受信メッセージ予測法による MPI 受信処理の高速化

岩本善行[†] 足立涼子[††] 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の手法として我々が提案してきた受信メッセージ予測法を MPI に対して実装した結果を報告する．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，受信処理などを先行実行することによって高速化を図る手法である．本手法は，プラットフォームや並列言語/ライブラリに依存しない手法であり，今回，世界的に広く普及している MPI ライブラリに実装することによって，本手法のプラットフォーム非依存性を実証した．さらに，NAS Parallel Benchmark プログラムを用いて，本手法の評価を行った．その結果をもとに，本手法の有効性や特徴について考察する．

High Speed Receiving Process of MPI by Receiving Message Prediction

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

This paper presents the implementation and evaluation results of the "receiving message prediction" on the MPI library. In the receiving message prediction, when a node processor is idle, it predicts a message which will receive next, and speculatively executes the process of a message reception. By implementing this method on the MPI library, which is used world-wide, we prove that this method is independent of the platforms. And using the NAS Parallel Benchmark programs, we evaluate this method. Results and discussion are also included in this paper.

1. はじめに

並列・分散処理が広く普及していく中で，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．

これらを背景として，我々はこれまでに，次に到着する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先行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いる[5),6),9)]．我々は，これまでに富士通 AP1000[1)] や A-NET マルチコンピュータ[2)] 上に本手法を実装し，AP1000 上においては最高で 29.8%の受信処理時間削減を達成している．

今回，さらに，Sun Enterprise 3500 およびワークステーション（WS）クラスタ上の Message Passing Interface（MPI）上に実装し，NAS Parallel Benchmark（NPB）プログラムを用いて評価を行った．

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について簡単に説明する．次に，Sun Enterprise 上の MPI に対する本手法の実装方法について述べる．最後に，本手法の評価として，NPB プログラムを実行した結果を示し，これらの結果をもとに考察する．

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** の上段に示す．通常実行時には，ブロッキング受信命令を実行したときにメッセージが到着していない場合，アイドル状態（メッセージ待ち

† 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
††† 電気通信大学大学院
The University of Electro-Communications

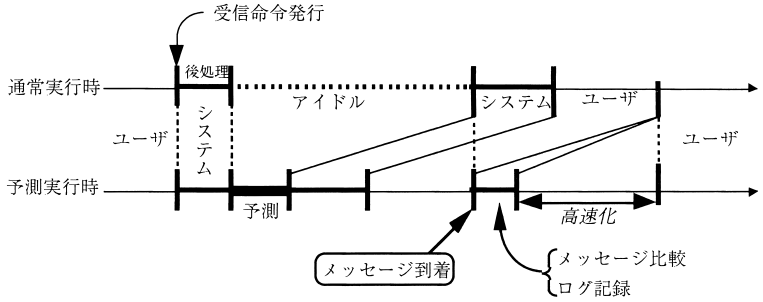

図 1 受信メッセージ予測時の流れ
Fig. 1 Flow of receiving message prediction.

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処理（後述）を行い，実行すべき処理を認識後，ユーザプログラム☆の実行が再開される．

受信メッセージ予測法では，図 1 の下段に示すように，このアイドル状態を利用して，これから到着するメッセージを過去の履歴などから予測し，受信処理およびそれに続くユーザプログラムの中で実行可能な部分を先行実行する．その後，実際のメッセージが到着すると，そのメッセージと予測したメッセージとを比較し，予測の成功/失敗を判定する．同時に，のちの予測のために，今回到着したメッセージを履歴として保存するための処理を行う．

以上のような流れによって，予測ヒット時の高速化を目指している．

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アルゴリズムについて，簡単に述べる．

2.2.1 予 測 項 目

本実装に使用した MPI ライブラリ内で送受信されるメッセージは，**図 2** のような構造体に格納され，処理される．

メッセージを予測する場合，1) メッセージ全体を 1 つのまとまりとして予測する，2) メッセージに含まれる項目を個別に予測する，の 2 通りが考えられる．1) の場合，全体を 1 つの項目として扱うため処理に必要な時間（メッセージの予測に必要な時間＋予測を行うために必要な履歴の取得にかかる時間）が少なくて済むが，予測成功率の低下は避けられない．一方，2) の場合，予測の精度は上がるが，メッセージに含まれる

link	次構造体へのポインタ
org_len	受信データサイズ
type	受信データのタイプ
to	送信先ランク
from	送信元ランク
ack_req	受信確認要求
len	送信データサイズ
msg	メッセージ本体

図 2 MPI におけるメッセージの構造体
Fig. 2 Message structure of the MPI.

