10/20 EVF
セッション #1－2
14:00～14:40

exa
EXA CORPORATION

スーパーコンピュータ「富岳」のシステム概要と性能検証事例

京 K computer から 富岳 Fugaku へ

駅名
変更

京 から 富岳 へ

K computer

Fugaku

2011年6月

10.5PFLOPS

１２MW

50倍※以上の
性能向上

※ PFLOPS値の比較

2020年6月

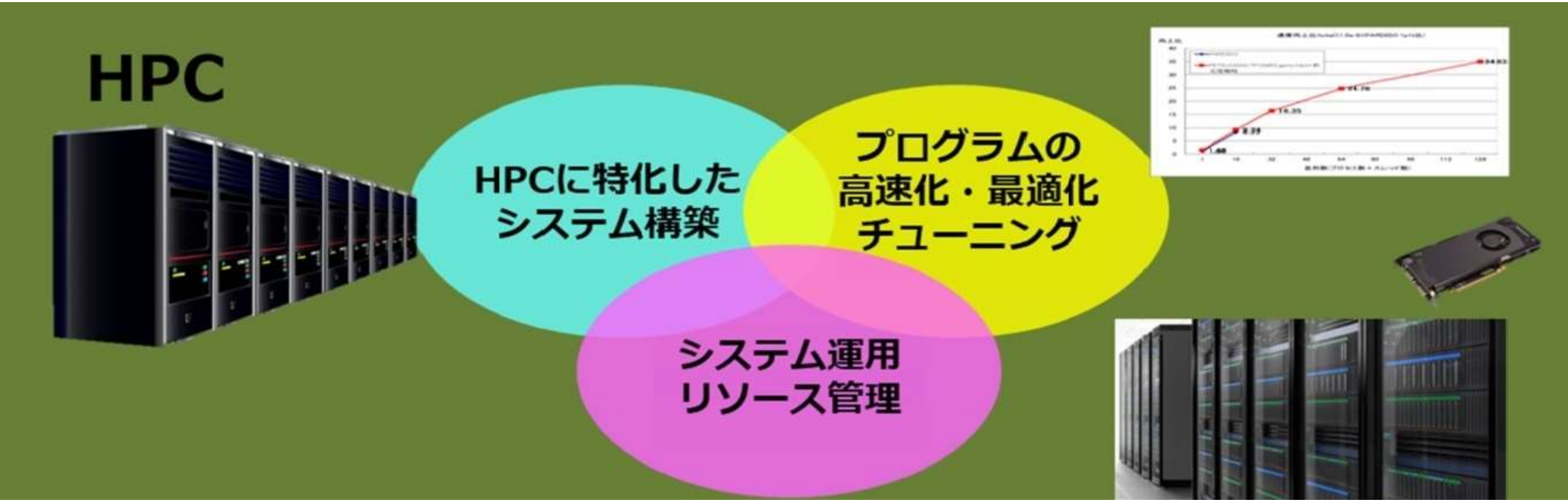
537PFLOPS

３０MW

Agenda

0.
1. HPCへの取り組み
2. スーパーコンピュータ「富岳」とは
3. 高速化の概要
4. 「富岳」性能検証事例
5. 「富岳」ビジネス

1.　HPCへの取り組み



エクサのHPCソリューション

- 各種HPCプラットフォームのご提案、導入、構築
- ジョブスケジューラ導入構築
- プログラム高速化・最適化チューニング
- 高速分散ファイルシステム構築 、階層型ストレージ構築
- 運用支援サービス

HPCソリューション経緯と高速化への取り組み

1980年代後半～1990年代中盤

- 3Dグラフィックス処理の高度化に関してお客様をサポート
- **プログラムのベクトル高速化**に関して、お客様へのサポートを実施
- コンベックス社にSEを派遣し、MPPシステムやリアルタイムOSの共同開発も実施

1990年代後半～2000年代後半

- シリコングラフィックス社、コンベックス/HP社の超並列システムの販売
- **並列処理を用いたプログラム高速化**に関して、お客様へのサポートを実施
- 障害対応だけでなく、大規模HPCシステムとしての運用に合わせた改善を提案、具現化
- GPGPUの黎明期に実システムでユーザをサポート

2010年代以降

- 高速共有ファイルシステムの構築
- 中規模～超大規模HSMシステムの構築
- 各種研究機関向け**プログラム高度化支援（IAクラスタ、XeonPhi）**

私とHPC

1989 → 1990 → 1991 → 1992 → 1993 → 1994 → 1995 → 1996 → 1997

Convex/HP社製ミニスーパーコンの開発/サポートを担当

・Convex C220（1988年～）
ベクトル型ミニスパコン
（ハイCPベストセラー機）

・HP Exemplar SPP1200（1994～
RISC並列型ミニスパコン
（ベクトル機から並列機へ）

私とHPC

2000 2001 2002 2003 2004 2005 2006 2007 2008 2009

R&D分野各種アプリケーション　技術サポート、開発、解析受託

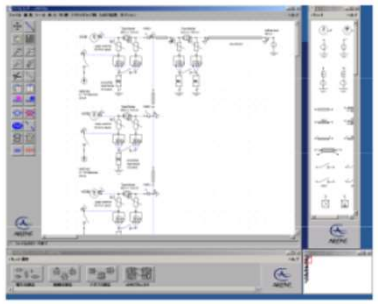
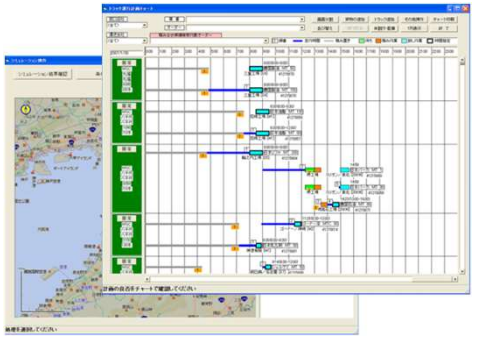


図9　ARENEによるウィンドファームモデル記述[5]

EDF製電力系統解析ソフト
ARENE

地球シミュレータ
大気大循環モデル

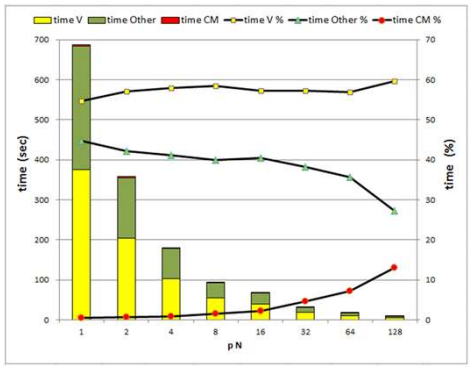


配送経路最適化シミュレータ
CTS

私とHPC

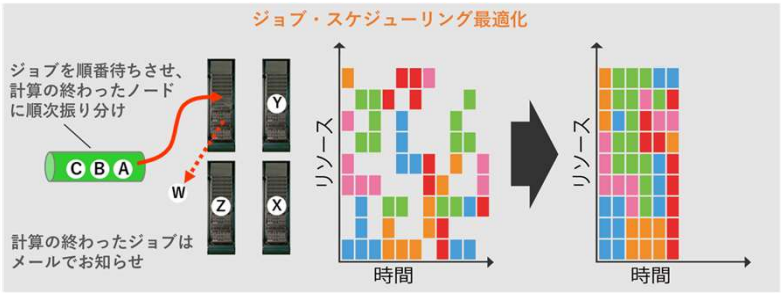
2010 2011 2012 2013 2014 2015 2016 2017 2018 2019

