遠隔メモリ操作に基づく高速メッセージパッシングライブラリFMPLの設計と開発

建部修見[†] 長嶋雲兵[†] 関口智嗣[†]
北林久義[††] 林田義幸[†††]

FMPLは一般的になってきた遠隔メモリ操作などの通信機構を前提として設計した汎用通信ライブラリである．低コストの一対一通信，集合通信を提供し，動的オーバヘッドの削減，通信と計算のオーバラップなどによる全体的な性能向上を目指している．日立SR8000において，FMPLは一対一通信の8バイト転送遅延で12.8 μsecを達成し，日立MPIの転送遅延を36%改善した．また16ノード8MBデータのブロードキャストではFMPLは877 MB/sを達成し，MPIのバンド幅を57%改善した．本ライブラリは通信ハードウェア性能を最大限に引き出す必要のあるアプリケーションだけではなくBLACSなどより上位の通信ライブラリの効率的な実装のためにも利用される．

Design and Implementation of FMPL: A Fast Message-passing Library Based on Remote Memory Operations

Osamu Tatebe,[†] Umpei Nagashima,[†] Satoshi Sekiguchi,[†]
Hisayoshi Kitabayashi[††] and Yoshiyuki Hayashida[†††]

A fast message-passing library FMPL has been designed and developed to maximize communication performance by utilizing general architectural communication support such as remote memory operations, as well as to maximize total performance by eliminating dynamic communication overhead and overlapping communication and computation. FMPL provides a low-cost general-purpose point-to-point communication and collective communication such as broadcast, barrier synchronization and reduction. On a Hitachi SR8000, FMPL achieves an 8-byte latency of 12.8 μsec., while MPI achieves 20 μsec. FMPL achieves bandwidth of 877 MB/s for 16-node broadcasting and improves 57% than MPI. FMPL is designed for building more highly functional message-passing libraries like BLACS as well as applications that need maximum performance.

1. はじめに

計算機クラスタを含む分散メモリ型の並列計算機では，通信性能を高めるため，リモートノードのメモリを直接更新，参照するハードウェア機構を備えているものが増えてきた．この遠隔メモリ操作を用いることにより，メッセージパッシングの一対一通信においてユーザ空間からユーザ空間へ零コピーでメッセージを転送することができ，メモリ間コピーのオーバヘッドを削減し，ハードウェアのピーク性能に迫る転送バンド幅を達成することができる[10),18),19)]．一対一通信を遠隔メモリ操作を用い零コピーで実装する多くの研究は，メッセージパッシングライブラリの標準であるMPI[12)]を実装している．しかしながら，MPIは遠隔メモリ操作の利用を前提としていないため，メッセージハンドリングオーバヘッドが大きくなってしまっている．そのため，ベンチマークプログラム，通信性能を要求されるアプリケーションなどには，遠隔メモリ操作などハードウェア通信サポートを前提とした低コストのメッセージパッシングライブラリが望ましい．

また，連立一次方程式解法，最小二乗問題，固有値問題などの行列演算パッケージのScaLAPACK[2)]では，通信層としてBLACS[3)]（Basic Linear Algebra Communication Subprograms）を用いている．BLACSは行列演算パッケージのための簡単で使いや

† 産業技術総合研究所グリッド研究センター
Grid Technology Research Center, National Institute of Advanced Industrial Science and Technology
†† 株式会社日立ビジネスソリューション基本ソフト事業部
Software Development Department, Hitachi Business Solution
††† 株式会社日立製作所ソフトウェア事業部
Software Division, Hitachi, Ltd.

すいインタフェースを目指して設計された通信インタフェースである．BLACS は，さらに汎用的で多くのプラットフォームで利用可能な MPI による実装が主に利用されるが，この場合，BLACS の配列のパック，アンパックなどのコピーオーバヘッドと，MPI のメッセージハンドリングオーバヘッドが生じてしまう．そのため，高性能を実現するためには MPI のような機能豊富な通信ライブラリではなく，低コストで汎用的な高速メッセージパッシングライブラリが望ましい．

本研究では，遠隔メモリ操作などのハードウェア通信サポートを前提として最適化した低オーバヘッドの高速メッセージライブラリ FMPL の設計，および日立 SR8000 上での実装，評価を行う．FMPL では，遠隔メモリ操作による軽い同期処理と転送処理を利用した低コストの一対一通信ライブラリと，その軽い一対一通信および通信ハードウェア機構を用いた集合通信を提供する．FMPL は通信性能が要求されるアプリケーションだけではなく，BLACS など，より上位のメッセージパッシングライブラリの効率的な実装のためにも利用される．

2. FMPL：高速メッセージパッシングライブラリ

FMPL は一般的になってきた遠隔メモリ操作などの通信機構を前提として設計した汎用通信ライブラリである．低コストの一対一通信，集合通信を提供し，動的オーバヘッドの削減，通信と計算のオーバラップなどによる全体的な性能向上を目指している．FMPL の基本的な設計方針は以下のとおりである．

- 遠隔メモリ操作，ハードウェアバリア機構，ハードウェアブロードキャスト機構など，一般的になってきたハードウェア通信機構の効率的な利用．
- 動的メモリ確保，キュー操作，未使用領域の検索，割込みなど動的オーバヘッドの削除．
- BLACS や MPI など，より高機能を提供するメッセージパッシングライブラリを効率的に実装するための柔軟性と低オーバヘッド．

2.1 従来の一対一通信の実装

一対一通信は，通信処理と同期処理を合わせたものであり，送信側は送信命令により送信領域を指定し，受信側は受信命令により受信領域を指定する．送信命令と受信命令は宛先，メッセージタグなどでマッチングがとられ，メッセージの転送が行われる．このとき，送信側が送信を発行するまでは送信領域が転送されることはなく，送信が終了したら送信領域を安全に書き換えることができる．また，受信側が受信を発行するまでは，受信バッファが上書きされてしまうことはなく，受信が終了したら受信領域にはデータが揃っていることが保証される．

一対一通信における送信，受信のマッチングはそれぞれの 2 プロセス間で FIFO 順序を保証する必要があること，およびメッセージタグなどのマッチングに必要な値のとりうる範囲が広いことなどにより，ヘッダ情報は通常キューで管理される．このとき，エンキュー，デキューなどにともなうメモリ管理，キュー中のエントリ検索などの動的な処理がともない，メッセージマッチング処理における実行時オーバヘッドとなってしまう．

また，メッセージマッチング処理は，後述する送信プロセスランクを指定しない受信などのために通常受信側で行われる．この場合，メッセージ長が短い場合はヘッダ情報とメッセージ本体の両方を受信側に送り（eager プロトコル），長い場合はヘッダ情報だけを送る（rendezvous プロトコル）．eager プロトコルは，通信遅延は短くなるが，一コピー通信となって受信側でユーザ指定の受信バッファへコピーする必要があるため，通常通信スループットが下がってしまう．一方，rendezvous プロトコルは遅延は増えるが，受信側でのコピーを減らすことができ，通信スループットを上げることができる．

これらのプロトコルを遠隔メモリ参照などの通信機構を利用して実装する場合，受信側へヘッダ情報などを転送する際のキュー管理の実装として，(1) Active Messages[16] などを利用してエンキューする，(2) PM[15] などのメッセージパッシング通信を利用する，(3) 送信側で書き込み領域の管理を行い遠隔メモリ書き込みを利用する，などの方法がある．(3) の場合は，書き込み領域の再利用のために受信側から送信側へのコントロールメッセージが別途必要となる．

