最新CPU･GPUを用いたParaView並列可視化

オープンCAEシンポジウム2014
2014.11.14

吉田 正典, 千田 哲明, 桜井 泰樹, 熊木 竜也

株式会社 爆発研究所

概要

- Haswell世代のCPUと最新世代GPUを用いてParaView並列可視化を行った. 本講演では, 並列可視化のスケーラビリティに関して報告します.

画像出典
http://www.univpc.com/

機器構成

構成1: Workstation

	1ノード当たりの仕様
CPU	Xeon E5-2699 v3 (18core/2.3GHz) x 2
MEM	64GB (8GBx8) DDR4
GPU	Quadro K5200 x 1
SSD	480GB
NIC	GbE

	1ノード合計
CPU	36core (Rpeak:1324.8GFLOPS)
MEM	64GB
GPU	Quadro K5200 x 1

構成2: 4 node PC Cluster

	1ノード当たりの仕様
CPU	Core i7 5960X (8core/3.0GHz) x 1
MEM	32GB (8GBx4) DDR4
GPU	Quadro K2200 x 1
HDD	1TB
NIC	GbE

	4ノード合計
CPU	32core (Rpeak:1536GFLOPS)
MEM	128GB
GPU	Quadro K2200 x 4

GPU

- Quadro K5200

CUDA Cores	2304 (192 x 12SM)
GPU Memory	8GB GDDR5
Max Power Consumption	150W
Single Float Performance	3.1TFLOPs (*1)

- Quadro K2200

CUDA Cores	640 (128 x 5SM)
GPU Memory	4GB GDDR5
Max Power Consumption	68W
Single Float Performance	1.3TFLOPs (*1)

