Vol.2012-HPC-133 No.15
2012/3/26

メニーコア混在型並列計算機における委託機構を用いたMPI通信基盤

吉永一美[†1,†5] 六車英峰[†1,†5] 辻田祐一[†1,†5]
堀敦史[†2,†5] 並木美太郎[†3,†5] 佐藤未来子[†3,†5]
下沢拓[†4] 石川裕[†4,†2]

現在，我々はエクサスケールを目指し，メニーコアプロセッサとマルチコアプロセッサが混在するハードウェア構成であるメニーコア混在型並列計算機上で，高スケーラブルで効率の良い MPI 通信を実現する基盤システムソフトウェアの研究開発を行っている．メニーコアプロセッサはマルチコアプロセッサと比較して，全体で高い演算性能を有する一方で，コアあたりの演算性能やメモリ容量は低い．我々はメニーコアプロセッサで MPI 関数を呼び出すと，適宜マルチコアプロセッサ側にある委託プロセスに通信処理を委託する仕組みを検討し，マルチコアプロセッサ側で負荷が大きな通信や入出力を，メニーコアプロセッサ側で高性能な演算を行うことで，全体の性能の向上を図っている．本稿では，この開発の中で既に実施したキャラクタデバイスによるものに加え，ポーリング方式によるメニーコア・マルチコアプロセッサ間の同期・通信機構を実装し，その予備評価を行った結果について報告する．さらに我々が想定する大規模環境での実現のためには，MPI のコミュニケータ情報の管理について特にメモリ使用量を抑える必要があるため，これに関する検討中の案を述べる．

MPI Communication Infrastructure Using Delegation Mechanism for a Hybrid Parallel Computer with Multi-Core and Many-Core CPUs

Kazumi Yoshinaga,[†1,†5] Hidetaka Muguruma,[†1,†5]
Yuichi Tsujita,[†1,†5] Atsushi Hori,[†2,†5]
Mitaro Namiki,[†3,†5] Mikiko Sato,[†3,†5]
Taku Shimosawa[†4] and Yutaka Ishikawa [†4,†2]

In this paper, we propose a scalable MPI communication infrastructure for a hybrid parallel computer with many-core and multi-core CPUs towards the exascale era. In general, many-core CPUs have high computing power, however available memory space per core is smaller as well as computing power per core is lower compared with multi-core CPUs. In order to realize a scalable MPI communication infrastructure, our proposal realizes a delegation mechanism from many-core CPUs to multi-core CPUs when an MPI function is called. We believe that this mechanism improves total performance by deploying heavy communications and I/O on multi-core CPUs and parallel computations on many-core CPUs. In this paper, we report feasibility study through performance evaluation between original MPI implementation and our delegation-based implementation. Here we have evaluated polling notification mechanism in addition to already proposed character device mechanism in the delegation-based one. Furthermore, we also propose an effective MPI communicator management mechanism which minimizes memory resource utilization on such huge scale parallel computing environment.

1. はじめに

スーパーコンピュータの計算性能は，昨年 11 月に発表された TOP500 において 10 ペタフロップスを達成し，トレンドグラフでは 2018 年にもエクサフロップスを達成することが予想されている．このエクサフロップスを達成するための技術として，Intel 社の Many Integrated Core (MIC)[1] や Single-chip Cloud Computer (SCC)[2] といった，メニーコアプロセッサに注目が集まっている．

メニーコアプロセッサは，一般的なマルチコアプロセッサと比較してコア数が多く並列演算に特化した性能を持つだけでなく，消費電力あたりの性能が高いという特長を持っている．エクサスケールの実現に向けては消費電力の増加を如何に抑えつつ性能を向上させるかが大きな課題となっており，この問題を解決すべく，今後メニーコアプロセッサを利用したスーパーコンピュータが HPC の一つの主流になると予想される．実際にテキサス大学の

†1 近畿大学
Kinki University
†2 理化学研究所 計算科学研究機構
RIKEN AICS
†3 東京農工大学
Tokyo University of Agriculture and Technology
†4 東京大学
The University of Tokyo
†5 独立行政法人科学技術振興機構 CREST
JST CREST

