# OpenFOAMのメニーコア・GPUへの対応に向けた取り組みの紹介

オープンCAEシンポジウム2017
2017年12月9日　名古屋大学

山岸孝輝　井上義昭　青柳哲雄　浅見曉
（高度情報科学技術研究機構）

ver 1.1

# OpenFOAMとGPU・メニーコア

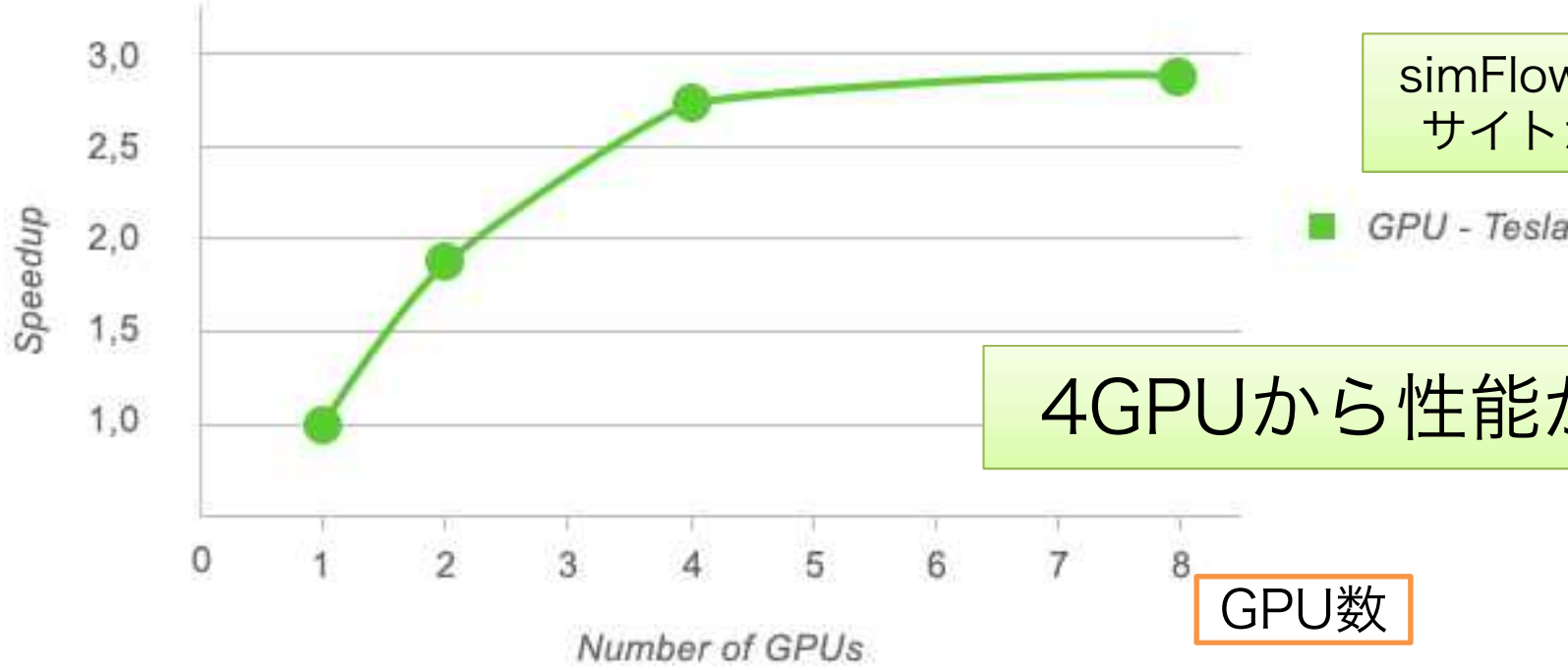
Githubでソースコード公開
https://github.com/Atizar/RapidCFD-dev

- GPU：NVIDIA GPU
  - 実装に手間＋性能出すのは大変
  - simFlow社によるGPU実装版（RapidCFD）
    - リリースは3年前、最後のbug fixは2年前
- メニーコア：Intel Xeon Phi KNL
  - 移植は比較的楽（Intelコンパイラ, MPI）
  - 性能については試行錯誤の段階

# RapidCFD性能評価事例

複数GPUでのスケーリング性能

加速率




simFlow社ウェブサイトから引用

4GPUから性能が鈍化

GPU数

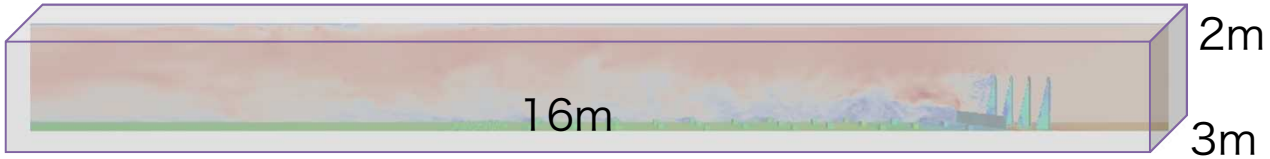
RapidCFD高速化@TSUBAME 3.0について紹介

# TSUBAME 3.0 測定環境

| | 項目 | 内容 |
|---|---|---|
| HW | TSUBAME 3.0 | SGI ICE XA |
| | **CPU** | Xeon E5-2680v4[**426GF**, 2.4GHz, 14core], **2socket** |
| | **GPU** | Tesla P100 with NVLink[**5.3TF**, 16GB], **4board** |
| | ノード間接続 | Intel Omni-Path 100Gb/s x4 |
| SW | C++ | Icc 17.0.4 |
| | MPI | OpenMPI 2.1.1 |
| | CUDA | CUDA 8.0.61 |
| AP | **OpenFOAM** | **2.3.0** |
| | **RapidCFD** | **2.3.1+Thrustライブラリによる GPU実装** |
| | 解析ソルバ | pisoFoam |
| 解析データ | （1）弱スケーリング | 192x128x128 or 128x128x128 cells/processor～ |

