Vol.2015-HPC-150 No.41
2015/8/6

メニーコア環境向けタスクモデルPVASを用いたMPIのEager通信高速化

島田 明男[1,a)]　須藤 敦之[1,b)]　堀 敦史[2,c)]　石川 裕[2,d)]

概要：近年メニーコアプロッサの HPC システムへの活用が進んでいる．メニーコアプロセッサを採用するシステムでは，多数の MPI プロセスが 1 ノード上で動作することになる．よって，ノード内通信の発生回数が増加し，より高速なノード内通信が求められる．高速なノード内通信を実現するため，様々なシングルコピー通信の方式が提案されているが，その適用対象は Rendezvous 通信のみにとどまっている．シングルコピー通信を実行するには，送信バッファと受信バッファの双方の準備が完了していなければならないが，Eager 通信時は受信バッファの準備が完了しているとは限らない．本研究では，シングルコピー通信を Eager 通信にも適用する方式を提案し，メニーコア環境向けの新たなタスクモデルである PVAS のアドレス空間共有機能を用いて，これを実現する．本研究で提案する方式を Open MPI に実装してベンチマークで評価した結果，小さなサイズのメッセージの送受信を高速化し，4.9%の性能改善を実現することができた．

1. はじめに

近年電力効率の観点から，コア単体のパフォーマンスを向上させるよりもコア数を増加させることで処理性能を高める CPU アーキテクチャが一般的になっており，マルチコアプロセッサのコア数は増加する傾向にある．また，低性能だが電力効率が高いコアを多数集約することで高い並列処理性能を実現するメニーコアプロセッサの HPC システムへの活用も進んでいる．

Message Passing Interface (MPI) は並列計算処理の通信規格であり，並列アプリケーションを構築するために用いられる．MPI プログラムを実行するプロセス (MPI プロセス) は，MPI の提供する通信ライブラリを用いて並列計算に必要なデータを互いに送受信する．MPI プログラムを HPC システム上で実行する場合，高い処理性能を実現するために，利用可能なコア数に応じて起動する MPI プロセスの数を増加させるのが一般的である．メニーコアプロセッサを採用するシステムではノードあたりのコア数が増加するため，多数の MPI プロセスが 1 ノード内で動作することになる．よって，従来よりもノード内通信（同一ノード内で動作するプロセス間の通信）が発生する回数が増加し，より高速なノード内通信が求められる．文献 [13] では，メニーコアプロセッサを搭載するノードで構成されるクラスタ上で MPI プログラムを実行した際に，ノード内通信の高速化がプログラムの実行性能向上に寄与することが報告されている．

MPI のノード内通信を実現する代表的な方式として，共有メモリを経由したダブルコピー通信が挙げられる．この方式では，送信プロセスと受信プロセスの双方がアクセス可能な共有メモリ領域を作成してアドレス空間越しにメッセージの送受信を行う．送信バッファから共有メモリ，共有メモリから受信バッファへのメッセージのコピーが通信のたびに要求されるため，通信遅延が大きくなる．この問題を解決するため，マルチコアプロセッサの登場以来，様々な方式のシングルコピー通信が提案されている [3][5][14][1]．シングルコピー通信では，送信バッファから受信バッファに直接メッセージをコピーするため，共有メモリを経由するダブルコピー通信よりも通信遅延が小さくなる．

主要な MPI ライブラリは，メッセージの送受信において Rendezvous と Eager と呼ぶ 2 つの通信プロトコルをサポートしている．Rendezvous 通信は，通信を行う双方の MPI プロセスの準備が完了してからメッセージの送受信を行う同期通信である．それに対し Eager 通信は，送信プロセスが受信プロセスの状態に関わらずメッセージの送信処理を開始，終了することができる非同期通信である．一般

[1] 日立製作所 研究開発グループ
[2] 理化学研究所 計算科学研究機構
[a)] akio.shimada.ht@hitachi.com
[b)] atsuhi.sutoh.ff@hitachi.com
[c)] ahori@riken.jp
[d)] yutaka.ishikawa@riken.jp

的に，Rendezvous 通信はサイズが大きいメッセージの送受信に，Eager 通信はサイズが小さなメッセージの送受信に向いているといわれる．

既に述べた通り，シングルコピー通信を用いると，ノード内通信を高速化することができる．しかし先行研究では，シングルコピー通信の適用先が Rendezvous 通信のみにとどまっている．シングルコピー通信は，送信バッファから受信バッファに直接メッセージをコピーするため，双方のバッファの準備が完了していなければ実行することができない．Rendezvous 通信は同期通信であるため，通信開始時に双方のバッファの準備が完了していることが保証されている．それに対し Eager 通信は非同期通信であるため，送信プロセスが送信処理を開始する際に，受信バッファの準備が完了しているとは限らず，これがシングルコピー通信の適用を妨げていた．

