受信メッセージ予測法における予測方式の検討

岩本善行[†] 大津金光[††]
吉永努[††] 馬場敬信[††]

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これまで提案してきた受信メッセージ予測法について，具体的な予測手法を提案・検討する．受信メッセージ予測法とは，並列計算機において，ノードプロセッサがアイドルとなったときに，その時間を利用して次に到着するメッセージを予測し，受信処理などを投機的に先行実行することによって高速化を図る手法である．この手法には，到着するメッセージを予測することが不可欠であり，その予測のための手法を具体的に 6 種類提案している．また，A-NET 計算機上では 3 種類のプログラム，MPI 環境では NAS Parallel Benchmark プログラムを用い，各プログラムを実行したときのメッセージ受信履歴を用いて，それぞれの予測手法を用いた場合の予測成功率を求めた．同時に，A-NET 環境では予測に必要となる時間などの評価も行った．それらの結果を示して考察する．

Prediction Methodologies for Receiving Message Prediction

Yoshiyuki Iwamoto,[†] Kanemitsu Ootsu,[††]
Tsutomu Yoshinaga[††] and Takanobu Baba[††]

This paper proposes and examines receiving message prediction for high performance message passing. In the receiving message prediction, when a node is idle, it predicts a message it will receive next, and speculatively executes the process of message reception. We propose six methods for message prediction. Using the execution trace of three programs on the A-NET multi-computer and the NAS Parallel Benchmark programs on the MPI, we calculate the success ratio for prediction using these methods. Results and discussion are also included in this paper.

1. はじめに

ノード間通信のオーバヘッドは逐次型計算機では考えられない問題であり，並列処理に固有の重要な課題である．これまでさまざまな並列計算機アーキテクチャや並列処理のための言語/ライブラリが提案されているが，この通信処理をいかに高速かつ容易に行うかということが，並列処理の性能向上と普及には重要である．

我々は，商用並列計算機として富士通 AP1000 と NEC Cenju-3 を取り上げ，メッセージ通信処理の評価を行ってきた．さらに，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きた．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[1)]や，アイドル時間を有効利用できることが明らかになってきた．

同時に，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[2)]．A-NET では並列オブジェクト指向言語 A-NETL を設計し，これを用いて OS 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．また，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．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動的評価を行っている．

一方で，計算機アーキテクチャの分野では，機械命令の実行を高速化することを目的としたマルコフ連鎖によるプリフェッチの予測[3)]や，分岐予測に対する予測手法の検討[4)]，データベースに対する投機的な検索[5)]など，投機実行に関する多くの研究がなされており，その有効性も確認されている．

† 宇都宮大学サテライト・ベンチャー・ビジネス・ラボラトリ
Satellite Venture Business Laboratory, Utsunomiya University
†† 宇都宮大学工学部
Faculty of Engineering, Utsunomiya University

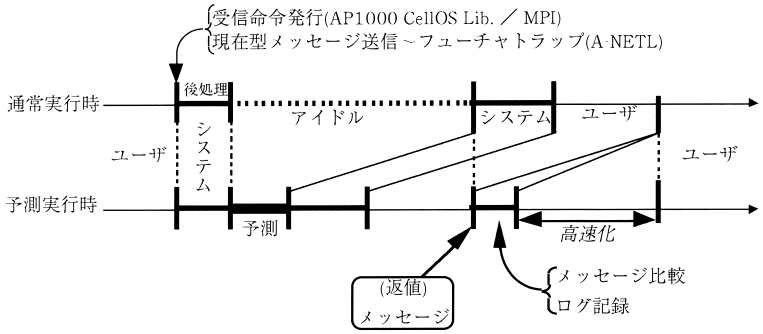


図 1 受信メッセージ予測時の流れ
Fig. 1 Flow of receiving message prediction.

これらを背景として，我々はこれまでに，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予測し，その結果に基づいて受信後の処理を投機的に実行することによって，高速化を試みる受信メッセージ予測法を提案しており[6]，最高で 29.8%の時間削減を実現している．

本稿では，最初に受信メッセージ予測法についてその処理の流れや予測方法，複数のプラットフォームに対する実装方針について説明する．次に，各予測方法を NAS Parallel Benchmark（NPB）プログラムに適用したときの予測成功率を求めた結果について説明する．さらに，これまでに実装が完了した A-NET マルチコンピュータを例として取り上げ，メッセージ転送機能とその実現方法について簡単に説明する．最後に，ベンチマークプログラムによる各予測方法での予測成功率[7]などのさまざまな評価について実験結果を基に考察する．

2. 受信メッセージ予測法

2.1 処理の流れ

一般的な並列分散処理系における通常のメッセージ通信時の流れを**図 1** の上段に，受信メッセージ予測を導入したときの流れを図 1 の下段に示す．通常実行時には一般的に，(1) ユーザプログラムが終了した場合，(2) 受信命令を実行したときに，メッセージが到着してない場合，(3) 他ノードへの現在型メッセージを送信しフューチャトラップが発生した場合，などに，システム（OS）に制御が移り，後処理を行う．後処理の終了後，次に実行す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．その後，他ノードからメッセージが到着すると，そのメッセージに対する領域確保処理などを行い，実行すべき処理を認識後，ユーザプログラム（メソッド）が実行される．

具体的な例としては，並列化ライブラリとして広く普及している MPI ライブラリなどの場合において，MPI_Recv 関数を実行したときに，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．

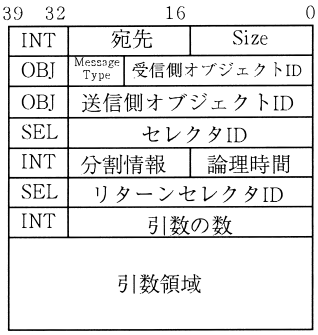


図 2 メッセージ構造
Fig. 2 Message structure.

