Vol.2015-HPC-150 No.17
2015/8/5

メニーコアプロセッサにおけるPVASタスクモデルによるMapReduceアプリケーションの性能評価

佐藤 未来子[†1] 島田 明男[†2] 吉永 一美[†3] 辻田 祐一[†3]
堀 敦史[†3] 並木 美太郎[†1]

本研究では，Intel XeonPhiを搭載するヘテロジニアスアーキテクチャシステムを対象に，CPUごとの異なる演算性能を有効活用するためのプログラム実行基盤Multiple PVAS（以下，M-PVAS）を研究開発している．M-PVASで実現しているグローバル仮想アドレス空間の有効性の検証を兼ねて，M-PVASの仮想アドレス空間共有モデルを用いたMapReduceフレームワークの実現方式を検討している．MapReduceフレームワークをM-PVAS上で実現する場合に生じる利点や問題点を知見として得るために，XeonPhi上で開発されている既存のメニーコア向けMapReduceフレームワークをM-PVASのPVASタスクモデルで試作した．本稿では，XeonPhi上で実現したM-PVASにおけるMapReduceアプリケーションの実現方式と簡易評価の結果について述べる．

1. はじめに

近年，アプリケーションプログラムの高速実行のためにシステムに搭載するプロセッサ数，コア数を増やし並列性能を高めることで性能向上を図る傾向にある．このような背景から，GPGPUやIntel XeonPhiなどのメニーコアアクセラレータを搭載するヘテロジニアスアーキテクチャシステムが広く利用されるようになった．メニーコアアクセラレータを有効に活用する際には，GPGPUであればCUDAに代表される様なアクセラレータに特化した並列プログラミングにより，計算性能を最大限に引き出すための並列アプリケーションの開発が必要となる．

このような背景から，近年ではビッグデータの処理基盤としてMapReduceの分散処理フレームワークが注目されている[1]．MapReduceはプログラマが定義したMapとReduceと呼ばれる計算を並列分散処理させるためのフレームワークである．近年では分散環境におけるHadoop[2]の他，京コンピュータにおけるKMR[3]，GPGPUやXeonPhiなどのメニーコア向けMapReduceフレームワーク[4][5][6][7]など，各種のプラットフォームでのMapReduce実現方式が提案されている．これらのMapReduceのフレームワークは，OSやランタイムライブラリが提供する並列実行基盤を応用して実現されている．したがって，並列プログラミングに長けていないプログラマであっても，並列化したい処理をMapとReduceで構築することでシステムに備わる並列分散処理の機能を簡単に利用できるという点がMapReduceフレームワークの特徴である．

本研究では，Intel XeonPhiを搭載するヘテロジニアスアーキテクチャシステムを対象に，CPUごとの異なる演算性能を有効活用するためのプログラム実行基盤Multiple PVAS（以下，M-PVAS）を研究開発している[8]．M-PVASでは，異なるCPU上のタスク同士が少ないオーバヘッドで連携できるように，ヘテロジニアスシステム上のグローバル仮想アドレス空間を提供し，異なるCPU上で実行するタスク同士が仮想アドレスを用いてお互いのアドレス空間をアクセスできる機能を実現している．お互いの仮想アドレス空間に対してダイレクトにアクセス可能なので，煩雑な通信プロトコルを利用することなく，データや制御の送受信が可能である．本研究では，M-PVASの実行基盤の検証を兼ねて，MapReduceフレームワークの実現方式を検討している．MapReduceフレームワークをM-PVASのグローバル仮想アドレス空間上で実現する場合に生じる利点や問題点を知見として得るために，XeonPhi上で開発されているメニーコア向けMapReduceアプリケーションをM-PVAS向けに試作した．本稿では，XeonPhi上で実現したM-PVAS実行基盤におけるMapReduceアプリケーションの実現方式と評価から得られた知見について述べる．

