Vol.2012-HPC-133 No.16
2012/3/26

メニーコア並列計算機における通信機構の設計

思　敏[†1]　石川　裕[†1,†2,†3]

本論文は、メニーコア並列計算機における DCFA（Direct Communication Facility for many-core based Accelerators）と呼ばれる直接通信機構を設計・実装する。DCFA は、計算ノード間が InfiniBand によって接続されるメニーコア間直接通信を提供する。評価結果により、サイズが Kbyte 単位のデータ通信に対して、DCFA はホスト間通信とほぼ同様な性能を達成する。また、DCFA 上の並列プログラミングインターフェースを提供するために、DCFA-MPI と呼ばれる MPI 通信ライブラリを設計する。現在、DCFA-MPI は実装中である。

Design of Communication Facility on Many-Core based Cluster

Min Si[†1] and Yutaka Ishikawa[?,†2,†3]

This paper designs and implements a direct communication facility, called DCFA, for many-core based cluster systems. The DCFA can provide direct communication between many-cores which are inserted in different computation nodes connected via InfiniBand. The evaluation results show that, for data transfer over Kbyte, the latency of DCFA delivers the same performance as that of host to host data transfer. This paper also designs an MPI library, called DCFA-MPI, for providing a parallel programming interface over DCFA. The implementation of DCFA-MPI is still in progress.

†1 東京大学 コンピュータ科学専攻
Department of Computer Science, University of Tokyo
†2 東京大学 情報基盤センター
Information Technology Center, University of Tokyo
†3 理化学研究所　計算科学研究機構
RIKEN Advanced Institute for Computational Science

1. はじめに

アクセラレータ型並列計算機とは、一台のホストサーバーに複数台の浮動小数点演算アクセラレータが PCI Express で繋がった計算ノードから構成される並列計算機である。GPGPU をアクセラレータにした並列計算機 (GPGPU クラスタ) が数多く存在する[1)]。一方、メニーコアをアクセラレータとする製品として、2011 年、Intel 社は Many Integrated Core(MIC) というメニーコアアーキテクチャに基づいた「Knights Corner」と呼ばれる新型プロセッサを発表した[2)]。MIC アーキテクチャは GPGPU のような他のアクセラレータと異なり、汎用プロセッサをベースとしたメニーコアであるためデバイスを直接操作できるという特長がある。

クラスタ環境上の並列計算には、計算ノード間の通信が不可欠である。PCI Express 規格により、PCI Express 上の 1 デバイスとするアクセラレータは通信デバイスなどの他のデバイスを初期化 (PCI Express Configuration) することはできない。GPGPU のようなアクセラレータの場合、通信デバイスにコマンドを発行できずに直接操作することはできない。このため、既存の GPGPU クラスタ環境では、異なる計算ノード上の GPU 間通信は、直接に送信側から受信側までデータ転送ができずにホスト経由となり、オーバヘッドが生じる。一方、MIC のようなメニーコア型アクセラレータはデバイスを初期化できないが、デバイスの制御アドレスさえわかれば、直接デバイスを操作できる。したがって、異なる計算ノード上のメニーコア間の直接通信は実現可能である。

本論文は、計算ノード間が InfiniBand によって接続されるメニーコア並列計算機におけるメニーコア間直接通信の実現を目的とし、Direct Communication Facility for manycore-based Accelerators (DCFA) という直接通信機構を設計する。DCFA では、計算ノード内のホストサーバーが通信デバイスを初期化してからその制御アドレスをメニーコア側に通知し、メニーコアは通信デバイスを各自で操作する。InfiniBand は Remote Direct Memory Access (RDMA) を提供し、HPC クラスタ環境の通信デバイスとして広く使われている。DCFA では、InfiniBand HCA（IB HCA）の内部構造をホスト側とメニーコア側のメモリ領域に分散し、メニーコア側はこれらの構造を制御して IB HCA 経由で通信する。DCFA を透明化してユーザが容易にアプリケーションを作成できることを目的とし、DCFA 上に MPI 通信ライブラリを設計する。本論文の実装では、InfiniBand として Mellanox 社の InfiniBand Host Channel Adapter (IB HCA)、メニーコアとして Intel 社の Knights Ferry、メニーコア実装環境として抽象化層 Accelerator Abstraction Layer (AAL)[3)] を使用した。

