並行言語NET/Cのマルチスレッド実装†

4L－6

青柳 洋一, 中林 嘉徳, 岩田 竜一, 上原 稔, 森 秀樹 ††
東洋大学工学部情報工学科†††

1 はじめに

共有メモリ型マルチプロセッサが普及しつつある現在、ハードウェアの性能を引き出して処理を行う並行処理への関心が高まっている。一般に並行プログラミングにおいては、資源アクセスや処理の実行のタイミングなど細かな制御が要求され、プログラミングが繁雑になりがちである。また、マルチプロセッサ環境を生かして効率的な実装を行おうとすると汎用性のないプログラムになってしまうことがある。我々が開発中の並行言語NET/Cは、オブジェクト間の通信方式にストリーム通信を採用することにより制御の流れを明確にし、並行プログラムをUNIXプロセスのパイプによる連結処理のようにシンプルな思考で記述できる仕様となっている。本研究では、NET/Cの実装を共有メモリ型マルチプロセッサ上のスレッドを利用して行い、コンパイル時にコード解析で得た情報により実行形態の選択をして効率化を行った。2節でNET/Cの特徴の紹介、3節でNET/Cのマルチスレッド化に当って行った仕様の拡張について述べる。4節でまとめと今後の方針を述べる。

2 NET/Cの特徴

特徴的な並行オブジェクト指向言語として、Chorusシステムのオブジェクト指向層としてOSと密接に実現され、システムレベルのスレッドを利用してオブジェクトを実現するCOOL[2]、ユーザレベルのスレッドでオブジェクトを実現し、runtimeの最大スレッド数を指定可能にして多様なスケジューリングをサポートしたcooC[3]などがあり、それぞれに優れた特性を持っている。

NET/Cはこれらの言語の優れた仕様を採り入れている。(1)NET/Cオブジェクトは COOLのようにアクタとして表すことによってオブジェクトの抽象度を高め、(2)値を返した後も実行を継続するreply文の導入とスレッドの利用によって並列性を引き出している。また、(3)仮想マシン上のシミュレーションで処理を実現することによって機種に依存しない汎用性とデバッギングに有利な環境を提供している。

本研究では、共有メモリ型マルチプロセッサ上で効率的なスレッドを提供するOS Mach2.5上でNET/Cのマルチスレッド化を行った。

3 スレッド利用による並列化

マルチプロセッサ環境では、プロセッサへの適切な負荷分散によって処理時間を短縮することが可能であり、並列処理の実現方法としてプロセスやシステムレベルで提供されるスレッドの利用などが考えられる。文献[4]において3つの実行単位(プロセス、スレッド、並行オブジェクトのシミュレート)の組み合わせによる実行形態の特性調査を行った結果、1台の共有メモリ型マルチプロセッサで並列処理を実現した場合、プロセスよりもスレッドを用いて実現した方が良い性能を得やすいことが確認された。このことからシステムレベルのスレッドの備わった環境では、並列化にスレッドの利用を前提とする。実行時のスレッド利用の有無によってスケジューリングの方法が異り、効率が大きく異る。以下にNET/Cのスケジューリング、アクタ、ストリームの実現方式について述べる。

スケジューリング runtimeのスケジューリングレベルの指定方法として、以下の3つのレベルを用意し、実行時に指定可能とした。

1. スレッドを利用しない実行：
   動作の再現性があるためデバッグに有効。
2. アクタとスレッドを1対1に対応させた実行：
   メソッドがcallされる度にアクタにスレッドを割り当てる。一見、細かな並列性を引き出すことができて効率が良さそうだが、スレッド数が多過ぎるとシステムの負担が大きくなり効率が低下する。
3. NET/Cコードの解析により最適と考えられる実行形態でスレッドを割り当てる方法：
   通常はこの方法で実行を行うが、コードに静的に解

†Multi-thread Implementation of Concurrent Language NET/C
††Yoichi Aoyagi, Yoshinori Nakabayashi,Ryuichi Iwata, Minoru Uehara, Hideki Mori
††† Department of Information and Computer Sciences, Toyo University

