Vol.2015-HPC-149 No.6
2015/6/26

# 密結合演算加速機構TCAアーキテクチャのIntel MICプロセッサへの適用

飯島 大貴[1,a)]　廣川 祐太[1]　塙 敏博[2]　朴 泰祐[1,3]

**概要**：GPU (Graphics Processing Unit) や Intel MIC (Many Integrated Core) アーキテクチャなどのアクセラレータを搭載したクラスタシステムでは，ノードをまたぐアクセラレータ間の通信が不可欠となる．しかし従来のクラスタシステムでは，ホストプロセッサとアクセラレータ間のデータ転送オーバーヘッドが大きいため，通信がボトルネックになってきた．そこで我々は，アクセラレータ間を直接結合することによって低レイテンシを実現する密結合並列演算加速機構 (Tightly Coupled Accelerators, TCA) アーキテクチャを提案している．TCA に基づく PEACH2 を開発し，ノード間を PCI Express (PCIe) で接続することで，プロトコル変換のオーバーヘッドを削減し，アクセラレータ間の直接通信を実現してきた．これまで，TCA は NVIDIA 社の GPU をターゲットとして開発されてきたが，GPU と同様に Intel MIC プロセッサも PCIe で接続されており，TCA を適用できる．本研究では，アクセラレータとして Intel MIC プロセッサを用いたクラスタシステムにおいて TCA アーキテクチャの適用を検討し，ホストプロセッサと MIC プロセッサ間の通信機能である SCIF と PEACH2 を組み合わせて Offload モデルにおける MIC 間通信を実現した．その結果，最小のレイテンシ 5.27$\mu$ 秒を達成し，通信を MIC プロセッサ向けに最適化した MPI 実装である MVAPICH2-MIC 2.0 や Intel MPI と比較して，より低いレイテンシで通信を行うことができた．

## 1. はじめに

近年，High-Performance Computing 分野におけるクラスタシステムには CPU だけでなく専用のアクセラレータ（演算加速装置）が搭載されることが多くなり，計算能力が大きく向上してきた．世界のスーパーコンピュータの性能のランキングである Top500 [1] でもアクセラレータを搭載した多くのスーパーコンピュータがランクインしている．アクセラレータは，代表的なものに NVIDIA 社製 GPU や Intel MIC (Many Integrated Core) アーキテクチャがある．

NVIDIA 社の GPU をアクセラレータとして利用する GPGPU 環境は，専用の言語である CUDA [2] を用いて記述する．一方，MIC プロセッサは，IA-32 をベースとしたメニーコアプロセッサであるため，汎用 CPU 向けのコードを少ない変更で利用することができる．

アクセラレータを用いるクラスタシステムにおける高性能計算では，ノードをまたぐアクセラレータ間の通信が不可欠となる．従来のクラスタシステムでは，異なる計算ノード上の GPU 間通信は，InfiniBand のようなネットワークを用いてホストプロセッサ間の通信を行うため，ホストプロセッサ-アクセラレータ間で明示的にデータ転送を行う必要があった．さらにホストプロセッサにアクセラレータを接続するための PCI Express (PCIe) と実際のネットワークとの間でプロトコルの変換が必要となり，オーバーヘッドが生じる．これらの問題を解決するために，我々はノードをまたぐアクセラレータ間を直接結合する，密結合並列演算加速機構 TCA (Tightly Coupled Accelerators) アーキテクチャを提案してきた [4]．また TCA アーキテクチャに基づき，FPGA を用いた通信機構として PEACH2 (PCI Express Adaptive Communication Hub version 2) を開発し，ノード間を PCIe で接続することで，アクセラレータ間の直接通信を実現してきた．PEACH2 は，PCIe パケットのままでデバイス間通信が可能なため，プロトコル変換のオーバーヘッドがなく，2.0$\mu$ 秒程度の低レイテンシ通信を実現している [4]．

一方，MIC プロセッサをアクセラレータとして搭載したクラスタシステムにおいても，ノード間の通信が不可欠であり，GPU と同様の問題が発生する．これまで，TCA 通信

[1] 筑波大学 大学院 システム情報工学研究科
Graduate School of System and Information Engineering, University of Tsukuba
[2] 東京大学 情報基盤センター
Information Technology Center, The University of Tokyo
[3] 筑波大学 計算科学研究センター
Center for Computational Sciences, University of Tsukuba
[a)] iijima@hpcs.cs.tsukuba.ac.jp

