VII. 高性能計算システム研究部門

1. メンバー

教授　　朴　泰祐、高橋　大介、建部　修見
准教授　　塙　敏博（客員、東京大学）
助教　　多田野　寛人、小林　諒平
フェロー　佐藤　三久
研究員　　藤田　典久
学生　　大学院生　22 名、学類生　6 名
学内共同研究員
安永　守利、和田　耕一、櫻井　鉄也、山口　佳樹、今倉　暁
（以上、システム情報系）
学外共同研究員
小柳　義夫（RIST）、石川　裕（理化学研究所）、
松岡　聡（理化学研究所）、中島　浩（京都大学）、
天野　英晴（慶應義塾大学）、後藤　仁志（豊橋技術科学大学）、
関口　智嗣（産業技術総合研究所）、中尾　昌広（理化学研究所）、
佐野　健太郎（理化学研究所）、川島　英之（慶應義塾大学）、
田中　昌宏（慶應義塾大学）、平賀　弘平（慶應義塾大学）

2. 概要

本研究部門では、高性能計算システムアーキテクチャ、並列プログラミング環境、GPU 利用技術、FPGA 利用技術、並列数値処理の高速化研究、分散システムソフトウェア、エクストリームビッグデータの基盤技術等の研究を行っている。

3. 研究成果

【1】　OpenCL から制御可能な GPU-FPGA 間 DMA 転送技術の開発（朴、小林）

CPU-GPU クラスタ構成である現在の HPC システムの性能を更に向上させるために、GPU に演算通信性能に優れている FPGA (Field Programmable Gate Array)を連携させ、双方を相補的に利用する GPU-FPGA 複合システムに関する研究を進めている。GPU、FPGA といった異なるハードウェアを搭載するシステム上では、各デバイスで実行される演算をどのようにプログラミングし、全デバイスを協調動作させるかが重要な課題となる。そこで本研究では、GPU-

FPGA 間連携をシームレスに行うためのデバイス間メモリ転送について着目し、その要素技術を提案している。

昨年度には、PCIe プロトコルをベースとする GPU-FPGA 間 DMA 転送技術を開発し、この技術により、CPU を介さない DMA 転送を FPGA が自律的に起動することが可能となった。ただしこれは、Verilog HDL のレベルで GPU-FPGA 間で DMA 転送が実現できるのを確認したのみであり、C 言語のようなユーザーフレンドリーなプログラミング言語からこの機能を制御することができなかった。この問題を、これまでの研究から得られた知見と Intel が提供する高位合成ツール Intel FPGA SDK for OpenCL の機能とを活用することによって解決した。その成果を図 1 に示す。

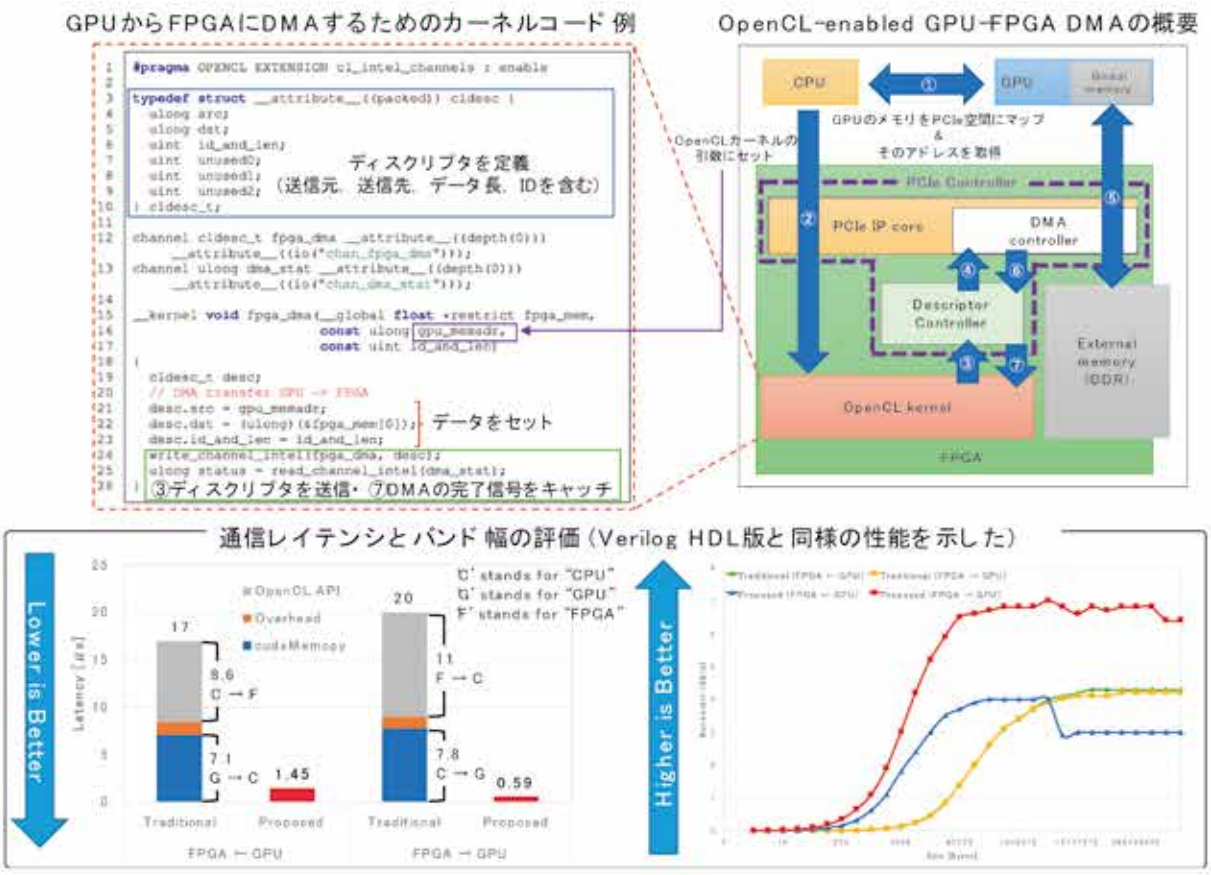


図 1 OpenCL から制御可能な GPU-FPGA 間 DMA 転送の実現

この技術により、GPU-FPGA 間 DMA 転送を OpenCL のレベルで制御できるようになり、通信レイテンシと通信バンド幅を評価したところ Verilog HDL 版と同様の性能を示すことを確認した。図から明らかなように、FPGA から GPU への通信は常に従来手法より性能が高く、最大で 7.0 GB/s (理論性能の 87.5%) のバンド幅が確認された。一方、GPU から FPGA への通信は、4 MB データサイズを超えると従来手法の方が性能が良い結果となった。この理由について NVIDIA の技術者と議論したところ、GPU の L2 キャッシュオーバーフローによるもの

だと考えられており、現在詳細に解析中である。そして、特にデータサイズが小さくなる細粒度並列処理を行う場合、低レイテンシ通信を実現できる提案手法の方が有利であることが確認された。

【2】 GPU-FPGA 連携のためのプログラム開発手法（朴、小林）

前項に示した通り、我々は GPU に演算通信性能に優れている FPGA (Field Programmable Gate Array)を連携させ、双方を相補的に利用するために GPU-FPGA 間でデータを送受信するための仕組みを開発している。GPU と FPGA をソフトウェア的に連携させるためには、ハードウェア及び周辺システムの開発だけでなく、GPU および FPGA を同一プログラムから呼び出し、提案した DMA 転送技術を活用しながら協調計算を行うためのプログラム環境が必要となる。そのため本研究では、GPU と FPGA のコンパイル環境や開発言語の組み合わせを調査し、GPU-FPGA 協調計算を行うための基本モデルを提案した。図 2 にその概要および成果を示す。

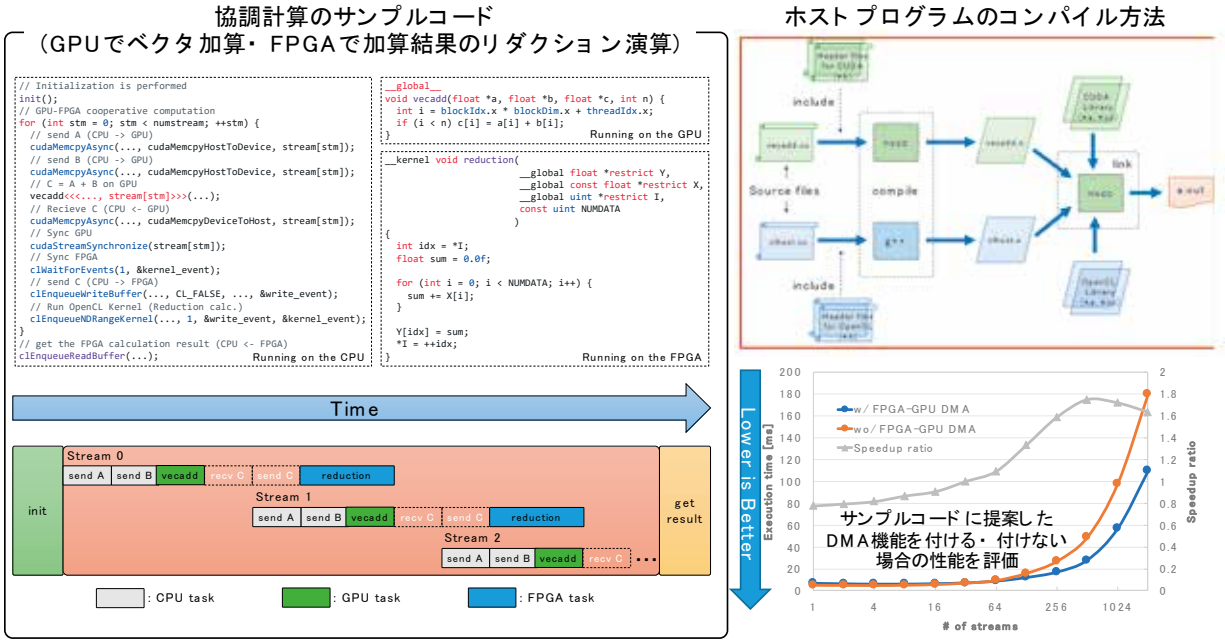


図 2 GPU-FPGA 協調計算するための基本モデルの提案

GPU プログラムは CUDA、FPGA プログラムは OpenCL で記述される。図に示すように、CUDA コードと OpenCL ホストコードは、nvcc と g++ を用いて分割コンパイルされ、生成されたオブジェクトファイルは、nvcc でリンクされることによって、実行ファイル (図の a.out) が生成される。協調計算するための例として、GPU でベクタ加算、FPGA で加算結果のリダクション演算を同時かつ非同期的に実行するサンプルコードを実装し、これを上記のコンパイル方法でコンパイルして実行させたところ、意図した通りの結果を示すことが確認さ

れた。また、このサンプルに前項目で述べた DMA 転送技術を付加した場合と付加しない場合の実行時間を比較し、通信が支配的な状況下では DMA 転送技術を付加した場合の方が優れていることも確認出来た。ただし、このような複数のプログラミング言語を用いたマルチリンガルプログラミングは、ユーザーに多大な負担を強いるので、今後の研究では GPU と FPGA が搭載された計算機システム上にて、両アクセラレータの統合的な制御を可能にするプログラミング環境について検討する。

現在、この手法の一つとして、米国 Oak Ridge National Laboratorry の Jeff Vetter 博士の研究グループとの共同研究を進め、同グループが開発中の FPGA をターゲットとした OpenACC コンパイラである OpenARC を用い、演算加速コードの評価及び我々の環境への実装を行なっている。ここでは、OpenACC を FPGA 及び GPU の共通プログラミング言語として位置付け、それぞれのデバイス用に書かれた OpenACC 化されたコード（カーネルコードに相当）を、GPU 用部分と FPGA 用部分に分割、前者を従来の PGI 製 OpenACC コンパイラで、後者を OpenARC でそれぞれ部分コンパイルし、生成されたオブジェクトを PGI コンパイラのホストコードと連結することで、OpenACC のみで両デバイスの演算加速プログラムを完結するプログラミング環境の研究を進めている。

現時点では両デバイス向けに OpenACC 化された部分を分離するメタコンパイラが開発中であるため、システムとして完成していない。メタコンパイラ部分は理化学研究所 R-CCS の並列プログラミング言語開発部門との共同研究で開発している Omni Compiler の派生である XcalableACC Compiler を流用して実装する予定である。このメタコンパイラが生成するコードを想定し、これを本項で述べた GPU と FPGA を両立させる実行環境に適用したところ、OpenACC のみのコードを正しく両デバイスで分担実行することが可能であることが実証された。次年度の研究では、OpenACC メタコンパイラを実装し、OpenACC による GPU+FPGA プログラミング環境の実現を目指す。

【3】 OpenCL による FPGA 上の演算と通信を融合した並列処理システムの開発（朴、小林）

最新の FPGA は最大で 100Gbps×4 の通信性能を有し、またそれらの通信機構は直接 FPGA に接続されているため、オーバーヘッドの少ない FPGA 間通信を可能とすることがわかっている。FPGA を単体のアクセラレータとして見ると、他のアクセラレータに絶対的な浮動小数点性能で優ることは難しい。しかしながら、FPGA が持つ演算能力と前述した強力な通信能力を組み合わせることで、より広いアプリケーション及び並列実行環境に FPGA を適用できると考えられる。本研究の目的は、高位合成で記述された FPGA アプリケーションから通信機構を操作し並列処理システムを実現することである。本研究では高位合成の開発環境として Intel FPGA SDK for OpenCL を用いる。本 SDK は OpenCL 言語に対していくつかの FPGA

に特化した拡張を加えており、その拡張の 1 つに「Channel」機構がある。Channel は OpenCL カーネル間でデータを直接交換するパイプを構築できる機能である。また、Channel の一つとして「I/O Channel」があり、これを用いることで OpenCL カーネルと外部ペリフェラルのデータ交換が可能となる。本研究では、OpenCL 開発環境に通信機構を操作するためのコントローラ (本研究では 40Gbit Ethernet) を追加し、I/O Channel を通じて OpenCL コードから通信を行える Channel over Ethernet (CoE) システムの開発を行った。

図 3 に Intel Arria10 FPGA を用いて行った CoE システムの結果を示す。この性能測定は、2 つの異なるノードにある FPGA を Ethernet スイッチ経由で接続して測定したものであり、スループットは 29.77Gbos (理論値のおよそ 75%)、最小レイテンシは 950ns という結果が得られた。

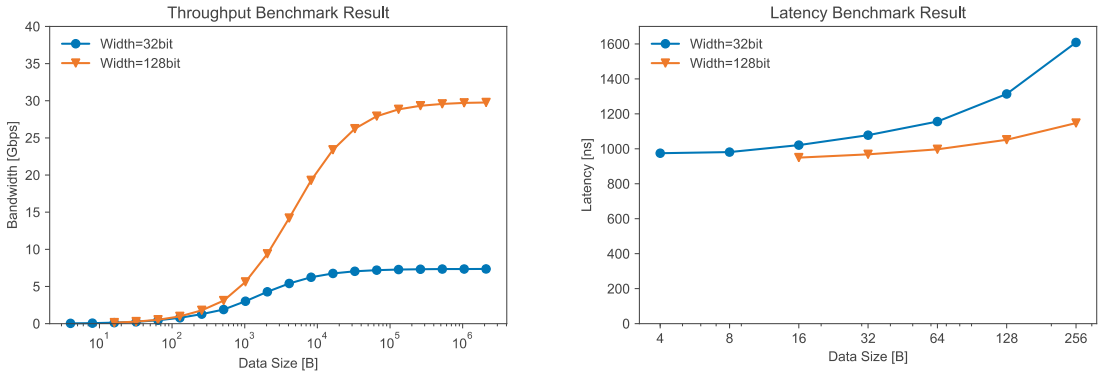


図 3 スループット（左）およびレイテンシ（右）の測定結果

図 4 に従来手法である InfinBand (IB) 経由の通信と CoE による調節通信の性能比較の結果と、CoE を袖領域交換および収束判定の Allreduce に適用した姫野ベンチマークの測定結果を示す。IB 環境には Mellanox ConnectX-4 EDR Host Channel Adapter (HCA)を用い、IB 通信ライブラリとして OpenMPI 3.0.1 を用いた。IB HCA は FPGA のメモリに PCIe バス経由で直接アクセスすることができないため、OpenCL API による CPU-FPGA 間転送と、MPI による CPU-CPU 間転送を store-and-forward で組み合わせて測定を行った。IB EDR の帯域は 100Gbps であり、40Gbps である CoE よりも高速であるにもかかわらず、CoE による通信は IB を用いた通信よりも高速であり、特に短・中サイズのメッセージで性能改善が大きい結果が得られた。CPU-FPGA 間の通信が並列計算時のボトルネックであり、それを FPGA 間の直接通信を用いることで解消できることがわかる。また、姫野ベンチマークの性能評価では、問題サイズ＝M，4FPGA 実行時に 23.7GFLOPS の性能が得られた。また、1 ノード実行時の性能と比べると 3.93 倍となっており、CoE の低遅延な通信によって高い並列化効率が得られていることがわかる。姫野ベンチマークで用いられているステンシル計算は、HPC アプリケーションで広

く用いられている計算手法であり、ステンシル計算に CoE を適用可能であることを示したことは、CoE が多くのアプリケーションに適用可能であることを示している。

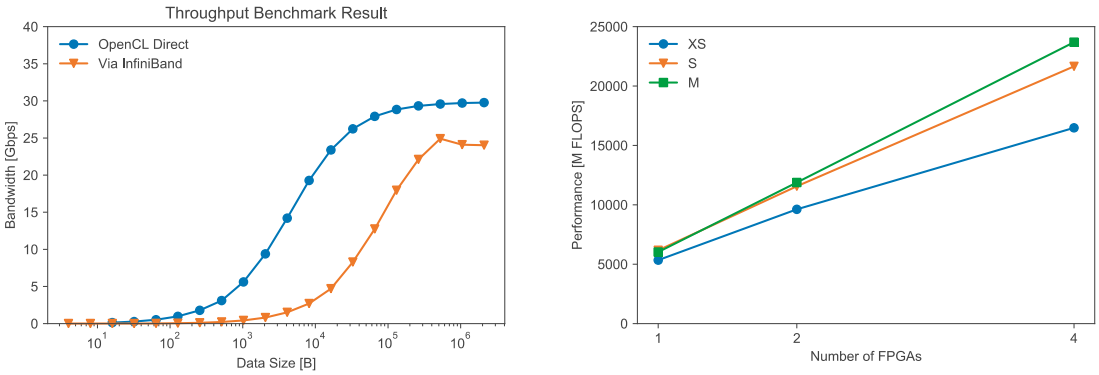


図 4 InfiniBand との性能比較（左） および姫野ベンチマークの測定結果（右）

今後は計算科学研究センターに本年度導入されたスーパーコンピュータ Cygnus への対応を進める予定である。Cygnus の各ノードは Intel Stratix 10 FPGA を 2 枚搭載し、システム全体で 64 枚の FPGA を有しており、それらの FPGA 間は 100Gbps の高速リンクによる 2D トーラスネットワークで結合されている。CoE の通信速度を 40Gbps から 100Gbps に向上させた 2D トーラスネットワークで、必要なパケットルーティング機構を組み込む予定である。この部分の実装には理化学研究所 R-CCS で開発中の FPGA 間通信ルータを共同研究に基づき導入する予定である。

