InfiniBandを用いたHPC向け通信ライブラリにおける大規模並列実行を可能にする省メモリ化設計

山口 訓央[1,a)]　高木 将通[1,b)]　堀 敦史[1,c)]　石川 裕[1,d)]

概要：InfiniBand を用いたハイパフォーマンスコンピューティング向けの通信ライブラリにおいて，通信性能低下を最小限に抑えながら，100 万ノードでの並列実行を可能にするメモリ消費削減手法を考察する．対象とする通信ライブラリは，MPI ライブラリとその下位に位置する低レベル通信ライブラリである．また，NUMA ノード内を OpenMP で並列化し，NUMA ノード間を MPI で並列化することを想定する．これらの通信ライブラリでは，並列実行ノード数と NUMA ノード数が増加した際のノードあたりメモリ消費量の増加が課題となる．この課題を解決するため，通信コンテキストの総数を制限する手法，また複数の通信相手で一つの資源を共有する手法，また複数の MPI プロセスでオブジェクトを共有する手法を考察する．本手法は，1 ノードあたり 4 MPI プロセスとした場合，100 万ノードを用いた並列実行において，1 ノードあたりメモリ消費量を 1.07 GB に抑えることができる．

1. はじめに

ハイパフォーマンスコンピューティング（HPC）向けコンピュータの性能向上は, 主にノードあたり演算器数の増大とノード数の増大によってもたらされてきた．ここで，メニーコアアーキテクチャのプロセッサを用いたコンピュータに注目する．これらのコンピュータのノード数は増大を続けることが予想され，100 万ノードに到達することが予想される．例えば，京コンピュータは 82,944 ノード [1]，Sequoia は 98,304 ノード [2] からなる．また，メモリへのアクセス遅延が同一であるコア群である NUMA ノードのノードあたりの数が増加することが予想される．これは，チップ内配線遅延がスケールしなくなったため，メモリポート近傍のコア群でのローカリティの活用が必要になる一方で，多数の演算器へデータ供給を行うためメモリポート数が増加することによる．例えば，富士通 SPARC XIfx プロセッサを用いた FX100 システムは 2 個の NUMA ノードを持つ [3], [4].

ノード数と NUMA ノード数が多いシステムでは，システムソフトウェアのメモリ消費増加が課題となる．ここで，システムソフトウェアのメモリ消費のうち，通信ライブラリのメモリ消費に注目する．さらに，通信ライブラリとしては，MPI ライブラリと，その下位に位置しネットワークハードウェアドライバと MPI ライブラリの橋渡しをする低レベル通信ライブラリに注目する．また，プログラミングモデルとしては，OpenMP で NUMA ノード内を並列化し MPI で NUMA ノード間を並列化するハイブリッド並列化を想定する．ノード数が増大するとノードあたりメモリ消費が増大する理由は，1 MPI プロセスあたりメモリ消費量の中にアプリケーションの MPI プロセス総数に比例して増大する要素があるためである．ノードあたり NUMA ノード数が増大するとメモリ消費が増大する理由は，ノードあたり MPI プロセス数が増大し，多くの実装では MPI プロセス間でオブジェクトを共有する設計になっていないため，ノードあたりメモリ消費量が単純にノードあたり MPI プロセス数倍に増加するためである．このような課題に対し，個々メモリ消費要素に対する手法の提案は行われてきたが，メモリ消費が支配的となる要素全てに対する包括的考察はなされてこなかった.

そこで本論文では，通信ライブラリにおける，100 万ノードを用いる並列実行を通信性能低下を最小限に抑えながら可能にする手法の網羅的考察を行う．メモリ消費削減手法については，MPI ライブラリや LLC で管理する通信コンテキストについてメモリ割り当てを行う数を制限する手法，また複数のソースランクに対して１つの通信バッファやネットワークハードウェアの通信コンテキストを割り当てる手法，同一ノードで動作する複数の MPI プロセスでオブジェクトを共有する手法を考察する．本手法を解析モ

[1] 理化学研究所計算科学研究機構
[a)] norio.yamaguchi@riken.jp
[b)] masamichi.takagi@riken.jp
[c)] ahori@riken.jp
[d)] ishikawa@is.s.u-tokyo.ac.jp

デルで評価した結果，ノードあたり 4 MPI プロセスとした場合，100 万ノードでの並列実行において，ノードあたりメモリ消費量を 1.07 GB に抑えられることがわかった．

これ以降の論文構成は以下の通りである．2 節で課題を分析し，3 節でメモリ消費削減手法を説明し，4 節で評価を行い，5 節で関連文献を論じ，最後に 6 節でまとめる．

2. 課題分析

本節では，対象とするアーキテクチャとプログラミングモデルを説明したのち，メモリ消費の分析を行い課題を明らかにする．

2.1 対象アーキテクチャ

まず，対象とするシステムのハードウェアとソフトウェアのアーキテクチャを説明する．またシステムは 2^{20} ノードを備える．各ノードは InfiniBand (IB) により接続されている．IB を選択した理由は，多くの HPC 向けコンピュータが IB を用いているためである．例えば，2015 年 7 月時点で，TOP500[5] のシステムのうち 51.4%が IB を用いている．また，IB は，Tofu-2[6]，Omni-Path[7] といった他のネットワークハードウェアと，メモリ消費の側面で多くの共通点を持つため，本稿での議論はこれらのネットワークハードウェアにも応用できるためである．MPI 通信ライブラリは MPICH version 3.2b3[8] を使用し，低レベル通信ライブラリとしては我々が開発している Low-Level Communication Library (LLC)[9] を使用する．

2.2 対象プログラミングモデル

対象システムでは，OpenMP などのスレッド並列化手法を用いて NUMA ノード内のコア単位で並列化し，MPI を用いて NUMA ノード単位で並列化すると想定する．この想定の妥当性を説明する．先に説明した NUMA ノードの導入により，システムのメモリアクセス遅延観点の階層は，NUMA ノード，ノード，システム全体の 3 階層となる．OpenMP などのスレッド並列化手法は単一階層に対する並列化を対象とするため，これに第一の階層内の並列化を担当させ，それ以外の階層の並列化を MPI に行わせる方法が一般的であると考えられるためである．

2.3 通信ライブラリのメモリ消費分析

本節では，MPICH と LLC のベースラインとなる実装について，機能ごとのメモリ消費の分析を行う．分析対象シナリオは，ユーザが `MPI_COMM_WORLD` と同じサイズのコミュニケータを `MPI_Comm_split()` を用いて c 個作成し，さらに一対一通信を用いて全てのプロセスが他の全てのプロセスと通信を行うプログラムを，ノードあたり m MPI プロセス，全体で n MPI プロセスで実行した場合である（c，m，n はパラメタ）．MPI の集団通信，グループ操作