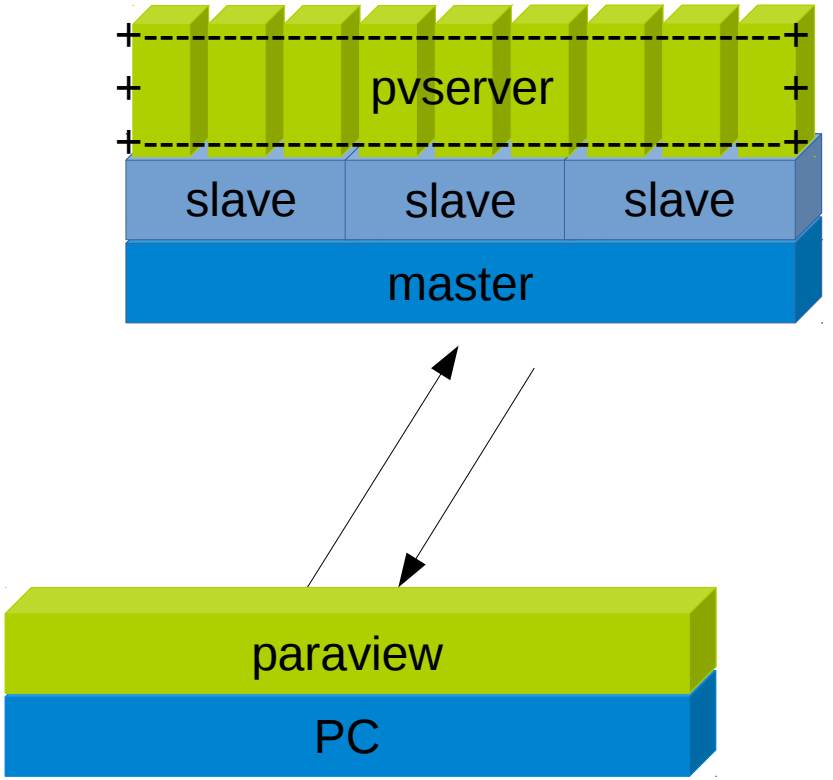
画像出典http://www.nvidia.com/

(*1)出典:http://www.nvidia.com

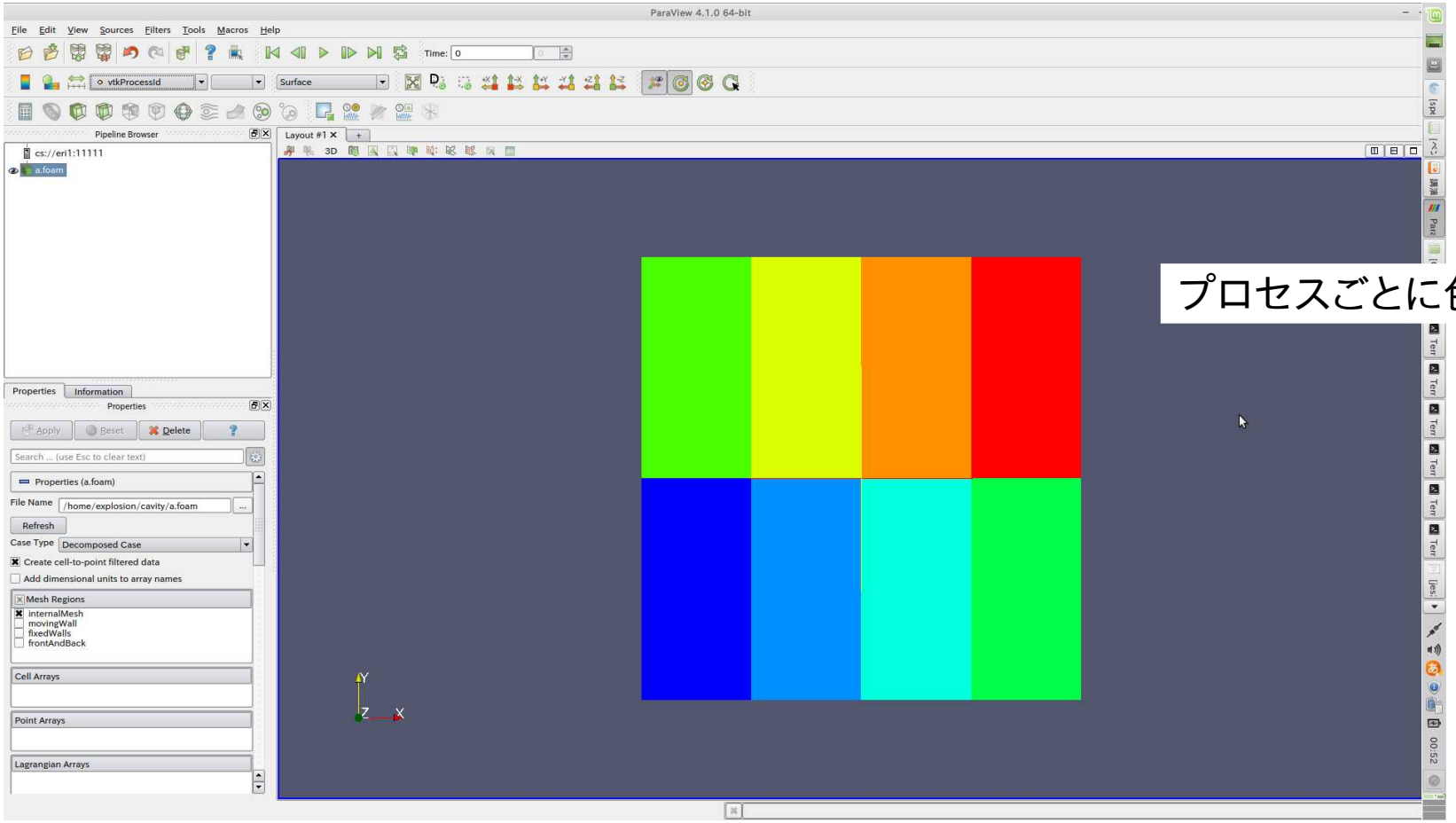
