HPC向け高可搬通信ライブラリの設計と評価

山口 訓央[2,a)]　畑中 正行[2,b)]　高木 将通[2,c)]　堀 敦史[2,d)]　石川 裕[1,e)]

概要：ハイパフォーマンスコンピューティング (HPC) 向けアプリケーションの通信ランタイムにネットワークハードウェア (NW-HW) 非依存通信 API を提供する通信ライブラリについて、ライブラリ内部を NW-HW に依存しない実装がなされた NW-HW 非依存レイヤと、NW-HW に依存する実装がなされた NW-HW 依存レイヤに分け、通信ライブラリを構成するコードのうちより多くのコードを NW-HW 非依存レイヤへ移動させることで、異なる NW-HW をサポートする際に新規開発が必要となるコード量を減らす設計を提案し、評価する。具体例として IB および Tofu 向け実装について、NW-HW 依存・非依存レイヤへのコードの分け方を説明する。さらに、設計の妥当性を、新規開発が必要となるコード行数の確認と、NAS Parallel Benchmarks を用いた、提案ライブラリを用いた MPI ライブラリと既存 MPI ライブラリとの性能比較により確認する。

キーワード：通信ライブラリ、通信ランタイム、ネットワークハードウェア、可搬性、コード再利用

Design and Implementation of a Portable Commnication Library for High Performance Computing

Abstract: We propose a design of increasing the amount of reusable code of a communication library that provides network hardware (NW-HW) independent communication API to communication run-times of high performance computing applications by modifying its implemention of low level protocols. Those protocols are sequenced by two modules, that are (1) state machines using NW-HW independent events and commands and (2) NW-HW dependent part that interprets NW-HW independent commands issued by the state machines and manipulates NW-HW. The amount of code that can be reused when the library switches its support to different NW-HW can be increased in this design. We explain this design using the working code that supports InfiniBand and Tofu. And the design is validated by comparing the amount of reusable code against the modules with the similar functionality and comparing communication performance of the MPI library implemented by using the proposed library and that of an exsiting MPI library using NAS Parallel Benchmarks.

Keywords: Communication library, communication runtime, network hardware, portability, code reuse

1. 導入

ハイパフォーマンスコンピューティング（HPC）向けコンピュータは、サーバがネットワークによって接続されたクラスタアーキテクチャを取ることが多い。これは、一つのサーバからなるノードで SPARC64[1]、Intel Xeon Phi[2] といったメニーコアプロセッサを用いることで多数の演算器を並列動作させた上で、さらに多数のノードを並列動作させて性能向上をはかるためである。このアーキテクチャでは、ネットワークハードウェア (NW-HW) の革新が性能向上で大きな役割を果たしてきた。例えば、RDMA による低遅延化 [3] や、ルータのプロセッサチップへのインテグレーションによる低遅延化 [4] などである。

NW-HW 性能向上は競争の形で起こるため、ある時点で複数の NW-HW が選択可能になったり、従来広く利用されていたものが置き換えられたりする。このため、NW-HW 性能向上を HPC 向けコンピュータの性能向上で享受する

[1] 東京大学大学院情報理工学系研究科
[2] 理化学研究所計算科学研究機構
[a)] norio.yamaguchi@riken.jp
[b)] mhatanaka@riken.jp
[c)] masamichi.takagi@riken.jp
[d)] ahori@riken.jp
[e)] ishikawa@is.s.u-tokyo.ac.jp

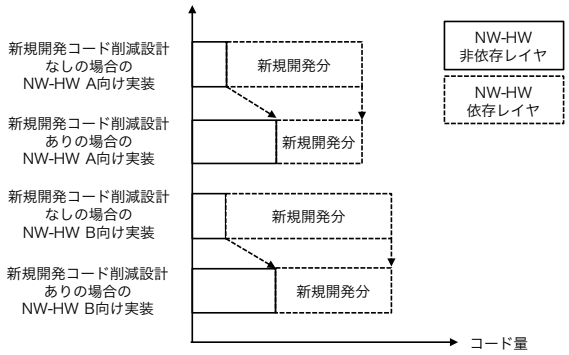

図 1 異なる NW-HW をサポートする際に新規開発が必要なコードを減らす設計と開発コストの関係.

ためには異なる NW-HW へ対応する必要、特に低開発コストで対応する必要がある。この開発コストを下げる方法を**図 1** を用いて説明する。通信ライブラリを、NW-HW 非依存レイヤ、NW-HW 依存レイヤに分けて設計するとする。この際に異なる NW-HW をサポートしようとすると、NW-HW 依存レイヤは新規開発となり、従ってこのサイズが大きいほど開発コストが大きくなる。そのため、通信ライブラリを構成するコードから、より多くのコードを NW-HW 非依存レイヤへ移動させることで、新規開発が必要なコード量を減らすことができ、従って、開発コストを低減できる。

ところが、従来の通信ライブラリでは、この新規開発が必要なコード量削減が不十分であった。例えば、MPI ライブラリの実装である MPICH[5] や OpenMPI[6] や Buffered Message Interface(BMI)[7] では、NW-HW 非依存・NW-HW 依存レイヤに分ける構造を取っているが、NW-HW 非依存レイヤで実装できる機能を NW-HW 依存レイヤで実装しており、従ってそれらの機能は異なる NW-HW をサポートする際に開発をしなおす必要があった。

この状況に対して我々は、広いアプリ通信ランタイムに NW-HW 非依存の API を提供する通信ライブラリについて、異なる NW-HW をサポートする際に新規開発が必要なコード量をより減らす設計を提案する。このライブラリを Low Level Communication Library (LLC) と呼ぶ。LLC がサポートするアプリ通信ランタイム、LLC 自身、NW-HW の位置づけを**図 2** に示す。サポートするソフトウェアモジュールは、MPI の通信ランタイム、PGAS 言語の通信ランタイム、スナップショットなど一時データを格納するファイルシステム、可視化、センサデータのデータ同化とし、LLC はそれらのアプリおよびアプリの通信ランタイムの下に位置し、NW-HW を操作する。NW-HW は、まず InfiniBand(IB) および Tofu[8] をサポートし、順次広げてゆく。

本稿の貢献は以下である。

- 異なる NW-HW をサポートする際に書き直しが必要となるコード量を減らす通信ライブラリのレイヤ設計を示し、実際に機能する IB および Tofu 向け実装について設計を説明し、またその妥当性を評価した。

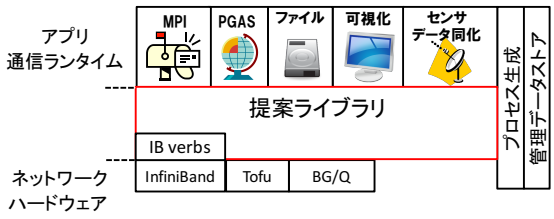

図 2 提案ライブラリ LLC、HPC 向けアプリ通信ランタイム、NW-HW の関係.

