参考資料2

White Paper on Next-Generation Advanced Computing Infrastructure
（次世代先端的計算基盤に関する白書）

第 1.0.0 版

2020 年 11 月

目次

詳細版

執筆者

全体取りまとめ　近藤正章（東京大学／理化学研究所）

アーキテクチャ WG

WG **リーダ**　三輪忍（電気通信大学），佐野健太郎（理化学研究所），谷本輝夫（九州大学）

WG **メンバー**　安島雄一郎（富士通），井口寧（北陸先端大学），井上弘士（九州大学），江川隆輔（東京電機大学），岡本有司（Spin Memory），小野貴継（九州大学），鯉渕道紘（国立情報学研究所），児玉 祐悦（理化学研究所），小林諒平（筑波大学），小松一彦（東北大学），佐藤雅之（東北大学），塩見準（京都大学），田邊昇（東京大学），中里直人（会津大学），吉川隆英（富士通研究所），福本尚人（富士通研究所），星宗王（NEC），三好健文（わさらぼ），宮島敬明（理化学研究所）

システムソフトウェア WG

WG **リーダ**　佐藤賢斗（理化学研究所），佐藤幸紀（豊橋技術科学大学）

WG **メンバー**　合田憲人（国立情報学研究所），遠藤敏夫（東京工業大学），小柴篤史（理化学研究所），小松一彦（東北大学），坂本龍一（東京大学），高野了成（産業技術総合研究所），滝沢寛之（東北大学），辻美和子（理化学研究所），ゲローフイ バリ（理化学研究所），中島耕太（富士通研究所），深井貴明（理化学研究所），山本啓二（理化学研究所），和田康孝（明星大学）

アプリケーション・ライブラリ・アルゴリズム WG

WG **リーダ**　深沢圭一郎（京都大学），今村俊幸（理化学研究所），中島研吾（東京大学／理化学研究所）

WG **メンバー**　岩下武史（北海道大学），小野謙二（九州大学），笠置明彦（富士通研究所），片桐孝洋（名古屋大学），白幡晃一（富士通研究所），住元真司（富士通研究所），高橋大介（筑波大学），寺尾剛史（理化学研究所），長坂侑亮（富士通研究所），椋木大地（理化学研究所），村上弘（東京都立大学）

システム運用 WG

WG **リーダ**　塙敏博（東京大学），野村哲弘（東京工業大学）

WG **メンバー**　大島聡史（名古屋大学），實本英之（理化学研究所），庄司文由（理化学研究所），滝澤真一朗（産業技術総合研究所），竹房あつ子（国立情報学研究所），藤原一毅 （国立情報学研究所），三浦信一（理化学研究所）

第 1 章

はじめに

シミュレーションやビッグデータ解析といった計算によるサイエンス／エンジニアリングの推進や人工知能の高度化など，高性能な計算環境の必要性は高まる一方である．これまで，我が国では 2011 年に世界初の 10 ペタフロップスを達成した理化学研究所の京コンピュータや，2020 年に計算速度（Top500），計算性能（HPCG），ビッグデータ解析（Graph500），AI 性能（HPL-AI）の 4 部門で世界 1 位となった同じく理化学研究所のスーパーコンピュータ「富岳」，そして各大学の基盤センターにおけるスーパーコンピュータなど，多くの高性能計算基盤を開発・運用してきた実績を持つ．今後も，計算科学によるさらなる成果の創出や，AI とビッグデータ技術のさらなる融合，Society5.0 といった新しい応用分野への展開などへ向けて，継続的な最先端高性能計算基盤の研究開発が求められている．

一方で，今後の高性能計算機の構築・運用を考えるにあたり，ムーアの法則の終焉や従来とは異なる利用形態への進化など，多くの技術的課題が待ち受けていることも事実である．そこで，将来の高性能計算環境，あるいは共用計算機資源の開発・運用においてどのような技術的課題があり，どのような研究開発が必要なのか，コミュニティーとしてどのような活動をしていくべきなのかなどに関してオープンに意見交換をする場として Next-Generation Advanced Computing Infrastructure: NGACI (次世代先端的計算基盤) が設立された．NGACI では主としてデバイス・アーキテクチャ，システムソフトウェア，数値計算ライブラリ・アルゴリズム，システム運用の 4 つのワーキンググループに分かれて活発な議論が行うとともに，2019 年から 2020 年 8 月にかけて計 7 回の全体ミーティングを開催して意見交換を行ってきた．現時点での参加者は 98 名にのぼる．本白書は，高性能計算基盤に関するこれまでの技術動向も踏まえて，NGACI での議論をまとめたものであり，主に 2028 年頃を想定した高性能計算機システム構成を外観するとともに，それに向けて必要となるアーキテクチャ，システムソフトウェア，数値計算ライブラリ・アルゴリズム，運用技術の課題と研究開発ロードマップについて述べている．

本白書の構成は以下の通りである．第 2 章でスーパーコンピュータの技術動向についてまとめる．第 3 章でアプリケーションの要求性能分析についてまとめる．第 4 章で次世代（2028 年頃）システムの候補についてまとめる．第 5 章で次世代型運用への要求についてまとめる．第 6 章で技術課題と研究開発ロードマップについてまとめる．第 7 章でまとめを述べる．

第 2 章

スーパーコンピュータの技術動向

2.1 ハードウェア技術の動向

2.1.1 デバイス（回路構成素子）

本節では 2020 年現在のデバイス技術と今後の動向を説明する．まず現在の主流である CMOS デバイスの現状とロードマップについて述べ，次に近い将来に有望とされる各種新デバイスの開発状況をまとめる．さらに今後 10 年間で最も重要になる先進パッケージング技術についても説明する．

2.1.1.1 CMOS デバイス

2010 年代では中頃に FinFET の導入とマルチパターニングの導入が相前後して起こった．2020 年代では前半に EUV 露光装置が導入され，後半までに GAA[*1] FET が導入されると予想される．2028 年頃の高性能計算機システムでは EUV 露光により FinFET の集積度が上がり，GAA FET を使用できる可能性もある．集積度は現在の 5-6 倍にあたる 500MTr/mm^2 程度に達すると予想される．

現在，最先端の CMOS ロジックデバイスでは FinFET と呼ばれるトランジスタが使用されている．FinFET ではトランジスタはフィンと呼ばれる垂直に立った薄い板状のシリコンに形成され，ゲート電極がフィンの側面と上面の合計 3 面に接触するように形成される．FinFET 以前に主流であったプレーナ型 MOSFET に比べ，FinFET ではリーク電流の問題が大幅に改善されている．今後 FinFET はトランジスタの下面にもゲート電極を形成する GAA FET に置き換えられると考えられている．GAA FET では，ナノシートと呼ばれる横に倒した薄い板状のシリコンにトランジスタを形成してゲート電極で包む構造が主流になると考えられている．単にフィンを倒すだけでは集積度が下がるが，ナノシートを垂直方向に積層することで FinFET よりも高い集積密度を実現すると期待されている．CMOS デバイスでは GAA FET で集積度の限界に達すると考えられており，さらに集積度を上げるには CMOS 以外のデバイスが必要と考えられている．新デバイスの開発状況については次節で述べる．

最先端プロセスの露光装置では，2000 年代後半の 45nm 世代で本格的に導入された光源波長 193nm の ArF 液浸露光が現在でも主流である．導入当初の最小配線ピッチは 160nm 程度であったが，現在では 1 つの回路パターンを複数のマスクで露光するマルチパターニングの採用により，最小配線ピッチは 40nm 程度まで縮小している．マルチパターニングによる製造はコストが高いので，2020 年代には露光装置が光源波長 13.5nm の EUV 露光に徐々に切り替わり，集積度はさらに向上すると見込まれている．

[*1] Gate-All-Around

マルチパターニングや EUV 露光による製造コストは高く，ダブルパターニングの導入された 20〜16nm 世代以降の先端半導体プロセスではトランジスタ単価が下がらない問題が起きている．すなわち，最先端の半導体プロセスを使用するハイエンド製品では，半導体プロセスの世代が進んで集積度が向上するに従いチップ単価が常に上昇してしまう．22nm 世代以前では集積度の向上に伴ってトランジスタ単価が低下していたため，多くの機能を最先端プロセスのチップに集積することでコストパフォーマンスが向上した．しかしトランジスタ単価が下がらない場合には，大きなチップは歩留りが悪くなりむしろコストパフォーマンス低下の要因になる．このトランジスタ単価問題への対処として，チップレットが注目されている．チップレットとは小さいチップのことであり，従来大きいチップで実現されていた機能を複数のチップレットにより実現することを狙う．チップレットはダイの状態で KGD[*2]を選別してパッケージに搭載することで，大きい単一のチップに比べて高い歩留りを実現する．ただしチップレット間を Die-to-Die インタフェースで接続する必要があるので性能，チップ面積の両面においてオーバヘッドはあるが，そのことを差し引いてもコストパフォーマンスの面で有利であるとされる．

以上の技術動向を踏まえて，最先端の CMOS ロジックデバイスを製造している半導体製造会社 3 社のロードマップを以下にまとめる．なお，プロセスの世代は便宜上各社が用いている nm 単位のテクノロジーノードで表記する．22nm 世代まではテクノロジーノードの値が 70% 小さくなる毎に集積度はおよそ 2 倍になることを示しており，集積度の向上が 2 倍未満の微細化の場合にはハーフノードと呼ばれる中間の値が使用されていた．また，半導体製造会社が異なっていてもテクノロジーノードの数値が同じであれば，ほぼ同等の集積度を表していた．しかし 16nm 世代以降では世代の更新間隔が伸びるとともに各種用途向けにプロセスが多様化したため，各社とも独自のテクノロジーノードの値を使用している．半導体製造会社が異なるとテクノロジーノードの数値が同じでも集積度は大きく異なっている点に注意が必要である．また，各プロセスの集積度の値には 2017 年に Intel の Mark Bohr が提唱した MTr/mm^2 を使用する [1]．FinFET ではトランジスタの周辺回路を含んだセルライブラリの構成が実際の集積度に大きく影響するため，MTr/mm^2 は面積の 60% が NAND2 セル，40% が Scan Flip Flop セルであった場合の単位面積当たりのトランジスタ数で集積度を表す．

Intel の最新プロセスは 14nm 世代の P1272 および 10nm 世代の P1274 である．P1272 はモバイル製品が 2014 年，ハイエンドサーバ製品が 2016 年に出荷開始された．P1274 はモバイル製品が 2019 年に出荷開始された．ハイエンドサーバ製品は 2021 年に出荷開始される予定である．Intel の次世代プロセスは 7nm 世代の P1276 である．P1276 は EUV 露光を使用し，グラフィックス用の製品が 2022 年から 2023 年頃に出荷開始される予定である．P1272，P1274 の集積度はそれぞれ $44.67MTr/mm^2$，$100.76MTr/mm^2$ である．P1276 は 10nm 世代に対し 2.4 倍の集積度を目指しているので最大 $250MTr/mm^2$ 程度と予想される．Intel は世代ごとに 2 倍の集積度を目指す方針であり，5nm 世代の P1278 では 400-500MTr/mm^2 に達すると予想される．5nm 世代は非常に高い集積度が目標になるため，FinFET から GAA FET へ移行する可能性もある．Intel のプロセスは 22nm 世代の P1270 までは 2 年おきに世代を更新していたが，14nm 世代への更新では 3 年を要した．現在移行中の 10nm への更新間隔はおよそ 5 年となる見込みである．次世代の 7nm への更新間隔は今のところ 3 年程度の見込みである．2028 年に稼働するシステムでハイエンドサーバ製品を採用することを考えると，7nm 世代，5nm 世代への移行が 14nm 世代程度にスムーズであれば，5nm 世代が想定される．一方，どちらかの世代への移行に 10nm 世代と同程度の期間がかかると，7nm 世代に留まっていると想定される．

TSMC の最新プロセスは 7nm 世代の N7 である．N7 はモバイル製品 (Apple) が 2018 年，ハイエンドサーバ製品 (AMD) が 2019 年に出荷開始された．N7 の集積度は $91.2MTr/mm^2$ である．次世代プロセスは 5nm

[*2] Known Good Die

世代の N5 である．N5 では EUV 露光を使用したモバイル製品が 2020 年に出荷開始される予定である．N5 の集積度は 7nm 世代の 1.8 倍を目標とすることから最大 170MTr/mm^2 程度と予想される．TSMC は 2019 年に 3nm 世代の N3 プロセスの半導体工場を建設を開始しており，2023 年に大量生産を開始するとしている．N3 の集積度は不明だが，N5 と同様の 1.8 倍向上を仮定すると 300MTr/mm^2 程度になると予想される．TSMC は集積度の向上が 2 倍に満たなくても 2，3 年でプロセス世代を更新しており，2028 年頃のハイエンドサーバ製品を考えると，N3 の次の 2nm 世代が想定される．N3 からさらに集積度が 1.8 倍向上すると 500MTr/mm^2 程度となる．また，時期と集積度を考慮すると，FinFET から GAA FET へ移行している可能性がある．

Samsung の最新プロセスは EUV 露光を使用する 7nm 世代の 7LPP である．既に 7LPP によるモバイル製品が 2019 年に，グラフィックス製品 (NVIDIA) が 2020 年に出荷が開始された．ハイエンドサーバ製品 (IBM) も 2021 年に出荷が開始される予定である．7LPP の集積度は 95.08MTr/mm^2 である．次世代プロセスは 5nm 世代の 5LPE である．5LPE はモバイル製品が 2020 年に出荷が開始される予定である．Samsung はさらに GAA FET を採用する 3nm 世代の 3GAE プロセスを準備しており，2021 年に量産を開始する予定である．5LPE は 7LPP に比べ面積を 20% 以上縮小し，3GAE は 7LPP に比べ面積を 40% 縮小する目標であるので，集積度は 5LPE が 120-130MTr/mm^2 程度，3GAE が 160MTr/mm^2 程度と予想される．当面は FinFET の 5LPE プロセスと GAA FET の 3GAE プロセスを並行して更新する予定になっている．2028 年頃のハイエンドサーバ製品では，トランジスタが GAA FET に一本化されているか，あるいは FinFET も並行して提供されているかは不明であるが，集積度は Intel，TSMC の計画と同程度の 500MTr/mm^2 付近を目標にすると予測される．

また，International Roadmap for Devices and Systems の More Moore 分冊 (以下 IRDS MM) に 2020 年から 2034 年の CMOS 技術のロードマップがあるので紹介する．IRDS MM ではピッチスケーリングに関するロードマップが以下のように示されている．

表 2.1　IRDS MM のピッチスケーリング・ロードマップ

Year of Prodocution	2020	2022	2025	2028	2031	2034
Contacted gate pich (nm)	48	45	42	40	38	38
Tightest metal pich (nm)	36	24	20	16	16	16
Number of tiers	1	1	1	1	2	4
Logic industry ”Node Range” Labeling (nm)	”5”	”3”	”2.1”	”1.5”	”1.0 eq”	”0.7 eq”
IDM-Foundry node labeling	i7-f5	i5-f3	i3-f2.1	i2.1-f1.5	i1.5e-f1.0e	i1.0e-f0.7e
Logic device structure options	FinFET	FinFET GAA	GAA	GAA	GAA-3D	GAA-3D
Mainstream device for logic	FinFET	FinFET	GAA	GAA	GAA-3D	GAA-3D

ここで 2028 年に稼働するマシンで利用できる CMOS デバイスは 2025 年の 2.1nm 世代であるので，2020 年の 5nm 世代からのピッチスケーリングではおよそ 2.06 倍の密度が実現されることになる．TSMC の 5nm 世代が最大 170MTr/mm^2 程度であることを考えると 2.1nm 世代は最大 350MTr/mm^2 程度のトランジスタ密度と見積られ，各社のロードマップに比べると過少見積りである．これは本節でも説明し，IRDS MM でも繰り返し述べられているように，2020 年代においてはピッチスケーリングだけではなくセルライブラリの構

成がトランジスタ密度に大きく影響することが原因である．

2.1.1.2 新デバイス

CMOS デバイスの微細化がコスト的にも技術的にも限界に近付くなかで，集積度以外の様々な観点から CMOS デバイスを超える性能を持つ新デバイスは Beyond CMOS と呼ばれ期待されている．2028 年頃の計算機システムに採用可能なほど新デバイスの製造技術が成熟する可能性は小さいが，2028 年頃のシステムには，その後に来る Beyond CMOS 時代のコンピューティングを見据えた設計が求められる．本節では有望な Beyond CMOS デバイスの例として，超電導デバイス，ナノフォトニック・デバイスの研究開発状況をまとめる．この他にも化合物半導体，CNT トンネル FET，ナノグラフェン，分子デバイス，スピンデバイスなどの新デバイスが Beyond CMOS として期待されている．

■超伝導デバイス　ジョセフソン接合は 単一磁束量子（SFQ：Single Flux Quantum） 回路の基本デバイスであり，2 つの超伝導体の間に薄い障壁層をはさんだ構造を有する．ジョセフソン接合を流れる電流が臨界値を下回ると，ジョセフソン接合間には電圧が発生せず超伝導体間を超伝導電流が流れる．一方，電流が臨界値を上回ると，ジョセフソン接合間に電位差が発生する．このようなジョセフソン接合を用いて超伝導ループを構成し，たとえば，超伝導ループに周回電流が流れる状態（超伝導ループ中に磁束が生じる状態）を論理値 1 とし，流れない状態（磁束が消失した状態）を論理値 0 とすれば，この超伝導ループの組合せにより論理ゲートを構成できる．超伝導ループ中に磁束が出入りする時に発生する微弱電圧パルスを情報担体とするため，論理ゲート当りの消費電力は，電荷の充放電を基本とする MOSFET 回路と比べて約千分の 1 以下である．また，無損失な電磁波信号伝搬による超高速性をも合わせ持つ．このようなデバイスの特性を活用し，数十ギガヘルツ程度で動作可能な極低温環境でのコンピューティングを実現する試みが注目を集めている．例えば，近年では米国 IARPA による国家プロジェクト C3（Cryogenic Computing Complex Program）が IBM などを中心として実施された．また，Microsoft Research，カリフォルニア大学サンタバーバラ校，ローレンスバークレー国立研究所なども研究開発に関連する発表を行っており，米国 MIT や中国 CAS では巨額の投資により最先端の 0.25μm 製造プロセスも投入された．日本においては，産総研 1.0μm プロセスを利用可能であり，様々なチップ試作結果も報告されている．重要なポイントは新奇デバイスを活用したコンピューティングモデルの再構築やマイクロアーキテクチャの再検討を行っている点であり，MOSFET を前提とした従来のコンピューティング技術とは異なる最適化が試みられている．例えば，信号伝搬速度の差を利用して情報を表現するレースロジックを前提とした取り組み [2] や，論理ゲート 1 段をパイプラインステージと見なすゲートレベル・パイプライン処理の導入 [3] など，アーキテクチャレベルにおいて様々な新奇デバイス活用法の模索が進められている．この SFQ 回路は 4 ケルビン程度の温度環境化で動作する．近年注目を集めている超伝導量子デバイスはミリケルビン環境が必要であり，ヘリウム希釈冷凍機内にはミリケルビン階層と 4 ケルビン階層が存在する場合が一般的である．このような極低温環境を前提とし，将来的には「量子デバイスによる超並列情報処理と SFQ 回路による制御・汎用処理の融合」といった方向性も期待される [4].

■ナノフォトニック・デバイス　フォトニック結晶は，屈折率の異なる人工結晶を数百 nm の周期で配置するナノ構造体である．フォトニック結晶が配置された領域では，結晶構造で決定される波長の光伝搬が禁止されるフォトニックバンドギャップが発生する．フォトニック結晶でシリコンなどの媒体を挟み込むことで，光を波長以下のスケールで強く閉じ込めることができる．良好な非線形効果や屈曲耐性などの理由から，長距離データ通信用途として，古くからフォトニック結晶ファイバの研究がなされてきたが，近年の半導体製造技術の進化を背景に，シリコンチップ上にフォトニック結晶を製造し，波長スケールで光の伝搬を制御する光集積

回路技術の研究が活発化している．我が国では，NTT が 2 次元ナノフォトニック結晶，京都大学が 3 次元ナノフォトニック結晶の実現に成功しているが，量産が可能な段階には至っていない．光素子の動作の遅延は，主に素子長で決定されるため，負荷容量の充放電により動作の遅延が律速される CMOS 集積回路よりも低遅延で動作することが期待されている．また，光回路は電気スイッチングを伴わず，かつ超高速に動作するため，省エネルギー性も期待されている．例えばシリコンフォトニクスを例に述べると，VLSI シンポジウム 2020 にて，15ns 遅延かつ 1.5W 電力の性能で動作する超高速・超低電力光スイッチが IBM により実現されていることが報告された．ナノフォトニック・デバイスはさらに小型の実装が可能で，素子長が数十 μm〜数百 μm で，従来の CMOS 論理ゲートと比較して，10 倍〜100 倍以上低遅延な動作を実現するナノフォトニック光変調器が報告されている．これをうけて，オンチップ通信を高速化する Optical network-on-chip (ONoC) の研究開発が活発化するのと同時に，チップ上でコヒーレント光を干渉・変調させて演算する光コンピューティングの研究が再び流行している．特に，光の干渉・変調に基づく光ニューラルネットワーク・アクセラレータの研究がマサチューセッツ工科大学，プリンストン大学，テキサス大学オースティン校などを中心として活発化している．例えばマッハ・ツェンダ干渉計をアレイ状にならべて，光の干渉だけで推論を行うナノフォトニック・ニューラルネットワークアクセラレータが [5] で発表されており，当該ニューラルネットワークアクセラレータは従来の電気電子方式と比較して 2 桁以上も高速に動作する．またマイクロリング共振器に基づく波長分割多重型のベクトル行列積 (VMM) を活用した再帰ニューラルネットワーク (RNN) アクセラレータの実現が [6] により実現されている．[6] で設計された RNN の面積は数百ミクロン四方，かつノード数 10 程度と小規模であるが，演算回路を大規模化することも原理上可能である．CMOS 集積回路では実現できない低遅延性を活用して，エッジやフォグ環境などでデータ処理を超低遅延に実現するハードウェアアクセラレータとして活躍することが将来のキラーアプリケーションとして期待される [7]．一方で，ニューラルネットワークの実現に必要な大規模メモリの全光実装は現状では困難であり，例えば大規模光ニューラルネットワークアクセラレータを設計する際には，スケーラビリティの点に問題がある．ナノフォトニック・デバイスの低遅延性と CMOS デバイスの高い実装密度という両者の利点を活かすため，CMOS 集積回路とナノフォトニックアクセラレータの協調設計という方向性も期待される．例えば [8] では，CMOS 回路とナノフォトニック VMM の協調設計により，不揮発性メモリベースの CNN アクセラレータよりもエネルギー効率 (スループット/電力) が 13 倍以上良い CNN アクセラレータを実現できることが述べられている．

2.1.1.3 先進パッケージング技術

■ダイ間接続技術　半導体プロセスの微細化ペースが減速し，さらに最先端プロセスによるトランジスタ単価が下がらなくなっていることにより，高性能，低コストを実現するために複数のダイ，異なるプロセスのダイのパッケージへの統合が重要となっている．複数のチップを統合したパッケージを MCM[*3]もしくは MCP[*4] と呼び，同一パッケージに異なる半導体プロセスのダイを統合することを HI[*5]と呼ぶ．現在の HI では 1 つのパッケージに大型のダイ 1 つ分のプロセッサと 4 スタック程度の HBM2 を統合しているが，2020 年代の前半には大型のダイ 2 つ分に相当する面積のプロセッサと，8 スタック程度の HBM2 を統合する見通しである．先進パッケージング技術には今後大規模な投資が行われる見通しであり，2028 年に稼働するシステムではさらに密度の高い統合パッケージを想定できる．

パッケージ基板の設計ルールは配線幅 10μm 程度とプリント配線板の配線幅 100μm 程度に対して 1 桁微細

[*3] Multi-Chip Module
[*4] Multi-Chip Package
[*5] Heterogeneous Integration

であり，MCP 上のダイ間はプリント基板上のパッケージ間よりも高い密度で配線できる．現在の半導体製造プロセスでのダイサイズは回路パターンのマスクの大きさが制限となって最大 26mm × 33mm である．これに対してパッケージ基板は 7〜10cm 角程度の大きさが製造可能で，複数のチップを搭載することができる．なお，プリント配線板の一般的な最大サイズは 50cm 角程度である．

近年では半導体製造プロセスの配線工程で配線層を形成した Si-IP(シリコン・インターポーザ) の上に複数のダイを搭載することで，LSI のグローバル配線と同等の配線幅 1μm 程度にまで密度を高めたダイ間接続も実現されている．TSMC の CoWoS[*6]では Si-IP を上向き，チップを下向きに向かい合わせてマイクロバンプで接続し，Si-IP は TSV[*7]を通じてパッケージ基板と接続する．2012 年の第一世代 CoWoS では Si-IP のサイズがマスクサイズに制約され，最大サイズはダイサイズと同じ程度であった．2015 年の第二世代 CoWoS ではマスク 2 枚を使用し Si-IP の最大サイズをマスクの 1.5 倍に拡大し，最大サイズ近い大きさのチップと HBM2 メモリを統合できるようになった．2017 年には最大サイズはマスクの 1.75 倍に拡大された．N5 プロセスの世代では CoWoS の Si-IP サイズは最大でマスクの 2 倍に拡大される予定である．TSMC はさらに将来に向けて 2500mm^2 すなわちマスクサイズの 3 倍の Si-IP を試作している．この Si-IP は大型のダイ 2 つと HBM2 を 8 スタック搭載することが可能であり，試作製造にはマスクを 4 枚使用していると推測される．

Intel は Si-IP と同等の配線密度を実現する EMIB[*8]を開発し，2017 年に FPGA 製品を出荷開始した．配線層を形成したシリコン片が上向きに配置される点は Si-IP と同じであるが，シリコン片はダイ間接続配線に必要最小限の大きさであり，パッケージ基板に埋め込まれる点が異なる．Si-IP ではパッケージ基板との接続に TSV が必要となったが，EMIB ではダイ間接続以外はチップとパッケージ基板が直接接続するので TSV が不要であり，また Si-IP のように大きいサイズは不要なので，マスクサイズが制約にならない利点がある．また，Intel は上下ともトランジスタを形成したチップを向かい合わせに接続する Foveros を開発し，2020 年にモバイル向け SoC を出荷開始した．下側になるチップはアクティブインターポーザと呼ばれ，TSV を形成され，上向きに配置される．さらに EMIB と Foveros を併用するパッケージを Co-EMIB と呼び，2022 年から 2023 年頃にグラフィックス製品の出荷を開始する予定である．このグラフィックス製品の仕様は公開されていないが，構想図では 8 つのコンピュートチップを搭載したアクティブインターポーザが 2 つと，合計 8 から 10 スタックの HBM2 がパッケージに統合されている．

この他に主にモバイル向けプロセッサでは FOWLP[*9]と呼ばれるパッケージ技術が使用されている．WLP[*10]ではパッケージ基板を使用せず，PCB 基板へのコンタクトをチップに直接形成する．FOWLP は入出力ピン数を確保するため，ダイサイズより大きな再配線層を形成する．ダイからはみ出した再配線層はモールドで支持され，再配線層は半導体製造プロセスの配線工程で形成される．FOWLP の最大サイズはマスクサイズに制約される．FOWLP ではパッケージ基板が不要であるためパッケージの厚みが薄く，またパッケージ基板による電源ロス，伝送ロスがないため低消費電力，高速伝送で有利である．TSMC は InFO と呼ぶ FOWLP を開発し，2016 年にモバイル向け APU を出荷開始し，現在では様々な種類のシリコンを統合する InFO のバリエーションを提供している．Samsung は FOWLP と同様の目的で FOPLP[*11]と呼ぶパッケージ技術を開発した．FOPLP は基板製造プロセスを使用し，2016 年にモバイル向け APU を出荷開始した．

米国 DARPA はトランジスタの微細化に代わり HI が今後のエレクトロニクスの進歩を支えるとの見地か

[*6] Chip-on-Wafer-on-Substrate
[*7] Through Silicon Via
[*8] Embedded Multi-die Interconnect Bridge
[*9] Fan-Out Wafer-Level Package
[*10] Wafer-Level Package
[*11] Fan-Out Panel-Level Package

ら，2017 年にチップレットの標準化を目指す CHIPS[*12]プロジェクトを開始し，さらに 2018 年から米国の半導体産業に 5 年間で 15 億ドルを投資するエレクトロニクス再興イニシアチブ ERI[*13]を開始した．2020 年には米国議会に CHIPS[*14] for America Act と American Foundries Act of 2020 の 2 つの法案が提出されている．前者は半導体産業への総額 120 億ドルの投資に関する法案で，内訳は DARPA ERI が 20 億ドル，NSF の半導体基礎研究が 30 億ドル，DOE の半導体基礎研究が 20 億ドル，商務省の先進パッケージングに関する国立製造研究所の設立が 50 億ドルとなっている．後者は半導体製造工場の誘致と支援を目的として商務省に 150 億ドル，国防総省に 50 億ドル，その他の研究開発に 50 億ドルを投資する．いずれの法案がどのような形で成立するにせよ，2020 年代に米国政府は先進パッケージング技術を根幹とした新しいマイクロエレクトロニクスに巨額投資を行うと考えられる．

■3D ウエハレベル・ヘテロジニアス・インテグレーション　2019 年 8 月に Cerebras Systems 社はウエハ・スケール・エンジン (WSE) という AI チップを発表した [9]．TSMC 16nm プロセスで製造され，17 x 9 のタイルで構成されている．シリコン面積は 46,225 mm^2（215mm 角），トランジスタ数は 1.2 兆個，AI 処理に最適化したコアを 400,000 個搭載する．各コアは 45kB の SRAM を持ち，WSE 全体では 18GB の SRAM を使用している．コアと SRAM 間の総バンド幅は 9PB/s にもなる．同社によると，WSE を搭載した単一のコンピュータ (CS-1) は，TPU2 を搭載した 10 ラックの Google AI コンピュータと比較して，1/5 の電力消費，1/30 の設置スペースで，3 倍のパフォーマンスを達成したという．CS-1 のサイズは，15 Rack Units (26.25 inches)，消費電力は 20KW，内部閉循環冷却液を用いた空冷システムを採用している．また初めてのケースでもあり，多くの新技術開発が必要であったようである．特に電源の供給が困難を極め，電源電圧 0.8V で電流 20,000 A をウエハー周辺から供給することは不可能であり，数百個の電源制御用 IC を搭載した回路基盤から 100 万本の銅の支柱を用いて WSE 上にポイント接続した．また，熱膨張係数が異なる銅，シリコン，グラスファイバー等の材料の調整と温度の管理が困難で解決までに 1 年半かかっている．

■マルチバンド RF 接続　新たな問題点として，電源供給の他に，コア間の配線，歩留まり向上のためのリダンダンシの追加，外部メモリとの接続，テスト，発熱等があげられる．これらの問題点を解決する手法として，3D ウエハーレベル・ヘテロジニアス・インテグレーション（異なる種類のウエハの貼合わせ）がある．電源供給用には，アナログプロセスで開発された電源用 IC と電源配線からなるウエハ，プロセッサ用に最先端プロセスで開発されたウエハ，メモリ用に DRAM や新メモリプロセスで開発されたウエハを貼合わせる相互接続（縦方向：Z) で配線効率の良いチップの開発ができる．また，この様に作られたチップをウエハサイズ限界まで大きくすることが可能だが，システムレベルの機能を実現するために，高スループットで低消費電力の優れた平面方向（X-Y）の相互接続が重要となる．チップ内グローバル・バス配線は，ロジック信号が配線の RC の影響を大きく受けるため，コンピューティングへの大きな制約となる．この基本的な物理制限を克服する効果的な方法として，電気信号から RF 信号への変更が提案されている．RF 接続に必要なドライバ，レシーバ，変調器，伝送路等に適した RF-SOI 等のプロセスを用いて開発されたウエハを追加することで，高度なマルチバンド RF 接続が可能になり，高速，低消費電力，フレキシブルな接続アーキテクチャーの構築，歩留まり向上により大口径のチップの開発ができるという．

一例として，周波数分割多元接続アルゴリズム（FDMA）に基づくマルチバンド RF 接続を用いて，オンチップのコア間通信を実現している．RF 接続で高速（0.18μm CMOS で 5〜10Gb/s)，低 BER（10-14 エ

[*12] Common Heterogeneous Integration and IP Reuse Strategies
[*13] Electronics Resurgence Initiative
[*14] Creating Helpful Incentives to Produce Semiconductors

ラー訂正なし)，シームレスな再構成可能な，共有伝送ラインを使用して複数の周波数帯域を介した複数のI/O 間の同時通信 RF 接続を実現している．先端プロセスを採用することで，帯域の数を増やし高速化，低消費電力化，小面積化が見込まれる．伝送路には，トップメタルレイヤを重ねてレイアウトし，他の配線層をグランドとして利用し低抵抗の伝送路をコア間に配置することで，マルチバンドの RF 接続と 1 つのベースバンドを利用できる．

また，チップ内オプティカル接続も可能であるが，接続距離が 300mm 以下であれば，RF 接続が有利である [10].

2.1.2 プロセッサ

本節では，2020 年現在においてスーパーコンピュータに採用されたプロセッサ製品群の概観と，現在公開されている短期的な将来動向について説明する．

現在，主要な地位を占めるスーパーコンピュータシステムへ利用されているプロセッサを長期的に安定して開発する企業は限られており，国外においては Intel, AMD, NVIDIA, IBM といった企業が主な供給元となっている．また，日本国内では富士通，NEC の 2 社が主な供給元となっている．スーパーコンピュータへの利用可能性が高いプロセッサはその他にも存在し，世界的には Ampere, Amazon, Huawei, Marvell, 国内では Pezy といった企業が自社での開発・実用化に乗り出している．これらの企業の製造されたプロセッサは，採用例や売り上げ規模としては大きくないものの，例として世界的なクラウドシステムの一翼を担う，搭載されたシステムが主要な世界ランキングで首位もしくは上位を得るなど，その動向は無視できない．また，世界ランキングで首位を得た中国のスーパーコンピュータ Sunway Taihulight に搭載されたプロセッサは中国国内で独自に開発されたものである．

上記の企業が製品として供給するプロセッサを，現代のスーパーコンピュータに利用されている経緯別に分類すると下記の通りである．

- 汎用パーソナルコンピュータを出自とするプロセッサ
- 画面描画用プロセッサを出自とするプロセッサ
- その他のプロセッサ

本節では，便宜上この分類に従い説明を行う．

2.1.2.1 汎用パーソナルコンピュータを出自とするプロセッサ

このカテゴリのプロセッサは，元々パーソナルコンピュータの中央処理装置 (CPU) として用いられていたものであった．しかし，ネットワークで多数のノードを接続したクラスタ並列機がスーパーコンピュータの主流になるにつれて，ノードの CPU としてコスト競争力の高いこのカテゴリのプロセッサが用いられるようになった．現在では大部分のシステムにおいてこのカテゴリに属するプロセッサが採用されている．

■Intel Xeon　2020 年現在，Intel は第 2 世代 Xeon Scalable Processor (コードネーム:Cascade Lake) を出荷している．本プロセッサは 14nm プロセスルールで製造され，ソケットあたり最大 28 コアを搭載可能であり，このときの最大同時スレッド実行数は 56 である．演算性能を求められるスーパーコンピュータ向けの機能として AVX-512 ベクトル拡張命令セットを備えており，最大ベクトル長 512 ビットの FMA 演算を可能としている．なお，最大 SKU である Xeon Platinum 8280 の場合，全コアで AVX-512 命令実行時の最大周

波数は 2.4GHz であり，この結果，倍精度浮動小数点演算性能は 2.15TFlops に達する．メモリシステムは DDR4-2933 規格に対応し最大で 6 チャネルを接続可能なため，データ転送能力は 140GB/s となる．また，Intel Optane Persistent Memory と呼ばれる不揮発メモリモジュールを用い，より大容量・低消費電力にすることも可能であるが，性能が低下する．外部拡張規格として PCIe 3.0 を備え，最大熱設計電力 (TDP) は 205W である．

今後の動向として，Intel は第 3 世代の Xeon Scalable Processor (コードネーム Ice Lake) をアナウンスしている．プロセスルールが 10nm+ となり，28 コア以上の SKU が登場することが予想される．メモリシステムは DDR4-3200 に対応し，8 チャネルまでの接続に対応する．外部拡張インターフェースは PCIe 4.0 にアップグレードされる．

■AMD EPYC　2020 年現在，AMD は第 2 世代 EPYC プロセッサ（コードネーム:Rome）を出荷している．本プロセッサは 1 つのソケットを複数のシリコンダイで構成し，その間をオンパッケージで接続するチップレット設計を特徴とする．シリコンダイは演算コアダイ (CCD) と IO ダイがあり，各ダイの製造プロセスは CCD が 7nm，IO ダイが 12nm である．IO ダイのプロセスルールが大きい理由は，IO ダイがソケット外通信を担っており，最新のプロセスルールを用いることによる省電力効果より歩留まりを優先したためである．スーパーコンピュータ向けのベクトル拡張命令セットとして AVX-2 を備えており，最大ベクトル長 256bits で Intel 製プロセッサに及ばないものの，コア数の増加によって演算性能の強化を実現している．最大 SKU である EPYC 7742 の場合，ソケットあたり 64 コア (8CCD) を搭載可能であり，倍精度小数点の FMA 性能は 2.304TFlops に達する．メモリシステムは DDR4-3200 に対応し，最大で 8 チャネルを接続可能なため，データ転送能力は 204GB/s に達する．外部拡張規格として PCIe 4.0 を備え，熱設計電力は 225W である．

今後の動向として，AMD は第 3 世代 EPYC(コードネーム:Milan) の 2020 年末投入，第 4 世代 EPYC(コードネーム:Genoa) の 2022 年末投入の計画を明らかにしている．第 3 世代 EPYC ではコアアーキテクチャが Zen3 に代わり，第 4 世代 EPYC ではメモリ規格として DDR5 への対応とプロセスノードの 5nm への移行などが計画されているもようである．

■IBM POWER　2020 年現在，IBM は第 9 世代 Power プロセッサ Power9 を出荷している．本プロセッサはテクノロジノード 14nm で製造され，ソケットあたり最大 24 コアを搭載可能である．

Power9 の大きな特徴としてメモリアーキテクチャをいくつかから選択可能なことである．Power9 は製品として 3 種類が用意されており，それぞれ Scale Out (SO), Scale UP (SU), Advanced I/O (AIO) と呼ばれる．SO タイプでは DDR 規格の DRAM を直接プロセッサに接続し，最大で 150GB/s のデータ転送能力が得られる．SU タイプではバッファとなるチップを介して DDR が接続される形式となり最大で 210GB/s のデータ転送能力が得られる．さらに，AIO タイプではバッファチップとの接続を Open Memory Interface (OMI) 規格でサポートし，最大 650GB/s のデータ転送能力が得られる．

今後の動向として，2021 年後半には Power10 が登場予定である．Power10 はテクノロジノード 7nm で製造され，チップあたり最大 15 コアを搭載可能である．また，周波数は 4GHz 程度になるとされ，チップあたり倍精度浮動小数点演算性能 3.84TFlops となる．また，Power10 はマルチチップソケット構成となっており，最大で 16 個のチップを 1 つのソケットとして扱うことができる．メモリ接続規格は OMI であり，データ転送能力はチップあたり 1TB/s である．

2.1.2.2 画面描画用プロセッサを出自とするプロセッサ

このカテゴリのプロセッサは，元々パーソナルコンピュータの画面描画処理用プロセッサ (GPU) である．これらのプロセッサは画面描画に必要な座標変換や明度，陰影といった計算のためにオンチップに多数の演算器を搭載している．2000 年代以降，この多数の演算器によって実現されうる性能が注目されるようになり，科学技術計算に利活用する研究が精力的に行われるようになった．そして，現在ではスーパーコンピュータ用のプロセッサとして主要な地位を築きつつある．

■NVIDIA Tesla 2020 年現在，NVIDIA は A100 (アーキテクチャ名:Ampere) を出荷している．本プロセッサの製造プロセスルールは 7nm であり，108 個のストリーミングマルチプロセッサ (SM) を搭載している．倍精度浮動小数点演算性能は 9.7TFlops に到達する．さらに，Tensor Core と呼ばれる行列演算専用の演算器を搭載しており，行列演算の場合の倍精度浮動小数点演算性能は 19.5TFlops に及ぶ．また，上記の倍精度浮動小数点形式は従来のシミュレーションを中心とした科学技術計算で主に用いられてきたが，混合精度計算を含む科学技術計算や機械学習などの新しい分野で使われる整数形式・低精度浮動小数点形式の演算能力にも主眼が置かれている．半精度，Bfloat16 の浮動小数点形式では 624TFlops，整数形式では INT8, INT4 でそれぞれ 1,248Tops，2,496Tops に到達する．メモリシステムは HBM2 に対応し，6 モジュールを接続しているため，データ転送能力は 1.56TB/s に達する．メモリ容量は 40GB である．最大電力は 400W である．本プロセッサは PCIe カード上に搭載されており，Intel Xeon や AMD EPYC 等の汎用プロセッサ等に PCIe 4.0 規格で接続する形で利用する．また，汎用プロセッサ側が対応していれば高速な NVLink 規格での接続も可能であり，PCIe 4.0 による接続が 64GB/s であるのに対して，NVLink による接続速度は最大 600GB/s と高速である．

■Intel 2020 年現在，Intel は新型 GPU である Xe をアナウンスしている．Xe は 10nm SuperFin テクノロジノードで製造される予定である．Xe には 4 つのマイクロアーキテクチャ，HPC & Machine Learning Optimized (HPC)，Machine Learning & Media Optimized (HP)，Discrete Gaming Optimized (HPG)，Low Power Optimized (LP) が存在し，スーパーコンピュータ用として利用されるのは HPC マイクロアーキテクチャ (コードネーム:Ponte Vecchio) である．Xe は Co-EMBI による Multi-tile GPU となっており，要求性能に応じて 4 つまでの計算ダイをオンパッケージに集積することが可能である．HPC マイクロアーキテクチャにおいては最大で 2 タイル程度が想定されており，その性能は単精度浮動小数点演算性能で 20〜40TFlops が予定されている．また，HBM2 がメモリとして採用される予定である．

LP マイクロアーキテクチャについてはすでに第 11 世代 Core プロセッサ (コードネーム:Tiger Lake) の一部である Iris Xe Graphics として正式発表されている．

2.1.2.3 その他のプロセッサ

本節ではこれまで説明してきたカテゴリに当てはまらないプロセッサについて扱う．このカテゴリのプロセッサとしては，スーパーコンピュータ用として設計されたものや，各国・各機関の独自製造技術の開発と確立を目的として設計されたもの，汎用パーソナルコンピュータの新規市場獲得を目的として設計されたもの等が挙げられ，登場の経緯は様々である．

■富士通 富士通は日本のナショナルフラッグシップスーパーコンピュータ用プロセッサとして 2019 年より A64FX を出荷している．また，本プロセッサは同社 FX シリーズ用のプロセッサとして民生用にも出荷され

ている． 本プロセッサは 7nm のプロセスルールで製造されており，搭載コア数は 52（うち演算用コア数 48）である．スーパーコンピュータ向けのベクトル拡張命令セットとして Scalable Vector Extension (SVE) を備えており，最大ベクトル長は 512bits である．この結果，最大 SKU である動作周波数 2.2GHz の場合，倍精度浮動小数点演算性能は 3.3TFlops に達する．なお，SVE はプロセッサ側に最大ベクトル長を問い合わせる機能があり，将来的に最大ベクトル長が伸びた場合でも実行バイナリの改変なしで動作する前方互換性を持つ．メモリは HBM2 に対応し，4 モジュールを接続しているため，データ転送能力は 1TB/s に達する．メモリ容量は 32GB である．外部拡張インターフェースとして PCIe 3.0 を備える．また，Tofu と呼ばれる独自の 6 次元メッシュ/トーラスネットワークに接続可能なインターフェースを持つ．

■NEC　NEC はベクトル型スーパーコンピュータとして SX-Aurora TSUBASA シリーズを 2018 年から販売しており．その主要部品であるベクトル型プロセッサ (VE) の第 2 世代を 2020 年より出荷している．本プロセッサは 16nm のプロセスルールで製造されており，搭載コア数は 10 である．VE はベクトル演算性能の向上を主眼に設計されており，最大ベクトル長は 16384bits と，他のプロセッサと比較してかなり大きい．最大 SKU である Type 20A の場合，周波数は 1.6GHz であり，倍精度浮動小数点演算性能は 3.07TFlops に達する．メモリシステムは HBM2 に対応し，6 モジュールを接続しているため，データ転送能力は 1.56TB/s に達する．メモリ容量は 48GB である．本プロセッサは PCIe カード上に搭載されており，Intel Xeon や AMD EPYC 等の汎用プロセッサに PCIe 3.0 規格で接続する形で利用する．

今後の動向として，2021 年頃に第 3 世代の出荷が計画されている．

2.1.3　メモリ技術

計算機システムの性能向上において，ロジックと同様にメモリ技術の進歩が大きな役割を担っている．近年の計算システムにおけるメモリサブシステムはオンチップメモリとオフチップメモリから構成されており，プロセスの微細化のみならず，設計とテクノロジの協調設計によるセルスケーリングにより，ムーアの法則には及ばないものの，高集積化と低消費電力化を成し遂げている．一方で，オフチップメモリの主要デバイスである DRAM は 10nm の壁に阻まれてスケーリングの鈍化が顕在化し，バースト転送を前提とした伝送技術，三次元積層技術により性能向上を維持している．本節では，オンチップメモリとオフチップメモリ，および新規メモリデバイスのトレンドについて述べる．

2.1.3.1　オンチップメモリ

オンチップメモリの主要なメモリ素子としては SRAM が用いられている．これまでは，SRAM のスケーリングは基本的にプロセスのスケーリングに伴って進行し，高集積化と低消費電力化を実現してきた．しかし，20nm 以降のテクノロジではプロセスのスケーリングが鈍化し，セルスケーリングによって高集積化と低消費電力化を図っているが，これまで通りのスケーリングを既存テクノロジの延長で維持することは困難とされている．また，オフチップメモリバンド幅と CPU の演算性能の乖離は拡大し続けるばかりであり，これらのギャップを埋めるべく 2020 年現在，ハイエンドの CPU では 1MB の L2 キャッシュ，60MB を超える LLC が搭載されるなど，オンチップメモリの容量は増加傾向にあり，この傾向は今後も続くことが予想されている [11]．SRAM はチップ上での記憶密度は低いが，CPU と同じプロセスで形成できるので，現在でも主要なオンチップデバイスとして活用されている．

近年では，オンチップメモリの記憶密度向上，低消費電力化の実現に向けて，大容量の LLC に eDRAM

表 2.2　DDR/HBM メモリの性能諸元 (HBM3 は予測)

	DDR					HBM			
Specification	DDR	DDR2	DDR3	DDR4	DDR5	HBM	HBM2	HBM2E	HBM3
Year	2000	2003	2007	2014	2020	2015	2017	2018	2022
Max Capacity per module (GB)			8	32	128	4	8	24	64
Max BW (GB/s)	～32	～8.53	～17	～25.6	～51.2	～128	～307	～410	～512
Year	2000	2003	2007	2014	2020	2015	2017	2018	2022

(1T-SRAM) が採用されるなど，SRAM を代替するデバイスについての検討が進められている．しかし，既存の 1T-SRAM は，平面状のキャパシタが一定の面積を必要とするため，メモリセル面積は高々 SRAM の 1/3 に留まっている．また，DRAM と同様に定期的なリフレッシュを必要とする．これらの課題を解決するべく近年では，リフレッシュを必要とせず，SRAM のマクロ半分以下の面積で実装可能な Bi-SRAM, SRAM-like-MRAM など大容量オンチップメモリの実現に向けた新メモリの開発が行われている．また，高速なオンチップメモリとして利用が期待されているスピン軌道トルク素子 (SOT) を用いた SOT-MRAM 等のプロトタイプ実装，実用化に向けた検討が進められている．

2.1.3.2　オフチップメモリ

現在オフチップメモリとして最も使われているのは DRAM で構成されるデバイスである．DRAM においては，現在第 4 世代の規格である DDR4 が広く活用されているが，DRAM のスケーリングは 10nm に向けて鈍化しており，DDR5 規格の制定も当初予定より大幅に遅れ，2020 年の 7 月にその規格が公表された [12]. 表 2.2 に DDR メモリ各世代の仕様を示す．DDR5 は，チップモジュールあたり 16Gbit の容量を有し，1.1V 供給電圧の下で，51.2GB/s のメモリバンド幅を達成することが期待されている．新世代の規格の DDR5 では DDR4 に比べて，2 倍のチャネル数と 2 倍のバースト転送長を採用するなどにより，2 倍のメモリバンド幅を達成しており，現在の 2020 年で既にいくつかのエンジニアリングサンプルが発表され，2021 年には市場に出ることが予想されている．次世代規格である DDR6 が制定されるまでは，DDR4 規格と同等の期間にわたり高性能計算分野で活用されることが予想される．

一方，より高いメモリバンド幅を要求する，富岳，SX-Aurora TSUBASA, NVIDIA GPU などのシステムでは，TSV とシリコンインターポーザを用いた HBM が採用されている．HBM は DDR と比べて高いバンド幅と電力効率を有し，第一世代ではモジュールあたり 128GB/s，最新の HBM2E では 410GB/s のメモリバンド幅を実現している．次世代の HBM3 では，ピンあたりのデータ転送速度を向上させることで 512GB/s のバンド幅が期待されるなど堅調な性能向上を遂げている．一方で，これらの二つのメモリデバイスには，容量，メモリバンド幅，コストのトレードオフがある．このため今後は，これらのトレードオフを考慮しながら，DDR と HBM が引き続き高性能計算システムのメモリ素子として利用されていくことが予想される．

大規模なシステムの消費電力においては，メモリサブシステムはシステム全体の 3 割の電力が消費される可能性があると言われる中 [13]，メモリサブシステムの省電力化の検討も精力的に進められている．その代表的な例が不揮発性メモリの活用である．不揮発性を有する新規メモリデバイスに関しては後述するが，2020 年現在，様々な民生品等，大規模システムにおける SCM（ストレージクラスメモリ）として活用されているのがフラッシュメモリである．フラッシュメモリは DRAM/SRAM と比べてビット単価が安い一方で，読み書

き速度が低いため，これまで DRAM/SRAM の代替として活用されるに至っていない．一方で，3 次元積層により，著しい勢いでその容量を増加させており，2020 年現在その積層数は 128 に達している [14]．積層数の増加によるワードラインの伸張に対する技術的な課題はあるものの，当面は一定の容量増加が期待されている．フラッシュメモリには NAND 型，NOR 型があるが，NAND 型が NOR 型に比べて書き込み速度が高く，NOR 型は読み取り速度が高いという特性を持つため，NOR 型は組み込み機器のコードストレージ，NAND 型は SSD 等のストレージデバイスとしての活用が進められている．DRAM の代替，DRAM との併用等を目的に，メモリタイプ SCM（M-SCM）に関する研究も進めらているが，高性能計算システムにおいて，DRAM の代替としての活用には技術的な課題が多いのが現状である [15].

2.1.3.3 新規メモリデバイス

DRAM スケーリングの限界を見据え，更なる省電力システムへの要求に応えるべく，新たなメモリデバイスの開発が精力的に進められている．特に，近年の省電力化を目的とした不揮発性メモリにおける技術革新には目を見張るものがある．これらの不揮発性メモリデバイスは組み込み用途のオンチップメモリとしての開発，DRAM に代わるメインメモリとしての開発も進められている一方で，ストレージとメインメモリのレイテンシの乖離を補完するデバイスとして注目を集めている．Flash 技術を用いた SSD (Solid State Drive) は，HDD (Hard Disk Drive) の代替デバイスとして活用され，メインメモリとストレージのレイテンシの差を大幅に短縮しているが，それでも双方のレイテンシは依然として大きく乖離している．さらに，近年のデータインテンシブアプリケーションの台頭，機械学習，深層学習に対する高いニーズと相まって将来の高性能計算システムにおいて，ストレージとメインメモリの性能の乖離を補完するメモリデバイスへの要求が高まっている．2020 年現在，不揮発性メモリデバイスとしては相変化メモリ（PCM: Phase Change Memory）, 抵抗変化メモリ（ReRAM: Resistive RAM）やその発展形であるカーボンナノチューブメモリ NRAM (Nanotube RAM), 磁気抵抗メモリ（MRAM: Magnetic RAM），などに関して開発が進められている．各不揮発性デバイスの開発状況，期待される応用分野について以下に述べる．

- PCM(Phase Change Memory)，3D XPoint Memory:
  PCM は不揮発性メモリとして早い段階から開発が進められてきたが，データの読み込み，書き込みのオーバーヘッドが共に大きく，広く普及するには至らなかった．しかし．PCM 技術を用いた 3D XPoint メモリの登場により，あらためて注目を集め，現在では HDD のキャッシュや SSD へ応用した製品として市場で一定の需要を満たしている．3D Xpoint メモリは NAND Flash と比較して高速であり，さらに DRAM と同等以上の記憶密度を実現していることから，DRAM の互換デバイスとしての検討も進められている．しかし DRAM と比べて， 読み込み，書き込みのレイテンシが大きいため，メインメモリとして利用した場合に DDR4 よりも 2.3 倍ほど低速であることが報告されている [16]．このことから，これらの不揮発性メモリは，今後も主に大容量であることと低レイテンシの特性を活かしていわゆるストレージクラスメモリとして利用されていくものと予想される．
- ReRAM (Resistive RAM), NRAM (Nanotube RAM):
  ReRAM は，これまで 20 年程度開発が続けられている不揮発性のメモリデバイスである．ReRAM は，1 個のセル選択トランジスタと 1 個の抵抗変化記憶素子から構成されるメモリセルを有し，電圧印加によりデータを書き換えるものであり，低消費電力かつその単純な構造から高密度実装が可能なデバイスとして注目を集めてきた．一方で，データの読み出しは比較的高速であるが書き込みのオーバヘッドが大きいことが主な理由で，大量生産ベースの市場への投入，普及には至っていないのが現状である．現

在は製造コストの低減，セルをマトリクス状に配置する 3D Xpoint 構造の導入による高性能化に取り組みながら，組み込み向けのメモリとして開発が続けられている [17].

ReRAM の派生形として，NRAM が挙げられる．NRAM は記憶素子にカーボンナノチューブを用いて，カーボンナノチューブを含んだ層を電極層で挟んだ比較的単純な構造を持ち，電圧の印加によりカーボンナノチューブの距離を調整することで，抵抗値を制御して書き込みを行う．NRAM は，1 兆回のデータの書き換えを可能にするなど，現存の不揮発性メモリの中では最も高い耐久性を有するだけでなく，DRAM と同等の性能を有することから，現在では DRAM を補完，もしくは DRAM にとってかわるメモリ素子として期待され，そのプロトタイプ実装が発表されるなど，精力的な開発が進められている [18].

- MRAM (Magnetic RAM):

  MRAM は，磁化の状態を利用するメモリである．DRAM や NAND 型フラッシュメモリではメモリセルに電荷を蓄積することでデータを保持することから，その構造は大きく異なる．MRAM は DRAM とほぼ同等の速度を実現可能であることに加えて，不揮発性を有することから DRAM，もしくは SRAM に代わる省電力デバイスとして注目されている．MRAM は，磁化の方向によって記憶素子の電気抵抗が変わることを利用したメモリであリ，磁気トンネル接合（MTJ：Magnetic Tunneling Junction）と呼ばれる基本素子でメモリセルが構成されている．従来は，磁化の反転には外部磁界を用いていたが，電子スピンのトルクを用いて磁化制御を可能にした STT(Spin-Transfer Torque) 方式を採用した STT-MRAM が近年注目されている．STT-MRAM は，Flash メモリ，PCM, ReRAM などの多くの不揮発性メモリと比較して特に書き込みの速度が早いことが知られている一方で，集積度と製造コストが高いことが知られている．しかし，これらの課題を克服すべく，多くのベンダが開発を進めている [19]. [20] では，リテンションが 10 年から数日に短縮されるものの，MTJ のサイズを大幅に縮小することで集積度の向上を図る SRAM-like-MRAM を開発して，大容量のオンチップメモリとしての MRAM の可能性を示唆している．また，SOT(Spin-Orbit Torque)-MRAM のように面積は大きくなるものの，サブナノ秒の書き込みが確認されるなど，オンチップメモリとしての可能性の検討も進められている．

上述の不揮発性メモリ以外にも，オンチップメモリである SRAM を置き換えるデバイスの開発も進められている．SRAM に代わるオンチップメモリとして Embedded DRAM (eDRAM) が高性能マイクロプロセッサに採用されているが，電力コストの高いリフレッシュ動作が必要であることに加えて，6 トランジスタ構成の SRAM の 1/3 のサイズに留まっている点が課題として挙げられている．これらの課題の解決を目的として Bi-SRAM と呼ばれる SRAM と同様に動作が可能なデバイスの開発が zeno 社によって進められ，同一プロセスの SRAM マクロと比較して，同等の静的電力でありながら，1/2 の面積，1/5 の消費電力が実現可能であることが報告されており，１次キャッシュ，2 次キャッシュにおける SRAM の代替デバイスとしての可能性を示唆している [zeno]．また，Flash 特有の多層積層による高集積化技術を活用しながら，高い応答性を有する 3D NOR-P CT-DRAM[3DNCD] などの DRAM の代替デバイスの開発も進められている．しかし，高性能計算機システムにおいては，膨大なメモリモジュールを必要とすることから，これらの技術の成熟度，量産の可否を見据えたシステム設計が必要になることは言うまでない．5〜10 年先を見据えると，新たなメモリ素子の研究開発が進められることは間違いないが，HBM, DDR の現世代，もしくは次世代規格が今後も主要なデバイスとして考えられ，段階的に詳述した新規デバイスの導入が進められるものと考えられる．

2.1.4 データ転送技術

本節では 2020 年現在のデータ転送技術と今後の動向を説明する．まず現在の主流である高速シリアル伝送について述べる．2028 年システムの開発時期には次々世代の 100Gbps 伝送を使用する各種規格製品が成熟していると期待できる．次に今後重要となるパッケージ内のダイ間伝送技術動向を説明する．2028 年システムの開発時期にはダイ間の高密度配線で 0.5pJ/bit 以下の低消費電力伝送が利用可能と考えられる．さらにこれらの伝送技術を使用する各種プロトコルと，筐体間の高速伝送に必須のメディアである光伝送技術について述べる．2028 年システムでは，特にダイ間接続のプロトコルにはキャッシュコヒーレンシを考慮した CCIX, CXL などのプロトコル，光伝送には Co-packaged Optics が想定される．

2.1.4.1 高速シリアル伝送

現在，InfiniBand EDR，Omni-Path，25G/100G イーサネットなど多くの筐体間伝送で伝送速度 25Gbps の高速シリアル伝送が使用されている．標準的な 100G QSFP28 MSA(SNIA SFF-8665) 準拠コネクタを介した電気ケーブル接続では，同軸ケーブルの太さにもよるが長さ 3〜5m 程度の伝送が可能である．これを超える伝送距離が必要な場合には伝送路の途中に光モジュールを挿入して光で伝送するが，光モジュールの電気側入出力も OIF CIE-28G-VSR もしくは IEEE CAUI-4 といった高速シリアル伝送である．

次世代の 50Gbps 伝送では InfiniBand が HDR 規格製品を 2019 年から出荷開始し，イーサネットも 2019 年から 400G 規格製品が発表されている．InfiniBand HDR は 4 レーンの QSFP56 コネクタを使用し，400G イーサネットは 8 レーンの QSFP56-DD(http://www.qsfp-dd.com/) または OSFP コネクタ (https://www.osfpmsa.org/) を使用する．50Gbps 伝送では伝送損失の悪化を避けるため，25Gbps 伝送と同程度の 25Gbaud 強の変調レートを使用し，1baud あたりパルス振幅変調で 2bit を転送する PAM4 技術を使用する．PAM4 ではビット誤り率が上昇するため，その補償として FEC(Forward Error Correction:前方誤り訂正) を適用する．400GBASE-R では FEC として RS(544,514,15,10) を使用する．FEC ブロックは 5440 ビットで，5140 ビットのメッセージと 300 ビットのパリティで構成され，最大 15 個のシングルビットエラーか，最大 150 ビットのバーストエラーを訂正できる．FEC の導入により 50Gbps 伝送でも 3m 程度の長さの電気ケーブル接続が可能になっている．50Gbps の伝送速度における FEC のオーバーヘッドは約 100ns の遅延に相当する．HPC の End-to-end 通信では最短遅延は 1us を下回り，スイッチ遅延も典型的には 100ns 以下であるので，1 ホップ毎に FEC による 100ns の遅延というのは大きい．そこで FEC の遅延を半減するため，25/50G イーサネットを推進する Ethernet Technology Consortium は 2720 ビットの FEC ブロックを使用する Low latency FEC を提案している．

次々世代の 100Gbps 伝送では変調速度を 25/50Gbps 伝送の 2 倍にあたる 50Gbaud 強，クロック周波数にして 25GHz 以上に引き上げる方式が有望である．25GHz 以上の周波数の信号は特にプリント基板における伝送損失が大きいので，伝送距離を短くする高密度実装が必要になる．ここで既存の装置の実装方式を維持する場合，伝送路の途中にリタイマと呼ばれる素子を挿入して伝送損失を補償する必要があり，コストと消費電力が増大する．伝送損失問題の別の対策として，プリント基板やコネクタ類を飛ばす Jump-over ケーブル (Samtec Flyover, Molex BiPass I/O) が提案されている．また，IC パッケージ基板に光モジュールを載せる Co-packaged Optics(後述) も研究開発が進んでる．さらに先の 200Gbps 伝送については今後の研究が期待される．50/100Gbps 伝送で高密度実装や Jump-over ケーブルが成功した場合，さらに変調速度を倍増する方向で検討が進むものと予想される．

2.1.4.2 Die-to-Die インタフェース

プロセッサの MCM (Multi-Chip Module)，MCP (Multi-Chip Package) 化と，同一パッケージに異なる半導体プロセスのダイを統合する HI (Heterogeneous Integration) の進展により，パッケージ内のダイ間接続が重要になっている．パッケージ内の接続距離は数十 mm 以下で短く損失は小さいため，既存の伝送技術を転用する場合でも低電圧伝送による低消費電力化や，より高速な伝送が可能である．AMD の IFOP (Infinity Fabric On-Package) 伝送技術はシングルエンド非対称終端信号技術の一つであり LPDDR4 SDRAM に適用されている LVSTL (Low Voltage Swing Terminated Logic) 伝送を使用し，5〜6Gbps の高速伝送と 2pJ/bit の低消費電力を実現している．なお，パッケージ基板の設計ルールは配線幅 10μm 程度とプリント配線板の配線幅 100μm 程度に対して 1 桁微細であるが，高速信号ではドライバの出力インピーダンスと伝送路の特性インピーダンスを整合する必要から配線幅が制約されるため，既存の高速伝送技術で高密度配線を実現することは困難である．

実際の製品で高速，高密度，低消費電力なダイ間伝送を実現するには，目標とする伝送速度，距離，配線密度において伝送損失，ノイズ耐性，クロストーク耐性などの条件を満たすだけでなく，想定する MCM/MCP 製造上の様々な制約も満たさなければならない．このため多数のベンダーが合意する標準技術の確立が欠かせない．OIF は 2018 年にダイ間接続を想定した CEI-56G-USR 規格，Co-Packaged Optics などでの利用が予想される CEI-56-XSR 規格を公開した．また，OCP Server プロジェクトの ODSA (Open Domain-Specific Architecture) サブプロジェクトは 2019 年に BoW (Bunch of Wires) インタフェース仕様を公開した (https://github.com/opencomputeproject/ODSA-BoW)．

近年では半導体製造プロセスで配線層を形成した Si-IP (シリコン・インターポーザ) の上に複数のダイを搭載することで，LSI のグローバル配線と同等の配線幅 1μm 程度まで密度を高めたダイ間接続も実現されている．また Intel は Si-IP と同等の配線密度を実現する EMIB (Embedded Multi-die Interconnect Bridge) 技術を開発した．さらに，個片化された複数ダイに跨る再配線層を形成する Multi-Chip FOWLP (Fan-Out Wafer-Level Package) も提案されている．これらの接続は伝送路が短く配線密度が極めて高いという特性があり，これに適した伝送技術が必要となる．現在，Si-IP で接続する HBM1/HBM2 が多くの製品に使用されており，その接続は JEDEC の JESD235 規格で標準化されている．HBM の信号はおよそ 1.1mm × 6.0mm のエリアのマイクロバンプから約 1600 本引き出される．データ信号は 1,024 ビットであり，2020 年に更新された JE235C 規格では伝送速度は 1.0〜3.2Gbps となっている．Intel の FPGA 製品では AIB(Advanced Interface Bus) が使用されており，伝送速度は 2.0Gbps でエネルギー効率は 0.85pJ/bit となっている．Intel の次世代伝送技術 MDIO は伝送速度 5.4Gbps，エネルギー効率 0.50pJ/bit，TSMC の次世代伝送技術 LIPINCON は伝送速度 8.0Gbps，エネルギー効率 0.56pJ/bit と発表されている．

2.1.4.3 ノード内インターコネクト（ノード内の相互接続）

ノード内インターコネクトにはプロセッサ間でキャッシュされた内容の一貫性（整合性）を保ちながらメモリを共有するためのキャッシュコヒーレントインターコネクトと，拡張デバイスを接続するための I/O インターコネクトがある．キャッシュコヒーレントインターコネクトはキャッシュコヒーレントプロトコルで通信し，システムアーキテクチャ依存でトポロジーも多様である．I/O インターコネクトは I/O プロトコルで通信し，システムアーキテクチャ非依存でトポロジーもスター型か，スイッチを挟む程度で簡素である．キャッシュコヒーレントプロトコルと I/O プロトコルはメモリモデルが異なるため，両者はブリッジで接続される．

キャッシュコヒーレントプロトコルは一般的に知的財産権で保護された技術を含み，各社の独自仕様であ

る．Intel が 2017 年から使用している UPI (Ultra Path Interconnect) は伝送速度 9.6〜10.4 GT/s の高速シリアル伝送でプロセッサ間を接続する．接続レーン数はプロセッサのモデルや接続ソケット数で異なる．前世代の QPI (QuickPath Interconnect) ではクロックとデータがそれぞれ別個の差動信号となっている差動パラレル転送であったが，UPI よりクロックとデータを重畳する高速シリアル伝送となり，伝送速度が向上した．AMD が 2017 年から使用している Infinity Fabric はソケット間を接続する IFIS (Infinity Fabric Inter-Socket) と，前節で紹介したパッケージ内接続の IFOP で構成されている．IFIS は伝送速度 10.76〜18 GT/s の高速シリアル伝送を使用している．IFIS は PCI Express や SATA と PHY を共有しており，プロセッサのモデルや接続ソケット数でレーン数が異なる．

I/O インターコネクトは一般的に仕様が標準化されている．現在主流の PCI Express では，2010 年に策定された Gen3 仕様 (PCI Express Base Specification Revision 3.0) に対応する製品が 2012 年頃から，2017 年に策定された Gen4 仕様対応製品が 2019 年頃から出荷されている．2019 年には Gen5 仕様が策定され，2021 年頃に Gen6 仕様が策定される予定である．伝送技術は高速シリアル伝送で，伝送速度は Gen3 が 8.0GT/s，Gen4，5 がそれぞれ 16，32GT/s となっている．Gen6 では PAM4 を使用して伝送速度を 64GT/s に引き上げる見通しである [21, 22]．レーン数は拡張デバイスによって異なり，GPU では 16 レーン，HPC インターコネクトでは 8 または 16 レーン，NVMe ストレージでは 4 レーンが主流である．PCI Express は Root Complex と呼ばれるブリッジによってキャッシュコヒーレント領域に接続する．キャッシュコヒーレント領域には記憶階層があるが，メモリモデルの違いから Root Complex が記憶階層上位のキャッシュに直接接続するのは難しく，多くの実装では記憶階層下位の主記憶に接続する．CPU による記憶階層操作の補助が必要となる場合すらある．そのため PCI Express を介した CPU と拡張デバイス間のインタラクションはオーバーヘッドが大きい．

CPU に PCI Express 経由でアクセラレータを接続する場合，PCI Express に由来するオーバーヘッドが CPU とアクセラレータ相互間の細粒度の協調動作を阻害する．これを解決するため，アクセラレータを記憶階層上位のキャッシュに直接接続できる新しいインターコネクトが提案されている．CCIX (Cache Coherent Interconnect for Accelerators, https://www.ccixconsortium.com/) は PCI Express 仕様を拡張してキャッシュコヒーレントプロトコルを定義したものである．2016 年に検討が始まり，2018 年に Gen1 仕様が策定された．伝送，電気，機構仕様は PCI Express から流用しており，典型的には PCI Express/CCIX 兼用ポートとして実装されることを想定している．CCIX のキャッシュコヒーレントプロトコルとトポロジーはある程度の柔軟性があるが，逆にシステムアーキテクチャ依存性を残している点は短所と言える．また，CCIX ベースでキャッシュコヒーレント領域を構築することも原理的には可能であるが，実際には独自仕様のキャッシュコヒーレント領域をベースとし，CCIX ブリッジと PCI Express Root Complex が PHY を共有する実装形態が主になると考えられる．CXL (Compute eXpress Link, https://www.computeexpresslink.org/) は 2019 年に Gen1 仕様が公開された新しいインターコネクトである．伝送，電気，機構仕様を PCI Express から流用している点は CCIX と同様だが，従来型のキャッシュコヒーレントプロトコルではなく，拡張デバイスが記憶階層とインタラクションする新しいプロトコルを追加している点が異なる．CXL は CXL.io，CXL.cache，CXL.memory の 3 つのサブプロトコルからなり，CXL.io は PCI Express のプロトコルの流用である．CXL.cache は拡張デバイスが主記憶をキャッシュし，CXL.memory は CPU が拡張デバイスのメモリをアクセルするためのプロトコルとなっている．今後 Chiplet 化が拡張デバイスにも波及して CPU と拡張デバイス間接続の物理層が Die-to-Die インタフェースに移行した場合，データ転送はより低遅延，高帯域となるため，CCIX や CXL のようなオーバヘッドの小さいプロトコルへの要求が高まると予想される．

2.1.4.4 光伝送

電気信号の配線による伝送は，帯域や距離の増大に応じて減衰するが，光信号の（適切な光路を用いた）伝送はほぼ減衰しない (0.05 dB/cm)．シリコンフォトニクスによる Co-packaged Optics の研究開発を鑑みると，今後徐々に光と電気 (OE) の境界線が LSI 側に近づく方向で研究開発が進むことが予想される．データセンタにおいては，スイッチング方式の最適化 (LEAF や SPINE など) によるスループット帯域の改善 (2015 年に 3Tbps から 2021 年には 25Tbps) が予測されている．しかし，サーバのラック部分に配置される AOC (Active Optical Cable) によるデータ通信速度の改善速度が追いついておらず，計算機間を繋ぐ光回線の高度化が今後重要な技術と予想される．光伝送のトレンドとしては，光の伝送方式の変化 (マルチモードファイバからシングルモードファイバへ)，高速光受信/送信機の開発，CPU と光素子の異種統合 (Co-packaged Optics) などがトレンドとして考えられる．光回路実装に向けては大きく 3 フェーズあり，現在はボードエッジ実装型のフェーズ 1，もしくは LSI の近傍に光パッケージを配置するフェーズ 2 (COBO, 1 チャネル 400 Gbps) が実現されている．Co-packaged Optics は第 3 フェーズである．シリコンフォトニクスの発展を背景に，2020 年現在の研究開発の主な対象となっている．フェーズ 2 以前と異なり，基板やパッケージ間を繋ぐ配線や減衰信号補償用の DSP が不要になるため，フェーズ 3 の実現により，大幅な帯域と電力の改善が見込まれている．フェーズ 3 の問題点としては，CPU に比べた光素子サイズであり，光素子の高密度実装が主な技術的課題点である．現状の Co-packaged Optics では，25 Tbps に追いつくためには，LSI 本体より光受信機・送信機のサイズが大きくなり，設計コストの増大が大きな課題点である．光モジュールの積層化，光入出力の高集積化技術などが今後の技術開発のトレンドになる．

2.1.5 ASIC/FPGA

限りある半導体リソース，消費電力，熱の制限，等々の理由で汎用プロセッサや GPU ではアプリケーションが要求する演算リソースを提供できない場合がある．用途が明確な場合には，アプリケーションに特化した ASIC を開発するか，あるいは FPGA の利用が考えられる．ASIC や FPGA 上に用途に特化したロジックを実装するメリットは，対象とするデータに特化した演算が可能になること，アプリケーションに内在する並列性を活用できること，メモリアクセスの効率化が可能であること，演算に対するオーバヘッドを削減できることである．

ASIC は，目的とするアプリケーションのためにすべてのリソースを利用することができる．そのため，電力や熱の制約の中で最も高い演算性能を発揮することができる．しかし，設計コスト (NRE) が高く，また製造後のプログラマビリティは犠牲となる．一方 FPGA は，アプリケーションロジックを実行時に半導体リソース上に展開することができる．そのため NRE は低く抑えられ，また，入出力データに依存するような実行時に最適化の余地がある．ASIC と比べるとリソース使用量および動作周波数の点で 10〜100 倍ほどの性能低下があるが，高密度の専用ロジックで実装されたインタコネクタや演算ユニットを混載することで性能低下の欠点をカバーする流れがある．

2.1.5.1 ASIC

用途が明確である場合には用途に特化した ASIC を開発することで，電力リソースや熱の制約の下で最も高い性能のプロセッサを得ることができる．EDA ツールの進化，IP コアの充実により開発コストが低くなっている．たとえば，Google が開発した Tensor Processing Unit(TPU) v1 の開発からデプロイまでは 15 ヶ月

表 2.3　アプリケーション特化プロセッサのスペック

	プロセス	Tr 数	周波数	面積	メモリ	演算性能	消費電力	発表年
TPU v1	28nm		700MHz		28Mbit (SRAM)	23TOPS (8bit MAC)	40W	2016
TPU v2					16GB (HBM)	45TOPS (bfloat)	200W	2017
TPU v3					32GB (HBM)	90TOPS (bfloat)	250W	2018
Cerebras Wafer Scale Engine	16nm (TSMC)	1,200 億		46,225 mm^2	18GB (SRAM)	3.3PFLOPS	15KW	2019
MN-Core	12nm (TSMC)		700MHz			32.8TFLOPS (double) 131TFLOPS (single) 524TFLOPS (half float)	500W	2018
MK2 IPU	7nm (TSMC)	594 億		832 mm^2	900MB (SRAM)			2020
Groq TSP	14nm	268 億	1.25GHz		220MB (SRAM)	1POPS (INT8) 250TFLOPS (FP16)		2019
SambaNova SN10	7nm (TSMC)	400 億			数百 MB (SRAM)	数百 TFLOPS		
Gaudi					32GB (HBM)		200-300W	2019
MDGRAPE-4A SoC	40nm		600-800MHz	256 mm^2	2MB (SRAM)	1.3PFLOPS (512 コア)		2019

であったと報告されている [23].

