計算機アーキテクチャ 119－1
（1996．8．27）

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

岩本善行　吉永努　馬場敬信

宇都宮大学工学部

我々の研究室では並列オブジェクト指向計算モデルに基づいてA-NETLを設計し、さまざまなソフトウェアシミュレーションを行ってきた。同時に、この言語を実現するための専用機としてA-NET計算機を設計し、プロトタイプとしてのハードウェアを完成させた。本稿では、この製作されたプロトタイプA-NETマシンを動作させて実際に得られたOSレベルのメッセージ通信性能の定量的な評価結果と、それをふまえた高速化のための改善方針について述べる。

Message passing performance of the A-NET multicomputer

Yoshiyuki Iwamoto　Tsutomu Yoshinaga　Takanobu Baba

Department of Information Science
Utsunomiya University

This paper describes the evaluation of message passing performance of the A-NET multicomputer . We have designed the A-NETL based on a parallel object-oriented computation model. Also we have designed and developed a prototype of the A-NET machine. Using the prototype machine, we evaluated the message passing performance. Based on the experimental results, a suggestions to get higher performance are also included in this paper.

1.はじめに

オブジェクト指向では問題領域に現われるオブジェクトの持つ機能や知識を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][3]を設計し、これを用いてOS[4][5]や応用プログラムの記述、および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計算機を完成させた。現在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ところである。

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たあと、上記のプロトタイプA-NET上で得られた動的評価をもとに、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述べ、それを踏まえて高速化のための改善方針について述べる。

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計算機はメッセージパッシング型の並列計算機である。A-NETL言語では過去型メッセージ送信、現在型メッセージ送信、未来型メッセージ送信、および現在型／未来型メッセージ送信に対するリターンメッセージ送信命令を定義する。A-NETLプログラム内でこの各命令は図1のように明示的に表記する。ここで、Objectは転送先オブジェクト名、Methodは起動するメソッドの名前を表わす。

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、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

過去型	Object Method:Argument
現在型	Val=Object Method:Argument
未来型	Val=?Object Method:Argumet

図1.A-NETLのメッセージ転送命令形式

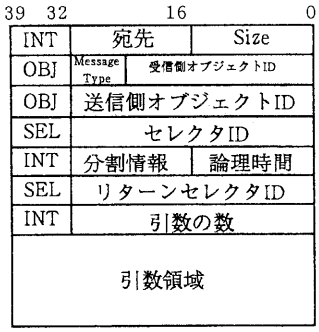

図2　メッセージ構造

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転送先オブジェクト／セレクタID、引数等を含んだメッセージパケットを生成する（図2）。メッセージの一部は1語をビットフィールドに分割して割り当てられる。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パケットを送信する。

受信側となったノードのルータはメッセージパケットを受け取ると、マイクロプログラムに対して割り込みを発生し、マイクロプログラムはあらかじめ設定されたOSの処理プログラムを起動するためにプログラムカウンタを設定し、メッセージ受信のためのOSの機械命令を実行する。OSはメッセージを解読し、指定されたメソッドを起動する（図3）。

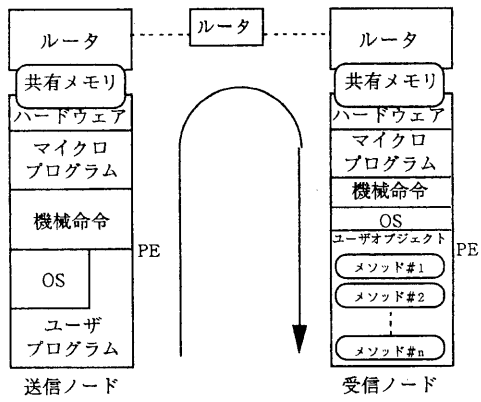

図3　メッセージ転送シーケンス

現在型メッセージ送信(sendN)、未来型メッセージ送信(sendF)でも同様なシーケンスでメッセージを送信する。ただし、現在型メッセージ送信では命令実行直後、未来型メッセージ送信では返値が必要になった時に、それぞれフューチャートラップが発生し、OSに処理が移る。

