Vol.2012-HPC-136 No.7
2012/10/3

Amazon EC2 GPUクラウドにおけるOpenFOAM流体計算の性能評価

西條 晶彦[1]　井口 寧[1,2]　松澤 照男[1,3]

概要：計算機をVMの形で使った時間の分だけ借りるクラウド形態の計算機の利用が進んでいる．本研究ではHPC分　野における HPC クラウド計算機として Amazo EC2 の GPGPU ノードを用いて，実用的な流体計算アプリケーションである OpeFOAM を用い，GPU クラウド上での並列血流解析計算による性能評価を行った．OpenFOAM の MPI 対応 GPGPU 線形ソルバを開発し適用した．その結果，8 並列の EC2 ノード上においてもスケールできることがわかった．

キーワード：GPU, Cloud, CFD

Akihiko Saijo[1]　Yasushi Inoguchi[1,2]　Teruo Matsuzawa[1,3]

1. はじめに

HPC のクラウド形態による利用が行われつつある．従来の据え置き (Inhouse) 型ではクラスタマシンを購入あるいはリースして集団で使うのに対し，クラウドでは個人が欲しいだけの計算機資源に対して料金を支払うことでその場で仮想マシンの形で計算機資源を借り受けるというものである．このような IaaS (Infrastructre as a Service) 形態のクラウド計算機利用は大規模データ解析などの分野では既に多く利用されており，据え置き型の PC の需要を置き換えている．しかしながらクラウドにおける HPC 分野での利用は未だ少ない．

クラウド HPC を用いるメリットは提供されるマシンが仮想マシンであるため，アプリケーションやライブラリ，コンパイラ環境を利用者個人が好きなように設定することが出来ることである．もちろん据え置き型のマシンも管理者権限があれば可能であるが集団で使う利用者の利便性が高いというものがある．また，単発的に小規模な計算を行うような使用例の場合，据え置き型の HPC を一定期間借りるよりもコストの面で有利な場合がある．一方，クラウド HPC のデメリットとしては遅い MPI 通信速度や仮想コア CPU のための性能低下，性能のブレがある．HPC クラウドが実際の研究用途に十分な性能があるかという問題はアプリケーションの性質に依存する．MPI 通信をほとんど必要としないアプリケーションの場合は高い性能が出る可能性がある一方，MPI 密結合な場合は高速な通信インフラを持つ据え置き型 HPC と比べて性能が非常に低下することが予想される．実際のクラウドアプリケーションの性能はプログラムの構成や通信のパターンに強く依存する．

HPC 向けのクラウドサービスに Amazon EC2 (Elastic Compute Cloud) がある．EC2 サービスの HPC 計算ノードとして CCI (Cluster Compute Instance) があり，HPC 向けの高性能ノードを持っている．CCI では NVIDIA の CUDA 対応 GPU カードを搭載した GPU CCI がある．HPC の計算を EC2 CCI で行う研究はいくつかあるが，ほとんどがマイクロベンチマークやそれに近いアプリケーションによる計算であり，クラウド計算機で実用的な HPC 計算をおこなうべきか否かという問いは未だ明らかではない．

本研究では Amazon EC2 の GPU CCI を用いて，実際的な流体科学計算に用いられる OpenFOAM を GPU CCI に対応させて実行，Inhouse 型の GPU クラスタと性能を比較した，Amazon EC2 におけるクラウド HPC 計算がどれほど有効であるかを調査した．

[1] 北陸先端科学技術大学院大学 情報科学科
Japan Advanced Institute of Technology and Science, School of Infomation
[2] 情報社会基盤研究センター
[3] シミュレーション科学研究センター

2. Amazon GPU Cluster Compute Instance

前述のように Amazon CCI は Amazon EC2 によって提供されるクラウド仮想マシンであり，科学技術計算に適した高性能仕様の計算機である．本研究では CCI のうち，GPU を搭載しているインスタンスである “Cluster GPU Quadruple Extra Large Instance” (cg1.4xlarge) [6] を On Demand 形態で使用する．比較のために Inhouse の Infiniband GPU Cluster (pcc-gpu) を用いる．それぞれの仕様は表 1 にまとめた．