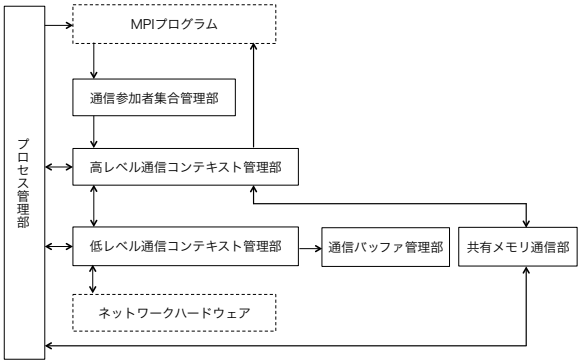

図 1 通信ライブラリのメモリ消費観点での構成

や，トポロジ操作によるメモリ消費は，一般的な使われ方においては支配的にならないため考慮しない．分析においては，MPICH と LLC だけでなく，IB のユーザレベルライブラリが消費するメモリも考慮するが，カーネルモジュールなどカーネルコードが消費するメモリは考慮しない．また，データ，BSS，ヒープ，anonymous mmap の各セグメントに存在するメモリだけでなく，テキスト，スタックの各セグメントに存在するメモリも考慮する．分析においては，構造体のメインメモリ上での占有サイズ（「メモリ上のサイズ」と呼ぶ）を用いた．これは，プログラムで定義されたサイズに，ABI やアロケータによって加えられるオーバーヘッドを加えたものである．メモリ上のサイズは，Intel Xeon E5-2670 v2 プロセッサ，Mellanox Connect-IB を備えるノード 32 台からなるシステムでプログラムを実行することで計測した．

メモリ消費の分析は，メモリ消費の観点で通信ライブラリの機能を要素に分解し，それぞれの機能要素ごとに行う．図 1 を用いて，これらの機能要素について説明する．

通信参加者集合管理部　通信参加者集合管理部は，MPI プロセスに通信で用いる名前を付け，さらに互いに干渉しない通信が行える MPI プロセスの集合を管理する．また，MPI プログラムからの通信指示を受けて，この集合と名前の組から，当該ディスティネーションランクへの通信を担当する，次に説明する高レベル通信コンテキストを見つけ出し，通信を指示する．

高レベル通信コンテキスト管理部　高レベル通信コンテキスト管理部は，MPICH および LLC 内部で MPI-API に近いレベルの通信コンテキストを管理する．通信コンテキストとは，あるデスティネーションランクとの通信の状態のことであり，プロトコルの状態や，バッファの位置などからなる．また，通信参加者集合管理部からの指示を受けて，当該ディスティネーションランクへの通信を担当する，次に説明する低レベル通信コンテキストを見つけ出し，通信を指示する．また，受信などのイベントを MPI プログラムに通知する．

低レベル通信コンテキスト管理部　低レベル通信コンテキスト管理部は，ネットワークハードウェアの通信コンテキ

ストを管理する．このコンテキストは，ハードウェアのコマンドキューやイベントキューなどからなる．また，高レベル通信コンテキスト管理部からの指示を受けて，ネットワークハードウェアに通信を指示する．また，受信などのイベントを高レベル通信コンテキスト管理部に通知する．

通信バッファ管理部　通信バッファ管理部は，アプリケーションが用意する通信バッファや，以下で説明する共有メモリ通信のバッファ以外の，通信ライブラリ内部で割り当てる通信バッファを管理する．通信バッファには，ディスティネーション側にあらかじめ用意した通信バッファに一度書き込む eager プロトコルと呼ばれる方法におけるバッファ，ハードウェアのチャネル型通信の受信コマンドに紐付けるバッファが含まれる．チャネル型通信とは，(1) ディスティネーションランク側であらかじめ受信バッファを割り当て，受信コマンドを HCA に挿入し，(2) ソースランク側で送信コマンドを HCA に挿入し通信を起動する，というステップで行われる通信のことである．また，低レベル通信コンテキスト管理部からの指示を受けて，バッファの割り当てを行う．

共有メモリ通信部　共有メモリ通信部は，通信バッファや，パケット到着通知機構などを備え，高レベル通信コンテキスト管理部からの指示を受けて，ノード内プロセス間の共有メモリを用いた通信を行う．また，受信などのイベントを高レベル通信コンテキスト管理部に通知する．

プロセス管理部　プロセス管理部は，MPI プロセスを起動し，また，MPI プロセス間の情報交換のための Key-Value Store (KVS) を MPI プロセスに提供する．

2.3.1 通信参加者集合管理部

MPICH では，通信参加者集合を構造体 `MPID_Comm` で実装し，MPI のコミュニケータの実装と，ライブラリ内部での通信参加者集合の管理に用いる．

図 2 を用いて，作成されるオブジェクトについて説明する．オブジェクトのそばに記されている数字は MPI プロセスあたりのメモリ上のサイズを示す．コミュニケータに関しては，まず，MPICH がデフォルトでユーザが使える `MPI_COMM_WORLD` を用意し，その後ユーザが MPI の関数を使ってコミュニケータを追加する．ライブラリ内部での通信参加者集合管理に関しては，`MPIR_ICOMM_WORLD` と呼ばれる全 MPI プロセスからなる集合と，`MPI_COMM_SELF` と呼ばれる自プロセスからなる集合を用意する．それぞれの `MPID_Comm` を作成する際には，以下のオブジェクトが作成され，`MPID_Comm` に関連づけられる．

通信コンテキストのポインタテーブル　MPICH は通信を行うプロセスのペアを Virtual Connection (VC) という概念で表現し，構造体 `MPIDI_VC_t` で実装する．`MPID_Comm` はランク番号で問い合わせると Virtual Connection を返す機能を提供し，エントリ数がコミュニケータのサイズであるテーブルで実装する．このテーブルは `MPID_Comm` の `vcrt` フィールドでポイントする．なお，`MPI_COMM_WORLD` と `MPIR_ICOMM_WORLD` は一つのテーブルを共有する．

集団通信用の `MPID_Comm` とランク変換テーブル　MPICH は集団通信を，MPI プロセスのノードごとの集合を作り，それぞれ代表プロセスを決定した上で，ノード内プロセス間の通信と代表プロセス間の通信の 2 段に分けて実行する．このために同一ノード上に存在するランクからなる `MPID_Comm` と，代表プロセスからなる `MPID_Comm` を作成し，`MPID_Comm` の `node_comm` フィールドと `node_roots_comm` フィールドでポイントする．また，ランク番号で問い合わせると，`node_comm` でのランク番号を返す機能を提供し，エントリ数がコミュニケータのサイズであるテーブルで実装する．このテーブルは `MPID_Comm` の `intranode_table` フィールドでポイントする．また，ランク番号で問い合わせると，そのプロセスが属すノードごとの集合の，さらにその代表プロセスの，`node_roots_comm` でのランク番号を返す機能を提供し，エントリ数がコミュニケータのサイズであるテーブルで実装する．このテーブルは，`MPID_Comm` の `internode_table` フィールドでポイントする．

