Vol.2015-HPC-148 No.34
2015/3/3

ACP ライブラリの集団通信インターフェース

本田 宏明[1,3,a)]　山田 博厚[2]　森江 善之[1,3]　南里 豪志[1,3]　高見 利也[1,3]

概要：Advanced Communication for Exa (ACE) プロジェクトでは，将来のエクサスケール環境でもスケーラブルに利用可能な Advanced Communication Primitives (ACP) ライブラリを開発中である．本研究ではこの ACP ライブラリを利用した集団通信のインターフェースならびに実装の提案を行った．集団通信インターフェースとして，中間バッファの使用の有無により 2 種類の Persistent 型の通信の提案を行ない，それぞれ不要になった通信についてメモリの解放を可能とした．また，ACP の Basic Layer でサポートされている遠隔間通信を利用することで，通信を実行するプロセスが通信パターンに対応する通信命令列を処理する見通しの良い実装方法を採用した．

1. はじめに

現在，ポストペタスケールやエクサスケールクラスのスーパーコンピュータの研究開発が進められている．エクサスケールクラスでは，種々の見積りがあるがおおよそ 10^6 ～ 10^7 プロセスの計算が可能な超高並列環境となり，プロセスあたりに利用可能となるメモリサイズは消費電力等の問題から 10GByte 程度となるのでは，との予想もされている．これに対し，現在使用可能なスーパーコンピュータでは通信ミドルウエアライブラリとして MPI を利用しており，大規模並列アプリケーションの多くは MPI を用いた実装がなされている．しかしながら，この MPI ライブラリは 1. Unexpected Message 処理，2. 通信用と管理制御用のバッファ，3. 他のデバイス非依存ならびにデバイス依存バッファのそれぞれの内部実装の特徴により，メモリ使用量において問題が有ることが住元らにより指摘されている [1]–[2]．その問題のため，実用的な MPI ライブラリ実装を利用する場合では，各プロセスにおける MPI ライブラリの要求メモリ量は 10^6 プロセス数で 10GByte を越えると試算されており，起動プロセスの増加とともにいずれプログラムの実行が困難になるのでは，と危惧されている．

この問題に対し，Advanced Communication for Exa (ACE) プロジェクトでは，エクサスケール環境においてもスケーラブルな性能を持つ通信ライブラリを目指し省メモリな Advance Communication Primitives(ACP) ライブラリを開発中である [3]．2 節に後述するが，この ACP ライブラリはグローバルメモリ技術ならびに片側通信技術を採用しており，省メモリとも併せ，上位通信ミドルウエアやアプリケーションとのコデザインを可能するするため，通信のための基本構成として必要不可欠な機能を準備するといった設計思想に基づいている．現在のところ，ハードウエア依存部分を吸収する Basic Layer ならびに，Basic Layer を利用した Middle Layer から構成されている．また，すでに Basic Layer ならびに Middle Layer の一部としてのチャネルインターフェース についてはリリース版が公開されており，UDP や Infiniband, Tofu のネットワーク環境にて利用可能となっている [3].

現在の Middle Layer の開発状況としては，Send-Recv 通信に似たチャネルインターフェースや，分散動的データ構造インターフェース，隣接間通信インターフェースの開発が進みつつあり [4]-[7]，さらに，集団通信に対応するライブラリの開発が必要になっている．

そこで本研究では，ACP 通信ライブラリを用いた集団通信ライブラリについて，そのインターフェースや実装の提案を行なうことを目的とした．インターフェースとして persistent 型を採用し，集団通信を実行する際の中間データの利用の有無により 2 種類の提案を行なった．また，その実装について，ACP Basic Layer のデータの送信関数の特徴の一つである遠隔間通信を利用し，パターン型通信を用いた見通しの良い実装を提案した．

2 節では ACP ライブラリの全体構成ならびにその特徴，また，集団通信を実装するために必要な Basic Layer についての特徴を説明する．3 節では，現在考案中の persistent 型の集団通信について，2 種類のインターフェース候補に

[1] 九州大学 情報基盤研究開発センター
[2] 九州大学 システム情報科学府
[3] JST-CREST
[a)] honda.hiroaki.971@m.kyushu-u.ac.jp

ついて説明する．4 節では，集団通信ライブラリの実装のための Basic Layer を用いたパターン型通信について述べる．5 節と 6 節では今後の課題ならびにまとめを述べる．