そこで本研究では，シングルコピー通信を Eager 通信に適用し，Eager 通信を高速化する方式を提案する．本方式の Eager 通信では，送信プロセスが送信処理を開始時に受信プロセスの受信キューを走査し，これから行う送信処理に対応する受信リクエストを検索する．対応する受信リクエストがキューに存在する場合，受信バッファの準備が既に完了していることが保証される．よって，シングルコピー通信を実行し，送信バッファから受信リクエスト中に記録されている受信バッファのアドレスにメッセージをコピーする．対応する受信リクエストがキューに存在しなかった場合，受信バッファの準備が完了していないと見なし，従来と同様にダブルコピー通信によるメッセージの送受信を実行する．本方式を用いることにより，送信処理開始時に対応する受信リクエストが既に発行されているケースでは，Eager 通信の通信遅延が従来よりも小さくなる．本方式の実装には，メニーコア環境向けの新たなタスクモデルである Partitioned Virtual Address Space (PVAS) [15][11][10] のアドレス空間共有機能を用いた．PVAS タスクモデルを用いると，送信プロセスと受信プロセスを同一アドレス空間で動作させることが可能になる．よって，送信プロセスが受信プロセスの受信キューと受信バッファにアクセスすることができるようになる．本方式の Eager 通信を Open MPI [7] に実装し，ベンチマークソフトによって評価したところ，小さなサイズのメッセージの送受信を高速化し，約 4.9%の性能改善を実現することができた．

本論文の構成は以下の通りである．次章にて，ダブルコピー通信を用いる既存の MPI ノード内通信の実装とシングルコピー通信について述べる．第 3 章にて，提案方式の概要と PVAS タスクモデルを用いた実装について述べる．第 4 章にてベンチマークソフトを用いた評価について述べ，第 5 章にて関連研究について述べる．最後に，本研究の結論を述べる．

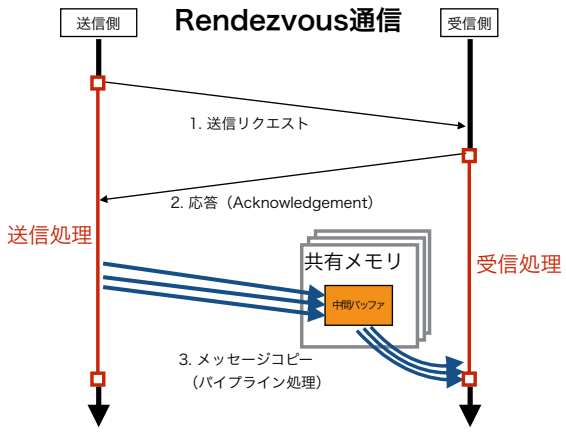


図 1 共有メモリを用いた Rendezvous 通信

2. 背景

近年のオペレーティングシステム（OS）は，仮想メモリによるメモリ保護機能を提供する．物理メモリと仮想アドレスのマッピングを管理するページテーブルがプロセスごとに用意され，プロセスはそれぞれ個別のアドレス空間で動作する．よって，プロセスは他のプロセスのメモリに直接アクセスすることはできない．

アドレス空間越しにノード内通信を行うために，主要な MPI ライブラリは，共有メモリによるダブルコピー通信を用いている．この方式では，通信を行う双方のプロセスがアクセス可能な共有メモリ領域を経由して，メッセージの送受信を行う．本章では，共有メモリを用いた MPI ノード内通信について，Redezvous 通信と Eager 通信双方の実装を説明する．また，高速なノード内通信を可能にするシングルコピー通信について述べる．

2.1 MPI ノード内通信の実装

本節では，共有メモリを用いた MPI ノード内通信の実装について述べる．Open MPI の実装をもとに説明するが，他の MPI ライブラリにおいても，ここで述べる実装と大きな違いはない．

2.1.1 Rendezvous 通信

図 1 は，共有メモリを用いた Rendezvous 通信の概要を示している．Rendezvous 通信では，通信を行う双方のプロセスが同期して通信を行う．まず，送信プロセスが送信リクエストを受信プロセスに送信する．受信プロセスは受信準備が完了したら，送信リクエストに対する応答を送信プロセスに送信する．受信プロセスからの応答を受信した送信プロセスは，送信バッファから共有メモリ領域上の中間バッファにメッセージをコピーすることで，メッセージの送信処理を行う．一方受信プロセスは，共有メモリ上の中間バッファから受信バッファにメッセージをコピーすることで，メッセージを受信する．

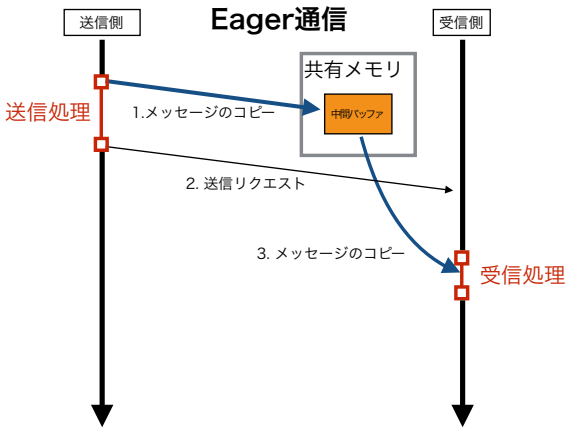


図 2 共有メモリを用いた Eager 通信

共有メモリを経由したメッセージのコピーは，パイプライン的に実行される．送信プロセスは，メッセージを固定サイズのチャンクに分割し，順に中間バッファにコピーしていく．各チャンクのコピーが完了したら，コピーを実行した中間バッファの完了フラグをセットする．受信プロセスは，完了フラグがセットされている中間バッファから，順にチャンクを受信バッファにコピーしていく．こうすることで送信側のメモリコピーと受信側のメモリコピーをオーバラップさせることが可能になり，通信遅延が小さくなる．

