リモートメモリ書き込みを用いたMPIの効率的実装

建部修見[†] 児玉祐悦[†]
関口智嗣[†] 山口喜教[†]

MPIはpoint-to-point通信における対応する送信と受信のマッチングに関するコストが大きく，通信遅延が大きくなる原因となっている．本研究では，ノンブロッキング受信が先行発行される通信パターンにおいて，送信時に受信側に問い合わせることなくリモートメモリ書き込みにより送信を行う方式を提案し，高並列計算機EM-Xに実装しその評価を行った．その結果，通信遅延15.3 μsec，スループット31.4 MB/sを達成し，他MPPに実装されているMPIに比べ優位な性能を示した．本手法は，他システムにおいても適応可能であり，ハードウェアスペックどおりの低遅延，高スループットを得るためには重要な方式と考えられる．

Efficient Implementation of MPI Using Remote Memory Write

Osamu Tatebe,[†] Yuetsu Kodama,[†] Satoshi Sekiguchi[†]
and Yoshinori Yamaguchi[†]

MPI point-to-point communication is a basic operation, however it requires runtime-matching of send and receive that causes to reduce performance. This paper proposes a new approach to send messages by remote memory write without inquiring of the receiver under a communication pattern such that the corresponding nonblocking receive is issued in advance. Basically, this approach makes it possible to gain low latency and high bandwidth as the hardware specification. MPI-EMX, our implementation of the MPI on the EM-X multiprocessor, achieves a zero-byte latency of 15.3 μsec and a maximum bandwidth of 31.4 MB/s, which can compete with commercial MPPs. This approach to reduce communication latency is widely applicable to other systems and is quite a promising technique for achieving low latency and high bandwidth.

1. はじめに

MPIはメッセージパッシングインタフェースの標準として'94にMPI-1[16]，'97にMPI-2[11]の仕様が制定された．ポータビリティを保ちつつ実行性能を極力落とさないために，実装方式は規定されず，また余分な処理を行わないようにデザインされている．また通信と計算のオーバラップを行うことにより，通信遅延を隠蔽することができるようにデザインされている．

Point-to-point通信はメッセージパッシングライブラリにとって基本的な操作というだけではなく，送信，受信によりとられる局所的な同期は並列プログラミングにおいて有用である．しかしながら，現在のMPIのpoint-to-point通信の実装では，送信元プロセッサにワイルドカードを用いる受信が可能であるため，対応する送受信のマッチング処理は一般に受信側で動的に行われる．この場合，送信時に一度受信側に問い合わせる必要があり，ハードウェアの性能に比べ通信遅延，通信オーバヘッドが大きくなってしまう原因となる．

本研究ではその処理を送信側，受信側の双方で行うことにより，ノンブロッキング受信が先行発行される通信パターンにおいて，送信時に受信側に問い合わせることなく，ユーザ空間からユーザ空間へのリモートメモリ書き込みにより送信を行う方式について提案と検討を行う．そして，高並列計算機EM-X上にリモートメモリ書き込み，リモートスレッド起動，I-structures[1]等の通信機構を用いMPIの実装を行い，MPIライブラリが効率的に処理をする余地のあるMPIプログラムについての解説を行うとともに，我々が実装したMPI-EMXの評価を行う．

2. MPIのpoint-to-point通信

MPIにおいて計算と通信のオーバラップを行い，通信遅延を隠蔽するためには，ノンブロッキング通信あるいはマルチスレッド処理を用いる必要があるが，マ

† 電子技術総合研究所
Electrotechnical Laboratory, AIST, MITI

ルチスレッド処理に関してはMPIでは規定されないため，ポータブルに実装するためにはノンブロッキング通信を用いることになる．

ノンブロッキング送受信は，意味的には実際の通信処理とは無関係にローカルに（自プロセスの処理だけで）終了し，それぞれの通信リクエストを返す．通信の完了確認，通信リクエストの解放のためには，`MPI_Wait()` 等を用いる．`MPI_Wait()` は送信または受信バッファが利用可能☆となるまでブロックし，受信情報を引数として渡されるステイタスに入れる．ブロッキング通信はこの通信完了までブロックされる．

ノンブロッキング通信を用い，通信と計算のオーバラップを行うためには，以下のように記述する．

```
/*  通信リクエスト発行  */
MPI_Irecv(recv_buf, n, datatype, source, tag,
          comm, &recv_req);
計算1          //  片側通信のために必要
MPI_Isend(send_buf, n, datatype, dest, tag,
          comm, &send_req);
計算2          //  ここでオーバラップ
// 送信リクエスト待ち
MPI_Wait(&send_req, &send_status);
計算3          //  ここでもオーバラップ
// 受信リクエスト待ち
MPI_Wait(&recv_req, &recv_status);
```

ノンブロッキング送受信が発行され `MPI_Wait()` 等により完了を待つ間が通信と計算のオーバラップが行われる余地のある部分となり，上のコードでは計算2と計算3がその部分にあたる．計算1はオーバラップのためには必要ないが，この論文における提案である片側送信を行うために必要な部分となる．

3. Point-to-point 通信のリモートメモリ操作による実装

3.1 リモートメモリ操作

リモートメモリ操作は他プロセッサのローカルメモリの操作を行うことであり，富士通 AP1000+ の PUT/GET，日立 SR2201 のリモート DMA 等のようにハードウェアによる実装と，Berkeley Active Messages を用いる等のソフトウェアによる実装がある．ハードウェアによる実装では，ローカルメモリを持つプロセッサの実行を妨げることなくリモートメモリ操作を行うことができるが，ソフトウェアによる実装ではローカルメモリを持つプロセッサがメッセージハンドラの起動等の処理を必要とするため，実際の通信遅延以外のオーバヘッドがある．

