次世代先端的計算基盤への期待と課題

理化学研究所 計算科学研究センター
プロセッサ研究チーム チームリーダー
佐野 健太郎

2022/9/16

本講演

- 自己紹介
- NGACI White Paper ver 1.0 高性能計算機アーキテクチャの技術動向
- NGACI White Paper ver 1.1 研究開発の方向性と課題
- 汎用CPUはスケールするか？
- 将来のアーキテクチャの可能性
- 将来アーキテクチャの課題
- おわりに

自己紹介：佐野 健太郎

- **専門：** 計算機アーキテクチャ、リコンフィギャラブル計算機システム
- **経歴とこれまでの研究**
  - ✓ 2000　博士（情報科学）　東北大学大学院情報科学研究科
  - ✓ 2000～2001　助手　東北大学大学院工学研究科
    2001～2005　助手　東北大学大学院情報科学研究科
  - ✓ 2005～2018　助教授（准教授）　東北大学大学院情報科学研究科
  - ✓ 2006～2007　在外研究員　インペリアルカレッジ・ロンドン
    HPCI 技術ロードマップ白書 アーキ執筆担当「アクセラレータ」
  - ✓ 2017～2018　チームリーダ兼務　理化学研究所計算科学研究機構
  - ✓ 2018～現在　チームリーダ　理化学研究所計算科学研究センター
  - ✓ 2019～現在　客員教授　東北大学大学院情報科学研究科
    NGACI (Next-Generation Advanced Computing Infrastructure) 白書 アーキ取りまとめ

- 3次元データ可視化の精度制御方式
- 可視化の並列アルゴリズム

- ベクトル量子化の並列アルゴリズム、FPGAによる専用ハードウェア

- シストリック計算メモリアレイ
- ストリーム計算用超長パイプライン
- データフローハードウェアコンパイラ
- 高性能計算向けFPGAクラスタ
- FPGAによる流体計算専用計算機
  FPGAによる津波計算専用計算機
  FPGAによる多体問題専用計算機
- ストリームデータ圧縮ハードウェア

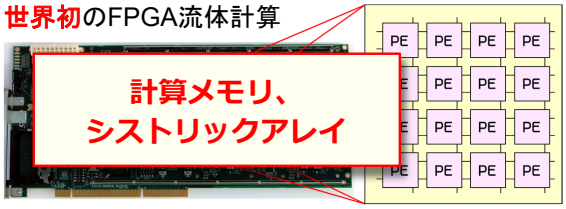
- **FPGAによる富岳機能拡張システムとそのアプリケーション**
- **粗粒度再構成可能プロセッサ(CGRA)**
- **次世代高性能計算機システム**

これまでの研究（2006年～）

命令実行型の「汎用プロセッサ」に代わる、特化型計算機アーキテクチャ

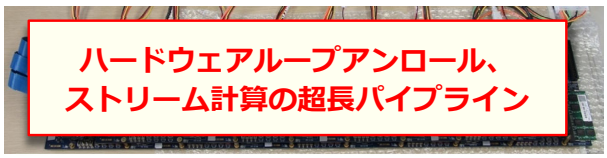
シストリック計算メモリアーキテクチャ（2006~2011）

- ✓ 流体, 電磁場, 伝熱問題のステンシル計算。自作FP演算器
- ✓ 世界初のFPGA流体計算



スケーラブルパイプラインアーキテクチャ（2008~2012）

- ✓ 複数FPGAによる超長パイプラインにより帯域一定で性能向上（260GF @1GB/s）



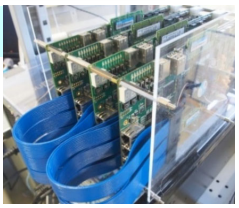
データ圧縮によるデータ帯域向上ハードウェア（2009~）

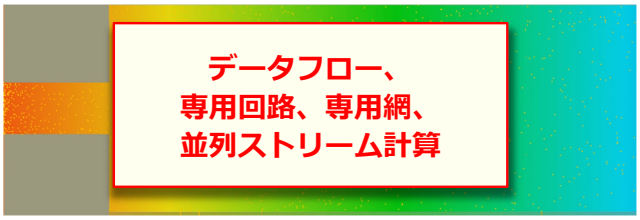
- ✓ PDEの解などの連続性を持つ浮動小数点データ向け
- ✓ 計[illegible]ト

データの冗長性排除・専用ハード処理

流体計算専用計算機（2012~2017）

- ✓ 複数FPGAによる高スケーラビリティ並列計算
- ✓ 実用問題のストリーム計算をFPGA上に専用ハードウェア化





これまでの研究（2015年～）

実用化に向けたシステム研究（ソフト・ハード）

後の実験システム
ESSPERの原型

密結合FPGAクラスタ

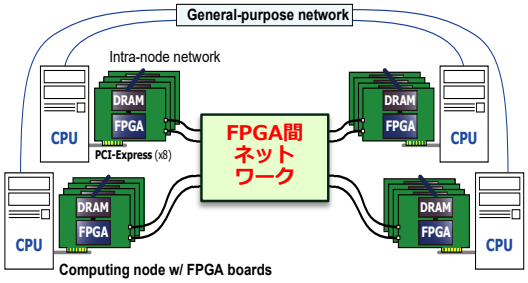
- ✓ データフロー計算をFPGAにオフロード
- ✓ FPGA間専用ネットワーク

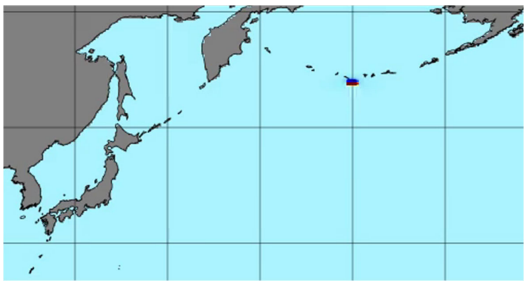
ストリーム計算回路コンパイラ

- ✓ Stream Processor Generator
- ✓ DFG記述からデータフローによるストリーム計算回路を自動生成

数値計算のアクセラレーション

- ✓ 津波シミュレーション専用計算回路
- ✓ 流体シミュレーション専用計算回路
- ✓ 多体問題専用計算回路






Hot News
FPGAが性能でついにGPU超え
2万段超のパイプラインで実現
外部メモリーとのデータのやり取りを大幅に削減
nkbp.jp/NE1606021

津波計算：GPU比で
実効性能 2倍
電力性能比 10倍

IEEE, ACM, Elsevier,
IEICE 論文誌

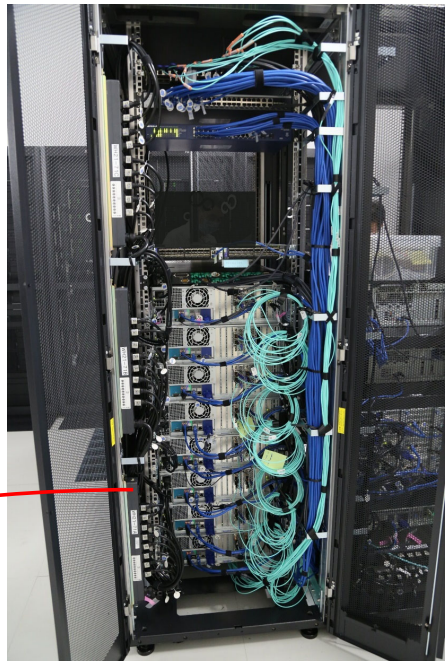
ESSPER
Elastic and Scalable System for High-Performance Re-configurable Computing

