令和５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６年５月３１日

国立大学法人神戸大学

牧野淳一郎

目次

本報告書は、文部科学省の令和5年度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5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。

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、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、国立大学法人東海国立大学機構名古屋大学、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、国立大学法人名古屋工業大学と連携し、再委託により研究開発を推進する。

２．2023 年度（報告年度）の実施内容

２－１．当該年度（2023 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、電力性能を予測すると共に、独自アクセラレータアーキテクチャについて、前年度定めたレファレンス設計についてシミュレーションを行い、N5ないしN3以降のプロセスルールでの動作速度・電力性能、アプリケーションカーネルでの性能を評価する。

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）

前年度に定めた対象アプリケーションに対して、信頼できるレベルの性能推定を行う。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、国立大学法人名古屋工業大学）

独自アクセラレータアーキテクチャについて、昨年度定めたレファレンス設計についてシ

ミュレーションを行い、N5ないしN3以降のプロセスルールでの動作速度・電力性能、アプリケーションカーネルでの性能を評価する。またDRAMダイをプロセッサダイと3次元実装する方針を採用した場合について動作速度・電力性能、アプリケーションカーネルでの性能を評価する。

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によるアプリケーション実行時性能評価を行う。

⑥ 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。

⑦ 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。

⑧ 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。

⑨ 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。

⑩ 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。

⑪ 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。

⑫ 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。

⑬ 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。

⑭ 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。

⑮ マテリアルサイエンス応用アプリケーションの検討
（国立大学法人東海国立大学機構名古屋大学）
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、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。

⑯ 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。

⑰ 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。

⑱ 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及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。

⑲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、必要に応じて調査ある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)

○　「富岳」の成果及び課題
○　評価指標として考慮すべき項目
○　技術的課題や制約要因の抽出
○　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、

● 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。
● 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、

○　「富岳」の成果及び課題
○　評価指標として考慮すべき項目
○　技術的課題や制約要因の抽出
○　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

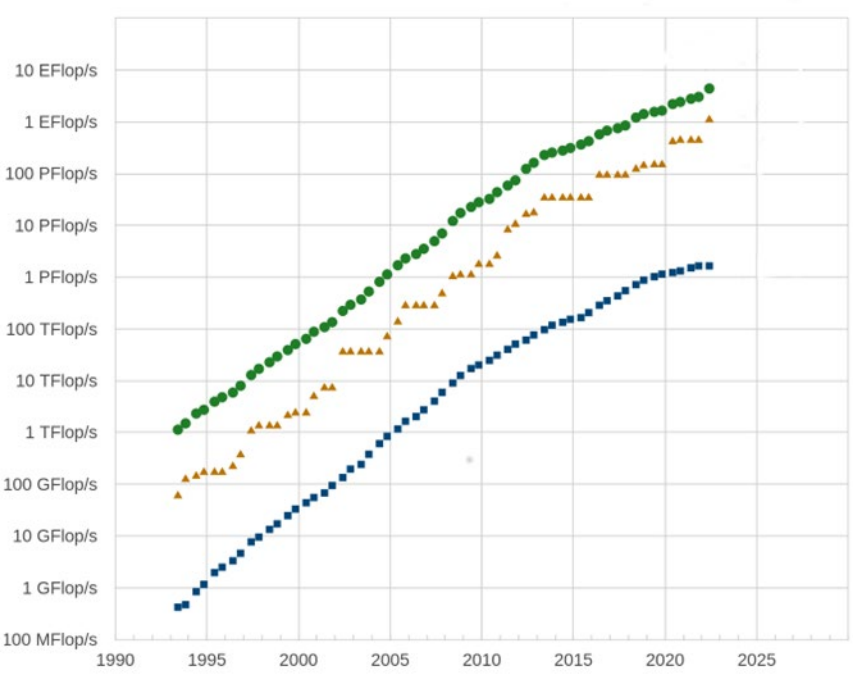

図1: Top500のトレンド

図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。

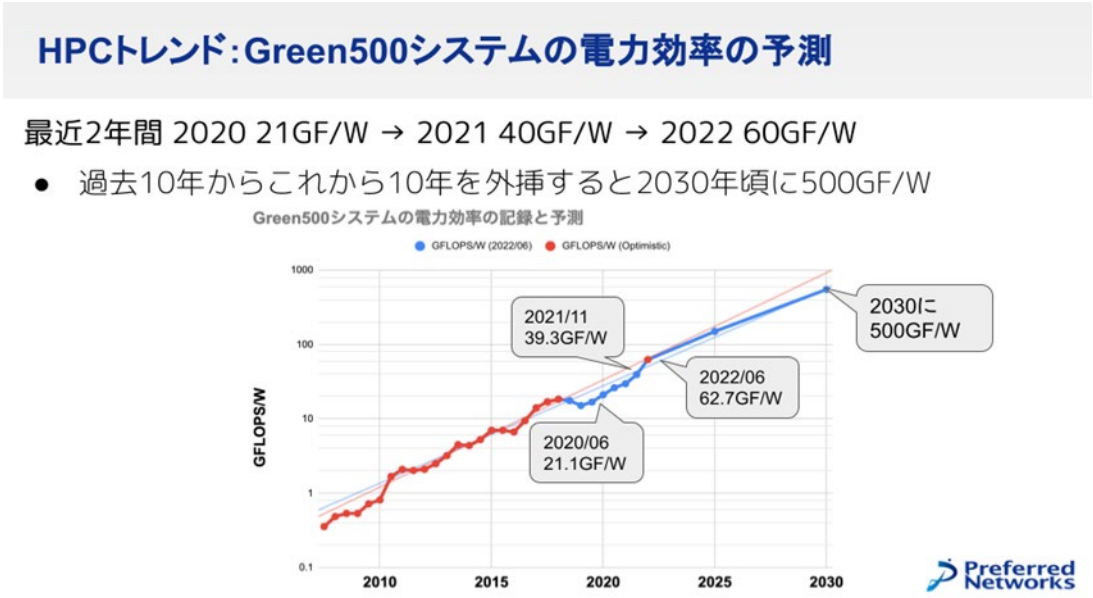

図2: 電力性能のトレンドと2030年予測

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5,000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5,000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 Alpha21064 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 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 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

- 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
- 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
- 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

- B/F 値の大幅な低下
- 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。

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を持つ 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。

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 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。

(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。

(a) 演算、L1 および L2 のレイテンシが非常に大きいこと
(b) 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
(c) x86 に比べて半分程度の OoO(アウトオブオーダー実行)資源
(d) x86 に比べて半分程度の L1,L2 バンド幅
(e) 比較的貧弱な分岐予測機構

この中で、(a)、(b)、(c)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(b)のアーキテクチャレジスタ数の問題は抱えているが、(a)、(c)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、(b)、(e)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、OoO 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、

- なるべく長いベクトルを
- 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 アーキテクチャも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おこな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

2023 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、電力性能のついての予測を更新すると共に、前年度に必要性が明らかになった DRAM バンド幅向上について具体的な検討を行った。詳細は④～⑥で述べる。

② システムソフトウェア・ライブラリ調査研究

システムソフトウェア検討チームでは、独自アクセラレータのレファレンスアーキテクチャをアーキテクチャ調査研究グループと共同で定義し、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を定め、処理系の開発を進めた。具体的には、OpenCL および OpenACC を MN-Core 後継アーキテクチャで実行する方法を定義し、試験的な実装と性能評価を進めた。詳細は⑦、⑧で述べる。

③ アプリケーション調査研究

昨年度定めた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、国立大学法人名古屋工業大学)

アーキテクチャ調査研究全体の統括を行うと共に、2023 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、前年度に必要性が明らかになった DRAM バンド幅向上について具体的な検討を行った。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 2 である。

表 2: 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及び、将来予測結果を示したものが、表 3 である。

