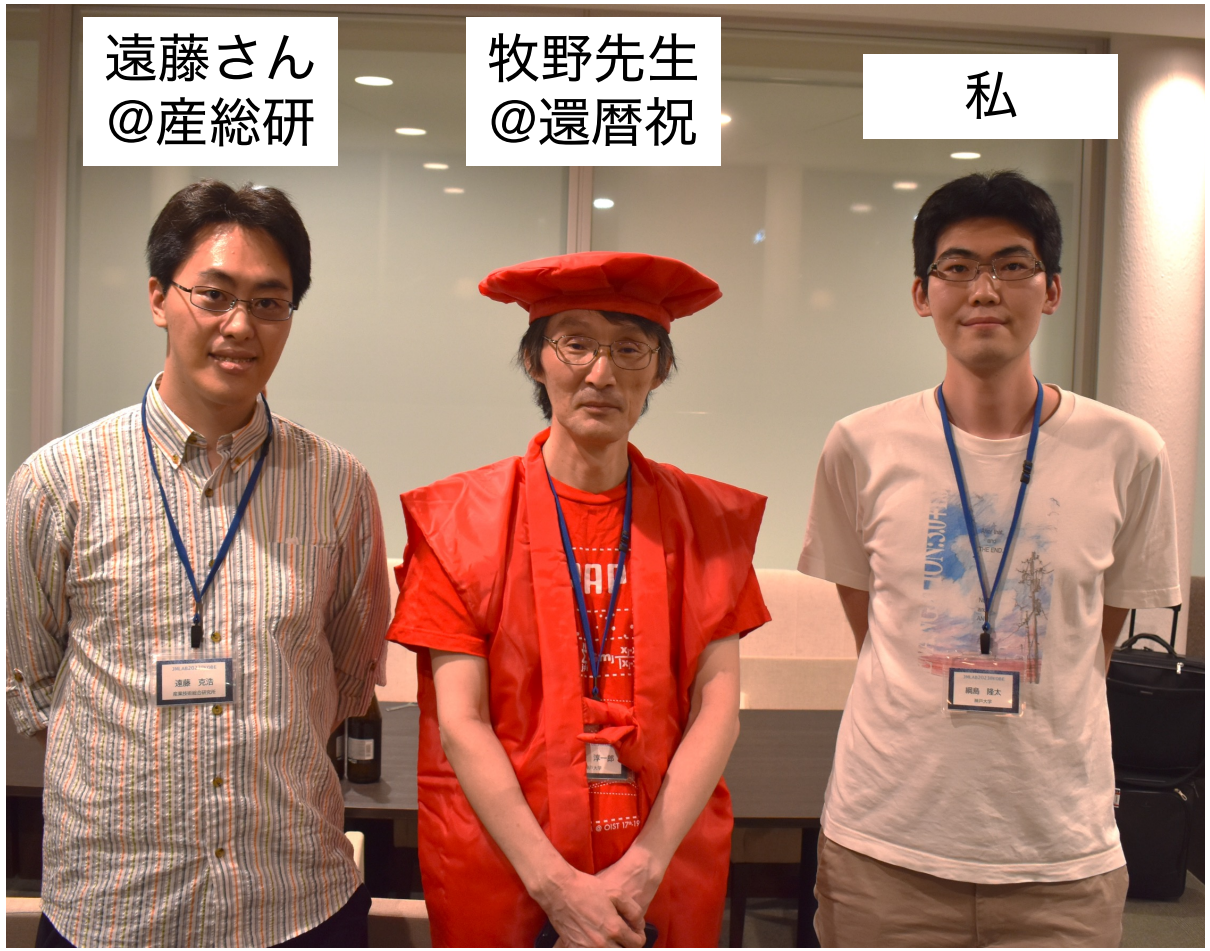
アクセラレータ用次世代プロセッサ MN-Coreの動向の紹介 （ポスト富岳調査の話を交えて）

綱島隆太

神戸大学大学院 理学研究科 惑星科学研究センター 特命助教

牧野研究室

自己紹介



去年のCPS研究会の写真

- 博士（工学）
  - 今年の3月取りました
  - コンピュータサイエンスが専門
- 神戸大CPSには昨年度から在籍
  - その前は別の大学の博士課程に在学
- 現在、MN-Coreのプログラミング環境が研究テーマ
  - OpenACC（C、Fortran）をMN-Coreで使えるようにしようとしている
  - このあたりは去年のCPSセミナーで話した（今日も触れる）

本日話す内容

- 次期国家スパコン調査研究（ポスト富岳FS）の説明
- ポスト富岳FSで神戸大チームが何をやっているかの説明
- 次世代国産アクセラレータであるMN-Coreの紹介
  - MN-Coreとは何か？
  - どんな感じに使えるのか？（プログラミング言語など私の仕事の話含む）
  - MN-Coreコミュニティの宣伝

今日の話の位置づけ

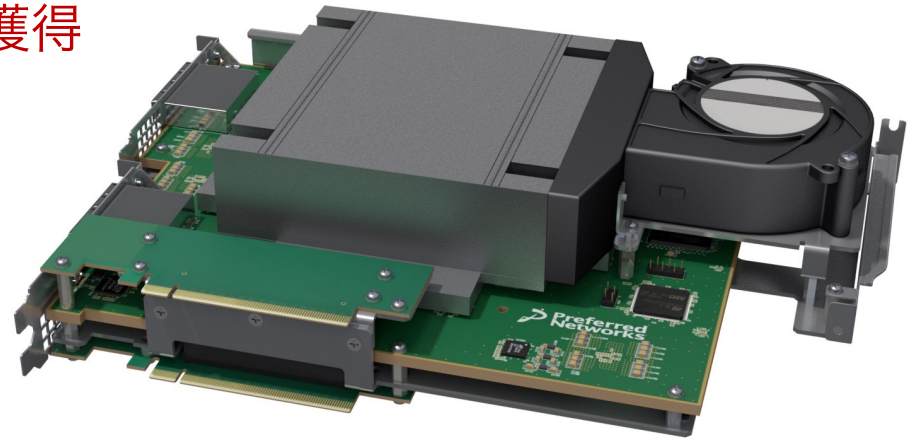
MN-Coreの宣伝、普及活動

MN-Coreの紹介

MN-Coreについて

- 神戸大学と㈱Preferred Networks (PFN) が開発したアクセラレータ
  - 汎用計算のみならず、深層学習の高速化を考慮した設計
  - アクセラレータとは？
    - CPUに接続して使う、より速いプロセッサを搭載したデバイス（GPUなどのこと）
  - 電力あたりの演算性能が同世代の既存プロセッサに比べ、非常に高い
    - MN-Coreを搭載したスパコンはGreen500で1位を3度獲得（2020～2021年）
    - GPUスパコンよりも高い電力効率だった（重要）

	2020年6月	2020年11月	2021年6月	2021年11月
ランキング	1位	2位	1位	1位
電力効率	21.11 GFlops/W	26.04 GFlops/W	29.70 GFlops/W	39.38 GFlops/W
実行効率	41%	53%	58%	64%



MN-Core Board（PFNより）

PFNについて

- AIを主軸に置いた日本の企業
  - 元々Chainerという深層学習ライブラリを作っていて国内外で有名になった
  - ソフトウェアメインの会社だが、
    神戸大学（牧野研）と共同でハードウェアとしてMN-Coreシリーズを開発