AAL は、メニーコア混在型システム向けのオペレーティングシステムのためのハードウェア抽象化層であり、ホスト側で動作する汎用 OS（AAL (Host)）、メニーコアプロセッサで動作するカーネル（AAL (Manycore)）、ホスト OS とメニーコア OS の通信機構（AAL (IKC)）の 3 部分から構成される。DCFA の実装は、AAL (Host) が提供するメニーコアメモリをホストにマップする操作、AAL (Manycore) が提供するホストメモリをメニーコアにマップする操作、AAL (IKC) が提供するホストとメニーコア間の通信操作を使用した。

本論文の構成は，以下のとおりである。第 2 章において、DCFA の設計と評価を行い，第 3 章で、DCFA-MPI を設計する。第 4 章でアクセラレータ型並列計算機の直接通信に関連する研究と、MPI 通信ライブラリに関連する関連研究を述べる。第 5 章で本論文のまとめと今後の課題について論じる。

2. DCFA の設計と評価

2.1 InfiniBand 通信の概要

2.1.1 通 信 構 造

InfiniBand とは、サーバー間相互接続用の I/O アーキティクチャ、通信装置を定義する業界標準規格の 1 つであり、サーバー間の低遅延且つ高速的な通信を提供する。現在、HPC コンピューティングやデータセンターで広く使われている。InfiniBand Architecture (IBA)[4] では、InfiniBand 通信スタックは複数の通信レイヤーで構成される。トランスポート層としてユーザプログラム向けのインタフェース（**図 1** を参照）が提供される。このインタフェースはキューモデルを使用し、下記構造体が定義されている。

- Queue Pair (QP)

QP は、Send Work Queue (SQ) と Receive Work Queue (RQ) の 2 つから構成される。ユーザープログラムは送信・受信をする時、必ず送受信要求を QP に入れる。送信・受信要求は、Work Queue Entity (WQE) を生成して対応のキュー（送信要求は SQ、受信要求は RQ）に格納することにより実現される。IB HCA は FIFO 順で WQE を処理する。

- Completion Queue (CQ)

SQ も RQ もそれぞれ 1 個の Completion Queue (CQ) に繋がる。送信・受信要求の処理が完了した時、IB HCA は Completion Queue Entity (CQE) を生成して CQ に入れる。ユーザプログラムは CQ をチェックして処理結果を確認できる。

- Memory Region (MR)

MR とは、1 個のメモリ領域及びその領域にアクセスする権限を定義する構造体である。

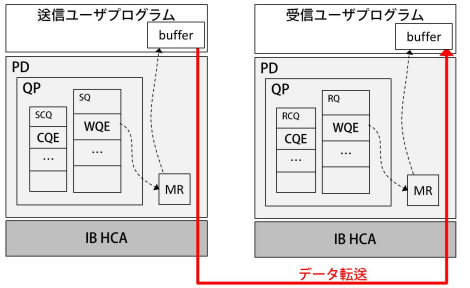

図 1 InfiniBand 通信インタフェース

図 2 InfiniBand のコンポーネント

ユーザープログラムは送受信する時、必ず前もって対象のデータバッファーの MR を登録しなければならない。

- Protection Domain (PD)

1 個のノードは同時に複数のノードと通信する可能性があるため、IBA は PD という構造体を定義して、QP のアクセス領域を制限する。QP と MR はそれぞれ PD に登録して、QP は同じ PD に登録された MR のみアクセスできる。

InfiniBand 規格は、InfiniBand Verb (IB Verb) と呼ばれるユーザレベルライブラリを定義している。このライブラリは IB HCA のカーネルドライバを経由して IB HCA にコマンドを送信する（**図 2** を参照）。キューモデルをメニーコア側に移行するには、キューの内部構造を解析する必要がある。本論文の実装に使用した Mellanox 社の IB Verb と IB HCA カーネルドライバのソース[5] から、QP も CQ も下記記述の通り、1 個のバッファー領域と 1 個の Doorbell レコードより構成されていることが分かる（**図 3** を参照）。