表 3: 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	Hopper Next	Hopper Next Next
製品発表年		2016	2017	2020	2022	2025	2028
プロセス(量産開始年)		N16(2015)	N12(2017)	N7(2018)	N4(2021)	N3E(2023)	N2(2026)
Spec	演算性能@FP16(TF)	21.2	125	312	989	1721	2761
	演算性能@FP64(TF)	4.7	7.8	19.5	67	108	173
	FP16/FP64	4.5	16.0	16.0	14.8	16	16
	周波数(MHz)	1328	1455	1410	1780	1905	2038
	TDP(W)	300	300	400	700	843	1060
	電力効率 GF/W@FP64	15.67	26.00	48.75	95.71	127.62	162.78
	B Tr 数	15.3	21	54	80	120	180
	Die Size(mm^2)	610	815	826	814	814	814

2023 年末になって、NVIDIA からは H200 と B200 という２つの新製品が発表されたが、H200 は H100 からメモリが HBM3 から HBM3e に変更され、メモリバンド幅が 3 TB/s から 4.8 TB/s になった以外に変更はない。B200 は TSMC の N4P プロセスで製造されるダイを２つ同一パッケージに搭載して、FP16 及び FP8 では 2.5 倍の性能になる一方 FP64 では 1.3 倍程度にとどまるものになっており、また電力性能の改善も小さい模様である。すなわち、今後の NVIDIA GPGPU は、表３の予測よりも AI 性能で若干高くなる一方、FP64 性能では大きく低下する可能性があることが明らかになった。これは、以下の２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 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 性能と FP16/8 性能にはトレードオフがある。

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。実際、H100 は長期に渡る供給不足が続いており、価格も高騰していた。TSMC の CoWoS 生産ラインの増強や、B200 の発表もあり、2023 年末以降ようやく H100 の供給には多少の余裕がみえてきた。２は、設計として FP64 と FP16/8 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。

NVIDIA 社の GPGPU の世代間の演算性能及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 及び 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 3 内の Hopper Next 及び Hopper Next 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 Hopper Next Next の世代、若干の遅延が生じると N3E の Hopper Next 世代となると思われる。それぞれの倍精度演算性能、電力、電力効率は、Hopper Next が 108 TF,843 W,127.62 GF/W@FP64、Hopper Next Next は 173 TF,1060 W,162.78 TF/W と予想される。すでに述べたように、実際には表３の数値に比べて FP64 性能が大きく低下する可能性が高いことには注意が必要である。

表 4: 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	MI300 Next	MI300 Next 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)	N2(2026)
Spec	演算性能 @FP16(TF)	8.2	26.4	26.5	186.4	383	1300	2225	3569
	演算性能@ FP64(TF)	0.512	0.768	6.6	11.54	47.85	81.3	139	223
	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	2247	2404
	TDP(W)	175	300	300	300	560	750	812	1022
	電力効率 GF/W@FP64	2.93	2.56	5.75	5.01	85.45	109.0	171.13	218.28
	B Tr 数	8.9	12.5	13.2	21	58.2	153.0	116.4	174.6
	Die Size(mm^2)	596	510	331	763.2	1448	>2000?	1450	1450

表 4 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、MI300Next は倍精度で 139 TF,812 W,171.13 GF/W、MI300Next Next では 223 TF,1022 W で 218.28 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 MI300X 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。

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、アクセラレータの金額の分析を実施した。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 5 である。

表 5: 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 6 である。

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。

表 6: 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]	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

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である。MAU-Shuttle についてはシミュレーション、他は実測である。

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、次世代システムの電力性能を見積もる。

現在までの MN-Core の数値を表 7 に示す。

表 7: 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

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。

- NVIDIA GPU
- PEZY-SCx
- Sunway SW26010(Pro)
- Intel Max GPU
- 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もち、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も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もち、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 次元格子構造をもち、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８に外部メモリの主要なパラメータについてまとめる。

表 8: 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 キャッシュのバンド幅も 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、メモリ階層構造及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 をもってくればよい。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 1pJ 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 0.001 ということは、オフチップメモリ 1 語アクセスにつき 1 万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。

(a) 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
(b) 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
(c) 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
(d) 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。

なお、Intel MAX GPU を搭載した大規模システムである Aurora は、2023 年 11 月の Top500 でようやく姿を現し、目標性能を実現できていないことが明らかになった。HPL 性能が名目ピーク性能の半分以下にとどまっており、消費電力が大きく目標を上回っている模様である。このようになった最大の原因は、浮動小数点演算性能に対して L2 キャッシュのバンド幅が大きく、またダイサイズも大きく、３次元実装等も使われているために、チップ内部でのデータ移動の消費電力が大きくなっているためと推測される。

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(c)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 FP64 70 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 900-1000 GF/W を N1.4 プロセスで実現することは可能と考える。

詳細は 4-5-1 で述べる。

4-4-5. 3D 実装メモリの検討

DRAM ベンダーに 3 DDRAM 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 及び 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。

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 及び 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

3D DRAM 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おこな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。

4-4-6. 演算器設計の検討

さらに、昨年度定めたレファレンス実装と比べ、より省面積・省電力・高並列・低レイテンシとなる新たな演算器回路アーキテクチャ設計を行った。

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。詳細は「次世代計算基盤選択肢案提案書」(別紙)にまとめた通りだが、ここでは概要を述べる。

4-5-1. 構成概要

1ノードの構成を以下にまとめる。

CPU 高性能 RISC-V(A社後継)
　チップレットあたりのコア数 Naコア
　クロック周波数 Nb GHz
　メモリ LPDDR5/6
　容量 128-256 GB
　バンド幅 1 TB/s
アクセラレータ PFN MN-Core4(現行のMN-Core2の2世代先)
　ソケットあたりのチップレット数 1個
　チップレットあたりのコア数 Nc個
　クロック周波数 Nd GHz
　メモリ B社の改良版 容量 96-192 GB
　バンド幅 Ne TB/s
半導体テクノロジー 2 nm

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ノードとなる。

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おそらく大きく上回り、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。

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。

表 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		C 社の改良型
	容量、I/O 性能	500 PB, Nf 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）	3.6 PFLOPS
演算性能　単精度（ノードあたり）	14.4 PFLOPS
演算性能　半精度（ノードあたり）	57.6 PFLOPS
演算性能　AI 向け半精度（ノードあたり）	57.6 PFLOPS
演算性能　整数演算（ノードあたり）	0.9 POPS
演算性能　AI 向け整数演算（ノードあたり）	230 POPS (FP8)
メモリ容量（ノードあたり）	3 TB
メモリバンド幅（ノードあたり）	240 TB/s
消費電力（ノードあたり）	10 kW
演算性能　倍精度（システム全体）	9 EFLOPS
演算性能　単精度（システム全体）	36 EFLOPS
演算性能　半精度（システム全体）	144 EFLOPS
演算性能　AI 向け半精度（システム全体）	144 EFLOPS
演算性能　整数演算（システム全体）	2.25 EOPS
演算性能　AI 向け整数演算（システム全体）	576 AIEOPS
メモリ容量（システム全体の合計）	9.4 PB
メモリバンド幅（システム全体の合計）	750 PB/s
消費電力（システム全体）	25 MW
電力性能比　倍精度（システム全体）	360 GFLOPS/W
電力性能比　単精度（システム全体）	1.44 TFLOPS/W
電力性能比　半精度（システム全体）	5.76 TFLOPS/W
電力性能比　AI 向け半精度（システム全体）	5.76 TFLOPS/W(AI)
電力性能比　整数演算（システム全体）	90 GOPS/W
電力性能比　AI 向け整数演算（システム全体）	23.0 TOPS/W(AI)

⑤ CPU 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、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。以下ではこの評価について述べる。

[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。

[評価方法] ここでは特に詳細な結果が得られたTenstorrent社のCPUの評価について述べる。この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。

FDPSはC++で記述されており、これを各評価環境向けにコンパイルして性能を評価した。まずFDPSの実行をプロファイラによって解析し、プログラムの実行時間に対して支配的な関数を探した。それらの関数の中からアクセラレータで実行可能な部分と、シーケンシャルな部分を分類し、後者を評価対象とした。

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。

そこで、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改造した。測定を行うX86-64 CPUとしては、現在最も高速な実装の１つであるAMD社のものを複数種類用いた。また、RISC-V CPUについては企業と交渉の結果、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たため、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、上記の課題を解決し、該当する関数同士を適切に比較できるようになった。