2.2 FMPL 零コピー一対一通信

FMPL の一対一通信は，低遅延，低オーバヘッド，高バンド幅を実現するために，遠隔メモリ操作を利用した送信側メッセージマッチング[18),19)] による零コピー通信が可能なように設計されている．零コピー通信とは，通信のための一時バッファを介することなく，直接ユーザ空間上のデータ領域にデータを送信することである．

FMPL では，メッセージマッチング処理を軽減するために，メッセージタグとして排他的なマッチング領域を指定する．マッチング領域は受信バッファの先頭アドレスを含むメッセージヘッダを書き込むために利用され，送信側はそれにより対応する受信が発行され

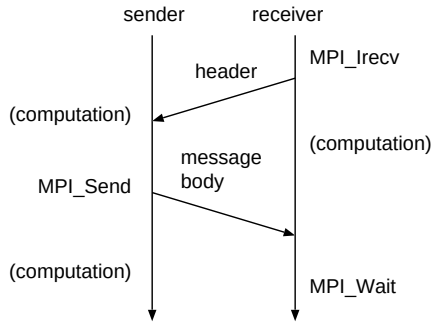

図 1 送信側メッセージマッチングにより，送信時に受信側に問合せをしないで遠隔メモリ書き込みを行う零コピー一対一通信の処理の流れ

Fig. 1 Zero-copy point-to-point communication implemented by immediate remote memory writes and sender-side matching mechanism.

たことと，そのメッセージヘッダの内容を知ることができる．通常のメッセージパッシングライブラリにおけるメッセージタグと異なり，指定されるマッチング領域は送信，受信のペアごとにそれぞれ排他的なことを前提としており，マッチング領域ではキュー管理を行わない．このことにより，動的メモリ確保，キュー操作，キュー検索などマッチングにともなう動的オーバヘッドをなくすことができる．この前提は，一対一通信が完了しない間はそれぞれの 2 プロセス間において別々のマッチング領域を指定するということであり，このことは最大利用可能なマッチング領域の数だけ同じ送受信ペアにおいてノンブロッキング送受信を発行できることを意味している．十分な数のマッチング領域が存在する場合は，毎回違う領域を指定するようにプログラムを変換することにより排他的な指定となるため，一般性が失われることはない．

送信側メッセージマッチングの処理の流れを**図 1** に示す．受信操作では，送信側のマッチング領域にメッセージヘッダを遠隔メモリ書き込みでただちに書き込む．送信操作では，マッチング領域に書き込まれたメッセージヘッダを確認することにより同期がとられ，遠隔メモリ書き込みで受信バッファにメッセージ本体を書き込む．一対一通信におけるネットワークの遅延は一往復の遅延であるが，その遅延はノンブロッキング受信を先行発行し，通信と計算をオーバラップさせることにより隠蔽することができる．これらの操作において，マッチング領域は排他的に指定されキュー操作を必要としないため，受信操作におけるメッセージヘッダの遠隔メモリ書き込みは，送信側に問い合わせることなくただちに行うことができ，また送信操作におけるメッセージ本体の遠隔メモリ書き込みも，対応する受信が発行されていればただちに行うことができる．

2.3 FMPL ー コピー ー 対一通信

零コピーの一対一通信では，送信関数は対応する受信関数が発行されるまで完了せず，同期通信あるいはランデブ通信のセマンティクスとなる．したがって，対応する受信関数がなかなか発行されない場合，送信側の受信待ちオーバヘッドが大きくなってしまう．また，たとえばプロセスのペアでデータを交換する場合，両方のプロセスがまずブロッキング送信を発行してしまうとデッドロックとなってしまう．これらを防ぐために，FMPL では一コピーの一対一通信（バッファモード）も提供する．このとき，送信処理ではまず一時バッファにメッセージをコピーし，そのメッセージは対応する受信が発行されたときに受信バッファにコピーされる．FMPL では，送受信の API を増やすことによる煩雑さを防ぐため，この一コピー一対一通信のために零コピー通信とは別の API を提供せず，その代わりプロトコルの切替えのためのタイムアウトを設定する API を提供し，動的に切替えを行う．

一コピー通信では eager プロトコルにより，一時バッファは受信側に確保し，受信側でメッセージマッチングを行う方が，受信時の遅延を短くすることができ望ましい．しかしながら，送信側メッセージマッチングの低オーバヘッド，低遅延を活用したいことと，ここで受信側メッセージマッチングに動的に切り替えるためには，切替え中の一貫性を保つために余分なコントロールメッセージと余分なヘッダ待避領域が必要となり[18]，一律に通信オーバヘッドが増えてしまうため，FMPL ではこの一時バッファを送信側にとる．

FMPL における一コピー一対一通信は，通信性能を最大限に発揮するためのものではなく，あくまで受信待ちオーバヘッドの回避，デッドロックの回避のために準備されたものである．したがって，一コピー一対一通信となってしまう場合は，ノンブロッキング受信を先行発行するなど，零コピー一対一通信となるようにプログラムを最適化することが望ましい．

2.4 送信プロセスランクを指定しない通信

送信側メッセージマッチングでは，受信関数は送信側のマッチング領域にメッセージヘッダを書き込むため，受信側は送信プロセスのランクを ANY ではなく明示的に指定する必要がある．一方で，サーバプログラムやアクティブメッセージ処理などのように，不特定多数から受信したメッセージを受信順に処理したいというような場合がある．

送信プロセスランクに ANY を指定して受信するためには，メッセージマッチングは受信側で行う必要が

あるが，FMPL の場合，前述のように送信側でメッセージマッチングを行っているため，このとき動的プロトコル切替えが必要になってしまう．FMPL ではこの動的プロセス切替えにともなうオーバヘッドと複雑さを避けるために，送信プロセスランクを指定しない通信のための別な API を提供する．この一対一通信は，通常の一対一通信とは別の通信ドメインを持ち，受信側メッセージマッチングを可能とする．

この設計は，受信側が送信プロセスランクを ANY で受信しているかは送信プロセスが知っているという仮定に基づいている．一対一通信は送信プロセスと受信プロセスのペアで構成され，送信は受信プロセスを指定する．このとき，送信側は少なくとも受信プロセスが送信したメッセージを受信することを知っており，さらに，どうしてそのメッセージを送信するのか，そのメッセージがどのように処理されるかも知っていることになる．つまり，送信プロセスは受信プロセスがどのように受信するかを知らないで，プログラムを書くことができないため，この仮定は妥当であると考えられる．

2.5 メッセージタグを指定しない通信

メッセージタグに ANY を指定する受信の場合は，送信プロセスランクが ANY で指定されない限り，送信側メッセージマッチングで実装することができるため，FMPL では受信のメッセージタグに ANY も提供している．

2.6 MPI 一対一通信との比較

MPI 一対一通信は，標準モード，バッファモード，同期モード，レディーモードの 4 モードがあり，さらにそれぞれのモードにブロッキング通信とノンブロッキング通信があり，主なものだけでも合わせて 8 種類の API が存在する．FMPL では基本的にブロッキング通信とノンブロッキング通信の 2 種類しか提供しないが，通信モードに関しては，零コピー通信，一コピー通信のプロトコル切替えのタイムアウトの設定によって行っている．バッファモード，同期モードはそれぞれタイムアウトを 0 あるいはなしにしたものである．標準モードは MPI では特にセマンティクスは規定されていないが，バッファリングすることが望ましいと記述されているため[12]，FMPL ではタイムアウトを適当な値に設定したものが相当している．レディーモードは FMPL では特に提供されないが，タイムアウトなしで代用する．

