MPI派生データ型通信支援機構のDIMMnet-2への実装と評価

宮部保雄[†] 宮代具隆[†] 北村聡[†]
田邊昇[††] 中條拓伯[†††] 天野英晴[†]

MPIの派生データ型は，メモリ上に不連続に存在するデータを1つの型として定義することで，1回の関数呼び出しで不連続なデータの通信を可能にし，効率的な並列アプリーションを開発するのに役立つ．しかし，多くのMPIの実装において派生データ型通信は，メモリへの不連続アクセスをともなう，一時領域を用いたコピーによるデータの連続化・不連続化処理がボトルネックとなり，性能が十分でない問題があった．本論文では，ネットワークインタフェース上に設けたハードウェア機構を利用して，このコピーを行わずに派生データ型通信を行う手法を提案する．この機構は，メモリへの不連続なアクセスを効率的に行うことができ，また，RDMAの起動時に，送信・受信側双方でのメモリアクセスパターンを指定できるという特長がある．この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，実装した機構を利用するMPIライブラリを開発した．そして，その性能を派生データ型を使った部分行列の転送により評価した．その結果，この機構を利用する場合，利用しない場合に比べて約5.2倍のバンド幅の向上と，転送に要する時間を約78.3%削減可能であることが分かった．

Implementation and Evaluation of Mechanisms for MPI Derived Datatypes Communication on DIMMnet-2

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

MPI derived datatypes, which allow users to communicate noncontiguous data with a single communication function, are useful for developing parallel applications. However, because of the overhead for pack/unpack operations which requires noncontiguous accesses to/from temporary memory space, the performance in many MPI implementations tends to be insufficient. In this paper, we propose to accelerate its performance by omitting pack/unpack operations using hardware mechanisms in a network interface. The mechanisms control noncontiguous memory access to the on-board SDRAM modules efficiently by specifying the memory access pattern in both the sender/receiver when starting RDMA (Remote Direct Memory Access) transfer. We implemented the mechanisms in the DIMMnet-2 prototype board: a network interface which is attached into a memory slot of the host PC, and developed the MPI library based on them. Using the mechanisms, the bandwidth of submatrix transfer by using derived datatypes is 5.2 times of the case without them, and the required time is also reduced by 78.3%.

1. はじめに

MPI（Message Passing Interfece）は，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けるアプリケーション開発で広く用いられている通信ライブラリである．MPIはアプリケーションの開発を簡便化するために多くの機能を提供するが，その1つに派生データ型がある．

派生データ型は，ユーザがアプリケーションの実行時に自由に定義できる型であり，複数のデータのメモリ上での配置情報を保持する．MPIの送受信関数は，送受信するデータの型として，この派生データ型を引数として与えることができる．これにより，メモリ上に不連続に存在する複数のデータを1回のMPI関数の呼び出しで送信することができる．このように便利な派生データ型であるが，多くのMPIの実装においてその利用が敬遠される場合が多かった．これは主として派生データ型を用いた場合の通信の性能が十分で

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

なかったことが原因であるが，最近，ソフトウエアの改良[1)~3)]などにより，その通信性能は若干改善されている．

このため，その便利さや，システム依存の最適化を MPI ライブラリに任せることができるなどの理由により，派生データ型通信を利用したアプリケーションが登場し，今後増加することが期待される．しかし，派生データ型の処理には，本質的に不連続なメモリアクセスがともない，キャッシュベースの CPU には向いておらず，ソフトウエアによる処理には限界があると考えられる．

そこで，ネットワークインタフェース上に，不連続なメモリ領域へのアクセスを効率的に行うことができるベクトルアクセス機構を実装し，この機構を用いた RDMA（Remote Direct Memory Access）により派生データ型通信を行う手法を提案する．この手法により，ホストの CPU 時間の浪費やキャッシュの汚染などを起こさずに，高速な派生データ型通信が可能になる．

本論文では，このベクトルアクセス機構を，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 の試作基板に実装した．そして，その機構を用いて派生データ型通信を行う MPI ライブラリを開発し，性能評価を行った．

以下，本論文では，2 章で派生データ型通信の問題点と我々が提案している手法の概要について述べ，3 章で DIMMnet-2 へのベクトルアクセス機構の実装を述べる．次に，4 章で実装した機構を利用する MPI ライブラリについて述べ，5 章でその評価を行う．そして，6 章でまとめと今後の課題について述べる．

2. 派生データ型通信と提案手法

2.1 一般的な派生データ型通信の問題点

派生データ型通信の性能に影響を与える主な要因は，以下の 2 つである．1 つは，派生データ型の MPI ライブラリ内部での表現手法である．送受信関数が呼ばれたとき，MPI ライブラリは派生データ型の情報から，処理に必要なデータを高速にデコードする必要がある．これについての研究はすでに行われており[1),2)]，その成果は MPI の模範実装である MPICH2[4)] などにも実装され，派生データ型通信の性能は若干改善された．

もう 1 つは，Pack/Unpack 処理の性能であり，前者の要因が改善された現在，性能を制限する主要因になりつつある．多くのネットワークインタフェースでは，メモリ上の連続したデータに対して，開始アドレスと長さを指定して転送する．このため，不連続なデータの転送が必要な MPI 派生データ型の実装では，通常，送信するデータを一時領域にコピーして連続化してから送信し，受信側の一時領域に連続に書き込む．そして，受信側では一時領域から最終的な受信領域にデータをコピーしながら，再びデータを不連続化する（**図 1**）．このデータの連続化処理を Pack，不連続化処理を Unpack といい，これらの処理に含まれる不連続なメモリアクセスをともなうデータコピーが，派生データ型通信の性能低下の要因となる．

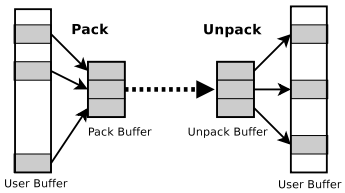


図 1 Pack 処理と Unpack 処理
Fig. 1 Packing and unpacking.