［結果］ 上記の解析の結果、Tenstorrent社およびSiFive社のRISC-V CPUはAMD社のx86-64 CPUの性能には総合的には及ばないものの、高性能計算においてシーケンシャルな部分を処理するCPUとして見た場合には比較的良好な性能を持ち、RISC-V CPUがボトルネックとはならずに十分な性能を提供できていることがわかった（両社における機密保持の観点より、詳細な性能差などの分析はここでは省略する）。特にCPUとアクセラレータ間のデータ転送に必要な時間が大きく改善されることを考慮すると、両者が密結合されたシステムを構成することで大きな性能向上を得られる可能性がある。

⑥ 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）

昨年度、計算ノードのレファレンス設計にあたって、ネットワーク側に必要な性能・機能を評価した。その結果、HPLでは、ネットワークを2次元格子(直接網)にマップした場合、4本のリンクのそれぞれの転送バンド幅が 11.5 GB/s以上必要となった。つまり、合計でほぼ50 GB/sとなる。また、HPCGでは、レファレンスアーキテクチャのアクセラレータの主記憶を32 GBとした場合、500^3程度のグリッドを格納する必要がある。B/F = 0.1を仮定すると実行効率は1.6%となり、この時のイテレーションの実行時間は1.2ミリ秒となる。袖のサイズは12 MB程度なので、通信時間を計算時間の1/3の0.4ミリ秒として必要なネットワーク帯域は6方向合計で30 GB/s程度となった。

この要求を満たす相互結合網を検討した。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、光電融合技術が現実的に利用可能である。例えば、光サーキットスイッチ(OCS:Optical Circuit Switch)と電気ネットワークを統合した光電ネットワーク構成が盛んに実用化されている。具体的には、Noctua 2スーパーコンピュータ(CALIENT S320 OCS)[1]、Google TPUv4クラスタ[3,4]、Jupiterデータセンターネットワーク[2]が挙げられ、多くのOCSの試作機が研究開発されている[5,7,8]。いずれのOCSも上記の要求帯域を満たしている。

OCSを用いたHPC向け相互結合網は、図3に示した3通りに分類されるが、近年は電気パケットスイッチ(EPS: Electric Packet Switch)とOCSを相補完的に使うハイブリッド型[9](図3.C)が主流である。そこで、ハイブリッド型を検討した。

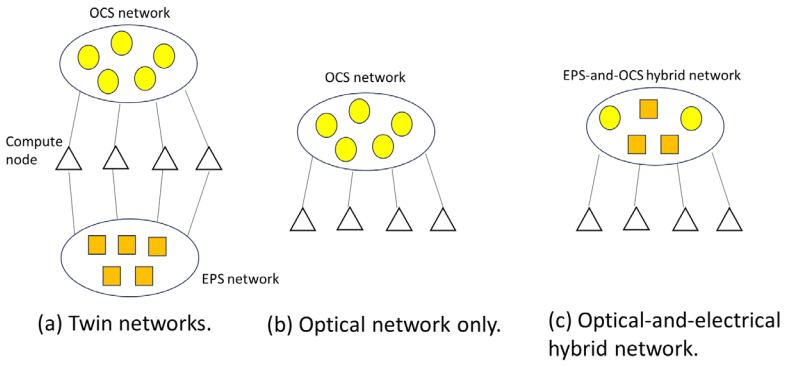

図3: 光サーキットスイッチと電気パケットスイッチを用いた結合網の分類

これらの既存のOCS技術は、(1)ユーザジョブ間の通信の干渉を完全に避けるため、(2)シス

テムの拡張性を確保するため、(3)ジョブ内の通信帯域を一時的に大きくするため、(4)耐故障性を向上するため、などの複数の目的で利用することができる。

ネットワークトポロジは、間接網としてFatツリー、Clos網のルートあるいはリーフ[10]の電気パケットスイッチをOCSにおきかえた構成、直接網としてDragonflyネットワークトポロジのラック間グローバルリンクをOCSに接続する構成を検討した。特に、Dragonflyネットワークトポロジの場合、ラック間グローバルリンク数を抑制した場合でもOCSの活用によりジョブ内の通信帯域を大きくする運用が可能であり[6]、ポート単価520ドル程度が期待できるためコストを抑えることも期待される[5]。

OCSの試作機、製品は以下の3つに分類される:(1)MEMSを用いた一般的なOCS(ミリ秒単位の経路切り替え時間)、(2)部分的な経路のみを設定可能なPartially reconfigurable OCS(ミリ秒単位の経路切り替え時間)[7]、(3)調整可能なレーザーなどを用いたカスタムOCS(ナノ秒単位の経路切り替え時間)[8]。(2)については、ラウンドロビン方式で通信対を更新する特殊な方法をとるためHPC分野での利用は難しいことが予想される。そこで、(1)と(3)を対象に、直接網、間接網を対象にした適性についても検討した。

以上の検討に基づき、OCSを用いた相互結合網の開発として2つの方向が考えられる。

(a)汎用MEMS OCSスイッチを用いた相互結合網の開発、および実運用での安定性・性能向上を目的とした応用手法の研究開発
(b)プロトタイプ実装やシミュレーションに基づく、高速な再構成が可能なカスタム化OCSを用いる相互結合網のピーク性能の向上を図る研究開発

[1] Noctua 2スーパーコンピュータ, https://pc2.uni-paderborn.de/de/hpc-services/available-systems/noctua2
[2] Leon Poutievski,et.al., Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking. SIGCOMM, pp.66–85, 2022
[3] Norm Jouppi,et. al., TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. ISCA, pp.1–14, 2023
[4] Hong Liu,et.al., Lightwave Fabrics: At-Scale Optical Circuit Switching for Datacenter and Machine Learning Systems. SIGCOMM, pp.499–515, 2023
[5] Weiyang Wang, et.al., TopoOpt: Co-optimizing Network Topology and Parallelization Strategy for Distributed Machine Learning Training Jobs, NSDI, pp.739-769, 2023
[6] Guangnan Feng, et.al.,GRAP: Group-level Resource Allocation Policy for Reconfigurable Dragonfly Network in HPC. ICS, pp.437–449, 2023
[7] William M. Mellette, et.al., RotorNet: A Scalable, Low-complexity, Optical Datacenter Network, SIGCOMM, pp.267–280, 2017
[8] Hitesh Ballani, et.al., Sirius: A Flat Datacenter Network with Nanosecond Optical Switching. SIGCOMM, pp.782–797, 2020
[9] Nathan Farrington, et.al., Helios: a hybrid electrical/optical switch architecture for modular data centers. SIGCOMM, pp.339–350, 2010
[10] Yiting Xia, et. al., A Tale of Two Topologies: Exploring Convertible Data Center Network Architectures with Flat-tree, SIGCOMM, pp.295–308, 2017

⑦ 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った。具体的には、アプリケー

ションを記述可能なデータ並列言語の詳細な仕様を定めた。データ並列言語のプロトタイプを実装するために、OpenCL APIをベースとしてホストAPIを定めてライブラリとして実装した。このAPIには、MN-Coreボードの初期化、メモリの割り当ておよび転送、演算カーネル実行のための関数を実現し、このAPIを利用することで、ユーザーがMN-Core用命令で記述した演算カーネルをハードウェア上で実行可能となった。

