令和６年度科学技術試験研究委託事業
「次世代計算基盤に係る調査研究」
（システム研究調査チーム）
成果報告書

令和７年５月３０日

国立大学法人神戸大学

牧野淳一郎

本報告書は、文部科学省の科学技術試験研究委託事業による委託業務として、国立大学法人神戸大学が実施した令和６年度「次世代計算基盤に係る調査研究」（システム研究調査チーム）の成果を取りまとめたものです。

目次：

事業の名称
「次世代計算基盤に係る調査研究」（システム研究調査チーム）

１．事業の目的
ポスト「富岳」時代の次世代計算基盤について、我が国として独自に開発・維持するべき技術を特定しつつ、要素技術の研究開発等を実施し、具体的な性能・機能等についての検討を行うことを目的とする。このため、国立大学法人神戸大学を代表機関として、参画機関である株式会社 Preferred Networks、国立大学法人東京大学、国立大学法人名古屋工業大学、大学共同利用機関法人情報・システム研究機構国立情報学研究所、公立大学法人会津大学、独立行政法人国立高等専門学校機構松江工業高等専門学校、学校法人順天堂、国立研究開発法人海洋研究開発機構、国立研究開発法人国立環境研究所、学校法人東洋大学、株式会社 Quemix、国立大学法人広島大学、国立研究開発法人産業技術総合研究所と連携し、再委託により研究開発を推進する。このうち、アーキテクチャ調査研究班では CPU 性能の詳細評価、アクセラレータ制御コントローラによる、より柔軟な制御の評価を中心に、システムソフトウェア・ライブラリ調査研究班では OpenCL/OpenACC コンパイラの実現性評価を中心に、アプリケーション調査研究班ではアプリケーション性能の詳細評価を中心に実施し、代表機関である国立大学法人神戸大学がプロジェクトの総合的推進を行う。

２．2024 年度（報告年度）の実施内容
２－１．当該年度（2024 年度）の業務計画
以下を目標とし研究開発を推進する。

① アーキテクチャ調査研究
（国立大学法人 神戸大学）
商用プロセッサについて、2028-2030 年までのコストパフォーマンス、電力性能の予測を改善すると共に、独自アクセラレータアーキテクチャについて、前年度までに定めたレファレンス設計についてより詳細な設計と性能評価を進める。

② システムソフトウェア・ライブラリ調査研究
（国立大学法人 神戸大学）
前年度に定めた独自アクセラレータのレファレンスアーキテクチャ、アプリケーションを記述可能なデータ並列言語のプロトタイプに対して、アプリケーション性能を評価する。また、粒子系、構造格子系、非構造格子向けフレームワーク/DSL についてもアプリケーション性能を評価する。

③ アプリケーション調査研究
（国立大学法人 神戸大学）
対象アプリケーションに対して、性能推定を改善する。このため、カーネル部分について実装ないし信頼できるサイクルカウントモデルを構築し、これによる推定を行う。

また、分担機関と連携し、再委託によって、以下の研究開発に取り組む。

④ アーキテクチャ調査研究の統括およびアクセラレータ評価
（株式会社 Preferred Networks）

独自アクセラレータアーキテクチャについて、昨年度までに定めたレファレンス設計のより詳細な設計と性能評価を進める。

⑤ CPU 評価
(国立大学法人 東京大学)
RISC-V 等による独自実装の汎用 CPU および商用 CPU の評価のため、RISC-V アーキテクチャ設計のシミュレーションおよび実機評価によるアプリケーション実行時性能評価を進める。

⑥ アクセラレータアーキテクチャ評価
(国立大学法人 名古屋工業大学)
アクセラレータプロセッシングエレメントの詳細な実装方式について、近年の乗算器・加算等の実装技術の進歩と半導体技術の進歩、実装する演算自体の特性を考慮した検討を行う。

⑦ ネットワークアーキテクチャ評価
(大学共同利用機関法人 情報・システム研究機構 国立情報学研究所)
ネットワークアーキテクチャについて、レファレンス設計のアプリケーション性能に与える影響をシミュレーションないしは通信量の詳細な推定に基づいて評価する。

⑧ システムソフトウェア・ライブラリ調査研究の統括およびコンパイラ検討
(公立大学法人 会津大学)
システムソフトウェア・ライブラリ調査研究全体の統括を行うと共に、アプリケーションを記述可能なデータ並列言語のプロトタイプでのアプリケーションカーネル記述、その性能評価を行う。

⑨ DSL 検討
(独立行政法人 国立高等専門学校機構 松江工業高等専門学校)
粒子系、構造格子系、非構造格子向けフレームワークの性能評価を行う。

⑩ アプリケーション調査研究の統括
(学校法人 順天堂)
アプリケーション調査研究全体を統括し、問題規模の設定、アルゴリズムの検討に対して必要なサポートを行う。また他グループへのアプリケーションからの提案点要求を整理し、コデザインを有効に進める。

⑪ 創薬と深層学習応用アプリケーションの検討
(株式会社 Preferred Networks)
汎用 CPU での実行プロファイルとアクセラレータで実行する部分の性能評価に基づき、創薬と深層学習応用アプリケーションについて定めたアプリケーションにおける性能評価を行う。

⑫ ゲノム科学アプリケーションの検討
(国立大学法人 東京大学)
汎用 CPU での実行プロファイルとアクセラレータで実行する部分の性能評価に基づき、ゲノム科学アプリケーションについて定めたアプリケーションの性能評価を行う。

⑬ 地震と構造物シミュレーションアプリケーションの検討
（国立研究開発法人 海洋研究開発機構）
汎用 CPU での実行プロファイルとアクセラレータで実行する部分の性能評価に基づき、地震と構造物シミュレーションについて定めたアプリケーションの性能評価を行う。

⑭ 気象・気候シミュレーションアプリケーションの検討
（国立研究開発法人 国立環境研究所）
汎用 CPU での実行プロファイルとアクセラレータで実行する部分の性能評価に基づき、気象・気候シミュレーションについて定めたアプリケーションの性能評価を行う。

⑮ ものづくりアプリケーションの検討
（学校法人東洋大学）
汎用 CPU での実行プロファイルとアクセラレータで実行する部分の性能評価に基づき、ものづくりについて定めたアプリケーションの性能評価を行う。

⑯ マテリアルサイエンス応用アプリケーションの検討
（株式会社 Quemix）
汎用 CPU での実行プロファイルとアクセラレータで実行する部分の性能評価に基づき、マテリアルサイエンス応用について定めたアプリケーションの性能評価を行う。

⑰ 素粒子・原子核物理応用アプリケーションの検討
（国立大学法人広島大学）
汎用 CPU での実行プロファイルとアクセラレータで実行する部分の性能評価に基づき、素粒子・原子核物理について定めたアプリケーションの性能評価を行う。

⑱ 宇宙・惑星科学応用アプリケーションの検討
（国立大学法人東京大学）
汎用 CPU での実行プロファイルとアクセラレータで実行する部分の性能評価に基づき、宇宙・惑星科学について定めたアプリケーションの性能評価を行う。

⑲ アクセラレータむけ最適化コンパイラの検討
（国立研究開発法人 産業技術総合研究所）
粒子系を主な対象として、MN-Core およびその類似アーキテクチャを有効に使う方法の検討を行う。

⑳ プロジェクトの総合的推進
（国立大学法人 神戸大学）
プロジェクト全体の連携を密としつつ円滑に運営していくため、運営委員会や技術検討会の開催等、参画各機関の連携・調整にあたる。特に、プロジェクト全体の進捗状況を確認しつつ計画の合理化を検討し、必要に応じて調査或いは外部有識者を招聘して意見を聞くなど、プロジェクトの推進に資する。プロジェクトで得られた成果については、積極的に公表し、今後の展開に資する。

２－２．実施内容（成果）
本項目では、実施内容の概要を述べると共に、成果の詳細について記載する。

⓪ 本調査研究の内容と各調査研究グループの目標

0-1. 本調査研究の前提としての調査研究の概要と神戸大チームへの要望

「次世代計算基盤に係る調査研究」公募要領では、以下のように内容がまとめられている。(公募要領 P6)

> ２．事業の概要
> （１）調査研究の内容
> 本調査研究では、ポスト「富岳」時代の次世代計算基盤の具体的な性能・機能等について、サイエンス・産業・社会のニーズを明確化し、それを実現可能なシステム等の選択肢を提案する。その際、我が国として独自に開発・維持するべき技術を特定しつつ、要素技術の研究開発等を実施する。
>
> 具体的には、システム（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）、新計算原理、運用技術を対象に、技術動向等を調査し、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具体的な性能・機能を明らかにする。

また、留意すべき事項として以下が挙げられている。(公募要領 P7、8)

① 「富岳」の成果及び課題
② 評価指標として考慮すべき項目
③ 技術的課題や制約要因の抽出
④ 日本が独自に保有すべき技術と国際協調する技術の特定

評価指標については以下のように述べられている。

> 評価指標には以下を含めることが望ましい。
>
> 演算性能、電力性能比、I/O 性能、コスト、運用可能性、生産性（アプリ開発のしやすさ）、商用展開・技術展開、カーボンニュートラルへの対応等

技術的課題や制約要因について、要点をまとめると、

① 半導体等技術等の原理的限界によるものと、条件により技術的に突破できるものとを精査する。
② 技術的課題以外でも、アカデミア・産業界双方における人材不足等次世代計算基盤の構築にあたって想定される課題・制約要因を考慮する必要がある。

となる。また、日本が独自に保有すべき技術と国際協調する技術の特定については、要点は以下の通りである。

> システム（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）、新計算原理、運用技術について、国内外の情報を調査し、国内の開発対象とすべき範囲を提示する。更に、これまで培った技術の継承とともに、我が国として新たに強みを持つことができる、あるいは我が国の強みを見出せる可能性のある領域を特定する。

また、神戸大学の提案が採択されるにあたっては、神戸大学チームとして全ての可能性を検討する、というよりは、理研チームと協力・分担して補完的な役割を果たすことが望まれている、と理解している。すなわち、上記の目標を、検討対象となる全てのハードウェアアーキテクチャに対して行う、というよりも、理研チームでは検討対象になっていない、国内のベンチャー企業等の開発した、HPC ないし AI・深層学習をターゲットにしたプロセッサ技術をベースにした場合に、ポスト「富岳」時代の次世代計算基盤の具体的な性能・機能がどのようなものになりえるか、また、それは同じ時期に利用可能になると想定される国内外の商用システムに比べてどのような利点・欠点を持つのか、を検討対象とする。もちろん、

① 「富岳」の成果及び課題
② 評価指標として考慮すべき項目
③ 技術的課題や制約要因の抽出
④ 日本が独自に保有すべき技術と国際協調する技術の特定

については必要な検討を行う。

0-2. 本調査研究の背景とアプローチの概要

0-1.にまとめた調査研究全体と神戸大チームへの要請を、本調査研究では以下のアプローチによって実現する。

本調査研究では、PFN・神戸大学の持つ世界最高性能のアクセラレータ実現技術と AI 応用フレームワーク/アプリケーションソフトウェア技術を最大限に活用しつつ、従来から重要な HPC アプリケーションおよび AI 利用等新しい応用の双方について高い実行効率を実現できるシステムの構成・評価をアプリケーショングループとの密接な協力によって実現する。

まず、なぜこのようなアプローチが有効と考えられるかをこれまでの HPC システムのトレンドとそれを形作ってきた制約要因からまとめ、さらに今後の課題は何か、それに対してどのようなアプローチが必要か、という観点からまとめる。

0-2-1. 過去の HPC システムのトレンドとその技術的背景

図 0-1 に、Top500 リストにおける Top1 システム(オレンジ三角)、500 位システム(青四角)、全体合計(緑丸)の 1993 年から 2022 年までのトレンドを示す。まず明らかにわかることは、Top1 と全体合計の両方が、2012-13 頃を境にしてそれ以降の性能向上率がそれ以前より明らかに低下していることである。Top1 は 1993 年のおよそ 100 GF から 2013 年前後の 34 PF まで、20 年間で 34 万倍、すなわち 1 年で 1.9 倍の増加率で指数関数的な性能向上を示してきた。一方、2013 年から 2022 年までの 9 年間では 34 PF から 1.1 PF までのおよそ 30 倍、1 年当たりでは 1.46 倍とほぼ半分の増加率となっている。電力性能を示す Green500 (図 0-2) を見ると 2012-13 年の 2-3 GF/W 程度から 2022/6 の 62.7 GF/W まで、およそ 25 倍の向上が実現されている。すなわち、最近 10 年間の Top1 性能向上は、ほぼ電力性能の向上だけによっている。さらに、2024 年 11 月時点での Green500 11 位の数値は 72.7 GF/W にとどまっており、それ以前の年間 40%程度の性能向上の傾向から明らかにずれている。

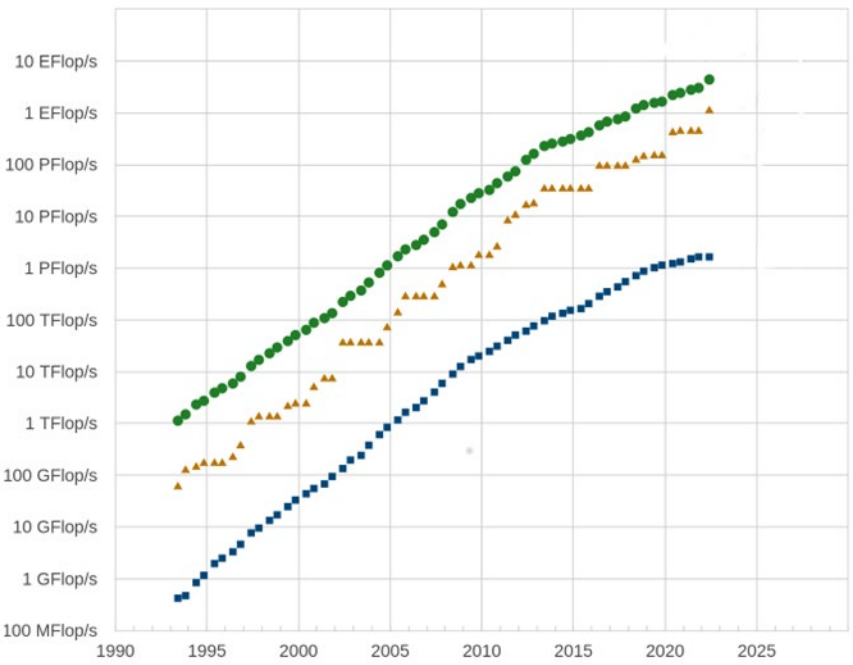

図 0-1: Top500 のトレンド

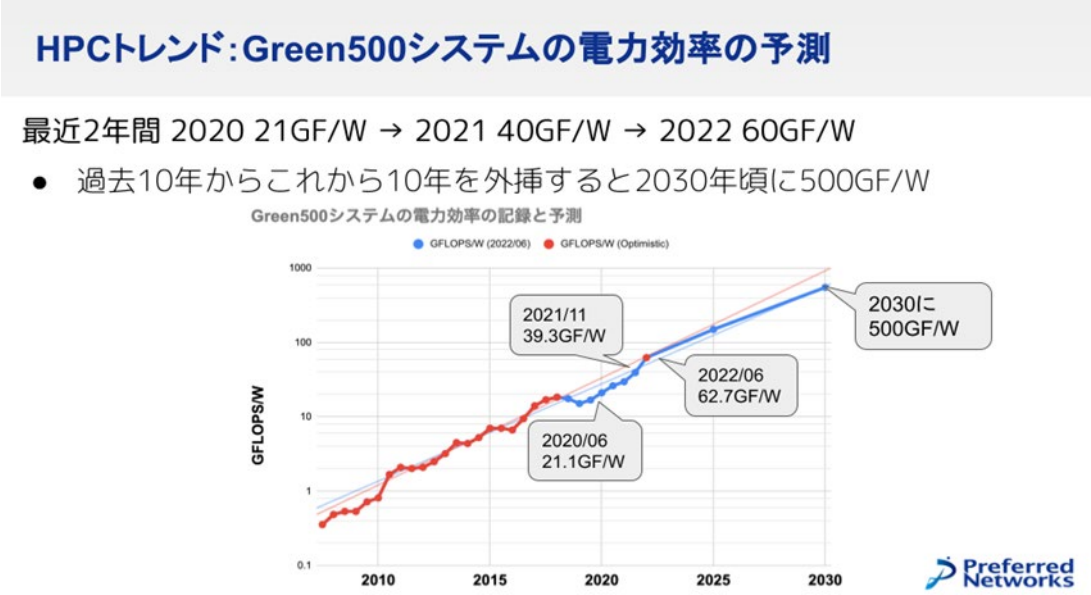

図 0-2: 電力性能のトレンドと 2030 年予測

一方、1993/11 に Top1 となった数値風洞の消費電力は 1 MW であり、電力性能は 0.24 MFLOPS/W であるが、これは当時の典型的な HPC システムとは言いがたい。事実上最後の ECL ロジックのシステムだからである。ほぼ同時期の Cray T3D はおよそ 0.5 MFLOPS/W であるので、こちらを典型的とすると、1993 年から 2013 年までの 20 年間の電力性能の向上はおよそ 5000 倍である。つまり、この期間は、10 年間当たりの電力性能向上はおよそ 70 倍

であり、さらに９倍程度の消費電力の増加によってほぼ 10 年間に 600 倍の性能向上を実現してきたということになる。

一方、2013 年以降は電力性能の向上率が 10 年で 30 倍程度に低下し、さらに Top10 クラスのシステムの消費電力が 20-40 MW と巨大になったため、それ以上に消費電力を増加させることが現実的ではなくなり、電力性能の向上率の低下もあわさって Top1 システムの性能向上率が大きく低下した。

1993 年から 2013 年までの HPC システムの性能向上にもっとも大きく貢献したのは半導体技術の進歩である。デザインルールは 0.8 μm から 45 nm まで 1/18 に縮小し、コア電圧は 3.3 V から 1 V 前後に低下している。理論上はこれらによって 180 倍の電力性能の向上が実現できる。実際の向上は上でみたように 5000 倍であるので、あと 28 倍程度を主にアーキテクチャの改良で実現したことになる。但し、45 nm の典型例は IBM BG/Q であるが、これは SOI プロセスを使っており半導体技術による改良分がさらに２倍程度あるかもしれない。それでも、10 倍以上をシステムアーキテクチャの改良によって実現している。

「システムアーキテクチャの改良」の内訳を見ていく。

T3D では、システム全体の消費電力のうちプロセッサの消費電力はごくわずかであった。1 ノードの消費電力はおよそ 300 W だが、プロセッサチップの消費電力は 20 W 前後(150 MHz 動作の場合)である。メモリおよびネットワークインターフェースが電力の大半を消費していることがわかる。

一方、BG/Q では、ノード当たり 80 W 程度の消費電力のうち 55 W をプロセッサチップが占める。メモリおよびネットワークの消費電力を相対的に大きく減少させていることがわかる。すなわち、全消費電力に対するプロセッサチップの消費電力の割合がほぼ 8 倍に増加しており、これが「アーキテクチャの改良」のほとんどを占める。これらは、基本的にはメモリバンド幅およびネットワークバンド幅の相対的な低下によって実現したものである。メモリ B/F では 2.5 から 0.2、ネットワーク B/F では 12 から 0.2 と大きく低下している。

このため、プロセッサアーキテクチャ自体の改良による消費電力の削減はそれほど大きなものではない。もっとも効果が大きいと思われるのはプロセッサコア当たりの演算数の増加である。Cray T3D システムに採用されていた Alpha 21064 プロセッサはサイクル毎に加算または乗算の１命令であったが、BG/Q は 4 FMA 8 演算となっている。この効果は２-３倍程度であることがわかる。