項目個別に処理する必要があるため，その処理に非常に多くの時間が必要となる．本研究では，速度の低下を最小限に抑えるため，1) の手法を採用してメッセージ全体を予測することとする．

なお，MPI ライブラリにおいては，受信関数の引数によって送信元（図 2 中の from），サイズ（同 org_len）などの項目が，ユーザによってあらかじめ指定される．このような項目についてはその値を，予測したメッセージに埋め込むこととする．

次に，予測したメッセージと，実際に到着したメッセージを比較する場合を考える．この場合においても，メッセージ予測時と同様に，1) メッセージ全体を 1 つとして予測の成功/失敗を判定する方法，2) メッセージに含まれる項目を個別に判定する方法，の 2 通りが考えられ，その特徴も先に述べたとおりとなる．加えて，プラットフォームによっては，2) の手法によって，予測に失敗した項目に関する処理のみを再実行することによって，予測失敗時のオーバヘッドを最小限に抑えることが可能となる．

本研究では，速度低下を抑えることとともに，今回実装した MPI ライブラリに関しては部分的に再実行することによる効果が少ないとの判断から，1) を採用し，メッセージ全体を 1 つとして予測の成功/失敗の判定を行うこととする．

☆ MPI ライブラリ内部の処理もユーザモードで実行されるが，以降特に断わらない限り，「ユーザプログラム」とは，MPI ライブラリにおける処理を除いたユーザプログラム部分（すなわちユーザが記述したプログラム部分）を指すこととする．

2.2.2 予測アルゴリズム

上記の予測項目から，ノード間で送受信されるメッセージ系列に存在する規則性[8),10)]に対して，具体的に考えられる予測アルゴリズムとして，

a) 直前に到着したメッセージのみを参照
b) 過去に到着したメッセージの各項目における最大値
c) 過去に到着したメッセージの各項目における平均値
d) 最も回数が多かったもの
e) マルコフモデルに基づく連鎖[3)]
f) 同一プログラムを前回実行したときのログ
g) stride 系列 $(\ldots, 6, 7, 8, \ldots)$ による予測
h) メッセージの発生周期（時刻）を用いた予測

などの方法が考えられる．ここで，b) や c) のような値をとる項目はメッセージのサイズ（図 2 中の len）などに限られる．各予測アルゴリズムの詳細については参考文献 11) に譲るが，本稿では，予測に必要となる時間が最も少なく，実装も容易である a) のアルゴリズムと，予測成功率が最も高くなると考えられる e) のアルゴリズムを採用する．

2.3 実　　装

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，富士通 AP1000 上のネイティブな並列ライブラリおよび MPI ライブラリ，A-NET マルチコンピュータ上に実装し，評価を完了している．ここでは，今回実装/評価を行った WS クラスタおよび Sun Enterprise 3500（以下，E3500）に対する実装方法について説明する．なお，本実装は，それぞれのプラットフォームにおける通信速度の違いを明確にするため，デバイス指定は両プラットフォームとも ch_p4 を使用しているが，一部のベンチマークプログラム（NPB の IS と SP）を除いて，E3500 で一般的に使用される ch_shmem と ch_p4 との速度差は 3%未満であったことを付け加えておく．

2.3.1 MPI_Recv 関数に対する実装

MPI は，現在，世界的に広く使用されている通信ライブラリであり，Sun 倏 nterprise のような商用並列計算機や WS/パーソナルコンピュータ（PC）クラスタなど，プラットフォームを選ばず広範囲に実装され，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 E3500 と WS クラスタの MPI に本手法を実装した．ここで使用した MPI ライブラリは，Argonne National Laboratory において開発/配付されている MPICH Ver.1.1.2 である．