2.1.2 Eager 通信

図 2 は，共有メモリを用いた Eager 通信の概要を示している．Eager 通信では，通信を行う双方のプロセスが同期せずに通信を行う．Eager 通信では，まず送信プロセスがメッセージを送信バッファから共有メモリ上の中間バッファにコピーする．そして，送信リクエストを受信プロセスに送信し，送信処理を完了する．受信プロセスは，受信準備が完了したら，メッセージを共有メモリ上の中間バッファから受信バッファにコピーし，受信処理を完了する．このように，共有メモリ上の中間バッファにメッセージを一時的に保存することで，受信プロセスの状態に関わらず，送信プロセスは送信処理を開始，終了することができる．

Eager 通信では，送受信プロセスが同期を行う必要がないため，その分 Rendezvous 通信よりも通信遅延が小さくなる．ただし，送信するメッセージサイズが大きい場合は，パイプライン的にメッセージをコピーする効果により，Rendezvous 通信の方が通信遅延が小さくなる．Eager 通信と Rendezvous 通信を切り替えるメッセージサイズを eager threshold と呼ぶ．主要な MPI ライブラリでは，MPI プログラムの実行時に，ユーザが eager threshold を変更できるようになっている．

2.2 シングルコピー通信

共有メモリによるダブルコピー通信では，2 回以上のメモリコピーが要求されるためノード内通信の通信遅延が大きくなる．そこで，1 度のメモリコピーでノード内通信を実行可能なシングルコピー通信の方式が提案されている．

2.2.1 OS 支援によるシングルコピー通信

KNEM [3] と LiMIC [5] は，OS カーネルの支援によるシングルコピー通信をサポートするためのカーネルモジュールである．OS のカーネルの支援によるシングルコピー通信では，メッセージの送受信を OS カーネルに委譲する．OS カーネルは，送信プロセスと受信プロセス双方が使用しているメモリにアクセスする権限をもっている．よって，OS カーネルは，送信プロセスの送信バッファから受信プロセスの受信バッファにメッセージを直接コピーすることが可能である．1 度のメモリコピーによって通信を完了することができるので，その分共有メモリを用いたダブルコピー通信よりも通信遅延が小さくなる．

2.2.2 メモリマッピングによるシングルコピー通信

XPMEM [14] と SMARTMAP [1] は，プロセスに他のプロセスのメモリを自身のアドレス空間にマップする機能を提供する．受信プロセスが，送信プロセスの送信バッファのメモリを自身のアドレス空間にマップし，マップした領域からメッセージを受信バッファにコピーすることで，1 度のメモリコピーでノード内通信が可能になる．XPMEM は Linux®[*1] のカーネルモジュールとして提供されている．SMARTMAP は，米国サンディア国立研究所の開発した軽量 OS である，Catamout [9] と Kitten [8] に実装されている．

3. 提案

前章で述べた通り，シングルコピー通信を用いるとメモリコピーの回数を減らし，MPI のノード内通信を高速化することが可能になる．しかし，シングルコピー通信は，送信バッファと受信バッファ双方の準備が完了していなければ，実行することができない．よって，通信開始時に，受信バッファの準備が完了しているとは限らない Eager 通信には適用されていなかった．そこで本研究では，シングルコピー通信を Eager 通信にも適用し，同一ノード内で発生する MPI の Eager 通信を高速化する手法を提案する．

本研究の提案する Eager 通信では，まず送信プロセスが受信プロセスの受信キューを走査し，これから実行する送信処理に対応する受信リクエストを検索する．対応する受信リクエストがキューに存在する場合，すでに受信バッファの準備が完了しているので，送信バッファから受信バッファに直接メッセージをコピーするシングルコピー通信を実行する．対応する受信リクエストが存在しなかった場合は，受信バッファの準備が完了していないとみなし，共有メモリを用いたダブルコピー通信を実行する．こうす

[*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．

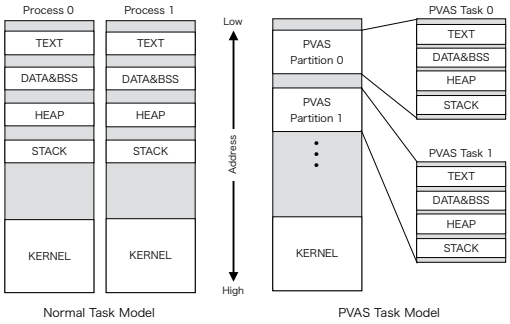


図 3 PVAS タスクモデルのアドレス空間

ることで，送信処理を開始する前に対応する受信処理が既に発行されているケースでは通信遅延が小さくなる．

本提案方式を主要な MPI ライブラリの一つである Open MPI [7] に実装した．実装には，メニーコア環境向けのタスクモデルとして本研究の著者らが提案している PVAS タスクモデル [15][11][10] を用いた．PVAS タスクモデルは，ノード内のプロセスを同一アドレス空間で動作させることを可能にする．送受信プロセスを PVAS タスクモデルによって同一アドレス空間で動作させ，送信プロセスが受信プロセスの受信キューと受信バッファにアクセスすることを可能にし，本研究の提案方式を実装した．

本章では，まず実装に用いた PVAS タスクモデルについて説明する．そして，PVAS タスクモデルを用いて本研究の提案方式をどのように Open MPI に実装したかについて述べる．

