サロゲートOIDを用いたポインタ書き換え方式

4G－4

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

永続プログラミング言語、オブジェクト指向データベースなどの実装では、二次記憶上における永続参照（永続ポインタ）の表現形式(OID)は、仮想メモリポインタと異なる形式を使用することが多く、主記憶内でポインタ書き換え(Pointer Swizzling)という技術が多利用される。

主記憶中の参照を全て仮想アドレスポインタに書き換える方式は、仮想空間を浪費する可能性がある。一方、永続参照をそのまま主記憶中で使用することは、以下で述べるように実行速度の低下につながる。

本稿では、サロゲートOIDと呼ぶ主記憶中での永続参照の表現形式を用いたポインタ書き換え方式について解説し、サロゲートOIDの使用がもたらす性能向上についての計測結果を報告する。

2　ポインタ書き換え

最初に、以下の3点

1. ポインタ書き換えの試み(のタイミング)
2. ポインタ書き換えの完了の保証(のタイミング)
3. 未書き換えのポインタの主記憶中での表現

に基づいてページフォールト時書き換え方式[1]を見てみる。この方式では、主記憶上での永続参照は全て仮想アドレスポインタによって表現される。プロセスの起動後、プログラムは、最初にアクセスされるオブジェクト(以下root)へのポインタを、例えばオブジェクト名を鍵としてシステムから取得する。その際システムは、仮想アドレス空間中に該当オブジェクトを含むページ（オブジェクトがページサイズよりも大きい場合には複数ページ）を格納するための空間を予約し、その予約空間内の該当オブジェクトの位置を示すポインタを返す。この予約領域には、MMUの機能を利用してアクセス保護がかけられる。

rootの内容が最初にアクセスされた時には、アクセス違反の例外が検出される。システムが用意する例外ハンドラは、該当ページを2次記憶から予約空間内へ読み込み、次にそのページ内に存在するポインタが指し示すオブジェクトを収容するための空間を確保し、同様にアクセス保護をかける。この様子を図1に示す。

ページフォールト時書き換え方式おいては、(1),(2)ともにページフォールト時に行なわれ、未書換えのOIDは存在しない。

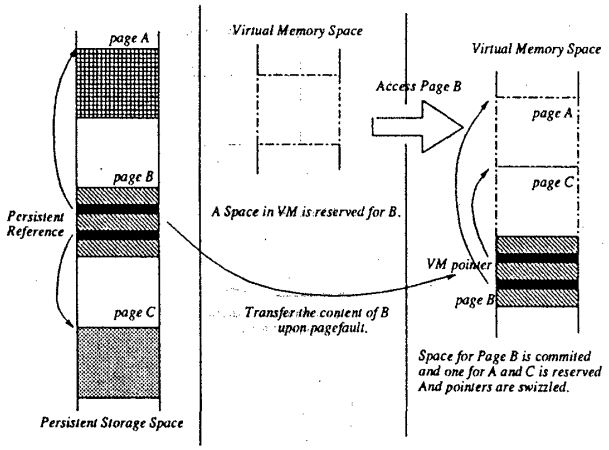

図1: Pointer Swizzling at Page Fault Time

次にExodus/Eの実装におけるポインタ書き換え方式(**発見時書き換え**)を見ると、オブジェクトが読み込まれた場合には、その中に含まれる永続参照(96bit)はそのまま主記憶へコピーされる。

ポインタへのdereferenceの際には、「ポインタが既に仮想アドレスに書き換えられているか否かを検査し、書き換え済みならば該当オブジェクトを主記憶中にそのままアクセスし、未書換えならば該当オブジェクトを読み込み、永続参照を仮想アドレスポインタ(+書き換え済みであることを示すフラグ)に書き換える」という処理が実行される。また、ポインタを主記憶中から読みだす際には、対象オブジェクトが既に読み込まれている場合に限り、ポインタを書き換える。これはポインタを早めに書き換えることで、書き換え回数の総数を減らすことを目的としている。

発見時書き換えにおいては、(1)はポインタの読みだし(発見時)、(2)はポインタのdereference時であり、未書き換えの永続参照のポインタの表現は2次記憶上と同様、つまり仮想メモリポインタのサイズよりも大きい。

---

[0] A Pointer Swizzling Method with surrogate OID
S.Suzuki, M.Kitsuregawa, M.Takagi
Institute of Industrial Science, University of Tokyo