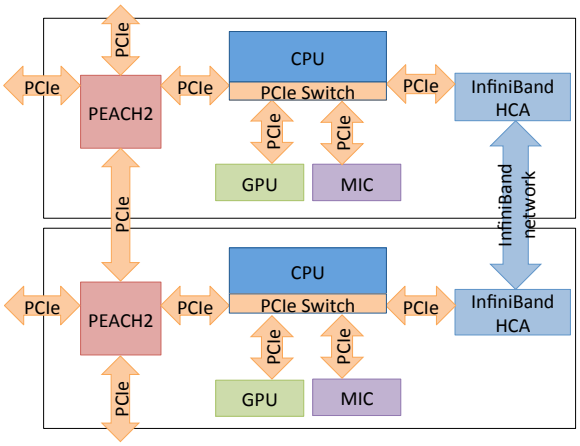


図 1　実験に用いた PEACH2 実験ノードの構成

機構は GPU をターゲットとして開発されてきたが，MIC プロセッサも PCIe でホストに接続されており，PEACH2 を用いて MIC プロセッサ間の直接通信を実現できると考えられる．そこで，本研究では，MIC プロセッサで用いられるホストとの通信インタフェース SCIF (Symmetric Communications Interface) と，PEACH2 とを連携させることで，MIC クラスタ環境に対して TCA 通信機構を適用する．

## 2. TCA (Tightly Coupled Accelerators)

我々はノードをまたぐアクセラレータ間の直接通信を実現する TCA (Tightly Coupled Accelerators) アーキテクチャを提案してきた [4]．従来のアクセラレータを搭載したクラスタシステムでは，ノードをまたぐアクセラレータ間の通信は，ホストプロセッサ上のメモリを経由して InfiniBand 等の汎用ネットワークを用いて間接的に行われていた．そのため，通信に伴うメモリコピーのオーバーヘッドが大きく，強スケーリングさせるのは困難であった．それに対し，TCA では，ノードをまたぐ演算加速装置間で直接通信が可能であるため，アクセラレータ間の通信レイテンシを最小限にし，強スケーリングを実現することができる．

現在，GPU や MIC プロセッサ等のアクセラレータは PCIe によって CPU と接続されており，PCIe をそのまま通信リンクに用いることで，ノードを超えたアクセラレータ間で直接通信が可能になる．そこで我々は PEACH2 (PCI Express Adaptive Communication Hub version 2) を実装し，TCA アーキテクチャの実証システムとして HA-PACS/TCA を開発してきた．本章では，本論文に関連する範囲で PEACH2 について概略を述べる．PEACH2 の詳細に関しては文献 [4] を参照されたい．

### 2.1 PEACH2 を用いたノードの構成

PEACH2 を用いたノードの接続の例を，**図 1** に示す．

PEACH2 は，FPGA により実装されており，PCIe Gen2 x8 のポートを 4 ポート持つ．うち 1 ポートが CPU ソケットに接続され，残り 3 ポートはノード間接続に用いられる．PEACH2 は PCIe のパケットを直接相手ノードにルーティングすることができるため，異なるノードに属する GPU 間の通信であってもすべて PCIe のパケットのまま行うことができる．InfiniBand を用いる場合には，InfiniBand と，InfiniBand のホストアダプタ (HCA) をホストプロセッサと接続する PCIe とでプロトコルが異なるため，プロトコルの変換がオーバーヘッドとなる．PEACH2 ではパケットヘッダを修正するだけで転送できるため，高速に通信することが可能となる．

PEACH2 は高性能な DMA コントローラ (DMAC) を内蔵しており，複数の DMA 要求をポインタでチェインしておくことで，1 度の操作で自動的に連続して転送が行われる Chaining DMA 機能を備えている．DMA 要求を指定するためのディスクリプタは，ホストプロセッサのメモリ上に作成する方法と，PEACH2 の内蔵メモリ上に作成する方法が用意されている．また，PEACH2 は現在，主に GPU をターゲットに開発されているが，ホストプロセッサのメモリを用いた通信により，ホストプロセッサ間の通信に用いることも可能である．

## 3. Intel MIC アーキテクチャ

Intel MIC アーキテクチャは，Intel IA-32 アーキテクチャに基づく汎用 CPU コアを多数搭載したメニーコアアーキテクチャであり，Intel Xeon Phi として高性能計算の分野向けにリリースされている．現在の MIC プロセッサは，PCIe でホストプロセッサに接続して使用する必要がある．MIC プロセッサは，512bit 長に対応した SIMD 命令をサポートしており，1 命令で複数のデータを処理することで性能が向上する．また，CPU コアが IA-32 アーキテクチャに基づいているため，汎用 CPU 向けに記述されたプログラムを少ない変更で，そのまま利用できる利点がある．