2013 年から 2022 年までの電力性能の 25 倍の向上の内訳をみていく。2022 年の電力性能のトップを実現したのは Frontiers であり、AMD MI250X GPU を使っている。従って、BG/Q とほぼ同時期に同様の消費電力を GPU で実現した Titan と比べるのが適切であろう。Titan は NVIDIA K20x であり、BG/Q とは異なり TSMC 28nm プロセスである。一方、MI250 は TSMC の「N6」プロセスである。

TSMC の場合 16FF 以降の FinFET プロセスでは CMOS スケーリング則の根拠となる「同一形状でのトランジスタの小型化」が成り立っていないので、TSMC の発表による消費電力の低下をレファレンスとする。28HPM、16FF、16FF+、N7 の間の電力削減はそれぞれ 48%、30%、60%とされているので、電力効率向上は７倍となる。従って、３倍強をアーキテクチャ改良で実現していると考えられる。このうち最大のものは、ベクトル演算から行列演算への

移行であると考えられるが、AMD MI250X アーキテクチャは倍精度演算に特化する等特殊な特徴を持つためその寄与も大きいと思われる。

なお、この、９年間で７倍の半導体の性能向上は、それまでの 20 年間の 180 倍に比べてそれほど小さくない。前者は年間 24%であるのに対して後者は 30%である。すなわち、半導体技術の進歩のスローダウン、いわゆるムーアの法則の限界は、2022 年までにおいては HPC システムの性能向上のスローダウンの主要な要因ではなかったことがわかる。プロセッサチップ以外の消費電力を限界近くまで削減したこと、消費電力が限界近くまで増加したことの２つの効果が大きい。

表 0-1 に K20x と MI250X の比較を示す。MI250X の名目倍精度ピーク性能は 95.7 TF と AMD の公式資料にはあるが、Top500 サイトの名目ピーク性能と公式の Frontiers の構成からはその半分以下の 44.8 TF がピーク性能となっており、なんらかの矛盾がある。ここでは消費電力は名目の値とし、ピーク性能は AMD 公式の半分とした値を参考値とした。

表 0-1: NVIDIA K20x と AMD MI250X

	FP64 peak(TF)	メモリバンド幅 (GB/s)	消費電力(W)	電力性能(GF/W)	B/F
K20x	1.31	250	235	5.57	0.19
MI250X	95.7(47.9)	3276.8	560	171(85.5)	0.03(0.07)

名目の電力性能は 15 倍であるが、B/F 値はほぼ 1/3 になっていることがわかる。AMD 公称値の半分のピーク性能としても B/F = 0.07 は「富岳」の 1/5 であり、多様なアプリケーションで高い性能を実現するのは容易ではない。仮に 95.7 TF が実現されているなら、B/F 値は実に 0.03 となり、極めて演算密度の高いアプリケーション以外では高い実行効率を実現することは困難ではないかと予想される。

すなわち、過去 10 年間の傾向をそれ以前と比較すると

① プロセッサチップの消費電力がシステム全体の消費電力の大きな割合を占めるようになった
② システムの消費電力が現実的な限界である 20-40 MW に到達した
③ 半導体プロセスの微細化による消費電力の削減の速度が若干低下した

ことによって HPC システムの性能向上率はそれまでのほぼ半分になったといえる。この性能向上率は

④ B/F 値の大幅な低下
⑤ 倍精度まで行列演算命令を採用することによる電力性能の向上

によってようやく実現できたものであり、どちらも多様なアプリケーションで高い実行効率を実現することを困難にするものである。さらに、2022 年以降を見ると電力性能向上率の低下が明らかになりつつある。これは、N6 以降のプロセスの性能向上率の低下と、アーキテクチャの改良による性能向上の限界の両方が要因になっているように思われる。

一方、この 10 年で飛躍的に発展し、今後もさらなる発展が期待できる深層学習分野では、16 ビット浮動小数点等の短い語長での行列・行列積が計算の主要部分となっており、これを高速に実現できることが極めて重要になった。NVIDIA は 2017 年に汎用 GPU としては世界で初めて 16 ビット行列乗算でユニット持つ Tesla V100 を発表し、16 ビット行列乗算で前世代の P100 の 6 倍以上の性能を実現した。深層学習では、他の HPC アプリケーションに比べて低い B/F 値が大きな制約にならないことが多く、おそらく市場規模として人工知能・深層学習のほうがその他の従来型の HPC よりも既に大きくなっている現在、市場原理に従って開発されていく HPC システムは深層学習を主要なアプリケーションとして開発されていくことになろう。NVIDIA は 2020 年の A100、2022 年の H100、そして 2025 年の B300 と新しい製品の出荷を始めているが、 H100 では FP8、B300 では FP4 をサポートし、逆に FP64 の性能は B300 になって低下している。機械学習以外の多くの HPC アプリケーションでは、FP32 の利用も進みつつある一方、依然として FP64 演算は重要であり、どのようにして FP64 性能を確保するかは大きな問題になりつつある。

ただし、ここ数年で深層学習の主要なワークロードになりつつある LLM は、これまでの多くのモデルで使われてきた CNN とは大きく違う。計算機アーキテクチャの側からみて重要な違いは、

① モデルパラメータの数が、CNN では 10 億を超えることはなかなかなかったのに対して LLM では兆を超えるものものあり、またこのパラメータあたりの計算量があまり多くない
② コンテクストを記憶する「KV キャッシュ」が極めて多くのメモリアクセスを必要とする

ということであり、これらは学習よりも推論でより大きな問題となる。一方、今後の LLM の利用では、推論のほうが学習よりもはるかに大きな計算資源を必要とするため、LLM 向けのプロセッサは非常に高いメモリバンド幅が必要になってきている。実際、現行の GPU の LLM 推論の実行効率は数%である。

0-2-2. 今後の HPC システムのトレンドの概要

前節で見たように、過去 30 年間については、HPC システムの性能向上は以下の 4 要因によっていた。

① プロセッサチップ以外の周辺ロジック、メモリ等の電力の削減
② 消費電力の増加
③ 半導体技術の進歩による 1 トランジスタの消費電力の削減
④ プロセッサアーキテクチャの進歩による演算当たりの電力の削減

今後は①、②は期待できないというのは既にみた通りである。③、④についてはどうだろうか？

詳細な分析は次章のアーキテクチャグループ報告で述べるが、以下に 2029 年に稼働するシステムを前提にして概要を述べる。

まず③の半導体技術についてである。現在のところ TSMC は 2025 年後半に N2 プロセスでの量産を始める計画であり、これまでと同様なペースでプロセス開発を続けるなら 2028 年には N1.4 (A14) が利用可能になると期待できる。N7、N5、N3、N2 の世代毎の電力削減は 30%、

30%、15%であるので、N2 から N1.4 を楽観的に 30%とし、さらに現在の計画通りに開発が進むとして、N7 からの電力性能向上は 3.4 倍となる。GAA プロセスの開発が多少遅延すると 2.5-3 倍となるであろう。ただし、この電力向上の見積もり間違いなく楽観的であり、そもそも N5 以降の性能向上がここまで大きくない。

④のプロセッサアーキテクチャについては予測は難しいが、外挿としてありえるのは行列乗算の単位を大きくすることでの省電力化であろう。倍精度で2×2の行列を扱っているものを4×4にするものである。これによる電力性能向上は行列命令の導入ほどは大きくなく、1.5 倍程度と考えられる。

従って、③、④から総合的には、3.75-5.1 倍程度の電力性能向上、すなわち Green500 で 240-320 GF/W、ボードレベルのピークではその 1.5 倍、360-480 GF/W となる。

これは N1.4 の場合の推定値であり、N2 の場合は Green500 で 170-225 GF/W、ボードレベルで 250-335 GF/W となる。

Green500 で 320 GF/W、平均電力が最大電力の 90%とすると、30 MW の電力リミットでの HPL 性能は 8.64 EF となる。

0-2-3. 実行効率と「使いやすさ」のトレンド

特にここ 15 年程度の HPC システムの、使う側からみた大きな問題は、高い実行効率を実現することが非常に困難、あるいは不可能になってきた、ということである。「京」でも、非常によく最適化・チューニングされた大規模アプリケーションの典型的な実行効率は 10-15%であった。もちろん、「京」は 2011、2012 の２年連続でゴードン・ベル賞を獲得しており、それらでは 50%前後の実行効率を実現したが、これらが典型的とは言いがたい。規則格子の数値粒子コードで 15%、不規則格子の有限要素法では 2-3%というのが典型的な効率であった。

「富岳」では、この効率が大きく低下した。「京」のコードそのままでは、規則格子等では 15%の効率が 5%以下、また 50%に近い効率を実現していた粒子法コードでも 10%以下等である。以下に「富岳」で実行効率が低くなる主要な要因をまとめる。

① 演算、L1 および L2 のレイテンシが非常に大きいこと
② SIMD 命令セットの HPC-ACE から SVE への変更のためアーキテクチャレジスタ数が大きく減少したこと
③ x86 に比べて半分程度の OoO(アウトオブオーダー実行)資源
④ x86 に比べて半分程度の L1,L2 バンド幅
⑤ 比較的貧弱な分岐予測機構

この中で、①、②、③はそれぞれが単独で問題というよりはこの３つが組み合わさって大きな問題になっているものである。例えば x86 は②のアーキテクチャレジスタ数の問題は抱えているが、①、③では「富岳」より余裕があり、結果的に実行効率低下の程度が小さい。

一方、上にみたような設計パラメータの選択は、②、⑤を別にすると高い電力性能とメモリバンド幅を汎用 CPU 設計で実現するためには必要なものであったように見える。大きな

演算レイテンシはおそらく低電圧での比較的高いクロック実現に貢献しているであろうし、0o0 資源やキャッシュバンド幅の削減は大きな消費電力削減に直結する。

なお、「京」においても、多くのコードの実行効率は決して高くはない。地球シミュレータのような古典的ベクトルプロセッサであれば、規則格子、不規則格子のどちらでも 50% ないしそれ以上の実行効率を実現できたであろう。

「京」において多くのアプリケーションの実行効率が地球シミュレータよりも大きく下がった基本的な要因は B/F 値の低下である。地球シミュレータは B/F = 4 を維持していたのに対して、「京」の B/F は 0.5 である。それでも単純に 1/8 まで効率が落ちるわけではなく、典型的には 1/4 程度であるのは、キャッシュ等がある程度は働いているからであるが、逆にいうとこれらのコードと「京」のアーキテクチャの組合せではキャッシュの有効性がその程度にとどまるということである。

ここで注意するべきことは、「京」における効率低下も、「富岳」における上でみた理由による性能低下も、原理的な限界ではない、ということである。

この問題は、以下のように理解することができる。

地球シミュレータのような、古典的なベクトルアーキテクチャに最適化されたアプリケーション向けに、キャッシュアーキテクチャを最適化することを考える。ベクトルアーキテクチャに最適化されたアプリケーションコードは、

① なるべく長いベクトルを
② なるべく連続に

アクセスするようなループ構造を持つ。これは、複数のプロセッサがそれぞれ高い並列度でメモリアクセスしても実行効率が高くなるように、ループ繰り返し回数を大きくする必要があるからである。このため、必然的にキャッシュに入らないような広い領域を連続にアクセスすることになり、キャッシュデータの再利用率は低くなる。

このようなメモリアクセスに対してなるべく高い実行効率を実現するためには、主記憶のバンド幅を、特に連続アクセスに対して高くとることが最重要であり、L2、L1 キャッシュのバンド幅は相対的に重要でなくなる。さらに、特に L2 キャッシュのレイテンシはあまり重要ではなくなる。再利用されるキャッシュというより、ストリームアクセスのためのバッファメモリ的な使われ方になるからである。

一方、規則格子差分法のアルゴリズム自体を、メモリ階層がある、すなわちオンチップメモリは比較的低いレイテンシと高いバンド幅でアクセスでき、オフチップメモリは高いレイテンシと低いバンド幅にならざるを得ないマシンで実行することを考える時、まずバンド幅のことだけを考えるなら、最適なアルゴリズムはキャッシュブロッキング、すなわち、例えば３次元格子なら、それをキャッシュメモリに入る程度の小領域に分割し、順番に処理する、という方法であるはずである。

特に、近年はテンポラル・ブロッキングと呼ばれる、そのようなブロックに対して１ステップではなく複数時間ステップの積分を行うことで必要メモリバンド幅をさらに減少させる方法の研究が世界的にも活発に行われている。

しかし、「京」・「富岳」のアーキテクチャではこのような方法で実効効率をあげることは困難であることがわかっている。これは、特に LLC である L2 キャッシュのバンド幅が小さく、レイテンシが大きく、容量が小さいためである。

すなわち、日本の国家プロジェクトである地球シミュレータ、「京」、「富岳」の系列では、前世代のアーキテクチャに最適化されたアプリケーションコードに対して、次世代の半導体技術の制限の中でアーキテクチャを最適化する、というアプローチに結果的になってしまったため、原理的なアプリケーション・アルゴリズムに対する最適解とは違うものになってしまっている。特に、上でみたように、アーキテクチャが本来有効な最適化の効果を阻害するため、結果的にアプリケーションの進化も阻害し、そのことがさらにアーキテクチャの進化を阻害する、という悪循環になる。

なお、これは「京」・「富岳」だけの問題というわけではなく、x86 や Arm 等の汎用プロセッサでも、また GPU でも同様にある問題であり、特に GPU は「京」・「富岳」とほぼ同様な問題を抱えている。非常に高い演算性能に対してキャッシュメモリは十分な大きさと速度を持っていないため、高い性能を出すにはレジスタデータの再利用が必須となっている。

一方、x86 等は、実際にキャッシュデータ再利用ができるアプリケーションコードに対して高い実行性能を実現できるが、そのために短いループ長でも性能が落ちない低レイテンシのキャッシュ、演算ユニット、さらに少ないアーキテクチャレジスタでもパイプライン実行ができるための莫大な OoO 資源が必要になり、これらの電力消費が高い電力効率の実現を妨げている。

ここまでをまとめると、地球シミュレータ・「京」・「富岳」の系列においては、「京」・「富岳」アーキテクチャデザインの時にレファレンスとなるアプリケーションコードが基本的に地球シミュレータ向けに最適化されたコードであったために、例えば規則格子差分法というアルゴリズムとしては基本的なものに対しても、アーキテクチャ設計が原理的な制約の下での最適点ではなく、前世代のアーキテクチャ向けに最適化されたアプリケーションコードに対する最適点となるため、アプリケーションとアーキテクチャの健全な共進化を阻害している。これは日本だけの問題ではなく、GPU アーキテクチャもまた汎用 CPU アーキテクチャと同様な困難を抱えている。

本調査研究では、上記の理由から、検討対象とするアプリケーションエリアに対して、既存のアプリケーションコードそのものを評価対象にするのではなく、検討するアクセラレータアーキテクチャ向けにデータ構造・ループ構造等、場合によっては数値スキーム自体の書き変えを行った場合の性能を評価し、その評価に基づいてアプリケーションとアーキテクチャのコデザインを進める。その時に、現在および将来の半導体技術の発展方向も考慮することで、長期にわたって有効なアプリケーションを開発するための方法を明らかにする。以下、調査項目別に成果を記載する。

①アーキテクチャ調査研究

（国立大学法人 神戸大学）

2024 年度は商用プロセッサについて、2028-2030 年までのコストパフォーマンス、電力性能についての予測を更新すると共に、DRAM バンド幅向上についてアップデートを進めた。詳細は④～⑥ならびに⑲で述べる。

②システムソフトウェア・ライブラリ調査研究
（国立大学法人 神戸大学）
システムソフトウェア検討チームでは、処理系の開発、評価を進めた。詳細は⑦、⑧で述べる。

③アプリケーション調査研究
（国立大学法人 神戸大学）
想定アプリケーションに対して、MN-Core での実装方法の検討と性能評価を進めた。具体的には、分子動力学、ゲノム解析、地震・構造物シミュレーション、気象・気候シミュレーション、構造解析(ものづくり)、マテリアルサイエンス、素粒子・原子核・宇宙の各アプリケーション領域で、想定アプリケーションの演算カーネルの性能評価を進めた。詳細は⑨～⑱で述べる。

④アーキテクチャ調査研究の統括およびアクセラレータ評価
（再委託先 株式会社 Preferred Networks）
アーキテクチャ調査研究全体の統括を行うと共に、2024 年度は商用プロセッサについて、2028-2030 年までのコストパフォーマンス、電力性能の予測を更新すると共に、DRAM バンド幅向上の予測を行った。さらに、演算器の実装方式について予備的な検討を行った。

4-1. 商用アクセラレータの電力性能トレンドの推定
商用アクセラレータとして NVIDIA 社および AMD 社の GPGPU のこれまでのトレンドを調査し、将来の予測を実施した。

まず、GPGPU の世代間の性能向上が、TSMC 社のプロセスノードの性能改善との相関を調査した。TSMC 社のプロセスノード間の性能改善が表 4-1 である。

表 4-1: TSMC 社のプロセス情報

プロセス情報	N16/12	N10	N7	N5	N3	N3E	N2
Speed Improvement@Same Power	50%	15% vs N16	20% vs N10	20% vs N7	15% vs N5	18% vs N5	15% vs N3E
Power Reduction@Same Speed	60%	35% vs N16	40% vs N10	40% vs N7	30% vs N5	34% vs N5	30% vs N3E
Logic Density		x2	x1.6	x1.8	x1.7	x1.6	不明
出典	TSMC HP	TSMC HP	TSMC HP	TSMC HP WikiChip	TSMC HP	TOM's Hardware	TOM's Hardware

上記 TSMC 社のプロセスノード間性能改善と、NVIDIA 社 GPGPU の電力性能トレンドとの相関の調査および、将来予測結果を示したものが、表 4-2 である。

2024 年末になって、NVIDIA からは B200・B300 の詳細と、さらに次世代の Rubin, Rubin Ultra の概要が発表され、２つの新製品が発表されるということになった。B200, B300 は H100 からの変更は大きくないが、 FP4 がサポートされること、FP64 の性能が大きく低下すること、メモリバンド幅が向上することが主な変更点である。さらに Rubin では、現在のところ FP4 と FP8 以外の性能は明らかにされていない。

表 4-2: NVIDIA 社 GPGPU のトレンドと将来予測

		既製品					予測	
製品名		P100	V100	A100 SXM	H100 SXM	B200	Rubin	Rubin Next
製品発表年		2016	2017	2020	2022	2025	2026	2028
プロセス(量産開始年)		N16(2015)	N12(2017)	N7(2018)	N4(2021)	N4P(2023)	N3(2024)	N2(2026)
Spec	演算性能@FP8(TF)	N.A.	N.A.	N.A.	1979	2250	8000	9600
	演算性能@FP16(TF)	21.2	125	312	989	1125	?	?
	演算性能＠FP64(TF)	4.7	7.8	19.5	67	18.5	?	?
	FP16/FP64	4.5	16.0	16.0	14.7	61	?	?
	周波数(MHz)	1328	1455	1410	1780	1837	1800	2000
	TDP(W)	300	300	400	700	700	900	1000
	電力効率 GF/W@FP64	15.67	26.00	48.75	95.71	26.5	?	?
	B Tr 数	15.3	21	54	80	108	120	180
	Die Size(mm^2)	610	815	826	814		814	814
	DRAM Size(GB)	610	815	826	814		814	814
	DRAM バンド幅(GB/s)	720	900	2000	3000	4000	6500	8000

これは、以下の 3 つを意味していると考えられる。

① NVIDIA が AI マーケット、特に FP8、FP4 が重要と考えられている生成 AI や LLM を主要なターゲットとした。
② NVIDIA アーキテクチャでは FP64/32 性能と FP16/8/4 性能にはトレードオフがあり、今後の製品では FP64/32 性能の向上は期待できない。
③ メモリバンド幅は HBM の世代交代により向上するものの、向上速度は低下する。

①は、ここ数年の AI マーケットの急速な膨張が今後も続くと想定されることから当然と考えられる。②は、設計として FP64/32 と FP16/8/4 が回路を共有していない、あるいは共有部分が小さいことを意味しており、興味深い。PFN の MN-Core では、精度が異なる演算間で可能な限り乗算器等のユニットを共有する構成（特許取得済）になっており、FP64 の性能をカットしても回路規模の大きな減少にはつながらない。また、そもそも演算性能に対してシリコン面積が小さく、他社のプロセッサに比べて大きな優位性を持っている。

