社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－ARC－165（8）
2005／12／1

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実装

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, Message Passing が用いられる. Message Passing において各プロセスはメッセージを明示的に送信, 及び受信することにより並列処理を実現する. Message Passing 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 を用いる手法が挙げられる. 本報告では, RDMA を用いて低レイテンシな Message Passing 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, ハードウェアコスト, パケット受信処理に要するクロック数を測定した. 測定の結果, 数%のハードウェア量, 40ns 程度のレイテンシの増加で IPUSH 機構を実装可能であることが示された.

Implementation of Packet Receiving Mechanism Supporting for Message Passing Model

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 mechanism can be implemented with a small hardware cost. The latency is increased only 40ns compared with 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, Message Passing が用いられる. Message Passing において各プロセスはメッセージを明示的に送信, 及び受信を行うことで並列処理を実現する. Message Passing 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 Remote Direct Memory Access(RDMA) を用いる手法が挙げられる.

RDMA は受信先のアドレスを事前に通知し合う等の手段によって, 送信側が受信先のアドレスを知る必要があるものの, 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.

一方, 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受信する等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 より大きくなる.

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 Message Passing 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得る. 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, 本報告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, 及び受信領域のアドレス管理を行う手法を提案する. 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

† 慶應義塾大学
Keio University
†† 東京農工大学
Tokyo University of Agriculture and Technology
††† (株) 東芝, 研究開発センター
Corporate Research and Development Center,Toshiba

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, Message Passing を実現するのが容易となる. また, この通信に関する全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 上に実装し, ハードウェアコスト, パケットの受信に要するクロック数を評価する. 以降, 本報告ではこの機構を Indirect PUSH(IPUSH) と呼称する.

以下, 本報告では 2 章で IPUSH 機構と同様の通信機構の実装について触れ, 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

IPUSH 機構の概念は DIMMnet-2 の前身である DIMMnet-1 の構想の中で提唱されている. ここでは, DIMMNet-1 のリモート間接書込み[4)5)], 及び RHiNET-2[6)] の VPUSH[7)] について触れる. 共に, 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 IPUSH 機構と同様の機能を提供している.

DIMMnet-1 のリモート間接書込みでは, 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 のパケットの処理等を全て 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 の最大バンド幅の 7%程度の性能であった. 処理の一部をハードワイヤード部で実行した場合でも 16%程度の性能しか出ず, RDMA と比較すると大幅に性能が低下してしまっている.

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, 及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.

RISC プロセッサの性能が低いということもあるが, これらは共にソフトウェアで通信処理を行っていることも原因と考えられる. 同様の傾向は他の PC クラスタ向けインターコネクトでも見られ, Myrinet/PCI-X E-Card[9)] の MPI レベルでの最小の送信レイテンシは約 3.5μs であり, 対して, QsNET II では 1.7μs である[10)]. 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

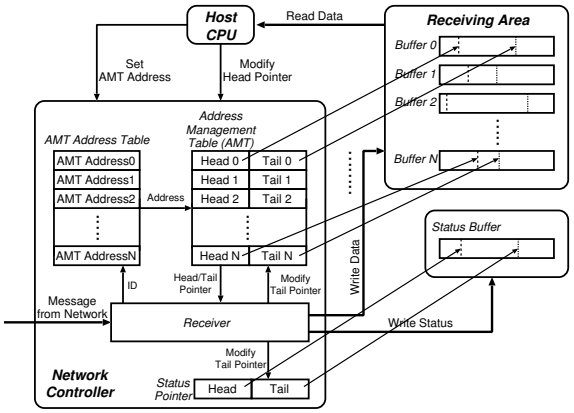

図 1 IPUSH 機構

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. InfiniBand でもベンダによって性能差が見られており, Mellanox 社[11)] の InfiniHost III Ex HCA Card と PathScale 社[12)] の InfiniPath でも同様の傾向が見られることから, InfiniHost III Ex HCA Card はネットワークインタフェース上の通信処理にソフトウェアが介在していると考えられる.