FPGAによる富岳の機能拡張 実験システム

FPGAクラスタ
実験試作機
ESSPER (別室)

「拡張により富岳に付加価値を与える方式の検討課題」

(FY2017～2020)



NGACI White Paper ver 1.0
高性能計算機アーキテクチャの技術動向

NGACI (Next-Generation Advanced Computing Infrastructure)

コミュニティにおいてオープンな意見交換を通じ、将来の高性能計算環境や共用資源の技術的課題、必要な研究開発を白書にまとめる活動

- ✓ ４つのWGにより将来のシステム像や課題を議論
- ✓ White paper 1.0.0　https://sites.google.com/view/ngaci/home
- ✓ 改訂版に向け議論を継続中

執筆協力者 (ver 1.0.0) > ver 1.1.0公開に向け準備中（ドラフトは完成）

- ✓ **取りまとめ　近藤（慶應大・理研）**
- ✓ **アーキテクチャWG　リーダ：三輪（電通大）、佐野（理研）、谷本（九大）**

  安島（富士通），井口（北陸先端大），井上（九大），江川（電機大），岡本（Spin Memory） 小野（九大），鯉渕（NII），小林（筑波大），小松（東北大），佐藤（東北大），佐藤（豊橋技科大），塩見（京大），田邊（東大），中里（会津大），吉川（富士通研），福本（富士通研），星（NEC），三好（わさらぼ），宮島（理研）
- ✓ **システムソフトWG　リーダ：佐藤（理研），佐藤（豊橋技科大）**

  合田（NII），小柴（理研），小松（東北大），坂本（東大），高野（産総研），滝沢（東北大），辻（理研），中島（富士通研），深井（理研），山本（理研），和田（明星大）
- ✓ **アプリ・ライブラリ・アルゴWG　リーダ：深沢（京大），今村（理研），中島（東大・理研）**

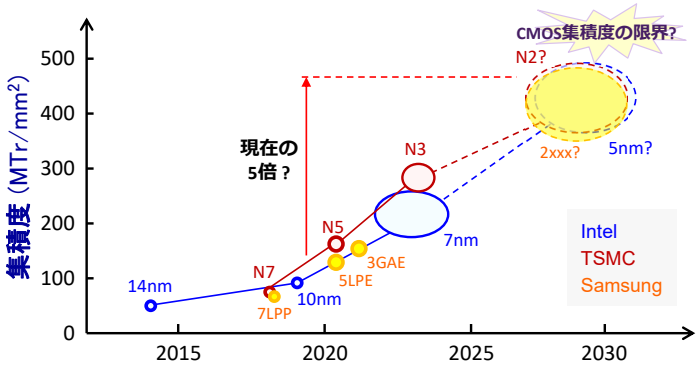
  岩下（北大），小野（九大），笠置（富士通研），片桐（名大），白幡（富士通研），住元（富士通研），高橋（筑波大），寺尾（理研），長坂（富士通研），椋木（理研），村上（都立大）
- ✓ **システム運用WG　リーダ：塙（東大），野村（東工大）**

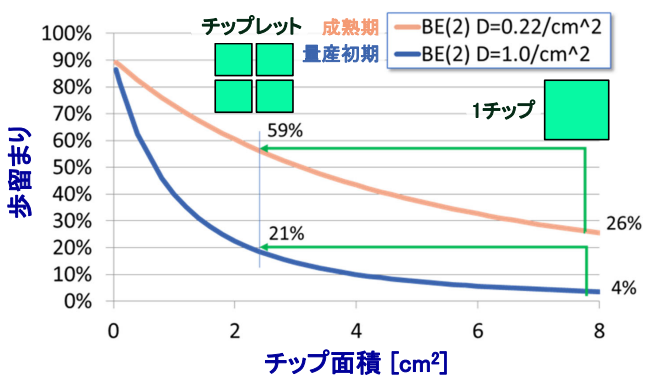
  大島（名大），實本（理研），庄司（理研），滝澤（産総研），竹房（NII），藤原（NII），三浦(理研)

NGACI WP アーキ関連部分 (ver 1.0)

ハードウェア技術の動向

- **CMOSデバイス・先進パッケージ技術**
  - ✓ 集積度は向上するも限界に近づく (FinFET → GAA FET)
  - ✓ トランジスタ単価は下がらず
  - ✓ 歩留まり向上によるコスパ改善 → チップレット
  - ✓ ダイ間接続技術、MCP技術の進展
    （アーキテクチャ制約条件の変化）
- **プロセッサ・メモリ（HPC向け）**
  - ✓ メニーコアCPU（Intel, AMD, IBM, Fujitsu, etc.）
  - ✓ GPU（NVidia, AMD, Intel）, ベクトル型プロセッサ（NEC）
  - ✓ メモリ：新デバイス研究が進むも既存技術の延長が主流（DDR, HBM）
- **データ転送技術**
  - ✓ 高速シリアル伝送（~400G）, パッケージ内ダイ間（Si-IP, EMIB, UCIe）
    ノード内接続（メモリ共有向け: QPI, IFIS, CXL、I/O向け: PCIe）
- **ASIC/FPGA**
  - ✓ ASIC　TPU等：様々なカスタム・AIプロセッサが登場
  - ✓ FPGA　XILINX, Intelの高性能FPGA：HPCやデータセンター事例が増
- **その他**　In-Memory computing技術（研究開発は続くが普及はまだ）



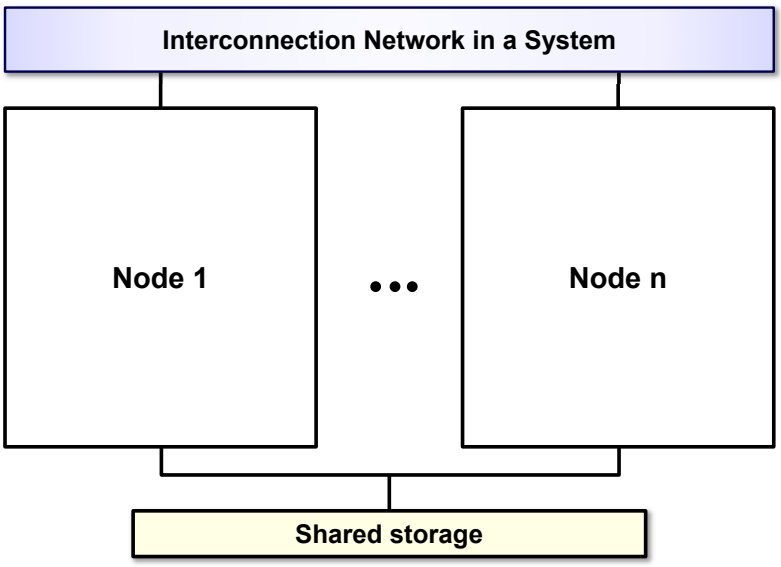


https://community.arm.com/arm-research/b/articles/posts/three-dimensions-in-3dic-part-1