このAPIライブラリを利用して、MN-Core用のN体計算カーネルの性能評価をハードウェア上でおこなった。DRAMからデータをPEに分配・回収する命令を含めて粒子数(N)に応じて演算カーネルを生成するプログラムを作成し、いくつかのNについて演算性能の測定と演算誤差の評価を行った。演算を倍精度で行った場合、粒子数Nが64 K, 12 8K, 256 Kの場合の1ステップ当たりの演算性能は1029, 1196, 1545 GFLOPSであった。ホスト側で計算した結果と比較して演算誤差の平均は約10^{-15}となった。

```
__kernel void mncore_kernel(__local double *GRF,   __local double *LM0, __local double *LM1)
{
   const double omega = 1.0e-3;

   double p[NBB][NBB][NBB];
   double A1[NB][NB][NB];
   double wrk1[NB][NB][NB];
   double wrk2[NB][NB][NB];
   double bnd[NB][NB][NB];

   const int nb_size1 = NBB*NBB*NBB;
   const int nb_size2 = NB*NB*NB;

   load_lm((double *)p,      LM0, nb_size1);

   load_lm((double *)A1,     LM1+0*nb_size2, nb_size2);
   load_lm((double *)wrk1, LM1+1*nb_size2, nb_size2);
   load_lm((double *)bnd,   LM1+2*nb_size2, nb_size2);

   double gosa = 0.0;
   for(int k = 1; k <= NB; k++) {
      for(int j = 1; j <= NB; j++) {
         for(int i = 1; i <= NB; i++) {
               double s0, ss;
               s0 = A1[i-1][j-1][k-1] * p[i+1][j ][k ]+A2[i-1][j-1][k-1] * p[i ][j+1][k ]
                 + A3[i-1][j-1][k-1] * p[i ][j ][k+1]+
                  B1[i-1][j-1][k-1] * ( p[i+1][j+1][k ] - p[i+1][j-1][k ]
                                                        - p[i-1][j+1][k ] + p[i-1][j-1][k ] )
                  + B2[i-1][j-1][k-1] * ( p[i ][j+1][k+1] - p[i ][j-1][k+1]
                                                        - p[i ][j+1][k-1] + p[i ][j-1][k-1] )
                  + B3[i-1][j-1][k-1] * ( p[i+1][j ][k+1] - p[i-1][j ][k+1]
                                                        - p[i+1][j ][k-1] + p[i-1][j ][k-1] )
                  + C1[i-1][j-1][k-1] * p[i-1][j ][k ] + C2[i-1][j-1][k-1] * p[i ][j-1][k ]
                  + C3[i-1][j-1][k-1] * p[i ][j ][k-1] + wrk1[i-1][j-1][k-1];
               ss = (s0 * A4[i-1][j-1][k-1] - p[i][j][k])*bnd[i-1][j-1][k-1];
               gosa = gosa + ss*ss;
               wrk2[i-1][j-1][k-1] = p[i][j][k] + omega *ss;
         }
      }
   }

   store_lm(LM1+3*nb_size2, (double *)wrk2, nb_size2);

   LM1[nb_size2*4+1] = gosa;
}
```

図4: OpenCLをベースにしたAPIを使った三次元ステンシル計算の演算カーネルの例

幅広いアプリケーションの性能評価をおこなうために、データ並列言語として、C言語をベースとしたカーネル記述用コンパイラのプロトタイプを実装した。この実装には、LLVMプロジェクトによるコンパイラフロントエンドclangを利用した。clangで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、そのIRをMN-Core用命令に変換する。これを使い、N体計算カーネルや、ステンシル計算のカーネルからMN-Core用命令を生

成し、シミュレータとハードウェア上で性能評価と計算結果の検証をおこなった。三次元ステンシル計算の例として姫野ベンチを元に修正を加えて実装した演算カーネルを示す（図4）。

この演算カーネルは、PEのローカルメモリ(LM0, LM1)にあるデータの演算のみを記述するものである。実アプリケーションでは、複数のPE間でのデータ転送や総和処理は別途記述する必要がある。階層的に接続されているPEの間でデータ転送等を実現するための手法について詳細を検討し、一部の実装を行った。

また、ユーザーが容易にMN-Coreへ移行できるようにするために、今年度はコンパイラ指示文（ディレクティブ）形式のAPIであるOpenACCについて本格的に検討及び実装を開始した。

OpenACCではGPUのような共有メモリ型のアーキテクチャを前提にしているが、これをこのまま分散メモリ型アーキテクチャのMN-Coreに当てはめようとすると、ユーザーが直感的に操作できるようなインターフェースではなくなり、HPCにおける科学計算の性能最適化に適さなくなることが検討の中で想定された。一方で、アクセラレータを対象としたものではないものの、ノード間での分散メモリ並列計算のためのAPIとして、MPIと、それをユーザーがディレクティブ形式で記述できるようにしたHPFrotran及びXcalableMPが存在する。分散メモリ型のAPIの利点として、共有メモリ型では制御が困難なメモリ通信について明示的に記述できるため、ユーザーによる最適化が容易となることが挙げられる。

```
void stencil() {
    ...
#pragma acc enter data create(temp[0:128][0:128][0:128]) ¥
copyin(b[0:128][0:128][0:128], a[0:128][0:128][0:128])  ¥
// MABの要素数は省略可
L2(a,b,temp[4][4]) L1(a,b,temp[2][4]) mab(a,b,temp) ¥
// 各次元の袖の要素を指定[マイナスの要素数:プラスの要素数]
shadow(a[1:1][1:1][1:1])
    ...
    for(count=0; count<N; count++) {
    ...
// reflectで袖交換の発生をコンパイラに知らせる
#pragma acc parallel loop reflect(a)
        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*b[i][j][k];
        } } }

        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 exit data copyout(a[0:128][0:128][0:128]) delete(b,temp)
    ...
}
```

図5: OpenACCをベースにしたAPIを使った三次元ステンシル計算の演算カーネルの例

そこで、MN-Core対応のOpenACCでは、MN-Coreのアーキテクチャに合わせる形で、これらのAPIを参考に既存のOpenACCを一部拡張することとした。具体的には、ユーザーがCPUとMN-Core、MN-Core内の階層間、MN-Core内のPE間のデータ転送を行う際に、明示的にデータ分割方法を示して、転送を指示できるOpenACCのclauseについて詳細に検討を行った。なお、OpenACCによる記述がどの程度簡潔にできるかについては、コンパイラのコード解析の実装コストと密接にトレードオフの関係にある。つまり、高度な解析の実装には開発期間が必要で

あり、その実装までは、コード中にある程度冗長に指示を明示する必要があることから、簡易な実装でも変換可能とすることを考慮してインターフェースの検討を行った。

今年度検討したAPIを用いたサンプルコードとして、3次元ステンシル計算の例を示す（図5）。赤字の部分がMN-Core用の拡張であるが、GPUからの移行のしやすさを考慮して、なるべく既存のOpenACCと記述上の差異が少なくなるようにした。また、拡張のうち、袖交換の指示記述についても、既存の計算科学ユーザーが移行しやすくするため、前述のノード間並列APIと互換性のあるものとした。

検討と同時に、上記のMN-Core対応OpenACCのプロトタイプ実装についても進めた。まず、OpenACCコンパイラの実装としては、OpenACCから前述のOpenCL APIベースのホストコードとC言語ベースのカーネル記述言語（以下、デバイスコード）へ変換する方式を採用することにした。こうすることで、言語処理系をOpenACCとデバイスコードで独立して複数作る場合に比べ、開発コストを抑えることができる。また、OpenACCから低レイヤーな記述を部分的に呼び出すことによる柔軟な最適化が可能となる。GPUでは、実際にOpenACCからCUDAを呼び出すことにより、局所的にユーザーによる詳細な最適化を行うことが可能である。

なお、デバイスコードはC言語ベースであることから、OpenACCプロトタイプ実装についてもC言語版を優先して進めることとした。

OpenACCとデバイスコードは並行して検討を進め、OpenACCコンパイラ実装に随時取り込んでいる。プロトタイプ実装では、1次元目に相当する最外ループがPE数である8192の倍数回のネストループついて、最外ループを分割して、MN-Core上の全PEでエレメントワイズな多次元配列の計算を行うコードが変換できるようにした。なお、変換後のホストコード及びデバイスコードは検討途中の仮のものであるため、引き続きコンパイラ実装と合わせて検討を進めていく。

⑧ DSL 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）

本年度は粒子法用シミュレーションコード開発フレームワーク(FDPS: Framework for Developing Particle Simulators)上に、近年提案されたParticle Mesh Multipole Method(PMMM)対応を行った。