☆ 送信の場合は，送信バッファのデータはすでにコピーあるいは転送されて書き込み可，受信の場合は受信バッファにデータを受信したという意味である．

共有メモリ計算機，分散共有メモリ計算機におけるロード，ストア命令も，ローカルメモリ，リモートメモリという概念は薄れるが，リモートメモリ操作と考えることができる．

3.2 片側送信

対応する送信と受信のマッチングは送受信先，メッセージタグ，コミュニケータにより行われるが，受信側で送信元，メッセージタグにワイルドカードを指定している場合，またMPIのプロセスがマルチスレッド実行している場合，また複数受信完了命令を実行している場合等では，その対応関係に非決定性がある．その非決定性を任意の静的順序で解消する場合，デッドロック，スターベイションの可能性があるため，それらの静的解析が必要となる．MPI-2では動的プロセス生成，また別々に立ち上がったMPIプロセスどうしの通信をサポートしており，対応する送受信の静的解析はますます困難なものとなっている．すべてを静的に解決することができないため，動的，実行時の効率的な送受信のマッチング機構は重要である．

ワイルドカード指定の受信の場合，基本的には受信した順番に送受信のマッチングを行えばよいため，マッチングは通常受信側で行われる．しかしながら，この受信側でマッチングを行う方法では，送信時に受信側へ必ず問合せを行う必要があり，その問合せは通信コストがかかるだけではなく受信側のプロセスに影響を及ぼすため，通信遅延が増大する原因となる．

我々のアプローチでは，送受信のマッチングを送信側で行い，

- （`MPI_Irecv()` を先行発行し，）`MPI_Irecv()` で送信側の連想メモリに受信アドレス，受信リクエストアドレス，メッセージタグ，コミュニケータを書き込む（以後この処理を受信要求という）．
- `MPI_Isend()` あるいは `MPI_Send()` ではローカルにある連想メモリを調べ，対応する受信要求があればリモートメモリ書き込みを行う．

という実装方式（**図1**）をとる．この方式により，送信側は送信時に受信側に問い合わせることなく，リモートメモリ書き込みによる送信を行うことができる．以後，この送信を片側送信と呼ぶ．ここでの連想メモリのキーは送信先，メッセージタグ，コミュニケータである．また，同一キーを持つ受信要求が複数起こる可能性があり，MPIの仕様によりそれらの順序を守らなければならないため，連想メモリのそれぞれのエントリはキューになっている必要がある．

しかしながら，受信の送信元としてワイルドカードを指定した場合，送信側に受信要求を送ることができ

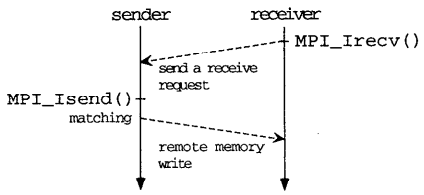


図 1 送信側でメッセージマッチングを行う片側送信
Fig. 1 One-sided send implementation by sender-side message-matching.

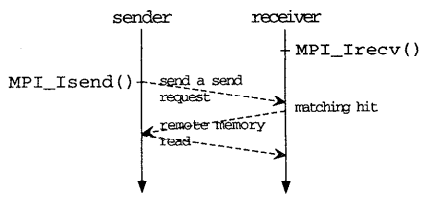


図 2 ノンブロッキング受信がすでに発行されている場合の同期プロトコルと get プロトコル
Fig. 2 Sync protocol and get protocol in the case that the corresponding nonblocking receive has been issued.

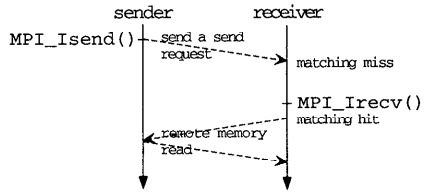


図 3 ノンブロッキング受信が発行されていない場合の同期プロトコル
Fig. 3 Sync protocol in the case that the corresponding nonblocking receive has not been issued.

ないため☆，この送信側のマッチングだけですべてのマッチングを行うことは効率が悪く，受信側におけるマッチングが必要となる．一方で，point-to-point 通信を片側送信で実装するためには送信側で対応する送受信のマッチングを行う必要があるため，送信側，受信側双方における送受信マッチングが必要になる．次章では矛盾のない送信側，受信側双方における送受信のマッチングのデザインについて述べる．

4. 送受信のマッチングの一貫性制御

送信側，受信側双方によるマッチングを行う場合，理想的には受信側がワイルドカード指定の受信であるかどうか，また送信に対しその対応する受信の発行が遅くないかどうかで，送信側でマッチングを行うか，受信側で行うかの判断を行えばよいが，それらはマッチングを行ってはじめて判断できるものであり，またいずれにしても送信側だけでは送信側，受信側のどちらでマッチングを行うのか判断することができない．そのため，一貫性を保つための制御が必要となる．

次節では，まず一貫性制御の前に，リモートメモリ操作を用いた受信側によるメッセージマッチングについてまとめる．

4.1 受信側によるメッセージマッチング

受信側で対応する送受信のマッチングを行う場合，いつマッチングを行うか，またメッセージのコピーを行うかによりいくつかプロトコルがある．メッセージのコピーを行わない零コピー通信の場合，送信側は送信バッファ，リクエストオブジェクトのアドレス，メッセージタグ，コミュニケータ（以後送信要求）を受信側に送り，対応する受信がすでに発行されているか調べるためにマッチングを行う．このマッチングが成功した場合，受信側はリモートメモリ参照によりメッセージの受信を行う（図 2）．失敗した場合，受信側は送信要求を連想メモリに格納し，対応する受信が発行されたときにリモートメモリ参照によりメッセージの受信を行う（図 3）．以後この方法を同期プロトコルと呼ぶ．同期プロトコルは MPIAP[14),15)] の protocol method とほぼ同じ方式で，一般にリモートメモリ参照を用い零コピー通信を実装するためにはこのプロトコルとなる．同期プロトコルはメッセージの余計なコピーを必要としないが，必ず MPI の同期モードの送信となってしまうため，デッドロックの可能性がある．

