計算機アーキテクチャ 125−30
（1997. 8. 22）

受信予測によるメッセージ転送処理の高速化

岩本善行　澤田康雄　大津金光　吉永努　馬場敬信
宇都宮大学工学部

本稿では、メッセージパッシング処理を高速化するために、A-NETマルチコンピュータのローカルOSを部分的にマイクロプログラム化して実装した結果について述べる。この実装によって、最高で68.3%の実行時間を削減できた。さらに、受信メッセージの予測によるメッセージ転送の高速化法を考案し、実際にA-NETマルチコンピュータ上に実装した結果について述べる。これによって、1回の予測あたり80.7マシンサイクルの先行実行を行うことが可能となった。

High Performance Message Passing by Receiving Prediction

Yoshiyuki Iwamoto　Yasuo Sawada　Kanemitsu Ootsu
Tsutomu Yoshinaga　Takanobu Baba
Department of Information Science, Faculty of Enginneering
Utsunomiya University

This paper describes an evaluation of message passing performance of the A-NET multicomputer with converting its local OS to a micro-program in order to get higher performance message passing. This implementation reduces the execution time by 68.3%. And we propose a way to get higher message passing performance by receiving message prediction. A result of this implementation to A-NET multi-computer is also included in this paper. This implementation can anticipate approximately 80.7machine-cycles.

1.はじめに

オブジェクト指向では問題領域ごとに現われる機能や知識をオブジェクトとして素直に表現することにより、自然で分かりやすいプログラム記述を目指している。また、並列オブジェクト指向計算モデルはメッセージパッシングを基本としているため、並列処理をそのまま素直にモデル化して記述することができる。

我々の研究室では並列オブジェクト指向トータルアーキテクチャA-NETに関する研究を行っている[1]。A-NETでは並列オブジェクト指向言語A-NETL[2]を設計し、これを用いてOS[3]や応用プログラムの記述、さらにワークステーション上でネットワークワイドなシミュレーションを行ってきたが、現在、プロトタイプとして16ノードのA-NETマルチコンピュータを完成させた[4]。このプロトタイプ機上でメッセージ通信性能に関するさまざまな評価を行っている[5]。

同時に、我々の研究室では、商用並列計算機として富士通AP1000[6]とNEC Cenju-3[7]を取り上げ、各マシンの上でメッセージピンポン処理やリダクション処理などのベンチマークプログラムを作成し、そのメッセージ送受信性能の評価を行っている[8]。現在、それぞれの並列ライブラリ、OSのソースプログラムの提供を受け、メッセージ送受信処理部分の分析を進めている。

この過程で、メッセージ転送処理において、受信

側での受信処理に時間が多くかかること[9]や、アイドル時間の有効利用が必要であることが明らかになってきた。

これらを背景として、この研究では、第一に、受信処理OSのファームウェア化によって、より高速なメッセージ受信処理を実現した。第二に、これから到着するであろうメッセージをアイドル時間を利用してあらかじめ予測し、その結果に基づいて投機的な実行を行うことによって、高速化を試みた。特に後者の受信予測に関する研究として、現在、計算機アーキテクチャの分野では、条件分岐での分岐予測、あるいは投機的実行といった研究が盛んになされている[13,14]。

本稿では、最初にA-NETLのメッセージ転送機能とその実現方法について述べる。次に、A-NETL言語レベルからみたメッセージ通信性能について定量的に計測した結果と、メッセージ受信を行うOSの部分をファームウェア化した場合の動的実行時間を計測した結果を示す。最後に、受信メッセージ予測の具体的な方法とその評価について述べる。

2.A-NETLのメッセージ転送機能

2.1 メッセージ転送の概要

A-NETマルチコンピュータはメッセージパッシング型の並列計算機である。A-NETL言語では過去型／現在型／未来型のメッセージ送信と、現在型／未来型メッセージに対するリターンメッセージ送信命令を定義する。A-NETLプログラム内で各命令は明示的に記述する。

