第1回並列計算セミナー

2010年10月2日 東京大学

Open∇FOAMの並列計算

今野 雅

(東京大学大学院工学系研究科建築学専攻)

OpenCAE

目次

OpenCAE

OpenFOAM概要

OpenCAE

Open∇FOAM®の歴史

- 1989年ー2000年：研究室のハウスコード
  - 開発元：インペリアルカレッジ・ゴスマン研(Star-CDの開発元)のHenry、Hrvoje、Niklas
- 1999年ー2004年：商用コード化(FOAM: Field Operation And Manipulation)、開発元：∇Nabla
- 2004年12月：オープンソース化(OpenFOAM)、開発元： OpenCFD

OpenCAE

OpenFOAM®の特徴

- 有限体積法の採用
- 3次元非構造格子に対応
- ポリヘドラル(任意多角形)格子に対応
- 領域分割型の並列計算が容易
- 化学反応、燃焼を含めた複雑な流れ場、乱流、熱伝達、固体力学、電磁場解析などを解析可能

OpenCAE

Open∇FOAM®の特徴

- オブジェクト指向型言語C++言語で記述された

  CFD、固体の応力解析等の連続体力学の分野で使用

  可能な汎用の数値計算クラス・ライブラリ
- オブジェクト指向型言語C++の特徴が活かされおり、支配方程式の解法の実装が非常に簡潔

OpenCAE

OpenFOAM®の特徴

圧縮性のNavier-Stokesの式

$$\frac{\partial U}{\partial t} + \nabla \cdot \phi U - \nabla \cdot \nu \nabla U = -\nabla p$$

陽解法で解くコード

OpenCAE

OpenFOAM®の特徴

```
solve(
      fvm::ddt(U)
    + fvm::div(phi,U)
    - fvm::laplacian(nu,U)
==  - fvc::grad(p)
    );
```

$$\frac{\partial U}{\partial t} + \nabla \cdot \phi U - \nabla \cdot \nu \nabla U = -\nabla p$$

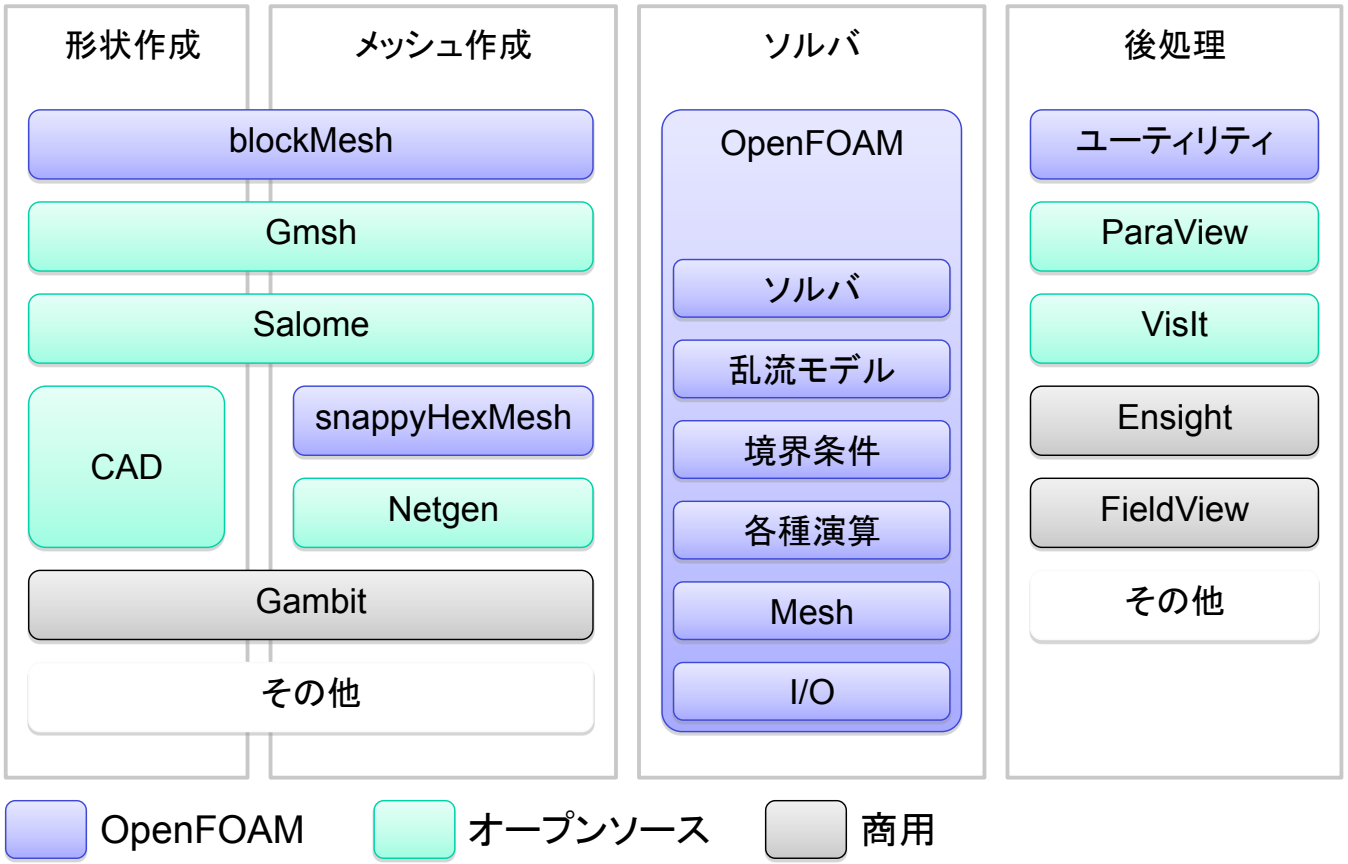
OpenCAE

OpenFOAM®の特徴

- フリーソフトウェア
  - 学生や研究者がCFD解析をトライするきっかけ
  - 超並列計算が低コストで可能
- オープンソース
  - 既存のソースを参照可能
  - 新しい計算モデルや離散スキームの実装が容易
- ライセンス ：GPL(General Public License)

OpenCAE

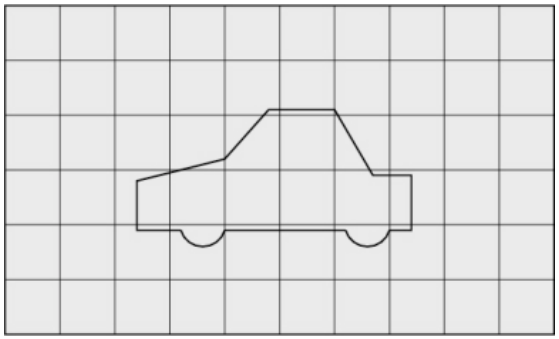
Open∇FOAM®の構成



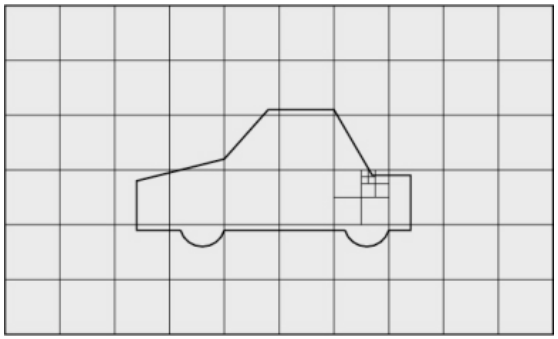
(引用元: 大嶋 拓也 (新潟大学)「イントロダクション: OpenFOAM概要」
第一回OpenFOAM講習会)