ASIC はそれぞれ目的に特化して開発されるためその性能を一様に比較することは難しいが，表 2.3 は，製造プロセス，面積，トランジスタ数，搭載メモリ量，公称スペックなどをまとめたものである．ここで取り上げた ASIC は，いずれも大規模なオンチップメモリ (SRAM) あるいは高速で大容量な HBM をチップ上に持つという共通した特徴がある．一方で，性能評価に用いている演算やベンチマークが同一ではないこと，演算器の構成が異なることから，ピーク性能や動作周波数でチップ同士を比較することには意味がないことに注意されたい．

以降，ASIC の開発事例を紹介する．

- Google Tensor Processing Unit(TPU):
  TPU は Google によって開発された，ディープニューラルネットワーク向けのプロセッサユニットである．行列演算に特化したシストリックアレイとデータ入出力機構を持つプロセッサである．2017 年に発表された TPU v1[23, 24] から，TPU v2，TPU v3 と開発が継続され [25, 26, 27]，2020 年に TPU v4[28] が発表された．TPU v1 は，28nm プロセスで製造された 65536 個の 8bit MAC 行列演算

ユニットと 28Mbit のオンチップメモリを搭載したチップで，動作周波数は 700MHz，消費電力 40W で演算性能は 23TOPS であった．ホストプロセッサとのインターフェイスは PCIe Gen3 x16．TPU v2，v3 のバージョンアップでは，それぞれ 1 チップ (2 コア) あたり 16GB あるいは 32GB の HBM を搭載し，演算性能が 45TOPS，90TOPS に向上した．2 次元トロイダルメッシュネットワークで接続して拡張することができ，512 コアの v2-512 あるいは 2048 コアの v3-2048 が運用されている．TPU v3 を 4906 個，数百個のホスト CPU マシンが専用ネットワークで接続されたコンピュータシステムのピーク演算性能は 430 PFLOPS を超える．TPU v4 では行列積の性能が TPU v3 の 2 倍になり，MLPerf による性能評価結果は 2.7 倍となった．

- Cerebras Wafer Scale Engine:
  Cerebras Wafer Scale Engine[29, 30, 31, 32, 33, 34] は，ディープラーニング向けのシステムに合わせて，GEMM，GEMV に特化したプロセッサである．1.2 兆個を越えるトランジスタが実装された 46,225 mm^2 のシリコンチップを使って，18GB の巨大なオンチップメモリを搭載していることが特徴である．これは，2020 年時点の最大クラスの GPU の 56 倍大きく，オンチップメモリ容量は 3,000 倍を越え，そのメモリバンド幅は 10,000 倍を越える．演算ユニットは 400,000 コアで演算処理性能は 40.6TFLOPS．外部とのインターフェースは 12 本の 100GbE．消費電力は 15KW である．2020 年に開催された HOT CHIPS32 では，第二世代は TSMC 7nm で製造される 2.6 兆個のトランジスタを搭載するチップ上に 85 万個の AI 演算用コアが搭載されるとアナウンスされた [35].
- PFN MN-Core:
  PFN の MN-Core[36, 37] は，行列演算の専用回路を搭載するプロセッサである．行列演算器（MAU）と 4 つのプロセッサエレメント（PE，Processor Element）を合わせたものが，ひとつの行列演算器ブロック（MAB，Matrix Arithmetic Block）である．プロセッサエレメントによって，行列演算器にデータが供給される．各プロセッサエレメントは整数演算器をもっており，ディープラーニングで頻繁に使用する命令もハードウェア実装されている．ピーク演算性能は，32.8 TFLOPS(double), 131 TFLOPS(single), 524 TFLOPS (half) で，消費電力 (設計値) は 500W．HGEMM を用いたベンチマークではボード当たり電力性能として 1.23TFLOPS/W(半精度) を達成．MN-Core を 160 ボード搭載したシステムにおいては，High Performance LINPACK を用いたベンチマークにおいてシステム全体で電力性能 21.10GFLOPS/W, ピーク性能 1.62PFLOPS(倍精度) を達成している．
- Graphcore Intelligence Processing Unit (IPU) :
  Graphcore Intelligence Processing Unit (IPU) [38, 39] は，機械学習アプリケーションを高速化するためのプロセッサである．混合精度浮動小数点数演算を高速に演算する演算器と大量の分散メモリを持つ．複数の IPU は IPU-Link により結合して利用できる．Colossus MK2 IPU processor は，TSMC 7nm で製造された 59.4B トランジスタを搭載する 823mm^2 のチップで，1472 個のコアと，900MB の SRAM を搭載する．メモリバンド幅は 47.5TB/s 相当である．
- Groq Tensor Streaming Processor (TSP):
  Groq Tensor Streaming Processor (TSP)[40] は，コンピュータビジョン，機械学習あるいは，AI ワークロード一般向けのプロセッサである．チップは 14nm プロセスで 268 億トランジスタ，220MB のオンチップメモリを搭載する．コアの動作周波数は 1.25GHz で，INT8 演算性能および FP16 演算性能はそれぞれ 1POPS，250TFLOPS である．メモリ帯域は 80TB/s．ホストプロセッサとのインターフェースは，PCIe Gen 4 x16 で，転送帯域は 31.5 GB/s である．
- SambaNova SN10 RDU:

SambaNova SN10 RDU[41] は，再構成可能なデータフロー計算プロセッサである．TSMC 7nm で製造された 400 億個のトランジスタによるチップで，数百 MB のオンチップメモリをもち，演算性能は数百 TFLOPS である．

- Intel(habana) Gaudi，GOYA:
  GAUDI[42] は，ディープラーニングの学習ワークロード向けのプロセッサで，ベクトルプロセッサである Tensor Processing Core(TPC) が搭載されたチップである．大容量で高速な HBM2 をプロセッサと同じダイに搭載し，また，1Tbps のインタコネクタで複数チップを接続してスケールアウトさせることができる．TPC コアは，GEMM 演算を高速に実行でき，FP32，BF16，INT8-32 のデータ型をサポートする．640 個のチップで構成されるシステムでは，MLPerf の一つである ResNet-50 の学習スループットは 844.8 画像/秒の性能を達成する．GOYA は GAUDI 同様 TPC コアを搭載した推論ワークロード向けのプロセッサチップである [42].
- MDGRAPE-4A SoC:
  MDGRAPE-4A[43, 44] は，分子動力学（MD）シミュレーション専用計算機である．40nm プロセスの面積 16.28mmx16.28mm のチップで，動作クロックは 600 ないし 800MHz．2MB のオンチップメモリを搭載する．プロセッサは，専用パイプライン：64 本，長距離演算ユニット：2 個，汎用演算コア（拡張 RISC-V）：16+1 個，32 ビット固定小数点ベクトル命令等の拡張演算ユニットで構成され，64 キロバイトの局所メモリを持つ．双方向 7.2 ギガバイト/秒 （計 6Gbps 72 レーン）のインターフェースを持ち，3 次元トーラスネットワークで 512 個のプロセッサコアが接続されている．システム全体の演算性能は約 1.3PFLOPS である．

2.1.5.2 FPGA

FPGA は，実行するアプリケーションに応じてロジックの構成を変更させられる半導体である．プログラマビリティのために効率は ASIC に比べると 1/10〜1/100 程度になるが，入出力データのパターンに応じて実行時の最適化ができるというメリットがある．製造プロセスの微細化に伴って，実装可能なロジックの規模が増加し，また動作周波数が向上している．加えて，マルチコアの ARM プロセッサや浮動小数点数演算ユニット，VLIW コア群のようなハードマクロを内蔵したヘテロジニアスプロセッサのような構成となり，ASIC に対する効率の面でのデメリットがカバーされる可能性がある．

ASIC と違って FPGA はチップの製造時点では用途は固定されていない．そのため，FPGA は，その時々のニーズに合わせたアクセラレーションに活用される．主な利用事例には，機械学習の推論や数値計算のほか，多彩な I/O と I/O に直結した演算リソースを活用した通信処理の高速化がある．

以降，高性能演算用の FPGA をリリースしている Xilinx と Intel の製品事例を紹介する．

■Xilinx UltraScale+/Versal　2020 年時点で Xilinx が提供する最新世代の FPGA である Virtex UltraScale+[45] および Versal シリーズ [46] のスペックは表 2.4 の通りである．プロセスは TSMC 7nm で製造され，2.5D 実装によりチップ内に再構成可能な論理回路ブロックの他に，高速トランシーバや高速かつ大容量なメモリを搭載する．Xilinx Versal シリーズは用途に合わせて内蔵ブロックの数や種類の異なる AI コア，プライムコア，プレミアムコアの 3 種類がラインナップされている．

■Intel Stratix/Agilex　2020 年時点で Intel が提供する高性能 FPGA である Stratix10[47] および Agilex シリーズ [48] のスペックは表 2.5 の通りである．Stratix10 は 14nm Tri-Gate プロセスで製造され，Agilex は

表 2.4　Xilinx の提供する高性能演算用 FPGA

	Virtex UltraScale+	Versal AI コア	Versal プライム	Versal プレミアム
ロジックセル数 (K)	862-8938	540-1968	336-2233	1575-7352
LUTs(K)	394-4086	247-900	336-2233	1575-7352
FMAX(MHz)	644-891	680-1150(INT,SIMD), 578-984(COMPLEX), 476-805(FP32)		
BlockRAM(Mb)	25.3-94.5	8-27	5-70	47-174
UltraRAM(Mb)	90.0-360.0	59-130	44-190	127-717
Accelerator RAM(Mb)		0-32		
HBM(GB)	0-16			
DSP 数	2,280-12,288	928-1,968	472-3,984	1,904-14,352
AI エンジン (プロセッサアレイ)		128-400		
シリアルランシーバ数	32-128	8-44	12-48	72-168
最大シリアル転送帯域 (Tbps)		2.9	5.2	18.1
Adaptable Engine Peak Perf(INT8 TOPS)	4.1-38.3	8-29	5-33	23-107
DSP Engine Peak Perf (FP32 TFLOPS)		1.5-3.2	0.8-6.4	3.1-23.1
AI Eingine Peak Perf (INT8 TOPS)		11-33		
AI Eingine Peak Perf (FP32 TFLOPS)		3-8		
DDR4 メモリバンド幅 (GBps)		51.2-102.4	25.6-102.4	76.8-102.4
LPDDR4 メモリバンド幅 (GBps)		68.3-136.5	34.1-136.5	102.4-136.5

10nm プロセスで製造されている．EMIB (Embedded Multi-die Interconnect Bridge (EMIB) により，チップ内に再構成可能な論理回路ブロックの他に，高速トランシーバや高速かつ大容量なメモリ (HBM) を搭載する．

表 2.5　Intel の提供する高性能演算用 FPGA

	Stratix10 GX	Stratix10 MX	Stratix10 DX	Agilex
ロジックエレメント数	378,000- 10,200,000	1,679,000- 2,073,000	1,325,000- 2.753,000	391,996- 2,692,760
ALM レジスタ数	512,640- 13,864,320	2,276,800- 2,810,880	1,797,120- 3,732,480	531,520- 3,651,200
M20K メモリサイズ (Mb)	30-253	120-134	107-229	38-259
eSRAM メモリサイズ (Mb)		94.5	0-94.5	0-36
HBM(GB)		8-16	0-8	
可変精度 DSP ブロック数	648-5,760	3,326-3,960	2,592-5,760	1,150-8,528
18x19 積算器	1,296-10,022	3,326-3,960	5,184-11,520	2,300-17,056
ピーク整数演算性能 (TMACS)	2.6-20.0	13.3-15.8	10.4-23.0	
ピーク単精度浮動小数点数 演算性能 (TFLOPS)	1.0-9.2	5.3-6.3	4.1-9.2	1.7-25.6
ピーク半精度小数点数 演算性能 (TFLOPS)				3.4-25.6
GXE(57.8Gbps) トランシーバ			4-8	
GXT(28.3Gbps) トランシーバ	16-64	16-64		0-16
GX(17.4Gbps) トランシーバ	8-48	8-32		0-8
GXP(16Gbps) トランシーバ			16-76	
PCIe 通信コア (Gen3 x16)	1-4	1-4		
PCIe 通信コア (Gen4 x16)			0-1	1-4

2.1.6 その他

2.1.6.1 Processing-In-Memory (PIM)

1970 年代に提案された Logic in Memory は，1990 年代に VLSI の進歩を背景に Processing in Memory (PIM) として復興され，さらに 2010 年代に Big data のコンテクストから Near Data Processing (NDP) または Near Memory Processor (NMP) と名前を変えながら再復興したアーキテクチャである．これらは記憶したデータの単純な読み書き以外の機能をメモリ側に持たせることを特徴とする．その狙いは以下の通りに大別される．

1. データ移動に伴うエネルギー消費の抑制
2. 演算器〜メモリ間のバンド幅の向上
3. 演算器〜メモリ間の遅延の抑制
4. 演算負荷および発熱の一局集中の抑制

以下に PIM の種類をまとめる

- 実装形態
  PIM を実装形態で分類すると，シングルチップ型（VIRAM[49]，UPMEM[50] のチップ，N3XT[51] 等)，DIMM 型 (DIMMnet-2[52]，UPMEM[50] 等)，HMC 型 (AMC[53] 等) がある．シングルチップ型は実装可能なメモリ容量に問題がある．DIMM 型は着脱が可能であり，複数チップを搭載可能なのでメモリ容量的に最も有利である．HMC 型は TSV で三次元的に積層した DRAM を有する Hybrid Memory Cube(HMC)[54] の Logic base 上にプロセッサを搭載する．このためメモリ容量の観点ではシングルチップ型と DIMM 型の中間になる．
- 記憶媒体
  PIM を記憶媒体で分類すると，SRAM 型，DRAM 型，SCM 型，混載型がある．シングルチップ型ではロジック用プロセスで形成される SRAM が多く容量に問題が出るが，UPMEM[50] のチップは DRAM プロセスで製造されており容量的には SRAM より有利だが，配線層が足りないなどプロセッサの性能に問題が出る．プロセッサはロジック用プロセスで製造して記憶部は DRAM とする DRAM 型 (DIMMnet-2[52]，AMC[53] 等) が最も主流だが，提案レベルでは MRAM や ReRAM といった SCM(Storage Class Memory) を積層するもの (N3XT[51]) や，DRAM に NAND Flash を組み合わせて階層メモリを構成する混載型の提案もある．
- プロセッサ
  PIM をプロセッサの複雑度と種類で分類すると，簡易ロジック型 (Automata processor[55]，Gather-Scatter DRAM[56]，RecNMP[57] 等)，簡易スカラ型 (UPMEM[50])，簡易ベクトル型 (DIMMnet-2[52])，CGRA 型 (NDA，Chameleon 等)，汎用スカラ型 (McIM 等)，汎用ベクトル型 (VIRAM[49]，AMC[53]，TensorDIMM[58] 等) がある．
- プログラマビリティ
  PIM をプロセッサのプログラマビリティで分類すると，特定のアクセスパターンやカーネルを加速するなどの機能固定型 (DIMMnet-2[52]，Gather-Scatter DRAM[56]，RecNMP[57] 等)，CPU と同等クラスのプログラマビリティを示す汎用型がある．

- PIM 間の連携
  PIM 間の連携で PIM を分類すると，ホスト経由型，メッセージ交換型，ccNUMA 型 (McIM[59] 等) に分類できる．同期に関してはホストとなる CPU のメモリ空間上に PIM 上の領域がマップされていることが多いためか，論文等での詳細な記載がないことが多い．

以下にこれまでの PIM 研究の概観をまとめる．

メモリチップ上に簡素なロジックや簡素な CPU を搭載するという流れが存在し，商品化事例が 2 件ある．その起源は古く 1970 年の Logic in memory[60] にあり，DRAM チップ上に形成された Micron 社の Automata processor 内蔵 DRAM[55] や，最近では 2019 年に DRAM チップ上に簡素な CPU を形成した UPMEM[50] が商品化された．これらは容量の観点で DRAM を用いるが，DRAM 用プロセスとロジック用プロセスには配線層数や設計ルールに大きなギャップがあり，DRAM 用プロセスでは CPU が低性能になってしまう上，浮動小数演算能力も低いのでスーパーコンピュータ向きではない．

一方，プロセッサとメモリを 1 チップ化してスーパーコンピュータを作るという流れは 1994 年の EXECUBE[61] に起源があり，その後ベクトルプロセッサを搭載した VIRAM[49] などが試作された．ただし，スーパーコンピュータへの実用化例は見当たらない．当時の LSI の集積度は現在とは比べ物にならないほど低く，それまでの PIM はチップを構成単位としていたため，メモリ容量面で実用的な PIM を作成するのは困難だったことが原因と考えられる．

やがて，メモリモジュール (DIMM) を PIM の構成単位とする DIMMnet-2[52] が試作され，PIM のメモリ容量の制約は緩和した．DIMMnet-2 は高性能クラスタ構成部品で，Gather 等のメモリアクセスのアクセラレータであったが，その前身の DIMMnet-1[62] 同様にネットワークインタフェースだったため，プロトタイプは高価な FPGA を使い，基板が大きすぎるなど商品レベルにはなく，商品化は行われなかった．

その後, DRAM を貫通ビア (TSV) で積層した Hybrid Memory Cube (HMC)[54] が登場するとメモリ容量や実装サイズの問題が解消され, 2011 年に HMC のネットワーク部を搭載する論理回路部であるロジックベースに Gather 機能を搭載する Memory accelerator[63] が提案された．これに引き続き HMC のロジックベースにプロセッサを搭載する提案が続き，HPC 向けには IBM によりベクトルプロセッサを搭載した AMC[53] が試作された．一方，最近発表されてきているグラフ処理を加速する PIM は Tesseract[64]，GraphPIM[65]，GraphH[66] など HMC 型が多い．

2010 年代半ば以来の AI ブームを背景に，データがある場所で処理をする Near Data Processing というキャッチコピーで PIM のリバイバルがなされ，メモリ容量的に大容量を実現でき，発熱密度も抑制できる DIMM 型を中心に現在でも研究発表が各所で活発に行われている．具体的には CAM ベースの DIMM 型としては検索を加速する AC-DIMM[67]，汎用 DRAM ベースの DIMM 型としては等間隔アクセスを加速する Gather-Scatter DRAM[56]，Tensor 処理を加速する TensorDIMM[58]，リコメンデーション処理を加速する RecNMP[57] などがある．DIMM 型ではないが 3D モノリシック集積技術で CNT ベースの CPU と NVM の一体化を目指す N3XT[51] もビッグデータ系アプリの加速を目指している．

最近の流れとして，2019 年には専用 PIM 上で効率動作可能な HPC や AI・ビッグデータ処理の各種専用機能を，多数の DIMM 上の高並列低電力汎用コアによりエミュレーションすることを目指している McIM[59] が提案されている．

以下に，重要と思われる PIM 開発の事例を紹介する．

- DIMMnet-2
  DIMMnet-2[52] は複数のアプリケーション群で汎用 CPU や GPU のネックになりやすい部分を補う

準汎用型の PIM の一例である．世界初の DIMM 型 PIM であり，不連続なメモリアクセスを連続化することで加速する．NAS CG(疎行列ベクトル積)，多次元 FFT(配列の転置)，インメモリデータベース検索，MPI 派生データ型通信 (不連続データ構造間の転送) などで大幅な加速が得られている．PC 上の DDR2-DIMM スロット上で稼働する FPGA によるプロトタイプが存在している．

- McIM

  McIM (Massive cores In Memory side)[59] は FLOPS-to-BYTES コンセプトの一実現形態として最近考案されたアーキテクチャである．その特徴に Near Memory Processing を標榜しており，汎用性を追求したアーキテクチャである．CPU コアをメモリ側へ部分的に移設することにより，メイン CPU パッケージのピン数・サイズ・冷却の限界性を突破しつつ，汎用性や互換性を崩さないようにキャッシュコヒーレント NUMA を構成し，システム全体のメモリバンド幅の絶対値と BF 比，メモリ容量，および汎用 CPU コア数を大幅に向上させる提案である．メモリ側の CPU コアはメインの CPU と同一 ISA を持ちつつも OoO の省略や低周波動作等で低電力化した LITTLE コアを想定し，モバイル分野で成功を収めている big/LITTLE アーキテクチャ向けのソフトウェア的最適化と連動して電力あたりの性能を向上させる．メモリ側の多数の LITTLE コアは，前述のメモリアクセスアクセラレータ DIMMnet-2[52] や Caffe2 の Gather 系関数用アクセラレータ RecNMP[57] などがメモリ側でハード的に実現した機能のエミュレートに用いることを想定している．さらに HPL を代表とする多くの演算ネックのアプリケーションにおいては，OoO の効果や通信能力よりもコア数 (演算器数) の方が性能に効くため，メモリ側の多数の LITTLE コアは効果的と考えられる．

- UPMEM

  UPMEM[50] はごく最近にビッグデータ系アプリの加速を目指して商品化された PIM の事例として注目する．UPMEM 社は，DRAM プロセス（2xnm) を用いて DRAM チップ (DDR4-2400) 内で動作するデータ処理ユニット（DPU）を搭載する PIM-DRAM を開発した． 各 DPU は 64 MB の DRAM にアクセスでき，チップ内には 8 個の DPU がメモリ内のデータを処理する．このメモリと DPU 間のデータ転送量は多いが，メインプロセッサと DRAM (DIMM) 間のデータ転送量を減らして性能を上げるアプローチである．ただし，DPU のプロセスが DRAM のプロセスのため，トランジスタ性能が低く多くの電力が DPU で消費される．この PIM-DRAM で構成された DIMM（UPMEM-DIMM) は，16 個の PIM-DRAM が搭載され，128 個の DPU が動作することになる．同じ条件で UPMEM-DIMM と通常の DIMM を用いて性能比較した場合，UPMEN-DIMM が 20 倍，エネルギー効率は 10 倍とある．また，この UPMEM-DIMM を使うためには，ほとんどのアプリで数百行程度のコード変更が必要となる．UPMEM のアプローチは浮動小数演算性能に力点が置かれていないため，HPC 系ユーザの多いスーパーコンピュータ用途には向かない．専用の DRAM チップはビット単価を増加させる点も留意が必要である．

- N3XT アーキテクチャ

  N3XT[51] は先端的デバイス技術を多面的に投入することにより高い次元の性能を目指す革新的アプローチの事例として注目する．N3XT (Nano-Engineered Computing Systems Technology) は，米国で進められている Rebooting コンピューティングで提案されているアーキテクチャである．最新のテクノロジ（FinFET, CNFET, 新型メモリー等）を用いて最適なプロセスで開発された CPU チップ，メモリーチップを縦方向に TSV を使わず，より小さい ILV(Inter Layer Via) を用いて，超高密度（モノリシック）相互接続する．このアーキテクチャでメモリ-コンピューティング帯域幅が大幅に増加し，メモリーウォールの問題を克服している．例えば，グラフ分析シミュレーションでは，2D プ

ロセッサコアは実行時間の 95.1 %をメモリアクセスに費やし，N3XT は実行時間のわずか 2.1 %をメモリアクセスに費やしている．

2.2 システムアーキテクチャ（システムハードウェア）の技術動向

2.2.1 ノードアーキテクチャ（計算ノードのハードウェア構成）

2.2.1.1 メニーコア CPU 型

本節では，2020 年 6 月現在のメニーコア CPU 型のノード構成とシステム構成について，実システムの例を挙げて説明する．表 2.6 に TOP500 のランキングの上位 50 位までのメニーコアシステムの 1 ノードの構成を示す．表において [1]-[50] は Top500 ランキングにおける該当順位のマシンを示す．メニーコア CPU 型のスパコンの大部分は Intel 社の Xeon の 2 ソケット機であることがわかる．2008 年にリリースされた Nehalem 世代から 2017 年にリリースされた Skylake 世代まで，スパコンでのシェアは比較的大きい．2019 年には AMD EPYC が複数チップを一つのパッケージに搭載するチップレット構成を取り，非常に高いコスト性能を達成し，シェアを伸ばしている．2016 年には Intel 社の Xeon Phi がリリースされ，高いコスト性能から多数のシステムで採用されたが，Xeon に対する優位性がなかったためか，製品としては終息している．世界一位となるようなシステムでは独自のプロセッサが開発されることがある．Sunway TaifuLight の SW26010 CPU や富岳の A64FX CPU が該当する．

表 2.6　Top50 のメニーコアスパコンのノード構成 (2020 年 6 月時点)

	Intel CPU	AMD CPU	その他
ソケット数	1[16,17,18,20,21,34], 2[8,13,28,30,32,37,38,40,41,42, 43,44,45,46,48,49], 1,2 混在 [11]	2[29,33,47]	1[1, 4, 36]
コア数	96[48],68[11,16,17,18,20,21], 64[34],56[8],48[13,37,38,40,41,42], 40[28,30,44,46],36[32],32[11,45],24[43,49], 20[39]	128[29,33,47],	260[4], 48[1,36]

2.2.1.2 メニーコア CPU & GPU 混載型

本節ではメニーコア CPU と GPGPU 混載型のシステムについて総覧する．最初に NVIDIA 社の GPU についてまとめる．次にシステムとしての CPU，GPU の構成をまとめる．

結論から言えば，2020 年現在では CPU の ISA は x86 ないし Power であり，これが 1〜2 個，GPU は V100 が標準で P100 や A100 も散見される．CPU ソケット当たり平均で 2 つの GPU を搭載したシステムが主流となっている．2020 年 6 月の Top500 にて，およそ 1/4 が (GPGPU) 混載型のシステムである．その中でも NVIDIA Volta 型のシェアがかなりの割合を締め，SMX2 を含めるとシステム数で約 22%，Pascal, Kepler 型はそれぞれ 3.6%，1.6% のシェアとなっている．Volta アーキテクチャは 2017 年 11 月より Top500 で見られるようになり，以後急速にシェアを拡大し，現在では 110 システムで搭載されている．Pascal はそれよりも早く，2016 年 11 月からの登場で，2018 年 11 月に 64 システムのピークとなった後，2020 年 6 月では 18 システムまで下落している．Pascal の多くが Volta に置き換わったものと推測される．

近年の NVIDIA の GPGPU の主要なアーキテクチャは，複数の CUDA Core から成る SM(Streaming Multiprocessor) を多数有することによって，高い演算性能を得ている．CUDA Core は 1 つの積和演算器を備えており，この積和演算器が実際の演算を実行する．SM 内の CUDA Core は同一の命令を実行するため，一命令で複数の CUDA Core を制御する構造になっている．

Kepler アーキテクチャ (K40) のリリースは 2012 年 Q2, チップ性能としては，倍精度で 1.43TFLOPS, 単精度で 4.29TFLOPS である．SM あたりの CUDA Core 数は 64bit として 64 であり，この SM をチップ全体で 15 有する．従ってチップ当たり 960 の CUDA Core を有している．一方で 32bit 演算の場合は，SM あたりの CUDA Core 数は 32bit として 192 と倍精度の 3 倍になり，チップ当たりの CUDA Core 数は 2,880 になる．クロック周波数は Boost 時に 875MHz であるため，Boost 時の理論演算性能は 1,680GFLOPS (64bit)，5,040GLOPS (32bit) の性能となる．レジスタは SM あたり 256KB でチップ当たり 3,840KB, L2 キャッシュはチップあたり 1,536KB である．メモリ容量は 12GB で消費電力は 235W である．

Pascal アーキテクチャ (P100) の主要な特徴は次の通りである．リリースは 2016 年 Q2, プロセスは 16nm である．チップ性能としては，倍精度で 5.3TFLOPS, 単精度で 10.6TFLOPS, 半精度で 21.2TFLOPS である．各チップには 4 つの NVLink チャネルを有しており，1 ノードに 4 つの P100 が搭載されている場合，1 つの GPU に着目すると，他の 3 つの GPU のうち 1 つとは 2 リンクで接続し，残りの 2 つとは 1 リンクで接続される．この NVLink は PCIe とは独立した P100 間の通信チャネルとして動作する．SM あたりの CUDA Core 数は 64bit として 32 であり，この SM をチップ全体で 56 有する．従ってチップ当たり 1,792 の CUDA Core を有している．クロック周波数は標準で 1,328MHz, Boost 時に 1,480MHz であるため，Boost 時の理論演算性能 5,304GFLOPS (64bit) の性能となる．レジスタは SM あたり 256KB なのでチップ当たり 14,336KB, L2 キャッシュはチップあたり 4,096KB である．メモリ容量は 16GB で消費電力は 300W である．Kepler と Pascal を比べると，倍精度演算性能は 3 倍強向上しているが，単精度なら 2 倍程度である．CUDA Core 数は 2 倍弱，クロック周波数 1.5 倍となっている．

もっとも最近の Volta (V100) は 2017 年 Q2 リリースである．チップ性能としては，倍精度で 7.8TFLOPS, 単精度で 15.7TFLOPS, 半精度で 31.4TFLOPS である．Volta は各チップに 6 つの NVLink チャネルを有しており，各ノードで 4 つの GPU を完全結合できる．SM あたりの CUDA Core 数は 64bit として 32 であり，この SM をチップ全体で 80 有する．従ってチップ当たり 2,560 の CUDA Core を有している．クロック周波数は標準で 1,328MHz, Boost 時に 1,530MHz であるため，Boost 時の理論演算性能は 7,834GFLOPS (64bit) の性能となる．SM あたりのレジスタ容量は 256KB と Pascal と同じである．チップ当たり 20,480KB, L2 キャッシュはチップあたり 6,144KB である．メモリ容量は 16GB で消費電力は 300W である．

Pascal から Volta への演算性能向上は，主として SM 数の増加とクロック周波数の向上によってもたらされており，その性能向上率は 1.5 倍弱である．その他に Volta では行列演算用に Tensor Core が SM 毎に 8 個有している．チップ当たり 640 の Tensor Core があり，それぞれ 64 の積和演算が実行可能なので，Tensor Core によって提供される演算性能は 121,242GFLOPS である．

2020 年 5 月に次世代の Ampere アーキテクチャのチップがアナウンスされた．本 GPU を用いたシステムは 2020 年 6 月現在稼働報告がないが，Ampere チップについて概略を紹介する．Volta に比べて SM 数が 80 から 108 に増加している．SM 内の CUDA Core 数は同じく 32 なので，SM 数の増加が演算性能の増加になっている．一方でクロック周波数がブースト時 1,530MHz から 1,410MHz にダウンしている．従って CUDA Core での理論演算性能は 9,746GFLOPS と計算できる．Ampere アーキテクチャでは新しい Tensor Core が搭載されている．Volta が 125TFLOPS だったのに対し，理論的には 624TFLOPS の演算が可能となっている．この Tensor Core は Volta に比べて精度を縮小するなど様々なモードでの動作が可能になって

いる模様である．

NVIDIA 社のチップは世代ごとにアーキテクチャもかなり変わるため，一概に性能向上率を求めることは困難である．しかしながら，2012 年の Kepler が 1.7TFLOPS, 2017 年の Volta が 7.5TFLOPS であることを考えると，年率でおよそ 1.38 倍の性能向上と考えることができる．最近では通常の演算性能に比べて Tensor Core の性能向上が著しい．Tensor Core の性能向上率を考えると，2017 年に 125TFLOPS, 2020 年に 624TFLOPS なので，年率で 1.7 倍程度の性能向上を達成していると言える．

NVIDIA 社の GPGPU アクセラレータの方向性としては，レジスタやメモリバンド幅がボトルネックとなっていることを受けて，Tensor Core など演算量の割にメモリアクセスが少ない演算要素に資源を割く方向で進展していると捉えることができる．この方向性は，メモリアクセスに関する大幅なブレークスルーが無ければ当面継続すると予想することができる．一方で，現在は行列演算用として Tensor Core が実装されているが，Tensor Core 以外にも市場で良く利用されるアプリケーションが出てくれば，それに向けたメモリアクセスが少ない演算要素を搭載する可能性は考えられる．

表 2.7 に GPU の年ごとの性能の変化をまとめる．基本的には NVIDIA の GPU しかないため，NVIDIA の製品の変化を示している．括弧内の数値は，行列積に限定した場合の性能値である．また性能は公表されている資料から得ているが，多くはブースト時の性能である．

GPU 間接続として，Pascal 以降は NVLink が利用できるようになっており，これを用いたシステムも多く構築されている．

表 2.7　代表的な GPGPU の諸元

年代	GPU の世代 (型番)	倍精度 [TFLOPS]	単精度 [TFLOPS]	半精度 [TFLOPS]	メモリバンド幅 [GB/s]	メモリ容量 [GB]	消費電力 [W]
2011.3Q	Fermi (M2070)	0.515	1.03	-	150	6	225
2012.4Q	Kepler (K20)	3.52	1.17		208	5	225
	Maxwell (HPC 製品なし)						
2016.2Q	Pascal (P100)	5.3	10.6	21.2	720	16	300
2017.2Q	Volta (V100)	7.8	15.7	(125)	900	16/32	300
2020.2Q	Ampere (A100)	9.7	19.5	312	1555	40	400
2020.3Q	Turing (Quadro RTX 5000)	(0.348)	11.2	22.3	448	16	230
2015.2Q	参考 E5-2699v4	0.634					145
2019.3	参考 EPYC 7742	3.482			204.8		225

表 2.8 に主要な GPGPU 混載システムの構成について俯瞰する．Top500 の上位にある GPGPU 混載システムを示している．システムによっては複数の異なるアーキテクチャのノードを統合している場合があるので，ここではシステム中で主要な GPGPU 混載ノードについて記載する．

Summit@ORNL は 2018 年稼働開始の GPU 混載型のシステムである．本システムの特徴は IBM Power9

表 2.8　主要な GPGPU 混載システムの構成

サイト名	Summit @ORNL[4]	Sierra @LLNL [5]	Frontera @Univ. of Texas (GPU nodes) [6,7]	Piz Daint @CSCS	ABCI @産総研 [9]	Selene@ NVIDIA [10]	HPC5 @Eni's Green Data Center [11]
CPU	IBM Power9	IBM Power9	Xeon E5-2620 v4	Xeon E5-2690v3	Xeon 6148	AMD EPYC 7742	Intel Xeon Gold 6252
ノード当たり CPU ソケット数	2	1	2	1	2	2	2
ノード当たりメモリ容量 (CPU)	512GB	256GB	128GB	64GB	384GB	384GB	192GB
GPGPU	NVIDIA V100s	NVIDIA V100	NVIDIA Quadro RTX5000	NVIDIA V100	NVIDIA A100 SXM2	NVIDIA V100	NVIDIA
ノード当たり GPGPU 数	6	4	4	1	4	8	4
GPGPU 演算性能	42T FLOPS	28T FLOPS	11.2TFLOPS(FP32)	4.7T FLOPS	(32.2T FLOPS)	77.6T FLOPS	31.3T FLOPS
ノード当たりメモリ容量 (GPGPU)	16GB×6	16GB×4	16GB×4	16GB	16GB×4		
GPU 間結合	NVLink (3GPU 間)				NVLink2 ×2		
GPU-CPU 間結合	NVLink				PCIe Gen3×16	PCIe Gen4×16	
ノード数	4,608	4,320	90	5,320	1,088	280	1,820
システム性能 (FLOPS)	194P FLOPS	126P FLOPS	4P FLOPS	25.326P FLOPS	37.2P FLOPS	23.0 FLOPS	70 FLOPS
相互結合網テクノロジ	Mellanox EDR 100G InfiniBand	Infini Band EDR	InfiniBand HDR (200Gb)	Dragonfly network topology (100Gb)	Infiniband EDR (Mellanox SV 7890x229)	Infiniband HDR	Mellanox HDR InfiniBand
稼働年	2018	2018	2019	2016	2018	2020	2020

プロセッサを 2 つ搭載していることと，ノード当たり 6 つと多少多めの V100 GPU を搭載していることである．Sierra@LLNL もまた IBM Power9 プロセッサを搭載しているが，ノード当たりの搭載数は 1 つであり，Summit に比べて半分である．GPU もまた同じ V100 であるが，搭載数は 4 つである．Summit に比べると，ノード数は概ね同じ規模であるが，それぞれのノードの規模が少し小さめである．Frontera@Univ. of Texas は複数のタイプのノードから構成されているが，本節では GPU ノードに焦点を絞る．本システムのうち GPU ノードのノード数は 90 と他のシステムに比べてかなり小さめである．加えて搭載 GPU が Quadro RTX 5000 × 4 であり，この点で他のシステムと大きく異なる．RTX はグラフィックス重視の GPU であり，単精度では 11.2TFLOPS 十分な演算性能を有するが，倍精度では 0.348TFLOPS と極端に性能が低下する．従って本システムでの GPU 利用はもっぱら単精度での演算に限られると推測される．

ここまでは米国のシステムであるが，Piz Daint@CSCS はスイス国立スーパーコンピューティングセンターのシステムである．CPU が Intel Xeon に GPU として NVIDIA P100 を 1 つ搭載したノードを 5 千ノードと，他のシステムに比べてノードが小規模でノード数が多いシステムとなっている．ABCI@産総研は同様に Intel Xeon と NVIDIA の GPU を用いているが，V100 SXM2 になっていることと，ノードの規模として CPU が 2，GPU が 4 と標準的な大きさのノードとなっている．

Selene@NVIDIA と HPC5@Eni's Green Data Center は民間のシステムである．Selene は NVIDIA の構築だけあり，最新の A100 GPU を 8 個と GPU を多く搭載したシステムとなっている．CPU も 2020 年度稼働ということで，AMD の EPYC プロセッサを採用している．HPC5 はイタリアのシステムであるが，CPU は Xeon が 2 つ, GPU は V100 が 4 つと標準的なシステムとなっている．

これらのシステムを俯瞰すると，2020 年段階ではノード当たりの CPU 数は 1〜2 個，CPU(ソケット) 当たりのメモリは 64GB〜192GB であると言える．GPU に関しては，利用 GPU は P100, V100, A100 であり，Top500 の上位システムでは V100 を搭載するシステムが多いことがうかがえる．GPU 数は CPU ソケット数に対して 1〜3，平均として 2 つであることが分かる．

今後の動向の予測について，個々の構成要素，つまりメモリ搭載量や CPU コア数，GPU 性能は年々進歩すると思われる．その半面，アーキテクチャ面では概ね 1CPU ソケットにつき 2GPU という形態は GPU システムが現れた当初からあまり大きな変化が無く，これは CPU-GPU 間のバンド幅によって制限されていると考えられるため，今後も大きな変化は無いと予想される．近年では NVLink が利用可能になってきているので，NVLink の性能を最大限引き出すようなアーキテクチャ上，ソフトウェア上のテクノロジが普及する場合は，CPU 当たりの GPU 数が増加する可能性もあり得る．

また GPU については現在はもっぱら NVIDIA 社の GPU ばかりであるが，他に数値演算として高い性能を有し，プログラミング環境が整っているアーキテクチャは殆ど見受けられないでの，この傾向はまだ数年は継続すると考えられる．一方で全く新しいアーキテクチャの GPU を容易なプログラミング環境で利用可能になれば，市場の流れは一気に変わる可能性も秘めている．

nVIDA 社の GPU は，倍精度浮動小数点演算性能の向上は少しずつであるが，その半面 Tensor コアの性能は飛躍的に向上している．Tensor コアでの演算はメモリやレジスタへの入出力が大きく制約されている問題があり，汎用のアプリケーションには利用が困難である．しかしながら，Tensor コアの特性をうまく利用する新しいアルゴリズムなどが開発されれば，この利用は一気に進む可能性があり，その場合は GPU を利用した実質的な数値演算性能は大きく向上するものと期待できる．

2.2.1.3 その他のシステムアーキテクチャ

以下，メニーコア CPU 型，メニーコア CPU & GPU 混載型以外の特徴を持つシステムのいくつかを紹介する．

- Cygnus:
  Cygnus は日本電気株式会社 (NEC) により，筑波大学計算科学研究センターに 2019 年 3 月末に導入され，本書執筆時において日本国内で唯一，FPGA を本格的に使用するスパコンである．Cygnus は，GPU 搭載ノードである「Deneb ノード」と，GPU・FPGA 搭載ノードである「Albireo ノード」の 2 種類の計算ノードからなるクラスタ型スパコンである．どちらのタイプのノードにも Intel Xeon Gold 6126 を 2 基と NVIDIA Tesla V100 を 4 基搭載し，Albireo ノードには，これらに加えて Intel Stratix10 FPGA を実装した BittWare 520N を 2 基搭載する．FPGA は，OpenCL によって利用可能であり，FPGA の演算性能は主に単精度浮動小数点で供給される．倍精度浮動小数点の演算性能は CPU と GPU によって供給され，Cygnus の倍精度浮動小数点理論ピーク演算性能は 2.43 PFLOPS である．表 2.9 に本システムのハードウェア構成を示す．

表 2.9　Cygnus スーパーコンピュータのハードウェア構成

	Deneb ノード	Albireo ノード
ノード数	46 + 3 (増設ノード)	32
CPU	Intel Xeon Gold 6126 (2.6GHz/12core) x 2	
メインメモリ	192 GiB (16 GiB DDR4-2666 ECC RDIMM x 12)	
GPU	NVIDIA Tesla V100 (32 GiB HBM2 PCIe Gen3 x16) x 4)	
FPGA	N/A	BittWare 520N x 2
ネットワーク	InfiniBand HDR100 x 4	

- Noctua:
  ドイツ Paderborn 大学で運用中のスーパーコンピュータであり，2018 年 9 月に稼働を開始している [68]．CPU のみの計算ノードと，CPU に加えて FPGA カードを搭載した FPGA ノードの 2 種類のノードから構成されており，相互接続した FPGA による高性能計算に関する研究が活発に行われている．システムには，Intel Stratix10 GX2800 FPGA が搭載された PCIe カードが 32 枚搭載されている（FPGA の諸元については 2.1.5 節を参照のこと）．FPGA は，OpenCL によりプログラム可能である．本システムの特徴として，光回線スイッチによる FPGA カードの相互接続が挙げられる．FPGA カードには 100Gbps の QSFP+ コネクタが 4 つ搭載されていおり，これが CALIENT S320 Optical Circuit Switch (OCS)[69] に接続されている．OCS の仕様のために，OCS 接続の帯域は 40Gbps である．S320 OCS は MEMS による光回線交換が可能であり，FPGA クラスタでアプリケーションを実行する場合には，まず OCS 上に FPGA 間の回線交換を設定した上で，それを用いて FPGA 間通信を行うアプリケーションを実行する．OCS は切り替え可能であるがそれにはある程度時間がかかるため，細粒度の通信ではなく，ある程度の通信単位で回線交換の切り替えを行うことを想定している．表 2.10 に Noctua スーパーコンピュータのハードウェア構成を示す．
- SX-Aurora TSUBASA:

表 2.10　Noctua スーパーコンピュータのハードウェア構成

システム	Cray CS500
プロセッサコア数	10,880
総メモリ容量	51 TiB
ピーク FP 性能	535 TFLOPS
計算ノード（256 ノード）	2x Intel Xeon Gold Skylake 6148, 2.4GHz, 40 コア, 192 GiB
FPGA ノード（16 ノード）	2x Intel Xeon Gold Skylake 6148, 2.4GHz, 40 コア, 192 GiB 2x Bitware 520N cards, 32 GiB DDR4 メモリ/ card
CPU ネットワーク	Intel Omni Path 100 Gbps, 1:1.4 blocking factor
FPGA ネットワーク	Application-specific interconnect - 4x40 Gbps links per FPGA - Connected via CALIENT S320 Optical Circuit Switch (OCS) - Configurable point-to-point connections to any other FPGA - Accessible as serial channels from OpenCL BSP
ストレージ	Cray ClusterStor L300N with NXD flash accelerator Lustre File System with 720 TB capacity

SX-Aurora TUSBASA は NEC のベクトル型スーパーコンピュータ SX シリーズの流れを汲む製品として 2018 年から出荷開始された．PCI express カードの形状にベクトルプロセッサと主記憶 (HBM) を搭載した Vector Engine を，Vector Host と呼ばれる標準的な x86 サーバに PCIe カードとして接続して構成される．Vector Engine カードは GPU や FPGA のようなアクセラレータと同じように接続されているが，ホストで実行されるプログラムの一部処理を実行（アクセラレート）するのではなく，コンパイルされた実行ファイルを丸ごと Vector Engine カード上で実行する（プログラミングとしてもアクセラレータ向けの書き換えをしなくてもよい）点に特徴がある．マルチノードシステムでのノード間通信もあくまで Vector Engine カードが主体となり，ホスト側プロセスを介さずに実行できる．ベクトルプロセッサの主記憶は HBM で構成され，1 プロセッサに 6HBM を接続しており，48GB のメモリ容量と最大 1.5TB/s のメモリバンド幅を提供する．B/F は最大で 0.62（モデル依存）であり，前機種の SX-ACE が 1.0 であったのと比較すると小さいが，他のプロセッサと比較すると大きな値を維持している．

現時点で NEC はベクトル型プロセッサを開発する唯一のベンダーであり，既存ユーザの求めるベクトルプロセッサの特徴（性能特性）を維持しつつ，汎用の x86 サーバやその周辺機器を利用することで，開発費や導入コストを低減すること，多くのユーザが使い慣れた Linux サーバのプログラミング環境を利用できることを訴求している．小規模（最小構成はタワーサーバに Vector Engine カード 1 枚だけを搭載したもの）から大規模（2020 年 7 月時点で国内の最大構成は核融合研の雷神システムであり，Vector Engine カード 4320 枚で構成される）まで幅広いシステムが提供され，NEC の発表資料によると 2019 年末で国内外のユーザ数は 100 以上，販売された Vector Engine カード枚数は 10000 枚以上となっている．Vector Engine カードの仕様と，雷神システムの仕様を表 2.11，2.12 に示す．

- Cerebras CS-1:
  2.1.5.1 で述べられた Cerebras Wafer Scale Engine を搭載したディープラーニング専用システムであ

表 2.11　Vector Engine カード

カードタイプ	Type10AE	Type20B
CPU コア数	8	
ベクトル理論演算性能	2.43TFlops	2.45TFlops
メモリ容量	48GiB（HBM2）	
メモリバンド幅	1.35TB/s	1.53TB/s

表 2.12　雷神システム

総 Vector Engine カード枚数	4320 枚（Type10AE）
総 Vector Host 台数	540 台
総理論演算性能（ベクトル演算部分のみ）	10.5PFlops
総メモリ容量（ベクトル演算部分のみ）	202TiB
ノード間転送性能	InfiniBand HDR200 x2 /Vector Host あたり

り，2019 年 11 月に米国デンバーにて開催された SC19 で発表された．ピッツバーグ スーパーコンピューティング センター (PSC) は 2 台の CS-1 の導入を決定しており，それらと Hewlett Packard Enterprise（HPE）の Superdome Flex サーバーと連携して動作する Neocortex システムを 2020 年後半に米国の研究者が利用できるように整備中である．また，アルゴンヌ国立研究所は，がんや外傷性脳損傷等の研究の推進に活用していた Cerebras CS-1 を，新型コロナウイルスのワクチン開発に転用したことを発表した．国内でも，2019 年 12 月に東京エレクトロンデバイス株式会社が Cerebras CS-1 の代理店契約を正式に締結し，受注を開始している．

Cerebras CS-1 は，ディープラーニング専用に設計された一枚の大型半導体チップ (Cerebras Wafer Scale Engine) 上で演算を行うため，機器同士の通信遅延も発生しないというのが，GPU ベースの HPC システムと比較しての主な利点である．なお，Cerebras CS-1 は AI アクセラレータとしての位置づけであり，スタンドアロンシステムではない．そのため，Cerebras CS-1 を利用するためには，制御サーバと CS-1 を Ethernet で接続する必要がある．表 2.13 に Cerebras CS-1 の仕様を示す．

表 2.13　Cerebras CS-1 のスペック

疎線形代数演算コア	400,000
オンチップメモリ	18 GB SRAM
メモリ帯域	9.6 PB/s
コア間帯域	100 Pb/s
システム I/O	1200 Gb/s: 12 x 100 Gig Ethernet
最大消費電力	20KW@208-240VAC (16A)
プロセス	TSMC 16nm FFC
冷却方式	内部閉循環冷却液を用いた空冷システム
サイズ	15 Rack Units (26.25 inches)

- GraphCore IPU-Machine M2000:

表 2.14　Top30 スパコンの結合網の構成 (2019 年 11 月時点)

NW Topology	# of Switch port	Link BW,Node BW	Top500 Ranking
Fat Tree	36(Tier2),648(Tier1)[1,2,10] 32(Tier2),576(Tier1)[4] 40(Tier2),800(Tier1)[5] 36(Tier2,3),648(Tier1)[8] 48(Tier2),768(Tier1) [14,15,19,23,30] 648(Tier1)[20]	100Gbps,200Gbps[1,2,8,11] 112Gbps,112Gbps[4] 200Gbps,100Gbps[5] 100Gbps,100Gbps [10,14,15,18,19,30] 100Gbps,800Gbps[20,26,29] 100Gbps,400Gbps[23]	19-Machines (1,2,4,5,8-11 14,15,17-20 23,24,26,29,39)
Dragonfly (direct)	48	37.5-42Gbps,84Gbps	5-Machines (6,7,13,27,28)
5-D Torus (direct)	10	16Gbps,Integrated	2-Machines (12,22)
Others			4-Machines (3,16,21,25)

2.1.5.1 で述べられた Graphcore Intelligence Processing Unit (Colossus Mk2 GC200) を 4 台搭載した 1U ラックマウント型計算サーバである．演算性能は 1 PetaFlops (AI 用途) である．また，180TB/sec の帯域をサポートする Exchange Memory (450GB) と呼ばれるメモリシステムを有している．この M2000 を 16 台挿せるラック型システムが IPU-POD64 であり，それぞれの M2000 は 2.8Tbps の帯域を有する IPU-Fabric で相互接続される．また，Graphcore 社は，M2000 や IPU-POD64 に展開される AI ワークロードを効率的に実行するためのランタイムを構築するためのプログラミングフレームワーク Poplar も提供しており，Poplar は，TensorFlow，PyTorch，ONNX，PaddlePaddle などの標準的な ML フレームワークと，Open BMC，Redfish，Docker コンテナ，Slurm や Kubernetes によるオーケストレーションなどの業界標準のコンバージド・インフラストラクチャツールをサポートしている．

2.2.2　インターコネクト

スパコンのインターコネクトは，チップ内，ボード内，ボード間に大別できる．ただし，チップ内/ボード内技術に関しては前章と重複するため，本節ではボード間結合網 (インターコネクト) のアーキテクチャに関する動向を述べる．

表 2.14 に Top500(http://www.top500.org)(2019 年 11 月時点) における上位 30 台の結合網の構成を示す．表において [1]-[30] は Top500 ランキングにおける該当順位のマシンを示す．表において 1 番目の項目はネットワークトポロジと，直接網 (direct)/間接網 (indirect) の別を表している．Others には結合網の構成やリンク帯域などの情報が不明瞭なマシンも含む．2 番目の項目はスイッチのポート数を表している．ただし，直接網と多くの Tier2/3 スイッチは 1 台のスイッチ ASIC で構成されているが，Tier1(director) スイッチはポート数が多いため，複数のスイッチ ASIC を組み合わせることで構成されている点に注意が必要である．そのため Tier1 スイッチは 1 台のラックを占有する場合が多い．スパコンのボード間結合網の構成は，コモディティ技術に大きく依存する．例えば，リンク帯域は，スパコン導入時に入手できるコモディティの高速シリアル I/O(2.1.4.1 節参照) により定まる．そのため，現時点のスパコンのリンク帯域は高々 100Gbps〜200Gbps である．Dragonfly，5-D Torus のリンク帯域が小さい．この理由は 2010 年代前半にリリースされている技術を使い続けているためである．リンクデバイスは，ラック内 (数 m 以内) はパッシブ電気ケーブル，ラック間はアクティブオプティカルケーブル (AOC: Active Optical Cable) を用いることが多い．安価かつ高信頼

である AOC が利用可能となったことにより，電気ケーブルを主に利用していた時代と比べて，マシンルーム内の配線長に関する制約が緩くなっているといえる．

表 2.14 のノードバンド幅は，計算ノードとスイッチ間の通信帯域を表している．30 位までのスパコンの通信帯域の特徴として，複数本のリンクを用いて通信帯域を大きくしている場合が多いことが挙げられる．ただし，複数本のリンクを，Link Aggregation (LAG) ではなく，異なるスイッチへ接続し，異なる経路を採用するように設計されている場合がままある．表には記述していないが，計算ノード内の I/O は PCIe を用いていることが多い．ただし，すべてのスイッチにノードが接続される直接網では，京，富岳，BlueGene/L, BlueGene/Q のようにルータとプロセッサを統合，あるいは密に接続する専用の実装を行っている場合がある．

ネットワークのスイッチに関しては，マイクロアーキテクチャの改良 [70] やクロスバのスループット向上により，スイッチ遅延を 60 ナノ秒〜100 ナノ秒に維持しつつ，次数が大きくなる，つまり高次元化の傾向にある．ネットワークスイッチの高次元化と AOC の採用が進んだことで，ラック間長距離リンクを多数用いる Dragonfly や Fat ツリーの利用が台頭しているといえる．

ネットワークの高次元化が進むほど 1) n 分割バンド幅向上 ($n > 1$)， 2) パケット転送の最大，平均ホップ数の削減，3) 選択可能な経路数の増加による輻輳回避性能の向上が顕著になる．1)，2) の側面に関しては，(配線制約がないと仮定した場合の) 各スパコンの次数とスイッチ台数に対して，理想のネットワークトポロジのカタログが既に公開されている [71]．特に 2) に関しては，理論下限値も明らかになっている．

結合網の性能指標は n 分割バンド幅，ホップ数の他にも多岐に渡る．これはメモリ帯域との相対性能比，計算ノード数の増加などのシステム構成の変更，対象アプリケーションなどにより，異なるボトルネックが生じるためである．そのため，優先して改善すべき性能指標は，都度異なる可能性がある点に注意が必要である．例えば，特定の MPI 操作や集合通信加速や，ジョブの仮想ネットワークトポロジから実ネットワークトポロジへのマッピングの効率 (指標としては Topology Embedding)，ハードエラー/ソフトエラー耐故障性能が挙げられる．

一般的に，結合網内の重要な機能は，ネットワークトポロジ (どのような経路が可能か？)，ルーティング (どの経路がパケットに有効か？)，調停 (パケットが，その経路をいつ利用できるか？), スイッチング (どのように経路をパケットに割り当てるか？) の 4 点である．しかし，ネットワークトポロジを除く 3 点は，デッドロックフリー最短経路ルーティング，カットスルースイッチングが用いられるなど，今後も技術傾向は変わらないと考えられるため，表 2.14 では省いている．なお，結合網の分類では，OmniPath，イーサネット，InfiniBand，Gen-Z，Fibre Channel などの独立型 (PC クラスタ型)，Tofu(京, 富岳) などマシン特化型結合網の違いが議論されることがある．ただし，両者はコモディティ技術から派生している．そのため，ロードマップ上重要となる性能指標であるスループット，電力性能 (pJ/bit) に関して，両者の違いは小さいと考えられる．そのため，ここでは独立型，マシン特化型結合網の比較，検討は行わない．

2.2.2.1 目的特化型結合網

特定の並列計算を目的とした専用マシンでは，上記のトレンドとは異なる結合網が開発されている．分子動力学計算専用計算機である Anton-2 では，同シミュレーションの高速化のために，通信遅延が特に小さい結合網を採用している．具体的には，チップ内リングネットワークとチップ間 3 次元トーラスネットワークを統合した特殊な構造をもつ [72]．隣接ノード間通信 99 ナノ秒であり，その内，2 台のルータを含むネットワーク部の遅延は 38 ナノ秒である．これは，最速の InfiniBand スイッチ遅延が 60 ナノ秒程度であることを考えると，極めて小さいといえる．

同専用のアクセラレータである MDGRAPE-4 SoC は汎用のプロセッサコアを含む構成をとる．当時，汎用の計算ノード内にアクセラレータを有する従来の並列システムでは，同計算において，アクセラレータとホストとなる汎用計算ノード間の通信帯域，計算ノード間の通信遅延が性能ボトルネックとなっていた．そこで，MDGRAPE-4 は SoC 間を 3 次元トーラス構造で接接続することにより，この問題を回避し，低遅延高バンド幅接続を達成している [73].

