プログラミング 8－24
（1996. 8. 29）

ストリーム通信オブジェクトのプラットフォームに応じた分散

青柳 洋一, 上原 稔, 森 秀樹
東洋大学工学部情報工学科

分散・共有メモリ型のマルチプロセッサやネットワーク接続された計算機など、多様な並列処理が利用されるようになってきた。並列処理の効率的な実現のためにはプラットフォーム毎にプロセッサや通信の速度に応じたオブジェクトの割り付けが必要である。並行言語NET/Cは簡潔な並行プログラミング実現を目指し、アクタによる並列性導出、ストリーム通信による制御の流れの明確化などを特徴としている。NET/Cは現在、シングルプロセッサにおいて動作しているが、言語の特性を引き出す並列環境における実装が求められている。本稿ではNET/Cの並列実装を行う上で必要な、マルチプロセッサにおけるアクタの配置決定アルゴリズムを提案する。

Distributed Allocations of Stream Communication Objects on Varous Systems

Yoichi Aoyagi, Minoru Uehara, Hideki Mori
Department of Information and Computer Sciences,
Toyo University

Recently, both various parallel architectures which have distributed/shared memory and networked computers become popular. In order to realize efficient parallel processing, it is required to allocate objects depending on processor and communication performance platform by platform. We have been developed concurrent language NET/C which aims simple concurrent programming, and features concurrency derivation by actor and simplification of control flows by stream communication. NET/C is running on a single-processor, but it is expected to implement NET/C on parallel environments. In this article, we propose algorithms for actor allocation which is required for parallel implementation of NET/C.

1 はじめに

並列処理が多様な環境で利用されるようになってきたが、並列処理はプログラムの複雑さ、システムの特性を生かした効率化と汎用性の両立の難しさなど、解決すべき問題を抱えている。効果的な並列処理の実現のためには、システムの機能を活用することが必要である。特定のプラットフォームに依存しないプログラミングの実現は、システム特有の機構(タスク生成、通信など)を直接用いず、上位レベルで共通インタフェースを用意することで可能だが、メモリ構成やプロセッサ性能、通信速度などをふまえたオブジェクトの割り付けが必要である。

並行言語NET/Cはアクタによる並列性導出、ストリーム通信による制御の流れの明確化、仮想マシン方式等を特徴としたデータフロー型の言語である。現在シングルプロセッサにおける並行動作のシミュレートで動作しているが、言語の特徴を生かすことのできる並列環境における実装が求められている。本研究では、NET/Cのマルチプロセッサへの実装に必要なアクタの配置決定アルゴリズムを提案する。2節で、NET/Cの概要を紹介し、3節で並列処理の効果的な実現について考察する。4節でアクタ配置決定のアルゴリズムを提案し、最後にまとめを述べる。

2 並行言語NET/C

NET/Cは簡潔な並行処理の記述を目指したデータフロー型の言語である。アクタとストリームの組み合わせにより、UNIXプロセスとパイプの連結処理のような簡潔な記述を可能にしている。アクタとストリームによる処理の流れはデータフローグラフで表現することが可能であり、各アクタの処理コスト、ストリーム流量は解析で比較的容易に求めることが可能である。図1に、NET/C Visual Editor[7]によって作成したグラフ表現の例を示す。図中の四角形はアクタ、矢印はストリーム、モニターの絵は標準出力を示している。図

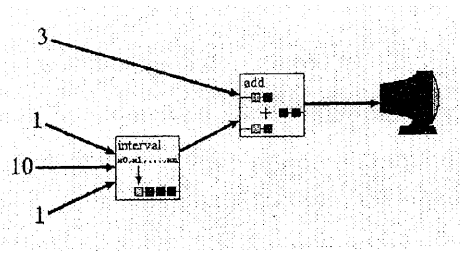

図1: NET/Cアクタとストリームを示すグラフ構造の例

中の数値“3”,“1”,“10”は無限に自身の値を出力し続ける定数アクタである。この例では、“3+interval(1,10,1)”の結果を出力する処理である。“interval”により1〜10の整数を生成し、“+”アクタで3を足して表示している。実行結果は“4 5 ... 13”となる。

NET/Cの特徴として以下のものがあげられる。

- アクタ
  - ストリームに対する繰り返し処理を行う並行実行の単位であり、プリミティブアクタや他のアクタの組み合わせにより簡潔に実現される
  - 値を返した後も実行を継続するreply文で並列性が導出される
- ストリーム通信
  - メッセージが連続したデータの流れとして扱われるため、メッセージの到着順が保証される
  - データに方向(入口と出口)を与え、通信対象を固定することにより、制御の流れの明確化がなされる
  - バッファリングにより非同期通信が容易に実現され、バッファサイズで通信の量子を調整できる
  - ストリームのバッファ管理を実行部で管理することで、プログラマの排他制御の負担を軽減する