【4】 疑似 MPI トレースファイルを用いた性能予測手法の研究（朴）

本研究では、理化学研究所 R-CCS との共同研究の下、多様な将来システムにおけるさまざまなアプリケーションの通信性能を平易に予測するために、実トレースを用いるネットワークシミュレータを拡張し、将来システムのサイズに合わせた疑似 MPI トレースファイルを生成して通信性能予測を行う手法である SCAMP を提案している。

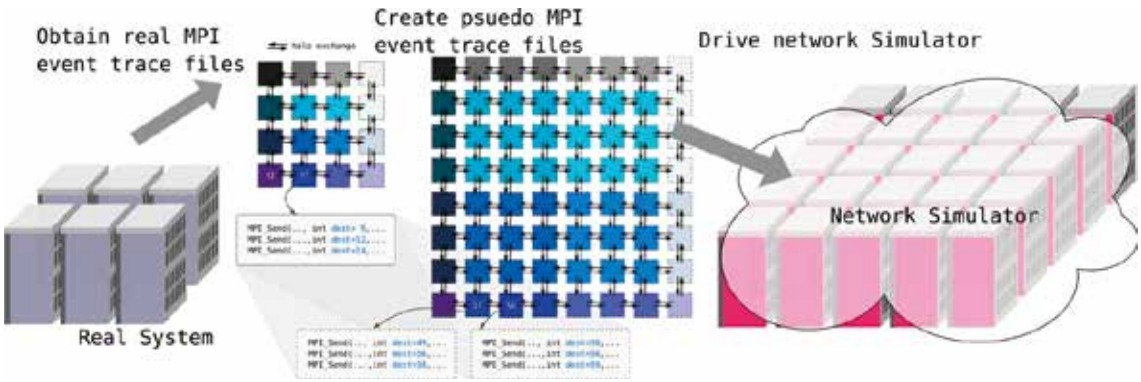


図 5 SCAMP 法の概要

大規模化し複雑化する HPC システムにおいて、システム導入当初からアプリケーションによる成果を早期創出するためには、システム設計段階からのアプリケーションとシステムのコデザインが重要である。アプリケーションとシステムのコデザインにおいては、アプリケーションの性能が発揮されるようシステムがデザインされる一方、アプリケーションもシステムに併せて最適化される、双方向的なアプローチがとられる。この過程では、複数のまだ存在しない将来システム候補のパラメータに対して、さまざまなアプリケーションの性能を推定する必要があるため、平易に利用できる性能予測ツールが必要とされる。

本研究では通信性能推定を対象として、特に実システム上での実行から得られる MPI トレースを用いる性能予測手法に焦点を当てる。トレースを入力として通信シミュレータを実行する手法は、(1) アプリケーションを実行し、各ランクの MPI トレースを集める、(2) 得られたトレースファイルを入力としてネットワークシミュレータを実行する、という機械的な作業で、幅広いアプリケーションに適用可能なため、広く用いられている。しかし、将来システムが現在のシステムよりも大規模な場合、それらに対応する MPI ランクのトレースファイルを得ることができず、通信性能の推定が行えないという問題があった。本研究では、実システムにおける実行で得られた少数の MPI トレースファイルを複製・編集して、将来システムの規模に合わせた数に水増し、大量の疑似 MPI トレースファイルを生成し、それをネットワークシミュレータの入力として用いる手法 SCAlable Mpi Profiler (SCAMP) 法を提案している。

図 5 に示すように、SCAMP 法では、

1. アプリケーションを実システムで実行し、実 MPI トレースファイルを得る。ファイルはランクごとに生成されるものとする
2. 実 MPI トレースファイルを複製し、将来システムの規模を考慮して MPI 関数の引数などの記録を編集し、疑似 MPI トレースファイルを生成する。1 つの実ファイルから 1 つ以上の疑似ファイルが生成される
3. 疑似 MPI トレースファイルと将来システムのノード数などのパラメータを入力として用いて、ネットワークシミュレータを実行する

ことで、大規模システム上での通信性能推定を行う。

本年度においては、疑似 MPI トレースファイルの自動生成手法の実装および性能評価を行った。SCAMP においては、将来システムの規模を考慮して MPI 関数の引数などの記録を編集し、1 つの実トレースファイルから、複数の疑似 MPI トレースファイルを生成する。疑似トレースファイルの生成は、平易な DLS などで行うか、もしくは自動的になされるべきであり、本年度は比較的単純な MPI アプリケーションについて、疑似トレースファイルを自動的に生成する手法を検討した。

表 1 MPI-event logs (4 nodes) and Pseudo MPI-event logs (8 nodes)

	Original file	Psuedo file
entering wall time	3038480.192285198	3038480.192285198
sendcount	512	256
sendtype	MPI_DOUBLE	MPI_DOUBLE
recvcount	512	256
recvtype	MPI_DOUBLE	MPI_DOUBLE
comm	MPI_COMM_WORLD	MPI_COMM_WORLD
returning wall time	3038480.192420677	3038480.192420677

疑似トレースファイルの例として、表 1 に 2048 要素の double 型配列を 4 ノードに分配し、なんらかの計算の後に MPI_Alltoall を行った場合のトレースファイルと、これを 8 ノードに置き換えて生成した疑似トレースファイルを示す。この例では、ノード数が倍になっていることからノードごとの配列の要素数が半分になり、MPI_Alltoall のバッファのサイズも半分になっている。本研究では、ターゲットランクの疑似 MPI トレースファイルを自動生成するために、LLVM-IR と呼ばれる中間言語を解析し、MPI 関数の引数を将来システムのノード数とランク数を用いて再計算し、特定のランクの実 MPI トレースファイルを編集する手法を実装した。

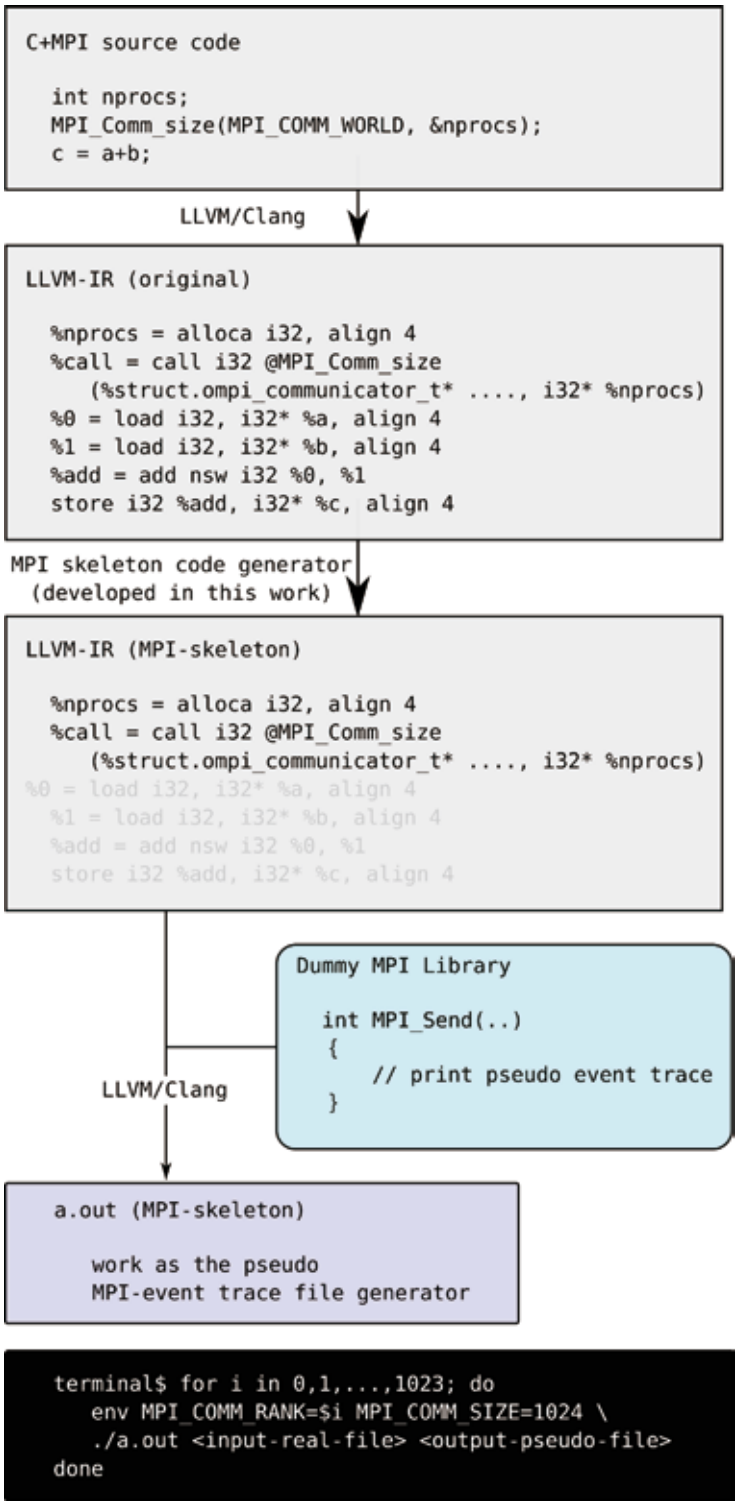


図 6 疑似 MPI トレースファイル生成の流れ

図 6 に疑似 MPI トレースファイル生成の流れを示す。まず、もとのアプリケーションソースコードを中間表現 LLVM-IR に変換する。変換された LLVM-IR コードを解析し、MPI 関数およびその引数の計算に必要な部分のみ抜粋し、MPI スケルトンコードを生成する。MPI ス

ケルトンコードを再度コンパイルし、ダミーMPI ライブラリとリンクして実行可能なバイナリファイルを生成する。このバイナリをターゲットシステムのサイズ、ランク、および実トレースファイルを入力として与えて実行すると、MPI スケルトンから呼び出されたダミーMPI ライブラリが、通信を行う代わりに、実トレースファイルの引数をターゲットサイズ、ランクを用いて MPI スケルトンコードにより再計算された引数と置き換え、疑似 MPI トレースファイルを生成する。

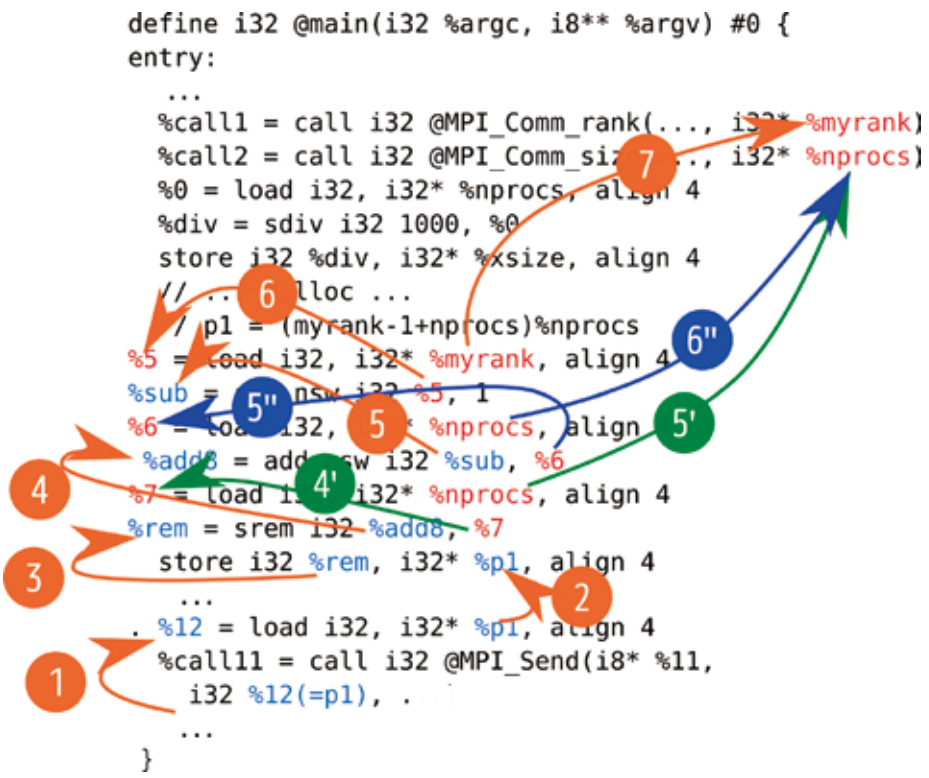


図 7　MPI スケルトンコードの生成の流れ

図 7 に、オリジナルの LLVM-IR を解析し、MPI スケルトンコードを生成する例を示す。この生成アルゴリズムは、MPI 関数を見つけた場合、通信性能に関係する引数 --- たとえばソースランクや、バッファの要素数 --- に必要な命令をターゲット変数の逆伝搬により抜き出す。

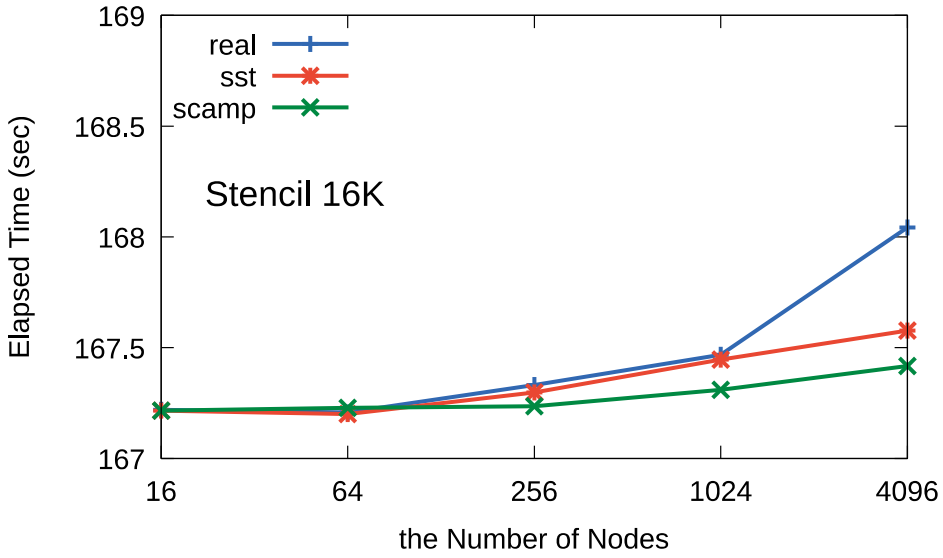


図 8　2 次元ステンシル計算の性能推定結果。Real は実実行時間、SST は対応するサイズの実 MPI トレースファイルを用いた推定時間、SCAMP は$4¥times 4$ノードの実ファイルから $8¥times 8$ $¥sim$ $64 ¥times 64$ ノードの疑似ファイルを生成した場合の推定時間。

図 8 に、2 次元ステンシル計算を用いた SCAMP 法の性能評価の結果を示す。ネットワークシミュレータとしては米サンディア国立研究所で開発されている SST/macro を用いた。16 ノードから 4096 ノードまでについて、京コンピュータ上での実実行時間、実 MPI トレースファイルを用いた推定時間、16 ノードから得られた疑似トレースファイルから 64 ノードから 4096 ノードまでの疑似トレースファイルを生成した推定時間について調査した。図から、SCAMP における推定結果は、ノード数の増加によるノード毎の計算時間のダイバーシティの増加を考慮していないため、実トレースを用いた推定よりはやや短いが、対応する実トレースによる SST/macro の推定時間にほぼ一致している。一方、実実行時間と実および疑似トレースファイルを用いた性能推定には、やや乖離がみられるが、これはネットワークシミュレータに実装されている通信アルゴリズムと京コンピュータの通信アルゴリズムの違いによるものと考えられる。

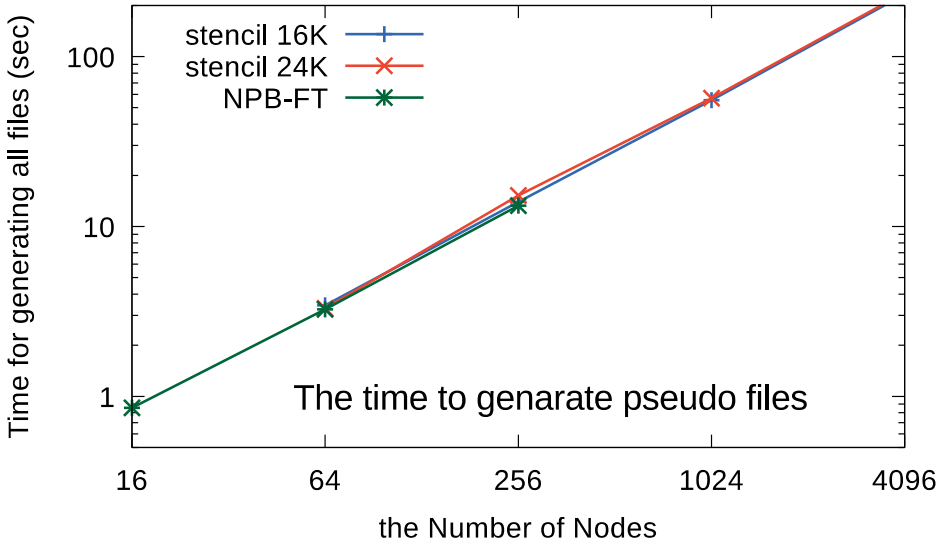


図 9　疑似トレースファイルの生成時間

次に、疑似トレースファイルの生成のコストを評価した。図 9 に、ステンシル、FFT について、16〜4096 個の疑似トレースファイルを生成した場合にかかった時間を示す。図 8 と比較すると、1 つのトレースの生成にかかる時間は実際の実行時間よりもじゅうぶんに短く、LLVM-IR の解析によるスケルトンコード生成は、MPI 関数の引数に不要な部分を効率的に除去していると考えられる。また、各ランクに対する生成は独立に行えることから、必要な個数の疑似トレースファイルの生成時間はノード数に対して、線形に増加している。これらは、スクリプトなどで平易に並列化可能である。

【5】　OpenMP タスク並列実行における FPGA オフローディングの性能最適化（朴）

理化学研究所 R-CCS との共同研究において、OpenMP における並列処理における演算加速装置として、FPGA (Field Programmable Gate Array)を想定する場合の演算カーネル作成に関する最適化の研究を行なった。従来の典型的な高速化の手法は大きな計算部分を 1 つの FPGA カーネルとして FPGA 上の論理ゲートをできるだけ多く利用し、性能を上げるのが一般的であった。つまり、大粒度の演算加速処理を行っていた。これに対し、OpenMP 4.0 からはタスクレベルでの並列処理が可能となり、かつ近年の FPGA は複数のカーネルを独立に論理回路上に配置し、これらを非同期に実行することが可能となっていることに着目し、中規模カーネルを複数、非同期実行することで全体の性能を向上させる試みを行った。現在、OpenMP 内の処理を直接コンパイルできる FPGA 環境は存在しないため、本年度の研究ではタスク管理のみをホスト CPU 上の OpenMP で行い、FPGA にオフローディングするカーネルは OpenCL で記述するというアプローチを取った。

昨年度までの共同研究では OpenMP のタスク処理自体を全て CPU 上のコア（マルチコアまたはメニーコア）で非同期処理し、全体性能を向上させる研究を行ったが、今年度はこれをさらに発展させ、FPGA におけるオフローディングに適用した。対象とするベンチマークはブロック化したコレスキー分解アルゴリズムで、昨年度までの研究により、比較的小さなデータを対象にした DGEMM ルーチンを非同期に並列実行することでタスク並列処理による性能向上が見込まれることがわかっている。DGEMM 自体を FPGA にオフローディングすることは OpenCL で比較的容易に記述可能であるため、このブロック化したコレスキー分解アルゴリズムを適用することにした。ただし、現在の FPGA は単精度浮動小数点演算はハード IP で高速化できるが、倍精度浮動小数点演算はまだ論理合成によって生成しないといけないため性能が低い。そこで、今回の評価では単精度 GEMM を用いた。