MPI に定義された関数の中で，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() を実現しているライブラリは**図 3** のような階層構造を持つが，今回はこの中の mpid（図 3 (a)）および p4lib（図 3 (b)）に対して変更を加えることによって，本手法を実装する．なお，今回対象としたデバイスは p4 であるが，受信処理時にアイドルが発生する部分を特定し，その箇所に対してメッセージを予測するための変更を行うことにより，他のデバイスにおいても容易に実装可能である．

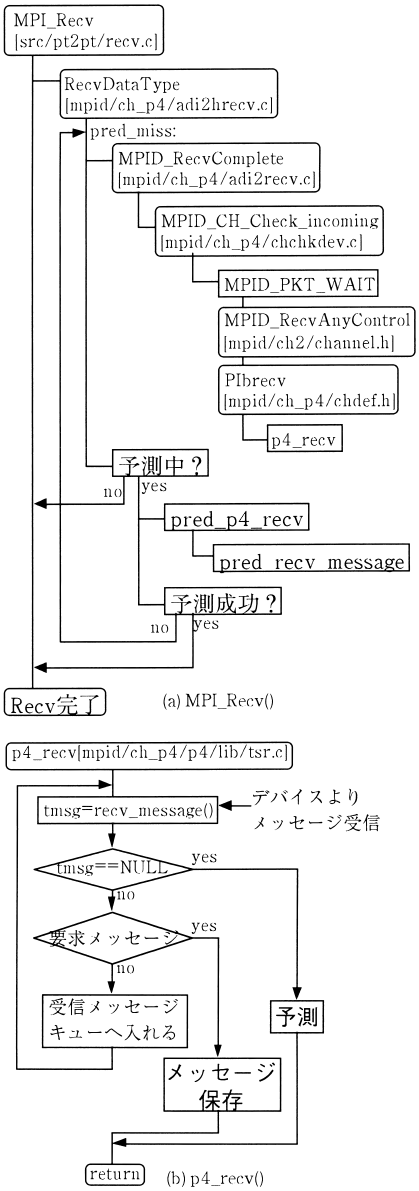

図 3 MPI_Recv に対する実装
Fig. 3 Implementation into the MPI_Recv.

また，本稿では，今回の実装によって受信処理部分のみの先行実行を行い，MPI_Recv() 関数の終了時点において，実際に到着するメッセージを待つこととする（図 3 (b) の pred_p4_recv および pred_recv_message が実際に到着するメッセージを受信する）．このことによって，MPI_Recv() 関数を終了させるために必要な部分（メッセージのヘッダ部分）のみを予測する（メッセージの本体部分は今回は予測せず，領域の確保のみとなる）ことで実装が完了し，また予測に失敗した場合に必要となるリカバリ処理も，非常に限られた部分のみで済ませることが可能となる．後述する予測の成功率もこのヘッダ部分のみの判定とした．

MPI_Recv() 関数内でのメッセージ受信後の処理としては，

a) バイトオーダの変換
b) メッセージ種別（コントロールメッセージ/ノーマルメッセージ）の認識
c) メッセージ到着待ちキュー内の検索/キューからの削除
d) メッセージサイズ（short/long/vlong）の認識
e) 受信バッファからユーザバッファへのコピー
f) 受信ステータス（受信サイズ，送信元番号，タグ，エラーなど）の設定

などがあり，MPI_Recv() を先行実行する場合，これらの処理をメッセージ到着前に完了することによって，高速化を図る．なお，今回はメッセージの本体を予測してないため，e) のようなメッセージ本体が不可欠な処理は，実際のメッセージが到着してからもう一度行うこととする．

2.3.2 先行実行の有効範囲

MPI_Recv() は MPI における基本的な関数であるため，この関数と受信処理部分を共用している MPI_Bcast()，MPI_Allreduce()，MPI_Alltoall()，MPI_Wait() 関数など，内部においてメッセージ受信処理をともなう関数群においても，先行実行が可能となる．

特に，MPICH における MPI_Bcast()，MPI_Allreduce() のようなリダクション関数は，**図 4** に示すような規則的な送受信アルゴリズムによって実現されているため，あるノードが通信する相手は決まっており，予測のための規則性という観点において有利となっている．