2. ACP 通信ライブラリ

2.1 全体構成

図 1 に全体構成を示す ACP 通信ライブラリは，エクサスケールの超高並列環境においてもスケーラブルな性能を持つ通信ライブラリを目指して開発が進めてられており，特に通信に関わるメモリ使用量を低く抑えることを目標としている．これまで MPI ライブラリでは，初期化後では，各プロセスがどのタイミングにおいてもどのプロセスからでもデータの受信を可能としてきた．これに対し，通信に必要な通信元ならびに通信先情報の管理やバッファ領域等の確保ならびに解放をライブラリ使用側が明示することで，ライブラリによる暗黙のメモリ使用量を抑えている．現状では次節に説明する Basic Layer とそれを利用する Middle Layer の 2 層で構成されている．

2.2 ACP Basic Layer と利用方法

ACP ライブラリの下層に位置している Basic Layer では Ethernet や Infiniband，Tofu の通信ネットワークデバイスを抽象化している．また，RDMA かつ片側通信のモデルに基づいており，大規模並列計算機環境の全ての分散メモリを単一のグローバルメモリとして表現することで，プロセス間の通信の送信先と送信元を指定可能としている．グローバルメモリ上のグローバルアドレス (GA) は 64 ビットの uint64_t 型にて実装されており，CPU や各種ネットワークデバイスのアトミックな処理を可能としている．この GA はそれぞれのデータを管理してるプロセス上にて，Basic Layer が提供する登録関数を用いて GA 変数に割り当てる．

Basic Layer を用いプロセス a に属する *a のデータをプロセス b に属する *b に送信する通信の手続きを図 2 に示す．通信の事前準備として，a において *a に対応する GA_a を生成し，同様に b において GA_b を生成する．その後，GA_a と GA_b を通信命令を実行するプロセス c に送信する．c は a もしくは b でも良い．この際，スターターメモリ領域と呼ばれる特別な領域を利用する．次に GA_a と GA_b によって指定された領域に対し，c により通信を実行する．通信後，事後処理として a，b，c の GA 組を解放する．このように，データの送信を発行するプロセスと，実際の送信元と送信先が同じプロセスに属していても，異なるプロセスに属している遠隔間の場合でも通信が可能である．これは GA が全プロセスにおいて単一であることに由来する．

図 3 に GA 間のデータコピーに対応する関数を示す．この関数以外にも各種の atomic 通信関数が用意されている．

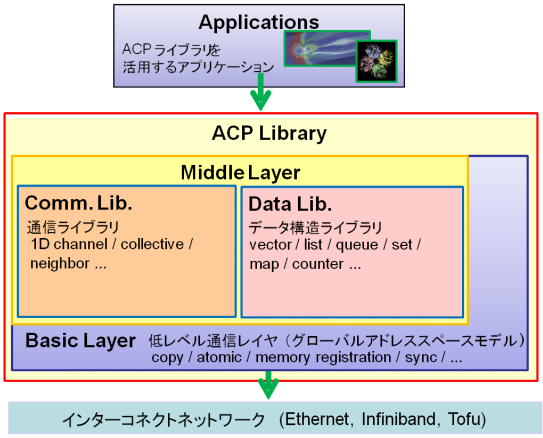

図 1 ACP 通信ライブラリ全体構成

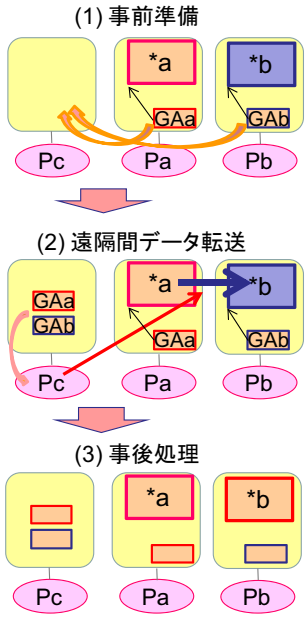

図 2 ACP Basic Layer による遠隔間データ転送

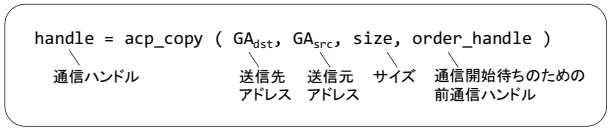

図 3 GMA 関数，acp_copy

データの送信は non-blocking を基本としており，データ送信終了の情報を保持するハンドルを関数の返り値として持つ．また，関数の引数として，データ送信元 GA，データ送信先 GA，通信サイズ，通信開始待ちのための前通信ハンドルを指定する．