OpenCAE

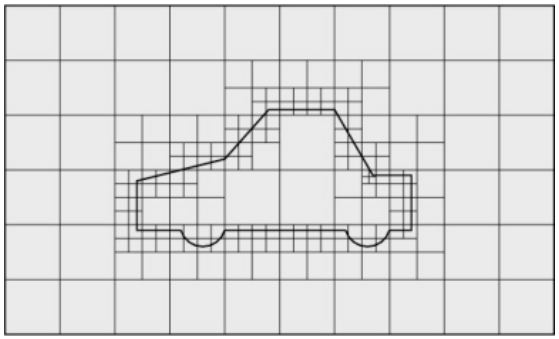
境界適合六面体格子生成ユーティリティ snappyHexMesh



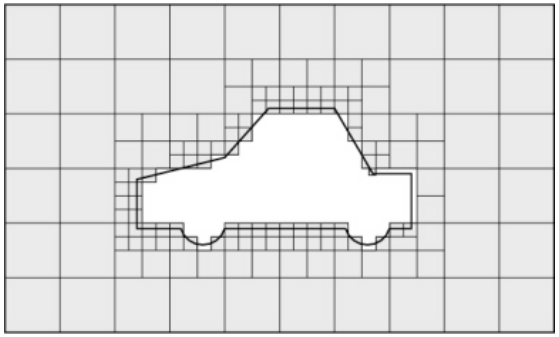
ベース格子



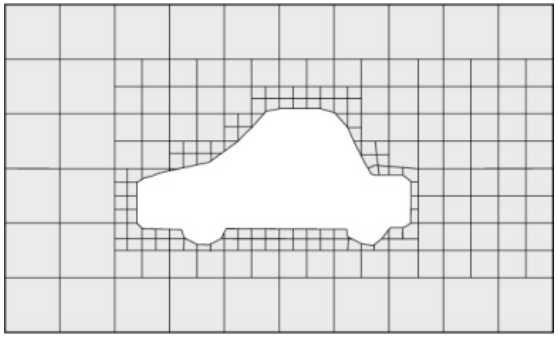
STL表面の細分割



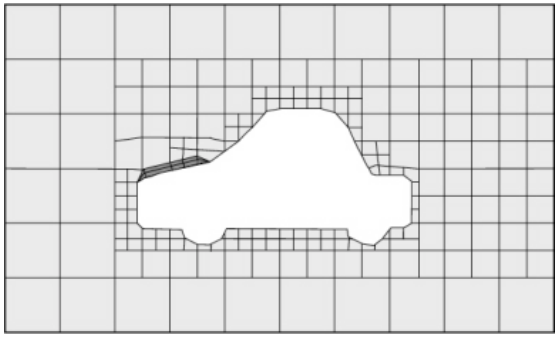
スムージング



階段状格子



境界適合(snap)



レイヤ付加

(図引用元: OpenFOAM User Guide Version 1.5)

OpenCAE

OpenFOAMでの並列計算

OpenCAE

OpenFOAMの並列計算

- プロセス間の通信方法はMPI(Message Passing Interface)通信のみ使用(ピュアMPI、またはフラットMPIという手法)
- 同一ノード内の通信に良く用いられるOpenMPには対応していない (ただし、流体計算の場合ピュアMPIでも十分)
- MPI：Open MPIやMPICHなどに対応

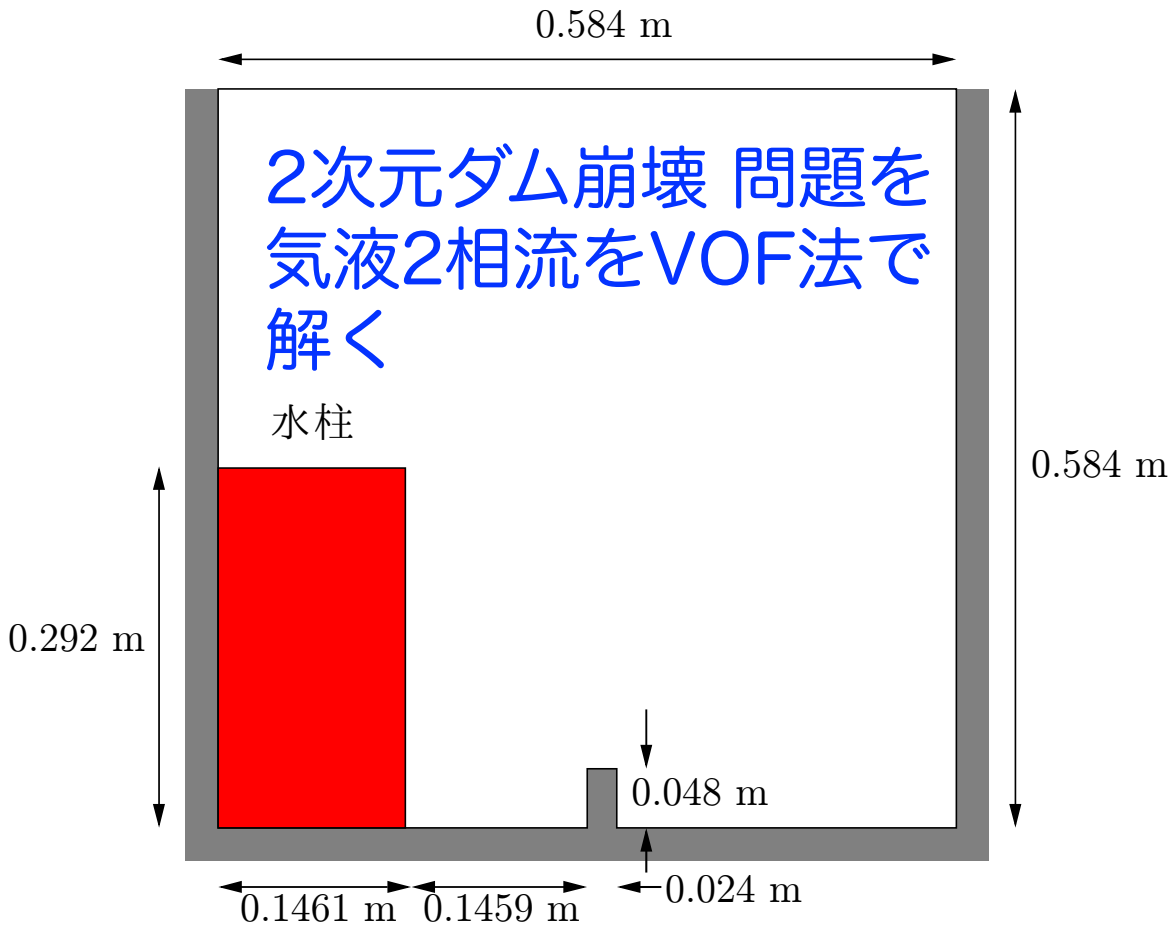
OpenCAE

OpenFOAMでの並列計算方法

- 格子生成 (blockMesh or snappyHexMesh or サードパーティのメッシャー)
- 領域分割 (decomposePar)
- 複数のソルバーをMPI通信で並列解析 (mpirun -np NP SOLVER -parallel)
- 分割結果再構築 (reconstructPar)

