Vol. 40 No. 5 情報処理学会論文誌 May 1999

メモリベース通信を用いた高速 MPI の実装と評価

森本健司[†] 松本尚[†] 平木敬[†]

本論文は，並列メッセージパッシングライブラリ MPI の共有メモリモデルに基づく高速実現 MPI/MBCF の実装方式と性能評価結果を述べる．MPI/MBCF では，共有メモリの特性を利用した *write* および *eager* の 2 種のプロトコルを混合して用いる．*write* プロトコルでは遠隔メモリ書き込みを用いてメッセージのバッファリングを必要としない通信を実現し，*eager* プロトコルではメモリベース FIFO を用いてライブラリに求められるメッセージのバッファリングを実現した．これら 2 つのプロトコルは送信および受信関数の先行関係により自律的かつ動的に切り替えられる．MPI/MBCF の性能をワークステーションクラスタ上で評価した．基本性能として round-trip time および peak bandwidth を測定し，実アプリケーションでの性能を調べるために NAS Parallel Benchmarks を実行した．評価の結果から，共有メモリ通信機能であるメモリベース通信を用いてメッセージパッシングライブラリを実現することの有効性が示された．

Implementation and Evaluation of a High Performance MPI Library with the Memory-Based Communication Facilities

Kenji Morimoto,[†] Takashi Matsumoto[†] and Kei Hiraki[†]

This paper describes an efficient implementation of the Message Passing Interface (MPI) library based on the shared memory model. Our implementation, called MPI/MBCF, combines two protocols to utilize shared memory communication facilities; the *write* protocol and the *read* protocol. In the *write* protocol, the remote write is used for communication with no buffering. In the *eager* protocol, the Memory-Based FIFO is used for buffering by the library. These two protocols are switched autonomously according to the precedence of send and receive functions. The performance of our library was evaluated on a cluster of workstations. We measured the round-trip time and the peak bandwidth, and executed the NAS Parallel Benchmarks. The results show that it is efficient to construct a message passing library with the MBCF which is based on the shared memory model.

1. はじめに

分散メモリ型並列計算機環境での通信モデルとして，メッセージパッシングモデルと共有メモリモデルとが広く用いられている．メッセージパッシングモデルではタスクの間に通信路を設け，この通信路に対してメッセージを send, receive することで通信を行う．このモデルはデータの転送媒体であるプロセッサ間通信ネットワークに着目し，プロセッサ間通信ネットワークをタスク間の通信路として仮想化したものである．

一方，共有メモリモデルでは，タスク間で共有するアドレス空間を設け，このアドレス空間に対する read, write☆により通信を行う．このモデルは通信対象であるメモリ空間に着目し，通信を遠隔プロセッサへのメモリアクセスとして仮想化したものである．

通信モデルとして，これらのモデルは一方により他方がエミュレート可能であり，相互に同等の表現力を持つ．従来はこの 2 つのうち，通信の実現方法としてメッセージパッシングモデルの方が効率的であるという主張がなされ，多くの通信ライブラリがメッセージパッシングモデルに基づいて設計・実装された．しかしながら，latency や bandwidth に代表される通信性能の向上には，プロセッサにおける命令オーバヘッドの減少，MMU やキャッシュメモリ等メモリを対象としたアーキテクチャ的サポートの利用が不可欠である．共有メモリモデルに基づく通信はこれらのサポートをより直接的に活用することが可能である．このことは共有メモリモデルに基づく通信がメッセージパッシングモデルに基づく

† 東京大学大学院理学系研究科情報科学専攻
Department of Information Science, Faculty of Science, University of Tokyo

☆ これらの操作は必ずしもマシン語命令の load, store による細粒度メモリアクセスを意味するわけではない．

通信に対して性能的に優位であることを示唆する．

以上の考察により我々は，高速な共有メモリ通信機能を用いたメッセージパッシング通信の実装は，メッセージパッシング通信機能を用いた実装に比べてより高い性能を得られることを推論する．本研究では，メッセージパッシングモデルでのプログラミングの標準的なインタフェイスである MPI（Message Passing Interface）Ver. 1.2[8),9)] の全関数・全機能を，低コストでリモートタスクのメモリにアクセスすることのできるソフトウェアメモリベース通信機能（MBCF: Memory-Based Communication Facilities）[7)] を用いて MPI/MBCF として汎用超並列オペレーティングシステム SSS–CORE 上に実装した．メッセージパッシングモデルには「リモートアドレス」の概念がないため，MPI に共有メモリ通信操作を直接的に適用することはできない．このため我々は，リモートアドレスの通知を低オーバヘッドで導入することにより，共有メモリ通信操作をメッセージパッシング通信に適用可能とした．

本論文では，まず 2 章で MBCF の概要を示し，3 章において MPI/MBCF における 1 対 1 通信の実現方式を述べる．4 章では MPI/MBCF の基本性能を評価する．5 章では NAS Parallel Benchmarks を用いた性能評価結果を示す．6 章において関連研究に触れ，7 章でまとめる．

2. ソフトウェアメモリベース通信 MBCF

2.1 MBCF の特徴

MBCF は高機能分散共有メモリシステム Strategic Memory System[6)] に対応する機能をソフトウェアで実現した遠隔メモリアクセス機構である．MBCF の特徴を以下に挙げる．

(1) **通信先メモリへの直接的な操作が可能**
通信先の固定された通信用バッファへメッセージを送るのではなく，通信先のユーザレベル仮想アドレス空間に対して read, write を実行して通信を行う．このため，システムによる余分なメッセージコピーを減らすことができる．

(2) **高速な保護・仮想化**
プロセッサ内部のページ管理機構・アドレス変換機構を用いてメモリの保護およびアクセスの仮想化を行う．これにより，プロセッサクロックレベルのスピードでの保護・仮想化を実現する．

(3) **汎用の通信ハードウェアを使用**
MBCF は多くの分散メモリ型並列計算機（MPP）が持つような専用ネットワークハードウェアを仮定せず，汎用ネットワークハードウェアを用いてソフトウェア的に実現される．ただし，高速な処理を行うために，以下に示すハードウェア機能をプロセッサが持つことが好ましい．
- 低オーバヘッドのアドレス空間切り替え
- 複数コンテクストが混在できる TLB
- ページエイリアス機能
- 物理アドレスタグを持つ高速なプロセッサキャッシュ

これらの機能は今日の高性能マイクロプロセッサの多くが実現している．

(4) 通信先での操作が高機能
通信対象がユーザレベルの仮想アドレス空間であり，パケット受信側が割り込みハンドラ内でソフトウェア的にパケットを処理するため，遠隔メモリへの単純な read, write のみならず，swap, FIFO write, fetch and add といった高機能処理を行うことができる．

(5) 通信の到着・順序を保証
MBCF システムが通信パケットの到着保証・順序管理を行うため，ユーザが再送制御等を行う必要がない．

2.2 MBCF の機能

MBCF の機能のうち，本論文で述べる MPI の実装 MPI/MBCF では遠隔メモリ書き込みおよびメモリベース FIFO を用いた．

遠隔メモリ書き込みはデータをリモートタスクのアドレス空間に直接書き込む機能である．データの送信側ユーザがシステムコールを呼び出しパケットを発行する．パケットには，受信側のホスト ID・タスク ID とともに，受信側でデータを展開するアドレス☆が記される．パケットが受信側の通信ハードウェアに到着した後，MBCF の割り込みハンドラにより通信ハードウェアのバッファからこのアドレスが指す領域へとデータがコピーされる．遠隔メモリ書き込みシステムコールはノンブロッキングであるが，ユーザは書き込みの終了・成否を表すフラグを得るために，明示的な ack を要求することができる．