一方，アイドルを発生させないための手法として，non-blocking 通信が使用される．これは，メッセージ受信処理が実行されたときに，要求したメッセージが到着してない場合でも，アイドルとならずに処理を継続するものである．たとえば，**図 5** において，処理 A 完了後に non-blocking 通信関数である MPI_Irecv() が実行されたとき，要求したメッセージが到着してない場合でも，アイドルとならず処理 B を実行する．しかし，このような non-blocking 通信を使用した場合でも，NPB などで多く見られるように，MPI_Irecv() において要求したデータが必要な処理（図 5 の処理 C）の前に MPI_Wait() や MPI_Waitall() などによって同期をとることが多い．結果的に，このときにアイドルが発生し，処理 C はメッセージの到着後に実行することとなる．今回の実装によって MPI_Wait() 関数を超えた先行実行が可能となるため，この MPI_Wait() においてアイドル状態が発生した場合においても，処理 C まで先行実行することが可能となり，より粗い粒度の先行実行が可能となる．

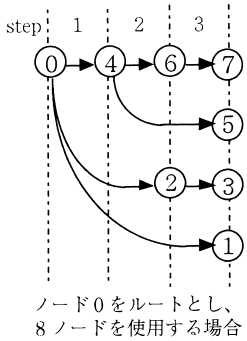

図 4 MPI_Bcast の送信アルゴリズム
Fig. 4 Algorithm of the MPI_Bcast.

処理 A
MPI_Irecv();
処理 B
MPI_Wait();
処理 C

図 5 non-blocking 通信に対する効果
Fig. 5 Effectiveness for non-blocking communication.

3. 評　　価

本手法の効果を調べるために，NASA により開発され，広く使用されている並列・分散計算機用ベンチマーク NAS Parallel Benchmark（以下，NPB）プログラム[12]を使用して評価した．評価内容は，

(1) 予測率/予測成功率
(2) 速度向上率
(3) アイドル時間の変化
(4) データ規模による速度向上率の変化

の 4 項目とした．なお，本評価に先立ち，E3500 上において NPB プログラムを用い，ソースプログラムの

表 1 システム構造
Table 1 Configuration of evaluation systems.

	CPU	Memory
E3500	Ultra SPARC-II(400MHz) × 8	3 GB
WSクラスタ	UltraSPARC-II(300MHz) × 2	512MB
	UltraSPARC(200MHz) × 2	320MB
	UltraSPARC-IIi(270MHz)	128MB
	UltraSPARC(167MHz)	128MB

解析および送受信メッセージの記録・分析[5),11)]などの初期評価を行っており，その結果もあわせて説明する．

以下，特に断わらない限り，NPB のデータセットは CLASS W を使用，E3500 および WS クラスタのシステム構成は**表 1** のとおりである．WS クラスタについては，各計算機が 100 Base-TX およびスイッチングハブによって接続されている．

また，プログラムを並列処理した場合に発生する実行時間のブレによる影響を抑えるため，それぞれの評価において，各ベンチマークプログラムを 20 回以上実行し，その結果をもとに予測成功率や実行時間などを求めている．なお，NPB の実行時間を計測するために用いた MPI_Wtime() の精度，すなわち MPI_Wtick() の値は 1 μ 秒である．

3.1 予測率および予測成功率

各プラットフォームにおける予測率および予測成功率を**表 3**，**表 4** にそれぞれ示す．ただし，表中の E3500 における予測アルゴリズムは直前予測（2.2.2 項 a)）を用い，WS クラスタ 1 は直前予測，WS クラスタ 2 はマルコフ予測（同 e)）を用いた結果である．

ここで，予測率は，

$$\text{予測率} = \frac{\text{予測回数}}{\text{メッセージ受信回数}} \times 100\,[\%]$$

として求めたもので，全メッセージ受信回数中で，予測/先行実行を行った回数を表す．なお，各ベンチマークプログラムにおいてやりとりされる全メッセージ受信回数を**表 2** に示す．