- QP

QP 作成時、IB Verb 内でホストメモリ上にバッファー領域が割り当てられてからカーネルドライバに渡される。このバッファーは SQ と RQ の 2 個のリングバッファーに分けられ、WQE がバッファー内に確保される。また、Doorbell レコードは IB HCA のメモリ空間にマップされた PCI アドレス空間より割り当てられ、新しい WQE が到着する通知を行うための構造として使われる。

- CQ

QP と同様に、CQ を作成する時にホストメモリ上にバッファー領域が割り当てられ、CQE

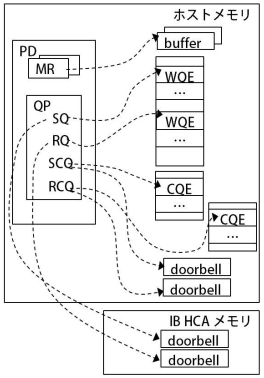

図 3 InfiniBand 内部構造

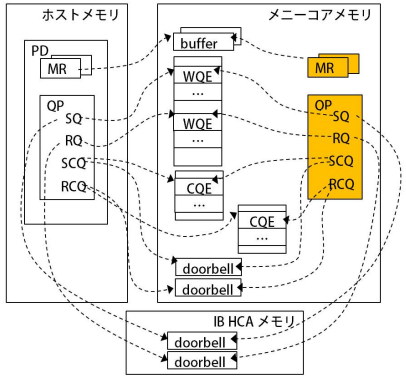

図 4 DCFA 内部構造

がバッファー内に確保される。CQ の Doorbell レコードは QP と異なり、ホストメモリ上に割り当てられ、バッファーのリング構造を維持するために用いられる。IB HCA は CQE を作成して、ユーザプログラムは CQE を取り出す。CQE を取り出す度に、ユーザプログラムは必ずこの Doorbell レコードをアクセス (Ring) して IB HCA 側に通知する。

2.1.2 通信プログラム

IB Verb を用いて、1 対 1 の InfiniBand 通信プログラム（IB 通信プログラムと省略する）は常に下記順序で実装される。

(1) InfiniBand デバイス初期化
IB HCA ドライバはデバイスを初期化する。

(2) InfiniBand 資源初期化
PD、CQ、QP をそれぞれ 1 個作成する。

(3) MR 登録
送受信のデータバッファーを前もって作成し、2 の PD に MR を登録する。RDMA 通信を行う場合は、RDMA 権限が許可される MR(RDMA-MR と省略する) を常に 1 個用意する必要がある。

(4) 他ノードと接続
他のノードと QP の情報を交換し、2 個の QP 間でコネクションを確立する。RDMA 通信を行う場合は、RDMA-MR の情報も交換して、QP に RDMA 権限を許可する。

(5) データ転送

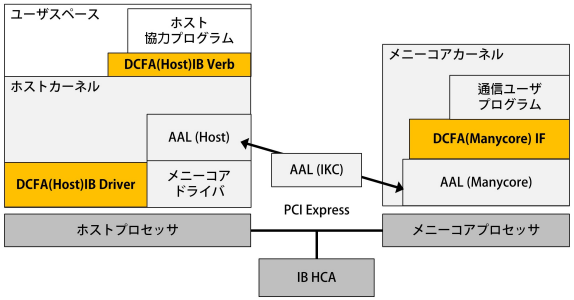

図 5 DCFA のコンポーネント

Send/Receive と RDMA の 2 種類の転送モードが定義されている。i) Send/Receive モードで送受信する場合、受信ノードは先に受信要求を出してから送信ノードは送信要求を出す。両方も CQ チェックで送受信完了を確認できる。ii) RDMA モードの場合は、ローカルノードは RDMA read か write 要求を出し、直接にリモートノードの RDMA-MR にデータを読取・書込む。RDMA 処理の完了は、RDMA 要求を発行したノードで CQ をチェックすることで確認できる。