Preferred Networks

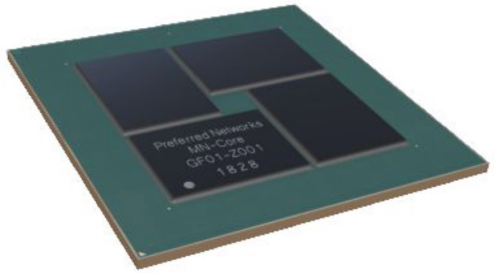
（PFNサイトより）

神戸大学（牧野研）とPFNの関係

- 共同研究している関係
  - 今はハードウェアだけでなく、 MN-Coreのシステムソフトウェアも
- MN-Coreシリーズは共同研究の成果物（大事）
  - なので、本当は全ての資料に「**神戸大学と**PFNが共同開発しているMN-Coreシリーズ」と書かれてないといけない
    - ~~商品自体はPFNから出ているので忘れられがち~~
    - ~~私も一回指摘があって再認識~~

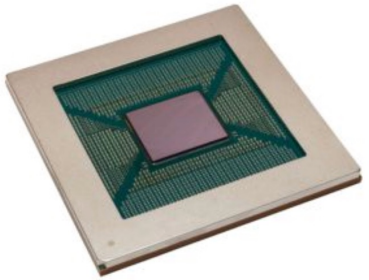
現行のMN-Core

第１世代



MN-Core
TSMC 12nmプロセスで製造
2020年稼働開始

第2世代



MN-Core 2
TSMC 7nmプロセスで製造
2023年稼働開始

（PFNサイトより）

- MN-Core（無印）とMN-Core2がある（2は去年出た）
  - このスライドでは主に無印を例に説明
  - 2のときは「MN-Core2」と呼ぶ
  - 両方に当てはまる話をあえて強調する時は「MN-Coreシリーズ」と呼ぶ

MN-Coreのスペック

MN-Core第一世代チップ仕様

消費電力 (W、予測値)	500
ピーク性能 (TFLOPS)	32.8 (倍精度) / 131 (単精度) / 524 (半精度)
電力性能 (TFLOPS / W、予測値)	0.066 (倍精度) / 0.26 (単精度) / 1.0 (半精度)

（PFNサイトより）

MN-Core 2 カタログスペック

	MN-Core 2	MN-Core 2（電力効率）
FP64	12 TFlops	37.24 GFlops/W
FP32	49 TFlops	148.9 GFLops/W
TF32	98 TFlops	297.9 GFlops/W
TF16	393 TFlops	1,192 GFlops/W

- 用語
  - FLOPS（Flops）：1秒あたりの浮動小数点数演算性能
  - FP64：倍精度 / FP32：単精度 / TPxx：深層学習用浮動小数点数
- 参考
  - 注意点として、理論性能（カタログ値）が高くても、実際の性能はこんなに出ない（実行効率については後述）
  - CPU：FP64：約5.5TFlops（https://note.com/ewewithbousai/n/n4f1fe69da4d4 より）
  - GPU：H100でFP64：最大68TFlops、FP32：最大134TFlops（Tensorコア不使用）

MN-Coreの設計

- 分散メモリ型SIMDアーキテクチャ
  - 毎サイクル、一つの命令で各PEが、別々のデータに対して同じ処理をする
  - PE（Processing Element）：CPUでいうコアに当たるもの
- 設計上の制約（**ハードウェア側制約**）
  - 回路面積とレイテンシの制約
    - コアが増えると回路が複雑化し、回路が長くなると遅延が発生
    - 演算器をたくさん配置しつつ、回路面積を抑える工夫が必要
  - 電力と発熱の制約
    - 電力と発熱は増加傾向
    - 回路の微細化で抵抗が増加
    - 同じ電力で演算性能を最大限引き上げる工夫が必要（特にGPUと比較して）

アクセラレータにおける性能とは何か？

- 演算性能＝並列処理性能
  - FLOPSは一度にできる演算の数が多ければ高い数値になる
  - つまり、（特にHPCでは）大前提としてとにかく並列数が多くなければならない
    - GPUは元々そういう並列プロセッサだったので、HPCでもうまくハマった
- 実行効率の問題
  - 実行効率：理論性能に対する実行時の実際の性能（実測値）の割合
    - 実測時の性能ことは「実効性能」という
  - 実際のHPCアプリケーションでは演算実行効率が数%とかが普通
    - 特に理論性能（理論最高演算性能）が高いと低くなることが多い

実行効率が低くなる原因

重要 ソフトウェア側制約

- 命令が上手くハマらない
  - 理論最高演算性能はあくまでベストエフォート
  - 例えば、理論最高性能は通常FMA命令を使っているが、積和にならない演算をするとその時点で理論最高性能まで出ない
    - FMA命令：通常掛け算と足し算で合計2命令必要なところを、一気に行える命令（実行時間短縮）
- 多くのアプリは通信律速
  - 並列計算では通常、各分割単位で計算中に依存する値を、袖交換などの通信によって交換する
  - 通信は理論演算性能に含まれないので、演算と通信を同時にしたり、そもそも演算のほうが通信より時間が掛かるケースでない限りは通信時間の分だけ実行効率が下がる
- チューニングが難しい
  - 上記を上手く回避しようとすると、現状では常人をやめることになる
  - そもそも実行しないと実際の性能がわからないので、チューニングに時間を要する

MN-Coreの設計

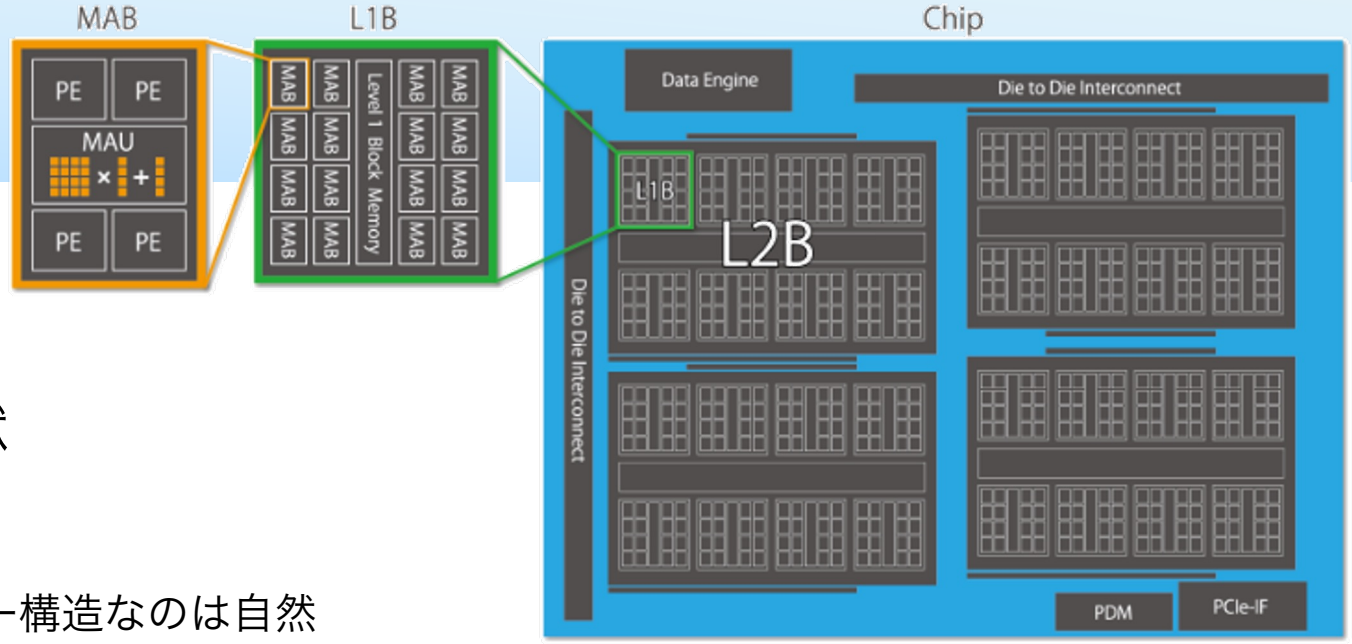
- 前3スライドの問題全てを考慮して設計
  - 特にチューニング難易度の問題はこれまでのプロセッサでは見直されてこなかった