システムアーキテクチャの技術動向 (v1.0)

HPCアーキテクチャ：
分散メモリ型超並列計算機

- ✓ A lot of "computing nodes" which are connected by Network
- ✓ No shared memory among nodes
- ✓ Nodes have a shared storage (shared disks and/or SSDs)



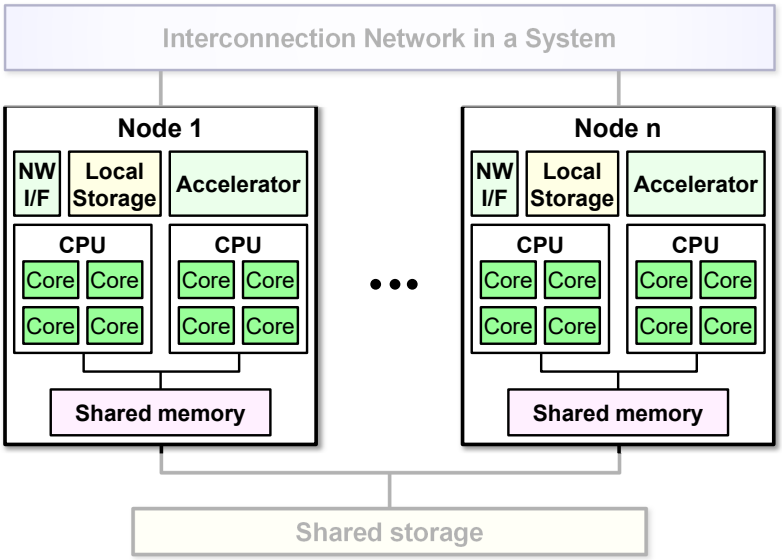
Distributed-memory parallel computer
Each node/processor has its own local memory with its own memory space.

Shared-memory parallel computer
Nodes/processors have a common memory space used by all of them.

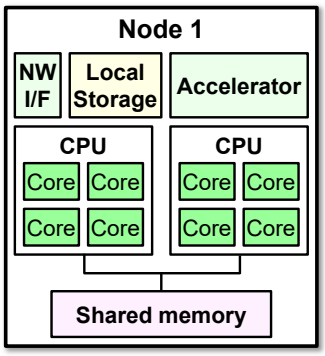
ノードアーキテクチャ

分散メモリ型 並列計算機

- ✓ Many-core CPUs with their shared memory
- ✓ Accelerators in some cases (typically GPUs)
- ✓ Other peripherals, such like network interface and local storage



システムアーキテクチャの技術動向 (v1.0)



- **ノードアーキテクチャ（計算ノードのハードウェア構成）**
  - ✓ メニーコアCPU 型
  - ✓ メニーコアCPU & GPU 混載型
  - ✓ その他：FPGA搭載型、ベクトルプロセッサ搭載型
- **インターコネクト**
  - ✓ 現状のトレンドの発展的な継続、Co-packaged Opticsを用いる新たな構成、の両方
  - ✓ ボード間結合網（Top500マシンのトレンド：Fat tree, dragonfly, 5D Torus）
  - ✓ 目的特化型結合網（Anton-s, MDGRAPE-4）
- **ストレージ（大規模化・複雑化・階層化）**
  - ✓ ストレージクラスメモリ、バーストバッファ、並列分散ファイルシステム、アーカイブストレージ
  - ✓ ストレージインタフェース（SATA, SAS, NVMe, Fiber Channel, USB）
  - ✓ ストレージネットワーク（NVMe-oF, iSCSI, FCoE）
  - ✓ ストレージ機能支援ハード（Computational Storage）

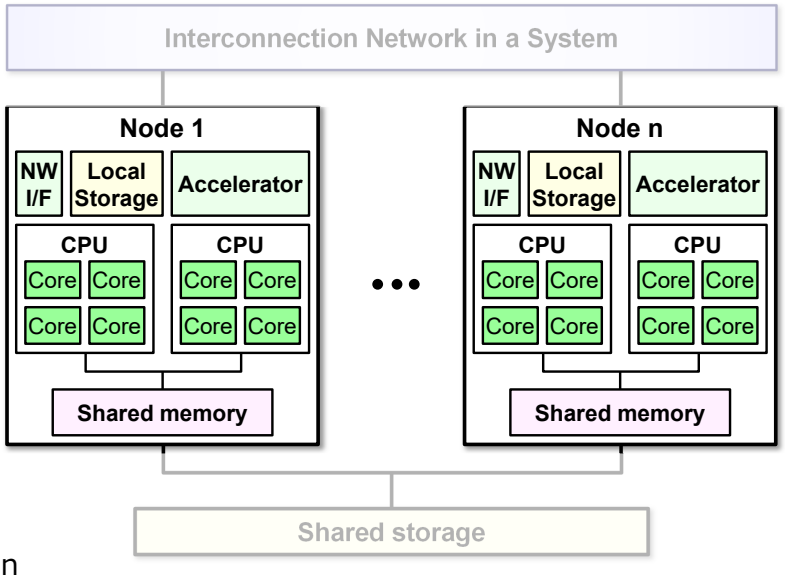
システム性能（ピーク、実効）

Peak performance and application performance

(system peak performance)
= (peak perf of node) x (# of nodes)

(application performance)
= (peak performance) x (efficiency)

✓ Efficiency is influenced by
- ➢ parallelism
- ➢ memory perf for required access
- ➢ network perf for required communication



2028年頃に実現可能な次世代システムの予測 (v1.0)

- **次世代システムの構成としていくつかのアーキテクチャタイプを検討**
- **汎用システム型**
  - ✓ **メニーコアCPU型**　富岳の構成の延長として考えられるシステム
  - ✓ **メニーコアCPU＆GPU混載型**　GPUとホストCPUで構成（現在多くのシステムでも採用）
  - ✓ **ベクトルプロセッサ混載型**　ベクトルプロセッサとホストCPUで構成（例： SX-Aurora TSUBASA ）
- **専用システム混載型（ムーアの法則減速により重要な検討事項に）**
  - ✓ **CPU拡張型**
    - ISA（SIMD）の専用的な命令をCPUに拡張機能として搭載（例：Intel AMXやARM SVEのFMMLAなど）
    - BFloat16やINT8、INT4などの応用に特化したデータ型の導入
  - ✓ **アクセラレータ主体型/ヘテロジニアス型**
    - システム搭載方式：チップ内拡張(SoCやMCM)、ノード内拡張、ラック間疎結合
    - アクセラレータ構成方式：専用、準専用、汎用
  - ✓ **Processing-In-memory主体型**
    - 演算器とメモリの密接実装によるメモリアクセスの高バンド幅化と低遅延化