過去型メッセージ転送命令は、コンパイラによって機械命令sendPにコンパイルされる。このとき、コンパイル時に得られたオブジェクトID、セレクタIDがオペランドとなる。この機械命令が実行されると、PE(Processing Element)のマイクロプログラムは、メッセージを転送するために必要な受信側のオブジェクト／セレクタID、引数等を含んだメッセージを生成する（図1）。これをルータとの共有メモリ領域に書き込み、ルータに通知後、次の機械命令の実行に移る。ルータはこの通知を受け、転送先オブジェクトに向けて、最短距離となるような隣接ノードのルータにメッセージを送信する。

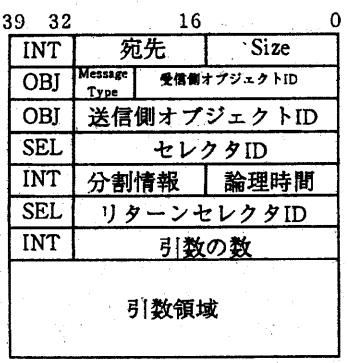

図1　メッセージ構造

受信側となったノードのルータは、このメッセージを受け取ると、マイクロプログラムに対して割り込みフラグをセットする。これを認識したマイクロプログラムは、あらかじめ設定されたOSのメッセージ処理プログラムを起動する。このOS内では、図2に示すように、メッセージを置くための領域確保やルータとのやりとりを行ったり、メッセージを解読し、指定されたユーザメソッドを起動する。

現在型メッセージ送信(sendN)、未来型メッセージ送信(sendF)においても、過去型と同様なシーケンスでメッセージを送出する。ただし、現在型メッセージ送信では命令実行直後にフューチャートラップを発生し、返値メッセージが到着するまで他メソッドを実行するか、キューに起動待ちのメソッドがない場合はアイドルとなる。未来型メッセージ送信では、返値が必要になった時にフューチャートラップが発生し、OSに処理が移行する。

リターンメッセージ送信においても、過去型メッセージ送信と同様のシーケンスであるが、メッセージ構造のリターンセレクタIDが不要となるため、メッセージサイズは1ワード少なくなるという違いがある。

2.2 OSでのメッセージ受信処理

ルータは、他ノードからのメッセージを受信すると、PEとの共有メモリに、到着したメッセージのサイズを書き込むことによって、ハードウェア的にメッセージ到着フラグをセットし、マイクロプログラムに通知する。

マイクロプログラムは機械命令境界、または、アイドル状態にあるときに、このフラグをチェックし、メッセージが到着している場合にはメッセージ受信処理のためのOSを起動し、メッセージ受信処理を開始する。

OSでは図2に示すように、

a) 到着したメッセージに対する処理

b) メッセージの認識
c) メソッドを起動するための処理
の3段階で、メッセージを処理する。

3.メッセージ受信のファームウェア化

3.1 方針

これまでは、メッセージ到着後の各処理をA-NETLで記述し、機械命令にコンパイルされたOSによって処理を行っていた。A-NETの機械命令セットは、A-NETL指向の高機能命令セットであり、コンテクストスイッチ時のレジスタ退避処理を削減することによる高速化と、オンチップメモリを前提とした、メモリーメモリ演算を行っている。また、動的データの型付けをサポートしているため、各機械命令を実行する場合にはタグチェックを行っている。このことによって、ユーザレベルでの記述性は高くなるが、OSレベルではメッセージ転送のオーバーヘッドとなっている。現在のA-NETマルチコンピュータでは、プロトタイプマシンとして様々な変更に柔軟に対応できるように、ファームウェアによってその機能を実現している。ここでは、実験的に、OSをファームウェア化することによって、高速化を試みた。このことによって、以下のような点の高速化が可能となる。