これ以降の論文構成は以下の通りである。2 節で関連文献を論じ、3 節で設計を説明し、4 節で IB および Tofu 向け実装を説明し、5 節で評価結果を説明し、最後に 6 節で結論を述べる。

2. 関連文献

Buffered Message Interface(BMI)[7] は並列ファイルシステム Parallel Virtual File System version 2 (PVFS2)[9] に NW-HW 非依存の通信 API を提供するライブラリである。NW-HW 非依存レイヤと NW-HW 依存レイヤに分ける構造を取るところは LLC と同一であるが、BMI は異なる NW-HW をサポートする際に新規開発が必要なコード量の削減を第 1 の目的としていないため、いくつかの機能について、NW-HW 非依存レイヤに実装することでそのようなコード量削減ができる場合にそのようにせず、NW-HW 依存レイヤに実装することを選択している。例えば、大メッセージの送信プロトコルである Rendezvous Protocol[10] がそのように実装されている。

uDAPL[11] はアプリ通信ランタイムに NW-HW 非依存の RDMA 通信 API を提供するライブラリである。uDAPL は、コネクション、RDMA コマンド、受信側がデータ格納アドレスをポストするタイプの送受信、仮想・物理アドレス変換テーブル登録、イベント通知といった、NW-HW に近い、抽象度の低い通信 API を提供する。LLC は、タグマッチを伴う送受信プロトコルといった、より抽象度の高い通信 API を提供する点が異なる。

MPICH[5] および OpenMPI[6] は、それぞれ MPI ライブラリの実装であり、レイヤ構造を採用し、`netmod` あるいは Byte Transfer Layer と呼ばれる NW-HW 依存レイヤを設けている。LLC は、このレイヤからさらに NW-HW 非依存コードを NW-HW 非依存レイヤに移動することで異なる NW-HW をサポートする際に新規開発が必要なコード量を削減している。

Portals[12] はアプリ通信ランタイムに NW-HW 非依存の通信 API を提供するライブラリであり、特に、NW-HW にインテグレートされたイベントドリブン処理支援機構を活用することによる低遅延化を目的とする。Portals は、Network Abstraction Layer と呼ぶ NW-HW 依存レイヤを

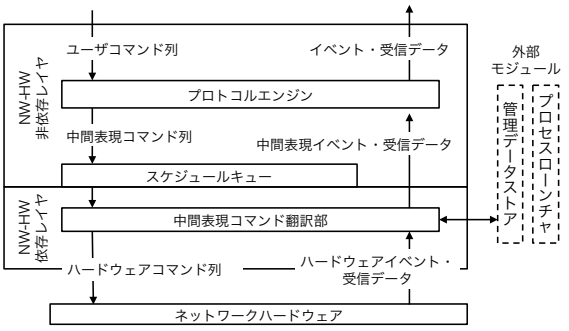

図 3 提案ライブラリ LLC のモジュール構成.

設けている。一方 LLC は、このレイヤに相当するレイヤからさらに NW-HW 非依存コードを NW-HW 非依存レイヤに移動することで異なる NW-HW をサポートする際に新規開発が必要なコード量を削減している。

PAMI[13] は、IBM Blue Gene/Q の NW-HW とプロセッサの持つ、ネットワークデータ到着時のスレッド高速起床機構、L2 キャッシュ上でのアトミック操作命令を活用して低遅延高スループットを目指した通信ライブラリである。しかし、PAMI は、Blue Gene/Q 以外のプロセッサと NW-HW をサポートすることを目的としない。一方、LLC は、IB と Tofu を始めとして複数の NW-HW をサポートする際に新規開発が必要なコード量削減を目的にしている。

PM2[14] は、Myrinet、Ethernet など異種の NW-HW を同時にサポートすることで、異種ネットワークが混在する環境をサポートすることを目的とした通信ライブラリである。LLC は、異なる NW-HW をサポートする際に新規開発が必要なコード量削減を目的としている点が異なる。

3. 設計

ノードとネットワークのワイヤを結ぶハードウェアを、IB の呼び方に合わせて Host Channel Aadapter(HCA) と呼ぶ。この節では、設計の全体構成を説明した後、個々のモジュールの構成と動作について、IB 向け、Tofu 向け実装での異なる NW-HW をサポートする際に新規開発が必要なコード量削減の観点で説明する。HPC 向けアプリケーションでの通信主体をピアと呼ぶ。例えば、MPI プログラムでは MPI プロセスがピアに相当する。