同期プロトコルでは，対応する受信が発行され，リモートメモリ参照が終了するまでブロッキング送信 `MPI_Send()`（あるいはノンブロッキング送信の送信完了命令）はブロックしてしまう．MPI の標準モードの送信をこの同期プロトコルで実装する場合，このブロックは性能低下の原因となってしまう．必要以上のブロックを防ぐためには，（システム，ライブラリ領域の）一次バッファを用い，一度送信バッファから別のバッファへコピーを行えばよい．この一次バッファはリモートメモリ操作を用いる場合，送信側，受信側どちらに持ってもかまわない．このとき，このコピーの前に，送信要求を送り送受信のマッチングを行うかどうかで 2 通りの方法がある．コピーの前に送信要求を送らない場合は，送信側あるいは受信側に一次バッファを確保し，コピーを行い，送信要求を送る．図 4 に，一次バッファを受信側に確保し，コピーにリモー

☆ 受信要求をブロードキャストしたとしても，対応する送信を 1 つだけ決定するための処理が必要となり，送信側だけでリモート書き込みによる送信を行うことはできない．が，ワイルドカード指定が送信元ではなくメッセージタグの場合は問題なく受信要求を送ることができる．

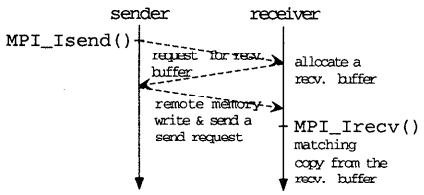


図 4 バッファプロトコル
Fig. 4 Buffered protocol.

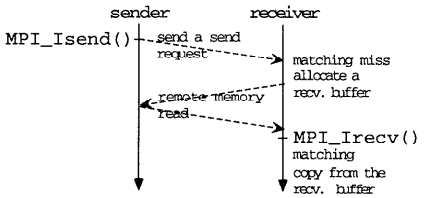


図 5 ノンブロッキング受信が発行されていない場合の get プロトコル
Fig. 5 Get protocol in the case that the corresponding nonblocking receive has not been issued.

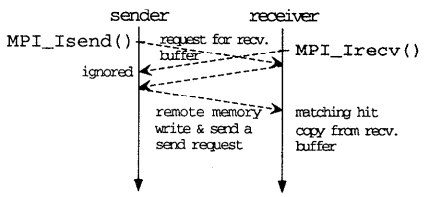


図 6 バッファプロトコルの場合
Fig. 6 In the case of buffered protocol.

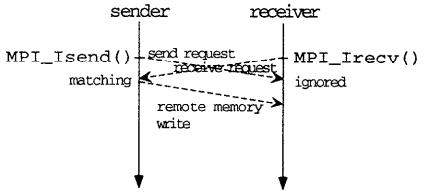


図 7 同期プロトコルまたは get プロトコルの場合
Fig. 7 In the case of sync and get protocols.

トメモリ書き込みを用いる場合を示す．同様の操作は受信側がリモートメモリ参照を用いても行うことができる．送信側に一次バッファを持つ場合は，コピーの後，同期プロトコルと同様となるが，送信完了はコピー終了後となる．以後，この方式をバッファプロトコルと呼ぶ．コピーの前に送信要求を送る場合は，すでに対応する受信が発行されていたら，同期プロトコル同様リモートメモリ参照により零コピーで通信が行われる（図 2）．対応する受信が発行されていなければ，受信側に一次バッファを持つバッファプロトコルと同様に受信側が一次バッファを確保し，送信側がリモートメモリ書き込みを行うか，受信側がリモートメモリ参照を行う（図 5）．以後，この方法を get プロトコルと呼ぶ．このプロトコルは MPICH[3] の get プロトコルとほぼ同じである．

4.2 双方によるメッセージマッチングの一貫性制御

送信側でメッセージマッチングを行い，リモートメモリ書き込みにより送信を行う片側送信は，`MPI_Irecv()` が先行発行されているコードを効率的に実行するためのデザインであるため，基本的なポリシとして送信発行時に対応する受信要求がない場合は，片側送信をあきらめる．このとき，

- 送信側は対応する受信要求があればリモートメモリ書き込みでメッセージを送信し，なければ送信要求を送る．
- 受信側は対応する送信要求があればリモートメモリ参照，あるいはローカルメモリコピーで受信する．送信要求がない場合，送信元がワイルドカード指定でなければ受信要求を送る．

しかしながら，このデザインでは送信と受信がほぼ同じタイミングで発行され，送信側は対応する受信要求がないので送信要求を送り，受信側も同様に対応する送信要求がないので受信要求を送った場合，お互いがお互いにマッチングを依頼することになり，正しくマッチングをとることができない．

基本的な解決策はどちらかの要求を無視することである．バッファプロトコルの場合は，送信要求により受信側へメッセージをコピーしてしまうため受信要求を無視するのが適当であり（図 6），同期プロトコル，get プロトコルの場合は，リモートメモリ書き込みの方が速いため，送信要求を無視するのが妥当である（図 7）．この解決策を実装するために，送信側，受信側ともに送信要求，受信要求を出したときに，要求を出した（相手にマッチングを依頼した）ことを示すキュー☆（送信，受信要求発行キュー）にそれらの要求を入れる．無視すべき要求が来た場合，その要求を無視し要求発行キューから取り除く．ただしこのとき，一貫性に問題がなく無視すべき要求が来ない場合にも，次の無視すべきでない要求が無視されないように要求発行キューから取り除く処理が必要となる．