3. IPUSH 機構

IPUSH 機構とは“メッセージの受信を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アドレスに従って, 受信側ホストの介在なくメッセージの受信を行う” 通信機構である. IPUSH 機構により, RDMA で低レイテンシな Message Passing を実現することが可能となる.

2 章より, IPUSH 機構ではメッセージ受信のレイテンシを抑えるために, アドレス管理, データの受信処理等を全てネットワークコントローラ上のハードウェアで実行する. 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する. これは, 一続きのメッセージが複数パケットに分かれて送信された場合, それらのパケットが連続して受信されるとは限らないため, メッセージをホストから読み出す際の処理が複雑になると考えられ, また, あるプロセスからのメッセージを優先的に読み出す等の操作が難しくなり, 柔軟性を欠くためである. 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 機構ではこの問題も同時に解決することを目指す.

3.1 IPUSH 機構の設計

IPUSH 機構の概観を図 1 に示す.

IPUSH 機構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ことで実現される.

- Address Management Table(AMT): 各プロセスの受信領

☆ ページアウトされない領域 (Pindown 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

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

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この ID をアドレスとして 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. 空きが無い場合は受信領域が空くまで待つか, メッセージの破棄を行い, 再送を送信元に要求することになる. これはネットワークコントローラの実装に依存する. 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, 処理が完了する. このテー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. しかしながら, AMT Addres Table を用いたことによるレイテンシの削減幅と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ことも可能である. 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.

AMT Address Table のエントリは各プロセス ID に対して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, 例えばプロセス ID=0〜9 で同一の受信領域を, プロセス ID=10〜19 で同一の受信領域を使用するというような設定が可能である. 図 2 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受信バッファに格納されるようになる.

一般に, 全ての並列アプリケーションで, あるプロセスが全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. 例えば, NAS Parallel Benchmarks では IS, FT では全てのプロセスと通信を行うが, CG, MG では一部のプロセスとしか通信を行わない. 従って, 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受信領域を確保し, それ以外のプロセスに対してはこの機構での受信を行わない, または共通の受信領域を 1 つ確保し, そこに受信することで, 受信領域を節約す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て設定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.

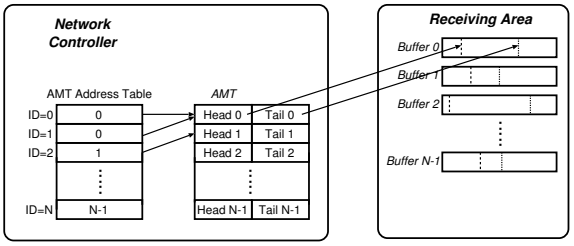

図 2 AMT Address Table の使用例

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, メッセージ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, メッセージサイズ等の情報を書き込み, Tail Pointer を更新する. ホストはこのバッファを読み出すことでメッセージ受信の検知, 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[13] のドアベルと同様の機構である.

4. DIMMnet-2 への IPUSH 機構の実装

本章では, 3 章で述べた IPUSH 機構を DIMMnet-2 に対して実装する場合について述べる.

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 バスに装着されるのに対し, DIMMnet-2 はホストの DDR-SDRAM スロットに装着される. メモリスロットに装着することにより, 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.

4.1.1 DIMMnet-2 試作基板

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. DIMMnet-2 試作基板の概観を図 3 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:10Gbps) 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-