3.1 全体構成

図 3 を用いて全体構成を説明する。NW-HW サポートの切り替えは、ソースレベルで、すなわちコンパイル前のコンフィギュレーションのフェーズで行われる。再利用可能コードを増やすために、NW-HW に依存しないイベント（中間表現イベントと呼ぶ）を受け取り、NW-HW に依存しないコマンド（中間表現コマンドと呼ぶ）または中間表現イベントを発行するステートマシンから構成されるプロトコルエンジンを設け、これを中心に通信ライブラリを構成する。また、プロトコルエンジンと NW-HW の間には、中間表現コマンドを翻訳して HCA を操作する中間表現コマンド翻訳部を設ける。さらに、通信に必要なネットワークアドレスの交換などを助けるために、管理データストアを設ける。管理データストアは、様々な実装を用いることができるように、本体は外部モジュールとし、API を変換するアダプタのみ実装する。さらに、プロセスを立ち上げるプロセスローンチャを設け、管理データストア同様外部モジュールとし、API を変換するアダプタのみ実装する。次に、通信コマンドの並列発行を可能にすることで通信遅延を削減するために、プロトコルエンジンと中間表現コマンド翻訳部の間に、中間表現コマンドのスケジュールを行えるキューを設ける。この機能については個々のモジュールを説明する際に改めて説明する。さらに、通信の中間バッファの割り当てを行うことでメモリ消費量を削減するために、プロトコルエンジンに、バッファ割り当て機能を追加する。この機能についてはプロトコルエンジンが担う個々のプロトコルを説明する際に改めて説明する。プロトコルエンジンとスケジュールキューが NW-HW 非依存レイヤを構成し、中間表現コマンド翻訳部が NW-HW 依存レイヤを構成する。NW-HW 非依存レイヤに必要な機能のうちより多くの割合を実装することで異なる NW-HW をサポートする際に新規開発が必要なコード量を削減する。

3.2 モジュール動作

図 3 を用いて、それぞれのモジュールの動作を説明する。

3.2.1 プロトコルエンジン

プロトコルエンジンは、ユーザコマンド列を入力して通信のプロトコルを実行する。ユーザコマンドは大きく分けると以下の 3 つである。

Send-Receive 送信側がデータの読み出し元アドレス・型・サイズ・ユーザタグを指定し、受信側がデータの書き込み先アドレス・型・サイズ・ユーザタグを指定すると、タグによるマッチングが行われ指定されたアドレスにデータが書き込まれる送受信コマンドである。マッチングについてさらに説明する。送信側はアプリ通信ランタイムの定めるルールに基づいてタグを生成してデータに付与し、同一のタグを持つ受信コマンドだけがそれを受け取ることができる。MPI の送受信コマンドの実装を容易にするために提供する。

Unsolicited-Send 送信側はデータの読み出し元アドレス・型・サイズを指定すると、受信側通信ライブラリの中間バッファにデータが書き込まれる送信コマンドである。アプリ通信ランタイムでの制御パケット送受信、あるいは Active Message の実装を容易にするために提供する。

キューによるイベント通知 アプリ通信ランタイムにデータ到着、バッファ上書き可能といったイベントを通知する。アプリ通信ランタイムではイベント駆動型プロ

グラミングが用いられることが多いため、これを容易にするためにイベントとイベントキューの形で提供する。

プロトコルはステートマシンによって実装され、HCAに指示を与える必要がある場合には、中間表現コマンドをスケジュールキューに出力し、中間表現コマンド翻訳部にNW-HWを操作させる。また、中間表現コマンド翻訳部から受け取ったイベントを用いてステートマシンのステートを遷移させる。

プロトコルエンジンが実行するプロトコルのうちIB向けおよびTofu向け実装での、異なるNW-HWをサポートする際に新規開発が必要なコード量削減の観点から重要なものは以下のブートストラップリモートメモリアクセス、コネクション、バッファ割り当てであり、以下で動作を説明する。また、データ送信は他のプロトコルの動作に関連するため合わせて動作を説明する。

3.2.1.1 ブートストラップリモートメモリアクセス

異なるノードに存在するピアが情報を低遅延で交換できるようにして、制御動作をサポートする。例えば、IBでは、初めて送信する際に送信側はNW-HWレベルでコネクションを張る必要があり、受信側の情報を必要とするため、この機能を必要とする。

3.2.1.2 コネクション

通信前に必要となるNW-HWレベルのコネクション動作を支援する。例えばIBでは、通信を開始するピアをイニシエータ、通信を受け取るピアをレスポンダと呼んだ際に、Queue Pair（QP）と呼ばれる構造体をイニシエータ、レスポンダに構築し、イニシエータのメモリ上に存在するQPにレスポンダのHCA物理ポートのネットワークアドレスとレスポンダのメモリ上に存在するQPのQP番号を入力する必要があり、この動作を行う。

3.2.1.3 バッファ割当

多くのNW-HWで、データ送信の前に、受信側が受信データを格納するアドレスを仮想・物理アドレス変換テーブルに登録し、HCAが変換テーブルを参照できるようにする情報を取得し、このアドレスと参照のための情報を送信側に伝える必要がある。送信すべきデータのサイズが小さい場合は、このステップを事前実行することで遅延を削減するプロトコルを用いる。このプロトコルは以下のステップからなる[15]。

(1) イニシエータはレスポンダのメモリに存在する既に割り当てられているリングバッファの、生産者ポインタが指すスロットにデータを書き込む。
(2) レスポンダはそのリングバッファのスロットから最終的な受信バッファへメモリコピーを行う。また、リングバッファが溢れないようにタイミングを見計らってイニシエータに消費者ポインタを通知する。

バッファ割当プロトコルは、このプロトコルで利用されるリングバッファ割り当てを行う。すなわち、レスポンダは初めて通信を行う際などにリングバッファを割り当て、そのアドレスを仮想・物理アドレス変換テーブルへ登録し、変換テーブル参照のための情報を取得し、イニシエータにこのアドレスと参照のための情報を伝達する。なお、リングバッファはメモリ消費削減のため、総サイズを制限し、割り当てポリシーに従って動的に割り当てる。割り当てポリシーにはfirst-come first-served[15]を用いる。

3.2.1.4 送信

NW-HWレベルで、データを送信側のメモリから読み込んで、受信側のHCAに渡し、受信側のメモリに書き込むには、送信側と受信側で様々なステップが必要であるため、これを実行する。例えば、先の節で説明した送信プロトコルをHCAに指令を与えたり、イベントを受け取ったりすることで実行する。また、ユーザコマンドのセマンティクスに基づいて、より高度なプロトコルを実行する。例えば、データが到着したときに、それがユーザデータであるか、制御パケットであるか識別し、ユーザデータであったらどの受信コマンドがそのデータを受け取るかパケットに付与されたタグを元に検索する、といったことを実行する。