OpenCAE

並列計算例



(図引用元: OpenFOAM User Guide Version 1.6和訳版)

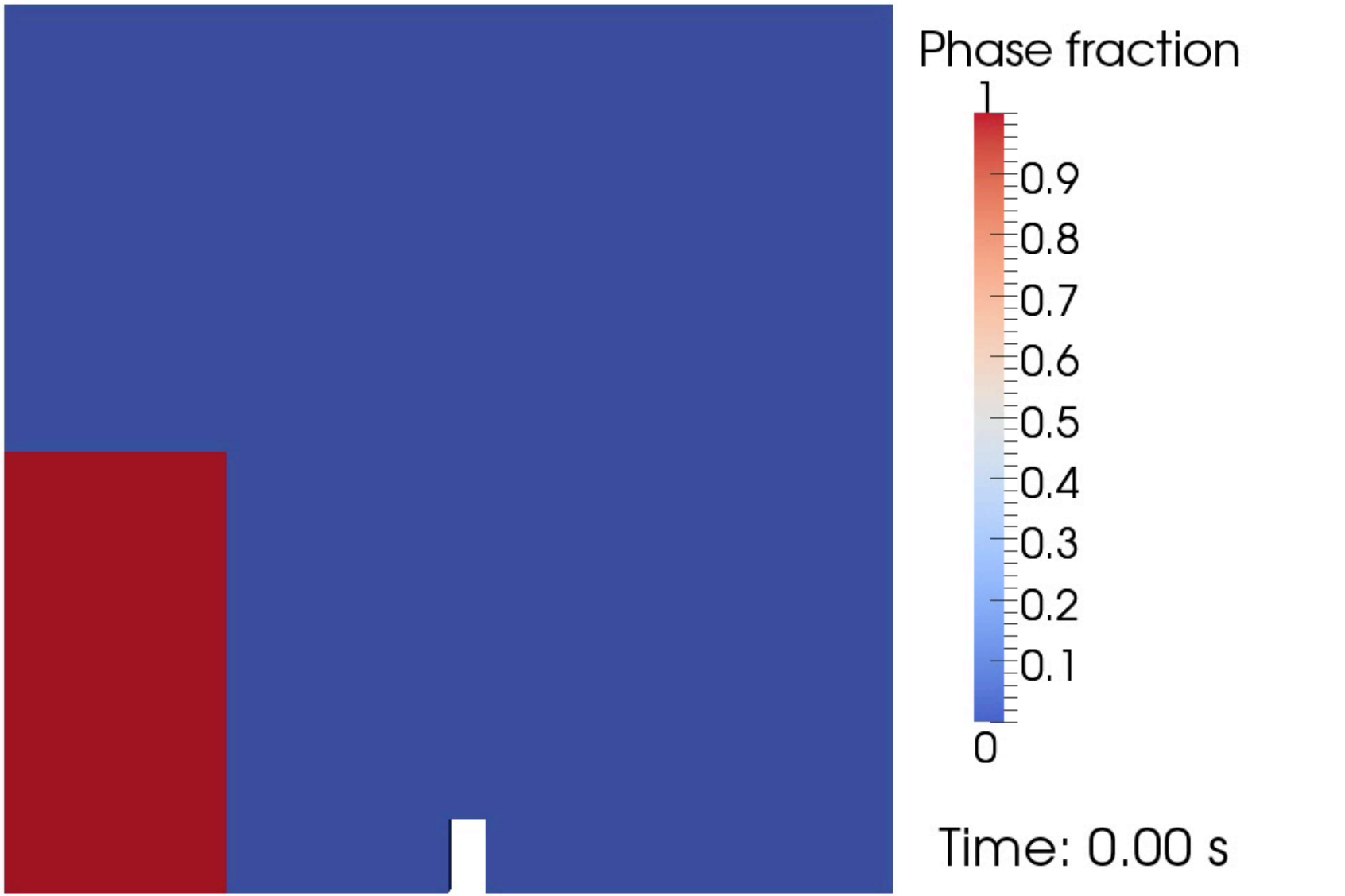
OpenFOAMチュートリアル

tutorials/
multiphase/
interFoam/laminar
のAllrunを実行すると出来る

damBreakFine

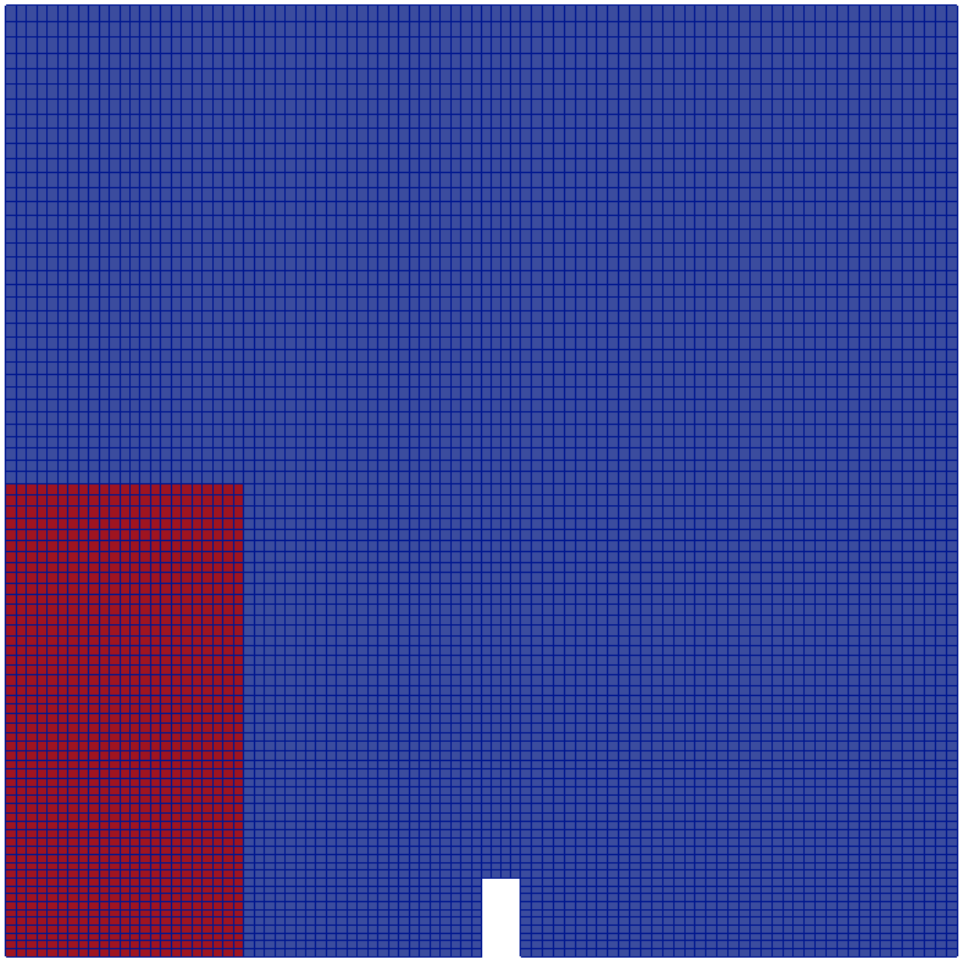
のケース

OpenCAE



OpenCAE

格子生成



格子数: 7700

設定ファイル

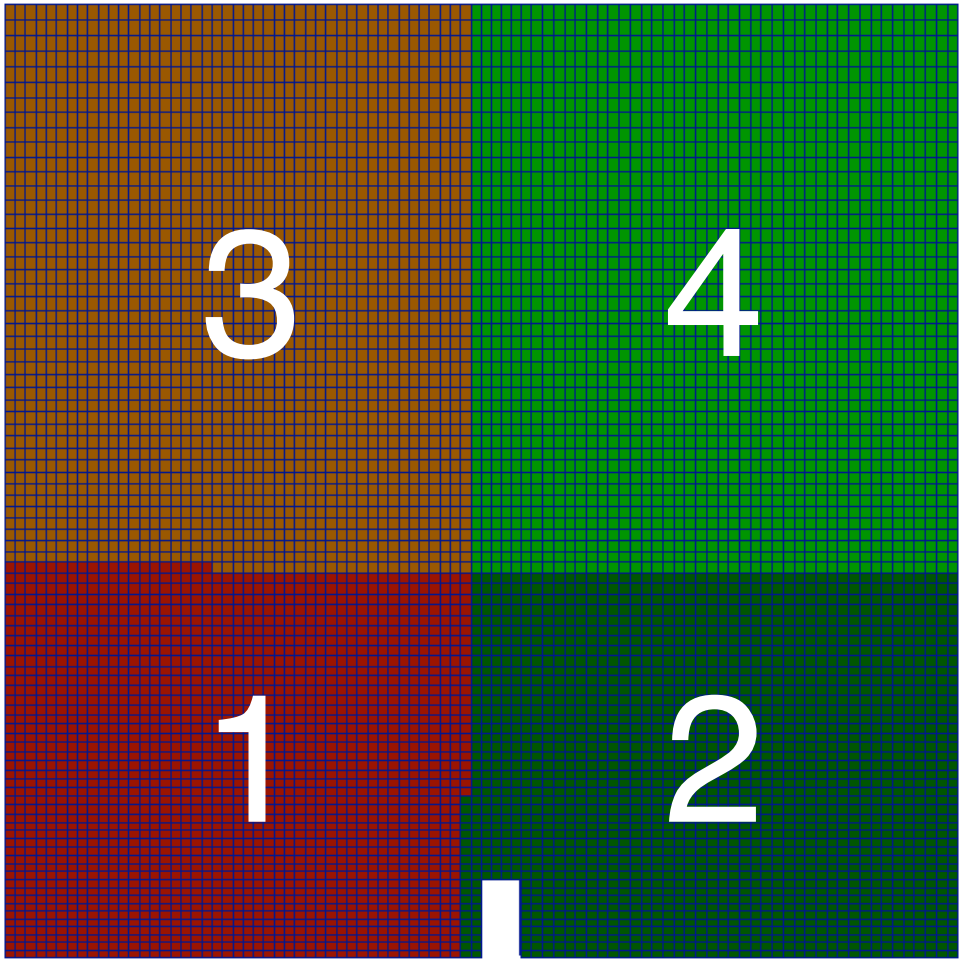
`blockMeshDict`

実行コマンド

```
blockMesh
```

OpenCAE

領域分割



単純にx,y方向にほぼ
等分割して4分割

設定ファイル

`decomposeParDict`

実行コマンド

```
decomposePar
```

OpenCAE

領域分割の主な設定

decomposeParDict

```
numberOfSubdomains 4;

method simple;          ←単純xyz方向分割
method hierarchical;    ←階層的に分割
method metis;           ←Metisライブラリ使用
method scotch;          ←Scotchライブラリ使用
method manual;          ←ファイルで明示的に指定