LLC は，通信を行うプロセスのペアを bundle of channels (BoC) という概念で表現し，構造体 `LLC_channels_t` で実装する．LLC においても MPICH と対応する通信参加者集合を管理し，構造体 `LLCD_comm_t` で実装する．この構造体はランク番号で問い合わせると BoC を返す機能を提供し，テーブルで実装する．

以上を合計して，通信参加者集合管理部によるノードあたりのメモリ消費量は以下のようになる．

$$((16c+32)m+\frac{8c+16}{m})n+(8c+16)m^2+(848c+2256)m+408c+816 \quad (1)$$

2.3.2 高レベル通信コンテキスト管理部

MPICH は高レベル通信コンテキストを VC で表現し，構造体 `MPIDI_VC_t` で実装する．. MPICH は初期化時に MPI プロセス一つにつき n 個の VC を用意する．

構造体自身のメモリ上のサイズは MPI プロセスあたり $480n+16$ バイトである．さらに VC 一つにつき，パケットヘッダを一時的に記憶するための領域を確保する（メモリ上のサイズは MPI プロセスあたり $64(n-1)$）．

LLC は高レベル通信コンテキストを BoC という概念で表現し，構造体 `LLC_channels_t` で実装する．これはネットワークハードウェアに近いレベルの通信コンテキストへのポインタ，eager プロトコルのバッファ，通信コマンドのキューなどを記録管理する．LLC は初期化時に MPI プロセス一つにつき n 個の BoC を用意する．構造体のメモリ上のサイズは $396n$ バイトである．

また，LLC は同時に起動されたプロセス群のランク番号と BoC を対応づけるテーブルを持つ（メモリ上のサイズは MPI プロセスあたり $8n+16$）．

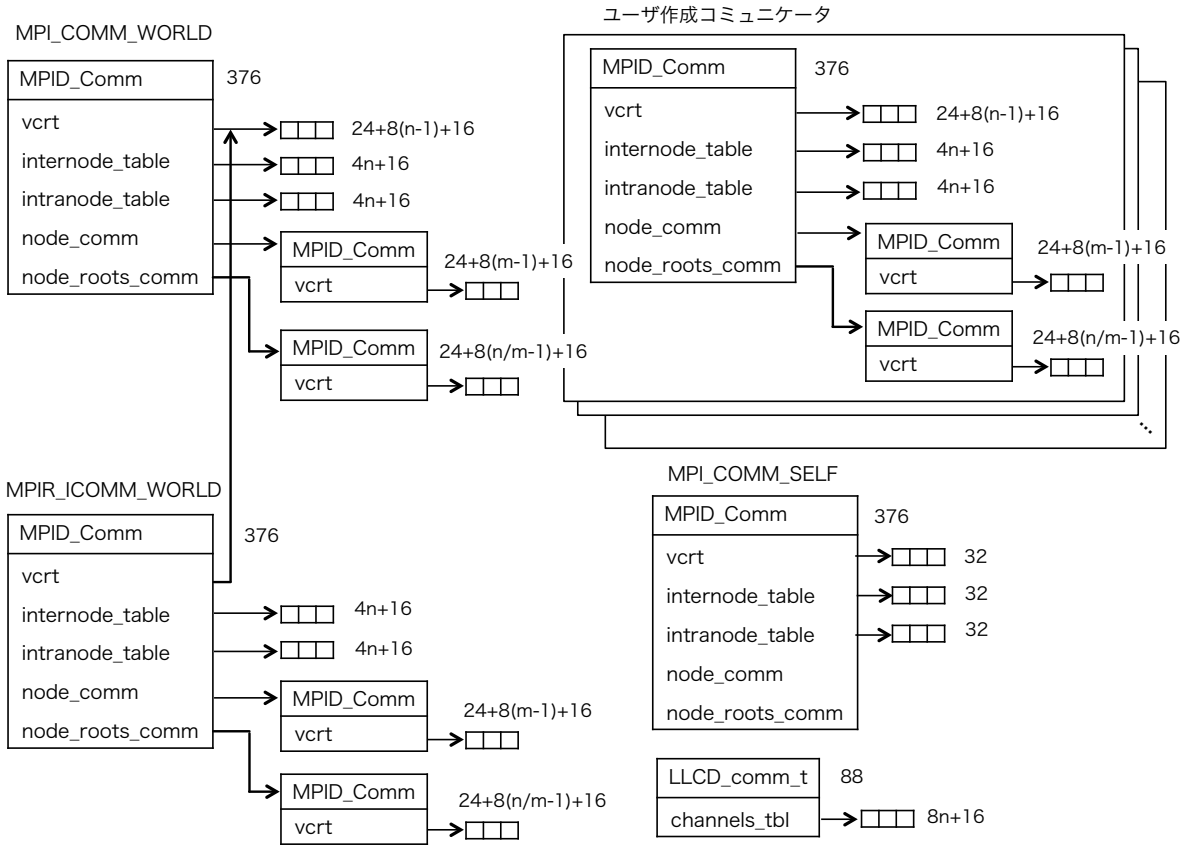

図 2 通信参加者集合管理部が作成するオブジェクト

以上を合計して，高レベル通信コンテキスト管理部によるノードあたりのメモリ消費量は以下のようになる．

$$(480n + 16 + 64(n-1) + 396n + 8n + 16)m \qquad (2)$$

2.3.3 低レベル通信コンテキスト管理部

IB は低レベルの通信コンテキストを Queue Pair (QP) という概念で表現する．LLC では Reliable Connection(RC) と呼ぶ, 信頼性がトランスポートレイヤで保証され, RDMA 通信を行うことができるトランスポートをデータ通信に用いる．対応する QP を RC-QP と呼ぶ．一つの RC-QP は特定の RC-QP とのみ通信が行える．LLC は 1 MPI プロセスにつき，RC-QP を $(n-m)$ 個用意する．IB はパケット到着及びコマンド結果をイベントで通知し，それらイベントのキューを Completion Queue (CQ) という概念で表現する．前者の CQ を受信 CQ，後者の CQ を結果 CQ と呼ぶ．QP には，Receive Queue (RQ) と呼ばれる受信コマンドキュー，Send Queue (SQ) と呼ばれる送信コマンドキュー，受信 CQ，結果 CQ を関連づけて用いる．さらに，送受信の際は，RQ あるいは SQ に Work Request (WR) と呼ぶコマンドを挿入する．RC-QP に対する RQ，SQ，受信 CQ，結果 CQ のエントリ数をそれぞれ 272，256，4096，4096 とする．なお，複数の RC-QP を作成する際は，受信 CQ，結果 CQ は一つずつ作成し，これを共有する．メモリ上のサイズは MPI プロセスあたり $672 + 31152(n-m)$ バイトである．