受信メッセージ予測法では，これらのアイドル状態を利用して，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等から予測し，OS での受信処理や以降に続くユーザプログラムの中で実行可能な部分を投機的に先行実行することによって，予測ヒット時の高速化や，予測失敗時においても先行実行によるキャッシュの先読みなどによる高速化を目的としている．

2.2 予測方法の検討

受信メッセージ予測法を実現するために必要となるメッセージの予測項目や予測方法について，以下に述べる．

2.2.1 予 測 項 目

一般的に，並列計算機内でやり取りされるメッセージは，**図 2** のような項目から構成される．これらの項目の中から実際に予測対象するのは，1) メッセージサイズ，2) 過去型/現在型などのメッセージの型，3) メッセージの送信元，4) 起動すべき関数（メソッド）

などの項目である．

1) のメッセージサイズを予測することによる効果としては，メッセージ到着前にあらかじめ格納領域を確保することが可能となることや，引数の数が計算可能となることなどがあげられる．また，実際に到着したメッセージのサイズが，予測して確保した領域のサイズ以下であれば，予測失敗時でも再確保する必要はない．

3) のメッセージの送信元については，2) のメッセージの型と関連し，返値を返すノードを特定することが可能となる．さらに，4) の起動すべき処理を特定することにより，その起動準備として一時変数（自動変数）領域の確保が可能となる．

一方で，メッセージに含まている，上記 1)〜4) 以外の項目は，上記の値から求められるなどの理由により，ここでは予測は考えない．

2.2.2 予測方法

上記の予測項目から，具体的に考えられる予測アルゴリズムとして，ここでは，

a) 直前に到着したメッセージのみを参照（**直前**）
b) これまでに到着したメッセージの最大値（**最大値**）
c) これまでに到着したメッセージの平均値（**平均値**）
d) 最も回数が多かったもの（**最頻値**）
e) マルコフモデルに基づく連鎖（**マ連鎖**）
f) 同プログラムを前回実行したときのログ（**ログ**）

の 6 種類を考える．

a) の**直前**参照はあらゆる項目について使用可能であり，最も容易に実装可能であるが，プログラムが複雑になった場合はその成功率は低下する．b) は，特に，メッセージサイズを予測するために有効であり，予測値以下であれば予測成功となるため，その成功率は高いと考えられる．また実装も容易であり，予測するために必要となるメモリ量も 1 word で十分である．c) も実装が容易であり，メモリ量も少なくて済む．引数の内容によって使用可能となるが，非数値項目を多く含むメッセージヘッダを予測しようとした場合には，応用範囲は非常に限られる．

d) の**最頻値**（mode）は，a) と同様にあらゆる項目に使用可能であるが，予測対象となる項目の出現回数を記録するために，予測対象項目数に比例したメモリが必要となる．e) もあらゆる項目に対して使用可能であり，統計的な手法として a)〜d) に比べて最も予測成功率が高いと考えられる．しかし，n 回前に到着したメッセージまでを参照するマルコフモデルを使用した場合，予測対象項目数 m に対して，$O(m^{n+1})$ のメモリ容量が必要となり，さらにその検索にも若干の時間が必要となるため，現実的には $n = 2 \sim 3$ 程度が限度となる．

最後に，f) のアルゴリズムは，同じプログラムの引数/データを変えて何度も実行する場合に使用可能となるアルゴリズムである．その予測成功率は，変更されたデータによってメッセージの発生に変化がなければ非常に高くなる．また，実行時に与えられるデータによってメッセージが発生しなくなる可能性がある場合には，先の実行で記録された履歴とのずれが生じるため，これを適正に補正することによって予測成功率を上げることができる．メモリ量に関しては，少なくともプログラム実行 1 回分のメッセージをすべて保存しておく必要があるため，特に通信量が多くなるような粒度の小さいプログラムでは，使用メモリ量が非常に多くなる可能性がある．

2.3 実装方針

受信メッセージ予測法は計算機のハードウェアアーキテクチャ/並列化言語/ライブラリに依存しない手法である．我々はこれまでに，ワークステーションクラスタ上の MPI ライブラリ，富士通 AP1000 上のネイティブなライブラリおよび MPI ライブラリ，さらに A-NET マルチコンピュータ上に実装している．ここでは，各プラットフォームにおける実装方針について，簡単に説明する．