例に挙げたこれら二つのマシンの結合網は，半導体の集積度の向上に頼らずに，古典的な計算ノードの構造的な通信性能問題 (汎用プロセッサ-アクセラレータ，汎用プロセッサ間) を，アーキテクチャの工夫により解決し，高バンド幅かつ低通信遅延を達成している点が特筆される．

2.2.2.2 その他の動向と今後の展望

結合網のアーキテクチャにおいて，2.1.4.4 で述べた光伝送技術の動向が今後の技術開発の大きな鍵を握る．つまり，これからも結合網はコモディティ技術，特にマーケットが大きいイーサネットに大きく影響を受けるであろう．コモディティ技術が AOC を使い続ける場合，スパコンのネットワーク構成は現状のトレンドのまま進む可能性が高い．ただし，(1) Active Electrical Cable (AEC) を部分的に用いる場合，(2) 光伝送のボーレイトの向上により信頼性を担保できる AOC の伝送距離が短くなる場合，その制約条件を満たすレイアウタブルなネットワークトポロジを中心としたアーキテクチャを採用することになる．

ネットワークスイッチに関して，今後，Co-packaged Optics (CPO) の成熟度が鍵を握る．データセンターの ToR (Top of Rack) 向けハイパースケール (超高次元) イーサネットスイッチ ASIC を対象にして，SerDes 電気処理のオーバヘッド削除，消費電力削減，パッケージ密度を向上させることが期待されている．

CPO は，OCP(Open Compute Project) Summit2018 において，51.2Tbps (128x400G) スイッチを想定したモックアップが示されたことで，注目を浴びた [74]．実際に，ブロードコム社は 2017 年末にハイパースケールイーサネットスイッチとして，Tomahawk 3 イーサネットスイッチ ASIC(12.8Tbps) を，2019 年末に同 4 ASIC(25.6Tbps) をニュースリリースしており，2020 年代前半に 51.2Tbps 級スイッチ ASIC を実現することが想定されている．つまり，40Tbps 級スイッチ ASIC まではオンボード optics，51.2Tbps 級スイッチ ASIC では CPO の利用を産業界は前提としている．

インテル社は，2020 年に 1.6Tbps シリコンフォトニクス (4x400GBase-DR4) を Barefoot 社 Tfino 2 スイッチ (12.8Tbps) に適用する計画を公表している．Rockley Photonics 社も CPO を用いたハイパースケールスイッチを提示 [75] し，他にも，RANOVUS 社 Odin シリコンフォトニクスエンジンを 51.2Tbps スイッチにおいて利用することを見据えている [76]．これらのスイッチング ASIC には，シリコンフォトニクスによる光ポートのみならず，従来の電気ポートの両方が接続可能である．よって，これらの技術で構成されたスイッチは ToR のみならず，HPC 分野においても柔軟に導入が可能である．実際に，Cray 社 Slingshot インターコネクトの Rosetta スイッチは，前述の Tomahawk 3 ASIC 技術を応用しているとみられる．

表 2.14 に示した Fat ツリー構成における Tier-1 スイッチは，現在，1 台のラックで構成されている．しかし，CPO が利用できるようになった場合，2020 年代後半にはラック内の数ユニットで実現できるであろう．さらに，もし仮に，CPO がプロセッサや GPU のパッケージに適用可能となった場合，そのパッケージからの I/O の開発が必要にはなるが，ノードあたり数 T〜数十 Tbps のインターコネクトの帯域を実現できる．DARPA PIPS プロジェクトでは，1 台の FPGA へ 100Tbps 級 CPO の実現を目指している．つまり，今後も劇的なノード内，ノード間結合網の性能向上が期待できる．

関連した動向として，DARPA fastNIC プロジェクトが挙げられる．すでに述べた通り，Top of Rack (TOR) スイッチ，プロセッサ，メモリは Tbps 級のデータ処理が可能である．一方，計算ノード内のネット

ワークインタフェース (NIC) の性能が数十〜100Gbps であるため，NIC が性能ボトルネックになっている．そのため，fastNIC プロジェクトでは，このボトルネックを解決することを目的として，NIC のスループットをハードウェア，ソフトウェアの本質的な再設計により 100 倍高速化することを目指している．成功した場合，通信がプロセッサの演算器レベルのデータ流のボトルネックにならない水準となる．

以上の動向より，CPO，オンボードオプティクスを中心とした光伝送技術，ハイパースケールスイッチ，NIC 革新がもたらす未来の結合網は，光伝送技術として従来の AOC のみを用いた結合網とは一線を画すことになるであろう．今後の結合網のアーキテクチャは，性能ボトルネックとならないように (コスト，消費電力を抑えつつ) 通信デバイスの異種混合を実現することで大きな発展可能性をもちうる．この点で，今後の結合網のアーキテクチャは，光伝送技術として AOC のみを前提とした現状のトレンドの発展的な継続と，CPO を用いる新たな構成の両方を見据えることになるであろう．

2.2.3 ストレージ

科学技術計算や各種データ解析において，必要とされるデータ容量は年々増大しており，ストレージシステムも大規模，複雑になってきている．

そしてこの科学技術計算の負荷はクラウドとは異なる特徴を持っている．まず負荷の特徴は，密結合されたノード上の数百万のプロセスが，一斉に一つのディレクトリに対して数百万のファイルを生成したり，同時にデータ書き込みを行ったりすることがあるという点である．次にデータの特性として，多次元のデータが多いと言うことが挙げられる．しかしストレージはデータを一次元で管理するため，適切にデータを配置しなければ特定のディスク領域に対して非常に多くの負荷が掛かってしまう事となる．このような特性を踏まえて，科学技術計算分野では階層ストレージを採用している [77].

例えば Trinity の後継として Crossroads が導入される 2023 年以降のストレージシステムの要求もストレージクラスメモリはまだ入っていないが，以下のような階層ストレージとなっている．

- ノード数: 20,000
- メモリ容量:2PB (DRAM)
- バーストバッファ: 4PB (NAND Flush)
- 分散ファイル: 100PB
- アーカイブ: 60PB

この大規模複雑化する要件を支えるために，ストレージシステムに関して様々なソフトウェア，ハードウェア技術が提案されてきている．例えばデータアクセス方法に対しては，ストレージ インタフェース (SAS, SATA など), ストレージ ネットワーク (iSCSI, FCoE など), ファイルシステム (Block, File, Object), その他のストレージ機能支援ハード (Computational Storage, Luster Appliance など) に関する研究が数多く行われている．これら新技術は容量増大やデータアクセス性能向上だけで無く，信頼性やスケーラビリティ強化にも着目して研究されている．本節ではこれらストレージ技術のうち，ハードウェアに関する内容について述べる．

2.2.3.1 階層ストレージの記憶メディア

■ストレージクラスメモリ 近年，DRAM よりも記憶密度が 2 倍高く，かつ容量あたりのコストは DRAM の 1/10 程度と安価になっている不揮発メモリをメモリスロットに差し，階層ストレージの一部にする動きが

ある (Intel Optane)．しかし，容量が DIMM の 2 倍程度しかないため，どの階層にどのデータを置くかの，アプリ依存の Domain Specific なデータマネジメントが活用に向けての鍵となっている．

■バーストバッファ [78]　SSD を搭載したバーストバッファノードをシステム内に配置し，それをローカルなファイルシステムにみせるようにしたもので，Cray DataWarp のミドルウェアよってサポートされた機能である．ユーザは必要なデータの一部をバーストバッファに転送しておくことで，グローバルファイルシステムに対するアクセスレイテンシを隠蔽できるようになる．計算ノード近傍に置かれた専用ノード上にバッファを配置する Local Burst Buffer 方式と，計算ノードとストレージの間に置かれた共用 IO ノード上に配置する Remote Shared Burst Buffer 方式がある．

■並列分散ファイルシステム　記憶メディア市場で主流となっているのは HDD と SSD である．それぞれの性能特性を比較すると SSD の方が優勢となる (表 2.15 参照)．各種市場調査結果によると [79]，2017 年頃より SSD の容量単価が下がってきたため高性能 HDD を置き換えるようになってきている．一方容量重視型のニアライン HDD に付いては容量単価が SSD よりも圧倒的に安価であるため，ニアライン HDD で容量を確保し，SSD ストレージで性能を確保するシステムが主流となってきている．

表 2.15　HDD と SDD の比較

	HDD	SSD	SSD の優位性
アクセス性能	5.5-8ms	0.1ms	5.5-8.0x
Random IO 性能	400io/s	6000io/s	15x
信頼性 (故障率)	2.5%	0.5%	4-19x
電力	3.86W	1W	4x

■アーカイブストレージ　例えば高エネルギー加速器研究機構 (KEK) では大量に生成される実験データを保存するために毎年 1〜2PB のストレージが追加で必要になるとしている [80]．このようにデータ解析対象となるデータは年々大規模になっているが，その中には殆どアクセスされない Cold Data が存在する．例えば文献 [3] では 80% のデータは保持は必要だが，殆どアクセスされないとされている．このようなコールドデータは容量単価の最も安い媒体への記録が望まれる．その解となるのがテープ媒体である． テープメディアの性能諸元を容量重視型のニアライン HDD と比較すると表 2.16 のようになる．

表 2.16　テープメディアと HDD の比較

	テープ (LTO10)	CO-HDD	テープの優位性
容量	6-10(TB/unit)	4-10(TB/unit)	1-1.5x
データ転送速度 (連続)	252-360(MB/s)	160-249(MB/s)	1.5x
製品寿命	30 年	3-5 年	6-10x
コスト	0.01($/GB)	0.05($/GB)	5x

テープストレージは Cold Data の保持に非常に有効であるため, 例えば NASA Advanced Supercomputing では 2.9PB の SATA RAID+ 380PB LTO という構成のストレージシステムを構築して持っている．また，KEK も 500PB 程度のテープシステムを整備する計画となっている．なお，テープメディアは連続アクセス性能では HDD 相当となっているが，ランダムアクセスは非常に遅い．そのため，Storage Class Memory と同様，テープでも，テープからディスク，ディスクからテープへ，どのデータをどのタイミングで移動するかの制御が非常に重要となっている．

2.2.3.2 ストレージインタフェース

メディアとストレージシステムと繋ぐインタフェースの規格としては，ATA をシリアル化した SATA, SCSI をシリアル化した SAS, 高速な SSD 接続を目指して転送速度を上げた NVMe, 筐体外のストレージエリアネットワークにも接続できる Fiber Channel, ストレージ以外の汎用デバイスも接続できる USB などが提唱されている．これらの性能諸元を比較すると，表 2.17 のようになる．

表 2.17　ストレージインターフェースの諸元 (出典：IT Skill Map, “インタフェース転送速度まとめ”)

	最大転送速度 (MB/s)	最大ケーブル長 (m)
USB3.1	1,250	1
NVMe	4,000	1
SAS3.0	1,200	10
SATA3	600	1
Fiber Channel	4,000	100,000

ローカルストレージについては，メディアの伝送性能とバランスがとれたインタフェースを選択する必要がある．そのため SSD は NVMe, HDD は高性能を狙うなら高価だが全二重通信方式の SAS，価格重視なら安価で半二重通信方式の SATA が多く選択されている．

2.2.3.3 ストレージネットワーク

分散配置された大規模ストレージを接続する方法としては，様々なインタコネクト伝送方式 (InfiniBand や Ethernet) の上に NVMe や SCSI のプロトコルを載せる方法が提案されている．インタコネクト伝送方式の詳細は 2.1.4 節 に記載されている．インタコネクト上にマップされた拡張ストレージインタフェースプロトコルとしては表 2.18 に示すものがあげられる．

表 2.18　拡張ストレージインタフェースプロトコル

元のプロトコル	拡張されたプロトコル	説明
NVMe	NVMe-oF	NVMe を Ethernet, InfiniBand, Fiber Channel 上に拡張
SCSI	iSCSI	SCSI を Ethernet 上に拡張
Fiber Channel	FCoE	Fiber Channel を Ethernet 上に拡張

この中でストレージエリアネットワークを最も安価に構築できるものは，専用機器が不要で既存のネットワーク環境上を利用できる iSCSI となる．しかし，その方式では SCSI リクエストを TCP/IP パケットに変換して処理を行う必要があるため，伝送の性能が低い．また，伝送の信頼性についても TCP/IP の再送機能に依存することになる．これに対して，FCoE は専用スイッチ (FCoE スイッチ) と専用ボード (Converged Network Adapter(CAN)) を必要とするが，Fiber Channel のプロトコルがそのまま使えるため性能が高く，また伝送信頼性や QoS も保証されるため大規模なシステムでも管理が容易になる．

これらストレージプロトコルをインタコネクト上に載せる方式以外に，I/O 処理による CPU 負荷を軽減するために，I/O カードが直接ストレージシステムにアクセスしてデータのやりとりを行う Remote Direct Memory Access(RDMA) 方式も提案されている．これらには iSER RoCE(iSCSI Extensions for RDMA, UDP を使用), iSER iWARP(TCP/IP を使用) 等がある．文献 [81] によると iSER のような RDMA 方式は，iSCSI に比べてレイテンシ 66% 削減，CPU での処理オーバヘッド 70% 削減の性能改善が見られる．しかし，

これらは Ethernet 上に構築されているため，大規模なシステムで信頼性を重視する場合には選択されない．

2.2.3.4 データアクセス方式

メディア上にデータをどのように配置し管理するかについては以下の 3 つの方式が提案されている [82]．本技術はストレージ管理のシステムソフトにも絡むため，この節では概要のみを扱う．なお，それぞれの方式で得失があるため，それぞれのストレージ階層に適した方式を導入する必要がある．

- Block Storage:
  - データをブロックに分割して管理
  - 構造化されたデータの管理が容易，高速アクセスが可能
  - スケーラビリティが無く，大規模ストレージには適さない
- File Storage:
  - 属性情報と階層フォルダ情報でデータを管理
  - 非構造データも扱える，属性管理されているため検索が高速
  - 属性管理手法にスケーラビリティがなく，大規模ストレージには適さない
- Object Storage:
  - 高度な属性情報 (Meta Data) を付与することで大規模分散管理
  - スケーラビリティがあるため大規模ストレージに適する
  - オブジェクト更新コストが大きいため，更新頻度が高いデータには適さない

2.2.3.5 ストレージ機能支援ハード

システムが扱うデータ量が増えるに従って，ストレージと演算処理部との距離が離れてきたため，ストレージに関する処理をストレージ近傍で行おうとする取り組みも行われている．これが Computational Storage である．Computational Storage には固定処理を行う Fixed Computational Storage Service(FCSS) と任意の処理をユーザが定義できる Programmable Computational Storage Service(PCSS) とがある．

- FCCS: 圧縮，暗号，ビデオコーディング，Erasure Coding, ファイルシステム処理 (Luster) などの処理オフロード
- PCCS: OS/コンテナのイメージローダ，ユーザ定義処理 (集合演算や変換など) オフロード

上記以外にも，ストレージシステムにおいては電力制御，アクセス管理，信頼性保証など非常に重要な処理が多数必要となっており，それぞれに専用ハードによる支援が求められている．これらも今後 Computational Storage が網羅していく領域となる．

2.3 システムソフトウェアの技術動向

2.3.1 基盤ソフトウェア (OS，仮想化，コンテナ)

2.3.1.1 コンテナ，仮想化技術

スーパーコンピュータなどの HPC シスムは高い処理性能を持つ一方で，近年の NUMA 構成や不揮発性メモリの台頭，CPU コア数の増加やアクセラレータの導入によってアーキテクチャの複雑化が進んでいる．そ

のため，高い並列性や複雑なメモリ階層によってアプリケーション開発コストが高いことやアプリケーションの移植性の低さが課題となっている．また，データ解析を主とする研究では，研究結果の再現性を保証することが重要であり，再現性の向上の点においても仮想化技術が求められている．しかしながら，Linux などの汎用 OS や KVM などのハイパーバイザといった従来の仮想化技術は，高性能が求められる HPC システムにおいては仮想化オーバヘッド (OS ノイズ) による性能低下が無視できない．このため，処理性能とユーザビリティのトレードオフが課題となっている．

一方で，コンテナ [83, 84] やユニカーネル [85, 86] といった軽量な仮想化技術がクラウドを中心に注目を集めている．仮想化オーバヘッドを抑えつつ HPC システムのユーザビリティを向上するために，HPC においてもこれらの軽量な仮想化技術の導入が検討されている．Shifter [87] や Singularity [88] は，HPC 向けに開発されたコンテナ技術である．Docker コンテナの稼働にはユーザが root 権限を持つことが要求されるが，これは多くの HPC システムでは許可されない．そのため，Shifter, Singularity は root 権限を持たないユーザがコンテナを管理できるように設計されている．Shifter は Cray (HPE)，Singularity は Comet supercomputer (SDSC) において導入，運用されており，Singularity は富岳 (理研) においても導入が予定されている [89].

アプリケーションコードと OS を同一アドレス空間で実行するユニカーネルは，OS ノイズ削減に特に効果的であると期待される．既存の HPC システムにおける OS カーネルの軽量化手法として，McKernel では，システムコール等の OS サービスの一部を削除した軽量カーネル (Light-Weight Kernel, LWK) を稼働することで，計算タスク実行中の OS オーバヘッドを削減している．一方で，ユニカーネルは機能のみをライブラリとしてアプリケーションコードに追加し，単一の実行バイナリを生成する．アプリケーション実行に必要最小限の機能のみで構築されること，同一アドレス空間で動くことから更に OS オーバヘッドを削減できると期待される [90].

2.3.1.2 不揮発性メモリ (NVRAM) の利用

近年，HPC システムにおいてグラフ処理やディープラーニングといった大規模データを処理するアプリケーション実行の要求が益々高まっており，大容量メインメモリの実現は次世代 HPC システムにおいて必須課題であると言える．DRAM は代表的な計算機の主記憶装置として広く用いられてきたが，データ保持のためのリフレッシュによる電力消費量の増加が課題となっており，特に電力制約の厳しい HPC システムにおいては，これ以上 DRAM を用いてメモリシステムを大容量化することは困難となっている．

そこで低電力な不揮発性メモリ (Non-Volatile Memory) の活用が注目されている [91]．2.1.3 節で紹介したように，STT-MRAM, Phase Change Memory (PCM) に代表される NVM はナノ秒オーダの短いレイテンシで読み書きできる．加えて，データを保持するためにリフレッシュ処理を行う必要がないため，DRAM よりもエネルギ効率が高いと考えられている．このように NVM は従来の DRAM に替わる次世代の主記憶装置として期待されている．

2019 年には NVM を用いた DIMM スロットに挿入可能なメインメモリモジュールの製品版，Intel Optane Persistent Memory [92] が登場したが，このような NVDIMM は，既にいくつかの HPC システムで導入されている (Oak Ridge Summit [93], Argonne Aurora supercomputer [94])．HPC クラスタでは，NVDIMM をバーストバッファとして利用し，I/O の高速化する用途でも使われている． Oak Ridge National Laboratory の HPC クラスタである Summit では，各ノードにバーストバッファとして利用できる NVDIMM が搭載されている [93]． このバーストバッファは大規模なデータ解析における事前のデータステージングやストレージへの書き込みが多いワークロードにおける書き込みの書き込み I/O の集約に利用され，これらワークロー

ドにおける処理時間の短縮に貢献する．

また，エクサスケール時代にはノード数が増加し，耐障害のためにチェックポイントを作成する必要がある． このチェックポイントの取得においても，NVDIMM の利活用が期待されている [93, 95]．

2.3.2 大規模並列/高性能計算

2.3.2.1 CPU・アクセラレータ

近年は，微細化によって増加したハードウェア資源はチップ内の並列度を高めるために使われており，ハードウェア資源量の増加に伴い，利用頻度の高いと想定される描画，乱数生成，暗号化などの多様な専用ハードウェアを，汎用プロセッサのチップ上に搭載することも可能となってきた．システムソフトウェアはこれらの並列化，複雑化するハードウェア構成要素を適切に管理することが求められ，高性能計算システムの実用時の性能を決める最重要なパーツの一つとなっている．今後，ムーアの法則が限界を迎えてハードウェア資源量の指数関数的増加が見込めない時代が到来すれば，システムソフトウェアの技術開発が実性能を高めるための最も有効な手段の一つとなることは明らかである．

Summit に代表される現在の大規模高性能計算システムでは，汎用プロセッサ (Central Processing Unit, CPU) に加えて描画処理ユニット (Graphics Processing Unit, GPU) を演算アクセラレータとして搭載する構成が一般的になっている．元来，GPU は描画処理専用プロセッサであり，CUDA の登場により GPU が高性能計算に本格的に利用できるようになった 2006 年においても，その機能は CPU と比較して制限的だった．しかし，高性能計算分野および機械学習分野での需要の高まりに応じて GPU は世代ごとに高性能化と高機能化を進め，CPU とほぼ同等といえるほどの機能を有するようになった．その結果として，現在の CPU と GPU は，両方とも性能的な特長の異なる汎用プロセッサと扱うこともできる．ただし，システムソフトウェアの視点から見ると，両者の扱いにはまだまだ異なる点も多い．例えば CPU では OS が稼働しているが，GPU は CPU 側のプロセスからドライバを介してアクセスされる前提になっており，その扱いは OS レベルでは大きく異なる．

今後は，GPU よりもさらに用途の限定的なハードウェアを搭載する可能性も指摘されている．そのような専用システムの代表例としてしばしば挙げられる Anton は分子動力学計算専用であり，専用 ASIC が 3 次元トーラス状に接続されたシステム構成となっている．このようにハードウェアを専用化して不要な回路を削減することができれば，より多くのハードウェア資源を対象アプリケーションのみに利用することができ，電力あたりの性能改善につながることは明らかである．ただし，その実現のためのコスト (金額) を考えると，かなり広範な応用分野で利用できなければ専用ハードウェアを ASIC として実装するのは困難であり，専用化というアプローチとは矛盾する側面がある．このため，専用ハードウェアを ASIC と比較して安価に実装可能な再構成可能デバイスの高性能計算分野での利用が，今後さらに進むことも予想されている．

2.3.2.2 次世代特化型ハードウェア

■データフロー型 データフロー実行モデルの歴史は古く，1970 年代に Jack Dennis 教授らが先駆的に研究を開始したことを契機に 90 年代前半までハードウェアから命令セット，プログラミングに至るレイヤにおいてデータフローの概念に基づくアプローチは盛んに研究されてきた．データフローコンピューティングの基本的な原則は，入力が揃い次第すぐに命令を実行していくことである [96]．コンピュータの処理方式としては極めて自然な方式であるが，プログラムに内在する並列性をデータフローマシンでどのように扱っていくかを考えていく過程で制御方式も含め局所的に処理を進めていくことを実現する実装の複雑さから，C に代表され

る手続き型言語と汎用 CPU による命令レベル処理というメインストリームの流れに取って代わり，データフロー実行モデルとしては冬の時代を迎えた．一方で，命令レベル並列性の利用の限界や，プロセッサの電力の問題から，ドメイン特化型の処理方式によるアーキテクチャが次々に開発される中で，データフロー実行モデルはそのモデルの単純さや美しさから派生する高効率性から，近年再び脚光を集めている．

データフロー実行モデルのアーキテクチャを考える上で，命令レベル，専用処理ユニットレベル，タスクレベルの 3 つが考えられる．命令レベルのデータフロー方式とは，命令セットとしてデータフローの概念に則ったマシン語を持つコンピュータである．90 年代までのデータフロー方式においては，水平方向に演算器を展開し，入力が揃った処理から順次演算を行っていく仕組みがとられていたが，並列度を上げるにしたがって演算の出力を任意の演算器の入力とするために必要なノード間接続網が複雑となり性能のボトルネックとなっていた．データフローの目指した水平方向の空間的な並列性の利用は，結果的に命令レベル並列性を活用したアウトオブオーダ方式のスーパースカラが行っている依存関係を満たすスケジューリングと等価と見なすことができる一方で，データ依存関係を命令レベルで記述することによりレジスタリネーミングが不要となるといったハードウェア構造におけるメリットもあり，2000 年代において Explicit Data Graph Execution (EDGE) アーキテクチャとして定義されるに至った．一方で，命令レベルのデータフローマシンで商業的に成功した事例はほとんどない．

専用処理ユニットレベルにおけるデータフローモデルとは，処理のデータフローグラフをハードウェアに展開し，対象のデータフローグラフの処理を直接実行する回路を構成することにより処理を進めるアプローチである．データフローのノードに対応する演算器をハードウェアで用意することになるが，データフローグラフのノードに対応する演算器における処理はレイテンシがかかるため，多段に結合されたノードにおいてはパイプライン的に処理を進めていく必要がある．そのため，ストリーミング処理のようなあるカーネルにおいて入力となるデータが常に変化していくパターンの処理との相性が良い．近年の深層学習向け専用チップにおける行列・ベクトル積の演算はこの考え方に基づき，演算器が空間方向，時間方向に展開されている．このような構成様式はシストリックアレイとも呼ばれ，Google TPU のような ASIC ベースの設計にも採用されている．

専用処理ユニットレベルにおいて，汎用性を高めようと試行する場合，粗粒度リコンフィギャラブル方式の回路を用意するアプローチがとられる．近年では，スタンフォード大の Kunle Olukotun 教授らの研究チームがループタイリングを HW 向け並列パターンテンプレートに応用する試み [97] を行っている．このテンプレートは Delite Hardware Definition Language(DHDL) として定義され [98]，Plasticine という SIMD-lane ベースの粗粒度リコンフィギャラブルチップにおける調整可能なアーキテクチャ構成のテンプレートとして活用されている [99]．これらの研究においては，DHDL は Scala による組み込み DSL として実装され，Chisel により ASIC 向けハードウェアが合成されるフローとして開発されている．

粗粒度リコンフィギャラブル方式は，実行中に回路構成を容易に変更できるため，プログラマビリティが高いといえる．一方で，ストリーム処理に特化することにより，実行中のプログラマビリティが必要ないとも考えられるため，FPGA による専用処理ユニットをアプリケーションごとに用意するというアプローチもある．FPGA でデータフロー方式の専用ユニットを作るシステムの代表的な例として Maxeler 社の DFE があり，金融分野のリアルタイムトレーディングや，リアルタイム地下資源探査のためのステンシル計算などに実応用されている [100][101][102].

3 つ目のタスクレベルのデータフロー方式とは，命令をはるかに超える粒度のタスクをデータフローグラフにおけるノードに定義し，タスク間の依存関係を記述しながら，依存のある前段のタスクがすべて完了次第すぐに処理を進める方式である．古くは 80 年台後半に，マクロデータフローとして，コンパイラの階層を中心に関数型言語と処理の階層化が論じられている [103]．近年においては，様々なタスクベースの処理の記述や，

ワークフローの概念とほぼ一致する．専用のハードウェアを考える試みとしては，細粒度のパイプライン並列性をオンチップスケジューラに構成可能なタスクとルールとして抽象化し，グラフ処理のような不規則なアプリケーションを効率的に FPGA に実装する手段を提案している清華大学の研究チームの取り組みなどがある [104]．このようなタスク並列の考え方は，スタンフォードの並列パターンテンプレート DHDL におけるメタパイプラインの概念と同様であり，階層的にデータフローを考えていく必要があることを示唆している．また，提案する細粒度タスク並列を FPGA に実装すると，メモリサブシステムがボトルネックになるという結果も示している．

2.3.2.3 メモリ階層，データ移動

単一メモリ空間上に異なる性質を持つメモリ階層を持つメモリアーキテクチャが広がりつつある．Intel 社のメニーコアプロセッサである第二世代 Xeon Phi (Knights Landing) は，広くサーバで用いられている大容量の DRAM メモリの他に，高バンド幅である MCDRAM を搭載している．Xeon Phi はいくつかのメモリモードをサポートしており，MCDRAM をキャッシュとして使うこともできるが，両者の性質をアプリケーション側で活用するために，ハード的には単一のメモリ空間上にマップしてシステムソフトウェアに見せることも可能である．この場合，OS が，それぞれのメモリを別々の numa ノードとして見せることで，アプリケーションが明示的に指定して使い分ける機能を提供している．同様に，Intel 社の不揮発メモリである Optane DC が搭載されるサーバでも，揮発メモリである DRAM と単一のメモリ空間上にマップしてしており，OS は異なる NUMA ノードとしてユーザに提供するため，単純に低速・大容量の揮発性メモリとして明示的に使用することができる．

2000 年代後半以降の主要なサーバーアーキテクチャは Cache Coherent NUMA 構成が主流である．複数のプロセッサから構成されるサーバにおいて，それぞれのプロセッサに実装されたメモリコントローラの先にローカルのメモリが接続され，これらのプロセッサ同士が相互接続され，全体としてキャッシュ一貫性をもつ共有メモリを構成する．このようなアーキテクチャでは，演算コアとメモリの位置関係によりアプリケーション性能が大きく変わる．この最適化のための研究が取り組まれている．最近では，単一のプロセッサであっても，内部のコアを分割し，Sub-numa cluster を構成し複雑化していることから，演算コアとメモリの割り当てはより重要な研究課題となる．

Optane DC の登場により，その不揮発性を活用する研究開発も進んでいる．各ページのアクセス頻度からどちらのメモリに配置すべきかを判断し，最適化を行う研究事例 [105] がある．この機能をハイパーバイザ上に実装することで，上位の VM におけるメモリの最適配置を実現しようとしている．また，プロセスの状態とデータを OS が定期的に不揮発メモリに保存することでチェックポイントを取得する機能を実現した研究事例 [106] がある．いずれも OS やハイパーバイザで実現されているため，アプリケーションの改変は必要なく，最適配置やチェックポイント機能を実現できる．

近年の高性能計算システムではアクセラレータを搭載する事例が多いが, GPU に代表されるアクセラレータと CPU との間は，独立したメモリ空間として構成されるのが基本である．通常の使い方では，cudaMemcpy のようなプログラマが明示的にアクセラレータ-CPU 間のデータ移動を記述する．API は，memcpy 型であり，転送元・転送先の双方のアドレスを指定してデータ移動を記述する．同期型だけでなく非同期型の API も提供されている．また，明示的に両者のメモリの違いを意識しなくてもプログラムを記述するための技術として，NVIDIA 社によって提供される Unified Memory がある．本技術を用いれば，両者のメモリの違いを意識することなくプログラマは単一のメモリ空間として扱うことができる．

2.3.2.4 通信，ネットワーク，インターコネクト，データ転送

以下に通信やネットワークにおける基盤ソフトウェア関連の動向をまとめる．

- ネットワークトポロジーとシステムソフトウェア:
  2000 年代以降，大規模，具体的には 1 万ノードを超える計算ノードの接続技術が研究開発された．接続台数が増えても台数の増加程度にしかコストが増大しないいわゆるスケーラブルな構成として，Torus 網が注目され，IBM BlueGene や京コンピュータに実装された．Torus 網における 1 対 1 通信のルーティング手法は従来から広く知られる次元順ルーティングが基本でありハードウェア実装されているが，集合通信の効率化に関する MPI 実装が研究されており，Torus 網において集合通信の衝突を避けつつ，効率性を高める手法が提案されている．
  一方で，一部のハイエンドを除く主に 1 万ノード未満のスーパーコンピュータの多くは，専用ネットワークではなく，InfiniBand に代表されるオープン技術が採用されている．InfiniBand においては，そのルーティングは SubnetManager と呼ばれるシステムソフトウェアが制御する．InfiniBand は原理上はあらゆるネットワークトポロジーに対応できる．あらゆるトポロジーに対してデッドロックフリーなルーティングとして LASH アルゴリズムや DFSSSP アルゴリズムが提案されて，標準的な SubnetManger である OpenSM に実装されている．一方で，実用的には，InfiniBand は多くの場合，2 段もしくは 3 段の Fat Tree 構成が広く採用されている．
  CRAY 社のインタコネクトである Aries や Slingshot は Dragonfly トポロジーを採用している．これは，全体全結合を多段にしたネットワークワークトポロジーである．ハードウェアによるアダプティブルーティング制御により高速性を実現している．全体全結合を基本コンセプトとしたトポロジーとして Mellanox 社は Dragonfly+ を提供している．また Slim fly は，小さい直径を実現している．上記のように，ルーティング制御については，ハード実装される部分も多いが，MPI や SubnetManager といったシステムソフトにより実現される部分があり，ネットワークシステムの進歩に合わせて技術開発が進んでいる．
- 通信オフロード:
  Mellanox 社は，CORE-Direct と呼ばれる集合通信オフロード機能を実装している．この技術では，複数のステップから構成される集合通信において，ホスト側までパケットを転送することなくネットワークアダプタ上で次のステップの通信を開始することができ，OS ノイズの影響を受けずに集合通信を進めることができる．実際に同社の MPI 実装と組み合わせることで利用できる．同様にスイッチ側のオフロード機能として SHARP が実装されている．これは，スイッチ側で集約計算をオフロードするもので，従来何フェーズかに分割して各計算ノード上で行っていた集約計算をスイッチ上で行うことで，加速することができる．
- 通信進捗処理:
  MPI の実装では，非同期通信を実現するために，進捗処理を実装する必要がある．進捗処理とは，計算プログラムのコンテキストの裏側で通信のために必要な制御のことであり，例えば，非同期に到着するメッセージを処理したり，集合通信のようにあるメッセージの受信により次のメッセージを送信の起動する処理がある．計算プログラムの裏側で行う必要があるが，典型的な実装では MPI_Test() 等が呼び出された際に MPI ライブラリ側にコンテキストが移った際に進捗処理を実施している．これをバックグラウンドで実施する方法として SMT (Simultanous Multi Threading) やアシスタントコアと呼ば

れる MPI や OS 処理等を担当する専用コアを用いる方式がある．これにより，MPI_Test 等の呼び出しに頼らない進捗処理を実現している．特に，アシスタントコアのような専用コアを用いる方式では，計算処理への干渉を最小化することができる．

2.3.2.5 ストレージ，I/O システム，データ管理

近年の HPC クラスタでは，数値計算のみならず，ビッグデータ解析や機械学習といったデータ処理のワークロードも多数存在する．また，モバイルデバイスや IoT 機器の普及，センサの高性能や測定機器の高性能化により，解析対象のデータサイズは増加の一途を辿っている．これらのことから，HPC ではより巨大なデータセットを用いた解析処理の需要が高まっている．例えば，画像分類で有名なデータセットである ILSVRC2012 は 1000 万の画像データで構成される合計サイズ約 154 GB のデータセットであり，[107] で行われた宇宙学向け機械学習の実験で用いられたデータセットは合計サイズ 1.4 TB であった．

解析対象のデータは，ユーザがクラスタ内のグローバルファイルシステムにアップロードし，その後グローバルファイルシステムから計算ノードに読み込まれ解析されるのが一般的である．グローバルファイルシステムには，並列ファイルシステムである Luster や GPFS などが広く利用されている．また，I/O の高速化のためにグローバルファイルシステムと計算ノードの間にバーストバッファと呼ばれる高速なストレージが配置される．このバーストバッファは各計算ノード毎に配置するもの，いくつかの計算ノードに対して一つずつ置くものなどがある．また，バーストバッファには NVMe や NVDIMM など高速なストレージデバイスが利用される．これらのシステムでは，大量の解析データの読み込みや解析結果の書き出しを高速化することを目的に研究開発が進められている．

2.3.3 プログラミング環境

2.3.3.1 プログラミングモデル

スーパーコンピュータで動作するコードをプログラミングしていく上では，並列性を如何に記述していくかがポイントとなるが，伝統的にプログラミング言語と，言語をベースとして特定の API やライブラリによる実行環境を提供するプログラミングモデルに分けて考えられてきた．すなわち，Fortran と C 言語ではベースとなるプログラミング言語の記述は異なるが，MPI を用いてプロセスレベルの並列性を記述していくということは上位のプログラミングモデルの点では同一と考えられる．

まず，並列プログラミングモデルの技術動向であるが，2000 年代中盤より CPU 主体としたシステムにおいて主要なプログラミングモデルとなったスレッド並列と分散並列のハイブリッドプログラミングモデル，具体的には OpenMP と MPI とのハイブリッドプログラミングモデルは，2020 年の時点でも同システムで広く用いられるプログラミングモデルであり，特に古典的な HPC アプリケーションの多くは，OpenMP+MPI ハイブリッドプログラミングモデルに基づいて記述されている．GPU を主体としたシステムにおいては，以前は CUDA などの GPGPU 向けのプログラミングモデルが主体であったが，近年では OpenACC や OpenMP などの指示文に基づくプログラミングが性能面でも CUDA を上回るようになっており，GPU システムの主体になりつつある．通信面においては，NVLink などの GPU 間の直接通信が利用可能になり，これをサポートする API のほか，GPU システムのノード数は単一 GPU の性能向上やノード当たりの GPU 数の増加による増加が抑えられる傾向にあることから，従来のホスト CPU を介した MPI 通信も用いられている．

近年，高い注目を集める AI/ML のアプリケーションにおいては，従来のプログラミングに加えて，カーネ

ル部分が最適化された専用ライブラリを用いることが主流となっており [108]，前処理，ライブラリの呼び出し，および後処理などが Python などを用いたワークフロープログラミングモデルに基づいて記述される．

データ分散や通信を抽象化して記述できる Partitioned global address space (PGAS) 言語は，従来の global address space に基づいて抽象化された通信に加えて，Fortran や XcalableMP における coarray のように，内部で最適化された片側通信を用いる片側通信をサポートしており，平易な記述で高い通信性能が得られるようになっている．

タスクベースのプログラミングモデルについては，2 種類のまったく異なる重要な方向性がある．1 つは細粒度タスク，もう 1 つは粗粒度タスクである．細粒度タスクは，OpenMP の task などに代表される，比較的軽い作業単位をスレッドなどで実行する．メニーコア CPU では，従来のループレベルの並列化などでは，コアを使い切ることができず十分な性能がでないことがあったが，task によってより柔軟に作業単位を割り当てることで効率化が図れる可能性があるほか，通信と計算のオーバーラップなども平易に記述でき，ランタイムに柔軟に実行することが可能になった．粗粒度タスクは，比較的大きな作業単位を，より大きく場合によってはヘテロジニアスな計算機リソースで実行する．MPI や PGAS 言語 XcalableMP で開発されたタスクをワークフローとして実行する枠組み [109] や，COMPSs[110] と呼ばれる Java，C，C++，Python でのアプリケーション開発をサポートし，クラスタやクラウドなどで逐次または MPI のタスクを実行するフレームワークなどがある．また，eFLOW4HPC プロジェクト [111] では，HPC シミュレーション，ビッグデータ解析，機械学習を融合するワークフローツールをめざしている．

生産性の向上という観点では，近年のドメイン特化型の言語の普及もプログラミングに係る重要な動向である．機械学習や深層学習に特化した各種フレームワークも Python をベースとするドメイン特化型言語とみなすこともできる．また，Julia のように数学的な処理の記述性を高めている言語も登場している．Julia は，Matlab と Python の良い点を併せ持ちつつ，LLVM をベースに JIT 方式のコンパイラとして構築されている．C 言語レベルで用意したコード記述と遜色ない性能が期待できる仕様であると同時に，専用の数値計算パッケージの利用や，BLAS や LAPACK のような一般的な数値計算ライブラリをシームレスに使うことも可能であるため，今後の HPC 分野での利用の普及が期待されている．

プログラミングにおいてデータ抽象化はとりわけ重要な技術である．しかしながら，現在，高性能計算で用いられるプロセッサの多様化が進み，それらの間でデータ抽象化とともに性能可搬性 (performance portability) を実現するのは容易ではない．現在，高性能を達成するためにメモリアクセスパターンに求められる制約がプロセッサごとに異なり，高い性能可搬性を実現するための制約になることが指摘されている [112]．すなわち，高い性能可搬性を実現するためには，メモリアクセスパターンをプロセッサごとに変化させる必要があるが，並列性を抽象化するプログラミングモデルが広く検討されてきたことと比較すると，メモリアクセスパターンの抽象化はあまり行われてこなかった．そのようなメモリアクセスパターンの多相性 (polymorphism) を実現するため，いくつかのプログラミングフレームワークが提案され，活発に開発されている [113][114][115][116][117][118]．例えば，Kokkos は，階層型メモリを持つ並列環境におけるプログラムの抽象化を可能にし，また，OpenMP, CUDA などをサポートするバックエンドを提供することで，多様なアーキテクチャに対して，高い性能可搬性を実現している．JARA も同様に C++ における抽象化のためのライブラリであり，バックエンドは GPGPU, OpenMP, SIMD プログラミングをサポートする．

プログラミング環境として，プログラミング言語，ライブラリ開発，コンパイラや開発ツールチェイン全般に言えることであるが，プログラミング環境の開発と維持には大きなコストがかかるため，コミュニティ全体としての動向も非常に重要である．過去においては，エンタープライズやクラウドなどメインストリーム IT の流れと HPC 向けの流れは独立した事象に感じられてきたが，近年，メインストリームのハイエンドの領域

では HPC とマージして方向性を作っていっているような動向もある．メインストリーム IT 技術の分野としても HPC のスケーラビリティを実現する技術を利活用したいとの流れから，HPC 発のプログラミング技術が，今後，標準化され，メインストリームの技術として展開されていく事例も増えることが予想されている．

このような流れにおいては，スーパーコンピュータにおいてもシステムソフトウェアはオープンソースベースで構築していく方がエコシステムを構築する上では好ましいため，スーパーコンピュータのプログラミング環境のオープン化やコモディティ化に対するニーズはますます高まっているといえる．近年，メインストリーム IT のシステムソフトウェアを構築する言語として Rust が普及しているが，スーパーコンピュータのシステムプログラミングにおいても高信頼性を担保するために C/C++ だけではなくこのような新しい言語を取り入れていくべきとの機運もある．一方で，アプリケーションソフトウェアのプログラミングに関しては，従来からの C/C++，Fortran でのカーネルの構築に加えて，Python からのカーネルの呼び出しや各種 DSL の利用，Julia による数学との親和性が高いプログラミング，PGAS 言語による局所性を意識したプログラミングへの展開など幅広いプログラミング環境が利用されれている．今後も，このような多様化に加えて，多くのユーザや開発者に支援されるプログラミング手法への集約化も同時になされつつ，プログラミング環境の進化が続いていくと考えられる．

2.3.3.2 コンパイラ

コンパイラはプログラムの性能最適化を司る基盤となるツールである．一方で，コンパイラの進化による実行速度の改善は，これまで 18 年で 2 倍程度であったといわれている．このコンパイラの進化は Proebsting's Law として知られており，Moore の法則の示す 18 か月で 2 倍よりもオーダが一つ異なるぐらいゆっくりとしたペースで改善してきた歴史がある．[*15]

このような性能向上に対するコンパイラの緩やかな進化の背景には，コンパイラが性能ではなく生産性に注力してきたこともある．すなわち，コンパイラの分野の最も顕著な貢献は，高級言語の利用が進んでいる現状においても，これまでよりもコード品質を落とすことなく高性能のコードを生成することを可能にしてきたことであるといえる [120]．更に，近年は抽象度が非常に高い言語やドメイン特化型言語の普及により，プログラミングの生産性が格段に上がりプログラムの開発コストは下がる傾向にある考えられる．一方で，プロセッサのハードウェアの構成が複雑化したため，階層的な並列性やデータ局所性を考えずに単純にコードを開発した場合に，ハードウェアの計算能力を十分に引き出さないまま実行が行われるということも多々見られるようになった．

このような背景の下，Google や Amazon はキャッシュの局所性を活用して性能を数 % でも改善していくことが，大きな経済的な効果をもたらすことをよく理解し，取り組みを始めている．また，国際的にも，科学的な切り口から，性能向上のためのソフトウェアのリファクタリング法を体系化する，新しいアルゴリズムを開発する，コンピュータハードウェアを能率化するなど様々な面でソフトウェアを改善する方法を模索していくことへの機運が高まっている [121]．コンパイラの観点では，ソフトウェア性能工学におけるソフトウェアを性能向上のために改善していくプロセスにおいて，コードの手作業での書き換えに依存してきたチューニング技術をコンパイラに取り組み，自動化するべきとの方向性が非常に重要である．

[*15] この法則を 2001 年を基準に検証した論文 [119] において，1965 年に最適化コンパイラの研究が始まったとして，その 36 年の成果が-O3 に反映されていると仮定し，その時点の最適化なしの-O0 との差を調べた．最適化の有無の差が SPEC INT において 4 倍より若干少ない程度であったが，4 倍と近似すると年率で 4% という計算になり，18 年で 2 倍という計算となり Proebsting's Law の法則と一致する結果となった．なお，浮動小数点処理を利用したアプリケーションを使ったベンチマークである SPEC FP では，36 年間で 8.1 倍だったので，年率 6% となるため，数値計算分野に対してはコンパイラの進化の速度は Proebsting's Law の示したものよりも若干速いペースであるがほぼ同じオーダでゆっくりと進んでいるというのが実状である．

コンパイラに関する近年の技術動向としてもう一つ重要なものに，オープンソースによるエコシステムの構築と，多層コンパイラとしての実装の進化がある．LLVM が登場したことを契機に，コンパイラ技術のオープンソース化が進み，オープンソースコミュニティの強力な支援により実用レベルのコンパイラとして普及するに至り，今や，オープンなコンパイラが最新の研究成果を反映した最適化機構を取り組むことによりベンダ製コンパイラを凌駕する最適化性能を達成することが可能となりつつある．このような流れにより，独自設計の言語から実用的な実行コードを生成することが研究室規模のチームでも可能となった．この流れに沿って，様々な言語が開発されると同時に，コンパイラの中のコード変換過程も，目的の最適化にフィットする形で中間言語 (IR) の抽象度をおのおの設定できるように機能の分割が進み，入力からバイナリを生成するために複数の IR を経由する多層コンパイラ（Multilayered compiler）によるコード最適化とコード生成が普及しつつある．

一般に，高級言語から，低水準の記述に変換していく過程で，様々な情報が失われていくといわれている．したがって，実行可能バイナリコードにある情報のみを使って，同等の動作をする高級言語の記述に直すためには様々な高度な推定をしていく必要がある．1 つの解として，最適化のために元のソースコードをデバッグ情報として付加しておくというアプローチも提案されているが，普及が進んでいない．そこで，多層コンパイラにおいては，適切な最適化を行うために必要となる情報が得られるレイヤ毎にコンパイラの最適化プロセスの変換を重ねていく．

多層コンパイラの構成は，DSL とも親和性が高く，レイヤ毎に高度な最適化を施していく上で好ましい構造となっていることが知られている．例えば，近年普及が進む機械学習向けのフレームワークにおいては，DSL の記述を，DSL レベル，グラフレベル，ループレベル，命令レベルの IR に順次変換しながら各々のレイヤでの高度な最適化を進めていく．これにより，これまで各レイヤでの最適化を見据えて人間のエキスパートが手動で行ってきたコードの書き換えが多層コンパイラにおける 1 つのレイアの最適化機構としての実装が進み，結果として，多層コンパイラで変換を重ねていくことにより，一連の高度な最適化の自動化が進んでいくであろうとの見通しである．

近年のハードウェアの高並列化の流れにおいては，コア内の SIMD ユニットは対応するデータ語長を拡大し，いわゆるロングベクトルといわれる 64 要素以上のベクトルもサポートするようになると同時に，各ベクトル要素に対する演算の機能を強化してきた．CPU ベンダ製のコンパイラにおいては，当然，自社のベクトル命令が効率的に生成されるように改良を進めるが，LLVM に代表されるオープンなコンパイラフレームワークにおいてもこのような流れにいかに追従するかという議論がなされている状況である．多層コンパイラの構成の面から考えると，ベクトル命令に対応するべきなのは，ループレベルの記述であるか，命令レベルの記述であるかという議論となる．LLVM の方向性としては，LLVM-IR にベクトル命令を表現できる拡張を追加することも検討されているように [122]，ループレベルの最適化の結果としてベクトル命令に変換するほうが，AVX512, ARM SVE, RISC-V V extension，NEC SX-Aurora といった近年開発されているベクトル命令に統一的に対応できるため好ましい．しかし，現状のところとしては，代表的なループレベルコンパイラの実装である Polly においてベクトル化を行った結果は後段のバックエンドが確実にベクトル化を行えるように依存関係を排除したループで記述された IR として出力され，各命令セットに対応する LLVM のバックエンドにおいて独立にそれぞれのベクトル化機構を通してベクトル命令に変換される流れとなっている．従って，アーキテクチャに依存しないロングベクトルへの対応に関しては，現状では課題が多い．

コンパイラにおける自動チューニングは，アルゴリズムとコンパイラによるコード変換の間にあるギャップを埋める技術として近年注目を集めている．数値演算ライブラリのレベルにおいては，複雑な性能最適化や自動チューニングの仕組みを内部に隠蔽することが継続して取り組まれており，利用者はそれらを意識すること

なく自動チューニングによる性能向上の効果のみを享受可能となってきた．しかし，すべての計算をライブラリ化することは不可能であり，ライブラリ化できたとしても強い制約を受ける場合がある．そのような場合に，コンパイラやコード生成器といったアプローチが有望である [123].

歴史的に，コンパイラによる性能最適化は性能低下を招かない場合にのみ適用 (do-no-harm philosophy) されてきた．しかし，自動チューニングの利用を前提にすれば，コンパイラは潜在的により積極的な最適化や特殊な最適化を試すことが可能となり，より高い性能を達成できる可能性がある．そのような自動チューニング機能を実現するためには，コンパイラに対してアプリケーション特有のコード変換や自動チューニングの性能パラメータの探索空間を与える必要があり，様々な研究がなされてきた [124][125][126][127][128][129]．多くの事例研究によって，手動でのコード最適化に匹敵，あるいは時として上回るような性能を自動チューニングが実現できることが報告されている．

2.3.3.3 カスタムアクセラレータや FPGA に向けてのコード生成

GPU やカスタム命令をもつアクセラレータを集積するヘテロジニアスなシステムにおいては，それぞれの ISA に対応するコンパイラが，実際のリソースを利用するために必要な実行コードを生成する．現状では，それぞれのデバイスの利用法に対応するディレクティブを挿入する，あるいは，独自のプログラミング言語で記述するなどの汎用的なプログラムからの移植が必要があり，生産性の面で課題もある．現状では，アクセラレータに向けたコード生成は，ランタイムライブラリ開発者あるいはアプリケーション開発者の努力によって活用されているのが現状であり，一般的なユーザがアプリケーションの生産性を保ったまま利活用するのは難しい状況である．

なお，FPGA や ASIC の開発フローを用いてドメイン特化型プロセッサを開発する過程においては，近年，高位合成 (High-level Synthesis, HLS) と呼ばれる C 言語レベルの記述にディレクティブを与えることによる回路の合成の技術が普及している．高位合成においても，多層コンパイラの考え方と同様に，HLS 記述から RTL というゲートレベルの記述に落とし込む高位変換層，RTL から実際のデバイスの回路データに落とし込むバックエンド層に大きく分類できる．前者は，LLVM などの一般的なオープンソースのコンパイラをフロントエンドをベースとして一般的に開発が進められている．後者は，別名でシリコンコンパイラといわれる技術であり，FPGA であればデバイスの開発ベンダーが提供する CAD ツールチェイン，ASIC であれば EDA ベンダの提供する半導体設計支援ツールチェインを用いて開発される．

2.3.3.4 デバッグ/テスト/再現性

アプリケーションやシステムソフトウェア開発において品質，信頼性に大きく関わるのがデバッグやテストである．開発プログラムが仕様通りまたは意図した挙動を示すかを確認（テスト）しバグがあればそれを取り取り除き（デバッグ），それが如何なる環境においても仕様通りに動くことを保証（再現性）することは，開発プログラムの品質を担保するために必要不可欠なプロセスであり，ソフトウェアの開発時間の大部分を占める．実際，ケンブリッジ大学と Rogue Wave ソフトウェアの共同レポートによると，“On average, software developers spend 50% of their programming time finding and fixing bugs.”[130] と発表されている．

このようにデバッグ，テスト，再現性確保による “ソフトウェアの信頼性” の保証は，ソフトウェア開発には必要不可欠なプロセスであり，特に HPC における並列アプリケーションなどではその複雑性からより効率的なデバッグ，テスト環境の利用がより難しくなっている．ここでの複雑性とは，システムの大規模，不均質性（システム内のヘテロ性），多様化（多様なシステムが開発され運用），確率的コンピューティングなどが含まれる．このため大規模システム上でも効率的なデバッグやテストを支援するためのツールが多く存在する．

まず再現可能なバグ（決定的バグ）に関して，１プロセス実行のみで解決できるバグであれば， printf, dbg, Valgrind（不正なメモリアクセス検知，メモリリーク検知など） [131] で比較的容易にデバッグが可能である．もし並列実行した場合にのみ発生する場合には，DDT（Distributed Debugging Tool） [132] や TotalView [133] などの並列デバッガを利用する必要がある．

もし対象となるバグの再現が困難な場合（非決定的バグ，Heisenbug などと呼ばれる），デバッグやテストは非常に難しくなる．具体的には，テスト環境（コンパイラ，システム環境，外部ソフトウェアバージョンなど）ではバグは発生しないが，プロダクション環境では発生するものや同じ環境実行したとしても（スレッドのスケジューリングやメッセージ交換のタイミングにより）バグの発生率が極端に少ない場合などがある．特に大規模 HPC アプリケーションでは，複数スレッドおよび複数プロセスでの実行が多いため，非決定的バグが内在する確率が高くなる．このため，HPC の分野では非決定的バグに特化したデバッグ/テストツールがいくつか存在する．

具体的には，MPI レコード-アンド-リプレイツールである ReMPI [134] がある．これは，MPI メッセージ受信の順番を毎回レコードに従って，リプレイすることにより特定のメッセージ受信の順番でのみ発生するバグを再現することができる．ただ，ReMPI はバグが発生するメッセージの順番をレコードできなければ，バグを再現することはできない，このためメッセージにノイズを加え異なるメッセージ受信順の発生を促す NINJA [135] やメッセージの受信順を網羅的に探索する MPI Formal Ajnalysis ツールである ISP [136] や DAMPI [137] がある．その他，MPI プロセスの特異的な挙動を検知する AutomaDeD [138] や MPI メッセージの特異的な送受信を検知する FlowChecker [139]，また MPI の仕様に従って API が使用されているかをチェックする MUST [140] など様々な大規模並列アプリケーションのためのツールが存在する．

その他，OpenMP やマルチスレッドスレッドプログラムにおける代表的な非決定的バグに，データ競合があり，それを検知するためのデータ競合検知ツールとして Archer [141] や，様々な環境（コンパイラ，コンパイラオプション，システム）で網羅的に実行し，計算結果の比較を自動的に行うツールである FLiT [142] なども存在する．また，近年大規模計算機の計算パワーを活用して網羅的にテストを行うテスト手法として Software fuzzing という手法がある．これはテスト対象のプログラムにランダムな入力を与え，大量に実行することで，メモリアクセス違反などの致命的なバグを自動で発見する手法である．

近年，実験結果の再現性は研究発表の信頼性を担保するものとして重要となっており，ACM では Artifact Review and Bading [143] というガイドラインを発表しており，HPC の分野においてもこのガイドラインに準拠した国際会議が増えつつある．特に ACM では，Repeatability（同一研究チームが同一環境において実験結果を再現できるレベル）， Replicability（異なる研究チームが同一環境において実験結果を再現できるレベル), Reproducibility（異なる研究チームが異なり実験環境においても実験結果を再現できるレベル）の三つのレベルの再現性を設け実験結果の再現性を評価している．

2.3.3.5 データ学習・推論フレームワーク（TensorFlow, PyTorch etc.）

2012 年に CNN の AlexNet が提案されてから DNN が活発に開発されるようになり，それに伴って DL フレームワークが開発されるようになった．2013 年に Caffe が開発され，その後，様々なフレームワークが開発されている．学習・推論ともに最もよく使われるものは PyTorch と TensorFlow である．Keras は TensorFlow を含む複数のフレームワークをサポートしており，抽象化した API を提供するフレームワークである．推論向けに NN をチューニングするフレームワークとして TensorRT や Caffe2 が挙げられる．その他，Chainer, MXNet, Deeplearning4j, Microsoft Cognitive Toolkit,PaddlePaddle といった DL フレームワークがある．ここ数年のフレームワークの利用傾向を見てみると，Tensorflow や PyTorch の利用が増えて

おり，この 2 つに集約されつつある [144]．機械学習や DL のモデルを表現する為の代表的なフォーマットに ONNX (Open Neural Network Exchange) がある．

これらのフレームワークのバックエンドでは，各ベンダが提供する low-level な数値計算ライブラリ・DL ライブラリが呼ばれる．例えば，CPU(Intel 向け) のライブラリとしては MKL DNN [145]，OneDNN [146]，DNNL，GPU 向けとしては cuDNN [147]，ベクトル向けとして veDNN [148]，DL for Fugaku (理研，富士通) が挙げられる．さらに，Arm 対応の OneDNN_aarch64 や Blas 系ライブラリ，その他 Eigen，SSL-II などの開発が進んでいる．通信ライブラリには Horovod や NCCL (NVIDIA) が挙げられる．Horovod は PyTorch や TensorFlow など複数の主要なフレームワークを複数プロセスで並列実行するためのライブラリである．NCCL は NVLINK を用いる複数 GPU 間の通信向けライブラリであり，PyTorch などのフレームワークや Horovod などのライブラリが対応している．多くのフレームワークでは MPI や NCCL を用いた複数プロセスでの実行に対応している．ハイパーパラメータの自動最適化ツールとして Optuna，NNI (Neural Network Intelligence)，Hyperas，DeepHyper などがある．

DL モデルからターゲットハードウェアで動作する低レベルコードを生成するコンパイラに TVM がある．DL の HW/SW スタックに対して End-to-end で自動的に最適化するものであり，Conv2D のような Primitive な Tensor operator はチューニングされたライブラリを使用する．複数オペレータの融合による新たなオペレータを導入することで高速化が見込まれるが，HW への実装コストが大きいという問題があるため，AutoTVM も提案されている．AutoTVM はハードウェアのコストモデルを機械学習で学習することで低レベルコードを効果的な探索空間の中で自動的に探索して最適化する．TVM でハードウェアアクセラレータを扱うためのフレームワークに VTA がある．FPGA 上に高位合成で作成したアクセラレータを利用して DL を高速化することが可能である．ハードウェア生成そのものを AutoTVM と同じように機械学習で行う AutoVTA も提案されている．多数のハードウェアへのコンパイラのための共通の中間表現として MLIR (Multi-Level Intermediate Representation) がある．

機械学習向け開発環境として Jupyter や Google collab がある．これらは Web ブラウザ上でインタラクティブに開発することが可能である．クラウド上で機械学習を行うためのツールとして AutoML(Google)，SageMaker(Amazon) がある．専門家でない人でも DL の学習を行えるように自動化や抽象化が行われている．ビッグデータへの分散処理フレームワークの Spark, Hadoop 等と DL フレームワークを連携させた I/O の高速化も行われつつある．docker や kubernates などのコンテナとの連携も進められている．

その他の機械学習ライブラリとしては scikit-learn, dlib, MATLAB などが挙げられる．他にも前処理を含めた機械学習のパイプラインを最適化するものに TPOT など，多種多様なフレームワーク・ライブラリが存在する．その他，配列操作を含む計算ライブラリとして NumPy, NumPy の GPU 対応版として CuPy (PFN)，データ解析ライブラリに Pandas, SciPy，グラフ描画ライブラリ Matplotlib などがある．

2.3.3.6 データ処理・解析フレームワーク（Hadoop, Spark）

ビッグデータを分析する場合に古くはデータウェアハウスが用いられてきたが，2000 年代に入ると大規模データを並列に処理する MapReduces プログラミングモデルや Google File System といった大規模データを効率よく格納できる分散ファイルシステムが登場し，より低コストでデータ分析環境を構築できるようになった．このような大規模なデータ処理・解析フレームワークの代表として Apache Hadoop が挙げられる．Apache Hadoop は HDFS と呼ばれる分散ファイルシステムや YARN と呼ばれるジョブスケジューラおよび MapReduce フレームワークから構成される．データの分析者は Hadoop フレームワーク上で Map と Reduce 関数という単純な関数をそれぞれ実装するだけで，容易にデータの高並列処理が可能となった．

Hadoop の MapReduce 処理はあくまでバッチ処理であったが，これをリアルタイム処理に拡張したのが Storm である．Storm も Map と Reduce 関数を実装して分析処理を行なうフレームワークであるが，入出力データがストリームであることが Hadoop と異なる．Hadoop がバッチ処理の MapReduce フレームワークに対し，Storm はリアルタイムストリーム処理の MapReduce フレームワークと言える．

Hadoop はデータの高並列分散処理が可能であったものの，その処理速度は HDFS に依存するものであった．MapReduce 処理を繰り返すような場合は，Reduce 結果を HDFS に保存し，再び Map 処理で HDFS で読み込む必要がある．このようなデータ処理をメモリ上で行なうことで数十倍もの高速化を図ったフレームワークとして Apache Spark がある．Apache Spark はバッチ処理だけではなくストリーム処理も同じフレームワーク (Spark Streaming) で処理できる．また，Map 関数と Reduce 関数もデータ構造に対する API として高度に抽象化されるようになり現在の分散データ処理フレームワークのデファクトスタンダードとなっている．近年では，MapReduce 処理をプログラミングしなくても，さらに抽象的な SQL によって MapReduce 処理やリアルタイムストリーム処理が実行できる環境が整ってきている [149] [150].

HPC でも大規模なデータ処理を行いたいという需要があるが，上記のような Hadoop や Spark フレームワークは専用のファイルシステム，資源管理システムの上に構築されているため従来の HPC のシステムと共存することが難しい．また，HPC の多くはバッチ処理が前提であり，常時データを処理しつづけるようなストリーム処理は多くの HPC センターでは対応していない．このため，HPC 用のシステムとデータ処理用の Hadoop クラスタといった用途に応じて複数のシステムを運用する必要がある．一方で，HPC システムで MapReduce を行うために Hadoop のファイルシステムや資源管理システムに依存しない MapReduce ライブラリの研究開発も行われている [151]．また，HPC 上で Spark 互換のインターフェイスを実現したフレームワークもある [152].

Hadoop や Spark クラスタを構築および運用するにはコストがかかるため，これらをクラウドで運用できるサービスも登場している．これらは Google のクラウドサービスでは BigQuery，Amazon では Athena と呼ばれるサービスで提供されており，拡張 SQL によってデータ分析が可能である．

今後，AI や IoT によりビッグデータ処理の需要は更に増すことが予想され，HPC においても Hadoop や Spark などのデータ処理フレームワークと従来からのバッチ処理との共存した動作環境を整えることが求められる．

2.3.4 性能解析ツール

2.3.4.1 プロファイラ，トレーサー，性能解析方法論

アプリケーションの性能最適化においては，「高並列性能最適化」と「CPU 単体性能最適化」の二つが重要であり，それぞれについて現状認識，問題把握の 2 つの作業フェーズを進めていく必要がある [153]．現状認識については，高並列と CPU 単体性能に共通してソースコードの構造を分析し，アプリケーション性能の現状を把握するための性能測定を行っていくことから始まる．しかしながら，これらの過程においては，今日においても，目視によるソースコード構造の調査や，職人技的な手作業でのコードの解析，分析など，スーパーコンピュータのチューニングに関する熟練した技能を持つごく一部の技術者の経験や勘に基づき進められることがほとんどである．

性能解析ツールにおいてプロファイラ，トレーサ，性能解析方法論の役割は，システムの特性に向けた性能最適化に対して，これまで経験的かつ職人技的な手作業で行われてきたコードの解析，分析に関するプロセスを自動化していくことにあり，経験に基づく直観ではなく，ソフトウェア性能工学を駆使して客観的に性能向

上計画を策定し，その後のコードの書き換え，リファクタリングを含めて自動化を進めていくことが求められている．以下に現状の具体的な性能解析技術の概要を順に説明し，現状の問題点を整理していく．

プロファイラとトレーサをどのように線引きするのかは，あいまいであるが，特定のイベントの事象について時系列的に発生状況のトレースを取得するものがトレーサーであり，プロファイラはトレースしたイベントを統計情報として扱って何かの事象の分析の指標とするものであるとここでは定義する．従って，どのようなイベントの発生をどの程度の精度で監視していくかで様々な実現方法が考えられる．

最も細粒度のものとして，命令レベルのプロファイリングが挙げられる．命令レベルのイベントは CPU の動作速度のオーダーで発生するため，その全てのトレースを保持していくことは，容量の面でほぼ不可能である．そこで，特定の指標に着目して統計的に取り扱っていくアプローチがとられる．具体的には，分岐命令のトレースによる分岐予測器の改善，メモリ参照トレースによるキャッシュの解析，プリフェッチ機構の改善への応用などがある．命令レベルのプロファイリングは，通常，動的バイナリ計装（Dynamic Binary Instrumentation）により実装される．Intel Pin, DynamoRIO は動的バイナリ計装技術の具体的な例であり，数多くの性能に関するプロファイラの実装のベースとして使われている．また，命令レベルのトレースを取得し，データを一時的にも保持していくことは性能の面でも大きなオーバーヘッドとなる．この命令レベルのプロファイリングのオーバーヘッドを低減するアプローチとして，トレースの取得タイミングをサンプリングにより実施するなどのアプローチも取られている．

命令レベルよりも粒度の大きいイベントのトレースとして，関数のコールフロー，システムコール，並列実行におけるスレッドや MPI のイベントに関するトレースなどがあげられる．これらについて，トレースしたデータを実行後に解析することにより，プログラムやシステムの実行状況，同期や負荷分散状況の理解することができる．また，コード実行時にこれらの事象による性能低下が発生した場合にリアルタイムに把握できるようになると，ランタイムのスケジューリングのポリシーの動的な変更などにも活用することができる．

プロファイラやトレーサーに加えて，ソースコードレベルの構造を解析し，可視化や統計処理することにより，プログラムの理解を支援するというアプローチもある．このようなソースコード解析は，プロファイラやトレーサーのようにコードを実行することなく解析を行うことから，静的解析ツールともいわれる．ソフトウェア工学の分野の機能検証に関する研究やセキュリティ脆弱性の検出に関する研究などもソースコードを入力とした静的解析ツールとして実現されているものも多く，これらの分野のコード解析技術と融合しつつ，HPC 分野のコード品質の向上や生産性を上げていくことは長期的な課題となっている．

性能計測法という観点では，カーネルレベル単位で処理時間を計測することに加えて，ハードウェア性能カウンタを用いた CPU の挙動との相関を取るアプローチや，GFLOPS 値，GOP 値，GTEPS 値などアプリケーション実行性能の指標となるスコアを策定して評価を行う方法などがある．アプリケーション実行性能を測るうえでは，ベンチマーキングは古くから行われてきた手法であるが，Top500 で用いられる Linpack に加えて，HPC チャレンジや，Graph500，HPCG といったより実用の問題に近い形のベンチマークの利用が広まっていることや，Fiber miniapp suite, ECP Proxy App など実アプリの挙動の再現を狙ったものの開発も進められている．AI 分野でも，MLPerf などが策定されているなど，アプリケーションの広がりに歩調を合わせる形でそのアプリのスタンダードとなるベンチマークが続々と構築されてきている．

性能計測を行った後，実際の性能解析を行っていく過程においては，実行後の統計データを解析する手法の確立なども，古くから課題となっている一方で，これらの解析に機械学習や深層学習を取り入れるというアプローチも試みられている．また，トレースや性能計測においてリアルタイムのイベント監視の機能を強化することにより，リアルタイム性の高い性能データ解析が可能となるため，このような取り組みも今後進められると予想される．

■スーパーコンピュータ富岳の設計における HW カウンタからのアプリ性能推定技術　近年ではシステム導入後の早期の成果創出のために，システムとアプリケーションのコデザインが行われている．システム設計時にアプリケーションの性能を性能推定ツールにより推定し，キャッシュのサイズやコア数などのアーキテクチャのパラメータを決定する試みが行われている．例えば，スーパーコンピュータ富岳の設計においては，実機 (FX100 クラスタ) のパフォーマンス API から得られた実行プロファイルを入力とするツールを用いて，富岳でのアプリケーションの性能を推定し，アプリケーションおよびシステムの最適化を行った．また小規模なカーネルに対するより詳細な性能推定および最適化には，ベンダーの内製 CPU シミュレータおよび，オープンソースのシミュレータ gem5 を富岳向けに拡張した CPU シミュレータが用いられた [154].