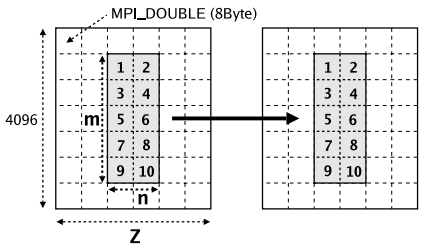


図 2 データの転送パターン
Fig. 2 Pattern of data transfer.

派生データ型通信の例として，**図 2** に示す 8 Byte の MPI_DOUBLE 型からなる $4096 \times z$ 行列の部分行列 $m \times n$ を別プロセスの $4096 \times z$ 行列へ転送することを考える☆．このような 2 次元，またはそれ以上の次元の行列の部分行列を別プロセスに転送する処理は，問題領域をブロック分割して各プロセスに割り当てる並列アプリケーションなどでよく用いられる．

この部分行列の転送に必要な Pack/Unpack 処理をホストの CPU で行った場合のバンド幅を，実際の MPI ライブラリを実行して計測した．まず，$4096 \times z$ 行列内の $m \times n$ 行列のデータを表す派生データ型を構築する．この型を用いて Pack 処理を行う MPI 関数 `MPI_Pack` と，Unpack 処理を行う MPI 関数 `MPI_Unpack` を複数回呼び出し，関数からリターンするまでの時間の合計からそれぞれの処理のバンド幅を求めた．計測は，5 章で述べる表 3 の環境で行い，MPI ライブラ

☆ これらの行列の要素は，行（n）方向に連続してメモリに格納されているものとする．

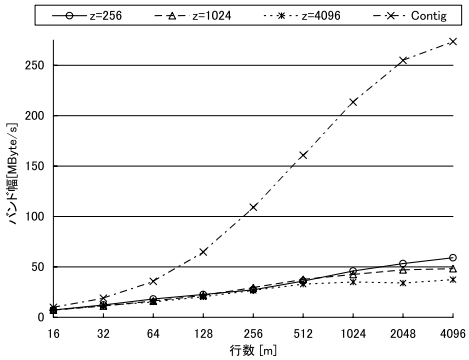


図 3 Pack 処理のバンド幅（$n = 1$）
Fig. 3 Bandwidth of Packing ($n = 1$).

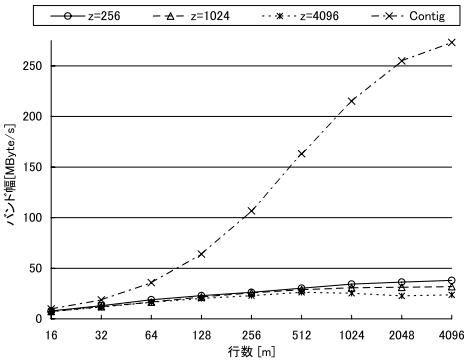


図 5 Unpack 処理のバンド幅（$n = 1$）
Fig. 5 Bandwidth of Unpacking ($n = 1$).

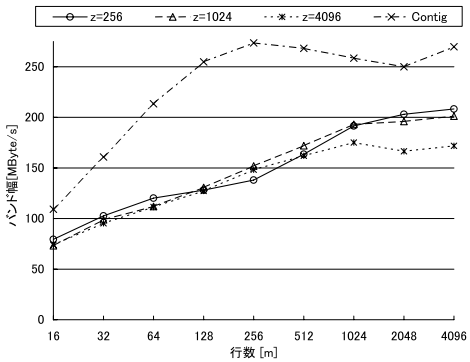


図 4 Pack 処理のバンド幅（$n = 16$）
Fig. 4 Bandwidth of Packing ($n = 16$).

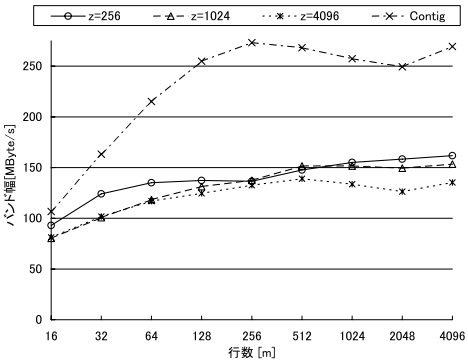


図 6 Unpack 処理のバンド幅（$n = 16$）
Fig. 6 Bandwidth of Unpacking ($n = 16$).

リには MPICH2（mpich2-1.0.2p1）を用いた．ここで，アプリケーション内で派生データ型を用いた通信が行われる場合，Pack/Unpack 処理を開始するまでに，ホストの CPU のキャッシュはそれまでの計算処理などのためにすべて使われており，Pack/Unpack 用のバッファなどはすべてキャッシュから追い出されていると想定できる．この状況を再現するために，それぞれの `MPI_Pack`/`MPI_Unpack` 関数の呼び出しの前に，Pack/Unpack 処理とは無関係な領域に対してキャッシュと同じサイズ分，ダミーアクセスを行った．

図 3，**図 4**，**図 5**，**図 6** に，n，m，z の値を変化させたときの，Pack/Unpack 処理のバンド幅を示す．これらの図の Contig とは，メモリ上に連続して存在する $m \times n$ 個の `MPI_DOUBLE` 型を表す派生データ型を構築し，この派生データ型を `MPI_Pack`/`MPI_Unpack` 関数に渡して連続データのコピーを行ったときのバンド幅である．連続データのコピーバンド幅を計測する際も，ダミーアクセスにより処理前にキャッシュがすべて使われている状況を再現した．

$m = 4096$ のときの Pack/Unpack 処理のバンド幅は，$n = 1$ のときで約 23.7〜59.1 MByte/s，$n = 16$ のときで約 135.4〜208.3 MByte/s であった．これらの値は，近年，PC クラスタなどに 10 Gbps クラスのネットワークが使われていることを考えると十分とはいえない．特に，$n = 1$ のときは Pentium4 のキャッシュラインサイズ 64 Byte 中，有効なデータが 8 Byte だけという状況になるため，$n = 1$，$m = 4096$ の Unpack 処理においては，連続データコピー時の 9〜14%のバンド幅まで性能が著しく低下している．