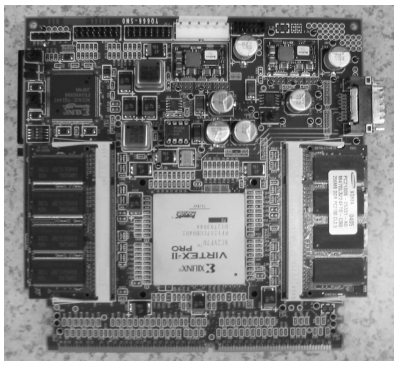

図 3 DIMMnet-2 試作基板の概観

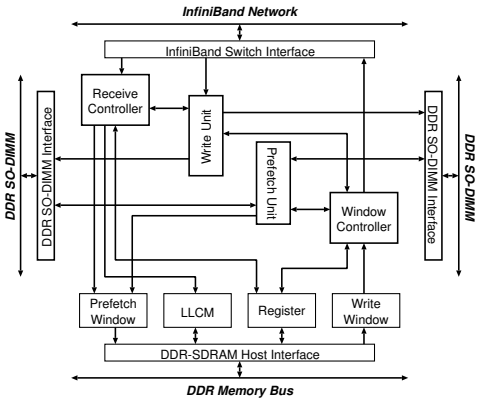

図 4 ネットワークコントローラのブロック図

DIMM はホストのアドレス空間上にはマップされず, 従って, 直接ホストから SO-DIMM にアクセスすることはできない. このような形態を取ることで, ホスト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る. ホストから SO-DIMM に対してアクセスするために, 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[14] のブロック図を図 4 に示す.

各ブロックの機能は以下の通りである.

- 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, 要求の発行等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 Unit:SO-DIMM のデータ読み出し制御部

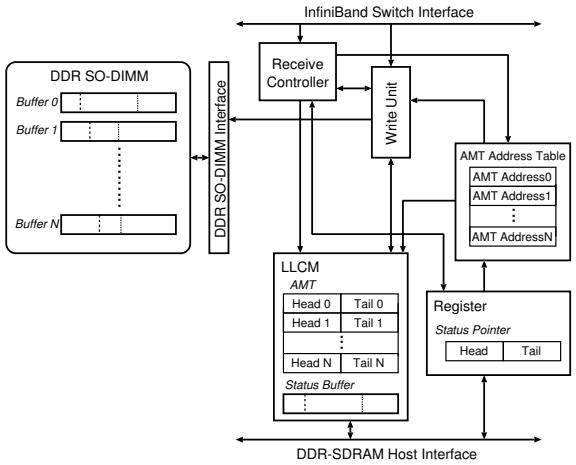

図 5 IPUSH 機構の実装

- Write Unit:SO-DIMM へ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.

パケットを SO-DIMM に受信する処理の流れは以下の通りである.

(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

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な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.

これらのブロック間の接続図を図 5 に示す.

IPUSH パケットを受信した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) パケットヘッダから, IPUSH パケットであると識別した場合, 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等を通知する
(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 からの出力を元に 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

この通信は eagerly に行われるため, 受信側のホストは都合のよ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得る.

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 Pindown 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, 圧迫を抑えるために受信領域を小さく確保することになり, この状況はより高い確率で起こり得る.

DIMMnet-2 では他の 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等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 1GByte 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 2GByte である. 一方で, 他の 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 64MByte[15], Myrinet/PCI-X E Card で 2MByte または 4MByte, Mellanox 社の InfiniHost III Ex HCA Cards シリーズで最大 128MByte である. これらの PC クラスタ向け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ネットワークインタフェース上に確保した場合, 各プロセスに確保可能な領域が小さく, ホストが頻繁にメッセージの受信をチェックする必要があり, オーバヘッドが大きくなると考えられる. この点で DIMMnet-2 のアーキテクチャは eagerly なメッセージ通信に適している.

また, ネットワークインタフェース上のメモリにメッセージを受信することによる利点として, 以下の機能を持つ IPUSH 用に確保した領域読み出し用命令を追加することが可能であることが挙げられる.

- AMT の Head Pointer をネットワークコントローラで更新
- リングバッファとして確保した領域の終端から始端に戻る場合の読み出し

この命令を追加することにより, ホストからの Head Pointer の更新はメッセージ受信ステータスのみとなる上に, ホストから 2 回の読み出し要求を必要とする終端から始端に戻る読み出しを, 1 回の読み出し要求で実行することが可能となる. 通常の受信領域読み出し命令にアドレス計算, リングバッファの終端を超えるかどうかの判定を行う論理を追加するだけで容易に実装可能であり, DIMMnet-2 にこの命令の追加を行った.