ParaView並列可視化

- Paraview Clientから pvserverへ接続

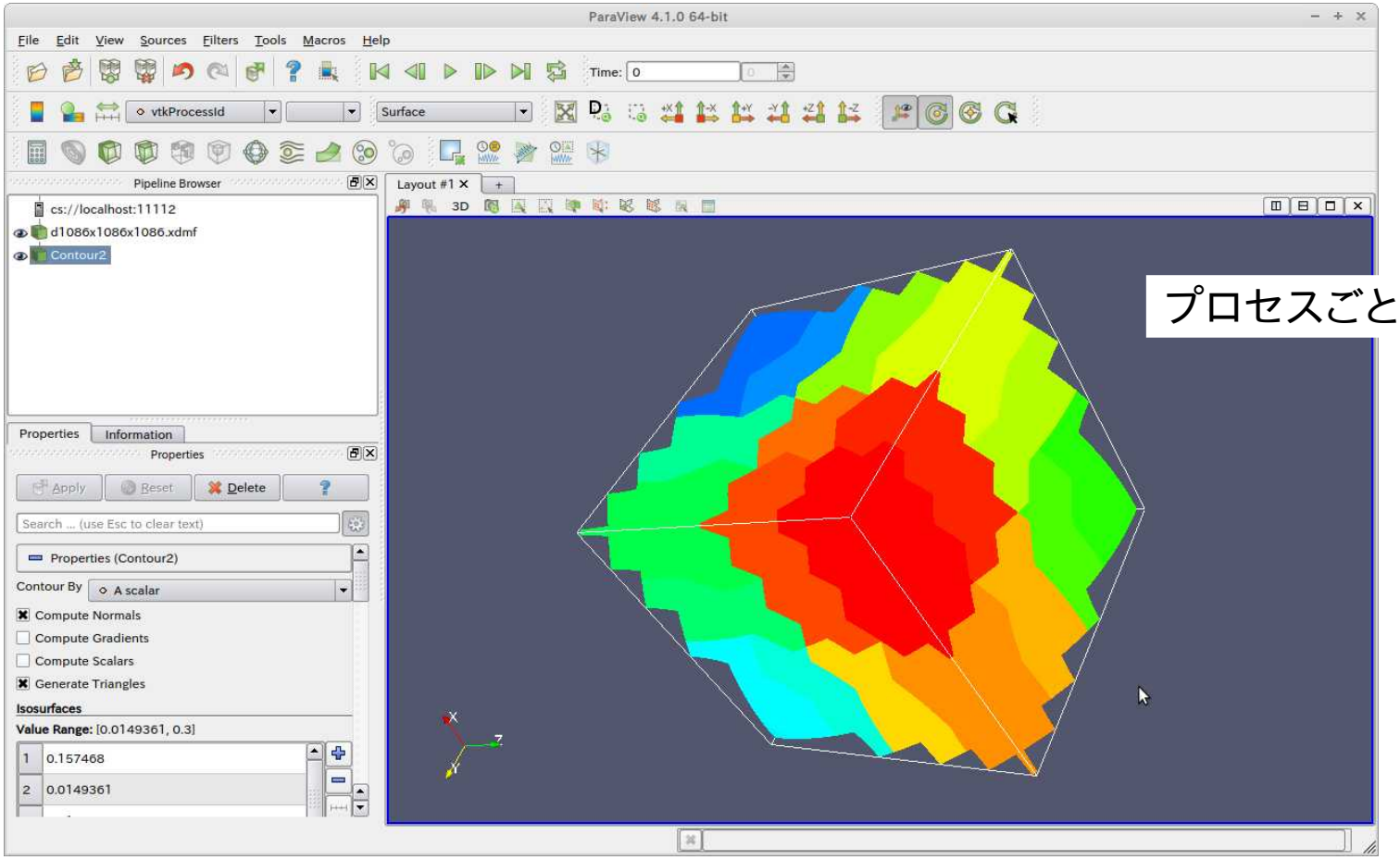
- pvserver
  - 並列データ読み込み
  - 並列レンダリング
  - 結果をクライアントに送る



