1. 調査研究の概要

1.1 調査研究の背景と目的

次世代の計算基盤には、従来のサイエンスにとどまらず、広範なSDGs・Society 5.0の実現に向けた課題解決のためのプラットフォームとしての役割が一層求められる。特に、今後の科学では、いわゆる「研究DX」により新たなイノベーションを起こすべく非常に高度なデジタルツインを中心としたものとなり、そのためには現象をモデル化することによる演繹的シミュレーションと、学習・AIに基づく現象の帰納的シミュレーション、さらには大規模なデータを収集・処理基盤と、データの両方のシミュレーションとの同化、などが密に連携する計算基盤の構築、及びその開発・運用が鍵となる。このような次世代計算基盤では、大規模高速な計算・データ処理が行えるスーパーコンピュータがその中心的な役割を担う。特に、比較的狭いシミュレーション領域を対象とした過去のものや、AIや特定領域の計算に特化したマシンと異なり、デジタルツインの基盤としてのスーパーコンピュータは、広範な計算手法・シミュレーション技法や大規模データを駆使しつつ、それらが密に連携しながら全体のワークフローの実行が可能である必要があり、単に特定のベンチやアプリで高い性能を達成するのみならず、幅広いアプリケーション分野で高効率を達成できる計算基盤の特性が必要となる。

しかしながら、その実現は非常に困難を極めると予想される。一般に、高い性能と広い適用可能性は相反する要求でもあり、現在理化学研究所計算科学研究センターで運用されているスーパーコンピュータ富岳の開発では、ハードウェアとアプリケーションのコデザインをHPCの産学オールジャパン体制で推し進めることで、それを実現可能なものとした。一方、次世代計算基盤では、ムーアの法則の終焉が近づきつつあるなか、半導体プロセス技術の進歩による演算あたりの大幅な電力改善が見込めず、高い性能をあまねく達成することはさらに難しくなると予測されている。実際、計算機科学コミュニティが調査・執筆したNGACI白書でも、2028年度に実現可能なシステムでは、許容電力の大幅増を仮定しても、富岳の3.3倍から10倍程度の性能向上にとどまると予想されている。また、近年のソフトウェア開発における複雑性の増大も、高い性能と広い適用可能性への大きな足かせになる。従来型の浮動小数点演算ピーク性能重視のシステム設計では、一部のベンチマークでは一見華々しい性能を見せつつも、実際のアプリケーションとの性能乖離が増大する一方となり、今後立ち行かなくなることは明らかであり、またソフトウェアのエコシステムの重要さやコストを軽視した計算基盤では、開発しても使うことのできないシステムになり、研究DXの基盤としては意味のないものとなる。

性能面に関しては、様々な文献や我々のベンチマークや性能モデルの多岐にわたる研究調査でも、大多数のアプリケーションはメモリやネットワークバンド幅など、基本的にデータ移動の性能によって律速される特徴を持つことがわかっている。次世代計算基盤では、FLOPSで語られるピーク性能重視から脱却し、データ移動、すなわちByteにより表現される指標をアーキテクチャとアルゴリズムの点から最適化・高効率化していくことが重要となる。本調査研究では、上記理念の達成に向け、アーキテクチャ設計の基本理念として演算精度も考慮しながら必要な計算性能は確保しつつ、電力制約の下でデータ移動を高度化・効率化する"FLOPS to Byte"指向のシステム構築を、アーキテクチャ開発からアルゴリズム設計、アプリケーション技術に至るまで実践し、実効的な性能を向上させるための次世代計算基盤を調査研究することが目的である。

アーキテクチャの調査研究では、電力制約の下でデータ移動を高度化・効率するデバイス技術やアーキテクチャ技術を中心に調査研究を実施する。近年までの大規模スーパーコンピュータでは、レイテンシよりも演算とデータ転送スループットを重視し、かつ並列度を上げた際の実効効率低下を緩和させるために、弱スケーリング思想に基づく設計が重視されてきた。一方、機械学習における推論処理や創薬系のアプリケーションでは同一サイズの問題を高速に解く強スケーリングが求められ、半導体技術によるDRAM容量の増加傾向も鈍化しつつある。これら背景を踏まえ、アーキテクチャ検討項目として、特に3次元積層メモリ技術の活用、データフロー的観点も踏まえた強スケーリング向けの要素技術、チップ間直接光通信技術を特に意識しながら、関連するアーキテクチャ技術を国内外複数ベンダと共に調査研究を実施する。令和6年度は、特に次世代計算基盤に向けたアーキテクチャの絞り込みを行い、その技術的実現可能性のさらなる調査や実現に要するコストに関して調査研究を行った。

システムソフトウェア・ライブラリの調査研究に関しては、これまでのフラグシップや第二階層システムのソフトウェアの開発工数や利用状況、そして既に構築されているエコシステムとの親和性も踏まえ、ソフトウェア資産として何をベースに、国内でどこまでを開発すべきか、もしくは国際協調が必要なソフトウェアに優先度をつけて明らかにしつつ、今後のロードマップを策定することが必要である。また、次世代計算基盤を産業界も含む幅広い応用分野での活用を促すためにも、従来のシステムソフトウェアだけでなくデータ利活用の促進、機械学習技術と第一原理シミュレーション、さらには大規模リアルタイムデータ処理の高度な融合、従来の共用HPCシステムとは次元の異なる高セキュリティの担保、などを主要検討項目とし、次世代計算基盤として行うべきソフトウェア開発について調査研究を行う。令和6年度は、これまでの調査研究や検討に基づきつつ、ソフトウェア開発の戦略をまとめたソフトウェアマトリックスを構築するとともに、アーキテクチャ調査研究グループにより絞り込まれた次世代計算基盤のアーキテクチャ用のシステムソフトウェアの開発戦略を検討した。

アプリケーションの調査研究に関しては、従来のサイエンスをさらに進化・深化させるのみにとどまらず、社会科学やSociety 5.0といった新しい応用分野への展開も見据え、アーキテクチャとアルゴリズムとアプリケーションの三者が連携するコデザインに基づく次世代計算基盤構築に向けた調査研究を実施する。特に、複数アーキテクチャを統一的に評価するための広範なベンチマークセットを構築し、それを利用したアーキテクチャ評価結果を踏まえてアルゴリズムやパラメータの改善を検討する。また、それをベンチマークセットへと更新し、性能モデルを構築した上で、探索的な評価を行う。このサイクルを回した新たなコデザインにより、各アプリケーションで高い実効性能を得るためのアーキテクチャとアルゴリズムを探求する。これにより、今後アーキテクチャの進化に伴って、どのようなアルゴリズムのクラスが大幅な進化が見込まれるか、という指標も抽出し、今後の発展につなげる。令和6年度は、絞り込まれたアーキテクチャでの基本設計やコデザイン評価が行えるように、当該アーキテクチャへ対応できるようこれまで開発されたベンチマークプログラムを更新した。また、当該アーキテクチャを前提とした際の各アプリ分野の対応状況等のヒアリングや、ターゲットサイエンスのアップデートを行った。

フラッグシップシステムの開発にあたっては、日本の半導体戦略とも整合していくことが重要であり、特に今後の日本が持つべき半導体技術の強みを活用しつつさらにそれを伸ばすための戦略や、開発されたシステムの幅広い産業展開を目的とした開発指針の策定も実施した。これらを総合し、ポスト「富岳」時代の次世代計算基盤の具体的な性能や機能、開発戦略について検討を行うことも本調査研究の目的として活動を行った。

1.2 調査研究の体制

1.2.1 調査研究の体制

上記の調査研究を実施するために、国内外の主用なHPC関連のベンダと、国内の各基盤センターのシステムで構成するナショナル・インフラストラクチャー・システム(NIS)、また様々なアプリケーション分野をリードする国立研究所の研究者と共に、図 1.2.1.1に示す体制で調査研究を実施した。

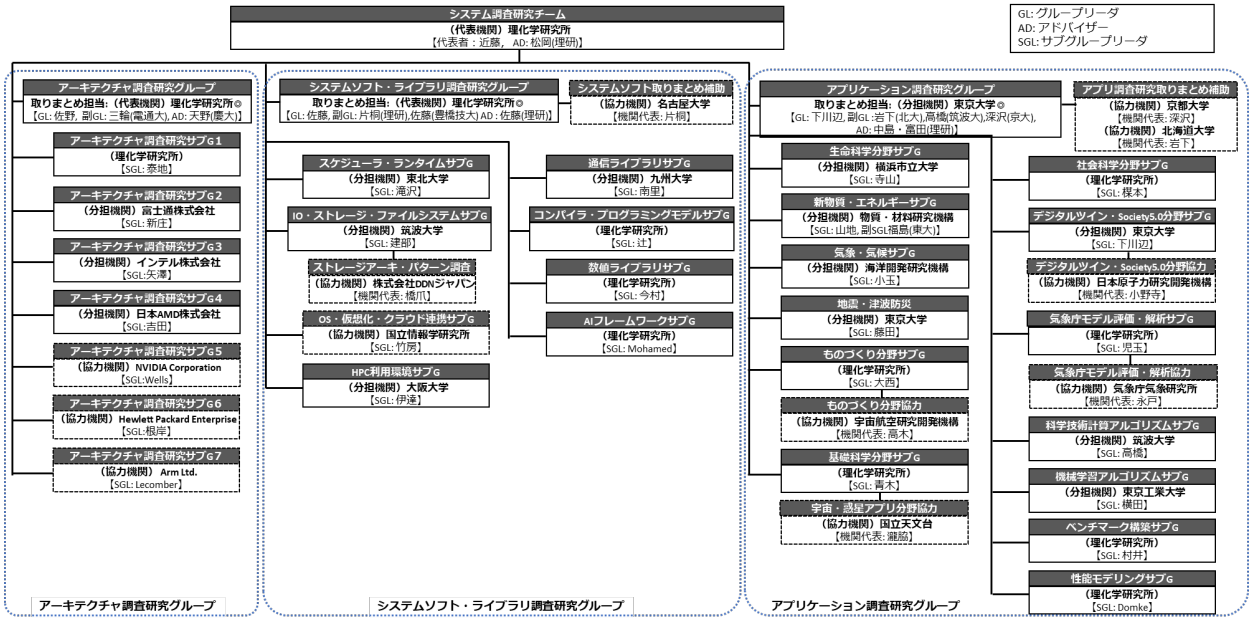


図 1.2.1.1 調査研究体制

2. アーキテクチャ研究グループ

2.1 調査研究の概要および方針

2.1.1 目的と方法

アーキテクチャ研究グループでは、科学技術研究分野、産業分野、および社会からの利用ニーズに対して有望なシステムの選択肢の探索およびその実現可能性の提示を目的とし、ムーア則の終焉や半導体技術およびパッケージング技術等の発展を見据えつつ、システム全体やその構成要素について考え得る技術的可能性に関する調査検討を行う。そのために、例えば、1. 汎用プロセッサ（CPU）、2. DRAM、3. SRAM（キャッシュメモリ等）やその3次元積層技術、4. アクセラレータ、5. 相互接続網、6. ストレージ、7. CMOS技術やパッケージング技術（3次元積層、チップレット等）、8. シリコンフォトニクス、9. 信頼性・可用性・保守性、などについて、技術動向と共に次世代計算基盤システム開発に適した将来技術を調査・探索し、その有効性および実現可能性を評価する。

ハードウェアシステムの開発および製造・設置に関しても具体的な検討を行うために、アカデミアに加え、関連分野において長年に渡り技術開発とシステム販売に携わってきたベンダを分担機関あるいは協力機関とし、上記の調査検討を実施する。特に、昨今の半導体業界やITC業界では国境を跨ぐ世界的なサプライチェーンや技術連携が不可欠であり、次世代計算基盤開発においても日本単独で全ての技術を網羅することは難しいと予想されることから、ベンダとして国内外の企業に参画頂き、特に日本が独自に保有し発展させるべき技術と国際的に連携すべき技術の候補を明らかにしつつ、開発すべきシステムとそのアーキテクチャの選択肢を提示することを目指す。半導体技術等の原理的限界と、技術的に突破できる見込みのあるそれ以外の課題を精査し、次世代計算基盤に求められる要件に対して優先順位を設定した上で解決策の実現可能性を検討する。

本年度では、グループとりまとめA0とサブグループA1～7の体制で、上記の調査研究を実施した。

A0. アーキテクチャグループとりまとめ	担当機関：理化学研究所（代表機関）
A1. アーキテクチャ調査研究サブグループ1	担当機関：理化学研究所（代表機関）
A2. アーキテクチャ調査研究サブグループ2	担当機関：富士通株式会社（分担機関）
A3. アーキテクチャ調査研究サブグループ3	担当機関：インテル株式会社（分担機関）
A4. アーキテクチャ調査研究サブグループ4	担当機関：日本ＡＭＤ株式会社（分担機関）
A5. アーキテクチャ調査研究サブグループ5	担当機関：NVIDIA Corporation（協力機関）
A6. アーキテクチャ調査研究サブグループ6	担当機関：Hewlett Packard Enterprise（協力機関）
A7. アーキテクチャ調査研究サブグループ7	担当機関：Arm Ltd.（協力機関）

「A0. アーキテクチャグループとりまとめ」では、以下の項目について調査研究を実施した。詳細は、それぞれの節に記載する。

- 昨年度に続く、半導体技術・パッケージング技術・アーキテクチャの技術動向調査（2.1.2 節）
- 目指すべきアーキテクチャの基本構成とシステム性能の検討（2.1.3 節）
- LLM の学習等に関する大容量メモリノードの評価（2.1.4 節）
- GPU ノードの性能評価（2.1.5 節）

アーキテクチャ調査研究サブグループ1～7では、マイクロアーキテクチャ、ノードアーキテクチャ、システムアーキテクチャ、関連するシステムソフトウェアやベンチマークによるアプリケーションの性能推定、および要素技術の調査等の個別に設定する対象について、調査検討を行った。詳細は、2.2～2.8節において述べる。

2.1.2　技術動向調査

2.1.2.1　技術動向調査の概要

スーパーコンピュータ、特にフラッグシップ機は、高い演算処理性能と電力あたり性能を実現するために最先端のハードウェア技術を駆使してシステム構築が行われる。そのため、次世代計算基盤のアーキテクチャ検討においては、システムを構成する様々なハードウェアの技術動向を十分理解した上で、2028年頃までに実用化可能な技術を見極める必要がある。本研究グループでは、種々のハードウェア技術動向について昨年度に続き調査を行った。

2.1.2.2　半導体・パッケージング・シリコンフォトニクス

昨年度の報告書には、以下の調査結果が記されている。半導体の高密度実装構造について、本年度は1nmノード以降の微細プロセスのトレンドとして、トランジスタ構造の3次元化が見られた。具体的にはIntelやTSMCがnMOSFETとpMOSFETを上下方向に配置し、最大50%の面積削減を実現するComplementary Field-Effect Transistor (CFET)技術を公表した。TSMCは48nmのゲートピッチ、Intelは60nmのゲートピッチの実現に成功している。IRDS 2022によると、トランジスタをFinFET (3nmノード、2022年)からCFET (1.5nm、2031年)にすることで、デジタル回路ブロックの面積がおよそ55%に、さらに3次元集積を施すことで2037年には0.5nmノードにて8%の面積になることが予想された。

Hybrid Bondingによる3次元チップ積層方式として、AMDはGPUアクセラレータMI300を発表し、2.5DのHBM実装と比べておよそ2.5倍の電力効率を実現している。IBMによるAIアクセラレータNorthPoleはNVIDIA社のL4と比べて約2.5倍のエネルギー効率改善を実現している。また、異なる高密度実装手法のテクノロジトレンドとして、チップ背面から電源を供給するPowerViaをIntelが提案し、90%を超えるセルユーティライゼーション、6%の動作性能改善を実現している。2024年よりIntel A20世代から実装予定である。SK HynixがDRAMのHybrid bondingに成功し、従来のマイクロバンプ方式のHBMとの性能差別化の可能性を示した。

CPOについても研究開発が進んでおり、TSMCはチップレットベースのCPOの性能を見積もっている。従来のIIIV族プラガブルベースの光モジュール（1.6Tbps）からシリコンフォトニクスベースのCPOに変更することで、約5倍の6.4Tbpsのバンド幅改善を見積もった。

昨年度に続く調査の結果、以上の技術動向には以下の更新が見られた。国際会議ISSCC2025では、Intelにより同社の半導体製造技術のロードマップとして、intel 16、intel UMC 12、intel 3（3-E, 3-T, 3-PT）、intel 18A、intel 14Aが紹介された。このうちintel 18Aは、2011年のFinFET導入以来最大の革新技術であり、インテル・ファウンドリーによる Gate-All-Around (GAA) トランジスタの実装であるRibbonFETにより、FinFETに比べて密度と性能が向上している。最適化されたリボンスタックにより、ワット当たりの優れた性能および最小電源電圧が可能となる。

また、サムスンからは、次世代パッケージング技術のロードマップが紹介された。シリコンインターポーザーに基づくI-Cube Sに代わり、2025年頃からはPanel-Level RDLベースのインターポーザーベースが利用可能となり、従来の100x100mm^2から130x130mm^2程度のパッケージサイズに拡大することにより、例えば16個のHBMを搭載可能となる。

2.1.2.3　汎用プロセッサ・メモリ

- **汎用プロセッサ**：以下では汎用プロセッサの技術動向を述べる。

  昨年度の報告書には、以下の調査結果が記されている。汎用プロセッサに関しては、本年度に入り、IntelとAMDはそれぞれ第5世代Xeonスケーラブルプロセッサと EPYC 97x4プロセッサの販売を開始した。その結果、汎用プロセッサのピーク演算性能は9.22TFLOPS（AMD EPYC9754における倍精度浮動小数点演算性能）、電力性能は25.6GFLOPS/Wにまで向上したものの、性能面でアクセラレータが優位な状況に変わりは

ない。例えばAMD MI300xのピーク演算性能は倍精度で163.4TFLOPS（EPYC 9754の18倍）、電力性能は218GFLOPS/W（EPYC 9754の8.5倍）である。また行列演算ユニットに関しては、第5世代Xeonスケーラブルプロセッサでは16x16x32の行列積を16サイクルで実行するようになった[2]。Intelの行列演算ユニットが現在サポートするデータ型はBF16とINT8のみであるが、Granite Rapidsでは半精度複素数型のサポートも予定されている[3]。

昨年度に続く調査の結果、以上の技術動向には以下の更新が見られた。AMDにより、TSMC 4nmテクノロジノードにより製造されるZen5 CPUがアナウンスされた。Zen5 ファミリには、第5世代のRyzen Desktop Processor、第3世代のMobile 向けRyzen AI Processor、および第5世代のEPYC Server Processorが含まれる。これらのCPUでは、3次元積層したSRAMによる3D V-Cacheや5GHzを超える動作周波数（最大で5.7 GHz）、およびFP256またはFP512のAVXデータパスがサポートされる。Zen4ではCPU dieの上に3D V-Cacheが積層されているが、Zen5では逆に3D V-Cacheの上にCPU dieが積層される。Zen5c 世代のEPYC9965 Processorは、Zen4, Zen4c世代と比べてそれぞれ2.6倍、2.0倍のHPC性能を達成する。

また、IBMからは、次世代の5.5GHz MicroprocessorであるTelum IIがアナウンスされた。Telum IIは、IBM ZメインフレームおよびLinuxOne'dシステム用であり、Samsung 5nmテクノロジノードで製造される。Processorと共に、gzip用のデータ圧縮アクセラレータとINT8での計算が可能なAIアクセラレータを搭載する。前世代のTelumと比べて、WattあたりのIPCは1.3倍の改善がなされた。

Intelからは、Intel Xeon 6 SoCであるGranite Rapids-Dがアナウンスされた。Granite Rapidsは、vRAN (Radio Access Network)およびEdgeを対象として設計されている。モジュール化された設計が採り入れられており、P-CoreであるCompute Dieと、各種アクセラレータとPCIeやEthernetが統合されたIO Dieが、EMIB（Embedded Muti-Die Interconnect Bridge）により2.5D接続される。

- **メモリ**：ここでは、主にオンチップおよびオフチップのメモリ技術の動向調査について述べる。

昨年度の報告書には、以下の調査結果が記されている。メモリ各社からHBM3eの量産が発表された。データ転送速度は9.2-9.8Gb/s、デバイス当たりで1.2-1.28TB/sに拡張されている。2024年第2四半期では24GB版の出荷が予定されており、36GB版の開発・評価も進んでいる。HBM4についても規格が検討されており、2025年頃に規格が出てくる見込みである。一方、DRAMの３D実装については、研究は各所で行われているが、実用化に向けた期待が高まっている。例えば、2023年のISSCCのPlenaryで発表したAMDのLisa Suの講演[4]では、すでに出荷が行われているSRAMを3D実装した3D V-Cacheの他に、DRAMのスタッキングを将来の技術と紹介しており、実用化に向けた開発については、今後も注目する必要がある。

昨年度に続く調査の結果、以上の技術動向には以下の更新が見られた。ISSCC2025にて、SamsungよりHBM4、HBM4eの規格案が発表された。HBM4、HBM4eの容量はそれぞれ36~48GB, 48~64GBで12、16のCore stack数によりそれらを実現する。IO幅はHBM3eの1024bitの倍の2024bitとなり、最大のPin speedはHBM4、HBM4eのそれぞれで8GBps以上 (実際にはHBM3eの9.2Gbpsと同程度と予想される)および12Gbps以上となる。これらの数値から、HBM4、4eのデータ転送速度はHBM3eの1.2TB/sと比べ、2倍以上の2.4TB/sに達すると考えられる。なお、HBM4、HBM4eでは、HBM3eまでのDRAM processによるbase dieではなく、Logic processによるbase dieが使用される。

また、ISSCC2025では、Samsungからは、カスタムHBM、CXL Memory、LPDDR-Wide IO (LPW)、LPDDR5X-PIM、次世代HBMの発表があった。カスタムHBMは、base dieにカスタムlogic、CHBM controllerおよびLPDDR controllerを実装することができ、xPU dieとの接続に通常のHBMよりも少ないピン数のDie-to-Die (D2D)を使用出来ることに加え、HBM ControllerをHBM base die側に移動したことに

よりxPU dieにより大きなxPU回路を実装可能となる。CXL MemoryはCXL (Compute Memory Link)によりメモリ拡張可能とする方式であり、CXL Memory Module (CMM)としてDRAMベースのCMM-D, Tiered memoryベースのCMM-H等が発表された。

LPWは多数の低速IOを有するLPDDRであり、LPDDR5と比べて66%増となる200GB/s超の帯域と、54%減となる1.9pJ/bのエネルギー消費を実現する。LPDDR5X-PIM（Processing in Memory）はDRAMの周辺回路部分に演算処理回路を実装したメモリであり、同時に多数のメモリバンクを稼働させながらメモリ内またはメモリ近傍で並列に演算処理を行うことができる。また、次世代HBMとして、アクセラレータなどを実装したBase dieの上にTSVを用いてHBMのCore dieを積層する3次元スタックメモリ方式が紹介された。これにより、例えばGPUとHBMの間の伝送距離やエネルギー消費を抑えることが可能となり、またデータ転送速度は従来の2.5D接続のHBMと比べ倍近くに向上する。

2.1.2.4 アクセラレータ・ノードアーキテクチャ・CPU 接続

昨年度の報告書には、以下の調査結果が記されている。2023年11月13日に最先端共同HPC基盤施設（JCAHPC: Joint Center for Advanced High Performance Computing）は、NVIDIA GH200 Grace Hopper Superchipを搭載するスーパーコンピュータシステム OFP-II（仮称）の仕様を公開した。GH200はArm Neoverse V2のIPライセンスを利用した72コアのArm CPU(Grace)と、NVIDIAのNVIDIA H100 GPU(Hopper)を1モジュール上に統合したコンピューティングモジュールであり、CPU用に120GB (OFP-IIは120GB版を採用している) のLPDDR5Xメモリ、GPU用に96GBのHBM3メモリをそれぞれ搭載している。そして最大の特徴は、CPU・GPU間のインターコネクトにNVIDIA NVLink-C2Cを採用しており、450GB/sの帯域を確保しながらCPUとGPUのそれぞれのメモリにあるデータにそれぞれの計算デバイスが透過的にアクセスできる仕組みとなっている点である。OFP-IIの運用は2025年1月を予定しており、現在は導入作業を実施している。

昨年度に続く調査の結果、上記の技術動向に関して本年度は以下の更新が見られた。HotChips2024において、NVIDIAはBlackwell GPUの技術を発表した。その技術的特徴は以下の通りである。

・レティクル制約下における最大のdie 2つをパッケージに統合
・208B transistors in TSMC 4NP
・20 PFLOPS FP4 AI
・8 TB/s Memory Bandwidth | 8-site HBM3e
・1.8 TB/s Bidirectional NVLink bandwidth
・High-speed NVLink-C2C Link to Grace CPU

また、第４世代のNVLink Switch Chipが報告された。これは、72 GPUの全結合を可能とする。

加えて、AMDはMI300X GPUの技術を発表した。そのCDNA3コアの技術的特徴は以下の通りである。

・Doubled low precision matrix ops/clk/cu
・2:4 structured sparsity support for INT8, FP8, FP16, BF16
・Additional 2x performance with sparsity enabled
・TF32 and FP8 numerical format support
・2-bit mantissa and a 5-bit exponent for training (E5M2)
・3-bit mantissa with a 4-bit exponent for inference (E4M3)
・OCP FP8 compliant
・Co-issue FP16 | FP32 | INT32 with FP16 | FP32 | FP64

また、INTELは、Gaudi 3 AI Acceleratorの技術を発表した。その技術的特徴および仕様は以下の通りである。

・2 compute dies connected over an interposer bridge
・8 HBM2e stacks
・Up to 900W with air cooling
・Up to 1200W with liquid cooling
・PCIe Gen5 x16・
・24x 200GbE RoCE via 48 112G PAM4 Serdes
・TDP (OAM) : 900W (Air) / 1200W (Liquid)
・Peak Compute (BF16) : 1835 TFLOPS
・HBM Capacity : 128 GB
・Peak HBM BW : 3.67 TB/s
・Peak PCIe BW (bi-directional) : 128 GB/s
・Embedded NIC BW (bi-directional) : 9.6 Tb/s

2.1.2.5 ネットワーク

昨年度は、UEC (Ultra Ethernet Consortium) に参加し、次世代のHPCおよびAIが要求する広帯域通信の実現を目的とするイーサネットのロードマップ調査を行った。また、長時間スケールの並列離散イベントシミュレーション(PDES) とサロゲートモデルによる学習とを組み合わせた、ハードウェア協調設計シミュレーションを高速化するプロジェクトにも参加した。

昨年度に続く調査の結果、本年度は以下の更新が見られた。ISSCC2025にて、BROADCOMは、次世代のEthernetスイッチ製品であるTomahawk 5を発表した。その技術的特徴は以下の通りである。

・Monolithic dieを採用
・TSMC 5nm process nodeで製造
・750+mm^2 areaのdie面積
・600億個のトランジスタ
・1.325GHzのコアクロック周波数
・DSP-based 100G SerDes

Tomahawk 1～5の5世代の製品は、それぞれ28nm, 16nm, 16nm, 7nm, 5nmのプロセス技術で製造されており、各世代間で約30%の消費電力削減が達成されている。このうち、15～20%がプロセス技術の改善によるものである。それぞれの世代の製品は、3.2, 6.4, 12.8, 25.6, 51.2 Tbpsの帯域, 35.9, 28.1, 17.2, 12.0, 8.8 Pj/bitのビットあたりのエネルギー消費を達成している。また、Tomahawk 5スイッチ製品では、TH5-BaillyというDirect Drive Co-Packaged Optics (CPO)を採用している。これにより、laneにおけるビットあたりエネルギー消費を削減している。

2.1.3 目指すべきアーキテクチャの基本構成とシステム性能の検討

2.1.3.1 昨年度に得られた複数アーキテクチャ候補の絞り込み

昨年度においては、各ベンダの提案アーキテクチャ候補について、以下を実施した。

- アーキテクチャ候補の詳細化と、キーテクノロジの整理
  （マイクロアーキテクチャ、ノードアーキテクチャ、システムアーキテクチャ候補とその設計空間等の詳細化）
- アーキテクチャ候補に対する性能やコストの見積り
- アーキテクチャ候補に対するベンチマークの性能推定
- アーキテクチャ評価および比較による、有望な候補の絞り込み

今年度においては、昨年設定した評価項目に更新を行った以下の項目に対して、A2～5の各ベンダサブグループから得られた有望なアーキテクチャ候補を比較検討し、さらなる絞込みを行った。

- ノードアーキテクチャ
  - マイクロアーキテクチャやノード構成
  - ノード内ネットワーク
  - 倍精度理論ベクトル演算性能 [FP64 TF]
  - その他精度の理論演算性能（半精度等の理論行列演算性能）[FP16/BF16 TF, FP8 TF]
  - メモリ帯域 [TB/s]
  - メモリ容量 [TiB]
  - Byte per FLOP
  - 最大消費電力（TDP） [W]
  - 電力あたり性能 [GF/W]
  - コスト
- システムアーキテクチャ
  - ネットワーク
  - 電力制約下でのシステム規模
  - 電力制約下でのシステム理論性能
  - 理研の選定したベンチマークによる性能推定
  - 最大消費電力
  - コスト
- その他
  - 開発ロードマップ
  - リスクの高い技術要素

2.1.3.2 目指すべきアーキテクチャの指針

複数のアーキテクチャ候補を比較検討した結果、実現可能であると考えられるシステム性能に関して以下の指針が得られた。

・既存HPCアプリケーションで現行の「富岳」の5～10倍以上の実効計算性能
・AI処理でZetta FLOPSスケールのピーク性能を念頭に50EFLOPS以上の実効性能
・シミュレーションとAIの融合により、総合的に数十倍のアプリケーション高速化
・「富岳」と同程度か増加するとしても大きく異ならない程度の消費電力

電力制約下で上記の目標性能を達成しつつもアプリケーションの実行性能を最優先とする「アプリケーションファースト」の理念を実現するためには、「富岳」で培ったアプリケーションソフトなどの資産を有効活用できる電力効率の高いCPUと帯域重視の演算処理加速部を組み合わせた、高帯域およびヘテロジニアスなノードアーキテクチャを基本構成が目指すべきシステムアーキテクチャであり、特に、「富岳NEXT」の各システム構成要素については、以下の方針が望ましいとの結論に至った。

CPU部： これまで「富岳」で蓄積されたアプリケーションやシステムソフトウェア資産が活用できるよう、可能なかぎり「富岳」とバイナリレベルで互換性を持つべき

加速部： プログラミングや性能最適化の観点からユーザに使い易いものとなるようオープンなソフトウェアエコシステムが整備されており、また「富岳NEXT」の稼働前からコード移植を効率的に実施できるよ

う、加速部アーキテクチャが現状で広く利用可能であることが必要。そこで、大規模なスーパーコンピュータにおいて既に活用実績を持つGPUに基づくアーキテクチャを加速部として導入すべき

計算ノード： 複数のCPU部ソケットと加速部ソケットが搭載され、CPU部と加速部同士はキャッシュコヒーレンスを有する高速リンクで接続されるべき。また、ノード内の加速部同士は、複数の加速部を利用した並列処理が高速に実行できるよう高帯域なスケールアップネットワークで接続されるべき

ネットワーク： 各計算ノード間は、大規模なアプリケーションを効率的に並列処理できるようスケールアウトネットワークにより接続され、また計算ノード内の複数ジョブを考慮して、CPUソケットのみならずGPUソケットもスケールアウトネットワーク向けのネットワークインターフェースに接続されるべき

ストレージ： 計算ノードまたはジョブローカルで使用できる高速ストレージまたはキャッシュと全計算ノードで共有するストレージシステムを持ち、各階層において要求される帯域、IOPS、容量を実現するための構成にすべき。また、実運用で長時間稼働させたとしても安定した性能が維持されることが必要

2.1.3.3 目指すべきシステム性能と想定するアーキテクチャの概要

ベンダサブグループの調査結果である複数のアーキテクチャ候補とその性能や前節でまとめたシステム性能の指針を検討した結果、「富岳NEXT」に対しては、表 2.1.3.1に示すシステム性能を目指すべきであるとの結論が得られた。40MWのシステム消費電力を上限として、主に従来のHPCアプリケーション向けのFP64ベクトル性能、AI向けの低精度演算性能であるFP16やBF16、FP8の行列演算性能とSparsityを利用する際の性能、メインメモリサイズおよびバンド幅に対し、目標値を与えている。これらの実現可能性は半導体やパッケージングなどの今後の技術動向、経済情勢、および世界情勢により影響され得るものであるものの、「富岳」と大きく変わらない消費電力に対し、演算性能のみならず大多数のアプリケーションにとって重要なメモリバンド幅の大幅な向上を目指している。

また、前節で示した目指すべきアーキテクチャ基本構成および各システム構成要素の方針を元に検討を行い、図2.1.3.1の想定するシステムアーキテクチャ構成をまとめた。シミュレーションとAIの融合に向けたシステムアーキテクチャを想定しており、CPUと加速部（GPU）を高速リンクで接続し計算ノードを構成する。計算ノードは、2 CPU、4 GPUの構成を出発点としながらも、今後は実際の構成を決定すべく設計空間探索を進める必要がある。加速部のメモリは、目指すべきシステム性能を達成するために、導入時期における先端メモリ技術を利用しながらも、大容量のためにDDR系のメモリ技術をも利用することを視野に入れるべきである。

強・弱スケーリングの並列計算のどちらにも適するシステムネットワークを目指し、Scale-upネットワークとScale-outネットワークを組み合わせた構成を想定する。特に、Scale-upネットワークでは加速部間を高速リンクにより直接接続すると共に、Scale-outネットワークでは、例えば多段のスイッチを用いて多数の計算ノードを相互に接続する構成を想定する。システム全体では、数万規模の加速部ソケットを想定する。

表 2.1.3.1 目指すべき次世代計算基盤のシステム性能

項目	CPU	加速部	富岳
合計ノード数	3400ノード以上		158,976
理論 FP64ベクトル性能	48 PFLOPS以上	3.0 EFLOPS以上	488 PFLOPS
理論 FP16/BF16行列演算性能	1.5 EFLOPS以上	150 EFLOPS以上	1.95 EFLOPS
理論 FP8行列演算性能	3.0 ELOPS以上	300 EFLOP以上	ー
sparsity考慮の 同理論性能	ー	600 EFLOPS以上	ー
メインメモリサイズ	10 PiB以上	10 PiB以上	4.85 PiB
メインメモリバンド幅	7 PB/s以上	800 PB/s以上	163 PB/s
合計消費電力	40 MW以下（計算ノードおよびストレージ）		約30 MW

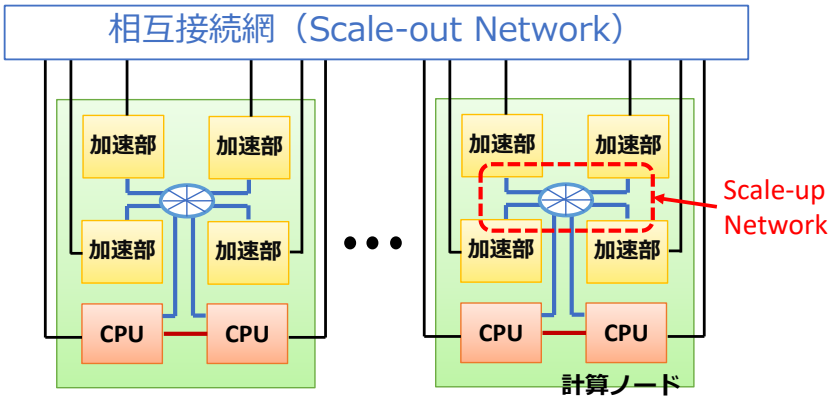


図 2.1.3.1　想定するシステムアーキテクチャの構成

2.1.4　AI アプリケーションに関する大容量メモリノードの評価

3TBもの大容量の主メモリを備えた大容量メモリサーバを用いて、大容量メモリを用いたCPU処理に対する以下のAI関連処理の性能評価を行った。

- LLM 推論のデコードステージにおける KV キャッシュクエリ
- 勾配による重み更新のためのモデル重みとオプティマイザ状態のオフロード

2.1.4.1　LLM 推論のデコードステージにおける KV キャッシュクエリ

LLMの推論は一般に、プリフィル（ユーザからのプロンプト全体を処理）とデコード（新しいトークンを1つずつ生成）の2つのステージから構成される。この2つのステージは、要求性能が大きく異なるため、それぞれに専用のハードウェアを使用した効率的な推論システムが望ましい。性能モデリングによると、デコードステージはメモリ参照に性能が制約されやすいベクトル-行列乗算の計算カーネルが支配的であるため、大容量メモリを搭載した計算ノードにおけるCPUを用いた計算でも十分な性能を達成できる可能性がある。

LLM推論のデコードステージにおけるKVキャッシュクエリに対する大容量メモリサーバの性能を評価するために、PyTorchディープラーニングフレームワークを使用し、Benchmarkerフレームワーク内のモジュールとして、独立したKVキャッシュアテンションを実装した。異なるLLMとシーケンス長に応じて、KVキャッシュアテンションのレイテンシを経験的に測定し、これを、GPUによる等価な処理およびそれに伴って発生するデータ移動の遅延と比較した。

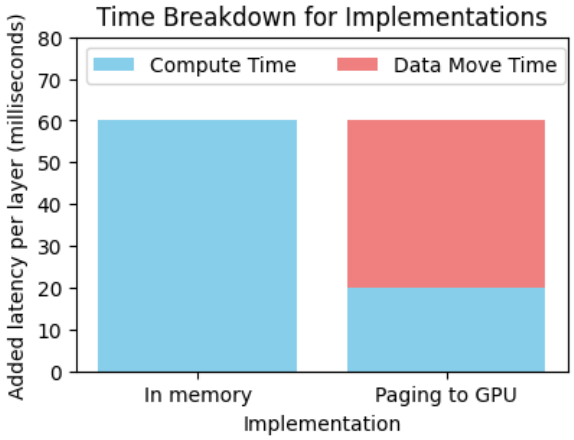


図 2.1.4.1　計算とデータ移動の時間内訳
(in-memoryは大容量メモリサーバでの実行を示す)

KVキャッシュアテンションを最も高速に実現する方法は、GPUメモリに完全に格納することである。しかし、限られたGPUメモリ容量のために、シーケンス長に制約が生じてしまう。この制限を克服するためのいくつかの方法が考えられる。性能の観点からは理想的な解決方法はKVキャッシュを複数のGPUに分散させることと考えられるが、これはシステムのコストを大幅に増大させる。限られたGPU資源を考慮すると、一般的な方法は、KVキャッシュを大容量メモリサーバなどの分割メモリに保持し、推論中にデータをページ単位でGPUに移動させて処理することである。我々は、大容量メモリサーバ上でCPUによりKVキャッシュを用いたアテンション計算（ベクトル行列積やSoftmax計算）を行うのが、CPU-GPUオフロードに

勝ることを確認した。尚、 PyTorchのCPUによるベクトル行列乗算はあまり最適化されておらず、その性能が今後改善される可能性がある。

2.1.4.2 勾配による重み更新のためのモデル重みのオフロード

ニューラルネットワークは通常低い数値精度で学習されるが、数値計算の安定性のために、重みを更新する際の重みのマスターコピーはFP32のような高精度で保存される。同様に、モデルの重みの4倍以上のメモリ空間を必要とするオプティマイザー・パラメータも高精度で保存される。オプティマイザ更新は主に演算性能ではなくメモリ帯域にその性能が制限されることから、GPUなどのアクセラレータ上で実行する必要性は低い。勾配による重み更新のためのモデル重みのオフロードに対する大容量メモリサーバの性能を評価するために、AdamWのような一般的なオプティマイザPyTorch実装を使用して、ベンチマークフレームワークにおけるモジュールとしてオプティマイザ更新を実装した。データをGPUに移動してGPUメモリで更新処理を行う場合と、大容量メモリサーバ上で更新処理を行う場合について処理時間の比較を行った。

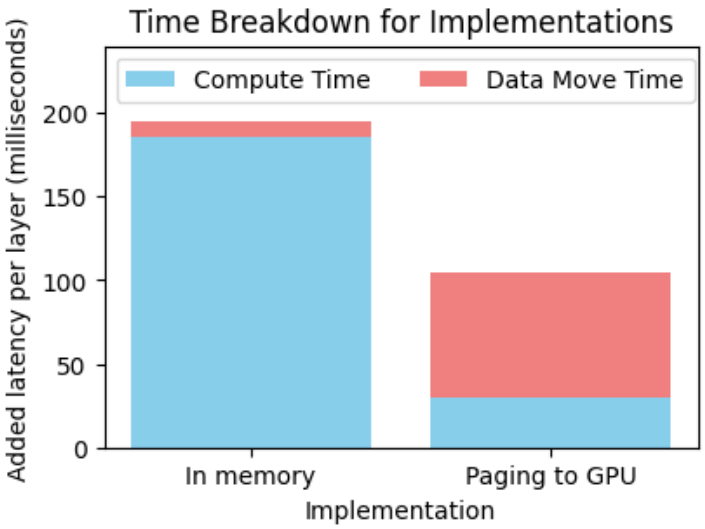


図 2.1.4.1 計算時間とデータ移動時間の内訳
（in-memory は大容量メモリサーバでの実行を示す）

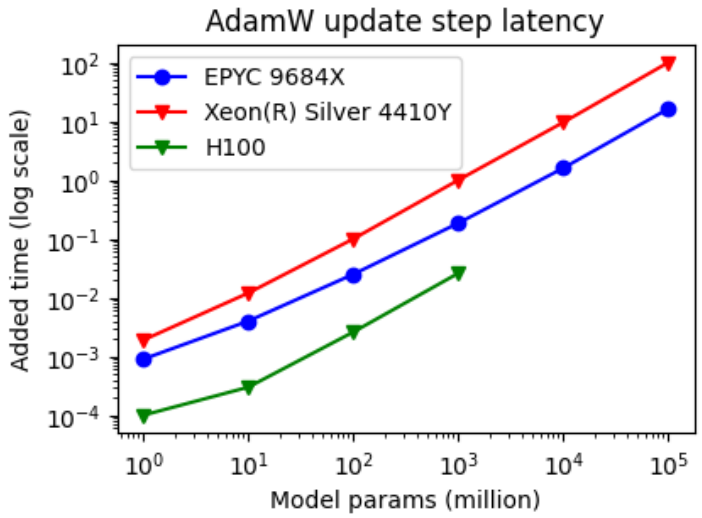


図 2.1.4.2 異なる CPU･GPU による AdamW 更新処理時間

その結果、GPU上でAdamWオプティマイザ・ステップを計算する場合は最大で10倍程度高速となるが、大きなメモリ容量を使用するため、あるサイズ以上のモデルではオプティマイザ状態をGPUメモリに保持することが不可能となることが確認された。そのため、非常に大きなモデルの学習の場合にはGPUの他のメモリを使用することが不可欠となる。また、データを大容量メモリサーバに置く場合は、GPUによる更新処理時間自体は大変短いもののデータをGPUに移動する時間が処理時間よりも大幅に長く、そのため、処理時間とデータ移動時間の合計は、大容量メモリサーバ上でのCPUによる更新処理と大差無くなる結果が得られた。より高いメモリ帯域を持つ大容量メモリサーバを用いることで、CPUによる処理時間はさらに短縮することが考えられる。

2.1.5 GPU ノードの性能評価

次世代計算基盤の目指すべきアーキテクチャとして、加速部にはGPUに基づくアーキテクチャを導入すべきという指針が得られたことから、GPUのアーキテクチャ特性や既存のコードでの性能効率を調査するために、現行で利用可能なGPU計算機をいくつか導入し、代表的なベンチマークアプリケーションにより評価を行った。特に、CPUと加速部との接続形態によるアプリケーションの実行性能への影響はアーキテクチャの決定にも重要となることから、ここでは同じGPUアーキテクチャでありつつCPUとGPUが密に接続されているアーキテクチャと、CPUとGPUが別のソケットとして相対的に疎に接続されているアーキテクチャの比較評価の結果を例として示す。なお、密結合のアーキテクチャとして、AMD社のMI300Aを搭載したGPUサーバを、また疎結合のアーキテクチャとして、同じくAMD社のMI300Xを搭載したGPUサーバを用いた。評価したベンチマークは、NAS Parallel Benchmarkのカーネル、および実アプリケーションのいくつかである。結果を図 2.1.5.1に示す。

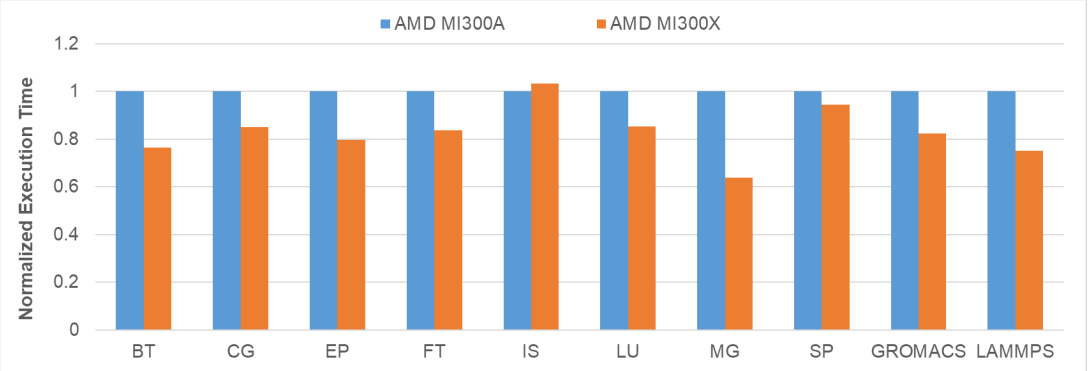


図 2.1.5.1 図 2.1.5.1 結合方式の違いによる性能比較の一例

図より、多くの場合で疎結合であるMI300Xの実行時間が短いことがわかる。MI300XではGPUソケットが全てGPUコアで構成されており、APU構成であるMI300Aに比べてメモリバンド幅を有効できたこと、また演算加速に用いるGPUコアの数が多くピーク性能が高いため、CPUとGPUが密結合されているMI300Aの利点よりもそちらの要因が性能向上に貢献したためと考えられる。このため、一般的なアプリケーション実行においてAPUのような密結合型の構成が必ずしも最良とはならないと言える。

2.2　アーキテクチャ調査研究サブグループ 1（理研）

2.2.1　調査研究の概要

強スケーリング性能を有する、準汎用～専用加速装置アーキテクチャの検討を行っている。理化学研究所 生命機能科学研究センター 計算分子設計研究チームにおいて蓄積がある分子動力学シミュレーション専用計算機を題材に、まず専用加速装置としての開発を実施し、それをベースとしてその汎用化・準汎用化された演算加速装置の可能性を検討する。特に、分子動力学シミュレーション専用計算機の演算能力の中核であった非結合力計算の加速に加え、同期機構・メモリ内演算などの周辺回路における加速を重点的に検討すると同時に、これまで限定的にしか行っていなかった結合力計算の加速の検討を行っている。

今年度は、昨年度にFPGA実装により基本動作が検証された回路設計を発展させ、以下の回路の改良・LSI実装のための設計検討を行った。

- 各モジュールの最適化
- LSI 全体の最適化・パラメータ化の強化
- 設計共有の強化・パラメータ設定の共通化
- ASIC 実装の具体的検討

2.2.2　各モジュールの機能強化・最適化

前年度に分子動力学計算に必要な回路設計（図 2.2.2.1）を行い小規模な計算が実行可能であることを回路シミュレーションやFPGA実装で確認した。論理回路としては必要な機能は完成したが、実行速度や回路規模には改善の余地があることも確認された。これらの結果に基づき、性能向上のための改良を行った。

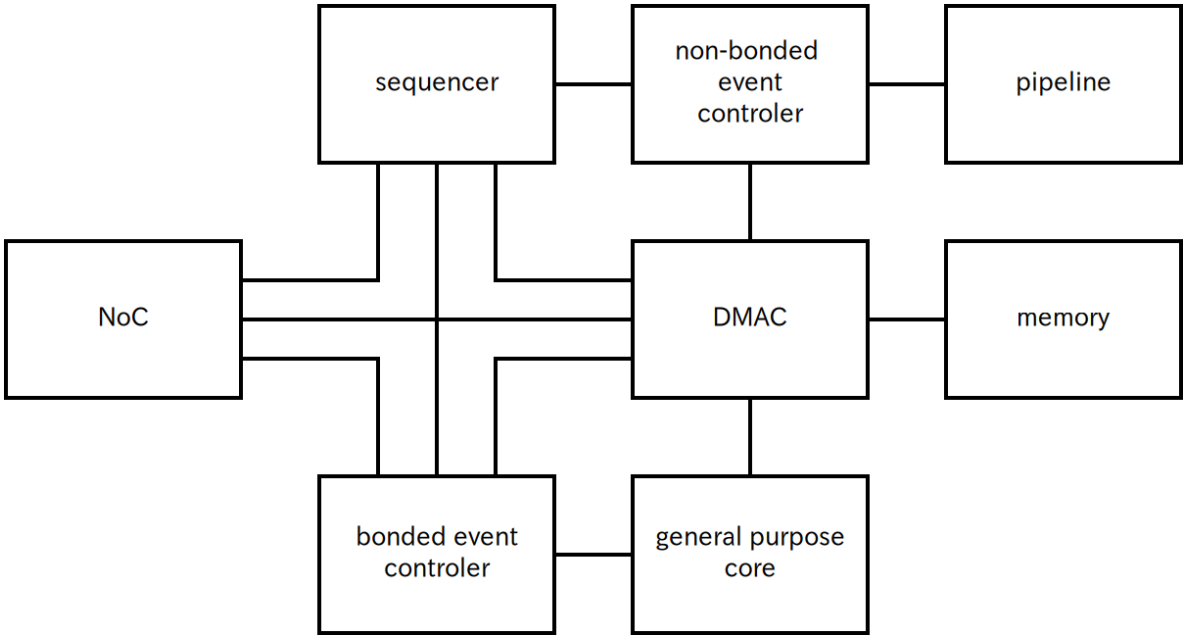


図 2.2.2.1　CU ブロック図

2.2.2.1　非結合力計算加速パイプライン

2.2.2.1.1　原子ペア振り分けアルゴリズムの改良

前年度は非結合力計算加速パイプラインを４つもしくは８つ並列に配置していたが、回路規模によって任意の

数を並列に配置できるように変更した。

以前の原子ペアを振り分けるアルゴリズムは１６個の原子ペアの中から有効なものを４つに分けて各パイプラインに分配する方式であり、４つに分ける際均等に分ける保証がなかったので幾つかのパイプラインがストールする可能性があった。また４つに分けるのでパイプラインの数が４の倍数しかサポートできていなかった。

このアルゴリズムを、原子ペアを各々のパイプラインに均等に振り分けるように改良した。これによりパイプラインの並列数を任意の数に設定することが可能になり、レイテンシは少し増えたがストールする事がなくなり、スループットの向上を達成できた。

今年度の小規模計算のためのFPGA実装用には一つのコアに対してパイプラインを４つ及び６つ並列に配置したものを試したが、FPGAに載せられる面積の制約により最終的には並列数は４とした。

2.2.2.1.2　パイプライン段数のパラメータ化

以前はパイプラインの段数が固定値だったがパラメータ化し、回路規模やクロック周波数に応じての調整を可能にした。

2.2.2.2　汎用コア

2.2.2.2.1　乱数生成

前年度の設計では、CPUに乱数生成モジュールは含まれていなかった。しかし、分子動力学シミュレーションのアルゴリズムの中には非常に大量の乱数を仕様するものがある。具体的には、ランジュバン動力学による時間積分である。これを高速に実行できるよう、各CPUにハードウェアに向けて最適化された軽量な乱数生成モジュールを搭載し、乱数生成命令を追加した。

乱数生成モジュールはパイプライン化し、各スレッドが同時に生成リクエストを発行できるように設計した。乱数生成命令は多くの場合x, y, zの3方向に対応する独立した乱数を要求するため、同時に4つの浮動小数点数を生成できるように設計した。また、標準正規分布に従う乱数を使用する機会が他の分布に対して非常に多いため、このサポートも導入し、乱数生成命令に標準正規分布と一様分布を加えた。

さらに、標準正規分布に従う乱数を生成するレイテンシは他に比べて非常に長いため、この命令は可能な限り早く実行するようコンパイラ内での優先順位づけを行った。

2.2.2.2.2　制御機構の統一化

前年度の設計では、CPU内の制御はそれぞれの計算モジュール・分岐モジュールが独立に行っていた。このため、それぞれのレイテンシの設定が分散し、データ書き込みポートでの衝突の制御が難しくなっていた。

今年度では各スレッドの進行状況の制御を行うモジュールを追加し、分散していたレジスタを集め、処理を集中させた。その過程で衝突検出などを自動化し、一部の命令のレイテンシを衝突がなくなるように変更した。

2.2.2.2.3　命令セットの整理

前年度の設計では、条件付きでデータを移動する命令が限られていた。分岐命令は成立判定を行ってからデータを移動するため、命令数が増加してしまう。今年度は、条件付きデータ移動命令を増強して分岐命令を置き換え、いくつかの関数で命令数を削減した。

また、前年度は固定値を設定する命令が限られていたため、定数をメモリから読み込む命令が多くあった。今年度は、いくつかあった固定値を設定する命令を一般化する形で命令を追加し、メモリアクセスの回数を減らした。

2.2.2.3 メモリ

実際のシミュレーションに近いデータで動作させ、いくつかの考慮漏れやボトルネックを発見したため、その修正と改善を行った。以下ではその中でも代表的なものを示す。

2.2.2.3.1 書き込みデータの損失耐性

ネットワーク越しにデータの読み書きが発生する場合、データがエラーによって必要な個数送られてこない場合がある。本システムのメモリは基本的にストリーム書き込みを行うため、事前に決定した長さのデータが送られてこない場合、データ待ち状態から復帰できなくなる恐れがある。この問題を回避するため、実際に送信されてくるデータ量をモニタし、本来よりも少ない量しか送られてこなかった場合、正常終了信号を出さずにアイドル状態に戻れるよう変更を行った。

この機能がメモリに備わっていない場合、確実にデータが揃うまでネットワークは受信したパケットの送信を開始できないことになり、パケットの長さ分のレイテンシがかかるようになる。メモリ内部で途中終了からの復帰をサポートすることで、ネットワークは受信完了と誤り検出の完了を待たずに続くノードへデータを送信できるようになり、レイテンシが向上した。

2.2.2.3.2 Direct Memory Access の並行処理

前年度の設計では、メモリと他の各モジュール間のデータ転送は同時に一つのみを処理していた。しかし、一部の転送は同時に実行することが可能である。例として、CPUの持つスクラッチパッドからメモリへの転送とメモリからネットワークへの転送は、異なるポートを使うため同時に実行できる。このような場合においてデータ転送を効率化するため、DMA Controllerの再設計を行った。

今年度に設計したDMA Controllerは、リクエストを中央で一括管理するのではなく、一度read側とwrite側に分割して各ポート側に保存し、中央ではそれらのリクエストで対応関係にあるものを接続するという構造に変更した。これによって、ポートのそれぞれが独立に状態を変更できるようになり、同時に複数の接続が成立するようになった。

ただし、特定のポートが混雑していた場合にリクエストが長い間実行されずに留まることのないよう、リクエスト順の入れ替えには制限をかけた。例えば、AとBを接続するリクエスト1と、AとCを接続するリクエスト2が発行されており、Bが使用中だった場合、リクエスト2が優先的に処理され、AとCが接続される。この処理中にBとDを接続するリクエスト3が発行され、かつBが開放された場合、リクエスト3は処理可能ではあっても、Cが使用可能になるのを待ってリクエスト1によるBへの接続が優先される。これがない場合、短いリクエストが大量に処理されている間、先んじて発行されていたリクエストの処理がいつまでも完了せず、それに控えたタスクの実行が開始されない恐れがあるからである。リクエスト1に依存するタスクはリクエスト3移行のデータ転送とはオーバーラップできる可能性があるため、リクエスト順序をある程度守って優先処理することによってタスクの並列度を向上させられると期待できるためである。

この変更によって、リクエストは必ずしも発行順に実行されるとは限らなくなったため、発行時にIDを付与し、完了時にもIDを提示して識別できるように変更した。

2.2.2.4 イベント制御

イベント制御の主な要素は、シーケンサ、結合力計算イベント制御、非結合力イベント制御である。各要素について、パラメータ化強化並びに後述するtoml形式でのパラメータ設定対応を実施した。

2.2.2.4.1 シーケンサ

計算ステップの進行を、座標積分、結合力計算、非結合力計算といった大きな粒度のタスクを管理する。タスク間の依存関係を管理し、タスクの完了を検知し依存関係から実行可能となったタスクを起動する。

実装の効率化、パラメータ化の強化を行った。

2.2.2.4.2　結合力計算イベント制御

結合力計算を管理する。結合力計算に必要な粒子座標の更新を監視し、計算可能となった結合力計算を汎用コアに指示する。結合力計専用メモリへの粒子座標格納や計算された結合力の後処理も行う。

モジュール構成の変更や中間データの処理の見直し、結合力管理データメモリの変更等で回路規模や実行効率を改善した。

2.2.2.4.3　非結合力イベント制御

非結合力計算を管理する。非結合計算に必要なセル(計算粒子系の分割単位)を監視し、計算可能となったセル間非結合力計算を非結合力計算パイプラインに指示する。パイプラインからのセルリード要求を処理しメモリや他の計算並列化単位からの読み出しを実行する。パイプラインからの計算結果ライト要求を処理しメモリや他の計算並列化単位への書き込みを実行する。モジュール構成の変更やセル管理方法の変更等で回路規模や実行効率を改善した。

2.2.2.5　オンチップネットワーク

2.2.2.5.1　パケットの順序を保存するよう変更

前年度に、最低限の機能を持つオンチップネットワークの設計を完了し、今年度はこれを使用して実際にパケットの送受信を行うシミュレーションを行った。

前年度の設計では、負荷が高まって一部の経路上でパケットのバッファが埋まった場合にネットワーク全体のスループットを向上させるため、チャネルを多重化して送信するパケットの切り替えを行っていた。このため、負荷状況とタイミングによっては、送信元と受信先のアドレスが同一のパケットでも送信順と異なる順番で受信される可能性があった。パケットによっては順序に依存して結果が変動する種類のデータが送られる場合があるため、この現象を避けるために経路計算アルゴリズムを変更し、順序が切り替わることのないようにした。この変更にあたって、ネットワーク全体の性能を損なわないよう、混雑している経路と同一の経路を通るパケットのみをブロックし、異なる経路を通るパケットには切替可能になるように変更を加えた。

2.2.2.5.2　パケットのエンコーダ・デコーダの自動生成

前年度には、パケットのデータフィールドの位置と幅は固定して設計を行っていた。しかしながら、本研究では各種設定をパラメータ化して、動的に変更しながら最適解を探索することを計画している。それにあたって、データフィールドの幅が変更される可能性があり、付随して位置も変動する可能性がある。そのような変更に対してロバストな設計となるよう、エンコーダ・デコーダを自動的に生成するようなシステムを開発した。

このシステムでは、まず設定ファイルにデータフィールドの名称とその幅を記述する。この設定ファイルは他のモジュールの設定と整合性が取れていなければならない。そのため、設定ファイルの読み込み部分に相互参照の仕組みを導入した。Chisel言語側では読み込んだ設定に従った幅を持つ構造体を合成し、さらにパケットからその構造体への変換モジュールを自動生成する。入力データからパケットを生成するソフトウェア側でも同様に、同じ設定ファイルから動的に幅を決定できるクラスを生成し、同一の規則に従ってパケットのバイナリデータに変換するようなライブラリを開発した。これによって、送受信するパケットのフォーマットを容易に変更可能になり、また各モジュールでのエンコード・デコードの一貫性も向上した。

2.2.2.6 オフチップネットワーク

チップ間通信のためのオフチップネットワークの検討をすすめた。チップ間通信はLSIプロセス等実装技術との関係が深いため、4.ASIC実装の具体的検討 とともに行い継続中」である。

2.2.3 LSI 全体の最適化・パラメータ化の強化

各機能モジュール個別の最適化と並行してLSI全体としての改良を実施した。

具体的には、モジュール構成の変更・パラメータの調整を行った。

2.2.3.1 モジュール構成の変更

機能単位ごとにモジュール化して開発をしていたが、全体を統合した評価でモジュール分割によりデータ転送やデータ処理の非効率化が生じた。そのため、機能分割を再検討し、処理効率を上げる構成変更をすすめた。

2.2.3.2 パラメータ化の強化

当初からパラメータ化を意識していたが、全体構成の変更等には不十分であったためパラメータ化を強化した。

2.2.4 設計共有の強化・パラメータ設定の共通化

2.2.4.1 設計共有の強化

モジュールはそれぞれ独自の機能を実装しているが、複数のモジュールで類似した処理が必要となることがあるため、利用頻度が高い回路記述をライブラリ化し再利用性を向上させた。具体的には通信データやモジュール間インターフェイスを定義・デコード・エンコードする関数群、算術演算回路、回路シミュレータから利用するインターフェイス等である。これらのライブラリを共有することで、設計、モジュール間接続、回路シミュレーションとのパラメータ共有が効率化された。

2.2.4.2 パラメータ設定の共通化

回路設計はChiselを使用しパラメータ化を行ったが、回路シミュレーションやFPGA・LSIへの実装時にはC++やスクリプト言語等様々な処理系を併用する。異なる処理系で設計パラメータを共有するため汎用の設定記述フォーマットであるtomlを採用した。tomlは平易なテキストフォーマットで複数の処理系でライブラリが提供されており、回路設計・実装・回路シミュレーションから計算データの生成まで設計パラメータを共有することが容易になった。また、回路記述やシミュレーションコードとは独立に記述するため整合性を維持したパラメータ変更が容易となった。

2.2.5 ASIC 実装の具体的検討

TSMC 7nm (N7)プロセスを用いたASICで実装し並列接続した構成（図 2.2.5.1）を前提とした設計最適化を開始した。本内容については、現時点では詳細は開示しない。

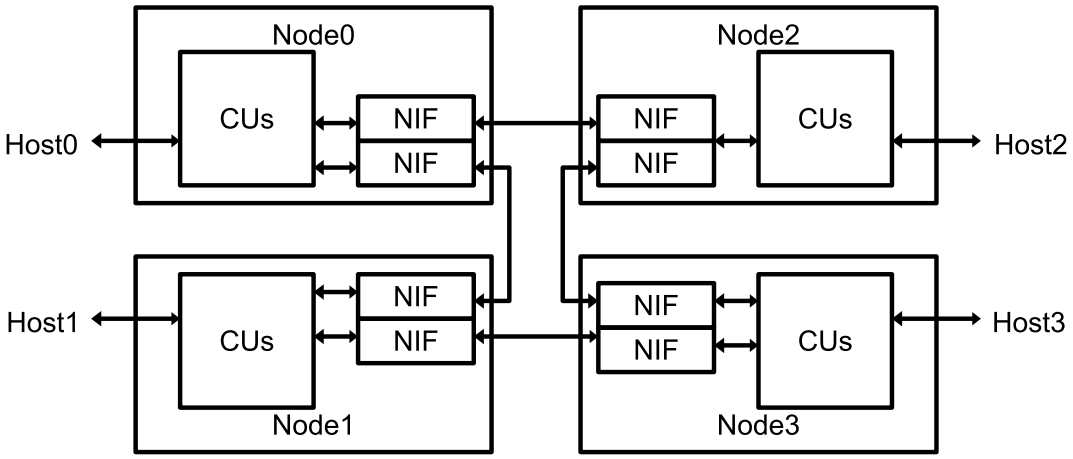


図 2.2.5.1　LSI 並列実装模式図

2.3 アーキテクチャ調査研究サブグループ 2（富士通）

本年度は富岳後継機のアーキテクチャについて、前年度までに実施した技術動向調査およびノード・システムアーキテクチャ検討の結果をベースに、開発に向けた具体化と絞り込み検討を実施した。また、初年度の技術動向調査から2年が経過したため、技術動向の変化に対応するために、見積り等で数値が必要となった際は技術動向の追跡調査を適宜実施した。

2.3.1 調査研究の概要

ノードアーキテクチャでは三次元実装とチップレットを含む先進パッケージング技術の活用と、高帯域メモリの統合を前提として、プロセッサとアクセラレータを含むヘテロジニアス・アーキテクチャの検討を実施した。本年度は、プロセッサはコア数を増やしたType-D、アクセラレータは再構成可能アーキテクチャにベクトル演算部を追加したType-F、メモリはプロセッサにDDR、アクセラレータにHBMを接続し、大型ソケットを複数接続するType-Dに絞り込んだ上で検討を開始した。

プロセッサではAI/HPC向けのCPU拡張として、SME(Matrix演算)の実現性検討を行った。具体的には、1.4nm/1.6nm半導体テクノロジ調査、前記テクノロジ情報を元にしたSMEのPPA評価(周波数、電力、面積評価)、SMEのマイクロアーキ制御方式の検討、SMEのソフトエコシステム開発状況の調査を行った。
1.4nm/1.6nm半導体テクノロジ調査については、ロジック部、SRAM部、IO部それぞれについて、半導体のテクノロジノード間の面積シュリンク係数、電力シュリンク係数の調査を行った。SMEのPPA評価については、FP64/FP32/FP16/BF16/FP8/INT16/INT8の各Matrix演算器とZAレジスタの構成検討、および、演算器のパイプライン段数、電圧条件、温度条件を振って、RTL合成によるトレードオフ評価を行い、周波数、電力、面積の概算値を算出した。SMEのマイクロアーキ制御方式の検討については、既存のSVEの制御回路との共存を考慮し、SMEをコアに配置した場合の制御方式の検討を行った。具体的には、OOOの制御単位、ZAレジスタのリネーミング有無、分岐予測ミス時の復元方法、非同期割り込み発生時の同期方法等の検討を行った。 SMEのソフトエコシステム開発状況の調査については、コンパイラ、数学ライブラリ、開発支援ツール、OS、仮想化のカテゴリごとに、SMEの現状のサポート状況と今後の開発予定を調査し、富士通開発機能の明確化を行った。また、本年度の最終成果として、これらの検討結果をもとに、CPU諸元の検討を行い、コア数、周波数、演算器、キャッシュメモリ、IO構成、3Dチップレット構成、ダイサイズ、Si-IPサイズ、PKGサイズ、ダイ電力、PKG電力の各諸元を策定した。

アクセラレータでは再構成可能アーキテクチャの改善とさらに進んだプロセス技術の可能性検討に取り組んだが、開発環境整備およびエコシステム醸成の目途が立たないことから第1四半期中にType-Fの検討深化を中止し、外部IPの活用を検討する方針に切り替えた。しかし入手可能な外部IPでは十分なエコシステム発展の見通しが得られず、最終的には既存GPUを利用する方針に転換した。

システムアーキテクチャでは大型ソケットを複数接続するFatノードと、パッケージ統合型光部品(CPO)によるシリコンフォトニクス統合を両立する新しいネットワークトポロジを検討した。当初は2つの大型ソケットによるノード構成を想定していたため、ソケット毎に2系統、合計で4系統のマルチレイル型のトーラス系ネットワークを検討した。既存GPUを利用したノードではさらにソケット数が増えることから、直接網型のx-fly系階層化ネットワークも検討した。しかし、CPOの高帯域に見合うCPU-GPU間帯域を確保する見通しが得られないこと、GPU間の密結合ネットワークと二重投資になることから、最終的にはCPU側にCPOを統合しない方針を採用した。さらに、下半期からは実装検討も開始し、電源供給、冷却を含む実現可能性の見積精度を向上した。

2.4　アーキテクチャ研究 サブグループ 3(Intel)

インテルは、「富岳」後継として検討が開始された「富岳NEXT」に向けた実現可能性に関する研究と調査を継続し、これまで実施した技術動向調査およびノード・システムアーキテクチャ検討の結果をベースに、より深い検討を実施した。

2.4.1　調査研究の概要

CPUとアクセラレータの要求が変更されたことで、強力なオープンソースソフトウェア環境と、容易にプログラム可能で手頃な価格の低電力システムの構築を可能にし、メモリの革新的なコンセプトを持つアクセラレータ構造に焦点を当て、リスク分析を含めた技術面での調査研究とコスト概算を実施した。インテルのオープンソースHPCソフトウェア環境（OneAPI）は、米国アルゴンヌ研究所で稼働しているAurora HPCシステムの初期ユーザから、非常に好意的なフィードバックが得られており、他のアクセラレータプラットフォームに比べより容易に使える環境との評価が多くのユーザから報告されている。したがって、インテルは、「富岳NEXT」の想定されるホストプロセッサと共に使用するOneAPIに基づくアクセラレータソリューションを提案し、調査研究を実施した。OneAPIの開発に使用できる既存のプラットフォームを、暫定的なHPC/AI向けGPUを早期プラットフォームとしてアプリケーションとパフォーマンスを検証するために利用した。大規模ファイルシステムにおける実証・評価は、「富岳」上でのソフト実装方法を検討・検証し、代替機を用いた最小構成での動作・性能検証を進めた。

メモリのコスト、帯域幅、容量、および電力が重要であるため、インテルは現在一般的に利用されているHBMに基づくプラットフォームよりも優れたプラットフォームを実現するための新しいメモリアーキテクチャとアクセラレータ構造の調査検討を進め、革新的なメモリと組み合わせたアクセラレータプラットフォームを提案した。リスクを低減する観点から、提案されたアーキテクチャは標準およびカスタムHBMもサポートでき、いずれのメモリにおいても利点を享受できるような革新的なアクセラレータ構造を調査した。新しいメモリは従来のHBMと異なりメモリをアクセラレータからオンサイトで取り外したり取り付けたりできるようになり、不具合のあるメモリや不具合のあるアクセラレータの特定を容易にし、それぞれ個別に修理・交換できることで、HBMを使った場合と比較して、システムの導入と検収をはるかに短期間で実施することが可能となる。HBMソリューションと比較して、提案された新しいメモリシステムはメモリダイの消費電力を削減するだけでなく、ホストメモリコントローラが持続的にメモリ帯域幅を得るために必要な動作を大幅に削減できるため、ホスト側の消費電力も大幅に削減される。この「富岳NEXT」研究調査において発案されたメモリ提案は、現在「富岳NEXT」プログラム外でも大きな支持を得ており、公開され利用可能になった際には業界に大きな影響を与えることが期待されている。

提案されたメモリアーキテクチャとアクセラレータ構造のモデリングは、計算負荷とメモリ負荷の両方が高いワークロードを実行する際に、同じ「富岳NEXT」の電力制限の元で標準的な構造のものに比べ5倍以上の電力を計算に供給できる能力を示している。これは、高い電力効率で非常に高いアプリケーション性能を実現できるアーキテクチャであることを意味する。

全体的なアーキテクチャは、スケールアップおよびスケールアウトのための標準ベースのインターフェイスをサポートし、柔軟なトポロジをサポートする。ホストインターフェースは、プロセッサ要素への標準インターフェイスを拡張することにより、キャッシュコヒーレンシを持ち非常に高い帯域幅での接続を実現する。

2.5　アーキテクチャ調査研究サブグループ 4 (AMD)

2.5.1　　調査研究の概要

AMDは、Supercomputing 2024においてHPC分野での継続的リーダーシップを披露した。最新のTOP500リストによれば、世界最速のスーパーコンピュータ上位10システムのうちの半数の５システム、また電力効率のランキングであるGreen500リストにおいては上位10システムのうち４システムにAMDの技術が採用されていることが示されている。さらに、Lawrence Livermore National Laboratory（LLNL）およびHewlett Packard Enterprise（HPE）との協力の下、AMD EPYC™ プロセッサ、AMD Instinct™ アクセラレータ、ROCm™ オープン・ソフトウェア・プラットフォームを採用した「El Capitan」が発表され、これによりTOP500で1位を獲得した。1 EFLOPSを初めて超えた「Frontier」はTOP500で2位に位置しており、これらの実績は、AMDが世界のHPC市場において揺るぎない地位を確立していることを示している。

こうした実績とHPC分野における豊富な専門知識を背景に、AMDは富岳NEXTの計算ノードアーキテクチャ候補技術を検討してきた。提案されたアーキテクチャは、演算器、メモリ、通信を密に統合することでデータ移動のオーバーヘッドを大幅に削減し、電力あたりの性能向上を実現することを目指している。さらに、AMDは理化学研究所から得たなフィードバックと、社内で実施した詳細な解析結果を活用し、提案されたアーキテクチャコンセプトおよび性能予測の精度向上に向けた取り組みを継続的に推進した。具体的には、システム全体の統合性能評価、電力効率の最適化、さらには大規模システムにおけるスケーラビリティ検証など、実際の運用を視野に入れた各種シミュレーションと検証作業を通じて、アーキテクチャの改善に必要な知見を蓄積してきた。

このように計算ノードアーキテクチャのコンセプトをさらに検討した上で、AMDはアクセラレータのピーク性能に関する初期予測を行った。また、理化学研究所で利用されるHPCアプリケーションや急速に需要が高まるAIアプリケーションを対象としたベンチマーク性能の予測結果とその手法の詳細をまとめたレポートを提出しており、これらの資料は今後の設計改善および性能向上のための基礎資料として、研究開発の方向性をより明確にする重要な役割を果たすと期待される。

富岳NEXTが掲げる方向性と求められる要件を正確に把握することで、より洗練されたシステム設計が可能となる。これまでの調査の成果からは、今後注力すべきプロセッサ、アクセラレータ、パッケージングや積層技術、メモリおよび通信技術、さらにはソフトウェアといった各領域が明確化された。AMDの幅広い専門知識、これまでの革新実績、並びに各種機関との協働の成果が、富岳NEXTの成功を力強く後押しする基盤となると評価される。

2.6 アーキテクチャ調査研究サブグループ 5（NVIDIA）

2.6.1 調査研究概要

生成AIにより、社会環境が大きく変化した。世界中の多くの企業で生成AIの活用が進み、データセンターの需要が大幅に増えたことで、アーキテクチャの実行効率や電力性能は社会的にもより一層重要なものとなった。科学技術計算で多用されてきた大規模な計算基盤を用いた演算のノウハウは、生成AIの学習においても多くのAIエンジニアにとって必要なものとなった。

富岳NEXTは、当初より電力性能やアプリケーション実行性能を重視し、AIの科学技術活用（AI for Science）やエコシステムの確立など未来志向のコンセプトとなっており、NVIDIAの開発方針と合致している。

本調査研究では、NVIDIAのアーキテクチャの方針や将来計画を報告するとともに、これまでNVIDIAが確立してきたエコシステムをいかに日本技術と組み合わせ、パートナーとともに新たな価値を創出できるかを検討した。

2.6.1.1 アーキテクチャ

昨今のAIの技術革新スピードは凄まじく、NVIDIAは多くの主要なAI モデル開発者（モデルビルダー）と協業することで、最新かつ最適なアルゴリズムをハードウェアに反映している。最近の例では、2022年末より販売を開始した HopperアーキテクチャにおけるTransformer Engineなどがある。AIアルゴリズムの技術革新は、数年前から見通すことは困難であり、技術の変化に応じてスピーディに対応できることが非常に重要と考えている。

本調査研究を通して、富岳NEXTの計算ノードアーキテクチャの候補とその性能について、理論ピーク性能、電力、電力効率、メモリバンド幅等のベースラインの見通しを提示した。

2.6.1.2 CPU との接続手法

NVIDIA GPUは従来より、他社製汎用CPUをホストプロセッサとしてきた。2023年からは、CPU―GPU間が高バンド幅でコヒーレントに接続可能な自社製Arm CPU―GPUを搭載した「Grace Hopper Superchip」の発売を開始し、より多様なワークロードに対応可能となった。しかしNVIDIAは自社CPUに特化するのではなく、引き続き他社製汎用CPUとの接続についても、積極的に技術開発を進めていく。本調査研究では、CPU―GPU接続について、具体的にどのような選択肢があるかを示した。

2.6.1.3 アプリケーションサポート

GPUスーパーコンピュータを成功に導くにあたって最も重要な挑戦の一つが、アプリケーションを並列化して高速化（以降：「GPU対応」という）とそのための開発者サポートである。NVIDIAでは、Fortran, C /C++/ Python など様々な開発言語に対応したソフトウェアを提供し、CUDAだけでなく、 ディレクティブベースのOpenACCや OpenMP、標準言語による並列化などユーザが選べる開発環境を提供している。

しかしそれでもなお、GPU対応を開発者だけで行うのは時に非効率であり、開発者同士の互助や、アクセラレータベンダからの支援は不可欠である。NVIDIAでは、これまでNVIDIA GPUを採用しているHPCIの各センターと連携し、数多くの開発者支援を行ってきた。これらのノウハウは、富岳NEXTにおいても活用できると考えている。

アプリケーションの中には、開発者が多数いるコミュニティコードや、複数の物理モデルが相互に関連しているコードなど、GPU対応に時間を要するものもある。これらは、できるだけ早い段階からGPUベンダとGPU化を進めることで、富岳NEXTの早期の成功につながるものと考える。

なお、アプリケーションベンチマークおよび、移植時の提案については、昨年度のレポートにて提出した。

2.7　アーキテクチャ調査研究サブグループ 6（HPE）

2.7.1　　調査研究の概要

ポストエクサスケール・システム・アーキテクチャに対してHPEが調査を行った複数のアプローチを以下に示す。

この10年の後半（2025～2030年）には、気候変動などの地球規模の課題に対応するために世界が必要とする科学的発見のペースを維持し、スマートシティなどの社会的目標を達成するために、HPCとAIを活用したワークロードとワークフローの組み合わせが不可欠になる。HPEは、インテリジェントHPCシステムの利用がこれらの課題に対応するために不可欠であると考え、ここで説明する複数の方法に投資している。さらなるコラボレーションにより、その効果を加速できる可能性がある。

HPEは、HPE社内で開発されたテクノロジ、業界の主要テクノロジ・パートナ（AMD、NVIDIA、Intelなどの長期的な実績を持つベンダ、CerebrasやNextSiliconなどの新興企業の双方）との協力、将来のワークロードやワークフローのニーズに関する重要な洞察を提供し、将来のシステムアーキテクチャを形成する顧客/エンドユーザとの協力を通じて、将来のシステムアーキテクチャのニーズを満たすことに注力している。

HPEは、将来のワークロードの新たなデータニーズに対応するための継続的な取り組みの一環として、最近、IntelからDAOS開発チームを買収し、DAOS（Distributed Architecture for Object Storage）オープンソースソフトウェアプロジェクトへの投資を大幅に拡大した。HPEは数年前からDAOSプロジェクトに関与しており、コミュニティにオープンソースの貢献を提供するとともに、DAOSストレージシステムの概念実証テストにおいて顧客と協力してきた。HPEは、DAOSが新たなHPC、HPC_AI、AIのアプリケーションやワークフローに必要なストレージ性能を提供する鍵になると考えている。（DAOS Linux Foundationのオープンソースプロジェクトの詳細は、https://daos.io/を参照のこと。DAOSの性能の可能性は、IO500（https://io500.org/）の最新の結果で確認できる。）HPEは、富岳NEXTのためのDAOSストレージ機能を共同で構築する用意がある。これには、将来的なソフトウェアおよびハードウェアの検討と実装も含まれる。

HPEは、異なるノードアーキテクチャに基づくコア処理技術について、主流のCPU/GPUベンダであるAMD、Intel、NVIDIAと引き続き協力している。これらのベンダは独自のローカルスケールアップ相互接続（xGMI、UPI、NVlink）を開発し続けており、PCIeやCXLといった業界標準の接続と比較して、ノード内で直接接続されたチップ内でより高い性能を発揮できるようにしている。昨年、Ultra Accelerator Link（UAL）コンソーシアムが設立され（https://www.ualinkconsortium.org/）、業界オープンでスケールアップ可能なインターコネクトが誕生した。HPE、AMD、Intel、富士通がメンバである。目標は、高速アクセラレータ間の相互運用性を高めることであり、これは富岳NEXTにとって重要な価値があると考えている。HPEは、理化学研究所と共に、富岳NEXTの実現技術としてUALを使用する可能性について調査したいと考えている。これには、HPEのUltra Ethernet対応Slingshotシリコン（下記参照）に基づく、ローカルスケールアップUAL接続からスケールアウトイーサネット接続へのブリッジ機能が含まれる可能性がある。

HPEは、新たな代替プロセッサ技術を注視しているが、これらの主流ベンダの投資レベルを考えると、ウェーハスケール統合や新しいダイナミック・データフロー・プロセッシング・エレメント（CerebrasやNextSiliconがその例）などの急進的なアプローチに競争力があると考える。

HPEは、HPE Slingshot Ethernetの発展に多大な投資を行っており、高度なプロセッサノードを結合し、バラ

ンスの取れたシステムアーキテクチャを構築する機能を提供している。これには、プロセッサとのチップレットレベルの統合の可能性や、ノード間通信にシリコンフォトニクスを使用する可能性も含まれる。HPEは、HPE Slingshotテクノロジのメリットを享受できる日本のプロセッサ開発企業との協力に前向きである。HPEは、Ultra Ethernet Consortium - UEC（https://ultraethernet.org/）の一員として、Slingshotによるエクサスケールの成功を通じて得られた知識を活用し、高性能イーサネットの共通オープンスタンダードの確立に取り組んでいる。UECの取り組みは、プロトコル、フィジカル、リンク、トランスポート、ソフトウェア、ストレージ、コンプライアンスといった異なる注力分野ごとにワーキンググループに分かれている。UECのウェブサイトには、それぞれの詳細が掲載されている。HPEは、理化学研究所や、理化学研究所が協力している他のテクノロジ・パートナが、UECへの参加を検討することを強く推奨する。UECでの取り組みは、HPCとAIのワークロードの需要に焦点を当てており、理化学研究所がターゲットとするアプリケーションの要件と合致する。この取り組みにより、富岳NEXT プログラムの一環としてUECを使用するリスクが軽減され、コンピュートノードの一部を構成するネットワークアダプタとスイッチングインフラ等のコンポーネント/技術を提供する日本企業との潜在的な協力の機会が得られる。

HPEは、富岳NEXTアーキテクチャ確立に貢献するためには、以下の分野でのコラボレーションに集中することが最も生産的であると考える。

- 富岳 NEXT の現在の方向性である SOC（System-On-Chip）アプローチによるプロセッシング技術と HPE Slingshot Ethernet インターコネクト統合の最適な展開を確保するためのコラボレーション。
- システムソフトウェアスタック、複雑なワークフローのためのサービスと API、HPC と AI のための生産的なプログラミング環境、システム管理。
- DAOS 高性能ストレージ・ソフトウェアに基づく、新たなワークロード・ニーズに対応する高性能ストレージ。
- HPE が新世代の直接液冷インフラを推進するシステムレベルのパッケージングにより、新たなハイパワープロセッシングエレメントのニーズに対応

富岳NEXT処理ノードの現在の目標は、20TB/秒のローカルメモリ帯域幅であるため、システムのバランスを保つためには、インターコネクト帯域幅の目標は5%の範囲、つまり1TB/秒が必要と考える。この先10年（2025～2030年）の業界標準イーサネットは1600Gb/秒に達するため、処理ノードごとに搭載する緊密に結合された複数のSlingshot Ethernetチャネルで目標を達成できるだろう。ネットワーク・エンドポイントは、プロセッシング・ノードのオンチップネットワーク上に分散配置される。（今日、マルチGPUノード上にマルチNICを分散配置するのと同様）この密度を満たすには、プロセッシングやインターコネクトスイッチ・デバイスと一緒にパッケージされた集積シリコンフォトニクスが必要になる可能性がある。

10年後の時点で、どのリンク技術が最もコストパフォーマンスに優れているかは不透明であり、利用可能な最速技術（例えば1600Gb/秒）が最も費用対効果が高いとは限らない。Slingshot Ethernet設計は、ノードごとに複数のエンドポイントを持ち、NICごとに複数のリンクを持つことができる高度に並列されたものである。これは、各ノードで実行されるアプリケーションの並行性の高さを反映したもので、多くのプロセスがネットワーク・トラフィックを生成する。このようなネットワーク設計の柔軟性は、必要なレベルのグローバル帯域幅をコスト効率よく提供するために活用することができる。

HPEは、ベースとなるプロセッサと相互接続技術とともに、次世代システムソフトウェアの要件を調査している。

HPEは、HPCM（HPE Performance Cluster Manager）とCSM（Cray System Management）によって、従来のHPCシステム管理環境と、現在のHPE製品が使用しているクラウド的なオペレーティング環境の組み

合わせを構築しようとしている。HPEは、HPCシステム管理のための新しいオープンソースプロジェクトOpenCHAMI（https://openchami.org）の創設メンバであり、日本のテクノロジ・パートナとの協業の機会を広げている。

プログラミング環境に関して、HPEは従来のアットスケールHPCアプリケーションを推進するためのCray Programming Environment（CPE）を進化させ続けている。AIワークロードへの注目が高まる中、多くのシリコンプロセッシングテクノロジは、HPCアプリケーションを実現する鍵となる従来の64ビット浮動小数点演算のニーズよりも、低精度演算の実装を優先している。HPEは、シリコンベンダが64ビット演算をエミュレートまたは代替するために低精度演算を使用するソフトウェア投資と提携することで、浮動小数点演算のパフォーマンス向上に向けた取り組みを継続する。また、CPEに新たな機能を追加し、ユーザが最新のシリコンプロセッサを最大限に活用できるようにすると同時に、HPCアプリケーションに必要な精度を維持し、正しい結果を保証することを目指す。

シリコンベンダによるAI分野の急速な進展に直面し、HPEはサポートOSの選択も見直している。RedHat Enterprise Linux（RHEL）は現在、AI向け商用OSのリーダであり、Rockyのバリエーションにも強い関心が寄せられている。UbuntuもAI分野で人気が高まっている。HPEは今後、これらの標準ディストリビューションをサポートし、カスタムカーネルの開発は行わない予定である。HPEは、最適なオペレーティング・システム・パフォーマンスが達成されるように、標準カーネルのチューニングについて顧客と協力しており、理化学研究所と協力して富岳NEXTで最も効率的なパフォーマンスが達成されるようにする用意がある。さらにHPEは、HPCとAIを組み合わせた分野で一般的な複雑なワークフローをサポートするためのソフトウェアであるUser Services Software(USS)を導入し、これらのワークフローをサポートする追加機能とインフラストラクチャを提供する。

2.8 アーキテクチャ調査研究サブグループ 7（Arm）

2.8.1 調査研究の概要

昨年に続き、Arm社の持つ半導体の知的財産（IP）である汎用プロセッサアーキテクチャ（例：Armv9）の技術に関する調査を継続した。特に、Scalable Matrix Extensions（SME）に関するArm社の動向を調査した。

SMEは、機械学習とハイパフォーマンス・コンピューティング・アプリケーションのさらなる強化を目指し、2次元行列タイル、2つのベクトルの外積をタイルに蓄積する命令、タイル幅に一致するベクトル長でSVE2の実行を許可するストリーミング機能、を提供するアーキテクチャ拡張である。また、第二世代のSME2では、当初利用可能であった外積と行列・行列積に加え、SMEから恩恵を得られるアプリケーションを増やすために、実行可能な命令を拡張している。ArmアーキテクチャやSMEに関する詳細情報、およびそれらに関するArm社の動向の多くは現時点では非公開であることから、本報告書では割愛する。

2.9 まとめと今後の課題

アーキテクチャ調査研究グループでは、グループ構成員およびA0～A7のサブグループにより、技術動向調査やアーキテクチャ検討を行い、目指すべきアーキテクチャの基本構成とシステム性能をまとめた。

- サブグループ A0：半導体技術・パッケージング技術・アーキテクチャの技術動向の継続調査、次世代計算基盤への要件や評価指標の整理と更新、他のサブグループ調査研究の方向付けと推進、内容のレビューやフィードバック。アーキテクチャ候補のさらなる絞込みと方向性の決定。 目指すべきアーキテクチャの基本構成とシステム性能の検討
- サブグループ A1,6,7：要素技術や技術動向の調査
- サブグループ A2～A5：提案アーキテクチャ候補の詳細化と評価。理研の選定したベンチマークによる性能推定

アーキテクチャ調査研究グループでまとめた目指すべきアーキテクチャの次世代計算基盤を具現化し実現するためには、今後は以下を実施していくことが求められる。

- 開発ロードマップの策定
- 開発体制の構築
- 具体的な技術やアーキテクチャ構成の検討。システム性能、アプリケーション性能、コスト等の推定（基本設計）
- 詳細設計、および製造に向けた準備

3. システムソフトウェア・ライブラリ調査研究グループ

3.1　調査研究の概要および方針

3.1.1　　概要

次世代計算基盤のシステムソフトウェア開発では、これまでのフラグシップシステムや第二階層システムのソフトウェアの開発工数や利用状況、そして既に構築されているエコシステムとの親和性も踏まえ、ソフトウェア資産として何をベースに、国内でどこまでを開発すべきか、もしくは国際協調が必要なソフトウェアへ優先度をつけて明らかしつつ、今後のロードマップを策定することが必要である。また、次世代計算基盤の産業界を含む幅広い応用分野での活用を促すためにも、従来のシステムソフトウェアだけでなくデータ利活用の促進、機械学習技術と第一原理シミュレーション、さらには大規模リアルタイムデータ処理の高度な融合、従来の共用HPCシステムとは次元の異なる高セキュリティの担保、などを主要検討項目と見据えた調査研究が重要となる。システムソフトウェア・ライブラリ調査研究グループ（以下、システムソフトG）では、2024年度も引き続き、今後の日本の次世代計算基盤として行うべきソフトウェア開発について調査研究を行い、その調査結果を各サブグループの調査研究分野でまとめるとともに、GPU向けプログラミングフレームワーク及びライブラリの評価を行った。

3.1.2　　方針

次世代計算基盤に係る調査研究は、候補アーキテクチャを取り巻くソフトウェアエコシステムの状況やその後の商業展開を見据えた調査が不可欠であり、網羅的なシステムソフトウェアの調査研究が必要である。このためシステムソフト・ライブラリ調査研究グループでは、コンパイラ・プログラミング、スケジューラ・ランタイム 、通信ライブラリ、 I/O・ストレージ・ファイルシステム、数値ライブラリ、AIフレームワーク、OS・仮想化・クラウド連携、自動チューニング、セキュリティに関する技術調査に加えHPCにおけるシステムソフトウェアの利用環境調査を行うことのより、利用実態から優先順位をつけそれに基づき、国内で開発・移植・チューニングすべきものと国際協調によりソフトウェア開発を行うべきかの決定を行う。

より具体的には、以下の調査研究計画（図 3.1.2.1）に従い、主に以下の3つの調査研究を行ってきた。

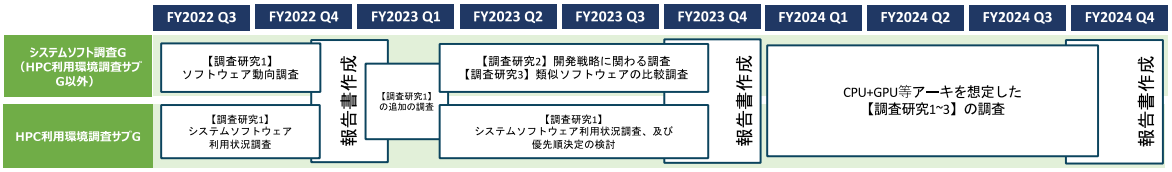


図 3.1.2.1　システムソフト G 調査研究計画

- (2022 年度計画)【調査研究 1】システムソフトウェアの動向調査と利用状況調査
- (2023 年度計画)【調査研究 2】システムソフトウェア開発戦略に関わる調査
- (2023 年度計画)【調査研究 3】類似ソフトウェアの比較調査
- (2024 年度計画) CPU+GPU 等アーキを想定した【調査研究 1~3】の調査

2024年度には、CPU+GPU等アーキを想定した【調査研究1~3】の調査を行った。2023年度の報告書で説明した通り、【調査項目1】では、 移植性、生産性を加味して現在のシステムソフトウェアの動向と将来トレンドの調査し、また大規模システムにおけるシステムソフトウェアの利用状況の調査を行う。【調査項目2】では、システムソフトウェア開発戦略に関わる調査、具体的には国内で独自開発もしくは海外連携により開発をするべきか、プロプラエ

タリソフトウェアとして開発もしくはオープンソースソフトウェアとして開発をすべきか、新規開発もしくは既存のソフトウェアエコシステムを活用（移植、チューニング）した開発をすべきか、を決定するための調査を行う。【調査項目3】では、類似ソフトウェアの比較調査を行う。例えば、MPI実装のOpenMPI、MVAPICHはAPIが標準化されているため、どちらのMPI実装を採用したとしてもアプリケーションユーザへの影響は比較的少ない。また、一方が他方の代替ソフトウェアとしての役割も果たすため両方のソフトウェアを導入することでソフトウェアの欠陥があった場合には、ユーザの利便性を損なうことなく他方のシフトウェアへ切り替えが行えるためリスク軽減に役立つ。一方で、標準化されていないシステムソフトウェアを新規に導入する場合には、アプリケーションユーザは移植のためのコーディングが必要となる。例えば、Intel社のOneMKLや富士通社のSSL-IIは共に数値計算ライブラリであるが、数値計算ライブラリでは標準化が十分でないものもありその場合、移植の作業が必要となるため、富岳で採用されたArmベースのCPU以外のアーキテクチャを採用する場合には、その影響をある程度見積もる必要がある。

3.2 各システムソフトウェア分野における調査報告

3.2.1 コンパイラ・プログラミングモデル

コンパイラ・プログラミングモデル・サブワーキンググループでは、LLVM/Flangの開発状況、LLVMにおけるHPC向けの最適化の状況、アクセラレータ向けソフトウェアスタック、アクセラレータ向けプログラミングモデル、CPUのみのシステムからアクセラレータを含むシステムに移行した事例、日本のアプリケーションの事例について調査した。

なお、本調査対象は現在でも開発が進行の項目を含み、2024年3Qまでの状況であることに注意されたい。

3.2.1.1 調査目的

LLVM/Flangに関する調査

2022 年度の報告にもあるように日本におけるHPC アプリケーション開発ではFortran がいまだに重要な比重を占めるため、これが将来どのようにサポートされるかを知ることが重要である。最新の状況を常に把握することが重要であるため、昨年度に引き続き「LLVM/Flangに関する調査」を行った。

LLVMにおけるHPC向け最適化

これまでプロプライエタリのコンパイラを提供してきた多くのベンダがオープンソース (OSS) のベースのコンパイラに移行しつつある。本年度は、LLVMにおけるHPC向けの最適化機能について状況を調査した。