並列フレームワーク開発、プログラム高速化、HPC基盤構築



IBM/理研開発フレームワーク「RIVER」

国立大学向けプログラム高速化



ジョブ管理ソフトウェア「Spectrum LSF」

「富岳利用環境整備 ～みんなの富岳～」

EXA CORPORATION

RIKEN

R-CCS

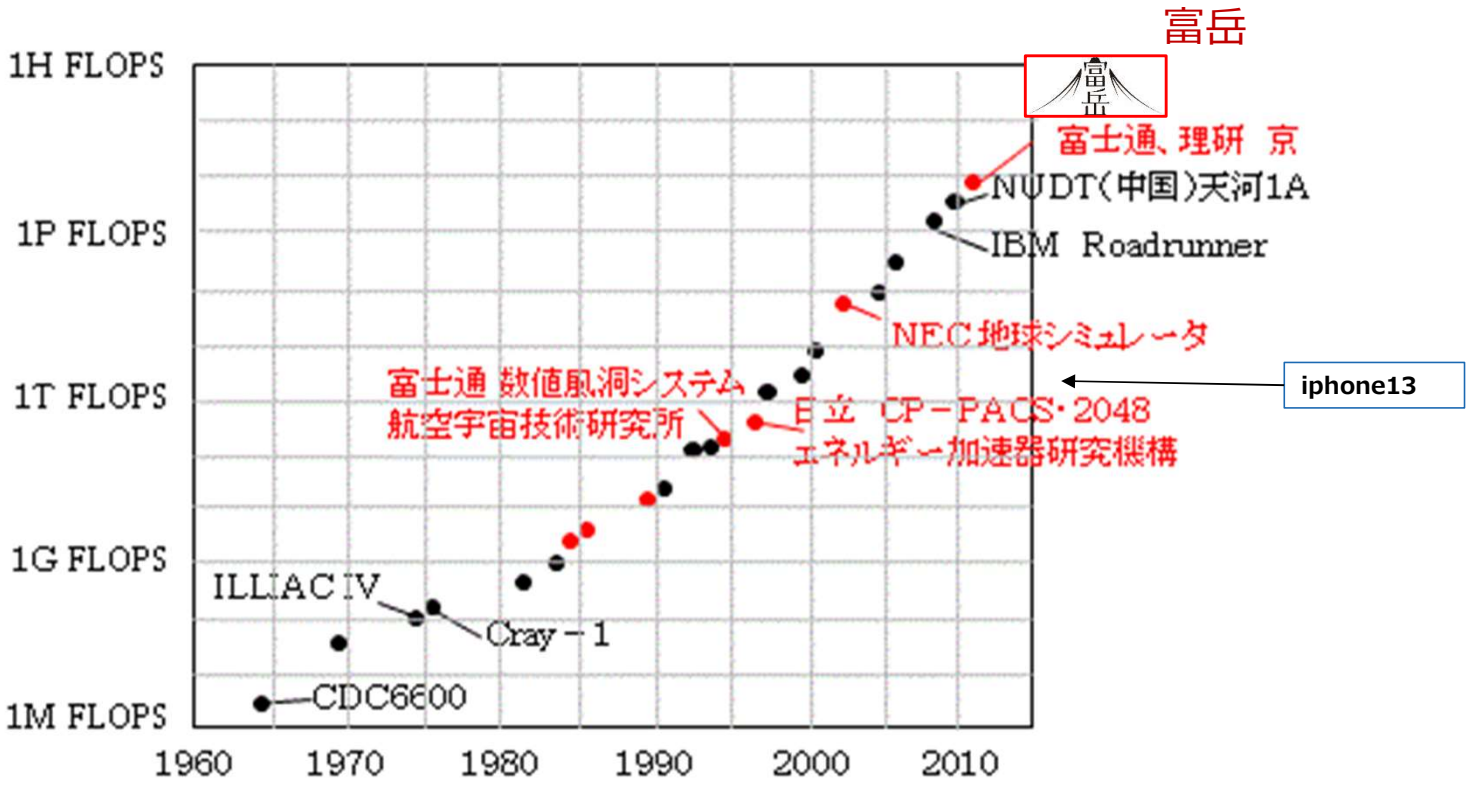
理化学研究所
計算科学研究センター
RIKEN Center for Computational Science

2．スーパーコンピュータ「富岳」とは？

スーパーコンピュータのお話

出典：https://www.r-ccs.riken.jp/outreach/topics/20211007-1/

スーパーコンピュータのお話



スーパーコンピュータの処理速度の推移(赤は国産機)
出典:http://www.kogures.com/hitoshi/history/super-computer/

「富岳」が目指したゴール

“「富岳」という名称の由来

”富士山”の異名である「富岳」という名前もこのマシンの特長を表しています。富士山の高さがスーパーコンピュータ「富岳」の性能の高さを表し、また富士山の裾野の広がりがスーパーコンピュータ「富岳」のユーザーの拡がりを意味しています。“

出典　：https://www.r-ccs.riken.jp/fugaku/about/

「富岳」が目指したゴール

富岳がターゲットとすべき9つの目標分野

1. 革新的な創薬
2. 個人向けの薬品や予防薬の開発
3. 地震や津波による被害の低減
4. 観測によるビッグデータを使った環境変動の予測
5. 高効率のエネルギー生成、変換/貯蔵、使用
6. 革新的なクリーンエネルギーシステム
7. 高性能の新機能デバイス
8. 近未来の製造業のための革新的な設計、製造プロセス
9. 基本的な物理法則と宇宙の進化

「富岳」が目指したゴール

»実アプリケーションでの性能
»使いやすいシステム
»すぐれた電力効率

1. スーパーコンピュータ「富岳」とは？

3期連続
スパコン性能同時4冠達成

Top500, HPCG, HPC-AI, Graph500

アプリケーションファースト
実アプリケーション性能を重視

汎用プロセッサ
スマホでも利用されるArmアーキテクチャ

高効率

	IAサーバ		富岳
性能	３０万台	＝	１台
電力	２００MW	＞	30MW

1. スーパーコンピュータ「富岳」とは？

全体構成

CPUあたりコア数： **48**
ノード当りCPU数： **1**
ラック当りノード数： **384ノード**
計算ラック数　：**432ラック**
計算ノード数　：**158,976ノード**