③については、LLM 推論では明らかにメモリバンド幅向上が望ましいが、メモリバンド幅向上のための手段は基本的に HBM の世代交代によるということを意味しているように見える。

HBM3e はデータ線が 1024 本であり、HBM3 の 6.4G bps を当初 9.6 Gbps まで上げるとされていた。実際に製品に採用されたものではまだ 8 Gbps にとどまっており、また開発中のものも 9.2 Gbps にとどまる模様である。

一方、HBM4 は、HBM3e と比べると以下の３つの点で異なる。第一は、データ線が 2048 本に増えることである。これはもちろんバンド幅の増加をもたらす。第二は、標準設計においても、いわゆるベースダイに DRAM プロセスではなくロジックプロセスが用いられることである。これは、ベースダイがより低消費電力で高速に動作することを可能にし、消費電力の削減を可能にする一方、無視できない価格上昇をもたらす。第三は、各社が計画している「Custom HBM」である。これは、ベースダイの設計をカスタマイズすることで、HBM4 で定義された標準のチップ間インターフェースではなく、例えば UCIe や他の D2D に特化した高速・低電力で面積も小さいインターフェース IP を使うことを可能にするものである。これらにより、ある程度の消費電力の削減が期待できる。

一方、システム全体として見ると、もちろん HBM の DRAM 自体およびロジックダイの DRAM I/F までの消費電力はある程度減る一方、ロジックダイ内部のデータ移動の消費電力は大きく減らすことは困難である。このことが、NVIDIA GPU のメモリバンド幅の向上があまり大きくないことの原因と考えられる。

このように、NVIDIA 社の GPGPU の世代間の演算性能および消費電力の改善は、B200 に至るまで基本的には TSMC 社の公称改善率以内に収まることが確認できた。

V100 および H100 SXM では演算性能が前の世代から大きく改善しているが、ここは Tensor コアの採用や Tensor コアの大型化、半導体プロセスの改善以上の周波数向上等により演算器当たりのトランジスタ数を改善したことによると思われる。

その結果、前の世代に比べてダイサイズの大型化、トランジスタ当たりの消費電力量が改善していない等の状況が見て取れ、基本的には TSMC 社の公称改善率以内での改善に収まるペースでの改善トレンドであると言える。

それをもとに将来の予測をしたものが表 4-2 内の Rubin および Rubin Next の数値である。

この予測においては、TSMC 社の半導体プロセス N3E、N2 の情報を元に、実際の NVIDIA 社 GPGPU の改善幅を仮定して算出している。

「富岳」NEXT 世代では、半導体プロセスの量産化が順調に進めば N2 世代の Rubin Next の世代、若干の遅延が生じると N3E の Rubin 世代となると思われる。これらでは、現在のところ FP64/FP32 の性能は不明であるが、B200 から大きく改善しない可能性が高い。

表 4-3 は AMD 社 GPGPU のトレンドと将来予測であり、こちらも NVIDIA 社 GPGPU と同様の分析を実施した。

こちらも NVIDIA 社と同様に、基本的には TSMC 社の公称改善率以内に収まることが確認できた。それをもとに MI300X より将来予測をすると、MI400X は倍精度で 131 TF, 792 W,165 GF/W、MI400 Next では 207 TF,800 W で 259 GF/W と予想される。昨年度の報告書からいくつか数字が変わっており、これは A16 のアナウンスを受けたものである。なお、MI300X はダイサイズ、トランジスタ数共にトレンドよりもかなり大きくなっているが、これは

MI300X ではハイブリッドボンディングを採用したキャッシュ・IO ダイと計算コアダイの３次元積層が採用されているからである。この３次元積層によりトランジスタ数を大幅に増やした一方、電力性能の改善は N6 から N5 へのプロセス変更に対して比較的小さくなっている。従って、将来予測でも本来電力性能を下げるべきだが、ここではその変更は行わず、高めの電力性能予測とした。

表 4-3: AMD 社 GPGPU のトレンドと将来予測

		既製品						予測	
製品名		MI8	MI25	MI50	MI100	MI250X-OAM	MI300X	MI400X	MI400 Next
製品発表年		2017	2017	2018	2020	2021	2023	2025	2028
プロセス(量産開始年)		N28	GF14	N7	N7	N6(2020)	N5	N3E(2023)	A16(2026)
Spec	演算性能 @FP16(TF)	8.2	26.4	26.5	186.4	383	1300	2080	3289
	演算性能@ FP64(TF)	0.512	0.768	6.6	11.54	47.85	81.3	131	207
	FP16/FP64	16.0	34.4	4.0	16.2	8.0	16.0	16.0	16.0
	周波数(MHz)	1000	1500	1725	1502	1700	2100	2100	2625
	TDP(W)	175	300	300	300	560	750	792	800
	電力効率 GF/W@FP64	2.93	2.56	5.75	5.01	85.45	109.0	165	259
	B Tr 数	8.9	12.5	13.2	21	58.2	153.0	245	300
	Die Size(mm^2)	596	510	331	763.2	1448	1017	1100	1100
	DRAM Size(GB)		16	16	32	128	192	192	256
	DRAM バンド幅(GB/s)		436	1020	1230	3200	5300	8000	12800

NVIDIA 社、AMD 社ともに予測値は当然上振れ、下振れが考えられるが、それを加味しても 300 GF/W の達成は電力効率の大幅な向上が必要なため困難に見える。

次に、アクセラレータの費用の分析を実施した。この部分は昨年度からアップデートはない。16/12 nm から 5 nm まで、ファウンドリから販売される Wafer の価格を調査し、それをもとに 3 nm、2 nm の価格を予測したものが表 4-4 である。

表 4-4: Wafer 価格と価格上昇率

プロセスノード	Wafer 価格	価格上昇率
16 nm/12 nm	$3,984	-
10 nm	$5,992	1.50 倍
7 nm	$9,346	1.56 倍
5 nm	$16,988	1.82 倍
3 nm	$30,901（予測）	1.82 倍（予測）

2 nm	$56,209（予測）	1.82倍（予測）

Wafer 価格の出典:
https://www.techpowerup.com/272267/alleged-prices-of-tsmc-silicon-wafers-appear

ここから、ファウンドリから販売される Wafer の価格は世代が進むごとに指数関数的に上昇しており、さらに、世代が進むごとに金額の上昇率自体が上がっている傾向が見てとれた。

3 nm、2 nm では5 nm 時点の1.82倍以上の倍率となることが容易に想像できるが、最低ラインである同じ倍率で試算している。

「富岳」では7 nm のプロセスを使用していたが、仮に本プロジェクトにて3 nm を採用すれば、wafer 価格は最低でも3.3倍、2 nm を採用すれば最低でも6倍の Wafer 価格となることになる。

本プロジェクトでは、この半導体価格の上昇が大きな制約となることが予想される。

4-2. 商用 CPU の電力性能トレンドの推定

商用 CPU のトレンド調査のため、2015 年以降に発売された Intel 社 CPU のうち、サーバー向けである Platinum ランクからモデルを抜粋したものが表 4-5 である。ここも昨年度からアップデートはない。

表 4-5: Intel 社 CPU のトレンド

コードネーム	発売年	ランク	モデル	コア数	最大周波数 [GHz] (AVX512)	TDP [W]	最大演算性能	GF/W
Sapphire Rapids	2022	Platinum	8490H	60	2.9	350	5568	15.9
IceLake	2019	Platinum	8368Q	38	3.3	270	4013	14.9
Cascade Lake	2019	Platinum	8280L	28	2.4	205	2150	10.5
Skylake	2015	Platinum	8180M	28	1.7	205	1523	7.4

2015 年発売の Skylake 世代から、2022 年発売の Sapphire Rapids 世代までの８年間で、最大演算性能は3.66倍に上昇している。しかしながら、2016 年発売の NVIDIA 社 GPGPU P100 と 2022 年発表の H100 SXM は、７年間で倍精度演算の演算性能が14倍まで上昇していることを考慮すると、CPU の演算性能の向上は極めて緩やかであることが伺える。

この演算性能の向上は最大周波数および演算コア数によってもたらされているが、Sapphire Rapids はチップレット技術を採用することで、より多くのコア数を実現している。これを除けばコア数はほぼ横ばいであり、その演算性能の向上は周波数の向上によってもたらされていることがわかる。

また、電力効率も８年間で２倍程度の改善と、NVIDIA 社 GPGPU が5.22倍であることを踏まえると、こちらも極めて緩やかであると言える。

4-3. MN-Core 後継の可能な電力性能の推定

PFN・神戸大学で、これまで開発を行ってきたアクセラレータについて実機での計測結果および設計時の評価結果をもとに電力性能の推定を行った。これは、MN-Core (12FFC プロセス)、MN-Core2 (N7 プロセス)、MAU-Shuttle (N5 プロセス)について、MAB と呼んでいるプロセッサブロックの電力性能と、その中の演算器の部分のみの電力性能をまとめたものを表 4-6 に示す。MAU-Shuttle についてはシミュレーション、他は実測である。

表 4-6: アクセラレータ・テクノロジーノード別推定電力性能

アクセラレータ名称 (動作電圧)	Technology Node	(MAB)電力性能 (TFLOPS/W@FP16)	演算器のみ (TFLOPS/W@FP16)	Target Clock (MHz)
MN-Core (0.55 v)	(12 nm Node)	1.82	NA	500
MN-Core2 (0.55 v)	(7 nm Node)	2.75	6.26	610
MAU-Shuttle (0.65 v)	(5 nm Node)	NA	8.55	1100

この数値と、プロセッサボード全体の消費電力を使い、さらに半導体技術の進歩を考慮することで、下に、次世代システムの電力性能を見積もる。

4-4. 独自アクセラレータアーキテクチャの検討

現時点で、アクセラレータのアーキテクチャとしては様々なものが提案・実装されている。本節では、まず 4-4-1 節でそれらを概観したあと、4-4-2 節以降で独自アクセラレータアーキテクチャについて検討する。

4-4-1. 既存アクセラレータアーキテクチャの比較

以下のものを検討対象とする。

① NVIDIA GPU
② PEZY-SCx
③ Sunway SW26010(Pro)
④ Intel Max GPU
⑤ MN-Core

これらについて、命令処理方式とメモリ階層、コア間通信の実現方法、結果としての面積・消費電力の観点から考察する。

4-4-1-1. 命令処理方式

ここで比較するのは、SIMD か MIMD かである。PEZY-SCx と Sunway は完全な MIMD である。Sunway はコア内で SIMD を採用しており、SIMD 幅は 256 ビット(Pro では 512 ビット)である。PEZY-SC2 まではスカラー演算、SC2 は 128 ビット SIMD である。これらにおける SIMD 方式は x86 の SSE/AVX 等と同様であり、１命令で処理されるデータ幅が広い。レジスタ内では要素毎の独立な並列演算となる。メモリアクセスはおそらく unaligned load はサポートしているものと思われる。ストライドアクセスや間接アクセスについては不明である。

PEZY-SCx では最大８スレッドが時分割でサイクル毎に切り替わって動作する。この機能により、演算や L1 アクセスのレイテンシを隠蔽でき、コンパイラによる命令スケジューリン

グやハードウェアによる OoO 実行を不要にしている。Sunway ではこのような SMT のサポートはなく、OoO 実行もリネームレジスタもないため、高い性能を出すには物理レジスタを注意深くアサインする必要がある。

NVIDIA GPU では、SM という単位内での SIMD である。A100 では、１チップに 128 個の SM があり、これが FP32 Cuda コア 64 個、Tensor コア４個を持つ。

１つの Tensor コアは、16 ビットの場合4×4の行列に対して$A \times B + C$の演算を行う。従って、データ幅は 256 ビット(積算項はその２倍)である。これが４つあることでデータ幅は 1024 ビット、積算項については 2048 ビットとなる。演算数は 256FMA である。

Cuda コアにおける行列演算でない FP32/FP64 演算の性能は Tensor コアによる行列演算性能の 1/16 であるので、おそらく FP32 ユニットは実装されておらず、FP16 ユニットが SM 当たり 16 個あるものと推測される。レジスタファイルは SM 当たり 256 kB である。

NVIDIA の資料では、Tensor コア毎に L0 Ｉキャッシュ、Warp Scheduler と Dispatch unit を持つことになっているので、この場合 SM の中で Tensor コア毎の４ユニットが MIMD 動作し、データ幅は 256 ビットということになる。つまり、AVX(2)命令を持つ x86 プロセッサのような、256 ビット幅の SIMD 命令を持つプロセッサの集合体とみなすこともできる。

NVIDIA GPU の特徴は、Tensor コア毎に 64 kB と巨大なレジスタファイルを持つこと、この巨大なレジスタファイルを利用して、非常に多数のスレッドを起動できることである。ここで注意するべきことは、この「スレッド」は SIMD ユニットの中で動いているので、通常のスレッドのような MIMD 動作するものではなく、ソフトウェアから見える SIMD ユニットの幅が増えたようなものであることである。これは、ハードウェアのレイテンシを効率的に隠蔽する効果を持ち、またアプリケーションの並列性の記述を簡潔にできる、極めて有効な機構である。

主記憶アクセスに対しては間接アクセス、すなわち、その時のスレッド毎に独立なアドレスを持って主記憶アクセスができる。これはもちろんレイテンシが高く、アドレスの分布によってはスループットも低いが、スレッド数が大きい時にはレイテンシは隠蔽される。

Intel MAX GPU は、単一の Xe コアに Vector Engine (VE)、Matrix Engine (XMX)がそれぞれ 8 個搭載され、VE は FP64/32/16 演算を 32/32/64 個、XMX は FP16 で 512 演算となっている。一方、データ幅は 512 ビットと記載されている。FP32 に対してはサイクル毎に 16 データで演算幅とあっているが FP64 では不足しているように見える。

Intel 発表資料ではチップ全体で FP64/32 52 TFLOPS、レジスタバンド幅 419 TB/s、L1 バンド幅 105 TB/s となっており、VE 数は 1024 であるので動作クロックは 1.6 GHz、ベクトル演算のデータ幅は FP64 で 1024 ビットとなる。FMA 演算では２演算に対して３入力１出力であり、サイクル毎に VE1 つで 4096 ビットのレジスタアクセスが発生するはずであるが、Intel 資料からは１サイクルで VE がアクセスできるレジスタは 2048 ビットで不足している(FP32 では問題ない)。

レジスタ構成、スレッドをサポートしているかどうか、主記憶アクセスにスレッド毎の独立アドレスをサポートするかどうか等は不明である。但し、レジスタファイルは Ve 当たり 64 kB と巨大であり、NVIDIA GPU と同様の SMT サポートがある可能性が高い。

MN-Core は完全 SIMD であり、チップ上の全てのユニットが同期して同一命令を処理する。間接アクセスは各ユニットが持つローカルメモリに対して行われる。

アプリケーションの実行効率については、MIMD が高く、SIMD では低いのではないかと考えがちであるが、これに明確な根拠があるとはいいがたい。その理由の１つは、MIMD といいながら、PEZY- SCx 以外の全てのプロセッサはコア内や SM 内は SIMD であることである。このため、条件分岐はマスク演算に置き換えられる。MIMD が有効に働くためには、プロセッサコアや SM 単位で違う内容を、条件分岐の形ではなく実行するようなコードである必要がある。これは、通常のループ並列化や OpenCL、Cuda 等による並列化で発生することは稀である。

特にコア数が大きい時には、並列化の効率に大きな影響を持つのは同期や放送・縮約等の大域通信のオーバーヘッドである。これについては 4-4-1-4 で述べる。

4-4-1-2. メモリ階層

全てのプロセッサはメモリ階層を持つが、その実装はプロセッサ毎に大きく異なる。以下それぞれについて述べる。

PEZY-SCx はノンコヒーレントな３層キャッシュを持つ。SC の特徴は、キャッシュラインが L2、L3 とメインメモリに近づくほど大きくなることであり、メモリコントローラやキャッシュコントローラを複雑にせずに高い実効転送バンド幅を実現している。ノンコヒーレントなので、他のコアが変更したデータを触るためには明示的にキャッシュをフラッシュする必要がある。フラッシュ命令は階層毎に用意されている。

もうひとつの特徴は、各 PE がキャッシュ階層とは独立にローカルメモリを持つことである。

Sunway では、プロセッサは１個の Management Processing Element(MPE)と 64 個の Computer Processing Element(CPE)に分かれており、MPE は I キャッシュと L1、L2D キャッシュを持ち、CPE は I キャッシュを持つが D キャッシュは持たない。その代わりに 64 kB のローカルメモリを持ち、主記憶とローカルメモリの間は DMA でデータ転送する。主記憶のストライドアクセスはあると推測される。間接アクセスはなく、必要なら複数の DMA を起動する。

64 個の CPE は8×8の２次元格子構造を持ち、x 方向、y 方向のそれぞれで Point-to-point、放送、総和をレジスタ間通信で行うことができる。この機能は並列化効率の向上に非常に有効である。

NVIDIA GPU は PEZY と同様なノンコヒーレントな階層キャッシュを持つ。ラインサイズは 128 byte 固定である。L1 を分割してローカルメモリとして使用できる。

Intel MAX GPU は階層キャッシュである。コヒーレンシサポート、ラインサイズは不明である。L1 を分割してローカルメモリとして使用できる。

MN-Core は、階層メモリ構造はとるがキャッシュではなく、階層間のデータ転送を命令で明示的に与える。このデータ移動命令も SIMD である。階層間での放送・総和等もサポートする。

4-4-1-3. オフチップメモリ

全てのプロセッサで、なんらかの形のオフチップメモリをつけている。表 4-7 に外部メモリの主要なパラメータについてまとめる。

表 4-7: オフチップメモリ構成

	ピーク演算性能(FP64)	ピークメモリ転送速度	B/F	備考
PEZY-SC2 4.1	4.1 TFLOPS	95 GB/s	0.023	DDR3
Sunway SW26010	3.06 TFLOPS	137 GB/s	0.045	DDR4
NVIDIA A100	19.5 TFLOPS	1.56 TB/s	0.080	HBM2
Intel Max GPU 52	52 TFLOPS	3.2 TB/s	0.062	HBM2e

(MN-Core については省略する)

Intel MAX GPU は、FP64 のベクトル演算に対する値としている。FP32 等では行列演算があり、FP64 でも行列演算のサポートがあるとピーク性能が２倍、B/F は半分の 0.03 になる。

「京」で B/F = 0.5、「富岳」でも 0.36 を維持したことを考えると、これらのアクセラレータでは B/F の値が非常に低くなっていることがわかる。特に、Intel Max GPU では L2 キャッシュの B/F も 0.25 と極めて低いものになっており、アプリケーション性能へのインパクトがあるのではないかと想像される。

4-4-1-4. コア間通信について

前節で、メモリ階層についてハードウェアの観点からまとめたが、本節ではどのような操作にどのようなオーバーヘッドがあるか、という観点からチップ内コア間通信がどのように実現されるかをまとめる。

* Point-to-point 転送

Sunway と MN-Core 以外の全てのプロセッサで、キャッシュを経由する。このため、キャッシュラインのフラッシュ、同期、リードのステップが必要となり、オーバーヘッドは大きくなる傾向がある(キャッシュフラッシュには容易にマイクロ秒オーダーの時間がかかる)。

Sunway では、64 個の CPE グループの中では、任意のコア間で x 方向、y 方向の２ステップのレジスタ間通信によってデータを送ることができる。このレイテンシはナノ秒オーダーであり、非常に高速である。

MN-Core では、各階層の共有メモリを経由することでデータ転送が可能である。SIMD である関係上コア間でランダムな通信では多数回のアクセスが必要になるが、規則的な移動ならば効率的な転送が可能である。