不連続なメモリアクセスをともなうデータコピーは，キャッシュブロッキングなどのメモリアーキテクチャを意識したコーディング手法による最適化で高速化が可能な場合がある．文献 5) では，MPI ライブラリに与えられた派生データ型のメモリアクセスパターンと TLB の状態などによりキャッシュブロッキングのブロックサイズなどの最適化のためのパラメータを動的に求めており，派生データ型を使わずに手作業で Pack/Unpack 処理を最適化した場合に近い通信性能を達成している．ただし，ホストの CPU によって Pack/Unpack 処理を行う場合，ホストの CPU の

キャッシュがその処理により汚染されるという大きな問題がある．

2.2 ストライドアクセス機能を有する計算機における派生データ型通信

メモリ上に等間隔に存在するデータへのアクセス（ストライドアクセス）を効率的に行うことができる機能を持った計算機には，その特長を活かして派生データ型通信を行う MPI ライブラリが実装されている．

Cray T3D における MPI の実装[6] では，メモリ上に連続しているデータを通信先の不連続なメモリ領域に転送する機能を利用して，明示的な Unpack 処理を一部省略している．また，NEC の SX シリーズ向けの MPI の実装である MPI/SX[7] では，Flattening of the Fly[8] と呼ばれる手法により，Pack/Unpack 処理に必要なデータコピーのメモリアクセスを，ベクトル型計算機が効率良く扱うことができるストライドアクセスに変換し，Pack/Unpack 処理を高速化している．これに対し，我々が提案する手法は Pack/Unpack 処理をハードウェアで高速化可能なうえに，汎用 PC など専用のハードウェアを持たない計算機においても適用可能であるという点で異なるといえる．

2.3 ネットワークインタフェースの機能を利用した派生データ型通信

一方，PC クラスタ向けのネットワークインタフェースの機能を利用して派生データ型通信を高速化させる試みも行われている．

文献 9)，10) では，InfiniBand の VAPI（Verbs Level API）が提供するギャザ・スキャッタ付きデータ転送機能を利用し，Pack/Unpack 処理をネットワークインタフェース上のプロセッサで走るファームウエアにオフロードしている．しかし，ネットワークインタフェース上のプロセッサはホストの CPU より周波数が大幅に低いため，サイズの小さな要素が多数集まって構成される派生データ型を処理する場合などに性能が低下するという問題がある．さらに，ネットワークインタフェースと主記憶の間の I/O バスを介して，サイズの小さな要素を多数やりとりすることは，それだけで性能低下の原因となる．

また，VAPI のギャザ・スキャッタ付きデータ転送機能は，通信相手側のプロセスによる処理が不要な RDMA を用いた転送を行う場合に，ギャザかスキャッタのどちらかしか行えないという制限がある．そのため，一時領域を利用した Pack 処理または Unpack 処理のどちらかが必要となる．ギャザとスキャッタの両方を VAPI の機能で行うには，通信相手側のプロセスで受信命令の発行が必要な send/receive による転送を行う必要があり，通信相手側のプロセスの通信オーバヘッドが大きくなるという問題がある．

2.4 提案手法の概要

本論文では，ネットワークインタフェース上に設けた，不連続なメモリアクセスを効率的に行うことができるハードウェア，ベクトルアクセス機構を利用することで，派生データ型処理に含まれる不連続なメモリへのアクセスを高速化する．この機構はネットワークインタフェース上に設けられることから，ホストには一般的な PC を利用することができ，並列システム全体を安価に構築することが可能である．

このベクトルアクセス機構は，ハードワイヤードロジックで構成され，メモリに対して，連続したデータへの単純なアクセスに加え，ストライドアクセスや，アドレスが格納されたリストを用いたデータへのアクセス（リストアクセス）を高速に行うことができる．これらのベクトルアクセス機構によるメモリへのアクセスは，ホストの CPU と独立して行われるため，ホストの CPU 時間の浪費やキャッシュの汚染などを起こさない．

さらに，本論文では，ベクトルアクセスによってメモリから読み出したデータをそのままネットワークに送出する機能と，ネットワークより受信したデータをそのままベクトルアクセスでメモリに格納する機能を，本機構に付加し，これら送信・受信側双方でのベクトルアクセスのパラメータ指定を通信の起動側がまとめて行えるようにする．この機能の利用により，一時領域を用いたデータコピーによる Pack/Unpack 処理を行わない転送が可能となり，派生データ型通信の高速化が可能となる．以降，本論文ではこの機能をリモートベクトルアクセス機能と呼称する．

3. DIMMnet-2 へのハードウェア実装

リモートベクトルアクセス機能の実現には，通信に関係するデータが格納された大容量メモリに，ベクトルアクセス機構が自由にアクセスできる必要がある．そのためには，I/O バスを介して細かなデータをやりとりするのを避けるためにも，大容量メモリがベクトルアクセス機構と同じネットワークインタフェース上に存在することが望ましい．

本研究では，大容量メモリをネットワークインタフェース上に備え，コントローラに機能の追加が容易な FPGA を用いている DIMMnet-2 試作基板を，リモートベクトルアクセス機能を備えるベクトルアクセス機構の実装対象とした．

3.1 DIMMnet-2 の概要

DIMMnet-2 は，ホストの DDR SDRAM バスに接続される 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．DDR SDRAM 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，結果として，PCI-X バスや PCI-Express に接続される一般の 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よりも，低レイテンシな通信を実現することが可能である．

DIMMnet-2 ではネットワークスイッチとケーブルに転送性能の優れた InfiniBand を採用している．

3.1.1 DIMMnet-2 試作基板

DIMMnet-2 試作基板は，Xilinx 社の FPGA（Virtex-II Pro XC2VP70-7FF1517C）を搭載しており，ここにネットワークコントローラなどを実装する．InfiniBand（4X: 10 Gbps）への接続は，Virtex-II Pro に内蔵されている RocketIO トランシーバを用いて行う．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．現在，256 MByte の SO-DIMM を 2 枚搭載している．

なお，DIMMnet-2 試作基板はコントローラに FPGA を用いているため，高い動作周波数での稼働が困難である．そのため，ホストとのインタフェースと SO-DIMM とのインタフェースは，DDR SDRAM の規格のうち最も周波数の低い PC-1600 に対応し，コントローラ全体は 100 MHz で動作させる．ただし，InfiniBand とのインタフェース部は，InfiniBand の仕様に合わせるため，125 MHz で動作させる．

3.1.2 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のブロック図を**図 7** に示す．

各ブロックの機能は以下のとおりである．

- LLCM：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：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

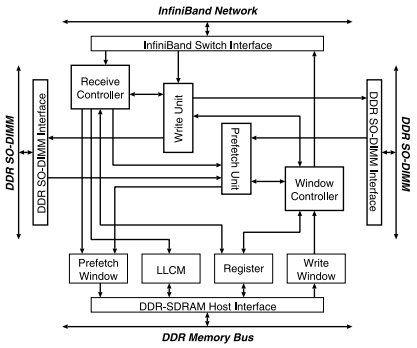


図 7 ネットワークコントローラのブロック図
Fig. 7 Block diagram of network interface controller.