2.3.1 MPI に対する実装

MPI（Message Passing Interface）は，現在，世界的に広く使用されている並列化ライブラリであり，NEC Cenju-3 や富士通 AP1000，ワークステーション（WS）クラスタなど，広範囲に実装され，活用されている．今回は，これらのプラットフォームの中から AP1000 および WS クラスタを選び，両プラットフォーム上の MPI に本手法を実装した．MPI の関数の中でメッセージ送受信に一般的に使用される関数には，ブロッキングを行う MPI_Send() および MPI_Recv() があげられる．MPI_Recv() 関数によるメッセージ受信後の処理として，52〜700 μ 秒程度を費やすため，この処理を投機的に先行実行することによって高速化を図る．今回はこの MPI_Recv() を実現しているライブラリに対して変更を加えることによって，本手法を実装する．

2.3.2 A-NET に対する実装

我々の研究室でプロトタイプとしてハードウェアを開発した並列オブジェクト指向計算機の A-NET マルチコンピュータは，処理を行う PE（Processing Element）とノード間通信を行うルータの 16 ノードの対によって構成される．PE はマイクロプログラムに

よって制御されている．これまでに，メッセージの送受信処理やオブジェクトの管理などを行うOSも，メッセージ受信処理の高速化のためにマイクロプログラム化して実装しており，この処理のために約33μ秒を費やす．受信メッセージ予測法を，このマイクロプログラム化したOSのメッセージ受信処理部に変更を加えることによって実装し，この時間を投機実行することによって高速化を図る．

3. 評　　価

2.2.2項で述べた6種類の予測アルゴリズムの予測成功率や，本手法による速度向上を調べるために，MPI上ではNPBプログラムの中からkernel 3種，Application 1種類のプログラムを取り上げ，実行した．また，A-NET計算機上においては，比較的メッセージの類似度が高い8Queens，マスタースレーブ型数値計算プログラムの台形公式による定積分，同一処理のループとして実行されるLife Gameの3種類を取り上げ，実行させて評価した．これらのプログラムを実行したとき，ある特定のノードに到着するメッセージ系列から，各予測アルゴリズムで，各予測項目がどの程度予測が成功するか，予測にはどの程度の時間が必要かなどを求め，その結果について詳しく考察した．

3.1 NPBによる評価

ここでは，基本的なベンチマークプログラムであるNPBにおけるメッセージ系列の特徴と，予測アルゴリズムとの関連を明らかにする．

NPBプログラムは，NASAにより開発され，広く使用されている並列スーパーコンピュータ用ベンチマークプログラムで，kernel 5種，CFD Application 3種から構成されている．今回対象としたのはこのベンチマークプログラムをMPIを用いて記述したものであり，この中からFT，IS，CG，BTを取り上げてNEC Cenju-3上で実行し，その関数名，int/doubleの型，および送信元のノード番号を，時間的系列に従って記録した．

この系列に対して，2.2.2項で述べた各予測アルゴリズムを適用したときの予測成功率を求めた．ただし，予測項目は数値ではないため，b) **最大値**および c) **平均値**は使用しない．さらに，今回はユーザレベルにおいてメッセージ系列を記録したため，ライブラリレベルで実現されている複数受信関数（複数のノードを対象とするリダクション処理など）は，ユーザレベルにおいて個々のメッセージの到着順序を認識し，これを記録することは不可能である．各プログラムを複数回実行したときに記録される系列は，そのアルゴリズムから考えてもほぼ決定的となるため，f) **ログ**による予測の成功率は100%となる．

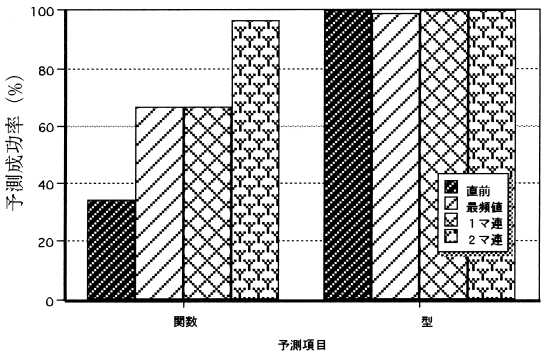


図3 CGに対する予測方式とその成功率
Fig. 3 Success ratio in CG.

本稿ではメッセージ総数が比較的多かったCGおよびBTでの結果のみについて説明する．また，これらのプログラムは1,000回を超えるメッセージの送受信があり，マルコフテーブルを作成するための十分な系列を確保できるため，e) **マルコフ連鎖**を，

e-1) 前回からの連鎖による予測（**1マ連**）

e-2) 前々回からの連鎖による予測（**2マ連**）

と次元を拡張して評価する．

3.1.1 CGによる評価

図3は，NPBのkernelの中から，正値対称な大規模疎行列の最小固有値を求めるための共役勾配法（CG）のプログラムによる評価結果である．このプログラムは，全対全通信のみを使用し，全ノード上でほぼ同時に通信が行われるため，ノードごとの通信パターンのばらつきが非常に少ない．また，このプログラムを実行した結果から，プログラムの中で使用される関数は，AABAAB···の単純な系列を持つことが分かる．この系列に直前予測を使用した場合には，

AAB<u>AA</u>BAAB···　A → A のため予測成功
AABA<u>AB</u>AAB···　A → B のため予測失敗
AABAA<u>BA</u>AB···　B → A のため予測失敗

となり，その予測成功率は理論的に1/3となる．系列に入る前後に初期化などの処理が入るが，系列中のメッセージ数が十分に多いため，図3の結果でも予測成功率はほぼ1/3となっている．同様に，**最頻値**および**1マ連**では約2/3，**2マ連**ではほぼ100%の予測が可能となっている．一方，型は，double型しか使用しないため，その予測成功率はいずれの予測アルゴリズムを使用してもほぼ100%となっている．

