VII. 高性能計算システム研究部門

1. メンバー

教授	朴 泰祐，建部 修見，高橋 大介，額田 彰， 塙 敏博（客員教授，東京大学）
准教授	多田野 寛人
助教	小林 諒平（9 月まで），藤田 典久
主任研究員	前田 宗則
研究員	平賀 弘平（2 月まで）
学生	大学院生 13 名，学類生 10 名
学内共同研究員	櫻井 鉄也，山口 佳樹，今倉 暁（システム情報系）
学外共同研究員	小柳 義夫（RIST），石川 裕（国立情報学研究所）， 松岡 聡（理化学研究所），佐野 健太郎（理化学研究所）， 中尾 昌広（理化学研究所），辻 美和子（理化学研究所）， 佐藤 三久（順天堂大学），天野 英晴（慶應義塾大学）， 川島 英之（慶応義塾大学），田中 昌宏（慶応義塾大学）

2. 概要

本研究部門では，GPU・FPGA 利用技術，並列プログラミング環境，ストレージシステム，並列入出力，分散システムソフトウェア，並列数値計算の高速化などの研究を行っている。今年度は以下の研究を行った。

- アプリケーションコードの GPU・FPGA 加速に関する研究
- 超高速ストレージシステム・Gfarm ファイルシステムの研究開発
- 並列数値アルゴリズムの GPU による高速化
- GPU プログラムのチェックポイントの研究
- 階層並列型数値解法の近似解精度改善
- 並列 FPGA 環境における FPGA 間通信に関する研究
- 複数の演算加速装置を統一的に扱えるプログラミング環境に関する研究
- CPU-GPU 密結合モジュールにおけるメモリシステムの性能評価に関する研究

3. 研究成果

[1] 分子動力学プログラム Amber の GPU 利用効率最適化（朴，小林）

計算科学研究センターの学際ハブ拠点形成事業の一貫として，生命科学分野の創薬に供する分子動力学法（MD: Molecular Dynamics）を用いた，レプリカ交換法によるペプチド膜透過シミュレーションの GPU 利用効率を大幅に向上させる研究を行った。本研究は

計算科学の社会実装における重要研究であり，株式会社 Ahead Biocomputing，NVIDIA 合同会社，東京工業大学（当時）との共同研究である。

ペプチド膜透過は人体における化学物質の体内への取り込みにおいて重要な現象であり，薬等がどのように浸透するかを解析するシミュレーション技術である。我々は MD シミュレーションコード Amber22 の GPU 版において，シミュレーションフェーズの各部で並列性が大きく変化し，部分的に必ずしも GPU の数千のコアを最大限に活かしきれていない点に着目した。また，本研究では最大 224 個のレプリカ交換を行うため，同数の並列プロセスを持つ MPI プログラムを実行する。そこで，各プロセスの GPU 負荷がタイミング的にアンバランスになることを利用し，最新の NVIDIA GPU に備えられた MPS (Multi-Process Service)機能を積極的に用いることで，少数の GPU でも同じ計算性能を得られることを証明した。その結果，レプリカ交換法による 224 プロセスの MD シミュレーションにおいて，Pegasus 上の最大 112 ノードを利用し，GPU コア利用率を最大で 2 倍に向上させることに成功した。

図 1 にコンセプトを示す。並列実行される，GPU を利用する複数プロセス間で計算フェーズによって並列度が動的に変化することを利用し，MPS により単一 GPU 上で最大 16 個の MPI プロセスで GPU を共有し，かつ各プロセスが要求する最大 GPU コア割合を意図的に大きく（最大で 100%）することで，結果的に GPU コアを 100%に近い効率で利用することを可能とした。その様子を**図 2** に示す。

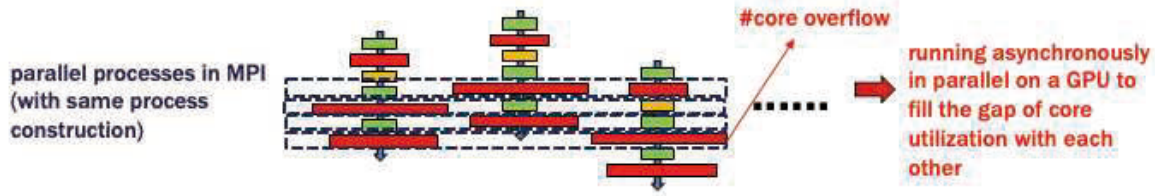


図 1　３つの GPU プロセスが互いに異なる並列度を持つことで，それらを組み合わせることで GPU 個を有効活用する様子

図 2　単一 GPU 上で最大 16 プロセスを実行した場合，MPS によってコア利用率が 100%まで上昇する様子（NVIDIA Nsight Systems によるモニタ）

これを Pegasus 上の最大 112 ノードまで展開して実験したプロセス数は 224 で固定である。MPS には CMATP (CUDA_MPS_ACTIVE_THREAD_PERCENTAGE)というパラメ

ータ設定があり，各プロセスが要求する最大コア利用率（%）を設定できる。単純計算すると，N プロセスが GPU を共有すると，conservative には “1/N” を設定するというのが一般的な考えであるが，MPS が動的にコア割当を変更してくれることを考慮し，これをより大きな値に設定することが鍵となるアイデアである。そこで，GPU 上の MPS 共有されるプロセス全体での CMATP を表す Aggregated-CMATP という概念を新しく設け，これを数百%まで敢えて拡大することで**図 2** に示すような状況を実現する。

これらの結果，GPU 当たりのプロセス数を 4〜16 まで変化させた場合，**図 3** に示すようにどのケースでも Aggregated-CMATP が 250 程度の場合に最大性能が得られることがわかった（Pegasus，H100 GPU）。当然，ノード数＝GPU 数が多い方が絶対性能は高くなるが，わずか 14 ノード（16 process/GPU）での GPU 利用効率は 112 ノード（2 process/GPU）の場合の 2 倍という結果になった。従って，112 ノードを用いる計算を少数実行するより，14 ノードでの計算を多数実行する方が効率が倍になるということで，単純計算で計算リソースを半分に減らすことができる。（1 回のシミュレーション時間は 112 ノードの方が当然短い。）

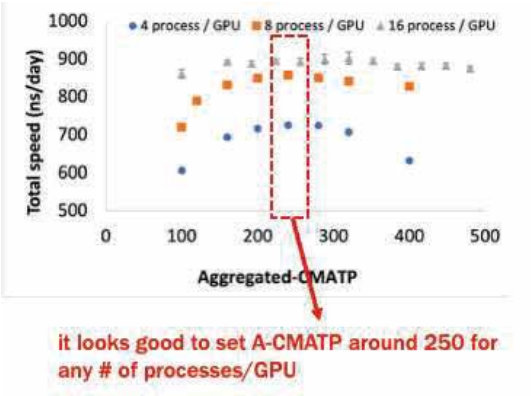


図 3　GPU 当たりプロセス数を変化させた場合の Aggregated-CMATP 調整による計算性能の最適化

この手法により，多数のケースのシミュレーションを実行する場合の使用リソース量を半減することが可能であることが結論づけられた。本研究の成果は国際会議 ICPP2024 にて発表された。

[2] 地域気象コード City-LES の高度化（朴，藤田）

CCS の地球環境研究部門・日下グループとの共同研究として，同グループが開発を続けている地域気象コード City-LES の新規追加機能であるドライミスト効果部分の GPU 化を行い，GPU スーパーコンピュータ Pegasus 及び 2025 年 1 月から運用開始した GPU・CPU 一体型モジュールを用いたスーパーコンピュータ Miyabi-G にコードを移植し，CPU

のみ使用した場合に比べ，ドライミスト部分単体では Pegasus では最大で 67 倍（4 ノード実行時比較），Miyabi-G では最大 20 倍（同様）の大幅な性能向上を達成した。また，City-LES コード全体でも Pegasus で最大 29 倍（16 ノード実行時比較），Miyabi-G で最大 21 倍（4 ノード実行時比較）とこちらも大幅な性能向上を達成した。また，Pegasus では最大 128 ノード，Miyabi-G では最大 32 ノードまでの Strong Scaling による並列処理性能評価を行った。問題サイズは X×Y×Z=512×512×512 で実空間では 5m メッシュ（2.56km×2.56km×2.56km）のスケールである。

同コードの GPU 化は 2022 年度までの日下グループとの共同研究により GPU 化が行われ，当時の Cygnus スーパーコンピュータで CPU に比べ最大 17 倍の性能向上を達成していたが，今回は新しく追加された，市街地歩道におけるドライミスト噴霧器による部分気温低下効果のシミュレーションが追加されている。同機能は CPU コードのみが提供され，これまでの GPU 化コードと組み合わせるとドライミスト機能のみのために一旦制御が CPU に移され，データ移動とカーネル起動オーバヘッドによって十分な性能が得られなかった。ドライミスト機能部分を完全に GPU 化することで，これらのオーバヘッドを低減した結果，大幅な性能向上につながった。なお，両システムで CPU 版に対する GPU 版の性能向上に差があるが，これは Miyabi-G の GH200 モジュールに搭載された Grace CPU が 72 コアの Arm アーキテクチャであり，Pegasus の CPU である 48 コア Xeon Gold に比べ性能が高いことが主な理由である。

ドライミスト機能の GPU 化は NVIDIA 製 GPU（Pegasus: H100，Miyabi-G: GH200 の GPU 部分である H100）上で OpenACC を用いて記述した（**図 5** 参照）。City-LES の基本部分と同様，並列プロセスは XY 次元の水平方向で MPI 分割し，Z 次元の鉛直方向は 1 つのプロセス内で逐次処理している（**図 4** 参照）。ドライミスト効果の計算では，鉛直報告については依存性があるため逐次処理し，水平方向は依存性のない空間並列性が利用できるため，GPU 内のコア間並列は OpenACC の loop independent 節によって最大の並列化を行っている。実際の計算は計算ボディ内で数十行分あるがここでは省略する。

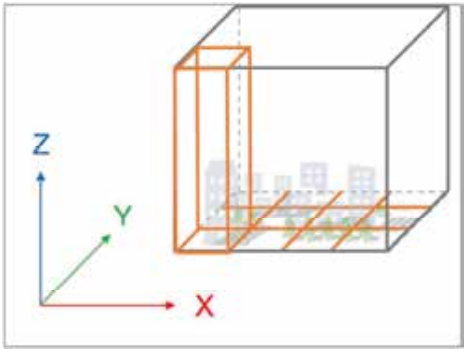


図 4　MPI プロセス分割

```
!$acc kernels
do k = max_mist,kds+1 ,-1
 !$acc loop independent
   do j= jts ,jte
 !$acc loop independent
      do i= its ,ite
 ! （演算ボディ省略）
      end do
   end do
 end do
!$acc end kernels
```

図 5 ドライミスト計算の並列化
（水平方向のみ GPU コア並列化）

並列処理の Strong Scalability について，**図 6** に Pegasus（左）及び Miyabi-G（右）のそれぞれについて示す。ファイル入出力等の計算の初期化及び終了部分を除いた正味のシミュレーション計算部分を 1000 タイムステップで評価している。Pegasus，Miyabi-G とも 1 ノード当たりの H100 GPU は 1 台のため，MPI プロセスは各ノードに 1 つのみとしており，**図 4** の 1 つの直方体が各ノードに割り当てられる。ここに示した空間並列性は，ノード台数を増やして Strong Scaling 性能を見ると，水平方向並列度が 512×512 であることから，数十〜百 GPU までシステム規模を増大させると，GPU 当たりの演算並列性が低下し，並列化効率が低下する原因となることが予想される。測定の結果，Pegasus の場合は 4 ノード実行を基準とした並列化効率は 32 ノードでもほぼ 1 で性能が順調にスケールしているが，Miyabi-G では 0.5 程度まで低下している。しかし，計算実行時間を比較すると，Miyabi-G では Pegasus の 50%〜30%程度まで短縮されていることから，Miyabi-G のノード当たりの実効性能が Pegasus に比べ大幅に向上していることがわかる。両システムのノード間奏後結合網は共に InfiniBand NDR200 であることから，通信のボトルネックが Miyabi-G でより大きいことが原因と推察される。この性能差については，Miyabi-G の H100 GPU の HBM3 メモリの理論ピークバンド幅が 4TB/s であるのに対し，Pegasus では 2TB/s と半分であることから，気象コードの特性により性能がメモリバンド幅に大きく律速されることが原因と考えられる。

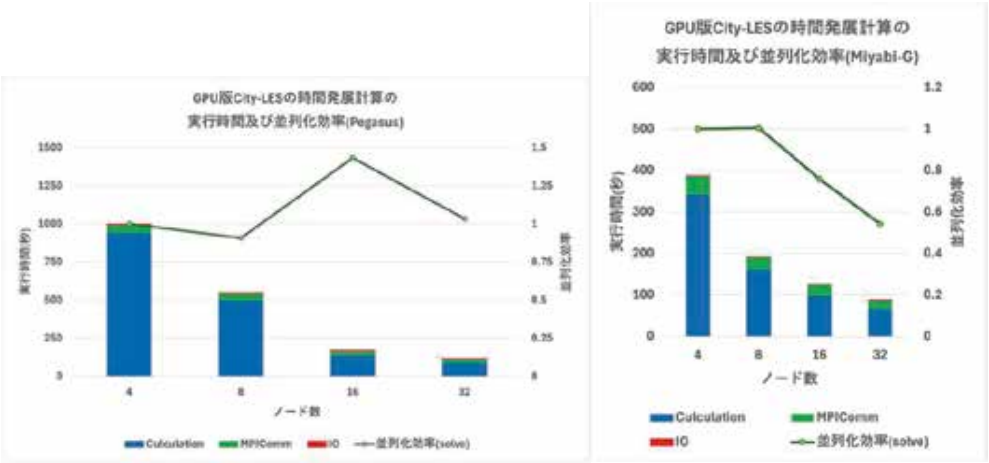


図 6　ノード数を 32 まで増やした場合の時間発展計算部分（初期化部分を除く）の Strong Scaling 性能（左：Pegasus，右：Miyabi-G）

以上の結果より，ドライミスト効果を含めた最新の GPU 版 City-LES コードは Pegasus，Miyabi-G という両システムで共に CPU のみの場合に比べ遥かに高い性能を得ており，今後の気象分野の研究において計算時間を大幅に短縮し，研究を加速することが期待される。本研究の内容は 2025 年 5 月の情報処理学会 HPC 研究会にて発表予定である。

[3] 地域気象コード City-LES の初期化部分高速化（朴）

同様に City-LES コードの高度化の一貫として，計算ボディに入る前の初期化部分の高度化を行った。同コードでは太陽光のトラッキングを Ray Tracing 手法によって行い，精細な温度シミュレーションを行うため，地表面の建物データを Ray Tracing に適した構造化データに変換しており，この部分の計算は非常に時間がかかる。シミュレーションの中心部分のタイムステップ数が十分に大きければ無視できるが，短時間シミュレーションではオーバヘッドが極めて大きいという問題があった。従来コードでは初期化部分がコード全体に組み込まれていて，ジョブ実行のたびに構造化データ作成処理が行われていたが，近年の気象グループの研究により，コードを初期化部分と計算ボディに分割し，初期化は 1 回のみですむように工夫されたが，この初期化について十分な最適化が行われていなかった。

気象グループによるオリジナルコードでは，XY 平面に空間分割された MPI 並列では隣接空間でのデータ依存性が強く，分散メモリ並列が難しかった。この点は問題の性質上の制約ではあるが，同コードはそれ以上の並列化を行っておらず，単一プログラムを完全に逐次処理していた。そこで，1 ノード内の CPU 処理に対してできる限りの並列化を行い，処理速度を向上させることにした。データ依存性の問題から，GPU 化は適当で

