並列オブジェクト指向言語 Fleng++ の優先通信の実装とストリーム操作の高速化

2Q-4

吉田 実, 田中 英彦
minoru@mtl.t.u-tokyo.ac.jp, tanaka@mtl.t.u-tokyo.ac.jp
東京大学工学部*

1 はじめに

コミッティッドチョイス型言語(CCL)は、ストリーム通信を行なうプロセスを利用して、オブジェクト指向プログラミングが可能である。CCL の一つである Fleng を基に したオブジェクト指向言語 Fleng++ は、このストリーム通信をするプロセスをオブジェクトとしてプログラマに見せることによって、また、継承機能、インスタンス変数の導入によりプログラムの記述性、可読性を高めたものである [1, 2]。一方、優先通信の実装によって、プログラムの実行の制御を行なうことができるようになりシステム記述が可能になった。

2 並列オブジェクト指向言語 Fleng++

Fleng++ は、プログラムのモジュール化の単位としてクラスと呼ばれるものを持っている。クラス定義は、継承クラスの指定、クラス名の宣言、インスタンス変数の宣言、メソッドの定義、ローカル述語の定義からなる。インスタンス変数とは、何度でも値を代入できる変数である。継承には多重継承を採用しており、他のクラスの定義を取り込むことができる。あるオブジェクトは、自分や他のオブジェクトにメッセージを送ることができる。オブジェクトは、逐次解釈、並列実行を原則としている。

Fleng++ では、あるオブジェクトにメッセージを非同期に送ることができる。インスタンス変数は副作用を持つので、メッセージ受信の順序によって結果が変化することがある。そのような場合には、メッセージを送る順序を指定したいことがある。これをメッセージの直列化と呼ぶ。Fleng++ では以下の規則を設けた。

- 自分自身へのメッセージは他のオブジェクトからのメッセージより優先される。
- 同じオブジェクトへのメッセージの送信は、メソッド定義の中で左から右へ深さ優先で行なわれる。

2.1 優先メッセージ

優先メッセージとは前節でのオブジェクトの直列化規則の枠を越えてメッセージの送信を行なうことができるものである。通常のメッセージは、直列化規則により決定される順序で受信されるが、優先メッセージはその規則を無視して可能な限り先に受信される。この機能は、緊急性のある処理、例えばオブジェクトの実行状態を変更する時などに使用する。特に、オブジェクトの実行を停止した後の再開のメッセージは優先メッセージを利用しないと受信されない。この機能により、OS などのシステムプログラミングを Fleng++ で記述することが可能になる。

3 インプリメント

3.1 オブジェクトの実行制御

あるオブジェクトの生成には、それに対応する述語が呼ばれる。その述語は、ストリームを引数に持ち、そのストリームをメッセージとして順に解釈する。

```
class_a([M|S]) :- method(M,I),class_a_aux(M,S).
class_a_aux(msg,S) :- ....., class_a(S).
```

のような構造になっていて、オブジェクトの生成は class_a/1 を呼び出すことで実現される。ストリームを解釈するゴールは常に 1 つで、これをリスナと呼ぶ。リスナ及びリダクションによって生じるゴールはリダクションにより新たなゴールを作り出して行く。これは、一つの単位を成す。よって、オブジェクトを実行制御の単位とすることができる。実行制御とは、実行停止、強制終了、実行再開、スケジューリングの制御などである。あるオブジェクトに実行が渡ると、オブジェクト内の実行単位であるゴールを取り出してはリダクションを進めて行く。オブジェクトの実行制御は、このリダクションを制御することで実現される。

3.2 優先通信の実現

前節の実行制御も制御命令がなるべく速く有効になるのでなければ、十分な効果は期待できない。そこで、図 1 のような構造でオブジェクトを実現する。優先通信をあるオブジェクトに送りたいときは、オブジェクトのストリームを表す場所を知っていればよい。優先メッセージの送信は、メッセージストリームの先頭に優先メッセージを入れることで実現する。優先メッセージの挿入とリスナのリダクションは排他的に実行されなければならない。

3.3 ストリーム通信の高速化

オブジェクトの参照を増やすためには、直列化の必要な所では append、そうでない所では merge が使われる。これらのストリーム操作は、ナイーブなインプリメントでは 1 つのメッセージ送信のたびに呼び出され、特に多入力の merge は、典型的な非決定的処理で効率が悪いものになる。Fleng++ コンパイラは、部分計算により可能な限りこれらの操作の呼び出しを

*"Imprimentation of Priority Communication and High Speed Stream Operation in Parallel Object-Oriented Programming Language Fleng++", YOSHIDA Minoru and TANAKA Hidehiko, Univ. of Tokyo, Faculty of Engeneering