以上のことから，単純で明らかな系列を持ち，さらに十分な通信数がある場合は，マルコフ連鎖の次元を

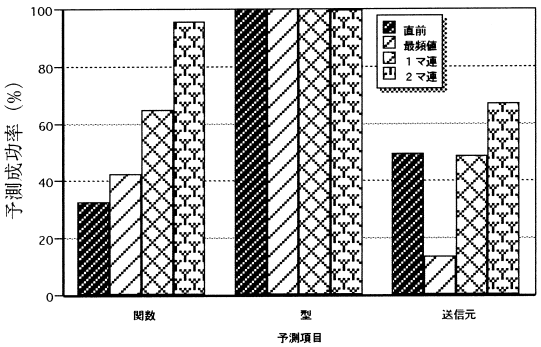


図 4 BT に対する予測方式とその成功率
Fig. 4 Success ratio in BT.

増加させることによってその予測成功率を大きく上昇させることが可能となるが，2.2.2 項で述べたように，メモリの使用量も増加することを考慮する必要がある．

3.1.2 BT による評価

図 4 は NPB の CFD Application である 5 × 5 block size ADI Interation による CFD Application（BT）に対して各予測方法を適用した結果である．このプログラムにおいては，東西南北上下の周囲 6 方向の特定のノードとのみ通信を行う．ほぼ同時に通信が行われるが，受信ノードと送信ノードは区別されていないため，CG 同様にノードごとの通信パターンのばらつきが少ない．また，関数系列は ABCCABCC···の系列を持ち，関数の予測成功率は**直前**を用いた予測アルゴリズムでは 32%程度となるが，アルゴリズムが複雑になるにつれて単調に増加していき，**2 マ連**では 95.7%となっている．また，本アプリケーションで使用される変数の型は，プログラム内では変数（すなわち，変数の型を表す変数）として記述されてはいるが，結果的には統一されているためその予測成功率も 100%近くなっている．一方，送信元は，その系列が AAABBBCCC···の系列を持つため，これまでに述べてきた予測アルゴリズムでは全般的に成功率を上げることは難しく，最高でも 67.0%にとどまっている．

3.1.3 NPB による評価のまとめ

ここでは説明しなかった FT（FFT）や IS（Integer Sort）においてもここで述べた 2 種のプログラムとほぼ同様の傾向を示しており，一般的なアプリケーションプログラムにおいて，そのメッセージ通信に規則性があることが実証できた．さらに，より複雑な予測方法を用いることによって，これらの規則性を有効に抽出することが可能となることが分かる．一方で，メッセージ数が少ないためここでは取り上げなかったが，FT における変数の型は，AAABAAAB···の系列を持ち，この系列にこれまでの予測アルゴリズムを適応した場合，逆に **2 マ連**では直前や **1 マ連**に比べて，予測成功率が同等かそれ以下になることも分かっている．

3.2 A-NET のメッセージ転送機能

先に述べたように，受信メッセージ予測法は計算機のアーキテクチャに依存しない高速化手法である．ここでは，その中からすでに実装が完了し，詳細な評価データの収集を完了した A-NET マルチコンピュータを例として取り上げる．この節では，後の説明のために，A-NET マルチコンピュータのメッセージ転送処理の説明と評価条件について簡単に説明する．

3.2.1 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の計 4 種を定義し，A-NETL プログラム内で各命令を明示的に記述する．

各メッセージ転送命令は，コンパイラによって対応する機械命令にコンパイルされ，このときに得られたオブジェクト ID，セレクタ ID などがオペランドとなる．この機械命令が実行されると，PE 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 2）．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転送する．

3.2.2 メッセージ受信処理

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する割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，

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
d) ユーザメソッドの実行

の 4 段階で，メッセージを処理する．基本的に，a)〜c) の処理と d) の一部はメッセージヘッダの予測のみで実行可能である．しかし，d) 全体を投機実行する場合にはメッセージの本体の予測が要求され，これには値予測（Value Prediction[9]）が必要となるため，今回は考えないこととする．

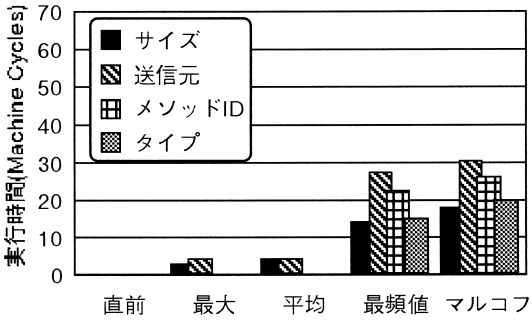


図 5 実行時間の評価（履歴取得時）
Fig. 5 Execution time for obtaining log.

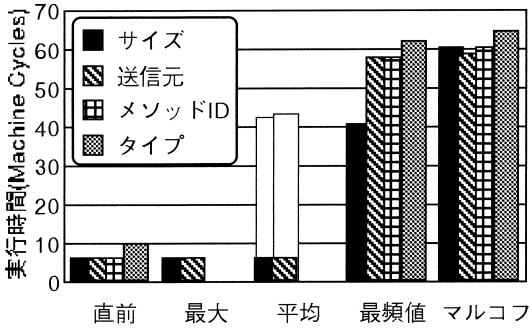


図 6 実行時間の評価（予測時）
Fig. 6 Execution time for prediction.