ただし，FMPL は送信プロセスランクを指定しない通信に関して，異なる通信ドメインとして別の API を提供しているところが大きく異なる．メッセージタグの範囲では，MPI は 0 から少なくとも 32767 まで指定できることが定められている[12]．一方，FMPL はメッセージタグとしてマッチング領域を指定し，その領域ではキュー管理をしないため，マッチング領域のインデックスの範囲は物理メモリ量，ハードウェア資源などに限られてしまう．

また MPI ではメッセージ本体について，データ型と配列長という指定であるが，FMPL では連続データサイズをバイト数で指定する．MPI では C 言語の構造体より汎用的なユーザ定義派生データ型を作成することができ，連続領域でないメッセージ本体を送受信することができるが，FMPL では連続領域しか扱わない．

2.7 集合通信

一対一通信は，分散メモリ型並列計算機のプログラミングの基本操作であり，木構造によるブロードキャスト，リダクションなど多くの集合通信も一対一通信を用いて効率的に実装できる．FMPL の提供する集合通信は，低オーバヘッドの FMPL 一対一通信を用いたものと，利用可能であればハードウェア通信機構を利用したもので構成される．一般的に，ハードウェア通信機構には利用条件の制約があり，またデータサイズ，ノード数などによりそれぞれの性能が変わるため，性能評価により適応的に効率的な実装を選ぶ．

3. FMPL API

本章では FMPL の API の解説を行う．

3.1 初期化と終了処理

初期化および終了処理の関数は以下のインタフェースとなる．

```
fmpl_init(ma, fa, count, ier)
fmpl_finalize(ier)
```

fmpl_init は FMPL の実行環境の初期化を行い，マッチング領域および受信完了を知らせるための受信完了フラグ領域を指定する．ma はマッチング領域の先頭アドレス，fa は受信完了フラグの先頭アドレス，count はその要素数である．ier はリターンコードであり，正常終了時は FMPL_SUCCESS が返り，エラーが発生した場合はエラーコードが入る．

3.2 一対一通信

3.2.1 ブロッキング通信

ブロッキング通信のための送信，受信関数は以下のインタフェースとなる．

```
fmpl_send(buf, size, dst, mai, comm, ier)
fmpl_recv(buf, size, src, mai, comm, ier)
```

fmpl_send は先頭アドレス buf，バッファサイズ size

で指定される送信バッファを，コミュニケータ comm のプロセスランク dst に送信する．mai はメッセージマッチングのためのマッチング領域のインデックスであり，ier はリターンコードである．コミュニケータは 3.3 節で説明する．fmpl_recv は先頭アドレス buf，バッファサイズ size で指定される受信バッファに，コミュニケータ comm のプロセスランク src から受信する．mai はメッセージマッチングのためのマッチング領域のインデックスであり，ier はリターンコードである．

マッチング領域のインデックスはメッセージタグに相当するものであるが，この領域ではキュー管理を行わないため，送信，受信のペアにおいて生存期間が重なる場合は同一のインデックスを用いることはできない．受信関数では，マッチング領域のインデックスとして FMPL_TAG_ANY を指定することができ，この場合送信関数のどのインデックスともマッチングがとられる．しかしながら，FMPL_TAG_ANY を指定した受信関数と，明示的にインデックスを指定した受信関数の間には，FIFO の関係は仮定されず，たとえば，同じ送信プロセスランクに対しインデックス 0 を指定した受信と，ANY を指定した受信が発行された場合，インデックス 0 を指定した送信は，受信の発行順序とは無関係にマッチングがとられる．

fmpl_send と fmpl_recv は，それぞれ送信領域に安全に書き込み可能となったとき，受信領域に全データを受信したときに完了する．零コピーでブロッキング通信を実装する場合，送信と受信はそれぞれ対応する受信と送信が発行されない限り完了せず，MPI でいう同期モードとなってしまう．FMPL では，この同期モードの通信を回避するために，送信データを一時的に待避するバッファ領域とモード切替えのタイムアウトをあらかじめ指定し，一コピー通信のバッファモードに切り替える．

```
fmpl_sendbuf_set(buf, size, ipara, ier)
fmpl_sendbuf_check(nsend, nspool, ier)
```

fmpl_sendbuf_set は一時待避用のバッファの先頭アドレス buf，バッファサイズ size およびタイムアウト時間 ipara を指定する．このとき fmpl_send は少なくとも ipara ミリ秒間は対応する受信のメッセージヘッダを待ち，タイムアウトした場合は，待避用バッファにコピーし fmpl_send は完了する．一時待避用のバッファは FMPL ライブラリにより管理され，ユーザプログラムにより変更を加えてはならない．

fmpl_sendbuf_check は待避用バッファを調べ，対応する受信が発行されていれば，送信処理を行う．nsend は，実行された送信処理数が入り，nspool はまだ待避用バッファに残っていて実行待ちの送信処理数が入る．

3.2.2 ノンブロッキング通信

以下はノンブロッキング通信のためのインタフェースである．

```
fmpl_isend(buf, size, dst, mai, comm, ier)
fmpl_irecv(buf, size, src, mai, comm, ier)
fmpl_isend_wait(dst, mai, comm, ier)
fmpl_irecv_wait(src, mai, comm, ier)
```

ノンブロッキング通信では fmpl_isend が完了してもまだ送信バッファを変更することはできず，変更してしまうと送信データが保証されなくなってしまう．同様に，fmpl_irecv が完了しても fmpl_wait_irecv を発行し完了するまでは，データを受信したことは保証されない．FMPL におけるノンブロッキング通信は，特に fmpl_irecv を前もって発行することにより効果的となり，このとき対応する fmpl_isend あるいは fmpl_send はただちに送信することができるため，送信待ちのオーバヘッドを減らすことができるだけではなく，それぞれの処理は送信側あるいは受信側だけのローカルな処理であり，計算と通信をオーバラップさせ通信オーバヘッドを隠蔽することができる．

3.2.3 送信プロセスランクを指定しない通信

以下は送信元を明示的に指定しない一対一通信のインタフェースである．

```
fmpl_send_any(buf, size, dst, mai, comm, ier)
fmpl_recv_any(buf, size, mai, comm, ier)
```

ここで，fmpl_send_any と fmpl_send の通信ドメインは別々のものであり，fmpl_send で送ったメッセージは fmpl_recv_any では受け取ることができず，また同様に fmpl_send_any で送ったメッセージは fmpl_recv では受け取ることができない．このことは，fmpl_send_any，fmpl_recv_any の実装に，受信側メッセージマッチングといった fmpl_send，fmpl_recv とは別の実装を可能とする．

3.3 集合通信

FMPL における集合通信は，プロセスグループを表すコミュニケータを用い，バリア同期，ブロードキャスト，リダクションを提供する．すべての集合通信は，指定したコミュニケータに含まれるすべてのプロセスにより呼び出す必要がある．