2.3.4.2　性能モデル・シミュレーション

性能モデル・性能シミュレーションは，アプリケーションの最適化のみならず，システム設計時の設計パラメータの選択にも重要な役割を果たすことから，メモリ，CPU，ネットワークなどさまざまなモジュールに対して，一時近似的な性能モデルから高精度のシミュレータまで多様なツールが提案されている．

高精度なシミュレータとして代表的なものにオープンな CPU シミュレータで OS も含めたフルシステムのシミュレーションと，OS をエミュレートしアプリケーションのみの性能を評価するシミュレーションの両者が可能な gem5[155] などがある．GPU を含むヘテロジニアスなシステムのシミュレーションも可能となっている [156, 157]．シミュレーション過程で仮想的なハードウェアに対するプロファイルを取得するため，メモリや CPU リソースなどのボトルネックが細かく調査可能であるという利点がある．一方，シミュレーションには通常の実行の数万倍の時間を要することから，カーネルの切り出しなどを行わずアプリケーション全体を対象にすることは現時点では難しい．

DSL を用いてアプリケーションやシステムを記述し，これらの情報に基づいて性能モデルを行う試みも多く行われている．これは，シミュレータほどの精度はないものの，高速であり，複数のハードウェアパラメータの組み合わせを網羅的に調査するなどの目的に向いている．アプリケーションおよびシステムの抽象的な記述から性能モデルを行う Aspen[158]，多様な選択肢を持つヘテロジニアスで階層的なメモリアーキテクチャについて性能モデルに基づく評価を行う Seina[159], CPU，メモリ，そして通信などさまざまなモジュールについて評価が可能な Structural Simulation Toolkit (SST) [160, 161] などがある．SST はネットワークについてはオンラインシミュレーションも可能である．

2.3.4.1 では，性能解析においてプロファイラやトレーサが積極的に活用されている例を述べたが，将来システムに対する性能モデルにおいても，プロファイラやトレーサは重要な役割を果たす．電力推定ツールである，McPAT[162, 163] は，OoO リソースなどのアーキテクチャパラメータ，CPU シミュレータなどが出力するキャッシュなどのハードウエア利用情報を用いて，電力モデルに基づき，チップ面積と推定電力を出力する．

トレースに基づいて性能推定を行う場合，通信性能においては現在システムが想定する将来システムと比較して小規模な場合，将来システムの性能推定に必要な十分なプロファイルデータが得られないという問題があったが，たとえば SCAMP 法 [164] では実通信トレースに追加してソースコードの自動解析から作成される疑似トレースを用いることでこの問題を回避している．Performance Nodel Normal Form は，特定のパラメータを入力とし，実行時間などの性能指標に近似する関数を構築する．数式の構築に用いられるパラメータは Score-P などで取得される．[165] では単一の，[166] では複数の入力パラメータについて関数を構築する手法が示された．近年ではより効率的にモデルを構築するためにディープラーニングの技術を応用する手法も提案されている [167]．数式の構築に用いられるパラメータは Score-P[168] などで取得される．将来システムに

おいて重要な役割を果たすと考えられるプログラミングについても，実タスクと空タスクの実行結果の比較からの性能モデル化とボトルネックの同定を行うアルゴリズムが開発されている [169].

前述の CPU シミュレータに加えて，近年では，量子計算機を GPU システム上でシミュレートする Qulacs[170] など，ポストムーアに向けてさまざまなデバイスが開発をサポートする性能ツールやシミュレータが提案されている.

また，アプリケーションの多様化に応じて，仮想環境やコンテナの利用も広がってきている．しかし，実行環境はハードウェアの共有状況やアプリケーションの特性に応じて状況が動的に変化するため，性能モデリングやシミュレーションによって決定的な情報を得ることが難しい.

2.3.5 利用高度化ツール

2.3.5.1 省電力化システム

アプリケーション実行時の消費電力を削減あるいはマネジメントするためには，アプリケーションの特性と同時に，それを実行するシステムおよびハードウェアの特性も併せて考慮しなくてはならない．さらには，実際に消費電力の削減あるいは調整が可能なハードウェアの要素（電力ノブ）を実際に操作するためのインタフェースが必要となる.

HPC システムを対象とした省電力化技術では，主として消費電力のキャッピング [171]，プロファイリングツール，アプリケーション実行時の消費電力に関するモデリング [172, 173]，HW の製造時ばらつきに対する対応 [174]，異なる HW 要素間（CPU-DRAM 間 [175]，CPU-GPU 間 [176]，メモリ階層間 [177] 等）での電力融通等に対応した技術が提案されている．しかし現状では，それぞれ個別の要素技術として提案されているところである.

この状況を考慮して，システム全体を視野に入れ，大規模システム向けの省電力技術をを整理統合していく取り組みも徐々に現れ始めている [178].

2.3.5.2 耐障害技術

大規模高性能計算は多くの計算リソースを使用することのより膨大な計算パワーを利用することが可能で，例えばより詳細なシミュレーションをより高速に実行することが可能となる．しかし，大規模計算機ではそれらを構成する計算ノードの増加により全体の部品点数が増加しシステムの信頼性を維持するのは容易ではない．例えば１つの計算ノードあたりの平均故障間隔（MTBF）が 10 年であったとしても，この計算ノード１０万台集めた大規模システムでは，MTBF はわずか 53 分程度 (= 10[year] x 365[days/year] x 24[hours/day] x 60 [mins/hour] / 100,000）である．これは平均して 53 分に一度どこかの計算ノードで故障が発生することを意味する.

実際に公表されているスーパーコンピュータの故障頻度を参照してみると，理化学研究所 計算科学研究センター（旧：計算科学研究機構）に設置されていた京コンピュータでは，CPU の MTBF が 7.33 days である．これは AFR（Annual Failure Rate）=0.06% とその部品 (CPU) 点数（以下#:で示す）=82,944 から見積もったものである．続いて，DIMM の MTBF が 34.4 days (AFR: 0.016%, #:663,552) [1] となっている．一方，イリノイ大学の構内にある National Center for Computing Applications(NCSA) に設置されている Bue Water では，CPU の MTBF が 3.33 days (AFR: 0.23%, #:49,258), DIMM の MTBF が 1.65 days (AFR: 0.112%, #:197,032)，GPU の MTBF が 6.86 days (AFR: 1.732%, #:3072) となっている [179]．中国江蘇省無錫市の国立スーパーコンピューターセンターに設置されている神威・太湖之光（英語

表記：Sunway TaihuLight）では，CPU の MTBF は時期にも影響するが概ね 4〜10days である．これらは，実際にハードウェアが物理的に故障し求められている QoS を供給できない障害を集計した，いわゆるシステム障害（System Failure）の統計情報である．

一方，システムの問題（ハードウェアとソフトウェアのいづれかの影響）により実際にアプリケーションが停止した回数を集計した情報（アプリケーション障害; Application failure）も公表されている．Lawrence Livermore National Laboratory (LLNL) が行った実験によると，レーザープラズマ相互作用アプリ (pF3D) を Coastal, Hera, Atlas の 3 つの大規模システムにおいてそれぞれ 283 万ノード時間，143 万ノード時間，137 万ノード時間実行した場合のアプリケーション障害による MTBF がそれぞれ 115 hour，64 hour， 33 hour と見積もることができる．また，アメリカ合衆国オークリッジ国立研究所に設置されている Titan では 112,000 計算ノードでの実行では 95.5%，4,000〜112,000 計算ノードでの実行では 61.9% のジョブが RAS(Reliability, Availability and Serviceability) に関わるイベントが発生し，また 24 時間以上の実行の 79.55%，8〜24 時間以上の実行の 49.6% のジョブが同様に RAS に関わるイベントが発生したと報告されている [180]．このようにアプリケーションの大規模実行や長時間実行での障害発生がいかに多く発生しているかがわかると同時に，これらのジョブ実行は容易ではないことが伺える．その他，近年問題視されているものとして無兆候データ破損（Silent Data Corruption; SDC）がある．これは宇宙線が誘発する中性子線によるビット反転が発生し，これによりアプリケーションのハング，収束問題や計算結果の誤りを引き起こす恐れがある．このようにハードウェアの故障自体（System failure）は比較的高信頼を保っているが，不具合によるアプリケーションの停止に着目すると無視できない頻度であることがわかる．

このためアプリケーションが停止したとしても実行を継続する技術が重要である．HPC の分野において広く利用されている技術としてチェックポイント・リスタート（C/R）がある．これはアプリケーションの計算の中間結果（チェックポイント）を信頼性の高い永続化ストレージに定期的に書き出し，もし障害が発生した場合，最新チェックポイントから中間結果を読み出しそこから計算を再開（リスタート）する手法である．実装方法として，システム側がアプリケーションのメモリイメージ全体をダンプするシステムレベルチェックポインティングとアプリケーション開発者が予めどの変数をチェックポイントするかをプログラム上で指定してチェックポイントを行うアプリケーションレベルチェックポインティングがある．前者のシステムレベルチェックポインティングでは BLCR，後者のアプリケーションレベルチェックポインティングでは，Scalable Checkpoint/Restart (SCR)，Fault tolerance interface (FTI) や VeloC が広く利用されている．その他にも Docker や OpenVZ といったコンテナシステムにも統合されているプロセスの C/R ツールである CRIU[9] や KVM などの VMM による VM の C/R 機能もある．

しかし，C/R では中間結果からリスタートすることを可能とするが，アプリケーションは停止してしまう．そのため，障害が発生した場合ユーザ側で再度ジョブをサブミットする必要がある．これは，MPI では MPI プロセスのうち一つでもプロセスが停止した場合，全ての MPI プロセスを停止するように作られているからである．この MPI の耐障害機能の拡張を加え，一部のプロセスが障害などにより停止したとしても，障害が発生していない MPI プロセスは引き続き実行可能にした MPI として ULFM (Fault Tolerant MPI) がある．さらに C/R と ULFM を組み合わせることにより，障害が発生したとしても自動的に回復し継続してアプリケーションを実行することができる Fenix[10] なども開発されている．

その他，C/R の高度化のための研究として，データ圧縮（SZ [181], ZFP, FPZIP, TEZIP），C/R 機構の最適化（モデリングによる最適化，AI を使った最適化）や障害予測と組み合わせた最適化などもある．障害予測による最適化とは具体的に，システムログなどから障害の発生を予測し [182]，それに基づいて実際の障害発生よりも前に ジョブのマイグレーションやチェックポイントの取得，障害発生するノードの隔離といっ

た回避処理 (Proactive action) を実行することで，アプリケーションが障害に巻き込まれずに実行を継続させる手法である．

また SDC 対策として SDC の検知を可能とするツールや [183, 184] や数値計算のアルゴリズムに SDC 検知・修正を加える ABFT（Algorithm-based fault tolerance）技術の開発 [185, 185] も盛んに行われている．

2.3.5.3 ディスアグリゲーション技術

ディスアグリゲーションとは，既存のモノリシックサーバを，高速なネットワークを介して通信しあう単一のリソースを包含する多数のプールに分割する，新しい計算パラダイムである [186, 187, 188, 189, 190, 191]．このモデルでは，プロセッサやメモリなどの緊密に結合されたハードウェアリソースを分解し，柔軟で動的なハードウェア構成を持つ論理的なサーバ群の構成を可能にする．また個々のリソースをそれぞれ独立して拡張，進化，管理することが可能となる．ディスアグリゲーション技術の必要性として，データセンター・HPC システムにおけるアプリケーションの利用資源量がまちまちであることが挙げられる．例えば NERSC HPC センターのアプリケーションの半数は，計算ノードのメモリの 20% 以下しか用いていないと報告されている [192]．一方，データ分析と機械学習のワークロードにおいては前例のない規模のメモリ容量が必要とされる．メモリモジュールがプロセッサと一体となっている従来のシステムをベースとしたサーバプラットフォームでは，ピーク時の計算に対するメモリ容量の比の不均衡は増加する．そのような状況でサーバ構成要素の利用効率を大幅に向上させるために，VM やコンテナによる計算機分割だけでなく，ディスアグリゲーションが注目されている．

さらに，構成要素が物理的に分散し，一部分の故障（電源消失など）が直ちに論理サーバ全体に波及しないことから，OS レイヤや各ハードウェア構成要素などのホットスワップ対応により，耐故障性が向上すると期待される．

ディスアグリゲーションの実現の難易度は，対象となる構成要素によって大きく異なる．たとえばストレージについては，ネットワークファイルシステムや分散ファイルシステムなどにより，ディスアグリゲーションが実現できていると考えることができる．またアクセラレータに関しても，NVIDIA GPU を対象にした rCUDA[193] などの，リモート GPU をローカル GPU であるかのように利用可能とする技術が提案されている．一方で，CPU-メモリ間や CPU-CPU 間のディスアグリゲーションは，必要となる低遅延・高帯域の制約の厳しさなどから，特に挑戦的な課題である．

CPU-メモリ間のディスアグリゲーションに関する既存研究の多くは，ハードウェアや OS に大きな変更を加えないことを前提に，OS ページングやハイパバイザの機能を用いてメモリのマッピングを再構成している [194, 195]．しかしながらこれらの手法はページフォールトなどのためのオーバヘッドが大きい．さらに，CPU-CPU 間のディスアグリゲーションの実現は困難である．機械学習や科学技術シミュレーションを含めた多彩なアプリケーションにおいて良好な速度性能・スケーラビリティを実現するためには，ハードウェア・OS などのレイヤを巻き込む必要がある．

ディスアグリゲーションされたシステムの核となるハードウェアコンポーネントは，計算ブレード群，低レイテンシのプログラム可能なネットワーク，大容量を持つメモリブレードである．このうち特にネットワークの技術革新により，高性能なディスアグリゲーションの実現可能性が飛躍的に高まっている．より具体的には，光インターコネクトが成熟しつつあり，ディスアグリゲーションに必要な低遅延・高帯域の実現が期待されている [7]．またこれまでのノード間通信ではなく，メモリアクセスやコヒーレンシのセマンティクスの通信のための規格も CXL や Gen-Z などとして整備されつつある [196, 197]．一方で実際に稼働するディスアグリゲーションシステムを構成するためには，6.2 節で述べるようにシステムソフトウェアのさらなる研究開発

が必要である．

2.3.6 資源管理

2.3.6.1 ジョブスケジューラ

従来の HPC システムではバッチジョブによる共有計算機の利用が主であるが，2028 年の HPC システムは様々な特性を持つアプリが仮想化された資源上で実行される．また，ジョブの時間特性・空間特性も大きく異なる物となる．

従来のバッチジョブでは予め計算資源量・計算時間を申告し計算資源を予約する利用が主であるが，これらの通常ジョブと合わせてリアルタイムジョブによる計算資源利用が想定される．リアルタイムジョブはジョブ実行の要求後，即座にジョブ実行が開始されるジョブを想定する．例えば，地震やゲリラ豪雨等の緊急ジョブがあげられる．また，センサデータ等から取得されるストリーミングデータを処理するストリーミングジョブがあげられる．さらに，生体情報等のセンサデータの扱いには，高いセキュリティレベルが求められる．また，何らかのイベント発生に応じてトリガされ，簡単な計算を行うファンクションジョブによる計算資源利用が想定される．また，マイクロサービスのように多数のサービスが協調しつつ計算を行うジョブも想定される．

従来のジョブは実行ファイル形式のアプリや Python コード等が計算ノード上で直接実行されるが，2028 年度の HPC システムでは異なる仮想化レベルの仮想計算機資源によってジョブが実行される．コンテナ仮想化を用いたジョブや，ベアメタル，ユニカーネル，VM による資源利用が考えられる．コンテナ仮想化を用いることで一貫したソフトウェア環境を構築し，環境設定にかかるコストを抑えることができる．さらに，CPU，メモリ，ストレージ，ネットワークの各リソースを OS レベルで仮想化するため，他の利用者とセキュアに分離された資源を利用できる．また，VM による利用も想定される．さらに，一部のユーザーは OS を含む環境を自前で準備することを要求しており，ベアメタルによる計算機利が重要となる．また，ユニカーネルによる利用が想定される．ユニカーネルを利用することでシステムソフトウェアをスリムにすることができ，OS オーバヘッドの削減・セキュリティリスクの低減といった要求に答えることができる．

これらの，アプリと仮想化動向の変化は時間特性を大きく変化させる．従来のジョブは，数分程度の短時間ジョブ，数時間程度の通常ジョブ，数日程度の長時間ジョブが想定されるが，実行時間が 1 秒に満たないサブミリジョブによる計算資源利用が必要となる．ストリーミング処理では入力データに対して簡単なフィルタを適用することが多く，サブミリジョブとなりうる．また，マイクロサービスでは，1 つのサービスの計算量が少ないことが多く，同様にサブミリジョブとなりうる．

アプリと仮想化の変化は空間特性も変化させる．従来はジョブ実行の際に利用ノードを申告し，ジョブ実行中は基本的にノード台数は変化することは無かった．しかし，リアルタイムジョブは入力されるデータ量，データ内容に応じて計算内容が変化することがあるため，利用するノード台数も動的に変化する．マイクロサービスの場合，リクエストに応じて計算を行うサービスが異なる．

2.3.6.2 システム全体の電力マネージメント

HPC システムの性能向上を阻害する要因として消費電力の増加が大きな問題となっており 2028 年の HPC システムにおいても省電力化・電力効率の改善は重要である．演算性能の向上のためにはアクセラレータを用いたシステムが普及することが考えられる．また，HPC システムに用いられる CPU やアクセラレータの省電力化が進み，アイドル時の消費電力が大きく削減されることが想定される．電力の差が大きくなることにより従来の HPC システムでは問題とならなかったことが課題となる．

2.3.6.3 リアルタイム処理

HPC の主な利用用途は古くはバッチ処理による数値計算であったが，近年では IoT やビッグデータ，AI などの膨大なデータおよび計算が必要な分野へも HPC の利用が進んでいる．このような利用の拡大に伴い，従来のバッチ処理だけでなく計算結果の価値が時間の経過とともに減少するような時間に制約のある計算への需要も増している．防災減災分野では，各地の IoT センサーから気象や地震などリアルタイム観測網からのデータを集め，HPC を用いてリアルタイムに環境を予測をしたいといったニーズや，AI 分野では対話的・探索的に HPC を利用したいというニーズがある．これらは従来型のバッチ処理一辺倒の資源管理システムでは実現することができず，今後，バッチ処理以外の HPC 利用形態に対応した新たな資源管理の手法が求められている．

2028 年にはバッチ処理以外にも，リアルタイム処理やイベント駆動型処理およびストリーミング処理がさらに求められると予想される．

2.3.7 外部資源連携

アプリケーションやデータの多様化に伴って，ネットワークを介して接続されるクラウドをはじめとする外部の計算資源とスーパーコンピュータを連携させて利用するニーズが高まっている．例えば，オンプレミスとクラウド上の計算資源を連携させて利用することによりシステムの可用性や処理能力を向上させるハイブリッドクラウドが，今後ますます普及することが予想されている．ハイブリッドクラウド環境を構築するため，オンプレミスやクラウド上の計算資源を制御するためのマルチクラウドコントローラや，オンプレミスの計算機上の過剰な処理をクラウドに移動して実行するクラウドバースティングを可能とするソフトウェアが開発されている [198, 199].

また，IoT プラットフォームを実装するためのソフトウェアが開発され，センサ等からクラウドにデータを収集するためのクラウドサービスも登場している [200, 201, 202]．エッジコンピューティング環境と合わせて IoT プラットフォームは今後さらに普及することが予想されている．

学術ネットワークの通信帯域は，世界各国で 400Gbps への増速が進められており，我が国の学術情報ネットワーク（SINET）においても，2022 年に通信帯域が 400Gbps に増速する他，その後 800Gbps への増速も検討されている．広域ネットワークの持つ通信帯域を活用して高速にファイルを転送するためには，広域ネットワークの特性を理解した上で効率良くデータを転送する技術が必要であり，複数の TCP コネクションを制御することでこれを実現する技術が開発されている [203, 204, 205].

2.4 数値計算ライブラリ／ミドルウェア／アルゴリズムの技術動向

2.4.1 数値計算ライブラリ

数値計算ライブラリは, 科学技術アプリケーションプログラムにおいて共通に使われる数値計算プログラムをライブラリの形にまとめたものである. 数値計算ライブラリには以下のルーチンなどが含まれている.

- 線形方程式
- 固有値問題
- 高速フーリエ変換

- 乱数

大規模数値シミュレーションを行う際には, 数値計算ライブラリの性能が実行時間に大きく影響することはいうまでもなく，大規模なシミュレーションコードの作成・保守・継続利用の面からも重要な役割を果たしている．現在，富岳等のエクサスケールスパコンにおいて高い実行効率を達成できる数値計算ライブラリを実現することを目標として，各国のエクサスケールプロジェクトにおいて各種ライブラリが開発されている．2028 頃に登場するスパコンシステムにおいてもその様相は変わることはない．2008 年頃，現在のエクサスケール級システム開発前に挙げられた，実現すべき機能として以下 2 点を強調していた．

- アプリケーション開発者および利用者からエクサフロップス級マシンの複雑なシステム構成が見えないように抽象化されたインターフェースを提供すること
- エクサフロップス級マシンの高い性能をできるだけ引き出すようなアルゴリズムおよび実装手法を用いること

これらは，2018 年でももちろんだが，2028 年頃のスパコンシステムのための数値計算ライブラリが実現する機能としても継続して実現すべき機能である．また，2008 年の SDHPC の HPCI ロードマップにおいて，2018 年頃の数値計算ライブラリ構築において挑戦的な技術課題として以下の 10 項目が挙げられていた．

1. 通信最適化
2. 通信回避
3. 高精度計算
4. 混合精度演算・精度保証計算
5. 精度をユーザが確認
6. フォールトトレラント
7. ソフトウェア的階層構造
8. HW 自動選択・情報収集
9. ヘテロジニアス環境
10. 高生産性のためのフレームワーク

当然多くの技術課題は 10 年間の間に成果として数値計算ライブラリに取り込まれ，また新しい課題が出現している．以下代表的な数値計算ライブラリについて，その概要と近年の動向をまとめる．

2.4.1.1 高速フーリエ変換

高速フーリエ変換（Fast Fourier Transform，以下 FFT）は科学技術計算において今日広く用いられているアルゴリズムである．

■現実的な計算規模や利用形態 N 点の複素数 FFT に対する演算量は $5Nlog_2N$，記憶容量は $32N$（入力と出力を別の配列にする場合）であるため，現実的な計算規模は記憶容量によって決まる．1 PFlops の実行性能を仮定した場合の 32768^3 点の 3 次元複素数 FFT の実行時間は約 8 秒であるため，実行時間の制約はほぼない状況である．一方で 32768^3 点の 3 次元複素数 FFT を計算するためには 8 PiB の記憶容量が必要になるが，これは「富岳」の総記憶容量の約 1.65 倍である．FFT の現実的な利用形態としては，もう少し小さい（例えば 4096^3 点）サイズの FFT を何度も繰り返して計算することが想定される．このようなサイズの 3 次元 FFT を並列化する場合，三つの次元（x，y および z 軸）のうちの一つの次元のみで分散すると，各軸にお

けるデータ数は MPI プロセス数以上となる必要がある．この制約を解決するためには，2 次元分割や 3 次元分割を用いる必要がある．

■通信最適・通信回避について　分散並列環境における FFT の実装においては，ノード間通信に全対全通信が用いられており，本質的に通信を回避することができない．全対全通信を行うためにいくつかのアルゴリズムが知られているが，ring アルゴリズムでは MPI プロセス数に比例して通信レイテンシが増大すること，メッセージサイズが MPI プロセスの二乗に反比例して小さくなることから，ノード数が数万のオーダーになると通信レイテンシが支配的になることが知られている．この問題に対処するために，二段階で全対全通信を行うアルゴリズムが提案されているが，通信量は ring アルゴリズムの二倍になる．つまり通信レイテンシと通信量はトレードオフの関係にあるため，通信量を増やしてでも通信レイテンシの影響を削減して通信時間を削減する通信方式が考えられる．また，MPI の one-sided communication を活用した通信最適化が考えられる．最近の大規模システムでは演算性能の向上率に比べて通信バンド幅の向上率が低くなっていることから，全対全通信に要する通信時間が全実行時間の大半を占めている．したがって，FFT において演算と通信をオーバーラップすることは以前のように効果的ではなくなっており，この傾向は今後も続くと予想される．また，演算量を増やしてでも通信量やメモリアクセス回数を削減するアルゴリズムとして，高速多重極展開方法（Fast Multipole Method，FMM）を用いて通信量を削減する手法，三角関数の値をテーブルにせずに on-the-fly で計算することが考えられる．

■高精度演算・混合精度計算・精度保証計算　FFT では重み付き和を計算しているため，データ数が増えるに従って情報落ちなどの影響により精度が低下することが知られている．FFT において高精度演算が必要であるのは多倍長乗算のような特殊な場合に限られており，科学技術計算では倍精度または 4 倍精度で計算すれば十分である．また FFT は反復解法ではないため混合精度計算によって計算時間，記憶容量や消費電力を削減することは難しい．また精度保証計算を用いた FFT アルゴリズムが提案されているが，並列化を行った例についてはまだ知られていない．

■HW 自動選択・ヘテロ　FFT はメモリ律速なアルゴリズムであり，GPU が持つ高いメモリバンド幅性能を活用することで性能改善を図ることができる．これまでに GPU を用いた FFT ライブラリが公開されている．

■ライブラリ開発動向　以下にライブラリ開発動向まとめる．

- 単体ノードに対応：
  - 国内：FFTE（筑波大）
  - 国外：FFTW（MIT），FFTX，SPIRAL（カーネギーメロン大），Intel MKL，IBM ESSL，AMD rocFFT，NVIDIA cuFFT
- 複数ノードに対応：
  - 国内：FFTE（筑波大），NUFFT（東工大）
  - 国外：FFTW（MIT），FFTX，SPIRAL（カーネギーメロン大），P3DFFT（サンディエゴ州立大），heFFTe（テネシー大），Intel Cluster MKL
- ハイブリッド CPU-GPU アーキテクチャの複数ノードに対応：
  - 国内：FFTE（筑波大），NUFFT（東工大）
  - 国外：heFFTe（テネシー大），AccFFT（テキサス大）

2.4.1.2 連立一次方程式

複数の未知変数間の線形の等式関係から構成される連立一次方程式は，方程式の離散化において時間発展の陰的スキームを選択することで必ず出現する．線形代数計算の最重要コアである．応用範囲は極めて広く，微分方程式の陰的解法，非線形方程式の線形近似（いわゆるニュートン法）による求解など，計算科学の創世記から扱われてきた問題である．

■現実的な計算規模や利用形態　連立一次方程式の求解ルーチンは，数値線形代数の分野では「線形ソルバ」と呼ばれることがある．一般に線形ソルバは係数情報を行列・ベクトルの形式で取り扱い，行列の「疎性（非ゼロ要素の出現度）によって，「密行列」「疎行列」と分けられる．

「密行列」では行列の要素を全てコンピュータ上に保持するため，大規模問題を扱うことは難しく，HPL などのベンチマークを除けば実用上は数万から数 10 万次元の問題を分散並列計算もしくは単一ノード上のスレッド並列環境で求解することが通常である．さらに，密行列の数値解法は $O(N^3)$ となるため，計算コストの観点からも負荷のかかるものに分類される．「富岳」などのハイエンド環境を用いれば, 数 10 万次元の問題を数分以内に解くことが可能である．2028 年ごろのシステムで能力の増強に応じて大規模問題を解くことも可能であるが，要求メモリが $O(N^2)$，計算量が $O(N^3)$ であることを考慮すると，10 倍規模の問題でも現実的な時間では取り扱いにくくなることが考えられる．更に，計算精度の限界や丸め誤差の影響も考慮しなくてはならない．その意味では，低ランク近似などの手法やスパース化を行い，「疎行列」解法を用いた近似解法を積極的に進めることも有用になる．

「密行列」と対をなす「疎行列」解法は，基本的に非ゼロ要素のみをメモリ上に格納し，行列を表現する．したがって，大規模な問題を扱うことが可能である，100 万から数 100 億次元までシステムの搭載メモリ量に応じて計算がなされる．多くの場合で，要求メモリ量は次元にほぼリニアであるため，2028 年ごろの搭載メモリ量の増加次第では 10 倍規模の大規模問題の実現性は大いに考えられる．疎行列ソルバには「直接解法」と「反復解法」の 2 種類の流れが存在する．前者は，密行列と同様の解法を疎行列構造に適用するものである．後者の「反復解法」は大きく分けて，定常反復法・共役残差法・共役勾配法・マルチグリッド法・その他が現状よく使われている．反復解法は行列・ベクトル積が実装上の中心となることが多く，インハウスコードにおいて独自実装するケースが多く見うけられる．反復法は，行列の条件数によって反復の振る舞いが大きく変化する．前処理を伴って反復の安定性や収束の加速をおこなうことが多く，前処理技法の研究も盛んである．一方，疎行列直接ソルバは実装が極めて困難であるが，悪条件行列でも求解可能なことがあり，問題の性質によっては直接ソルバが必須となる場合がある．

■高精度演算・混合精度計算・精度保証計算　線形ソルバも高精度な計算結果が必要な場合があるが，1) 高精度演算フォーマットの利用, 2) 反復改良による精度改良を行う高精度ソルバなどが存在している．また，計算結果に精度を要求しない前処理部分に低精度演算を積極的に採用した混合演算ソルバも出現している．今後は，ハードウェアが得意とする精度（低精度，普通精度，高精度）を自動チューニング技術等を活用し，自動的に演算精度選択する線形ソルバの出現が期待されている．

■ソルバ開発動向　既存アーキテクチャに対する，計算原理はほぼ実装がなされており，求解が困難な問題への更なるアプローチや異機種間での統一インタフェースの整備などが，現状ソルバ開発のトレンドと考えられる．また，AI（機械学習）を活用した計算手法の導出や，準最適な前処理選択，自動運転に相当するような反復法の自動制御技術も重要である．また，欠損情報の補完技術や逆のスパース化技術の活用も重要である．一

方で，量子計算機や新計算原理に基づくアーキテクチャに対しても線形ソルバ技術の開発は必要となる．

主なソルバ開発動向をリストアップすると以下のようになる．

- ソルバ実装
  - 問題個別化と汎用化の２極: 国内のヘビーユーザや重点課題ユーザは個別にソルバ部分を開発しアプリに組み込んで開発することが多い．アプリ独自のデータ構造や管理との相性も関連すると思われる．
  - 汎用：(密) LAPACK, ScaLAPACK, PLASMA, SLATE, MAGMA
  - (疎反復系) kokos, PETSc
  - (疎直接ソルバ)SuperLU, PARDISO, MUMPS, Dissection 汎用の反復系ソルバは，構造格子や非構格子の区別をせずにデータ管理や袖通信などを利用者から隠ぺいするものが多く，ライブラリは解法の内部特性を向上させ，利用者はシミュレーションそのものの特性部分の強化に専念させる方向にある．
- spMV(疎行列ベクトル積カーネル) のみを取り出した高性能化技術の追求
  - MPK (Matrix Powers Kernel)
  - CA(Cmmunication Avoidance) 技術
  - 行列 format
  - H 行列技術との連携
  - FMM も将来的には関連
- 前処理
  - 軽量化と重量化の２極化
    - 軽量 Jacobi, Red-Black, SOR: 前処理単体を高速化し，反復回数の増加分にも対応する．
    - 重量 AMG, GMG：１回あたりの前処理は高価だが，非収束性の改善や全体の反復回数を削減する．
  - 混合精度 (FP16 や INT8 など更なる低精度演算の利用)
  - (実装) Hypre など
- 新しいプログラミング技術・実装技術との連携
- メーニイコア・分散環境下でのタスク並列・効率的な動的スケジュールなど
- Fortran/C/C++ 以外の言語処理系との柔軟な連携
- JIT などの実行時オンラインコード生成技術

2.4.1.3 固有値計算・特異値分解

固有値計算は線形代数計算の重要なコアである．主に量子力学分野の計算物理や計算化学分野で使われることが多く，応用分野は極めて多岐にわたる（固有値計算が，量子力学計算の主要部分となる理由は，シュレーディンガーの波動方程式から導かれるエネルギー関係式 FC=SC ε が一般化固有値問題の関係になっているからである（実際は，非線形の関係式であるため，SCF などの自己無撞着な計算により反復計算することが多い））．また，実用上は固有値計算と特異値計算は一体的な扱いをすることが多く，社会科学やネットワーク工学におけるデータ解析分野で利用されている．近年では，テンソルの次元縮約 (TensorNetwork や TensorNormalization) のアルゴリズム内部で重要な役割を果たしており，情報抽出や圧縮技術のコア技術となっている．

■現実的な計算規模や利用形態　密行列の固有値計算の計算コストは $O(N^3)$ であり，数値解法の中では比較的高コストに分類されるが，HPC 技術の応用により「京」コンピュータ上の全システムを使用して $N=10^6$ の計算を 1 時間で達成することができる．「富岳」規模の計算機を利用することで $N=10^6$ 以上の計算が現実的な時間で可能となる．2028 年のシステムでは $N=10^6$ 程度の固有値計算はメモリが許せば，1000 ノード以下の小規模ノードでも計算可能になると考えられる．小規模問題の需要も当然高くなることが予想されるが，各種レイテンシの影響で問題単体での高速化には限界がある．バッチ処理やマルチタスク処理内で大量の固有値計算タスクを処理することや，行列積のようにプログラミング言語の標準的な組み込み関数となる可能性もある．

■通信最適・通信回避について　密行列固有値計算のボトルネックはメモリバンド幅と通信レイテンシであることがベンチマークからも判明しており，次世代スパコンにおいて問題が顕著化する可能性が高い．前処理計算でもある密行列の Householder 三重対角化で，通信回避アルゴリズムが考案されある程度のボトルネック解消がなされているが，ネットワーク性能の向上がプロセッサ性能を大きく下回る現状では，さらなる通信最適化・回避策の提案が必要となる．密行列系アルゴリズムにおいても，特定のプロセスグループとの定型的な非同期的通信で実現可能なアルゴリズム開発が必要である（例えばブロックヤコビ法など）．疎行列計算では，MPK(Matrix-Powers Kernel) の技術による行列ベクトル積の通信回避アルゴリズムが有効であるが，階層的なメモリ階層がより深く速度格差が大きくなることが予想されため，伝送路に応じたデータ移動と動的タスク割り付けを同時に考慮した通信最適化・通信回避アルゴリズムの実現が必要となる．

■高精度演算・混合精度計算・精度保証計算　固有値計算も高精度な計算結果が必要な場合があるが，1) 高精度演算フォーマットの利用, 2) 反復改良による精度改良を行う高精度ソルバなどが存在している．マルチデバイス・マルチプラットフォーム化が進み，ハードウェアが得意とする精度（低精度，普通精度，高精度）による固有値計算の加速が期待される．同様に，内部アルゴリズムで混合精度計算を組み合わせ，通信最適化，さらには消費電力削減などにつなげる方法が今後出現すると期待される．

■耐故障性　対称行列の固有値計算はレイリー商の変分計算ともとらえられ，ある程度の誤差耐性アルゴリズムの研究も期待できる．更に，多項式や周回積分型フィルタを用いることで，欠落した部分空間基底の識別・除去技術も提案されており，高度なフォールトレジリエンスを保証した固有値ソルバ構成の可能性は高い．

■HW 自動選択・ヘテロ　近代的なハードウェアに適応した固有値ソルバは，その完全系が出現しているわけではない．スレッド並列，GPU, MPI 並列などの各種並列処理階層のソフトウェアを効果的に組み合わせた利用は可能であるが，複数のソフトウェア利用が不可欠でありインタフェースやプログラミングモデルの違いを如何に吸収して統一的に扱うかが課題である．

■高生産性　固有値ソルバは内部アルゴリズムや対象となる計算機プラットフォームの変化に対応すべく，Fortran 以外にも C, C++ などの近代的言語への展開が早い．特に, Eigen3 や Elemental などでは，テンプレートを活用した抽象的実装により，計算精度や様々な内部アルゴリズムの実装の効率化が図られている．固有値ソルバに限らず同様のフレームワーク整備は将来の数値計算ライブラリにおいて共通の課題である．

■新アーキテクチャ対応　量子計算機向けアルゴリズム：「位相推定アルゴリズム」と「変分型アルゴリズム」など考案されている．前者は，量子状態の重ね合わせから固有値を乱託する量子アルゴリズムであり，後者は，量子計算機と古典計算機のハイブリッド型探索アルゴリズムである．

■ソルバ開発動向

- 国内：EigenExa（密行列: 理研）, z-Pares(疎行列: 筑波大).
- 国外：ELPA（密行列: ドイツのグループ), DPLASMA, SLATE, ScaLAPACK, PLASMA, MAGMA(密行列: テネシー大学), Elemental(密行列), SLEPc(疎行列), Trillinos+Kokos や FASTEig(疎を中心), 定番ではあるがエクサ以降のメンテが不明な ARPACK, PARPACK などもある.

2.4.2 高次データ・テンソル計算関係

2 個以上のインデックスを持つ大容量データ（テンソル）の解析技術=高次元データ解析も重要性が増しており，特異値分解の自然な拡張である高次特異値分解を利用した主要情報の抽出や情報圧縮がこれに関連する．また，単純な分解だけでなく，様々な制約条件を加えた分解手法も存在し，社会科学やネットワーク情報抽出に利用されている.

2.4.2.1 低ランク近似技術

重力の多体問題等の N 個のオブジェクト間に働く相互作用を扱う解析では，一般に相互作用は N × N の密行列で表現される．しかし，自由度 N に対して $O(N^2)$ の計算量や記憶容量が必要な解析は大規模化が難しく，スケーラブルな解法とならない．ここでクーロン力や重力のように相互作用の大きさがオブジェクト間の距離の 2 乗に反比例するようなケースでは，その性質を使うことで計算量を削減することができる．本技術の典型的な手法が高速多重極展開（FMM）法やツリー法，H 行列，H2 行列である．ここではこれらの中でライブラリとしての実装が容易な H 行列について述べる.

H 行列は，密行列の近似手法の一種である．対象とする行列を部分行列に分割し，部分行列の一部を低ランク近似することで演算量や記憶容量を低減する．低ランク近似手法には，ACA (Adaptive Cross Approximation) やランダマイズド SVD 等が用いられている．H 行列は応用上 H 行列・ベクトル積の形で用いられることが多い．即ち，本来密行列ベクトル積演算が必要な応用で，密行列を H 行列に変換することで大幅に演算量や記憶容量を低減することができる．理想的な場合，H 行列を用いることで，行列ベクトル積の演算量を $O(n^2)$ から O(n logn) に低減することができる.

■現実的な計算規模や利用形態 H 行列を用いた場合，100 万自由度程度の境界要素解析を大きな計算機資源を必要とすることなく実行することができる．H 行列を利用する場合に留意すべき点として，近似の対象となる密行列を実際に構築することなしに H 行列を生成する必要があるということがある．解くべき問題や H 行列の利用形態によっては一旦，密行列を生成することもありうるが，その場合一時的に $O(N^2)$ の記憶容量，演算量が必要となる．従って，大規模解析を指向して H 行列を用いる場合を想定した場合，H 行列のライブラリへの入力は対象とする密行列の要素を計算する関数であることが望ましい．また，このような大規模解析では，実際に密行列を生成して，近似の精度を比較するというのは事実上，不可能である場合が多い．従って，応用上の解析精度が十分であるかはライブラリの利用者が十分に気を配る必要がある.

■通信最適・通信回避について 分散並列環境での実装を考えた場合，H 行列の生成過程において，部分行列の低ランク化については部分行列毎に独立に実行可能であり，通信の影響は低い．一方，H 行列ベクトル積については，メモリ律速なアプリケーションであり，分散並列実装では通信コストが支配的となるケースが生じ

得る．Lattice-H 行列では，全体行列を 2 次元分割し，その各部分において H 行列を生成することで，H 行列が持つ演算量と記憶容量のオーダーを変えずに通信の最適化を図っている．

■高精度演算・混合精度計算・精度保証計算　H 行列の部分行列の一部を低精度化したり，部分行列内のデータを混合精度化したりすることによる解析の高速化に関する報告がなされている．H 行列はそれ自体が密行列の近似であり，上述の通り，元の行列との誤差が事実上計算できない場合がある．従って，混合精度演算を用いた場合，その影響を事前に知ることは容易ではない．ライブラリの使用者は解析全体の精度が十分か留意する必要がある．

■HW 自動選択・ヘテロ　H 行列ベクトル積はメモリ律速なアプリであり，GPU が持つ高いメモリバンド幅性能を活用することで性能改善を図ることができる．

2.4.3　数値計算ミドルウェア

2.4.3.1　インタフェース

スーパーコンピュータ富岳の開発前の数値計算ミドルウェアのインタフェースのポイントは，コード開発の生産性を高めつつアプリケーションのプロトタイプ開発の効率化や開発後の保守にも役立つコード開発環境を構築することであった．従来は，汎用性を考慮した提供機能の抽象化レベルと実行性能のトレードオフを調整し，また抽象化と生産性のバランスをとり「あるアプリケーション範囲」をカバーする粒度で設計されたものが多い．この場合，アプリケーションのカテゴリやデータ構造と解法を絞り，フレームワークの設計を行うことが肝要である．アプリケーションのライフサイクルは 10 年を超えるものもあり，計算機ハードウェアのサイクルよりも長いため，新規のアプリ開発と既存アプリのポーティングの 2 つを考慮する必要がある．上記の考え方は，今後も大きく変わらない．最後のライフサイクルの議論については，ハードウェア・アーキテクチャの変化が短期・長期ともに予想できない面があるため，最大限に考慮すべき内容である．

本分野の動向は以下である．DOE による FASTMath softwaretool には，数多くのミドルウェアが収納されている．また，離散化ツールとして，（非）幾何学的格子生成ツール，AMR，メッシュ分割，UQ ツール，数値最適化ツール，統計処理ツールなどが開発されている．

高生産性のためのミドルウェアを以下にまとめる．

- METIS などのオーダリングツール：分散，GPU，階層的実装が必要.
- 格子生成ツール：AMR，動的な Space Filling Curve，直交格子，一般曲線座標.
- DSL: FDPS, OpenFOAM, FreeFEM++ などは今後の HPC プラットフォームに対応することが必要.
- Matlab などの統合環境: Matlab などを標準的なプラットフォームとする分野もある．HPC プラットフォーム対応，アクセラレータへの自動オフロード，分散資源＋スケジューラ連携などが課題となる．
- データフォーマット：反復解法との関連あり，データ形状と性能との関連，自動チューニング，データ移動やアーカイビングと関連してデータ圧縮などを含む.

2.4.3.2 I/O

2.4.3.3 電力制御

現在のスーパーコンピュータの構成は，高コア数 CPU のみから成る大ノード並列システムや 1 ノードに複数 GPU を搭載することでノード性能を向上させ，より少ないノード数での高い性能を目指した構成が多い．そこに，機械学習などの AI 開発向けに専用アクセラレータが出てきており，システム構成は更にヘテロ化が進んでいる．このように近年のスパコンの性能向上は並列性（SIMD，コア数，ノード数）やアクセラレータに依るところが大きく，その結果，消費電力が多くなってきている．プロセスの微細化が限界を迎えつつあることを考えると，今はまだ消費電力はシステムの設計において重要な役割を果たしているだけだが，今後は，ジョブ実行時におけるリソース要求にノード数，メモリ量や計算時間に加えて消費電力値が入ってくることが考えられる．

CPU 自体やアクセラレータ自体の省電力設計は進んでいるが，現在のようなヘテロな構成では，特定のアプリケーションを実行する際，あまり使用しない機器に無駄な電力を使っていることが考えられる．例えば，データ転送が主な処理の場合は CPU やアクセラレータはほとんど使われない．逆に，計算が主でありデータ転送がほとんど無い処理では，メモリは電力をそれほど使わない．このような状況が想定されるために，消費電力のコントロール機構が必要である．2011 年に発表された Sandy Bridge 以降の Xeon は RAPL（Running average power limit）と呼ばれる電力制御インターフェースが利用でき，CPU や DRAM の電力制限が可能となっている．また，富岳の CPU である A64FX にも電力制限機能が搭載されており，CPU と HBM2 メモリの消費電力制限が可能である．これらは，ユーザ側で電力コントロールが可能となっているが，ユーザにはほとんど利用されておらず，システムとして消費電力値取得や縮退運用などで利用されている．消費電力を制限する場合，基本的には計算性能は変わらない（無駄な電力を使っていた場合）か，性能は下がる（無駄な電力を使っていなかった場合）．そのため，ユーザとしては率先的に制限を課す理由がなかった．

そのような中，電力制御機構を利用した研究も行われており，HPC 系ベンチマークの種類によっては何を制限しても性能に影響が無い，消費電力の削減が期待できるなど知見が蓄積されている．スーパーコンピュータ富岳でも，複数の省電力モードを搭載している．具体的には，クロック周波数の変更，メモリアクセス頻度の制限，命令発行幅の制限，整数演算パイプラインの制限，浮動小数点演算パイプラインの制限などである．また，想定最大電力に応じて電力安定動作機構の電力を調整することで電力を削減するための「エコモード」や，動作しないノードやコアの（スタンバイ）電力を削減するリテンションモードの仕組みも導入されている．

2.4.3.4 可視化技術

シミュレーションのデータを理解するためには，可視化やデータ分析作業がシミュレーション自体と同様に重要な処理である．これまで，シミュレーションの可視化は，計算結果をファイルに書き出し，そのデータファイルに対して処理をする，いわゆるポスト処理方式が主流であった．このポスト処理は低速なファイル I/O 処理を含むため，CPU の演算処理が高速になったとしても，スループットを向上できない問題点がある．加えて，ポスト処理は計算規模が大きくなるほど，問題点が深刻になる．この問題点を緩和するため，シミュレーション結果データをファイルに書き出さず，計算と同時に可視化処理やデータ処理を行う In Situ 技術の開発が進んでいる．In Situ 可視化の場合には，シミュレーション結果から直接可視化画像を並列処理により生成し，元のデータサイズに比べて遥かに小さいラスターイメージやベクトルグラフィクスプリミティブなどの可視化データを出力する．これらの可視化データは，ファイル出力されるか，遠隔地にあるユーザ PC へストリーミング転送され，ビューアによって可視化することができる．In Situ 処理のカテゴリの中には，デー

タを別プロセスへ転送し，より柔軟な処理ができる In Transit 処理も研究されている．

In Situ 処理では，生データから可視化データに変換処理をする際にシミュレータが利用するメモリに加えて作業用のメモリ空間を必要とするため，計算要素数に対する厳しい制約条件となる．このメモリサイズの観点からは，低速でも大容量のメモリ空間を扱える可能性のある NV-RAM への期待は大きい．

ソフトウェア的な観点からは，シミュレータの並列処理と可視化の並列処理は必ずしも同じ並列数で計算するわけではない．この場合，領域分割型のデータは，M プロセスのシミュレーションと N プロセスの可視化処理の間でデータをマッピングする技術が必要になり，アプリケーションレベルのデータを抽象化して，ミドルウェアで吸収すべき処理となるだろう．また，シミュレータと可視化アプリ間の連携度の点では，Loosely-coupled, Tightly-coupled の 2 方式が提案されており，それぞれ特徴がある．また，ソフトウェア構築の観点からは，C/C++ のモノリシックな構築方法から，python などのスクリプトを利用した動的な振る舞いを可能にする構築方法が試行されている．

イメージ生成方法については，専用ハードウェア（GPU）を利用する方法やマルチコア・メニーコアで効率的なレイトレーシング処理など多様な選択肢がある．並列レンダリングについては，ソートラスト型のイメージ重畳方式が最もスケーラブルな処理であり，インターコネクトの特性に応じた重畳アルゴリズムが研究されている．

近年は，イメージ生成の高速処理のため，画像化の処理を機械学習により高速化する方法や，視覚的な精度を吟味し，混合精度演算による精度劣化を許容したイメージ生成方法などの研究も進むであろう．

2.4.4 数値計算・アプリケーションを支える重要技術

2.4.4.1 自動チューニング

■背景 汎用的に行われる自動チューニング (AT) 技術について，数値計算ライブラリに必要な技術項目についてこれまで多くの検討が行われてきた．特に従来からのエクサフロップス環境のための自動チューニング技術については，以下の事項の達成を目指すものである．

- エクサフロップス環境で想定される多様な計算機環境に自ら柔軟に適応できる，ソフトウェア性能を自動最適化できる枠組み（ソフトウェア構築法）を提供する．
- 少なくともコンパイラが行うことができない最適化（コード最適化，自動並列化，など）を提供する．

以上のことで，より高い実行性能を実現するソフトウェアを，低い開発工数（低開発コスト）で行える仕組みを提供するのが AT 技術である．

研究開発の背景としては以下がある．

- 非均質プロセッサ（マルチコア CPU，ベクトル機構を持つ CPU， 演算アクセラレータ (GPU など), の混合アーキテクチャ）が普及した
- 実行時に判明するなんらかの電力制約がある
- 通信性能の劇的な向上が見込めない（通信バンド幅性能の向上に対して，相対的に通信が高レイテンシになった）
- 大規模問題実行時に演算精度が劣化する事例がある
- 高性能を達成するためには，最強となる唯一のアルゴリズムは存在しない．実行時に定まる情報をもとに，アルゴリズムを選択することが必須である

- 人工知能（AI) 処理を実行する場合，実行時にパラメタ（ハイパーパラメタ）の調整が必要
- 混合精度演算による高速化の機会が増大する（「半精度演算」の活用機会が増える）

■AT 適用のタイミングと AT シナリオ　ソフトウェアに AT を適用する場合，どのタイミングで AT を行うかで，AT 対象となる処理が規定されるほか，AT 適用の難易度も変化する．特に数値計算ライブラリに AT を適用する場合，以下の 3 つのタイミングで AT 適用できることが知られている： インストール時 AT，実行起動前時 AT，および，実行時 AT．ここで実行起動前時 AT とは，ユーザがライブラリをコールする直前に行う AT のタイミングである．実行起動前時 AT はコード最適化に必要なユーザ知識を与えることが可能な AT タイミングであるため，得られる AT の効果が大きい．現時点で開発された代表的な AT 機能付き数値計算ライブラリと AT タイミングは以下になる．

- インストール時 AT：ATLAS
- インストール時＋実行起動前時：FFTW, Spiral, ppOpen-HPC
- 実行起動前時＋実行時：Xabclib

一方，どのように AT を適用するのかを定めることも AT 機能を実装する際に重要となる．これは AT シナリオと呼ばれ，扱うプログラムやアルゴリズムの特性により定まる．AT シナリオを定めた後，AT を実装するためのライブラリやツールは多数開発されている．たとえば，特定の演算に特化したライブラリインタフェースを提供するライブラリで，疎行列反復解法のための AT インタフェース OpenATLib が知られている．また，より汎用的には，数値計算処理を指向しつつも，汎用的な計算機言語の一部としてコード変換の方法（「レシピ」と呼ばれる）の登録を仮定するスクリプトである，CHiLL や POET が知られている．

■現在の達成技術　数値計算ライブラリに適用されている AT 機能としては以下が達成されている．

- 【計算機ハードウェアの階層化を考慮した最適化】たとえば, マルチコア対応 (階層メモリ最適化), GPU 対応，周波数が異なる CPU などのヘテロアーキテクチャに対応した AT 方式 [BONSAI][ppOpen-AT の階層化 AT 機能].
- 【電力最適化機能（一部分)】たとえば，ユーザがプログラムの明示的な場所を指定するなどの条件付き，もしくは，実行後に電力量が判明する環境において，プログラム全体の電力最適化を行う AT 方式 [ppOpen-AT による電力最適化].
- 【高度化された AT のための性能モデル】たとえば，複数のパラメタの同時最適化の AT 方式に適用可能な性能モデル [d-Spline モデル]，ベイズ推定など統計処理を利用した性能モデル [ATMathCoreLib]，および，サロゲーションモデル.
- 【より高度な AT 機能を搭載した計算機言語・開発環境】たとえば，コード変換機能として，ループアンローリングやループスプリットなどのループ変換機能を搭載した AT 専用の計算機言語 [ppOpen-AT]. および，GPU やその他最新の計算機最適化を指定するディレクティブに柔軟に適用できる AT 機能を搭載可能な開発環境 [Xevolver].
- 【AI (Deep Learning, DL) を活用した AT 方式の開発】数値計算処理の性能チューニングへの AI 技術を適用した AT 方式を適用する仕組みの研究開発．たとえば，疎行列-ベクトル積（SpMV）に対して，適する疎行列データフォーマットの選択や実装方式の選択を DL を用いて行う AT 方式の開発 [疎行列形状画像を用いた前処理選択方式]．または，疎行列反復解法において，疎行列構造が定まった時点で DL による AT を用いて適する前処理方式を自動選択する AT 方式．また，処理する疎行列が特定の数

値反復解法で収束するのか，しないのかの収束性を予測する AT 方式.

- 【DL における混合精度演算の自動化】DL を実行する際，半精度演算化をする際の精度劣化を予想し，適用可能と予測する際に半精度演算を行う Automatic Mixed Precision (AMP).

2.4.4.2 浮動小数点演算を支える技術

有限桁の浮動小数点演算による数値計算には様々な誤差が発生し，その蓄積によって計算精度の低下，場合により全く無意味な結果となるケース，計算結果の再現性が失われるといった，計算結果の信頼性を損なう問題が生じる．問題の大規模化・複雑化によりその影響は無視できなくなるだけでなく，低精度演算器の活用においては誤差レベルが大きくなることから，次世代高速計算機において浮動小数点演算の諸問題に対する，アーキテクチャ，ソフトウェア，アプリケーションの各レベルでの対応が必要不可欠である．このためには以下のようなアプローチが考えられる．

1. 精度保証付き数値計算：誤差上限を事後的に計算する手法
2. 精度検証法：丸め誤差を確率的に見積もる手法
3. 再現性のある計算法：環境非依存で同一入力同一計算法に対して実行毎に同じ結果を得る手法
4. 高精度演算法：低精度演算を用いて高精度（high-precision and/or accurate）を実現する手法
5. 混合精度計算・精度自動最適化：複数の演算精度を活用する計算手法，演算・データ精度の自動最適化

■精度保証付き数値計算　精度保証付き数値計算に関するアルゴリズムは多く提案提案されており，線形数値計算に対しては幅広い問題に対して提案がなされている．また，100 万次元を超える密行列に対する連立 1 次方程式と標準固有値分解に対して精度保証が可能であることを京コンピュータを用いて示されている．さらに，固有値分解に対しては大規模電子状態計算に応用された結果が発表されており，今後, 様々なアプリケーションへの応用が期待されている．微分方程式系の数値計算に対する精度保証法の研究も進んでおり，様々な方程式に対する解の存在や一意の保証法や数値解の区間包含法が提案されている．

■精度検証法　数値計算の精度検証法とは，丸め誤差の影響を確率的に評価する方法である．精度保証法は様々な誤差に対応できるが，検証法は基本的に丸め誤差のみである．しかし，検証法は精度保証が理論的に難しい場合でも対応できる点や，実装・導入・実行時間コストが小さい等の優れた点があり，精度保証法とは別に実用化を検討すべきである．以下に，近年の動向をまとめる．

- 基本的なアイディアは丸めモードを確率的に変更して演算を行う確率的丸め（ランダム丸め）に基づく方法である．複数回実行して得られた複数の結果のうち一致する桁を正しいと見なす．代表的なものに Discrete Stochastic Arithmetic（DSA），Monte Carlo Arithmetic（MCA）がある．
- 確率的丸めはコンパイラベースで実装でき，ユーザが既存のコードを使って簡単に計算精度の見積もりを行える．この点が精度保証と比べた最大の利点である．しかし精度保証と異なり解の存在を数学的に 100% 保証するものではない．ただし計算精度の把握，コードの精度最適化などの用途には十分であると考えられる．
- 確率的丸めはソフトウェア実装できるが，SIMD や GPU 等のベクトル型命令で高速実行が難しい．これを解消する研究もいくつか行われている．一方，FPGA を用いれば確率的丸めを効率的に実行できるというアイディアもある．
- 確率的丸めは一般的な最近接偶数丸めよりも計算精度が高くなるという研究がある．Graphcore 社の

深層学習向けプロセッサには確率的丸めがハードウェア実装されている．

■再現性のある計算法　浮動小数点演算の丸め誤差の入りかたは計算順序に依存する．並列計算などにおいて計算順序が保証されない環境では同一入力同一アルゴリズムであっても計算毎に結果が異なる場合がある．科学的な議論の妨げになるほか，ソフトウェア開発においてもバグの特定を困難にする．計算順序を変更しうる要因の多い近年の高性能計算機アーキテクチャでは何らかの対策が必要である．以下，技術的な要点と近年の動向を記す．

- 再現性というと一般にビットレベルの再現性を指すが，不正確な結果を再現しても無意味であるので，一般の浮動小数点演算以上の精度で再現性が確保できることが前提条件である．
- 少なくとも BLAS 等の基本的な数値計算ライブラリや演算を伴う MPI 集団通信においてはビットレベルの再現性をサポートするモードが欲しい．
- コンパイラの FMA 抑制オプションは GPU/CPU の結果の一致を見るのに有効であったが，プロセッサがアトミック命令を持つ場合に，コンパイラオプションで実行順序が固定できるような機能が欲しい．
- すでに MKL や cuBLAS 等のベンダ向けライブラリでは，ある制約下で計算順序を固定し再現性を保証するモードがサポートされている．
- 産業利用などにおいては品質保証の観点からビットレベルの再現性が重要であるという話が多い．上記ベンダライブラリにおける再現性のサポートはそれを裏付けている可能性がある．
- BLAS において特殊な方法で再現性を保証する方法がいくつか提案されている（ReproBLAS, ExBLAS, OzBLAS など）．これらは単に再現性があるだけでなく高精度な結果を返すことができる．しかしいずれも総和・内積ベースの演算に対する手法のみの提案であるとともに，level-3 ルーチンの実行時間オーバーヘッドが非常に高く（通常の実装の数十倍以上），実装も非常に複雑である．BLAS でさえこのような状況であるから，ソルバが入る LAPACK となると，ビットレベル再現性のサポートは容易ではないことが想像できる．
- ビットレベル再現性よりも精度保証や精度検証によって丸め誤差量が客観的に把握できれば十分ではないかという意見もある（参考：Weak Numerical Reproducibility [Imamura et al. CRE20]）．
- ARM において High-Precision Anchored（HPA）というハードウェア的な再現性（と高精度演算）の実現法が提案されている．SIMD アーキテクチャ上に簡単な拡張で実現でき，SVE における実装例（シミュレーション）が示されている．

■高精度演算法　精度（accuracy, precision）を高くする計算法．結果の accuracy と precision の両方を高めるいわゆる多倍長精度演算法の他に，accuracy のみを高める K 倍精度アルゴリズム DotK/SumK などの手法が知られている．また，連立 1 次方程式や固有値分解，特異値分解に対しては残差反復法が，行列乗算に対しては尾崎スキームが知られている．高精度計算法は低精度演算器を用いてより高精度な浮動小数点演算をエミュレーションする方法にも応用ができる．

■混合精度計算・精度自動最適化　混合精度計算には反復改良などのアルゴリズム的な改良法と，既存アルゴリズムの実装において精度を最適化する精度最適化の 2 種類のアプローチがある．ECP 混合精度チームによる線形計算の混合精度演算（アルゴリズム的な改良含む）に関するサーベイ論文 FixMe(citation) が参考になる．

2.4.4.3 耐故障

ここではアプリケーション視点での耐故障について，次世代計算基盤となるシステムにおいてその必要性や技術要件を述べる．日本のフラグシップスーパーコンピュータシステムでの耐故障対策は，システムの安定性がアプリケーション実行に支障になるほど悪くなかったため，計画的に研究開発されシステム運用されることがなかったと言える．ただ，一部のユーザにおいては，アプリケーション実行時におけるジョブ実行時間制限から，アプリケーションの実行時間を稼ぐ手段として利用されている．しかしながら，今後のシステムの巨大化とプログラム実行の長時間化と共にシステムの信頼性は低下していくと想定されるため，計画的なシステムへの組み込みが必要になっていくと考えられる．アプリケーション視点での耐故障技術の概要について以降で述べる．

■アプリケーション・数値演算ライブラリにおける耐故障について　アプリケーションの耐故障実現はその実行形式により大きく実現手法が異なる．アプリケーションの実行形式として，1) 入力データが固定で再実行可能な再実行可能型，2) 実時間で入力されるリアルタイム処理型のアプリケーションに分類される．また，耐故障実現の方式としては，次の 3 つに分類される．

- 計算中断再実行方式：アプリケーション実行中断を伴い，アプリケーションを再実行する．
- 冗長リソース代替計算継続方式：予め代替計算リソースを確保し冗長実行，もしくは，故障時に予備計算リソースに処理を代行させアプリケーション実行を継続する．
- 縮退リソース計算継続方式：代替リソースを確保せず故障部を除去してアプリケーション実行を継続する．

耐故障実現のために，計算資源，ネットワーク，ストレージなどの最低限のアプリケーション実行に必要な計算機リソースが必要であるため，最低限の耐故障を確保するハードウェアとシステムソフトウェアの整備が必要である．最低限の計算機リソースが確保ができアプリケーションが実行可能かの機能はシステムソフトウェアレベルで提供されるべきである．

■耐故障機構の必然性　アプリケーション実行において新たな耐故障機構を必要とするかは通常の実行において十分な信頼性が確保されているかに大きく依存する．一般的には，システムの MTBF 以上の時間がアプリケーション実行に必要な場合は，再実行含めて何らかの耐故障機構を用いる必要がある．しかし，アプリケーション実行に必要とされる MTBF は並列実行の場合，利用する計算リソース規模と実行時間により変化するので注意が必要である．計算リソースを 2 倍にしてアプリケーションの実行時間が半分になるのであればアプリケーション実行への影響は同じだが，一般には計算リソース規模が増える毎にアプリケーションが安全に実行可能な時間は減少することになる．

以前より，アプリケーションへの耐故障機構の実装と試行を積極的に続けているのは米国である．これは，システム自体の信頼性が日本に比べて低いうえ，24 時間以上という長時間実行も多いからと考えられる．米国に比べて日本は，アプリケーションを含めた耐故障機構を Flagship システムの運用に組み込んで来ていない．アプリケーションの実行効率向上とシステム自体の信頼性向上の追求により，システムの MTBF がある程度許容範囲であると同時に，MTBF が問題になる大規模でかつ長時間実行が問題になるほど多くないことが関連しているものと考えられる．

■耐故障手法　ここでは，実際にアプリケーションが取り得る耐故障手法について整理する．既存の耐故障手法としては先に述べた 計算中断再実行方式，冗長リソース代替計算継続方式，縮退リソース計算継続方式であるが，どの方式においても故障時に故障ノードの持つ情報を何らかの形で引き継ぐ必要がある．このために耐故障機構は，定期的に一貫性が確保された情報を他のノードがアクセスできる形で格納している．この手法として良く用いられるのがチェックポイント・リスタート方式である．

