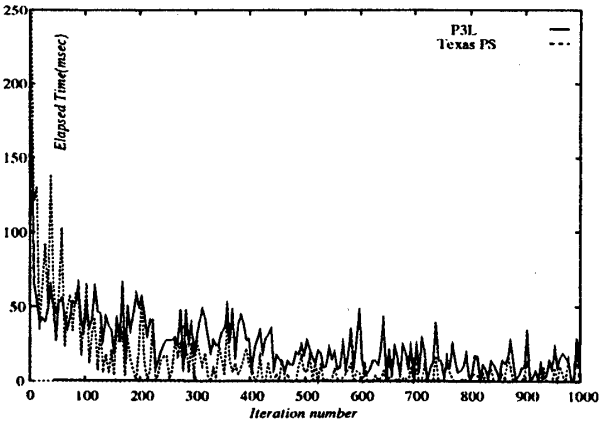


図 2: oo1-Lookup (1st-1000th), clean cache

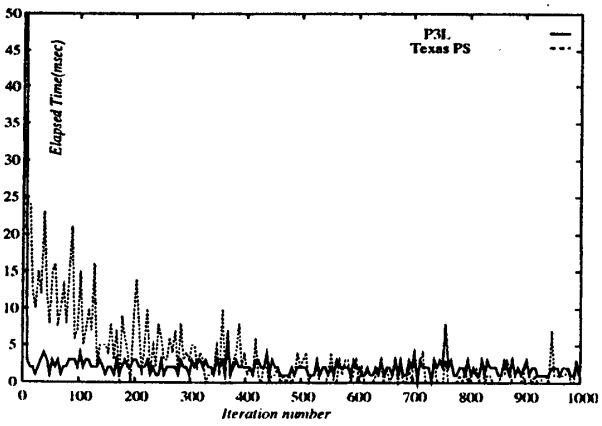


図 3: oo1-Lookup (1st-1000th), warm cache

## 参考文献

[1] S. Suzuki, et al. "An Efficient Pointer Swizzling Method for Navigation Intensive Applications", to appear in proc. of Sixth Intl. Workshop on Persistent Object Systems, Trascon France 1994

[2] Paul R.Wilson,'Pointer Swizzling at Page Fault Time: Efficiently Supporting Huge Address Space on Standard Hardware', ACM Computer Architecture News, Vol 19, No.4 June 1991

[3] 鈴木 喜連川 高木,' 永続プログラミングシステム P3L の OO1 ベンチマークを用いた性能評価' 情報処理学会 第50回, 1994

[4] J. Eliot B. Moss, 'Working with Persistent Objects: To Swizzle or Not ot Swizzle', Trans. on Software Engineering Vol 18, Aug 1992

[5] R.G.G. Cattel and J. Skeen, 'Engineering Database Benchmark', Database Engineering Group, Sun Micro Systems Technical Report 1990