ないと判断し，共有メモリを用いた CPU コア間並列を適用するため，処理を OpenMP によって並列化した。

```
icol_old = icol
!$omp parallel do private(jj, ii) reduction(+:icol) schedule(static)
   collapse(2)
do kk = kbp1_rs(nn), kbp2_rs(nn)
    do jj = jbp1_rs(nn), jbp2_rs(nn)
        !if( (jj == jms) .or. (jj == jme) .or. (ibp2(nn) >= ite) ) cycle
        if ((jj == jms_rs) .or. (jj == jme_rs) .or. (ibp2_rs(nn) >=
            ime_rs - 1)) cycle
        if (fluid_flag_rs(ibp2_rs(nn) + 1, jj, kk) == 0.0d0) cycle
        icol = icol + 1
    end do
end do
!$omp end parallel do
do nn = icol_old + 1, icol
    if (nn == F_col_rs(icr)) then
        irow = irow + 1
        icr = icr + 1
    end if
end do
```

図 7　建物構造データ作成コードの中核部分の OpenMP 化

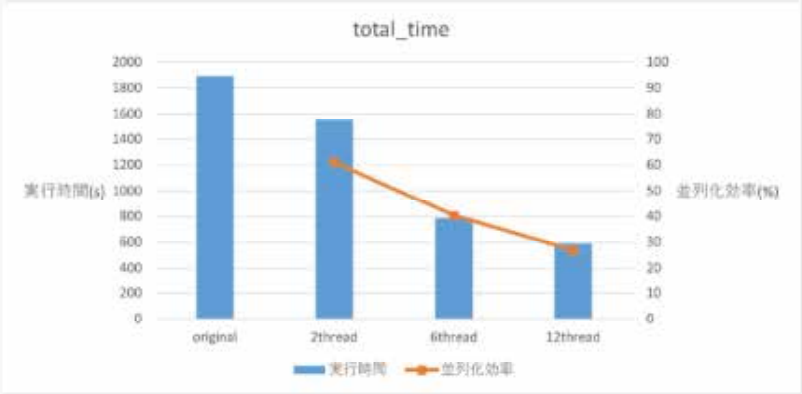


図 8　OpenMP によるマルチスレッドコードの実行性能と並列化効率

図 7 に建物構造データ生成部分の中核部分の抜粋を示す。OpenMP 化では最低限のデータ依存関係に配慮し，並列化を行っている。 **図 8** に Cygnus スーパーコンピュータ上の最大 12 スレッドを用いた場合の性能向上を示す。Original が OpenMP 化を行っていない逐次実行版で，並列化効率はスレッド（＝コア）数によって期待される性能に対する効率を表す。12 スレッド時に最大の性能が得られ，Original に比べ 3.2 倍の性能が得られている。

MPI 並列を用いたより高度な並列化が望まれるところではあるが，先述のデータ依存性の問題解決と，初期化部分が本体ループと切り離され，同一物理モデルに関しては一回の実行で済むことから，1 ノードでの OpenMP 並列化をもって解決とすることとした。

[4] 宇宙物理コードの GPU・FPGA 複合演算加速プログラミング（朴，藤田，小林）

我々は昨年度まで，GPU と FPGA というマルチ演算加速デバイスを用いた協調計算コンセプト CHARM (Cooperative Heterogeneous Acceleration with Reconfigurable Multidevices) の下，これまで OpenACC のみの単独記述によるプログラミングを可能とする言語処理系 MHOAT (Multi-Hetero OpenACC Translator)を開発してきた。しかし，同環境はプロトタイプ開発には成功したものの，より完成度の高い CHARM 環境対応プログラミングフレームワークのためには商用環境の利用が確実である。多種演算加速デバイス向けの統一的プログラミング環境としては，Intel oneAPI が代表として挙げられる。しかし，同環境は「単一コードを異なるデバイス向けにそれぞれコンパイル・実行する」ことを前提としており，複数の異機種デバイスが混載された環境での実行を標準的にはサポートしていない。

Intel oneAPI では DPC++言語でプログラムを記述する。標準的な oneAPI 環境では Intel 製の GPU 及び FPGA 向けのバイナリが生成され，さらにサードパーティモジュールを用いることで NVIDIA 製 GPU にも対応できる。そこで，CHARM コンセプトの下，NVIDIA GPU と Intel FPGA の混載システムである Cygnus スーパーコンピュータを対象として両デバイスを DPC++でプログラムし，oneAPI 環境で実行する試みを行った。まず，簡単なトイコード（両デバイスでそれぞれ異なる簡単なループ処理を行い，結果を交換し交互に実行する）に対して GPU 向け，FPGA 向けの部分コンパイルを行い，最後に関連ライブラリを含めて実行モジュールを生成するスクリプトと Makefile 記述を確立した。　　**図 9** に処理の流れを示す。この結果，コードは正しく動き両デバイスを用いることができたが，FPGA カーネルの起動時のオーバヘッドが極めて大きく（実行時間の数十倍），このままでは問題があることがわかった。これについては原因を調査中である。

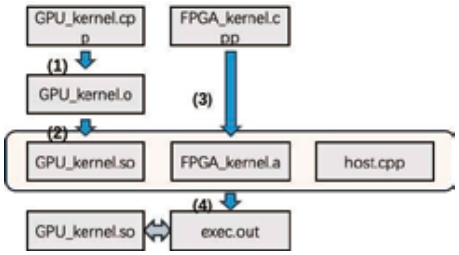


図 9　oneAPI 環境における CHARM コードのコンパイル手順

次のステップとして，実アプリケーションコードの DPC++記述を行い，NVIDIA GPU と Intel FPGA で部分的にコンパイルし，合成させる試みを行った。対象コードは従来より CHARM 向けに開発してきた宇宙物理シミュレーションコード ARGOT である。しかし，後者についてはトイコードほど単純ではなく，FPGA 向けのコード変更が大幅に必要

であることがわかった。そこで，今年度の研究では DPC++記述された ARGOT コードの全て部分を NVIDIA GPU で実行し，機能及び性能の検証を行うことにした。DPC++による公開された実用コードの例は非常に少なく，本研究はその意味においても重要である。

しかし，ARGOT コードの膨大な行数を手作業で DPC++に移行するのは非現実的である。そこで，Intel が提供している CUDA コードを DPC++の原型である SYCL に変換する SYCLomatic ツールを用いることにした。結果的に，ARGOT コードの SYCL 化をほぼ自動的に行うことができ，これを元に oneAPI 向けの DPC++コードを作成した。**図 10** に，オリジナルの GPU 版 ARGOT コードの一部（CUDA 記述）が SYCL に変換された例を示す。

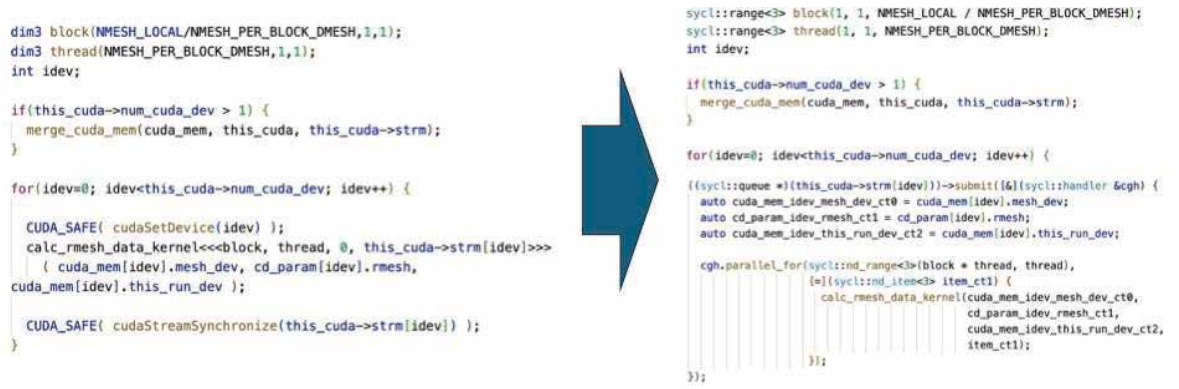


図 10　SYCLomatic による CUDA コードの DPC++への変換

図 11 に ARGOT コード全体を CUDA から DPC++に変換した場合の性能評価を示す。先述したように，この実行は CHARM モデルではなく NVIDIA GPU（V100）のみでの実行であるが，CUDA コードをそのまま NVIDIA CUDA 環境で実行した場合と，SYCLomatic を経て DPC++に変換し，これを oneAPI 環境で実行した場合の評価である。つまり，前者が直接的な CUDA デバイス実行であるのに対し，後者はコード変換を経て同じハードウェア上で実行していることになる。まず，両コードの計算出力は一致し，正しく計算されていることを確認した。次に性能であるが，GPU カーネルの全体的な処理時間，CPU と GPU の間のデータ転送，その他にばらつきはあるものの，総実行時間は DPC++の方が短縮されるという興味深い結果となった。詳細な性能解析は今後の課題であるが，一つの原因と考えられるのは，CUDA というデバイスパラメータを直接指定する記述に対し，DPC++という抽象度の高い記述に変換され，それが最適化された結果，CUDA 記述を上回る性能が得られたという可能性である。しかし，実行時間は約半分に短縮されているため，これだけで説明できるかは定かでない。今後の検証が必要である。

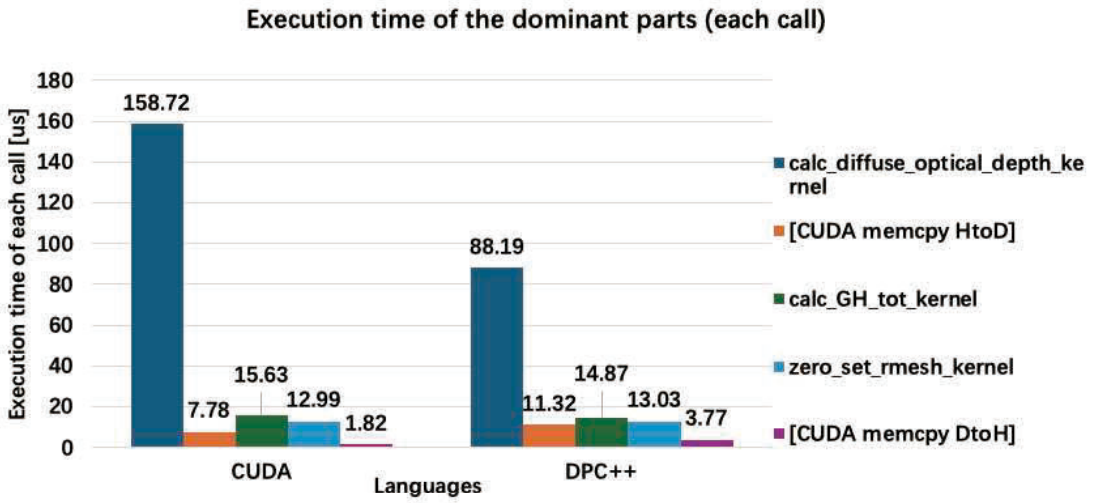


図 11　CUDA 版 ARGOT コードの直接 GPU 実行と SYCLomatic→DPC++変換での oneAPI 実行の比較

本研究の結果は IEEE Cluster2024 におけるポスター発表で公表済みである。

[5] 超高速ストレージシステムの研究（建部）

スーパーコンピュータシステムでは，演算性能の向上に対しストレージ性能の向上が追い付いていないため，ストレージ性能が大きなボトルネックとなっている。この問題を解決するため，スーパーコンピュータの計算ノードのローカルストレージシステムを活用した並列キャッシングファイルシステム CHFS/Cache の研究開発を行っている。CHFS/Cache では，計算ノードの不揮発性メモリ，ローカルストレージを用いて高並列な分散キーバリューストアを構築し，その上に高並列なファイルシステムの設計を行った。キャッシュ機能については，既存システムで問題であった小さいファイル処理におけるオーバヘッドを解決するため，利用者にとって無理のない範囲で並列ファイルシステムとの間の一貫性の緩和について考察し，設計を行った。さらに，フラッシュ時の性能低下を解決するため，I/O-aware フラッシング方式を提案し，性能低下が抑えられることを示した。

令和 6 年度においては，CHFS/Cache の適用範囲をさらに広げ，性能向上させるための設計を行った。並列キャッシングファイルシステムにおいては，ファイルシステムの全ての機能を実現する必要はなく，主にファイルに対する読込，書込の性能を向上させることを目的として機能を限定して設計がなされていたため，ファイル名やディレクトリ名の変更などの操作は実現できなかった。一方で，適用範囲を広げるためにはその機能も必要である。そのため，読込，書込の性能を保ったままそれらの機能を実現するための設計を実施した。ファイル名，ディレクトリ名の変更を行うためには，名前空間の管理が必要であり，その名前空間

の管理をメモリ上で行うことにより性能を保ったまま機能向上を可能とした。さらに，近年の CPU ではコア数が増加している。一方でこれまでの設計ではコア数の増加に見合った性能向上をしない問題があった。この問題を解決するため CPU 内における資源を共有しない設計を実施した。この設計の場合，負荷分散が問題となるが，負荷を均等にするための新たなハッシュ方式も提案した。これらの設計をもとに FINCHFS を実装し，コア数が増えた場合でも性能向上することが確かめた。

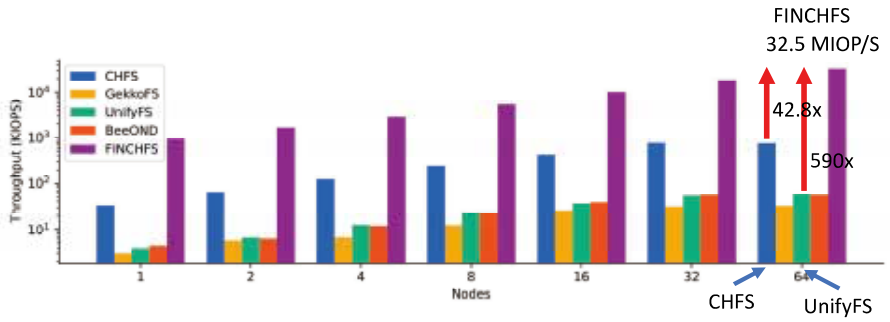


図 12　MDtest Hard Write による性能評価

図 12 に 3,901 バイトのファイルを並列プロセスが同一ディレクトリに作成するベンチマークを用いた性能を示す。性能は 1 秒間に作成したファイル数で示され，各ノード 32 プロセスのウィークスケーリングの性能評価を実施した。比較対象として最先端のシステムにおける性能も示している。64 ノードにおいて FINCHFS は 32.5 MIOPS の性能を達成し，CHFS の 42.8 倍，UnifyFS の 590 倍の性能であった。これらの成果は IEEE 国際会議 CLUSTER おいて発表した。

また，計算ノードの不揮発性メモリの性能を十二分に活用するための並列 1 次ストレージシステム PEANUTS の設計を行った。PEANUTS では，不揮発性メモリの全領域をメモリ登録し，全領域を RDMA 可能とし，その領域を並列 1 次ストレージとして利用する。また，ストレージサーバを設置せず，I/O ライブラリを設計，実装することにより並列プログラムが直接不揮発性メモリをアクセスするようにした。書込みにおいては，ログ形式でローカルの不揮発性メモリに書込み，同期とファイルクローズのタイミングで書込み領域の共有を行う。これにより，各プロセスはファイルの全領域のアクセスが可能となる。最先端のシステムとの比較では書込，読込性能が 2 倍から 6 倍であった。本成果は国際会議 Euro-Par において発表した。

[6] 分散ファイルシステムおよびグリッド・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。

今年度はユーザの利用が多い gfptar の高度化を実施した。gfptar は多数のスモールファイルを効率的に Gfarm に保管するためのツールであり，多数ファイルを並列にアーカイブ，圧縮して Gfarm に保存することが可能である。ユーザから，ファイル数が億単位になるとメモリの少ないマシンではメモリが枯渇し実行できないという報告があった。ファイルリストをメモリに保持できないことが問題であった。この問題に対応するためメモリにファイルリストを保持するのをやめ，データベースを利用するよう改修を行った。この改修によりメモリが少なくても実行可能となった。また，これまで最低限の機能だけしか実装していなかったが，希望の多かった機能を実装した。例えば，アーカイブ作成時にネットワーク等でエラーが発生すると，これまではまた初めからやり直しとなっていたが，途中からの再開を可能とした。これらの gfptar の高度化を実施し，Gfarm バージョン 2.8.6 として 2024 年 12 月 16 日に公開した。