3. 環境セットアップと MPI 性能評価

3.1 環境セットアップ

本研究では Amaozon EC2 の GPU CCI を利用する．GPU CCI のノードは現在，アメリカ合衆国東部 (Virgina) 地域のみで提供されており，日本から利用する場合は操作やファイルの転送にやや遅延がある．また，デフォルトでは最大 2 ノードしか GPU インスタンスを立ちあげられないので，それ以上のノード数を利用する場合はさらに Amazon に申請して利用可能ノードを引き上げてもらう必要がある．

CCI を利用することは非常に容易である．ブラウザからアクセスできる管理コンソールが用意されており，その上で VM，ストレージの管理，ネットワークの設定などを行う．最初にインスタンスを設定したのち，それをテンプレートとした VM を複数立ち上げることで仮想クラスタを構築できる．また，API を経由して端末からコマンドで VM を設定することもできる．

計算に使用する OS として Cluster GPU Amazon Linux AMI の 2012.03 版を使用した．これは Red Hat Enterprise Linux をベースにした Amazon の提供するマシンイメージであり，MPI 通信と GPU を実行するのに必要な HVM (Hardware Virtual Module) や CUDA を使うためのツールキットがあらかじめインストールされている．

EC2 CCI 上のクラスタ構築ツールとして StarCluster[7] や OpenFOAM の EC2 用 VM である Cloud-Flu[8] があるが，本研究では GPU コードを使うためツールは用いず，GCC コンパイラで OpenFOAM をコンパイルし，システムにインストールした OpenMPI を使用してクラスタ環境を構築した．また Xeon の機能である HyperThreading は性能低下やプログラムの予期せぬ終了を招くことがあるので無効にした．

ストレージは各ノードがそれぞれ OS のためのローカルのルートストレージを持つ．並列計算のためには各ノードから NFS などで永続ストレージである EBS(Elastic Block Store) ボリュームをマウントすることもできるが，OpenFOAM は各ノードのローカルストレージで I/O を行うことが出来るため共有ストレージは使用しなかった．

3.2 MPI ベンチマーク

Intel MPI Benchmarks (IMB) による MPI 通信性能の比較を行った．比較するのは流体の並列計算で用いる，ノード間の袖領域 (Ghost Cell) の通信と倍精度浮動小数点の縮約通信である．最初の例は 2 ノード（インスタンス）間で IMB PingPong のメッセージサイズを変化させて実行時間を計測した． 図 1 が CCI (cg1.4xlarge) と Inhouse クラスタ (pcc-gpu) の結果である． メッセージサイズが大きくなるに従い，CCI の通信は 6 倍近く遅くなることがわかる．縮約通信はサイズを 8bytes に固定し，ノードの並列数を変化させた．図 2 が結果である．CCI における縮約通信のレイテンシは非常に悪く，100 倍遅いということがわかる．このような遅い通信を避けるコードでなければ EC2 上では高速にならないことがわかる．