制約への対応

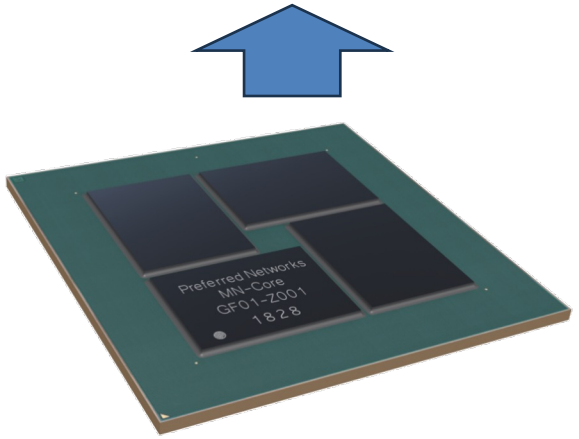
- 行列演算ユニットMAUの実装（一発で処理できる命令の増加と集約）
  - 深層学習向けだけど、汎用の行列計算にも使える
  - 今でこそGPUにも同様のものが搭載されているが、それより前に設計済だった
- 通信の明示的な制御
  - 通信を事前に極限まで削減可能に
- チューニングの妨げとなるロードバランシング、性能最適化のハードウェア実装を削減
  - 事前に性能予測を可能に
  - コンパイラでの解決を目指す
- 上記に伴って、チップ面積及び電力あたりの演算性能を向上

MN-Coreの構造

- 設計方針として極力単純化
  - チップ面積当たりの演算器の割合が高い
  - 面積及び電力あたりの演算性能向上に貢献
- チップ内部はツリー階層構造（→）
  - 回路延長の削減と演算器詰め込みを考慮するとツリー構造なのは自然
  - 通信バッファのための階層（L2B、L1B）が存在
  - **L2B** (Level 2 Broadcast Block) x 4個（チップ1個あたり）
    - **L1B** (Level 2 Broadcast Block) x 8個（L2B 1個あたり）
      - **MAB** (Matrix Arithmetic Block) x 16個（L1B 1個あたり）
  - MAB 1個あたりには以下を内蔵
    - **PE** (Processing Element、通常の演算器) x 4個
    - **MAU** (Matrix Arithmetic Unit、行列演算器) x 1個
  - **PEは1パッケージ（ボード）あたり合計で8192個存在**



