2M−4

A'UM-90のボラタイルオブジェクトの実装方式

小西 弘一 丸山 勉 小長谷 明彦 吉田 かおる† 近山 隆†
日本電気(株) C&Cシステム研究所 †(財) 新世代コンピュータ技術開発機構

1 はじめに

本論文は、オブジェクト指向モデルの枠内で条件分岐を扱うために並列オブジェクト指向言語 A'UM[吉田 90] に導入されたボラタイルオブジェクトの実装方式の最適化手法を示す。

ボラタイルオブジェクトは、これを生成する親オブジェクトとは並列に動作する独立したオブジェクトである。しかし、このモデルにしたがって、実行時に実際にボラタイルオブジェクトを生成して並列に動作させると、処理コストが大きくなり、条件分岐のように頻繁に行われる処理には適さない。一方、ボラタイルオブジェクトを親オブジェクトの処理の一部として実行すると、処理コストを押えることはできるが、並列に実行した場合には起こり得ないデッドロックが生じる可能性がある。

A'UM-90 処理系 [小西 90] では、前記のデッドロックをコンパイラの静的なスケジューリングによって回避することで、条件分岐に使われているボラタイルオブジェクトを、親オブジェクトが行う処理の一部として実行することを可能にし、処理コストを低減する。

2 ボラタイルオブジェクトによる条件分岐

オブジェクト指向モデルを持つ言語では、未だ制御構造の標準的な表現方法が確立していない。C++ や CLOS など多くの言語では手続き型言語の制御構造を採り入れて使っている。一方、純粋なオブジェクト指向言語の Smalltalk-80 では、制御構造を扱うために、ブロックという、手続きを抽象化するためのデータ型を用意している。

A'UM では、条件分岐をボラタイルオブジェクトによって表現する。本来、オブジェクト指向モデルでは、なんら特別の機構を導入せずとも、メッセージによるメソッド選択の機構を用いて条件分岐を実現することができる。しかし、この実現方法には、そのまま実行すると効率が悪い、記述が繁雑になる、などの問題があった。ボラタイルオブジェクトを用いれば、メソッド選択の機構によって実現される条件分岐を簡潔に記述することができ、さらに第4節で述べる実装方式によって、充分な効率で実行することができる。

ボラタイルオブジェクトは、メソッド定義の中に埋め込まれた名前の無いクラス定義によって記述され、そのメソッドの実行時に生成される。ボラタイルオブジェクトのメソッドの中からは、そのボラタイルオブジェクトを生成したメソッドの中の変数を参照することができる。

A'UM での条件分岐は、次の手順によって行われる。まず、大小比較や、データ型判定などの条件判断メソッドが実行され、判断結果として、ブール値オブジェクト'true あるいは'false が生成される('true あるいは'false は A'UM でのブール値の表現)。次に、ブール値オブジェクトに対して、:who_are_you というメッセージが送られる。これを、受け取ったブール値オブジェクトは、その値に応じてメッセージ:'true または :'false のどちらかを、:who_are_you メッセージの引数として渡された宛先に送る。条件分岐では、この宛先としてボラタイルオブジェクトが指定される。このボラタイルオブジェクトには、メッセージ:'true および:'false に対応するメソッドが定義されており、受け取ったメッセージに従ってどちらかのメソッドを実行する。こうして、条件判定の結果に応じて異なる処理が行われる。

```
:foo(^X) ->                    (1)
    (Y > 3) ? (                (2)
    :'true ->                  (3)
        X :reset;              (4)
    :'false ->                 (5)
    ),                         (6)
    X :get(^Y).                (7)
```

図 1: 逐次処理によるデッドロックの例

3 逐次実行によるデッドロックの可能性

A'UM-90 では、条件判断に使われるボラタイルオブジェクトを親オブジェクトの処理の中に埋め込んで実行する。これにより、条件分岐の実行効率を上げることができる。しかし、A'UM のボラタイルオブジェクトは、通常のオブジェクトを生成して親オブジェクトと並列に実行するのと同じ意味を持つため、単純に親オブジェクトの処理に埋め込んで実行すると、本来生じないはずのデッドロックが生じる恐れがある。