TACC が，Intel MIC アーキテクチャ製品の Knights Corner[3] を搭載する，メニーコア混在型並列計算機 Stampede を 2013 年に導入することを発表している[4]．

しかしメニーコアプロセッサには，マルチコアプロセッサよりもコア単体の性能が低く，コアあたりのメモリやコアが有するキャッシュメモリの容量が少ないという欠点が存在する．そのため，バッファ領域を十分に確保できず通信や I/O などの実装が困難である上，さらにメモリ・キャッシュ領域の逼迫によりアプリケーションプログラムの性能低下を招いてしまう．

また，エクサスケールを実現する上でもう一つ問題となるのが，アプリケーションのスケーラビリティの確保である．現在並列アプリケーションの構築に広く用いられている MPI をエクサスケール計算環境に移植した場合，プロセス数が劇的に増大する．その結果，コミュニケータ情報の管理に用いるメモリ使用量が増加し，情報検索に要する時間も非常に大きくなるため，スケーラビリティの確保が困難になることが予想される．特にメニーコアプロセッサはメモリ領域が小さいため，既存の MPI 実装をそのまま移植することは現実的ではない．スケーラビリティを確保する方法として，MPI+OpenMP などのハイブリッド MPI を用いるプログラミング手法も存在するが，既存のアプリケーションの移植性などを考慮すれば，エクサスケールコンピュータ上で効率的に動作する MPI 通信基盤システムが必要不可欠である．

このような背景から我々は，汎用マルチコアプロセッサをホスト CPU として利用し，メニーコアプロセッサをアクセラレータとして利用するメニーコア混在型並列計算機に着目し，その上で動作する MPI 通信基盤システムの構築を行なっている．提案するシステムでは，マルチコアプロセッサに通信や I/O などの処理を委託することで，メニーコアプロセッサの性能低下を回避し並列演算に専念させるとともに，MPI コミュニケータを階層的に管理することでメモリ消費量を低減した，スケーラブルな MPI 通信基盤システムを目指す．

今回我々は，既に開発したキャラクタデバイスでの同期機構に加え，ポーリング機構方式も構築し，両者の性能評価を行った．その結果，ポーリング機構方式の方がより短い時間で同期が行えるが，一方で CPU 使用率が高くなるため，今後 MPI 通信方式等に応じて動的に最適な方式を選択できる仕組みの必要性を確認した．さらに，今後のエクサスケールに向けて，コア当たりのメモリ容量が少なくなる傾向を鑑み，MPI コミュニケータの効率的な管理方法について検討を行った．

本稿では，2 章において提案する MPI 通信基盤システムソフトウェアの設計について述べ，3 章では，模擬環境上での試験実装を用いた通信性能の評価を行う．4 章では，現在検討中である MPI コミュニケータ情報の階層的な管理手法について示し，5 章で関連研究について述べる．そして 6 章で本稿のまとめを行う．

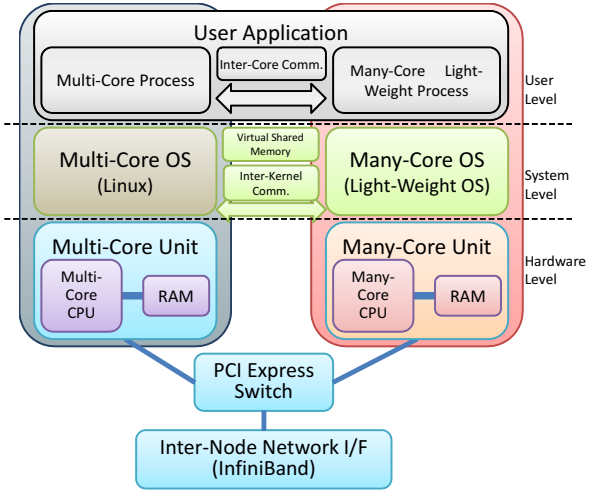

図 1 対象とする計算機環境の概念図

2. MPI 基盤システムソフトウェアの設計

2.1 対象とする環境

本研究では，マルチコアプロセッサとメニーコアプロセッサがノード内に混在した，メニーコア混在型並列計算機アーキテクチャ環境を対象とする．**図 1** にシステムの概略図を示す．