3 サロゲート OID

P3L[1]において採用した書き換え方式は、全節で紹介した2つの方式の中間的なものとなっている。

オブジェクトを読み込む際には、オブジェクト中のポインタを特殊な値 ((void *)-1) で置き換えるとともに、置き換えを行なったポインタのアドレスと、その OID の対応を Addr2Oid テーブル内に記憶する。これによって、主記憶中での永続参照を仮想ポインタと同様のサイズで表現することが可能となる。マーキングのために使用するこの特殊な値を、サロゲート OID と呼んでいる。ただし、この際にオブジェクト内の参照が指し示すオブジェクトが既に読み込み済みの場合には、サロゲート OID でなく、仮想記憶アドレスで置き換える。

仮想アドレスでの書き換え完了の保証が行なわれるのは、ポインタが主記憶中から読み出された段階である。この時のポインタがサロゲート OID ならば、Addr2Oid テーブルを使って対象オブジェクトを調べ、読み込みを行なう。引続き、発見時書き換えと異り、サロゲート OID が指し示すオブジェクトの主記憶中の有無に関わらず書き換えを行なう。

サロゲート OID は、発見時書き換えにも適用が可能である。その場合には、サロゲート OID のコピー時に Addr2Oid テーブルに新しいエントリを追加するか、値によって区別可能な複数のサロゲート OID を使用する必要がある。

4 サロゲート OID 使用の利点

ページフォールト時書き換えと比較した場合には、仮想記憶空間の消費を抑えることができるという利点がある。しかしながら、その利点は、ポインタの値がサロゲート OID であるか、仮想アドレスポインタであるかという検査が必要になるという欠点と対になっていることに注意する必要がある。

一方、サロゲート OID を使用しない発見時書き換えと比べた場合には、性能上の利点以外に、

- 既存のコンパイラに加える変更が少なくてすむ。
- 発見時に書き換え済みの保証を行なう場合、標準ランタイムライブラリの多くの関数が再コンパイルなしで使用可能である。

という利点がある。ポインタのサイズが異る場合にはポインタを引数に取る関数に関しては、既存のバイナリ形式のライブラリは明らかに利用不可能である。P3L の方式を使用すれば、vprintf, writev, readv などの内部でポインタを生成するようなポインタ型引数の dereference を行なう関数以外は呼びだし可能である。

[1] 当研究室で実装した永続 C 言語 [2]

5 OO1 ベンチマーク

仮想アドレスポインタを使用した場合と、サイズの大きなポインタを使用した場合の、OO1 ベンチマーク [3] の3つの処理のうち、前方探索における実行速度を計測した。Part オブジェクトは全て主記憶中に置き、I/O は一切行なわない。

プログラミングには C++ を用い、g++2.5.7 でコンパイルを行ない、SparcStation /10 上で実行した。サイズの大きなポインタはコピーコンストラクタ、dereference 演算子 (− >,*)、添字演算子 ([]) などを定義した Smart Pointer クラス (class Ref) を記述し、ポインタの動作を模擬することで実現した。class Ref のメンバ関数は出来る限りインライン展開したが、型の依存性のため一部不可能だった。そこで class Ref を使用するオーバヘッド自体を把握するために、仮想アドレスポインタと同一サイズ (4 Bytes) の Ref クラスを用いた場合 (4B-Ref) の実行時間も合わせて計測した。結果を以下に示す。

	Reg. PTR	4B-Ref	8B-Ref	12B-Ref
(msec)	1242	1529	1779	2052
(ratio)	1.0	1.23	1.43	1.65

この結果から、96bit の永続参照を主記憶中で使用した場合には、8バイトのネイティブポインタを使用した場合で 42%(smart pointer を利用した場合には 34%) 程度のオーバヘッドが生じることがわかる。ただし、永続オブジェクトを扱う場合には、ポインタの書き換え完了の検査が不可避であるが、今回の実験ではそのための検査のコードを挿入していないので、相対的なオーバヘッドは実際にはこれよりも小さな値となる。

6 結論

サロゲート OID の実行性能向上への寄与の程度を OO1 ベンチマークを用いて計測し、30%-40%(通常のコンパイラで翻訳したプログラム比) 程度の効率向上を可能とすることを確認した。

参考文献

[1] 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

[2] 鈴木 喜連川 高木,' 永続的プログラミング言語 P3L 処理系の GCC と Exodus Storage Manager による実装' 情報処理学会 第 48 回, 1993

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