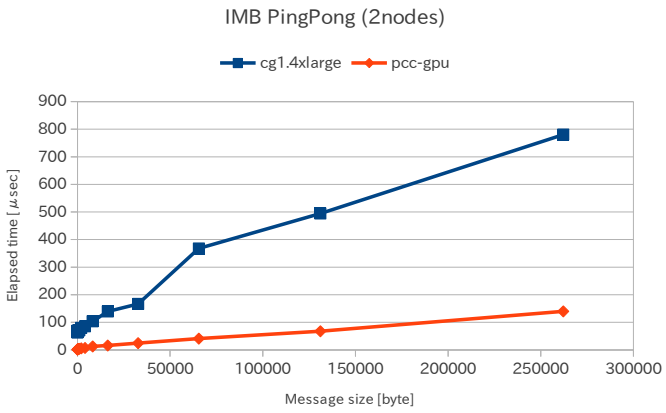

図 1 メッセージサイズに対する 2 ノード間通信時間: EC2 CCI vs. Inhouse Cluster

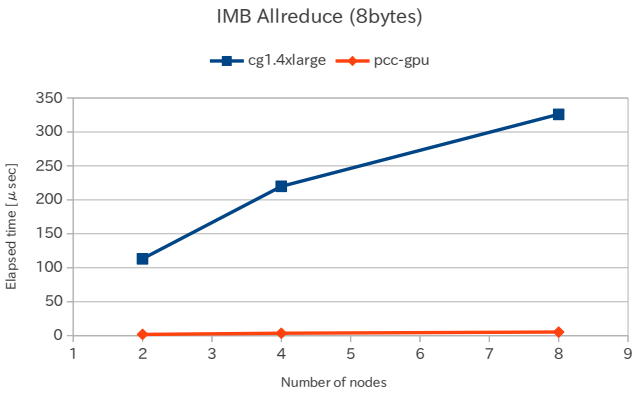

図 2 ノード数に対する全体縮約通信時間: EC2 CCI vs. Inhouse Cluster

表 1 EC2 GPU Cluster Instance と Inhouse GPU Cluster の仕様
Table 1 Specifications

名前	cg1.4xlarge	pcc-gpu
CPU	Intel Xeon X5570 2.93 GHz	AMD Opteron 6136 2.4 GHz
CPU 数 (コア数)	2(8 w/o HyperThreading)	2(16)
メモリ	22 GB	32 GB
GPUs	NVIDIA Tesla M2050 × 2	
ネットワーク	10 Gigabit Ethernet	Infiniband QDR
OS	Cluster GPU Amazon Linux AMI 2012.03	CentOS 6.2
コンパイラ	GNU GCC 4.4.6 Options: -O2 -fPIC	
CUDA Version	NVIDIA CUDA 4.2	CUDA 4.1
MPI Library	Open MPI 1.5.3	MVAPICH2 1.7

4. OpenFOAM GPU 流体計算

以上のベンチマークの結果を考慮し，OpenFOAM による血管内流れ解析の計算を行う．

本研究での血管内流れ解析には定常非圧縮性粘性流体の物理モデルを用い，圧力解法には SIMPLE (Semi-Implicit Methods Pressure-Linked Equations) 法を用いる．支配方程式は定常非圧縮粘性流れの連続の式と (外力項のない) Navier-Stokes 方程式（運動量方程式）である．

$$\begin{cases} \nabla \cdot (\rho \mathbf{U}) = 0, \\ (\mathbf{U} \cdot \nabla)\,\mathbf{U} - \nabla \cdot (\nu \nabla \mathbf{U}) = -\nabla P \end{cases} \tag{1}$$

OpenFOAM における SIMPLE 法ソルバではこの 2 式を有限体積法 (FVM) により離散化し，以下の圧力解法スキームを用いて物理量を算出する [3].

節点 (Node) p における運動量方程式は次のように離散化される．

$$\begin{aligned} a_p \mathbf{U}_p &= H(\mathbf{U}) - \nabla P \\ \Rightarrow \mathbf{U}_p &= \frac{H(\mathbf{U})}{a_p} - \frac{\nabla P}{a_p}, \\ where \quad H(\mathbf{U}) &= -\sum_{n \in NEIGH(p)} a_n \mathbf{U}_n. \end{aligned} \tag{2}$$

ここで a_p は移流項を上流差分で離散化して得られる U_p の係数，$H(\mathbf{U})$ は p 点の近接セルの離散化行列に $\mathbf{U}$ を作用させたものを表している．

連続の式の離散化は次のようになる．

$$\nabla \cdot \mathbf{U} = \sum_{f \in FACE} \mathbf{S}\mathbf{U}_f \tag{3}$$

ここで $\mathbf{S}$ は FVM の検査体積の境界面に外向きに垂直な面ベクトルであり，$\mathbf{U}_f$ はその面での速度である．

境界面 f における速度は離散化された運動量方程式において補間を用いて得ることが出来る．

$$\mathbf{U}_f = \left(\frac{H(\mathbf{U})}{a_p}\right)_f - \frac{(\nabla P)_f}{(a_p)_f} \tag{4}$$