2.2 OSのメッセージ関連処理

2.2.1 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に対してハードウェア的にメッセージ到着割り込みを発生する。これを受けてPEではOSを起動し、図4に示した順序で処理を行う。以下に流れに沿って簡単に説明する。

a) メッセージが到着してからメッセージがメモリに書き込まれるまで

共有領域から到着メッセージサイズを読み込み、必要なサイズを現在領域に確保し、その先頭アドレスをルータに通知する。

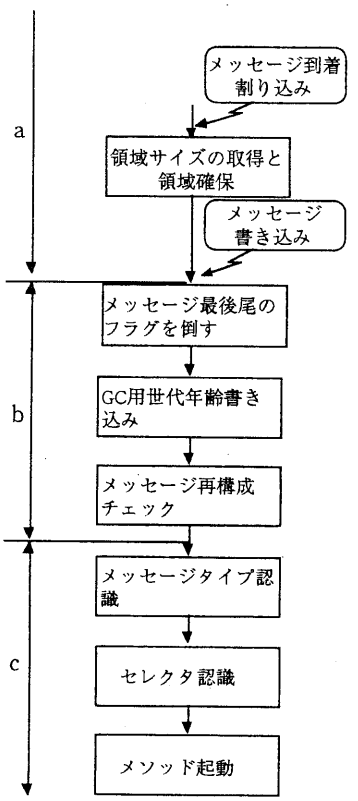

図4　メッセージ到着時のOSの動作

ルータではPEからメモリのバスを奪い、確保された先頭領域から受信したデータを順次書き込む。このためルータがメッセージを書き込んでいる間はPEがロックされた状態となる。書き込み終了後、ルータが使用していたPEのバスを解放し、共有メモリのフラグを倒すことによって、PEは書き込みが終了したことを認知する。この一連の処理はルータとのやり取りのための不可欠な処理である。

b)メッセージパケットに対する処理

OSはルータがメッセージの最後尾を識別するために使用したフラグ（フューチャーフラグと共用）をクリアし、GC用の年齢を設定する。さらにメッセージ分割されているときはその再構成を行う。

c) メソッドを起動するための処理

送られてきたメッセージのタイプを以下の3種類に分けて識別する。

1.リターンメッセージ

2.動的オブジェクトの起動／消去

3.ユーザオブジェクトの起動

ここで、ユーザオブジェクトの起動と認識した場合には、そのメソッドを検索し、実行する。

2.2.2 送信命令のフューチャートラップ処理

現在型、未来型のメッセージ送信を行うと、フューチャートラップ割り込みを発生し、OSに処理を移行する。

OSは、現在実行中のユーザプログラムのプログラムカウンタ、各種レジスタなどの環境をサスペンデッドコンテクストとしてサスペンデッドリストに退避し、フューチャートラップの原因となった変数のアドレスをリターン情報として登録する。その後、実行可能状態となっている他のオブジェクトがないかレディリストを検索し、あった場合そのオブジェクトを実行する。実行可能状態のオブジェクトがない場合は新たなメッセージが到着するまでアイドル状態となる（図5）。

返値がリターンメッセージとして到着すると、領域確保等通常メッセージと同様の処理を行った後、リターンメッセージとしてその引数である返値を先のリターン情報に設定し、コンテクストが復帰するのに必要な変数が全部揃ったらサスペンデッドリストからレディリストにコンテクストをスケジューリングして実行の順番を待つ。

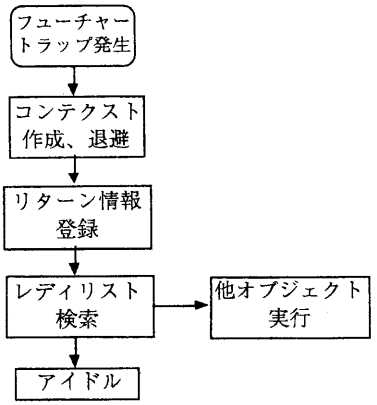

図5　フューチャートラップの処理

3.メッセージ転送性能評価

以下ではA-NETLで簡単な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した結果得た、動的な実行時間を過去型／現在型／未来型／リターンメッセージ／メッセージ受信の順に述べる。

各評価で使用したプログラムは2ノード間で引数のないメッセージを送受信するものである。このときのパケットサイズは最小となり7語(35バイト)となる。また、A-NET計算機は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(MC)は167ns(6MHz)で動作する。

3.1 過去型メッセージ送信処理

A-NETLで記述された過去型メッセージ送信はコンパイラによって機械命令に翻訳される。この命令の実行はメッセージパケットを作成し、ルータに通知するだけである。この実行にOSは介在せず、この機械命令を実現するマイクロプログラムによって、実行される。

このとき機械命令を実行するのに要した時間は約65MC(=10.8μs)である（表1）。

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	65	687	2,582
未来型	65	687	2,582
リターン	70	-	-