```

OpenCAE

領域分割方法の特徴

- simple, hierarchical: 構造格子的な格子には有用
- metis (Metisライブラリを使用)
  - プロセッサ間の通信量に大きく影響する分割領域間の界面数を最小化しようとする。 また、複雑な格子でも重み係数に応じて格子数を自動的に分配する
  - ライセンスにより商用利用や再配布が自由ではない
- scotch (Scotchライブラリを使用)
  - フリーソフトライセンスでMetisと互換APIを持つ
  - Metisとほぼ同等の機能

OpenCAE

領域分割の主な設定

decomposeParDict

```
simpleCoeffs
{
    n       ( 2 2 1 );  ←xyz方向の分割数
}
hierarchicalCoeffs
{
    n       ( 2 2 1 );
    order xyz;          ←分割方向の順番
}
```

OpenCAE

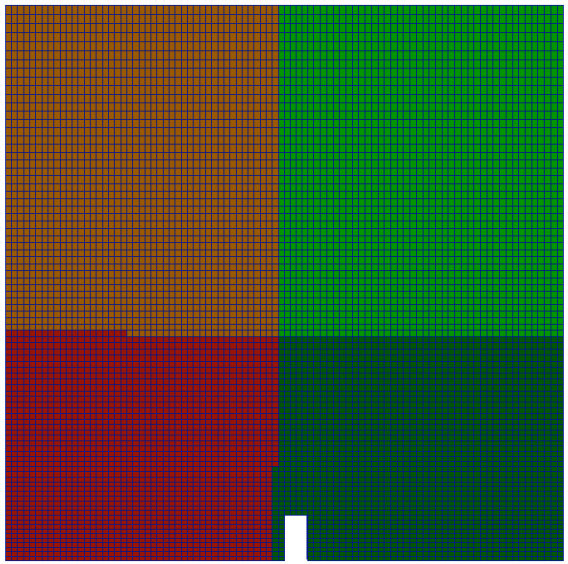
領域分割の主な設定

decomposeParDict

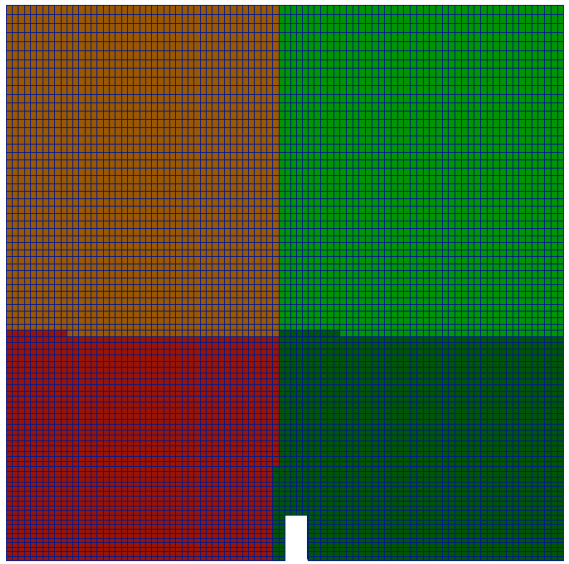
```
metisCoeffs
{                    ↓プロセッサ毎の格子数の重み係数
  processorWeights ( 1 1 1 1 );
}

scotchCoeffs
{
  processorWeights ( 1 2 3 4 );
}
```