3. ACP を利用した集団通信のインターフェース

3.1 Persistent 型の集団通信

実際のアプリケーションの実行の際には，同一の配列に対し同じ種類の集団通信が複数回実行される場合が多い．

また，実際の配列変数は異なるが，同一の意味を持つ多数の集団通信が繰返し実行される場合もある．これに対し，ACP ライブラリの利用には，図 2 に示したように，通信の前処理として通信対象領域に対する GA を取得し，通信を発行するプロセスに送信する必要がある．そのため，同一の配列を送信する際には対象配列の GA の取得ならびに交換を 1 度だけ行う前処理を，また，対象の配列変数が異なる場合には通信経路に敷設される中間バッファの領域を示す GA の取得ならびに交換を 1 度だけ前処理を行ない (create)，この際に得られる GA 情報やバッファ領域，通信経路情報をハンドルとして通信に利用し (send)，対象としている通信がアプリケーションを通し不要になった際には解放 (free) する仕組みが必要となる．この create, send, free からなる通信は一般に Persistent 型の通信と呼ばれているが，通信事前処理に関わるオーバーヘッドの削減や，通信のための GA 情報ならびに通信命令列情報や各プロセス上で準備される中間バッファを廃棄可能とすることで省メモリなライブラリ実装を可能である．

3.2 2 種類の集団通信インターフェース

表 1-3 に 2 種類の Bcast 型通信のインターフェース候補を示す．2 つのインターフェースは，通信のために中間バッファを使用しないかするかの違いがある．

表 1 における acp_create_bcast_direct 関数では，図 2 にて示した通信の事前準備を行ない，通信を制御するための GA 組の情報を複数含むハンドルを返す．np_group は集団通信のグループのランク番号からなる配列 *group の要素数を示す．*array は通信対象配列，arraysize は通信対象の最大サイズを示す．acp_send_bcast_direct 関数では，create 関数から取得したハンドルを利用し，図 2 にて示したように GA の組で指定されたアドレス間での通信を行なう．この際集団通信を実現するため，複数の GA 組による複数回の通信が実行される．offset は create 関数で指定した配列の先頭からのアドレスであり，size は実際に通信を実行するデータサイズを示す．このように，create 関数には引数に通信対象配列を指定しており，Bcast 内の各通信過程では中間のバッファを利用せず直接に通信対象配列を変更する．

これに対し，表 2 における acp_create_bcast_buffered 関数では，引数に通信対象の配列を指定しておらず，送信を行なう acp_send_bcast_buffered 関数において指定している．これにより，1 つの通信ハンドルを種々のデータの送信に利用可能である．こちらの関数は内部実装として通信のための中間バッファを利用しており，create 関数において buffersize 引数を指定してこのバッファ領域のサイズを指定する．こちらのインターフェースの利用ケースとして，送信対象の配列変数は異なるが，同一の意味を持つ多数の集団通信が繰返し実行される場合や，送信配列サイズが大きく，バッファを利用しない通信では通信バンド幅の問題にて性能低下が発生する場合に利用されると考えている．

3.3 集団通信上位ライブラリの実装

前節に示した 2 種類の Persistent 型の通信インターフェース候補は一般ユーザが直接利用することも可能ではあるが，アプリケーションから利用する際には，buffered 型の create 関数のバッファサイズ引数をユーザから明示的に指定せず，デフォルト値やデータ送信遅延の測定結果等から最適な値を算出する方法をとることも可能である．また，create や free 関数をユーザに明示的に指定させるのではなく，最初に通信関数が呼び出された場合のみ create を行なうように，create は send の内部に実装する方法もある．free についても，自動メモリ解放等の革新的な技術を利用することで free を明示的に記述しない実装も可能かもしれない．そのような場合でも，上記の create や free 関数は暗黙的に利用可能であるため，今回検討しているインターフェースは上位の集団通信インターフェースを実装する道具として位置付けられると考えている．そのため，通信の Primitives を準備するという考えに基づき 2 種類の Persistent 型の通信インターフェースを ACP Middle Layer の候補とした．

3.4 関連研究

3.4.1 MPI における Persistent 型通信

同一の引数を利用した通信が繰返し実行される場合に対し，2 度目以降の通信の開始に関わるオーバーヘッドならびにメモリ使用量を抑えるため，MPI ライブラリにおいても Persistent 通信についての仕様が提案されている．しかしながら，仕様に解放のための関数は定義されておらず，ライブラリのメモリ使用量をユーザから制御可能とはしていない．現状の MPI3.0 規格では，Send, Recv のみ Persistent Communication Request について記述がなされている．集団通信については，今後の MPI 規格を決定する MPI Forum においても議論の段階にある [11].

