4D−4

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

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

本論文ではC言語を基にした永続的プログラミング言語P3Lの実装について述べる。Gnu C Compiler[1]に永続オブジェクトのインクリメンタルなロードのためのコード生成機能を付加したものをコンパイラとし、オブジェクトサーバとしてWisconsin大学のExodusプロジェクトで開発されたExodus Storage Managerを使用している。以前の実装[2]では、コンパイラがC言語を出力するトランスレータとして実装されていたため速度および安定性が十分でなかったし、また商用のサーバを用いたためプログラミングツールとしての開放性が制限されていた。今回の実装では以上のような問題点が解消されている。

2 P3Lコンパイラ

次の二つの拡張がGCCに対して加えられている。

1. オブジェクトの内容をアクセスする命令列の前に、そのオブジェクトがすでに読み込まれているか否かを判断し、必要に応じ読み込みを行うための命令列を挿入する機能。以下この命令列を存在検査コードと呼ぶ。
2. オブジェクトの読み込みの際に必要となる関数を、プログラム中で定義された型ごとに生成する機能。生成された関数は存在検査コードから呼び出される。以下この関数をswizzle関数と呼ぶ。

2.1 存在検査

コンパイラは構文解析時にボトムアップにAST(抽象構文木)を生成してゆく。AST中のノードを組み立てる際に、オペレータがoperator-'*'(dereference)で、かつ、そのノードの型がポインタ型であるか、ポインタを含む構造体である場合にそのノードに印をつける。オペレータがoperator-'.'(member extraction)またはoperator-'[]'(array indexing)であり、ノードの型がうえの条件に合致するときには、aggregateオペランドを指定する枝から印を引き上げる。この処理の例を、図1に示す。

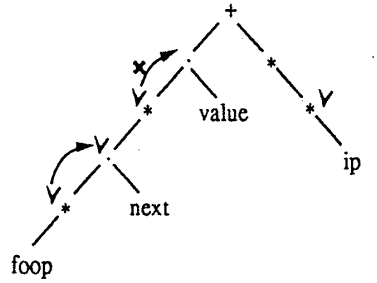

図1: ASTのマーキング

```
struct foo {
     struct foo * next;
     int value
} * foop ;
struct int **ip;
```

という宣言のもとに、
式foop->next->value+**ip
に対する構文木が作られている。矢印の一つに×がついているのは、引き上げを行なうノードが整数型であるため印が消去されるからである。

コンパイラは構文木の生成後にRTL(Register Transfer Language)で表現されるコードを出力する。RTLは構文木よりも低レベルの言語独立な表現形式であり各種の最適化がRTLコードを対象に行なわれたあと、最終的に機械命令へ変換される。P3Lでは、ASTからRTLへの変換に際し、上で述べた印のついたノードを変換をする直前に、そのノードのオペランドのアドレスに対して存在検査コードを挿入する。この命令列は、オペランド(ポインタ)の内容を検査し、もしもポインタが主記憶中のアドレスを指していなければ、オペランドのアドレスを引数としてswizzle関数を呼び出すという動作をする。

このようにして生成されたコードの実行時の振舞いの例を図2に示す。左上の図は連結木のヘッダがデータベースから仮想記憶上に読み込まれた状態である。アプリケーションプログラムは先頭のオブジェクト(A)をアクセスするために、ヘッダ中のポインタ領域を読み出そうとする。この読み出しは、上で述べた条件に合致するので、実際のアクセスが起こる前に、このポインタに対し存在検査が実行される。この結果オブジェクトAの型に対応するswizzle関数が呼び出され、オブジェクトAの内容が

[0] Implementation of a persistent programming language P3L based on GCC and Exodus Storage Manager
S.Suzuki, M.Kitsuregawa, M.Takagi
Institute of Industrial Science, University of Tokyo