表 1 ベクトルアクセス命令一覧
Table 1 List of vector access instructions.

アクセスの種類	SO-DIMM に対して	
	Read（VL 系列）	Write（VS 系列）
連続アクセス	VL	VS
ストライドアクセス	VLS	VSS
リストアクセス	VLI	VSI

- Prefetch Unit：SO-DIMM のデータ読み出し制御部
- 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 DIMMnet-2 ドライバによりユーザプロセスの仮想アドレス空間にマップされる．SO-DIMM には，ホストから直接アクセスすることはできず，Prefetch Window と Write Window を介してアクセスする．このような形態をとることで，ホストのチップセットによる制約などを受けずに，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っている．

ベクトルアクセス機構の中心となるのは，Prefetch Unit と Write Unit である．それぞれの Unit は，Window Controller の指示に従い，SO-DIMM と Prefetch Window または Write Window 間でデータを転送する．この転送命令は，現在までに**表 1** に示す SO-DIMM のデータを読み込む VL 系列の 3 命令と，SO-DIMM にデータを書き込む VS 系列の 3 命令が実装され，SO-DIMM への不連続なアクセスを高速で行うことができるようになっている．ストライドアクセス命令，またはリストアクセス命令は，4/8/16/32/64/128 Byte の連続するデータを 1 つの要素とし，最大で 255 個までの要素を 1 回の命令発行でアクセスする．命令発行時には，データの読み込み元/書込み先のアドレスのほか，要素の大きさ（DType），

アクセス回数（Iteration），ストライドの大きさ（ストライドアクセスの場合），アドレスが格納されたリストのアドレス（リストアクセスの場合）などのパラメータを指定する．

3.2 リモートベクトルアクセス機能

本論文では，データの送信側における VL 系列のベクトルアクセスと，データの書き込み側における VS 系列のベクトルアクセスを，通信の起動側が 1 度にまとめて指定できるリモートベクトルアクセス機能を実装した．この機能は，通信起動側のメモリのデータを通信相手のメモリに書き込む RDMA Write（リモートベクトルライト）にも，通信相手のメモリのデータを通信起動側のメモリに書き込む RDMA Read（リモートベクトルリード）にも適用可能である．

リモートベクトルアクセス起動時に指定された DType や Iteration などの，送信側と受信側での 2 つのベクトルアクセスの起動パラメータが，それらで処理するデータサイズにおいて矛盾を起こす場合，データの受信側で SO-DIMM への書き込みが正しく行われなくなる．この問題は，パラメータの一部をネットワークコントローラ内部で計算することで解決できるが，ハードウェア量や遅延の増加を招くため，本実装ではホストのソフトウェアで，パラメータの整合性を保証する．

また，DIMMnet-2 は InfiniBand のパケットの MTU を 2 KByte としているため，転送するデータサイズに応じてパケットの分割を行う．分割されたパケットには，そのすべてに受信側で行わせるベクトルアクセス命令の起動パラメータがヘッダに付加される．このパラメータは，送信側の Window Controller によってパケットごとに適切な値が計算されるため，データの受信側はパケットの分割を意識することなく，ベクトルアクセスによる SO-DIMM への受信データの書き込みを行うことができる．

通信相手のプロセスが通信の起動に関与しない RDMA では，データの転送が完了したことを検出するのに工夫が必要となる．InfiniBand の RDMA を用いた MPI の実装[11] では，RDMA で送信するデータの最後にフラグを書き込み，プロセスはこのフラグをポーリングすることでデータの到着を検出している．DIMMnet-2 ではデータの受信が完了すると，どのプロセスが，SO-DIMM のどの領域に，どれだけのデータを書き込んだかの情報を含む受信ステータスを LLCM 上に確保されたリングバッファに書き込む．これにより，RDMA 起動前に，送信するデータにフラグを付加する処理が不要になる．なお，受信ステータスの書き込みを行うか否かは，リモートベクトルアクセス起動時に指定することができる．

4. MPI ライブラリの実装

本章では，リモートベクトルアクセス機能を利用した MPI ライブラリの実装について述べる．

4.1 MPICH2

MPICH2[4] は，アルゴンヌ国立研究所が中心となり開発されている MPI の模範実装である．我々はかつて，この MPICH2 をベースに DIMMnet-2 上における MPI の実装を行った[12]．本論文では，この実装を改造することで，DIMMnet-2 のリモートベクトルアクセス機能を利用した派生データ型通信を実現した．ただし，DIMMnet-2 の SO-DIMM と通常のメモリを同じようにユーザプログラムからアクセスするためのドライバの整備が完了していないため，派生データ型通信の対象は，DIMMnet-2 の SO-DIMM 上のデータに限定している．

MPICH2 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．この通信層で，Eager および Rendezvous の 2 つのプロトコルを実装することを前提として MPICH2 全体が設計されている．

Eager プロトコルでは，送信側はデータを受信側の所定の一時領域に転送し，受信側はそのバッファから受信領域にデータをコピーする．低遅延な転送が可能であるが，バッファ間のコピーが必要であり，大きなデータの転送には向かない．

一方，Rendezvous プロトコルでは，データの送受信に先立って，送信側，受信側で同期をとり，データを転送する．同期に遅延がかかるものの，DIMMnet-2 のリモートベクトルアクセス機能など，RDMA によるデータ転送を利用すれば，一時領域からのデータのコピーを行わず，データを直接，送信領域から受信領域に転送できるという利点がある．