同様に，予測成功率は，

$$\text{予測成功率} = \frac{\text{予測成功回数}}{\text{予測回数}} \times 100\,[\%]$$

として求めた．以下に，今回用いたベンチマークプログラムそれぞれについて，予測率および予測成功率の関係をベンチマークの特徴とともに詳細に述べる．

3.1.1 CG

CG は共役勾配法によって正値対称な大規模疎行列の最小固有値を求めるプログラムである．表 3 より，このベンチマークの予測率はきわめて小さい．このことから，CG においてはアイドルがほとんど発生していないこと，すなわち，予測およびそれに続く先行実行はほとんど行われていないことが分かる．CG の初期評価結果からも，メッセージ送受信の発生には規則性があること，アイドル時間が非常に短いものが多いことが分かっており，この評価結果はこれらの初期評価結果を反映したものとなっている．

表 2 全メッセージ数
Table 2 Number of messages.

IS	SP	CG	EP	MG
2,322	38,623	23,618	115	35,108

（単位：回）

表 3 予測率
Table 3 Prediction ratio.

	E3500	WSクラスタ 1	WSクラスタ 2
IS	4.46	3.55	3.87
SP	0.08	0.05	0.04
CG	0.004	0.008	0.004
EP	40.37	30.13	28.95
MG	3.48	3.19	2.49

（単位：%）

表 4 予測成功率
Table 4 Success ratio.

	E3500	WSクラスタ 1	WSクラスタ 2
IS	1.46	1.38	6.64
SP	8.00	6.67	0.00
CG	25.00	0.00	0.00
EP	10.33	8.51	1.33
MG	1.43	2.34	2.19

（単位：%）

また，予測成功率に関しては，E3500 の CG は他のベンチマークプログラムに比べて最も大きくなっているが，この値は全 4 回予測中 1 回の予測に成功したことを示しており，予測の回数自体が少ないことから，性能への影響はほとんどないと考えられる．

3.1.2 SP

SP はスカラー 5 重対角方程式の求解を行うプログラムである．このベンチマークについては，CG に次いで予測率が低くなっている．この理由については CG と同様と考えられる．よって，予測成功率も比較的高い値となっているが，CG 同様性能への影響は少ないと考えられる．SP を用いた初期評価結果からも，SP のアイドル時間が短いことが分かっており，また，メッセージには規則性があるがその予測には複雑な手法が必要との結果を得ている．

3.1.3 EP

EP は乗算合同法による一様乱数，正規乱数の生成を行うプログラムである．このプログラムの予測率は最も高くなっており，アイドル時間が頻繁に発生

していることが分かる．また，予測成功率も比較的高い値となっている．一方，表 2 より，EP で発生するメッセージはほかに比べて最も少ないことも分かる．これは，EP において使用されいてる MPI 関数が MPI_Allreduce() 4 回のみであることが理由である．同時に，MPI_Allreduce() のみが使用されていることから，2.3.2 項で述べたような予測のための規則性が発生し，予測成功率が高いことにつながっている．

3.1.4 IS

IS は大規模整数ソートを行うプログラムである．この予測率は，今回使用したベンチマークプログラムの中で 2 番目に高い値となっており，比較的アイドル時間が発生していることが分かる．一方，予測成功率に関しては，直前予測のアルゴリズムを使用した E3500 および WS クラスタ 1 が 1.4%程度と低い値となっているのに対し，マルコフ予測では約 6.6%となっている．IS における MPI 関数の発生系列は，MPI_Allreduce()〜MPI_Alltoall()〜MPI_Alltoallv() の 3 つの関数の流れを 1 つのイテレーションとして，ループを繰り返すという構造になっており，これによって，直前予測のアルゴリズムでは予測は難しく，マルコフ予測が適していることが分かる．一方，本実装/評価で使用した MPICH における MPI_Alltoall() および MPI_Alltoallv() の実装は，各ノードが全ノードに対して，ほぼ同時にメッセージの送受信を行うため，ノードの配置状況やノード間を接続するネットワークの混雑状況などによりメッセージの到着順序が一定とならず，このことが予測成功率の向上に障害となっている．