メモリベース FIFO はユーザが自分のアドレス空間に確保した領域をリモートタスクから書き込まれる FIFO（リングバッファ）とする機能である．ユーザが確保した領域を FIFO として登録するためのシステムコール，送信側がリモートタスクの FIFO にデータを書き込むためのシステムコール，および受信側がローカルにある

☆ このアドレスは受信側タスクの仮想アドレス空間での値である．

FIFO からデータを読み出すためのシステムコールが提供される．リモートタスクの FIFO への書き込みは前述の遠隔メモリ書き込みの 1 つのモードとなっており，受信側割り込みハンドラが FIFO の境界管理を行う．遠隔メモリ書き込みと同様，FIFO への書き込みシステムコールはノンブロッキングであり，ユーザは明示的な ack を要求することができる．メモリベース FIFO はユーザのメモリ資源が許す範囲内で複数用意することができる．メモリベース FIFO は，ユーザレベルで直接アクセスすることのできる高機能なメッセージ通信と捉えることができる．

2.3 MBCF の性能

ワークステーションを 100BASE-TX の Hub で接続した環境での MBCF の性能を以下に示す．性能測定に使用した機器は，ノードワークステーションとして Axil 320 model 8.1.1（Sun SPARCstation20 互換機，85 MHz SuperSPARC CPU × 1），ネットワークインタフェイスとして Sun Microsystems Fast Ethernet SBus Adapter 2.0（各ワークステーションに搭載），およびネットワークとして SMC TigerStack 100 5324TX（ノンスイッチング Hub）および Bay Networks BayStack 350T（スイッチング Hub）である．OS としてワークステーションクラスタ版 SSS–CORE Ver. 1.1a を使用し，2 ノード間での遠隔書き込みの one-way latency および peak bandwidth を測定した．

One-way latency は，送信側における書き込み要求システムコールの直前から受信側における書き込みの完了までの時間であり，実験では送信側に ack が返ってくるまでの時間を測定し ack のための時間を差し引くことにより算出した．**表 1** は遠隔メモリ書き込み（MBCF_WRITE）およびメモリベース FIFO 書き込み（MBCF_FIFO）の one-way latency をデータサイズを変えて測定したものである．測定にはノンスイッチング Hub を使用した．

Peak bandwidth は，遠隔書き込みを連続して行った時の転送性能である．**表 2** はノンスイッチング Hub（半二重）およびスイッチング Hub（全二重）での MBCF_WRITE および MBCF_FIFO の peak bandwidth をデータサイズを変えて測定したものである．

遠隔メモリ書き込みの one-way latency 24.5 μs, peak bandwidth 11.48 MB/s（半二重ネットワーク）および 11.93 MB/s（全二重ネットワーク）という値は 100BASE-TX のハードウェア性能（最大 bandwidth 12.5 MB/s）を効率良く引き出した値であり，専用の内部相互結合網を持つ分散メモリ型並列計算機とほぼ同等の性能を持つことが示された[5]．

3. 1 対 1 通信関数の実装

MPI Ver. 1.2 で定められている関数群のうち，1 対 1 通信の基本となるノンブロッキング標準モード送信関数 `MPI_Isend()` およびノンブロッキング受信関数 `MPI_Irecv()` の MPI/MBCF における実装に関して詳述する．その他の通信関数の多くはこの 2 関数を利用して実装されるか，もしくは類似の方式により実装される．

3.1 MPI 標準の要請

まず，MPI 標準において送受信が満たすべき要件がどのように定められているかを簡単にまとめる．

送受信の相手を特定するには，通信を行う集団（およびコンテクスト）を規定するコミュニケータとその集団の中での通し番号であるランクとを指定する．同じ送受信の組の中でのメッセージの識別にはタグが用いられる．ある送信と受信とが対応するとは，

(1) コミュニケータが一致し，かつ
(2) コミュニケータの中でのランクが対応し，かつ
(3) タグが一致する

ことを意味する．ただし，受信側が指定するランクおよびタグにはワイルドカード `ANY` の使用が認められている．

ライブラリはメッセージの到着を保証しなければならない．順序に関しては，コミュニケータ等が異なる場合を除きメッセージの追い越しを禁止しなければならない．すなわち，

- ある送信元から同じ送信先を指定した 2 つのメッセージが発行され，それらが同じ受信に対応する場合，先に発行されたメッセージが受信される
- ある受信側で 2 つの受信が発行され，それらが同じメッセージに対応する場合，先に発行された受信が満たされる

ことを保証しなければならない．さらに，上記の条件で禁止されない追い越しに関しては禁止してはならず，通信の進行を保証しなければならない．

送信が発行され，かつ対応する受信が発行されていない場合の挙動により，送信は 4 つに分類される．すなわち，

- ライブラリが提供するバッファ領域を用いて送信を完了させる（ユーザの送信用バッファを解放する）標準モード
- 送信側ユーザが用意するバッファ領域を用いて送信を完了させるバッファモード

表 1 100BASE-TX における MBCF の one-way latency（単位：μs）
Table 1 One-way latency of MBCF with 100BASE-TX (in microseconds)

data-size (bytes)	4	16	64	256	1024
MBCF_WRITE	24.5	27.5	34.0	60.5	172.0
MBCF_FIFO	32.0	32.0	40.5	73.0	210.5

表 2 100BASE-TX における MBCF の peak bandwidth（単位：Mbytes/s）
Table 2 Peak bandwidth of MBCF with 100BASE-TX (in Mbytes/s)

data-size (bytes)	4	16	64	256	1024	1408
MBCF_WRITE, 半二重	0.31	1.15	4.31	8.56	11.13	11.48
MBCF_FIFO, 半二重	0.31	1.14	4.30	8.53	11.13	11.45
MBCF_WRITE, 全二重	0.34	1.27	4.82	9.63	11.64	11.93
MBCF_FIFO, 全二重	0.34	1.26	4.80	9.62	11.64	11.93

- 対応する受信の発行を待つ同期モード
- 受信の先行発行を予期するためエラーとなるかもしれないレディモード

である．標準モードにおいてライブラリが提供すべきバッファ領域のサイズには下限はなく，バッファを提供せずに標準モードを同期モードのように扱うことも許される．ただし，ライブラリが提供するバッファ領域が小さいあるいは存在しない場合，送信と受信の順序によってはプログラムがデッドロック状態に陥る可能性がある．

3.2 実装方式

前節で記した MPI 標準での要請に基づきながら，ノンブロッキング標準モード送信関数 `MPI_Isend()` およびノンブロッキング受信関数 `MPI_Irecv()` を MBCF 上で実装する方式について述べる． MPI では通信関数の呼び出し形式にリモートアドレスの指定が含まれないため，共有メモリ通信操作を通信に適用するためには送信バッファのアドレスを受信側に通知する（ないしは受信バッファのアドレスを送信側に通知する）必要がある． MPI/MBCF では受信用バッファのアドレスを送信側に通知することで遠隔メモリ書き込みを適用可能とする．