# 解析モデル（弱スケーリング）

風洞測定部




1プロセッサあたり200 or 300万格子を設定

| GPU or CPU | 格子数 |
|---|---|
| 1 | 128x128x128　200万 |
| 2 | 256x128x128　400万 |
| 4 | 512x128x128　800万 |
| 8 | 256x256x256　1600万 |
| 16 | 512x256x256　3200万 |

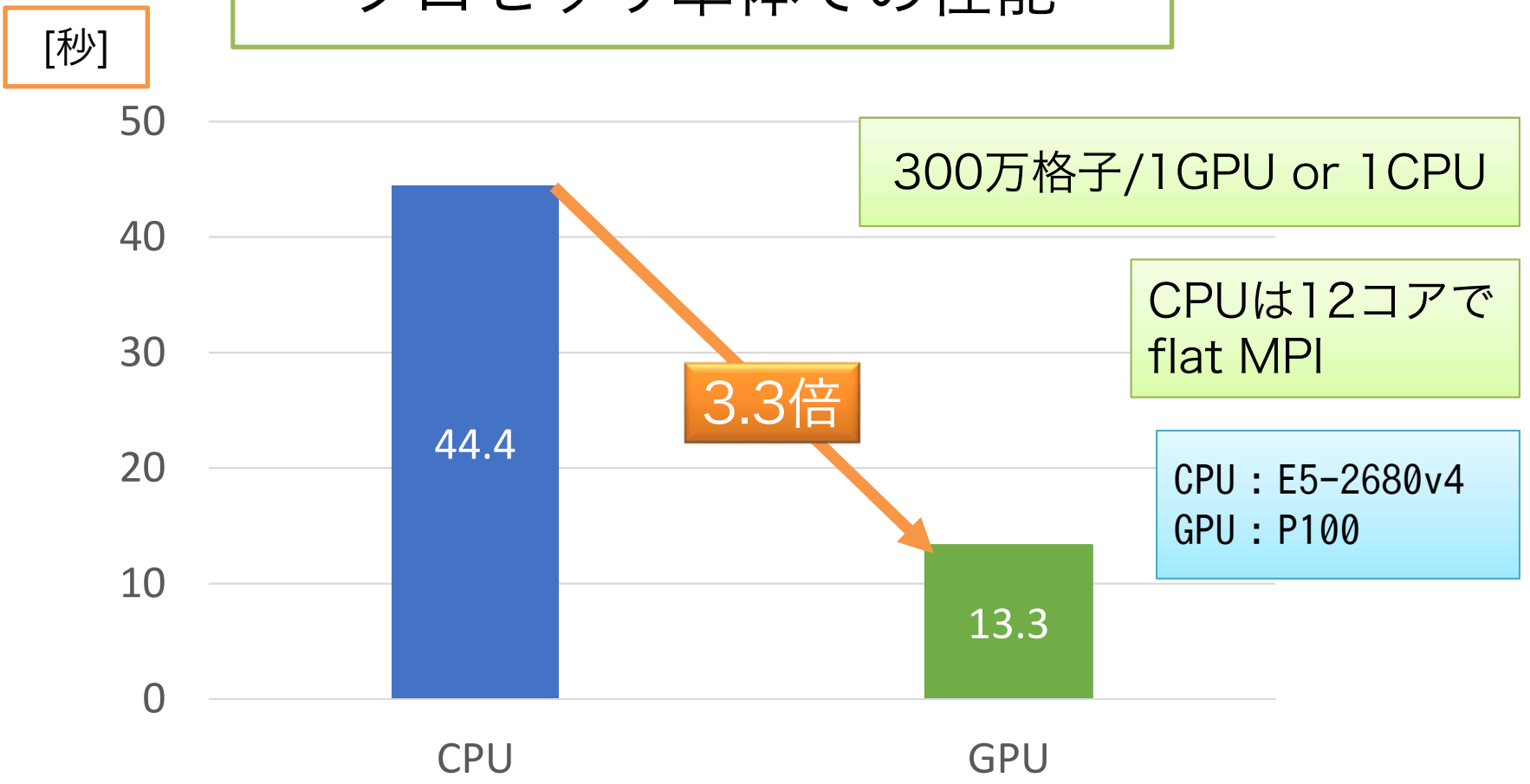
200万格子での設定

200万格子/GPU or CPU で一定（弱スケーリング）

理想的には性能（経過時間）はスケール（一定）

# CPU vs GPU（ASIS）

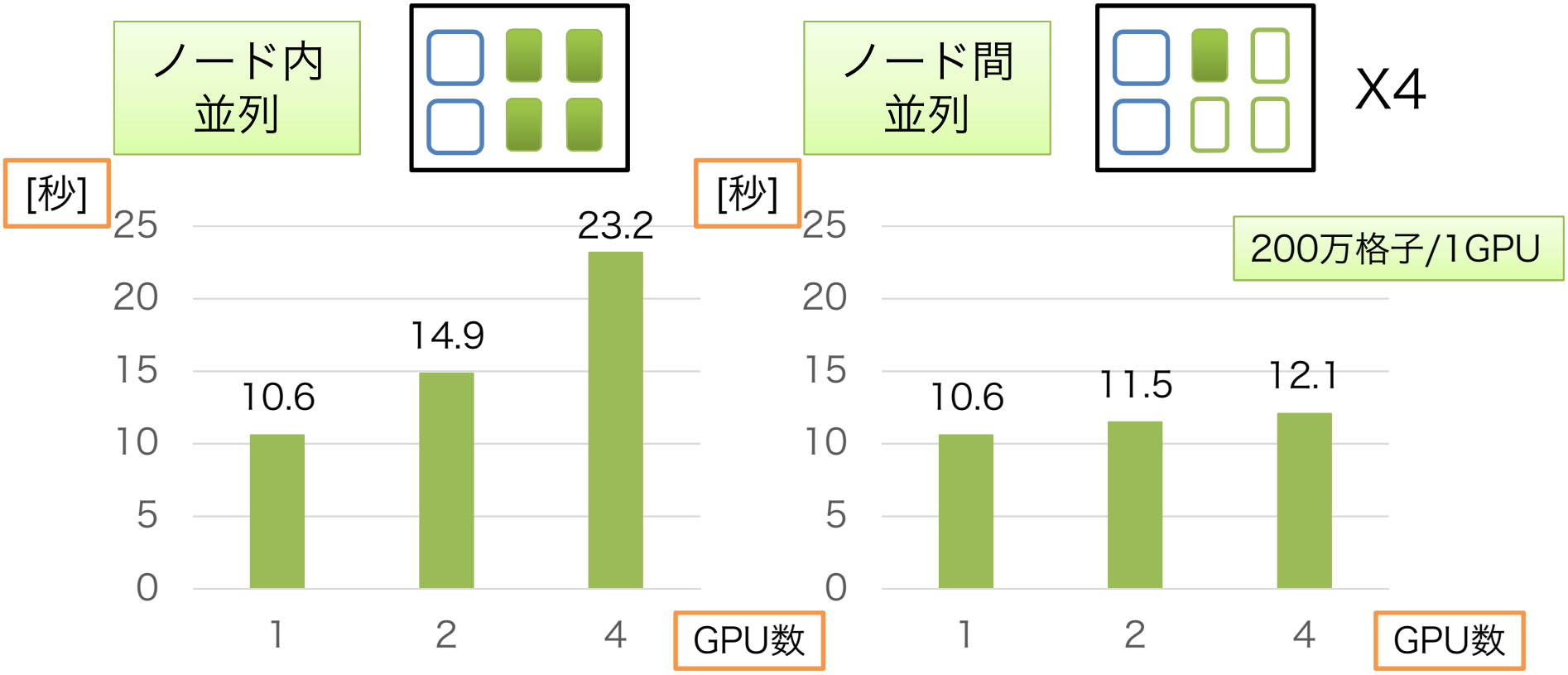
プロセッサ単体での性能




同世代のCPUに比べて3.3倍高速

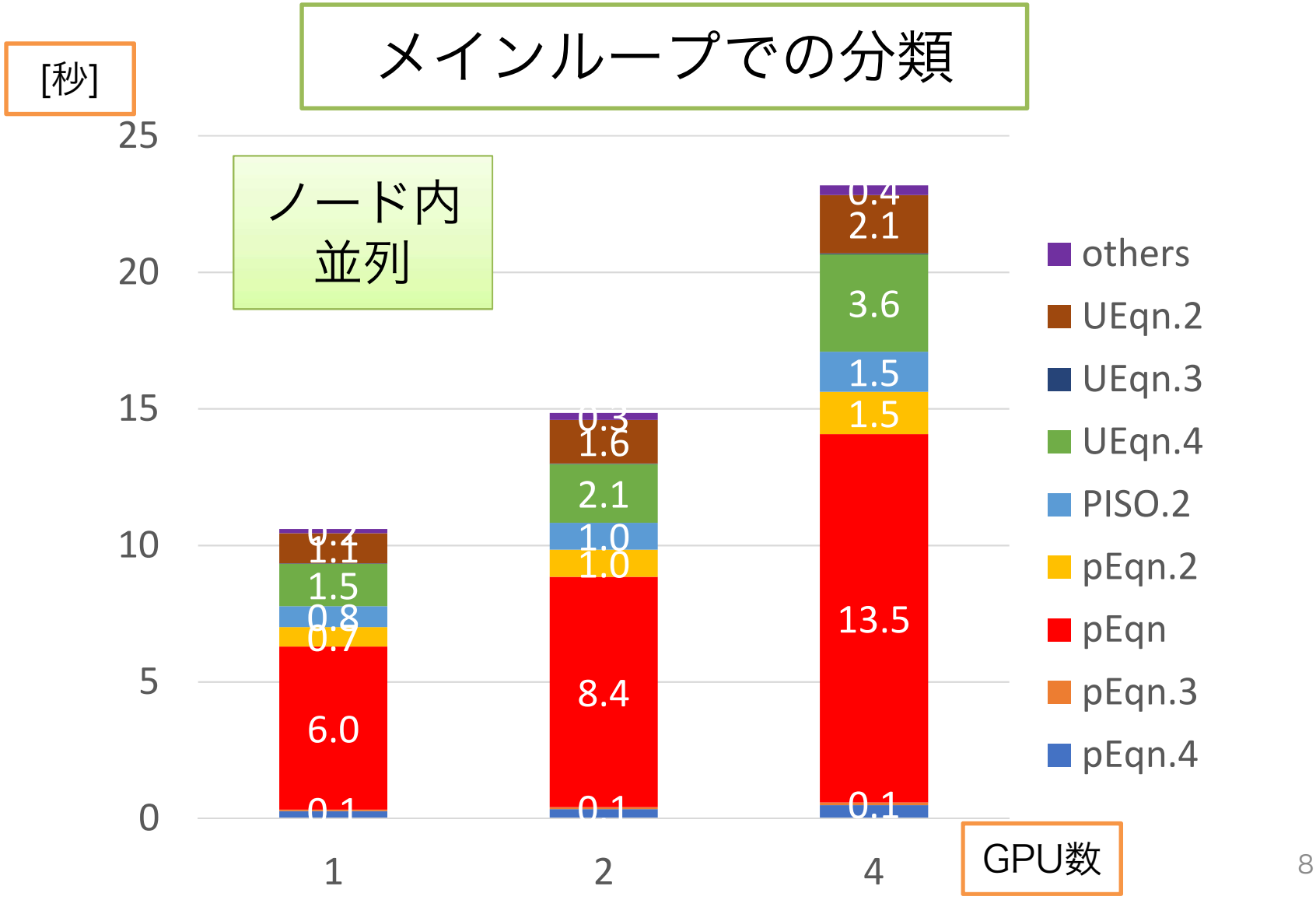
# GPU 弱スケーリング性能（ASIS）

2種類のGPU配置方法で計測（1→4GPU）




GPUのノード内並列で性能劣化　4GPUで2倍低速

# 処理プロセスごとに分類して計測



# ソースコード解析 性能阻害箇所

```
template<>
scalar sumProd(const gpuList<scalar>& f1, const
gpuList<scalar>& f2)
{
    if (f1.size() && (f1.size() == f2.size()))
    {
        thrust::device_vector<scalar> t(f1.size());

        thrust::transform
        (
            f1.begin(),
            f1.end(),
            f2.begin(),
            t.begin(),
            multiplyOperatorFunctor<scalar,scalar,scalar>()
        );

        return thrust::reduce(t.begin(),t.end());
}
    else
    {
        return 0.0;
    }
}
```