チェックポイント・リスタート方式は，ノード故障が発生した際には，代替ノード，もしくは，残ったノード群で再分配することにより再実行 (リスタート) する．チェックポイント・リスタート方式には，システムレベルとアプリケーションレベルがあるが，システムレベルは OS を含むシステムレベルのサポートが必要なうえ，格納するデータ量が巨大になるため，アプリケーションが再実行に必要な情報のみ格納するアプリケーションレベルのチェックポイント・リスタート方式が多く用いられている．

チェックポイント・リスタート方式には，様々な高速化手法が研究され利用されているが，並列計算機においては，複数ノード間の通信で実際にチェックポイントデータが変更されるため，複数ノード間での通信と連携してデータの一貫性のあるチェックポイントを実行する必要がある．

耐故障を実現するライブラリとフレームワークとしていくつかの既存研究がある．その概要を次に示す．

- ULFM (User Level Fault Mitigation, https://fault-tolerance.org/):
  FT-MPI(Fault Tolerant MPI) などの耐故障通信ライブラリの研究が発展して標準の MPI 通信ライブラリ規格化を進めており，一部の機能は MPI 規格として規定されている．並列計算機上の通信ライブラリとして，上述した冗長リソース代替計算継続方式と縮退リソース計算継続方式の 2 つの耐故障機構を実現可能な MPI 上での通信ライブラリ仕様であり実装である．
- Fenix (https://github.com/epizon-project/Fenix):
  ULFM を用いて MPI プログラムの耐故障処理をサポートするアプリケーションフレームワークである．既存の MPI プログラムに幾つかの処理を加えることで耐故障性を実現する．機能としてはプロセス故障により通信コミュニケータの耐故障処理とデータリカバリ処理を提供する．プロセス故障への対応は故障時に新たなプロセスを割り当てて冗長リソース代替計算継続方式と残りのプロセス群で処理を継続する縮退リソース計算継続方式をユーザ選択で実現可能できる．また，データのリカバリにおいても Non-local Data ベースと，Local Data ベースのデータリカバリが選択可能である．
- Resilient Kokkos(https://github.com/kokkos/kokkos)：
  kokkos は C++ ベースで性能可搬性を実現するプログラミングモデルであるが，Kokkos のモデルにチェックポイント・リスタート機能の実装が ECP プロジェクトの一つとして進行中である．
- CRAFT(Checkpoint-Restart and Automatic Fault Tolerance) ライブラリ〔Shahzad et al. 2018〕:
  FAU Erlangen/Nuremberg で開発された数値ライブラリで，ULFM-MPI（User Level Failure Mitigation）を使用することによって，MPI プログラム実行中にノード故障が発生してもアプリケーションの実行を継続可能であり，スペアノードの有無の両者に対応している．CRAFT は並列固有値ソルバー〔Shahzad et al. 2018〕，並列有限要素法〔Fukasawa et al. 2018〕など様々なアプリケーションに適用されている．大規模並列環境では，ULFM-MPI のような耐故障性を有する通信ライブラリの存在が不可欠であり，システムソフトウェアグループとの密接な協力が必要である．
- 故障フリー計算手法他 (approximate computing):
  一つのアプローチとして，計算手法自体を耐故障化容易な手法を用いるものがある．例えば SPMD に比べ Master-Worker 方式や MapReduce 方式などは故障発生時の耐故障実装が容易である．また，厳

密解を必要とせずある程度の計算誤差を許す問題の場合は故障発生時に故障プロセスを切り離すだけで処理を続けても精度に影響するが許容範囲な場合があり得る．あるいは計算原理として積極的な近似計算を実行する場合はプロセス故障が発生しても，そのプロセス故障を近似誤差として取り込める場合があり得る．

以上述べた耐故障を実現するライブラリとフレームワークが実際に次世代システムでのターゲットアプリケーションに適応可能かどうかは，アプリケーション修正，性能，生産性，システムリソース，実現性と言った様々な側面から考える必要がある．既に相応の実稼働コードがある場合は，プログラミング言語の変更と言った大きな修正は現実的でない場合が多いため，既存コードに最低限のコード追加・修正で実現可能な方式が望まれる．

2.4.4.4 機械学習フレームワーク

■アプリケーションとの関係 機械学習・DL を用いるアプリケーションとしては，画像を用いるものとして，画像認識 (AlexNet (2012), ResNet (2015), …), 画像検出 (Faster-RCNN, …), セグメンテーションが，音声などの時系列データを用いるものとして，時系列認識 (RNN), 音声認識 (DeepSpeech,…) が，その他，レコメンデーション (Collaborative Filtering, DLRM, …) や深層強化学習 (DQN, A3C,…) が挙げられる．

主に画像認識や時系列認識で用いられる NN には Convolution (GEMM, FFT), Fully-connected(全結合または内積) (GEMM, SLIDE), Pooling, BatchNormalization, LSTM といった処理がある．Convolution のアルゴリズムとしては直接計算する方法に加え，GEMM や FFT などがある．また Fully-connected では通常 GEMM で計算される．これらのアルゴリズムは cuDNN などの計算ライブラリでチューニングされたコードが呼び出されるようになっている．Convolution は計算インテンシブな処理であり，Fully-connected はメモリインテンシブな処理である．CPU 向けに Fully-connected の計算で GEMM の代わりにハッシュテーブルを引くことで発火するニューロンのみに対して処理を行う SLIDE アルゴリズムも提案されている．

その他，マルチタスク学習，転移学習，シミュレーションとの融合させた学習などが進んでいる．マルチタスク学習は複数のタスクを組み合わせた学習である．複数の画像認識のタスクを組み合わせて汎用的な画像認識を行うものや，画像と音声など異なる種類のデータを用いるものがある．転移学習は学習済みのモデルを用いて別のタスクを学習ものである．シミュレーションとの融合としては，大きく分けて，(1)AI の計算を HPC で高速化するものと，(2)HPC の計算を AI で補うものがある．(1) では，HPC の数値計算を AI の代理モデルで軽量化して高速化する．(2) では，AI によって数値計算のモデルを構築し，解析の幅を広げる (第一原理のモデルが存在しないところを補完，または高精度化する)．MLPerf HPC ベンチマークでは宇宙物理学のダークマターモデルのパラメータ推定を行うもの (CosmoFlow) と，異常気象が起きている地域をセグメンテーションするもの (DeepCAM) がある．これらはいずれも (2) に分類される．

機械学習アプリケーションを用いる産業の例としては自動車の技術革新が挙げられる．CASE (Connected, Autonomous/Automated, Shared, Electric) への技術課題の一つとして AI 技術への計算性能と電力効率の向上が挙げられており，将来のアルゴリズムにも対応できる動的再構成プロセッサ (DRP) による AI 専用プロセッサの開発が進められている．

■シミュレーションと機械学習を融合させたアプリケーション (AI for HPC) の分類 シミュレーションと機械学習を融合させたアプリケーションについて，シミュレーション・学習・推論それぞれの計算量の比率によって分類する．

シミュレーションが重いケース　予め大量のシミュレーションを実行しておき，生成されたデータを用いて学習・推論を行うもの

シミュレーション時間 > 学習時間 ≫ 推論時間

例

- 代理モデル (シミュレーションを AI の推論に置き換え)　シミュレーションの入出力の関係 (回帰) を学習して推論 (エミュレーション)　構造，流体，電磁気シミュレーションなど
- パラメータ推定　CosmoFlow：COmoving Lagrangian Acceleration (COLA) 法を用いた大規模な N 体問題シミュレーションで得られる物質の配置を用いて，逆問題 (逆関数) を学習
- ソルバ，前処理，収束条件などを推定 (東大地震研，片桐先生論文など)　様々な行列に対して，各種ソルバや前処理ごとにシミュレーションを実行し，その際に得られる行列の特性や収束性を学習することで，別の行列に対して高速なソルバや前処理を推論する
- シミュレーション結果に対して後処理 (HPC の計算結果を AI で高精度化)
- AI がシミュレーション結果を説明

シミュレーション時間と学習時間の比率は論文に記述されていないことが多く不明だが，シミュレーション時間は典型的には 1 つあたり 10〜60 分程度かかり，学習データを作成するために，数十〜数百のシミュレーションを実行すると考えられ，並列化の度合いによるが数時間〜数十時間程度かかると考えられる．学習はシミュレーションで得られたデータをニューラルネットワーク等で反復して読み込んで計算するのに並列化して実行することで数十分〜数時間程度で完了すると考えられる．

学習が重いケース　観測データを用いて学習するもの

学習時間 ≫ 推論時間 ≫ シミュレーション時間 = 0(観測データを使用)

例

- 検出 (セグメンテーション)　DeepCAM：過去の気象観測データの異常気象のセグメンテーションを学習し，未来の観測データから異常気象を推論する
- 観測データの補完

観測データを用いて学習する場合はシミュレーションの時間はゼロとなる．または，観測データとシミュレーション結果を組み合わせて学習を行うことも考えられ，その場合はシミュレーションの比率が学習と同等になることも考えられる．

シミュレーションと学習が共に重いケース　シミュレータと AI がインタラクティブに計算するもの

学習時間, シミュレーション時間 ≫ 推論時間

例

- 強化学習　BMW グループが開発している物流ロボット (NVIDIA が開発しているロボット向け 3D 仮想空間シミュレータ Isaac Gym を使用)

シミュレーションの結果を用いて AI で推論を行い，推論の結果をシミュレータにフィードバックすることを繰り返しながら学習を行う場合，シミュレーションと学習が共に重くなると考えられる．シミュレータの処理の重さによって比率は異なる．上記の物流ロボットの例の場合はおそらくシミュレータの処理は軽く，学習がボトルネックになると想像される．

推論が重いケース　推論時間 >> 学習時間, シミュレーション時間

例

- 翻訳サービス

推論は通常数ミリ秒〜1 秒程度で完了すると考えられるが，一旦学習が完了したモデルを用いてクラウド環境で大量の推論リクエストを即時処理するような場合は推論が重くなるケースも考えられる．

2.4.4.5 アルゴリズム

■グラフ処理 人やもの，データであるなどの関係性を，頂点と辺を用いて表したグラフに対して，単純な関係性だけでなく背景にあるものを分析するためのグラフ処理に関する研究開発が盛んに行われている．グラフ構造を持つものとしては，ソーシャルネットワークサービスにおける人と人の関係性や，ハイパーリンクに基づく webpage の関係性 [206]，タンパク質などの分子構造など多岐にわたる分野で例が挙げられる．また，近年機械学習への注目が非常に高まっているが，その一つであるニューラルネットワークもグラフ構造を持ったもののひとつであると言える．

各分野にて扱われるグラフのサイズは，年々より大きなものとなってきている．現在でも数兆の辺を持つグラフが登場しており，グラフのサイズが大きい場合，必要となる計算時間を短縮するためには大規模な計算機群を効果的に用いることが重要となり，大規模なグラフを処理するための基盤技術が必要になる．また，計算科学チームだけでなく，より幅広いユーザの利用を考えた際，大規模グラフ解析を行うための基盤技術を踏まえたフレームワークやライブラリ開発の必要性も高いと言える．

大規模なグラフ処理に関する性能を表す指標として，Graph500[207] や GraphChallenge[208] などが提案されており，このようなベンチマークでは頻繁に扱われる処理であるグラフの探索（幅優先探索等），最短経路，関連性検索（triangle counting）などをピックアップし，重点的な性能最適化を促している．また，上記の処理に限らず，最小費用フローなどの最適化問題やクラスタリング，Pagerank なども広くベンチマークとしても用いられるような処理である．このようなベンチマークにおいて，現在では 1 兆頂点を持つようなグラフを扱う場合もあり，グラフデータをどのような形で保管するかも重要となる．

巨大なグラフを大規模並列に行う際，効果的に処理するための研究が様々な面で行われており，計算機に合わせた技術が必要となる．以下，概要をまとめる．

- データの格納形式と配置:
  データ格納形式としては，リスト形式（各頂点について，どの頂点と辺を持っているかを管理）の他，隣接行列として保管するなど様々な手法がある．グラフは疎な特性を持つことが多く，隣接行列として表す場合には疎行列の格納形式が用いられる．処理の高速化においては局所性を考慮した配置を行う必要があり，特にデータサイズが巨大である場合には，キャッシュレベルだけでなく下の主記憶メモリ，場合によっては SSD 等の補助記憶装置まで含めた最適化が行われている．
- 通信方法:
  多くの処理においては各頂点毎に処理が進められていき，処理結果を各計算ノード間でやりとりする．やりとりの方法としては push/pull に大別され [209]，対象となるグラフ処理やグラフデータによって，優位となる手法が異なる．
- アクセラレータを含めた多様な環境への対応:
  GPU 等のアクセラレータの高並列性を活用するための研究開発も行われている．一方で，グラフ処理ではデータ通信のメモリバンド幅だけでなく低レイテンシ性が求められるため，GPU クラスタにおい

ても CPU のみを使用する場合があり，課題であると言える．

分散並列環境において，効率的にグラフ処理を行うための基盤技術だけでなく，より使いやすい形で提供することが必要となる．支援するためのフレームワークは Pegasus, Pregel, GraphX, Giraph, GraphLab など多数提案されており，様々な環境への対応の有無や実装などにおいてそれぞれ異なるものとなっている．また，フレームワークレベルではなく，GraphBLAS などのライブラリとしての提供も提案されている．グラフ構造を隣接行列として表現し，行列計算の延長として扱っているため，グラフ処理に限らない行列計算に関する知見を活用することが可能となっている．

■時間並列計算　計算機のコア/CPU 数の処理能力とそれらを接続するインターコネクト性能の乖離が大きな状況では，データの局所性を利用した効率の良い計算が要求される．例えば，領域分割型の並列計算はある程度のスケーラビリティを保つことができるが，ノード数が増えると性能は飽和する．このように空間並列性能が飽和した状況で更に並列性能を向上するためには，新たな並列化の軸が必要となり，時間進行型のシミュレーションにおいては時間方向の並列性を取り入れる研究が進められている．

近年の時間並列計算の基礎となっているのはマルチレベル時間進行法であり，Parareal 法，SDC 法，PFASST 法，MGRIT 法などがある．これらの研究結果をもとに，XBraid ライブラリが開発されている．これまでの研究で，放物型偏微分方程式では高い効果が認められているが，一方で双曲型偏微分方程式では収束性に問題があり，更なる研究が進められている．具体的なアプリケーションとしては，フェーズフィールド法や燃焼シミュレーション（化学反応を伴う流動問題のうち，化学反応部分），地下浸透流計算などの報告例がある．また，反復過程における計算速度を改善する方法として，混合精度計算の取り組みも報告されている．

2.4.4.6　量子コンピューティングの応用

現状では量子コンピューティングの活用に向けた適用事例の検討が進められているところである．下記の分野での活用が期待されている．

- 材料/素材開発 ・創薬/ヘルスケア ・交通/物流 ・金融サービス ・メディアテクノロジ
- サイバーセキュリティ ・AI/機械学習

中でも量子化学シミュレーションに基づいた材料/素材開発は量子コンピュータの活用が最も期待されている分野のひとつであり，産業界を巻き込んだ活用事例の検討が進められている．現状の活用事例としては，電気自動車搭載バッテリの分子シミュレーションや有機材料の光学特性制御が挙げられる．ここでは，材料/素材開発に絞って，量子コンピューティングにおける動向を示す．その他の事例に関しては，例えば参考文献 [1] を参照されたい．

産業界においては材料/素材開発の効率化・加速化が望まれている．また，省エネルギー，蓄エネルギー，創エネルギー，CO2 固定化・有効利用に向けた新材料の創出は世界規模の喫緊の社会的課題である．これらの問題の解決には，合成に先立って材料の性能をシミュレーションで予測する技術の有効活用が有望である．例えば，様々な材料の組合せに対して精度の高い量子化学シミュレーションに基づき網羅的かつ系統的なハイスループット・シミュレーションを実施することでデータベースを構築し，既知の動作原理に基づきスクリーニングすることでデータベースの中から最適な材料候補を提案することが可能である．一方，動作原理が確立されていない場合は，シミュレーションだけでは膨大な材料の組合せの中から所望の性質を与え得る組合せを導き出すことは困難である．かわりに，実験によって合成されて測定された実測データと高精度な量子化学計算によって得られたシミュレーションデータを人工知能に基づいたインフォマティクスを使って分析すれば，

未知の化合物の性質を予測することができる．また，所望の性質に関連が高い物性が何かも見つけることが可能であり，そこから設計指針を確立することができる．最近ではこのような機械学習モデルを使ったインフォマティクス推測により，新材料の分子設計と提案が盛んに行われるようになっている．所望の性質を持つ新材料を設計する際には，莫大な候補空間の中から最適な候補材料を求めるためにベイズ最適化が通常用いられるが，従来のベイズ最適化の計算コストはデータ数の 3 乗に比例して増加するため，学習モデルを構築する場合に重要度が高いと思われる説明変数のみを使用する制限が現状ではある．

2.5 運用に関する技術動向

2.5.1 従来のスパコン利用方式

2.5.1.1 バッチ処理とインタラクティブ処理

スーパーコンピュータの利用方法としては，バッチ処理型とインタラクティブ処理型がある．スーパーコンピュータは一般に非常に高価な計算資源であり，利用可能な計算資源に対してユーザからの要求が超過することを想定し，共用のスーパーコンピュータではバッチ処理を採用することが多い．

■バッチ処理の利点

1. ジョブ実行の時間帯の偏りを平滑化，システム利用の効率化
2. 時間帯をシフトすることにより電気代などランニングコストの削減（夜間電力の使用，など）

ユーザの要求するリソースに応じて，複数のジョブキューが用意され，ユーザは明示的に希望する（あるいは許可された）ジョブキューを指定してジョブをサブミットする．また，同時実行可能ジョブ数に制限を設けるなどして公平性を高めている．

■バッチ処理のスケジューリングアルゴリズム　バッチ処理によりジョブを実行するためには，あらかじめバッチジョブスクリプトを記述する必要があり，それをバッチジョブスケジューラにサブミットすることでジョブを登録する．バッチジョブスケジューラは，先行するジョブが終了しリソースに空きができると，決められたアルゴリズムに従って次のジョブを選んで投入する．アルゴリズムとしては，FIFO (First-In First-Out) や Fairshare が多く用いられている．通常，back-filling を行い，利用率の向上を図るが，この場合は事前にジョブの実行タイミングを知ることが困難になる．

FIFO:　先にサブミットされたジョブから順に実行する．

Fairshare:　fairness に応じて優先順位を決める．スケジューリング候補になるジョブに対して，そのユーザーまたはグループの残りバジェットの割合が多いほど優先順位が高くなる．

Back-filling:　あるジョブが要求したリソースに対してリソース自体が不足している場合，先行ジョブが終了してリソースが空くまで待つ必要があり，利用可能なリソースは全てそのジョブのためにブロックされてしまう．このとき，ブロックされているリソースを使用しても本来のジョブ実行に干渉しない場合に，後続の無関係なジョブと順序を入れ替えて先に実行することを言う．

アプリケーションの中には，1 つのジョブで目的の計算を行うものだけでなく，複数のジョブが連携して計算を進めるものが存在する．複数のジョブを関連づけて実行する方法として以下がある．

アレイジョブまたはバルクジョブ：　パラメータサーチを行うような場合，1 度のジョブサブミッションで同

時に多数のジョブを実行する機能.

チェインジョブまたはジョブチェーン，ステップジョブ： ジョブの依存関係を指定して実行する機能.

■バッチ処理の問題点 バッチ処理の問題点としては，基本的にジョブがいつ実行されるかわからないため，リアルタイムな処理には対応できない点が挙げられる．また，バッチジョブスケジューラによって，オプションの指定方法やコマンド体系が異なっており，システムごとにジョブスクリプトの変更が必要になる．フラッグシップシステムと第 2 階層システムはもちろん，第 2 階層システム同士でも記述方法が異なるため，ソフトウェアを導入するシステムごとに正しく動作するスクリプトの書き方を確認する労力が必要であり，ソフトウェア普及の障害となることもある．このような問題を解決しうるものの例としては高生産並列スクリプト言語 Xcrypt があげられる．各システムのジョブスケジューラに対応した設定ファイルを用意することで各システムの違いを吸収し統一されたジョブ実行環境を実現する．また，一連の複雑なジョブ処理の管理を行うことができる．

ジョブ投入時に指定した実行時間しか保証されないことも問題になる．指定した実行時間を超過してしまうとジョブは打ち切られ，結果を取得できない．それを迂回する技術としてチェックポイントリスタートを，システムレベル，アプリケーションレベルで実現する手段もあるが，多くの利用者は荒い見積もりによる「これ以上はかからないだろう時間」や，スケジューラが許可する最大実行時間を指定している．このような正確性の低い時間を指定してジョブが投入されると，Back-filling が十分に働かず資源利用率の低下につながるだけでなく，利用者のジョブ実行待ち時間も増える．スケジューリングアルゴリズムを改善することにより本問題を解決する研究はいくつもある（5 章で示す）が，実運用システムへの適用事例は，我々の知る限りはない．現在の運用での解決方法としては，利用者を教育し，より正確な実行時間の指定を促す程度である．しかしながら，東工大の TSUBAME3 では，この考えを発展させて，ジョブに対する課金に，正確な実行時間を指定するインセンティブを組み込む運用を行っている．具体的には，正確な実行時間を指定するほど，ジョブ実行に要した費用が割引される方針である．

■インタラクティブ処理 スパコン によってはインタラクティブ実行が可能なシステムもある．多くのジョブスケジューラがインタラクティブ実行機能に対応しているが，運用のポリシーとしてその機能を有効にする必要がある．この場合，リソースが空いていれば，通常の Linux コマンドラインがそのまま使用でき，X Window System などの GUI も利用可能である．リソースが空いていない場合には，利用可能になるまで待つ必要がある．通常はデバッグや小規模な実行の確認，プリポスト処理などに使われ，あくまでもバッチ処理の補助的な役割である．しかし大規模なプリポスト処理や可視化処理などは必要なため無視はできない．インタラクティブ実行の提供形態としては，バッチジョブシステムの機能としてインタラクティブジョブ用のキューが設けられている場合が多いが，バッチジョブシステムとは別に実行開始時刻を指定してジョブの予約を行える仕組みも一部では用いられている．例えば九州大学 情報基盤研究開発センターの ITO と名古屋大学 情報基盤センターの「不老」では富士通社の UNCAI というシステムによって，Web ブラウザ上で実行開始時刻を指定したジョブの実行が可能になっている．外部設備との連携，例えば他機関のスパコンと連携して大規模分散ジョブを実行したり，センサデータを定期的に取得・更新するなどの使い方をする場合には，実行開始時刻を確定させる仕組みは重要になるだろう．また，東工大では利用者ポータル経由でインタラクティブジョブの起動をサポートしており，Jupyter Lab が簡単に利用できる．可視化処理についてはソフトウェアや接続機器の都合で Windows 環境などと連携が必要となることもあり，この場合は計算ノードとは別に Windows 端末を用意することになるが，共有ストレージに直接高速アクセスできる環境または十分な容量を持つ高速な

ローカルストレージが必要となる．

インタラクティブ利用と関連して，インタラクティブジョブ，あるいはバッチジョブで確保したノード間での SSH 利用に対する要望も近年は多く上がっている．従来のスパコン利用では複数ノードを使った計算の大半は MPI による並列化が行われていたが，近年の深層学習フレームワークでは，並列学習のためのリモートプロセス起動に SSH が使われているものもある．そのほかにも，リモートプロセス起動が SSH 固定のオープンソース系のツールもある．リモートプロセス起動を MPI ベース，あるいはジョブスケジューラが提供するリモートプロセス起動手段（qrsh など）ベースに書き換えることにより利用者側での対応は可能であるが，成果創出までのリードタイムが大きくなる．東工大などすでにノード間 SSH を許可しているセンターもあるが，将来，より多くの利用者への利用拡大，オープン系のツールの導入を目指すのであれば，SSH 等，より汎用的なリモートプロセス起動・ログインツールの対応を行った方が良い．

2.5.1.2 課金・資源割り当て方式

■ノード時間に基づくトークン方式 多くの情報基盤センタースパコンや産総研 ABCI では，利用権をトークン（ポイント）の形で事前に購入し，ジョブ実行やストレージ利用で減算するトークン方式を採用している．特に東工大ではノード時間に単純比例ではなく，スケジュール負荷を勘案しジョブの予定実行時間を勘案したポイント消費計算式を採用，ポイントを多く支払うことにより高優先度での実行もサポートしている．複数のスパコンを運用している機関や複数のサブシステムを有するスパコンを運用している機関においては，ポイントを共通して利用するために，ポイント変換の制度を有していたりサブシステムごとに消費するポイントが異なるといった対応をとっている．東工大や名大では多くのポイントを消費することで優先的に実行される制度もとりいれている．

■完全定額方式 九大ではノード時間に関係のない完全定額方式を採用している．この制度は最大で利用できるノード数により利用料金が決まる．ノード時間に基づくトークン方式と比べると，システムが混雑していない場合は最大ノード数までジョブを実行し続けられるというメリットがある一方，ジョブを実行していなくても料金がかかることや一時的にノード数を増やすことができない点がデメリットである．

■ノード固定，ノード年に基づく割り当て方式 いくつかのスパコンでは一定期間ノードを占有できる利用制度を提供している．本方式ではシステムが混雑している場合でも定められたノード数までのジョブを自由に実行可能であるため，常に一定程度のジョブを実行したい利用者にとって需要がある．本方式による利用料金は同じ時間を従量制で利用した場合よりも割高に設定されているはずであるため，システムの混雑度が低く通常のジョブがすぐに実行される状況においては，ユーザにとって本方式を利用するメリットはない．本方式を提供するシステムでは，契約したユーザが実際にはジョブを実行していなかった場合にシステムの稼動率が低下するという問題が生じる．そこで九大と名大ではノード準占有制度を運用しており，固定利用の対象ノードであっても使われていないときには一般ユーザによる短時間ジョブが実行されるようになっている．ただしこの制度はセキュアな環境が欲しくてノード固定を利用するユーザには不向きという問題がある．

■その他 「バッチ処理の問題点」で述べたように，東工大ではジョブ投入時の確度でインセンティブを与えている．

2.5.1.3 ソフトウェア環境の整備とコンテナ環境の活用

スパコンはユーザが管理者権限を持てないためアプリケーションのインストールに制限が大きい．一方で複雑化したソフトウェア，他のソフトウェア群と連携して使うソフトウェア，特定バージョンのソフトウェアを必要とする事例も増えている．運用側でそれらの全てに柔軟に対応するのは容易ではない．これらの問題はコンテナ環境を使うことで解消できる場合があり，活用が広がりつつある．一般的にコンテナ環境としては Docker が多く用いられているが，Docker はコンテナの実行に管理者権限が必要なため，スパコン上で一般利用者に利用させることは難しい．これにたいして Singularity は一般ユーザ権限でコンテナを実行可能であり，また fakeroot 機能を利用すればスパコン上（ログインノード上，計算ノード上）でコンテナ構築も可能なため，スパコン上でコンテナ環境を利用するのに適している．現在，「富岳」，TSUBAME3，ABCI, Reedbush, Oakforest-PACS, Oakbridge-CX,「不老」が公式にコンテナ環境に対応している．

一方，計算ノードから直接インターネットなどの外部ネットワークにアクセスできないスパコンも多く，計算ノード上でコンテナを使う（コンテナ環境を構築する）場合には不便となることがある．これはコンテナを用いる場合以外でも問題になることがあり，特にログインノードと計算ノードのアーキテクチャが異なる環境においては実際に計算を行う計算ノードと同等の環境からインターネットに接続できないことによる不便をより強く感じる原因となり得る．

2.5.1.4 ストレージの運用

ストレージ技術そのものは他章に譲り，ここではストレージの運用の現状について述べる．

■ストレージ構成の現状　ストレージはどのスパコンでも非常に重要な構成要素であり，高速化と大容量化を進めてきた．ストレージの構成としては HDD RAID による共有ストレージシステムを 1 式備えるのが基本である．SSD RAID はより高い性能が期待できるが，コストパフォーマンスの観点から大規模には採用されにくい．

従来よりテープアーカイブを用いた 2 次記憶装置も一部では使われてきたが，HDD ストレージの速度・容量・容量単価の増加などの要因によりスパコンの一部としての導入事例は珍しくなっている．一方，名大では「不老」にて追記式（Write Once）の光ディスクアーカイブによるコールドストレージを導入した．これは長期保存・低消費電力（稼働していないディスクは電力を消費しない）とシステム更新時の利便性向上（利用料を支払ったメディアを利用者に渡すことで，システム運用終了後も活用できる可能性を残す）を狙っている．

一部のスパコンでは高速なデータアクセスのための階層型ストレージも使われている．最も単純な構成としては計算ノードに高速な SSD を搭載するものが考えられるが，BeeGFS のように計算ノードローカルな SSD を共有して使えるようにするものもある．また，IME のように SSD を別筐体に搭載した製品も使われている．限られた容量の SSD にどのようにデータを出し入れするか，何をする際の性能が改善されるのかなどを正しく考えて導入・利用する必要がある．

2.5.1.5 外部システムとの連携

■センター間連携　多くの大学が SINET による高速なネットワークにより接続されているが，スパコンのストレージまで十分高速なネットワークでアクセスできるとは限らない．ネットワーク的に 40Gbps 未満で接続されている可能性や，40Gbps で接続されていたとしても gateway/firewall の性能上限の都合でより低い速度でしかアクセスできない可能性がある．今後，計測データの取り込みなどを考えると障害となる．信頼

された利用者同士であれば L2-VPN などである程度緩和できる可能性があり，一部の共同研究等では実際に L2-VPN アクセスを行っている．

2.5.2 クラウドと HPC

クラウド利用形態の分類は以下の通りである．

IaaS (Infrastructure as a Service): サーバ，ストレージ，ネットワークなどのインフラをインターネット経由でサービスとして提供する利用形態．サーバは多くの場合，仮想マシンとして提供されるが，物理マシンのまま直接提供するものを，特に「ベアメタルクラウド」と呼ぶ．2020 年 9 月現在，Amazon EC2 (Elastic Compute Cloud), Google Compute Engine, Microsoft Azure IaaS, Oracle Cloud Infrastructure/IaaS などが代表的なサービスである．

PaaS (Platform as a Service): ソフトウェアを稼働させるためのプラットフォームをインターネット経由でのサービスとして提供する利用形態．2020 年 9 月現在，Google Apps Engine, Microsoft Azure, Oracle Cloud Infrastructure/PaaS などが代表的なサービスである．

SaaS (Software as a Service: 必要な機能を持つソフトウェアだけをサービスとして提供する利用形態．2020 年 9 月現在，Google の Gmail に代表される G Suite などが代表的なサービスである．

従来のクラウドでは，

- サーバの構成が HPC 向けではない
- サーバ間のネットワーク性能が低い，あるいは性能が安定しない
- ストレージ性能が不足している

などの点から，HPC 分野での実際的な利用は難しいと考えられてきた．しかし近年，HPC クラウドとして，HPC 向けのサーバ構成，またサーバ群を InfiniBand などで密結合し，Lustre のような並列ファイルシステムを提供する HPC クラウドのサービスが増えつつある．加えて，このような HPC クラウドをプラットフォームとして利用した上で，独自に ISV の環境を構築し，SaaS として時間単位でサービスを提供する，Rescale や Xtreme-D などの HPC クラウドベンダーも現れている．

そのためフラッグシップシステムやセンタースパコンと HPC クラウドとの連携についても進められており，例えば「富岳」と Oracle Cloud が SINET により接続され連携することが発表されている [210]．また，九州大学の ITO システムは Rescale と連携しており，ScaleX の提供する Web GUI から，パブリッククラウドに計算ジョブを投入するのと同様の手順で ITO に計算ジョブを投入できる．

多様な使い方への対応，非スパコンユーザのスパコンへの入口として，HPC クラウドとスパコンの連携は今後より重要となっていく可能性がある．

2.5.3 新しい利用形態: Society5.0 を支えるプラットフォーム

Society5.0 時代には，IoT デバイスで全ての人とモノとがつながり，様々なデータや知識が共有され，それによって新たな価値を生み出すことが期待される．未来投資会議を経て 2018 年 6 月に閣議決定された「未来投資戦略 2018」においても，「『Society 5.0』『データ駆動型社会』への変革」と戦略目標が立てられ，Society 5.0 の実現に向けて多くの具体的な施策が掲げられている．これらを実現するには，IoT デバイスや既存の

ビッグデータなど良質なデータを有機的に結びつけて，解析する環境が必要である．

そこで，Society 5.0 時代の根幹をなす，大量のデータを処理するためには，従来のスパコンに匹敵する処理基盤が求められる．例えば理研 R-CCS に設置されているスーパーコンピュータ富岳や東大情報基盤センターで調達中の BDEC システムでは，従来のスパコンをベースにしながら，外部データを取り込みリアルタイムにビッグデータ処理を実現することを念頭においている．

一方，昨今の情勢として，センサ情報などから大量データの取得・利用が可能になってきたこと，大量のデータからの高次的な情報抽出が機械学習などにより容易になってきたこと，さらには，データそのものを共有するという考え方が浸透したことにより，データの価値が高まり，これらのデータ利活用を促進する場が求められている．このような利活用に向けた情報基盤を提供し，全国の大学・研究所が参加するコミュニティ形成に向けたデータ活用社会創成プラットフォーム計画が進められている．その一例として，データ活用社会創成プラットフォーム計画を支える情報基盤の中核となる基盤システム mdx[FixME 情報基盤センター 2018 年度年報] について述べる．

データ活用社会創成プラットフォーム基盤システム mdx は，データ中心科学や，ビッグデータ解析やデータ科学的手法への期待が高い分野に向けて，全国共同利用に提供する情報基盤として導入が予定されているシステムである．2020 年 2 月に東京大学柏 II キャンパスに設置予定である．mdx では以下を実現する．

1. セキュア，大容量ストレージと高性能な計算基盤
2. 広域データ収集ネットワークと連携した情報基盤
3.「プラットフォームのためのプラットフォーム」の提供

従来のスパコンと同様の高性能なハードウェアや，堅固な運用体制に加えて，プロジェクト間の明確な隔離による高いセキュリティレベルでの運用，さらに高い可用性を実現する．SINET が提供するモバイル仮想閉域網により，インターネット上の IoT デバイスから計算基盤に至るまでプロジェクトごとに閉じたセキュアな環境の構築が可能になる．一方で，ユーザには柔軟に計算リソースの割り当てを可能にし，システムソフトウェアの選択も自由に行えるようにする．より大規模な機械学習処理やデータ同化シミュレーションなどのために，他のスパコンのような外部計算資源との連携も実現する．

これらを実現するため，mdx では従来のスパコンで用いられてきた技術に加えて，仮想化，コンテナ技術，SDN (Software Defined Network) や NFV (Network Function Virtualization) 技術，計算資源管理・オーケストレーション技術など，クラウドで用いられてきた技術も取り入れ，これらの高度な要求に応えるプラットフォームの実現を目指している．

2.5.4 設備と運用技術

2.5.4.1 省電力対応と緊急時・災害時の対応

スーパーコンピュータは大電力を使用するシステムであるため省電力技術が重要である．さらに，本体ハードウェアだけではなく施設や運用制度としての省電力対応も必要である．そこで，そのような施設や制度の面での省電力技術や対応についてまとめる．

さらに 2011 年 3 月 11 日に発生した東日本大震災では，東日本を中心に広範囲で電力不足状態となり，大電力を消費するスーパーコンピュータも停止や縮退運転を余儀なくされた．また，地球温暖化や異常気象により夏の暑さや冬の寒さが厳しさを増しており，大学単位や自治体単位での電力逼迫という状況もたびたび生じている．そのため，将来のスーパーコンピュータにおいても，通常運転より使用する電力を削減した状態での運

転が必要となる可能性がある．さらに 2020 年 3 月以降の新型コロナウイルス感染症対策における緊急事態時には，出勤禁止やリモートワーク推奨などの措置がとられ，スーパーコンピュータ運用に関する一部業務が停止・変更される自体も発生した．そこで，ここでは東日本大震災や真夏・真冬の電力逼迫時，新型コロナウィル感染症対策時のスーパーコンピュータの運転状況についてまとめる．

■電力逼迫時の対応事例　東大では柏に大規模システムが移る前，都内の電力消費量が多いお盆前後にシステムを全面的に停止していた．東日本大震災後，電力逼迫時期に SR11000 システムを完全停止．SR16000 を有していた九大の利用を促した．（利用負担金を東大から出した．）

東工大では東日本大震災後，TSUBAME2 において電力逼迫時期に 40% 規模の縮退運転を実施，当該時期の全ノード利用を休日や夜間に誘導するなどを行った．後に電力消費量に基づいたノードの自動的な起動・終了機能を実装，夏季の電力逼迫時には平日昼間のみ電力キャッピング運用を実施した．その際にシステム都合で停止可能なジョブや，1 時間未満の短時間ジョブのみを実行するノードおよびジョブクラスを設定した．

九大ではキャンパス全体の電力需要が高まる夏季の電力使用量を削減する（最大電力量を抑える）ため，ノード固定利用に提供している分のノードを夏季は停止している．

名大では FX100 運用時，キャンパス全体のピーク電力削減のため，夏季に一部ノードを停止して運用していた．「不老」では，キャンパス電力逼迫時の電力削減は昼間の電力消費量が問題であることから，夜間しか実行されなくなる可能性があるノードとそれにあわせた短時間キューを用意しピークシフト運用可能な体制を準備した．ただし，新型コロナウィル感染症対策によりキャンパスで活動する人が大きく減ったためか，電力逼迫状態にならなかったため，ピークシフト運用は実施しなかった．

■新型コロナウィルス感染症対策の運用事例　新型コロナウィルス感染症対策のための緊急事態宣言が発令された際には，多くの情報基盤センター群にて技術職員や SE が出勤停止となった．そのため，アカウント発行などの事務手続きの停止や遅れ，システム障害時の対応の遅れが発生した．また，これを機に事務手続きのデジタル化が進んだ組織もあった．

一方，理研では「新型コロナウイルス対策を目的としたスーパーコンピュータ「富岳」の優先的な試行的利用」，HPCI では「新型コロナウイルス感染症対応 HPCI 臨時公募課題」を募集し，構築中の「富岳」を含むスーパーコンピュータの計算資源を提供した．

参考文献

[1] Mark Bohr. “Continuing Moore’s Law”. In: *Intel Technology and Manufacturing Day* (2017).

[2] G. Tzimpragos et al. “A Computational Temporal Logic for Superconducting Accelerators”. In: *the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS’ 20)*. Mar. 2020, pp. 435–448.

[3] K. Ishida et al. “32 GHz 6.5 mW Gate-Level-Pipelined 4-bit Processor using Superconductor Single-Flux-Quantum Logic”. In: *2020 Symposia on VLSI Technology and Circuit (VLSI symposium)*. June 2020.

[4] A. Holmes et al. “NISQ+: Boosting quantum computing power by approximating quantum error correction”. In: *International Symposium on Computer Architecture (ISCA)*. June 2020, pp. 556–569.

[5] Y. Shen et al. “Deep Learning with Coherent Nanophotonic Circuits”. In: *Nature Photonics* 11 (2017), pp. 441–446.

[6] A. N. Tait et al. “Neuromorphic Photonic Networks using Silicon Photonic Weight Banks”. In: *Scientific Reports* 7.7430 (Aug. 2017).

[7] K. Kitayama et al. “Novel frontier of photonics for data processing - Photonic accelerator”. In: *APL Photonics* 4 (Sept. 2018), pp. 090901-1–090901-24.

[8] W. Liu et al. “HolyLight: A Nanophotonic Accelerator for Deep Learning in Data Centers”. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE2019)*. 2019, pp. 1483–1488.

[9] S. K. Moore. “Huge chip smashes deep learning’s speed barrier”. In: *IEEE Spectrum* 57.1 (2020), pp. 24–27.

[10] SW. Tam et al. “RF-Interconnect for Future Network-On-Chip”. In: *C. Silvano, M. Lajolo, G. Palermo (eds) Low Power Networks-on-Chip*. Springer, 2010.

[11] International Roadmap for Devices and Systems. “Systems and Architectures (2020 Edition)”. In: *IEEE* (2020).

[12] JEDEC. “JEDEC Publishes New DDR5 Standard for Advancing Next-Generation High Performance Computing Systems”. In: (2020).

[13] et al. Peter Kogge. “ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems”. In: *DARPA* (2008).

[14] Gwan-Hyeob Koh. “ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems”. In: *2020 Symposium on VLSI Technology and Circuits (Short Course)* (2020).

[15] International Roadmap for Devices and Systems. “Beyond CMOS (2020 Edition)”. In: *IEEE* (2020).

[16] J. Liu and S. Chen. “Initial Experience with 3D XPoint Main Memory”. In: *IEEE 35th International Conference on Data Engineering Workshops (ICDEW)*. 2019, pp. 300–305. DOI: 10.1109/ICDEW.2019.00009.

[17] Y. Chen. “ReRAM: History, Status, and Future”. In: *IEEE Transactions on Electron Devices* 67.4 (2020), pp. 1420–1433. DOI: 10.1109/TED.2019.2961505.

[18] Bill Gervasi. “Architecture for Carbon Nanotube Based Memory (NRAM)”. In: *Hotchips30*. 2018.

[19] Mark Webb. “Annual Update on Emerging Memories”. In: *Flash Memory Summit*. 2018.

[20] Mustafa Pinarbasi. “MRAM: Memory for the Edge···and Beyond”. In: *Spin memory JEDEC Mobile & IoT Conference*. 2019.

[21] https://pcisig.com/pci-express%C2%AE-60-specification-track-revision-03-complete.

[22] https://pcisig.com/pcie%C2%AE-60-specification-version-05-now-available-member.

[23] https://dl.acm.org/doi/10.1145/3079856.3080246.

[24] https://cloud.google.com/blog/products/gcp/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu.

[25] https://cloud.google.com/tpu and https://www.matroid.com/scaledml/2018/jeff.pdf.

[26] https://cloud.google.com/tpu/docs/tpus.

[27] https://cloud.google.com/tpu/docs/system-architecture and https://dl.acm.org/doi/10.1145/3360307.

[28] https://cloud.google.com/blog/products/ai-machine-learning/google-breaks-ai-performance-records-in-mlperf-with-worlds-fastest-training-supercomputer.

[29] https://www.cerebras.net/introducing-the-cerebras-cs-1-the-industrys-fastest-artificial-intelligence-computer/.

[30] https://sc19.supercomputing.org/presentation/?id=exforum151&sess=sess369.

[31] https://www.hotchips.org/hc30/0tutorials/T2_Part_3_HotChips30TrainingTutorial.pdf.

[32] https://www.businesswire.com/news/home/20191119005046/en/Cerebras-Systems-Unveils-CS-1-Industry%E2%80%99s-Fastest-Artificial.

[33] https://secureservercdn.net/198.12.145.239/a7b.fcb.myftpupload.com/wp-content/uploads/2020/01/The-Cerebras-CS-1-Product-Overview-rev20200112.pdf.

[34] https://mlhardware.github.io/2020/cerebras.pdf.

[35] *Software Co-design for the First Wafer-Scale Processor (and Beyond)*. https://www.hotchips.org/program/.

[36] https://projects.preferred.jp/mn-core/.

[37] https://mlhardware.github.io/2020/pfn.pdf.

[38] https://mlhardware.github.io/2020/graphcore.pdf.

[39] https://www.graphcore.ai/developer.

[40] https://mlhardware.github.io/2020/groq.pdf.

[41] https://mlhardware.github.io/2020/sambanova.pdf.

[42] https://habana.ai/.

[43] https://www.riken.jp/press/2019/20191118_2/index.html.

[44] https://www.ssken.gr.jp/MAINSITE/event/2014/20141029-sci/lecture-02/SSKEN_sci2014_taiji_presentation.pdf.

[45] https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus.html.

[46] https://japan.xilinx.com/products/silicon-devices/acap/versal.html.

[47] https://www.intel.com/content/www/us/en/products/programmable/fpga/stratix-10/features.html.

[48] https://www.intel.com/content/www/us/en/products/programmable/fpga/agilex.html.

[49] C. E. Kozyrakis and D. A. Patterson. "Scalable, vector processors for embedded systems". In: *IEEE Micro* 23.6 (2003), pp. 36–45.

[50] F. Devaux. "The true Processing In Memory accelerator". In: *2019 IEEE Hot Chips 31 Symposium (HCS)*. 2019, pp. 1–24.

[51] M. M. Sabry Aly et al. "Energy-Efficient Abundant-Data Computing: The N3XT 1,000x". In: *Computer* 48.12 (2015), pp. 24–33.

[52] N. Tanabe et al. “A New Memory Module for COTS-Based Personal Supercomputing”. In: *Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'04)*. 2004, pp. 40–48.

[53] R. Nair et al. “Active Memory Cube: A processing-in-memory architecture for exascale systems”. In: *IBM Journal of Research and Development* 59.2/3 (2015), 17:1–17:14.

[54] J. T. Pawlowski. “Hybrid memory cube (HMC)”. In: *2011 IEEE Hot Chips 23 Symposium (HCS)*. 2011, pp. 1–24.

[55] P. Dlugosch et al. “An Efficient and Scalable Semiconductor Architecture for Parallel Automata Processing”. In: *IEEE Transactions on Parallel and Distributed Systems* 25.12 (2014), pp. 3088–3098.

[56] V. Seshadri et al. “Gather-Scatter DRAM: In-DRAM address translation to improve the spatial locality of non-unit strided accesses”. In: *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2015, pp. 267–280.

[57] Liu Ke et al. “RecNMP: Accelerating Personalized Recommendation with near-Memory Processing”. In: *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture(ISCA)*. 2020, pp. 790–803.

[58] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. “TensorDIMM: A Practical Near-Memory Processing Architecture for Embeddings and Tensor Operations in Deep Learning”. In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture(MICRO)*. 2019, pp. 740–753.

[59] 田邊昇 and 田浦健次朗. “ムーア則終焉直後に向けた高性能汎用計算機アーキテクチャの初期検討”. In: **情報処理学会研究報告**. Vol. 2019-HPC-172. 13. 2019, pp. 1–9.

[60] H. S. Stone. “A Logic-in-Memory Computer”. In: *IEEE Transactions on Computers* C-19.1 (1970), pp. 73–78.

[61] P. M. Kogge. “EXECUBE-A New Architecture for Scaleable MPPs”. In: *1994 International Conference on Parallel Processing Vol. 1*. Vol. 1. 1994, pp. 77–84.

[62] N. Tanabe et al. “MEMOnet: network interface plugged into a memory slot”. In: *Proceedings IEEE International Conference on Cluster Computing. CLUSTER 2000*. 2000, pp. 17–26.

[63] Noboru Tanabe et al. “A Memory Accelerator with Gather Functions for Bandwidth-Bound Irregular Applications”. In: *Proceedings of the 1st Workshop on Irregular Applications: Architectures and Algorithms*. 2011, pp. 35–42.

[64] J. Ahn et al. “A scalable processing-in-memory accelerator for parallel graph processing”. In: *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 2015, pp. 105–117.

[65] L. Nai et al. “GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks”. In: *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2017, pp. 457–468.

[66] G. Dai et al. “GraphH: A Processing-in-Memory Architecture for Large-Scale Graph Processing”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.4 (2019), pp. 640–653.

[67] Qing Guo et al. “AC-DIMM: Associative Computing with STT-MRAM”. In: *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*. 2013, pp. 189–200.

[68] https://pc2.uni-paderborn.de/hpc-services/available-systems/noctua.

[69] https://www.calient.net/products/s-series-photonic-switch/.

[70] Bob Alverson et al. “Cray XC Series Network”. In: *https://www.cray.com/sites/default/files/resources/CrayXCNetw* (2012).

[71] グラフゴルフ. “The Order/degree Problem Competition”. In: *http://research.nii.ac.jp/graphgolf/* ().

[72] B. Towles et al. “Unifying on-chip and inter-node switching within the Anton 2 network”. In: *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 2014, pp. 1–12. DOI: 10.1109/ISCA.2014.6853238.

[73] 泰地 真弘人. “分子動力学計算専用計算機 MDGRAPE-4 の開発”. In: **学技術計算分科会** *2014* **年度会合, サイエンティフィック・システム研究会** (2014).

[74] https://datacenterfrontier.com/optics-to-the-chip-the-key-to-super-fast-networks-of-the-future/.

[75] https://www.lightwaveonline.com/optical-tech/components/article/14170143/rockley-photonics-demos-copackaged-optics-optoasic-switch-system-with-multiple-partners.

[76] https://ranovus.com/ranovus-announces-a-strategic-collaboration-with-ibm-te-connectivity-and-senko-advanced-components-for-design-and-delivery-of-multi-vendor-co-packaged-optics-solutions-for-data-centers/.

[77] https://www.snia.org/sites/default/files/SDC/2017/presentations/Storage_Architecture/Tutorial_Grider_Gary_Bent_John_Storage_Lessons_from_HPC_Extreme_Scale_Computing_Driving_High_Performance_Data_Storage%20%281%29.pdf.

[78] http://ceur-ws.org/Vol-1800/short4.pdf.

[79] https://www.elinfor.com/knowledge/with-the-rapid-growth-of-large-capacity-ssd-in-storage-market-will-it-replace-hdd-p-11265.

[80] https://www.snia.org/sites/default/files/DSI/2016/presentations/Osamu_Shimizu_Role_of_Tape_Technology_rev.pdf.

[81] https://www.snia.org/sites/default/files/ESF/FCoE-vs-iSCSI-vs-iSER-Final.pdf.

[82] https://www.snia.org/sites/default/files/ESF/SNIA-Block-File-Object-Storage-Webcast-Final.pdf.

[83] Reid Priedhorsky Alfred Torrez and Timothy Randles. “HPC container runtime performance overhead: At first order, there is none”. In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC ’19)*. 2019, poster.

[84] A. J. Younge et al. “A Tale of Two Systems: Using Containers to Deploy HPC Applications on Supercomputers and Clouds”. In: *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 2017, pp. 74–81.

[85] Stefan Lankes, Simon Pickartz, and Jens Breitbart. “HermitCore: A Unikernel for Extreme Scale Computing”. In: *Proceedings of the 6th International Workshop on Runtime and Operating Sys-*

tems for Supercomputers. ROSS '16. Kyoto, Japan: Association for Computing Machinery, 2016. ISBN: 9781450343879. DOI: 10.1145/2931088.2931093.

[86] Ali Raza et al. "Unikernels: The Next Stage of Linux's Dominance". In: *Proceedings of the Workshop on Hot Topics in Operating Systems*. HotOS '19. Bertinoro, Italy: Association for Computing Machinery, 2019, pp. 7–13. ISBN: 9781450367271. DOI: 10.1145/3317550.3321445.

[87] Lisa Gerhardt et al. "Shifter: Containers for HPC". In: *Journal of Physics: Conference Series* 898 (Oct. 2017), p. 082021. DOI: 10.1088/1742-6596/898/8/082021.

[88] .

[89] Yutaka Ishikawa. "富岳スーパーコンピュータの開発状況". In: **第 *2* 回 *HPCI* コンソーシアムシンポジウム**. 2019.

[90] T. Shimosawa et al. "Interface for heterogeneous kernels: A framework to enable hybrid OS designs targeting high performance computing on manycore architectures". In: *2014 21st International Conference on High Performance Computing (HiPC)*. 2014, pp. 1–10.

[91] Jagan Meena et al. "Overview of Emerging Non-volatile Memory Technologies". In: *Nanoscale Research Letters* 9 (Sept. 2014), pp. 1–33. DOI: 10.1186/1556-276X-9-526.

[92] *Intel® Optane™ Persistent Memory*.

[93] *Summit supercomputer*.

[94] *Aurora supercomputer*.

[95] William M. Jones, John T. Daly, and Nathan DeBardeleben. "Application Monitoring and Checkpointing in HPC: Looking towards Exascale Systems". In: *Proceedings of the 50th Annual Southeast Regional Conference*. ACM-SE '12. Tuscaloosa, Alabama: Association for Computing Machinery, 2012, pp. 262–267. ISBN: 9781450312035. DOI: 10.1145/2184512.2184574.

[96] Rishiyur S. Nikhil. "Technical Perspective: Back to the Edge". In: *Commun. ACM* 62.6 (May 2019), p. 82. ISSN: 0001-0782. DOI: 10.1145/3323921.

[97] Raghu Prabhakar et al. "Generating Configurable Hardware from Parallel Patterns". In: *ASPLOS '16: SIGPLAN Not.* 51.4 (Mar. 2016), p. 651665. ISSN: 0362-1340. DOI: 10.1145/2954679.2872415.

[98] David Koeplinger et al. "Automatic Generation of Efficient Accelerators for Reconfigurable Hardware". In: *Proceedings of the 43rd International Symposium on Computer Architecture*. ISCA '16. Seoul, Republic of Korea: IEEE Press, 2016, p. 115127. ISBN: 9781467389471. DOI: 10.1109/ISCA.2016.20.

[99] Raghu Prabhakar et al. "Plasticine: A Reconfigurable Architecture For Parallel Paterns". In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ISCA '17. Toronto, ON, Canada: Association for Computing Machinery, 2017, p. 389402. ISBN: 9781450348928. DOI: 10.1145/3079856.3080256.

[100] L. Gan et al. "Solving Mesoscale Atmospheric Dynamics Using a Reconfigurable Dataflow Architecture". In: *IEEE Micro* 37.4 (2017), pp. 40–50.

[101] H. Fu et al. "Scaling Reverse Time Migration Performance through Reconfigurable Dataflow Engines". In: *IEEE Micro* 34.1 (2014), pp. 30–40.

[102] O. Pell and V. Averbukh. “Maximum Performance Computing with Dataflow Engines”. In: *Computing in Science Engineering* 14.4 (2012), pp. 98–103.

[103] Vivek Sarkar and John Hennessy. “Partitioning Parallel Programs for Macro-Dataflow”. In: *Proceedings of the 1986 ACM Conference on LISP and Functional Programming*. LFP ’86. Cambridge, Massachusetts, USA: Association for Computing Machinery, 1986, p. 202211. ISBN: 0897912004. DOI: 10.1145/319838.319863.

[104] Zhaoshi Li et al. “Aggressive Pipelining of Irregular Applications on Reconfigurable Hardware”. In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ISCA ’17. Toronto, ON, Canada: Association for Computing Machinery, 2017, p. 575586. ISBN: 9781450348928. DOI: 10.1145/3079856.3080228.

[105] T. Hirofuchi and R. Takano. “RAMinate: Hypervisor-based Virtualization for Hybrid Main Memory Systems”. In: *the Seventh ACM Symposium on Cloud Computing*. 2016.

[106] J. Hyun Kim et al. “µSnap: Embracing Traditional Programming Models for Persistent Memory through OS Support”. In: *IEEE 7th Non-Volatile Memory Systems and Applications Symposium*. 2018.

[107] Amrita Mathuriya et al. “CosmoFlow: Using Deep Learning to Learn the Universe at Scale”. In: *ArXiv* arXiv:1808.04728 (2018).

[108] *Systems and Architectures, International Roadmap for Devices and Systems (IRDS™) 2020 Edition*. 2020 (accessed September 22, 2020).

[109] Miwako Tsuji et al. “Multiple-SPMD Programming Environment based on PGAS and Workflow toward Post-Petascale Computing”. In: *Proceedings of the 2013 International Conference on Parallel Processing (ICPP-2013)*. IEEE, 2013, pp. 480–485.

[110] Daniele Lezzi et al. “Enabling e-Science Applications on the Cloud with COMPSs”. In: *Euro-Par 2011: Parallel Processing Workshops*. Ed. by Michael Alexander et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 25–34. ISBN: 978-3-642-29737-3.

[111] *https://www.bsc.es/research-and-development/projects/eflows4hpc-enabling-dynamic-and-intelligent-workflows-the-future*.

[112] H. Carter Edwards, Christian R. Trott, and Daniel Sunderland. “Kokkos: Enabling manycore performance portability through polymorphic memory access patterns”. In: *Journal of Parallel and Distributed Computing* 74.12 (2014). Domain-Specific Languages and High-Level Frameworks for High-Performance Computing, pp. 3202–3216. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2014.07.003.

[113] H. Carter Edwards et al. *Kokkos C++ Performance Portability Programming EcoSystem*. https://github.com/kokkos/kokkos. 2014.

[114] Kate Gregory and Ade Miller. *C++ AMP*. Microsoft Press, 2012.

[115] Edward A Luke and Thomas George. “Loci: A rule-based framework for parallel multi-disciplinary simulation synthesis”. In: *Journal of Functional Programming* 15.3 (2005), p. 477.

[116] Ronald Garcia and Andrew Lumsdaine. “MultiArray: a C++ library for generic programming with arrays”. In: *Software: Practice and Experience* 35.2 (2005), pp. 159–188.

[117] Intel. https://www.cilkplus.org/.

[118] *https://github.com/LLNL/RAJA*.

[119] Kevin Scott. "On Proebsting's Law". In: *echnical Report CS-2001-12. University of Virginia* (Apr. 2001).

[120] Mary Hall, David Padua, and Keshav Pingali. "Compiler Research: The next 50 Years". In: *Commun. ACM* 52.2 (Feb. 2009), pp. 60–67. ISSN: 0001-0782. DOI: 10.1145/1461928.1461946.

[121] Charles E. Leiserson et al. "There' s plenty of room at the Top: What will drive computer performance after Moore' s law?" In: *Science* 368.6495 (2020). ISSN: 0036-8075. DOI: 10.1126/science.aam9744. eprint: https://science.sciencemag.org/content/368/6495/eaam9744.full.pdf.

[122] LLVM vector predication roadmap. *https://llvm.org/docs/Proposals/VectorPredication.html*.

[123] P. Balaprakash et al. "Autotuning in High-Performance Computing Applications". In: *Proceedings of the IEEE* 106.11 (2018), pp. 2068–2083.

[124] A. Tiwari et al. "A scalable auto-tuning framework for compiler optimization". In: *2009 IEEE International Symposium on Parallel Distributed Processing*. 2009, pp. 1–12.

[125] A. Hartono, B. Norris, and P. Sadayappan. "Annotation-based empirical performance tuning using Orio". In: *2009 IEEE International Symposium on Parallel Distributed Processing*. 2009, pp. 1–11.

[126] Q. Yi et al. "POET: Parameterized Optimizations for Empirical Tuning". In: *2007 IEEE International Parallel and Distributed Processing Symposium*. 2007, pp. 1–8.

[127] Kazuhiko Komatsu et al. "Xevolver: A code transformation framework for separation of system-awareness from application codes". In: *Concurrency and Computation: Practice and Experience* 32.7 (2020), e5577.

[128] T. Katagiri and D. Takahashi. "Japanese Autotuning Research: Autotuning Languages and FFT". In: *Proceedings of the IEEE* 106.11 (2018), pp. 2056–2067.

[129] Yukinori Sato, Tomoya Yuki, and Toshio Endo. "An Autotuning Framework for Scalable Execution of Tiled Code via Iterative Polyhedral Compilation". In: *ACM Trans. Archit. Code Optim.* 15.4 (Jan. 2019). ISSN: 1544-3566. DOI: 10.1145/3293449.

[130] *Cambridge University Study States Software Bugs Cost Economy $312 Billion Per Year*. 2013 (accessed September 30, 2020).

[131] *Valgrind*. 2020 (accessed September 30, 2020).

[132] *DDT*. 2020 (accessed September 30, 2020).

[133] *TotalView*. 2020 (accessed September 30, 2020).

[134] *ReMPI: MPI Record-and-Replay*. 2020 (accessed September 30, 2020).

[135] *Ninja: Noise Injection Agent Tool*. 2020 (accessed September 30, 2020).

[136] Anh Vo et al. "Formal Verification of Practical MPI Programs". In: *SIGPLAN Not.* 44.4 (Feb. 2009), pp. 261–270. ISSN: 0362-1340. DOI: 10.1145/1594835.1504214.

[137] A. Vo et al. "A Scalable and Distributed Dynamic Formal Verifier for MPI Programs". In: *SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. 2010, pp. 1–10.

[138] Ignacio Laguna et al. "Large Scale Debugging of Parallel Tasks with AutomaDeD". In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage*

and Analysis. SC '11. Seattle, Washington: Association for Computing Machinery, 2011. ISBN: 9781450307710. DOI: 10.1145/2063384.2063451.

[139] Ehab Al-Shaer and Saeed Al-Haj. "FlowChecker: Configuration Analysis and Verification of Federated Openflow Infrastructures". In: *Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration*. SafeConfig '10. Chicago, Illinois, USA: Association for Computing Machinery, 2010, pp. 37–44. ISBN: 9781450300933. DOI: 10.1145/1866898.1866905.

[140] A. Hück et al. "Compiler-aided Type Tracking for Correctness Checking of MPI Applications". In: *2018 IEEE/ACM 2nd International Workshop on Software Correctness for HPC Applications (Correctness)*. 2018, pp. 51–58.

[141] S. Atzeni et al. "ARCHER: Effectively Spotting Data Races in Large OpenMP Applications". In: *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2016, pp. 53–62.

[142] G. Sawaya et al. "FLiT: Cross-platform floating-point result-consistency tester and workload". In: *2017 IEEE International Symposium on Workload Characterization (IISWC)*. 2017, pp. 229–238.

[143] *Artifact Review and Badging*. 2020 (accessed September 30, 2020).

[144] Peter Hans Graf. *Deep Learning with Vector Processor*. 2nd NEC Aurora Forum. 2018.

[145] *Intel Math Kernel Library for Deep Learning Networks*. 2020 (accessed September 23, 2020).

[146] *OneAPI Deep Neural Network Library*. 2020 (accessed September 23, 2020).

[147] *NVIDIA cuDNN*. 2020 (accessed September 23, 2020).

[148] *Vector Engine DNN Library*. 2020 (accessed September 23, 2020).

[149] Matei Zaharia et al. "Apache Spark: A Unified Engine for Big Data Processing". In: *Commun. ACM* 59.11 (Oct. 2016), pp. 56–65. ISSN: 0001-0782. DOI: 10.1145/2934664.

[150] *Google BigQuery*. 2020 (accessed October 05, 2020).

[151] M. Matsuda, N. Maruyama, and S. Takizawa. "K MapReduce: A scalable tool for data-processing and search/ensemble applications on large-scale supercomputers". In: *2013 IEEE International Conference on Cluster Computing (CLUSTER)*. 2013, pp. 1–8.

[152] *Frovedis: FRamework Of VEctorized and DIStributed data analytics*. 2020 (accessed October 05, 2020).

[153] 南一生. **第 *2* 章アプリケーションの性能最適化**. 2018.

[154] Mitsuhisa Sato. *Co-design for Post-K ARM for HPC - Co-design Opportunities, Birds of a Feather, ISC*. 2018.

[155] gem5. *https://www.gem5.org/*.

[156] gem5-gpu. *https://github.com/gem5-gpu/*.

[157] Kuba Kaszyk et al. "Full-System Simulation of Mobile CPU/GPU Platforms". In: *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2019, pp. –. DOI: 10.1109/ISPASS.2019.00015.

[158] Kyle L. Spafford and Jeffrey S. Vetter. "Aspen: A domain specific language for performance modeling". In: *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 2012, pp. 1–11. DOI: 10.1109/SC.2012.20.

[159] Ivy B. Peng and Jeffrey S. Vetter. "Siena: exploring the design space of heterogeneous memory systems". In: *SC '18: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. 2018, pp. 427–440. DOI: 10.1109/SC.2018.00036.

[160] Curtis L. Janssen et al. "A simulator for large-scale parallel architectures". In: *International Journal of Parallel and Distributed Systems* 1.2 (2010), pp. 57–73.

[161] Sandia National Labs. *http://sst-simulator.org/*.

[162] McPat. *https://www.hpl.hp.com/research/mcpat/*.

[163] Sheng Li et al. "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures". In: *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2009, pp. 469–480.

[164] Miwako Tsuji, Taisuke Boku, and Mitsuhisa Sato. "Scalable communication performance prediction using auto-generated pseudo MPI event trace". In: *HPCAsia2019*. 2019, pp. 53–62. DOI: 10.1145/3293320.3293323.

[165] Alexandru Calotoiu et al. "Using automated performance modeling to find scalability bugs in complex codes". In: *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 2013, pp. 1–12. DOI: 10.1145/2503210.2503277.

[166] Alexandru Calotoiu et al. "Fast Multi-parameter Performance Modeling". In: *IEEE International Conference on Cluster Computing (CLUSTER)*. 2016, pp. 172–181. DOI: 10.1109/CLUSTER.2016.57.

[167] Marcus Ritter et al. "Learning Cost-Effective Sampling Strategies for Empirical Performance Modeling". In: *IEEE International Parallel and Distributed Processing Symposium (IPDPS2020)*. 2020, pp. 884–895. DOI: 10.1109/IPDPS47924.2020.00095.

[168] Dieter Mey et al. "Score-P: A Unified Performance Measurement System for Petascale Applications". In: *Competence in High Performance Computing 2010*. 2010, pp. 85–97. DOI: 10.1007/978-3-642-24025-6_8.

[169] Sergei Shudler et al. "Isoefficiency in Practice: Configuring and Understanding the Performance of Task-based Applications". In: *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '17)*. 2017, pp. 131–143. DOI: 10.1145/3018743.3018770.

[170] Qulacs. *http://docs.qulacs.org/en/latest/*.

[171] Bo Wang et al. "Dynamic Application-Aware Power Capping". In: *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*. E2SC'17. Denver, CO, USA: Association for Computing Machinery, 2017. ISBN: 9781450351324. DOI: 10.1145/3149412.3149413.

[172] Cedric Nugteren, Gert-Jan van den Braak, and Henk Corporaal. "Roofline-Aware DVFS for GPUs". In: *Proceedings of International Workshop on Adaptive Self-Tuning Computing Systems*. ADAPT '14. Vienna, Austria: Association for Computing Machinery, 2014, pp. 8–10. ISBN: 9781450325141. DOI: 10.1145/2553062.2553067.

