# メッセージパッシングモデルを支援するパケット受信機構のDIMMnet-2への実装と評価

北村 聡[†] 宮部保雄[†] 中條拓伯[††]
田邊 昇[†††] 天野英晴[†]

PC クラスタに代表される分散メモリ型の並列システムにおいては，各ノードで動作するプロセス間の通信に，メッセージパッシングが用いられる．メッセージパッシングにおいて各プロセスはメッセージを明示的に送信，および受信することにより並列処理を実現する．メッセージパッシングの実装方法として，リモートプロセスのメモリ領域へ直接データを転送する RDMA を用いる手法があげられる．本論文では，RDMA を用いて低レイテンシなメッセージパッシングを実現するために，受信側のネットワークコントローラがあらかじめ設定された受信領域にメッセージを受信する IPUSH 機構について述べる．IPUSH 機構では，メッセージ受信領域のアドレス管理をネットワークコントローラで行うため，ホストのオーバヘッドが低いなどの利点がある．IPUSH 機構をメモリスロット装着型ネットワークインタフェースである DIMMnet-2 の試作基板に実装し，通信性能を従来の RDMA と比較した．測定の結果，ネットワークインタフェース上の SO-DIMM 間で，ping-pong 転送時の最大バンド幅が 643 MByte/s，最小のレイテンシが 1.78 $\mu$s と，従来の RDMA と同等の性能を持つことが示された．

# Implementation and Evaluation of Packet Receiving Mechanism Supporting for Message Passing Model on DIMMnet-2

Akira Kitamura,[†] Yasuo Miyabe,[†] Hironori Nakajo,[††]
Noboru Tanabe[†††] and Hideharu Amano[†]

On distributed memory systems like PC clusters, a message passing is performed. If the interconnect supports Remote Direct Memory Access (RDMA), message passing is usually implemented with it. In such systems, parallel processes on each node must send a message indicating receiving processes explicitly. In this paper, we propose the IPUSH mechanism which supports low overhead message passing using RDMA. In the mechanism, a network controller receives massages in a pre-defined address space, and completely manages them in order to reduce the overhead of the host. We implemented the mechanism in DIMMnet-2 prototype board, which forms a PC cluster by attaching into the memory slot of host PC. The result of evaluation shows that the maximum bandwidth of the mechanism is reached about 643 MByte/s and the minimum latency is about 1.78 $\mu$s at ping-pong communication. These results are comparable with the performance of traditional RDMA.

## 1. はじめに

PC クラスタに代表される分散メモリ型の並列システムにおいては，各ノードで動作するプロセス間の通信にメッセージパッシングが用いられる．メッセージパッシングにおいて各プロセスはメッセージを明示的に送信，および受信することで並列処理を実現する．メッセージの送受信において，リモートプロセスのメモリ領域へ直接データを転送する Remote Direct Memory Access（RDMA）をサポートしないインターコネクトの場合，一度カーネルのバッファにメッセージをコピーするなどの操作により，メッセージの送受信を実現する．しかしながら，カーネルを介した場合，受信のオーバヘッド，レイテンシは RDMA より増大する．

一方，Myrinet [1] や QsNET [2]，InfiniBand [3] のような，ネットワークインタフェースが RDMA をサポートするインターコネクトでは受信先のアドレスを事前

† 慶應義塾大学
Keio University
†† 東京農工大学
Tokyo University of Agriculture and Technology
††† 株式会社東芝研究開発センター
Corporate Research and Development Center, Toshiba Corporation

に通知しあうなどの手段によって，送信側が受信先のアドレスを知る必要があるものの，受信側が明示的な受信処理を行う必要がなく，受信側ホストのオーバヘッドが小さいという利点がある．そのため，RDMAをサポートしたインターコネクトでは，RDMA を用いてメッセージパッシングを実現することで高い性能を得ることができる．しかしながら，複数プロセスから単一の受信領域に対して RDMA を行う場合，各プロセスは独立に RDMA でメッセージを送信してしまうため，受信メッセージが書き潰されてしまうことが起こりうる．これを防ぐために，送信プロセスごとに受信領域を分け，送信側，受信側のいずれか，または双方で，メッセージ受信領域のアドレスをメッセージを受信（送信）した分だけインクリメントするといった対策をとる必要がある．また，受信領域を分けた場合，通信プロセス数が増大した際のスケーラビリティに乏しいという問題も存在する．さらに，メッセージを到着した順番に読み出すことが難しくなる．

そこで，本論文では受信側のネットワークコントローラがメッセージの受信先を指定し，メッセージの受信，受信領域のアドレス管理を行い，さらに受信したメッセージごと，またはパケットごとに受信ステータスを全プロセス共通のバッファに格納することにより，メッセージの到着順序の検知手法を提供することが可能なメッセージ受信方式を提案する．この受信ステータスには送信側のプロセス ID やデータサイズといった，受信したデータに関する情報が含まれており，受信ステータスをメッセージごとに格納するかパケットごとに格納するかは送信側がメッセージの送信要求を発行する際に指定可能なものとしている．

この手法により，送信側は RDMA でメッセージを送信でき，受信側においてもメッセージの到着順序を検知することができるため，メッセージパッシングを実現することが容易となる．また，この通信に関するすべての処理をハードウェアで行うため，高い性能を得ることが可能となる．このための機構を PC クラスタ向けネットワークインタフェースである DIMMnet-2 の試作基板上に実装し，通信性能の評価を行った．さらに，本機構では受信領域を複数の送信プロセスで共有することが可能な構造となっている．これにより，メッセージの受信頻度の高いプロセスには個別のメッセージ受信領域を確保し，頻度の低いプロセスにはそれらのプロセス全体でメッセージ受信領域を共有することで，メッセージ受信のための領域のサイズを削減することが可能となる．本評価では受信領域の使用量の削減についても検討を行った．

以降，本論文ではこの機構を Indirect PUSH（IPUSH）と呼称し，従来の RDMA を PUSH と呼称する．

以下，本論文では 2 章で IPUSH 機構と同様にメッセージパッシングをネットワークコントローラが支援する研究について触れ，3 章で本機構の設計について，4 章で DIMMnet-2 への実装についてそれぞれ述べ，5 章で DIMMnet-2 に搭載した場合の評価を示す．最後に 6 章でまとめる．

## 2. 関連研究

本章では IPUSH 機構の関連研究として，受信側のネットワークコントローラがメッセージの受信先を指定する方式のメッセージ受信機構，および，ネットワークコントローラが MPI のメッセージ受信処理をホスト CPU に代わり行う手法について述べる．

### 2.1 メッセージ受信機構

IPUSH 機構における “受信側のネットワークコントローラがメッセージの受信先を指定する” というメッセージ受信の手法は DIMMnet-1 のリモート間接書き込み[4),5)] や RHiNET-2[6)] の VPUSH[7)] でも実現されている．これらはともに，ネットワークコントローラである Martini[8)] に搭載された RISC プロセッサを用いたソフトウェア処理とハードワイヤードで実装された通信機構が協調処理を行うことによって実現されている．

DIMMnet-1 のリモート間接書き込みでは，送信側で受信側のメッセージ受信先アドレスが格納された領域を指すポインタ値をセットし，受信側にメッセージを送信する．受信側では，そのポインタ値が指す領域に格納された受信先のアドレスを取得し，そのアドレスに従ってメッセージを格納する．受信先のアドレスの取得，フロー制御，ACK パケットの処理などをすべて RISC プロセッサで実行している．しかし，RISC プロセッサが例外処理用の簡素なものであり，性能が高くなかったため，RDMA と比較すると大幅に性能が低下してしまっている[5)]．

RHiNET-2 で実装された VPUSH では，受信したメッセージのアドレス値の変更，および受信領域のアドレス管理のみを RISC プロセッサで行い，パケットのヘッダ解析や主記憶への DMA はハードワイヤード部で行われる．VPUSH は RDMA の最大バンド幅の 38%程度の性能を示したが，やはり，オンチッププロセッサの性能から，RDMA よりバンド幅の低下，受信側のレイテンシの増加が見られている[7),8)]．VPUSH ではパケット長によらず，RISC プロセッサの処理時

間が 6.6 $\mu$s で一定である．この値はデータサイズが 2,048 Byte のパケット受信の処理時間の約 60%に相当する．文献 7) によると，データサイズが 2,048 Byte 時の VPUSH のバンド幅が 180.3 MByte/s であり，仮にこのソフトウェアオーバヘッドが 0 になった場合は，処理時間が約 40%程度になることから，バンド幅は約 450 MByte/s に到達すると予測される．実際にはソフトウェアの処理と同等の機能のハードウェアによるオーバヘッドが加わり，若干低い値になると考えられるが，この値は RHiNET-2 の最大バンド幅に匹敵する値である[8]．