LLC では，フロー制御をデータ通信用 QP を流用して行う．そのため，制御通信のコンテキストのために，追加でメモリを消費する必要はない．また，ランク番号と RC-QP を対応づけるテーブルを持つ．メモリ上のサイズは MPI プロセスあたり $8n + 16$ バイトである．

以上を合計して，低レベル通信コンテキストが消費するノードあたりのメモリ量は以下のようになる．

$$(672 + 31152(n-m) + 8n + 16)m \qquad (3)$$

2.3.4 通信バッファ管理部

eager 通信のバッファとして, 1 エントリが 4KB でエントリ数 16 のリングバッファを，1 MPI プロセスごとに，ノード間通信が必要な MPI プロセス数分用意する．また，リングバッファ管理用の構造体をリングバッファ数分用意する．メモリ上のサイズは MPI プロセスあたり $1360(n-m)+16$ である．また，IB で送受信する際には，アドレス領域を HCA に登録する必要があり，1 登録あたり 128 B 消費する．さらに登録をキャッシュして登録の実効コストを低減させる．キャッシュは 4 ビットを radix とする radix-tree で実装する．また，登録数に 65536 の上限を設ける．これらのメモリ消費量は最大 16078021 と見積もる．また，送信の際にはパケット構築のためのメモリ領域を HCA による DMA の完了を確認するまで確保し続ける必要がある．

このメモリ消費は複数の送信について重なっていき，完了していない送信についての和まで増加する．この消費量は327680と見積もる．

以上を合計して，通信バッファ管理部によるノードあたりの消費メモリ量は以下のようになる．

$$(1360(n-m)+16+4096\times16(n-m)+ \\ 16078021+327680)m \tag{4}$$

2.3.5 共有メモリ通信部

MPICHは，整数iで問い合わせると同一ノード上に存在する第i番目のMPIプロセスのランク番号を返す機能，ランク番号で問い合わせるとiを返す機能，整数jで問い合わせると同一ノード上に存在しない第j番目のMPIプロセスのランク番号を返す機能を提供し，全てテーブルで実装する．これらメモリ上のサイズはMPIプロセスあたり$4n+16+4n+16+4(n-m)+20$である．

また，プロセスごとの受信キュー，データ到着イベントキュー，バッファプールを持つ．　さらに，ランク番号で問い合わせると対応する受信キューのポインタを返す機能，ランク番号で問い合わせるとバッファのフリーリストのポインタを返す機能を提供し，テーブルで実装する．これらのメモリ上のサイズはMPIプロセスあたり$8n+16+8n+16$である．さらに，シーケンス番号をMPIプロセスごとに管理し，これをテーブルで実装する．メモリ上のサイズはMPIプロセスあたり$2n+16$である．

共有メモリ通信部によるノードあたりのメモリ消費量は以下のようになる．

$$30mn+65716m^2+4129094m+2256 \tag{5}$$

2.3.6 プロセス管理部

MPIプロセスは，ランク番号で順序付けられるIBを用いたリングネットワークを作成するため，プロセス管理部が提供するKVSを用いて隣接ノードのネットワークアドレスを交換する．次に，作成したリングネットワークを用いて通信相手のRC-QPのネットワークアドレスを取得する．またこれ以外に，共有メモリ情報，ホスト名，MPICHが割り振ったランク番号とLLCが割り振ったランク番号との対応関係の情報を交換する．

図3を用いてKVS部分のメモリ消費を説明する．MPICHでは`mpiexec`コマンドがターゲットノードに`hydra_pmi_proxy`と呼ばれるプロセスを起動し，さらに`hydra_pmi_proxy`がMPIプロセスを`fork()`および`exec()`により生成する．そして，`mpiexec`がルート，各MPIプロセスがリーフとなる木構造を構成し，この木を使ってKVS機能を提供する．リーフがプットを行うと，値はリーフのプット用キャッシュに格納される．同期操作が行われるとこれらの値はキャッシュから取り除かれ，ルートに送られる．さらにルートから全リーフへブロードキャストが行われ，リーフのゲット用キャッシュに格納される．リーフがゲットを行うと，値はゲット用キャッシュあるいはルートから取得される．なお，リングネットワークの消費メモリは，第2.3.3節で説明する低レベル通信コンテキストに含める．

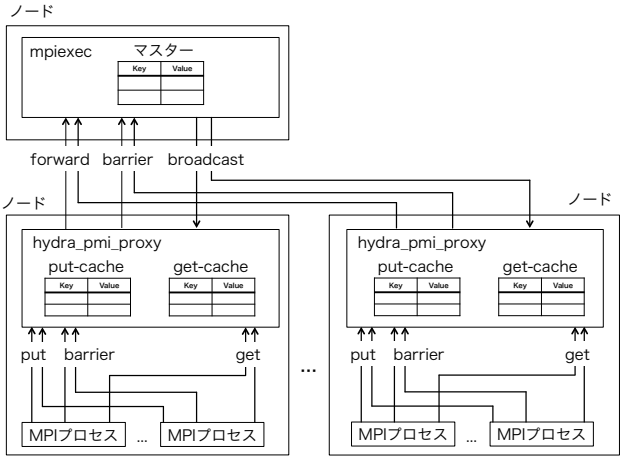

図 3 Key-Value Store の put/get のステップと作成されるオブジェクト

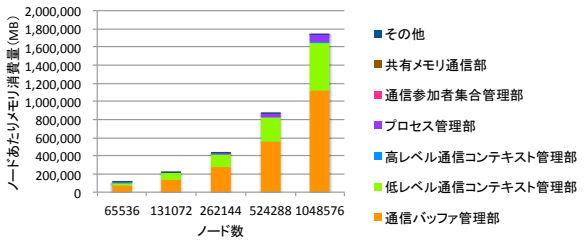

図 4 ベースライン実装の通信ライブラリのノードあたりメモリ消費量

プロセス管理部によるノードあたりのメモリ消費量は以下のようになる．

$$179384+1096\times(14n+2n/m)+ \\ 32272+(64+8)(14n+2n/m) \tag{6}$$

2.3.7 その他

メモリ割り当てのログや，ソースの解析から分類ができなかったメモリ消費量は，機能を変更しても変わらないベースラインとして計上する．このノードあたりの値は，通信ライブラリの初期化直後のメモリ消費の値や，実行時のプログラムの各セクションの大きさから以下のように推測した．

$$65540m^2+13067907m \tag{7}$$

2.4 議論

パラメタについて，$m=4, c=10$とする．図4にノードあたり4 MPIプロセスの場合の，ベースライン実装の通信ライブラリのノードあたりメモリ消費量を示す．2^{20}ノードを用いた並列実行におけるメモリ消費は1.74 TBと

