静的解析によるメソッド探索の高速化

3N－7

中村 宏明　小野寺 民也　竹内 幹雄
日本アイ・ビー・エム (株) 東京基礎研究所
{nakamura,onodera,mtake}@trl.ibm.co.jp

1　はじめに

効率のよいオブジェクト指向言語の実装のためには、メソッド探索を高速化することが重要である。このために種々のメソッド・キャッシュ法が提案されてきた。ルックアップ・キャッシュ [1] やインライン・キャッシュ [2] は市販の Smalltalk 処理系などで実際に使われている。キャッシュ法が改善されるにしたがって、キャッシュがミスしたときのオーバーヘッドがメソッド探索の性能に与える影響が大きくなってきている。したがってキャッシュがミスした場合のメソッド探索も高速化しなければシステム全体の性能は向上しない。

プログラムの静的解析によってメソッド探索を高速化する手法としてタイプ推論を用いる方法やディスパッチ表を用いる方法が提案されている。タイプ推論は時間がかかることが問題で、高速化の追求は行なわれているものの（例えば [3]）、コンパイルの１フェーズとなるほどに実用的な時間では実行できない。

ディスパッチ表は可能なクラスとセレクタの組み合わせに対して予めメソッドを計算して表に保存しておくものであるが、単純な実装では表が巨大になる。現在では表の充填率を 100%近くに圧縮する方法が開発されているが [4]、Smalltalk などでは実用化されるほどには小さくならない。また２つの値を元にして表を引くことと、圧縮した場合にはセレクタ正当性のチェックが必要になることから、ある程度以上の高速化は望めない。

本稿ではプログラムの静的解析によるメソッド探索を高速でかつ実用的なものにする手法について考察する。

2　二分決定木によるメソッド探索

図 1の構造をもつプログラムを考える。メソッド m がクラス A と クラス D で定義されている。すべてのクラスで m が受信可能なので、ディスパッチ表では m のエントリはすべてのクラスに対応して用意される。ここで多くのクラスが継承によって一つのメソッドを共有していることに着目すると、新たなメソッドの探索方法が考えられる。

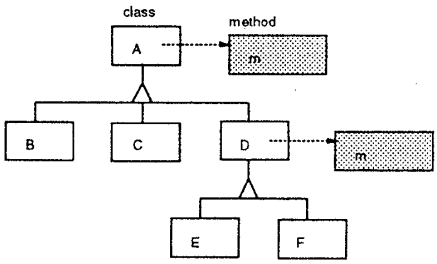

図 1: クラス階層とメソッド定義の例

```
if (D のサブクラス) {
    D::m を実行
} else {
    if (A のサブクラス) {
        A::m を実行
    } else {
        エラー処理
    }
}
```

図 2: 二分決定コード

図 1のプログラムで起動されるメソッドは、D のサブクラスを除いて A のサブクラスでは A の m、D のサブクラスでは D の m であるから、図 2のような二分決定コードをコンパイル時に生成し、プログラム実行時に m の呼び出しで図 2のコードによってメソッドの探索を行なうようにすればよい。この方法では圧縮が自然に行なわれていて、またセレクタ正当性のチェックがあらかじめ埋め込まれている。

このようなコードが高速に動作するためには、条件検査が軽く、条件検査の回数が少なくなることが重要である。我々は次のような特徴をもつ二分決定木生成アルゴリズムを既に開発している。

- クラス階層の解析によって各クラスに固有の番号を割り当て、番号の大小比較によってサブクラスのチェックを行なう
- バランスした二分木にコードを展開する。

3　ディスパッチ表との組合せ

Optimization of Method-Dispatch by Static Analysis
Hiroaki Nakamura, Tamiya Onodera, and Mikio Takeuchi
IBM Research, Tokyo Research Laboratory

メソッド定義数	セレクタ数	セレクタ数累計
1	7513 (74.9%)	7513 (74.9%)
2	1371 (13.7%)	8884 (88.6%)
3	482 (4.8%)	9366 (93.4%)
4	212 (2.1%)	9578 (95.5%)
5	118 (1.2%)	9696 (96.7%)
..	...	...
243	1 (0.0%)	10025 (100%)