アクセラレータ向けソフトウェアスタック

アクセラレータは次世代機において重要な役割を果たすと考えられる。デバッガ等のツールも含めたソフトウェアスタックについて状況を調査した。

アクセラレータ向けプログラミングモデル

新たにアクセラレータを利用するユーザや、既存のコードを利用するユーザ、そのどちらにとっても、どのようなプログラミングモデルが利用可能なのかは重要である。アクセラレータをプログラミングする際のプログラミングモデルについて調査した。

システム移行に対する事例

新たなシステムに向けたアプリケーションの移植について、事例を知るために、最近システムの更新をした国内外のスーパーコンピュータセンターの移行事例について調査した。

アプリケーションのベンチマーク

現時点で、日本のアプリケーションがどの程度アクセラレータに最適化されているかしるために、調査およびベンチマークを行った。

3.2.1.2 調査内容

LLVM/Flangに関する調査

LLVMプロジェクトでは、PGIコンパイラを基にしたOSSのFortranフロントエンドであるFlang (現行のFlangと区別するためにClassic-Flangとも呼ばれる) を開発していたが、商用ベースのコンパイラに基づいてオープンソースのコンパイラを開発することは制約が大きく、2018年に新たに現行のFortranフロントエンドの開発を開始した（LLVM-Flang、または単にFlangと呼ばれる）。昨年度の調査の時点では、LLVM-Flangは現時点でFortran 95までの規格にほぼ対応するものの、最適化機能は開発途上であり、SPECCPU2017ベンチマークの性能は先行するオープンソースコンパイラであるGFortranと比較して1.53倍遅かった。本年度は、主に富士通がLLVM Developers' Meeting、Linaro Connect、LLVMコミュニティの各種オンラインミーティングなどに参加して収集した情報を中心に本年度の夏の時点での状況を報告する。

LLVMにおけるHPC向け最適化

LLVMにおける新たなベクトル化モデルである Vectorization Plan (VPlan)について調査した。機械学習等を導入した最適化についても調査した。また、ClangおよびFlangでテストコードを用いてベクトル化の可否の状況についてベンチマークを行った。

アクセラレータ向けソフトウェアスタック

現時点で比較的幅広く用いられているNVIDIA、AMD、およびIntel社のアクセラレータに向けたソフトウェアスタックについて、機能のカバー状況およびホストCPUにArm64を利用する場合のサポート状況について調査した。なお、各社のソフトウェアスタックの概要は昨年度の報告書を参照されたい。

アクセラレータ向けプログラミングモデル

アクセラレータ向けプログラミング言語およびモデルについて、以下の観点サポート状況を調査した。

GPU向けネイティブ言語：CUDA、HIP、DPC++

- ディレクティブ言語：OpenACC、OpenMP
- 抽象化言語：OpenCL、Kokkos、RAJA、AdaptiveCpp、alpaka、gpu.cpp

また、行列計算によって性能評価を行った。

システム移行に対する事例

米オークリッジ国立研究所のSummitからFrontierへの移行事例、日JCAHPCAのOakforest PACS からMiyabiへの移行の事例について調査した。SummitはNVIDIA社のGPUからなるクラスタ、FrontierはAMD社のCPUとGPUからなるクラスタである。Oakforest PACSは、Intel社のCPUからなるクラスタ、MiyabiはNVIDIA社のCPUとGPUからなるクラスタである。

アプリケーションのベンチマーク

富岳等で大規模に実行された経験があり、かつ、GPU化されているアプリケーションについて、既存のGPU環境でのベンチマークと富岳との性能の比較を行った。

3.2.1.3 調査結果

LLVM/Flangに関する調査

表 3.2.1.1 LLVM/Flang の Fortran 規格および OpenMP 規格への対応状況

Fortran 規格	対応状況（2024 年 7 月）
Fortran 95	実用上完了
Fortran 2003	ほぼ完了
Fortran 2008–2018	一部の機能をサポート フロントエンドでの構文解析が可能
主要な未実装機能	**対応状況（同上）**
Parameterized derived type	未実装
Coarray	通信ライブラリや中間表現について議論中
Assumed-rank	2024 年 5 月より開始
Flang OpenMP	**対応状況（同上）**
OpenMP 1.1	実用上完了
OpenMP 2.5	完了に向け実装中

OpenMP 3.0 以降	一部の機能をサポート OpenMP4.5, 5.0 の一部のフロントエンド構文解析が可能 SPECspeed2017 が動作

表 3.2.1.1にFortran規格およびOpenMP規格への対応状況を示す[321-1,321-2,321-3,321-4,321-5]。

Coarray機能の実装については完了まで時間を要する可能性が高い。ただし、われわれが昨年度調査に用いたスーパーコンピュータ富岳におけるコンパイラ・ライブラリ利用状況調査に用いたデータからCoarrayの利用率を調査したところ、Coarrayの利用率は非常に低いことがわかっている。OpenMPについては、1.1は実用上に用いられている機能について実装が完了、2.5は完了に向けて実装中である。

現行のLLVM/Flangのコマンドは昨年度までflang-newであったが、flangとなったことから、実装が進展しておりFlangのコミュニティ内でもLLVM-Flangが実用に近いとの判断と考えられる[321-6]。Arm社がLLVMベースで開発するArm Compilerについても、これまではClassic-Flangをベースにしてきたが、2024年にLLVM-Flangベースのコンパイラがexperimentalバージョンとして発表された。

LLVMのサブプロジェクトである Multi-Level Intermediate Representation (MLIR) について述べる。MLIRは、CPU、GPU、テンソルコア、FPGA、などのさまざまな計算プラットフォームを利用するためのソフトウェアフレームワークである[321-8]。"2024 Euro LLVM Developers' Meetingにおいても、MLIR関連のセッションが4割を占めるなど近年では注目を集めている。MLIRには、そのエコシステムを拡張するための Dialects と呼ばれるメカニズムがある。Dialects により、新たな操作、属性、および型の定義が可能となる[321-9]。LLVMのレポジトリに取り込み済のDialects としては以下のような例がある：

- Fortran(FIR, HLFIR)、
- OpenMP, OpenACC, MPI
- tensor, vector, linear algebra, affine
- Arm Neon, Arm SVE, Arm SME
- GPU, NVIDIA GPU, AMD GPU, Xe GPU

また、提案段階の Dialects としては以下のような例がある：

- C/C++ (ClangIR)
- SYCL

ClangIR [321-10] は、C/C++に対する Dialect であり、Clang 抽象構文木とLLVM-IR の中間的な位置づけにある。2024年の2月に、ClangIRをLLVMレポジトリにアップストリームすることが承認された。C/C++固有のイデオムでのプログラム診断や性能最適化の改善が期待される。

上述のようにMLIRにおいては、Armやアクセラレータ向けのDialectsが導入済であること、HPC・AI 向けの最適化に ClangIR が活用できる可能性があることから、今後も動向に注視していく必要がある。

LLVMにおけるHPC向け最適化

Vectorization Plan (VPlan) は、LLVMにおけるベクトル化候補を要に記述するモデルである[3.2.1-11]。近年、誘導変数とリダクション変数の区別等のモデルの表現能力の拡張、SIMDループ以外のpreheaderやexit block等もモデル化対象とするなどのモデルの表現能力の拡張などが見られ、より表現能力を増している。将来の課題としては、外側ループSIMDや余りループの扱いの改善等がある[321-12, 321-13]。

Armアーキテクチャに関しては、SVE2を活用した最適化として、ヒストグラム計算のベクトル化が、提案されている[321-14]。

さらに、LLVMにおける自動ベクトル化に関するオンライン会議 "Vector Improvements Call" が2024年5

月から毎月開催されており、ループベクトル化と Superword Level Parallelism (SLP) ベクトル化を対象とした議論が行われている。Arm, SiFive, Intel, Apple, 富士通などの多様なベンダが参加し、開発者の協調によるベクトル化機能のさらなる改善が期待される[321-15]。

近年の動向としては、機械学習 (ML) をLLVMに導入する動きがある。GoogleがLLVMのために開発した Machine Learning Guided Compiler Optimizations (MLGO) Framework [321-16] は、最適化のために機械学習を活用する枠組みで、MLモデルの学習用のフレームワークが公開され、このフレームワークで学習したMLモデルをLLVM本体に取り込む実装が反映される。関数インライン展開、レジスタ割り付けなどが可能である。大規模言語モデル (LLM) の応用研究としては、最適化パスの選択 [321-16] 、外部関数の効率的な利用 [3.2.1-17]、コンパイラの障害を効率的に探すためのテストの作成 [3.2.1-18]、ソースコードの言語変換 [3.2.1-19]などが行われている。また学習のためにLLVM-IRの大規模なデータセットも整備されている[3.2.1-20, 3.2.1-21]。これらの動向についても引き続き注目していく必要がある。

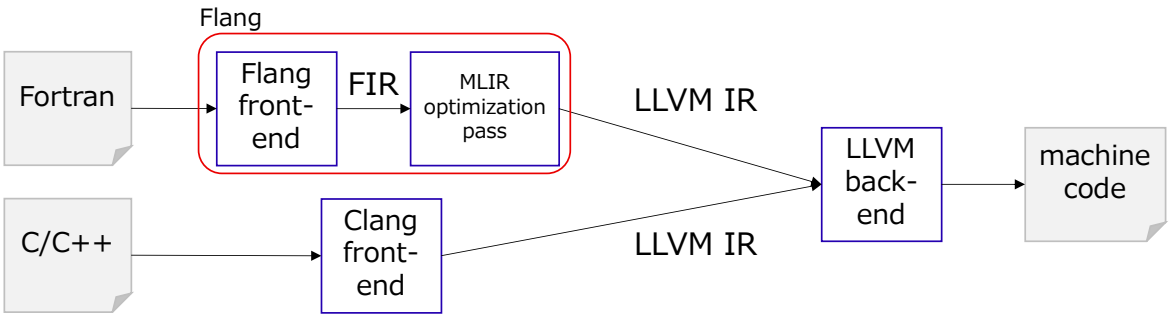


図 3.2.1.1 LLVM の最適化パス

図 3.2.1.1に、LLVM/Flangにおける最適化パスを示す。LLVM IRでは表現できない Fortran 固有の情報を用いて、一時配列の除去のようなバックエンドでは難しい最適化や、組み込み関数の単純化のような Fortran 固有の最適化を行う。次いで、LLVMバックエンドにおける最適化を行う。LLVMバックエンドにおける最適化はC/C++と共通である。

つづいて、Flangのベクトル化能力について、富士通が検証した結果を述べる。LLVMでは以下のような流れでループのベクトル化を行っている：

1. ベクトル化の可否の判定
2. ベクトル化方法の候補を複数作成
3. コストモデルにより各候補のコストを計算
4. もっともコストが少ない候補を選択

本報告では、1. のベクトル化の可否の判定に焦点をあてた。なお、ベクトル化が可能であっても、最終的に 4. によってベクトル化が却下される可能性もあることに留意されたい。

ベンチマークとしては "Test Suite for Vectorizing Compilers (TSVC) [321-22]" を用いた。TSVCは、元はFORTRAN 77で記述されたテストセットであったが、Malekiら [321-23] によってCに移植され、CとFortranの両版が存在する。CおよびFortranで実装されている128種類のループをコンパイルし、これらの結果を比較した。

評価に使用したLLVMのバージョン、最適化オプションは表 3.2.1.2である。ターゲットアーキテクチャとしてはA64FXを用いた。開発の進展によるベクトル化可能ループの変化を調査するために、CおよびFortranのそれぞれについて複数のバージョンのLLVMを用いた。

表 3.2.1.2 評価に用いたコンパイラと最適化オプション
（※LLVM 19 で追加された SIMD 化促進のための機能を有効にするオプション）

コンパイラ	バージョン	ターゲット CPU	最適化オプション	選定理由
Flang/LLVM	19.1.0-rc2	A64FX	-Ofast -mcpu=a64fx -flang-experimental-integer-overflow※1	**評価対象**
Flang/LLVM	18.1.8	A64FX	-Ofast -mcpu=a64fx	開発半年間(v18→v19)の改善度合を評価
Clang/LLVM	19.1.0-rc2	A64FX	-Ofast -mcpu=a64fx	バックエンドの最適化能力を評価
Clang/LLVM	18.1.8	A64FX	-Ofast -mcpu=a64fx	開発半年間(v18→v19)の改善度合を評価

表 3.2.1.3 Clang および Flang におけるベクトル化の可否

ベクトル化の可否		ループ数 (LLVM18)	ループ数 (LLVM19)
Clang	Flang		
○	○	73	77
○	×	11	10
×	○	2	1
×	×	42	40

表 3.2.1.3に各言語、各LLVMバージョンに関するベクトル化の可否に関するループ数を示す。Clangにおいてベクトル化可能なループ数は、LLVM18では84本、LLVM19では87本であり、Flangにおいてベクトル化可能なループ数は、LLVM18では75本、LLVM19では78本であり、いずれも版を重ねることで性能が向上している。Clangでベクトル化可能であるもののFlangでは不可能であったループはLLVM18では11本、LLVM19では10本であり、比較的差が大きい。ただし、FORTRAN 77からCに移植する過程で、Cプログラムとして自然な形になるように変更されたループが8本あり、類似の変更を行うことでFlangでも8本はベクトル化が可能になると見られることから、ClangとFlangのベクトル化能力は同等になりつつあると考えられる。Clangではベクトル化不可、Flangではベクトル化可のループがLLVM18で2本、LLVM19で1本あった。残る1本はmemcpyの置き換えを行うか否かの判断の差で、性能上の大きな問題はない。また、いずれもベクトル化できなかったループがLLVM18で42本あったのにたいして、LLVM19では40本と、減少している。

Clangではベクトル化可能であったもののFlangでは不可能であった10のループについて、表 3.2.1.4に示す。

表 3.2.1.4 Flang でベクトル化できなかったコードとその原因

ループ	原因分類	原因概要	是正が必要
s115	オーバーフロー	整数演算のオーバーフローを考慮してしまう	
s233	添字解析	特定パターンのストライドアクセスに未対応	○
s243	オーバーフロー	整数演算のオーバーフローを考慮してしまう	
s257	添字解析	特定パターンのストライドアクセスに未対応	○
s2101	添字解析	特定パターンのストライドアクセスに未対応	○
s314	リダクション	リダクション変数の更新直後にストアされる	○
s316	リダクション	リダクション変数の更新直後にストアされる	○
s3111	リダクション	リダクション変数の更新直後にストアされる	○
s3113	リダクション	リダクション変数の更新直後にストアされる	○
s471	コードの違い	Fortran版だけ内部で呼ぶ関数が別ファイルに定義	○

これらの結果については、すでに富士通から、LLVM/Flangの開発コミュニティと共有されており、原因分類が「オーバーフロー」となっているループ115, 243、およびリダクションに問題のあるループ314については、すでに議論を行っている。ループ316, 3111, 3113 についても課題として共有されている。図 3.2.1.2で赤字で示したループ 234, 2101, 314 については、CおよびFORTRAN 77で本質的なコードの差がないにもかかわらず、Flangのみベクトル化不可であったループである。ループを以下の図 3.2.1.2に示す：

s243

```
do i = 1,n-1
  a(i) = b(i) + c(i) * d(i)
  b(i) = a(i) + d(i) * e(i)
  a(i) = b(i) + a(i+1) * d(i)
end do
```

s2101

```
integer ld, n
real aa(ld,n), bb(ld,n), cc(ld,n)
 …
do i = 1,n
  aa(i,i) = aa(i,i) + bb(i,i) * cc(i,i)
end do
```

s314

```
x = a(1)
do i = 2,n
  if(a(i) .gt. x) x = a(i)
end do
dummy(a,b,…,cc,x)
```

図 3.2.1.2 ベクトル化不可のコード例

ループ243は、a(i) と a(i+1) がメモリ領域上重ならないことを判定できず、ベクトル化適用不可と判断されている。FortranとCの添字表現が異なることから、Flangではアドレス計算において整数オーバーフローの可能性を排除できないことから計算式の簡略が進行せず、解析が不十分な結果になる。言語仕様に起因したフロントエンド出力に課題があるが、言語規格を根拠として特定の計算ではオーバーフローしないことを想定できるため、LLVM IRの内部表現であるnswフラグを適用して、Flangの出力IRを改善することで、アドレス計算の簡略化を促進することが可能である。

ループ2101は、配列サイズが可変の場合に、添字 (i, i) のストライド幅 (ld+1) を認識できず、ベクトル化が

不可と判断される例である。関数内では配列のサイズが不変であるため、ベクトル化が可能である。これは、ストライド幅の計算処理において、配列宣言時のサイズが即値のケースしか受け入れていないためである。関数内においてはストライド幅が不変であっても、ストライド幅の計算を断念されてしまう。解決のためには、ベクトル化のストライド幅の計算処理において、即値ではない場合を受け入れるように拡張する必要がある。これはClangでも起こりえる問題であるが、Cの場合にはFortranのように多次元配列の形状に引数変数を用いることがないため、Clangでは本ループはベクトル化が可能であった。

ループ314は、ループ内のリダクション変数が参照渡しの場合、ループ内に条件分岐があるなどの特定のパターンではリダクションのベクトル化が適用されない例である。これは、LLVMのリダクションのパターン認識処理で、リダクション変数が参照渡しされたケースはベクトル化の対象外として判定しているためである。if文において更新されたリダクション変数を一時レジスタに格納し、後にストアすることで、ベクトル化が可能である。ループ2101と同様にCの場合も同様の問題が起こりえるが、CではFortranのように関数の引数のデフォルトが参照渡しではないため、Clangでは本ループはベクトル化が可能であった。

前述のように、これらの課題はすでに富士通からLLVM/Flangの開発コミュニティと共有されており、いくつかの問題点についてはすでに議論が開始されており、Flangにおけるベクトル化率の向上が期待される。

アクセラレータ向けソフトウェアスタック

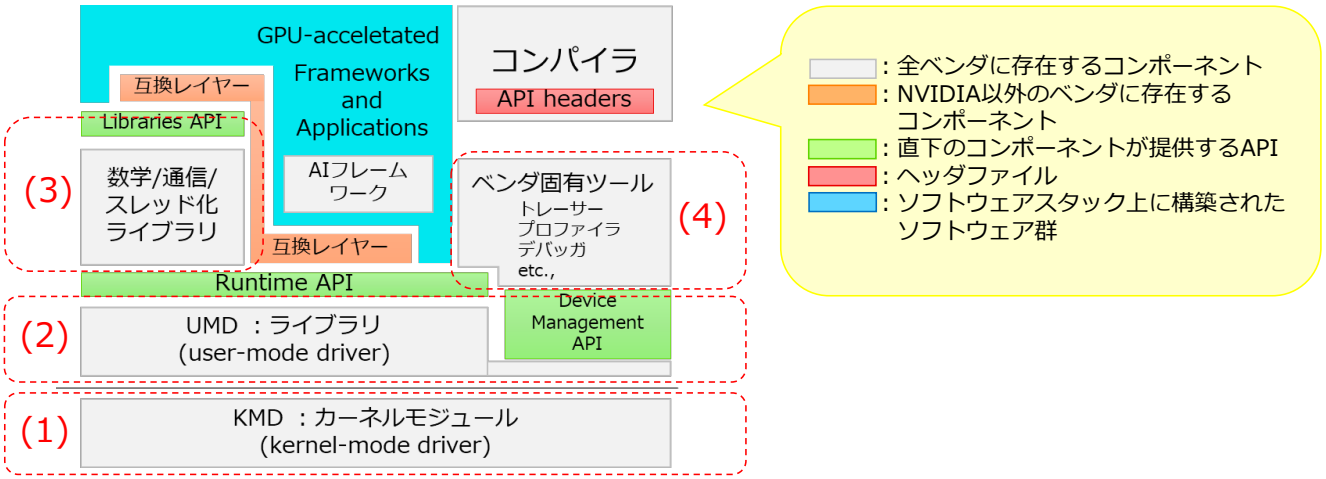


図 3.2.1.3 NVIDIA、AMD、Intel の GPU 向けソフトウェアスタック

図 3.2.1.3にGPU向けソフトスタックの標準的な構成を示す。以降では、これらのうちNVIDIA、AMD、およびIntelのすべてのベンダに向けて提供されている

(1) カーネルモジュール (KMD)

(2) ユーザモードドライバライブラリ (UMD)

(3) 数学、通信、スレッド化ライブラリ

(4) ベンダ固有ツール（トレーサ、プロファイラ、デバッガなど）

についてまとめる。

表 3.2.1.5 カーネルモジュールの対応状況

	NVIDIA	AMD	Intel	
			現行	開発中
ドライバ名	NVML	amdgpu	i915	Xe
OSS化	プロプラ版とOSS版の並行開発	Linux upstreamでOSS化	Linux upstreamでOSS化	Linux upstreamでOSS化
Arm64サポート	正式サポート	公式サポート無し(Arm64向けパッチ有り)	サポート無し	サポート無し(複数アーキに対応できる設計)

表 3.2.1.5 に、(1) カーネルモジュール (KMD) の対応状況を示す。Arm64サポートはホストCPUとしてArm64CPUが利用できるかどうかを示す。NVIDIAのみがホストCPUとしてArm64をサポートし、AMDおよびIntelはx86のみに対応している。ただし、AMDにおいてはドライバの一部をArm64でビルドするためのパッチが提供されている。また、Intelが現在開発中のXeドライバは複数のホストCPUを意識した設計となっており、将来的にArm64がサポートされる可能性がある。

表 3.2.1.6 ユーザドライバライブラリの対応状況

	NVIDIA	AMD	Intel
Driver API	CUDA driver API	ROCT-Thunk-Interface	Level Zero API
Runtime API	CUDA runtime API	ROCm System Runtime API	
GPU管理API	NVML	ROCm SMI Library	
OSS化	全てプロプライエタリ	全てOSS	API部分のみOSS, 他はプロプライエタリ
Arm64対応	公式に対応	公式には未対応(非公式のArm64版有り)	未対応

表 3.2.1.6に(2) ユーザモードドライバライブラリ (UMD)の対応状況を示す。UMDにおけるランタイムライブラリは、ユーザ空間とカーネルドライバの間のやりとりを担うとともに、GPGPUアプリケーションに必須のAPIを提供する。Driver API は、カーネルドライバとやり取りするためのAPI。原理的にはDriver APIだけでGPGPUアプリを作成することも可能である。Runtime API は GPGPUアプリの開発用APIであり、Driver APIの上に実装されている。利用が簡便である一方で、細かな制御が難しい。GPU管理用APIはGPUの状態取得や各種設定の変更に用いられる。ホストCPUとしてのArm64の対応状況としては、NVIDIAは公式に対応し、AMDは非公式なArm64版が存在する。

表 3.2.1.7 数学、通信、スレッド化ライブラリの対応状況

NVIDIA (Arm64対応)	AMD (Arm64未対応)	Intel (Arm64未対応)
cuBLAS	rocBLAS	oneMKL
cuFFT	rocFFT	
cuRAND	rocRAND	
cuSOLVER	rocSOLVER	
cuSPARSE	rocSPARSE	
AMG-X (※Arm64対応不明)	rocALUTION	
CUB / Thrust	rocPRIM / rocThrust	oneTBB
cuTENSOR	composable kernel	oneDNN, oneDAL
cuDNN	MIOpen	
NCCL	RCCL	oneCCL

表 3.2.1.7 に (3) 数学、通信、スレッド化ライブラリの対応状況を示す。すべてのベンダ向けでほぼ同等機能を持つライブラリが提供されている。ホストCPUとしてのArm64の対応状況としては、NVIDIAは公式に対応し、AMDおよびIntelは未対応である。

近年、需要が高まっている量子コンピュータ向けのソフトウェアスタックについては、NVIDIAはもっとも充実した環境を提供しており、同GPUに最適化されたシミュレータや複数の原理の異なる量子コンピュータ実機をサポートするQPU-agnostic なプラットフォームであるCUDA-Qを提供している。AMD社は、米オークリッジ国立研究所等と共同でシミュレータの開発をしている。[321-48] Intel社は、量子コンピュータ向けの総合的な開発環境であるIntel Quantum SDKを提供している。将来的には同社の量子コンピュータを利用するためのSDKとして設計されており、現在のところは、シミュレータも含めてGPUには対応していない。[321-49]

表 3.2.1.8 ベンダ固有ツール の対応状況

	NVIDIA (Arm64対応)	AMD (Arm64未対応)	Intel (Arm64未対応)
Profile/Trace API	CUPTI	roctracer, rocprofiler	なし
システムワイドの分析ツール	Nsight systems	rocprof, omnitrace	Intel VTune Profiler
カーネルの性能分析ツール	Nsight compute	rocprof, omniperf	
デバッガ(gdb)	cuda-gdb	ROCgdb	Intel Distribution for GDB
動的解析ツール	NVIDIA Compute Sanitizer	なし	Intel Inspector
診断ツール	DCGM Diagnostic	ROCm Validation Suite	なし

表 3.2.1.8に (4) ベンダ固有ツール の対応状況を示す。NVIDIAがもっとも充実している一方で、AMDおよびIntelはNVIDIAで提供されている一部の機能が提供されていない。AMDでは1つの目的に対して複数のソフトウェアが提供されている一方、Intelは1つのツールが複数の機能をカバーしている。ホストCPUとしてのArm64の対応状況としては、NVIDIAは公式に対応し、AMDおよびIntelは未対応である。

以上から、GPU向けのソフトウェアはNVIDIA向けがもっとも充実しており事実上の標準となっている。AMDおよびIntelは、多くの機能でNVIDIAと同レベルのソフトウェアを提供している。とくにAMDはI/FをNVIDIAとそろえてNVIDIA向けとの互換性を広く意識しているといえる。また、ホストCPUとしてのArm64の対応状況としては、NVIDIAがほぼ対応済である一方、AMDおよびIntelは部分的な対応にとどまっている。ただし、Intel oneAPIの公開が部分的なものにとどまっている一方で、AMDのGPU向けのソフトスタックのROCmは完全にOSSとして公開されているために、現状ではAMD ROCmのほうがArm64対応を進めやすいといえる。

アクセラレータ向けプログラミングモデル

本章では、アクセラレータ向けプログラミング言語およびモデルについて、以下の観点からまとめる：

- ・GPU 向けネイティブ言語：CUDA、HIP、DPC++
- ・ディレクティブ言語：OpenACC、OpenMP
- ・抽象化言語：OpenCL、Kokkos、RAJA、AdaptiveCpp、alpaka、gpu.cpp

また、基本的な性能評価の結果について述べる。

表 3.2.1.9 GPU 向けネイティブ言語

	言語	ホストCPU	GPUサポート状況		
			NVIDIA GPU	AMD GPU	Intel GPU
CUDA	C/C++、Fortran	x86、Arm64	○	×	×
HIP	C/C++	x86	○	○	×
DPC++	C++	x86	○	○	○

GPU向けネイティブ言語について表 3.2.1.9 に概要をまとめた。

CUDAは、NVIDIAが提供する同社のGPU向けのプロプライエタリなプログラミングモデルである。3つのうち唯一、ホストCPUとしてArm64をサポートしている。C/C++の言語拡張としてスタートしたが、現在ではFortranにも正式に対応している。HIPは、AMDが提供する同社のGPU向けのオープンソースのプログラミングモデルである。NVIDIA GPU上でも動作する。C/C++の言語拡張である。Fortranには対応していないが、FortranのCとの相互運用機能によりCで書かれたHIPカーネルルーチンの呼び出しができる。ホストCPUとしてはx86のみをサポートする。また、AMDが提供するHIPIFYというツールを用いることで、CUDA C/C++で記述されたソースコードをほぼ自動的にHIP C/C++に変換することができる。LLVM/Clangの公式レポジトリには、CUDAおよびHIPの対応がある[321-25, 321-26]。Flang CUDAに関する取り組みもはじまっている[321-27]。DPC++は、Intelが提供するクロスプラットフォーム向けのオープンソースのプログラミングモデルである。ホストCPUとしてはx86をサポートする。C++の言語拡張であり、CおよびFortranには対応していない。マルチプラットフォーム向けをうたっており、NVIDIAおよびAMDのGPU上でも同じソースコードが動作する。CUDA C/C++からのコード変換ツールとしてSYCLomaticが提供されている。DPC++コンパイラは、LLVMに基づくが、LLVMの公式レポジトリではなく、Intelの自社レポジトリにて公開されている。

つづいて、ディレクティブ言語について述べる。ディレクティブ言語は、既存の逐次ソースコードに指示文を挿入することで並列化やアクセラレータへのオフロードなどを行う言語の総称である。GPU向けのディレクティブ言語として広く用いられているものは、Open Multi-Processing (OpenMP) と Open accelerators (OpenACC) である。

OpenMPは、もとは共有メモリ・マルチコアのCPUに向けたプログラミングモデルとしてC/C++/Fortranのディレクティブ拡張として開発された。2013年にリリースされた、OpenMPバージョン4.0からアクセラレータのサポートが開始された。

OpenACCは、NVIDIA等のベンダ主導で開発されたCPU-GPUのヘテロジニアスシステムに向けたプログラミング言語で、現在は、非営利組織であるOpenACC Organizationによって管理されている。

表 3.2.1.10 GPU 向けディレクティブ言語（※ AMD と Intel は、OpenACC の直接的なサポートはないが、OpenACC を OpenMP に変換するツールを提供している）

	OpenACCサポート状況		OpenMPサポート状況	
	言語	ホストCPU	言語	ホストCPU
NVIDIA	C/C++、Fortran	x86、Arm64	C/C++、Fortran	x86、Arm64
AMD	※	※	C/C++、Fortran	x86
Intel	※	※	C/C++、Fortran	x86

表 3.2.1.10に、NVIDIA、AMD、およびIntel製のGPUにおけるOpenMPおよびOpenACCのサポート状況を示す。OpenMPは、すべてのGPUでC/C++/Fortranにおける利用が可能である。ただし、ホストCPUとしてArm64が利用可能なのはNVIDIAのGPUとの組み合わせのみである。LLVMにおいては、ClangおよびFlangにおいて、OpenMPによるGPUオフロードをサポートしている。OpenACCは、NVIDA GPU向けにはC/C++/Fortran の利用が、x86 および Arm64 をホストCPUとして可能である。しかし、AMD GPUおよびIntel GPUでの利用可能性は限定的であり、代わりにSource-to-sourceのツールやコンパイラが提供されている。また、近年では、ClangおよびFlangプロジェクトにおいてOpenACCのサポートが開始されている。

つづいて、抽象化言語について述べる。本稿では、抽象化言語として、プラットフォーム非依存の言語拡張を用いて、マルチプラットフォームをサポートする言語・言語拡張を扱う。

表 3.2.1.11 GPU 向け抽象化言語（※：Kokkos と RAJA の SIMD 機能は Experimental）

	言語	CPU抽象化	GPUサポート状況		
			NVIDIA GPU	AMD GPU	Intel GPU
OpenCL	C/C++	Serial	x86/Arm64	x86	x86
Kokkos	C++	Serial/OpenMP/SIMD※	x86/Arm64	x86	x86
RAJA	C++	Serial/OpenMP/SIMD※	x86/Arm64	x86	x86
AdaptiveCpp	C++	Serial/OpenMP	x86/Arm64	x86	x86
alpaka	C++	Serial/OpenMP	x86/Arm64	x86	x86
gpu.cpp	C++	−	x86	x86	x86

抽象化言語の概要を表 3.2.1.11に示す。多くの抽象化言語はC++の拡張であり、CおよびFortranには対応していない。Kokkos、RAJA、AdaptiveCPP、alpalaはGPUのみならず、CPUにおけるSIMDやOpenMPなどの並列化も可能である。これらの言語はマルチプラットフォームを考慮して設計されたため、NVIDIA、AMD、およびIntelのすべてのGPUがサポートされているが、ホストCPUとしてArm64を用いることができるのは現時点ではNVIDIA GPUとの組み合わせのみである。

OpenCLは、Khronos Groupが提供するクロスプラットフォーム向けの並列プログラミング言語である[321-29]。他の抽象化言語と異なり、テンプレートなどC++の機能に依存した言語拡張ではないため、Cでも用いることができる。ただし、GPU向けネイティブ言語に近いプログラミングが必要である。OpenCLのランタイムライブラリに依存する。

Kokkosは、クロスプラットフォームでの性能ポータビリティを目指して、米サンディア国立研究所 (SNL) が中心となって開発したオープンソースのプログラミングモデルである[321-30]。現在では、Linux Foundation 内の High Performance Software Foundation (HPSF) のプロジェクトの1つとなっている[321-31]。C++の言語標準機能である mdspan は Kokkos の多次元配列の抽象化機能であるview構造がもとになっている。また、Kokkos のエコシステムとして、数学カーネル、プロファイラ、デバッガなどが存在する。

RAJAは、Kokkos同様にクロスプラットフォームでの性能ポータビリティを目指して開発された言語である[321-32]。開発主体は、米ローレンスリバモア国立研究所 (LLNL) である。関連プロジェクトとして、メモリ管理の抽象化をになうCHAI [321-33] がある。RAJAは単体ではデバイスメモリの確保機能を持たないため、CHAIを用いるか、GPUネイティブ言語によるメモリ確保が必要である。また、それ以外にもハードウェア固有の定義を設定する必要があり、他の抽象化言語と比較して抽象化度は低いといえる。

SYCLは、Khronos Groupが提供するクロスプラットフォーム向け規格である[321-34]。SYCL規格の実装および拡張した実装として、AdaptiveCPPやIntelの実装したDPC++がある。

AdaptiveCppは、SYCLに基づくクロスプラットフォーム向けのプログラミングモデルである[321-35]。DPC++との差として、AdaptiveCppはホストCPUとしてArmのサポートを明示している[321-36]。開発当初はhipSYCLという名称で、SYCLでHIPをサポートするOSSだった。記述方法は、IntelのDPC++とほぼ同様であり、DPC++特有の拡張以外は、DPC++のソースコードをそのままコンパイルすることが可能である。

alpakaは、C++ヘッダのみで構成される並列カーネル高速化のためのオープンソースな抽象化ライブラリである[321-37]。Kokkosと類似した言語で、NVIDIA GPU向けのホストCPUとしてArm64をサポートしている。RAJAと同様にプログラム内にハードウェア固有の設定を記述する必要がある。

gpu.cppは、WebブラウザからGPUを利用する枠組みであるWebGPUを利用したオープンソースのポータブルなGPU実行ライブラリである[321-38]。公開は2024年7月であり、開発はまだ途上であるが、githubのstar数を急速に伸ばしている。CPU向けの並列化はサポートされていない。GPUはNVIDIA、AMD、およびIntelをサポートするが、ホストCPUとしてはx86のみである。記法はOpenCLと近い。

CUDA、OpenCL、Kokkos、RAJA、AdaptiveCpp、alpakaの性能評価を実施した。評価対象は、倍精度浮動小数点行列の行列積 (DGEMM) とし、行列のサイズは 20,000 x 20,000 とした。ハードウェアや言語に合わせた特殊なチューニングは実施せず、各プログラミングモデルのチュートリアルを参考として基本的な記述方法で記述した。実行に使用したCPUおよびGPUは以下である：

CPU： Intel(R) Xeon(R) Gold 6338 CPU @ 2.00GHz

GPU： NVIDIA A100-PCIE-40GB

共有メモリは利用せず、カーネルモデルのストリーミング処理も行わない。CUDAにおける実装の例を図 3.2.1.4 に示す。行列の1要素の演算を1スレッドに割り当てる実装となっている：

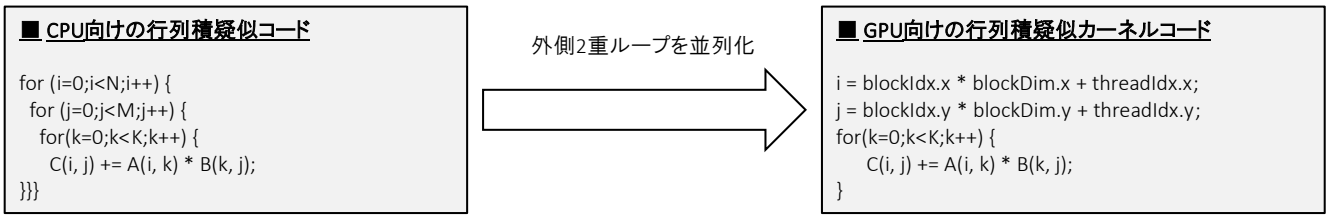


図 3.2.1.4 CUDA による行列演算の GPU 向け並列化

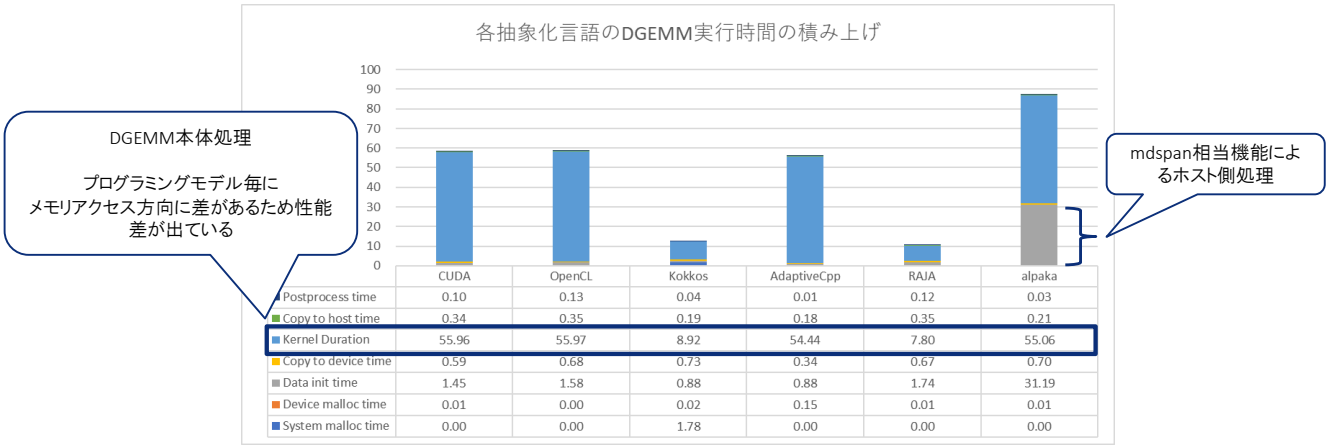


図 3.2.1.5 実験結果（縦軸は秒）

実験の結果を図 3.2.1.5に示す。CUDAと比較して、Kokkos、RAJAでより高い性能、OpenCL、AdaptiveCppは同程度、alpakaのみより低い性能になった。Kokkos、RAJAは、内部的にメモリアクセス方向を変更する最適化を行っており、これにより性能を向上させている。ほかの抽象化言語でも同様の最適化を行えば、同等の性能を出すことは可能であると考えられる。alpakaは、mdspan相当の機能である多次元配列を用いたホスト側の処理が遅く、GPU側の処理事態はCUDAと同等であった。以上より、これらの抽象化言語は、言語やハードウェアごとにチューニングしない状態であれば、概ねCUDAと同程度の性能が出せるポテンシャルがあると考えられる。

Kokkosは、米国のエクサスケールシステムにおける中心的な抽象的言語である。米国におけるエクサスケールシステムへのプログラム移行と言語については3.2.1.5で述べる。米国外においても、多くの研究所が積極的に採用しており、とくに仏国のCEAのエクサスケールプロジェクトCExAでは、Kokkosを採用し、また開発やユーザトレーニングに貢献するとしている[321-47]。

システム移行に対する事例

まず、米オークリッジ国立研究所 (ORNL) において、2022年より稼働したスーパーコンピュータ Frontier について述べる。Frontier のノード構成は以下である：

CPU: AMD Epyc 7713 x 1

GPU: AMD Instinct MI250X GPU x 4

表 3.2.1.12 Frontier における GPU 向けプログラミング言語

Provider	language	OpenACC	OpenMP GPU Offload	備考
AMD	C/C++	x	o	
	Fortran	x	o	
Cray	C/C++	x	o	
	Fortran	**o**	o	OpenACC 2.0 Partial support for OpenACC 2.x/3.x
ORNL	**C/C++**	**o**	-	https://csmd.ornl.gov/project/clacc (OpenACC->OpenMP)
	Fortran	x	-	
GNU	C++	x	?	
	Fortran	x	?	

表 3.2.1.12 にFrontierでGPU向けにサポートされているプログラム言語およびコンパイラのプロバイダを示す。なお、HIPは当然サポートされているため省略した。OpenACCは、AMDのコンパイラではサポートされていないため、これを補うために、CrayがFortran向けOpenACCコンパイラを、ORNLがC/C++コンパイラclaccを提供している。claccはLLVMに基づくオープンソースなコンパイラで、OpenACCを解釈してOpenMPのソースコードに変換し、C/C++向けのOpenMPコンパイラでコンパイルする。

ORNLでは、Frontier以前にはSummitというNVIDIA GPUからなるスーパーコンピュータを運用していた。これらのアプリケーションのAMDへのポーティング支援として、Frontier Center for Accelerated Application Readiness (CAAR) では8つのアプリケーションの支援を行った。 表 3.2.1.13にアプリケーションの概要とポーティング前後のプログラミング言語を示す[321-39]。

表 3.2.1.13 CAAR のアプリケーションと移植前後の言語

Application	Description	(main) GPU Programming	Summit to Frontier
Cholla	Galaxy Evolution	CUDA cuLibrary	HIPify, HIPifly rocLibrary
CoMet	Genome-Wide Epistasis Studies	CUDA	HIP
GESTS	Turbulence Simulations	OpenMP Offloading cuLibrary	OpenMP Offloading rocLibrary
LBPM	Lattice Boltzmann Methods	CUDA	HIP
LSMS	Density Function Theory	CUDA cuLibrary	HIP rocLibrary
NAMD	Molecular Dynamics	CUDA cuLibrary	HIP rocLibrary
NuCCOR	Nuclear Physics	CUDA	HIP
PIConGPU	Particle-In-Cell	様々なハードウェア (CPUs, GPUs, FPGA) やプログラミング (HIP, CUDA, OpenMP host, OpenMP offloading, OpenACC) をサポートするAlpaka バックエンドを利用	

SummitのアプリケーションはCUDAが主流であり、CUDAとHIPは書き換えが容易であり、そのためのツールが多数提供されているため、CUDAからHIPへの書き換えで対応していた。CPUのみのシステムである富岳では異なる対応が必要になると考えられる。

続いて、米エクサスケールコンピューティングプロジェクト (EPC) でのアプリケーション開発およびポーティングについて調査した[321-40]。ECPでは24 specific application projects (62 application codes が含まれる) と6つのsupporting co-design projects を移植した。前述のCAARが、Frontierのみを対象としていたのに対して、ECPのアプリケーションはFrontierに加えてインテルGPUシステムであるAuroraを対象としていることに注意されたい。

表 3.2.1.14 ECP のアプリケーションと supporting co-design projects

Support	Description	Applications
CEED	Unstructured grids	FEM ExaAM, ExaSMR, Ristra, MARBL, SPARC
AMReX	Adaptive Mesh Replacement	AMR ExaWind, Combustion-Pele, MFIX-Exa, WarpX, ExaSky, ExaStar
CODAR	Data reduction and analysis	WDMApp, NWChemEx, CANDLE
CoPA Particles	Spectral methods	EXAALT, ExaAM, WDMApp, ExaSky, WarpX, MFIX-Exa
ExaGraph	Graph traversal	combinatorial logic ExaBiome, ExaWind, NWChemEx, SPARC,EMPIRE,
ExaLearn	ML	AI ExaSky, CANDLE

表 3.2.1.14にsupporting co-design projects とapplication projectsを示す。複数のアプリケーションから共通して利用されるカーネル部分をsupporting co-design projectsとして、FrontierおよびAurora向けに徹底的に最適化し提供することで、トータルの移植コストを下げている。Thomasら[321-40]によれば、Fortranで記述されていたコードの何本かはプロジェクトによってC++に置き換えられ、KokkosやRAJAなどのC++拡張の抽象化言語によってGPU向けの実装がなされた。2016年のプロジェクトスタート時には31パーセントであったFortranコードの占める割合は、2021年には18パーセントまで下降した。また、同文献によれば、GPUプログラミングモデルとして、Kokkosのみを用いるコードが11本、RAJAのみを用いるコードが6本あったのに対して、SYCL/DPC++のみを用いるコードはなかった。SYCL/DPC++で実装されているコードは、同時にCUDAおよびHIPのバージョンでも実装されていた。このことから、ECPプロジェクトでは、SYCL/DPC++はインテルGPU向けのプログラミング言語として、KokkosおよびRAJAをマルチプラットフォーム向けの言語として位置付けていると考えられる。

日本でのポーティング事例として2025年1月運用開始予定のMiyabiについて述べる[321-41]。Miyabiは、東京大学と筑波大学からなる最先端共同HPC基盤施設（JCAHPC）が導入したNVIDIA GPUからなるスーパーコンピュータである。JCAHPCは、2022年3月までインテルXeon Phiプロセッサからなるスーパーコンピュータ Oakforest-PACS (OFP) を運営していた。OFPでのアプリケーションをMiyabiで実行させるためには、CPUシステムからGPUシステムへのポーティングが必要となる。プログラミングモデルとしては。OpenACCもしくはC++ StdParを推奨するものとした。自ら移植するユーザの支援として定期的なハッカソン、サポートサイト[321-42] を開設した。また、多くのユーザを有するコミュニティコード19種類を選定し、サポート移植を行った。サポート移植の対象のコードのうち、16本がFortranのみ、1本がFortranとC、1本がC、1本がC++で記述されており、これらの多くはOpenACCによりGPU向けにポーティングされた。

3.2.1.4 アプリケーションのベンチマーク

本調査研究で、評価に用いるために整理されたアプリケーションセット（以降、FSベンチマークと呼ぶ）について、GPU化の状況と現在利用可能なGPUシステムにおける性能評価を行った。FSベンチマークを表 3.2.1.15 に示

す。

表 3.2.1.15 FS ベンチマークのアプリケーション

Program	Type	Language	GPU	Strong/ Weak	Fugaku	a100	qc-a100	qc-h100	HBW2	qc-gh200	qc-mi250	LUMI	Note
GENESIS	Molecular Dynamics (MD)	Fortran	CUDA	Strong	1,2,4		1,2		1	1,2,4,8	*G1	*G1	G1: AMD GPU code is not released.
SCALE	Weather/ climate	Fortran	OpenACC	Strong	2,3,4		2		N/A	*SL1	*SL2	N/A	SL1: Runtime error (segmentation fault) SL2: Compile error w/ -fopenacc
LETKF	Data assimilation	Fortran	OpenACC	Strong	2,3,4		2		N/A	2,3,4	*SL2	N/A	
FrontFlow /blue	Computational Fluid Dynamics (CFD)	Fortran	OpenACC	Weak	1,2,4	1,2			1	1,2,4	N/A	N/A	
SALMON	Density Functional Theory (DFT)	Fortran	OpenACC	Strong	1,2,3,4	1,2			N/A	1,2,3,4	N/A	N/A	
LQCD-DWF-HMC	Lattice Quantum ChromoDynamics (LQCD)	C++	OpenACC	Strong, Weak	1,2,3,4		1,2	1	1	1,2,3,4	*L1	*L1	L1: Unable to create valid AMD GPU code, Measuring again

6本のうち、4本のコードがFortranとOpenACC、1本のコードがC++とOpenACC、1本がFortranとCUDAで記述されていた。FotranとCUDAでの記述は、カーネル部分のみCとCUDAで記述し、Fortranから関数呼び出しを行うものである。

表 3.2.1.16 評価対象システム

Machine	CPU	# of CPUs	cores/CPU	Total	GPU	GPU
Fugaku	Fujitsu A64FX	1	48	48	N/A	0
a100	AMD EPYC 7763	2	64	128	NVIDIA A100	8
qc-a100	AMD EPYC 7713	2	64	128	NVIDIA A100	8
qc-h100	AMD EPYC 9534	2	64	128	NVIDIA H100	4
qc-gh200	NVIDIA GH200 Grace	1	72	72	NVIDIA H200	1
HBW2	Intel Xeon Platinum 8480+	2	56	112	NVIDIA H100	4

評価対象となるCPUおよびGPUについて表 3.2.1.16に示す。AMDのGPUに向けたOpenACCのコードのコンパイルができないため、本評価では対象外とした。

表 3.2.1.17 GENESIS の評価結果

MACHINE	Compiler	# of nodes	# of cores	# of GPUs	# of MPI procs	# of OMP threads	Time (sec)
Fugaku	FJ	1	48	0	16	3	124.4
Fugaku	FJ	2	96	0	32	3	73.4
Fugaku	FJ	4	192	0	64	3	42.7
qc-a100	GNU+CUDA	1	128	8	16	8	25.6
qc-a100	GNU+CUDA	2	256	16	16	16	20.1
qc-gh200	GNU+CUDA	1	72	1	4	18	25.5
qc-gh200	GNU+CUDA	2	144	2	8	18	23.1
qc-gh200	GNU+CUDA	4	288	4	16	18	20.0
qc-gh200	GNU+CUDA	8	576	8	32	18	18.6
HBW2	Intel+CUDA	1	112	4	8	14	18.6

創薬のための分子動力学のコードであるGENESIS[321-43, 321-44]の評価結果を表 3.2.1.17に示す。GENESISはカーネル部分をCUDAで実装している。単体ノード性能においては、CPU+GPUノードにおいて、富岳の1ノードと比較して最大で6.7倍の性能向上が見られた。ただし、強スケーリングにおける性能向上には課題が残った。

表 3.2.1.18 SCALE+LETKF の評価結果

SCALE

MACHINE	Compiler	# of nodes	# of cores	# of GPUs	# of MPI procs	# of OMP threads	Time (sec)
Fugaku	FJ	2	96	0	12	8	85.66
Fugaku	FJ	3	144	0	12	12	46.86
Fugaku	FJ	4	192	0	12	16	41.316
qc-a100	nvfortran	2	256	16	12	1	76.2

LETKF

MACHINE	Compiler	# of nodes	# of cores	# of GPUs	# of MPI procs	# of OMP threads	Time (sec)
Fugaku	FJ	2	96	0	12	8	85.66
Fugaku	FJ	3	144	0	12	12	46.86
Fugaku	FJ	4	192	0	12	16	41.316
qc-a100	nvfortran	2	256	16	12	1	76.2

表 3.2.1.18にデータ同化による気象アプリケーションであるSCALE+LETKF[321-45]の結果を示す。単一ノードではメモリ量不足となるため、2ノードからの測定とした。現時点ではGPUの導入による性能向上が見られなか

った。

表 3.2.1.19 FrontFlowBlue の評価結果

MACHINE	Compiler	# of nodes	# of cores	# of GPUs	# of MPI procs	# of OMP threads	Time (sec)
Fugaku	FJ	1	48	0	4	48	28.5
Fugaku	FJ	2	96	0	8	96	30.5
Fugaku	FJ	4	192	0	16	192	32.2
a100	nvfortran	1	128	8	4	4	107
a100	nvfortran	2	256	16	8	8	134
qc-gh200	nvfortran	1	72	1	4	4	89.8
qc-gh200	nvfortran	2	144	2	8	8	114
qc-gh200	nvfortran	4	288	4	16	16	126
HBW2	nvfortran	1	112	4	4	4	90.3

表 3.2.1.19に 空力学向け流体力学のコードであるFrontFlowBlueの評価結果を示す。富岳と比較して、メモリスループットおよびCPUとGPUとのデータ転送の影響で、GPUを用いるほうが性能が低いという結果になった。本コードはMiyabiのサポート移植プログラムに選択され、GPU向けの最適化が進んでいることから、今後の性能向上が期待される[321-46]。

表 3.2.1.20 SALMON の評価結果

MACHINE	Compiler	# of nodes	# of cores	# of GPUs	# of MPI procs	# of OMP threads	Ground state (sec)	Time evolution (sec)
Fugaku	FJ	1	48	0	4	12	133.93	13.982
Fugaku	FJ	2	96	0	8	12	72.248	7.682
Fugaku	FJ	3	144	0	12	12	55.409	5.385
Fugaku	FJ	4	192	0	16	12	39.396	4.343
a100	nvfortran	1	128	8	1	1	442.443	4.275
a100	nvfortran	2	256	16	2	1	227.794	3.068
qc-gh200	nvfortran	1	72	1	1	1	319.059	2.405
qc-gh200	nvfortran	2	144	2	2	1	177.114	1.407
qc-gh200	nvfortran	3	216	3	3	1	111.513	1.182
qc-gh200	nvfortran	4	288	4	4	1	86.185	1.013

表 3.2.1.20に物性の第一原理計算のコードであるSALMON[321-48]の評価結果を示す。SALMONのコードは大きく基底状態の計算 (Ground State) と 時間進化 (Time evolution) に分かれている。GPUの導

入により、時間進化においては性能の向上が見られたが、基底状態計算においては性能が低下した。これは、基底状態におけるコードのポーティングが不十分で、行列関等のライブラリについてCPU向けのライブラリを用いているからである。GPU向けに最適化したライブラリにおきかえることでさらなる性能の向上が期待される[321-47]。

表 3.2.1.21 LCD-DWF-HMC の評価結果

MACHINE	Compiler	# of nodes	# of cores	# of GPUs	# of MPI procs	# of OMP threads	32x8x8x12 /node (GFLOPS)	64x32x16x12 /node (GFLOPS)
Fugaku	FJ	1	48	0	4	12	397	422
Fugaku	FJ	2	96	0	8	12	402	426
Fugaku	FJ	3	144	0	12	12	401	411
Fugaku	FJ	4	192	0	16	12	397	420
qc-a100	nvhpc-hpcx/24.9	1	128	1	1	NA	699	1831
qc-a100	nvhpc-hpcx/24.9	2	256	2	2	NA	265	492
qc-gh200	nvhpc-hpcx/24.9	1	72	1	1	NA	1123	4077
qc-gh200	nvhpc-hpcx/24.9	2	144	2	2	NA	521	1730
qc-gh200	nvhpc-hpcx/24.9	3	216	3	3	NA	499	1723
qc-gh200	nvhpc-hpcx/24.9	4	288	4	4	NA	470	1649
qc-h100	nvhpc-hpcx/24.9	1	128	1	1	NA	983	2287
HBW2	nvhpc/24.3	1	112	1	1	NA	915	3303

表 3.2.1.21 に 量子色力学のコードである LCD-DWF-HPC (Hybrid Monte-Carlo (HMC) simulation program for Lattice QCD (LQCD) with domain wall fermions (DWF)) [321-49]の結果を示す。本コードのみ評価結果が実行時間ではなくFLOPSで提供されている点に注意されたい。GPUの導入により高速化が可能となっている。

以上より、ターゲットアプリケーションについては、GPUの利用により性能向上が見られるもの、GPU化がされているものの最適化がまだ不十分であるものがある。本評価に時点においては、Miyabi向けの最適化がされたコードを用いていないため、一部のコードについては性能が向上している可能性がある。ただし、その場合でもGPU化はOpenACCをメインで行われていることから、現時点ではIntelおよびAMDのGPUでは実行することができない。

参考文献

[321-1] ""Status of Flang" round table", 2024 European LLVM Developers' Meeting, , https://llvm.swoogo.com/2024eurollvm/session/1917539/round-tables, 2024

[321-2] ""Implementation of Coarray in flang-new" round table", 2024 European LLVM Developers' Meeting, https://llvm.swoogo.com/2024eurollvm/session/1917538/round-tables, 2024

[321-3] Leandro Lupori, Carlos Seo, "State of LLVM Flang Development", LinaroConnect, https://resources.linaro.org/en/resource/4SFvjFkHNfR28g2x3qgnUG, 2024

[321-4] Mark LeAir, "Flang Open Source Fortran Report", https://j3-fortran.org/doc/year/24/24-135.txt, 2024

[321-5] The Flang Team, "Combined Flang Community Call Notes", https://docs.google.com/document/d/1Z2U5UAtJ-Dag5wlMaLaW1KRmNgENNAYynJqLW2j2AZQ

[321-6] Michael Larabel, "LLVM's Modern "Flang-New" Fortran Compiler Renamed To "Flang", https://www.phoronix.com/news/LLVM-Flang-New-To-Flang

[321-7] Will Lovett, "Introducing the next generation of Arm Compiler", LinaroConnect, 2024

[321-8] Wikipedia "MLIR (software)", https://en.wikipedia.org/wiki/MLIR_(software)

[321-9] Multi-Level IR Compiler Framework, "MLIR Language Reference", https://mlir.llvm.org/docs/LangRef/#dialects, 2024

[321-10] LLVM, "ClangIR (CIR), https://llvm.github.io/clangir/

[3.2.1-11] LLVM, "Documentation, Vectorization Plan", https://llvm.org/docs/VectorizationPlan.html (2024/12/27閲覧)

[321-12] Florian Hahn, "VPlan: Status Update and Future Roadmap", 2023 LLVM Developers' Meeting, 2023

[321-13] Florian Hahn, " VPlan Progress Update August 2023 ", https://discourse.llvm.org/t/vplan-progress-update-august-2023/73003, 2023

[321-14] Paschalis Mpeis, "[RFC] Vectorization support for histogram count operations ", https://discourse.llvm.org/t/rfc-vectorization-support-for-histogram-count-operations/74788/1, 2023

[321-15] Aditya K, " Monthly Vectorizer online sync-up ", https://discourse.llvm.org/t/monthly-vectorizer-online-sync-up/78978/1, 2024

[321-16] The MLGO Authors, "Infrastructure for MLGO - a Machine Learning Guided Compiler Optimizations Framework.", https://github.com/google/ml-compiler-opt, 2024/12/27閲覧

[3.2.1-16] Dejan Grubisic, Chris Cummins, Volker Seeker, Hugh Leather, "Compiler generated feedback for Large Language Models", https://arxiv.org/abs/2403.14714, 2024

[3.2.1-17] Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, Amir Gholami, "An LLM Compiler for Parallel Function Calling", https://arxiv.org/abs/23.2.1.04511, 2024

[3.2.1-18] H. Tu, Z. Zhou, H. Jiang, I. N. B. Yusuf, Y. Li and L. Jiang, "Isolating Compiler Bugs by Generating Effective Witness Programs With Large Language Models," IEEE Transactions on Software Engineering, vol. 50, no. 7, pp. 1768-1788, doi: 10.1109/TSE.2024.3397822, 2024

[3.2.1-19] Hasan Ferit Eniser, Hanliang Zhang, Cristina David, Meng Wang, Maria Christakis, Brandon Paulsen, Joey Dodds, Daniel Kroening, "Towards Translating Real-World Code with LLMs: A Study of Translating to Rust", https://arxiv.org/abs/2405.11514, 2024

[3.2.1-20] Aiden Grossman, Ludger Paehler, Konstantinos Parasyris, Tal Ben-Nun, Jacob Hegna, William Moses, Jose M Monsalve Diaz, Mircea Trofin, Johannes Doerfert, "ComPile A Large IR Dataset from Production Sources", https://llvm-ml.github.io/ComPile/

[3.2.1-21] Aiden Grossman, Ludger Paehler, Konstantinos Parasyris, Tal Ben-Nun, Jacob Hegna, William Moses, Jose M Monsalve Diaz, Mircea Trofin, Johannes Doerfert, "ComPile: A Large IR Dataset from Production Sources", https://arxiv.org/abs/2309.15432, 2023

[3.2.1-22] D. Callahan, J. Dongarra and D. Levine, "Vectorizing compilers: a test suite and results," Supercomputing '88:Proceedings of the 1988 ACM/IEEE Conference on Supercomputing, Vol. I, pp. 98-105, doi: 10.1109/SUPERC.1988.44642., 1899

[321-23] S. Maleki, Y. Gao, M. J. Garzaran, T. Wong and D. A. Padua, "An Evaluation of Vectorizing Compilers," 2011 International Conference on Parallel Architectures and Compilation Techniques, Galveston, pp. 372-382, doi: 10.1109/PACT.2011.68., 2011

[321-25] LLVM Compiler Infrastructure, "Compiling CUDA with clang",https://llvm.org/docs/CompileCudaWithLLVM.html, 2024/12/27閲覧

[321-26] LLVM Compiler Infrastructure, "AMD GPU Support", https://clang.llvm.org/docs/HIPSupport.html#amd-gpu-support, 2024/12/27閲覧

[321-27] The Flang Team, "Building flang runtime for accelerators", https://flang.llvm.org/docs/GettingStarted.html#building-flang-runtime-for-accelerators, 2024/12/27閲覧

[321-28] https://openmp.llvm.org/openacc/Overview.html

[321-29] The Khronos Group, "OpenCL", https://www.khronos.org/opencl/

[321-30] Kokkos Project, "Kokkos", https://kokkos.org/

[321-31] https://kokkos.org/blog/blog-post-08/

[321-32] Lawrence Livermore National Laboratory , "RAJA Portability Suite", https://computing.llnl.gov/projects/raja-managing-application-portability-next-generation-platforms

[321-33] https://github.com/LLNL/CHAI

[321-34] The Khronos Group "SYCL Overview",https://www.khronos.org/sycl/

[321-35] AdaptiveCPP

[321-36] https://github.com/AdaptiveCpp/AdaptiveCpp?tab=readme-ov-file#adaptivecpp-formerly-known-as-hipsycl--open-sycl
[321-37] Benjamin Worpitz, "Investigating performance portability of a highly scalable particle-in-cell simulation code on various multi-core architectures", Master Thesis, Technische Universitat Dresden, 2015
[321-38] https://github.com/AnswerDotAI/gpu.cpp
[321-39] Oak Ridge Leadership Computing Facility, "Frontier Center for Accelerated Application Readiness ", https://www.olcf.ornl.gov/caar/frontier-caar/
[321-40] Thomas M. Evans, Andrew Siegel, Erik W. Draeger, Jack Deslippe, Marianne M. Francois, Timothy C. Germann, William E. Hart, and Daniel F. Martin, "A Survey of Software Implementations Used by Application Codes in the Exascale Computing Project", https://crd.lbl.gov/assets/Uploads/ijhpc-2021.pdf
[321-41] 中島研吾, "（計算・データ・学習）融合による革新的スーパーコンピューティングとその先にあるもの", 2024
[321-42] 最先端共同HPC基盤施設（JCAHPC), "GPU移行 ",https://jcahpc.github.io/gpu_porting/
[321-43] Computational Biophysics Research Team, R-CCS “GENESIS (GENeralized-Ensemble SImulation System)”,https://www.r-ccs.riken.jp/labs/cbrt/
[321-44] GENESIS: a hybrid-parallel and multi-scale molecular dynamics simulator with enhanced sampling algorithms for biomolecular and cellular simulations
Jaewoon Jung, Takaharu Mori, Chigusa Kobayashi, Yasuhiro Matsunaga, Takao Yoda, Michael Feig, Yuji Sugita, "GENESIS: a hybrid-parallel and multi-scale molecular dynamics simulator with enhanced sampling algorithms for biomolecular and cellular simulations", Wiley Interdisciplinary Reviews: Computational Molecular Science 5, 310--323, (2015)
[321-45] Data Assimilation Research Team, RIKEN Center for Computational Science, "SCALE-LETKF-RIKEN", https://github.com/SCALE-LETKF-RIKEN/scale-letkf/
[321-46] 塙俊博「最先端共同HPC基盤施設の新スパコンシステムMiyabiの運用開始に向けて/東大情報基盤センターの目指す『計算・データ・学習』とデータ利活用を融合した革新的なスーパーコンピューティング」PCCC24（第24回PCクラスタシンポジウム）「サステイナブルなHPCに向けて」 https://www.pccluster.org/ja/event/pccc24/data/PCCC24_5-9_itc-u-tokyo_hanawa.pdf
[321-47] CEA, “CExA project”, https://cexa-project.org/
[321-48] AMD, “Quantum Comoputing”, https://www.amd.com/en/solutions/quantum-computing.html
[321-49] Intel, “Intel Quantum SDK”https://www.intel.com/content/www/us/en/developer/tools/quantum-sdk/overview.html

3.2.2　ジョブスケジューラ・ランタイム調査研究グループ

3.2.2.1　調査目的

次世代計算基盤は多種多様かつ膨大な量の計算資源から構成される超大規模システムになることが想定される。一方で、次世代計算基盤で実行されるアプリケーションや実行方法も現在よりさらに多様化する。時間的特性、空間的特性、資源分離レベルの異なる多数のジョブを、多種多様な計算資源に適切に割当てることは、それ自体が挑戦的な技術的課題である。特に、現在および将来において我が国で開発されるシステムやアプリケーションの効率的なスケジューリング技術およびその実行支援技術は、我が国が独自に検討しなければならない。それらの技術の研究開発を進めるために、国際協調の下で研究開発が進んでいるジョブスケジューリング技術の動向把握と、我が国が独自に追加検討すべき機能要件の調査を行う。

3.2.2.2　調査内容

ジョブスケジューラとその関連技術について、情報基盤センターごとに現在の利用状況および今後の運用方針に関して調査して要求要件の整理を行うほか、アプリケーションの効率的なスケジューリング技術およびその実行支援技術に関して国際協調の下で研究開発が進んでいるジョブスケジューリング技術の動向把握と我が国が独自に追加検討すべき機能要件の調査を行う。

令和4年度には、実運用システムで広く使われている有力なジョブスケジューラを把握するために、TOP 500 Listにおける上位100システムでの採用状況、および各情報基盤センターの採用状況を調査し、国際的な採用状況の趨勢を明らかにしている。また、今後研究開発すべきジョブスケジューリング技術の課題を明確にするため、文献調査やベンダとのミーティングを行い、今後予想されるジョブスケジューリングの技術的課題についても調査を進めた。特に、富士通株式会社の技術者とのミーティングを複数回行い、京や富岳のジョブスケジューラ開発を振り返るとともに今後に向けて検討すべき課題についても議論した。さらに、ワークフロー管理システムや継続的プロファイリングを実現するためのモニタリングツールなどについても調査を行った。令和5年度には、システム大規模化に向けたジョブスケジューラの負荷分散に関してさらに調査を進めるとともに、富士通株式会社の技術者との定期的なミーティングを継続し、京や富岳におけるジョブスケジューラ開発の実績に基づいて今後に向けて検討すべき課題について議論を継続した。

令和6年度には、性能分析や耐障害性などを実現する実行支援ツール、およびワークフロー実行基盤に関する文献調査を進めつつ、アクセラレータを搭載する方向性が明らかになったことから、GPUを共有利用するための既存技術についても調査を進めた。また、世界的に広く使われているオープンソースソフトウェアから基盤とすべきソフトウェアを選定し、候補アーキテクチャにおいて効率の良い資源管理を実現するために必要な開発要素を調査した。これまでの調査結果から、基盤とするジョブスケジューラの有力候補としてはSlurm Workload Manager (以下 Slurm) [1]を選択し、コンテナ仮想化技術や運用ツール、アクセラレータ対応に必要な機能のうちSlurmのプラグインとしての実装の可否を、富士通株式会社の技術者と定期的なミーティングを開催して議論・調査した。

3.2.2.3　調査結果

3.2.2.3.1　性能分析や耐障害性などを実現する実行支援ツール

次世代計算基盤は、システムの大規模化・複雑化に伴って運用・管理においてさまざまな課題が顕在化することが予想される。現在、システムの大規模化に伴う耐障害性の確保や、障害発生時に安定したシステム運用を可能とする要素技術の研究開発が精力的に進められている。このため、システムの異常を速やかに検知する技術や、将来発生しうる障害・異常を予測する技術に関する調査を昨年度に引き続き行った。

次世代計算基盤は大規模システムとなることが避けられず、CPUやメモリ、ネットワークの競合などにより、予期しない性能低下などの異常が発生する。このような異常は、システム障害を直接引き起こすわけではないものの、アプ

リケーションの計算効率およびシステムの運用効率を大幅に低下させる可能性がある。そのため、機械学習と大規模なシステムログ・モニタリング情報を活用し、これらの異常を予測・検知する技術の開発が進められている。しかし、膨大なシステムログやモニタリング情報に適切なラベルを付与することは困難であり、そのコストも無視できない。そこで文献 [2]では、能動学習を用いることでラベル付きデータの量を抑えつつ、高精度に異常診断を行う手法が提案されている。提案フレームワークでは、まず少量のラベル付きデータを用意し、教師あり分類器を訓練する。その後、訓練済みの分類器を用いてログ・モニタリング情報から異常の検出を試みる。この際、ラベルの付いていないデータが多いため、クエリ戦略に基づき、分類器が異常検知・予測に有用と判断した特徴量にラベルを付与し、新たなラベル付きデータを追加して分類器を再訓練する。この作業を繰り返すことで、少ないラベル付きデータで高精度な異常検知・予測を実現する。本手法の評価には、ノードあたり806個のモニタリング情報と6つのアプリケーションを使用した。その結果、能動学習を導入しない場合には5619個のラベル付きサンプルが必要であるのに対し、提案手法では256個のラベル付きサンプルで同等のF1スコアを達成できることが明らかになった。

一方、システムの故障・障害発生を予測する手法として、自然言語処理を用いたアプローチがいくつか提案されている。従来のシステムログを用いた障害検知・予測手法では、ログから特定のIDを抽出して入力データとする方法が主流であった。しかし、このアプローチでは、ログの情報が十分に活用されず、重要な情報が失われる可能性がある。そこで文献 [3]は、システムログをそのままテキストデータとして扱い、言語モデルを用いて障害の予兆を検知する手法としてFALL (Prior Failure Detection in Large-Scale Systems Using Language Models)を提案している。FALLでは、正常なログデータのみを用いてモデルの訓練を行う。異常検知においては、正常・異常のログメッセージ間での最小限の語彙変化に注目し、微細な違いを強調するシャープニング技術を導入している。また、ログメッセージの語彙が限定的である点に着目し、効率的な異常検知・予測を実現している。システムログデータはシステムごとに差異はあるものの、比較的容易に取得可能である。そのため、大量のセンサ・観測データを集約・モニタリングする手法と比較して導入しやすく、同等の予測精度を達成できることが報告されている。

現在、これまでの報告にも示してきた通りさまざまな異常検知・障害発生予測の手法が検討されているが、迅速な異常検知と高精度な予測を可能とするための手法が確立されているとは言い難いのが現状である。特に、モニタリング情報やシステムログは対象とするシステムによって異なるため、システムに依存した手法が多く、既存手法をそのまま新たなシステム適用するのが困難である。システム設計の早期段階でシステムモニタリングの情報集約・解析手法と事前に検討することで、システムの高い信頼性を担保することが可能になると考える。

3.2.2.3.2　ワークフローの実行基盤

ワークフロー実行基盤は多数開発されているため [4]、学術論文で言及されている回数やGitHub上でのリポジトリのスター数などの観点から、特に注目が集まっていると想定されるApache Airflow、Prefect、およびGlobus Flowsの3種の実行基盤に着目し、調査した。

Apache Airflow [5]は、2015年にAirbnb社の内製システムとして開発が始まり、後にオープンソースソフトウェアとして公開され、Apacheソフトウェア財団に移管された。現在はAmazon社やAstronomer社などの企業がAirflowに基づくワークフロー実行基盤をSoftware as a Service (SaaS) として提供している。Airflowでは、ワークフローを構成する各々のタスクの種類に応じたOperatorと呼ばれるオブジェクトを生成し、これらを接続してワークフローを表現する有向非巡回グラフ (Directed Acyclic Graph, DAG) を構成する。Operatorはプラグインによって追加可能であり、Python関数を実行するもの、コマンドを実行するもの、各種データベースやストレージに対する読み書きを行うもの、Slackへのメッセージ送受信を行うものなど、種々のOperatorが公開されている。タスクの実行環境としては、Celery [6] (Python向けジョブキュー) やKubernetes [7]などに対応しており、柔軟な運用が可能である。

Prefect [8] は、Apache Airflowの元開発者によって開発が開始され、2017年にオープンソースソフトウェア

として公開された。同名の企業が提供するSaaSは、ワークフローの実行を制御するスケジューラを提供するものであり、タスクの実行環境はユーザが準備する必要がある。Prefectは、Airflowよりも洗練されたユーザインターフェース（ワークフローを定義するためのAPIやワークフローの実行を制御・監視するWeb UI等）を訴求している。Prefectではタスクを Python関数として定義し、タスク間の依存関係を関数の引数や返り値として記述する。これにより、Pythonスクリプトを記述するような感覚でワークフローを記述することができる。なお、Airflowのように多様なOperatorは事前に用意されておらず、外部サービスとの連携は基本的に各種ライブラリなどを使用してユーザがプログラムすることが想定されている。

Globus Flows [9] は、アルゴンヌ国立研究所やシカゴ大学を中心としたGlobusチームによって開発され、Globusエコシステムの一部として提供されている。Globus Flowsは、Globus Transfer [10] (大容量データ転送サービス) やGlobus Compute [11] (Function as a Serviceの一種) などのGlobusサービスを組み合わせ、ワークフローを構成することを目的としている。特定の種類のタスクを実行するサービスは「Action Provider」と呼ばれ、ワークフローはこれらAction Providerを順に呼び出していくシーケンスとして定義されている。Action ProviderはRESTful APIに従ったインターフェイスのみが定義されており、内部の実装は自由に行うことができる。このため、Globusが提供するサービス(Globus TransferやGlobus Search)を利用したり、Globus Computeで配備した関数をAction Providerとして使用したり、Action Provider SDKを用いて独自のものを作成することが可能である。

以上の基本的な調査結果を踏まえ、さらに、(1) ワークフローの記述方法、(2) ワークフローの起動方法、(3) タスク間のデータ移動方法、(4) タスクの実行環境、の4つの視点から、調査対象のワークフロー実行基盤を比較した結果を以下にまとめる。

- **ワークフローの記述方法**: Prefect では、タスクは Python の関数として記述され、依存関係は関数の引数および戻り値から自動的に導出される。DAG は動的に構築されるため、if/else や while などの制御構文も使用可能である。これに対して、Apache Airflow では各タスクに対応するオブジェクトを作成し、オブジェクト間に依存関係を設定する。ただし、最近のバージョンでは Airflow でも Prefect のような Pythonic な記述も可能となっている。一方、Globus Flows では、ワークフローは JSON ファイルで記述され、一直線の DAG（各タスクの上流タスクおよび下流タスクはそれぞれ高々1 つ）しか記述できない。
- **ワークフローの起動方法**: いずれのワークフロー実行基盤も Web API や Web UI を通じてワークフローを起動する仕組みを共通して備えている。Airflow には Sensors という Operator の種類があり、日時、時間周期、ファイル作成などのイベントを契機にワークフローを起動することができる。Prefect はそのような特別な仕組みは備えておらず、ユーザのスクリプトやインフラ側でイベントを監視し、API を呼び出す形を取る。Globus Flows も Prefect と同様に特別な仕組みは備えていない。
- **タスク間のデータ移動方法**: Airflow では、XComs [12] という仕組みを用いてデータベースやオブジェクトストレージを経由してタスク間でデータを受け渡す。Prefect では、関数の引数および戻り値としてメモリを介して自動的にデータが受け渡されるが、大容量のデータについてはローカルストレージやオブジェクトストレージに読み書きして受け渡すことが推奨されている [13]。Globus Flows では、データはファイルとして Globus Transfer により移動される。
- **タスクの実行環境**: Airflow では、タスクは Kubernetes や Celery クラスタ内の複数ワーカに分散して実行される。Prefect においては、1 つのワークフローは基本的に 1 つのワーカ上で実行され、分散実行を行う場合は別途 Dask [14] や Ray [15] などの Python 用分散処理フレームワークを組み合わせる必要がある。Globus Flows では、タスク実行環境はタスクの機能を提供する Action Provider に任されている。Action Provider として Globus Compute を使用する場合、各タスクは独立した Globus Compute 関数として配

備し、その際に関数を実行する計算資源を指示する。各関数は負荷に応じて Globus Compute の機能により自動的にスケールアウト・インされる。

以上のように実行基盤によって長短は存在するものの、ワークフローのプログラミングモデルや実行技術は既に十分確立されていると言える。一方、システム外部の計測機器や計算基盤と連携したワークフローの実現は未だ困難である。既存のワークフロー実行基盤は基本的にバッチ実行を前提としており、投入されたワークフローのタスクはバッチジョブのように計算資源が利用可能になり次第、順次実行される。しかし、測定技術やネットワーク技術の向上により、次世代計算基盤には、放射光施設や高圧電子顕微鏡などの大型実験施設や、大規模センサーネットワークなどのデータ源から継続的に生成される膨大なデータをリアルタイムに解析し、科学的知見を生み出したり、実験の制御にフィードバックしたりする役割が期待される。そのため、次世代計算基盤向けのワークフロー実行基盤には、地理的に分散した複数のデータ源と計算資源の間で大容量データを転送し、リアルタイムに処理する機能が求められる。

3.2.2.3.3　GPU の共有利用技術

GPUの飛躍的な性能向上にともない、単一のプログラムで1基のGPU全体の性能を使い切ることが困難になっている。そのため、1基のGPUの計算資源を分割し、複数のプログラムで効率的に共有利用するための技術が開発・提供されている。表 3.2.2.1にNVIDIA社が自社のGPU向けに提供する共有利用技術のうち、代表的なものをまとめる。

GPUを使用するプログラムを同時に複数起動した場合、デフォルトではTime Slicing [16] によって多重実行される。Time Slicingは時分割多重化の一種であり、CPUのマルチタスキングと同様に、定期的にプログラムをプリエンプションし他のプログラムにコンテキストスイッチングする。Time Slicingでは任意の時点においてGPU上で実行されているプログラムは常に1つであるため、プログラム間の性能干渉は小さい。一方で、GPUの資源を使い切らないプログラムを複数多重実行してもGPUの資源利用率を向上させることはできない。また、GPUのコンテキストは巨大(数百MB) であるため、コンテキストスイッチングのオーバーヘッドを無視できない。

Multi-Process Service (MPS) [17] は空間分割多重化の一種であり、GPUの資源を複数のプログラムで共有し、同時に実行可能にさせる。MPSでは資源使用量が少ないプログラムを複数多重実行することにより、GPUの資源利用率を向上させることができる。また、各プログラムが利用するスレッド数の割合をパーセンテージで指定することにより、各プログラムへの演算性能の割当を柔軟に制御できる。一方、メモリやL2キャッシュ等の資源は全てのプログラムによって共有されているため、プログラム間でこれらの資源に起因にする性能干渉が発生する可能性がある。

Multi-Instance GPU (MIG) [18] はMPSと同じく空間分割多重化の一種であるが、ハードウェア機能により資源間の隔離を強固にしている。具体的には、1つのGPUを複数のGPUインスタンスと呼ばれる単位に分割する。各GPUインタンスにはStreaming Multiprocessor (SM)、メモリ、L2キャッシュ等の資源が静的に割り当てられており、GPUインタンスの間ではメモリ帯域幅やキャッシュ帯域幅の干渉は発生し得ない。ただし、GPUインタンスへの資源割当割合はハードウェアによって制約されており、MPSのように柔軟な分割設定はできない。具体的には、事前に定義された組み合わせ (GPUプロファイル) の中から選択しなければならない。また、GPU上でプログラムが実行されている間は分割設定を変更できない。

表 3.2.2.1 NVIDIA GPU における共有利用技術

	Time Slicing	Multi-Process Service (MPS)	Multi-Instance GPU (MIG)
性能の分割	不可	可 (%で指定)	可 (プロファイルから選択)
メモリ帯域の分割	不可	不可	可
メモリ保護	可	可	可
エラーの隔離	不可	不可	可
分割数	制限なし	48 (Volta以降)	7 (A100, H100, H200) 4 (A30)
分割の設定タイミング	実行時	プログラム起動時	アイドル時 (Hopper以降)

このようにNVIDIA社から提供されているGPUの共有利用技術は一長一短があり、要件に応じて使い分ける必要がある。また、これらのNVIDIA社の提供する共有利用機能に加えて、NVIDIA GPUに関する独自の共有利用技術の研究開発も活発に行われている。

文献 [19] はGPUの共有利用に関する先駆的な研究であり、深層学習用GPUクラスタにおける効率的なGPU共有利用のためのフレームワークAntManを提案している。AntManは、ジョブを高優先度ジョブ (単独実行時と同等の性能を保証) と低優先度ジョブ (ベストエフォートの性能) に分類し、これらを多重実行することによってGPUの利用率を向上させる。具体的には、低優先度ジョブからのカーネル投入タイミングを制御し、ホスト側での処理などにより高優先度ジョブのカーネルがGPU上で実行されていない間隔に実行する。これにより、高優先度ジョブの性能を劣化させることなく低優先度ジョブを実行することができる。また、各ジョブのGPUメモリ使用量の変動によってメモリ不足が発生することを防ぐため、GPUメモリが不足する場合はテンソルを単位としてデータをGPUメモリからホストメモリへ退避させる。

文献 [20] はKubernetesで管理された深層学習向けのGPUクラスタを対象とし、GPUの利用率を向上させるためのミドルウェアIADeepが提案している。IADeepはジョブ間の干渉による性能劣化を実行履歴に基づいてオンライン学習し、性能劣化が小さいと予測されるジョブの組み合わせを同一GPUにスケジュールする。スケジューリングを行う際には、多重実行による性能劣化とジョブキューでの待ち時間のトレードオフを考慮し、ジョブをいつ実行すべきか判断する。また、深層学習においてはミニバッチサイズがSM利用率、PCIe帯域幅、メモリ使用量などの資源使用量と相関することに着目し、ジョブ間の干渉が少なくなるように動的にバッチサイズを調整するように深層学習フレームワークを拡張している。

文献 [21] は、既存のGPU共有利用技術の多くにおいて深層学習フレームワークに対する修正が必要であることに着目し、フレームワークに依存しない多重実行技術TGSを提案している。TGSはCUDA Driver APIをフックし、ユーザのプログラムからのカーネル投入頻度を制御することにより、GPUの資源使用量を制御する。また、MPSやMIGでは各プログラムへの資源割当をユーザが決定しなければいけない課題に着目し、プログラムへの資源割当を自動的かつ適応的に行う仕組みを提案している。具体的には、AntManと同様にジョブを高優先度ジョブと低優先度ジョブに分類し、高優先度ジョブの性能を監視しながら低優先度ジョブからのカーネル投入頻度を制御する。具体的には、高優先度ジョブの性能が劣化しない範囲で低優先度ジョブからのカーネル投入の頻度を最大化する。

文献 [22] もフレームワーク非依存の多重実行技術Orionを提案している。OrionもAntManやTGSと同様に高優先度ジョブと低優先度ジョブを想定し、低優先度ジョブのカーネルを投入するタイミングを制御する。その際に、プロファイラ (NVIDIA Nsight SystemsおよびCompute) によって計測した各カーネルの特性を考慮する。具体的には、律速要因 (演算またはメモリ) が異なるカーネルの組み合わせは多重実行による性能劣化が少ないこと

に着目し、高優先度ジョブのカーネルと律速要因が異なる場合のみ低優先度ジョブのカーネルを多重実行する。また、カーネルが使用するSM数を計算し、閾値以下の場合のみ低優先度ジョブのカーネルを多重実行することにより、SMの競合を抑制する。

以上のようにGPUの共有利用を実現するためには、ジョブスケジューラ、GPUランタイム、およびアプリケーションフレームワークを連携させる必要がある。ジョブの性能劣化を抑制しつつGPUの資源利用率を向上させるためには、資源分割技術のみならず、同時に実行させるジョブの組み合わせを判断する技術が欠かせない。そのため、GPUの共有利用機能を実装する場合には、ジョブスケジューラやスケジューリングポリシの設計段階から検討し、設計に反映する必要がある。

3.2.2.3.4 ジョブスケジューラの開発要素に関する調査

3.2.2.3.4.1 コンテナ仮想化技術に関する開発要素の調査

次世代計算基盤では、スケジューラ管理下でコンテナワークロードが実施されると想定される。そのため、富岳のジョブスケジューラおよびSlurmジョブスケジューラのコンテナ利用方法について調査し、次世代計算基盤に向けた望ましい利用方法の選定と不足機能・開発要素の抽出を行った。

富岳では、すべてのジョブがDocker [23] 経由で動作している。デフォルトでは、ホストのファイルシステムがそのままコンテナ内に公開される設定となっている。以下のように、コンテナイメージを指定して実行した場合、当該イメージに含まれるルートファイルシステムに置き換わるという動作をする。

実行コマンド例)

```
pjsub -L jobenv=docker -x PJM_JOBENV_DOCKER_IMAGE=<image name> <script>
```

したがって、現状でもDockerを経由したコンテナイメージの利用は可能である。ただし、MPI等のノード間並列計算でコンテナイメージを指定する場合には、必要なライブラリやコマンドを含む専用のコンテナイメージの作成が必要となる。このため、イメージ作成の手順が煩雑となり、Technical Computing Site (TC-S)などシステムソフトウェアパッケージの入手が求められる等、一般ユーザにはイメージの作成が困難という点で課題が残っている。

次に、Slurmで管理したシステム上でのコンテナ利用を考える。ここでは、ユーザがログインノード上でコンテナ操作命令を実行し、計算ノード上でコンテナを起動することを想定する。その場合、以下の3つの選択肢がある。

- Singularity/Apptainer を利用したコンテナ起動

  実行コマンド例)

  ```
  srun singularity run <container image> <command>
  ```

  (Apptainer利用時には、上記コマンド例の中の`singularity`の部分が`apptainer`に置き換わる)

  SlurmのジョブとしてSingularity [24] やApptainer [25] を経由してコンテナを起動し、コマンドを実行する。富岳で採用されている方式と比較すると、特別なコンテナイメージを作成することなく、より簡単にノード間並列計算を実行できる。また、システム側で特別なデーモン等を立ち上げておく必要もなく、通常のプロセスと同様にコンテナを実行できるため、構築・運用・保守が容易という利点もある。ただし、Open Container Initiative (OCI)準拠のコンテナイメージ (Dockerのコンテナイメージなど)をそのまま利用することが以前はできず、Singularity Image File (SIF)形式のイメージに変換して実行する必要があった。SIFへ変換すると起動できなくなるOCIイメージも存在したため、後述のSlrumの--containerオプションが開発されたという経緯がある。Singularity CE 4.0.0以降ではOCI準拠のイメージを直接起動(OCI-mode)できるようになってこの問題は解決されたが、Apptainerでは未解決のままとなっている。

- Slurm の--container オプションを利用したコンテナ起動

実行コマンド例)

```
srun --container=<コンテナバンドル> <command>
```

ジョブ実行時に--containerオプションをつけ、コンテナバンドルを指定してコマンドを実行する。ここで、コンテナバンドルとはコンテナイメージ内のファイルを展開したものである。Singularity/Apptainerを利用する場合と比較すると、SIFファイルへの変換が不要という利点がある。このため、変換処理が理由でOCI準拠のコンテナイメージが実行できなくなる恐れがない。しかし、コンテナイメージを展開してコンテナバンドルを作成する必要があり、コンテナ起動の操作が煩雑になるという課題がある。また、Podman [26]やSingularity [24][25]、containerd [27]、CRI-O [28]といった高レベルなコンテナランタイム経由でのコンテナ操作はできない。

- Slurm の scrun を低レベルコンテナランタイムとして利用したコンテナ起動

  実行コマンド例)

  ```
  docker run <container image> <command>
  ```

  Docker等によるコンテナ操作の低レベルコンテナランタイムとしてscrunを利用し、コンテナを起動してコマンドを実行する。SlurmやSingularity/Apptainerのコマンドを学ぶことなく、Dockerコマンドのみでコンテナを起動できるという利点がある。しかし、利用できるDockerコマンドに制限があることや、開発中の実験的機能であることから品質が不十分で、商用サポートの範囲外となっていることなどが課題として残っている。

「Dockerなどを利用して実行していたコンテナワークロードを、Slurm上で容易に実行可能か」と「ポスト富岳の高い計算力を活用可能か」の観点から評価した結果を表 3.2.2.2にまとめる。

表 3.2.2.2 コンテナ実行方式に関する評価項目

(○：利用可能（制限なし）△：利用可能（制限あり）×：利用不可能)

評価項目	富岳	Singularity/Apptainer	Slurm (--container)	Slurm (scrun)
任意の高レベルコンテナランタイムコマンドの実行 (例：docker run)	× dockerdに対するアクセス手段なし	○ 任意コマンドを実行可能	× 高レベルコンテナランタイムを介してコンテナを管理していない	△ 実行できるコマンドに制限あり
Docker CLIとの互換性	× Docker CLIは利用不可	△ Docker CLIに類似したCLIが利用可能	× Docker CLIは利用不可	○
既存コンテナイメージの再利用性	△ 並列ジョブ実行の際に追加パッケージのインストールが必要	○	△ コンテナバンドルの作成が必要	○
ノード間並列計算	○	○	○	× Slurmのオプション指定が不可能なため

従来からバッチジョブによる利用が一般的であること、およびノード間並列計算を容易に実行できるという点から、HPC利用ユーザ向け利用方法としてはSingularity/Apptainerによるコンテナ利用が有望である。また、開発への参加の容易さという観点では、Apptainerのほうが容易であり有望である。ただし、現状のApptainerではOCI-

modeをサポートしていないため、そのサポートが運用で求められる場合には開発が必要である。一方、非HPC利用ユーザ向け利用方法としては、Dockerコマンドのみでジョブスケジューラとの連携が可能である点からSlurm(scrun)が有望である。ただし、現在ではSlurm(srun)は実験的な機能という扱いであり、次世代計算基盤に向けて実行可能なコマンドラインオプションの拡張やノード間並列計算への対応が必要となる可能性がある。

3.2.2.3.4.2 運用ツール移行に関する開発要素の調査

富岳のジョブスケジューラ(TC-S)からSlurmへ移行するにあたり、課金機能やユーザ管理機能などシステム運用ツールの移行が必要となる。しかし、移行に伴う開発要素が不明確であるため、開発要素の明確化や開発難易度の見積もりを行った。システム運用ツールがジョブスケジューラに要求する機能を4つのグループにまとめ、整理したものを表 3.2.2.3に示す。表 3.2.2.3の太字で示されている4つの機能のみ、開発が必要となる可能性がある。

表 3.2.2.3 システム運用ツールから必要とされるジョブスケジューラの機能

(太字はSlurm向けに開発が必要となる可能性のある項目)

グループ	概要	要求するスケジューラの機能
ジョブ・ノード管理機能	運用・保守時に必要となる機能群	- ジョブ停止・中断 - ノード中で実行中のジョブ取得 - ノード切り離し/組み込み - 資源予約
ジョブ・ノード情報収集機能	ユーザ向け情報を収集する機能群	- **外部公開用情報** - **実行中ジョブの消費電力情報**
フック機能	スケジューラに独自処理を追加するための機能群	- ジョブ投入可否判定（ジョブ投入時） - ジョブ実行可否判定（プロローグ） - スケジューリング参考情報設定（ジョブ投入時） - 外部プロセスの呼び出し（プロローグ/エピローグ） - ジョブ投入可否判定（ジョブ投入時） - 外部DBと連携したポイント減算（ジョブ投入時） - 外部DBと連携したポイント加減算（ジョブ終了時） - **スケジューリング可否判定（スケジューリング時）**
ジョブ操作・表示制御機能	アクセス制限などセキュリティ観点で必要となる機能群	- ジョブ操作の対象制限機能 - **ジョブ情報の表示制限機能**

まず、ジョブ・ノード管理機能グループに関しては、SlurmでもTC-Sも同等の機能が提供されているか、ほかの機能で代替可能となっており、移行に際して新たに開発が必要な機能は見つかっていない。

ユーザ向け情報を収集するためのジョブ・ノード情報収集機能グループに関しては、2つの機能とも開発が必要になる可能性がある。富岳の外部公開情報は、その情報収集に監視ツールPrometheus [29]、情報公開にGrafana [30]を利用しており、TC-S用に独自のPrometheusプラグインが実装・利用されている。富岳で外部公開されていた機能をすべて公開するためには情報収集プラグインの作りこみが必要であり、特に実行待ちノード時間積の情報を収集するPrometheusプラグインが必要である。また、TC-Sで用いられているPower API [31]と比較して、SlurmがバックエンドでIPMI等の分解能は低い。例えば、Power APIではCPUの消費電力をコア部分とそれ以外で個別に収集できるが、IPMIでは同等の情報を収集できない。このため、富岳と同様の情報公開を求める場合には、Slurmのプラグイン開発が必要となる。求められる情報の内容や分解能に依存するが、バックエンドにPower APIを使うSlurmのプラグインの開発を仮定すると、その難易度は比較的低いと想定される。

スケジューラに独自の機能を追加するためのフック機能グループに関しては、ほとんどの機能がすでにSlurmでも提供されているが、スケジューリング時のスケジューリング可否判定機能(スケジューリングが成功したか否かを判定する機能)のみ開発が必要となる可能性がある。Slurmではスケジューリングのフェーズでフックする機能を提供していない。ジョブスケジューリング後のジョブ実行前に独自処理を追加することで代替することは可能であり、その代替に要する開発難易度は低い。しかし、結果的に実行されないジョブまで不必要にスケジューリングすることになり、性能劣化やユーザからの見え方に差異が生じる懸念がある。そのような懸念が表面化した場合、Slurm本体の開発が必要となり、開発の難易度は比較的高いと想定される。

他のユーザのジョブに対する操作等、一般ユーザを限られた範囲で管理者として設定するジョブ操作・表示制御機能グループでは、基本的にSlurmはTC-Sと同等の機能を提供している。ただし、TC-Sでは情報表示の際にジョブ名、ユーザ名、グループ名をマスクして匿名化できるのに対して、Slurmでは同等のマスク機能がない。ABCIで行われているようにラッパーコマンドを提供する場合には、その難易度は比較的低い。しかし、情報表示の設定機能の改良のためにSlurm本体を修正するほうが望ましいが、その開発難易度は高いと想定される。

表 3.2.2.4 ジョブスケジューラの機能に関する開発項目のまとめ

機能名	開発項目	開発難易度	備考
外部公開用情報の収集機能	情報収集プラグインの移行開発	低	
	取得不可情報の出力機能開発	高	取得すべき情報が判明次第、開発要否が決定
実行中ジョブの消費電力情報収集機能	Slurmプラグイン開発(acct_gather_energy)	低	IPMI使用不可、要PowerAPI対応など要件に応じて開発要否が決定
スケジューリング可否判定(スケジューリング時)	代替フックポイントを利用した移行開発	低	代替フックポイントの利用により性能・ユーザービューの劣化の懸念
	フックポイント追加開発	高	代替フックポイントでは対応不可の場合、要開発
ジョブ操作・表示制御機能	ジョブ情報の一部マスク機能開発	低～高	設定機能の改良による実装の場合は高

3.2.2.3.4.3　アクセラレータ対応状況に関する開発要素の調査

次世代計算基盤にアクセラレータを搭載する公算が高いため、基本となるジョブスケジューラとしてSlurmを想定した場合のアクセラレータ対応状況を調査した。

SlurmのGRES(Generic RESource)プラグインを利用してアクセラレータ資源のジョブスケジューリングを行い、運用している事例は多い。GRESプラグインは設定ファイルの読み込み等のベンダ非依存の処理を担当し、デバイス情報取得やクロック数設定等のベンダ依存の処理はそれぞれのGPUプラグインが用意されている。そのほか、NVIDIA GPUのMulti-Process Service (MPS) [17] を利用するための補助プラグインも提供されている。ジョブ資源管理のために、Taskプラグインが用意されており、sched_setaffinity()やcgroupを用いて資源割当や制限を設定できる。

以下のような運用実績から、基本的なアクセラレータ対応機能は現行のSlurmにおいてもすでに網羅的に提供されていると考えられる。

- ドイツ 重イオン研究所(GSI)は NVIDIA V100 と AMD MI100 を混載するシステムを運用している [32]。
- フランス 国立科学研究センター(CNRS-IDRIS)[33]は世代の異なる複数種類の GPU を保有し、それぞれに専用 Partition を作成して運用している。また、NVIDIA GPU の Multi-Process Service (MPS) [17] によって、各 GPU を複数のジョブで共有する機能も利用している。
- アメリカ ジョージマディソン大学[34]は複数種類の GPU を保有し、そのうち NVIDIA A100 を Multi Instance GPU (MIG) [18] 機能を用いて複数のインスタンスに分割して運用している。
- ドイツ ジーゲン大学は、Slurm GRES プラグインの sharding 機能を用いて NVIDIA V100 を 64 個の shard に分割して運用している [35]。そのため、GPU のタイムスライスを各ジョブに割り当て、共有して利用することができる。
- アメリカ ケンブリッジ大学は、Intel Data Center GPU Max 1550 を搭載したシステムを運用している [36]。

スケジューラ レベルで1つのNVIDIA GPUを複数のジョブで共有する機能として、MPS, Time-slicing, MIG, vGPUが挙げられる [37]。上記の運用実績からもわかる通り、SlurmはNVIDIAのvGPU以外の共有機能を明にサポートしている。SlurmによるvGPUのサポートは示されていないが、MIGが利用できる場合にvGPUをサポートする必要はない。

NVIDIA GPU以外の対応状況として、AMD GPUはデフォルトでMPS相当の共有機能を利用できる[38][39]が、MIG相当の機能はAMD GPUでは明に提供されておらず、Slurmでも明にサポートされていない。また、Intel GPUにもMIG相当の機能があるが、Slurmでは明にはサポートされていない。Time-slicing [16] はベンダ非依存の汎用機能であることがSlurmの公式マニュアルに記述されており、AMDやIntelのGPUでも利用できることが期待される。

次世代計算基盤でアクセラレータ間にインターコネクトがある構成が採用された場合、複数アクセラレータを利用するマルチノードジョブの効率的な実行のためにはアクセラレータ間インターコネクトを考慮した資源割当が必要となる。SlurmではThree-dimension、Hierarchical、Blockという3種類のトポロジプラグインが利用可能であり、図 3.2.2.1のNVL (NVLink)のような構成の場合には、Blockが有効である。アクセラレータ間インターコネクトを採用し、図 3.2.2.1のNVLのようなトポロジの場合にはBlockが利用できると考えられる。それ以外のトポロジの場合にはSlurm本体もしくはプラグインの開発が必要となる。

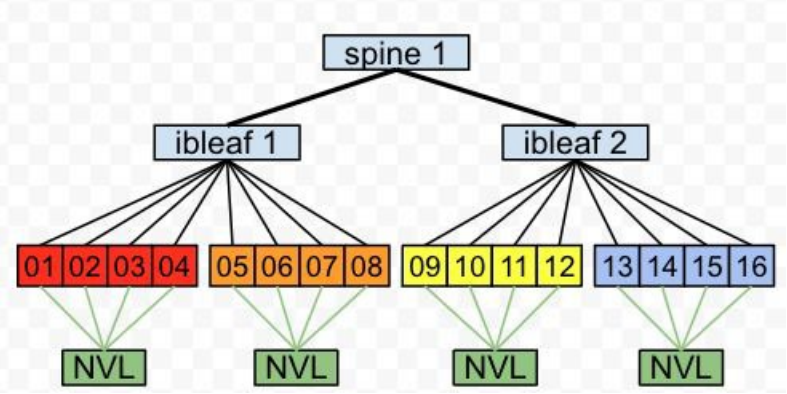


図 3.2.2.1 アクセラレータ間インターコネクトの例

（出典：https://slurm.schedmd.com/SLUG24/NVIDIA-Craig_Tierney.pdf）

アクセラレータの諸機能と連携するジョブスケジューラの機能として、以下の機能についても調査した。

- GPU ヘルスチェック機能

  多数のGPUを搭載する大規模システムにおいては、不健全なGPUを検出してスケジューリング対象から除外することが求められる。

  クラスタ環境でNVIDIA GPUを管理・制御するためのOSSのツール集としてNVIDIA Data Center GPU Manager (DCGM) [40] が提供されている。GPU搭載ノード上ではnv-hostengineというデーモンが動作し、GPUのヘルスチェックも同デーモンが担当する。ユーザはdcgmiコマンドを利用してnv-hostengineにアクセスし、操作できる。また、MIGの単位でDCGM機能を適用できる。DCGMヘルスチェック機能としては、dcgmi healthとdcgmi diagが該当する。前者は、GPUの各種パラメータ(メモリ、温度、電力等)が異常値になっていないことを一定間隔で確認する。後者はGPUを占有してテスト用ワークロードを実行し、負荷テストを行う。

  AMD GPUを管理・制御するためにAMD ROCm Data Center tool (RDC) [41] が提供されており、DCGMとの互換性を強く意識した構成となっている。ヘルスチェック機能の一部に関しては、別のツール集であるAMD ROCm Validation Suite (RVS) [42] を呼び出すことで実現されている。RDCデーモンにアクセスするrdciコマンドでは、dcgmiコマンドと比較して一部の機能が利用できない。例えば、dcgmi health相当の機能はrdciでは確認できない。また、dcgmi diag相当の機能では、DCGMで提供されている10種類の負荷テストのうち2種類(メモリテストとメモリ負荷テスト)のみが提供されている。他にもMIG相当機能への対応は記述されていないなど、RDCの機能は現時点ではDCGMに見劣りする状況となっている。

  Slurmは一定間隔でヘルスチェックプログラムを実行できるようになっており、ジョブ実行開始時や終了時に独自プログラムを実行することもできる。これらの機能によってSlurmとDCGM/RDCを連携させることができる。このため、ジョブスケジューラの観点では、連携のために必要な開発項目は見当たらない。RDCはDCGMと比較して見劣りする状況であることから、RDCやRVSの改修が必要となる可能性はある。

- GPU 情報モニタリング機能

  NVIDIAもAMDもそれぞれツール集を提供しており(上述のDCGMおよびRDC)、SlurmやPrometheusとの連携もできる。このため、これらのGPU統計情報収集機能やSlurmの標準機能で必要なモニタリングは可能であると考えている。ただし、GPU情報の監視、アラート発行、および可視化に関しても、要件次第ではプラグインの改修等が必要となる可能性はある。

- GPU 電力・クロック制御機能

省電力化のために、ジョブが割り当たっていないアイドル時にGPUを省電力状態に移行させることが望まれる。このために、クロック周波数制限機能を利用することが想定される。まず、NVIDIA GPUはアイドル時にクロック周波数を自動的に低下させる機構を持っている。また、NVIDIA Management Library (NVML) [43]で最大電力やクロック周波数を制限することが可能であり、MIG単位での設定も可能である。AMD GPUにも同様の制御関数が提供されている。これらの機能はSlurmの標準機能でジョブスケジューラと連携できる。

3.2.2.4 参考文献

[1] SLURM WORKLOAD MANAGER, HTTPS://SLURM.SCHEDMD.COM/, (参照2025/2/7)

[2] B. AKSAR ET AL., "RUNTIME PERFORMANCE ANOMALY DIAGNOSIS IN PRODUCTION HPC SYSTEMS USING ACTIVE LEARNING," IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 35, NO. 4, PP. 693-706, APRIL 2024.