3.2.3 評価条件

今回動的評価を行うために本手法の実装/評価に使用した A-NET マルチコンピュータは，各ノードがそれぞれ独立に 30 MHz で動作する．各 PE を制御するマイクロプログラムは，1 マシンサイクル（MC）を 5 フェーズ 167 ns で実行する．ルータ間接続のトポロジは静的に可変であるが，今回は 4 次元のハイパーキューブ網を用いた．

3.3 予測アルゴリズムの動的評価

3.3.1 評価項目

受信メッセージ予測法に対するメッセージ予測アルゴリズムについての評価は，

1) 実装の容易さ
2) 予測に必要となる時間
3) 履歴を記録するために必要となるメモリ量
4) 予測成功率
5) 速度向上
6) 適用可能範囲

の，6 項目とした．本稿ではこれらの中から，ページ数の制限上，2)，4)，および 5) の結果を述べる．

3.3.2 予測に必要な時間の評価

図 5 に各予測アルゴリズムを実現するために必要となる履歴を取得するのにかかる時間を，**図 6** に履歴を用いて実際に予測するときに必要となる時間を示す．ただし，図 6 の平均の白抜き部分は，A-NET マルチコンピュータにおいて平均を求めるための除算 1 回に要した時間を示している．これらの図より，第 1 に，全予測方法にわたって履歴取得にかかる時間が予測時間よりも小さいことが分かる．これは，各アルゴリズムの実装に際して，履歴取得時の負荷を可能な限り減らし，その分予測実行時の時間を使うようにデータ構造などを考慮して実装したことを反映している．このような方針をとった理由は，1) メッセージの履歴を記録するのは，受信したメッセージを処理しているとき（プロセッサは通常実行の状態である）に行う必要があり，これが通常処理のオーバヘッドとなるため，高速に行う必要がある．2) 実際の予測は，投機実行時（プロセッサがアイドル状態となったとき）に行うものであり，このアイドル状態のために，ある程度時間を使うことが可能であることによる．

第 2 に，**直前**/**最大**/**平均**に対して**最頻値**/**マルコフ**がきわめて大きな値を示しているが，これは，2.2.2 項で述べたようなテーブルの検索・登録に時間がかかるためである．特に，履歴の取得において，予測内容に対して最も時間がかかっているのは送信元であるが，これは，送信元となる種類の数が他に比べて多いため，メッセージのテーブルを検索する手間がより多くかかるためである．

表 1 予測成功回数
Table 1 Number of prediction success.

	8Queens	Life Game	台形公式
予測対象数	82	296	297
ヘッダのみ	82	254	244
予測失敗	0	42	53

（単位：回）

3.3.3 予測成功率

表 1 に 3 種類のベンチマークプログラムを実行したときの，予測対象メッセージ数（予測対象数），ヘッダのみ予測成功（ヘッダのみ），予測失敗のそれぞれの回数を示す．

ヘッダのみ予測が成功した場合は，OS 内のメッセージ受信処理完了後から再実行を行う．また，引数を含めたメッセージ全体の予測が成功した場合には，投機実行を通常実行に切り替えて実行を継続できるが，予測が失敗した場合は，OS 内のメッセージ受信処理からすべて実行し直さなければならない．このように予測の成功/失敗の判断を段階的に行うことによって，可能な限り再実行のオーバヘッドを減らすことが可能と

表 2 受信処理時間の増減と Q11 における受信メッセージ分類
Table 2 Fluctuation of receiving message time and classification of message in Q11.

	増減 [MC]	Q11 [回]
予測なし	+20	263
予測成功	−41	244
予測失敗	+32	53

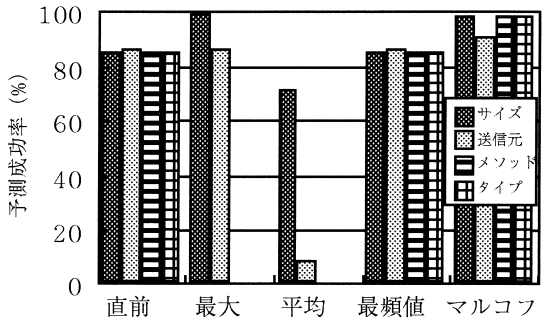


図 7 台形公式による定積分における予測成効率
Fig. 7 Success ratio on Q11.

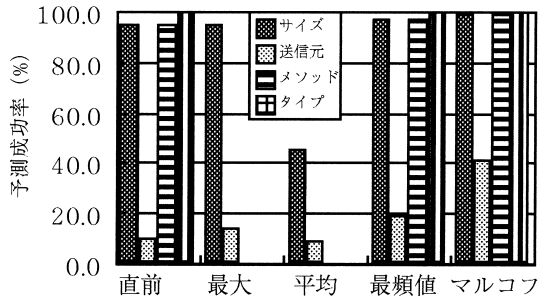


図 8 Life Game における予測成効率
Fig. 8 Success ratio on Life Game.

