メモリベース通信を用いた高速 MPI の実装方式

森本 健司　　松本 尚　　平木 敬
{morimoto, tm, hiraki}@is.s.u-tokyo.ac.jp
東京大学大学院理学系研究科情報科学専攻

概要

メモリベース通信の機能である遠隔メモリ書き込みとメモリベース FIFO を用いて、 MPI (Message Passing Interface) の通信機能を実装した。メモリベース通信は汎用超並列オペレーティングシステム SSS–CORE において共有メモリを実現する通信機能である。この機能のうち、遠隔メモリ書き込みにより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、メモリベース FIFO により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。この実装の性能をワークステーションクラスタ上で評価し、共有メモリを実現する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を構築することの有効性を検証する。

Implementation of high performance MPI with the Memory-Based Communication Facilities

Kenji Morimoto, Takashi Matsumoto, and Kei Hiraki
Department of Information Science, Faculty of Science, University of Tokyo

Abstract

We implemented communication functions of the MPI (Message Passing Interface) standard, using *Remote Write* and *Memory-Based FIFO* of the Memory-Based Communication Facilities. The Memory-Based Communication Facilities realize the shared memory model on the SSS–CORE, a general purpose massively parallel operating system. Among those facilities, *Remote Write* is used for communication without message buffering, and *Memory-Based FIFO* for message buffering provided by the MPI library. This paper shows evaluation of the implementation on a cluster of workstations, and verifies whether it is effective to construct a message passing library with the Memory-Based Communication Facilities 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プログラミングパラダイムとして、共有メモリモデルとメッセージパッシングモデルとが広く用いられている。共有メモリモデルではタスク間で共有されるアドレス空間が存在し、タスク間の通信はこの共有されたアドレス空間への load, store[1] によって行われる。一方、メッセージパッシングモデルではタスクの間に明示的な通信路を設け、この通信路に対して通信すべきメッセージを send, receive することで通信が行われる。

これら 2 つのモデルは一方により他方を実現可能であるが、ライブラリとして提供され実現が容易であるメッセージパッシングモデルがこれまでのところ多く用いられている。しかし、プログラミングモデルとしてではなくシステム (ハードウェアおよびオペレーティングシステム) が提供する機能として考えた場合、通信の最適化や効率、柔軟性の観点で共有メモリモデルの方が原理的に優っている [9]。そこでは、分散メモリ型並列計算機システムとしてはまず、共有メモリモデルでのプログラミングを支援する高性能・高機能な通信機能を実現し、その上で必要に応じてメッセージパッシングモデルを用いる

[1] これらは必ずしもマシン命令レベルの細粒度のメモリアクセスを意味する訳ではない。

ためのランタイムルーチン等を構築する。

当研究室で開発が進められている汎用超並列オペレーティングシステム SSS–CORE [7] では、低コストでリモートのメモリにアクセスすることのできるメモリベース通信機能 (MBCF: Memory-Based Communication Facilities) [10] により、共有メモリモデルでのプログラムの実行を高速に行うことが可能である。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ースである MPI (Message Passing Interface) Ver. 1.1 [2] の通信関数を SSS–CORE 上で MBCF を用いて実装した。この実装の基本的な通信性能に関して評価を行うことにより、共有メモリモデル上に構築したメッセージパッシングモデルに基づくプログラムの実行効率を検証する。

本稿では、まず 2 章で MBCF の特徴および性能について述べる。3 章では MBCF を用いて MPI を効率的に実装するための手法を説明する。4 章ではワークステーションクラスタを用いた性能評価実験とその結果を示し、5 章でまとめる。

2 メモリベース通信 MBCF

2.1 MBCF の特徴

MBCF は、ハードウェアによる遠隔メモリアクセス実現機構 Memory-Based Processor [5] および高機能分散共有メモリシステム Strategic Memory System [6] を基に考案された、ソフトウェアによる遠隔メモリアクセス機能である。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。同時に、一般的なメモリ保護機構によるメモリの保護およびアクセスの仮想化が実現される。