3.1 PVAS タスクモデル

図 3 は，既存のタスクモデルと PVAS タスクモデルのアドレス空間レイアウトを示している．既存のタスクモデルでは，プロセスが個別のアドレス空間で動作する．TEXT/BSS/HEAP/STACK といったメモリセグメントは，各プロセスの個別のアドレス空間にマッピングされる．それに対し，PVAS タスクモデルでは，複数のプロセスが同一アドレス空間で動作することを可能にする．PVAS タスクモデルでは，1 つの仮想アドレス空間を PVAS パーティションと呼ぶセグメントに分割し，同一アドレス空間で動作させる各プロセスに割り当てる．各プロセスは，自身の PVAS パーティションをプライベートなメモリ領域として利用することができる．PVAS タスクモデル上で実行されるプロセスを，通常のプロセスと区別するため，PVAS タスクと呼ぶ．PVAS タスクは各自，個別のメモリセグメント (TEXT/BSS/HEAP/STACK) を自身の PVAS パーティション内に保持する．通常のプロセス同様，自身の BSS/HEAP/STACK セグメントをプライベートなデータ領域として利用することができる．TEXT セグメントのメモリは同一バイナリを実行する PVAS タスク同士で共有される．同一アドレス空間で動作する PVAS タスクは，同一ページテーブルを使用して，メモリのマッピング情報を管理する．各 PVAS タスクの独立性を確保するため，PVAS タスクは，mmap() や munmap() といった仮想メモリ操作を，自身の PVAS パーティションを対象に実行することはできるが，他の PVAS タスクに割り当てられた PVAS パーティションを対象にして実行することは出来ない仕様となっている．

PVAS タスクと通常のプロセスの違いは，他のプロセスとアドレス空間を共有しているか否かのみである．PVAS タスクは，通常のプロセスと同様，独自のメモリセグメント (TEXT/BSS/HEAP/STACK)，ファイルディスクリプタ，プロセス ID，シグナルハンドラ等を持つ．よって，ソースコードへの改変無しに，既存の MPI プログラムを PVAS タスクとして実行することができる．PVAS タスクとして同一アドレス空間で動作する MPI プロセス同士は，load/store 命令によって，互いのメモリにアクセスすることができる．よって，送信プロセスの送信バッファから受信プロセスの受信バッファにメッセージを直接コピーすることが可能になる．また，通信キューや通信リクエスト，コミュニケータといった MPI ライブラリの管理データに，通信を行うプロセス同士が相互にアクセスすることが可能になる．

PVAS タスクモデルを, Intel® Xeon Phi™[*2] 用の Linux カーネルに実装した。Xeon Phi 用の Linux カーネルは，Intel Manycore Platform Software Stack [4] の一部として配布されている．実装の詳細については，文献 [11] に詳しい．

3.2 実装

3.2.1 SM BTL

Open MPI の通信ライブラリはモジュール化されており，通信アルゴリズやデータ送受信の方式を柔軟に追加，変更することができる．Byte Transfer Layer(BTL) は，メッセージデータの送受信を実行するレイヤで，様々な通信方式をサポートするコンポーネントによって構成されている．共有メモリによるダブルコピー通信は SM BTL コンポーネントによってサポートされている．

図 4 は，SM BTL における Eager 通信の実装を示している．この実装の送信プロセス側では，まず送信するメッセージを共有メモリ上の中間バッファにコピーする．コピーが完了したら，共有メモリ上の送信キューに，送信リクストを追加する．追加する送信リクエストには，メッセージをコピーした中間バッファのアドレス（実際には共有メモリ領域上でのオフセット値）を記録しておく．送信キューは，送信プロセスと受信プロセスの双方がアクセス

[*2] Intel，Xeon Phi は，アメリカ合衆国および/またはその他の国における Intel Corporation の登録商標または商標です．

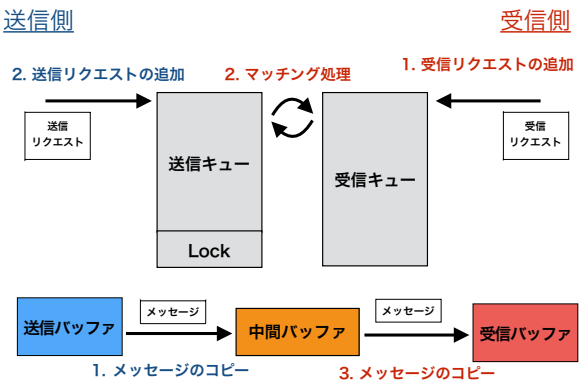


図 4 ダブルコピー通信を行う Eager 通信の実装 (SM BTL)

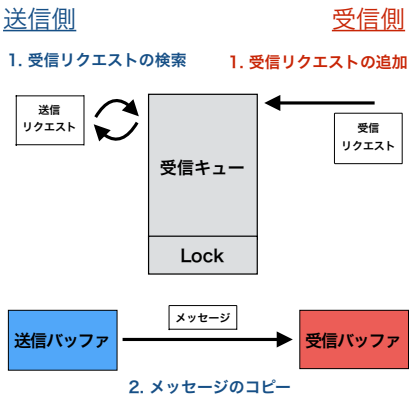


図 5 シングルコピー通信を行う Eager 通信の実装 (PVAS BTL)