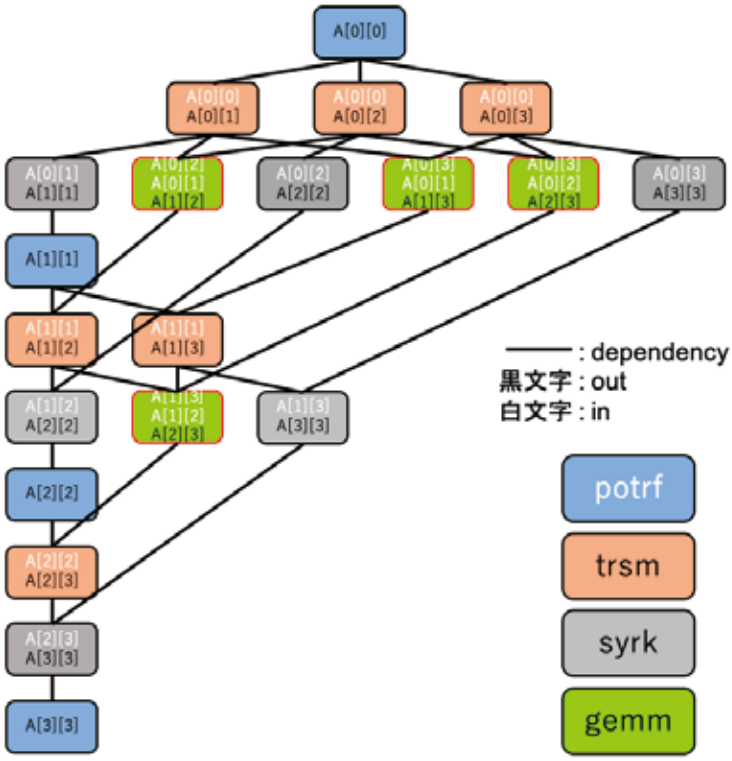


図 10　ブロック化されたコレスキー分解のタスクフローグラフ

図 10 にブロック化したコレスキー分解アルゴリズムのタスクフローを示す。ホスト CPU と FPGA は PCIe バスによって接続されており、いくつかの FPGA ドライブ関数を使ってデータの転送やカーネルの起動が可能である。write_data_to_fpga 関数はホストから FPGA へのデータ転送を行う関数、read_data_from_fpga 関数は FPGA からホストにデータ転送を行うプログラムである。 また、 enquque_request_to_fpga 関数はホストから FPGA にオフロードの依頼を行う関数である。 このプログラムでは、 行列データが既に FPGA DDR メモリに転送され

ていることを想定している。 図 10 に示されている通り、 GEMM が必要とするデータは TRSM タスクによってのみ更新されることから、 GEMM のオフロードを行う直前ではなく TRSM タスク終了時に FPGA へのデータ転送を行うことにより、 FPGA ホスト間のデータ転送を削減できる。

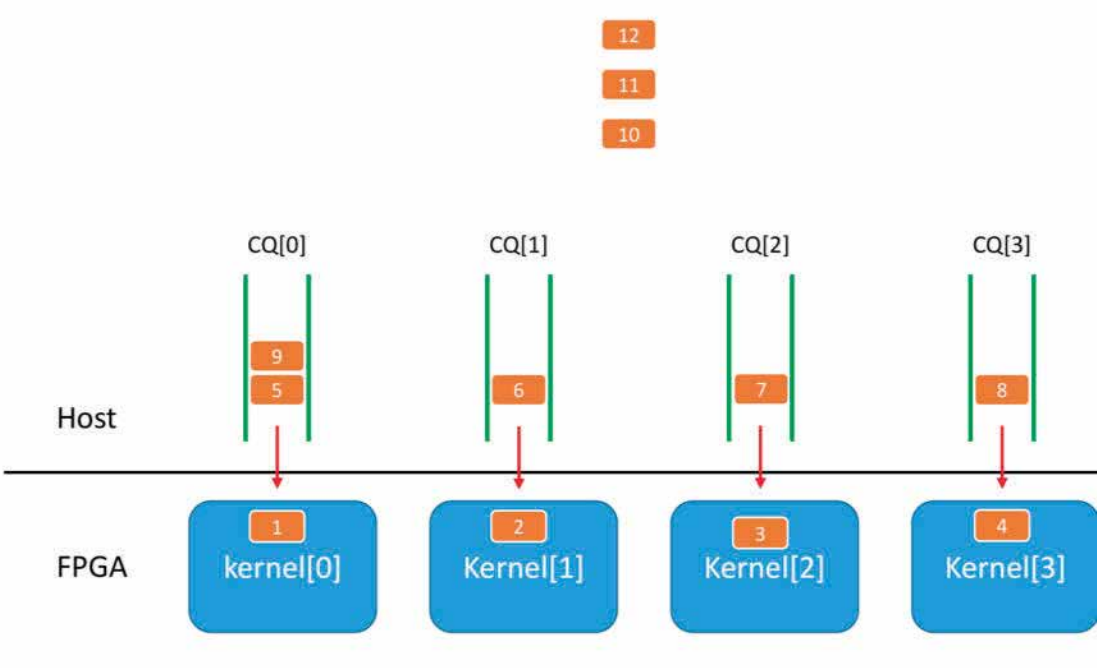


図 11　FPGA カーネルの非同期実行とホスト上のキューの様子

図 11 に、ホスト CPU から複数の FPGA カーネルを非同期に実行するためのコマンドキューの状況を示す。また、Listing 1 にこの処理を行うためのホスト上の OpenMP コードを示す。各カーネルへはタスクは 1 つずつしか投入実行できないが、各カーネル毎にキューを用意し、それらの切り替えをホスト上でプログラムすれば比較的均等にタスクを振り分けることが可能となる。この実装では FPGA カーネルはどれも同じ GEMM ルーチンを実行するので、タスクのキューへの振り分けは容易に決定できる。GEMM タスクについては、enqueue_request_to_fpga 関数や read_data_from_fpga 関数の実行後、taskyield でこれを適宜中断する。 ホストでのタスクが中断されている間も、FPGA ではバックグラウンドでオフロードされた計算の実行やデータ転送が行われる。 この手法により FPGA へのオフロード処理の非同期化および CPU/FPGA での計算のオーバーラップを実現する。 中断されたタスクが再開された際、 オフロードした計算が終了しているか、 もしくはデータ転送が終了したか確認を行う。 それらが終了していない場合、 再度中断し他のタスクを実行する。

Listing 1　プログラムの概要

```
void cholesky(const int ts、 const int nt、 float* A[nt][nt])
{
#pragma omp parallel
#pragma omp single
  for (int k = 0; k < nt; k++) {
//create POTRF task
#pragma omp task depend(out:A[k][k])
    {
       omp_potrf(A[k][k]、 ts、 ts);
    }
    for (int i = k + 1; i < nt; i++) {
//create TRSM task
#pragma omp task depend(in:A[k][k]) depend(out:A[k][i])
     {
       omp_trsm(A[k][k]、 A[k][i]、 ts、 ts);
       write_data_to_fpga(A[k][k]、 event0);
       waitForFinish(event0);
     }
    }
    for (int i = k + 1; i < nt; i++) {
      for (int j = k + 1; j < i; j++) {
//create GEMM task
#pragma omp task depend(in:A[k][i]、 A[k][j]) depend(out:A[j][i])
       {
         enqueue_request_to_fpga(gemmKernel、 。。。、 event1);
         do {
#pragma omp taskyield
           checkStatus(event1、 &ret);
         } while (ret != done);
         read_data_from_fpga(A[j][i]、 event2);
         do {
#pragma omp taskyield
           checkStatus(event2、 &ret);
```

```
        } while (ret != done);
      }
    }

//create SYRK task
#pragma omp task depend(in:A[k][i]) depend(out:A[i][i])
    {
      omp_syrk(A[k][i]、 A[i][i]、 ts、 ts);
    }
  }
}
#pragma omp taskwait
}
```

オフロードされる GEMM の計算を行うカーネルは、 NDRange モデルを用いて実装した。その際、 ブロッキングアルゴリズムを使用し、 ブロック当たりの大きさを 64 x 64 とした。DDR メモリへのアクセスを削減するため、 各ブロックを計算時に M20K Block RAM にロードし再利用した。 また、"num_simd_work_items"属性をパラメータとし、 カーネルインスタンスあたりのリソース使用量の調整を行った。 また、 カーネルインスタンスあたりのリソース使用量に応じて複製を行った。 他の最適化として、 ツールチェインが提供する 1KB メモインターリブを使用せず、 各データが確保されるメモリバンクの明示を行った。

以上のようなプログラミングを筑波大学計算科学研究センターの FPGA クラスタ実験システムである PPX (Pre-PACS-X)の 1 ノードに実装して性能評価を行った。表 2 に PPX システムの仕様を示す。

表 2　評価環境 PPX の仕様

CPU	Xeon E5-2660 v4 @2.00GHz x 2
Host DRAM	DDR4-2400 16GB x4
GPU	NVIDIA Tesla P100 PCIe x2
FPGA Board	BittWare A10PL4
FPGA	Intel Arria10 (10AX115N3F40E2SG)
FPGA DRAM	DDR4-2133 4GB x2
InfiniBand	Mellanox ConnectX-4 EDR
OS	CentOS 7.3 64bit

FPGA Compiler	Intel FPGA SDK for OpenCL 17.1.2
CPU Compiler	Intel C Compiler 18.0.1
CPU BLAS Library	Intel Math Kernel Library
Thread Library	Argobots 1.0b1

FPGA 上の GEMM カーネルの機能と回路規模を制御するパラメータとして、1 つのカーネル内の GEMM 処理を行う多重ループの最内側ループでの SIMD 並列性の制御を行った。SIMD の幅を 16、 8、 4 の 3 段階に分け、それぞれを Type1、 Type2、 Type3 と名付ける。SIMD 幅が広いカーネルは論理回路のゲート数を消費する。今回用いた Intel Arria10 FPGA では、約 70%の DSP リソースを用いるようにしたところ、Type1、 Type2、 Type3 のカーネルを実装できるインスタンス数は 1、 2、 4 個となった。例えば Type1 GEMM カーネルを 1 つ実行する場合と、Type3 GEMM カーネルを 4 つ同時に実行した場合では、基本的な理論ピーク性能は等しくなる。しかし、実際には論理合成の制約から各 Type のカーネル生成では動作周波数に差が見られた。これは実行性能を決める重要なファクタである。

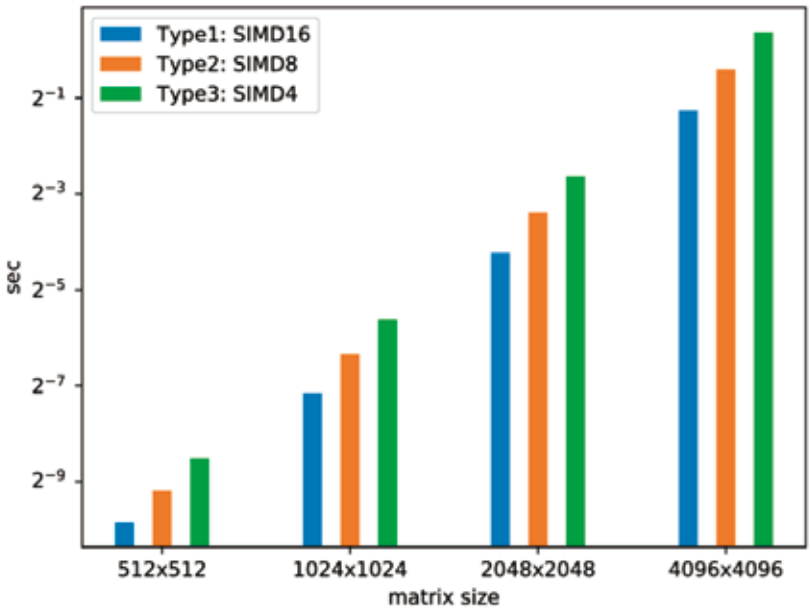


図 12　SIMD 幅を変えた 3 種類の DGEMM カーネルの実行時間比較

図 12 に 3 種類のカーネルの 1 つずつを実行した場合の実行時間を示す。問題サイズを 512x 512 から 4096 x 4096 まで変えた結果、全ての場合で Type1 から Type3 まで概ね回路規模に比例した実行時間短縮が見られた。まず、最も小規模である Type3 カーネルでは演算性能は安定しており、問題サイズを X x X とした時、X を 2 倍にすると演算時間はほぼ 8 倍になって

いるが、GEMM の演算量は X の 3 乗に比例するためこれはリーズナブルである。Type1 になると問題サイズが小さい場合はこの関係が崩れ、小さい問題では効率が悪くなっている。

次に、これらのカーネルを非同期並列実行可能としたプログラムで全体のスループットを評価した。Type1 カーネルは 1 つしかインスタンスがないため、全ての GEMM オフローディングは逐次的に FPGA に投げられるが、Type2 では 2 つのインスタンス、Type3 では 4 つのインスタンスが非同期並列実行可能である点が重要である。結果を図 13 に示す。

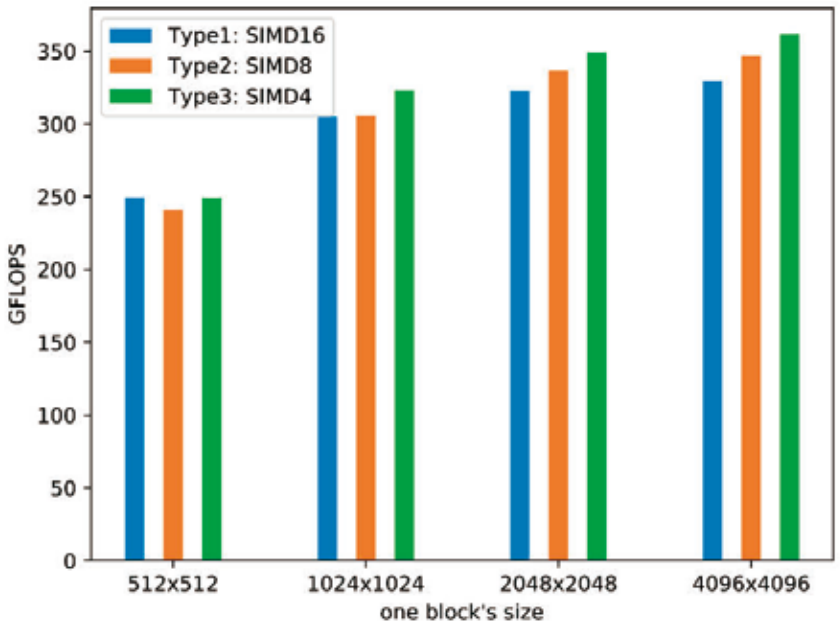


図 13　各 Type の DGEMM カーネルについて非同期同時実行を行わせた場合の性能

まず、どの Type のカーネルにおいても問題サイズが大きいほど性能が向上しており、これはカーネル起動のオーバーヘッド等が相対的に小さくなるためと思われる。次に、Type 別で見ると問題サイズが大きい場合、Type3 の総合性能（SIMD 幅の小さいカーネルを複数同時実行する）が Type1 を最大で約 10%上回っている。これには小さいカーネルを生成することで動作周波数が若干高いカーネルを論理合成できる可能性があることと、粒度の小さいカーネルの非同期実行というストラテジが性能を向上させる可能性があることが示された。

最後に、ブロック化コレスキー問題を OpenMP タスクを使って非同期並列実行可能としたプログラムで全体の性能を評価した。結果を図 14 に示す。

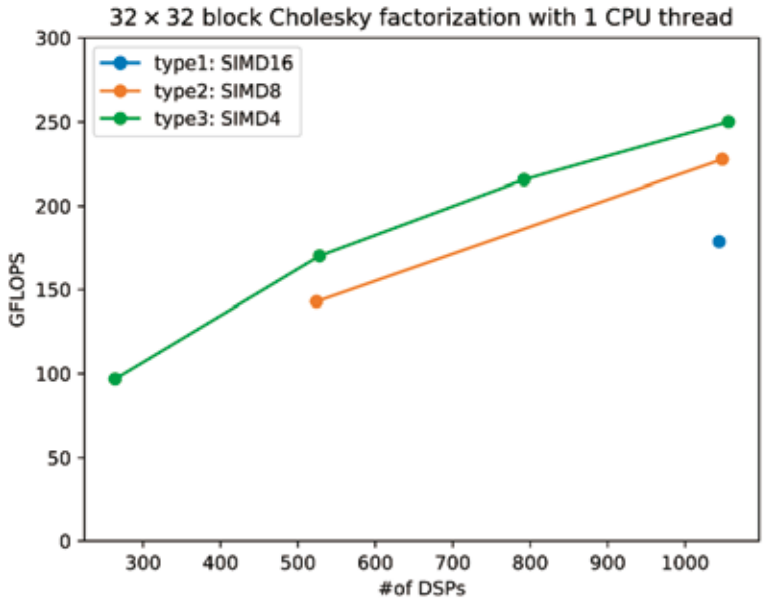


図 14　ブロック化コレスキー分解の非同期カーネル実装による性能評価

この結果から、このストラテジを実際の問題に適用した場合、同じ個数の DSP（この場合、1000 個程度）を消費した場合の性能は、Type1 が最も低く、Type3 が最も高いことがわかる。これらの結果より、同規模の比較的小さいカーネルを複数インスタンス生成でき、かつそれらを同時実行可能な場合、全体的な性能を向上させる可能性があることが示された。特筆すべきは、図 14 で示された以上の性能が、実際のコレスキー分解プログラムで示されたことである。さらなる詳細解析が必要であるが、我々のタスク並列 OpenMP プログラムに複数カーネルの非同期同時実行を組み合わせるアプローチが有効であることが示されたと言える。

【6】　地域気象乱流コード Large Eddy Simulation の並列 GPU 化（朴）

地球環境研究部門の日下博幸教授の研究グループと共同で、同グループによって開発された地域気象乱流コード Large Eddy Simulation (LES)の GPU 化を行なっている。本年度の研究では、CPU 版での実行で全実行時間の 70%程度を占める主要関数群について、カーネル単位での CUDA 化を行なった。その上で、GPU へのカーネルオフローディングの際に毎回、必要なデータを GPU のグローバルメモリにコピーし、カーネル実行終了後に CPU のメインメモリに引き上げるという作業、すなわち CPU-GPU コピーと GPU-CPU コピーの時間がオーバーヘッドとなることから、(1) GPU カーネルから次の GPU カーネルに引き継がれるデータ、かつ CPU 側では必要としないデータについては GPU のグローバルメモリに残してデータコピー時間を節約する、(2) 大部分の処理を GPU カーネル化することで、データのメインは GPU のグローバルメモリ上に常駐させ、必要に応じて CPU に戻る際に CPU-GPU メモリコピーを行う、という 2 種類の最適化を行なった。

対象とする LES コードは MPI 化されており、さらに CPU のマルチコアを有効利用するように各 MPI プロセスは OpenMP によるスレッド並列化されている。つまり、OpenMP+MPI のハイブリッドコードとなっている。マルチスレッド化されたコードであることが GPU 実装の CUDA 化におけるヒントとなっている。MPI 並列については大きな変更を加えなくてすむように、1 つの MPI プロセスが 1 つの GPU を担当するように実装した。例えば、実験環境 PPX（Pre-PACS-X、表 3 参照）では 2 ソケットの CPU に 14 コアがそれぞれ実装されており、CPU 版の LES コードでは 1 ノード当たり 2 MPI プロセスを走らせ、各プロセスが 14 スレッドの OpenMP 実行を行う。PPX には 2 台の GPU（NVIDIA Tesla P100 x2 または同 V100 x2）が搭載されているので、CPU の 1 ソケット分の処理を GPU の 1 台に対応させ、1 つの MPI プロセスが 1 つの GPU を管理するように実装した。