PMMMでは計算領域を8^3程度の荒い規則格子に区切り、FMM (Fast Multipole Method: 高速多重極法) 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。FMMでは多重極展開係数を局所展開係数に変換する演算(M2L)が大きな計算量を占めるが、PMMMではここに高速フーリエ変換(FFT)を用いた畳み込み積分を適用し高速化させる。Particle-Mesh法などもFFTにより遠距離力成分を計算するが、PMMM法では荒く切った格子でのFFTを行えばよく、また展開係数ごとに独立に計算を行う事ができるため、大規模システムでの並列化効率が良い。

本年度はPMMMとBarns-Hut tree法を組み合わせられるようFDPSの改良およびPMMMモジュールの開発を行った。まず、FDPS側でPMMMの規則格子に沿うようにツリー構造を構築し、各ツリーノードの多重局展開係数を求める。次に近傍セルからの相互作用を計算する。その後、PMMMモジュールに計算したツリー構造等を渡すことで、FDPS側で計算した多重局展開から

局所展開係数を計算し相互作用を行う事ができる。

計算結果はエヴァルト法で計算した結果と一致していることを確認した。また宇宙論的N体シミュレーションも行い、従来と同様の計算ができていることが確認できた。

⑨ 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を抽出しての評価が中心となっている。これに関しては会津大学で開発中の、C言語をベースとしたカーネル記述用コンパイラがうまく対応できるようになれば、アプリケーション研究者にとっては面倒な部分がかなり解消されることになる。この点で種々のアプリケーションを取り込む上で敷居を低くすることにつながり、期待が高い。

一方で、ものづくりアプリケーションの分野ではプログラムのPython版を経由することでMN-Core性能評価や最適化を検討できることを示してくれた。この方法は、他のアプリケーションでも利用可能であり、もう一つの期待の方向である。

さらに、これまでのアプリケーション分野での検討に加えて、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。そこで、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。このアプリケーション・ソフトウェアの開発を行った企業と協議し、ソフトウェアの使用の許諾を得るとともに、AMD社製128コアのPCの導入を行った。

⑩ 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)

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。具体的には、創薬については、分子動力学シミュレーションソフトウェアLAMMPSのベンチマークから一つ選び、ボトルネックである相互作用計算カーネルのMN-Coreでの最適化実装を与え、相互作用リストによる粒子データの索引がボトルネックとなることを明らかにした。今年度は、粒子法についてOpenCLライクな表記での高効率な実装の可能性を検討し、適切なグループ化を行うスキームでは索引参照は問題にならないことを確認した。深層学習応用については、独自アクセラレータアーキテクチャの先端プロセスでの性能諸元を仮定して、MN-Coreで動作確認済みの既存アプリケーションのいくつかについて性能評価を行った。

⑪ 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。具体的には、国の全ゲノム等実行計画の難病解析での実施プロトコルに採用されているParabricksについてパフォーマンスをまとめ、比較対照とした。

全ゲノムシークエンス解析において、対応が必要とされるゲノムデータのファイルサイズは以下の通りである（500人の実データの平均値を元に算出：表10）。

表10: 全ゲノムシークエンス解析において、対応が必要とされるゲノムデータのファイルサイズ

ファイル形式	1人[GB]	2,000人[TB]	20,000人[TB]
fastq	55	110	1,100
bam	65	130	1,300
vcf	22	44	440

一人について実際の実行時間は以下の通りである（東京大学医科学研究所Shirokane5を使用）。GPU枚数とSSDのみ条件を振っている。

表11: 一人分の全ゲノムシークエンス解析にかかる実行時間

	メモリ指定	CPU** コア数	fq2bam (m)	Haplotype Caller(m)	bgzip + validate_pbrun (m)
V100* 1枚 SSDあり	256 GB	16	157.70	108.9	129.6
V100 1枚 SSDなし	256 GB	16	164.65	102.1	
V100 2枚 SSDあり	256 GB	16	78.52	56.2	
V100 2枚 SSDなし	256 GB	16	87.20	56.5	

* NVIDIA Tesla V100 SXM2
** Intel Xeon Gold 6126

20,000人について実施すると考えるとき、V100 2枚でのべ3,750日要する。V100 16枚であれば469日である。目標としては、この解析を数ヶ月以内に終えることが望ましい。

国内では、東北メディカル・メガバンク10万人のWGSの完了が近い。また、国際的な取り組みに目を移すと、米国TOPMedプロジェクトやAll of Usプロジェクトは5万人、英国UK Biobankは50万人のWGS解析プロジェクトを完了し論文報告しているため、上記の人数に基づくパフォーマンス指標を設定することは決して無茶なものではない。ただしこれら、現実に数万人を解析しているプロジェクトでは、上記をそのまま実施せず簡略化したプログラムをパイプラインの一部に作用することで高速化している面もある。

上でみたように、現状では Parabricks で NVIDIA V100 2 枚を使って 1 人当たりの解析に 270 分程度かかっている。これでは 10 万人のデータの処理は困難であり、ポスト「富岳」時代に想定されるさらに規模が大きな解析には対応困難である。

一方、昨年になって、CPU ベースでこの解析を 20 分程度で行えると主張する解析ツールが複数アナウンスされた(QIAGEN CLC Light Speed Module
https://digitalinsights.qiagen.com/plugins/clc-lightspeed-module/ AAS-G1

https://www.dnaform.jp/ja/products/aas-g1/ 等)。

bwa-mem の場合、レファレンスゲノムに対して BWT を行ってできるテーブルと補助データ構造を、リード配列に応じてほぼランダムにアクセスするため、主記憶のランダムアクセス性能がボトルネックとなる。アクセスする範囲は数 GB 程度となる。bwa-mem の実行プロファイルを調査した結果、bwa-mem は CPU 向けには高度にチューニングされており、メモリアクセス回数から期待できる性能が出ているとわかった。

AAS-G1 については国内企業の開発であり、神戸大学と共同研究契約もあるため、アルゴリズムとベンチマークカーネルを提供してもらって性能解析を行った。現状では 2 ソケットで 128 コアの AMD CPU ベースのシステムで、標準的な x30 程度のゲノムデータを 8 分程度で処理できる。

AAS-G1 では、bwa-mem が使っている BWT テーブルの代わりに suffix tree を使い、これをさらに 2 階層に分割することで各ステージではツリーデータ構造全体がキャッシュメモリに入るようにすることでメモリアクセス自体を減らし、さらにキャッシュヒット率をあげている。このことにより、bwa-mem に比べて大幅な高速化を達成している。また、Parabricks ではあまり高速化されていない variant caller も、アルゴリズムの改良により大きく高速化している。

AAS-G1のツリーサーチ部分をMN-Core アーキテクチャで実行する可能性について検討した。基本的に、ツリーのデータサイズを小さくしているため、これがローカルメモリに入るところまで小さくすることで非常に高速なツリーサーチができる。また、高速なDRAMがあるので、ツリー構造自体はホストからの転送ではなく、DRAMからのロードにすることができる。このため、想定するポスト「富岳」アーキテクチャでは、ホストからリードデータを転送する速度がボトルネックになると予測された。

想定するポスト富岳構成では、ホストCPUとアクセラレータの間のデータ転送バンド幅が1 TB/sと現状の典型的な2ノードCPUサーバーの4倍程度あるため、総合的には10倍程度の加速が期待でき、x30のゲノムデータ解析を1分程度で行なうことが期待できる。これは、現状でのNVIDIA V100 2枚を使った処理の300倍の速度であり、100万人からさらに大きな規模のデータを容易に解析可能になると期待できる。

⑫ 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)

地震シミュレーションの対象は、長期の地殻変動、地震直後の地殻変動、断層から地表までの地震動伝播、地表近傍の堆積層内での地震動の増幅、地盤と構造物の連成を考慮した構造物の地震時挙動など多岐に渡る。これらの挙動は対象物の幾何形状及び地表における応力フリーのなどに代表される境界条件の影響を強く受けるため、幾何形状を適切にモデル化出来、かつ、境界条件の取り扱いに優れた、低次非構造四面体要素を用いた有限要素解析が望まれる。一方で、地震シミュレーションでは領域が広大かつ要求分解能が高く大規模問題となる。加えて、地殻や地盤などの構造情報の不完全さや観測データを用いた最適化・データ同化のニーズも高まりつつあるため、これらのシミュレーションのさらなる高速化が望まれる状況にある。