IRDS: IEEE International Roadmap for Devices and Systems

汎用システム型の性能予測 (v1.0)

- **ベースラインとしての予測**
- **システムコンポーネント毎に以下の文献データから予測**
  - ✓ **プロセッサ**：　**IRDS** Roadmap - Systems and Architectures (2017 and 2020 edition)
    - ソケットあたりコア数:　70コア, SIMDビット長：2048-bit x 2, クロック周波数：3.9GHz
    - CPUソケットのTDP:　351W
  - ✓ **GPU**
    - 保守的な予測：　NVIDIA社の過去のハイエンドGPUの性能をもとに線形で外挿
    - 積極的な予測：　将来のCPUの性能予測値に現行のGPU/CPUの性能比を乗じることで予測
  - ✓ **ネットワーク**：　“Ethernet Alliance Roadmap 2018”
    - リンクあたり1.6 TByte （100Gbps x 16レーン）
    - ノードあたり1リンク （リンク数増加によってアプリのカバー範囲が変わらないため）
  - ✓ **ストレージ**：　“Lustre: The Next 20 Years”, HPC-IODC Workshop, 2019.
    - LustreでI/O性能が1.36x/年、容量が1.38x/年で向上するとの予想を利用
- **制約：システム全体の電力**
  - ✓ **システム電力バジェット：30, 40, 50MW** （cf. 富岳では28.3MW） および **PUE=1.1** (Power Usage Effectiveness)
  - ✓ **CPU（またはGPU）の電力バジェットの比率： 60, 70, 80%**

2028年のシステム予測性能（ピーク性能）（v1.0）

メニーコア型

- ✓ 最も積極的な予測でも**最大1.8 EFLOPS**
  (**富岳の性能の3.37倍** @ 50MW, CPU 80%)
- ✓ 制約は、電力 (電力性能比の向上は高々2倍程度)

システム電力 / CPU電力	30MW			40MW			50MW			28.3MW
	60%	70%	80%	60%	70%	80%	60%	70%	80%	
ソケット数	46620	54390	62160	62160	72520	82880	77700	90650	103600	158,976
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	8.3×10^6
PFLOPS	815	950	1086	1086	1267	1448	1358	1584	1810	537
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	163
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	4.85
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	0.33
総 I/O 性能 (TB/s)	34	34	34	34	34	34	34	34	34	富岳

GPU混載型

- ✓ 最も積極的な予測で**最大14.0 EFLOPS**
  (**富岳の性能の33.5倍** @ 50MW, CPU&GPU 80%)

積極的予測：CPUの性能予測値に現行のGPU/CPU性能比を乗じて予測

システム電力 / CPU/GPU電力	30MW			40MW			50MW		
	60%	70%	80%	60%	70%	80%	60%	70%	80%
GPU 数	50661	59104	67548	67548	78806	90064	84435	98508	112580
総コア数	3.4×10^9	3.9×10^9	4.5×10^9	4.5×10^9	5.2×10^9	6.0×10^9	5.6×10^9	6.5×10^9	7.5×10^9
PFLOPS	8083	9431	10778	10778	12574	14371	13472	15718	17963
HBM 総 BW (PB/s)	334	390	445	445	520	594	557	650	743
HBM 総容量 (PB)	4	5	6	6	7	8	8	9	10

- IRDS Roadmap のCPUコア数は保守的過ぎるかもしれない
- あくまでピーク性能の予測であり、実効性能はまた別
- 日本に限らない話
  既存方式に限界 → 新方式の必要性（チャンス到来！？）

アプリケーションの要求性能分析

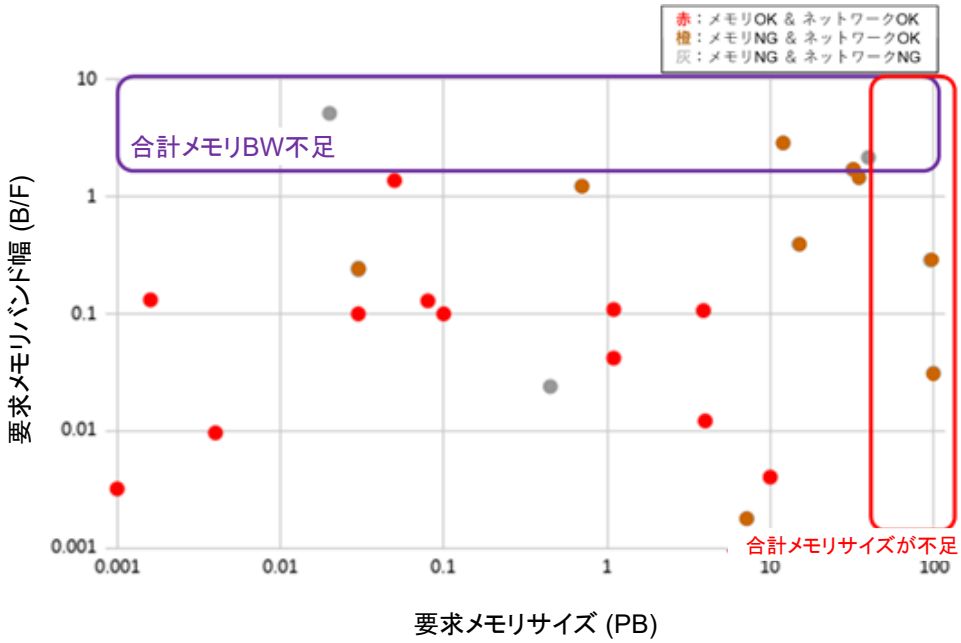
- 計算科学ロードマップやアンケートに基づき37個のアプリの要求性能を解析
  - そのうちノード数の要求について記載のないものは除く
  - より多くのアプリ（重点課題やビッグデータアプリ）の分析は今後の課題
- 分析の目的
  - 性能要求の分析により必要なシステムのタイプを分類
  - 汎用的なシステム構成によりどの程度のアプリケーションがカバーできるかの調査
- 分析の際に仮定するシステム構成（メニーコア型・GPU混載型の積極的な予測の場合）

	Manycore (50MW, CPU80%)	GPU (50MW, CPU80%)
# of CPU Sockets or GPUs	103,600	112,580
# of total cores	7,252,000	1.2 x 10^9
PFLOPS (double)	1,810	17,963
DDR total BW (PB/s)	228	—
HBM total BW (PB/s)	683	743
Total Size of DDR (PB)	39	—
Total Size of HBM (PB)	9	10

近藤 正章, “次世代先端的計算基盤の開発に向けたNGACIでの取り組み”, 第11回JCAHPCセミナ, 2022年5月.

アプリケーションの要求性能との比較（WP1.0.0より）

- メニーコア型システム構成（システム電力50MW, CPU80%）の場合



近藤 正章, “次世代先端的計算基盤の開発に向けたNGACIでの取り組み”, 第11回JCAHPCセミナ, 2022年5月.