(6) InfiniBand 資源解放処理
PQ、CQ、QP、MR など全部の InfiniBand 資源を解放する。

2.2 DCFA の設計

2.2.1 コンポーネント構成

第 2.1.1 節で説明した InfiniBand のキューの内部構造の分析から、メニーコア側がキューのバッファーと Doorbell レコードにアクセスできればキューを使用できるということがわかる。以上の考えで、ホストメモリに割り当てる構造を全てメニーコアメモリに移行するという DCFA の設計方針が決められる。DCFA は、ホスト側の改造された IB Verb（DCFA(Host) IB Verb）、カーネルドライバ（DCFA(Host) IB Driver）、メニーコア側の IB インターフェース（DCFA(Manycore) IF）、の 3 つのコンポーネントより構成され、メニーコアハードウェア抽象化層 Accelerator Abstraction Layer（AAL）を通じて協同動作する（**図 5** を参照）。

- メニーコアメモリマップ
AAL(Host) が提供するメモリマップ操作である aal_host_map_memory 関数と aal_host_unmap_memory

関数を用いて、メニーコアのメモリ空間の物理アドレスを指定してその領域をホスト側のカーネルスペース内にマップできる。DCFA(Host) IB Verb と DCFA(Host) IB Driver は、マップされた領域を使用して、メニーコア側からもホスト側からも IB HCA 側からもアクセスできる構造体を作成できる。逆に、AAL(Manycore) が提供する aal_mc_map_memory 関数と aal_mc_unmap_memory 関数を用いて、ホストのメモリ空間の物理アドレスをメニーコアのメモリ領域へもマップできる。これにより、DCFA(Manycore) IF はホストと同様に、IB HCA のメモリ空間にマップされた PCI アドレス空間にアクセスできる。

- IB Verb の改造（DCFA(Host) IB Verb）

IB Verb に、PCI Express デバイス上のメモリ領域の割当・解放機能を追加することで、メニーコア上のメモリ領域を割り当てる機能が実現される。割当ての詳細は以下のとおりである（**図 4** を参照）。

– QP

QP のバッファーをメニーコアのメモリ空間にマップされた領域から割り当てる。Doorbell レコードは依然に IB HCA のメモリ空間にマップされた PCI アドレス空間より割り当てる。

– CQ

CQ のバッファーも Doorbell レコードもメニーコアのメモリ空間にマップされた領域から割り当てる。

– MR

MR に登録するデータバッファーはメニーコアのメモリ空間にマップされた領域より割り当てる。

- IB HCA カーネルドライバの改造（DCFA(Host) IB Driver）

IB HCA カーネルドライバ内には、ユーザスペースから指定したメモリアドレスの DMA アドレスを取得する関数がある。この関数は元々ホストメモリのアドレスのみ処理する。この関数を変更して、メニーコアメモリにマップされた領域のアドレスであっても正常に DMA アドレスを取得できるように改造する。

- DCFA(Manycore) IF のデザイン

DCFA(Manycore) IF は、一部の InfiniBand 通信関数と、ホスト側との通信関数を提供する。メニーコア側で動作するユーザプログラムはこのインターフェースにより、IB HCA を経由して、外部ノードと直接通信することができる。オリジナルの InfiniBand 通信と一致するように、メニーコア側でもキューモデルが定義される（図 4 を参照）。但し、現在、メニーコア側のメモリ管理機制はまだ設計していないため、PD 構造は定義されていない。また、InfiniBand の資源初期化・解放等の処理はホスト協力プログラムで処理するため、ホスト側との通信が必要となる。この通信は AAL (IKC) で実現する。