IPUSH 機構とリモート間接書き込み，VPUSH では，いずれもメッセージの受信先を受信側で指定するという点では同様であるが，リモート間接書き込みと VPUSH では送信プロセスごとにメッセージ受信バッファを分けた際のメッセージの到着順序の検知手法を提供していない．また，複数プロセスでメッセージ受信バッファを共有した際に，複数パケットに分かれた単一のメッセージがプロセス間でインタリーブして受信された場合のメッセージの読み出しに対応していない．IPUSH 機構では受信ステータスをメッセージごとまたはパケットごとに全送信プロセス共通のバッファに格納することでメッセージまたはパケットの受信順序をホスト側に提供しつつ，複数プロセスで受信バッファを共有する場合にはパケット単位で受信ステータスを格納することで受信メッセージがインタリーブされた場合にもメッセージを正常に読み出すことができる．

### 2.2 ネットワークコントローラによる MPI のメッセージ受信処理

MPI ではメッセージ受信の際に用いるメッセージキューとして posted receive queue と unexpected message queue の 2 つのキューを用いて，メッセージの受信処理を行う．posted receive queue は受信要求がメッセージの到着前に行われた際に，その要求をバッファリングするためのものであり，unexpected message queue はメッセージの到着が対応する受信要求より先であった場合に，受信データをバッファリングするためのものである．

そのため，メッセージを受信した際には posted receive queue を探索し，そのメッセージに対応する受信要求が存在するかどうかを調べる．対応する受信要求が存在する場合には，その要求に対するメッセージとして受信側のプロセスに受信され，posted receive queue から受信要求が削除される．対応する受信要求が存在しない場合には，受信したメッセージを unexpected message queue に格納する．

受信要求が発行された場合には，まず unexpected message queue を探索し，対応する受信メッセージがすでに受信されているかどうかを調べる．すでに受信されている場合はそのメッセージが受信側のプロセスに渡され，受信されていない場合は，その要求が posted receive queue に格納される．このように，MPI においてはメッセージの受信の際にキューの探索処理が行われるが，アプリケーションによってはキューの探索範囲が大きくなり，その結果，メッセージの受信処理に要するレイテンシが増加することになる[9]．

そのため，これらのキューの探索処理をネットワークインタフェース上のネットワークコントローラで行うことでホスト側の負荷の軽減，および探索処理の高速化をはかる研究が数多く行われている．Myrinet の MX [10] や QsNET の Tports [11] ではマッチングのための API が用意されている．また，MX を利用した MPICH [12] である MPICH-MX が Myricom 社[15] より提供されている．これらはプログラマブルなネットワークコントローラを利用し，ソフトウェアで実行されるものであるが，文献 13) では，posted receive queue と unexpected message queue の探索処理を行うハードウェアを提案している．しかしながら，キューの探索範囲を削減する手法ではないため，これらは文献 9) であげられている問題を本質的には解決してはいない．

一方，IPUSH 機構では，受信バッファを送信プロセスごとに独立して確保することが可能であり，このバッファを unexpected message queue として用いると，特定のプロセスからのメッセージのみが格納されているために，探索範囲が削減されると考えられる．

## 3. IPUSH 機構

IPUSH 機構とは “メッセージ受信側のネットワークコントローラがメッセージの受信先を指定し，受信を行う” メッセージ受信機構であり，受信領域のアドレス管理や，メッセージやパケットの到着順序の検知手法の提供，メッセージ受信用のバッファの使用量を抑える機能もあわせ持つ．

2 章で触れたリモート間接書き込みと VPUSH において，従来の RDMA に比べて大幅な性能低下を起こしていたことから，メッセージの受信処理をネットワークコントローラで高速に行う場合にはネットワークコントローラに高い性能が求められるといえる．この場合，高い性能を持つ受信機構の実現方法として，Myrinet のようにプログラマブルな高性能ネットワー

クコントローラ上でソフトウェアで実行するか，ハードワイヤードで専用の受信処理部をネットワークコントローラ上に搭載する 2 通りの手法があげられる．

ソフトウェアを用いた処理は柔軟性を持つ一方，一般に，同等の機能をハードワイヤードで実装した場合に比べ，処理に要するレイテンシが大きくなる．このことは文献 14) から読み取れる．文献 14) によると，Myrinet/PCI-X E-Card [15] の MPI レベルでの最小の送信レイテンシは約 3.5 $\mu$s であり，対して，QsNET II では 1.7 $\mu$s である．この差は主にネットワークインタフェースでのレイテンシによるものである．Myrinet のネットワークコントローラ LANai-XP が RISC プロセッサであり，この上でソフトウェアを用いて通信処理を行っているのに対し，QsNET II の Elan4 では RISC プロセッサを搭載しているものの，基本的な通信処理はハードウェアで実装されていることが影響していると考えられる．

そのため，IPUSH 機構ではアドレス管理，メッセージの受信処理などをすべてネットワークコントローラ上にハードワイヤードで実装する．これにより，送信側は RDMA でメッセージを送信し，受信側は IPUSH 機構で受信することで，低レイテンシなメッセージパッシングを実現することが可能となる．

メッセージの受信領域はプロセスごとに個別の領域をリングバッファとして確保可能とする．しかしながら，全プロセスに対して異なるメッセージ受信領域☆を確保すると受信領域を確保する領域☆☆が圧迫されるため，スケーラビリティに乏しくなる．そこで，IPUSH 機構では複数の送信プロセスでバッファを共有可能とすることで，この問題を解決する．

### 3.1 IPUSH 機構の設計

IPUSH 機構の概観を**図 1** に示す．

IPUSH 機構の機能を満たすために必要な要件は以下の事柄である．

- 送信プロセスごとに独立したメッセージ受信領域の制御
- メッセージまたはパケット受信時の受信ステータスの制御

送信プロセスごとに独立したメッセージ受信領域を確保し，アドレス管理を行うために，IPUSH 機構ではネットワークコントローラのメッセージ受信処理部に以下に示す 2 つのテーブルを追加する．

- Address Management Table（AMT）：各プロセスの受信領域の Head Pointer と Tail Pointer を管理するテーブル
- AMT Address Table：AMT にアクセスするためのアドレスを管理するテーブル

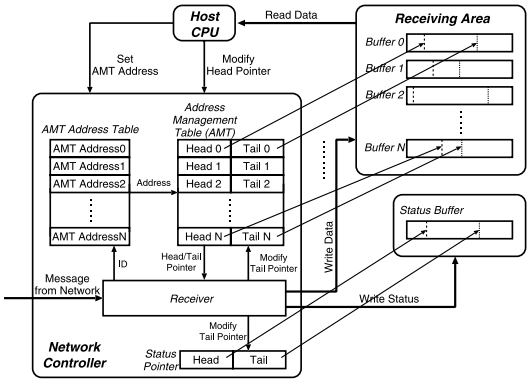


**図 1** IPUSH 機構
Fig. 1 Overview of IPUSH mechanism.

一般に並列システムにおいては，各プロセスに固有の ID が振られている．そこで，ネットワークコントローラ内で受信処理を行う Receiver は，メッセージ受信時に送信プロセスの ID を使用して AMT Address Table にアクセスする．AMT Address Table から出力された値を AMT のアドレス値とし，その ID に対応したプロセスの Head Pointer と Tail Pointer を得る．Head Pointer と Tail Pointer から受信したメッセージのサイズ以上に空きがあるか受信領域をチェックし，空きがある場合に受信処理を継続する．空きがない場合は以下のいずれかの手法をとることでメッセージの受信がなされることを保証する必要がある．

- 受信領域が空くまで処理を停止し，受信側のホストに受信領域が一杯であることを通知する．その後，ホストがメッセージの読み出しを行い，受信領域に空きをつくることでメッセージの受信処理を継続する．
- メッセージの破棄を行い，送信元にメッセージの再送を要求する．

この処理は，IPUSH 機構を既存のインターコネクトに搭載する場合は，そのインターコネクトで採用されている，メッセージの受信ができない場合の処理をそのまま利用することで実現可能である．たとえば，DIMMnet-1 や RHiNET-2 のネットワークコントローラである Martini ではメッセージが受信できない場合には NACK が送信元に返信される．

受信可能な場合はこの Pointer に従って，メッセージを受信領域に書き込む．書き込み完了後は AMT の Tail Pointer を更新し，処理を完了する．このテー