NGACI White Paper ver 1.1
高性能計算機アーキテクチャの技術動向

2028年のシステムの予測性能（更新版）

- 現在NGACIで議論中のシステム性能予測データの一部を紹介
  - 三輪先生＠電通大、安島さん＠富士通、塩見先生＠阪大に感謝いたします

- 仮定するプロセッサのタイプ
  - Latency Sensitive（LS）
    - データベースなどランダムアクセス処理にも対応可能なアーキテクチャ
    - 大容量メモリ、大容量LLC、深いキャッシュ階層
    - 例）Xeon, EPYC, Amazon Graviton, etc.
  - Bandwidth Centric（BC）
    - グラフィックス、HPCなどストリーミング処理
    - 高帯域メモリ、帯域×遅延に必要十分なローカルメモリ／キャッシュ
    - 例）Tesla, Radeon Instinct, Intel Xe, A64FX, SX-Aurora Tsubasa, etc.
  - Compute Centric（CC）
    - 学習、推論などAI処理
    - 高密度な演算器、必要十分な入出力帯域
    - 例）Google TPU, Cerebras WSE, Tesla D1, Esperanto ET, Graphcore Colossus, SambaNova, etc.

佐野補足: 2022/9/16
ver 1.1.0 のドラフトは完成
（公開に向け準備中）
その他の執筆者の皆様にも感謝

性能推定法: ピーク性能
TSMCのN2ノードを対象とし
・ロジック密度
・SRAM密度
・トランジスタ消費電力
・トランジスタ速度
の向上から現行品を外挿し推定
CCはオフチップメモリを持たず。

2028年のシステムの予測性能（更新版）

- ソケットあたりの性能比較

	LS model	BC model	CC model
FP64 Peak Performance	5.9 TFlops	12.8 TFlops	86.8 Tflops
Memory Bandwidth	614 GB/s	4 TB/s	173.6 TB/s
Memory Capacity	1.5 TB	512 GB	1.7 GB
B/F	0.25	0.31	2.0
Power Consumption	351 W	140 W	286 W

- 50MW、80%におけるシステム性能の見積り

	LS model	BC model	CC model
ソケット数	103,600	259,740	127,145
総コア数 (Millionコア)	6.63	258,526.49	523.96
PFLOPS	615	3,324	11,036
総メモリバンド幅 (PB/s)	63	1,063	22,072
総メモリ容量	159	132	0.22

総評

３つの異なる方向性に全振りした場合の推定値。
実際には、これらのハイブリッドを含め、
カバーすべき性能や、プログラミングモデル
等を考慮してアーキテクチャを探索

LS, BCのCPUベースでは十分な性能向上は不可

- **LSは汎用の高性能CPUベース**
  - ✓ データセンター向けシステムを持ってきた場合
- **BCはHPC向けの広帯域CPU, GPU等をベース**
  - ✓ 現在のスパコンの進化系
- **CCはオンチップメモリのみでバンド幅を稼ぎ、リソースの大部分を演算器に振ったもの**
  - ✓ 汎用CPUを用いずプログラマブルアクセラレータのみで構成するような極端な想定での推定

50MW、80%におけるシステム性能の見積り (TSMC 2N想定)

	LS model	BC model	CC model
ソケット数	103,600	259,740	127,145
総コア数 (Millionコア)	6.63	258,526.49	523.96
PFLOPS	615	3,324	11,036
総メモリバンド幅 (PB/s)	63	1,063	22,072
総メモリ容量	159	132	0.22

汎用CPUはスケールするか？

（メニーコアCPUベースで十分か？）

議論の前提

- **ソケット or ノード性能 で検討**
  - ✓ ここではシステム性能は考えず（コストや電力バジェットでシステム規模や性能が決まるため）
- **メニーコアCPUの実効性能を左右するアーキテクチャ要素は何か**
  - ✓ メニーコアCPUの限界
  - ✓ NGACI WPで見積もったのは、あくまでピーク性能
- **現在のCPUアーキテクチャの延長線上・巨大化で不十分なら、どのような選択肢があり得るか**
  - ✓ メニーコアCPUの限界を知ることは、取るべきアーキテクチャ指針を検討する出発点

メニーコアCPUの性能スケーリングの限界

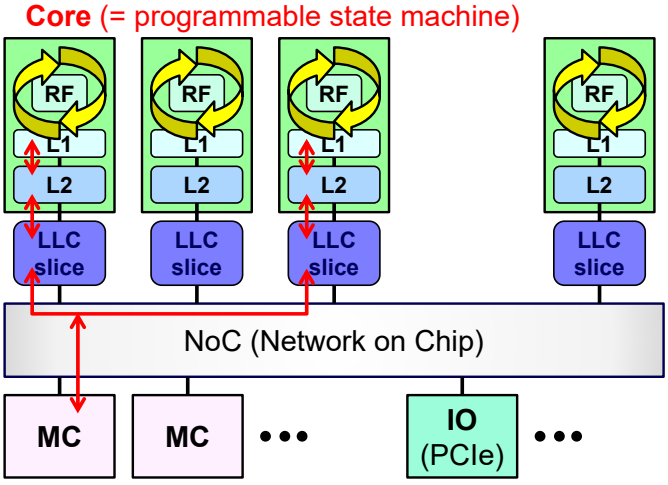
Core performance is difficult to increase.
- ✓ Limited frequency, limited parallelism
- ✓ Extra hardware required to boost IPC

Moving data via memory doesn't scale.
- ✓ NoC and cache can be a bottleneck of data supply from external memories.
- ✓ Inter-core data-movement is inefficient.
  > latency in writing/reading cache memories

NoC itself and shared LLC doesn't scale.
- ✓ Scalable NoC is challenging.
- ✓ $-coherence protocol gives higher pressure.

No more improvement in perf/power.

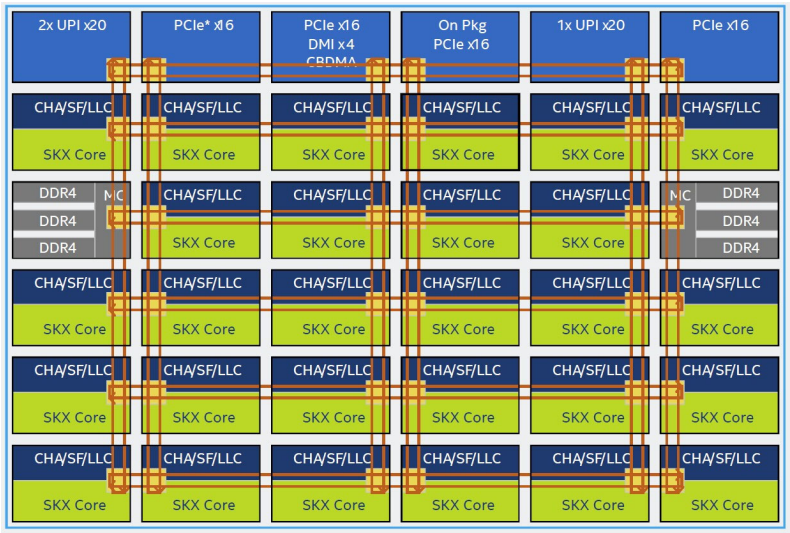
汎用でプログラマフレンドリではあるが、向上に限界



データ移動の効率化、制御の局所化・非同期化が重要

Intel Skylake-SP Xeon Processor (28 Cores)