* 同期・放送・縮約

Sunway と MN-Core 以外の全てのプロセッサで、キャッシュを経由する PtP 通信によって実装される。このため、関係するプロセッサ数の対数オーダーのオペレーションが必要になり、レイテンシは極めて大きくなる。NVIDIA GPU では典型的には 100 マイクロ秒程度が総和にはかかる。この時間は多くのアプリケーションでは致命的といえるほどに長く、アクセラレータの広い範囲のアプリケーションへの適用を妨げる大きな要因になっている。

Sunway では、64 個の CPE グループの中では、縦、横の２回のオペレーションで同期・放送・縮約ができる。このレイテンシはナノ秒オーダーであり、非常に高速である。

MN-Core では、全体 SIMD であるため同期は不要である。また、各階層の共有メモリ経由で放送・縮約が可能であり、チップ全体に対してナノ秒オーダーでの放送・縮約が可能である。

コア間縮約のオーバーヘッドは、多くのアプリケーションで並列化効率に直結する。単純な例としては行列ベクトル積がある。コア間縮約が遅いため使えない場合には、結果の１要素の演算は１コアで行う必要があり、行列の１次元方向の要素数よりも多くのコアを使うことは難しい。縮約が十分速いなら、１つの結果要素のための内積演算を複数コアで並列に行えるため、並列化効率が飛躍的に向上する。また、多くのアプリケーションで全体での最大値や総和といった演算が実際に発生する。典型的な例は有限要素法の反復計算で現れる内積演算である。MPI によるノード間通信での総和は、近年は IB スイッチが縮約をサポートすることもあり高速化しており、チップ内での総和のサポートは必要度が高いものになっている。

4-4-2. 電力コストについて

ここでは、命令実行方式、メモリ階層構造およびオフチップメモリについて、消費電力の観点から検討する。

まず、MIMD/SIMD の選択と電力コストについては、命令フェッチ・デコード・スケジュールユニットのコストを考えるとその個数が少ないことが望ましいのは自明である。但し、汎用 CPU のようなコア内 SIMD 命令では、無制限に SIMD 幅を増やすことはどこかの時点で消費電力増加につながる。これは、レジスタ内のシャッフル命令等がビット幅に比例以上のコストがかかること、演算ユニット、メモリユニットともに物理的に大きく、配線長が長くなるため、配線遅延をカバーするためには大電力のドライバが必要になること等による。これを避けるためには、原理的にはシャッフル命令を排除、またはそのスループットを下げ、さらにメモリ・レジスタ間転送で非アラインド転送を排除して、演算器の近くに L1D を持ってくればよい。これはしかし、キャッシュ階層を持つアーキテクチャでは困難である。

これらの観点から、「電力コストだけを考えた時には」命令実行方式としてはなるべく大きな単位での SIMD 方式が望ましいが、幅広い SIMD 方式ではメモリ階層の構造も合わせて考える必要があることがわかる。

次に、メモリ階層構造について検討する。4-1 節の数値からは、2029 年頃のアクセラレータについて、半導体技術の進展、アーキテクチャの進歩から想定されるボードレベルの電力当たり性能は 500 GF/W となる。この値から、「１演算につきデータをどの程度の距離チップ内で移動可能か」が決まる。チップ内の電気信号によるデータ移動に必要な消費電力は、配線長さ 1 の単位長当たりのキャパシタンス c と信号の電圧振幅 V によって、以下のように与えられる。

$$w = \frac{cV^2}{2}$$

配線からグラウンドプレーンまでの距離と配線の幅ないし高さが同程度で絶縁体の誘電率があまり大きくない時、c は次式の程度である。

$$c = 0.2(\mathrm{fF}/\mu\mathrm{m})$$

電圧振幅 V を 0.6 V、配線長を 10 mm とした時、W は 0.36 pJ である。従って、64 ビットのデータを 5 mm 移動すると、12 pJ のエネルギーが消費される。

仮に電力性能の目標を 1 TFLOPS/W とすると、これは 1 演算当たり 1 pJ である。すなわち、64 ビットデータを 0.43 mm 移動するだけで 1 pJ 消費する。B/F = 1 の LLC があるとして、演算器からメモリセルまでの平均配線長が$0.43 \times 8 = 3.5$ mm を超えると、LLC までのデータ移動だけで 1 TFLOPS/W を実現するための電力をすべて消費してしまうことがわかる。従って、ある程度の規模でチップ内コアから物理的に共有される LLC を持つ限り、2029 年頃に半導体技術的には実現可能と考えられる 1 TF/W を実現するのは実際には不可能である。

これに対する対応としては以下のようなものが考えられる。

① 共有キャッシュを排除し、Sunway のようなメインメモリへの DMA のみ、あるいは MN-Core のようなチップ内ネットワークのみとする。
② キャッシュメモリのバンド幅を下げる。
③ ３次元実装、チップ上光通信等の技術でデータ移動のエネルギー消費を下げる。
④ 上の(1)-(3)の組合せ等。例えば、PEZY-SC のようにキャッシュとローカルメモリを組み合わせて、さらに DMA やチップ内ネットワークも持つ。

消費電力とコストだけを考えると①が望ましいが、既に GPGPU にポートされているアプリケーションや、特に OpenMP 等で記述されているアプリケーションのポータビリティを考えると、キャッシュがあるメモリ構成が望ましい。一方、OpenACC の場合には配列のレイアウト等を指定できることもあり、むしろ DMA やチップ内ネットワークのほうが効率的な実装ができる。

最後に、オフチップメモリについて検討する。

HBM3 を使った場合に、データ転送のエネルギー消費は 2-3 pJ/bit と言われている。これは、DRAM チップ上、サブストレート上、プロセッサチップ上でそれぞれ 5-10 mm 程度のデータ移動があり、特に DRAM チップ上では駆動電圧をあまり下げられないことから原理的な限界といえる。このことから、DRAM へのデータ移動の消費エネルギーが全体の 20%、目標電力性能が 1 TF/W とすると、１演算当たり DRAM アクセスの消費エネルギーが 0.2 pJ、移動できるデータは演算当たり 0.07-0.1 ビット、B/F の値としては 0.008-0.0125 となる。

B/F が 0.01 ということは、オフチップメモリ 1 語アクセスにつき 1 千演算程度しないとメモリバンド幅リミットになる、ということである。大規模な密行列の操作が主体となるアプリケーションではそのような状況もありえるが、大多数のアプリケーションではこれは現実的ではない。

このことは、HBMx の現在の実装の延長では、多くのアプリケーションが有効に利用できるオフチップメモリをアクセラレータにつなげることは困難である、ということを意味する。

従って、より低い消費電力でアクセスできるオフチップメモリ実装方式の検討は必須である。そのオプションとしては以下のようなものが考えられる。

① HBMx メモリの設計はそのままで、ロジックダイ上に３次元実装する
② HBMx メモリの実装を変更し、特に DRAM ダイ上でのデータ移動を減らす
③ カスタムあるいは今後期待される標準品 DRAM をロジックダイ上に３次元実装する
④ SRAM ダイをロジックダイ上に 3 次元実装する

本調査研究では、これらのオプションについて、消費電力とアプリケーション性能への影響の観点から比較検討した。今年度は各ベンダーの開発状況をアップデートし、可能な性能・容量を推定した。

4-4-3. メモリ階層のオプション

本調査研究では、人的な資源の制約もあり、チップ内メモリ階層については①のチップ内ネットワークをベースにしてアプリケーション実装はどのようになるか、キャッシュに比べてどのような優劣があるかを検討する。オフチップメモリについては、コスト・バンド幅の観点からは最善といえる③のカスタム DRAM メモリの３次元実装をレファレンスとし、他のオプションではどのようになるかを比較検討した。

4-4-4. MN-Core 後継のベースラインアーキテクチャの決定

ここまでのアーキテクチャ、メモリ階層の検討結果を踏まえ、本調査研究でのアプリケーション評価、アーキテクチャ評価のレファレンスとなるアーキテクチャを策定した。

電力性能については、MN-Core は TSMC の 12FFC プロセスで製造され、ボードレベルで FP6470 GF/W の性能を実現している。従って、単純に TSMC のノミナルな低電力化率を使うと、N2 プロセスで 434 GF/W、N1.4 では 620 GF/W となり、アーキテクチャ改良なしで GPU 等の世界のトレンドよりも高い電力性能が実現できることになる。実際には動作電圧の設定等の問題があり、ここまで上がることはありそうにないが、ここから非常に大きくずれるものではない。従って、いくつかのアーキテクチャ改良を加えることで、FP64 で<<機密情報>> GF/W を N1.4 プロセスで実現することは可能と考える。

ベースラインアーキテクチャのパラメータは以下のようになる。これは N2 世代を想定したものである。

① 演算器は倍精度4×4(単精度 8×8、FP16 16×16 BF16/Int8 32×32)の行列ベクトル乗算ユニット
② <<機密情報>>
③ <<機密情報>>
④ <<機密情報>>
⑤ <<機密情報>>
⑥ <<機密情報>>
⑦ <<機密情報>>
⑧ <<機密情報>>

昨年度の想定アーキテクチャと比べて、プロセスを N1.4 から N2 に一世代戻し、その代わり 2 ダイを 1 パッケージに入れる構成とした。これは、各社がマルチダイの GPU の出荷を始めており、インターポーザーやパッケージの価格が下がってきているからである。また、

3DDRAM の容量・速度を大きくした。これは、後で述べるように、3DDRAM 開発を始めたベンダーが増え、より最先端に近い DRAM プロセスが利用可能になる見込みがたってきたからである。アクセラレータの外部メモリは、プロセッサコアのローカルメモリの延長として実装される。すなわち、キャッシュ階層ではなく、プロセッサコアが直接アクセス可能な単一の DRAM ブロックが各プロセッサコアに直結するものとする。

4-4-5. 3D 実装メモリの検討

まず、昨年度の検討について概要を述べ、そのあとに今年度のアップデートをまとめる。昨年度は DRAM ベンダーに 3DDRAM マクロの設計、電力シミュレーションを委託し、実現性と性能について様々な面からの検討を行った。想定した構成は、DRAM ベンダーが提供するマクロを２次元的にしきつめたセミカスタム DRAM とロジックダイのハイブリッドボンディングによる接合である。DRAM は高密度(現在10 μｍピッチ、ポスト「富岳」時点では5 μｍピッチ)の TSV で４ないし８層程度スタックされるとした。現在 HBM 等で使われている TSV は 50 μｍ程度のピッチであり HBM ダイ当たりパッド数は数千程度だが、この技術ではさらに２桁程度多いパッドが実現できる。このため、HBM では必要になっている DRAM ダイ上でのシリアル・パラレル変換が不要になり、また多数の小さなマクロ毎にパッドを置くことで DRAM 上でもロジックダイ上でもデータ移動を小さくできる。この結果、連続アドレス（同一ページ内)リードについては現在の製造技術で 0.6 pJ/bit 以下のアクセスエネルギーが実現可能という結果が得られた。4 pJ/bit 程度である HBM3/3e に比べて極めて大きなエネルギーの削減である。

実現性については以下の点を検討した。

① 構成方法。具体的には、基板側にくるのがロジックダイか DRAM ダイか
② パッケージ外に引き出す信号の SI
③ 電源供給方法と PI
④ 冷却方法
⑤ DRAM および 3D 接合の欠陥発生率と冗長設計手法

以下これらについて概要を述べる。

4-4-5-1. 構成方法

DRAM を含めた 3D 積層の構成方法として、単純には

a) ロジックダイを下(DRAM とパッケージのサブストレートの間)に置く
b) DRAM を下(ロジックダイとパッケージのサブストレートの間)に置く

の 2 通りがありえる。近年の CPU や GPU では、SRAM ダイとロジックダイの集積が使われるケースが多くなっているが、例えば AMD の Zen 4 CPU では a)、同じ AMD の MI-300X GPU では b)、Intel MAX GPU では(a)とどちらも使われている。但し、Zen 4 CPU では「Face down」と呼ばれる、配線層が基板側になるアプローチがとられており、SRAM ダイとプロセッサダイの接続はプロセッサダイに TSV をあけている。この TSV のピッチが比較的大きいためか、3D 接続が可能なように設計された Zen 4 コアはこの機能を削除した Zen 4c コアに比べて 1.5 倍程度の面積になっている。

一般には、a)を採用すると以下の問題が発生する。

a1) ロジックを Face up で実装する必要があるため、電源、信号等の基板側にいく全てのパッドが TSV で実現される必要があり、ダイエリアを消費する。また、SERDES や PCIe 等の高速信号では、負荷として加わる TSV とそこまでの配線を考慮する必要がある。元々の IP 自体が Face down の実装を前提にパッド配置を決めているので、物理実装の新規開発が必要になる可能性もある。
a2) DRAM ダイがロジックダイとヒートシンクの間に入るため、ロジックダイの温度が上昇する。
a3) DRAM の電源供給もロジックダイの TSV を経由するため、ロジックダイのエリアを消費する。

一方、b)を採用すると以下の問題が発生する。

b1) ロジックダイと基板間の全ての信号が積層した DRAM の TSV 経由となるため、そのインピーダンスを考慮する必要がある。
b2) ロジックダイの電源のために DRAM ダイに非常に多くの TSV が必要となり、エリアを消費する。また、カスタム設計が必要になる可能性がある。
b3) DRAM の主な冷却パスがそれ自体高温なプロセッサダイ経由となるため、DRAM の温度が非常に上がり、安定動作が困難になる可能性がある。

これらについて、シミュレーションによって検討した。詳細は以下の項目で述べる。

4-4-5-2. パッケージ外に引き出す信号の SI

上の a)、b)双方について spice シミュレーションによる評価を行った。シミュレーションでは、a)の場合には追加されるロジックダイ上の配線と TSV、b)の場合には追加されるロジックダイ上の配線と DRAM ダイ上の TSV を加え、基板上の配線までを考慮したシミュレーションを行った。信号としては PAM4 を採用した PCIe gen6 の信号を例にした。

結果として、b)の場合でも、TSV の配置等を配慮すれば十分な信号品質が得られることがわかった。これは、逆にいうと、この配慮が必要ということであり、b)の場合にロジックの高速信号については十分なケアが必要ということだが、原理的な問題はないことが示された。

4-4-5-3. 電源供給方法と PI

ロジックダイの消費電力が圧倒的に大きいため、b)のケースについて検討を行った。今回検討したマクロの場合には、TSV に使用できるエリアほぼ全てを使い、DRAM 信号以外を基本的にすべき電源とグラウンドに使っても、DRAM TSV での電圧降下がまだ大きいという結果になった。これは、ロジックダイのコア電圧 0.4 V、消費電力 800 W で電流 2000 A を想定した場合である。原理的には TSV エリアと数を増やすことで対応可能ではあるが、a)のアプローチのほうが好ましいといえる。ただし、ここまで面積あたりの消費電力を増やす必要はなくなってきている。

4-4-5-4. 冷却方法

ロジックダイの消費電力を 800 W 程度とした場合、発熱が一様という単純な仮定の下では a)のケースでのロジックダイの DRAM ダイが上にあることでの追加の温度上昇が 5.1 K となった。これ自体は十分小さいが、発熱が完全に一様ではないことには注意が必要である。MN-Core の場合には、通常の CPU とは異なり、演算器とそれに接続したローカルメモリが

チップエリアの大半を占めるため、発熱は一様に近いが、PCIe 等の IP では局所的に発熱が高いエリアが存在する。これらについてより詳細な熱シミュレーションが必要である。

b)のケースではこの問題はなく、ロジックダイの消費電力が非常に大きい場合には b)のアプローチが安全であるといえる。

4-4-5-5. DRAM および 3D 接合の欠陥発生率と冗長設計手法

高密度のハイブリッドボンディングを使うためには、現在のところで各社とも WoW(Wafer-on-wafer)の工程を必要としている。ポスト「富岳」の時期には CoW(Chip-on-wafer)や CoC(Chip-on-chip)の手順も可能になってくる可能性があるが、ここでは WoW を前提に欠陥発生率とその対応についての検討を行った。

欠陥は、基本的に以下の 4 箇所で発生するものと考えられる。

① ロジックダイ自体
② DRAM ダイ自体
③ DRAM ダイの TSV 加工と DRAM ダイ間のハイブリッドボンディング
④ ロジックダイと DRAM ダイのハイブリッドボンディング

WoW の場合、ロジックダイ、DRAM ダイに事前に十分なテストをするのは困難であるため、単純なやり方では総合的な良品率がこの 4 箇所での良品率の積になってしまい、非常に悪くなる。従って、これを防ぐ必要がある。

防ぐための基本的なアプローチは冗長設計である。MN-Core の場合には、既に PE レベルで冗長設計を行っており、ある程度の良品率の改善を得ている。また、SRAM マクロ等ではマクロ内でも冗長設計が採用されている。DRAM についても、マクロ内での冗長性とマクロ自体の冗長性の両方を適用する。

問題はハイブリッドボンディングであるが、これについては、DRAM の制御線については複数の TSV を 1 シグナルに割り当てることでエラーレートを減らし、本数の多いデータ線については代替線を用意することでエラーレートを減らすことで対応可能であるという結論になった。

但し、Cu マイグレーション等による経年変化等についてはまだ十分な検討はできていない。

4-4-5-6. まとめ

3DDRAM について、実際にハイブリッドボンディングでのテストチップ製造経験のあるベンダーにシミュレーション等の評価を委託して、設計上の問題点について検討を行った。ポスト「富岳」のような極めて発熱が大きい場合には、ロジックダイが上(基板と反対側)にくるアプローチが望ましいが、この場合には DRAM ダイを通した電源供給について TSV の増強等の配慮が必要であること、また PCIe や SERDES 等の高速信号についても事前の検討が必要であることが確認できた。また、WoW の製造方法をとることによる不良品率の増加については、基本的に各部での冗長設計により対応するべきという結論になった。なお、発熱分布等についてさらに検討が必要であるが、ロジックダイが下にくる方式を否定するものではない。

これらから、DRAM についてセミカスタム設計に対応するベンダーであれば、WoW による超多ピン接続で、20 TB/s を超える高いバンド幅と 0.6 pJ/bit を下回る低い消費エネルギーを両立させることは可能であるといえる。しかしながら、当初計画していた試作による評価は行うことができなかったので、良品率、信頼性についての評価は十分とはいえない。

また、現在のところ DRAM についてセミカスタム設計に対応するベンダーは最先端の DRAM プロセスを持つファブではなく、ニッチ向けの供給を行っている二線級のファブである。このため、消費電力はともかく、面積当たりの容量があまり大きくない、という問題が現状ではある。これについては、これらのファブのプロセス改善と、最先端の DRAM ファブのこのような設計への対応の両方について、ある程度の進展は期待できると考える。

4-4-5-7. 2024 年度アップデート

2023 年度末の段階では、実際の検討を依頼したベンダー(以下 A 社)以外でカスタム設計の 3D 積層 DRAM を製造することは現実的ではなかったが、2024 年度末までに B 社、C 社の 2 社が利用可能になってきた。これらのベンダーの 2024 年度末時点での技術開発状況を表 4-8 に示す。

表 4-8: カスタム 3D 積層 DRAM ベンダー比較

	A	B	C
プロセスルール(nm)	<<機密情報>>	<<機密情報>>	<<機密情報>>
アクセスエネルギー(pJ/bit)	<<機密情報>>	<<機密情報>>	<<機密情報>>
容量(Mbit/mm^2)	<<機密情報>>	<<機密情報>>	<<機密情報>>

昨年度検討していたのは A 社であったが、C 社ではほぼ３倍の容量と半分程度のアクセスエネルギー(ビット当り)を達成できる見込みであり、ポスト「富岳」の時点までにさらにプロセスの微細化が期待できる。このため、レファレンス設計での容量、バンド幅を大きくしている。

4-4-6. 演算器設計の検討

演算コア、内部ネットワークは昨年度から変更していない。

4-5. 全体構成案の策定

CPU、ネットワークの検討と合わせて、アプリケーション性能のより具体的な評価と、システム全体としての検討のため、ポスト「富岳」の構成の想定案を作成した。これは昨年度からアップデートしているので、その数値を以下に示す。