プロセスごとに分割領域を割り当て



プロセスごとに分割領域を割り当て



プロセスごとに色分け

Paraviewセットアップ

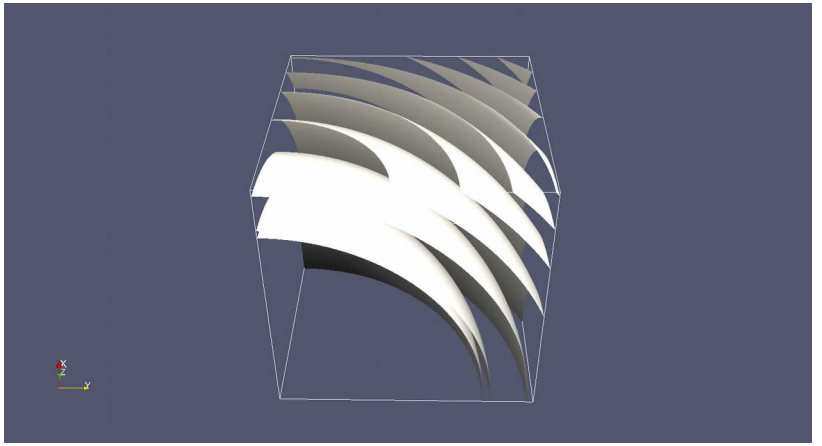
- Paraview 4.1.0 (**nvidia**/libGL, OpenMPI)
- Paraview 4.1.0 (**mesa**/libGL, OpenMPI)
  - OSはXubuntu 14.04
  - 1つのシステムに上記2種類のParaviewをセットアップ
  - OpenGL Libraryを切り替えて使用

    GPU rendering
    CPU rendering

Benchmark Dataset

データ 1

「XDMFファイル」



格子数　　ファイルサイズ

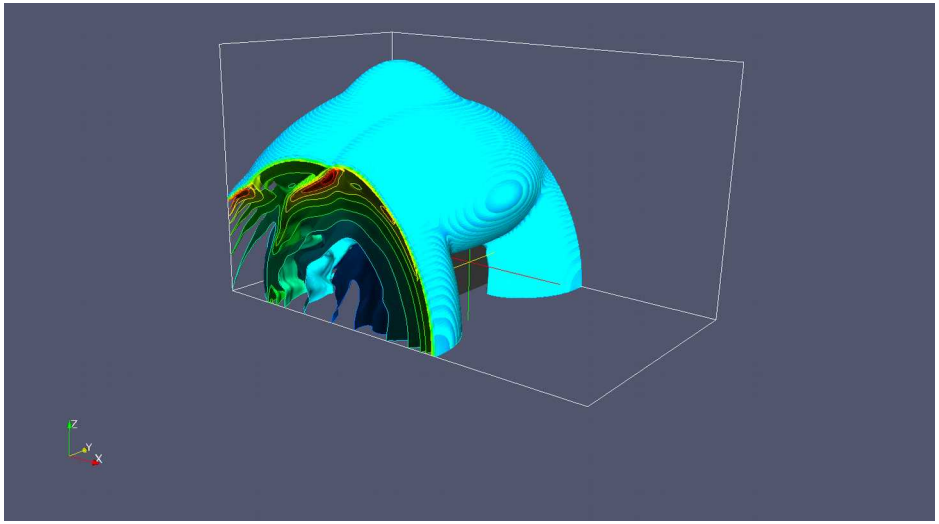
- 80M　・・・0.6G
- 160M ・・・1.2G
- 320M ・・・2.4G
- 640M ・・・4.8G
- 1280M・・・9.6G

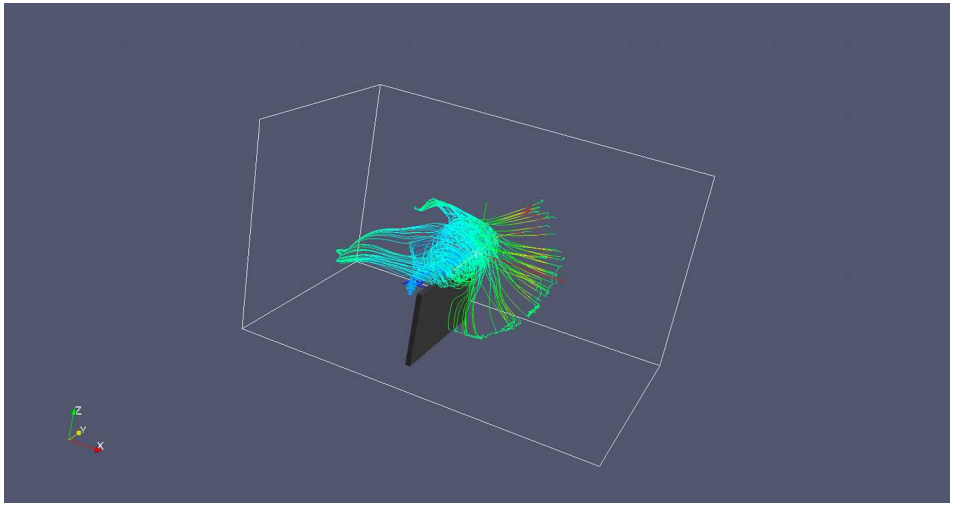
- 直交等間隔構造格子
- 球対象プロファイル
- 全ノードに配置(RAMDISK使用)