Algorithm 1 SIMPLE 法
1: 境界条件の設定
2: **repeat**
3: 離散化された運動量方程式を解いて中間的な速度場を算出
4: セル界面の質量流束の計算
5: PCG 法で圧力方程式を解き，不足緩和を適用
6: セル界面での質量流束を修正
7: 新しい圧力場から速度場を修正
8: 境界条件の更新
9: **until** 圧力場と速度場が閾値以下に収束するまで

上式を離散化された連続の式に代入して圧力の方程式が得られる．

$$\begin{aligned} \nabla \cdot \left(\frac{1}{a_p} \nabla P\right) &= \nabla \cdot \left(\frac{H(\mathbf{U})}{a_p}\right) \\ &= \sum_f \mathbf{S} \left(\frac{H(\mathbf{U})}{a_p}\right)_f \end{aligned} \tag{5}$$

この圧力の方程式を中央差分で離散化すること圧力場の線型方程式 $A.\mathbf{x} = \mathbf{b}$ が得られる ($\mathbf{x}$ は圧力の節点のベクトル $[P_1, P_2, \ldots, P_N]$, $\mathbf{b}$ は対応する右辺のベクトル)．係数行列 A は対称となり，方程式は前処理付き CG 法によって高速に解くことが出来る．

SIMPLE 法では現在のステップの速度場から中間的な速度場を算出し，これを用いて圧力の線型方程式を解く．収束条件のループを減らすために，不足緩和を行う [4]．新しい圧力場が連続の式を満たすように速度場を修正する．圧力場と速度場が収束するまでこのループを繰り返す．ここでは、この速度場修正のための反復を CG 法の反復と区別するために前者を外部反復，後者を内部反復と呼ぶ．

4.1 線型ソルバの GPU 化

前処理付き CG 法はアルゴリズム 2 に擬似コードの形で表す**?**．ここでのベクトル $\mathbf{p}$ などの表記は前節の物理的な圧力などと関係がない．

PCG 法で計算時間のかかるものはこのうちの疎行列ベクトル積 (SpMV) と前処理の適用部である．本研究では GPU における SpMV のルーチンに Li と Saad による

Algorithm 2 Parallel Preconditioned Conjuagte Gradient

1: Given $\mathbf{x}_0$.
2: Let $\mathbf{p}_0 = \mathbf{b} - A\mathbf{x}_0$, $\mathbf{z}_0 = M^{-1}\mathbf{r}_0$, $\mathbf{r}_0 = \mathbf{p}_0$, $k = 0$.
3: **repeat**
4: MPI Send GHOST CELLS of $\mathbf{p}_k$.
5: $\mathbf{q}_k = A\mathbf{p}_k$
6: MPI Recv GHOST CELLS of $\mathbf{q}_k$.
7: $\alpha_k = \mathbf{p}_k^T\mathbf{r}_k / \mathbf{p}_k^T\mathbf{q}_k$
8: MPI Allreduce SUM α_k.
9: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$
10: $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k\mathbf{q}_k$
11: $\mathbf{z}_{k+1} = M^{-1}\mathbf{r}_{k+1}$
12: $\beta_k = \mathbf{r}_{k+1}^T\mathbf{q}_k / \mathbf{p}_k^T\mathbf{q}_k$
13: MPI Allreduce SUM β_k.
14: $\mathbf{p}_{k+1} = -\mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$
15: $k = k + 1$
16: **until** $(\|\mathbf{r}_{k+1}\|/\|\mathbf{r}_0\| \leq \epsilon)$

線型ソルバライブラリである CUDA ITSOL[2] を用いる．SpMV の高速化には疎行列格納形式として行列要素を対角方向に保持する JAD (JAgged Diagonal) 格納形式を用いる．