[3] J. JEONG, I. BAEK, B. BANG, J. LEE, U. SONG AND S. B. KIM, "FALL: PRIOR FAILURE DETECTION IN LARGE SCALE SYSTEM BASED ON LANGUAGE MODEL," IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, DOI: 10.1109/TDSC.2024.3396166.

[4] AWESOME-WORKFLOW-ENGINES, HTTPS://GITHUB.COM/MEIRWAH/AWESOME-WORKFLOW-ENGINES, (参照2025/2/6)

[5] APACHE AIRFLOW, HTTPS://AIRFLOW.APACHE.ORG/, (参照2025/2/6)

[6] CELERY - DISTRIBUTED TASK QUEUE, HTTPS://DOCS.CELERYQ.DEV/EN/STABLE/, (参照2025/2/6)

[7] KUBERNETES, HTTPS://KUBERNETES.IO/, (参照2025/2/7)

[8] OPEN SOURCE ORCHESTRATION | PREFECT, HTTPS://WWW.PREFECT.IO/OPENSOURCE, (参照2025/2/6)

[9] GLOBUS FLOWS, HTTPS://DOCS.GLOBUS.ORG/API/FLOWS/, (参照2025/2/6)

[10] GLOBUS TRANSFER, HTTPS://WWW.GLOBUS.ORG/DATA-TRANSFER, (参照2025/2/6)

[11] GLOBUS COMPUTE, HTTPS://DOCS.GLOBUS.ORG/COMPUTE/, (参照2025/2/6)

[12] XCOMS - AIRFLOW DOCUMENTATIONHTTPS://AIRFLOW.APACHE.ORG/DOCS/APACHE-AIRFLOW/STABLE/CORE-CONCEPTS/XCOMS.HTML, (参照2025/2/6)

[13] PROCESS BIG DATA – PREFECT, HTTPS://DOCS.PREFECT.IO/V3/DEVELOP/BIG-DATA, (参照2025/2/6)

[14] DASK | SCALE THE PYTHON TOOLS YOU LOVE, HTTPS://WWW.DASK.ORG/, (参照2025/2/6)

[15] RAY, HTTPS://DOCS.RAY.IO/EN/LATEST/, (参照2025/2/6)

[16] I. S. OLMEDO, N. CAPODIECI, J. L. MARTINEZ, A. MARONGIU AND M. BERTOGNA, "DISSECTING THE CUDA SCHEDULING HIERARCHY: A PERFORMANCE AND PREDICTABILITY PERSPECTIVE," 2020 IEEE REAL-TIME AND EMBEDDED TECHNOLOGY AND APPLICATIONS SYMPOSIUM (RTAS), 2020, PP. 213-225.

[17] HTTPS://DOCS.NVIDIA.COM/DEPLOY/MPS/INDEX.HTML (参照2025/1/31)

[18] HTTPS://DOCS.NVIDIA.COM/DATACENTER/TESLA/MIG-USER-GUIDE/INDEX.HTML (参照2025/1/31)

[19] W. XIAO, S. REN, Y. LI, Y. ZHANG, P. HOU, Z. LI, Y. FENG, W. LIN, AND Y. JIA, "ANTMAN: DYNAMIC SCALING ON GPU CLUSTERS FOR DEEP LEARNING," OSDI'20: PROCEEDINGS OF THE 14TH USENIX CONFERENCE ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION, PP. 533-548, NOV. 2020.

[20] W. Chen, Z. Mo, H. Xu, K. Ye and C. Xu, “Interference-Aware Multiplexing for Deep Learning in GPU Clusters: A Middleware Approach,” SC23: International Conference for High Performance Computing, Networking, Storage and Analysis, 2023, pp. 1-14, Nov. 2023.
[21] B. Wu, Z. Zhang, Z. Bai, X. Liu and X. Jin, “Transparent GPU Sharing in Container Clouds for Deep Learning Workloads,” NSDI'23: 20th USENIX Symposium on Networked Systems Design and Implementation, 2023.
[22] F. Strati, X. Ma, A. Klimovic, “Orion: Interference-aware, Fine-grained GPU Sharing for ML Applications,” EuroSys '24: Proceedings of the Nineteenth European Conference on Computer Systems, pp. 1075-1092, Apr. 2024.
[23] Docker, https://www.docker.com/, (参照2025/2/7)
[24] Singularity Community Edition, https://sylabs.io/singularity/, (参照2025/2/7)
[25] Apptainer, https://apptainer.org/, (参照2025/2/7)
[26] Podman, https://podman.io/, (参照2025/2/7)
[27] containerd, https://containerd.io/, (参照2025/2/7)
[28] CRI-O, https://cri-o.io/, (参照2025/2/7)
[29] Prometheus, https://prometheus.io/, (参照2025/2/7)
[30] Grafana, https://grafana.com/, (参照2025/2/7)
[31] FUJITSU Software Technical Computing Suite V4.0L20, ジョブ運用ソフトウェア APIユーザーズガイド Power API, https://software.fujitsu.com/jp/manual/manualfiles/m220008/j2ul2462/02z002/j2ul-2462-02z0.pdf, (参照2025/2/7)
[32] GSI, Virgo, https://hpc.gsi.de/virgo/, (参照2024/10/22)
[33] IDRIS, The Missions and Objective of IDRIS, http://www.idris.fr/eng/info/missions-eng.html, (参照2024/10/22)
[34] George Mason University, Office of Research Computing – Wiki, https://wiki.orc.gmu.edu/mkdocs/, (参照2024/10/22)
[35] University of Siegen, Scientific Computing service of ZIMT, https://cluster.uni-siegen.de/?lang=en, (参照2024/10/22)
[36] University of Cambridge, Research Computing Services HPC Documentation, https://docs.hpc.cam.ac.uk/hpc/index.html, (参照2024/10/22)
[37] https://developer.nvidia.com/blog/improving-gpu-utilization-in-kubernetes/, (参照2025/2/2)
[38] Manos Pavlidakis et al., “Arax: a runtime framework for decoupling applications from heterogeneous accelerators”, SoCC’22.
[39] github, Does a single AMD GPU can be shared among containers? Things like MPS of Nvidia, https://github.com/ROCm/ROCm-docker/issues/62, (参照2024/11/5)
[40] NVIDIA DCGM: Manage and Monitor GPUs in Cluster Environments, https://developer.nvidia.com/dcgm, (参照2025/2/7)
[41] ROCm Data Center Tool (RDC), https://github.com/ROCm/rdc, (参照2025/2/7)
[42] ROCm Validation Suite, https://github.com/ROCm/ROCmValidationSuite, (参照

2025/2/7)
[43] NVIDIA MANAGEMENT LIBRARY (NVML), HTTPS://DEVELOPER.NVIDIA.COM/MANAGEMENT-LIBRARY-NVML, (参照2025/2/7)

3.2.3　通信ライブラリ

3.2.3.1　調査目的

本サブグループでは、広範なSDGs、Society 5.0の実現に向けた課題解決のためのプラットフォームとなる次世代大規模高速計算機において求められる通信ライブラリの機能と性能の調査を行っている。この通信ライブラリに求められる要件として、以下が挙げられる。

- シミュレーション、AI、デジタルツイン、ワークフロー実行などの多様なソフトウェアに対し、電力制約の下でデータ移動を高度化、効率化するため、特に強スケーリングやデータフローのためのソフトウェア技術
- 既存エコシステムとの親和性や国際協調を踏まえたシステムソフトウェア整備
- 幅広いアプリケーションについて、アーキテクチャ、アルゴリズムとのコデザイン
- 機械学習と第一原理シミュレーションとリアルタイムデータ処理の融合
- アプリケーションによっては従来の HPC システムとは次元の異なる高セキュリティの担保

3.2.3.2　調査内容

本サブグループでは、次世代スーパーコンピュータへの採用が想定されるネットワーク技術や様々なアプリケーションを考慮し、スケーラブルな並列計算の実現に向けて必要となる新しい通信ライブラリのインターフェイスや実装技術を調査する。

3.2.3.3　調査結果

3.2.3.3.1　集団通信ライブラリ性能調査

【調査研究の背景】

集団通信実装は一般に、アプリケーションの並列度や転送長に依存し適切な集団通信アルゴリズムが異なるばかりか、通信媒体特性および通信物理形状によって実装方式が異なる。実際、チップレット技術の進化に伴い CPU 内の大型化・複雑化が進んでいる。例えば、CPU 内においてもコア間や NUMA 間やソケット間で異なる通信の媒体特性と物理形状を有するため、以前よりこうした特性・形状に応じた組合せを可能とする階層型集団通信実装のフレームワーク自体は存在する。

しかしながら、こうした階層型のフレームワークのバックエンドとなる各階層の実装については、プロダクションレベルの高効率な実装ならびに評価につながっていない。とりわけ、近年の処理データ量の増大に伴うメモリ媒体・演算器の高速化に対して、従来からの非効率な集団通信実装が残存していることは、省電力面も含め大きな課題である。CPU か GPU かどうかに関係なく、基本的なデータ移動の抑制と、演算器をアイドルさせないための通信の媒体特性と物理形状に合致した効率的な階層実装が重要である。

ここでは、活発な開発が続くベンダ主導の UCC [1] 集団通信ライブラリを例として取り上げ、効率的なノード内の CPU 側階層実装の観点から、現状の階層化実装の評価ならびに課題の分析を実施した。

【集団通信実装の動向】

一般に、集団通信を実装するには、次の３つの方法がある: (1) 一対一通信の上での集団通信実装、(2) 既存集団通信実装の合成、(3) 集団通信のオフロード、の３つである。(1) は MPI_Isend() や MPI_Irecv() のような一対一通信実装を使って、集団通信を実装する方法である。(2) は Allreduce = Reduce + Bcast や Allreduce = Reduce-scatter + Allgather のように、既存の Reduce 等の実装を組み合わせて、新しい Allreduce 集団通信を合成するような方法である。(3) は例えば Intel PIUMA [7] のようなデバイスに Allreduce 処理の一部をオフロードするような方法である。

表 3.2.3.1 主な集団通信実装

ライブラリ	OSS 陣営 (1)	OSS 陣営 (2)
MPI 実装	Open MPI [2]	MPICH [3]
集団通信実装 (1) MPI 実装固有	tuned: ompi/ompi/mca/coll/tuned/	mpich/src/mpi/coll/
集団通信実装 (2) ベンダ実装[1]	UCC [1] - Unified Collective Communication	OFI (libfabric) [6] - `fi_collective(3)`[2]
一対一通信実装	UCX [5] - Unified Communication X	OFI (libfabric) - OpenFabrics Interface

こうした方法で集団通信を実現した、主な集団通信実装を表 3.2.3.1に示す。例えば、Open MPI では MPI 実装固有の伝統的な tuned 、プロプラエタリな HCOLL[4] 、業界標準を狙う UCC [1] の 3 つの集団通信実装が利用可能である。但し、集団通信ライブラリ UCC では、下位の一対一通信実装として OFI (libfabric) はサポートしていないこと、また同様に OFI の fi_collective では UCX をサポートしていないことに注意が必要である。

【UCC 集団通信ライブラリの概要】

UCC は図 3.2.3.1に示すように上位 Collective Layer (CL) と、下位 Transport Layer (TL) の 2 階層からなる。TL は互いに独立なトランスポートの実装からなる。例えば、NCCL (NVIDIA GPU インターコネクト向け集団通信実装)、RCCL (AMD GPU インターコネクト向け集団通信実装)、UCP (従来からの InfiniBand 等のインターコネクト向け集団通信実装) といった TL がサポートされている ([1] の Fig. 1 を参照)。CL は、最新リリースで basic と hier の 2 つの CL 実装が利用可能である [3]。但し、デフォルトで有効な CL は basic であり、hier CL を使用するには明示的に hier を有効にする必要がある。basic CL は非階層型の単純な CL であるのに対し、hier CL は階層型の CL であり、複数の TL を越えた組合せが可能である。

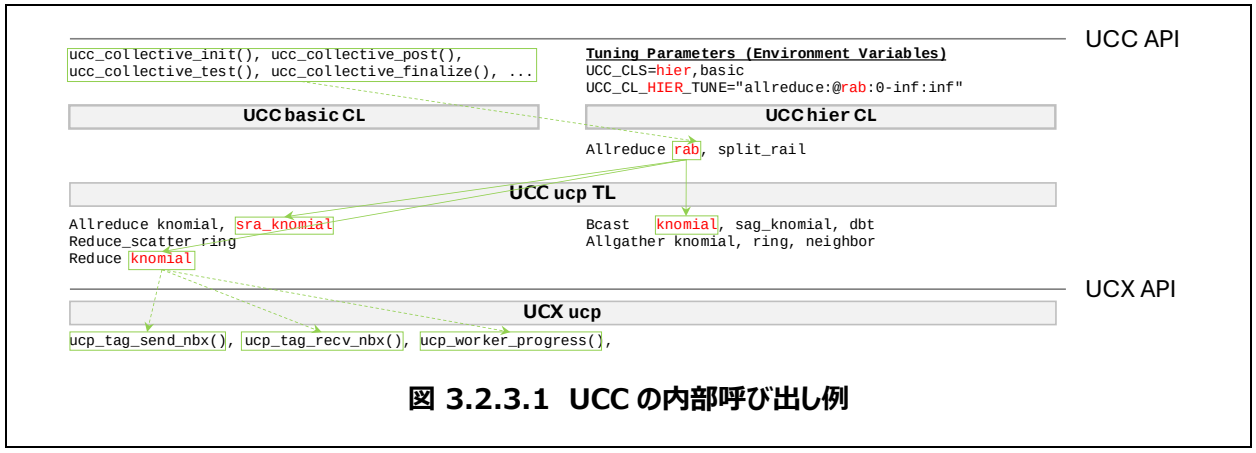


図 3.2.3.1 UCC の内部呼び出し例

【UCC 集団通信ライブラリの評価方針】

評価に当たっては、活発な開発が続く UCC 集団通信ライブラリを階層型集団通信実装の例として取り上げる。また、UCC の階層型実装 hier CL のうち、成長分野である AI でも重要な集団通信である Allreduce を調査対象として選択し、かつ、評価・分析対象は階層型の各階層の実装のうち、ノード内 (intra-node) 実装を対象とした。さらに、UCC hier CL の Allreduce 実装として、表 3.2.3.2に示す 2 つのアルゴリズム実装が現在利用可能であるが、ここでは基本的な Allreduce rab 実装を評価対象として選択した。なお、本調査環境の場

[1] 加えて、プロプライエタリな HCOLL (Mellanox Hierarchical Collective) [4] ライブラリが Open MPI 及び MPICH で利用可。

[2] `fi_collective (3)` 呼出しは現在 MPICH で未サポート。

[3] UCC の master ブランチには執筆の時点で doca_urom [8] CL が追加されている。

合、UCC hier CL Allreduce rab 実装で使用する下位の TL 実装は表中の UCC TL 列のようになる 。

表 3.2.3.2 UCC hier CL の Allreduce 実装

Name	Description	UCC TL

Allreduce `rab`	Reduce (intra-node)
	Allreduce (inter-node)
	Bcast (intra-node)
		`ucp` TL Reduce `knomial`
		`ucp` TL Allreduce `sra_knomial`
		`ucp` TL Bcast `knomial`

Allreduce `split_rail`	Reduce_scatter (intra-node)
	Allreduce (inter-node)
	Allgather (inter-node)
		`ucp` TL Reduce_scatter `ring`
		`ucp` TL Allreduce `sra_knomial`
		`ucp` TL Allgather `ring`

【UCC 集団通信ライブラリの分析】

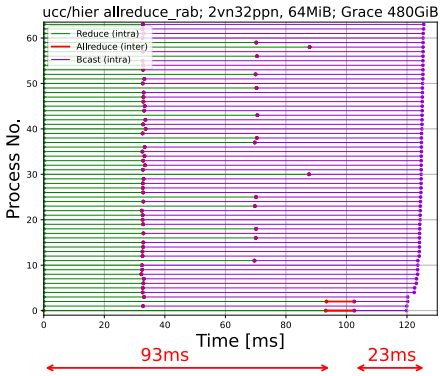
UCC hier CL Allreduce rab のオリジナル実装に対する、表 3.2.3.3に示す測定環境の下での性能測定結果を、図 3.2.3.2に示す。Reduce (intra-node) で選択される UCC TL は表 3.2.3.2より UCC ucp TL Reduce knomial であるので、木構造の k-nomial の Reduce と推測され、32 ppn では多く見積もっても 6 段程度の木構造で、1 プロセスの dst[] = src1[] + src2[] (float 型 16Mi 個の配列) 縮約演算処理に 5 ms かかると仮定しても、図 3.2.3.2で示すような 93 ms は効率的な実装とは言い難い。また、同図の Bcast (intra-node) で選択される UCC TL も UCC ucp TL Bcast knomial であるので、同様の木構造と考えられるが (実際、Bcast 完了時刻から通信段らしき形跡が存在する)、NUMA 内での Bcast として木構造を選択することが妥当かは疑問がある。

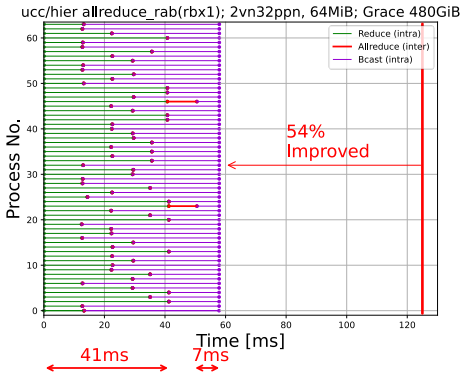
このため、Reduce (intra-node) に対しては、データ移動の抑制を狙った共有メモリ型の UCC TL 実装に、また Bcast (intra-node) に対しては、同様の共有メモリ型、かつ root 以外のプロセスがほぼ同時にコピー処理を開始する UCC TL 実装に、それぞれ置き換えた結果を図 3.2.3.3に示す。

表 3.2.3.3 動作環境・測定条件

Kind	Item	Description
諸元	CPU	NVIDIA Grace (Neoverse V2) 3.1GHz 72 コア × 2 ソケット
	メモリ	LPDDR5 240GiB × 2
測定条件	通信パターン	Allreduce 転送長 64MiB (float × 16Mi 個)
	通信プロセス配置	2 コンテナ、32 PPN[4]

[4] PPN (the number of Processes Per Node); ノード内プロセス数。





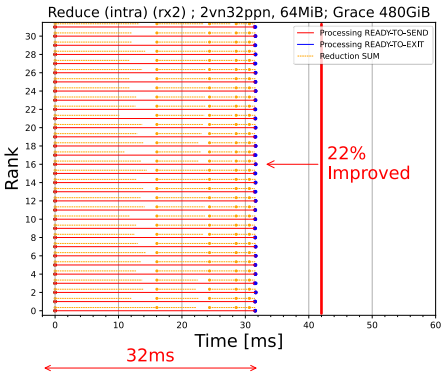
横軸に時刻 [ms] をとり、階層型 Allreduce の処理内訳を通信プロセスごと(0-63)に描画。緑がReduce (intra-node)、赤がAllreduce (inter-node)、紫がBcast (intra-node) の処理区間を示す。各(仮想)ノードの代表プロセス (ここでは2プロセス) がAllreduce (inter-node) を実行する。

図 3.2.3.2 Allreduce rab性能

図 3.2.3.3 最適化版Allreduce rab

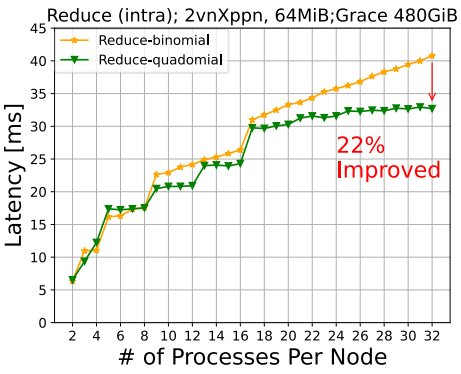
図 3.2.3.3が示すように、UCC hier CL Allreduce rab 全体のレイテンシが 54 % 改善され、ノード内の集団通信実装において、少なくとも UCC 階層型実装のうち、UCC TL のノード内実装の高効率化に向け改善の余地があることがわかった。

さらに、Reduce (intra-node) の通信段の後半 (root プロセスに近い段) では、木構造の子孫のプロセスのほとんどが Reduce を抜けてしまっており Reduce の処理に寄与しない一方で、木構造の親プロセスは典型的にはシングルプロセスで縮約演算を実行するので、ノードのメモリ帯域を使い切れていない可能性が高い。木構造の各段の頂点プロセス(縮約演算プロセス)配下の子孫プロセス全員が Reduce から抜けず縮約演算の領域を分担した実装の効果を、 図 3.2.3.4に示す。図 3.2.3.3の改善からさらに 22 % の改善が可能であることがわかった。



横軸に時刻 [ms] をとり、Reduce 内部の処理内訳をノード内 32 プロセスごと (ランク 0~31) に描画。赤はプロセスが Reduce 処理に参加している区間、青は終了同期の区間を示す。橙は各通信段の縮約演算を開始した時刻のプロット。全プロセスが最終段まで Reduce 演算処理に参加する。

図 3.2.3.4 Reduce 協調演算



縦軸に Reduce の処理時間 [ms] 、横軸に PPN (Processes per Node) をとり 2 ~ 32 と変化させ、通信段数の影響をみた。橙のプロットが Reduce binomial で、緑が Reduce quadnomial。

図 3.2.3.5 Reduce quadnomial

また、Reduce (intra-node) の通信段を減らすことによって、木構造の中間ノードの通信作業バッファ数の削減と共に、作業バッファからの Load/Store 回数を削減するために、binomial 木 (radix 2 k-nomial 相当) から quadnomial 木 (radix 4 k-nomial) への変更 (および、それに伴う縮約演算 dst[] = src1[] + src2[] + src3[] + src4[] への変更) の効果を、**図 3.2.3.5**に示す。32 ppn 場合、**図 3.2.3.3**の改善からさらに 22 % の向上が可能であることがわかった。

3.2.3.3.2　非ブロッキング集団通信の通信隠蔽効果調査

【調査研究の背景】

非ブロッキング通信は、あるプロセス間の通信と、それらのプロセス上での計算を並行して進行させることにより、見かけ上、通信に要する時間の一部を計算時間で隠蔽することを可能とする。このうち、MPI-3.0規格において採用された非ブロッキング集団通信は、多くの並列プログラムで頻出する、複数のプロセス間でのデータ複製や集約のような定型通信である集団通信について、通信時間の隠蔽を図るものである。集団通信は、計算機の大規模化に伴って所要時間が増大するため、並列プログラムのスケーラビリティ向上の手段として、この非ブロッキング集団通信が期待されている。しかし、アプリケーションや数値計算ライブラリなどで通信を隠蔽するにはアルゴリズムの変更が必要となるため、非ブロッキング集団通信を活用したスケーラビリティ向上には、通信隠蔽の効果をプログラマが正しく認識する必要がある。一方、非ブロッキング集団通信による通信隠蔽効果は、計算などの通信以外の処理と並行して集団通信アルゴリズムを推進させる機構の実装技術に大きく依存する。そこで、現在利用可能な推進機構について、通信隠蔽効果の傾向を分析した。

【非ブロッキング集団通信における推進機構実装技術の動向】

現在利用可能な非ブロッキング集団通信の推進機構実装技術としては、MPIライブラリの一対一通信を用いる基本実装、集団通信アルゴリズムの推進を担当するためのプログレススレッドによる実装、およびCPU以外の装置に集団通信を依頼するオフロード機能を用いた実装の 3 種類がある。

このうち基本実装では、通常、集団通信のアルゴリズムをMPIの非ブロッキング一対一通信の組み合わせで実装する。アルゴリズム内の各一対一通信の間には依存関係があり、それに基づいて実行可能な通信を順番に発行していく必要がある。この実行可能な通信の確認と発行はMPI関数の中でしか行えないため、非ブロッキング集団通信開始の関数（MPI_Iallreduceなど）を呼び出した後、通信完了待ちの関数（MPI_Waitなど）を呼ぶまでの間に他のMPI関数が呼ばれなければ、実際に隠蔽できる通信はアルゴリズムの最初のステップで発行できる一対一通信だけである。そのため、通信と並行して実行する計算中に適切な頻度でMPI関数（MPI_Testなど）を挿入しなければ、基本実装による通信隠蔽効果は期待できない。

一方プログレススレッドによる実装では、MPIプログラムを実行するスレッドとは別に、非ブロッキング集団通信アルゴリズムの推進を担当するプログレススレッドを起動する。これにより、計算中にMPI関数を挿入しなくても、アルゴリズムのステップを進めながら実行可能な通信を発行できるため、容易に高い通信隠蔽効果が期待できる。ただし、一つのプロセスに対して一つのプログレススレッドが起動するため、ノード内のプロセス数に応じてプログレススレッドが消費するCPU資源が増え、計算性能が低下する。これに対し、「富岳」およびその互換機ではアシスタントコアと呼ばれる技術が提供されている。これは、計算用のCPUコアとは別にOSのデーモンプロセスやファイルI/Oなどの処理用にCPUコアを用意することで、計算用CPUコアを計算に専念させて計算効率を向上させるものである。Fujitsu MPIには、プログレススレッドをこのアシスタントコアに割り当てる機能が提供されており、これにより計算性能を低下させることなく高い通信隠蔽効果を実現できる。

これに対してオフロード機能による実装では、ネットワーク装置など、ホストCPU以外の装置が提供する通信オフロード機能を用いて集団通信を実行する。例えばNVIDIA社のInfiniBandではスイッチ装置にSHARPと呼ばれる集団通信オフロード機能が用意されており、AllreduceやBroadcastなどのいくつかの集団通信を実行できる。一方、NVIDIA社のBlueFieldは、NIC装置に汎用プロセッサを搭載しており、これを利用して任意の集団通信のオフロードが可能である。また、富士通社のTofu-Dインターコネクトで提供されている Tofu Barrier機能は、24バイトまでのAllreduceやReduceなどの集団通信をNIC装置でオフロード実行可能である。これらのオフロード技術は、集団通信アルゴリズムの推進にCPU資源を必要としないため、計算性能を低下させることなく高い通信隠蔽効果を実現できる。しかし、通常、オフロード先の処理装置の資源量や性能の制約により、ホストCPUでの集団通信

に対して、参加可能なプロセス数の減少や集団通信の通信速度の低下を伴うことが多い。

【各推進機構による通信隠蔽効果の分析】

非ブロッキング集団通信の推進機構について、基本実装以外の実装技術の利用の可否は、使用するインターコネクトネットワークやMPIライブラリに依存する。しかし本報告の執筆時点で「富岳NEXT」のインターコネクトネットワークの仕様に関する情報はまだほとんど得られていない。そこで今回は、全国共同利用計算機センターで利用可能な推進機構を対象とし、表 3.2.3.4に示すそれぞれの環境で基本実装とそれ以外の実装技術の通信隠蔽効果を比較した。このうちWisteria-Oでは 144ノード（6x6x4の3次元形状を指定）、玄界では128ノードを利用した。Wisteria-Oと玄界の仕様は、それぞれ[9]と[10]の通りである。

表 3.2.3.4 通信隠蔽効果の計測環境

計算機	インターコネクトネットワーク	MPIライブラリ	推進機構
Wisteria-O (東京大)	Tofu-D	Fujitsu MPI	プログレススレッド（アシスタントコア）
玄界 (九州大)	InfiniBand	MVAPICH 3.0	プログレススレッド
		MVAPICH2 2.3.7	オフロード（SHARP）
		HPC-X 2.17	オフロード（SHARP）

通信隠蔽効果は後述の通信隠蔽率を評価指標とし、計測には米国の Ohio州立大学で開発されている OSU Micro Benchmarks 7.5のうち、MPI_Iallreduceを対象とした osu_iallreduce、および MPI_Iallgatherを対象とした osu_iallgatherを利用した。なお、SHARPは MPI_Allgatherのオフロード機能を持っていないため、osu_iallgatherの計測対象から除外した。また、メッセージサイズとして64byte、512byte、4Kbyte、32Kbyte、256Kbyteと変えながら、ノード内プロセス数 1、2、4についてそれぞれ通信隠蔽率を計測した。

ここで通信隠蔽率 (Overlap) は、通信のみを実行した場合の通信時間 Tcomm、通信と計算をオーバーラップした場合の計算時間 Tcomp、および通信と計算をオーバーラップした場合の全体の所要時間 Tall より、以下の式で計算される値である。：

$$Overlap = 100 - \frac{Tall - Tcomp}{Tcomm} \times 100$$

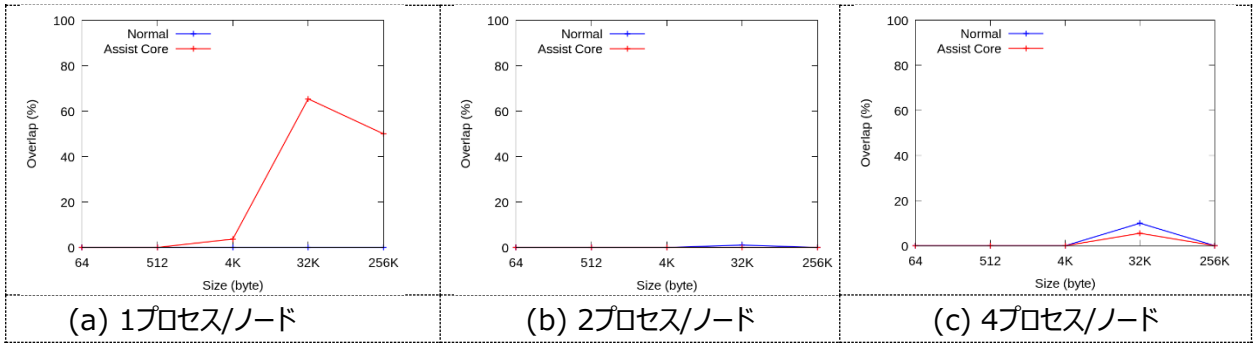
Wisteria Odyssey上で、アシスタントコアを有効にした場合のプログレススレッドによる osu_iallreduceと osu_iallgatherの通信隠蔽率を、それぞれ図 3.2.3.6 (a)、(b)、(c) および図 3.2.3.7 (a)、(b)、(c) に示す。横軸はメッセージサイズ（byte）である。また、Normalが基本実装での通信隠蔽率、Assist Coreがアシスタントコアを用いた場合の通信隠蔽率を示している。

なお、Fujitsu MPIでアシスタントコアを利用する際、動作モードを 1 から 3 の中から選択できる。モード 1 はプログラム中でプログレススレッドを動作させる区間を指定し、その区間でのMPI関数呼び出しを禁止するもの、モード 2 はプログラム中でプログレススレッドを動作させる区間を指定し、その区間でのMPI関数呼び出しを許可するもの、モード 3 はプログレススレッドを動作させる区間を明示せず、非ブロッキング集団通信関数呼び出し時に自動的にプログレススレッドが起動するものである。このうちモード 2 とモード 3 では、通信隠蔽率の向上を確認できなかった。そのため本報告ではモード 1 の結果のみを掲載する。そのため、アシスタントコアを用いた場合の計測には、OSU Micro Benchmarksのプログラム中に、プログレススレッドを動作させる区間を指示する関数

（FJMPI_Progress_start()、FJMPI_Progress_stop()）を挿入したものを用いた。

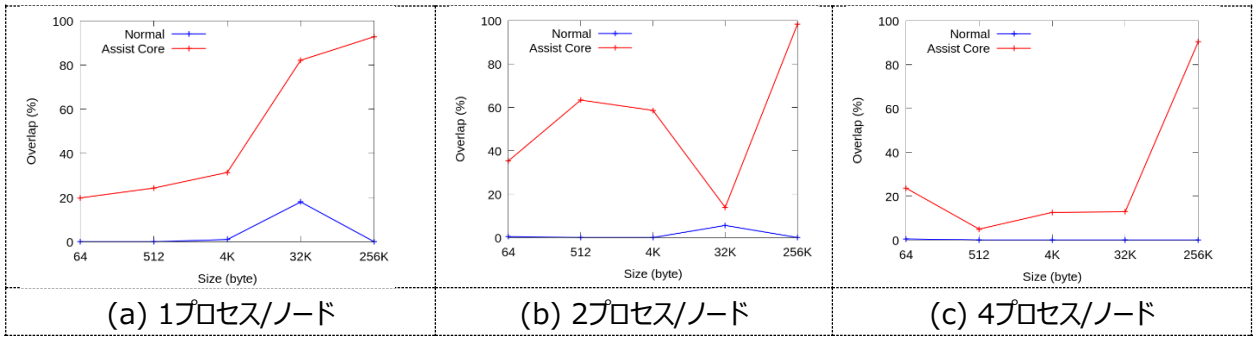
図 3.2.3.6(a)および図 3.2.3.7(a)に示す通り、ノード内プロセス数が1の場合、メッセージサイズ 32Kbyte 以上で50%以上の通信隠蔽率を達成できている。一方、図 3.2.3.6(b)、(c)に示す通り、ノード内プロセス数が２および４の場合、osu_iallreduceでは通信隠蔽率がほぼ 0%となった。これは、アシスタントコアに集団通信アルゴリズム推進のための通信処理や計算処理が集中しボトルネックとなった結果、プログレススレッドを用いた場合の集団通信の所要時間が増加し、全体の実行時間に占める通信時間が増加したことが原因であると考えられる。

これに対して図 3.2.3.7(b)および(c)に示す通り、osu_iallgatherではノード内プロセス数が2以下ではある程度の通信隠蔽率が見られ、ノード内プロセス数が4の場合でもメッセージサイズが 256Kbyte以上であれば90%程度の通信隠蔽率となった。これは、MPI_Iallgatherのアルゴリズム推進にはほとんど計算能力を必要としないことから、アシスタントコアへの処理の集中による影響が少なかったことが理由として考えられる。



(a) 1プロセス/ノード　(b) 2プロセス/ノード　(c) 4プロセス/ノード

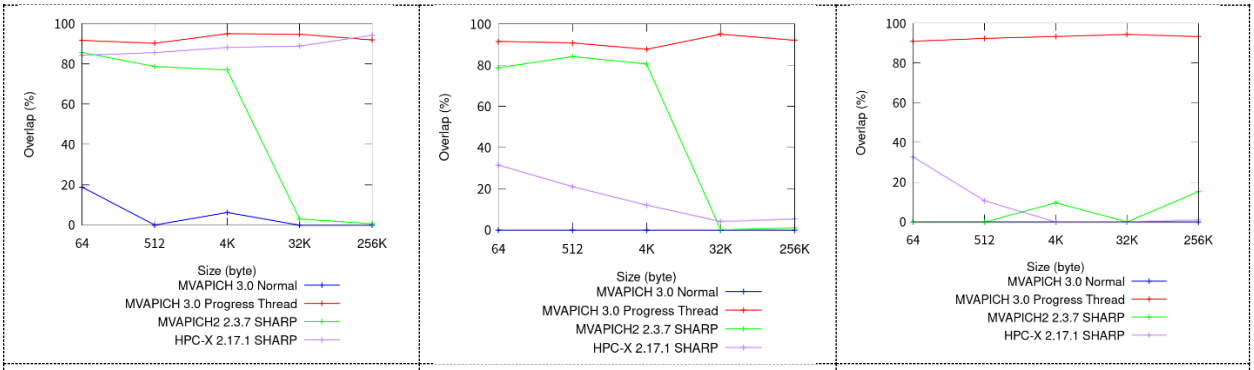
図 3.2.3.6　Wisteria Odyssey における osu_iallreduce の結果（144 ノード）



(a) 1プロセス/ノード　(b) 2プロセス/ノード　(c) 4プロセス/ノード

図 3.2.3.7　Wisteria Odyssey における osu_iallgather の結果（144 ノード）

一方、玄界上でのプログレススレッドおよびSHARPによる osu_iallreduceの結果を図 3.2.3.8に示す。基本実装としてはMVAPICH 3.0のもの（MVAPICH 3.0 Normal）を用い、プログレススレッドも MVAPICH 3.0 のもの（MVAPICH 3.0 Progress Thread）を用いた。一方 SHARPによるオフロード実装としては、MVAPICH2 2.3.7のもの（MVAPICH2 2.3.7 SHARP）と HPC-X 2.17.1のもの（HPC-X 2.17.1 SHARP）を用いた。これは、MVAPICH 3.0では玄界のジョブスケジューラ（Fujitsu TCS）上でSHARPを認識できなかったため、オハイオ州立大学のMVAPICH開発グループに相談し、旧バージョンのMVAPICH2 2.3.7に対して、Fujitsu TCS上でSHARPを認識するよう改変したものを提供してもらい、使用した。また、Open MPIをベースにNVIDIA社が開発しているMPIライブラリ HPC-Xでも SHARPによる非ブロッキング集団通信のオフロードが可能であったことから、比較のため利用した。



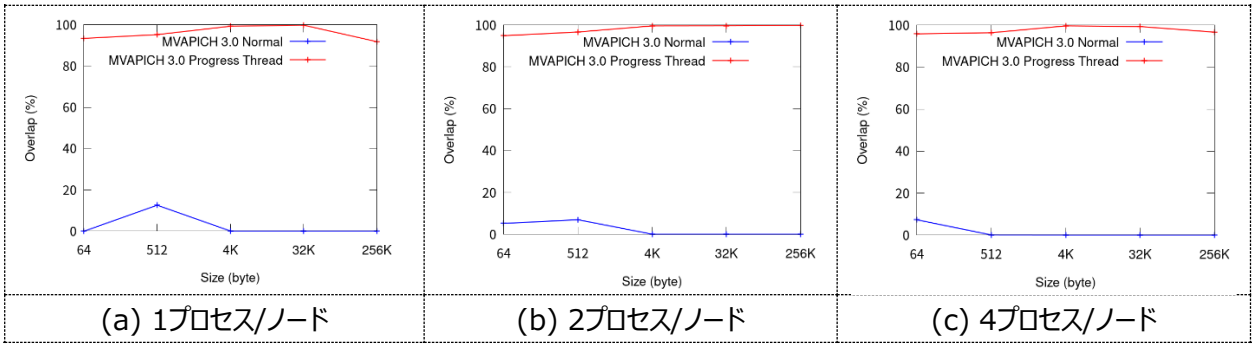
(a) 1プロセス/ノード　(b) 2プロセス/ノード　(c) 4プロセス/ノード

図 3.2.3.8　玄界における osu_iallreduce の結果（128 ノード）

MVAPICH 3.0のプログレススレッドでは、ノード内プロセス数によらず90%以上の通信隠蔽率が得られている。OSU Micro Benchmarksでは計算部分がシングルスレッドで実行されるため、計算に使用するCPUコア数はプロセス数と同じである。また、計測に用いた玄界はノード内のCPUコア数が120コアであるためコア数に余裕があったことから、プログレススレッドが計算性能に影響を与えることなく、計算と通信を効率良くオーバーラップ出来ていることがわかる。

一方SHARPによるオフロード実装のうちMVAPICH2 2.3.7では、ノード内プロセス数が 2 以下、かつメッセージサイズが4Kbyte以下の条件で、80%以上の通信隠蔽率を達成できている。一方 HPC-X 2.17.1では、ノード内プロセス数が1の場合は計測対象のすべてのメッセージサイズで80%以上の通信隠蔽率を達成できている。しかしノード内プロセス数が 2以上になると、ほとんど通信隠蔽できなかった。ノード内プロセス数やメッセージサイズが増えると通信隠蔽効果が低下する原因について、MVAPICHの開発グループ、およびNVIDIA社に問い合わせたところ、InfiniBandスイッチの集団通信オフロード機能のために用意された計算資源の性能及び容量の限界によるものであるとの回答があった。また、MVAPICH2 2.3.7とHPC-X 2.17.1で通信隠蔽率の傾向が違うのは、非ブロッキング集団通信の実装アルゴリズムの違いによるものであると考えられる。

最後に玄界上でプログレススレッドを用いたosu_iallgatherの結果を図 3.2.3.9に示す。MVAPICH 3.0のプログレススレッドにより90%以上の通信隠蔽率が達成できている。これはosu_iallreduceの場合と同様に、プログレススレッドが計算性能に影響を与えることなく、計算と通信を効率良くオーバーラップ出来ているためであると考えられる。



(a) 1プロセス/ノード　(b) 2プロセス/ノード　(c) 4プロセス/ノード

図 3.2.3.9　玄界における osu_iallgather の結果（128 ノード）

今回の計測結果より、アシスタントコアによるプログレススレッドとSHARPによるオフロードでは、特にノード内プロセス数が 4以上では通信隠蔽の効果を得るのが困難であることが分かった。一方、通常のCPUコアによるプログレスス

レッドでは、ノード内プロセス数が4の場合でも高い通信隠蔽効果が得られている。しかし、実アプリケーションではノード内のほとんどのCPUコアを使用するため、プログレススレッドによる計算性能の低下が予想される。その結果、非ブロッキング集団通信の利用により通信隠蔽はできるものの、計算時間が長くなり、アプリケーションの性能向上に寄与しない、という状況が予測される。

この解決策として、3.2.3.3.1節で議論した階層型集団通信アルゴリズムの設計において、ノード間通信に参加するプロセス数を低減することや、複数のプログレススレッドを一つのCPUコアに割り付けて計算性能の低下を抑制すること、などが考えられる。

3.2.3.4 開発項目

【集団通信ライブラリ性能調査】

業界標準を狙う有力な集団通信ライブラリである UCC における CPU 側のノード内の実装を、処理効率の観点から評価・分析を行った。その結果、UCC は階層型集団通信実装のフレームワークとして豊富な機能を有している一方で、バックエンドのノード内実装はデータ移動の抑制や媒体特性等の活用面でまだまだ改善の余地があることがわかった。また、UCC 集団通信ライブラリ全体をオフロードする試みも進行していると見られ、計算と通信のオーバーラップの新しい局面にも注意する必要がある。

- 強スケーリング実現に向けた開発要素
  計算と通信のオーバーラップの向上のために、一対一通信のオフロードに限らず、DOCA UROM のような集団通信のオフロードフレームワークの動向にも注視する必要がある。
- 業界標準インターフェイスに向けた開発要素
  UCX のような一対一通信ライブラリのみならず、業界標準を狙う UCC 等の集団通信ライブラリを一体として検討する必要がある。
- 高効率運用に向けた開発要素
  高効率な集団通信ライブラリに向け、汎用の集団通信実装に加え、通信の媒体特性や物理形状ごとに特化した実装を検討すると共に、これら実装をアプリケーションから簡便に追加・変更・合成できる仕組みを検討する必要がある。

3.2.3.5 参考文献

[1] M. G. Venkata, et al., "Unified Collective Communication (UCC): An Unified Library for CPU, GPU, and DPU Collective," IEEE Symposium on High-Performance Interconnects (HOTI) 2024, 2024

[2] https://www.open-mpi.org/

[3] https://www.mpich.org/

[4] https://docs.nvidia.com/networking/display/hpcxv2210/hcoll

[5] P. Shamis, et al., "UCX: an open source framework for HPC network APIs and beyond," IEEE Symposium on High-Performance Interconnects (HOTI) 2015, 2015

[6] P. Grun, et al., "A Brief Introduction to the OpenFabrics Interfaces - A New Network API for Maximizing High Performance Application Efficiency," IEEE Symposium on High-Performance Interconnects (HOTI) 2015, 2015

[7] K. Lakhotia, et al., "Accelerating Allreduce With In-Network Reduction on Intel PIUMA," IEEE Micro Volume 42, Issue 2, 2022, 2022

[8] https://docs.nvidia.com/doca/sdk/doca+urom/index.html

[9] HTTPS://WWW.CC.U-TOKYO.AC.JP/SUPERCOMPUTER/WISTERIA/SYSTEM.PHP
[10] HTTPS://WWW.CC.KYUSHU-U.AC.JP/SCP/SYSTEM/GENKAI/

3.2.4　I/O・ストレージ・ファイルシステム

3.2.4.1　調査目的

次期ストレージシステムアーキテクチャについてのフィジビリティスタディを行う。

3.2.4.2　調査内容

富岳アプリケーションにおけるストレージの利用状況についての調査結果をまとめ、次期ストレージシステムアーキテクチャについての検討、およびI/Oベンチマークプログラムの整備を行う。

3.2.4.3　調査結果

調査結果として、富岳アプリケーションにおけるストレージ利用状況の調査結果、次期ストレージシステムアーキテクチャおよびI/Oベンチマークプログラムについて以下にまとめる。

3.2.4.4　富岳アプリケーションにおけるストレージ利用状況の調査結果のまとめ

これまでの大規模アプリケーションユーザに対するヒアリングの結果、ストレージに関して以下のような問題があることが判明した。

(1) I/Oに関して多くのユーザが不満を持ち、そのワークアラウンドとしてさらに多大な負担を課している
I/O処理においてエラーが発生してそれまでの多くの時間費やしてきた計算が無駄になってしまう、またI/O処理に多大な時間がかかってしまう問題が発生している。これはアプリケーション性能とは見合わないI/O性能しか提供しておらずI/O性能の限界に達してしまっているためである。そのため、ユーザはそのI/O性能の限界に達しないよう、ワークアラウンドとしてアプリケーションの改変を強いられている。これはユーザの本来の研究においては不要なことである。必要なI/O性能があればユーザはアプリケーション性能の向上に時間を費やすことができ、サイエンスの発展を促進することができる。

(2) 持続可能なシステムではない
ストレージシステムの開発が終了してしまっている。そのため、ユーザのフィードバックがあってもソフトウェアの改修による対応ができない。システムソフトウェアは持続的に改修を行い、問題を解決してバージョンアップを行うことは必須である。

(3) ストレージシステムの要件がハードウェア仕様に活かされていない
これまでのフラグシップシステムにおいて、ストレージシステムのハードウェアは単純にバンド幅と容量のみが検討され、ストレージシステムのソフトウェア要件についてはあまり考慮されていなかった。富岳においては一次ストレージシステムとしてLLIOが開発されたが、計算ノードの16ノードのうち1ノードにSSDを搭載しただけのI/OノードでI/O処理を行うことが求められ、I/O処理を行うためのメモリの圧倒的な不足等によりストレージシステムの限界性能が低くなっている。このようなことが起こらないように、ストレージシステムのシステムソフトウェアの要件もハードウェア仕様に活かしていくことが重要である。

次期ストレージシステムにおいては、ユーザにI/O性能やストレージ容量を意識させないシステム作りが重要である。特に、膨大なデータを扱う昨今のAIスパコンにおいては高いI/O性能をもったストレージが標準となっている。

3.2.4.5　次期ストレージシステムアーキテクチャ

ストレージ性能については、演算性能、メモリサイズ、ノード数などとのバランスが重要である。特に、全システムを用いた全プロセスからのアクセスにも耐えられるメタデータ性能、アクセス性能が求められる。ファイルシステムのアクセスパターンにおいて性能低下が起こらないようにする必要がある。並列アプリケーションから単一のファイルをアクセスす

るシングルシェアードファイル（SSF）アクセスパターンは多くのアプリケーションで利用されるため十分なバンド幅が求められる。また、フラグシップシステムにおいては他に同様の規模の試験環境を準備することが難しく、テストが困難である。これを改善するため、例えばストレージ仮想化により、用途や利用方法に応じたストレージシステムの柔軟な再構成を可能とする、また柔軟な運用を可能とするストレージ基盤の構築が望ましい。また、ストレージシステムの構成要素については、最先端の技術をいち早く導入し、求められる性能と安定性を満たすことが求められる。

富岳NEXTで想定されるシステムの総メモリ容量はCPU、GPUともに14PBである。14 PBのデータを1分で読み書きするためには250 TB/sほどの性能が求められる。この要求性能はSSFアクセスパターンにおいても同様である。必要な容量としてはメモリサイズの300倍とすると4.2 EBとなる。また、各計算ノードで100プロセス実行されるとすると、計35万プロセスからのアクセスとなる。そのアクセスを問題なく処理するためには100M IOPSほどのメタデータ性能が必要となる。なお、この性能は、例えばファイル作成等のメタデータ操作を1秒間に1億（100M）回実行可能な性能である。これらの性能要求に対し、一つのストレージシステムで対応するのは現実的ではないため、階層的なストレージシステムの構成を検討する。

ストレージ構成は3階層で検討すると、それぞれ以下のような要件とすることができる。

表 3.2.4.1 ストレージ階層の要件

ストレージ階層	容量	性能要件と特徴
第 1 階層	40 PB	100M IOPS のメタデータ性能 SSF における 250 TB/s のバンド幅を持つ安定したストレージシステム
第 2 階層	450 PB	10M IOPS のメタデータ性能 SSF における 50 TB/s のバンド幅を持つ安定したストレージシステム
第 3 階層	4.2 EB	広域で安全、高速に全国スパコンからアクセス可能なストレージシステム

第1階層では、アプリケーションからのアクセスを直接処理する必要があり、100M IOPSおよびSSFにおいて250 TB/sの性能が求められる。一方で、容量としては一時的に少なくとも総メモリ容量の3倍ほどの容量のデータを保持できればよいと考えられるため40 PBとなっている。主にこの第1階層は第2階層のストレージシステムのキャッシング層として機能することを想定している。第2階層では、アプリケーションからの直接のアクセスではなく、主に第1階層からのアクセスを想定し、10M IOPSおよび50 TB/sの性能を要件としている。容量は第1階層の約10倍の450 PBとしている。第3階層は、容量を重視し、4.2 EBの容量を要件としている。第3階層は、富岳NEXTだけではなく、HPCI共用ストレージとして全国のスパコンから高速にアクセス可能となることを想定している。研究で必要となるデータサイズは増大の一方であり、また実験、計算データの保全も求められており容量は重要である。

次期ストレージシステムでは2030年頃に利用可能なデバイスを想定し、ネットワークはPCIe Gen7の16レーンで200 GB/s、SSDはPCIe Gen6の4レーンで30 GB/s、60～240 TB、HDDは50 TBとする。

第1階層においては、計算ノードにローカルストレージを搭載することにより実現する。ノード数が3,500ノードとすると250 TB/sを達成するためには各ノードにPCIe Gen6の4レーンで接続されるストレージが3～4基搭載されていればいい。また、計算ノードのローカルストレージを利用する並列キャッシングファイルシステムを構成するためには、計算ノードにそのために必要なメモリサイズ、および計算CPUコアを確保することが必要となる。ほかの構成としては、I/Oノードを別に設ける方法もある。その場合、I/OノードにはPCIe Gen6の4レーンで接続されるストレージを16基

搭載すれば計算ノード数の1/4、32基搭載すれば1/16のI/Oノードが必要となる。また計算ノードとI/Oノードの間は少なくともPCIe Gen7の16レーンのネットワークで接続されていれば十分である。ソフトウェアとしては、第2階層のキャッシングを行うキャッシングファイルシステムであることが求められる。キャッシングファイルシステムについては現時点で製品となっているものは存在しないが、既存の小さいファイルアクセスにおける性能低下問題を解決するためには国産のCHFS/Cacheのようなアーキテクチャが有望と考えられる。

第2階層においては、1.8 PB、200 GB/sのファイルサーバを250ノード準備することにより、450 PB、50 TB/sを達成することができる。必要に応じてメタデータサーバが32ノード程求められる可能性もある。ソフトウェアとしては、現在大きな実績があり、大規模システムにおける問題への持続的な対応がなされているLustreが有望と考えられる。10M IOPSのメタデータ性能、およびSSFで50 TB/sの性能についてもLustreの開発ロードマップに存在している。ほかの可能性としてはDAOSが考えられる。DAOSはPost Lustreとして開発が進められていたが、IntelのOptane事業の撤退によりアーキテクチャの大幅な変更が必要となっている。現在開発はDAOS Foundationが主体となっている。米国のスーパーコンピュータAuroraにおいて大規模システムにおける問題を解決中であるが、AuroraはOptaneを利用している。Optaneを利用しないDAOSについてはまだ開発途中であり、大規模システムでは利用実績がない。

第3階層においては、容量が重要であり、例えば60 PBのストレージサーバを70台準備することにより4.2 EBの容量を達成することができる。加えて、SSDを搭載したメタデータサーバについては6ノード程必要である。ソフトウェアとしては、現在HPCI共用ストレージで実績があり、大規模システムにおける問題への持続的対応がなされているGfarmが有望である。広域で安全、高速に全国のスーパーコンピュータからアクセス可能な安定したストレージシステムであり、OAuth 2.0/OpenID Connectにより、認証、認可が可能となっている。S3 I/F、Nextcloudに対応し、パブリッククラウドとも連携が可能である。また、gfptarにより、多数ファイル、大規模ストレージへの対応が完了している。

3.2.4.6 I/O ベンチマーク

次期ストレージシステムの性能評価を行うため、富岳においてI/O性能が問題になっているアプリケーションについてベンチマークプログラムを準備した。

3.2.4.6.1 Athena++

宇宙物理学用（輻射）磁気流体シミュレーションコードであり、国際協力により開発を進めている。典型的な空間時間2次精度のHLLD+CT法を用いたMHDソルバを用いている。ベンチマークプログラムの入力ファイルは、1プロセスから32,768プロセスまで2x2x2倍間隔で計算セットアップが準備されている。Athena++のファイル出力部はOpenMP並列化されていないため、入出力については、ハイブリッド並列はあまり意味をもたない。初期条件生成コード（src/pgen/blast.cpp）はほぼ一様な定数が入った初期データを生成するため、（HDF5のzlib/szip等）圧縮を行うとファイルサイズが小さくなってしまうため注意が必要である。そのため、圧縮なしで用いるか、初期条件を乱数で置き換える必要がある。出力は、格子を、POSIXによるVTK形式（プロセスごとのファイル）、HDF5形式（SSF）で出力し、リスタートファイル（rst）をMPI-IOで出力する。用意されている入力ファイルでは、VTK、HDF5、rstファイルが1セットずつcurrent working directoryに出力される。Athena++のI/Oベンチマークコードはhttps://github.com/tsukuba-hpcs/athena/tree/benchにある。ライセンスはBSD 3-Clauseライセンスである。

3.2.4.6.2　R2D2

R2D2は太陽のための輻射磁気流体コードであり、中央差分と非線形の人工粘性を用いて高精度かつ安定的に解いている。R2D2では、太陽表面付近では現実的に輻射輸送を解き、太陽深部では微小擾乱を正確に解く手法を取ることで、太陽内部から表面までを一貫して解くことができる。出力はMPI-IOによりSSFアクセスパターンで書き出す。本ソフトウェアは公開ソフトウェアではなく再配布は禁じられており、共同研究者のみが使ってよいというルールになっている。I/Oベンチマークコードはストレージ性能評価のために開発者の許可を得てhttps://github.com/tsukuba-hpcs/R2D2-IOにおいてある。

3.2.4.6.3　NICAM

NICAMは非静力学正20面体格子大気モデルであり、全球雲解像モデルGCRM（Global Cloud Resolving Model）として使用する。NICAMのシミュレーションではプロセスごとにファイルを書き出すが、富岳においてはその後の緯度経度データに変換するI/O処理が問題となっているため、その処理をI/Oベンチマークとして準備した。入力データとしては、京コンピュータのグランドチャレンジにおける実験データにおけるglevel-9（圧縮後250 MB）～glevel-13（圧縮後43 GB）が準備されている。データサイズが大きいため公開レポジトリにはおいていない。

3.2.5　数値ライブラリ

3.2.5.1　調査目的

数値計算ライブラリは、科学技術アプリケーションプログラムを中心として共通に使われる、典型的な数値アルゴリズム群をコード化しライブラリの形にまとめたものである。高度な数学やプログラミング技術が投入され、実現されており、通常、利用者が短期間に作成することは困難である。利用者からは数値計算ライブラリが提供する高性能かつ高信頼性を根拠に、過去現在そして将来にわたって継続的な利用が期待されている。数学的な分類として各種存在するが、数値計算の最も基本的な機能として以下の４手法は必須である。

- 線形方程式（連立一次方程式）
- 固有値問題（特異値計算も含む）
- 高速フーリエ変換
- 乱数生成

数値計算ライブラリがアプリケーションプログラムの中核的な部分に使われていることから、その性能がアプリケーションに大きく影響する。性能の指標はさまざま存在するが、例えば実行時間（速度）、消費電力、計算精度などがある。また、大規模な計算では並列化の程度によって、利用する計算機アーキテクチャや問題サイズへの制限、並列化効率にも影響する。さらに、昨今では、旧来のFortranだけではなくC、C++に加えてPythonやJuliaなどの新言語から呼び出し可能かなど言語バインディングの問題もある。さらに、バックエンドのランタイムモデルとの親和性を考慮した設計になっているかなど、最新のプログラミング言語との連携がなされているかの観点、ひいては、保守性・継続性の面からも調査が必要である。

3.2.5.2　調査内容

本数値ライブラリ調査研究サブグループでは、2028年頃に登場する次世代スパコンシステムにおいて、現在の「富岳」級システムでの動向調査を進め、その延長線上の技術外挿ならびにいくつかの技術動向予測を加え、次世代スパコンにおける数値計算ライブラリの様相についてまとめる。

2022年度報告書では、2022年8月に発足以降進めてきた、「富岳」開発者としての数値計算ライブラリ開発時の振り返りを通じて、現状の問題点や次期システムへの引継ぎ点をまとめた。さらに、2023年度には既存の数値計算ライブラリの動向調査の継続、各種ライブラリに求められる機能とその対応状況など、様々な観点からの比較検討の中間報告をまとめた。2024年度は最終年度として未調査部分を含め本調査の主たる目的となる実際の次世代計算基盤開発にむけた提言に向けた調査を進めた。主に以下の3項目について調査を進めそれらについて、詳細をまとめる。

- 既存ライブラリの動向調査・機能調査結果
- 数値ライブラリの性能評価
- 開発項目選定に向けた調査検討

3.2.5.3　調査結果

3.2.5.3.1　既存ライブラリの動向調査・機能調査結果

数値計算ライブラリの機能は2022年動向調査の中でも主に10階層に分けて進めてきた。本年度はその中の未調査内容についてさらに踏み込んだ継続調査を進めた。数値ライブラリが扱うデータ形式を「密行列」もしくは「疎行列」で分けたとき、前者は昨年度の調査でかなりの部分を調査しており、昨年度は後者を中心にした調査結果をまとめた。今年度は未調査部分、特に米国アルゴンヌ国立研究所に設置されたAuroraシステムが採用しているインテル社の開発環境oneAPIに含まれるSYCLを基盤とする数値計算ライブラリについて、当該内容を以下にまとめ

る。

3.2.5.3.1.1　SYCL の概要と処理系

SYCLは単一のソフトウェア記述で複数のアーキテクチャプラットフォームでの実行形式を作成できるC++をベースとるすコンパイラとランタイムの環境である。OpenCLのデータパラレル並列実行モデルを元に開発が始まり、2015年のver.1.2が初出のバージョンであり、ver.2.2は2016年にC++14とOpenCL 2.2をターゲットとして導入された。SYCL2020が最新の規格でありC++17とOpenCL 3.0をターゲットとしている。ホストCPUと演算デバイスのメモリモデルはBufferであったが、2020年よりホストとデバイスでのメモリの確保とデータのコピーを明示するUnified Shared Memory(USM)が導入された。OpenGLやOpenCL、Vulkan、グラフィックスと並列計算のための中間言語SPIR-Vの規格作成で知られるKHRONOS groupがSYCLの仕様の策定を行っている.初期のバージョンはOpenCLの拡張として構築されたため、C++のクラスのヘッダの取り込みは#include <CL/sycl.hpp>でオブジェクトクラスはcl::syclであったが、現在のバージョンではC++のクラスのヘッダの取り込みは#include<sycl/sycl.hpp>オブジェクトクラスはsyclとなっている。

SYCLはヘテロジニアスな演算デバイスにデータをオフロードして演算を実行するアーキテクチャを想定しており、CPUに結合されているGPUあるいはFPGA、さらにメモリを共有する他のCPUコアで演算を実行するために単一のコードから実行形式をコンパイル時に生成するAOT、あるいはデバイスにコードを送り実行時にデバイスでコンパイルを行なうJITを利用する。

SYCLの処理系には次のものがある。

- Intel の実装による Data Parallel C++ (DPC++)は LLVM ベースのコンパイラであり、数値演算ライブラリの MKL やタスク並列のための C++テンプレートライブラリ TBB と共に、oneAPI のプログラミングインターフェースを提供する。Intel Xeon マルチコア CPU (x86_64)、Xe GPU(spir64)向けのバリナリーを生成する。
- AdaptiveCpp (hipSYCL/OpenSYCL)はオープンソースコミュニティーによる開発であり CUDA、ROCm と OpenMP および Intel GPU をサポートする。また OpenCL SPIR-V デバイスへのバックエンドを持つ。
- copdeplay 社により oneAPI DPC++の NVIDIA GPU あるいは AMD GPU へのバックエンドとランタイムが提供されている。
- 東北大学 サイバーサイエンスセンター 滝沢研究室による neoSYCL は NEC ベクトルプロセッサ SX-Aurora TSUSBASA のベクトルコンパイラ向けに、SYCL の計算カーネルを分離して抽出し、Buffer メモリモデルによって記述されたメモリの読み書きの依存関係からホストとデバイス間でのメモリ転送を指定する。
- triSYCL は SYCL の研究プロジェクトであり、SYCL の規格策定のフィードバックに利用されてきた。C++23 の記述を用い、SPIR デバイスをターゲットとし、OpenMP と TBB による CPU での並列実行を実現している。

3.2.5.3.1.2　SYCL のメモリモデルと並列実行モデル

ヘテロジニアス環境では演算するデータがホストとデバイスのメモリのどちらにあるかを何らかの形でユーザが記述する必要があるが、SYCLでは3つの方法が用意されている。またヘテロジニアスなハードウェア構成で効率よく演算を行なうためにはホストでの演算、デバイスでの演算、およびホストとデバイス間でのデータの転送をできるかぎりオーバーラップして実行することが必要となる。このためSYCLでは配列データに関する演算操作をタスク、sycl::eventとして記述して、その実行順序あるいはタスクの依存関係をキュー、sycl::queueを生成し、ランタイム環境によるタスクコントロールで非同期実行を実現している。SYCLには次の3種類のメモリモデルが用意されている。

- Unified Shared Memory (USM)のうち、デバイスでのメモリ確保を行ない、そのポインターをホストから参照するもの。sycl::malloc_device<double>(int N, sycl::queue Q) により N の大きさの倍精度浮動小数点 double 型の 1 次元配列がデバイスに確保される。ホストでの配列の確保は通常の C++のメモリ確保を用い、ホストとデバイスの間のデータ転送は、キューに登録するタスクの sycl::event オブジェクトを生成した際に参照できる sycl::handler のメンバ関数 memcpy()あるいは copy()を用いる。このメモリモデルを用いる場合は、タスクの依存関係、あるいはタスク間の同期操作はユーザが直接明示する必要がある。
- USM のうち、デバイスからも参照できるホストメモリとホストとデバイスの双方からアクセスできる共有メモリを利用するもの。ホストメモリは sycl::malloc_host<double>(int N, sycl::queue Q)により確保される。デバイスでのメモリの確保は sycl::malloc_shared<>()を用いる。デバイスでの演算は後述する C++のラムダ式を用いた parallel_for 文によって実行するが、ループ内部でホストメモリをデバイスがアクセスする前にデータがコピーされる。sycl::event オブジェクトとして記述される演算が終了した後の同期操作の後にはデバイスに確保されているメモリは共有メモリとしてホストからアクセスができる。
- Buffer/Accessor を用いるもの。sycl::buffer により通常の std::array クラスによる一次元配列からインスタンス化を行うか、sycl::range{}により一次元または二、三次元の配列のオブジェクトのインスタンス化をおこなう。デバイスからは sycl::accesor()を用いて読み書きのモードを指定してローカルなオブジェクトにマッピングをおこなってからアクセスする。ホストからは sycl::host_accesor()を用いて読み書きのモードを指定する。sycl::event により、デバイスでのデータの初期化、デバイスでの演算、ホストからのデータのアクセスをタスクとして記述するが、accessor の読み書きモードの指定からタスクが利用するデータの RAW、WAR、WAW の依存関係を決定することができるため、キューにタスクの依存関係を指定する必要はない。

3.2.5.3.1.3 C++ の parallel_for 文と lambda 式によるデータパラレル演算

演算カーネルはsycl::handlerのメンバ関数parallel_for()により、並列実行されるループでの添字を受け取って記述される。

```
int N;  sycl::queue Q;
float *a_device = sycl::malloc_device<float>(N, Q);
sycl::event ev = Q.submit([&] (handler &h) {
  h.parallel_for(range<1>{N}, [=](id<1>, i) {   a_device[i] =  ...  }) };
```

GPUストリームプロセッサあるいはCPUのSIMD演算では階層構造をもつ多次元の配列のループ演算を用いることができる。sycl::range<2>{100, 100}は二次元の100x100の添字集合を表し,

```
paralallel_for (range<2>{N, N}, [=](id<2> idx) { int i = idx[0]; int j = idx[1]; });
```

はその添字集合を分割して並列実行する際の添字をidx[0], idx[1]に割り付ける。

sycl::nd_rangeはグローバルな添字集合に対してワークグループのサイズを指定してparallel_forの並列実行の割り付けを行なう。

```
paralallel_for (nd_range<2>{range<2>{N, N}, range<2>{B, B}}, [=](id<2> idx) {
  int i = idx[0];  int j = idx[1]; });
```

上の例ではnd_rangeはN x Nの二次元の添字集合全体から2 番目のrange<2>で示されるB x Bの大きさの work_groupを生成する。NをBの倍数とするとき、入れ子の二重ループとして並列実行できる。

```
range<2> num_groups{N/B, N/B}; range<2> group_size{B, B};
h.parallel_for_work_goup(num_groups, group_size, [=](group<2> grp) {
  int ib = grp.get_id(0); int jb = grp.get_id(1);
  grp.parallel_for_work_item([&](h_item<2> it) {
    int i = ib * B + it.get_local_id(0); int j = jb * B + it.get_local_id(1); }); });
```

ここでCUDA環境と同様にwork_groupの中ではデバイスはローカルなワーク配列にアクセスすることができ、sycl::group_barrier(grp)を用いて同期を取ることができる。

groupの中にハードウェアに依存したsub_group が設定される。nd_item itで示される添字のオブジェクトに対して sycl::sub_group sg = it.get_sub_group() により、sub_groupが取得でき、そのsub_groupの大きさは get_local_range()、またsub_groupの中での添字はget_local_id()で得られる。sub_groupの中でのローカルな同期を取ることができる。sub_groupはGPUストリームプロセッサのwarpサイズやCPUのSIMD長にあわせて生成される。

3.2.5.3.1.4 oneMKL での BLAS/Lapack ライブラリ

密行列向けの行列演算のBLASと線形問題ソルバのLapack及び、疎行列の行列演算のsparseBLASがCPUの x86_64アーキクチャとGPUのSpir64アーキテクチャ向けに提供されている。疎行列直接法ソルバのPARDISOや経路積分にもとづく疎行列の固有値問題ソルバFEAST、あるいはRCIインターフェイスによる疎行列向けGMRES/CG反復法ソルバはoneAPIのフレームワークでは提供されていない。BLAS ルーチンはUSMとBufferとも密行列は一次元配列と整合寸法を用いて表現される。Fortran タイルの列優先形式のoneapi::mkl::blas::column_majorがnamespace oneapi::mkl::blasのデフォルトとなっており、行優先形式はoneapi::mkl::blas::raw_majorを明示する必要がある。LapackのルーチンはnetlibのCインターフェイス実装のLAPACKEと異なり、列優先形式のみが利用できる。USM データで保持された行列とベクトルを扱うBLAS ルーチンはタスクsycl::event の依存関係を明示する必要があるため、gemm()関数など全ての BLASルーチンは sycl::eventを戻り値にとるよう定義されており、内積を計算する BLAS 1のdot()関数は戻り値をCBLAS で利用する引数の直後にポインター渡しで受け取る仕様になっている。Bufferデータの場合は依存関係を記述しないため戻り値はvoidである。BLASルーチンの名前は数値データ型の精度と実数か複素数を表す接頭字s、d、c、zを有していたが、oneMKL版ではdgemm()の代りにgemm()となる。特にgemm()は数値データ型がC++ のTemplateを用いて記述されており、行列 A、B、Cそれぞれの型はテンプレートTa、Tb、Tcとスカラalpha、betaはテンプレート Tsの型を持つ。単一精度演算ではTa=Tb=Tc=Ts となり、半精度のsycl::half、単精度 float、倍精度 double、単精度複素数std::complex<float>、倍精度複素数std::complex<double>をとる。半精度と整数と単精度の間の混合精度演算は sycl::half、oneapi::mkl::bfloat16、std::int8_t、std::int32_tとfloatのいくつかの組合せが提供されている。

正方行列の軸選択付LU分解を計算するLapackルーチンdgesv()はC言語のインターフェイスであるLAPACKE実装では、列優先と行優先形式のどちらの行列も扱うことができ、その内部のインターフェイスであるFortran版と等価な LAPACKE_dgesv_work()を行優先形式から列優先形式への変換を施してから利用する。LAPACKE版はFortran によるLapackルーチンが内部の作業配列が必要な場合でもユーザに代って一時的配列の確保と開放を担当していた。IntelによるoneAPI版では作業配列は計算を実行するデバイスに確保する必要があるため、一時的配列の確保と開放はUSMあるいはBufferメモリモデルに留意してユーザが明示する必要がある。gesvはUSM版のみが準備されている。一時作業配列は gesv_scratchpad_size()により取得して、malloc_device<>により演算デバイスにより確保する。syclのメモリ管理とイベントに関する記述法は同一のため、次の例の syevd()で詳細を述べる。

実数の対称行列の固有値と固有ベクトルを分割統治法により計算するLapackルーチンsyevd()は、固有値を

格納する配列と内部での作業配列を引数に渡す必要がある。

```
using namespace oneapi::mkl
sycl::device dev;
// here, preparatoin of error_handler with try & catch
sycl::queue Q(dev, error_handler);
std::int64_t wrk_size = syevd_scratchpad_size<double>(Q, job::vec,
uplo::lower, n, lda);
// Allocate memory on device
double* A_dev = sycl::malloc_device<double>(A_size, Q);
double* w_dev = sycl::malloc_device<double>(w_size, Q);
double* wrk_dev = sycl::malloc_device<double>(w_size, Q);
sycl::event ev_send = Q.submit([&](sycl::handler &h) {
  h.memcpy(A_dev, A.data(), A_size * sizeof(double)); });
sycl::event ev_syevd = lapack::syevd(Q, job::vec, uplo::lower,
        n, A_dev, lda, w_dev, wrk_dev, syevd_scratchpad_size, {ev_send});
sycl::event ev_rscv = Q.submit([&](sycl::handler &h) {
  h.memcpy(A.data(), A_dev, , A_size * sizeof(double), {ev_syevd} );
  h.memcpy(w.data(), w_dev, , w_size * sizeof(double), {ev_syevd} ); });
sycl::free(wrk_dev, Q);   sycl::free(A_dev, Q);   sycl::free(w_dev, Q);
```

syevd() の演算モードを固有ベクトル計算するように job::vec と設定しているため、固有ベクトルはデバイスに確保されている行列の配列A_devに上書きされる。scyl::devにはCPUあるいはGPUを指定することで単一のコードでそれぞれのアーキテクチャに準備されているsyevd()ルーチンを実行することができる。ホストからのデータ転送とデバイスでの演算はsycl::eventのタスクとして定義し、それぞれに依存関係を記述している。

3.2.5.3.1.5 oneMKL での疎行列向け sparseBLAS ライブラリ

疎行列に対する演算ルーチンはsparseBLASとして提供されるが、以前のinspector-executor法を含むいくつかのバージョンが統合されている。線形方程式の反復解法であるKrylov部分空間法は疎行列と密のベクトルの積の演算である SpMVが主たる演算である。y=αop(A)x+βyは nrow x nrowのサイズの疎行列AをCSR形式で格納するとき、hAをoneapi::mkl::sparse::matrix_handle_tオブジェクトと、CSRデータは std::vector<int> ia、std::vector<int> ja、std::vector<double> vaとホストに準備されているとして、次のように記述される。

```
using namespace oneapi::mkl::sparse;
sycl::queue Q(dev, error_handler);
int* ia_dev = sycl::malloc_device<int>((nrow + 1), Q);
int* ja_dev = sycl::malloc_device<int>(nnz, Q);
double* a_dev = sycl::malloc_device<double>(nnz, Q);
matrix_handle_t hA = nullptr;
init_matrix_handle(hA);
sycl::event ev_send = Q.submit([&](sycl::handler &h) {
  h.memcpy(ia_dev, ia.data(), (nrow + 1) * sizeof(int));
  h.memcpy(ja_dev, ja.data(), nnz * sizeof(int));
  h.memcpy(a_dev, a.data(), nnz * sizeof(double));});
sycl::event ev_set = set_csr_data(Q, hA, nrow, nrow, index_base::zero, ia_dev, ja_dev, a_dev, {ev_send});
sycl::event ev_opt = optmize_gemv(Q, transeose::notranse, hA, {ev_set});
sycl::event ev_gemv = gemv(Q, transeose::notranse, alpha, hA, x, beta, y, {ev_opt});
sycl::event ev_rel = spars::release_matrix_handle(Q, &hA, {ev_gemv});
```

optimize_gemv()はSpMV演算の前に疎行列パターンを解析して前処理をおこなうinspector-executor手法を実現する。matrix_handle_tはCOO形式からもset_coo_data()を用いてデータを設定することができる。

二つの正方疎行列AとBの積の演算から新しい疎行列Cを生成する場合、Cの非零パターンはAとBの非零パターンから決ることに注意する。疎行列AとBをデバイスにコピーした後の、行列Cに関する操作は次のステップとなる。

```
using namespace oneapi::mkl::sparse;
sycl::queue Q(dev, error_handler);
matrix_view_descr viewA = matrix_view_descr::general;
matrix_view_descr viewB = matrix_view_descr::general;
matrix_view_descr viewC = matrix_view_descr::general;
matrix_handle_t hA, hB;  // already set
matrix_handle_t hC = nullptr;
matmat_descr_t descr = nullptr;
matmat_request req;
int* ic_dev = sycl::malloc_device<int>((nrow + 1), Q);
sycl::event ev_setC = set_csr_data(Q, hC, nrow, nrow, index_base::zero,
                           ic_dev, (int *)nullptr, (double *)nullptr, {});
init_matmat_descr(&descr);
set_matmat_data(descr, viewA, transpose::nontrans, viewB, transpose::nontrans, viewC);
// stage 1: work estimation
sycl::event ev1 = matmat(Q, hA, hB, hC, matmat_request::get_work_estimation,
                                 descr, nbuf, nullptr, {ev_setA, ev_setB, ev_setC});
// stage 2: compute
sycl::event ev2 = matmat(Q, hA, hB, hC, matmat_request::compute,
                                  descr, nullptr, nullptr, {ev1});
// Sep 3.1 get nnz
std::int64_t *c_nnz = sycl::malloc_host<std::int64_t>(1, Q);
sycl::event ev3_1 = matmat(Q, hA, hB, hC, matmat_request::get_nnz,
                                    descr, c_nnz, nullptr, {ev2});
// Step 3.2  allocate final c matrix arrays
double *vc_dev = sycl::malloc_device<double>(c_nnz[0], Q);
sycl::event ev3_2 = set_csr_data(Q, hC, nrow, nrow, index_base::zero,
                                     ic_dev, jc_dev, vc_dev,  {ev3_1});
// Step 3.3  finalize into C matrix
sycl::event ev3_3 = matmat(Q, hA, hB, hC, matmat_request::finalize,
                                   descr, nullptr, nullptr, {ev2_2});
// Sort C matrix and copy to host
sycl::event ev_sort = sort_matrix(Q, hC, {ev3_3});
double *vc_host = sycl::malloc_device<double>(c_nnz[0], Q);
Q.copy(vc_dev, vc_host, c_nnz[0], {ev_sort});
```

行列Cの演算はルーチン内部のデータ形式を用いて、デバイス上で実行される。ホストからはまず行列Cの非零要素数をmatrix_handle_t hCから取得しCSR形式の記憶に必要な二つの整数の添字の配列と倍精度の配列を確保したのち、デバイスからホストへのcopy()によりデータを取得する。このルーチンはoneAPIのフレームワークより前のMKLライブラリではmkl_sparse_sp2mm()の関数名疎行列データはC言語構造体sparse_matrix_tに格納して利用していたものである。

3.2.5.3.1.6　SYCL と OpenCL の相互運用

SYCLのparallel_for()に記載するlambda式によるデータパラレル実行のカーネルの記述の代わりに、sycl::queueへのサブミットの前段階でOpenCLのコードを文字列として記述して、parallel_for()はそのOpenCLのkernelの関数名を文字列として指定する。OpenCLのコードはJITコンパイラに渡される。

3.2.5.3.1.7　SYCL と oneTBB の相互運用

DPC++ではSYCLのタスクのランタイムはoneTBBのスケジューラを用いているが、SYCL のキューをoneTBBのtbb:task_groupに埋めこんで、DAGを構成して非同期実行を実現することができる。

3.2.5.3.2　重要ライブラリの個別機能調査ならびに性能調査結果

次期システムに向けた数値ライブラリ整備において、特に性能を重視するライブラリ機能については各種並列処理機構を活用してハードウェア性能を引き出せるかの予測につながる調査も重要である。昨年度に実施した密行列線形代数の代表的なシングルノード用ライブラリを対象とした性能実測に基づく評価分析では、BLAS機能範囲でSIMD対応やレジスタブロッキングによる妥当な程度の性能チューニングが実現できていることは確認できたが、LAPACK機能範囲でメニーコアでのスレッド並列性能に改善の検討余地があると考えられた。

本年度はより広汎な並列処理機構をターゲットとするために比較的新しいアプローチを採用したライブラリを対象として、机上調査および性能実測に基づく評価分析を行った。具体的には、密行列線形計算OSSライブラリで、スレッド並列性能の改善のためにタスク並列を用いて開発されたPLASMA[11][12]と、各種アクセラレータへの対応を狙って開発されたSLATE[13][14][15][16][17][18][19][20][21][22][23][24]を評価対象とした。評価分析の結果、PLASMAについては、ベンダ提供のLAPACKライブラリと比較して機能および性能で対抗できるレベルに至っておらず、開発も活発でないため普及する傾向にはないと考えられる。SLATEについては、開発をバックアップしていたECPプロジェクト終了後の開発やメンテナンスの継続に注視が必要であるが、GPUやマルチコアを備えた多ノードシステム向けのライブラリ[20]として、次期システムに対する課題の技術的検討に適していると考えられる。評価分析の詳細について、PLASMAとSLATEの構築条件と測定環境を3.2.5.3.3.1節、PLASMAの調査結果を3.2.5.3.3.2節から3.2.5.3.3.4節、SLATEの調査結果を3.2.5.3.3.5節から3.2.5.3.3.8節に示す。

3.2.5.3.2.1　評価対象ライブラリと測定環境

ノードあたり搭載演算器数の増加について従来システムでの対処例を調査分析するため、タスク並列を用いて開発された密行列線形計算OSSライブラリであるPLASMAをまず評価対象とした。各種アクセラレータへの対応を含む同機能分野のOSSライブラリとしてはSLATEを評価対象とした。PLASMAおよびSLATEは内部にスレッド並列階層を持つため内部で利用する基本行列演算ライブラリとしては逐次版MKLを結合し、比較対象となるNetlib版のLAPACKおよびScaLAPACKの内部利用ライブラリはスレッド並列版MKLを結合した。なおNetlib版LAPACKの一部にはOpenMPディレクティブが近年入ってきているためにパラレルリージョンのネスト発生に注意が必要であるが、今回の評価対象範囲では影響しない。性能改善余地を明確化する意味でベンダライブラリのIntel MKLも測定対象とした。

利用したライブラリの版数および構築条件を表 3.2.5.1に示す。

表 3.2.5.1 利用ライブラリと構築条件

ライブラリ	構築条件
PLASMA 23.8.2	`cmake -DPLASMA_WITH_MKL=ON ..`
SLATE 2024.05.31	`cmake –Dblas=mkl –Dgpu_backend=none ..`
Netlib ScaLAPACK 2.2.0	`cmake –DCMAKE_Fortran_FLAGS=-fallow-argument-mismatch -DBLAS_LIBRARIES='-lmkl_gf_lp64 -lmkl_gnu_thread -lmkl_core -lgomp' -DLAPACK_LIBRARIES=' -lmkl_gf_lp64 -lmkl_gnu_thread -lmkl_core -lgomp' ..`
Netlib LAPACK 3.11.0	`cmake -DBUILD_SHARED_LIBS=ON -DLAPACKE=ON -DBUILD_TESTING=ON -DUSE_OPTIMIZED_BLAS=ON -DBLAS_LIBRARIES='-lmkl_gf_lp64 -lmkl_gnu_thread -lmkl_core -lgomp -lpthread -lm -ldl' ..`
Intel MKL 2024.1	-

評価機としては、一般的なXeon機として以下のCPUを搭載したマシンを用いた。昨年度の評価条件と同様に動作周波数は2GHz設定としている。

- Intel Xeon Gold 6338
  - Icelake 世代, 2GHz 設定, 32 コア×2 ソケット, メモリ DDR4 256GiB, 1 ノードピーク性能 4096GFLOPS

PLASMAライブラリ概要

BLAS階層のデータ並列方式によるスレッド並列化に頼る構成ではメニーコア化したCPUの性能を引き出せなくなったことで、PLASMAライブラリはタスク並列を用いた密行列線形計算OSSライブラリとして開発された。タスク並列では生成されたタスク内でのタスク生成による入れ子の並列化を実装しやすく、昨年度の成果報告書で課題として挙げた階層的並列化の技術的検討に適している。受け渡された行列を内部的にタイル状に分割してデータアクセスの局所性が改善するように再配置し、各タイルまたは複数タイルに対する処理の依存関係により構成されるDAG(Directed Acyclic Graph)に基づいて並列化を行うことを特徴としている。バージョン2.8.0までは独自ランタイムQUARKで実装していたがバージョン17.01からはOpenMP仕様のタスク並列を利用している。

plasma_dgesvの性能測定および分析

階層的並列化の効果が見えやすい例として、一般行列の連立一次方程式求解ルーチンの性能測定結果を図3.2.5.1に示す。対応するPLASMAライブラリのドライバルーチンはplasma_dgesvであり、内部の実装としてはright-lookingアルゴリズムのループ構成によりdgetrfとdtrsmとdgemmの処理ブロックを呼び出して全体行列のLU分解を行っているが、上層階層にあたるタスクの優先度を指定するpriority節の利用によりlook-ahead方式と類似した処理の重ね合わせを実現している。ここで複数のタイルで構成される列パネルをLU分解するdgetrfの処理ブロックがクリティカルパスとなっている。その処理に1スレッドのみを割り当てた場合(緑実線プロット)は、行列サイズNが8000以下で従来のBLAS階層のデータ並列方式(橙色プロット)よりも性能が出ていない。PLASMAでは、このような列パネル分解に割り当てる下層階層で用いるスレッド数を、mtpf (Maximum number of Threads for Panel Factorization)と呼ぶパラメータで指定できるように構成されている。mtpf値の調整により点線プロットで示されるように2倍程度の性能改善が得られ、階層的並列化の有効性が確認された。

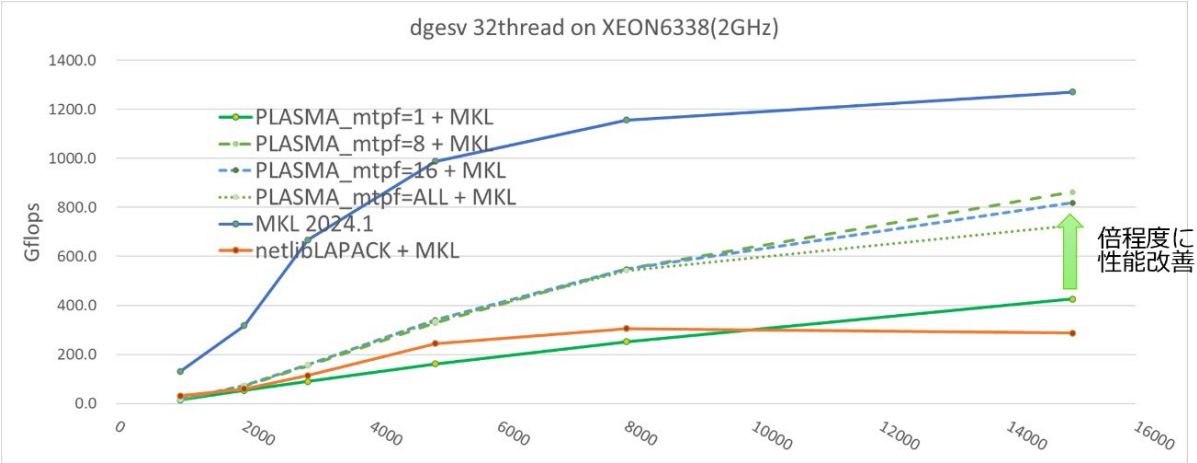


図 3.2.5.1 DGESV 機能の性能

問題行列サイズN=5000でmtpf値を変えた場合のタイムライン表示の違いを図 3.2.5.2に示す。各タイムライン表示の縦軸がスレッド番号、横軸が経過時間であり、各スレッドにおける1クロックあたりの浮動小数点演算数を色で表現している。青色は演算密度が高いdgemm処理であり、緑色は演算密度が低いことを示している。mtpf=1の場合はdgemm処理と次のdgemm処理の間の演算密度が低い区間が長く、そこで1スレッドでのdgetrfによる列パネル分解が行われている。mtpf値の設定によりdgetrf処理に割り当てられるスレッド数を調整することで性能が改善されている。どのスレッドをその列パネル分解に割り当てるかは動的に決まっているが、dgetrf内で行われるピボット選択のように緊密な同期が必要なケースでは、そのタスクを近傍のコアに集める制御ができればNUMAの程度が強いアーキで性能が改善する可能性も考えられる。より一般には、アルゴリズム側の処理の階層構造とハードウェア側の階層的な並列機構のマッピングを最適化する課題になると考えられる。また最適なmtpf値は自動決定できることが望ましい。MKLとの性能差の原因は不明確であるが、行列を分割するブロックサイズやmtpf値相当の固定的パラメータの調整のみで解消しない程度に大きいと思われるため、そのようなマッピング最適化の余地もあると推測される。