- 仮想マシン (VM) 方式
  - コンパイラはプラットフォームに依存しないVMコードを出力し、コードの実行はVM実行部によって行われる
  - VM実行部による解析用情報の収集により、デバッグに有利な環境を提供できる

VMによるインタプリート実行は速度を要求される処理には不向きなため高速な実行手段が求められていた。また、NET/Cによる並列性導出の効果を有効にするために、並列環境における実装が望まれている。

2.1 NET/C実行系高速化の試み

実行部の高速化の試みとして、共有メモリ型マルチプロセッサOMRON Luna88k (2プロセッサ) のOS Mach2.5におけるマルチスレッド実装[6]、NET/Cコードを C言語に変換することによるマシンにNativeなコード生成[5]などを行ってきた。

共有メモリにおけるマルチスレッド化 マルチスレッドによる処理の実現により共有メモリを活用した高速な通信が可能なため、システムによる柔軟なスケジューリングに期待してアクタとスレッドの1対1対応を試みた。しかし、小アクタが多数生成されるような処理では十分な性能が得られなかった。NET/Cは、言語の特徴として細粒度のアクタが多数生成されるため、スレッド管理のコストや通信オーバヘッドが増大したのが原因であった。この問題への対処として、周期の等しいアクタ群を合成することにより通信やコンテキストスイッチのコストを削減する方法などが考えられる。

Nativeコード生成 Nativeコード化においては、NET/Cプリミティブに対応したCライブラリを用意し、NET/Cコードを Cにトランスレートして Nativeなコードを生成した。実行のNative化にあたり、汎用性と効率を重視し、POSIX 1003.4a準拠のpthreadライブラリを利用して並行処理を実現した。pthreadの利用は、共有データの活用によってデータコピーを抑えることができ、プロセスを用いた場合に比べて効率化を行いやすい。また、Mach Cスレッドに類似したインタフェースを備えるため、Machへの移行を行いやすいと考えこの方法を用いた。VM方式で行っていたスタック操作等VM特有の表現をCに適した表現に改め、解析用情報の管理を排して効率を追求し、8倍程度の高速化を実現した。

NET/Cの特徴を引き出すためには、並列環境における実装が必須であり、文献[6]のマルチスレッド実装の方式の改善、分散メモリにおける実装と評価が求められている。

3 マルチプロセッサにおける効果的なタスク割り当て

一般に利用されている並列処理環境として、分散メモリ／共有メモリ型のマルチプロセッサや、ネットワークに接続された計算機を利用したもの[1],[3]等がある。ここでは、分散メモリ型マルチプロセッサとしてAP1000[8], 共有メモリ型マルチプロセッサとしてOMRON Luna88kを用いることにした。ネットワーク接続された計算機による並列処理については、通信速度や計算機の性能のバラツキ等を考慮に入れる必要があるため、今回は対象から外した。

NET/Cで実現される処理は、実行がある程度進むと定常状態に移行し、パイプラインを形成するような処理に適している。図2はproducer-consumerの例であり、NET/Cアクタの定常状態への移行を示している。図中の円はアクタ、矢印はストリームを表している。

1. `main`アクタから`prod`、`cons`アクタを生成し、それぞれ互いを結ぶストリームの書き込み側、読み出し側を引数として受けとる

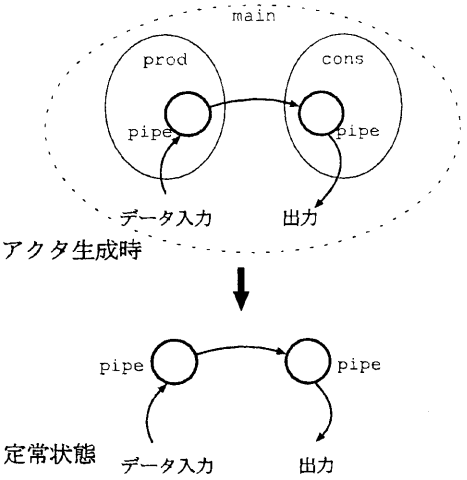

図 2: アクタの定常化

2. prod でpipe アクタを生成する。pipe は、ストリームの連結をするプリミティブアクタであり、ここでは引数で渡されたストリーム端とデータ入力ストリームの連結を行っている。cons 側も同様に、ストリーム端と出力ストリームをpipe で連結する