コミュニケータ作成は以下のインタフェースを用いる．

```
fmpl_comm_create(comm, key, ier)
```

fmpl_comm_create は全プロセスで呼び出し，key によりそのプロセスグループに参加するかどうかを決定

する．プロセスグループ番号は comm に返される．全プロセスが参加する FMPL_COMM_WORLD はあらかじめ定義されている．

3.3.1 ブロードキャスト

ブロードキャストは，以下のインタフェースとなる．

```
fmpl_bcast(buf, size, root, comm, ier)
```

fmpl_bcast はプロセスランク root の先頭アドレス buf とサイズ size で指定される送信バッファをコミュニケータ comm の他の全プロセスにブロードキャストする．コミュニケータ comm に含まれる全プロセスは同じ放送元プロセス root を指定して fmpl_bcast を呼ぶ必要がある．

3.3.2 リダクション

リダクションは，以下のインタフェースとなる．

```
fmpl_allreduce(buf, count, func, comm,
               work, ier)
fmpl_reduce(buf, count, func, root, comm,
            work, ier)
```

fmpl_allreduce は，buf のデータを func で指定されるリダクション処理し，全プロセスの buf を更新する．work は buf と同じ大きさの作業領域である．func では，fmpl_vsum，fmpl_vmax，fmpl_vmin があらかじめ提供され，それぞれ総和，絶対値最大，絶対値最小を計算することができるほか，ユーザ定義関数を指定することもできる．それぞれの関数の先頭文字 v には BLACS 同様にデータ型を表す文字が入る．count は buf の要素数が入る．fmpl_reduce は，buf のデータを func で指定されるリダクション処理し，プロセス root の buf を更新する．

3.3.3 バリア同期

バリア同期は以下のインタフェースで与えられる．

```
fmpl_barrier(comm, ierr)
```

fmpl_barrier はコミュニケータ comm のすべてのプロセスが fmpl_barrier を発行するまでブロックする．

4. SR8000 における通信ハードウェア

SR8000 のノードは基本的には 8-way SMP であり，プロセッサは PowerPC アーキテクチャを基に疑似ベクトル機構[8]，拡張レジスタ，拡張命令，LTLB などを追加したプロセッサである．それぞれのノードには協調型マイクロプロセッサ機構と呼ばれる複数のプロセッサをいっせいに高速に起動するハードウェア機構がついており，並列ループのループ分割による高速並列実行の支援をしている．

SR8000 のネットワークは多次元クロスバネットワークである．8 ノード構成では一次元クロスバ，64 ノード構成では二次元クロスバと，ノード数に応じてクロスバの次元が上がっていく．それら多次元クロスバネットワークを用い，SR8000 ではリモート DMA 転送，ハードウェアバリア機構，ハードウェアブロードキャスト機構を備えている．

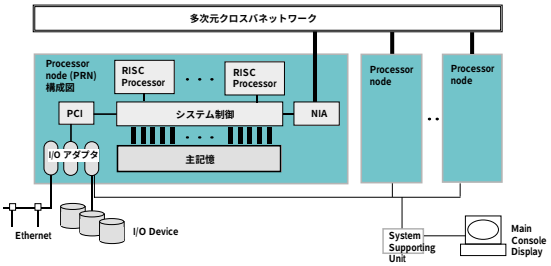

図 2 SR8000 のプロセッサノードの構成
Fig. 2 Processor node diagram of SR8000.

図 2 に SR8000 のプロセッサノードの構成を示す．SR8000 のプロセッサノードは大きくプロセッサ，メモリ，NIA（Network Interface Adapter）で構成される．ノード間通信において NIA は送信と受信の 2 つの処理を実行する．送信処理では，NIA はプロセッサからの送信指示に従ってローカルメモリからデータを取り出して，ノード間通信の単位であるパケットを構成し，ノード間ネットワークへ送り出す．パケットは，ヘッダ部とデータ部からなる．ヘッダ部には，受信ノード番号，送信データサイズ，リモート DMA 識別子などの情報が格納される．受信処理では，NIA はノード間ネットワークからパケットを受信してデータを取り出し，ローカルメモリに書き込む．

4.1 リモート DMA 転送

通常のデータ転送方式では，ユーザプログラムから別のノードのユーザプログラムへ送信するデータを，一度ユーザプログラム空間からカーネル内の送信バッファへコピーした後，パケット単位に編集して相手のノードへ送信する．受信ノードでは，カーネル内の受信バッファに受信したパケット単位のデータから受信データを生成し，ユーザプログラム空間内へコピーすることによって受信データを渡す．このためユーザプログラム空間とカーネル空間の間で少なくとも 2 回のデータコピーが発生し，データ転送の通信遅延が大きくなる．

これに対しリモート DMA 転送方式（**図 3**）では，ユーザプログラム空間内の送信データ領域（仮想メモリ）とカーネル内のリモート DMA 転送用の送信領域（物理メモリ）をあらかじめ一対一にマッピングしておく．同様に，受信ノード側でもカーネル内のリモート DMA 転送用の受信領域（物理メモリ）とユーザプロ

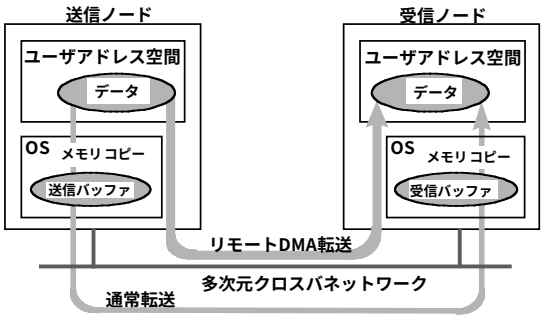

図 3 リモート DMA 転送
Fig. 3 Remote DMA transfer mechanism.

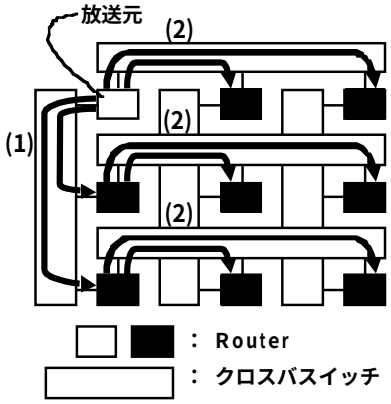

図 4 ハードウェアブロードキャスト機構
Fig. 4 Hardware broadcast mechanism.

グラム空間内の受信データ領域（仮想メモリ）をあらかじめ一対一にマッピングしておく．そのうえで多次元クロスバネットワークを経由した DMA 転送によって，ノード間にまたがったユーザプログラム空間内の送受信領域間でデータを直接転送することができる．

メッセージの送信は NIA の制御レジスタに転送制御テーブル（TCW）の先頭アドレスを書き込むことで起動される．TCW には，送信データのアドレス，サイズ，相手先ノード，リモート DMA 識別子，オフセットなど，データ転送に必要な情報が格納される．

このように，カーネルを介さずデータを送受信することができるため，ノード間データ転送の通信遅延を小さくすることができ，高速なノード間通信を実現することができる．

4.2 ハードウェアブロードキャスト機構

図 4 にハードウェアブロードキャスト機構を示す．同時に複数のハードウェアブロードキャストが実行されないように，システム中に一意に「シリアライズクロスバ」が決められる．二次元クロスバの場合，シリアライズクロスバとして決められている y-クロスバから，全 x-クロスバを介してシステム中のすべてのノードにデータを転送する．ノードからシリアライズクロスバへのデータ転送は x-クロスバを経由して一対一転送で行う．二次元クロスバの各クロスバは完全クロスバになっているため，シリアライズクロスバはその完全クロスバ性を利用し，各 x-クロスバへの展開を同時並行動作的に行うことができる．

ブロードキャストの起動は，通常のリモート DMA 転送と同じであるが，リモート DMA 領域の受信フィールドに対する送信権獲得時にブロードキャスト属性を指定する．