Benchmark Dataset

データ 2

「OpenFOAM計算結果」





格子数　　ファイルサイズ

- 1.5M　・・・4.3G
- 6.75M・・・17G

- rhoCentralFoamの計算結果
  - 防爆壁を含む三次元計算(*1)
- 全ノードに配置(RAMDISK使用)

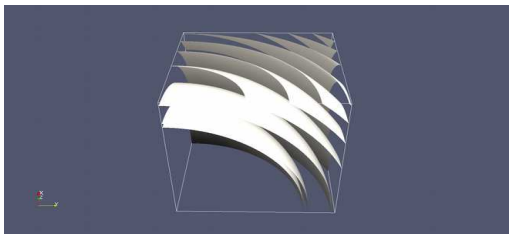
(*1)「OpenFOAMを用いた爆風伝搬の並列計算」　熊木竜也,吉田正典 株式会社爆発研究所
2012年度　火薬学会

Benchmark Menu

データ 1
「XDMFファイル」

タスク

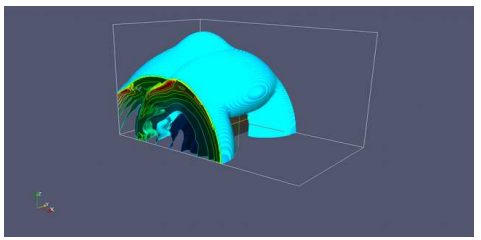
- 等値面を描画

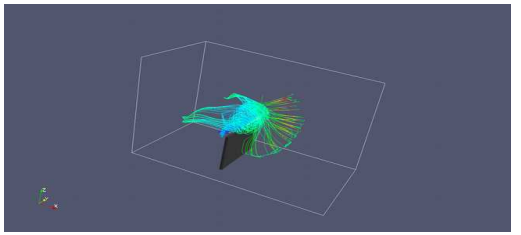


データ 2
「OpenFOAM計算結果」

タスク

- 等値面の時間発展
- 流線の時間発展



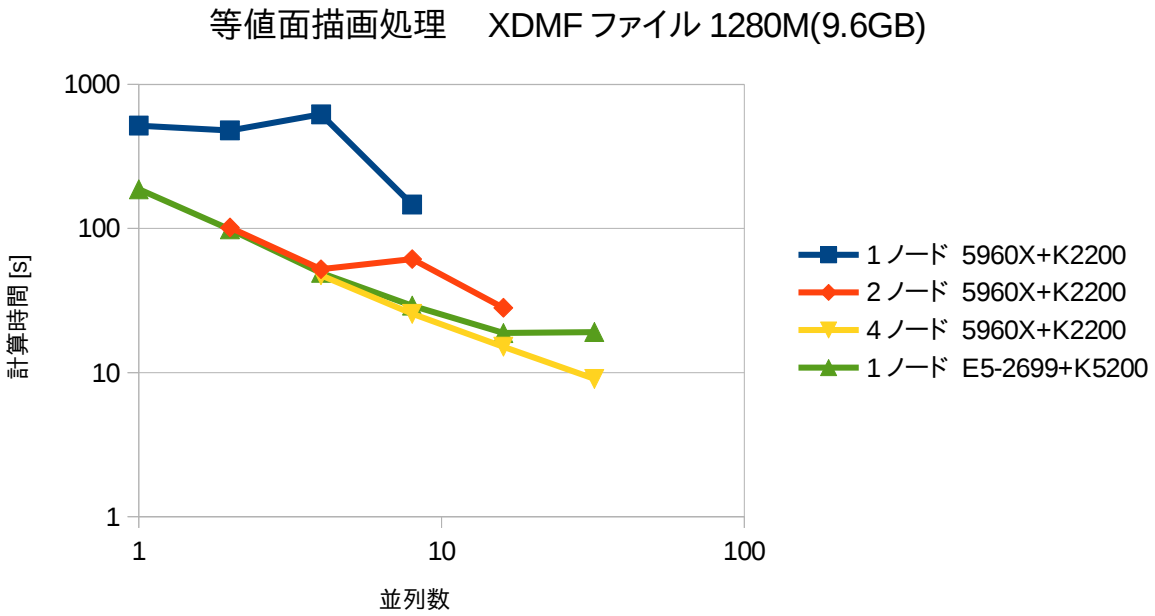


いずれも処理時間をParaview > Tools > Timer logで取得
全プロセスの平均処理時間を記録