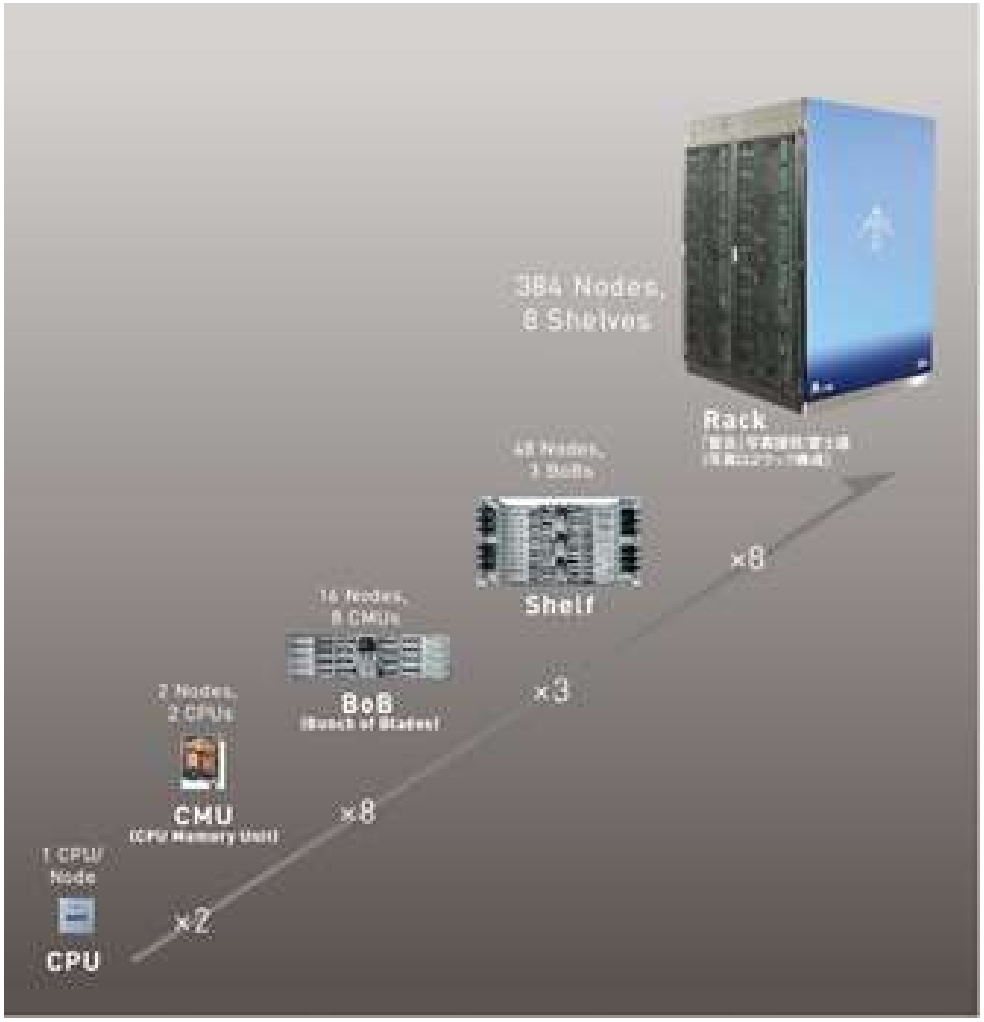
総コア数： 7,630,848 コア

高い単体ノード演算性能
3.3792TFlops/ノード（倍精度）

高帯域メモリ
HBM2 32GB/ノード 1TB/S

広帯域インターコネクト
Tofu D

大規模ノード
158,976ノード



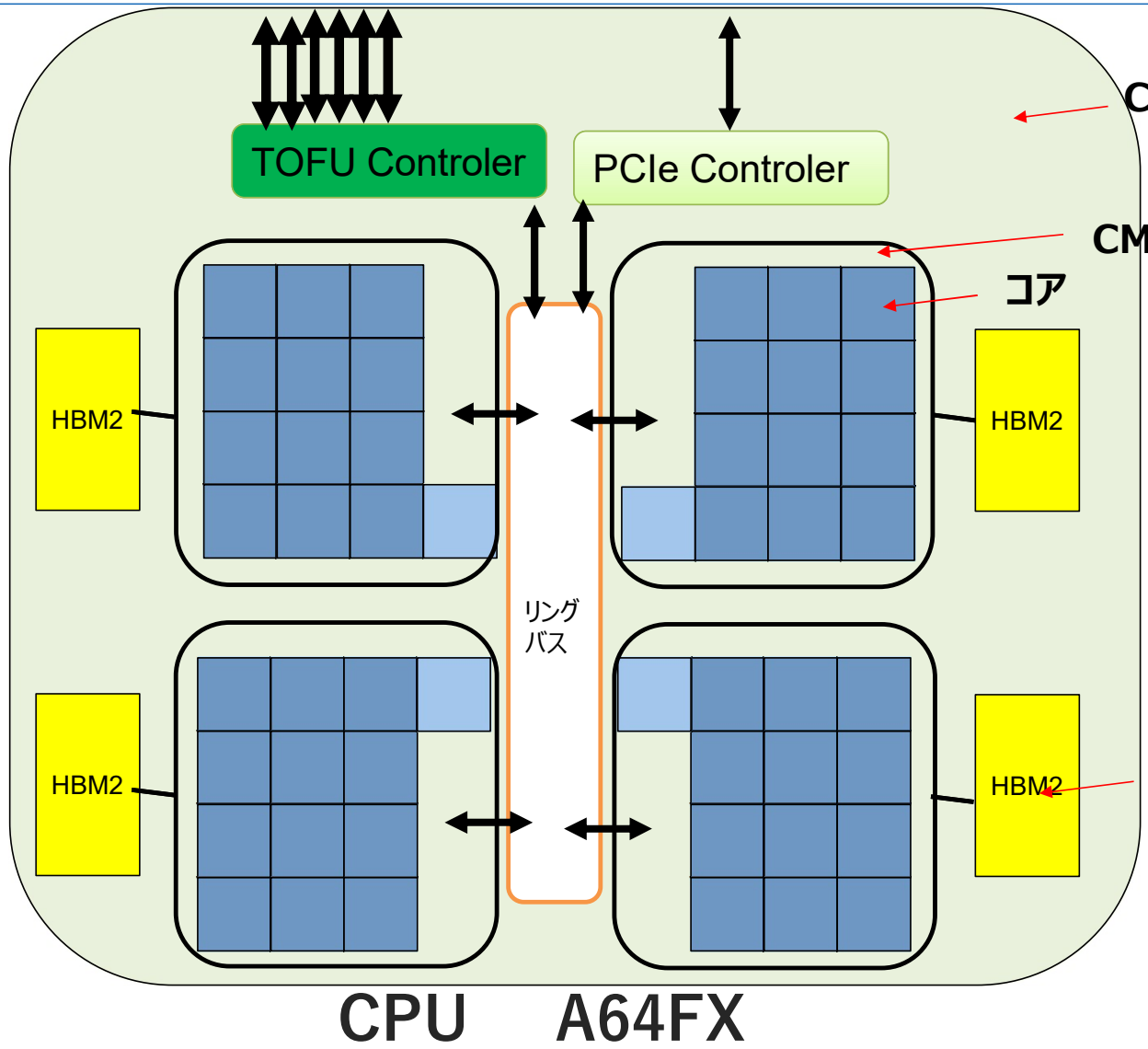
スーパーコンピュータ「富岳」とは？

- 「ノード」とは、
  - ひとつのメモリ空間を共有する複数の演算コアから構成される
  - ノードにはそれぞれひとつのOS（RHEL8）が稼働している
  - ノード間は高速なネットワークで接続されている。
  - スパコンではログイン用のノードと計算用の多数のノード群から構成されるのが一般的。

- ログインノードと計算ノード
  - 富岳のログインノード（CPUはIntel）は6台
  - 富岳の計算ノード（CPUはA64FX）は158,976ノード

- ノード間を接続する高速ネットワーク
  - 一般的にはInfiniband
  - 富岳ではTofu-Dインターコネクト
    - X-Y-Zの3次元 x 双方向同時通信＝6次元トーラス

スーパーコンピュータ「富岳」とは？



CPU（＝４CMG　＝総計算コア 48)