4. パターン通信を利用した集団通信の実装

4.1 通信依存関係と ACP を利用した遠隔間通信

集団通信は一般的に通信の組合せとして記述可能であり，親ランクのデータを自ランクを経由して子ランクに送信する場合など，それぞれの通信には依存関係がある．これに対し，acp_copy では通信先，通信元，依存関係にある前通信の指定をすることで，依存関係のある素通信過程を記述可能としている．図 4 参照．また，この際に通信先と通信元について両方とも遠隔プロセスであることを許しているため，依存関係のある複雑な通信のパターンに対し，通信の命令列として 1 つのプロセスのみにより処理することも可能である．

表 1　Bcast 型通信インターフェース (中間バッファなし)

名称	定義
通信経路生成	acp_col_t acp_create_bcast_direct(int root, int np_group, int *group, void *array, int arraysize)
通信実施	acp_request_t acp_send_bcast_direct(acp_col_t handle, int offset, int size)

表 2　Bcast 型通信インターフェース (中間バッファ利用)

名称	定義
通信経路生成	acp_col_t acp_create_bcast_buffered(int root, int np_group, int *group, int buffersize)
通信実施	acp_request_t acp_send_bcast_buffered(acp_col_t handle, void *array, int size)

表 3　共通インターフェース

名称	定義
通信完了待ち	int acp_wait_col(acp_request_t request)
通信経路解放	int acp_free_col(acp_col_t handle)

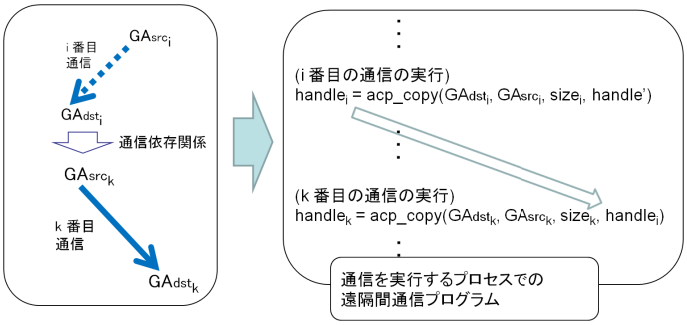

図 4　ACP の遠隔間通信を利用した依存関係を含む複数の通信の記述

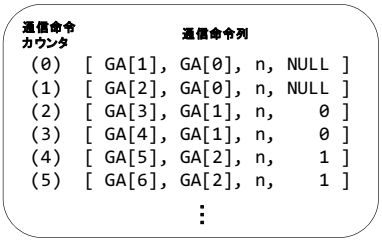

図 5　Bcast 通信に対応する通信命令列

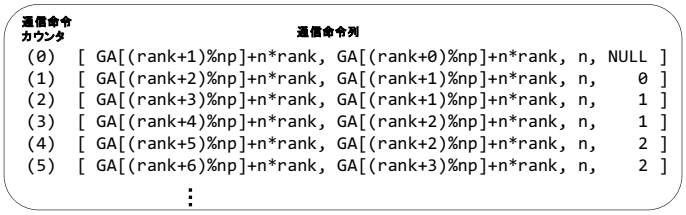

図 6　各ランクの Allgather 通信に対応する通信命令列

4.2　通信命令のための前処理関数

通信命令列を作成するには，1. 通信を実行するルートランクが，各集団通信に対応する通信パターンデータとして送信元ならびに送信先情報と通信の依存関係のリストを準備し，2. 通信元ならびに通信先として通信に関与するデータアドレスから GA の情報を各ランクが生成し，3. 通信を実行するルートランクが，各プロセスと通信することで遠隔間通信に必要な GA を収集する必要がある．これらの手続きそれぞれについて，ACP を利用する際に便利なツールとしての前処理関数を実装した．

