In Situ/In Transit アプローチを用いた大規模数値解析における ポスト処理効率化

In Situ/In Transit Approaches for Post-Processing in Large-Scale Numerical Simulation

○ 堤誠司, JAXA, 相模原市中央区由野台 3-1-1, E-mail: tsutsumi.seiji@jaxa.jp
藤田直行, JAXA, 調布市深大寺東町 7-44-1
伊藤浩之, 株式会社菱友システムズ, 名古屋市港区大江町 6-19　菱興ビル南館
大日向大地, 富士通株式会社, 長野市鶴賀緑町 1415 大通りセンタービル
井上敬介, みずほ情報総研株式会社, 千代田区神田錦町二丁目 3 番
松村洋祐, みずほ情報総研株式会社, 千代田区神田錦町二丁目 3 番
高橋慧智, 奈良先端科学技術大学院大学, 生駒市高山町 8916 番地 5
Greg Eisenhauer, Georgia Institute of Technology, North Ave NW, Atlanta, GA, USA
Norbert Podhorszki, Oak Ridge National Laboratory, 1 Bethel Valley Rd, Oak Ridge, TN, USA
Scott Klasky, Oak Ridge National Laboratory, 1 Bethel Valley Rd, Oak Ridge, TN, USA

To overcome the difficulty in visualizing large-scale simulation results in HPC, we implemented in situ and in transit visualization using VisIt/Libsim and ADIOS2 into an in-house application for computational fluid dynamics (CFD), and evaluated both forms on a heterogeneous HPC system of the Japan Aerospace Exploration Agency (JAXA). It is found that both frameworks can be used for batch and interactive visualization commonly conducted by researchers and engineers. The in situ approach costs less to implement in applications and is easier to use. Conversely, the in transit approach is advantageous in terms of lower execution overhead and its flexibility for using separate resources in a heterogeneous HPC system. It is therefore desirable to use both approaches as needed. Based on the knowledge obtained herein, this paper summarizes suggestions for research and development of the in situ and in transit visualization frameworks, the HPC system, and queue policy for interactive jobs, in order to realize in situ and in transit visualization in a heterogeneous HPC system.

1．はじめに

スーパーコンピュータの演算性能増加に伴い，数値シミュレーションの解析規模は増大する一方である．数値流体解析(CFD)においては，CFD プログラム自身は CFD・計算科学の研究者を中心に精力的に研究され，超並列スーパーコンピュータへの移植が継続的に行われている．近年のスーパーコンピュータは演算性能が増加する一方で，メモリ帯域，ディスク帯域はそれほど増加せず，演算性能との差が年々広がる一方である．例えば，Oak Ridge National Laboratory (ORNL)の Titan スーパーコンピュータでは，演算性能と IO 性能の差が 5 桁にもなった[1]．その結果，CFD 解析のワークフロー全体を見ると，I/O がホットスポットとなる可視化といったポスト処理に関する作業コストが増大し，Capability Computing だけではなく Capacity Computing ですら困難になりつつある．本研究ではディスク領域への I/O をすることなく，主記憶上にある計算中のデータを直接可視化する tightly-coupled in situ 可視化(以下, in situ 可視化), 及び in transit 可視化の利用を試行する．in situ 可視化では，主記憶上にある計算プロセスのデータを可視化プロセスと共有することから，データコピーが発生しない(zero-copy)ことが特徴である．一方，計算プロセスと可視化プロセスが CPU やメモリ帯域を共有するため，計算性能に影響があることが懸念される．また，in situ 可視化で用いるライブラリが異常終了した場合，計算プロセスも連動して異常終了してしまう．in transit 可視化では，計算プロセスと可視化プロセスを分離し，計算プロセスの主記憶上にあるデータを可視化プロセスの主記憶へ Remote Direct Memory Access (RDMA)等で転送する．そのため，柔軟に計算リソースの配分が可能となり，ヘテロジーニアスな HPC のリソースを有効活用できる．また，計算性能に与える影響が最小限に抑えることが可能で，可視化プロセスの異常終了に計算プロセスが影響を受けることはないといった堅牢性という観点でもメリットがある．ただし，計算プロセスの主記憶から可視化プロセスの主記憶へデータ転送をするフレームワークを CFD プログラムへ組み込まねばならず，また HPC での利用方法が従来よりも複雑になる．更に，データ転送に関するオーバーヘッドが明確ではなく，ヘテロジーニアスな HPC での利用方法など，実績は必ずしも豊富にあるとは言えない．そこで，CFD の研究者やエンジニアが行う典型的な可視化作業に in situ/in transit アプローチを適用し，これらに関する知見を取得することを目的に，JAXA のインハウス CFD プログラムである upacs-LES に in situ/in transit 可視化を実装した．そして, 典型的な HPC システムである JAXA Supercomputer System Generation 2(JSS2)[2]において試行した．本研究では，これらの試行結果について報告する．