総和計算処理にて
性能が悪化と判明

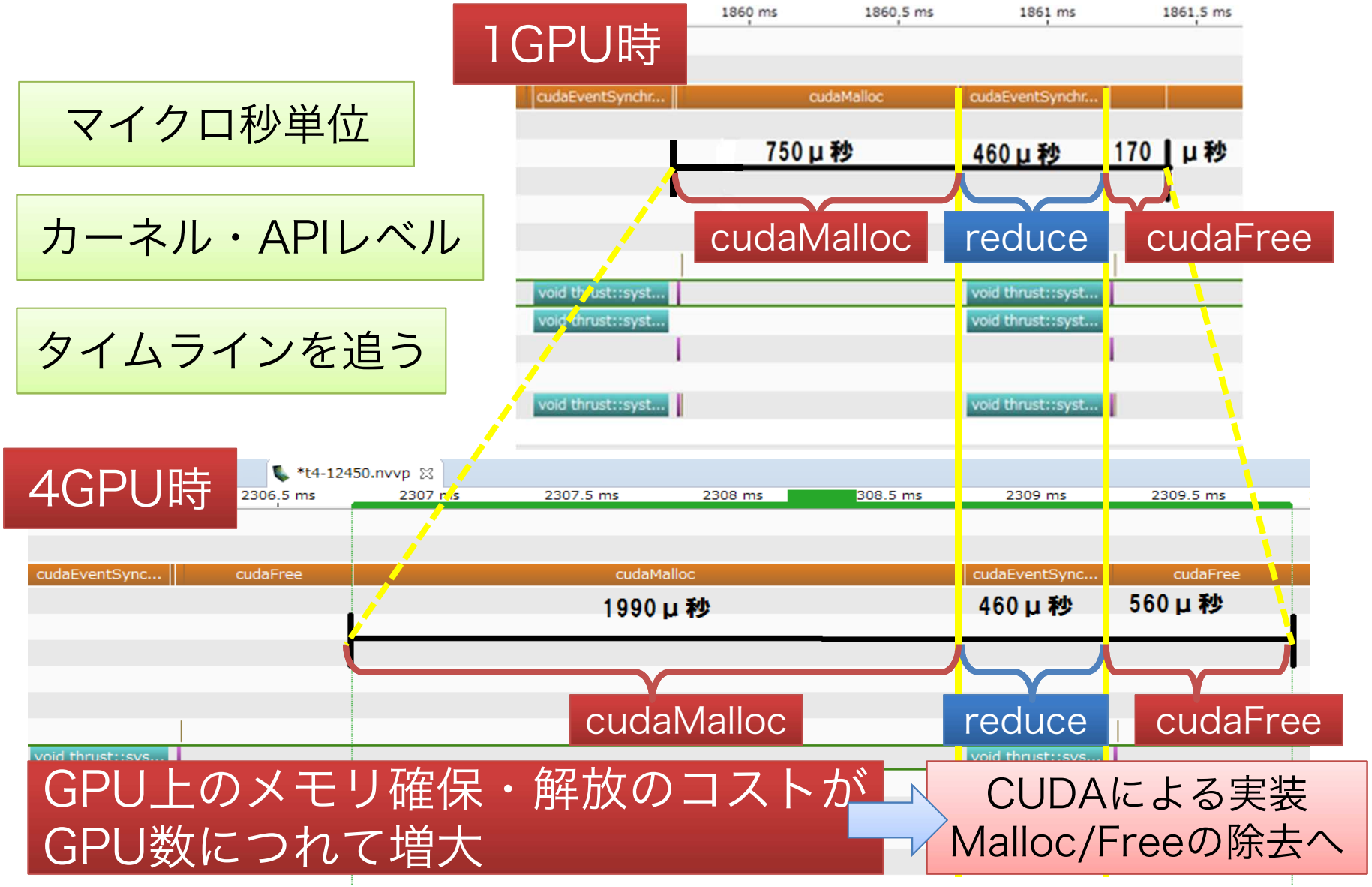
総和計算（thrustライブラリ実装）

# Visual Profilerによる詳細分析

マイクロ秒単位

カーネル・APIレベル

タイムラインを追う

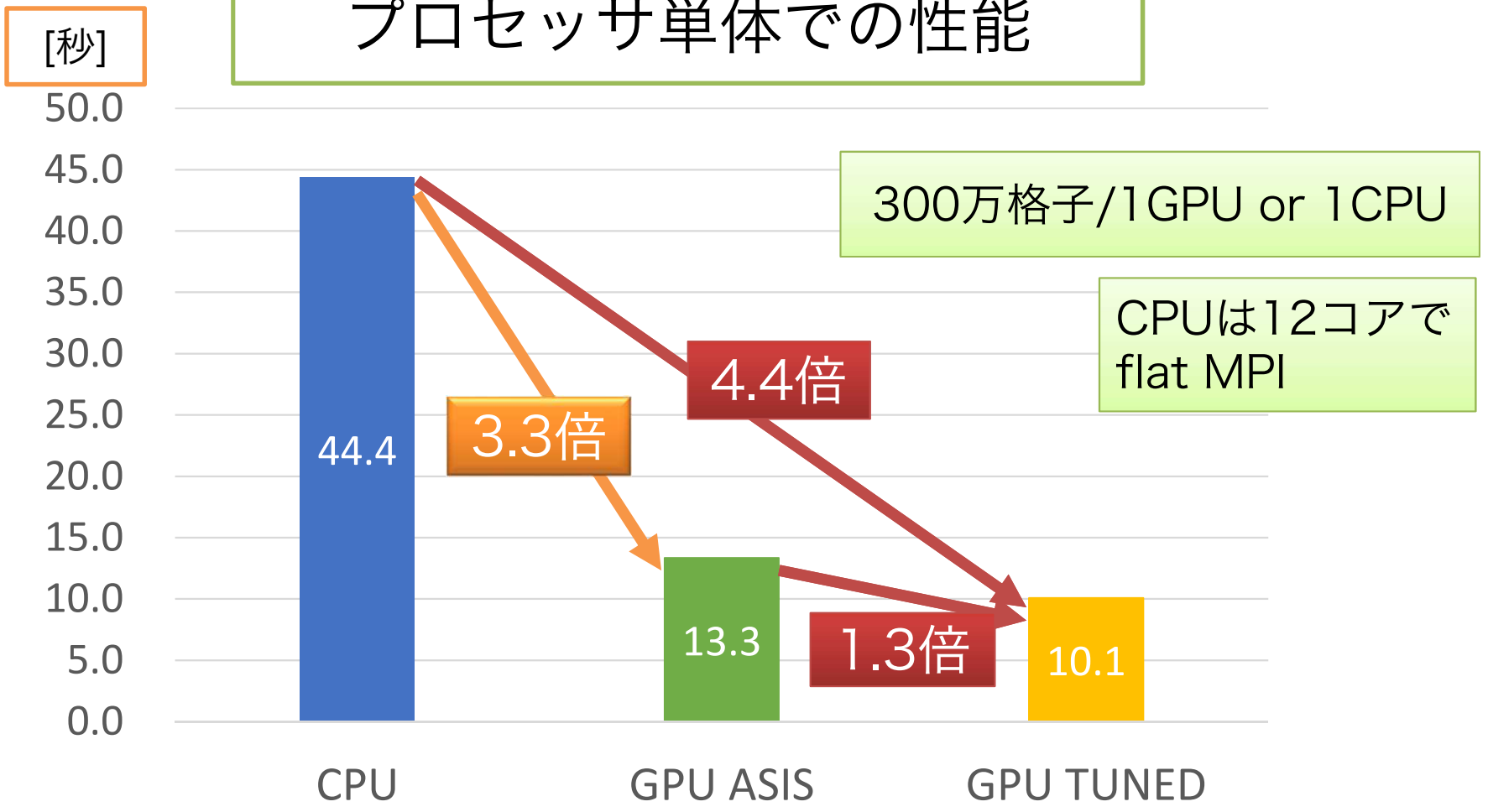



GPU上のメモリ確保・解放のコストが
GPU数につれて増大

CUDAによる実装
Malloc/Freeの除去へ

# CPU vs GPU（TUNED）

プロセッサ単体での性能

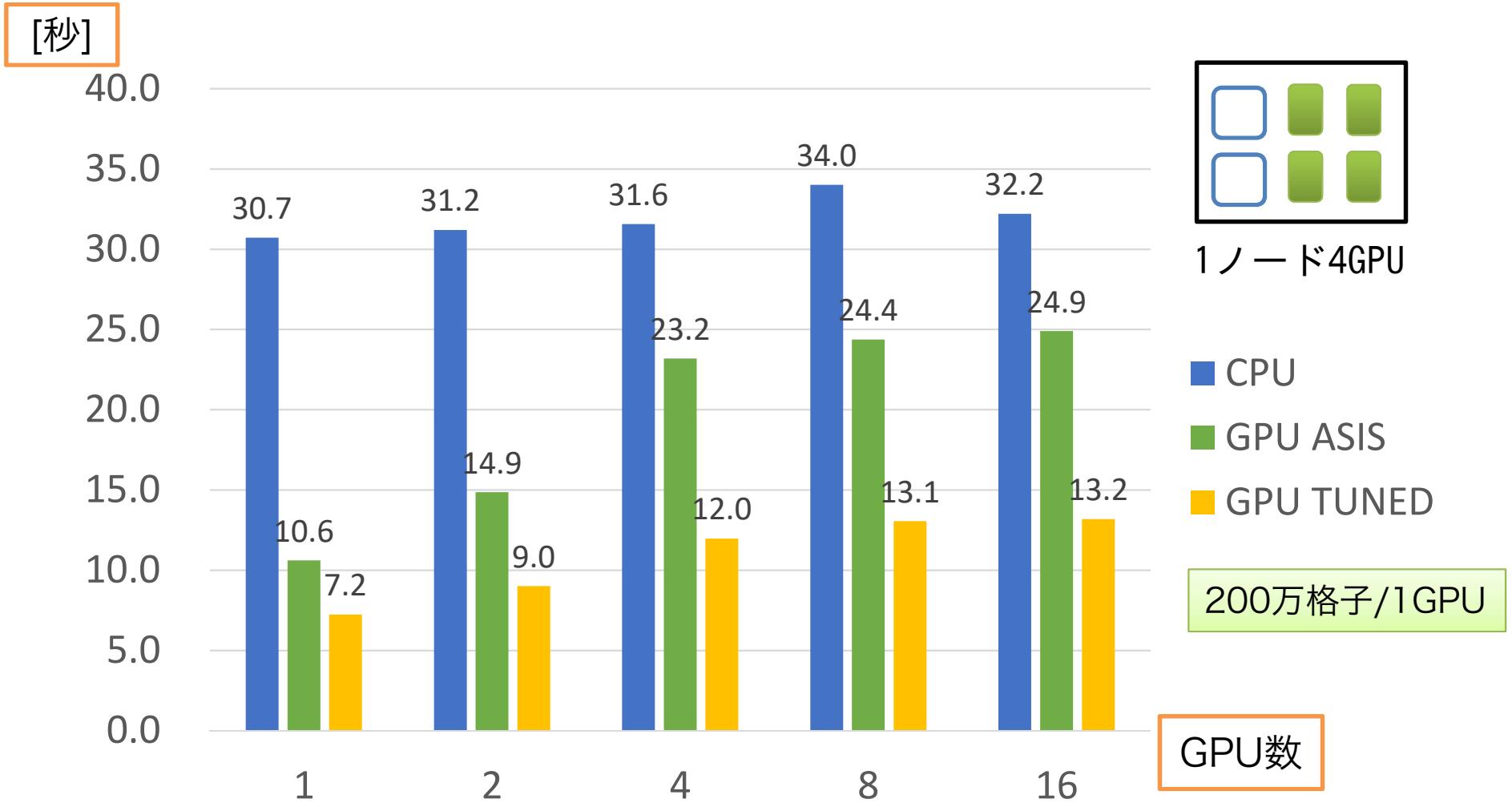



300万格子/1GPU or 1CPU

CPUは12コアで
flat MPI

3.3倍 → 4.4倍 高速に

# GPU 弱スケーリング性能（TUNED）




多数GPU設定（4, 8, 16）にて2倍改善

# RapidCFD 改善の余地について

- OpenFOAMのversionが古い
  - 最新の5.0よりも約3年前のもの
- Thrustライブラリでの GPU実装
  - 柔軟な最適化が難しい
  - コンピュートケイパビリティが3.0に設定
  - プロファイラ結果の解析が困難

CUDA

```
Time(%)     Time      Calls        Avg        Min        Max  Name
0.47%  50.343ms      21318  2.3610us  1.6960us  49.216us  Foam::reduce_C_5(double*,double*,int)
```

Thrust

```
Time(%)     Time      Calls        Avg        Min        Max  Name
 0.47%  50.761ms        523  97.057us  5.3120us  1.0682ms  void thrust::system::cuda::detail::bulk_::detail::launch_by_value<unsigned