3.2.2 スケジュールキュー

データ送信は、一連の資源獲得・解放と見なすことができ、送信と資源はある時点で複数存在するので、これを利用して並列化する。例えば、Tofuを例に挙げると、ノード内ルータへ投入するポート（内部入力ポートと呼ぶ）、ノード内ルータの出力ポート、さらにネットワークの経路上にあるノード、受信ノード内ルータの入力ポートなどが資源と見なすことができ、通信はこれらの資源を次々と獲得解放していく。この時に複数の通信と複数の資源が存在する場合は、複数の通信の一部あるいはすべてを並列化することができる。例えば、複数のデータ通信で、別々の内部入力ポートと別々の出力ポートを並列に使用することによって、１ノードからのデータ送信を並列化することができる[16]。スケジュールプロトコルは、この資源割り当てによる並列化を行う。なお、スケジュールキューは、ソースの枠組みが存在するのみで、現時点ではまだスケジュールアルゴリズムが実装されていない。

3.2.3 中間表現コマンド翻訳部

中間表現コマンド翻訳部は、中間表現コマンドをハードウェアコマンドに翻訳して、HCAに指示を出す。また、HCAから発生したイベントを中間表現イベントに翻訳してプロトコルエンジンに渡す。

3.2.4 管理データストア

管理データストアは、ピア間の通信に必要な情報の交換をkey-value store (KVS)のAPIで提供する。例えば、IBでは、あるピアへの送信を開始する前に、送信先ノードのHCAポートのネットワークアドレスを必要とする。そこで、送信元ピアは、このネットワークアドレスを管理デー

タストアを用いて取得する。管理データストアは、IB 実装では memcached、Tofu 実装では独自 KVS をサポートする。

4. IB および Tofu 向け実装

この節では、IB 向け実装と Tofu 向け実装において動作が大幅に異なるため異なる NW-HW をサポートする際に新規開発が必要なコード量削減のための設計が課題となるプロトコルについて、コードの NW-HW 非依存レイヤと依存レイヤへの切り分け方の具体例を説明する。

4.1 ブートストラップリモートメモリアクセス

IB での機能要件を説明する。ピアのペアが通信するためには、送信側、受信側に Queue Pair (QP) という構造体が必要である。さらに、通信の両サイドのピアをイニシエータ、レスポンダと呼んだ際に、イニシエータでは、QP にレスポンダの HCA の物理ポートの GID、LID と呼ばれるネットワークアドレスに相当する情報と、レスポンダに存在する QP の QP 番号と呼ばれる情報を登録する必要がある。さらに、レスポンダ側では、データが書き込まれる先のメモリエリアの仮想・物理アドレス変換テーブルへの登録と、変換テーブル参照の正当性を示す鍵の取得が必要で、イニシエータ側はその仮想アドレスと鍵を必要とする。このプロトコルの IB 向け実装詳細は **A.1.1 節**に示す。

このプロトコルについては、KVS 部分は NW-HW 非依存部、すなわち管理データストアへ切り分け、プロトコルは NW-HW 依存レイヤ、すなわち中間コマンド翻訳部に切り分けて実装した。また、最新の Mellanox IB HCA の機能を用いると、1 つの QP を用いて複数のピアとデータの信頼性を保った通信が行えるため、$n-1$ 個ではなく 1 つの QP を作成するだけにできるが、最新 HCA が入手できなかったため、上記の実装を利用している。

Tofu での機能要件を説明する。 ピアのペアの通信には、データ受信時に受信側のメモリに仮想アドレスを用いてデータを書き込めるようにするための、仮想・物理アドレス変換テーブルのエントリを作成する必要がある。さらに、送信側がパケットに付与する、受信側が受信時にそのテーブルを参照するために使用するタグについて送信側と合意を取る必要がある。

このプロトコルについては、プロトコルは NW-HW 依存レイヤ、すなわち中間コマンド翻訳部のみで実装した。

4.2 コネクション

IB での機能要件を説明する。送信の前に送信側と受信側は QP を作成し、送信側は送信側 QP に受信側ネットワークアドレスである GID、LID と、受信側 QP の QP 番号を入力する必要がある。受信側は受信側 QP に送信側 GID、LID と、送信側 QP の QP 番号を入力する必要がある。このプロトコルの IB 向け実装詳細は **A.1.2 節**に示す。

このプロトコルにおいて、ステートマシンを用いたプロトコルシーケンス部分を NW-HW 非依存レイヤ、すなわちプロトコルエンジンとスケジュールキューに切り分け、コネクションプロトコル用のコマンドスロットへの書き込み、コマンド受信とステートマシン遷移は、NW-HW 依存レイヤ、すなわち中間コマンド翻訳部へ切り分けて実装した。なお、IB の send/receive コマンドを用いれば、CAS を省略することができるが、send/receive コマンドがない、あるいは send/receive コマンドの遅延あるいはメモリ消費コストが大きい NW-HW への移植容易性向上の実験のため CAS を用いる実装とした。

Tofu での機能要件を説明する。Tofu では NW-HW レベルでのコネクション動作は不要である。そのため、Tofu では、このプロトコルは起動されない。

4.3 バッファ割当

ピア A がピア B にデータを送信しようとしているとして、A をイニシエータ、B をレスポンダと呼ぶ。IB での機能要件を説明する。IB では、レスポンダがリングバッファとして用いられる中間バッファを実行時に割り当て、仮想・物理アドレス変換テーブルに登録し、テーブル参照の正当性を示す鍵を受け取り、仮想アドレス、リングバッファのスロット数、鍵をイニシエータに知らせる必要があり、イニシエータはそれらの情報を用いてリングバッファの生産者ポインタ、消費者ポインタを初期化する必要がある。このプロトコルの IB 向け実装詳細は **A.1.3 節**に示す。

このプロトコルにおいて、ステートマシンを用いたプロトコルシーケンス部分を NW-HW 非依存レイヤ、すなわちプロトコルエンジンとスケジュールキューに切り分け、コネクションプロトコル用のコマンドスロットへの書き込み、コマンド受信とステートマシン遷移は、NW-HW 依存レイヤ、すなわち中間コマンド翻訳部へ切り分けて実装した。