2.1.1 ハードウェア環境

ノード内には，ホスト CPU として用いるマルチコアプロセッサと，アクセラレータとして用いるメニーコアプロセッサが存在する．メニーコアプロセッサはアクセラレータボードに搭載され，一枚あるいは複数枚が PCI Express バスを経由してマルチコアプロセッサと接続される．

各ノード間は InfiniBand 等の高速ネットワークにより接続され，大規模な並列計算機クラスタを構成する．

2.1.2 ソフトウェア環境

本研究が対象とする環境においては，マルチコアプロセッサとメニーコアプロセッサの双方において OS が動作していることを想定している．マルチコアプロセッサ上では，一部をカスタマイズされた Linux などの汎用 OS が動作し，マルチコアプロセッサ側資源の管理や通信などの処理を行うとともに，メニーコアプロセッサ側の資源管理も行う．一方のメニーコアプロセッサ上では，メニーコアボード上の CPU とメモリを管理する軽量な OS が動作している[5]．

両 OS のカーネル間にはカーネル間通信機構を備えており，両プロセッサ間の連携が可能である．また，マルチコアプロセッサ側とメニーコアプロセッサ側双方のメモリ領域は，PCI Express バスを通じて互いにマッピング可能であり，仮想的な共有メモリ空間が作成できる．ユーザアプリケーションでは，これらの機構を用いることにより，コア間通信機構が実現されていることを想定している．

2.2 通信委託機構を備えた MPI 通信基盤

本研究が提案する MPI 通信基盤システムでは，メニーコアプロセッサで発行された MPI 命令を，メモリやキャッシュが潤沢に存在するマルチコアプロセッサに委託することで，効率的な通信や I/O を実現する．その結果コア数が多く高い並列演算能力を持つメニーコアプロセッサは，並列度の高い演算処理の実行に専念する事となり，マルチコアプロセッサとメニーコアプロセッサ，両者の特長を活かした効率的な MPI プログラムの実行が可能な MPI 通信基盤システムソフトウェアを実現する．

MPI 基盤システムソフトウェアの概略図を**図 2** に示す．メニーコアプロセッサで発行された MPI 命令の委託処理を行うために，マルチコアプロセッサ上に MPI delegatee（以下，Delegatee と呼ぶ）という代行処理サーバプロセスを起動する．ただし，全ての MPI 命令を Delegatee に委託するのではなく，ノード内での通信はクライアントプロセス間で直接通信を行うことで，処理依頼のオーバーヘッドを回避した高速な通信を実現する．

MPI 基盤システムソフトウェア上での MPI プログラム実行は以下のように行われる．

(1) mpiexec などの MPI 起動プログラムが実行されると，まずマルチコアプロセッサ上に Delegatee を起動する．
(2) Delegatee は，並列演算や MPI 処理を行うクライアントプロセスを，メニーコアプロセッサ上に起動する．
(3) クライアントプロセス内で MPI 命令が発行されたとき，その通信相手がボード内であればクライアントプロセス間で直接通信を行い，処理を完了する．ボード外またはノード間をまたがる通信であった場合，コア間通信機構を用いて MPI 処理依頼を Delegatee へと通知する．処理依頼に必要な情報として，MPI 命令の ID，その MPI 命令に必要な引数，通信に使用するデータ領域のアドレスなどを同時に送信する．
(4) MPI 処理依頼を受け取った Delegatee は，受け取った命令 ID から実行する MPI 命令を決定し，受け取った引数を用いて実際に MPI 処理を実行する．送受信に利用するデータ領域には，メニーコアボードのメモリの指定されたアドレス領域をマッピングして用いることで，マルチコアプロセッサとメニーコアプロセッサ間でのデータコピーのない通信を実現する．
(5) MPI 処理の返り値などの実行結果は，コア間通信機構を用いてクライアントプロセスへと送信される．