### 3.1 プログラミングモデル

MIC プロセッサのプログラミングモデルは，Native モデルと Offload モデル，Symmetric モデルが提供されている．

- Native モデル：C，C++，Fortran のプログラムに，MIC プロセッサ用のオプションを付けてコンパイルし直すことで，MIC プロセッサ上でそのまま実行することが出来る．
- Offload モデル：GPGPU のように，アクセラレータとしてプログラムの一部分を指定して MIC プロセッサ上で実行させる方式である．既存のプログラムに対して，指示文を挿入することで，offload するプログラム範囲，データ転送を指示する．
- Symmetric モデル：MPI を用いてホストプロセッサ

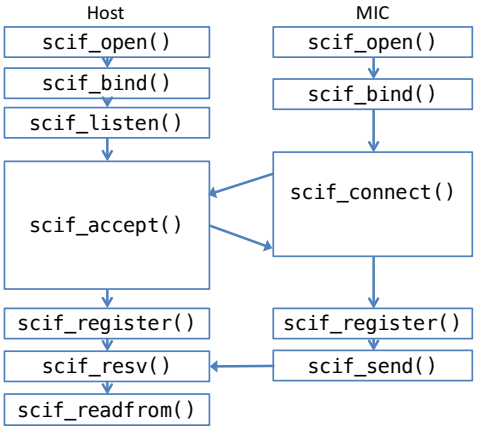


図 2　SCIF プログラムの例（SCIF Users Guide [6] より）

と MIC プロセッサにそれぞれ MPI プロセスを割り当て，プログラムを実行する方式である．

### 3.2 Intel MPSS (MIC Platform Software Stack)

Intel は，Intel MPSS (MIC Platform Software Stack) [5] と呼ばれるソフトウェアを MIC プロセッサシステム向けにリリースしている．MPSS は，MIC プロセッサ上で動作する組み込み Linux や，MIC プロセッサに最適化された GCC，ドライバ群から構成されている．MIC プロセッサでは，ホストから Linux を起動し，論理的にはホストプロセッサとは独立のノードとして動作する．また，ドライバを介してホストと論理的に接続されている．このホストと MIC プロセッサ間の通信機能は SCIF (Symmetric Communications Interface) [6] と呼ばれている．

## 4. Symmetric Communications Interface (SCIF)

SCIF は，ノードが MIC プロセッサ（Intel Xeon Phi）およびホストプロセッサ（Intel Xeon）で構成されるノード内のプロセッサ間通信のための仕組みを提供している．特に，ホストプロセッサと MIC プロセッサ間では，対称的な API を提供し，PCIe を介しての通信を抽象化している．API として，TCP/IP と類似のコネクション型の通信方式を用いている．

### 4.1 SCIF の API

SCIF において，プロセスがアクセスする実体をエンドポイントと呼ぶ．ここでは，主な API を紹介し，次節で SCIF を用いたデータ転送の手続きを示す．**表 1** に，主な API を示す．

### 4.2 SCIF によるホストプロセッサと MIC プロセッサ間の通信手順

**図 2** に SCIF によるプログラムの実行例を示す．

SCIF により，RMA 操作を行うためには，まずホストプロセッサと MIC プロセッサ間でコネクションを確立する必要がある．以下のような手続きとなる．

(1) ホストプロセッサ上のプロセスが，scif_open() を呼び出し，エンドポイントディスクリプタを取得する．
(2) ローカルポートに新しいエンドポイントをバインドする scif_bind() を呼び出す．
(3) scif_listen() を呼び出し，リスニングエンドポイントとしてエンドポイントを指定する．
(4) 接続の要求を受け入れるため，scif_accept() を呼び出す．
(5) MIC プロセッサ上のプロセスが，scif_open() し，エンドポイントディスクリプタを取得する．
(6) scif_bind() を呼び出し，ローカルポートに新しいエンドポイントをバインドする．
(7) scif_connect() を呼び出し，ホストプロセッサのポートへ接続の要求を行う．

接続が確立された後に，それぞれのメモリのマップの登録を scif_register() で行い，登録したアドレスのオフセットを scif_send(), scif_receive() で送受信を行う．送受信したオフセットの情報を元に，scif_readfrom(), scif_writeto() などにより RMA 操作を行うことができる．

