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

2006－ARC－167（9）
2006－HPC－105（9）
2006／2／27

DIMMnet-2における通信ライブラリMPI-2の実現

荒木健志[†]　森　拓郎[†]　金井　遵[†]
田邊　昇[††]　天野英晴[†††]　並木美太郎[†]
中條拓伯[†]

本論文では，メモリスロット搭載型ネットワークインタフェース DIMMnet-2 における，通信ライブラリ MPI-2 の設計案，及び実装を述べる．実装には，アルゴンヌ国立研究所とミシシッピ大学が開発した MPICH2 を利用した．通信の実装は，MPICH2 の通信層 ADI において実現した．現段階では，0bytes 転送において，Eager プロトコルで 5.12us，Rendezvous プロトコルで 13.11us の遅延が認められた．

Design and Implementation of MPI-2 Communication Library on DIMMnet-2

Takeshi Araki,[†] Takuro Mori,[†] Jun Kanai,[†] Noboru Tanabe,[††]
Hideharu Amano,[†††] Mitaro Namiki[†] and Hironori Nakajo[†]

DIMMnet-2 is a Network Interface plugged into Memory Slot on commodity PC. In this paper, We introduce a Design and Implementation of MPI-2 Communication Library on DIMMnet-2. Using MPICH2-ADI Layer, The 0Bytes transfer latency is 5.12 us(Eager Protocol) and 13.11us(Rendezvous Protocol).

1. はじめに

コモデティプロセッサの進化に伴い，パーソナルコンピュータ（PC）を利用した安価なクラスタの利用が広がっている．PC を利用したクラスタにおいて，ボトルネックになりやすいのはネットワーク性能とメモリ性能である．特にメモリ性能は，PC のメモリ搭載容量の限界，コモデティプロセッサのキャッシュアーキテクチャの弱点等により，SX-6i のようなベクトル型スーパコンピュータとの性能差が著しい．

これらの問題点を解決するシステムとして，メモリスロット搭載型ネットワークインタフェース DIMMnet-2 が開発された[6]，DIMMnet-2 では，内部への大容量 SO-DIMM の搭載，SO-DIMM へのベクトル間接アクセス，メモリスロットの低遅延性を利用したユーザレベルアクセス，Transport 層を独自のハードウェアにより処理する InfiniBand ネットワーク等により，PC の弱点を克服するクラスタ用ネットワークとして，最適なシステムとなっている．

近年，クラスタ上の並列アプリケーション開発においては，メッセージ通信ライブラリとして，米国アルゴンヌ国立研究所が策定した Message Passing Interface（MPI）[1] を利用するのが主流になっている．現在では，片方向通信，ダイナミックプロセスマネージメント等，より柔軟性を持たせた MPI-2 が策定されている．MPI に準拠したライブラリは，新規開発のクラスタシステムにおいて例外なく採用され，実装されている．

本論文では，DIMMnet-2 を利用したクラスタにおいて，MPI-2 に準拠した通信ライブラリを実現し，新規開発のクラスタシステムにおける MPI-2 ライブラリ実現の道筋を明らかにする．

2. DIMMnet-2

DIMMnet-2 はメモリスロットに搭載され，ユーザレベルからのアクセスが可能なハードウェアである．そのため，複数プロセスからのアクセスに対する排他制御は，１プロセス毎にハードウェア資源を割り当てることで実現している．DIMMnet-2 が１プロセスに割り当てる資源を以下に列挙し，図１にユーザからみ

† 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
††† 慶應義塾大学
Keio University

た DIMMnet-2 の概観を示す.

(1) SO-DIMM
(2) Write Window(WW)
(3) Prefetch Window(PW)
(4) Low Latency Memory(LLCM)
(5) User Register(UR)