なるので，通信ライブラリはこのような並列実行を実現できないと言える．

パラメタのうち，MPI プロセス数 n の値が他に比べて大きいため，n 比例項が支配的となる．そこで，ノードあたり 4 MPI プロセス，2^{20} ノードの場合について，機能ごとのメモリ消費量の n 比例項と値を表 1 の第 2 列と第 3 列に示す．「その他」を除く全ての機能が n 比例項を持ち，いずれも数 GB 以上のメモリを消費しているため，これらの n 比例項を削減する必要があることがわかる．

3. 設計

この節では，第 2.4 節で明らかになった課題に対する解決策を考察する．すなわち，支配的なメモリ消費量を持つ機能に対しメモリ消費削減手法を考察する．

3.1 通信参加者集合管理部

集団通信用の `MPID_Comm` とランク変換テーブルについては，ランク番号で問い合わせると `node_comm` でのランク番号を返す機能の実装を，テーブルによる実装からハッシュ関数で振り分ける m 個のリストによる実装に変更する．これらのリストの長さは短いため，この変更によるオーバーヘッドはハッシュ関数適用とロード命令一回分とキー比較分で小さいと考えられる．また，同一ノードに存在する MPI プロセス間で内容が同一であるためノードで一つのみ作成し共有する．さらに，ランク番号と `node_roots_comm` でのランク番号の変換機能のテーブル，同じ理由でノードで一つのみ作成し共有する．通信コンテキストのポインタテーブルについては，第 3.2 節で説明するようにアクティブな通信コンテキストの総数を γ に制限することに合わせて，テーブルによる実装からハッシュ関数で振り分ける γ 個のリストによる実装に変更する．これらのリストの長さは短いため，この変更によるオーバーヘッドはハッシュ関数適用とロード命令一回分とキー比較分で小さいと考えられる．このメモリ消費削減手法により，式 1 は以下で置き換えられる．

$$(4c+8)n+((40\gamma+856)c+104\gamma+2208)m+ \\ (448+20\gamma)c+40\gamma+960 \tag{8}$$

3.2 高レベル通信コンテキスト管理部

通信コンテキストは初期化時ではなくオンデマンドで作成し，またアクティブな通信コンテキスト数を γ 個に制限する（γ はパラメタ）．この制限は，アクティブな通信コンテキストがノードあたり γ 個を超えた場合最も未来に使用されると予想される通信コンテキストを削除することで行う．この手法は通信コンテキスト削除と再作成の際にオーバーヘッドを生じるため，ある MPI プロセスから見たデスティネーションランクの集合のサイズが大きいとき，また集合の時間変化が大きい際にオーバーヘッドが大きくなる．多くの MPI プログラムでは多対多の通信は頻繁には行われない．また一対多や多対一の通信については MPI の集団通信を用いることができる．このため，ある MPI プロセスから見たディスティネーションランク数は多くの時間窓で $\log n$ のオーダ以下になると考えられる．また γ は 1024 前後を想定するため，この手法によるオーバーヘッドは小さいと考えられる．

この手法により，式 2 は以下に置き換えられる．

$$480\gamma+16+64(\gamma-1)+ \\ (380+16)\gamma+8n+16 \tag{9}$$

3.3 低レベル通信コンテキスト管理部

IB は，RC の他に Dynamically Connected (DC) と呼ばれるトランスポートを持つ．DC は，RC に比較してメモリ消費を抑えることができるが，通信パターンによっては性能が大幅に低下することがある．このため，性能低下を最小限にしつつメモリ消費を抑えるような，RC-QP 数と DC-QP 数の組み合わせを見つける．この手法として，著者らの論文 [10] で説明されている手法を用いる．以下ではこの手法を説明する．

RC トランスポートは，チャネル型通信，RDMA 通信が行える．チャネル型通信は，(1) ディスティネーションランク側であらかじめ受信バッファを割り当て，WR に受信バッファを関連付け，QP の RQ に WR を挿入し，(2) ソースランク側で QP の SQ に WR を挿入して通信を起動する，というステップで行われる．RC トランスポートを用いた場合，一つの RC-QP は特定の RC-QP としか通信が行えないため，MPI プロセス 1 個あたり，$n-m$ 個の RC-QP を用意せねばならず，メモリ消費が大きくなる．また，アクティブな RC-QP の数が多くなると性能が低下する [11]．DC トランスポートは，チャネル型通信，RDMA 通信が行える．また，ノードあたり 1 個の DC-QP を用意するだけで，任意のランクペア間の通信が可能になるため，メモリ消費を抑制できる．しかし，1 つの DC-QP を用いて，短い時間の間に多数のディスティネーションランクに送信を行うと性能が低下する [12]．また，短い時間の間に多数のソースランクが 1 つの DC-QP に送信を行うと性能が低下する [13].

このため，以下のようなステップで DC-QP，RC-QP の数を調整することで，性能低下を最低限に抑えながら，メモリ消費を抑制する．起動時に，データ通信用にノードあたり 2 個の DC-QP を用意し，後述する割り当てが行われるまで，各 MPI プロセスはこの DC-QP を用いてチャネル型通信を行う．2 個の DC-QP を用意するのは，先に述べた DC-QP の性能低下問題に対処するためである．DC-QP 1 個あたりの RQ，SQ，受信イベントキュー，結果イベントキューのエントリ数，受信用 WR の数をそれぞれ 2048,

表 1　機能ごとのノードあたりメモリ消費量

機能	ベースライン実装の n 比例項	ベースライン実装のメモリ消費量	削減手法適用時の n 比例項	削減手法適用時のメモリ消費量
通信バッファ管理部	$66896mn$	1.12 TB	-	151 MB
低レベル通信コンテキスト管理部	$31160mn$	523 GB	$24n$	106 MB
高レベル通信コンテキスト管理部	$948mn$	15.9 GB	$8n$	34.5 MB
通信方法とバッファの動的切替部	-	-	$8mn$	407 MB
通信参加者集合管理部	$((16c+32)m+\frac{8c+16}{m})n$	3.32 GB	$(4c+8)n$	204 MB
共有メモリ通信部	$30mn$	521 MB	$6mn$	118 MB
プロセス管理部	$(\frac{2336}{m}+16352)n$	71.0 GB	-	216 KB
その他	-	53.3 MB	-	53.3 MB
合計	$((16c+99066)m+$ $\frac{8c+2352}{m}+16352)n$	1.74 TB	$(14m+4c+40)n$	1.07 GB

