Vol.2017-HPC-158 No.13
2017/3/9

DPDK によるデータ転送を用いた MPI 通信の実装

島田明男† 早坂光雄†

概要：MPI は並列計算のための通信規格であり，HPC アプリケーションの開発に広く用いられている．現在，MPI の実装として，様々な MPI ライブラリが OSS として公開されている．MPI は，あくまで通信規格であり，データ転送を実行するための方式やデバイスまでは定義していない．多くの MPI ライブラリは，データ転送を実行するためのレイヤが独立した構造をとっており，ユーザの環境に合わせてデータ転送方式やデバイスを MPI プログラムの実行時に柔軟に切り替える事ができる．本研究は，MPI ライブラリの新たなデータ転送モジュールとして，DPDK によるデータ転送モジュールを提案する．DPDK を用いると，ユーザ空間から直接 NIC を操作し，高速にデータ転送を行うことができる．Open MPI に DPDK を用いるデータ転送モジュールを追加し，ベンチマークソフトウェアによって，簡易評価をおこなった。その結果，TCP/IP ソケットによるデータ転送を用いた場合と比較し，最大で 77%，MPI 通信の性能を向上することができた．

キーワード：DPDK，MPI，Open MPI

MPI Implementation using DPDK Framework

AKIO SHIMADA† MITSUO HAYASAKA†

Abstract: MPI is a communication standard for parallel computation and widely used for developing HPC applications. Currently, a variety of MPI libraries is published as OSS. MPI is just a communication standard and does not define a method and device for data transmission. In case of major MPI libraries, data transmission layer is independent from other part of an MPI library. Then, users can switch a data transmission method and device according to their environments when executing MPI applications. In this research, we propose a new data transmission module leveraging DPDK in MPI library. DPDK enables user programs to directly access and use the functions of NIC. As the result, fast data transmission can be achieved. We embedded a DPDK based transmission module into Open MPI and evaluated it by benchmark software. The results show that the proposed data transmission module can improve MPI communication performance by up to 77%, compared with a data transmission module using TCP/IP socket.

Keywords: DPDK, MPI, Open MPI

1. はじめに

Message Passing Interface (MPI)[1]は，並列計算のための通信規格であり，並列アプリケーションの実装に広く用いられている．MPI はライブラリレベルの並列化をサポートしており，並列処理を実行する各プロセスが MPI ライブラリの提供する通信 API を用いて，並列処理の実行に必要な計算データを互いに送受信することができる．Open Source Software (OSS)として Open MPI[2]や MPICH[3]といった MPI ライブラリが公開されている．

MPI は，あくまで通信規格の定義であり，MPI 通信によって送受信されるデータが，どのような方式やデバイスで転送されるかは MPI ライブラリの実装依存となっている．多くの MPI ライブラリは，データ転送を行うレイヤが独立しており，レイヤ内に用意されたデータ転送モジュールを切り替えることで，MPI 通信のデータ転送に様々な方式やデバイスを用いることが可能になっている．例えば Open MPI は，データ転送を行うレイヤを Byte Transfer Layer(BTL)と呼び，他のレイヤから分離独立させている．また，BTL レイヤには，IB Verbs によって Infiniband 通信を実行するモジュール（openib btl）や TCP/IP ソケットによって Ethernet 通信を実行するモジュール（tcp btl）等が用意されている．

本研究は Ethernet をノード間のデータ転送に用いる環境を対象とする．複数ノードで並列計算を高速に実施するには，ノード間通信がボトルネックとなるため，より高速なノード間通信が求められる．そこで本研究は，MPI 通信のデータ転送に Data Plane Development Kit (DPDK)[4]を用いることを提案する．DPDK は Network Interface Card (NIC)の機能をユーザ空間から直接利用可能にし，高速に Ethernet 通信を行うためのフレームワークである．通常，Ethernet 通信には TCP/IP ソケットが用いられる．TCP/IP ソケットを用いる Ethernet 通信はデータ転送時に OS カーネルを経由する必要があり，アプリケーションから OS カーネルへのコンテキストスイッチが通信のオーバヘッドとなる．対して，DPDK を用いると，アプリケーションを実行するユーザ空間のプロセスが OS カーネルを経由せずに Ethernet 通信を実行することが可能になるため，ノード間のデータ転送を高速に行うことができる．現在，様々なベンダの NIC が DPDK に対応している．そこで， Open MPI

† (株)日立製作所 研究開発グループ
Research & Development Group, Hitachi Ltd.