4.2 DIMMnet-2 の派生データ型通信

本論文では，Rendezvous プロトコルによる派生データ型通信を実装する．RDMA 機能によって Rendezvous プロトコルを実装する場合，RDMA Write を使う手法と，RDMA Read を使う手法が考えられるが，RDMA Read による実装は，RDMA Write による実装に比べ，次の 2 つの利点がある[13]．

- プロトコルを制御するパケットの数が少なく済む．
- 受信側が Rendezvous を開始する要求を送信側から受け取りさえすれば，あとは受信側が送信側から独立して，データ転送処理を行うことが可能であり，送信側が通信終了まで通信処理とは別の処

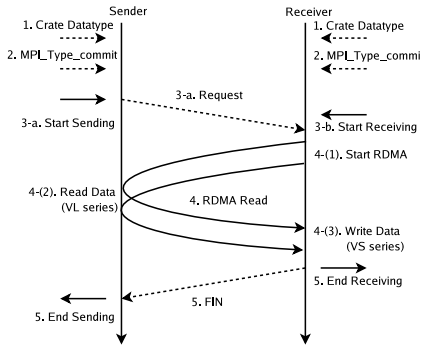


図 8 DIMMnet-2 の派生データ型通信のプロトコル
Fig. 8 Protocol of derived datatypes communication over DIMMnet-2.

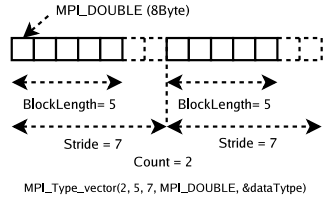


図 9 `MPI_Type_vector` の例
Fig. 9 Example of `MPI_Type_vector`.

表 2 パラメータリストの例
Table 2 Example of parameter list.

	DType	Iteration	Offset	Stride
1	32 Byte	2	0 Byte	56 Byte
2	8 Byte	2	32 Byte	56 Byte

理を行うことができる時間が長い．

そのため，今回は DIMMnet-2 のリモートベクトルリードを用いた実装を行った．今回実装した Rendezvous プロトコルによる派生データ型通信の流れを**図 8** に示す．

1 派生データ型の構築

MPI の派生データ構築用の関数が呼び出されると，MPI ライブラリはその型情報を DIMMnet-2 のベクトルアクセス命令の起動パラメータとして主記憶上に保持する．

`MPI_Type_vector` によって構築される派生データ型は，DIMMnet-2 のストライドアクセス命令の起動パラメータのリストで表される．たとえば，`MPI_Type_vector` の呼び出しによって，**図 9** のような派生データ型を構築した場合，MPI ライブラリは**表 2** のような，ストライドアクセス命令の起動パラメータのリストを生成する．一方，`MPI_Type_indexed` が呼び出された場合は，派生データ型の要素のメモリ上での配置情報をアドレスのリストとして主記憶上に記録し，DIMMnet-2 のリストアクセス命令の起動パラメータのリストを生成する．

なお，現在の実装では，`MPI_Type_struct` で構築される派生データ型や，派生データ型を要素とする派生データ型などを扱うことができない．このような複雑な派生データ型については，Flattening on the Fly[8] の手法を用いることで，DIMMnet-2 のストライドアクセス命令ですべて扱うことができるようにする予定である．

2 `MPI_Type_commit` 関数の呼び出し

MPI では，派生データ型構築用の関数を用いて構築した派生データ型を通信関数に渡す場合，構築した派生データ型を引数に `MPI_Type_commit` 関数を呼び出す決まりになっている．本論文の MPI の実装では，この `MPI_Type_commit` が呼ばれたときに，`MPI_Type_indexed` の呼び出し時に主記憶上に記録されたアドレスのリストを SO-DIMM に書き込み，通信関数の呼び出しに備える．一般的に 1 度構築された派生データ型はその後何度も使い回されるため，この処理にかかるコストは，アプリケーション全体の性能には大きな影響を与えないと考えられる．

3-a 送信関数の呼び出し

送信を行う関数が呼ばれると，送信側はただちに送信側の派生データ型を表すベクトルアクセス命令の起動パラメータのリストと，送信するデータの先頭アドレスの情報が含まれる送信要求パケットを受信側に送る．

3-b 受信関数の呼び出し

受信を行う関数が呼ばれると，受信側はそれ以前に送られてきた送信要求パケットに Tag や Rank などの条件が一致するものがないか検索する．なければ，要求を MPI ライブラリ内のキューに格納し，条件が一致する送信要求パケットが送られてくるまで待つ．

4-(1) リモートベクトルリードの実行

条件に一致する送信要求パケットが見つかると，データの送信側と受信側の派生データ型を表すベクトルアクセス命令の起動パラメータの情報から，リモートベクトルリードの起動パラメータを決定し，実行する．表 2 のように複数のベクトルアクセスが必要な派生データ型や，Iteration の数が 255 を超え，ベクトルアクセスを分割して実行する必要がある場合には，リモートベクトルリードは複数回実行される．このとき，最後のリモートベクトルリードの起動のときにだけ，受信ステー

タスの書き込みを行うフラグを有効にする．

4-(2) VL 系列のベクトルアクセスによるデータ読み込み

リモートベクトルリードを要求するパケットを，データの受信側から受け取った送信側の DIMMnet-2 は，要求パケットに含まれる起動パラメータに従って VL 系列のベクトルアクセス命令を実行し，SO-DIMM からデータを読み出す．そして，データ送信側の DIMMnet-2 は，リモートベクトルリードを要求するパケットに含まれていた，データの受信側で起動させる VS 系列のベクトルアクセスの起動パラメータのリストを，SO-DIMM から読み出したデータと合わせてデータの受信側に送り返す．これらの動作は，DIMMnet-2 のネットワークコントローラがホストの CPU とは独立に行う．

4-(3) VS 系列のベクトルアクセスによるデータ書き込み

データ受信側の DIMMnet-2 は，送信側より受け取ったパケットに含まれる起動パラメータに従って，VS 系列のベクトルアクセス命令を実行し，SO-DIMM に受信したデータを書き込む．この動作も，DIMMnet-2 のネットワークコントローラがホストの CPU とは独立に行う．