2. 関連研究

MapReduceは並列分散処理のためのフレームワークとして様々な並列計算機環境で実現されている．本研究のように，メニーコアによるアクセラレーションを適用したMapReduceフレームワークもいくつか提案されている．phoenix++[6]やMRPhi[7]は汎用CPUを対象としたMapReduceフレームワークであり，Intel XeonPhi上での並列処理も可能である．スレッドベースでmaster/workerモデルでMap処理，Reduce処理を並列化する．Mars[4]はGPGPUへ並列計算をオフロードするためのMapReduceフレームワークを提案しており，近年では複数のGPGPUを対象とする計算オフロード方式も提案されている[5]．

本研究で対象としているIntel XeonPhiでは，MRPhi [7]においてマルチコアCPU向けに提案されたPhoenix++ [6]をXeonPhi向けにMap処理のベクトル化，SIMD命令によるハッシュ管理などの最適化が提案されている．これによりXeonPhi上でのMap処理，Reduce処理の高速化が実現されている．そしてMrPhiにおいて，MPIによる分散オペレーションによりホストから複数台のXeonPhi上のMapReduce処理

†1 東京農工大学
†2 (株)日立製作所
†3 理科学研究所 AICS

を稼働させ，データを分散させる方式が提案されている[9]. MapReduce処理の並列化に関しては既存の汎用OSで提供されるPOSIX ThreadやMPIといった並列実行基盤を応用するアプローチである．

3. 目標

本研究の目的は，MapReduce フレームワークにおいて M-PVAS のプログラム実行基盤の有効性を検証することである．従来の MapReduce フレームワークの研究で利用される，スレッド等の並列実行基盤の機能を M-PVAS の空間共有を活用した通信制御に置き換え，知見を明らかにすることが本研究の目標となる．本研究ではその第一ステップとして，既存の MapReduce フレームワークである MRPhi の master/worker モデル制御を，M-PVAS 実行基盤において構築し，M-PVAS の特徴であるアドレス空間共有を活用した場合の MapReduce アプリケーションにおけるデータ転送や制御の性能を評価する．

4. M-PVAS

M-PVAS は，マルチコア CPU とメニーコア CPU 上の個々のタスクが同一の仮想アドレス空間上で動作するプログラム実行基盤である．図 1 にシステム構成，図 2 に M-PVAS で管理する仮想アドレス空間の概念図を示す．図 1 に示すように，M-PVAS では各 CPU 上で OS が資源を管理し，OS 間でメモリ管理情報を交換しあうことで，“M-PVAS Address Space”という単一の大域的な仮想アドレス空間を形成する．M-PVAS における各 OS は，PVAS（Partitioned Virtual Address Space）のアドレス空間を管理する．PVAS においてコアを割り当てる実行実体であるプロセスを“PVAS Task”と呼び，図 2 中央の“PVAS Address Space”に示すとおり，一つの仮想アドレス空間上に，個別の PVAS Task のアドレス空間“PVAS Partition”を配置する．これにより，PVAS Task 間では，従来の汎用 OS における共有メモリを要すること無く，仮想アドレス参照により Task 間でデータを共有することができる．M-PVAS では，どの CPU からもこの大域的な仮想アドレス空間をアクセスできる．また，M-PVAS により，異なる PVAS 空間に属する PVAS Task であっても，仮想アドレスを用いた直接メモリアクセスができるので，複数 CPU 間でメモリベースのデータ転送や制御を行える．

5. M-PVAS による MapReduce フレームワーク

本研究では，XeonPhi 向けの最適化が施された既存の MapReduce フレームワークである MRPhi を M-PVAS のプログラム実行基盤上に構成する．XeonPhi 向けに施された最適化をそのまま活用して MPVAS 上へ適用させることにより，並列実行制御やデータ制御にかかるオーバヘッドの違いを明らかにできると考えた．