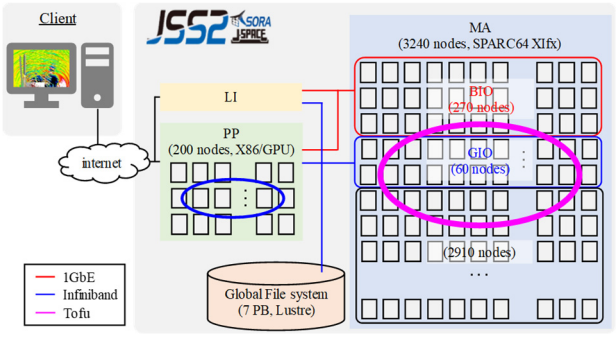

Fig. 1 Overview of JSS2 system.

2．HPC システム，CFD プログラムの概要

本研究では，ヘテロジーニアスな HPC の典型例である，2015 年から本格運用が始まった JSS2 を用いた．本研究に関連する JSS2

の構成を Fig. 1 に示す．3.49 PFlops の演算性能を持つメインシステム(SORA-MA, 以下, MA と呼ぶ)は, 3240 ノードの Fujitsu FX100 (SPARC64 XIfx)から構成される．大規模な計算リソースが必要となる数値計算は MA で実行する．JSS2 はログインノード(SORA-LI, 以下では LI と呼ぶ)の他，プリポスト処理専用の PC クラスタ(SORA-PP, 以下, PP と呼ぶ)がある．PP は 1 ノード辺り Intel Xeon E5-2643v2 が 2 個，NVIDIA Quadro K2000 が 1 枚を搭載し，全 200 ノードから構成される．JSS2 は管理ネットワークの 1GbE，PP や Global File System，MA を接続する infiniband, MA 内の Torus Fusion (Tofu) interconnect の 3 つのヘテロジーニアスなネットワークから構成される．それぞれ独立したネットワークであり，相互に通信できるようにはなっていない．PP や LI は，MA の計算ノードのうち IO ノード(うち, 1GbE で結合された 270 ノードを BIO, Infiniband で結合された 60 ノードを GIO と呼ぶ)で物理的に結合されている．遠隔地にあるユーザ端末からは, インターネットを介して LI と PP にアクセスが可能である．

本研究では JAXA のインハウス CFD プログラムである upacs-LES [4, 5, 6, 7]をベースに, 多成分系に拡張した upacs-mc-LES を用いた．upacs-mc-LES は有限体積法をベースとし，マルチブロック構造格子を利用する．圧縮性 3 次元 Navier-Stokes 方程式を支配方程式とし，Large-Eddy Simulation (LES)が可能である．

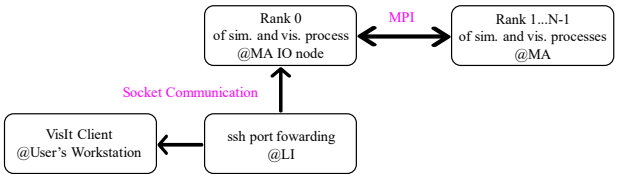

Fig. 2　Implementation of VisIt/Libsim for interactive visualization in JSS2.