3.2.1 遠隔メモリ書き込みによる通信

送信関数ではまず，受信側からの送信要求により受信バッファのアドレスを通知されているかどうかを調べる．送信関数が呼ばれた時点で送信側が受信バッファのアドレスを通知されている場合，**図 1** に示すように MBCF の遠隔メモリ書き込みにより通信を実現する．実線の矢印は送信関数により駆動されるデータの移動を表す．この場合， MPI ライブラリによるバッファリングは行われず，効率の良い通信が可能となる．

この通信を実現するために受信関数では，受信関数が呼ばれた時点で通信データが到着していない場合に，送信側に送信要求を送り受信バッファのアドレスを通知する．通知は送信側に用意されたメモリベース FIFO に対して行う．このメモリベース FIFO は後述する通信データのバッファリングのための FIFO とは別に設け，また，送信要求の発行元プロセス毎に別の FIFO を用意する． FIFO を複数設けることで種類の異なるメッセージの混在を避けることができ，ライブラリによる FIFO の操作が容易になる．

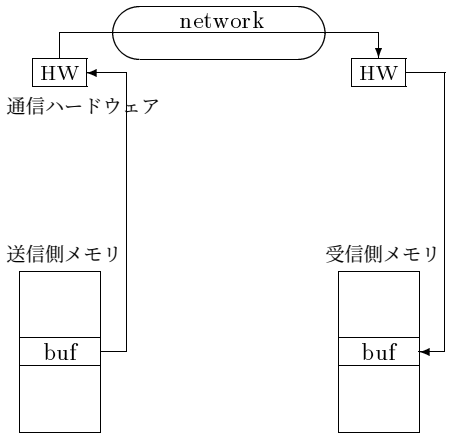


図 1 バッファリングを伴わない通信
Fig. 1 Communication with no buffering

3.2.2 メモリベース FIFO による通信

送信関数の呼び出しが受信関数の呼び出しに先行する場合，送信側は受信側が指定する受信バッファのアドレスを知ることができない．この場合には標準モードの送信は同期モードの送信のように受信の発行を待ち，受信バッファのアドレスの通知を待って通信を行うという実装も可能である☆．しかし MPI 標準では，送信が先行した場合にもあらかじめライブラリが用意した領域に通信データをバッファリングすることで，対応する受信の

☆ 送信バッファのアドレスを受信側に通知し，受信側が遠隔メモリ読み出しにより通信を行う実装もこれと本質的に同じである．

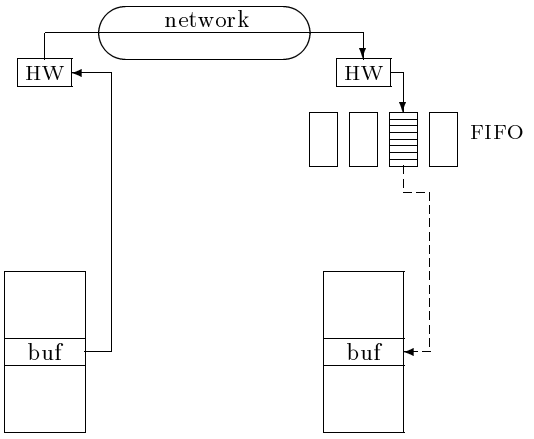


図 2 バッファリングを伴う通信
Fig. 2 Communication with single buffering

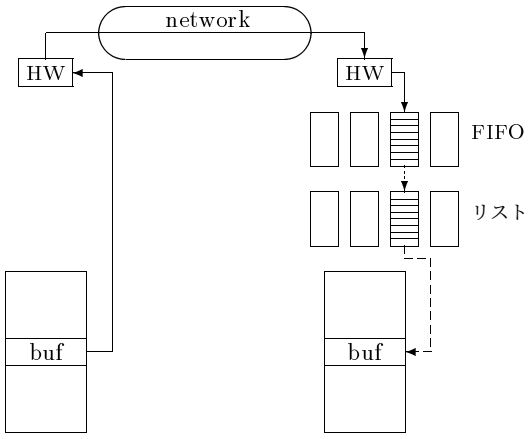


図 3 2 段階のバッファリングを伴う通信
Fig. 3 Communication with double buffering

発行を待たずに送信が完了するような実装を推奨している．通信効率の観点からも受信の発行を待たずに通信を開始する方がよい．

MPI/MBCF では標準モードの送信に対し，必要に応じて受信側でのデータのバッファリングを行う．バッファリングは固定領域で行う必要があるため，メモリベース FIFO により通信データのバッファリングを実現する．受信側は送信元プロセス毎に別の FIFO を用意し，送信側が自分の送るメッセージの入るべき FIFO を指定する．異なる送信元からのメッセージが 1 つの FIFO に混在しないため，受信側ライブラリによる FIFO の操作が容易になる． MPI 標準では，異なる送信元からの複数のメッセージの扱いに関しては，処理の順序および公平性について何も要求しない．よって FIFO からメッセージを取り出す際に FIFO 間で調停を行う必要はなく， FIFO を複数用意することは性能低下の要因とはならない．**図 2** は受信側のメモリベース FIFO を用いてバッファリングを行う場合の通信の様子である．図中，実線の矢印は送信関数によるデータの移動を，破線の矢印は受信関数によるデータの移動を表す．

3.2.3 交差する通信のためのバッファリング

送信側がメッセージを送り出す順番と受信側がメッセージを受け取る順番とが一致する場合には，上記の 2 つの方法で 1 対 1 通信を処理することが可能である．しかし，タグまたはコミュニケータが異なる 2 つのメッセージは 3.1 節に記した通り追い越しが認められており，送受信関数呼び出しの対応が交差することがある．受信側 FIFO にこのようなメッセージが送られた場合，受信関数が FIFO からメッセージのヘッダを読み出した後に，そのメッセージが現在扱っている受信と対応しない送信のものであることが判明する．受信関数ではこの対応しないメッセージを一旦 FIFO から読み出して退避させ，後続のメッセージヘッダを FIFO から読み出す．

FIFO から取り出したもののその時点ではユーザの受信バッファに送ることのできないメッセージを管理するために，リスト構造を持たせたバッファを用意する．リストは FIFO 同様に送信元毎に設ける．このリストを介した受信の様子を**図 3** に示す．実線の矢印は送信関数によるデータの移動を，点線の矢印はメッセージに対応しない受信関数によるデータの移動を，破線の矢印は対応する受信関数によるデータの移動を表す．この方式が用いられる場合，受信側でのメッセージのコピー回数が増加し，さらにリスト構造を管理するオーバヘッドが加わるため，通信性能は図 1，図 2 の場合に比べ低下する．しかし，多くの MPI アプリケーションでは送信側と受信側とで通信操作の順番が対応しており，この方式が用いられる頻度は低い．

3.2.4 実装方式のまとめ

以上に述べた実装方式をまとめると，ノンブロッキング標準モード送信 `MPI_Isend()` は MPI/MBCF において以下のように実装される．

(1) 受信側からの送信要求を調べる．送信と対応するものがあれば遠隔メモリ書き込みにより受信バッファへデータを転送し，終了．対応するものがなければ (2) へ．
(2) 受信側のメモリベース FIFO に対しメッセージ（ヘッダおよびデータ）を転送し，終了．