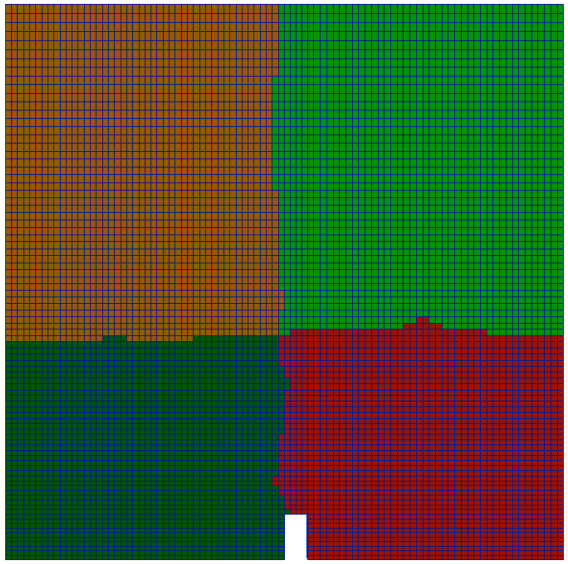
OpenCAE



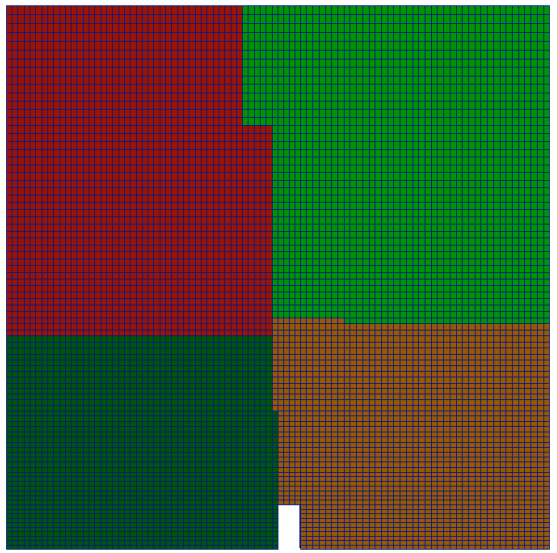
simple



hierar
chical



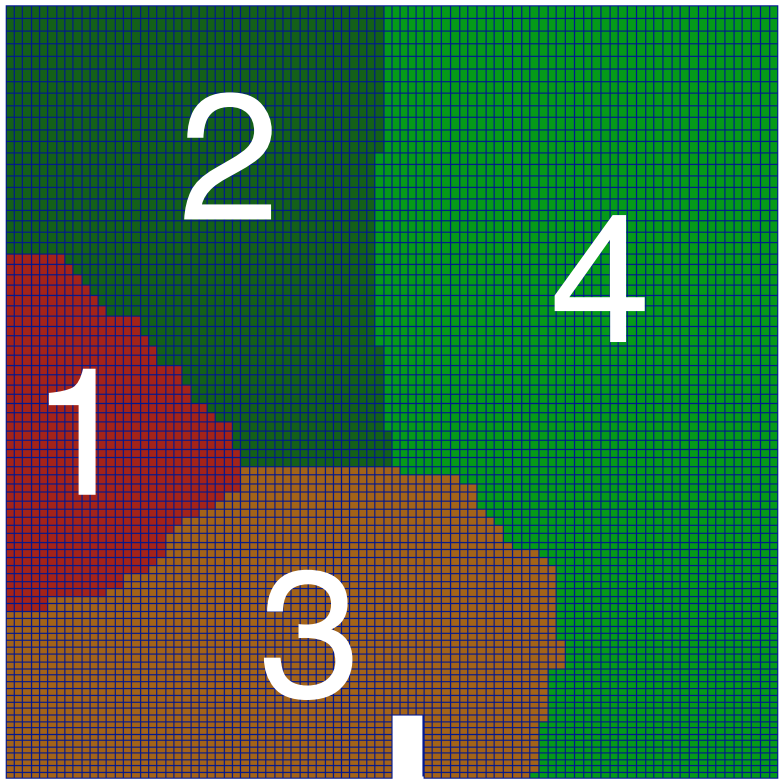
metis



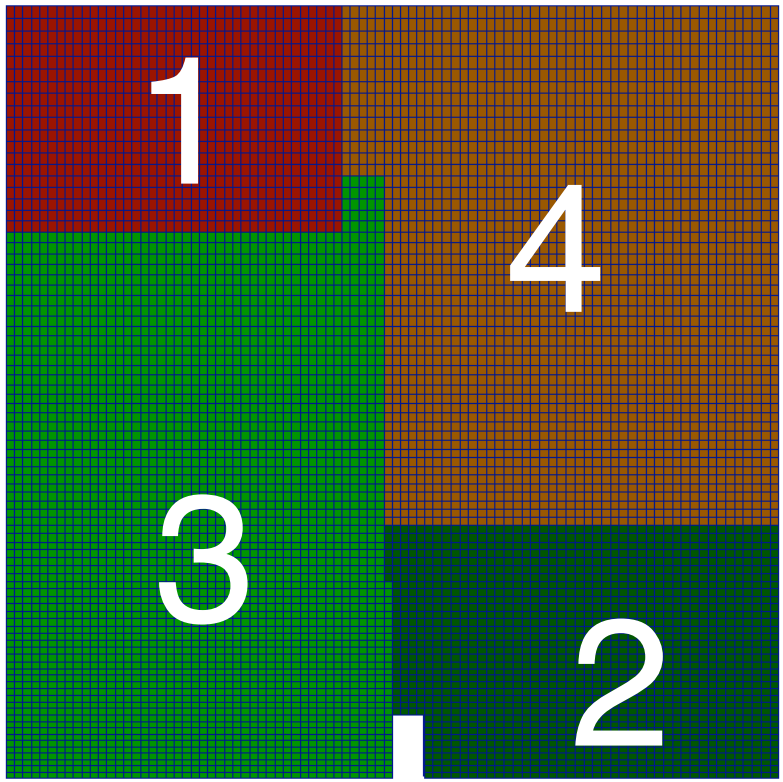
scotch

重み付き領域分割例

```
processorWeights ( 1 2 3 4 );
```



metis



scotch

領域分割結果

領域分割前

```
constant/polyMesh        ← 格子ファイル
0/{U, p, alpha1等}       ← 初期値
```

領域分割結果

```
processorN/                  ← プロセッサディレクトリ
  constant/polyMesh/         ← 分割済格子ファイル
  0/{U, p, alpha1等}         ← 分割済初期値
```