5 通信の終了

受信側は，最後に実行されるリモートベクトルリードによるデータ書き込みが行われたことを，受信ステータスにより検出する．検出後，ただちに送信完了パケットを送信側に送り，送信操作を終了させる．その後，送信完了パケットを送信した受信側は，受信操作を終了する．

5. 評　　価

本章では，リモートベクトルアクセス機能を利用した MPI 派生データ型通信の評価を示す．

5.1 評価手法

表 3 に評価環境を示す．DIMMnet-2 を搭載した PC を 2 台，InfiniBand スイッチを介して接続した．この 2 台の PC 間で，図 2 に示した $4096 \times z$ 行列の部分行列 $m \times n$ の転送を，本論文で実装した MPI ライブラリを用いて行い，1 回の転送が完了するまでの時間（レイテンシ）とバンド幅を計測した．なお，本論文の MPI の実装では，すべてのデータ転送は Rendezvous プロトコルによって行われる．

DIMMnet-2 は基板上の 2 枚の SO-DIMM に 8 Byte ずつアドレスを振り分けているため，SO-DIMM に対するアクセスの単位が 8 Byte 以下，ストライドが 16 の倍数の場合，アクセスが片方の SO-DIMM のみに集中し，性能が低下する．この性能低下を避けるために，DIMMnet-2 を用いた評価では $4096 \times z$ 行列を実際には $4096 \times (z+1)$ 行列として SO-DIMM 上に確保した．そのため，MPI の送受信関数に渡す派生データ型 `dataType` は，`MPI_Type_vector(m, n, z + 1, MPI_DOUBLE, &dataType)` で構築される．

表 3 評価環境
Table 3 Evaluation environment.

Node	CPU	Pentium4 2.6 GHz
	Chipset	VIA VT8751A
	Memory	PC-1600 DDR SDRAM 512 MByte ×1
		DIMMnet-2 ×1
Switch		Voltaire ISR6000
OS		RedHat8.0（Kernel 2.4.27）
Controller		gcc-3.3.6（-O3 -march=pentium4 -msse2）

5.2 評価結果

レイテンシの計測結果を**図 10** と**図 11** に，バンド幅の計測結果を**図 12** と**図 13** に示す．これらの図の Contig とは，SO-DIMM 上に連続して存在する $m \times n$ 個の `MPI_DOUBLE` 型を DIMMnet-2 で連続転送した場合の値である．

$m = 4096$，$z = 4096$ において，$n = 1$ の場合，レイテンシは約 258.2 μs，バンド幅は約 123.4 MByte/s となった．一方，$n = 16$ の場合，レイテンシは約 1058.5 μs，バンド幅は約 494.2 MByte/s となった．

$n = 1$，$m = 4096$，$z = 256$ における派生データ型通信のバンド幅は，連続データ転送バンド幅の約 33%と，ホストの CPU による Unpack 処理のバンド幅の連続コピー性能に対する割合（9〜14%）に比べ高いものになっている．ただし，ここで $z = 1024$ と $z = 4096$ のときの性能値は，$z = 256$ の性能値に比べ低い値となっている．これは，現在，DIMMnet-2 の SO-DIMM の Row アドレスが 4 KByte ごとに切り替わるようになっているため，ストライド間隔が 8 KByte を超える $z = 1024$ と $z = 4096$ ではアクセスごとに Row アドレスが切り替わるオーバヘッドが生じるためである．ただし，このオーバヘッドは SO-DIMM への Row アドレスと Bank アドレスの割当てを見直し，そして，SO-DIMM インタフェースを改良することで現在は 1 度に 1 つしかアクティブにできていない SO-DIMM 1 枚あたり 4 つある内部のバンクを複数アクティブにし，同時に使用することができるようになると大幅に削減できるものと考えている．

また，$n = 16$ のときのバンド幅に注目すると，その値は約 550 MByte/s（$z = 256$），約 500 MByte/s

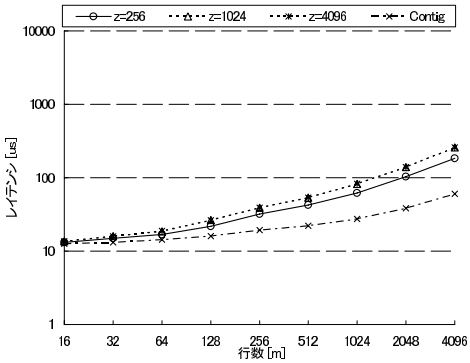


図 10 リモートベクトルアクセスによる部分行列転送レイテンシ（$n = 1$）
Fig. 10 Latency of submatrix transfer using remote vector access ($n = 1$).

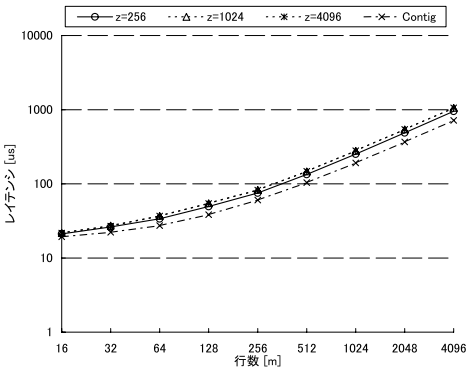


図 11 リモートベクトルアクセスによる部分行列転送レイテンシ（$n = 16$）
Fig. 11 Latency of submatrix transfer using remote vector access ($n = 16$).

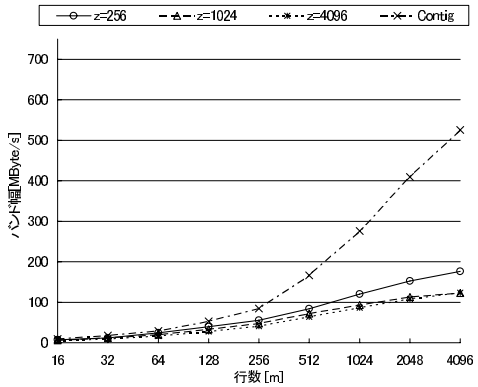


図 12 リモートベクトルアクセスによる部分行列転送バンド幅（$n = 1$）
Fig. 12 Bandwidth of submatrix transfer using remote vector access ($n = 1$).