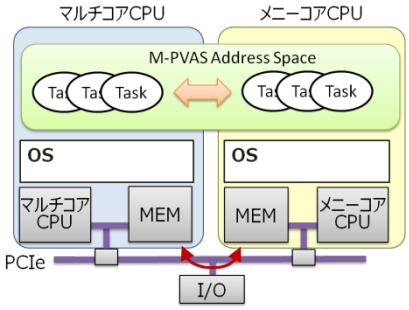

図 1 M-PVAS のシステム構成

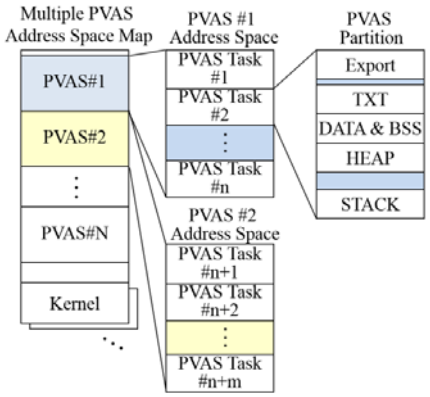

図 2 M-PVAS の仮想アドレス空間の概念図

5.1 MRPhi の MapReduce フレームワーク構成と課題

図 3 に MRPhi における MapReduce フレームワークのタスク構成を示す. MRPhi では一つのプロセスが Master となり，Master が生成した Worker スレッドに対して，Map 処理，Reduce 処理を依頼して並列実行する構造となっている．MRPhi のフレームワーク自体はスレッドモデルで構成されているため，Master と Worker スレッド間ではデータ共有可能である. 実際にフレームワーク内部では, MapReduce の処理対象データを Master で管理し，稼働する Worker の個数で分割し, 担当領域の先頭アドレスのみを Worker スレッドへ通知することで，フレームワーク内部でのメモリコピーを抑制し，各 Worker スレッドによる Map 処理を実行できる構成としている．

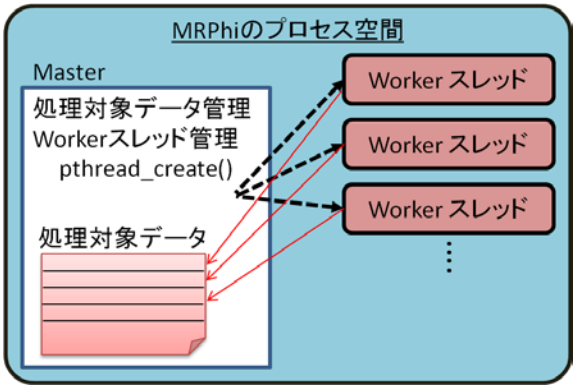

図 3 MRPhi における MapReduce フレームワークの構成

このように，MRPhi のメニーコア CPU 向け MapReduce フレームワークでは，スレッド並列による並列化が図られている．これにより，データ共有を図り，処理対象データのタスク間コピーを避ける設計となっている．しかし，Worker スレッドを用いた並列化は各 CPU で OS が管理するコアに限定されるという制約が発生する．また，MRPhi の処理対象データは，各 CPU 上の OS が管理するファイルやメモリ上のデータである．XeonPhi の場合，HDD 等の I/O をサポートしていないため，あらかじめ RAM ファイルとして転送しておくなど,データコピーしておくこととなる．しかし，XeonPhi のメモリ容量の限界から一度に大量の処理対象データを扱うことはできない．

また，MRPhi を用いた予備評価の結果を図 4 に示す．MRPhi で実行可能なサンプルアプリケーション[7]を Xeon および XeonPhi で実行させ，Xeon における MapReduce 処理時間で正規化した結果である．本予備評価では，XeonPhi で高速化できるものと高速化しないものがあることが判明した．この結果の要因は主に，アプリケーションの Map 処理において XeonPhi で得意とする並列演算処理を多く含むものと，テキスト処理のようなデータ処理であるかの違いであった．この結果から，アプリケーションの特質に応じて Xeon 上の Worker と XeonPhi 上の Worker を使い分けることも MapReduce の性能向上に有効であり，MapReduce フレームワークにおける課題となる．