また，SIMPLE 法では圧力と速度が収束するまで外部反復を繰り返す．その度に線型ソルバは何度も呼ばれるため全体の高速化のためには線型ソルバの初期化時間も高速化する必要がある．係数行列 A を OpenFOAM の行列格納形式から JAD 形式に変換し，この行列から導かれる前処理行列 M を作成しなおす必要がある．ここで，式 (5) の左辺から分かるように SIMPLE 法では与えられる係数行列の値は外部反復ごとに変更されるが，非ゼロ要素の位置は節点の結合度によって決まるため変化しない．これより，内部反復の最初に JAD 形式への要素の並び替え順をキャッシュしておくことで，2 回目の外部反復以降では行列値のみを並べ替えるだけになり高速に変換できる．

4.1.1 AMG 前処理

前処理計算の並列性と収束性は線型ソルバの性能を決める 2 大因子である．一般に並列度の高い前処理子は収束性が落ちるというトレードオフの関係にある．しかし，マルチグリッド前処理は高い並列性と収束を持った前処理であり，高い並列性が求められる GPU 計算と相性がよい．本研究では幾何格子構造の情報を利用せずに行列から代数的に粗行列を生成することができる代数的マルチグリッド (Algebraic MultiGrid) 前処理を用いる．本研究では CUDA による AMG の実装に CUSP ライブラリ [9] の smoothed_aggregation コードを用いた．

AMG 前処理は強力な反面，メモリを大量に使うという欠点がある．これを前処理子を float で確保することと，前処理子を 1 ノードに搭載されてる 2 枚の GPU カードにそれぞれ分け，前処理された配列を P2P 通信する．1 ノードに 1MPI プロセスを置くことで MPI 並列数を最小に下げることができる．

4.1.2 通信隠蔽

OpenFOAM では領域分割による MPI 並列化をサポートしている．MPI と GPU を組み合わせたハイブリッド並列を行うためには，MPI 通信したいデータを GPU のデバイスメモリから CPU のホストメモリへ PCIe バス経由でメモリ転送，MPI 通信をおこなってか通信先ノードのホストメモリからデバイスメモリへメモリ転送を行う必要がある．通信したい境界値のデバイスホスト通信，MPI による通信，ホストデバイス通信と，境界値ではない領域内部の値の SpMV 計算を CUDA Stream を用いることで同時に実行，通信時間を計算時間で隠蔽する．

4.2 計算条件

計算条件を設定する．計算対象は人間の胸部大動脈 (Thoracic Aorta) を MRI スキャンすることによって得られた血管構造を ANSYS 社の Gambit によって格子化し，OpenFOAM 用格子に変換した．格子化の細分度を調整することで，SMALL, MEDIUM, LARGE の計算格子を生成した．図 3 が格子化した大動脈メッシュの SMALL である．この図において左の矢印で示した部分が流入口，その他 4 つの出口は流出口であり，指定の境界条件を与える．

計算格子の大きさ，計算条件の詳細についてはそれぞれ表 2，表 3 にまとめている．

並列化は OpenFOAM 付属の Scotch 自動分割ライブラリ [10] によってサイズがほぼ等しくなるように領域分割することにより行う．例として，図 4 が領域を 4 分割して色分けしたものである．

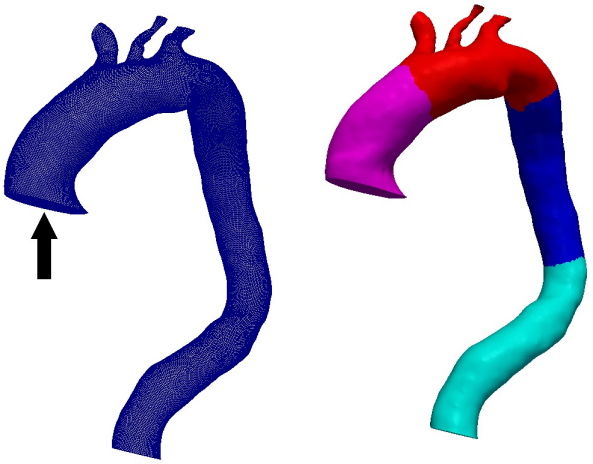

図 3 胸部大動脈の格子

図 4 領域分割の例

表 2 計算格子
Table 2 Meshes

格子名	SMALL	MEDIUM	LARGE
セル数	1,912,272	2,980,302	5,144,730
セル界面数	3,874,336	6,031,206	10,382,979
サイズ [MB]	155	311	543