4-5-1. 構成概要

1 ノードの構成を以下にまとめる。

<<機密情報>>

これらが１モジュールを構成し、８モジュールを相互接続＋制御サーバーにつないだものが１ノードとなり，それを 2500 個繋いだものが全体のシステム構成となる（表 4-9）。

表 4-9：システム構成案

＜システム構成（全体）＞

ノード数		2500 ノード
ストレージ		DDN Exascaler の改良型
	容量、I/O 性能	500 PB, 5 TB/s
ノードの接続（インターコネクト）		1.6 Tb/s Ethernet(800 Gbps x 2)
特記事項		多ポートスイッチによる Fat tree ネットワーク

＜性能＞

演算性能　倍精度（ノードあたり）	<<機密情報>>
演算性能　単精度（ノードあたり）	<<機密情報>>
演算性能　半精度（ノードあたり）	<<機密情報>>
演算性能　AI 向け半精度（ノードあたり）	<<機密情報>>
演算性能　整数演算（ノードあたり）	<<機密情報>>
演算性能　AI 向け整数演算（ノードあたり）	<<機密情報>>
メモリ容量（ノードあたり）	<<機密情報>>
メモリバンド幅（ノードあたり）	<<機密情報>>
消費電力（ノードあたり）	<<機密情報>>
演算性能　倍精度（システム全体）	10 EFLOPS
演算性能　単精度（システム全体）	<<機密情報>>
演算性能　半精度（システム全体）	160 EFLOPS
演算性能　AI 向け半精度（システム全体）	<<機密情報>>
演算性能　整数演算（システム全体）	<<機密情報>>
演算性能　AI 向け整数演算（システム全体）	<<機密情報>>
メモリ容量（システム全体の合計）	<<機密情報>>
メモリバンド幅（システム全体の合計）	2 EB/s
消費電力（システム全体）	<<機密情報>>
電力性能比　倍精度（システム全体）	<<機密情報>>
電力性能比　単精度（システム全体）	<<機密情報>>
電力性能比　半精度（システム全体）	<<機密情報>>
電力性能比　AI 向け半精度（システム全体）	<<機密情報>>
電力性能比　整数演算（システム全体）	<<機密情報>>
電力性能比　AI 向け整数演算（システム全体）	<<機密情報>>

１ノードの特徴としては、2029 年頃に想定される他のアーキテクチャでの構成に比べて、１ノードあたりの FP64 演算性能、メモリバンド幅の双方で大きく上回り、電力は同程度ないしはそれ以下になることで大幅に電力性能を向上させていること、また１ダイ当りの性能が高いために価格性能比も良好であることがあげられる。

表 4-10 にアクセラレータのみの比較をまとめる。本検討結果は、かなり楽観的に見積もった他社 GPU に比べて、

- FP64 演算の電力あたり性能でほぼ２倍
- 電力あたりのメモリバンド幅でほぼ７倍

と、特にメモリバンド幅については桁違いというのに近い高い性能を実現する。

表 4-10: アクセラレータ比較

	本検討	NVIDIA	AMD
消費電力(W)	<<機密情報>>	1000	800
FP64 性能(TF)	<<機密情報>>	N.A.	207
電力効率 GF/W@FP64	<<機密情報>>	N.A.	259
メモリバンド幅(TB/s)	100	8.0	12.8
電力効率 GB/J	100	8.0	16.0

もちろん、この高い面積あたり性能、電力あたり性能は、アクセラレータ１ノード内での階層キャッシュによる共有メモリアーキテクチャから分散メモリとチップ内ネットワークを持つ分散メモリアーキテクチャへの移行によって実現されるものであるが、アプリケーション実行効率の観点からもこの移行は必要なことであり、また実行効率の予測・制御が容易になるという大きな利点があると考えている。

⑤CPU 評価

（再委託先 国立大学法人 東京大学）

RISC-V 等による独自実装の汎用 CPU および商用 CPU の評価のため、昨年に引き続き、主にシミュレーションによるアプリケーション実行時性能評価を行った。特に、現在搭載を検討しているいくつかの企業で開発中の RSIC-V CPU について性能解析を中心に評価を行った。また、RISC-V における未成熟なソフトウェアスタックを補うためのバイナリトレースレーション技術の検討を行った。以下ではこれらのうち、主に性能解析の結果について述べる。

[背景] 本評価において RISC-V CPU を主に評価したのは、低コストでアクセラレータと密結合したシステムを作ることができるためである。現在高性能 CPU の市場で支配的な x86-64 CPU は IP として設計が提供されることはなく、完成したチップしか供給されていない。このため、アクセラレータとは外部バスを通じて通信せざるを得ず、通信バンド幅やレイテンシにボトルネックを生じる。別の代表的な高性能 CPU である ARM CPU は IP としても提供されているものの、その価格は非常に高く、また内部を独自にカスタマイズする事は基本的にできない（あるいは極めて高額なライセンス料を必要とする）。これに対し、RISC-V CPU は比較的柔軟な形で IP やチップが提供されており、低コストで独自のシステムを作成する観点では非常に魅力的である。

一方で、RISC-V CPU のハードウェアやそのソフトウェアスタックはまだ成熟しているとは言えず、実際に高性能計算等に用いた場合の性能はよくわかっていない。そこで本評価では代表的な RISC-V CPU のベンダーである Tenstorrent 社と SiFive 社の

協力を得て、それぞれの提供している CPU の性能についての調査や、一部データ提供による我々独自の評価・解析を行った。

[性能評価方法] 性能評価は基本的に昨年度までに確立した手法に基づくが、今年度は測定オーバーヘッドの低減による精度の向上と、より新しい商用 CPU の評価に基づく将来の性能の推定を行った。

評価ではプログラム内に含まれるシーケンシャルな処理に着目し、それらを CPU で実行した場合の性能を評価した。シーケンシャルな処理に着目するのは、プログラム内のデータ並列性の高い処理はアクセラレータで実行する想定であるためである。アムダールの法則として知られるように、プログラムの並列化を進めていくと最終的に並列化が難しいシーケンシャルな部分が残り、性能に対して支配的となる。従って、計算機システムとしては処理の大半を担うアクセラレータに加え、シーケンシャルな処理を行う CPU の性能も重要となる。

評価では昨年に引き続き大規模並列粒子法シミュレーションのための汎用高性能ライブラリである FDPS (https://github.com/FDPS/FDPS) を用いた。FDPS を対象としたのは、依存関係などが少ないため発展途上の RISC-V のソフトウェアスタックでもバイナリの生成が容易である一方で、並列化可能な部分とシーケンシャルな部分をそれぞれ適度に含んでいるためである。

この評価では以下のような点が課題となる：
・x86-64 CPU と RISC-V CPU の間で駆動周波数や同じ処理を行うために必要な命令数が違うため、単純に実行時間を比較するだけでは意味がある比較とならない。
・x86-64 CPU の実機において関数毎の実行サイクル数を正確に低オーバーヘッドで測定することは簡単ではない。
・各関数は呼びだし毎に実行サイクル数が大きく変化するため、同じ呼び出し同士で比較を行う必要がある。
・機密保持の観点より、各企業において開発中の CPU 上での測定を我々が直接行う事ができない。

上記の課題に対し、まずプログラムに対して一定の変更を行い、測定対象となる関数の呼び出し回数と実行サイクル数を x86-64 CPU 上で軽量に取得できるよう一定のコードを挿入した。この測定のために挿入するコードは、昨年度までは Linux システムコールを介してパフォーマンスカウンタの値を取得するものを使用していたが、その後の評価により、使用するカーネルのバージョン等に依存してサイクル数等がかなり変動することがわかった。このため、本年度はカウンタに直接アクセスを行うなどの軽量な方法で、より精度の高い測定を行う方法を検討した。

測定を行う X86-64 CPU としては、これまで測定を行ってきた AMD 社の Zen2 世代のものに加え、本年度はより新しい Zen4 世代のものを測定した。RISC-V CPU については企業と交渉の結果、これまでにプログラム内の各命令がリタイアしたサイクル数の系列を得ることができているため、これを解析して x86-64 CPU における実行結果の該当する部分を割り出し比較した。これらにより、該当する関数同士の特性を適切に比較できる。

[評価結果] 特に Tenstorrent 社の実装は x86-64 CPU と比較してクロックサイクルあたりの実行命令数がかなり高く、比較的低い動作周波数でも高い性能が達成できる見込みである事がわかった（機密保持の観点より、詳細な性能差などの分析結果はここでは省略する）。このため Tenstorrent 社の実装は、電力効率上でかなり有利であることが期待できる。

また、AMD 社の CPU については、昨年度の評価からは２世代進んだものを新たに用いて評価したが、クロック周波数の向上による総合的な性能向上は確認できたものの、測定した範囲ではクロックサイクルあたりの実行命令数はそれほどには改善していない事がわかった。Tenstorrent 社の実装はクロックサイクルあたりの実行命令数の観点で Zen4 世代を含む x86-64 CPU よりも依然として相当に高い結果を示しており、当面はこの傾向が維持される見込みである。内部構造等の比較検討を行ったが、これらの差は、AMD 社（と Intel 社の）x86-64 CPU が 5GHz～6GHz 等の極めて高いクロック周波数を達成するために、同時実行命令数や各種のバッファのサイズなどを（相対的に）あまり大きくしないよう設計している事に主に起因すると推測している。

⑥アクセラレータアーキテクチャ評価
（再委託先 国立大学法人 名古屋工業大学）

アクセラレータプロセッシングエレメントの詳細な実装方式について、近年の乗算器・加算等の実装技術の進歩と半導体技術の進歩、実装する演算自体の特性を考慮した検討を行った。具体的には、まず、単一の演算を行う場合に使える技法が行列演算に適用可能かの検討および行列演算に特有に使える技法があるかの検討を行った。次に、半導体技術の進歩により、既存のレイテンシ削減 vs 回路面積増加のトレードオフのバランスがどのように変化したかを検討した。特に、浮動小数点数積和演算器（内積演算器含む）の複数のレイテンシ削減手法について、レイテンシと回路面積の両面から比較検討を行った。その結果、内積演算器では正規化に必要なコストが内積長に反比例して小さくなるため、既存のレイテンシ削減技法を積極的に取り入れることが可能であると見出した。さらに、既存の単一積和演算器では大きくなりすぎて現実的ではないと考えられてきた回路構成も再考した。また、AI 向けには非常に低精度な浮動小数点数が用いられるが、その演算器はシフトに多くのトランジスタが必要であり、その削減が重要であることを明らかにした。さらにそれを軽減するための方策として、データ交換形式・内部表現形式・内部演算精度、などの面からリーズナブルな面積性能比（面積当たりの性能）を実現できる構成を検討した。

⑦ネットワークアーキテクチャ評価
（再委託先 大学共同利用機関法人 情報・システム研究機構 国立情報学研究所）

これまでに取得したアプリケーション要求に基づき、ネットワークアーキテクチャを検討した。そして、昨年に引き続き、性能に与える影響をシミュレーションないしは通信量の推定に基づいて評価した。

これまでの相互結合網は、InfiniBand を用いた間接網やカスタム設計した直接網が用いられることが多かった。しかし、本必要帯域であれば、最新の光電融合技術が現実的に利用可能である。昨年度に調査した通り、光サーキットスイッチをデータセンターネットワークや並列計算機の結合網に導入する場合、InfiniBand などの電気ネ

ットワークと独立した系として持つツインネットワーク、電気ネットワークを用いずに光サーキットスイッチのみで構成する場合、電気ネットワーク間に光サーキットスイッチを挿入するハイブリッドネットワークがある。現状、TopoOpt[7-1]などの一部の研究開発が光サーキットスイッチのみでネットワークを構成することを目指しているが、多くはハイブリッドネットワークである。なお、ハイブリッドネットワークにおいて直接網と間接網の区分は、元となる電気ネットワークの構成に基づく。これは、光サーキットスイッチは、実際にトラヒックが流れる前に回線を設定するため、アプリケーション側からはリンクの一部のように見えるためである。

図 7-1 に光サーキットスイッチを導入した研究開発の分類を示す。図における「主目的」とは光サーキットスイッチを結合網に導入する理由である。同「ネットワークトポロジ」は元となる電気スイッチのネットワークトポロジである。同「専用 OCS(Optical Circuit Switch)デバイス」とは、3 次元 MEMS や光パッチパネルなどのコモディティとして商用化されている光サーキットスイッチデバイス以外のデバイスを指す。Palomar 光サーキットスイッチ[7-2]、2022 年に発表された進化した Jupiter[7-3]、Tale of Two[7-4]は、典型的な Fat ツリー構造の Google データセンターにおいてジョブ内通信遅延を削減する構成を可能とする研究成果である。一方、同目的を達成しつつコストを抑えるべく、直接網の Dragonfly ネットワークトポロジを対象にした研究である Flexfly[7-5]、 GRAP[7-6]がある。同様にトーラスにおいてサブトポロジの embedding を容易にすることで通信遅延を削減する目的を達成することができる TPUv4[7-7]などが報告されている。また、Allreduce 処理を高速化するために光サーキットスイッチのみでネットワークを構成する TopoOpt が提案されている。これらにおいて専用光サーキットスイッチを用いる理由は、商用光サーキットスイッチと比べて故障率の改善、光サーキットスイッチのスイッチ時間の短縮の２点が挙げられる。なお、いずれの光サーキットスイッチもポート数が 100 を越える高次元化を達成しており、現実的に大規模結合網の構築が可能となっている。

一方で課題としては、昨年の報告書でも簡単に指摘した通り、通信パターンの予測性、リアルタイム性への対応とそのためのソフトウェアが依然、挙げられる。

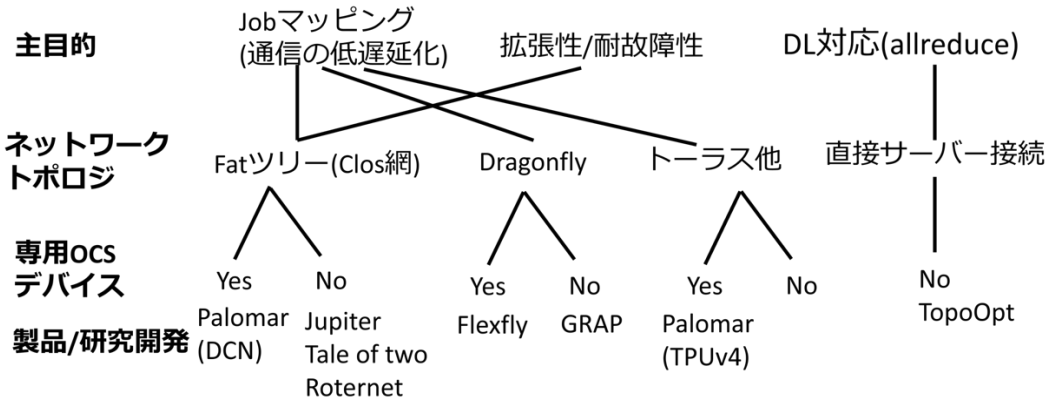

図 7-1: 光サーキットスイッチを利用したデータセンターと並列計算機の分類

図 7-1 においてネットワークトポロジがトーラス、専用 OCS デバイスの利用を想定していない区分にあたる研究開発が空欄である。しかし、異なる通信パターンを有するアプリケーションを１つの並列計算機に安価にサポートする一般的な場合、この空欄

となっている部分を埋める研究開発が重要となる可能性がある。特に通信パターンが予測できない場合、Tale of Two[7-1]のように局所的にランダム性を導入して各パケットが経由する平均電気パケット数を削減することが効果的であることが期待できる。すでに我々は、若干の光サーキットスイッチの導入によりネットワークの(経由電気スイッチ数の点での)直径、平均ホップ数とも大幅に削減できることを解明した。

現在、光サーキットスイッチの商用化、高次元化が進んだ結果、Google のデータセンターや同並列計算機での利用が進んでいるが、今後開発される並列計算機についても光サーキットスイッチの導入が効果的である場合が増えると考えられる。

[7-1] Weiyang Wang, et al., TopoOpt: Co-optimizing Network Topology and Parallelization Strategy for Distributed Machine Learning Training Jobs, NSDI, pp.739-769, 2023[7-2] Hong Liu, et al., Lightwave Fabrics: At-Scale Optical Circuit Switching for Datacenter and Machine Learning Systems. SIGCOMM, pp.499-515, 2023
[7-3] Leon Poutievski,et.al., Jupiter Evolving: Transforming Google's Datacenter Network via Optical Circuit Switches and Software-defined Networking. SIGCOMM, pp.66-85, 2022
[7-4] Yiting Xia, et al., A Tale of Two Topologies: Exploring Convertible Data Center Network Architectures with Flat-tree, SIGCOMM, pp.295-308, 2017
[7-5] Ke Wen, et al., Flexfly: Enabling a Reconfigurable Dragonfly Through Silicon Photonics, SC'16, pp.166-177, 2016
[7-6] Guangnan Feng, et al., GRAP: Group-level Resource Allocation Policy for Reconfigurable Dragonfly Network in HPC. ICS, pp.437-449, 2023
[7-7] Norman P. Jouppi, et al., TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. ISCA, pp.1-14, 2023

⑧システムソフトウェア・ライブラリ調査研究の統括およびコンパイラ検討 (再委託先 公立大学法人 会津大学)

システムソフトウェア・ライブラリ調査研究全体の統括を行うと共に、アプリケーションを記述可能なデータ並列言語のプロトタイプでのアプリケーションカーネル記述、その性能評価を行った。

MN-Core 向けの OpenCL API をベースとしたホスト API を利用して、MN-Core 用の演算カーネルを記述する C 言語をベースとしたカーネル用コンパイラの詳細設計を行った。その仕様に基づき、広く利用されているコンパイラフレームワークである LLVM を利用してカーネル用コンパイラを実装した。このコンパイラでは、Clang により C 言語で記述された演算カーネルのソースコードを LLVM IR(中間言語)に変換し、LLVM IR を MN-Core 用命令に変換する。その際、適切なレジスタおよびローカルメモリ割り当てを行う。また、カーネル内でのベクトル化を指定オプションに応じて適用する。MN-Core 用命令への変換の実装として、主として会津大学で実施した実装と、主として神戸大学で実施した実装があり、両者を比較検討した。

コンパイラの実装では、アプリケーションで必要な倍精度演算に対応した三角関数の MN-Core 用実装を行い、数学関数ライブラリの初期的な評価も実施した。

カーネル用コンパイラを利用して、積分変数変換に三角関数を必要とする、ファインマンループ積分の Quasi-Monte Carlo(QMC)積分法による性能評価を行った。このカーネルは、PE 間のデータ移動が最低限であり、並列化率が非常に高いアプリケーションである。性能は 1 秒あたりの積分点の処理数(単位は秒あたり 10^9 points)で性能評価した。結果を表 8-1 に示す。

表 8-1: QMC による数値積分の性能評価。1 秒あたりの処理数を 10^9 の単位（GPoints/sec）で示す。

積分点数（$\times 10^9$）	MN-Core (GPFN2)
1.22	0.8563657535
2.147	0.9931252071
8.599	1.189134253
12.14	1.220734277
24.29	1.246782282

このタイプのデータ移動のないアプリケーションでは、問題サイズが十分大きければ、PE の稼働率が高くなりより高性能となる。結果、MN-Core での性能は最大で約 1.2 GPoints/sec であった。この性能を GPU(NVIDIA H100)での同じ計算と比較すると、おおよそ 4 分の 1 の性能に相当する。なお、MN-Core(GPFN2)と H100 では、チップ実装の半導体プロセス世代は、それぞれ 12nm と 4nm であり、大きな隔たりがある。一方で、MN-Core 用命令のプロファイリングにより、この QMC カーネルにおける、積和命令の割合は約 25%にとどまっていることがわかった。また、命令スケジューリング上必要な nop 命令の割合も大きく、まだ性能向上の余地がある。この評価から、レジスタおよびローカルメモリの割り当ては十分最適化されているが、LLVM IR からの MN-Core 用命令への変換時に、不必要な命令をより削減する最適化が必要であることがわかった。
さらに、昨年度に引き続き、今年度もユーザーが容易に MN-Core へ移行できるようにするために、コンパイラ指示文（ディレクティブ）形式の API である OpenACC プログラミング環境の検討および開発を行った。