そこで、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った。具体的には、地震動評価シミュレーションの中核部分をEbE-methodとして抽出し、既存の計算機である「富岳」に搭載されているA64FX CPU(Armv8.2-A)及びABCIのNVIDIA A100 GPU(SXM4 40 GB)における性能を測定した。これにより、MN-Coreでの実測値と比較することで、当アプリケーションの適合性を評価できる状況となった。

具体的には、EbE-methodは、シミュレーションのコアカーネルである非構造四面体二次要素による疎行列ベクトル積計算を抽出したものである。現状の計算機よりも演算性能に対してメモリ容量・メモリ帯域が相対的に小さくなる計算機においても性能を発揮できるような検討を進めていくため、本コアカーネルでは行列格納型の疎行列ベクトル積ではなく、行列ベクトル積の都度要素行列を生成し、疎行列ベクトル積をオフラインで計算するEbE-methodにより演算を実施している。表12にA64FX CPU 1個、及び、A100 GPU 1個で得られた性能値を示す。この演算ではCRSなどにより疎行列をメモリ上に保持しなくてもよいため、省メモリとなるだけではなく、アーキテクチャによっては疎行列をCRSで保持した疎行列ベクトル積よりも高速に疎行列ベクトル積が実行可能となる場合がある。実際、演算性能に対してメモリ帯域が相対的に大きいA64FX CPUではCRSとEbE-methodは同程度の計算速度になるが、演算性能が相対的に高いA100 GPUではEbE-methodがCRSの約3倍高速になることが分かった。今後、このような省メモリ容量・省メモリアクセスのアルゴリズムの有効性の更なる検証が期待される。

表12: Elapsed time and obtained performance of EbE-method computed in FP32.

System	Elapsed time	Ratio to FP32 peak	Ratio to peak memory bandwidth
A64FX	0.70 s	9.6%	7.5%
A100 SXM4 40 GB	0.14 s	12.7%	12.7%

⑬ 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、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、Python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Python再実装〉
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。また、性能評価にはソフトウェアパッケージ「IcoAtmosBenchmark v1」を用いる。本パッケージはNICAMのステンシル演算カーネルを抽出したものであり、「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。本パッケージには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）とステンシル計算に用いるメトリクス計算カーネル、さらにプロセス間通信カーネルが含まれている。このうち、６つ

の演算カーネルをPythonを用いて再実装を行った。Pythonを選択した大きな理由は、近年Google社が開発したPython上の機械学習フレームワークであるJAXの利用を想定したことによる。JAXが持つ主な機能は、1)自動微分の計算が容易、2)その場(Just In Time:JIT)コンパイルによる計算高速化、3)vmapによるベクトル化、4)pmapによる並列化である。このうちHPCにとって有用なのはJITコンパイラとしての機能である。NumPyを用いて、普通のPythonのソースコードとして記述された任意の数値計算処理を、その場でコンパイルしてCPUのみならず、GPUやTPU等のアクセラレータ上で高速に実行することが可能になり、HPCアプリケーションを書く研究者側の自由度が向上する。MN-Coreに関しても、JAXを用いて記述されたPythonコードをMN-Core上で動作させるための開発が行われており(https://tech.preferred.jp/ja/blog/jax-on-mncore/)、将来的にJAXは本調査研究で対象とする独自アクセラレータアーキテクチャでサポートされることが期待される。

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
```

図6: 演算カーネル（水平拡散）のソースコードの一部

Python Ver.1 (抜粋)

```
for l in range(0, lall):
    for k in range(0, kall):
        for d in range(0, nxyz):
            for g in range(gminm1-1, gmax):
                ij     = g
                ip1j   = g + 1
                ip1jp1 = g + iall + 1
                ijp1   = g + iall

                vt[g,d,TI] = ( (   2.0 * coef_intp[g,0,d,TI,l]
                                 - 1.0 * coef_intp[g,1,d,TI,l]
                                 - 1.0 * coef_intp[g,2,d,TI,l] ) * scl[ij    ,k,l]
                             + ( - 1.0 * coef_intp[g,0,d,TI,l]
                                 + 2.0 * coef_intp[g,1,d,TI,l]
                                 - 1.0 * coef_intp[g,2,d,TI,l] ) * scl[ip1j  ,k,l]
                             + ( - 1.0 * coef_intp[g,0,d,TI,l]
                                 - 1.0 * coef_intp[g,1,d,TI,l]
                                 + 2.0 * coef_intp[g,2,d,TI,l] ) * scl[ip1jp1,k,l]
                             ) / 3.0
```

図7: 図6に同じ。ただしPythonを用いて、Fortranを模倣して記述されたもの。

まず第一段階として、オリジナルのFortranコードをなるべく模倣した形で、Pythonへの移植を実施した。図6にオリジナルコードの一部を示す。この計算は隣り合う３つの六角形の各中心点（分割された正20面体格子系の格子点であり変数の配置された地点）から、３つの

六角形が共有する六角形の頂点の一つへ値を内挿する操作となっており、また1とkのループはそれぞれ、分割されたタイルと鉛直層毎に操作を繰り返すことを表している。これに対して、図7にforループを用いてPythonで実装した結果を示す。NumPyを用いることで、多次元配列をFortranで宣言した形状に揃えたこともあり、両者のソースコードは非常によく似ている。一方で、演算速度については両者は大きく異なる。表13に６つの演算カーネルの経過時間の結果をまとめる。計測にあたっては、AMD Ryzen5 5560Uを搭載したノートPCを用い、Windows11上のWSL2にて実行環境を構築して計算を行った。表13より、オリジナル版とPython版(Ver.1)では、実行時間に400-3,400倍もの違いが出た。

表13: IcoAtmosBenchmark v1の実行時間比較

カーネル名	説明	経過時間 (Fortran)[sec]	経過時間 (Python v1)[sec]
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	0.010	20.2
dyn_divdamp	3次元データに対して3次元方向に格子点を参照するステンシル計算	0.014	11.1
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	0.008	3.3
dyn_horiz_adv_flux	3次元データに対して水平2次元方向のフラックス項を計算	0.018	13.5
dyn_horiz_adv_limiter	3次元データに対して水平2次元方向のフラックス制限項を計算	0.037	22.8
dyn_vert_adv_limiter	3次元データに対して鉛直1次元方向のフラックス制限項を計算	0.003	10.3

次に、実行時間の短縮を測るために、NumPy多次元配列の要素毎の計算に用いられる「スライシング」という実装への変更を行った。図8にソースコードの例を示す。スライシングを用いる場合、右辺と左辺の配列の次元数を一致させる必要がある。また、前述のFortran版、Python(Ver.1)版ではループ内でのみ用いられる中間配列について、鉛直層とタイルの次元が無い小さな配列サイズを用いていたが、スライシングを用いる場合は最終的なターゲットとなる配列の次元・サイズに合わせる必要がある。これにより、演算の並列性を確保することが可能になるが、一方で一時的に消費するメモリサイズが増えるという問題がある。表14にスライシングで記述したPython(Ver.2)版の経過時間を示す。Ver.2でのコードはFortranに近いアルゴリズムの記述スタイルをやめることによって、dyn_diffusionでVer.1から58倍程度の高速化を実現した。さらに、Ver.2を元にJAXを用いた実装を行った。JAXでの実装においても、単純にforループで記述したPythonコードにJITコンパイルを指定しただけでは、ほとんど高速化は得られず、以下のようにスライシングを行う、

```
at[i:n].set(moderator)
```

という記述方法をとること（Ver.3）で、CPUではNumPyと同程度の速度を達成した（表14）。図8(下)に示したソースコードは、JAXで記述した際の例であり、NumPyのスライシング（図8(上)）と比較した場合、ほとんど同一のコードとなる。表14にはGPUを用いて計算した結果も示している。このとき、GPUにはNVIDIA A100を用いた。倍精度演算性能としては本実験でのCPU計算環境と比較して、50倍弱の違いがあるが、経過時間としてはあまりCPUと変わらない結果となった。これは、問題サイズがあまり大きくないことに起因する。演算性能の向上は今後の課題であるが、GPUの利用に対してプログラムを一切書き換える必要がない点は非常に高く評価できる。