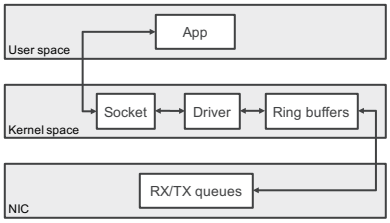

図 2 Linux のパケット通信

の BTL レイヤに DPDK を用いるデータ転送モジュールとして「dpdk btl」を追加し，DPDK によるデータ転送をサポートする MPI ライブラリを開発した．

DPDK を用いる場合，Ethernet パケットの受信の検知に割り込みを用いることができない．そのため，ユーザプロセスがポリングでパケットの受信を検知して受信処理を実行する必要がある．本研究は，小さいサイズのデータを送受信する非同期通信と大きいサイズのデータを送受信する同期通信でプロセスを分け，前者をポリング専用のプロセス、後者を MPI アプリケーションの実行プロセス（MPI プロセス）にポリング処理させることで，パケットの受信処理を効率的に行う方式を提案した．

ベンチマークによって MPI 通信の通信遅延を評価したところ, dpdk btl を用いた場合, tcp btl を用いた場合と比べ, 通信遅延を最大で約 77%低減することができた．

2. DPDK

2.1 DPDK の概要

DPDK は NIC を用いた通信を，OS カーネルを経由せずに実行するためのフレームワークである．図 2 に示すように, Linux®[1]における従来の NIC を用いた通信は TCP/IP ソケットを経由してユーザ空間のアプリケーションと OS カーネル内のドライバがデータを授受する．OS カーネル内のドライバは，リングバッファを経由して NIC の送受信キュー（RX/TX キュー）に送受信リクエストを挿入する．NIC は挿入された送受信リクエストにしたがって，他のノードの NIC とパケットを送受信する．通信を実行する際に，必ずユーザ空間から OS カーネル空間へのコンテキストスイッチとデータコピーが発生し, それがオーバヘッドとなる．

対して，DPDK による通信は，図 1 に示すように OS カーネルを経由せずにアプリケーションが直接 NIC の機能を利用することができるため高速な通信が可能となる．UIO ドライバが NIC のメモリ空間とアプリケーションを実行するプロセスのメモリ空間をマッピングすることでユーザ空間のアプリケーションが NIC の機能を，OS カーネルを経由せずに利用することが可能になる．アプリケーションが NIC の機能を利用する際は DPDK の提供するライブラリを用いる．

1 Linux は、Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。

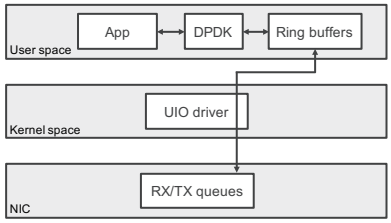

図 1 DPDK のパケット通信

DPDK は共有メモリとして，アプリケーションがアクセス可能なメモリプールを作成する．メモリプールからはアプリケーションのデータを NIC に転送する際に用いる中間バッファと，アプリケーションがデータの送受信リクエストを NIC に発行するためのリングバッファが確保される．リングバッファに登録された通信リクエストは，head/tail ポインタによって管理される．アプリケーションは共有メモリプールから自身の利用するメモリを確保することが可能である．

DPDK フレームワークは，ユーザ空間で動作するため，パケットの受信処理に割り込みおよび割り込みハンドラを用いることが出来ない．よって，パケットの受信を CPU によるポリングで検知し，受信処理を実行する必要がある．ユーザプログラムにポリングのための機能を提供するコンポーネントを Poll Mode Driver (PMD)と呼ぶ．

DPDK の提供するネットワーク機能は OSI 参照モデルにおける L2 レイヤ（データリンク層）に相当する．よって，通信を実行するためには，通信先の NIC のポートの MAC アドレスを特定する必要がある．また，L3/L4 レイヤ（TCP/IP）に相当する処理を実行する場合は， DPDK ライブラリの上位に，ユーザ自身が TCP/IP の処理を実装する必要がある．