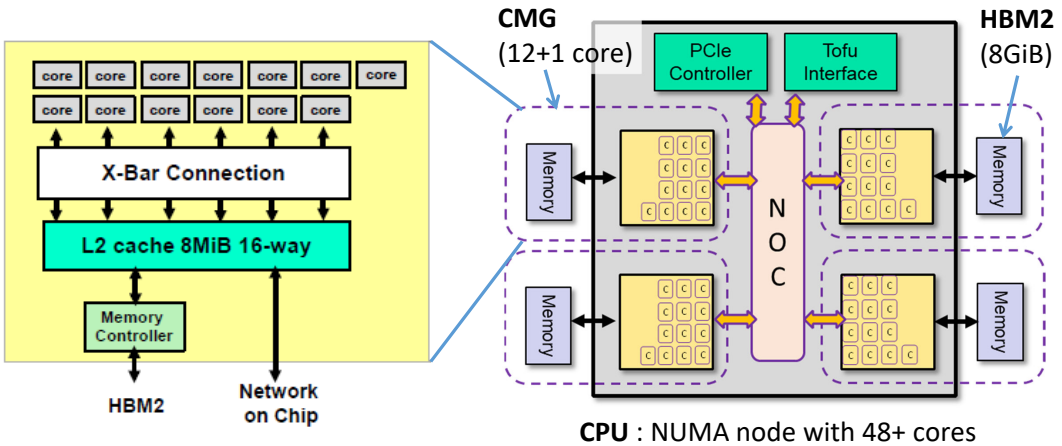
- **Ice Lake-SP is similar.**
- **6x6 Mesh**
- **Five rows of core & LLC slices**
  - ✓ Average latency is similar to Brdwell,
  - ✓ but more cores
- **Each of two MCs are on each side**
- **All IOs are at the top.**



CHA– Caching and Home Agent SF– Snoop Filter LLC– Last Level Cache
SKX Core– Skylake Server Core UPI – Intel® UltraPath Interconnect

From HotChips29
https://www.hotchips.org/wp-content/uploads/hc_archives/hc29/HC29.22-Tuesday-Pub/HC29.22.90-Server-Pub/HC29.22.930-Xeon-Skylake-sp-Kumar-Intel.pdf

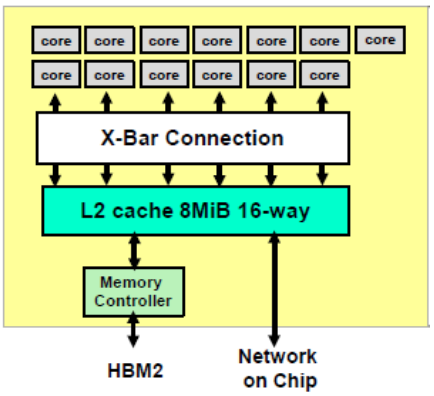
Fugaku A64FX Processor (48 Cores)

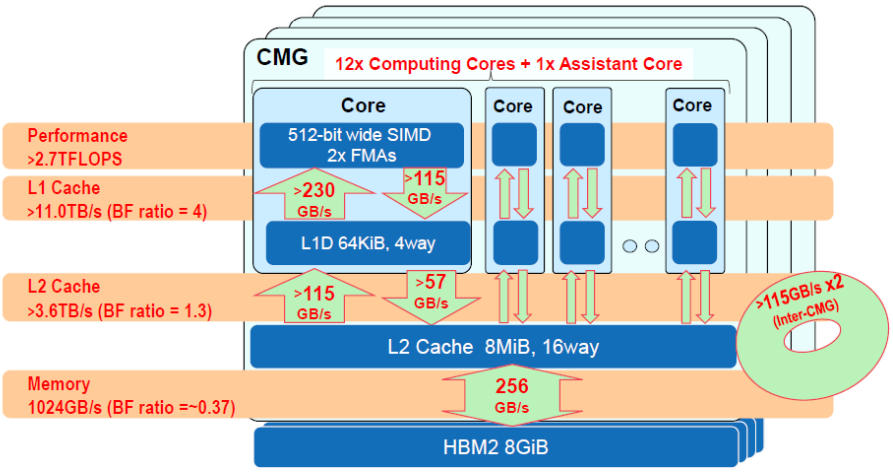


CPU : NUMA node with 48+ cores

- **L2 cache is shared by 13 cores with X-bar, per CMG**
- **Four CMGs are connected by a bi-directional ring for NUMA**

Performance of CMG and NoC

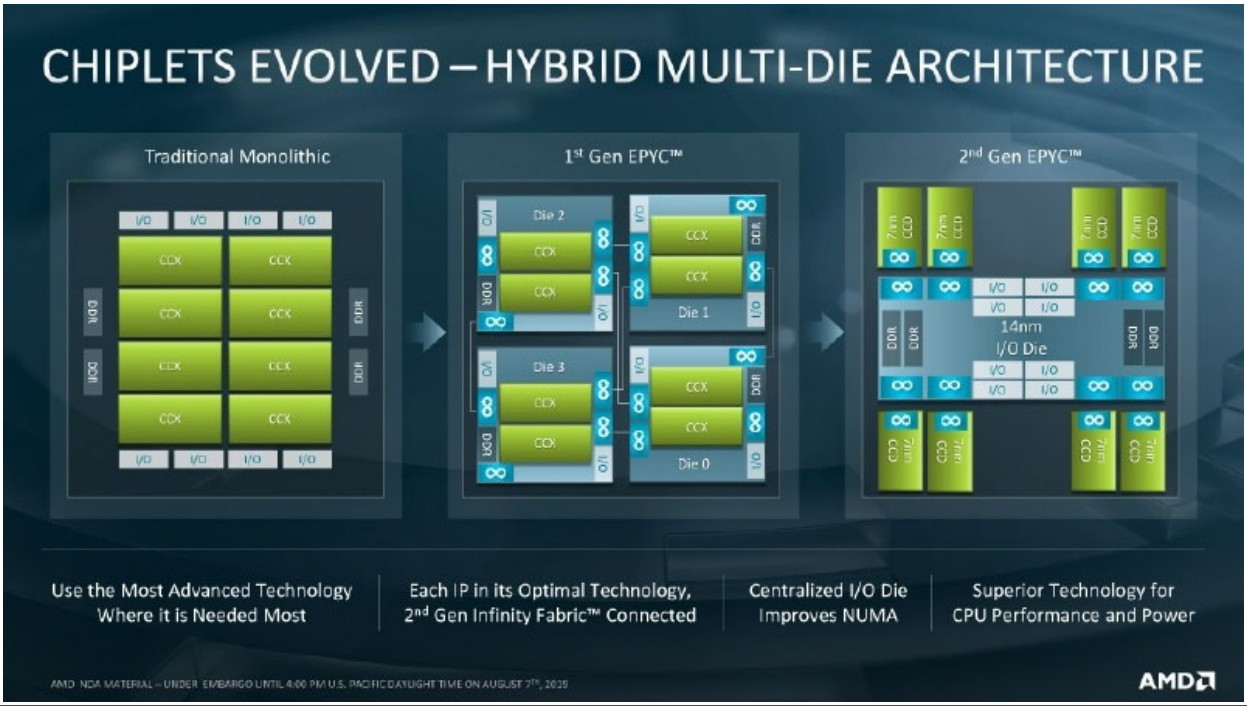




- **Clustered structure**
  - ✓ Four CMGs (core memory groups)
  - ✓ Memories are shared by CMGs (NUMA)
- **Each CMG has 12+1 cores** (ARMv8.2)