```
Python Ver.2 (抜粋)

vt[gminm1-1:gmax,0:kall,0:nxyz,0] =
            ( (   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
```

```
Python Ver.3 (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
)
```

図8: 図6に同じ。ただしPythonを用いて、NumPyのスライシング手法を用いて演算を記述したもの（上）と、JAXを用いて演算を記述したもの（下）。

表14: IcoAtmosBenchmark v1の実行時間比較（その２）

カーネル名	経過時間 (Fortran)[sec]	経過時間 (Python v1)[sec]	経過時間 (v2)[sec]	経過時間 (v3,CPU)[sec]	経過時間 (v3,GPU)[sec]
dyn_diffusion	0.010	20.2	0.35	0.95	0.75

以上をふまえ、FortranからPython(JAX)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。

- JAX を用いた場合、ユーザープログラミングの視点からはほぼ GPU の有無を意識することなくプログラムを記述することが可能であり、また NumPy を用いた記述に習熟している場合は NumPy→JAX への移行は学習コストが極めて小さく済む。
- Fortran で記述した場合、do ループによる構造化によって配列の演算を行うが、Python(NumPy, JAX)で記述した場合にはスライシングによる配列の演算を採用することが必須である。これはアルゴリズムの記述方法としては大きな方針転換であり、機械的な書き換え作業だけでは完了できない。
- 演算性能については、問題サイズが小さい場合は Fortran よりも未だ低速であるが、適切なスケーリング性能評価を進めることによって、十分に許容可能な演算性能を得ることが期待される。

⑭ 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、ものづくり分野ソフトウェアADVENTUREに関するMN-Coreベンチマークコード群の開発を行った。

ADVENTUREは非構造格子を用いた有限要素解析と領域分割法を用いた並列計算が特徴であり、C言語、MPI、OpenMPで開発されている。ADVENTUREモジュールの1つであるADVENTURE Solidは、領域分割法に基づく構造解析コードであるが、そのなかで計算カーネルとなる部分領域

ソルバーの実装に関して、メニーコアおよびGPU・アクセラレータ向けに有効な実装となりうるLocal Schur Complement(LSC)アプローチに着目し、そのスカラー型計算機向けのミニアプリの実装を行った。本アプローチでは、部分領域に関する剛性行列をSkyline形式に代わり、LSC行列として保持することにより、各部分領域の求解過程において、反復ごとに行列分解を行っていた従来手法に比べ、LSC行列を用いた対称密行列ベクトル積演算のみが行われ、これにより大幅な計算時間の短縮を実現した。

また、ファイル入出力、有限要素解析部分をライブラリ化し、それらを呼び出し連立一次方程式の求解を行うPythonアプリの開発を行った。本アプリは2023年度にMN-Core実機評価済みコードを整備したものである。これにより、非構造格子から得られる疎行列を係数とする連立一次方程式を用いたMN-Core性能評価および最適化が可能となり、さらに汎用CPU向け既存コードを活用して低コストにMN-Core向けコードの開発が可能となった。さらに、領域分割法におけるサブドメインごとの疎行列データを入力とし、それらを読み込んで領域分割法の並列求解部分の性能評価を行うC+OpenMPアプリの開発を行った。これにより、ADVENTUREをそのままMN-Coreに移植する場合の性能評価が可能となった。

⑮ 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　国立大学法人東海国立大学機構 名古屋大学)

マテリアルサイエンス応用のアプリケーションとして、いわゆる第一原理計算、すなわち多体電子系の量子論に基づいた密度汎関数理論(DFT)に立脚した物質計算の重要性は確固たるものであり、次世代計算機アーキテクチャ上での、その高速計算は将来のマテリアルサイエンスでのひとつの大きなターゲットである。名古屋大学では、株式会社Quemixとの共同で、第一原理計算コードの内、2011年に ゴードン・ベル賞を受賞した実空間DFTコード (RSDFT: Real-Space DFT) の汎用CPUとアクセラレータでの性能を評価した。さらにRSDFTコードの発展形である、物質内イオンと電子のダイナミクスを明らかにするRS-CPMD (Real-Space Car-Parrinello Molecular Dynamics) コードの性能評価も行った。

具体的にはアプリケーションのカーネル部分を抽出し、現在のアーキテクチャ上でのボトルネックを明らかにする。それを踏まえ、今後の高速化および多機能化の指針を示す。またノード間通信、I/Oにおける問題点も調査する。さらにはハードウェア加速装置(例えばGPU)を導入した場合の性能向上の可能性を予測し、可能ならば具体的なベンチマーク計算により、その可能性を検証する。

1. コード調査 (カーネル部分特定、および現在のアーキテクチャでのボトルネック、ノード間通信、I/Oについて)
   「富岳」コンピュータ上でのRSDFTコードの実行により、その重要なカーネルは、部分空間反復法 (Rayleigh-Ritz法) による外部固有値問題の求解部分であり、特に問題サイズが大きい場合 (10万原子規模) は、Gram-Schmidt直交化および部分空間対角化がボトルネックとなり、いずれもターゲット物質のサイズNの3乗に比例する計算コストを要することが明らかになっている。ただしボトルネック部は行列積演算が主となるため、計算全体でも40～50%の高い実行効率を達成することが可能である (10,000ノード使用時) 。一方、小～中規模サイズ (1万原子以下) では差分行列および非局所擬ポテンシャルと呼ばれるランク1の行列の演算部分が、N^3のボトルネック部に比べ無視できない計算時間を要するようになり、それに伴って実行効率も下がる。RS-CPMDコードでの主要カーネルは、Gram-Schmidt直交化に対応するラグランジュ乗数部分の計算であり、システムサイズ3乗に比例する計算コストとなる。また一般的にMDは時間方向になるべく長くシミュレーシ

ョンを行う必要があり、１ステップに要する計算時間を抑えるために系のサイズを大きく取れない。そのために、高効率で実行可能なN^3のボトルネック部の計算と、低効率の差分および非局所ポテンシャルの行列演算の計算時間が同程度となり、全体的な実行効率も下がる傾向にある。

ノード間通信は、N^3ボトルネック演算部分で必要となるリダクション（MPI_Allreduce）と、差分および非局所擬ポテンシャル行列演算に伴う隣接通信（MPI_Isend, MPI_Irecv）が主となる。並列化は空間および固有ベクトルの軸について行うことが可能で、大規模系では両者の軸についての並列化を組み合わせた場合、計算時間全体の30%ほどが通信時間となる。小〜中規模系では隣接通信のコストが主となるが、演算部分と合わせ、この部分のコスト削減が次世代機で実現できれば大幅な性能向上が見込めると思われる。I/Oについては、計算の最後に、大きくても使用メモリサイズ相当のデータを書き出すのみで、現在のターゲットサイズ、10,000 - 100,000原子規模計算では、それほどの重大な問題は引き起こさないと思われる。

2. GPUを導入した場合のベンチマークおよび性能向上予測と「富岳」との比較
GPUに対応したRSDFTコードを用いて、グリッド数4,374,000点、固有ベクトル本数2,592本（1,000原子規模）の系において「富岳」との比較を行った。「富岳」では108ノード（12MPI/ノードで実行）を用いた場合、反復計算１ステップの実行時間は15.7秒であった。一方、NVIDIA DGX A100を用いてGPUを8台搭載したノード1つで同じ計算を実行したところ、実行時間は16.8秒であった。主要カーネルごとの計算時間の内訳を表15に示す。

表15:「富岳」およびDGX A100（8 GPU）でのSCF反復計算1ステップの実行時間比較。単位は秒。「富岳」は108ノード、DGX A100は1ノードを使用。SDは部分対角化、CGは共役勾配法、GSがGram-Schmidt直交化で、SCF1ステップの中で実行される各主要カーネルでの計算時間である。

	SCF 1 step	SD ($O(N^3)$)	CG ($O(N^2)$)	GS ($O(N^3)$)	Others
「富岳」	15.7	4.50	7.33	0.97	2.90
DGX A100	18.6	4.25	9.56	3.07	1.72

RSDFTには元々、差分および非局所擬ポテンシャルの通信を束ねて通信回数を削減する機能が実装されており、この機能がほぼそのままGPUオフロードの効率化に利用できた。その結果、「富岳」であまり効率の出ていない部分が加速され、「富岳」100ノード相当の計算時間がGPU機１ノードで達成できるという結果が得られた。問題サイズ依存性や、プロセス並列との組み合わせなど、より効率化するための検討が今後の課題となる。

⑯ 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。

将来のMN-Core様アクセラレータの予測性能に基づいてカーネル部分の机上性能推定値の見積もりを行った。１アクセラレータ当たりの格子サイズを$16^3 \times 32$とし

てクォーク伝搬関数計算プログラムの性能を机上見積もりした。倍精度計算の場合に現在の世代のMN-CoreのPEのローカルメモリに全てのデータが乗ることを確認し計算に必要なデータはローカルメモリとレジスタから供給できることを確認した。このときのPE当たりの格子サイズは$2^3 \times 2$と見積もった。反復法を2,000回反復したときの実行時間は約1.6秒と見積もった。データが全てメモリーにある場合の見積もり時間約4.1秒に比べて約2.5倍の高速化が期待できた。