するため，キューへの追加，削除時には，ロックによる排他制御を行う必要がある．キューへの追加後，送信プロセスは送信処理を完了する．受信側では，まず受信リクエストを自身のローカルメモリ上に存在する受信キューに追加する．そして，通信のプログレス処理の中で送信キューと受信キューを走査し，通信リクエストのマッチングを行う．マッチングするリクエストが存在すれば，送信リクエスト中に記録されている共有メモリ上の中間バッファから受信リクエスト中に記録されている受信バッファのアドレスにメッセージをコピーし，受信処理を完了する．このように，メッセージを送受信するために必ず 2 度のメモリコピーが通信のたびに発生する．

3.2.2 PVAS BTL

SM BTL をもとにして，本研究の提案する Eager 通信をサポートする PVAS BTL コンポーネントを実装した．図 5 は，PVAS BTL における Eager 通信の実装を示している．この方式では，まず送信プロセスが受信プロセスの受信キューを走査し，これから実行しようとしている送信処理に対応する受信リクエストを検索する．対応する受信リクエストがキューに存在しなかった場合，SM BTL と同様に，図 4 が示す通常の Eager 通信を実行する．もし対応する受信リクエストが存在する場合は，受信リクエストに記録されている受信バッファのアドレスを参照し，送信バッファからメッセージを受信バッファにコピーする．そして，受信リクエスト中に通信が完了したことを示すフラグをセットして送信処理を完了する．受信側は，通信のプログレス処理の中で受信キューを走査して通信完了フラグがセットされている受信リクエストを検索し，当該受信処理を完了する．送信処理を開始する前に，対応する受信リクエストが発行されているケースでは，シングルコピー通信が可能になり，1 度のメモリコピーでノード内通信を実行することができる．

PVAS BTL の実装を可能にするために，MPI プロセスを PVAS タスクとして起動するように Open MPI のプロセスマネージャを改変した．PVAS タスクとしてノード内の MPI プロセスを起動することで，送信プロセスが受信プロセスの受信キューと受信バッファにアクセスすることが可能になる．

PVAS BTL の実装では，受信キューに送信プロセスと受信プロセスの双方がアクセスするため，排他制御を行うためのロックを受信キューに追加した．よって，排他制御のオーバヘッドにより，通信遅延が増加する懸念がある．そこで，送信プロセスが受信キューのロックを取得できなかった場合，ロックの再取得は行わず，図 4 で示す通常の Eager 通信を実行するようにし，排他制御による影響を最小限にとどめる実装とした．

PVAS BTL は，Eager 通信だけではなく，Rendezvous 通信においてもシングルコピー通信を用いる．シングルコピー通信を用いる Rendezvous 通信の実装については，文献 [15] に詳しい．

表 1 評価環境

プロセッサ	Intel Xeon Phi coprocessor 5110P 物理コア数 60（4 HT）, 1.053 GHz, 32 KB L1 キャッシュ, 512 KB L2 キャッシュ, 8 GB メインメモリ
OS カーネル	Xeoh Phi 用 Linux カーネル (Intel MPSS 3.2.1)
MPI ランタイム	Open MPI 1.8

4. 評価

本研究で提案する Eager 通信の評価を行った．評価環境を表 1 に示す．表に示すように，評価には Intel 社のメニーコアプロセッサである Xeon Phi を用いた．評価は複数ノードではなく，単一ノードで行った．

4.1 通信レイテンシ

提案する Eager 通信のレイテンシを OSU Micro-Benchmarks [12] の osu_latency ベンチマークにて評価した．osu_latency は Pingpong 通信を実行し，Peer-to-Peer

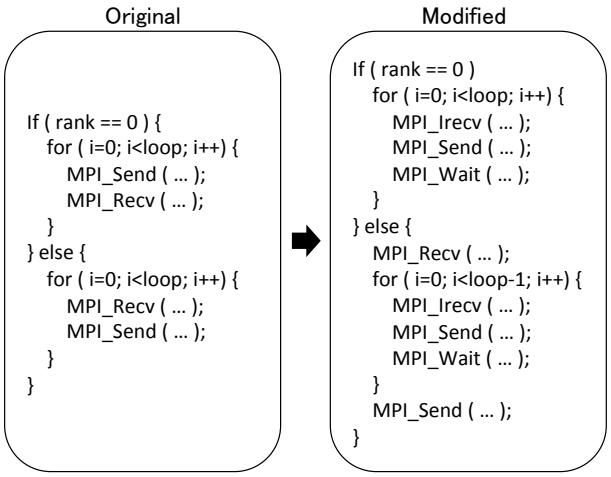


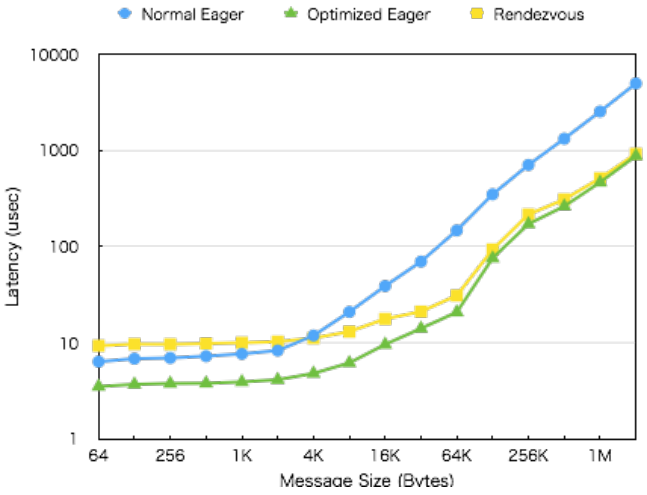
図 6 送信を行う前に次回の通信の受信を発行する

通信の通信遅延を測定するベンチマークである．本研究で提案する Eager 通信は，送信処理開始前に対応する受信リクエストが既に発行されているケースで通信遅延が小さくなる．そこで図 6 に示す通り，通信を行う各プロセスが送信を行う前に，`MPI_Irecv()` によって次回の通信の受信を事前に発行するように，osu_latency のソースコードを改変した．こうすることで，常に高速化された Eager 通信が実行され，提案手法の有用性を評価することができる．