Timer Log (320M data, 4n32pの例)

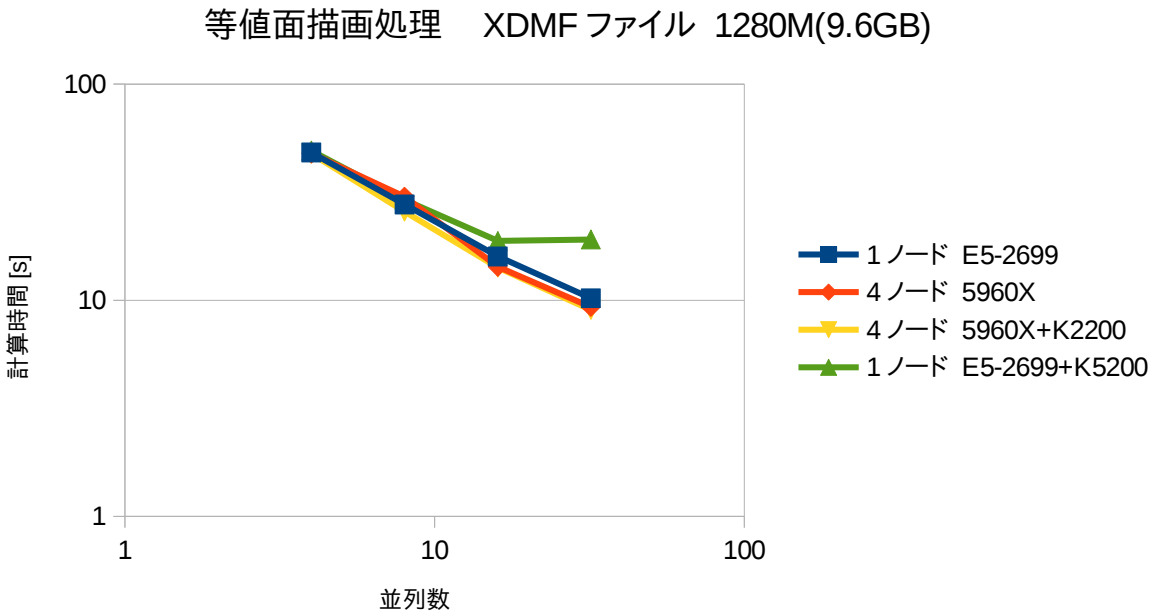
```
Render Server, Process 0
Execute vtkXdmfReader id: 3512,  0.057566 seconds
FullRes Data Migration,  0.028664 seconds
   Dataserver gathering to 0,  0.015983 seconds
   Dataserver gathering to 0,  0.011153 seconds
Execute vtkXdmfReader id: 3512,  0.058754 seconds
Execute vtkPVContourFilter id: 3667,  1.36551 seconds
RenderView::Update,  0.064183 seconds
   vtkPVView::Update,  0.062643 seconds
Still Render,  1.2849 seconds
   OpenGL Dev Render,  0.620826 seconds
```