4.3　通信命令列例

4.3.1　bcast 通信

図 5 に遠隔間通信を利用した Bcast のための通信命令列を示す．通信命令列の各行は acp_copy 関数の一組の引数に対応している．図 4 参照．ルートランクは 0 としており，2 分木アルゴリズムによる n バイトのデータ転送をしている．最初は 0 ランクの配列から 1 と 2 のランクの配列へと送信するが，そのため，GA[0] から GA[1] ならびに GA[0] から GA[1] へと，送信元と送信先についてグローバルアドレスによる指定がなされている．また，これらの通信には依存関係にある前通信は無いため NULL を指定している．次の命令は同様に 2 分木に従い，GA[1] から GA[3] と GA[4] へのデータの送信を指定するが，この際には GA[1] の箇所にデータが到達している必要があるため，(0) 番目の通信完了を待つ必要がある．

4.3.2　allgather 通信

安島らが ACP ライブラリを利用した non-blocking 型の Allgather アルゴリズムを既に提案している [4] が，これに対応する遠隔間通信を利用した rank 番目のランクにおける通信命令列を図 6 に示す．プロセス数は np とする．また，この通信では送信分の n バイトのデータが受信配列の $n * rank$ の位置から保存されている事を前提にしている．最初の送信を除き，各ランクのデータは 2 分木アルゴリズムにより 2 つの子ランクへと送信されるが，その際に，$n * rank$ 分だけシフトしたアドレスからのデータを送信することで gather に対応する処理を実現している．他のランクにおいても，遠隔間通信として rank の値は異なるが同じ通信命令列が実行される．異なるランクの遠隔間通信同士には依存関係がないため，通信命令を発行しているランク間での同期処理の必要がないシンプルな記述が可能である．

4.4　メモリ使用量見積り

前節に示した 2 種類の集団通信インターフェース候補の direct，buffer 型の通信においては，もしも全ての通信の命令をルートランクのみで実行する場合では，全てのプロセ

スの通信対象配列に対する 64Bit の GA 情報をルートランクに保存する必要がある．このため，プロセス数が 10^6 の場合には 8MByte 程度必要となる．また通信命令列の保持のため，1 つの命令に 24Byte 必要とする場合では，Bcast 型通信の場合には 2 分木アルゴリズムで 40MByte 必要となる．通信命令を実行しない各プロセスに必要な通信管理情報は，通信対象配列に対応する GA 情報と，buffered の場合には内部に保持された中間バッファサイズのみである．このうち中間バッファサイズはライブラリユーザから制御可能であるため，エクサスケールクラスにおける 10GByte 程度のメモリ制限内に収める事が可能であると予想している．

4.5 関連研究

4.5.1 libNBC

Hoefler らが開発している Non-Blocking Collective library (libNBC) では，集団通信等に現われるパターンを処理するための内部実装として，各プロセスが処理すべき通信命令列を記した通信のスケジュールを実行するための機構を実装している [8],[9]．通信命令列は拡張バッカス・ナウア記法により記述されており，自由な記述が可能となっている．これにより通常の集団通信以外に隣接間通信や種々の通信型にも応用されている．通信演算命令列の記述はプロセスが動作しているローカルな通信についてのみであり遠隔間通信の実装はされていない．

4.5.2 Mellanox Core-Direct library

Mellanox 社が提供する最新の Infiniband には，NIC に加算器等の演算器が搭載されている．集団通信等において各プロセスが実施すべき通信ならびに演算命令列をタスクリストとして NIC のドライバに渡すことが可能であり，RDMA 通信の際に演算を含めた動作を実行させることが可能となっている．これを Mellanox では Core-Direct 機能と呼んでいる [10]．ただし，この機能は Mellanox により正式にはサポートされていないため動作保証がなく，ドキュメントも非常に少ないのが現状である．

5. 今後の課題

本稿では Reduce 処理のインターフェースや実装について明示的には言及してこなかった．Reduce は提案の集団通信のインターフェースのうち，バッファ利用の方法を用いることで容易に実装が可能ではあるが，本研究で利用した遠隔間通信の通信命令列のみを利用するだけでは実装出来ず，各プロセスにデータを送信する際に演算の要求が必要となる．ACP ライブラリは通信のため Primitives を用意することを設計方針としているため，計算と混在が必要な Reduce などの通信は ACP より上位のライブラリとして実装するなど，ACP ライブラリ中に全く演算を含めないかについてはまだ議論が必要となっている．また例えば，Allgather 処理の後，各プロセスにて和を計算する Allreduce 実装のように，通信と演算を完全に分離する方法も有るが，通信対象となるデータ量が多い場合にはプロセス数の増加とともに現実的な方法ではなくなり，いつでも利用可能は方法とはいえない．