CMG（＝１３コア　＝計算コア12+補助コア1）

コア

ARMアーキテクチャ
+SVE拡張

HBM2（＝８GB)　（CPU全体では32GB）

1. スーパーコンピュータ「富岳」とは？

- **CPU単体は一般的な性能＋α、メモリバンド幅は非常に大きい**

  - **演算性能**

    - **Flops値**
      - 1秒当たりの浮動小数点演算回数　　（**A64FXでは　2.7 TFlops　（Xeonでは2TFlops）**）

  - **メモリバンド幅**

    - 1秒あたりのメモリデータ転送量　　（ **A64FXでは　1TB/s　（Xeonでは140GB/s）**）

HPCアプリケーションは多くがメモリ律速

スーパーコンピュータ「富岳」とは？

- SVEとは
  - Scalable Vector Extension
    - 512ビット長のSIMD演算を行う拡張機能

- SIMD演算とは
  - Single Instruction Multiple Data　演算
    - 1命令で複数のデータを同時に処理する

- 512ビット長のSIMD演算とは
  - 512ビット＝64バイト＝　倍精度浮動小数点（8バイト長）を同時に8個処理できる。

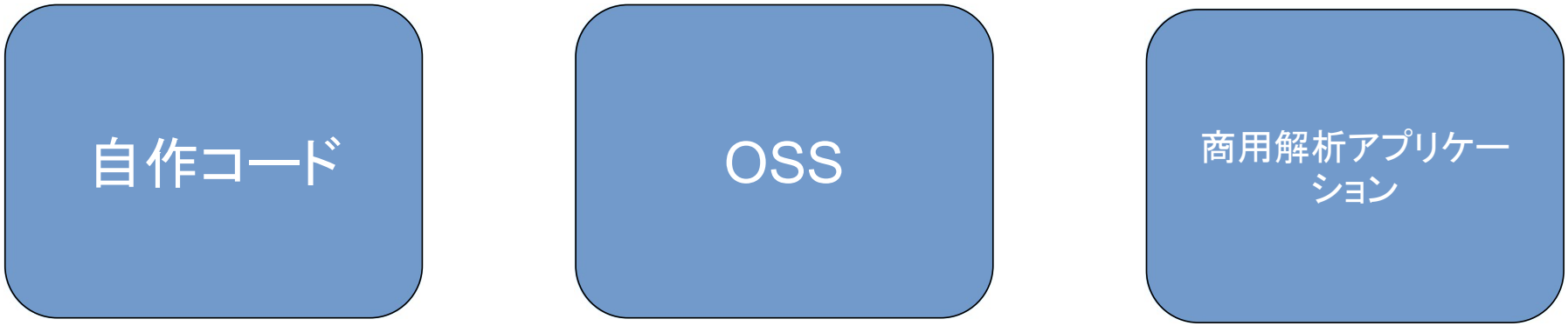
スーパーコンピュータ「富岳」とは？

IAサーバで稼働するプログラムは
再コンパイルで富岳での稼働が可能

【OS】Redhat 8.2

【コンパイラ】富士通コンパイラ、GNUコンパイラ

Fortran、C、Python



スーパーコンピュータ「富岳」とは？

- 「富岳」で利用可能な主なOSS

シミュレーション（分子動力学）
- GENESIS
- GROMACS
- LAMMPS
- N2P2

シミュレーション（量子化学）
- ABINIT-MP
- NTChem

シミュレーション（物性物理）
- Quantum ESPRESSO

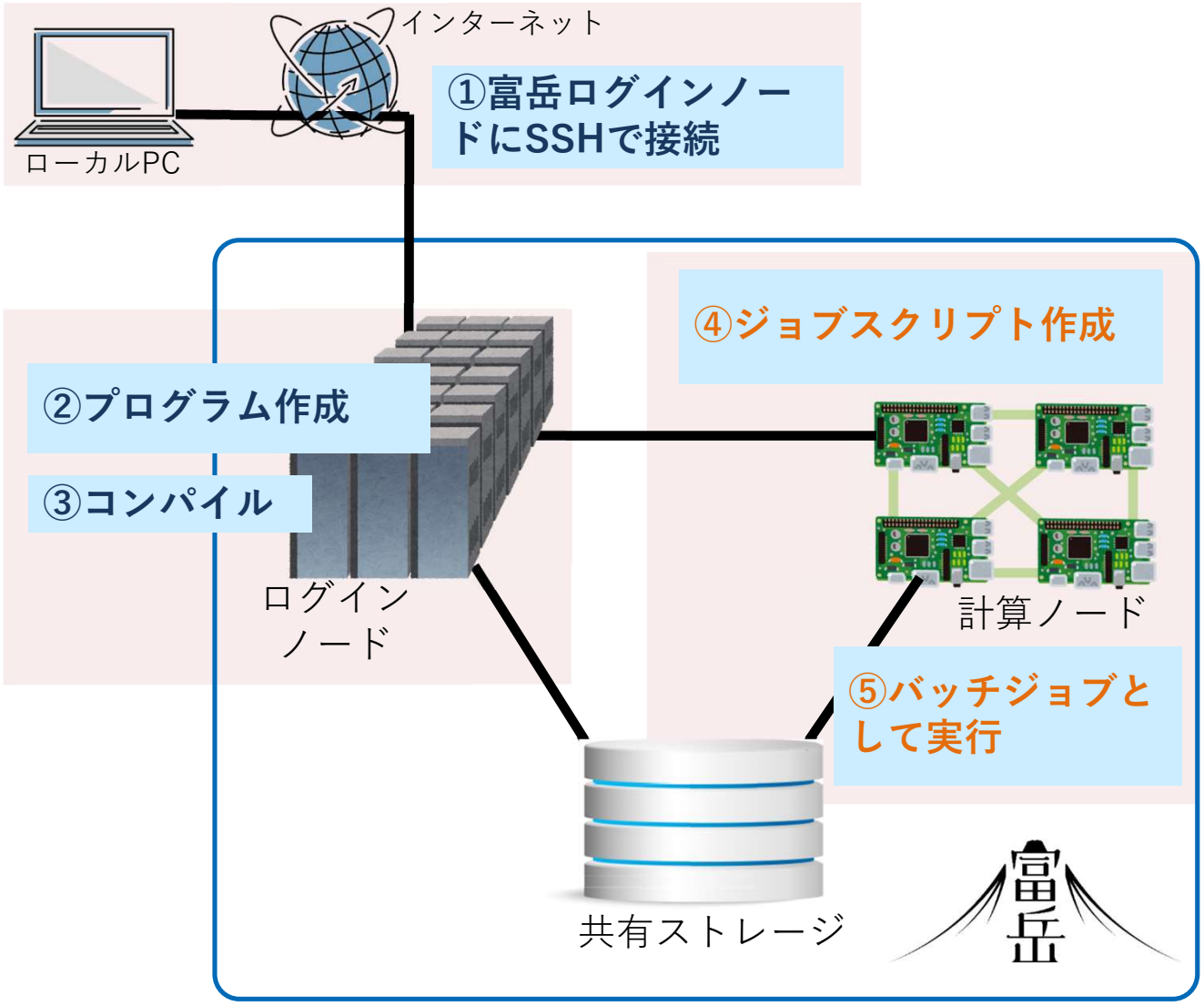
シミュレーション（流体解析）
- OpenFOAM

シミュレーション（気象／気候）
- SCALE

データサイエンス（機械学習）
- Chainer
- DNNL_aarch64
- Horovod
- Keras
- PyTorch
- scikit-learn
- TensorFlow

データサイエンス（統計解析/データ分析）
- pandas
- R

富岳利用の流れ　ログイン～コンパイル～ジョブ投入



Sample.sh

```
#! /bin/bash
#PJM -L  "node=1"              # 割当ノード数
#PJM -L  "rscgrp=small"        # リソースグループの指定
#PJM -L  "elapse=2:00:00"      # 経過時間制限
#PJM -s                        # 統計情報出力

./himenoBMTxp < In.txt         #実行
```

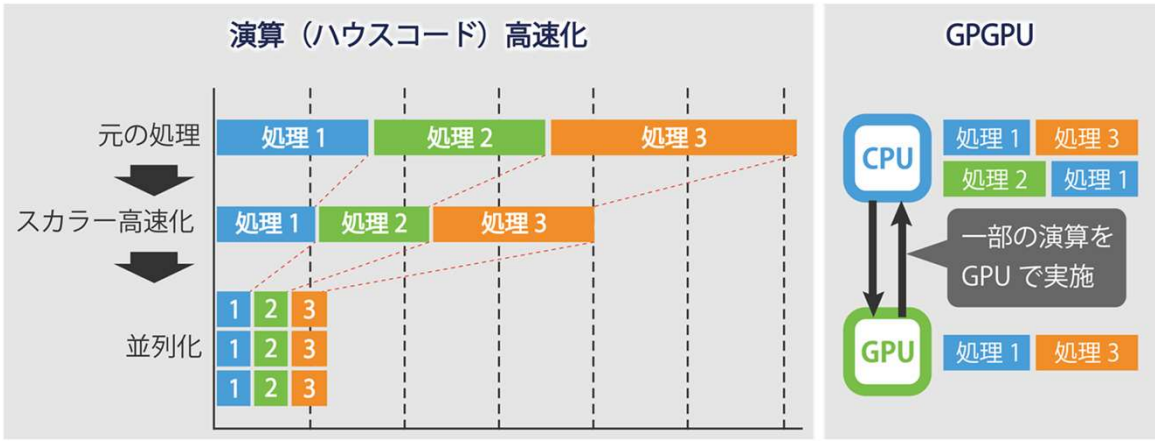
$ **pjsub ./Sample.sh**
[INFO] PJM 0000 pjsub Job 6958584 submitted.

⇒計算が終わるとファイルが出力される
Sample.sh.6958584.err
Sample.sh.6958584.out
Sample.sh.6958584.stats

3．高速化の概要

高速化とは

- 単体CPUにおける性能向上
- 並列実行による性能向上

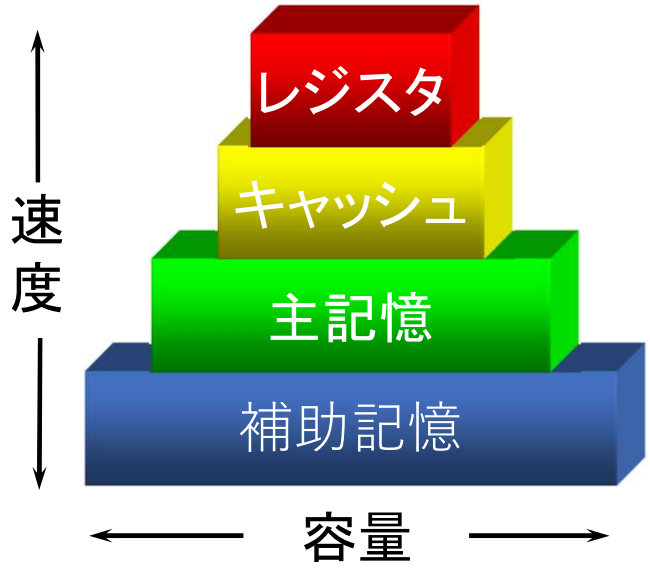


単体CPUにおける性能向上

≒　メモリアクセスの高速化

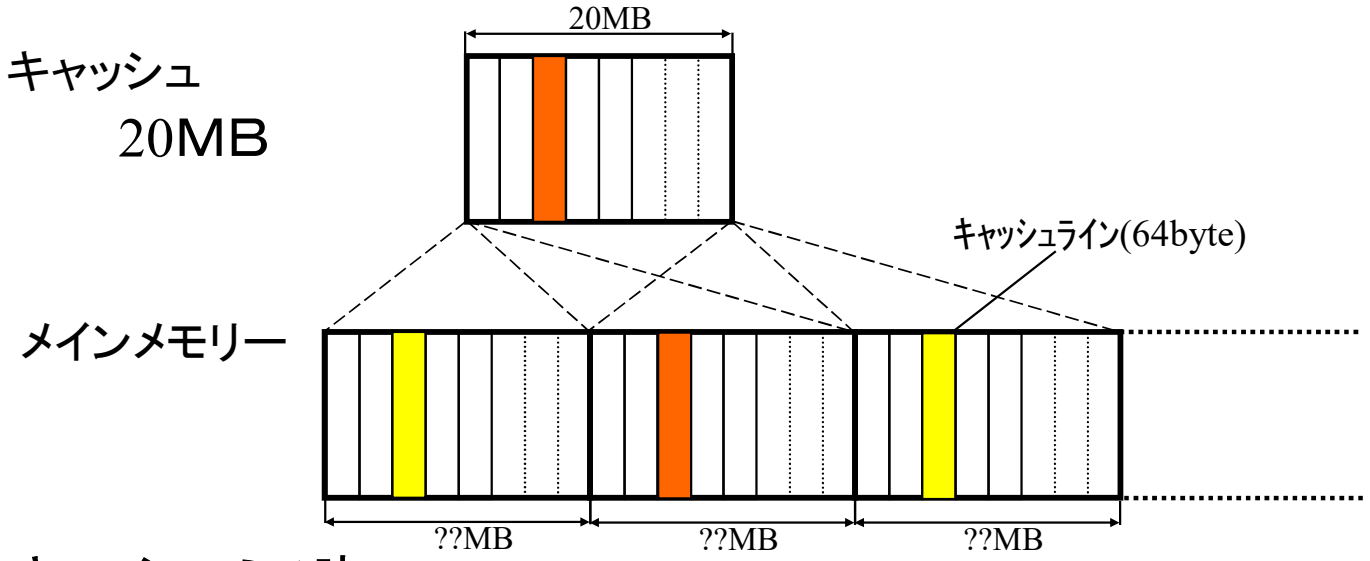
メモリアクセスの高速化

- 階層構造メモリアーキテクチャ



メモリアクセスの高速化

キャッシュライン



キャッシュ・ミス時

1. 次の式で対応アドレスを計算

   (Cache location) = (Virtual memory Address) MOD (Cache size)

