5P−9

永続的プログラミング言語P3Lの実装におけるポインタ書き換え方式のコスト評価

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

永続的プログラミング言語の実装技術の一つにポインタ書き換え(Pointer Swizzling)がある。ポインタ書き換えを行なった場合には、仮想空間内のUID(広域的なオブジェクトID)を仮想アドレスに書き換えることで、UIDと仮想アドレスの対応表を検索する回数を削減でき、効率的なオブジェクトアクセスが可能になる。しかしながら、その場合でも、ポインタを使ってオブジェクトをアクセスするためにはポインタの書換えが既に行なわれているか否かを検査(Residency Check、ポインタ検査)する必要がある。本論文では、P3Lにおいて採用されたポインタ書き換え方式を用いた場合のポインタ検査のためのオーバヘッドについて評価検討する。

2 各種ポインタ書き換え方式

まず、P3Lにおいて採用したポインタ書き換え方式を他の2つの方式と比較して説明する。

2.1 (a)Swizzling at Pointer Dereference

最初に図1の(a)に示す方式について説明する。この方式はもっとも直観的な方法で、ポインタがオブジェクトアクセスに使われた場合(ポインタがdereferenceされた場合)にポインタ検査を行なう。PS-Algolの実装において採用された。この方式の欠点は、ポインタを介したオブジェクトアクセスの頻度は一般に非常に高いので、検査頻度も同様に高くなってしまうことである。コンパイラの共通式除去と同様のオプティマイズ(検査済みであることが静的に検証できる場合には検査を省略する)を行なうことで頻度の低下をはかることは出来るが、広域的なオプティマイズは困難なため、大幅な効果は期待できないと思われる。この方式のラフな性能評価については、[1]を参照願いたい。その結果によると、この方式はオプティマイズを行なっても次に述べる方式よりも低いオーバヘッドを実現できない。また[2]でも指摘されているように、この方式には実際に書き換えを行なう前にUID形式ポインタのコピーが作られてしまうという問題もある。UID形式のコピーが作られることは、実際の書き換え操作回数の増加につながる。[?]ではこの方式をSwizzling upon Derefernceと呼んでいる。

2.2 (b)Swizzling at Pointer Fetch

P3Lではこの方式を用いている。(a)の方式を取った場合に検査対象となるポインタは、必ずメモリから読み出されたものである。従って、ポインタが使われる時ではなく、読み出される段階で早めに検査してしまおうというのが、この方式の発想である。もっとも、全てのポインタの読みだしを検査対象とする必要はなく、ヒープ内からのポインタ読みだしのみを検査する。しかし、静的にヒープからの読みだしを検出することは不可能なので、実際にはポインタを使ったポインタ読みだしを検査対象としている。一般にポインタの読みだしの回数は、そのポインタの使用回数よりもずっと少ないので、この方式を使用することでResidency Checkの回数を(a)に比べ大きく削減できる。書き換えをスタック上のポインタに限定するものの、同様の方式を[?]でも採用しており、Swizzling at Discovery(ポインタ発見時の書き換え)と呼んでいる。

2.3 (c)Swizzling at Page Fault Time

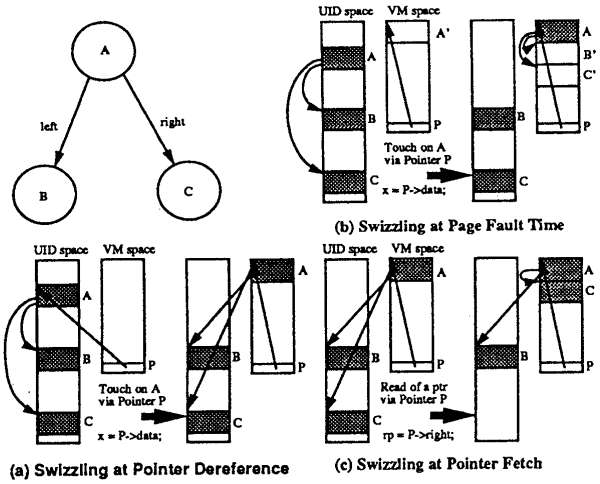
この方式は[?]において提唱された方式で、MMUによるメモリ保護を利用した方式である。原理は以下のとうり。プログラム起動時には、図1(b)に示すように、rootポインタが指すページを仮想記憶内に割り当て(実メモリは割り付けない)、該当ページをアクセスプロテクトしておく。プログラムがrootポインタのdereferenceを行なった時には、MMUが生成する例外でハンドラが非同期に起動される。このハンドラは該当ページに実メモリを割り当てるとともに、対応するページをディスク上から、割り当てたメモリに読み込む。その際には上記と同様に、ページ内のポインタが指すオブジェクトのための領域を仮想記憶内に割り当て、該当ページをアクセスプロテクトする。そして、ページ内の全てのポインタを割り当てた仮想記憶のアドレスで書き換える。従ってアプリケーションを実行しているプログラムには、UIDは一切見えず、Residency CheckはMMによっておこなわれるため検査命令の挿入によるオーバヘッドが生じない。一方欠点としては、OS, ハードウェア依存性があり移植性が低いこと、仮想アドレスを多く消費するのでPage Tableのためにメモリ資源が余分に使われること、TLBミスの増加などが上げられる。