5.2 M-PVAS の MapReduce フレームワークの設計方針

既存 OS 上で CPU を超えた分散並列を図る場合には，MPI 等の通信インタフェースを活用したプロセス制御とデータ通信が必要となる．4 章において示したとおり，M-PVAS の特徴は，異なる CPU 上に構成できる単一の大域仮想アドレス空間上の PVAS Task 同士が，仮想アドレスを用いたメモリベースの通信により CPU を超えた制御を行えることである．そこで，M-PVAS では，次の二つの方針で MapReduce フレームワークを設計している．

(1) M-PVAS 空間上の Master/Worker Task

M-PVAS により，Xeon・XeonPhi の両 CPU を活用した MapReduce 処理の高速化を図る．ホストである Xeon 上にはデータ転送や Worker を管理するための Master を，Xeon と XeonPhi 上には Worker を同一の M-PVAS 空間上に稼働させる．Master は，Worker の処理性能とデータ転送性能を見極めながら MapReduce 処理を Worker へ分散させる．その際，Master と Worker 間，および，Worker 同士のデータ共有により，コピーレスなデータ転送やタスク制御による性能向上を図る．

(2) XeonPhi での大容量データ処理への対応

Xeon のファイル I/O を XeonPhi の Task から活用する機能を MapReduce のフレームワーク内に備える．これにより，XeonPhi 上では扱えないファイルベースの大容量データを，XeonPhi 上の Worker において処理できるようにする．

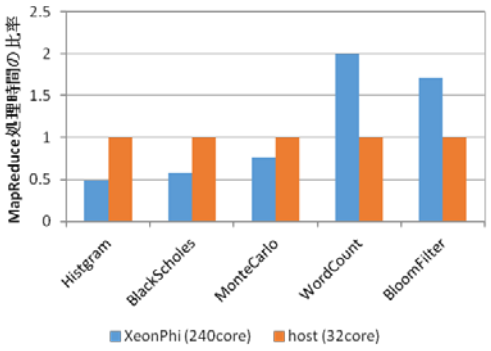

図 4　MRPhi を用いた予備評価の結果

5.3 M-PVAS の MapReduce フレームワークのタスク構成

本 MapReduce フレームワークの設計方針に基づき，MapReduce フレームワークのタスク構成を検討している．M-PVAS 空間には自由に PVAS Task を配置できるため，各種の構成が考えられる．そのため，MapReduce フレームワーク向けにメモリ共有による Task 間通信の効率を考えて Master/ Worker Task の配置を検討した．

図 5 と図 6 に M-PVAS の MapReduce フレームワークのタスク構成を示す．各 PVAS 空間には，MapReduce 処理を行うための Worker Task を生成し，Worker Task が行う MapReduce 処理の対象となるデータを保持するためのメモリは，それぞれの PVAS 空間上に確保する．ヘテロジニアスアーキテクチャでは, 別 CPU のメモリと自身のメモリとのデータアクセスコストに差があるため，MapReduce 処理を効率よく実行させるために，処理対象データをあらかじめ同一 PVAS 空間に配置しておくことにした．

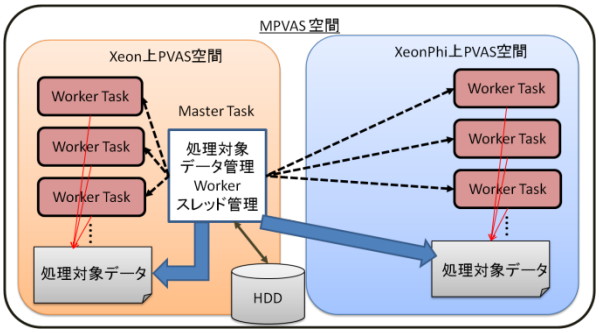

図 5　M-PVAS MapReduce フレームワークの構成(1)

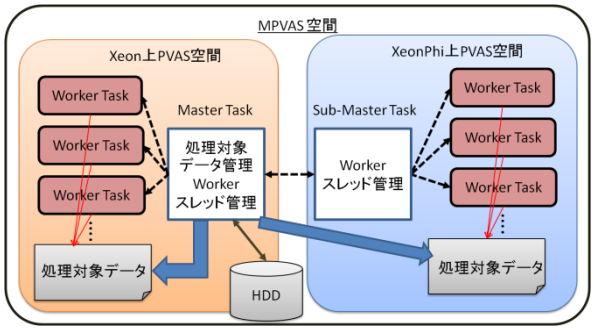

図 6　M-PVAS MapReduce フレームワークの構成(2)