さらに，Gfarm がアクセストークンを用いて利用可能となったことから，HTTPS ゲートウェイの整備を進めた。これにより Gfarm のクライアントがなくても HPCI 共用ストレージへのアクセスが可能となった。成果は https://github.com/oss-tsukuba/gfarm-http-gateway で公開した。

[7] GPU クラスタにおける並列数論変換の実現と評価（高橋）

数論変換（number-theoretic transform，以下 NTT）は，離散 Fourier 変換を有限体に一般化したものであり，準同型暗号，多項式乗算，多倍長精度乗算などに広く用いられている。NTT の実装がいくつか提案されている。本研究では，GPU クラスタにおいて並列 NTT を実現し性能評価を行った。

n点 NTT は$\mathbf{F}_p = \mathbf{Z}/p\mathbf{Z}$（$p$は素数）において以下のように表すことができる。

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

ここで，ω_nは 1 の原始n乗根である。式(1)は高速 Fourier 変換（fast Fourier transform，以下 FFT）と同様のアルゴリズムを適用することで，演算回数を$O(n \log n)$に削減することができる。Out-of-place FFT アルゴリズムとして知られる Stockham FFT アルゴリズムを基数 2 の NTT に適用すると，**図 13** の Algorithm 1 に示す基数 2 の Stockham NTT アルゴリズムが得られる。

Algorithm 1 基数 2 の Stockham NTT アルゴリズム

Input: $n = 2^q$, $X_0(j) = x(j)$, $0 \le j \le n-1$, and ω_n is the primitive n-th root of unity
Output: $y(k) = X_q(k) = \sum_{j=0}^{n-1} x(j)\omega_n^{jk} \bmod p, 0 \le k \le n-1$

```
1: l ← n/2
2: m ← 1
3: for t from 1 to q do
4:    for j from 0 to l − 1 do
5:       for k from 0 to m − 1 do
6:          c0 ← X_{t−1}(k + jm)
7:          c1 ← X_{t−1}(k + jm + lm)
8:          X_t(k + 2jm) ← (c0 + c1) mod p
9:          X_t(k + 2jm + m) ← ω_n^{jm}(c0 − c1) mod p
10:      end for
11:   end for
12:   l ← l/2
13:   m ← 2m
14: end for
```

図 13　基数 2 の Stockham NTT アルゴリズム

Algorithm 1 の 8 行目では剰余加算が行われ，9 行目では剰余減算と剰余乗算が行われる。Montgomery 乗算や Shoup 乗算を用いることで時間の掛かる除算を実質的に行うことなく，加算，減算，乗算，ビットマスク，およびシフト演算のみで剰余乗算を行えることが知られている。また，four-step FFT アルゴリズムとして知られる手法は NTT に適用可能であるので，MPI と OpenACC を用いて four-step NTT を並列化した。

性能評価にあたっては，four-step NTT の GPU 実装，CPU と GPU 間の転送時間を含む four-step NTT の GPU 実装，および six-step NTT の CPU 実装の性能を比較した。測定に際しては，weak scaling における順方向 NTT を連続 10 回実行し，その平均の経過時間を測定した。GPU クラスタとして，筑波大学計算科学研究センターに設置されている Pegasus（150 ノード）のうち 1~32 ノードを用いた。コンパイラは NVIDIA HPC Compilers 23.9 を用いた。コンパイルオプションは GPU 実装では -fast -acc=gpu -gpu=cc90 を，CPU 実装では -fast -mp -tp=sapphirerapids を指定した。MPI ライブラリは OpenMPI 4.1.5 を用いた。各ノードあたりの MPI プロセス数は 1, 各 MPI プロセスあたりのスレッド数は 48 に設定した。N点 NTT の Giga-operations per second（Gops）値は$(3/2)N\log_2 N$より算出している。

並列 NTT の weak scaling 性能（$N = 2^{30}$ ×ノード数）を**図 14** に示す。**図 14** から分かるように，GPU 実装が CPU 実装よりも高速であるが，4 ノード以上において CPU と GPU 間の転送時間を含む GPU 実装は CPU 実装とほぼ同じ性能になっている。ノード数が増加するに従

って全対全通信のメッセージサイズが小さくなり，通信バンド幅が小さくなる。したがって，GPU 実装と CPU 実装の性能差は，ノード数が増加するに従って小さくなる。

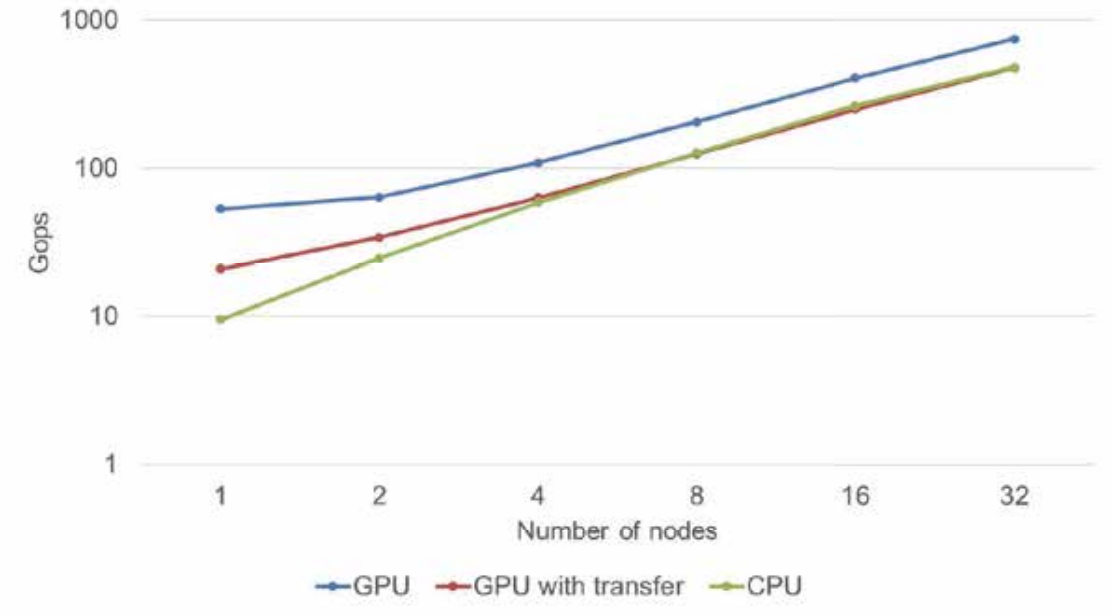


図 14　並列 NTT の weak scaling 性能（$N = 2^{30}$ ×ノード数）

[8] 数学定数に対する 2 進 BBP 型公式の計算における除算（高橋）

πに対する Bailey-Borwein-Plouffe（BBP）公式は，前のビットをすべて計算することなく，πの特定のビットを計算できることが知られている。これまでに数学定数に対する BBP 型公式がいくつか提案されている。BBP 型公式の計算で最も時間を要するのはべき剰余であるが，級数の項の除算も無視できない時間を要する。この除算は exact division を用いることで効率的に計算できることが知られているが，exact division を 2 進数で計算する場合，除数は奇数でなければならない。本研究では，数学定数に対する 2 進 BBP 型公式の計算における除算に exact division を用いる方法を提案する。

BBP 公式は以下の式で表される。

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k}\left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6}\right) \qquad (2)$$

16 進数で$n+1$桁目から始まるπの数桁を計算することを考える。これは$\{16^n\pi\}$の計算と等価であることに注意する。ここで$\{\cdot\}$は小数部を示す。式(2)から以下の式が得られる。

$$\{16^n\pi\} = \left\{4\{16^n S(1)\} - 2\{16^n S(4)\} - \{16^n S(5)\} - \{16^n S(6)\}\right\} \qquad (3)$$

$$S(j) = \sum_{k=0}^{\infty} \frac{1}{16^k(8k+j)} \qquad (4)$$

$$\{16^n S(j)\} = \left\{\left\{\sum_{k=0}^{n} \frac{16^{n-k}}{8k+j}\right\} + \sum_{k=n+1}^{\infty} \frac{16^{n-k}}{8k+j}\right\} = \left\{\left\{\sum_{k=0}^{n} \frac{16^{n-k} \bmod (8k+j)}{8k+j}\right\} + \sum_{k=n+1}^{\infty} \frac{16^{n-k}}{8k+j}\right\} \quad (5)$$

$\bmod (8k+j)$が式(5)の 1 番目の総和の分子に現れるのは，小数部だけを計算すればよいからである。式(5)の 1 番目の総和の分子$16^{n-k} \bmod (8k+j)$は，バイナリ法を用いて効率的に計算することができる。式(5)の 2 番目の総和の分子では，16 の指数が負になっているので，残りの項が浮動小数点演算の計算機イプシロンより小さくなるまで計算すればよいことになる。最終結果を 16 進数に変換することにより，$n+1$桁目から始まるπの数ビットが得られる。

Exact division を 2 進数で計算する場合，除数は奇数でなければならない。しかし，式(5)の 1 番目の総和において分数の分母 $8k+j$は，$j = 4, 6$では偶数になるため，exact division を直接使うことはできない。その場合，2 進 BBP 型公式のべき剰余に Montgomery 乗算を適用する方法を用いることで，分数の分母を奇数にすることができる。1 ワードの被除数を 1 ワードの除数で割ったmワードの商の小数部の計算を固定小数点演算で行うアルゴリズムを**図 15** の Algorithm 2 に示す。

Algorithm 2 1 ワードの被除数を 1 ワードの除数で割った m ワードの商の小数部の計算

Input: x, N, r, μ such that $0 \le x < N$, $0 < N < \beta$, $\gcd(\beta, N) = 1$,
$r = (\beta^m \cdot x) \bmod N$, $\mu = N^{-1} \bmod \beta$

Output: $Q = \lfloor (\beta^m \cdot x)/N \rfloor$

1: **if** $r = 0$ **then**
2: **return** 0
3: $q_0 \leftarrow (-r \cdot \mu) \bmod \beta$
4: **for** j **from** 1 **to** $m-1$ **do**
5: $q_j \leftarrow [\{(\beta - 1) - \lfloor (q_{j-1} \cdot N)/\beta \rfloor\} \cdot \mu] \bmod \beta$
6: **return** $Q = \sum_{i=0}^{m-1} q_i \beta^i$.

図 15　1 ワードの被除数を 1 ワードの除数で割った m ワードの商の小数部の計算

式(2)の BBP 公式を用いてπを計算する際に, exact division を用いた提案手法と exact division を用いない従来手法の実行時間を比較した。**表 1** は，Intel Core i3-8121U と Intel Xeon Gold 6230 でπの 16 進10^{10}桁目を計算するのに必要な実行時間を示している。提案手法は従来手法に比べて Intel Core i3-8121U で約 1.05 倍，Intel Xeon Gold 6230 で約 1.23 倍高速であることが分かる。提案手法が従来手法よりも高速である理由としては，提案手法が整数除算命令の代

わりに exact division を用いているためであると考えられる。また，Intel Core i3-8121U では，Intel Xeon Gold 6230 よりも提案手法と従来手法の性能差が小さくなっている。これは Intel Core i3-8121U が整数除算命令に要するサイクル数が少なくなっているためである。

表 1　πの 16 進10^{10}桁目計算の実行時間（秒）

	従来手法	提案手法
Intel Core i3-8121U	2675.36	2543.13
Intel Xeon Gold 6230	464.41	378.61

[9] OpenACC で GPU 化されたアプリケーションの透過的チェックポイントの研究（額田）

スパコンなどの共用の計算機システムは一般的にジョブ管理システムによって運用され，ユーザ間の利用機会の公平性を担保するためにジョブあたりの最長実行時間の制限が設けられている。この時間を超える計算を行う場合には計算途中の状態をファイル等に保存するチェックポイント技術が必要になる。アプリケーションの開発者がこの機能を組み込むこともあるが，大規模なアプリケーションや多人数で開発が続いているアプリケーションでは難しく，DMTCP のようなバイナリを改変することなくそのままチェックポイント機能を実現するシステムレベルチェックポイントソフトウェアが有用である。現状筑波大学のスパコンを含め多くのスパコンでは NVIDIA 社製 GPU を搭載し，様々なアプリケーションの高速化に活用されている。これまで額田らは CUDA アプリケーションに対応するように DMTCP を拡張する研究を行ってきた。

既存の CPU 用コードを CUDA で書き換える場合には CPU メモリに加えて GPU メモリを管理するということだけでなく，GPU で実行するループ部分などを専用のカーネル関数として定義する必要があり，このためのコードの修正コストが大きいという問題がある。この点を解決するため OpenACC が開発され，OpenMP でマルチスレッド化を行うのと同様の書式でループ部分の GPU 化が実現できる。この OpenACC が普及して来ており，近年では CUDA より多く使われているため OpenACC アプリケーションのチェックポイント対応も重要になってきた。

まず OpenACC の内部的な挙動を分析する必要がある。CUDA アプリケーションでは明示的に CUDA API 関数を呼び出し，またコンパイル時の中間コードも全て確認することができるため非公開の API 関数を含めて対応することができる。一方で OpenACC の場合は OpenACC 用のランタイムライブラリがリンクされ，ドキュメントが一切無い非公開 API 関数が用いられている。中間コードの確認もできず，唯一確認できるのは PTX という

GPU カーネル部分の中間コードである。strace コマンドなどでさらに解析を進めたところ，”libcuda.so.1”という CUDA Driver API 関数を実装しているライブラリがロードされていることが判明した。そこでこのライブラリを置き換えて API 関数の呼び出しをモニタリングしたところ，まず OpenACC ランタイムライブラリが約 75 の API 関数のシンボルが存在するかとチェックしているようで，追加でカーネル実行関連など 3 つの API 関数が実際に使用されていた。

CUDA アプリケーションの透過的チェックポイントでは CUDA API にモニタリング機能を追加し，チェックポイント保存時に GPU 側のデータを全て CPU メモリに保存することで実現している。OpenACC の機能だけを使用して作成されたバイナリであればこれらの API 関数に機能追加するだけでよい。実際にはメモリ確保など GPU の状態に変更を加える API 関数の呼び出しのみをモニタリングすればよい。しかしあらゆる OpenACC アプリケーションでチェックしたわけではなく他の API 関数が使用される可能性もある。また OpenACC アプリケーションが追加で直接 API 関数を呼ぶこともあるため，結局全ての API 関数に対応する必要がある。CUDA Driver API 関数の数は 500 を超えており，また一部は新旧複数のバージョンが存在する。それら全てに手作業で対応することは現実的ではなく，プログラムで自動生成する方法を採用した。GPU を 1 度しか初期化できないという制約があるため，チェックポイント対応では GPU アクセス専用のサーバプロセスを用意し，再開時には新たにこのプロセスを起動するというのが唯一の方法である。このため全ての CUDA API 呼び出しはプロセス間で通信が必要になる。さらに GPU の状態に変更を加える API 関数の場合，保存されたファイルから実行を再開する時に同じ順番で再実行することができるように必要な引数等を保存する。その他の API 関数の場合は何もしない。このような処理を加えるため，GPU の状態に変更を加える API 関数の選別だけ行い，その後のコード生成は自動的に行う。関数の引数情報についてはプログラム開発用ヘッダファイル”cuda.h”から抽出することができる。

自動生成の例として cuModuleGetFunction という API 関数について説明する。この関数は CUmodule 型のモジュールの中で指定した名前のカーネル関数のハンドル CUfunction を取得する API 関数である。ヘッダファイル cuda.h では以下のように宣言されている。

```
CUresult CUDAAPI cuModuleGetFunction(CUfunction *hfunc, CUmodule hmod, const char *name);
```

CUDA Driver API では CU で始まる型は構造体へのポインタとなっているものが多いが，その情報は利用していない。引数の中で hfunc はポインタ型であるのでこの関数の出力とみなし，hmod はポインタではないため入力とみなす。name はポインタ型ではあるが char 型であり，また const がついているため文字列の入力と判断する。この API 関数呼び出しをプロセス間で転送するために引数等をプロセス間共有メモリで渡す。そのための構造体を

以下のように生成する。まず API 関数毎に引数のデータを持つ構造体の定義を生成する。そしてそれら全ての構造体を union として持つ構造体を生成する。

```
typedef  struct {
        CUfunction  hfunc;
        CUmodule  hmod;
        const  char  *name;
}  s_cuModuleGetFunction;
…
typedef  struct  {
         int   op;
         CUresult   res;
         union  u  {
                  s_cuInit  cuInit;
                  s_cuModuleGetFunction  cuModuleGetFunction;
                  ….
          }
```