[173] K. Ahmed, K. Yoshii, and S. Tasnim. "Thermal-Aware Power Capping Allocation Model for High Performance Computing Systems". In: *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. 2019, pp. 1488–1493.

[174] Yuichi Inadomi et al. “Analyzing and Mitigating the Impact of Manufacturing Variability in Power-Constrained Supercomputing”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’15. Austin, Texas: Association for Computing Machinery, 2015. ISBN: 9781450337236. DOI: 10.1145/2807591.2807638.

[175] O. Sarood et al. “Optimizing power allocation to CPU and memory subsystems in overprovisioned HPC systems”. In: *2013 IEEE International Conference on Cluster Computing (CLUSTER)*. 2013, pp. 1–8.

[176] T. Komoda et al. “Power capping of CPU-GPU heterogeneous systems through coordinating DVFS and task mapping”. In: *2013 IEEE 31st International Conference on Computer Design (ICCD)*. 2013, pp. 349–356.

[177] Eishi Arima et al. “Footprint-Aware Power Capping for Hybrid Memory Based Systems”. In: *High Performance Computing*. Ed. by Ponnuswamy Sadayappan et al. Cham: Springer International Publishing, 2020, pp. 347–369. ISBN: 978-3-030-50743-5.

[178] *The HPC PowerStack*. 2020 (accessed September 17, 2020).

[179] C. D. Martino et al. “Lessons Learned from the Analysis of System Failures at Petascale: The Case of Blue Waters”. In: *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 2014, pp. 610–621.

[180] R. A. Ashraf and C. Engelmann. “Analyzing the Impact of System Reliability Events on Applications in the Titan Supercomputer”. In: *2018 IEEE/ACM 8th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS)*. 2018, pp. 39–48.

[181] Jon Calhoun et al. “Exploring the feasibility of lossy compression for PDE simulations”. In: *The International Journal of High Performance Computing Applications* 33.2 (2019), pp. 397–410. DOI: 10.1177/1094342018762036. eprint: https://doi.org/10.1177/1094342018762036.

[182] *Long term failure analysis of 10 petascale supercompute*. 2020 (accessed September 17, 2020).

[183] Era Ajdaraga Krluku, Marjan Gusev, and Vladimir Zdraveski. “Bi-Source Verification Against Silent Data Corruption in High Performance Computing”. In: *Proceedings of the 9th Balkan Conference on Informatics*. BCI’19. Sofia, Bulgaria: Association for Computing Machinery, 2019. ISBN: 9781450371933. DOI: 10.1145/3351556.3351567.

[184] Leonardo Bautista-Gomez and Franck Cappello. “Detecting Silent Data Corruption for Extreme-Scale MPI Applications”. In: *Proceedings of the 22nd European MPI Users’ Group Meeting*. EuroMPI ’15. Bordeaux, France: Association for Computing Machinery, 2015. ISBN: 9781450337953. DOI: 10.1145/2802658.2802665.

[185] Panruo Wu et al. “Silent Data Corruption Resilient Two-Sided Matrix Factorizations”. In: *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. PPoPP ’17. Austin, Texas, USA: Association for Computing Machinery, 2017, pp. 415–427. ISBN: 9781450344937. DOI: 10.1145/3018743.3018750.

[186] Sebastian Angel, Mihir Nanavati, and Siddhartha Sen. “Disaggregation and the Application”. In: *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*. 2020.

[187] Paolo Faraboschi and et al. “Beyond processor-centric operating systems”. In: *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*. 2015.

[188] Kevin Lim and et al. “Disaggregated memory for expansion and sharing in blade servers”. In: *ACM SIGARCH computer architecture news* 37.3 (2009), pp. 267–278.

[189] Kostas Katrinis and et al. “Rack-scale disaggregated cloud data centers: The dReDBox project vision”. In: *2016 Design, Automation & Test in Europe Conference and Exhibition (DATE)*. 2016.

[190] Y. Shan et al. “LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation”. In: *the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2018.

[191] A. Roozbeh et al. “Software-Defined “Hardware” Infrastructures: A Survey on Enabling Technologies and Open Research Directions”. In: *IEEE Communications Surveys & Tutorials* 20.3 (2018), pp. 2454–2485.

[192] J. Shalf et al. “Photonic Memory Disaggregation in Datacenters”. In: *OSA Advanced Photonics Congress (AP) 2020*. 2020.

[193] J. Duato et al. “rCUDA: Reducing the number of GPU-based accelerators in high performance clusters”. In: *2010 International Conference on High Performance Computing & Simulation*. 2010, pp. 224–231.

[194] J. Gu et al. “Efficient memory disaggregation with InfiniSwap”. In: *In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 2017, pp. 649–667.

[195] Blake Caldwell et al. “FluidMem: Memory as a Service for the Datacenter”. In: *arXiv:1707.07780*. 2017.

[196] Gen-Z. Consortium et al. “Gen-z overview (Available: http://genzconsortium. org/wp-content)”. In: *Tech. Rep.* (2016).

[197] D. Sharma. “Compute Express Link (Available: https://docs.wixstatic.com/ugd/0c1418d9878707bbb7427786b70c3 in: *CXL Consortium White Paper*. 2019.

[198] *Prime Cloud Controller*. URL: `http://www.primecloud-controller.org/`.

[199] *Altair PBS Works*. URL: `https://www.altair.com/pbs-works/`.

[200] *AWS IoT Core*. URL: `https://aws.amazon.com/jp/iot-core/`.

[201] *Azure IoT*. URL: `https://azure.microsoft.com/ja-jp/overview/iot/`.

[202] *Google Cloud IoT*. URL: `https://cloud.google.com/solutions/iot/`.

[203] **恒速度ファイル転送**. URL: `https://www.sinet.ad.jp/connect_service/service/cbrftp/`.

[204] *FDT:Fast Data Transfer*. URL: `https://fast-data-transfer.github.io/`.

[205] “mdtmFTP and its evaluation on ESNETSDN testbed”. In: *Future Generation Computer System* 79 (2018), pp. 199–204.

[206] Amy N Langville and Carl D Meyer. *Google's PageRank and beyond: The science of search engine rankings*. Princeton university press, 2011.

[207] `https://graph500.org/`.

[208] `https://graphchallenge.mit.edu/`.

[209] Maciej Besta et al. “To push or to pull: On reducing communication and synchronization in graph computations”. In: *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. 2017, pp. 93–104.

[210] `https://www.itmedia.co.jp/news/articles/2007/02/news093.html`.

第 3 章

アプリケーションの要求性能分析

3.1　次世代システムの予測性能とアプリケーションの要求性能

本節ではアプリケーションの性能要求を分析するにあたり，まずは 2028 年頃に実現される最も汎用的なシステムを仮定しその性能を予測する．将来のシステム構成を検討するにあたり，2019 年から 2020 年にかけてサイエンス向け（従来のシステムの延長），新規応用分野向け，そして新利用形態（Society5.0）向けの構成を予測する分科会をそれぞれ複数回にわたって開催し，サイエンスロードマップを達成するために最適と考えられるシステム構成を議論した．その中でサイエンス向けとして 37 種類のアプリケーションに関して分析を行い，必要な演算性能・メモリ量・メモリ帯域・通信性能・ストレージなどに関して議論を行った．また，特定のアプリケーションではなく，計算手法や計算問題などに対して議論を行い，必要とされるシステム構成も分析した．さらに，Society5.0 などに代表される従来型のスパコン利用とは異なる利用形態をとる可能性のあるアプリケーション向けには，どのような構成が必要となるのかなど議論を行った．これらの分科会での議論の結果，予測される汎用的な計算機システムに対して，各アプリケーションが必要とするシステム構成は異なっており，特にプロセッサ・メモリ・ネットワーク・ストレージの中で重視される構成は複数に分類されることが分かった．また，従来の汎用的なシステム構成では高性能が得られていない計算手法や計算問題では，その計算や処理向けに特化された専用ハードウェア（FPGA や専用回路など）の利用による計算性能の向上が期待できることも分かった．新利用形態では，計算性能の要求よりもスパコンと外部システムの連携プラットフォームが重要であると示された．その中でもデータを利用・収集・蓄積するためのネットワークやデータ解析環境，それらをリアルタイムに処理することができる計算環境，そこで実行されるサービスを維持するための広域分散共用計算機環境など，これまでに実装されていないシステム環境が必須であると分かった．

将来のサイエンス向け・従来型の汎用的システム構成を検討する前提として，2020 年に導入され世界最高性能となったスーパーコンピュータ富岳とほぼ同程度の消費電力（30MW〜50MW）で実現できることを制約条件とする．

詳細は次章で述べるが，本ロードマップでは CPU ベースの汎用システムを検討するにあたり，システム全体，および CPU の電力バジェットとしてそれぞれ 3 種類の値を仮定し，計 9 つのシステムの性能予測を行った．その 9 つのシステムの中で最も高性能なシステム全体の電力バジェットが 50MW で，かつ CPU の電力バジェットが 80% のシステムを前提に，計算科学ロードマップに記載されたアプリケーションの要求性能と比較する．ただし，計算科学ロードマップにはソケット数に関する記述がないため，本書では「ソケット数＝ノード数」として比較を行う．

前提とする 2028 年に実現される CPU・ネットワーク・ストレージ性能，メモリ性能，およびシステム性能

表 3.1　2028 年頃における CPU・ネットワーク・ストレージの予測性能

CPU	コア数	70
	周波数 (GHz)	3.9
	SIMD 長 (bit)	2048
	SIMD ポート数	2
	TFLOPS	34.9
	TDP (W)	351
NW	バンド幅/リンク (Tb/s)	1.6
Storage	容量 (EB)	3.45
	スループット (GB/s/node)	7.38

表 3.2　2028 年頃におけるメモリの予測性能

	DDR	HBM
チャネル数	12	6
バンド幅 (GB/s)	184	1100
総バンド幅 (GB/s)	2208	6600
総容量 (GB)	384	96

表 3.3　2028 年頃におけるシステムの予測性能（PUE1.1 を想定）

	30MW, CPU60%	50MW, CPU80%
ソケット数	46620	103600
総コア数	3263400	7252000
PFLOPS (倍精度)	1629	3620
DDR 総バンド幅 (PB/s)	102	228
HBM 総バンド幅 (PB/s)	307	683
DDR 総メモリ量 (PB)	17	39
HBM 総メモリ量 (PB)	4	9

はそれぞれ表 3.1，表 3.2，表 3.3 のようになる．分科会で分析したアプリケーションの要求する性能がこの予測システムで十分かを図 3.1 に示す．計算科学ロードマップにならい，縦軸を要求 B/F 比，横軸を要求メモリ容量として各アプリケーションをグラフにマップしている．図の赤，橙，灰色のポイントは，それぞれ，想定システムが要求性能を満たす，メモリに関する要求を満たさないアプリケーション，メモリに関する要求とネットワークに関する要求の双方を満たさないアプリケーションを表す．なお，計算科学ロードマップのアプリケーションの中には要求ノード数が記載されていないものがあり，そのようなアプリケーションはグラフから除外してある．一部外れるアプリケーションもあるが，予測システムでカバーできないアプリケーションの多くは要求メモリ量が 40PB を超えるか，要求メモリバンド幅／演算性能（B/F）が 1.4 を超えている．その他では，メモリ帯域や演算性能自体が不足している．1 つのアプリケーションのみネットワーク性能だけが予測システムの性能では不足しているものもあった．これらの結果から，要求される性能は以下の 4 種類に分類するのが適当であると結論付けられた．

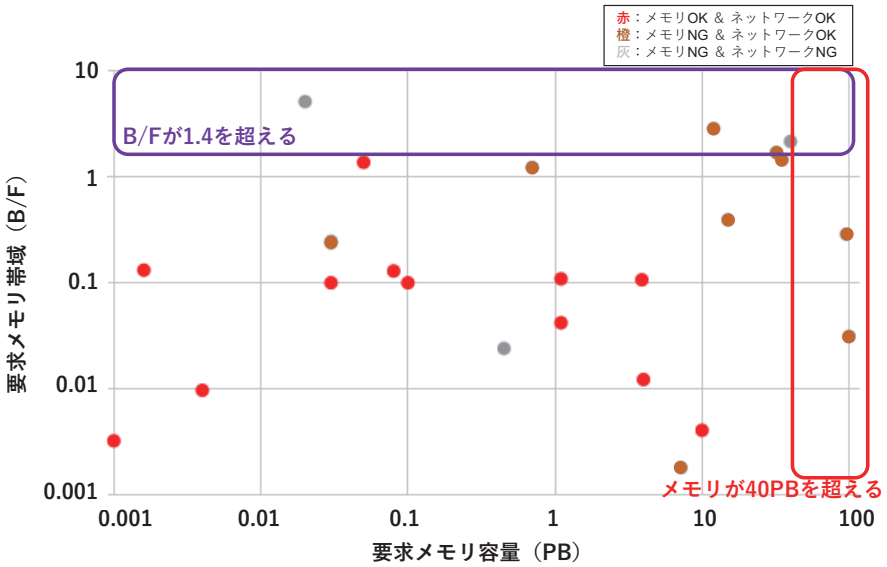

図 3.1　2028 年に予測されるシステム構成に対するアプリケーションの要求性能の対比（システム電力 50MW，CPU 電力 80% の場合）

- **汎用型**: 従来の計算機システム的であり，汎用的に様々な問題に適用可能な構成
- **メモリ容量・帯域重視型**: 汎用型から演算性能を落としてメモリ性能により多くの資源を割いた構成
- **演算重視型**: 汎用型からより演算性能を高めた構成
- **ネットワーク重視型**: 他計算ノードとの高い通信性能が必須のネットワーク重視の構成

3.2　要求性能に対するアプリケーション分析

3.1 節で示したように，今回対象としたアプリケーションの多くは，汎用システム型の性能を要求しており，従来型の性能仕様の延長を必要としている．様々な分野のアプリケーションが含まれており，汎用システム型計算機に対する期待は依然として高い．一方で，従来型の延長では要求を満たせないアプリケーションも多数ある．それらのアプリケーションが要求する性能はいくつかに分かれるが，ここでは，メモリ性能，演算性能，ネットワーク性能により分類した．さらに，その性能を要求する理由についても分析する．これらは，汎用型システムと比較してある項目について数倍の性能を要求していることから，新しく研究開発が必要な項目である．また，次世代システムのハードウェアへの要求ではないが，ポスト処理に関する要求もあった．さらに，汎用型システムの延長ではなく，全く別の専用的なハードウェアやシステムを要求するアプリケーションや計算問題もあった．これらに関しても分析を行った．

3.2.1　汎用システム型要求アプリケーション

今回検討したアプリケーション中では，以下のアプリケーションやアプリケーション分野が従来の延長である汎用システム型を要求している．

- 素粒子・原子核アプリケーション（具体的な名称は不明）
- rmcsm
- paraDMRG
- GELLAN
- MODYLAS
- 生命科学
- 創薬医療× 6
- FFB
- ADVENTURE
- Front-Comp
- GT5D
- WHC
- Realtime cerebellum
- NEURON + Stochastic
- SCALE-RM
- P3T

従来型の延長とは言え，微細化などのデバイス技術の発展が難しくなっている現状では，ここで予測している汎用システム型の性能を実現するのも簡単ではなく，これまで以上の研究開発が必要となる．

3.2.2 メモリ性能要求アプリケーション

汎用システム型の従来性能仕様の延長では，要求性能が十分に満たせないアプリケーションの中で，メモリに対する性能要求として，容量・転送速度・アクセス手法といった面での拡張を必要としていた．メモリバンド幅への要求はこれまでにも多くあり，計算機システムとして対応が望まれる項目である．

3.2.2.1 高メモリバンド幅の要求

以下に高メモリバンド幅を要求するアプリケーションと，その要件・理由について述べる．

- NTChem：予測性能の 2.5 倍のメモリバンド幅が要求されている．
- SMASH：Hartree-Fock 法および密度汎関数法計算において最も計算時間を要する二電子積分の見積り計算のため，予測性能より 2.67 倍のメモリバンドが必要となる．
- バリオン 2 体波動関数計算：時間方向に 20 点程度計算する必要があり，この計算には高速フーリエ変換 (FFT) が用いられるため，高いメモリバンド（予測性能の 3.33 倍）が必要となる．
- NICAM：陽解法計算時において予測性能ではメモリバンドが不足する．メモリ・キャッシュの多階層化に伴うレイテンシの増加を最小限にする必要がある．
- LQCD：行列ベクトルメモリアクセスにおいて，予測性能ではメモリバンドが不足する．
- GAMERA：非構造格子のランダムアクセスがあり，メモリバンド幅の高速化が必要．また，従来型の SIMD ではランダムアクセスが非常に高コストになってしまうため，近年の GPU に搭載されているような SIMD の各レーンにおいて柔軟に効率的なデータアクセスが実行できる手法が必要である．また，SIMD レーン数・コア数の増加に伴いデータレカレンスがあるとオーバヘッドが大きくなるため，ハー

ドウェアによるサポートが必要である．

- SCALE-RM：ステンシル計算を SIMD 化するための高速な転置やストライピングロードのために高いメモリバンド幅が必要となる．

3.2.2.2 大メモリ容量の要求

以下に，より大きなメモリ容量を要求するアプリケーションと，その要件・理由について述べる．

- NTChem：電子相関計算では，中間データ量が系の 3 乗もしくは 4 乗に比例して増加するため，各ノードに分散させてもノード当たりの必要メモリ量は多くなり，予測値ではメモリが不足する（予測性能の 5.83 倍の容量が必要）．
- 大気環境 CHASER-LTKF：格子サイズと時間積層分から計算される要求メモリサイズは予測性能の 3.33 倍になり，容量が不足する（格子点数: 1200 万，水平解像度 80km， 鉛直 60 層と， ステップ数: 530 万 (dt=1 分, 10 年積分)，アンサンブル数: 100 メンバを想定）．
- EM-PIC：プラズマ粒子の数を 10〜100 倍利用するためにはメモリ容量が不足する．

3.2.2.3 特別なメモリアクセス手法の要求

以下は，特別なメモリアクセスパターンに対しての高速化を要求するアプリケーションである．

- SCALE-SDM：粒子-流体の連成計算をするために，配列に対するリストアクセスが必要となる．

3.2.3 高い演算性能要求

高い演算性能への要求には，ノード数を増やすことでシステム全体の演算性能向上を要求するものや，最小プロセス実行単位であるコアの数を増やしノード性能の向上を要求するもの，SIMD 演算など演算機構自体に対する要求があった．一方で，SIMD 演算の bit 幅が使い切れず，減らすことを要求するアプリケーションもあり，bit 幅が増える予測次世代システムの仕様とは反対の要求となっているものも存在する．

3.2.3.1 ノード数要求

ノード数増強による高演算性能を要求するアプリケーションについて述べる．

- GAMERA：100m 格子間隔で 1000 シナリオの計算をするためには，予測性能の 1.93 倍の演算性能が必要となる．
- NICAM：格子点数の増加に伴う計算時間増大をカバーするためには，予測性能の 3.86 倍の演算性能が必要となる．
- データ同化 NICAM-LETKF：データ同化部分の計算量増加のため，予測性能の 1.54 倍演算性能が必要となる．
- AmaTeRAS：全太陽計算を高解像度で行うためには，予測性能の 9.65 倍の演算性能が必要となる．

3.2.3.2 ノード性能（コア数）要求

コア数増強による高演算性能を要求するアプリケーションについて述べる．

- CNN：認識において処理する動画データが巨大になるため，ノード性能が 100TF 必要となる．
- ニュートリノ輻射輸送：6 次元ボルツマン方程式を解く際に速度空間をノード内で解くために，予測性能より 5.71 倍高いノード演算性能が必要となる．

3.2.3.3 演算機構に関する要求

演算機構に対して特別の要求を持つアプリケーションについて述べる．

- LQCD：複素数を隣接 SIMD レーンで連携して演算することでベクトル長を半減できるような命令が必要[*1]．
- SCALE-RM：依存関係があるために SIMD 化が難しいループがあり，また巨大なループ計算においてレジスタやアウトオブオーダー資源が不足するため，それら資源の増強が必要．
- NICAM：SIMD 演算においてビット幅が大きくなると使い切れない可能性が高く，ビット幅を半分にして SIMD 演算ユニットを倍にするなどの工夫ができるハードウェア機構が必要．

3.2.4 ネットワーク性能要求

ネットワーク性能への要求としては，ノード間通信レイテンシへの要求はあまりなく，通信バンド幅を要求するアプリケーションがあった．これらは集団通信によるものがほとんどである．また，特別な専用的ネットワークを要求するアプリケーションもあった．

3.2.4.1 ノード間ネットワーク帯域を要求

ノード間ネットワークの高帯域化を要求するアプリケーションについて述べる．

- GreeM（N 体シミュレーション）：粒子間相互作用演算にかかわる集団通信のために予測性能の 16 倍のネットワーク帯域が必要となる．
- 精密格子 QCD：CG 並列計算のために 4 次元隣接通信が必要であり，その通信には予測性能の 34 倍のネットワーク帯域が必要となる．
- AmaTeRAS：輻射部分の計算に集団通信が必要であるため，予測性能の 26 倍のネットワーク帯域が必要となる．

3.2.4.2 特別なネットワーク機構を要求

特別なネットワークトポロジーを要求するアプリケーションについて述べる．

- 精密格子 QCD：Allrecue 専用ネットワークや 4 次元構造格子を並列化しやすいネットワークトポロジが必要．

[*1] A64FX では命令は存在するがマイクロコード実行で遅い．

3.2.5 ポスト処理性能要求

ハードウェア開発の面からは見落とされがちだが，計算機システムの性能が発展するにつれて出力データが巨大化しており，そのデータに対するケアも必要である．

- GENESIS：データ増減に対する入力/出力速度不足のため，リアルタイム可視化が必要となる．

3.2.6 専用的システムの要求

これまでに無い機能的・性能的特徴を持ったシステムを要求するアプリケーションや問題もあった．そのようなシステムの実現には研究開発が必須であり，これまで以上の計算機システム分野の発展が必要となる．要求される専用的なシステムは様々だが，今回分析したものでは下記のような機能・性能が要求されている．

- 負荷分散支援ハードウェア（粒子計算）
- エージェント処理のハードウェア実装（エージェントシミュレーション）
- GPU の不要な部分を削除した疎行列専用ハードウェア（疎行列計算）
- ランダムアクセスをメモリフレンドリな順番に変換する支援機構をもつハードウェア（モンテカルロ計算）
- 効率の良いオンチップバッファ・リダクション専用ネットワーク・軽量専用パイプライン・高並列・FFT 専用ハードウェア（N 体計算，粒子計算）
- メモリフレンドリな参照パターンとなるようにグラフ処理のスケジューリングを支援するハードウェア（グラフ処理）
- グラフ処理に向いたデータ再利用が実現できるキャッシュ・オンチップバッファ（グラフ処理）
- デジタル回路による Spike の符号化・処理と情報伝搬専用ネットワーク（ニューロモーフィック）
- 超伝導チップ・CMOS アニーリング・デジタルアニーラ，シミュレーテッド分岐マシン（イジング模型による最適化問題）
- FPGA などのデータフロー型かつ大量のレジスタをもつハードウェア・通信と計算のオーバラップ・低レイテンシ通信機構（FFT）
- ソーティングのハードウェアアルゴリズム実装（ソーティング）

3.3 アプリケーションの要求性能との比較

3.1 節の図 3.1 では，計算科学ロードマップのアプリケーション中システム消費電力 50MW，CPU の電力 80% のシステムがカバーするアプリケーション範囲を示した．その詳細な分析結果として，表 3.4 に各アプリケーションの要求性能と想定システムのピーク性能の比較結果を示す．表の各行はアプリケーションを表しており，表の第 3-9 列がアプリケーションの要求性能を，第 10–14 列が想定システムの各ハードウェアコンポーネントとの比較結果を表している．比較結果の列の○と×は，そのハードウェアコンポーネントがそのアプリケーションの要求性能を満たす，あるいは，満たさないことを表す．最も右端の列は総合評価結果であり，第 10–14 列に記載した全てのハードウェアコンポーネントがアプリケーションの要求性能を満たす場合に○，それ以外の場合は×としている．

表 3.4　各アプリケーションの要求性能と電力 50MW（CPU 電力 80%）のシステム性能の比較結果

分野	アプリケーション名	要求ノード数	演算性能 (TFlops /node)	メモリ帯域 (TB/s/ node)	メモリ容量 (GB/ node)	インターコネクト帯域 (Tb/s/ node)	ファイル I/O 帯域 (GB/s/ node)	ストレージ容量 (PB)	ノード	CPU	メモリ	インターコネクト	ストレージ	総合
素粒子・原子核		40,000	7.75	1	2	1.25	0.045	60	○	○	○	○	○	○
		4,100	7.561	0.073	0.976	0.088	0.039	5	○	○	○	○	○	○
		1,000	5.6	0	1	0	0.1	0.001	○	○	○	○	○	○
	rmcsm	6,000	5	0.5	5	0	0.017	0	○	○	○	○	○	○
		4,100	4.146	21.22	4.878	53.75	0.006	120	○	○	×	×	○	×
物質科学	HPhi	1,000,000	0.032	0.001	100	1.025	0.009	340	×	○	○	○	○	×
エネルギー・資源	NTChem	18,000	11.111	16.111	1,944.44	0.001	0.002	0.1	○	○	×	○	○	×
	SMASH	100,000	10	17	320	0.014	0.003	0.32	○	○	×	○	○	×
	paraDMRG	100,000	31	1.3	11	0.05	0.01	0	○	○	○	○	○	○
	GELLAN	100,000	2.7	0.033	40	0.018	0	0.02	○	○	○	○	○	○
	MODYLAS	100,000	5	0.5	1	0.003	0.001	500	○	○	○	○	○	○
脳科学・AI	WHC	50,000	30	3.2	78	0	0	1	○	○	○	○	○	○
	Realtime cerebellum	50,000	11.2	1.22	22	0	0	5	○	○	○	○	○	○
	NEURON K+ Stochastic	100,000	7.6	1	0.016	0	0.13	160	○	○	○	○	○	○
	CNN (Forward & Back Prop)	16,000	100	23.75	1.875	0.25	0.006	175	○	×	×	○	○	×
	CNN (Forward)	6,900	100	24.638	4.348	0.425	0.493	290	○	×	×	○	×	×
地震・津波	GAMERA	200,000	2.8	1.1	75	0.025	0.005	1	×	○	○	○	○	×
気象気候	NICAM	400,000	1.225	3.5	30	0	0	230	×	○	○	○	○	×
	SCALE-RM	40,000	1.15	1.575	1.25	0	0	33	○	○	○	○	○	○
	CHASER-LETKF	32,000	0.203	0.438	1,250.00	750	2.5	50	○	○	×	×	×	×
	NICAM-LETKF	160,000	1.375	1.688	4.375	0	0.5	10	×	○	○	○	×	×
宇宙・天文	GreeM	100,000	6.9	0.028	100	25	0.3	300	○	○	○	×	○	×
		100,000	200	0.36	72	0.125	0.06	2,600.00	○	×	○	○	○	×
	EM-PIC	100,000	1.6	0.46	960	0.125	10	2,000.00	○	○	×	○	×	×
	P3T	10,000	3.1	0.01	0.1	0.001	0.001	400	○	○	○	○	○	○
	AmaTeRAS	1,000,000	1	0.024	0.45	41.25	0	20	×	○	○	×	○	×

図 3.1 に示したように，想定システムは要求メモリ容量が 40PB 未満，かつ，要求 B/F 比が 1.4 未満のアプリケーション（計 18 個）の多く（計 12 個）をカバーする．その一方で，40PB 以上のメモリ，あるいは，1.4 以上の B/F 比を必要とするアプリケーション（計 7 個）はカバーしない．また，表に示すように，従来の汎用システムの延長では 960GB/node 以上の超大容量メモリ，あるいは，16TB/s/node を超える超高メモリ帯域の実現は困難であり，これらのアプリケーションの要求性能を満たすためには，4.2 節の中で述べるようなドメイン特化型のシステムの利用や，演算に必要なデータビット幅を積極的に低ビット化して実効的なバンド幅をあげるなどの方策が必要と考えられる．

次に，要求メモリ容量が 40PB 未満，かつ，要求 B/F 比が 1.4 未満のアプリケーションに着目する．表より，想定システムが要求性能を満たせないアプリケーションは，160,000 以上のノード数（NICAM-LETKF），100TFLOPS/node 以上の CPU 性能（CNN），23.75TB/s/node 以上のメモリ帯域（CNN），25Tb/s/node 以上のインターコネクト帯域（GreeM），0.493GB/s/node 以上のファイル I/O 帯域（CNN）を要求している．ノード数，あるいは，CPU 性能が要求に満たないアプリケーションについては，より低性能な CPU を用いて大ノード数化する，あるいは，より高性能な CPU を用いて小ノード数化することで要求性能を満たせる可能性があり，次世代システムの最適なノード数については今後検討の余地がある．一方，メモリ，インターコネクト，ファイル I/O の各帯域が不足しているアプリケーションの要求性能を満たすためにはシステム設計の大幅な変更が必要であり，ドメイン特化型のシステムも候補の 1 つと考えられる．

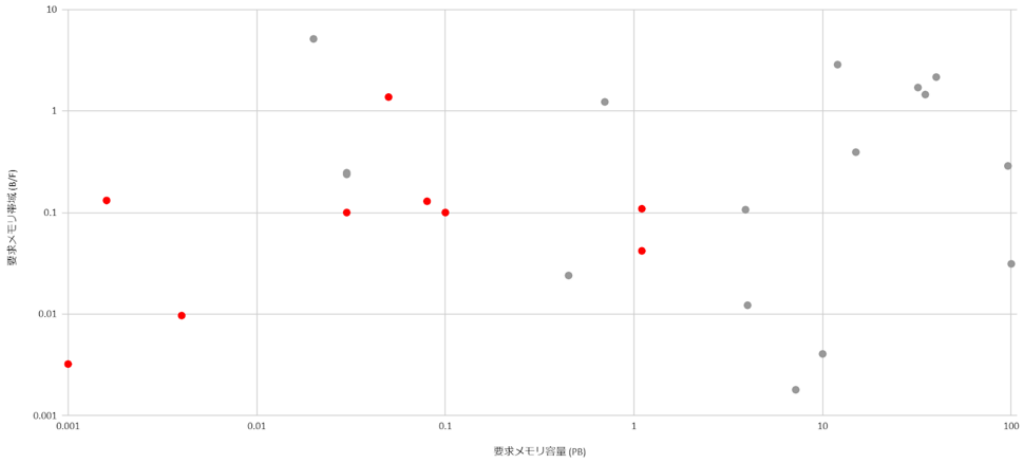

図 3.2　50MW, GPU 電力 80% のシステムがカバーする計算科学ロードマップのアプリケーション（赤：要求性能を満たすアプリ，灰：要求性能を満たさないアプリ）

次に，GPU ベースのシステムについて分析する．システム全体の電力バジェットが 50MW，GPU の電力バジェットが 80% のシステムについて，計算科学ロードマップに記載されたアプリケーションの要求性能と比較する．ただし，計算科学ロードマップには GPU 数に関する記述がないため，本書では「GPU 数＝ノード数」として比較を行う．

縦軸を要求 B/F 比，横軸を要求メモリ容量として各アプリケーションをマップしたグラフを図 3.2 に示す．また，表 3.5 に各アプリケーションの要求性能と想定システムのピーク性能の比較結果を示す．

図 3.2 より，想定システムは要求メモリ容量が 3PB 未満，かつ，要求 B/F 比が 1.4 未満のアプリケーション（計 14 個）の多く（計 10 個）をカバーする．その一方で，3PB 以上のメモリ，あるいは，1.4 以上の B/F 比を必要とするアプリケーション（計 11 個）はカバーしない．表に示すように，従来の汎用システムの延長では 30GB/node 以上のメモリ容量，あるいは，16TB/s/node を超えるメモリ帯域の実現は困難であり，これらのアプリケーションの要求性能を満たすためには，4.2 節の中で述べるようなドメイン特化型のシステムの利用や，演算に必要なデータビット幅を積極的に低ビット化して実効的なバンド幅をあげるなどの方策が必要と考えられる．

次に，要求メモリ容量が 3PB 未満，かつ，要求 B/F 比が 1.4 未満のアプリケーションに着目する．表より，想定システムが要求性能を満たせないアプリケーションは，160,000 以上のノード数（NICAM-LETKF），

表 3.5　各アプリケーションの要求性能と電力 50MW（GPU80%）のシステム性能の比較結果

分野	アプリケーション名	要求ノード数	演算性能 (TFlops /node)	メモリ帯域 (TB/s/ node)	メモリ容量 (GB/ node)	インターコネクト帯域 (Tb/s/ node)	ファイル I/O 帯域 (GB/s/ node)	ストレージ容量 (PB)	ノード	CPU	メモリ	インターコネクト	ストレージ	総合
素粒子・原子核		40,000	7.75	1	2	1.25	0.045	60	○	○	○	○	○	○
素粒子・原子核		4,100	7.56	0.073	0.98	0.088	0.039	5	○	○	○	○	○	○
素粒子・原子核		1,000	5.6	0	1	0	0.1	0.001	○	○	○	○	○	○
素粒子・原子核	rmcsm	6,000	5	0.5	5	0	0.017	0	○	○	○	○	○	○
素粒子・原子核		4,100	4.15	21.22	4.88	53.75	0.006	120	○	○	×	×	○	×
物質科学	HPhi	1,000,000	0.03	0.001	100	1.025	0.009	340	×	○	×	○	○	×
エネルギー・資源	NTChem	18,000	11.11	16.111	1,944.44	0.001	0.002	0.1	○	○	×	○	○	×
エネルギー・資源	SMASH	100,000	10	17	320	0.014	0.003	0.32	○	○	×	○	○	×
エネルギー・資源	paraDMRG	100,000	31	1.3	11	0.05	0.01	0	○	○	○	○	○	○
エネルギー・資源	GELLAN	100,000	2.7	0.033	40	0.018	0	0.02	○	○	×	○	○	×
エネルギー・資源	MODYLAS	100,000	5	0.5	1	0.003	0.001	500	○	○	○	○	○	○
脳科学・AI	WHC	50,000	30	3.2	78	0	0	1	○	○	×	○	○	×
脳科学・AI	Realtime cerebellum	50,000	11.2	1.22	22	0	0	5	○	○	○	○	○	○
脳科学・AI	NEURON K+ Stochastic	100,000	7.6	1	0.02	0	0.13	160	○	○	○	○	○	○
脳科学・AI	CNN (Forward & Back Prop)	16,000	100	23.75	1.88	0.25	0.006	175	○	×	×	○	○	×
脳科学・AI	CNN (Forward)	6,900	100	24.638	4.35	0.425	0.493	290	○	×	×	○	×	×
地震・津波	GAMERA	200,000	2.8	1.1	75	0.025	0.005	1	×	○	×	○	○	×
気象気候	NICAM	400,000	1.23	3.5	30	0	0	230	×	○	×	○	○	×
気象気候	SCALE-RM	40,000	1.15	1.575	1.25	0	0	33	○	○	○	○	○	○
気象気候	CHASER-LETKF	32,000	0.2	0.438	1250	750	2.5	50	○	○	×	×	×	×
気象気候	NICAM-LETKF	160,000	1.38	1.688	4.38	0	0.5	10	×	○	○	○	×	×
宇宙・天文	GreeM	100,000	6.9	0.028	100	25	0.3	300	○	○	×	×	○	×
宇宙・天文		100,000	200	0.36	72	0.125	0.06	2,600.00	○	×	×	○	○	×
宇宙・天文	EM-PIC	100,000	1.6	0.46	960	0.125	10	2,000.00	○	○	×	○	×	×
宇宙・天文	P3T	10,000	3.1	0.01	0.1	0.001	0.001	400	○	○	○	○	○	○
宇宙・天文	AmaTeRAS	1,000,000	1	0.024	0.45	41.25	0	20	×	○	○	×	○	×

100TFLOPS/node 以上の GPU 性能（CNN），23.75TB/s/node 以上のメモリ帯域（CNN），41.25Tb/s/node 以上のインターコネクト帯域（AmaTeRAS），0.493GB/s/node 以上のファイル I/O 帯域（CNN）を要求している．ノード数，あるいは，GPU 性能が要求に満たないアプリケーションについては，より低性能な GPU を用いて大ノード数化する，あるいは，より高性能な GPU を用いて小ノード数化することで要求性能を満たせる可能性があり，最適なノード数については今後検討の余地がある．一方，メモリ，インターコネクト，ファイル I/O の各帯域が不足しているアプリケーションの要求性能を満たすためにはシステム設計の大幅な変更が必要であり，同じくドメイン特化型のシステムも候補の 1 つと考えられる．

第 4 章

次世代 (2028 年頃) システムの候補

次世代先端的計算基盤システムの構築にあたり，その課題や必要となる技術開発を検討するためには，まずそのシステムイメージを掴むことが必要である．そこで，本節では主に 2028 年頃に必要とされるであろう次世代フラグシップ計算基盤を対象として，その構成や性能について予測・検討する．また，ムーアの法則の終焉を間近に控え，一部専用型システムや新計算原理など，より高い性能，高い効率を持つシステムの候補についても言及する．

4.1 汎用システム型

フラッグシップ機を開発する上での制約条件は多数存在するが，本項では最大の制約条件がシステムの電力バジェット（電力消費量）であるとの観点に立ち，前章で述べた各ハードウェアコンポーネントのトレンド（傾向）から 2028 年頃の汎用的なシステムの構成と性能を検討・予測する．予測値の幅は電力バジェットの幅であり，富岳と同程度から 2.2 倍を想定している．予測の対象は汎用 CPU に HBM を追加したメニーコア CPU 型システムと，HPC 向けに HBM のみを搭載したプロセッサが主体となるメニーコア CPU&GPU 混載型システムである．予測結果はメニーコア CPU 型システムでは 815〜1810PFLOPS，メニーコア CPU&GPU 混載型システムでは保守的な見積もりで 888〜1,975PFLOPS，積極的な見積もりで 6,280〜13,956PFLOPS となった．富岳の最大ピーク性能は 537PFLOPS であるので，この予測結果は 2028 年には富岳の 1.5〜3.4 倍の性能を持つ汎用システムを単純に購入することが可能であること，各国が従来通り HPC 専用システムを継続的に開発・導入することで保守的な見積もりで富岳の 1.7〜3.7 倍，積極的な見積もりでは富岳の 12〜26 倍の性能を実現可能なことを表している．富岳と同種，同規模のシステムが 8 年後に 2.4〜12 倍の性能向上を達成すると考えると，年率の性能向上率は 1.1〜1.4 倍であり，これはムーアの法則が有効であった時期の HPC 専用システムの年率性能向上率約 1.9 倍（ムーアの法則自体は年率 1.6 倍）と比較すると大幅に低い．ムーアの法則が減速している技術環境で従来通り HPC アプリケーションを加速し，科学技術の発展に貢献するには，次節で述べる専用システム混載型を含めてより積極的に HPC アプリケーションを加速する新しい設計が求められるであろう．

4.1.1 メニーコア CPU 型

本項では 2028 年頃のメニーコア CPU 型システムの性能を予測する．以下，4.1.1.1 では予測方法を説明し，続く 4.1.1.2 では予測されたシステム性能を述べる．

4.1.1.1 予測方法

2028 年頃のフラッグシップ機の電力バジェットを，これまでに開発されたフラッグシップ機の消費電力をもとに予測する．2011 年に TOP500 で 1 位を獲得した京コンピュータが 12.7MW の電力を消費しており，2020 年 6 月に TOP500 で 1 位を獲得した富岳が 28.3MW の電力を消費していることから，フラッグシップ機の許容電力は 1.73MW/年で増加している．したがって，単純計算では，富岳よりも 13.8MW 多くの電力を消費するシステム（総消費電力 42.1MW のシステム）が 2028 年には許容される可能性がある．ただし，スーパーコンピュータの消費電力は 20-40MW が運用上の限界とも言われており [211, 212]，システムの電力バジェットが上記のように単純にスケールしない可能性もある．そこで，2028 年頃のフラッグシップ機の電力バジェットとして，単純にスケールした場合（40MW）に加え，保守的にスケールした場合（30MW），および，積極的にスケールした場合（50MW）の 3 通りのシナリオを想定する．

2028 年頃のシステムの PUE は 1.1 を想定する．京コンピュータの PUE は 1.3 弱であったが [213]，冷却技術の進歩によりスーパーコンピュータの PUE は減少傾向にあると考えられる．富岳の PUE は本書の執筆時点では未公開であるが，現在運用中のスーパーコンピュータには PUE が 1.1 を下回るシステムもあり [214]，2028 年頃にはフラッグシップ機においても PUE が 1.1 を下回ることが予想される．

CPU，メモリ，インターコネクト，ストレージ等の各ハードウェアコンポーネント（構成要素）の電力バジェットの比率は，現在のシステムのそれと大きく変わらないと予想する．スーパーコンピュータの電力バジェットの内訳はほとんど公表されていないため，データセンターの消費電力の内訳を参考にする．文献 [215] によると，Google のデータセンターは，CPU がシステム全体の 33%，メモリが 30%，インターコネクトが 5%，ストレージが 10%，その他が 22% の電力を消費している．高い計算性能が要求されるスーパーコンピュータではより多くの電力バジェットを CPU 等の演算装置に割り当てていることが予想されるため，CPU の電力バジェットはマシン本体の電力バジェットの 60-80% と想定して見積もりを行う．例えば，システム全体の電力バジェットが 30MW，かつ，CPU の電力バジェットがマシン本体（PUE は 1.1）の電力バジェットの 60% のとき，CPU の電力バジェットは 16.4MW（=30MW × 0.6/1.1）となる．

CPU とメモリの構成は IRDS Systems and Architectures の 2017 年版および 2020 年版の予測 [216, 11] をベースにする．2020 年版 によると，2028 年の CPU のソケットあたりの TDP は 351W である．この値を用いて，上記の CPU の電力バジェットの範囲内で上記のソケットが最大何個システムに搭載できるかを試算する．また，IRDS SA の 2020 年版 ではソケットあたりのメモリ構成も予測されており，本項ではこのメモリ構成を採用してもメモリの電力バジェットを超過しないものと仮定して 2028 年のシステムのメモリ構成を予測する．

インターコネクトに関しては，Ethernet Alliance Roadmap 2018[217] より，2028 年のシステムのリンク 1 本当たりのスループットを 1.6Tb/s と見積もった．これは，2.1.4 項で述べたように，2028 年には次々世代のデータ転送技術である 100Gbps 伝送を採用した（100Gbps×16 レーンの）製品が登場すると予想されるためである．ノードあたりのリンク数は 1 本を仮定する．これは，リンク数を現実的な範囲で増加させても，3.3 節で述べたアプリケーションのカバー範囲が変わらなかったためである．レイテンシに関しては，3.2 節で述べたアプリケーションの中に極端に低いレイテンシを要求するものがなかったため，本項では考慮しない．

ストレージに関しては，Lustre の I/O 性能が 1.36x/年，容量が 1.38x/年で向上すると予想されていることから [218]，2018 年の Summit のストレージ性能（2.5TB/s, 250PB）をもとに 2028 年のシステムの I/O 性能を 34TB/s，ストレージ容量を 3.45EB と見積もった．

表 4.1　次世代のメニーコア CPU 型システムの諸元

	30MW			40MW			50MW		
	60%	70%	80%	60%	70%	80%	60%	70%	80%
ソケット数	46620	54390	62160	62160	72520	82880	77700	90650	103600
総コア数	3.3×10^6	3.8×10^6	4.4×10^6	4.4×10^6	5.1×10^6	5.8×10^6	5.4×10^6	6.3×10^6	7.3×10^6
PFLOPS	815	950	1086	1086	1267	1448	1358	1584	1810
DDR 総 BW (PB/s)	102	120	137	137	160	182	171	200	228
HBM 総 BW (PB/s)	307	358	410	410	478	547	512	598	683
DDR 総容量 (PB)	17	20	23	23	27	31	29	34	39
HBM 総容量 (PB)	4	5	5	5	6	7	7	8	9
インジェクション BW (Tb/s)	1.6	1.6	1.6	1.6	1.6	1.6	1.6	1.6	1.6
総 I/O 性能 (TB/s)	34	34	34	34	34	34	34	34	34
総ストレージ容量	3.45	3.45	3.45	3.45	3.45	3.45	3.45	3.45	3.45

4.1.1.2　システムイメージ

次世代のメニーコア型 CPU システムの諸元を表 4.1 に示す．総理論演算性能は，最も保守的な設計を行った場合（システム全体の電力バジェットが 30MW，CPU の電力バジェットが 60% の場合）に 815PFLOPS（倍精度）に達すると予想される．また，最も積極的な設計を行った場合（システム全体の電力バジェットが 50MW，CPU の電力バジェットが 80% の場合）には総理論演算性能が倍精度で 1.81EFLOPS に達すると予想される．スーパーコンピュータ富岳のピーク性能は 537PFLOPS であるから，本構成の性能は富岳の 1.5-3.4 倍程度と予想される．

2028 年の CPU の諸元を表 4.2 に示す．この表は，IRDS Systems and Architectures の 2017 年版および 2020 年版で予測されている 2028 年の CPU の性能値から主要な項目の値を抜粋して作成したものである．2028 年の CPU は 70 個のコアを搭載し，3.9GHz で動作することが予想される．SIMD 長は 2048 ビット，SIMD ポート数は 2 である．したがって，2028 年の CPU の理論演算性能は倍精度で 17.472TFLOPS である．なお，システム全体の総コア数は 3.3–7.3×10^6 個になると予想される．

メモリに関しては，次世代機では DDR と HBM の両方を搭載し，アプリケーションの要求（容量とバンド幅）に応じて 2 つのメモリを使い分けるシステムを仮定する．なお，IRDS Systems and Architectures 2020 年版には 2028 年の DDR のバンド幅に関する記載がなかったため，2017 年版 に記載の値を使用する．表より，次世代機では，DDR の総メモリ量は 17-39PB，総バンド幅は 102-228PB/s になると予想される．また，HBM の総メモリ量は 4-9PB，総バンド幅は 307-683PB/s になると予想される．

表 4.2　2028 年の CPU の諸元（IRDS Systems and Architectures の 2017 年版および 2020 年版より抜粋）

	CPU
コア数	70
周波数 (GHz)	3.9
SIMD 長 (bit)	2048
SIMD ポート数	2
TFLOPS	17.472

表 4.3　2028 年のメモリの諸元（ソケットあたり，IRDS Systems and Architectures の 2017 年版および 2020 年版より抜粋）

	DDR	HBM
チャネル/モジュール数	12	6
BW (GB/s)	184	1100
総 BW (GB/s)	2208	6600
総容量 (GB)	384	96

DDR と HBM それぞれのソケットあたりの諸元を表 4.3 に示す．この表は，IRDS Systems and Architectures の 2017 年版および 2020 年版で予測されている 2028 年の DDR と HBM の性能値から主要な項目の値を抜粋して作成したものである．DDR のチャネル数は 12，各チャネルのバンド幅は 184GB/s になると予想される．この値は，2.1.3 項で述べた DDR5 のチャネルあたりのバンド幅の 3.6 倍であり，IRDS は 2028 年までに DDR6 または DDR7 が実用化されることを想定しているものと思われる．したがって，DDR の総バンド幅は 2,208GB/s，総容量は 384GB である．一方，HBM のモジュール数は 6，モジュールあたりのバンド幅は 1,100GB/s になると予想される．この値は，2.1.3 項で述べた HBM3 のモジュールあたりのバンド幅の 2.1 倍であり，IRDS は 2028 年までに HBM4 が実用化されることを想定しているものと思われる．したがって，HBM の総バンド幅は 6,600GB/s，総容量は 96GB である．

4.1.2　メニーコア CPU & GPU 混載型

本項では 2028 年頃のメニーコア CPU&GPU 混載型システムの性能を予測する．以下，4.1.2.1 では予測方法を，4.1.2.2 では予測されたシステム構成を述べる．

4.1.2.1　予測方法

2028 年頃のメニーコア CPU&GPU 混載型システムの予測方法は，基本的には 4.1.1.1 と同じである．すなわち，2028 年頃のフラッグシップ機の電力バジェットとして 30, 40, 50MW の 3 通りを想定し，PUE=1.1 の時の GPU の電力バジェットからシステムに搭載可能な GPU の総数を予測する．また，本項で想定する CPU&GPU 混載型システムではアプリケーションはすべて GPU 上で実行されるものとし，CPU と GPU のハイブリッド実行は行わないものとする．このようなシステムでは CPU の消費電力が GPU の消費電力に比べて各段に小さくなると考えられることから，CPU の電力バジェットは無視できるものとし，GPU の電力バジェットのみを考慮する．すなわち，GPU の電力バジェットとして，マシン本体の電力バジェットの 60, 70, 80% の 3 種類を想定する．

2028 年の GPU 性能は，NVIDIA 社の過去 13 年間のハイエンド GPU の性能をもとに線形で外挿する．具体的には，M1060（2008 年），M2090（2011 年），Tesla K20X（2012 年），Tesla K40（2013 年），PCIe 版の Tesla P100（2016 年），Tesla V100（2017 年），Tesla A100（2020 年）の 7 つの GPU のコア数や周波数などの諸元から，2028 年の GPU の諸元を予測する．その結果，4.1.2.2 で述べるように，2028 年の GPU の理論演算性能として 24.338TFLOPS（倍精度）という値が得られた．ただし，上記の値はチップあたりの性能で

表 4.4　次世代のメニーコア CPU&GPU 混載型システムの諸元（保守的な見積もりを行った場合）

	30MW			40MW			50MW		
	60%	70%	80%	60%	70%	80%	60%	70%	80%
GPU 数	36525	42613	48701	48701	56818	64935	60876	71022	81168
総コア数	4.1×10^8	4.7×10^8	5.4×10^8	5.4×10^8	6.3×10^8	7.2×10^8	6.8×10^8	7.9×10^8	9.0×10^8
PFLOPS	888	1037	1185	1185	1382	1580	1481	1728	1975
HBM 総 BW (PB/s)	85	100	114	114	133	152	143	167	191
HBM 総容量 (PB)	1	2	2	2	2	3	3	3	4

表 4.5　次世代のメニーコア CPU&GPU 混載型システムの諸元（積極的な見積もりを行った場合）

	30MW			40MW			50MW		
	60%	70%	80%	60%	70%	80%	60%	70%	80%
GPU 数	36525	42613	48701	48701	56818	64935	60876	71022	81168
総コア数	2.9×10^9	3.4×10^9	3.8×10^9	3.8×10^9	4.5×10^9	5.1×10^9	4.8×10^9	5.6×10^9	6.4×10^9
PFLOPS	6280	7327	8373	8373	9769	11165	10467	12212	13956
HBM 総 BW (PB/s)	241	281	321	321	374	428	401	468	535
HBM 総容量 (PB)	3	4	4	4	5	6	5	6	7

比較すると，IRDS Systems and Architectures の 2020 年版における CPU の性能予測値（17.472TFLOPS）の 1.39 倍に過ぎず，現行の CPU と GPU の性能差（8.2-10.3 倍）を考慮するとかなり保守的な見積もり結果と言える．そこで，より積極的な見積もりとして，4.1.1 項で示した 2028 年の CPU の性能予測値に現行の CPU と GPU の性能差を乗じることにより，2028 年の GPU 性能の予測も行う．

なお，2028 年頃のメモリ，インターコネクト，ストレージに関する各仮定は 4.1.2.1 と同様とする．

4.1.2.2　システムイメージ

次世代のメニーコア CPU&GPU 混載型システムの諸元を表 4.4 と表 4.5 に示す．総理論演算性能は，保守的な見積もり（表 4.4）のもとでは，最も保守的な構成を仮定した場合（システム全体の電力バジェットが 30MW，GPU の電力バジェットが 60% の場合）に 888PFLOPS，最も積極的な構成を仮定した場合（システム全体の電力バジェットが 50MW，CPU の電力バジェットが 80% の場合）には 1.98EFLOPS に達すると予想される．これらの性能はスーパーコンピュータ富岳の 1.7-3.7 倍程度に相当する．一方，積極的な見積もり（表 4.5）のもとでは，最も保守的な構成を仮定した場合に 6.28EFLOPS（富岳の約 11.7 倍），最も積極的な構成を仮定した場合には 14.0EFLOPS（富岳の約 26.0 倍）に達すると予想される．

GPU の諸元を表 4.6 に示す．2028 年の GPU は，11,131 個または 78,658 個のコアを搭載し，2.186GHz で動作することが予想される．理論演算性能は保守的または積極的な見積もりに応じて 24.338TFLOPS または 171.95TFLOPS（いずれも倍精度），TDP は 448W である．なお，システム全体の総コア数は 407-6,385M 個になると予想される．また，メモリに関しては，HBM のバンド幅が 2,354GB/s または 6,600GB/s，容量

表 4.6　GPU の諸元

	Conservative GPU	Aggressive GPU
コア数	11131	78658
周波数 (GHz)	2.186	2.186
TFLOPS	24.338	171.95
HBM 総 BW (GB/s)	2354	6600
HBM 総容量 (GB)	52	96
TDB (W)	448	448

が 52GB または 96GB になると予想される．

4.1.3　その他（ベクトルプロセッサ）

本項では 2028 年頃にベクトルプロセッサを主たる演算構成要素として採用した場合のシステム構成について述べる．

4.1.3.1　予測方法

現時点でいわゆるベクトルプロセッサを商用化しているのは NEC の SX-Aurora TSUBASA シリーズのみである．もちろん近年のプロセッサはベクトル拡張命令を備えることも多く，そのベクトル長も Intel Xeon（AVX512）や富岳の A64fx（arm SVE）では 512bit まで拡張されているが，SX-Aurora TSUBASA のベクトル長は 16Kbit（=64bit×256word）であり，大きな開きがある．このため本項の議論においては，ベクトルプロセッサとして SX-Aurora TSUBASA シリーズの延長線上にあるシステムについて予測し，仮にこれをフラッグシップ機開発の制約条件に合わせた場合のシステム構成を予測する．

4.1.3.2　システムイメージ

まず Vector Engine の性能予測を行う．SX-Aurora TSUBASA は，2.2.1.3 に示したように PCI express カードタイプの Vector Engine と，標準的な x86 サーバである Vector Host によって構成される．ここでは 2028 年頃に予想できる Vector Engine の仕様について述べる．Vector Engine の代表的な仕様 (再掲) と 2028 年の予想仕様を表に示す．ベクトルプロセッサではメモリバンド幅が非常に重要であることから，メモリバンド幅では最新のテクノロジを活用し，B/F=0.5 以上を維持する範囲で演算性能を向上させると予想した．メニーコア型 CPU と比較すると，プロセッサの電力バジェットに対してメモリ性能強化に力点を置き，演算性能については半分以下という予測値になっている．現行のメニーコア型 CPU とベクトルプロセッサの演算性能差（2.433/2.768）からするとこの見積もり値は極めて保守的であり，実際には演算性能をより向上させる可能性もある．

次に，システム全体のイメージについて述べる．前項までの議論と同様，システム開発時の最大の制約条件は電力バジェットであると仮定する．SX-Aurora TSUBASA と同様のアーキテクチャ，すなわち PCI express カードタイプの Vector Engine と標準的な x86 サーバをホストに持つアーキテクチャで後継機の開発が進むと仮定すると，構成上はメニーコア CPU&GPU 混載型システムに非常に近いものになると推測できる．ここでは 4.1.2 項の議論と同様に，アプリケーションは主として Vector Engine 上で実行され，CPU と

のハイブリッド実行は行わないとし，CPU の消費電力が Vector Engine の消費電力と比較して十分に小さいと考え，Vector Engine の電力バジェットのみを考慮することにする．2028 年頃にフラッグシップ機の電力バジェットとしては 30,40,50MW の 3 通りを想定し，Vector Engine の電力バジェットとしてマシン本体の電力バジェットの 60,70,80% の 3 種類を想定する．

4.2 専用システム混載型および新たなる可能性

前節では，メニーコア CPU 型や GPU 混載型等の従来のアーキテクチャに基づくシステム構成では，ムーアの法則の減速により富岳と同規模のシステムの場合には 2028 年頃に富岳の 2.4〜15 倍の性能向上しか達成できないことが予測された．またコストや電力には制約があるので，システムの規模は 10 どころか 2 倍に拡大するのも困難である．その一方でアプリケーションが要求する性能はますます高くなっている．この問題を解決するためには，従来の汎用システム型とは異なる「専用指向の方式」を導入する新しいアプローチを検討する必要である．従来の汎用システム型では基本的に HPC アプリケーション全般の高性能を目標としていた．これはサーバ向け CPU やグラフィックス向け GPU をそのまま転用する場合でも，HPC 向けにカスタマイズされた CPU や GPU を開発する場合でも同様である．これに対し「専用指向の方式」では HPC アプリケーションの特定のドメインにおいて，汎用システム型に対してコスト性能，電力性能またはその両方での大幅な優位を得ること目標とする．「専用」には専用計算機，専用演算器のように特定の処理しか実行できない狭義の「専用」から，様々な処理を実行できるシステムだが特定のドメインでは性能優位性を発揮する広義の「専用」まであるが，ここでは基本的にシステムとして広義の「専用」の実現を目指す．

専用指向の方式を取り入れた「専用システム混載型」のアーキテクチャにはまだ主流と言える構成方法はなく，汎用システム型に選定したアプリケーション向けの専用指向の方式を取り入れる構成，HPC アプリケーションを複数のドメインに分類してそれぞれに向けた専用指向の方式を組み合わせる構成など幅広く検討評価する必要がある．また，各アプリケーションは様々な特性の処理の組み合わせで構成されているので，複数種類の専用指向の方式を組み合わせて利用することでより高い性能が得られる可能性がある．さらに，専用指向の方式は本質的に利用アプリケーションが限定されるため汎用システム型より稼働率が低い問題も解決しなければならない．これらの課題を解決する技術の候補としては近年のデータセンターにおける分散コンピューティング分野で議論されている Composable/Disaggrigated Infrastructure があるが，共通の要素技術を 2028 年頃に構築する並列コンピューティングシステムに展開可能か，特にハードウェア技術が十分成熟するか見極める必要がある．また，極めて高速な共有ストレージを有しワークフローを活用するシステムなど他の候補も考えられる．候補となる技術について実現性，既存プログラミングモデルとの親和性を含めた比較評価が必要である．

本節では，「広義の専用」に基づくシステム拡張やその他の可能性として，CPU 拡張型，アクセラレータ主体型，Processing-In-Memory 等によるメモリと CPU の混載型，新計算原理マシンについてその特徴と可能性，および技術的課題を述べる．最後に，計算科学ロードマップのアプリケーションが要求する性能に対し，専用システム混載型のアーキテクチャがもたらす恩恵の試算を示す．

4.2.1 CPU 拡張型

一般用途でも使用頻度の高いアプリケーションでは，専用機能が汎用 CPU に拡張機能として搭載される場合がある．例として各種 CPU の AI 向けの命令セット拡張を紹介する．x86 命令セットでは Intel Advanced

Matrix Extension (AMX) が拡張された．AMX は 8,192 ビットの行列レジスタを 8 本搭載拡張し，8 ビット整数および 16 ビット浮動小数点 BF16 の行列積演算を行う．Arm A64 Armv8 命令セットでは SVE に FMMLA，NEON に BFMMLA 命令が拡張された．FMMLA 命令は FP32 および FP64 の 2×2 行列積和演算，BFMMLA 命令は BF16 の 2×4 行列と 4×2 行列の積和演算を行う．POWER 命令セットでは行列積演算を補助する VSX Matrix-Multiply Assist (MMA) 命令群が追加された．MMA は VSX レジスタに格納された行ベクトル，列ベクトルの内積を結果行列に加算する命令群で，浮動小数点では倍精度，単精度，半精度，BF16，整数では 16 ビット，8 ビット，4 ビットと様々な数値データ型に対応する．以上に挙げたような AI 向け拡張機能は今後の HPC アプリケーションにおいて，混合精度演算への活用や，AI 向けアルゴリズムの応用による活用が期待される．

ここで，CPU 拡張型の専用機能については将来性に疑問がある点に留意したい．ムーア則が健在でデナード則が破綻していた時期では，利用可能なトランジスタ数は増えるが電力密度の問題でチップ上の全てのトランジスタを動作させることができないため，CPU に利用頻度の低い専用機能を追加することは理に適っていた．しかし現在ではコストが原因でムーア則も破綻しているため，利用頻度の低い機能を最先端プロセスで製造する CPU に搭載することはコスト面のデメリットが大きい．今後 CPU に拡張される専用機能は HPC に限らない一般用途も含めて特に利用頻度の高いものに限定されると考えられる．

4.2.2 アクセラレータ主体型 / ヘテロジニアス型

4.1 節で述べた通り，従来のアーキテクチャの延長線上である汎用システムでは，アプリケーション毎に求められる要求性能や電力性能比を達成できない可能性がある．また，4.2.1 節の CPU 拡張型でも，性能を十分に向上できない可能性がある．この問題を解決する一つの方法として，個々の計算問題やドメインに対して設計された専用または準専用の様々なハードウェアアクセラレータを汎用システムに搭載し，演算素子・記憶素子・通信路といったハードウェア資源やそれらを駆動するための電力などを効率良く使用して対象問題の性能を向上させるアクセラレータ主体型/ヘテロジニアス型の方式が考えられる．一方で，この方法ではアクセラレータの対象となる問題のみが恩恵を受けることから，性能向上やコストなどのトレードオフを考慮する必要がある．また，完全に専用ではなくプログラミング可能性を持つ準専用のアクセラレータの場合には，そのプログラミング生産性が問題となる．以下，本節では，アクセラレータのシステム搭載方式，およびアクセラレータの構成方式についてその可能性を挙げ，将来予測と課題を述べる．

4.2.2.1 アクセラレータのシステム搭載方式

ここで検討する独立して動作可能な専用アクセラレータには，そのシステムスへの搭載方法として以下のようなものが考えられる．

- **構成 1: チップ内拡張型（CPU・アクセラレータのマルチチップパッケージ構成）**
  チップ内において，チップレットとして実装された専用アクセラレータを，インターポーザまたは 3 次元実装によりパッケージ内で密結合する構成．パッケージ内に複数種類のアクセラレータを混載する場合も考えられる．外部メモリを CPU やアクセラレータ間で共有する構成や個別の外部メモリを持たせる構成などが考えられるが，パッケージ内で太く低遅延の帯域で結合できることから，CPU とアクセラレータ間においてメモリ空間を共有する方式が，プログラミング生産性の観点では有望である．
- **構成 2：ノード内拡張型（CPU・アクセラレータの密結合構成）**

ノード内において，専用アクセラレータのチップまたはパッケージを，PCI-Express, CXL, CAPI 等のインターコネクトで CPU と密結合する構成．例えば，アクセラレータチップを搭載した PCI-Express カードをノードに搭載するような方式が考えられる．この場合，CPU やローカルストレージ，NIC，アクセラレータを相互に結合するノード内インターコネクト（PCI-Express ブリッジなど）の構成が重要であり，誰がアクセラレータのメモリ空間を参照可能か，あるいはアクセラレータが他のどのコンポーネントを直接参照可能か，等のノード内接続方式にも検討の余地がある．

- **構成 3：問題特化ノードやラックの疎結合構成**
  システム内において，問題特化型の計算ノード，あるいはラックを何種類か設け，それをシステムネットワークで結合する構成．後者の場合には，いわゆるラックスケールコンピューティングに相当する．問題特化型のノードやラックはそれら単体でも動作するようなものを想定する．疎結合でありノード内と比べて狭い帯域，大きな遅延での接続となることから，大きな粒度でのタスクを高速化するような方法等，利用形態を工夫する必要がある．

4.2.2.2 アクセラレータの構成方式

対象とする計算アルゴリズムドメインの広さやプログラム可能性により，アクセラレータの構成には，専用，準専用，汎用，の 3 種類が考えられる．

専用アクセラレータは，その名の通り，特定の計算問題またはドメインのみを高速に処理可能な構成である．特定の用途を対象とすることでその分設計や実装を最適化でき，汎用プロセッサや GPU のような汎用のアクセラレータと比べて低消費電力で高い演算性能を達成する．得られる演算性能の犠牲になるものは柔軟性であり，これは設計コストの増加とプログラマビリティの低下を引き起こす．システムに期待される複数の問題に対応するには，それぞれの専用アクセラレータを混載し集合させることが必要となる．しかしながら，ある専用アクセラレータは他の処理に使用されないという状況が生じ得るため，ハードウェア資源の利用効率という観点でその是非を検討する必要がある．このような専用アクセラレータの例としては，行列演算アクセラレータ，FFT アクセラレータ，等が考えられる．また，2020 年において，問題特化型の演算装置の典型的な対象はディープニューラルネットワークの学習・推論処理である．パイプライン並列性やオンチップメモリの活用が高い演算性能の実現に寄与している．今後も，特殊な演算処理，または特有の並列性やデータの再利用可能性を有する問題に対しては，それをハードウェアにより実現する専用アクセラレータが有望であると考えられる．

準専用アクセラレータは，例えばデータフローといった特定の計算モデルや計算問題ドメインに対して，ある程度のプログラム可能性を与えた構成である．専用構成と比べて汎用性の高まる分処理効率を犠牲にすることとはなるが，単体のアクセラレータでも専用と比べてより広い範囲を扱えるため，ハードウェア資源の利用効率は専用構成よりも改善される．性能向上と効率のバランスを考慮した，適切なアーキテクチャ設計が求められる．例としては，データフロー（ウェーブフロント）プロセッサ，コンボリューションプロセッサ，等が考えられる．