図 5 と図 6 の構成の違いは，Master Task の構成である．図 5 の場合は，Xeon 上 PVAS 空間に生成した Master Task がすべての Worker Task の制御と処理対象データの管理を担うモデルである．図 6 の場合は，Xeon と XeonPhi の両方に Master Task を設け，ホスト上の Master Task を中心に CPU ごとに MapReduce の処理を分割し，各 CPU 上の Sub-Master Task が Worker Task へ MapReduce 処理を分散させるモデルである．Mater Task の Worker Task 管理を分散させ，Worker Task の同期管理などの制御を CPU ごとに行うことで制御効率を向上できると考えた．図 7 に示すとおり，M-PVAS における CPU 間制御通信を模擬した小サイズのデータ通信性能は，2〜12 μ 秒を要する．図 5 のモデルの場合，Worker Task の Map 処理，Reduce 処理に必要なデータアドレス，データサイズなどの引数を通知する場合に，数 μ 秒のオーバヘッドが発生することになる．一方で，図 6 のモデルでは，Sub-Master Task のためにコアを消費することになり，Worker Task を減らす必要がある．したがって，これらのモデルの選択は，MapReduce アプリケーションで並列化する処理によるところが大きい．よって，いくつかのアプリケーションでこれらのモデルを実装し，性能評価を進めていく．現在は，両方のモデルでの試作を進めている．

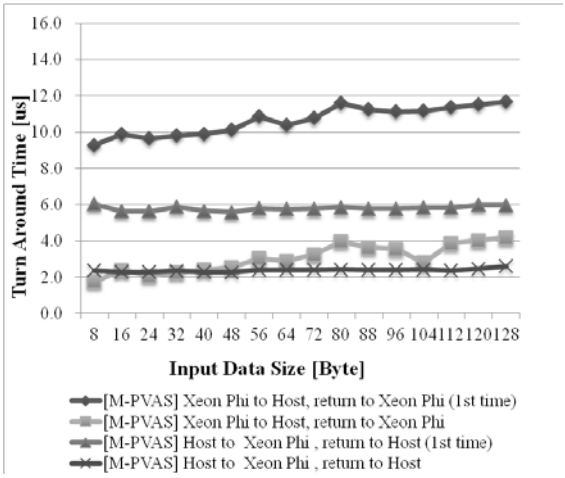

図 7　M-PVAS における Xeon・XeonPhi 間データ通信性能

6. XeonPhi 上での MapReduce フレームワークの試作

本研究では，まず POSIX Thread ベースのフレームワークと PVAS のアドレス空間共有を活用したフレームワークとを比較するために，既存の MapReduce フレームワークである MRPhi の master/worker モデル制御を M-PVAS 実行基盤において試作した．表 1 にヘテロジニアスアーキテクチャの計算機環境の仕様を示す．MapReduce フレームワークの構成は，図 6 の各 CPU に Master Task を備えるモデルに準じて，各 PVAS 空間で MapReduce 処理を行う構成とした．現状の MRPhi においても，5.1 節で述べたとおり Master と Worker スレッドを同一 CPU 上で実行するモデルとなっているため，最も近いモデルで性能比較する方針とした．なお，処理対象データは，あらかじめ MapReduce を実行する CPU のメモリへあらかじめ転送しておき実行した．まずは Task 制御の違いについてのみ評価した．

表 1　ヘテロジニアスアーキテクチャの仕様

Many-core	CPU	Intel Xeon Phi 5110P (60 cores, 240 threads, 1.053GHz)
	Memory	GDDR5 8GB
	OS	Linux 2.6.38
Multi-core	CPU	Intel Xeon E5-2650 x2 (8 cores, 16 threads, 2.6GHz)
	Memory	DDR3 64GB
	OS	Linux 2.6.32 (CentOS 6.3)
Intel CCL	MPSS	Version 3.4.3