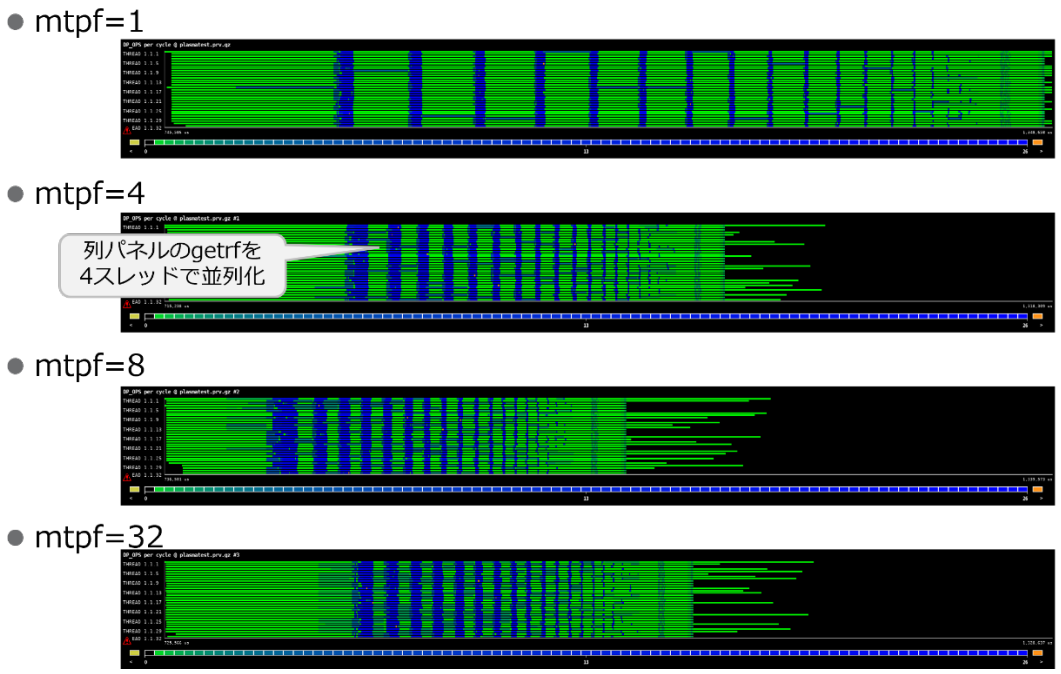


図 3.2.5.2 PLASMA の DGESV 機能のタイムライン分析

PLASMAライブラリに対する見解

以下に挙げるPLASMAの特徴はメニーコア化への対応として妥当なアプローチだと考えられ、それを実現したライブラリとしては評価できる。

- タスク並列方式を採用
- DAG に基づく並列性抽出
- ネストによる階層的並列化
- 各コアで効率の高い処理粒度とするブロッキングサイズ調整が可能
- キャッシュ利用効率の良いタイル行列化

一方で以下に挙げる点は、実用にあたっては問題となる可能性がある。

- 開発が活発ではなくメンテナンスが継続されない懸念がある
- 固有値関連機能などの重要機能をサポートしていない
- データ再配置のために入力行列サイズと同程度のメモリを消費する
- 利用インターフェイスが LAPACK と異なる
- スレッド間同期処理の一部がビジーループで独自実装されている

現状ではベンダ提供のLAPACKライブラリと比較して機能および性能で対抗できるレベルに至っていないこともあり、普及する傾向にはないと考えられる。

SLATEライブラリ概要

SLATEライブラリは、GPUやメニーコアを備えた多ノードシステム向けに、LAPACKおよびScaLAPACK機能をカバーする密行列線形計算OSSライブラリを提供することを目的として、米国 Exascale Computing Projectのサポートを受けてテネシー大のICLが主体となり開発された。開発当初はNVIDIAのGPUクラスタ向けであったがAMDやIntelのGPUにも対応が行われている[24]。ルーチン呼び出し時に渡すパラメータ内のターゲット指定により、内部的にbatched-BLAS形式となる処理については各社ベンダライブラリを利用してGPUにオフロードできるように構成されている。C++を用いて分散格納された行列を抽象化された行列オブジェクトの受け渡しで利用する方向性としているが、LAPACKやScaLAPACKの従来API利用からの移行に配慮した呼び出しAPIも用意している。またオーバーロードやテンプレートを用いて、変数型に依らない実装となっている。

DGESV機能の性能測定および分析

一般行列の連立一次方程式解法機能ルーチン[21]の性能測定結果を図 3.2.5.3に示す。測定マシンの32コア2ソケットの64コアマシンにおいて、プロセス数とスレッド数の組み合わせを変えたパターンで探索し、各ライブラリで最良の性能を示した。ScaLAPACKおよびPLASMAとの比較を主眼として、アクセラレータを利用しないメニーコアシステムでの性能評価としている。ノード内プロセス並列を用いるSLATEおよびScaLAPACKでのプロセスグリッド形状はPxQ=4x8で、各プロセスが2スレッドのハイブリッド並列実行である。SLATE(赤プロット)は従来ライブラリScaLAPACK(黒プロット)に対して2割程度の高速化が確認できた。一方でSLATEに含まれるLAPACK互換APIを用いた場合(橙プロット)の性能は、タスク並列方式を採用しDAGに基づく並列性抽出やネスト並列による階層化を用いているという観点で類似のアプローチを用いているにも関わらず、PLASMA(緑プロット)よりも大幅に劣っている。SLATEにおいても列パネル分解を複数スレッドで行う階層的並列化は可能であり、plasma_dgesv測定時のmtpf値はSLATEのデフォルト値に合わせて利用スレッド数の半分とする条件に合わせてある。詳細は不明であるが、SLATEのLAPACK互換APIを用いた場合には、整合寸法を持つTileと見なすことにより入力された行列データをTileの集合に並べ替えはせずに受け渡しているためにデータアクセスの局所性がPLASMAよりも悪いことや、

列パネル分解の下層の並列化においてomp parallelのネストで実装していることによる動的スレッド生成ペナルティなどが影響していると推測される。

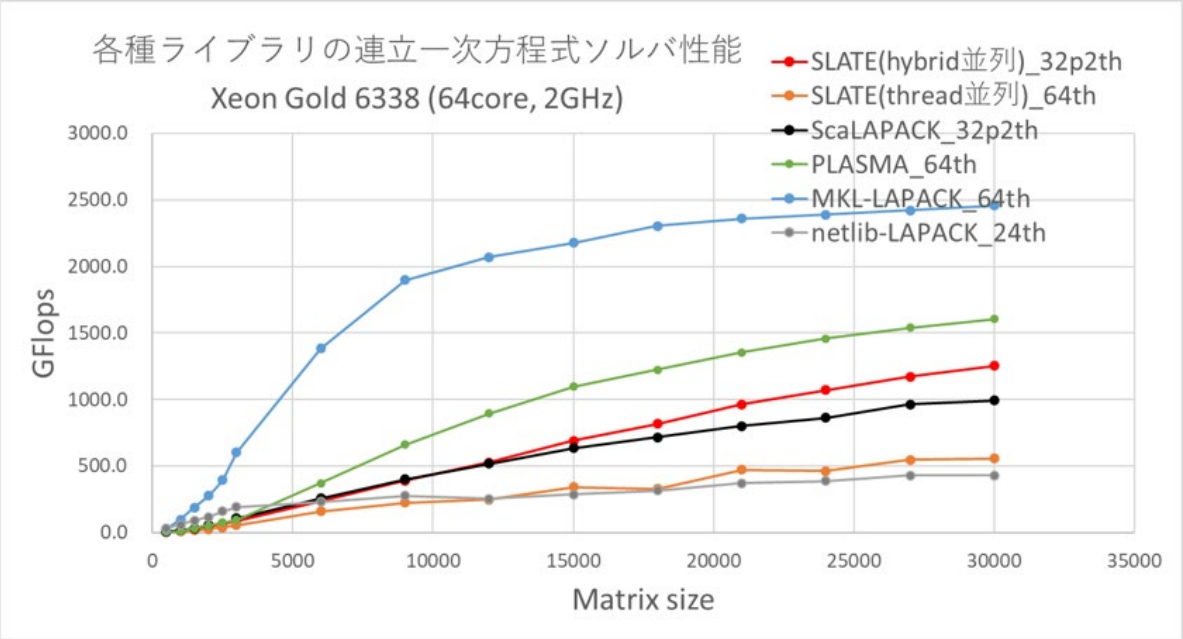


図 3.2.5.3 DGESV 機能の性能

SLATEのハイブリッド並列実行時の行列サイズN=10000でのDGESV処理のタイムライン分析を図 3.2.5.4に示す。全体に他スレッドまたはプロセス間通信の待ちによるアイドル状態が多く、処理効率が悪くなっている。特に薄茶色で示される列パネルの分解の処理効率が低く、処理が進むにつれて徐々に短くなるべきところ、区間幅が不自然に変動していることが全体性能を悪化させており、階層的並列化の下層での同期待ちが影響している疑いがある。この列パネル分解は常にホストCPU側のタスク並列方式で処理する実装になっている一方、演算量の大半を占めるdgemmによるtrailing-submatrix更新処理の時間割合は全体時間に対してCPU実行ですら小さい。CPU側に残る列パネル分解処理がボトルネックとなっている状況のままではdgemm部分をGPUで加速しても大きな効果が見込まれず、現時点の実装では、N<10000程度の場合にGPU利用による性能改善は期待できないと推測される。

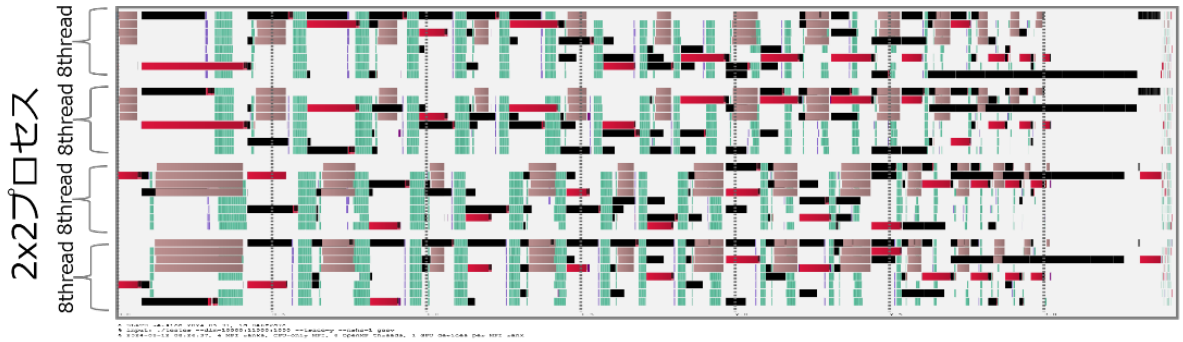


図 3.2.5.4 SLATE の DGESV 機能のタイムライン分析

DSYEVD機能の性能測定および分析

分割統治法による実対称密行列固有値問題求解機能[22][23]の性能グラフを図 3.2.5.5に示す。測定マ

シンの32コア2ソケットの64コアマシンにおいて、プロセス数とスレッド数の組み合わせを変えたパターンで探索し、各ライブラリで最良の性能を示している。固有値だけでなく固有ベクトルも計算する条件とし、演算量を$13n^3/3$とみなした換算性能値を用いた。なおSLATEは三重対角化の処理に2-stageアルゴリズムを用いているため実際の演算量は2倍程度となり、MKLも行列サイズによっては内部アルゴリズムの切り替えにより演算量が異なるように見えるが、グラフ上で比較しやすくするために換算性能値を用いている。今回の評価条件ではSLATEのハイブリッド並列(赤プロット)よりもScaLAPACK(黒プロット)が高性能であり、SLATEをScaLAPACKの代替とするには性能が不足しているとみられる。また、SLATEのLAPACK互換API性能(橙プロット)が低い傾向はDGESVと同様の問題であり、現状ではLAPACK機能の代替ライブラリとしての期待は薄い状況が確認された。

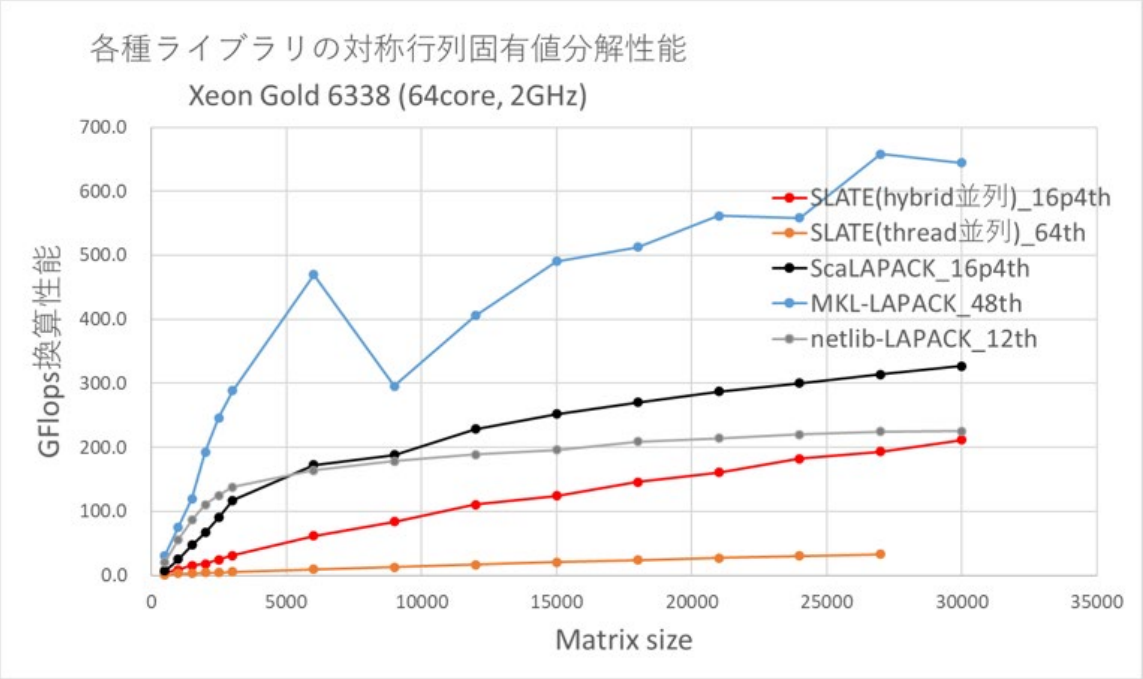


図 3.2.5.5 DSYEVD 機能の性能

SLATEのハイブリッド並列実行時のDSYEVD処理の行列サイズN=6000でのタイムライン分析を図 3.2.5.6 に示す。全体に他スレッドまたはプロセス間通信の待ちによるアイドル状態が多く、処理効率が悪くなっている。特にhb2st(対称バンド行列から三重対角行列への変換処理)が1プロセスのみでの実行となっていることが全体性能を悪化させている。ソースコード中にも現時点では複数プロセス実行に対応できていない旨のコメント行があり、開発元でも課題は認識されていると思われる。また、stev(三重対角行列の固有値・固有ベクトル求解)も動作区間がまばらで処理効率が悪い。仮に各プロセスで冗長に計算したとしてもNetlib-LAPACK程度の性能は達成できるはずで、さらにstev内部のdlaed4相当の処理は並列化可能である。前節のDGESVと同様に、GPUで処理可能な部分をオフロードして効果を得るには、CPU側での処理を改善する必要があると考えられる。

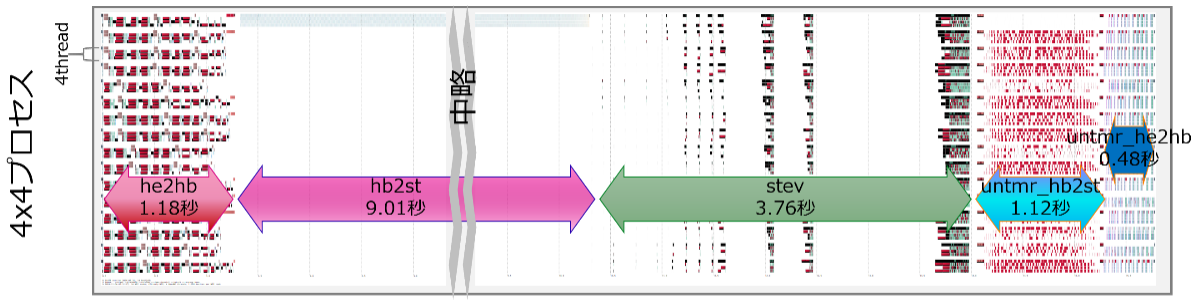


図 3.2.5.6　SLATE の DSYEVD 処理のタイムライン分析

SLATEライブラリに対する見解

SLATEライブラリは密行列線形計算の主要機能をサポートしつつあり、各社GPU(NVIDIA，AMD，Intel)および分散並列・スレッド並列に対応したポータビリティのあるライブラリとしての可能性を持っていると評価できる。また、C++の採用による開発側および利用側のメリットとして、テンプレートやオーバーロードを用いて冗長な記述を避けた開発が可能であることや、抽象化された行列オブジェクトの受け渡しによりモダンなプログラミングで利用可能であることがあげられる。

一方で懸念事項としては、ScaLAPACKの代替を目的としているがScaLAPACK性能を越えていないケースもあり、ハードウェア性能を十分に引き出せる設計であるか現時点では不明確である。例えばCPU側でクリティカルパスを処理するように構成されていたり、粒度の小さい末端処理での緊密な同期が必要な個所の実装に問題を含んでいたりする可能性がある。また、MKLとの性能比較から具体的な改善課題を抽出すれば、CPUのみでの実行に限らずGPU利用時にも応用可能と期待できる。

実用的な品質や性能となるように、ECPプロジェクト終了後も開発やメンテナンスが継続されるか注視が必要であるが、少なくとも次期システムに対する課題の技術的検討に適していると考えられる。

3.2.5.3.3　GPU 環境に特化した重要ライブラリの個別機能調査ならびに性能調査結果

AI 加速のみならず様々な分野のコンピューティングを加速するGPUが現在のスーパーコンピュータを牽引している。先例として米国のECP(Exascale Computing Project)で開発された、Frontier や Aurora などはAMD社のMI250, Intel社のXe GPU(コードネーム: Ponte Vecchio) に基づく。また、ドイツJSC（Juelich Supercomputer Center）が導入を進めるJUPITERでは NVIDIA社のGH200[29]（コードネーム:Grace Hopper）が採用されている。各社GPUの仮想アーキテクチャには類似点が多いものの、各社プログラミング環境や数学ライブラリに大きな違いがある。次世代計算インフラ整備の中でGPUもしくはそれに相当するアクセラレータが技術トレンドとして最も有力視される。既存GPUシステム特に数値計算ライブラリの差異を考慮したうえでシステムソフトウェア基盤に対する基本機能の対応状況や実行性能などを理解しなくてはならない。各社GPUに対応する数値計算ライブラリは数少ないが、テネシー大学が開発するMAGMA[26][27]はNVIDIA社のCUDA, AMD社のHIPの両者に同一の機能を提供する。次期システムに対するGPU性能を公平に比較検討する際のベンチマークコードとして適していると考えられる。評価分析の詳細について、NVIDIAとAMD社の最新GPU環境においてMAGMAライブラリが提供するBLAS部分の性能結果を示す。以下、主要なカーネルについての性能結果（対次元性能[FLOPS]）のグラフを示すが、本報告の趣旨は現時点での一般的な実装が示す性能の傾向を把握することにあるため、性能上限に対する比率や性能の安定性などを中心にコメントする。性能改善点や内部の詳細な分析は、今後のライブラリの改良にゆだねることにとどめる。

評価対象ライブラリ関数と測定環境

GPU環境における基本行列演算ライブラリとしてはMAGMAのBLAS機能MAGMABLASの性能を評価する。CPU環境における標準BLASとほぼ同等の関数群に加えて、上位のMAGMAのLAPACK相当に対応する各種拡張関数、更にはAI向けに強化された低精度演算機能が追加されている。今回の調査では、CPU環境との比較が明瞭となるようレベル2と3から倍精度と単精度の行列行列積、行列ベクトル積についてそれぞれ行列の形状に応じて測定する。利用したライブラリの版数および測定対象関数の条件を表 3.2.5.2に示す。

表 3.2.5.2 利用ライブラリと測定対象関数と条件 1 -

測定対象のMAGMAテストスイーツの対応関数

行列積:	testing_dgemm	(倍精度行列行列積)
	testing_sgemm	(単精度行列行列積)
	testing_dsymm	(倍精度対称行列行列積)
	testing_ssymm	(単精度対称行列行列積)
	testing_dtrmm	(倍精度三角行列行列積)
	testing_strmm	(単精度三角行列行列積)
行列ベクトル積	testing_dgemv	
	testing_sgemv	
	testing_dsymv	
	testing_ssymv	
	testing_dtrmv	
	testing_strmv	

条件：行列は正方行列とし、行列の次元は { 512, 1000, 1024, 2000, 2048, 3000, 4000, 4096, 5000, 6000, 7000, 8000, 8192, 9000, 10000, 12000, 12288, 16384, 20000, 20480, 24576, 28672, 30000, 32768, 36864, 40000, 40960, 45056 }の28通りである。
データはMAGMAライブラリが提供する乱数ルーチンによって初期値を生成する。
なおtesingスイーツからはプロプラエタリライブラリ(NVIDIAの場合cuBLAS[28], AMDの場合ROCm/hipBLAS[32][33][34]の当該カーネル)とMAGMABLASのカーネルの２実装の測定がなされる。

評価機としては、NVIDIA社とAMD社の最新GPUを機軸として以下のマシンを用いた。

GPU環境１ ：NVIDIA社 GH200 superchip [29]
＊CPU: NVIDIA Grace
Arm Neoverse V2 CPU, Armv9 with 4x128b SVE2, 72コア,
3.1GHz設定(3.0GH SIMD), LPDDR5Xメモリ(ECC)
＊GPU: NVIDIA H100 TensorCore GPU
HBM3 96GB, 4000GB/sの帯域性能
実数演算機構の性能

FP64(ベクトル)	34 TFLOPS/FP64(Tensor Core)	67 TFLOPS
FP32(ベクトル)	67 TFLOPS	

GPU環境２ ：AMD社 MI250 [30][31]

＊CPU: AMD EPYC 7713 64-Core Processor x2

＊GPU: AMD Intrinsic MI250

HBM2e 128GB, 3200 GB/sの帯域幅(2GCDの合計)

2GCD構成

実数演算機構の性能

FP64(ベクトル)	45.3 TFLOPS/FP64(マトリックス)	90.5 TFLOPS
FP32(ベクトル)	45.3 TFLOPS/FP32(マトリックス)	90.5 TFLOPS

測定に利用したMAGMAライブラリはリリースバージョン2.8.0、それぞれのGPU環境での構築条件は以下のとおりである。

GPU環境1: NVIDIA GH200

CUDA 12.5

CPU側BLAS: OpenBLAS 0.3.9

MAGMAコンパイル時はsm90を指定し、その他の最適化オプションはデフォルト指定

GPU環境2: ADM MI250

Rocm 6.0.0

CPU側BLAS: OpenBLAS 0.3.9

MAGMAコンパイル時はgfx90aを指定し、その他の最適化オプションはデフォルト指定

DGEMM/SGEMM

一般行列の行列行列積ルーチンの性能測定結果を図 3.2.5.7、図 3.2.5.8に示す。GH200ではsuperchipのGPU部分のみ、MI250では1GCDのみを利用した性能評価である。横軸が行列次元、縦軸が性能値(GFLOPS)である。青線はベンダ提供(プロプラエタリ)cuBLASライブラリの倍精度版、緑色が単精度版である。一方で、赤色はMAGMAが提供するMAGMABLASで提供される関数の倍精度版、紫色が単精度版である。

GH200については倍精度と単精度浮動小数点の理論性能値は67TFLOPSと同等のはずであるが、cuBLASでは倍精度の方が10~15%程度の性能が高い結果となっている。cuBLASのGEMMはcutlassと呼ばれるGEMM最適化ツールによって生成されているが、その中でも近年ではベクトルユニットとTensorCoreユニットを用いたコード生成方法に差が出てきており、実測結果にも現れたものと見られる。MAGMAもコード内部を分析すると、伝統的な階層的ブロッキング技法を駆使しているが、性能的にはベンダ提供版には3倍程度の差をあけられている。また、コードの解析を進めているが、倍精度と単精度の性能差に２倍の差異がみられる点はベクトルユニットのみを用いてコード生成されていることに起因している。

MI250については倍精度と単精度浮動小数点の理論性能値は２GCD構成でいずれも90.5TFLOPS(マトリックス演算器使用時) 、ベクトルユニット使用時は45.3TFLOPSである。なお、HIP環境からは1GCDのみをアクセスする機能のみ提供されており同ピーク性能の半分が上限となる。hipBLASでは倍精度の方はピーク性能の60~80%、単精度版は80%強の性能値を示している。大きな性能変動は見られないが、倍精度版の想定性能は改善の余地があると考えられる。MAGMABLASについては、ソースコードの解析からGH200のCUDA版と同等の振る舞いをするはずであるが、ピーク性能差が倍精度と単精度とで差がないため、実測にみられる差が明確ではなくどこに由来するかは今後明らかにしなくてはならない。

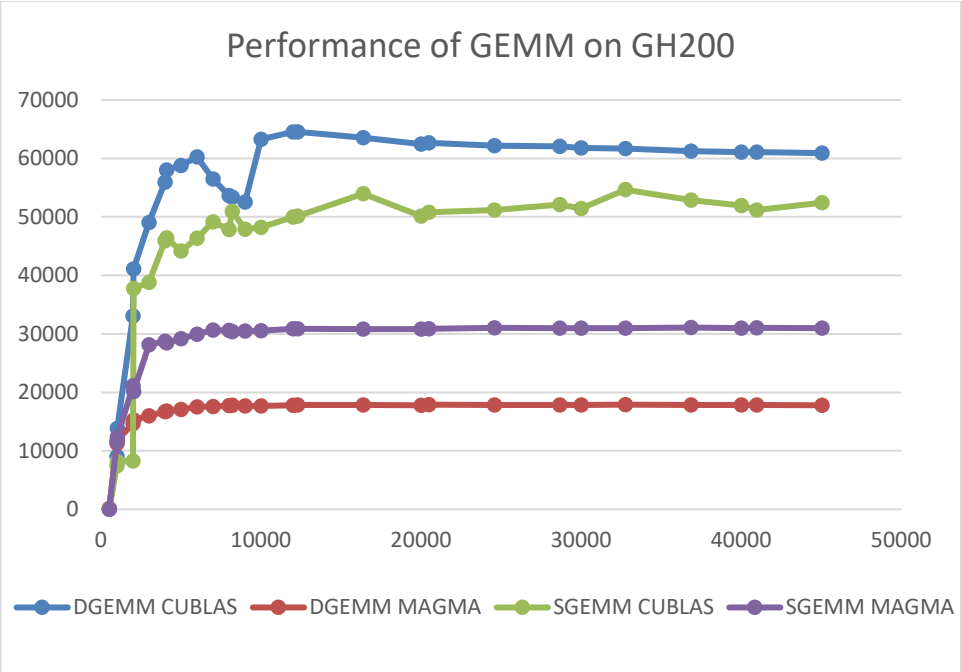


図 3.2.5.7　GH200 における GEMM カーネルの性能

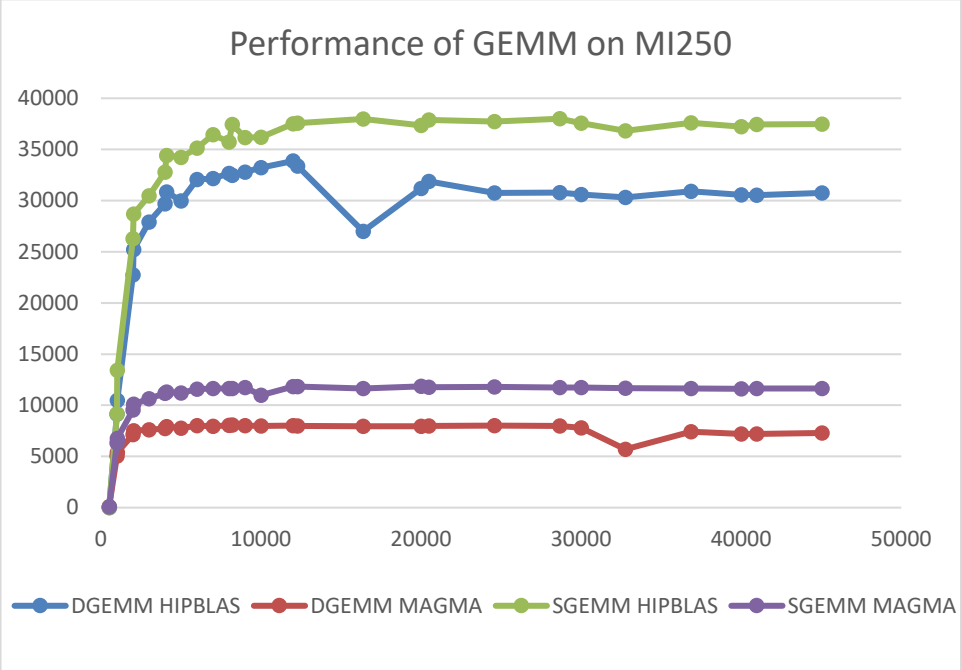


図 3.2.5.8　MI250 における GEMM カーネルの性能

DSYMM/SSYMM

実対称行列と一般行列との行列積ルーチンの性能測定結果を図 3.2.5.9、図 3.2.5.10に示す。横軸が行列次元、縦軸が性能値(GFLOPS)である。GH200ではsuperchipのGPU部分のみ, MI250では1GCDのみを利用した性能評価である。青線はベンダ提供(プロプラエタリ)cuBLASライブラリの倍精度版、緑色が単精度版である。一方で、赤色はMAGMAが提供するMAGMABLASで提供される関数の倍精度版、紫色が単精度版である。性能上限は行列積の性質からGEMMと同程度であり、SYMMの振る舞いは原理的にはGEMMと大きく変わらない。

GH200については行列の対称性を使うことで生じる何らかのオーバーヘッドもしくは適切な最適化が困難となることから性能劣化がみられる。ただし、倍精度版と単精度版との性能差はGEMMほど大きくはない。MAGMABLASはcuBLAS版から大きく性能面で差がついている。これはベクトル性能とTensorCore性能の比以上であり、最適化の違いが大きく表れていることになる。

MI250については倍精度と単精度浮動小数点の理論性能値は２GCD構成でいずれも90.5TFLOPS(45.3TFLOPSベクトルユニット性能)であるが、HIP環境からは1GCDのみをアクセスする機能のみ提供されており同ピーク性能の半分が上限となる。hipBLASでは倍精度の方はピーク性能の75~80%、単精度版は80%強の性能値を示している。大きな性能変動は見られないが、倍精度版の想定性能は改善の余地があると考えられる。MAGMABLASについては、ソースコードの解析からGH200のCUDA版と同等の振る舞いをするはずであるが、ピーク性能差が倍精度と単精度とで差がないため、実測にみられる差が明確ではなくどこに由来するかは今後明らかにしなくてはならない。

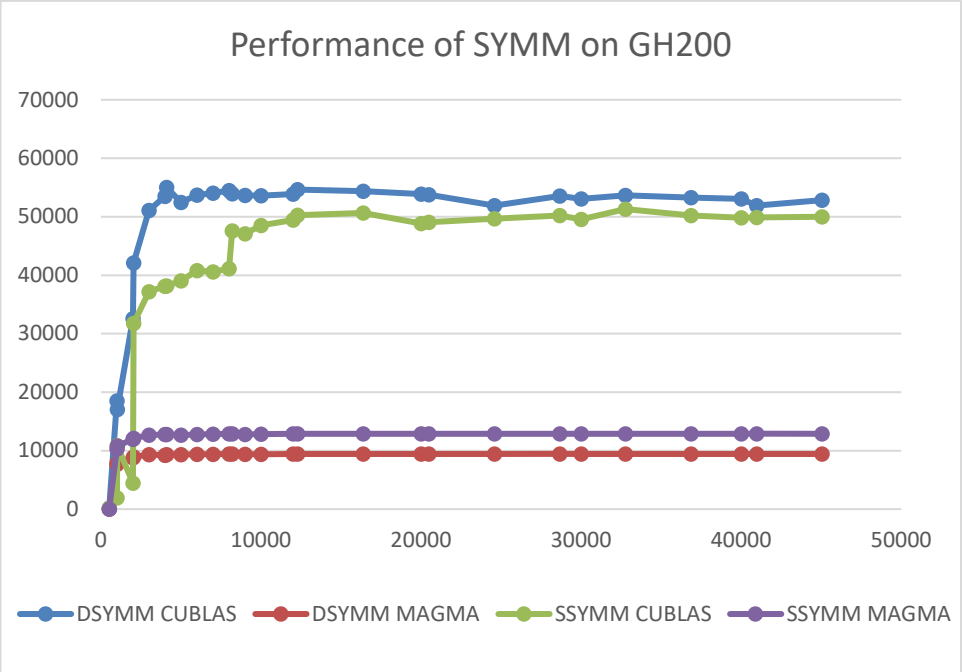


図 3.2.5.9　GH200 における SYMM カーネルの性能

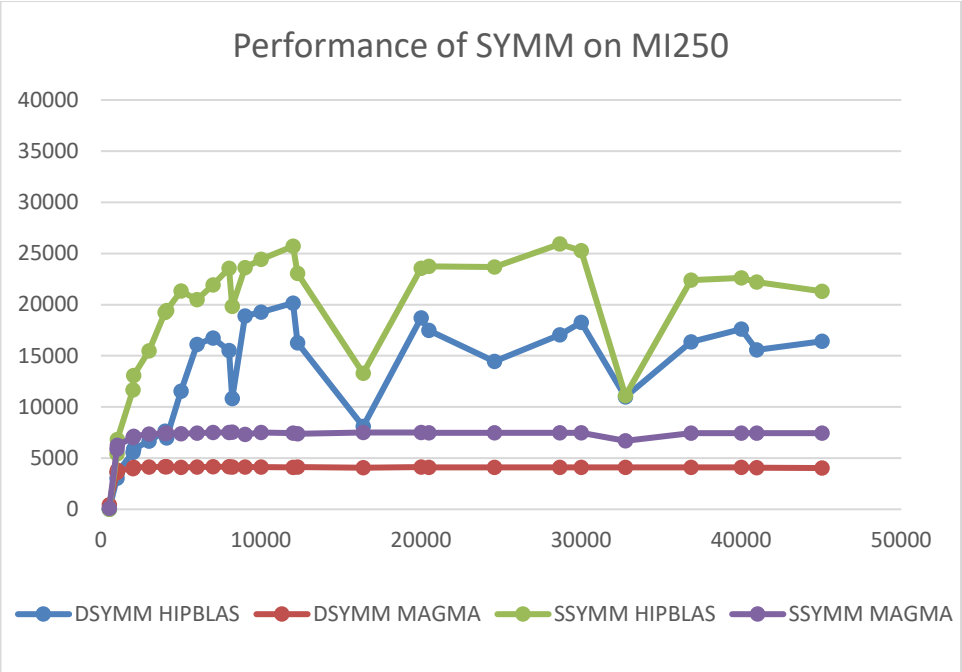


図 3.2.5.10　MI250 における SYMM カーネルの性能

DTRMM/STRMM

実三角行列と一般行列との行列積ルーチンの性能測定結果を図 3.2.5.11、図 3.2.5.12に示す。横軸が行列次元、縦軸が性能値(GFLOPS)である。GH200ではsuperchipのGPU部分のみ、 MI250では1GCDのみを利用した性能評価である。青線はベンダ提供(プロプラエタリ)cuBLASライブラリの倍精度版、緑色が単精度版である。一方で、赤色はMAGMAが提供するMAGMABLASで提供される関数の倍精度版、紫色が単精度版である。性能上限は行列積の性質からGEMMと同程度であり、TRMMの振る舞いは原理的にはGEMMと大きく変わらないが、三角行列処理時の負荷バランスの不均衡が起こりうるのでそれを注視しなくてはならない。

GH200の結果ではcuBLASとMAGMABLASの性能に差がないということが確認できる。これは倍精度、単精度いずれも同様である。倍精度はSYMMなどと同様に高い性能を示している。単精度も他のカーネルと同様だが、理論ピーク上の倍精度と差がみられる点で性能改善の余地がある。なお、MAGMABLASのコードから、TRMMは内部の小規模な三角行列と一般行列積GEMMの 3 か所に分けて計算を行っていることがわかる。これらから、性能面での傾向はGEMMに依存していることは容易に理解できる。

一方、MI250の方でも性能的にはhipBLASとMAGMABLASの傾向は近いものがあり、実装上 3 か所に分けているようであるが、非常に不安定な性能を示していることがわかる。性能の極端な落ち込みは16384, 32768など2ベキの次元にみられることから、キャッシュラインの競合回避が不十分である可能性があり、今後リリースにおける改善が望まれる。

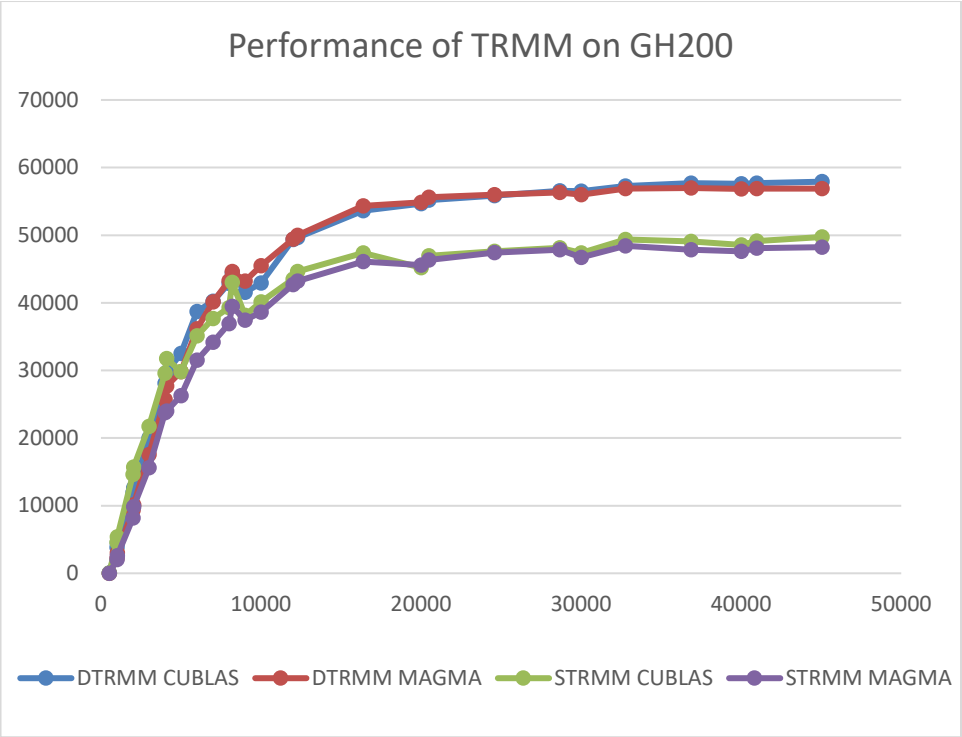


図 3.2.5.11　GH200 における TRMM カーネルの性能

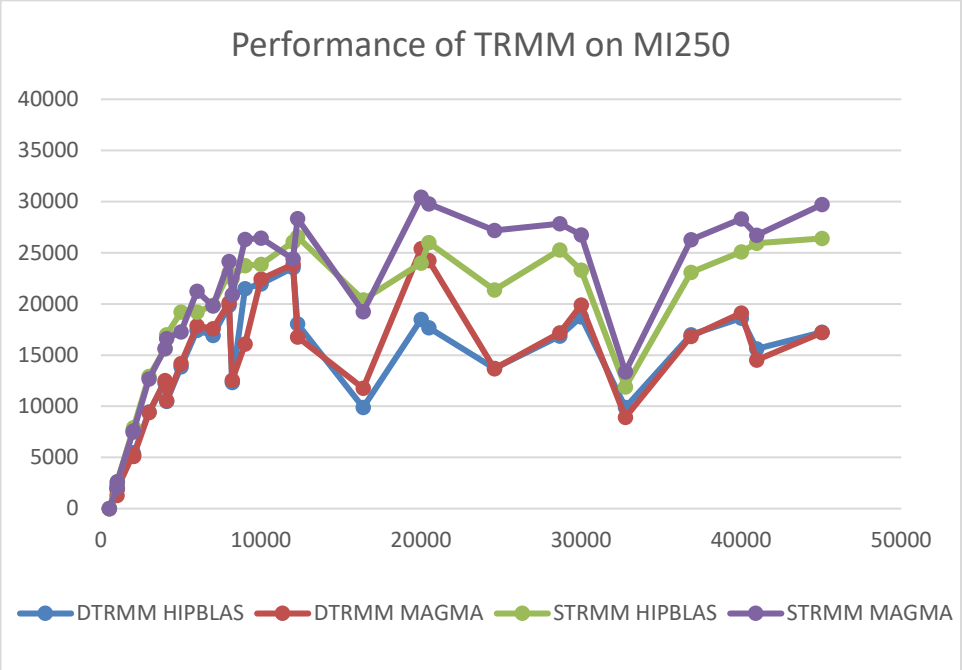


図 3.2.5.12　MI250 における TRMM カーネルの性能

DGEMV/SGEMV

実一般行列のベクトル積ルーチンの性能測定結果を図 3.2.5.13、図 3.2.5.14に示す。横軸が行列次元、縦軸が性能値(GFLOPS)である。GH200ではsuperchipのGPU部分のみ、MI250では1GCDのみを利用した性能評価である。青線はベンダ提供(プロプラエタリ)cuBLASライブラリの倍精度版、緑色が単精度版である。一方で、赤色はMAGMAが提供するMAGMABLASで提供される関数の倍精度版、紫色が単精度版である。性能上限はメモリバンド幅[GB/s]*計算強度[FLOPS/B]で計算可能である。

GH200の場合の性能上限は、倍精度で4000[GB/s]*2/8[FLOPS/B]=1000[GFLOPS]、単精度で4000[GB/s]*2/4[FLOPS/B]=2000[GFLOPS]と見込まれる。性能結果から倍精度、単精度版ともに80~90%程度の性能上限に迫っている。cuBLASでは性能の落ち込みが一か所みられる箇所があるが、cuBLAS, MAGMABLASいずれも概ね次元に対して滑らかな性能を示している。

MI250の場合は、1GCDの性能上限が、倍精度で3200[GB/s]/2*2/8[FLOPS/B]=402[GFLOPS]、単精度で3200[GB/s]/2*2/4=800[GFLOPS]である。hipBLASについては性能上限に近い振る舞いを示しているが、不規則なジャギー性能がみられる。MAGMABLASは性能上限の半分程度しか達成できていないが、倍精度版では1000万次元以下の小規模問題で性能がよいなど、MI250版の実装には改善点があることが読み取れる。

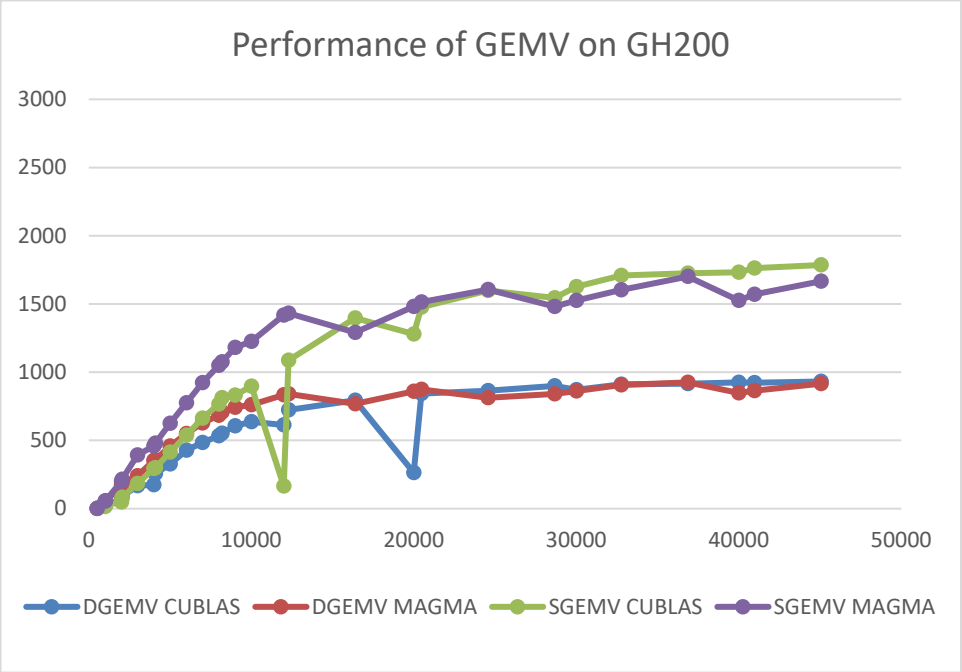


図 3.2.5.13　GH200 における GEMV カーネルの性能

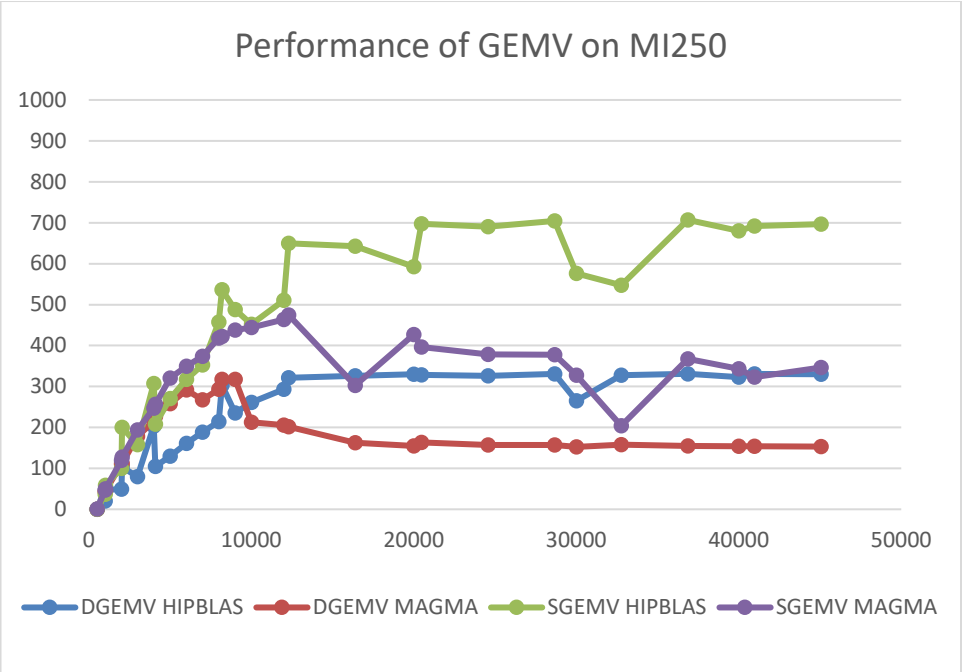


図 3.2.5.14　MI250 における GEMV カーネルの性能

DSYMV/SSYMV

実対称行列のベクトル積ルーチンの性能測定結果を図 3.2.5.15、図 3.2.5.16に示す。横軸が行列次元、縦軸が性能値(GFLOPS)である。GH200ではsuperchipのGPU部分のみ、MI250では1GCDのみを利用した性能評価である。青線はベンダ提供(プロプラエタリ)cuBLASライブラリの倍精度版、緑色が単精度版である。一方で、赤色はMAGMAが提供するMAGMABLASで提供される関数の倍精度版、紫色が単精度版である。性能上限はメモリバンド幅[GB/s]*計算強度[FLOPS/B]で計算可能であるが、対称性を活用して計算強度がGEMVの半分になる点に注意が必要である。

GH200の性能上限は倍精度が4000[GB/s]*4/8[FLOPS/B]=2000[GFLOPS], 更に単精度が4000[GB/s]*4/4[FLOPS/B]=4000[GFLOPS]である。cuBLASでは性能上限の75~80%特に倍精度版の方が高い傾向にあることが判る。cuBLASの実装についてはアトミック演算を活用したAhmadらの方式を利用していることが知られており、MAGMABLASはワーク配列にスレッドブロック毎に格納して最後にギャザーする止まり木方式であり、両者のオーバーヘッドが差に出たと考えられる。特にMAGMABLASはハードウェアに対する適切なタスク配分ができていないため、測定次元での線形な性能上昇が観測されていると考えられる。

MI250では1GCDでの性能上限は倍精度、単精度いずれも800[GFLOPS], 1600[GFLOPS]である。hipBLAS, MAGMABASともに測定範囲では性能が飽和しておらずまだ上昇傾向にある。cuBLASのような性能立ち上がりの好調さがみられないので、その部分についてはアルゴリズム改良を指摘したい。

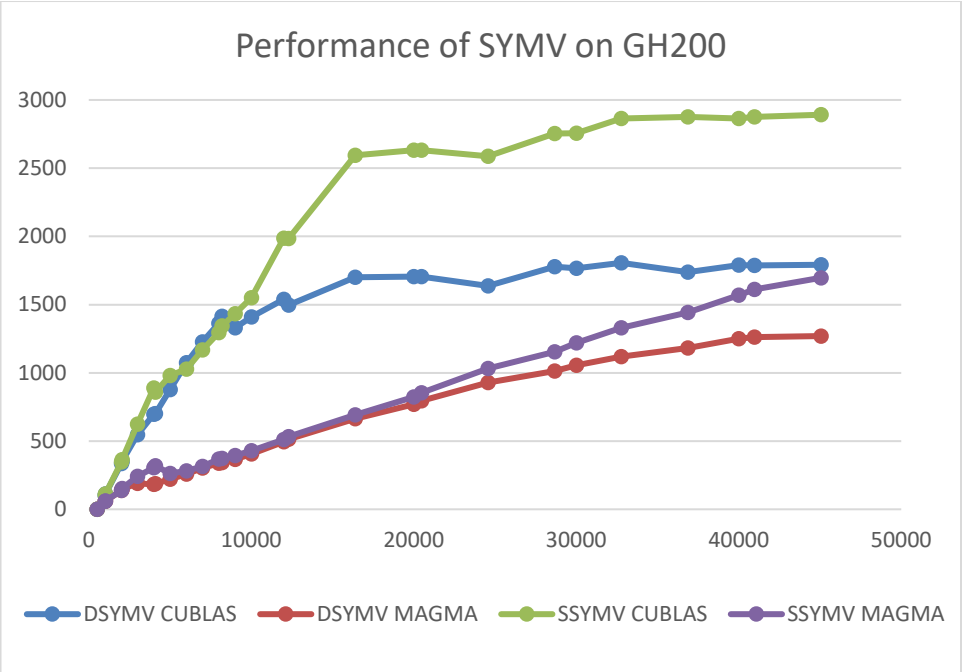


図 3.2.5.15　GH200 における SYMV カーネルの性能

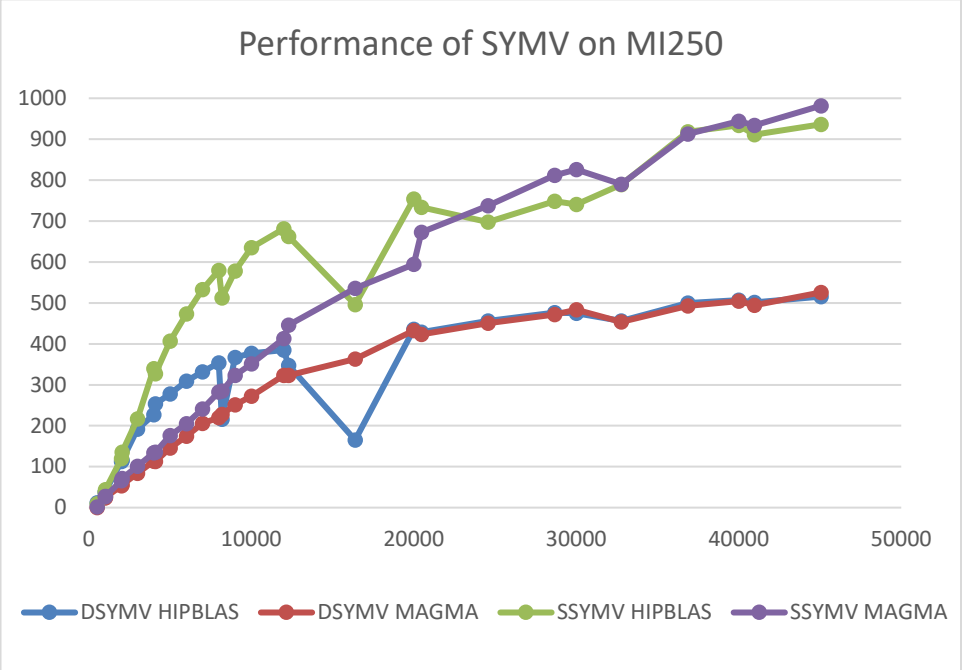


図 3.2.5.16　MI250 における SYMV カーネルの性能

DTRMV/STRMV

実三角行列のベクトル積ルーチンの性能測定結果を図 3.2.5.17、図 3.2.5.18に示す。横軸が行列次元、縦軸が性能値(GFLOPS)である。GH200ではsuperchipのGPU部分のみ、MI250では1GCDのみを利用した性能評価である。青線はベンダ提供(プロプラエタリ)cuBLASライブラリの倍精度版、緑色が単精度版である。一方で、赤色はMAGMAが提供するMAGMABLASで提供される関数の倍精度版、紫色が単精度版である。性能上限はメモリバンド幅[GB/s]*計算強度[FLOPS/B]で、GEMVと同様である。

本カーネルについてはGH200、MI250いずれも性能上限には届いていない。cuBLASの性能は性能上限の10分の一以下である。hipBLASは半分以下の性能に留まる。MAGMAが線形にスケールしている状況は、GPUコアなどの資源割付が適切に行われず計算負荷不均衡が極めて悪いと解釈できる。cuBLASは性能が飽和状態になっているにもかかわらず、性能上限には大幅に届いていない状況である。性能の安定性については、cuBLAS、MAGMABLASともに問題はない。hipBLASは特定次元でジャギーな傾向がみられる。SYMVカーネルの演算を半分キャンセルした形でTRMVは実装が可能なはずであり、性能はSYMVの50%に迫ることが期待できるはずである。

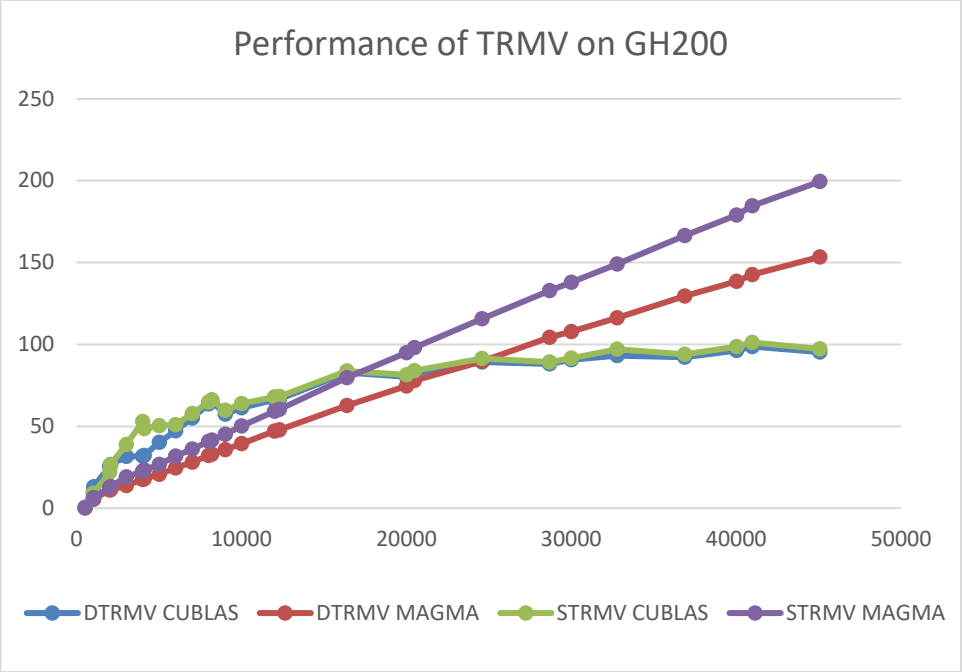


図 3.2.5.17　GH200 における TRMV カーネルの性能

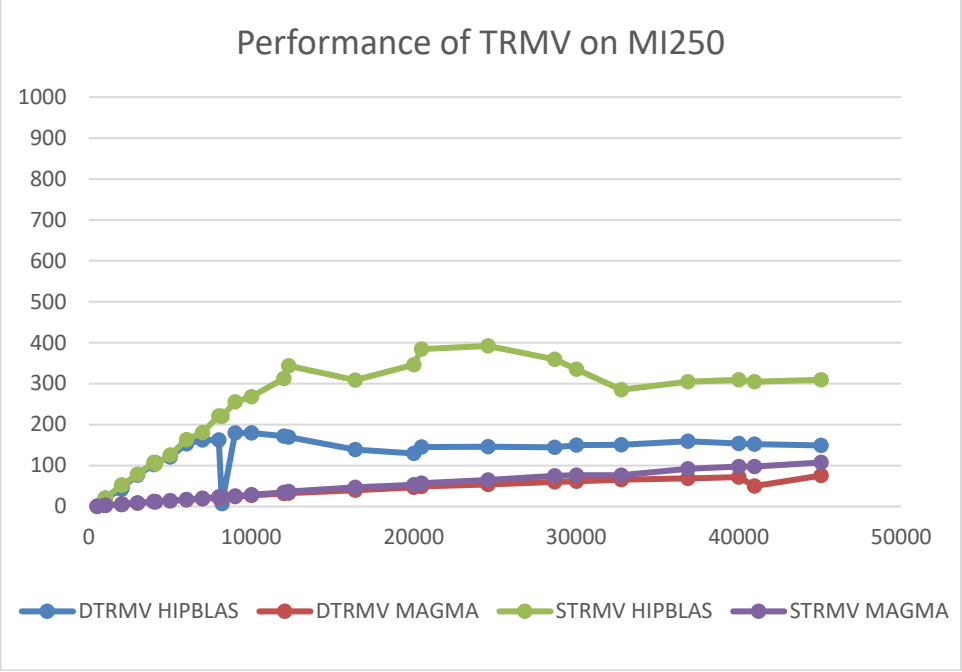


図 3.2.5.18　MI250 における TRMV カーネルの性能

性能評価まとめ

GPUにオフロードすることが想定される十分な粒度であるレベル3BLAS、レベル2BLASカーネルについてMAGMAライブラリのtestingスイーツを利用して性能を測定した。現状のライブラリの性能を把握する目的で実施したため、十分な解析や多くのコメントはできていないが、個別の関数、ライブラリについての傾向は把握できている。表 3.2.5.3、表 3.2.5.4は、「行列次元45056における各カーネルの性能測定データ」と「理論ピーク性能から算出される性能上限値」との比を「対ピーク性能比」としてまとめている（ピーク性能や性能上限の根拠は前段までの各項で示している。またLevel2 BLASではメモリバンド幅とアルゴリズムから定まる数値であり、ハードウェアの理論ピーク性能とは異なる）。対ピーク性能比が0.7を超えるものを赤字で協調しており、性能特性が読みとれるようにしている。

特に重要なカーネルであるDGEMM(FP64、倍精度版)については、cuBLASで90%を超える十分な性能が達成されている。プロプラエタリのcuBLASではTRMVを除いていずれも75%以上の性能を発揮している。hipBLASはGEMM, GEMV以外ではピーク性能の66%にも満たない。OSSのMAGMAはLevel3 BLASについては性能最適化での不足が見受けられる。Level2 BLASについてはcuBLASとMAGMAに大きな差はないが、hipBLASとMAGMAの間には1:2程度の差がある。このように「GEMM以外のその他カーネル」における性能最適化には各ライブラリの濃淡が顕著にみられ、十分に実施されているとは言えないものが見られた。

仮に、これらの改良版や後継バージョンが次世代計算基盤の一部となることが想定される場合には、十分な改良が必要であることが認識される。特に、プロプライエタリであるcuBLASの性能は多くは十分な性能を見せているが、オープンソース版のMAGMABLASやhipBLASは絶対性能としての達成度が不足するものが見られたり、キャッシュラインの影響による不安定性などがあり、将来のソフトウェアスタックの中で十分競争力のある選択肢となるためには更なる改善が期待される。

表 3.2.5.3　GH200 における BLAS カーネルの対ピーク性能比
（%で表示、赤字は 66%以上, 青字は 33%以下）

	GEMM	SYMM	TRMM	GEMV	SYMV	TRMV
FP64(cuBLAS)	91%	77%	86%	92%	89%	10%
FP64(MAGMA)	27%	14%	85%	86%	63%	14%
FP32(cuBLAS)	76%	74%	73%	88%	72%	5%
FP32(MAGMA)	46%	19%	72%	79%	40%	9%

表 3.2.5.4　MI250 における BLAS カーネルの対ピーク性能比
（%で表示、赤字は 66%以上, 青字は 33%以下）

	GEMM	SYMM	TRMM	GEMV	SYMV	TRMV
FP64(hipBLAS)	68%	36%	38%	82%	64%	37%
FP64(MAGMA)	16%	9%	38%	38%	65%	19%
FP32(cuBLAS)	83%	47%	58%	87%	59%	39%
FP32(MAGMA)	26%	16%	66%	43%	61%	13%

3.2.5.3.4　想定されるシステム上で重要度の高いライブラリの選定とまとめ

2023年度の活動報告の中で、重要性の高いライブラリの候補として

- 数値計算ライブラリ１　:BLAS: 数値線形代数の基本演算パッケージライブラリ
- 数値計算ライブラリ２　:密行列関連のライブラリ
  - 逐次版数値線形計算ライブラリ：LAPACK、SLATE
  - 分散並列向け数値線形計算ライブラリ：SLATE, DPLASMA:
- 数値計算ライブラリ３　:密行列固有値計算ライブラリ EigenExa
- 数値計算ライブラリ４　:疎行列係数値計算ライブラリ
  - PETSc:分散並列 PDE, MUMPS/SuperLU/Dissection, ARPACK-NG/FEAST などの既存 OSS
- 数値計算ライブラリ５　:FFT ライブラリ
  - FFTX-SPIRAL(FFTW 後継として)、国産 FFTE

などを挙げた。選定の主な理由として

1) 利用者が潜在的に利用する可能性が極めて高いソフトウェア
2) 国際的なデータセンターで通常利用が見込まれるソフトウェア
3) 国内のスパコンユーザが希望するソフトウェア
4) 国内研究者による国際競争力が高く継続的な開発が見込まれるソフトウェア

の４点について考慮した。2023年度の報告書にある利用調査サーベイの中、最も利用者数が多いBLAS、LAPACK、ならびにFFTX-SPIRALは重要度が極めて高く、当該ソフトウェアは業界標準的な存在として高い指示を得ているものであり、選定することには異論はない。また世界をリードする国産のHPCソフトウェア群についても、開発を継続することによるソフトウェアならびにシステムの優位性を最大化するという観点からEigenExa[35], FFTE[36]などを最終候補とする。アプリケーション開発からの重要性の変化が昨年度から大きく変化したという事象は見られないため、昨年度報告書と同様に最終提言として上記の５項目のライブラリについて、次世代計算基盤開発において優先度を高く設定すべきライブラリとしたい。

3.2.5.4　調査報告まとめ

本数値ライブラリ調査研究サブグループでは、2028年頃に登場する次世代スパコンシステムにおいて、現在の「富岳」級システムでの動向調査を進め、その延長線上の技術外挿ならびにいくつかの技術動向予測を加え、次世代スパコンにおける数値計算ライブラリの様相についての調査を実施した。主に、科学技術計算一般に利用される数値計算ライブラリ群について調査をし、富岳NEXT-Software-Matrixをまとめたが、数値計算の対象ごとに多くのソフトウェアが存在し、その情報を系統的に整理することは極めて困難を伴うことが理解できた。ハードウェア環境の新たな展開や進化に伴い、新しい数値計算ソフトウェアも必要となったため、年度毎に新規調査項目を追加しつつ調査を進める形となった。なお、初年度(2022年)は一般的なオープンソース群、昨年度(2023年)はCPUを中心として性能が必要なカーネル関数について、今年度(2024年)はCPU環境で未評価のものとGPU環境での中核カーネル群についての性能評価に基づいた既存のスパコン環境の理解を深めた。さらに、次世代スパコンに必要となる数値ライブラリ群の中で優先度を高く設定すべき項目を提言としてまとめた。

なお、本報告・提言では実際の次世代計算基盤環境となるハードウェアアーキテクチャ、更には現状のCPUもしくはGPU環境からの変化追従すべき部分への考慮は実施していない。つまり、環境変化に連動したリスク評価は含まれていない。本年度の調査研究で実施したGPUにおける性能評価からは、従来環境であるCPUのソフトウェアで期待されていた性能を超えるものはいくつか達成されているが、すべての機能でその恩恵が得られるわけではない。

CPU版からの移植などのリスクの発生は十分に予想可能である。さらにGPUやアクセラレータを活用するためのプログラミング環境やシステムランタイムは個々のベンダに大きく依存している現状がある。今年度調査を行ったIntel社のoneAPIを活用するためのsyclプログラミング環境への移行や、NVIDIA社のCUDA、 AMD社向け環境のHIP/ROCmへの移行にはコードの修正に伴うコストの他、数世代先における当該環境の継続性についても、プログラム資産維持の観点からもリスク評価が伴わなくてはならない。さらに、オープンソースソフトウェアの積極的な利用はコミュニティベースのソフトウェアエコ環境構築活動の一環として歓迎したい。しかしながら、数学ソフトウェアは開発者のライフタイムとリンクすることが多く、開発から10年または20年経過後にコミュニティで真にメンテナンス可能な体制が維持できるか、もしくは次世代計算基盤環境の構築時にコミュニティに積極的に働きかけるなど外部を意識した活動が必要になる可能性がある。今後は、ここに上げたようなリスクに関する詳細評価を次世代計算基盤の開発開始とともに実施し、数値計算ライブラリを含むシステムソフトウェア開発と移行期間のリスクヘッジについても明確な数値化や対策方針を今後共有していく必要性がある。

参考文献

[1] SYCL2022 SPECIFICATION (REVISION 9): HTTPS://REGISTRY.KHRONOS.ORG/SYCL/SPECS/SYCL-2020/HTML/SYCL-2020.HTML

[2] SYCL REFERENCE：HTTPS://GITHUB.KHRONOS.ORG/SYCL_REFERENCE

[3] INTEL ONEAPI PROGRAMMMING GUIDE：HTTPS://WWW.INTEL.COM/CONTENT/WWW/US/EN/DOCS/ONEAPI/PROGRAMMING-GUIDE/2025-0/OVERVIEW.HTML

[3] ONEAPI GPU OPTIMIZATION GUIDE：HTTPS://WWW.INTEL.COM/CONTENT/WWW/US/EN/DOCS/ONEAPI/OPTIMIZATION-GUIDE-GPU/2025-0/OVERVIEW.HTML

[4] ONEMKL – DATA PARALLEL C++ DEVELOPER REFERECNE：HTTPS://WWW.INTEL.COM/CONTENT/WWW/US/EN/DOCS/ONEMKL/DEVELOPER-REFERENCE-DPCPP/2025-0/OVERVIEW.HTML

[5] ONEAPI FOR NVIDIA GPUD 2025.0.0：HTTPS://DEVELOPER.CODEPLAY.COM/PRODUCTS/ONEAPI/NVIDIA/2025.0.0/GUIDES/INDEX

[6] ONEAPI FOR AMD GPUS 2025.0.0：HTTPS://DEVELOPER.CODEPLAY.COM/PRODUCTS/ONEAPI/AMD/2025.0.0/GUIDES/INDEX

[7] ADATPIVECPP：HTTPS://GITHUB.COM/ADAPTIVECPP/ADAPTIVECPP

[8] NEOSYCL：HTTPS://GITHUB.COM/TOHOKU-UNIVERSITY-TAKIZAWA-LAB/NEOSYCL

[9] Y. KE, M. AGUNG, H. TAKIZAWA, “NEOSYCL: A SYCL IMPLANTATION FOR SX-AURORA TSUBASA”, HPCASIA ’21 PROCEEDINGS, DOI:101145/3432261.3432268

[10] TRISYCL：HTTPS://GITHUB.COM/TOHOKU-UNIVERSITY-TAKIZAWA-LAB/NEOSYCL

[11] PLASMA 23.8.2: HTTPS://GITHUB.COM/ICL-UTK-EDU/PLASMA/RELEASES/TAG/23.8.2

[12] DONGARRA, JACK, ET AL. "PLASMA: PARALLEL LINEAR ALGEBRA SOFTWARE FOR MULTICORE USING OPENMP." ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE (TOMS) 45.2 (2019): 1-35.

[13] ABDELFATTAH, A., H. ANZT, A. BOUTEILLER, A. DANALIS, J. DONGARRA, M. GATES, A. HAIDAR, J. KURZAK, P. LUSZCZEK, S. TOMOV, ET AL., “ROADMAP FOR THE DEVELOPMENT OF A LINEAR ALGEBRA LIBRARY FOR EXASCALE COMPUTING: SLATE: SOFTWARE FOR LINEAR ALGEBRA TARGETING EXASCALE,” SLATE WORKING NOTES, NO. 01, ICL-UT-17-02: INNOVATIVE COMPUTING

Laboratory, University of Tennessee, June 2017.
[14] Gates, M., J. Kurzak, A. Charara, A. YarKhan, and J. Dongarra, “SLATE: Design of a Modern Distributed and Accelerated Linear Algebra Library,” International Conference for High Performance Computing, Networking, Storage and Analysis (SC19), Denver, CO, ACM, November 2019.
[15] SLATE webページ：https://icl.utk.edu/slate/
[16] SLATE githubリポジトリ：https://github.com/icl-utk-edu/slate
[17] SLATE Users’ Guide
[18] SLATE Developers’ Guide
[19] SLATE tutorial
[20] Kurzak, Jakub, et al. "Linear systems solvers for distributed-memory machines with GPU accelerators." Euro-Par 2019: Parallel Processing: 25th International Conference on Parallel and Distributed Computing, Göttingen, Germany, August 26–30, 2019, Proceedings 25. Springer International Publishing, 2019.
[21] YarKhan, A., M. Al Farhan, D. Sukkari, M. Gates, and J. Dongarra, ”SLATE Performance Report: Updates to Cholesky and LU Factorizations,” Innovative Computing Laboratory Technical Report, no. ICL-UT-20-14: University of Tennessee, October 2020.
[22] Akbudak, Kadir, et al. “SLATE performance improvements: QR and eigenvalues” SLATE Working Notes, no.17, Innovative Computing Laboratory, University of Tennessee (2021): 04-2021.
[23] Gates, Mark, Ali Charara, and Asim YarKhan. “ Implementing Singular Value and Symmetric/Hermitian Eigenvalue Solvers.” SLATE Working Note 13 (Revision 2023-06).
[24] SLATE port to AMD and Intel platforms, 2021
[25] S. Tomov, J. Dongarra, and M. Baboulin. “Towards dense linear algebra for hybrid GPU accelerated manycore systems.” Parallel Matrix Algorithms and Applications, Parallel Computing, Vol. 36, No.5-6, pp.232-240, 2010
[26] MAGMA library, ICL University of Tennessee: https://icl.utk.edu/magma/
[27] MAGMA GitHub repository. https://github.com/icl-utk-edu/magma
[28] cuBLAS: The API Reference Guide for cuBLAS, the CUDA Basic Linear Algebra Subroutine library. https://docs.nvidia.com/cuda/cublas/index.html
[29] NVIDIA GH200 Grace Hopper Superchip Architecture (Performance and Productivity for Strong-Scaling HPC and Giant AI Workloads), https://resources.nvidia.com/en-us-data-center-overview-mc/en-us-data-center-overview/nvidia-grace-hopper-superchip-architecture-whitepaper
[30] AMD Instinct MI250性能諸元:
https://www.amd.com/en/products/accelerators/instinct/mi200/mi250.htm
[31] AMD Instinct MI250 microarchitecture (online article), AMD ROCm document,
https://rocm.docs.amd.com/en/latest/conceptual/gpu-arch/mi250.html
[32] AMD: hipBLAS documentations.
https://rocm.docs.amd.com/projects/hipBLAS/en/latest/
[33] AMD: ROCm/hipBLAS GitHub repository. https://github.com/ROCm/hipBLAS

[34] AMD: ROCBLAS GITHUB REPOSITORY. HTTPS://GITHUB.COM/ROCM/ROCBLAS
[35] EIGENEXA： GITHUB REPOSITORY: HTTPS://GITHUB.COM/RIKEN-RCCS/EIGENEXA
[36] FFTEホームページ：HTTPS://WWW.FFTE.JP/

3.2.6 AI フレームワーク

3.2.6.1 調査目的

AIによる手法は現代科学において重要かつ不可欠なものとなり、スーパーコンピューティングシステムにおける主要なワークロードの1つとなった。FSプロジェクトのこの部分の目的は、AIワークロードを実行するための次期フラッグシップスーパーコンピュータの準備状況を評価し、促進することである。

3.2.6.2 調査内容

本編の内容は、以下の通りである：

- まず、昨年の研究に基づき、次世代コンピューティングプラットフォームのアーキテクチャを検討し、科学アプリケーションにおける高スループット推論のための主要な AI フレームワークの活用方法を分析した。
- 次に、科学分野における AI 推論のシームレスな統合に必要なハードウェアサポートについて検討し、評価を行った。これらの結果をもとに、今後検討すべき技術要件を整理した。

3.2.6.3 調査結果

比較実験と分析: 本研究では、重みのみの量子化手法を適用した場合のスピードアップ効果を評価するための実験を行った。具体的には、広く使用されている大規模言語モデル（LLM）である**LLaMA-2-7B** および **LLaMA-2-13B** に対し、AWQ アルゴリズムを用いて 4 ビット量子化を適用した。その後、量子化されたモデルを単一の **NVIDIA H100 GPU** 上で **TensorRT-LLM** および **LMDeploy** の 2つの異なる推論フレームワークを用いてデプロイし、バッチサイズおよびコンテキスト長が異なる入力系列に対するスピードアップを評価した。

プリフィルレイテンシ、デコーディングレイテンシ、およびエンドツーエンドレイテンシのスピードアップ結果を表 3.2.6.1 に示す。結果から以下の重要な観察が得られた。

1. **重みのみの量子化はデコーディング段階の高速化に大きく寄与し、エンドツーエンドのレイテンシを改善する。**
   これは、量子化された低精度の重みテンソルを **高帯域幅メモリ（HBM）** から高速にロードできるためであり、メモリアクセスのオーバーヘッドが大幅に削減されることに起因する（前述の「効率的な分析」セクションを参照）。
2. **プリフィル段階では、重みのみの量子化がレイテンシを増加させる可能性がある。**
   これは、プリフィル段階のボトルネックがメモリアクセスではなく**計算コスト**であるためである。したがって、活性化を量子化せずに重みのみを量子化しても、レイテンシにはほとんど影響を与えない。さらに、重みのみの量子化では **FP16 への逆量子化（de-quantization）** が必要となり、これが追加の計算オーバーヘッドを発生させ、結果的にプリフィル段階の処理を遅延させる。
3. **バッチサイズおよび入力系列長が増加するにつれて、重みのみの量子化によるスピードアップの効果は次第に低減する。**
   これは、バッチサイズや入力系列長が大きくなると、レイテンシの大部分を**計算コスト**が占めるようになるためである。重みのみの量子化は主に**メモリアクセスコスト**を削減するが、バッチサイズと入力系列長が増加するにつれて、計算コストが支配的になり、全体のレイテンシに対する影響が相対的に小さくなる。
4. **重みのみの量子化は、大規模モデルにおいて特に大きなメリットをもたらす。**
   モデルサイズが大きくなるにつれ、重みの保存とアクセスに必要なメモリ量が比例して増加するため、メモリアクセ

スのオーバーヘッドが顕著になる。重みを量子化することで、メモリ使用量を削減し、メモリアクセスのオーバーヘッドを効果的に低減できる。

以上の結果から、重みのみの量子化は特に**デコーディング段階の最適化や大規模モデルの推論**において有効であることが示された。一方で、プリフィル段階や大規模バッチ処理における影響は限定的であり、今後の最適化手法の検討が必要となる。

表 3.2.6.1 参考成績

		TensorRT-LLM				
	B	128	256	512	1024	2048
LLaMA-2-7B	1	1.06/2.40/2.37	0.90/2.38/2.34	0.92/2.30/2.28	0.88/2.19/2.17	0.91/2.00/1.98
	2	0.88/2.10/2.05	0.91/2.07/2.04	0.89/2.01/1.98	0.91/1.92/1.89	0.88/1.78/1.76
	4	0.92/1.72/1.67	0.89/1.67/1.64	0.90/1.61/1.58	0.87/1.53/1.51	0.84/1.42/1.40
	8	0.91/1.43/1.36	0.88/1.38/1.33	0.83/1.33/1.28	0.77/1.25/1.21	0.78/1.16/1.14
	16	0.91/1.43/1.36	0.88/1.38/1.33	0.83/1.33/1.28	0.77/1.25/1.21	0.78/1.16/1.14
	B	128	256	512	1024	2048
LLaMA-2-13B	1	1.24/2.51/2.50	0.89/2.45/2.47	0.94/2.34/2.42	0.90/2.18/2.32	0.83/1.94/2.16
	2	0.90/2.51/2.50	0.95/2.45/2.47	0.90/2.34/2.42	0.83/2.18/2.32	0.80/1.94/2.16
	4	0.96/1.80/1.76	0.91/1.78/1.74	0.83/1.73/1.69	0.80/1.65/1.62	0.83/1.54/1.52
	8	0.91/1.86/1.77	0.83/1.81/1.73	0.80/1.73/1.66	0.82/1.62/1.56	0.75/1.46/1.41
	16	0.84/1.84/1.69	0.81/1.77/1.63	0.82/1.63/1.53	0.78/1.46/1.39	OOM
		LMDeploy				
	B	128	256	512	1024	2048
LLaMA-2-7B	1	1.30/2.11/2.09	0.94/2.07/2.05	0.90/2.03/2.02	0.88/1.97/1.96	0.94/1.92/1.91
	2	1.03/2.24/2.20	0.90/2.19/2.15	0.88/2.11/2.08	0.93/1.97/1.95	0.85/1.78/1.76
	4	0.90/2.18/2.10	0.87/2.12/2.05	0.93/2.01/1.96	0.92/1.86/1.83	0.92/1.64/1.62
	8	0.92/1.92/1.77	0.91/1.82/1.71	0.92/1.65/1.57	0.93/1.45/1.41	0.94/1.28/1.26
	16	0.92/1.92/1.77	0.91/1.82/1.71	0.92/1.65/1.57	0.93/1.45/1.41	0.94/1.28/1.26
	B	128	256	512	1024	2048
LLaMA-2-13B	1	1.32/2.34/2.32	0.94/2.31/2.28	0.92/2.22/2.20	0.94/2.15/2.13	0.94/2.01/1.99
	2	1.06/2.42/2.36	0.92/2.37/2.32	0.94/2.29/2.25	0.94/2.15/2.12	0.95/1.95/1.93
	4	0.93/2.36/2.26	0.94/2.29/2.21	0.94/2.18/2.12	0.95/2.01/1.97	0.96/1.78/1.75
	8	0.92/2.24/2.10	0.93/1.93/2.02	0.94/1.81/1.89	0.94/1.65/1.71	0.95/1.45/1.49
	16	0.93/2.02/1.85	0.94/1.90/1.76	0.94/1.73/1.63	0.95/1.50/1.45	OOM

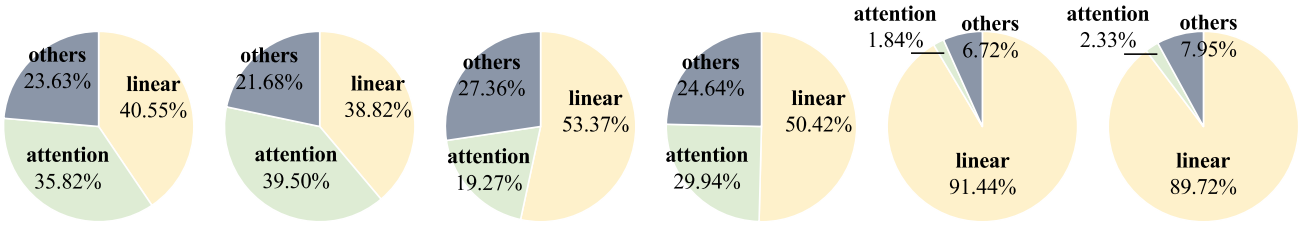
HuggingFace の実装を使用して、異なるモデルおよびコンテキスト長に対する推論実行時間をプロファイリングする。図 3.2.6.1に示すプロファイリング結果から、アテンション演算子および線形演算子が推論時間の大部分を占め、両者の合計時間がしばしば全体の 75%以上に達することがわかる。そのため、演算子レベルでの最適化の多くは、これら2つの演算子の性能向上に注力される。

さらに、実行時間のごく一部しか占めない演算子が多数存在し、それらが演算子の実行スケジュールを細分化し、CPU側でのカーネル起動コストを増大させる。この問題に対処するため、計算グラフレベルでは、高度に融合された演算子を実装する最適化推論エンジンが一般的に採用されている。

アテンション演算子の最適化: 標準的なアテンション計算（例：PyTorchの実装）では、クエリ行列 (Q) とキー行列 (K) の積が計算されるが、これにより入力系列長に対して二次的な時間・空間計算量が生じる。図 3.2.6.1に示すように、コンテキスト長が増加するにつれてアテンション演算子の時間割合も増大する。これは特に長い系列を扱う際に、メモリ容量と計算能力への高い要求につながる。

GPU上での標準アテンション計算における計算・メモリのオーバーヘッドを軽減するため、カスタマイズされたアテンション演算子が不可欠である。**FlashAttention**は、アテンション計算全体を単一のメモリ効率の高い演算子として統合し、メモリアクセスのオーバーヘッドを削減する。具体的には、入力行列 (Q, K, V) およびアテンション行列を複数のブロックに分割することで、全データのロードを不要にする。

FlashAttentionを基盤とする **FlashDecoding**は、デコーディング時の計算並列性を最大化することを目的としている。デコーディング手法の適用により、Q 行列はデコーディング中にベクトルのバッチへと変換されるが、並列化がバッチサイズの次元に限定されると計算ユニットの利用が困難になる。FlashDecoding は、この問題に対処するため、系列次元に沿った並列計算を導入する。これにより、softmax計算の同期オーバーヘッドが若干増加するものの、特に小さなバッチサイズや長い系列において顕著な並列性の向上を実現する。



(a) Llama2-7B, 128 context length　(b) Llama2-7B, 2k context length　(c) Baichuan2-13B, 128 context length　(d) Baichuan2-13B, 2k context length　(e) Mixtral-8x7B, 128 context length　(f) Mixtral-8x7B, 2k context length

図 3.2.6.1 プロファイリング結果

表 3.2.6.2 調査結果

					Hugging face		
					Hugging face		
					Hugging face		
					Hugging face		

表 3.2.6.2は、LLMの推論で広く使用されているプロダクションレベルのソフトウェアを収集しました。各ソフトウェアが正常に動作することを確認するために、異なるハードウェア上で性能をテストしました。対象となるハードウェアには、x86 CPU、Arm CPU、NVIDIA GPU、AMD GPU、Intel GPUが含まれます。各フレームワークを機能に基づいて分類し、整理しました。表 3.2.6.2 にその結果をまとめています。 Microsoft、NVIDIA、Hugging Faceは、これらのフレームワークの主要な貢献者です。 DeepSpeed、Megatron-LM、Colossal-AIは、分散型トレーニングと最適化を目的として設計されています。 TransformersとLoRAは、モデルのファインチューニングに広く使用されています。TritonとTensorRT-LLMは、特にNVIDIAのGPUで効率的な推論に焦点を当てています。SkyPilotとvLLMは、分散型推論ソリューションを提供します。LangChainとBabyAGIは、LLMを使用したアプリケーション構築により重点を置いています。

3.2.7　OS・仮想化・クラウド連携

3.2.7.1　調査目的

次世代情報基盤のためのオペレーティングシステム(OS)および仮想化技術に関する調査、検討を行う。次世代情報基盤では、メニーコアCPUやアクセラレータの導入が加速し、コンテナのような仮想化技術の必要性は高まっている。性能、省電力、ユーザビリティ、セキュリティの観点で、次世代スーパーコンピュータ向けのOSおよび仮想化技術について調査検討する。

3.2.7.2　調査内容

R4年度は、既存スパコンにおけるOS利用状況と軽量OSに関する調査、HPC分野における仮想化技術の動向調査、およびアプリケーションユーザからのOS、仮想化技術、クラウド連携への要望調査を行った。また、R5年度は、仮想化技術、特にHPC分野におけるコンテナ仮想化技術の評価および技術要件の整理と、欧米スパコンセンターにおけるコンテナ技術の導入状況の調査を行った。R6年度は、HPC分野におけるコンテナ仮想化技術について追加調査を行い、取り組むべき課題を報告する。

3.2.7.3　調査結果

本年度は、(1)HPC分野におけるコンテナ仮想化技術の追加調査、(2)OS、仮想化、クラウド連携に関して次世代計算基盤において取り組むべき課題をまとめる。

3.2.7.3.1　HPC 分野におけるコンテナ仮想化技術の追加調査

HPC向けのコンテナ仮想化技術の動向ついてはR5年度の報告で概ね完了しているため、ISC2024併設の10th Annual High Performance Container Workshop (HPCW'24)およびSC24併設のCANOPIE-HPC Workshop (CANOPIE-HPC 2024)から興味深い発表について紹介する。主に以下について、報告する。

- ポータビリティと性能
- KubernetesとHPC／バッチスケジューラの連携
- 科学ワークフロー実行と応用技術

[ポータビリティと性能]

コンテナは、Dockerfileのようなテキストでアプリケーションプログラムの構成情報を記述し、それをもとにそれぞれの環境で利用できるコンテナイメージ（バイナリファイル）を生成し、実行する。しかしながら、ホスト側で利用しているライブラリやデバイスドライバとDockerfileで想定しているものとに齟齬が生じていると正常に実行することができないといったポータビリティの課題がある。また、性能向上のためホストのハードウェアを直接利用するための標準的な手続きをどうするか、といった課題もある。これらを解決するため、以下のような試みがある。

CEAとLLNLが共同で2020年からWi4MPIと呼ばれるMPIのラッパーモジュールを開発している[1-1]。MPIは実装ごとに異なるバイナリインタフェースを利用する必要があるため、Wi4MPIでダイナミックリンクで各MPIライブラリを利用できるようにしている。これにより、1つのコンテナイメージでホストごとのMPI実装を利用できるようにしている。また、MPIライブラリのバイナリインタフェース(MPI Application Binary Interface (ABI))を標準化する試みもあり、MPICHではプロトタイプを実装済み、Open MPIもサポート予定となっている[1-2]。

コンテナ実行でネイティブなネットワーク性能を出すため、MPI層ではなく通信フレームワークLibfabric (https://ofiwg.github.io/libfabric/)に手をいれるアプローチもある[1-3]、Libfabricは、MPIライブラリとネットワークハードウェアの間のミドルウェアで、異なるプロバイダをサポートすることができる、LibfabricとAWSの

OFI NCCLプラグイン(https://github.com/aws/aws-ofi-nccl)を用いることで、NVIDIAのNGCコンテナでのAllReduceが通常のコンテナより高性能実行ができることを示していた、また、CDI (Container Device Interface)と呼ばれる、コンテナがアクセスできるホスト上のデバイス情報（GPUなど）を記述するための仕様がある[1-4]、コンテナランタイムがCDIファイルの内容をもとにコンテナ構成を更新することで、コンテナ内のアプリが直接ホスト上のデバイスを利用できるようになる、Singularity、 Podman、 Cri-o、 Containerd、 Docker はCDI対応済または対応予定である。

[1-1] WI4MPI - HOW TO IMPROVE CONTAINER PORTABILITY, MARC JOOS (CEA), HPCW'24, 2024.
[1-2] MPI APPLICATION BINARY INTERFACE (ABI) STANDARDIZATION, JEFF HAMMNOND (NVIDIA), HPCW'24, 2024.
[1-3] CONTAINERS AND LIBFABRIC, ALBERTO MADONNA (ETH/CSCS), HPCW'24, 2024.
[1-4] CDI - THE CONTAINER DEVICE INTERFACE, EDUARDO ARANGO (NVIDIA) , HPCW'24, 2024.

[KubernetesとHPC／バッチスケジューラの連携]

クラウドの基盤ミドルウェアのデファクトスタンダードと考えられているKubernates (K8s)とHPCのバッチスケジューラを連携させ、デジタルツインのようなデータフローとHPCの連携やリアルタイムAI処理などを高速に実行可能にするための基盤技術が多数開発されている。

ドイツのGWDGでは、LLM推論サービスを提供するChat-AIシステムを開発しており、Webベースのフロントエンドから必要に応じてSSH経由でSlurmにジョブを投入することができる[2-1][2-2]、カスタムモデルのサブミット、セキュリティのための隔離、地理的冗長化等のため、Scalable Kubernetes Inference Platform (SKIP)を開発している、ユーザのモデルやランタイムに悪意のあるコードが含まれる可能性があり、隔離実行のためにHigh levelのruntimeはContainerd、Low levelはgVisorを用いている、gVisorは動的確保、GPU pass-throughをサポートしている、エフェメラルクラスタ用のPodmanベースのWarewulf Containerも開発している。

Sandia NLでは、K8sとOpenShiftでコンテナオーケストレーション as a Serviceを行ってきたが、HPCを含めた多様な計算をシームレスに行えるようなComputing-as-a-Serviceを実現するため、SlurmとK8sを統合したフレームワークK-Foundryを開発している[2-3]、K8s-likeなAPIを提供し、インタラクティブや応答性の高いタスクを実行できる、また、バッチジョブはK8sのジョブとして投入される、今後、FluxとFuzzballに対応する予定である。

Do IT NowのGarcia氏は、K8sとバッチスケジューラとのギャップを埋めるプロジェクトとして、HPK (High-Performance Kubernetes、 https://github.com/CARV-ICS-FORTH/HPK)、SUNK (Slurm in Kubernetes) (https://www.coreweave.com/blog/sunk-slurm-on-kubernetes-implementations)、Flux (https://flux-framework.org/)を挙げていた、また、ブリッジオペレータとしてslurm-k8s-bridge (slinky) (https://gitlab.com/SchedMD/slinky/slinky-sched-plugin)、hpc-connector (https://gitlab.com/primageproject/hpc-connector)、 Torque-Operator (https://link.springer.com/book/10.1007/978-3-031-40843-4)、Bridge-operator (https://arxiv.org/abs/2207.02531)などがある、彼らも、クラスタ配備とコンフィグ管理を行い、 Slurm / K8s / Warewulf Clusterな環境構築とワークフロー管理ツールを提供している[2-4]。

CSCSのMujkanovi氏は、EUのHPC-Cloudプロジェクトを紹介していた[2-5]、SlurmとK8sのインテグレーションを行うLEXIS、K8sとSlurmのブリッジやワークフローサポートを行うCYBELE、K8sクラスタにSlurmを構築するEVOLVE、HPC機能を統合するクラウドスタックを提供するOpenCUBE、Cray EXのHPC-Cloud対

応機能を提供するOpenCHAMI (Open Composable Heterogeneous Adaptable Management Infrastructure)などを挙げていた、OpenCHAMIは、クラウド的にHPCシステム管理ソフトウェアを開発するための標準であり、Software-Defined Clusterを開発している。

K8s上でジョブキュー用APIとコントローラを提供するKueueというプロジェクトもある[2-6]。Kueueでは、ジョブ開始時にポッドを作成し、停止時にポッドを削除する、今後、preemptionとcheckpointing、割当手法のフレーバ、クラスタオートスケーラ、予算管理、ダッシュボードやマルチクラスタサポート機能を提供予定である、

他にも、K8s基盤上で計算機クラスタの構築・管理をする需要が高まっており、Open Cluster Management (OCM)、Clusternet、Kubernetes Fleet Manager、Karmadaなどのプロジェクトがあるが、その手続きが標準化されていない、よって、クラスタインベントリ（設定ファイルに相当するもの）を定義するための標準となるClusterProfile API [2-7]がコミュニティで議論されている。

K8sをインフラ・ミドルウェアとするプロジェクトが多数であるが、OpenStack上にSlurmを構築するプロジェクトとしてStackHPCも紹介されていた[2-8]、国内でも、国立情報学研究所ではOpenHPCをベースとするHPCクラスタを主要なクラウドやmdx、オンプレで構築するための支援ツールを提供している[2-9]。

[2-1] AI/ML: SLURM-BASED LLM CHAT SERVICE, JONATHAN DECKER (GWDG, GERMANY), HPCW'24, 2024.

[2-2] PATH TO LOW-LATENCY HPC CONTAINERS FOR AI, JONATHAN DECKER (GWDG, GERMANY), HPCW'24, 2024.

[2-3] K-FOUNDRY: USING KUBERNETES-LIKE CONTROL PLANES WITH CUSTOM RESOURCE DEFINITIONS TO DEPLOY CONTAINERIZED HPC APPLICATIONS ACROSS MULTIPLE COMPUTING PLATFORMS, A. M. BELTRE (SNL) , CANOPIE-HPC 2024, 2024.

[2-4] CONTAINERS: KEYSTONE IN CONVERGED COMPUTING, ALBERTO GARCIA (DO IT NOW), HPCW'24, 2024.

[2-5] CONVERGENCE AND HETEROGENEITY NINA MUJKANOVIC, CSCS, HPCW'24, 2024.

[2-6] KUEUE: HTTPS://GITHUB.COM/KUBERNETES-SIGS/KUEUE

[2-7] CLUSTERPROFILE API: HTTPS://GITHUB.COM/KUBERNETES/ENHANCEMENTS/BLOB/MASTER/KEPS/SIG-MULTICLUSTER/4322-CLUSTER-INVENTORY/README.MD

[2-8] STACKHPC, MODUS OPERANDI, HPCW'24, 2024.