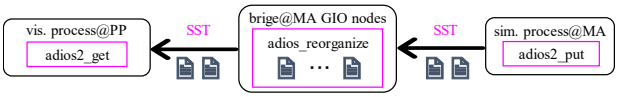

Fig. 3　Implementation of ADIOS2 for interactive visualization in JSS2.

3．In Situ/In Transit 可視化への要求

CFD の研究者やエンジニアが行う典型的な可視化作業において必須となる機能を，以下にまとめる．in situ/in transit 可視化ライブラリの選択，CFD プログラムへの実装，更には HPC での利用にあたって留意しなければならいポイントである．

1. CFD のポスト処理として行われる可視化は，対象とする流れ場の物理現象を理解するために行うインタラクティブ可視化と，予め指定された断面，物理量で画像や動画等を作成するバッチ可視化に分けられる．インタラクティブ可視化，バッチ可視化の両方が可能でなければならない．
2. インタラクティブ可視化は可視化アプリを用いて対話的に作業を行うことから，計算プロセスの実行中，常に動いている必要はない．必要な時に可視化アプリを起動して計算プロセスに接続(Attach)し，また作業が終われば切断する(Detach)機能が必須である．

4．In Situ/In Transit 機能の実装

Bauer らがまとめているように，in situ 可視化に関しては数多くの研究が行われており，利用可能なライブラリは多い[1]．本研究では SENSEI[7]のバックエンドライブラリの 1 つでもある VisIt/Libsim[8]を選択した．VisIt は大規模可視化が可能な GUI を持ち，自由に Attach/Detach が可能で，また数多くのアプリケーションにおいて利用実績があるためである[9]．in transit 可視化に関してはこの 10 年間で GLEAN, ADIOS が開発され[10-12]，また近年では ADIOS2[13], libIS[14]など，新しいライブラリの研究開発が行われている．本研究では，ADIOS2 を使って in transit 通信機能を実装した．ADIOS2 は writer と reader 間で自由に Attach/Detach が可能な通信エンジンを持っており，さらに JSS2 のような示すヘテロジーニアスな HPC に実装する際，異なる通信エンジンをステージング可能な機能を有するからである．VisIt/Libsim，ADIOS2 の実装はマニュアル通りに実装した．詳細は[7,13]を参照されたい．一方，JSS2 のようなヘテロジーニアスな HPC で in situ アプローチと in transit アプローチを使ったバッチ可視化，インタラクティブ可視化を行う場合，それぞれで利用方法は大きく異なる．バッチ可視化の場合，in situ アプローチであれば計算プロセスと可視化プロセスを同一ノード内に連動して実行されることから，VisIt/Libsim を導入した CFD プログラムを従来通りジョブスケジューラを使ってジョブ投入すればよい．in transit アプローチであっても，計算プロセスと可視化プロセスをいずれも MA 内で動かすことを想定すると，両プロセスを同時に 1 つのジョブスクリプトでジョブ投入すればよいため，従来通りの利用が可能である．