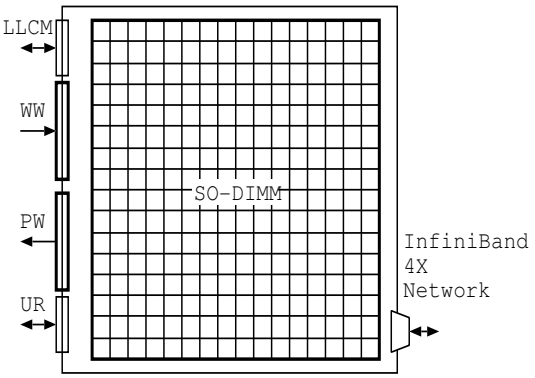

図 1 DIMMnet-2 概観

DIMMnet-2 を扱うプロセスは，デバイスドライバの mmap() システムコールを使い，プロセスの仮想アドレス領域を，DIMMnet-2 が存在する物理アドレス領域にマップする．その後，ユーザレベルから (2)(3)(4)(5) の資源にアクセスを行う（ユーザレベルアクセス）．プロセスは，(4)UR を経由して DIMMnet-2 コントローラに対し，コマンドの発行，状態の取得を行う．直接アクセス不可能な (1)SO-DIMM には，書き込みアクセスの場合，(2)WW にデータを書き込み，UR に所定のコマンドを発行することで行う．逆に読み込みアクセスの場合は，UR にコマンドを発行し，(3)PW にデータ読み込んだ後，PW 経由でアクセスする．これらを間接アクセスと呼ぶ．以下に UR 経由に発行可能なコマンドを列挙する．

(1) Vecter Load(VL)
(2) Vecter Store(VS)
(3) Remote Vecter Load(RVL)
(4) Remote Vecter Store(RVS)
(5) Indirect Vecter Store(IVS)
(6) Block On The Fly(BOTF)

VL，VS は，SO-DIMM へのベクトルアクセス命令である．連続領域へのアクセスの他，ベクトルストライドアクセス，ベクトルリストアクセス，ベクトル Packed リストアクセスが可能である．VL は SO-DIMM から PW にデータを転送し，逆に VS は WW から SO-DIMM にデータを転送する．通信命令 RVS, RVL はネットワーク経由で他ノードの SO-DIMM 領域に対し，同様のアクセスを行う．IVS は，通信相手先ノードにデータを転送すると，そのノードにおいて，データが SO-DIMM に構成されたリングバッファに書き込まれる命令である．データが書き込まれる位置は，DIMMnet-2 コントローラにより LLCM に書き込まれる Next Write Addr を用いてプロセスに通知される．プロセスは，データを読み込み後，LLCM の Next Read Addr をそのサイズ分インクリメントすることで，リングバッファを構成している．NWA 〈 NRA となった場合，データの受信操作は停止されるので，リングバッファが上書きされることはない．IVS は，データ転送元プロセスごとに SO-DIMM リングバッファを確保できる（IPUSH）[7]．従って，ハードウェアによる一対一の仮想コネクションを構成することが可能となっている．BOTF は，WW に書き込まれた DIMMnet-2 のパケットイメージを直接ネットワークに転送する命令である．これにより，DIMMnet-2 によるパケット構築遅延を減らし，遅延の少ないデータ転送が可能である[8]．同時に，WW に書き込むパケットヘッダの TYPE フィールドを書き換えることで，RVS，RVL，IVS 等，DIMMnet-2 が扱う様々なパケットを構築することができる．

3. MPICH2

MPI の実装においては，MPICH[2] が広く利用されている．これは MPI の策定元であるアルゴンヌ国立研究所とミシシッピ大学が作成した MPI の模範実装である．MPICH は様々なシステムへの移植性を考慮し，MPI 層と通信層を切り分けている．MPI-2 に準拠した MPICH2 を元に開発されたシステムには InfiniBand 上の MPI-2 実装である MVAPICH2[4] や，スーパコンピュータ BlueGene/L のメッセージパッシングサービス[5] 等が存在する．

3.1 Abstruct Device Interface(ADI)

MPICH2 の通信層は Abstruct Device Interface（ADI）[3] と呼ばれる．この層は下層に Channel（CH），RDMA Channnel（RDMA CH）等複数の通信層を持つ（図 2）．通信層は，上層ほど，柔軟性は高くなるが，実装が困難になり，逆に下層ほど，実装は容易だが，柔軟性が低くなるため，システムの性能をフルに発揮するのが困難になる．DIMMnet-2 への実装においては，内蔵 SO-DIMM を扱う必要等，より柔軟性を必要とすると考えられるので，最上層の ADI を利用した．