☆ ページアウトされない領域（ピンダウン領域）
☆☆ ホストの主記憶やネットワークインタフェース上のメモリ

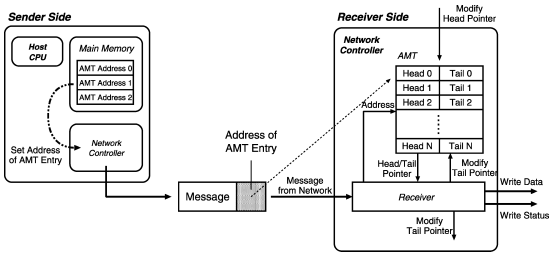


**図 2** AMT Address Table を用いない IPUSH 機構
Fig. 2 Overview of IPUSH mechanism without AMT Address Table.

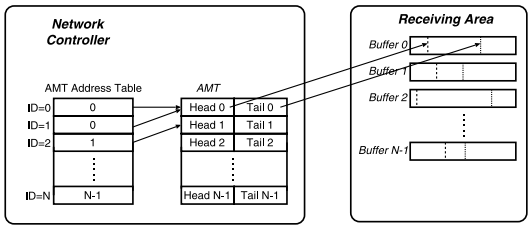


**図 3** AMT Address Table の設定例
Fig. 3 Example for AMT Address Table setting.

ブルはホスト側からも参照可能であり，受信したメッセージを別のバッファにコピーした場合などに Head Pointer の更新を行う．

AMT Address Table をコントローラ内部に搭載することで，送信側がネットワークコントローラに対してメッセージ送信要求を発行する際に，受信先のアドレスを指定する必要がなくなり，要求発行に要するレイテンシが削減されると考えられる．また，AMT Address Table を用いることで受信側においてもメッセージの読み出し後の Head Pointer の更新処理などをネットワークコントローラ上で行うことが容易となる．後述する DIMMnet-2 への実装の際には，AMT Address Table を利用して Head Pointer の更新をネットワークコントローラ側で行うようにしている．しかしながら，AMT Address Table のエントリ数はそのネットワークがサポートする最大プロセス数が大きくなるに従って増加するため，このような場合にはハードウェアコストが増大する．そこで，実装するハードウェアコストによっては AMT Address Table をネットワークコントローラ内部に搭載せずに IPUSH 機構を実現するという選択もありうる．この場合は，送信側で受信側の AMT のアドレスを指定してメッセージを送信する必要がある（**図 2**）．

AMT Address Table のエントリは各プロセス ID に個別に対応している．図 1 では AMT Address0 がプロセス ID=0，AMT Address1 がプロセス ID=1……というように対応する．両テーブルの初期値は並列プロセス起動時にホストから書き込む．その際に AMT Address Table の各エントリに書き込む値を同じ値にすると，AMT から読み出される Pointer 値は複数のプロセスで同一のものになる．このようにすることで，たとえばプロセス ID=0〜9 で同一のリングバッファを，プロセス ID=10〜19 で同一のリングバッファを使用するというような設定が可能である．**図 3** は，ID=0 と 1 のプロセスが同じ AMT のエントリを参照するように設定した例である．このような設定を行うと，ID=0 と 1 のプロセスから受信されたメッセージは同一のリングバッファに格納されるようになる．

上記のテーブルに加え，受信したメッセージを受信した順番に読み出すための手段を提供することも必要である．そこで，パケットごと，メッセージごとに受信ステータスを格納する領域をリングバッファとして主記憶に確保する．このリングバッファの Head Pointer と Tail Pointer はネットワークコントローラ内に保存される．ネットワークコントローラはステータス領域に送信元 ID，メッセージサイズなどの情報を書き込み，Tail Pointer を更新する．ホストはこのリングバッファの Tail Pointer が更新されていることを検知することで，メッセージが受信されていることを検出し，ステータスを読み出すことによってメッセージの読み出しに必要な情報の取得を行う．ホストからはステータスを読み出した後に受信ステータスの Head Pointer を変更する．これは VIA [18] のドアベルと同様の機構である．

メッセージ単位だけでなく，パケット単位でも受信ステータスを格納できるようにしているのは，複数プロセスで共通のバッファを使用するように AMT Address Table を設定した場合に，メッセージが複数パケットに分割されて受信された際に受信バッファを共有しているプロセス間でメッセージがインタリーブされて受信される可能性があるためである．この場合，受信ステータスがメッセージ単位で格納されていると，インタリーブされて受信されたパケットのどこからどこまでが同一プロセスからのメッセージを構成するのかを識別することができなくなり，メッセージを正常に受信することができなくなる．

PM [19] では，複数の通信相手からのメッセージが単一の受信バッファに到着順に格納されるために，IPUSH 機構のように受信側のネットワークコントローラが受信先アドレスを指定できる受信機構をハードウェアで持たない場合，ネットワークコントローラ上のプロセッサを利用したソフトウェア処理を行うことによって受信先アドレスを制御するか，PM/RHiNET の実

装[6] のように，送信元のプロセスごとに独立したバッファを確保する必要がある．前者の場合はオーバヘッドが大きく，後者の場合はスケーラビリティに乏しいうえに，メッセージを到着順に読み出すことができなくなる．そのため，受信ステータスの管理機構を持つIPUSH 機構は PM を実装する際にも有利となると考えられる．

プロセスごとに独立したバッファを確保している場合には，受信メッセージのインタリーブは起きないため，メッセージ単位でステータスを格納しても問題は起こらず，また，格納されるステータスの数が削減されるために，バッファを調べるレイテンシを抑えることが可能となる．さらに，受信したメッセージのマッチングの際にも，各送信プロセスごとにバッファが独立しているために，調べるバッファの範囲を削減することができ，文献 9) で述べられている，アプリケーションによっては unexpected message queue や posted receive queue の探索範囲が大きくなるという問題も解決可能である．

パケット単位で受信ステータスを格納するか，メッセージ単位で格納するかはパケットヘッダに 1 ビットのフラグを用意し，送信側がメッセージ送信要求時にそのフラグを指定することで選択可能である．

### 3.2 メッセージ受信領域の削減

一般に，すべての並列アプリケーションで，あるプロセスがすべてのプロセスと通信を行うというということはなく，一部のプロセスとのみ通信を行うアプリケーションも存在する．たとえば，NAS Parallel Benchmarks の IS，FT では集団通信を主に行うが，CG，MG では一部のプロセスとしか通信を行わない．したがって，AMT Address Table の設定により，通信頻度の高いプロセスに対しては個別のリングバッファを確保し，それ以外のプロセスに対しては IPUSH 機構での受信を行わない，または共通のリングバッファを 1 つ確保し，そこに受信することで，受信領域全体のサイズを抑えることが可能となる．このように IPUSH 機構はアプリケーションに応じた設定をすることで受信領域の使用量を柔軟に変えることができる．

通信頻度の低いプロセスに対して IPUSH 機構を用いない場合，IPUSH 機構を使用するパケットと使用しないパケットの区別を受信側のネットワークコントローラで行う必要があるが，これはそれぞれを別の通信プリミティブとする，IPUSH 機構を用いることを示すフラグを 1 ビット，パケットヘッダに追加するなどの対応により解決可能である．後述の DIMMnet-2 への実装の場合は，パケットヘッダ内のプリミティブの種別を示すフィールドの値を PUSH と IPUSH で別のものにすることで区別している．

**表 1** IPUSH 機構における受信領域削減効果
Table 1 Decrease in the usage of receiving area.

| アプリケーション | 通信プロセス数 | 総受信領域サイズ | 削減率 |
|---|---|---|---|
| MG | 9 | (9+1)×N | 76.6% |
| CG | 4 | (4+1)×N | 92.2% |
| IS | 1 | (1+1)×N | 98.4% |
| LU | 4 | (4+1)×N | 92.2% |
| SP | 6 | (6+1)×N | 89.1% |
| BT | 6 | (6+1)×N | 89.1% |

N：各リングバッファのサイズ

実際に AMT Address Table の設定によってどの程度受信領域を削減できるかを，NAS Parallel Benchmarks のトレースデータをもとに検討する．アプリケーションには Class A の MG，CG，IS，LU，SP，BT を選択し，プロセス数は 64 プロセスとする．

IPUSH 機構を用いない場合には各プロセスに個別のリングバッファを確保するものとし，IPUSH 機構を用いる場合には通信を行うプロセスに対しては個別のリングバッファを確保し，それ以外のプロセスに対しては共通のリングバッファを 1 つ確保することとする．したがって，IPUSH 機構を用いない場合には 64×N Byte の受信領域を，IPUSH 機構を用いる場合には (通信を行うプロセスの数+1)×N Byte の受信領域を確保することになる．N はリングバッファ1 つあたりの容量である．

検討結果を**表 1** に示す．表 1 の各項目はそれぞれアプリケーション，通信相手となるプロセスが最も多いプロセスの通信プロセス数，必要となる受信領域の総サイズ，PUSH と比較した際の使用領域の削減率を示す．