また，連想メモリによるメッセージマッチングが成功し，エントリを消去，あるいは失敗し要求発行キューにエントリ入れるという操作はクリティカルセクションとなるため，マルチスレッドで実行している環境では排他ロックが必要となる．

☆ キューと書いているが，送信要求，受信要求と同様に連想メモリ，ハッシュ表を用いる．

4.3 通信の順序

MPIでは，通信の順序関係について2プロセス間ではプログラムの順序と一致することを保証している．我々の提案している方法は，送信元にワイルドカード指定を行わない限り，連想メモリがFIFOであり，ネットワークがFIFOを保証していれば通信の順序関係は保証される．連想メモリ，ネットワークがFIFOであれば送信，受信要求を出した順番を2プロセス間で保証することができるため，発行した順にマッチングを行うことができるからである．

問題になるのは，送信元にワイルドカードの指定をする受信を含む場合である．送信元にワイルドカードを指定した場合，受信要求を出さないため，後続の受信により受信要求を出してしまうと，送信側でその要求とマッチングを行ってしまう．したがって，MPIの要求するFIFO性を守るためには，送信元がワイルドカード指定の受信を発行した後は，その受信に対応する送信が発行されるまで受信要求を発行することができない．このとき，対応する送信とのマッチングが完了し，溜っていた受信要求をすべて発行した後，再び受信要求を発行することができる．

5. MPI-EMX

送信側，受信側双方による送受信のマッチングを行うMPIの実装を高並列計算機EM-X[5]上に行った．

5.1 EM-X

EM-Xは電総研で開発された分散メモリ型並列計算機であり，要素プロセッサEMC-Y[4]はデータ駆動機構に基づくパケット送出機構，パケットマッチング機構を備え，また基本的な実行単位となる強連結ブロック内ではノイマン型計算機に基づく逐次実行機構を備えている．EM-Xは現在80 PEのシステムが16 MHzで動作しており，ネットワークはサーキュラオメガ網である．

強連結ブロックはアトミックに実行され，割込み等により途中で中断することのない命令ブロックであり，1入力パケットあるいは2入力パケットで起動される．パケットはアドレス部，アドレスタグ部，データ部，データタグ部の2ワード☆で構成されている．パケットが到着すると，入力バッファに格納される．2入力待合せパケットの場合はここでマッチングがとられ，もう一方がまだ到着していなければメモリに待避させられる．入力バッファ中のパケットは強連結ブロックを実行可能なパケットであり，それらはFIFOで実行される．また，強連結ブロックはオペランドセグメントと呼ばれる関数フレームを用いることによりコンテキストを保存することができる．以後，強連結ブロックの起動のことをスレッド起動，他プロセッサのスレッド起動を特にリモートスレッド起動と呼ぶ．

I-structures[1]は関数型言語に導入された1-write, n-readのデータ構造であり，EM-Xでは2入力待合せパケットを用い1-write, 1-readのデータ構造として実装されている．またI-structuresをFIFOキューを用い複数read，複数writeに拡張したQ-structures[2]も同様に実装され，キューに関してはソフトウェアで実装されている．

リモートメモリ操作はリモートスレッド起動で実装されているが，SYSWR, SYSRDパケットと呼ばれる特殊パケットに関してはハードウェアで実装され，入力バッファにそれらのパケットが入ると，実行ユニットを通らないでそれぞれリモートメモリ書き込み，リモートメモリ参照を行う．

プログラミング言語としてはC言語にマルチスレッド処理，大域ポインタ，I-structures, Q-structures等の拡張を行ったEM-C[13]が開発されている．

5.2 MPI-EMXの実装

我々の提案している片側送信の実装のためには，リモートメモリ操作のほかに送信要求，受信要求の発行が必要である．これらの操作は他プロセッサの連想メモリへの書き込みであるが，連想メモリの各要素はキューであり，また書き込む前に対応する要求がすでに送られたかどうか調べるために他プロセッサの要求発行キューを調べる必要がある．MPI-EMXではこれらの処理をリモートスレッド起動により実装した．このとき，クリティカルセクションでは排他ロックを掛ける必要がある．

片側送信では連想メモリのキューの操作が主な通信遅延となるため，アセンブリ言語を用い高速化を図った．アセンブリにより強連結ブロックとして記述したことにより，排他ロックフリーな実装となっている．

メッセージの通信にはリモートメモリ操作を用いているが，メッセージの通信だけではなくMPIの受信リクエストオブジェクトへの送信元，メッセージタグ，サイズの書き込みもリモートメモリ操作を用いている．

また受信完了を調べるためにはポーリング，割込みという方法がよくとられるが，EM-Xは2入力パケットのマッチング機構を用いI-structuresをサポートしており，受信待ちにはI-structuresの局所同期機構を用いた．この機構によりプロセッサに負荷を与えるこ

☆ 1ワードはデータ32ビット，タグ6ビットの38ビットである．

となく，理想的な受信確認を行うことができる．ここで問題になるのは，MPI_Test() では受信完了かどうかチェックする必要があるが，I-structures を用いる場合 read がブロックしてしまいそのチェックを行うことができないことである．この問題に対して我々は I-structures にブロックしない非同期 read 操作を加え，EM-X に実装を行った．この非同期 read 操作は，write が発行されていれば通常の read と同じであり，発行されていない場合はブロックしないですぐに戻るというもので，EM-X では I-structures のタグの部分を調べることで実装を行っている．

対応する送受信のマッチングに用いる連想メモリとして Q-structures の機構の使用も考えられるが，いくつかの問題点がある．1 つの問題は Q-structures のアドレス空間で，MPI では送受信プロセス，メッセージタグ，コミュニケータの 3 ワード（12 バイト，あるいは 96 ビット）が必要となることである．もう 1 つの問題は受信でワイルドカード指定をした場合，Q-structures のアドレスを定めることができないということである．