MN-Coreチップの中身の構造

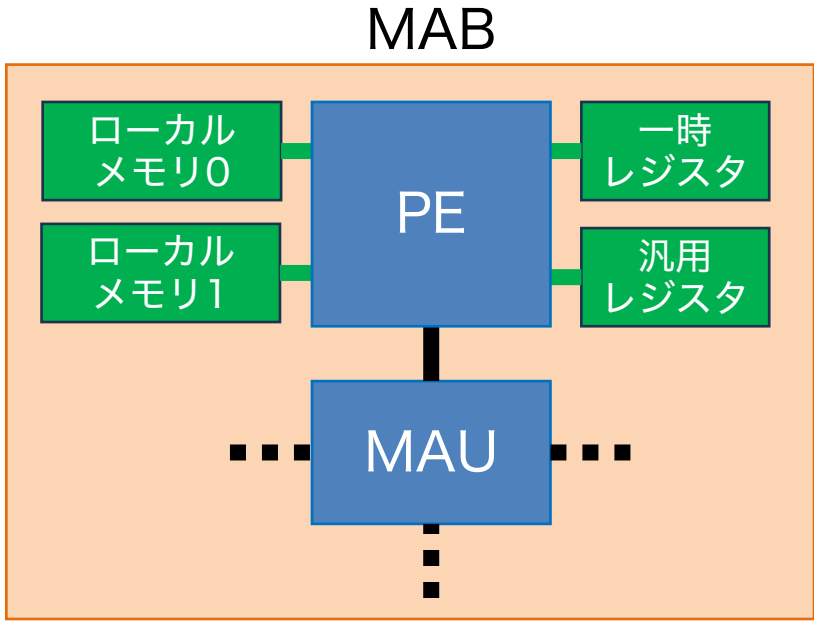


（図はPFNサイトより）

MN-Coreパッケージ（4チップ搭載）

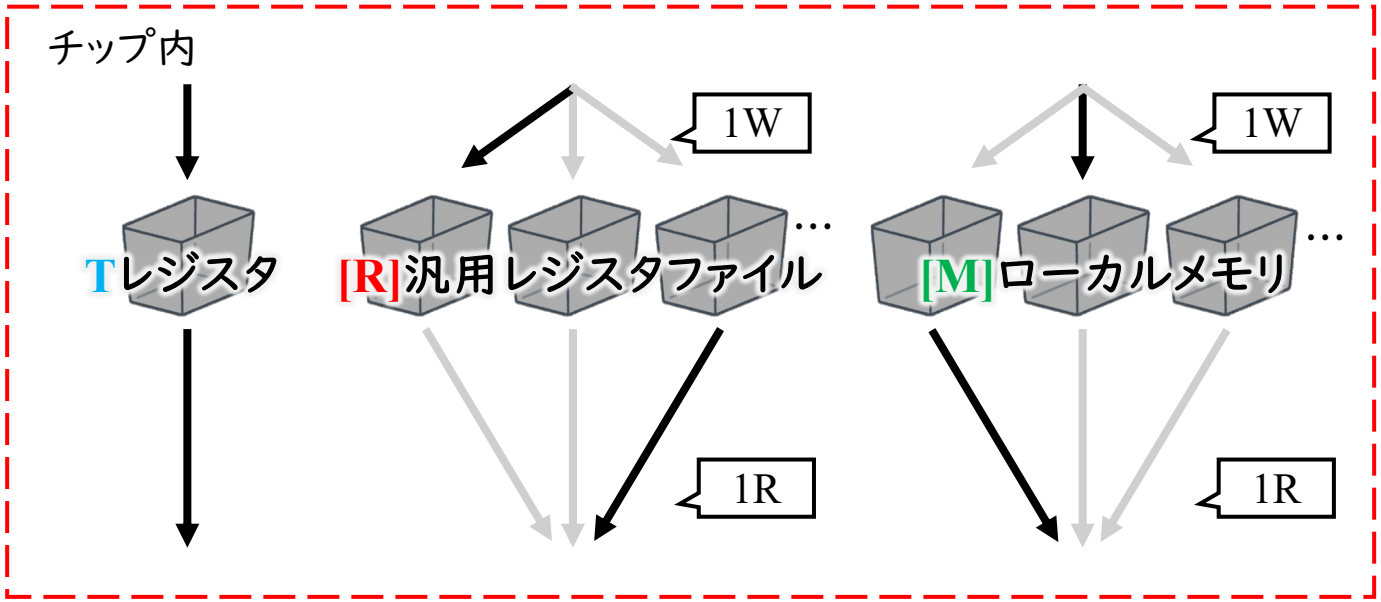
MN-Coreのメモリ構造

- コヒーレントキャッシュメモリは無い
  - 回路面積と消費電力の増加、レイテンシの原因
    - キャッシュのコヒーレンシを保つためには暗黙的にデータ転送が発生
    - キャッシュの動作を類推しなければならないブロッキングを含め、HPCにおけるチューニングでは不利
- **MN-Coreは分散メモリ型**
  - 8192PEそれぞれにローカルメモリ（SRAM）が存在
    - 1PEあたり72KiB、1ボードあたり合計576MiB
    - PEから直接参照可
    - MAB内ではMAUと共有



レジスタのポートが最小限

- MN-Coreのローカルメモリ、レジスタはマルチポートではない
  - MN-Coreでは汎用レジスタのみR/Wがデュアルポート、その他はシングルポート
    - 通常、レジスタはマルチポート
  - ポート数の削減で回路の複雑化回避し、回路面積や遅延、消費電力を削減



by 遠藤さん@産総研

MN-Coreの動作

- 完全なSIMD動作
  - 全PE、MAUが同じ命令で動く
    - 条件分岐はマスクレジスタにより各条件ごとに分けて実行
- スレッドやアウトオブオーダーなどの負荷分散、性能最適化機構をハードウェア実装していない
  - **これにより、事前の静的な性能予測ができる**
- 分散メモリにより、データ移動や並列動作は全てプログラマブル
  - 1命令あたりのサイクル数も決まっているので、性能予測しやすい
- 通常の演算、行列演算、分散メモリ通信のオーバーラップ実行が可能
  - PE、MAU、Broadcast Memoryは条件が揃えば同時並行して動かせる
  - 全てがSIMDで同期実行しているからこそ制御可能

実はGRAPEを引き継いだ設計

- 吉川先生@筑波大は「GRAPE-DR使ってた人なら同じようにアセンブリ書けば使えそう」という感想だった
- N体問題専用計算機GRAPEを汎用設計したものがGRAPE-DR
  - 実はMN-Coreはその子孫（開発時名称：GRAPE-PFN2）
  - PFN CEOの西川さんは元々東大平木研で後述するGRAPE-DRのコンパイラを研究していて、そのあたりもMN-Core開発に繋がったらしい
- 一言で言えば「アクセラレータとしてスマートな理念・設計」
  - ハードウェア制約を考慮しながら、利用実態に沿って実効性能を高める設計
  - 去年、やることが進まないときに（右も左もわからなかったので）牧野さんの個人ページのMN-Core関連の記事を読み漁っていたときの感想
    - ~~これによってMN-Coreの設計理念を少しわかった気でいる~~

MN-Core特徴まとめ

ポイント（GPUやCPUとの違い）

- ハードウェア上での負荷分散、性能最適化のための機構を極力削減
  - もっと簡単に言ってしまえば、設計を見直してみて不要な機能は捨てている
  - あって当たり前になっているが、高機能なハードウェア機構は回路面積に占める割合が高く、性能チューニングの複雑化の原因
  - ここを削る代わりに、演算器を多く配置してチップ面積あたり＆電力あたりの演算性能を高めている
  - **そもそも、物理的なハードウェアの制約からはどう頑張っても逃れられない**
    - MN-Coreの設計は、ユーザービリティに加えて、この点を指摘している
  - 性能を追求するアクセラレータにおいて、これまでの高機能、ロードバランシング重視の設計に対するアンチテーゼ

ただし

- こういう設計をすると、コンパイラはハードで捨てた機能の分、賢くある必要が出てくる
- つまり、 MN-Coreが本当にすごいということは、コンパイラを開発できないと証明できない
  - 他のプロセッサと同じレベルのハードルで、より高い性能が発揮できるということを実証しなければいけない