(1)XDMFファイル 等値面描画処理



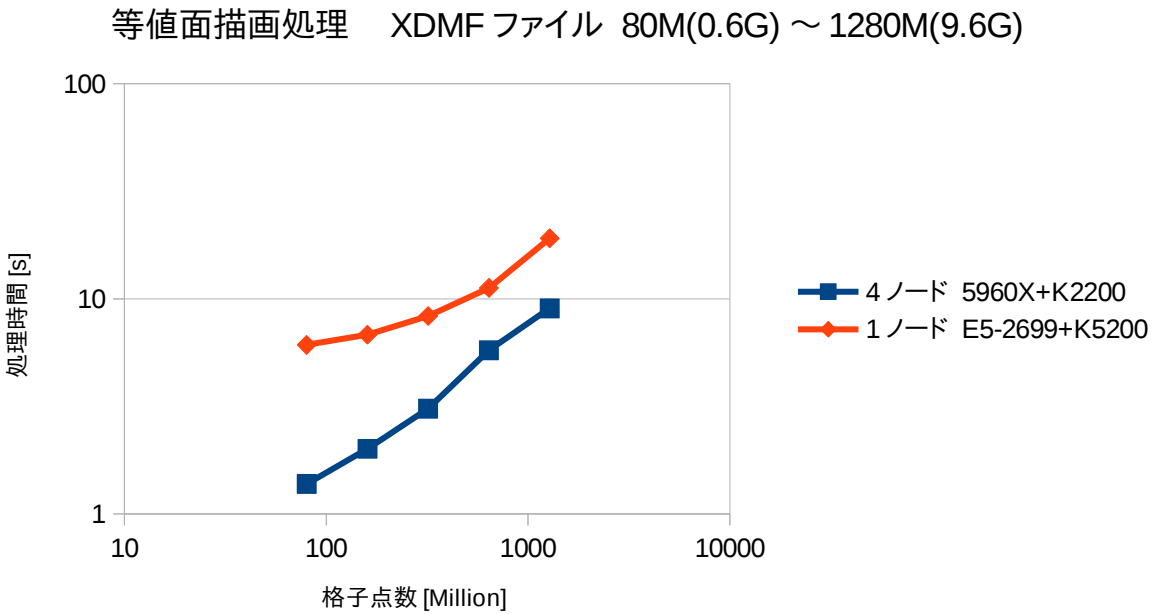
- 4ノードPCクラスタが最も良いスケール性を示した。
  等値面描画処理における**分散ディスクの有効性**が確認できた。
- 「1ノードワークステーション16並列」よりも
  「4ノードPCクラスタ16並列」が良い結果を示した。

(2)XDMFファイル 等値面描画処理



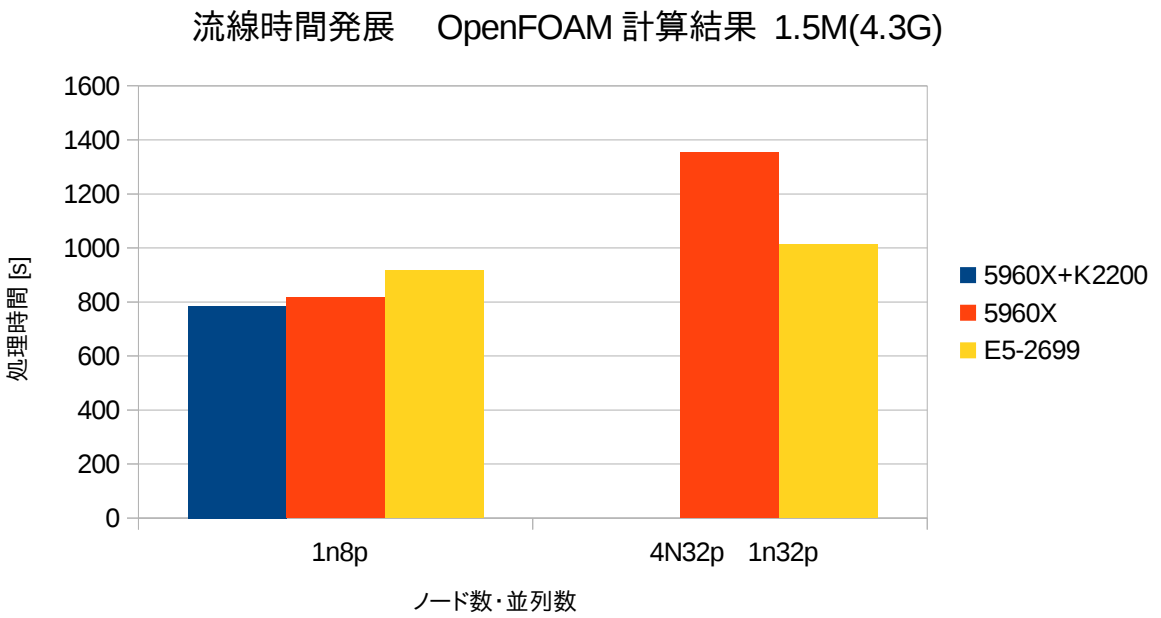
- 1ノードワークステーション（GPU有り,K5200）のスケール性が16並列当たりから飽和を示す理由は、1枚のGPUに対して32プロセスが同時に処理を要求していることが原因と考えられる
- 4ノードPCクラスタでは、僅かであるが「GPU有り,K2200」の場合の方が「GPU無し(CPUによるOpenGL処理)」よりも良いスケール性を示した。