本年度の研究では 70%コード実行部分の GPU カーネル化と MPI 化を組み合わせた実装までを行なった。主要データを GPU 側に置いて CPU-GPU 及び GPU-CPU データコピー時間を短縮するところまでは実装が完了せず、現状では全てのデータは CPU 側にあり、GPU カーネル実行のたびに GPU のグローバルメモリへのコピーとその CPU メモリへの引き上げが必要である。

以上のようにして PPX の 1 ノード、2 GPU での実行を行なった結果、演算結果が正しいことを確認した。そこで、1 ノードでの 2 MPI プロセス（2 GPU）による性能評価を行なった。結果を図 15 に示す。CPU のみ（14 スレッド x プロセス）、GPU と CPU（Tesla P100 x2 または Tesla V100 x2）の併用について合計 3 種類の実行時間を測定した。計算は 100 タイムステップの総実行時間で、MPI 通信を含む。ただし、現時点のコードでは MPI 通信時間は軽微である。図 15 のグラフは積み上げ棒グラフで、CPU での実行（GPU 化されたコードでは CPU に残された処理）、GPU での実行、GPU から GPU へのデータコピー時間、CPU から GPU へのデータコピー時間、MPI 通信時間に分けて示している。実際には多数のカーネルや関数、さらに MPI 通信が実行されているが、各部分はそれぞれ対応する処理時間の総和となっている。

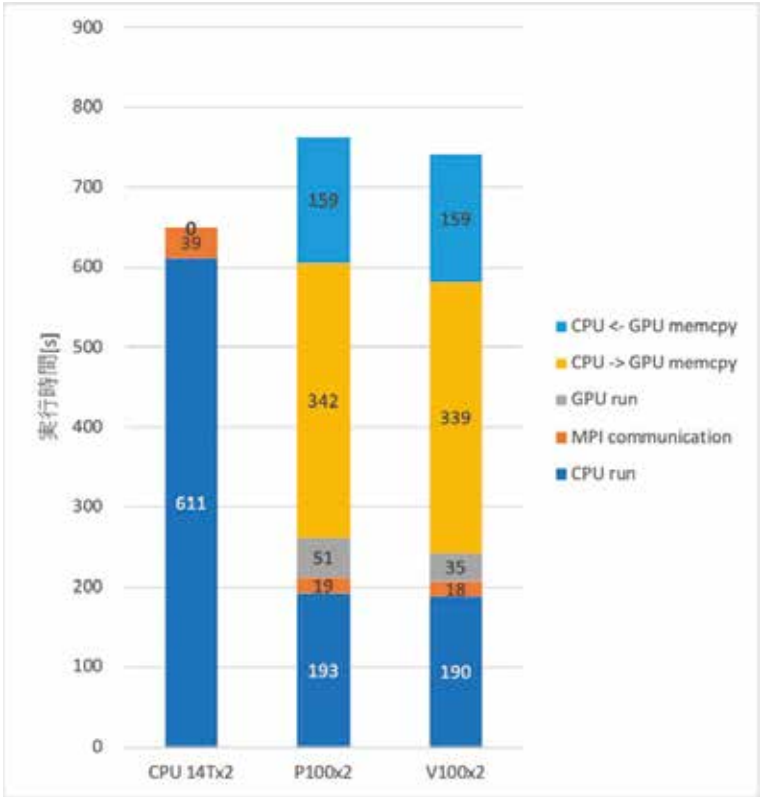


図 15　LES コードの CPU 及び GPU（Tesla P100 及び Tesla V100）での実行時間の比較

結果より、現時点では GPU 化されたコードの総実行時間は CPU のみの実行より 10%程度多くかかっていることがわかる。すなわち、GPU 化による計算時間は短縮されなかった。しかし、内訳を見ると CPU のみの総演算時間 611[sec]に対し、GPU 化ではこのうち 193[sec]または 190[sec]が残り、約 420[sec]の CPU 実行時間は 51[sec]または 35[sec]の GPU 実行までに短縮された。総演算時間で見れば CPU のみの 611[sec]が GPU 化により 244[sec]（P100）または 225[sec]（V100）まで短縮され、演算時間は 37~40%にまで短縮、演算性能としては 2.5～2.7 倍に加速されている。なお、GPU 化した場合の CPU 時間に若干の差があるのは P100 を実装したノードと V100 を実装したノードの CPU にバージョンの差があり、性能がわずかに違うためであるが､総実行時間に対し3.7%程度の差であるのでここでは無視することとする。

よって、この GPU 性能の低さは棒グラフの上部 2 つのセクション、CPU->GPU データコピーと GPU->CPU データコピーによって引き起こされている｡この部分の総時間は498~501[sec]で極めて大きい。上述のように、データの大部分を GPU 側メモリに置くことで、この時間は大幅に短縮されることが見込まれる。仮にこの部分が 1/3 程度に短縮されれば、GPU 化で 2 倍以上の高速化が期待できる。次年度はこの方針で実装を改良し、さらに高速化を目指す方針である。

【7】 メニーコアプロセッサ向けアプリケーション性能向上 （朴）

昨年度に継続し、量子物性研究部門の矢花一浩教授の研究グループとの共同研究の下、同グループで開発中の物性第一原理計算コード ARTED（Ab initio Real Time Electron Dynamics simulator）のメニーコアプロセッサ向け性能最適化を行った。H29 年度に本格稼働を開始した、JCAHPC における世界最大の Intel Xeon Phi（コード名 KNL: Knights Landing）クラスタである OFP（Oakforest-PACS）の全ノードを利用した同コードの性能評価を実施し、極めて良好な weak scaling 及び strong scaling 性能を達成している。

OFP 上の計算ノードに実装された Xeon Phi（KNL）は、従来のアクセラレータとしてのコプロセッサである Xeon Phi（コード名 KNC: Knights Corner）ではなく、self-bootable な CPU として動作する。このため、ノード間通信やメモリアクセスにおいて、KNC を用いていた COMA よりシンプルで効率的な処理が行えるだけでなく、KNC の約 3 倍の理論ピーク性能が実現され、大幅な性能向上が実現できる。しかし、基本的な性能チューニングは KNC とほとんど変わらないため、COMA 上で開発された KNC 向けの ARTED をほぼそのまま、OpenMP と MPI の呼び出し方を変更するだけで移植できる。図 16 に ARTED の最重要カーネル部分である、3 次元 25 点実空間ステンシル計算の主要部分コードを示す。ここでは non-temporal store の活用、512bit SIMD 命令の演算数と配置にフィットさせたデータ配列宣言、整数剰余演算を省くためのテーブル置き換え等の最適化を行っている。最終的には、KNL が採用する 512bit SIMD 命令の intrinsic を用いてベクトル化を手動実装し、ステンシル計算カーネルでは KNL の理論ピーク性能の 25%の実効性能が達成された。

```
real(8),   intent(in)  :: B(0:NLz-1,0:NLy-1,0:NLx-1)
complex(8),intent(in)  :: E(0:NLz-1,0:NLy-1,0:NLx-1)
complex(8),intent(out) :: F(0:NLz-1,0:NLy-1,0:NLx-1)

#define IDX(dt) iz,iy,modx(ix+(dt)+NLx)
#define IDY(dt) iz,mody(iy+(dt)+NLy),ix
#define IDZ(dt) modz(iz+(dt)+NLz),iy,ix

do ix=0,NLx-1
do iy=0,NLy-1
!dir$ vector nontemporal(F)
do iz=0,NLz-1
  v=0; w=0
  ! z-computation
  v=v+Cz(1)*(E(IDZ(1))+E(IDZ(-1))) ...
  w=w+Dz(1)*(E(IDZ(1))-E(IDZ(-1))) ...
  ! y-computation
  ! x-computation
  F(iz,iy,ix) = B(iz,iy,ix)*E(iz,iy,ix) &
  &           + A          *E(iz,iy,ix) &
  &           - 0.5d0*v - zI*w
end do
end do
end do
```

図 16 Xeon Phi 向けに最適化された 3 次元 25 点ステンシルコードのカーネル部分

また、KNL においては高バンド幅メモリである MCDRAM（16 GiB/node）と低バンド幅メモリである DDR4（96 GiB/node）をうまく利用することが必要であるが、ARTED コードは元

来、ノード上の全データを一括してスキャンするのではなく、3 次元ステンシル計算に必要な部分コピーを計算バッファにコピーしてから処理するようになっている。このため、この計算バッファを MCDRAM 上に確保することにより、ほとんど苦労せず flat mode における MCDRAM の有効利用が可能である。他の多くのコードが flat mode への移植困難のため、効率の落ちる cache mode を用いているのに対し、性能を最大化することが可能となった。計算のワーキングセットが MCDRAM の容量に収まる場合と、これを超える場合について、演算時間はリーズナブルであり FLOPS 値が落ちていない様子を図 17 に示す。

OFP 上ではその大規模並列性を活かし、波数空間での並列性を全計算ノードに展開した。ここで、各波数空間における 3 次元空間は比較的小さいため、これを複数ノードで並列する domain decomposition 法は用いない。これにより、ノード間通信コストを最小限にし、高い並列処理効率が望める。図 18 に、128 ノードまでの COMA 及び OFP における strong scaling 性能を示す。

もう一点、特筆すべき事項として、OFP の全ノード利用実験を通じて発見された重要な現象がある。KNL では一般的な Xeon CPU と同様、turbo boost mode が備わっており、プロセッサの電力消費目標を超えない範囲で瞬間的に動作周波数を向上させることができる。しかし、この boost 効果は演算コア単位で行われ、基本的に非同期である。また、多数の KNL プロセッサを用いる OFP のようなシステムでは各ノードの turbo boost も非同期に発生する。結果として、同一コードを同一データ量で処理しているにもかかわらず、その性能はノード毎にバラつくという現象が発生する。この現象は、OFP を部分的に利用している状態ではあまり目立たないが、本研究のような全系利用ではその影響が如実に現れており、性能低下を引き起こしていることがわかった。図 19 に、全系利用での ARTED 計算において、性能が最高のノードと最低のノードでの各部分処理の内訳を示す。ARTED では 1 つのハミルトニアン計算の中で数回の全系による MPI_Allreduce 処理が発生し、ここで全ノードの同期が取られるため、結果として最低速度のノードに性能が律速され、システム性能が相対的に低下することになる。現時点でこの現象に対する有効な手法は見つかっておらず、プロセッサを提供する Intel 社との協議においても重要な懸案となっている。

以上の成果は、現在矢花グループが中心に開発を行っているアプリケーション SALMON（Scalable Ab initio Light-Matter simulator for Optics and Nanosciences）に取り込まれる。SALMON は ARTED の発展的アプリケーションで、本共同研究が実施した内容をほぼすべてを活用できることが期待される。なお、2018 年 10 月に本研究の中心的実施者である廣川祐太氏が学位取得後、量子物性研究部門の研究員に配属されたため、2018 年度後期の成果は量子物性研究部門にて別途報告するので参照されたい。

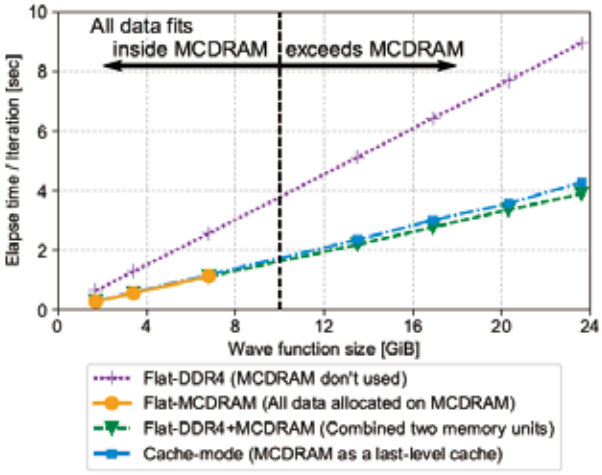


図 17　OFP の 1 ノード上で MCDRAM 容量を超えるデータを扱う場合の処理時間（MCDRAM 容量内のデータ処理に比べ線形に増加するのみで FLOPS 値は低下していない）

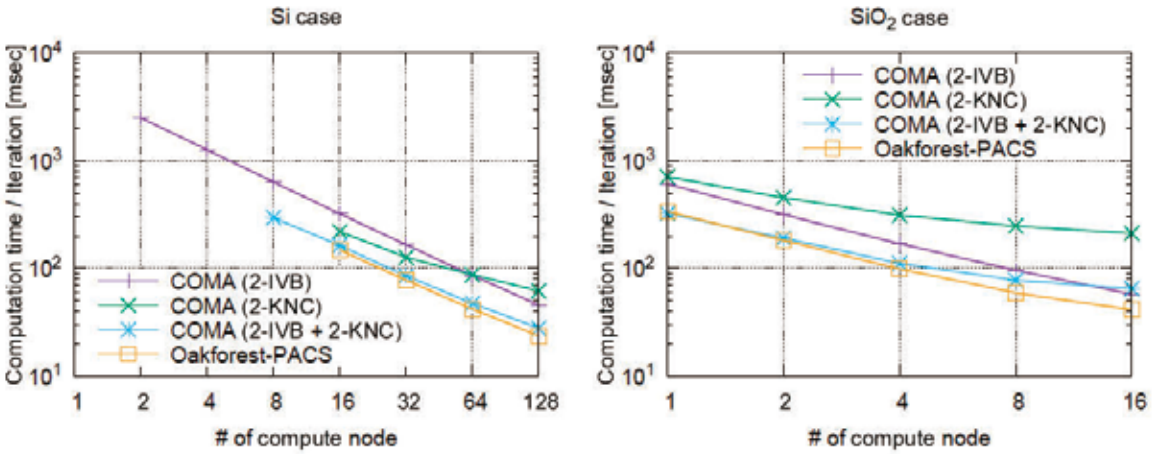


図 18　128 ノード及び 16 ノード（物質はそれぞれ Si 及び SiO_2）使用時の COMA（KNC）と OFP（KNL）の strong scaling 性能

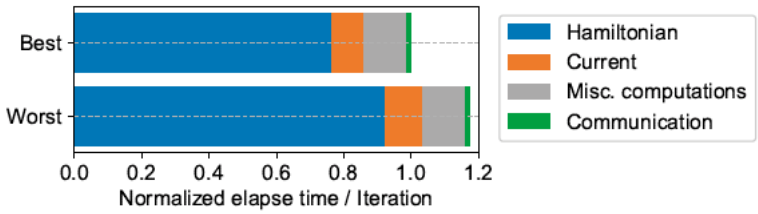


図 19　全系計算における turbo boost 状況に差により発生するノード間性能のバラつき

【8】 Intel Xeon Phi クラスタにおける 2 次元分割を用いた並列実数 3 次元 FFT の実現と評価（高橋）

高速フーリエ変換（fast Fourier transform、以下 FFT）は科学技術計算において今日広く用いられているアルゴリズムである。これまでに提案されてきた並列 3 次元 FFT における典型的なデータ分散方法としては、3 つの次元（x、y および z 軸）のうち 1 つの次元（例えば z 軸）のみが分割される。この場合、z 軸におけるデータ点数は MPI プロセス数以上となる必要がある。FFTW および Intel MKL（Math Kernel Library）は、この 1 次元分割を用いている。

最近のスーパーコンピュータでは、性能を向上させるためにコア数やプロセッサ数が増加する傾向にある。例えば、Sunway TaihuLight は 2018 年 11 月に TOP500 リストの 3 位にランクされ、そのコア数は 1000 万個を超えている。このようなシステムにおいては、MPI と OpenMP を用いたハイブリッド並列プログラミングにより MPI プロセス数を減らしたとしても、MPI プロセス数は最大で 1 万個以上になる。したがって、z 軸で 1 次元分割を行った場合、z 軸におけるデータ点数が 1 万点以上でなければならないことになり、3 次元 FFT の問題サイズに制約が生じることになる。この問題に対処する方法として、2 次元分割を用いた並列 3 次元 FFT が提案されている。しかし、Intel Xeon Phi クラスタにおける 2 次元分割を用いた並列 3 次元実数 FFT の実装はまだ報告されていない。そこで Intel Xeon Phi クラスタにおいて 2 次元分割を用いた並列 3 次元実数 FFT を実装して評価を行った。

FFT は離散フーリエ変換（discrete Fourier transform、以下 DFT）を高速に計算するアルゴリズムとして知られている。DFT は次式で定義される。

$$y(k) = \sum_{j=0}^{n-1} x(j)\omega_n^{jk}, \qquad 0 \le k \le n-1 \quad (1)$$

ここで、$\omega_n = e^{-2\pi i/n}$、$i = \sqrt{-1}$である。

DFT の入力データ$x(j)$が実数の場合、2 つのn点実数 DFT は 1 つのn点 DFT を使って効率的に計算することができる。複素数入力データの実数部と虚数部に 2 つの実数データを入れて、複素 DFT を実行する。

$$x(j) = x_1(j) + ix_2(j), \qquad 0 \le j \le n-1 \quad (2)$$

ここで、$x_1(j)$および$x_2(j)$は 2 つのn点実数入力データである。

複素 DFT の出力データ$y(k)$に関して、DFT の複素共役性から以下の式が成り立つ。

$$y(k) = y_1(k) + iy_2(k), \qquad 0 \le k \le n-1 \quad (3)$$

$$\bar{y}(n-k) = y_1(k) - iy_2(k), \qquad 0 \le k \le n-1 \quad (4)$$

ここで、$y_1(k)$および$y_2(k)$はそれぞれ$x_1(j)$および$x_2(j)$のn点実数 DFT である。

2 つのn点実数 DFT の出力データ$y_1(k)$および$y_2(k)$は式(3)および(4)より、

$$y_1(k) = \frac{1}{2}\{y(k) + \bar{y}(n-k)\}, \qquad 0 \le k \le n-1 \quad (5)$$

$$y_2(k) = -\frac{i}{2}\{y(k) - \bar{y}(n-k)\}, \qquad 0 \le k \le n-1 \quad (6)$$

と計算することができる。

3 次元 DFT は以下の式で与えられる。

$$y(k_1,k_2,k_3) = \sum_{j_1=0}^{n_1-1}\sum_{j_2=0}^{n_2-1}\sum_{j_3=0}^{n_3-1} x(j_1,j_2,j_3)\omega_{n_1}^{j_1k_1}\omega_{n_2}^{j_2k_2}\omega_{n_3}^{j_3k_3},$$

$$0 \le k_1 \le n_1-1, \qquad 0 \le k_2 \le n_2-1, \qquad 0 \le k_3 \le n_3-1$$

ここで、$\omega_{n_r} = e^{-2\pi i/n_r}$ $(1 \le r \le 3)$、$i = \sqrt{-1}$である。

DFT の共役対称性と row-column アルゴリズムに基づく 3 次元実数 FFT アルゴリズムは以下のようになる。

Step 1: n_2n_3組のn_1点 multicolumn 実数 FFT

$$x_1(k_1,k_2,k_3) = \sum_{j_1=0}^{n_1-1} x(j_1,j_2,j_3)\,\omega_{n_1}^{j_1k_1}$$

Step 2: 転置

$$x_2(j_2,j_3,k_1) = x_1(k_1,j_2,j_3)$$

Step 3: $n_3 \cdot (n_1/2+1)$組のn_2点 multicolumn 複素 FFT

$$x_3(k_2,j_3,k_1) = \sum_{j_2=0}^{n_2-1} x_2(j_2,j_3,k_1)\,\omega_{n_2}^{j_2k_2}$$