各アプリケーションにおいて，大幅に使用領域を削減可能となっている．したがって，AMT Address Table をアプリケーションに適した設定にすることにより，受信領域の使用量を抑えることが可能と考えられる．

MG は一部のプロセスが 9 プロセスと通信を行うが，多くのプロセスは 6 プロセスと通信を行うため，表 1 に示された値よりも受信領域の使用量は少なくなる．IS では各プロセスの通信を行うプロセス数が 1 となっているが，IS はプログラムの実行時間測定範囲内において MPI_Alltoall や MPI_Reduce 関数による集団通信しか行わないため，通信プロセス数はこれらの関数の実装に依存する．しかしながら，仮にすべてのプロセスと通信を行うような MPI の実装でも，複数のプロセスで 1 個のリングバッファを利用するよう

な設定を行うことで受信領域を削減することは可能である．

一般のアプリケーションにおいても，海洋や大気のシミュレーションのように問題を格子状に区切り，各格子点をプロセスに割り当て，隣接する格子点を割り当てられたプロセスとのみステップごとに通信を行うようなアプリケーションの場合は通信相手が静的に定まるために，本手法を用いてバッファの使用量を抑えることが可能と考えられる．また，同一のアプリケーションをパラメータを変更して複数回にわたり実行する場合には，初回実行時にトレースデータを取得し，その結果を次回以降の AMT Address Table の設定に反映することで本手法を適用することが可能と思われる．

トレースデータの取得が難しい場合には，複数ノードで同一のバッファを共有することで，ある程度はバッファの使用量を抑えることが可能となると考えられるが，この場合はバッファ使用量の削減効果は限定されると思われる．

IPUSH 機構以外のメッセージ受信領域の使用量の削減が可能な手法として，PC クラスタ向けインターコネクトである Maestro2 [20] に実装された Active Zero-copy [21] があげられる．Active Zero-copy は並列データベースサーバなど，通信のタイミングやサイズが不規則なシステムにおいて通信性能を向上させるための手法である．一般に，ゼロコピー通信でメッセージを送受信する場合，メッセージの受信領域はページアウトされないピンダウン領域となっており，通信開始前にあらかじめピンダウンされている．しかし，通信のタイミングやサイズが不規則な状況では，ピンダウンされた領域のサイズが不足する可能性がある．また，サイズの不足を防ぐためにピンダウン領域を大きめに確保すると，ページアウトされないために主記憶を圧迫することになる．そのため，事前に送受信を行うメッセージのサイズを通知しあい，そのつど必要なサイズだけピンダウンを行うことで上記の問題を防ぐことも考えられるが，その場合はメッセージサイズの通知の分だけ性能が低下することになる．

そこで Active Zero-copy では，メッセージがネットワークインタフェースに到着するとネットワークインタフェースがホストに割込みをかけ，ホストが到着したメッセージサイズ分の領域をピンダウンし，そこにメッセージを受信する．受信後，受信側のプロセスがそのピンダウン領域にアクセス可能になるように，受信プロセスのアドレス空間にマッピングを行う．この手法により，メッセージサイズを通知しあう手法よりも高いスループットを達成している．

しかしながら，Active Zero-copy はメッセージが到着するたびにピンダウンとマッピングのためのシステムコールが必要になり，通信のオーバヘッドが大きくなるという欠点が存在する．一方，IPUSH 機構では，並列プロセス起動時に受信領域の設定を AMT や AMT Address Table 行うだけであり，受信のたびにシステムコールを発行することもないためメッセージ受信時のオーバヘッドは低く抑えられる．

## 4. DIMMnet-2 への IPUSH 機構の実装

本論文では IPUSH 機構を DIMMnet-2 試作基板に実装する．IPUSH 機構自体は汎用性があり，他のインターコネクトにも適用可能であるが，DIMMnet-2 試作基板はネットワークコントローラに FPGA を採用しており，ネットワークコントローラの構成の変更が容易であったため，今回，IPUSH 機構の実装対象とした．

### 4.1 DIMMnet-2

DIMMnet-2 は現在我々が開発を行っている，PC クラスタ向けインターコネクトのネットワークインタフェースである．一般に PC クラスタ向けインターコネクトのネットワークインタフェースが PCI-X バスや PCI-Express に接続されるのに対し，DIMMnet-2 はホストの DDR-SDRAM バスに接続される．メモリバスに接続することにより，ホストからネットワークインタフェースへのアクセスレイテンシを低く抑えられ，結果，低レイテンシな通信を実現することが可能となる．DIMMnet-2 ではシステム構築のコスト削減のために，ネットワークスイッチとケーブルに InfiniBand を採用している．

DIMMnet-2 は現在，ネットワークコントローラに FPGA を採用した試作基板を用いて開発が進められている．

#### 4.1.1 DIMMnet-2 試作基板

DIMMnet-2 試作基板の概観を**図 4** に示す．

コントローラ部に Xilinx 社の Virtex-II Pro XC2VP70-7FF1517C を用い，Virtex-II Pro に内蔵されている RocketIO トランシーバを用いることで InfiniBand（4X:10 Gbps）に接続することが可能となっている．また，ノート PC 用の DDR SO-DIMM を 2 枚搭載しており，通信用のバッファとして使用するほか，ホストのデータ記憶領域として使用する．SO-DIMM はホストのアドレス空間上にはマップされず，したがって，直接ホストから SO-DIMM にアクセスすることはできない．このような形態をとることで，ホ

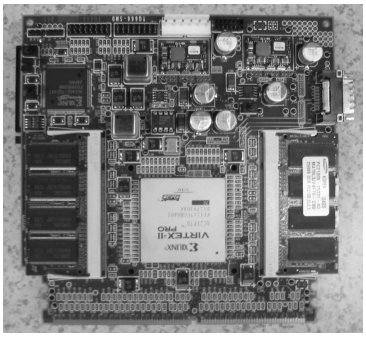

**図 4** DIMMnet-2 試作基板の概観
Fig. 4 Picture of DIMMnet-2 prototype board.

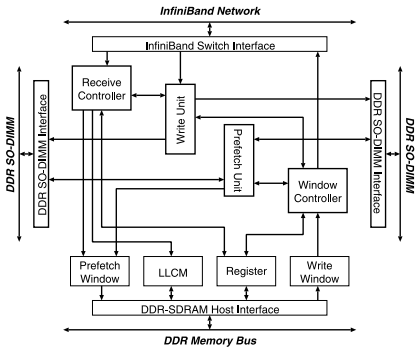


**図 5** ネットワークコントローラのブロック図
Fig. 5 Block diagram of network interface controller.

ストのチップセットによる制約などを受けずに，ホストのメモリスロット 1 本あたりに搭載可能なメモリ容量よりも大きな記憶領域を持つことが可能になり，結果として，ホストに搭載可能な最大メモリ容量よりも大きな記憶領域を提供することが可能となる．FPGA 内の RAM やレジスタをホストのアドレス空間にマップし，RAM に対してデータの書き込み，レジスタに対して要求の発行を行うことで SO-DIMM に対する読み書きや，ネットワークへのパケットの送出を行う．

### 4.2 DIMMnet-2 ネットワークコントローラ

DIMMnet-2 ネットワークコントローラ[22] のブロック図を**図 5** に示す．

各ブロックの機能は以下のとおりである．

- LLCM（Low Latency Common Memory）：ホストからもコントローラからも読み書き可能なメモリ領域
- Prefetch Window：ネットワークインタフェース上の SO-DIMM から読み出したデータを格納する領域
- Register：ネットワークコントローラの設定，ステータス，要求の発行などに使用するレジスタ群
- Write Window：SO-DIMM に格納するデータを書き込む領域
- Window Controller：ホストからの各種要求やパケット送信処理部
- Receive Controller：ネットワークから受信したパケットの処理部
- Prefetch UnitCOLONSO-DIMM のデータ読み出し制御部
- Write Unit：SO-DIMM へのデータ書き込み制御部

これらのブロックのうち，LLCM，Prefetch Window，Register，Write Window の 4 つのブロックがホストのアドレス空間にマップされ，ホストから直接アクセス可能である．

LLCM や Register はホストからもネットワークコントローラからも値の変更が行われることから，ホスト CPU のキャッシュにキャッシュ可能な領域に設定されていると，ホストからの値の変更がキャッシュにのみ格納され，ネットワークコントローラに反映されなくなり，また，ネットワークコントローラからの値の変更を認識できなくなるという問題が発生する．そのため，LLCM や Register はキャッシュされない領域に設定している．