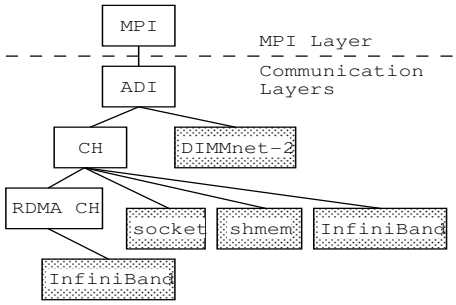

図 2 MPICH2 レイヤ構成

3.2 Eager，Rendezvous プロトコル

MPICH2-ADI 層では，Eager 及び Rendezvous 二つの通信プロトコルを実装することを前提として設計されている．図 3 にその様子を示す．

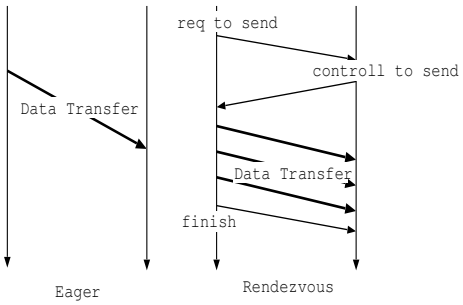

図 3 Eager，Rendezvous プロトコル

Eager プロトコルは，送信側はデータを所定のバッファに転送し，受信側はそのバッファから受信領域にコピーするプロトコルである．送信側は所定のバッファにデータを転送すると直ちに送信関数からリターンできるので低遅延な転送が可能である．反面，受信バッファからの受信領域へのコピーを行うので，一定以上のデータ転送においてバンド幅を低下させる．

Rendezvous プロトコルは，あらかじめ，送信側，受信側で同期を取り，データを転送する方式である．同期の遅延がかかるものの，ゼロコピー転送をサポートしたシステムでは，データを直接送信領域から受信領域に転送できるため，バンド幅を低下させずに通信を行える．

4. 仮想コネクション

MPI によるデータ転送では，通信ノード毎に以下に挙げる二点を保証しなければならない．

(1) データ到着
(2) データの到着順序

一対一のプロセス同士の通信において，この二つを保証した通信路を仮想コネクションと呼ぶ．仮想コネクションでは，ある通信相手からのデータが必ず，(1)(2) を保証する必要がある．仮想コネクションを実現する方法は，TCP/IP に代表されるカーネルの処理による方法，プロセスの自身の処理よる方法等が存在するが，DIMMnet-2 においては，Transport 層をハードウェアで実現し，さらに，IVS 命令によって，通信元毎にリングバッファを構成できるため，仮想コネクションを IVS 命令を用いてハードウェアにより実現できる．よって本実装では IVS 命令によるリングバッファを利用して仮想コネクションを実現する．

5. プロトコル設計

本章では，DIMMnet-2 における Eager，Rendezvous プロトコルの設計案を示す．現段階では，SO-DIMM 間の転送のみで通常のメモリ領域からの転送を想定していない．MPI のデータには，そのデータの特性を受信側に伝える Envelope を付加しなければならない．Envelope は IVS で実現された仮想コネクションに対し，IVS をパケットタイプに指定した BOTF 転送で送信する．BOTF で転送することで，低遅延な Envelope 転送が行え，さらに，受信側による Envelope の解析と送信側のデータ転送を同時に行うことができ，受信側においては，IVS リングバッファ内で，Envelope とデータを 1 つの状態で存在させることができる．プロトコルはプロトコルの切り替えに対応するため，4 つのフェーズに分かれる．

(1) Envelope Phase（Eager，Rendezvous）
(2) Reply Phase（Rendezvous）
(3) Data Transpher Phase（Eager，Rendezvous）
(4) Finish Phase（Rendezvous）

Envelope Phase において，送信側は，Envelope にプロトコルの指定を付加して送信する．受信側は，Envelope とプロトコルの指定を受信することで，次に行う動作を決定する．Eager の場合，送信側が直ちに Data Tranfer Phase に移行しデータを送信するので，受信側にはデータが既に届いている可能性が高い．Rendezvous を指定された場合，受信側は Reply Phase に移り，データ送信先を指定したデータ送信許可パケットを送信する．データ送信許可パケットを受信した送信側は，受信先に RVS でデータを送信する．データの転送を終了すると，最後に Finish Phase に移り，Finish パケットを送信する．受信側は Finish パケットを受信することで，データの転送終了を感知する．Eager プロトコルを図 4 に，Rendezvous プロ