## 5. PEACH2 による MIC プロセッサ間通信

現行の MIC プロセッサにおいて，3.1 節で述べたように，Offload, Native, Symmetric の 3 通りの使い方が可能である．MIC プロセッサ間直接通信を実現することを考えると，Native モデルと Symmetric モデルに相違はないため，ここでは Offload モデルとそれ以外の 2 通りについて考えることにする．

### 5.1 Offload モデル

Offload モデルでは，指示文を用いてホストプロセッサから MIC プロセッサに転送するデータを指定する．ホストプロセッサ〜MIC プロセッサ間の転送のみが指定可能であり，MIC プロセッサ間では直接通信を行うことが出来ない [*1]．従って，MIC プロセッサ上にあるデータを他の MIC プロセッサに送信する場合，一旦ホストに転送した上で MPI により送信し，さらにリモートホスト上で MIC プロセッサに転送する必要がある．PEACH2 を用いることにより，MIC プロセッサ間の直接データ転送が可能になる．一方で，PEACH2 による DMA は，ホストから指示するか，あるいは MIC プロセッサから SCIF を介して PEACH2 の DMA 操作を指示できるような機構が必要になる．

[*1] 同一ノード内の複数デバイス間についても直接転送はできない．

表 1　SCIF の API

| | |
|---|---|
| エンドポイントと接続 | |
| scif_open() | エンドポイントを作成する |
| scif_bind(epd, pn) | エンドポイントをポートに bind する |
| scif_listen(epd, backlog) | エンドポイントを接続待機状態にする |
| scif_connect(epd, dst) | エンドポイントに接続の要求を行う |
| scif_accept(epd, peer, newepd, flags) | 接続の要求を受け入れる |
| scif_close(epd) | エンドポイントを閉じる |
| メッセージング | |
| scif_send(epd, msg, len, flags) | メッセージを送信する |
| scif_recv(epd, msg, len, flags) | メッセージを受信する |
| メモリ登録 | |
| scif_register(epd, addr, len, offset, prot_flags, map_flags) | SCIF のアドレス空間にメモリの登録をする |
| scif_unregister(epd, offset, len) | メモリの登録を解除する |
| RMA | |
| scif_readfrom(epd, loffset, len, roffset, rma_flags) | リモートからローカルへ RMA 転送する |
| scif_writeto(epd, loffset, len, roffset, rma_flags) | ローカルからリモートへ RMA 転送する |
| ページ管理 | |
| scif_get_pages(epd, offset, len, pages) | SCIF のアドレス空間に登録されたメモリのページ情報を取得する |
| scif_put_pages(pages) | 取得したページ情報を削除する |

### 5.2　Native, Symmetric モデル

Native モデル，Symmetric モデルでは，通常のホストにおける MPI と同様に，MPI の API を記述することで通信が可能である．実際には，MIC プロセッサにおいては，SCIF インタフェースを介してホストに対して通信を依頼し，ホストが MPI 通信を行うことになる．PEACH2 デバイスを用いて MIC プロセッサに直接アクセスが可能であれば，低レイテンシでの通信が実現できる．この場合，SCIF を介した PEACH2 の DMA 操作を指示するための機構が必須になる．

### 5.3　実装

本研究における最終的な目的は MIC プロセッサ間直接通信の実現であるが，まず SCIF を用いて PEACH2 デバイスを MIC プロセッサ側で扱えるようにしなければならない．そこで本研究では，SCIF を用いてホストプロセッサと連携して PEACH2 による通信を実現することにした．また，TCA 通信機構が GPU 用に開発されていることから，MIC プロセッサへの適用は，GPU と同様のプログラミングモデルをとる，Offload モードを利用した．

本実装は，4 章で述べた SCIF の通信機能を用い，ホストプロセッサを経由する形で，以下の流れで行った．

(1) SCIF の RMA により MIC プロセッサからホストプロセッサに転送
(2) PEACH2 の CPU メモリ通信機能を用いてホストプロセッサ間の通信を行う
(3) SCIF の RMA によりホストプロセッサから MIC プロセッサに転送

また，概略図を図 **3** に示す．SCIF の RMA 転送は，ホストプロセッサと MIC プロセッサで，それぞれのメモリを SCIF のアドレス空間に登録し，事前にオフセット情報の送受信を行うことで，RMA 転送を行う．また，PEACH2 によるホストプロセッサ間通信では，PEACH2 の DMA の対象を CPU メモリに確保することで，通信を行う．また，SCIF アドレス空間にマップした領域と PEACH2 で DMA の対象となる領域の間は memcpy() によりメモリコピーを実行しデータを更新している．