まず、MN-Core 対応 OpenACC の追加仕様について検討を進めた。仕様は C 言語版と Fortran 版をそれぞれ定めた。基本的に OpenACC 1.0 までの仕様に準拠し、それに加えて一部の 2.0 以上の仕様に対応、加えて MN-Core 用の独自拡張を定義する。OpenACC 2.0 以上の仕様は主に分割コンパイラ対応のためのものを優先的にサポートする。なお、OpenACC にはディレクティブ形式 API の他にランタイム関数も存在するが、ディレクティブで同様の記述が可能であることから MN-Core 対応 OpenACC には現状含めないこととした。

デバイス固有の独自拡張の内容は、MN-Core 上の各階層でのデータ分割の指示、ホストと PE 間の集団通信、PE 間の集団通信が挙げられる。これらは OpenACC が共有メモリ型を前提としているのに対し、MN-Core が分散メモリ型アーキテクチャであることから、よりハードウェアに近い制御を可能にして性能チューニングの際のユーザービリティを高めるために定義される。

現在の仕様で想定される３次元の７点ステンシル計算コードの記述例を示す（リスト8-1）。aがC言語版で、bがFortran版である。コード例では袖交換のための独自拡張API として赤字の箇所の、袖領域を指示する shadow 節と通信タイミングを指示するreflectディレクティブを記述している。なお、各階層のデータ分割は、１～３次元についてデフォルトの分割方法を定めることで、デフォルトで良い場合には記述を省略できるようにした。このコード例でも省略されている。また、独自拡張以外にも、OpenACC 標準仕様で実装依存となっている並列階層の指示などの部分について、デバイス固有の解釈を定義した。gangはPEレベル並列とし、vectorはベクトル命令レベル並列である。

リスト8-1: 想定される3次元の7点ステンシル計算コードの記述例。a）C言語版，b）Fortran版。

a)

```
#pragma acc parallel create(temp[0:128][0:128][0:128]) ¥
copy(a[0:128][0:128][0:128]) shadow(a[1:1][1:1][1:1])
for(count=0; count<N; count++) {
#pragma acc loop gang vector collapse(3)
    for(int i=1; i<128-1; i++){
        for(int j=1; j<128-1; j++){
            for(int k=1; k<128-1; k++){
                 temp[i][j][k] =
                    c1*(a[i-1][j][k]+a[i+1][j][k]
                    +a[i][j-1][k]+a[i][j+1][k]
                    +a[i][j][k-1]+a[i][j][k+1])
                    +c2*a[i][j][k];
    } } }
#pragma acc loop gang vector collapse(3)
    for(int i=1; i<128-1; i++){
        for(int j=1; j<128-1; j++){
            for(int k=1; k<128-1; k++){
        a[i][j][k] = temp[i][j][k];
    } } }
#pragma acc reflect(a)
}
```

b)

```
!$acc parallel create(temp(1:128, 1:128, 1:128)) &
!$acc copy(a(1:128, 1:128, 1:128)) shadow(a(1:1, 1:1, 1:1))
do count=1, N
!$acc loop gang vector collapse(3)
    do i=2, 128-1
        do j=2, 128-1
            do k=2, 128-1
                temp(i,j,k) = &
                    c1*(a(i-1,j,k)+a(i+1,j,k) &
                    +a(i,j-1,k)+a(i,j+1,k) &
                    +a(i,j,k-1)+a(i,j,k+1)) &
                    +c2*a(i,j,k)
    end do; end do; end do
!$acc loop gang vector collapse(3)
    do i=2, 128-1
        do j=2, 128-1
            do k=2, 128-1
                a(i,j,k) = temp(i,j,k)
    end do; end do; end do
!$acc reflect(a)
end do
!$acc end parallel
```

この仕様に基づいて、OpenACC から OpenCL 相当の低レイヤー言語へ変換する C 言語版のプロトタイプコンパイラの実装を進めた。１次元ベクトル計算については、

OpenACC から変換した低レイヤーコードをさらにコンパイルし、実機上で動作することを確認した。また、上で示したような 7 点ステンシル計算コードの例や姫野ベンチマークコードについても基本的な変換に対応させた。前者についてはアセンブリレベルまで変換することにより、性能予測を行っており、理論上の限界性能に相当する性能のコードに変換できることを確認した。

⑨DSL 検討
（再委託先 独立行政法人国立高等専門学校機構 松江工業高等専門学校）

主に粒子法用の内部 DSL である FDPS(Framework for Developing Particle Simulators)上で動作する PMMM(Particle Mesh [PM] Multipole Method)ライブラリの開発やそれを FDPS から呼ぶための仕様策定および API 開発を行った。さらに、FDPS を用いた Ewald 法の開発を行い、PMMM 法との比較サンプルコードや宇宙論的 N 体シミュレーションコードを追加した。また、FDPS の一部の API および実装を C++17 を用いて追加、変更することで、アプリケーションユーザの利便性を向上させ、将来的な機能拡張が容易になるよう改良した。

PMMM とは計算領域を荒い規則格子に区切り、FMM (Fast Multipole Method) と同様に近傍セル中の粒子からの相互作用は直接法で計算し、遠方のセル中の粒子からの寄与は局所展開を用いる事で評価する。 FFT(高速フーリエ変換)を用いて、局所展開係数は多重極展開係数を畳み込み積分を行うことで求める。これにより、従来用いられてきた PM 法に比べ、大規模なフーリエ変換を必要とせず、また高精度化も容易になっている。

FDPS では、自セルを含む近傍セル中の粒子との相互作用には Barnes-Hut Tree 法を使えるよう実装しており、オリジナルの PMMM 法に比べて効率よく計算できるようにしている。

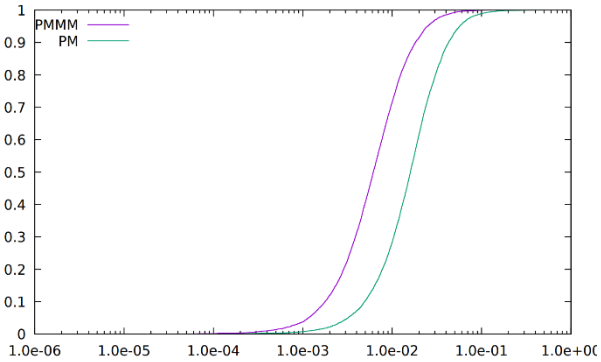

図 9-1: 規格化した累積粒子数分布。横軸は FDPS を用いて PMMM 法と TreePM 法で計算した重力加速度と Ewald 法で計算した重力加速度との差。

図 9-1 では横軸に FDPS を用いて PMMM 法と TreePM 法で計算した重力加速度と Ewald 法で計算した重力加速度との差、縦軸には規格化した累積粒子数分布をプロットした。粒子は立方体中に一様ランダムに 8192 個の粒子を配置している。また、PMMM 法では格子を 8^3、展開係数 $p = 4$ としている。ツリー法のオープニングクライテリオン θ は PMMM 法、 TreePM 法で共に 0.5 とした。TreePM 法に比べ精度が改善されていることがわかる。

⑩アプリケーション調査研究の統括
（再委託先 学校法人 順天堂大学）

アプリケーション調査研究全体を統括し、問題規模の設定、アルゴリズムの検討に対して必要なサポートを行った。また他グループへのアプリケーションからの提案・要求を整理し、コデザインを有効に進めた。それぞれのアプリケーション研究者にとって、アプリケーション全体での評価は厳しいため、基本的にはカーネル部分の評価が中心となっている。そこで、このカーネル部分での評価を元に、アプリケーション全体での性能予測をいくつかのアプリケーションで行った。具体的には、カーネル部分以外の主要な計算方法や演算量、通信量・回数などの見積もりをそれぞれのアプリケーション全体で行ってもらい、アプリケーション全体での性能の見積もりが可能になった。これにより、ポスト「富岳」でどのくらいの規模の計算が、どのくらいの時間で行えるかの見通しを出せるようになった。その結果、どのような科学的なインパクトがあるかを推測可能となった。この見積もり課程を共有することで、推測可能なアプリケーションが徐々に増えた。

また、アプリケーション開発側からは、そのままのコードが動くという意味で、会津大学を中心に開発中のコンパイラへの期待は大きいが、コンパイラのみでアプリケーションの性能が大きく改善するような期待は抱いてはおらず、そのままのコードが実際の性能はともかく、そのまま動くということに価値があることをフィードバックした。

表 10-1 に、本調査研究で対象としたアプリケーションタイプについて、計算タイプ、アクセラレータへの実装方法のタイプ、期待できる効率をまとめた。多くのアプリケーションタイプで効率が 10-15%となっているが、これは、基本にベクトル演算のピーク性能として行列演算のピーク性能の 1/4 を想定したためであり、ベクトル演算のピーク性能に対する効率は 40-60%になっている。

表 10-1: アプリケーションの分類と期待される効率。

アプリケーション	タイプ	アプローチ	性能(効率)
創薬と深層学習応用	粒子・密行列	DSL	30%
ゲノム科学	ツリー探索	汎用環境	CPU のみの 数倍
地震と構造物	FEM(EbE)	汎用環境	20%
気象・気候	FEM(EbE)	汎用環境	12%
ものづくり	疎/密行列	ライブラリ	10%
マテリアルサイエンス	密行列	ライブラリ	50%
素粒子・原子核	規則格子差分	汎用環境	15%
宇宙・惑星科学	粒子	DSL	10%

このことから、既存のアルゴリズムを高効率で実行することだけを考えるなら、ベクトル演算のピーク性能をあげるべきとも考えられるが、これは実際にはオンチップメモリバンド幅の向上を必要とし、消費電力の増加を招く。

行列乗算器を有効に利用ようにアプリケーションを変えていく方向も考えられ、もっとも自明な方向はサロゲートモデルだが、これは全てのアプリケーションで利用できるものではない。

一方、規則格子差分や有限要素差分で、内部反復を行列ベクトル積や行列行列積に書換えられる、不連続ガラーキン法を使う、といったことは考えてられるであろう。不連続ガラーキン法にはノード間の通信量や通信回数を減らせる、というメリットもあり、今後主流になっていくと考えられる。

粒子法でも、ツリーノードと粒子の相互作用をそのまま計算するツリー法ではなく、さらにノード間相互作用を導入する高速多重極法(Fast Multipole Method: FMM)では、ノード間相互作用は行列行列積で表現できる。FMMも、周期境界の場合でも Particle Mesh Multipole Method（PMMM）法との組合せで高性能を少ない計算量で実現できるようになってきており、従来使われていたTree PMやParticle-Mesh Ewaldにとってかわっていくことが期待できる。これらを考慮すると、無理にベクトル演算のピーク性能をあげるよりも、アプリケーションで行列乗算ユニットを有効に使うことが将来的には望ましいともいえる。

一方、このようなアプリケーションの変化には時間がかかるので、電力性能に大きな影響がない反応でベクトル性能をあげることは望ましい。

上記のアプリケーション分野での検討において、バイオ分野でのヒトゲノムの高速相同性解析に関するメニーコア・アーキテクチャの有効性検証も実施した。近年バイオ分野で次世代シーケンサーの能力が飛躍的に高まり、シーケンサーからのデータからゲノムを再構成し、変異を見つけるソフトウェアの高速化のニーズが高まっている。現在開発に取り組んでいるプロセッサであるメニーコア・アーキテクチャで、この速度が飛躍的に高めることが可能であることが示唆されていることから、AMD のメニーコア・プロセッサにより、どの程度の高速化が可能であるか、計測する準備を行った。一つはこのアプリケーション・ソフトウェアの開発を行った会社と協議し、ソフトウェアの使用の許諾を得ると共に、AMD 社製 64core の PC の導入を行った。その結果、ヒトゲノムの 30 倍の量のデータで行う標準的なテストで、従来の相同性解析ソフト（BWA-GATK）が約 7 分かかったものが、５分程度で処理が終わり、標準ゲノムとの差の指摘精度はほぼ同等というものであった。このことからゲノムの相同性解析に関してはメニーコア・アーキテクチャの方が、同等の性能で、より高速化可能であることを示しており、有望であることがわかった。本ソフトウェアでは、既存のソフトウェアで採用している Burrows-Wheeler 変換の代わりに、その元になっている接尾辞木そのものを用いることで、探索時のメモリアクセスの回数を大幅に減らしている。さらに、木構造を階層分割することで、分割した木はキャッシュに完全に入るようにし、キャッシュヒット率を向上させ、大きく性能を上げている。この考え方は、各プロセッサエレメントが比較的大きなメモリを持ち、それぞれが独立にメモリアクセスできる提案アーキテクチャでもそのまま利用可能である。このため、現在７分程度である x30 のデータの処理を、さらに半分以下にできる可能性があると考えられる。

⑪創薬と深層学習応用アプリケーションの検討
（再委託先 株式会社 Preferred Networks）

昨年度までで創薬と深層学習応用アプリケーションについて想定するアプリケーションを定め、既存のアルゴリズムでの演算カーネルの抽出、ノード間通信、I/O の必要量の推定を行い、新しいアルゴリズムの検討の方向を明らかにした。

今年度は分子動力学シミュレーションの性能推定の改善と、今後重要になるLLM 推論の効率推定を行った。分子動力学シミュレーションについては、MN-Core 2 上での長距離相互作用計算を行う最適化された演算カーネルを高レベルの記述から生成するシステム PIKG で実装し、演算部分の実行効率を測定した。演算カーネル自体については、ベクトル演算モードの理論限界の性能を実際に達成できることが示された。

LLM 推論については、提案システムの FP8 性能に対する B/F 値は 32PF に対して 100TB/s であり、0.003125 である。バッチサイズとして 64 を仮定すると、行列乗算時にメモリ律速にならないためのバンド幅は 0.0078125 であり、この部分で 40%、アテンション部分の時間等も考慮すると 30% 程度の実行効率となる。バッチサイズを大きくすることでより高い実行効率も期待できる。GPU では、メモリバンド幅の限界のため、実行効率をあげるためには非常にバッチサイズを大きくする必要がある。これは、並列化の手法にもよるが数千から数万になる。このため、ユーザーあたりのトークン出力速度が小さくなり、UI としては反応が遅いものになる。一方、提案システムでは数百程度のバッチサイズで十分な実行効率を実現でき、ユーザーあたりの出力トークン速度を１桁程度高めることができる。

⑫ゲノム科学アプリケーションの検討
（再委託先 国立大学法人 東京大学）

汎用 CPU での実行プロファイルとアクセラレータで実行する部分の性能評価に基づき、ゲノム科学アプリケーションについて定めたアプリケーションの性能評価を行った。今年度は、最近新たに様々な種類のゲノム配列解析データが算出されつつある状況を踏まえ、昨年度実施した Illumina 社 NovaSeq6000 シーケンサーの配列データの解析に関わるパフォーマンスだけでなく、Oxford Nanopore Technologies 社PromethION シーケンサーのロングリード配列データの解析、および Element Biosciences 社の AVITI シーケンサーの配列データ解析のパフォーマンス評価を実施した。

Nanopore PromethION はナノスケールの穴を DNA 分子が通過する際に発生する電流をセンサで計測することで塩基配列を得る技術である。NovaSeq6000 と比較すると非常に長い塩基配列を読むことができるロングリードシーケンサーの一つである。電流データである fast5 から、塩基配列データである fastq ファイルに変換するには guppy（v 5.0.17）ソフトウェアを使用した。これは３層の CNN、５層の RNN、Linear CRF Encoder からなる Deep Learning フレームワークで、2 GPU（NVIDIA Tesla V100 SXM2）を使用したところ 815 m CPU 時間（user+sys）を要した。その結果得られた塩基配列データは N50（長さの中央値に対応）が約 9400 塩基対（bps）であった（NovaSeq6000 の場合は 150 塩基対なので、中央値で 60 倍を超える長さの配列データを得た）。総塩基数は 20 Gbps で、通常 100 Gbps 前後である NovaSeq6000 よりも少なく、シークエンス深度は 7x 程度である（昨年度報告した NovaSeq 結果では 30x であった）。さらに、PromethION 技術では DNA メチル化（下記のワークフローでは、具体的には５メチル化シトシン）を検出できるためこれをさらに実施した。minimap2 を用いたアラインメントに 57808.554 sec CPU 時間（10 CPU 使用で real 5950.485 sec）、メチル化を検出するプロセス（nanopolish call-methylation）に 5970m CPU 時間（10 CPU 使用で real 2978 m）であった。これらをまとめると、PromethION による配列解析は、NovaSeq の結果と比較しても（下記参照）、高速化を要する現状であると考えられた。

AVITI は、Illumina 社特許が失効し始めたために世界中で新たに出現したシーケンス機器の一つであるが基本原理は異なり、Illumina 社シーケンス結果と比較すると精度の高さで知られている。実行時間について、同じ検体由来の DNA 解析結果による NovaSeq とも比較した結果を示す（表 12-1）。

fq2bam において AVITI シーケンサーの方が大きく実行時間が速くなっているが、アルゴリズムは同じものである。CloudBreak は NovaSeq より塩基コールの品質が高く、UltraQ はさらに高い。実行時間のトレンドが見えていることからも、精度が上がっているため、ベストマッチの配列位置を探索しやすく、実行時間の高速化となったものと考えられる。

表 12-1: シーケンサー・キットの違いによるゲノム解析の実行時間の比較。

シーケンサー・キット	塩基長	メモリ指定 *	GPU	CPU	fq2bam (m)	Haplotype Caller (m)
NovaSeq6000	150	960G	2	80	67.78	24.80
AVITI CloudBreak	150	480G	2	80	35.95	27.90
AVITI UltraQ	150	960G	2	80	27.60	20.62

* 最大メモリ使用量はいずれも 400G 以下

⑩で述べたように、新しいアルゴリズムも検討した。これは提案システムでの処理時間を 5 分以下にできるものであり、現在の GPU 上のソフトウェアに比べて非常に大きな高速化が期待できることがわかった。これは、オンチップ分散メモリアーキテクチャのため、各コアのローカルメモリの中ではランダムアクセスの性能も高いことを有効に利用したものである。

⑬地震と構造物シミュレーションアプリケーションの検討
（再委託先 国立研究開発法人 海洋研究開発機構）

地震と構造物シミュレーションアプリケーションの検討では、地震波伝播シミュレーションで使われている Element-by-Element(EbE)法による有限要素法カーネルについて実行効率の評価を進めた。その結果、EbE 法ではある程度メモリバンド幅が低い場合でも高い実行効率が得られるため、A64fx や A100 と同様の理論ピークに対して 10%程度ないしはそれより高い実行効率が期待できることが分かった。

具体的には、本解析での時刻歴解析では、時刻毎に $Ax = f$ なる方程式を反復型 solver（共役勾配法）にて求解するが、その共役勾配法内の主な計算となる行列ベクトル積 $q = Ap$ について EbE 法によりオンフライで計算しており、この演算の性能により本解析の性能が左右される。すなわち、オンフライで要素毎に要素剛性行列と関連する変数の積を計算し足しこんでいく（EbE カーネル部分）ため、計算量が多くなり、この効率的な処理を実現することがポイントとなる（通常の有限要素法では A をメモリ上に保持するため必要メモリが多くなるが、EbE 法ではその必要がなくなる。そのため大自由度の問題を扱う本解析では EbE 法を用いている）。実際、全体の計算量の内訳は、おおよそ EbE カーネル部分が６～７割、共役勾配法の諸計算が２割強、変数の時間方向の update が１割弱となっている。全体の計算量の６～７割を占める EbE カーネル部分に対して、メモリアクセスや演算の実装の工夫を行うことで、