ハードウェアブロードキャストを利用するためには，以下のような条件が必要となる．

(1) プロセスは別々のノードで実行される．
(2) プロセスを実行するパーティションは，排他属性およびグローバル属性が設定されている．
(3) 実行するパーティションは，共用属性のパーティションと重なりあっていない．
(4) プロセスは，直方体形状に配置されている．直方体形状には，直線および長方形の形状も含まれる．

さらに，ブロードキャストするアドレスに関しても，送信領域と受信領域のオフセットが各ノード間で同一であるという制限がつく．

4.3 ハードウェアバリア機構

図 5 に二次元クロスバの場合のハードウェアバリア機構を示す．ハードウェアバリア機構は，二次元クロスバの各クロスバが完全クロスバであることを利用し，各 x-クロスバ上で並行して接続されているノードからの同期点到達判定を行い，それを y-クロスバに伝える．このとき特定の y-クロスバに集めるのではなく，すべての y-クロスバに x-クロスバでの同期を伝える．このことにより，各 y-クロスバにおいて，並行動作的に xy 平面の同期を確認し，各 y-クロスバから x-クロスバを介さずに並行して各ノードに同期成立を伝えることができる．

ハードウェアバリア機構は SR8000 のリモート DMA 転送ライブラリの hmpp_barrier を用いる．hmpp_barrier では同期要因を表すカラーとして 0 から 7 の 8 種類が用意されている．

ハードウェアバリア機構はハードウェアブロードキャストの適用条件のときのみ利用可能となる．hmpp_barrier はそれ以外の場合でも使用可能であり，その場合はハードウェアブロードキャスト機構とリモート DMA 機構をソフトウェア的に組み合わせてバリア同期がとられる．

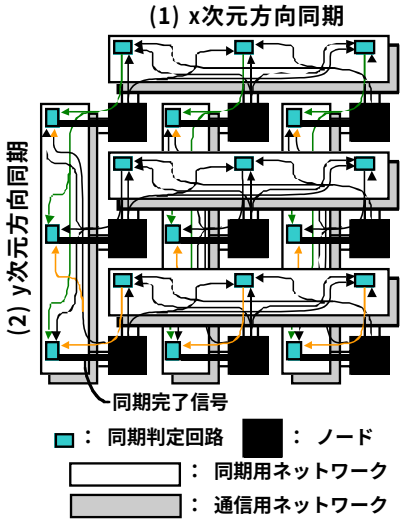

図 5 ハードウェアバリア機構
Fig. 5 Hardware barrier mechanism.

5. FMPL の SR8000 上での実装

5.1 一対一通信

SR8000 のリモート DMA 転送は，TCW の作成と，そのアドレスを NIA の制御レジスタに書き込むことにより起動される．TCW は一度作成すれば，内容の変更がない限り再利用することができ，その場合は，combuf_kick_tcw あるいは combuf_kick_tcw_fast を用い NIA の起動のみで高速に送信することができる．この送信は **TCW 再利用型送信**と呼ばれている．送信，受信のマッチングに利用されるマッチング領域への書き込みは，マッチング領域作成時に TCW を作成することにより，この TCW 再利用型送信により送信することができる．

図 6 に零コピー通信の処理の流れを示す．零コピー通信ではユーザ空間およびマッチング領域はリモート DMA 転送可能なようにリモート DMA 領域となっている．受信が発行されると，受信領域のヘッダ情報をマッチング領域に設定し，そのヘッダ情報を送信ノードに combuf_kick_tcw_fast を用い TCW 再利用型送信により送信する．一方，送信側は，そのオフセットを送信情報にセットし，リモート DMA 転送により送信領域を受信領域に転送し，マッチング領域をリセットする．

対応する受信のメッセージヘッダがタイムアウト時間をすぎても到着しない場合，メッセージは送信側の一時待避バッファにコピーされる．

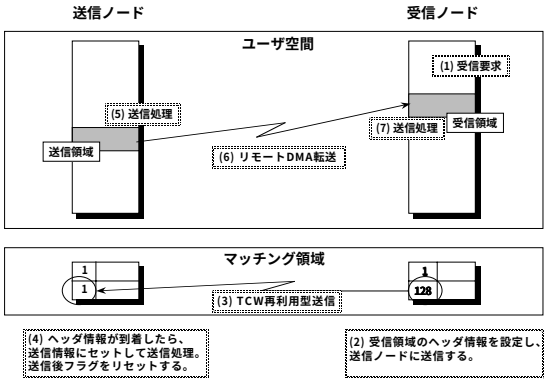

図 6 零コピー通信の処理の流れ
Fig. 6 Implementation of zero-copy blocking point-to-point communication.

5.2 集合通信

ブロードキャストは，FMPL の低コスト一対一通信を二分木状に利用したソフトウェアブロードキャストと，SR8000 のハードウェアブロードキャスト機構を利用したものの 2 種類を実装した．ハードウェアブロードキャストは 4.2 節の制限がつくが，このうち特に，全ノードにおいて送信，受信領域のオフセットが同一という制限の実行時チェックは実質的に難しい．したがって，あらかじめ 2 面のバッファを静的リモート DMA 領域として用意し，メモリコピーとハードウェアブロードキャストをパイプライン処理でオーバラップさせながら利用する．これら 2 つの実装は，ノード数，データサイズを変えて性能評価を行い，効率的な方が選ばれる．

全実行ノードを含むバリア同期の場合，4.3 節の制限が満たされればハードウェアバリア同期機構を利用することができる．しかしながら hmpp_barrier は制限が満たされない場合でも，ハードウェアブロードキャスト機構とリモート DMA 機構をソフトウェア的に組み合わせてバリア同期がとられる．また，任意のコミュニケータに対するバリア同期として，FMPL の一対一通信の軽い同期操作と同様の TCW 再利用型送信を二分木状に用いることによりバリア同期をとるソフトウェアバリア同期も実装した．

リダクションはハードウェア通信機構がないため，FMPL 一対一通信を二分木状に利用し実現している．

5.3 BLACS-FMPL

BLACS は行列演算パッケージ ScaLAPACK および PBLAS の通信層である．BLAS で行列演算の一般的な基本演算を定義したのと同様に，BLACS では一般的な基本通信が定義されており，行列演算のための簡単で使いやすいインタフェースを目指し設計されて

いる．

BLACS-FMPL は FMPL の上の BLACS の実装であり，Netlib[9] で公開されている MPI による実装（MPIBLACS）を基にしている．BLACS の一対一通信のセマンティクスはブロック通信であり，また送信はローカルに終了する必要があるため，暗黙的にバッファモードが仮定されている．FMPL の一対一通信は，タイムアウトを適切に設定することにより，このセマンティクスを満たすことができる．

行列演算では部分行列，三角行列の送信，受信を行うことが多いため，BLACS の一対一通信では送信，受信バッファは二次元（部分）配列あるいは二次元配列の下三角部分あるいは上三角部分を指定する．以下は BLACS の送信関数のインタフェースである．

```
vGESD2D( ICONTXT,
         M, N, A, LDA, RDEST, CDEST )
vTRSD2D( ICONTXT, UPLO, DIAG,
         M, N, A, LDA, RDEST, CDEST )
```