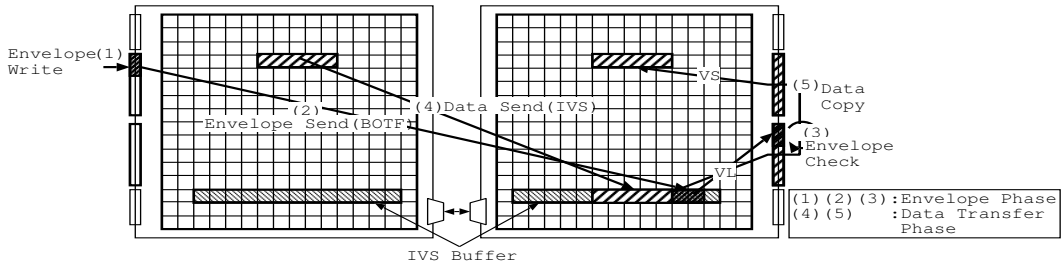

図 4 Eager プロトコルによる転送

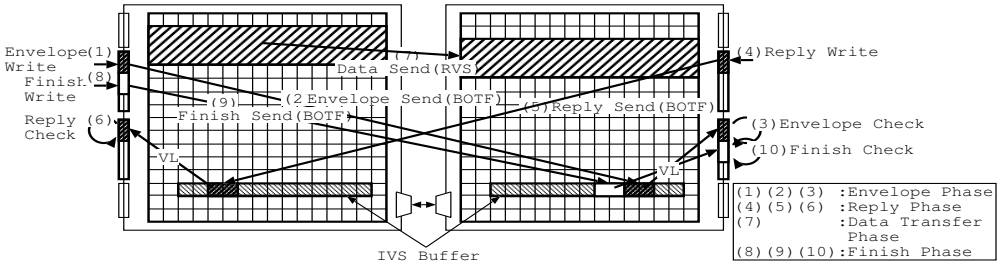

図 5 Rendezvous プロトコルによる転送

トコルを図 5 に示す.

6. ADIver3.0 による通信の実現

本章では，実際に ADIver3.0 において，第 5 章に示したプロトコルを実現する詳細について述べる．

6.1 Request

ADI の通信では，基本的に，Request を関数間で発行，解析することで通信が実現されている．Request が挿入される Queue には以下の三種類が存在する．

(1) Send Request Queue
(2) Receive Unexpected Queue
(3) Receive Posted Queue

Send Request Queue はコネクション毎に存在し，他の Queue はプロセスに 1 つずつ存在する．送信を行う関数は，呼び出されると直ちに送信操作に入る．Eager の場合は直ちに全てのデータを送信するので，Request は発行しない．Rendezvous の場合，送信処理は直ちに行われないので，Request を発行し，送信処理を Progress Engine に依頼する．Unexpected Queue には，まだ受信が発行されていないが既に到着したデータに対応する Request を挿入する．受信を行う関数は，まず，Unexpected Queue を検索し，自身が受け取るデータが既に到着していた場合．対応する Request を取得し，受信操作を行う．まだ到着していなかった場合は，Request を発行，Receive Posted Queue に挿入し，受信処理を Progress Engine に依頼する．

6.2 Controll Packet と Envelope 転送

ADI では，実際のデータ転送に加え，プロトコルを制御する Controll Packet を転送する．図 6 に Controll Packet を示す．全ての Controll Packet には，先

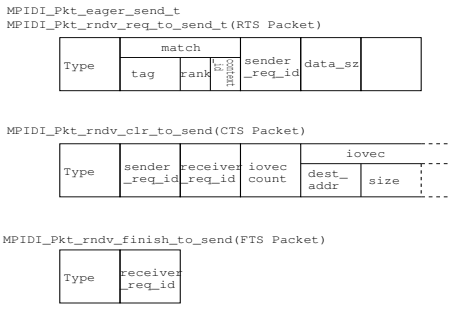

図 6 Controll Packets

頭に Type フィールドが存在し，Packet を受信したプロセスは，このフィールドを参照することで，Packet の種類を感知する．

Eager プロトコルでデータを送信する場合，データのヘッダとして Controll Packet を送信する．このヘッダは，MPIDI_Pkt_eager_send_t に示すデータ構造をもち，match と data_sz で表される Envelope を含む．

Rendezvous プロトコルは，Request To Send（RTS），Controll To Send（CTS），Finish To Send（FTS）の三種類の Packet で制御される．送

信側は，最初に送信要求にあたる RTS Packet を送信する．RTS Packet には，Envelope が含まれ，そのデータ構造は MPIDI_Pkt_rndv_req_to_send_t で表され，データ構造は Eager プロトコルにおけるヘッダ（MPIDI_Pkt_eager_send_t）と等しい．また，sender_req_id フィールドに，自身が送信のために発行した Request を特定する ID を設定する．RTS Packet を受信した受信側は，送信許可にあたる CTS Packet を送信側に送り返す．この Packet には，送信側の RTS Packet で送信された sender_req_id と，自身が受信のために発行した Request を特定する receiver_req_id が含まれる．CTS Packet を受信した送信側は，sender_req_id から対応する Request を特定して送信処理を行い，最後に送信終了を示す FTS Packet を受信側に送り返す．この Packet には，CTS Packet で送られた receiver_req_id が含まれており，受信側は，この ID から対応する Request を特定し，受信操作を終了させる．