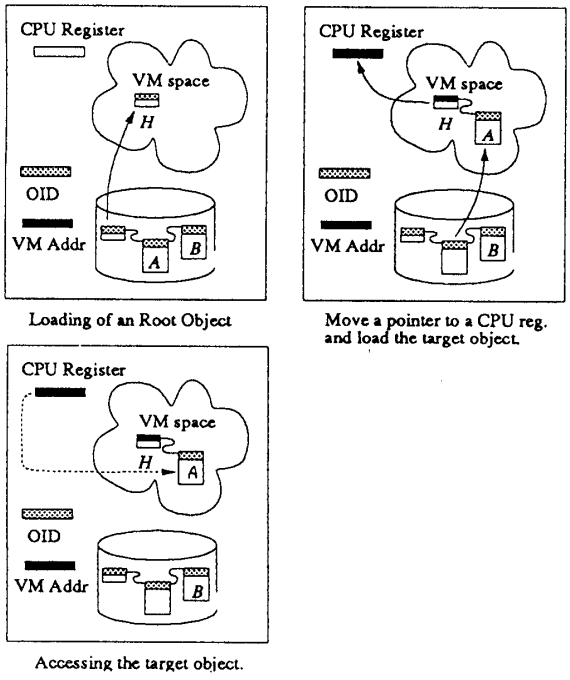

図 2: An Example of Object Faulting

データベースから読み出されるとともに、内部のポインタ型メンバの調整が行なわれる。最後に、存在検査の対象となったポインタはオブジェクトの読み込まれたアドレスで書き換えられる。以上の状況が右上の図に示されている。その後左下の図にあるようにオブジェクトAの内容が参照される。これがオブジェクトBへの参照であれば、その結果Bの読み込みが行なわれる。

2.2 Swizzle 関数

すでに述べたように、swizzle 関数は引数として渡されたアドレスに存在するポインタが指し示すオブジェクトの読み込みを行なって、そのポインタを読み込みアドレスで書き換えるとともに、読み込んだオブジェクト内のポインタ型のメンバを調節する。この調節が必要になるのは、Exodus Stroage Manager 中でのオブジェクト識別子のサイズが 96bit あり、主記憶中におかれたオブジェクトの中に埋め込むことが不可能なためである。そこで swizzle 関数はまず、読み込まれたオブジェクト中の OID を仮想アドレスに変換することを試みる。具体的には swizzle 関数は P3L のランタイムが管理している OID と仮想アドレスの対応表 (OID2ADDR) を検索する。対応するエントリが見つかれば、エントリ中の仮想アドレスを該当ポインタに設定する。検索に失敗すれば該当 OID で示されるオブジェクトが読み込み済みでないことを記録するために、ポインタに (void *)-1 を設定する。そして、別のテーブル (MARKER2OID) に、-1 を設定したポインタのアドレスとそのポインタに対応づけられた OID の関連を記録する。swizzle 関数は、このテーブルを検索して読み込むべきオブジェクトの OID を知る。

3 Exodus Storage Manager(ESM)

ESM は Wisconsin 大学の Exodus project[3] で開発されたオブジェクトサーバであり、同大学で開発された WiSS (Wisconsin Storage System) を基礎にしている。このサーバは 3 種類ほどのプラットフォームをサポートしている。使用中の構成を図?? にしめす。ESM のコードにはクライアントライブラリを含め、一切変更を加えていない。P3L が現在利用している ESM の機能は以下のとうり。

1. OID(Object) の生成
2. Object 単位の Read/Write
3. トランザクション制御

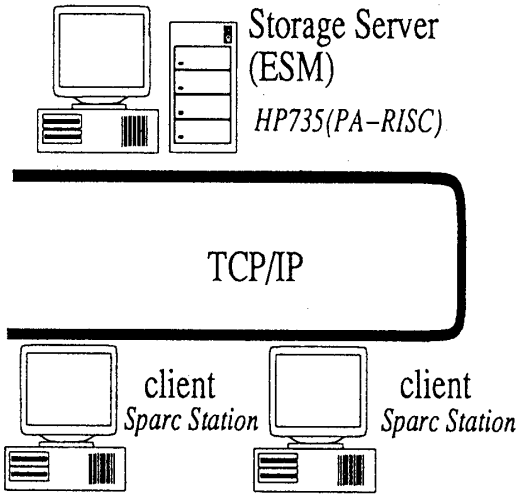

図 3: 使用中の構成

4 今後の課題

ESM とは異なるサーバの使用を可能とするため、生成する swizzle 関数の仕様を外部から指定できるようにコンパイラの変更を進めている。

参考文献

[1] 'Using and Porting GNU CC', in the distribution of Gnu C Compiler, Free Software Foundation
[2] 鈴木 喜連川 高木,' 永続的プログラミング言語 P 3 L の実装方式', 「北の国から」データベースワークショップ 92 年 7 月
[3] Carey, M., et. al.,'The EXODUS Extensible DBMS Project: A Overview' in Readings in Object Oriented Databases., S. Zdonik and D.Maier,eds.,Morgan-Kaufman,1989