2. 汎用の通信ハードウェアを用いる
   MBCF はソフトウェアにより実現されているため、多くの高並列計算機 (MPP) が持つような専用通信ハードウェアは不要である。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替え
   - 複数コンテクストの混在できる 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。

3. オーバヘッドが少ない
   MBCF をユーザレベルから使用するためのシステムコールや割り込みハンドラは、OS の他の機能のためのシステムコール・割り込みハンドラとは分離して実装される。このため、MBCF の処理に際して通信・同期に関係のない操作を行う必要がない。

4. 通信先での操作が高機能
   要求受信側が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 といった複合処理を行うことができる。

5. 通信の到着保証、順序保証がなされている
   システムにより通信パケットの消失・順序管理がなされており、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、今回の実装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のアドレス空間に直接書き込む。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[2]が記されている。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から書き込まれる

[2] 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 (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、リモートに用意された FIFO にデータを書き込むためのシステムコール、およびローカルな 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 の境界管理を行なう。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。

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。計測に使用した機器は、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ースとして 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 (Hub) として SMC TigerStack 100 5324TX 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 Round-trip time および Peak bandwidth を計測した。

Round-trip time は、書き込み要求システムコールの直前から書き込み完了が送信先から通知されるまでの時間として測定した。表 1 は 100Base-TX での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 Round-trip time をデータサイズを変えて計測したものである。

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 100Base-TX での MBCF_WRITE の Peak bandwidth をデータサイズを変えて計測したものである。

遠隔メモリ書き込みの Round-trip time 49 μs, Peak bandwidth 11.24 MB/s という値は 100Base-TX のハードウェア性能をほぼ使い切り、専用の内部相互結合網を持つ MPP とほぼ同等の性能を持つことが示された [8]。

3 1 対 1 通信関数の実装

MPI Ver. 1.1 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 つ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一致し、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。

ライブラリは通信の到着を保証しなければならない。順序に関しては

- 同じ受信に対応する 1 つの送信元からの 2 つの送信
- 同じ送信に対応する 1 つの送信先での 2 つの受信

の間の順序関係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。即ち、

- ライブラリが提供するバッファ領域を用いる標準モード
- 送信側ユーザが用意するバッファ領域を用いるバッファモード
- 対応する受信の発行を待つ同期モード
- エラーとなるかもしれないレディモード

表 1: 100Base-TX における MBCF の Round-trip time

data size (byte)	4	16	64	256	1024
MBCF_WRITE (μs)	49	55	68	121	344
MBCF_FIFO (μs)	64	64	81	146	421

表 2: 100Base-TX における MBCF の Peak bandwidth

data size (byte)	4	16	64	256	1024	1408
Peak bandwidth (Mbyte/s)	0.29	1.06	4.03	8.28	10.86	11.24

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[3]。

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()` を効率良く実装した。

送信関数ではまず、受信側から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行う。実線の矢印が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。

一方の受信関数では、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、受信プロセス毎に別の 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。

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

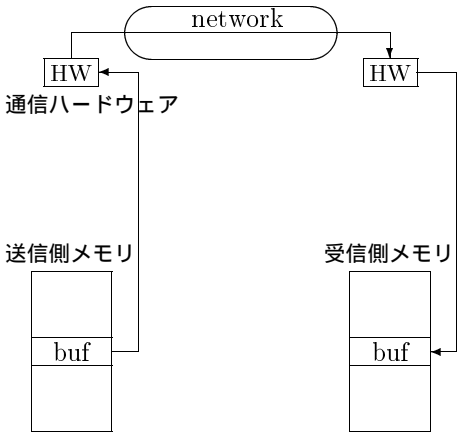

図 1: バッファリングを伴わない通信

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つ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、対応する受信の発行を待たずに送信が終了することが強く求められている。通信効率の観点からも受信の発行を待たずに送信を開始する方がよい。バッファリングは固定領域で行うため、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。送信プロセス毎に別の FIFO を用意し、送信側が自分の送るメッセージの入る FIFO を指定する。異なる送信元からのメッセージがひとつの 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[4]。