OpenCAE

並列計算

設定ファイル(同ノード内計算では不要)

machines

```
host1 cpu=N1    ←ホスト名とプロセッサ数を指定
host2           ←ホスト名だけも良い
```

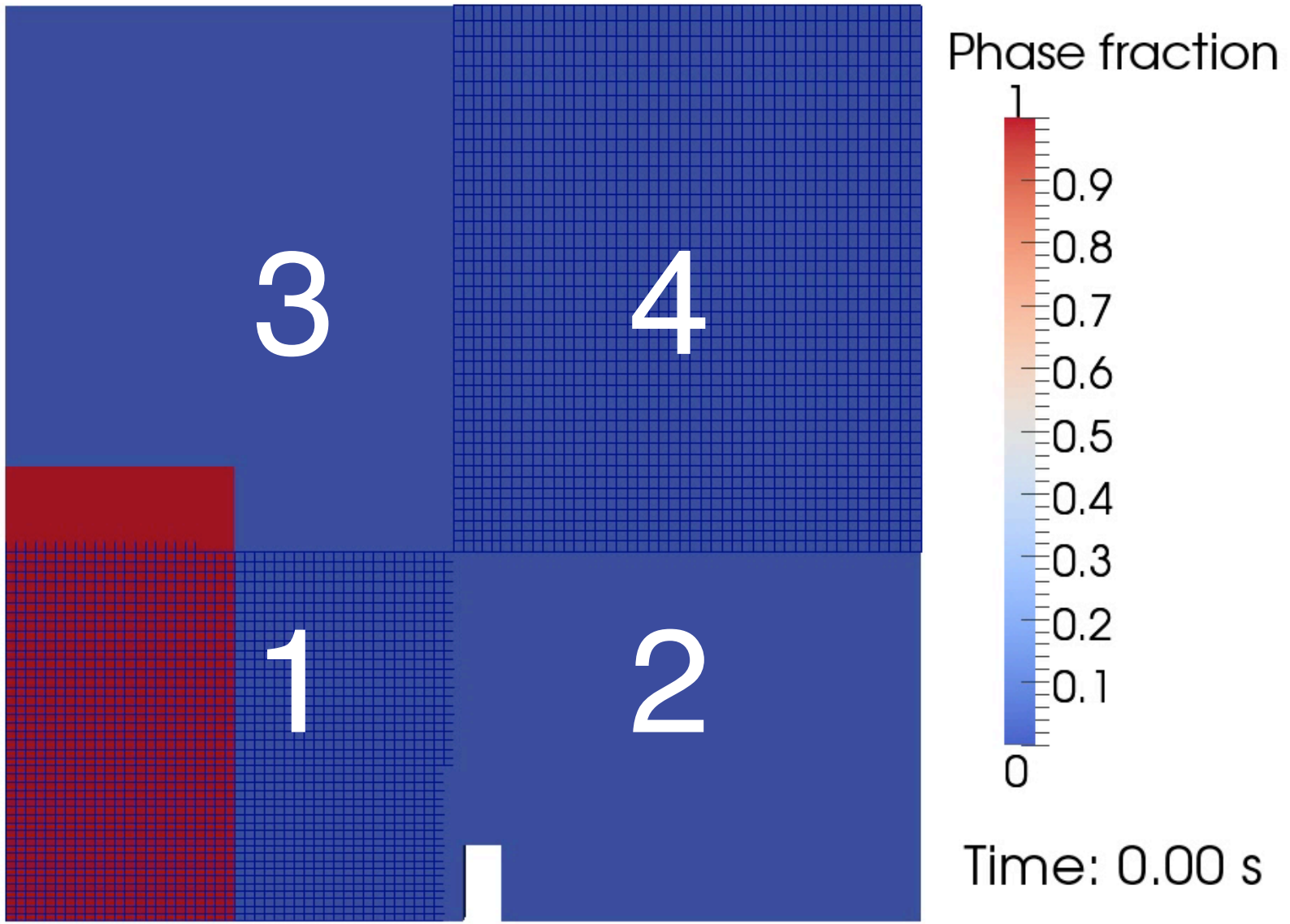
実行コマンド

```
mpirun --hostfile machines-np 4 interFoam -parallel
```

または以下(分割数やmachinesの有無等を検出して実行)

```
foamJob -p interFoam
```

OpenCAE



OpenCAE

分割結果再構築

実行コマンド

```
reconstructPar                    ←全時刻、全ての場を再構築
```

```
reconstructPar -latestTime        ←最終時刻のみ
```

```
reconstructPar -time 0.1:0.4      ←時刻指定
```

```
reconstructPar -fields “(U p)”    ←場の指定
```

OpenCAE

東京大学版T2Kオープンスパコンにおける OpenFOAMの並列計算

OpenCAE

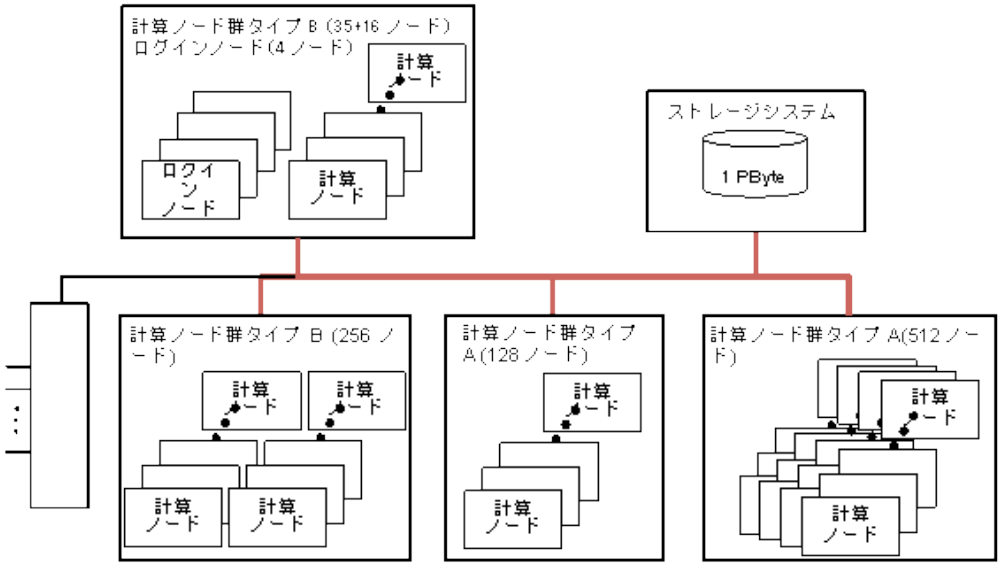
東京大学版T2Kオープンスパコン

(引用元:東京大学情報基盤センターHA8000クラスタシステム利用の手引)

OpenCAE

全体構成

総理論演算性能	140.1344TFLOPS
総主記憶容量	31.25TB
総ノード数	952
ストレージ装置容量	1PB(RAID6)