3 P3Lにおける書き換え方式のコスト評価

3.1 評価方式

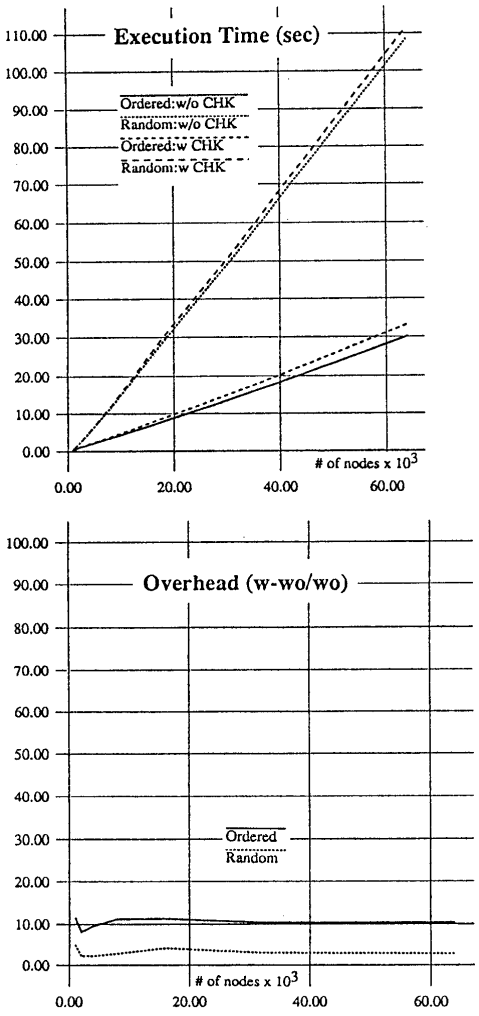
今回の評価対象はAVL木のノード追加プログラムとした。[1]において、オープンハッシュを用いた単語の出現回数をカウントするプログラムを利用したのに比べ、今回のテストプログラムはポインタナビゲーション主体のアプリケーションとなっている。比較の方式としては、Residency Checkの全く入らない通常のプログラム([4]に掲載されているものを使用した。)と、P3Lトランスレータを通してResidencyCheckを付加したものを、ともにC++コンパイラで実行形式を生成し、それぞれの実行時間を計測した。

[0] Cost Evaluation of the Pointer Swizzling Method in the implementation of a Persistent Programming Language P3L
S.Suzuki, M.Kitsuregawa, M.Takagi



← 図1 各種ポインタ書き換え方式

↓図2 計測結果



コンパイルには g++1.37 を利用し、Sequent S81 (i386 16MHz, 64KB cache) 上でプログラムを実行した。追加するノード数を 1000 個から 64000 個まで 2 倍にしながら、計測を 6 回行なった。1 回の計測について、

- (1) ポインタ検査なしで、要素の大きさの順に追加を行なう場合 (Ordered:w/o CHK)
- ポインタ検査をしながら、要素の大きさの順に追加を行なう場合 (Ordered:w CHK)
- ポインタ検査なしで、要素をランダムな順で追加を行なう場合 (Random:w/o CHK)
- ポインタ検査をしながら、要素をランダムな順で追加を行なう場合 (Random:w CHK)

の 4 つの場合を調べている。

3.2 結果

計測結果を 図 2 に示す。上のグラフはプログラムの実行時間を示している。要素の大きさ順に挿入を行なった場合の方が、実行時間が大幅に小さくなっている。これは、ツリーのトラバースが常にもっとも左の枝へと行なわれるためにキャッシュの効果が出ていると考えられる。また、木の平均の高さが低くなっていることも影響している。

重要なのは、図 2 の下のグラフである。このグラフはポインタ検査をした場合に生じるオーバヘッド「検査付きの実行時間 - 検査なしの実行時間) ／検査なしの実行時間」を図示したものである。大きさ順の追加の場合には、全体の実行時間が少ないだけにポインタ検査のオーバヘッドが大きめに出ている。

4 終りに

今回のプログラムが、ポインタ・ナビゲーションが主体のプログラムであることを考慮すると、ランダムに挿入した場合の 3% という数値は満足のいくものである。実際に B-Tree のナビゲーションを行なう場合には、ひとつのノード内に複数のレコードが格納されるので、これよりもよい結果となろう。しかしながら、大きさ順の挿入をした場合の 10% という数値は思っていたよりも大きな数値であった。これには、命令数の増加以上に、条件ジャンプによるプリフェッチキューのフラッシュが大きく効いていると思われる。今後は、P3L コンパイラを変更し、命令の再配置や SPARC の Tagged Pointer 演算時の例外生成機能を利用することで、10% を 5% 程度にしたい。

参考文献

[1] 鈴木, 喜連川, 高木 ' 永続的プログラミング言語におけるオブジェクト識別子の主記憶内表現について ', 情報処理学会データベース研究会 83-5, 1991

[2] A Seth J. White and David J. DeWitt, 'A Performance Study of Alternative Object Faulting and Pointer Swizzling Strategies', Technical Report University of Wisconsin, June 1992

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

[4] Aaron M.Tenenbaum, Yedidyah Langsam and Moshe J. Augenstein, 'DATA STRUCTURES USING C' PRENTICE HALL ISBN 0-13-199746-7