EM-X の通常のリモートスレッド生成は，オペランドセグメントの要求，引数のリモートメモリ書き込み，リモートスレッド起動という処理となるが，引数が 2 ワード以下でかつオペランドセグメントが必要ない場合は，特殊パケットハンドラを用いることにより特殊パケットだけでスレッド起動を行うことができる．我々の実装では送信要求，受信要求，要求発行キューからの取り除きをリモートスレッド起動で実装しているが，要求発行キューからの取り除きに関しては引数が少ないため，特殊パケットハンドラにより実装を行った．

その他，送信/受信リクエスト領域は固定長領域となるため，その領域の確保，解放を，フリーリストを用いたアセンブリ記述を行い最適化を行った．MPI_Wait() もアセンブリ記述を行い高速化を図っている．

5.3 他システムにおける実装について

片側送信は EM-X に限らず，リモートメモリ操作が実装されている他システムに対しても実装を行うことができる．ただし，送信要求，受信要求の実装に関してはリモートスレッド起動あるいは何らかの FIFO 性が保証されるメッセージ送信機構が必要となる．EM-X ではこの処理にリモートスレッド起動 (アクティブメッセージ) + (プロセス単位の) 共有連想メモリで実装し，要求発行キューの検索と連想メモリへの書き込みを行っているが，FIFO の保証される受信バッファを用いる場合は，割込み，あるいはポーリングを用い，同等の処理を行うこととなる．

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Barrier(MPI_COMM_WORLD);
if (rank == 1) {
    time = MPI_Wtime();
    for (i = 0; i < NUM_ITER; ++i) {
        MPI_Irecv(rbuf, 1, MPI_INT, 2, 100,
                  MPI_COMM_WORLD, &req);
        MPI_Wait(&req, &st);
        MPI_Isend(sbuf, 1, MPI_INT, 2, 100,
                  MPI_COMM_WORLD, &req);
        MPI_Wait(&req, &st);
       /*  or MPI_Request_free(&req); */
    }
    time = MPI_Wtime() - time;
}
else if (rank == 2) {
    time = MPI_Wtime();
    for (i = 0; i < NUM_ITER; ++i) {
        MPI_Isend(sbuf, 1, MPI_INT, 1, 100,
                  MPI_COMM_WORLD, &req);
        MPI_Wait(&req, &st);
        /*  or MPI_Request_free(&req); */
        MPI_Irecv(rbuf, 1, MPI_INT, 1, 100,
                  MPI_COMM_WORLD, &req);
        MPI_Wait(&req, &st);
    }
    time = MPI_Wtime() - time;
}
```

図 8 通常の pingpong のコード
Fig. 8 Naive code for pingpong.

6. 性能評価とプログラミング例

この章では MPI-EMX の性能評価と，MPI ライブラリが効率的に実行できる余地のあるプログラミングについての解説を行う．

6.1 Pingpong

MPI を用いた pingpong のプログラムは，通常図 8 のように一方が送信し，受信を行い，もう一方が受信し，送信を行う．図 8 ではそれらを繰り返して 4 バイトの通信遅延時間を計測している．

しかしながら，このプログラムでは，MPI_Irecv() を先行発行しているわけではないため，片側送信が行われるとは限らない．そこで，このプログラムを図 9 のようにあらかじめすべての MPI_Irecv() を先行発行してから pingpong を始めるように書き換える☆．このプログラムでは図 8 と異なり，先行発行した分の受信リクエストが必要になる．この場合，送信を行うときにはすでに受信要求が出ているため，つねに受信側に問い合わせることなくリモートメモリ書き込みによる片側送信となる．また，このプログラムではノンブロッキング受信を先行発行した後に MPI_Barrier()

☆ ベンチマークとしての意味はこのプログラムで正しいが，一般的には等しい受信アドレスのノンブロッキング受信を先行発行すると競合状態となるため，異なる受信アドレスを割り当てる必要がある．

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Barrier(MPI_COMM_WORLD);
if (rank == 1) {
    time1 = MPI_Wtime();
    for (i = 0; i < NUM_ITER; ++i) {
        MPI_Irecv(rbuf, 1, MPI_INT, 2, 100,
                  MPI_COMM_WORLD, &req[i]);
    }
    time1 = MPI_Wtime() - time1;
    MPI_Barrier(MPI_COMM_WORLD);
    time2 = MPI_Wtime();
    for (i = 0; i < NUM_ITER; ++i) {
        MPI_Isend(sbuf, 1, MPI_INT, 2, 100,
                  MPI_COMM_WORLD, &r);
        MPI_Wait(&r, &s);
        MPI_Wait(&req[i], &st[i]);
    }
    time2 = MPI_Wtime() - time2;
}
else if (rank == 2) {
    time1 = MPI_Wtime();
    for (i = 0; i < NUM_ITER; ++i) {
        MPI_Irecv(rbuf, 1, MPI_INT, 1, 100,
                  MPI_COMM_WORLD, &req[i]);
    }
    time1 = MPI_Wtime() - time1;
    MPI_Barrier(MPI_COMM_WORLD);
    time2 = MPI_Wtime();
    for (i = 0; i < NUM_ITER; ++i) {
        MPI_Wait(&req[i], &st[i]);
        MPI_Isend(sbuf, 1, MPI_INT, 1, 100,
                  MPI_COMM_WORLD, &r);
        MPI_Wait(&r, &s);
    }
    time2 = MPI_Wtime() - time2;
}
else {
    MPI_Barrier(MPI_COMM_WORLD);
}
```

図 9 ノンブロッキング受信を先行発行する pingpong のコード
Fig. 9 Pingpong code issuing nonblocking receives in advance.