Tofu での機能要件を説明する。Tofu では、レスポンダがリングバッファとして用いられる中間バッファを実行時に割り当て、仮想・物理アドレス変換テーブルに登録し、リングバッファの生産者・消費者ポインタをレスポンダ上のメモリに保存し、イニシエータにこれらのポインタを通知し、また、イニシエータがこの生産者・消費者ポインタをイニシエータ上のメモリに保存する必要がある。なお、レスポンダの HCA が変換テーブルを参照する際に用いるタグについては静的に合意した値を用いる。このプロトコルの Tofu 向け実装詳細は **A.1.3 節**に示す。

このプロトコルにおいて、ステートマシンを用いたプロトコルシーケンス部分を NW-HW 非依存レイヤ、すなわちプロトコルエンジンとスケジュールキューに切り分け、NW-HW レベルのコマンド送受信、送受信検知とステート

表 1 LLC/IB 評価環境のパラメタ

コンポーネント	パラメータ
ノード プロセッサ	Intel Xeon E5-2670v2, 2.501 GHz, 10-physical core, 20-logical core, 2-socket
ネットワーク	Mellanox ConnectX-3, 6.79 GB/s
I/O バス	PCI Express 3.0, 8GT/s 8-lane, 7.88 GB/s

表 2 LLC/Tofu 評価環境のパラメタ

コンポーネント	パラメータ
ノード プロセッサ	SPARC64 VIIIfx, 2.0 GHz, 8-core,
ネットワーク	Tofu, 5.0 GB/s, 1 ノードあたり外部 10 ポート、内部 4 ポート

マシン遷移は、NW-HW 依存レイヤ、すなわち中間コマンド翻訳部へ切り分けて実装した。

5. 評価

提案ライブラリの設計の妥当性を検証するため、２つの評価を行った。１番目は、異なる NW-HW をサポートする際に新規開発が必要なコード量が実際に減少しているかの確認、２番目は、NAS Parallel Benchmarks を用いた既存通信ライブラリとの通信性能比較である。

第１の評価、第２の評価のために、IB を用いた LLC（LLC/IB と呼ぶ）と Tofu を用いた LLC（LLC/Tofu と呼ぶ）を実装した。さらに第２の評価のために、LLC/IB と LLC/Tofu を用いて、MPICH[17] の HCA を制御するプラグインモジュールを実装することで、MPI ライブラリを実装した（それぞれ MPICH/LLC/IB、MPICH/LLC/Tofu と呼ぶ）。LLC/IB は Intel C Compiler バージョン 13.0.0 20120731 を用いて-O2 オプションを付けてコンパイルした。LLC/Tofu は Fujitsu C Compiler バージョン 1.2.0 20130730 を用いて-O3 オプションを付けてコンパイルした。MPICH のバージョンは 3.1 で、configure のオプションは-enable-fast=O2, nochkmsg, notiming, ndebug を用いた。MPICH/LLC/IB の評価には**表 1** に示すパラメタを持つクラスタを用い、MPICH/LLC/Tofu の評価には**表 2** に示すパラメタを持つマシンを用いた。NAS Parallel Benchmarks からは BT、CG、EP、FT、IS、LU、MG、SP を用いた。MPICH/LLC/IB においてはデータセットは C、プロセス数は 8（BT と SP は 9）を用い、Intel C/Fortran Compiler バージョン 13.0.0 20120731 を用いて`-O3 -xavx` オプションを付けてコンパイルした。MPICH/LLC/Tofu においてはデータセットは D、プロセス数は 32（BT と SP は 36）を用い、Fujitsu Fortran Compiler バージョン K-1.2.0-15 あるいは Fujitsu C Compiler バージョン 1.2.0 20130730 を用いて、`-Kfast,parallel,ocl` オプションを付けてコンパイルした。また、MPICH/LLC/IB の通信性能比較対象として、MVAPICH2-2.0 を用い、MPICH/LLC/Tofu の通信性能比較対象として、Fujitsu MPI Library-1.2.0 を用いた。

表 3 MPICH-netmod/IB、LLC/IB、LLC/Tofu の異なる NW-HW をサポートする際に新規開発が必要なコード行数.

ライブラリ	新規開発が必要なコード行数（割合）	全体コード行数
MPICH-netmod/IB	12496 (100.0%)	12496
LLC/IB	6764 (50.2%)	13595
LLC/Tofu (参考)	4156 (58.6%)	7096

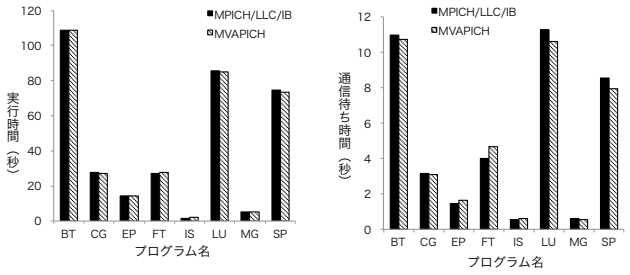

図 4 MPICH/LLC/IB の NAS Parallel Benchmark での実行時間と、通信待ち時間.

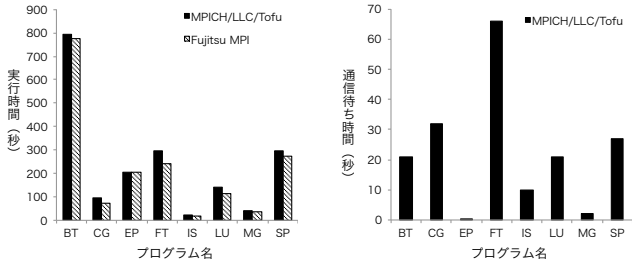

図 5 MPICH/LLC/Tofu の NAS Parallel Benchmark での実行時間と、通信待ち時間.