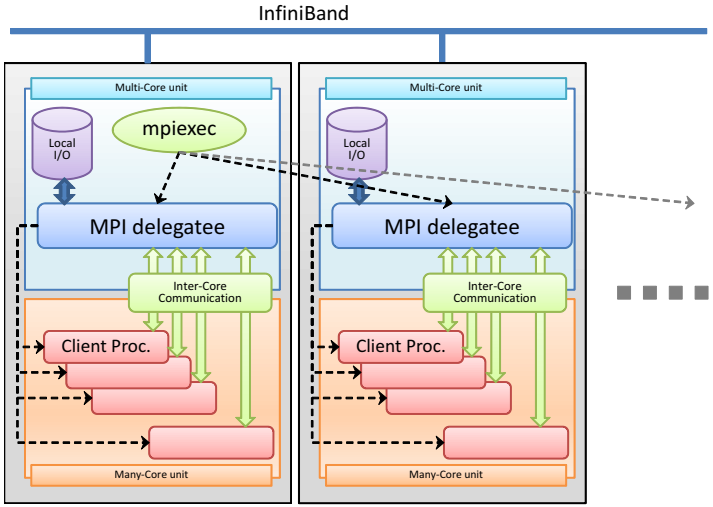

図 2 提案する MPI 基盤システムの概略図

3. 通信性能の予備評価実験

3.1 実験目的

MPI 処理を Delegatee に委託することによりオーバーヘッドが生じ，通信性能が低下することが懸念される．現時点では実装対象のハードウェアであるメニーコアプロセッサボー

Vol.2012-HPC-133 No.15
2012/3/26

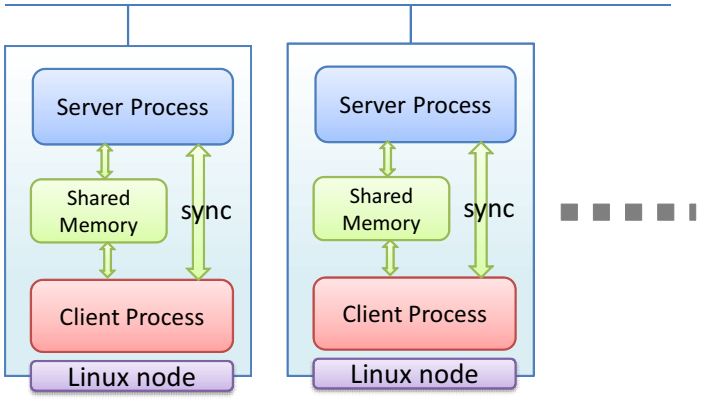

図 3 模擬評価環境の構成

ドが存在しないため，我々はこの性能への影響がどの程度かを見積もるために，汎用 Linux クラスタ上に構築した予備評価環境を利用し，性能の評価を行なっている．

前回の研究報告[6] において我々は，カーネルレベルで通信通知を行うキャラクタデバイスを用いた MPI 処理委託システムの模擬環境を作成し，MPI 処理委託によるオーバーヘッドが通信に与える影響の予備評価を行った．本稿では，ユーザレベルでの通信通知機構としてポーリングによるメモリ監視を行う機構を実装し，その性能と特性を評価し，キャラクタデバイスによる機構と比較検討する．

3.2 模擬環境の実装詳細

評価に用いた模擬環境の概略図を図 3 に示す．

模擬環境を構成する各要素の詳細は，以下のとおりである．

サーバプロセス

サーバプロセスは，提案するシステムの Delegatee に相当するものである．プログラム実行時に，利用するノード上に MPI プロセスとして起動され，クライアントプロセスからの MPI 処理依頼を受け取り処理を行う．

クライアントプロセス

クライアントプロセスは，メニーコアプロセッサ上のクライアントプロセスを模擬するものであり，サーバプロセスから起動される．クライアントプロセスではサーバプロセスを介した通信を行うために，MPI 処理依頼の発行を行う．