ノンブロッキング受信 `MPI_Irecv()` は以下のように実装される．

(1) メモリベース FIFO から既に取り込まれたメッ

セージのリスト（このリストは先行する受信の(2)のステップにおいて形成される）を調べる．受信と対応するものがあればリストから受信バッファへデータをコピーし，終了．対応するものがなければ(2)へ．

(2) メモリベース FIFO からメッセージのヘッダを取り込み，受信との対応を調べる．ヘッダが対応するならばメッセージのデータを FIFO から受信バッファへ取り込み，終了．対応しないならばメッセージのデータを FIFO から処理待ちのメッセージのリストへ取り込み，(2)へ．FIFO が空ならば(3)へ．

(3) 送信側に送信要求を送り，終了．

メッセージと送信要求にはそれぞれ通し番号を付与し，送信要求が最新のメッセージ到着状況を反映しているかどうかを送信側において調べる．これにより，対応するメッセージと送信要求とが同時に発行された場合には送信側において送信要求を棄却し，送信と受信とを正しく対応させることを可能とする．送信要求は棄却される可能性があるため，送信要求を出した受信の全てが必ず遠隔メモリ書き込みによりデータを受け取るとは限らない．したがって，受信操作の(2)においてメモリベース FIFO から取り込まれたヘッダはまず，以前に発行されたペンディング中の受信のリストに対して比較される必要がある．

受信関数において送信元に `ANY` が指定されている場合，この受信のための送信要求は発行されない．さらに，この受信が満たされるまで後続の受信による送信要求は保留される．

3.3 1 対 1 通信の実行の流れ

前節ではデータの移動を中心に MPI/MBCF の 1 対 1 通信関数の実装方式を説明した．既に述べられている通り，メッセージの送信に遠隔メモリ書き込み（MBCF_WRITE）を用いるかメモリベース FIFO 書き込み（MBCF_FIFO）を用いるかは，対応する送信関数 `MPI_Isend()` と受信関数 `MPI_Irecv()` との先行関係によって決定される．本節ではこの先行関係による場合分けを基に，送信側および受信側の実行の流れを中心に MPI/MBCF での 1 対 1 通信の進行を説明する．

3.3.1 送信が受信に先行する場合

この場合，**図 4** に示す通り，送信関数によるメッセージの送信にはメモリベース FIFO が用いられ，遅れて発行された受信関数はこのメッセージを FIFO から取り出し受信バッファに格納する．この通信の流れは，MPI ライブラリの代表的な実装である MPICH[4] において *eager* プロトコルと呼ばれる通信パターンに対応する．

3.3.2 受信が送信に先行する場合

この場合，**図 5** に示す通り，受信関数がまずメモリベース FIFO により送信要求を送信側に送り，遅れて発行された送信関数がこの送信要求に応えて遠隔メモリ書き込みにより受信バッファにメッセージを書き込む．この通信の流れを以下では *write* プロトコルと呼ぶ．

3.3.3 送信と受信とが同時に発行される場合

この場合，**図 6** に示す通り，送信側・受信側ともにまず自分が先行しているかのように振る舞う．すなわち，送信側はメモリベース FIFO 書き込みによりメッセージを転送し，受信側は送信要求を発行する．この時点でメッセージが受信側に送られるため，実行は *eager* プロトコルに従う必要がある．受信側では，受信を完了させようとする `MPI_Wait()` ないしは後続の受信関数により FIFO からメッセージが取り出され受信バッファにコピーされる．一方，送信側では，後続の送信関数により FIFO から送信要求が取り出されることになるが，この送信要求に付与された番号は古い値であるため要求は棄却される．

4. 基本性能の評価

4.1 評価環境

3 章で述べた方式により実装された MPI ライブラリ MPI/MBCF の送受信関数の性能を評価するため，2.3 節と同じ環境において送受信の round-trip time および peak bandwidth を測定した．

MBCF が提供されている OS としてワークステーションクラスタ版 SSS–CORE Ver. 1.1a を用いた．比較のため，同一の機器の上で OS として SunOS 4.1.4 を，MPI の実装として MPICH Ver. 1.1[4] を用いた場合の値を測定した．ただし，デバイスドライバの制約により SunOS 4.1.4 を用いた場合にはネットワークは半二重に固定されている．MPICH は Argonne National Laboratory および Mississippi State University において開発された MPI の実装であり，ワークステーションクラスタに対する実装ではメッセージパッシング通信である TCP ソケットによる通信を行う．

本論文で述べた実装において導入された *write* プロトコルの有効性を検証するために，受信が送信に先行しても送信要求メッセージを発行せず，よって遠隔メモリ書き込みを使用しない実装を併せて用意した．以下では送信要求を用いる実装を SR と，送信要求を用いない実装を NSR と略記する．

4.2 Round-trip time

まず，メッセージサイズを変えながらノンスイッチ

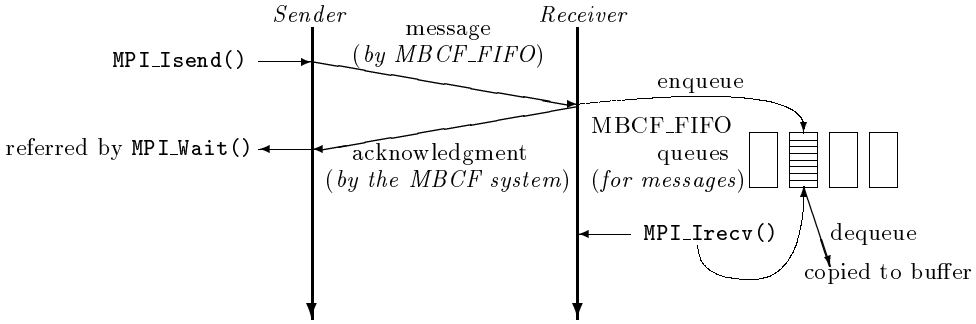


図 4 送信が受信に先行する場合の実行の流れ（*eager* プロトコル）
Fig. 4 Execution sequence where send precedes receive (*eager* protocol)

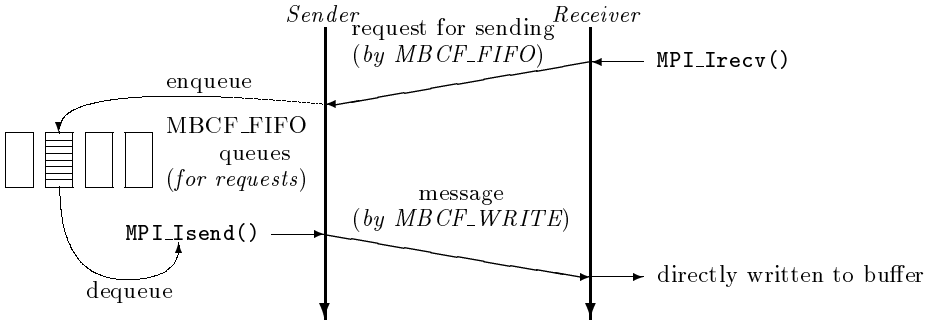


図 5 受信が送信に先行する場合の実行の流れ（*write* プロトコル）
Fig. 5 Execution sequence where receive precedes send (*write* protocol)