ボラタイルオブジェクトは、メッセージが届くまで処理を始めない。ボラタイルオブジェクトを親オブジェクトの処理と並べて順番に実行すると、ボラタイルオブジェクトがメッセージを待っている間、親オブジェクトの処理も中断する。もし、たまたまボラタイルオブジェクトよりも後に回された親の処理の中に、ボラタイルオブジェクトが待っているメッセージが届くために必要な処理があれば、ボラタイルオブジェクトはいつまでもメッセージを待ち続けることになり、デッドロックが生じる。コンパイラは、親オブジェクトの処理とボラタイルの実行順序を決める際に、上に述べたようなデッドロックが起きないように実行順序を決めなければならない。

図1に、そのまま逐次実行するとデッドロックが起きるプログラムの例を示す。これは、メッセージ:foo に対するメソッドの定義である。行(2)から(6)までが条件分岐処理を表しており、このうち、'? ('から')'までがボラタイルクラス定義である。行(2)で調べている条件 Y > 3 が真ならば行(4)の処理を行い、条件が偽ならば何もしない。一方行(7)はメッセージの送信を表す式であり、メッセージ:get の引数として ^Y を X に送っている。Y に付随している'^'は、Y の値をメッセージ:get が X に起動するメソッドが決めることを示している(同様に、行(1)の X に付随している'^'は、X の値をメッセージ:foo の発信者が決めることを示している)。

行(2)の条件判定 Y > 3 を行おうとしたとき Y の値がまだ決まっていなければ、この判定処理は中断して Y の値が決まるのを待つ。モデル上は行(2)から(6)までの条件分岐処理と、行(7)の送信 X :get(^Y) は並列に行われるので、条件処理分岐が中断している間に Y が X に送信されて X が Y の値を決める。こうして、条件分岐処理を再開することができる。しかし、オブジェクト内の処理を逐次実行する実装においては条件分岐処理の中断はこのメソッド全体の中断であり、行(7)の送信は行われない。そのため、Y の値はいつまでも決まらず、条件分岐もいつまでも再開されない。こうしてデッドロックが生じる。

Implementation of Volatile Objects in A'UM-90
Koichi Konishi, Tsutomu Maruyama, Akihiko Konagaya,
Kaoru Yoshida †, Takashi Chikayama †
NEC Corporation, †ICOT

```
'foo/+':                                          (1)
        create_mjoint R1;                         (2)
        send R0, 'get/-', 1, R1;                  (3)
        create_integer R2, 3;                     (4)
        if_gt R1, R2, Label_for_true,
            Label_for_false;                      (5)
        gt R1, R2, R3;                            (6)
        who R3, R1;                               (7)
        create_mjoint R2;                         (8)
        create_volatile R3, R2,
                    #if_then_else;                (9)
        connect R3, R1;                           (10)
        send_self 'continue/+++', R0, R2;         (11)
        wait;                                     (12)
                                                  (13)
'continue/+++':                                   (14)
        if_true R1, Lable_for_true,
                        Label_for_false;          (15)
                                                  (16)
Label_for_true:                                   (17)
        send R0, 'reset', 0;                      (18)
                                                  (19)
Label_for_false:                                  (20)
        close R0;                                 (21)
        descend;                                  (22)
```

図 2: 最適化した条件分岐コード

コンパイラは、デッドロックを防ぐために行(7)の処理を先に行うコードを出さなければならない。モデルでは、条件分岐処理と行(7)の送信は並列に行われる。しかし、これらを逐次的に実行する場合には、デッドロックの発生はプログラマの意図ではないから、行(7)の送信を先に行うコードこそが正しいコードである。

4 条件判断の最適化

第2節で述べた言語モデル上の処理手順は、このまま実行するには処理コストが大きく、条件判定のように頻繁に行われる作業には適さない。ボラタイルオブジェクトの主な用途は条件分岐であって、この場合ボラタイルの機能のごく一部しか必要としないので、この場合には最適化を行って、もっとコストの小さい方法で実行する。最適化手法は、条件判定を行おうとした時に条件判定を直ちに行うことができるかどうかで、2通りに分かれる。