表 1 IPUSH Area Size

	Before	After	Diff
Block RAM	146/328(44%)	152/328(46%)	+6
Slices	15952/33088(48%)	16390/33088(49%)	+438

合成ツール: Xilinx ISE 6.3i SP3
デバイス: XC2VP70-7FF1517C

5. 評　　価

本章では DIMMnet-2 に実装した IPUSH 機構の評価を示す.

5.1 ハードウェア量

表 1 に IPUSH 機構を追加する前と後のネットー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, Block RAM の使用量が 2%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. しかし, 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 Block RAM の使用量を抑えることが可能である.

一方, スライス数の増加は全体の 1%であり, IPUSH 機構が低いハードウェアコストで実装可能であると示された.

5.2 パケット受信処理に要するクロック数

IPUSH によるパケットと, 通常の RDMA によるパケットの受信処理に要するクロック数を RTL シミュレーションで比較する. この結果を表 2, 表 3 に示す. パケットサイズはパケットヘッダ 16Byte(2 フリット), データサイズ 8Byte の計 24Byte である. これらの表中の (RC), (WU) はそれぞれ, Receive Controller, Write Unit の処理であることを示す.

AMT と AMT Address Table の 2 個のテーブルを引き, 受信領域のアドレス計算を行っているにもかかわらず, IPUSH は

表 2 IPUSH パケット (24Byte) の受信処理の詳細

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

表 3 RDMA パケット (24Byte) の受信処理の詳細

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

RDMA より 4 クロックの増加で済んでいる. これは, IPUSH では 1st Header を SWIF から受信する際に, 同時に Header のプロセス ID に相当する部分を AMT Address Table にフォワードし, Receive Controller がヘッダ解析を行っている間にテーブル引きが完了するためである. DIMMnet-2 試作基板のネットワークコントローラは 100MHz で動作するため, 処理時間は 40ns の増加となるが, 2 章で示したソフトウェアで通信処理を行うインターコネクトと, ハードウェアで行うものとの差を考慮すると, 十分に低い値であると言える.

6. まとめと今後の課題

本報告では, 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, 及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 に対し実装し, ハードウェアコスト, 受信処理に要するクロック数を示した. その結果, 低いハードウェアコストで低レイテンシな IPUSH 機構を実装可能であることが示された.

現在, IPUSH 機構は実機上で動作検証を行っており, 今後, ネットワークスイッチを介したプロセス間の通信レイテンシの測定や AMT Address Table を用いることによる有効性を示していく予定である.

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, 開発にご参加頂いている (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, 宮代氏に感謝致します.

参 考 文 献

1) Nanette J. Boden, Denny Cohen, Robert E. Felderman, Alan E. Kulawik, Charies L. Seitz, Jakov N. Seizovic and Wen-King Su: Myrinet - A gigabit per second local area network, *IEEE Micro*, Vol. 15, No. 1, pp. 29–36 (1995).
2) Fabrizio Petrini, Wu-chun Fang, Adolfy Hoisie, Salvador Coll and Eitan Frachtenberg: The Quadrics Network: High-Performance Clustering Technology, *IEEE Micro*, Vol. 22, No. 1, pp. 46–57 (2002).
3) InfiniBand Trade Association: http://www.infinibandta.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) Myricom: http://www.myri.com/.
10) LLOYD DICKMAN: BEYOND HERO NUMBERS: FACTORS AFFECTING INTERCONNECT PERFORMANCE, *PathScale White Paper* (2005).
11) Mellanox: http://www.mellanox.com/.
12) PathScale: http://www.pathscale.com/.
13) Intel: Intel Virtual Interface(VI) Architecture Developer's Guide Reveision 1.0 (1998). Available from ftp://download.intel.com/design/servers/vi/intel.pdf.
14) 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).
15) Quadrics: http://www.quadrics.com/.