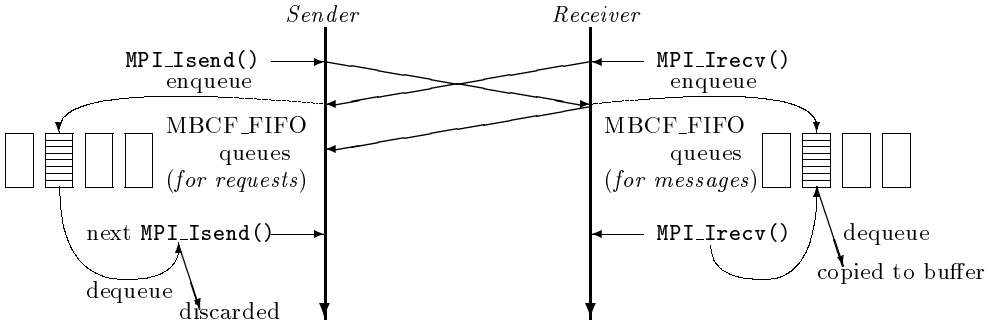


図 6 送信と受信とが同時に発行される場合の実行の流れ
Fig. 6 Execution sequence where send conflicts with receive

ング Hub 上で round-trip time を測定した．2 つの MPI プロセスが各々前もってノンブロッキング受信 `MPI_Irecv()` を起動し，一方は

(1) `MPI_Send()`（ブロッキング送信）
(2) `MPI_Wait()`（受信完了待ち）

を，他方は

(1) `MPI_Wait()`（受信完了待ち）
(2) `MPI_Send()`（ブロッキング送信）

を実行する．**表 3** に前者のプロセスにおける送信の開始から受信の完了までを round-trip time として測定した結果を示す．なお，SSS-CORE 上の MPI/MBCF では精度 0.5 μs のシステムクロックが利用可能なため 1 回のメッセージの往復毎に時間を測定しその最小値を用いた．一方，SunOS 上の MPICH では精度 10 μs のより粗い測定だけが可能なため，メッセージ 1024 往復に要した時間の平均値を用いた．

MPI/MBCF での通信遅延が，常に TCP 上の MPICH の遅延を下回っている．SR と NSR とを比較した場合，SR では受信が先行した場合には *write* プロトコルが適用されてメモリベース FIFO 書き込み

表 3 100BASE-TX における MPI 送受信の round-trip time（単位：μs）
Table 3 Round-trip time of MPI with 100BASE-TX (in microseconds)

message size (bytes)	0	4	16	64	256	1024	4096
SR	71	85	85	106	168	438	1026
NSR	112	137	139	154	223	517	1109
MPICH	968	962	980	1020	1080	1255	2195

より高速な遠隔メモリ書き込みを用いることができ，さらにメモリコピーの回数が 1 回減るため，SR の方が遅延が小さくなっている．

SR でのメッセージサイズ 0 byte 時の round-trip time 71 μs という値を MBCF 遠隔メモリ書き込みの one-way latency 24.5 μs（メッセージサイズ 4 byte 時）と比べると，MPI ライブラリを実現するための付加的なオーバヘッドが小さいことが分かる．

4.3 Peak bandwidth

次に，メッセージサイズを変えながらノンスイッチング Hub 上およびスイッチング Hub 上で peak bandwidth を測定した．2 つの MPI プロセスのうち一方のプロセスは送信を繰り返し，もう一方は受信を繰り返す．前者において送信を開始する直前から最後に 2 つのプロセスの間で同期をとるまでを通信時間とし，総メッセージ転送量をこれで割ったものを peak bandwidth として測定した．**表 4** はメッセージサイズを 4 byte から 1 Mbyte まで変化させた時の各方式の peak bandwidth である．このうち，メッセージサイズ 4 Kbyte までに関して測定した値をグラフにしたものが**図 7** である．SRH, NSRH は半二重ネットワーク（ノンスイッチング Hub）上での SR, NSR の peak bandwidth を，SRF, NSRF は全二重ネットワーク（スイッチング Hub）上での SR, NSR の peak bandwidth をそれぞれ表す．

一般にメッセージサイズが小さいと送受信操作のオーバヘッドのため bandwidth は低いが，MPI/MBCF はメッセージサイズに対する peak bandwidth の立ち上がりが MPICH/TCP と比較して早く，メッセージサイズが小さくても bandwidth を得やすいことが示された．これは，MPI/MBCF では細粒度通信が必要なアプリケーションや通信の統合が充分になされていないアプリケーションにおいても通信効率を保つことを示している．

半二重ネットワークにおいては NSR の peak bandwidth の値は SR の値をわずかに上回っている．なぜなら，SR では受信が先行した場合ないしは受信と送信とが同時に発行された場合に *write* プロトコルに従おうとして発行される送信要求のメッセージが，データを含むメッセージの通信の妨げとなるためである．全二重ネットワークにおいては送信要求メッセージはデータ通信に干渉しないため，SR と NSR との間に bandwidth の差は見られない．

SR の peak bandwidth 10.15 MB/s（半二重ネットワーク）および 11.86 MB/s（全二重ネットワーク）という値は，100BASE-TX のハードウェア性能の限界である 12.50 MB/s やこれを用いた際の MBCF の性能 11.48 MB/s, 11.93 MB/s に近い値である．

5. NAS Parallel Benchmarks による性能評価

5.1 NAS Parallel Benchmarks

NAS Parallel Benchmarks（NPB）は，NASA Ames Research Center において開発された航空力学数値シミュレーションプログラムを基にした，並列計算機向けのベンチマークである．問題とそれを解くアルゴリズム，問題サイズのクラス，プログラミングモデルを定めた NPB 1.0[1] と，MPI を用いた実際のプログラムを提供する NPB 2.x[2] とがある．以下の 5 つのカーネルプログラムおよび 3 つの計算流体力学（CFD）アプリケーションからなっている．

- カーネル
  - **EP** 乗算合同法による正規乱数の生成
  - **MG** 簡略化されたマルチグリッド法による 3 次元ポアソン方程式の解法
  - **CG** 共役勾配法による正値対称疎行列の最小固有値問題の解法
  - **FT** 高速フーリエ変換による 3 次元偏微分方程式の解法
  - **IS** 大規模整数ソート
- CFD
  - **LU** Symmetric SOR による LU 分解
  - **SP** 非優位対角なスカラ五重対角方程式の解法
  - **BT** 非優位対角な 5×5 ブロックサイズの三重対角方程式の解法

NPB 2.x では IS のみ C + MPI で，IS 以外は Fortran90 + MPI で記述されている．8 つの問題それぞれに関して問題サイズが小さい方から順に class S（サンプル），class W（小規模ワークステーションクラスタ向け），class A（中規模ワークステーションクラス

表 4 100BASE-TX における MPI 送受信の peak bandwidth（単位：Mbytes/s）
Table 4 Peak bandwidth of MPI with 100BASE-TX (in Mbytes/s)

message size (bytes)	4	16	64	256	1024	4096	16384	65536	262144	1048576
SRH	0.14	0.53	1.82	4.72	8.08	9.72	10.15	9.78	9.96	10.00
NSRH	0.14	0.54	1.89	4.92	8.54	10.21	10.34	10.43	10.02	9.96
SRF	0.14	0.57	1.90	5.33	10.22	11.68	11.77	11.85	11.85	11.86
NSRF	0.15	0.59	1.98	5.51	10.58	11.70	11.78	11.81	11.82	11.82
MPICH	0.02	0.09	0.35	1.27	3.54	6.04	5.59	7.00	7.77	7.07