[2-9] 学認クラウドオンデマンド構築サービスのアプリケーションテンプレート: HTTPS://GITHUB.COM/NII-GAKUNIN-CLOUD/OCS-TEMPLATES

[科学ワークフロー実行と応用技術]

多くの分野でコンテナベースの科学ワークフロー実行を支援する仕組みを構築している、また、それに伴う応用技術について紹介する。

NOAAでは、GFDL気象・天候モデルのワークフロー実行基盤（FRE）を20年以上前から開発しており、簡単なコマンドでモデルの実行や後処理ができるようになっている[3-1]、PythonベースのFRE makeツールでdockerfileを生成し、Podman buildを実行できるようにしている、ビルド時間や実行時間がベアメタルに比べ遅い、各領域の科学者への教育が必要といった課題もあるが、2026年にはコミュニティで利用できるようにする予定である、CANOPIE-HPCでの発表[3-2]では、MPI bindingを利用することでベアメタルと同等の性能が得られることを示すなどの進展があった。

HPLabsとOak Ridge NLでは、ServerlessでHPC科学ワークフロー(WF)の実行・評価を行うフレームワークを、K8sのKnativeワークフローマネージャとWfCommonsフレームワークを拡張して実装している[3-3]、ベアメタルとServerlessでWFの実行時間、消費電力、CPU/メモリ利用量を比較し、Serverlessでは低オーバーヘッドで資源の有効活用ができることを示した。

NERSCでは、HPC用コンテナとしてShifter、 Podman-HPC、 Apptainerをサポートする他、 DMTCP (Distributed MultiThreaded Checkpointing)を利用してコンテナでCheckpointing/Restarting (C/R)を実装し、NERSCのPerlmutterで利用できるようにしている[3-4]、DMTCPのGPUプラグインを開発し、GPUコンテナにも対応予定である。また、Perlmutterでのコンテナ実行のベストプラクティスとして、CPU pinningやGPU仮想化による性能最適化、イメージのサイズと権限の最小限化と脆弱性スキャンによるセキュリティ対策、監視やロギング、監査の仕組みの必要性が報告されていた[3-5]。

HPEのBrayford氏はコンテナを利用したワークフロー実行に関する取り組みを紹介した[3-6]、コンテナベースでHPCアプリ実行環境を構築するPeriscope Tuning Framework、コンテナ＋MPIで千台規模の並列化を可能にする分散AIフレームワーク、QuantEx分散量子シミュレータのワークフロー、バイオインフォマティクスのワークフローなど、多くの分野でコンテナが活用されている。また、医療分野では暗号化されたセキュアなエンクレーブ（コンテナランタイム）が必要であり、そのためのコンフィグレーションの難しさや性能劣化が課題となっていることを挙げていた、CANOPIE-HPCでは、AMD SEV-SNPで秘密計算の性能への影響も定量的に示していた[3-7]。

セキュリティの観点では、HPCではコンテナに特権ユーザ権限を与えないで実行するように配慮されているが、PriedhorskyらはHPCではLinuxの特権操作が実際に実行されなくてもよいと仮定して、Linux seccompフィルタでルートエミュレーションモードを実装した[3-8]、このフィルタでは、一部の特権システムコールをインターセプトして何もせずに成功を返すようにしている、このアプローチは一貫性がないが、多くのDockerfileがビルド可能で、HPCアプリコンテナに必要な完全に非特権のワークフローを簡単に実現できるようにしている。

参考文献

[3-1] Climate Model Container Building for Domain Scientists, Tom Robinson (NOAA) , HPCW'24, 2024.

[3-2] Comparative Analysis of a Containerized Compilation Process Versus Bare-Metal Compilation Runs, D. Singh, T. Robinson (NOAA), CANOPIE-HPC 2024, 2024.

[3-3] Enabling HPC Scientific Workflows for Serverless, A. A. D. Silva (HPL), 他, CANOPIE-HPC 2024, 2024.

[3-4] Containerized Checkpoint-Restart Mechanisms for HPC, M. Timalsina (LBNL), N. Tyler (NERSC), CANOPIE-HPC 2024, 2024.

[3-5] Containerization in HPC Environments: Challenges in Optimizing performance and security in Scientific Computing, V. Saravanan (U. Texas), K. Z. Ibrahim (LBNL), CANOPIE-HPC 2024, 2024.

[3-6] Other Workflows, David Brayford (HPE), HPCW'24, 2024.

[3-7] Deploying Containers on Secure HPC Systems, D. Brayford (HPE), CANOPIE-HPC 2024, 2024.

[3-8] Zero-Consistency Root Emulation for Unprivileged Container Build, Reid Priedhorsky (Los Alamos NL)他, CANOPIE-HPC 2024, 2024.

3.2.7.3.2　OS、仮想化、クラウド連携に関して次世代計算基盤において取り組むべき課題

欧米の大規模スパコンセンターにおいてコンテナ技術の活用がますます進められおり、ポータビリティや性能向上のための技術の標準化も進んでいることが明らかとなった。また、デジタルツインや生成AIといった多様なアプリケーションへの対応のための基盤としてK8sとHPCの連携技術の開発も盛んに行われており、様々な科学ワークフロー実行のための技術も成熟しつつある。一方、国内でのHPCのためのコンテナ活用は遅れを取っており、富岳NEXT開発においてコンテナ技術の高度な活用が期待される。

3.2.8　セキュリティ

3.2.8.1　調査目的

本調査研究では、システムソフトウェア・ライブラリの技術を俯瞰し、サブグループを横断してセキュリティの面での調査研究を実施する。システムソフトウェア・ライブラリに関する各サブグループと密接に連携し、各サブグループがまとめた技術動向調査に関してシステム全体からの視点での評価指標の選択や優先順位の検討を取りまとめていく。加えて、複数のサブグループをまたぐマルチレイアに渡る検討が必要なセキュリティの面での考察を、サブグループを横断する形で実施する。

3.2.8.2　調査内容

セキュリティの技術分野の特徴としてシステムソフトウェア・ライブラリに関する要素技術について横断的に検討する必要があるという特性があるため、システムソフトウェア・ライブラリのセキュリティに関連する項目を先導的に調査検討する。また、要素技術にかかわる調査研究だけではなく、システム全体からの視点で検討することを試みる。アプリケーションを実行する個々のユーザの視点で必要とするセキュリティレベルを確保するための暗号化、改ざん防止技術（ブロックチェーン）等に関してのソフトウェア技術を調査し、演算性能やI/O性能等に与える因子を評価しながらソフトウェア技術全般に関わる横断的な評価と検討を進める。

3.2.8.3　調査結果

3.2.8.3.1　Arm CPU のセキュリティ技術の歴史的進化と技術動向

組込み分野やIoT、マイコンの分野などで伝統的に利用されてきたArm CPUのセキュリティ技術の歴史的流れや最新の技術動向について、セコム株式会社IS研究所の研究員からヒアリングを行った。そのヒアリングの内容に技術的な補足をすることにより本調査研究項目を進めた。

ArmのCPUにおいてセキュリティ技術が登場したのは、2002年にArm v6向けに発表されたTrustZoneという機能が始まりであった。TrustZoneは、ハードウェアでメモリ空間へのアクセスを制御することにより論理的にハードウェアのリソースを分離するしくみである。TrustZoneは、CPUおよびOSのゾーンをSecureとNon-secureに論理的に分ける機能を提供することを通してセキュアな処理の根本となるルート・オブ・トラスト（Root of Trust）を提供する。TrustZoneのSecureゾーンにおいてTEE（信頼された実行環境）を実現するためには、セキュアな方法でブートするOSや信頼された実行環境TEEにNon-secureなゾーンからアクセスするAPIが必要になる。それらTEEを実現するためのオープンソースの実装の代表例として、OP-TEEが普及している。OP-TEEにより、システムをSecure WorldとNormal Worldに分離して、Normal WorldからSecure Worldへのアクセスをコントロールし限定的にすることにより、Normal Worldからの攻撃から保護することが可能となる。

TrustZoneは、歴史的に組込み分野やIoT、マイコンの分野などへの応用を想定して設計されていたため、非力な組込み向けCPUやマイコンでの利用を前提としたセキュリティ技術であった。このため、Secure Worldでは、Secure worldの管理者による信頼されたアプリケーションのみが動作する前提であった。一方で、VMやコンテナベースの仮想化技術が進展し、Secure worldでユーザが独自の管理者からは信頼されていないアプリケーション（3rd partyのアプリ）を動作したいという要求も高まってきた。このような要求に対しては、x86ベースのサーバ環境においてはAMD SEVのような仮想化技術基盤であるハイパーバイザと連携しつつ対処する技術が開発されるに至った。このような状況の中で、Arm環境においてもVM技術やハイパーバイザと連動できるセキュアな仕組みの開発への機運が高まっていった。

Arm CCA（Confidential Compute Architecture）は2021年にArm v9.0向けに発表されたセキュリティの機能であり、Arm環境においてもVM技術やハイパーバイザと連動できるセキュアな仕組みを提供する。CCAはTrustZoneでは実現できなかった3rd partyのアプリをTEEで動くように、実行環境をRealmとして分離する。VM

がハイパーバイザを通してソフトウェアレベルで仮想化された実行環境であるとみなすとすると、Realmはそれより低水準なファームウェアレベルでハードウェアが分離された実行環境とみなすことができる。また、これまでArmが得意としてきた組込みやマイコン分野への応用だけではなく、クラウドのデータセンターで稼働する仮想サーバやHPC環境に直接的に展開することも可能な機密性を備えたセキュリティ技術である。しかしながら、2025年2月現在のところArm CCAを実装したCPUは製品化されておらず、シミュレーション環境等による機能のデモンストレーションがされているにとどまっている。

CPUコア内、あるいは、メモリコントローラなどのシステムとしてSoCチップに集積される機能に関しては、Intel TDXやAMD SEVなどVMと連携する製品が市場に投入されている状況となっている。また、ArmにおけるCCAの動向に加えて、RISC-VにおいてもAP-TEEと呼ばれるVM連携型のTEEも規格化が進んでおり、同一SoCチップ内にある環境は、機密環境が普及しつつあるといえる。一方で、機密コンピューティングの適応を進めるにつれ、PCIeに代表されるIOバスを経てデータが転送される環境での機密環境が必要となるアプリケーションも散見される。そこで、PCIeの先にあるデバイスとの通信を含め機密環境を実現するためのプロトコルとして、PCI TEE Device Interface Security Protocol (TDISP)という規格が2022年に標準化され、通信経路を含む機密環境の構築が進行中である。これまでに、x86系のCPUにおいては、AMD SEV-TIO、Intel TDX Connectのようにに TDISPにより直接的にデバイスをセキュアに割り当てる規格が発表されているが、2025年2月現在、これらの機能を実装したCPUは製品化されていない。Arm CCAにおいても、TDISPをサポートした通信経路によりGPUやアクセラレータを直接的にTEEに割り当てることができるように実装が進んでいくと予想される。

CPUやSoCの外にあるデバイスをセキュアに利用するためには暗号化ハードウェアと連携して高速化していくことは有用である。一方で、暗号化と安全な鍵の交換だけでは、信頼された実行環境TEEの内部で悪意のあるデータ書き換えが起こった場合に対しては無力となる。もし、そのような意図しない書き換えによりデータに不整合が発生すれば、システムが暴走する、あるいは、暗号が解けなくなるなどの影響が考えられる。そこで、意図しないデータ書き換えが起こっていないことを担保するための監視を行う手法として、メモリのインテグリティチェックが注目されている。インテグリティチェックは、2015年に発表されたIntel SGXにてサポートされた機能で、AES暗号されたデータブロックとして主記憶とキャッシュメモリの間のメモリイベントに対してキャッシュラインを単位としたメモリイベントのメタ情報を木構造のリーフに格納し、ルートに向かってハッシュ値を取っていくMerkle Tree（ハッシュ木）に基づくデータ構造をハードウェア上で構築していくことにより、メモリの整合性を監視する。Merkle Treeにおいては、メモリの内容がアップデートされるごとに、対応するリーフとその上流にあるノードのすべてのハッシュ値を更新する。Merkle Treeのルートを外部からアクセスできない領域に保存することにより意図した書き換え以外の状態の変化を検出することにより安全性を担保する。しかしながら、Merkle Treeを更新する処理の負荷のため、メモリ読み書き全般の読み書きが遅くなる。そのため、確実にデータを守る必要がある重要なところのみをエンクレーブ領域として保護するという発想の下で、当初は90MBぐらいの小さなエンクレーブ領域のみしか構築できなかった。エンクレーブ領域で様々な処理を行いたいというニーズから、領域のメモリサイズが徐々に拡大する方向でSGXの機能は進化を続けたが、次第に処理の面でのオーバーヘッドの問題が課題となってきた。そこで、第3世代のIntel Xeon Scalable Processorと呼ばれる世代から、Total Memory Encryption Multi-Key (TME-MK)というハードウェアによるメモリ暗号化とMessage Authentication Code (MAC)を利用したインテグリティチェックにより、オーバーヘッドの課題の解決を図り、エンクレーブ領域のメモリを大きくできるように方向性を変更した。Intel TDXにおいても、TME-MKを用いたハードウェアレベルでのインテグリティチェックが提供されている。一方、AMDのCPUにおいては、SEV-SNPと呼ばれるメモリのページレベルでインテグリティチェックを行う機能が実装され、2021年に製品化されている。Armにおいては、現状ではメモリインテグリティに関する機能の提供は2025年2月時点では製品化されていない。CCAの規格化においてIntegrityに関する記述もあることから対応が進んでくると予想される。

3.2.8.3.2　GPU 環境でのセキュリティ技術

GPU環境でのセキュリティ技術について、文献を中心に技術動向の調査を行った。GPUにおいて、機密コンピューティングに対応する製品は、2022年にリリースされたNVIDIAのH100が初めてであった[CACM 2024]。機密コンピューティングの実現には、データやコードが攻撃者から読まれないようにする秘匿性と、攻撃者から実行環境が改ざんされないようにするインテグリティの確保が必要となる。H100においては、CPU側に機密性のあるVMを作り、GPUをその機密VMからのみ機密性を保ちつつアクセスできるように通信経路も暗号化して、TEEを形成することにより機密コンピューティング環境の形成を実現した。しかしながら、現状では、CPU側は通信経路の暗号化をソフトウェアで実行しているため、CPU/GPU間のバンド幅が4GB/sとなることが報告されている。GPUへのデータの転送が少ないアプリケーションにおいては、GPUでの計算が支配的となりCPU/GPU間データ転送は隠蔽されるが、GPUへのデータ転送が多いアプリケーションでは機密コンピューティングを実現するための性能オーバーヘッドが観測されている[CACM 2024]。一方で、将来的にPCIeの先にあるデバイスとの通信を含め機密環境を実現するためのプロトコルTDISPに準拠する暗号化ハードウェアが搭載されるようになると、機密コンピューティングによる性能オーバーヘッドは大幅に小さくなるであろうと予想される。現状ではH100の機密コンピューティングはIntel TDX、AMD SEV-SNPでの動作がサポートされている。

ArmのCPUの環境においては、GPUをシステムに含む場合の機密コンピューティング環境が現時点ではサポートされている製品は発表されていない。従って、マルチテナントで複数ユーザがGPUやアクセラレータを利用するとき、アクセレレータを含めたEnd-to-endのデータ機密性は十分に担保できない状況にある。2021年に発表されたArm CCAの機能は、Data-in-useのフェーズを中心として、CPUやメモリにおける機密性を担保することを主な目的としてきた為、PCIeバスの先にあるアクセラレータやネットワークと接続する場合のData-in-motionのフェーズの機密性までは十分なサポートがされてこなかった。他方で、PCIe-5にて規格化されたIDE (Integrity and Data encryption)という機能によりPCIレベルのデータ転送における暗号化やインテグリティチェックがサポートされたことを利用して、PCIeも含めてTEE環境を構築できるプロトコルレベルの規格であるPCI TEE Device Interface Security Protocol (TDISP)をサポートするような拡張を行うことにより、効率的にアクセラレータの実行環境のセキュリティを担保できることが[USENIX Security 2024]にて示されている。

[CACM 2024] G. Dhanuskodi et al. Creating the First Confidential GPUs. Communication of ACM, Jan. 2024, pp. 60–67.

[USENIX Security 2024] S. Sridhara et al. ACAI: Protecting Accelerator Execution with Arm Confidential Computing Architecture.

3.2.8.4　開発項目

横断的セキュリティに関する開発項目として、次の項目が挙げられる。

- スーパーコンピュータ利用における性能を重視する運用を踏まえると機密コンピューティングは性能とのトレードオフもあるため、ユーザが選択的に利用できる環境をシステムソフトウェアの面から整備すること
- GPU に代表される演算加速器やノード間通信における機密コンピューティング技術の実現

3.2.9　自動チューニング

3.2.9.1　調査目的

自動チューニング(Auto-tuning、AT)技術は、アルゴリズム、ソフトウェア、およびハードウェアに現れる性能パラメータを調整する技術であり、実行時や入力データに基づく性能向上の手段として有効な技術であることが知られている。本調査では、数値計算ライブラリ（アルゴリズム）を中心として、アプリケーションからシステムソフトウェアまでのAT適合性や有効性を調査し、次世代システムの性能見積もり、チューニング、利便性に資することを目的とする。

以下の2目的がある。

- 目的１：サブグループの縦串としての AT 技術調査
- 目的２：AT 最新技術調査

3.2.9.2　調査内容

- 目的１：サブグループの縦串としての AT 技術調査

[AT適用可能性の調査]：

FSで定めるベンチマーク調査やアプリケーションのチューニングを通して、AT適用可能な性能パラメータを調査する。加えて、既存のATソフトウェアについての性能調査を行う。

- 目的２：AT 最新技術調査

[GPUアクセラレータ活用を前提としたシステムにおけるAT機能の調査]：

GPUアクセラレータ活用に対するAT機能の研究、および、ATに関連するフリーソフトウェアを調査し、次期システムへの適用可能性を調査する。

[新奇ハードウェアのAT]：

新奇ハードウェアとして想定される量子コンピュータ（含む、関連技術）におけるAT適用可能性と効果を調査する。

3.2.9.3　調査結果

- 目的１：サブグループの縦串としての AT 技術調査

[AT適用可能性の調査]：

本年度は、科学技術計算の基本演算に対するAT機能を調査した。ここでは、GPUアクセラレータに限定しないコンパイラ最適化技術の一環として、Clang や LLVMとATの関連論文を調査した。調査結果を表 3.2.9.1に示す。

表 3.2.9.1では、ATの対象としてはループが多い。また、ループ交換やループに現れるタイルサイズなどの性能パラメータを変更するAT機能を提供している。一方で、プラグマやディレクティブでATの指示を与える、「注釈ベース」のATシステムが多い。演算処理の対象は、疎行列積など科学技術計算の中核をなすカーネルを対象としているものが多いが、LU分解やFDTD法など、アプリケーションソフトウェア側に近い数値計算処理を対象にしているものも存在する。

表 3.2.9.1 Clang や LLVM を用いた AT 方式

対象	概要	方式	要求システム	備考
ループ変換 [1]	ディレクティブベース、ループ交換、ループタイルサイズ	-	Polly's ビルトイン BLIS 最適化	
SOLLVE プロジェクト開発ブランチ for LLVM [2]	機械学習ベースのオートチューナー、Clang/Polly ループ最適化プラグマ、ループタイリング、ループ交換、アレイパッキング、複数ループ変換	ベイジアンベースの自動チューニング	Yopt	PolyBench ベンチマーク
LLVM-IR [3]の自動チューニング	実プログラムの低レベル IR 表現の自動 polyhedral 最適化ツール	LLVM 中間言語表現 (LLVM-IR)、Polyhedral 最適化	Polly (clang と gcc)	PolyBench 2.0 benchmark suite
OpenMP ベースの並列化コード[4]	注釈言語、経験的性能チューニングシステム (注釈付き C ソース コードを入力として、注釈付きコードの最適化されたコード候補を多数生成)	Orio 注釈アプローチ	Orio, PluTo (オートチューナー)	AXPY 疎行列-ベクトル積 (CSR) 疎行列演算(PETSc) 2 次元有限要素法時間領域 (FDTD) LU 分解

表 3.2.9.1上の参考文献

[1] Michael Kruse, Hal Finkel, Xingfu Wu, Autotuning Search Space for Loop Transformations, Proc. of 2020 IEEE/ACM 6th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC) and Workshop on Hierarchical Parallelism for Exascale Computing (HiPar), pp.12-22 (2020) DOI Bookmark: 10.1109/LLVMHPCHiPar51896.2020.00007

[2] Xingfu Wu, Michael Kruse, Prasanna Balaprakash, Hal Finkel, Paul Hovland, Valerie Taylor, Mary Hall, Autotuning PolyBench benchmarks with LLVM Clang/Polly loop optimization pragmas using Bayesian optimization, Concurrency and Compuataion: Practice and Experience (08 November 2021) https://doi.org/10.1002/cpe.6683

[3] TOBIAS GROSSER, ARMIN GROESSLINGER, and CHRISTIAN LENGAUER, POLLY – PERFORMING POLYHEDRAL OPTIMIZATIONS ON A LOW-LEVEL INTERMEDIATE REPRESENTATION, Parallel Processing Letters, VOL. 22, NO. 04,(2012) https://doi.org/10.1142/S0129626412500107

[4] Albert Hartono, Boyana Norris, P. Sadayappan, Annotation-based empirical performance tuning using Orio, Proc. of 2009 IEEE International Symposium on Parallel & Distributed Processing, (2009) DOI: 10.1109/IPDPS.2009.5161004

- 目的２：AT 最新技術調査

[GPUアクセラレータ活用を前提としたシステムにおけるAT機能の調査]：

ここでは、次期システムへの展開を考慮し、GPUアクセラレータを対象としたAT機能調査を行った。調査の前提として、他グループで調査を行っている機械学習関連のATは除いた。また、タイトルに「auto-tuning」が明示されている論文に限定に調査した。

本調査では、対象の処理を以下の4つに分類して調査を行った。

1. 基礎線形代数計算（BLAS、密行列）
2. 疎行列演算（SpMV、ステンシル）
3. FFT
4. システム関連

表 3.2.9.2に、調査結果を示す。

表 3.2.9.2 GPU アクセラレータを前提とした AT 機能の調査結果

分類	対象処理	AT 対象	AT 方式	要求システム	備考
基礎線形代数計算（BLAS、密行列）	GEMM (LU) [1][2][4][5][25] GEMM (Cholesky) [6]	ブロックサイズ (Update: GPU, Other: CPU) エネルギー [25]	モデルベース [1] 全探索 [2][5] 確率修正再書込ルール[4] ベンチマークテスト(全探索)[25]	cuBLAS MAGMA[2] OpenCL[4] BEAST[25]	Fermi データパス[2], K40、M40、P100[6] Kepler K20[25]
	SNRM2, SGEMV, STRSV[3]	ブロックサイズ[3] スレッドブロック[3] アンローリング段数 [3] アルゴリズムトレードオフ [3]	全探索[3]	CUDA	Tesla[3]
疎行列演算(SpMV、ステンシル)	SpMV[7][10]	CRS[7], ELLR-T[10], ブロックサイズ [7], スレッド[7], Warp 数[7], パティング [10]	全探索 [7] モデルベース[10]	CUDA	GeForce 9500 GTX[7] GeForce GTX 285[10]
	7 点・13 点ステンシル[8] 3 次元ステンシル [25]	ブロックサイズ (2 次元、3 次元)[8][25] ループアンローリング [25]、インプレースかフォワードプレース[8] スレッドブロック [9][10]、ブロック・Warp・L1・アンローリング [9] 疎行列データ構造 (DIA, CRS) [9] ロードバランシング [25] 通信方法[25]	モデルベース (全探索)[8] Orio(コード生成に関する経験的手法)[9]、全探索[25]	CUDA、Orio、PETSc[9]	GeForce GTX680, Tesla C2070[8] Tesla C2070、Tesla C1060 [9] GTX 275、C1060[25]
FFT	FFT	Kernel radix [11][13][14]、 スレッドブロック [11][13][14]、 パティング[11]、 行と列のカーネル [14]	全探索[11][13] (経験的手法 [14])、 インストール時 (コード生成)、および実行時 [13]	CUDA・OpenCL[13] UHFFT、FFTW[13]	GeForce GTX 285、480 [11] Tesla C1060、C2050 ATI 6870、5850 [13] NVIDIA GTX280、GTX260、8800GTX[14]
	Non-Uniform Fast Fourier Transform (NUFFT)	オーバーサンプリング率[12]	全探索[12]	cuFFT、cuSPARSE[12]	Tesla K40[12]
システム関連	プログラム (ディレクティブベース)	アンロール[19][27], タイリング、分割 [19]、スレッド数 (CPU と GPU)、スレッドアフィニティ、格子構成[20]、	HMPP (全探索)[19] PCPASEARCH[20] Nearest Neighbor [22]、	HMPP[19] CUDA、OpenCL[19] OpenMP+MPI[20] OpenACC[22]	Tesla C2050[19][27] GTX 1070, 1060、Tesla K20m [20]

		OpenACC (gang, vector)[22] CPU-GPU 選択[27]	FIBER (モデルベース) [27]	ppOpen-AT, OMPCUDA[27]	
	Domain Specific Language (DSL)	カーネル、Warp、およびスレッド[28]	SA, GA, Hill climbing [28]	CUDA、nvcc[28]	Tesla K40[28]
	最適化アルゴリズム	アプリケーションのハイパーパラメータ[21]	Amortized ベイズ最適化[21]	SciKit-Opt, Hyper-Opt[21]	RTX 2070 Super, A100[21]
	混合精度演算	アプリケーションレベルの精度チューニング(最大と、ユーザ指定の閾値より小さい誤差)[23][24]	遺伝的アルゴリズム(GA)[23] FISet サーチ[24]	CUDA LULESH[23][24]	FP64, FP32 or FP16[23]
	消費電力	ワークグループサイズ、スレッド当たりのワークグループ、キャッシュ、ベクトル幅、ループアンローリング、パディング[29]	MOKAT (GA)[29]	CUDA, OpenCL[29] NVIDIA NVML[29]	

表 3.2.9.2上の参考文献

[1] Gregorio Bernabe, Javier Cuenca, Luis-Pedro Garcia, and Domingo Gimenez, Tuning Basic Linear Algebra Routines for Hybrid CPU+GPU Platforms, ICCS 2014, Volume 29, Pages 30–39 (2014)

[2] Jakub Kurzak, Stanimire Tomov, Jack Dongarra, Autotuning GEMM Kernels for the Fermi GPU, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 23, NO. 11, pp.2045-2057, NOVEMBER 2012 2045

[3] Hans Henrik, Brandenborg Sørensen, Auto-Tuning of Level 1 and Level 2 BLAS for GPUs, Concurrency Computat.: Pract. Exper. 2012; 00:1–23 (2012)
DOI: 10.1002/cpe

[4] Michel Steuwer, Toomas Remmelg, Christophe Dubach, Matrix multiplication beyond auto-tuning: Rewrite-based GPU code generation, 2016 International Conference on Compliers, Architectures, and Sythesis of Embedded Systems (CASES), (October 2016), DOI: 10.1145/2968455.2968521

[5] Yinan Li, Jack Dongarra, Stanimire Tomov, A Note on Auto-tuning GEMM for GPUs, Proc. of Computational Science – ICCS 2009, pp. 884–892 (2009)

[6] JACK DONGARRA, MARK GATES, JAKUB KURZAK, PIOTR LUSZCZEK, YAOHUNG M. TSAI, Autotuning Numerical Dense Linear Algebra for Batched Computation with GPU Hardware Accelerators, PROCEEDINGS OF THE IEEE, Vol. 106, No. 11, pp. 2040-2055, (November 2018)

[7] Ping Guo, Liqiang Wang, Auto-Tuning CUDA Parameters for Sparse Matrix-Vector Multiplication on GPUs, Proc. of 2010 International Conference on Computational and Information Sciences, (2010) DOI: 10.1109/ICCIS.2010.285

[8] Wai Teng Tang, Wen Jun Tan, Ratna Krishnamoorthy, Yi Wen Wong, Shyh-Hao Kuo, Rick Siow Mong Goh, Stephen John Turner, Weng-Fai Wong, Optimizing and Auto-Tuning Iterative Stencil Loops for GPUs with the In-Plane Method, Proc. of 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS), ()2013 DOI: 10.1109/IPDPS.2013.79

[9] Azamat Mametjanov, Daniel Lowell, Ching-Chen Ma, Boyana Norris, Autotuning Stencil-Based Computations on GPUs, Proc. of 2012 IEEE International Conference on Cluster Computing (CLUSTER12), pp. 266 – 274, (2012) DOI: https://doi.org/10.1109/CLUSTER.2012.46

[10] Francisco Vázquez, José Jesús Fernández, Ester M. Garzón, Automatic tuning of the sparse matrix vector product on GPUs based on the ELLR-T approach, Parallel Computing, Volume 38, Issue 8, pp. 408-420 (2012)

[11] Akira Nukada, Satoshi Matsuoka, Auto-tuning 3-D FFT library for CUDA GPUs, Proc. of SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, No.: 30, pp. 1 – 10, DOI: https://doi.org/10.1145/1654059.165409

[12] Teresa Ou, gNUFFTW: Auto-Tuning for High-Performance GPU Accelerated Non-Uniform Fast Fourier Transforms, Technical Report No. UCB/EECS-2017-90, (May 12, 2017) http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-90.html

[13] Yan Li, Yun-Quan Zhang, Yi-Qun Liu, Guo-Ping Long, Hai-Peng Jia, MPFFT: An Auto-Tuning FFT Library for OpenCL GPUs, Journal of Computer Science and Technology, Volume 28, pp. 90–105 (2013)
[14] Yuri Dotsenko, Sara S. Baghsorkhi, Brandon Lloyd, Naga K. Govindaraju, Auto-tuning of fast fourier transform on graphics processors, ACM SIGPLAN Notices, Volume 46, Issue 8, pp. 257 – 266, https://doi.org/10.1145/2038037.1941589
[15] Nicolas Weber, Michael Goesele, Auto-Tuning Complex Array Layouts for GPUs, Proc. of Eurographics Symposium on Parallel Graphics and Visualization, (2014) https://diglib.eg.org/handle/10.2312/pgv.20141085.057-064
[16] Robert Lim, Boyana Norris, Allen Malony, Autotuning GPU Kernels via Static and Predictive Analysis, 2017 46th International Conference on Parallel Processing (ICPP) (07 September 2017), DOI: 10.1109/ICPP.2017.61
[17] Richard Schoonhoven, Bram Veenboer, Ben Van Werkhoven, K. Joost Batenburg, Going green: optimizing GPUs for energy efficiency through model-steered auto-tuning, Proc. of 2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), pp. 48-59 (2022) DOI: 10.1109/PMBS56514.2022.00010
[18] Milo Lurati, Stijn Heldens, Alessio Sclocco, Ben van Werkhoven, Bringing Auto-tuning to HIP: Analysis of Tuning Impact and Difficulty on AMD and Nvidia GPUs, 30th International European Conference on Parallel and Distributed Computing (Euro-PAR 2024), arXiv:2407.11488 [cs.DC] (16 Jul 2024)
[19] Scott Grauer-Gray, Lifan Xu, Robert Searles, Sudhee Ayalasomayajula, John Cavazos, Auto-tuning a High-Level Language Targeted to GPU Codes, Proc. of 2012 Innovative Parallel Computing (InPar) (2012) DOI: 10.1109/InPar.2012.6339595
[20] Paweł Czarnul and Paweł Rosciszewski, Auto-tuning methodology for configuration and application parameters of hybrid CPU+GPU parallel systems based on expert knowledge, Proc. of 2019 International Conference on High Performance Computing & Simulation (HPCS), (2019) DOI: 10.1109/HPCS48598.2019.9188060
[21] Willemsen, F.-J.; van Nieuwpoort, R.; van Werkhoven, B., Bayesian Optimization for auto-tuning GPU kernels, Proc. of 2021 International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS 2021), (2021) DOI: 10.1109/PMBS54543.2021.00017
[22] Alberto Magni, Dominik Grewe, Nick Johnson, Input-aware auto-tuning for directive-based GPU programming, Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units (GPGPU-6), pp. 66-75 (2013) https://doi.org/10.1145/2458523.2458530
[23] Pradeep V Kotipalli, Ranvijay Singh, Paul Wood, Ignacio Laguna, Saurabh Bagchi, AMPT-GA: automatic mixed precision floating point tuning for GPU applications, Proc. of ICS '19: Proceedings of the ACM International Conference on Supercomputing, pp. 160-170 (2019) https://doi.org/10.1145/3330345.3330360
[24] Ignacio Laguna, Paul C. Wood, Ranvijay Singh, Saurabh Bagchi, GPUMixer: Performance-Driven Floating-Point Tuning for GPU Scientific Applications, Proc. of High Performance Computing (ISC High Performance 2019), LNCS, volume 11501, pp. 227–246 (2019)
[25] Hartwig Anzt, Blake Haugen, Jakub Kurzak, Piotr Luszczek, Jack Dongarra, Experiences in autotuning matrix multiplication for energy minimization on GPUs, Concurrency Computat.: Pract. Exper. 27:5096–5113 (20 May 2015) DOI: 10.1002/cpe.3516
[26] 渡辺 祐也、遠藤 敏夫、松岡 聡、GPUクラスタにおける科学技術計算の自動最適化、情報処理学会研究報告HPC、HPC124 (2010)
[27] 片桐 孝洋、大島 聡史、平澤 将一、本多 弘樹、HxABCLibScript:非均質計算機向け自動チューニング記述言語拡張 、研究報告ハイパフォーマンスコンピューティング（HPC）、Vol. 2011-HPC-129、No.13、pp. 1-8 (2011-03-08)
[28] Authors: Michael Vollmer, Bo Joel Svensson, Eric Holk, Ryan R. Newton, Meta-programming and auto-tuning in the search for high performance GPU code, Proceedings of the 4th ACM SIGPLAN Workshop on Functional High-Performance Computing (FHPC2015), pp. 1 – 11 (2015) https://doi.org/10.1145/2808091.2808092
[29] Sunbal Cheema, Gul Khan, GPU Auto-tuning Framework for Optimal Performance and Power Consumption, Proceedings of the 15th Workshop on General Purpose Processing Using GPU (GPGPU '23), pp.1 – 6, (2023) https://doi.org/10.1145/3589236.3589241

表 3.2.9.2から、AT対象としては、GPU上のブロックサイズやWarpサイズなどのGPUハードウェアパラメタを対象とするAT機能が多い。一方、GPUにおいても、プログラム上のループタイルサイズやループアンローリングを対象にしているものが多い。そのため、これらのコード最適化機能は、GPUでも必須となる機能であることがわかる。またAT方式としてはモデルベースのソフトウェアもあるが、多くは全探索を採用している。GPUに対するシステム要求では、CUDAベースのソフトウェアが多い。

表 3.2.9.3に、オープンソースソフトウェア（OSS）となっているATツールを示す。

表 3.2.9.3 オープンソースソフトウェア(OSS)となっている AT ツール

ソフト名	概要	AT 方式	要求システム	備考
OpenTuner	https://opentuner.org/ API 提供型(domain-specific multi-objective program autotuner)	multi-armed bandit with sliding window, area under the curve credit assignment (AUC Bandit)	python 3.7+、sqlite3	
ABCLibScript (ppOpen-AT)	http://www.abc-lib.org/format.htm ディレクティブ専用言語	多項式近似、d-Spline （ブラックボックス型）	cpp コンパイラ (gcc, HITACHI, Fujitsu, Intel and PGI)	
GPTune	https://github.com/gptune/GPTune API 提供型	ベイズ最適化 （ブラックボックス型）	OpenMPI (4.0 or higher) or CrayMPICH (8.1.23 or higher) Python (3.7 or higher) BLAS/LAPACK, SCALAPACK (2.1.0 or higher) Mpi4py	MPI に対応 DOE 計算機・ソフトをサポート
ytopt	https://github.com/ytopt-team/ytopt API 提供型	ベイズ最適化 （ブラックボックス型）	Python autotune, dh-scikit-optimize, ConfigSpace Synthetic Data Vault (SDV)	Synthetic Data Vault (SDV)によるオンラインチューニング
Optuna	https://optuna.org/ API 提供型	ベイズ最適化 （ブラックボックス型）	Python 3.7 or newer	DL ハイパーパラメータ最適化用

表 3.2.9.3から、Pythonベースや、Pythonから呼び出せるAPIを有するATツールが多い。また、AT方式では、ブラックボックス型が多い。そのため、方式としてはベイズ最適化ベースで実装されているOSSが多いといえる。これらは、ツールの対象を広げるには、対象のアプリケーション特性の制約を無くす必要があることから、ブラックボックス型のATが多いと予想される。また、探索空間が広くなる際の効率的探索から、ベイズ最適化を採用しているツールが多いと想定される。

一方、表 3.2.9.3のOSSについては、特殊なシステム要求がない。そのため、次期システムである「富岳NEXT」においても、これらのOSSは利用可能と思われる。

[新奇ハードウェアのAT]：

昨年は、量子関連技術へのAT適用として、サポートベクターマシン(SVM)への疑似量子アニーラへの適用について、テストデータについて線形分離可能と非線形分離可能のデータを採択し、誤分類を伴うデータセットでの評価を行った[1][2]。性能評価の結果、線形分離可能のデータについては、疑似量子アニーラと古典計算機によるSVMの解の精度は同等であることを明らかにした。

本年度は、疑似量子アニーラの性能評価に関して、複数のハードウェアを用いた性能評価を行うことで、詳細な性能調査を行った[3]。その結果、各社のハードウェアの違いで、得られる解の品質が異なることを明らかにした。

3.2.9.4 開発項目

ATソフトウェアに関する開発対象は、上記のGPTune、および、AIのハイパーパラメタチューニングで広く使われているOptunaを開発対象として検討する。

- ソフトウェア 1 ：GPTune
  米国 DOE で開発された、ベイズ最適化ベースの自動チューニングツールである。Python からの利用に加えて、MPI 版の実行(mpi4py)での並列実行機能も提供されている。主な対象は、数値計算ソフトウェアの性能パラメタチューニングであるが、AI のハイパーパラメタチューニングにも適用可能である。
- ソフトウェア 2：Optuna
  Preferred Networks 社の製品であり、ハイパーパラメータ自動最適化フレームワークである。ベイズ最適化ベースの最適化を採用しており、主な対象は AI のハイパーパラメタチューニングである。

参考文献

[1] 水木ほか、SVM による誤差を含むクラス分類における CMOS アニーリングマシンの性能評価、情報処理学会第 86 回全国大会、pp.1-2 (2024-03-15)

[2] R. Fukuhara ほか, Performance Evaluation of CMOS Annealing with Support Vector Machine, Proc. of IEEE MCSoC2024 (2024-12-18)

[3] 水木ほか、SVM による誤差を含むクラス分類における多種疑似量子アニーラの性能評価、情報研報、Vol.2024-HPC-198 (2025-3-18)(発表予定)

3.2.10　HPC 利用環境調査

3.2.10.1 調査目的

本サブグループ（HPC利用環境サブグループ）では、ポスト「富岳」時代の次世代計算基盤のあり方を検討・考察することを視野に、高性能計算（HPC：High Performance Computing）システムを構成するシステムソフトウェア・ライブラリの利用状況について調査を行うことを目的とする。今日においては高性能計算システムの利用は多岐にわたり、学術、産業の枠を超え、多様な方法で利用されている。そのため、ポスト「富岳」時代の次世代計算基盤においては、大規模・大容量なデータを扱うシミュレーションや数値計算だけでなく、今日急拡大傾向にあるAI（人工知能：Artificial Intelligence）、ML（機械学習：Machine Learning）、DL（深層学習：Deep Learning）に代表されるデータ分析からの計算需要・ニーズを考慮しなければならない。また、今日においては、大規模な科学計測機器、IoTセンサなどの多種多様なデータ源から発生するデータを速やかに高性能計算システムに収容し、高性能計算システムでの解析・計算処理に利用したいという状況もある。IoTセンサ等で取得された観測データを計算シミュレーションの中間結果と比較し、計算シミュレーションの精度を向上させようとするデータ同化（Data Assimilation）への期待も高い。さらに、わが国が国際社会と協調して積極的に取り組んでいる、持続可能な開発目標（SDGs：Sustainable Development Goals）から派生する社会課題解決の視点からも、ポスト「富岳」時代の次世代計算基盤は検討・考察されなければならない。加えて、わが国においては、サイバー空間（仮想空間）、フィジカル空間（現実空間)を高度に融合させることにより、経済発展と社会課題の解決を両立する人間中心の社会を目指すSociety 5.0がわが国の目指すべき未来社会として科学技術政策の一つとして掲げられており、ポスト「富岳」時代の次世代計算基盤を検討・考察する上で重要なキーワードともなる。さらに、今日の学術・産業界における研究活動は、ますますグローバル化、ボーダレス化する傾向にあり、地理的に分散して推進される研究開発に携わる研究者らのチームとしての研究活動の生産性向上という視点からも、ポスト「富岳」時代の次世代計算基盤のあり方は検討・考察されなければならないと考える。本グループ（システムソフトウェア・ライブラリ調査グループ）では、ポスト「富岳」時代の次世代計算基盤について、広範なSDGs・Society 5.0の実現に向けた課題解決のためのプラットフォームを実現すべく、高度なデジタルツインによる研究DX基盤となり得る次世代システムについて、サイエンス・エンジニアリングと産業・社会のニーズも考慮しながら、それを実現可能なシステム等の選択肢を提案することを目的とする。当該目的のために、本サブグループでは、次世代計算基盤のアーキテクチャ、システムソフトウェア・ライブラリ、アプリケーションの調査と要素技術の検討を行い、技術的課題や制約要因を抽出しつつ、次世代計算基盤に求められる性能・機能要件を明らかにする。また、我が国として独自に開発・維持するべき技術に関しても検討する。

3.2.10.2 調査内容

本サブグループでは、上記目的達成を鑑み、データサイエンス、AI技術、シミュレーションの融合、リアルタイムデータ処理、データ駆動型科学の計算ニーズ・要求を収容可能な高性能計算(HPC)システムのアーキテクチャについて調査を行うとともに、わが国の科学技術の発展・高度化に寄与できる次世代計算基盤の中核となりうる計算基盤技術、ネットワーク技術、データ基盤技術の利用環境について調査検討を行う。一昨年度の2022年度には、HPCI構成機関を対象として、ウェブ等において公開されている情報をもとにしたシステムソフトウェア・ライブラリ導入状況を調査した。こうした調査では、計算機システムに導入されているものの、実際は利用されていないシステムソフトウェア・ライブラリがあるといったシステムソフトウェア・ライブラリの正確な利用状況・実態を捉えることはできない。前年度の2023年度には、そのような背景から、HPCI構成機関を対象として、システムソフトウェア・ライブラリの正確な利用状況・実態について、ウェブによる公開情報、書面やアンケートによる調査ではなく、HPCI構成機関においてスーパーコンピューティングシステムの実際の運用に携わる管理者に対して直接対面にてヒアリングを行う形式にて調査を実施した。また、公開情報に記載されていないシステムソフトウェア・ライブラリを調査し、2022年度に整理し

たシステムソフトウェア・ライブラリ調査表を拡充した。最終年度となる2024年度は、2023年度から継続し、HPCI構成機関およびスーパーコンピュータを運用するわが国の国立研究所への直接対面によるヒアリング調査とともに、国外の米国、アジア、欧州から計算機センターをピックアップしシステムソフトウェア・ライブラリの導入状況についてウェブへの公開情報をもとに実施した。

3.2.10.2.1 調査方針

システムソフトウェア・ライブラリの利用状況・実態の調査手段としてはアンケートや質問状等の書面による聞き取りが考えられる。しかし、回答側の回答データの利用に関する不安や、質問の意図が明確に伝わらないことによる利用状況・実態の正確な回答が得られない可能性といったデメリットがある。こうした懸念と考察から、本サブグループでは、2023年度、および2024年度に、直接対面ヒアリングによるシステムソフトウェア・ライブラリの利用状況・実態を調査した。より具体的には、2022年度に取得した「アプリ利用者を対象としたアンケートによるシステム・ライブラリの利用状況・利用実態調査」結果を参考としながら、2022年度後半から2024年度中期にかけて検討・作成した口頭ヒアリング項目の調査を行った。これにより、ウェブによる公開情報、書面やアンケートによる調査ではなく、実際運用に携わる計算機管理者からの生の声を本利用状況調査に反映させることを狙う。また、同時に、前年度実施したアプリ利用者・開発者向けアンケート結果の分析をさらに掘り下げるとともに、利用者の現在の利用状況と利用希望の合致点・不一致点を明確にすることをねらう。

3.2.10.2.2 調査方法

3.2.10.2.2.1 調査対象機関

上記の調査方針に基づき、本サブワーキンググループでは、2023年度および2024年度に下記HPCI構成機関の計算機システム管理者に対する口頭ヒアリングを計画した。具体的な対象構成機関は、全HPCI構成機関のうち国立大学の機関に該当する下記の9機関、および理化学研究所計算科学研究センター、産業技術研究所、海洋研究開発機構とした。なお、口頭ヒアリングの時間は、ヒアリング対象となる計算機システム管理者や本サブワーキンググループのメンバの体力・集中力を想定して、1機関あたり2時間を目安とした。また、2024年度に実施した一部のヒアリング項目については、2023年度の調査をうけて追加分となったため、2023年度にヒアリングを実施した機関に対して書面調査として行っている。

- 北海道大学情報基盤センター
- 東北大学サイバーサイエンスセンター
- 筑波大学計算科学研究センター
- 東京大学情報基盤センター (*)
- 東京科学大学情報基盤センター（東京工業大学学術国際情報センター）
- 名古屋大学情報基盤センター
- 京都大学学術情報メディアセンター
- 大阪大学 D3 センター（サイバーメディアセンター）
- 九州大学情報基盤研究開発センター
- 理化学研究所計算科学研究センター (*)
- 産業技術研究所 (*)
- 海洋研究開発機構 (*)

3.2.10.2.3 口頭ヒアリング項目

HPCI構成機関の計算機システム管理者に対する対面口頭ヒアリングでは、本サブワーキンググループのメンバが

対象となるHPCI構成機関を訪問し、対面にて口頭ヒアリングを行うことを想定する。その際、3.2.10.2.1に記載したが、2022年度に取得し た「アプリ利用者を対象としたアンケートによるシステム・ライブラリの利用状況・利用実態調査」結果を参考としつつ、利用者のシステムに対する期待と管理者視点からの利用者理解の合致点や差異（ギャップ）を明確にすることを目指し、設計をおこなった。また、本サブワーキンググループのメンバの誰がヒアリングしても管理者の方々にできるかぎり同じ意図・ニュアンスが通じるようヒアリング文言についても検討を行った。

表 3.2.10.1 ヒアリング項目の構成概要

1. HPC センターについて	
(1)	貴センターの利用者の現状と傾向について
(2)	現システム整備状況や運用・整備方針等について
2. HPC センターの現状について	
(1)	導入ソフトウェア・ライブラリの利用状況や整備状況について
(2)	運用サービスについて
(3)	クラウド利用環境について
(4)	仮想計算機技術に対する必要性と利用状況について
(5)	データストレージ、データ基盤について
(6)	システムセキュリティ、モニタリングとユーザ利便性について
(7)	エコシステム、コミュニティ形成について
3. 新しいチャレンジについて	

口頭ヒアリング項目の構成としては、「1.HPCセンターについて」、「2.HPCセンターの現状について」、「3.新しいチャレンジについて」の3パートとした（表 3.2.10.1）。HPCI構成機関と一言で表現してもそれぞれの機関は異なる大学に設置され、またその利用者となる研究者層も異なれば、各HPCI構成機関が重点をおく研究分野や計算分野も多種多様に異なると考えられる。本サブワーキンググループのミッションであるHPC利用環境を調査するに際しては、単にどのようなソフトウェアがどのように利用されているかということだけではなく、その背景となるHPCI構成機関の状況もまた把握することが重要である。そのような視点から、本サブワーキンググループでは、最初のパート「1.HPCセンターについて」では、計算機システムの想定利用者（想定研究分野、想定計算ニーズ、想定利用形態)、計算機システムの整備状況、整備方針（運用状況、調達方針等）を理解することを目的として設定した。

次に、「2.HPCセンター の現状について」パートでは、2022年度に作成した「HPCI構成機関におけるシステムソフトウェア導入状況」について再確認を行うとともに、それらのソフトウェアがどの程度利用されているかを口頭にて確認する。その後、計算機システムにおけるニーズ、要求（仮想計算機技術、クラウド利用、データ基盤等）についての設問を設けた。これにより、ウェブからの情報、書面・アンケートでは得られない、正確かつ活きた、HPC利用環境の最新状況を把握するように務める。

最後に、「3.新しいチャレンジについて」パートでは、各構成機関において今後のスーパーコンピューティングシステム導入にむけた技術課題や解決しなければならないとか考えている問題についてヒアリングする。より具体的には、利

用者から寄せられる新しい計算ニーズや要望の充足に向け、どのようなシステムソフトウェアが今後必要となると考えられるか、またどのような技術課題をクリアしないといけないかをヒアリングする。

これらの３パートにより、各HPCI構成機関におけるHPC利用環境の相違・共通点、今後の課題を明確にすることを狙う。

3.2.10.3 口頭ヒアリング調査結果

2023年度は、3.2.10.2.2.1に記載したとおり、9機関のHPCI構成機関を対象として、対面ヒアリング調査を実施した。2024年度に (*) を記した組織に対してヒアリング調査を実施した。表 3.2.10.2にヒアリング実施日、ヒアリングにご協力いただいた計算機管理者、および、本サブワーキンググループでヒアリング対応を行ったメンバを記す。なお、表内で（主）と記しているのは、ヒアリングの主となる質問者を意味している。また、(注)とあるのは、ヒアリング担当と対象者の両方を兼ねていることを意味している。また、どのような雰囲気で対面ヒアリングが行われているかを示すために、その一例として、図 3.2.10.1に海洋研究開発機構の計算機管理者の方々への対面ヒアリング実施中の様子を付記しておく。

表 3.2.10.2 対面ヒアリング調査実施完了機関とヒアリング対象計算機関管理者（敬称略）

日時	HPCI 構成機関	ヒアリング対象 計算機管理者		ヒアリング担当
2023 年 8 月 29 日 14:00～16:00	大阪大学 D3 センター（サイバーメディアセンター）	速水 智教 寺前 勇希 上野 雅矢 向園 悠 伊達 進	特任助教 専門職員 技術職員 特任技術職員 教授 (注)	遠藤　新（主） 伊達　進 山下　晃弘 木戸　善之
2023 年 8 月 31 日 10:00～12:00	九州大学情報基盤センター	美添 一樹 南里 豪志 平島 智将	教授 准教授 主任 (技術職員)	遠藤　新（主） 伊達　進 山下　晃弘 木戸　善之
2023 年 9 月 26 日 10:00～12:00	東北大学サイバーサイエンスセンター	滝沢 寛之 下村 陽一 高橋 慧智 大泉 健治 小野 敏	教授 特任准教授 助教 技術専門員/共同研究支援係長 技術専門職員/共同利用支援係長	遠藤　新（主） 伊達　進 山下　晃弘 木戸　善之
2023 年 10 月 3 日 13:00～15:00	京都大学学術情報メディアセンター	深沢 圭一郎 疋田 淳一	准教授 掛長	遠藤　新（主） 伊達　進 山下　晃弘 木戸　善之

日時	HPCI 構成機関	ヒアリング対象 計算機管理者		ヒアリング担当
2023 年 10 月 17 日 15:00～17:00	筑波大学計算科学研究センター	建部 修見 小林 弘幸	教授 技術職員	木戸　善之(主) 伊達　進 山下　晃弘 遠藤　新
2023 年 11 月 21 日 10:00～12:00	東京科学大学情報基盤センター（東京工業大学学術国際情報センター）	遠藤 敏夫 野村 哲弘 渡邉 寿雄 安良岡 由規	教授 マネジメント准教授 マネジメント准教授 技術専門員	木戸　善之(主) 伊達　進 山下　晃弘 遠藤　新
2023 年 11 月 22 日 10:00～12:00	名古屋大学情報基盤センター	片桐 孝洋 星野 哲也 田島 嘉則 山田 一成 高橋 一郎	教授 准教授 主席技師 主任技師 技術補佐員	木戸　善之（主） 伊達　進 山下　晃弘 遠藤　新
2023 年 12 月 8 日 15:00～17:00	北海道大学情報基盤センター	深谷 猛 更科 高広 吉川 潤	准教授 チーム長 特定専門職	遠藤　新（主） 伊達　進 山下　晃弘 木戸　善之
2024 年 7 月 8 日 10:00～12:00	理化学研究所計算科学センター	佐藤 賢斗 庄司 文由 井口 裕次 村井 均 山本 啓二 岡本 光央	 部門長 副部門長 技師	遠藤　新（主） 伊達　進 山下　晃弘 速水　智教 曽我　隆
2024 年 7 月 16 日 10:30～12:00	産業技術総合研究所	滝澤 真一朗 高野 了成 大西 尚樹 小川 宏高 山本 智実 竹井 和也	チーム長 副連携研究室長 特定集中研究専門員 事業プロデューサ ベンダ SE 現場リーダ	遠藤　新（主） 山下　晃弘 速水　智教 曽我　隆
2024 年 7 月 16 日 16:00～18:00	東京大学情報基盤センター	塙　敏博 佐藤　孝明 中張　遼太郎 昆野　長典	教授 係長 主任	遠藤　新（主） 伊達　進 山下　晃弘 速水　智教

日時	HPCI 構成機関	ヒアリング対象 計算機管理者	ヒアリング担当
		須貝　佳義	曽我　隆
2024 年 7 月 17 日 13:00～16:00	海洋研究開発機構	上原　均　教授 矢崎　琢也　事務副主幹 佐々木　朋樹 斎藤　友一　グループリーダー代理 小笠原　輝美　技術スタッフ 甲斐　恭　技術副主任 石黒　駿 大倉　悟　グループリーダー代理	遠藤　新（主） 伊達　進 山下　晃弘 速水　智教 曽我　隆

図 3.2.10.1　海洋開発研究機構での対面ヒアリングの様子

以下、本調査結果の概要を述べた後、各ヒアリング項目に対する回答の詳細をまとめる。

3.2.10.3.1 調査結果の概要

本報告書でまとめる内容は、各HPCI構成機関に対し個別に2時間にわたる対面でのヒアリング調査の文脈でえられたものである。それゆえ、一部を切り出し、どの結果が重要であるというのは本サブワーキンググループの所感に基づく側面が大きくなり、また誤解を生じさせうる危険がある。しかしながら、あえて次期スーパーコンピューティングシステムを考える上で、特に重要と思われる調査結果項目を含むもの（6点：2024年度追加のヒアリング項目を含む）を、本サブワーキンググループの所感に基づきながら、ここでは概要としてまとめる。

(1) サービス利用者の実態を把握するソフトウェア、アプリケーションについて

本調査では、「サービス対象利用者の実態を把握・管理するためにアプリケーション、ソフトウェア等を導入されますか？」「サービス対象利用者の実態を把握・管理するためのアプリケーション、ソフトウェアで「このような」アプリケーションがあれば良いと思うアイデア等があれば教えてください」と各HPCI構成機関に対して質問を行なっている。本項目は、本ワーキンググループとして、次期スーパーコンピューティングシステムを検討していく上で、各HPCI構成機関の利用実態を理解する上で重要な質問項目となる。そのために、各HPCI構成機関がどのように利用者のサービス利用実態を把握しているかの情報を取得しているかを質問した。しかし、以下の報告に示す通り、利用者のジョブ実行時間、ジョブの並列度、ジョブ起動回数などジョブスケジューラから取得される情報に留まっている現状がある。同時に、CPUやGPUといった計算資源が実際どれくらい利用されているのか？といったデータを収集したいと考えているものの、データ収集・解析ができていない現状もある。このような現状と実態については、次期・次々期のスーパーコンピューティングシステムを検討していく上で、サービス利用者の実態を把握するソフトウェア、アプリケーションの必要性については検討の価値があると思われる。

(2) 運用体制・人員、自動化省力化

本調査では、各HPCI構成機関の運用体制・人員について調査を行なっている。より具体的には、運用体制について、「システム運用に携わる人員や役割分担について教えてください」、「運用支援をベンダ等に依頼しておられますか？」、「運用の自動化、省力化等について取り組まれていることや、導入しているシステムソフトウェアはありますか？」と質問を行なっている。この調査では、一言で「運用」と表現した場合でも、各構成機関での「運用」が意味することが異なる可能性があり、またそれにより体制・人員が異なりうる。次期スーパーコンピューティングシステムにおいても、大規模なスーパーコンピューティングシステムが効率的に利用者に利用されるには当該システムの効率的な運用は必要不可欠である。さらに、大規模なシステムでは運用負荷を軽減するための自動化や省力化もまた重要である。そうした観点から、各構成機関における運用体制の現状を調査した。この結果、本年度調査したHPCI構成機関においては、少なくとも7名以上の運用チームによってスーパーコンピューティングシステムを運用していること、また、常駐・非常駐の差はあるが全ての構成機関でベンダに依頼していることもわかる。すなわち、HPCI構成機関のスーパーコンピューティングシステムの運用は大学とベンダの混成チームで行われていることがおおいのが現状である。自動化・省力化に関しては、各HPCI構成機関で多岐に渡って取り組まれている現状があり、スケジューラの機能拡張の必要性が高い状況が見えてきている。

(3) スケジューラについて

本調査では、各HPCI構成機関に対して「利用者からの計算要求を処理する仕組みとして何を利用されてますか？」の問いに、全てのHPCI構成機関がスケジューラと回答している。また、スケジューラを採用している理由としては、システム調達の結果ベンダからの提案に基づくスケジューラが導入されている状況がある。言い換えれば、どのHPCI構成機関としても調達時においては、特定のスケジューラに限定しているわけではないが、調達結果としてベンダの提案スケジューラが採用されている。また、スケジューラにおけるジョブ最長時間の設定については、アプリ開発者の視点からは長時間の設定が望まれるが、本年度調査したHPCI構成機関からは、24時間から720時間と幅広い回答が得られた。また、24時間以上のジョブはチェックポイントをとってもらって対応するのがよい、という意見もあった。管理者視点からは長時間ジョブを許可している場合にメンテナンスや障害時の対応がとりにくくなるということであった。この点でアプリ開発者の視点と管理者の視点にギャップがある点が浮き彫りになっている。

(4) クラウド利用について

本調査では、クラウド利用環境について「現在、何らかの用途で民間クラウドベンダの提供するサービスを導入されていますか？」と質問を行なっている。この結果、現在は提供を終了しているものも含めて7つのHPCI構成機関においてクラウド利用環境の導入実績があった。さらに、これら7つのHPCI構成機関のうち4つのHPCI構成機関におけるクラウド環境の利用用途としては共通して、オンプレミスの計算需要を民間クラウドベンダの提供する計算環境にオフロードするクラウドバースティングを意図していた。しかし、導入実績のあるなし関係なく、ほとんどのHPCI構成機関において、クラウド計算資源の利用料金の高さがクラウドバースティングの導入における課題であるという回答も得ている。基本的に多くのHPCI構成機関では計算によって消費される電気代を基準に利用料金を定めているため、オンプレミスの計算資源の利用料金はクラウドの計算資源と比べると非常に安価となる。この利用料金の差が各HPCI構成機関の運用において扱いづらいことが、クラウドバースティングを導入するにあたっての障害となっていることも明らかになった。

(5) GPU への対応状況、エコシステム、コミュニティ形成（2024 年度追加のヒアリング項目）

2023年度のヒアリング調査の結果、および他グループの調査結果を踏まえて、2024年度は下記 3 項目についてのヒアリングを行っている。これらの項目については、2023年度にヒアリング調査を行ったHPCI構成機関も対象とした。そうした機関に対しては、書面調査にてヒアリングを実施し、回答が得られたもののみをまとめている。

(a) GPUへの対応状況について（回答は、表 3.2.10.8、表 3.2.10.9にまとめた。）

(b) エコシステムについて（回答は、表 3.2.10.18にまとめた。）

(c) コミュニティ形成について（回答は、表 3.2.10.19にまとめた。）

(a) の調査では、各HPCI構成機関の整備方針に関連して、NVIDIA以外のGPU製品に対する関心についてのヒアリングを行っている。その結果、NVIDIA以外のGPU製品に対する利用者からの関心は、1つのHPCI構成機関にて一部の利用者からのAMDのGPU製品に関心がある旨を聞いたとの回答以外、特にないとの回答であった。また、管理者からの視点としては、NVIDIA以外のGPU製品は、GPUをシステムに導入するうえで候補としてあがるが、利用者にとっての使いやすさという面で懸念があるとの回答が多かった。実際に、HPCI構成機関における導入システムの事前評価に関してヒアリングしたところ、AMDのGPU製品も含めた事前評価は実施されているものの、指定するソフトウェアが利用可能であるかどうかという評価項目が重要視される傾向にあった。このように、NVIDIA以外のGPU製品に対する事前評価が実施されているものの、NVIDIAに有利な評価項目が重要視されていることから、NVIDIA以外のGPU製品の使いやすさに関する懸念が払拭されない限りは、実際に広く導入されることはまだ少ないだろうと見込まれる。

(b) の調査では、各HPCI構成機関にて創出した成果の社会還元をサポートする取り組みや、導入システムにおいてそうした成果を活用する取り組み（標準技術の採用や要望のあるアプリケーションの導入等）といったエコシステムについてのヒアリングを行っている。その結果、アプリケーションエコシステムに関しては、各HPCI構成機関にて創出した成果を社会還元する取り組みを実施するHPCI構成機関は2つと少ない数であった。一方で、要望のあったアプリケーションの導入や講習会の開催といった取り組みは多くのHPCI構成機関において実施されていた。イノベーションエコシステムに関しては、共通して取り組まれていることはないものの、6つのHPCI構成機関において取り組んでいるとの回答があった。具体的には、富岳技術のデファクトスタンダード化や企業利用に対する計算資源の活用支援といったものであった。また、エコシステムの必要性については、約半数のHPCI機関において必要であるとの回答が得られた。ただし、必要であると回答した内訳としては、エコシステムを取り込んで利用者にとっての利便性を高める方向のものが多かった。以上の通り、エコシステムの重要性は認識されつつあるものの、現状では創出した成果を積極的に社会還元しようとする取り組みはまだ部分的であると思われる。

(c) の調査では、研究や企業利用のコミュニティを形成する取り組みについてヒアリングを行っている。その結

果、研究コミュニティに関して、学際領域における直接的なコミュニティ形成を実施しているHPCI構成機関は2つ、講習会や利用者交流会を開催しているHPCI構成機関は4つであった。産業利用コミュニティに関しては、シンポジウムの開催やSociety 5.0推進拠点の設置といった直接的なコミュニティ形成に取り組むと回答したHPCI構成機関は2つ、広報活動や計算資源のトライアルユースといった企業利用の推進に取り組むHPCI構成機関は5つであった。このように、各HPCI構成機関におけるコミュニティ形成の取り組みはまだ小規模であると思われる。これは、汎用目的のシステムを導入することが多い各HPCI構成機関では、東京科学大学から頂いたコメント（表 3.2.10.18）にあるように、特定分野の十分な人数を集めることが困難であることが一因であると考えられる。

(6) 新しいチャレンジ

本調査では、各HPCI構成機関の次のスーパーコンピューティングシステム導入にむけたチャレンジ項目についてのヒアリングを行なっている。その結果、インタラクティブ利用については現状のシステムでインタラクティブバッチなどの仕組みで利用可能になっているものの課題意識がある。また、リアルタイム処理などへの要求もまた課題としてあげられている。ストレージサービスの高度化・強化もまたチャレンジとしてあげられる結果となっている。詳細は以下の報告を参照されたい。

3.2.10.3.2 HPC センターについて：センターの利用者の現状と傾向

3.2.10.3.2.1 利用者の研究分野

本ヒアリング項目では、図 3.2.10.2に示すスライドを用いてヒアリングを行った。本ヒアリング項目では、(1) 情報学、(2) 化学、(3) 工学、(4) 数物系科学、(5) 複合領域、(6) 医歯薬学、(7) 総合理工、(8) その他と分類し、HPCI構成機関における利用者層の研究分野についてヒアリングした。その結果、正確な数字を有しているHPCI構成機関と正確な数字を有していないHPCI構成機関があることが判明した。また、利用者の研究分野データを有していても、データの分類解析ができておらず本ヒアリング項目への正確な回答が難しい現状があることも判明した。それゆえ、表 3.2.10.3に示す回答には、ヒアリングの際に得られた管理者からの体感・所感的な回答もまた、HPC利用環境の実態を把握・理解する際の重要なデータとなると考え、記載している。

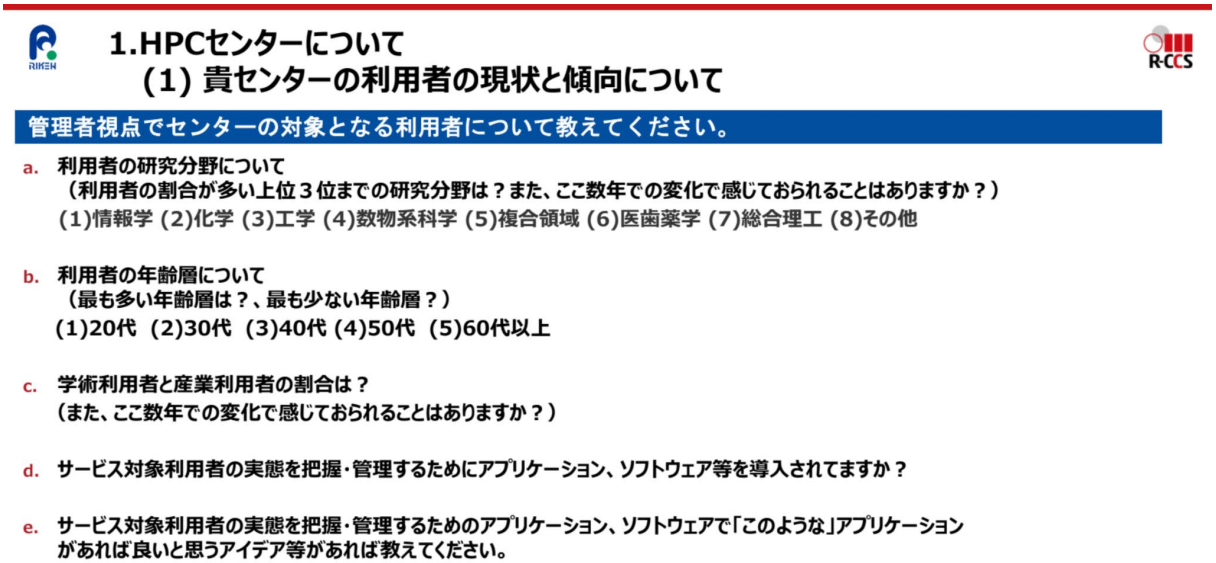


図 3.2.10.2 センターの利用者の現状と傾向についてのヒアリング項目

表 3.2.10.3 構成機関の利用者プロファイル

構成機関	分野	年齢層	学術利用と産業利用の割合
大阪大学 D3 センター	数物系科学 36.8% 工学 28.9% 医歯薬学 13.2% 化学 5.3% 取得方法：利用申請時の分野情報から集計	25-34 歳 28.95% 35-44 歳 26.32% 45-54 歳 21.05% 55-64 歳 15.79% 65 歳以上 7.89% 24 歳以下 2.63% 取得方法: 利用申請時の登録ユーザ所属情報から集計。	平均 10% 産業利用 コメント：資源量の 15%を産業利用に割り当てることができる。近年産業利用が増加傾向。
九州大学情報基盤センター	理工系 (化学) 生物系 取得方法: 利用申請時の分野情報に基づく。 コメント： 化学で Gaussian/VASP が使われている印象。利用傾向における新たな分野の出現は体感的にはみられない。	大学院生（修士）21% 准教授 15% 教授 13% 取得方法： 利用申請時の登録ユーザ職位情報から集計。 コメント： 年齢データは取得してない。	1-2% (申請ベース) 産業利用 コメント： 全体の利用者は増えているが、民間からの利用はそれほどでもない。P100 世代で古いというのもあるかもしれない。

東北大学サイバーサイエンスセンター	流体 気象・地球物理 取得方法：利用時間に基づく。 コメント：近年での理代分野傾向に変化はない。	助教 准教授 コメント：上記は、管理者の日常運用管理系経験に基づく体感的な回答。	ほぼ学術だが (産業利用は数%だけ)、共同研究の企業利用はある コメント： 利用時間ベース(約 3%) 利用申請責任者ベース（企業契約は少ないが、学術共同研究の企業利用はある）
京都大学学術情報メディアセンター	工学系　約 40% 数物系科学　32% 化学 12% 情報学 コメント：近年、利用分野割合順序に変化はない。化学と生物系は若干増加傾向。	大学院生　6 割 学術研究者　4 割 コメント：年齢データは取得していない。時間があれば利用者職種別など集計は可能。利用者層分析はあまり重視してない。	体裁としては全て学術利用だが、共同研究での企業利用はある。 コメント：利用申請の代表者が企業の方の申請はない。大学に何らかの身分がある方が申請している。
筑波大学計算科学研究センター	マテリアルサイエンス ライフサイエンス 素粒子 コメント：ライフサイエンスと素粒子は同数で 2 位。	36 歳以下（227 人） 40 歳以上（159 人） コメント：ユーザから申告いただく情報は、36 歳以下、40 歳位以上のみしか集計していない（37-40 が抜けている）。教育利用もあり若年層が多い。	産業利用はほとんどない。
東京科学大学情報基盤センター	工学院(工学)　18% 物質理工学　10% 理学院　8% 情報理工学院　7% (情報学)	大学院生 (半々か学生が多いくらい) コメント：統計は取ってない。上記の回答は	産業利用は非常に減少している。 コメント：産業利用は上位互換の ABCI に流れていると思われる。現状産業利用はほぼ成果非公開の

	取得方法：アカウント数からの集計 コメント：利用金額ベースとなると上記と異なる結果となる	肌感覚。	み。
名古屋大学情報基盤センター	工学系 材料系（化学） 情報学 コメント：2020 年以降、情報学の利用が増加傾向にある。	(2) 30 代 学生も多いが、学生よりも研究者による利用が多い。 コメント：統計は取っていない。上記の回答は肌感覚。	3～5%：産業利用 コメント：資源量の 5%を産業利用に割り当てている。共同研究の形で学術利用の中にも産業利用があるが、それでも学術利用が多い。
北海道大学情報基盤センター	計算化学 気象 工学 コメント：上記は、負担金ベースにおける肌感覚の回答。	学生　200 人 助教　80-90 人 講師　170 人 准教授　150 人 教授　160 人 コメント：年齢データは取得していない。上記の回答はおおよそのアカウント数。	10～15%：産業利用 コメント：資源量の 10～15%ほどを産業利用に割り当てているおり、その全てが使われている。基本的に、学術利用優先のポリシー。
理化学研究所計算科学センター	バイオ・ライフ 物質・材料・化学 環境・防災・減災 コメント： 富岳の利用者選定業務は登録機関（RIST)が担当している。採択課題数：重点分野 18 課題。	把握していない コメント： 富岳の利用者選定業務は登録機関（RIST)が担当している。	産業課題：14 件 一般課題：45 件 若手課題：　9 件 コメント： 全体 68 件の中で企業出身者が一人以上いる課題はざっくり 6 割程度である。

産業技術総合研究所	情報学（人工知能） コメント： 人工知能に関するほぼ全ての領域。 （アルゴリズムや並列、応用分野のバイオサイエンスなど。最近では量子関係でも利用。）	把握していない	学術：産業=4:6 コメント： 企業利用が 50%の割合で伸びている。 ・2019 年度 学術：産業＝ 6: 4 ・2023 年度 学術：産業＝ 4:6 と逆転 (LLM などの大規模利用で学術も伸びている)
東京大学情報基盤センター	地球科学・宇宙科学 エネルギー物理学 材料科学 工学 コメント： ・GPU 搭載システムでは、情報科学（特に AI)、バイオインフォティックス、生物化学、材料科学などが多い。 ・部局単位等の大口ユーザ契約が占めている部分が多く、学内利用者：外部（学外）利用者の割合は半々程度。	院生 30% 学部生 15% コメント： ・データ収集していないが、職域で勘案すると学生（学部・院生）で 50%程度（概ね 20 代）。 ただし、教育利用で 200 人程度毎年登録（全体の 3 割程度）	企業利用者数は 10%程度(共同研究、企業の単独利用者の合計) 企業申請制度の企業利用者数は、1% コメント： アカウントベースで 10%程度が企業利用者

海洋研究開発機構	地球科学、海洋地球科学 ・圧倒的に地球科学が多い。（大気、海洋、地震、環境など） ・工学、総合理工系もある。	感覚的に 30 代～50 代 ・データ収集していないため感覚的。 （20 代、60 代は少ない）	学術利用が圧倒的 ・ほぼ学術利用者で産業利用は 1 割以下

表 3.2.10.3から、多くのHPCI構成機関では、利用者の研究分野は工学、数物系科学、理学といった、いわゆる理科系であることがわかる。一方、いわゆる文化系での利用はほとんど見られないこともわかる。また、意外ではあるが、情報学分野の利用者が上位3位に入る構成機関は2つだけであり、少数派であることもわかる。

3.2.10.3.2.2　利用者の年齢層

また、利用者の年齢層に関する調査結果を、表 3.2.10.3の第3列に示した。当該項目については、利用申請時のデータから統計をとっているHPCI構成機関もあれば、年齢層については取得していないHPCI構成機関もあるのが実情である。明確な数値データがない場合は空白とするのではなく、対面ヒアリングによって得られた肌感覚的な回答を表にまとめている。これらの結果から、どの構成機関においても、大学院生を含む若い研究者層がスーパーコンピューティングシステムを多く利用していると見込まれる。

これは本サブワーキンググループの推測ではあるが、今回の調査対象となるHPCI構成機関はすべて国立大学法人に設置されており、各大学における研究・教育(人材育成)に資することが期待されている。それゆえ、大学院生などの若い研究者の利用が多いとも考えられる。また、スーパーコンピューティングシステムを利活用する研究分野では、理論構築、プログラム作成・実行、性能チューニング、データ解析、可視化などの分野のエキスパートが研究チームを形成の上利用することも多く、若い研究者がスーパーコンピューティングシステムを実際に利用していることも考えられる。この点は、国内研究機関に設置されたスーパーコンピューティングシステムでは状況が違うと想像される。

3.2.10.3.2.3　学術利用と産業利用の割合

次に、各構成機関における学術利用と産業利用の割合の調査結果を表 3.2.10.3の第4列に示す。当該項目のねらいは、HPC利用環境を調査するに際して、これらの利用者層の違いがシステムの整備計画に差異を生み出すかどうかを検証することをねらった。

表 3.2.10.3から、どの構成機関においても産業利用の制度が導入されていることがわかる。しかし、産業利用の割合は、程度の差はあるが基本的に学術利用の割合より非常に小さいこともわかる。これは、今回の調査対象としたHPCI構成機関が全て国立大学法人に設置されていることに起因している。すなわち、産学連携を意識しつつも大学内の利用者を優先した結果と考えられる。このように学術と産業での利用者層の傾向がほぼ同じことから、利用者層の違いによるシステムの整備計画の差異は、今回の調査対象としたHPCI構成機関の間では生じないと考える。ただし、ポスト「富岳」時代の次世代計算基盤においても同じ利用者層の傾向になるとは限らないため、本調査を参考とする際はその点について留意する必要があるとも考える。

3.2.10.3.2.4　利用者の実態を把握・管理する方法

利用者の実態を把握・管理する方法の調査結果を表 3.2.10.4に示す。当該項目では、以下の3項目を把握することをねらい、「利用者の実態」をあえて明確に定義せずにヒアリング調査を実施した。

- 何を利用者の実態として考えているか？
- 利用者の実態を把握・管理するために利用しているアプリケーション、ソフトウェア
- 利用者の実態を把握・管理するために、あれば良いと考えるアプリケーション、ソフトウェア

表 3.2.10.4　利用者の実態を把握・管理する方法についてのヒアリング結果

構成機関	サービス利用者の実態を把握・管理するアプリケーション、ソフトウェア導入状況	サービス利用者の実態を把握・管理するために「あればいいな」と思うアプリケーション、ソフトウェア
大阪大学 D3 センター	スケジューラ 統計情報 コメント：利用者のジョブ統計を採取して並列数などの分析に利用する場合がある。XALT を導入して GPU の利用携帯を調査しようとしたが、モニタリングする場合の性能への影響が見極められていないため、進んでいない。	オープンソースの利用件数を取れるソフトウェアシステムの負荷計測と分析 コメント：RIST の mpirun を改修して、実行回数を取れるものがあるが、なかなか利用者に普及してない。利用者に普及させて使ってもらうのも課題。オープンソースのソフトウェアを使う人が多いが実行件数が取れてない。また、次のシステム調達にむけて、システム負荷の計測と分析できるソフトウェアがほしい。
九州大学情報基盤センター	ジョブ統計情報 コメント：利用者のジョブ統計を採取（ソフトウェアの実行数）	ライセンス費がかかるアプリの利用実績採取 アプリのログ切り出し機能利用者のジョブ統計を集計するソフト コメント：利用者のジョブ統計を取っているが、ソフトウェアの実行数などの集計をできる管理ソフトがあるとうれしい。商用ソフトウェアの見直しのためにライセンス費がかかるアプリの利用実績は採取したい。

東北大学サイバーサイエンスセンター	統計情報 自作ツール コメント：システム統計情報等から利用者毎の実行時間等やアプリ利用統計情報を蓄積・分析している。センター導入のアプリ利用統計から不要な ISV ライセンス費を適切化した。	・各種情報の DB 化ツール （自大学の必要な分析項目は自身でカスタマイズ）
京都大学学術情報メディアセンター	module Grafana 有償ソフトウェアのライセンス コメント：・有償ライセンスの適正化と導入 OSS の利用度合いから OSS の選別のためにライセンスの利用実態は適宜集計している。OSS は module のロード回数を収集している（利用度合いの確認程度）。既存および自製ツール。ノード稼働状況、ジョブ投入状況を Grafana で可視化。	困っている利用者を抽出したりするために、ユーザがどれくらいのジョブを投げたりとか、どういうエラーをどのくらい出しているとかを集計できるソフトウェア。 また、そういったジョブの規模感に応じて、次のシステム導入の検討材料としたい。
筑波大学計算科学研究センター	特になし。 コメント：基本、アプリは利用者が準備するスタンスでアプリの利用動向は不明。	特になし。 コメント：利用動向については HPCI リストにある理研殿提供アプリは（ラッパー設定があるので）採取されているが、センターでは分析していない。
東京科学大学情報基盤センター	特になし。 コメント： ・利用者動向の把握のために特別に何かを入れているというよりは既存の仕組みを使ってデータをついでに取っている。 ・提供しているパッケージについてはカウントしているが分類が難しい。	特に強く検討していない。 コメント：・ISV アプリに対してアプリ課金の方を実施方針。 ・ユーザに対する影響を最小限にするという観点から、監視系ソフトウェア導入を強くは考えていない。

名古屋大学情報基盤センター	module コメント： 前システムではジョブスクリプトから解析していたが、現在ではモジュールロードのログのみで利用傾向の把握は正確ではない。	特になし。 現ツールでは利用資源の有無（GPU がどれだけ利用されている）までで、シングル、MPI ジョブなのかのジョブ特製の把握まではできていない。
北海道大学情報基盤センター	特になし。 ・（アカウント等の）データベースのデータを見るぐらい。 ・学内はデータが連携しており、オンラインで、資源追加やトークン申請することも含めてシステム化をされている	特になし。 ・アプリの使用実績等は把握できれば便利だと思うが現状はできていない。
理化学研究所計算科学センター	・富士通製ミドルウェアの機能（ジョブスクリプトの保存など）および自作ソフトウェア。 ・アカウンティング機能はある。 ・外部利用ライブラリ、利用コンパイラや利用オプション等（利用形態・状況、改善点把握のため）これらを SQLDB の保存（分析可能）	実行中アプリの挙動と電力や性能などとの関係を把握したい(自作中）。汎用アプリ・ソフトでは実現不可と思われる。
産業技術総合研究所	Prometheus Zabbix ・ジョブスケジューラレベルで収集可能な情報を収集。（ジョブログを分析） ・外部から取得可能なリソース使用状況を収集。	・現在は、収集データを後日マニュアル集計しているが、リアルタイム集計できればと考えている。
東京大学情報基盤センター	質問受付管理システム ・質問受付管理システム（自製：Web とメールを連動）	・インハウスコードの利用者が主のため動向把握は難しい。必要が出てきた段階でモジュールファイルの範囲内で対応してもらうレベル。

	・サービスで提供している ISV については利用動向を収集している。主にライセンスの使用が適切に利用させているかの判断等をしている。	
海洋研究開発機構	・利用状況ということでは、リクエストの統計情報を採取（統計情報の採取切り口は仕様を提示して開発させた）し、分析している。	利用状況のアラート機能 採取統計情報の分析機能 ・短期間に多量のジョブ投入など突発的な利用状況の変化に対して運用制限を緩和するようなしくみが必要。または、変化の発生をアラートしてくれるような通知機能であっても良い。

表 3.2.10.4の第2列目には、ヒアリング実施時における、各HPCI構成機関における「サービス対象利用者の実態を把握・管理するための導入済みアプリケーション、ソフトウェア」についてのヒアリング結果を示している。これらの結果から、各センターでは①ユーザニーズの把握やコスト最適化へのフィードバック（利用アプリケーションの傾向やISV等のコスト最適化）②運用負荷に対応する安定稼働へのフィードバック③利用者支援へのフィードバックの 3 点が現状の主な実態把握の目的と見られる。具体的にはジョブの統計情報、ジョブスケジューラやmoduleといったソフトウェアやツールからのログ情報によって、多くの構成機関がサービス対象利用者の実態を把握・管理しようと勤めていることがわかる。同時に、HPCI構成機関でのサービス対象利用者の実態把握については、利用者のジョブ実行時間、ジョブの並列度、ジョブ起動回数に焦点を留めており、また、それも十分にデータを収集できてない現状があることがわかる。すなわち、システムより生成されるログ情報をもとに、管理者らが実態を把握するために多大な時間と労力を使い、利用者のジョブ実行時間、ジョブの並列度、ジョブ起動回数といったデータを収集するに留まっているのが現状である。加えて、CPUやGPUといった計算資源が(要求された計算資源に対して)実際どれくらい利用されているのか？といった部分のデータを収集したいと考えているものの、実際はデータの収集・解析ができてない状況も明らかとなった。しかし、東北大学サイバーサイエンスセンターにおいては、自作ツールによりライセンスを必要とする商用ライセンスだけではなく、任意のオープンソースソフトウェアの起動回数を把握できるソフトウェアを自作・利用していることは特筆すべき点である。

表 3.2.10.4の第3列目には、「サービス対象利用者の実態を把握・管理するための導入済みアプリケーション、ソフトウェア」についての現状に基づき、各HPCI構成機関に対して実施した「サービス利用者の実態を把握・管理するために「あればいいな」と思うアプリケーション、ソフトウェア」についての調査結果をまとめた。この結果からは、現状ではログ情報に基づいたサービス対象利用者の実態把握・管理ができてない現状から、より簡便かつ効率的にどのような頻度でソフトウェアやアプリケーションが稼働しているのか、の情報を把握できるアプリケーション、ソフトウェアを求める声が多くのHPCI構成機関が得られた。また、取得されたログ情報を利用者ごとに分類する、収集されたログ情報を実態把握・管理のためにデータベース化するといった、ログ情報の解析や分類のために利用可能なツールを求める声もあった。さらに、スーパーコンピューティングシステムがどのような負荷で実行されているかを計測し、システムの安定運用に役立てることができるソフトウェアを求める声もあった。また、東京科学大学情報基盤センターや大阪大学D3センターのコメントとして記載があるが、サービス対象利用者の実態を把握・管理するためのソフトウェアの導入はスーパーコンピューティングシステムとして提供する計算性能に影響を与えうる。そのため、HPCI構成機関の所有す

るスーパーコンピューティングシステムに対して、このようなソフトウェアやアプリケーションを積極的に導入できないという事情もある。一方、本年度ヒアリング調査は実現できてないが、富岳ではSpackレシピの利用状況、共有ライブラリのロード状況、コンパイラオプションの利用状況などの情報を取得し、リアルタイムでの利用情報を公開している。これは本サブワーキンググループとしてのコメントであるが、性能に対して低侵襲な、サービス対象利用者の実態を把握・管理するためのアプリケーション、ソフトウェアの開発は、現状の利用者からの利用状況を正確に把握するとともに、システム運用中にみられる利用ニーズ・要望の変遷を把握し富岳での経験を下にし、かつその実績を発展させ、次期、次々期といった将来のスーパーコンピューティングシステムの設計・開発に備えるためにも、重点をおく必要があるのではないかと考える。

3.2.10.3.3 HPC センターについて：計算機システムの整備状況・整備方針

本ヒアリング項目では、図 3.2.10.3、図 3.2.10.4、図 3.2.10.5に示すスライドを用いてヒアリングを行った。このヒアリング項目の設定は、各HPCI構成機関での運用対象となるシステムの詳細、体制を明確にし、本ヒアリング調査における他項目に対するヒアリング結果を理解する際の一助とすることをねらっている。

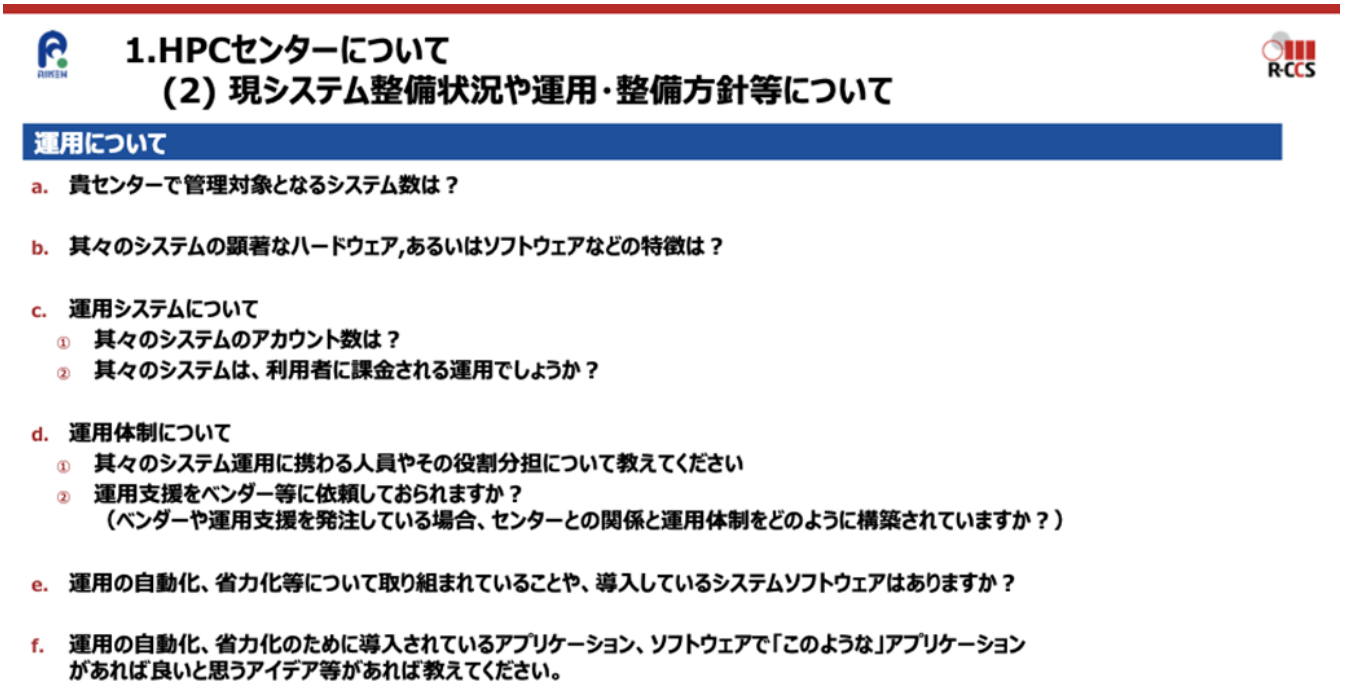


図 3.2.10.3 現システム整備状況や運用・整備方針についてのヒアリング項目 1

図 3.2.10.3に示すヒアリング項目では、まず各HPCI構成機関において所有するスーパーコンピューティングシステムおよび運用体制・状況について整理する。具体的には、ヒアリング対象となるHPCI構成機関の管理者らが考えるスーパーコンピューティングシステムのハードウェアおよびソフトウェア的な特徴を整理する。その上で、当該スーパーコンピューティングシステムで運用されているアカウント数および課金による運用であるかという視点で運用について調査する。一方、当該HPCI構成機関における運用体制、とりわけ、運用に携わる人員・役割、運用支援をベンダに依頼しているかどうかを整理する。その上で、運用負荷を軽減するための自動化や省力化に向けて取り組んでいることをヒアリングするとともに、今後運用の自動化・省力化にむけてこのようなアプリケーションがあればよいと考えられるアイデアについて質問する。

1.HPCセンターについて
(2) 現システム整備状況や運用・整備方針等について

システム整備方針について(1)

a. **システムの整備方針について**
   - ① **最も優先している方針は？**
     (1)利用者拡大 (2)現利用者の重視 (3)重点分野の有無 (4)電力 (5)収益性 (6)他センターに対する競争力
     (7)システム規模 (8)その他
   - ② **システム整備を行う上で「何」を問題・課題と考えておられますか？**
   - ③ **其々のシステムの整備方針に違いがありますか？**

b. **同一システム内（クラスタ内）で異なるノード構成が組み合わされたハイブリッドクラスタシステムの必要性はありますか？**

c. **オープンベースのシステムソフトウェアを用いたスパコンシステムの整備を検討する場合、管理者視点として「何」がメリット、デメリットと考えますか？**

d. **商用ベースのシステムソフトウェアを用いたスパコンシステムの整備を検討する場合、管理者視点として「何」がメリット、デメリットと考えますか？**

図 3.2.10.4 現システム整備状況や運用・整備方針についてのヒアリング項目 2

次に、図 3.2.10.4に示すヒアリング項目では、当該HPCI構成機関がシステム整備を推進する上での優先方針について調査する。その上で、その優先方針に基づいた整備計画において、現状どのようなことが問題・課題となっているのかをヒアリングする。複数システムを所有する場合は、それぞれの整備方針・計画について伺う。また、今日では計算要求・ニーズの多様化を背景とすることから、同一のアーキテクチャではなく、CPUノードやGPUノードといった異なるアーキテクチャを組み合わせたハイブリッドシステムとしてスーパーコンピューティングシステムを設計・構築することがある。そういった視点から、当該HPCI構成機関でのハイブリッドクラスタシステムに対する必要性を調査する。さらに、オープンソースベースあるいは商用ベースのシステムソフトウェア活用による次期富岳システムの可能性を鑑み、システム管理者視点でのメリット、デメリットを抽出することをねらう。

1.HPCセンターについて
(2) 現システム整備状況や運用・整備方針等について

システム整備方針について(2)

a. **システム構成について**
   HPCI参画機関が整備しているシステムで、Xeon＋GPU,EPYC+GPUタイプのCPU構成システムが多くなってきています。
   昨今ではN社以外にもI社、A社のGPU製品が出てきており、TOP500の１、２、３にランクインしている状況です。
   - ① **I社、A社のGPU製品について、ユーザサポート、利用者交流機会等での利用者の関心の度合いを教えてください。**
   - ② **A社、I社ののGPUについて、センターとしてどのように考えるかについて教えてください。**

b. **導入システム決定時または、将来導入システム整備に向けて事前評価を実施した経験はありますか？**
   （Yesの場合）
   - ① **評価構成について教えてください。**
     （CPU：x８６系（I社、A社）,独自CPU　CPU＋GPU組み合わせ:x８６＋GPU（N社、I社、A社）　など）
   - ② **評価システムのソフトウェアスタックについて重要視する点（した点）を教えてください。**

図 3.2.10.5 現システム整備状況や運用・整備方針についてのヒアリング項目 3

最後に、図 3.2.10.5に示すヒアリング項目では、今日のアクセラレータGPUのほとんどがNVIDIA社である状況を鑑み、他社（具体的には、AMDおよびIntel等）のGPU製品を選択する可能性といった、各HPCI構成機関の

整備方針を調査する。具体的には、「HPCI参画機関が整備しているシステムで、Xeon＋GPU、EPYC+GPUタイプのCPU構成システムが多くなってきています。昨今ではN社以外にもI社、A社のGPU製品が出てきており、TOP500の１、２、３にランクインしている状況です。」を前提条件とし、他社のGPU製品について、ユーザサポート、利用者交流会等での利用者の関心度合い（a-1）、それらの他製品に対する計算機センターとしての考え（a-2）、加えて将来導入するシステムの整備時に向けて、それらの他社製品に対する事前評価の経験があるかどうか（b-1）、ある場合においては、どのような知見が得られたか（b-2、b-3）について調査する。

これらの調査結果を表 3.2.10.5、表 3.2.10.6、表 3.2.10.7、表 3.2.10.8および表 3.2.10.9にまとめる。

3.2.10.3.3.1 管理対象とするシステム数、システムごとの顕著な特徴（ハードウェア、ソフトウェアの観点から）および、アカウント数

表 3.2.10.5第2列に対象となるシステムおよび当該システムのHW/SW特徴、表 3.2.10.5第3列に各HPCI構成機関で発行しているアカウント数、表 3.2.10.5第4列に運用に際して利用者に課金を行なっているかどうかの回答をまとめた。この結果を見ると、今回調査対象としたHPCI構成機関では、同一システムとしてハイブリッドシステムとして構成している場合あるいは異なる別のシステムとする場合の相違はあるが、異なるプロセッサやアクセラレータなど異なるアークテクチャを運用していることがわかる。また、今回調査対象としたHPCI構成機関の全てが利用者に課金を行なっていることもわかる。アカウント数は370～13,000と各構成機関によって幅があることもわかる。

表 3.2.10.5 管理対象となるシステムおよび運用についてのヒアリング結果

構成機関	対象システムおよび、HW/SW 特徴	アカウント数	利用者に課金	運用体制	ベンダ運用支援との関係
大阪大学 D3 センター	SQUID OCTOPUS 特徴： ・SQUID: CPU, GPU, ベクトルのハイブリッド構成 ・OCTOPUS: CPU, GPU, メニーコア CPU、大容量メモリのハイブリッド構成	全体で 800-1000 コメント： SQUID:OCT=6:4	YES 独自のポイント方式による従量課金	教員　2 名 技術職員　4 名 補助(事務)1 名 コメント： 技術職 4 名のうち 1 名はストレージ担当	ベンダ契約（調達時） SE,CE の常駐なし 利用者問い合わせ 利用者向け講習会 チューニング支援 コメント： 運用支援をベンダ契約(支援