   ※ ダイレクトマップド・キャッシュの場合

2. 参照データを含むキャッシュライン（★byte）をデータキャッシュに転送

メモリアクセスの高速化

実例

チューニング前

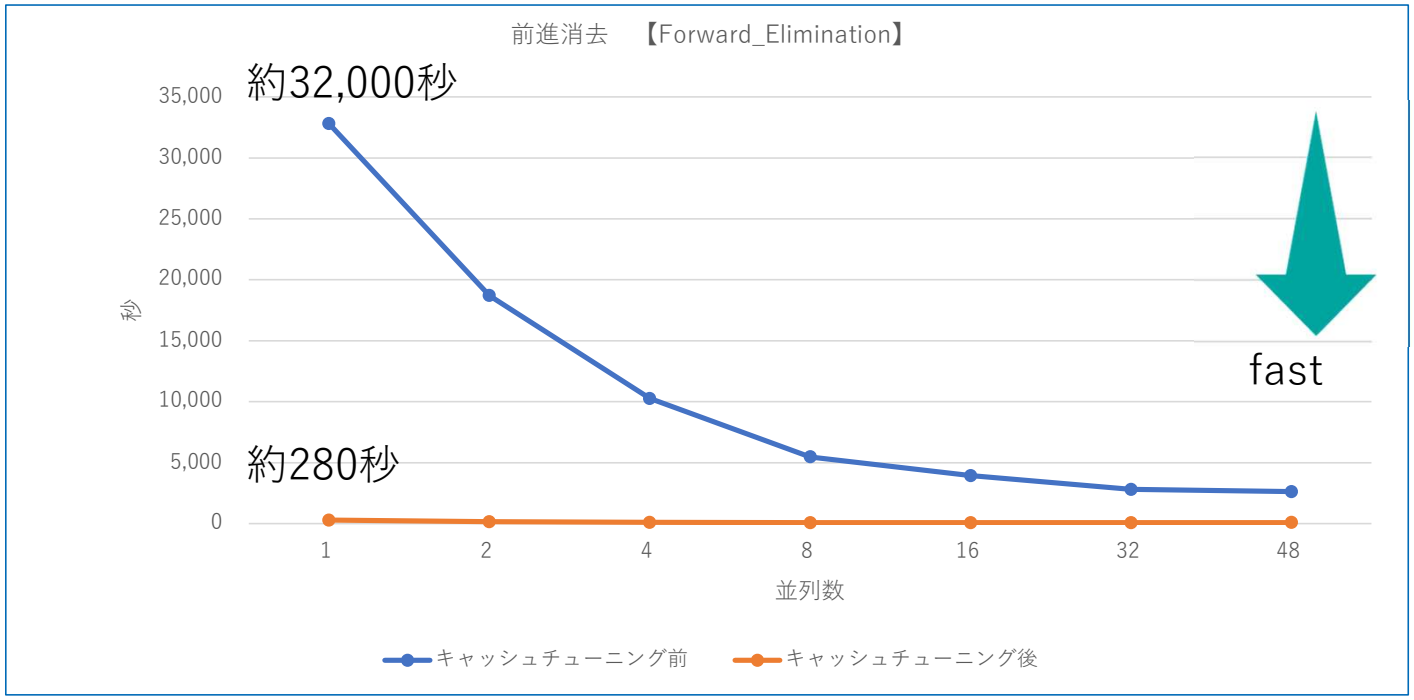
```
X(dim,Col)=B(dim,Col)/A(dim,dim)
do k=dim-1,1,-1
   do j=k+1,dim
      temp=temp+A(k,j)*X(j,Col)
   end do
   X(k,Col)=(B(k,Col)-temp)/A(k,k)
end do
```

行と列を入れ替える

チューニング後

```
      X(dim,Col)=B(Col,dim)/A(dim,dim)
do k=dim-1,1,-1
   do j=k+1,dim
      temp=temp+A(j,k)*X(j,Col)
   end do
   X(k,Col)=(B(Col,k)-temp)/A(k,k)
end do
```

キャッシュミスを軽減し、CPUへの転送速度を向上



32000秒→280秒

114倍の高速化

メモリアクセスの高速化

言語で異なる多次元配列データの並び方

C

```
float   a[3][3];
```

[0][0]	[0][1]	[0][2]	[1][0]	[1][1]	[1][2]	[2][0]	[2][1]	[2][2]	

Fortran

```
DIMENSION   A(3,3)
```

(1, 1)	(2, 1)	(3, 1)	(1, 2)	(2, 2)	(3, 2)	(1, 3)	(2, 3)	(3, 3)	

メモリアクセスの高速化

AOS（構造体の配列）とSOA（配列の構造体）

AOS

```
struct position  {   float x ;
                     float y ;
                     float z ; }  p[3];
```

p[0].x	p[0].y	p [0].z	p [1].x	p[1].y	p[1].z	p[2].x	p[2].y	p[2].z	

SOA

```
struct  position {   float x[3] ;
                     float y[3];
                     float z[3] ; }  p;
```

p.x[0]	p.x[1]	p.x[2]	p.y[0]	p.y[1]	p.y[2]	p.z[0]	p.y[1]	p.z[2]	