ICONTEXT は通信コンテキストであり，送信バッファは M, N, A, LDA により二次元配列として指定される．一次元配列であっても，必ずこの指定となる．RDEST, CDEST はコンテキストの二次元プロセスグリッドにおけるプロセス番号である．UPLO は上三角（台形）あるいは下三角を指定し，DIAG は対角部分は 1 として処理されないか，処理するかを指定する．関数名のはじめの v は I, S, D, C, Z のいずれかであり，それぞれデータ型が整数，単精度実数，倍精度実数，単精度複素数，倍精度複素数であることを表している．

MPIBLACS では，それらの部分行列，三角行列の送受信のたびにユーザ定義派生データ型を作成しており，この部分が大きなオーバヘッドとなっている．BLACS-FMPL ではこのデータ型生成のオーバヘッドと，データパッキング，アンパッキングのオーバヘッドを可能な限り削減している．まず，送信，受信する部分行列が連続かどうかを調べ，もし連続であればパッキングを行わない．仮にパッキングが必要な場合でも，あらかじめ静的に割り当てられたリモート DMA 領域を 2 面用意しておき，パック，アンパックと送信，受信をパイプライン処理によりオーバラップさせる．

6. 性能評価

この章では，本研究により作成した FMPL の基本通信の性能評価を行う．評価にあたり，産業技術総合研究所先端情報計算センタ（TACC）の日立 SR8000 64 ノードのうち 16 ノードを使用した．TACC の SR8000 のプロセッサの動作周波数は 250 MHz であり，ノードあたりのピーク性能は 8000MFLOPS となる．ネットワークは 64 ノードまでは二次元クロスバであり，それぞれのリンクのバンド幅は単方向 1,000 MB/s である．

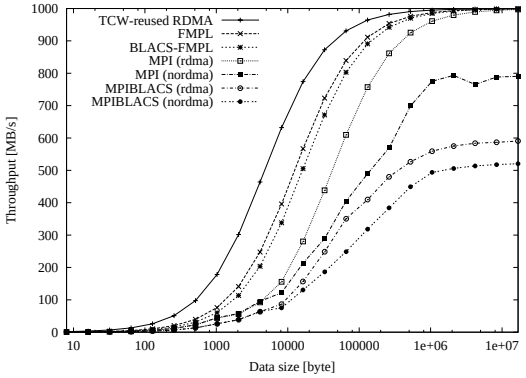

図 7 一対一通信のバンド幅
Fig. 7 Bandwidth of point-to-point communication.

6.1 一対一通信

図 7 に TCW 再利用型送信，FMPL，BLACS-FMPL，MPI および MPIBLACS の一対一通信のバンド幅を示す．このバンド幅は ping-pong 通信の通信遅延を基に求めている．TCW 再利用型送信は受信関数を明示的に発行しないため一対一通信ではないが，SR8000 のピーク性能を示すために計測した．MPI は日立提供のものであり，MPIBLACS は netlib で提供されている MPI ベースのものである．括弧付きの rdma はコンパイル時に `-rdma` オプションを付けて，ユーザプログラム中の静的領域を静的リモート DMA 領域とし MPI の送受信を零コピー通信としたものであり，nordma は送信側，受信側でそれぞれユーザ領域とリモート DMA 領域の間のコピーが発生するバッファ通信としたものである．

FMPL，BLACS-FMPL および MPI で零コピー通信をしたものはいずれもネットワークのピークバンド幅の 1,000 MB/s にほぼ近い性能を達成しており，16 MB のデータサイズでは，FMPL，BLACS-FMPL は 999 MB/s，MPI（rdma）は 997 MB/s を達成した．MPI（nordma）は送信時にユーザ空間からリモート DMA 領域へ，受信時にリモート DMA 領域からユーザ領域にコピーが必要であり，16 MB で 791 MB/s とピークバンド幅の 80%程度しか達成していない．MPIBLACS では送信，受信にユーザ定義派生データ型を作成し，データをパックしてから MPI で転送しているためさらにオーバヘッドが増えている．そのため，16 MB のときのバンド幅は rdma と nordma でそれぞれ 591 MB/s，522 MB/s となっている．その一方

表 1 8 バイト転送時の送信，受信処理の内訳
Table 1 Breakdown of point-to-point communication.

送信	初期処理	ヘッダ受信	データ送信
	0.5	5.0	8.1
受信	初期処理	ヘッダ送信	データ受信
	0.5	6.1	5.6

[μsec]

表 2 600 回ノンブロッキング受信を先行発行した場合の，連続して先行発行する受信のギャップと 4 バイト送信の遅延
Table 2 4-byte one-way latency and gap of succeeding nonblocking receive calls of point-to-point communication with six hundred outstanding nonblocking receive calls.

	先行発行受信ギャップ	送信遅延
FMPL	7.03	9.90
MPI	370.1	945.8

[μsec]

で，BLACS-FMPL はデータが連続かどうかを調べ，不必要なコピーを省いているため，ピークバンド幅に近い性能を達成している．

8 バイト転送時の片道の通信遅延は FMPL で 12.8 μsec，BLACS-FMPL では 15.7 μsec，MPI ではいずれの場合も 20 μsec，BLACS ではいずれの場合も 34 μsec であった．MPI では，1 度目の送受信で受信フィールドの送信権を獲得するための combuf_get_sendright を発行し，1 度目の送受信が遅くなるが，この通信遅延は 2 度目以降の遅延である．

FMPL では，受信発行時に送信側のマッチング領域へメッセージヘッダの書き込み，送信時に受信側へメッセージ転送と，メッセージは 1 往復となる．日立 MPI は，短いメッセージの場合 eager プロトコルとなり，送信側はメッセージヘッダと本体の両方を受信側に送り，受信側は受信完了と一時バッファ再利用可を示す ack を送信側に送るため，同じく 1 往復であるが，MPI ではメッセージ処理における動的オーバヘッドが大きいため，この遅延の差となっている．

FMPL における，8 バイト転送時の通信遅延の内訳を**表 1** に示す．この内訳はプロセッサのマシンサイクルカウンタを用い測定した．TCW 再利用型送信の遅延は 4.7 μsec であるため，低コストで一対一通信が実現できていることが分かる．ヘッダ送信時間に比べ，データ送信時間が大きいのは，ヘッダ送信は TCW 再利用型送信を行っているのに対し，データ送信は TCW を修正してからリモート DMA 転送を行っているためである．

表 2 に，600 回ノンブロッキング受信を先行発行した場合の，連続して先行発行する受信のギャップと 4 バイトの送信時間を示す．すべての送信に対応する受信のメッセージヘッダはすでに届いているため，送信はただちにリモート DMA 送信によりメッセージ本体を書き込むだけとなり，FMPL の送信の遅延は 9.9 μsec であった．この遅延も ping-pong 通信の通信遅延を基に求めている．日立 MPI は多くのノンブロッキング受信を先行発行した場合，オーバヘッドが大きくなっている．これは，それぞれのノンブロッキング受信発行時にそれ以前に発行されてまだ対応する送信が決まっていないすべてのノンブロッキング受信に対するチェックをしているためである．FMPL ではペンディングのノンブロッキング受信のキューを持たないため，そのようなオーバヘッドはまったくない．

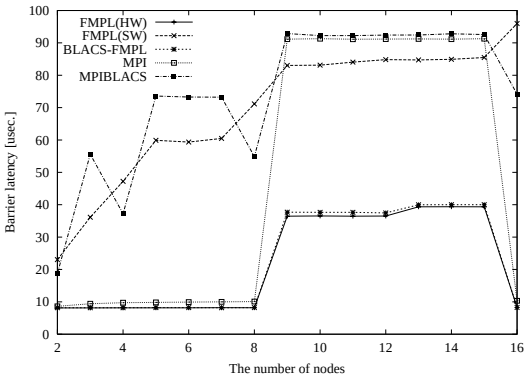

図 8 バリア同期時間
Fig. 8 Barrier synchronization.