2.2 パケットの受信処理

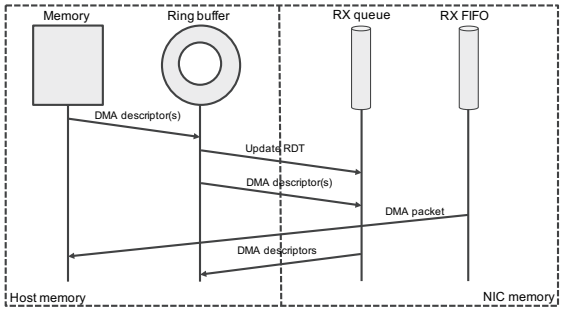

図 3 パケットの受信処理

エラー! 参照元が見つかりません。は DPDK を用いるパケットの受信処理を示している．まず，アプリケーションが DPDK の受信用 API を呼び出すと, 受信データを格納するための中間バッファが共有メモリプールから確保される．そして，中間バッファのアドレスがリングバッファに登録される．次に DPDK ライブラリは Receive Descriptor Tail (RDT) register を更新して受信リクエストが発行されたことを NIC に通知し，受信キュー（RX キュー）に中間バッ

ファのアドレスが登録される．NIC が外部から受信したパケットは, 登録された中間バッファに Direct Memory Access (DMA)によってコピーされる．DMA によってデータがコピーされた中間バッファのアドレスは，リングバッファの head ポインタを移動することで，通知される．アプリケーションは，発行した受信リクエストが実行されたかどうかを PMD によって検知する．受信を検知したら，アプリケーションが中間バッファから自身の受信バッファに受信したデータをコピーし，受信処理が完了する．

2.3 パケットの送信処理

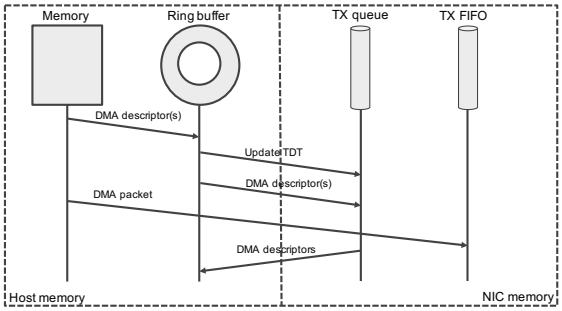

図 4 パケットの受信処理

エラー! 参照元が見つかりません。は DPDK を用いるパケットの送信処理を示している．まず，アプリケーションが DPDK の送信用 API を呼び出すと，共有メモリプールから中間バッファが確保され，送信対象のデータがアプリケーションの送信バッファから中間バッファにコピーされる．そして，中間バッファのアドレスがリングバッファに登録される．次に DPDK ライブラリは Transmit Descriptor Tail (TDT) register を更新して送信リクエストが発行されたことを NIC に通知し，中間バッファのアドレスを送信キュー（TX キュー）に登録する．NIC は，登録された中間バッファから Direct Memory Access (DMA)によって送信対象のデータを自身のメモリにコピーし，当該データを外部のノードに送信する．DMA によってデータがコピーされた中間バッファのアドレスは, リングバッファの head ポインタを移動することで，通知される．

2.4 flow director

NIC が受信したパケットをどの RX キューに登録するかは，flow director によって制御することができる．flow director は，特定の IP アドレスからのパケットや，特定のポート番号に対するパケットを，どの RX キューに振り分けるか制御するフィルタを登録することができる．flow director は, 受信したパケットの L3/L4 ヘッダとフィルタを参照し，受信したパケットを登録する RX キューを選択する．キューの判別は，キューに割り当てられた ID によって行う．

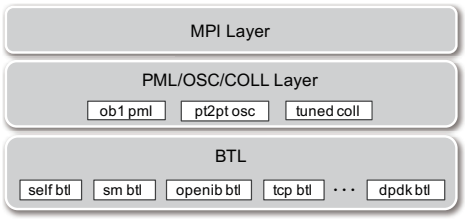

図 4 Open MPI のレイヤ

3. Open MPI

3.1 Open MPI の概要

Open MPI は MPI の実装の一つであり，OSS として公開されている．Open MPI は図 4 に示すようなレイヤ構成になっている. Open MPI の各レイヤには複数のモジュールが用意されており, モジュールを切り替えることで MPI プログラムを実行する際に，実装を切り替え可能になっている．

MPI レイヤはアプリケーションに通信 API を提供するレイヤである．並列処理を実行する MPI プロセスは，このレイヤが提供する通信 API を用いて, 他の MPI プロセスと通信する. PML/OSC/COLL レイヤはそれぞれ, point-to-point 通信/片方向通信/集団通信を管理するためのレイヤである. 送受信リクエストや送受信先の管理等を行い，データの送受信を下位レイヤに要求する. ob1 pml, pt2pt osc, tuned coll といったモジュールが用意されている．