OpenCAE

計算ノード

ベクトル型計算機のSR11000等と異なり、
普通のラックマウント型PCによるスカラ型

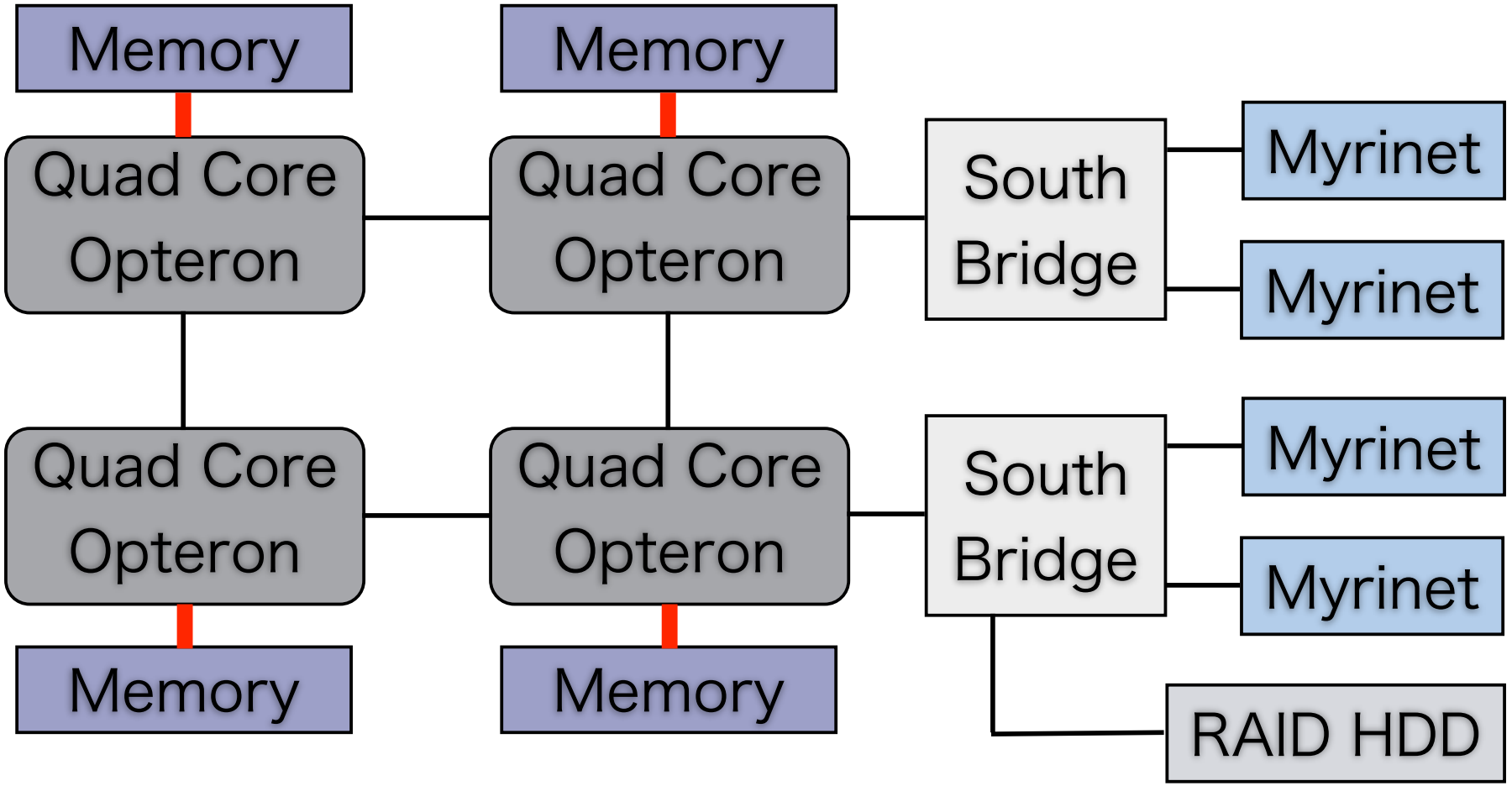
OpenCAE

計算ノードスペック

ノード	理論演算性能	147.2GFLOPS
	CPU数	4 (Quad Core)
	コア数	16
	主記憶容量	32GB(936ノード) 128GB(16ノード)
CPU	CPU (周波数)	AMD Opteron (2.3GHz)
	キャッシュメモリ	L2: 512kB/コア L3: 2MB/コア
	理論演算性能	9.2GFLOPS/コア

OpenCAE

アーキテクチャー



OpenCAE

ネットワークのスペック

構成	フルバイセクションバンド ネットワーク
ネットワークカード	Myrinet-10G
カード当りのバンド幅	1方向当り1.25GB/s (双方向に通信可能)
ノード当りのバンド幅 (カード数)	タイプA：5GB/s (4本) タイプB：2.5GB/s (2本)

OpenCAE

計算環境

- OpenFOAM Version: 1.6.x (git版)
- コードのカスタマイズ: 無し
- コンパイラ: gcc-4.3.3 (-O3 -msse3)
  - Intel-11.0 (-O3 -msse3)でも速度は変らず
- MPIライブラリ: Myrinet用のMPICH2-MX
- 並列化手法: ピュアMPI

OpenCAE

並列化効率の計算条件

- 対象問題： チャンネル流 (OpenFOAMチュートリアルのchannel395の格子数を増したもの)
- 乱流モデル：Smagorinsky LESモデル
- 格子数： 400万、3200万 (分割法: Metis)
- 時間計測：1Timeステップ終了後～50Timeステップ終了後までのCPU時間 (初期設定やファイルIO除外)