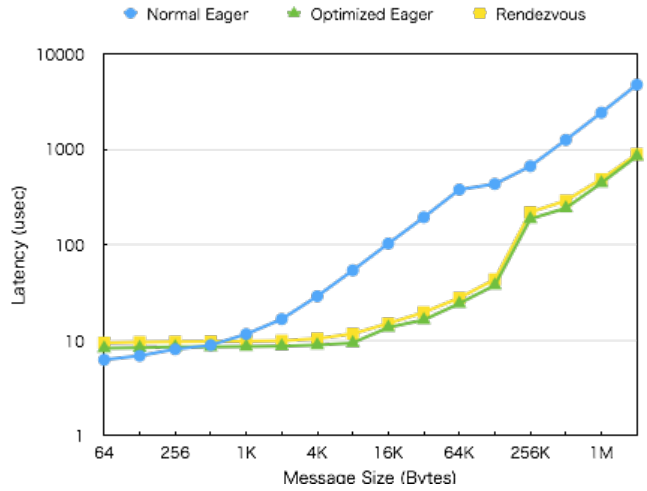
PVAS BTL において，本研究で高速化した Eager 通信を用いる場合と既存の Eager 通信を用いる場合，Rendezvous 通信を用いる場合のそれぞれで，通信レイテンシを測定した．Eager 通信の測定の際は，メッセージサイズが大きくなっても常に Eager 通信が実行されるよう，eager threshold を十分大きな値に設定した．

本研究の提案する Eager 通信では，送信プロセスが受信プロセスのメモリ上のキューにアクセスする．よって，送信プロセスと受信プロセスが異なるコア上で動作する場合は，CPU のキャッシュミスが発生する確率が高くなり，性能が低下する可能性がある．そこで，通信を行う双方のプロセスが同一コア上で動作する場合とそうでない場合の 2 通りで測定を行った．測定結果を図 7 に示した．*Normal Eager* は既存の Eager 通信の性能を，*Optimized Eager* は本研究で提案した Eager 通信の性能を示す．*Rendezvous* は Rendezvous 通信の性能を示している．

通信プロセスが同一コア上で動作する場合、本研究で提案する Eager 通信の方が従来の Eager 通信よりも常に高速となる．通信プロセスが異なるコア上で動作する場合はキャッシュを共有しないため，キュー探索のオーバヘッド等が大きくなり，メッセージサイズが 256 Bytes 以下のときは提案する Eager 通信の方がレイテンシが大きくなった．メッセージサイズが 512 Bytes 以上のときは，シングルコピー通信の効果により，本研究で提案する Eager 通信の方が高速となった．この結果をもとに，通信プロセスが



(a) 通信プロセスが同一コアで動作



(b) 通信プロセスが異なるコアで動作

図 7 通信遅延の測定結果（osu_latency）

異なるコア上で動作し，かつメッセージサイズが 256 Bytes 以下の場合は，常に通常の Eager 通信を実行し，メッセージサイズがそれより大きい場合は提案した Eager 通信を実行するように PVAS BTL を改良した．

Rendezvous 通信と本研究で提案する Eager 通信は，どちらもシングルコピー通信を用いるので，ある程度メッセージサイズが大きくなると，ダブルコピー通信を行う通常の Eager 通信よりも高速となる．Rendezvous 通信と本研究で提案する Eager 通信を比べると，同期のためのオーバヘッドが無い分，本研究で提案する Eager 通信の方が高速となる．しかし，メッセージサイズが大きくなるほど，通信遅延に占めるメモリコピーのオーバヘッドが同期のためのオーバヘッドよりも相対的に大きくなり，その効果は小さくなる．

4.2 LULESH

次に，米国 Lawrence Livermore National Laboratory によって開発された流体力学の計算コードである LULESH [6]