6.3 Progress Engine

ADI の通信において，通信関数で直ちに通信が終了しなかった Request については，全て Progress Engine と呼ばれる関数群によって行われる．以下に Progress Engine に含まれる関数を示す．

(1) MPID_Progress_start()
(2) MPID_Progress_wait()
(3) MPID_Progress_end()

Progress Engine の中心は (2) の関数である．(1)(3) は (2) の関数が呼び出される前後で呼び出され，スレッドにおけるロックの取得など，(2) の関数の全処理と後処理を行う．以下に (2) が行う処理の概要を示す．

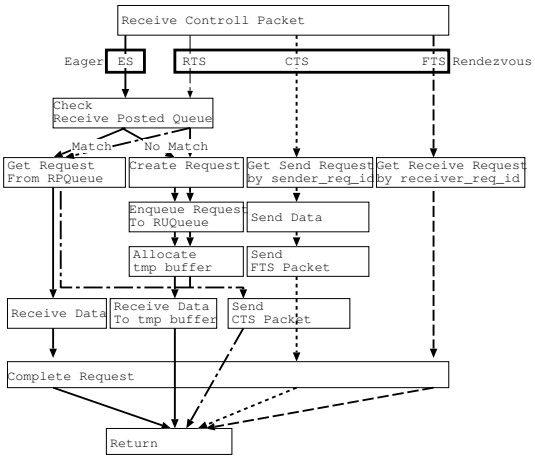

図 7 MPID_Progress_wait の処理

7. MPI 関数による通信処理の例

ここでは，ADI を利用する MPI のブロッキング通信関数 MPI_Send() の処理を示す．

```
MPI_Send()
{
  MPID_Send(Request)

  MPID_Progress_start(Request)
  while(!Request->completed) {
    MPID_Progress_wait()
  }
  MPID_Release_request(Request)
  MPID_Progress_end()
}
```

MPI_Send() はまず，対応する ADI 関数（MPID_Send()）を呼び出し，送信処理を依頼する．Rendezvous プロトコルの場合，直ちに，送信処理は終了しないので，MPID_Send() は Request を発行し，MPI_Send() に渡す．MPI_Send() は while 文中で，MPID_Progress_wait() を呼び出し，Request が Completed になるまで Busy Wait する．MPI 関数はすべて類似した方法で実装されている．例えばノンブロッキング通信である．MPI_ISend() の場合，Request がユーザに渡され，ユーザは MPI_Wait() にその Request を渡す．MPI_Wait() 内で，上記の While 文と同様の処理が行われることにより，通信の終了を保証することができる．

8. 評　　価

8.1 評 価 環 境

表 1 に，評価環境を示す．ネットワークは，DIMMnet-2 が搭載されたノードを 2m の InfiniBand 4X ケーブルで対向接続した．CPU の Cache 属性は，MTRR によって，WW を WriteCombine，PW を Uncachable に設定して測定した．

表 1 評価環境

CPU	Pentium4 2.6GHz
Chipset	VIA VT8751A
Memory	PC1600 DDR-SDRAM 512MBytes × 1
DIMMnet-2	DIMMnet-2FPGA.ver 100MHz
Network	InfiniBand 4X 2m 対向接続
OS	RedHat9 kernel-2.4.26 DIMMnet-2.ver
Compiler	gcc-3.2.2(-O3)

8.2 評価方法と結果

プロトコルの遅延を測るため，ブロッキング通信

関数 MPI_Send()，MPI_Recieve() で 0Bytes 転送の 10 万回 Ping Pong を行い遅延を計測した．(1)Eager, (2)Rendezvous それぞれの結果を表 2 に示す．参考のため，(3)BOTF（IVS) 及び (4)VL の遅延，(5) ライブラリレベルでの Eager プロトコルの遅延（0Bytes 転送）を載せ，それらの結果から (6)MPI 関数呼び出し遅延を含むプロトコルの遅延，(7) ライブラリ遅延，通信を除いた処理遅延である (8) ホスト遅延を算出した．