512，40000，2048，2048 とする．このメモリ消費はノードあたり $243520+164624$ である．ディスティネーションのデータ通信用 DC-QP のネットワークアドレスはハッシュ関数で振り分けられる γ 個のリストで記憶する．このメモリ消費はノードあたり $8\gamma+(24+4+8)\gamma+16$ である．γ 個でよいのは，DC-QP を用いた通信を行う可能性があるのはアクティブな高レベル通信コンテキストに対応するディスティネーションランクのみであるためである．さらに，ディスティネーション DC-QP ごとに送信時に使用するパケットヘッダを記録し再利用する．このメモリ消費はノードあたり $(72+8)\gamma$ である．

また，クレジットを用いたフロー制御用に DC-QP を用意する．DC-QP 1 個あたりの RQ，SQ，受信イベントキュー，結果イベントキューのエントリ数，受信用 WR の数をそれぞれ 2048，512，40000，2048，2048 とする．このメモリ消費はノードあたり 168736 である．また，それぞれの WR に対して 256 B の受信用バッファを割り当て関連付ける．このメモリ消費はノードあたり 524304 である．また，ディスティネーションのフロー制御用 DC-QP のネットワークアドレスをハッシュ関数で振り分けられる γ 個のリストで記憶する．このメモリ消費はノードあたり $8\gamma+(24+4+8)\gamma+16$ である．

実行時に，ソースランクごとに通信方法を，DC-QP を用いたチャネル型通信，RC-QP を用いた RDMA 通信，DC-QP を用いた RDMA 通信の 3 種のいずれかに切り替える．これは以下のステップで実現する．まず，実行時にソースランクごとの通信回数の履歴を取っておく．そして，一定時間ごとに，ある時間窓に対する履歴に対して，遅延を最小にするような通信方法の割り当てを見つけ，第 3.5 で説明する通信方法の切り替え用の DC-QP を用いたハンドシェイクにより通信方法を切り替える．この割り当ての計算と，通信方法の切り替えの 1 回のオーバーヘッドは大きいが，切り替え判断の間隔を秒オーダに設定することで抑える．このときに RC-QP を用いた RDMA 通信が選択された場合，RC-QP を新たに割り当てる．なお，以降の切り替えで同一ソースランクの通信方法が DC を用いた RDMA 通信や DC を用いたチャネル型通信に切り替えられた場合，RC-QP は不要になるが，破棄の時間コストが大きいため破棄を行わず，保持しておいて再利用を行う．この RC-QP の数はノードあたり最大 128 個に制限する．RC-QP のメモリ消費は，ノードあたり最大 $672+31152\times128$ バイトである．RC-QP を用いた RDMA 通信または DC-QP を用いた RDMA 通信が割り当てられるソースランク数を制限したことによるオーバーヘッドについて考察する．多くの MPI プログラムでは多対多の通信は頻繁には行われない．また一対多や多対一の通信については MPI の集団通信を用いることができる．このため，ある MPI プロセスから見たソースランク数は多くの時間窓で $\log n$ のオーダ以下になると考えられる．このため，それらのソースランク全てに RC-QP を用いた RDMA 通信あるいは DC-QP を用いた RDMA 通信が割り当てられることが期待される．そのため，このオーバーヘッドは小さいと考えられる．

また，ランク番号で問い合わせると RC-QP を返す機能を提供し，ハッシュ関数で振り分ける 128 個のリストで実装する．メモリ上のサイズはノードあたり $8\times128+(8+4+8)\times128+16$ バイトである．これらのリストの長さは短いため，この変更によるオーバーヘッドはハッシュ関数適用とロード命令一回分とキーの比較分で小さいと考えられる．また，DC-QP を用いた RDMA 通信が選択された場合には，QP はチャネル型通信で用いている DC-QP を流用し，DC-QP の追加は行わない．また，高レベル通信コンテキストの作成は，通信方法の切り替え用の DC-QP を用いて行う．このために，全プロセスの通信方法の切り替え用の DC-QP のネットワークアドレスを記憶する．これはテーブルで実装し，メモリ消費はノードあたり $24n+16$ である．

この手法により，式 3 は以下に置き換えられる．

$$24n+168\gamma+5092928 \tag{10}$$

3.4 通信バッファ管理部

受信回数が多いソースランクに対してのみ独立した通信バッファを割り当て，他のランクには共有バッファを割り当てることで，通信バッファのメモリ消費を削減する．

まず，起動時には，DC-QP のチャネル型通信を用いて通信を行う．チャネル型通信では受信側であらかじめ DC-QP の RQ に WR を挿入しておく必要がある．2 つの DC-QP のそれぞれに 2048 個の WR を挿入し，それぞれの WR に対して，4 KB の受信用バッファを割り当て，WR に関連付ける．メモリ上のサイズはノードあたり 16785408 である．また，同一ノードに存在する複数プロセスで受信用バッファを共有することで，ノードあたり MPI プロセス数増加に伴うメモリ消費増加を抑制する．

1 ノードあたり，1 エントリが 4KB で 16 エントリのリングバッファを 1024 個用意し，第 3.3 節に説明した通信方法の切り替えのときに RC-QP あるいは DC-QP に割り当てていく．割り当てのための情報交換は，通信方法の切り替えに用いる DC-QP を流用する．また，同一ノードに存在する複数プロセスでリングバッファを共有することで，ノードあたり MPI プロセス数増加に伴うメモリ消費増加を抑制する．なお，リングバッファのエントリのサイズは通信性能に影響を与えるため，アプリケーションを用いた評価により設定する必要が有る．

この手法により，式 4 は以下に置き換えられる．

$$\begin{aligned}&16785408 + 1360\times1024 + 16+\\&65536\times1024 + (16078021 + 327680)m\end{aligned} \tag{11}$$

3.5 通信方法とバッファの動的切替部

第 3.3 節および第 3.4 節で説明した通信方法とバッファの動的切り替え機能のメモリ消費を説明する．このメモリ領域は，通信履歴と，ソースランクごとの最適な通信方法の選択に用いるテーブルに分けられる．通信履歴は，ソースランクごと，また一定期間ごとの受信回数を 64 ビット整数で記憶する．時間軸方向のエントリ数は履歴のサイズが 64 MB で頭打ちになるように，MPI プロセスあたり以下に設定する．

$$\begin{cases}1024, & \text{if } p \le 2^{13}\\ \frac{1024}{2^{\lceil \log_2 p \rceil - 13}}, & \text{if } p > 2^{13}\end{cases} \tag{12}$$