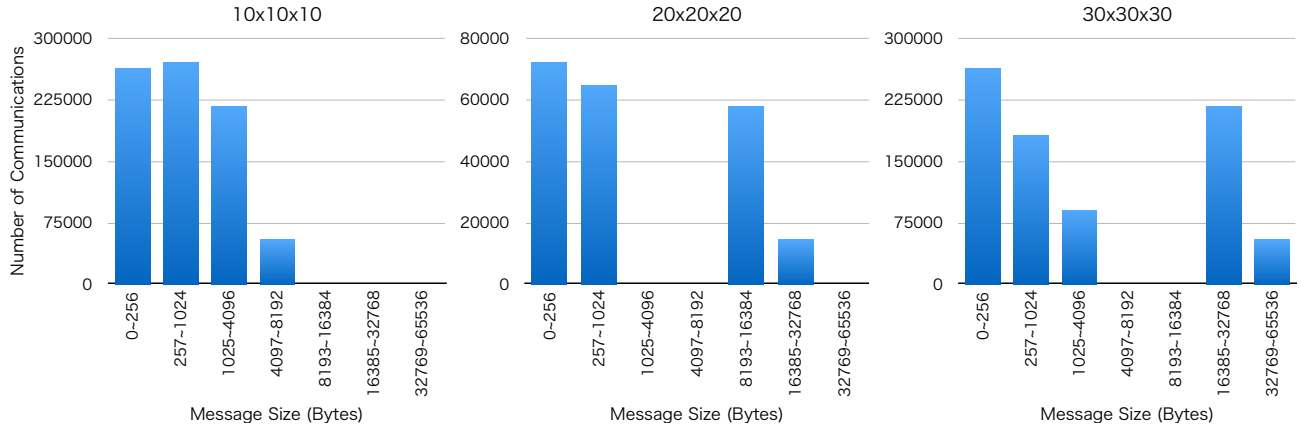


図 8 メッセージサイズごとの通信回数

を用いて性能測定を行った．このベンチマークでは，通信性能だけでなく，計算処理を含めた総合的な実行性能を測定することができる．LULESH は，通信において，送信処理を実行する前にそれに対応する受信リクエストが既に発行されているものの割合が他のベンチマークと比べて高く，本研究の提案する Eager 通信を評価するのに適していると判断した．問題サイズや MPI プロセス数を変えて LULESH の通信パターンを解析したところ，どの場合でも，65%以上の通信において，送信処理の前に対応する受信リクエストが発行されていた．

測定に用いた Xeon Phi コプロセッサーは 60 の物理コアを持ち，各コア上で 4 つのハードウェアスレッドが動作する．よって，OS が認識する論理コア数は 240 となる．しかし，LULESH では，プロセス数は任意の数の 3 乗という制限があるため，測定時のプロセス数は 216（6 の 3 乗）とした．計算対象のメッシュサイズは，測定に用いた Xeon Phi のメモリサイズでも実行可能な 10×10×10，20×20×20，30×30×30 とした．

LULESH の実行結果を表 2 に示す．測定は, eager threshold を 1 KB から 64 KB の間で変化させて行った．PVAS BTL において，既存の Eager 通信を用いた場合のピーク性能と本研究で提案した Eager 通信を用いた場合のピーク性能を比較した時の性能改善値をパーセンテージで表中に示した．また，本研究で提案した Eager 通信を用いた場合において，ピーク性能となったときの eager threshold を併記した．いずれのメッシュサイズにおいても，本研究の提案する Eager 通信が，既存の Eager 通信よりも高い性能を示している．特に問題サイズが小さい 10×10×10 のメッシュサイズでは，4.9%性能が改善しており，他のメッシュサイズと比べて高い効果を示した．

PVAS BTL では，Rendezvous 通信にシングルコピー通信を適用し，メモリコピー回数の増加によるオーバヘッドを回避している．それに対し，本研究の提案する Eager 通信では，同期のためのオーバヘッドとメモリコピー回数の

表 2 LULESH の実行結果（NP=216）

問題サイズ	10×10×10	20×20×20	30×30×30
性能改善値	4.9% (e.t.=8 KB)	0.7% (e.t.=16 KB)	0.2% (e.t.=32 KB)

増加によるオーバヘッドを共に回避することができる．しかし，メッセージサイズが大きい場合，同期のためのオーバヘッドよりもメモリコピーによるオーバヘッドの方が相対的に大きくなるので，本研究の提案手法で高速化された Eager 通信と Rendezvous 通信の性能差は小さくなる．これは，前節の通信レイテンシの評価を見ても明らかである．本研究の提案する Eager 通信は，サイズが小さいメッセージの送受信において，その効果を発揮する．

サイズが大きいメッセージが多数送受信されるケースでは，本研究の提案する Eager 通信を用いるメリットは小さいといえる．データサイズが大きいほどメモリコピーのオーバヘッドは大きいので，サイズが小さいメッセージの送受信をシングルコピー通信を用いて高速化するよりも，サイズが大きいメッセージの送受信をシングルコピー通信を用いて高速化する方が，全体の実行性能に与える影響は大きい．よって，サイズが大きいメッセージが多数送受信されるケースでは，本研究の提案する Eager 通信を用いてサイズが小さいメッセージの送受信をシングルコピー通信によって高速化しても，全体の実行性能にほとんど影響を与えることはない．一方，大きなサイズのメッセージの送受信には，シングルコピー通信を適用した Rendezvous 通信を用いればよく，本研究の提案する Eager 通信を用いる意義は薄い．

以上から，本研究の提案する Eager 通信は，サイズが小さいメッセージの送受信が通信の大半を占めるケースで有効であると考えられる．これを確認するため，ベンチマーク実行中に行われる通信の回数をメッセージサイズごとに測定した．図 8 のグラフの横軸はメッセージのサイズ，縦軸は当該サイズのメッセージがベンチマーク実行中に送受信された回数を示している．本研究の提案する Eager 通信

が高い効果を示した 10×10×10 のメッシュサイズの場合，サイズが小さいメッセージ (8 KB 以下) の送受信が通信の大半を占めていることがわかる．それに対し，他のメッシュサイズでは，ベンチマーク実行中に，サイズが大きいメッセージ（16 KB 以上）が多数送受信されている．