BTL レイヤは上位レイヤから要求に従って, データを他の MPI プロセスに転送する役割を負っている．ノード内データ転送のためのモジュールとして self btl や sm btl がモジュールとして用意されている．self btl は同一プロセスを対象とするデータ転送に用いられる．sm btl は共有メモリによるノード内データ転送をサポートするモジュールである. ノード間データ転送のためのモジュールとしては，openib btl や tcp btl が用意されている．openib は IB verbs による Infiniband 通信を, tcp btl は TCP/IP ソケットによる Ethernet 通信をデータ転送に利用する．

3.2 データの送受信

本節では Open MPI におけるデータの送受信処理について述べる．説明は，MPI_Send/Recv を用いた Peer-to-Peer 通信を例に行う．Open MPI は，非同期通信である eager 通信と同期通信である rendezvous 通信をサポートしている．まず，eager 通信について述べる．

3.2.1 eager 通信

Open MPI は, 送信側が発行した送信処理（MPI_Send 等）に対応する受信処理（MPI_Recv 等）が呼び出されていなくても、送信側はデータを送信することができる．このような非同期通信を eager 通信と呼ぶ．図 6 は eager 通信が実行された時のデータ送受信処理を示している．まず，送信側でアプリケーションが MPI_Send を実行すると，PML レイヤの送信 API が呼ばれ，MPI 通信を制御するための情

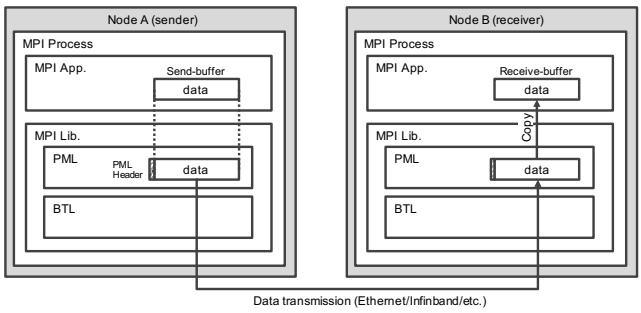

図 6 データの送受信（eager 通信）

報(送信先のランクやタグ, 送受信バッファのアドレス等)がヘッダとして送信対象データに付与される．そして，PML レイヤは，BTL レイヤの送信 API を呼び出し，データの送信をリクエストする．リクエストを受け取った BTL レイヤは送信先の BTL レイヤにデータを送信する．

受信側の MPI ライブラリは, MPI_Wait や MPI_Recv 等のブロッキング API を MPI アプリケーションが呼び出すと，内部で progress 処理を実行する．progress 処理では，PML レイヤおよび BTL レイヤによるデータの受信処理が行われる. まず BTL レイヤが送信プロセスからデータを受信し，受信したデータを PML レイヤが用意した eager 通信用バッファに格納する. PML レイヤは受信したデータに対応する受信処理（MPI_Recv や MPI_IRecv）が発行されているかどうかを，データに付与されたヘッダ情報と受信リクエストを格納するキューを参照してチェックする（受信リクエストは MPI_Recv 等の受信 API が呼ばれた際に発行される）. もし対応する受信処理が発行されていたら受信バッファにデータをコピーして, 受信リクエストのステータスを”完了”にする．対応する受信処理がなければ，受信したデータを unexpected キューに挿入する．unexpected キューで管理されるデータは，対応する受信処理が発行された際に，受信バッファにコピーされる．MPI ライブラリは progress 処理の実行契機となった MPI 関数の処理が終わるまで, progress 処理を繰り返す.

3.2.2 rendezvous 通信

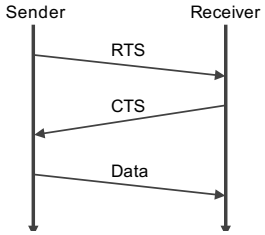

図 7 rendezvous 通信の概要

Open MPI は，同期通信である rendezvous 通信をサポートする．rendezvous 通信は，送信処理に対応する受信処理が発行されてから，データの送受信を開始する．図 7 は rendezvous 通信の概要を示している．まず送信側の MPI ライブラリが Ready-to-send(RTS)メッセージを受信プロセスに送信する．受信側の MPI ライブラリは，受信した RTS に対応する受信処理がアプリケーションから発行されたら, Clear-to-send(CTS)メッセージを送信プロセスに送信する．送信プロセスが CTS を受信した時点で同期が完了し, データの送受信が開始される．RTS および CTS は MPI ライブラリ内で作成され，eager 通信で送受信される．