3.2 現在／未来型メッセージ送信

現在型メッセージは機械命令終了直後にフューチャートラップが発生し、OSに移行する。また、未来型メッセージは、メッセージ送信後、次の命令の実行に移り、返値を格納する変数が使用された時点で返値が格納されてないとフューチャートラップが発生する。現在型、未来型ともフューチャートラップが発生する時点が異なるだけで、OSの処理は同一となる。

メッセージを送信するまでは過去型メッセージ送信と同じく65MC程度だが、フューチャートラップ発生後OSの処理が行われ、アイドル状態になるまでに687MC、返値メッセージを受信後ユーザコンテクストが復帰するまでに2,582MC必要とする。

3.3 リターンメッセージ送信処理

過去型メッセージ送信と同様にOSは介在せず、機械命令のみで実行される。

このとき必要となる時間は70MC程度である。

3.4 メッセージ受信処理

メッセージ受信時に行われるOSの動作に必要な実行時間は、2.2.1の流れをもとにすると以下のようになる。

a)メッセージ到着から書き込まれるまで
226MC

b)メッセージパケットに対する処理
222MC

c)メソッドを起動するための処理
326MC

各機能ブロックごとにほぼ近い値となっているが、この中のほとんどの時間をデータ移動命令やaddなどの単純な演算命令が占めている。

4．高速化のための改善方針

これまでに述べてきた性能評価をもとに、メッセージ通信性能をより高速化するための改善の可能性を示す。

4.1 機械命令レベルアーキテクチャ

A-NET OSはA-NETLで記述されており、コンパイラによって機械命令に変換される。この時使用される機械命令セットはユーザがA-NETLで記述しコンパイルしたものと同一の命令セットである。機械命令定義は、A-NETLと1対1に対応しており、メモリ－メモリ演算、動的データ型付けをサポートしているが、OSの高速化のためにはかえって足枷となっている。

このことを改善するために次に述べるような2項目の対策を検討した。

4.1.1 メモリーメモリ演算

機械命令のオペランドはすべてメモリを対象としたものであり、PEがマイクロレベルで使用するために用意されているレジスタ群は使用できない。このため、必要なオペランドやデータを取得するためには機械命令ごとにそのつどメモリアクセスを行う必要がある。また、このとき必要とする実行時間は、メモリからの読み出しが2MC、書き込みが1MCであるのに対して、レジスタアクセスは共に1MCで終了することができる。

近年集積回路の技術の向上により、かなり大規模な回路でもVLSIとして1チップ化が可能となっている。A-NET PEもメモリと共に1チップ化し、より高速化をはかることが当初からの目標である。1チップ化することにより、このメモリをレジスタ操作程度に高速化することが可能と考えられる。

また、メモリから読み出されたデータは機械命令境界ごとにメモリに戻されるが、OSで使用される変数は機能ブロックごとの局所性が強いため、連続した機械命令で使用される場合には、メモリアクセスのオーバヘッドとなる。OSが実行されるシステムモードにのみレジスタを使用することを許し、連続使用されるデータを機械命令境界を超えてレジスタに保存することにより、高速化を図ることができる。例として、領域確保を取り上げると、現在139MC必要としているが、これを上記のような変更を加えた場合、50MCで実行可能となり、約65%の実行時間を削減することになる。

さらに、メッセージ最後尾のフラグを操作するために必要とされた実行時間93MCに対しては、9MCで実行が可能であり、90%以上の時間を削減することが可能である（図6）。

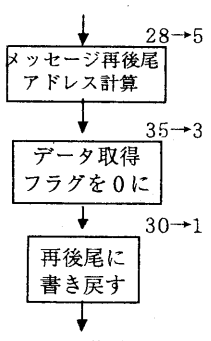

図6 メッセージ再後尾のフラグに対する操作

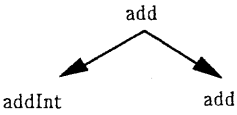

図7 データ型に応じた命令セットの特化

4.1.2 データ型チェックの削除

A-NETアーキテクチャの特徴として、動的データの型付けのサポートが挙げられる。

機械命令のオペランドとして指定されたデータに対して、機械命令がその型によって適切な処理を行う。機械命令はA-NETで定義されるデータ型すべてに対して、適切な処理やエラーとしての認識を行わなければならない。このデータ型処理のオーバヘッドを削減するためハードウェアとしてタグ処理ユニット(TPU)を用意し、その効果も確認されている[6]。