Performance numbers of CMG and NoC

Chipletにより制約・コストは変化

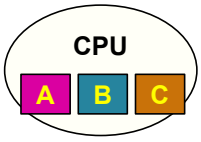


将来のアーキテクチャの可能性
（どのような選択肢があり得るか）

アーキテクチャ改良の方式 (CPU + X)

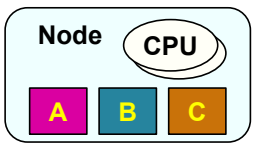
前提：既存CPUよりも高性能かつ高電力効率のアクセラレータによる拡張が必要かもしれない

- **CPU拡張方式**
  - ✓ CPU内にアクセラレータ要素を追加
    （拡張のバリエーションについては後で）



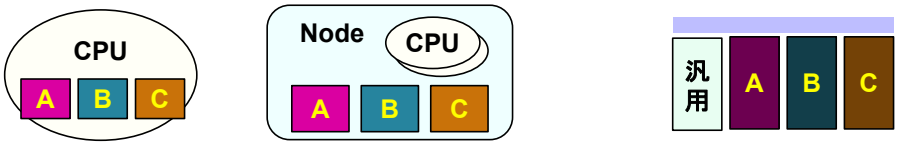
- ✓ **コア内拡張**
  - ➢ SIMD, 命令pipeline拡張
  - ➢ 特定機能エンジン(Matrix/Tensor/FFT)
  - ➢ キャッシュ近傍処理 (rd-md-wr))
- ✓ **コア外拡張**
  - ➢ 独立したアクセラレータXをNoCに接続
  - ➢ メモリ内/近傍処理, NW IF近傍処理

- **ノード拡張方式**
  - ✓ ノード内にアクセラレータ部を追加
  - ✓ PCI-ExpressやCXL等でCPUと結合



- **特化型ノードやラックの疎結合方式**
  - ✓ システム内に独立したアクセラレータ
    専用ノードやラックを追加
  - ✓ ネットワークで結合

アーキテクチャ改良の方向性



拡張方式 / 結合方法	チップ内結合	パッケージ内結合	I/Oによる結合	ネットワークによる結合
CPU拡張方式	○	○		
ノード拡張方式	➢ CPUダイ上で結合	○	○	
特化型ノード・ラックの疎結合方式		➢ Chiplet(2.5D), 3D結合	△	○

CPUとメモリ共有　　CPUとメモリ分散

- **保守的方向性**
  - ✓ プログラマビリティ、コード資産を重視
  - ✓ 拡張によりCPUの性能を向上させる方式
  - ✓ CPUと統一的にプログラムが可能
- **革新的方向性**
  - ✓ 可能性や性能向上を追求、生産性は将来開発に期待
  - ✓ CPUとは別のアーキを設けて性能を確保する方式
  - ✓ CPUは別のプログラムモデル必要

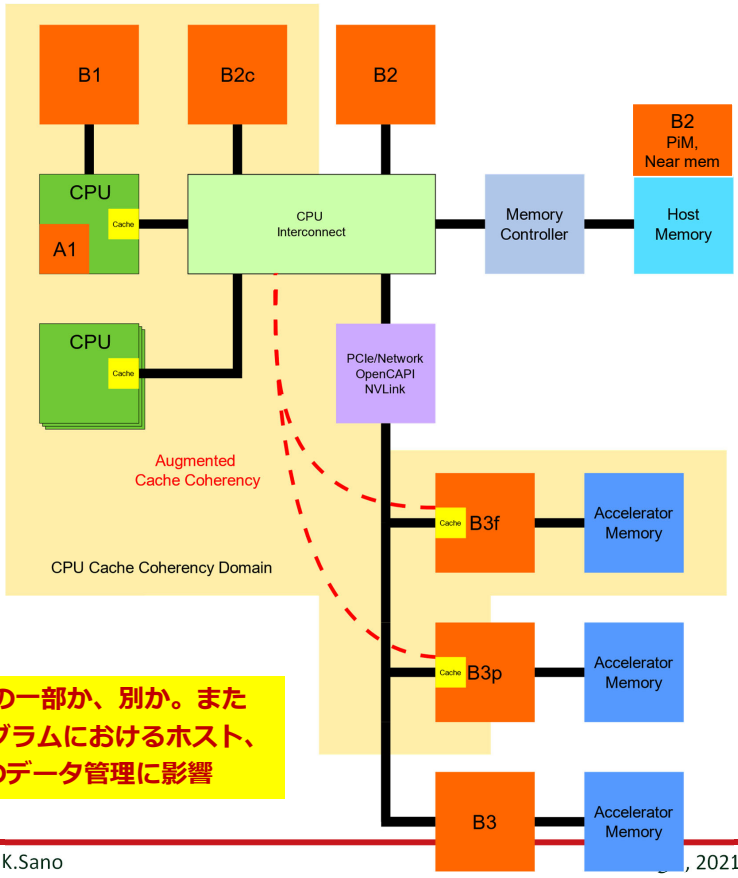
CPU・ノード拡張方式の分類

CPUと独立した内部状態の有無 (A, B)

- ✓ **A)** Stateless (100% controlled by CPU)
- ✓ **B)** Has its own state (CPU may still control ops.)

メモリ空間共有, キャッシュコヒーレンシ (1,2,3)

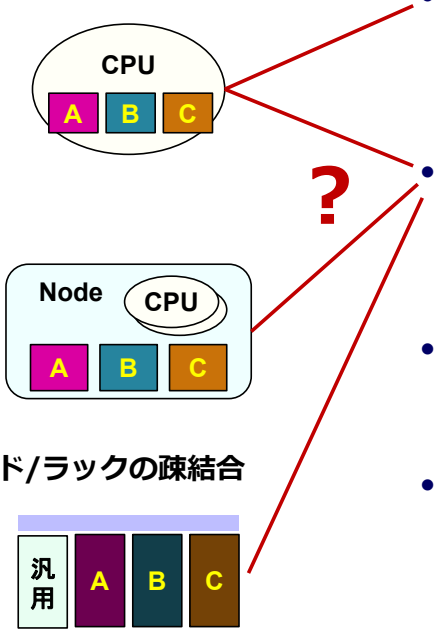
- ✓ **1) Depending on CPU**'s load-store unit
- ✓ Has its own load-store unit
  - ➢ **2) Direct access to main memory (e.g., CXL)**
    Sometimes it has its own memory and cache:
    - ✧ **2c)** with cache coherency
  - ➢ **3) In-direct access through PCIe, NVLink**
    Usually with its own memory,
    and/or augmented cache coherency:
    - ✧ **3p)** Partial/one way coherency
    - ✧ **3f)** Full coherency through address translation



CPUの一部か、別か。またプログラムにおけるホスト、Accのデータ管理に影響

アクセラレータ部の構成方式 (What's X?)