インタラクティブ可視化はバッチ可視化より複雑である．VisIt/Libsim を使った in situ アプローチでインタラクティブ可視化を行う場合，ユーザ端末で起動する VisIt Client と，可視化プロセスの先頭プロセス(Rank0)の間でソケット通信を行う必要がある．そのため，Fig. 2 に示すように，可視化プロセスの先頭プロセス(Rank0)を MA の IO ノードに割り振り，LI ノードを介してポートフォワードし，VisIt Client と先頭プロセス(Rank0)でソケット通信を確立する．一方，in transit アプローチでインタラクティブ可視化を行う場合, PP が持つ GPU 資源を有効利用することが望ましい．MA と PP は異なるネットワークで構成され，MA から PP へと直接データ転送することはできない．PP と MA が直結されているのは IO ノードであることから，Fig. 3 に示すように，Tofu 内の通信と MA から PP への Infiniband を使った通信を橋渡しするブリッジプロセス(adios_reorganize)を, Infiniband で物理的に結合された GIO ノードに立ち上げる．そして，MA 内で実行する計算プロセスから GIO ノードのブリッジプロセスに一旦データを転送し，次に，ブリッジプロセスから PP 上の可視化プロセスにデータを転送する．計算プロセスからブリッジプロセス，またブリッジプロセスから可視化プロセスへはそれぞれ通信エンジンとして ADIOS2 に実装されている Sustainable Staging Transport (SST)を利用した．PP はユーザ端末からインターネットを介してアクセスできることから，PP 上の可視化プロセスとユーザ端末はソケット通信が可能であり, VisIt Client を使ったインタラクティブ可視化が可能となる．SST の Reader プロセスと Writer プロセスは，それぞれが同時に起動しておく必要がなく，MA に投入する計算プロセス，ブリッジプロセス，PP に投入する可視化プロセスの起動順序は気にしなくてもよい．また，SST は Attach/Detach が可能な通信エンジンであることから，必要な時に可視化プロセスを起動して，計算中の最新の結果を可視化可能である．ただし，計算プロセスから可視化プロセスに転送されるデータは，一旦，ブリッジプロセスでバッファされることから，adios_reorganize におけるメモリ使用量に注意が必要である．今回，JSS2 において SST を利用する場合，技術上の問題から RDMA 通信はできず，IPoIB (IP over InfiniBand)を利用せざるを得なかった．

5．結果と考察

H3 ロケット打上げにエンジン排気ジェットから発生する空力

騒音の解析を対象に, in situ/in transit 可視化を JSS2 にて試行した. 計算メッシュ数は 2.3 億セルで，可視化用に転送するボリュームデータの物理量は 14 個あり, 転送データは 1 ステップあたり合計で 30 GB である．計算プロセスは 102 ノード(1632 プロセス)を利用した．なお，1 ステップ辺りの計算時間は約 10 秒である．

更に，バッチ可視化のみに関しては，より大規模なロケットエンジン排気ジェットの空力騒音解析に関しても試行した．計算セル数は約 61 億セル，可視化用に転送するボリュームデータの物理量は 14 個あり，転送データは 1 ステップあたり 526 GB である．計算プロセスは 400 ノード(1600 プロセス)を利用した. 1 ステップ辺りの計算時間は約 55 秒である. 次期の JAXA スパコンで通常実行する解析の規模が数十億セルであると考えられることから，この規模に対する in situ/in transit 可視化性能を調べることとした.