次にこの構造体を使用してサーバプロセスに中継する処理を行うコードを生成する。まず入力となる引数を書き込む。文字列データの場合はポインタではなくそのデータを構造体内の data エリアにコピーする必要がある。なお，両プロセスにおいてこの共有メモリは同じメモリアドレスにマップされている。そして API 関数の ID を設定し，別プロセスに実行を依頼する。別プロセスでの実行が終わると構造体に API 関数からの出力が書き込まれているのでそれをアプリケーションが指定したアドレスに返す。この API 関数は実行再開時に再実行する API に含まれているため，この構造体の複製を API ログに追加している。

```
CUresult cuModuleGetFunction(CUfunction *hfunc,CUmodule hmod,const char *name){
        CUresult res;
        PROLOGUE(cuModuleGetFunction);
        shm->u.cuModuleGetFunction.hmod = hmod;
        shm->u.cuModuleGetFunction.name = shm->data;
        strcpy(shm->data, name);
        shm->datalen = strlen(shm->data) + 1;
        shm->op = op_cuModuleGetFunction;
        send_cmd(0);
        recv_cmd();
        *hfunc = shm->u.cuModuleGetFunction.hfunc;
        res = shm->res;
        if (res == CUDA_SUCCESS) apilog_add(shm);
        EPILOGUE(cuModuleGetFunction);
        return res;
};
```

サーバプロセス側も同様にコードを生成する。サーバ側は共有メモリにある ID に対応する関数を呼び出し，共有メモリにある引数を指定して本来の API 関数を呼び出すシンプルなものである。

OpenACC 専用のソフトウェアとして実装することのメリットは少ない。開発中の同システムでは OpenACC 用の CUDA Driver API だけでなく，CUDA アプリケーション向けの CUDA Runtime API についても同様に対応する。OpenMP の target キーワードによる GPU 利用についても OpenACC と共通のランタイムライブラリが利用されるためそのまま対応できる。Fortran 2008 の do concurrent という並列化構文をコンパイラが GPU 実行箇所と解釈する Standard Parallelism というプログラミング手法もある。この場合には CUDA Managed memory という CPU からも GPU からもアクセス可能なメモリを利用する前提になっているが，GPU 用に別プロセスを用意しているチェックポイント用のソフトウェア実装ではオーバヘッドが大きく現実的ではない。このため do concurrent は対象外としている。OpenACC 等でも CUDA Managed memory を使用することはできるが，チェックポイントを使う場合には同様にオーバヘッドを増加させてしまうため非推奨である。CUDA Fortran では CUDA Runtime API を利用するという点では CUDA C/C++と同じである。CUDA Fortran には OpenACC のようにループ部分の GPU 化を容易にする”!$cuf kernel”指

示詞が用意されている。これを使用した場合は OpenACC と同じランタイムライブラリが追加で使用される。また CUBLAS，CUFFT 等の NVIDIA が提供する数値ライブラリなどをアプリケーションが使用する場合には追加の対応が必要になる。これらのライブラリは CUDA Driver API などを経由せず直接 GPU にアクセスしてしまうため，これらのライブラリも置き換えて別プロセスに中継する等が必要になる。このような複数の GPU プログラミング手法，各種ライブラリへの対応も今後進めていく予定である。

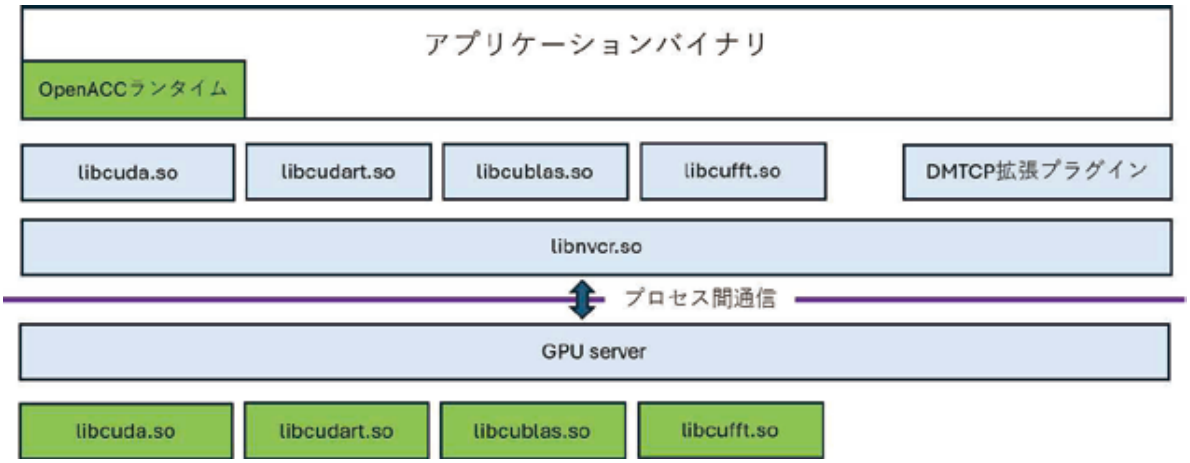


図 16　ソフトウェアの構成図

図 16 にソフトウェアの構成図を示す。アプリケーションバイナリと OpenACC のランタイムライブラリについてはそのまま使用し，その他のランタイムライブラリとの間に本ソフトウェアが割り込む。アプリケーションプロセスに読み込ませる各ランライムライブラリは共通の libnvcr.so に実装された API を通じてサーバプロセスと通信する。

[10] 鞍点型連立一次方程式に対する階層並列型数値解法の近似解精度改善（多田野）

鞍点型と呼ばれる連立一次方程式：

$$\begin{bmatrix} A & B \\ C^{\mathsf{T}} & O \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{g} \end{bmatrix} \tag{1}$$

は偏微分方程式に対するメッシュレス離散化法，構造解析，及び非圧縮流体解析などにおいて現れ，高速・高精度求解手法が必要とされている．ここで，$A \in \mathbb{R}^{n\times n}$ は正則な大規模疎行列，$B, C \in \mathbb{R}^{n\times m}$ $(n \geq m)$ は列フルランク行列，$O \in \mathbb{R}^{m\times m}$ は零行列であり，$\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^m$ は未知ベクトル，$\boldsymbol{f} \in \mathbb{R}^n, \boldsymbol{g} \in \mathbb{R}^m$ は既知ベクトルである．この鞍点型連立一次方程式の係数行列を構成する行列 B, C の列数 m が大きい場合，同方程式はクリロフ部分空間反復法による求解が極めて困難になることが知られている．我々はこの状況を打破するために，同方程式をそのまま解くのではなく，同方程式のブロック構造を利用して変形を施し，計算の主要部

を行列 A と複数の連立一次方程式の求解部分に帰着させる方法を開発した．この複数の右辺ベクトル間には互いに依存関係がないため，列方向に分割することで少数の右辺ベクトルをもつ複数の連立一次方程式に分割できる．分割された各方程式は同時に求解が可能であり，さらに各方程式も並列に求解可能であることから，本数値解法は階層型の並列性をもつ．複数右辺連立一次方程式を解いて得られた近似解を用いて鞍点型連立一次方程式の解ベクトルを計算するが，得られる近似解の精度が十分でないことがある．令和 6 年度は，鞍点型連立一次方程式の階層並列型数値解法の近似解精度改善を目標として研究を実施した．

鞍点型連立一次方程式 (1) に対する階層並列型数値解法のアルゴリズムを**図 17** に示す．計算の主要部は複数右辺ベクトルをもつ連立一次方程式 $A\hat{X} = \hat{B}$ の求解であるが，右辺項 $\hat{B} \in \mathbb{R}^{n\times(m+1)}$ を列方向に分割することにより，P 個の連立一次方程式：

$$A\hat{X}^{(j)} = \hat{B}^{(j)} \ (j = 0, 1, \ldots, P-1)$$

が得られる．ここで，$\hat{X}^{(j)}, \hat{B}^{(j)} \in \mathbb{R}^{n\times s}$, $s \equiv (m+1)/P$ である．これらを並列に解くことにより，高速化を図っている．得られた解行列 $\hat{X} \in \mathbb{R}^{n\times(m+1)}$ を用いて鞍点型連立一次方程式 (1) の解ベクトル $\boldsymbol{x}, \boldsymbol{y}$ を計算するが，精度が十分でない場合の精度改善手法が必要である．

Input: $A \in \mathbb{R}^{n\times n}, B, C \in \mathbb{R}^{n\times m}, \boldsymbol{f} \in \mathbb{R}^n, \boldsymbol{g} \in \mathbb{R}^m$
Output: $\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^m$

1. **Set** $\hat{B} = [B\ \boldsymbol{f}]$
2. **Solve** $A\hat{X} = \hat{B}$ for $\hat{X} = [\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, \ldots, \hat{\boldsymbol{x}}_m, \hat{\boldsymbol{x}}_{m+1}]$
3. **Set** $U = [\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, \ldots, \hat{\boldsymbol{x}}_m], \boldsymbol{v} = \hat{\boldsymbol{x}}_{m+1}$
4. **Compute** $S = C^{\mathsf{T}}U$ and $\boldsymbol{t} = C^{\mathsf{T}}\boldsymbol{v} - \boldsymbol{g}$
5. **Solve** $S\boldsymbol{y} = \boldsymbol{t}$ for $\boldsymbol{y}$
6. **Compute** $\boldsymbol{x} = \boldsymbol{v} - U\boldsymbol{y}$

図 17　鞍点型連立一次方程式に対する階層並列型数値解法のアルゴリズム

本研究では，階層並列型数値解法に反復改良法を導入することにより，近似解の精度改善を行った．鞍点型連立一次方程式 (1) の真の解を $\boldsymbol{x}^*, \boldsymbol{y}^*$，近似解を $\boldsymbol{x}, \boldsymbol{y}$, 補正項を $\Delta\boldsymbol{x}, \Delta\boldsymbol{y}$ とすると，以下の関係式が成り立つ．

$$\boldsymbol{x}^* = \boldsymbol{x} + \Delta\boldsymbol{x}, \qquad \boldsymbol{y}^* = \boldsymbol{y} + \Delta\boldsymbol{y}.$$

近似解 $\boldsymbol{x}, \boldsymbol{y}$ に対する残差を $\boldsymbol{r}, \boldsymbol{q}$ とすると，以下の関係式が得られる．

$$\begin{bmatrix} A & B \\ C^{\mathsf{T}} & O \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{y} \end{bmatrix} = \begin{bmatrix} \boldsymbol{r} \\ \boldsymbol{q} \end{bmatrix}. \tag{2}$$

即ち，鞍点型連立一次方程式 (2) を解くことで補正項 $\Delta\boldsymbol{x}, \Delta\boldsymbol{y}$ が得られ，近似解精度の改善が可能となる．本研究では，**図 17** のアルゴリズムで得られた近似解を初期解 $\boldsymbol{x}_0, \boldsymbol{y}_0$ として，以

下の計算を繰り返すことで近似解の精度向上を図る．

1. 補正項の計算：$\begin{bmatrix} A & B \\ C^{\top} & O \end{bmatrix}\begin{bmatrix} \Delta \boldsymbol{x}_k \\ \Delta \boldsymbol{y}_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_k \\ \boldsymbol{q}_k \end{bmatrix}$
2. 近似解の更新：$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta \boldsymbol{x}_k,\ \boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \Delta \boldsymbol{y}_k$
3. 残差の更新：$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - A\Delta \boldsymbol{x}_k - B\Delta \boldsymbol{y}_k,\ \boldsymbol{q}_{k+1} = \boldsymbol{q}_k - C^{\top}\Delta \boldsymbol{x}_k$

上述のとおり，補正項 $\Delta \boldsymbol{x}_k, \Delta \boldsymbol{y}_k$ を求めるには鞍点型連立一次方程式 (2) を解く必要があるが，**図 17** のアルゴリズムによる求解では再び複数右辺をもつ連立一次方程式の求解が必要となってしまう．補正項の具体形は，

$$\begin{cases} \Delta \boldsymbol{x}_k = A^{-1}\boldsymbol{r}_k - A^{-1}B\Delta \boldsymbol{y}_k \\ \Delta \boldsymbol{y}_k = (C^{\top}A^{-1}B)^{-1}(C^{\top}A^{-1} - \boldsymbol{q}_k) \end{cases}$$

で与えられる．ここで，行列 $U = A^{-1}B \in \mathbb{R}^{n\times m}$ と $S = C^{\top}A^{-1}B \in \mathbb{R}^{m\times m}$ は初期解 $\boldsymbol{x}_0, \boldsymbol{y}_0$ を求める際に計算されているため再利用が可能である．$\Delta \boldsymbol{y}_k$ を得るには行列 S を係数行列にもつ連立一次方程式を解く必要があるが，S が LU 分解されていれば前進・後退代入のみで計算が可能である．以上より，反復改良法部分において新たに必要になるのはベクトル $\boldsymbol{v}_k \equiv A^{-1}\boldsymbol{r}_k \in \mathbb{R}^n$ であるが，これは連立一次方程式 $A\boldsymbol{v}_k = \boldsymbol{r}_k$ を解くことにより求める．

数値実験により反復改良法付き階層並列型数値解法の性能を評価する．行列 A として，SuiteSparse Matrix Collection で公開されている行列 poisson3Db（サイズ n：85,623，非零要素数：2,374,949）と torso3（サイズ n：259,156，非零要素数：4,429,042）を用いた．行列 B, C は乱数行列で与え，列数 m は $m = 4, 8, \ldots, 5000$ とした．また，反復改良法の反復回数は，0, 1, 2 とした．実験環境として，筑波大学計算科学研究センターのスーパーコンピュータ Pegasus の 4 ノードを用い，1 ノードあたりの MPI プロセス数は 4 とし，1 プロセスあたりの OpenMP スレッド数は 12 とした．性能評価は以下の観点で行った．

観点 1：鞍点型連立一次方程式の近似解の相対誤差：$\varepsilon = \left\| \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{y}^* \end{bmatrix} - \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} \right\|_2 \Big/ \left\| \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{y}^* \end{bmatrix} \right\|_2$

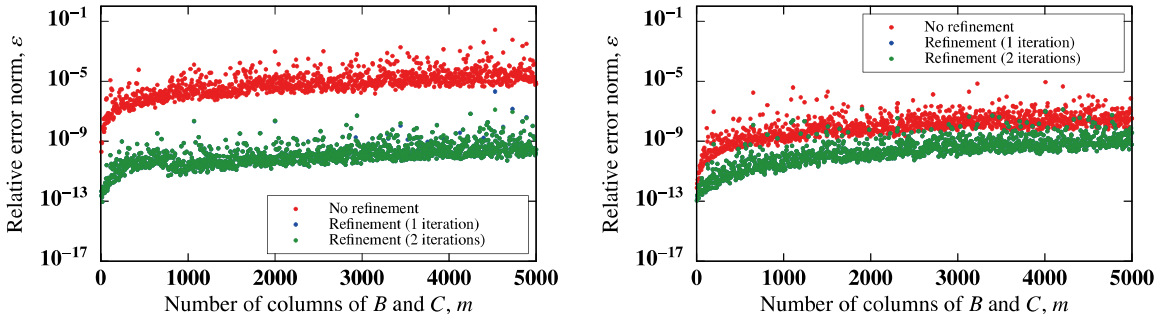
観点 2：鞍点型連立一次方程式の真の相対残差：$\delta = \left\| \begin{bmatrix} \boldsymbol{f} - A\boldsymbol{x} - B\boldsymbol{y} \\ \boldsymbol{g} - C^{\top}\boldsymbol{x} \end{bmatrix} \right\|_2 \Big/ \left\| \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{g} \end{bmatrix} \right\|_2$

観点 3：鞍点型連立一次方程式の求解に要した時間

図 18 に，行列 B, C の列数 m の変化に対する鞍点型連立一次方程式の近似解の相対誤差 ε の変化を示す．2 つのテスト行列について，反復改良法を適用することにより，近似解の相対誤差を小さくなっていることがわかる．行列 poisson3Db では，反復改良法により近似解精度が 2〜6 桁程度，torso3 では 1〜3 桁程度近似解精度が向上した．行列により反復改良法の効果に差が見られるものの，近似解精度を向上させることができた．