2.2.2 通信プログラム

以上の改造で、メニーコア上で動く 1 対 1 の IB 通信プログラムは下記順序で実装される（**図 6** を参照）。

(1) InfiniBand デバイス初期化
ホスト側の IB HCA ドライバはデバイスを初期化する。

(2) InfiniBand 資源初期化
メニーコア側は資源初期化のパラメータを定義し、「IB 資源初期化」コマンドをホスト側に発行する。ホスト側はコマンドを受信してから、第 2.2.1 節で述べたメモリ割当の通り、PD、CQ、QP をそれぞれ 1 個作成する。作成後、ホスト側は「初期化済」通知と、データ転送に必要となる QP の初期化情報をメニーコア側に返信する。メニーコア側は通知を受けたら、DCFA(Manycore) IF の関数を呼出し、自分の資源初期化を行う。ここで注意すべきことは、メニーコア側で行う資源初期化は、ホスト側と同様の API を使えるように構造体を作成するだけで、IB HCA に要求を出していない。

(3) MR 登録
メニーコア側が初期化を行うと同時に、ホスト側は MR の登録を行う。登録後、ホスト側は「登録済」通知と、データ転送に必要となる MR の登録情報をメニーコア側に返信する。(2) と同様に、メニーコア側は通知を受けてから MR 構造体を作成する。

(4) 他ノードと接続
メニーコア側は他ノードと接続する時、「接続」コマンドをホスト側に出して、ホスト側からの「接続済」通知を待つ。

(5) データ転送
ホスト通信プログラムと同様に、Send/Receive モード若しくは RDMA モードでデータを転送できる。(2)、(3) の段階で作成した QP、CQ、MR の構造体を利用して、DCFA(Manycore) IF の関数を呼出し、データを IB HCA 経由で直接に他ノードへ転送する。

(6) InfiniBand 資源の解放処理

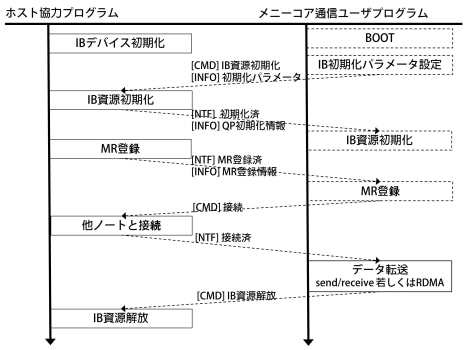

図 6 DCFA の通信プログラム

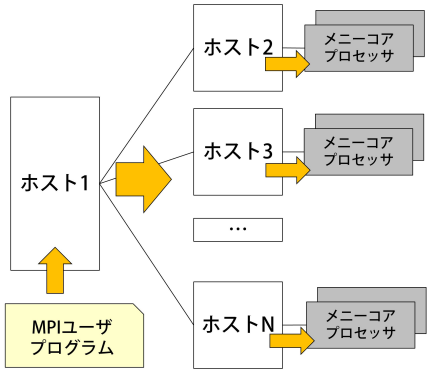

図 7 DCFA-MPI の全体図

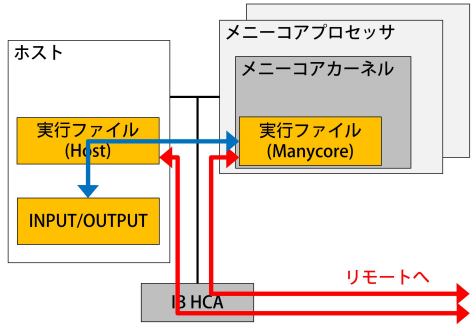

図 8 DCFA-MPI における 1 ノード上のタスク割り当て

表 1 実験環境

マシン	Intel Workstation
M/B	Intel S5520SCR
CPU	Intel Xeon X5680 3.33GHz x 2
IB HCA	Mellanox MT26428
メニーコアカード	Knights Ferry x 1
オペレーティングシステム	Red Hat Enterprise Linux Server release 6.1
Mellanox OFED Version	1.5.3

メニーコア側は「IB 資源解放」コマンド出し、ホスト側は全部の InfiniBand 資源を解放する。

2.3 DCFA の評価

DCFA の性能を評価するために、IB Verb を利用したホスト間の 1 対 1 通信（ホスト）と、DCFA を利用したメニーコア間の 1 対 1 通信（DCFA）をそれぞれ実行し、Send/Receive 転送モードと RDMA write 転送モードの遅延相対値をそれぞれ算出した。通信プログラムの概要は、第 2.1.2 節と第 2.2.2 節で述べた処理内容と同様のプログラムを用いた。評価環境は**表 1** に示す。