５．１　バッチ可視化

in situ アプローチによるバッチ可視化は, 計算プロセスと可視化プロセスが同一ノードに割り振られることから，資源の競合が起き，性能低下が懸念される．一方，in transit アプローチでは，計算プロセスと可視化プロセスを別々の計算資源に割り振るため計算資源を競合する心配がないが，データ転送に関するオーバーヘッドがかかる．そこで，2.3 億メッシュ数の H3 ロケット空力騒音解析において，CFD の 10 タイムステップごとに約 14M ポリゴンの可視化画像を生成し，計算の実行や可視化の実行に要した性能を比較した．in situ アプローチの場合，1632 に分割された計算・可視化プロセスを，MA 102 ノードで実行した．一方，in transit アプローチの場合，102 ノードに割り付けた 1632 プロセスの計算プロセスとは別に，データを受信し Libsim で可視化する可視化プロセスを MA 内に同時起動した．可視化プロセスは 1 ノード辺り 1 プロセスとし，2，4，8，16，32 プロセスの全 5 ケースで実施した．in situ アプローチで要した全実行時間に対して in transit アプローチの計算プロセスが要した実行時間の割り合いを，Fig. 4 に示す．in situ アプローチでは, 総計算時間の 1 割が可視化に要していることが分かる．次に，in transit アプローチの結果を比較すると，可視化プロセスのプロセス数を変化させても CFD 計算に要する時間は当然ながら変化がないが，データ転送に要する時間が大きく異なる結果となった．可視化プロセスが 2 プロセスの場合，in situ アプローチよりも 2.1 倍も計算時間を要しており，そのうち約 6 割が ADIOS2 によるデータ転送が占めている．プロセス数が 4 になると大きく減少し，8 プロセス以上ではほぼ無視できる程度の計算時間となり，解析時間とデータ転送時間を併せて in situ アプローチよりも 9 割程度と高速化した．今回，実際の利用シーンを想定し，可視化プロセスがデータ受信を完了するまでは送信側(計算プロセス)はデータ転送を一旦停止する(ブロックする)するという設定とした．可視化プロセス数が小さい 2，4 プロセスの場合，ネットワーク帯域が狭く受信時間が長くなったので，可視化プロセスがデータを受信しレンダリングが完了するまで，計算プロセスがブロックされたことを示している．その間は計算プロセスがブロックされてしまい，結果的に Fig. 4 のようにデータ転送に係るコストが増大したと考えられる. in transit アプローチで可視化プロセスが計算プロセスよりも遅い場合は，データ転送をブロックせずに，データを捨てるように設定した方が性能は伸びるが[15]，データ転送に関する設定は利用シーンに依存する．次に，in situ アプローチの計算時間に対する可視化プロセスの処理時間の割り合いを Fig. 5 に示す．上述の通り，可視化プロセス数が 2，及び 4 の場合は，in situ 可視化よりもそれぞれ 2.5 倍，1.4 倍と計算時間が増加した．8 プロセス以上になると，in situ の場合と比べて 9 割程度まで実行時間が減少した. in transit アプローチの可視化とデータ受信に係る割り合いを比較すると，8 プロセスでデータ受信に要する時間は最小になり，その後，16，32 とプロセス数を増やすと逆に増加している．一方，可視化処理に関してはプロセス数を増やすほど減少している．並列度を上げることによってデータ受信に要する時間が増加した理由は定かではなく，今後，詳細に調べる必要がある.

次に，より大規模な 61 億セルの空力騒音解析について議論する. いずれも CFD の 10 ステップごとに約 11M ポリゴンの可視化画像を生成し，in transit アプローチでは 78 ノード(78 プロセス)で実行した．Figures 4, 5 と同様に，in situ アプローチに要した実行時間に対する割合を Fig. 6 に示す．in situ アプローチの場合，実行時間の約 1 割が可視化画像の生成に要している．一方，in transit アプローチでは可視化プロセス側のプロセス数が十分大きいため，計算プロセス全体の中でデータ転送にかかる時間が無視できるほど小さく，ほぼ CFD プログラムの計算時間のみである．in transit アプローチの可視化プロセスを見ると，実行時間全体の中で約 1 割が可視化に要しており，残りはデータの受信である．計算プロセスの実行時間より可視化プロセスの実行時間が短いことから，データはブロックされることなく転送されていたことが分かる．以上より，数十億セル規模の計算結果に対しても，ネットワーク帯域を十分確保できれば，計算時間を増やすことなくバッチ可視化が可能であることが分かった．また，バッチ可視化の場合は，in situ アプローチであっても in transit アプローチであっても, いったん CFD プログラムに組み込んでしまえば，従来と同様にジョブ投入すればよいため使い勝手は良いことが分かった.