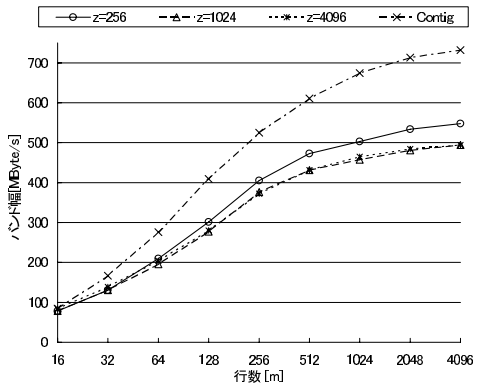


図 13 リモートベクトルアクセスによる部分行列転送バンド幅（$n = 16$）
Fig. 13 Bandwidth of submatrix transfer using remote vector access ($n = 16$).

（$z = 1024, 4096$）で飽和している．これについて，RTL シミュレーションにて調査を行った，$n = 16$ における SO-DIMM へのデータの読み書きを担当するブロック，Prefetch Unit と Write Unit のバンド幅を計測すると，たとえば $z = 4096$ において MTU に最も近い大きさのパケット☆を 1 つ処理する場合，その値は約 716 MByte/s と連続データ転送バンド幅と比較して十分な値であることが分かった．しかし，複数のパケットを連続して送信する場合，Prefetch Unit が SO-DIMM より読み出したデータを Window Controller が，InfiniBand とのインタフェース部に受け渡す際に，不必要に FIFO にデータを溜めてしまう実装上の問題があることが分かった．この問題により SO-DIMM からのデータ読み出しと InfiniBand ネットワークへのパケットの送出処理が並行して行われなくなり，バンド幅が約 500 MByte/s 程度で制限されてしまっている．

☆ データが 1920 Byte．ヘッダが 24 Byte.

5.3 ホストの CPU で Pack/Unpack 処理を行う場合との比較

DIMMnet-2 のリモートベクトルアクセスを用いる場合と，ホストの CPU で Pack/Unpack 処理を行う場合とで，派生データ型通信の性能を比較する．ここで本論文における MPI ライブラリは，DIMMnet-2 の SO-DIMM 上のデータのみを通信の対象としている．一方，ホストの CPU が Pack/Unpack 処理できるのは，ホストの主記憶上のデータである．そのため，ここではホストの CPU による Pack/Unpack 処理の性能と DIMMnet-2 の連続データ転送性能より，ホストの CPU で Pack/Unpack 処理を行った場合の派生データ型通信の性能を算出する．

まず，ホストの CPU で Pack/Unpack 処理を行う

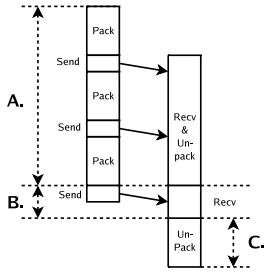


図 14 セグメント Pack/Unpack 処理
Fig. 14 Segment packing and unpacking.

表 4 セグメント数とレイテンシ（$m = 4096$，$n = 1$，$z = 4096$）
Table 4 Number of segments and latency ($m = 4096$, $n = 1$).

d	t_{pack}	$t_{transfer}$	t_{unpack}	$Latency$
2	481.83	38.38	721.98	1726.42
4	234.31	27.40	323.53	1295.37
8	124.43	22.19	156.13	1190.56
16	76.67	19.36	89.32	1371.40
32	50.07	16.03	50.59	1743.26

t_{send} は 2.24 で一定（単位：μs）

場合のレイテンシを求める．MPICH2 などの MPI ライブラリは，転送するデータをセグメントと呼ばれる固まりに分けて処理を行い，ある程度，Pack 処理とデータの送信を並行させている．ここでは，最も理想的な並行処理が行われた場合を想定する．理想的な並行処理が行われる場合，最後の 1 つのセグメントを除いたすべてのセグメントのデータ転送と Unpack 処理にかかる時間が，送信側の Pack 処理と送信関数の呼び出しに隠蔽される．この様子を**図 14** に示す．この場合，全体のレイテンシは，A：送信側における Pack 処理と送信関数の呼び出しにかかる時間，B：最後のセグメントを転送する時間，C：最後のセグメントを Unpack する時間の合計となる．$m \times n$ 部分行列を Pack/Unpack するのに必要な時間をそれぞれ $t_{pack}(m,n)$，$t_{unpack}(m,n)$，連続する行列の要素を x 個転送するのに必要な時間を $t_{transfer}(x)$，1 回の送信関数の呼び出しに必要な時間を t_{send} とおくと，$m \times n$ 部分行列を d 個のセグメントに分けて送信する場合のレイテンシ $Latency(m,n,d)$ は，次式で求めることができる．

$$\begin{aligned} &Latency(m,n,d) \\ &= t_{pack}(\frac{m}{d},n) \times d + t_{send} \times (d-1) \\ &\quad + t_{transfer}(\frac{mn}{d}) + t_{unpack}(\frac{m}{d},n) \end{aligned}$$

$m = 4096$，$n = 1$，$z = 4096$ の場合において，この式の各項の値を表 3 の環境で測定した結果を**表 4** に示す．ここで t_{pack} と t_{unpack} は，2 章の Pack/Unpack 処理のバンド幅と同じ手法で計測した値である．$t_{transfer}$ は，本論文で実装した MPI ライブラリを用いて連続データの転送を行うことで求めた．t_{send} は，本論文で実装した MPI ライブラリの `MPI_Isend` 関数の呼び出しにかかる時間である．

表 4 における最小のレイテンシを，$m = 4096$，$n = 1$，$z = 4096$ としたときの，ホストの CPU で Pack/Unpack 処理を行う際の通信全体のレイテンシと見なすと，その値は約 1190.6 μs（$d = 8$）である．同様に，$m = 4096$，$n = 16$，$z = 4096$ におけるホストの CPU で Pack/Unpack 処理を行う場合のレイテンシを求めると約 3828.3 μs（$d = 8$）となった．これにより，リモートベクトルアクセスを用いることで，$m = 4096$，$z = 4096$ においてレイテンシを約 78.3%（$n = 1$），約 72.4%（$n = 16$）削減できる可能性が示された．