並列実行による性能向上

- OpenMPによるスレッド並列
- MPIによるプロセス並列
- OpenMP＋MPIによるハイブリッド並列
- （GPUを用いた並列）

4. 「富岳」性能検証事例

高速化作業の流れ

- ホットスポット解析
- 単体CPUでの高速化
- OpenMP
- MPI

プログラム内でCPU時間を多く消費する部分(ホットスポット)を特定

単体CPUでの高速化
（≒メモリアクセスの効率化）

OpenMPによるスレッド並列化

MPIによるプロセス並列化

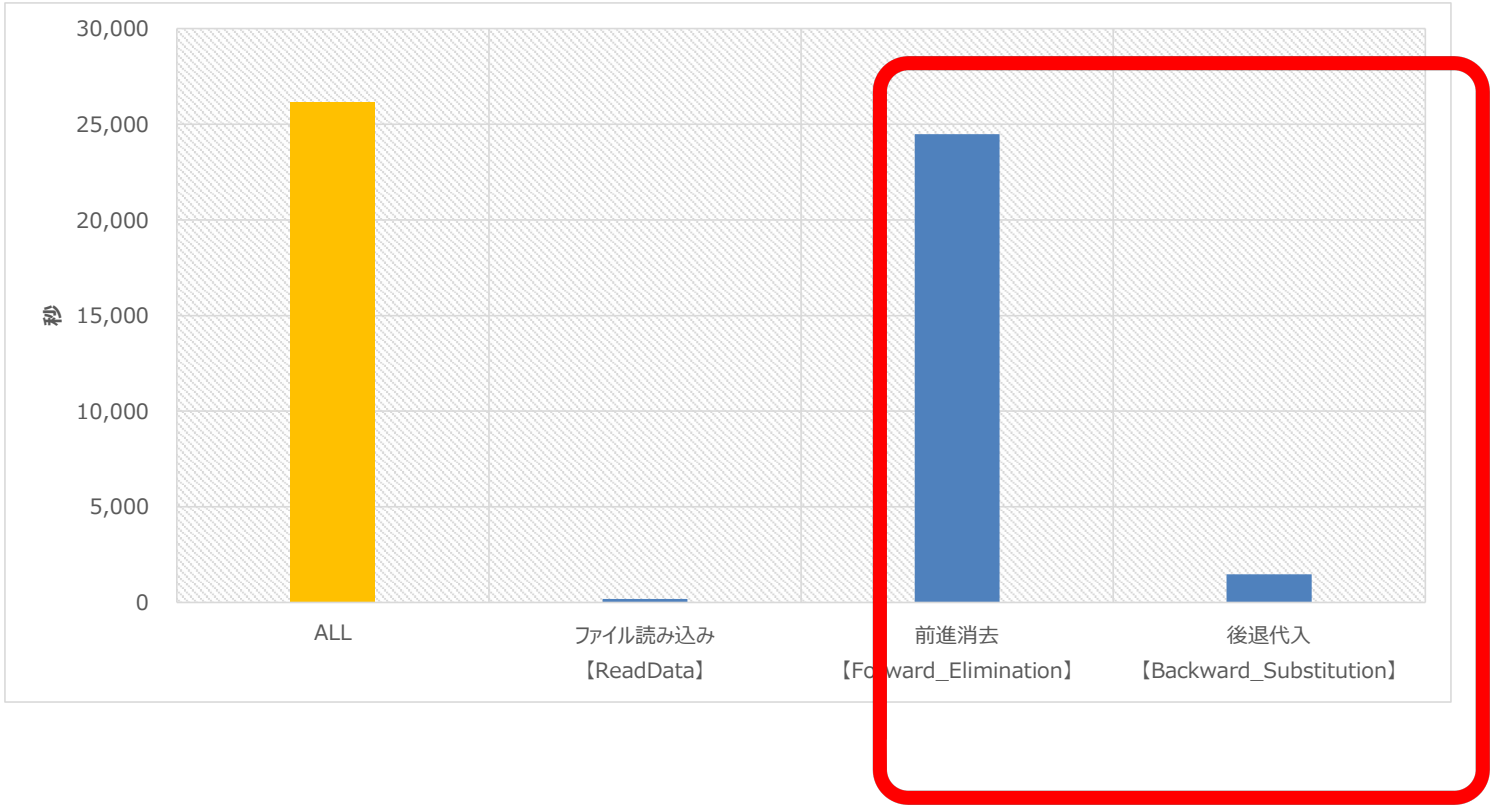
高速化作業の流れ（ホットスポット解析）

解析ツールでプログラムの各箇所の実行時間を集計

・プロファイラ

・時間計測ルーチン

プログラム内でCPU時間を多く消費する部分（ホットスポット）を特定



- ホットスポット解析
- 単体CPUでの高速化
- OpenMP
- MPI

高速化作業の流れ　（単体CPUでの高速化）

実例

チューニング前

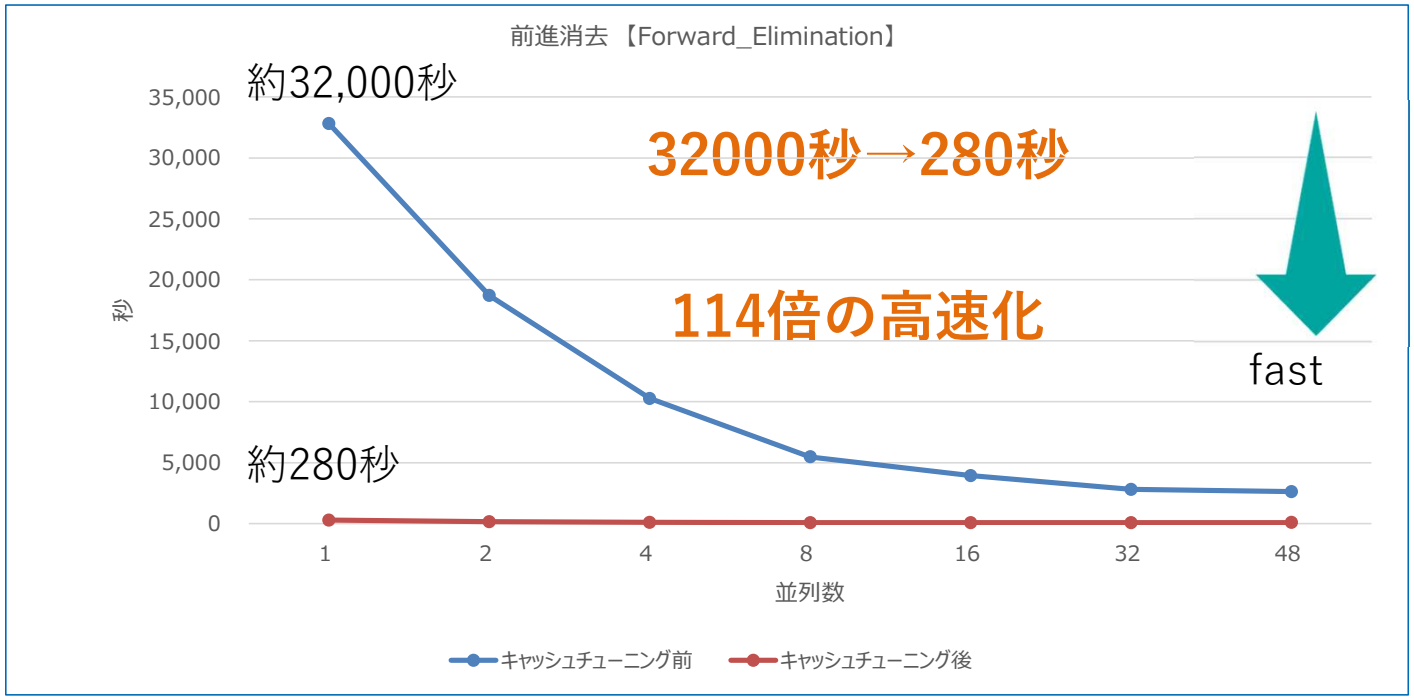
```
X(dim,Col)=B(dim,Col)/A(dim,dim)
do k=dim-1,1,-1
   do j=k+1,dim
      temp=temp+A(k,j)*X(j,Col)
   end do
   X(k,Col)=(B(k,Col)-temp)/A(k,k)
end do
```

行と列を入れ替える

チューニング後

```
      X(dim,Col)=B(Col,dim)/A(dim,dim)
do k=dim-1,1,-1
   do j=k+1,dim
      temp=temp+A(j,k)*X(j,Col)
   end do
   X(k,Col)=(B(Col,k)-temp)/A(k,k)
end do
```

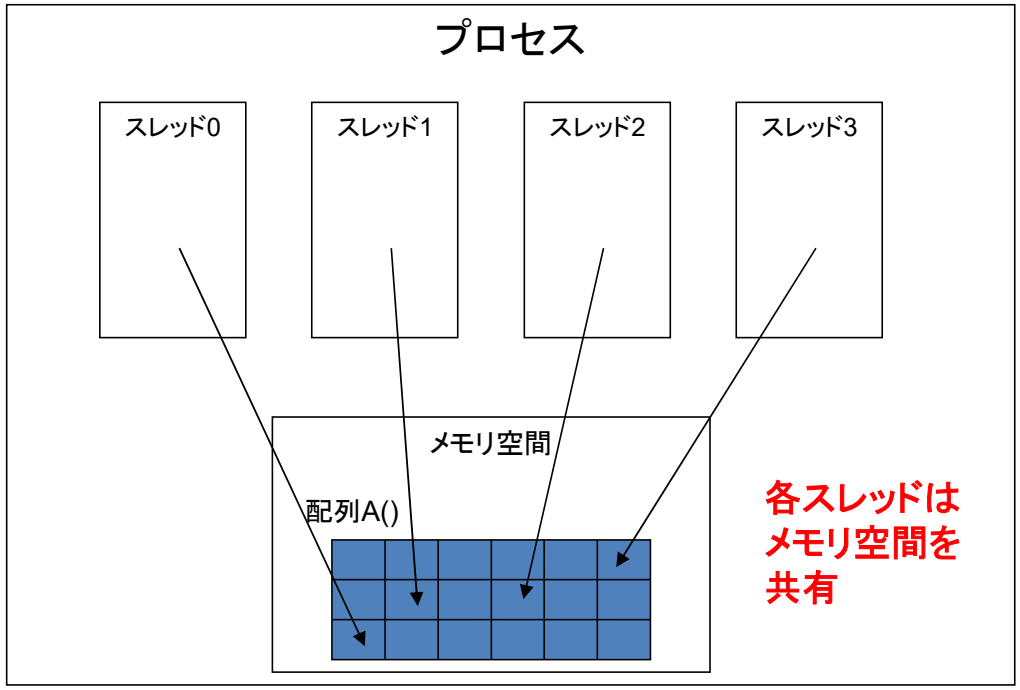
キャッシュミスを軽減し、CPUへの転送速度を向上



- ホットスポット解析
- **単体CPUでの高速化**
- OpenMP
- MPI

高速化作業の流れ　（OpenMPによるスレッド並列化）

OpenMPは同じメモリを共有する複数の「スレッド」が計算を分担して実行する方法です。



- ホットスポット解析
- 単体CPUでの高速化
- **OpenMP**
- MPI

高速化作業の流れ　（OpenMPによるスレッド並列化）

プログラム抜粋

```
!       ======= Backward Substitution =======
!$omp parallel do private(Col,k,temp,j)
        do Col=1,dim
           X(dim,Col)=B(dim,Col)/A(dim,dim)

           do k=dim-1,1,-1
              temp=0.0d0
              do j=k+1,dim
                 temp=temp+A(k,j)*X(j,Col)
              end do
              X(k,Col)=(B(k,Col)-temp)/A(k,k)
           end do
        end do
```

コメント文を挿入するだけ
プログラムは変えない

このdoループが複数のコアで分割されて平行実行される。

コンパイルコマンド

```
$ frtpx –Kfast -Kopenmp -o himenoBMTxp himenoBMTxp.f90
```

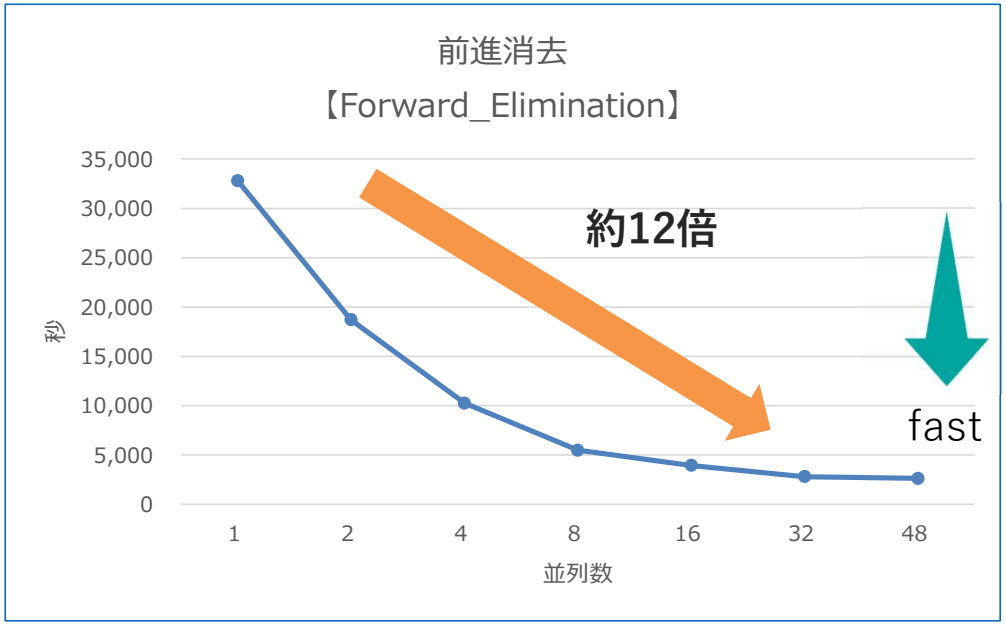
実行例

```
$ export  OMP_NUM_THREADS=48
$ himenoBMTxp
```