通信方法の選択は一時的なテーブルを用い，そのサイズは MPI プロセスあたり $(8\times1024 + (4+4+8)\times1024 + 16)\times(3+13) + 8n + 16$ バイトである．

切り替えに伴う情報交換および同期は別に設けた 1 つの DC-QP を用いたチャネル型通信で行う．この DC-QP 1 個あたりの RQ，SQ，受信イベントキュー，結果イベントキューのエントリ数，受信用 WR の数をそれぞれ 2048，512，2048，512，2048 とする．これらのメモリ消費量は MPI プロセスあたり 165008 である．また，それぞれの WR に対して，256 B の受信用バッファを割り当て，WR に関連付ける．このメモリ消費量は MPI プロセスあたり 524304 である．

通信方法およびバッファ切り替えのメモリ消費量は以下となる．

$$\begin{aligned}&\begin{cases}8\times1024mn, & \text{if } p \le 2^{13}\\ 67108864m, & \text{if } p > 2^{13}\end{cases} +\\&((8\times1024 + (4+4+8)\times1024 + 16)\times(3+13)+\\&8n + 16 + 165008 + 524304)m\end{aligned} \tag{13}$$

3.6 共有メモリ通信部

整数 i で問い合わせると同一ノード上に存在する第 i 番目の MPI プロセスのランク番号を返す機能は m エントリのテーブルで実装する．また，ランク番号で問い合わせると i を返す機能は，ハッシュ関数で振り分けられる m 個のリストで実装する．これらのリストの長さは短いため，この変更によるオーバーヘッドはハッシュ関数適用とロード命令一回分とキーの比較分で小さいと考えられる．ランク番号で問い合わせると対応する受信キューのポインタを返す機能，ランク番号で問い合わせるとバッファのフリーリストのポインタを返す機能は，ランク番号で問い合わせると i を返す機能を用いて，m エントリのテーブルで実装する．この変更によるオーバーヘッドは，1 回のテーブル引き分で小さいと考えられる．

この手法により，式 5 は以下に置き換えられる．

$$6mn + 65756m^2 + 4129094m + 2256 \tag{14}$$

3.7 プロセス管理部

KVS の Key-Value ペアの管理方法を，ルートに集約する方法から，ハッシュ関数で決定されるノードで管理する方法に変更する．なお，この変更により，KVS の get の遅延が増大するため，この方法については評価して採用可否を判断する必要がある．

また，ベースライン実装のルートは Key-Value ペア一つあたり固定サイズ（Key-Value ペアの最大長）のメモリを割り当てていたが，Key-Value ペアのサイズ分だけ割り当てる方法に変更する．Key-Value ペアの最大長は n に依存するため，計測値を元に 64 と算出し，この値をメモリ消費量計算に用いる．なお，KVS を用いて交換すべき情報のうち，リングのための RC-QP のネットワークアドレスは，通信方法の切り替えに用いる DC-QP のネットワークアドレスに変わる．この手法により式 6 は以下に置き換えられる．

$$179384 + (64+8)(13m + 2) + 32272 \tag{15}$$

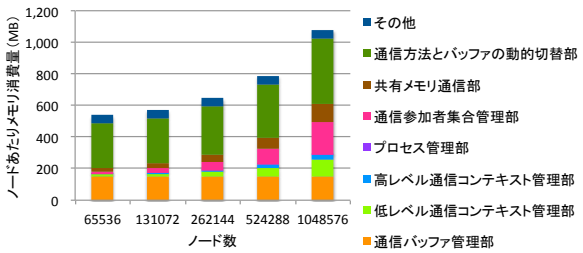

図 5 メモリ消費削減手法を適用した場合の通信ライブラリのノードあたりのメモリ消費量

4. 評価

パラメタについて，$\gamma = 1024, c = 10$ とする．なお，MPI プロセス数を n，ノードあたり MPI プロセス数を m で表す．図 5 にノードあたり 4 MPI プロセスの場合の，メモリ消費削減手法を適用した場合の通信ライブラリのノードあたりのメモリ消費量を示す．また，ノードあたり 4 MPI プロセス，2^{20} ノードの場合について，機能ごとのメモリ消費量の MPI プロセス数 (n) 比例項と値を表 1 に示す．表からわかるように，メモリ消費が支配的となる機能すべてにおいて n 比例項を削減することができ，結果としてメモリ消費量を大幅に削減できている．また，他機能のメモリ消費を削減するため追加した通信方法とバッファの動的切替機能のオーバーヘッドは，n 比例項で見た場合 $8mn$ となり，この機能によってもたらされた他機能の削減幅を大きく下回る．

5. 関連研究

Liu らは，MVAPICH の実装において，受信バッファと低レベル通信コンテキストのメモリ消費を削減する手法を提案した [14]．起動直後はすべてのランクで共通の受信バッファを用い，基準を満たした最初の s 個のソースランクに対し独立した受信バッファと低レベル通信コンテキストを割り当てていく（s はパラメタ）．こうすることで，これらによるメモリ消費を抑制する．彼らは，大規模並列実行でのメモリ消費量を評価していない．

住元らは，「京」スーパーコンピュータ向け MPI ライブラリにおいて，受信バッファのメモリ消費を削減する手法を提案した [15]．この手法では，通信バッファについてサイズの大きなバッファの総数を制限する．具体的には，起動直後はすべてのソースランクに対して，2 KB の独立したバッファを用いて通信を行う．そして，基準を満たした最初の s 個のソースランクに対し 32KB〜1MB の独立したバッファを割り当てる（s はパラメタ）．バッファについては，彼らの手法は本論文で考察している手法と同じである．しかし，彼らの手法では，通信参加者集合管理や通信コンテキスト管理のためのメモリ消費は削減しない．彼らの手法では，ノードあたり 1 プロセス，9,216 ノードにおいてノードあたりメモリ消費は 1.52 GB であり，本論文で考察した手法では 195 MB である．この差の最大の原因は，彼らの手法では 1 つの通信バッファのサイズが最大 1MB で，本論文で考察した手法の 64KB より大きいことである．

Matthew らは，IB を用いた MPI ライブラリにおいて，低レベルコンテキストのメモリ消費を削減する手法を設計評価した [16]．IB において初期に導入された Reliable Connection (RC) トランスポートを用いて通信を行う場合，ソースランク側に，デスティネーションランク 1 つにつき 1 つの QP を必要とするが，2007 年に製品導入された eXtended Reliable Connection (XRC) トランスポートを用いて通信を行う場合は，ソースランク側に，デスティネーションランクが存在するノード 1 つにつき 1 つの QP を用意すればよい．XRC を用いて MPI ライブラリを実装することで，ノードあたり MPI プロセス数が増大した際の，Queue Pair 構造体によるメモリ消費増大を抑制する．彼らは，低レベル通信コンテキストのメモリ消費の評価は行っているが，主要な要素全てを含んだ評価を行っていない．