でバリアをとっているが，このバリアは測定のために入れたものであり，通常は必要ない．

図 8 と図 9 のプログラムを用い，双方によるマッチング，あるいは受信側によるマッチングを行った，それぞれのプロトコルにおける MPI-EMX の pingpong 転送の片道の時間を**表 1** に示す．片道の転送時間は pingpong 転送の往復の時間の半分としている．表中，先行発行の欄の括弧付けの数値はノンブロッキング受信を先行発行するオーバヘッドである．受信を先行発行するプログラムでは，双方でメッセージマッチングを行う場合，つねに送信側でマッチングが成功するため，プロトコル間の差はなく，この時通信遅延は 15.4 μsec であり，また零バイトの通信遅延は 15.3 μsec であった．双方によるマッチングの場合の先行発行のオーバヘッドは大きくなっているが，片側通信には低遅延と相手プロセスに影響を与えず，通信と計算のオーバラップ実行を行うことができるため，実アプリケーションではこの効果が大きいと考えられる．6.3 節で後退代入のプログラムを用いこの評価を行う．

表 1 MPI-EMX の 4 バイトの通信遅延の片道時間
Table 1 One-way 4-byte latency of the MPI-EMX.

	both sides			only receiver side		
	buf.	get	sync	buf.	get	sync
通常	36.1	25.4	27.9	42.7	21.3	22.1
先行発行	15.4 (21.6)			42.9 (12.1)	20.5	21.2

(μsec)

表 2 4 バイトの通信遅延の片道時間
Table 2 One-way 4-byte latency.

	AP	AP+	Cenju-3	SR2201	Wiz
通常	72.8	26.1	48.9	32.6	199
先行発行	80.3	27.0	50.2	79.9	197

(μsec)

図 8 の通常の pingpong のプログラムでは，同期プロトコル，get プロトコルの場合，双方でマッチングを行うと，送信側のマッチングがすべて失敗してしまい，その分がオーバヘッドとなり受信側だけのマッチングの方が速くなっている．が，バッファプロトコルは初めにメッセージを受信するプロセスの送信側のマッチングがすべて成功し，双方でマッチングをする方が速くなっている．

また図 8 の pingpong のプログラムでは図 9 のように先行発行を行わなくても get プロトコルはすべて零コピー通信となってしまい，get プロトコル，同期プロトコルともに低遅延となっている．このとき，先行発行は逆にオーバヘッドとなっている．

同期プロトコルと get プロトコルについては，get プロトコルの方が少し速くなっているが，pingpong のプログラムの場合，両プロトコルでマッチングの回数，マッチングが成功する場所は等しく，その差はプロトコルの差ではなく，単に実装上の問題である．

次に同プログラムを AP1000，AP1000+，Cenju-3，SR2201，Wiz で実行させた場合の pingpong の片道の時間を**表 2** に示す．Wiz[7] は 333 MHz の DEC AlphaStation を CISCO Catalyst 5000 スイッチを用い Fast Ethernet 100Base TX でつないだクラスタである．AP1000，AP1000+ では MPIAP[14),15)] を用い，Cenju-3 では mini-MPI を用いた．SR2201 は HI-UX/MPP，Wiz は MPICH 1.1.0 を用いている．

その他のシステムでは送信側で送受信のマッチングを行っていないあるいは零コピー通信をサポートしていないため図 8 および図 9 のプログラムによる差があまりない．それどころか，SR2201 のように倍以上遅くなってしまったり，Cenju-3 の mini-MPI のように 1000 程度の `MPI_Irecv()` の先行発行でも実行で

表 3 MPI-EMX の後退代入の性能（MFlops/#PE）
Table 3 Performance of back substitution using the MPI-EMX (MFlops/#PE).

		both sides			only receiver side		
		buf.	get	sync	buf.	get	sync
w/o yield()	通常	2.55	2.33	2.17	2.55	0.71	0.71
	先行発行	3.10			2.55	0.71	0.71
w. yield()	通常	2.73	2.78	2.84	2.73	2.78	2.84
	先行発行	2.84			2.74	2.85	2.85

(#PE = 5, N = 2000, n of block = 25)

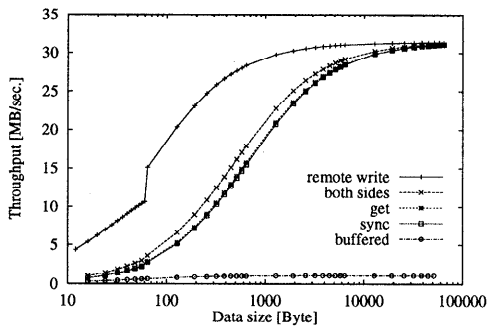


図 10 EM-X の通信スループット
Fig. 10 Throughput on the EM-X.

きないシステムもあり☆，現在使用可能な MPI では受信を先行発行しても功を奏さないシステムが多いのも現状である．

しかしながら重要なのは図 9 のように `MPI_Irecv()` を先行発行するプログラムは，片側送信，零コピー通信等 MPI ライブラリが効率的に実行を行う余地があるということである．

6.2 スループット

`MPI_Irecv()` を先行発行する pingpong プログラム（図 9）を用い，通信スループットの測定を行った．基本的にこの通信パターンでは送信側は受信要求の確認，リモートメモリ転送を行うだけであり，スループットに関してはほぼリモートメモリ転送の性能が出ることが期待される．

図 10 に MPI-EMX のそれぞれのプロトコルのスループットとリモートメモリ書き込みのスループットを示す．一番上の remote write は用いているリモートメモリ書き込みの性能で，この関数は 64 バイト単位でループアンローリングを行っているため，64 バイトあたりで段差ができている．