3.1.5 MG

MG はマルチグリッド法を実現するプログラムである．このベンチマークプログラムでは，表 2 より全受信メッセージが非常に多く，その中で予測率も IS に次いで高い値となっている．実際に予測した回数も，他のベンチマークプログラムに比べて 10 倍以上と最も多く，非常にアイドルの発生しやすいプログラムとなっていることが分かる．また，予測成功率はそれほど高い値ではないが，この原因としては，実際に受信するメッセージサイズが毎回変化する（ただし，MPI_Recv() の引数によって指定されるサイズは一定である）ため，2.2.1 項で述べたメッセージ全体を 1 つとして予測の成功/失敗を判定することが問題となっている．しかし，メッセージサイズの変化は，受信回数 138 回ごとの周期を持っていることが分かっており，メッセージの予測/判定をメッセージの項目ごとに行うことによって，成功率の向上を図れる可能性がある．

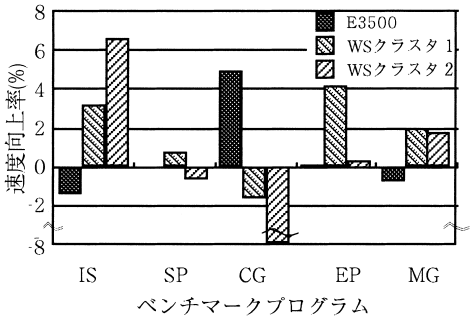

図 6 速度向上率
Fig. 6 Speed-up ratio.

3.2 速度向上率

図 6 に，E3500 および WS クラスタにおいて実行した NPB プログラムの総実行時間における速度向上の割合を示す．特徴的な点について，以下に述べる．

1) CG は，予測成功率の値によって速度の向上/低下がはっきりしている．予測が成功した場合の速度向上率は比較的大きいが，WS クラスタ 1/2 の予測成功率はゼロであり，速度は低下している．なお，WS クラスタにおける予測成功率は 1，2 ともにゼロであるが，向上率は大きく異なっている．これは，マルコフ予測よりも，直前予測の方が，予測するために必要となる履歴の記録時間が短くて済むため，オーバヘッドも少なくなることが原因である．

2) SP も CG 同様に予測回数/予測成功率が少ないため，実行速度にも大きな影響は見られない．

3) EP は，前節で述べた予測率/予測成功率はともに高い値となっているにもかかわらず，実行速度に大きな変化は見られない．EP において使用される MPI 関数として MPI_Allreduce() が 4 回発生すると前節で述べたが，これらの関数が実行されるのは，EP の処理の主要な部分がすべて実行された後（すなわちプログラムの終了直前）であるため，プログラム全体の実行速度に大きな影響を与えなかったことが理由である．

4) IS ではマルコフ予測の成功率が最も高くなっており，これにともなって速度向上率も全体で最も高い約 6.8%となった．一方，直前予測を用いた E3500 と WS クラスタ 1 に関しては，予測成功率がほぼ同一であるにもかかわらず，速度向上率は逆になる結果となった．

E3500 と WS クラスタの最も特徴的な違いとして，ノード間を接続するネットワークのアーキテクチャがある．各ノードは，E3500 では 2.6 GB/s の Gigaplane バスによって結合されている一方，WS クラスタは 100 Mbps の Ethernet によって結合されている．この

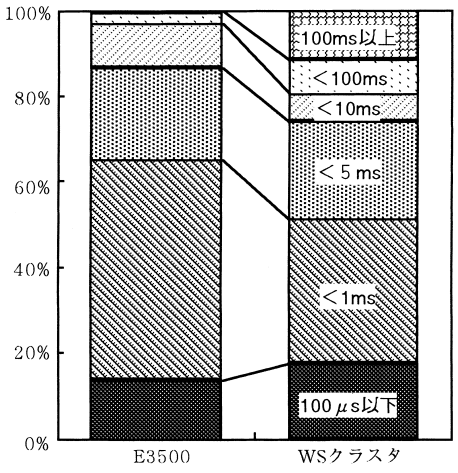

図7 ISにおけるアイドル時間の分布
Fig. 7 Distribution map of idle time on IS.