表 2 性能測定結果

	Latency
(1)MPI Eager Protocol	5.12us
(2)MPI Rendezvous Protocol	13.11us
(3)DIMMNet-2 BOTF(IVS) 32Bytes	2.04us
(4)DIMMnet-2 VL 128Bytes	0.68 us
(5)Library Eager Protocol	3.87us
(6)Protocol Latency（(1)-(5)）	1.25us
(7)Library Latency（(5)-(3)-(4)）	1.15us
(8)Host Latency（(6)+(7)）	2.40us

結果より，MPI 関数の呼び出し，及び Request の処理によるプロトコルの計算に，1.25us(6) の遅延が認められる．ライブラリ遅延が 1.15us(7) あり，合計すると，ホスト側での遅延が 2.40us(8) であることがわかる．同じく MPICH2 を利用して実装されている InfiniPath MPI のホスト遅延が 0.5us（AMD Opetron 2.8GHz）[9] となっている．現在の実装では，WW へのデータ書き込みに C 言語の標準関数 memcpy() を使い，PW のキャッシュ属性を Uncachable で測定している．そのため，ホスト側での遅延がかなり大きく，まだ最適化の余地があると考えられる．

8.3 結論と今後の課題

本論文では，MPICH2 を DIMMnet-2 に実装し，プロトコル性能を測定した．結果として，同じく MPICH2 を利用した InfiniPath MPI に比べて，約 5 倍のホスト遅延がかかっていることを確認した．現在では，DIMMnet-2 へのアクセスを C 言語標準の関数を利用して実現しており，さらに安定性のため，PW の Cache 属性を Uncachable にして実装している．今後の課題としては，SSE2 命令，CLFLUSH 命令等，アセンブラによる最適化を進め，さらに，MPI のデータタイプ通信，片方向通信，通常メモリからのデータ転送等の実装を進める予定である．

謝辞 本研究は，総務省戦略的情報通信研究開発推進制度の一環として行われたものである．DIMMnet-2 のハードウェアについて，多大なご支援とご教授を賜った慶應義塾大学の北村氏，宮部氏，宮代氏，基本となるデバイスドライバを提供していただいた同，伊澤氏に感謝いたします．DIMMnet-2 ソフトウェア開発において，多くのご示唆を頂いた東京農工大学の濱田氏，羅氏，池田氏に感謝いたします．同時に，DIMMnet-2 の開発，議論に参加頂いたすべての方々に感謝いたします．

参 考 文 献

1) The Message Passing Interface standard http://www-unix.mcs.anl.gov/mpi/
2) The MPICH and MPICH2 homepage. http://www-unix.mcs.anl.gov/mpi/mpich
3) W. Gropp, E. Lusk, D. Ashton, R. Ross, R. Thakur, and B.Toonen. MPICH2 Abstract Device Interface Version 3.4 Reference Manual: Draft of May 20, 2003. http://www-unix.mcs.anl.gov/mpi/mpich/adi3/adi3man.pdf
4) J. Liu, W. Jiang, P. Wyckoff, D. K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen. Design and Implementation of MPICH2 over InfiniBand with RDMA Support. In Int'l Parallel and Distributed Processing Symposium (IPDPS 04), April 2004.
5) G. Almasi, C. Archer, J. G. Castanos, M. G. X. Martorell, J. E. Moreira, W. Gropp, S. Rus, and B. Toonen. MPI on BlueGene/L: Designing an Efficient General Purpose Messaging Solution for a Large Cellular System. Submitted for publication to the 2003 Euro PVM/MPI workshop.
6) 北村 聡，濱田 芳博，宮部 保雄，伊澤 徹，宮代 具隆，田邊 昇，中條天野 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).
7) 北村 聡，宮部 保雄，中條 拓伯，田邊 昇，天野 英晴: 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，情報処理学会アーキテクチャ研究会，Vol.2005- ARC-165(DesignGaia'05), pp. 39-44 (2005).
8) 宮部 保雄，北村 聡，濱田 芳博，宮代 具隆，伊澤 徹，田邊 昇，中條 拓伯，天野 英晴，DIMMnet-2 低遅延通信機構の実装と評価，情報処理学会アーキテクチャ研究会，2005-ARC-163, May.2005.
9) LLOYD DICKMAN: BEYOND HERO NUMBERS: FACTORS AFFECTING INTERCONNECT PERFORMANCE, PathScale White Paper (2005). http://www.pathscale.com/