双方でマッチングを行う場合がスループットの立ち上がりが一番良く，512 バイトでピークの半分の性能を超え，1.6 MB のデータでは 31.4 MB/sec の性能であった．このスループットはリモートメモリ書き込みのものとほぼ等しい性能である．が，pingpong の節でみたように零バイトの通信でも遅延が 15.3 μsec もあるため，立ち上がりが少し鈍くなっている．

EM-X は通信遅延がきわめて小さいことと，このプログラムでは受信側はほかに計算をすることなく受信のためにただ待っているだけであるため，受信側の処理が速く，同期プロトコル，get プロトコルを用い受信側だけのマッチングによるスループットとの差はわずかなものになっている．バッファプロトコルは必ず一次バッファを割り当て，コピーを行うためスループットはデータサイズを増やしても大きくならない．

6.3 後退代入

後退代入はガウスの消去法の求解部に現れ，上三角行列の解法である．上三角行列を行ブロックサイクリックに分割し，パイプライン処理により並列実装を行っている．ノンブロッキング受信を先行発行するプログラムは，プログラム中のすべての受信をノンブロッキング受信として先行発行している．またこの先行発行の時間も計時に含まれている．

表 3 に 5 台のプロセッサを用い，行列の大きさ 2000，ブロックサイズ 25 の場合の結果を示す．単体プロセッサにおける性能は 3.66 MFlops であり，先行発行の場合，双方でマッチングを行うときは 85%の並列効率が出ている．

後退代入のプログラムの場合，EM-C の生成するコードではブロック中のデータの計算の二重ループが強連結ブロックとなり，実行中は他の強連結ブロック（スレッド）は起動できない．表中 w. `yield()` は `yield()` をループ中で明示的に呼び，強連結ブロックを中断させているプログラムである．

受信側だけでメッセージマッチングを行う場合，同期プロトコル，get プロトコルはループ中で `yield()` を呼ばないと，受信側で送信側が生成したスレッドの起動が遅れ，きわめて性能が低下してしまっている．

しかしながら，`yield()` は計算にとっては純粋なオーバヘッドである．受信の先行発行と双方によるメッセージマッチングを行う場合，片側送信となり受信側のプロセスに影響を及ぼさないため，`yield()` を

☆ 表 2 の Cenju-3 のデータは 100 回先行発行したものである．

呼ぶ必要がない．受信の先行発行を行うと，その先行発行のためのオーバヘッドはあるが，片側送信と組み合わせることにより，全体としての性能を上げることができ，片側送信は重要な実装方式といえる．

7. 関連研究

受信側が送信側にコントロールメッセージを送り，送信時に受信側に問合せなしにリモートメモリ書き込みにより送信を行うという片側送信に関しては同時期に独立に MPI/MBCF[9),10)] でも研究されている．MPI/MBCF では，メモリベース通信機構 MBCF[8)] を用い，Q-structures に相当するメモリベース FIFO をプロセス数だけ用い，送受信のマッチングのための連想メモリとしている．ここで，FIFO の先頭の要求が対応しない場合は，その要求を別バッファに退避させている．また，ワイルドカード指定の受信に関しては，プロセス数分の FIFO を順番に検索することで実装している．

MPICH[3)] は ADI と呼ばれる仮想デバイスインタフェースを用いたポータブルな MPI の実装であり，そのインタフェースに合わせることにより MPI-FM[6)], MPI-AM[17)], MPI-PM[12)] 等，新たな通信メカニズムに対するポーティングが容易となっている．また，MPICH は MPI のモデル実装として HI-UX/MPP 等多くのシステムで使われている．MPICH を用いた多くのシステムでは，ADI のさらに下のチャネルインタフェースを定義することでポーティングを行っているが，チャネルインタフェースで提供している eager, rendezvous, get の 3 つのメッセージ通信プロトコルはいずれも送信側がコントロールメッセージを受信側に送り，送受信のマッチングは受信側で行われる．

MPIAP[14),15)] は富士通 AP1000, AP1000+, AP 3000 における MPI の native な実装であり，送信方式として in-place method, protocol method という 2 つの方式を実装している．さらに受信方法もポーリング，シグナル（割込み），それらを合わせた方法を実装し，比較検討を行っている．が，やはりそれらはすべて受信側で送受信のマッチングを行っている．

8. おわりに

MPI の point-to-point 通信において，`MPI_Irecv()` が先行発行されるパターンで，対応する送信が受信側に問い合わせることなくリモートメモリ書き込みにより送信を行う片側送信の実装についての提案と検討を行った．片側送信は，リモートメモリ操作をサポートするマシンで効果的に実装することができ，EM-X における実装では，通信遅延が 15.3 μsec であった．この遅延は他 MPP システムに比べても少ないが，それよりもむしろこの遅延の大部分はプロセッサローカルの処理であり，他プロセスの影響を受けないということが重要である．スループットに関してはほぼピーク性能であった．片側送信は，単に遅延を減少させ，スループットを向上させるだけではなく，受信プロセスの実行に影響を及ぼさないためプログラム全体の性能を上げることができ，ハードウェアスペックどおりの性能をメッセージパッシングで出すためには有効な実装方式といえる．

ここで `MPI_Irecv()` が先行発行されるパターンは，現在の処理系では必ずしも功を奏するとは限らないが，重要なのは MPI の実行時の処理として，ハードウェアの機能を用い効率的に実行する余地を与えるプログラミングを行うということである．

現在，point-to-point 通信，バリア同期，一部のリダクション演算，Cartesian トポロジ等が実装されており，今後フルスペックの仕様をサポートしていく予定である．

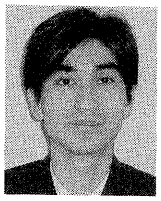
謝辞 本研究を遂行するにあたり貴重なご助言，ご討論いただいた大蒔和仁部長，佐藤三久氏，坂根広史氏，山名早人氏，小池汎平氏ほか，ご協力いただいた諸氏に感謝いたします．