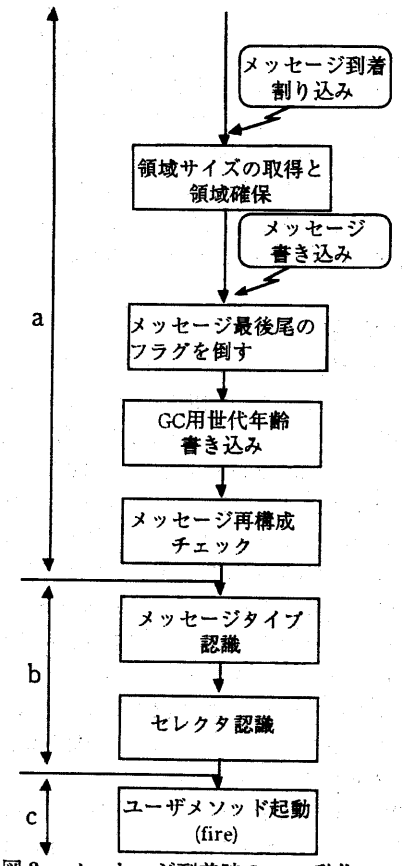

図2　メッセージ到着時のOSの動作

第1に、OSでのアドレス演算やメッセージ認識などで一時的に使用される変数などに対して、メモリーメモリ演算を行わず、マイクロプログラムレベルで使用可能なレジスタを使用することによって、メモリアクセスのオーバーヘッドを削減することが可能となる。

第2に、OSで扱うデータは多くの場合あらかじめ静的に型が決定可能であり、動的に型のチェックを行う必要はない。

第3に、A-NETプロトタイプマシンのマイクロアーキテクチャは、ALU演算、タグ操作、メモリアクセス、シーケンス制御などを同時に制御できる水平型マイクロ構成となっているが、これをA-NETLレベルで生かすことはできない。このため、例えば、ワードに対するビットフィールド操作は機械命令で1命令20MC程度かかるのに対し、マイクロプログラムレベルでは1MCで可能となる。

これらの理由によって、メッセージ処理の高速化が可能となる。具体的には、今回、

1. フューチャートラップ後の処理
2. 領域サイズの取得と領域確保
3. メッセージに対する操作
4. メソッド起動

などの処理をマイクロプログラム化することによって、高速化を試みた。

3.2 評価

上記の変更を行ったOSと、これまでのOSに対して、簡単なサンプルプログラムを作成し、動的な評価を行った。この結果を表1に示す。

過去型メッセージに対しては、送信／受信ともOSは起動しないため、これまでの実行マシンサイクル

表1　メッセージ送信処理時間　(単位MC)

メッセージ種類	機械命令実行時間	アイドルまで	コンテクスト起動時間
過去型	65		
現在型／未来型			
マイクロ化前	65	687	2,582
マイクロ化後	65	420	818