ポスト富岳FSと将来のスパコンの動向

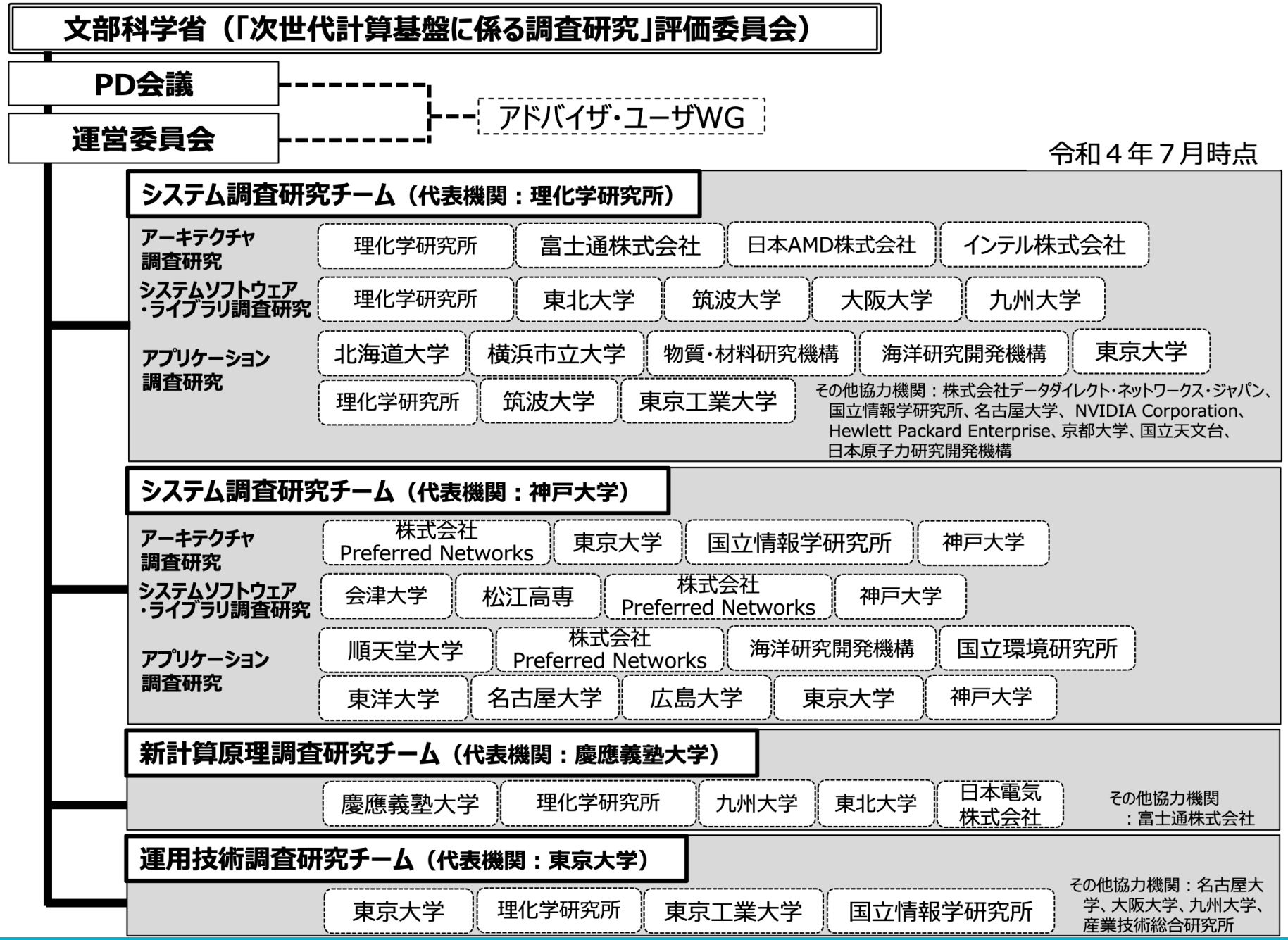
FSによってMN-Core周りの開発（特に汎用/HPC向け）が推進中

ポスト富岳FSについて

- 富岳の後継スパコンの調査研究（以降、FSと呼ぶ）
  - ポスト富岳は2030年ごろに稼働予定
- 文部科学省の事業（正式名称：次世代計算基盤に係る調査研究）
  - FS：フィージビリティ・スタディ、事前の調査研究
  - 2022年に公募があり、その中の一つとして神戸大チームの提案が採択された
    - 神戸大チームの提案では、MN-Coreをアクセラレータとして搭載したスパコンを調査研究
    - なお、神戸大と名乗っているが、神戸大で実際に担当している部署はCPS（特に牧野研）
  - ポスト富岳FSは今年度いっぱいまで継続

「次世代計算基盤に係る調査研究」実施体制

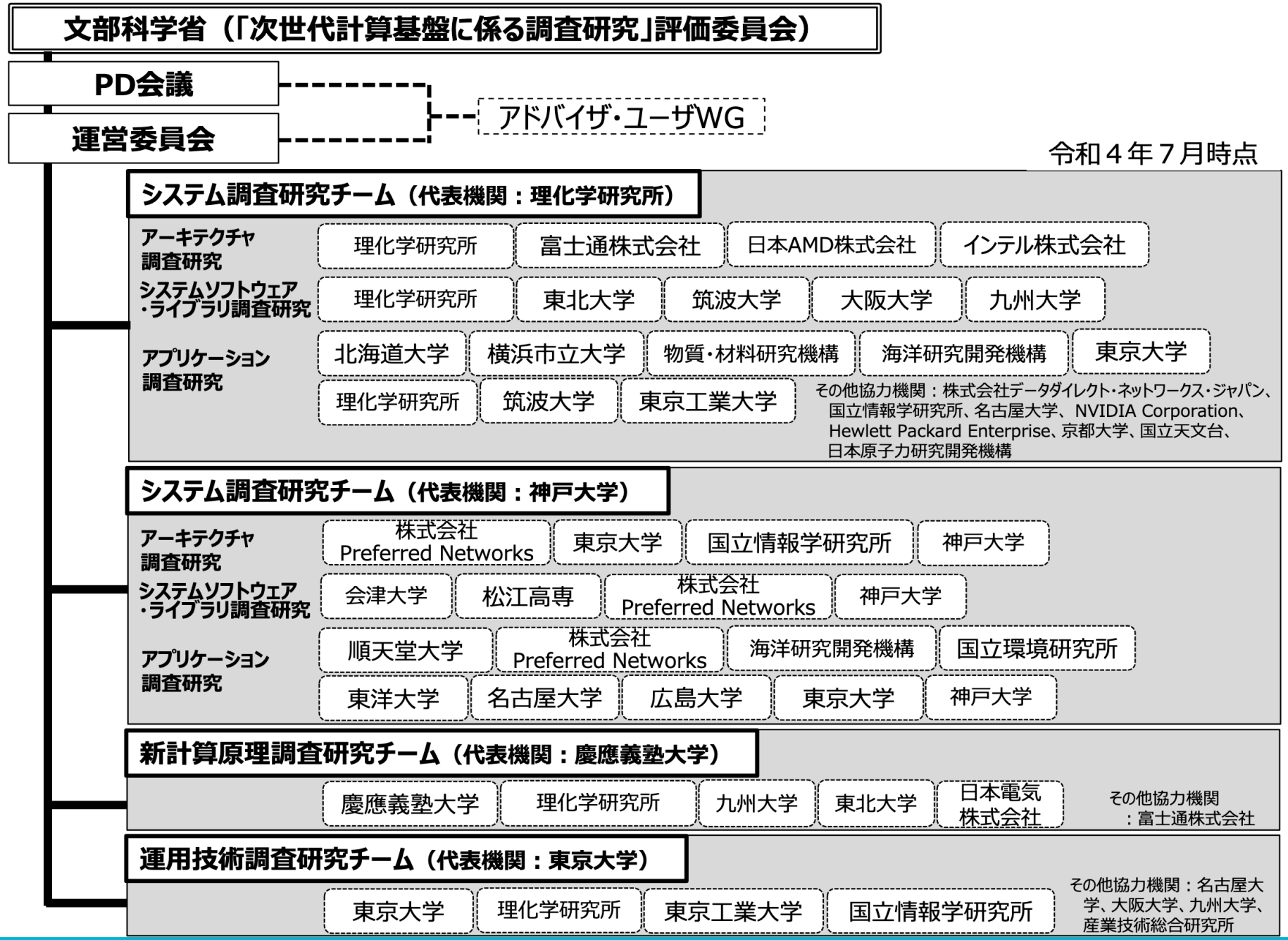
（図は文科省より）



- システム調査研究が理研と神戸大で2チームある
- 要するに2つのシステムが提案された
  - 理研では別のアクセラレータの搭載が提案された
  - **その中には海外のGPUメーカーも含まれている**

「次世代計算基盤に係る調査研究」実施体制

（図は文科省より）



- あとの2チームはシステム（ハードを含めた全体の設計）ではない
- 新計算原理は量子コンピュータの調査チーム
- 運用技術は名前の通り、運用関連
  - ジョブの割当て
  - ストレージ、OSなどのリソース分割など

富岳からポスト富岳へ

- 富岳は設計が色々大変だった
  - 多数のコアで、キャッシュや遅延をどうするか、限界も見えていた
    - 詳しくは牧野さんのページを見に行くと色々書かれています
  - それをベースにどうしていくかというのが今回のFSで検討された
- 例えば、最も大きいところとして、最初からアクセラレータを搭載する方針になった
  - ただし、そのアクセラレータにどのようなものを搭載するかが議題となった