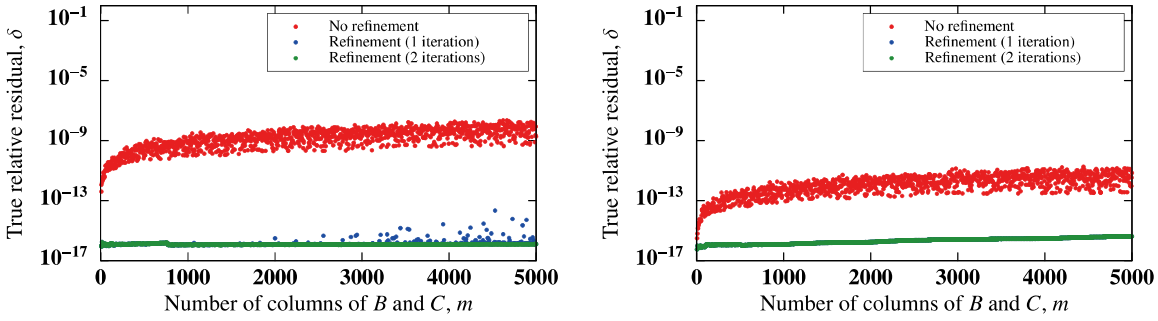
図 19 に，行列 B, C の列数 m の変化に対する鞍点型連立一次方程式の真の相対残差 δ の変化を示す．同図に示すように，反復改良法を適用することにより真の相対残差の値も十分小さくできている．poisson3Db については，1 回の反復改良法の適用では不十分な箇所が見られたが，2 回適用することで真の相対残差をさらに小さくすることができた．

行列 B, C の列数 m の変化に対する鞍点型連立一次方程式の求解時間の変化を，**図 20** に示す．同図より，反復改良法を適用した場合であっても，両テスト行列ともに計算時間を大幅に増加させることなく近似解精度改善を実行できることがわかる．行列 poisson3Db では，反復改良法の適用なしの計算時間と比較して，1 回反復，2 回反復の計算時間はそれぞれ，平均で約 1.02 倍，約 1.03 倍に増加し，行列 torso3 ではそれぞれ，平均で約 1.02 倍，約 1.02 倍に増加した．



(a) (b) torso3.

図 18　行列B, Cの列数mの変化に対する近似解の相対誤差の変化



(a) (b) torso3.

図 19　行列B, Cの列数mの変化に対する真の相対残差の変化

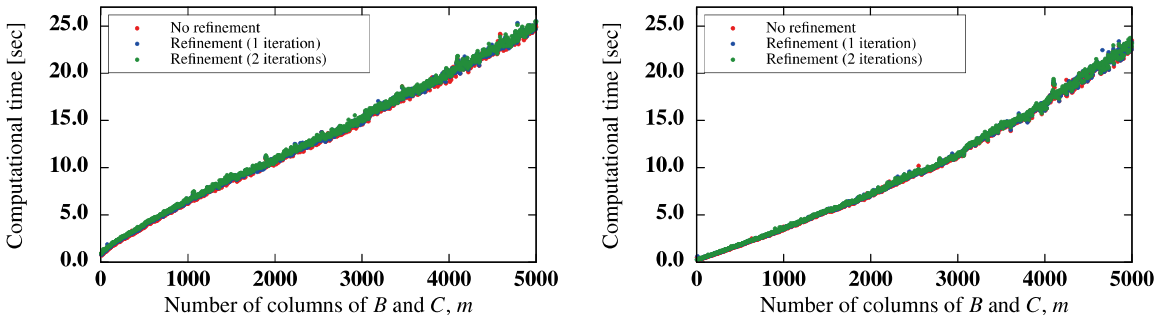


図 20　行列B, Cの列数mの変化に対する求解時間の変化

[11] ブラックホール降着円盤シミュレーションの GPU 加速（小林，朴）

ブラックホールの降着円盤は，X 線連星，活動銀河核，ガンマ線バーストなどの高エネルギー現象を駆動する中心的なエンジンとして機能し，磁気回転不安定性による乱流，激しい加熱，相対論的ジェットの形成など，複雑な物理現象を発生させる。伝統的な一次元理論モデルと一般相対論的磁気流体力学（GR-MHD）シミュレーションは，これらの現象の多くの側面の再現に成功してきたが，高吸積率環境で重要な役割を果たす放射冷却と放射圧の効果を再現する点で不十分であった。この限界を克服するため，輻射とプラズマ動力学の相互作用を組み込んだ一般相対論的放射磁気流体力学（GR-RMHD）シミュレーションが開発された。しかし，輻射輸送方程式の導入は計算コストを大幅に増加させるため，高精度多次元シミュレーションを実現するためには高度な計算戦略が必要となる。

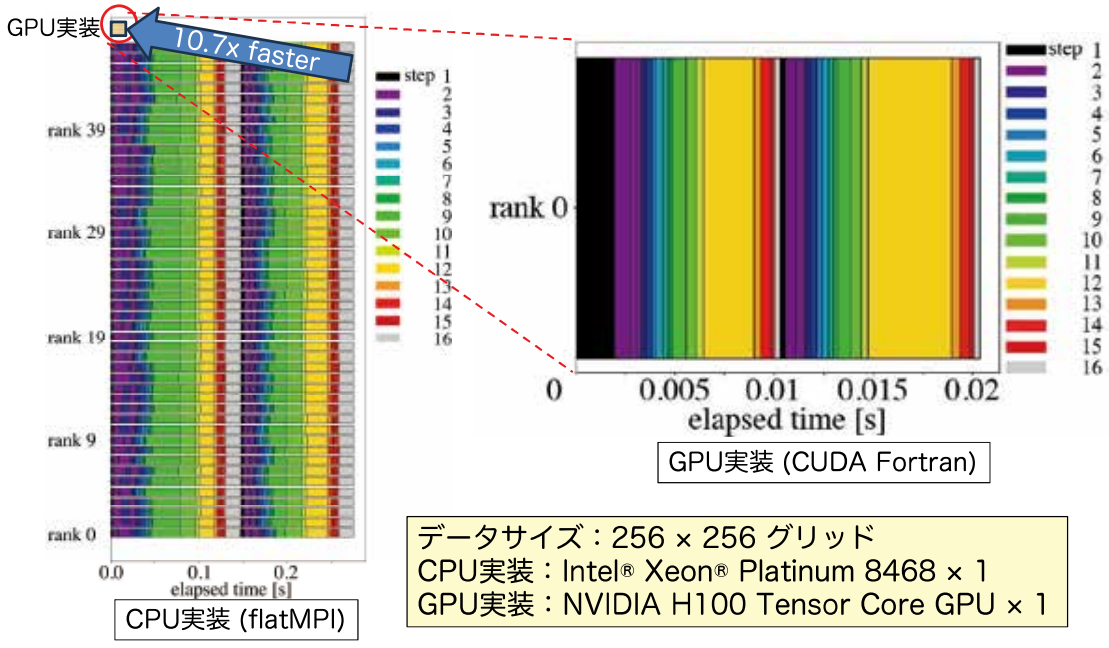


図 21　CPU 実装との性能比較

そこで本研究では，宇宙物理部門と連携して CUDA Fortran と MPI を用いた GR-RMHD シミュレーションコードを開発し，flatMPI をベースとした CPU 実装とシミュレーションコードの実行速度を比較した．**図 21** は，GR-RMHD シミュレーションコードをデータサイズ 256×256 グリッドで実行した際の CPU 実装および GPU 実装の性能比較を示している．CPU 実装では，Intel® Xeon® Platinum 8468 を使用しており，48 個の MPI プロセス (1 プロセス / コア) をマッピングしている．シミュレーションコードにおいて支配的な処理は step 12 であり，並列処理を行ったとしても総実行時間の約 30%を占める．一方，step 9 も step 12 とほぼ同程度の時間を要しており，これはダイナモ項の積分計算を行うために MPI Allreduce 関数が呼び出されるためである．OpenMP を用いた実装ではこの実行時間の短縮は可能であるが，最大でも約 1.4 倍の高速化にとどまる．一方，GPU 実装ではほぼ全ての処理が加速され，CPU 実装と比べて最大 10.7 倍の高速化を達成した．この演算加速により，ブラックホール降着円盤のダイナミクスやジェット形成のより詳細な解析が可能となり，高エネルギー天体のエネルギー生成機構に関する新たな知見の獲得が期待される．

[12] 並列 FPGA システムによる深層学習推論処理の高速化（小林，藤田，朴）

近年，深層学習モデルを用いた画像認識や音声認識，自然言語処理といったタスクは，データセンターからエッジデバイスに至るまで幅広い領域で急速に普及している．とりわけ，推論フェーズにおいては，低遅延かつ高スループットを同時に実現することが強く求められるが，従来の GPU ベース実装ではリアルタイム性の確保やレイテンシ変動の抑制において依然として課題が残されている．

そこで本研究では，FPGA のパイプライン並列性と低レイテンシ性を活用し，かつ既存のソフトウェア開発手法との親和性を維持しつつ，マルチ FPGA を並列利用することによる深層学習推論処理の高速化手法を提案した．**図 22** にその概要を示す．既存の OpenCL ベース CNN アクセラレータである PipeCNN をベースに，ResNet-50 の畳み込み層や残差加算層を各 FPGA 上に実装し，FPGA 間のデータ転送を我々が開発した OpenCL から制御可能な FPGA 間通信フレームワークである CIRCUS で実行することによって，ユーザは OpenCL の抽象度に留まったまま，マルチ FPGA に跨がる CNN 推論ハードウェアを実現できる．

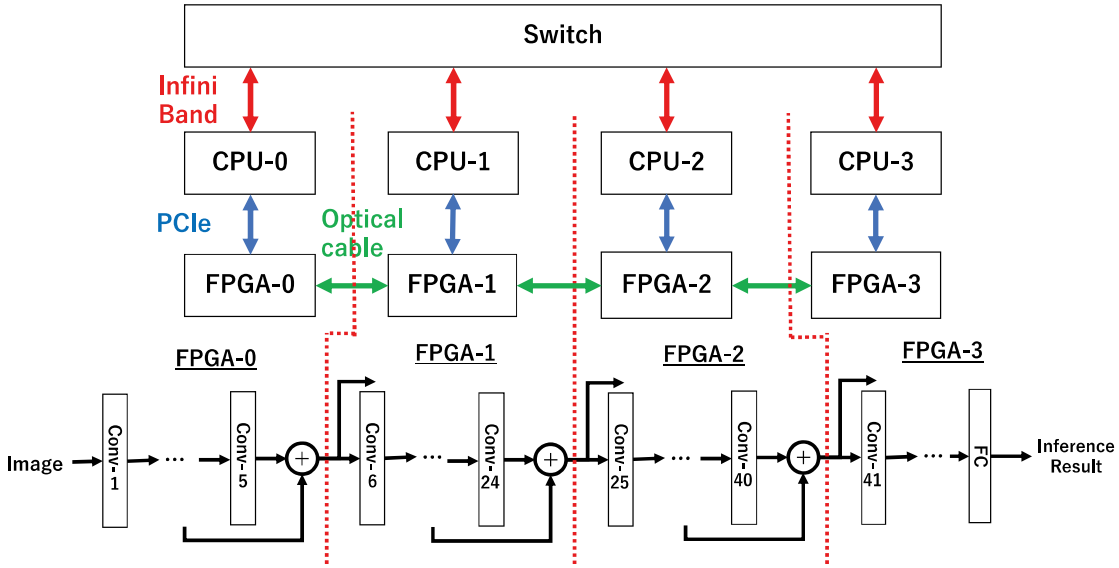


図 22　ResNet-50 推論モデルのマルチ FPGA 実装

提案手法の評価は，BittWare 520N FPGA ボードを最大 4 枚使用した．**図 23** は FPGA ボードの台数に応じた推論レイテンシおよびスループットを示す. 図中の「No data parallelism」とは各 FPGA に実装された ResNet-50 のデータパスが 1 つのみ，「Data parallelism = 6」とはそのデータパスが 6 つ複製された構成であることを示す．データパスの複製により後者は複数の入力バッチを同時に処理するデータ並列を活用することが可能となり，複製数が多くなればなるほど，推論スループットは向上する．しかしながら，複製によるスループット向上はある一定の複製数までしか効果がなく，これは複製したデータパス間で FPGA の外部メモリの競合が発生するためである．したがって，この状態から更に推論スループットを向上させるためには複数の FPGA を活用したモデル並列を併用することが有効であり，**図 23** の右側はその証左となっている．ただし，並列効率は 50 %程度に留まっており，これは外部メモリの競合以外に，複製したデータパスの間で，FPGA 間通信のためのチャネル競合が発生しているためである．そのため，No data parallelism の構成では，レイテンシは FPGA の数によらずほぼ一定に保たれるが，Data parallelism = 6 の構成では，FPGA の数を増やす毎にレイテンシが 30 msec ずつ増加している．これらの外部メモリ競合の緩和や通信アルゴリズムのさらなる最適化が今後の課題である．

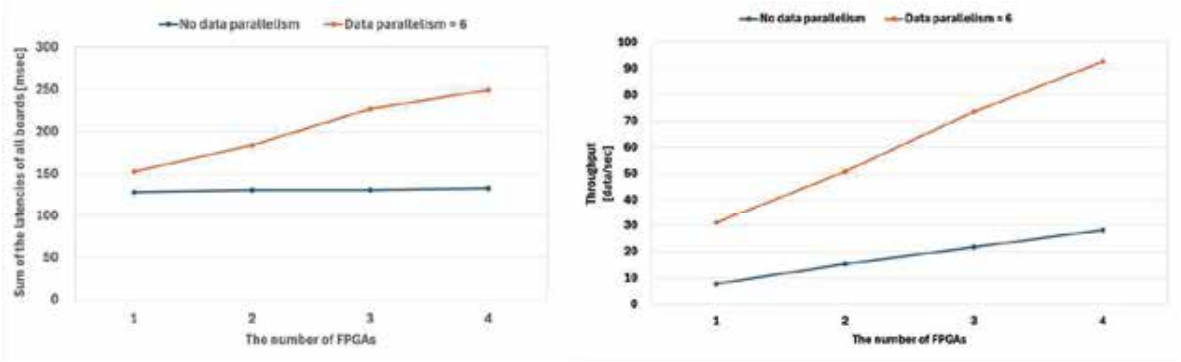


図 23　推論レイテンシおよび推論スループットの評価

[13] 並列 FPGA 環境における FPGA 間通信に関する研究（藤田，小林，朴）

筑波大学計算科学研究センターでは，直接接続された複数 FPGA 上で利用可能な通信フレームワーク CIRCUS (Communication Integrated Reconfigurable CompUting System) を開発している。しかしながら，CIRCUS はフロー制御が実装されておらず，アプリケーションで通信バッファの残量を管理しなければバッファが溢れ，通信が消失するという制限がある。この制限の影響により，これまでの研究で集団通信のような複雑な通信を CIRCUS 上に実装することが困難であることが明らかとなっていた。本年度は，この問題を解決するために CIRCUS の通信プロトコルの改善に取り組んだ。

フロー制御の問題を解決するために，オープンソースの FPGA 間通信プロトコルである Kyokko を CIRCUS に組み込むことを検討している。これまでの研究で，我々が対象としている FPGA ボードにおける動作確認を完了しており，本年度は CIRCUS への適用を行う。Kyokko を CIRCUS に組み込むために，両者を接続し Kyokko 通信の制御を行うモジュールの開発を行った。また，ルータ内部でパケット交換を行っているクロスバを改良し，フロー制御へ対応できるようにした。