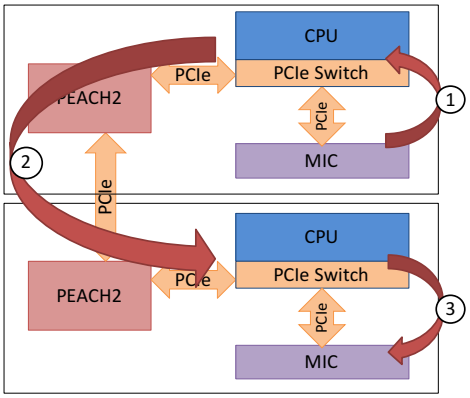


図 3　提案システムの実装方針

最終的に本研究の目的を達成するためには，ホストプロセッサのメモリを介さない通信を実現しなければならない．具体的には，SCIF を用いて MIC プロセッサ上のメモリを PEACH2 の DMAC で扱えるようにする必要がある．これには，表 1 で述べた，scif_get_pages() を用いることで MIC プロセッサ上のデータの PCIe アドレスを取得し，PEACH2 によって直接 DMA が可能になると考えられるが，今後の課題である．

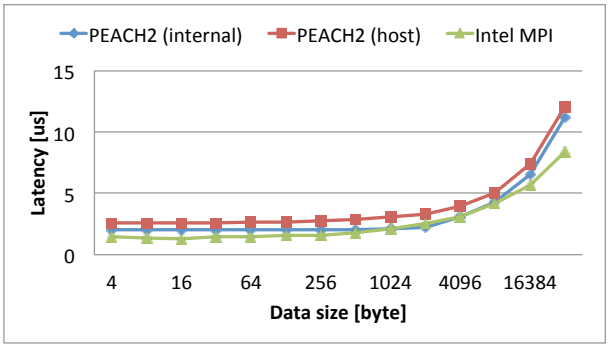


図 4　PEACH2 による通信のレイテンシ

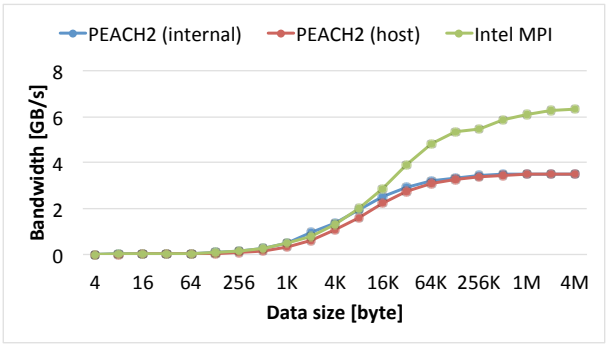


図 5　PEACH2 による通信のバンド幅

表 2　実験環境の構成

| ハードウェア | |
|---|---|
| CPU | Intel Xeon-E5 2670v2 2.5GHz |
| Memory | DDR3 1866MHz, 128GB |
| Motherboard | Supermicro X9DRG-QF |
| InfiniBand | Mellanox Connect-X3 FDR Dual-port |
| MIC | Intel MIC プロセッサ 5110P |
| ソフトウェア | |
| OS | CentOS 6.5 |
| Compiler | gcc 4.4.7, icc version 15.0.2 |
| MPI | Intel MPI Version 5.0 Update 2 |
| MPSS | MPSS version 3.5 |

## 6. SCIF による MIC プロセッサ間の PEACH2 通信の性能評価

ノードをまたぐ MIC プロセッサ間の SCIF を利用した PEACH2 による通信性能を調査するため，まず SCIF や PEACH2 による通信の予備評価を行い，提案システムの性能評価を行う．また，SCIF 性能の予備評価において，64 bytes 未満のデータサイズにおける性能の変化について評価する．以降の測定は，**表 2** で示す実験環境を用いて行った．

### 6.1 PEACH2 のホストプロセッサ間通信性能

5.3 節の実装方針を踏まえ，まず，PEACH2 によるホストプロセッサ間通信の性能を測定した．通信レイテンシおよび通信バンド幅を，**図 4** および**図 5** にそれぞれ示す．

測定は，ホストプロセッサメモリにディスクリプタを準備して PEACH2 に搭載された DMAC によって通信を行う方式 (host) と，PEACH2 に内蔵されたメモリにディスクリプタを準備して，DMAC によって通信を行う方式 (internal) で行う．host の場合は，通信開始時に PEACH2 からホストプロセッサメモリを読みに行くコストが発生するため，MIC プロセッサに適応させる際には，PEACH2 の内蔵メモリにディスクリプタを準備する方式を採用する．注意点として，ディスクリプタの設定は最小限にし，プログラムの実行開始時にすべて通信の定義を行っていることが望ましい．通信ごとに定義を行う場合，それぞれオーバーヘッドが発生し，性能低下につながるためである．