FSでのMN-Coreの一番のネックは何か？

- 汎用プログラミング言語環境が存在しないこと
- 現状、Cで書けない
  - Fortranでも書けない
    - **アプリで評価ができない**
  - 通常、プロセッサの高級言語インターフェイスとしてCは必ず用意されている
    - マイコンでもなんでも、まずアセンブリに近いレベルの低レイヤー制御ができないと困るため
    - もし無いと、単純に性能が出せない、細かいハードウェアの機能が使えない
  - 特に、HPCでは性能重視なので、スパコン用途でCやFortranが書けないのではハッキリ言ってダメ
    - これはアプリの既存のコード記述やライブラリ開発の点も含まれている

では、現状どうなっているか？

- AI向けにPyTorchで書けるようになっている
  - だが、あくまでPythonライブラリ
  - そこから（他の高級言語を経ずに）直接アセンブリを生成している
    - 通常の高性能Pythonライブラリはその前段階にC言語が挟まっている（ライブラリ呼び出し）
    - PFNのソフトウェア環境を作った人たちがAI以外のことを考えていなかった様子
- 深層学習では行列演算が主であるため、それに特化されている
  - AIのように処理が決まっている場合は、関数化して、既に最適化されたコードを呼び出せば良い
  - だが、スパコンではそうではない
  - 例）自由記述によるアプリ開発、HPCにおける性能チューニングなど
- お世辞にも、AI以外をやらせたときに使いやすいとは言えない
  - 性能面でもよろしくない

高級言語I/Fが存在するとはどういうことか？

- コンパイラが存在するということ
- なので、コンパイラの開発をしなければならない
  - PFN社内だけではなにかと進まないので、ここを私もやっている
- そうじゃないと、HPC向けにはアセンブリを書くことになる
  - 常人には無理（一部FSのためにやっている人はいる）

ポスト富岳FSの動向と今後

- ポスト富岳としては理研側提案システムから新規性を削いだものが採用された
  - 神戸大の提案を採用しない理由は、簡潔に言うと
    「まだ実現性が低く見えるので選択を決断できない」みたいな感じ
    - だが、既存の設計ベースで5年後にユーザーの満足するシステムが作れるのかは極めて疑問
    - なお、量子コンピュータもまだ早いということで見送りになった（こっちは妥当）
- しかし、次期スパコンでは10年に一度のシステム入れ替えではなく、
  稼働させながらマシンのラックを半分ずつ入れ替えるという計画になった
- このポスト富岳稼働から5年後？に入るポストポスト富岳の公募が
  来年初めにある
  - ここで神戸大チームの提案が再び採択される可能性がある
  - そうすると、MN-Coreの国家スパコン搭載に向けた研究は継続する見込み

個人的な見解（モチベーション）

- 国産アクセラレータ技術がちゃんとあるべき
- それがフラッグシップスパコンに載っているべき
- スパコンの性能を左右するアクセラレータがちゃんと設計されているべき
  - 必ずしも多数派の設計である必要はない
  - 設計の研究開発にちゃんと投資すべき
- 自分たちで使いやすいスパコン、アクセラレータを考えるべき
- 先述のように、今やってることは将来的なアクセラレータ設計のベースになり得る話であり、そうなったら今の仕事はかなり評価される（はず）

MN-Coreは
将来どんな感じに使えるのか？

MN-Coreの汎用環境実現の位置づけ

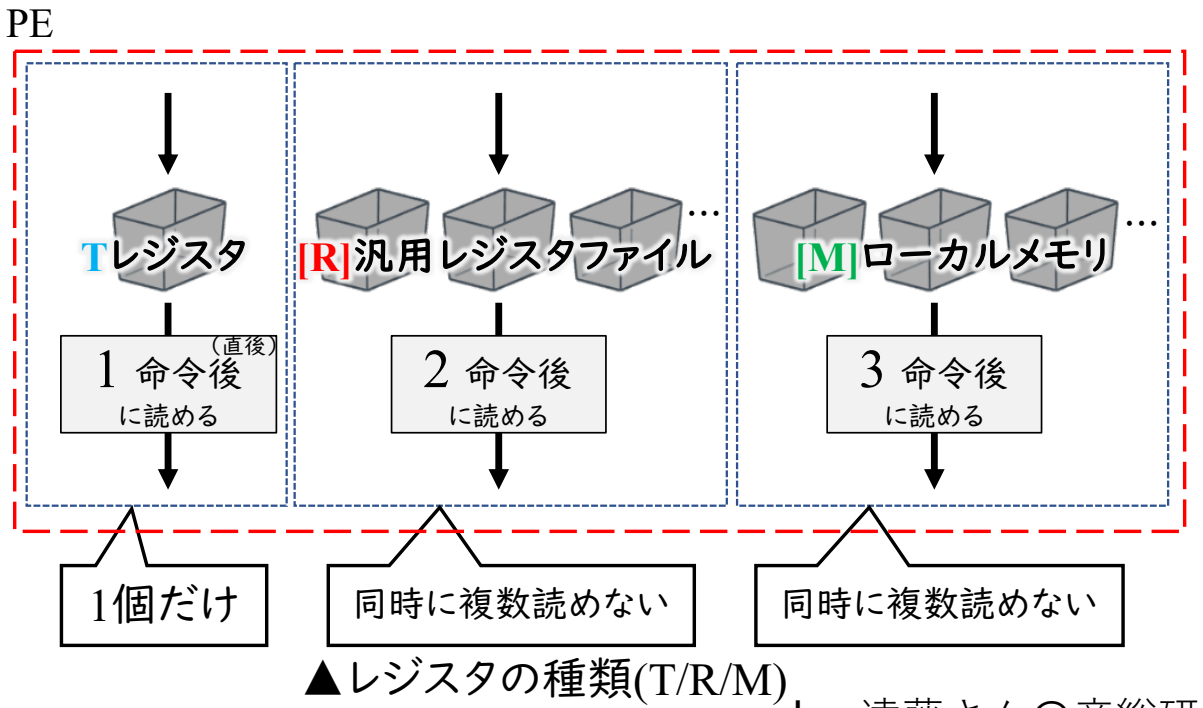
- きっかけとしてはポスト富岳FSだけど、FSのためだけではない
- これを契機に、例え国家スパコン採用されなかったとしても、MN-Coreの汎用プログラミング環境等の整備は継続する
  - そうしないと、完全にAIと運命共同体になる
    - つまり、AIのトレンドが移り変わると共に寿命が終わる可能性がある
  - AI以外の用途でも広がるようにしないと、プロセッサの生き残りを考えていくうえではまずい
  - その前に、販路を広げるためにもHPCユーザー向けの環境を作るべき

賢いコンパイラは開発できるのか？

- 実装コストは高いが、理論上は可能
- MN-Coreの動作が決定論的であり、事前に予測評価できるため
  - むしろこれによって、コンパイラの最適化精度は上がるはず

重要な問題の一つ：レジスタ割付

- MN-Coreにはローカルメモリも含め、複数種類のレジスタが存在
  - 書き込み後に読み出しできる動作サイクル数、物理的な個数が異なる
  - 制約の中で、遅延がなるべく最小限になるように使わないといけない
    - そうしないと性能が出ない
    - 結構クリティカルな問題
- ここは遠藤さん@産総研が既に解決済み
  （一昨年10月のCPSセミナーで発表）
  - これは組合せ最適化問題
  - シミュレーテッドアニーリングで
    上手くいくことがわかった
  - 他の箇所も同様に組合せ最適化問題として
    うまく解ける可能性がある（特に通信周り）