(3)XDMFファイル 80M(0.6G)～1280M(9.6G)



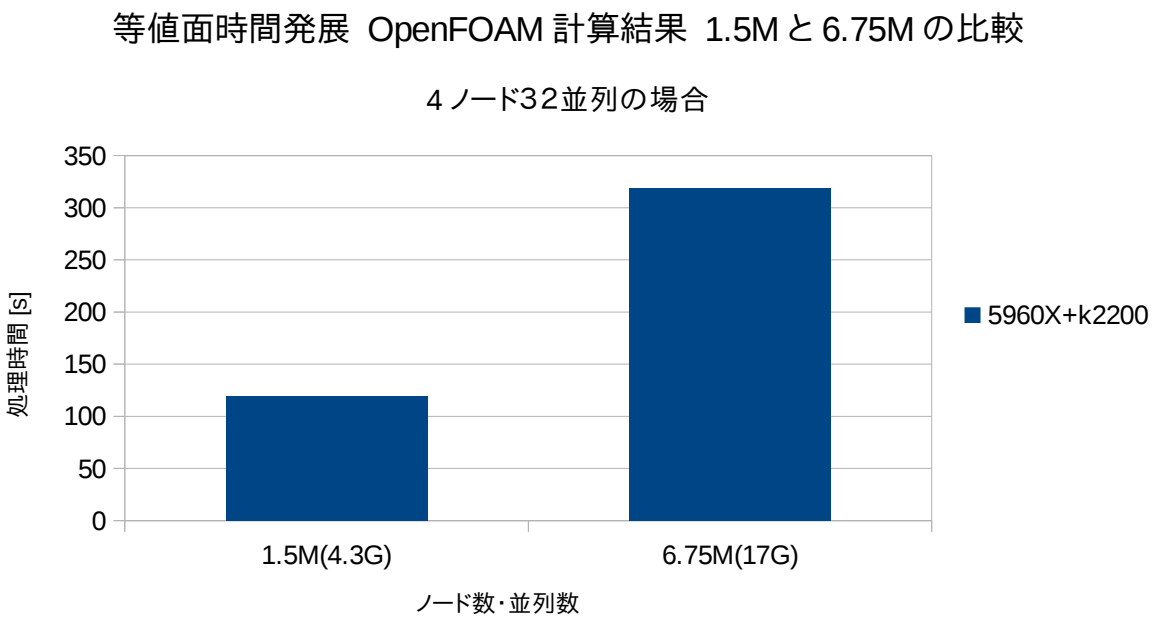
- 1ノードワークステーションは、データサイズが大きくなるに連れて性能が低下した。
- 4ノードクラスタは、データサイズに対してリニアなスケール性を示した。
  分散ディスク、複数のGPUに負荷分散していることの効果が現れていると
  考えられる。

(4)流線時間発展 1.5M(4.3G)



- 「4ノードPCクラスタ（32並列）」と比較して、
  「1ノードワークステーション（32並列）」は処理時間が短い。
  MPI通信がボトルネックになっていると考えられる。
- 4ノードPCクラスタは、Gigabit Ethernetで接続しているため、
  Infinibandや10GbE,40GbEを用いることで性能改善の見込みがある。

(5)等値面時間発展



- 4ノード32並列の場合
  データサイズを約4倍（4.3Gから17G)に設定しても、
  処理時間は約3倍にとどまり、良いスケール性を示している