表 1 実験環境の仕様

PC ノード	CPU	Intel Xeon E5530 (2.4GHz)
	Memory	DDR3-10660 SDRAM 8GB
	NIC	Broadcom iNetXtreme II BCM5716
	OS	fedora 12
	MPI	MPICH2-1.4
ネットワークスイッチ	HP 2910-24G al Switch (J9245A)	

共有メモリ領域

両コア間での仮想的な共有メモリ領域を模擬するために，サーバプロセスとクライアントプロセス双方からアクセス可能な共有メモリ領域を作成した．この共有メモリ領域は，サーバプロセスへの MPI 処理依頼に用いるだけでなく，クライアントプロセスの送受信メモリ領域としても利用する．実際のシステムにおいては，メニーコアプロセッサ側に存在するクライアントプロセスの送受信データ領域をマルチコアプロセッサ側がマッピングすることで，マルチコアプロセッサとメニーコアプロセッサ間でのデータコピーを伴わない通信を実現する．この仕様を再現するために，クライアントプロセスのデータ送受信に用いる領域を予め共有メモリ領域に確保しておき，MPI 代行処理依頼を受け取ったマルチコアプロセッサはその領域を参照することで，プロセス間でのメモリコピーの発生しない環境を作成した．

同期 (通信通知) 機構

同期機構は，両プロセス間での通信を通知するためのものである．前回の研究報告において実装したキャラクタデバイスによる機構の他，今回新たに共有メモリ領域を用いたポーリング監視機構を実装した．

3.3 実 験 環 境

実験に用いた汎用 Linux クラスタ環境の仕様を**表 1** に示す．このクラスタは 9 台のノードから構成されており，各ノード間は 1000BASE-T ネットワークにより接続されている．

3.4 実 験 概 要

実験環境上で，通常の MPICH2 を利用した直接通信による性能と，模擬環境を利用した Delegatee を介した通信による性能の比較を行った．Delegatee を介した通信では，キャラクタデバイスを用いた方式と，ポーリング機構を用いた方式の二種類を利用し，計三種類の通信方式について，比較と評価を行った．

3.4.1 一対一通信

まず，クラスタ内の 2 ノード間で Ping-Pong 通信を行い，その実行時間を比較した．実

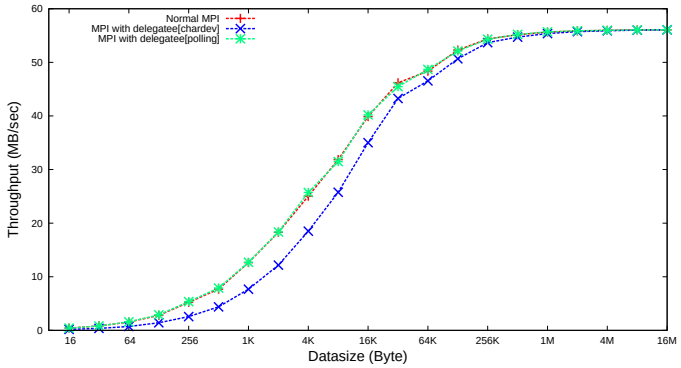

図 4 2 ノード間での一対一通信の結果

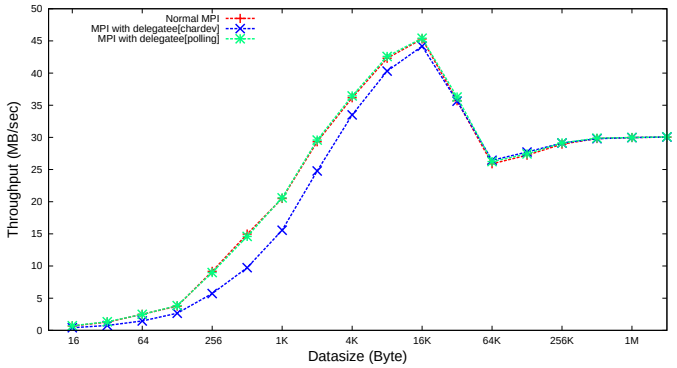

図 5 8 ノードでの Alltoall 通信の結果

行結果を**図 4** に示す．横軸は送信データサイズ，縦軸はスループットである．

キャラクタデバイスを用いた通知処理の場合，転送データサイズが小さいとき，通常の MPICH2 による通信と比較してデータサイズ 8KB の場合で 10%，256Byte 以下の場合は 40%以上の性能低下が生じている．一方でデータサイズが増加した場合は，実際の通信時間が全体の処理時間に占める割合が大きくなるため，性能低下はほぼ見られない．