表 2 と**図 24** に Kyokko-CIRCUS の通信性能評価の結果を示す。通信レイテンシについては，Kyokko-CIRCUS はフロー制御がない旧実装よりも約 40ns 早いという結果が得られた。また，通信帯域については，100Gbps の FPGA ボード間の通信環境で最大 83Gbps の帯域が得られた。古い実装では CIRCUS が通信は 90Gbps 以上の帯域が得られており，Kyokko-CIRCUS は十分な通信帯域が得られていない。より大きなメッセージ長を通信すれば，さらに高い性能が実現できると考えられる。しかしながら，256KB よりも大きい通信を行うと，ベンチマークが安定して動作しないという問題があり，性能測定が行えていない。不具合により通信データが破損したり消失したりしている可能性が高く，現在原因を調査中である。

表 2. 通信レイテンシの比較

	Kyokko（新実装・フロー制御あり）	SL3（旧実装・フロー制御なし）
1-hop	463 ns	500 ns
2-hops	717 ns	749 ns
増加分	+254 ns	+250 ns

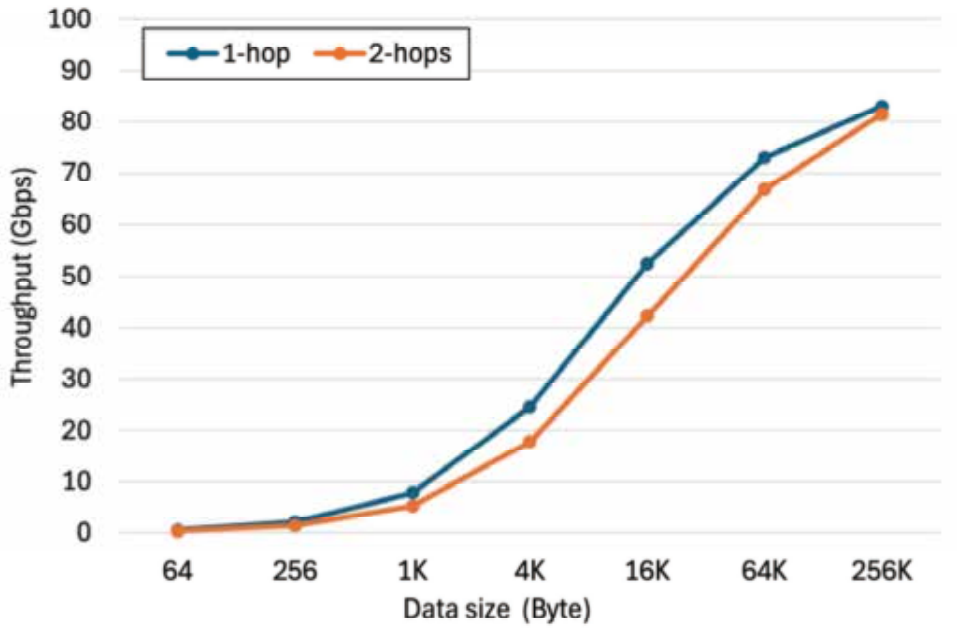


図 24　Kyokko-CIRCUS の通信性能評価の結果（通信バンド幅）

[14] 複数の演算加速装置を統一的に扱えるプログラミング環境に関する研究（藤田，小林，朴）

科学技術計算の分野ではスーパーコンピュータを用いて大規模な計算が行われているが，性能の向上に伴って消費電力も増大していることが問題となっており，単に性能が高いだけでなく，電力あたりの性能(Performance per watt)が高いことも重要視されている。電力問題を解決するために，演算加速装置(アクセラレータ)を搭載する計算機が広く用いられている。科学技術計算の分野では Graphics Processing Unit (GPU)が演算加速装置として広く用いられている。

演算加速装置の普及に伴い，多様性が増しているという問題が存在する。GPU だけでも複数ベンダーの製品が存在し，それぞれのベンダーが別々の開発環境を提供しており，開発環境が分断されている状況である。アプリケーションを複数の GPU で実実行可能とするためには，それぞれの GPU に対して個別のプログラミングが必要となり，コストが非常に高い。複数の演算加速装置を統一的に扱える開発環境のニーズが高まっている。本

研究の目的は，複数の演算加速装置を統一的に扱えるプログラミング環境を実現することである。本目的を達成するためには，プログラミング言語面だけでなく，様々な演算加速装置を扱うためのソフトウェアの整備が求められる．我々は本目的を実現するために「UniSYCL」開発環境を研究開発している。本研究は，米国 Oak Ridge National Laboratory (ORNL) と共同で実施しており，複数の演算加速装置環境(Multi Heterogeneous)を扱えるランタイムフレームワーク「IRIS」を用いて複数種類対応を実現している。

中性子輸送をモンテカルロ法でシミュレーションするベンチマークコード XSBench に UniSYCL を適用し性能評価をした結果を**図 25** に示す。NVIDIA GPU と AMD GPU が同じ計算機に搭載された環境で実験しており，IRIS が処理を複数の GPU に自動で振り分ける。なお，IRIS によるスケジューリングを行えるようにするために，XSBench のコードを変更しタスク分割のモデルを用いて計算を実行するように変更している。今後は，実アプリケーションを持ちた性能評価を行う予定である。

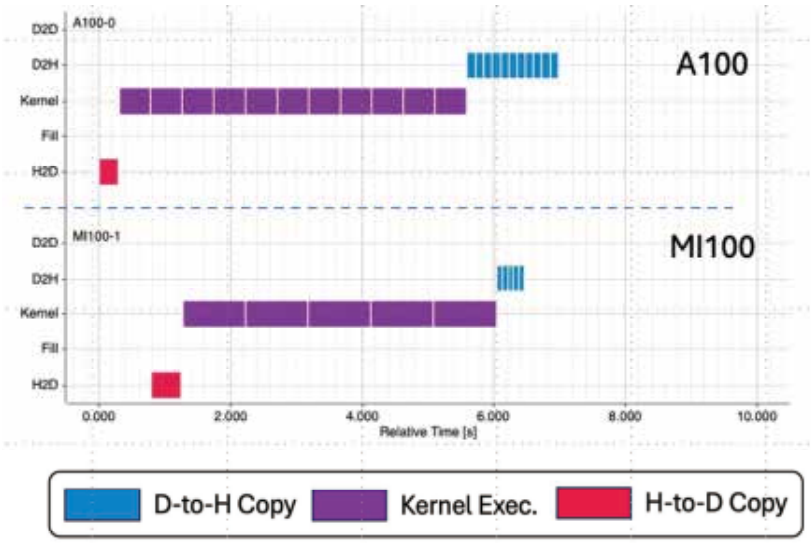


図 25　UniSYCL を用いた XSBench を実行する際のタイムライン。NVIDIA A100 GPU と AMDMI100 GPU が搭載された環境で実行し，IRIS が自動的に異なる GPU に処理を振り分けている。

[15] CPU-GPU 密結合モジュールにおけるメモリシステムの性能評価に関する研究（藤田，朴）

2025 年 1 月から稼働開始した JCAHPC の新スーパーコンピュータ Miyabi-G では各計算ノードに NVIDIA GH200 モジュールが搭載されており，Grace CPU と Hopper GPU (H100) による共有メモリプログラミングが可能である。従来の GPU クラスタの計算ノードでは，CPU と GPU は別のデバイスであり，PCI Express (PCIe) バスを通して接続されるのが一般的で，PCIe バスの転送速度がボトルネックとなり，さらに両デバイスのメモリアド

レス空間が独立で，プログラムがやりにくいという問題があった。GH200 では，CPU-GPU を 1 つのモジュールとして構築し，その間を NVLink-C2C という Proprietary なバスで接続している。PCIe バスの帯域が 64GB/s であるところ，NVLink-C2C は 450GB/s の帯域（PCIe バスの 7 倍）で CPU-GPU 間を接続しており，通信のボトルネックが解消され，性能向上が期待できる。また，GH200 のメモリアーキテクチャは，CPU と GPU の各メモリドメインが NUMA 構造で結合されているとみなせる。ただし，Grace CPU のメモリである LPDDR5X のピークバンド幅が 512GB/s であるのに対し，Hoppe GPU (H100)のそれは HBM3 技術により 4TB/s に大幅に増強されており，このような性能的に非対称の NUMA は一般的な CPU のそれとは違い，極めて特殊である。

GH200 では System Allocated Memory (SAM) と呼ばれる，Unified Memory (UM) をより高度に用いるアーキテクチャ上の機能が実装されている。GH200 においても，CPU と GPU がそれぞれメモリを持つ構造は従来環境と同じであるが，NVLink-C2C を通して CPU-GPU 間で Cache Coherent が保たれ，アクセス頻度に応じたメモリ間の自動データ移動（Migration）が可能である。SAM は，malloc 関数など通常の（CUDA API ではない）関数を用いて確保したメモリが，CPU と GPU の両方からアクセスできる UM 実装である。

本研究では，GH200 のメモリシステムおよび SAM に関する性能評価を行う。SAM を用いると，データ移動に関するプログラミングが不要になるため，アプリケーションの GPU 化を容易に行えるようになることが期待されている。しかしながら，GH200 は新しい NVIDIA GPU システムであるため，SAM の性能は十分に評価されていない。

図 26 に性能評価結果を示す。これは，CPU メモリ（LPDDR5X）に配置した SAM メモリ領域を GPU が複数回アクセスした時の性能変化を示したものである。初めは全ての領域が CPU メモリにあるため性能が低いが，Migration によって徐々に GPU メモリ（HBM3）に移動していき性能が向上する様子を示したものである。データ移動はシステムによって自動的に行われる。Migration の進行に伴って性能が高くなるが，Migration が完了したとしても，従来手法（SAM ではない）で確保したメモリ領域より性能が 10〜20%低いことが判明した。より詳細な性能解析および挙動解析は今後の課題である。SAM を利用することにより，ユーザは両デバイス間のデータの移動を明示的に行う必要がなくなり，かつ GPU から見ると HBM3 の高バンド幅メモリをデータの場所を意識せずに利用できるため，プログラミング上の利便性が向上すると考えられる。実アプリケーションで SAM を利用した際の性能評価について今後実施する予定である。

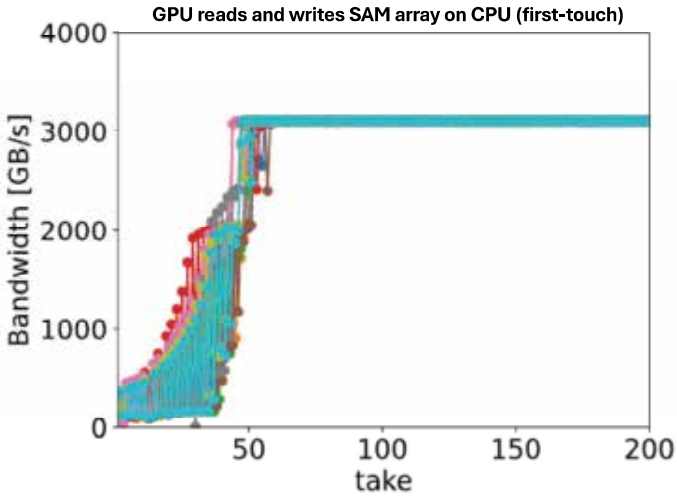


図 26　SAM のメモリアクセス性能評価（LPDDR5X→HBM3 の Migration あり）

一方, 従来の NVIDIA GPU と InifniBand による相互結合網では, PCIe バスを活かした GPU と HCA インタフェースの間で直接データ転送を行う GDR (GPU Direct RDMA)を用いて，特に短いメッセージのノード間通信において CPU メモリを介さない低レイテンシ通信が可能であった。しかし，NVLINK-C2C で GPU と CPU が結合され，さらに PCIe バスが CPU 側に実装されている GH200 では，GDR がどのように働くのか，そもそも機能するのかも定かではない。そこで， MPI によるノード間通信について，cudaMalloc+GDR と SAM で GDR を意識しない通信を行った場合について比較した。CPU からの MPI 通信と 2 つの方式を比較した相対性能を**エラー! 参照元が見つかりません。**に示す。1MB までの小規模メッセージでの性能は後者の方が前者より 10〜20%程度高かった。これは「GDR は短メッセージ長で高速」という従来の予想を裏切るものであった。こちらについても詳細は調査中であるが，CUDA 上のメモリを意識せず SAM に任せてデータを確保し，GPU で計算を続けていれば MPI 通信性能も劣化せずむしろ向上するという可能性を示している。本研究の内容は 2025 年 5 月の情報処理学会 HPC 研究会にて発表予定である。

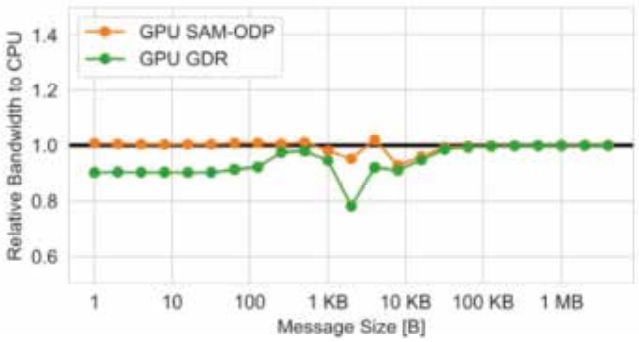


図 27　SAM+ODP と cumaMalloc+GDR の MPI 性能比較

4. 教育

1. Wentao Liang，修士（工学），A research on the programming environment for multi-heterogeneous accelerated computing，筑波大学大学院理工情報生命学術院システム情報工学研究群修士論文，令和 7 年 3 月（指導：朴泰祐）
2. 北爪 開人，修士（工学），高性能計算における FPGA 間通信のフロー制御に関する研究，筑波大学大学院理工情報生命学術院システム情報工学研究群修士論文，令和 7 年 3 月（指導：朴泰祐）
3. 鈴木 拓実，修士（工学），並列 FPGA システムによる深層学習推論処理の高速化，筑波大学大学院理工情報生命学術院システム情報工学研究群修士論文，令和 7 年 3 月（指導：朴泰祐）
4. 丸山 泰史，修士（工学），分散ファイルシステムのためのローカル SSD 最適化の提案と評価，筑波大学大学院理工情報生命学術院システム情報工学研究群修士論文，令和 7 年 3 月（指導：建部修見）
5. 落道 智也，修士（工学），Intel Advanced Matrix Extensions（AMX）を用いた行列乗算，筑波大学大学院理工情報生命学術院システム情報工学研究群修士論文，令和 7 年 3 月（指導：高橋大介）
6. 長橋 朋也, 修士(工学), MPI/OpenMP 並列化によるスライドパズルの Zero-Aware Pattern Database の構築, 筑波大学大学院理工情報生命学術院システム情報工学研究群修士論文, 令和 7 年 3 月（指導：高橋大介）
7. 関 拓己，修士（工学），Fortran アプリケーションの半自動 GPU 化，筑波大学大学院理工情報生命学術院システム情報工学研究群修士論文，令和 7 年 3 月（指導教員：額田彰）
8. 前田 椋祐，学士（情報工学），非同期 I/O と非同期通信処理を統合管理するスケジューリング設計の検討，筑波大学情報学群情報科学類卒業論文，令和 7 年 3 月（指導：建部修見）
9. 宮内 遥楓，学士（情報工学），ユーザ空間並列ファイルシステムのためのシステムコールフックライブラリの設計と評価，筑波大学情報学群情報科学類卒業論文，令和 7 年 3 月（指導：建部修見）
10. 小浦方 基，学士（情報科学），RPC の高速化のためのシリアライズ方式の検討，筑波大学情報学群情報科学類卒業論文，令和 7 年 3 月（指導：建部修見）
11. 竹島 駿介，学士（情報工学），OpenACC アプリケーションを対象とするチェックポイント機能の実現，筑波大学情報学群情報科学類卒業論文，令和 7 年 3 月（指導教員：額田彰）

12. 渡邊 瑞己，学士（情報科学），OpenACC アプリケーションにおけるホスト・デバイス間通信の遊休 GPU を利用した高速化，筑波大学情報学群情報科学類卒業論文，令和 7 年 3 月（指導教員：額田彰）
13. 阿部 崇人，学士（情報科学），GPU 化都市気象コードにおけるドライミスト効果機能の拡張，筑波大学情報学群情報科学類卒業論文，令和 7 年 3 月（指導：朴泰祐，藤田典久）
14. 堀之内 航，学士（情報工学），都市気象シミュレーションコードの構造データ生成の高速化に関する研究，筑波大学情報学群情報科学類卒業論文，令和 7 年 3 月（指導：朴泰祐，藤田典久）
15. 白井 拓翔，学士（情報工学），CPU・GPU Unified Memory を持つ GPU における MPI 通信に関する研究，筑波大学情報学群情報科学類卒業論文，令和 7 年 3 月（指導：藤田典久，朴泰祐）
16. 吉田 智，学士（情報工学），GPU・CPU 一体型モジュールにおける Unified Memory 使用時の性能評価，筑波大学情報学群情報科学類卒業論文，令和 7 年 3 月（指導：藤田典久，朴泰祐）