3. 実行が進むと pipe アクタ以外のアクタは消滅し、定常状態になる

NET/C においては、この定常状態で形成される処理が効率的に実行されることが重要である。

マルチプロセッサで効果的な並列処理を実現するためには、処理の種類に応じてプロセッサや通信の性能など多様な要因を考慮したタスクの割り当てが必要となる。スケジューリング方針として、プロセッサの状況に応じて方針を決定する動的な方法、コンパイル時に得られる解析情報などから決定する静的な方法 [2] がある。NET/C による処理は定常状態に落ち着くため、静的なスケジューリングが適している。本研究では、解析によって得られる情報 (各アクタの処理時間やストリーム流量など) からアクタ配置を決定する方法を提案する。

NET/C では、アクタ間の関係をグラフによって表すことができるため、グラフ理論を適用したスケジューリング・アルゴリズムが実現しやすいと考えられる。任意数のプロセッサに対するタスク割り付けを静的に行うスケジューリングのアルゴリズムとして、CP/MISF(Critical Path/Most Immediate Successors First) 法、DF/IHS(Depth First/Implicit Heulistic Search) 法 [4] 等が提案されている。CP/MISF 法は、タスクグラフを生成したときに、出口のノードからタスクへの最長パス (クリティカルパス) の長さからタスクの優先順位を決定するアルゴリズムである。クリティカルパス長は、それ以上プロセッサ数を増やしてもタスクの先行制約のために実行時間を短縮できない限界を示している。

NET/C におけるアクタは永続的なループ処理を行うため、このアルゴリズムをそのまま適用することはできないが、次節で紹介するアクタ合成にこのアルゴリズムの一部を応用して利用する。

4 アクタ配置のアルゴリズム

ここでは、アクタ合成による通信時間やコンテキストスイッチの負荷の削減を実現するアルゴリズム、分散メモリ型マルチプロセッサにおいて必要な、アクタのセルへの割り付け (グルーピング) のアルゴリズムについて提案する。

4.1 アクタ合成

アクタ合成は、同一周期で処理を行うアクタ群を 1 つのループにまとめることにより、通信コスト、コンテキストスイッチのコスト削減を図り、実行時間の短縮を行うことを目的としている。

先行制約のある処理においては、クリティカルパス上の大きな処理がボトルネックとなって、プロセッサの空きが生じる場合がある。図 3に NET/C のアクタグラフの適用例を示す。

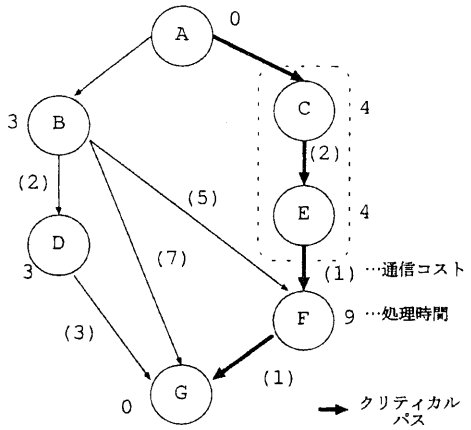

図 3: NET/C のアクタグラフ適用例

図 3において、

- 円 (ノード) はアクタを表し、ノード内の英字はアクタの種類、矢印はストリームと先行制約を示している。ノード A と G はダミーノードである。
- ノードの横に示した数字、( ) 内の数字はそれぞれ、グローバルなサイクルにおけるアクタの処理時間、ストリーム通信の時間を表している。NET/C は、それぞれのアクタが独自のループ処理を行い全体でパイプライン処理を形成するため、全体の処理が一巡する時間 (グローバルサイクル) に対する割合で各時間を示している。太矢印はクリティカルパス (最長パス) を示している。

図 3のクリティカルパス上のアクタ C,E,F について見ると、F の処理時間が他に比べ大きい。アクタ F の処理中は先行制約のために F の実行が終るまで他のアクタは実行を待たねばならない。図 4にアクタ C,E,F の実行可能なタイミングを示す。アクタ C と E の負荷を合計しても、その時間は F よりも短いため、アクタ合成による実行時間短縮が可能なことが分かる。以下をアクタ合成適用の基準とする。