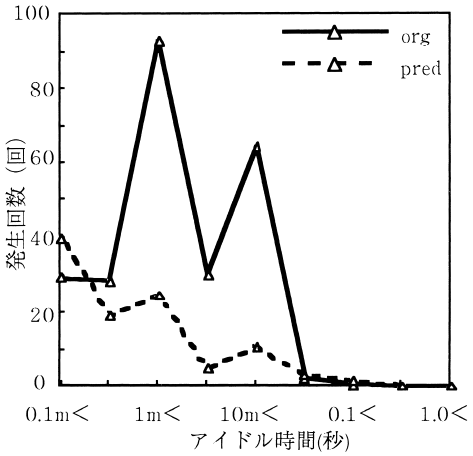

図8 ISにおけるアイドル時間長とその発生回数
Fig. 8 Number of idle length on IS.

ように，WS クラスタに比べて E3500 はきわめて高速なネットワークで接続されているため，メッセージの転送に要する時間，すなわち，メッセージ受信時に発生するアイドル時間も短くなる．**図7** の IS におけるアイドル時間の分布に示すように，IS では 1 m 秒以上のアイドル時間が，E3500 では全体の 35%を占めているのに対し，WS クラスタでは全体の半分となる 49%を占めている．このことからも，E3500 のアイドル時間は WS クラスタのアイドル時間に比べて短いことが分かる．

今回の実装は，アイドル時間が存在した場合，そのアイドル時間の長さにかかわらず予測/先行実行を行うため，E3500 のようにアイドル時間が短い場合，予測/先行実行の処理（実測値で 0.75〜0.80 m 秒程度）を行っている途中にメッセージが到着し，その確認は先行実行終了後となるため，予測が外れていた場合にはオーバヘッドが大きくなる．

以上のことから，本手法は，ノード間通信がプロセッサ性能に対して遅い場合に，より有効となることが分かる．

5) MG では，予測成功率と対応して速度向上率が上昇していることが明らかである．

全体的に，前節で述べた各ベンチマークプログラムの予測成功率が高いものほど，速度向上率も高くなっている．

3.3 アイドル時間の変化

図8 に，E3500 の評価の中から IS におけるアイドル時間の長さとその発生回数を示す．横軸はアイドル時間の長さを，縦軸はそのアイドル時間の長さが発生した回数を示しており，アイドル時間が発生しなかった場合は図中に含まれていない．たとえば，図より，実装前（org）は 1 m 秒から 5 m 秒までの長さのアイドル時間の発生回数が約 93 回と最も多いことが分かる．

実装前と実装後（pred）を比較すると，全体的にアイドル時間の発生回数が減少していることが分かる．アイドルが発生した全回数においても，実装前の 249 回に対し，実装後は 104 回と，半分以下に減少している．さらに，図より，アイドル時間が発生した場合でもその時間が減少していることが分かる．このように，アイドル発生の回数/アイドル時間の長さという 2 つの観点から，本手法によってアイドル時間の有効利用が達成されていることが分かる．

表5 データ規模による速度向上率の変化
Table 5 Speed-up ratio on E3500.

	Class S	Class W
IS	−21.98	−1.38
SP	−0.21	0.01
CG	0.19	5.02
EP	−0.20	0.03
MG	−14.29	−0.72

（単位：%）

3.4 データ規模による実行速度の変化

表5 に，E3500 上で実行した NPB のデータの規模（NPB の class）と速度向上率の関係を示す．表より，全体的にデータ規模の小さい Class S（ample）より，さらにデータ規模の大きい Class W（orkstation）の方が速度向上率は高くなっていることが分かる．本手法はメッセージを予測対象としており，従来行われていた分岐予測[4]よりも，1 回の予測による処理の粒度が大きい．このことから，データ規模が大きくなるほど予測による先行実行可能な処理のサイズが大きく，速

度向上も大きくなる．本実験結果からも，本手法は規模の小さなデータよりも，ある程度の規模を持つデータにおいて効果を発揮することが分かる．

4. 考　　察

これまでに得られた評価結果から，本手法が有効となるアプリケーションの特徴をまとめると，以下の 5 点となる．

(1) アイドル時間が存在すること

本手法は，アイドル時間を利用して，次に到着するメッセージを予測/先行実行するため，アイドル時間が存在することは必須である．また，アイドル時間の長さとしては，ある程度（本実験では 1 m 秒程度）以上必要であることが分かるが，この長さは WS クラスタにおいてアイドル発生回数の約半分を占めることからも，実用上妥当な値であることを実験的に証明した．