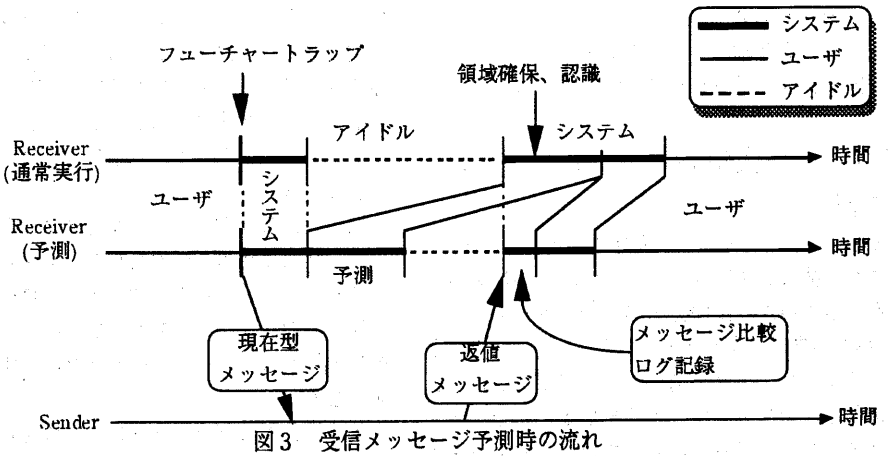

図3　受信メッセージ予測時の流れ

(MC)数と等しくなっている。

現在／未来型のメッセージ送信命令が実行されたあと、OSの処理に移るまでの65MCはこれまでと同じであるが、フューチャートラップが発生してOSが起動してから、後処理が終了しアイドル状態になるまでの時間が、これまでの687MCから420MCとなり、38.9%削減された。また、リターンメッセージ到着後、ユーザコンテクストが起動するまでの2,582MCが818MCとなり、68.3%削減された。これは、先に挙げた理由の中で、マイクロプログラム内でレジスタを使用することが可能となったことが最大の理由と考えられる。

4.受信メッセージ予測

4.1 方針

通常のメッセージ通信時の流れを図3の最上段に、受信メッセージ予測を導入したときの流れを図3の中段に示す。通常実行時には一般的に、(1)ユーザプログラムが終了(terminate)した場合と、(2)他ノードへの現在型メッセージを送信しフューチャートラップが発生した場合とに、システム(OS)に制御が移り、その後処理を行う。この処理の終了後、次に実行されるべきコンテクストがない場合、アイドル状態(メッセージ待ち状態)となる。他ノードからメッセージが到着すると、2.2節で述べたように、そのメッセージに対する領域確保を行い、起動すべきメソッドを認識後、実際にユーザメソッドが実行される。

富士通AP1000の場合でも、l_arecv命令を使用したときに、この命令の引数としてタスクIDなどを指定し、そのメッセージが到着していない場合は、メッセージが到着するまでアイドルと同様な状態となる。

メッセージ予測では、これらのアイドル状態を利用し、これから送られてくるであろうメッセージを過去の履歴からあらかじめ予測し、実行可能な部分を投機的に実行することによって、予測ヒット時に高速化することを目的にしている。予測方式としては、

1) 前回のメッセージのみを参照する
2) 過去のメッセージの平均や発生頻度を参照
3) メッセージ発生系列のマルコフ連鎖
4) 前回の実行が完結したときのトレース

などによる手法が考えられる。1)、2)、3)、4)の順で実装が複雑になり、予測に必要となる時間もかかるが、的中率も上がることが期待される。

4.2 実装

この受信メッセージ予測の効果を調べるための第1ステップとして、A-NETマルチコンピュータ上のA-NET ローカルOSに対して、1)の予測方式を実装した。このOSは、3章で高速化したOSである。

A-NETマルチコンピュータにおいて、PE起動後に受信するメッセージの種類とその流れを、図4に示す。A-NETマルチコンピュータでは、最初に、A-NETローカルOSを読み込むためのメッセージを受信する。このメッセージに対して読み込み可能である返答を返し、ホストから全PEに同一のOSが送られて、OSをシステム領域に読み込む。読み込み終了後、マイクロプログラムによって自動的にOSを起動し、初期化などの処理を行う。

次に、OS起動終了のメッセージをホストに返すことによってユーザオブジェクト転送可能となり、ホストのコマンドラインに対して、ユーザがself load命

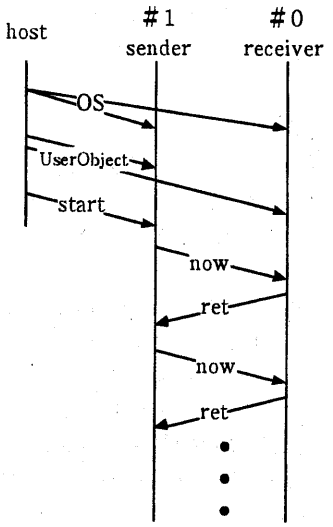

図4　A-NETマルチコンピュータにおける受信メッセージの種類

令を発行することによって、各ノードに対してホストからユーザオブジェクトを順次送る。この後、さらにユーザがホストから起動メッセージを発行して、ユーザメソッドが起動され、各ノード同士でメッセージのやり取りを行いながらユーザプログラムが実行される。