ポーリング機構を用いた場合はデータサイズが小さいときもほぼ性能が低下することなく通信できている．

3.4.2 Alltoall 通信

次に，8 ノードを用いた Alltoall 通信の実行時間を比較した．その実験結果を**図 5** に示す．スループットの値は，1 つのプロセスからの全送信データサイズをデータ転送時間で割ったものである．

Alltoall 通信の場合も一対一通信と同様の傾向が出ており，キャラクタデバイスを用いた場合の性能低下は，通信データサイズが 1KB 以下のとき 15〜40%程度生じているが，16KB 以上の場合 1%以下と非常に小さくなる．ポーリング機構を用いた場合，今回の実験においては性能低下は確認できなかった．

3.5 考　　察

キャラクタデバイスを用いた処理依頼のオーバーヘッドは，今回の実験環境では 50〜100μs 程度であった．一方，ポーリング機構を用いた場合は，通常の MPICH2 による通信と比較して 1〜3μs 程度のオーバーヘッドに留まっており，キャラクタデバイスを用いた場合と比較して高性能を示した．

しかし，ポーリング機構ではメモリを監視し続ける必要があるため，非常に高いプロセッサ使用率を示しており，無駄な消費電力の増加を引き起こすとともに，他処理への影響も懸念される．一方キャラクタデバイスを用いた場合は CPU 使用率に大きな変動はなかった．エクサスケールを目指す上で消費電力の増大は無視できない問題であり，通信性能だけに着目して常にポーリングによる通信通知を行うことは非現実的である．

通信時のオーバーヘッドが性能低下に直結すると考えられる同期通信においては，ユーザレベルでのポーリング機構による通知を用いることで高速な通信を行い，非同期通信においてはオーバーヘッドが生じるが CPU 使用率の低いカーネルレベルでの通知を行うなど，通信特性などを考慮した使い分けが必要であると考えられる．

また，キャラクタデバイスを用いた場合に非常に大きく性能に影響が現れた，データサイズが小さい場合の通信については，メニーコアプロセッサから直接 InfiniBand を介した通信を行うなど，オーバーヘッドを回避した高速な通信を実現する必要がある．今回の実験では，ポーリング機構を用いた場合，転送データサイズが小さい場合も通信性能の低下は見られなかった．しかし，今回の実験環境は Gigabit Ethernet による通信を用いており，InfiniBand のような低遅延広帯域のネットワークで接続される実際の大規模並列計算機環境上では，実際のデータ転送時間が大幅に改善されるために処理依頼のオーバーヘッドの影響が大きくなる可能性があり，今後更に確認を進めていく必要があると考えている．