6.2 バリア同期

図 8 にバリア同期時間を示す．FMPL（HW）はハードウェアバリア機構を利用するための hmpp_barrier を用いたものである．TACC の SR8000 は 64 ノード構成であるが，16 ノードのパーティション構成が 8×2 と y-クロスバ方向で分割されているため，ノード数が 2 ノードから 8 ノードまでと 16 ノードのときに物理ノード構成が矩形となり，ハードウェアバリア機構が利用できる．FMPL（SW）では二分木で通信しバリア同期をとっているため，バリア同期時間はプロセス数 p に対し $O(\log p)$ となっている．本ソフトウェアバリアの実装では，プロセス数分のバッファ領域を利用しているため，2 の巾乗のプロセス数以外のときは通信段数が 1 段少なくなっている．FMPL（HW）はすべてのノード数で FMPL（SW）より速いため，全実行プロセスでバリア同期を取る場合は，FMPL（HW）がつねに利用される．FMPL-BLACS は FMPL（HW）とほぼ同様の性能となっている．

MPI ではノード数が 2 ノードから 8 ノードまでと 16 ノードのとき，ハードウェアバリア機構を用いて，非

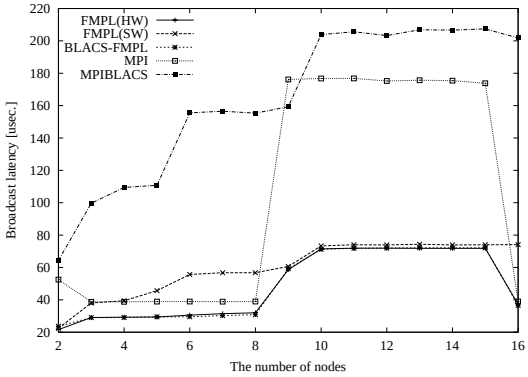

図 9 長さ 8 KB の配列のブロードキャスト時間
Fig. 9 Broadcast latency (8 KB data).

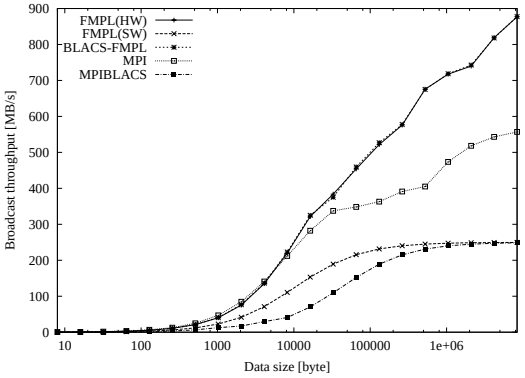

図 10 16 ノードにおけるブロードキャストのバンド幅
Fig. 10 Broadcast bandwidth with 16 nodes.

常に高速にバリアをとっている．しかしながらノード数が 9 ノードから 15 ノードまでのときは，このハードウェアバリア機構を用いておらず，この場合は FMPL（SW）より遅くなっている．

6.3 ブロードキャスト

図 9 に長さ 8 KB の配列のブロードキャスト時間を示す．ブロードキャストの測定では，ルートプロセスを反復ごとに変えて，反復させて測定した．FMPL（HW）は 2 面の静的リモート DMA 領域を交互に利用し，ブロードキャストとコピーをオーバラップさせている．FMPL（SW）は一対一通信を用い二分木で通信するため，ブロードキャスト時間はプロセス数 p に対し $O(\log p)$ となっている．ブロードキャストの場合，FMPL（HW）はつねに FMPL（SW）より高速になっている．MPI では `-rdma` オプションの有無にかかわらず通信遅延はほとんど変わっていないため `-rdma` オプションをつけたものを示している．MPI では，バリア同期と同様にノード数が 2 ノードから 8 ノードおよび 16 ノードのとき，ハードウェアブロードキャスト機構が利用され，高速にブロードキャストを実現しているが，ノード数が 9 ノードから 15 ノードのときはハードウェア機構を利用することができず，オーバヘッドが大きくなっている．

図 10 に，16 ノードの場合のブロードキャストのバンド幅を示す．測定では，先ほどと同様にルートプロセスを反復ごとに変えて，反復させて測定した．ブロードキャストのバンド幅は，データサイズを遅延で割ったものである．FMPL（HW）はデータサイズが 8 MB のとき，877 MB/s を達成し，まだピーク性能に向かってバンド幅は上昇している．一方で，FMPL（SW）は 250 MB/s 程度しかでていない．FMPL-BLACS はほぼ FMPL（HW）と同じ性能を示している．FMPL は 8 MB のとき MPI のブロードキャストバンド幅を 57%も改善している．

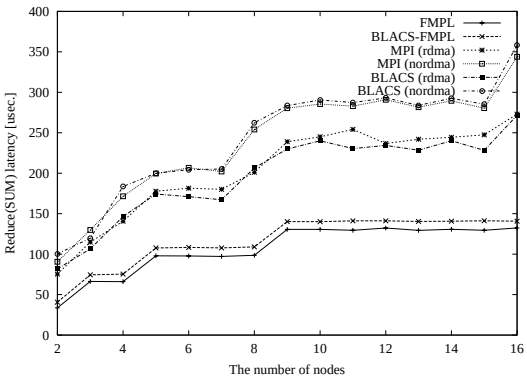

図 11 長さ 8 KB の配列要素の総和計算時間
Fig. 11 Array reduction of total sum (8 KB data).

6.4 リダクション

図 11 に長さ 8 KB の配列要素の総和を求めるリダクション fmpl_reduce の経過時間を示す．リダクション時間の計測は，ブロードキャストと同様にルートプロセスを反復ごとに変え，反復させて測定した．

リダクションはいずれのライブラリも二分木で通信して求めるため，リダクション時間はプロセス数 p に対し $O(\log p)$ となっている．8 KB 程度の小さい配列であれは，各プロセスの演算処理に比べ，通信遅延が大きく影響し，FMPL のリダクション時間は他の半分程度となっている．

7. 関連研究

高速メッセージハンドリングのための通信機構として，Active Messages[16]（AM）と Fast Messages[11]（FM）が提案されている．これらは，メッセージにそのメッセージを処理するハンドラのアドレスが含まれており，メッセージタグなどによるマッチングや検索などの必要なく，メッセージを処理することができ

る．しかしながら，AM も FM もそれらのメッセージ間の同期をとるための方法は提供していない．MPI-AM[17]，MPI-FM[6] は MPICH[4] を利用した MPI の実装であるが，いずれも受信側にメッセージマッチングのためのキューを導入している．一方で，FMPL ではそれらキュー管理は必要ない．また，MPI-FM は *upcall* と *gather* と呼ばれるテクニックを用い，送受信側双方でメモリ間コピーを削除している．

PMv2[13)~15)] は Myrinet，Ethernet，shmem のヘテロな環境における，低コストのメッセージパッシング通信および可能であれば遠隔メモリ操作を提供している．PMv2 のメッセージパッシング通信はメッセージタグを持たず，FIFO キューを提供するものである．MPICH-SCore[14] は MPICH と PMv2 を利用した MPI の実装であるが，これも受信側にメッセージマッチングのためのキューを導入している．ランデブプロトコルでは，MPICH-SCore は零コピー通信を用いている．