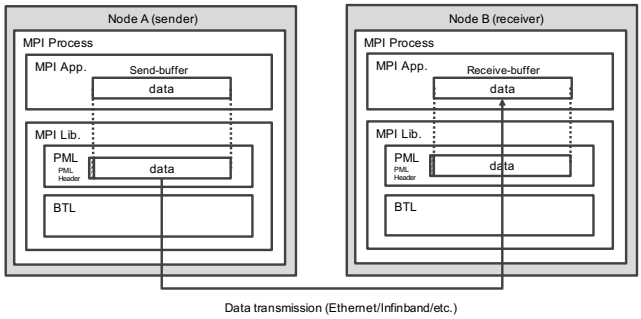

図 5 データの送受信（rendezvous 通信）

図 5 は rendezvous 通信におけるデータの送受信を示している．まず，送信側の MPI ライブラリは eager 通信のときと同様の方法でデータを受信側の BTL レイヤに送信する．

受信側では，eager 通信のときと同様に，progress 処理のなかで PML レイヤおよび BTL レイヤによるデータの受信処理が行われる. まず BTL レイヤが送信プロセスからデータを受信し，受信バッファにデータを格納する．そして，PML レイヤには，受信したデータの情報のみを受け渡す. PML レイヤは, BTL レイヤから渡された情報と受信リクエストを格納するキューを参照して，該当する受信リクエストのステータスを”完了”にする. eager 通信のときのように，eager 通信用バッファと unexpected キューを用いてデータを一時的に保存することはなく，直接受信バッファにデータがコピーされる.

eager 通信と rendezvous 通信を比較すると, 送受信対象のデータが小さいときは同期のコストが無い分 eager 通信の方が高速になる. 一方, データサイズが大きいケースでは，eager 通信用バッファから受信バッファにデータをコピーするコストが大きくなり，rendezvous 通信の方が高速になる．Open MPI は，eager 通信と rendezvous 通信を切り替えるデータサイズ（eager_limit）を MPI アプリケーションの実行時に切り替えることができる.

4. dpdk btl の設計と実装

本研究は Open MPI の BTL レイヤに DPDK によるデータ転送をサポートするモジュールである dpdk btl を追加した(図 5)．本章は dpdk btl の設計と実装について述べる.

4.1 設計

第 1 章で述べた通り，DPDK を用いる場合，Ethernet パケットの受信の検知に割り込みを用いることができないため，ユーザプロセスがポリングでパケットの受信を検知して受信処理を実行する必要がある．dpdk btl に，パケット

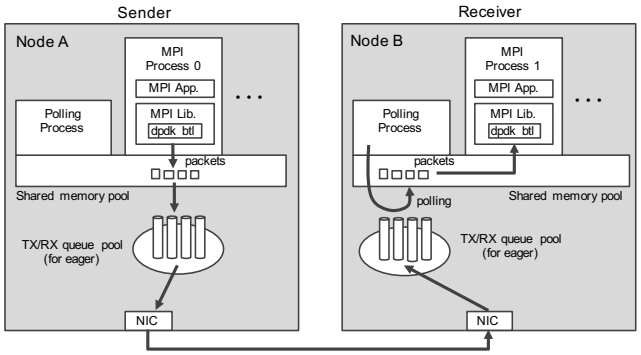

図 9 dpdk btl の eager 通信

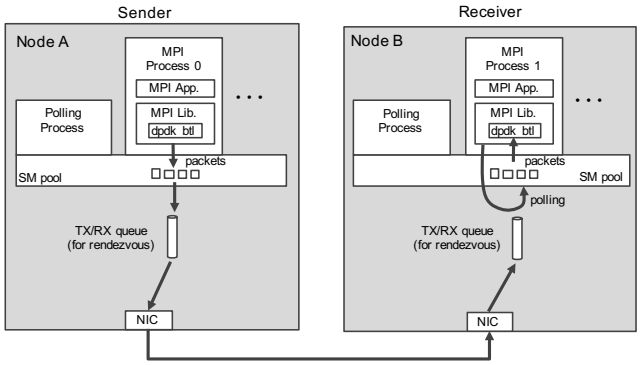

図 8 dpdk btl の rendezvous 通信

の受信処理を実装する方式として以下の 2 つが考えられる．

(1) **MPI プロセス方式**

MPI プロセス（MPI アプリケーションを実行するプロセス）自身にポリングを実行させる．

(2) **専用プロセス方式**

ポリングを実行するための専用プロセスをノード上で起動する．

MPI プロセス方式は，MPI プロセス自身にポリングを実行させる．ポリングを実行できるのは MPI ライブラリが実行されている間のみである．NIC のキューの長さは固定の値に決まっているため，ポリングを実行可能なタイミングが限定される MPI プロセス方式では，キュー溢れによるパケットロスが発生する確率が高くなる．