Prefetch Window もネットワークコントローラから値が変更される領域であるが，ここはキャッシュ可能な領域に設定している．DIMMnet-2 ではネットワークインタフェース上の SO-DIMM にホスト CPU から直接アクセスできない構造になっているために，SO-DIMM のデータを読み出す際に，一度 Prefetch Window に対象データを読み出してからそのデータに対してホスト CPU がアクセスするという手順を踏む．そのため，Prefetch Window がホスト CPU にキャッシュされる領域として設定されていると，SO-DIMM から読み出した値をホスト CPU が読み出す場合に，そのデータがキャッシングされてしまい，新たなデータを SO-DIMM から Prefetch Window に読み出した場合に，そのデータに対してアクセスしようとしてもキャッシュのデータを参照してしまうため，新たに SO-DIMM から読み出したデータを参照することができなくなる．したがって，本来ならば，Prefetch Window はキャッシュされない領域にした方が確実に SO-DIMM のデータを読み出すことができるが，その場合，Prefetch Window からのデータの読み出しバンド幅が著しく低下する．そのため，DIMMnet-2 では Prefetch Window をキャッシュ可能な領域に設定し，新たに SO-DIMM のデータを読み出す場合には，

**表 2** PUSH パケット（24 Byte）の受信処理の詳細
Table 2 Detail of transaction of PUSH (24 Byte packet).

| 処理内容 | クロック数 |
|---|---|
| (RC) SWIF にパケットが到着したことを検出 | 0 |
| (RC) SWIF へパケット（ヘッダ部）の読み出し要求 | +1 |
| (RC) 1st Header 受信 | +1 |
| (RC) 2nd Header 受信 | +1 |
| (RC) 1st Header を解析し，Write Unit を起動 | +1 |
| (WU) SO-DIMM I/F に渡すサイズ，アドレスの計算 | +2 |
| (WU) SWIF へパケット（データ部）の読み出し要求 | +1 |
| (WU) SWIF からデータ部の受信 | +1 |
| (WU) SO-DIMM へのデータ書き込み権要求 | +1 |
| (WU) SO-DIMM へのデータ書き込み権取得 | +1 |
| (WU) SO-DIMM へのデータの書き込み | +6 |
| (RC) パケット受信ステータスの書き込み | +4 |
| 合計 | 20 |

Prefetch Window がキャッシュされているキャッシュラインをフラッシュしてからプリフェッチ命令でデータをキャッシュに格納することで，高い読み出しバンド幅を得る方式をとっている．

#### 4.2.1 PUSH 時のパケットの受信処理

PUSH によるパケットを SO-DIMM に受信する処理の流れは以下のとおりである．

(1) InfiniBand Switch Interface（SWIF）がパケットを受信したことを Receive Controller に通知する．
(2) 通知を受けて，Receive Controller がパケットのヘッダ部分を受け取り，解析する．
(3) Receive Controller がパケットのデータ部の受信先，サイズなどを Write Unit に通知する．
(4) Write Unit が SWIF からパケットのデータ部を受け取り，SO-DIMM に書き込む．
(5) SO-DIMM への書き込み終了後，Receive Controller が LLCM にパケット受信ステータスを書き込む．

**表 2** に示すように，DIMMnet-2 において，この処理に要するクロック数は 20 クロックであり，DIMMnet-2 のネットワークコントローラは 100 MHz で動作するため，0.2 $\mu$s の処理時間となる．表 2 の (RC)，(WU) はそれぞれ，Receive Controller，Write Unit の処理であることを示す．

### 4.3 IPUSH 機構の DIMMnet-2 への実装

IPUSH 機構は受信処理を行うブロックである Receive Controller，Write Unit，LLCM に対して変更を加えることで実現する．AMT はホストからもコントローラからも値を更新するため，LLCM 上に設ける．一方で，AMT Address Table は一度ホストから設定が完了すれば，そのプロセスの起動中は書き換えられる頻度は低いと考えられるため，コントローラ内部に設け，ホストからは Register を介して設定する．DIMMnet-2 ネットワークコントローラはパケット受信ステータスを扱う機能を持っており，これをそのまま IPUSH 機構のメッセージ受信ステータスに使用する．受信ステータスの格納先は LLCM 上に確保され，受信ステータスの Head Pointer と Tail Pointer は Register 上に確保される．これらの初期値設定はプロセス起動時に行う．DIMMnet-2 ではメッセージの主な受信先は SO-DIMM であるため，各プロセスに対するメッセージ受信用のリングバッファは SO-DIMM 上に確保する．

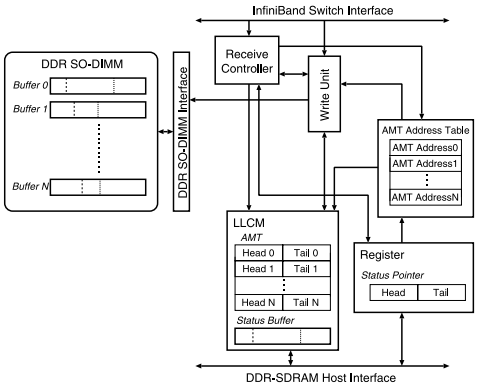


**図 6** IPUSH 機構の実装
Fig. 6 Implementation of IPUSH mechanism.

これらのブロック間の接続図を**図 6** に示す．

IPUSH 機構によりパケットを受信する際の処理の流れを以下に示す．

(1) SWIF がパケットを受信したことを Receive Controller に通知する．
(2) 通知を受けて，Receive Controller がパケットのヘッダ部分を受け取る．
(3) AMT Address Table に対し，送信元のプロセス ID でアクセスする．
(4) パケットヘッダを解析し，Write Unit に転送データサイズなどを通知する．
(5)
- AMT Address Table の出力が LLCM と Write Unit に転送される．
- LLCM は AMT Address Table の出力に対応した AMT のエントリを Write Unit に転送する．

(6)
- Write Unit が AMT からの出力をもとに SO-DIMM に対してデータを書き込む．
- Head Pointer と Tail Pointer から受信領

**表 3** IPUSH パケット（24 Byte）の受信処理の詳細
Table 3 Detail of transaction of IPUSH (24 Byte packet).

| 処理内容 | クロック数 |
|---|---|
| (RC) SWIF にパケットが到着したことを検出 | 0 |
| (RC) SWIF へパケット（ヘッダ部）の読み出し要求 | +1 |
| (RC) 1st Header 受信 & AMT Address Table にアクセス | +1 |
| (RC) 2nd Header 受信 | +1 |
| (RC) 1st Header を解析し，Write Unit を起動 | +1 |
| (WU) LLCM から Head と Tail を取得 | +1 |
| (WU) 受信領域の始端を計算 | +1 |
| (WU) 受信領域の終端を計算 | +1 |
| (WU) 受信領域に空きがあるかどうか判定 | +1 |
| (WU) SO-DIMM I/F に渡すサイズ，アドレスの計算 | +2 |
| (WU) SWIF へパケット（データ部）の読み出し要求 | +1 |
| (WU) SWIF からデータ部の受信 | +1 |
| (WU) SO-DIMM へのデータ書き込み権要求 | +1 |
| (WU) SO-DIMM へのデータ書き込み権取得 | +1 |
| (WU) SO-DIMM へのデータの書き込み | +6 |
| (RC) パケット受信ステータスの書き込み | +4 |
| 合計 | 24 |

域に空きがないと検知された場合は，空きができるまで待機状態となる．

(7) SO-DIMM への書き込み終了後，Write Unit は AMT に対して Tail Pointer の更新を行う．

(8) Receive Controller が LLCM に対してパケット受信ステータスを書き込み，Status Pointer の Tail Pointer を更新する．

**表 3** に示すように，DIMMnet-2 において，この処理に要するクロック数は 24 クロックとなり，PUSH よりも 4 クロックの増加となっている．これは 0.24 $\mu$s の処理時間となり，0.04 $\mu$s の処理時間増加となる．表 3 の (RC), (WU) はそれぞれ，Receive Controller，Write Unit の処理であることを示す．

この通信では，受信側のホストは都合の良いときにパケット受信ステータスの領域を読み出すことでパケットの受信を検知する．しかしながら，実際に受信側の都合の良いときのみにメッセージの読み出しを行った場合，メッセージの受信領域が足りない状況が起こりうる．特に PCI-X バスなどに装着される一般的なネットワークインタフェースにおいて，メッセージの受信領域を主記憶上に確保する場合，受信領域は主記憶に常駐する，ページアウトされないピンダウン領域であるため，受信領域を大きく確保すると主記憶を圧迫することになる．そのため，圧迫を防ぐために受信領域を小さく確保することになるが，この場合，受信領域の不足はより高い確率で起こりうるうえに，ホストからメッセージの受信検知のためのポーリングを頻繁に行う必要が出てくるため，ホストのオーバヘッドが増加する．したがって，オーバヘッドを抑えつつ，IPUSH 機構によるメッセージの送受信を実現するには，受信領域内の各リングバッファのサイズを大きめに確保し，AMT Address Table の設定により複数のプロセスで 1 つのリングバッファを共有するといった設定を行う必要があると考えられる．