					全体の 80%程度を依存)
九州大学情報基盤センター	ITO (Batch:2, インタラクティブ 2)	2600 コメント：年度末には約 3000	YES システムごとに課金。月額定額制。 コメント：負担金額の課金根拠は電気代。	教員　　3 名 幹部　　2 名 職員　　3 名 技術員　3 名 コメント： 幹部：課長、課長補佐	ベンダ契約（調達時） SE 2 名, CE 2 名常駐 利用者問い合わせ 利用者向け講習会 チューニング支援(月 1, 2 件契約)
東北大学サイバーサイエンスセンター	AOBA-A, B AOBA-S 特徴： ・ベクトルプロセッサ。AOBA-B は AMD プロセッサ。 ・Gaussian ・MATLAB(サイトライセンスで学内だけでなく全利用者が利用可) コメント： 調達単位でのシステム数。ユーザ管理と課金管理は一元化されている（ように見せている）	1540 コメント：年度末に向けて増加する。	YES 完全従量制 ・使用時間に基づく完全従量制 ・前払い ・電力量料金ベースに単価を設定。	教員　　3 名 技術職員 5 名 事務補佐員 1 名 (非常勤)	ベンダ契約（調達時） SE, CE 常駐 高速化支援、設定変更、講習会 コメント： 窓口業務、講習会は基本大学で実施。

京都大学学術情報メディアセンター	Camphor3 Laurel3 Cinamon3 Gardenia クラウド 特徴： Campor3 (CPU) Laurel (CPU) Gardenia (GPU) クラウド:試験運用 Camphor3, Laurel3 も搭載メモリ量とメモリバンド幅が異なるシステム	1400 程度 全システムでアカウントは共通管理。 コメント： 各システムでアカウントが重複する場合があるが、ユニークなユーザ数は 1400 名程度。 Camphor3(550), Laurel3(1100), Cinnamon(150), Ardenia (400)	YES 申請ノード種別ごとに消費電力を基に基本負担額を設定。 コメント： 定額で年間利用可能であるがノード固定ではない。（申請ノード数の占有利用形態も可）	教員　3 名 技術職員　3 名 事務補佐員 2 名 (非常勤) 教員は運用員会(方針等)、共同研究企画、仕様策定中心。運用は技術職員＋ベンダ常駐 2 名＋バックヤード	ベンダ契約（調達時） SE 1 名、CE 1 名　常駐＋バックヤード ユーザ問い合わせ調査 システム変更 講習会（導入部分のみ） 運用支援 コメント： 利用者窓口は大学。ベンダは大学支援の形。Redmine をつかって管理。
筑波大学計算科学研究センター	Pegasus Cygnus wisteria-O 特徴： GPU と FPGA の混載。 コメント： Cygnus: 共同研究で使用 Pegasus: HPCI,	370 全システムでアカウントは一元化され、両方(Pegasus, Cygnus)利用可能で区別されない。 (Pegasus はマテリアル・素粒子を初めほ	YES 定額制 一般利用 20% HPCI　30% 学際　50% 学際共同・国際共同利用、センター独自	教員　14 名 技術職員　3 名 事務　2 名 (非常勤) 教員は運用連絡会で合議。運用は技術職員とベンダ中心。	ベンダ契約（調達時） SE,CE 常駐は不明。（リモート） ユーザ問い合わせ調査 システム変更 計画・定期保守

	一般利用 wisteria-O:（東大運用を借用） GPU だけでは計算が難しかったアプリケーションの性能を高め、これからの時代の Strong scaling 並列処理に対する演算加速技術の確率を目的	とんどのユーザが利用）	公募に 50%目安で割り当て。 残りは筑波大学内の教育利用	事務は総合窓口（申請関連受付など）	コメント： 利用者窓口は大学側。ベンダは大学の支援という形。
東京科学大学情報基盤センター	TSUBAME 3.0 特徴： GPU 搭載のホモジニアスクラスタ ・GPU を使わないユーザのために C グループで資源を仮想化（分割）したサービスを提供している。	13000 程度 (学部生アカウント約 1000 含む) ・学部生演習でも利用	YES ポイント制 ・学部生は無課金利用可能(10 分程度） ・１GPU までは無課金利用可能(24H 迄、インタラクティブキュー４台で共有利用）。その他授業目的で特別制度あり。	教員　4+α 名 技術職員　6 名 (兼務のため、実質 3 名程度) ・教員は、教育文系も入っている ・運用は技術職員＋ベンダ	ベンダ契約（調達時） 常駐 3 名 ユーザ問合せ調査 システム変更計画・定期保守 コメント： ・チューニング業務は含まず。 ・利用者窓口は、大学側。質問一次回答はベンダ、但し大学回答が必要なものは大学で引き取

					って回答。 ・ベンダ常駐はリモートワークを許可
名古屋大学情報基盤センター	TYPE1(超並列) TYPE2(大規模GPU) TYPE3(大規模メモリ+インタラクティブ) クラウド(CPU) ・アカウントとストレージは一元化 ・OSS の整備は TYPE2 に注力、機械学習系ソフトは全サブシステム用意。 ・ログインノードは 24node（用途・サブシステム毎に分けている）用意。	ヒアリング時間の都合上見送り	YES 前払い従量金ポイント制 ・ポイントは１万円（５万ポイント）から	教員　3名 技術職員 ・常勤　4名 ・非常勤　3名	ベンダ契約（調達時） コメント： ・講習会等は大学内実施のものとベンダ発注のものを用意。 ・利用者窓口は、大学側。質問一次請けは大学。 　技術的なものはベンダ調査を依頼。 ・チューニングはできる範囲で、大学側で実施。それ以上は（仕様に明記はないが）ある程度ベンダが調査する。
北海道大学情報基盤センター	２システム ４クラウド 特徴：	約 1000 ・アカウントは一元化	YES ノード時間積の従量制（ト	教員　３名 技術サポート４名 受付事務　４名	ベンダ契約（調達時） ・利用者窓口

	・Xeon クラスタ ・Xeon Phi クラスタ ・クラウドで 4 つ（仮想、物理、GPU,ストレージ）のサブシステムが存在する。		ークン） 定額・占有利用 ・課金額は、電気代ベース ・基本負担金あり	(兼務) ・スパコン以外に学内システムも対応	は、大学側。質問一次請けは大学。 ・技術的なものやインストール等はベンダ作業を調整して対応。 ・講習会等は調達契約。実施を別途調整。 ・チューニング依頼等はあまりない。
理化学研究所計算科学センター	1 Fugaku 単一：A64FX ・約 16 万ノードからなる超大規模システム。 ・ソフトも整備中。	2741 (発行総数) ・アクティブ数は、10%程度。	あり ・基本は、資源量（NH)割当て ・有償利用もあり	35 名（電力や冷却に 10 人, それ以外は、システム、搬送システム、ハードウェア、ソフトウェアを担当）	保守契約 富士通株式会社へ発注。発注仕様に、連絡体制（週イチの連絡階の開催など）を定義することなどで関係構築。
産業技術総合研究所	1 ABCI-2.0 サーバあたり x4 または x 8 GPU 搭載 ・ファイルシステ	約 3000 弱	課金：Yes ・従量課金（GPU サーバを 1 時間利用する金額を単金としている） コメント：	23 名 （管理者、責任者含む） コメント： 実務担当は 15 名程度 ・産総研と AIST ソリュー	保守契約 ・AIST ソリューション経由 ・運用主体は産総研と AISTT ソリューションズでハードの監視

	ム：一部は共用領域を FLASH ・セキュアなオブジェクトストレージ		・性能差が異なる GPU サーバは性能計数で単価を設定。 ・運用コストも入ている（今後、他の DC 業社のコストと大きく乖離しない設定とする予定)	ションズと共同運用 ・複数のワーキンググループ（WG)体制(運営 WG,サービス提供のための運用技術 WG,設備 WG など）	や保証、ソフトのアップデートはベンダ。
東京大学情報基盤センター	2 ・ Wisteria/BDEC-01 は 2 つのクラスタからなるシステムであるが、1 システムとして管理。 ・Oakbridge-CX（更新中 →Miyabi）	全体で約 3000 名程度 ・年度毎の更新 ・無償利用の公募制度もあり	課金：Yes ・前払いの従量課金（トークン） ・基本は電気代に基づいて設定された単位時間あたりの料金形態	教員 6＋職員 5 名でシステム毎にチームを構成 ・職員側は、システム毎にメイン担当者＋サブ 1 名の設定でその先にベンダという体制。運用方針の決定、利用者問い合わせ対応を実施。	保守契約 ・個人情報に関係するユーザ管理は大学側 ・ユーザ対応の一次窓口はベンダ(対応状況は大学側でトレース）
海洋研究開発機構	3 ・ES (Earth Simulator) ・EA (Earth Analyzer) ・TS (テープドラ	ES：70 名 ・EA は 4 月から稼働なのでまだ数十名程度。 ・TS は 40 グ	内部は、課金なし 外部資金で利用される方のみ課金 ・産業利用	センター：8 名 メインとサブを複数人で横断して対応 ・運用は、シ	保守契約 利用者サポート契約 ・ベンダは常駐して稼働状況を監視。

	イブ）	ループ前後	（有償利用）を含め外部資金で利用される方のみ課金。 ・テープシステム等は、テープカートリッジ等を購入すれば無償で利用可能。	ステム毎にメインとサブを複数人で横断して３システムを対応（あるシステムのメイン担当は他のシステムのサブ担当を兼ねる）。 ・他に利用者サポートのメンバが数人(3名）。	・ユーザ対応の一次窓口はセンター。内容に応じてベンダへ依頼。 ・利用者サポートは別契約（チューニング、デバッグや移植関連）。

3.2.10.3.3.2　システムごとの運用に携わる人員・役割分担、ベンダによる運用支援の活用

次に、各HPCI構成機関における運用体制および運用支援におけるベンダとの関係について調査した。表 3.2.10.5第5列に各構成機関において運用支援に関わる人員数と役割、表 3.2.10.5第6列にベンダによる運用支援との関係をまとめた。表 3.2.10.5第5列の結果をみると、どの構成機関においても、教員、技術職員、事務員から構成される、少なくとも7名以上の運用チームによってスーパーコンピューティングシステムを運用していることがわかる。しかし、教員、技術職員、事務員の役割は各構成機関により多少異なることも見てとれる。表 3.2.10.5第6列にベンダによる運用支援の活用状況をまとめた。この結果からは、どのHPCI構成機関もベンダの運用支援を活用していることがわかる。いわゆるベンダのSE、CEについては、常駐、非常駐の差はあるが、全てのHPCI構成機関でベンダに依頼していることがわかる。利用者からの問い合わせについては、構成機関で窓口業務を行い、その上でベンダに対応を依頼する形態が多いこともわかる。スーパーコンピューティングシステム上で動作するプログラムのチューニング支援については、ベンダとの契約に含まれる場合と含まれない場合がある。これらからわかるのは、HPCI構成機関のスーパーコンピューティング運用においては、大学とベンダの混成チームで行われることがおおく、さらに、その大部分はベンダに委ねているケースが多い。ただし、その運用支援業務の切り分けは各HPCI構成機関によってことなり、共通ではないことがわかる。逆に言えば、スーパーコンピューティングシステムを導入した後に、HPCI構成機関の教員、職員のみの内部メンバで運用するケースは皆無であることがわかる。

3.2.10.3.3.3　運用の自動化、省力化等の取り組み

次に、3.2.10.3.3.2でのヒアリング結果に基づき、各HPCI構成機関での運用自動化・省力化についての取り組みについてヒアリングを行った。具体的には、以下の3点についてヒアリングを行った。

- どのような取り組みを実施しているか？
- そのために導入しているアプリケーション、ソフトウェアがあるか？
- あれば良いと思うアプリケーション、ソフトウェアはなにか？

ヒアリング結果は表 3.2.10.6に記す。

表 3.2.10.6 管理対象となるシステムおよび運用についてのヒアリング結果

構成機関	運用の自動化・省力化への取り組み、そのために導入しているソフトウェア	運用の自動化・省力化のためにあればいいなと思うソフトウェア
大阪大学 D3 センター	省電力運用 運用分析(サービス状況の監視) コメント： ・スケジューラ提供機能を利用している。改善点は多いが概ね満足。 ・統計情報の集計などは自作ツールを作成	スケジューラ機能をベースにサービス状況を監視、公開できる機能　（モニタリング負荷影響なく実現したい）
九州大学情報基盤センター	特になし。	スケジューラの機能 ・夏場に省電力を要請された際に自動で縮退運転に切り替える機能が欲しい 省電力について ・最新機種では待機電力も低いため何(DC-OFF 等)もしてない。 ・総量規制が発生する婆は手動で実施。
東北大学サイバーサイエンスセンター	スケジューラによる省電力運用 エンドユーザとのコミュニケーションツールの導入（従来の電話、メールの他に Slack を導入した）	システム監視の高度化 ・ユーザ領域のストレージ監視 　ストレージがいっぱいになることが多いので、多くなりそうだったら利用者に連絡したい。 ・運用状況の分析自動化
京都大学学術情報メディアセンター	システム監視 Zabbix システム稼働状況の可視化 Grafana 申請受付、処理の自動化（自製) スケジューラによる省電力運転 ・申請受付、処理の自動化：受け付けた利用申請の内容から、ログイン環境のメール自動送付、ジョブ利用環境の	省人化 ・利用申請からシステムへの反映まで現在は若干人が途中に入るが、ここを省人化できる自製ツールを検討

	作成や利用スクリプトのテンプレート作成などを自動化(まだ完全ではないが。)	
筑波大学計算科学研究センター	運用監視系はベンダ業務 コメント： ・申請・登録の負荷は、一般利用も公募のため利用者登録等の作業も随時に発生しない。 ・ジョブの同時実行可能数を自動調整する仕組みを導入予定	運用最適化 省人化 一般利用も HPCI 利用も優先度やジョブサイズによりキューを細かく設定。利用負荷状況に合わせて微調整が手動であることが課題。（ジョブの同時実行可能数を自動調整する仕組みを導入予定）
東京科学大学情報基盤センター	ユーザアカウントの自動発行 課題管理：Redmine コメント： ・利用者の予算管理も「人が対応しないと進まない」ことはないようにしている。 ・大学とベンダ間の課題管理はRedmain。	特になし。 コメント： ・運用状況の偏りによる構成変更に対応できるシステム設計をしている。構築段階で人が主にする必要があるシステムというのを極力作らない。 ・リモートでのシステム運用（を前提に考えている）を行っている。
名古屋大学情報基盤センター	ユーザアカウントの自動発行 大学自作ツール コメント： ・プリミティブな機能はベンダが提供。上物ツールを大学内で要望に合わせて作成。 ・システム監視等はベンダが実施。	スケジューラの待ち時間最小化のための機能が欲しい。 ・現状では、長い待ち時間が発生した場合は手動で調整している。
北海道大学情報基盤センター	ユーザアカウント発行をオンライン化（自製） ・学外利用者にパスワードを通知する際に、郵送に頼る場合がまだある。（学内利用者へは、大学が持つ既存の認証基盤を活用）	利用者へのパスワード情報の通知が基本的に郵便ベース。自動化、オンライン化できる仕組みを考えたい。特に海外の利用者対応で。

	ジョブ数に合わせて自動的に縮退運転する機能を導入 ・ジョブキューが飽和しているため実際には機能していない様子。	
理化学研究所計算科学センター	コメント： ・LC4RI (NII 開発の OSS) を活用した運用操作(手順)の管理・半自動化（富士通） ・チャットポット等による問合せの自動化も検討中。 ・環境に大きな変動が予測される場合は、手動で構成変更することもある。	未知のエラーや対処の学習と再発時の対処の自動化
産業技術総合研究所	アカウント関連はオンライン化 ・計算ノードの構成自動化：ansible ・セキュリティの改ざん検知:Tripwire ・その他、Grafana,Prometheus など	なし
東京大学情報基盤センター	登録システムは自製 ・計算ノードの構成自動化：ansible ・Grafana 等に取り組んでいる ＃構成や運用自動化についてはベンダに検討・提案いただいている。	運用の見える化で（計算機本体の消費電力状況は進んできたが）、冷却、周辺装置などの電力状況について時系列で全体を把握できるようなものが必要と考えている。
海洋研究開発機構	登録システムは自製 ユーザ管理と資源管理も自製 ・一つ登録すれば関連する登録が全て行われるシステムを構築している。	申請利用者の確認と ID 連携 稼働状況の通知 開発環境整備の改善 ・短期間に多量のジョブ投入など突発的な利用状況の変化に対して運用制限を緩和するようなしくみが必要。または、変化の発生をアラートしてくれるような通知機能であっても良い。 ・利用者の開発環境にはバリエーションが多く（使い方に変化？）、モジュール管理も厳

		しい状況になりつつある。開発環境を容易に整備しやすいものがあれば良い。

運用の自動化・省力化への取り組みとそのために導入しているソフトウェアのヒアリング結果は表 3.2.10.6の2列目にまとめた。この結果から、各HPCI構成機関ですでに実施している運用の自動化・省力化の取り組みは、モニタリング、省電力、アカウント管理、コミュニケーション手段の4つに大別できる。モニタリングとしては、スケジューラが持つ統計情報の自製ツールによる集計処理や、ZabbixとGrafanaを活用したシステム監視と可視化の取り組みがある。この取り組みの特筆すべき恩恵としては、東京科学大学情報基盤センターにおいて、Zabbixを用いたシステムの死活監視によって物理的な死活監視作業が不要になった結果、常駐サポートのリモートワークが可能となった事があげられる。省電力としては、スケジューラの機能として縮退運転を自動化する試みがなされている。ヒアリングの範囲では、NEC製のスケジューラであるNQSVとOSSのSlurmの提供する機能を活用していることを確認した。アカウント管理としては、取り組んでいる各HPCI構成機関において独自に自製したツールを利用してアカウント管理を自動化していることがうかがえ、汎用的なソフトウェアを利用しているわけではないことがわかる。また、取り組んではいるものの完全な自動化までは達成できていない機関が存在することから、アカウント管理に苦労している様子もうかがえる。コミュニケーション手段としては、利用者とのやり取りやベンダとのやり取りを簡易化する取り組みがある。具体的には、Slackや課題管理ツールであるRedmineを活用している。

運用の自動化・省力化のためにあれば良いと思うソフトウェアのヒアリング結果は表 3.2.10.6の3列目にまとめた。あれば良いと回答を得られたものとしては、スケジューラに関する要望が多い結果となった。内容としては、ジョブキューのパラメータ調整（同時実行可能ジョブ数等）の自動化や省電力運用、運用状況の分析と多岐にわたっており、スケジューラの機能拡張の必要性が高い状況にある様子がうかがえる。その他の回答としては、東北大学サイバーサイエンスセンターがアイデアとしてあげたシステム監視の高度化がある。既存の取り組みとしてシステム監視を行い、異常発生時にベンダにアラートを飛ばす仕組みを導入している機関はあるが、利用者に対して連絡する仕組みはヒアリングの範囲では確認できていない。

3.2.10.3.3.4　最も優先するシステムの整備方針、システム整備上の問題・課題、システムごとの整備方針の違い

本ヒアリング項目では、各HPCI構成機関が考えるシステム整備方針における優先事項、システム整備上の課題、ハイブリッドクラスタシステムの必要性、オープンソースと商用ベースソフトウェアに対する考え方、およびGPUの整備方針をヒアリングした。表 3.2.10.7と表 3.2.10.8、表 3.2.10.9にその結果を記す。

表 3.2.10.7の第2列目、第3列目にシステム整備方針における優先事項、システム整備上の課題にヒアリング項目の結果を示している。本サブワーキンググループのヒアリング調査においては、各HPCI構成機関においては(1)-(7)全ての項目を重視していることを感じ取れた。あえて何を重視するという観点で回答をいただいているが、多くのHPCI構成機関で当該機関の有する現利用者を重視する方針であることが見てとれた。また、その上で、利用者の拡大を重視している方針も明らかになった。この重視する項目は概ね共通と言っていいと考えられる。一方、システム整備上の課題については、各構成機関で異なる点も確認できる。例えば、大阪大学D3センターでは、ジョブ要求に対する適切な資源割り当てを行う仕組み（システムソフトウェア）に課題意識を有している一方、京都大学学術情報メディアセンターにおいてはスーパーコンピューティングシステムを設置するための物理的制約(スペース)に加えてアプリケーション性能の期待値とシステム性能の乖離をあげている。また、筑波大学計算科学研究センターで

は、時事進展する新技術とセンターの整備方針とのマッチングという点を最大の課題としてあげている。また、九州大学情報基盤センターでは、新システムで導入予定のストレージのオンライン/オブジェクトストレージ、および、課金制度の変更を課題意識として回答している。北海道大学情報基盤センターでは、電力、設置スペースをあげている。名古屋大学情報基盤センターでは、いかにして利用者ニーズに追従していくかを課題としてあげている。多様に課題意識はことなるものの、スペースについては北海道大学情報基盤センター、東京科学大学情報基盤センター、京都大学学術情報メディアセンターで共通の課題意識となっていることは注目に値する。

表 3.2.10.7 システム整備方針についてのヒアリング結果

構成機関	システム整備方針 最も優先している方針は？ (1)利用者拡大, 2)現利用者の重視, (3)重点分野の有無, (4)電力, (5)収益性, (6)他センターに対する競争力, (7)システム規模, (8)その他 システムごとに整備方針の違いはあるか？	システム整備上の課題	ハイブリッドクラスタシステムの必要性について	オープンソースベースシステムソフトウェア vs 商用ベースシステムソフトウェア
大阪大学 D3 センター	(1)利用者拡大 ・より多様な利用者が利用できるシステムを目指している ・また、より適切なコストで最大限のサービスを提供すること。 システム整備方針の違いは特にない	最適な資源量提供サービスの実現 ・適切なコストという面では、例えば最小単位のノードの資源が大きい（電力）ため 1 GPU しか使わないユーザへ４、８ GPU 資源を提供してしまっている。	必要である ・多様性の受け入れと、ドメインコンピューティングでの利用	ライセンス費 ソフトウェア品質 OSS の場合、ライセンス費削減できる点はメリット。ソフトウェア品質は、OSS の場合、安定運用、障害/セキュリティ対応にリスク（ベンダのサポート力次第）

九州大学情報基盤センター	利用者の利便性を追求 ・目的別システム（バッチ、対話型）を導入 ・インタラクティブ性（対話型）を進化させたい ・Open OnDemand の導入 ・ワークフロー的なワークロードにも対応（富岳導入のものを試験導入） システム整備方針の違いはない	ストレージ ・新システムで導入（オンライン/オブジェクトストレージ・サービス） ・課金体系の見直しも予定（定額制→従量？）：利用効率の観点で検討	必要である 次期もハイブリッドの予定 コメント： ・次期では、GPU搭載数が異なるノードが混在予定(x２、x４） ・CXL（Compute Express Link）技術の導入も検討。	ライセンス費 ソフトウェア品質 ライセンス費削減はメリット。安定運用、障害/セキュリティ対応にリスク（ベンダのサポート力次第）
東北大学サイバーサイエンスセンター	(1)-(8)まで全て重要 コメント： システム依存だが、東北大学では電力はあまり問題にならなかった。 システム整備方針の違いはない ・導入システムを	導入直後でなし。 ・特徴のある（ベクトル特化）システムを導入しているので共通的なものはない。	必要である 利用者の多様性やシミュレーション手法の適応性などで全部 vector というわけにもいかないし、全部 x86 というわけにもいかない。CXL になると使い勝手の問題が大きい。	・(品質・サポート面で)オープンベースでやるとリスクがあるとは言い難い。オープンベースのほうが情報量多い。サポートは OSS でも購入可能。

	全てセットで考えている。			
京都大学学術情報メディアセンター	(2)現利用者の重視 (4)電力 (7)システム規模 ・リプレイスして、今よりもよりよく使えるシステムを整備したい。 システム整備方針として下記の通り。 ・性能重視 ・汎用性重視 ・大容量 ・GPU (需要トレンド) ・スペックは既存利用者のアプリ諸元を加味して継続性を担保できるような形で考えている。	物理的制約 実アプリ性能 予算 ・建屋的なものや水冷などの設備的なものなどの物理的な制約の方が大きくなりつつある。 ・アプリ性能が期待通りにならないケースがあり、利用者に負担が大きい。	必要である ・汎用性を考えるとクラスタ単位のハイブリッドである必要がある。 （単純にサブシステムが一つのストレージにつながるというものではなく、ドメインコンピューティング的なイメージ）	コスト セキュリティ サポート セキュリティ観点で商用でなければリスクが高いものは商用で、セキュリティが関係しない、または担保でき、安定稼働可能である場合は、コストメリット＋調達仕様を満足できればオープンを選択。 ・指定ソフトがあれば商用を導入せざるを得ない。

筑波大学計算科学研究センター	(2)現利用者の重視 (1)利用者拡大 (3) 重点分野の有無 ※　(1)(3)は同程度重視。 利用アプリ分野(マテリアル・生物・素粒子）でデイ残しながら設計。アプリ（超並列などのスタイル含め）を高速に動かすというスパコンを設計し、現状で最適なマシンを導入する。 システム整備方針として下記の通り。 ・性能重視 ・汎用性重視 ・アプリ（ワークロード）が多様でストレージ、メモリ、ネットワークなどの最適解を求めるのが整備方針。そのためにシステムが分かれて	新技術とセンター要求のマッチング 予算 FPGA などはいくつかユースケースがあって、元々宇宙のアプリにGPU だけ速くならないもの、GPUで速くなるものがあって、それを両方使えるようなシステム案がベースになっている	基本的に必要なし 費用的観点からが主な理由。使いやすさでいうと各ノードとシングルソケットで、シングルGPU。ノード内もNUMA とか考えなくてそのまま早くなるという感じで、それを複数ノードみんな均一。	コスト 必要であれば ISC でも OSS でも導入する。OSS のサポートメンバもいる。OSS が良いのは利用者が多いから。利用者が多いということは使い方がわかっている。問題がおきても、対応オプションが多いこと。使えるオープンソースを選定することが重要で、利用価値がある。 オープンソースでもサポート有料で使用していることがおおい。 商用ベースは選択肢がない場合は、利用価値が高ければ proprietary なものでも選択する。

	いる。			
東京科学大学情報基盤センター	(2)現利用者の重視 (7)システム規模 ※ (2) のための (7) ・現利用者も新規利用者も大事。TSUBAME もスパコンの敷居を下げるという当初からの目的もある。 ・(3) 重点分野は特に意識したことはない。 システム整備方針：N/A	建屋、電力 ・調達という意味では建屋、電力などの制限の中で実現可能性の検討。現利用者への継続性という観点で大きなアーキテクチャ変更を避けた正当進化的な感じのシステムを検討せざるを得ない。	必要ない ・ハードウェア的にハイブリッドにした場合、需要の偏り等が発生するので、あまり望ましいとは考えていない。	コードのポータビリティ ・ベンダーロックインになることは望ましくないと考えている。 ・開発環境にあたるソフトウェアは OSS が望ましい ・ソフトウェア的な TSUBAME のコピーを必要に応じて作れることをメリットと考えている。
名古屋大学情報基盤センター	(1)利用者拡大 (4)電力 サブシステムごとに環境を変えている。 ・計算性能向上、効率的運用、セキュリティ強化 ・全体としては違いはないが、各サブシステムで使い	利用者ニーズへの追従 ・AI 研究、データ科学への対応など。 ・アカウントに限らず、様々なデータを収集してもらうこと。	必要である ・そのために 4 つのサブシステムを導入している	管理コスト ・OSS は更新が多くまたアプリによってはバージョン固定となることがあり最新を入れられないことがある。 ・OSS のサポートは仕様で定義。 ・運用系にあたるソフトは何でも可

	方が異なるため対応が異なる。 TYPE1 は超並列で並列環境整備。 TYPE2 は AI で Docker 利用、仮想化技術対応など整備内容が異なる。			
北海道大学情報基盤センター	(2)現利用者の重視 (4)電力 (5)収益性 ※ (4), (5) は同程度重視。 ・システムのアーキテクチャを変える調達の時は(2)が最も重要 新しい技術へのチャレンジ ・クラウドは新しい技術を評価するということを含めて、チャレンジしている。スパコン側は安定してユーザにとって使い勝手がいいものにしようという方針で取り組んでいる。	電力 設置スペース ・センター内の研究的な側面とは切り離して調達を検討。 ・調達毎に新しい技術にチャレンジすることを命題にしている。	必要性は不明。 ・次期システムはハイブリッドになり、部分的に GPU を搭載した形になる。 ・ただ、 1 システム内で構成するか、 2 サブシステムで導入するかはまだわからない。	コスト セキュリティ ・OSS はトレンドから外れると更新が期待できず継続性に課題が残ることも考えられる。 ・ポータビリティの問題は資源の種類やソフトのバージョンの違いなど要因が様々で OSS だけの問題とも言えない。 ・セキュリティインシデント対応が大変そう ・商用ベースであれ、OSS ベースであれ調達時にサポート入れているのでベンダの力量次第の面がある。

理化学研究所計算科学センター	(2)現利用者の重視 (4)電力 (1)利用者拡大 コメント： ・(2)として「使いやすさ」、これは(1)にもつながる。 ・利用者拡大としては間口を広げる方向で検討	・利用アプリ側の要望に沿う方向で決まる。 ・アプリケーションを透明性、公平性を持って選び、そのパフォーマンスを最適化するという整備が基本。また、更新期間(10 年)が長く、陳腐化への対応（更新の追随性）も課題。	メインシステムをハイブリッドにするかという問題は、計算科学ロードマップに示されるアプリ性能を達成できるものを整備することに依存するが、プリポスト環境に必要性があれば導入する。	・トレードオフがある。OSS の利用環境はのぞましいが、アーキテクチャ的にニッチなものになると OSS が採用してくれない。 ・自前で頑張ってもソフトウェアエコシステムに乗り遅れることもあるし、Update が個々の利用グループで承認されないケースも発生する。 ・OSS と言えども、Update の受け入れでカスタム状態が発生しうる）のデメリットの裏返しとしてプロプライエタリが良い場合もある。 ・商用にもベンダ次第でサポートリスクはある。
産業技術総合研究所	(3)重点分野 ・今後生成 AI 分野を強化していく。	電力 ・フルシステムを運用していく上で電力キャップをいかに抑える形でシステムを運用して	導入時期が異なる 2 種類の GPU サーバから構成されているが、基本的にはヘテロな資源提供は不要。要求リソース量などで分割提供は実	先進性、コスト vs 安定性 ・最先端の研究成果がいち早く公開され、試せるというメリットがある。

		いくかということが課題となると考えている。 ・スパコン調達であるためにある意味タイムリーな整備ができないということも課題。	現済み。	・安定したサービスを提供する視点では、ベンダからサポートを受けられない可能性がある。 ・産総研では、整備したシステムをリファレンスとして民間に技術移転することも目指しており、移転しにくい（OSS だけど安定性に問題があるなど）と移転に課題が残るため,運用スタックとアプリ開発のソフトウェアスタックを是々非々で選択している。
東京大学情報基盤センター	先進性 (4)電力（効率） ・利用者には負担となったが、次期システムでは、GPU への移行を進めた。	・CPU 主体ユーザを GPU への移行を進めるために、2 年程度前から移行をサポートする必要があった。その意味では、費用面だけでなく時間も必要であった。	必要 ・適材適所の計算資源が利用できることは重要。	先進性とコスト vs 安定性、利用の継続性 ・構築システムのシステムソフトウェアはほぼ OSS を利用。デメリットとしては、「保証されない」という点があるが、調達なのでベンダのサポート力と費用のトレードになる。

				・OSS の選定では最低限のサポートレベルを決めて選定。 ・（アプリは利用者マターとしているため）システムソフトウェアとシステムサービスについてはサービスレベルを決めて選定。
海洋研究開発機構	(2)現利用者(性能含めて) (3)重点分野 ・基本的に現利用者の利用を把握して、どれくらいの性能が必要かなどを重視する。 ・研究者からの要望で AI を使いたい（＝GPU 導入）ということも重要だと考えている。	実行性能 ・利用者のプログラムがどれ位の性能を出せるかということに対して見積もり、対応していくこと。 ・電気などの高騰もあるが、高性能半導体調達では予算面も課題。	必要（ES で導入） ・EA （Earth Analyzer)も一部は大容量のメモリが必要になることを考慮して使い方による選択を可能にしている。	コスト vs 利用継続性 ・研究ではディスクサイドで新しいアルゴリズムを試して、規模大きくしていく。最初のステップの環境(OSS が多い）と親和性が高いことが望まれるように感じる。 ・システム調達後（運用中）に使用している OSS の開発、サポート（セキュリティ対応など）が終了したり、有償ライセンスに切り替わることによる影響がある。 ・地球科学の研究

				者は、プログラムの公開ということには関心が薄いかもしれない。研究成果として出るデータや分析結果の公開に重点を置いているので、ISVが必要であれば使うというスタンス。

3.2.10.3.3.5　ハイブリッドクラスタシステムの必要性、オープンベースおよび商用ベースソフトウェアを用いた整備に対する管理者の考え方

続いて表 3.2.10.7の第4列目、第５列目に、それぞれハイブリッドクラスタシステムの必要性、オープンベースおよび商用ベースソフトウェアの導入に対する考え方についての回答結果をまとめた。これらから、今回対象としたHPCI構成機関のほとんどが、ハイブリッドクラスタシステムの必要性を認める結果となっている。ただし、本回答には、システムの定義、すなわち、“ハイブリッドシステム”とは何を意味するか？（本サブワーキンググループの定義では、同一システム内（クラスタ内）で異なるノード構成が組み合わされたシステムとしていた）というところに議論があり、1システム内で複数のアーキテクチャを導入する、異なるアーキテクチャのシステムを複数導入するという点で考え方にブレがある点を記載しておきたい。しかしながら、1構成機関で複数の異なるシステムを保有する必要性を概ね全ての構成機関で有していると言える。オープンベースおよび商用ベースのソフトウェア導入については、概ね２つの考え方に分類されるといえる。1つの考え方は、オープンベースのソフトウェアは商用ライセンスを削減できるが、セキュリティやベンダサポートにリスクがあるという考え方である。もう一方の考え方は、オープンソースでもベンダのサポートをできるので問題にはならず、またオープンソースの方が利用者も多く情報も多いという考え方である。どちらの考え方においても、オープンソースの導入はベンダのサポート力に依存する部分も大きいように思われる。

3.2.10.3.3.6　GPU の整備方針

表 3.2.10.8に、GPUの整備方針に関連して質問した、I社、A社のGPUに対する関心についての回答結果をまとめた。これらから、今回対象としたほとんどのHPCI構成機関では、各々が把握している範囲においてI社、A社のGPU製品に対する利用者からの積極的な関心はないという結果となっている。システムを導入するHPCI構成機関の視点としては、I社、A社のGPUも、GPUを導入するうえでの候補として捉えられている。ただし、電力効率や価格といった面では魅力があるものの、利用者にとっての使いやすさという面で懸念のあるGPUであるとも捉えられる傾向にある。特に、東京大学では、実際にNVIDIA以外のGPUが採用された場合に備えた対策を準備している。

表 3.2.10.8　GPU の整備方針について：I 社、A 社の GPU に対する関心

構成機関	I 社、A 社の GPU 製品について、ユーザサポート、利用者交流機会等での利用者の関心の度合いを教えてください。	I 社、A 社の GPU について、センターとしてどのように考えるかについて教えてください。
大阪大学 D3 センター	一部の利用者から A 社の GPU 製品について関心がある旨を聞く場合はある。しかし、センターとして、A 社の GPU を評価できる環境もないため、ユーザサポート、利用者交流機会等の場は設けられていない。 I 社については、GPU ではないが、Gaudi を N 社の GPU の代替にならないかを検討したことがある。 センターとしては、利用者のワークロードに最適（性能面、つかいやすさ）であることを優先している。 回答としては、現状では、I 社、A 社の GPU 製品について、利用者に対する情報提供、利用者支援に至っていない。	調達においては、利用者のワークロードに最適（性能面、つかいやすさ）であることを優先している。当然コスト的な視点も検討に入る。 現状の具体的な考えとしては、A 社、I 社の製品について、N 社の GPU を利用する利用者に対しても、GPU を利用していない利用者に対しても、情報が不足しており、現状では、N 社の製品しか選択肢がないと思える。
九州大学情報基盤センター	今のところ、具体的な問い合わせや要望を聞いたことはない。ただ、自作ソフトを公開している利用者から、GPU 化に向けて OpenACC のように N 社限定のものと、OpenMP のように可搬性のあるもののどちらを使うか迷っている、という話を聞いたことはある。	電力対性能、価格対性能では、N 社の GPU と同等もしくはそれ以上の魅力がある。特に GPU についてはハウスコードよりも既存アプリケーションを利用するユーザが多いと思われるため、アプリケーションが A 社、I 社の GPU にどの程度対応するかに注目している。
東北大学サイバーサイエンスセンター	（未回答）	（未回答）

京都大学学術情報メディアセンター	現行システムを含めて、I 社、A 社の GPU を導入したことはないので、これらの GPU に向けた講習会等は実施していない。また、その他の講習会等で、これらの GPU に関する要望は特に聞いていない。	GPU を備えたシステムを導入する場合の候補となるかと思います。
筑波大学計算科学研究センター	利用者は NVIDIA でコードを最適化しており、あまり他の GPU 製品を利用するメリットは感じていない。	基本中立である。
東京科学大学情報基盤センター	NVIDIA 以外の GPU は、利用機会・各種アプリ対応などに関して大きく劣り、ユーザからの問い合わせはほぼ無い状況である。	左記のような問題はあるが、供給できる性能と、導入コスト・電力コストのバランスに応じて、ぜひ前向きに検討したい。ただし、GPU ベンダを巻き込んだ講習会などを導入の十分前から進める必要がある。
名古屋大学情報基盤センター	本学の GPU ユーザは AI 関連のユーザが多く、ソフトウェアの整備が進んでおり、現状問題なく使えている N 社の GPU から乗り換えるモチベーションは少ない。	A 社の GPU 製品については、計算機システムを導入する立場として、コスパの面で関心はある。しかしソフトウェア周りが弱い（AI、HPC では Fortran への対応）ことがわかっており、本学の HPC ユーザは Fortran 利用者が多いため、試用する環境は欲しいが、大規模に導入する予定はない。I 社の GPU は現時点で検討に値しない。
北海道大学情報基盤センター	現在稼働中のスパコンは GPU を搭載しておらず、利用者と接した範囲では、N 社の GPU に対する関心ですら限定的であり、I 社や A 社に関しては話題になった記憶がない。	現時点でセンターとしての具体的な考えはない。
理化学研究所計算科学センター	あったとしても把握できていない。	・ソフトスタックとしては未熟だが、N 社の CUDA レベルであれば変換が可能であると認識している。 ・OpenACC については課題があると認識し、議論を始めている。
産業技術総合研究所	特に関心はない様子。 ・N 社 GPU とソフトウェアスタック	・具体的な計画があるわけではないが、設置要件等が合致すれば、小規模でも他社 GPU を試したいという潜在的な利用者がいること

	を提供するというサービスからこれら以外の GPU を使用したいという要望は承知していない。 ・生成 AI の開発者に計算機利用料を支援するという枠組み（経産省）では、N 社や N 社以外では AWS の生成 AI トレーニングで使っている様子。	も考えられるので検討したいと考えている。
東京大学情報基盤センター	新システムへの GPU 導入にあたって、今後 NVIDIA 以外の GPU を導入することになった場合に対する懸念はある。ただし、利用者からの I 社、A 社の GPU に対する具体的な関心や要望といったものはない。	センターとして、マクロベースでの開発や利用者の要望への対応ができるように準備している。
海洋研究開発機構	特に関心はない様子。 ・I 社、A 社の GPU が一般的に使える状況が少ないのでまだないと考える。	センターで実行されているプログラムが ・どれだけ動くか（ポータビリティ） ・どれだけの性能で動くか（性能） が関心の対象。ただ、現在は確認できる状況ではないが、確認したい。

また、表 3.2.10.9に、導入システムの事前評価について質問した回答結果をまとめた。この質問によって、導入システムの事前評価においてNVIDIA以外のGPU製品を対象とした事前評価がなされているかどうかから、各HPCI構成機関のNVIDIA以外のGPU製品に対する関心度合いを把握することをねらった。この表から、まず、質問したHPCI構成機関のうち、約半数の機関において事前評価を実施していることがわかる。次に、それらの機関では、NVIDIAのGPUに加えてA社のGPU製品を採用した構成について事前評価している場合が多いこともわかる。I社のGPU製品に関しては、東京大学のみ評価の対象としている。さらに、こうした事前評価で利用するソフトウェアスタックについてほとんどのHPCI構成機関は、指定するソフトウェアが利用可能であるかどうかという点を重要視していることもわかる。これは、各HPCI構成機関の重要視するワークロードが動作可能であるかという移植性が重要な評価項目となっているためと考えられる。事前評価をしているわけではないものの、大阪大学もアプリケーションの移植性を重要視しているとのことであった。

表 3.2.10.9　GPU の整備方針について：導入システムの事前評価

構成機関	導入システム決定時または、将来導入システム整備に向けた事前評価について、事前評価を実施した経験はありますか？	（事前評価の経験がある場合）評価構成について教えてください。（CPU：x86 系（I 社、A 社）、独自 CPU、CPU＋GPU 組み合わせ：x86 系＋GPU（N 社、I 社、A 社）など）	（事前評価の経験がある場合）評価システムのソフトウェアスタックについて重要視する点（した点）を教えてください。
大阪大学 D3センター	なし ・事前評価の必要性は、今後必要であると考えている。 ・現状では評価機がなく、事前評価は実施できていないのが現状。	N/A	N/A 性能に加えて、利用者のプログラミング容易性や、現在 GPU を利用している利用者の移植性は重視したい。ソフトウェアスタック整備状況などは基準点となりうる。
九州大学情報基盤センター	あり	CPU: x86 系（I 社、A 社） CPU＋GPU 組み合わせ：x86 系＋GPU（N 社、A 社）	公開されている多くのアプリケーションやライブラリ等を改変せずに使用できること。
東北大学サイバーサイエンスセンター	（未回答）	（未回答）	（未回答）
京都大学学術情報メディアセンター	あり	x86＋GPU（N 社と A 社）	C と Fortran から GPU 利用向けに OpenMP が動くこと。
筑波大学計算科学研究センター	基本的にあり ・新しいアーキテクチャが候補となるときは基本的に実施している。	事前評価はベンチマークを提供して実施してもらっている。アーキテクチャはこちらで指定する訳ではなくベンダの提案構成となる。	指定したベンチマークが実行可能であること。それ以外は仕様書で指定する。

東京科学大学情報基盤センター	あり ・小規模システムで実施したことがある。	事前に入手可能な x86 系 CPU と、NVIDIA GPU、ごく一部の AMD GPU。	この時点では基本的なソフトウェア開発環境を用いた。
名古屋大学情報基盤センター	あり	・x86 系 + N 社 GPU ・x86 系 + A 社 GPU	Fortran への対応
北海道大学情報基盤センター	なし	N/A	N/A
理化学研究所計算科学センター	なし	N/A	N/A
産業技術総合研究所	なし （過去の調達では、未発表製品：GPU を調達しているので、現行品での BM 評価は実施している）	N 社のみ実施	AI のワークロードとしてオープン系のデファクトまでとは言わないが、広く使われているソフトウェアを採用し、評価している。
東京大学情報基盤センター	あり ただし、調達決定後にベンチマークで確認している。	A 社、I 社とも実施。 CPU もかなりの数を実施。	利用者のアプリケーションをベースとして評価している。
海洋研究開発機構	なし	N/A	N/A

3.2.10.3.4 HPC センターの現状について：導入ソフトウェア・ライブラリの利用状況や整備状況

本ヒアリング項目より「2. HPCセンターの現状について」のパートになる。本パートでは、2022年度に作成した「HPCI構成機関におけるシステムソフトウェア導入状況」について再確認を行うとともに、それらのソフトウェアがどの程度利用されているかを口頭にて確認する。その後、計算機システムにおけるニーズ、要求として運用サービス（主に、利用者からの計算要求を処理する仕組み）、クラウド利用、仮想計算機技術、データ基盤等についてヒアリングした。

まず、本ヒアリング項目では、図 3.2.10.6に示すスライド及び前年度に作成したシステムソフトウェア・ライブラリ調査表を用いて、導入システムソフトウェアの再確認及び導入ソフトウェア・ライブラリの利用状況・整備状況のヒアリングを行った。

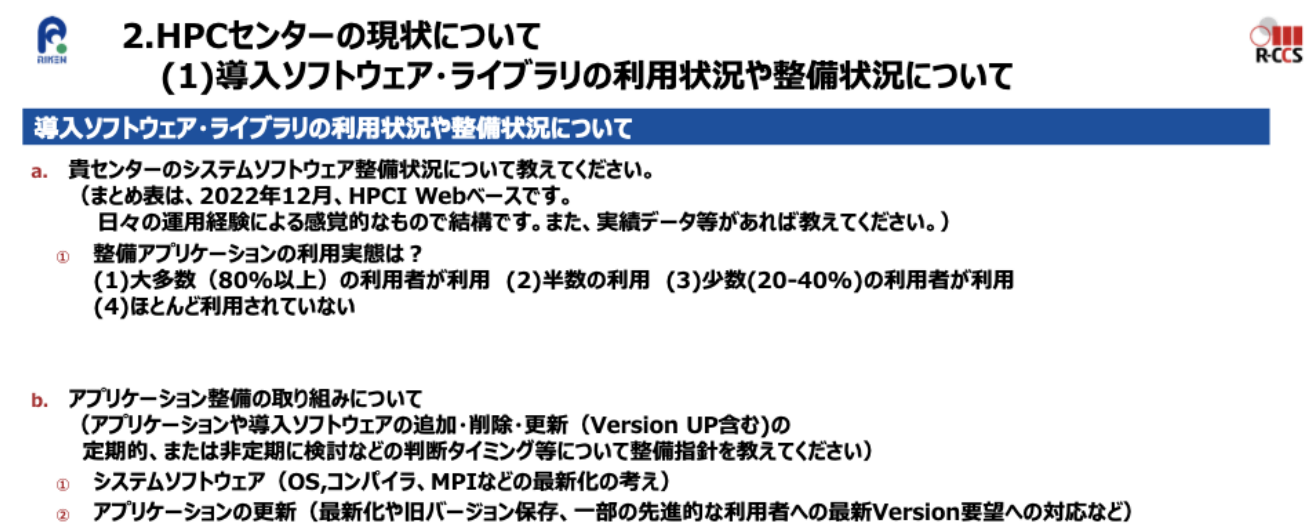


図 3.2.10.6 導入ソフトウェア・ライブラリの利用状況や整備状況についてのヒアリング項目

3.2.10.3.4.1 システムソフトウェア整備状況について教えて下さい

当該項目では、「HPCI構成機関におけるシステムソフトウェア導入状況」を再確認することで、Web上には掲載していなかったシステムソフトウェア・ライブラリを洗い出すことをねらった。再確認の方法としては、前年度に作成したシステムソフトウェア・ライブラリ調査表を見せながら、項目にはないけれど実は導入しているソフトウェア・ライブラリがないか質問した。また、導入しているシステムソフトウェア・ライブラリに関してはその利用実態について質問し、実際の利用実態を把握しているかどうかを確認するとともに、各HPCI構成機関の計算機システムごとに利用されているシステムソフトウェア・ライブラリの実態の把握もねらった。ヒアリングの時間内に聞くことができなかった部分に関しては、後日メールによる情報共有をしていただくことでシステムソフトウェア・ライブラリ調査表に抜けや、利用実態について確認した。本成果報告では、現時点で得られている確認結果から以下の2点について報告する。

- 導入しているシステムソフトウェア・ライブラリの利用実態を把握しているかについて
- 国プロのアプリケーションの利用実態について

まず、導入しているシステムソフトウェア・ライブラリの利用実態を把握しているかについて、ある程度までの利用実態はデータに基づいて把握できているHPCI構成機関が多い印象を受けた。今回調査対象としたHPCI構成機関のうち、7つのHPCI構成機関においては、利用者によるモジュールロードの実行時にログを残す仕組みを導入している。このログを見ることで、導入しているシステムソフトウェア・ライブラリの利用の有無はある程度わかるようになっている。しかし、この方法で記録可能なログには、開発時でのモジュールロードの実行ログや並列分散実行時に各計算ノードで実行したモジュールロードのログも含まれてしまうため、利用の有無以上に正確な利用実態を調査するには不向きな方法である。また、この方法では、利用者自身で用意したアプリケーションの実行等は追跡できないため、その点も実態の把握が困難となる。

次に、国プロのアプリケーションの利用実態について、今回対象とした各HPCI構成機関に質問したところ、7つのHPCI構成機関からは、ほとんど使われていないとの回答が、2つのHPCI構成機関からは、使われているようだが少ない、もしくは多くはないとの回答が得られた。利用実態に関してこうした明確な回答を得られているのは、国プロアプリケーションのモジュールロードのみに関しては、ラッパーが記録したログより正確な実行回数を把握する仕組みが調査対象とした全てのHPCI構成機関で導入されていることが理由である。

3.2.10.3.4.2 アプリケーション整備の取り組みについて

当該項目では、アプリケーション整備の取り組みとしてシステムソフトウェアとアプリケーションの追加・削除・更新の

整備方針についてヒアリングを行った。ヒアリング結果を表 3.2.10.10に示す。

表 3.2.10.10 アプリケーション整備の取り組みについてのヒアリング結果

構成機関	システムソフトウェアの整備方針（OS、コンパイラ等）	アプリケーションの整備方針
大阪大学 D3 センター	・安定稼働を優先し、問題発生時に更新の要否を判断。	・OSS については利用者希望にあわせて積極的に導入 ・ISV については導入することでシステム利用増が見込めると判断した際に導入 ・バージョンアップは、依存関係などを考慮して慎重に行う。年度末に行うことが多い。 ・セキュリティ対応では、都度判断。緊急性が高いと判断されたものは行う。
九州大学 情報基盤センター	・安定稼働を優先し、問題発生時に更新の要否を判断。	・主体的にはあげない。利用者からの要望に基づいて更新。 ・機械学習系のアプリケーションは利用者側で準備する仕組み。
東北大学 サイバーサイエンスセンター	年数回の保守時（年度末、期末）にまとめて整備の見直しをかける。	・統計に基づき、利用実績のないアプリケーションは計算機システムの更新時に削除。
京都大学 学術情報メディアセンター	・年数回の保守時（年度末、期末）にまとめて整備の見直しをかける。 ・Environment modules を採用しており、デフォルトバージョンは年度末の保守時にのみ更新する。 （・障害対応や脆弱性対応、サポート切れの際にも適宜見直す場合がある。）	
筑波大学 計算科学研究センター	・基本は年度末、期末の計画定期保守時点で見直しをかける。 ・障害対応や脆弱性対応、サポート切れでも適宜見直す場合がある。	・アプリケーションは利用者側で準備する方針。
東京科学大学 情報基盤センター	・副作用のあるものは年数回の定期メンテナンス時に安定バージョンに更新。 ・副作用のないものは新しいバージョンが出るたび適宜更新。	

	・Environment modules を採用しており、旧バージョンは保存。 （デフォルトバージョンは年度頭にのみ変更）	
名古屋大学 情報基盤センター	N/A（ヒアリング時間の都合上見送り）	
北海道大学 情報基盤センター	・年度末の保守時に、ベンダからの提案に基づいて実施。 （デフォルトバージョンの更新もこの時期に検討）	・アプリケーションの更新は、利用者の要望に応じて実施。 ・アプリケーションの追加は、利用者の要望に応じて不定期に検討。 ・アプリケーションの削除は、脆弱性対応が困難な場合のみ。
理化学研究所 計算科学センター	OS：RHEL8 を採用しており、保守タイミングでの最新化を検討。（セキュリティ関連は内容により適用時期を判断） MPI などミドルウェア：基本はバグ FIX のみ。ベースとした OSS の版数 update への追従は課題。	Update 内容のリスク（性能低下や不具合）を判断しながら実施。 ・古い版を残しながら実施することもある。 ・OSS に関しては年 1,2 回。要望も含めて検討。
産業技術総合研究所	定期メンテナンスに合わせて実施。 ・OS 等のメジャーアップデートは基本 4 月	アプリケーション、ライブラリ等システム停止が不要なものは不定期にアップデート。
東京大学 情報基盤センター	非定期に実施するものと定期(1 回/月)で実施するものがある。 ・定期(1 回/月)では、HW 関連、予防保守、OS アップデート等を実施。 ・安定しなければ戻すこともある。 ・セキュリティ関連は随時。	非定期に実施するものと定期(1 回/月)で実施するものがある。 ・アプリの更新は基本、ユーザからのリクエスト次第。定期(1 回/月)に実施。
海洋研究開発機構	定期保守時に実施。 ・安定バージョン、デフォルトバージョンに変更があれば検討。 ・コンパイラや MPI もアプリケーションと同じ考えで実施。	定期保守時に実施。 ・バージョン依存という人も結構いるので、古いバージョンを残しておいて新しいバージョンを試しに使っていただく。少しずつ移行する形。 ・ユーザ要望で新バージョンへ移行というのは多い。

システムソフトウェアの整備方針のヒアリング結果は表 3.2.10.10の列2にまとめた。結果からは、回答を得られ

たHPCI構成機関のほとんどでは、定期的なシステムソフトウェアのバージョン更新の方針を採用していることがわかる。さらに、この定期更新の実施を保守と合わせて実施する傾向が強いこともわかる。これは、定期更新による利用者への影響を少なくしようという方針を、今回対象としたHPCI構成機関が有することを示していると考えられる。特に、京都大学学術メディアセンターや東京科学大学情報基盤センター、北海道大学情報基盤センターでは、Environment modulesによって有効化されるシステムソフトウェアのデフォルトバージョンの変更も年1回のみに収まるように整備しており、利用者に影響が出る項目への慎重さがうかがえる。

アプリケーションの整備方針のヒアリング結果は表 3.2.10.10の列3にまとめた。結果からは、システムソフトウェアの整備方針とは異なり、アプリケーションの整備を定期的に実施する機関と不定期に実施する機関にわかれていることがわかる。また、方針として部分的ないしは一律でアプリケーションの準備は利用者がすることと事前に決めているHPCI構成機関があることもわかる。これらは、アプリケーションの整備の実施に積極的なHPCI構成機関と消極的なHPCI構成機関に分かれていることを示していると考える。可能性として、この分かれ方は、アプリケーションの利用実績を把握しているかどうかによって生じていると考えられる。積極的にアプリケーションを整備する京都大学学術情報メディアセンター、東京科学大学情報基盤センターでは、「3.2.10.3.2.4利用者の実態を把握・管理する方法」で言及した通り、内製のツール等で導入アプリケーションの利用実績をある程度把握している。このため、実際に使われており、バージョン更新の需要が高いと分かっているアプリケーションのみを計算機システムとして導入しているため、積極的な整備方針を採用していると考えられる。一方で、データとしては導入アプリケーションの利用実績を把握していないHPCI構成機関では、主に利用者の要望でしか導入アプリケーションのバージョン更新の需要を知ることができないため、利用者の要望を受けての消極的な整備方針を採用していると考えられる。

3.2.10.3.5 HPC センターの現状について：運用サービス

本ヒアリング項目は、図 3.2.10.7、図 3.2.10.8に示すスライドを用いてヒアリングを行った。

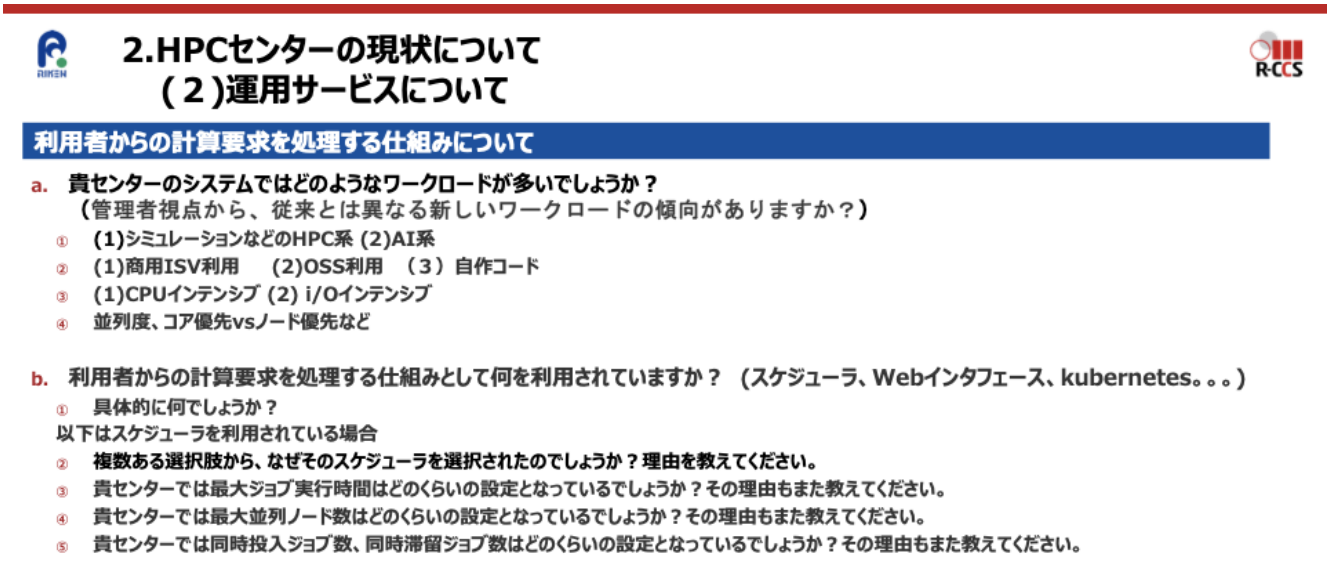


図 3.2.10.7 運用サービスについてのヒアリング項目 1

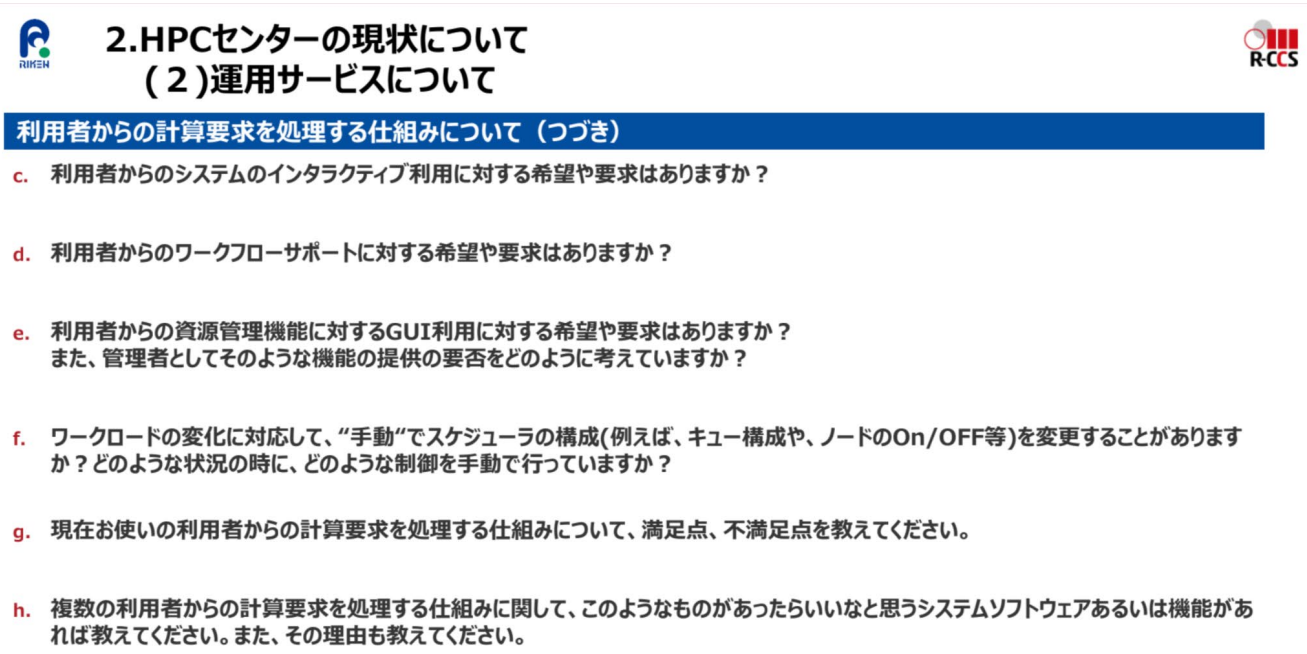


図 3.2.10.8　運用サービスについてのヒアリング項目 2

3.2.10.3.5.1　ワークロードの傾向

表 3.2.10.11に図 3.2.10.7に対応するヒアリング調査結果を示す。表 3.2.10.11の第2列目には、各HPCI構成機関の所有するスーパーコンピューティングシステムについて、どのようなワークロードが多いのか？を調査した。より具体的には、ワークロードが

- HPC 系、AI 系
- 商用ISV 利用、OSS 利用、自作コード
- CPU インテンシブ、I/O インテンシブ、メモリ
- 並列度、コア優先 vs ノード優先など

であるのかという点からヒアリングを行った。この結果から、東京科学大学情報基盤センター、名古屋大学情報基盤センター以外のHPCI構成機関では、主にHPC系のワークロードが主であることがわかる。これらのHPCI構成機関においてAI系のワークロードがある一定数存在していることもわかる。実行されるプログラムの商用ISV、OSS利用、自作コードへの分類結果からは、東北大学サイバーサイエンスセンターで実行されるプログラムは自作コードが多いことがわかる。それ以外の構成機関においては自作コードの割合が全体的に少ない傾向がみてとれる。実行されるプログラムの並列度、例えば、コア優先、ノード優先なのかについての質問項目については、今回対象となったHPCI構成機関のほとんどで1基の計算ノード内の複数コアを用いるようなノード内並列ジョブが多いということが判明した。

表 3.2.10.11　運用サービスについてのヒアリング結果 1

構成機関	どのようなワークロードが多いか？ a-1: HPC or AI a-2: 商用 ISV 利用 OSS 利用	利用者からの計算要求を処理する仕組みとして何を利用しているか？ その選択理由は？	最大ジョブ実行時間	最大並列ノード数	同時ジョブ投入数、同時ジョブ滞留数

	自作コード a-3: CPU-intensive I/O-intensive memory-intensive a-4: 　並列度、コア優先 vs ノード優先など				
大阪大学 D3 センター	a-1: HPC HPC 系がほとんど。AI は一定数あると思うが、GPGPU が古いためそれほど多くない。 a-2: (1)-(3)のどれが多いというのは不明だが、全てある。 a-3: memory-intensive a-4: ノード内がほとんど	スケジューラ NEC 製 理由： (1)アセット継承 (2) ベンダ提案（調達結果） ・基本は調達仕様を満足するスケジューラ(ベンダー提案)を採用 ・～512node までのジョブの最大実恋時間は一律。 ・デバッグ利用は 10 分が最大。 ・インタラクティブ利用は SQUID：10 分、OCTOPUS:120H が最大。	120 時間 ・利用者からの相談で柔軟に対応可。 ・あまり長くすると緊急メンテナンス計画や障害対応に影響がある。	512 ・インターコネクトのホップ最小(レイテンシ)ノード数。	現状制限している。 ・制限は無くしたいと思っているが。性能影響を考慮して制限がかかっている。スケジューラの問題。

九州大学情報基盤センター	a-1: HPC HPC 系がほとんど。AI は一定数あると思うが、GPGPU が古いためそれほど多くない。 a-2: (1)-(3)のどれが多いというのは不明だが、全てある。 a-3: CPU-intensive a-4: ノード内がほとんど	スケジューラ富士通製 理由： (1)ベンダ提案 (調達結果) ・調達仕様書には「それ」しかできない仕様ではない)	バッチ 168 時間 対話型 24 時間 ・180H を超えると資源利用の公平性が担保できないため。 ・ Batch:~4node:48H,~16node:24H,~64node:12H,~128node:6H ~256node:6H	バッチ 256 ノード ・ 128 node までの利用が常態。 ・256 node があまり流れることはない。	滞留ジョブ数 128 前後 ・備考：滞留ジョブ数（=スケジューリング対象になっているもののみ)
東北大学サイバーサイエンスセンター	a-1: シミュレーション a-2: a-3: (3)自作コード a-4: 1VE が多い。 ・AOBA_B は 32VE 近く使っている人います。 （これまでは、システムサイズガッチ盛	スケジューラ NEC 製 理由： (1)ベンダ提案 (調達結果) ・調達仕様を満足するスケジューラをベンダ提案	72-720 時間 (デフォルト 3-4 日) ・ユーザのジョブの統計をとって何時間で終わるかをみて決定。 （ほとんどものが	256VH ・256VH. はシステムの半分	混雑具合で変更 (監視して、手動変更)

	ったことにより 1 VE が多い) ・AOBA-S は始まったばかりなのでわからない		72h 以内で計算している)		
京都大学学術情報メディアセンター	a-1: システムごとに異なる。AI 系の利用は「まだまだ」の感じ。 a-2: ・全体の傾向としてノード数の多いジョブは OSS の傾向あり。 ・ISV もいくらかはある(が今回のシステムでは商用ライセンスを縮小している)。昔からの利用者は自作コードも所有している。 a-3: 不明 ・利用実態は把握できていない。 a-4: ・大規模もある。 ・１ノードジョブを投入する利用者は多い。	スケジューラ SLURM 理由： (1)ベンダ提案 (調達結果) ・調達仕様を満足するスケジューラをベンダ提案 ・大学独自の仕様（ノード保証、フェアシェアなど）は開発。	336 時間 ・以前のシステムでは月一回の定期保守を予定していて、2WK あれば完了するだろうと考え。それ以上は資源占有のリスクも鑑みて別途申請。 ・現在は、定期保守を３ヶ月に１回程度見込んでいる。	・申し込みには制限をかけていない。 ・各システムの 1 割前後のノード数で申し込まれることが多い。	同時投入はユーザあたり 2000。 滞留は 100 万。 ・スケジューラの応答が著しく悪化する場合は、手動で見直している。 ・シスシステのスケジューラではスケーラビリティが向上しているので、上記を暫定値として設定している。（以前のシステムは PBS)

筑波大学計算科学研究センター	a-1: HPC 系 主にシミュレーションだが、AI 系も最近増えている。 a-2:　OpenFORM ・全体の傾向としてノード数の多いジョブは OSS の傾向あり。 ・ISV もいくらかはある(が今回のシステムでは商用ライセンスを縮小している)。昔からの利用者は自作コードも所有している。 a-3: 概ね CPU-intensive だが、IO-intensive もある。 a-4: ・大規模もある。 ・1 ノードジョブを投入する利用者は多い。	スケジューラ NEC 製 JupyterHub 理由： (1)ベンダ提案(調達結果) ・調達仕様を満足するスケジューラをベンダ提案 ・大学独自の仕様もある。	24 時間 ・大規模なジョブ実行も保証(資源占有のリスクも鑑みて)するため。24H 以上はチェックポイントをとって実行。	120 ノード (物理ノード数まで) ・インタラクティブも 120 ノード。	同時投入はプロジェクトあたり 200 ジョブ (見直し予定) ・スケジューラの応用が著しく悪化する場合は、手動で見直している。
東京科学大学情報基盤センター	a-1: HPC 系、AI 系どちらもマジョリティ a-2: ISV はそれなりにいる。OSS は	スケジューラ Altair Grid Engine CLI 形式ではなく、GUI 形式（Web インタフェース）によるジョブ受付の仕組み	24 時間 ・スケジューリング性能やメンテナンス等を考慮	・140 ・最大並列は、上半期は全体の半分。下半期は全体の 4 分の 1 ま	投入ジョブ数は制限なし。 同時実行数は制限 30,32 などで制限。

	AlphaFold を含めればいる。自作コードはあまりいない。 コメント：プロセスアカウンティングをすれば具体的な割合をとれるかもしれない a-3: ・アプリの計算が多様化し（時間の 90%以上 AI、最後の数%だけ GPU で計算など）ている。ユーザもセンターも把握していない（できていない）。 a-4: ・シングルノードがそれなりに多い ・並列度 1 がマジョリティ	も用意（JupyterLab ベースで構築） 理由： (1)ベンダ提案 (調達結果) ・調達仕様を満足するスケジューラをベンダ提案 ・アルティアは燕の初期にある程度作り込みを行ったため継承しているがこだわりはない。	・予約ノードによって、長時間実行は可能	で可能。	・同時実行数制限は平日、土日で自動での構成変更を可能にしている。

名古屋大学情報基盤センター	a-1: Type-1 は　HPC 系 Type 2 は　AI 系 a-2: サブシステムごとに異なる a-3: Type1 では CPU-intensive がおおい。 Type2 は CPU-intensive と IO-intensive の両方ある a-4: TYPE 1 は並列度 TYPE 2 はアプリ依存	スケジューラ TYPE3 は会話利用ノードも設定 理由： (1)ベンダ提案 (調達結果)	・1 週間 ・シェアクラスは 2 週間 ・超並列は 24 時間 ・無制限は申し込みで可能にしている。 ・シェアクラスとは、1node/4GPG の内 1GPU を使う、1node/4 ソケットで 1 ソケットだけ使うクラス。	TYPE1 全ノード予約 2304 ノード ・TYPE1 常時稼働時の最大は 768 ノード ・TYPE2 常時稼働時の最大は 64 ノード　(ＡＩシングルが多い、並列対応のものは 49 オーバも存在するが少ない)	同時投入数 100. 同時実行数は制限.(50 などで制限) ・以前は同時投入数 300 を可能にしていたがジョブ数が多くなり絞ってきた。 ・同時実行数も負荷により制限を変えるが、通常は 50。
北海道大学情報基盤センター	a-1: HPC 系 a-2: 商用 ISV、OSS が多い。 ・自作コードは一定数あり。しかし、年々減っている印象。 ・計算化学（Gaussian)は多量。 ・OSS で導入した問い合わせが多い。	スケジューラ 理由： (1)ベンダ提案 (調達結果) コメント： カスタマイズがかなり入っている。 　- ノードの ON/OFF 　- 占有ノードが空いている時、短時ジ	240 時間 ・設定としては、24H,72H,240H ・占有ノードは 336H(14 日間)	256 ノード ・シェアノード:500、占有ノード:500 位に分ける ・シェアノードもいくつかのキューに分割す	同時投入は 128。 同時実行数は状況で設定 ・40~42,44 まで。

	a-3: CPU-intensive ・IO 関連の問い合わせはほとんどない。 a-4: ・シングル多数 ・HPCI ユーザは並列も多い。	ョブでシェアする機能など		るので 256 ノード並列が最大	
理化学研究所計算科学センター	a-1: HPC ・AI はまだ少ない。 a-2: ISV/OSS いずれも利用 a-3: いずれもあてはまる。さまざまなアプリ。 ・CPU かメモリで言えばメモリ。 a-4: いずれも当てはまる ・（参考値）ユーザの 61%は mpi 利用。ジョブ数 25%程度。 ・1 ノードジョブも多い。	スケジューラ経由がメイン ・富士通製ミドルウェア（TCS)	概ね以下の通り： ＊３８４ノード以下：７２時間 ＊３８５ノード以上：２４時間	・１２２８８が通常運用時の最大 ・１２２８９以上は毎月定期的に実行期間を設ける運用	同時滞留のみ：1000 ジョブ (バルクは 2000) 理由：スケジューラの性能限界
産業技術総合研究所	a-1: AI a-2: OSS 利用、自作コードが多い ・ISV 利用ユーザは存在。ただし、ISV	スケジューラ経由がメイン ・Altair Grid Engine (必要コア数を割り当ててノードを共有利	12–168 時間 予約では 30 日間 コメント：	512 並列 ・V100 1088 の半分（512) まで認めて	同時投入数は 1000 同時実行ジョブ数は 200

	は基本提供していない。 a-3: メモリが多い ・GPU インテンシブということでメモリ。 a-4: 基本 1 ノード以下が多い ・小粒な学習ジョブが多い（2022 年度統計では、90%程度だったが、生成 AI の LLM の登場前後で大規模ジョブが多くなるという変化がある)	用)	・ジョブの種類（バッチ、インタラクティブ）やノード共有（分割利用）、占有で変わるが、 -インタラクティブ:最大 12 時間 -バッチ:最大 168 時間(1WK)	いる (ジョブの要求ノード数と実行時間についてはノード時間積が同じになる枠組み。ノード数が多ければ実行時間が短くなる仕組み)	滞留ジョブ合計は、1000.
東京大学情報基盤センター	a-1: 両方ある ・GPU の有無で傾向が違う。 a-2: 自作コードが大半 ・大半がインハウスコード。 （OSS だと OpenFOAM 等の利用が見られるが、研究ミッションによって様々） a-3:	スケジューラ経由がメイン ・Miyabi は、Altair PBS Professional ・Miyabi 以外は富士通製ミドルウェア（TCS)	48 時間 ・システム共通 ・最大並列数の場合は、24H。他は 48H。 ・これまでの運用経験値から決定（1 回/月定期保守も考慮)	クラスタサイズの¼ ・最大並列ノード数は大体システムの 1/4(1/3~1/4)ぐらいのサイズとしている。	・運用状況を収集して適宜見直している（偏った投入の発生により他の利用者への影響がない程度に制限しているのが実情)

	a-4: CPU はノード優先 GPU はコア優先 ・GPU 搭載クラスタは、ノード分割利用（コア優先:GPU 数指定の利用）とノード単位利用が半々程度である。				
海洋研究開発機構	a-1: HPC ・GPU、CPU もあるため今後 AI は増えると考えている。 a-2: 自作コードが大半 ・地球科学関連の完全な OSS は意外と少ない。 a-3: メモリ ・メモリバンド幅インテンシブが多い a-4: 1 ノードが多い CPU はソケット単位 ・ジョブで見ると 1 ノード/ジョブは多い。（ベクトルは VE 単位）	スケジューラ経由がメイン EA は VMware ・EA は VMware V モーション等を有効にしている。	48 時間	最大利用量で制限 ・ベクトルはード数 x 実行時間（ノード時間積：リソースセット）で制限。制限を超えなければ利用可能。	制限はない ・割り当てられたリソース量を超えるとジョブ投入が拒否される。 ・同時投入、滞留については制限はないが、資源を占有しない設定をしている。 （同時実行数もグループで制限）

3.2.10.3.5.2　利用者からの計算要求を処理する仕組みについて

本ヒアリング項目では、利用者からの計算要求を処理する仕組みについてヒアリングを行う。この際、ほとんどのスーパーコンピューティングシステムで採用されているスケジューラやジョブ管理ステムという用語を用いることなく"利用者からの計算要求を処理する仕組み"という表現をしている。これは本サブワーキンググループのメンバおよびヒアリング対象の管理者の双方に対して、計算機資源の管理メカニズム＝スケジューラという固定観念を排除してヒアリングを行い、各HPCI構成機関での取り組みを正確にヒアリングしたかったことによる。本ヒアリング項目では、まず"利用者からの計算要求を処理する仕組み"について伺った後、なぜその仕組みを選択しているのかを伺う。もしその仕組みがスケジューラである場合は、アプリ開発者からの関心事項である最大ジョブ実行時間、最大並列ノード数、同時投入ジョブ数、同時滞留ジョブ数といったスケジューラパラメータについて質問した。

表 3.2.10.11第3列目にそのヒアリング調査結果を示す。"利用者からの計算要求を処理する仕組み"については今回調査したすべてのHPCI構成機関でスケジューラが採用されている。筑波大学計算科学研究センターでは、スケジューラに加えてJupyterHubを用いた仕組みを運用している。また、スケジューラを採用している理由としては、システム調達の結果ベンダからの提案に基づくスケジューラが導入されている状況がある。言い換えれば、どのHPCI構成機関としても調達時においては、特定のスケジューラに限定しているわけではないが、調達結果としてベンダの提案スケジューラが採用されている。

続いて、表 3.2.10.11の第4～6列目にスケジューラパラメータについての調査結果を記す。調査結果からジョブ最長時間については24時間から720時間と幅広い回答が得られた。しかし、必ずしもこれらの設定値は絶対ではなく柔軟に対応している構成機関もあることに注意されたい。長時間ジョブを許可している場合にメンテナンスや障害時の対応がとりにくくなるため、長いジョブ実行時間を設定できないという管理者の声もあった。また、公平性が保てなくなるという観点から、180時間以上を設定していないという声もあった。また、24時間以上のジョブはチェックポイントをとってもらって対応するのがよい、という意見もあった。ジョブ実行時間の設定については、幅広く相違がある。最大ジョブ並列数についての考え方も多様であった。京都大学学術情報メディアセンターでは、最大ジョブ並列数について制限を設けていないが、それ以外の構成機関では多様に設けていることがわかる。同時ジョブ数、同時ジョブ滞留数については、京都大学学術情報メディアセンター、東京科学大学情報基盤センターがずば抜けて大きな数字を設定、あるいは無制限としていることが判明した。これらの構成機関に共有している点は国産ベンダのスケジューラを採用していない点があげられる。

次に、図 3.2.10.8に示すヒアリング調査項目に対する回答を表 3.2.10.12に示す。表 3.2.10.12の第2列目にはインタラクティブ利用及びワークフローサポートに対する利用者からの希望や要求に対する回答をまとめた。今回調査したHPCI構成機関の全てでインタラクティブ利用を提供している。ワークフローサポートに対する要求は全体を通じてそれほど多くみられないのが実情である。しかし、ワークフローサポートに対して全く要望がないわけではなく、利用者が工夫してジョブを連携させているケースも見られるようである。資源機能に対するGUI利用に対する希望や要求については、九州大学情報基盤センターにおいて「Python利用で需要あり」との回答を除いて、「特にない」という回答が大多数となった。

次に、ワークロードの変化に対応してスケジューラ構成を手動で変更することがあるかという質問に対しての回答を第4列に示している。大阪大学D3センターでは大規模ジョブが投入された際にスケジューラの設定を変更する場合があると回答している。また、東北大学サイバーサイエンスセンターにおいては週単位で変更しているとの回答であった。東京科学大学情報基盤センターにおいてはスケジューラのパラメータを自動調整する仕組みをすでに導入しており、筑波大学計算科学研究センターでも検討を開始している。これらの回答から、ワークロード変化に対するスケジューラの構成や設定の変更については、運用の省力化やコスト削減の観点からも研究開発の余地があるように感じられる。また、第6列目に「計算要求を処理する仕組みについて、このようなものがあったらいいなと思う機能」についてのヒアリング調査結果を示しているが、筑波大学計算科学研究センターにおけるリアルタイム要求への対応、

東京科学大学情報基盤センターにおけるシステム外に存在するセンサの監視サーバとの連携といった要求がある点は、利用者からの要望を深掘りすることにより新たなシステムソフトウェアの必要性が見えてくるとも考えられる。

表 3.2.10.12 運用サービスについてのヒアリング結果 2

構成機関	c: インタラクティブ利用に対する利用者からの希望や要求 d: ワークフローサポートに対する利用者からの希望や要求	資源管理機能に対する GUI 利用に対する希望や要求	ワークロードの変化に対応して、手動でスケジューラの構成を変更することがあるか？	利用者からの計算要求を処理する仕組みについての満足点、不満点	このようなものがあったらいいなと思う機能
大阪大学 D3 センター	c: インタラクティブ提供済み (スケジューラのインタラクティブバッチ機能) Python 用フロントエンドも提供済み d: ジョブ連携は提供 (スケジューラの機能)	特になし。	大規模ジョブ投入時の交通整理程度	特になし	特になし
九州大学情報基盤センター	c: 利用希望あり ・可視化のためにデータダウンロードに	Python 利用で需要あり。	特になし。	対話型利用の Web 予約をスケジューラ IF に統一したい。	特になし。

	利用している例もある d: 希望、要求あり。 ・富岳のワークフローを試験的に導入			・利用者からのシステムのインタラクティブ利用 　インタラクティブと予約を組み合わせる。	
東北大学サイバーサイエンスセンター	c: 希望は最近よくある ・数値計算の研究者ではなくて、AI をやりたい利用者で要望が増えている。しかし、システムとしてはサポートできてない（余力がない） d: あまりない ・津波シミュレーションでは使用。 ・AI でもあまりない ・clone を使っ	特にない ・フロントエンドで X アプリは利用可能(スケジューラ機能のインタラクティブキュー)	設定変更を週で実施	・普通に動いていて当たり前だから満足。不満点はいろいろある。 (SLURM ではできるけどベンダ製ではできない機能がある。例えば、ワークフローも SLURM の方が複雑に表現できる。利用者に要望があるかは不明)	・ハードウェア障害の時に、被災したジョブの次スケジュールを特別扱いにするなどの機能。

	ているユーザがいる(サポートされている範囲)				
京都大学学術情報メディアセンター	c: インタラクティブは提供済みであり、使われている。要望は特にない。 ・伝統的にインタラクティブ利用可能な環境を提供している。 ・スケジューラのインタラクティブバッチを利用可能にしている。 d: 特にない。 ・スケジューラの機能等を使ってユーザが工夫して利用している様子。	特になし。 ・Open OnDemand はテスト予定を考えている。	特になし。	・新システムに移行してこれから評価	資源保障の仕組みは他機関利用者から評価されている ・その他 SDCV を使えるような要求として、アプリケーションションサーバの位置付けで少量のサーバを（GPU 搭載もあり）フロントエンドとして用意している

筑波大学計算科学研究センター	c: インタラクティブは提供済みであり、利用されている。（TYPE4（クラウド）では使われていない） d: 特にない。 ・アプリで用意されているものは使われているかもしれない。	特にない。 ・ジョブの実行状況を表示するコマンドを提供しているが要求は上がってない	フェアシェア的な考えを導入？　ジョブの負荷情報に合わせて、低優先度のキューの ON/OFF と通知の自動化を検討中(バジェットの再配分での不公平感の軽減)	現在特になし。 ・新システムなのでこれから評価	・リアルタイムの要求に応えるバッチで使うノードシェアをしながら、インタラクティブの即時性を実現したい（バッチの実行待ち、空きノードを割り当てるようなイメージ)
東京科学大学情報基盤センター	c: あると嬉しいというレベルの要望 ・基本的にはバッチで対応してもらうように支援 d: ・聞かれはするけどあまり積極的には対応していない ・ユーザ自身がシステムが	ノード予約(GUI) 空きノード情報などの参照運用関連（全て web)	・構成変更自体は自動化 ・制限変更の判断	・ジョブスループットの向上 ・実行時間制限やシステムの高負荷状態には問い合わせが多い。 ・技術的な理由でポイント反映が遅れることにもクレームが多い。	・センサ監視サーバの導入の要望がある。

	対応する範囲でやってもらう				
名古屋大学情報基盤センター	c: 提供済み ・デバッグ等で絶対必要なもの。 ・TYPE4(クラウド）ではあまり使われていない。 d：特にない。 ・ワークフロー制御という言葉自体をユーザが知らないので、便利さを知らない。そういうツールがあることも知らない。TYPE1 でシミュレーションしたあと機械学習する場合は、ジョブの連携がいるのだが、まだ技術的にできてないとい	web はあったらよいレベル。 JuypterNotebook 的なインターフェイスはよく利用されている。 ・以前は、ベンダ機能で提供されていたが、今はない。強い要望ではないですけど、あれば良いというレベル。	制限変更(同時実行数）の判断(人が判断) ・資源の少ないクラスタ(TYPE4)はよく変更する。	強い不満はない。 ・特殊な使い方（ゲノム系）には現在の資源要求量に不満がある模様。	・大容量の安価なファイルシステムの設置要望がある。

	う感じ。				
北海道大学情報基盤センター	c: 提供済み ・デバッグ等で絶対必要なもの。 ・TYPE4(クラウド）ではあまり使われていない。 d:　特にない。	特になし。 ・スクリプトの書き方は問い合わせがくるが、WEB で指定できないかなどの問い合わせはない。	特になし。	・待ち時間が長いというクレームが多い。	・投入ジョブに対して現在の状況で待ち時間をある程度正確に出してくれる機能があると良い。
理化学研究所計算科学センター	c: 提供している ・インタラクティブジョブはサポート済 d: あり ・WHEEL などをサポート中。	利用状況表示のみ 配賦された資源の利用状況を表示する GUI は整備済。資源追加などはサポートデスクのツール（ZENDESK)で運用	講習会など期間限定かつ特別な利用要請への対応時。運用改善のための変更もあり得る。	「計算要求を処理する仕組み」=スケジューラであれば、スループットの向上が望まれる。	スケジューラであれば、スループットの向上が望まれる。
産業技術総合研究所	c: 提供している ・インタラクティブジョブはサポート済(jupyiter 利用も）	あり ・ジョブスケジューラのアレイジョブ、ジョブ連携機能を利用され	・電力削減等で稼働ノード数を手動制御しことはあるが、常時発生する状況ではない。	・リソースの割り当て数制御に関するスケジューラパラメータのチューニングのし易さ。	・オンプレとシームレスに使えるクラウドバースティングを可能にする資源管理があっても良

	d:あり ・ジョブスケジューラのアレイジョブ、ジョブ連携機能を利用されている。 ・上記以上の要望は特にない。	ている。 ・上記以上の要望は特にない。		・投入して実行されない、予約もいつ取れるかわからないというような状況に対する改善。	いかと考える。且つ、個々のジョブからプログラマブルなインターフェイスで資源要求し、利用できる資源管理の提供を望む。
東京大学情報基盤センター	c: 提供している ・サービスレベルとしては、要求ベースで対応（GPU のインタラクティブ利用の長時間化の要求等）。 d: あり ・多量の小ジョブを投入したいという要望等がよくある。ツール(Xcrypt 等)紹介等をしている。	特にない ・要望としては、各利用者のトークン量を確認するというレベル（Web で公開）	あり ・以前のシステムでメモリモデル（フラットまたは、キャッシュモード）によってクラスタを分けたことがある。利用に偏りが大きく手動で構成ノード数を変更していた。	・「100%に近い混雑になると全然流れない」などの声はあるが、それ以上は特にない。 ・スケジューラの変更についても変更手順を公開しているので、それほど問題が起きるとは考えていない。	・消費電力量を収集できるように仕様を出していたが、現在ではほぼ実現されている。 ・他には、GPU のマルチインスタンスを動的に可能にする仕掛けを要求している。

海洋研究開発機構	c: 提供している ・データのダウンロードや jyupyter での利用など使い方の変化があり、長時間の利用要望がある。 d: あり ・現在の利用では難しい要求はない。 ・アンサンブルの要求は強い。 ・利用者が使いこなせていない面もある。	特にない ・スケジューリング状況の見えるかは対応しているが、更にわかりやすさの改善要求はある。（例えば、グループの中での利用状況など）	あり ・ただし、半年に一度程度。	・同時実行数の調整（制限の緩和）程度。	・利用状況、混雑度に合わせた制限の緩和が自動的に行われると良い。 ・アンサンブルや CG ソフトの多量の画面作成の状況が把握し易いものがあると良い。 ・利用状況と電気使用料とか消費電力量の状況がリンクする統計情報の収集があると良い。

3.2.10.3.6 HPC センターの現状について：クラウド利用環境

本ヒアリング項目では、図 3.2.10.9に示すスライドを用いて今回調査対象としたHPCI構成機関におけるクラウド利用環境に関する取り組みを調査した。具体的には、以下の4点についてヒアリングを実施した。

- 何らかの用途で民間クラウドベンダの提供するサービスを導入しているかどうか
- 導入している場合は、クラウドの何に期待して導入したか？何が期待外れだったか？導入していない場合は、今後の計画やクラウドの何が困難・課題であるか？
- 利用者からの何らかのクラウド利用に対する要望
- クラウド利用を推進する場合に、このようなシステムソフトウェアがあるといいと考えるアイデアは？

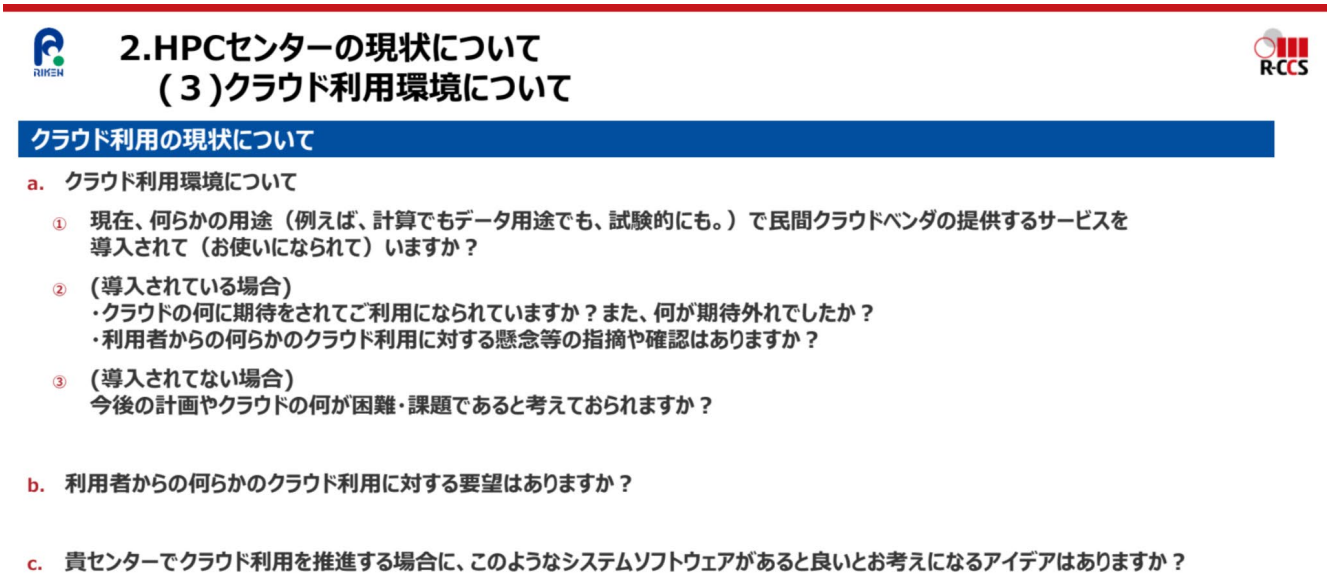


図 3.2.10.9　クラウド利用環境についてのヒアリング項目

ヒアリング結果は表 3.2.10.13に記す。「何らかの用途で民間クラウドベンダの提供するサービスを導入しているかどうか」をヒアリングした結果は、表 3.2.10.13の列2にまとめた。また、導入していると回答した場合にヒアリングした「クラウドの何に期待しているか？」、「何が期待外れだったか？」と、導入していないと回答した場合にヒアリングした「今後の計画やクラウドの何が困難・課題であるか？」の結果は表 3.2.10.13の列3にまとめた。さらに、利用者からの何らかのクラウド利用に対する要望は表 3.2.10.13の列4にまとめた。最後に、「クラウド利用を推進する場合に、このようなシステムソフトウェアがあるといいと考えるアイデアは？」は表 3.2.10.13の列5にまとめた。

表 3.2.10.13　クラウド利用環境についてのヒアリング結果

構成機関	何らかの用途で民間クラウドベンダの提供するサービスを導入しているかどうか	導入している場合：クラウドの何に期待しているか？何が期待外れだったか？ 導入してない場合:今後の計画やクラウドの何が困難・課題であるか？	利用者からの何らかのクラウド利用に対する要望	クラウド利用を推進する場合に、このようなシステムソフトウェアがあるといいと考えるアイデアは？
大阪大学 D3センター	OCI、Azure（本格運用はまだ）	クラウドバースティング ・利用者還元策としての計算資源	特になし。	利用したクラウド計算資源にかかった料金を利用者ごとに記録するツールが欲しい。

		課題： ・クラウド計算資源の高い利用料金が運用におけるネックとなる。 ・産業利用等において民間クラウドの利用がセキュリティ上問題ないかの問い合わせ対応に苦慮することが予想される		
九州大学情報基盤センター	Rescale ScaleX ・利用されていない（試験的な導入）	クラウドバースティング 課題： ・利用料金が高い。 ・ビジネスモデル（制度）問題がある。クラウド利用契約を大学事務として仲介業（利用者ー大学ー業者）ができない。	・Cloud native なワークロードをスパコンで使いたいという利用要望あり（オンプレミスバースティング）。	AWS、Microsoft 等のクラウドからスパコンへジョブを投げる理研の研究に期待。
東北大学サイバーサイエンスセンター	特になし。 ・前システムで導入していたクラウド利用環境は終結した。 ・AOBA-S が使われリソース不足になる場合、外部の利用を検討する。	クラウドバースティング 課題： ・データのロケーションを意識せずに使えると嬉しい。	ナノテラス関係で、クラウドストレージとの連携などでの問い合わせはある。（S3 対応ゲートウェイを導入予定）	オンプレミス・クラウド間でのデータ移動の手間をなくすためのステージング機能があるとよい。

京都大学学術情報メディアセンター	OCI（ヒアリング時は試験的に 1 ノードのみを導入） クラウドストレージとの連携（s3fs 等）	クラウドバースティング ・利用者還元策としての計算資源 ・調達が遅れた場合の予備的計算資源 課題： ・クラウドバースティングに関して、利用に特別な負担金は求めていないため試験運用を通じて費用バランスを検討。	クラウドストレージとの連携や S3 対応の要望あり。	ストレージ I/O が性能のボトルネックにならないように隠匿する仕組みがあるとよい。
筑波大学計算科学研究センター	特になし。	NextCloud：センター内、HPCI 共用ストレージのデータ共有に提供。 課題： 環境相違への対応 ・アプリのコンパイルし直し等が発生するケースがあり負担大。 ・異なるシステムであることによる運用負担が懸念。	特にない。 ・クラウドストレージ連携機能用途も検討していく。現状はオンプレストレージをマウントしているがクラウドストレージ連携も検討必要と考えている。	特になし。
東京科学大学情報基盤センター	特になし。	情勢に応じた対応は必要な一方、現在は選択肢として考えていない。	特になし。 ・企業利用でスパコンへのアクセス	・経済的な困難さの解決が重要（クラウドバースティングが念頭にあ

		課題： ・利用制度のモデル検討 ・センターで導入したときに利用料金的に（運用経費ベースのクラウド利用料と比べて高額になる）経済的な点が成り立たないと考えている	制限要求（会社からのみアクセス可能にしたい。自宅からさせたくないなどプロプラではできない制限）をクラウドでは対応可能かもしれない。	り）
名古屋大学情報基盤センター	特になし。 ・計画はしているが予算都合（クラウドは料金が高くなる）。	次の計算機システムでは、クラウドストレージとの連携のために S3 対応を検討予定。 課題： ・計算資源としての利用料金が高い。	クラウドバースティング的な利用の要望があった。	・経済的な困難さの解決が先決（クラウドバースティングが念頭にあり）
北海道大学情報基盤センター	特になし。 ・オンプレで提供。クラウドで 4 つ（仮想、物理、GPU,ストレージ）のサブシステムが存在する。 ・ルート権限も渡しているのでセキュリティ課題がある。	利用の予定なし。	要望はある。	・セキュリティ関係
理化学研究所	・導入している	データ連携、互換	コストキャッピン	

計算科学センター	(Oracle, AWS(整備中))	CPU での動作検証環境 課題： コスト	グ	
産業技術総合研究所	・導入している コメント： ・ユーザ向けには導入していない。運用支援の観点で、例えば DNS などを AWS で利用。 ・クラウドサービスのバックエンドとして ABCI を使っていただくことは可能。	クラウドへの期待面： DNS など運用支援の観点 課題： クラウドのリソースをオンデマンドで確実に確保できないのが現状としてある。サービスの安定供給という面では難しいと感じている。	2024 度に限っては、ABCI のシステム更新により、AWS 等へのバースティング要望はある。	・クラウドプロバイダーとの契約はシステムの課金の枠組みが課題としてある。利用者権限でバースティングを可能にする仕組みが望ましいと考えている。 ・クラウド利用ではデータ共有が課題と感じており、データ共有のための仕組みが欲しい。
東京大学情報基盤センター	特になし ・トライしたが、現在ではやっていない。 ・運用関係ではやっていないが、今後可能性はあると考えている。	運用（管理）関係では検討する可能性があるが、計算資源の調達やストレージ利用としてはとして考えていない。	あることはあるが、要望を確認すると mdx で解決可能なことがほとんど。	
海洋研究開発機構	特になし	検討継続中（前回調達では導入も検討）。 当センターではデータストレージが	利用者は研究成果の公開という面で利用してみたいと思っている様子だが、データが大規	コストが最大のネックだが、経済安全保障の関係でセキュリティ周りの要件が高い。求め

		大きく（データ保存ニーズが高い）、クラウド利用ではデータ転送等に費用がかかりすぎコスト面で導入断念。	模ということを認識しているため断念している様子。	られる要求を満たす、認証などのセキュリティ体系が具備されることが必要。

「何らかの用途で民間クラウドベンダの提供するサービスを導入しているかどうか」のヒアリング結果としては、現在は提供を終了しているものも含めて4つのHPCI構成機関においてクラウド利用環境の導入実績があった。さらに、これら4つのHPCI構成機関におけるクラウド環境の利用用途としては共通して、オンプレミスの計算需要を民間クラウドベンダの提供する計算環境にオフロードするクラウドバースティングを意図していた。これは、HPCI構成機関の計算機システムの管理者らの間において、結果の良し悪しはともかく、クラウド利用環境に対する注目度が大きいことを示していると考えられる。

しかし、導入実績のあるなし関係なく、ほとんどのHPCI構成機関において、クラウド計算資源の利用料金の高さがクラウドバースティングの導入における課題であるという回答も得ている。基本的に多くのHPCI構成機関では計算によって消費される電気代を基準に利用料金を定めているため、オンプレミスの計算資源の利用料金はクラウドの計算資源と比べると非常に安くなる。この利用料金の差が各HPCI構成機関の運用において扱いづらいことが、クラウドバースティングを導入するにあたっての障害となっている。実際、クラウドバースティングの経済的困難さの解決が先決であるとのコメントをいくつかのHPCI構成機関からいただいている。

また、東北大学サイバーサイエンスセンターや京都大学学術情報メディアセンター、東京科学大学情報基盤センターからは、クラウドバースティングに関して、オンプレミスに用意してある共有ストレージとクラウドの計算環境の間でのデータ移動も課題としてあげられている。計算性能を重視する場合、オンプレミスとクラウドの距離がもたらす通信遅延の影響を最小限にするために、計算に必要となるデータはクラウド側のストレージに配置する構成が採用される。しかし、この構成では、計算を始める前に計算に必要なデータをオンプレミスの共有ストレージからクラウド側のストレージに移動させ、計算が終わった後に計算で出力されたデータをクラウド側のストレージからオンプレミスの共有ストレージに移動させるという手間が必要となる。利用者にとって、この手間はクラウドバースティングの利便性を下げる要因となる.