一方、コンパイル技術の発展により静的なデータ型推論の有効性も主張され、我々の研究室ではコンパイラによるデータ型推論の研究も進み、ある程度の見通しも持っている[7]。特に、OSレベルで使用されるデータを調べてみるとほとんどの場合あらかじめデータ型は静的に決まっている。例として、他ノードから送られてくるメッセージのサイズは、整数型であり、ルータもそのことを前提として処理を行っている。このように、データ型が既知である変数をOSレベルにおいて使用するために、全てのデータ型に対応するようなジェネリックな命令を使用することによる、データ型処理のオーバヘッドが問題となる。これを静的データ型つけに対応した命令セットに特化（図7）するように命令セットを用意することによって、このオーバヘッドを削減することが可能となる。

以上のような2項目の改善を行った結果、従来1,558MC必要であった実行時間は、488MCとなる。このような高速化を実現するためには(i)ローカルOSを機械命令を使用せずにファームウェア化する、(ii)機械命令セットをデータ型に対して特化し、レジスタ－レジスタ演算を活用できるようにするなどその機能レベルを低くする、ことが考えられる。

4.2 通信モデルの変更

現在のA-NETの通信モデルはメッセージによってオブジェクトIDとメソッドIDを指定し、OSがそのIDをもとに実行すべきプログラムカウンタ

を辞書から求めていた。これをメッセージ中にプログラムカウンタを埋め込み、直接PCにロードして実行するようなアクティブメッセージモデルとして、以下の2種類の場合を考える。

a)アクティブメッセージのみを実装した場合

この場合、図4のメッセージ到着時のOSの動作において、メッセージタイプ認識(86MC)、セレクタ認識(90MC)、メソッド起動の一部(443MC)をユーザが行うことになり、OSの動作は軽減される。このとき、実行時間は939MCとなり、従来の実行時間(1558MC)に対して約40%の時間が削減される。

b)実行モデルを極小まで削減した場合

現在のプロトタイプA-NET計算機で考えられる最小の処理として、領域を確保し、読み込み、即座に実行した場合を考える（図8）。

図の中で各処理ブロックの右上に示した数字はOSとして機械命令を実行した場合に必要となるMC数で、括弧内は4.1節の変更を加えた場合（ファームウェア化した場合）のMC数である。

この場合、先にOSの処理として示したメッセージの再構成やタイプの認識などは、すべてユーザが行うことになる。また、レディリストが存在しないため、それまでに実行していたコンテクストの退避や複数のオブジェクトの実行待ちなどができず、メッセージが到着した時にはすでにPEがアイドル状態となっていなければならない。このためPEがアイドル状態であることを検出できるようにする、あるいはルータに着信メッセージのバッファを持たせる、などの新たなアーキテクチャサポートが必要となってくる。

このモデルにした場合のOSの実行時間は図より268MC(17%)であり、さらにファームウェア化した場合は60MC(3.8%)程度まで削減を実現することが可能となる。

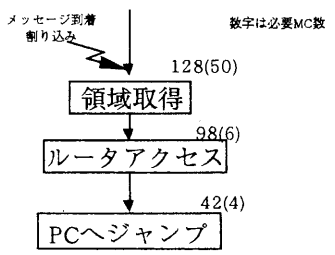

図8　最小処理の場合

5.おわりに

本稿ではA-NET計算機のメッセージ転送性能を定量的に評価し、さらに高速化のための方法をアーキテクチャレベルから再検討し示した。ローカルOSのファームウェア化、命令セットをデータ型に対して特化したりレジスタ－レジスタ演算を活用したりしてその機能レベルを低くすること、アクティブメッセージの採用による処理の簡略化、などがOSの高速化に有効であることが確認された。今後の課題として、OSを順次マイクロ化することによって、A-NET計算機の高速化を行った場合の性能を評価する予定である。

謝辞

本研究は、一部文部省科学研究費（基盤(c)課題番号08680346）,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。

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,T.Yoshinaga and T.Furuta: Programming and Debugging for Massive Parallelism:The Case for a Parallel Object-Oriented Language A-NETL , Proc OBPDC'95 (to be published from Springer) ,(1995).

[3] 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).

[4] T.Yoshinaga and T.Baba:A Local Operating System for the A-NET Parallel Object-Oriented Computer , Journal of Information Processing , Vol.14 No.4(1991).

[5] 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).

[6] 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).

[7] 古田貴寛,齊藤宣人,月川淳,吉永努,馬場敬信:並列オブジェクト指向言語A-NETLのワークステーションクラスタへの実装, SWoPP'96,pro8-25発表予定,(1996)