Intel 社との秘密保持契約により本論文に評価結果の詳細を述べられないが、サイズが Kbyte 単位のメッセージに対して DCFA はホスト間通信とほぼ同様の性能が達成できていることを確認している。今回、ホストを介在したメニーコアメモリ間通信を実装していないので、ホストを介在した場合の通信遅延差を比較出来なかった。ホストを介在した場合はメニーコアからホストへの要求依頼のコストが上乗せされるので、より通信遅延が増大することは自明である。

3. DCFA-MPI 通信ライブラリの設計

クラスタ環境上で並列アプリケーションを作成するには、MPI 通信ライブラリを使うのが主流である。メニーコアの直接通信を実現した DCFA の上に、本論文は DCFA 上の MPI 通信ライブラリ（DCFA-MPI）も設計する。このライブラリを通して、DCFA の直接通信という利点を活用できるメニーコアクラスタ環境上に動作する並列プログラミングインターフェースの提供を図る（**図 7** を参照）。ライブラリはまだ実装中であるため、以下、基本設計を述べる。

3.1 DCFA-MPI のタスク分割

DCFA では、メニーコアから直接に IB HCA を通じて他ホスト若しくは他ホスト上のメニーコアへデータを転送できるため、計算ユニットとなったメニーコアは各自で通信処理を担当することが可能である。しかしながら、メニーコアは小さいキャッシュサイズやインオーダー実行などの特徴があるため、MPI プロセス生成、MPI-IO や集団通信部分のような重い処理はホスト側で処理した方が効率が良いと考えられる。以上の考察から、1 ノード上で実行するタスクを以下のように分割する（**図 8** を参照）。i) 浮動小数点演算はメニーコ

アで担当する。ii)MPI プロセス生成、MPI-IO はホスト CPU で担当する。iii) MPI の通信処理を Point-to-Point 通信と、Collective 通信やユーザ定義データ通信などの重い通信処理に分類して、Point-to-Point 通信はメニーコア若しくはホスト各自で担当し、重い通信処理はホスト側にオフロードする。

3.2 DCFA-MPI のプログラミングモデル

DCFA-MPI を用いて、普通のクラスタ環境上で実行する MPI 並列プログラムを容易にメニーコアクラスタ環境に移行できる。第 3.1 節で述べたタスク分割を基本方針とし、以下は DCFA-MPI が既存 MPI プログラムをコンパイル・実装する例を記述する。**図 9** は普通のクラスタ環境上で実行する MPI プログラムであり、以下 6 ステップより構成される。i)MPI_Init で MPI 資源を初期化する。ii) 入力値を読込む。iii) 各ノードがループ内で計算を行い、MPI_Send/MPI_Recv で各ノード間の Point-to-Point 通信を行う。iv)Collective 通信の MPI_Reduce で集約処理を行う。v) 計算結果を出力する。vi)MPI_Finalize で資源を解放する。

DCFA-MPI でコンパイルして生成したメニーコア側の実行ファイルは**図 10** で示す。i)MPI 資源の初期化には InfiniBand 資源の初期化や MPI プロセス管理の初期化があるため、ホスト側で実行する。メニーコア側は「MPI_Init」コマンドを発行し、ホスト側からの初期化情報を受ける。ii) 入力読込はホスト側で担当し、メニーコア側は「INPUT 読込」コマンドを発行し、ホスト側から配った入力値を受ける。iii) 各メニーコアはループ内の計算を行い、Point-to-Point 通信を実行する。iv)MPI_Reduce には全体ノードへの送信が発生し重い通信となるため、ホスト側で実行する。メニーコア側は「MPI_Reduce」コマンドと最新のパラメータ値を発行し、ホスト側からの集約結果を受ける。v) メニーコア側は「OUTPUT 出力」コマンドと計算結果を発行し、ホスト側は結果を出力する。vi) メニーコア側は「MPI_Finalize」コマンドを発行し、終了する。