Step 4: 転置

$$x_4(j_3,k_1,k_2) = x_3(k_2,j_3,k_1)$$

Step 5: $(n_1/2+1)\cdot n_2$組のn_3点 multicolumn 複素 FFT

$$x_5(k_3,k_1,k_2) = \sum_{j_3=0}^{n_3-1} x_4(j_3,k_1,k_2)\,\omega_{n_3}^{j_3k_3}$$

Step 6: 転置

$$y(k_1,k_2,k_3) = x_5(k_3,k_1,k_2)$$

Step 1 では、n_2n_3組のn_1点 multicolumn 複素 FFT および式(5)、(6)を用いて、n_2n_3組のn_1点 multicolumn 実数 FFT を計算することができる。もしn_2n_3が 2 で割り切れない場合、残りのn_1点実数 FFT は、入力データの虚数部が 0 であるn_1点複素 FFT を用いて計算する。Step 1 におけるn_1点実数 FFT の出力データは conjugate-even 対称性に従うので、$(n_1/2+1)$点の複素出力データのみを記憶すれば十分である。

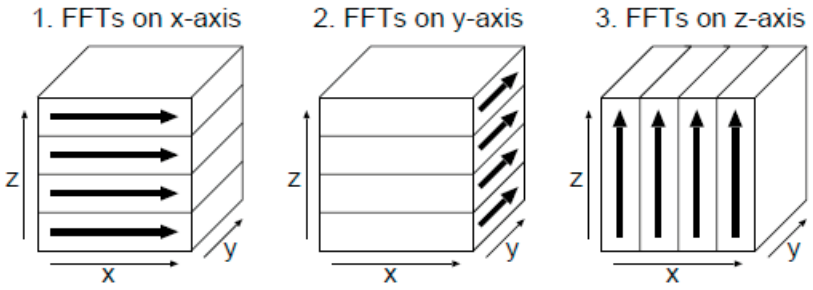


図 20　1 次元分割を用いた場合の並列 3 次元実数 FFT

図 20 は、z 軸上で初期データを 1 次元分割した並列 3 次元実数 FFT を示している。これは、並列 3 次元実数 FFT における典型的なデータ分散方法である。2 次元分割を用いた並列 3 次元実数 FFT の提案する実装は、DFT の共役対称特性および 2 次元分割を用いた並列 3 次元複素 FFT に基づいている。

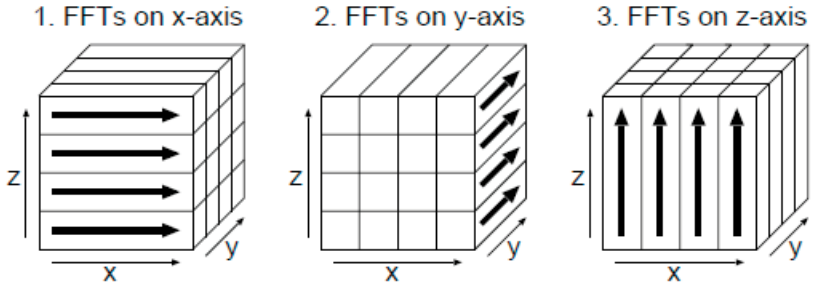


図 21　2 次元分割を用いた場合の並列 3 次元実数 FFT

図 21 は、y 軸と z 軸の初期データを 2 次元分割した並列 3 次元実数 FFT を示している。Intel Advanced Vector Extensions 512（Intel AVX-512）命令を使用して FFT カーネルをベクトル化した。OpenMP を使用して multicolumn FFT を並列化すると、各 FFT ステップの最外側ループが複数のスレッドに分散される。この場合、最外側ループ長は Intel Xeon Phi プロセッサに対して十分な並列性がない可能性があるが、OpenMP の collapse 節を使用することによって、最外側ループの並列性を拡張することができる。

性能評価にあたっては、提案する並列 3 次元実数 FFT である FFTE（version 7.0、2 次元分割）の性能を FFTE（version 7.0、1 次元分割）、FFTW（version 3.3.8）および P3DFFT（version 2.7.7）と比較した。測定に際しては、weak scaling と strong scaling における実数 FFT を連続 10 回実行し、その平均の経過時間を測定した。なお、FFT の計算は倍精度実数で行い、三角関数のテーブルはあらかじめ作り置きとしている。入力と出力では同じデータ分散を使用した。FFTW では、”measure” planner を使用した。Xeon Phi クラスタとして、最先端共同 HPC 基盤施設（JCAHPC）に設置されている Oakforest-PACS（8208 ノード）の 1～2048 ノードを用いた。FFTE および P3DFFT に対しては、コンパイラは Intel Fortran Compiler version 18.0.1.163

を用い、コンパイルオプションは”mpiifort -O3 -xMIC-AVX512 -qopenmp”を指定した。FFTW および P3DFFT に対しては、コンパイラは Intel C Compiler version 18.0.1.163 を用い、コンパイルオプションは”mpiicc -O3 -xMIC-AVX512 -qopenmp”を指定した。MPI ライブラリは Intel MPI 2018.1.163 を用いた。各ノードあたりの MPI プロセス数は 4、各 MPI プロセスあたりのスレッド数は 64 に設定し、環境変数“KMP AFFINITY=balanced”を設定して flat/quadrant モードで MCDRAM のみを用いて実行した。$N = 2^m$点実数 FFT の GFlops 値は$2.5N \log_2 N$より算出している。

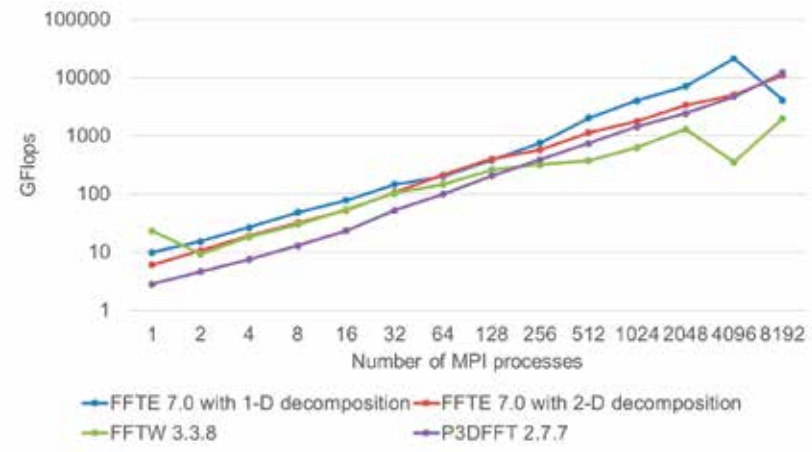


図 22　並列 3 次元実数 FFT の weak scaling 性能（$N = 256 \times 512 \times 512 \times$ MPI プロセス数）

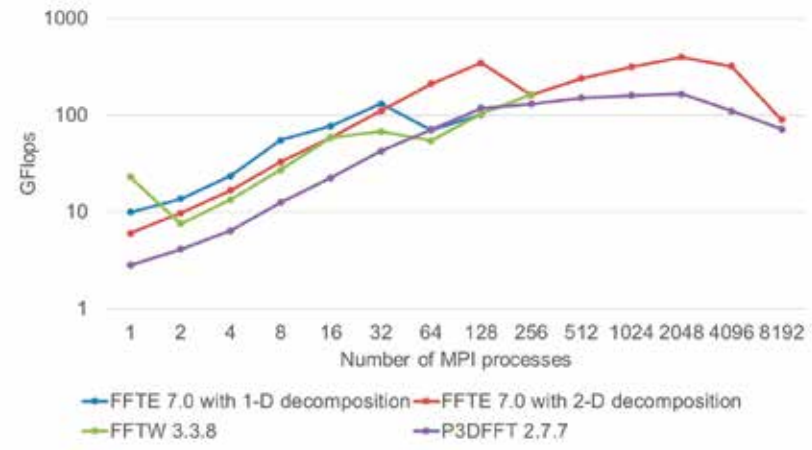


図 23　並列 3 次元実数 FFT の strong scaling 性能（$N = 256 \times 512 \times 512$）

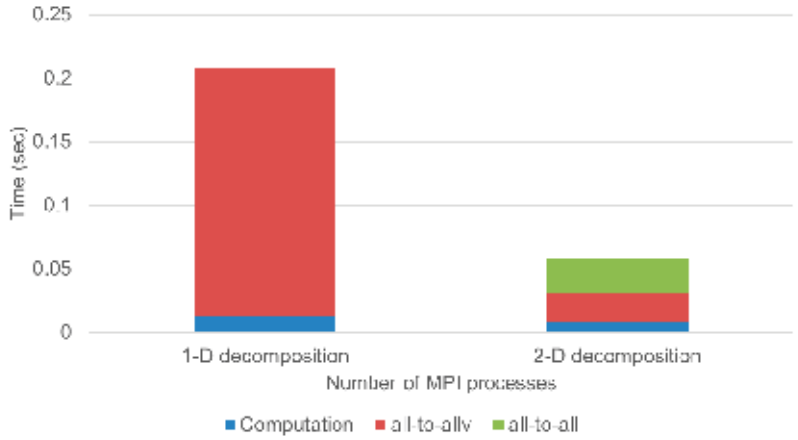


図 24　FFTE 7.0 の実行時間の内訳（$N = 1024^3$、512 MPI プロセス）

並列 3 次元実数 FFT の weak scaling 性能（$N = 256 \times 512 \times 512 \times$ MPI プロセス数）を図 22 に示す。図から分かるように、64、128 および 8192 MPI プロセスにおいて FFTE（1 次元分割)は FFTE（2 次元分割）よりも高速であることが分かる。これは、1 次元分割における全体の通信量は 2 次元分割の約半分であるのが原因であると考えられる。また、FFTE（2 次元分割）は 1、16 MPI プロセスを除いて FFTW よりも高速であり、8192 MPI プロセスを除いて P3DFFT よりも高速である。

並列 3 次元実数 FFT の strong scaling 性能（$N = 256 \times 512 \times 512$）を図 23 に示す。図から分かるように、FFTE（2 次元分割）は 64、256 MPI プロセスにおいて FFTE（1 次元分割）よりも高速である。これは 2 次元分割の方が 1 次元分割よりも通信レイテンシの影響が少ないことが原因であると考えられる。また、FFTE（2 次元分割）は 1、256 MPI プロセスにおいて FFTW よりも高速であり、P3DFFT よりも高速である。

FFTE 7.0 の実行時間の内訳（$N = 1024^3$、512 MPI プロセス）を図 24 に示す。図から分かるように、2 次元分割の通信時間は 1 次元分割よりも短くなっている。これは 1 次元分割における all-to-allv 通信のメッセージサイズが 32KB であるのに対して、2 次元分割における all-to-all(v)通信のメッセージサイズが 16MB であるのが原因であると考えられる。

【9】　Intel AVX-512 命令を用いた複数の整数除算の高速化（高橋）

整数除算は多くのアプリケーションで広く用いられている演算の一つである。一般的に除算は加減乗算に比べて遅いことが知られている。多くのプロセッサでは整数加減乗算の SIMD 命令がサポートされているが、整数除算の SIMD 命令をサポートしているプロセッサはほとんど存在しないのが現状である。

Intel 64 命令セットの div 命令は 128 ビットの被除数と 64 ビットの除数に対する符号なし整数除算を行う。Intel SVML（Short Vector Mathematical Library）にはベクトル化された整数

除算の組み込み関数が含まれているが、被除数および除数は 8 ビットから 64 ビットまでであり、128 ビットの被除数と 64 ビットの除数に対する符号なし整数除算には対応していない。また、逆数を用いて整数除算を求めるアルゴリズムが提案されているが、ベクトル化は考慮されていない。そこで、Intel AVX-512 命令を用いて複数の 128 ビットの被除数と 64 ビットの除数に対する符号なし整数除算を高速化し、性能評価を行った。

符号なし整数除算は被除数をA、除数をBとすると、商$Q = \lfloor A/B \rfloor$および剰余$R = A - BQ \ \ (0 \le R < B)$で定義される。提案手法は複数の 128 ビットの被除数と 64 ビットの除数に対する符号なし整数除算を再帰除算により行う。Algorithm 1 に再帰除算アルゴリズムを示す。被除数が 128 ビットで除数および商が 64 ビットの符号なし整数除算は、Algorithm 1 において$\beta = 2^{32}$、$n = m = 2$、$A < \beta^m B$とすると、96 ビットの被除数と 64 ビットの除数に対する符号なし整数除算を 2 回行うことで計算することができる。

Algorithm 1　再帰除算アルゴリズム

Algorithm 1 RecursiveDivRem

Input: $A = \sum_0^{n+m-1} a_i \beta^i$, $B = \sum_0^{n-1} b_j \beta^j$, $\beta^n/2 \le B < \beta^n$, $n \ge m$
Output: quotient Q and remainder R of A divided by B

1: **if** $m < 2$ **then return** $Q := A \operatorname{div} B$, $R := A \bmod B$
2: $k \leftarrow \lfloor m/2 \rfloor$, $B_1 \leftarrow B \operatorname{div} \beta^k$, $B_0 \leftarrow B \bmod \beta^k$
3: $(Q_1, R_1) \leftarrow$**RecursiveDivRem**$(A \operatorname{div} \beta^{2k}, B_1)$
4: $A' \leftarrow R_1 \beta^{2k} + (A \bmod \beta^{2k}) - Q_1 B_0 \beta^k$
5: **while** $A' < 0$ **do** $Q_1 \leftarrow Q_1 - 1$, $A' \leftarrow A' + \beta^k B$
6: $(Q_0, R_0) \leftarrow$**RecursiveDivRem**$(A' \operatorname{div} \beta^k, B_1)$
7: $A'' \leftarrow R_0 \beta^k + (A' \bmod \beta^k) - Q_0 B_0$
8: **while** $A'' < 0$ **do** $Q_0 \leftarrow Q_0 - 1$, $A'' \leftarrow A'' + B$
9: **return** $Q := Q_1 \beta^k + Q_0$, $R := A''$.

96 ビットの被除数と 64 ビットの除数に対する符号なし整数除算を行う際には、上位 64 ビットの被除数と上位 32 ビットの除数に対する符号なし整数除算で商を近似する。除数Bが$\beta^n/2 \le B < \beta^n$に正規化されている場合、正確な商を$Q$とすると商の近似値は$Q$、$Q+1$、$Q+2$のいずれかになる。したがって、剰余$R = A - BQ$が$0 \le R < B$の範囲に収まるように商の近似値を補正することにより正確な商が得られる。

性能評価にあたっては、複数の 128 ビットの被除数と 64 ビットの除数に対する符号なし整数除算の性能を、提案手法と Intel 64 命令セットの div 命令で比較した。除数は$1 \sim 2^{64} - 1$の範囲の乱数とし、被除数は$0 \sim 2^{127} - 1$の範囲の乱数で商が$2^{64} - 1$以下になるものとした。256 要素の符号なし整数除算を 100 万回実行し、その平均の経過時間から 1 秒あたりの符号なし整数除算回数（Mops）を算出した。

表 3　Intel Xeon Phi 7250 における複数の 128 ビットの被除数と 64 ビットの除数に対する符号なし整数除算の性能

	性能（Mops）
提案手法	32.763
div 命令	13.985

評価環境として、Intel Xeon Phi 7250 の 1 コア、1 スレッドを用いた。コンパイラは Intel C compiler 18.0.3.222 を用い、コンパイルオプションは”icc -O3 -xMIC-AVX512”を用いた。

Intel Xeon Phi 7250 における複数の 128 ビットの被除数と 64 ビットの除数に対する符号なし整数除算の性能を表 3 に示す。表から提案手法が div 命令よりも約 2.34 倍高速であることが分かる。

【10】 エクストリームビッグデータの基盤技術（建部）

エクストリームビッグデータ（EBD）アプリケーションの実行に求められる、数万～数十万プロセスからの並列アクセスを想定した IOPS、プロセス数に比例した読込、書込アクセスバンド幅性能を目標として、分散オブジェクトストアの研究を行っている。

アプリケーショングループとのコデザインにより、オブジェクトストアに対するアクセスにおいても POSIX インターフェースが有用であることが分かった。そのため、POSIX インターフェースでのアクセスを実現するため、分散メタデータサーバ PPMDS の研究開発を進めた。PPMDS は分散キーバリューストアを用いた分散メタデータサーバである。POSIX で必要となる階層型の名前空間とその操作を実現するためには、複数キーの変更をアトミックに行う必要がある。PPMDS ではこの問題に対し、動的ソフトウェアトランザクショナルメモリをベースとしたトランザクショナルキーバリューペアを用いて、分散トランザクションを実現した。階層型のディレクトリ構造は、親ディレクトリ ID とエントリ名をキーとするキーバリューで管理する。キーバリューペアはハッシュにより分散される。ディレクトリリスティングは、親ディレクトリ ID の前方一致検索で行う。ディレクトリの移動や削除などの操作は、複数のメタデータサーバに格納されるキーバリューペアを一貫して変更する必要があり、そのために分散トランザクションが必要となる。本設計に基づいた PPMDS を実装し、性能評価を行った。

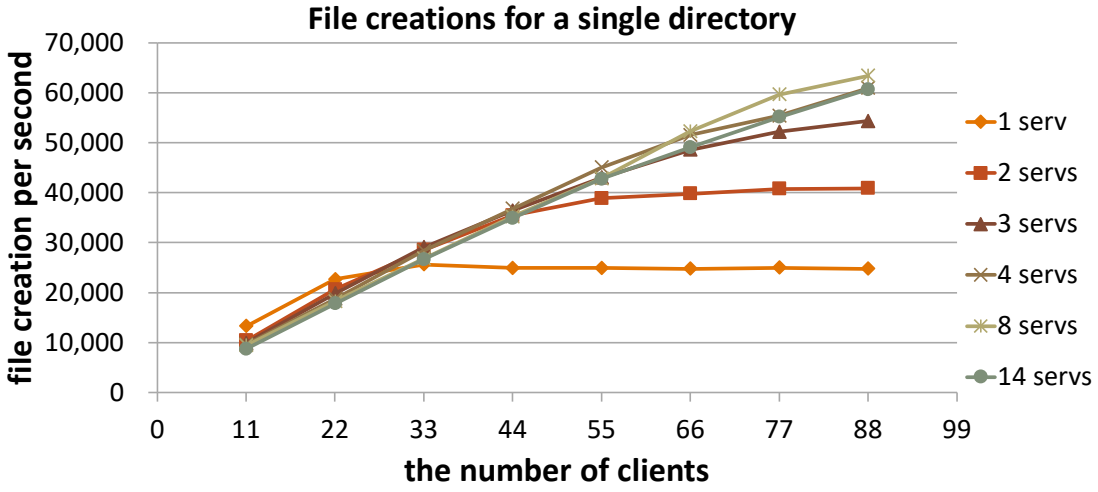


図 25　メタデータサーバのスケーラビリティ

図 25 にクライアント数を増やした時のファイル作成性能を示す。ファイル作成性能は 1 秒間に作成したファイル数を示している。メタデータサーバ数を増やして性能を評価したところ、メタデータサーバ数を増やすと性能が向上し、スケーラブルな性能を示した。本性能は、他の分散メタデータサーバの IndexFS に比べサーバあたりの性能が高い。本成果は The Fifth International Conference on Social Networks Analysis, Management and Security (SNAMS-2018)に併設したワークショップにおいて発表し、複数あるワークショップ全体におけるベストペーパアワードを受賞した。