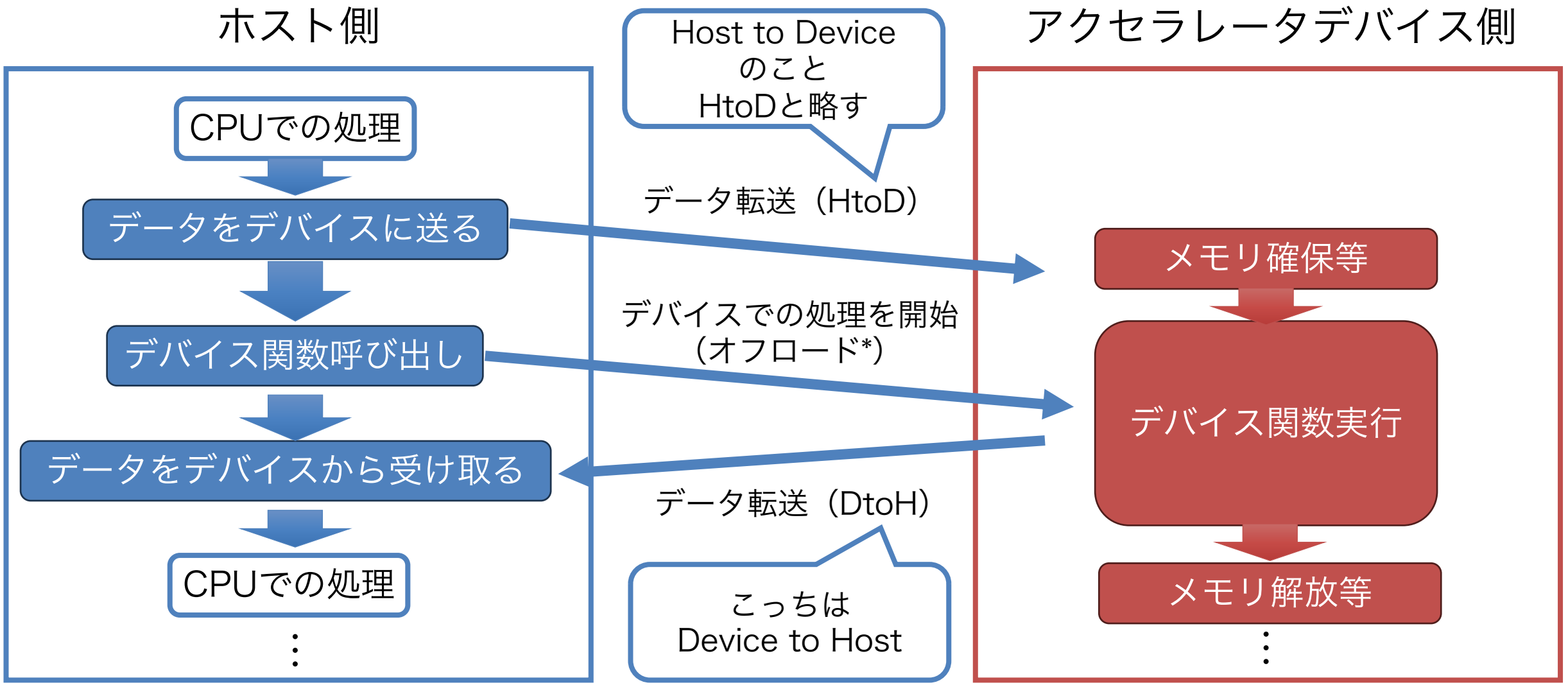
▲レジスタの種類(T/R/M)

by 遠藤さん@産総研

汎用プログラミング言語：MNCL

- GPUには汎用言語としてCUDAやOpenCLがある
  - ただし、そのままだと分散メモリ型の記述や階層メモリ確保などに難がある
- これらのMN-Coreバージョンが MNCL
  - まだ仕様検討段階
  - コンパイラも順次開発中
- C-likeとFortran-likeな仕様ができる予定
- BLASなどのライブラリも用意される予定

アクセラレータの基本的な使い方



*オフロード：ホストの一部処理をアクセラレータデバイスで実行すること

MNCLの記述イメージ

ホストコード

```
#include "MN_acc.h"
int main()
{
float array1[32][32][32];
float array2[32][32][32];
float * * * array3;

array3 = ((float * * * )(malloc((((sizeof(float)) * (32)) * (32)) *
(32))));
{
int i;
for(i = (0); i < (32); i++) {{{
int j;
for(j = (0); j < (32); j++) {{{
int k;
for(k = (0); k < (32); k++) {{
(array1[i][j][k]) = ((float)1.0f);
(array2[i][j][k]) = ((float)1.0f);
(*((*((*(array3 + i)) + j)) + k)) = ((float)1.0f);
}}}}}}}}}

_ACC_MN_runtime_HtoD(array1, 0, 32*32*32);
_ACC_MN_runtime_HtoD(array2, 0, 32*32*32);
_ACC_MN_runtime_HtoD(array3, 0, 32*32*32);

_ACC_MN_runtime_kernel_kick("main_kernel0");

_ACC_MN_runtime_DtoH(array1, 0, 32*32*32);
_ACC_MN_runtime_DtoH(array2, 0, 32*32*32);
_ACC_MN_runtime_DtoH(array3, 0, 32*32*32);
}
```

デバイスコード

```
__kernel void main_kernel0(
__global float*** _arg_array1,
__global float*** _arg_array2,
__global float*** _arg_array3
) {
    __private float array1[2][2][1];
    __private float array2[2][2][1];
    __private float array3[2][2][1];

    dist(array1, _arg_array1, 2*2*1);
    dist(array2, _arg_array2, 2*2*1);
    dist(array3, _arg_array3, 2*2*1);

    {
        int i;
        for(i = 0; i < 2; (i)++)
        {
            {
                int j;
                for(j = 0; j < 2; (j)++)
                {
                    {
                        array1[i][j][0] += array2[i][j][0] * array3[i][j][0];
                        {
                            float b = 2.0;
                            {
                                int l;
                        #pragma clang loop vectorize_predicate(enable)
                                for(l = 0; l < 4; (l)++)
                                {
                                    array1[i][j][0] *= b;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    collect(_arg_array1, array1, 2*2*1);
    collect(_arg_array2, array2, 2*2*1);
    collect(_arg_array3, array3, 2*2*1);
}
```

私の研究していること

OpenACCのコード例（オフロード対象処理の指定）

```
#pragma acc parallel loop
for (i = 0; i < N; i++) {
    out[i] = in1[i] * in2[i];
} // ↑ 既存のコードはそのまま
```

- OpenACCコンパイラの開発
  - CUDAやOpenCLですら人類には大変なので、世の中にはOpenACCという便利なAPIがある
  - OpenACC：コンパイラ指示文形式で追記するだけのアクセラレータ用API
    - コンパイラ指示文の他の例：OpenMP（CPUスレッド並列実行用）、XMP（ノード間並列用）
    - C、C＋＋、Fortranの規格がある
    - まずはCのコンパイラを実装予定
    - Fortranは実装予定だが、C++は未定
  - 簡単な分、細かい部分の指定は難しいが、部分的にMNCLで記述できるようにすることで解決予定
  - 分散メモリ周り（袖交換等）の指示は一部仕様を拡張する（XMP、HPF参考に）