5.1 異なる NW-HW をサポートする際に新規開発が必要なコード行数

LLC/IB とほぼ同じ機能を持った通信ライブラリとしては、MPICH-3.1 の NW-HW 依存モジュールである `netmod` の IB 向け実装 (2014/6/18 版)[18]（MPICH-netmod/IB と呼ぶ) があり、LLC/IB と MPICH-netmod/IB を比較することで、異なる NW-HW をサポートする際に新規開発が必要なコード量を実際に削減できているか確かめる。**表 3** に示すように、MPICH-netmod/IB では全てが新規開発が必要なコードであるが、LLC/IB では全体の 50.2%が新規開発が必要なコードである。このことから、LLC/IB は実際に新規開発が必要なコード量を削減していることが確かめられる。

LLC/Tofu とほぼ同じ機能を持った通信ライブラリとしては、Fujitsu MPI がベースとしている、OpenMPI の NW-HW 依存モジュールである BTL の Tofu 向け実装があるが、ソースコードが公開されていないため比較はできない。

5.2 通信性能

NAS Parallel Benchmarks のプログラムの実行時間を

MPICH/LLC/IB(MPICH/LLC/Tofu) と既存実装とで比較することで、通信性能を評価する。MPICH/LLC/IB は MVAPICH/IB と比較し、MPICH/LLC/Tofu は Fujitsu MPI と比較する。実行時間に加えて、演算と通信のオーバーラップを行った後になお残った通信時間である、通信待ち時間を測定する。**図 4** に MPICH/LLC/IB を MVAPICH と比較した結果を示す。実行時間は MVAPICH に比較して最大 1.995%増加、平均 0.3094%減少している。この実行時間変化はほぼ通信待ち時間変化に起因している。通信待ち時間は MVAPICH に比較して最大 16.25%増加、平均 0.1125%減少している。

図 5 に MPICH/LLC/Tofu を Fujitsu MPI と比較した結果を示す。実行時間は Fujitsu MPI に比較して最大 34.74%、平均 15.05%増加している。なお、Fujitsu MPI のソースコードは公開されていないため、ソースコード変更が必要な通信待ち時間測定は行わなかった。

6. 結語

異なる NW-HW をサポートする際に新規開発が必要なコード量を削減する、ハイパフォーマンス向けコンピュータの通信ライブラリの設計を提案・評価した。ライブラリを、NW-HW に依存しない実装が可能な、中間表現コマンド・イベントを受け取り中間表現コマンド・イベントを発行するステートマシンであるプロトコルエンジンからなる NW-HW 非依存レイヤと、その指令により NW-HW に依存する動作をし、従って NW-HW に依存する実装がなされたモジュールからなる NW-HW 依存レイヤに分け、NW-HW 非依存レイヤになるべくコードを移動させることで新規開発が必要なコードを削減した。また、実際に動作する IB および Tofu 向け実装について NW-HW 依存コード・非依存コードへの切り分け具体例を説明した。

また、IB 向け実装で全体の 50.2%、Tofu 向け実装で全体の 58.6%のコードが新規開発が必要なコードであることを確認した。さらに、提案ライブラリを用いて MPI ライブラリを実装し、NAS Parallel Benchmarks を用いて既存実装と実行時間を比較した。IB 向け実装では MVAPICH と比較して実行時間は平均 0.3094%減少し、Tofu 向け実装では Fujitsu MPI と比較して実行時間は平均 15.05%増加することを確認した。今後の課題としては、既存実装の性能に近づけるための最適化、また QLogic (Intel) の IB HCA の API である PSM のサポートが挙げられる。

7. 謝辞

本論文の一部は、文部科学省「レイテンシコアの高度化・高効率化による将来の HPCI システムに関する調査研究」で実施された内容に基づくものである。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。本論文の結果の一部は、理化学研究所のスーパーコンピュータ「京」を利用して得られた。

参考文献

[1] Maruyama, T., Yoshida, T., Kan, R., Yamazaki, I., Yamamura, S., Takahashi, N., Hondou, M. and Okano, H.: Sparc64 VIIIfx: A New-Generation Octocore Processor for Petascale Computing, *IEEE Micro*, Vol. 30(2), pp. 30–40 (2010).

[2] Intel Corporation: Intel Xeon Phi Coprocessor Instruction Set Architecture Reference Manual, Vol. Reference Number 327364-001 (2012).

[3] InfiniBand Trade Association Std.: InfiniBand™ Architecture Specification, Vol. 1, Rel. 1.2.1 (2007).

[4] 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. ISC*, pp. 498–507 (2014).

[5] Gropp, W., Lusk, E., Doss, N. and Skjellum, A.: A High-Performance, Portable Implementation of the MPI, Message Passing Interface Standard, *Parallel Computing*, Vol. 22(6), pp. 789–828 (1996).

[6] Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L. and Woodall, T. S.: Open MPI: Goals, concept, and design of a next generation MPI implementation, *In Proc. 11th European PVM/MPI Users' Group Meeting*, pp. 97–104 (2004).

[7] Carns, P., III, W. L., Ross, R. and Wyckoff, P.: BMI: A Network Abstraction Layer for Parallel I/O, *In Proc. Workshop on Communication Architecture for Clusters (IPDPS Workshops)* (2005).

[8] Ajima, Y., Sumimoto, S. and Shimizu, T.: Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers, *IEEE Computer*, Vol. 42(11), pp. 36–40 (2009).

[9] : The Parallel Virtual File System, version 2, `http://www.pvfs.org/pvfs2`.

[10] Sur, S., Jin, H.-W., Chai, L. and Panda, D. K.: RDMA Read Based Rendezvous Protocol for MPI over InfiniBand: Design Alternatives and Benefits, *In Proc. PPoPP*, pp. 32–39 (2006).

[11] DAT Collaborative: User-Level Direct Access Transport APIs (uDAPL), `http://www.datcollaborative.org/udapl.html`.

[12] Hemmert, S., Barrett, B. W. and Underwood, K.: Using Triggered Operations to Offload Collective Communication Operations, *In Proc. EuroMPI*, pp. 249–256 (2010).

[13] Kumar, S., Mamidala, A. R., Faraj, D. A., Smith, B., Blocksome, M., Cernohous, B., Miller, D., Parker, J., Ratterman, J., Heidelberger, P., Chen, D. and Steinmacher-Burow, B.: PAMI: A Parallel Active Message Interface for the Blue Gene/Q Supercomputer, *In Proc. IPDPS*, pp. 763–773 (2012).

[14] Takahashi, T., Sumimoto, S., Hori, A., Harada, H. and Ishikawa, Y.: PM2: A High Performance Communication Middleware for Heterogeneous Network Environments, *In Proc. SC* (2000).