汎用アクセラレータは，言葉としては矛盾しているように聞こえるが，超汎用な FPGA 等の回路再構成可能デバイス上に専用や準専用のアクセラレータを実装するような，アクセラレータ実装のフィールドプログラマブルな汎用プラットフォームのことである．ここで，フィールドプログラマブルとは，システムに組み込んだままで構成を変更可能であることを意味する．回路再構成可能デバイス FPGA そのものにはそれを実現するためのオーバーヘッドがあるものの，構成できる回路の自由度により，汎用アーキテクチャの苦手とするよ

うな処理性能向上の可能性がある．また，粗粒度再構成可能アーキテクチャ (CGRA) のように，対象とするアクセラレータ群に対して，適切な演算粒度やメモリシステムを持つ回路再構成可能デバイスのアーキテクチャ開発も考えられる．

4.2.2.3 専用・準専用アクセラレータの候補例

ここでは，専用・準専用アクセラレータの候補として考えられる計算問題や計算ドメインの例について，性能要件とその理由，および専用システムとして有望な実装形態や現状の動向，課題についてまとめる．

- N 体問題：
  - **概要**: ある計算領域に分布された多数の粒子の相互作用から系全体のシミュレーションを行う手法．例えば，N 体問題や古典分子動力学法 (MD)，個別要素法 (DEM)，SPH 法，MPS 法などがあげられる．計算領域を格子で分割する FEM や FDM などの格子法とは異なる手法.
  - **ニーズ**: MD は創薬，DEM や SPH，MPS は土木や航空分野に利用されており，より大きな系を解いたり，一定の系をより高速に解く様な，社会的ニーズが高い.
  - **性能への影響**: 粒子 1 つの計算量は格子点 1 つの計算量に比べ非常に大きいため，データだけでなく命令の量が多い点が特徴として挙げられる．また，粒子が移動するため，ランダムアクセスやノード間の通信データ量が異なる点がボトルネックとなる．さらに，各計算手法にはそれぞれのボトルネックが存在する．例えば，MD では FFT, N 体では相互作用計算，DEM などでは動的負荷分散がそれに当たる.
  - **克服方法**: ボトルネックとなるアルゴリズムに特化した解決策の提案．例えば，MD では FFT 以外のアルゴリズムの採用，N 体では相互作用計算のハードウェア実装，DEM などでは動的負荷分散の改良などが考えられる.
  - **現状**: MD では MDGRAPE-4A，N 体問題では Anton，Anton2 などの専用計算機が開発されている．ソフトウェアでは，MD では GROMACS や GENESIS，N 体問題では FDPS が大規模シミュレーションを目的として開発されている．また，MPS や DEM では商用のアプリケーションも存在する.
  - **課題**: ソフトウェア，ネットワーク，アーキテクチャなどの検討.
- FFT：
  - **概要**: 時間領域の信号から周波数領域の信号への変換する解析の手法.
  - **ニーズ**: 数値流体力学などの数値解析の分野ではスペクトル解析のための手法として用いられている．また，畳み込み演算の高速化の手法として多くの分野で利用されている．例えば，古典分子動力学計算の分野で用いられている．また，FFT を行った後に，逆 FFT を行うアプリケーションも多い.
  - **性能への影響**: 広い分野で利用されているが，大規模な点数の FFT はそれ自体がアプリケーション全体のボトルネックとなってしまう．これは，B/F 比が大きいことが 1 つの要因であり，単体ノードではメモリバンド幅ボトルネックとなり，複数ノードでは通信のレイテンシがボトルネックとなる．なお，計算の特性が十分に知られており，理論性能なども容易に導出が可能である.
  - **克服方法**: 計算と通信のオーバーラップやメモリ帯域の向上が必須と考えられる．計算の特性が十分に知られているため，データフローなどの考え方を元にした実装も有効であると考える.
  - **現状**: 大規模な FFT は，専用の高速・低レイテンシ密結合網を用いたシステムにより，高速化が

進んでいる．例えば，Tofu-D を搭載した理研/富士通のスーパーコンピューター 富岳や，NVLink を搭載した NVIDIA 社の GPU クラスターなどが挙げられる．ソフトウェアの最適化は，システムの性能を有効に引き出すアルゴリズム実装が盛んである．

  - **課題**: アーキテクチャ，通信性能，メモリ性能の一体化研究が必要．

- グラフ処理：
  - **概要**: 複数の実体が存在し，個々の実体間の関係性を表現する数学モデル，およびその関係性を解析する情報処理手法．
  - **ニーズ**: グラフ構造で表現された，実体間の関係性を評価・最適化できる点．エッジコンピューティングやソーシャルメディアなどの発展に伴い，実体間の繋がりや関係性が重要視されつつある．
  - **性能への影響**: 実用アプリケーションが必要とするノード数が非常に大きく，メモリに収まらない．キャッシュの利用効率が悪い．メモリアクセスの局所性が低い．大規模並列化を行うと，グラフ処理問題をどのように分割して CPU に割り当てるか，という問題が困難になる．
  - **克服方法**: メモリアクセスフレンドリになるよう，デバイス・回路・アーキテクチャ・ソフトウェア・アルゴリズム設計が有望．
  - **現状**: DARPA HIVE プロジェクトのような，アーキテクチャレベル (ハードウェアアクセラレータ) の方向性，GraphX などのソフトウェアレベルの方向性も有望．GraphPIM などもある．
  - **課題**: ソフトウェア，アーキテクチャ，回路，デバイスの一体化研究が必要．
- ニューロモーフィック計算：
  - **概要**: 既存の DeepLearning よりも，より忠実に脳の神経細胞網の挙動を模擬した脳型計算の方法．Spiking ニューラルネットワーク (SpikingNN) などの脳神経細胞網シミュレーションや，それに準じた情報処理．
  - **ニーズ**: 脳ならではの学習・情報処理が可能となる点，または低消費電力で機械学習のような処理が実現できる点で，期待されている．
  - **性能への影響**: ニューロン間の情報伝達を Spike により行うために，汎用計算機では必ずしも効率良く実現できない．また，ニューロン間の結合もその数やトポロジが複雑でかつ変化し得るため，ネットワークがボトルネックとなり得る．ニューロンの動作（発火など）も，汎用 CPU では複雑な関数を計算する必要があり，高速化の余地がある．
  - **克服方法**: Spiking の処理や伝達に適したアーキテクチャ，ネットワーク，回路，デバイスによる専用システム化が有望．
  - **現状**: 大規模並列機を用いた脳シミュレーションが行われている．IBM Synopse, Intel Loihi, Spinnaker, その他研究機関等において，各種専用チップが開発されている．
  - **課題**: アーキテクチャ，システム，回路，デバイスに研究開発が必要．また，モデル化やプログラミングも必要であるが，そのような研究を行っている国際的な研究コミュニティとの連携が有望．
- 量子回路シミュレーション：
  - **概要**: NISQ やエラー耐性量子コンピュータといった，量子ゲート型量子計算機の量子回路をシミュレーションする計算問題．量子ビットの重ね合わせ状態を模擬する．
  - **ニーズ**: 実用的な量子コンピュータのシミュレーションには，量子ビット数を増やしたり，エラー耐性アルゴリズムを適用する必要があり，量子ビットの状態に対して指数関数的に演算数とメモリサイズが増大する．

- **性能への影響**: 重ね合わせ原理のために，量子ビット数 N に対して，$O(2^n)$ の演算数とメモリサイズが必要となる．特にメモリサイズの要求は膨大で，たとえば 100 量子ビットの場合には 約 1.27×10^{30} バイトものメモリ容量が必要となる他，バイト当たりの演算数が大きくないため，メモリ容量・および帯域が性能を制限することとなる．データ移動と容量ネックの問題がある．
- **克服方法**: 演算部は汎用システムでも良い可能性があるが，主記憶容量をかなり大きくする必要がある．また，大規模並列化をする場合には，ノード間の通信帯域もデータサイズに見合ったものに高める必要がある．
- **現状**: 量子回路シミュレーションは，大規模並列計算機や GPU での最適化実装が試みられている．FPGA 等による専用ハードの取り組みもある．
- **課題**: メモリの容量と帯域のトレードオフをどれだけ改善できるか．またノード間帯域や遅延・通信について，シミュレーションアルゴリズムとのコデザインが必要．

- エージェントシミュレーション：
  - **概要**: 複雑系は多くの学術分野で創発的な現象の模擬に活用されているが，細粒度並列性，動的なロードバランス変化などがあり，既存の SIMD のようなアーキテクチャの恩恵を受けにくい．
  - **ニーズ**: 経済，社会，交通流，市場動向，人流，神経系，ネットワークなど，自然科学，社会科学，情報科学など多く学術分野で研究・応用されている．
  - **性能への影響**: 動的にロードバランスが変化する点，並列性は十分にあるが，各エージェントの計算負荷は個々の状況により大きく変わるため，ロードバランスをとりにくい点，エージェント間の通信パターンも動的に変化し得る点，階層の深い if 文が主体で SIMD が効きにくい点などがあげられる．
  - **克服方法**: 負荷分散を支援するハードウェア機構の導入で，負荷分散の問題はある程度緩和できる可能性がある．エージェント処理をハードウェアに展開し，かつオンチップメモリを使用した，高スループットパイプライン処理が考えられる．
  - **現状**: 高生産性スクリプト言語による記述が多いが，性能最適化が効きにくい．処理をある程度パターン化した DSL などがある．
  - **課題**: 細粒度スレッド多並列性能の向上，負荷の重い処理（ソートなど）のハードウェア処理と既存システムとの連携，高性能な DSL などが必要である．

4.2.3 Processing-In-memory 主体型

4.2.3.1 概要

2.1.6.1 で概観した通り PIM(Procesdsing in Memory) の歴史は古いが，スーパーコンピュータへの利用の観点からは新たなる可能性を持つ技術である．特に，ビッグデータが価値を生む AI 全盛の時代，ムーア則終焉から CMOS 代替技術成熟までを支える加速・節電原理として重要性が増しており，図 4.1 に示すように広範な応用における高い加速率が予想されている．ゆえに 2028 年のスーパーコンピュータにおける国際競争力を確保するには PIM は軽視できないと考えられる．

2028 年のスーパーコンピュータに向けた PIM の技術開発では (1) 利用者のニーズを踏まえた活用分野の適切な選択，(2) 低電圧な最新デバイス利用した場合の構成や利得の吟味，(3) 円滑なソフトウェア資産移行や性能最適化を可能にするアーキテクチャや利用技術の研究開発が重要である．

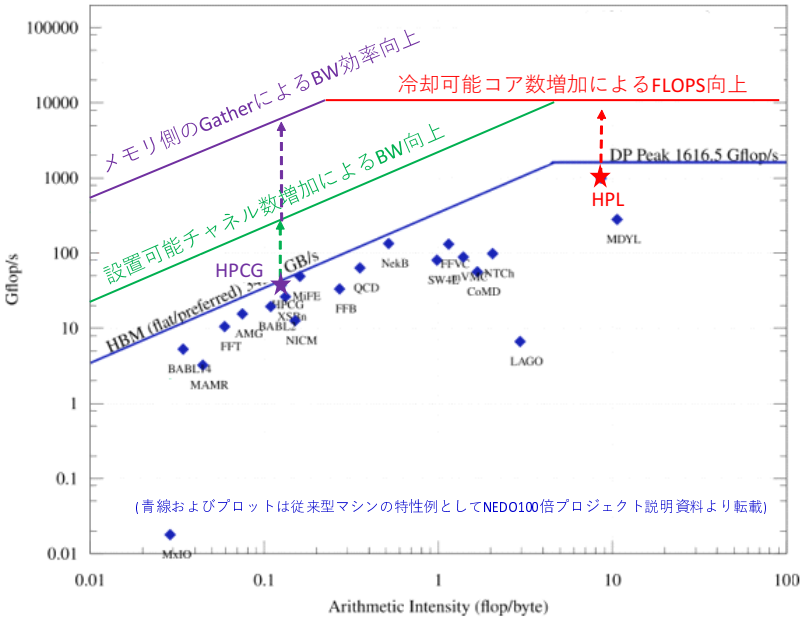

図 4.1　Roofline モデル上での PIM 導入による効果

具体的には，PIM の最も有望な活用分野はメモリアクセスの加速であり，HPCG を典型例とする実効バンド幅により性能が決まることが多い広範な HPC 用途やグラフ解析等において，極めて有望である．一方，低電圧・低周波・簡素・高並列なコアが高バンド幅メモリと近密実装されることにより，電力当たりのメモリバンド幅だけでなく，電力当たりの FLOPS も大幅に向上するゆえ，HPL を典型例とする演算能力により性能が決まる HPC 用途においても有望である．ソフトウェア資産移行性の観点からはホスト CPU と共通 ISA かつ ccNUMA という選択肢の実用化が望ましい．その選択肢を取った場合，ホスト CPU と PIM の特性を適材適所で使い分ける最適化技術の開発が鍵となる．

4.2.3.2　応用分野と PIM を使用した想定システム

■メモリ参照（連続参照，間接参照，等間隔参照，圧縮・解凍）の高速化

富岳ユーザへのアンケート結果から，メモリバンド幅強化に対する強い要望，疎行列を代表とする間接参照などの不連続アクセスバンド幅に対する強い要望が多く寄せられている．メモリ参照加速用 PIM によるこれらのニーズへの想定される貢献は以下の通りである．

1. 演算器とメモリの密接実装によるローカルアクセス時のメモリバンド幅の向上および遅延の短縮
2. HBM 等の多ピンを消費する高バンド幅メモリの分散配置によるピンネック回避とシステム全体の総バンド幅の向上 (例：6HBM/ノード→ PIM12 個× 2HBM/PIM = 24HBM/ノードの追加)
3. Gather/Scatter 機能のメモリ側での実行による不連続アクセスにおけるライン内の有効データ率向上と実効メモリバンド幅の向上 (例：128 バイト/ライン内 8 バイト有効→ 16 倍)
4. 圧縮・解凍による計算時間とバンド幅のトレードオフによるメモリバンド幅の向上

■検索

科学技術計算においてボトルネックになるものが存在するか微妙ではあるが，ビッグデータ解析アプリケーションにおいて検索は基本的な処理であるとともに，データがあるところでの処理が有効になる典型的な PIM 向けアプリケーションである．検索対象とするデータサイズで 3 つのアプローチが存在し，PIM とカテゴライズされるのは 1 と 2 である．

1. 少量なデータ向け：比較回路とデータサイズが同量〜100 倍程度な場合
   CAM の類を用いて比較を行うアプローチがある．通信経路や通信すべき相手を極めて短時間に求めたいようなニーズに対応できる．容量が限られるとともに，エネルギー消費に問題が出やすい．
2. 中〜大規模なデータ向け：データサイズが主記憶に入りきる場合
   主記憶が 3D Xpoint の商用化などで安価に大容量化が可能になってきた今日，最も現実的な状況設定である．メモリ側に配置された比較回路や CPU コアはそれらが紐づけられたローカルメモリアクセスによって処理が完結するため高速化・低電力化する．比較回路をメモリ側に置かない場合でもレコードが行方向に記憶される一般的なインメモリデータベースでは，等間隔アクセスを PIM 側で加速することでフィルタリングが行われ，キャッシュラインの利用率が向上し同様の効果が得られる．
3. 超大規模なデータ向け：データサイズが主記憶に入りきらない場合
   Flash 等のストレージにデータがあり，ストレージ機器に配置された CPU や FPGA やコントローラ上でデータのフィルタリングを行うアプローチ (PIM というよりはインストレージ処理) がある．

■より汎用なロジックや数値計算

PIM には汎用の CPU コアとメモリを緊密に実装した特定の応用と紐づいていない汎用タイプのものが存在する．それらは主に搭載メモリ容量や電力制約などから制約される CPU コア性能の限界性などによって応用範囲に向き不向きができる．近年では HBM2 や HMC を代表とする 3D 積層 DRAM によって PIM のメモリ容量の限界性は緩和してきている．2.1.6.1 で紹介した最新の汎用 PIM である McIM を例に考えると，DIMM スロットの電力制約や冷却能力の限界性に伴い，メモリ側に配置される CPU コアは OoO 機構が省略されたり，周波数を抑制される．一方，緩いピン数制約の下で HBM2 などのメモリと緊密に実装されることで高バンド幅と高い BF 比を実現できる．メインの CPU よりソケット数を多く取れるので全体として多くの CPU コアを利用できる．これらを総合すると，McIM などの現代的汎用 PIM 向けのアプリケーションの大まかな特徴が導き出される．その特徴は以下の通り．

1. 条件分岐の観点からはループ構造のように単純 (OoO が無い弊害が少ない)
2. 間接参照などメモリバンド幅で性能が決まる (高バンド幅と高い BF 比を活かせる)
3. アクセスのローカリティが高い (多くのコアが効率的に並列動作する)

具体的には 1 および 2 が成り立つのがルーフラインモデルにおける左側斜線付近のアプリケーション (例えば HPCG など) であり，1 および 3 が成り立つのがルーフラインモデルにおける右上平端部付近のアプリケーション (例えば HPL など) である．

■想定するアプリケーション例や専用システム

PIM が適すると考えられるアプリケーションは大別して HPL 型と HPCG 型の 2 パターンがある．アプリケーション上のクリティカルパスの条件分岐コストがボトルネックではないことが必要だが，ユーザは自分の

アプリケーションがどちらかに近い場合は PIM による加速を期待できる．

概要で示した図 4.1 は，汎用 CPU ノード単体での各種 HPC 系アプリケーションのルーフラインモデルに即したプロットを含んでいる．このベンチマークセットでは演算がネックとなる HPL 型は HPL と MODYL の 2 種に留まることから，HPC 系では比較的少数派であることが判っている．非 HPC 系では深層学習の学習過程はこれに近い．これらはムーア則の終焉の影響を直接受けるが，PIM により電力当たりの FLOPS が向上するため，悪影響をある程度緩和できる．

残りの HPC 系アプリケーションは殆どが HPCG と類似しており，メモリバンド幅がネックとなっている．ホットスポットに間接参照を含まないアプリケーションの場合は図の緑色のライン下側になり，HPCG のようにホットスポットに間接参照を含むアプリケーションの場合は図の紫色のライン下側付近まで加速されることが期待できる．非 HPC 系では ReLU 型ニューロンを用いる場合などの枝刈された深層学習の学習過程と推論過程やグラフ解析は本質的に疎行列処理となるので，FLOPS では表現できない場合はあるもののメモリアクセス特性は HPCG 系に属する．

上記の議論はノード単体性能に限るので，PIM が適するかどうかは通信ネックか否かとも関連する．一方，PIM の導入によりキャッシュを含む高バンド幅でアクセスできるメモリ容量が増加することから，アプリケーションのノード当たりのフットプリントを大幅に向上できる．このため，ノード当たり主記憶容量やキャッシュ容量が少ないことが原因の通信は排除され，多くの通信 (実態はノード内遠隔メモリアクセス) はノード間通信路より低オーバーヘッドで高バンド幅な経路で行われ，同じ性能を得るための使用ノード数を削減できる．その結果，ノード間通信頻度も減るため，通信がネックになるケースの減少も期待できる．

PIM による通信そのものを加速する可能性も DIMMnet-1 や DIMMnet-2 の研究で探求されており，特に MPI の派生データ型通信などの不連続通信を扱うライブラリを用いる HALO 通信などが計算処理で隠蔽できていないアプリケーションは，PIM による通信の加速による恩恵が期待できる．

上記は概ね McIM などの汎用的な PIM が想定する加速可能なアプリケーションであるが，その HPC における汎用性からこれが設置される場合は，アプリ毎にコデザインされた専用マシンと，汎用的な PIM 上でのエミュレーションで代替する場合の性能や電力を比較し，開発コストや運用コストに見合う場合は前者を開発すべきである．計算資源を食い尽くす (専用マシンへの移住がペイする) 傾向が高いアプリケーション群が専用マシンのねらい目である．それらのアプリケーション群を中心としたその見極めは今後の重要な研究課題の一つと言える．

参考文献

[11] International Roadmap for Devices and Systems. “Systems and Architectures (2020 Edition)”. In: *IEEE* (2020).

[211] Flagship 2020 Project. “2017 Annual Report”. In: *The 2016 Inrnational Conference for High Performance Computing, Networking, Storage and Analysis (Birds of a Feather)* (2016).

[212] P. Messina and S. Lee. “The U.S. Exascale Computing Project”. In: *https://www.r-ccs.riken.jp/wp-content/uploads/2019/08/Flagship_2020_Project_2017.pdf* (2019).

[213] 山本啓二 et al. “スーパーコンピュータ「京」の運用状況”. In: **情報処理** 55.8 (2014), pp. 786–793.

[214] *ABCI* **について**. URL: `https://abci.ai/ja/about%5C_abci/`.

[215] Luiz André Barroso, Jimmy Clidaras, and Urs Hoelzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool Publishers, 2013.

[216] International Roadmap for Devices and Systems. “Systems and Architectures (2017 Edition)”. In: *IEEE* (2017).

[217] *Ethernet Alliance: The 2018 Ethernet Roadmap*. URL: `https://ethernetalliance.org/the-2018-ethernet-roadmap/`.

[218] Andreas Dilger. “Lustre: The Next 20 Years”. In: *HPC I/O in the Data Center Workshop* (2019).

第 5 章

次世代型運用への要求

本章では，来るべき Society5.0 時代を踏まえて，センサデータや外部データベースの利活用，研究データの流通も含めたわが国の次世代計算基盤全体に求められる利用環境について議論する．

5.1 新しい利用形態とシナリオ

5.1.1 新しい利用形態

2.5.1 節でも述べたように，これまでのスパコンでは，バッチ処理を基本として，ログインノード以外ではシステム外部との接続を許さない運用が一般的であった．しかし，観測データやセンサデータ，外部データベースを直接取り込みながらリアルタイムに処理を行うことを考えると，システムの設計や運用が大きく異なってくる．

まず，将来の高性能計算基盤における利用シナリオを示すことにより，設計や運用に向けた問題点を明らかにする．

5.1.2 利用シナリオの例

5.1.2.1 リアルタイム観測によるデータ同化＋リアルタイムシミュレーションによる予測

■地震観測データによるデータ同化＋地震波動シミュレーション 日本全国には，気象庁，防災科学技術研究所，大学等研究機関の設置した約 1,300 の地震観測施設があり [219]，高精度のもの (Hi-net など [**jisin2**]) では 1 拠点当たり 100Hz, 27bit データが送信される．東京大学地震研究所を中心に，全国地震観測データ流通ネットワーク “JDXnet” が運用されており，地震波形データが各拠点でアーカイブされている他，ネットワーク接続することでリアルタイムに取得することができる．

このような観測データを用いて，データ同化手法を用いてシミュレーションの初期値として地震波動場を常に更新しておく．あらかじめ決めておいたスレッショルドを超える地震が発生した場合には，この波動場を元にしてアンサンブル計算による強震動シミュレーションを行う．2020 年現在の計算機の能力では，数百ノード程度の計算リソースがあれば実際の地震波動の 10 倍程度の速度でシミュレーションが完了する．

■ゲリラ豪雨予測 情報通信研究機構 (NICT) などが中心になり，フェイズドアレイ気象レーダーが大阪大学や埼玉大学などに設置され，30 秒から 1 分程度で雨雲の高速三次元観測が可能になっている．理研 R-CCS では，開発した「SCALE-LETKF」コードを使用して，2016 年には京コンピュータとフェーズドアレイ気象

レーダとを用いて，高精細なゲリラ豪雨の予測に成功している．また 2020 年現在も，大阪大学に設置されたフェーズドアレイ気象レーダと 3D ナウキャストによる関西地方の降水予報を実施している．さらに 2020 年度中，Oakforest-PACS (OFP) システム上で，埼玉大学に設置された世界初のマルチパラメータ・フェーズドアレイ気象レーダ（MP-PAWR）の観測データに基づき，リアルタイムシミュレーションを実施する計画である．OFP で実証実験を実施した (2020/8/25-9/5).

5.1.2.2 ビッグデータに基づく予兆検知，診断

■医用画像処理の事例 膨大な CT 画像や MRI 画像から疑わしい病変を洩れなく見つけ出すことは，習熟が必要であり，人数が限られた熟練した医師にとって大きな負担になっている．コンピュータ支援診断／検出 (computer-assisted diagnosis/detection, CAD) ソフトウェアの開発が行われてきており，高性能・高精度化を目的として Deep Learning を用いた学習が注目を集めている．一旦学習したパラメータは，限定された条件には適用できるものの，測定機器が異なったり，撮影条件が異なるような場合には，実際に得られたデータを使用して継続的に学習をし直すことにより高い性能を維持することが可能になる．そのためには，パラメータ自動探索に基づく DL 学習フローの構築が必要である．各施設のデータを用いて学習を行い，学習データのみを施設間で共有することで個人情報漏洩等のリスクも防ぐことができる．

■風車の異常予兆検知の事例 風力発電に用いる風車は，地理的にアクセス困難な場所に設置されていたり，故障の際の部材交換や修理対応に長期間を要するため，破損等が起きて発電が停止した場合に，損失が莫大となる．風車に標準的に備えられている監視制御システム SCADA などのログデータを機械学習を用いて分析することで，異常予兆検知を可能にする研究が進められている [220]．複数の事業者の間で，個々のサービスに関わる情報は秘匿しつつ，メンテナンス等に必要なデータについては共有することで，より精度を高めることが可能になる．

5.2 設備・管理

ムーアの法則が終焉に近づくことにより，HPC システムの消費電力上昇は今後さらに顕著になる可能性がある．省エネルギーに対するニーズは今後ますます増加するため，省電力／省エネルギー化を見込んだ HPC/データセンターのプランニングの検討が進んでいる．

■施設運用の観点での問題 消費電力に関連して，現状施設運用の観点で課題となりつつあるのが，DVFS 等の HPC システム側の省エネルギー機能の拡充や利用拡大によるアイドル時とピーク消費電力のギャップ拡大があげられる．これにより，契約事項を含む電力サプライヤとの調整や，サイト内での調整等が必要になることがあり，今後さらなるギャップ拡大を想定した対処を考えていく必要がある．また，アイドル時とピーク消費電力のギャップ拡大により，冷却設備の負荷変動も問題となる．熱負荷が急上昇・急降下することで，それに冷却設備が追随できない可能性もあり，予め負荷変動を想定した冷却設備の構築や，負荷変動の抑制などを検討していく必要がある

さらに，今後は温水冷却と DVFS による性能低下の問題も考えていく必要がある．特に，それらによる省エネルギーによる正の効果と性能低下による負の効果について定量的な評価が必要になる他，低コスト・エネルギー重視か，短アラウンドタイム重視か等，アプリケーション要求やシステム運用ポリシーにも応じて柔軟にシステム・施設運用が行える方法の確率も検討事項である．

■施設運用の観点で重要となる技術　設備設置や施設運用の観点から重要となる，省電力に関連する技術を以下にリストアップする．

- 外気導入，湖水利用，海水利用等や，サーモサイフォン等のさらなる省エネルギー化を実現するための冷却技術
- 投入されたジョブの情報と過去に実行されたジョブ履歴から，これから実行されるジョブの消費電力のプロファイルを予測し，それを元に冷却設備等の運転を最適化する技術
- HPC システムや冷却機器等から出力されるログを収集し，分析することにより得られた定量的な根拠に基づく運用改善を実現するための技術
- センターの改修や新設において，施設に関連する先端的な技術を総体的に踏まえ，省エネルギー化を実現するための設計技術

■課題　冷却等の HPC システムをサポートするための技術は着実に進展しており，最新の技術動向を常にキャッチアップしつつ，進展に貢献することも求められるが，国内における活動は相対的に見て活発とは言えない．国際的な取り組みに積極的に参加するとともに，この分野における人材育成について真剣に検討する必要がある．

5.3　ユーザ利用・課金モデル

5.3.1　Service Level Agreement(SLA)

■従来のスパコンにおける SLA　従来，国立研究所および大学等の計算機センターにおけるスーパーコンピュータでは，利用規約等は定めてあるものの，その中で明示的に Service Level Agreement(SLA) を定義してきていなかった．これは，研究用スーパーコンピュータが一般のホスティングサービス等と異なり，新規デバイス等を積極的に採用することによりその性能を引き出す設計とすることが多く，一般サービス等と比較して運用の安定性を事前に数値として担保することが出来ないことに起因している．運用の実際としては，明文化された SLA がないことを盾に無保証の運用を野放図に行っているわけではなく，障害発生時には運用者の裁量と資源の範囲内において，失われた計算時間・トークンの保証などの対応を行う場合が多い．

Society5.0 時代にはより社会においてミッションクリティカルとなるアプリケーションの実行基盤となり得ることも踏まえ，これからのスパコンにおける SLA の在り方を今後定義していく必要がある．

参考文献

[219] *2019/4* **現在の高感度地震計**. https://www.jishin.go.jp/database/observation_station/spots/.

[220] https://www.nedo.go.jp/news/press/AA5_101010.html.

第 6 章

技術課題と研究開発ロードマップ

6.1 デバイス・アーキテクチャ

6.1.1 汎用システム型

6.1.1.1 技術課題

4.1 節で述べた想定システムの実現に必要な技術を以下にまとめる．

まず想定 CPU に関しては表 4.2 に示したように，3.9GHz で動作するコアが 70 個搭載されており，消費電力は 351W 以内である．この高い周波数と電力効率を同時に満たすことは現在最先端のプロセスルールや EUV や GAA などの次世代半導体製造技術を実用化した 2nm 相当のプロセスルールを用いても不可能であり，同等の性能を達成するには周波数を 2/3 程度に引き下げる代わりにコア数を 1.5 倍程度に増やす必要がある．予測されるパッケージング技術の進歩により，この程度のコア数増加はチップレット化により十分実現可能である．なおソケットあたりの集積度は 4 倍以上の向上が見込まれるので，消費電力の増大を許容すればコア数をもっと増やすことも可能である．想定 CPU の SIMD 長は現行 CPU の 4 倍（2048 ビット）となっているが，実際には SIMD 長を拡大するのではなく行列積和演算ユニットの追加により要素あたりの演算数を増加する方向にある．SIMD レジスタに 16 要素が格納できれば 4×4 行列の積和演算が可能であり，この場合はレジスタ長が同じでも演算性能は 4 倍になるため，想定性能を実現できる．ただし SIMD レジスタにベクトルではなく行列を格納する場合，メモリ上に SIMD レジスタに直接格納できるように要素を連続配置するのはプログラミングの負担が大きいので，中間バッファの様なプログラミングを容易にするメカニズムの導入が望まれる．

いずれにしても，電力効率等の面で既存の汎用システムのアーキテクチャだけでは今後性能がスケールしないことも考えられるため，演算ユニットやメモリアクセスの点で加速機構の一部アーキテクチャ要素を汎用 CPU に取り入れるなどの工夫が必要となる．その際に，アプリケーションユーザから利用しやすいシステムにするためには，幅広いアプリケーションに対応しつつ，ソフトウェアも従来の汎用システムとの互換性が高い構成を検討していくことが重要である．そのためには，アプリケーション分野の研究者と密に連携しつつ，コデザインを今まで以上に推進していくことが必要である．

上記の要素技術の多くは，表 4.6 の想定 GPU を実現する上でも必須である．特に，想定 GPU は Tesla V100 の 1.3 倍の TDP で最大 12.9 倍のコア数に相当する性能を実現する必要があり，これを半導体プロセスの微細化だけで実現することは不可能である．上記性能を実現するためには，単純にコア数の増加のみに頼るのではなく，例えば Tensor コアのような行列演算ユニットを FP32 あるいは FP64 に拡張するなど，GPU

アーキテクチャの大幅な刷新が必要と考えられる．ただし，Tensor コアはピーク性能に比べて実効性能が低いという報告もあり，ボトルネックの解析や実効性能向上の方策検討も重要である．

想定 Vector Engine の実現においても半導体プロセスの微細化は必須である．ベクトルプロセッサでは特にメモリバンド幅が重要であるため，HBM の多チャネル化にかかる技術，具体的にはプロセッサに対して複数の HBM モジュールを接続するための技術，例えば接続の微細化，線長制限の緩和，インタポーザの歩留まり改善，なども特に重要となる．これらの技術によって B/F を維持または強化することが必須と考える．また，ベクトルプロセッサは一部アプリケーションで非常に高い実効性能を発揮しながら，ベクトル化率の低いアプリに対して実行性能が低いという問題を抱えており，ベクトルプロセッサ自身のアーキテクチャ改善や，ホスト CPU を用いたハイブリッド実行のためのプログラミング環境の強化など，性能を補うための工夫が必要となると考えられる．

メモリに関しては広帯域化と大容量化の両方が必要である．HBM や HMC などのスタックド DRAM ついては，積層 DRAM 数の増加に加え，メモリモジュールあたりの帯域の増加やチャネル数の増加が課題として挙げられる．DDR DRAM については DRAM 容量の増加やチャネル数の増加が課題となる．HBM については実際の技術トレンドは今回使用した想定よりもチャネルあたりの帯域は 2，3 割低いと予測されるが，パッケージング技術の進歩によりソケットあたり 2 倍以上のチャネルを搭載できると予測されるので，帯域，容量とも実現可能と考えられる．また，メモリ容量を増やすという意味では，SCM や DCPM などのストレージクラスメモリの利用は有力なオプションの 1 つである．さらに，メモリの省電力化の観点から，不揮発性メモリの実用化も課題の 1 つとして挙げられる．

インターコネクトに関してはノードあたりの帯域幅の増加が必要である．InifiniBand については NDR の次の規格である XDR の実用化が期待されるが，CPU，メモリの性能向上に対して十分ではない．さらなる広帯域化に向けて，シリコンフォトニクスやコパッケージドオプティクスなどの光伝送技術の実用化が重要な課題である．

ストレージに関してはファイル I/O 帯域の増加が必要である．ストレージシステムの I/O 帯域はストレージネットワークの帯域によって律速されており，広帯域なストレージネットワークを実現する上での課題はインターコネクトの課題とほぼ同じである．また，AI やビッグデータを含む Society5.0 アプリケーションでの利用を想定したストレージアーキテクチャの再検討も今後の課題となる．

6.1.1.2 研究開発ロードマップ

CPU・GPU のアーキテクチャ検討 2021–23:**要素技術調査とワークロード検討** 3 次元積層技術やシリコンフォトニクスなど，将来利用可能になるデバイステクノロジやそのパラメータ，コストも含めた実現可能性の調査を行う．また，アーキテクチャ設計に影響を及ぼす可能性のある将来的なワークロードを選定する．

2022–24: **コデザインによるアーキテクチャ候補の探索** SIMD 型や行列演算ユニットの構成，メモリ階層など，複数のアプリケーションとのコデザインにより共通して有効な要素を洗い出しつつ，電力コストや実効性能の点からアーキテクチャ候補を探索する．

2025–28: **ハードウェア設計・開発** 探索されたアーキテクチャ候補に基づき，ハードウェアの設計および開発を行う．

ネットワーク・ストレージ 2021–23:**要素技術調査とワークロード検討** 新デバイスや実装技術など，将来利用可能になるテクノロジを調査する．また，AI やビッグデータ，Society5.0 に資するアプリケーション群を定義し，運用技術も踏まえ将来に必要なネットワーク・ストレージに関する要素技術を

調査する．

2023–24:**アーキテクチャ検討**　調査結果に基づき，ネットワーク・ストレージアーキテクチャ候補を選定する．

2025–28: **ハードウェア設計・開発**　探索されたアーキテクチャ候補に基づき，ハードウェアの設計および開発を行う．

6.1.2 専用システム・PIM 混載型

6.1.2.1 技術課題

専用的な加速機構や PIM を混載するシステムを構築する場合，その実現に必要な技術課題を以下にまとめる．

1. 論理的なものの選択とその最適実装の探求
   - ホスト CPU と演算加速機構・PIM との区別と役割分担 (ホストはクリティカルパス上の逐次処理 or 低並列性な処理 or アクセスローカリティが低い処理とノード間通信関連)
   - 演算加速機構・PIM 側にオフロードする機能 (並列度が高く OoO の利得が少ない処理全般，アプリ由来の専用機能や Scater/Gather 等のメモリ側で実行するメリットがある処など)
   - 演算加速機構・PIM 側の命令セット
   - ホスト CPU と共有するメモリ空間の有無と一貫性維持方式・通信方式 (ノード内一貫性維持あり，一貫性維持なし)
   - 異なる演算加速機構・PIM 間で共有するメモリ空間の有無と一貫性維持方式・通信方式 (ノード内一貫性維持あり，一貫性維持なし)
   - 演算加速機構・PIM 側のメモリ階層 (L1,L2,L3,scratchpad, 高バンド幅 DRAM, 大容量 DRAM,SCM 等) とその制御
   - 演算加速機構・PIM 上で OS を動かすか否か (OS を動かす，動かさない)
   - 演算加速機構・PIM 上の資源の仮想化対応 (アドレス変換の仕組みはホストと共通，対応しない)
   - 演算加速機構・PIM 側の電力・機能・性能調整ノブ (アプリで使わないユニットの電源を切るなどの縮退モードの設置，DVFS による電力制御)
   - 演算加速機構・PIM 側の性能カウンタ (ホストと基本的に同仕様，別仕様，なし)
   - 演算加速機構・PIM 側の耐故障対策 (ホストと基本的に同仕様，別仕様，なし)
2. 物理的なものの選択とその最適実装の探求
   - モジュールの実装形状 (2.5D パッケージ，チップレット・3 次元実装 (TSV 型 3D)，モノリシック 3D (ILV 型 3D)，WSI)
   - 既存規格 (PCI,DIMM,HMC) への対応や修正
   - ホスト〜演算加速機構・PIM 間および演算加速機構・PIM 間結合網のトポロジやバンド幅
   - 搭載するメモリの種類 (HBM,DRAM,3D Xpoint,XL-FLASH, その他)
   - 演算加速機構・PIM 側のメモリバンド幅，メモリ遅延，メモリ容量
   - 演算加速機構・PIM 側コアの能力とコア数
   - ホスト当たりの演算加速機構・PIM 数

- 演算加速機構・PIM 側の電力バジェット
- 冷却方法
- 電力供給
- 開発コスト削減法 (ホストと演算加速機構・PIM の共通 chiplet 化の検討など)

3. プログラミング関連の選択とその最適実装の探求
   - 演算加速機構・PIM 側資源の見せ方 (NUMA node・その他)
   - 対応するプログラミングモデル (共有メモリ上のデータ並列・タスク並列，PGAS，メッセージ交換，MapReduce，その他)
   - ライブラリとして提供する支援機能 (領域確保・解放，不連続アクセス，疎行列処理，密行列処理，深層学習，グラフ解析，データ解析，Collective 通信，スレッド，その他)
   - API(One API，その拡張，独自)
   - 対応するプログラミング言語，コンパイラ，デバッガ，プロファイラ
   - アドバイザ (演算加速機構・PIM 上で実行させるメリットがある場所の判定とメリットの予測)

6.1.2.2 研究開発ロードマップ

演算加速機構のアーキテクチャ検討 2022–23:**加速機構として実現すべきアプリケーション・カーネルの調査** 将来のスーパーコンピュータ利用を見据えた際に，演算加速機構や PIM が真に必要となり得るアプリケーションやカーネルを調査する．また，汎用システムに統合できる可能も同時に調査する．

2023–24: **演算加速機構・PIM のアーキテクチャ検討と予備評価** 演算加速機構が必要なアプリケーションがある場合，コデザインも含めてアーキテクチャとハードウェア要素技術を検討する．また，汎用システムと比べての性能・電力の定量的なメリットを明らかにする．また，必要なシステムソフトウェアに関して，その開発可能性をシステムソフトウェアの研究者とともに検討する．

2025–28: **ハードウェア設計・開発** 探索されたアーキテクチャ候補に基づき，必要となるハードウェアの設計および開発を行う．

6.2 システムソフトウェア

6.2.1 基盤ソフトウェア (OS，仮想化，コンテナ)

6.2.1.1 コンテナ，仮想化技術に関する要件，要求

HPC システムにおけるコンテナやユニカーネルの役割としては，計算資源の性能を損なうことなく，アプリケーションが要求する実行環境を提供できることが求められる．HPC クラスタではアプリケーションの再利用性が課題となっており，サポートされる並列ライブラリ (MPI 等) の種類やバージョンが違うために，同一コードを異なるクラスタ上で実行することが難しい．また，アプリケーションによって要求される CPU コアや通信・ストレージといったハードウェア資源が異なることから，最適な資源分配が難しく，アプリ側で明示的に指定することは移植性や再利用性を損なってしまう．このようなソフトウェア/ハードウェアの違いを抽象化し，ユーザが意識せずともアプリケーションの要求に応じて計算資源を利用できることが望まれる．

6.2.1.2 不揮発性メインメモリ (NVDIMM) に関する要件，要求

NVDIMM を HPC システムで活用するためには，アプリケーションの互換性と NVDIMM の性能の両立が求められる．アプリケーションの互換性を担保するためには，DRAM システム向けの過去のソフトウェア資産を容易に引き継げることや，揮発性のキャッシュとのコヒーレンシを取れることが求められる．性能面については，NVDIMM は 2028 年においても書き込みレイテンシは DRAM よりも大きい (5 倍〜10 倍程度) と予想されており [221]，この遅い書き込み性能を隠ぺいするようなメモリ管理機構が必須となる．

6.2.1.3 コンテナ，仮想化技術に関する技術課題

コンテナは前述の通り，一部の HPC システムで導入・運用が進んでいるものの，導入から日が浅いこともあり，実際の処理性能や運用上の課題について明らかになっていない点が多い．また，HPC システムにおけるコンテナの導入・性能評価に関する研究報告は少なく，ほとんどがネットワークレイテンシや単一アプリケーションの処理性能といった基礎的なデータを提示するのみにとどまっている [83, 84]．そのため，まず実際の HPC システムにおいてコンテナ環境における HPC アプリケーションの要求分析・性能分析を網羅的に行うことが重要となる．その上で，HPC アプリケーションの実行に最適化されたコンテナイメージや，HPC ユーザの多数のリクエストに応じた資源分配を行うためのコンテナスケジューラといったコンテナ実行環境の研究開発が必要である．

ユニカーネルの多くは，クラウド環境において KVM などのハイパーバイザによって提供される仮想 CPU/メモリ上で動作することを想定されている．一方で，性能要求の高い HPC システムでは，仮想化レイヤの多重化は OS オーバヘッドの増加を招くことから，ハイパーバイザの利用は適切ではないと考えられる．HermitCore [85] は，ユニカーネルをハイパーバイザを介さず物理コア上で直接動作させている．この方式ではハイパーバイザに起因する仮想化オーバヘッドを無視できるが，このようなベアメタル環境でのユニカーネル実行を可能にするためには，物理的なメモリ空間や CPU コアを分割，分配するための新たな CPU・メモリ管理機構が必要となる．HPC システムにおいてユニカーネルを導入するためには，このような処理性能に特化したユニカーネル実行環境の研究開発が重要と考えられる．

6.2.1.4 不揮発性メインメモリ (NVDIMM) に関する技術課題

NVDIMM はその性質上，DRAM よりも書き込みが遅いため，高い性能を実現するためには NVDIMM と DDR4 や HBM を混載する必要がある． このため，NVDIMM を主記憶として利用する場合にはどのデータを DRAM を置き，どのデータを NVDIMM に置くかを考慮する必要がある． このメモリの割り当てを効率よく行うために，様々な手法が研究されており，中にはアプリケーションの書き換えが必要なものもある． しかしながら，これまでのソフトウェア資産を利用するためには，大きなアプリケーションの改変を必要とする手法は現実的ではないと考えられる．

また，NVDIMM をローカルストレージとして利用する場合，NVDIMM とブロックデバイスでは，データの永続化に関する処理などが異なるため，追加の対応が必要となる． 従来のブロックデバイスであれば，I/O 命令によってデータを転送しており，I/O 命令が完了すればデータの永続化も同時に完了したものとみなせた． 一方，NVDIMM ではデータの書き込みはメモリアクセスであり，メモリアクセスは CPU でキャッシュされるため，このキャッシュをフラッシュしなければ永続化されない，という違いがある． このような違いがあるため，旧来のファイルシステムをそのまま流用すると，キャッシュフラッシュが多発してしまい，非効率である． 一方でファイルシステムとしてのインターフェイスを変更してしまうと，アプリケーション

に対して膨大な改造が必要となる． このため，現在 NVDIMM を効率よく利用しつつアプリケーションの改変を少なく済ます手法も研究されている．

Linux では，アプリケーションが NVDIMM など一部メモリ空間に直接アクセスするためのインターフェイスを提供する DAX と呼ばれる機能が提供されている． また，ext4 と xfs の 2 つのファイルシステムはすでにこの DAX に対応している． しかし，DAX 対応のファイルシステムであっても，依然としてアプリケーションが直接 NVDIMM にアクセスするよりもオーバーヘッドが大きい．また，アプリケーションが直接 NVDIMM にアクセスするよう改造する際に利用可能なライブラリとして，PDPK や libpmem などが開発されている． しかしながら，これらのライブラリを用いても，既存のアプリケーションで対応するには人手での開発が必要となる． 今後は，これらのファイルシステム やライブラリを利用して，NVDIMM を用いたアプリケーション実行の実現が進みつつ，より低い開発コストでより高い性能を引き出すためのファイルシステムやライブラリの研究が進められると考えられる．

6.2.1.5 研究開発ロードマップ

コンテナなどの新しい仮想化技術を HPC システムで活用するための要求分析や性能分析，および分析結果に応じた HPC システム特化型コンテナ/ユニカーネル 実行基盤の研究開発があげられる．また，NVDIMM を HPC システムで活用するためには，DRAM との書き込み性能差，不揮発性を考慮したメモリアロケーション機構やキャッシュ制御システムの検討が必要である．具体的には，以下の研究ロードマップで研究開発を行う．

コンテナ/ユニカーネル実行基盤 2021–2022 年 HPC システムにおける既存コンテナ (Singularity 等)，ユニカーネル (HermitCore 等) の要求分析・性能分析
2022–2024 年 HPC システムのためのコンテナ実行基盤の研究開発
2024–2026 年 HPC システムに特化したユニカーネル実行環境の開発
2026–2028 年 次世代 HPC システム向けの実装と導入
2028 年– 実 HPC システムにおける各仮想化環境の運用と機能改善の検討

コンテナ/ユニカーネル実行基盤 2020–2021 年 既存の NVDIMM 用ライブラリ・ファイルシステムを用いた HPC アプリケーションの実装
2021–2023 年 NVDIMM 利用時の HPC アプリケーション実行性能の解析および DRAM との比較，メモリアクセスパターン・キャッシュ挙動の分析
2023–2026 年 DRAM/NVDIMM のハイブリッドメモリ管理機構の研究開発
2026–2028 年 次世代 HPC システム向けの実装と導入
2028 年– 実 HPC システムにおける運用と機能改善の検討

6.2.2 大規模並列/高性能計算

6.2.2.1 CPU・アクセラレータに関する技術課題

近年の CPU の性能向上は，主にコア数と SIMD 幅/ベクトル長の増加によって実現されてきた．IRDS のロードマップを見る限り，この傾向は今後も長期的に続くことが強く示唆される．GPU の高機能化，言い換えれば汎用化もこれまでと同様に進むことが予想される．ハードウェア構成としては CPU が DDR メモリを搭載し，GPU が HBM メモリを搭載するものが多く，両者の性能的な特長の違いを生じる要因の一つに

なっている．ただし，高性能計算分野を対象として開発された国産プロセッサである Fujitsu A64FX や NEC SX-Aurora TSUBASA Vector Engine は，すでに HBM を搭載する CPU であると言える．今後は，搭載コアやメモリの機能と性能が異なるプロセッサがさらに多数登場し，それらを主要アプリケーションや用途に合わせて使いこなす必要性が増すことが予想される．

次世代システムで利用可能な CPU やアクセラレータを考えると，現在よりも得手不得手の差が大きくなることは深刻な技術的課題といえる．例えば，IRDS のロードマップによると 2028 年ごろの CPU の SIMD 幅は 2048 ビットであり，倍精度浮動小数点であれば 32 要素のベクトル積和演算を 1 命令で行うことが想定される．したがって，ベクトル化できないカーネルループで積和演算もできなければ，単純計算で理論最大性能は少なくとも 1/64 になる．汎用性の高い CPU ですら処理によってこれだけの性能差が生じることから，より汎用性の低いプロセッサでの得手不得手の差はさらに顕著になると容易に予想できる．

2020 年時点での CPU と GPU との大きな違いとして，GPU 側では OS が動作していないことが挙げられる．このため，CPU の存在なしに GPU を利用することはできず，GPU を搭載する高性能計算システムは自ずとヘテロジニアスな構成となる．このことは CPU と GPU とが連携しなければ高性能計算アプリケーションを実行できないことを意味しており，円滑かつ効率的な連携ためにシステムソフトウェアの果たす役割は大きい．例えば，NVIDIA 社は Unified Memory と呼ばれる CPU と GPU との共有メモリを実現する機能と提供し，CPU と GPU のメモリ空間が異なることに伴うプログラミングの煩雑さを軽減している．しかし，実際にはデータの配置を意識しなければ高性能を達成することはできず，性能の観点からは改善の余地がある．さらには，CPU と GPU の両方を，それぞれの性能特性を考えて適切に役割分担させる点に関しては，現在ではプログラマの責任になっており，システムソフトウェアでの支援が期待される課題であるといえる．

高性能計算システムが専用 ASIC や FPGA として実装された多様な専用ハードウェアを搭載することを想定すると，システムソフトウェア技術として考えなければならないことは，そのように性能も機能も全く異なるハードウェア資源群をいかに効率的に連携させるか？という点である．CXL や CCIX といった次世代インターコネクト技術の標準化が進む昨今，そのような標準インタフェースで接続された多様なハードウェアを統一的に管理，利用できるシステムソフトウェアの技術開発は今後期待される課題の一つとなり得る．現在においても，アプリケーションをタスクに分割し，タスク間の依存関係に基づいて実行する必要性が高まっている．今後も，タスクやさらに細かい粒度にまでアプリケーションを分割し，それを多種多様なハードウェアが連携して実行する技術がより重要になってくるものと予想される．このためには，いくつかの既存研究 (古くは StarPU など) でも見られるように，タスク間の依存関係を記述すための言語処理系と，それをデータフロー処理として効率よく実行する実行時システムの両方の技術を研究開発していく必要がある．

6.2.2.2 量子アニーリングに関する技術課題

大規模な組み合わせ問題を量子アニーリングマシンで解くためには，多くの量子ビットが必要となる．例えば，2048 量子ビットを持つ D-wave 2000Q において，最もシンプルな問題と言われる巡回セールスマン問題を解く場合，わずか 8 都市程度しか解けない．各量子ビット間の相互接続が必要な場合，量子アニーリングマシンへのキメラグラフへのマッピングにより，取り扱うことが可能なビット数が全結合の場合に 64bit 程度に制限されるためである．(量子ビットのアクティブ状態などによってさらに減少しうる．) 2020 年 2 月に D-wave Hybrid による従来計算機と量子アニーリングマシンによるハイブリッドによる計算サービスが提供され，解ける問題サイズが緩和されたが，その詳細は公開されていない．量子アニーリングマシンのみを活用し高速に組み合わせ最適問題を解くためには，量子ビット数の増加が不可欠となる．量子ビット数の不足の要因の一つに，全結合が必要な問題を，キメラグラフと呼ばれる量子アニーリングマシンの構造に埋め込む必要

がある．全結合の問題キメラグラフへのマッピングはビット数的なオーバーヘッドも大きい．

また，量子計算以外の実行時間が大きいという点も技術的課題として挙げられる．アニーリングによる計算は数マイクロ秒と非常に小さいのに対して，埋め込み処理や Qubo の準備に時間がかかる．例えば，クラスタリングを QA で行う場合に，埋め込み処理の時間に約 97%，Qubo 準備に約 1% の時間がかかるのに対して，実際のアニーリング時間が 0.1% 程度と，アニーリング以外の処理に時間がかかることが分かっている．

さらに，量子アニーリングマシンやデジタルアニーラなど，アニーリングベースの計算機の普及を見越すと，それらを利用しやすくするための開発環境などが重要になる．量子アニーリングマシンやデジタルアニーラを使うためにはそれぞれが提供している API を利用したプログラミングが必要になる．例えば，D-wave マシンの場合は，独自の API を Python や C, Matlab から呼ぶ必要があり，デジタルアニーラなどとは異なる．そのため，各マシンに応じたプログラムが必要となり，プログラムの生産性・再利用性が低くなってしまう．また，解きたい問題の組み合わせ問題を Qubo 問題に落とし込む必要性がある．解きたい問題のハミルトニアン化および Qubo への変換方法など，容易にアニーリングを利用するための開発環境整備が必要になる．

■研究開発ロードマップ　量子ビット数の制限を解決する方法として，新しい量子アニーリングマシンの開発がある．次世代の D-wave 量子アニーラは 5640 量子ビット数と，D-wave 2000Q の 2048 量子ビットに比べ 2 倍となる予定である．次世代の D-wave 量子アニーラではキメラグラフよりも効率的に量子ビットを活用することが可能なペガサスグラフが採用される予定である．量子ビット数の問題や量子計算以外のオーバーヘッドを解決する方法して，問題の埋め込みの効率化が挙げられる．キメラグラフ (やペガサスグラフ) へ問題を埋め込む際に，余分な量子ビットが使われてしまうため，より効率的な埋め込みの研究開発が必要である．埋め込みアルゴリズム改良による量子ビット数の節約や高速化が期待される．

量子アニーリングマシンやデジタルアニーラを利用する開発環境の研究開発も必要である．各アニーリングマシンを統一的に利用ができる QA/SA のフレームワークの研究開発が進むことが予想される．

また，さまざまな計算のハミルトニアンによる定式化が進むと考える．一般的な問題をアニーリングマシンで解くためには，問題の定式化とその応用が必要であり，組み合わせ問題を加速できるアニーリングマシンを活用するための定式化が進み，アニーリングマシンが利用されることが予想される．これに伴い，Qubo 生成の自動化，Qubo 分割により並列計算，QA ・DA・HPC の解の精度・実行速度・問題サイズなどに応じた計算プラットフォームの使い分けなどに関する研究開発が進むと考えられる．

6.2.2.3　メモリ階層およびデータ移動に関する技術課題

メモリ階層の複雑化が進んでおり，これをアプリケーションが使いこなすために必要なシステムソフトウェアの実現が求められる．性能差のあるメモリを組み合わせを制御する技術，異なる距離を持つメモリと演算コアの制御最適化，不揮発メモリの活用，アクセラレータ・CPU 間のメモリ空間の制御技術が必要とされる．

性能差のある階層メモリの取り扱い手法　これまでの技術開発により，各種別のメモリを直接ユーザへ提供する機能は確立されている．具体的には C 言語において専用の malloc() 関数を用いて，各種別のメモリを確保し，直接操作できる．一方で，ポインタを直接取り扱わない言語や Python 等で取り扱われる抽象度の高いオブジェクトにおいてこれらを利用するためには，言語処理系を拡張する必要がある．

NUMA 環境でのプロセス配置　今後，チップ内 Sub-numa cluster の技術が広がり，NUMA 構成については，ますます階層化・複雑化が進むと考えられる．また複数種別のメモリも Numa node して表現されるため，これらも含めた演算コアとメモリの最適配置技術を確立してくい必要がある．また，一般的に並列プログラムは，プロセスとスレッドの両方を利用したハイブリッド並列技術が用いられているが，

プロセス分割の粒度，プロセスとスレッドの割り当ての最適化について，今後も研究開発を継続していく必要がある．

不揮発メモリ活用　高性能計算において不揮発メモリの不揮発性を活用する用途としては，チェックポイント技術が有望である．既に，システムソフトウェアの観点で，プロセスのチェックポイント自動取得機能が研究されているが，並列処理系においては，並列プロセス全体での一貫性をもったチェックポイント取得が研究課題となる．既存の並列処理系におけるチェックポイント取得技術に対してその書き出し先を不揮発メモリとするところから始め，課題抽出が必要となる．

同一ノード内プロセス間データ転送　階層メモリ間でのデータ移動は，プログラマが明示的にソフトウェアでコピーを記述する基本的な手段は提供されているが，ノード内 DMA 等の手段によるバックグラウンドでのデータ移動手段の提供が必要である．また，ハードによるキャッシュ制御技術との優劣の議論はい必要であるが，階層メモリ間でのデータ配置最適化をシステムソフト側で行う仕組みの検討が必要である．異なるメモリ空間の間のデータ移動は cudaMemcpy 等すでに成熟した手法があるが，API は memcpy 型であり，標準的なプロセス間通信 API である MPI とは異なるセマンティクスである．非同期インタフェースも実現されているが，例えば MPI の非同期通信と同時に完了を単一の API で待つことはできない．ノード内のデータ移動とノード間通信をそれぞれ非同期に記述した場合に，プログラマが記述しやすい API の提供が課題となる．

以上の課題を整理すると，システムソフトウェアとして取り組むべき課題は，同一ノード内プロセス間データ移動，NUMA 環境でのプロセス配置制御，不揮発メモリ利用によるチェックポイントの 3 点といえる．

■研究開発ロードマップ

同一ノード内プロセス間データ転送　2021–23: **全体設計・API 設計**　ハード側が提供する DMA やコアといったノード内データ転送の機能をどのように提供するか，また，これらのハード機能の基礎性能特性を見極めて，システムソフトとしての全体構成を設計する必要がある．また，GPU メモリのような同一プロセス内の異なるメモリ空間と MPI に代表されるプロセス間データ転送の統合的な API を設計する必要がある．

2023–24: **試作による効果検証**　設計されたプロセス間データ転送機能のプロトタイプを実装し，設計通りの機能・性能を満たせそうか確認する必要がある．

2025–28: **実システム適用可能レベルの実装**　対故障性や例外処理といった実システム運用に耐えうる実装およびテスト検証を実施する必要がある．

NUMA 環境でのプロセス配置制御　2021–23: **ターゲットとなるメモリアーキテクチャの基本設計**　2028 年に実現される計算環境においては，メモリアーキテクチャの基本設計が非常に重要なポイントとなるためこの概要が設計されたのちにシステムソフト設計を進める必要がある．

2023–24: **シミュレーションによる方式検討・評価**　ターゲットとなるメモリアーキテクチャにおいて，適切なプロセス配置制御方式を設計においては，当面は実システムが存在しないため，シミュレーションによる方式設計・検証評価を進める必要がある．

2025–28: **実システム適用可能レベルの実装**　実システムが実現したら，実際のプロトタイプ実装を進め，実機上で基本動作が確認できるレベルの検証が必要である．

2028–30: **実システム上での効果検証およびチューニング**　実機上のプロトタイプによる基本検証が完了したら，実用的ワークロードでの効果検証やチューニングを進める必要がある．

不揮発メモリ利用によるチェックポイント　2021–22: **全体設計**　チェックポイント機能については既に並列処理系におけるチェックポイント機能の実現例があることから，これをベースに不揮発メモリを活用する手法の全体設計を進める．

2023–26: **試作による効果検証，設計最適化**　設計に基づき，施策を実装し，効果検証および性能評価を行い，設計最適化を進める．

2026–28: **実システム適用可能レベルの実装**　耐障害性を考慮した実システム適用可能なレベルの実装を行い，実用レベルの機能を実現する．

6.2.2.4　通信，ネットワーク，インターコネクト，データ転送に関する技術課題

ネットワークアーキテクチャを活かした計算処理への干渉を最小限とするスケーラブルな通信機構提供のためのシステムソフトウェアの実現が求められる．具体的には，ルーティング制御，集合通信とデータ型のオフロード，高効率な進捗処理実現が課題となる．

ルーティング制御　ケーブル配置といった物理的な制約から 1 万台を超える領域では Torus 網が有利であるものの，数百〜数千台の規模ではネットワーク上の位置関係が比較的フラットである Fat Tree や，小さい直径を実現できる Dragonfly 系のトポロジーが利用されている．すなわち，2028 年のネットワークアーキテクチャは，その両方を実現可能な構成要素から構築されるのが好ましく，ルーティングについては，ハード実装が必須であるアダプティブルーティングと，Torus や Fat Tree で用いられるスタティックルーティングの双方を実現できるようにする必要がある．このためには，その柔軟性を実現するためにハード実装が必須である部分を除き，ルーティング機能の多くの部分をソフトウェアで定義可能に構成されるのが好ましい．したがって，このようなネットワークを制御するためのルーティング制御，経路異常検知と障害回避制御を実現するシステムソフトウェア開発が必要である．また，集合通信においてはネットワークの幾何学的特徴を活かして経路競合を削減する高効率なルーティングおよび通信順序を制御する MPI 技術の開発が必要である．

集合通信とデータ型　ネットワークアダプタやスイッチ上での集合通信のオフロード機能を活用するための MPI 実装に代表されるシステムソフトウェア技術の研究開発が必要である．ネットワークアダプタやスイッチ上に実装される集合通信やデータ型変換のオフロード機能は，計算ノード上で動作する MPI に代表されるシステムソフトウェアと協調動作させる必要がある．この協調設計が重要であるため，ネットワークアダプタやスイッチのオフロード機能の制御の一部は書き換え可能なファームウェアで実装されるのが好ましく，ネットワークシステム設計者は，これらの機能と計算ノード上で動作するシステムソフトウェアが効率よく動作するように全体設計を行う必要がある．

進捗処理　非同期通信の進捗処理については，計算処理に対するオーバヘッドをできるだけ低減させる実装が重要である．進捗処理の実装方法には，計算スレッドが MPI 処理を呼び出した際に実行する方法，計算ノード上で動作する別スレッドが実行する方法がすでに実用化されている．さらなる高速性と計算処理への性能干渉を低減させるためには，ネットワークアダプタ上のオフロードエンジンを強化し，進捗処理を実装する方法がある．この実装は，2.2 節で言及したネットワークアダプタ上でのオフロード機能と合わせて制御する方式であるが，メッセージの中間バッファのような大規模なメモリ資源は計算ノード上に配置するなど，計算ノード上のシステムソフトウェアと一体化した実装方式を研究開発する必要がある．

■研究開発ロードマップ

ルーティング制御　2021–23:**全体設計（ネットワークアーキテクチャ設計）**　ルーティング制御はネットワークアーキテクチャの設計，特にネットワークトポロジーとスイッチが担当するルーティング機能に大きく依存するので，2028 年にむけたネットワークアーキテクチャの基本設計をはじめに実施する必要がある．

2023–25: **ルーティング制御ソフトウェアの試作**　ターゲットとなるネットワークアーキテクチャに向け，ルーティング制御を実現するために必要なシステムソフトウェアとして機能を確定させ，設計・試作を実現する．

2025–28: **実システム適用可能レベルの実装**　耐障害性や保守オペレーションを含めた実システムでの利用のために必要となる機能を実装し，実際に実用的に利用可能なルーティング制御を実現する．

集合通信・データ型オフロード　2021–23: **全体設計（ネットワークアーキテクチャ設計）**　集合通信・データ型オフロード機能は，ネットワーク設計，特にハード（スイッチおよびネットワークアダプタ）側で実現されるオフロード機能とシステムソフトウェアの役割分担の設計が重要であり，両者を横断的に考えた設計を進める必要がある．

2023–25: **オフロード機構を模擬したシミュレーション環境上での試作**　ネットワークハードウェアが提供するオフロード機能はこの時点では実装できていないので，オフロード機能を模擬した環境上でシステムソフトウェアの設計・施策を進める．

2026–28: **実システム上での試作・改良**　実際のネットワークハードウェアとの組み合わせで，試作したソフトウェアの動作・性能を検証し，必要な改良を実施する．

2028–30: **実システム適用可能レベルの実装**　耐障害性を含めた実用的に利用可能なレベルの実装を実現する．

進捗処理　2021–23: **全体設計（ネットワークアーキテクチャ設計）**　進捗処理機能は，ネットワークアダプタおよびアシスタントコア等の設計を含めて全体を適切に設計する必要がある．

2023–25: **オフロード機構を模擬したシミュレーション環境上での試作**　ネットワークアダプタのオフロード機構を利用する場合は，この時点でのハード実装が存在しないので，シミュレーション環境上で試作を進める．

2026–28: **実システム上での試作・改良**　ネットワークアーキテクチャとの組み合わせで，試作したソフトウェアの動作・性能を検証し，必要な改良を実施する．

2028–30: **実システム適用可能レベルの実装**　耐障害性を含めた実用的に利用可能なレベルの実装を実現する．

6.2.2.5　ストレージ，I/O システム，データ管理に関する技術課題

分散深層学習などの新たなアプリケーションと NVDIMM などの新たなハードウェアの登場により，ストレージシステムに求めらる要件が変化し，これに応えるために新たな技術課題がでてきている．

ノード数とデータサイズ について　大量の計算ノードを用いて巨大なデータの解析を行う場合，グローバルファイルシステムから計算ノードへのデータ読み込み処理の高速化が課題となる．現在はバーストバッファ を用いてデータをステージングすることで，読み込み時間をジョブから隠蔽している．しかし，ノード数の増加に伴いバーストバッファ用のノード増加すると，ステージングの際のグローバルファイ

ルシステムへの負荷が増大するため，ステージングの高速化が次の課題となる．また，データの合計サイズだけではなく，データを構成するファイル数も爆発的に増大している．データを読み込む際にはデータの格納場所を知るためのメタデータの問い合わせはファイル数 × ノード数 となるため，小さくて大量のデータへのアクセスするワークロードにおいてはデータ転送のスループットだけではなくメタデータ処理のスループットの改善も重要な課題となる．