A-NETマルチコンピュータでのこれらの一連の流れに対して、メッセージサイズなどのログを記録するが、OS起動後にノードが受信する最初の2つのメッセージ(ユーザオブジェクト要求、ユーザによる起動)は、ユーザオブジェクト自身のサイズが広範囲で可変であることや、過去のログが少ないことなどから、予測が不可能であるため、受信回数をカウントするのみとする。

3回目以降のメッセージに対しては、図1に示すメッセージの中で、前回のメッセージを参照して予測するという方式を実現するために、到着した各メッセージのメッセージサイズ(size)とセレクタIDを記録する。この2項目のみを記録するのは、OS内の解析の結果、メッセージにおけるこのほかの項目に対して予測した場合に可能となる先行実行処理はほとんど存在せず、よって高速化の効果が得られないとの理由である。

この上記2項目によって期待できる効果は、メッセージサイズを予測することによって、メッセージ受信後の処理のうち、メッセージを読み込むための領域をあらかじめ確保することと、受信メッセージの引数へのポインタをあらかじめ設定することが可能となる。さらに、セレクタIDを予測することによって、これから起動すべきメソッドで必要となる一時領域の確保などが可能となる。この2つの項目に対して順次実装し、そのときの評価を取得した。

4.3 評価

受信メッセージ予測の効果を調べるため、A-NETLで簡単なベンチマークプログラムを記述し、そのプログラムを実際にプロトタイプマシン上で実行することによって、動的な評価を得る。

この評価で使用したプログラムは、図4に示す2ノードを使用し、一つのノード(sender)が現在型(now)のメッセージを送信し、これに対してもう一つのノード(receiver)が整数値を持ったリターンメッセージ(ret)を返すものである。このときのメッセージサイズは、sender側から送出される最小のメッセージサイズとなり7語(35バイト)、receiver側では8語(40バイト)となる。また、A-NETマルチコンピュータは30MHzのクロックで動作し、1マシンサイクル（マイクロプログラムを1つ実行）は5フェーズからなる。すなわち、1マシンサイクルは167ns(6MHz)で動作する。

ベンチマークは、10回のメッセージピンポンを行う。それぞれのノードは、アイドル状態になると、次に来るであろうメッセージを予測し、4.2節で示した領域確保やポインタ設定を投機的に実行する。このベンチマークでは、各ノードにのるユーザメソッドは1種類であり、また、各ノードが受信するメッセージも1種類であるため、予測は最初のユーザメッセージでは失敗、以降は必ず成功するため、それぞれの評価を取ることが可能である。

この結果、表2に示すように、予測によって縮小された時間は、メッセージサイズのみを予測した場合で18.9MC、セレクタIDまで予測して、OSで実行可能な部分をすべて行った場合80.7MCの先行実行が可能となった。A-NETマルチコンピュータではデー

表2　受信メッセージ予測による削減　(単位MC)

	予測導入前	サイズのみ	セレクタIDまで
予測実行時間	0	18.9	80.7
オーバーヘッド			
ログ取得	0	—	12
予測失敗	0	20	51
予測成功	0	15.0	46.0

タ移動を行うMV機械命令を8サイクル程度で実行するため、この先行実行により10機械命令程度が先行実行されることになる。

一方、受信予測を行うためのオーバーヘッドとして、予測に使用するためのログ取得に12MC必要である。更に、予測成功した場合でのオーバーヘッドとして、実際に受信したメッセージと予測項目との比較時間は、サイズのみを予測して実際のメッセージと比較した場合で15MC、予測可能なすべての項目を比較した場合で46MCとなる。予測に失敗した場合のオーバーヘッドとしては、各項目を比較後、予測を破棄するまでに、サイズのみの場合で20MC、全項目の予測時で51MCが必要である。