第一の最適化手法が適用できるのは、プログラムの実行が条件判定にさしかかった時点で、その条件判定演算の引数が判明していて、それが数値や文字列などの基本データ型のオブジェクトである場合である。この場合は、ただちに判定を行い、結果に基づいてコード中の適切な位置にジャンプする。ボラタイルオブジェクトの定義に書かれたメソッドは、この後親オブジェクトによって実行される。親オブジェクトがボラタイルオブジェクトのコードを実行するので、ボラタイルオブジェクトのコードは、レジスタを経由して親オブジェクトのスコープに属する変数にアクセスすることができる。この手順を用いる場合、条件分岐のオーバーヘッドは、条件判定演算の引数が判明しているかどうかの検査だけで済む。

コンパイラは、上述の手順を実現するために、コード中の条件分岐に関係する部分の先頭に即時の条件分岐のための命令を置く。図2に、図1の条件分岐を実行するコードを示した。この図の行(5)に即時分岐命令が置かれている。即時条件分岐命令には、これが行う条件判定の種類に応じて何種類かがあるが、いずれも条件判定の引数が判明しているかどうかの検査と、判明していた場合には、条件判定とその結果に応じたジャンプを行う。引数が判明していない場合には、それ以上何もしない。

条件判定演算の引数が判明していないか、判明していても、基本データ型のオブジェクトでない場合には、第二の最適化手法を用いる。この場合、条件判定は実際にメッセージを送って条件判定メソッドを起動することによって行い、その結果生成されるブール値オブジェクトにも実際に:who_are_you メッセージを送る(行(6),(7))。しかし、ボラタイルオブジェクトは生成せず、代わりに、ブール値に対応するメッセージを受信させるための同期用オブジェクトを生成する(行(9))。この後、条件分岐を行おうとしているオブジェクトは、メソッドの実行を中断する(行(12))。このとき、このオブジェクトはメッセージを作って自分のメッセージキューの先頭に置くことによって実行再会後の処理を示す(行(11))。このメッセージには、中断されるメソッドで使用中の一時変数を引数として持たせる。同期用オブジェクトは、受け取ったメッセージに応じてブール値'true または'false を生成し、親オブジェクトに渡して親を起こし、消滅する。親オブジェクトは、自分でキューに置いたメッセージに従ってメソッドを起動し、渡されたブール値オブジェクトを調べてジャンプする(行(15))。この場合もボラタイルオブジェクトの処理は親オブジェクトが行い、ボラタイルのコードはレジスタを経由して親のスコープの変数を共有することができる。

5 おわりに

以上、並列オブジェクト指向 A'UM のボラタイルオブジェクトが条件分岐に使われている場合の実行方式の最適化手法を説明し、条件分岐のために特別な機構を言語モデルに導入しなくても、充分な効率を持つ条件分岐機構を実現できることを示した。今後の課題としては、逐次実行の際のデッドロックのコンパイラによる検出方式や、および条件分岐以外の用途に用いられる、一般の場合のボラタイルオブジェクトのための効率のよい実装方式の検討がある。

謝辞

A'UM-90 処理系の開発に参加してくださった、日本電気技術情報システム開発株式会社の柳田氏、小柳氏、丹下氏に感謝致します。

参考文献

[吉田88] K. Yoshida and T. Chikayama, "A'UM – A Stream-Based Object-Oriented Language –," Proc. Int'l Conf. on Fifth Generation Computer Systems (FGCS '88), pp. 638-649, ICOT, November 1988.

[吉田90] Kaoru Yoshida, "A'UM: A Stream-Based Concurrent Object-Oriented Programming Language," Ph. D thesis, Keio University, March 1990.

[小西90] 小西、丸山、小長谷、吉田、近山、「並列オブジェクト指向言語 A'UM-90」, JSPP'90 論文集, 1990 年 5 月

[丸山90] 丸山、小西、小長谷、吉田、近山、「ストリームに基づく並列オブジェクト指向言語 A'UM-90– ストリーム分散実装方式 –」、情処学会第41回全国大会講演論文集、2E-2, 1990 年 9 月