クラウドバースティング以外のクラウド利用環境の利用実績としては、京都大学学術情報メディアセンターにおいて、クラウドストレージと連携するための仕組みとしてs3fs等を導入していることが確認できている。また、東北大学サイバーサイエンスセンターからは、利用者からの要望に応じて現在クラウドストレージとの連携のためにS3対応の環境を用意しているところであるとの情報をいただいた（2023年9月時点）。さらに、名古屋大学情報基盤センターからは、今後の計画として、クラウドストレージとの連携を念頭にS3に対応した環境の導入を検討しているという回答が得られている。このことから、HPCI構成機関の計算機システムの管理者らはクラウドストレージとの連携も重要な取り組みと捉えていると考えられる。

3.2.10.3.7 HPC センターの現状について：仮想計算機技術

本ヒアリング項目では、図 3.2.10.10に示すスライドを用いて調査を行った。

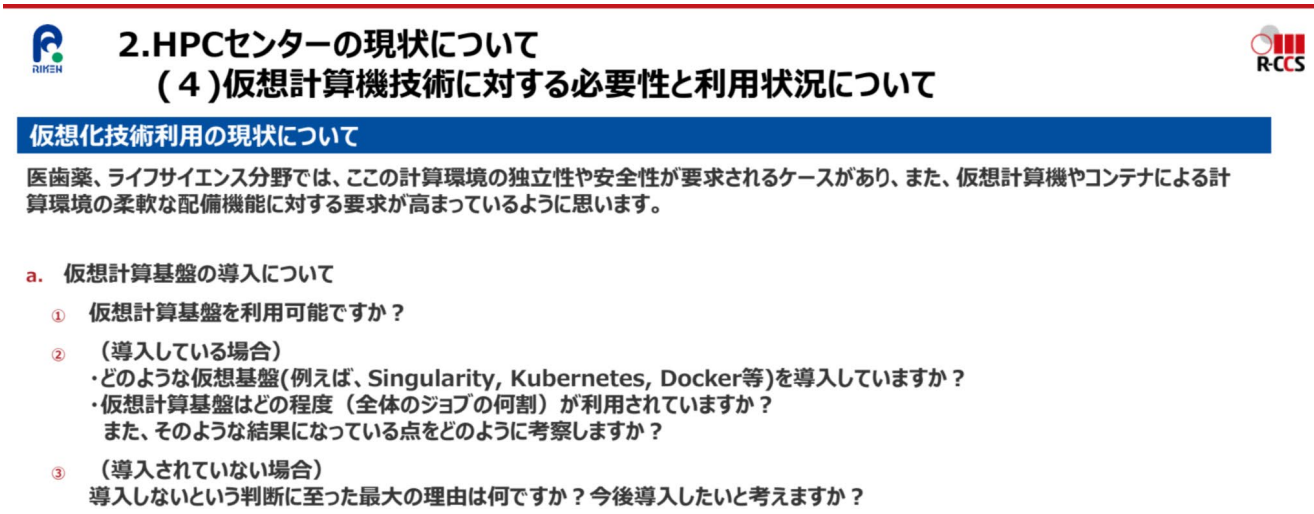


図 3.2.10.10　仮想計算機技術についてのヒアリング項目

表 3.2.10.14に図 3.2.10.10で示したヒアリング項目に対する調査結果をまとめた。第2列目に仮想計算基盤の導入状況に関する調査結果をまとめた。今回調査したHPCI構成機関ほとんどすべての構成機関でSingularityを仮想計算基盤として採用・導入していることが判明した。続いて、導入している仮想計算基盤の利用実態についての調査結果を第3列目、導入されていない場合における導入していない理由および今後導入したいかどうかに関する調査結果を第4列目にまとめた。導入している多くの構成機関では、その利用実績について把握できていないことがわかる。回答が得られた大阪大学D3センターにおいても、ジョブ数を基準として約1.5%程度であった。肌感覚程度ではあるということであるが、東京科学大学情報基盤センターにおいても、一桁パーセント貞であろうとの回答であった。導入していないHPCI構成機関において、仮想計算基盤を導入していない理由としては、現行システムの導入時に設計に入っていなかったということであり、次期システムでは仮想計算基盤を導入する予定ということであった。これらの結果から、今日のスーパーコンピューティングシステムに対しては、利用実態はいまだあきらかではないものの、管理者視点では必要不可欠と考えていることが明らかである。

表 3.2.10.14　仮想計算機技術についてのヒアリング結果

構成機関	仮想計算基盤を利用できるか？ どのような仮想基盤を導入しているか？	(導入している場合) 仮想計算基盤はどの程度（全体のジョブの何割）が利用されているか？ また、そのような結果になっている点をどのように考察するか？	(導入されてない場合) 導入しないという判断に至った最大の理由は何か？ 今後導入したいと考えるか？
大阪大学 D3センター	YES Singularity	1.5%程度 (ジョブ数)	N/A

九州大学情報基盤センター	YES Singularity	不明 ・利用実態は把握できてない。	N/A
東北大学サイバーサイエンスセンター	YES Singularity (AOBA-B)	不明 ・利用実態は把握できてない。	N/A
京都大学学術情報メディアセンター	YES Singularity	不明 ・利用実態は把握できてない。	N/A
筑波大学計算科学研究センター	YES Singularity	不明 ・利用実態は把握できていないが、使用頻度が高いとも思えない（問い合わせもない）。環境により IB が使えるか、センターの MPI を利用などがコンテナの再作成などの問題があり利用限定的になっていると思われる。	N/A
東京科学大学情報基盤センター	YES Singularity	不明 ・感覚的には一桁%	N/A
名古屋大学情報基盤センター	YES Singularity	利用実態は把握できていないが、使用頻度が高いとも思えない（問い合わせもない）。環境により IB が使えるか、センターの MPI を利用などがコンテナの再作成などの問題があり利用限定的になっていると思われる。	N/A
北海道大学情報基盤センター	No	N/A	・現行システム設計時は対象に入っていなかった。また、これまで強い要望がなかった。 ・次期では GPU も仮想化も導

			入する予定。
理化学研究所計算科学センター	YES Singularity, Docker	Singularity（利用者極わずか）、Docker（利用者なし）	N/A
産業技術総合研究所	YES ・Singularity のみ。 ・Docker は一時導入したが、現在は制限。	・現在統計をとっていないので不明だが多くはない。Python 仮想環境で利用されている方は多い印象。	N/A
東京大学情報基盤センター	Yes ・全システムで Singularity は利用可能。	・Singularity/Enroot ・Singularity(Apptainer)は全システムで利用可能。利用も割と多い印象（月平均モジュールロード回数：790 回程度）	N/A
海洋研究開発機構	Yes ・Singularity	把握できていない。	N/A

3.2.10.3.8 HPC センターの現状について：データストレージ、データ基盤

本ヒアリング項目は、図 3.2.10.11、図 3.2.10.12に示すスライドを用いてヒアリングを行った。

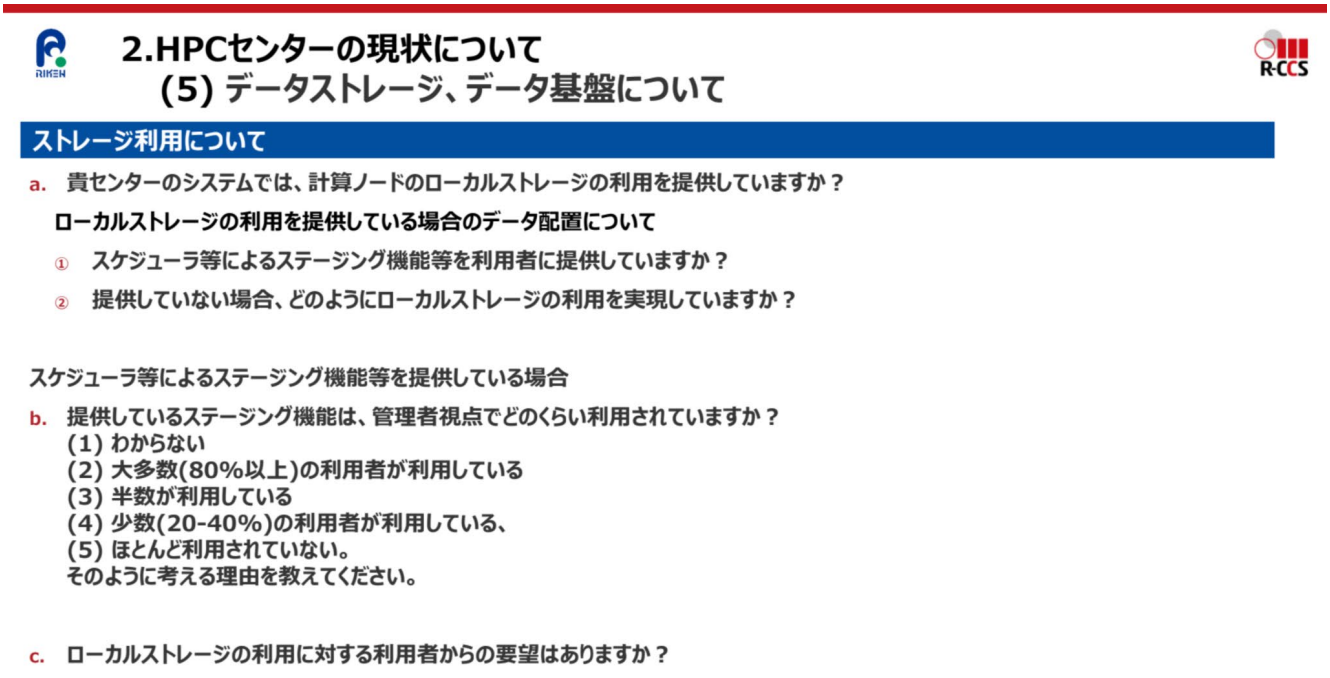


図 3.2.10.11 データストレージ、データ基盤についてのヒアリング項目 1

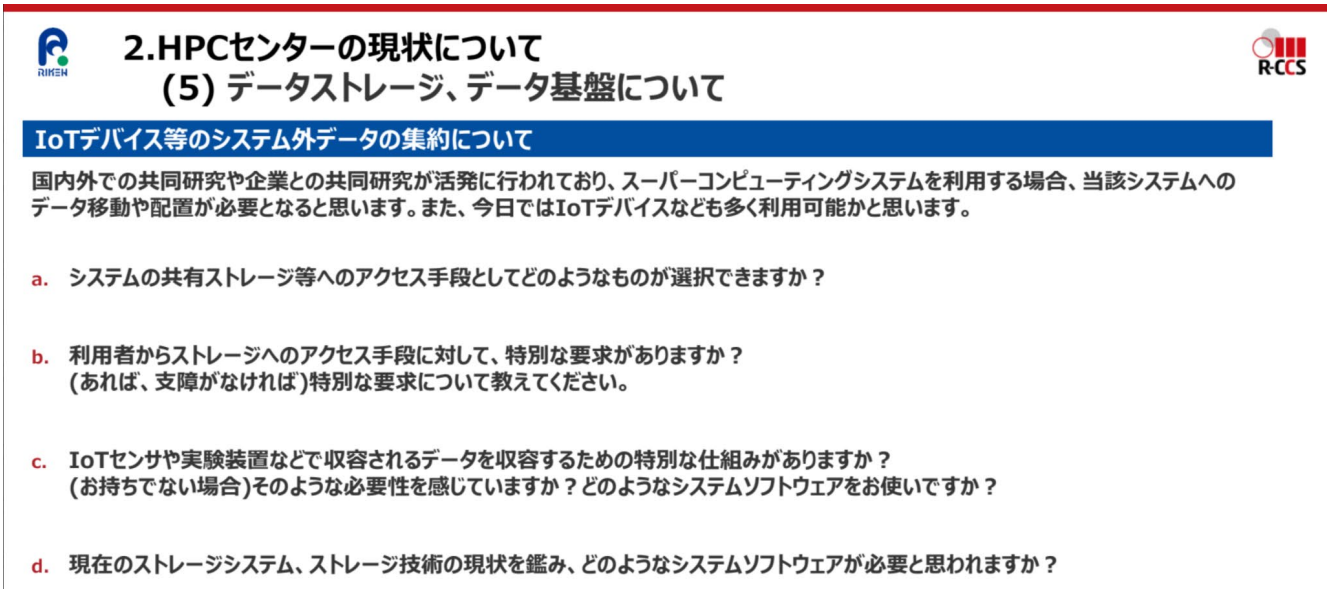


図 3.2.10.12 データストレージ、データ基盤についてのヒアリング項目 2

3.2.10.3.8.1 ステージング機能の導入状況・実態、ローカルストレージの利用方法、ローカルストレージに対する利用者からの希望や要求

表 3.2.10.15に図 3.2.10.11に示すヒアリング項目に対する回答をまとめた。第2列目にスケジューラ等によるステージング機能等を利用者に提供しているかどうか、第3列目にスケジューラ以外のローカルストレージの利用方法があるかどうか、第4列目にはステージング機能の利用実態に対する回答をまとめている。この結果から、今回調査したHPCI構成機関全てでステージング機能を提供していないことが判明した。しかし、実態としては、計算ノード内にローカルストレージを用意し、利用可能としていることがわかる。第3列目の調査結果からは、利用者がローカルストレージを利用する方法は利用者自身がcp等を用いてジョブ内でコピーすることで十分であると考える管理者視点がみてとれる。

次に、第5列目に今日利用者からのローカルストレージの利用希望があるかを質問した。この結果として、利用者はすでにジョブ内からデータをローカルストレージに移動させることができることからか、利用者から利用希望の声がそれ以上上がっていないという意味で「特になし」とほとんどのHPCI構成機関から回答が得られている。大阪大学D3センターではローカルストレージの利用はオプション機能の位置付けとしており利用を推奨していないのに対し、京都大学学術情報メディアセンターではIOが厳しいジョブの出力先としてローカルストレージの利用をアドバイスすることがあると利用をすすめることがあり対照的でもある。

表 3.2.10.15 データストレージ、データ基盤についてのヒアリング結果 1

構成機関	スケジューラ等によるステージング機能等を利用者に提供しているか？	スケジューラ以外のローカルストレージの利用方法は？	ステージング機能の利用実態	ローカルストレージの利用希望
大阪大学 D3 センター	提供していない。 ・ローカルストレージは提供 スケジューラに付随するステージング機能および、セキュアなステージング機能をクラウド利用時も含めて研究中。	ユーザがジョブ内でコピー（cp 等） オプション機能の位置付けであり、利用を推奨していない。	左記の通り。	特になし。
九州大学情報基盤センター	提供していない ・ローカルストレージは提供（/tmp 等で利用可能） ・ステージングのメリットがない（read once write once の利用）。 gaussian がスクラッチファイルをよく見かける。	ジョブ内で利用	左記の通り。	特になし。

東北大学サイバーサイエンスセンター	提供していない ・ローカルストレージは提供（スクラッチファイル用）	ユーザがジョブ内でコピー（cp 等）	左記の通り。	特になし。
京都大学学術情報メディアセンター	提供していない ・ローカルストレージは提供	ジョブ内で利用	左記の通り。	特になし。 ・IO が厳しいジョブの出力先変更先等をアドバイスすることがある程度。
筑波大学計算科学研究センター	提供していない ・ローカルストレージは提供 将来的にキャッシング並列ファイルシステムを提供する際に、ステージング機能も提供する見込み	ジョブ内で利用（SCP 等）	左記の通り。	特になし。
東京科学大学情報基盤センター	提供していない ・ローカルストレージは提供	ジョブ内で利用	左記の通り。	あまりない。 ・機械学習のデータセットとかそういったものをパーシステントに欲しいみたいな話はあったが、難しいため断った
名古屋大学情報基盤センター	提供していない ・ローカルストレージは提供	ジョブ内で利用	左記の通り。	特になし。

北海道大学情報基盤センター	提供していない ・ローカルストレージは提供	ジョブ内で利用	左記の通り。	特になし。 ・要望はないが、使い方の問い合わせがあれば説明。
理化学研究所計算科学センター	提供していない ・階層型のファイルシステムを採用	なし ・ステージング機能はある。公開していない。 ・階層型ファイルは利用必須（主に階層を見せない運用で利用	左記の通り。	利用者によるが、性能・割当容量の増加要望あり
産業技術総合研究所	提供していない ・ステージングは利用者サイドで自ら実施。 （手順書を提供）	なし	なし	特になし コメント： ・大規模言語モデルのチェックポイント目的 ・小規模ジョブが多いので利用頻度は高くないと思われる。他には性能のアイソレーションをしたいというケースもある。
東京大学情報基盤センター	現行システムでは、ローカルはない。 各システムのストレージ(並列ファイル)利用。	以前は、BeeOND（BeeGFS On-Demand）も入れていたが、利用頻度は不明。	不明	特になし ・メモリが十分に実装されているわけではないので、LLM 的なことを実行するケースを想

	・以前のシステムでは「提供」＋BeeOND（BeeGFS On-Demand）も入れていた。 ・次期システム(miyabi)にはローカルストレージを導入予定で検討中。			定してノード内に高速 SSD を準備した。
海洋研究開発機構	GPU ノードだけローカルストレージをつけている。 ・利用者は少数	なし	なし ・利用者が手動で利用	正確に把握していないが少数 ・GPU ノードだけ 16TB のローカルストレージをつけている。

3.2.10.3.8.2　システム外データの集約：システムの共有ストレージ等へのアクセス手段

表 3.2.10.16に図 3.2.10.12に示すヒアリング項目に対する各HPCI構成機関からの回答をまとめた。ネットワークやIoT技術の発展により、スーパーコンピューティングシステム外で発生するデータを高効率に解析したいという要求や要望を生み出しつつあるように本サブワーキンググループでは考える。実際、データ同化シミュレーションやリアルタイム処理、緊急時処理などではシステム外のデータを利活用することを前提としているケースがある。本ヒアリング項目では、今日システム外で発生するデータを利活用しようとする利用者に対して、各HPCI構成機関がどのように対応しようとしているかを調査した。第2列目にシステムの共有ストレージ等へのアクセス手段としてどのようなものが選択できるかに対する各HPCI構成機関からの回答をまとめた。結果からは、どのHPCI構成機関はSSHベースのファイル転送の仕組みを提供している。加えて、S3プロトコル、NextCloudの提供によって、スーパーコンピューティングシステムのストレージ等へのアクセス手段を提供しようとする動きが見える。

第３列目には、利用者からストレージへのアクセス手段に対する特別な要求があるかどうかを質問したヒアリング結果を示した。この結果からは、京都大学学術情報メディアセンター、東北大学サイバーサイエンスセンターからはネットワークで移動させることができないデータをスーパーコンピューティングシステムで処理したいという要望があることがわかる。京都大学学術情報メディアセンターでは、利用者からHDDの持ち込み利用をしたいという要望がある。また、大阪大学D3センターでは、この回答はシステムセキュリティのヒアリング項目とも関係するが、ジョブ実行中に外部URLよりデータをダウンロードし、データ処理するオープンソースソフトウェアについて利用者からの希望があり、特別に特定ポートの開放を許可する処置をしている事例があることを伺えた。

第4列目にはIoTセンサや実験装置などで生成・収集されるデータをスーパーコンピューティングシステムのストレージに収容する特別な仕組みがあるかに対する回答をまとめた。京都大学学術情報メディアセンターおよび筑波大学

計算科学研究センターでは、このような要求に対して、仮想サーバホスティングを提供しているとの回答があった。また、東北大学サイバーサイエンスセンターでは、ハウジングで対応予定との回答であった。大阪大学D3センターでも、IoTセンサや実験装置から直接収容する仕組みはないが、実験装置やIoTセンサのデータを研究室で集約されたデータを収容するサービスを、オブジェクトストレージを通じて提供している。

第5列目に「ストレージシステム、ストレージ技術の現状を鑑み、どのようなシステムソフトウェアが必要と思うか？」についての結果をまとめた。その結果、データ暗号化やデータ保存に関連したソフトウェアの開発の必要性、大規模データ移動に資する高速データ転送ソフトウェアや転送状況モニタリングソフトウェア、階層型ファイルシステム、キャッシングファイルシステムなどの回答が得られた。

表 3.2.10.16 データストレージ、データ基盤についてのヒアリング結果 2

構成機関	システムの共有ストレージ等へのアクセス手段としてどのようなものが選択できますか ？	利用者からストレージへのアクセス手段に対して、特別な要求がありますか ？	IoT センサや実験装置などで収容されるデータを収容するための特別な仕組みがありますか?	ストレージシステム、ストレージ技術の現状を鑑み、どのようなシステムソフトウェアが必要と思われますか ？
大阪大学 D3 センター	SSH NextCloud (オンラインストレージ) オブジェクトストレージ S3 (Cloudian)	なし ・Job 内で外部 URL よりデータダウンロードし、データ処理する OSS を利用しするものがあり、特定ポートの開放を特別申請（審査あり）処理。	なし ・IoT センサや実験装置から直接収集する手段の提供はないが、実験装置のデータを研究室で集約したものを（研究室サーバから）集約するサービスは提供（マルチテナントのオブジェクトストレージ・サービス）	高速データ転送ソフト 転送状況モニタリング 階層型ファイルシステム (HFS) ・大規模・大容量のデータを高速に集約に集約する高速転送ソフトの導入が必要（+データ転送の負荷状況のモニタリング）。 ・HFS（バックアップ、共有ストレージ同期）

九州大学情報基盤センター	・ローカルストレージは、/tmp 等で利用可能（HDD、SSD） ・ステージングのメリットがない（read once write once の利用）。gaussian がスクラッチファイルを利用するのをよく見かける。	特になし	特になし	なし
東北大学サイバーサイエンスセンター	S3 ・DDN　S3DS 導入中	・ネットワークで up/down load できない大学外からのデータを処理したい要望あり。	なし ・IoT 機器等に対する特殊な要望については、ハウジングで対応予定。 (専用線)	特になし。
京都大学学術情報メディアセンター	SSH(SCP/SFTP) HpFP	SSHFS 今後 S3 を予定 ・ネットワークの細い利用者からは HDD の持ち込みを相談されることがある。	仮想サーバホスティング、SSHFS を提供	データ暗号化 データ保存(ユーザ任意の Back UP)
筑波大学計算科学研究センター	NextCloud(WebDav) SSH (SCP/sftp)	特になし。	仮想サーバホスティングを提供 SSHFS	キャッシングファイルシステム
東京科学大学情報基盤センター	SSH(SCP) RSYNC	特になし。	特になし。 インターネットには自由にアクセスできるので、セン	特になし その都度、nextcloud 的なものを使えるように

			ターから外にアクセスする分には制限はない	しておいて、センサーデータだとかをそれ経由でというのが考えられるが、今言われて気付いたというぐらいにはあんまり検討してない。
名古屋大学情報基盤センター	SSH(SCP, SFTP など)	特になし	特になし。	特になし。
北海道大学情報基盤センター	SSH(SCP, SFTP など)	特になし ・保持しているデータ量が多いユーザが少ない。	特になし	・Backup やシステム更新時のデータ移行の容易化
理化学研究所計算科学センター	HPCI 共用ストレージとの交換 クライド API（S3 など）	特になし ・世の中の技術動向によりセンター側で提供サービスを見直す	・センター内では消費電力計測の仕組みを構築 ・HPCI ストレージ経由での連携例はある。	多数ファイルの一括操作への耐性を有するファイルシステム
産業技術総合研究所	SSH 経由(scp/sftp など) S3 互換の API 経由	なし	・企業ユーザからは物理的に分離した領域の提供を求められることがある。	データアクセスに対する QoS 保証を求められる
東京大学情報基盤センター	SSH 経由	なし ・VPN を貼ってやりとりしたいという要望はある。(利用者のストレージからデータを安全	特になし ・MDX を利用すると可能。	大規模ストレージを導入して用途毎に使い分けできるよう考えたい。

		に転送するチャネルが欲しいという要望)		
海洋研究開発機構	SSH 経由	なし ・所外の方では、接続スピードの改善を相談されている。	・観測機器で撮ったデータをテープで持ってるのを持ってきて転送している利用者はいる。	データが大容量になって移行も難しくなってきている。新しい技術によるソリューションが必要。S3 の導入も検討中。

3.2.10.3.9 HPC センターの現状について：システムセキュリティ、モニタリングとユーザ利便性

本ヒアリング項目では、図 3.2.10.13に示すスライドを用いて調査を行った。スーパーコンピューティングセンターを保有する組織では、システムへの不正アクセス、データ漏洩などのセキュリティインシデントが発生した場合の影響が大きいため、システムセキュリティに対して多大な労力を行っていると予想される。また、システムの挙動をモニタリングすることは、計算システムの性能に影響を与えるというトレードオフの問題もありうる。また、頑強なセキュリティ対策は、ユーザの利便性に影響を与える。そのような観点から、本ヒアリング項目ではHPCI構成機関のシステムセキュリティ、モニタリングについての調査を行った。具体的には、以下の4項目のヒアリングを実施した。

- 計算ノードから外部への直接アクセス、外部から計算ノードへの直接アクセスを許可していますか？
- 許可あるいは禁止している場合、どのようなアクセス制御の方針で運用されていますか？
- 利用者から計算ノード内外へのアクセスに対する要望はありますか？要望への対応アイデアがありますか？
- 計算ノード内外へのアクセス制御をするにあたって、どのようなシステムソフトウェアをお使いになられていますか？

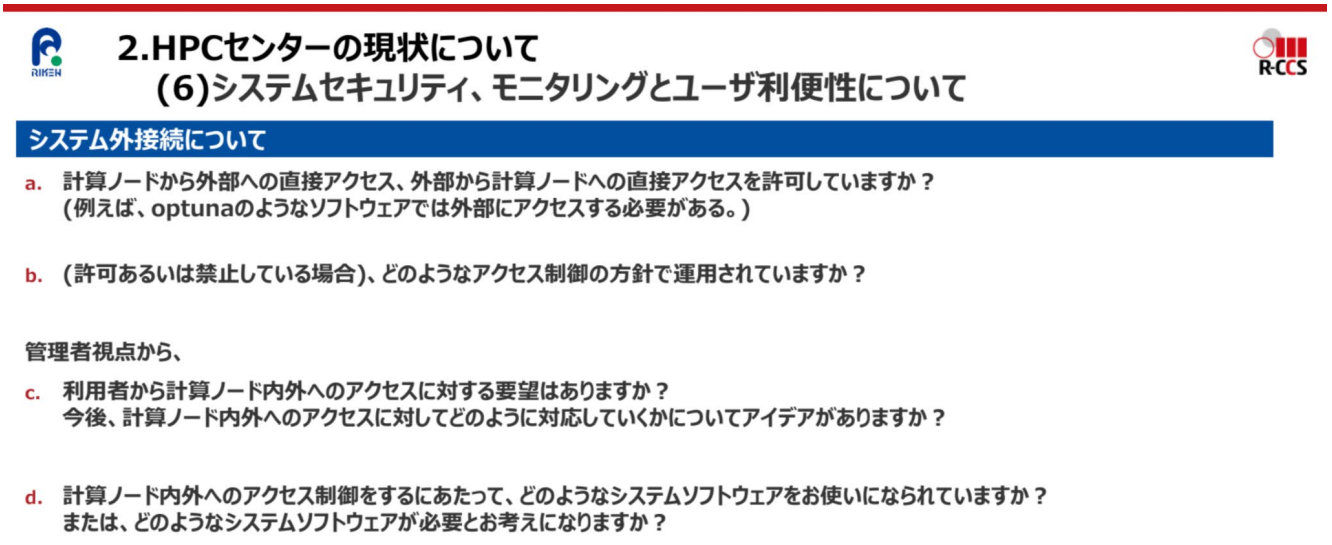


図 3.2.10.13 システムセキュリティ、モニタリングとユーザ利便性についてのヒアリング項目

表 3.2.10.17に図 3.2.10.13に示すヒアリング項目に対する回答をまとめた。第2列目に計算ノードから外部および外部から計算ノードへのアクセス可否についての結果をまとめている。今回調査対象としたHPCI構成機関の管理者からの回答では、おおむねほとんど（東北大学サイバーサイエンスセンターを除く）が計算機内部からの外部へのアクセスを許可し、外部から内部へのアクセスは禁止している。ただし、内部から外部へのアクセスについても、条件付きとしている場合と無制限の場合があることが判明した。続いて、第3列目にどのようなアクセス制御の方針で運用されているかについての回答をまとめている。基本的なアクセス制御方針はNATに基づく構成機関がおおくみられる。東北大学サイバーサイエンスセンターは外部へのアクセスはフロントエンドサーバからのみであり、この計算ノード向けあるいは計算ノードからのアクセスに対しては最も厳しい方針で運用をしていると考えられる。次に、第4列目に利用者から計算ノード内/外へのアクセスに対する要望、および、今後計算ノード内/外へのアクセスに対してどのように対応していくかについてのアイデアについてのヒアリング結果をまとめている。今回調査対象となったHPCI構成機関の多くは特にない、という回答であったが、データ転送の相談を受けている構成機関もあった。そうした相談については、構成機関内で対応方針について検討・議論し柔軟に対応しているとのことであった。第5列目に計算ノード内/外へのアクセス制御をするに際してどのようなアクセス制御をしているか、またどのようなソフトウェアが必要かについてのヒアリング結果をした結果、IPS、VPN、firewallなどの回答が得られた。

表 3.2.10.17 システムセキュリティ、モニタリングとユーザ利便性についてのヒアリング結果

構成機関	計算ノードから外部への直接アクセス、外部から計算ノードへの直接アクセスを許可しているか？	(許可あるいは禁止している場合)、どのようなアクセス制御の方針で運用されていますか？	利用者から計算ノード内/外へのアクセスに対する要望はありますか？ 今後、計算ノード内/外へのアクセスに対してどのように対応していくかについてのアイデアがありますか？	計算ノード内/外へのアクセス制御をするにあたって、どのようなシステムソフトウェアをお使いになられていますか？または、どのようなシステムソフトウェアが必要とお考えになりますか？
大阪大学 D3 センター	内部から外部のみ OK（条件付き）	NAT	ない ・mdx のようなシステムの利用も一つの解。	ない。
九州大学情報基盤センター	内部から外部のみ OK	NAT	ない。	ない。 ・特にないが、PB クラスのデータを外部へ download

				要求があり対応検討予定。
東北大学サイバーサイエンスセンター	内部から外部、外部から内部、いずれも NG	・外（GitHub 等など）へのアクセスは、FE からのみ。	特になし	特になし
京都大学学術情報メディアセンター	内部から外部のみ OK (条件付き) ・専用クラスタコースを提供：計算ノードを運用から切り離してユーザが 直接ログイン可能になる。	NAT	データ転送の相談 ・相談・要望があると内部で対応方針を決め柔軟に対応。	VPN
筑波大学計算科学研究センター	内部から外部のみ OK (条件付き)	firewall 多要素認証 ・国別フィルタを導入	データ転送の相談 ・相談・要望があると内部で対応方針を決め柔軟に対応。	IPS
東京科学大学情報基盤センター	内部から外部 OK	firewall 多要素認証 ・国別フィルタを導入	特になし。	firewall
名古屋大学情報基盤センター	内部から外部のみ OK (条件付き)	firewall 多要素認証 ・国別フィルタを導入	あまりない。 ・センターとしての管理ポリシー（全体）で決めて	・ベンダに依頼している

			いる。データの持ち込み、持ち出しは HDD やコールドストレージで実施できるようにしている。 ・FE を切り離して専用線や VPN 対応は特殊対応。	
北海道大学情報基盤センター	内部から外部のみ OK (条件付き)	firewall	ない	特になし。
理化学研究所計算科学センター	許可している	外部からは SSH or HTTPS と VPN、内部から外部への通知は基本的にすべて許可（セキュリティホワイトペーパー：https://www.r-ccs.riken.jp/wp/wp-content/uploads/2021/07/f_security_whitepaper_202107.pdf から）	・ISV のライセンスサーバへのアクセスなどの要望があり、現在に至る。	特になし。
産業技術総合研究所	許可している（内->外）	firewall ・内部から外部へは特定ポートのみ。	・AI では、クローリングするケースがよく見られる。計算ノードでのクローリング自体がシステム（NW ,Storage)のボトルネックになるケースもある。	NAT+firewall

			スロットリング等の検討が必要。 ・ISV のライセンスサーバへのアクセスなどの要望があり、対応している。	
東京大学情報基盤センター	許可している（内→外) ・外→内は要申請	基本的に Outgoing はあまり制限していない。 inComing はデフォルト NG（プロトコル申請許可制）	ある ・利用者要望を取り入れて現在の運用になっている	NAT＋firewall（OS 標準機能とその外側のルータでも実施）
海洋研究開発機構	許可していない ・必要性があることは認識しており、所内のセキュリティ要件を満たせる解を検討中。	N/A	ある ・利用するアプリケーションの所外のライセンスサーバへの接続など。	経路自体ない

3.2.10.3.10 HPC センターの現状について：エコシステム、コミュニティ形成について

本ヒアリング項目では、図 3.2.10.14に示すスライドを用いて調査を行った。

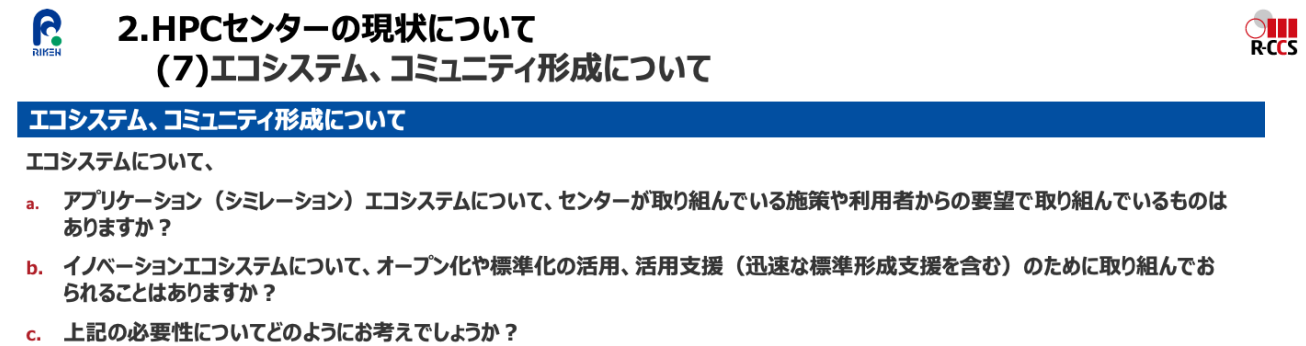


図 3.2.10.14　エコシステム、コミュニティ形成についてのヒアリング項目

表 3.2.10.18に、図 3.2.10.14で示したヒアリング項目のうちエコシステムに関する項目に対する調査結果をまとめた。まず、アプリケーションエコシステムに関しては、傾向として2種類の取り組みが見られる。1つ目は、利用者からの要望に応じたOSSやVASPの導入である。2つ目は、アプリケーションやGPU利用に関する講習会の開催である。これらの取り組みは、外部で創出された成果をHPCI構成機関内で活用するもの（エコシステムへの参画）と解釈できる。その一方で、各HPCI構成機関内で創出した成果を社会還元する取り組みとしては、理化学研究所（SPACK活用・レシピ整備への参画）や産業技術総合研究所（AIモデルの共有プラットフォーム）の取り組みがあげられるが、まだ数は少ない。

また、イノベーションエコシステムに関しては、HPCI構成機関全体で共通して取り組まれているものはないことがわかる。個別に観察した場合は、大阪大学によるデータ基盤の整備や、理化学研究所による富岳技術のデファクトスタンダード化といった社会還元の取り組みが見られる。また、産業利用の促進という観点では、計算資源の活用支援や、コミュニティポータルの運用といったことも取り組まれている。

これらエコシステムの必要性に関しては、約半数のHPCI構成機関が必要であると考えている。ただし、傾向としては、各HPCI構成機関で生み出された成果の社会還元をサポートするというよりは、エコシステムを取り込むことで利用者にとっての利便性を高めることが重要視されていると思われる。

表 3.2.10.18　エコシステムについてのヒアリング結果

構成機関	アプリケーション（シミュレーション）エコシステムについて、センターが取り組んでいる施策や利用者からの要望で取り組んでいるものはありますか？	イノベーションエコシステムについて、オープン化や標準化の活用、活用支援（迅速な標準形成支援を含む）のために取り組んでおられることはありますか？	上記の必要性についてどのようにお考えでしょうか？

大阪大学 D3 センター	利用者からの要望があるオープンソースアプリを積極的に導入している。	データ基盤 ONION を試験的に整備・運用し、産学共創のための計算基盤を整備している。	継続的な実施が必要不可欠と考える。また、そのために利用者の声を今以上に取り込める体制整備・拡充が必要であると考える。
九州大学情報基盤センター	RIST による国プロアプリの整備に協力しているが、センター独自の取り組みはまだない。	今のところ、具体的な取り組みはない。	今まで利用者から要望がほとんど無く、対応していなかった。今後要望が増えるようであれば、何らかの取り組みが必要になる。
東北大学サイバーサイエンスセンター	（未回答）	（未回答）	（未回答）
京都大学学術情報メディアセンター	・利用者から要望のあった OSS をプリインストールしている。 ・アプリケーションを用いた講習会を実施している。	特になし。	・アプリケーションエコシステムについては、利用者の満足度向上のため，継続して実施する必要性があると考える。 ・イノベーションエコシステムについては、現時点のシステム運用ならびに利用者支援において、検討していない。
筑波大学計算科学研究センター	特になし。	基本オープンソースを利用している。	必要に応じて対応する。
東京科学大学情報基盤センター	・一部アプリケーション(VASP)においてスパコン用ライセンスを取得し、ライセンス所持者がコンパイル済みバイナリを利用可能なように整備している。 ・OSS について、利用者からの要望があり、かつ他にも利用者が期待できる場合には、導入初年度は 10 本、その後は年に	特になし。	非常に重要な視点ではあるが、当センターのスーパーコンピュータのような汎用目的の場合には、特定分野の十分な人数の利用者を集めるのが難しい。目的の限られたスパコンでの施策実施がうまくいくポイントになるのではないかと考える。

	2 本の OSS の導入を行うことを仕様書に盛り込んでおり、実際に活用している。		
名古屋大学情報基盤センター	GPU プログラミングに関する講習会や、利用者が課題を持ち込むミニキャンプを開催。	・スタートアップの企業を含むスーパーコンピュータを民間利用制度（公開型、非公開型）がある。 ・活用支援として、無料で使えるトライアルユース制度を設けている。 ・産学共同研究による利用形態では、スタートアップ企業も廉価にスパコンを使える規定があり、産学のイノベーションに資する制度がある。	イノベーション促進のための産学共同研究にスーパーコンピュータを利用し活動支援する制度は、AI 開発や Society5.0 推進のために必要性があると考える。
北海道大学情報基盤センター	module の機能で代表的なアプリケーションやライブラリの利用を可能にしている。他のセンター等の富士通製のスパコンと類似の環境になっていると思われる。	特になし。	フラグシップのスパコンを頂点として、標準化や共通化が進むのは利用者の利便性が向上するように思う。スパコンの基本的な利用方法など（例：SSH 接続の方法）を含めて、国内のコミュニティで冗長な部分を効率化できるとよいのではないかと思う。
理化学研究所計算科学センター	・SPACK 活用・レシピ整備へ参画している。 ・「バーチャル富岳」構想に基づく互換 CPU 利用環境を提供している	富岳技術の defacto standard 化や調達システムへの標準技術採用に取り組んでいる。	必須である。（必要性は認識している）

産業技術総合研究所	・ABCI 上で作成されたデータセット、AI モデルを公開するためのカタログサービスを提供している。 ・コミュニティと協力して AI モデルの開発し、開発したモデルを共有するためのプラットフォーム（AI ハブ）を整備中である。また、LMM ハッカソンに取り組んでいる。	・コミュニティポータルを運営している。 ・コミュニティと協力して AI モデルを開発し、開発したモデルを共有するためのプラットフォーム（AI ハブ）を整備中である。また、LMM ハッカソンに取り組んでいる。	必要である。
東京大学情報基盤センター	アカウント付き講習会を多く開催し、啓蒙活動している。GPU に関して積極的に取り組んでおり、ツールの使い方などの講習会を多く開催している。	GPU に関して積極的に取り組んでおり、ツールの使い方などの講習会を多く開催している。	非常に重要だと考えている。裾野を広げるためにも、大事だと思って力を入れている。
海洋研究開発機構	特別なものはない。	特別なものはない。	慎重な対応が必要である。 当センターが公開するデータや解析結果は防災関連で利用されることが多く、データのオープン化の必要は把握しているが経済安定性にも影響を及ぼすものがあるため、慎重な対応が必要と考える。

表 3.2.10.19に、図 3.2.10.14示したヒアリング項目のうちコミュニティ形成に関する調査結果をまとめた。まず、研究コミュニティの形成に関しては、利用者同士の交流機会を創出する取り組みから、利用者講習会を開催する取り組みまで、程度に差はあるものの多くのHPCI構成機関において共通してコミュニティ形成の取り組みがなされている。研究者・研究室レベルの計算資源からセンターレベルの計算資源への連携という観点では、九州大学によるワークフローツールの整備や、北海道大学における計算資源の類似性確保といった取り組みが見られる。

産業利用コミュニティ形成に関しては、大阪大学によるシンポジウムの開催や、理化学研究所によるSociety 5.0推進拠点の設置、産業技術総合研究所によるGENIACプロジェクトへの参画といったコミュニティ形成の取り組みが見られる。産業利用の推進という観点では、大阪大学や東京科学大学による広報活動や、筑波大学や東京

大学における産業向けのトライアルユースの実施が見られる。

表 3.2.10.19 コミュニティ形成についてのヒアリング結果

構成機関	研究コミュニティの形成について取り組んでおられることはありますか？研究者や研究室レベルの資源とセンターレベルの資源の連携やコミュニティ形成と運用のための基盤技術の整備状況など。	産業利用コミュニティの形成について取り組んでおられることはありますか？研究成果・技術のマネタイズを目的とした産業利用活性化や企業間の協力促進など。
大阪大学 D3 センター	利用者交流会を数年前から実施している。若い研究者や学生が計算機センターの運用面や利用面に参画し、HPC、AI 分野の裾野裾屋拡大に取り組みつつある。 利用者拡大に対しては、世界と伍する学生支援枠となづけた公募型利用制度を設立し、学部、大学院の学生が学位を取得する際のスーパーコンピュータ利用支援を行っている。 運用面に対しては、利用者交流会のなかで、大学院生が講師となり、Python セミナーを開催するなどを行い、セミナー受講の障壁を下げつつ、将来の人材育成に取り組んでいる。	産業利用制度の広報を積極的に行っている。また、毎年開催する Cyber HPC Symposium では、産学の研究者からの研究事例、話題提供を行い、センターおよび産業利用制度の促進に尽力している。 また、動画による研究紹介 HPSC では、企業の方の研究成果公開についても積極的に取り組んでいる。
九州大学情報基盤センター	研究者グループのファイル共有手段として NextCloud を提供している。また、研究室の計算機とセンターの計算機を連携させた利用のためのワークフローツールとして、OPEN WHEEL と ML Flow を提供している。	今のところ、具体的な取り組みは無い。
東北大学サイバーサイエンスセンター	（未回答）	（未回答）

京都大学学術情報メディアセンター	該当なし。	該当なし。
筑波大学計算科学研究センター	計算物理学研究センター設立時から、学際共同利用プログラムを通じて研究コミュニティの形成に取り組んでいる。	学際ハブ拠点の事業の一環として企業のスパコンお試し利用を実施している。
東京科学大学情報基盤センター	学内の研究室間コミュニティなどに関してはマンパワーの問題もありあまり進められていないが、GPU ベンダや複数大学センターによるプログラミング講習会などには積極的に運営参加・呼びかけ等をしている。	センターとして、学会・イベントへのブース参加を行い、産業利用を含む潜在ユーザへの広報活動を行っている。
名古屋大学情報基盤センター	名古屋大学 HPC 計算科学連携研究プロジェクトにより、高性能計算と計算科学の学際領域分野の研究コミュニティ形成に取り組んでいる。GPU 利活用について、研究室レベルの小規模計算から、センターの大規模 GPU 計算へ繋げるための講習会を行っている。	本センターの講習会は企業技術者も参加可能であり、企業におけるスーパーコンピュータ資源の活用を促進することで、産業利用コミュニティの形成を目指している。
北海道大学情報基盤センター	現スパコンは研究室の環境との類似性を意識して x86 アーキテクチャを採用しており、主要な OSS を導入した。	現状、特に取り組みはないが、主要な利用者（大学）が開発したソフトウェアに関して、ソフトウェア込みでのスパコンの民間利用の需要があるという話を聞いている。
理化学研究所計算科学センター	様々な観点で取り組んでいる ・Society5.0 の推進拠点としてコンソーシアムの形成には意識的に取り組んでいる。 ・様々な組織と協定を結んでいる。	センターとして、2021 年度 Society 5.0 推進拠点を設置し推進。
産業技術総合研究所	・コミュニティポータルを運営。 ・コミュニティと協力して AI モデルの開発、開発したモデルを共有するプ	経済産業省の GENIAC プロジェクトで形成された AI 開発者のコミュニティに参画し、ABCI の利用を訴求している。

	ラットフォーム（AI ハブ）を整備中。また、LMM ハッカソンを取組み中。	
東京大学情報基盤センター	大規模データを扱うなど、様々な観点で相談は来るので、協力できるところは協力するという姿勢で取り組んでいる。（実験系、観測系の人など様々な分野と繋がりありますし、病院、社会科学の人からも相談があり対応している）	トライアルユース制度をベースに産業利用を支援している。
海洋研究開発機構	ユーザ会を実施する程度。	産業利用が少ないので特にない。

3.2.10.3.11　　新しいチャレンジについて

本ヒアリング項目では、図 3.2.10.15に示すスライドを用いてヒアリングを行った。本ヒアリング項目では、各HPCI構成機関が考える新しい計算ニーズや要望に対するチャレンジについて調査した。図 3.2.10.13に示すように、本ヒアリング調査の「1.HPCセンターについて」、「2.HPCセンター の現状について」で実施した調査項目とも関連させ、(1) インタラクティブvsバッチの利用携帯について、(2) IoTセンサ等のシステム外からのリアルタイムデータ処理について、(3) 仮想計算機やコンテナなどへの対応を一例として提示し、スーパーコンピューティングシステムを実際に利用・運用される観点から、各HPCI構成機関の管理者が次期スーパーコンピューティングシステムにむけてのシステムソフトウェアを明らかにすることを目的とした。

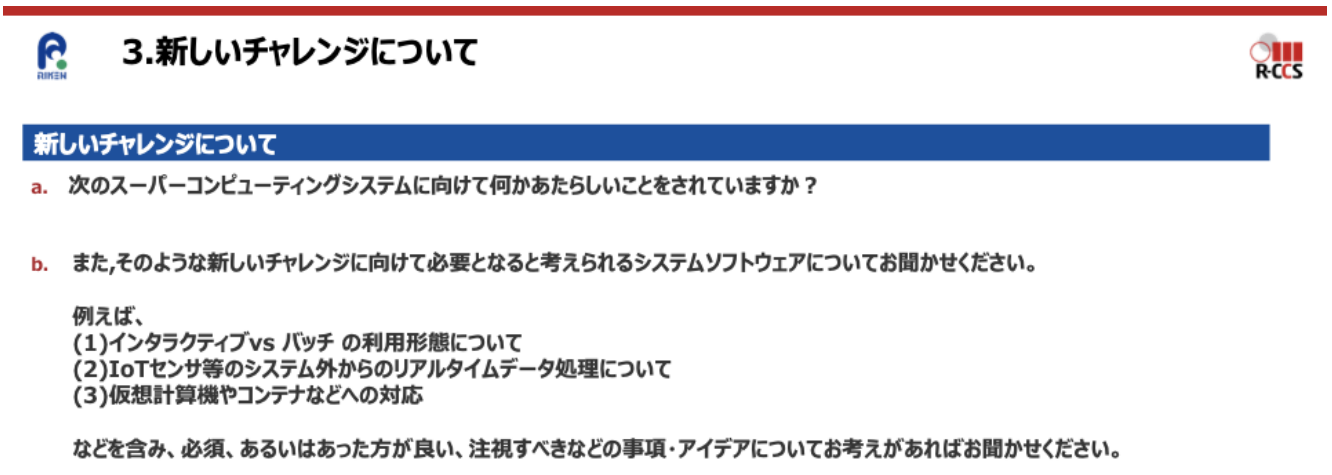


図 3.2.10.15　新しいチャレンジについてのヒアリング項目

表 3.2.10.20に図 3.2.10.15に示すヒアリング項目の結果を記す。これらの結果からはストレージ、インタラクティブ処理、システム外のデータ連携（リアルタイム処理含む）、AI系の強化に関しては複数のHPCI構成機関が

チャレンジであると回答している。一方、各HPCI構成機関での問題意識も多様に異なることもみて取れる。これらの問題意識をさらに調査することにより、次期のスーパーコンピューティングシステムの形を想像することにつながりうる。大阪大学D3センターではMassive CPUや量子コンピュータの動向を注視しながら次世代のハイブリッドシステムを検討している。筑波大学計算科学研究センターではハイブリッドスパコン、特徴別ホモジニアススパコンと各々明確に特徴を区別しながら整備する方針である。九州大学情報基盤センターは同一の物理ノード上でCXL、ワークフロー、Open OnDemandなどを混在させる仕組みを挙げている。また東北大学サイバーサイエンスセンターはコンテナへの対応を検討、理科学研究所計算科学センターでは生成AIを活用したサポートサイトのQ/Aシステムを検討している。

表 3.2.10.20 新しい計算ニーズ・要望に対するチャレンジに対するヒアリング結果

構成機関	次のスーパーコンピューティングシステムに向けて何かあたらしいことをされていいますか？ また、そのような新しいチャレンジに向けて必要となると考えられるシステムソフトウェアについてお聞かせください。
大阪大学 D3 センター	・ハイブリッドシステム 　既存の CPU、GPU,ベクトルなどの技術動向を抑えながら、これから出てくるような Massive ~cpu や量子コンの行方を注視しながら次の世代の Hybrid を検討。 ・ストレージ（データ集約・共有基盤）の強化
九州大学情報基盤センター	・インタラクティブ利用の進化 ・課金体系の変更：従量制（資源利用効率の向上） ・同じ物理ノード上で混在させる仕組み。 　CXL、OpenOnDemand、 ワークフロー、MLFlow がチャレンジ。 ・ストレージサービス（次期で導入予定）
東北大学サイバーサイエンスセンター	・インタラクティブバッチ（AOBA-S は導入済み）や IoT センサ等の外からのデータ集約はやろうとしている。Jupyter 環境の整備。 ・コンテナへの対応は検討（ベクトルエンジンに対応した singularity など）。
京都大学学術情報メディアセンター	・利用需要の変化に対応していけるように検討（現在では、AI 系とかの強化）を推進。Python 環境。 ・データ連携の仕込み（センサ等のシステム外からのリアルタイムデータ処理需要を見越して）
筑波大学計算科学研究センター	・リアルタイム要求に対応できることを検討（まだ、ソリューションも出ていないが）。専用システムではなく汎用システムとしてのソリューション。 ・ハイブリッドスパコン x 1 vs 特徴別ホモジニアススパコン x n

	（コストと使い勝手、最先端技術取り込み）では、整備方針として複数スパコン。整備方針は各々明確に特徴を区別していく。ただし、シミュレーション高度化やデータ共有の観点でクラスタ連携は必要。インターコネクト、ストレージなどの要素技術やクラスタ共通のプログラミング？アプリケーション環境をクラスタ連携可能にするものにしたい。 ・また、できることや、要望には対応。基本性能（メモリ、ストレージなど）を可能な限り向上。
東京科学大学情報基盤センター	・（次の調達が）6年後なのでドラスティックに新しく(何かを始める)とかは、今はない。 ジョブ分割とかそういうその細かい改良や既存の枠組みの中で運用中に少しずつ数増やしていくとかはある。 ・GPU も TSUBAME 当初のような難しさは無くなりコモディティのレベルになってきた（GPU 使うためにそれほど特殊な技能が必要なくなってきたライブラリとか、もうアプリインストールするだけでも使えるようになってきている）。東工大には GPU ユーザが来るので（利用事例を含め）使いやすさという面で取り組みは今後も継続する。
名古屋大学情報基盤センター	・クラウドストレージサポートは検討。 ・低消費電力の COLD ストレージ技術を技術が間に合えば次期でも検討。 ・サブシステム間のジョブ連携（シミュレーションと AI の連携）や AI の実行基盤となりつつある仮想化環境の高速化（IO 等も）もまだまだ研究的な意識が強い印象であるが watch していく。
北海道大学情報基盤センター	・GPU の導入。 ・ユーザニーズをある程度考えていれたものがあって、それが定常的に高い稼働率で推移している。要は結局求められるものをいれる方法が一番使ってもらえるというのを実感している。そういう意味で、次期に関しても取り組む。
理化学研究所計算科学センター	・生成 AI を活用したサポートサイトでの Q/A システム ・稼働率向上に向けたスケジューラ調整 ・省電力化に向けたジョブの電力特性の把握と省電力機構の利用モードの最適化 など
産業技術総合研究所	・ファシリティ側の制約の中でいかに高性能なシステムを提供できるかを追求する。

	・ABCI 登場の頃とは状況が大きく変わってきている。高性能なシステムの提供に加えて ＡＩ開発を加速するために方法論の確立のための利便性強化や、効率的なモデル開発の仕組みの提供などで、アカデミアと民間への橋渡しを支援できるようにしたい。
東京大学情報基盤センター	・MPI 互換の API でどんな環境でも通信できる仕組みを研究開発している。これにより、適材適所の利用により既存のスパコン連携の広がりを進め、データサイエンス・Ai x シミュレーションなどの広がり、数値計算や量子スパコン連携のプラットフォームを実現することを目指している。 ・運用面でもユーザビリティを向上させるような取り組み（OpenOndemand や Jupyter からスパコンを利用しやすくする&ワークロードスケジュールを容易にすることなど）を進めている。また、ストレージの柔軟性のある使い方（共用ストレージや他のストレージとの連携）を進める予定。
海洋研究開発機構	・技術動向は見つつ新しい技術を取り入れて，実行効率高める計算ノードを整備していく。 ・一方で研究が進めば，計算が進めばデータが出てきますので，それをどう保管，保存するのとか大規模なストレージシステムのあり方を考え、整備していく。 ・上記を実現する大規模なシステムなのでコスト面，予算の面や電力の面の制限のバランスをどうカバーするのかも重要でバランスを考えた取り組みをしていく。 ・技術的なことについては，生成 AI とか（GPU どう取り入れる）の新技術を取り入れていく。

3.2.10.4利用状況調査

次に、国内（HPCI参画機関：14機関、29システム）、国外における計算機センターを欧州、米国、アジアよりピックアップ（11機関、22システム）し、システムソフトウェア、ライブラリ、アプリケーション導入状況をまとめた(付録１、2)。国内の導入状況は、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)にて公開されている状況をまとめている。当該海外調査は、HPCIにてリスト化されているシステムソフトウェア、ライブラリ、アプリケーションの導入の有無について調査している。国内利用状況調査は、2023年度に続き、2024年度のリストついて調査し、海外は2023年度のリストを元に調査を実施している点に注意されたい。また、海外導入状況は、調査当時に掲載されているウェブ等の公開情報を元にしているため、必ずしも当該計算機センターの保有するスーパーコンピュータへのシステム導入状況が最新であるとは限らない点には注意されたい。また、海外調査対象機関では情報公開範囲にばらつきがあり公開されていない情報も多くあると推測している。そのため、海外調査機関で公開されている情報は、今回調査対象としたシステムソフトウェア、ライブラリ、アプリケーション以外に多く導入されているものがあることに注意されたい。

図 3.2.10.16に「国内調査のソフトウェア分類と数」、「国内調査のソフトウェア導入数」を示す。また、図3.2.10.17に「海外調査のソフトウェア分類と数」、「海外調査のソフトウェア導入数」を示す。

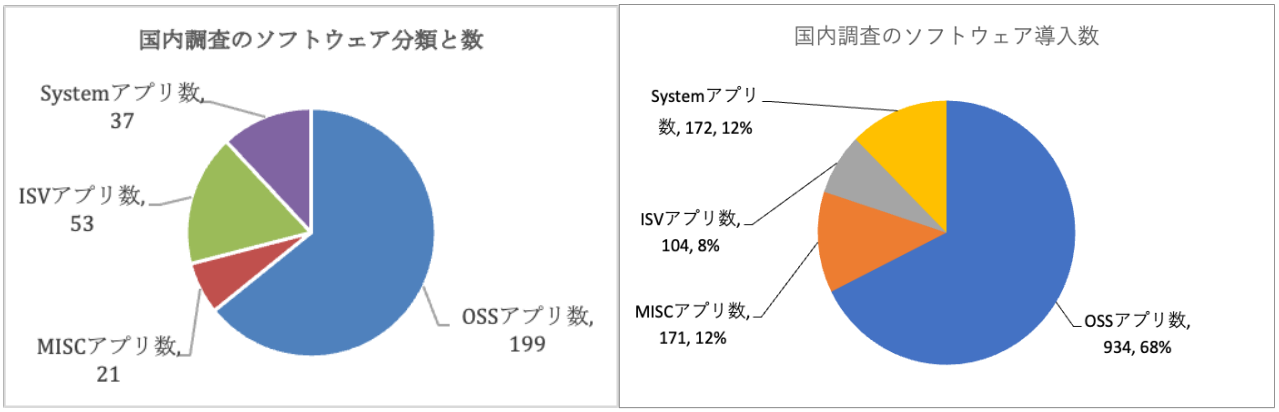


図 3.2.10.16　国内調査のソフトウェア分類と数と国内調査のソフトウェア導入数

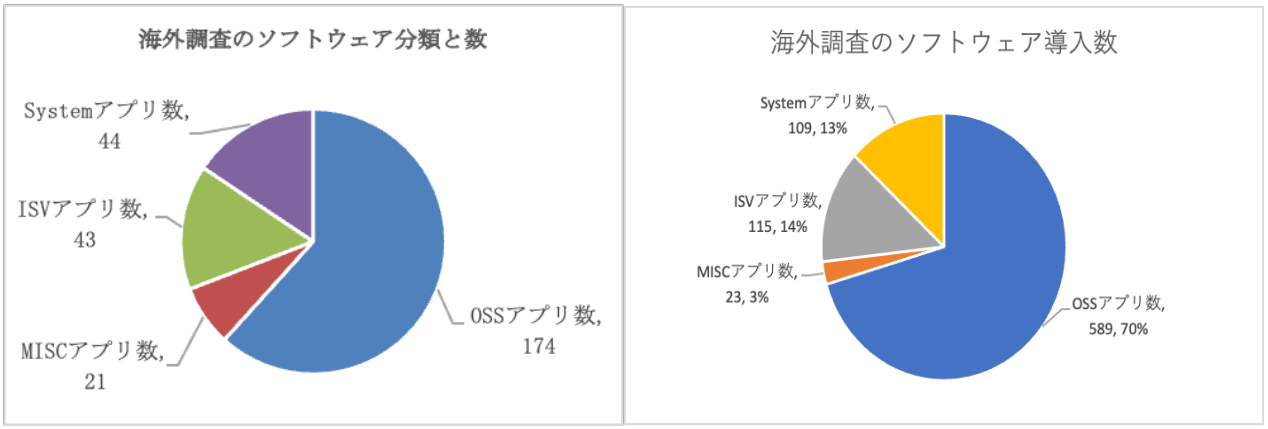


図 3.2.10.17　海外調査のソフトウェア分類と数 と海外調査のソフトウェア導入数

国内と海外調査では、調査対象の「ソフトウェア分類と数」が異なるのは、国内はHPCIアプリリスト（RISTの整備アプリリスト）の2024年度版を使用し、海外は2023年度版を使用しているためである。導入数を見ると国内、海外の違いは、ISV,OSSとも海外調査機関の導入数比率が高い結果となった。しかし、「導入比率が高い＝利用者が多い」と一概には判断できないが、導入率が高いソフトウェアは利用要求が高いと考えられる。また、ISVについては「ISV=産業利用の傾向が高い」と考えた「産業利用者が多い」とも結論づけることはできないが、アプリケーション市場への貢献は高いと判断できる。一方、国内調査ではヒアリング対象とした機関の多くは、「アプリケーションの用意を利用者マター」としていることが多いことがわかっており、実際の導入数は同等程度と考えられるかもしれないが、海外調査対象機関の方が利用者へのOSS ,ISV利用の負担は低いと感じられる。

次に、ソフトウェア分類毎の導入数、導入率を示す。ただし、以下のランクからは最先端の研究を支えているソフトウェアであることを推察できるが、先端性を判断することはできないと考える。先端性の高いソフトウェアは、研究開発の現場で開発中であるため利用者（研究者）自身が導入し開発しているなど導入数が少ないケースを想像できる。

表 3.2.10.21は、調査対象ソフトウェア全体での導入率の高いソフトウェアリスト（上位30）である。各々の調査対象機関の利用者の分野が個々に異なるためリストから読み取ることは難しいが、1/3のソフトウェアは国内、海外とも上位にランクイン（黄色の編みかけ部分）していることがわかる。これらのアプリケーションは国内、海外の

利用者（研究者）にとってキーアプリケーションとなっている可能性が高い。

表 3.2.10.21　調査対象ソフトウェアの国内、海外ランク（上位 30）

	国内		導入数	導入率		海外		導入数	導入率
1	PHASE/0	MISC	21	72%		HDF5	OSS	18	82%
2	ABINIT-MP	MISC	20	69%		NetCDF	OSS	18	82%
3	FrontISTR	OSS	20	69%		FFTW	OSS	17	77%
4	GNU コンパイラ	OSS	20	69%		CMake	OSS	15	68%
5	Quantum ESPRESSO	OSS	20	69%		GROMACS	OSS	15	68%
6	GROMACS	OSS	19	66%		Boost	OSS	14	64%
7	HDF5	OSS	19	66%		CUDA	System	14	64%
8	R	OSS	19	66%		LAMMPS	OSS	14	64%
9	FrontFlow/blue	MISC	18	62%		NAMD	MISC	14	64%
10	OpenFOAM	OSS	18	62%		Python	OSS	14	64%
11	OpenMX	OSS	18	62%		MATLAB	ISV	13	59%
12	AkaiKKR	MISC	17	59%		OpenFOAM	OSS	13	59%
13	FFTW	OSS	17	59%		PETSc	OSS	13	59%
14	GENESIS	OSS	17	59%		CP2K	OSS	12	55%
15	MODYLAS	MISC	17	59%		Ncview	OSS	12	55%
16	NetCDF	OSS	17	59%		NWChem	OSS	12	55%
17	Python	OSS	17	59%		OpenMPI	OSS	12	55%
18	TensorFlow	OSS	17	59%		Singularity	System	12	55%
19	HΦ	OSS	16	55%		VASP	ISV	12	55%
20	LAMMPS	OSS	16	55%		Gaussian	ISV	11	50%
21	ALAMODE	OSS	15	52%		Intel コンパイラ	System	11	50%
22	FFVHC-ACE	MISC	15	52%		METIS	OSS	11	50%
23	mVMC	OSS	15	52%		ParaView	OSS	11	50%
24	NTChem	MISC	15	52%		R	OSS	10	45%
25	Phonopy	OSS	15	52%		AMBER	ISV	9	41%
26	PyTorch	OSS	15	52%		Gnuplot	OSS	9	41%

27	SALMON	OSS	15	52%		GNU コンパイラ	OSS	9	41%
28	SMASH	OSS	15	52%		Intel MKL	System	9	41%
29	Gaussian	ISV	14	48%		Julia	OSS	9	41%
30	Intel コンパイラ	System	14	48%		PAPI ライブラリ	OSS	9	41%

：国内、海外ランクイン

次に、ISV（表 3.2.10.22）、OSS（表 3.2.10.23）の分類別の導入数、導入率を示す。参考として、MISC（表 3.2.10.24 MISCソフトウェアの国内、海外のランク（上位19））の結果を記載しておく。ISVの結果は、上位20位にランクしているソフトウェアの内、約50%のソフトウェアが国内、海外に共にランクインしている結果となった。同様にOSSの結果は、上位30位以内のソフトウェアの約60%以上が 国内、海外に共にランクインしている結果となった。

表 3.2.10.22　ISV ソフトウェアの国内、海外のランク(上位 20)

ISV	国内	導入数	導入率		海外	導入数	導入率
1	Gaussian	14	48%		MATLAB	13	59%
2	AMBER	8	28%		VASP	12	55%
3	AVS/Express	7	24%		Gaussian	11	50%
4	CuDNN	5	17%		AMBER	9	41%
5	MATLAB	5	17%		IDL, ENVI	8	36%
6	NCCL	5	17%		Abaqus	7	32%
7	VASP	5	17%		CuDNN	7	32%
8	Arm Forge	4	14%		Mathematica	6	27%
9	Mathematica	4	14%		NCCL	6	27%
10	GaussView	3	10%		ANSYS Fluent	5	23%
11	IDL, ENVI	3	10%		ANSYS Mechanical	4	18%
12	MAGMA	3	10%		Molpro	4	18%
13	Marc	3	10%		GaussView	3	14%
14	MicroAVS Pro	3	10%		LS-DYNA	3	14%
15	Molpro	3	10%		MAGMA	3	14%
16	MSC Nastran	3	10%		TotalView	3	14%

17	NAG 数値計算ライブラリ	3	10%		ANSYS Workbench	2	9%
18	LS-DYNA	2	7%		Arm Forge	2	9%
19	V-FaSTAR	2	7%		HyperWorks	2	9%
20	Advance/FrontFlow/red	1	3%		Vampir	2	9%

：国内、海外ランクイン

表 3.2.10.23　OSS ソフトウェアの国内、海外のランク(上位 30)

OSS	国内	導入数	導入率		海外	導入数	導入率
1	FrontISTR	20	69%		HDF5	18	82%
2	GNU コンパイラ	20	69%		NetCDF	18	82%
3	Quantum ESPRESSO	20	69%		FFTW	17	77%
4	GROMACS	19	66%		CMake	15	68%
5	HDF5	19	66%		GROMACS	15	68%
6	R	19	66%		Boost	14	64%
7	OpenFOAM	18	62%		LAMMPS	14	64%
8	OpenMX	18	62%		Python	14	64%
9	FFTW	17	59%		OpenFOAM	13	59%
10	GENESIS	17	59%		PETSc	13	59%
11	NetCDF	17	59%		CP2K	12	55%
12	Python	17	59%		Ncview	12	55%
13	TensorFlow	17	59%		NWChem	12	55%
14	HΦ	16	55%		OpenMPI	12	55%
15	LAMMPS	16	55%		METIS	11	50%
16	ALAMODE	15	52%		ParaView	11	50%
17	mVMC	15	52%		R	10	45%
18	Phonopy	15	52%		Gnuplot	9	41%
19	PyTorch	15	52%		GNU コンパイラ	9	41%
20	SALMON	15	52%		Julia	9	41%
21	SMASH	15	52%		PAPI ライブラリ	9	41%

22	PETSc	14	48%		ParMETIS	9	41%
23	FFX	13	45%		PyTorch	9	41%
24	ParaView	13	45%		SIESTA	9	41%
25	CMake	12	41%		TensorFlow	9	41%
26	Parallel NetCDF	11	38%		LAPACK	8	36%
27	VisIT	11	38%		Parallel NetCDF	8	36%
28	Keras	10	34%		Quantum ESPRESSO	8	36%
29	OpenMPI	10	34%		zlib	8	36%
30	ParMETIS	10	34%		Anaconda	7	32%

：国内、海外ランクイン

表 3.2.10.24　MISC ソフトウェアの国内、海外のランク（上位 19）

MISC	国内	導入数	導入率		海外	導入数	導入率
1	PHASE/0	20	69%		NAMD	14	64%
2	ABINIT-MP	20	69%		GAMESS	8	36%
3	FrontFlow/blue	20	69%		ABINIT-MP	1	5%
4	AkaiKKR	19	66%				
5	MODYLAS	19	66%				
6	FFVHC-ACE	19	66%				
7	NTChem	18	62%				
8	GAMESS	18	62%				
9	NAMD	17	59%				
10	FrontFlow/red	17	59%				
11	GEANT4	17	59%				
12	REVOCAP	17	59%				
13	MyPresto	17	59%				
14	SALS	16	55%				
15	GRRM	16	55%				
16	h5z-zfp	15	52%				
17	mapsplice2	15	52%				
18	SCALE	15	52%				

19	Tinker	15	52%				

：国内、海外ランクイン

ソフトウェアの利用状況調査では、ほぼ同一の調査対象のソフトウェアリストについて国内、海外での利用状況調査を実施し、利用動向の調査を行ったが、個々のソフトウェアの利用頻度まで調査するまでに至らなかった。この点については、海外は調査不可能であるし、国内の調査対象機関へのヒアリング調査では、module（Environment Modules等）による調査が可能であるとの見解をいただいたが、正確にアプリケーションレベルの実行頻度や利用者数まで統計情報を分析することが難しいとの意見をいただいた。このため、今後の調査機会がある場合、統一的なデータ採取手法を確立した上で調査の必要がある。

また、どのようなソフトウェア利用(利用頻度を含めて）があることを知ることは、システム整備の効率化や運用負荷の想定、必要な整備、運用予算の確保等で有効であるかもしれないが、利用ソフトウェアの内容から研究内容を想定されることを懸念する一部利用者（特に産業利用者）も存在するため、データの取り扱いについて注意が必要である。

全調査対象機関の調査結果詳細は、下記の付録として添付されている。詳細は以下の付録資料1、2を参照願いたい。

- 付録 1 日本主要 HPC 拠点調査表（調査のフォルダ）
- 付録 2 海外利用状況調査表

3.2.10.5ヒアリング調査および利用状況調査から見えてくるもの

本節では、3.2.10.4節ヒアリング調査、および、3.2.10.5節利用状況調査から、多くのセンターで抱えている問題点、そして、次期システムで取り組むべきシステムソフトウェアの観点から課題や改善点について、本サブグループとしての考えをまとめる。また、3.2.10.6節に、本サブワーキンググループとして考える改善点を付記する。

3.2.10.5.1 多くの計算機センターで抱えている問題点

多くの計算機センターは、スーパーコンピューティングシステムの提供する性能については注意深くシステムソフトウェアやライブラリ、ツールについての整備を行なっている現状がある。一方、そのスーパーコンピューティングシステムを効率的に動かす、また、どのように使われているかを把握するといった点でのシステムソフトウェアやライブラリについて、時間・労力・予算の点で優先度が低くなっている傾向がみられる。スーパーコンピューティングシステムの大規模化が進展するに伴い、このシステムを効率的に動かす、実態を把握するといった視点の重要性はますます高くなっていると考えられる。どのような高性能なスーパーコンピューティングシステムも、それを実際に効率的に動作させることができなければ、利用者はその恩恵をうけることは難しい。そのような視点から、本サブワーキンググループでは、多くの計算機センターが共通的に抱えている問題点として下記 3 項目をとりあげたい。

(1) 省電力化

電力総量規制等はここ数年では発生していないが、電力料金等の高騰は運用費へのインパクトが大きくなっている。 このため、消費電力量を把握しサービスレベルを維持しつつ省電力化へ対応する運用の備えが必要と考えているセンターは多い。

(2) 運用省力化

シミュレーションの高度化やデータ分析の多様化により複数のヘテロアーキテクチャ・クラスタの必要度が増し、各センターの運用クラスタ数が増加する傾向にある。このため、システムの安定運用を担う運用員の負担増に対応する更なる運用省力化が必要と考えているセンターは多い。

また一方で、運用サービスは利用者の平等性等を確保するなどの各センターの運用ポリシーによって設計され、運用ソフトウェアの運用制限パラメータ等を設定している。これらの制限パラメータは、運用の偏りや急激な運用状況の変動発生時に制限解除や緩和、制限強化などの対応が必要であるが、現状は運用員が判断し手動で切り替える運用となっていることが多い。このため、運用クラス多数の増加により運用監視の容易化や運用状況の変動に対応した運用省力化が必要と考えているセンターは多い。

(3) サービス利用者の実態把握

多くのHPCI構成機関におけるサービス利用者の利用実態把握は、ジョブスケジューラから取得される情報に留まっている。サービス利用者の利用実態は、システムの調達や運用における意思決定の判断材料となる重要な情報である。このため、より幅広い利用実態を把握するためのソフトウェア、アプリケーションが必要となると考える。 実際にヒアリングにて、商用ソフトウェアの見直しや、利用に困っているサービス利用者の抽出等といった目的のためにサービス利用者の利用実態を把握したいとの意見をいただいている。

また、われわれの調査では、アプリケーションの利用実態についてもヒアリングしたが、HPCI構成機関側としてはあまり関心がない傾向があったためか、Environment modulesやSpack等によるアプリケーションのロード回数を調査するのみで済ませるHPCI構成機関や、利用実態を調査していないHPCI構成機関が多数であった。このため、次々期のスーパーコンピューティングシステムに向けた将来的な調査を見据えると、サービス利用者の利用実態として、アプリケーションの利用実態も把握することが今後必要となると考える。

3.2.10.5.2 次期システムに向けて取り組むべき開発や改善点

3.2.10.5.1節に記した問題点に基づき、次期システムでのシステムソフトウェア観点で、下記はシステムソフトウェア観点でも取り組むべき開発や改善点としてあげられる。

(1) 省電力化

複数のヘテロアーキテクチャ・クラスタを運用するセンターが増えている状況で、消費電力のセーブが必要となった場合の対応計画の策定には、サービスの全停止ではなく一部停止などサービスレベルを可能な限り維持した対応が必要である。このため、計算資源単体ではなく資源の関係性を把握できる消費電力状況等の見える化が必要となる。

近年、計算資源そのものの消費電力状況の見える化は、達成しつつあるが冷却設備や周辺装置などを含めた消費電力状況についてサービス単位や時系列で全体を把握できるものが必要となると考えられる。また、消費電力制御のための運用変更を安全、且つ容易に操作可能な運用ソフトウェアの対応が必要になると考える。

(2) 運用省力化

システム監視の高度化と運用状況の分析自動化、運用変動に合わせた運用変更の容易化が必要。資源利用の超過などの監視は、各センターで設定（制限パラメータ等）し運用されているが、事象発生時や制限抵触時の通知に留まっている。個々の運用ソフトウェアでは障害耐抗性を備えた機能を有しているが、サービス利用量（複数の監視項目の負荷変動で判断するもの）の変動や資源利用量の変動に急激な変化を観測した場合に運用員へのアラート通知や予め定めておいた運用変更（制限パラメータの緩和や、一時解除など）の運用変更の容易化を可能とする運用ソフトウェアの対応が必要になると考える。

(3) サービス利用者の実態把握

効果的なシステム調達や運用のためには、サービス利用者の利用実態の把握が必要である。現状における主要な利用実態の把握方法はジョブスケジューラから統計情報を取得することであるが、利用に困っているサービス利用者を抽出するなどといった重要な情報を分析するまでは手が回っていないのが現状である。そのため、ジョブスケジューラから取得できる統計情報を集計するソフトウェア、アプリケーションの開発や、集計・分析のためのノウハウを共有する取り組みが必要であると考える。

また、アプリケーションの利用実態把握に関しては、モジュールのロード回数を記録する仕組みを採用するHPCI構成機関がある一方で、記録を取っていないHPCI構成機関も存在する。そのため、そうしたモジュールロードの回数を記録する仕組みをHPCI全体において共通化することが必要と考える。

3.2.10.5.3 その他の取り組むべき課題

本サブワーキンググループでは、ヒアリング調査を通じて、ワークフローツールの必要性や利用状況についても調査を行った。しかしながら、こちらは各センターによって“ワークフロー“の定義や考え方が異なる状況があった。しかしながら、現状、計算ニーズやワークロードの多様化に基づきヘテロアーキテクチャ・クラスタ型のセンターが増えている傾向がある。そういったことから、これはセンター共有の課題ではないが、異なるアーキテクチャ間でのアプリケーションの錬成の仕組みは取り組むべき課題といえる。今後、複数のヘテロアーキテクチャ・クラスタを運用するセンターが増え、シミュレーション、データ分析の適材適所利用が進むと考えられる。この時、クラスタを跨るアプリケーションの連成や実行順序制御のため異なるMPI間の通信や、クラスタを跨るアプリケーションの実行順序制御のためのワークフロー等の仕組みを提供する必要があると考えられる。次期スーパーコンピューティングシステムの形態にもよるが、このような項目も取り組むべき課題としてあげられる。

3.2.11　補足調査：GPU 向けプログラミングフレームワーク及びライブラリの評価

近年の高性能計算 (HPC) において、GPUは計算性能向上の鍵となるハードウェアコンポーネントである。しかし、GPUのアーキテクチャはメーカーごとに異なり、また、それぞれのGPU向けに提供されるプログラミングフレームワークやライブラリの性能差がアプリケーションの実行効率に大きく影響を与える。そのため、特定のGPUを対象とするシステム設計やソフトウェア開発においては、事前に各種プログラミング環境の特性を正確に把握し、最適な選択を行うことが重要となる。本評価では、次世代のHPCシステムにおけるGPU選定やプログラミングモデルの有効性を検討するため、異なるGPUアーキテクチャ (NVIDIA GH200 Grace Hopper Superchip, AMD MI250) に対してベンチマークを実施し、それぞれのプログラミングフレームワーク (CUDA, HIP, OpenMP) の性能を比較する。特に、CUDAからHIPへの移行が適切に行えるか、OpenMPのGPU対応が実用的かどうかを評価することで、GPU間の移植性やパフォーマンスの違いを明確にし、今後の開発方針の指針とすることを目的とする。

3.2.11.1 ベンチマーク設定

候補となるGPUの性能を評価するためのベンチマークを実施した。作業は理化学研究所のR-CSS Cloud環境を用いて行った。具体的には以下の2種類の計算機上において実施した。

- gh200：NVIDIA GH200 Grace Hopper Superchip
- mi250：AMD MI250 64GB x 8

本件では、HeCBench の各ベンチマークに対して最大で5通りのベンチマークを実行した。1. および 4. は gh200、2. 3. 5. はmi250において実施した。実行が不可能であった場合を除いて、基本的にHeCBench内で提供されている makefile の ターゲット run を実行することでベンチマーク評価を行った。また、実行時間は time コマンドを用いて計測した。

1. CUDA版 (*-cuda)
2. HIP版 (*-hip)
3. CUDA版をHIPIFYによってHIP化した版 (*-higified)
4. OpenMP版をCUDAにおいて実行した版 (*-omp_nvc)
5. OpenMP版をHIPにおいて実行した版 (*-omp_aomp)

（「*」はそれぞれのベンチマークプログラム名が入る）

3.2.11.2 ベンチマーク結果

HeCBench の各ベンチマークの実施状況を下表 3.2.11.1に示す。target はベンチマークが構築できた数、done は構築できた数のうち実施できた数を示し、%はその割合である。

表 3.2.11.1　ベンチマーク実施状況

	cuda	hip	hipified	omp_nvc	omp_aomp	total
target	487	483	487	317	318	2092
done	476	462	417	262	289	1906
%	97.7	95.7	85.6	82.6	90.9	91.1

cuda、hip、hipified によるHeCBenchベンチマークの実行時間を比較するため、cudaでの実行時間を1とした場合の、hip, hipifiedの相対実行時間を表及びグラフで示す。このため、cudaで実行できなかったベンチマークは含めない。hip、hipified で実行できなかった場合、実行時間は0とした。

傾向を比較するため、実行時間が10倍超、10倍～2倍、2倍～1.2倍、1.2倍～0.8倍、0.8倍～0.5倍、

0.5倍～0.1倍、0.1倍未満となる数を下表 3.2.11.2で示す（0となる場合は0.1倍未満と別にカウント）。

表 3.2.11.2 実行時間比の分類

	10～	10～2	2～1.2	1.2～0.8	0.8～0.5	0.5～0.1	0.1～	0	total
hip/cuda	11	55	66	60	69	144	45	16	475
hipified/cuda	14	47	58	60	62	131	41	62	475

これにより、hipまたはhipifiedがcudaより2倍～10倍実行速度が速い（実行時間が短い）ものが最も多いことがわかった。今後は異なる環境・問題設定で評価を続ける予定である。

各値を表及びグラフにしたものを示す（表 3.2.11.3～表 3.2.11.7、図 3.2.11.1～図 3.2.11.12）。値はhip/cuda の比が大きい順に並べた。(全475項目)

なお、グラフでは横軸の最大値を「10倍」としたため、10倍以上となっている主なベンチマーク（bh ～ prefetchまでの11項目）については、グラフ中に表を挿入して値を示した。

（表・グラフは次ページより）

表 3.2.11.3 ベンチマーク実行時間比較（1）

name	cuda/cuda	hip/cuda	hipified/cuda
bh	1	823.020	1173.319
ccs	1	55.970	55.970
gpp	1	34.909	34.284
convolution3D	1	26.259	0.000
blas-gemmEx2	1	19.544	0.000
blas-gemmEx	1	19.430	0.000
phmm	1	17.451	17.451
mrc	1	13.355	13.390
swish	1	13.173	13.185
zoom	1	10.294	10.340
prefetch	1	10.269	32.538
stsg	1	8.110	8.103
zeropoint	1	8.056	8.047
gemv	1	7.969	0.000
voxelization	1	7.750	7.754
mcmd	1	7.640	7.663
kurtosis	1	7.615	7.629
metropolis	1	7.337	0.000
prna	1	6.257	6.329
ring	1	5.710	5.710
radixsort2	1	5.580	5.526
mixbench	1	5.522	5.522
zerocopy	1	5.265	45.559
bilateral	1	4.890	1.912
mcpr	1	4.760	4.758
nqueen	1	4.755	4.732
openmp	1	4.340	4.342
lebesgue	1	4.205	2.218
aop	1	3.945	4.522
sw4ck	1	3.899	3.876
attention	1	3.882	3.937
all-pairs-distance	1	3.561	3.544
mnist	1	3.512	3.576
interval	1	3.431	3.443
popcount	1	3.399	3.144
relu	1	3.369	0.000
wmma	1	3.319	0.000
grrt	1	3.272	3.041
deredundancy	1	3.235	3.235
perplexity	1	3.210	1.578
cm	1	3.184	3.181
reaction	1	3.160	3.165
pnpoly	1	3.013	3.013
streamCreateCopyDestroy	1	2.984	2.993
nlll	1	2.874	2.874
d3q19-bgk	1	2.853	2.863
streamOrderedAllocation	1	2.845	2.950
dwconv1d	1	2.843	2.860
qrg	1	2.801	2.801
epistasis	1	2.794	2.793

name	cuda/cuda	hip/cuda	hipified/cuda
iso2dfd	1	2.742	2.677
lsqt	1	2.739	0.000
degrid	1	2.727	2.869
scel	1	2.715	5.286
addBiasResidualLayerNorm	1	2.662	0.000
michalewicz	1	2.662	2.664
memcpy	1	2.612	2.643
mdh	1	2.488	0.000
medianfilter	1	2.464	2.511
surfel	1	2.434	1.958
mandelbrot	1	2.227	2.151
vote	1	2.196	0.000
tonemapping	1	2.080	2.080
myocyte	1	2.030	2.554
softmax-fused	1	2.020	0.000
morphology	1	2.017	2.027
su3	1	1.943	1.946
md5hash	1	1.941	1.941
atomicSystemWide	1	1.919	2.181
tridiagonal	1	1.877	1.913
merkle	1	1.866	1.914
jacobi	1	1.852	0.000
gru	1	1.831	1.832
word2vec	1	1.821	1.822
nbnxm	1	1.791	0.000
mis	1	1.784	1.778
matern	1	1.774	1.773
adv	1	1.758	1.784
car	1	1.721	1.721
rainflow	1	1.718	1.724
mallocFree	1	1.679	0.000
streamUM	1	1.670	1.634
match	1	1.644	1.644
mtf	1	1.634	1.630
tsp	1	1.616	2.146
cmembench	1	1.607	1.606
mriQ	1	1.560	6.399
pcc	1	1.545	1.547
tissue	1	1.530	1.529
cmp	1	1.529	1.531
maxFlops	1	1.520	1.520
heat	1	1.510	1.530
extend2	1	1.504	1.500
qem	1	1.480	1.532
p4	1	1.475	1.529
rle	1	1.427	0.000
cobahh	1	1.426	1.400
logan	1	1.422	1.455
blockAccess	1	1.415	1.411
randomAccess	1	1.407	1.392

表 3.2.11.4 ベンチマーク実行時間比較（2）

name	cuda/cuda	hip/cuda	hipified/cuda
ludb	1	1.404	0.000
loopback	1	1.394	1.391
memtest	1	1.390	1.390
hypterm	1	1.382	1.378
f16max	1	1.375	1.374
blas-fp8gemm	1	1.373	0.000
snake	1	1.370	1.371
bitcracker	1	1.362	1.364
convolutionSeparable	1	1.359	1.294
fsm	1	1.355	1.357
feynman-kac	1	1.351	1.365
wsm5	1	1.339	1.337
bonds	1	1.334	1.335
rayleighBenardConvection	1	1.331	1.440
wlcpow	1	1.315	1.264
kmeans	1	1.311	1.304
bezier-surface	1	1.295	1.295
streamcluster	1	1.294	1.519
jaccard	1	1.290	0.000
tpacf	1	1.284	0.000
knn	1	1.282	1.274
bm3d	1	1.278	1.261
quicksort	1	1.276	1.271
spaxpby	1	1.276	0.267
perlin	1	1.273	1.269
simpleMultiDevice	1	1.258	1.265
kalman	1	1.255	1.256
blas-gemmStridedBatched	1	1.234	0.000
vol2col	1	1.221	1.217
thomas	1	1.218	1.226
chacha20	1	1.217	1.215
hexciton	1	1.216	0.000
permutate	1	1.191	1.195
lulesh	1	1.185	1.042
kernelLaunch	1	1.179	1.189
ge-spmm	1	1.179	1.178
bitpermute	1	1.175	0.000
sssp	1	1.173	1.189
accuracy	1	1.169	1.168
vanGenuchten	1	1.153	1.151
xsbench	1	1.140	0.903
sparkler	1	1.139	1.137
minisweep	1	1.136	1.151
lid-driven-cavity	1	1.129	1.132
fluidSim	1	1.127	1.111
blas-dot	1	1.124	1.125
miniFE	1	1.122	1.146
pns	1	1.118	1.118
diamond	1	1.113	1.131
sosfil	1	1.107	1.105

name	cuda/cuda	hip/cuda	hipified/cuda
pool	1	1.106	1.148
ssim	1	1.103	1.101
blas-gemmBatched	1	1.087	1.088
merge	1	1.085	1.085
mmcsf	1	1.063	1.068
murmurhash3	1	1.055	1.063
doh	1	1.051	1.050
md	1	1.051	1.053
cross	1	1.050	1.050
boxfilter	1	1.047	1.047
segment-reduce	1	1.045	1.055
minmax	1	1.044	1.040
sampling	1	1.036	1.035
spm	1	1.036	1.039
sddmm-batch	1	1.025	1.021
gelu	1	1.022	1.008
testSNAP	1	1.014	0.991
nw	1	1.011	0.913
hotspot3D	1	1.010	1.012
page-rank	1	1.010	1.009
gd	1	1.003	2.245
clink	1	0.993	0.897
hbc	1	0.973	0.972
hpl	1	0.972	0.000
spsm	1	0.957	0.000
norm2	1	0.956	0.924
minkowski	1	0.953	1.248
ntt	1	0.946	0.944
hybridsort	1	0.942	0.940
hellinger	1	0.927	0.928
f16sp	1	0.914	55.424
dense-embedding	1	0.908	0.908
keccaktreehash	1	0.902	0.902
shuffle	1	0.894	0.000
crossEntropy	1	0.889	0.888
logic-resim	1	0.886	0.885
aidw	1	0.885	4.411
mt	1	0.880	0.884
logprob	1	0.875	0.000
srad	1	0.873	0.872
sort	1	0.864	0.863
pathfinder	1	0.864	0.865
zmddft	1	0.845	0.848
ldpc	1	0.845	0.845
background-subtract	1	0.831	0.889
halo-finder	1	0.827	1.428
kmc	1	0.827	0.831
shmembench	1	0.824	0.824
atomicCAS	1	0.821	0.848
bitonic-sort	1	0.811	0.809

表 3.2.11.5 ベンチマーク実行時間比較（3）

name	cuda/cuda	hip/cuda	hipified/cuda
rsbench	1	0.800	0.803
lr	1	0.791	0.789
channelSum	1	0.790	0.788
idivide	1	0.787	0.787
atomicIntrinsics	1	0.783	0.784
recursiveGaussian	1	0.780	0.782
channelShuffle	1	0.779	0.657
pitch	1	0.768	0.768
fresnel	1	0.768	0.000
asmooth	1	0.766	0.819
expdist	1	0.766	0.766
laplace	1	0.762	0.762
remap	1	0.756	0.760
particle-diffusion	1	0.749	0.748
overlay	1	0.738	0.738
minibude	1	0.733	0.739
qkv	1	0.729	0.726
extrema	1	0.723	0.725
atomicAggregate	1	0.722	0.000
linearprobing	1	0.719	0.717
bspline-vgh	1	0.714	0.714
ss	1	0.699	0.699
compute-score	1	0.693	0.693
segsort	1	0.681	0.000
split	1	0.681	0.579
sheath	1	0.679	66.950
softmax	1	0.675	0.000
present	1	0.670	0.670
threadfence	1	0.669	0.691
flame	1	0.669	0.674
softmax-online	1	0.667	0.000
allreduce	1	0.665	0.663
ne	1	0.652	0.654
nonzero	1	0.650	0.649
haccmk	1	0.650	0.650
affine	1	0.633	0.608
marchingCubes	1	0.629	0.000
permute	1	0.622	0.620
rtm8	1	0.621	0.642
wyllie	1	0.620	0.583
mrg32k3a	1	0.611	0.610
streamPriority	1	0.611	0.310
floydwarshall2	1	0.603	0.622
lavaMD	1	0.594	0.593
adam	1	0.593	0.592
multimaterial	1	0.593	0.598
asta	1	0.588	0.591
rodrigues	1	0.587	0.587
geodesic	1	0.585	0.583
spmm	1	0.582	0.587
radixsort	1	0.581	0.577
interleave	1	0.575	0.576
nosync	1	0.574	0.794
lif	1	0.570	0.570
eigenvalue	1	0.562	0.561
meanshift	1	0.559	0.599
hwt1d	1	0.554	0.552
hmm	1	0.549	0.553
miniWeather	1	0.541	0.536
mask	1	0.538	0.539
jenkins-hash	1	0.537	0.538
opticalFlow	1	0.535	0.000
langford	1	0.528	0.530
unfold	1	0.524	0.522
scan	1	0.523	0.529
sobol	1	0.522	0.522
che	1	0.519	0.519
tensorAccessor	1	0.510	0.492
copy	1	0.507	86.110
bitpacking	1	0.504	0.502
concat	1	0.499	0.495
henry	1	0.492	0.768
stddev	1	0.492	0.676
lud	1	0.490	0.491
minimod	1	0.490	0.491
cc	1	0.487	0.000
lanczos	1	0.484	0.472
stencil1d	1	0.482	0.489
ert	1	0.478	0.478
reverse	1	0.475	0.501
atomicPerf	1	0.471	823.926
ising	1	0.462	0.463
fpdc	1	0.462	0.459
ccsd-trpdrv	1	0.462	0.466
svd3x3	1	0.459	0.478
ans	1	0.457	0.478
sad	1	0.451	0.451
rowwiseMoments	1	0.449	0.448
hausdorff	1	0.440	0.440
sph	1	0.426	0.426
ced	1	0.424	0.421
heartwall	1	0.423	0.414
lci	1	0.421	0.421
layernorm	1	0.419	0.000
wordcount	1	0.419	0.419
black-scholes	1	0.416	0.414
romberg	1	0.414	0.410
lrn	1	0.408	0.412
reverse2D	1	0.399	1.421
keogh	1	0.396	0.396