表 1: メソッド定義数の分布 (VisualSmalltalk 3.0)

	大きさ
ディスパッチ表 (充填率 100%)	3.45 MB
二分決定木 (IBM POWER のコード)	5.51 MB
参考：仮想イメージ	3.81 MB

表 2: メソッド探索器の大きさ (VisualSmalltalk 3.0)

二分決定木による方法では同一セレクタに対するメソッド定義数が多くなると処理時間が増大する。しかし表1に示すように ParcPlace-Digitalk 社の VisualSmalltalk 3.0 を例にとるとメソッド定義数が 1 のセレクタが全セレクタの 75% であり、たいていのセレクタに対するメソッド定義は数個であるといえる。このため二分決定木による方法は多くの場合に有効に動作するといえる。

一方でメソッド定義数が大きいセレクタも存在し、これらを二分決定木でメソッド探索すると多段の条件検査を通過することになって時間がかかる。このようなメソッド定義数が大きいセレクタに対してはディスパッチ表で扱うようにすれば、性能の低下を押えることができる。つまりメソッド定義数の大小によって二分決定木とディスパッチ表の性能の優劣が逆転するので、メソッド定義数に応じて両者を使いわける。

文献 [5] では、まずディスパッチ表で探索し、そこで衝突するメソッドをクラス検査によってさらに分類する方法を提案しているが、単純なディスパッチ表より遅くなる。我々の方法では、ディスパッチ表を使うか二分決定木を使うかがセレクタごとに決定しており、どちらか単独のものより探索速度が向上する。

4 メソッド探索器の遅延生成

メソッド探索器の大きさの概略を調べるために、VisualSmalltalk 3.0 に対応したディスパッチ表と二分決定木を C のプログラムとして生成し、それを最適化オプションをつけて IBM RS/6000 の POWER マシン・コードにコンパイルし、その大きさを測定した。その結果どちらの手法でも、単純な実装では表2のように仮想イメージを膨張させてしまうことが分かった。二分決定木ではセレクタごとに異なる探索コードができるためコード量が大きくなる。

アプリケーション	使用セレクタ数
ブラウザ	781 (7.7%)
コンパイラ	354 (3.5%)
Smopstone ベンチマーク	172 (1.7%)

表 3: 使用セレクタ数 (VisualSmalltalk 3.0)

このような欠点を取り除くためには、メソッド探索器の生成を必要になるまで遅らせればよい。表3に示すように大抵のアプリケーション・プログラムの実行に使われるセレクタには非常に偏りがあるので、実行時に必要な部分だけを生成することによって記憶量を大幅に削減することができる。我々はディスパッチ表と二分決定木の両方について、遅延生成によって記憶量を削減する方法について検討を行なっている。

5 おわりに

我々はオブジェクト指向言語の高性能な実装方式の研究を目的として、実験的な Smalltalk 処理系を作成している。現在は二分決定木とディスパッチ表と組み合わせるときのセレクタ当りのメソッド数のしきい値の検討を行なっている。

Smalltalk を対象として実験を行なっているが、これは Smalltalk ではスケーラビリティの検証のための大規模なプログラムが手に入りやすいことが理由である。本研究の手法は他のオブジェクト指向言語にも適用できる。

参考文献

[1] David Unger and David Patterson : Berkeley Smalltalk - Who knows where the time goes, in Smalltalk-80 Bits of History, Words of Advice, Addison-Wesley, 1983.

[2] David Unger : The Design and Evaluation of a High Performance Smalltalk System, MIT Press, 1987.

[3] Ole Agesen : The Cartesian Product Algorithm - Simple and Precise Type Inference of Parametric Polymorphism, in ECOOP'95, 1995.

[4] Karel Driesen and Urs Hölzle : Minimizing Row Displacement Dispatch Table, in OOPSLA'95, 1995.

[5] Jan Vitek and R. Nigel Horspool: Taming Message Passing - Efficient Method Look-Up for Dynamically Typed Languages, ECOOP'94, 1994.