本研究では集団通信の実装として ACP Basic Layer の遠隔間通信の機能を用いたが，これにより実装は見通し良くなったもの，従来の各プロセスが自身に関わる通信を分担する方法に比較しどの程度の性能低下となるのかの性能評価が必要であり現在実施中である．

4 節にも言及したが，ACP による集団通信ライブラリが使用するメモリ量は十分少ないことが期待できる．すでに秋元らにより MPI ライブラリについてはメモリ利用量を測定可能なライブラリが開発されている [12]．これを実際のアプリケーションを対象とした ACP の通信にも利用可能とし，使用メモリの測定を実施する予定である．

6. まとめ

Advanced Communication for Exa (ACE) プロジェクトでは，将来のエクサスケール環境でもスケーラブルに利用可能な Advanced Communication Primitives (ACP) ライブラリを開発中である．本研究ではこの ACP ライブラリを利用した集団通信のインターフェースならびに実装の提案を行った．集団通信インターフェースでは，通信の前処理として集団通信に関わるハンドルを交換し，そのハンドルに基づき通信を実行し，通信が不要になった場合にハンドルを解放可能とする Persistent 型の通信を採用した．また実装については，ACP の Basic Layer でサポートされている遠隔間通信を利用することで，通信を実行するプロセスがパターン化された通信命令列に従い通信を行なう方法を用いた．これにより不要な同期を省く事が可能となる．また，提案実装におけるライブラリの Bcast 型通信における消費メモリ量について見積り，10^6 プロセスが参加する 2 分木アルゴリズムの通信では，最大 48MByte 程度であり，エクサスケール環境におけるプロセスあたりのメモリ量の制限に収まることが分った．

謝辞 本研究は，科学技術振興機構 戦略的創造研究推進事業 (CREST) の研究領域「ポストペタスケール高性能計算に資するシステムソフトウエア技術の創出」の研究課題「省メモリ技術と動的最適化技術によるスケーラブル通信ライブラリの開発」の一部として実施された．

参考文献

[1] 住元真司, 安島雄一郎, 佐賀一繁, 野瀬貴史, 三浦健一, 南里豪志, エクサスケール通信向け ACP スタックの設計思想, 情報処理学会研究報告 2014-HPC-143-8(2014).
[2] Shinji Sumimoto, Yuichiro Ajima, Kazushige Saga, Takafumi Nose, Naoyuki Shida and Takeshi Nanri, ACP: Advanced Communication Primitives for Exa-scale Sys-

tems, 11th International Meeting High Performance Computing for Computational Science (VECPAR2014), Eugene, Jun.2014.
[3] ACE Project, http://ace-project.kyushu-u.ac.jp/index.html.
[4] 安島雄一郎, 佐賀一繁, 野瀬貴史, 三浦健一, 住元真司, ACP 基本層の設計思想とインターフェース, 情報処理学会研究報告 2014-HPC-143-9(2014).
[5] 佐賀一繁, 安島雄一郎, 野瀬貴史, 三浦健一, 住元真司, ACP 基本層の実装と初期評価, 情報処理学会研究報告 2014-HPC-143-10(2014).
[6] 安島雄一郎, 佐賀一繁, 野瀬貴史, 志田直之, 住元真司, ACP の分散動的データ構造インターフェース, 情報処理学会研究報告 2014-HPC-146-18(2014).
[7] Takeshi Nanri, Takeshi Soga, Yuichiro Ajima, Yoshiyuki Morie, Hiroaki Honda, Taizo Kobayashi, Toshiya Takami, and Shinji Sumimoto, Design and Implementation of Channel Interface as a Memory Efficient Communication Model, Annual Meeting on Advanced Computing System and Infrastructure (ACSI2015), Tsukuba, Jan.2015.
[8] Torsten Hoefler and A. Lumsdaine, Design, Implementation, and Usage of LibNBC Open Systems Lab, Indiana University, presented in Bloomington, IN, USA, School of Informatics, Aug. 2006.
[9] LibNBC - Nonblocking MPI Collective Operations, http://htor.inf.ethz.ch/research/nbcoll/libnbc/, and references there in.
[10] RDMA Aware Networks Programming User Manual, Rev.1.6, http://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf.
[11] Message Passing Interface Forum, http://www.mpi-forum.org/.
[12] 秋元秀行, 安島雄一郎, 安達知也, 岡本高幸, 三浦健一, 住元真司, DMATP-MPI: MPI 向け動的メモリ割当分析ツール, 情報処理学会研究報告 2013-HPC-138-14(2013).