表 3.2.11.6 ベンチマーク実行時間比較（4）

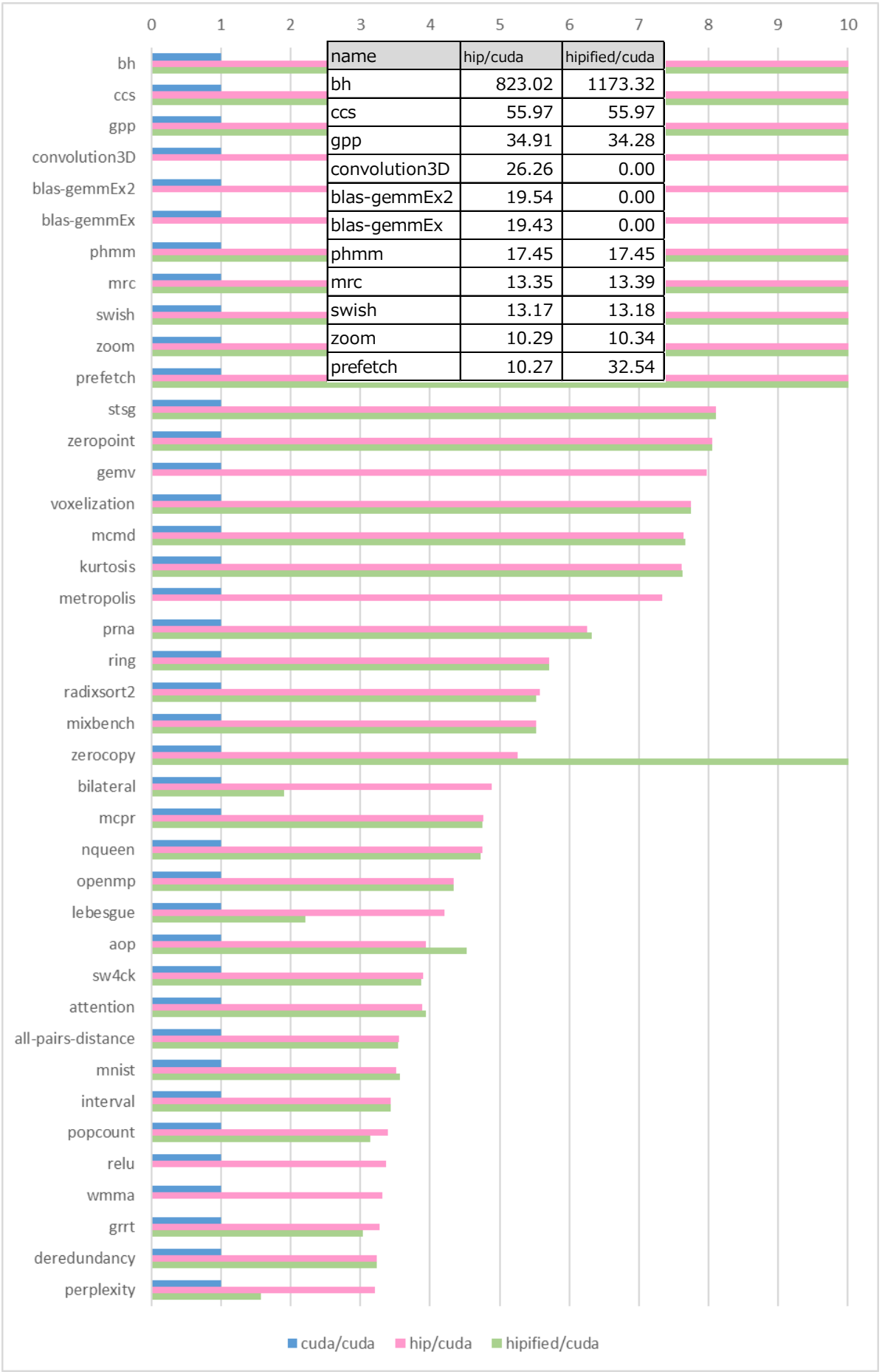
name	cuda/cuda	hip/cuda	hipified/cuda
particlefilter	1	0.395	0.392
vmc	1	0.394	0.396
score	1	0.393	0.000
maxpool3d	1	0.392	0.370
gels	1	0.392	0.000
tensorT	1	0.390	0.392
lda	1	0.386	0.386
aligned-types	1	0.379	0.379
tqs	1	0.374	0.379
chemv	1	0.374	0.374
matrixT	1	0.373	0.373
particles	1	0.372	0.372
bincount	1	0.367	0.367
binomial	1	0.362	0.362
blas-gemm	1	0.358	0.358
ace	1	0.352	0.349
easyWave	1	0.349	0.346
rng-wallace	1	0.337	0.335
fft	1	0.334	0.334
stencil3d	1	0.332	0.329
dslash	1	0.324	0.655
matrix-rotate	1	0.320	0.333
triad	1	0.319	0.319
d2q9-bgk	1	0.318	0.320
sptrsv	1	0.317	0.359
cfd	1	0.313	0.317
daphne	1	0.312	0.316
spsort	1	0.311	0.313
dispatch	1	0.311	0.317
is	1	0.310	0.312
attentionMultiHead	1	0.310	0.000
rotary	1	0.309	0.300
fwt	1	0.299	0.297
pso	1	0.295	0.291
projectile	1	0.293	0.229
dwconv	1	0.286	0.284
dp	1	0.285	0.284
haversine	1	0.282	0.317
graphExecution	1	0.279	0.279
clenergy	1	0.277	0.277
secp256k1	1	0.277	0.277
multinomial	1	0.276	0.278
sobel	1	0.275	0.288
layout	1	0.273	0.273
spgeam	1	0.273	0.273
gaussian	1	0.271	0.273
libor	1	0.269	0.269
scan2	1	0.268	0.270
mr	1	0.264	0.264
spnnz	1	0.263	0.262

name	cuda/cuda	hip/cuda	hipified/cuda
ddbp	1	0.263	0.263
collision	1	0.261	0.000
divergence	1	0.252	0.252
scan3	1	0.251	0.246
lombscargle	1	0.244	0.244
leukocyte	1	0.244	0.214
spgemm	1	0.230	0.229
bfs	1	0.230	0.220
spmv	1	0.228	0.225
sps2d	1	0.228	0.226
face	1	0.224	0.224
spd2s	1	0.224	0.226
sss	1	0.210	0.210
warpsort	1	0.207	0.000
babelstream	1	0.205	0.205
winograd	1	0.204	0.204
b+tree	1	0.199	0.199
hogbom	1	0.198	0.179
rsc	1	0.198	0.210
p2p	1	0.196	0.194
laplace3d	1	0.196	0.187
burger	1	0.193	0.193
bscan	1	0.187	0.046
log2	1	0.185	0.187
atomicReduction	1	0.181	0.181
entropy	1	0.180	0.180
axhelm	1	0.178	0.178
lfib4	1	0.167	0.151
amgmk	1	0.166	0.163
hungarian	1	0.163	0.161
geam	1	0.163	0.163
atan2	1	0.160	0.162
gibbs	1	0.157	0.155
resnet-kernels	1	0.151	0.039
complex	1	0.146	0.145
dxtc1	1	0.146	0.150
langevin	1	0.144	0.167
si	1	0.142	0.000
determinant	1	0.142	0.142
nms	1	0.140	0.137
fdtd3d	1	0.138	0.131
aes	1	0.137	0.135
cbsfil	1	0.136	0.137
conversion	1	0.134	0.134
overlap	1	0.134	0.000
intrinsics-cast	1	0.132	0.133
warpexchange	1	0.132	0.115
inversek2j	1	0.131	0.131
hotspot	1	0.127	0.125
bsw	1	0.126	0.000

表 3.2.11.7 ベンチマーク実行時間比較（5）

name	cuda/cuda	hip/cuda	hipified/cuda
histogram	1	0.125	0.128
simplemoc	1	0.123	0.119
flip	1	0.120	0.120
distort	1	0.115	0.113
seam-carving	1	0.114	0.114
xlqc	1	0.114	0.111
nn	1	0.112	0.110
concurrentKernels	1	0.112	0.112
sortKV	1	0.110	0.109
dxtc2	1	0.109	0.109
aobench	1	0.107	0.107
urng	1	0.107	0.107
grep	1	0.105	0.105
backprop	1	0.102	0.102
clock	1	0.098	0.094
s3d	1	0.098	0.095
ga	1	0.095	0.095
chi2	1	0.091	0.091
filter	1	0.086	0.086
goulash	1	0.081	0.082
atomicCost	1	0.081	0.081
ecdh	1	0.080	0.080
pointwise	1	0.078	0.122
wedford	1	0.077	0.000
frechet	1	0.073	0.073
bwt	1	0.064	0.065
convolution1D	1	0.059	0.059
debayer	1	0.058	0.068
eikonal	1	0.051	0.048
colorwheel	1	0.048	0.048
mpc	1	0.044	0.000
gabor	1	0.041	0.066
damage	1	0.040	0.040
resize	1	0.037	0.037
rfs	1	0.037	0.037
sa	1	0.030	0.030
dropout	1	0.030	0.030
depixel	1	0.027	0.027
crc64	1	0.025	0.025
qtclustering	1	0.025	0.025
floydwarshall	1	0.024	0.025
s8n	1	0.024	0.024
dpid	1	0.020	0.000
fpc	1	0.020	0.020
blockexchange	1	0.020	0.020
rushlarsen	1	0.016	0.016
nbody	1	0.011	0.011
contract	1	0.010	0.010
gamma-correction	1	0.009	0.009
dct8x8	1	0.008	0.008

name	cuda/cuda	hip/cuda	hipified/cuda
crs	1	0.008	0.008
pad	1	0.007	0.000
coordinates	1	0.007	0.007
gmm	1	0.006	0.003
bsearch	1	0.006	0.006
simpleSpmv	1	0.006	0.000
tsa	1	0.002	0.001
fhd	1	0.001	0.001
cooling	1	0.000	0.001
assert	1	0.000	0.000
bicgstab	1	0.000	117.647
bn	1	0.000	0.000
btree	1	0.000	0.000
ccl	1	0.000	0.000
columnarSolver	1	0.000	0.000
egs	1	0.000	0.000
gc	1	0.000	0.000
glu	1	0.000	0.757
graphB+	1	0.000	0.000
intrinsics-simd	1	0.000	0.000
logic-rewrite	1	0.000	0.000
minimap2	1	0.000	0.337
pingpong	1	0.000	0.000
rsmt	1	0.000	0.000
slit	1	0.000	0.000



name	hip/cuda	hipified/cuda
bh	823.02	1173.32
ccs	55.97	55.97
gpp	34.91	34.28
convolution3D	26.26	0.00
blas-gemmEx2	19.54	0.00
blas-gemmEx	19.43	0.00
phmm	17.45	17.45
mrc	13.35	13.39
swish	13.17	13.18
zoom	10.29	10.34
prefetch	10.27	32.54

図 3.2.11.1 ベンチマーク実行時間比較（1）

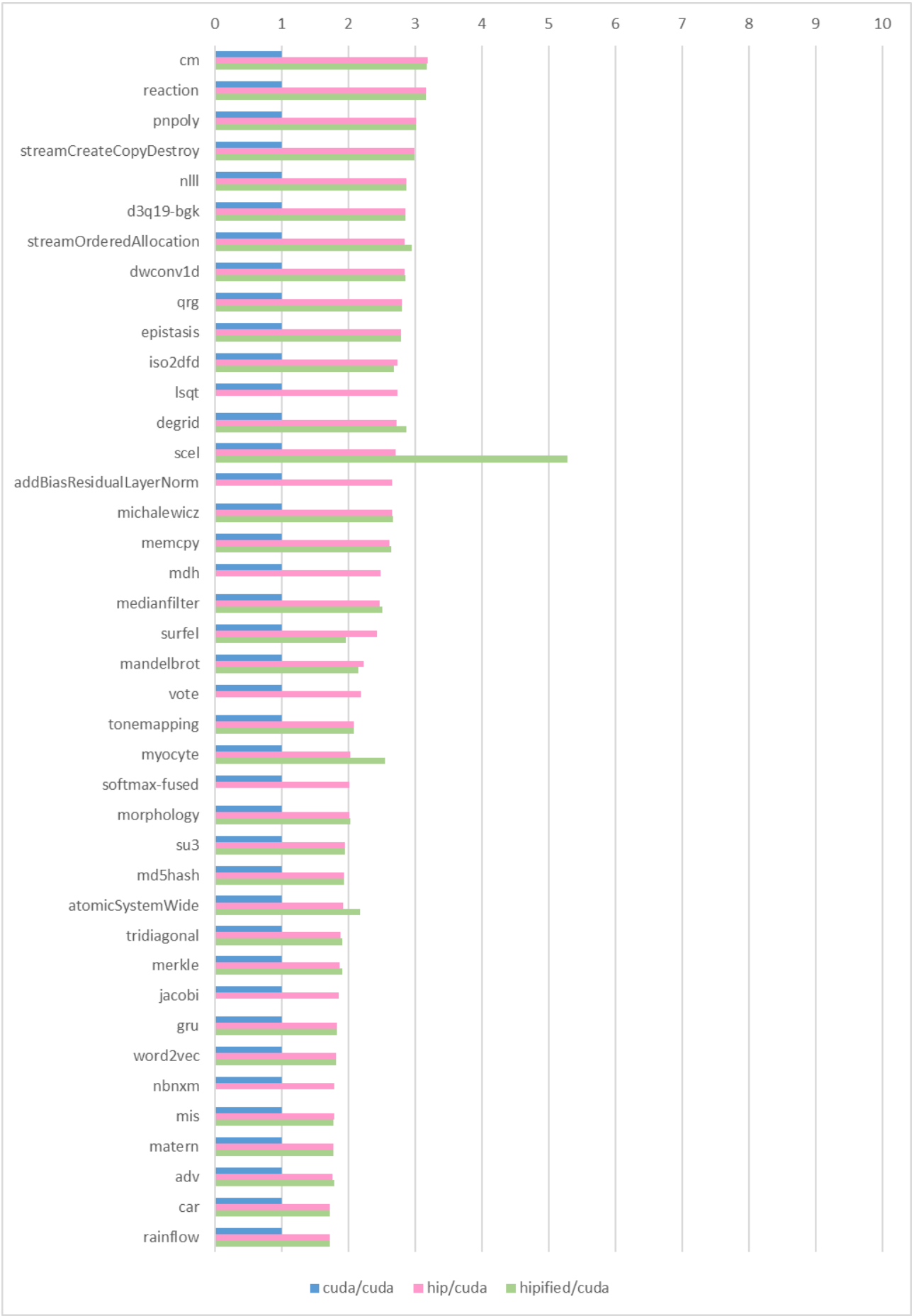


図 3.2.11.2 ベンチマーク実行時間比較（2）

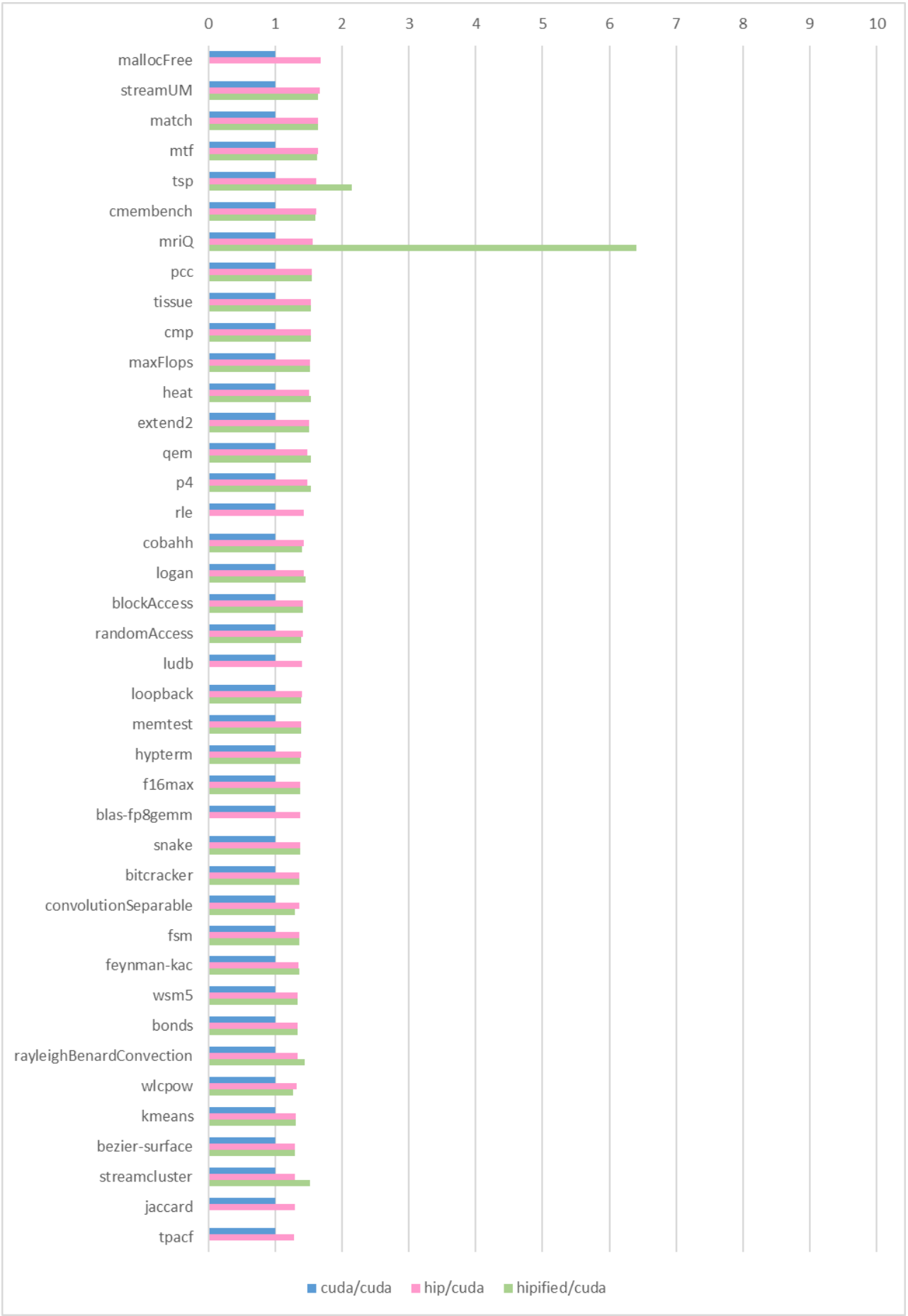


図 3.2.11.3 ベンチマーク実行時間比較（3）

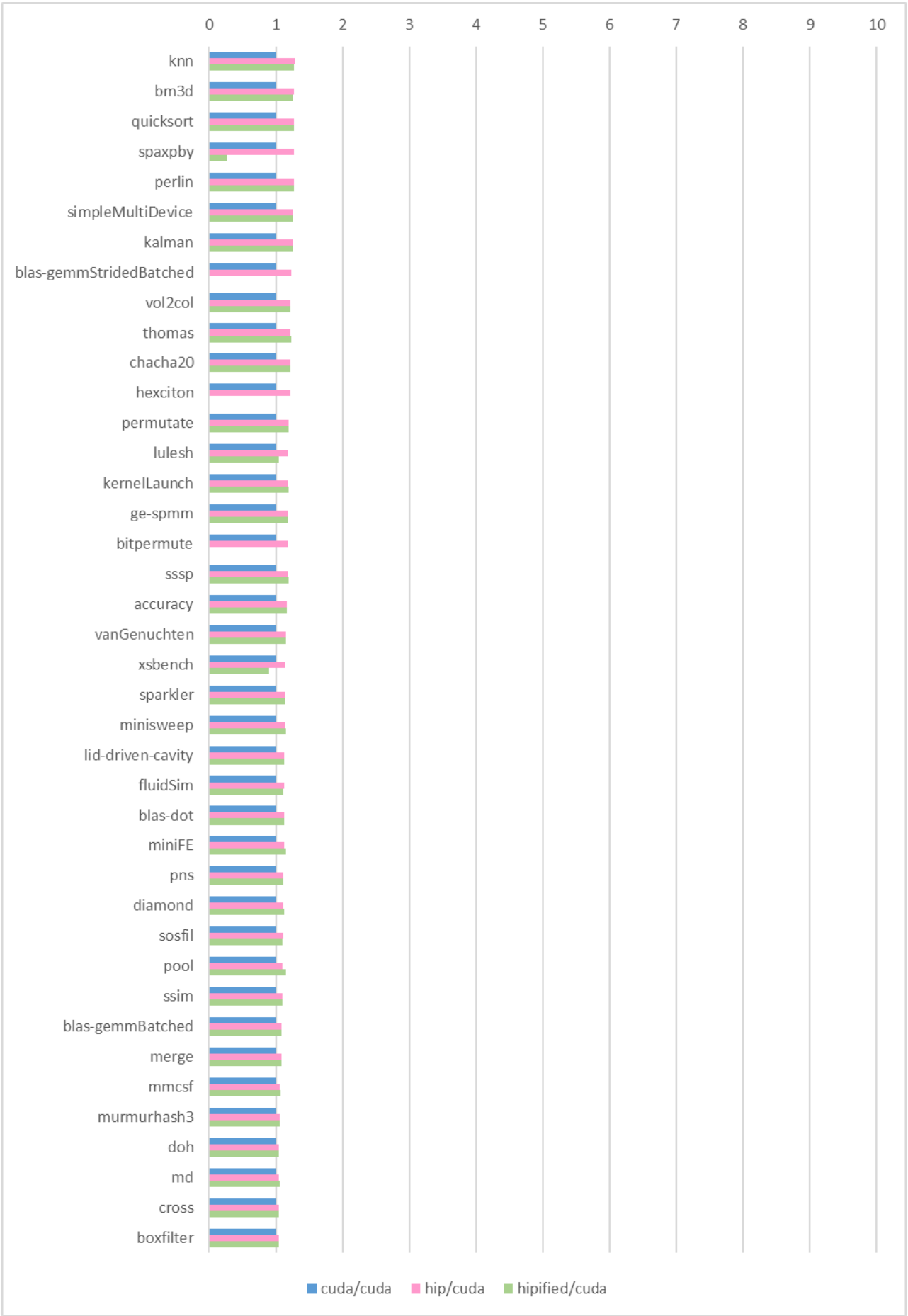


図 3.2.11.4 ベンチマーク実行時間比較（4）

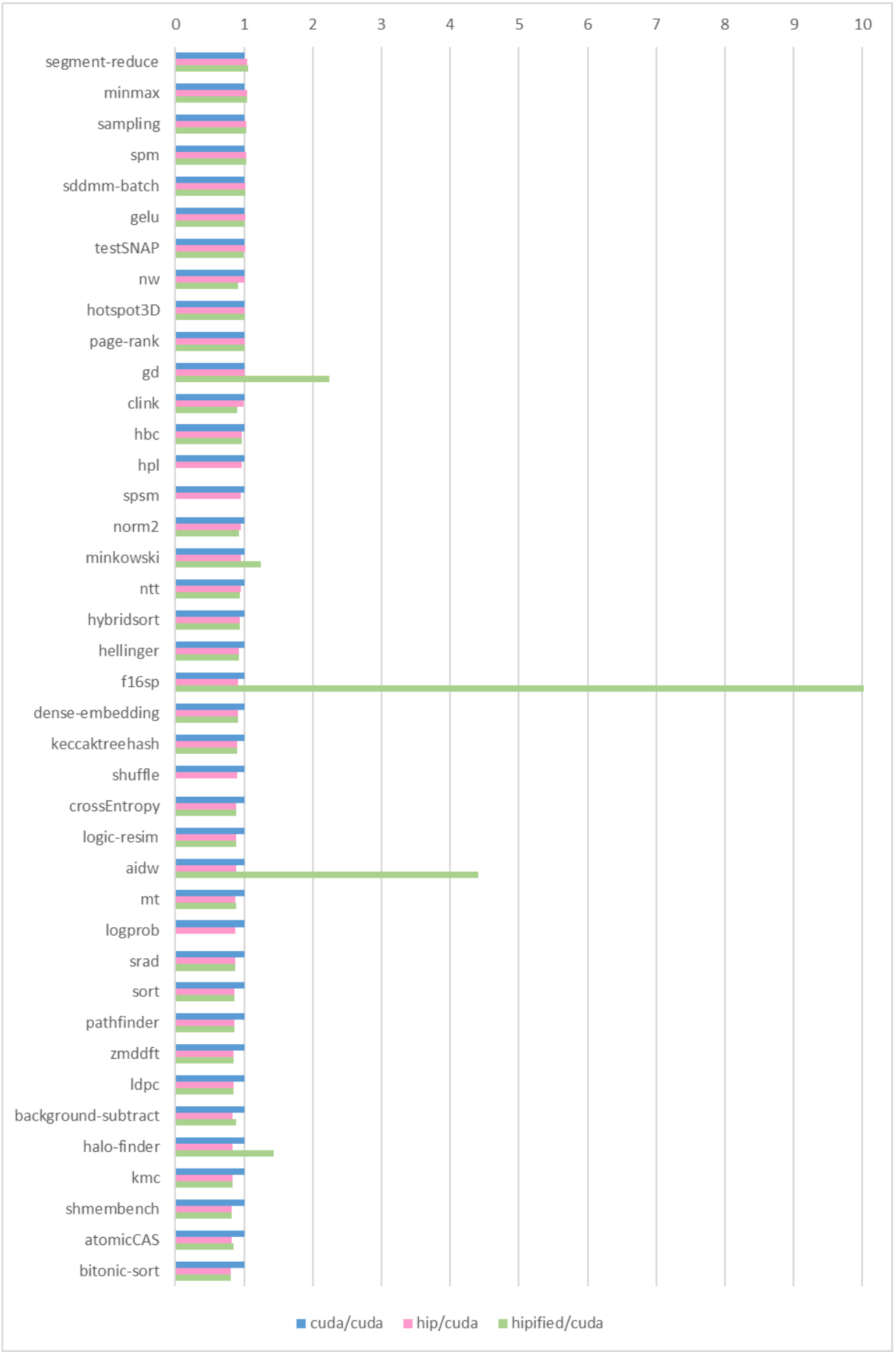


図 3.2.11.5 ベンチマーク実行時間比較（5）

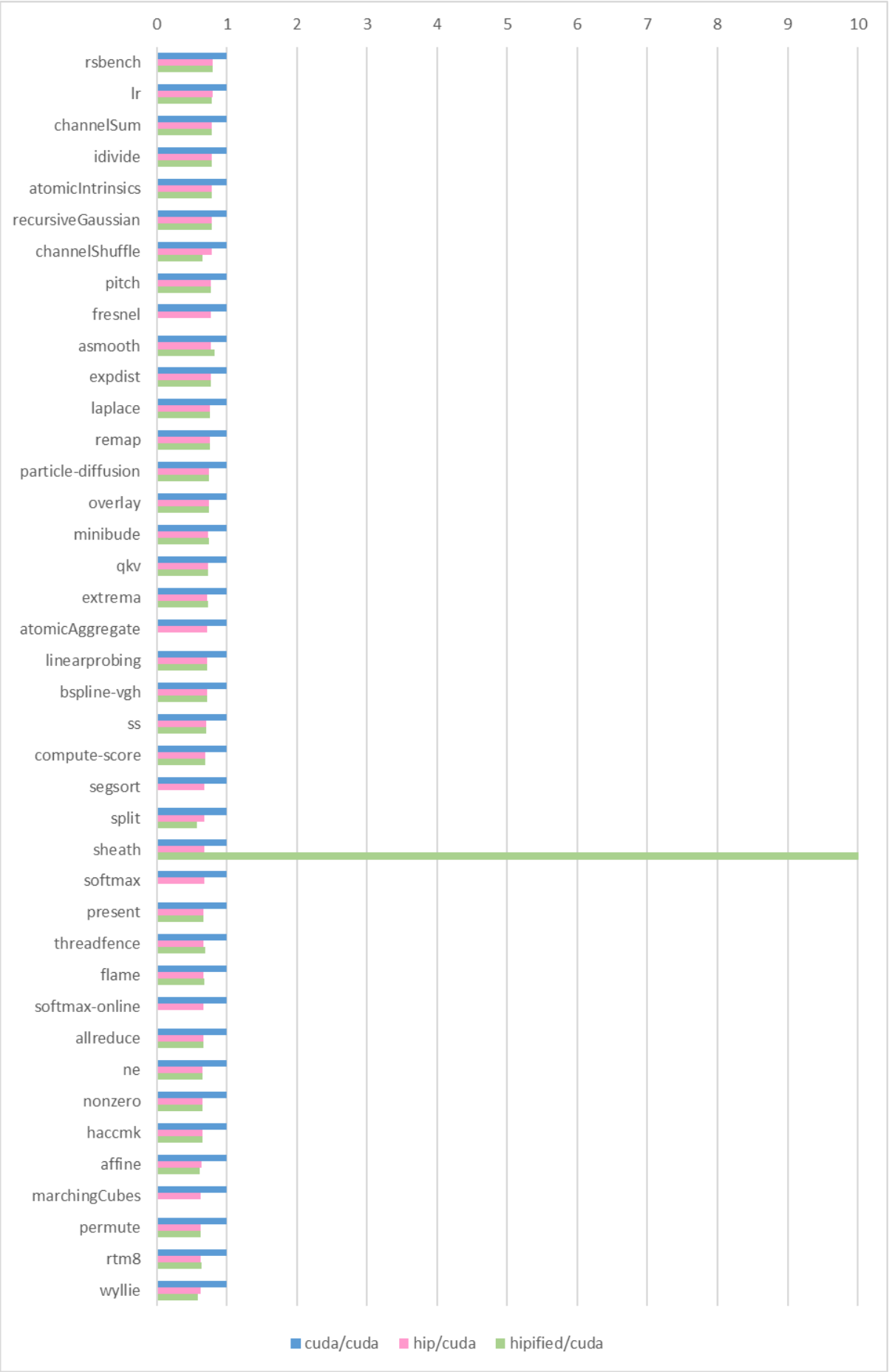


図 3.2.11.6 ベンチマーク実行時間比較（6）

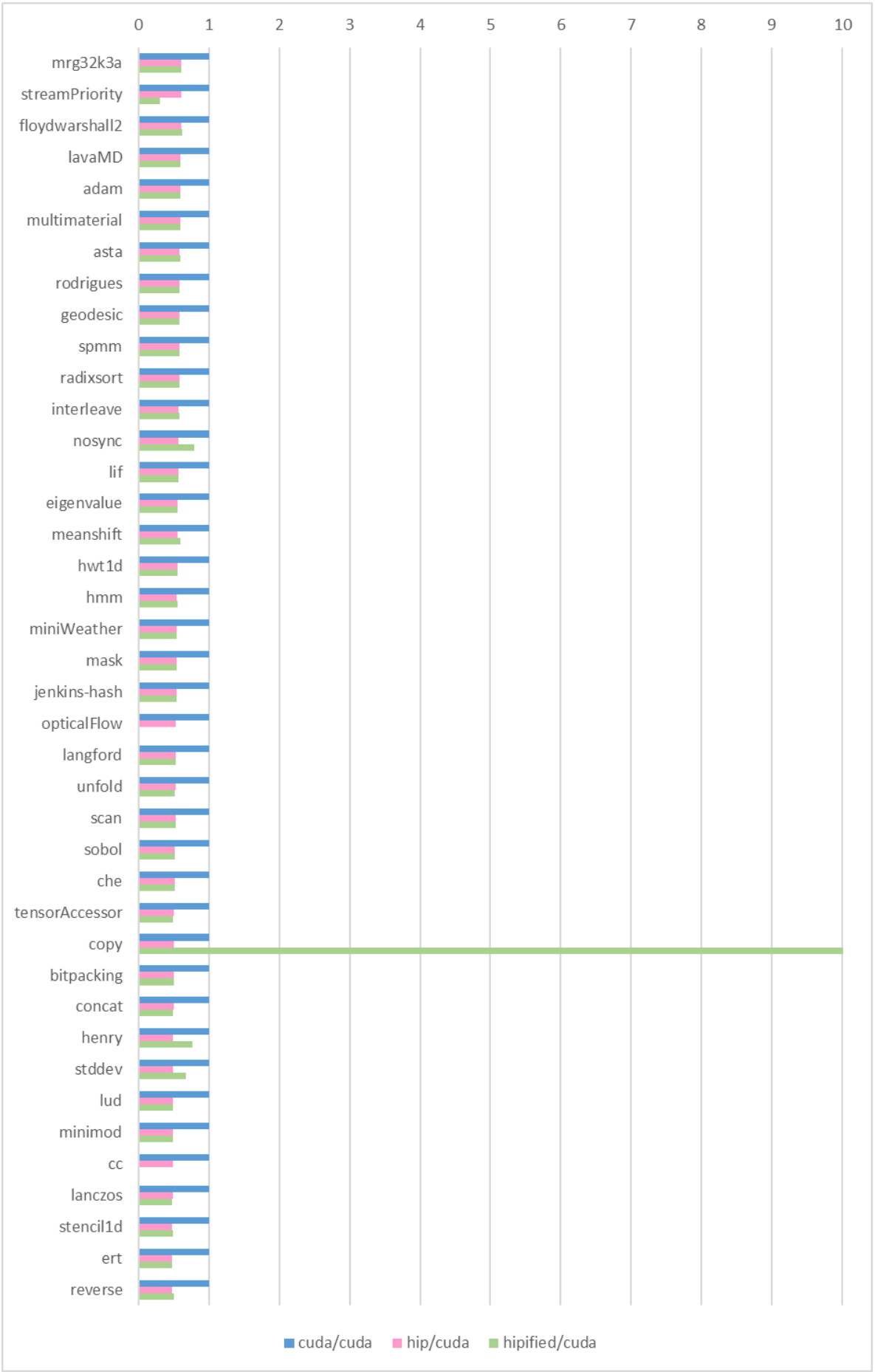


図 3.2.11.7 ベンチマーク実行時間比較（7）

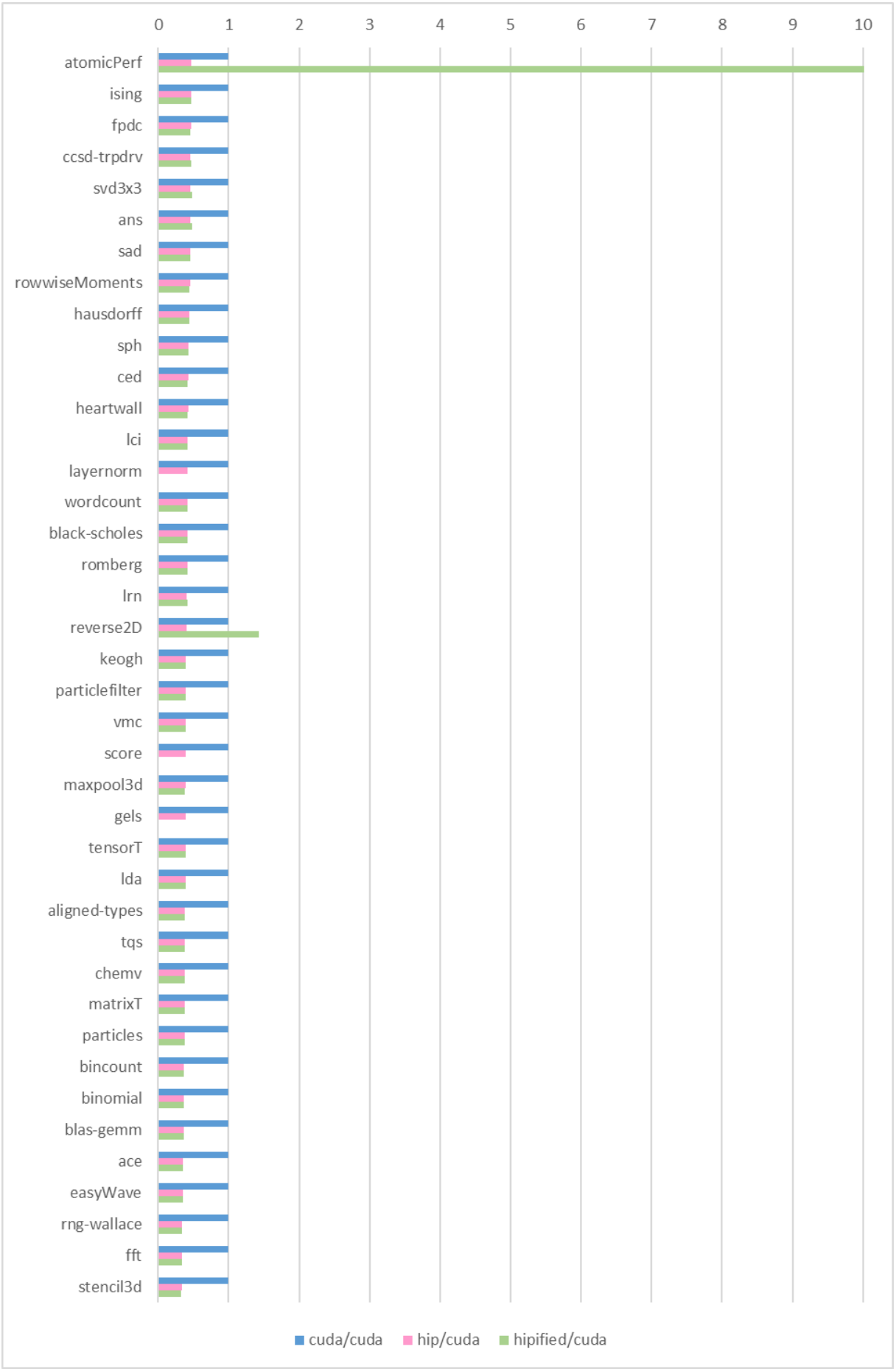


図 3.2.11.8 ベンチマーク実行時間比較（8）

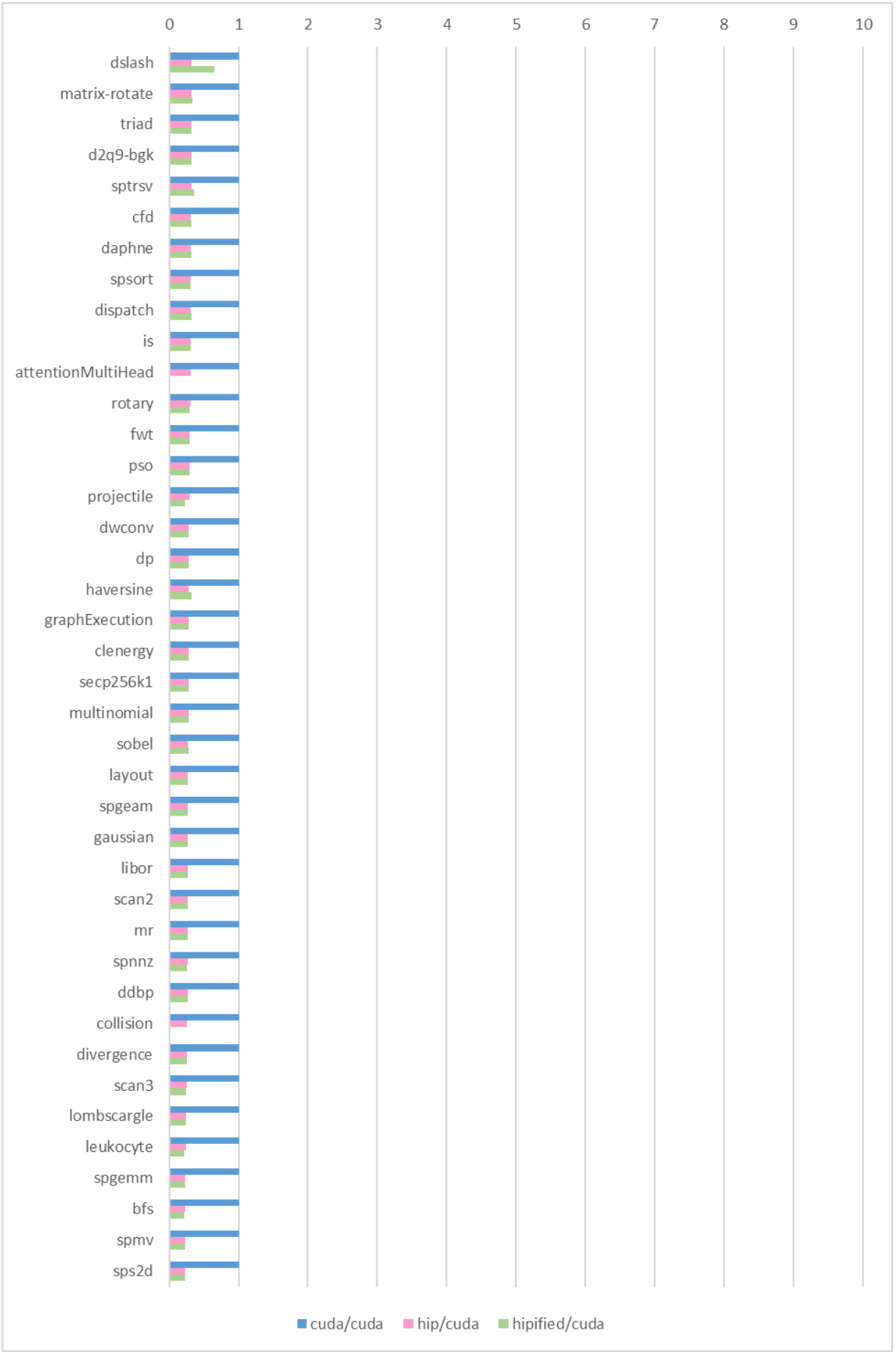


図 3.2.11.9 ベンチマーク実行時間比較（9）

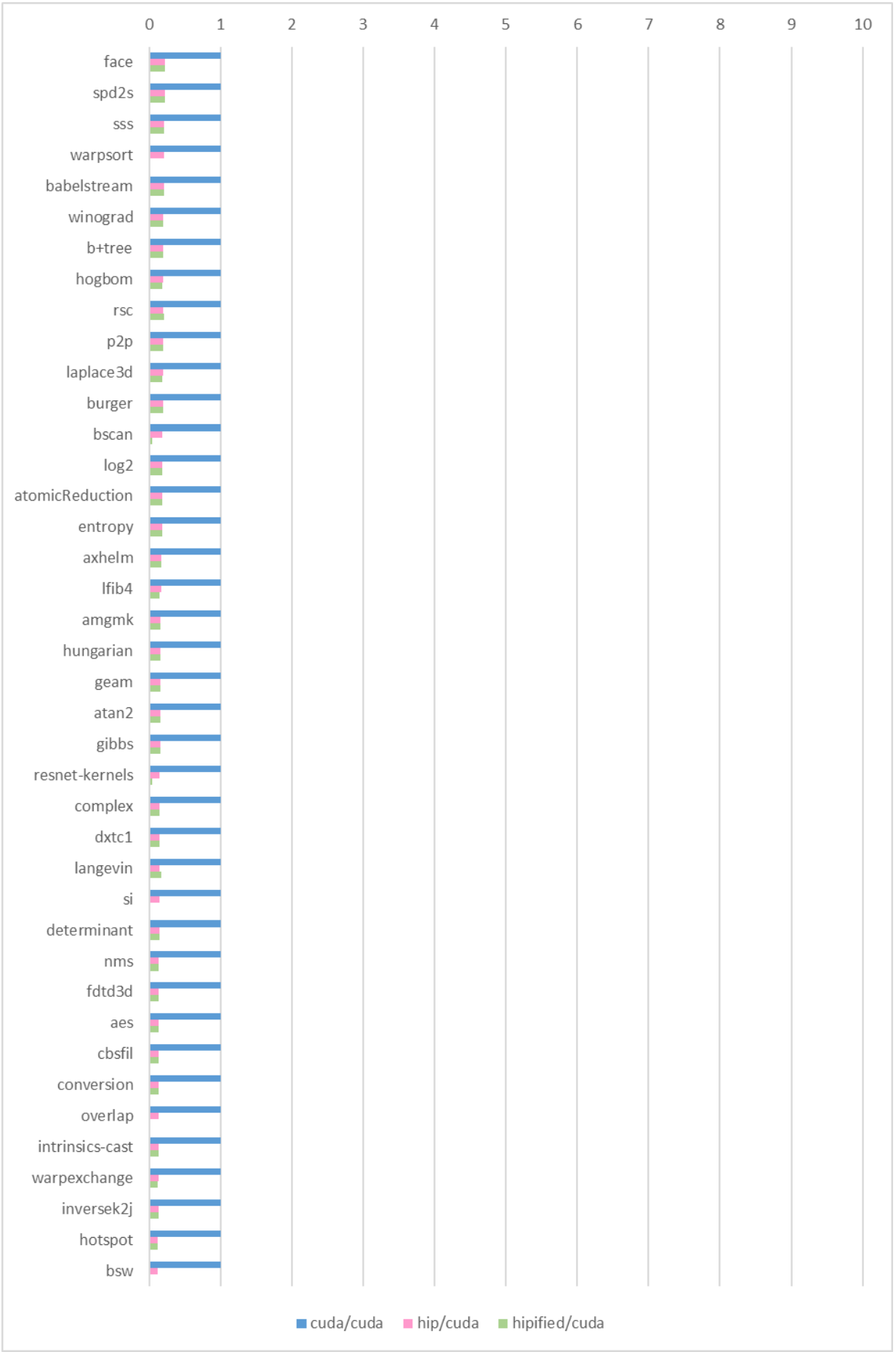


図 3.2.11.10 ベンチマーク実行時間比較（10）

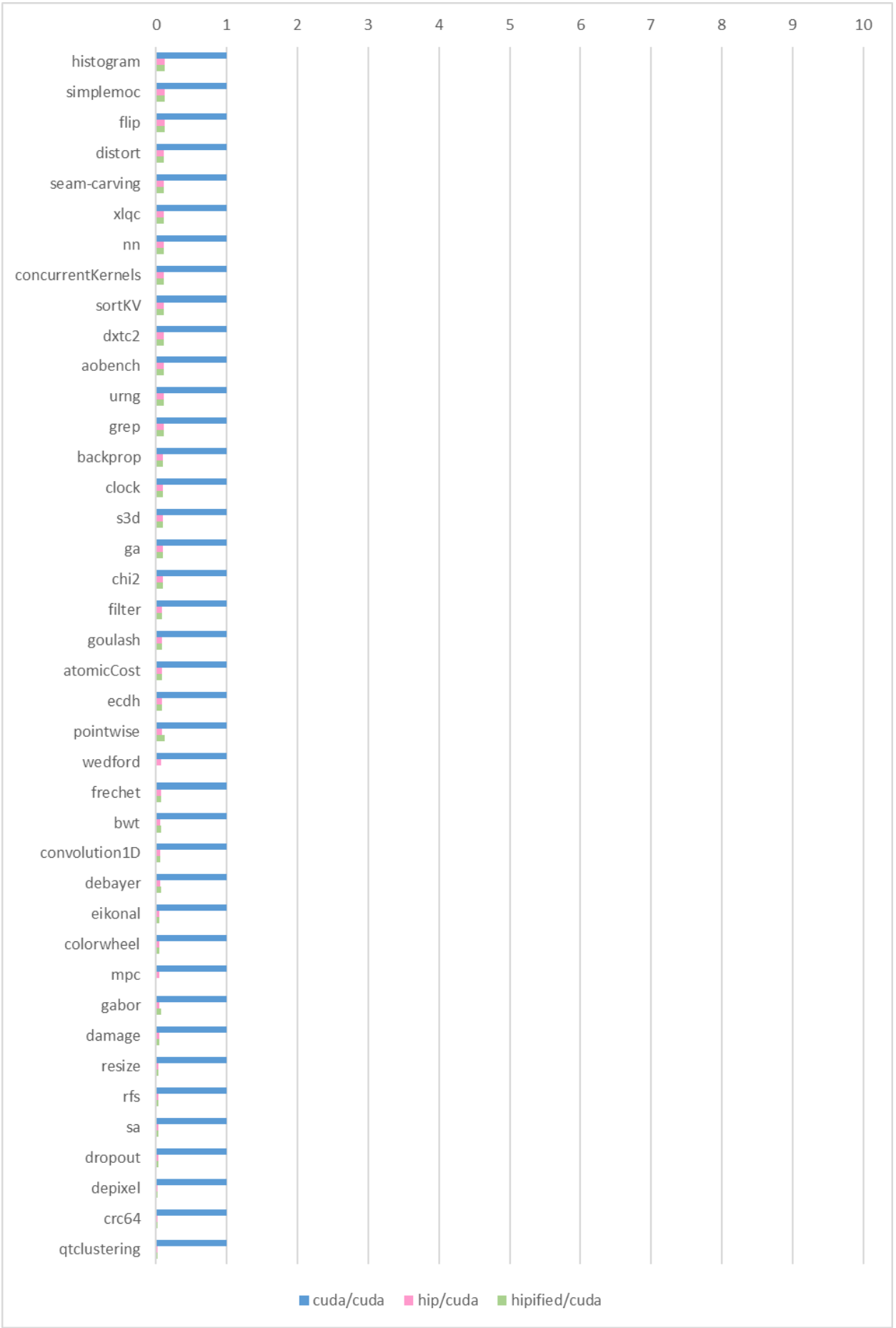


図 3.2.11.11　ベンチマーク実行時間比較（11）

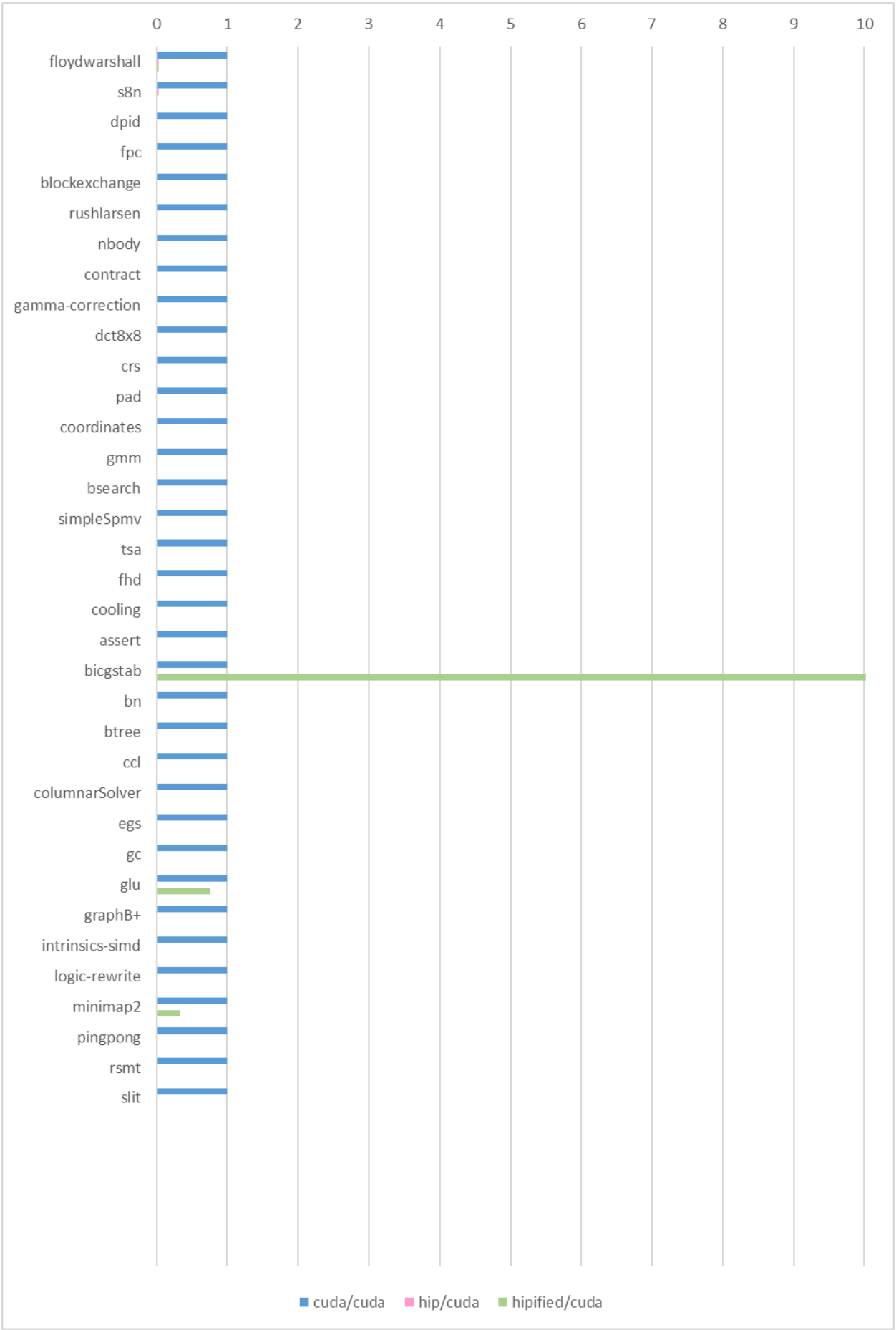


図 3.2.11.12　ベンチマーク実行時間比較（12）

OpenMP版をCUDAにおいて実行した版(omp_nvc)と、OpenMP版をHIPにおいて実行した版(omp_aomp)の実行時間を比較するため、omp_nvc での実行時間を1とした場合の、omp_aomp の相対実行時間を表及びグラフで示す。このため、omp_nvcで実行できなかったベンチマークに含めない。omp_aompで実行できなかった場合、実行時間は0とした。

傾向を比較するため、実行時間が10倍超、10倍～2倍、2倍～1.2倍、1.2倍～0.8倍、0.8倍～0.5倍、0.5倍～0.1倍、0.1倍未満となる数を下表 3.2.11.8で示す（0となる場合は0.1倍未満と別にカウント）。

表 3.2.11.8 実行時間比の分類

	10～	10～2	2～1.2	1.2～0.8	0.8～0.5	0.5～0.1	0.1～	0	total
omp_aomp/omp_nvc	9	46	33	36	35	65	19	12	265

これにより、omp_aompがomp_nvcより2倍～10倍実行速度が速い（実行時間が短い）ものが最も多いことがわかった。今後は異なる環境・問題設定で評価を続ける予定である。

各値を表及びグラフにしたものを示す（表 3.2.11.9～表 3.2.11.11、図3.2.11.13～図3.2.11.19）。値はomp_aomp/omp_nvc の比の値が大きい順に並べた。(全265項目)

なお、グラフの横軸の最大値を「10倍」としたため、10倍以上となっている主なベンチマーク（mis ～ miniWeatherまでの9項目）については、グラフ中に表を挿入して値を示した。

（表・グラフは次ページより）

表 3.2.11.9 ベンチマーク実行時間比較（1）

name	omp_nvc/omp_nvc	omp_aomp/omp_nvc
mis	1	145.278
ccs	1	55.970
bilateral	1	54.259
ss	1	46.400
sheath	1	43.377
convolution3D	1	41.151
lulesh	1	40.219
atomicPerf	1	30.294
miniWeather	1	10.224
aidw	1	7.669
mcmd	1	6.799
mriQ	1	6.073
lebesgue	1	5.471
mdh	1	5.422
mixbench	1	5.387
epistasis	1	5.195
simplemoc	1	5.152
murmurhash3	1	5.117
degrid	1	5.077
openmp	1	5.050
popcount	1	4.976
mcpr	1	4.708
car	1	4.305
gpp	1	4.290
channelSum	1	4.229
xsbench	1	3.980
minkowski	1	3.766
nqueen	1	3.349
clenergy	1	3.342
multimaterial	1	3.250
interval	1	3.190
doh	1	3.176
boxfilter	1	3.152
qrg	1	2.963
iso2dfd	1	2.905
che	1	2.851
minimap2	1	2.781
hexciton	1	2.614
perplexity	1	2.590
softmax	1	2.553
md5hash	1	2.404
deredundancy	1	2.399
reaction	1	2.382
mandelbrot	1	2.369
laplace	1	2.289
tonemapping	1	2.242
affine	1	2.241
concat	1	2.203
extrema	1	2.202
memcpy	1	2.155
gd	1	2.093
inversek2j	1	2.066
surfel	1	2.063
phmm	1	2.026
lsqt	1	2.018
feynman-kac	1	1.942
overlay	1	1.925
pnpoly	1	1.906
lid-driven-cavity	1	1.852
lci	1	1.789
rainflow	1	1.778
asmooth	1	1.771
memtest	1	1.770
match	1	1.741
keogh	1	1.740
complex	1	1.712
kmeans	1	1.639
histogram	1	1.581
cobahh	1	1.577
maxFlops	1	1.549
tensorT	1	1.537
cmp	1	1.536
lr	1	1.448
hotspot3D	1	1.419
ldpc	1	1.406
bezier-surface	1	1.391
kernelLaunch	1	1.387
fluidSim	1	1.381
pool	1	1.370
heat2d	1	1.326
reverse	1	1.319
tissue	1	1.316
snake	1	1.305
bitonic-sort	1	1.290
knn	1	1.270
tridiagonal	1	1.245
permutate	1	1.241
sobel	1	1.202
thomas	1	1.199
ntt	1	1.140
spm	1	1.138
bspline-vgh	1	1.132
merge	1	1.118
channelShuffle	1	1.108
jacobi	1	1.100
hypterm	1	1.088
loopback	1	1.076
wsm5	1	1.073
black-scholes	1	1.069
ecdh	1	1.061

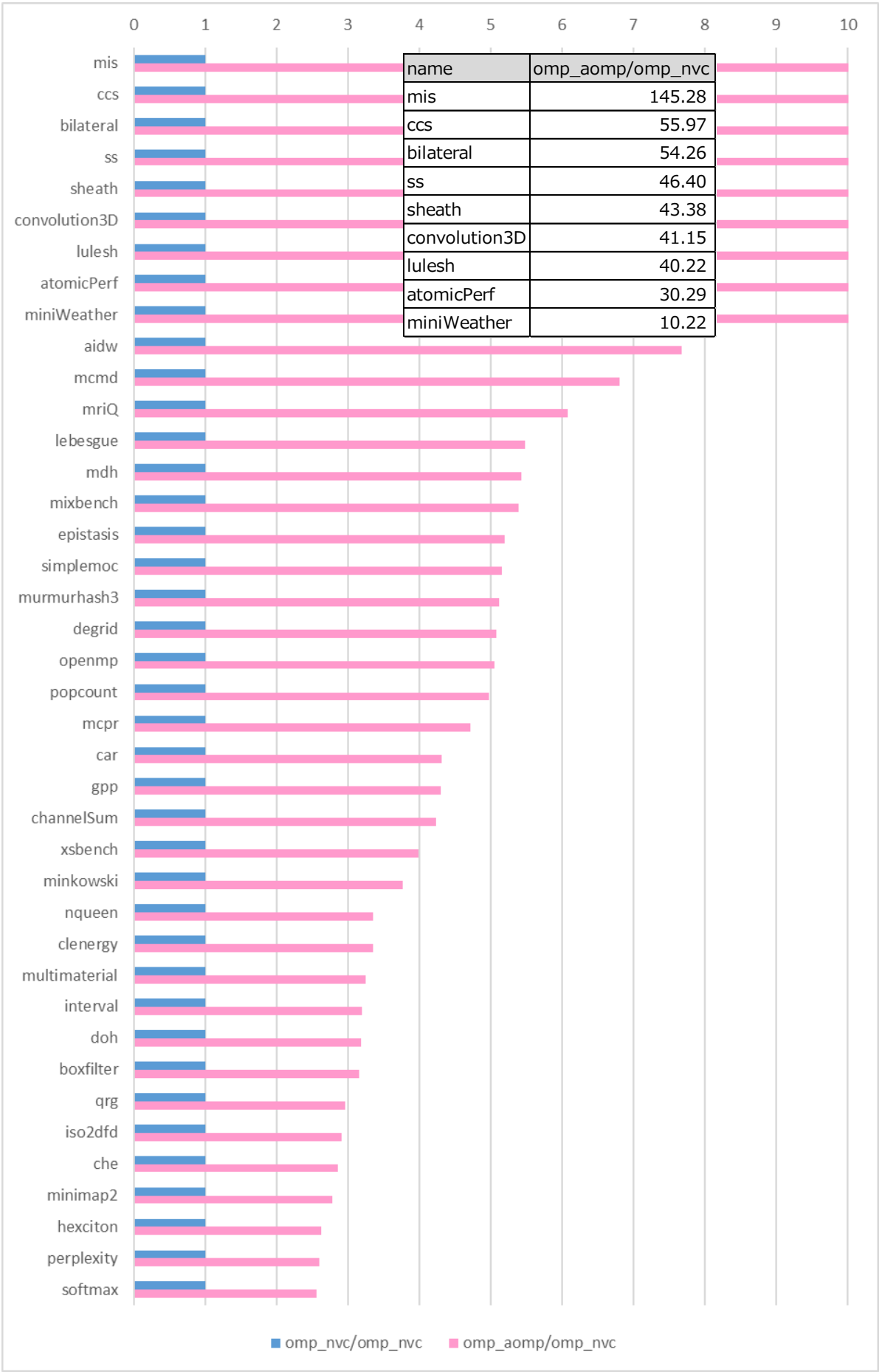
表 3.2.11.10 ベンチマーク実行時間比較（2）

name	omp_nvc/omp_nvc	omp_aomp/omp_nvc
myocyte	1	1.060
particle-diffusion	1	1.054
gaussian	1	1.054
miniFE	1	1.020
norm2	1	1.018
md	1	0.993
page-rank	1	0.961
mt	1	0.959
divergence	1	0.956
matern	1	0.952
hellinger	1	0.949
dp	1	0.931
bonds	1	0.928
su3	1	0.924
projectile	1	0.915
hybridsort	1	0.891
d2q9-bgk	1	0.877
metropolis	1	0.874
clink	1	0.856
keccaktreehash	1	0.852
haccmk	1	0.848
convolutionSeparable	1	0.843
ne	1	0.828
chacha20	1	0.826
henry	1	0.767
heat	1	0.760
rtm8	1	0.745
vanGenuchten	1	0.745
eigenvalue	1	0.742
scan	1	0.740
stddev	1	0.732
winograd	1	0.728
rodrigues	1	0.725
chi2	1	0.707
ising	1	0.690
sosfil	1	0.669
babelstream	1	0.669
present	1	0.665
interleave	1	0.640
fwt	1	0.633
stencil1d	1	0.632
triad	1	0.631
background-subtract	1	0.626
geodesic	1	0.626
nw	1	0.621
radixsort	1	0.608
medianfilter	1	0.591
jenkins-hash	1	0.585
wyllie	1	0.552
minibude	1	0.551
hwt1d	1	0.548
adv	1	0.540
ddbp	1	0.540
cfd	1	0.538
lanczos	1	0.535
lif	1	0.535
wlcpow	1	0.517
sph	1	0.516
split	1	0.505
lrn	1	0.499
aligned-types	1	0.495
sobol	1	0.486
flip	1	0.477
hausdorff	1	0.461
svd3x3	1	0.458
scan2	1	0.455
meanshift	1	0.442
sptrsv	1	0.440
easyWave	1	0.437
pso	1	0.431
hmm	1	0.417
wordcount	1	0.408
dense-embedding	1	0.406
laplace3d	1	0.400
mask	1	0.386
maxpool3d	1	0.381
s3d	1	0.374
atomicReduction	1	0.362
attention	1	0.350
srad	1	0.350
randomAccess	1	0.346
lud	1	0.343
expdist	1	0.339
burger	1	0.338
layout	1	0.335
libor	1	0.335
entropy	1	0.333
distort	1	0.333
matrix-rotate	1	0.327
haversine	1	0.321
rng-wallace	1	0.311
dslash	1	0.303
lavaMD	1	0.288
conversion	1	0.273
atomicCost	1	0.260
sampling	1	0.243
lombscargle	1	0.242
compute-score	1	0.228
langevin	1	0.227
bfs	1	0.221

表 3.2.11.11 ベンチマーク実行時間比較（3）

name	omp_nvc/omp_nvc	omp_aomp/omp_nvc
romberg	1	0.206
aobench	1	0.201
atomicIntrinsics	1	0.197
cbsfil	1	0.196
rsc	1	0.194
b+tree	1	0.191
hogbom	1	0.190
leukocyte	1	0.189
urng	1	0.181
axhelm	1	0.179
amgmk	1	0.177
nn	1	0.174
atan2	1	0.168
zmddft	1	0.164
all-pairs-distance	1	0.163
ga	1	0.162
accuracy	1	0.160
mallocFree	1	0.156
goulash	1	0.136
swish	1	0.125
extend2	1	0.116
xlqc	1	0.116
zeropoint	1	0.106
glu	1	0.103
dxtc2	1	0.094
fdtd3d	1	0.089
testSNAP	1	0.082
aes	1	0.082
colorwheel	1	0.081
scel	1	0.065
fft	1	0.065
ccsd-trpdrv	1	0.065
grep	1	0.064
floydwarshall	1	0.058
backprop	1	0.051
convolution1D	1	0.050
s8n	1	0.045
filter	1	0.043
secp256k1	1	0.041
crc64	1	0.027

name	omp_nvc/omp_nvc	omp_aomp/omp_nvc
contract	1	0.017
rushlarsen	1	0.016
cross	1	0.016
nbody	1	0.011
gmm	1	0.011
cooling	1	0.008
sad	1	0.007
p4	1	0.005
gamma-correction	1	0.005
simpleSpmv	1	0.005
damage	1	0.004
depixel	1	0.002
tsa	1	0.001
ace	1	0.000
adam	1	0.000
aop	1	0.000
asta	1	0.000
chemv	1	0.000
columnarSolver	1	0.000
diamond	1	0.000
grrt	1	0.000
idivide	1	0.000
pointwise	1	0.000
sort	1	0.000
vol2col	1	0.000



name	omp_aomp/omp_nvc
mis	145.28
ccs	55.97
bilateral	54.26
ss	46.40
sheath	43.38
convolution3D	41.15
lulesh	40.22
atomicPerf	30.29
miniWeather	10.22

図 3.2.11.13 ベンチマーク実行時間比較（1）

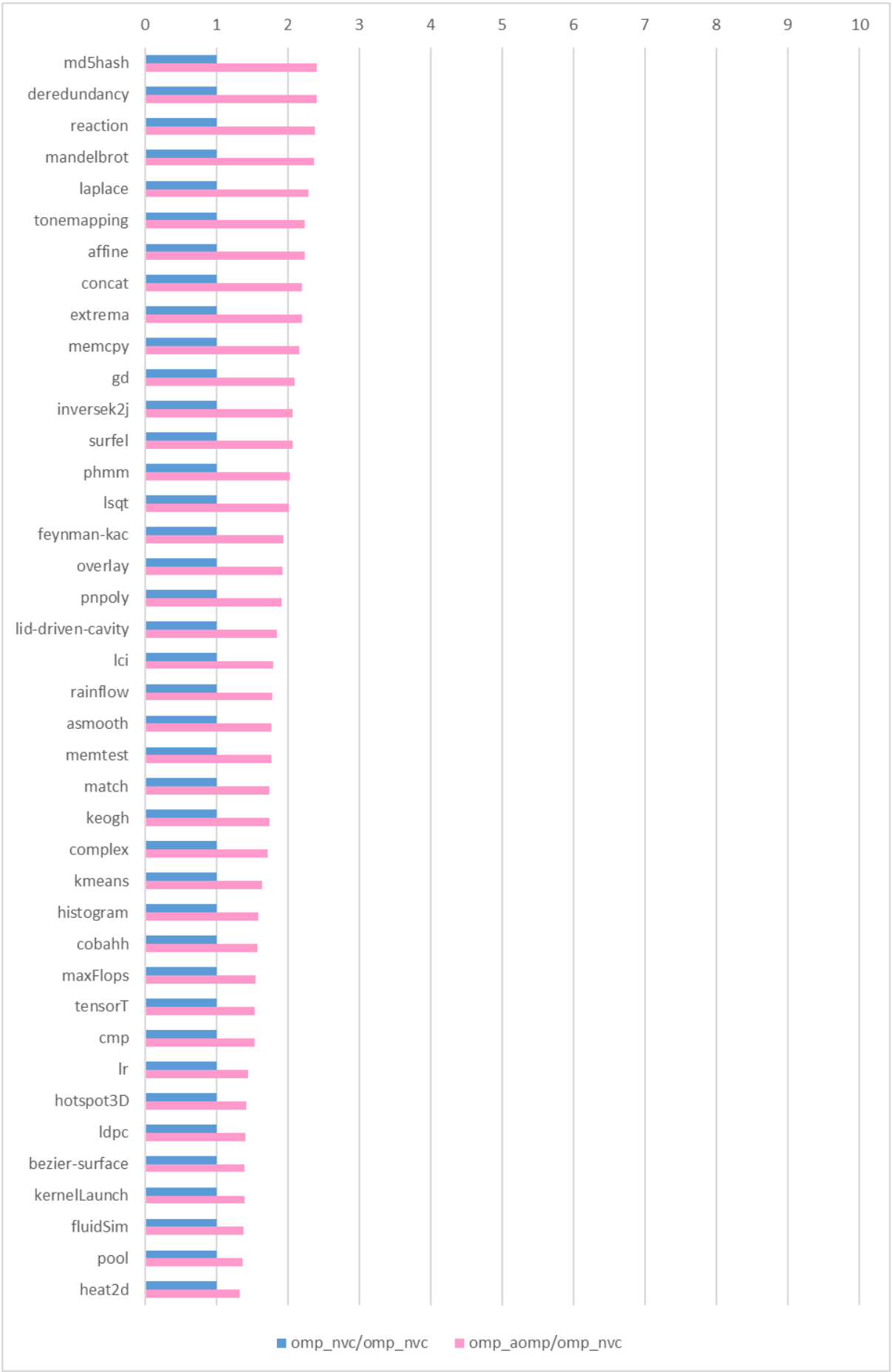


図 3.2.11.14 ベンチマーク実行時間比較（2）

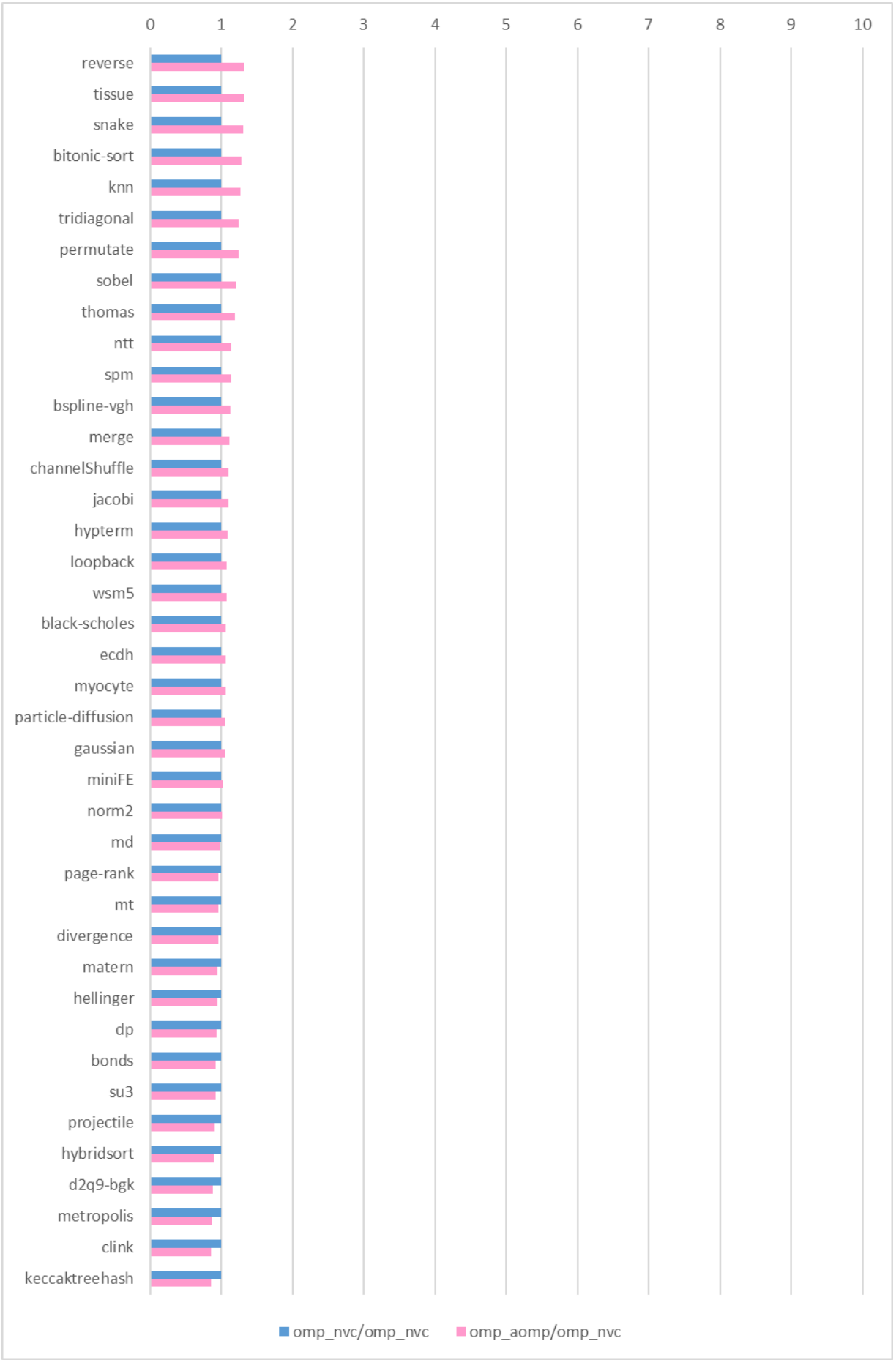


図 3.2.11.15 ベンチマーク実行時間比較（3）

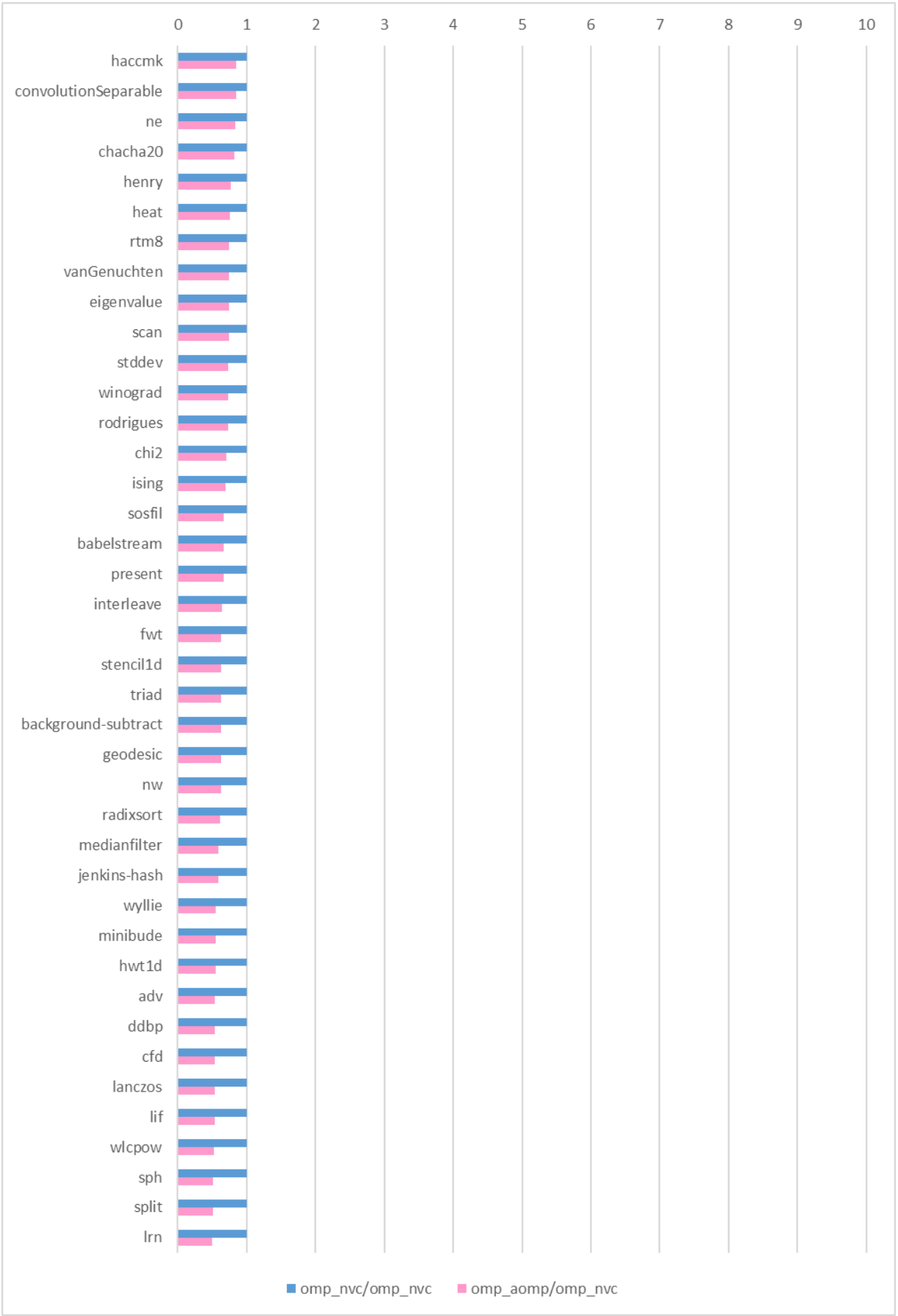


図 3.2.11.16 ベンチマーク実行時間比較（4）

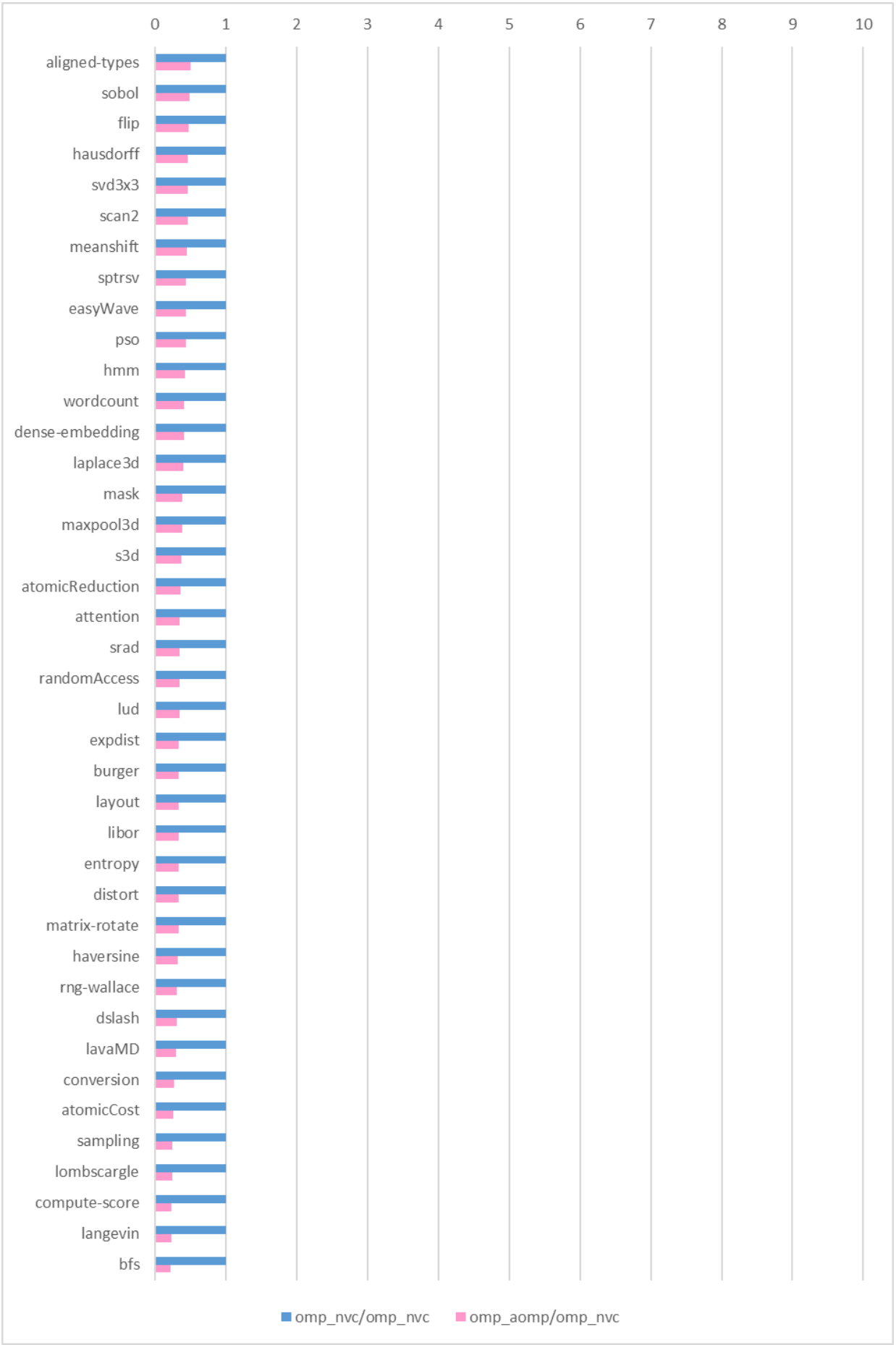


図 3.2.11.17 ベンチマーク実行時間比較（5）

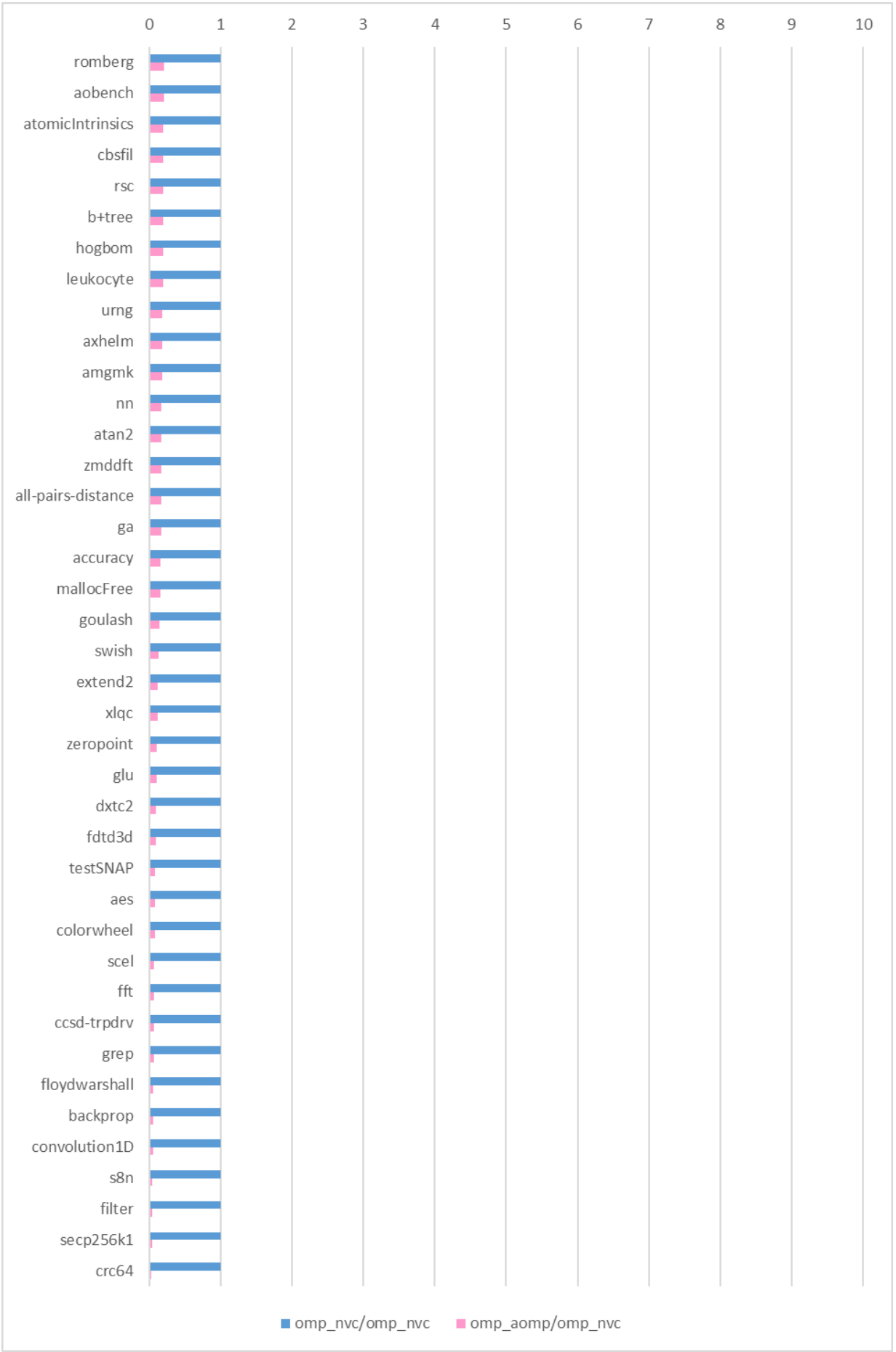


図 3.2.11.18 ベンチマーク実行時間比較（6）

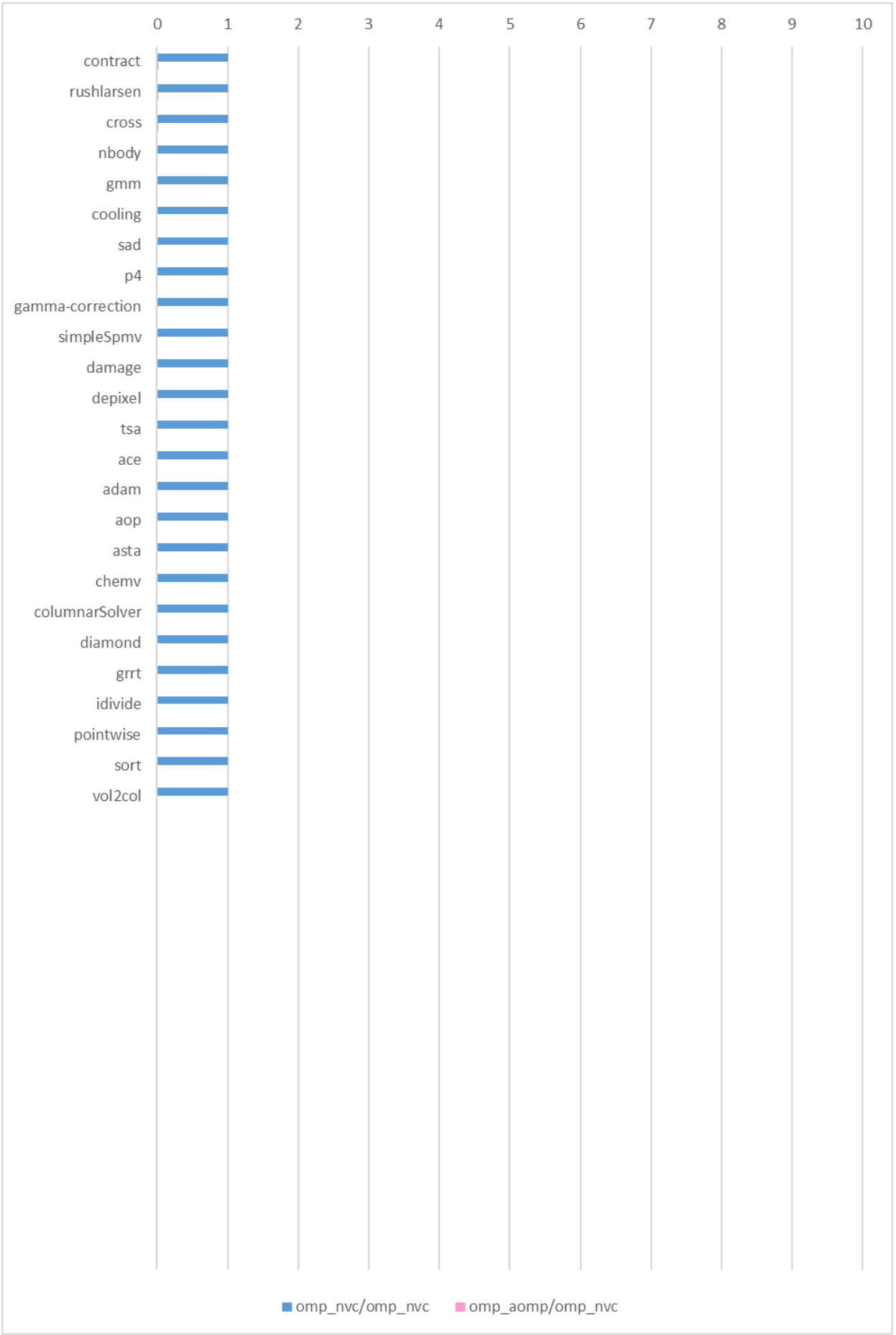


図 3.2.11.19 ベンチマーク実行時間比較（7）

3.3 まとめ

2024年度には、CPU+GPU等アーキを想定した【調査研究1~3】。さらに、次世代計算基盤におけるソフトウェア開発戦略策定の一環として、GPU向けプログラミングフレームワーク及びライブラリの評価を行った。その他、各サブグループの調査研究について以下にまとめる。

- **コンパイラ・プログラミングモデル**：LLVM-Flang の開発経緯や現状を調査し、Fortran 95 対応や GFortran との性能差を分析した。LLVM の HPC 向け最適化では、VPlan のベクトル化や機械学習を活用した最適化を調査し、Clang/Flang でのベンチマークを実施。アクセラレータ向けソフトウェアスタックやプログラミングモデル（CUDA, HIP, OpenMP 等）の対応状況を評価し、行列計算で性能を比較した。さらに、Summit から Frontier、Oakforest PACS から Miyabi への移行事例を調査し、富岳と GPU 環境でのアプリケーションベンチマークを行った。
- **スケジューラ・ランタイム**：性能分析や耐障害性を実現する実行支援ツールやワークフロー実行基盤の文献調査を実施。アクセラレータの共有利用技術を調査し、資源管理の最適化に向けた開発要素を整理。ジョブスケジューラの基盤として Slurm を選定し、コンテナ仮想化技術や運用ツール、アクセラレータ対応機能の Slurm プラグイン実装の可否を調査。
- **通信ライブラリ**： 次世代スーパーコンピュータへの採用が想定されるネットワーク技術や様々なアプリケーションを考慮し、スケーラブルな並列計算の実現に向けて必要となる新しい通信ライブラリのインターフェイスや実装技術を調査。
- **I/O・ストレージ・ファイルシステム**：富岳アプリケーションにおけるストレージの利用状況についての調査結果をまとめ、次期ストレージシステムアーキテクチャについての検討、および I/O ベンチマークプログラムの整備を行った。
- **数値ライブラリ**： 次世代スパコン向け数値計算ライブラリの提言に向けた調査を進め、既存ライブラリの動向・機能調査、性能評価、開発項目選定に関する検討を実施。2022 年度の「富岳」のライブラリ開発の振り返り、2023 年度のライブラリ比較調査を踏まえ、未調査部分を補完し、次世代計算基盤に求められる数値ライブラリの要件を整理した。これらの成果を基に、次世代スパコンに適したライブラリの開発指針を明確化した。
- **AI フレームワーク**：本編では、富岳 NEXT の候補設計の性能予測に用いる AI モデルのベンチマーク開発、AI ソフトウェアスタックのリスト作成とテストを実施。昨年の研究を踏まえ、次世代計算プラットフォームのアーキテクチャを分析し、科学アプリケーションでの高スループット推論の AI フレームワーク活用を検討。さらに、AI 推論の統合に必要なハードウェアサポートを評価し、今後の技術要件を整理した。
- **OS・仮想化・クラウド連携**： HPC 分野におけるコンテナ仮想化技術の追加調査を実施し、これまでの研究を踏まえて取り組むべき課題を整理。
- **セキュリティ**：セキュリティ技術はシステムソフトウェア・ライブラリに横断的に関わるため、関連項目を横断的に調査・検討。システム全体の視点からも検討を行い、暗号化や改ざん防止技術（ブロックチェーン）などのソフトウェア技術を調査。演算・I/O 性能への影響を評価し、横断的な評価を行った。
- **自動チューニング**：AT 技術の調査として、ベンチマーク調査やアプリケーションのチューニングを通じて AT 適用可能な性能パラメータを分析し、既存 AT ソフトウェアの性能を評価。さらに、GPU アクセラレータ活用を前提とした AT 機能や関連フリーソフトウェアを調査し、次期システムへの適用可能性を検討。また、量子コンピュータを含む新奇ハードウェアにおける AT 適用可能性とその効果についても調査を行う。
- **HPC 利用環境**：2024 年度は、データサイエンス、AI 技術、シミュレーションの融合を支える HPC アーキテクチャの調査を実施。次世代計算基盤の中核となる計算基盤技術、ネットワーク技術、データ基盤技術の利用環境を検討。2023 年度に続き、HPCI 構成機関および国立研究所の管理者に対する直接ヒアリング調査を

継続し、正確なシステムソフトウェア・ライブラリの利用状況を把握。さらに、米国、アジア、欧州の計算機センターを対象に、ウェブ公開情報をもとに導入状況を調査した。

本調査を通じて、次世代計算基盤の構築に向けた各分野の技術的課題や要件を明確にし、今後の研究開発の方向性を示すことができた。特に、CPU+GPUアクセラレータを想定し、多岐にわたる要素技術の調査を行い、次期システムに求められる機能やそのためのソフトウェア開発の方向性について知見を深めた。これらの調査研究の成果や知見が次期システム開発に取り入れられることを望む。