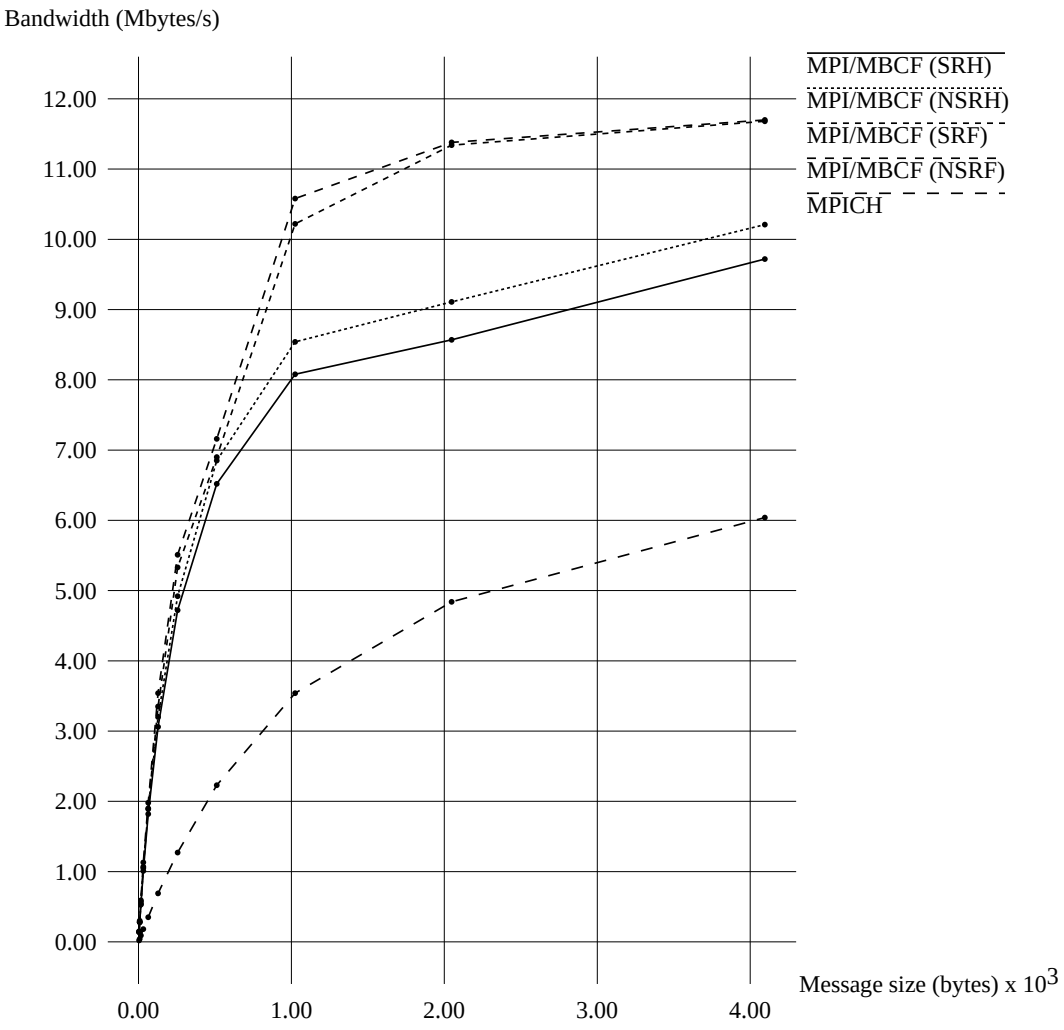


図 7 100BASE-TX における MPI 送受信の Peak bandwidth
Fig. 7 Peak bandwidth of MPI with 100BASE-TX

タ向け），class B（中規模並列計算機向け），class C（大規模並列計算機向け）というクラス分けがなされている．

5.2 評価環境

2.3 節および 4.1 節と同じワークステーションクラスタ環境で，SSS–CORE 上の MPI/MBCF を用いて NPB Rev. 2.3 を実行し，ワークステーションの台数を変化させて実行時間を測定した．ネットワークにはスイッチング Hub を用い，半二重モードで測定した．これは全二重モードでの通信を行えない SunOS に合わせ

て条件を揃えるためである．比較のため，同一の機器の上で SunOS 4.1.4 上の MPICH Ver. 1.1 を用いた場合の実行時間を測定した．

コンパイラとして gcc-2.7.2.3 および g77-0.5.21 を用いた．8 つの問題のうち FT のプログラムは g77 ではコンパイルできないため評価の対象から省いた．

問題サイズとして class W を用いた．これは，class A のプログラムが，SSS–CORE 上では動作したものの SunOS 4.1.4 上ではメモリ不足により実行できなかったためである．

4 章と同様に，MPI/MBCF のうち送信要求を用いる実装を SR と，送信要求を用いない実装を NSR と略記する．

5.3 評価結果

まず**表 5** に，7 つのベンチマークプログラムの特性を示す．これらは MPI/MBCF のソース中に計数コードを挿入し，ノード数 8（SP, BT は 9），送信要求ありの条件の下でプログラムを実行し，測定したものである．表の項目のうち，通信データレートは MPI 関数によって送信されたデータのバイト数（ヘッダ部分は除く）を全ノードに関して合計し，実行時間で割ったものである．通信メッセージレートは送信されたメッセージの個数を全ノードに関して合計し，実行時間で割ったものである．遠隔メモリ書き込み利用率は，全ノードから送信されたデータのうち，対応する受信が先行していたために MBCF の遠隔メモリ書き込みにより転送されたもののデータ量（バイト数）の割合である．

表 6 に EP の実行結果を示す．2^{26} 個の乱数を求めている．EP では通信は実行結果の最終的な収集においてのみ発生し，実行時間のほとんどは浮動小数点数の演算である．このため，この結果は通信性能ではなくワークステーションの浮動小数点数演算性能を表していると言える．

表 7 に MG の実行結果を示す．問題サイズは $64 \times 64 \times 64$，イテレーション数は 40 である．分割境界でのデータのやりとりのため，数百 byte から数 Kbyte 程度のメッセージの 1 対 1 通信が非常に頻繁に実行される．このため，メッセージサイズが小さい場合にも効率良く通信を行うことのできる MPI/MBCF が MPICH に大きく優っている．ほぼ全ての受信関数において送信元としてワイルドカード `MPI_ANY_SOURCE` が指定されているため，SR において送信要求を発行することができず，遠隔メモリ書き込みの利用が妨げられている．送信元を静的に指定するようにプログラムを書き換えることにより SR ではさらなる性能向上が見込まれる．

表 8 に CG の実行結果を示す．問題サイズは 7000，イテレーション数は 15 である．リダクション操作のための集団通信を 1 対 1 通信関数で記述したメッセージサイズ数十 Kbyte 程度の通信，および同じくメッセージサイズ数十 Kbyte 程度の通常の 1 対 1 通信が実行される．通信の頻度は MG ほど高くはなく，また，メッセージのサイズが MG より大きいため，全般的に性能は MG でのものより向上している．さらに SR では，遠隔メモリ書き込みを全データの半分以上に対して適用することで通信の効率を上げていることが分かる．