測定結果の図 4 より，最小レイテンシは，internal が 2.03$\mu$ 秒，host が 2.60$\mu$ 秒となっていることがわかる．また図 5 より，バンド幅は最大で internal が 3.51GB/秒，host が 3.50GB/秒を達成している．Intel MPI[7] は，最小レイテンシが 1.42$\mu$ 秒であり，バンド幅は 6.3GB/秒を達成しているので，CPU 間通信においては PEACH2 が劣ってしまう．

### 6.2 RMA 転送性能

SCIF を用いて MIC プロセッサのメモリをホストプロセッサに転送を行うにあたり，その転送性能の調査を行った．測定には表 2 の環境を用いた．

**図 6** にレイテンシ，**図 7** にバンド幅を示す．また，**表 3** に凡例について，通信の起動の違いと，通信に用いる関数についてまとめてある．

SCIF の公式リファレンス [6] によると，データサイズは 64 の倍数のデータサイズでないと，性能が出ないことが記述されているため，図 6，図 7 の性能調査では，64 bytes よりも大きいデータサイズの測定となっている．64byte 未満の場合については，次の節で述べる．64 bytes の時点で，順に 1.75$\mu$ 秒，1.70$\mu$ 秒，5.18$\mu$ 秒，5.22$\mu$ 秒となっており，低いレイテンシで転送を行なえていることがわかる．MIC プロセッサ側で通信を起動するとホストプロセッサ側で通信を起動した場合に比べレイテンシが大きい．これは，MIC プロセッサ側で通信起動処理を行う際，各コアの入出力性能がホストプロセッサに比べ劣ることなどが原因と考えられる．これ以降の測定で用いている実装では，ホストプロセッサで転送の制御を行い，この問題の影響を受けないようにしている．

また，大きな転送サイズにおけるバンド幅に着目すると，7GB/秒に達している．ホストプロセッサと MIC プロセッサの接続に用いている PCIe 接続は Gen2 x16 であり，片方向あたり 8GB/秒のバンド幅を持つが，ヘッダによるロス約 10%を考慮すると，十分なバンド幅が得られていることがわかる．

表 3　図で用いる凡例について

| 通信の起動＼通信関数 | scif_readfrom() | scif_writeto() |
|---|---|---|
| ホストプロセッサ | MIC → HOST by HOST | HOST → MIC by HOST |
| MIC プロセッサ | HOST → MIC by MIC | MIC → HOST by MIC |