なる．

この表より，ヘッダのみの予測成功率は 82.2〜100%となり，性質の異なるアプリケーションにおいても比較的高い予測成功率となることが分かる．

次に，**表 2** の 3 種のベンチマークプログラムの中から特徴的な 2 つを取り上げ，それぞれの予測アルゴリズム，予測項目と予測成功率の関係を，**図 7**，**図 8** に示す．図 7 は台形公式による定積分，図 8 は Life Game である．

図 7 の台形公式による定積分は，数値演算に対するマスタースレーブ型のリダクション処理をデータを変更しながら 20 回繰り返すプログラムである．スレーブ側のノードにおける受信処理は，すべてマスタ側から送られてくるメッセージであるため，どの項目についてもほぼ 100%予測を成功させることができる．また，マスタ側ノードにおける受信処理は，スレーブとなる全ノードからほぼ同時にメッセージが送られてくるため，**直前**/**平均**/**最頻値**による予測は困難となる．マスタ側とスレーブ側のトータルとしては，約 80%を超える程度の予測成功率となる．

また，メッセージサイズの予測については，予測した値を用いてメッセージ領域を確保するため，実際に到着したメッセージサイズより大きな値として予測してあれば，領域の再確保を行う必要がないため，予測は成功とする．よって，サイズを最大で予測した場合の予測成功率はきわめて高くなる．一方，**マルコフ連鎖**による予測成功率が他のアルゴリズムより高くなっているのは，マスタノード側で受信するメッセージが同時に多数のスレーブノードから送られてくる場合であっても，各スレーブで行われる処理やメッセージ転送距離の違いなどにより到着に規則性が発生するためである．

図 8 の Life Game では，全ノードが Cell として同等な立場で動作し，周囲 8 方向の特定のノードとのみ通信を行う．この場合もほぼ同時に通信が行われるが，台形公式による定積分のベンチマークプログラムのように明確に受信ノードと送信ノードは区別されていない．よって，各ノードで送受信されるメッセージの類似性が非常に高くなり，サイズ/メソッド/タイプについての各予測アルゴリズムにおける予測成功率は，**マルコフ**を最高として，**最頻値**，**最大**，**直前**の順に高くなる．ただし，送信元については，通信が同時に起きる影響が大きく，また，メッセージ転送距離もすべて 1 であるため，各ノードで受信するメッセージの送信元を予測することはどの方式を用いても低い値となっている．

3.3.4 速度向上

図 9 は，受信メッセージ予測法による実行結果の具体的な例として，台形公式による定積分のベンチマークプログラムを直前の予測方式を用いて実行し，その実行履歴中から，マスタとなるノードのある時点での様子を示したものである．通常実行時（図 9 上）には，ユーザプログラムが 460 MC 動作した後サスペンドし，システムが 17 MC 動作した後にアイドル状態（約 3,000 MC 程度）となる．また，返値メッセージの到着後，OS による受信処理に 200 MC を要している．

予測実行時には，予測失敗時に元の状態に復帰するために必要となる情報を記録する処理などのために，ユーザプログラムのオーバヘッドとして 6 MC（動作時間 460 MC の 1.3%）増加している．ユーザプログラムがサスペンドした後，システムは合計 381 MC 動作しているが，このうち，従来のアイドル時を利用して

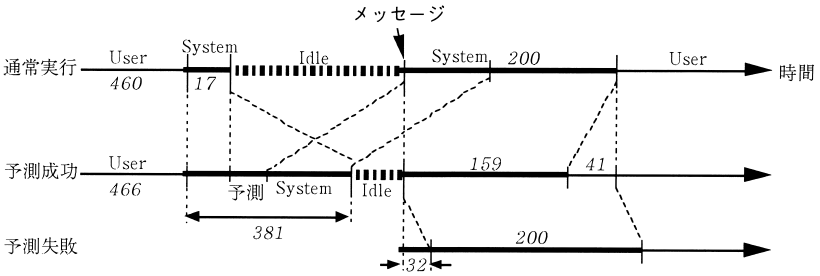


図 9 台形公式の積分実行時のマシンサイクル数
Fig. 9 Machine cycles of Q11.

メッセージの予測と受信処理に要した時間は 364 MC である．返値メッセージが到着した後，到着したメッセージと予測したメッセージとの比較や履歴の取得に 32 MC を要し，その後受信処理などが行われるが，投機的先行実行による効果によって，合計 159 MC の動作でユーザプログラムに復帰可能となり，通常実行に比べて 41 MC（200 MC の 20.5%）の高速化となった．よって，予測成功時においては，オーバヘッドと高速化の効果を総合すると，$41 - 6 = 35$ MC の高速化となる．

一方，予測を失敗した場合のオーバヘッドについては，通常の受信処理（200 MC）に，メッセージ比較と履歴取得（計 32 MC）が加わるため，予測実行しない場合に比べて 32 MC の増加となる．さらに，アイドル時間がないために予測・投機処理を行わない場合においても，到着したメッセージを履歴として保存するために 20 MC を要する（表 2）．

これらの値を台形公式による定積分のプログラム全体に適用した場合，予測対象とならないメッセージ 263 回，予測成功 244 回，予測失敗 53 回であって，本手法実装前の 295,911 MC に対して，294,427 MC となり，1,483 MC の速度向上となる．