int=0,
thrust::system::cuda::detail::bulk_::detail::cuda_task<thrust::system::cuda::detail::bulk_::parallel_group<thrust::system::cuda::detail::bulk_::concurrent_group<thrust::system::cuda::detail::
bulk_::agent<unsigned long=1>, unsigned long=0>, unsigned long=0>, thrust::system::cuda::detail::bulk_::detail::closure<thrust::system::cuda::detail::for_each_n_detail::for_each_kernel,
thrust::tuple<thrust::system::cuda::detail::bulk_::detail::cursor<unsigned int=0>, thrust::zip_iterator<thrust::tuple<thrust::detail::normal_iterator<thrust::device_ptr<Foam::Vector<double>
const >>, thrust::detail::normal_iterator<thrust::device_ptr<Foam::Vector<double> const >>, thrust::detail::normal_iterator<thrust::device_ptr<double>>, thrust::null_type, thrust::null_type,
thrust::null_type, thrust::null_type, thrust::null_type, thrust::null_type, thrust::null_type>>,
thrust::detail::wrapped_function<thrust::detail::binary_transform_functor<Foam::dotOperatorFunctor<Foam::Vector<double>, Foam::Vector<double>, double>>, void>, unsigned int,
thrust::null_type, thrust::null_type, thrust::null_type, thrust::null_type, thrust::null_type, thrust::null_type>>>>(unsigned long=1)
```

# 新しいGPU版OpenFOAM開発へ

- 最新のOpenFOAMをベースにして
- 開発の方針、特徴
  - GPU実装はすべてCUDA
  - GPUDirect RDMA転送に対応
- 目標
  - 高速な大規模実行の実現
  - GPUを用いた大規模アプリに適用しうる知見の蓄積

# メニーコア：Intel Xeon Phi KNL

OpenFOAMの基礎調査として
RIST所有のKNL搭載PCクラスタ
（4ノード）で性能評価

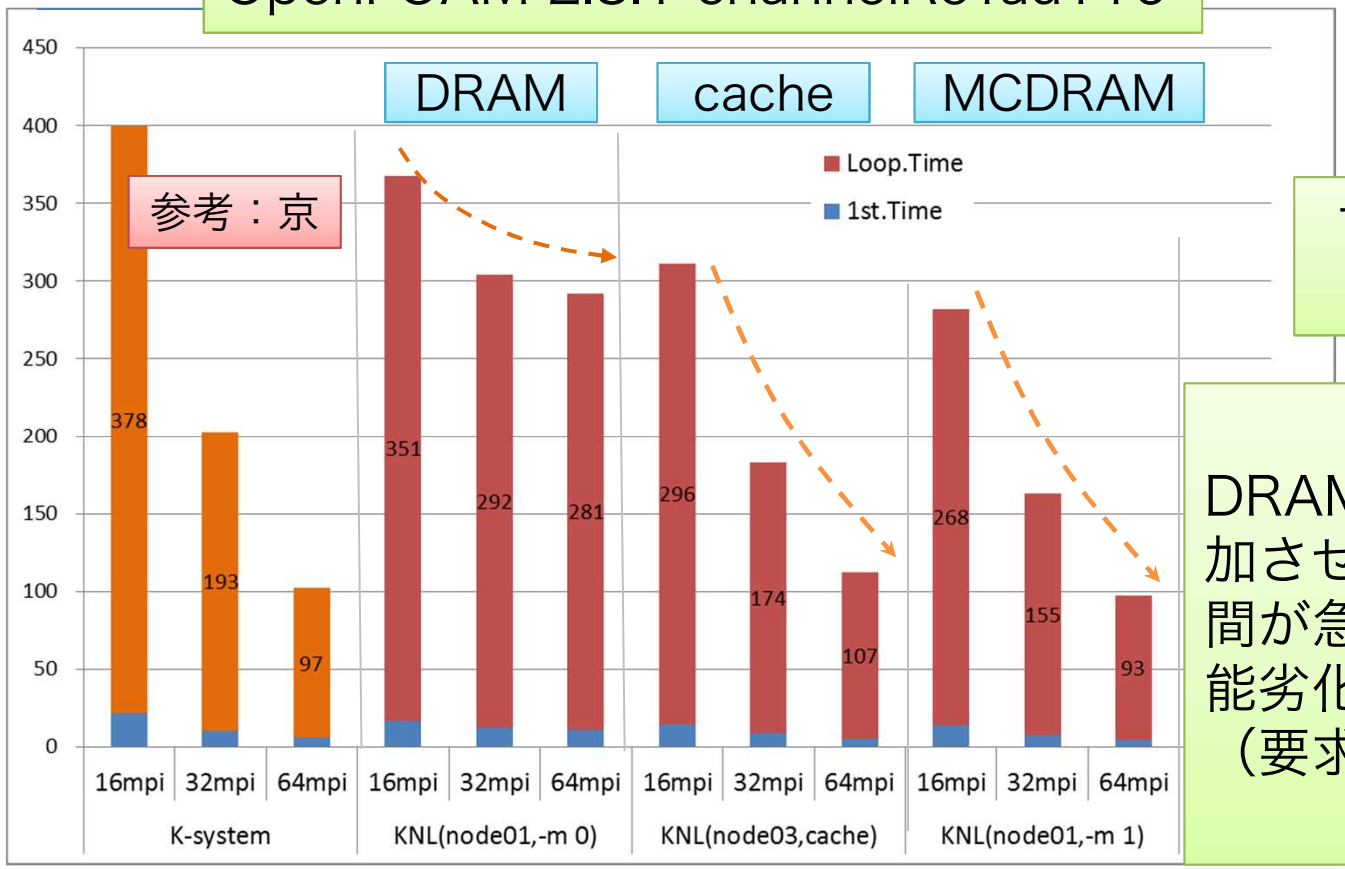
| | 項目 | 内容 |
|---|---|---|
| HW | KNL搭載PCクラスタ | CX6000M1/CX1640M1x4 |
| | **ノード数** | **4** |
| | モード設定 | flat/cache/MCDRAM + Quadrant/AlltoAll/SNC2/SNC4 |
| | **CPU** | Intel **Xeon Phi 7250**[**3TFLOPS**, 1.4GHz, **68core**] |
| | | Intel Omni-Path |
| SW | C++/Fortran | Icc 17.0.1 |
| | MPI | Intel MPI 2017 update 1 |
| | コンパイルオプション | -O3 -xMIC-AVX512 |
| AP | **OpenFOAM** | **2.3.1（RIST最適化版）** |
| | 解析ソルバ | pimpleFoam |
| 解析データ | **強スケーリング** | **240x130x96 cells/1-64processros** |