【11】 極端気象予測を拓くビッグデータ機械学習基盤の研究（建部）

豪雨・突風・高温などの極端気象は人類に甚大な被害をもたらすが、その予測は極端気象に関する膨大な知識が必要である。本研究では、その知識を効率的に生成する機械学習基盤の構築を目的とする。平成 30 年度は、極端気象に関する知識を獲得するため，深層学習の効率化を図った。深層学習では、学習の効率をあげるため、複数の学習を同時に行うミニバッチ学習が行われる。しかしながら、このミニバッチのサイズによって GPU の利用効率が変わり、最適なミニバッチサイズを設定するのは試行錯誤が必要であった。そのため、この試行錯誤を自動化するための研究を行った。GPU の利用効率が向上する間はミニバッチサイズを増やしていくという単純な方法では、データサイズが大きくなりデータの入力がボトルネックとなるなど、GPU の計算以外のところがボトルネックとなるケースにおいて最適な値を推定することが困難であった。そのため、1 エポック当たりの平均計算時間が最小となるようにミニバッチサイズを増やしていく手法の提案を行った。本手法により、GPU の計算以外のところがボトルネックとなるケースにおいても十分なミニバッチサイズを推定することが可能となった。今後は、特に学習データが大きくなった場合の検討を進める。学習データが大き

くなると、学習データをいかに高速に入力するかが問題となる。これまでの研究では、メモリからの入力を高速化する研究はあるが、ストレージからの入力を高速化する研究は少ない。ストレージからの入力を高速化する研究をすすめていく。

【12】 ビッグデータサイエンスに関する研究（建部）

ビッグデータを用いたサイエンスの研究として、すばる望遠鏡の Hyper Sprime-Cam（HSC）で撮影された天体データのパイプラインデータ処理の高速化を行った。HSC は 116 枚の CCD で構成され、それぞれの CCD は 4,272×2,272 ピクセルである。一晩で約 300GB のデータが生成される。パイプラインデータ処理はその天文データを天文学の研究で利用できるようにデータ処理を行うものである。超新星など時間的な変化の速い物体を検出するためには、高速にデータ解析を行う必要がある。

これまでのパイプラインデータ解析では、ファイルシステムとして GPFS が用いられ、並列処理フレームワークとして scatter-gather のみ利用可能な MPIpool が用いられていたが、計算ノード数を増やした場合、並列データ解析の性能が頭打ちになるという問題があった。この問題を解決するため、計算ノード数を増やしても性能が頭打ちにならない Gfarm ファイルシステムを用いた。またデータ解析のワークフローに対し、Pwrake を用い、タスク間の細粒度な依存性によってデータ解析を進めることにより、計算ノードの CPU コアの利用率の向上とタスクのオーバラップを実現し、この問題の解決を図った。

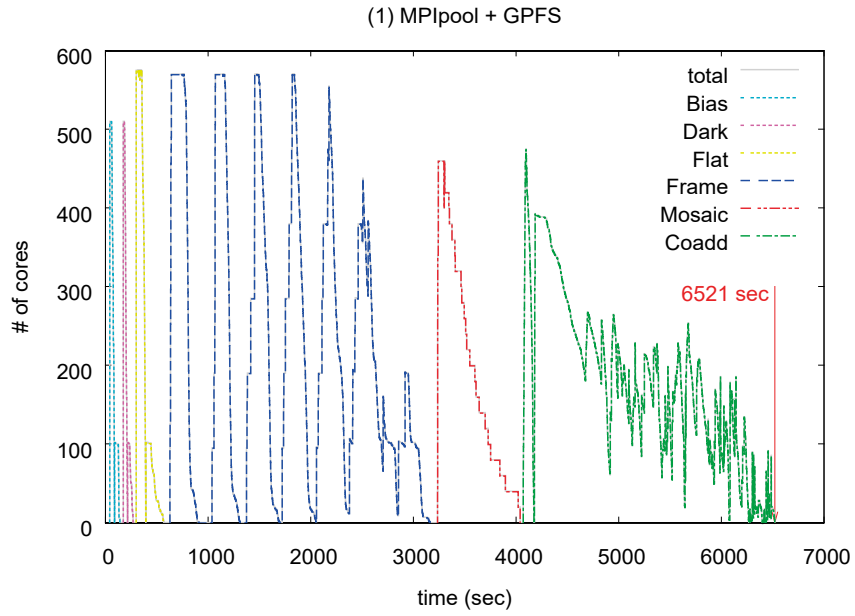


図 26 これまでの方法によるデータ解析の模様

図 26 に一晩の 1/4 のデータを用いたデータに対するこれまでの方法によるデータ解析の模様を示す。横軸は経過時間であり、縦軸は同時に利用している CPU コア数を示している。Frame 計算では、574 コアの全ての計算が終了するのを待っている時間が何度か生じている。また各データ解析のステップで終了を待っている。これらにより、コアの利用率が下がっている。

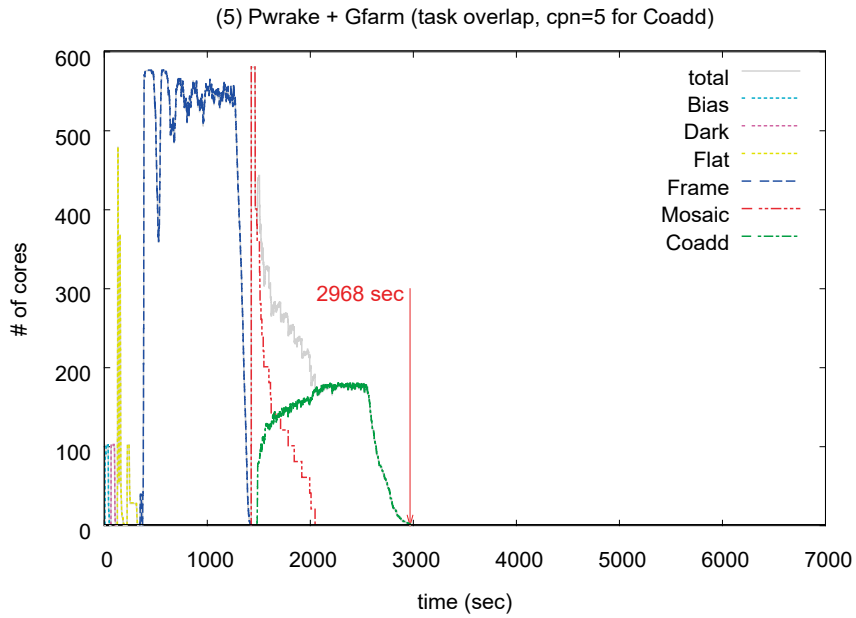


図 27 提案手法による実行の模様

図 27 に提案手法による実行の模様を示す。Frame 計算中の待ち時間がなくなり、またその他の計算がオーバラップして実行されていることにより、大幅に計算時間の短縮が可能となっている。データ解析の時間については、2.2 倍の高速化を実現した。この成果は IEEE International Conference on Cluster Computing において発表した。

【13】 分散ファイルシステム及びグリッド・クラウド技術に関する研究（建部）

文部科学省が進める革新的ハイパフォーマンスコンピューティングインフラ（HPCI）の HPCI 共用ストレージ、素粒子物理学データ共有システム JLDG のシステムソフトウェアとしても利用される Gfarm ファイルシステムの研究開発を行った。

本年度は、ファイル複製が正しく存在することをパトロールする時間の効率化、ファイルシステムノードを一時的に読込しかできないようにできるよう改善を行った。

Gfarm ファイルシステムには、ファイル複製の格納場所とその複製数を指定する機能がある。HPCI 共用ストレージでは、その機能を利用して東拠点に複製を 2、西拠点に複製を 2 配

置し、東拠点あるいは西拠点だけの片拠点しか利用できない場合も HPCI 共用ストレージの運用を可能としている。しかしながら、このような設定をしている時、とくにファイル数が多くなった時に、ファイル複製が正しく存在することをパトロールする時間が長くなってしまう問題があった。本パトロールでは、全ファイルについて、指定されたファイルシステムグループに指定された数の複製が配置されているかをチェックしている。この集合操作に対し、ビット演算を用いることにより処理速度の効率化を図った。これにより、これまで約 1 億ファイルのパトロールに対し、3 時間ほどかかっていたものが 5 分ほどで終了するようになった。

ファイルシステムノードの保守や障害発生時など、特定のファイルシステムノードを読込のみしかできないようにしたいという要求がある。これまでも，特定のファイルシステムノードに対し読込しかできなくする設定はあったが、全てのファイル複製が読込しかできないファイルシステムノードに格納されている時に、そのファイルの更新ができない問題があった。また、読込しかできなくしていても、ファイル削除はできてしまうため、厳密にストレージに変更を加えないという状態ではなかった。これらの問題を解決するため、全てのファイル複製が読込しかできないファイルシステムノードに格納されている場合は、そのファイルを変更するときに、更新可能なファイルシステムノードに動的にファイル複製を作成する機能の開発を行った。また、読込しかできない状態のストレージに変更を加えないようにするため、ファイル削除の遅延を行うようにした。これにより、読込のみ可能としたときもユーザが通常通りのアクセスが可能となり、また読込のみ可能としたストレージに変更を加えなくなり、状態の保全が可能となった。

これらの成果は平成 31 年 3 月 22 日にリリースした Gfarm バージョン 2.7.13 に含まれている。

【14】 ブロッククリロフ部分空間反復法の近似解精度向上に関する研究（多田野）

n 次行列 A と L 本の右辺ベクトルをもつ連立一次方程式

$$AX = B \tag{1}$$

は、素粒子物理学分野や固有値計算の内部問題などにおいて現れ、同方程式の求解時間、近似解精度はアプリケーションの性能に大きな影響を及ぼす。同方程式の数値解法の 1 つであるブロッククリロフ部分空間反復法は、計算時間、反復回数の観点において、ベクトルが 1 本用の数値解法であるクリロフ部分空間反復法よりも優れた性能を示すことがある。しかしながら近似解精度については、計算過程で生じる誤差の影響で精度劣化が生じることがある。我々はこれまでの研究において、高精度近似解の生成が可能なブロッククリロフ部分空間反復法として、Block BiCGGR 法を開発した。Block BiCGGR 法では Block BiCGSTAB 法で発生

する近似解精度劣化を回避できるが、右辺ベクトル数が多い場合は、収束性に難があることが明らかになった。

そこで平成 30 年度は、Block BiCGSTAB 法の収束性を維持しつつ、高精度近似解を生成できる手法の開発を行った。Block BiCGSTAB 法において、連立一次方程式(1)の第 $k+1$ 番目の近似解 X_{k+1}と対応する残差 $R_{k+1} = B - AX_{k+1}$ の漸化式は、$n \times L$ 補助行列 P_k , T_k を用いて以下のように表される。

$$X_{k+1} = X_k + P_k\alpha_k + \zeta_k T_k, \tag{2}$$

$$R_{k+1} = R_k - (AP_k)\alpha_k - \zeta_k(AT_k). \tag{3}$$

式 (2), (3) 中に現れる α_k は L 次行列、ζ_k はスカラーパラメータである。式 (3) の計算では、まず行列 AP_k , AT_k を行列 A に関する行列積で計算し、その後 α_k と ζ_k を乗じることで R_{k+1} が計算される。行列 AP_kと α_kの積の計算で生じる誤差が、近似解の精度劣化に大きな影響を及ぼす。近似解と残差の漸化式更新量の間には、$-A$ 倍の関係があるが、この関係が崩れてしまうことが原因である。そこで漸化式の再構築を行い誤差が発生しやすい計算を避けることで、近似解と残差の漸化式更新量の間の関係を極力保持することを考える。

本研究では、漸化式の複数回分の更新量をグループ化する。この手法は group-wise 更新手法と呼ばれる。近似解 X_{k+1}の漸化式 (2) の s 反復分の更新量をグループ化し、以下のように変形する。但し、反復番号 k を $k = ms + j$ と表す。ここで、$m \equiv \lfloor k/s \rfloor$, $0 \le j \le s-1$である。

$$X_{ms+j+1} = X_0 + \sum_{l=0}^{m-1} U_s^{(l,s)} + U_{j+1}^{(m,s)} = X_{ms} + U_{j+1}^{(m,s)}.$$

ここで、行列 $U_q^{(l,s)}$ は以下で定義される。

$$\boldsymbol{U_q^{(l,s)}} \equiv \sum_{i=0}^{q-1} (\boldsymbol{P_{ls+i}\alpha_{ls+i} + \zeta_{ls+i}T_{ls+i}}).$$

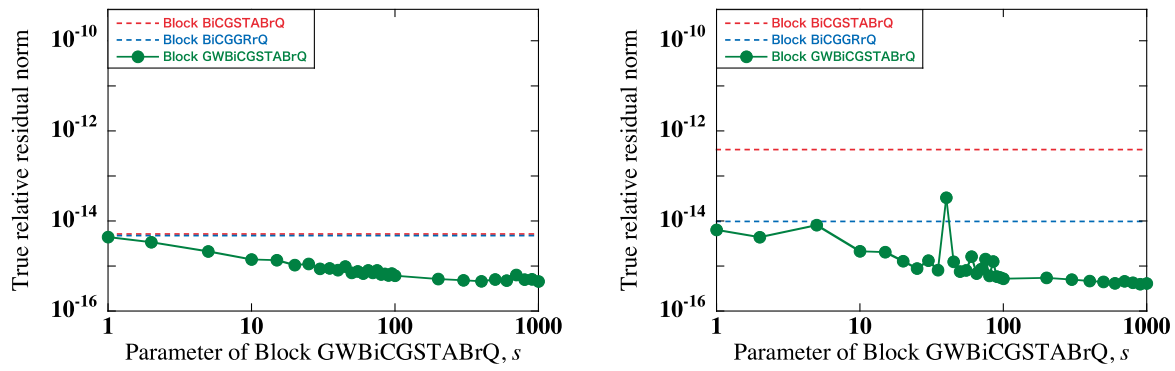
近似解 X_{ms+j+1} に対する残差 R_{ms+j+1} は、以下の漸化式で計算する。

$$R_{ms+j+1} = R_{ms} - AU_{j+1}^{(m,s)}.$$

この方法では、漸化式の複数回分の更新量に対してまとめて係数行列 A を乗じているため、近似解と残差の漸化式更新量の関係を保つことができると期待される。Block BiCGSTAB 法に対して、漸化式の group-wise 更新手法を組み込んだ手法を Block GWBiCGSTAB 法と名付けた。ブロッククリロフ部分空間反復法は、一般的に右辺ベクトル数が多いと数値的不安定性の影響で、残差の収束性が悪化する。反復毎に残差を構成するベクトルの正規直交化を行うことで、この数値的不安定性が緩和されることが知られている。本研究では、Block GWBiCGSTAB 法に残差の正規直交化を組み込んだ Block GWBiCGSTABrQ 法も併せて構築した。

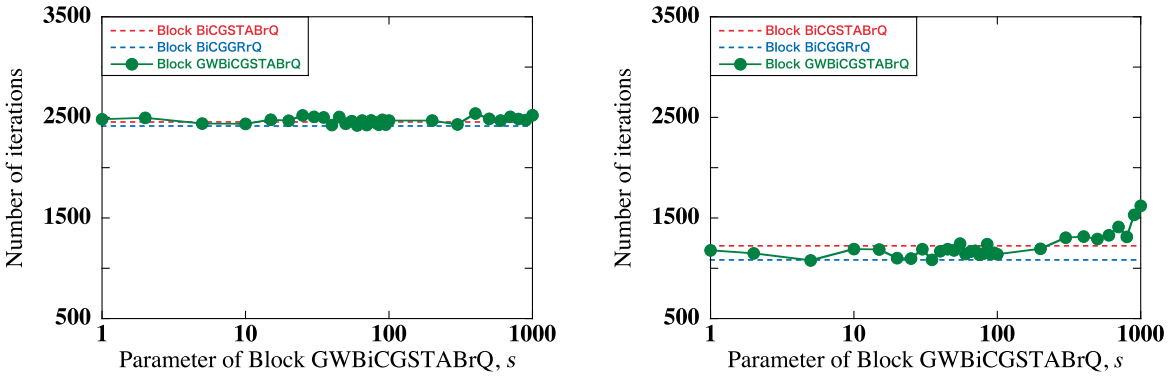
数値実験において、Block GWBiCGSTABrQ 法、Block BiCGSTABrQ 法、Block BiCGGRrQ 法の 3 解法について、真の相対残差、反復回数、右辺ベクトル 1 本あたりの求解時間の観点で比較を行う。真の相対残差は $\|B - AX_k\|_F / \|B\|_F$ で定義され、近似解精度の指標となる値である。この値が十分に小さければ、高精度の近似解が得られていることを示す。テスト問題として、格子量子色力学計算（QCD）で現れる連立一次方程式を用いた。同方程式のサイズ n は 1,572,864、非零要素数は 80,216,064 である。係数行列内に現れるパラメータ κ は 0.1359 とした。右辺項 B は $B = [e_1, e_2, \ldots, e_L]$ とした。但し、e_j $(j = 1, 2, \ldots, L)$ は n 次元基本ベクトルである。実験環境は、CPU：Intel Xeon E5-2620v3 2.4GHz (6 cores)×2、メモリ：64GiB DDR4 2133MHz、コンパイラ : Intel Fortran ver. 19.0.0、コンパイルオプション : -qopenmp -axCore-AVX2 であり、OpenMP を用いて 12 スレッド並列で計算した。

図 27〜29 に、Block GWBiCGSTABrQ 法のパラメータ s を変化させたときの真の相対残差、反復回数、右辺ベクトル 1 本あたりの計算時間の変化をそれぞれ示す。右辺ベクトル数 L は 1, 4 とした。図 27(a)より、 $L = 1$ の場合は Block BiCGGRrQ 法、Block BiCGSTABrQ 法も真の相対残差が 10^{-14} を下回っており高精度の近似解が得られているが、Block GWBiCGSTABrQ 法で s を大きくしていくと、従来法以上の近似解精度が得られている。また、図 27(b)に示すように、右辺ベクトル数が増えると Block BiCGSTABrQ 法の近似解精度は誤差の影響で劣化しているが、Block GWBiCGSTABrQ 法では Block BiCGGRrQ 法よりも高精度の近似解が得られている。図 28 より、Block GWBiCGSTABrQ 法の反復回数は、従来法とほぼ同程度であることが分かる。しかしながら、パラメータ s を大きくし過ぎると、反復回数が徐々に増加する傾向にあることも明らかになった。図 29 に示すように、右辺ベクトル 1 本あたりの計算時間に関しても Block GWBiCGSTABrQ 法は従来法と同程度であった。しかしながら、パラメータ s を大きくすると反復回数が増加する傾向にあり、その影響で計算時間も徐々に増加している。



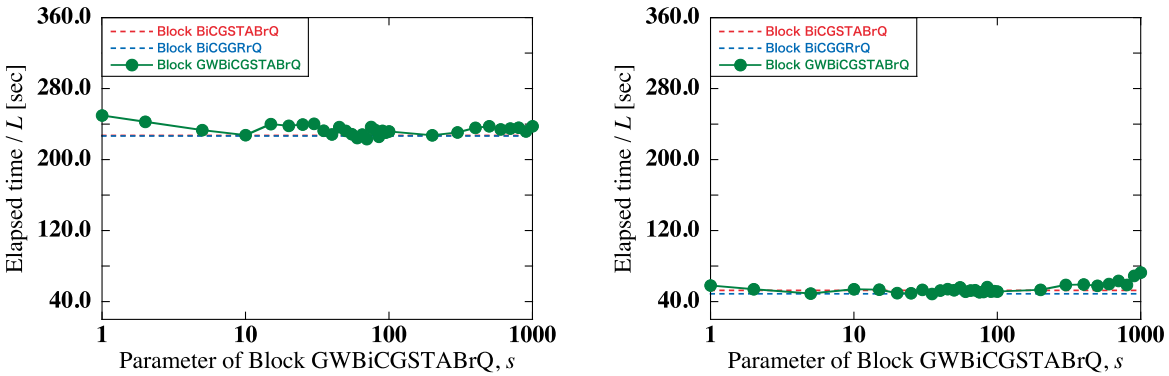
(a) 右辺ベクトル数 $L = 1$ の場合　　(b) 右辺ベクトル数 $L = 4$ の場合