メニーコア側からのコマンドを受けるホストモジュールは**図 11** で示す。i)MPI が起動する最初に、ホスト側は管理プロセスを起動し、メニーコア側からのコマンドを待つ。ii) コマンドを受信したら、対応の処理を行う。1 台のホストに複数個のメニーコアプロセッサが存在する場合、効率的な並列コマンド処理機構を考える必要がある。

4. 関 連 研 究

本章はクラスタ環境の直接データ転送に関連する研究と、アクセラレータ型並列計算機上の MPI 通信ライブラリに関連する研究を述べる。

```
MPI_Init( );
INPUT読込;
for(;;;){
        浮動小数点演算 ;
        MPI_Send( );
        MPI_Recv( );
}
MPI_Reduce( );
OUTPUT出力;
MPI_Finalize( );
return 0;
```

図 9 MPI 擬似コード

```
ホストに[MPI_Init] コマンドを発行する;
ホストから初期化情報を受信する;
メニーコア側のMPI資源を初期化する;

ホストに[INPUT読込]コマンドを発行する;
ホストからINPUT値を受信する;
for(;;;){
   浮動小数点演算 ;
   MPI_Send( );
   MPI_Recv( );
}
ホストに[MPI_Reduce]コマンドとパラメータ値を発行する;
ホストから集約結果を受信する;


ホストに[OUTPUT出力]コマンドを発行する;


ホストに[MPI_Finalize]コマンドを発行する;
return 0;
```

図 10 DCFA-MPI がコンパイルしてメニーコア側の実行擬似コード

```
[MPI起動]
  ホスト管理プロセスが立ち上がる;

[MPI_Init]
  MPI資源を初期化する;

[INPUT読込]
  INPUTを読み込む;
  INPUT値を各メニーコアに配る;

[MPI_Reduce]
  メニーコアから変数値を受信する;
  MPI_Reduceを実行する;
  MPI_Reduceの結果をメニーコアに送信する;

[OUTPUT出力]
  メニーコアからOUTPUT値を受信する;
  OUTPUTを出力する;

[MPI_Finalize]
  MPI資源を解放する;
```

図 11 MPI 擬似コードが使用した DCFA-MPI 内でインストール済のホストモジュール

4.1 OPIOM

OPIOM[6] は並列ファイルシステムにおいて Myrinet を経由してリモートディスクへの直接 I/O を実現する機構である。リモート読取操作を例として説明する。リモート読取操作とは、リモートサーバーのディスクよりデータを読取ることで、一般には、i) ディスクからカーネルメモリスペースへのデータ転送、ii) カーネルメモリスペースからユーザアプリケーションメモリ上のバッファーへの転送、iii) ユーザアプリケーションメモリのバッファーから Myrinet を経由してリモートメモリバッファーへの転送、の 3 つの転送段階から構成される。OPIOM では、i) と ii) の段階をなくし、Myrinet 経由でデータをディスクからリモートメモリバッファーへ直接に転送できる。この設計は Myrinet だけでなく、DMA 転送と PCI アドレスメモリマップを提供する全ての通信デバイスに適用できると考えられる。

4.2 GPUDirect

2010 年に NVIDIA 社から公開された GPUDirect v1.0[7] は、GPGPU クラスタ環境における GPU からのデータ転送を加速する技術である。GPUDirect 以前の転送は、i) 送信 GPU のメモリ領域から所属ホストのメモリ領域へのデータ転送、ii)GPU がアクセスできるホストメモリ領域より InfiniBand がアクセスできるホストメモリ領域へのデータ転送、iii)InfiniBand がアクセスできるホストメモリ領域より InfiniBand 経由でリモートノードメモリへのデータ転送、の 3 段階から構成されていた。GPUDirect v1.0 技術は ii) の転送を除去するだけで、GPU のメモリ領域からホストメモリへの転送が依然残る。