表 9 に IS の実行結果を示す．2^{20} 個の整数のソートを 10 イテレーション行っている．各イテレーションで 1 Mbyte 程度のメッセージを全対全の集団通信関数により交換している．各ノードでの計算時間が短いため，台数が増えるにつれこの集団通信の時間が支配的となるが，MPI/MBCF では集団通信関数を受信起動が先行するように実装しているため SR において効率的に通信を行っていることが示される．

表 10 に LU の実行結果を示す．問題サイズは $33 \times 33 \times 33$，イテレーション数は 300 である．メッセージサイズ数百 byte 程度の 1 対 1 通信でデータをやりとりしている．メッセージサイズは小さいものの通信頻度が高くないため，MPI/MBCF, MPICH ともに台数効果が出ている．LU においても MG 同様に受信関数の送信元指定に `MPI_ANY_SOURCE` が用いられており，SR での遠隔メモリ書き込みの利用の機会を減らしている．

表 11 に SP の実行結果を示す．問題サイズは $36 \times 36 \times 36$，イテレーション数は 400 である．メッセージサイズ数十 Kbyte 程度の 1 対 1 通信が呼ばれる．通信の頻度は低く，通信パターンが受信先行となっているため，特に SR で大きな台数効果を得ている．

表 12 に BT の実行結果を示す．問題サイズは $24 \times 24 \times 24$，イテレーション数は 200 である．数 Kbyte から数十 Kbyte 程度のメッセージを 1 対 1 通信でやりとりしている．SR で遠隔メモリ書き込みが多く使われているが，多少の通信遅延の増加は隠蔽されるような通信・計算パターンのため SR, NSR ともに大きな台数効果を得ている．

5.4 評価のまとめ

まず MPI/MBCF と MPICH とを比較すると，全プログラムを通じて MPI/MBCF による実行効率が MPICH の実行効率を上回っている．特に，小さなメッセージを頻繁に通信し合う場合（MG）に MPICH がオーバヘッドの大きさから性能を大きく落とすため，MPI/MBCF と MPICH との差が大きくなっている．

表 5 NPB ベンチマークプログラムの特性
Table 5 Characteristics of NPB programs

プログラム	EP	MG	CG	IS	LU	SP	BT
通信データレート (Mbytes/s)	0.00	9.68	12.69	13.58	1.89	7.83	5.32
通信メッセージレート (個/s)	4	4670	2138	466	1199	421	488
遠隔メモリ書き込み利用率 (%)	51.10	0.01	53.33	99.22	13.37	49.01	47.24

表 6 NPB EP の実行時間（単位：秒）
Table 6 Execution time of NPB EP (in seconds)

ノード数	1	2	4	8
SR [speed-up]	121.14 [1.00]	60.51 [2.00]	30.30 [4.00]	15.15 [8.00]
NSR [speed-up]	121.15 [1.00]	60.59 [2.00]	30.30 [4.00]	15.15 [8.00]
MPICH [speed-up]	125.56 [1.00]	60.61 [2.07]	32.13 [3.91]	16.25 [7.73]

表 7 NPB MG の実行時間（単位：秒）
Table 7 Execution time of NPB MG (in seconds)

ノード数	1	2	4	8
SR [speed-up]	37.34 [1.00]	22.61 [1.65]	14.05 [2.66]	7.44 [5.02]
NSR [speed-up]	37.32 [1.00]	22.62 [1.65]	14.05 [2.66]	8.01 [4.66]
MPICH [speed-up]	38.81 [1.00]	31.30 [1.24]	21.01 [1.85]	13.72 [2.83]

表 8 NPB CG の実行時間（単位：秒）
Table 8 Execution time of NPB CG (in seconds)

ノード数	1	2	4	8
SR [speed-up]	69.16 [1.00]	37.69 [1.83]	20.94 [3.30]	11.24 [6.15]
NSR [speed-up]	69.13 [1.00]	38.54 [1.79]	21.44 [3.22]	11.81 [5.85]
MPICH [speed-up]	68.75 [1.00]	40.01 [1.72]	27.79 [2.47]	14.59 [4.71]

MPI/MBCF の実装のうち NSR はメモリベース FIFO による *eager* プロトコルのみを用いており，MPI の実装方式として MPICH に近い．よって，同一の機器の上で NSR の実行効率が MPICH の実行効率を上回ることから，MPI の実装において SunOS 上の TCP に対して SSS–CORE 上のメモリベース FIFO が優位であることが示される．

さらに，MPI/MBCF の実装のうち SR と NSR とを比較すると，通信パターンが受信先行となる場合（CG, IS, LU, SP）には SR が遠隔メモリ書き込みを用いて効率良く通信を行っている．実験に用いた半二重ネットワークにおいては SR ではデータの通信がプロトコルメッセージにより妨害される可能性があるにもかかわらず，実験結果からは全てのプログラムにおいて SR は NSR と同等ないしはそれ以上の実行効率を得ていることが分かる．このことから，本論文で提案した *eager* プロトコルと *write* プロトコルとの併用が，基本性能に関してだけではなく実アプリケーションの実行においても有効であることが示される．NSR はメッセージ通信のみを用いた実装であり，これに対して SR が優位であることは，メッセージパッシングライブラリの実現において共有メモリモデルが有効であることを示している．

6. 関連研究

MPI ライブラリが提供するバッファを介さない MPI 通信の実装として，富士通 AP1000, AP1000+, AP3000 のリモートコピー機能である put, get を利用した MPIAP[10),11)] や Cray T3D の Shared Memory Access library を利用した CRI/EPCC MPI[3)] がある．これらはいずれも専用の通信網を持つ MPP での実装である．ポータブルな MPI の実装として広く用いられている MPICH では *get* プロトコルが用意されており，遠隔メモリ読み出しを利用することができる．いずれの実装においてもバッファリングを避ける通信では，まず送信側が特殊なメッセージを受信側に送り，それを受けて受信側が遠隔メモリ読み出しを実行するという送信側駆動・受信側マッチングのプロトコルを採用している．この方式ではデータの転送開始までに必ずパケットが 1 往復することになり，通信遅延が増大する．

本研究と同時期に実装がなされた MPI-EMX[12)] では EM-X のリモートメモリ書き込みを用いて *write* プロトコルを実現しており，`MPI_Irecv()` が先行する場合の通信効率を上げている．しかし，EM-X は AP3000 や T3D と同様に専用の通信網を持つ MPP であり，汎

表 9 NPB IS の実行時間（単位：秒）
Table 9 Execution time of NPB IS (in seconds)

ノード数	1	2	4	8
SR [speed-up]	10.16 [1.00]	6.35 [1.60]	4.51 [2.25]	2.90 [3.50]
NSR [speed-up]	10.16 [1.00]	6.35 [1.60]	4.69 [2.17]	3.72 [2.73]
MPICH [speed-up]	10.25 [1.00]	7.09 [1.45]	5.61 [1.83]	4.81 [2.13]

表 10 NPB LU の実行時間（単位：秒）
Table 10 Execution time of NPB LU (in seconds)

ノード数	1	2	4	8
SR [speed-up]	1034.09 [1.00]	537.23 [1.92]	289.65 [3.57]	164.55 [6.28]
NSR [speed-up]	1034.56 [1.00]	541.21 [1.91]	294.00 [3.52]	169.63 [6.10]
MPICH [speed-up]	1081.51 [1.00]	611.92 [1.77]	320.70 [3.37]	185.04 [5.84]