図 27　パラメータ s が真の相対残差に与える影響



(a) 右辺ベクトル数 $L = 1$ の場合　　(b) 右辺ベクトル数 $L = 4$ の場合

図 28　パラメータ s が反復回数に与える影響

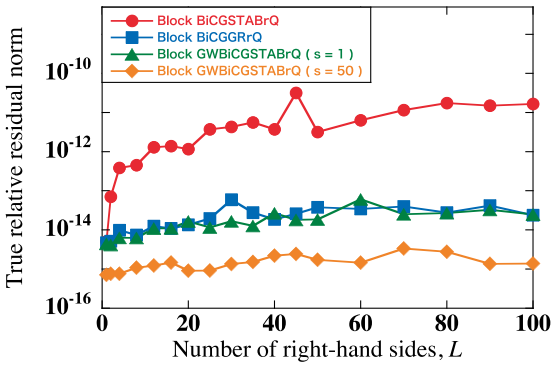


(a) 右辺ベクトル数 $L = 1$ の場合　　(b) 右辺ベクトル数 $L = 4$ の場合

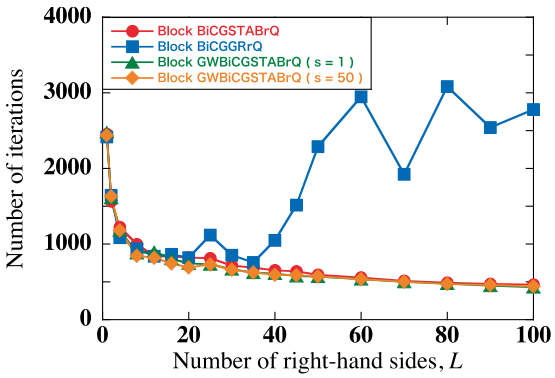
図 29　パラメータ s が右辺ベクトル 1 本あたりの計算時間に与える影響

次に、右辺ベクトル数 L を変化させたときの真の相対残差、反復回数、右辺ベクトル 1 本あたりの求解時間を評価する。図 30 に、右辺ベクトル数 L を変化させたときの真の相対残差、反復回数、右辺ベクトル 1 本あたりの求解時間の変化を示す。図 30(a)に示すように、Block BiCGSTABrQ 法では右辺ベクトル数の増加に伴い真の相対残差が増加しており、近似解の精度劣化が発生している。Block BiCGGRrQ 法と Block GWBiCGSTABrQ 法($s = 1$)の真の相対残差は同程度であった。一方、Block GWBiCGSTABrQ 法($s = 50$)では、真の相対残差が 10^{-15} 付近まで減少しており、非常に高精度な近似解を生成できている。また、図 30(b), (c)に示すように、Block BiCGGRrQ 法では右辺ベクトル数の増加に伴い収束性が悪化しており、反復回数、求解時間が増加している。一方、Block GWBiCGSTABrQ 法は Block BiCGSTABrQ 法と同程度の性能を示しており、近似解精度と収束性の両面で優れていることが分かった。

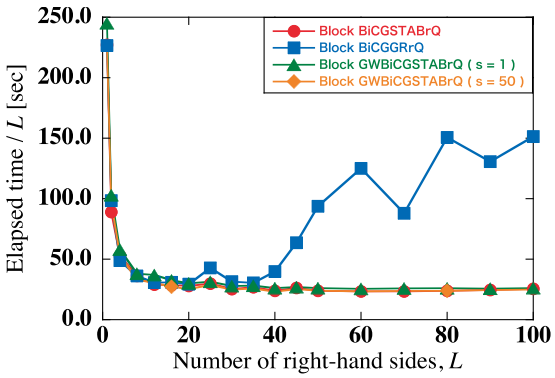
現時点では、Block GWBiCGSTABrQ 法のパラメータ s はユーザが事前に設定する必要があるが、漸化式のグループ化を動的に行えるようになれば、より使いやすい解法になることが期待される。これは次年度以降の今後の課題である。



(a) 真の相対残差の変化



(b) 反復回数の変化



(c) 右辺ベクトル 1 本あたりの求解時間

図 30　右辺ベクトル数 L の変化が真の相対残差、反復回数右辺ベクトル 1 本あたりの求解時間に与える影響

4. 教育

1. 廣川祐太，博士（工学），最先端高性能計算システムにおける第一原理電子動力学シミュレーションの子デザイン，筑波大学システム情報工学研究科博士論文，平成 30 年 9 月（指導：朴泰祐）
2. 西村慧，修士(工学)，高性能 FPGA システムのメモリチャネルを有効利用するためのホスト FPGA 連携システム，筑波大学システム情報工学研究科修士論文，平成 31 年 3 月(指導:朴泰祐)
3. 岩井厚樹，修士（工学），大規模環境における IO の性能評価に関する研究，筑波大学大学院システム情報工学研究科修士論文，平成 31 年 3 月（指導：建部修見）

4. 小林淳司，修士（工学），シミュレータを用いた分散メタデータサーバ PPMDS の性能評価，筑波大学大学院システム情報工学研究科修士論文，平成 31 年 3 月（指導：建部修見）
5. 梶原顕伍，修士（工学），合意手法 Raft を用いた分散 KVS の高性能化，筑波大学大学院システム情報工学研究科修士論文，平成 31 年 3 月（指導：建部修見）
6. 中村泰大，修士（工学），メニーコアマシンにおけるトランザクション処理の高性能化，筑波大学大学院システム情報工学研究科修士論文，平成 31 年 3 月（指導：建部修見）
7. 渡邉敬之，修士（工学），並行実行木 Masstree の高性能構築手法に関する研究，筑波大学大学院システム情報工学研究科修士論文，平成 31 年 3 月（指導：建部修見）
8. 小田健斗，学士（工学），メニーコアプロセッサにおける冪剰余計算の性能評価，筑波大学情報学群情報科学類卒業論文、平成 31 年 3 月（指導：高橋大介）
9. 諸戸雄治，学士（工学），微分カーネルを用いた 2 次元畳み込み積分の高速化，筑波大学情報学群情報科学類卒業論文、平成 31 年 3 月（指導：高橋大介）
10. 杉原航平, 学士（工学）, ノードローカルバーストバッファのための並列 IO の設計，筑波大学情報学群情報科学類卒業論文，平成 31 年 3 月（指導：建部修見）
11. 高橋宗史，学士（工学），高性能計算に向けた分散オブジェクトストレージ Ceph の性能評価，筑波大学情報学群情報科学類卒業論文，平成 31 年 3 月（指導：建部修見）
12. 横田健太，学士（工学），Hadoop における Erasure Coding の性能評価，筑波大学情報学群情報科学類卒業論文，平成 31 年 3 月（指導：建部修見）
13. 畑中智之，学士（工学），コンテナ型仮想化における NVMe-oF の性能評価筑波大学情報学群情報科学類卒業論文，平成 31 年 3 月（指導：建部修見）

5. 受賞、外部資金、知的財産権等

受賞

1. Best Paper Award, Kohei Hiraga, Osamu Tatebe, Hideyuki Kawashima, “PPMDS : A Distributed Metadata Server Based on Nonblocking Transactions”, The Fifth International Conference on Social Networks Analysis, Management and Security, Valencia, Spain, October 17, 2018
2. Workshop Best Paper, Yasuhiro Nakamura, Hideyuki Kawashima, Osamu Tatebe, “Integrating TicToc with Parallel Logging”, 6th International Workshop on Computer Systems and Architectures, Gifu, Japan, November 29, 2018

外部資金

1. 文部科学省高性能汎用計算機高度利用事業費補助金　朴泰祐（代表）, H29～33 年度, 23,400 千円（H30 年度），「次世代演算通信融合型スーパーコンピュータの開発」
2. 科学研究費補助金　基盤研究 (B), 朴泰祐（代表），H30～32 年度，6,110 千円（H30 年度），「再構成可能システムと GPU による複合型高性能計算プラットフォーム」
3. 理化学研究所受託研究　朴泰祐（代表）, H27～H32 年度, 6,600 千円（H30 年度），「ポスト京の並列プログラミング環境およびネットワークに関する研究」
4. 科学研究費補助金　基盤研究 (C), 高橋大介（代表）, H28～30 年度, 1,300 千円（H30 年度）,「メニーコア超並列クラスタにおける有理数演算ライブラリに関する研究」
5. JST CREST，建部修見，主たる共同研究者，H25～H30 年度, 7,150 千円，EBD：次世代の年ヨッタバイト処理に向けたエクストリームビッグデータの基盤技術
6. 科学研究費補助金　基盤研究(B)（一般），建部修見，代表，H29～H31 年度，5,070 千円，極端気象予測を拓くビッグデータ機械学習基盤の研究
7. NEDO，建部修見，分担，H30～H32 年度，7,345 千円，実社会の事象をリアルタイム処理可能な次世代データ処理基盤技術の研究開発
8. 共同研究（富士通研究所），建部修見，代表，H30 年度，3,564 千円，大量データ管理に向けたストレージシステムに関する研究

知的財産権

（該当なし）

6. 研究業績

(1) 研究論文

A) 査読付き論文

1. Miwako Tsuji, Taisuke Boku, Mitsuhisa Sato, “Scalable Communication Performance Prediction Using Auto-Generated Pseudo MPI Event Trace.”, Proc. of HPC Asia 2019, Guangzhou, Jan. 15th, 2019.
2. Yuta Hirokawa, Taisuke Boku, Mitsuharu Uemoto, Shunsuke A. Sato, and Kazuhiro Yabana: “Performance Optimization and Evaluation of Scalable Optoelectronics Application on Large Scale KNL Cluster”, ISC High Performance 2018, Frankfurt, June 2018, https://doi.org/10.1007/978-3-319-92040-5_11
3. Norihisa Fujita, Ryohei Kobayashi, Yoshiki Yamaguchi, Yuuma Oobata, Taisuke Boku, Makito Abe, Kohji Yoshikawa, and Masayuki Umemura: Accelerating Space Radiate Transfer

on FPGA using OpenCL, International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART 2018) pp.6:1-6:7 (June 2018)

4. Yutaka Watanabe, Jinpil Lee, Taisuke Boku and Mitsuhisa Sato, "Trade-off of offloading to FPGA in OpenMP Task-based programming", Proc. of Int. Workshop on OpenMP 2018 (IWOMP2018), 12 pages, Barcelona, Sep. 2018.
5. Takahiro Katagiri and Daisuke Takahashi, “Japanese Autotuning Research: Autotuning Languages and FFT”, Proceedings of the IEEE, Vol. 106, No. 11, pp. 2056-2067, 2018. (invited paper)
6. Daisuke Takahashi, “Computation of the 100 quadrillionth hexadecimal digit of π on a cluster of Intel Xeon Phi processors”, Parallel Computing, Vol. 75, pp. 1-10, 2018.
7. Takuya Edamatsu and Daisuke Takahashi, “Acceleration of Large Integer Multiplication with Intel AVX-512 Instructions”, Proc. 20th IEEE International Conference on High Performance Computing and Communications (HPCC-2018), pp. 211-218, 2018.
8. Takayuki Tanabe, Hideyuki Kawashima, and Osamu Tatebe, “Integration of Parallel Write Ahead Logging and Cicada Concurrency Control Method”, Proceedings of 2nd IEEE International Workshop on Big Data and IoT Security in Smart Computing (BITS), pp.291-296, 2018
9. Masahiro Tanaka, Osamu Tatebe, and Hideyuki Kawashima, “Applying Pwrake Workflow System and Gfarm File System to Telescope Data Processing”, Proceedings of 2018 IEEE International Conference on Cluster Computing, pp.124-133, 2018
10. Kohei Hiraga, Osamu Tatebe, and Hideyuki Kawashima, “PPMDS: A Distributed Metadata Server Based on Nonblocking Transactions”, Proceedings of The Second International Workshop on Data Science Engineering and its Applications, pp.202-208, 2018
11. Yasuhiro Nakamura, Hideyuki Kawashima, and Osamu Tatebe, "Integrating TicToc with Parallel Logging", Proceedings of the 6th International Workshop on Computer Systems and Architectures, pp.105-111, 2018
12. Harunobu Daikoku, Hideyuki Kawashima, and Osamu Tatebe, "Skew-Aware Collective Communication for MapReduce Shuffling", Proceedings of the 6th Workshop on Scalable Cloud Data Management, pp.3331-3340, 2018
13. Hiroto Tadano, Ryosei Kuramoto, Accuracy improvement of the Block BiCGSTAB method for linear systems with multiple right-hand sides by group-wise updating technique, Journal of Advanced Simulation in Science and Engineering, Vol. 6, No. 1, pp. 100—117, 2019.

14. Ryohei Kobayashi, Norihisa Fujita, Yoshiki Yamaguchi, Taisuke Boku: OpenCL-enabled High Performance Direct Memory Access for GPU-FPGA Cooperative Computation (short paper), Proceedings of Workshops of HPC Asia 2019, 4 pages, (January 2019)

B) 査読無し論文

1. 中道 安祐未, 小林 諒平, 藤田 典久, 朴 泰祐: GPU・FPGA 混載ノードにおけるヘテロ演算加速プログラム環境に関する研究, 研究報告ハイパフォーマンスコンピューティング（HPC）/2019-HPC-168(10)/pp.1-7 (March 2019)
2. 小林 諒平, 藤田 典久, 山口 佳樹, 朴 泰祐: 異デバイス間での PCIe 通信を実現する OpenCL 対応 FPGA モジュールの提案と検証, IEICE-RECONF2018-63/IEICE-118(432)/pp.107-112 (January 2019)
3. 藤田 典久, 小林 諒平, 山口 佳樹, 朴 泰祐: OpenCL による FPGA 上の演算と通信を融合した並列処理システムの実装及び性能評価, 研究報告ハイパフォーマンスコンピューティング（HPC）/2018-HPC-167(9)/pp.1-9 (December 2018)
4. 小林 諒平, 藤田 典久, 山口 佳樹, 朴 泰祐: OpenCL と Verilog HDL の混合記述による GPU-FPGA デバイス間連携, 研究報告ハイパフォーマンスコンピューティング（HPC）/2018-HPC-167(11)/pp.1-10 (December 2018)
5. 横野 智也, 藤田 典久, 山口 佳樹, 大畠 佑真, 小林 諒平, 朴 泰祐, 吉川 耕司, 安部 牧人, 梅村 雅之: FPGA による宇宙輻射輸送シミュレーションの演算加速, IEICE-RECONF2018-25/118(215)/pp.35-40 (September 2018)
6. 藤田 典久, 小林 諒平, 山口 佳樹, 朴 泰祐, 吉川 耕司, 安部 牧人, 梅村 雅之: 並列 FPGA システムにおける OpenCL を用いた宇宙輻射輸送コードの演算加速, 研究報告ハイパフォーマンスコンピューティング（HPC）/2018-HPC-165(27)/pp.1-8 (July 2018)
7. 廣川 祐太, 朴 泰祐, 矢花 一浩, "AVX-512 Intrinsics で実装されたステンシル計算の Scalable Vector Extension への展開", 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), 2019-HPC-168(4), pp.1-7, 2019 年 2.
8. 辻 美和子, 村井 均, 佐藤 三久, 朴 泰祐, Serge Petiton, Nahid Emad, Joachim Protze, Christian Terboven, Matthias S. Müeller, "MYX：マルチ SPMD プログラミングモデルにおける実行時正当性チェック", 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), 2019-HPC-168(6), pp.1-7, 2019 年 2 月.
9. 辻 大亮, 多田野 寛人, 朴 泰祐, 池田 亮作, 佐藤 拓人, 日下 博幸, "都市気象 LES コードの並列 GPU 環境における高速化", 研究報告ハイパフォーマンスコンピューティング(HPC), 2019-HPC-168(8), pp.1-7, 2019 年 2 月.

10. 小林 諒平, 阿部 昂之, 藤田 典久, 山口 佳樹 朴 泰祐: GPU-FPGA 複合システムにおけるデバイス間連携機構, 研究報告ハイパフォーマンスコンピューティング（HPC）/2018-HPC-165(26)/pp.1-8 (July 2018)
11. 小林 諒平, 藤田 典久, 大畠 佑真, 山口 佳樹 朴 泰祐: 複数の FPGA による分散ソーティングの実現に向けた予備評価, 電子情報通信学会技術研究報告 : 信学技報/118(63)/pp.65-70 (May 2018)
12. 佐藤 駿一，高橋 大介，“GPU における SELL 形式疎行列ベクトル積の性能評価”，日本応用数理学会 2018 年度年会講演予稿集, 2 pages, 2018.
13. 高橋 大介，“Intel AVX-512 命令を用いた複数の整数除算の高速化”，日本応用数理学会 2018 年度年会講演予稿集, 2 pages, 2018.
14. 佐藤 駿一，高橋 大介，“GPU における SELL 形式疎行列ベクトル積の実装と性能評価”，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），2018-HPC-164(3), 6 pages, 2018.
15. 堀江 悠樹, 川島 英之, 建部 修見, 耐ビザンチン障害性を持つ分散合意手法の調査, 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（OS）, 2018-OS-143(15), 5 pages, 2018 年 5 月
16. 岩井 厚樹, 建部 修見, サイエンティフィックビッグデータアプリケーションのためのベンチマークセットの提案, 研究報告ハイパフォーマンスコンピューティング（HPC）, 2018-HPC-165(4), 5 pages, 2018 年 7 月
17. 畑中 智之，建部 修見，コンテナ型仮想化における NVMe-oF の性能評価, 研究報告ハイパフォーマンスコンピューティング（HPC）, 2018-HPC-167(13), 7 pages, 2018 年 12 月
18. 高橋 宗史，建部 修見, 高性能計算に向けた分散オブジェクトストレージ Ceph の性能評価, 研究報告ハイパフォーマンスコンピューティング（HPC）, 2018-HPC-167(15), 8 pages, 2018 年 12 月
19. 杉原 航平, 建部 修見, ノードローカルバーストバッファのための MPI-IO の設計, 研究報告ハイパフォーマンスコンピューティング（HPC）, 2019-HPC-168(22), 7 pages, 2019 年 3 月
20. 芹沢 和洋, 建部 修見, 深層ニューラルネットワークにおける訓練高速化のための自動最適化, 研究報告ハイパフォーマンスコンピューティング（HPC）, 2019-HPC-168(25), 10 pages, 2019 年 3 月
21. 渡部 裕, 李 珍泌, 佐野 健太郎, 朴 泰祐, 佐藤 三久, "FPGA へのオフロード最適化のための SPGen と OpenCL の統合の検討", 研究報告ハイパフォーマンスコンピューティング(HPC), 2019-HPC-168(11), pp.1-10, 2019 年 2 月.

22. 渡部 裕, 李 珍泌, 朴 泰祐, 佐藤 三久, "OpenMP タスク並列実行における FPGA オフローディングの性能最適化", 研究報告ハイパフォーマンスコンピューティング (HPC), 2018-HPC-165(25), pp.1-7, 2018 年 7 月.