一方，DIMMnet-2 においては，他の PC クラスタ向けネットワークインタフェースと異なり，メッセージの受信領域がネットワークインタフェース上の SO-DIMM であるため，この問題は起こりにくくなる．この SO-DIMM 領域はホストのアドレス空間上にマップされていないため，ホストのチップセットなどに制限されずに大容量の SO-DIMM を搭載することが可能である．仮に 1 GByte の SO-DIMM をこの試作基板上に搭載した場合，SO-DIMM の総容量は 2 GByte である．一方で，汎用バスに接続される PC クラスタ向けネットワークインタフェースに搭載されているメモリ容量は，QsNET II のネットワークインタフェースである QM500 PCI-X Network Adapter で 64 MByte[11]，Myrinet/PCI-X E Card で 2 MByte または 4 MByte[15]，Mellanox 社の InfiniHost III Ex HCA Cards シリーズで最大 128 MByte[16] である．DIMMnet-2 のアーキテクチャはネットワークインタフェース上のメモリ領域に受信バッファを確保することで受信バッファサイズを大きくできるという点で IPUSH 機構によるメッセージの受信に適しているといえる．

## 5. 評　　価

本章では DIMMnet-2 に実装した IPUSH 機構の評価を示す．本評価では IPUSH 機構のハードウェアコストと IPUSH 機構でパケットを受信した際に，PUSH に比べ，どの程度性能低下が起こるかを調べる．

### 5.1 ハードウェア量

**表 4** に IPUSH 機構を追加する前と後のネットワークコントローラのハードウェア量を示す．

IPUSH 機構を追加したことによりコントローラ内部で使用する RAM 領域が増加し，結果として Block RAM の使用量が増加している．これは AMT Address Table に DIMMnet-2 試作基板でサポートする最大プロセス数分のエントリを確保したためであるが，ネットワークでサポートする最大プロセス数が増加すると AMT Address Table のエントリ数も増加する．そのため，場合によっては AMT Address Table を送信側の主記憶に設け，送信側で受信側の AMT のエントリの位置を指定するように変更することでコントローラ

**表 4** IPUSH area サイズ
Table 4 IPUSH area size.

| | Before | After | Diff |
|---|---|---|---|
| Block RAM | 149/328 (45%) | 155/328 (47%) | +6 |
| Slices | 15792/33088 (47%) | 15913/33088 (48%) | +121 |
| Clock | 95.649 MHz | 95.662 MHz | +0.013 |

合成ツール：Xilinx ISE 7.1i SP4
デバイス：XC2VP70-7FF1517C

内部に搭載する RAM 領域を削減する必要があると考えられる．

一方，スライス数の増加は全体の 1%程度であり，これは主に AMT から読み出した Head Pointer と Tail Pointer からメッセージ受信領域の空き領域を計算するための論理や，AMT などとの接続部であるため，ネットワークのサポートする最大プロセス数が増加してもそれほどハードウェア量が増加するとは考えにくい．そのため，IPUSH 機構を低いハードウェアコストで実装可能であると考えられる．

動作周波数は ISE の Synthesize 完了後の値である．IPUSH 機構実装後の方が動作周波数が向上しているが，わずかなものであり，IPUSH 機構がコントローラの動作周波数に大きな影響を与える可能性は低いと考えられる．なお，動作周波数が 100 MHz を下回っているが，配置配線後は制約を満たすために 100 MHz での動作が可能である．

### 5.2 IPUSH と PUSH のレイテンシの比較

DIMMnet-2 を搭載した 2 台の PC を InfiniBand スイッチに接続し，IPUSH 機構を用いた場合と用いない場合でそれぞれ ping-pong 転送を行った際のバンド幅とレイテンシを比較する．送信側の処理はどちらもまったく同じであり，受信処理のみ異なるため，それぞれの機構の通信性能を比較することが可能となる．

ping-pong 転送においては，**表 5** のように条件を変更して 5 通りの測定を行った．

(1) と (2) の条件で PUSH と IPUSH でメッセージを受信した際の性能差を比較する．これは，表 2，表 3 で示される以外のレイテンシの差が発生しないことを調べるためのものである．続いて，(3) と (4) の条件で受信したメッセージをホストにコピーするまでのバンド幅を測定し，ホストから AMT の内容を更新するオーバヘッドを含めた性能差を比較する．(5) の条件は (4) の条件を，Prefetch Window のメモリ属性をホスト CPU にキャッシュされない Uncachable 領域にしたものである．(5) の条件はキャッシュのフラッシュとプリフェッチ命令が性能にどの程度効果があるかを (4) の条件と比較するためのものである．

**表 5** 測定条件
Table 5 Evaluation condition.

| 条件 | データの格納先 | メッセージ受信機構 | Prefetch Window |
|---|---|---|---|
| (1) | SO-DIMM | PUSH | ———— |
| (2) | | IPUSH | ———— |
| (3) | 主記憶 | PUSH | Write Back |
| (4) | | IPUSH | |
| (5) | | | Uncachable |

**表 6** 測定環境
Table 6 Evaluation environment.

| | | |
|---|---|---|
| Node | CPU | Pentium4 2.6 GHz (2625.918 MHz)<br>L2 Cache 512 KByte |
| | Chipset | VIA VT8751A |
| | Memory | PC-1600 DDR-SDRAM<br>512 MByte ×1 DIMMnet-2 ×1 |
| | OS | RedHat8.0 (Kernel 2.4.27) |
| Network | InfiniBand Switch | Voltaire ISR6000 |
| | Switch Module | SW-6IB4C |
| | cable | 2 m |
| Compiler | gcc | 3.3.5 (compile option: -Wall) |

評価環境を**表 6** に示す．

ping-pong 転送の処理内容を以下に示す．

(1) ping-pong 開始側（CPU A）が Command Register にパケット送信の要求を発行する（**図 7** A)）．
(2) 要求を受けて CPU A 側のネットワークコントローラが SO-DIMM のデータをパケットに加工し，送出する（図 7 B)）．
(3) 送出されたパケットが InfiniBand Switch を通過する（図 7 C)）．
(4) CPU B 側のネットワークコントローラがパケットを受信し，SO-DIMM にデータを受信する（図 7 D)）．
(5) CPU B 側のネットワークコントローラがパケット受信ステータスを LLCM に書き込み，Status Pointer の値を変更する（図 7 E)）．
(6) CPU B が Status Pointer のポーリングによって，パケットの受信を検知する（図 7 F)）．
(7) パケット受信ステータスを LLCM から読み出す（図 7 G)）．
(8) 受信したデータを SO-DIMM から Prefetch Window に読み出す要求を発行する（図 7 H)）．
(9) SO-DIMM から Prefetch Window にデータが読み出される（図 7 I)）．
(10) Prefetch Window へのデータの読み出しが完了したことを検知する（図 7 J)）．