本評価に使用した A-NET マルチコンピュータにはプロセッサキャッシュが存在しないため，2.1 節に述べたキャッシュに対する先読みの効果は得られないが，AP1000 における評価では，受信処理時間のうち，最高で約 52.9%，平均で約 30.5%を削減したことを付け加えておく．

3.3.5 考　　察

これまでに述べてきた結果および，3.3.1 項で述べた評価内容の結果から，以下のようなことがいえる．

(1) 今回求めた予測成功率に加え，予測に必要なメモリ容量等を総合的に考慮すると，各項目に適する予測アルゴリズムは，1) メッセージサイズのように上限や下限がある程度決定可能な値に対する予測には**最大値**（最小値），2) メッセージの通信距離や，メソッドの起動順序など物理的/論理的に強い規則性が見られる場合には，**マルコフ連鎖**，3) メッセージタイプのようにプログラム中でほぼ統一されている場合には**直前**または**最頻値**の各予測アルゴリズムを使用するのが適する．

(2) 表 2 に示した予測成功/失敗回数より，Q11 の予測成功率は，約 82.2%と求められる．予測成功率と全実行時間との関係を考えた場合，仮に，同予測アルゴリズムにおいて予測成功率が 100%となった場合，全実行時間の約 4,378 MC が削減可能になると求めることができる．同様に，予測成功率が 0%となった場合には，約 11,770 MC の増加となる．これらの値より，予測成功率が約 72.9%を超えた場合に全体の性能を向上させることが可能であると求めるられる．

(3) 各予測アルゴリズムにおいて，全体の処理性能に影響するのは，予測を行うために必要となるログを取得する時間である．一方で，実際に予測を行う処理は，アイドル時間に行われるため，全体の性能には影響しない．

(4) 3.3.4 項で述べた評価における予測アルゴリズムは**直前**を用いている．他の予測アルゴリズムを用いた場合を考えると，1 回の受信処理において，さらに図 5 に示したログ取得時間が必要となる．たとえば，現在，**直前**で行っているメッセージサイズの予測を**最頻値**で行った場合，1 回の受信処理時間が約 14 MC 増加するため，予測成功時の従来の速度向上である 41 MC が，$41 - 14 = 27$ MC の速度向上となる．同様に，全予測項目をマルコフを用いて予測した場合には，予測が成功した場合でも 1 回の受信処理時間が約 95 MC 増加するため，$41 - 95 = -54$ となり，予測が成功したにもかかわらず，54 MC のオーバヘッドとなる．このことから，Q11 においては，予測成功率と速度向上のトレードオフの点から，**マルコフ**よりも**直前**のアルゴリズムが適しているといえる．このように，複雑な予測アルゴリズムによって予測成功率を高めることが必ずしも速度向上につながるとはいえないことが明ら

かになった．

4. 関連研究

受信メッセージ予測法やその関連研究に関しては，文献 6) に述べた．さらに，今回取り上げた NPB のノード間通信パターンの解析として，Parallel Virtual Machine（PVM）を用いて記述した NPB における例[8]がある．ここでは，Silicon Graphics のマルチプロセッサワークステーションを Ethernet などにより接続し，NPB の科学アプリケーションプログラムを用いて実験している．この結果，通信周期やメッセージサイズ，通信距離などには高い局所性があると報告しており，受信メッセージ予測法の前提である “メッセージ通信の規則性” の 1 つと考えることができる．

一方，予測アルゴリズムに関しては，ここ数年，投機実行のための Value Prediction の研究としてさまざまな予測方式が提案され，その成功率等が報告されている．この中では，SPECint92 に対して，same 系列（例：$\cdots 3, 3, 3 \cdots$），stride 系列（例：$\cdots 4, 5, 6, 7 \cdots$）など，本稿で述べた予測アルゴリズムより単純であると考えられるアルゴリズムを使用し，機械命令レベルにおいて値予測を行っている．その結果，平均で 59.5%の予測が可能であり，このとき全体で 16.2%の実行時間の短縮が図られると報告[9]している．本稿では，より高度な予測アルゴリズムを用いており，また，メッセージのヘッダ部分の予測に重点をおいているため，単純に比較することは難しいが，この報告における予測成功率に比べてさらに予測成功率を高くすることが可能であり，単純なメッセージ本体であればこれを予測することも十分に可能性があると考えられる．

5. おわりに

本稿では，メッセージ転送処理を高速化するための手法として受信メッセージ予測法について述べ，6 種類のメッセージ予測アルゴリズムを評価した．その結果から，以下のようなことがいえる．

(1) 本手法を適用するためのプラットフォームとしてここでは MPI ライブラリ，および A-NET マルチコンピュータを取り上げ，各プラットフォームに対する実装方針を検討した．結果として，各プラットフォームにおいて約 33〜700 μ 秒の処理が投機実行可能であり，その実装もライブラリや OS などソフトウェアに対してのみ変更を加えることによって可能であることを示した．

(2) 本手法の初期評価として NPB を取り上げ，各予測アルゴリズムにおける予測成功率を求めた．その結果，一般的な並列アプリケーションプログラムのメッセージ通信には規則性があることを実証した．

(3) 本手法の第 2 の評価として A-NET マルチコンピュータに実装した結果，その予測成功率はすべての予測対象項目において 80%以上の予測成功率となることが分かった．