A64fx では FP32 ピーク比で 9.6%、A100 では FP32 ピーク比で 12.7%という実効性能が得られており、演算量の多い EbE 法を用いているものの、time to solution の観点からも効率的な解析が実現されている。

上記を踏まえ、実際にベンチマークコードを机上検討した結果では、MN-Core では EbE 法が必要とする主記憶への間接アクセスを高速に実行できるため、主記憶アクセス時間が計算時間に遮蔽可能であることがわかった。このため、実行効率はベクトル性能に対して 80%程度（積和にならない部分があるため）、理論ピークの 20%程度が期待できることが分かった。

⑭気象・気候シミュレーションアプリケーションの検討
（再委託先 国立研究開発法人 国立環境研究所）

汎用 CPU での実行プロファイルとアクセラレータで実行する部分の性能評価に基づき、気象・気候シミュレーションについて定めたアプリケーションの性能評価を行った。具体的には、一昨年度選定した検討アプリケーションである全球非静力大気モデル NICAM（準構造格子ステンシル）の力学カーネル群を題材に、Julia での再実装を行い CPU および GPU での性能評価を実施した。また、プログラム記述言語の変更に伴う問題点の調査を行った。

<NICAM 力学カーネルの Julia 再実装>
一昨年度の検討において、気象・気候アプリケーションからは格子法を用いた大気モデルの流体力学過程を対象とし、アプリケーションは全球非静力大気モデル NICAM を選定した。性能評価にはソフトウェアパッケージ「IcoAtmosBenchmark v1」に含まれる NICAM のステンシル演算カーネルを利用した。このパッケージは「富岳」開発プロジェクトである FLAGSHIP2020 プロジェクトや、日独仏エクサスケールコンピューティングのためのソフトウェア開発研究事業 SPPEXA の一つである AIMES プロジェクトにおいて整備・利用されたものであり、昨年度においてもこれを利用しステンシル演算カーネルの Python での再実装と評価を行っている。今年度は、６つの演算カーネル（水平拡散、水平発散、鉛直三重対角行列演算、水平移流フラックス計算、水平移流制限項計算、鉛直移流制限項計算）のうち、水平拡散と鉛直三重対角行列演算の２つについて Julia を用いた再実装を行った。再実装する言語として Julia を選定した理由は、Python とその利用コミュニティが発展させている機械学習ライブラリ、可視化ライブラリとの親和性が高いこと、アーキテクチャ・低レベル言語環境の異なる多種の GPU への対応を進めており、将来のプログラム移植が容易であることが期待されること、既存の気象・気候アプリケーションが過度に依存している Fortran と言語仕様が似ている（配列添字の開始が 1 である、多次元配列のメモリ格納順が Column major order である）ことが、アプリケーションのプログラム再実装に際して有利なためである。MN-Core は Julia をサポートしていないが、集中的な開発資源の投入を行えばサポート実現のための障壁はあまり高くないと期待する。

Python 実装の際と同様の方針で、まずはオリジナルの Fortran コードをなるべく忠実に再現した形で、Julia への移植を実施した。リスト 14-1 にオリジナルコードの一部を示し、リスト 14-2 に同じ区間を Julia で記述した際のコードを示す。両者は非常に似た記述を行うことが可能であり、ループを do 構文と for 構文で記述するかや、継続行の扱い、配列の添え字を書く際の括弧の種類、程度しか違いが存在しない。昨年度調査を行った Python への移植では、Google JAX を用いた高速化を達成するた

めに NumPy で用いられる「スライシング」記法に書き換える必要があった他、配列添え字やループの開始インデックスを注意深くずらす必要があった（リスト 14-3）。Julia の場合は後述する最適化を適用した後も、for 文を用いたループでの記述を変更する必要がなく、「スライシング」に相当する Julia の「ブロードキャスト」演算を有効にするための記述変更は必ずしも高速化するわけではなかった。これら As Is コードの経過時間について表 14-1 に示す。計測にあたっては、Intel Core i9-12900K を搭載したワークステーションを用い、Ubuntu 22.04 上に実行環境を構築して計算を行った。言語環境はそれぞれ GNU Fortran 11.4.0、Python3.12.8、Julia 1.10.8 を用い、スレッド並列化は適用していない。As Is コードでは Python と Julia の計算時間に大きな違いはなく、いずれも Fortran と比較して 100〜1000 倍低速であった。

リスト 14-1: 演算カーネル（水平拡散）のソースコードの一部(昨年度報告書より再掲)。

```
Fortran Ver. (抜粋)

do l = 1, lall
   do k = 1, kall
      do d = 1, nxyz
         do g = gminm1, gmax
            ij     = g
            ip1j   = g + 1
            ip1jp1 = g + iall + 1
            ijp1   = g + iall

            vt(g,d,TI) = ( ( + 2.0 * coef_intp(g,1,d,TI,l) &
                             - 1.0 * coef_intp(g,2,d,TI,l) &
                             - 1.0 * coef_intp(g,3,d,TI,l) ) * scl(ij    ,k,l) &
                         + ( - 1.0 * coef_intp(g,1,d,TI,l) &
                             + 2.0 * coef_intp(g,2,d,TI,l) &
                             - 1.0 * coef_intp(g,3,d,TI,l) ) * scl(ip1j  ,k,l) &
                         + ( - 1.0 * coef_intp(g,1,d,TI,l) &
                             - 1.0 * coef_intp(g,2,d,TI,l) &
                             + 2.0 * coef_intp(g,3,d,TI,l) ) * scl(ip1jp1,k,l) &
                         ) / 3.0
         enddo
      enddo
   enddo
enddo
```

リスト 14-2: リスト 14-1 に同じ。ただし Julia を用いて記述したもの。

```
Julia Ver. (抜粋)

@inbounds begin
for l = 1:ADM_lall
   for k = 1:ADM_kall
      for d = 1:ADM_nxyz
         for g = gminm1:gmax
            ij     = g
            ip1j   = g + 1
            ip1jp1 = g + iall + 1
            ijp1   = g + iall

            vt[g,d,TI] = ( ( + 2.0 * coef_intp[g,1,d,TI,l]
                             - 1.0 * coef_intp[g,2,d,TI,l]
                             - 1.0 * coef_intp[g,3,d,TI,l] ) * scl[ij    ,k,l]
                         + ( - 1.0 * coef_intp[g,1,d,TI,l]
                             + 2.0 * coef_intp[g,2,d,TI,l]
                             - 1.0 * coef_intp[g,3,d,TI,l] ) * scl[ip1j  ,k,l]
                         + ( - 1.0 * coef_intp[g,1,d,TI,l]
                             - 1.0 * coef_intp[g,2,d,TI,l]
                             + 2.0 * coef_intp[g,3,d,TI,l] ) * scl[ip1jp1,k,l]
                         ) / 3.0
         end
      end
   end
end
end # @inbounds
```

リスト 14-3: リスト 14-1 に同じ。ただし Python を用いて、Fortran を模倣して記述されたものから、JAX の JIT コンパイラを有効にするために配列スライシングに近い形で演算を記述したもの(昨年度報告書より一部編集して再掲)。

```
Python(JAX) Ver. (抜粋)

vt = vt.at[gminm1-1:gmax,0:kall,0:nxyz,0].set(
      ( (   2.0 * coef_intp[gminm1-1:gmax,None,0,0:nxyz,TI,0]
          - 1.0 * coef_intp[gminm1-1:gmax,None,1,0:nxyz,TI,0]
          - 1.0 * coef_intp[gminm1-1:gmax,None,2,0:nxyz,TI,0] ) * scl[gminm1-1:gmax          ,0:kall,None,0]
      + ( - 1.0 * coef_intp[gminm1-1:gmax,None,0,0:nxyz,TI,0]
          + 2.0 * coef_intp[gminm1-1:gmax,None,1,0:nxyz,TI,0]
          - 1.0 * coef_intp[gminm1-1:gmax,None,2,0:nxyz,TI,0] ) * scl[gminm1:gmax+1          ,0:kall,None,0]
      + ( - 1.0 * coef_intp[gminm1-1:gmax,None,0,0:nxyz,TI,0]
          - 1.0 * coef_intp[gminm1-1:gmax,None,1,0:nxyz,TI,0]
          + 2.0 * coef_intp[gminm1-1:gmax,None,2,0:nxyz,TI,0] ) * scl[gminm1+iall:gmax+iall+1,0:kall,None,0]
      ) / 3.0
)
```

表 14-1：カーネルの実行時間比較

カーネル名	説明	経過時間[sec]		
		Fortran	Python	Julia
dyn_diffusion	３次元データに対して水平２次元方向に格子点を参照するステンシル計算	0.007	12.2	12.2
dyn_vi_rhow_solver	３次元データに対して鉛直１次元方向に３重対角行列を解くステンシル計算	0.006	1.89	0.93

次に、カーネルのうち dyn_diffusion に対して、実行時間を短縮するための高速化手法を適用した。検討の結果、最も劇的な改善がみられたのは配列サイズを決めている変数について const を用いて宣言することであった。配列サイズを格納する変数は全プログラムファイルで共有することが多く、そのため global 変数の扱いとなる。global 変数の値と型はいつでも変更される可能性があるため、コンパイラの最適化を著しく阻害する。それらの変数の const 宣言により、メモリアロケーションの回数とメモリ使用量がそれぞれ 100,000,000 分の 1、10,000,000 分の 1 に減少し、As Is コードの 800 倍程度まで高速化した。実際にフルアプリケーションにおいて、該当する配列サイズの const 宣言を行うことが出来るかについては、実行時の前処理を工夫することによって可能である。すなわち、Fortran の場合コンパイル済みの実行バイナリを事前に準備し、実行時にパラメータファイルを読み込ませて配列サイズを確定させているが、Julia の場合は事前コンパイルが必要ではないため、実験設定に合わせて必要な配列サイズを記述したプログラムファイルを生成し実行することで、実験毎に配列サイズを変更しつつ、const 宣言を適用することによる高速化を達成可能である。さらに、配列添字の範囲検査を省略する@inbounds マクロを適用し、オーバーヘッドを削減すると共に、固定長配列のためのパッケージ StaticArrays.jl を利用し、一部のサイズの小さい mutable な一次元ワーク配列に対して MVector 型での宣言を適用することで静的かつ高速な固定長配列に変更した。加えて、ブロードキャスト演算についても評価を実施した。ほとんどの区間では@.マクロを最内ループ内の各演算行の先頭に適用し、元のアルゴリズム記述を変えずにブロードキャスト演算への変換を行うと共に、部分配列のアクセスがある場合は@view マクロを併用して配列のアクセス範囲を適用した。

これらの高速化の適用結果について表 14-2 に示す。配列サイズの const 宣言で Fortran の実行速度に肉薄するところまで改善し、さらに@inbounds マクロの適用に

よって Fortran と同等の速度となった。StaticArrays の適用によって、Fortran よりも高速化するケースがあることも示された。一方で、ブロードキャスト演算については for ループのまま計算する場合よりもメモリアロケーションやメモリ使用量が増加し、結果としてプログラム修正がかなり必要であるにも関わらず、実行時間がある程度以上高速化しなかった。これは部分配列アクセスなどが存在する場合に、都度ビューを設定する必要があり、メモリアロケーション回数を削減しづらいことに起因すると考えられる。

ここまで CPU 上での Julia への移植は非常に開発コストが低く高性能であることが示されたが、GPU での最適化を進める場合、CPU での最適化とは全く異なる戦略を取る必要が明らかになった。Julia の GPU 最適化では、配列プログラミングとカーネルプログラミングのいずれかを用いる。前者の場合は、用いる配列をすべて GPU デバイス用の配列（仮に GPU 用配列と呼ぶ）に置き換え、実行する。GPU 用配列はループを用いた演算には向いておらず、基本的にブロードキャスト演算が要求される。そのため、Python(JAX)を用いた場合と同じく、かなり多くのカーネルを書き換える必要が発生し、ループで記述していた Fortran コードをそのまま移植し性能を発揮出来るという Julia の利点は発揮できなくなる。後者のカーネルプログラミングを採用した場合は、カーネルが GPU 用コードに事前コンパイルされるため、ループでの記述が許容される。しかし、そのままの関数をまるまるカーネル化しようとしても計算結果が一致しないケースが多く、カーネル内での中間配列準備や、並列化したくない演算、非同期で実行できない演算等を制御するために、一つの大きなカーネルをいくつもの小さなカーネルに細分化しなければいけない。これまでの OpenACC の言語仕様の進展で見られたように、Julia での GPU カーネルプログラミングにおいても、これらの挙動のコントロールについて仕様や機能が今後追加されることを期待する。dyn_diffusion を用いて配列プログラミングとカーネルプログラミングの書き方で GPU 最適化を適用した結果、経過時間は Apple M1 Ultra の GPU(48 コア)を用いてそれぞれ 28 秒、9 秒であった。これは CPU を用いた演算と比べて圧倒的に低速であり、さらなる検討が必要である。

表 14-2：dyn_diffusion カーネルの高速化適用結果

	経過時間[sec]	メモリアロケーション	
		回数	サイズ
Julia As Is コード	12.2	405M	8.3GiB
ケース 1			
配列サイズの const 宣言	0.015	4	793KiB
＋@inbounds マクロの適用	0.0072	4	793KiB
＋StaticArrays の適用	0.0039	3	792KiB
ケース 2			
配列サイズの const 宣言+ブロードキャスト演算	0.21	11K	772MiB
＋@inbounds マクロの適用	0.22	11K	772MiB

以上をふまえ、Fortran から Julia へのプログラム記述言語を変更した場合の利点・問題点について以下にまとめる。

- Julia を用いた場合、少なくとも CPU 上での実行では言語の学習コストや移行先の言語への特別な配慮（インデックス番号の扱いやメモリ配置順の違い）をほとんど意識することなく移植が可能であり、またループを基本とした記述スタイルを変えずに Fortran と遜色ない速度で計算を行うことが可能である。
- Julia での計算高速化において型や値、配列サイズを事前に確定しておくことは非常に重要である。実行時に配列サイズが変わらない、型が変わらない、immutable である、配列外参照の検査が必要ない、等の追加情報を適切に付与することで速度とメモリ使用量の最適化が促進される。
- GPU の利用においては、CPU 最適化時に有効だったループでの演算をやめ、ブロードキャスト演算に切り替える必要があるが、なるべくコードの書き換えコストをかけずに高速化することが現時点では難しく、例外処理等について細かい配慮を必要とする。

今年度は、さらに、NICAM ベースのベンチマークコードである IcoAtmosBenchmark について、現行の PFN 開発環境での MN-Core への実装と性能評価をおこなった。ベクトル演算での理論ピーク性能 8TF に対して、３つのカーネルで 4.178TF, 1.9TF, 3.922 TF と 25-50% の効率が実現できた。これはもちろんデータがオンチップ SRAM にはいる場合であるが、ポスト富岳提案では仮に DRAM にデータがあってもそれを SRAM にロードする時間はほぼ無視できるようになるため、同等の効率が期待できる。興味深いことは、全てのベンチマークカーネルで A64fx よりも高い実行効率が実現できていることであり、大きなオンチップメモリをソフトウェアが有効利用できる提案アーキテクチャの特性があらわれている。

⑮ものづくりアプリケーションの検討
（再委託先 学校法人 東洋大学）

汎用 CPU での実行プロファイルとアクセラレータで実行する部分の性能評価に基づき、ものづくり分野のアプリケーション性能評価を行った。具体的には、次世代計算基盤に対しものづくりアプリケーションが求める演算数やメモリ容量を見積もるため、2023 年度に整備したオープンソース CAE ソフトウェア ADVENTURE のベンチマークコードを用いた調査を実施した。

非構造格子有限要素解析ソフトウェア ADVENTURE は、領域分割法を用いた並列計算を特徴とし、C 言語、MPI、OpenMP で開発されている。計算カーネルとなる部分領域ソルバーの実装においては、メニーコアおよび GPU・アクセラレータへの有効な実装となりうる Local Schur Compliment(LSC)アプローチに着目し、スカラー型計算機向けのミニアプリを開発してきた。このアプローチにより、部分領域の剛性行列を LSC 行列として保持することで、従来の反復ごとの行列分解を、LSC 行列を用いた対称密行列ベクトル積演算に置き換え、大幅な計算時間の短縮を実現してきた。

この方法では、計算時間のほぼ全部となる DDM 反復の間の演算は固定係数の行列ベクトル積で。行列側の要求 B/F は 2 となる。これを提案システムで実行すると、DRAM の B/F 値は 0.2 であるため 10% の実行効率が期待できる。行列ベクトル積なので行列演算ユニットが有効に使えるわけではなく、実効的なピーク性能が 1/4 であることを考慮すると 40% の実行効率となる。

ADVENTURE の利用促進と MN-Core 等の次世代計算基盤への適応を目指し、ファイル入出力や有限要素解析部分をライブラリ化した Python アプリを開発した。これにより、非構造格子由来の疎行列を係数とする連立一次方程式を用いた MN-Core の性能評価や最適化が可能となり、既存コードの再利用による効率的な開発が可能となった。また、領域分割法の並列求解部分の性能評価を行う C+OpenMP アプリを開発し、ADVENTURE の MN-Core への直接的な移植時の性能を評価可能とした。

次世代計算基盤に対する要求調査では、ADVENTURE のベンチマークコードを用いた評価から、大規模問題においてノードあたり初期計算時に 10^{12} オーダー、反復計算時に 10^{9} オーダーの演算数、行列計算用に数百メガバイトのメモリ容量が必要となることが明らかになった。

これまで主に、応力解析ソフトウェア ADVENTURE_Solid を中心に性能評価を行ってきたが、新たなターゲットアプリケーションとして、高周波電磁界解析ソフトウェア ADVENTURE_FullWave に着目し、その GPU への移植と性能評価を実施した。情報通信機器や医療機器の電磁環境適合性対応に不可欠な高周波電磁界解析は、数十から数百億要素という大規模モデルを扱う必要があり、その計算負荷は極めて高い。ADVENTURE_FullWave の GPU 移植は、このような大規模計算を効率的に処理するために、アクセラレータが不可欠であることを示唆する。

⑯マテリアルサイエンス応用アプリケーションの検討
（再委託先 株式会社 Quemix）

RSDFT は、密度汎関数法（DFT）に基づく第一原理計算により様々なマテリアルサイエンスのシミュレーションを行うためのコードであり、実空間差分法により DFT の基礎方程式である Kohn-Sham 方程式を解く実装となっている。数値計算的には、空間をメッシュに切って得られる行列に対する固有値問題を、固有値の小さいものから順に、系に含まれる電子の個数に相当する本数の固有ベクトル求めるというものになる。RSDFT では、その求解に部分空間反復法（Rayleigh-Ritz 反復法）を採用している。主要な計算は、部分空間対角化（SD）、Gram-Schmidt 直交化（GS）、および共役勾配法（CG）からなり、SD および GS の演算量はシステムサイズ（＝原子数）の３乗に比例し、CG はシステムサイズの２乗に比例するため、計算全体の演算量もシステムサイズの３乗に比例することになる。

表 16-1 に、以前に検討を行った A100×8，１ノードの環境で、1000 原子規模の RSDFT 計算を実際に実行した際の実行時間とその内訳を示す。

表 16-1：A100×8，１ノードの環境で 1000 原子規模の系に対する実行時間（実測）。（単位：秒）

	SCF (1 step)	SD	GS	CG	Others
DGX A100	18.6	4.25	3.07	9.56	1.72

表 16-1 のデータを元にして、ポスト「富岳」提案環境での性能評価を行う。評価を行う問題サイズ（グリッド点数，バンド本数（∝電子数））および、問題を乗せるのに必要なノード数を表 16-1 の系も併せて表 16-2 に示す。

表 16-2: 性能評価を行う問題サイズ

原子数	バンド本数	グリッド点数	ノード数	メモリ/ノード (GB)	差分袖データ (一方向) (MB)
1000	2592	4374000	1	170	-
8000	20736	35251200	8	340	0.547
16000	41472	70502400	128	340	0.362