ハードウェアとソフトウェアスタック　近年 NVDIMM, HBM など新たなメモリやストレージが登場したことで，これらハードウェアの種類は多岐にわたっている．その中で，スループットやレイテンシ，容量や電力効率といった様々な観点から階層的なストレージシステムをどのように構築するかが課題となる．また，これら階層的なストレージシステムをソフトウェアからどのように利用するかも技術的な課題となる．特に，メモリシステムは HBM2 と DDR4, NVDIMM が混載され，これをどのようにアプリケーションから利用するべきか，アプリケーションからこれらを容易に利用できるような仕組みを提供できるかどうかが課題となる．また，NVDIMM などの近年登場した高速なハードウェアに対してこれまでの低速なハードウェアを前提とした I/O ソフトウェアスタックは逆にボトルネックとなっている．例えば NVDIMM をストレージとして利用する場合，これまでのファイルシステムはブロックデバイスを想定した I/O の最適化が行われているものの，NVDIMM はバイトアクセス可能であるため不必要な処理が多い．このようなボトルネックを解消し，実アプリケーションに適用していくことが技術的な課題となる．

ジョブ実行開始時の I/O について　先述の通り，近年ではスクリプト言語で記述されたジョブの実行による大量の動的リンクなど，それ以前のジョブ開始時のバイナリ読み込みとは異なる I/O パターンがジョブの開始時に発生している．大量の動的リンクでは小サイズで大量のファイルアクセスが大量のノードから発生することでグローバルファイルシステム に対して高負荷を与え得る．この新たな I/O パターンによる高負荷に対処することが技術的な課題となり得る．

チェックポイントについて　先述の通り，今後高い並列度での耐障害性を実現するには，大量のチェックポイントをグローバルな領域に書き出す必要がある．このためには，チェックポイントに必要なメモリ上のデータを高速にグローバルな領域に転送する機構の実現が課題となる．

アクセラレータへのデータ転送について　現在アクセラレータへのデータ転送は NVLink などアクセラレータ毎に独自の機構を提供することで対応している．しかし，アクセラレータ毎に独自の機構を開発することは開発コストの増大に繋がり，新たなアクセラレータの開発が難しくなる．今後はアクセラレータによらない共通のデータ転送の仕組みを開発していくことが課題になると考えられる．

■研究開発ロードマップ　データ読み出しの高速化は，様々なハードウェアとそれを利活用するソフトウェアの研究開発によって実現していくと思われる．高速なネットワークや I/O デバイス，NVDIMM 向けのソフトウェアスタックがすでに研究開発されており，今後これらを用いてアプリケーションが高速なハードウェアに対応することで技術課題の解決に繋がると考えられる．例えば高速な Ethernet デバイスなどをターゲットにした DPDK, 高速な SSD 向けのソフトウェアスタックである SPDK, NVDIMM 向けの PDPK や libpmem などがあり，これらを利用したソフトウェアの例もある．一方で，これらフレームワークやライブラリを利用するにはある程度の開発コストが必要であり，この開発コストの低減と高い性能の実現を両立するための研究開発が続けられると思われる．さらに， Persistent Memory over Fabric や Approximate Memory などの新たなハードウェアについても研究開発が行われており，これらについてもソフトウェアでどのように利用するべきかが研究開発され，技術課題の解決につながると考えられる．また，小サイズで大量のファイル

への I/O については，コンテナなどを用いてイメージファイルをマウントすることで，グローバルファイルシステム へのメタデータの問い合わせを減らし高速化できることが示されている [222]．このため，この技術課題は今後コンテナの利用やデータセットをイメージファイルとして扱う機構が整備されれば解決され得る課題と言える．ジョブ開始時のライブラリ読み込みについても研究が進んでおり，例えば Spindle [223] がある．いくつかの代表ノードのみがグローバルファイルシステムからライブラリのファイルを読み出し，これを他ノードに転送していくことでグローバルファイルシステム への負荷を低減しつつライブラリ読み込みを高速化する手法である．今後このような仕組みが広くシステムに組み込まれていくことで，ジョブ開始時のライブラリ読み込みの問題は解決できると考えられる．また，チェックポイントについては NVDIMM を用いた高速な永続化方法が研究されている．これと合わせて Persistent Memory over Fabric (PMoF) を用いた手法が今後研究されることで，大規模並列にも対応可能なチェックポイントの機構を構築できると考えられる．最後にアクセラレータへのデータ転送については，NIC に汎用的なプロセッサを搭載した SoC である Smart NIC を用いたデバイス間の通信機構が研究されており，この研究が進むことで，アクセラレーターによらないデバイス間の通信機構が実現できると考えられる．

6.2.3 プログラミング環境

プログラミング環境は今後多様化がますます拡大すると予測されているアーキテクチャやハードウェアに対して，どのようにユーザがアプリケーションの記述を行うべきかの抽象度を決定づける最重要なレイヤであり．今後のプログラミング環境の発展は，システムの性能やコーディングの生産性をシームレスに改善していくためにはますます重要となる領域であり，継続的かつ重点的な研究開発が求められている．特に，多様なアーキテクチャに対してどういう風にサポートしていくかという面は HPC 分野に携わる万人の関心事であり，プログラミングに係る研究者や開発者に向けられる期待も大きい．

特にヘテロジニアスな処理要素を持つ構成におけるハードウェアの抽象化におけるコーディング容易性と性能の両立は重要な鍵となるポイントである．伝統的に行われてきた API を提供する形でライブラリとして抽象化する手法に加えて，ディレクティブで抽象化する手法，さらには，ドメイン特化型言語（DSL）やアプリケーション特化型のフレームワークにより抽象化していく方法など様々なものが提案されている．対象となるハードウェアも，主に空間方向の並列性を活用する Wide-SIMD のような汎用的なものから，FPGA のような個別なアーキテクチャ，更には，柔軟なパイプライン構成により時間方向の並列性も活用するデータフロー型なども考えられる．もちろん，もう少し広い範囲の CPU/GPU のヘテロな環境といった既存の延長線上のシステム構成も十分に次世代のシステムとして想定できることもあり，システムの設計は多様化していくことが予想される．本節では，それらに対応するプログラミング言語，言語モデルに対するコンパイラ，そしてツール群についての方向性における課題等を網羅的にまとめることを試みる．

専用なものと汎用なものをどう作り分けていくか，差分をカバーするアプローチはどうすればよいのかという探索を行っていく必要がある．対象のアーキテクチャを問わず同じコードが対応するコンパイラバックエンドにおいて高度に最適化されるのが理想的であるが，その実現のためには，DSL とコンパイラの話の切り分けを考える必要もあるし，ライブラリなのか，ハードウェアの機能なのかも整理して取り組む必要もある．マルチレイヤコンパイラとして抽象度に応じた変換を重ねていくことにより，コンパイラの変換器としての役割の効率を上げていく継続的な努力も必要である．近年普及している AI 処理フレームワークは，既存のプログラミング言語の上にメタプログラミング技術を活用して実装される DSL とも見ることができる．例えば TensorFlow においては，HPC には詳しくないユーザでも十分記述可能な Python でモジュールで書いて，

性能に影響があるところは低水準言語で書かれたライブラリにアクセスする API が用意されているという実装である．このような抽象度の階層化は重要であり，並列分散処理に不慣れなユーザに対しても，大規模計算リソースをスケーラブルに利用できるような整備が必須である．フレームワークとして言語を拡張していく DSL の構築法に加えて，これまでの言語とは全く違う新しい言語として DSL を開発するアプローチもその分野に精通するユーザにとっては魅力的であり，例えば Julia のような行列演算を主体とする新規の言語も新たに開発され注目を集めている．

性能とコーディング・ポーティングしやすさには相反するものもあり，例えばその目標とするゴールも立場によって多様である．アプリケーションの開発者は DSL を用いて簡単にプログラミングできると考えられている面もあるが，CS 研究者はハードウェア抽象化による性能向上を狙っているという側面もあり，継続したコデザインによりその両方を考えるべきフェーズとなっている．このような方向性は Sycl や OneAPI が目指しているゴールとも一致し，ハードウェアに向けて適切なランタイムシステムの開発と，将来的なヘテロ環境での統一的な資源割当方式の確立がもとめられている．

6.2.3.1 プログラミングモデル

■次世代のプログラミングモデル　次世代のプログラミングモデルは，次世代のアーキテクチャトレンドに対応する必要がある．この観点から，次世代のプログラミングモデルには (1) 2020 年時点でのプログラミングモデルとその拡張，および (2) 新たなアーキテクチャ，アプリケーションへの対応や新旧のアーキテクチャの連携の 2 つの方向性が必要とされる．

前者については，OpenMP+MPI のハイブリッドプログラミングモデルに加えて，プロセッサ内のコア数の増加やトポロジの複雑化に対応する必要があり，OpenMP3.0 以降で導入された OpenMP-task などの細粒度タスクプログラミング，物理コア数よりも多くの論理スレッドを用いることで物理コアのアイドルを防ぐ Oversubscription プログラミングを積極的に利用する必要があると考えられる．

ネットワークで接続されたセンサなどの IoT デバイスから得られるデータを活用することが期待される Society5.0 時代においては，外部デバイスやエッジと呼ばれるデバイス近傍の計算資源と，従来の HPC システムとの連携が重要になると考えられる．そのようなアプリケーションにおいては，リアルタイム処理を平易に実現するプログラミングモデルが必要とされるとともに，必要なタスクを最適な計算機資源にオフロードする粗粒度タスクに基づくワークフロープログラミングが重要になると考えられる．量子計算機やニューロモーフィックコンピュータなどの新たなアーキテクチャと，従来の HPC システムの連携においても，既存システムからこれらのアーキテクチャにタスクをオフロードする – もしくはその逆 – 粗粒度タスクプログラミングは有用であると考えられる．

PGAS 言語については，2.3.3.1 でも述べたように，従来の生産性の高さに加えて高速な片側通信をサポートするなど性能面でも優位性が見られ，また，OpenMP task など他のプログラミングモデルとの連携も比較的容易であり，次世代システムにおいても有用であると考えられる．

■要件・要求と技術課題　OpenMP+MPI や OpenACC などの既存のプログラミングモデルは，現在システムの延長線上のシステムでの利用が期待されるが，そのような将来システムにおいて予想される新たな特徴，たとえば SIMD 幅増など，にどのように対応するかが課題となる．メニーコア化とコア間トポロジの複雑化に対応するために，アルゴリズムによってスレッド/プロセスを変更するなどより柔軟なプログラミングが必要となるかもしれない．

OpenMP などを用いた細粒度のタスクプログラミングについては，データ locality を損なわないタスクの

スケジューリングなど最適化が重要となる．効率的な実行を平易に実現するために，自動最適化などとの技術の連携も必要となると考えられる．自動最適化については後述する．ワークフローを中心とする粗粒度なタスクプログラミングについては，society5.0 時代の豊富なセンサ情報を用いた新たなアプリケーションを実現することが課題と考えられる．計算資源の柔軟な利用のためにコンテナベースでスーパーコンピュータを利用する試みが行われているが，このようなコンテナと外部センサとの連携を可能にするミドルウエアの開発が必要となると予測される．

続いて，ノード内コア内のプログラミングモデルについて技術的課題を述べる．自動最適化とコンパイラについて，コンパイラへのコード変換/生成の指示や，性能パラメータ探索を一般的な高性能計算のプログラマが行うことは困難である．このため，いかにしてそのような機構の存在を隠蔽し，利用者が意識することなく自動チューニングを利用できるようにすることは，未だに解決されていない技術課題である [123]．また，より多くのコード変換や性能パラメータを自動チューニングで利用できるようになることは，それだけ多くの組合せの性能を評価し，発見的に適切な組合せを見つける必要がある．爆発的に増加する組合せ数に対応し，実用的な時間内に妥当な組合せを見つけることも重要な技術課題として挙げられる．データ抽象化プログラミングモデルについては，現在，多くの場合は C++ ライブラリ等の形式でメモリアクセスパターンの多相性が実現されているが，開発サイクルが長く，一度開発されると長い間利用されると言われている高性能計算アプリケーションにおいて，特定のライブラリに完全に依存したコードを書くことには一長一短がある．このため，データ構造の抽象化に関しては，その標準仕様の確立が今後求められることだろう．また，メモリ技術の進展に伴ってメモリバンド幅は増加している一方で，そのアクセス遅延は変わらず，むしろ悪化する懸念もある．このため，リストアクセスやバンクコンフリクトなど，アクセス遅延が陽に実行時間に現れる可能性のあるメモリアクセスは，将来のシステムにおいては極力さけることが求められる．現在の HBM において規則的なメモリアクセスパターンだけに限定して考えても，アプリケーション内で扱う配列数が多い場合にはバンクコンフリクトの懸念が高まり，特定の配列サイズにおいて極端な性能劣化が見られることが指摘されている．このように，メモリアクセスパターンに着目した性能最適化や，それを支援するためのプログラミング環境やモデルは今後さらに重要な技術課題になると予想される．

■ロードマップ　ワークフローなどの粗粒度タスクプログラミングにおいては，クラウドやコンテナ技術に基づくスーパーコンピュータなどの計算資源とセンサなどのデータ入力システムの連携が必要となる．2022 年までにこれを実現するために必要なミドルウエアなどの技術要件を調査し，2025 から 28 年までに連携を実現する．同様の枠組みで古典計算機と量子計算機などの新アーキテクチャの連携についても考慮する．ロードマップは新アーキテクチャの開発ロードマップに依存するが，2028 年ごろまでには新アーキテクチャの実機もしくはシミュレータと古典計算機の連携をめざす．

6.2.3.2　コンパイラ

ポストムーアのデバイスが揃うまでの過渡期の間は，今の原理のコンピュータで効率よく動くソフトウェアとして再コーディングし，スケーラブルに性能向上を持続する必要性がある．その面で，コンパイラはコード変換の自動化を司る最重要な要素技術の１つであるといえる．

■要件・要求と技術課題　性能チューニング技術として手作業で行ってきたリファクタリング作業の自動化が引き続き求められる．コンパイラへのコード変換/生成の指示や，性能パラメータ探索を一般的な高性能計算のプログラマが行うことは困難である．このため，いかにしてそのような機構の存在を隠蔽し，利用者が意識することなく自動チューニングを利用できるようにすることは，未だに解決されていない技術課題である

[123]．また，より多くのコード変換や性能パラメータを自動チューニングで利用できるようになることは，それだけ多くの組合せの性能を評価し，発見的に適切な組合せを見つける必要がある．爆発的に増加する組合せ数に対応し，実用的な時間内に妥当な組合せを見つけることも重要な技術課題として挙げられる．

以下に具体的な項目を挙げる．

- マルチレイヤコンパイラの高度化
- 長いベクトル長への対応
- 自動チューニング機構や，性能プロファイリング機構と連携したコード生成法の確立と，エンドユーザーへの環境提供
- JIT を用いた部分評価 (Partial evaluation) によるカーネルのランタイム再構築
- OpenMP などディレクティブベースのコードの最適化

6.2.3.3 デバッグ/テスト/再現性

■要件・要求と技術課題　確率的コンピューティングでは，非決定的・確率的に発生する計算の非厳密性（ビット反転など）により計算結果にエラー（確率的エラー）が生じる．この確率的エラーは非決定的に発生するためアプリケーションの信頼性の検証が従来型の厳密解を保証するコンピュータに比べ非常に難しくなる．確率的コンピューティング時代に向けたアプリケーション開発では，確率的エラーが発生したとしても計算結果が許容範囲内におさまるエラー耐性の高い計算に関しては確率的コンピューティングにより高速化をし，厳密解が必要なエラー耐性の低い計算に関しては厳密解を保証できる従来型の計算資源を利用したハイブリッドな計算が主流になると予想される．これを実現するためには，アプリケーションカーネル毎にエラー耐性を定量的に評価する必要がある．しかし，確率的エラーは非決定的に発生するために，検証のために特定のカーネルの特定のデータにエラーを注入することができなため，実際の確率的コンピュータ上でのアプリケーションの信頼性の評価をすることはできない．

■ロードマップ　アプリケーションのソフトエラー耐性検証環境の構築し，信頼性とその時の計算性能の評価を行う．これを実現するために，我々はモデリング，シミュレーション，エミュレーション，仮想技術等を用いて確率的エラーを特定の箇所に決定的に注入できる様に制御可能なプラットフォームを構築する．具体的には，2022 年までに技術の調査を行い， 2025 年までにプロトタイプ実装をし 2028 年までに候補アーキテクチャ上で検証実験を行う．

6.2.4 データフレームワーク

6.2.4.1 データ学習・推論フレームワーク（TensorFlow, PyTorch etc.）

■要件・要求　様々なプロセッサの登場が今後も見込まれる中，それらの計算性能を引き出すデータ学習&推論フレームワークの実装が求められる．

■技術課題　多種多様なプロセッサをデータ学習&推論フレームワークに活用する場合，各プロセッサの性能を引き出すためのバックエンドの開発が必要不可欠となる．例えば，GPU の場合，Tensorflow のバックエンドから GPU を利用するためのオフロード向け API が提供されており，このオフロード用 API から GPU を呼び出す．NVIDIA GPU の場合，cudnn などが呼ばれている．

GPU 以外のアクセラレータや専用プロセッサを Tensorflow のバックエンドとして活用する場合には，

GPU 寄りの実装となっているオフロード用 API に合わせてバックエンドを開発する必要がある．

また，Tensorflow では各レイヤー内で計算が完結する．例えば，CNN 層, プーリング層などの各レイヤー内でそれぞれ計算が行われる．そのため，レイヤー同士をまたいだ一気通貫の計算が難しい点が挙げられる．GPU などのアクセラレータの場合，レイヤーごとの計算が終わる度にホスト側にデータを転送する必要が生じ，ホストとアクセラレータ間のデータ転送がボトルネックになりやすい．また，大規模な並列性を抽出しにくいという問題がある．

複数の GPU と次世代ストレージによるローカルストレージ (LFS) を備えた単一計算ノードもしくは小規模環境のように，I/O 性能がボトルネックとならない環境では，CPU/GPU の性能を効率良く活用することができる．しかし，近年単一計算ノードのメモリでは収まりきらないほど巨大な学習モデルをモデル並列により構築することにより，より正確な学習モデルの構築ができると報告されており，そのためには大規模計算環境においてより多くの学習データを高速に処理する必要がある [224].

複数のユーザーがバッチスケジューラを介して使用する大規模”共用”計算機では，各計算ノードに搭載されている LFS は計算中のデータの一時保存に限られるため，永続保存が必要なユーザーの学習データはグローバルファイルシステム（GFS）に保存される．そのため，実行時に全ての計算ノードが学習データを読み込むために同時に GFS へアクセスすることとなり，計算ノードあたりの I/O 性能が低くなる．したがって，大規模共用計算機における大規模深層学習では，学習フェーズで使用する計算ノード数 (つまり CPU や GPU 数) を増やし計算性能のみを増やしても，並列 I/O 性能がボトルネックとり，学習フェーズの速度がそれ以上向上しない「大規模深層学習のスケール化の問題」が起こる．その他，モデルのサイズが大きくなると勾配の平均値を求める際の集団通信（MPI Reduction など）が長くなり，全体の学習時間に閉める通信時間が膨大になってしまう．また，大規模環境では 1 プロセスあたりのミニバッチのサイズが小さくなるため，これにより収束が悪くなり推論精度に影響を与えるという問題もある．

■研究開発ロードマップ　専用プロセッサやアクセラレータなど各プロセッサの登場に合わせたデータ学習&推論フレームワークのバックエンドの研究・開発が求められる．特に，データ学習および推論のネットワークを考慮したバックエンドの実装が必要になると考えられる．

大規模共用計算機を最大限利活用するためには，大規模深層学習のスケール化による問題を解決することが非常に重要である．具体的には，上記で挙げた (1) I/O 性能の問題，(2) 大規模モデル学習における集団通信の問題，(3) 極小ミニバッチによる収束性の低下，などある．このためこれらの問題を解決するために，2022 年までに (1)(2)(3) を解決するための既存手法の調査や技術検討を行い，2025 年までに (1)(2) を解決するため，下位のストレージ・アーキテクチャやネットワーク・トポロジーなどを考慮し I/O や通信を総合的に最適化するミドルウェアのプロトタイプを開発し，2028 年には候補アーキテクチャ上で検証実験を行う．この検証実験では，学習率を動的に変えるなど新しい学習方法を開発し (3) の問題の解決をはかる．

6.2.5　性能解析ツール

6.2.5.1　プロファイラ，トレーサー，性能解析方法論

■要件・要求と技術課題　プロファイラ，トレーサ，性能解析方法論については，プロファイリング，性能解析法を高度化し，リファクタリングやチューニングの生産性を上げることが課題として挙げられる．システムおよびアプリケーションの巨大化複雑化にともない，アプリケーション性能チューニングの工数は増加しており，このために性能分析の自動化が必要とされる．同様に，トレースを用いる性能解析は，イベントログの増

大を避けられないため，on-the-fly 処理，リアルタイム処理などの In-situ プロファイリングが必要とされる．

また，エンドユーザーでも容易に解析が行えるようなシステムソフトウェアのツールチェインとしての整備も重要である．特に，FPGA や GPU などの CPU 以外の演算器においては，統一的で平易な手順かつアプリケーション開発者から「読み取りやすい」性能プロファイリング手法および基準の確立が望まれる．

6.2.5.2 性能モデル・シミュレーション

■要件・要求と技術課題 性能モデル・性能シミュレーションは，推定精度と実行速度の間にトレードオフが存在する．とくにサイクルアキュレートな CPU シミュレータでは，数秒程度の計算カーネルのシミュレーションに数日を要することもあり，この傾向は OoO やマルチコアなど，複雑なアーキテクチャに対するシミュレーションについて顕著である．一方，シミュレータの内部の処理も複雑になることから，シミュレーションツールの並列化は難しい．同一モジュールに対してもモデルベース，シミュレータベースなど複数のツールを組み合わせることで，混合精度演算のように，高速高精度な推定を提供することを検討する必要がなる．

性能モデル・性能シミュレーションについては，2.3.4.2 でも述べたように，CPU，メモリ，通信など個別のシミュレータや性能推定ツールは多く存在する．システム全体を統合したモデル化・シミュレーションを平易かつ高速に行えるようにすることは重要な課題である．さらに，それぞれの要素について，アーキテクチャのトレンドを見据え，シミュレータや性能推定ツールを更新しつづけることが必要となる．

量子計算機シミュレータの課題としては，現実的な時間でシミュレートできる量子ビット数を増やすこと，古典計算機にはないノイズなどをシミュレートすることなどがあげられる．量子ビット数については 1 ビット増やすごとにシミュレータが扱うデータ量は倍になるため，通信の削減などが必要とされる．

6.2.5.3 ロードマップ

プロファイラ・トレーサ・性能解析方法論および，性能モデル化・性能シミュレーションの基本的なロードマップについては，アーキテクチャのトレンドを見据えて適用範囲・対象システムの拡充し，シミュレータや性能推定ツールを更新しつづけることが必要となる．同時に，メモリ，ネットワークなどといった解析対象やアーキテクチャによらない要素技術について，以下のようなロードマップが想定される：

-2025
- In situ プロファイリングのためのインターフェースやライブラリの開発
- 入力データのチェックポインティングを含めたカーネル部の自動判定と切り出しなどの平易化

-2028
- シミュレーションのさらなる高速化と詳細化
- システム全体を統合したモデル化・シミュレーション

6.2.6 利用高度化ツール

6.2.6.1 省電力化システム

■技術課題 個々のアプリケーションに着目した省電力化に関連する技術課題は，上述の技術動向からもわかる通り，主に a) アプリケーションの構造解析・プロファイリング， b) 電力制御最適化アルゴリズム， c) システムソフトウェア（ジョブスケジューラ，OS）との連携，の 3 点に大別することができる．以下に，それぞれの項目に関連する具体的な技術課題を挙げる．

- **アプリケーションの構造解析・プロファイリング**:　実アプリケーションはその構造や実行状況に応じて消費電力に対する特性が動的に変化する．そのため，適切に消費電力の削減を行うためには，アプリケーションの構造を解析し，DVFS や PG を適用するタイミングを抽出するとともに，実行性能と実行時消費電力に関連する情報を取得する必要がある．また，システム内に多種多様な計算資源が含まれることになるため，ヘテロジニアスな環境や製造時のばらつきを考慮したプロファイリング技術も重要である．詳細なプロファイル情報はその容量が大きくなるため，どのようにプロファイル結果を保持・活用するかも考慮する必要がある．
- **電力制御最適化アルゴリズム**:　他項目でも言及されている通り，大規模な計算機システムにおいても，今後は仮想化技術やコンテナの利活用，バッチジョブ以外のストリーム処理やリアルタイム性の高い処理も取り扱う必要がある．つまり，特性の異なる様々なアプリケーションや実行状況に応じて，適切に電力制御のタイミングと程度を決定するアルゴリズムの研究開発が重要となる．また，このような状況においては，様々な専門性を持つユーザの様々なアプリケーションを実行させることになるため，省電力制御最適化処理を自動化するソフトウェアフレームワークの実現が求められる．
- **システムソフトウェアとの連携**: システムの特性や運用ポリシによって，各アプリケーションに対する要求は大きく異なる．例えば，各アプリケーションの実行エネルギーを最小化する，消費電力を一定以下に保つ，と言った要求があり，しかもそれは動的に変わりうる．このような状況に対応するためには，システム全体を担うジョブスケジューラとの連携が必要不可欠となる．さらに，ジョブスケジューラとの連携を考慮した場合には，システム全体の要請に応じて，個々のアプリケーションに対する要求が変化するため，それに応じて動的に省電力制御のアルゴリズムや条件を変化させることが求められる．

■研究開発ロードマップ　以上から，アプリケーション単位での省電力技術においては，2028 年に向けて以下のような流れで技術開発が進められることが望ましい：

- **〜2022 年ごろ**: 仮想環境・コンテナ上で動作するアプリケーションにおける消費電力制御・低消費電力化に対する要件を調査し，低コストなプロファイリング手法・アプリケーション構造解析手法の開発
- **〜2024 年ごろ**: プロファイリングおよびアプリケーション構造解析結果に基づく消費電力・性能のモデリング手法に関する研究開発と仮想環境への適用と電力制御最適化手法およびその自動化に関する研究開発
- **〜2026 年ごろ**: ジョブスケジューラとの連携手法の技術検討，アプリケーション・ジョブスケジューラ間，アプリケーション・電力ノブ間の統一インタフェースに関する研究開発，ジョブスケジューラとの連携を前提とした，動的な消費電力最適化・マネジメント手法の研究開発
- **〜2028 年ごろ**: 実システムにおける統合

6.2.6.2　耐障害技術

■技術課題　大規模高性能計算は多くの計算リソースを使用することにより膨大な計算パワーを利用することが可能である．しかし，それらを構成する計算ノードの増加により全体の部品点数が増加し，またそれらを制御するためのソフトウェアもより複雑化するため，アプリケーション障害率が増加する．一方，システムの大規模化によりメインメモリの総容量が増加し，チェックポイントのサイズが増加するが，並列ファイルシステムのスループットの増加は著しくない．このため，障害率とチェックポイントのオーバーヘッドが同時に増加する傾向にある．このため将来の大規模システムにおいても高い計算効率（FT も含めた計算時間あたりの有

効計算）を維持する必要がある．

■研究開発ロードマップ　これらの課題を解決するために，いかに効率よく C/R からの I/O を行うかが重要となる．このため，技術課題に向けた研究開発ロードマップとして，(1) AI を活用したチェック（シミュレーションデータ）ポイントデータの圧縮や (2) AI を活用したチェクポイントリスタート機構の最適化を行う．

(1) において， AI 技術の登場により AI を利用したデータ圧縮が登場してきている．これは，時系列的に発生するセンサーデータの時間発展を予め AI に学習させ，この学習させた AI モデルにより同じセンサーから次に出力されるデータを予測し，その差分だけを保存することにより圧縮率を劇的に上げる手法である．センサーデータに限らず，時間発展を扱うシミュレーションにおける定期的なチェックポイントも時系列データとしてみなせるため，この AI 圧縮技術を応用することによりチェックポイントの高速化が見込める．これは，チェックポイントに必要なデータサイズを削減できるからである．この研究開発のロードマップとして 2022 年までに圧縮技術の調査をし，2025 年までにプロトタイプ実装をし，2028 年には候補アーキテクチャ上で検証実験を行う．

一方，(2) において，C/R のチェックポイントの間隔等のチューニングには，これまでモデリングが用いられてきた，しかし近年 Multi-level かつ Asynchronous なチェックポイントなどの登場により正確に C/R の振る舞いをモデリングすることが困難になってきた．一方正確に C/R の振る舞いを捉えチューニングを行う手法としてシミュレーションを行う方法があるが，膨大な量のシナリオをシミュレーションし，その中から最適なチューニング値を見つけるのは，ジョブを実行するたびにこのシミュレーションを走らせるため，時間がかかり現実的ではない．このため，AI を活用した C/R 機構の最適化を行う．これは予めあらゆるシナリオでシミュレーションによりデータを生成し AI に学習させ，ユーザが最適な C/R のチューニング値を知りたい時に，AI 推論で瞬時に出力するものである．この研究開発ロードマップとして，2022 年までに技術の調査をし，2025 年までにプロトタイプ実装を行い，2028 年までに候補アーキテクチャ上で検証実験を行う．

6.2.7　資源管理

6.2.7.1　ジョブスケジューラ

■技術課題　様々な特性を持つアプリの実行や仮想化レベルの異なる計算機による共有計算機資源利用が求められる．さらに，様々な特性を持つアプリが混在する共有計算機資源の効率的な利用が必要である．また，データ活用ニーズの拡大により，機微情報を扱う処理をセキュリティを確保しつつ，共有資源上で実行したいという要求が想定される．しかし，異なる利用モデルとセキュリティレベル (資源分離レベル) を有する計算資源を共有計算機資源環境で提供するためのスケジューリング手法やシステムソフトによるサポートは現状で不十分である．

異なる時空間特性を持つアプリが混在する共有計算機資源の効率的な資源利用を行うための最適化が必要である．また，資源分離レベルが大きく異なる資源を統合的に制御する資源管理技術が重要となる．合わせて，資源分離レベルの異なる計算資源を共有計算機資源上に最適配置する技術が必要である．

■研究開発ロードマップ　コンテナ，VM，ベアメタル，ユニカーネル等の利用について，HPC センタごとに要求・要件の調査が必要である．また，セキュリティレベルの異なる処理の資源分離レベルについても要求要件の調査・定義が必要である．さらに，資源分離レベルの異なる計算資源の統合管理が可能な資源管理システムの開発が必要である．資源管理システムの実現と合わせて資源分離レベルの異なる計算資源の最適配置技術の開発が必要である．

6.2.7.2 システム全体の電力マネージメント

■技術課題 大規模ジョブが実行される際に瞬間的に数 MW 程度の急激な消費電力変動が生じる．そのため，急激な負荷変動に応じた冷却施設の制御が必要であり，ジョブ実行と連携し冷却施設の制御を行う必要がある．また，急激な消費電力変動は，地域の電力網に悪影響を与えるため急激な負荷変動を抑制することも重要であり，急激な負荷変動を抑制する電力管理システムが必要となる．データセンタ全体のアイドル時電力とピーク時の電力の差について上限を設定できる電力管理システムが必要となる．

また，HPC システムでは多数のジョブが実行されているが，ジョブによって消費電力に大きな差が生じる．そのため，システム全体の省電力化のためには，ジョブ間で電力を融通することが重要となる．さらにシステム全体の消費電力の電力予測が重要となる．電力予測によって電力消費の需要に応じた冷却設備の準備やジョブ間の電力融通が可能となる．

上記を踏まえて，ジョブ実行に応じた冷却システムの制御が必要となる．計算機室内の温度が下がるまでには時間がかかるため，ジョブの実行前にジョブが消費する電力を予測する必要がある．

急激な負荷変動を抑制するためには CPU やアクセラレータの DVFS 機能を用いて消費電力のピーク値を段階的に変化させることで緩和可能である．しかしながら，大規模な環境で同期しつつ周波数を変更するための技術が必要である．運用面では急激な負荷変動を蓄電池や蓄熱プールによって吸収することが可能であるが，蓄電池や蓄熱プールの電力・熱容量を考慮したシステム全体のスケジューリングが課題である．

ジョブ間で電力を融通するためには各ジョブとリソースマネージャ間で電力情報を共有する仕組みが必要であり，電力情報をリソースマネージャとジョブ間でやり取りする API の設計が重要となる．また，消費電力・冷却状況に応じたジョブの配置によって，電力効率を向上させることが課題である．

さらに電力予測における予測精度向上が重要となる．また，効率的な実行のための計算資源・電力資源の最適化が重要となる．

■研究開発ロードマップ 蓄電池や蓄熱プールの電力量・熱容量を考慮した資源管理システムの研究開発が必要である．また，冷却施設等の設備と連携可能な資源管理システムの研究開発が重要となる．さらに，電力融通をおこなうスケジューラの研究開発，消費電力や冷却状況に応じたジョブの配置最適に関する研究開発が必要となる．

6.2.7.3 リアルタイム処理

■技術課題 HPC ではバッチ処理に特化したシステムが多数であったが，バッチ以外の計算需要としてリアルタイム処理・ストリーミング処理が求められている．リアルタイム処理をバッチ処理の延長，つまりある許容時間内にバッチ処理が開始すればよいと考えると，リアルタイム処理の実現には高優先度のジョブが実行中のジョブを押しのけて実行できる仕組みが必要である．

一方，ストリーミング処理や対話的処理はバッチ処理とは相容れない機能である．ストリーミング処理では常に送られてくるデータを処理し続けるプロセスが動き，バッチジョブとは違って処理に明確な終わりはない．またイベント駆動型，対話的処理を実現するにも，なんらかの永続的に動き続けるサービスプロセスがバッチ処理と共存して動くことが求められる．

これらはもちろん HPC 以外のシステムでは単体のシステムとして動作している．技術的な課題は，どのようにしてこれら各種システムとバッチ処理システムを HPC の 1 つのシステムとして効率的に動作させるかということである．

リアルタイム処理では高優先度のジョブを実行中のジョブに割り込ませて効率的に実行する仕組みが必要である．高優先度ジョブが要求するノード数や実行時間がシステムリソースの空きに収まればすぐにジョブは実行できるが，実際には様々なジョブが実行中のため，ジョブを実行するにはノードと時間の両方が空くの待たなければならない．割り込ませて実行するには実行中ジョブを停止 (kill) して実現できるが，停止させられたジョブのそれまでに実行してきた計算リソースが無駄になる．このため，チェックポイント・リスタート技術を用いたジョブをプリエンプトできる資源管理システムを開発する．

ストリーミング処理に関してはバッチやストリーミング処理を始めとする多種多様な利用需要に答えるために，計算ノードをどの処理に割り当て，需給に合わせてノードを増減できるメタスケジューラを開発することが望まれる．

6.2.8 外部資源連携

6.2.8.1 クラウドや IoT との連携

■技術課題 スーパーコンピュータのサービスにおいても，ハイブリッドクラウド技術を活用することにより，計算機システム保守によるジョブの停止，また繁忙期および電力逼迫時のジョブの処理待ちを回避する等，システムの可用性や処理能力の向上を実現することが必要である．従来のスーパーコンピュータの運用では，バッチ方式により計算資源を複数のユーザ間で共有する方式が主流であり，クラウド（IaaS）のようにユーザが計算資源を専有する方式とは異なる．またクラウドにおいても，VM やベアメタルといった資源分離の方式が異なる．このような運用形態または資源分離方式が異なるシステム間においても，計算資源の管理を容易にし，クラウドバースティングのような計算資源の融通を柔軟に実現することが必要である．また，スーパーコンピュータとクラウドの計算資源間で，遠隔のデータに安全かつ高速にアクセスできることも必要である．

センサ等から収集される IoT データ等をリアルタイムに解析するニーズは HPC 分野でも拡大することが予想される．スーパーコンピュータのサービスにおいても，データ収集機能を提供し，ユーザがデータ収集から解析までのワークフローをシームレスに実行できることが必要である．また，医療分野や企業利用等では機微情報を含むデータを解析するニーズが高い．スーパーコンピュータ上でのデータ解析では，高い計算能力を活用した大量のデータを扱うことが期待される．そのため，広域ネットワークの安全性や高速性の機能と連携しながら，安全かつ高速なデータ収集を実現する必要がある．

広域ネットワーク上の様々な拠点で生成・蓄積される大量のデータをスーパーコンピュータ上で解析するため，将来の広域ネットワークの高速な通信帯域を活用して，高速にファイルを転送できることが必要である．既存の広域ネットワーク上での高速ファイル転送技術の多くでは，複数の TCP コネクションを作成し，データを並列に転送することで高速通信を実現している．800Gbps 超の広域ネットワーク上での高速通信を実現するためスケーラビリティの向上が必要である．また，通信速度の向上に伴い，メモリアクセス性能の向上も必要であり，これらの複数の要素でバランスのとれた方式が必要である．

■研究開発ロードマップ スーパーコンピュータやクラウド上の計算資源を統一的なインタフェースで管理可能なソフトウェアを開発する必要がある．特に，これらの計算資源の運用方式や資源分離方式の違い，さらに資源制御のためのインタフェースの違いを吸収できることが必要である．スーパーコンピュータやクラウドの計算資源間で安全かつ高速な通信が可能なネットワークをオンデマンドに作成する技術を開発する必要がある．特に，安全な通信については，スーパーコンピュータやクラウド上で実際にユーザに割り当てられている計算資源間で外から分離されたネットワーク（VPN 等）を設定する等，ユーザまたはアプリケーションレベ

表 6.1　電力制御に関するロードマップ

達成時期	研究開発内容
2020 － 2021	富岳における電力モードの実装
2022 － 2025	アプリケーション毎の電力特性知見蓄積
2026 － 2027	アプリケーション毎に最適な電力設定の研究
2028 －	l アプリケーション毎に最適な電力設定を自動でセットするライブラリの開発

ルでのネットワーク資源分離を実現することが必要である．また，遠隔の計算資源に保存されているデータへの高速アクセスを実現することが必要である．

広域ネットワークを介して安全かつ高速にデータを収集するソフトウェアを開発する必要がある．特に，センサ等から送信されるデータをネットワーク上でユーザまたはアプリケーション毎に適切に分離するため，センサと計算資源間でオンデマンドに VPN を作成してデータを送信する等，安全にデータを転送することが必要である．また，5G のモバイルネットワークの性能を活用し，ネットワークの特性に応じて柔軟かつ高速にデータを収集することが必要である．

800Gbps 超の広域ネットワーク上での高速通信を実現するためスケーラビリティの高いデータ転送技術を開発する必要がある．また，高速に受信するデータを遅滞なくメモリに保存することが必要であり，ネットワークの通信性能とメモリアクセス性能をバランス良く連携させる方式の開発も必要である．

6.3　数値計算ライブラリ・アルゴリズム

6.3.1　数値計算ミドルウェア

6.3.1.1　電力制御

現在では，電力制御機構を利用した研究により，消費電力を制限しても計算性能に影響が無く，諸費電力の削減が期待できるなど知見が蓄積されている．今後は，このような知見をさらに蓄積し，その知見に基づき電力構成を最適にするような機能を持ったライブラリが必要不可欠となる．また，このような知見を貯めるためにもアプリケーション毎にシステムに最適な電力供給量を調べられる機能も必要となる．もちろん，消費電力を調整することに対応したハードウェア自体も必要である．すでに可能な CPU やメモリだけでなく，GPU などの加速器，ネットワークカード，スイッチ，ストレージと，アプリケーションによって必要な機器の電力は異なるため，それぞれの消費電力を調整できるハードウェア的機能は必須と言える．CPU であれば周波数や利用コア数を減らすなど消費電力削減には，そのハードウェア構成により手法が複数存在する．アプリケーションによっては，そのハードウェアの何が性能に効果があるのかは変わるため，単純に消費電力を減らすのでは無く，何を変更して減らすかを選択できる機能とその情報が必要となる．このような知見により，アプリケーションに最適な電力設定が自動で選択されるライブラリが次世代システムでは必要であろう．

6.3.2　数値計算・アプリケーションを支える重要技術

6.3.2.1　自動チューニング

■10 年後の想定技術　今後の計算機構成のトレンドから，以下に対応する必要がある．

- 【超非均質プロセッサ対応】：より複雑な非均質プロセッサの普及．たとえば，GPU に加えて，FPGA

や量子コンピュータがアクセラレータとして搭載される計算機環境への AT システムの対応.

- 【超低オーバヘッド電力最適化対応】: 電力測定技術の進展により実行時に判明するより多数の電力制約が判明する環境での電力最適化．たとえば，低いオーバヘッドでミリ秒単位で現在の電力が得られる，または，CPU，メモリ，ハードウェア，ネットワーク機器単位での電力が得られる環境での電力最適化技術.
- 【超高レイテンシ通信対応】: 低い通信性能の向上率を考慮した通信処理の最適化．特に通信レイテンシの大幅な向上が期待できない状況下での通信処理の自動最適化機能．または光通信技術等の新技術により，現在よりも相対的に通信バンド幅が増した計算機環境での，通信処理の自動最適化機能.
- 【AI と AT の融合】: AI 処理におけるパラメタチューニングへ AT を適用することの高い要請への技術的な回答．特に，ハイパーパラメタやネットワークアーキテクチャに起因するパラメタを自動最適化する機能を，汎用性の高い AT 技術を適用することで達成する技術．さらに大規模重回帰分析等の数値計算技術を駆使して AI の出力結果の妥当性を検証することで，AI ソフトウェアの開発工数の削減に AT 技術を適用する技術開発など.
- 【混合精度演算対応】: 半精度演算やアプリケーション特化型演算器 (GPU の Tensor コアなど) の普及による，混合精度演算を数値計算処理に適用する場合の実装方式の自動最適化機能.
- 【量子コンピュータと AT】量子コンピュータを数値計算に利用する際に現れる性能パラメタ（アルゴリズム上と量子コンピュータのハードウェア上の双方）に対する性能チューニングに AT 技術を適用する.
- 【社会システムと AT】Socity5.0 など IoT を活用した社会システム内部の性能チューニングに対し，ユーザに知られずに浸透可能な AT 技術の開発．狭義には，Socity5.0 を支える AI 処理や数値計算処理の性能向上に AT 技術を適用すること．広義には，IoT 機器，エッジ，ネットワーク，およびロボットなどの人に接する機器全てについて，実行性能や品質向上のために AT 機能が適用され，ユーザに知らずして性能向上が達成できる社会システム構築で必要とされる AT 技術の開発.

■ロードマップ 以下，自動チューニングに関するロードマップを示す.

- **2020 年〜2022 年**: AI と AT の融合，混合精度演算対応
- **2022 年〜2026 年**: 超低オーバヘッド電力最適化対応，超非均質プロセッサ対応，電力最適化対応
- **2026 年〜2028 年**: 量子コンピュータと AT
- **2028 年〜**: 社会システムと AT

6.3.2.2 耐故障（FT, FR)

■次期 Flagship システムにおける耐故障機構への要求と研究課題 本節では次期 Flagship システムにおける耐故障機構の要求と研究課題について述べる．これまでに述べてきたように，耐故障機構を実装するためには一定の冗長計算リソースと計算リソース間でデータを共用できる仕組みが必要である．例えば，現状主流である Checkpoint/Recovery 方式では，処理を継続するに必要な記憶領域と Checkpoint 処理時に最低でも MTBF 時間より高速で，かつ，必要なチェックポイントデータを 2 回分格納できるだけの領域が必要である．また，故障発生時のダメージを最小限に抑えるため計算時間に比べて十分に短く，一定時間単位でチェックポイント時間のオーバヘッドを最小限にする必要がある.

次に耐故障機構の導入が必要となったと仮定した場合，そのシステム実現への要件について議論する． 耐

故障機構への要求として以下の 3 つが上げられる．

1. アプリケーションへの導入が容易： 既存アプリケーションに対して大幅な改修が不要で最小限の追加コードや変更で導入が可能，Add-on 形式で機械的に修正可能なものが望ましい．
2. アプリケーション性能低下が極小化され許容範囲： 耐故障機構導入によるオーバヘッドやチェックポイント実行時間のオーバヘッドが許容範囲である． 例えば，1 時間毎のチェックポイントを採取する際に実行時間の 10% 程度が許容範囲と仮定すると 6 分程度でのチェックポイント採取が必要である．
3. 耐故障以外に積極的にユーザが導入したくなるものが必要：利便性の追求例として，長時間実行を前提としたタイムシェアリングとして活用，システム故障で中断したジョブは自動的に再実行処理などのジョブ実行支援機能の他，直前の状態から再実行や格納データを活用した省計算パワー計算など積極的な省リソース計算への応用も期待される．

次にこれらの耐故障機構への要求を満たすための研究課題について述べる．

アプリケーションへの導入を容易にするための課題としてはチェックポイント・リカバリ方式への課題と冗長リソース代替計算継続方式，縮退リソース計算継続方式の課題がある．それぞれについてまとめる．

- チェックポイント・リカバリ方式への課題
  - 通信ライブラリの耐故障対応： 例えば，MPI プログラムを耐故障対応にする ULFM では通信を司るコミュニケータに対して幾つかの関数を使い，故障検知，コミュニケータの修復を行うが，その処理は煩雑である．Fenix は ULFM 対応を初期化関数と MPI 関数のオーバライドにより簡易化し複雑な処理を簡略化している．
  - データ領域の耐故障対応の簡略化： プログラムを耐故障対応にするにはデータ領域のチェックポイント・リスタート対応もプログラムする必要がある．Fenix では，チェックポイントするデータ領域の登録とポリシー設定で半自動化している．
  - 既存アプリケーションへの修正極小化への課題
    - 計算リソースの概念化： 定型処理のライブラリ化，標準化
    - 非定型処理記述
- 冗長リソース代替計算継続方式，縮退リソース計算継続方式の課題
  - 通信ライブラリの耐故障対応：チェックポイント・リスタート方式で対応可能，ただし，縮退リソース計算継続方式はプロセスの再配置が課題
  - 故障プロセスデータの引継ぎ手法確立：チェックポイント・リスタート方式で対応可能，ただし，縮退リソース計算継続方式はプロセスデータの再配置が課題
  - 既存アプリケーションへの修正極小化：チェックポイント・リスタート方式で対応可能

アプリケーション性能低下が極小化され許容範囲に抑制するための課題としては，Checkpoint/Recovery: オーバーヘッド極小化，間隔等の自動化，Reconfiguration: 再実行時のデータ配置バランス確保がある．

システム信頼性が悪くない場合の導入障壁低減を実現するための課題としては，必要悪ではない積極的利用用途を考える必要があり，洗練された王道スタイルを醸成するためには，Template による簡単導入・簡単に On/Off 可能にしていくなどの工夫が必要である．

■耐故障実現に向けた研究の方向性　本節では耐故障実現に向けた研究の方向性について述べる．アプリケーション・数値計算ライブラリレベルでの耐故障を中心に考えてきたが，アプリケーション適応コストの極小化

と実行性能低下防止を実現するためには，ある程度のアプリケーションとシステムソフトウェアとの協調が重要である．このためには，アプリケーションへの適応性を考慮しながら，幾つかの実行スタイルに分類し，その実行スタイル毎に一つの王道的な記述パターンを追求するのが重要である．これまでに述べてきたが，耐故障対応を実質的に推進するためには，単なる必要悪として耐故障対応だけでなく，システム運用を合わせたアプリケーション実行環境としての利便性の追求が重要である．このため，システム運用観点では耐故障機構を活用した長時間ジョブの時分割実行への応用やアプリケーションユーザのアプリケーション実行時間を節約する意味でのチェックポイントデータを活用した異なるアルゴリズムやプログラム実行を可能とする計算手法の探求が重要である．

■耐故障を支える技術に関するロードマップ　以下，耐故障を支える技術に関するロードマップを示す．

- **2020 年〜2022 年**:
  - ユーザ利便性を追求した耐故障実現手法の確立
  - ユーザ利便性を追求した耐故障機構のシステム運用方式確立
  - アプリケーションタイプ毎の耐故障実現手法の確立
- **2022 年〜2025 年**:
  - アプリケーションーシステム協調型耐故障保証手法確立
  - 利便性追求型耐故障機構のシステムソフトウェア実現手法確立
  - アプリケーションタイプ毎の耐故障実現手法の実アプリケーション適応
- **2026 年〜2028 年**:
  - ジョブスケジューラ連携の利便性追求型耐故障実行環境構実現
  - 王道アプリケーション構成・記述パターンの確立
- **2028 年〜**:
  - 利便性追求型耐故障実行環境の実システム運用

6.3.2.3　機械学習フレームワーク

■今後の動向　ハードウェアやアルゴリズムの多様化が進んでおり，ハードウェアやアルゴリズムの変化への追随や，汎用性と効率化のトレードオフなどを考慮した DL アクセラレータ設計が進んでいくと考えられる．それに伴って AutoTVM，AutoVTA，AutoML といったライブラリ・フレームワークの発展が進み，それによって抽象化された設計思想による最適化/自動化がさらに進んでいくと考えられる．今後登場する多種多様な AI アクセラレータに対して性能を引き出すためにフレームワーク側での対応も進むと考えられる．一方で，機械学習フレームワーク自体はここ数年は主に PyTorch と TensorFlow への収束に向かっており，フレームワーク自体にはそれ程変化がない可能性がある．

機械学習の方向性としては，マルチタスク学習を用いたビッグデータを用いた広域的な汎用モデルの生成や，転移学習を用いた個々の用途に応じた局所用モデル生成といった発展が進むと考えられる．スーパーコンピュータ等の大規模計算環境においては，これらの学習において計算機性能によって差別化をすることが重要であると考えられる．例えば，スーパーコンピュータなどの大規模な計算機の上でビッグデータを用いて汎用的なモデルを学習し，学習済みのモデルを用いてオンプレミスの計算機で転移学習を行うといった利用方法が考えられる．

シミュレーションとの融合が進むにつれて，シミュレーションと機械学習フレームワークの統合も進むと考

えられる．機械学習フレームワークがシミュレーションを実行しながら学習するような機能を搭載することや，各種シミュレーションを共通フォーマット化して機械学習の I/O に利用するなどが考えられる．NVIDIA ISSAC Gym のような 3D シミュレーション環境を用いて，強化学習などの機械学習を高速化する手法がさらに発展すると考えられる．

機械学習の性能評価では，コアやメモリの性能に起因した学習時のスループット性能や推論時のレイテンシ性能だけでなく，認識精度も重要な評価対象となる．認識精度に影響する機械学習のモデル選択やパラメータのチューニングは計算能力を対価に自動化を行う事が可能であり，より効率的なオートチューニングは今後も発展していくと考えられる．一方で，認識精度を向上させるために多様，且つ，大量のデータセットを必要とする点は普遍的であり，今後の機械学習技術の普及と求められる認識精度の向上から，より大規模な機械学習用のデータベースが必要とされ，これらのデータベースを中心とした機械学習のプラットフォームの展開が一般化される．近年ではパブリックな機械学習用のデータベースを容易に利用できるサービスが普及しており，これらのデータベースを抱える Amazon，Microsoft，Google では，それぞれが提供するクラウドサービスと連携して利用できるようになっている．また，エンドユーザに向けたより抽象化したインタフェースや開発環境を用いたクラウドサービスの開発も先行しており，機械学習の分野ではクラウド基盤を中心としたエコシステムが形成されていくと考えられる．

多くのエンドユーザはより抽象化されたインタフェースや開発環境を介して機械学習をクラウドサービスの一環として利用する事になる．一例として Amazon SageMaker や Google Colab などがあげられ，動的に拡張されていくデータベースとは容易に連携し，機械学習開発時にフレームワークレベルでの細かい設定等は隠蔽，あるいは自動化されており，専門的な知識を持たない多くのユーザ層を対象としたプラットフォームが商業的に普及すると考えられる．以上より，大規模な計算基盤で機械学習をアプリケーションとして取り扱う場合はクラウドとの連携が不可欠となる．外部のデータベースに高速にアクセスできる伝送路やキャッシュするためのファイルシステムだけでなく，クラウドのポータル側から計算基盤へインスタンスを生成して実行可能となるようなインタフェースの連携等が必要となる．これらのエンドユーザとは別に，各種フレームワークを直接利用する開発者向けのシステムも併用しておく必要があり，多様化するユーザの用途に合わせた対応が必要とされる．

■機械学習フレームワークに関するロードマップ

- **2020 年〜2021 年**:
  - 現アーキ向け各種フレームワーク改良・ライブラリ高速化
  - 現アーキでのアプリ性能特性分析と性能ボトルネック特定
- **2022 年〜2025 年**:
  - 外部データベース・クラウドインタフェースとの連携，外部計算資源との連携（転移学習，マルチタスク学習），シミュレーションとの連携
  - アプリ性能特性を考慮した新アーキ検討
- **2026 年〜2027 年**:
  - 外部データベースを活用したオートチューニング機能開発
  - 新アーキ向けフレームワーク改良・ライブラリ高速化
- **2028 年〜**:
  - 新アーキでの各種フレームワーク・ライブラリの実システム運用
  - 外部データベース連携機能の実システム運用

6.3.2.4 アルゴリズム

■グラフ処理 ものやデータの増加に伴い，多様な分野において今後さらに巨大なグラフが登場し，より効果的に処理することが求められると想定される．結果として，従来より大規模なグラフに関しての処理を可能にする基盤の必要性がますます重要となると考えられる．グラフ処理においては大量のデータのやりとりが行われるため，ノード間の通信性能やメモリ容量・帯域・レイテンシが重要となる．予測される計算機システムにおいて，CPU やアクセラレータとのバランスを考慮したアルゴリズム設計がより必要になると考えられる．

より大規模な並列化が必要となり，かつ CPU だけでなくアクセラレータの使用等も踏まえた場合，動作させるだけで困難となる．より使い勝手のよい形でのフレームワークの提供が重要であるのは変わらないと考えられる．加えて，Python などのスクリプト言語でのグラフ解析への需要も考える必要があると考えられる．

従来のグラフ処理アルゴリズムへの大規模並列化の検討だけでなく，より幅広い範囲のアルゴリズムの検討が必要となる．以下，概要をまとめる．

- 動的に変化するグラフ:
  動的に頂点や辺の追加や削除が行われるようなグラフを考える必要がある．この場合，バッチ処理とストリーム処理の双方を考慮する必要がある．また，変更箇所が一部である場合には，解析のためのアルゴリズムをグラフ全体で行う必要はなく，差分のみを適切に扱うことで同等の結果を得ることが可能となる．動的に変化する大規模なグラフに適したアルゴリズムを理論側と実装側で協調して考える必要がある．
- 複雑なグラフの扱い（Property Graph）:
  頂点や辺が単一ではなく複数の情報を持つようなグラフを扱う頻度が増すと考えられる．この場合，グラフを解析するためのアルゴリズムが異なってくるだけでなく，データ構造の見直しが必要になる．また，グラフの規模にかかわらず property の数が多い場合には結果的にメモリの使用量や計算量の増加ということもあり，従来のグラフ処理とは異なるボトルネックが発生すると考えられる．
- Graph Neural Network:
  グラフ構造を持ったデータに対して，機械学習を行うという取り組みが 2010 年代後半に新たに登場し着目を集めている．画像などの規則正しい構造を持ったものを扱うだけでなく，より一般化したグラフ構造を持ったデータを扱うことが可能となる．従来のグラフ処理や機械学習処理とも異なり，それぞれの特徴を考えた設計が必要となる．理論面を含め，大規模な状況におけるアルゴリズムの設計から開始していく必要がある分野である．

また，グラフ処理に特化した新アーキテクチャ（Intel PUMA Graph Processor など）に合わせたアルゴリズムの開発とプログラミング環境等が必要になることが想定される．アーキテクチャ側と一体化した形でのソフトウェアやアルゴリズムの研究が必要である．

以下，グラフ処理に関するロードマップを示す．

- **2020 年〜2021 年**: 富岳クラスでの高速なグラフ処理手法の拡充，多様なグラフ処理アルゴリズムに関する基礎研究
- **2022 年〜2025 年**: 大規模並列環境下でのフレームワークやライブラリの拡充
- **2026 年〜2027 年**: 新アーキ向けの開発や実装
- **2028 年〜**: 幅広いグラフアルゴリズムについて，各アーキテクチャにおいて高速な処理を可能とする

フレームワークやライブラリの開発

6.3.2.5 量子コンピューティング

■アプリケーション，適用分野 ゲート型量子コンピュータについては，NISQ 量子コンピュータであることが予想されることから，量子コンピュータ単体での利用ではなく現在と同じく量子・古典ハイブリッドアルゴリズムによる利用がほとんどであると予想される．このような量子・古典ハイブリッドアルゴリズムにおいて量子コンピュータは一種のアクセラレータとして利用される．特に，計算規模に対し古典計算機では指数関数的に計算量が増大するが量子コンピュータでは多項式時間で結果を得られる計算に効果的である．現在のゲート型量子コンピュータと比較して実装される量子ビット数は飛躍的に増加することが期待されるため，古典コンピュータ上では実行が現実的に困難な問題への適用がより現実的なものとなることが期待される (例えば，強相関量子系の理論研究，量子化学計算における多軌道の Full-CI 計算，高分子のタンパク質を分子動力学法で取り扱う創薬の研究など)．同様に量子機械学習の発展に伴って，現在 AI により発展が期待されている応用技術（材料開発，医療，金融など）により量子コンピュータを活用した事例が発現してくることが期待される．アニーリング型量子コンピュータ (イジングマシン) についてはより大規模，かつより多くの条件を含んだ組み合わせ最適化問題に利用が可能になるため，特に一般企業においての利用が加速すると予想される．

ゲート式量子コンピュータに関しては QAOA や VQE などの既存のアプリケーションでの利用もさることながら，NISQ 量子コンピュータ利用におけるキラーアプリケーションの開拓は非常に重要であると考えられる．そのために，他分野と連携した活動が求められる．アニーリング式量子コンピュータに関しては，Ising 型の組み合わせ最適化問題に落とし込める実応用問題（最近で言えば新型コロナ関連がまさにそれであるが）に目敏く反応して，迅速な調査と情報発信を継続して行うことで，一般企業からの利用需要を高めることが肝要．量子暗号化技術など量子コンピュータのみで完結するアプリケーションが実現することが期待される．また，量子位相推定など NISQ 量子コンピュータでは取り扱いが困難な量子アルゴリズムについてもその取扱いが可能となり，量子多体系の理論的研究手法としてより一般的になることが期待される．また，量子暗号の堅牢性から一般企業も含めたより広範な利用も期待される．

■従来型計算機とのハイブリッド利用 現在のゲート式量子コンピュータの利用はその量子回路の設計から，つまりアセンブリ言語を取り扱うことと同等のプログラミングが必要であることに対し，量子コンピュータを利用する部分について，特に汎用的に利用できる量子アルゴリズムについてはそのライブラリ化が進むことが予想される．量子・古典ハイブリッドアルゴリズムにおける量子コンピュータのアクセラレータ的な利用がより容易になることが期待される．

アニーリング式量子コンピュータでは現在人的作業がある程度必要となっている 1）解決したい実応用問題を Ising 型の最適化問題に落とし込む作業，並びに 2）落とし込まれた最適化問題を実機の量子ビット間結合のグラフにマッチするように書き換える作業，が量子ビット数の増大に伴って効率よく古典計算機上で自動実行されることが期待される．ゲート式量子コンピュータと従来計算機とのハイブリッド利用においては，量子ビット数の増加とともに取り扱う情報量の増加がそれに伴って増加することが予想される．また，例えば機械学習での利用などにおいて取り扱う情報量もそれに伴って巨大となることが予想されることから，量子コンピュータ利用のセットアップから結果の処理まで HPC の利用が不可欠になると考えられる．量子コンピュータと HPC を広帯域の専用回線で接続することなどが必要になると考えられる．

アニーリング式量子コンピュータに関しては，量子ビット数の増加に伴って前述の実応用問題の Ising 型への最適化問題の落とし込み作業＆実機の利用に即した最適化問題の書き換え処理を高速に行うためのアルゴリ

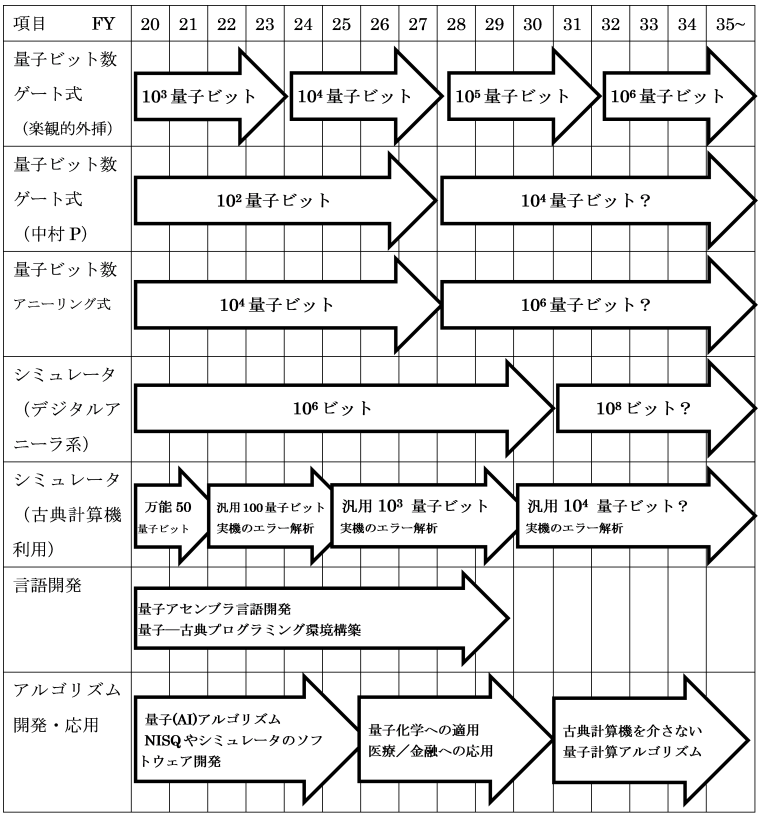

図 6.1　今後の技術動向に関するロードマップ

ズムが特に必要．量子・古典ハイブリッドアルゴリズムにおけるアクセラレータとしての役割は規模や安定性の立場からより有効なものとなることが期待される．量子計算機とやり取りをする古典的情報量に合わせて古典計算機が担う演算量が速やかに増加するアルゴリズムを実施する場合においては，巨大な問題規模を取り扱う立場からも HPC と密に連携したシステムの構築が望まれる．現状，超伝導素子を利用する量子コンピュータは冷凍機内に設置されているが，この構造は 2030 年頃でも変わらないと予想される．冷凍機内では，一般に信号が微弱であるために，量子コンピュータと外側にあるスーパーコンピュータなどを高速に接続することが難しい．そのため，ハイブリッド利用の際の利用形態を含めて今後検討していくことが必要となる．

参考文献

[83] Reid Priedhorsky Alfred Torrez and Timothy Randles. “HPC container runtime performance overhead: At first order, there is none”. In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC ’19)*. 2019, poster.

[84] A. J. Younge et al. “A Tale of Two Systems: Using Containers to Deploy HPC Applications on Supercomputers and Clouds”. In: *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 2017, pp. 74–81.

[85] Stefan Lankes, Simon Pickartz, and Jens Breitbart. "HermitCore: A Unikernel for Extreme Scale Computing". In: *Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers*. ROSS '16. Kyoto, Japan: Association for Computing Machinery, 2016. ISBN: 9781450343879. DOI: 10.1145/2931088.2931093.

[123] P. Balaprakash et al. "Autotuning in High-Performance Computing Applications". In: *Proceedings of the IEEE* 106.11 (2018), pp. 2068–2083.

[221] *International Technology Roadmap for Semiconductors 2013 Edition*.

[222] Nicholas Chaimov et al. "Scaling Spark on HPC Systems". In: *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*. HPDC '16. Kyoto, Japan: Association for Computing Machinery, 2016, pp. 97–110. ISBN: 9781450343145. DOI: 10.1145/2907294.2907310.

[223] Wolfgang Frings et al. "Massively Parallel Loading". In: *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*. ICS '13. Eugene, Oregon, USA: Association for Computing Machinery, 2013, pp. 389–398. ISBN: 9781450321303. DOI: 10.1145/2464996.2465020.

[224] Yanping Huang et al. "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism". In: *CoRR* abs/1811.06965 (2018). arXiv: 1811.06965.

第 7 章

おわりに

本書は，今後の高性能計算機の構築・運用に向けた技術課題とどのような研究開発が必要となるかを，主としてデバイス・アーキテクチャ，システムソフトウェア，数値計算ライブラリ・アルゴリズム，システム運用の面から検討し，それを報告書としてまとめたものである．本書での重要項目を以下にまとめる．

- 幅広いアプリで利用可能なアクセレレータアーキテクチャやその演算器構成の検討が必要
- 多くのアプリケーションが高バンド幅および大容量のメモリを要求しており，データ移動を高効率に行えるアーキテクチャ，メモリ技術，ソフトウェア最適化がさらに重要になる
- Society5.0 へのさらなる貢献のための次世代運用技術に関しても検討が必要
- アプリケーション・システムの協調設計（コデザイン）がこれまで以上に求められる
- アプリ要求の精緻化と重点課題アプリや今後の Society5.0 関連アプリケーションを含むより多くのアプリケーションでの分析が今後必要になる

なお，検討は最初のターンを終えたばかりであり，今後は本報告書を基にして他の多くの研究者やスーパーコンピュータのユーザ，またアプリケーション分野の研究者の方々とさらなる検討・議論を重ねる必要がある．本報告書はその都度更新していくことで 2028 年頃という次の開発の区切りのためだけでなく，継続的な議論結果を反映した報告書となるように今後 NGACI の活動を続ける予定である．

最後に，本書の執筆にご尽力頂いた執筆者の方，議論に参加頂いた方，ご助言・ご意見を頂いた方に感謝の言葉を述べ，本書のまとめとしたい．