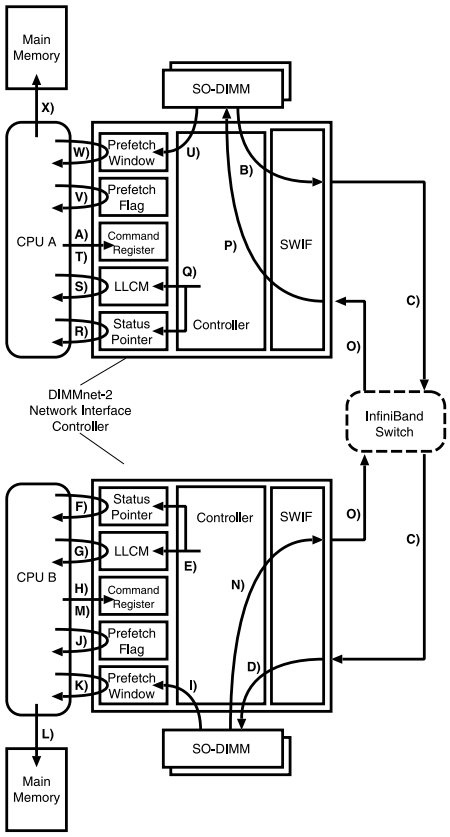


**図 7** ping-pong 転送
Fig. 7 Ping-pong communication.

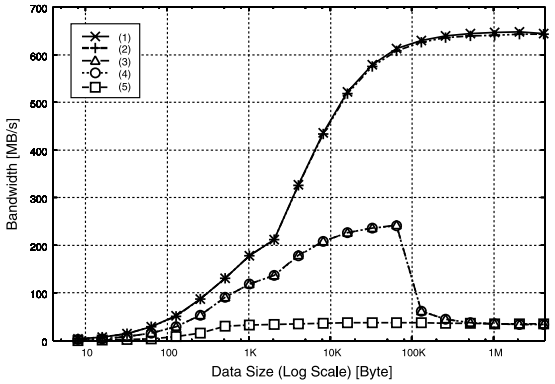


**図 8** ping-pong 転送時のバンド幅
Fig. 8 Bandwidth of ping-pong communication.

(11) CPU B が Prefetch Window のデータを読み出す（図 7 K)).
(12) Prefetch Window から読み出したデータを主記憶にコピーする（図 7 L)).
(13) CPU B が受信したパケットと同サイズのパケットを，送出する要求を発行する（図 7 M)).
(14) 要求を受けて CPU B 側のネットワークコントローラが SO-DIMM のデータをパケットに加工し，送出する（図 7 N)).
(15) 送出されたパケットが InfiniBand Switch を通過する（図 7 O)).
(16) CPU A 側のネットワークコントローラがパケットを受信し，SO-DIMM にデータを受信する（図 7 P)).
(17) CPU A 側のネットワークコントローラがパケット受信ステータスを LLCM に書き込み，Status Pointer の値を変更する（図 7 Q)).
(18) CPU A が Status Pointer のポーリングによって，パケットの受信を検知する（図 7 R)).
(19) パケット受信ステータスを LLCM から読み出す（図 7 S)).
(20) 受信したデータを SO-DIMM から Prefetch Window に読み出す要求を発行する（図 7 T)).
(21) SO-DIMM から Prefetch Window にデータが読み出される（図 7 U)).
(22) Prefetch Window へのデータの読み出しが完了したことを検知する（図 7 V)).
(23) CPU A が Prefetch Window のデータを読み出す（図 7 W)).
(24) Prefetch Window から読み出したデータを主記憶にコピーする（図 7 X)).

これらの処理内容のうち，条件 (1)，(2) では G)〜L)，S)〜X) は実行対象外となる．

測定結果を**図 8** に示す．SO-DIMM 間で ping-pong 転送した際の最大バンド幅は PUSH で約 644 MByte/s であり，IPUSH で約 643 MByte/s であった．最小の RTT(Round Trip Time)/2 はヘッダ 16 Byte，データ 8 Byte の計 24 Byte 転送時で PUSH で約 1.74 $\mu$s であり，IPUSH で約 1.78 $\mu$s であった．また，ホストからのデータの読み出しを含めた際の最大バンド幅は PUSH で約 242 MByte/s，IPUSH で 241 MByte/s，最小の RTT/2 は PUSH で 3.84 $\mu$s，IPUSH で 3.89 $\mu$s となった．これらの結果より，IPUSH 機構は PUSH と同等の性能を持つといえる．

条件 3)，4) ではメッセージサイズが 64 KByte より大きくなるとバンド幅が急激に低下している．これは SO-DIMM から読み出すサイズが大きいために，キャッシュラインのフラッシュとプリフェッチが間に合わなくなり，ホスト CPU のキャッシュにヒットしなくなったためと考えられる．条件 4) と 5) でメッセージサイズが大きい場合のバンド幅がほぼ同じ値であることから，キャッシュへのヒットが大きく影響していることが分かる．元々，DIMMnet-2 では，Prefetch

**表 7** ping-pong 転送時のレイテンシの内訳（header: 16 Byte, data: 8 Byte）
Table 7 Breakdown of ping-pong transaction (header: 16 Byte, data: 8 Byte).

| 処理内容 | レイテンシ [$\mu$s] | |
|---|---|---|
| | RDMA | IPUSH |
| A) パケット送信要求の発行 | 0.093 | |
| B) パケットの送出処理 | 0.428 | |
| C) ネットワーク通過遅延 | 0.385+$\alpha$ | |
| D) パケット受信処理 | 0.472 | 0.512 |
| E) ステータス書き込み | 0.040 | |
| F) パケットの受信検知 | 0.095〜0.284 | |
| G) パケット受信ステータスの読み出し | 0.312 | |
| H) 受信したデータの読み出し要求の発行 | 0.101 | |
| I) SO-DIMM から Prefetch Window への読み出し | 0.240 | 0.250 |
| J) Prefetch Window への読み出し完了検知 | 0.095〜0.284 | |
| K), L) Prefetch Window のデータを主記憶にコピー | 0.605 | |
| M) パケット送信要求の発行 | 0.093 | |
| N) パケットの送出処理 | 0.428 | |
| O) ネットワーク通過遅延 | 0.385+$\alpha$ | |
| P) パケット受信処理 | 0.472 | 0.512 |
| Q) ステータス書き込み | 0.040 | |
| R) パケットの受信検知 | 0.095〜0.284 | |
| S) パケット受信ステータスの読み出し | 0.312 | |
| T) 受信したデータの読み出し要求の発行 | 0.101 | |
| U) SO-DIMM から Prefetch Window への読み出し | 0.240 | 0.250 |
| V) Prefetch Window への読み出し完了検知 | 0.095〜0.284 | |
| W), X) Prefetch Window のデータを主記憶にコピー | 0.605 | |
| 実測値（条件 (1), (2)）RTT | 3.48 | 3.56 |
| RTT/2 | 1.74 | 1.78 |
| 実測値（条件 (3), (4)）RTT | 7.68 | 7.78 |
| RTT/2 | 3.84 | 3.89 |

$\alpha$：InfiniBand ケーブルなどの遅延

Window に対するプリフェッチと，ホスト CPU 上での演算をオーバラップさせることで高い性能を得ることができる．今回，プリフェッチ命令として用いている PREFETCHNTA 命令はメモリアクセスがない間にプリフェッチを行う命令であるため，ping-pong 転送のようなフラグチェックなどメモリアクセスしか行わないベンチマークではデータサイズが大きくなるとプリフェッチが効果的に機能しないと考えられる．

24 Byte 転送時のレイテンシの内訳を**表 7** に示す．表 7 に示されるレイテンシのうち，B)，D)，E)，I)，N)，P)，Q)，U) は RTL シミュレーションにより得た値である．IPUSH 機構によりパケットを受信した際のクロック数は表 3 より 24 クロックであり，PUSH によるパケットの受信処理に要するクロック数は表 2 より 20 クロックである．4 章で述べたとおり，DIMMnet-2 のネットワークコントローラは 100 MHz で動作するため，受信処理のレイテンシの増加は 0.04 $\mu$s となる．また，I)，U) の処理において，IPUSH 機構を用いた場合の方が 0.01 $\mu$s だけレイテンシが増加しているが，これは SO-DIMM のデータを読み出す処理の際に，読み出し完了後に AMT の Head Pointer 更新をネットワークコントローラが行うようにしたためである．表 7 の A) や H) に示されるように，キャッシュされない領域へのアクセスレイテンシが比較的大きく，性能に影響を与えることが予想されたために，このような方式をとった．

C)，O) の値は文献 23) から得たものであり，その他の値については実機を用いた測定により得た．F)，J)，R)，V) の値はホスト CPU から DIMMnet-2 のホストにマップされている領域に対するポーリングとなるため，値は一定ではない．PC-1600 DDR-SDRAM モジュールに対するアクセスレイテンシの測定を行った結果，約 0.189 $\mu$s であった．したがって，Status Pointer のポーリングのレイテンシが最大になるのは，ホストから Status Pointer の値を読み出した直後にフラグが変化する場合であり，この場合でアクセスレイテンシの約 1.5 倍，レイテンシが最小になるのが，読み出し開始直前に Status Pointer の値が変化する場合であり，この場合に約 0.5 倍になると仮定した．実機での測定では多数回実行し，その平均をとっているために，ポーリングレイテンシの変動はさほど影響なく，PUSH と IPUSH で有意な差は見られなかった．

条件 1)，2) 間で RTL シミュレーションから得られた PUSH と IPUSH の受信処理のレイテンシの差と同等の性能差が出ており，条件 1)，2) ではホストの影響が少ないことから，条件 1)，2) 間のレイテンシの差は受信処理のレイテンシの増加がそのまま影響したものと考えられる．また，条件 3)，4) 間では 0.05 $\mu$s と差がわずかに大きくなっているが，これらはすべてネットワークコントローラ内部での処理内容の差に由来するものであり，PUSH も IPUSH もホスト側で行う処理によるレイテンシには差が見られないと考えられる．そのため，ホストの処理を含めても IPUSH 機構を付加したことによる性能低下はわずかであるといえる．