表 3 計算条件
Table 3 Simulation parameters

ソルバ	simpleFoam (OpenFOAM-2.1.1)
物性値	動粘性係数 $\nu = 3.33 \times 10^{-6}$ [Pa.s](血液)
乱流モデル	層流モデル
流入側境界条件	流入速度 $V = 0.263$ [m/s] (Re = 3000)
流出側境界条件	圧力 $P = 0$ [Pa]
外部収束緩和係数	圧力・速度ともに 0.6
外部反復収束条件	$\|\delta P\|_1 \leq 1.0 \times 10^{-6}$ and $\|\delta V\|_1 \leq 1.0 \times 10^{-6}$
線形ソルバ	圧力 GPU-AMG-CG 法，速度 ILU-BiCG 法
内部反復収束条件	相対残差ノルム $\|\mathbf{r}\|_1 \leq 1.0 \times 10^{-8}$

4.3 結果

GPU CCI において 1 ノードにおける格子サイズのスケーリング (図 5) と，LARGE メッシュに対するノード並列度に対するスケーリング (図 6, 7) のベンチマークを行った．参考として，1 ノード内の CPU による DICCG 法によるスケーリンググラフを載せている．格子サイズのスケーリングでは GPU CCI と Inhouse Cluster は CPU 計算では GPU CCI の方がやや高速，GPU 計算ではほぼ同じという結果となった．GPU CCI の CPU 計算が Inhouse Cluster よりも高速なのは CPU のアーキテクチャが異なり，高い周波数を持つためであると考えられる．GPU 計算では共に CPU よりも 4 倍から 6 倍の性能向上が行えた．

MPI の非常に大きなレイテンシにも関わらず，ノードに対するストロングスケーリングでは 8 並列で CPU の内部反復では 9 倍，外部反復では 8 倍と，EC2 上のソルバも高い並列度を保ってスケールすることができた．

