9F−2 並列オブジェクト指向言語A-NETLへの実行順序制御機構の導入と実行性能の改善

田口弘史　吉永努　馬場敬信
tagtag,yoshi,baba@infor.utsunomiya-u.ac.jp
宇都宮大学工学部*

1 はじめに

我々は並列オブジェクト指向概念に基づいたトータルアーキテクチャA-NETを開発している[1, 2]。

今回、A-NET計算機においてその動作単位であるオブジェクトの自律的な動作[3]を補助するために、メソッドの実行順序や実行可能条件の判断機構をA-NETL言語に付加した[4]。

さらに、かねてより問題となっていたシステムの実行比率の軽減を目的とし、従来ローカルOSが自動的に行なう予定であったメソッド実行条件の判断処理をやめ、A-NETLコンパイラが、オブジェクト内に実行条件判断ルーチンを選択的に付加する方式に変更した。

本稿では、実行順序制御の実装の方法および実行性能の改善の効果について述べる。

2 メソッド実行の制御

A-NETの動作単位であるオブジェクトは、普通、複数のメソッドを持っている。これらのメソッドは任意のオブジェクトからのメッセージで起動されるが、その実行に関しては受けとったメッセージをそのまま起動する単一受信待機と、複数のメッセージを受信するまで待機して起動する複数受信待機がある。

実行順序制御では、オブジェクトの自律的な動作を補助する目的で、メソッドの実行に制約条件を与える。具体的には、

- 優先順位の付加
  あるオブジェクト内におけるメソッドに対して優先順位を設け、優先度の高いメソッドから実行させる。これはメッセージの追い越しを防ぐ。
- 二重起動の禁止
  あるメソッドが未来トラップの状態にある場合、同じメソッドを追加起動することを抑止する。これは、未来トラップで休眠状態にあるオブジェクトにおいて、後から起動されたメソッドによる状態変数の変更を防ぐために必要である。
- 変数の比較
  あるオブジェクトの持つ状態変数およびメソッドのメッセージ引数を比較し、その結果が"真"になるまで待機する。メッセージの起動条件をより柔軟に記述できるように用意した。

の3種類の条件を導入する。

A-NETLで記述されたプログラムは、従来スーパーバイザとなる特殊なオブジェクトを用意し、同期をとりながら実行されるものが多く見られた。実行順序制御機構の導入により、オブジェクト内の各種の情報を活かしてメソッドを実行することが見込まれ、同期のためのメッセージの抑制が期待でき、そのため、非同期MIMD型であるA-NET計算機の性能を一層引き出すことができる。

また、簡潔な記述によってオブジェクト内に条件判断処理を付加することができるので、ユーザは複雑な制御条件をメソッド内に記述する必要はなく、ユーザプログラムの開発補助となり得る。

3 実行順序制御条件の記述

実行順序制御機構を用いるための条件の記述はオブジェクト内の先頭において行なう(図1参照)。

```
object   cell[MAX]
control
    SR  { newage < enquete }
    SEQ { enquete  (gen   = generation),
          newage   (gen   = generation),
          newage   (count = accept)     }
    LOCK{ singleAns }
state
    generation=0.
    count=0.
    accept=0.
    ...
methods{
    enquete:   gen {  ...  }
    newage:    gen {  ...  }
    singleAns: gen {  ...  }
    ...
}
```

図1: 実行順序制御の記述例

オブジェクト名の直後、予約語controlに続いて条件を列挙する。条件記述は以下の3種類の方法で行なう。

- `SR { methodA < methodB }`
  優先順位の付加を行なう。この場合methodBはmethodAに優先する。実装上は`count(methodA) < count(methodB)`が起動の条件である。ここで、`count()`はメソッドが実行終了した回数である。
- `LOCK{ methodA }`
  メソッドの二重起動を禁止する。この場合、methodAは、未来トラップ中に追加(二重)起動されることはない。このとき、`start(methodA) = count(methodA)`が起動の条件である。ここで、`start()`はメソッドが実行を開始した回数である。
- `SEQ { methodA( var1 < var2 ) }`
  任意の変数の比較を行なって実行を制御する。この場合、methodAは`var1 < var2`であるときにのみ起動される。比較演算子は<の他に=,>が使用できる。これはそのまま実装上の起動の条件となる。var1,var2は、そのオブジェク

*Description of Execution Sequence Control in A-NETL and Its Implementation , Hiroshi TAGUCHI , Tsutomu YOSHINAGA , and Takanobu BABA , Utusnomiya University.