# メモリ モード変更

https://github.com/opencae/OpenFOAM-BenchmarkTest

OpenFOAM 2.3.1 channelReTau110

PCG,前処理DIC



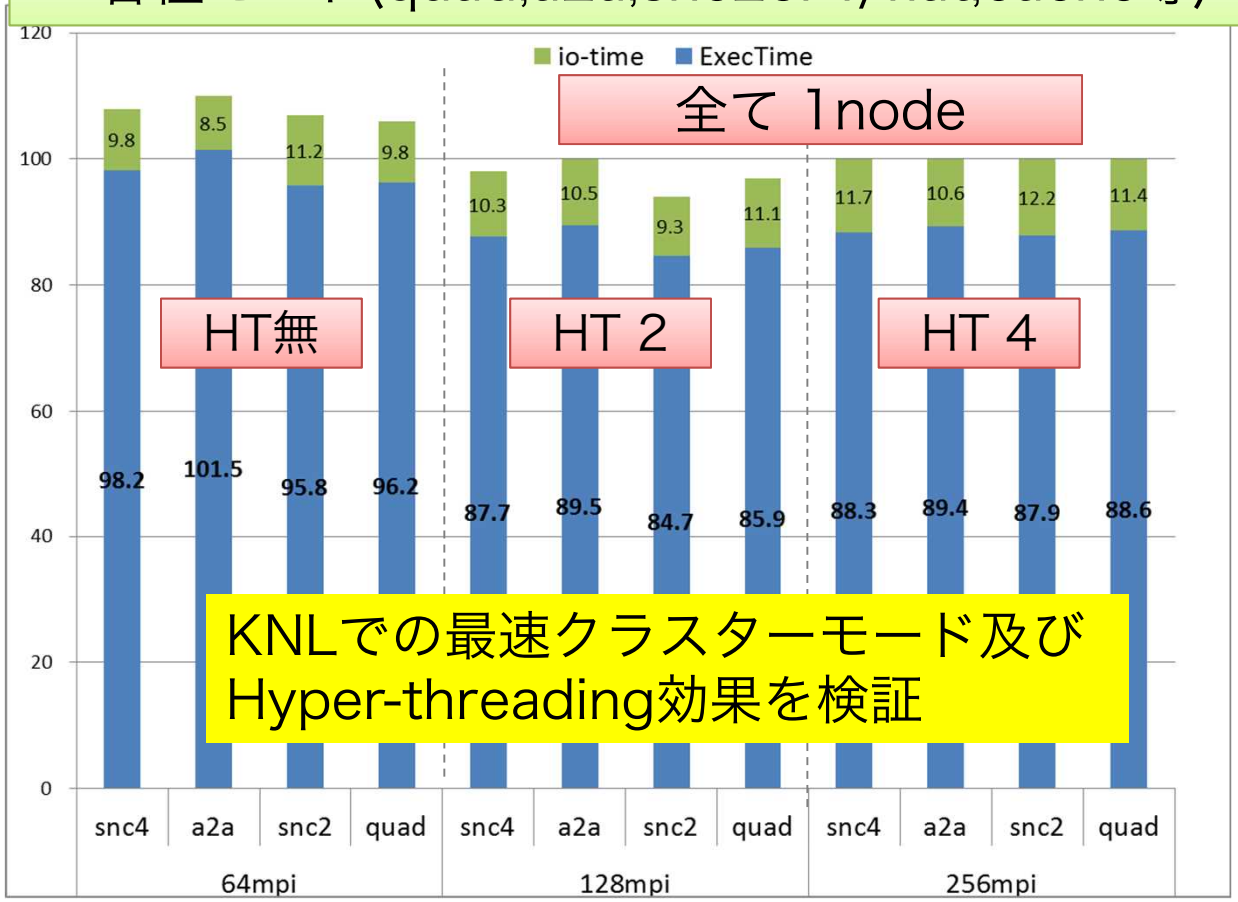

1ノード内並列数の増加
（16/32/64PE）

DRAM設定時、並列数を増加させると演算処理部の時間が急増、スケーリング性能劣化
（要求メモリバンド幅大）

メモリモード変更（DRAM→cache/MCDRAM）で3倍弱高速に

# クラスターモード＋Hyper-threading

各種モード(quad,a2a,snc2or4/flat,cache等)×Hyper-threading



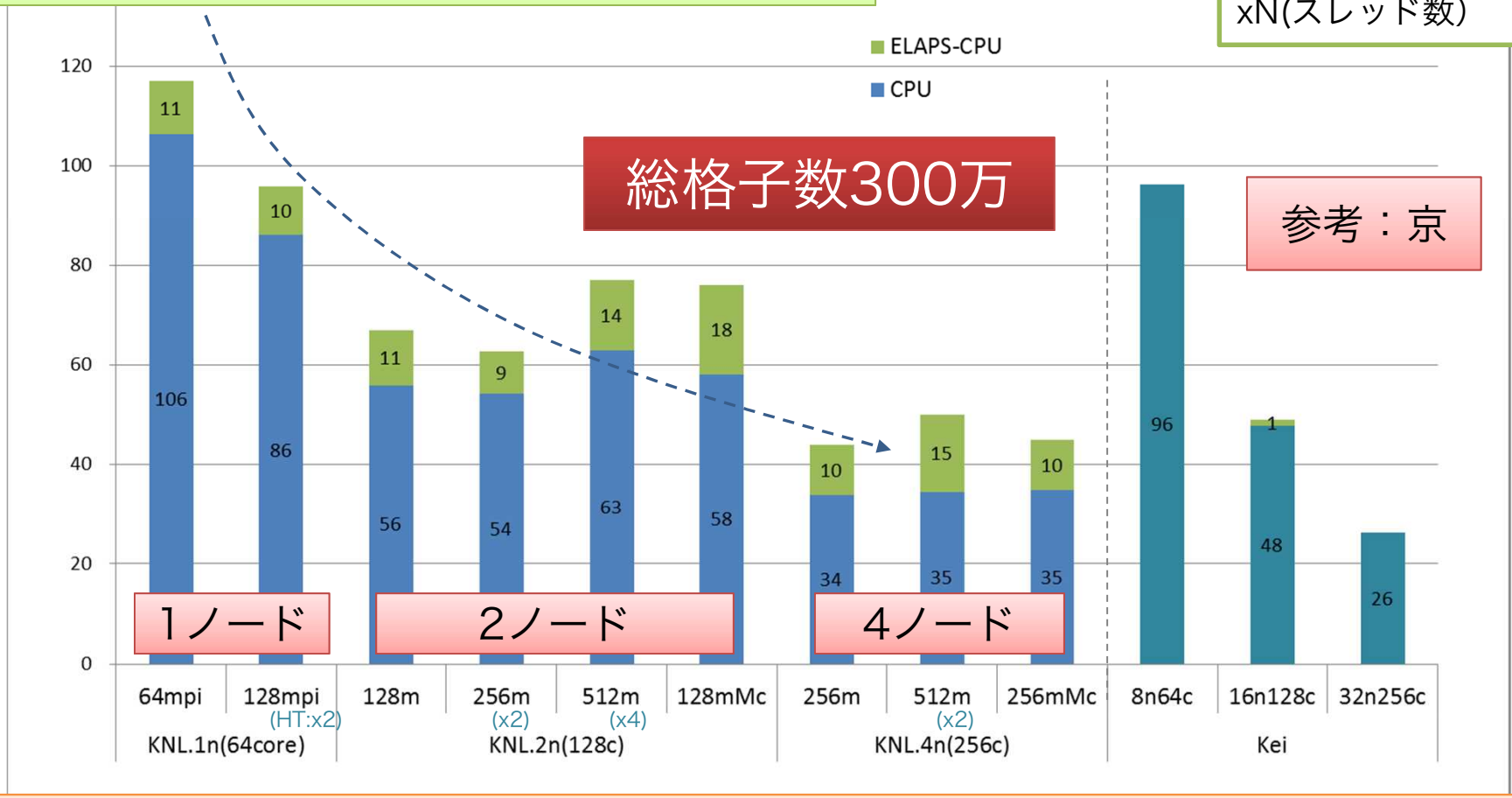

- All-to-all (a2a): Uniform mesh interconnect
- Quadrant (quad): Four virtual address spaces (one NUMA domain)
- Sub-NUMA-2 (snc2): Two distinct NUMA domains
- Sub-NUMA-4 (snc4): Four distinct NUMA domains

最速：snc2/flat+128mpi[64corex2HT]
最大で17%のコスト削減

# マルチノード＋HyperThreading

KNL/4node構成(quad, cacheモード)

Mc: McDram
HT:HyperThread
xN(スレッド数)




1-4ノードまで クラスターモード、HTに依存せずスケール

# まとめ

- RapidCFDの性能評価と最適化
  - 総和計算をCUDA実装→無駄なメモリ確保・解放コストを削減
  - GPU単体では3.3→4.4倍高速に（CPU比較）
  - 複数GPUでのスケーリング性能は最大で2倍改善
  - 新しいGPU版OpenFOAM開発へ
- KNL4ノードでのOpenFOAMの性能評価
  - メモリモード変更（DRAM→cache/MCDRAM）で3倍弱高速に
  - クラスターモード・HTの設定は最大で17%影響
  - 4ノードまではクラスターモード・HTの設定に依存しないでスケーリング

# 謝辞


- 本発表の内容は，HPCIシステム利用研究課題「流体・粒子の大規模連成解析を用いた竜巻中飛散物による建物被害の検討（課題番号：hp170055），課題代表者：菊池浩利（清水建設（株）技術研究所）」の高度化支援に基づくものです.