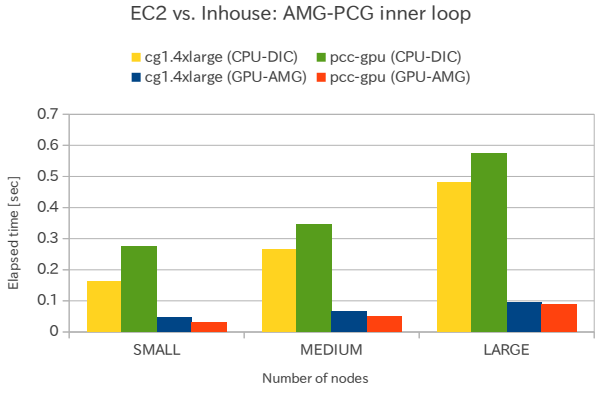

図 5 格子サイズに対する CCI と Inhouse Cluster の，CPU による DIC-CG と GPU によるの AMG-CG 内部反復の比較

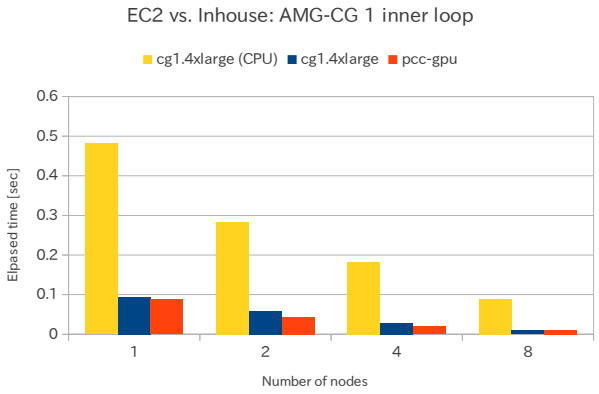

図 6 並列度に対する GPU CCI と Inhouse Cluster の AMG-CG 内部反復の比較: LAREG メッシュ

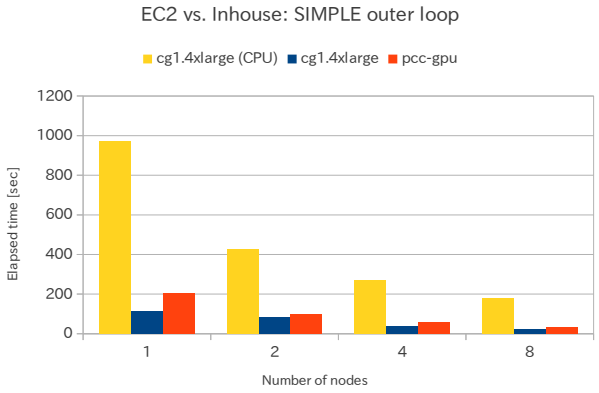

図 7 並列度に対する GPU CCI と Inhouse Cluster の SIMPLE 外部反復の比較: LAREG メッシュ

5. 関連研究

EC2 クラウド HPC の性能測定を行った研究はいくつかある．Zhai ら [11] は EC2 で IMB と NPB を実行を行い，コストも含めた性能測定を行なっている．

6. 結論

本研究では Amazon EC2 の GPU CCI インスタンスを用いて IMB の性能測定を行い，通信をなるべく行わない GPU-AMG-CG ソルバを開発して，血流流体の解析に適応して EC2 上で計算を行った．その結果，EC2 上においてもスケールすることができた，

参考文献

[1] Malecha Ziemowit M, Miroslaw Lukasz, Tomczak Tadeusz, Koza Zbigniew, Matyka Maciej, Tarnawski Wojciech, Szczerba Dominik. “GPU-based simulation of 3D blood flow in abdominal aorta using OpenFoam”. Archives of Mechanics, 2011, vol. 63, No 2, pp. 137-161

[2] R.Li, Y.Saad. “GPU-accelerated preconditioned iterative linear solvers,” Report umsi-2010-112, Minnesota Supercomputer Institute, University of Minnesota, Minneapo-

lis, MN, 2010.
[3] The SIMPLE algorithm in OpenFOAM - OpenFOAMWiki, 入手先 ⟨http://openfoamwiki.net/index.php/The_SIMPLE_algorithm_in_OpenFOAM⟩
[4] J.H.Ferziger, M.Peric. “Computational Methods for Fluid Dynamics.” Springer-Verlag Berling, Heidelberg, 1996.
[5] Y.Saad. “Iterative Methods for Sparse Linear Systems”. PWS Publishing Co.,Massachusetts, MA, 2000.
[6] Amazon: EC2 Instance Type (online): 入手先 ⟨https://aws.amazon.com/ec2/instance-types/⟩
[7] Star: Cluster 入手先 ⟨http://web.mit.edu/star/cluster/⟩
[8] Alexey Petrov, Andrey Simurzin Cloud Flu. 入手先 ⟨http://sourceforge.net/apps/mediawiki/cloudflu/index.php?title=Main_Page⟩
[9] Nathan Bell and Michael Garland. “Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations”, 2012. 入手先 ⟨http://cusp-library.googlecode.com⟩
[10] SCOTCH: A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs. “Proceedings of HPCN’96”, Brussels, Belgium. LNCS 1067, pages 493-498. Springer, April 1996. F. Pellegrini and J. Roman. 入手先 ⟨www.labri.fr/perso/pelegrin/scotch/⟩
[11] Yan Zhai, Mingliang Liu, Jidong Zhai, Xiaosong Ma, and Wenguang Chen. “Cloud versus in-house cluster: evaluating Amazon cluster compute instances for running MPI applications”. In State of the Practice Reports (SC ’11). ACM, New York, NY, USA, Article 11,10 pages. 2011