[3] 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、多くの不注意なプログラムがデッドロック状態に陥るおそれがある。

[4] 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 間で調停を行う必要はなく、FIFO を複数用意することは送信元を特定している限り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 を用いる場合の通信の様子である。図中、実線の矢印が送信関数によるデータの移動を、破線の矢印が受信関数によるデータの移動を表す。

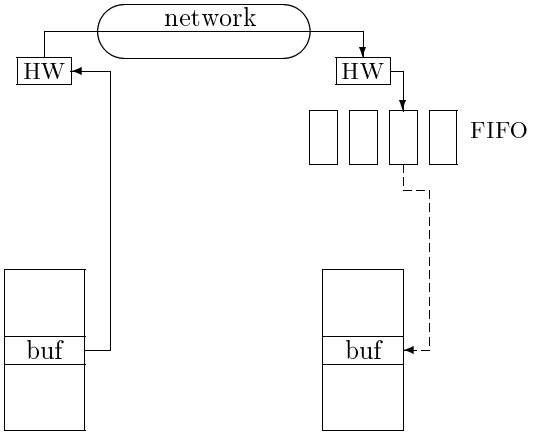

図 2: バッファリングを伴う通信

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ユーザの受信バッファに直接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。リストも 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。実線の矢印が送信関数によるデータの移動を、点線の矢印が対応しない受信関数によるデータの移動を、破線の矢印が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操作の順番が一致しており、この方式が用いられる頻度は低い。

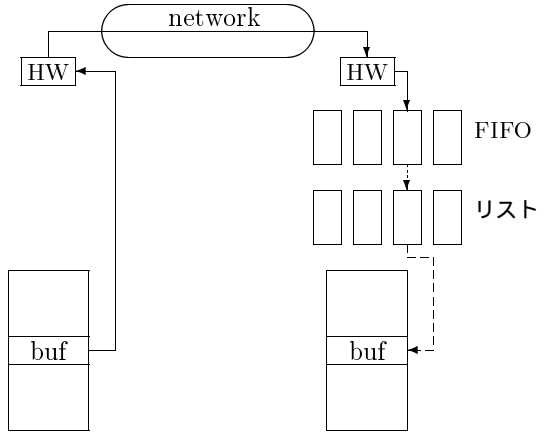

図 3: 2 段階のバッファリングを伴う通信

以上をまとめると、ノンブロッキング標準モード送信 `MPI_Isend()` は以下のように実装される。

1. 受信側からの送信要求の有無を調べる。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。なければ 2 へ。
2. 受信側のメモリベース FIFO に対しメッセージの転送を開始し、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 から取り込んだ送信メッセージのリストを調べる。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 から送信メッセージヘッダを取り込み、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を FIFO から受信バッファへ取り込み、終了。ヘッダが対応しないならばメッセージをリストへコピーし、引き続き FIFO を調べる。FIFO が空になったら 3 へ。
3. 送信側のメモリベース FIFO に送信要求メッセージを送り、終了。

4 実験および結果

3 章で述べた方式の MPI 送受信関数の性能を評価するため、2.3 節と同様の環境において送受信の Round-trip time および Peak bandwidth を計測し

た。ただし、 Hub によるネットワークは半二重のため、ネットワークが全二重となっている場合の Peak bandwidth を調べるためにネットワークスイッチとして Bay Networks BayStack 350T を併用した。

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 [3] を用いた場合の値を計測した。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 の実装である。ワークステーションクラスタに対する実装では TCP ソケットによる通信を行う。

今回の実装において、受信が送信に先行する場合に遠隔メモリ書き込みを用いることの影響を調べるために、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が使用されないような実装を併せて用意した。以降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。

まず、メッセージサイズを変えながら Hub 上で Round-trip time を測った。 2 つの MPI プロセスのうち一方のプロセスは

1. `MPI_Irecv()`
2. `MPI_Isend()`
3. 送信を完了させる `MPI_Wait()`
4. 受信を完了させる `MPI_Wait()`