表 11 NPB SP の実行時間（単位：秒）
Table 11 Execution time of NPB SP (in seconds)

ノード数	1	4	9
SR [speed-up]	1277.42 [1.00]	352.34 [3.63]	153.96 [8.30]
NSR [speed-up]	1276.39 [1.00]	352.77 [3.62]	165.01 [7.74]
MPICH [speed-up]	1391.16 [1.00]	475.27 [2.93]	231.66 [6.01]

表 12 NPB BT の実行時間（単位：秒）
Table 12 Execution time of NPB BT (in seconds)

ノード数	1	4	9
SR [speed-up]	617.67 [1.00]	155.19 [3.98]	67.13 [9.20]
NSR [speed-up]	617.44 [1.00]	155.21 [3.98]	67.65 [9.13]
MPICH [speed-up]	627.29 [1.00]	214.14 [2.93]	96.02 [6.53]

用ハードウェア環境で共有メモリ通信メカニズムを用いて MPI ライブラリを実装する本研究の方式と異なっている．

7. ま と め

低コスト・高機能な共有メモリ通信メカニズムであるメモリベース通信を用いて MPI ライブラリ MPI/MBCF を実装した．メモリベース通信の機能の 1 つであるメモリベース FIFO を使用することで，MPI においてライブラリが提供すべきバッファ操作を高速に実現した．さらに，遠隔メモリ書き込み機能を適用することでバッファを介さない通信を可能とした．

通信の基本性能を 100BASE-TX で接続されたワークステーションクラスタにおいて測定し，round-trip time 71 μs, peak bandwidth 10.15 MB/s（半二重ネットワーク）および 11.86 MB/s（全二重ネットワーク）という値を得た．さらに並列アプリケーション NAS Parallel Benchmarks の実行では，通信メッセージサイズが小さなアプリケーションや受信の発行が対応する送信の発行に先行する通信パターンのアプリケーションにおいて，遠隔メモリ書き込み機能を用いない実装や SunOS 上の MPICH/TCP に比べて MPI/MBCF が効率良く通信を行えることが実証された．以上の結果から，共有メモリ通信機能であるメモリベース通信をメッセージパッシング通信ライブラリのベースとして用いることの有効性が示された．

本研究の実験での比較は，汎用ハードウェアを用いた全く同一の機器の上で行われたものであり，オペレーティングシステムおよび通信ライブラリを効率の良いものとすることでアプリケーションに変更を加えることなくその実行効率を大きく改善することが可能であることを示している．このことはさらに，専用の通信ハードウェアを導入することなく高性能なワークステーションクラスタを構築することの有効性を示すものである．

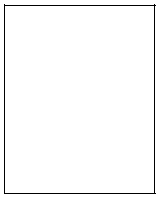
謝辞 本研究の一部は情報処理振興事業会（IPA）が実施している独創的情報技術育成事業の支援を受けた．

参 考 文 献

1) Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V. and Weeratunga, S.: THE NAS PARALLEL BENCHMARKS, Technical Report RNR-94-007, NASA Ames Research Center (1994).

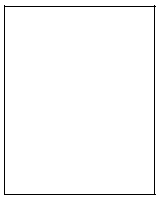
http://www.nas.nasa.gov/NAS/NPB/.
2) Bailey, D., Harris, T., Saphir, W., Wijngaart, R., Woo, A. and Yarrow, M.: The NAS Parallel Benchmarks 2.0, Technical Report NAS-95-020, NASA Ames Research Center (1995). http://www.nas.nasa.gov/NAS/NPB/.
3) Cameron, K., Clarke, L. and Smith, G.: CRI/EPCC MPI for CRAY T3D (1995). http://www.epcc.ed.ac.uk/t3dmpi/Product/.
4) Gropp, W., Lusk, E., Doss, N. and Skjellum, A.: A High-Performance, Portable Implementation of the MPI Message-Passing Interface Standard, *Parallel Computing*, Vol. 22, No. 6, pp. 789–828 (1996).
5) Matsumoto, T. and Hiraki, K.: MBCF: A Protected and Virtualized High-Speed User-Level Memory-Based Communication Facility, *Proc. of Int. Conf. Supercomputing*, pp. 259–266 (1998).
6) 松本尚, 平木敬: キャッシュインジェクションとメモリベース同期機構の高速化, 情報処理学会研究報告 93-ARC-101, Vol. 93, No. 71, pp. 113–120 (1993).
7) 松本尚, 平木敬: 汎用超並列オペレーティングシステム SSS–CORE のメモリベース通信機能, 第 53 回情報処理学会全国大会講演論文集 (1) , pp. 37–38 (1996).
8) Message Passing Interface Forum: MPI: A Message-Passing Interface Standard (1995). http://www.mcs.anl.gov/mpi/.
9) Message Passing Interface Forum: MPI-2: Extensions to the Message-Passing Interface (1997). http://www.mpi-forum.org/.
10) Sitsky, D. and Hayashi, K.: Implementing MPI for the Fujitsu AP1000/AP1000+ using Polling, Interrupts and Remote Copying, 並列処理シンポジウム JSPP '96 論文集, pp. 177–184 (1996).
11) Sitsky, D. and Mackerras, P.: System Developments on the Fujitsu AP3000, *Proc. of 7th Parallel Computing Workshop* (1997).
12) 建部修見, 児玉祐悦, 関口智嗣, 山口喜教: リモートメモリ書き込みを用いた MPI の効率的実装, 並列処理シンポジウム JSPP '98 論文集, pp. 199–206 (1998).

(平成10年9月7日受付)
(平成11年3月5日採録)



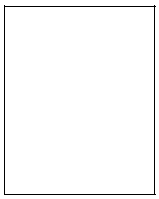
森本 健司

1997 年東京大学理学部情報科学科卒業. 1999 年同大学大学院理学系研究科情報科学専攻修士課程修了. 同年 4 月より同専攻博士課程に在籍. 並列分散計算に関する研究に従事. 他に並列化／最適化コンパイラ, 並列分散オペレーティングシステム, 並列計算機アーキテクチャに興味を持つ.



松本 尚（正会員）

1962 年生. 1985 年東京大学工学部計数工学科卒業. 1987 年大阪市立大学大学院理学研究科物理学専攻修士課程修了. 日本アイ・ビー・エム（株）東京基礎研究所研究員を経て, 1991 年 11 月より東京大学大学院理学系研究科情報科学専攻助手. 並列計算機アーキテクチャ, 並列分散オペレーティングシステム, 最適化コンパイラに関する研究に従事. 他に数値計算による制約解消系, グラフィックス, ニューラルネットワーク等に興味を持つ. 電子情報通信学会, 日本ソフトウェア科学会, ACM 各会員.



平木 敬（正会員）

1976 年東京大学理学部物理学科卒業. 1982 年同大学大学院理学系研究科物理学専攻博士課程修了. 理学博士. 1982 年通商産業省工業技術院電子技術総合研究所入所. 1988 年より 2 年間 IBM 社 T. J. Watson 研究センタ客員研究員. 1990 年より東京大学理学部情報科学科（現在大学院理学系研究科情報科学専攻）に勤務. 現在, 超並列アーキテクチャ, 超並列超分散計算, 並列オペレーティングシステム, ネットワークアーキテクチャなどの高速計算システムの研究に従事. 日本ソフトウェア科学会会員.