5. 受賞，外部資金，知的財産権等

受賞

1. 北爪 開人，リコンフィギャラブルシステム研究会優秀講演賞（若手講演賞），2024 年 11 月

外部資金

1. 科研費基盤研究（A），朴泰祐（代表），R3～R6 年度，6,900 千円，「多重複合演算加速機構を用いた次世代スーパーコンピューティング」
2. 科研費基盤研究（A），建部修見（代表），R4～R8 年度，7,350 千円，「次世代ストレージアーキテクチャの研究」
3. NEDO，建部修見（分担），R5～R7 年度，32,498 千円，「超分散コンピューティング基盤の研究開発」
4. 特別共同研究，建部修見（代表），R4～R7 年度，31,667 千円，「スケーラブルなデータストア及び高速データ処理に関する共同研究事業」
5. 文部科学省委託，建部修見（分担），R6 年度，32,670 千円，「HPCI の運営（HPCI 共用ストレージ用大規模分散ファイルシステムの機能整備など）」
6. 科研費基盤研究（C），高橋大介（代表），R4～R6 年度，1,000 千円，「メニーコア超並列クラスタにおける多倍長演算に関する研究」

7. 文科省委託研究，高橋大介，建部修見（分担），R6 年度，6,000 千円，「次世代計算基盤に係る調査研究」
8. 科研費基盤研究（C），多田野寛人（代表），R5～R7 年度，1,200 千円，「鞍点型連立一次方程式に対する高速・高精度階層並列型解法の開発」
9. 科研費若手研究，小林諒平（代表），R4～R6 年度，600 千円，「GPU・FPGA 複合型グラフ構造データ分析基盤の創出」
10. JST 先端国際共同研究推進事業「次世代のための ASPIRE」，小林諒平（分担），R5～R8 年度，2,800 千円，「次世代の高効率計算基盤を実現する適応型データ圧縮ハードウェアの探求」
11. 科研費基盤研究（B），小林諒平（分担），R6～R9 年度，300 千円，「２温度一般相対論的輻射磁気流体計算で解くブラックホール流の構造・電子温度・放射」
12. 科研費基盤研究（C），小林諒平（分担），R5～R7 年度，300 千円，「多要素協調型 Approximate Computing 実現に向けた HPC アプリケーション解析手法」
13. 科研費基盤研究（C），藤田典久（代表），R6～R8 年度，1,500 千円，「複数の演算加速装置に対応できる次世代高性能システムのためのプログラミング環境」

6. 研究業績

(1) 研究論文

A) 査読付き論文

1. Taisuke Boku, Masataka Sugita, Ryohei Kobayashi, Shinnosuke Furuya, Takuya Fujie, Masahito Ohue, Yutaka Akiyama, "Improving Performance on Replica-Exchange Molecular Dynamics Simulations by Optimizing GPU Core Utilization", Proceedings of 53rd International Conference on Parallel Processing (ICPP), pp. 1082-1091, 10.1145/3673038.3673097, 2024
2. Hiroyuki Kusaka, Ryosaku Ikeda, Takuto Sato, Satoru Iizuka, Taisuke Boku, "Development of a Multi-Scale Meteorological Large-Eddy Simulation Model for Urban Thermal Environmental Studies: The "City-LES" Model Version 2.0", Journal of Advances in Modeling Earth Systems, Vol. 16, Issue 10, 10.1029/2024MS004367, 2024
3. Yutaka Watanabe, Miwako Tsuji, Taisuke Boku, Mitsuhisa Sato, "Design and performance evaluation of UCX for the Tofu InterconnectD on Fugaku towards efficient multithreaded communication", The Journal of Supercomputing, Vol. 80, pp. 20715-20742, 10.1007/s11227-024-06201-x, 2024
4. Kohei Hiraga, Osamu Tatebe, "PEANUTS: A Persistent Memory-Based Network Unilateral Transfer System for Enhanced MPI-IO Data Transfer", Proceedings of 30th International

European Conference on Parallel and Distributed Computing (Euro-Par), pp. 439-453, 10.1007/978-3-031-69766-1_30, 2024

5. Sohei Koyama, Kohei Hiraga, Osamu Tatebe, "FINCHFS: Design of Ad-Hoc File System for I/O Heavy HPC Workloads", Proceedings of IEEE International Conference on Cluster Computing (CLUSTER), pp. 440-450, 10.1109/CLUSTER59578.2024.00045, 2024
6. Daisuke Takahashi, "Parallel Implementation of Number-Theoretic Transform on GPU Clusters", Proceedings of 24th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2024), Part III, Lecture Notes in Computer Science, Vol. 15253, pp. 204-218, 10.1007/978-981-96-1542-1_12, 2025
7. Daisuke Takahashi, "On the Division in the Computation of Binary BBP-Type Formulas for Mathematical Constants", Proceedings of 4th International Conference on Numerical Computations: Theory and Algorithms (NUMTA 2023), Part II, Lecture Notes in Computer Science, Vol. 14477, pp. 323-330, 10.1007/978-3-031-81244-6_32, 2025
8. Shota Kawakami, Daisuke Takahashi, "Implementation and Evaluation of Octuple-Precision Fast Fourier Transform on GPU", Proceedings of 2024 IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA 2024), pp. 287-294, 10.1109/ISPA63168.2024.00044, 2024
9. Hongjian Zhang, Akira Nukada, Qiucheng Liao, “FCUFS: Core-Level Frequency Tuning for Energy Optimization on Intel Processors”, Prooceedings of IEEE International Conference on Cluster Computing (CLUSTER), pp. 214-225, 10.1109/CLUSTER59578.2024.00026, 2024
10. Ryohei Kobayashi, Hiroyuki R. Takahashi, Akira Nukada, Yuta Asahina, Taisuke Boku, Ken Ohsuga, "Accelerating General Relativistic Radiation Magnetohydrodynamic Simulations with GPUs", Proceedings of International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia). pp. 72–79, 10.1145/3712031.3712032, 2025
11. Hiroto Tadano, “Performance evaluation of a hierarchical parallel solver with iterative refinement for saddle point problems in a parallel computing environment”, Proceedings of 43rd JSST Annual International Conference on Simulation Technology (JSST2024), pp. 339-342, 2024
12. Takumi Suzuki, Ryohei Kobayashi, Norihisa Fujita, Taisuke Boku, “Accelerating Deep Learning Inference with a Parallel FPGA System”, HEART '25: Proceedings of the 15th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies, pp.49–56, 10.1145/3728179.3728186, 2025
13. Norihisa Fujita, Beau Johnston, Narasinga Rao Miniskar, Ryohei Kobayashi, Mohammad Alaul Haque Monil, Keita Teranishi, Seyong Lee, Jeffrey S. Vetter, Taisuke Boku, “CHARM-

SYCL & IRIS: A Tool Chain for Performance Portability on Extremely Heterogeneous Systems", Proceedings of 2024 IEEE 20th International Conference on e-Science (e-Science), pp. 1-10, 10.1109/e-Science62913.2024.10678717, 2024

B) 査読無し論文

1. Mingzhe Yu, Osamu Tatebe, Distributed K-FAC Over Unstable Networks (unreferred), 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. 2024-HPC-195, No. 13, pp. 1-7, 2024 年 8 月
2. Mingzhe Yu, Osamu Tatebe, Asynchronous Decentralized Distributed K-FAC: Enhancing Training Efficiency and Load Balancing in Heterogeneous Environments (unreferred), 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. 2024-HPC-197, No. 5, pp. 1-10, 2024 年 12 月
3. 前田 椋祐，中野 将生，建部 修見，分散ファイルシステムにおける通信イベントと I/O イベントの非同期スケジューリングを統合した非同期 I/O の性能評価，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）, Vol. 2024-HPC-197, No. 16, pp. 1-9, 2024 年 12 月
4. 高橋 大介，数学定数に対する 2 進 BBP 型公式の計算における除算について，日本応用数理学会 2024 年度年会講演予稿集，2024 年 9 月
5. 高橋 大介，GPU クラスタにおける並列数論変換の実現と評価，日本応用数理学会 2024 年度年会講演予稿集，2024 年 9 月
6. 長橋 朋也，高橋 大介，MPI/OpenMP 並列化によるスライドパズルの Zero-Aware Pattern Database の構築，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. 2024-HPC-195，No. 23, pp. 1-6, 2024 年 8 月
7. 川上 昌汰，高橋 大介，GPU における 8 倍精度高速フーリエ変換の実装と評価，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）, Vol. 2024-HPC-194，No. 2, pp. 1-7, 2024 年 5 月
8. 塙 敏博，建部 修見，中島 研吾，朴 泰祐，下川辺 隆史，三木 洋平，山崎 一哉，住元 真司，高橋 大介，額田 彰，多田野 寛人，小林 諒平，藤田 典久，成瀬 彰，GH200 の予備性能評価，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）, Vol. 2024-HPC-195, No. 4, pp. 1-11, 2024 年 8 月
9. 小林 諒平，高橋 博之，額田 彰，朝比奈 雄太，朴 泰祐，大須賀 健，GPU 演算加速による一般相対論的輻射磁気流体シミュレーションコードの性能評価，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）, Vol. 2025-HPC-198, No. 60, pp. 1-8, 2025 年 3 月

10. 多田野 寛人，並列計算環境における鞍点型連立一次方程式に対する反復改良付き階層並列型解法の性能評価，日本応用数理学会 2024 年度年会講演予稿集，2 pages, 2024 年 9 月
11. 北爪 開人，藤田 典久，小林 諒平，朴 泰祐，並列 FPGA 間通信フレームワーク CIRCUS へのフロー制御の実装と評価，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）, Vol. 2025-HPC-198, No. 58, pp. 1-9, 2025 年 3 月
12. 北爪 開人，上野 知洋，吉井 一友，木山 真人，藤田 典久，小林 諒平，佐野 健太郎，朴 泰祐，適応型帯域圧縮ハードウェアプラットフォームの Chisel 実装と評価，リコンフィギャラブルシステム研究会, 電子情報通信学会技術研究報告, Vol. 124, No. 188, pp. 41 - 46, 2024 年 9 月
13. 小林 諒平，GPU・FPGA 連携による高性能計算，DA シンポジウム 2024 論文集, Vol. 2024, pp. 293-293, Aug. 2024.
14. 藤田 典久，Beau Johnston，小林 諒平，Mohammad Alaul Haque Monil，Narasinga Rao Miniskar，Keita Teranishi，Seyong Lee，Jeffrey S. Vetter，朴 泰祐，多様な環境におけるマルチ・タスク・ミニベンチマークの評価と Performance Portability，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. 2024-HPC-195, No. 3, pp. 1-10, 2024 年 8 月

(2) 国際会議発表

A) 基調講演

1. Osamu Tatebe, Recent Trends in Ad-hoc HPC File Systems and Caching File Systems, 4th Workshop on Re-envisioning Extreme-Scale I/O for Emerging Hybrid HPC Workloads (REX-IO), Kobe, Japan, September 24, 2024

B) 招待講演

1. Taisuke Boku, "Kernel or Mini-app, which should we take?", The Future of Benchmakrs in Supercomputing Workshop (in ISC2024), Hamburg, Germany, May 2024
2. Taisuke Boku, "Japan’s First GH200-base Supercomputer Miyabi and Status Report of Japan’s Next Generation National Flagship System", China HPC and AI Forum 2024, Shenzhen, Aug. 2024
3. Taisuke Boku, "Status Report on Post-Fugaku Project ~ Feasibility Study on Next Generation National Flagship System in Japan", Int. Workshop on Clusters, Clouds, and Data for Scientific Computing (CCDSC 2024), Chemin de Chanze, France, Sepember 5, 2024

C) 一般講演

1. Taisuke Boku, Masataka Sugita, Ryohei Kobayashi, Shinnosuke Furuya, Takuya Fujie, Masahito Ohue, Yutaka Akiyama, "Improving Performance on Replica-Exchange Molecular Dynamics Simulations by Optimizing GPU Core Utilization", 53rd International Conference on Parallel Processing (ICPP), Visby, Sweden, August 15, 2024
2. Kohei Hiraga, Osamu Tatebe, "PEANUTS: A Persistent Memory-Based Network Unilateral Transfer System for Enhanced MPI-IO Data Transfer", 30th International European Conference on Parallel and Distributed Computing (Euro-Par), Madrid, Spain, August 29, 2024
3. Sohei Koyama, Kohei Hiraga, Osamu Tatebe, "FINCHFS: Design of Ad-Hoc File System for I/O Heavy HPC Workloads", IEEE International Conference on Cluster Computing (CLUSTER), Kobe, Japan, September 27, 2024
4. Hiroki Ohtsuji, Munenori Maeda, Reika Kinoshita, Masahiro Miwa, Osamu Tatebe, Scalable RPC Layer Towards Millions of IOPS per Server, 9th International Parallel Data Systems Workshop (PDSW), Work-in-progress presentation, Atlanta, GA, November 17, 2024
5. Osamu Tatebe, National HPCI Shared Storage, Documenting and Classifying Research Data Storage Infrastructures, Singapore, March 10, 2025
6. Daisuke Takahashi, “Implementation of Parallel 3-D Real FFT with 2-D Decomposition on Manycore Clusters”, The 14th AIMS Conference, Abu Dhabi, UAE, December 20, 2024.
7. Daisuke Takahashi, “Parallel Implementation of Number-Theoretic Transform on GPU Clusters”, 24th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2024), Macau, October 31, 2024.
8. Shota Kawakami, Daisuke Takahashi, “Implementation and Evaluation of Octuple-Precision Fast Fourier Transform on GPU”, 2024 IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA 2024), Kaifeng, China, October 30, 2024
9. Hiroto Tadano, “Performance evaluation of a hierarchical parallel solver with iterative refinement for saddle point problems in a parallel computing environment”, 43rd JSST Annual International Conference on Simulation Technology & 23rd Asia Simulation Conference (JSST 2024 and AsiaSim 2024), Hyogo, Japan, September 2024
10. Takumi Suzuki, Ryohei Kobayashi, Norihisa Fujita, Taisuke Boku, “Accelerating Deep Learning Inference with a Parallel FPGA System”, HEART 2025: The International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies 2025, May 2025.

11. Ryohei Kobayashi, Hiroyuki R. Takahashi, Akira Nukada, Yuta Asahina, Taisuke Boku, Ken Ohsuga, “Accelerating General Relativistic Radiation Magnetohydrodynamic Simulations with GPUs”, HPC Asia 2025: International Conference on High Performance Computing in Asia-Pacific Region, February 2025.
12. Norihisa Fujita, Beau Johnston, Narasinga Rao Miniskar, Ryohei Kobayashi, Mohammad Alaul Haque Monil, Keita Teranishi, Seyong Lee, Jeffrey S. Vetter, Taisuke Boku, “CHARM-SYCL & IRIS: A Tool Chain for Performance Portability on Extremely Heterogeneous Systems”, 20th IEEE International Conference on e-Science 2024, Osaka, Japan, September 19, 2024.
13. Takumi Suzuki, Ryohei Kobayashi, Norihisa Fujita, Taisuke Boku, “Accelerating Deep Learning Inference with Multiple FPGAs”, 2nd Workshop on FPGA Technologies for Adaptive Computing (FTAC 2024), June 4, 2024.
14. Kaito Kitazume, Norihisa Fujita, Ryohei Kobayashi, Taisuke Boku, “Preliminary Evaluation of Flow Control on the Inter-FPGA Communication Framework CIRCUS”, 2nd Workshop on FPGA Technologies for Adaptive Computing (FTAC 2024), Kyoto, June 4, 2024.