(2) 国際会議発表

A) 招待講演

1. Taisuke Boku, "Oakforest-PACS: Japan's Fastest Intel Xeon Phi Supercomputer and Its Applications", IXPUG Conference Middle East, Jeddha, Saudi Arabia, Apr. 24, 2018.
2. Taisuke Boku, "Oakforest-PACS: World Largest KNL/OPA Cluster and Its Performance Optimization", IPCC Asia Summit 2018, Chengdu, China, May 10, 2018.
3. Taisuke Boku, "Hybrid Accelerated Parallel Computing, GPU+FPGA=?", CODESIGN Workshop 2018 (at HPC China 2018), Qingdao, China, Oct. 19, 2018.
4. Taisuke Boku, "Overview and Scientific Results of Oakforest-PACS", KNL Workshop in Korea, Daejon, South Korea, Nov. 6, 2018.
5. Taisuke Boku, "What's the Next Step of Accelerated Supercomputing?", Supercomputing Frontier Europ 2019, Warsaw, Poland, Mar. 13, 2019.
6. Taisuke Boku, "MUST system applied to high level language approach in MYX project", SPPEXA Workshop 2019, Versilles, France, Mar. 21, 2019.
7. Daisuke Takahashi, “Automatic Tuning for Parallel FFTs on Cluster of Intel Xeon Phi Processors”, Parallel Fast Fourier Transforms (PFFT) (held in conjunction with IEEE HiPC 2018), Radisson Blu Bengaluru Outer Ring Road, Bengaluru, India, December 17, 2018.

B) 一般講演

1. Yuta Hirokawa, Taisuke Boku, Mitsuharu Uemoto, Shunsuke A. Sato, and Kazuhiro Yabana: “Performance Optimization and Evaluation of Scalable Optoelectronics Application on Large Scale KNL Cluster”, ISC High Performance 2018, Frankfurt, June 2018
2. Ryohei Kobayashi, Norihisa Fujita, Yoshiki Yamaguchi, Taisuke Boku: OpenCL-enabled High Performance Direct Memory Access for GPU-FPGA Cooperative Computation, IXPUG Workshop at HPC Asia 2019, January 2019
3. Norihisa Fujita, Ryohei Kobayashi, Yoshiki Yamaguchi, Yuuma Oobata, Taisuke Boku, Makito Abe, Kohji Yoshikawa, and Masayuki Umemura: Accelerating Space Radiate Transfer on FPGA using OpenCL, International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART 2018), June 2018

4. Daisuke Takahashi, “Implementation of Parallel 3-D Real FFT with 2-D Decomposition on Intel Xeon Phi Clusters”, SIAM Conference on Computational Science and Engineering (CSE19), Spokane Convention Center, Spokane, Washington, USA, March 1, 2019.
5. Takuya Edamatsu and Daisuke Takahashi, “Acceleration of Large Integer Multiplication with Intel AVX-512 Instructions”, The 20th IEEE International Conference on High Performance Computing and Communications (HPCC-2018), Mercure Exeter Rougemont Hotel, Exeter, United Kingdom, June 28, 2018.
6. Takayuki Tanabe, Hideyuki Kawashima, and Osamu Tatebe, “Integration of Parallel Write Ahead Logging and Cicada Concurrency Control Method”, 2nd IEEE International Workshop on Big Data and IoT Security in Smart Computing (BITS), Sicily, Italy, June 18, 2018
7. Masahiro Tanaka, Osamu Tatebe, and Hideyuki Kawashima, “Applying Pwrake Workflow System and Gfarm File System to Telescope Data Processing”, 2018 IEEE International Conference on Cluster Computing, Belfast, North Ireland, September 11, 2018
8. Kohei Hiraga, Osamu Tatebe, and Hideyuki Kawashima, “PPMDS: A Distributed Metadata Server Based on Nonblocking Transactions”, the Second International Workshop on Data Science Engineering and its Applications, Valencia, Spain, October 15, 2018
9. Yasuhiro Nakamura, Hideyuki Kawashima, and Osamu Tatebe, "Integrating TicToc with Parallel Logging", the 6th International Workshop on Computer Systems and Architectures, Takayama, November 29, 2018
10. Osamu Tatebe, “Memory-Storage Hierarchy”, Big Data and Exascale-Computing 2, Bloomington, IN, USA, November 30, 2018
11. Harunobu Daikoku, Hideyuki Kawashima, and Osamu Tatebe, "Skew-Aware Collective Communication for MapReduce Shuffling", the 6th Workshop on Scalable Cloud Data Management, Seattle, USA, December 10, 2018
12. Osamu Tatebe, “Gfarm/BB – Gfarm file system for burst buffers”, CCS - LBNL Collaborative Workshop, Berkeley, CA, USA, March 7, 2019
13. Hiroto Tadano, Ryosei Kuramoto, An improvement of the Block BiCGSTAB method by reconstructing recursions, The 37th JSST Annual International Conference on Simulation Technology (JSST2018), Hokkaido, Japan, Sep. 18, 2018.

(3) 国内学会・研究会発表

A) 招待講演

（該当なし）

B) その他の発表

1. 中道 安祐未, 小林 諒平, 藤田 典久, 朴 泰祐: GPU・FPGA 混載ノードにおけるヘテロ演算加速プログラム環境に関する研究, 第 168 回情報処理学会ハイパフォーマンスコンピューティング研究会, 2019 年 3 月
2. 小林 諒平, 藤田 典久, 山口 佳樹, 朴 泰祐: 異デバイス間での PCIe 通信を実現する OpenCL 対応 FPGA モジュールの提案と検証, 電子情報通信学会リコンフィギャラブルシステム研究会, 2019 年 1 月
3. 藤田 典久, 小林 諒平, 山口 佳樹, 朴 泰祐: OpenCL による FPGA 上の演算と通信を融合した並列処理システムの実装及び性能評価, 第 167 回情報処理学会ハイパフォーマンスコンピューティング研究会, 2018 年 12 月
4. 小林 諒平, 藤田 典久, 山口 佳樹, 朴 泰祐: OpenCL と Verilog HDL の混合記述による GPU-FPGA デバイス間連携, 第 167 回情報処理学会ハイパフォーマンスコンピューティング研究会, 2018 年 12 月
5. 廣川 祐太, 朴 泰祐, 矢花 一浩, "AVX-512 Intrinsics で実装されたステンシル計算の Scalable Vector Extension への展開", 第 168 回情報処理学会ハイパフォーマンスコンピューティング研究会, 2019 年 2 月 26 日.
6. 辻 美和子, 村井 均, 佐藤 三久, 朴 泰祐, Serge Petiton, Nahid Emad, Joachim Protze, Christian Terboven, Matthias S. Müeller, "MYX：マルチ SPMD プログラミングモデルにおける実行時正当性チェック", 第 168 回情報処理学会ハイパフォーマンスコンピューティング研究会, 2019 年 2 月 26 日.
7. 辻 大亮, 多田野 寛人, 朴 泰祐, 池田 亮作, 佐藤 拓人, 日下 博幸, "都市気象 LES コードの並列 GPU 環境における高速化", 第 168 回情報処理学会ハイパフォーマンスコンピューティング研究会, 2019 年 2 月 26 日..
8. 横野 智也, 藤田 典久, 山口 佳樹, 大畠 佑真, 小林 諒平, 朴 泰祐, 吉川 耕司, 安部 牧人, 梅村 雅之: FPGA による宇宙輻射輸送シミュレーションの演算加速, 電子情報通信学会リコンフィギャラブルシステム研究会, 2018 年 9 月
9. 藤田 典久, 小林 諒平, 山口 佳樹, 朴 泰祐, 吉川 耕司, 安部 牧人, 梅村 雅之: 並列 FPGA システムにおける OpenCL を用いた宇宙輻射輸送コードの演算加速, 第 165 回情報処理学会ハイパフォーマンスコンピューティング研究会, 2018 年 7 月
10. 小林 諒平, 阿部 昂之, 藤田 典久, 山口 佳樹 朴 泰祐: GPU-FPGA 複合システムにおけるデバイス間連携機構, 第 165 回情報処理学会ハイパフォーマンスコンピューティング研究会, 2018 年 7 月

11. 小林 諒平, 藤田 典久, 大畠 佑真, 山口 佳樹 朴 泰祐: 複数の FPGA による分散ソーティングの実現に向けた予備評価, リコンフィギャラブルシステム研究会, 2018 年 5 月
12. 佐藤 駿一，高橋 大介，“GPU における SELL 形式疎行列ベクトル積の性能評価”，日本応用数理学会 2018 年度年会，名古屋，2018 年 9 月 5 日.
13. 高橋 大介，“Intel AVX-512 命令を用いた複数の整数除算の高速化”，日本応用数理学会 2018 年度年会，名古屋，2018 年 9 月 4 日.
14. 佐藤 駿一，高橋 大介，“GPU における SELL 形式疎行列ベクトル積の実装と性能評価”，情報処理学会第 164 回ハイパフォーマンスコンピューティング研究会，東京，2018 年 5 月 7 日.
15. 堀江 悠樹, 川島 英之, 建部 修見, 耐ビザンチン障害性を持つ分散合意手法の調査, 第 143 回システムソフトウェアとオペレーティング・システム研究発表会，沖縄，2018 年 5 月 22 日
16. 岩井 厚樹，建部 修見，サイエンティフィックビッグデータアプリケーションのためのベンチマークセットの提案, 第 165 回ハイパフォーマンスコンピューティング研究発表会，熊本，2018 年 7 月 30 日
17. 建部 修見，Gfarm ファイルシステムの概要と最新機能，Gfarm シンポジウム，東京，2018 年 10 月 26 日
18. 畑中 智之，建部 修見，コンテナ型仮想化における NVMe-oF の性能評価，第 167 回ハイパフォーマンスコンピューティング研究発表会，沖縄，2018 年 12 月 17 日
19. 高橋 宗史，建部 修見，高性能計算に向けた分散オブジェクトストレージ Ceph の性能評価，第 167 回ハイパフォーマンスコンピューティング研究発表会，沖縄，2018 年 12 月 17 日
20. 建部 修見，Gfarm ファイルシステムの概要と最新機能，Gfarm ワークショップ，鹿児島，2019 年 2 月 1 日
21. 杉原 航平, 建部 修見, ノードローカルバーストバッファのための MPI-IO の設計, 第 168 回ハイパフォーマンスコンピューティング研究発表会, 石川, 2019 年 3 月 7 日
22. 芹沢 和洋, 建部 修見, 深層ニューラルネットワークにおける訓練高速化のための自動最適化, 第 168 回ハイパフォーマンスコンピューティング研究発表会，石川，2019 年 3 月 7 日
23. 多田野 寛人, ”精度混合型前処理付きクリロフ部分空間反復法”, 研究会「データ駆動科学と高速計算科学」，東京大学駒場キャンパス，2018 年 7 月 17 日.

24. 多田野 寛人，”漸化式の再構築による Block BiCGSTAB 法の近似解精度改善”，2018 年並列／分散／協調処理に関する『熊本』サマー・ワークショップ（SWoPP2018），熊本市国際交流会館，2018 年 7 月 31 日.
25. 多田野 寛人，倉本 亮世，”Block BiCGSTAB 法の近似解高精度化と数値的安定化”，日本応用数理学会 2018 年度年会，名古屋大学東山キャンパス，2018 年 9 月 4 日.
26. 倉本 亮世，多田野 寛人，”Shifted Block BiCGStab 法の近似解精度劣化の数値的原因解析”，日本応用数理学会 2018 年度年会（ポスターセッション），名古屋大学東山キャンパス，2018 年 9 月 4 日.
27. 齋藤 歩，平吹 勇司，多田野 寛人，高山 彰優，神谷 淳，”断層画像データから 3D ボリューメトリックモデルの再構成：Global ICCG 法による高速化”，【非線形問題の解法と可視化に関する研究会】2018 年度第 2 回研究会，自然科学研究機構核融合科学研究所，2019 年 1 月 24 日.
28. 渡部 裕, 李 珍泌, 佐野 健太郎, 朴 泰祐, 佐藤 三久, "FPGA へのオフロード最適化のための SPGen と OpenCL の統合の検討", 第 168 回情報処理学会ハイパフォーマンスコンピューティング研究会, 2019 年 2 月 26 日.
29. 渡部 裕, 李 珍泌, 朴 泰祐, 佐藤 三久, "OpenMP タスク並列実行における FPGA オフローディングの性能最適化", 第 166 回情報処理学会ハイパフォーマンスコンピューティング研究会, 2018 年 7 月 23 日.

(4) 著書、解説記事等

A） 多田野 寛人，数値線形代数の数理と HPC，第 1 章 1.2.1 定常反復法，1.2.2 クリロフ部分空間反復法，櫻井 鉄也，松尾 宇泰，片桐 孝洋（編），共立出版，2018.

7. 異分野間連携・国際連携・国際活動等

1. 日独仏 3 カ国国際共同研究 SPPEXA, 朴 泰祐（日本代表 PI）, "MYX: MUST correctness checking for YML and XMP programs", JST

8. シンポジウム、研究会、スクール等の開催実績

1. XcalableMP ワークショップ，つくば, 2018 年 11 月 1 日
2. Korea-Japan HPC Winter School，ソウル, 2019 年 2 月 20 日～21 日
3. Gfarm シンポジウム，東京，2018 年 10 月 26 日
4. Gfarm ワークショップ，鹿児島，2019 年 2 月 1 日

9. 管理・運営

組織運営や支援業務の委員・役員の実績

1. 朴 泰祐：理化学研究所客員主管研究員
2. 朴 泰祐：HPCI 連携サービス委員会委員長
3. 朴 泰祐：HPCI コンソーシアム理事
4. 朴 泰祐：PC クラスタコンソーシアム理事
5. 朴 泰祐：学際大規模情報基盤共同利用・共同研究拠点（JHPCN）運営委員
6. 朴 泰祐：筑波大学情報環境企画室委員
7. 朴 泰祐：筑波大学ネットワーク管理委員会委員
8. 高橋 大介：筑波大学情報環境機構学術情報メディアセンター運営委員会委員
9. 高橋 大介：理化学研究所客員主管研究員
10. 高橋 大介：HPCI 利用研究課題審査委員会レビューアー
11. 高橋 大介：HPCI 連携サービス運営・作業部会委員
12. 高橋 大介：学際大規模情報基盤共同利用・共同研究拠点（JHPCN）課題審査委員
13. 高橋 大介：情報処理学会論文誌ジャーナル／JIP 編集委員会委員
14. 高橋 大介：情報処理学会ハイパフォーマンスコンピューティング研究会幹事
15. 建部 修見：HPCI 共用ストレージ運用部会副部会長
16. 建部 修見：HPCI 連携サービス運営・作業部会委員
17. 建部 修見：理化学研究所客員主管研究員
18. 建部 修見：情報通信研究機構協力研究員
19. 建部 修見：HPCI 利用研究課題審査委員会レビューアー
20. 建部 修見：学際大規模情報基盤共同利用・共同研究拠点（JHPCN）課題審査委員
21. 建部 修見：東京工業大学学術国際情報センター共同利用専門委員
22. 建部 修見：特定非営利団体つくば OSS 技術支援センター理事長
23. 建部 修見：SNIA 日本支部エクストリームストレージ研究会研究会長
24. 建部 修見：情報処理学会論文誌コンピューティングシステム編集副委員長
25. 多田野 寛人：日本応用数理学会「行列・固有値問題の解法とその応用」研究部会 運営委員.
26. 多田野 寛人：情報処理学会ハイパフォーマンスコンピューティング研究会運営委員.
27. 多田野 寛人：情報処理学会論文誌コンピューティングシステム論文誌編集委員.
28. 小林 諒平：SWoPP2018 実行委員 コンピュータシステム研究会担当幹事
29. 小林 諒平：電子情報通信学会リコンフィギャラブルシステム研究会専門委員
30. 小林 諒平：電子情報通信学会コンピュータシステム研究会専門委員

10. 社会貢献・国際貢献

1. Taisuke Boku: Steering Committee Chair, International Conference on High Performance Computing in Asia-Pacific Region (HPCAsia)
2. Taisuke Boku: Steering Committee Member, Intel eXtreme Performance Users Group (IXPUG)
3. Taisuke Boku: Organizing Committee Member, HPCAsia 2019
4. Taisuke Boku: Scientific Committee Member, Supercomputing Frontier Europe 2019
5. Taisuke Boku: Program Committee, IPDPS2018
6. Taisuke Boku: Organizing Co-Chair, IXPUG Workshop HPCAsia 2019
7. Taisuke Boku: Committee Member, SC18 Test of Time
8. Taisuke Boku: Organizing Committee Member, IXPUG Workshop ISC 2018
9. Taisuke Boku: Committee Member, CCGrid2019 Scale Challenge
10. Taisuke Boku: Program Committee, HPC China 2018
11. Taisuke Boku: Organizing Co-Chair: ATiP Workshop at SC18
12. Daisuke Takahashi: Program Committee, The 16th IEEE International Symposium on Parallel and Distributed Processing with Applications (IEEE ISPA 2018)
13. Daisuke Takahashi: Program Committee, International Conference on High Performance Computing in Asia Pacific Region (HPC Asia 2019)
14. Daisuke Takahashi: Program Committee, 6th International Workshop on Large-scale HPC Application Modernization (LHAM 2018) in Conjunction with 6th International Symposium on Computing and Networking (CANDAR'18)
15. Daisuke Takahashi: Program Committee, Parallel Fast Fourier Transforms (PFFT) in Conjunction with 25th IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC 2018)
16. Daisuke Takahashi: Organizing Committee, Parallel Fast Fourier Transforms (PFFT) in Conjunction with 25th IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC 2018)
17. Daisuke Takahashi: Program Committee, The 18th IEEE International Conference on Computer and Information Technology (CIT 2018)
18. Daisuke Takahashi: Program Committee, The 3rd International Workshop on GPU Computing and AI (GCA'18) in Conjunction with 6th International Symposium on Computing and Networking (CANDAR'18)
19. Daisuke Takahashi: Program Committee, 20th IEEE International Conference on High Performance Computing and Communications (HPCC-2018)

20. Daisuke Takahashi: Publicity Committee, The 18th International Conference on Computational Science and Its Applications (ICCSA 2018)
21. Daisuke Takahashi: Program Committee, The International Conference on Computational Science (ICCS 2018)
22. Daisuke Takahashi: Program Committee, The 13th International Workshop on Automatic Performance Tuning (iWAPT 2018)
23. Daisuke Takahashi: Program Committee, IEEE 12th International Symposium on Embedded Multicore SoCs (MCSoC-18)
24. Osamu Tatebe: Program Committee, IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2018)
25. Osamu Tatebe: Program Committee, IEEE International Conference on Cluster Computing (Cluster 2018)
26. Osamu Tatebe: Program Committee, International Supercomputing Conference
27. Osamu Tatebe: Poster Committee, IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC18)
28. Osamu Tatebe: Program Committee, International Workshop on Advances in High-Performance Computational Earth Sciences: Applications & Frameworks (IHPCES 2018)
29. Hiroto Tadano: Publication Co-Chair, The 37th Annual International Conference on Simulation Technology (JSST2018).
30. Hiroto Tadano: Publication Co-Chair, The 38th Annual International Conference on Simulation Technology (JSST2019).
31. Hiroto Tadano: Deputy Chair, 48th International Conference on Parallel Processing (ICPP2019).
32. Ryohei Kobayashi: Program Committee, International Conference on Field-Programmable Technology (FPT'18)
33. Ryohei Kobayashi: Program Committee, 6th International Workshop on Computer Systems and Architectures (CSA'18) held in conjunction with CANDAR'18

11. その他

海外長期滞在、フィールドワークなど

（該当なし）