次に，ホストの CPU で Pack/Unpack 処理を行う場合の通信バンド幅を求める．通信全体のバンド幅は，最もバンド幅が低い処理に律速される．今回の評価で行った転送パターンの場合，図 3〜図 6 よりそれは Unpack 処理であった．よって，図 5 と図 6 より，ホストの CPU で Pack/Unpack 処理を行う場合の通信全体のバンド幅は，$m = 4096$，$z = 4096$ において，それぞれ，約 23.7 MByte/s（$n = 1$），約 135.4 MByte/s（$n = 16$）と見なすことができる．これにより，リモートベクトルアクセスを用いることで，$m = 4096$，$z = 4096$ においてバンド幅を，それぞれ，約 5.2 倍（$n = 1$），約 3.6 倍（$n = 16$）向上させることができる可能性が示された．

6. まとめと今後の課題

本論文では，MPI 派生データ型通信を，ネットワークインタフェース上に設けたリモートベクトルアクセス機能を備えたベクトルアクセス機構を利用することで高速化する手法を提案した．この機構は，メモリ上の不連続領域への効率的なアクセスを行うことができる．また，RDMA による転送を行う場合に，データの送信・受信側双方のメモリアクセスパターンを，通信起動側がまとめて指定できる．そして，この機構を DIMMnet-2 に実装し，この機構を利用して派生データ型通信を行う MPI ライブラリの開発を行った．評価の結果，派生データ型を用いて部分行列を転送した場合に，この機構を利用すると，利用しない場合に比べ，約 78.3%のレイテンシの削減と，約 5.2 倍のバンド幅の向上が見込まれることが分かった．この結果により，

本提案手法の有効性を示すことができたと考える．

DIMMnet-2 のベクトルアクセス機構の持つプリフェッチ機能などを活用すると，1 ノードのあたり計算能力を向上させることが可能である[14)]．今後は，これらの機能と派生データ型通信を組み合わせたアプリケーションを開発し，実機上での評価を行っていく．

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日立情報通信エンジニアリング株式会社の今城氏，岩田氏，上嶋氏，東京農工大学の濱田氏，慶應義塾大学の渡邊氏，大塚氏に感謝いたします．

参 考 文 献

1) Gropp, W., Lusk, E. and Swider, D.: Improving the Performance of MPI Derived Datatypes, *Proc. 3rd MPI Developer's Conference*, pp.25–30 (1999).
2) Ross, R.B., Miller, N. and Gropp, W.: Implementing Fast and Reusable Datatype Processing, *Proc. 10th EuroPVM /MPI Conference*, pp.404–413 (2003).
3) Byna, S., Gropp, W., Sun, X-H. and Thakur, R.: Improving the Performance of MPI Derived Datatypes by Optimizing Memory-Access Cost, *Proc. IEEE International Conference on Cluster Computing*, pp.412–419 (2003).
4) MPICH2 home page. http://www-unix.mcs.anl.gov/mpi/mpich2/
5) Byna, S., Sun, X-H., Thakur, R. and Gropp, W.: Automatic Memory Optimizations for Improving MPI Derived Datatype Performance, *Proc. 13th European PVM/MPI Users' Group*, pp.238–246 (2006).
6) Cameron, K., Clarke, L.J. and Smith, A.G.: CRI/EPCC MPI for CRAY T3D, *Proc. 1st European Cray T3D Workshop* (1995).
7) Golebiewski, M., Ritzdorf, H., Traff, J. and Zimmermann, F.: The MPI/SX Implementation of MPI for NEC's SX-6 and Other NEC Platforms, *NEC Research & Development*, Vol.44, No.1, pp.69–74 (2003).
8) Träff, J.L., Hempel, R., Ritzdorf, H. and Zimmermann, F.: Flattening on the Fly: Efficient Handling of MPI Derived Datatypes, *Proc. 6th European PVM/MPI Users' Group*, pp.109–116 (1999).
9) Wu, J., Wyckoff, P. and Panda, D.: High Performance Implementation of MPI Derived Datatype Communication over InfiniBand, *Proc. 18th International Parallel and Distributed Processing Symposium*, p.14a (2004).
10) Santhanaraman, G., Wu, D. and Panda, D.K.: Zero-Copy MPI Derived Datatype Communication over InfiniBand, *Proc. 11th European PVM/MPI Users' Group*, pp.47–56 (2004).
11) Liu, J., Wu, J. and Panda, D.K.: High Performance RDMA-based MPI Implementation over InfiniBand, *International Journal of Parallel Programming*, Vol.32, No.3, pp.167–198 (2004).
12) 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：DIMMnet-2 における通信ライブラリ MPI-2 の実現，情報処理学会ハイパフォーマンスコンピューティング研究会，Vol.2006-HPC-105, pp.49–54 (2006).
13) Sur, S., Jin, H.-W., Chai, L. and Panda, D.K.: RDMA read based rendezvous protocol for MPI over InfiniBand: Design alternatives and benefits, *Proc. 11th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pp.32–39 (2006).
14) 田邊 昇，安藤 宏，箱崎博孝，土肥康孝，中條拓伯，天野英晴：プリフェッチ機能を有するメモリモジュールによる PC 上での間接参照の高速化，情報処理学会論文誌：コンピューティングシステム，Vol.46, No.SIG 12(ACS 11), pp.1–12 (2005).

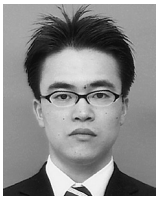
(平成 19 年 1 月 22 日受付)
(平成 19 年 5 月 7 日採録)

宮部 保雄

2007 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，日本電気株式会社勤務．在学中は，計算機アーキテクチャに関する研究に従事．

宮代 具隆

2007 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，ソニー株式会社勤務．在学中は，計算機アーキテクチャに関する研究に従事．



北村 聡（学生会員）

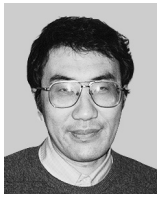
2005 年慶應義塾大学大学院理工学研究科開放環境科学専攻前期博士課程修了．現在，同後期博士課程に在学．計算機アーキテクチャに関する研究に従事．

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．

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院准教授．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）．



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．