MPI-EMX[18] は電総研 EM-X データ駆動型並列計算機[5] 上の MPI の実装であり，送信側メッセージマッチング，受信側メッセージマッチングの動的プロトコル切替えの実装を行っている．動的プロトコル切替えのためには，送受信側双方にそれぞれ 2 つのメッセージキューとコントロールメッセージが必要となる．MPI-EMX では，遠隔メモリ操作，遠隔スレッド起動のほか，I-structures[1] を利用し，ポーリング，割込みなしにメッセージマッチングを行っている．

MPI/MBCF[19] は MBCF[7] を利用した MPI の実装であり，MBCF の提供するメモリベース FIFO を用いて送信側および受信がメッセージマッチングを行っている．

8. まとめと今後の課題

高速メッセージパッシングライブラリ FMPL の設計を行い，日立 SR8000 上に実装を行った．FMPL は遠隔メモリ操作を利用した汎用メッセージパッシングライブラリであり，送信側メッセージマッチング，メッセージキューの操作や動的メモリ確保など動的オーバヘッドの削減により，低遅延，高バンド幅を実現している．FMPL の一対一通信 API では，通信モードにより多くの API を準備するのではなく，通信モードの切替えはタイムアウト設定により行っている．

集合通信では，ハードウェア通信機構を利用したものと，FMPL 一対一通信を利用したものを適応的に利用する．FMPL は通信ハードウェア性能を最大限に引き出すことが必要なアプリケーションのためだけではなく，BLACS など上位のメッセージパッシングライブラリの構築のために設計されている．

SR8000 では，一対一通信の同期操作は受信側からのリモート DMA 転送で実装され，低コストを実現するために TCW 再利用型送信を用い SR8000 のネットワークハードウェアの起動のみで送信を行っている．この受信側からのリモート DMA 転送で，送信側に受信領域のヘッダ情報が伝えられ，送信側はリモート DMA 転送によりメッセージを書き込む．この方式により，通信遅延は 8 バイトのメッセージの一対一通信で片道 12.8 μsec を達成した．この通信遅延は MPI の 20 μsec，BLACS の 34 μsec を大幅に改善している．ノンブロッキング受信の先行発行により，FMPL の遅延はさらに 9.9 μsec に縮まる．通信バンド幅は，MPI に比べデータサイズが数 KB から数百 KB あたりで特に性能差が出ている．ブロードキャストでは，静的リモート DMA 領域を 2 面利用し，パイプライン処理によりコピーとブロードキャスト処理をオーバラップさせ，MPI のブロードキャストのバンド幅を 57%改善した．

現在 FMPL は SR8000 上の実装がほぼ終わり，今後，最適化された BLACS-FMPL を作成していく予定である．FMPL および BLACS-FMPL は，TACC で配布されることが予定されている．FMPL は SR8000 専用のものではなく，また他のプラットフォームでも実装していく予定である．

謝辞 たくさんの貴重なコメントをいただいた査読者の方々に感謝いたします．本研究を遂行するにあたり貴重なご助言，ご討論いただいた寺倉清之つくば先端情報センター長，グリッド研究センターのメンバ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. Prog. Lang. Syst.*, Vol.11, No.4, pp.598–632 (1989).
2) Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D. and Whaley, R.C.: *ScaLAPACK Users' Guide*, SIAM (1997).
3) Dongarra, J. and Whaley, R.C.: A User's Guide to the BLACS v1.1, Technical Report CS-95-281, University of Tennessee (1995).
4) Gropp, W., Lusk, E., Doss, N. and Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface stan-

dard, *Parallel Computing*, Vol.22, pp.789–828 (1996).
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* (*ISCA'95*), 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) Matsumoto, T. and Hiraki, K.: MBCF: A protected and virtualized high-speed user-level memory-based communication facility, *Proc. 1998 International Conference on Supercomputing* (*ICS98*), pp.259–266, ACM (1998).
8) Nakamura, H., Imori, H., Nakazawa, K., Boku, T., Nakata, I., Yamashita, Y., Wada, H. and Inagami, Y.: A Scalar Architecture for Pseudo Vector Processing based on Slide-Windowed Registers, *Proc. 1993 International Conference on Supercomputing* (*ICS93*), pp.298–307, ACM (1993).
9) netlib: http://www.netlib.org/, http://phase.hpcc.gr.jp/mirrors/netlib/
10) 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* (*ICS98*), pp.243–250, ACM (1998).
11) Pakin, S., Lauria, M. and Chien, A.: High performance messaging on workstations: Illinois fast messages (FM) for Myrinet, *Proc. Supercomputing '95* (1995).
12) Snir, M., Otto, S., Huss-Lederman, S., Walker, D. and Dongarra, J.: *MPI: The Complete Reference*, The MIT Press (1996).
13) Sumimoto, S., Tezuka, H., Hori, A., Harada, H., Takahashi, T. and Ishikawa, Y.: The design and evaluation of high performance communication using a Gigabit Ethernet, *Proc. 1999 International Conference on Supercomputing* (*ICS99*), pp.260–267 (1999).
14) Takahashi, T., Sumimoto, S., Hori, A., Harada, H. and Ishikawa, Y.: PM2: A High Performance Communication Middleware for Heterogeneous Network Environments, *Proc. Conference on High Performance Networking and Computing* (*SC2000*) (2000). CD-ROM.
15) Tezuka, H., Hori, A., Ishikawa, Y. and Sato, M.: PM: An Operating System Coordinated High Performance Communication Library, *Proc. High-Performance Computing and Networking '97*, Lecture Notes in Computer Science, Vol.1225, pp.708–717 (1997).
16) von Eicken, T., Culler, D.E., Goldstein, S.C. and Schauser, K.E.: Active Messages: A Mechanism for Integrated Communication and Computation, *Proc. 19th International Symposium on Computer Architecture*, pp.256–266 (1992).
17) Wong, F.C. and Culler, D.E.: *Message Passing Interface Implementation on Active Messages*. http://now.CS.Berkeley.EDU/Fastcomm/MPI/
18) 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，情報処理学会論文誌，Vol.40, No.5, pp.2246–2255 (1999).
19) 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 の実装と評価，情報処理学会論文誌，Vol.40, No.5, pp.2256–2268 (1999).

(平成 13 年 9 月 3 日受付)
(平成 14 年 2 月 13 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．独立行政法人産業技術総合研究所グリッド研究センター．グリッドコンピューティング，並列数値アルゴリズム，並列計算機システムの研究に従事．理学博士．日本応用数理学会，ACM 各会員．

長嶋 雲兵（正会員）

昭和 30 年生．昭和 58 年北海道大学大学院理学研究科修了．理学博士．独立行政法人産業技術総合研究所グリッド研究センター．計算化学，情報化学，広域分散並列処理．IEEE，日本化学会，日本コンピュータ化学会，応用数理学会各会員．

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．平成 13 年 4 月より独立行政法人産業技術総合研究所．以来，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．

北林　久義

昭和 44 年生．平成 3 年拓殖大学政経学部経済学科卒業．同年日立ビジネスソリューション（株）入社，以来，主に行列計算ライブラリの研究・開発に従事．

林田　義幸

昭和 40 年生．昭和 59 年熊本県立天草工業高校電気科卒業．同年日立製作所入社，以来，ソフトウェア事業部にて主に FORTRAN コンパイラの研究・開発に従事．