評価の方法であるが、演算時間については、表 16-1 の結果を元に、ノード当たりのグリッド点数とバンド本数から、システムサイズの３乗のスケーリングを用いてノード当たりの演算量を算出し、A100 とポスト「富岳」の性能比を鑑みて評価する。複数ノードを使用する計算では、これらにさらにノード間通信に要する時間を上乗せする。各ルーチンでの主要な通信は、SD では、ベクトルの内積から行列要素を計算するために必要な Allreduce、および差分の行列を作用させるために必要な差分の袖の隣接通信である。GS では、SD と同様な内積計算が必要で、そのために Allreduce が必要となる。SD および GS の Allreduce はバンド本数の２乗個の要素をまとめて一回で行うとする。CG はベクトルの本数×８回の内積計算の Allreduce、および差分の隣接通信となる。CG の内積の Allreduce はベクトル複数本分をまとめて処理することができるため、ここでは表 16-1 の計測を行ったときと同じ 128 本分をまとめて処理することとした。また Allreduce についてはバタフライ　アルゴリズムを想定し、さらに Allreduce および差分の隣接通信共にレイテンシは無視することとした。

表 16-3 に評価結果をまとめる。なお、表 16-1 にあった Others の項目は、最終的な物理量計算のための時間であり、他の主要ルーチンに比して寄与が小さいため、今回の見積もりからは除外した。

表 16-3: ポスト「富岳」提案環境での計算時間評価（単位：秒）。カッコ内は通信時間。

原子数	SCF (1 step)	SD	GS	CG
1000	5.27	1.33	0.96	2.98
8000	170 (0.84)	85.1 (0.28)	61.4 (0.02)	23.7 (0.53)
16000	79.2 (1.11)	42.6 (0.38)	30.7 (0.03)	5.92 (0.70)

現在、第一原理計算を実行すると、数百原子以下の系においても、Self Consistent Field (SCF) 反復計算の１ステップに数十秒から数百秒を要することはざらにある。それを考えると、今回の見積もりは、かなり少ないノード数で 10000 原子規模の系を、現在と同程度の計算時間で扱えるようになること示しており、リアルな材料の状況に忠実な原子モデルを用いたシミュレーションが実行可能になるという有望な結果といえる。

⑰素粒子・原子核物理応用アプリケーションの検討
(再委託先 国立大学法人 広島大学)

汎用 CPU での実行プロファイルとアクセラレータで実行する部分の性能評価に基づき、素粒子・原子核物理について定めたアプリケーションの性能評価を行った。具体的には、素粒子・原子核物理の理論的計算のために必要な格子量子色力学計算アプリケー

ションの中で最も時間のかかるクォーク伝搬関数を反復法により計算するソルバー部分の評価を行った。クォーク伝搬関数を反復法により計算する部分のみを抜き出してベンチマーク化したプログラム CCS QCD Solver Benchmark を元に評価した。

前年度のアクセラレータ単体で評価した机上性能を基に、弱スケーリングおよび現在の「富岳」と同程度のネットワーク性能を仮定し複数ノードでの並列実行での性能を見積もった。格子量子色力学計算アプリケーションで将来常用的に実行される格子サイズを 256^4 と設定し、1アクセラレータあたりの格子サイズを $16^3 \times 32$ と設定しその性能を評価した。

昨年度の1アクセラレータを用いた倍精度での BiCGStab 法 2000 反復での実行時間はローカルメモリにデータが乗る場合は、約1.6秒であった。全体格子サイズ 256^4 をノードあたり格子サイズ $16^3 \times 32$ に分割すると、並列度 $16^3 \times 8$ の 32768 ノードでの並列計算となる。4次元で並列化するため、ステンシルカーネルの差分計算で8方向の隣接通信が発生する。BiCGStab 法の1反復で2×2回のステンシルカーネルを用いた行列ベクトル積がある。4次元の8方向のそれぞれに、データサイズ 384 KiB の袖交換が発生する。「富岳」の ToFuD のネットワークの性能（レイテンシ：約 0.5 μsec、リンクあたりスループット：6.35 GB/s @ 1MB）と同様のネットワークを仮定する。データサイズは「富岳」の最大スループットのときのデータサイズの約 1/3 なので通信時間はこの3倍かかると仮定し、約 162 μsec と見積もった。昨年度見積もったアクセラレータボード内での chip 間での全方向の袖交換の時間の見積もりは 約 150 μsec であったので、ノード間の1方向の袖交換通信時間はこれとほぼ同じと見積もられた。8方向の通信を同時に行うことができる場合ステンシルカーネルを1回かける計算の袖交換通信時間は約 162 μsec である。8方向同時に通信できない場合はこの時間の整数倍の時間がかかると見込まれる。8方向同時に通信ができる場合の袖交換通信時間は BiCGStab 法の1反復で 約 162×2×2＝648 μsec であり、2000 反復で約 1.29 秒と見積もった。またベクトルの内積やノルムの計算のための縮約のための時間を見積もった。「富岳」の 27648 ノードでの縮約の性能はデータサイズが 12 Byte までは約 10 μsec, 12 Byte 以上では 約 60 μsec から増えていく。将来の計算機での 32768 ノードでの縮約性能がこの「富岳」と同等性能であると仮定する。BiCGStab 法の1反復での実数縮約は9回行われるので、1反復中での縮約時間は約 900 μsec であり、2000 反復で約 1.8 秒と見積もった。前年度に見積もったノードでの計算時間とこれらの通信時間を合わせると、格子サイズ 256^4 の問題を並列度 $16^3 \times 8$ で分割したときの計算時間は（1.6 + 1.29 + 1.8 =）約 4.7 秒と見積もった。これは袖交換のノード内通信とノード間通信を同時に行わない場合の見積もりである。前年度見積もりのアクセラレータ実行見積もり時間の約 1.6 秒にはノード内袖交換通信の時間約 1.2 秒が含まれている。ノード間通信とノード内通信を同時に行うことができる場合は、計算時間は約 2.5 秒と見積もった。

並列度が大きい場合に縮約のための時間がボトルネックになるため、ノードあたりの性能を上げて並列度を低くすること、および、反復法計算アルゴリズムに、アクセラレータに特化した前処理行列を導入し、並列度を低くしても計算時間が伸びないようにする工夫を取る必要があると考えられる。GPU 向けへの実装として、Overlapped Restricted Additive Schwarz (RAS)法という固定反復法が前処理行列として用いられている。RAS 法はノードあたりの格子サイズを増やしベクトル長を長く取りながら、反復回数を抑えることができるため GPU に適した方法であるが、国内に 100 ノード以上の並列実行を利用できる環境がないため RAS 法の高並列度での効率については

不明である。また、MN-Core 様のローカルメモリを利用するアーキテクチャへの RAS 法の適用についての検討、および、RAS 法以外の MN-Core 様アーキテクチャへの前処理アルゴリズムの調査開発が必要である。

⑱宇宙・惑星科学応用アプリケーションの検討
(再委託先 国立大学法人 東京大学)

宇宙・惑星科学の粒子系シミュレーションにおいて、計算コストが最も高いのは重力や流体の相互作用計算である。相互作用計算のための計算カーネル生成プログラム「PIKG」を用いて生成した計算カーネルを使用し、汎用 CPU (Intel x86-64, Fujitsu A64FX) および GPU (NVIDIA GH200) で重力相互作用計算および流体相互作用計算の性能評価を行った。シングル CPU コアおよびシングル GPU での性能(Flops、単精度ピーク性能)は、表 18-1 の通りである。

⑲で述べたように、重力カーネルや流体カーネルに対して、高レベルの記述から MN-Core を含む多様なアーキテクチャ向けの最適化したコードを生成する PIKG システムの改良を進めた。本項での CPU、GPU での高い実行効率は PIKG を使うことで得られたものである。さらに、MN-Core の重力カーネルでも、ベクトル演算のピーク性能に対して 65.2%と、H100 や A64fx より高く、ABC512 使用時の EPYC9474F と同等で現存のアーキテクチャではトップクラスの効率を実現できた。

表 18-1: 計算性能比較。演算数は重力 27、流体 (力) 101、流体 (密度・圧力) 73。

カーネル	Fujitsu A64FX SVE		AMD EPYC™ 9474F AVX2		AMD EPYC™ 9474F AVX512		NVIDIA Grace CPU and Hopper H100 GPU	
重力	37.3 Gflops	29.4%	65.8 Gflops	50.2%	90.6 Gflops	69.1%	22.5 Tflops	38.0%
流体(力)	19.8 Gflops	15.4%	29.4 Gflops	22.4%	81.5 Gflops	62.1%	1.88 Tflops	2.8%
流体(密度・圧力)	21.9 Gflops	17.1%	15.1 Gflops	11.5%	87.6 Gflops	66.8%	0.555 Tflops	0.64%

⑲アクセラレータむけ最適化コンパイラの検討
(再委託先 国立研究開発法人 産業技術総合研究所)

粒子系を主な対象として、MN-Core およびその類似アーキテクチャを有効に使う方法の検討を行った。前年度に引き続き、粒子多体系相互作用計算のユーザー定義カーネルを MN-Core およびその類似アーキテクチャ上で実装する最適化コンパイラの開発を進めている。この最適化コンパイラは、リスト 19-1 に示すような簡便に記述可能なユーザー定義カーネルを、MN-Core およびその類似アーキテクチャ上で実行できるバイトコードに変換すると共に、そのバイトコードを利用するホスト側コードを生成する。

リスト 19-1: ユーザー定義カーネルの例（重力相互作用）

```
F32 eps2
rij = EPI.pos - EPJ.pos
r2 = rij * rij + eps2
rinv = rsqrt(r2)
mrinv = EPJ.mass * rinv
mrinv3 = mrinv * rinv * rinv
FORCE.acc += rij * mrinv3
```

バイトコードの変換の際には、命令列の順序をアニーリングにより最適化する技術を導入し、データの無駄なレジスタ間移動や命令間依存関係による待機命令の挿入を抑制する手法を開発しており、今年度は、MN-Core だけではなく、その他の類似アーキテクチャでも対応できるように、汎用性を向上させる改修を行った。いくつかの検証の結果、相互作用計算部分の実行効率は専門家による手動の最適化と同等の水準を達成していることが明らかになった。また、開発した命令列最適化手法を具体的な粒子多体系シミュレーションに適用したときの性能を比較するために、GPU での実行と、実機での実行の比較も行った。本最適化コンパイラに関する詳細な実行結果について、論文投稿中である。

⑳プロジェクトの総合的推進

プロジェクト全体の連携を密にし、円滑に運営していくため、運営委員会や技術検討会の開催等、参画各機関間の調整を行った。

２－３．活動（研究会の活動等）

①ミーティングの実施

研究開発課題責任者、コーディネーター、各分担先代表者を構成メンバーとするミーティングを設置し、開催した。

・ 全体ミーティング（隔週金曜日）
・ アーキテクチャグループミーティング（隔週木曜日）
・ システムソフトウェアグループミーティング（隔週木曜日）
・ Post「富岳」ソフトウェアミーティング（毎週木曜）

②次世代計算基盤に係る調査研究に関する合同ワークショップ～フィージビリティスタディ結果報告～

12 月 27 日　東京大学　本郷キャンパス　（ハイブリッド）

③シンポジウム「ポスト富岳で拓くアプリケーションの未来」

3 月 25 日　コングレスクエア日本橋　（ハイブリッド）

２－４．実施体制

業　務　項　目	実　施　場　所	担　当　責　任　者
①アーキテクチャ調査研究	兵庫県神戸市灘区六甲台町１番１号 国立大学法人神戸大学	大学院理学研究科 特命教授　牧野 淳一郎
②システムソフトウェア・ライブラリ調査研究	兵庫県神戸市灘区六甲台町１番１号 国立大学法人神戸大学	大学院理学研究科 特命教授　牧野 淳一郎
③アプリケーション調査研究	兵庫県神戸市灘区六甲台町１番１号 国立大学法人神戸大学	大学院理学研究科 特命教授　牧野 淳一郎
④アーキテクチャ調査研究の統括およびアクセラレータ評価	東京都千代田区大手町一丁目６番１号　大手町ビル２階 株式会社 Preferred Networks	代表取締役社長 西川 徹
⑤CPU 評価	東京都文京区本郷七丁目３番１号 国立大学法人東京大学	大学院情報理工学系研究科 准教授　塩谷 亮太
⑥アクセラレータアーキテクチャ評価	愛知県名古屋市昭和区御器所町字木市２９番 国立大学法人名古屋工業大学	工学専攻情報工学系プログラムネットワーク分野 助教　小泉 透
⑦ネットワークアーキテクチャ評価	東京都千代田区一ツ橋二丁目１番２号 大学共同利用機関法人情報・システム研究機構 国立情報学研究所	アーキテクチャ科学研究系 教授　鯉渕 道紘
⑧システムソフトウェア・ライブラリ調査研究の統括およびコンパイラ検討	福島県会津若松市一箕町大字鶴賀字上居合９０番地 公立大学法人会津大学	コンピュータ理工学部 教授　中里 直人
⑨DSL 検討	島根県松江市西生馬町１４－４ 独立行政法人国立高等専門学校機構 松江工業高等専門学校	情報工学科 准教授　岩澤 全規
⑩アプリケーション調査研究の統括	東京都文京区本郷二丁目１番１号 学校法人順天堂	健康データサイエンス学部開設準備室　特任教授 姫野 龍太郎
⑪創薬と深層学習応用アプリケーションの検討	東京都千代田区大手町一丁目６番１号　大手町ビル２階 株式会社 Preferred Networks	代表取締役社長 西川 徹
⑫ゲノム科学アプリケーションの検討	東京都文京区本郷七丁目３番１号 国立大学法人東京大学	大学院新領域創成科学研究科 教授　鎌谷 洋一郎
⑬地震と構造物シミュレーションアプリケーションの検討	神奈川県横須賀市夏島町２番地１５ 国立研究開発法人海洋研究開発機構	付加価値情報創生部門 部門長　堀 宗朗

⑭気象・気候シミュレーションアプリケーションの検討	茨城県つくば市小野川１６番地２ 国立研究開発法人国立環境研究所	地球システム領域 主任研究員　八代 尚
⑮ものづくりアプリケーションの検討	東京都文京区白山五丁目２８番２０号 学校法人東洋大学	総合情報学部 教授　塩谷 隆二
⑯マテリアルサイエンス応用アプリケーションの検討	東京都中央区日本橋２丁目１１番２号 株式会社 Quemix	プロダクトマネージャー／研究員　岩田 潤一
⑰素粒子・原子核物理応用アプリケーションの検討	広島県東広島市鏡山一丁目３番２号 国立大学法人広島大学	大学院先進理工系科学研究科 准教授　石川 健一
⑱宇宙・惑星科学応用アプリケーションの検討	東京都文京区本郷七丁目３番１号 国立大学法人東京大学	大学院理学系研究科 准教授　藤井 通子
⑲アクセラレータむけ最適化コンパイラの検討	東京都千代田区霞が関一丁目３番１号 国立研究開発法人産業技術総合研究所	機能材料コンピュテーショナルデザイン研究センター 研究員　遠藤 克浩
⑳プロジェクトの総合的推進	兵庫県神戸市灘区六甲台町１番１号 国立大学法人神戸大学	大学院理学研究科 特命教授　牧野 淳一郎

様式第２１

学 会 等 発 表 実 績

委託業務題目「次世代計算基盤に係る調査研究」（システム研究調査チーム）
機関名　国立大学法人神戸大学
１．学会等における口頭・ポスター発表

発表した成果（発表題目、口頭・ポスター発表の別）	発表者氏名	発表した場所（学会等名）	発表した時期	国内・外の別
Feasibility study for the next flagship supercomputer development and high-resolution climate modeling efforts in Japan（口頭）	八代 尚	8th ENES HPC Workshop on “HPC for high-resolution climate and weather modelling”	2024.5	国外
Efforts toward optimization of global non-hydrostatic atmospheric model on GPU supercomputer（口頭）	八代 尚 山崎 一哉 荒川 隆 松岸 修平 Shereo Intyisyar 中島 研吾	JpGU Meeting 2024	2024.5	国内
Use of Local LLMs for Research Code Development（口頭）	中里 直人	2024 InPEx(The International Post-Exascale Project) Workshop	2024.6	国外
Development of a weather/climate model for Japan’s new flagship machine following Fugaku（口頭）	八代 尚	Workshop on Global Storm-Resolving Analysis Bridging Atmospheric and Cloud Dynamics	2024.6	国内
Optimization, Translation, and Transformation of an Atmospheric Model to Utilize GPUs: A Case of NICAM（口頭）	八代 尚	The 21st Annual Meeting of Asia Oceania Geosciences Society (AOGS2024)	2024.6	国外
Held and Suarez (1994)再訪(口頭)	大淵 済	神戸大学理学部大気セミナー	2024.6	国内
光×コンピューティングの研究動向(口頭)	鯉渕 道紘	第84回OPT（Optical Packaging Technology)公開研究会	2024.7	国内
ポスト富岳世代のMN-Coreベースアクセラレータ対応OpenACCのインターフェイスとコンパイラの検討及び開発(口頭)	綱島 隆太	IPSJ第195回HPC研究会	2024.8	国内

ポスト・エクサ、ポストムーア時代の HPC と AI(口頭)	牧野 淳一郎	【オンラインセミナー】第6回 アドバンス・シミュレーション・セミナー	2024.9	国内
アクセラレータ用次世代プロセッサ MN-Core の動向の紹介（ポスト富岳調査の話を交えて）(口頭)	綱島 隆太	EPnetFaN	2024.9	国内
Climate modeling at Exascale: Status, Challenges and Collaboration Opportunities (口頭)	八代 尚	International Computing in the Atmospheric Sciences Symposium (iCAS2024)	2024.9	国外
大気大循環の運動エネルギー 散逸の理解に向けて…の下準備を兼ねて モデル解像度を変えたとき水平拡散係数をどう調整するか？(口頭)	大淵 済	惑星大気計算に関するワークショップ	2024.11	国内
研究報告Ⅱ システム調査研究の概要と検討結果（口頭)	牧野 淳一郎	次世代計算基盤に係る調査研究に関する合同ワークショップ～フィージビリティスタディ結果報告～	2024.12	国内
Development of Global Atmospheric Model NICAM towards Fugaku-NEXT era (口頭)	八代 尚	HANAMI High-Level Symposium	2025.1	国外
Development of OpenACC for MN-Core: Part of the Post-Fugaku FS by Kobe University(ポスター)	綱島 隆太	第7回 R-CCS 国際シンポジウム	2025.1	国内
HPC とスペクトル・モデル(口頭)	大淵 済	福岡大学理学部セミナー	2025.2	国内
大気大循環モデルの解像度を変えたら水平粘性係数をどう調節したらいいの？Takahashi+ 2006 から約20年… 懸案の問題 次元解析と Held and Suarez 1994 実験による考察の初期報告(口頭)	大淵 済 岩山 隆寛	GFD セミナー2025 春	2025.3	国内
様々な浮動小数点演算形式の評価プラットフォームとしての FPGA と GPU の比較(口頭)	中里 直人	第193回 HPC 研究会	2025.3	国内
MN-Core アーキテクチャの	中里 直人	シンポジウム「ポス	2025.3	国内

ためのプログラミングモデル（口頭）		ト富岳で拓くアプリの未来」		
星一つ一つまで再現した銀河形成シミュレーションへ向けて（口頭）	藤井 道子	シンポジウム「ポスト富岳で拓くアプリの未来」	2025.3	国内

２．学会誌・雑誌等における論文掲載

掲載した論文（発表題目）	発表者氏名	発表した場所（学会誌・雑誌等名）	発表した時期	国内・外の別
ポスト富岳世代の MN-Core ベースアクセラレータ対応 OpenACC のインターフェイスとコンパイラの検討及び開発	綱島 隆太 遠藤 克浩 中里 直人 井町 宏人 牧野 淳一郎	情報処理学会研究報告, Vol.2024-HPC-195 No.1	2024.8	国内