トの持つ状態変数、または、そのメソッドのメッセージ引数が指定できる。

4 実行性能の改善

4.1 実現の方針

実行順序制御機構は、ローカルOS内に実行イメージ(コンテクスト)の退避を行なう専用のキューを設け、それを利用することで実現する。このキューに対して、退避及び再起動の要求を与え、実行順序制御機構として作用させるが、これらの要求はユーザプログラムからのトラップの形式をとることとした。

当初、実行順序制御のためのメッセージのデータ構造を作成し、ローカルOSにおいて特定のフラグを参照することで実現する方法を考えていた。しかし、ローカルOSのスケジューリングのメインループ内にフラグやデータ構造の参照を追加することは、効率を考えた場合好ましくない。

そのため、実行順序制御のためのトラップを用意し、A-NETLコンパイラが、要求に見合う条件判断部とトラップ命令とをユーザプログラムの前処理部として選択的に挿入する方法を用いる。キューに退避したコンテクストの再起動は、そのオブジェクト内の状態変数またはメソッド実行回数の変化をきっかけとして行なうため、後処理部としてユーザ定義のメソッドの最後にキューの先頭を再起動するトラップを付加する。

また、複数のメッセージの到着によって起動する複数受信待機の機能も、トラップを用いて実現することとした。

4.2 実行性能改善の効果予測

複数受信待機、実行順序制御ともに、プログラムを作成した時点で静的に利用するかしないかが決定しているため、コンパイラがその判断を行うことが可能であり、それによってローカルOSが行う条件判断をユーザプログラム内に選択的に転嫁することができる。これらの条件判断はローカルOSのスケジューリングのメインループに存在するため、それを選択的にユーザプログラムに移すことは、ローカルOSの負荷を大幅に軽減し、かつ全体の実行性能の向上につながる。

図2は、ローカルOSのスケジューリングの流れを示している。図中、`MRCtrap`は複数受信待機、`XTCtrap`は実行順序制御、`CQStrap`はコンテクストキュー活性化用に用いるトラップである。破線の矢印はコンパイラが選択的に生成する流れであり、その部分を通るか否かはコンパイル時に判断する。

これに見られるように、複数受信待機と実行順序制御機構の条件判断をローカルOSからユーザのプログラムに転嫁している。その効果は、前述のとおり、ローカルOS内部における、メッセージの構造体を解析してその情報に基づく条件判断を省略することが可能となったことである。

この効果が大きいのは、その最も頻繁に起る単一メッセージ受信待機かつ実行順序制御を使用しない場合であり、メッセージの起動一回に対して3つの条件判断を省略することができる。これは、プログラムが大量のメッセージをスケジューリングすることで進行することを考慮すると、大きな性能向上を導くことになる。

また、実行順序制御機構の利用はメッセージの発行数を減少させる。実行条件を有効に活用することで、同期を行うためのメッセージを極力抑えることが可能となるためである。このことは、ユーザのプログラム記述が簡潔になるばかりでなく、同期のためのオーバヘッドが軽減できるため、要素プロセッサにおける実行時間に占めるユーザの割合いを引き上げることにつながる。

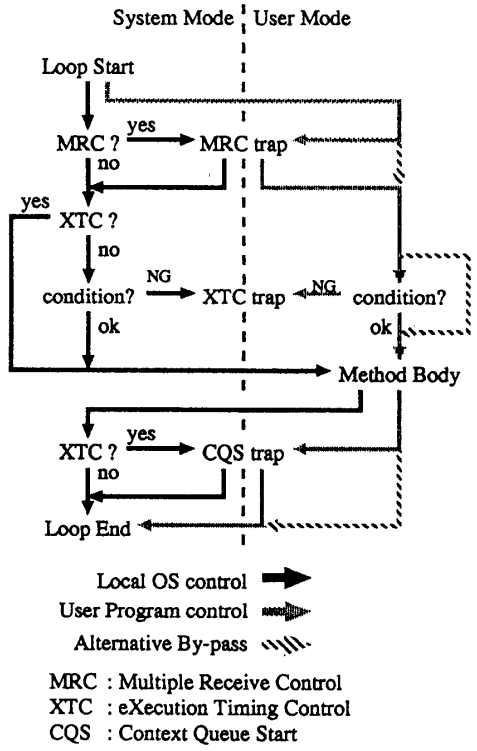

図2: ローカルOSの処理の流れ

5 おわりに

A-NETLに実行順序制御機構を導入し、ローカルOSの改良を行って実行性能の向上を試みた。現在、実装を終えており詳細な評価を行う準備を進めている。同一のプログラムを実行順序制御の使用を含め幾種類か用意し、それぞれの比較をA-NETシミュレータ上で実験する予定である。

また、ユーザプログラムから発行されるトラップは、実行順序制御条件の種類にかかわらず同一のものであり、これにより、新たな条件判断機構の導入は容易であり、拡張性の高い実現方法となったことを言い添える。

参考文献

[1] Baba, et al., "A Parallel Object-Oriented Total Architecture: A-NET", Proc. Supercomputing '90, pp.278-285(1990)

[2] 馬場 他, 並列オブジェクト指向トータルアーキテクチャA-NETにおける言語とアーキテクチャの統合, 信学論招待論文 Vol.J75-D-I, No.8, pp563-574, 1992

[3] 丸一 他, "自律的エージェントからなる組織の計算モデルと分散協調問題解決への応用", 情報処理学会論文誌,Vol.31 No.12(1990)

[4] 津田, "A-NETLへの実行順序制御機構導入とコンパイラの拡張", 宇都宮大学卒業論文 (1992)