6.1 M-PVAS における試作

図 8 に，試作した M-PVAS の MapReduce フレームワークの概念図と使用した M-PVAS 関数を示す．各関数仕様の詳細に関しては[10]を参考にされたい．

まず M-PVAS では，Task を生成する前処理として，M-PVAS 空間と，Xeon 上と XeonPhi 上での PVAS 空間を生成する（mpvas_create 関数）．その後，その M-PVAS 空間と PVAS 空間を ID 指定することで，所望の空間上に PVAS Task を生成する（mpvas_spawn 関数）．本 MapReduce フレームワークでは，Master Task を生成すると同時に，あらかじめ Master Task を実行するコア以外のすべてのコア上に Worker Task を生成しておく．Master Task に対して，Map 処理と Reduce 処理に用いる Worker Task 数を指定することで，アプリケーションプログラマが定義した Map 処理，Reduce 処理を所望の並列度で実行する．

現状では，ファイル I/O を用いた処理対象データ管理を実装していないため，あらかじめ処理対象データをメモリ上に保持した状態で MapReduce アプリケーションが実行される．Master は，処理対象データの仮想アドレスを管理し，各 Worker Task に対して，処理対象データの仮想アドレスと担当するデータサイズを通知する．この PVAS Task 間のデータ通知に用いる共有領域として，PVAS では export segment が用意されている（図 2 参照）．本 MapReduce フレームワークでは，export segment を Master で生成し（mpvas_

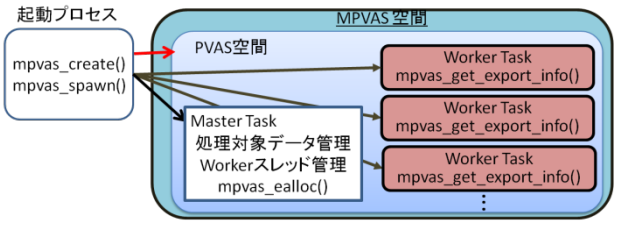

図 8　M-PVAS で試作した MapReduce フレームワークのタスク構成と M-PVAS 関連の API

ealloc 関数), 各 Worker Task が本 export segment の仮想アドレスを得る（mpvas_get_export_info 関数）ことで，Master と Worker 間のデータや制御通知をメモリ共有により行っている．

6.2 Monte Carlo アプリケーションによる比較

本研究で試作した MapReduce フレームワークにおいて，Monte Carlo シミュレーションのサンプルアプリケーションを用いた実行性能比較を行った．現状では，Xeon 単体と XeonPhi 単体のそれぞれで実行した結果の比較となっている．MRPhi の内部構造には手を加えずに，文献[7]の最適化を流用して，6.1 節に示した構成で PVAS Task 化を図った場合，元の MRPhi の実装（original Monte Carlo）に比べて PVAS 化の実装（PVAS 化 Monte Carlo）における Xeon では 6.8%の実行時間増加があったものの，XeonPhi では 2.4%と若干ではあるが実行時間の短縮が図れ，ほぼ同等の性能で実行できるという結果を得た．2.4%の実行時間削減の要因については，本 PVAS 化を行う際に Map 処理内で使うテンポラリのバッファを Worker Task で保持する必要があり，この若干の変更がデータアクセスの効率化につながったと考える．

Monte Carlo による評価だけでは，多くの知見は得られないが，POSIX Thread での実装から M-VPAS 実行基盤の実装へ移行することによる性能低下は，メニーコア上では起きなかったことから，今後も積極的に MapReduce フレームワークの M-PVAS 化をすすめる．M-PVAS の検証のためには，まずは 5.3 節に示した検討中の Worker Task 制御を M-PVAS のデータ共有を生かしながら実施し，比較評価により M-PVAS に最適な MapReduce フレームワークの実現方式を提案する．また，ファイル I/O を含む処理対象データの管理方式を検討する必要がある．