を繰り返し、もう一方は

1. `MPI_Irecv()`
2. 受信を完了させる `MPI_Wait()`
3. `MPI_Isend()`
4. 送信を完了させる `MPI_Wait()`

を繰り返す。このメッセージのやりとりを 1024 回行い、 1 回の往復に要する時間の平均を Round-trip time として測定した。

表 3 はメッセージサイズを 0 byte から 1 Kbyte まで変化させた時の各方式の Round-trip time である。図 4 はこれをグラフに表したものである。 MBCF を用いて実装した MPI による通信遅延が、常に TCP 上の MPICH による遅延を下回っている。 MPICH ではメッセージサイズの増加に対する Round-trip time の増加が他に比べて小さいが、これは送受信に関わる操作のオーバヘッドが大きいために通信そのもののオーバヘッドが隠されるためである。 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メモリベース FIFO 書き込み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 に関しては精度 0.5 μs のシステムクロックを用いて詳細な測定を行った。メッセージサイズが 0 byte の時の Round-trip time が最小で 92 μs となった。この値を MBCF 遠隔メモリ書き込みの通信遅延 49 μs (メッセージサイズ 4 byte 時) と比べると、 MPI 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。

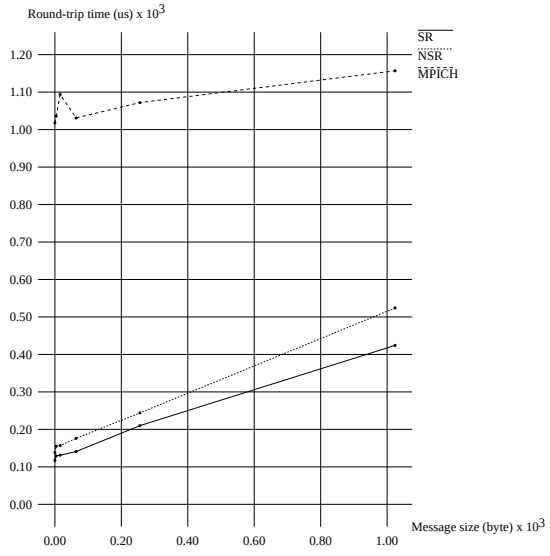

図 4: 100Base-TX における MPI 送受信の Round-trip time

次に、メッセージサイズを変えながら Hub 上およびスイッチ上で 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 `MPI_Isend()`, `MPI_Wait()` を繰り返し、もう一方は `MPI_Irecv()`, `MPI_Wait()` 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。ただし、通信遅延の隠蔽のために、メッセージ長に応じて複数の送信をオーバラップさせた。

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、 4 Kbyte までの値をグラフにし

表 3: 100Base-TX における MPI 送受信の Round-trip time

Message size (byte)	0	4	16	64	256	1024
SR (μs)	117	129	131	141	210	424
NSR (μs)	138	155	157	176	244	524
MPICH (μs)	1018	1036	1094	1031	1072	1157

表 4: 100Base-TX における MPI 送受信の Peak bandwidth

Message size (byte)	4	16	64	256	1024	4096	16384	65536	262144	1048576
SRH (Mbyte/s)	0.13	0.50	1.72	4.30	7.61	9.17	9.87	9.72	9.48	9.48
NSRH (Mbyte/s)	0.13	0.53	1.78	4.63	8.14	9.76	9.98	9.79	9.56	9.50
SRF (Mbyte/s)	0.13	0.53	1.85	5.27	10.23	11.09	11.21	10.61	11.08	11.21
NSRF (Mbyte/s)	0.14	0.54	1.82	5.14	10.41	11.14	11.23	10.62	11.08	11.22
MPICH (Mbyte/s)	0.02	0.08	0.34	1.21	3.37	5.76	5.33	6.67	7.41	6.75