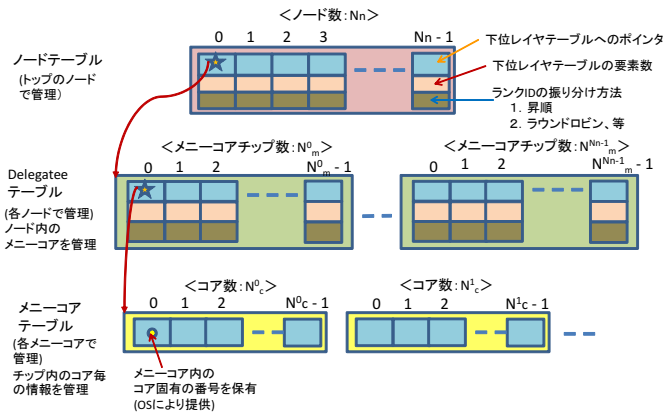

図 6 階層的なコミュニケータ情報のデータ構造

4. 階層的な MPI コミュニケータ管理手法の提案

エクサスケールの実現における MPI 通信基盤ソフトウェアが抱える問題の一つとして，コミュニケータの管理手法が挙げられる．MPICH2 に代表される現在の MPI 実装の多くは，各 MPI プロセスが所属するコミュニケータ内の全プロセスの情報を保持するため，情報量が膨大になり，エクサスケールにおいては実現が不可能である．そのため，各実装でもメモリ消費を低減させる試みが行われている[7)8)]．我々の研究においても，コミュニケータの効率的な管理手法は重要な課題の一つである．ここでは，現在検討しているコミュニケータ管理手法について説明する．

4.1 階層的な MPI コミュニケータ情報の設計

2.2 節で述べたように，メニーコアプロセッサ上のクライアントプロセスからの MPI 通信は，

(1) 同一プロセッサボード内での直接通信
(2) 同一ノード内での別プロセッサボードとの通信
(3) ノード間にまたがる通信

の三形態に大きく分類される．この利用形態を踏まえ，我々は**図 6** のようなコミュニケータ情報の管理方式を考えている．

このデータ構造はシステム上に基本データとして保有し，MPI プログラム起動毎にこの基本データの必要部分をコピーして，コミュニケータ情報を生成する．このデータは各ノードに一つずつ Delegatee を起動することを想定しており，最上位レイヤであるノードテーブルが全 Delegatee(全ノード) のテーブルへのポインタを保持し，そのポインタが指し示す下位レイヤの Delegatee テーブルの要素数も保持している．また，MPI のランク ID の振り分け方式等 (例えば，順方向やノード間のラウンドロビンなど) についての設定情報も保有することを可能としている．このテーブルは MPI プログラムを起動したノード上で管理する．Delegatee テーブルには，管理するメニーコアプロセッサ群のメニーコアテーブルへのポインタと，そのテーブルに格納されている要素数，MPI のランク ID の振り分け方式が格納される．メニーコアテーブルは各メニーコアプロセッサボード内の全コアの情報を保持し，メニーコア内部でのランク番号に対応する各要素に一意的な各コアの ID を保持する．この ID は OS によって決定されるものである．メニーコアテーブルは，メニーコアプロセッサ内の一つのコアに配置する通信マネージャによって管理し，一つのメニーコアボード上で一つのみテーブルを保持してメニーコア上の各プロセスから参照する形をとることで，メモリ使用量の削減を図る．

また，コミュニケータ形成時に各テーブルには付加情報として，自身におけるランク ID の振り分け方式と，ランク ID 振り分け時にオフセットとなる情報（メニーコアテーブルであればコミュニケータ内のメニーコアプロセッサの総数，Delegatee テーブルであればコミュニケータ内の Delegatee の総数）を保持する．この付加情報により，探索するランク ID が自身の管理下に存在するかを特定することが可能となる．

4.2 通信時の動作

このコミュニケータ管理方式を用いると，前述した通信形態のうち，(1) で示した同一プロセッサボード内の通信を行う場合，メニーコアプロセッサ内に保有するメニーコアテーブルのみで通信が管理でき，低遅延な通信が可能と考えている．他方，(2) または (3) 同一プロセッサボード外との通信を行う場合は，まずメニーコアテーブルとその付加情報から同一プロセッサボード内に通信相手のランクが存在しないことを確認する．続いて Delegatee に通信依頼を行い，Delegatee は Delegatee テーブルとその付加情報を参照することで，自身の管理するノード内に通信相手のランクが存在するかどうか確認する．存在する場合はそのプロセッサボードのメニーコアテーブルを参照して通信相手を特定し，通信を行う．存在しない場合はさらに上のレイヤであるノードテーブルを参照し，通信相手の含まれる Delegatee テーブルを特定し，最終的にメニーコアテーブルまで辿り通信を行う．