- **CPU拡張**
- **ノード拡張**
- **特化型ノード/ラックの疎結合**



- **固定機能アクセラレータ**
  - ✓ 特定の計算問題またはドメインのみを高速処理可能な構成
  - ✓ 最も高性能・高電力効率であるが、柔軟性やプログラマビリティが低い
  - ✓ 例）Matrix engine, Tensor engine, FFT engine, DL engine
- **（準）プログラマブルアクセラレータ**
  - ✓ 複数の計算モデル・問題を高速処理可能＋プログラマビリティを持つ構成
  - ✓ 固定機能アクセラレータに比べ、性能や電力効率で劣る
  - ✓ 例）GPGPU, データフロープロセッサ（CGRA, Coarse-Grain Reconfigurable Arrays）
- **計算モデルによる分類**
  - ✓ ノイマン型（GPU）　命令の逐次実行ベース
  - ✓ 非ノイマン型（FPGA, CGRA）　問題を回路に展開し実行。データ駆動型
- **特定計算ドメインの候補**
  - ✓ 疎行列計算、N体問題、FFT、ソート、グラフ処理、脳型計算、量子計算シミュレーション、エージェントシミュレーション、他？
  - ✓ 演算子、依存関係、データ構造（メモリ参照パタン）などで特徴づけ

アプリ・アルゴリズムからの要求 (What's X for?)

- **個別の要求**
  - ✓ SCALE-SMD: 配列へのリストアクセス
  - ✓ 精密格子QCD: Allrecue専用NW、4次元構造格子を並列化しやすいNW
  - ✓ GENESIS: リアルタイム可視化のための高いポスト処理性能
- **専用ハードウェア機構への期待**
  - ✓ 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）
  - ✓ 疎行列専用ハードウェア（疎行列計算）
  - ✓ メモリアクセスのスケジューリング支援機構（モンテカルロ計算、グラフ処理）
  - ✓ アニーリング（イジング模型による最適化問題）
  - ✓ データフロー型かつ大量のレジスタを持つハードウェア（FFT）
  - ✓ ソーティングのハードウェア実装（ソーティング）

将来アーキテクチャの課題

研究開発の目標

- **電力制約のためにシステム性能向上には電力性能比の改善が必須**
  - ✓ WPの推定からも、既存CPUアーキの延長では目標達成は困難
  - ✓ CPUに代わる、電力性能比を向上できるアーキを導入の必要性

どのようなオプションがあり得るか？
アプリ・ユーザを考慮した時、
何が許されるか？

- **性能・機能・特性において、達成すべきは何か？その優先順位は？**
  - ✓ **計算性能** （ピーク性能, アプリ実効性能）
  - ✓ **電力性能比** （システムの総合性能を左右）

  } 計算そのものの価値に影響

  - ✓ **コスト** （開発コスト, 製造コスト）
  - ✓ **汎用性** （種々の計算問題に対する計算効率）
  - ✓ **生産性** （プログラムし易さ, 既存コード再利用性, 性能可搬性）

  } 利用者数に影響
  アプリの広がりに影響

  - ✓ **運用可能性** （共用設備またはクラウド利用などに適すか）

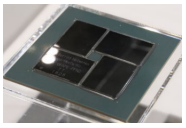
  ——— システム稼働率に影響

  - ✓ （耐故障性、保守性、等）

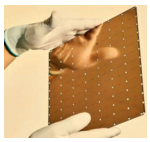
課題

- **汎用CPU（コア）部と拡張部Xの割合は？**
  - ✓ 拡張部 X の汎用性が低い場合には?（どのドメインか）
  - ✓ 複数方式を組み合わせるか?
    例）CPU内拡張（強化型CPU） + Accによるノード拡張
- **プログラミングモデルは?**
  - ✓ CPU用のプログラミング方式を適用できるか？
    C/C++ (11,14,17), FORTRAN, Python
    OpenMP, MPI,
  - ✓ アクセラレータ用プログラミングが求められるか？
    OpenCL, SYCL, DPC++, etc.
  - ✓ ライブラリ、フレームワーク提供で十分か?
- **必要な（人的）コストを誰が担うか？**
  - ✓ CPUコンパイラの拡張, ライブラリ・フレームワーク開発
    - ➢ **ベンダやシステムソフトコミュニティ**
  - ✓ アクセラレータ用コードマイグレーション
    - ➢ **ユーザ**

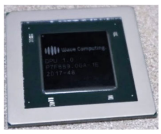

Graphcore


PFN MN-Core


Groq


Cerebras

SambaNova


Wave computing


XILINX
VERSAL
(ACAP)

近年のプログラマブルアクセラレータの例
多くの場合汎用コンパイラ(C/C++/FORTRAN)は未提供

ユーザからの要件は？

- **圧倒的な性能・規模の向上？**
  - ✓ xx倍速くなって欲しい？
  - ✓ yy倍大きな規模を計算したい？
  - ✓ そのために許容できる犠牲は？
- **コードポータビリティ？**
  - ✓ 既存コード（特にCPU向けのC/C++, FORTRAN）の移植は許容できない?
  - ✓ 指示子を入れる程度（OpenMP, OpenACC等）ならOK？
  - ✓ 妥協案として、ホットスポットのみをAcc向けに書き換えるのはOK？
- **Time-to-solution？圧倒的な生産性？**
  - ✓ 新規コードが簡単に書けて、それなりの性能が出ればいい？
  - ✓ 素人プログラミングから玄人（高性能・大規模）プログラミングへの移行が楽であって欲しい？

おわりに

- **既存CPUアーキテクチャの延長線のみでは厳しい時代**
  - ✓ 電力効率を上げるための拡張、という方向性
  - ✓ 拡張方式、アクセラレータ構成方式により、達成可能な性能や使い易さが変わる
  - ✓ 汎用性重視なら既存CPUベースである程度の性能を達成（しかし向上は限定的）

- **舵取りに必要なのは、ユーザの要求と許容**
  - ✓ どのような種類の計算をターゲットにするか
  - ✓ 性能と、生産性、既存コード資産
  - ✓ コンサバか、アグレッシブか

- **システム開発側とユーザの意思疎通がこれまでにも増して重要**
  - ✓ 対話を継続していきましょう
  - ✓ 今後の大きなトレンドを見据えて議論しましょう