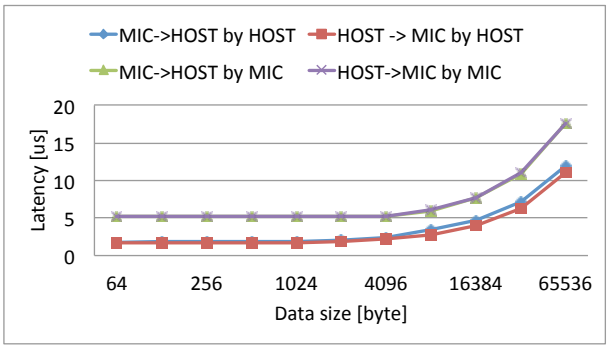


図 6　SCIF RMA レイテンシ

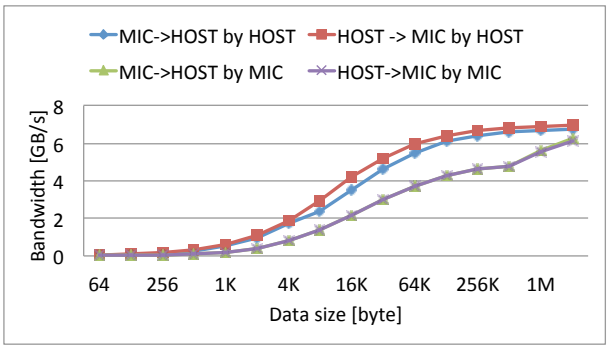


図 7　SCIF RMA バンド幅

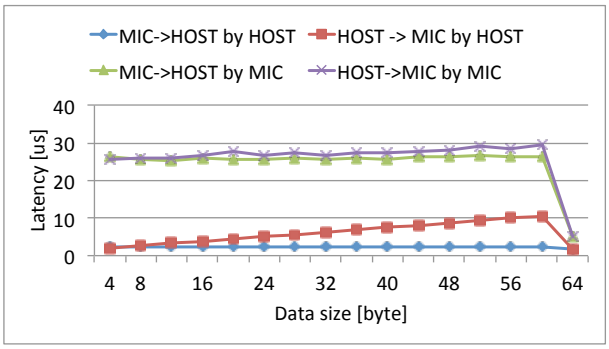


図 8　4 bytes 刻みでの SCIF RMA レイテンシ

### 6.3　64 bytes 未満の SCIF RMA 転送

SCIF の公式リファレンス [6] における，データサイズは 64 の倍数のデータサイズでない場合の性能低下を調査する．64 bytes 以下のデータサイズに対して，4 bytes 刻みでのレイテンシを示したものを**図 8** に示す．

図 8 より，MIC プロセッサによる通信起動の結果が顕著で，20$\mu$ 秒以上のレイテンシが発生する．また，ホストプロセッサによる通信起動でも，scif_writeto() の性能が低下しており，10$\mu$ 秒程度まで増加してしまう．現在，この問題の回避策を検討しているが，有効な手段が見つかっていない．以降，64 bytes 以上を対象として測定を行った．

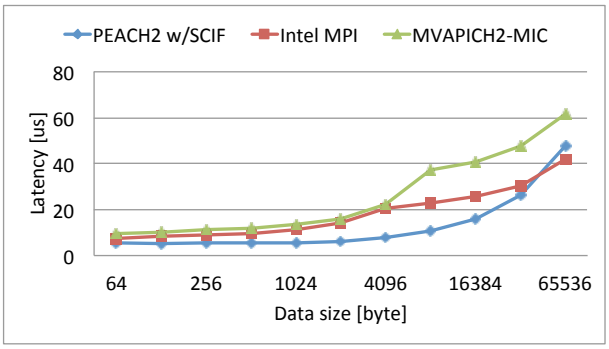


図 9　提案システムの通信時間測定結果

### 6.4　ノードをまたいだ MIC プロセッサ間の通信性能

**図 9** に，ノードをまたぐ MIC プロセッサ間通信のレイテンシを測定した結果を示す．凡例の “PEACH2 w/SCIF” が今回実装したシステムである．また，Intel MPI[7] による性能を比較対象として示している．さらに，Ohio State University によって開発されているオープンソースの MPI 実装である MVAPICH2 に対して MIC プロセッサ向けの最適化を適用した MVAPICH2-MIC 2.0[8] を用いた．

レイテンシは，64 byte から 2048 bytes まで 5$\mu$ から 6$\mu$ 秒前後で推移している．64 bytes の時点では，レイテンシは 5.27$\mu$ 秒となっている．MIC プロセッサ間の MPI による通信レイテンシは MVAPICH2-MIC では 9.56$\mu$ 秒であり，提案システムのレイテンシは 4$\mu$ 秒以上も上回ることができる．また，Intel MPI においても，64 bytes の時点で，7.44$\mu$ 秒なので，2$\mu$ 秒ほど，低いレイテンシを実現している．さらに，MVAPICH2-MIC を用いた公式に発表されている [9] 最小のレイテンシは 6.7$\mu$ 秒であるが，これよりも低いレイテンシを達成している．ただし，SCIF の転送性能の際に述べた，64 bytes よりも小さなメッセージサイズにおける通信は，現状の実装では，SCIF が原因となり，レイテンシが大幅に増加してしまうため，図 9 の測定においても，64 bytes 以上の評価となっている．

また，**図 10** に示すバンド幅においては，レイテンシの測定結果をもとに算出した．64KB 以上で，Intel MPI に劣ってしまう．また，MVAPICH2-MIC に対しても，128KB で性能が劣る．よって，提案するシステムは，小さいデータサイズにおけるレイテンシの削減を利点としている．

さらに，性能の律速が MIC プロセッサに対する Read や Write に偏ってないかを確認するため，一方のノードをホストプロセッサのメモリを対象とした評価を行った．バンド幅の比較を**図 11** に示す．