[15] Liu, J., Wu, J. and Panda, D. K.: High Peformance RDMA-Based MPI Implementation over InfiniBand, *In*

Proc. ICS, pp. 295–304 (2003).
[16] Hatanaka, M., Hori, A. and Ishikawa, Y.: Optimization of MPI Persistent Communication, *In Proc. EuroMPI*, pp. 79–84 (2013).
[17] Argonne National Laboratory: MPICH, `http://www.mpich.org`.
[18] Takagi, M., Nakamura, Y., Hori, A., Gerofi, B. and Ishikawa, Y.: Revisiting Rendezvous Protocols in the Context of RDMA-Capable Host Channel Adapters and Many-Core Processors, *In Proc. EuroMPI*, pp. 85–90 (2013).

付　　録

A.1 プロトコル実装

この節では LLC が提供するプロトコルの IB 向け、Tofu 向け実装の詳細を説明する。

A.1.1 ブートストラップリモートメモリアクセス

IB 向け実装のプロトコルを**図 A·1** を用いて説明する。ピアが n 個存在するとする。あるピアを整数を用いてピア i のように表現する。ピア p がブートストラップリモートメモリアクセスを準備しているとする。

ピア i がピア j に送信を行えるようにするためのブートストラップ用 QP で、ノード i に存在するものを $\mathrm{QP}_{i,j}$ と表す。通信相手のピアのネットワークアドレスを管理用データストアの備える KVS を用いて取得する。

I1 ピア p は初期化時に、ブートストラップ用メモリエリアを仮想・物理アドレス変換テーブルへ登録し、鍵を取得し、これらを管理用データストアに登録する。ピア p はピア $0, 1, \cdots, p-1, p+1, \cdots, n-1$ と通信するために、n-1 個の QP からなる QP 集合 $\{\mathrm{QP}_{p,i} | i = 0, 1, \cdots, p-1, p+1, \cdots, n-1\}$ を作成する。またピア p の GID、LID、$\mathrm{QP}_{p,i}, i = 0, 1, \cdots, p-1, p+1, \cdots, n-1$ の QP 番号と鍵を管理用データストアに登録する。

I2 他ピアの KVS へのデータ登録完了を待つため、バリア同期を行う。

I3 ピア p は $\mathrm{QP}_{p,i}$ をピア i と通信可能にするために、ピア i の HCA のポートのアドレス GID、LID と、$\mathrm{QP}_{i,p}$ の QP 番号を管理用データストアから取得して、$\mathrm{QP}_{p,i}$ に入力する。また、ピア i のブートストラップ用メモリエリアの仮想アドレスと鍵を管理用データストアから取得する。これを、$i = 0, \cdots, p-1, p+1, \cdots, n-1$ について繰り返す。

I4 他ピアが通信可能状態になるのを待つためバリア同期を行う。

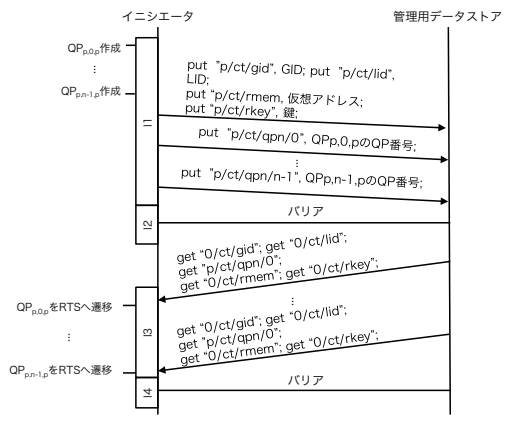

図 A·1 ブートストラップリモートメモリアクセス準備を行う IB 向けプロトコルのタイムラインダイアグラム.

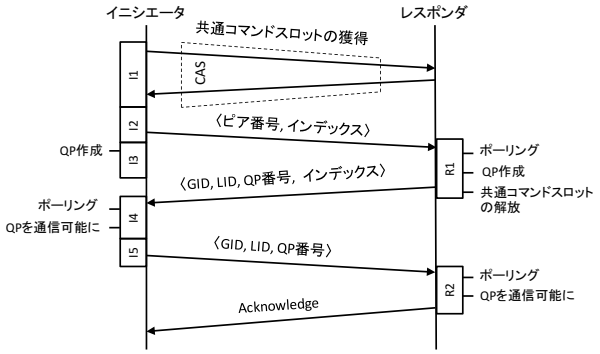

図 A·2 コネクションプロトコルの IB 向け実装のタイムラインダイアグラム.

I4 他ピアが通信可能状態になるのを待つためバリア同期を行う。

Tofu での実装を説明する。ピア p がブートストラップリモートメモリアクセスを準備しているとする。B1. ピア p は初期化時に、ブートストラップ用メモリエリアを仮想・物理アドレス変換テーブルへ登録し、受信側がこのテーブルを参照するためのタグを n 個のピアで共通に使用する静的な値に設定する。

A.1.2 コネクション

IB 向け実装のプロトコルを**図 A·2** を用いて説明する。通信を行おうとする際に、送信側ピア A から受信側ピア B へのコネクションが確立していない場合に、コネクションプロトコルが実行される。以降ピア A をイニシエータ、ピア B をレスポンダと呼ぶ。

I1 イニシエータはレスポンダのメモリ上のブートストラップ用メモリエリアに Compare-and-swap(CAS) コマンドを送り、同じくブートストラップ用メモリエリア上に存在するコネクションプロトコル用の共通コマンドスロットにロックをかける。共通コマンドスロットは一つしか存在せず、この後のステップでプロトコルのトランザクションごとのスロットを割り当てる。

I2 イニシエータはレスポンダが使用する、イニシエータのブートストラップ用メモリエリアに存在するコネクションプロトコル用のコマンドスロットを割り当て、そのインデックスとピア番号を共通コマンドスロットに書き込む。