D) ポスター発表

1. Haruka Miyauchi, Sohei Koyama (advisor), Osamu Tatebe (advisor), Design of Reliable and Efficient Syscall Hooking Library for a Parallel File System, The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC), ACM Student Research Competition Undergraduate, Atlanta GA, November 19-21, 2024
2. Ryosuke Maeda, Masaki Nakano, Osamu Tatebe, Asynchronous scheduling of communication and I/O integrated with Rust, SupercomputingAsia 2025, Singapore, March 10-13, 2025
3. Toshihiro Hanawa, Kengo Nakajima, Yohei Miki, Takashi Shimokawabe, Kazuya Yamazaki, Shinji Sumimoto, Osamu Tatebe, Taisuke Boku, Daisuke Takahashi, Akira Nukada, Norihisa Fujita, Ryohei Kobayashi, Hiroto Tadano, Akira Naruse, “Preliminary Performance Evaluation of Grace-Hopper GH200”, IEEE International Conference on Cluster Computing (CLUSTER), Kobe, Japan, pp. 184-185, 10.1109/CLUSTERWorkshops61563.2024.00050, 2024
4. Wentao Liang, Norihisa Fujita, Ryohei Kobayashi, Taisuke Boku, “Using SYCLomatic to Migrate CUDA Code to oneAPI Adapting NVIDIA GPU”, IEEE International Conference on Cluster Computing (CLUSTER), Kobe, Japan, pp. 192-193, 10.1109/CLUSTERWorkshops61563.2024.00054, Japan, September 2024
5. Kaito Kitazume, Norihisa Fujita, Ryohei Kobayashi, Taisuke Boku, “Preliminary Evaluation of Kyokko for Inter-FPGA Communication Framework CIRCUS”, IEEE International

Conference on Cluster Computing (CLUSTER), Kobe, Japan, pp. 194-195, 10.1109/CLUSTERWorkshops61563.2024.00055, September 2024

6. Norihisa Fujita, Beau Johnston, Ryohei Kobayashi, Keita Teranishi, Seyong Lee, Taisuke Boku, Jeffrey S. Vetter, “Unified Programming Environment for Multiple Accelerator Types with Programming, Performance and Compiler Portability”, ISC High Performance 2024, Hamburg, Germany, May 2024

(3) 国内学会・研究会発表

A) 招待講演

1. 朴 泰祐，「ポスト富岳」FS の状況とアプリケーション開発の準備について，第 8 回 HPC ものづくり統合ワークショップ，東京，2024 年 12 月
2. 朴 泰祐，「ポスト富岳」に向けたアプリケーション開発〜何を準備すべきか〜，富岳成果創出加速課題（材料物理化学・燃料電池）シンポジウム2025，東京，2025 年 3 月
3. 小林 諒平，GPU・FPGA 連携による高性能計算，DA シンポジウム 2024−システムと LSI の設計技術−，2024 年 8 月

B) その他の発表

1. Mingzhe Yu, Osamu Tatebe, Distributed K-FAC Over Unstable Networks (unreferred), 第 195 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発表会，徳島，2024 年 8 月 8 日
2. Mingzhe Yu, Osamu Tatebe, Asynchronous Decentralized Distributed K-FAC: Enhancing Training Efficiency and Load Balancing in Heterogeneous Environments (unreferred), 第 197 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発表会，沖縄，2024 年 12 月 16 日
3. 前田 椋祐，中野 将生，建部 修見，分散ファイルシステムにおける通信イベントと I/O イベントの非同期スケジューリングを統合した非同期 I/O の性能評価，第 197 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発表会，沖縄，2024 年 12 月 17 日
4. 高橋 大介，数学定数に対する 2 進 BBP 型公式の計算における除算について，日本応用数理学会 2024 年度年会，京都，2024 年 9 月 16 日
5. 高橋 大介，GPU クラスタにおける並列数論変換の実現と評価，日本応用数理学会 2024 年度年会，京都，2024 年 9 月 14 日

6. 長橋 朋也，高橋 大介，MPI/OpenMP 並列化によるスライドパズルの Zero-Aware Pattern Database の構築，第 195 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発表会（SWoPP 2024），徳島，2024 年 8 月 9 日
7. 川上 昌汰，高橋 大介，GPU における 8 倍精度高速フーリエ変換の実装と評価，第 194 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発表会，横浜，2024 年 5 月 8 日
8. 多田野 寛人，並列計算環境における鞍点型連立一次方程式に対する反復改良付き階層並列型解法の性能評価，日本応用数理学会 2024 年度年会，京都，2024 年 9 月.
9. 多田野 寛人，反復改良法による鞍点型連立一次方程式に対する階層並列型解法の近似解精度改善，日本応用数理学会第 21 回研究部会連合発表会，岡山，2025 年 3 月
10. 小林 諒平，高橋 博之，額田 彰，朝比奈 雄太，朴 泰祐，大須賀 健，“GPU 演算加速による一般相対論的輻射磁気流体シミュレーションコードの性能評価”，第 198 回ハイパフォーマンスコンピューティング・第 14 回量子ソフトウェア合同研究発表会，2025 年 3 月
11. 北爪 開人，藤田 典久，小林 諒平，朴 泰祐，“並列 FPGA 間通信フレームワーク CIRCUS へのフロー制御の実装と評価”，第 198 回ハイパフォーマンスコンピューティング・第 14 回量子ソフトウェア合同研究発表会，2025 年 3 月
12. 北爪 開人，上野 知洋，吉井 一友，木山 真人，藤田 典久，小林 諒平，佐野 健太郎, 朴 泰祐, “適応型帯域圧縮ハードウェアプラットフォームの Chisel 実装と評価”，リコンフィギャラブルシステム研究会，新潟，2024 年 9 月.
13. 藤田 典久，Beau Johnston，小林 諒平，Mohammad Alaul Haque Monil，Narasinga Rao Miniskar，Keita Teranishi，Seyong Lee，Jeffrey S. Vetter，朴 泰祐，“多様な環境におけるマルチ・タスク・ミニベンチマークの評価と Performance Portability”，第 195 回ハイパフォーマンスコンピューティング研究発表会，徳島，2024 年 8 月.
14. 塙 敏博，建部 修見，中島 研吾，朴 泰祐，三木 洋平，下川辺 隆史，山崎 一哉，住元 真司，高橋 大介，額田 彰，藤田 典久，小林 諒平，多田野 寛人，田浦 健次朗，細川 颯介，髙橋 淳一郎，成瀬 彰，“GH200 の予備性能評価”，第 195 回ハイパフォーマンスコンピューティング研究発表会，徳島，2024 年 8 月

7. 異分野間連携・産学官連携・国際連携・国際活動等

異分野間連携（センター内外）

- 地球環境研究部門との共同研究における地域気象コードの GPU 化と高度化
- 宇宙物理研究部門との共同研究における GPU+FPGA 多重演算加速環境の初期天体シミュレーションコードの性能向上への適用

- 宇宙物理研究部門との共同研究におけるブラックホール降着円盤の数値シミュレーションの GPU 加速
- 素粒子物理研究部門と共同で Gfarm ファイルシステムを用いた Japan Lattice Data Grid（JLDG）の構築，運用

産学官連携

- 学際ハブ拠点形成事業「AI 時代における計算科学の社会実装を実現する学際ハブ拠点形成」における生命科学シミュレーションコード Amber の高スループット実行に関する研究（アヘッド・バイオ・コンピューティング社との共同研究）
- 特別共同研究事業により筑波大学スモールリサーチラボ（SRL）富士通データコンピューティング基盤共同研究拠点を設置し, 次世代データコンピューティング基盤のコア技術を創出

国際連携・国際活動

- 米国オークリッジ国立研究所とのヘテロ環境におけるプログラミングに関する共同研究
- 米国エネルギー省と文部科学省が締結しているシステムソフトウェアに関する共同研究契約により，米国アルゴンヌ国立研究所とのストレージに関する共同研究

8. シンポジウム，研究会，スクール等の開催実績

1. Gfarm シンポジウム 2024，東京（ハイブリッド），2024 年 9 月 6 日
2. Gfarm ワークショップ 2024，沖縄，2024 年 12 月 18 日

9. 管理・運営

1. 朴泰祐：情報環境委員会委員
2. 朴泰祐：計算科学研究センターセンター長
3. 朴泰祐：計算科学研究センター次世代計算機システム研究開発室室長
4. 朴泰祐：計算科学研究センターHPCI 推進室室員
5. 朴泰祐：計算科学研究センター情報セキュリティ委員会委員長
6. 建部修見：情報環境企画室室員
7. 建部修見：ネットワーク管理委員会委員
8. 建部修見：情報セキュリティ技術小委員会委員
9. 建部修見：基幹ネットワーク小委員会委員
10. 建部修見：計算科学研究センター副センター長

11. 建部修見：計算科学研究センター高性能計算システム研究部門主任
12. 建部修見：計算科学研究センター高性能計算システム運用開発室室長
13. 建部修見：計算科学研究センター情報セキュリティ委員会委員
14. 建部修見：計算科学研究センター研究倫理委員会副委員長
15. 建部修見：計算科学研究センター情報環境委員会部局技術責任者
16. 建部修見：計算科学研究センター計算機システム運用委員会委員長
17. 建部修見：計算科学研究センター次世代計算機システム研究開発室員
18. 建部修見：計算科学研究センタービッグデータ・AI 連携推進室室員
19. 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
20. 高橋大介：計算科学研究センター計算科学振興室室長
21. 高橋大介：計算科学研究センター先端計算科学推進室室員
22. 高橋大介：計算科学研究センター情報セキュリティ委員会委員
23. 高橋大介：計算科学研究センター次世代計算機システム研究開発室室員
24. 高橋大介：計算科学研究センター学際計算科学連携室室員
25. 高橋大介：計算科学研究センター情報セキュリティ委員会委員
26. 額田彰：計算科学研究センター先端計算科学推進室室員
27. 額田彰：計算科学研究センター高性能計算システム運用開発室室員
28. 多田野寛人：計算科学研究センター計算科学振興室室員
29. 多田野寛人：計算科学研究センター計算機運用委員会委員
30. 多田野寛人：計算科学研究センター情報セキュリティ委員会委員
31. 小林諒平：計算科学研究センター計算機システム運用委員会委員
32. 小林諒平：計算科学研究センター先端計算科学推進室室員
33. 小林諒平：計算科学研究センター次世代計算機システム研究開発室室員
34. 小林諒平：計算科学研究センター高性能計算システム運用開発室室員
35. 藤田典久：計算科学研究センター高性能計算システム運用開発室室員

10. 社会貢献・国際貢献

1. 朴泰祐：理化学研究所客員主管研究員
2. 朴泰祐：PC クラスタコンソーシアム理事
3. 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
4. 朴泰祐：「富岳」成果創出加速課題領域総括
5. 朴泰祐：「次世代計算基盤に関する調査研究」Program Director
6. 朴泰祐：HPCI 計画推進委員会委員
7. 朴泰祐：「富岳」課題推進委員会委員

8. Taisuke Boku: Steering Committee Chair, International Conference on High Performance Computing in Asia-Pacific Region (HPCAsia)
9. Taisuke Boku: Steering Committee Member, IEEE Cluster
10. Taisuke Boku: Steering Committee Member, International Conference on Parallel Processing
11. Taisuke Boku: 2024 ACM Gordon Bell Prize Committee Member
12. 建部修見：HPCI セキュリティインシデント即応委員会委員
13. 建部修見：HPCI 連携サービス委員会委員
14. 建部修見：HPCI 連携サービス運営・作業部会委員
15. 建部修見：HPCI 利用研究課題審査委員会レビューアー
16. 建部修見：理化学研究所客員研究員
17. 建部修見：情報通信研究機構協力研究員
18. 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
19. 建部修見：東京科学大学情報基盤センターTSUBAME 共同利用作業部会委員
20. 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
21. Osamu Tatebe: Poster Committee, IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC24)
22. Osamu Tatebe: Program Committee, 38th IEEE International Parallel & Distributed Processing Symposium (IPDPS)
23. Osamu Tatebe: Program Committee, 30th International European Conference on Parallel and Distributed Computing (Euro-Par 2024)
24. Osamu Tatebe: Program Committee, IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid 2024)
25. Osamu Tatebe: Program Committee, IEEE International Conference on Cluster Computing (CLUSTER 2024)
26. Osamu Tatebe: Program Committee, 53rd International Conference on Parallel Processing (ICPP 2024)
27. Osamu Tatebe: Program Committee, ACM International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2024)
28. Osamu Tatebe: Program Committee, 9th International Parallel Data Systems Workshop (PDSW 2024)
29. Osamu Tatebe: Program Committee, 5th Workshop on Extreme-Scale Storage and Analysis (ESSA 2024)
30. Osamu Tatebe: Program Committee, 4th Workshop on Re-envisioning Extreme-Scale I/O for Emerging Hybrid HPC Workloads (REX-IO '24)

31. Osamu Tatebe: Program Committee, 14th International Workshop on Advances in High-Performance Computational Earth Sciences: Numerical Methods, Frameworks & Applications (IHPCES 2024)
32. Daisuke Takahashi: The International Journal of High Performance Computing Applications Editor
33. Daisuke Takahashi: The 19th International Workshop on Automatic Performance Tuning (iWAPT 2024) Program Committee Member
34. Daisuke Takahashi: The 38th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2024) Program Committee Member
35. Daisuke Takahashi: The 24th International Conference on Computational Science and Its Applications (ICCSA 2024) Publicity Committee Member
36. Daisuke Takahashi: The International Conference on Computational Science (ICCS 2024) Program Committee Member
37. 高橋大介：理化学研究所客員主管研究員
38. 高橋大介：HPCI 利用研究課題審査委員会レビューアー
39. 高橋大介：HPCI 連携サービス運営・作業部会委員
40. 高橋大介：情報処理学会論文誌査読委員
41. 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運営委員
42. 多田野寛人：日本シミュレーション学会理事
43. 多田野寛人:日本シミュレーション学会「非線形現象の高性能数値解析技術研究委員会」委員
44. 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
45. 多田野寛人：日本応用数理学会「非線形問題の高性能解法と可視化技術」研究部会幹事
46. Hiroto Tadano: Committee, The 43rd JSST Annual International Conference on Simulation Technology & The 23rd Asia Simulation Conference (JSST 2024 and AsiaSim 2024)
47. 小林諒平：理化学研究所客員研究員
48. 小林諒平：電子情報通信学会コンピュータシステム研究専門委員会幹事
49. 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
50. 小林諒平：電子情報通信学会英文論文誌 D 編集委員
51. 小林諒平：電子情報通信学会論文誌 Low-Power and High-Speed Chips and Systems 特集号編集幹事
52. 小林諒平：電子情報通信学会論文誌 Forefront Computing 特集号編集幹事
53. 小林諒平：電子情報通信学会 ISS ソサイエティ誌編集幹事

54. 小林諒平：電子情報通信学会 ISS ソサイエティ誌編集委員
55. 小林諒平：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幹事
56. 小林諒平：アダプティブコンピューティング研究推進体広報イベント WG 副グループ長
57. 小林諒平：SWoPP 2024 実行委員（コンピュータシステム研究会）
58. 小林諒平：xSIG 2024 プログラム委員
59. Ryohei Kobayashi: Digital Chair, IEEE Cluster 2024
60. Ryohei Kobayashi: Web Liaison, ICS 2024
61. Ryohei Kobayashi: Program Vice Chair, COOL Chips 27
62. Ryohei Kobayashi: Program Committee, SupercomputingAsia 2025
63. Ryohei Kobayashi: Program Committee, FPL 2024
64. Ryohei Kobayashi: Program Committee, ICPP 2024
65. Ryohei Kobayashi: Program Committee, Euro-PAR 2024
66. Ryohei Kobayashi: Program Committee, FTAC2024
67. Ryohei Kobayashi: Program Committee, HEART2024
68. 藤田典久：理化学研究所客員研究員
69. 藤田典久：情報処理学会論文誌コンピューティングシステム（ACS）編集委員
70. Norihisa Fujita: Special Session Chair, Performance Optimization and Auto-Tuning of Software on Multicore/Manycore Systems 2024 (POAT2024)
71. Norihisa Fujita: Workshop Committee Member, ISC 2024

11. その他

なし