並行言語 NET/C の Native 実行系の実装†

4N－5

青柳 洋一, 上原 稔, 森 秀樹 ††
東洋大学工学部情報工学科†††

1 はじめに

並列処理への期待が高まりつつある現在、使いやすく効率のよい並行オブジェクト指向言語が求められている。一般に並行処理においては、排他制御など処理の依存関係の制御が必要とされ、プログラムが複雑になりがちである。また、マルチプロセッサなどの環境を生かした効率的な実装を行おうとすると、システム特有の機能を用いるために汎用性が失われてしまう傾向がある。並行言語 NET/C はストリーム通信を基本とすることにより、制御の流れの明確化を図り、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簡潔に記述できることを特徴としている。また、仮想マシン (VM) 方式で NET/C コード生成部と VM 実行部を切り放すことにより汎用性を得ている。しかし、VM のインタプリート実行はスピードを要求される処理には不向きであり、高速な実行系が求められていた。

本研究では、NET/C コードを C 言語にトランスレートし、マシンに Native なコードを生成して実行させることで NET/C の高速な実行を可能にした。2 節で NET/C の概要を述べ、3 節で NET/C の効率化の考察、4 節で Native 実行の評価、5 節でまとめを述べる。

2 NET/C の特徴

NET/C の特徴として以下のものがあげられる。

ストリーム通信 通信対象を固定し、データに方向性を持たすことにより制御の流れを明確にし、静的解析を行いやすくする。ストリーム通信のバッファ管理を実行部で管理することにより、プログラマの排他制御の負担を軽減する。

並行実行の単位アクタ アクタは各々コンテキストを持ち、値を返した後も実行を継続する reply 文の導入により、並行性を導出している。

VM 方式 コード生成部と実行部の分離により汎用性と解析に有利な環境を提供している。

しかし、VM のインタプリート実行は速度が抑えられてしまうため効率のよい実行系が求められていた。

3 NET/C 実行部の効率化

NET/C 実行系の高速化の手段として (1)VM のマルチスレッド化、(2)アクタ合成、(3)Native コード化などが考えられる。

VM のマルチスレッド化 処理系のマルチスレッド化を行った研究として、MultithreadSmalltalk[2] では、Smalltalk プロセスを Mach のスレッドに割り当て、動的コンパイルする方法で Smalltalk の高速化を実現している。文献[3] では、Mach スレッドを用いマルチプロセッサ上で VM の負荷分散を実現したが、細かなスレッドが多数生成されるような処理では、管理のためのコストが大きくなり、効率を得にくいものとなった。

アクタ合成 producer-consumer の対のように、同じ周期で処理を行う複数アクタを同一ループにまとめれば、排他制御やコンテキストスイッチなどアクタ管理のコスト削減が可能となる。NET/C ではストリーム通信に基づく通信対象が固定され、静的な解析を行いやすいためアクタ合成を実現しやすいと考えられる。

Native コード化 VM で行っているステップ解釈実行を C に置き変えコードの Native 化を行い、解析用に保持していた情報管理を廃して効率化を追求することで高速化が可能と考えられる。

本研究では、NET/C コードの Native 化による高速化を行った。VM 上のシミュレートで実現していた並行処理を C 言語に置き換える際に、並行処理の実現方法、効率、汎用性が重要と考えた。並行処理の実現方法としてスレッドの利用、プロセスの利用などが考えられるが、効率の点で見るとスレッドによる方法は共有データの活用によってデータコピー量を抑えることができ、プロセスを利用する場合に比べて効率化を行いやすい。また、スレッ

†Implementation of Native Execution System for Concurrent Language NET/C
††Yoichi Aoyagi, Minoru Uehara, Hideki Mori
††† Department of Information and Computer Sciences, Toyo University

ドの実現を POSIX 1003.4a 準拠のスレッドライブラリ pthread[1] を利用して行うことにより、汎用性を提供できると考えた。本研究では、pthread 利用による NET/C の Native 実行系を実現することにした。文献 [3] の VM のマルチスレッド化との違いは (1) コードの実行がインタプリート方式でなくマシンに Native なコードの実行である (2) システム依存のスレッドを利用しないため並列性はないが汎用性がある、という点である。VM のコードを C に変換するにあたり、以下の方法を用いた。

- ストリームの実現は、共有バッファを生成し、mutex の機構を用いて管理。
- 並行実行、逐次実行するアクタを区別し、並行動作させるアクタは pthread の機構を用いて実現。
- アクタへの引数は、vararg() を用いて一般化しやすいように、引数サイズを 8 バイトに固定して渡す。

表 1に従来の実行方式と本研究の方式の比較を示す。

	従来の VM 実行	Native 実行
実行	インタプリート	マシンのコード
並行処理	並行シミュレート	pthread の利用
ストリーム	共有バッファ	共有バッファ,mutex

表 1: 従来の実行と本方式の比較

4 評価

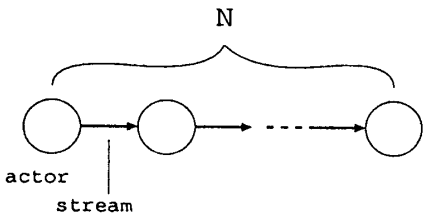

図 1: パイプラインの例題

Native 化によるアクタ生成、ストリーム通信の性能改善を見るため、図 1のように、アクタとストリームを順次作成してゆくパイプライン処理の例題を作成し、従来の実行方式と Native 実行の時間の比較を行った。測定は SunOS 4.1.4 上で行った。

- アクタ生成数を 200 としたとき、従来の VM 方式で 11 秒かかっているのに対し、Native 実行では 1.5 秒に改善されている。

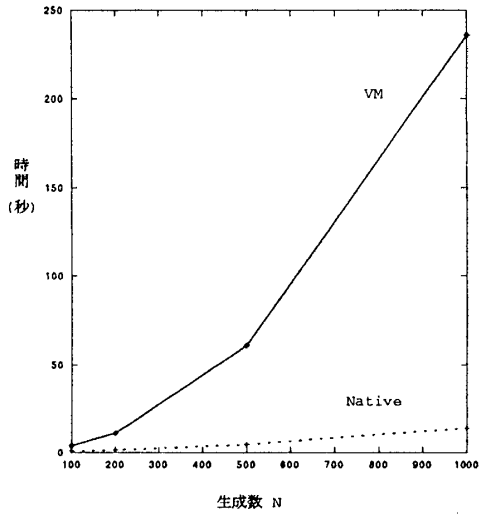

図 2: アクタ生成数と実行時間の関係

- Native 方式ではアクタ数と実行時間の関係がほぼ比例しているのに対し、VM 方式ではアクタ数を増やすに従って時間が大きく増している。これは、VM のスケジューリングにおいて優先順位を設定していないため、休止アクタに対してもコンテキストスイッチが均等に行われたため、と考えられる。

5 まとめ

NET/C 実行系の Native 化によって冗長性を排し効率を改善することができた。本方式では NET/C コードを元として Native コードを生成しており、C へ変換する前に VM によって NET/C コードの動作チェックが行われていることが前提となっている。今後、アクタ合成による効率化を行ってゆく。

参考文献

[1] Frank Mueller. Pthreads Library Interface. , 1993. July 22.

[2] 緒方和博, 栗原聡, 稲荷幹夫, 土居範久. MultithreadSmalltalk. コンピュータソフトウェア, July 1994. Vol.11 No.4.

[3] 青柳洋一, 中林嘉徳, 岩田竜一, 上原稔, 森秀樹. 並行言語 NET/C のマルチスレッド実装. 情報処理学会 第 51 回全国大会論文集, 1995.