OpenACCの記述イメージ

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

- ステンシル計算（差分法）の例
  - こんな感じで、ディレクティブ（太字）をCPUの逐次コードに追記するだけで、さっきの図の青背景と赤背景の処理ができる
  - 赤字は袖領域の指定と、袖交換の指示（独自拡張仕様）
  - 簡単に使える分、コンパイラ実装は大変

アプリの性能

- まだ汎用言語環境ができてないので、アセンブリやPyTorchでの評価のみ
- 姫野ベンチマークの実行効率 73.4%[1]
  - 先週発表があったMN-Core2の数字
  - アセンブリで記述
  - 驚異的数値（GPUでは高くても3%など）
    - FMA使用率等を考慮するとほぼ理論限界
- OpenFDTDもチューニング中ながら実行効率 2桁%[2]
  - こちらもアセンブリで記述

[1]安達知也, 牧野淳一郎, “MN-Core 2 での姫野ベンチマークの性能評価”, 第18回アクセラレーション技術発表討論会.
[2]坂本亮, 福重俊幸, “MN-Core 2 へのOpenFDTD の実装のエネルギー効率”, 第18回アクセラレーション技術発表討論会.

普及状況や今後について

FOCUS（神戸市）にMN-Core2導入決定

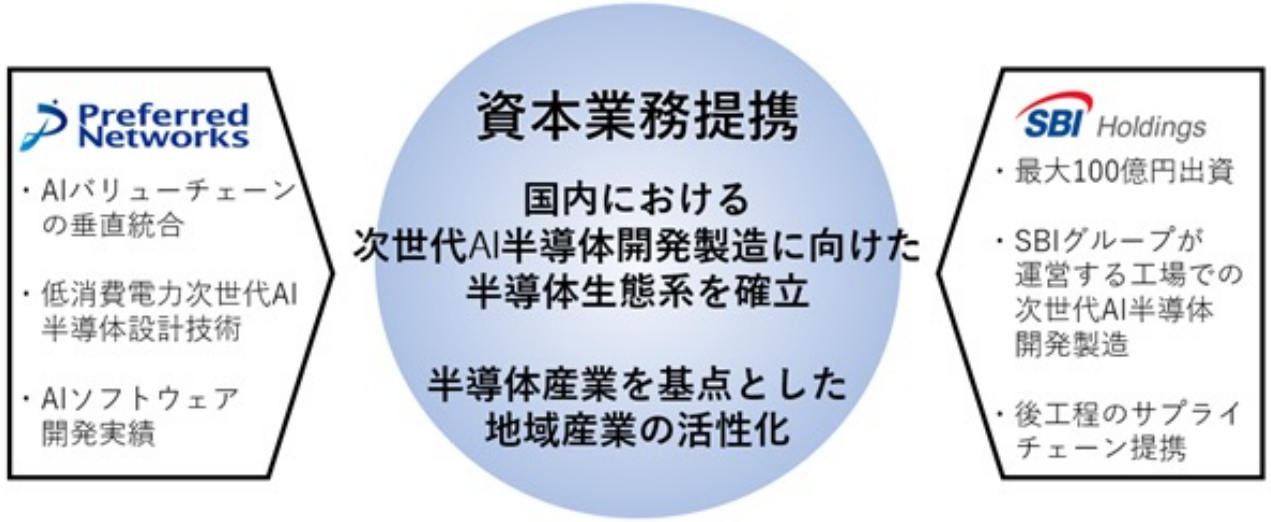
News Release 2024.09.10

PFNと計算科学振興財団、 AIプロセッサーMN-Core 2 搭載システムの導入に合意

（PFNサイトより）

- CPS向かい、 理研 計算科学研究センターの隣の施設が計算科学振興財団FOCUS
  - 共同利用施設みたいな感じ
- MN-Coreシリーズ初の市販導入事例
- 「アクセラレータ活用の並列プログラミング環境を直接体験可能なセミナー等を通して、利用拡大を推進する予定」と記事に書かれている
  - OpenACCもおそらくやる？
  - 汎用言語はまだなので当面はPyTorch限定？
- なんにしろ、これで着々と利用者拡大と普及が進んでいくと思われる

PFNがSBIと資本提携



（PFNサイトより）

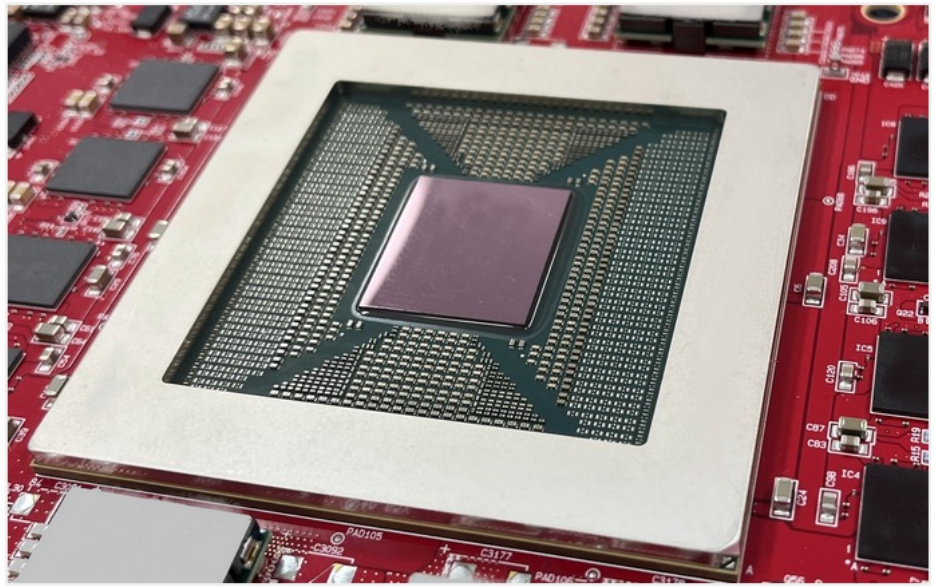
- SBIは投資会社
  - ネットバンクとかNISAで聞いたことがあるかも？
- たくさん投資してくれるらしい
  - 100億円はFSと比べても相当大きい
  - つまり、継続してプロセッサ開発や言語処理系周りの開発を続けられる

勉強会、コミュニティなど

（YouTubeより）

- FSをきっかけにユーザーコミュニティを広げようとしている
- MN-Core2の仕様やエミュレータを公開している
- MN-Core勉強会
  - PFNでやっている
  - 第一回の録画がYouTubeにある
    - https://www.youtube.com/live/U4HjE2S8wAY
  - 今後も継続予定
  - 参加すると入れるDiscordコミュニティがある
    - MN-Coreについて、社員とかに質問可能
    - いち早くMN-Coreの情報に触れられる

コンテスト



Event 2024.08.05

賞金付きプログラミングコンテストMN-Core Challengeを初開催

（PFNサイトより）

- MN-Coreチャレンジ
  - アセンブリを書くコンテスト
  - 先週第一回が終わったが、こちらも継続予定
  - 問題やチュートリアルはまだ公開されている
    - アセンブリが書けるとハードの理解も進むので、興味が湧いた人は是非アクセス