図 11 より，“local MIC → remote CPU” と “local CPU

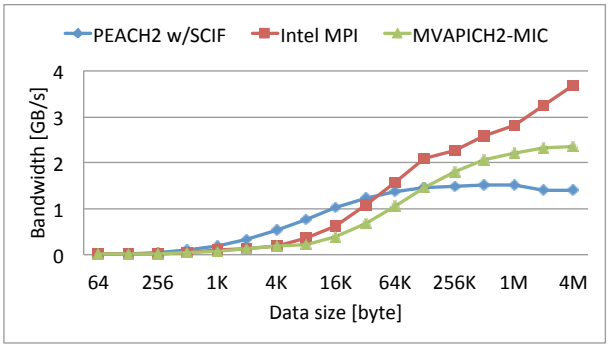


図 10　提案システムのバンド幅測定結果

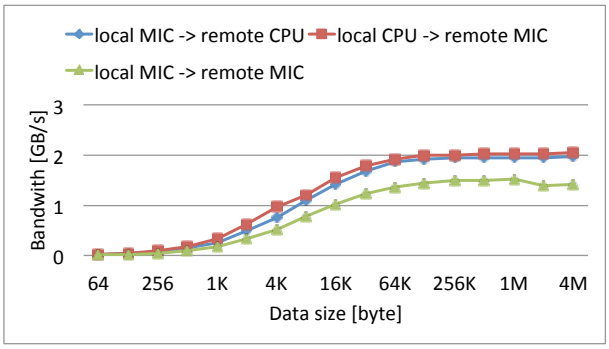


図 11　提案システムのバンド幅比較

→ remote MIC”，すなわち MIC プロセッサに対する Read と Write を比較して，大きな差が見られないことから，どちらかがボトルネックになっているということはないことが分かる．

## 7. おわりに

### 7.1 まとめ

MIC プロセッサに対し，TCA 通信機構を適用するための調査および実験を行った．SCIF を用いた実装を検討するため，MIC プロセッサ間の通信に必要な情報の収集を行った．また，MPSS で提供されている SCIF を用い，ホストプロセッサと MIC プロセッサ間の通信と，PEACH2 によるホストプロセッサ間通信を組み合わせて実装した．ホストプロセッサと MIC プロセッサ間における，SCIF を用いた RMA の性能を測定し，実装を進めた．PEACH2 によるノードをまたぐ MIC プロセッサ間の通信性能の測定結果から，Intel MPI による，MIC プロセッサ間通信に対し，32KB までは低いレイテンシで通信できている．また，MVAPICH2 を MIC プロセッサ向けに最適化した MPI 実装である MVAPICH2-MIC 2.0 に対しても，128KB まで低いレイテンシで通信が実現した．

### 7.2 今後の課題

今後は，実装をさらに進め，PEACH2 や SCIF のパフォーマンスのチューニングを行い，より低レイテンシな通信環境を構築する．また，SCIF の制約により，64 bytes 未満の通信のレイテンシが増大する問題を解決する必要がある．また，本来の目的である MIC プロセッサ間直接通信に関して，SCIF のカーネル空間用の関数の scif_get_pages() を用いた実装を行う予定である．

## 謝辞

本研究の一部は文部科学省特別経費 JST-CREST 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」，研究課題「ポストペタスケール時代に向けた演算加速機構・通信機構統合環境の研究開発」による．

## 参考文献

[1] TOP 500 The List. 入手先 ⟨http://www.top500.org⟩
[2] NVIDIA CUDA Zone. 入手先 ⟨https://developer.nvidia.com/cuda-zone⟩
[3] Visit to the National University for Defense Technology Changsha, China. 入手先 ⟨http://www.netlib.org/utk/people/JackDongarra/PAPERS/tianhe-2-dongarra-report.pdf⟩
[4] 塙 敏博, 児玉 祐悦, 朴 泰祐, 佐藤 三久. Tightly Coupled Accelerators アーキテクチャに基づく GPU クラスタの構築と性能予備評価. 情報処理学会論文誌コンピューティングシステム (ACS)，pp.14-25，2013.
[5] Intel MPSS Users Guide. 入手先 ⟨http://registrationcenter.intel.com/irc_nas/4834/mpss_users_guide.pdf⟩
[6] Symmetric Communications Interface (SCIF) Users Guide. 入手先 ⟨http://registrationcenter.intel.com/irc_nas/4834/scif_userguide.pdf⟩
[7] Intel MPI Library. 入手先 ⟨https://software.intel.com/en-us/intel-mpi-library⟩
[8] MVAPICH overview. 入手先 ⟨http://mvapich.cse.ohio-state.edu/overview/⟩
[9] MVAPICH2-MIC 2.0 Microbenchmarks. 入手先 ⟨http://mvapich.cse.ohio-state.edu/ performance/mic-pt_to_pt/⟩