単精度と倍精度を組み合わせた混合精度での反復法の加速を用いる場合はアクセラレータでは単精度の計算を主に行う。倍精度の机上計算と同様の見積もりを単精度の場合に行った。データサイズは単精度により半分となるので全てのデータをローカルメモリ搭載でき問題とならないがベクトル演算に使えるベクトル長（並列度）が稼げないので演算性能の向上は限定的であると見積もった。反復法を2,000回反復したときの実行時間は上述の倍精度の約1.6秒と同程度か少し高速化すると見込まれた。

⑰ 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）

宇宙の構造形成シミュレーションや惑星形成シミュレーションで使われる粒子間相互作用計算を行うライブラリ「Framework for Developing Particle Simulators（FDPS）」のMN-Coreでの性能評価を行った。

宇宙・惑星科学で天体間に働く重力を計算するカーネル（重力相互作用カーネル）を「⑦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」で開発したコンパイラでコンパイルし、アセンブラを出力した。その結果、1相互作用計算が22ステップとなり、これを重力相互作用計算の演算数30演算分と数えると、22ステップでの理論ピーク性能44演算に対して、30/44 = 0.68、つまり、重力相互作用カーネルについては理論ピークの68%で計算できることがわかった。行列乗算に対して、普通の演算の性能が1/4程度であるので、それを考慮すると、理論ピークの17%程度で実行できると予想される。重力相互作用計算では、通常、混合演算を行うため、今後、単精度を混ぜることによって、理論ピーク性能の25%くらいまで改善を図る。

⑱ 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及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。具体的には、与えられたユーザーコードをMN-Coreで実行可能な形式に変換するコンパイラの開発にあたって、アニーリングを併用した最適化アルゴリズムを導入することで、命令列の順序を最適化する手法の改良のための研究開発を進めた。命令列の順序を最適化することで、レジスタ間の無駄なデータ移動や命令間の依存関係による待機命令の挿入が抑制されることによって、計算の高速化が期待される。開発するコンパイラは、ユーザーコードを解析して疑似命令列を生成する部分（以下Aという）と、アニーリングにより疑似命令列順序を最適化する部分（以下Bという）に分かれる。

本年度の手法の改良点は、次の通りである。まず、Aの改良として、ユーザーコードを疑似命令に変換する際に生ずる余分なデータ移動命令を検出する部分のプログラムの改良を行

った。従来手法では、余分なデータ移動の検出のためにコードを何度もエミュレートする手法を取っていたため、Aの実行に時間がかかっていた。一方、改良手法では、動的計画法に基づく計算により、コードの走査を大幅に削減できる手法を開発した。これにより、Aの実行は、A全体で約70%程度高速化された（ユーザーコードによって計算時間が大きく異なるため、複数の異なるユーザーコードをコンパイルしたときの平均的な値）。

またBの改良として、条件分岐命令への対応を行った。MN-Coreでは、条件分岐命令を、命令実行のマスクによって実現している。そのため、分岐実行のためには余分なコストがかからないものの、マスクの対象を計算する論理計算を実行する必要がある。本改良では、マスクの対象を計算する論理計算を含めた命令列の最適化処理を実装した。改良にあたっては、マスク処理に対応するため、アニーリング割付コードインターフェースの修正も行った。その結果、LJ多体粒子系の計算においては、計算処理に実質的に関与しないデータ移動命令および命令間の依存関係による待機命令の割合を、最適化の前後で30%から10%程度に削減することに成功した。

⑲ 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）

②MODEST-23 JM

4 月 17-19 日　沖縄科学技術大学院大学(OIST)シーサイドハウス

③シミュレーション天文学のこれまでとこれからーハードウェア・アプリケーション・サイエンスー

9 月 4-6 日　神戸大学 統合研究拠点 コンベンションホール（ハイブリッド）

④CPS セミナー「OpenACC for MN-Core」

10 月 10 日 神戸大学 惑星科学研究センター セミナー室（ハイブリッド）

⑤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中間報告～

12 月 19 日　東京大学　浅野キャンパス　武田先端知ビル 5F　武田ホール

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	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	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	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　最高研究責任者 岡野原 大輔
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ネットワークアーキテクチャ評価	東京都立川市緑町１０番３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
⑦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
⑧DSL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
⑨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　学校法人順天堂大学	健康データサイエンス学部 特任教授　姫野 龍太郎

⑩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　最高研究責任者　岡野原 大輔
⑪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
⑫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
⑬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
⑭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
⑮マテリアルサイエンス応用アプリケーションの検討	愛知県名古屋市千種区不老町１番　国立大学法人東海国立大学機構 名古屋大学	未来材料・システム研究所　特任教授　押山 淳
⑯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
⑰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
⑱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
⑲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	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
Accelerating 128-bit Floating-Point Matrix Multiplication on FPGAs（ポスター発表）	河野 郁也 中里 直人 中田 真秀	The 31st IEEE International Symposium On Field-Programmable Custom Computing Machines	2023年5月9日	国外
Evaluation of Various Arithmeic for Linear Algebra on GPU and FPGA（口頭発表）	中里 直人 村上 誉 河野 郁也 中田 真秀	ICIAM 2023	2023年8月21日	国内
シミュレーション天文学、専用計算機、汎用スパコン、深層学習（口頭発表）	牧野 淳一郎	研究会「シミュレーション天文学のこれまでとこれから -ハードウェア・アプリケーション・サイエンス」	2023年9月4日	国内
オリジナルアクセラレータアーキテクチャによる超低消費電力次世代計算基盤の評価（口頭発表）	牧野 淳一郎	第17回アクセラレーション技術発表討論会	2023年9月28日	国内
128-bit 浮動小数点演算による行列積とアプリケーションの性能評価（口頭発表）	中里 直人 河野 郁也 中田 真秀	第17回アクセラレーション技術発表討論会	2023年9月28日	国内
OpenACC for MN-Core（口頭発表）	綱島 隆太	CPS セミナー	2023 年 10 月 10 日	国内
Approximate Interconnec-tion Networks forParallel Processing（口頭発表）	鯉渕 道紘	The 13th International Symposium on Photonics and Electronics Conver-gence(ISPEC 2023)	2023年11月2日	国内
Photonic Approximate Communication Highlight-ing Ultimate Nature of Light（口頭発表）	鯉渕 道紘	The 11th International Workshop on Computer Systems and Architect-ures, in conjunction with CANDAR 2023	2023年12月1日	国内
研究報告IIシステム研究調査の概要と検討状況（口頭発表）	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	2023年12月19日	国内

研究報告IIシステム研究調査の概要と検討状況 （口頭発表）	牧野 淳一郎	「次世代計算基盤を利用した成果の最大化に向けて」に関する意見交換会	2024年1月15日	国内
Evaluation of POSIT Arithmeic with Accelerators （口頭発表）	中里 直人 村上 誉 河野 郁也 中田 真秀	HPC Asia 2024	2024年1月27日	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
なし				