4.3 MVAPICH2-GPU ライブラリ

MVAPICH2-GPU ライブラリ[8][9] は、GPGPU クラスタ上の GPU 間の通信を支持する MPI 通信ライブラリである。このライブラリでは、GPGPU がデバイスを操作できない欠点のため、CUDA データ転送関数を利用して、GPU メモリ内のデータをホストメモリに CUDA コピーしてから InfiniBand 通信を行う方法で GPU 通信を行っている。

5. おわりに

メニーコアをアクセラレータとして搭載するクラスタが今後増えていくと考えられる。本論文はこのようなクラスタ環境におけるメニーコア間の直接通信を提供する機構 DCFA を設計・実装した。DCFA では、ホストを介せずにメニーコアは自ら IB HCA を操作して通信を行う。評価結果により、サイズが Kbyte 単位のデータ通信に対して、DCFA はホスト間通信とほぼ同様な性能を達成した。本論文は、さらに、メニーコアクラスタ上の並列プログラミング環境を提供するために、DCFA を用いた MPI 通信ライブラリ DCFA-MPI を設計した。現在、DCFA-MPI を実装している。

今後の課題として、本論文が記述した DCFA-MPI の基本機能を実現した上に、各計算ノードのホストを効率的に資源管理してタスクをバランスする機構を考慮する必要がある。その例として、ホストがメニーコア側の演算を待つ場合、ホストの資源を浪費せずに、他の演算タスクや、ディスク I/O タスクに切替えるというケースが挙げられる。

謝辞 Intel 社からは Knights Ferry の利用支援並びに技術的議論に参加頂いた。ここに感謝する。本研究の一部は、科学技術振興機構「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」の支援を受けた。

参 考 文 献

1) TOP500.org: TOP500 Supercomputing Sites, TOP500.org (online), available from ⟨http://www.top500.org⟩ (accessed 2011-11).
2) Intel Corporation: First Intel Many Integrated Core Co-processor Demonstrated to Deliver Performance Above 1 TFLOPS.
3) 下沢 拓, 石川 裕, 堀 敦史, 並木美太郎, 辻田祐一：メニーコア向けシステムソフトウェア開発のための実行環境の設計と実装, 情報処理学会研究報告. [システムソフトウェアとオペレーティング・システム], Vol.2011, No.1, pp.1–7（オンライン）, 入手先⟨http://ci.nii.ac.jp/naid/110008583265/⟩ (2011-07-20).
4) InfiniBand Trade Association: *InfiniBandTM Architecture Specification, Volume 1, Release 1.2.1* (2007).
5) Mellanox Technologies: Mellanox OpenFabrics Enterprise Distribution for Linux (MLNX_OFED), `http://www.mellanox.com/content/pages.php?pg=products_dyn&product_family=26&menu_section=34`.
6) Geoffray, P.: OPIOM: off-processor IO with Myrinet, *Proceedings of First IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp.261–268 (online), DOI:10.1109/CCGRID.2001.923202 (2001).
7) Mellanox Technologies: NVIDIA GPUDirect Technology - Accelerating GPU-based Systems, `http://www.mellanox.com/pdf/whitepapers/TB_GPU_Direct.pdf`.
8) Wang, H., Potluri, S., Luo, M., Singh, A., Sur, S., Panda, D., Wang, H., Potluri, S., Luo, M., Singh, A. K., Sur, S. and Panda, D. K.: MVAPICH2-GPU: optimized GPU to GPU communication for InfiniBand clusters, *Computer Science - Research and Development*, Vol. 26, No. 3, pp. 257–266 (online), DOI:10.1007/s00450-011-0171-3 (2011).
9) Wang, H., Potluri, S., Luo, M., Singh, A., Ouyang, X., Sur, S. and Panda, D.: Optimized Non-contiguous MPI Datatype Communication for GPU Clusters: Design, Implementation and Evaluation with MVAPICH2, *Proceedings of IEEE In-*

ternational Conference on Cluster Computing (CLUSTER), pp.308 –316 (online), DOI:10.1109/CLUSTER.2011.42 (2011).