専用プロセス方式は，ポリング専用のプロセスが常にポリングを実行し，パケット受信の検知を行う．常にポリングを実行できるので，上記のキュー溢れを回避することができる．しかし，専用プロセスはノード内の全 MPI プロセスに送られてきたパケットを処理することになるため，ある MPI プロセスに多量のパケットが送信されてきた場合，そのパケットでキューが溢れ，他の MPI プロセスに送信されてきたパケットがパケットロスになる確率が高くなる．

そこで本研究は，送受信するデータサイズが小さい eager 通信に専用プロセス方式を適用し，送受信するデータサイズが大きい rendezvous 通信に MPI プロセス方式を適用するパケット受信処理を提案する．eager 通信時は，受信プロセスが MPI ライブラリを実行している保証がないため，専用プロセスにポリングを実行させ，キュー溢れによるパケットロスを回避する．一方，rendezvous 通信時は受信プロセスが MPI ライブラリを実行していることが保証されるため，MPI プロセスにポリングを実行させる．大きなデータのパケットは MPI プロセス自身に処理させることで，専用プロセスの通信キューが特定の MPI プロセスに向けて送られて来たパケットで溢れるのを回避し，効率的にパケットの受信処理を行う．

特定の MPI プロセスに多量の小さいデータのパケットが送信されてきた場合も，キュー溢れが発生する可能性が高くなる．このキュー溢れに対処するため，ポリング専用プロセスに複数のキューを割り当て，パケットの格納先をキュー間で分散させる．これにより，NIC 内の固定だったキュー長を大きくした場合と同じ効果を発揮させ，パケットロスを抑制する．

4.2 実装

4.2.1 eager 通信

図 9 は dpdk btl における eager 通信を示している．まず，送信プロセスの dpdk btl が上位レイヤから受け取った送受信データからパケットを作成し，DPDK が用意する共有メモリプール上の中間バッファに格納する．作成した各パケットは，どの送受信データのパケットかを示す情報をパケットのヘッダに記しておく．そして，送信リクエストを eager 通信用の TX キューに登録する．

dpdk btl は eager 通信に用いる TX/RX キューをあらかじめ確保しておき，キューのプールを作成する．そして，データの送信実行時にプールからいずれかの TX キューを獲得し，通信に用いる．キューの確保は MPI_Init の延長で実行される btl_init と呼ぶ BTL レイヤの関数内で，あらかじめ行っておく．MPI_Init は，MPI ライブラリに初期化処理を実行させるための関数で，通常，アプリケーションの起動後に 1 度だけ呼び出される．

送信プロセスの dpdk btl は，flow director の機能を用いて受信側のキュープール内にある何れかの RX キューを通信先のキューに指定する．各ノードで eager 通信用に確保される TX/RX キューの ID は，btl_init の実行時に MPI プロセス同士で交換しておく．また，各ノードで使用される NIC のポートの MAC アドレスの情報も，このとき交換しておく．本研究は，キューの ID と MAC アドレスの交換を TCP/IP ソケットによる通信で実装した．

受信側では，ポリング専用プロセスが，プール内に存在する eager 通信用の RX キューに対してポリングを実行し，パケットの受信を検知する．ポリング専用プロセスは，MPI アプリケーションの実行前に，事前に起動しておく必要がある．ポリング専用プロセスは複数の MPI プロセスと通信を行うので，eager 通信用の RX キューを複数割当て，キュー溢れによるパケットロスを軽減できるようにした．ポリング専用プロセスは，パケットの受信を検知したら，パケ

ットを格納している中間バッファのアドレスを受信プロセスに通知する．アドレスの通知にはプロセス間通信を用いる．本研究の実装は，アドレスの通知に共有メモリによるプロセス間通信を用いた．受信プロセスの dpdk btl は，progress 処理の中でパケットが受信されていないかどうかを，ポリングプロセスからの通知によりチェックする．もしパケットを受信していたら，当該パケットから送受信データを復元し，上位レイヤの用意する eager 通信用バッファにデータをコピーし，受信処理を完了する．