実行方式	1	2	3
アクタ管理	仮想機械による集中管理	スレッドによる分散管理	スレッドによる分散管理
アクタ実行	仮想機械	Native 実行	仮想機械
ストリームの排他制御	不要	要	要

表 1: アクタの実現方式による管理の違い

析できない要素が含まれる場合は最適な形態を割り当てることが難しい。このようなときには2の方法で実行を行うが、2の方法でパフォーマンスが得られない場合や動作が不安定な場合は1の方法を用いて処理を行う。最適な実行形態は以下のようにして求める。

(a) NET/C 仮想マシンの各命令コードやプリミティブの実行時間、通信コスト等の値は、NET/C セットアップ時の調査により平均値が既知とする。

(b) コンパイルで生成された NET/C コードを一定の特性 (loop、直線的、条件分岐の直前など) を示す処理単位で複数パートに分け、各パートについて実行時間の予測値を計算する。

(c) 各パートの予測時間を得て文献 [4] に基づき適切と考えられる実行形態を決定し、NET/C コードと関連付けられたファイルにセーブする。
… 実行時に NET/C コードと一緒にロードし、この実行形態情報に従ってアクタへの実行単位の割り付けが行われる。

マルチスレッド化する上での主な変更点は表 1の通りである。

NET/C アクタ アクタは NET/C コードのシミュレートまたはスレッドによって割り当てられる実行単位で、アクタ毎にプログラムカウンタ、メソッド情報などのコンテキストを保持している。アクタは NET/C コード中にメソッドコールが発生する度に生成される。

アクタ管理は、1 の方式では仮想機械によって一括管理されるのに対し、2,3 の方式では個々のスレッドの権限で管理が行われる。

アクタ実行は、1 の方式では仮想機械によるインタプリタ方式で行われ、2 ではマシンに Native なコードで実行される。3 の方式ではスレッド毎にインタプリート実行される。

ストリーム通信 ストリームは、極性を持つバッファであり一方が書き込み側 (out) で他方が読みだし側 (in) となっている。データに方向性を持たせることによって NET/C プログラミングにおける制御の流れを明確にしている。ストリームは、アクタの生成前にアクタ同士を関連付けて生成される。

表 1の排他制御は、1 の方式ではストリームを構成する変数への同時アクセスが生じないため排他制御が不要なことを示している。

4 まとめ

- 通信に占める割合が計算処理の割合に比べて大きい問題では、マルチスレッド化による効果が小さい。解析時にこのような予想が可能な問題については、マルチプロセッサ上であってもスレッドを使わない実装によって、システムへの余計な負荷を小さくすることが可能である。
- 条件分岐先が実行時に決まる、ループの繰返し数を特定できないなど静的に実行時の動作や実行時間を推測できない場合がある。また、システムによる最適化の影響まで厳密に解析するのは難しい。これらの影響が大きい場合は静的な解析だけでは不十分であり、動作時の状況を解析して調整することが必要である [1]。動的な tuning をサポートするビジュアルなデバッガと組み合わせて調整を行うようにする。

参考文献

[1] Prabha Gopinath, Thomas Bihari, and Rajiv Gupta. Compiler Support for Object-Oriented Real-Time Software. *IEEE SOFTWARE*, 1992. September.

[2] Rodger Lea, Christian Jacquemot, and Eric Pillevesse. COOL: SYSTEM SUPPORT FOR DISTRIBUTED PROGRAMMING. *COMMUNICATION OF THE ACM*, 1993. September.

[3] R. Trehan, N. Sawashima, A. Morishita, I. Tomoda, A. Inoue, and K. Maeda. Concurrent Object Oriented C (cooC). *ACM SIGPLAN NOTICES*, 1993. February.

[4] 青柳洋一, 中林嘉徳, 岩田竜一, 上原稔, 森秀樹. 密結合マルチプロセッサにおける異粒度オブジェクトの実行系の評価. 日本ソフトウェア科学会第 11 回大会論文集, 1994.