I3 イニシエータは QP を作成する。

R1 レスポンダはポーリングによって共通コマンドスロットへの書き込みを検知し、QP を作成し、レスポンダのメモリ上にあるイニシエータが書き込むコネクションプロトコル用のコマンドスロットを確保し、イニシエータ側での QP 作成のための情報、すなわち GID、LID、QP 番号の情報、さらにレスポンダのメモリ上に存在するイニシエータが書き込むコマンドスロットのインデックスを、イニシエータが I2 で通知したコネクションプロトコル用のコマンドスロットに書き込む。イニシエータが I2 で書き込んだ共通

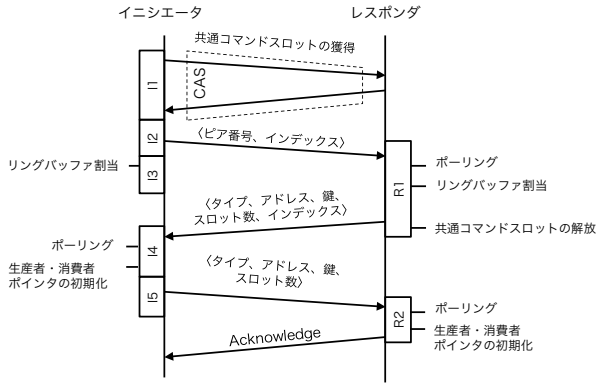

図 A·3 バッファ割り当ての IB 向けプロトコルのタイムラインダイアグラム.

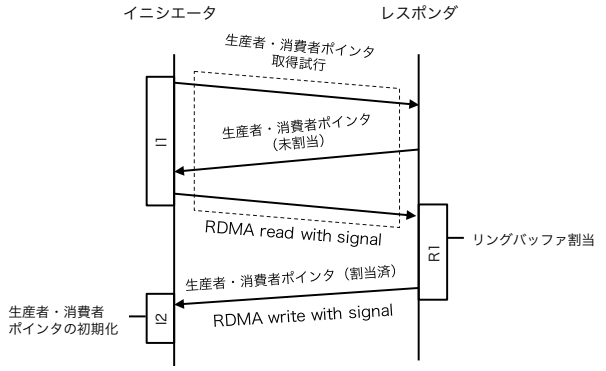

図 A·4 バッファ割り当ての Tofu 向けプロトコルのタイムラインダイアグラム.

コマンドスロットを解放する。

I4 イニシエータはポーリングによってレスポンダの書き込みを検知し、レスポンダが書き込んだ情報を用いて QP を送信可能状態にする。

I5 イニシエータはレスポンダ側での QP 作成のための情報、レスポンダが R1 で通知したコネクションプロトコル用のコマンドスロットに書き込む。

R2 レスポンダはポーリングによってイニシエータの書き込みを検知し、イニシエータが書き込んだ情報を用いて QP を送信可能状態にし、acknowledge をコマンドスロットに書き込む。

A.1.3 バッファ割当

IB 向け実装のプロトコルを**図 A·3** を用いて説明する。IB 向け実装ではバッファ割り当てのプロトコルは、コネクションプロトコルに含まれる形で実装している。そこで、バッファ割り当て部分だけを抜き出して説明する。

I1 イニシエータはレスポンダのメモリ上に存在するブートストラップ用メモリエリアに Compare-and-swap コマンドを送り、同じブートストラップ用メモリエリア上に存在するコネクションプロトコル用の共通コマンドスロットにロックをかける。

I2 イニシエータはレスポンダが使用する、イニシエータのブートストラップ用メモリエリアに存在するコネクションプロトコル用のコマンドスロットを割り当て、そのインデックスとピア番号を共通コマンドスロットに書き込む。

I3 イニシエータはレスポンダが書き込む、イニシエータのメモリ上に存在するリングバッファを割り当てる。

R1 レスポンダはポーリングによって共通コマンドスロットへの書き込みを検知し、イニシエータが書き込む、レスポンダのメモリ上に存在するリングバッファを割り当てる。イニシエータが書き込む、レスポンダのメモリ上に存在するコネクションプロトコル用のコマンドスロットを確保し、イニシエータがレスポンダのメモリ上に存在する中間バッファにデータを送信するための情報、すなわち、リングバッファのタイプ、仮想アドレス、鍵、スロット数、さらにイニシエータが書き込むレスポンダ側に存在するコマンドスロットのインデックスを、イニシエータが I2 で通知したコネクションプロトコル用のコマンドスロットに書き込む。このタイプは、リングバッファのメモリが逼迫した際に複数のイニシエータで共有するリングバッファを割り当てる為に用いるが、紙幅の都合上説明を省略する。イニシエータが I2 で書き込んだ共通コマンドスロットを解放する。

I4 イニシエータはポーリングによってレスポンダの書き込みを検知し、レスポンダが書き込んだ情報を用いてイニシエータのメモリ上に存在する生産者・消費者ポインタを初期化する。

I5 イニシエータはレスポンダがイニシエータのメモリ上に存在する中間バッファにデータを送信するための情報を、レスポンダが R1 で通知したコネクションプロトコル用のコマンドスロットに書き込む。

R2 レスポンダはポーリングによってイニシエータの書き込みを検知し、イニシエータが書き込んだ情報を用いてレスポンダのメモリ上に存在する生産者・消費者ポインタを初期化し、acknowledge をコマンドスロットに書き込む。

Tofu 向け実装のプロトコルを**図 A·4** を用いて説明する。

I1 イニシエータは RDMA read コマンドを用いて、レスポンダのメモリ上に存在する生産者・消費者ポインタを取得しようとする。初期化直後はこのメモリエリアには、未割り当てを示すデータが入っており、イニシエータはこれを検知してレスポンダからの制御パケットを待つ。

R1 レスポンダはイニシエータの生産者・消費者ポインタ取得試行を検知して、イニシエータが書き込む、レスポンダのメモリ上に存在するリングバッファを割り当て、イニシエータへの通知付き RDMA write コマンドを用いて、レスポンダのメモリ上に存在する、生産者・消費者ポインタの値をイニシエータに通知する。

I2 イニシエータはレスポンダからの生産者・消費者ポインタ通知を、NW-HW の通知機構を通じて検知して、イニシエータのメモリ上に存在する生産者・消費者ポインタを初期化する。