(4) メッセージ受信 1 回あたりユーザプログラムのオーバヘッドとしては動作時間の 1.3%増加すること，予測成功時の速度向上として受信時間の 20.5%を削減すること，台形公式の定積分のプログラム全体では予測成功率 82%，1,483 MC の速度向上であることなどが分かった．

(5) これらの値をもとに詳細な考察を述べた．また，オーバヘッドの原因を指摘し，その解決手法を述べた．

以上のことより，本手法の有効性と，メッセージ処理機構を持つさまざまな並列プラットフォーム上に対する幅広い応用が可能であることが確認できた．

今後，さらなる速度向上のための可能性として以下のような手法が考えられる．

(1) 今回の評価は，メッセージのヘッダを予測し，受信処理の投機実行を行うことによって，速度の向上を目指している．速度向上に大きな影響を与える投機実行可能な処理の量は，メッセージの本体を予測してユーザプログラムを投機実行することによって，大きく増加させることが可能である．

(2) 3.3.5 項 3)，4) において述べたとおり，ログを取得する時間がオーバヘッドとなっているため，これらの処理を動的に行わず，プログラムのコンパイル時などに，メッセージの系列をあらかじめ取得しておき，予測に使用する，静的な予測手法を用いることによって，オーバヘッドを最小に抑えることが可能となる．

謝辞 本研究は，一部文部省科学研究費（基盤 (C) 課題番号 12680328, 基盤 (B) 課題番号 10558039, 奨励 (A) 課題番号 09780237），並列・分散処理研究推進機構の援助による．

参考文献

1) Iwamoto, Y., Ooguri, K., Yoshinaga, T. and Baba, T.: A Comparis of Communication Performance in the NEC Cenju-3 and FUJITSU AP1000, *Proc. 1st CENJU WORKSHOP*, pp.60–64 (1997).

2) 吉永 努，澤田 東，広田 守，阿部大輝，岩本善行，馬場敬信：A-NET マルチコンピュータのシステム性能評価，電子情報通信学会論文誌，Vol.J81-D-I, No.4, pp.368-376 (1998).
3) Joseph, D. and Grunwald, D.: Prefetching using Markov Predictors, *Proc. International Symposium on Computer Architecture* (*ISCA'97*), pp.252–263 (June 1997).
4) 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, pp.55–60 (1997).
5) 佐々木敬泰，高山 毅，弘中哲夫，藤野清次：投機的実行を用いたデータベース処理—マルチトランザクション環境下での高速化，情報処理学会研究報告（SWoPP'97），97-ARC-125, pp.127–132 (1997).
6) 岩本善行，澤田 東，阿部大輝，澤田康雄，大津金光，吉永 努，馬場敬信：メッセージ転送処理の高速化法とその評価，情報処理学会論文誌，Vol.39, No.6, pp.1663–1671 (1998).
7) Iwamoto, Y., Ootsu, K., Yoshinaga, T. and Baba, T.: Message Prediction and Speculative Execution of the Reception Process, *Proc. 11th IASTED International Conference, Parallel and Distributed Computing and Systems* (*PDCS'99*), pp.329–334 (1999).
8) Kim, J-S. and Lilja, D.J.: Characterization of Communication Patterns in Message-Passing Parallel Scientific Application Programs, *CANPC '98*, pp.202–216 (1998).
9) 岩田 靖，安里 彰，新井正樹，小沢年弘，木村康則：投機的実行のためのデータ予測可能性，情報処理学会研究会報告（SWoPP'98），98-ARC-130, pp.55–60 (1998).

(平成 11 年 5 月 6 日受付)
(平成 12 年 7 月 5 日採録)



岩本 善行（正会員）

1999 年宇都宮大学大学院工学研究科生産・情報工学専攻博士後期課程修了．現在，同大学サテライト・ベンチャービジネス・ラボラトリ非常勤研究員．博士（工学）．主に，並列計算機アーキテクチャおよび初等中等教育における情報教育に興味を持つ．



大津 金光

1993 年東京大学理学部情報科学科卒業．1995 年同大学大学院修士課程修了．1997 年同大学院博士課程退学，同年より宇都宮大学工学部助手となり現在に至る．理学修士．高性能計算機システム，特にマイクロプロセッサアーキテクチャに興味を持つ．



吉永 努

1986 年宇都宮大学工学部情報工学科卒業．1988 年同大学大学院修士課程修了．同年より宇都宮大学工学部助手．1997 年から翌年にかけて電子技術総合研究所・客員研究員．2000 年 8 月より電気通信大学大学院情報システム学研究科助教授．博士（工学）．並列計算機アーキテクチャ，リコンフィギュラブル・コンピューティングなどに興味を持つ．電子情報通信学会，IEEE 各会員．



馬場 敬信

1970 年京都大学工学部数理工学科卒業．1975 年同大学大学院博士課程単位取得退学．同年より電気通信大学助手，講師を経て，現在宇都宮大学工学部教授．工学博士．1982 年より 1 年間メリーランド大学客員教授．計算機アーキテクチャ，並列処理等の研究に従事．電子情報通信学会，IEEE 各会員．1992 年情報処理学会 Best Author 賞．著書「Microprogrammable Parallel Computer」（MIT Press),「コンピュータアーキテクチャ」（オーム社）等．