Ethernet による通信では，通信経路上のスイッチやルーターで起きる輻輳やエンドポイントでのキュー溢れ等の理由によりパケットロスが発生する．dpdk btl は，送信プロセスとポリングプロセスで ack パケットのやりとりを行い，パケットロスに対する制御を実施する．送信プロセスは，送受信データを含むパケットを送信したら，送信したパケットに対する ack パケットがポリングプロセスから送信されてくるのを待つ．送信したパケットには，ack パケットの受信に利用する RX キューの ID をヘッダ情報として記しておく．ポリングプロセスは，パケットの受信を検知したら，当該パケットに対応する ack パケットを作成し，送信プロセスに送信する．このとき，受信したパケットのヘッダが示す RX キューに対して送信を行う. 送信プロセスは，ack パケットを受信したら送信処理を完了する．送信プロセスは，ある程度の時間 ack パケットが返ってこなかったら，送受信データを含むパケットをポリングプロセスに再送する．あらかじめ設定しておいた再送回数を超えたら，通信エラーとして処理する．

4.2.2 rendezvous 通信

図 8 は dpdk btl における rendezvous 通信を示している．dpdk btl の rendezvous 通信では，ポリングプロセスを経由せずに，MPI プロセス同士が直接データを送受信する．送信プロセスの dpdk btl は、上位レイヤから受け取った送受信データからパケットを作成し，DPDK が用意する共有メモリプール上の中間バッファに格納する．各パケットのヘッダには，受信プロセスの受信バッファのアドレスとパケット内のデータが送受信データのどの部分に該当するかを示すオフセット値を記録しておく．受信バッファのアドレスは，上位レイヤから受け取ることができる．そして，送信リクエストを rendezvous 通信用の TX キューに登録する．dpdk btl は rendezvous 通信に用いる TX/RX キューをあらかじめノード上で動作する各 MPI プロセスに割り当てておく．キューの割当は btl_init 内で実行する．各 MPI プロセスが使用している TX/RX キューの ID は，btl_init の実行時に MPI プロセス同士で交換しておく．キューの ID の交換は TCP/IP ソケットによる通信で行う. 送信プロセスの dpdk btl は，flow director の機能を用いて受信プロセスに割り当てられた RX キューを通信先のキューに指定する．

受信プロセスの dpdk btl は, progress 処理の中でポリングを実行し，パケットの受信を検知する．パケットの受信を検知したら，当該パケットのデータを，パケットのヘッダ内の情報を参照し，受信バッファに格納していく．全てのデータを受信バッファに格納したら，受信処理を終える．

rendezvous 通信は，eager 通信と同様に，MPI プロセス間でパケットロスに対する制御を実施する. eager 通信のときと同様の方法で ack パケットの授受をおこない，パケットロスに対処する．

5. 評価

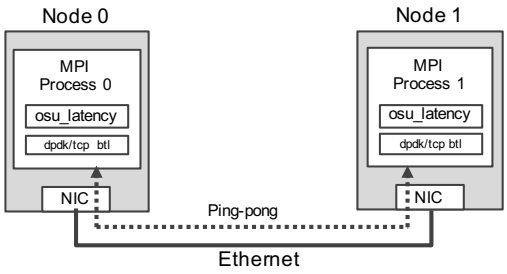

図 10 実験環境

dpdk btl の評価を実施した．評価環境を表 1 に示す。表で示すノードを 2 台用意し，図 10 に示すように Ethernet によって直結して通信性能を測定した．測定には OSU Micro-Benchmarks[5]の osu_latency ベンチマークを用いた．OSU Micro-Benchmarks は，MPI ライブラリの通信性能を測定するために広く用いられている．osu_latency ベンチマークは　各ノード上で 1 個の MPI プロセスを起動し，MPI プロセス間の ping-pong 通信の通信遅延を測定する．測定は，dpdk btl を用いた場合と tcp btl を用いた場合の 2 通りで行い，両者の結果を比較した．評価結果を図 11 のグラフに示した．

表 1 実験ノード

CPU	Intel®2 Xeon® CPU E5-2690 2.9GHz
メモリ	DDR3 SDRAM 64 GB
NIC	Intel® Corporation 82599 10 Gigabit TN Network Connection
OS	Linux® (Debian 8.4)

TCP EAGER は tcp btl の eager 通信を用いたときの，*TCP RNDV* は tcp btl の rendezvous 通信を用いたときの測定結果を示している．*DPDK EAGER* は dpdk btl の eager 通信を用いたときの，*DPDK RNDV* は dpdk btl の rendezvous 通信を用いたときの測定結果を示している．グラフの横軸は，送受信するデータのサイズを, 縦軸は通信遅延を示している．