以上により，各プロセスがプロセス全体の情報を管理する方法よりも大きく情報量を減ら

すことができ，かつ効率的な参照も可能になると考えている．

5. 関連研究

並列環境のハードウェアトポロジを考慮した MPI 通信を考案した研究として，Rashti らの論文[9] がある．この研究では，hwloc(Hardware Locality) と言うソフトウェアライブラリの機能を MVAPICH2 内部に組み込むことで，各ノードを構成する CPU とそのコア，キャッシュ，メモリなどの情報を得ると共に，InfiniBand によるネットワーク接続構成を ibtracert ユーティリティを用いて確認し，並列環境全体のハードウェアトポロジを把握している．そしてその上で最適な通信トポロジを形成することにより，高性能な MPI 通信を実現している．この研究は一般的なマルチコアプロセッサのみを用いたクラスタ環境を対象としているものであり，ヘテロジニアスなメニーコア混在型並列計算機環境を考慮した場合，本稿で提案するようにノード内の通信やプロセッサアーキテクチャの差異も考慮した通信を設計する必要がある．

6. おわりに

本稿では，メニーコア混在型並列計算機において，高スケーラブルで高効率な MPI 通信を実現する，基盤システムソフトウェアについて述べた．今回新たに評価したポーリング機構を用いた通信通知方式は，キャラクタデバイスを用いた通信の通知方式と比較して高性能であるが，処理に CPU を多く消費するため，システムではそれぞれの特徴を考慮し適宜使い分ける必要がある．また，MPI コミュニケータの効率的な管理を行うために，階層的なコミュニケータの管理手法について提案を行った．

今後は，より高速なネットワークを用いた MPI 処理依頼のオーバーヘッドの影響の確認や，コミュニケータの管理手法について更なる検討を進める．

謝辞 本研究は，科学技術振興機構（JST）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」研究課題「メニーコア混在型並列計算機用基盤ソフトウェア」によるものである．

参考文献

1) Intel: Intel®Many Integrated Core Architecture, Intel Corp. (online), available from ⟨http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html⟩ (accessed 2012-2-27).
2) Howard, J., Dighe, S., Hoskote, Y., Vangal, S., Finan, D., Ruhl, G., Jenkins, D., Wilson, H., Borkar, N., Schrom, G. and et al.: A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS, *2010 IEEE International SolidState Circuits Conference ISSCC*, pp.108–109 (2010).
3) Intel: Intel Unveils New Product Plans for High-Performance Computing, Intel Corp. (online), available from ⟨http://www.intel.com/pressroom/archive/releases/2010/20100531comp.htm⟩ (accessed 2012-2-27).
4) TACC: “Stampede’s” Comprehensive Capabilities to Bolster U.S. Open Science Computational Resources, Texas Advanced Computing Center (online), available from ⟨http://www.tacc.utexas.edu/news/press-releases/2011/stampede⟩ (accessed 2012-2-27).
5) 佐藤未来子，辻田祐一，堀　敦史，並木美太郎：マルチコア・メニーコア混在型並列計算機向け OS の構想，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-OS-118, No.6, pp.1–6 (2011).
6) 吉永一美，六車英峰，辻田祐一，堀　敦史，並木美太郎，佐藤未来子，下沢　拓，澤田武男，石川　裕：メニーコア混在型並列計算機における MPI 通信基盤の提案，情報処理学会研究報告 [ハイパフォーマンスコンピューティング (HPC)]， Vol.2011-HPC-132, No.5, pp.1–6 (2011).
7) Goodell, D., Gropp, W., Zhao, X. and Thakur, R.: Scalable Memory Use in MPI: A Case Study with MPICH2., *EuroMPI’11*, pp.140–149 (2011).
8) Balaji, P., Buntinas, D., Goodell, D., Gropp, W., Hoefler, T., Kumar, S., Lusk, E., Thakur, R. and Traeff, J.L.: MPI on Millions of Cores, *Parallel Processing Letters*, Vol.21, No.1, pp.45–60 (2011).
9) Rashti, M. J., Green, J., Balaji, P., Afsahi, A. and Gropp, W.: Multi-core and Network Aware MPI Topology Functions., *EuroMPI*, Lecture Notes in Computer Science, Vol.6960, Springer, pp.50–60 (2011).