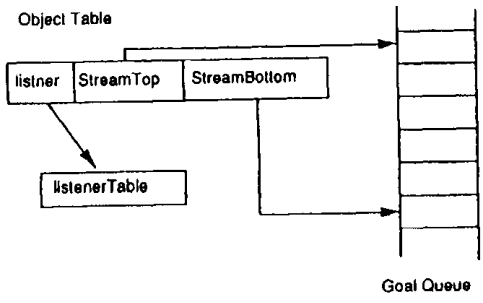

図 1: オブジェクトの構造

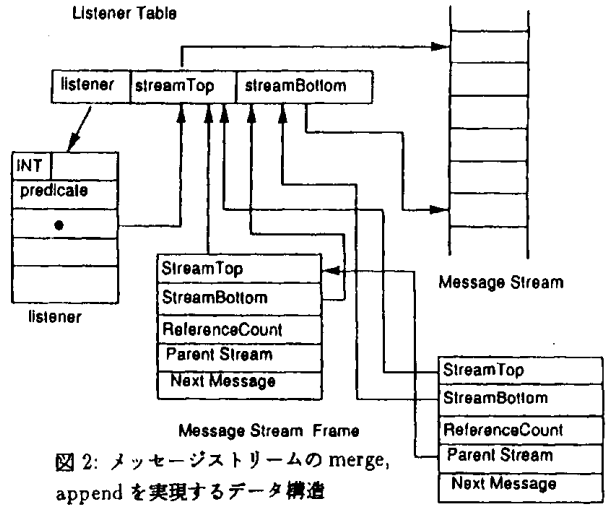

図 2: メッセージストリームの merge, append を実現するデータ構造

なくしているが、やはり必要とされる場所がある。merge については、ストリームの最後に付け加え、ストリームの最後を指しているポインタを書き換える。この方法は、引数が増えてもコンスタントオーダで実行できる。append ついては、直列化規則により第2引数には、常に、第1引数のメッセージ送信が終了した時に送るべきメッセージが1つだけ溜っているはずである。このことを利用することで、append の処理は簡単になる。図2にデータの構造を示した。

通常のメッセージの送信

ストリームへのメッセージの送信の必要のある場合は、メッセージストリームフレーム (MSF) へのポインタを持つ。通常メッセージの送信は以下の操作を行なう。

1. MSF の StreamBottom を読む。
2. そのポインタの指すリスナテーブルの StreamBottom を読む。
3. そのポインタの指すストリームの最後に送信するメッセージを付け加え、StreamBottom を更新する。

最後の操作は、他の同様な操作に対して排他的に行なわなければならない。

優先メッセージの送信

優先通信の送信は、上記の StreamBottom の代わりに StreamTop を使う。

メッセージストリームの参照の増減

メッセージストリームの参照の増加時には、MSF の RefernceCount を1つ増加させる。一方、減少時は RefereceCount を1つ減少させるのであるが、もし、その値が0になったら、NextMessage を送信して、その MSF は終了する。その場合、ParentStream が存在すれば、その MSF の参照数を同様に1つ減少させる。ここで、レファレンスカウントを行なうのは、ストリームの終了を知るためである。

上記のインプリメントによりメッセージ送信は、常にコンスタントオーダで行なうことができる。

リスナのメッセージ受信

リスナは、引数にそのメッセージストリームへのポインタを持っている。そのポインタを通じてアクセスするので、リスナのリダクションとストリームの先頭の書き換えは排他的でなければならない。リスナのリダクションは、そのゴールの中に1つだけリスナがあり、それが次のリスナになる。

3.4 メソッドサーチ

メッセージ送信の高速化と共に、受信も高速化が重要である。メソッドサーチは、そのメッセージ受信の中心部である。メッセージを見て、それがどのメソッド呼び出しかを判定するためには、ハッシュを使って実装する。

4 プログラミングの実験環境

現在、EM や優先通信を実装した Fleng++ を開発中をすすめている。Fleng++ は、上記の拡張を行なった Fleng に変換される。そして、そのコードは、機械語を意識した AIL という中間言語にコンパイルされる。AIL は、C や機械語に変換される。現在、C へのコンパイラが利用できる。ユーザは、処理系から Fleng++ のプログラムをコンパイルし、最終的な実行可能モジュールを読み込んで使用する。

5 まとめ

優先通信はシステムプログラミングに必要な機能であり、また、通常のプログラミングにおいても、プログラムのモデル化を容易にし、プログラムを効率の良いものにすることができる。また、ストリームを中心とした言語では、merge、append の高速化は重要である。これらの実装法について報告した。今後は、実験環境の充実と実際のプログラミング経験による言語諸機能や環境へのフィードバックを行なう予定である。また、我々が開発中の並列推論マシン PIE64[3] 上へのインプリメントの詳細設計を同時に行なう。

参考文献

[1] 中村宏明, 田中英彦, "並列オブジェクト指向言語 FLENG ++ の実装", 第38回情報処理学会全国大会 6Q4(1989)

[2] 中村宏明 "Committed-Choice 型言語に基づいたオブジェクト指向プログラミング・システムの研究", 東京大学工学部情報工学専攻修士論文 (1989)

[3] Hanpei Koike and Hidehiko Tanaka, "Multi-Context Processing and Data Balancing Mechanism of the Parallel Inference Machine PIE64", Proc. of Int. Conf. on FGCS 88', pp.970 - 977.