条件 3)，4) の場合の RTT の値は，表 7 の合計値よりも大きな値となっているが，表 7 の各項目の測定と ping-pong 全体の測定では rdtsc 命令など測定用の命令を挿入する箇所が変わってくることから，ホストの挙動が変化するため，キャッシュラインのフラッシュやプリフェッチのタイミングが変わってしまうなどのホスト側の不確定な要素があるため，原因を探るのは難しいと思われる．

## 6. まとめと今後の課題

本論文では，受信側のネットワークコントローラがメッセージの受信先を指定し，メッセージの受信，および，受信領域のアドレス管理を行う IPUSH 機構を提案した．IPUSH 機構を DIMMnet-2 試作基板に対し実装し，通信性能の評価を行った．その結果，IPUSH 機構を用いた場合の ping-pong 転送での最大バンド幅がネットワークインタフェース上の SO-DIMM 間で約 643 MByte/s であり，最小のレイテンシが 24 Byte 転送時で約 1.78 $\mu$s と，従来の RDMA と同等の性能を持つことが示された．また，SO-DIMM に受信したメッセージをホストから読み出す処理を加えても，従来の RDMA とほぼ同等の性能を持つ．さらに，受信領域の使用量の削減についても検討を行い，多くのアプリケーションで高い削減率を達成できる可能性があることが示された．

IPUSH 機構の課題として，MPI などの上位のレイヤからどのようにして AMT Address Table の最適な設定を行うかを検討する必要がある．また，Myrinet や QsNET で行われているマッチングの処理を IPUSH 機構に追加することについても検討する価値があると思われる．

**謝辞** 本研究は総務省戦略的情報通信研究開発推進制度（SCOPE）の一環として行われたものである．DIMMnet-2 の開発に関する議論，開発にご参加いただいている（株）日立 IT の今城氏，岩田氏，上嶋氏，東京農工大学の濱田氏，荒木氏，慶應義塾大学の西助手，渡邊氏，大塚氏，伊澤氏，宮代氏に感謝いたします．

## 参 考 文 献

1) Boden, N.J., Cohen, D., Felderman, R.E., Kulawik, A.E., Seitz, C.L., Seizovic, J.N. and Su, W-K.: Myrinet — A gigabit per second local area network, *IEEE Micro*, Vol.15, No.1, pp.29–36 (1995).
2) Petrini, F., Fang, W-C., Hoisie, A., Coll, S. and Frachtenberg, E.: The Quadrics Network: High-Performance Clustering Technology, *IEEE Micro*, Vol.22, No.1, pp.46–57 (2002).
3) InfiniBand Trade Association. http://www.infiniba-ndta.org/
4) 田邊 昇，山本淳二，工藤知宏：メモリスロットに搭載されるネットワークインタフェース MEMnet，情報処理学会アーキテクチャ研究会，Vol.ARC-134-13 (SWoPP'99)，pp.73–78 (1999).
5) 田邊 昇，濱田芳博，三橋彰浩，山本淳二，今城英樹，中條拓伯，工藤知宏，天野英晴：DIMMnet-1 における Martini オンチッププロセッサによる通信の性能評価，情報処理学会アーキテクチャ研究会，Vol.ARC-150-11 (DesignGaia2002), pp.53–58 (2002).
6) 大塚智宏，渡邊幸之介，北村 聡，原田 浩，山本淳二，西 宏章，工藤知宏，天野英晴：分散並列処理用ネットワーク RHiNET-2 の性能評価，先進的計算基盤システムシンポジウム SACSIS2003, pp.45–52 (2003).
7) 渡邊幸之介，大塚智宏，天野英晴：ネットワークインタフェース用コントローラチップ Martini における乗っ取り機構の実装と評価，情報処理学会論文誌，Vol.45, No.SIG 11, pp.393–407 (2004).
8) 山本淳二，渡邊幸之介，土屋潤一郎，原田 浩，今城英樹，寺川博昭，西 宏章，田邊 昇，上嶋利明，工藤知宏，天野英晴：高性能計算をサポートするネットワークインタフェース用コントローラチップ Martini，情報処理学会論文誌，Vol.43, No.SIG 6, pp.122–133 (2002).
9) Brightwell, R. and Underwood, K.D.: An Analysis of NIC Resource Usage for Offloading MPI, *Proc. 18th International Parallel and Distributed Processing Symposium* (*IPDPS'04*) (2004).
10) Myricom: Myrinet Express (MX): A High-Performace, Low-Level, Message-Passing Interface for Myrinet Version 1.1 (2006). Available from http://www.myri.com/scs/MX/doc/mx.pdf
11) Quadrics. http://www.quadrics.com/
12) MPICH Home Page. http://www-unix.mcs.anl.gov/mpi/mpich/
13) Underwood, K.D., Hemmert, K.S., Rodrigues, A., Murphy, R. and Brightwell, R.: A Hardware Acceleration Unit for MPI Queue Processing, *Proc. 19th IEEE International Parallel and Distributed Processing Symposium* (*IPDPS'05*) (2005).
14) Dickman, L.: Beyond Hero Numbers: Factors Affecting Interconnect Performance, *PathScale White Paper* (2005).
15) Myricom. http://www.myri.com/
16) Mellanox. http://www.mellanox.com/
17) PathScale. http://www.pathscale.com/
18) Intel: Intel Virtual Interface (VI) Architecture Developer's Guide Reveision 1.0 (1998). Available from ftp://download.intel.com/design/servers/vi/intel.pdf
19) Takahashi, T., Sumimoto, S., Hori, A., Harada, H. and Ishikawa, Y.: PM2: High Performance Communication Middleware for

Heterogeneous Network Environment, *SC2000*, pp.52–53 (2000).
20) Aoki, K., Yamagiwa, S., Ferreira, K., Campos, L.M., Ono, M., Wada, K. and Sousa, L.: Maestro2: High Speed Network Technology for High Perfromace Computing, *Proc. IEEE International Conference on Communications*, pp.1033–1037 (2004).
21) Yamagiwa, S., Aoki, K. and Wada, K.: Active Zero-copy: A performance study of nondeterministic messaging, *Proc. 5th International Symposium on Parallel and Distributed Computing* (2005).
22) 北村 聡, 濱田芳博, 宮部保雄, 伊澤 徹, 宮代具隆, 田邊 昇, 中條拓伯, 天野英晴：DIMMnet-2 ネットワークインタフェースコントローラの設計と実装, 情報処理学会論文誌, Vol.46, No.SIG 12, pp.13–26 (2005).
23) 濱田芳博, 荒木健志, 西 宏章, 田邊 昇, 天野英晴, 中條拓伯：bDais: DIMMnet-1/InfiniBand 間ルータの評価, 情報処理学会アーキテクチャ研究会, Vol.2004-ARC-159 (SWoPP'04), pp.145–150 (2004).

(平成 18 年 1 月 27 日受付)
(平成 18 年 5 月 9 日採録)

**北村 聡**（学生会員）

2005 年慶應義塾大学大学院理工学研究科開放環境科学専攻前期博士課程修了．現在，同後期博士課程に在学．計算機アーキテクチャに関する研究に従事．

**宮部 保雄**

2005 年慶應義塾大学理工学部情報工学科卒業．現在，同大学大学院理工学研究科開放環境科学専攻前期博士課程に在学．計算機アーキテクチャに関する研究に従事．

**中條 拓伯**（正会員）

1961 年生．1985 年神戸大学工学部電気工学科卒業．1987 年同大学大学院工学研究科修了．1989 年神戸大学工学部助手の後，1998 年より 1 年間 Illinois 大学 Urbana-Champaign 校 Center for Supercomputing Research and Development（CSRD）にて Visiting Research Assistant Professor を経て，現在東京農工大学大学院共生科学技術研究部助教授．プロセッサアーキテクチャ，並列処理，クラスタコンピューティング，高速ネットワークインタフェースに関する研究に従事．電子情報通信学会，IEEE CS 各会員．博士（工学）．

**田邊 昇**（正会員）

1985 年横浜国立大学工学部卒業．1987 年同大学大学院工学研究科修了．同年（株）東芝に入社．1998 年より 2001 年まで新情報処理開発機構つくば研究センターに出向．並列処理，超並列計算機，ベクトルプロセッサ，PC クラスタ向けネットワークインタフェース，メモリアーキテクチャに関する研究に従事．現在,（株）東芝・研究開発センター勤務．工学博士．2005 年度情報処理学会山下記念研究賞受賞．電子情報通信学会会員．

**天野 英晴**（正会員）

1986 年慶應義塾大学大学院理工学研究科電気工学専攻博士課程修了．現在，慶應義塾大学理工学部情報工学科教授．工学博士．計算機アーキテクチャの研究に従事．