(2) 通信の発生が実行中に分散していること

3.2 節の 3) で述べた EP のように，通信の発生が偏っている場合にも本手法は有効にはたらくが，その効果は小さなものとなる

(3) 主要な処理がループで実現されていること

IS のように，3 つの処理を 1 つのイテレーションとして，これをループによって繰り返し処理を行っている場合，MG のように受信内容に周期がある場合などは，マルコフ予測によって予測成功率を上昇させることが可能である．

(4) ノード間のデータ転送速度

本手法は，WS クラスタのようなネットワーク速度の遅い分散環境において特に有効であることが実験から分かった．Ethernet を用いた WS/PC クラスタのような分散環境は，専用バスを用いた並列計算機に比べ，非常に低コストで実現可能であり，今後ますます普及していくものと考えられる．このような環境において，全体の処理性能を向上させるのに本手法が有効であることから，本手法が今後普及する可能性は高いと考えられる．

(5) データ規模

3.4 節で述べたとおり，本手法は，メッセージを予測対象としているため，1 つの予測あたりの処理粒度が大きく，予測による処理の速度向上も大きくなる．よって，今後，ますます大規模化していくアプリケーションに対して，効果を発揮するものと期待できる．

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である受信メッセージ予測法について簡単に説明し，MPI に対する実装方法と評価・考察を述べた．結果をまとめると，以下のとおりとなる．

(1) これまでの実装および今回の Sun Enterprise 上の MPI およびワークステーションクラスタ上の MPI に対する実装から，本手法はプラットフォームに依存しない手法であることを実証した．またその実装も，ハードウェアの変更をともなうことなく，ソフトウェアレベル（ライブラリレベル）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) 今回実装の対象とした MPI の関数はブロッキング受信関数の MPI_Recv() であるが，波及効果により MPI に含まれる多くの関数群に対しても，性能が向上することを示した．

(3) 評価として，予測率や速度向上率などを実験的に求め，各ベンチマークプログラムの特徴とあわせてその結果を検証した．その結果として，本手法により，最高で 6.8%の速度向上を達成した．

(4) ノードのアイドル発生回数およびその時間をそれぞれ減少させ，ノードを有効活用することが可能であることを実験的に明らかにした．

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，今後普及する可能性があることなどが確認できた．

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），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 Comparison 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) 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).
6) 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).
7) Kim, J.-S. and Lilja, D.J.: Characterization of Communication Patterns in Message-Passing Parallel Scientific Application Programs, *CANPC '98*, pp.202–216 (1998).
8) 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).
9) 足立涼子，岩本善行，大津金光，吉永　努，馬場敬信：異なるプラットフォームにおける受信メッセージ予測法の性能評価，情報処理学会研究会報告（SWoPP'2000），00-ARC-139, pp.73–78 (2000).
10) 平澤将一，松本　尚，平木　敬：ソフトウェア高レベルデータ値予測方式の予備評価，情報処理学会研究会報告（SWoPP'2000），00-CPSY-3 (2000).
11) 岩本善行，大津金光，吉永　努，馬場敬信：受信メッセージ予測法における予測方式の検討，情報処理学会論文誌，Vol.41, No.9, pp.2582–2591 (2000).
12) http://www.nas.nasa.gov/NAS/NPB

(平成 12 年 8 月 31 日受付)
(平成 13 年 2 月 1 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．同年より，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．2001 年 4 月より，栃木県立那須清峰高等学校教諭．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．

足立　涼子

1999 年宇都宮大学工学部情報工学科卒業．現在，同大学大学院工学研究科情報工学専攻博士前期課程在学中．並列計算機アーキテクチャに興味を持つ．

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．高性能計算システム，特にマルチスレッド化と実行時最適化システムに興味を持つ．

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等に興味を持つ．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 賞受賞．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．