OMP_NUM_THREADS
環境変数で並列度を指定

高速化作業の流れ　（OpenMPによるスレッド並列化）

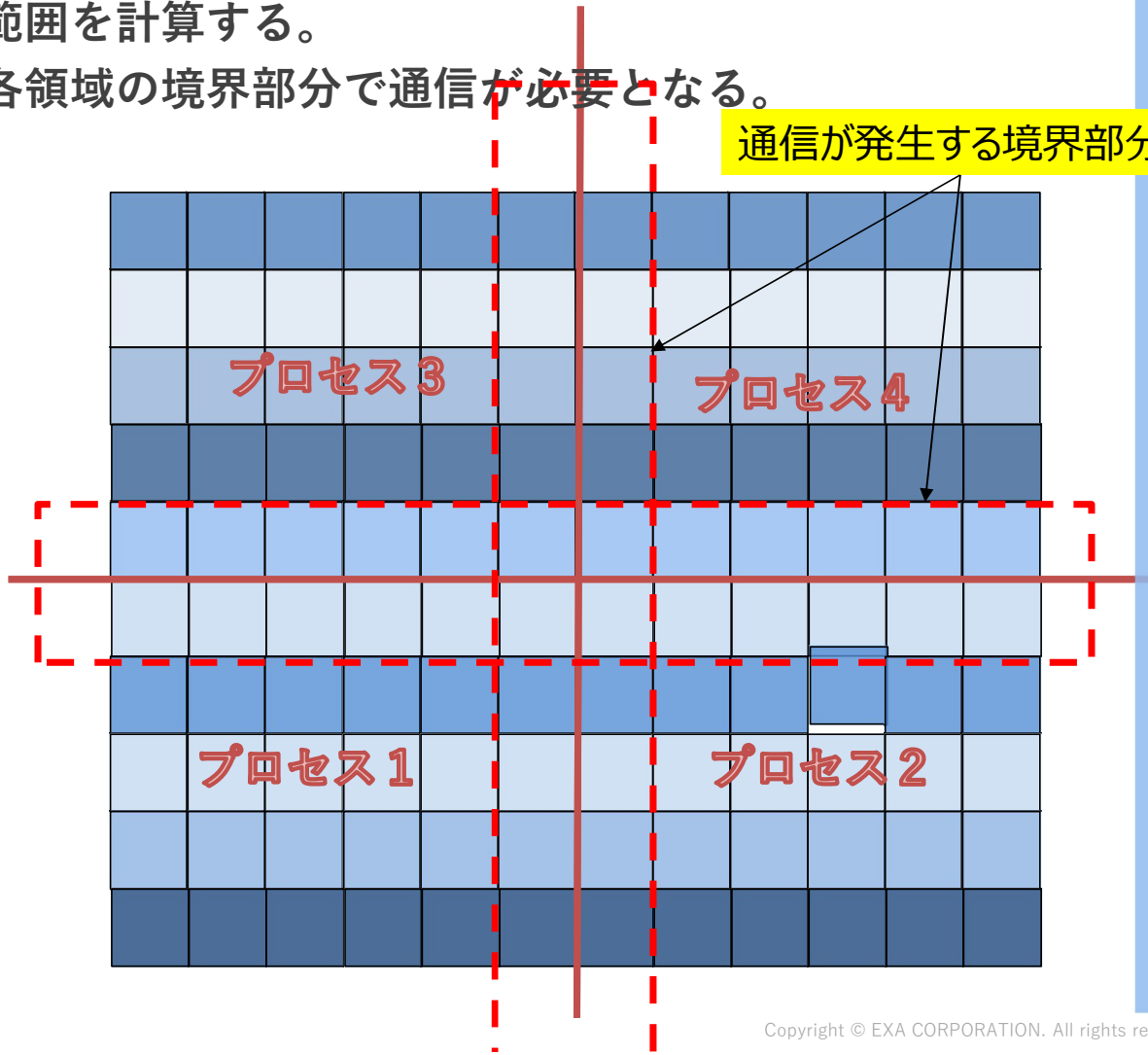
検証結果



OpenMPで並列数を増やすと高速化したが、頭打ちになる
OpenMPによる並列数の最大値はA64FXのコア数である48

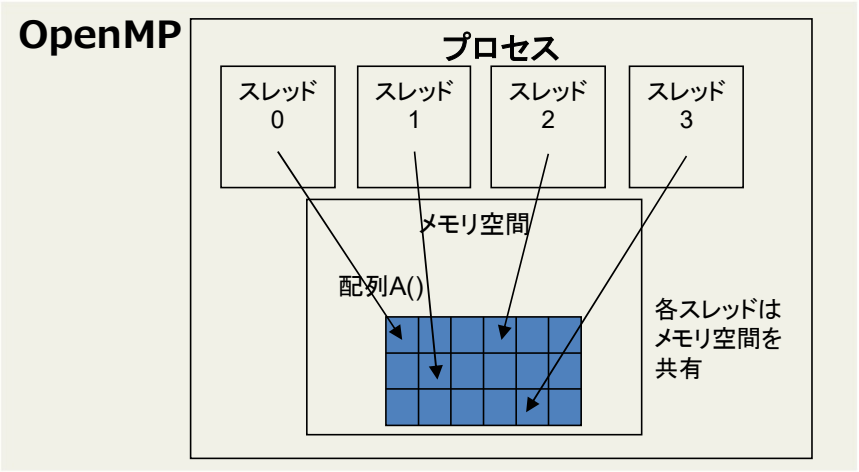
高速化作業の流れ　（MPIによるプロセス並列化）

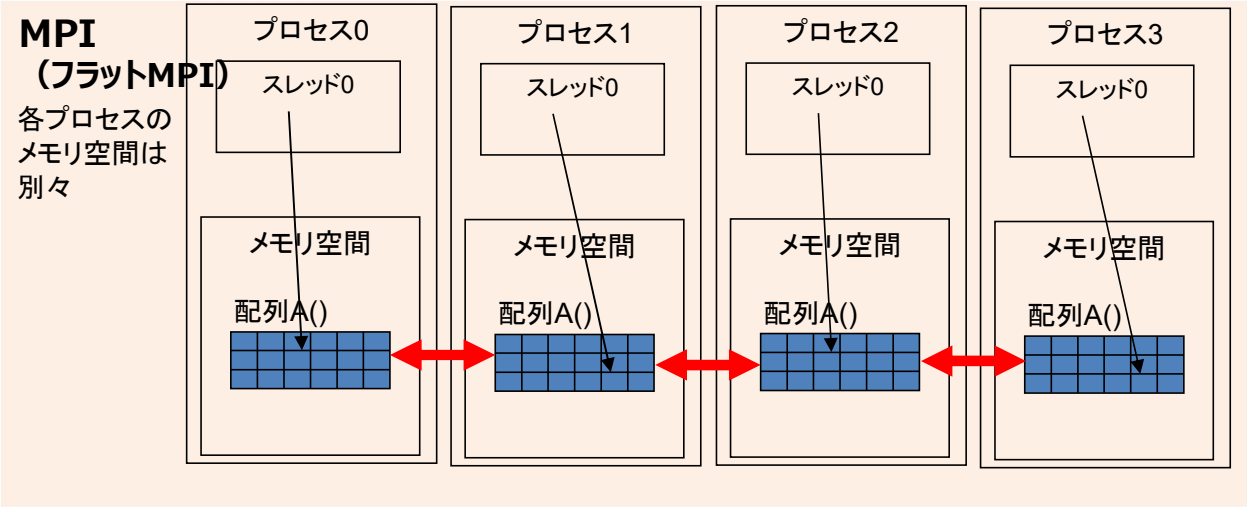
- 全体の計算領域を分割し、各プロセスが自らの担当範囲を計算する。
- 各領域の境界部分で通信が必要となる。




```
#include "mpi.h"

int main(int argc, char **argv)
{

        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD,
&numprocs);
        MPI_Comm_rank(MPI_COMM_WORLD,
&myid);

        if(0 == myid) {
        /*主プロセスのみが処理する部分*/
        }

        /* 通信処理 */
        MPI_Bcast(data, MAXSIZE, MPI_INT, 0,
MPI_COMM_WORLD);

        /* 各プロセスがID
に応じて担当する領域を並列に処理 */
        for(i=myid*(N/4);i<myid*(N/4+1);i++){
            …
        }
        MPI_Finalize();
}
```