5. 関連研究

Hybrid MPI [2][13] は，同一ノード内で動作する全 MPI プロセスがアクセス可能な共有メモリ領域をメモリプールとして作成する．各 MPI プロセスは，Hybrid MPI によって定義された `mmap()` と `sbrk()` を実行することで，使用するメモリをこのメモリプールから確保する．通信用バッファが，このメモリプールから割り当てられている場合，送信バッファから受信バッファに直接メッセージをコピーすることが可能になり，MPI のノード内通信を 1 度のメモリコピーで実行することが出来る．

Hybrid MPI は，メッセージサイズが小さい場合は，送信リクエストの中にメッセージ自体を含めて送信キューに追加する最適化を行う．受信プロセスは，送信リクエスト中のマッチング情報（コミュニケータ ID，タグ，MPI ランク）を参照した後に，送信リクエスト中のメッセージを自身の受信バッファにコピーする．マッチング情報とメッセージは連続したメモリ領域に保存されているため，送信バッファからメッセージをコピーするよりも，キャッシュミスが起きる確率が低くなり，サイズが小さいメッセージの送受信において通信遅延が小さくなる．Hybrid MPI の最適化と本研究で提案する Eager 通信は共存可能であり，両者を組み合わせることで，さらなるアプリケーションの実行性能改善が期待できる．

6. まとめ

本研究では MPI のノード内通信において，Eager 通信にもシングルコピー通信を適用する方式を提案した．本研究の提案手法では，送信プロセスが送信処理を開始する前に受信プロセスの受信キューを検索する．対応する受信リクエストが既にキューに追加されている場合は，受信バッファの準備が完了しているのでシングルコピー通信を実行する．もし，対応する受信リクエストがキューに存在しない場合は，受信バッファの準備が完了していないと見なし，共有メモリを用いたダブルコピー通信を従来通り実行する．こうすることで，送信処理を開始する前に対応する受信リクエストが発行されているケースでは Eager 通信を高速化することができる．

提案方式を PVAS タスクモデルのアドレス空間共有機能を用いることで Open MPI に実装し，評価した．評価の結果，本研究の提案する Eager 通信は，サイズが小さいメッセージが多数送受信されるケースで有効であることがわかった．ベンチマークによる測定では，最大で 4.9%の実行性能改善を実現することができた．

謝辞 本研究の一部は，科学技術振興機構 (JST) の戦略的創造研究推進事業「CREST」における研究領域「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ェア技術の創出」によるものである．また，本研究を推進するにあたり，多数の有用なコメントをいただいたテネシー大学 George Bosilca 教授に感謝の意を表する．

参考文献

[1] R. Brightwell, K. Pedretti, and T. Hudson. SMARTMAP: operating system support for efficient data sharing among processes on a multi-core processor. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, Piscataway, NJ, USA, 2008.

[2] A. Friedley, G. Bronevetsky, T. Hoefler, and A. Lumsdaine. Hybrid mpi: Efficient message passing for multi-core systems. In *Proceedings of the 2013 ACM/IEEE conference on Supercomputing*, NY, USA, 2013.

[3] B. Goglin and S. Moreaud. KNEM: a Generic and Scalable Kernel-Assisted Intra-node MPI Communication Framework. *Journal of Parallel and Distributed Computing (JPDC)*, 73(2):176–188, Feb. 2013.

[4] Intel Corporation. Intel manycore platform software stack, https://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss.

[5] H.-W. Jin, S. Sur, L. Chai, and D. K. Panda. Limic: Support for high-performance mpi intra-node communication on linux cluster. In *ICPP*, pages 184–191, 2005.

[6] Lawrence Livermore National Laboratory. Hydrodynamics challenge problem, 2012.

[7] Open MPI. Open Source High Performance Computing, http://www.open-mpi.org/.

[8] Sandia National Laboratory. Kitten Lightwaight Kernel, https://software.sandia.gov/trac/kitten.

[9] Sandia National Laboratory. Open Catamount, http://www.cs.sandia.gov/ rbbrigh/OpenCatamount/.

[10] A. Shimada, B. Gerofi, A. Hori, and Y. Ishikawa. PGAS Intra-node Communication towards Many-Core Architecture. In *6th Conference on Partitioned Global Address Space Programing Model*, Santa Barbara, California, USA, 2012.

[11] A. Shimada, B. Gerofi, A. Hori, and Y. Ishikawa. Proposing A New Task Model towards Many-Core Architecture. In *Proceedings of the ACM international workshop on manycore embedded systems 2013*, Tel-Aviv, Israel, June 2013. ACM.

[12] The Ohio State University. OSU Micro-Benchmarks, http://mvapich.cse.ohio-state.edu/benchmarks/.

[13] U. S. Wickramasinghe, G. Bronevetsky, A. Lumsdaine, and A. Friedley. Hybrid MPI: A Case Study on the Xeon Phi Platform. In *Proceedings of the 4th International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '14, pages 6:1–6:8, New York, NY, USA, 2014. ACM.

[14] M. Woodacre, D. Robb, D. Roe, and K. Feind. The SGI AltixTM 3000 Global Shared-Memory Architecture.

[15] 島田 明男, 堀 敦史, 石川 裕. 新しいタスクモデルによるメニーコア環境に適した MPI ノード内通信の実装. **情報処理学会論文誌コンピューティングシステム** *(ACS)*, 8(2):36–54, 2015-06-16.