参考文献

1) Arvind, R., Nikhil, S. and Pingali, K.K.: I-structures: Data structures for Parallel Computing, *ACM Trans. Programming Languages and Systems*, Vol.11, No.4, pp.598–632 (1989).
2) Bic, L.F., Nicolau, A. and Sato, M.: *Parallel Language and Compiler Research in Japan*, Kluwer Academic Publishers (1995).
3) 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, pp.789–828 (1996).
4) Kodama, Y., Koumura, Y., Sato, M., Sakane, H., Sakai, S. and Yamaguchi, Y.: EMC-Y: Parallel Processing Element Optimizing Communication and Computation, *Proc. 1993 International Conference on Supercomputing*, pp.167–174 (1993).
5) Kodama, Y., Sakane, H., Sato, M., Yamana, H., Sakai, S. and Yamaguchi, Y.: The EM-X Parallel Computer: Architecture and Basic Performance, *Proc. 22nd Annual International Symposium on Computer Architecture*, pp.14–

23 (1995).
6) Lauria, M. and Chien, A.: MPI-FM: High Performance MPI on Workstation Clusters, *Journal of Parallel and Distributed Computing*, Vol.40, No.1, pp.4–18 (1997).
7) 益口摩紀, 建部修見, 関口智嗣, 長島雲兵, 佐藤三久:アルファワークステーションのクラスタetlwizの性能評価, 情報処理学会研究報告, HOKKE'98, 98-HPC-70, pp.61–66 (1998).
8) Matsumoto, T. and Hiraki, K.: MBCF: A protected and virtualized high-speed user-level memory-based communication facility, *Proc. 1998 International Conference on Supercomputing*, pp.259–266, ACM (1998).
9) 森本健司, 松本 尚, 平木 敬:メモリベース通信を用いた高速MPIの実装方式, 並列処理シンポジウムJSPP'98論文集, pp.191–198 (1998).
10) 森本健司, 松本 尚, 平木 敬:並列アプリケーションによるMPI/MBCFの評価, 情報処理学会研究報告, 98-HPC-72, pp.103–108 (1998).
11) Message Passing Interface Forum: *MPI-2: Extensions to the Message-Passing Interface* (1997). http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html.
12) O'Carroll, F., Tezuka, H., Hori, A. and Ishikawa, Y.: The design and implementation of zero copy MPI using commodity hardware with a high performance network, *Proc. 1998 International Conference on Supercomputing*, pp.243–250, ACM (1998).
13) Sato, M., Kodama, Y., Sakai, S., Yamaguchi, Y. and Koumura, Y.: Thread-based Programming for the EM-4 Hybrid Dataflow Machine, *Proc. 19th Annual International Symposium on Computer Architecture*, pp.146–155 (1992).
14) Sitsky, D. and Hayashi, K.: Implementing MPI for the Fujitsu AP1000/AP1000+ using Polling, Interrupts and Remote Copying, *Proc. Joint Symposium on Parallel Processing 1996*, pp.177–184 (1996).
15) Sitsky, D. and Hayashi, K.: An MPI library which uses polling, interrupts and remote copying for the Fujitsu AP1000+, *Proc. International Symposium on Parallel Architectures, Algorithms, and Networks*, IEEE (1996).
16) Snir, M., Otto, S., Huss-Lederman, S., Walker, D. and Dongarra, J.: *MPI: The Complete Reference*, The MIT Press (1996).
17) Wong, F.C. and Culler, D.E.: *Message Passing Interface Implementation on Active Messages*. http://now.CS.Berkeley.EDU/Fastcomm/MPI/.

(平成10年9月4日受付)
(平成11年3月5日採録)



建部 修見(正会員)

昭和44年生. 平成4年東京大学理学部情報科学科卒業. 平成9年同大学大学院理学系研究科情報科学専攻博士課程修了. 同年電子技術総合研究所入所. 並列数値アルゴリズム, 並列計算機システム, 広域分散計算の研究に従事. ハイパフォーマンスコンピューティングに興味を持つ. 理学博士. 日本応用数理学会, ACM各会員.

児玉 祐悦(正会員)

1962年生. 1986年東京大学工学部計数工学科卒業. 1988年同大学大学院情報工学専門課程修士課程修了. 同年通産省電子技術総合研究所入所. 現在, 同所情報アーキテクチャ部主任研究官. データ駆動型計算機等の並列計算機システムの研究に従事. 特にプロセッサアーキテクチャ, 並列性制御, 動的負荷分散, 並列探索問題等に興味あり. 情報処理学会奨励賞, 情報処理学会論文賞(1990年度), 市村学術賞(1995年)等受賞. 電子情報通信学会, IEEE各会員.

関口 智嗣(正会員)

昭和34年生. 昭和57年東京大学理学部情報科学科卒業. 昭和59年筑波大学大学院理工学研究科修了. 同年電子技術総合研究所入所. 情報アーキテクチャ部主任研究官. データ駆動型スーパコンピュータSIGMA-1の開発等の研究に従事. 並列数値アルゴリズム, 計算機性能評価技術, ネットワークコンピューティングに興味を持つ. 市村賞受賞. 日本応用数理学会, SIAM, IEEE各会員.

山口 喜教(正会員)

1972年東京大学工学部電子工学科卒業. 同年通商産業省工業技術院電子技術総合研究所入所, 計算機方式研究室長等を経て, 1999年筑波大学電子・情報工学系教授(電子技術総合研究所併任), 工学博士. 高級言語計算機, 並列計算機アーキテクチャ, 並列実時間システム等の研究に従事. 1991年情報処理学会論文賞, 1995年市村学術賞受賞. 著書「データ駆動型並列計算機」(共著, オーム社). IEEE-CS, ACM, 電子情報通信学会各会員.