スレッド並列（OpenMP）とプロセス並列（MPI）





メリット：プログラムの変更がなくてすむ

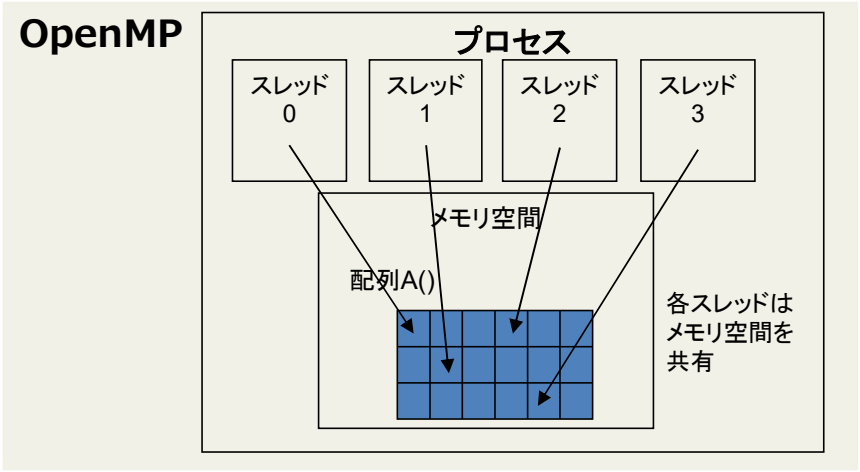
デメリット：**ノードをまたぐことができない！**

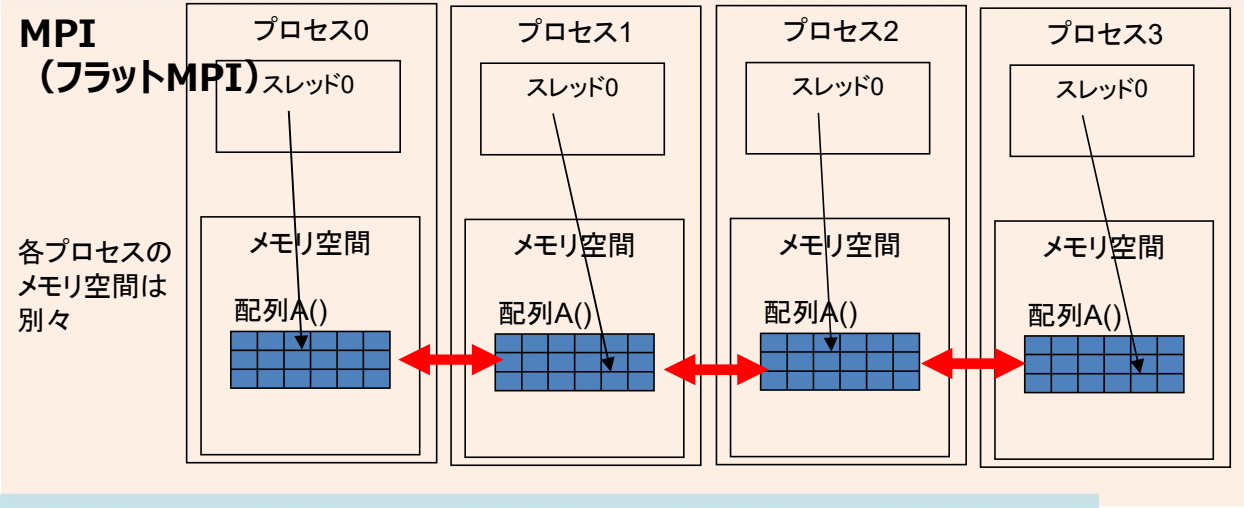
メリット：**ノードをまたぐことができる！**

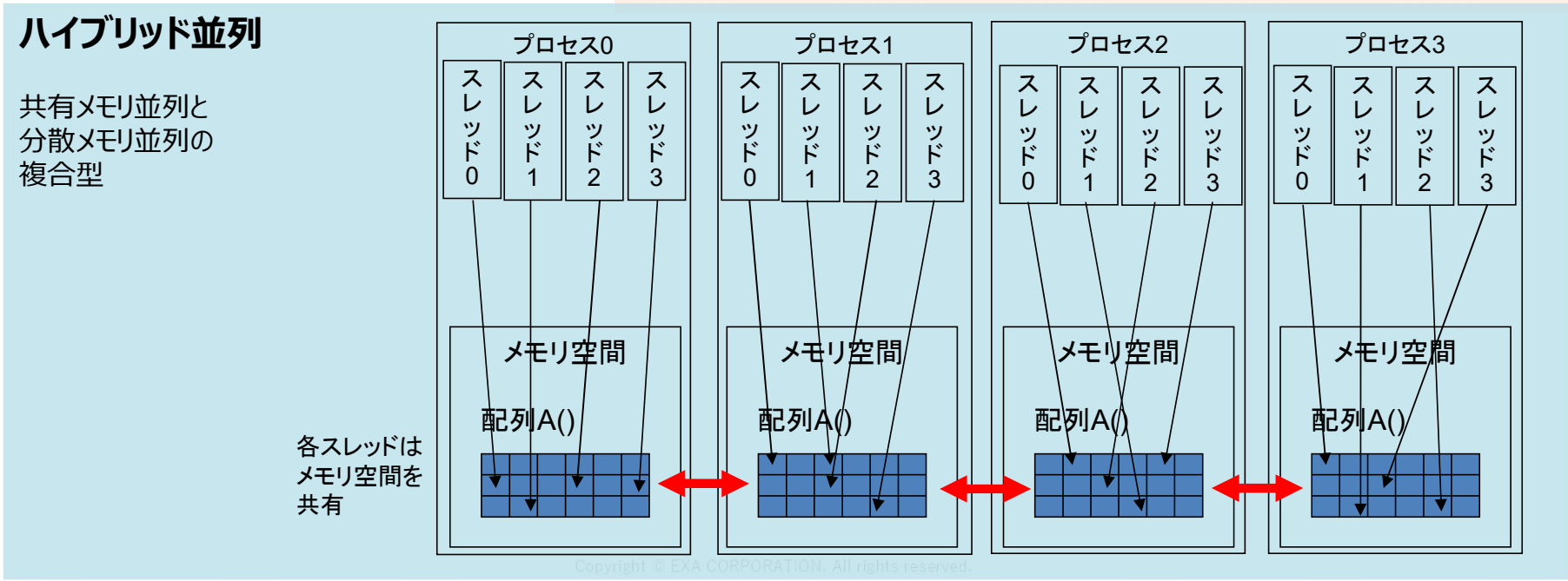
デメリット：明示的に通信のコーディングが必要
通信は（高速とはいえ）NW経由

すなわち、富岳の巨大なノード数を活用するには
MPI並列化が必須！

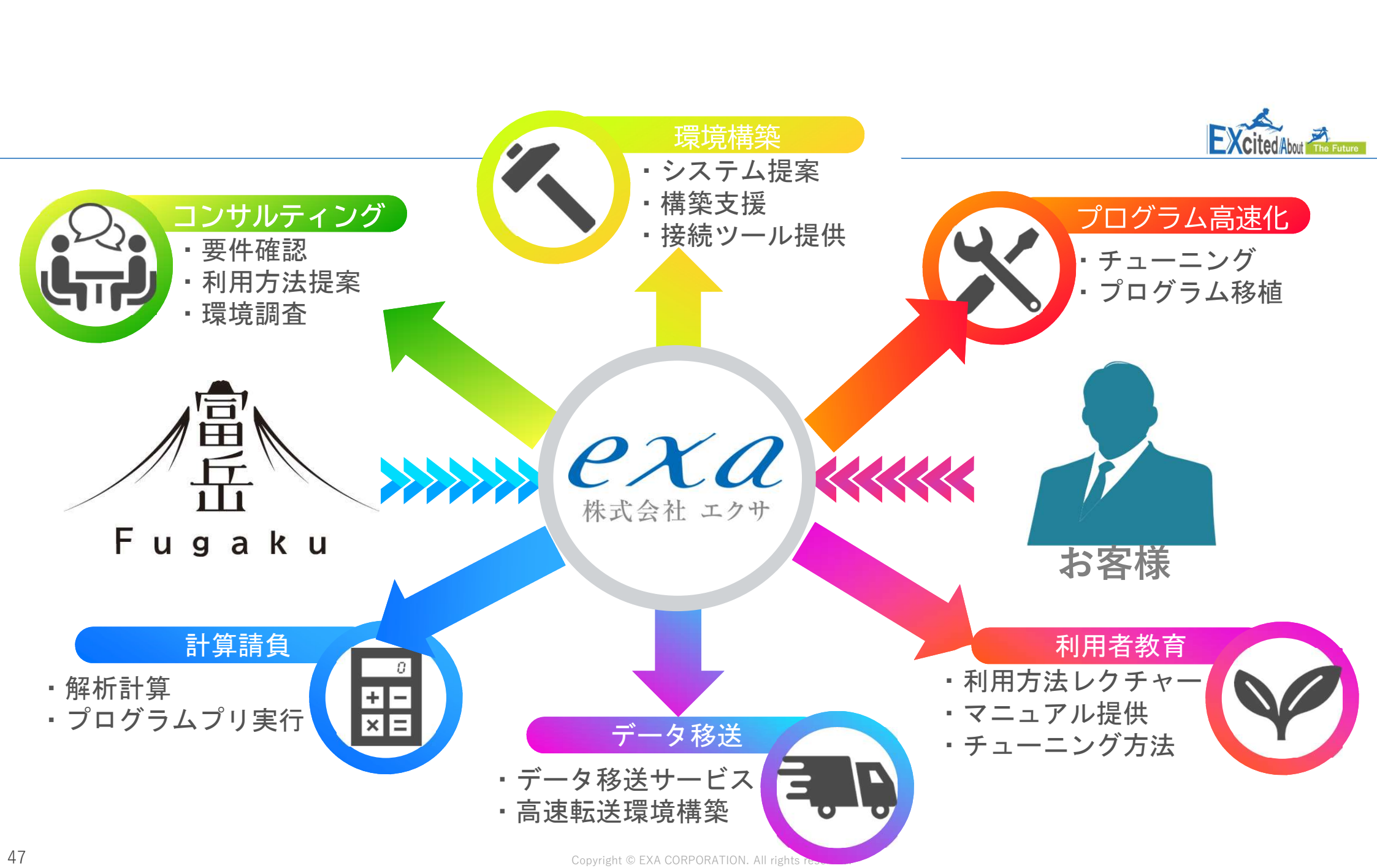
ハイブリッド並列　（OpenMP+MPI）







5.「富岳」ビジネス



ご清聴ありがとうございました

EXA CORPORATION