グラフに示す通り, 送受信するデータサイズが 13 KBytes 以下の場合，dpdk btl の eager 通信が最も通信遅延が小さく

2 Intel, Xeon は, アメリカ合衆国およびその他の国における Intel Corporation またはその子会社の商標または商標です。

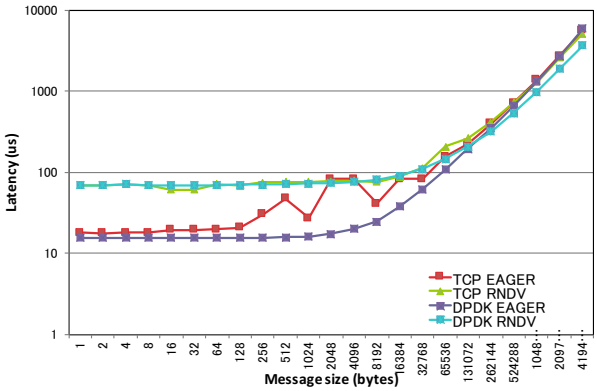

図 11 osu_latency の実行結果

なっている．一方，送受信するデータサイズが 26 KBytes 以上の場合，dpdk btl の rendezvous 通信が最も通信遅延が小さい．dpdk btl と tcp btl を比較すると，いずれのデータサイズでも dpdk btl の通信遅延が小さいという結果になった．最も良いケース（データサイズが 2 KBytes のとき）では，通信遅延を約 77%改善することができた．OS カーネルを経由せずにデータ転送可能な分，dpdk btl の方が tcp btl よりも高速になっていると考えられる．

6. 関連研究

6.1 usNIC

DPDK と同様，OS カーネルを経由せずに Ethernet 通信を行うためのフレームワークとして Cisco®3社の usNIC が挙げられる．DPDK は様々なベンダの NIC に対応しているが，usNIC は Cisco®社の特定の NIC にしか対応していないという違いがある．

Open MPI には，usNIC を用いたデータ転送をサポートするモジュールとして「usinic btl」が BTL レイヤに実装されている．ポリング専用のプロセスを持たない等，dpdk btl とは異なる設計となっている．より詳細な比較を行うためには，usinic btl の実装を調査する必要がある．

6.2 mTCP

DPDK を用いた TCP/IP の実装として mTCP が提案されている．mTCP は OS カーネルに実装されている TCP/IP ソケットの機能をマルチコア環境向けに最適化し，ユーザレベルで再実装したものである．mTCP の提供するソケット API を呼び出すように tcp btl を変更することで，DPDK を用いたデータ転送を MPI 通信に適用することができると考えられる．

dpdk btl は，TCP/IP 層が担うアドレッシングや伝送制御の機能を必要最低限に簡易化して Open MPI の BTL レイヤに組み込んでいる．それと比べ，mTCP のソケット API を tcp btl から呼び出す方式では，mTCP 内で実行される高度な TCP/IP 層の処理が MPI 通信の大きなオーバヘッドになると予測される．

3 「Cisco」はシスコおよび/または関連会社の登録商標です。

7. まとめ

本研究は，DPDK によるデータ転送を行うモジュールを Open MPI の BTL レイヤに実装し，DPDK の高速なデータ転送を MPI 通信に適用した．OSU Micro-Benchmarks を用いて評価したところ，TCP/IP ソケットによるデータ転送を用いた場合と比べ，通信性能を最大で 77%向上し、目標を達成することが出来た．

今後の課題は以下である。本研究は，osu_latency による通信性能のみの評価をおこなった．通信だけではなく計算処理も実行するミニアプリケーション等を用い，より詳細な評価を行うこと，その評価結果をもとに dpdk btl の実装を最適化することなどが挙げられる。

参考文献

[1] MPI Forum, http://mpi-forum.org.
[2] Open MPI: Open Source High Performance Computing, https://www.open-mpi.org .
[3] MPICH: High-Performance Portable MPI, https://www.mpich.org.
[4] DPDK, http://dpdk.org.
[5] OSU Micro-benchmarks, http://mvapich.cse.ohio-state.edu/benchmarks/
[6] EunYoung Jeong and Shinae Wood and Muhammad Jamshed and Haewon Jeong and Sunghwan Ihm and Dongsu Han and KyoungSoo Park, mTCP: a Highly Scalable User-level TCP Stack for Multicore Systems, NSDI'14