10回のメッセージで総合すると、サイズのみを予測した場合で、

$12\times10+20+15.0\times9-18.9\times10=86$

となり、実行時間が増加するが、セレクタIDまで予測した場合では、

$12\times10+51+46.0\times9-80.7\times10=-222$

で222MCの高速化が実現できる。

5.おわりに

本稿ではA-NETマルチコンピュータのローカルOSをマイクロプログラム化することによって、メッセージ転送性能を高速化し、その結果を定量的に評価した。さらに、受信メッセージ予測の手法を提案し、A-NETマルチコンピュータに実装した結果から高速化の有効性を確認した。

今後、受信メッセージ予測方法に対しては、マルコフ連鎖を利用した予測などに変更したときの効果の確認、次のメッセージが到着するまでのアイドル時間を予測することによって受信メッセージ予測自体を行うかどうかを決定するような方法の実装などを行う予定である。さらに、AP1000やCenju-3など他アーキテクチャへの実装とその評価を行う予定である。また、予測した項目に対して、OSの処理のみならず、ユーザプログラムの先行実行を行うための手法についても検討したい。

謝辞

本研究は、一部文部省科学研究費（基盤(c)課題番号08680346,09680324,奨励(A)09780237），電気通信普及財団、並列・分散処理研究推進機構の援助による。また、OSのソースプログラムを提供いただいた富士通並列処理研究センターおよび、NEC C&C研究所 並列処理センターに感謝する。

参考文献

[1] 吉永努馬場敬信:並列オブジェクト指向トータルアーキテクチャA-NET －マルチコンピュータ開発と言語実装の現状－,情報処理学会第52回全国大会,6-97(1996).

[2] T.Baba and T.Yoshinaga:A-NETL:A Language for Massively Parallel Object-Oriented Computing , Proc. 1995 Programming Models for Massively Parallel Computers, pp.98-105 , Oct. 9-12 (1995).

[3] 田口弘史,吉永努,馬場敬信:A-NETローカルOS第3版－メッセージ受理動作の高速化とその評価－,コンピュータシステムシンポジウム , pp.51-58, 1993 .10 (1994).

[4] 吉永努,馬場敬信:並列オブジェクト指向トータルアーキテクチャA-NETのノードプロセッサ,電子情報通信学会論文誌 , Vol.J79-D-1 No.2 pp.60-68 (1996).

[5] 澤田東, 阿部大輝, 広田守, 岩本善行, 吉永努, 馬場敬信: A-NETマルチコンピュータの通信性能,JSPP'97, pp13-20 (1997).

[6] D.Sitsky, K.Hayashi: Implementing MPI for the Fujitsu AP1000/AP1000+ using Polling, Interrupts and Remote Copying, JSPP'96, pp.177-184(1996).

[7] T.Hirose, T.Hosomi, T.Maruyama and Y.Kanoh: The inter-processor communication of Cenju-3 and it's evaluation, JSPP'95, pp241-248(1995).

[8] 岩本善行,吉永努,馬場敬信: 言語レベルからみたA-NETマルチコンピュータのメッセージパッシング性能, 情報処理学会研究報告(SWoPP '96), 96-ARC-119, pp.1-6(1996).

[9] Y.Iwamoto, K.Ooguri, T.Yoshinaga and T.Baba: A Comparison of Communication Performance in the NEC Cenju-3 and FUJITSU AP1000, Proc. of the FIRST CENJU WORKSHOP, pp60-64(1997).

[10] D.Joseph and D.Grunwald: Prefetching using Markov Predictors, Proceedings of the International Symposium on Computer Architecture (ISCA'97), June (1997).

[11] 児島彰, 弘中哲夫, 高山毅, 藤野清次: 複数分岐での投機的実行の有効性, 情報処理学会研究報告,97-ARC-123-10, pp55-60(1997).