OpenCAE

並列化効率の算出法

スピードアップ比

$$= \frac{1\text{ノード時の計算時間}}{N\text{ノード時の計算時間}}$$

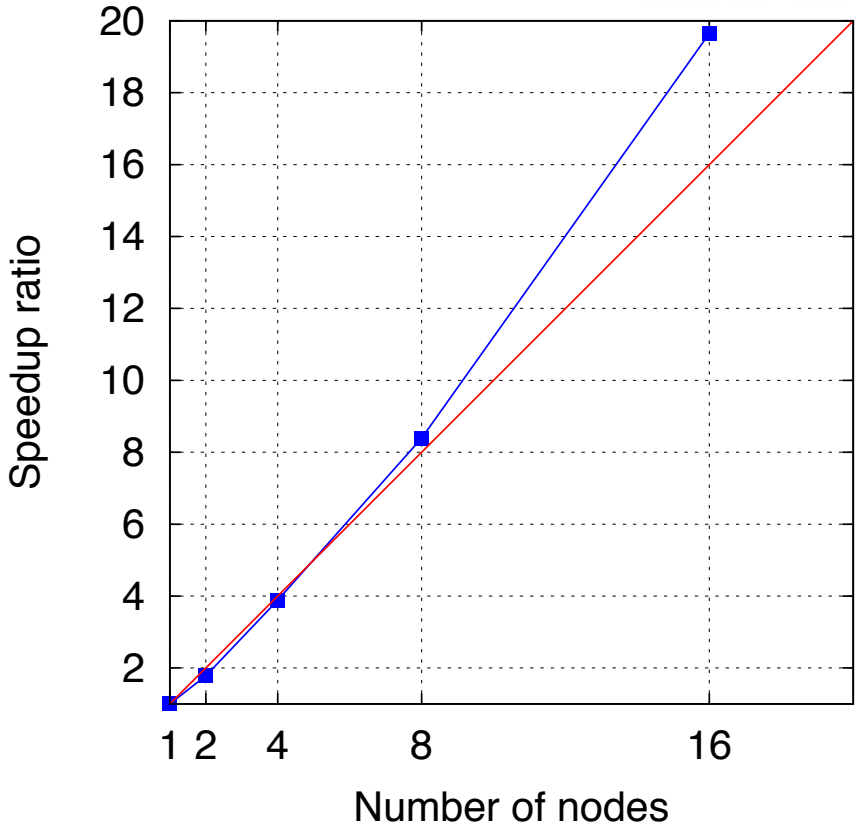
並列化効率

$$= \frac{\text{スピードアップ比}}{\text{ノード数}N}$$

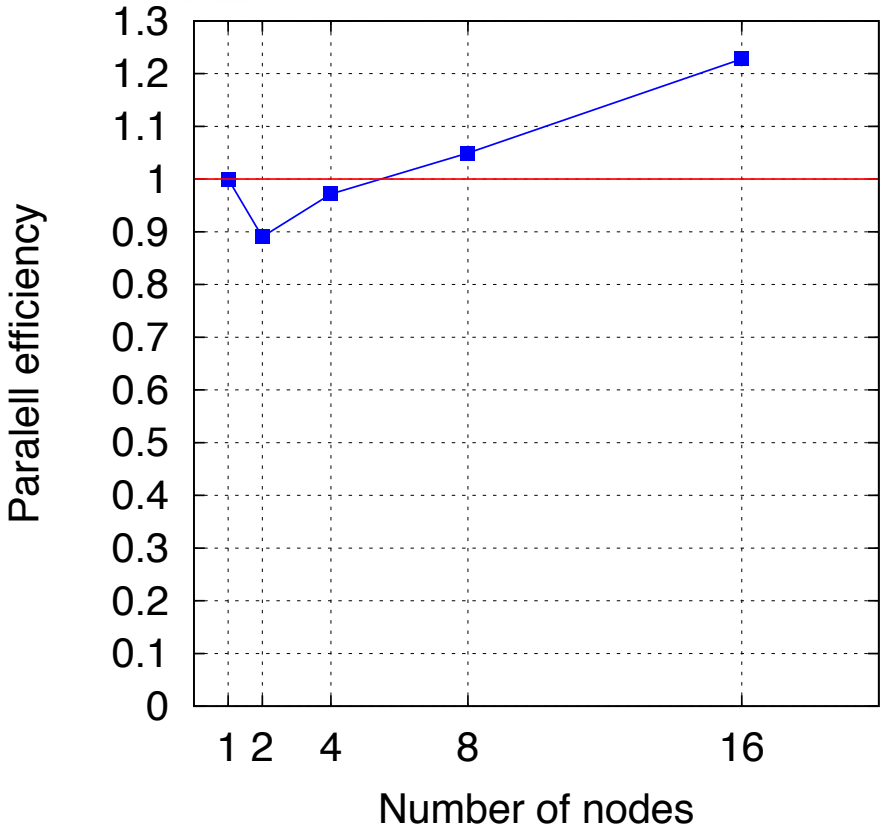
1ノード(16コア)から16ノード(256コア)まで
のスピードアップ比と並列化効率を計測

並列化効率

格子数: 400万



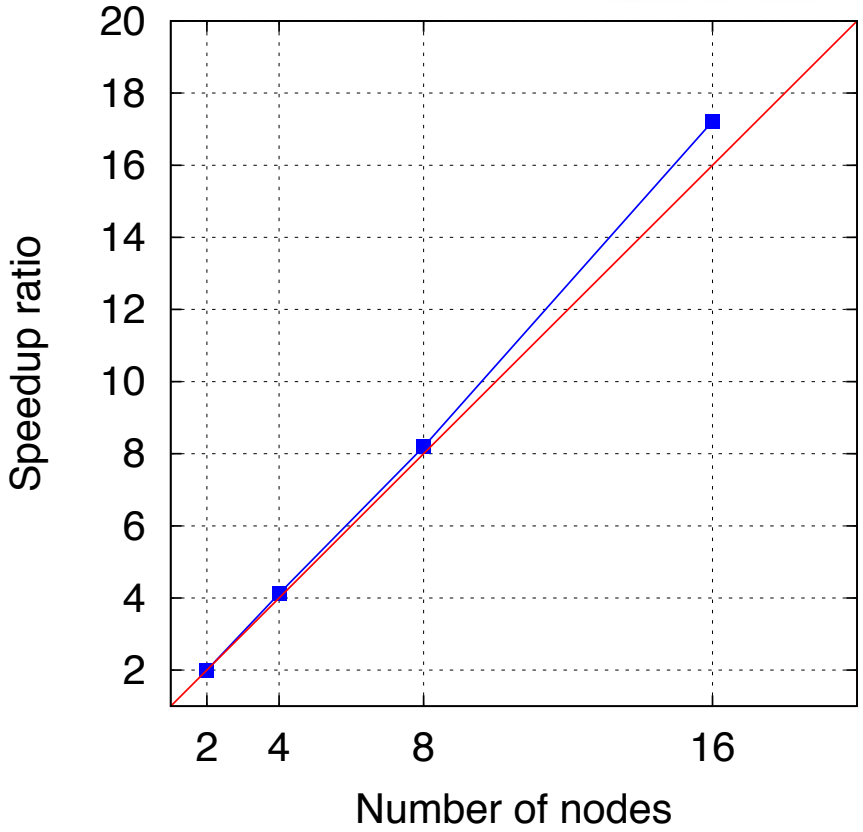
スピードアップ比



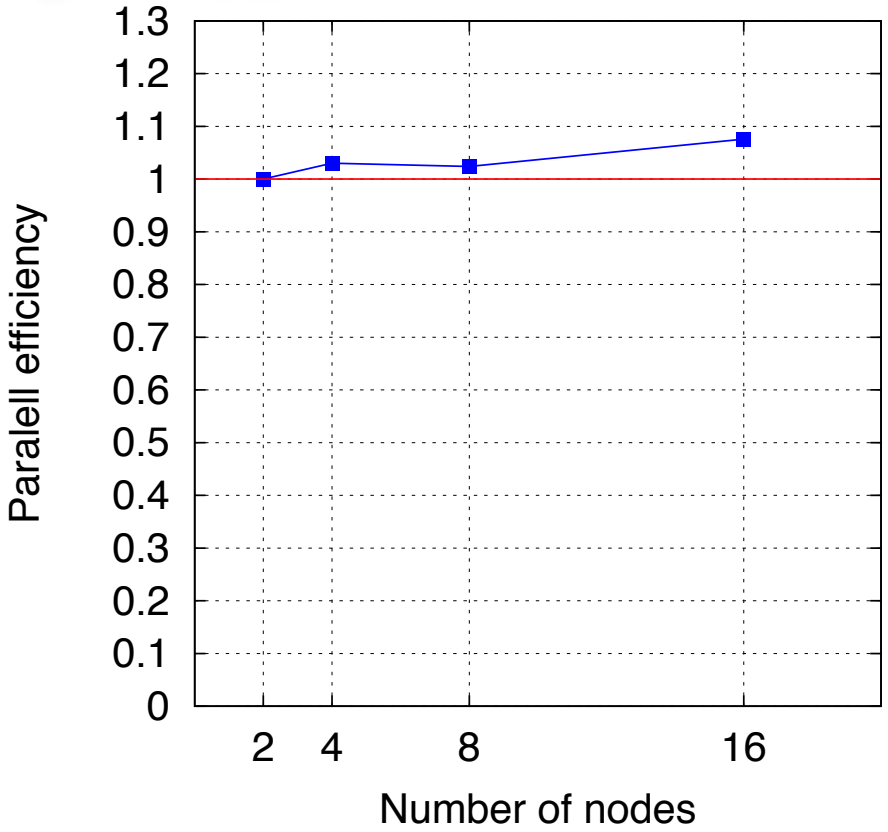
並列化効率

並列化効率

格子数: 3,200万



スピードアップ比



並列化効率

オープンCAEシンポジウム

- 2010年12月5日(日) 9：30-18：20頃
- 全体講演+技術講演+講習会+Ｌ．Ｔ．
- 東京大学工学部一号館
- 一般講演募集 (オープンソースCAE関連全部、15分)
- オーガナイズドセッション募集 (テーマ「オープンソースの実力、商用ソフトウェアの底力(仮)、15分)
- ライトニングトーク募集(1人5分、宣伝もOK)

OpenCAE

質問があればどうぞ

OpenCAE