Pavan らは，100 万プロセッサを用いた並列実行を行う際に，メモリ消費の点で阻害要因となる MPI ライブラリの機能についてメモリ消費削減手法を提案した [17]．ランク番号で問い合わせるとプロセッサ ID を返す機能について，テーブルを用いる実装からランク番号を用いた計算による実装に置き換える方法を提案している．また，コミュニケータに保存している集団通信の最適化情報のサイズを削減する方法，低レベル通信コンテキストを通信が必要になったときに初めて作成する方法を提案している．彼らは，コミュニケータのメモリ消費の評価は行っているが，主要な要素全てを含んだ評価を行っていない．

吉永らは Intel Xeon などのホストプロセッサに，PCI バス経由で Intel Xeon Phi などのメニーコアプロセッサを接続した構成のシステムにおいて，MPI ライブラリのメモリ消費量を削減する手法を提案した [18]．彼らの手法では，メニーコアプロセッサ上で動作する複数 MPI プロセスからの通信要求を，パケットのカプセル化によって一つの通信コンテキストに関連付けることで，通信コンテキストと通信バッファのメモリ消費量をノードあたり MPI プロセス数分の一に削減する．この手法は本論文で考察している手法とは直交し，本論文で考察している手法にさらにこの手法を適用した場合，通信バッファ，通信コンテキスト，通信参加者集合については，本論文での評価の値から，さらにノードあたり MPI プロセス数分の一に削減されることが期待される．

6. まとめ

100 万ノードを用いた並列実行では，通信ライブラリのメモリ消費の増加が課題となる．そこで，メモリ消費を分

析し，通信コンテキストについてメモリ割り当てを行う数を制限する手法，また複数の通信相手で一つの通信バッファや通信コンテキストを共有する手法，また複数の MPI プロセスでオブジェクトを共有する手法を考察した．本手法を解析モデルで評価した結果，1 ノードあたりメモリ消費量を 1.07 GB に抑えられることがわかった．

今後の予定としては，性能オーバーヘッドを評価し，メモリ消費削減と性能のトレードオフを考察することが挙げられる．

7. 謝辞

本論文の一部は，文部科学省「特定先端大型研究施設運営費等補助金（次世代超高速電子計算機システムの開発・整備等 ）」で実施された内容に基づくものである．

参考文献

[1] RIKEN AICS: K computer, http://www.aics.riken.jp/en/k-computer/ (2011).
[2] Lawrence Livermore National Laboratory: ASC Sequoia, https://asc.llnl.gov/publications/Sequoia2012.pdf (2012).
[3] Fujitsu Ltd.: FUJITSU PRIMEHPC FX100, http://img.jp.fujitsu.com/downloads/jp/jhpc/primehpc/primehpc-fx100-datasheet-ja.pdf.
[4] Fujitsu Ltd.: SPARC64^{TM} XIfx ： Fujitsu's New Generation Processor for HPC, http://www.fujitsu.com/global/Images/20140811hotchips26.pdf (2014).
[5] The TOP500 project: Supter Computer TOP500, http://www.top500.org.
[6] Ajima, Y., Inoue, T., Hiramoto, S., Uno, S., Sumimoto, S., Miura, K., Shida, N., Kawashima, T., Okamoto, T., Moriyama, O., Ikeda, Y., Tabata, T., Yoshikawa, T., Seki, K. and Shimizu, T.: Tofu Interconnect 2: System-on-Chip Integration of High-Performance Interconnect, *In Proc. of ISC*, pp. 498–507 (2014).
[7] Intel Corp.: Intel® Omni-Path Architecture: Enabling Scalable, High Performance Fabrics, https://plan.seek.intel.com/LandingPage-IntelFabricsWebinarSeries-Omni-PathWhitePaper-3850 (2015).
[8] Argonne National Laboratory: MPICH, http://www.mpich.org.
[9] 山口訓央，畑中正行，高木将通，堀　敦史，石川　裕：HPC 向け高可搬通信ライブラリの設計と評価，*IPSJ SIG Notes*, Vol. 2014-HPC-145(15), pp. 1–9 (2014).
[10] Takagi, M., Yamaguchi, N., Gerofi, B., Hori, A. and Ishikawa, Y.: Adaptive Transport Service Selection for MPI with InfiniBand Network, *In Proc. of Workshop on Exascale MPI (ExaMPI)* (2015).
[11] Koop, M. J., Jones, T. and Panda, D. K.: MVAPICH-Aptus: Scalable high-performance multi-transport MPI over InfiniBand, *In Proc. of IPDPS*, pp. 1–12 (2008).
[12] Subramoni, H., Hamidouche, K., Venkatesh, A., Chakraborty, S., and Panda, D. K.: Designing MPI Library with Dynamic Connected Transport (DCT) of InfiniBand: Early Experience, *In Proc. of ISC*, pp. 278–295 (2014).
[13] Crupnicoff, D., Kagan, M., Shahar, A., Bloch, N. and Chapman, H.: Dynamically-Connected Transport Service, US Patent 8,213,315 (2012).
[14] Liu, J., Wu, J. and Panda, D. K.: High Peformance RDMA-Based MPI Implementation over InfiniBand, *In Proc. of ICS'03*, pp. 295–304 (2003).
[15] 住元真司，川島崇裕，志田直之，岡本高幸，三浦健一，宇野篤也，黒川原佳，庄司文由，横川三津夫：「京」のための MPI 通信機構の設計，先進的計算基盤システムシンポジウム論文集，Vol. 2012, pp. 237–244 (2012).
[16] Koop, M. J., Sridhar, J. K. and Panda, D. K.: Scalable MPI Design over InfiniBand using eXtended Reliable Connection, *In Proc. of IEEE Cluster* (2008).
[17] Balaji, P., Buntinas, D., Goodell, D., Gropp, W., Kumar, S., Lusk, E., Thakur, R. and Träaff, J. L.: MPI on a Million Processors, *In Proc. of EuroMPI*, pp. 20–30 (2009).
[18] Yoshinaga, K., Tsujita, Y., Hori, A., Sato, M., Namiki, M. and Ishikawa, Y.: A Delegation Mechanism on Many-Core Oriented Hybrid Parallel Computers for Scalability of Communicators and Communications in MPI, *In Proc. of the 2013 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP) Euromicro*, pp. 249–253 (2013).