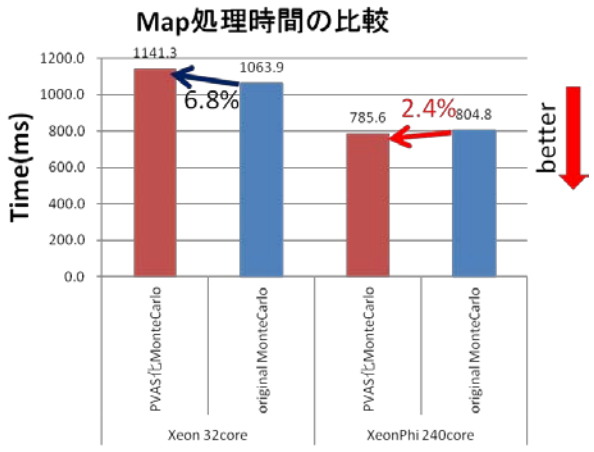

図 9　MonteCarlo アプリケーションでの比較評価

7. おわりに

本研究では，MapReduce フレームワークにおいて M-PVAS のプログラム実行基盤の有効性を検証するために，既存の MapReduce フレームワークである MRPhi の master/worker モデル制御を，M-PVAS 実行基盤において構築し，M-PVAS の特徴であるアドレス空間共有を活用した場合の MapReduce アプリケーションにおけるデータ転送や制御を試作した．元の MRPhi の実装に比べて PVAS 化の実装において，Xeon では 6.8%の実行時間増加があったものの，XeonPhi では 2.4%と若干ではあるが実行時間の短縮が図れ，ほぼ同等の性能で実行できるという結果を得た．今後は，評価対象の MapReduce アプリケーションを増やし，Worker Task の制御方式やファイル I/O を含む処理対象データの CPU 間転送方式の検討を続け，MapReduce フレームワークを対象とした M-PVAS のプログラム実行基盤の有効性を検証していく．

謝辞　本研究は，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．

参考文献

1) J. Dean and S. Ghemawat: Mapreduce: Simplified data processing on large clusters, in OSDI, 2004.
2) Welcome to Apache Hadoop (online), available from http://hadoop.apache.org.
3) M. Matsuda, N. Maruyama, and S. Takizawa: K MapReduce: A scalable tool for data-processing and search/ensemble applications on large-scale supercomputers, in CLUSTER , IEEE Computer Society, pp. 1-8, 2013.
4) B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wnag: Mars: a mapreduce framework on graphics processors, in PACT, pp. 1-8, 2008.
5) J. A. Stuart and J. D. Owens: Multi-gpu mapreduce on gpu clusters, in IPDPS, pp. 1068-1079, 2011.
6) J. Talbot, R. M. Yoo, and C. Kozyrakis: Phoenix++: modular mapreduce for shared-memory systems, in Proc. of the second international workshop on MapReduce and its applications, pp.9-16, 2011.
7) M. Lu, L. Zhang, H. P. Huynh, Z. Ong, Y. Liang, B. He, R.S.M. Goh, and R. Huynh: Optimizing the MapReduce framework on Intel Xeon Phi coprocessor, IEEE International Conference on Big Data, pp.125-130, 2013.
8) M. Sato, G. Fukazawa, A. Shimada, A. Hori, Y. Ishikawa, and M. Namiki: Design of Multiple PVAS on InfiniBand Cluster System Consisting of Many-core and Multi-core, in EuroMPI/ASIA '14, pp.133-138, 2014 .
9) Lu, M., Liang, Y., Huynh, H., Liang, O., He, B., and Goh, R.: MrPhi: An Optimized MapReduce Framework on Intel Xeon Phi Coprocessors, IEEE Transactions on Parallel and Distributed Systems, vol.PP, no.99, pp.1-14, 2014.
10) M-PVAS Function Documentation (online), available from http://www-sys-aics.riken.jp/releasedsoftware/software/d_pvas/group__mpvas.html