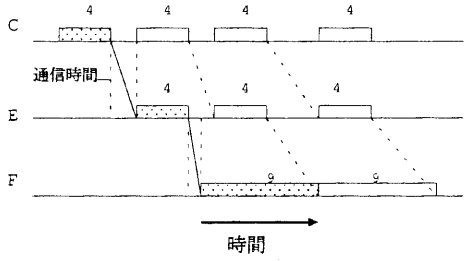

図 4: アクタの実行可能なタイミング

- ボトルネックとなるアクタと直前のアクタ群の処理時間の和を比較したとき、アクタ群の時間の和の方が小さいこと (合成後の通信時間は 0 と見ることができる)
- 一直線上にアクタとストリームが並び、他の先行制約を受けるストリームが結ばれていないこと

この他にも、グラフの末端に位置するアクタ群で、合成による通信時間のコスト削減により、実行時間の短縮が可能なことが自明な場合にも適用する。

NET/C は、ストリーム通信により通信対象が固定され、静的解析が行いやすいため、アクタ合成が実現しやすいと考えられる。

4.2 グルーピング

分散メモリ型マルチプロセッサにおいてはアクタを各セルに振り分けるアルゴリズムが必要である。振り分けにあたり、通信量の多いアクタはできるだけ同一セル内に納め、計算負荷が特定のセルに偏らずに分散されることが望ましい。ここでは、ナップザック問題のアルゴリズムを応用してアクタをセルに振り分ける方法を提案する。図 5において、

1. 振り分けられるアクタには、計算時間の重みと通信時間の重みづけがされている
2. セルで受ける計算負荷の制限を
   (アクタの負荷の総和)/(セル数)
   に少し余裕を持たせた値に設定する

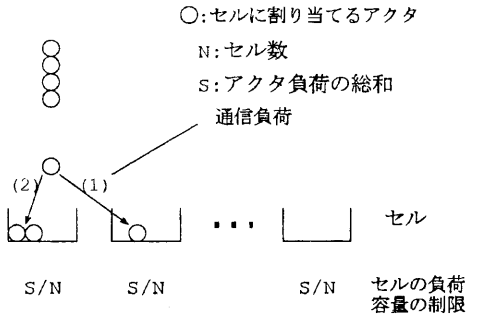

図 5: ナップサック問題によるアクタのセルへの振り分け

3. セルの制限容量を越えるまで、アクタをセルにつめてゆく。このとき、既に振り分けられているアクタとの通信の結び付きの強さで、アクタを入れるセルを判断する

4. セル間通信の総量が最も小さい組合せをアクタ配置とする

ナップザック問題は総当たりで解くと、組合せの数が膨大になる。遺伝的アルゴリズムを用いて最適解を見つけ出すことにより、より少ない試行回数で解を求めることが可能である [9]。

5 まとめ

NET/C をマルチプロセッサへ実装する上で必要なアクタ割り付けアルゴリズムを提案した。クリティカルパス上の先行制約のために生じる、プロセッサのアイドル時間を緩和し、通信オーバヘッドを削減する方法としてアクタ合成を提案した。分散メモリ型マルチプロセッサの任意台数のセルへアクタを割り付ける方法として、ナップザック問題の応用による方法を提案した。

今後の課題として、通信の準備に要するコストの大きい環境においては、通信所用時間はバッファリングの影響が大きい。このような場合について適切なバッファサイズの設定を行う方法について調査する。

参考文献

[1] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. PVM: Parallel Virtual Machinei - A User's Guide and Tutorial for Networked Parallel Computing. , 1994.

[2] Prabha Gopinath, Thomas Bihari, and Rajiv Gupta. Compiler Support for Object-Oriented Real-Time Software. *IEEE SOFTWARE*, September 1992. pp.45-50.

[3] MPI 日本語プロジェクト訳 MPI フォーラム. MPI:メッセージ通信インタフェース標準 (日本語ドラフト). , 1996.

[4] 笠原博徳, 成田誠之介. マルチプロセッサ・スケジューリング問題に対する実用的な最適及び近似アルゴリズム. 信学論 (D), July 1984. vol.J67-D,No.7,pp.792-799.

[5] 青柳洋一, 上原稔, 森秀樹. 並行言語 NET/C の Native 実行系の実装. 情報処理学会 第 52 回全国大会論文集, 1996. No.5,pp.67-68.

[6] 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. No.5,pp.41-42.

[7] 田中義一, 上原稔, 森秀樹. ビジュアルデバッガにおけるスケーリング手法の提案. 情報処理学会 第 52 回全国大会論文集, February 1996. No.5,pp.43-44.

[8] 富士通株式会社. AP1000 プログラム開発手引書 - C 言語インタフェース. , 1994.

[9] 北野宏明. 遺伝的アルゴリズム. 産業図書, 1993.