たものが図 5 である。SRH, NSRH は半二重ネットワーク (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) 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 を得やすいことが分かる。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。これは、SR では受信が先行した場合に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。半二重ネットワークでの SR の Peak bandwidth は 9.9 MB/s となっており、これは 100Base-TX のハードウェア性能の限界である 12.5 MB/s やこれを用いた際の MBCF の性能 11.2 MB/s に近い値である。

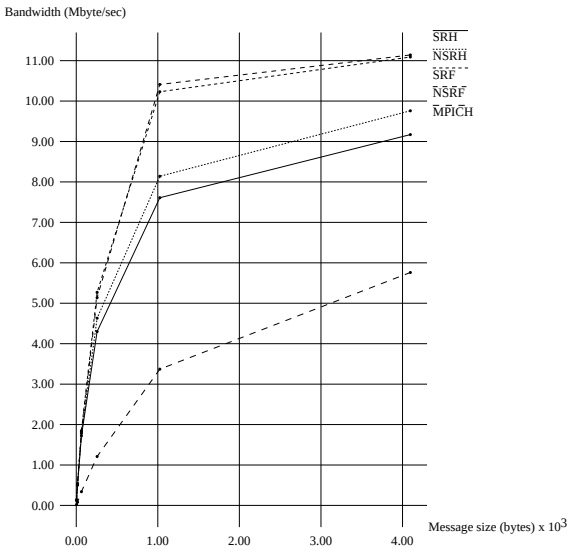

図 5: 100Base-TX における MPI 送受信の Peak bandwidth

5 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+ のリモートコピー機能である put, get を利用した実装 [4] や Cray T3D の Shared Memory Access library を利用した実装 [1] 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。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。

6 まとめ

低コスト・高機能な遠隔メモリアクセス機能であるメモリベース通信を用いて高速な MPI を実装した。メモリベース通信の機能の一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 92 μs, Peak bandwidth 9.9 MB/s という値を得た。これらの値より、共有メモリを実現する機構であるメモリベース通信をメッセージパッシング方式の実現のベースとすることの有効性が示された。

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、および新情報処理開発機構 (RWCP) の支援を受けた。

参考文献

[1] Kenneth Cameron, Lyndon J. Clarke, and Gordon Smith. CRI/EPCC MPI for CRAY T3D. `http://www.epcc.ed.ac.uk/t3dmpi/Product/`, September 1995.

[2] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. `http://www.mcs.anl.gov/mpi/`, May 1995.

[3] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A High-Performance, Portable Implementation of the MPI Message-Passing Interface Standard. *Parallel Computing*, Vol. 22, No. 6, pp. 789–828, September 1996.

[4] David Sitsky and Kenichi Hayashi. 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, June 1996.

[5] 松本尚, 平木敬. Memory-Based Processor による分散共有メモリ. 並列処理シンポジウム JSPP '93 論文集, pp. 245–252, May 1993.

[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. 計算機アーキテクチャ研究会報告 No.79-1, pp. 113–120. 情報処理学会, August 1993.

[7] 松本尚, 平木敬. 汎用並列オペレーティングシステム SSS-CORE の資源管理方式. 日本ソフトウェア科学会第 11 回大会論文集, pp. 13–16, October 1994.

[8] 松本尚, 平木敬. 100BASE-TX によるメモリベース通信の性能評価. コンピュータシステムシンポジウム論文集, pp. 101–108. 情報処理学会, November 1997.

[9] 松本尚, 平木敬. 共有メモリ vs. メッシージパッシング. 情報処理学会研究報告 97-ARC-126, Vol. 97, No. 102, pp. 85–90. 情報処理学会, October 1997.

[10] 松本尚, 駒嵐丈人, 渦原茂, 竹岡尚三, 平木敬. 汎用超並列オペレーティングシステム：SSS-CORE — ワークステーションクラスタにおける実現 —. 情報処理学会研究報告 96-OS-73, Vol. 96, No. 79, pp. 115–120. 情報処理学会, August 1996.