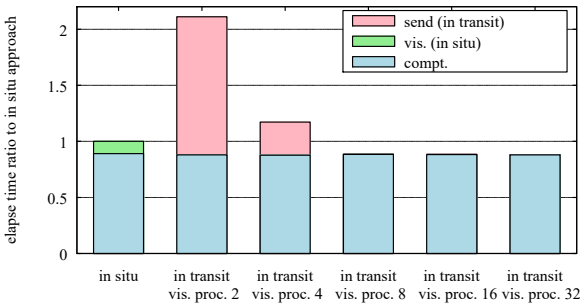

Fig. 4　Comparison of elapse time between the in situ approach and simulation processes of the in transit approach for 230M cells simulation.

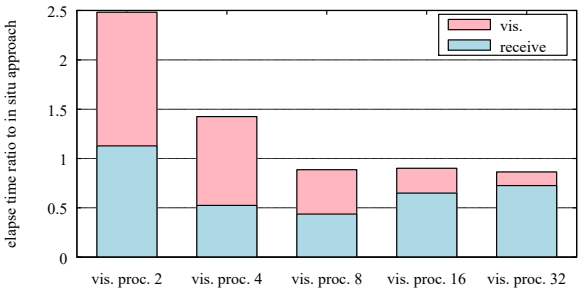

Fig. 5　Comparison of elapse time between the in situ approach and visualization processes of the in transit approach for 230M cells simulation.

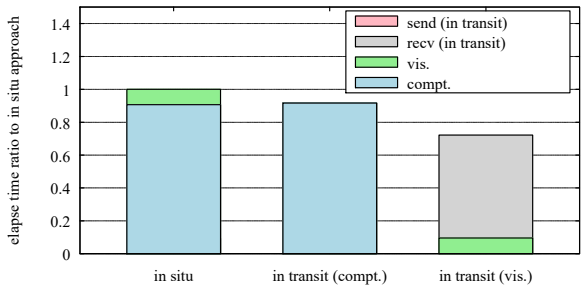

Fig. 6 Comparison of elapse time between the in situ and in transit approaches for 6.1B cells simulation.

5．2 インタラクティブ可視化

in transit アプローチでインタラクティブ可視化を行う場合，MA の GIO ノードにブリッジプロセスを割り付ける必要がある．JSS2 のジョブスケジューラは，現状，ノードを指定してジョブを割り付けることができない．61 億セルの計算では，ブリッジプロセスに要求されるメモリ量が非常に大きく，十分な GIO ノード数を現状の MA で確保することはほぼ不可能である．そこで，2.3 億セルの計算に対してのみ，インタラクティブ可視化を試行した．

上述したように，2.3 億セルの空力騒音計算では 1632 個に分割された計算プロセスを 102 ノードの MA に割り付けた．in transit アプローチで必要となるブリッジプロセスは，CFD プログラムから転送されたデータを一旦貯めるバッファとなる. adios_reorganize を使う場合，転送データ量の約 6 倍のメモリが使用されることが分かった．通信データ量が 30GB であり，かつ MA は 1 ノード辺り 32GB であることを考慮して，本研究では 8 ノードの GIO に 128 プロセスのブリッジ(adios_reorganize)を割当てた．VisIt/Libsim と ADIOS2 の reader から構成される可視化プロセスに関しては，ADIOS2 は受信側にバッファを持たないことから，メモリ使用量に対するインパクトは VisIt/Libsim のみを考慮すればよい. そこで，過去の経験をもとに，可視化プロセスは 2 ノード(16 プロセス)とした．MA に計算プロセスとブリッジプロセスを投入し，実行し始めた後，可視化プロセスを起動してブリッジプロセスから 1 つ目のデータを受診するまで，約 5 分を要した．その間，ユーザ端末から VisIt Client を起動して可視化プロセスの VisIt Sever に接続を行った．上記の in transit アプローチで，インタラクティブに可視化した H3 ロケットの空力騒音解析結果を Fig. 7 に示す．一旦，ブリッジプロセスとの間で接続が確立できれば，VisIt Client からブリッジプロセスのキューにある最新のデータを転送し，そして可視化結果を表示するまで約 1 分であった．ここで，JSS2 における計算，ブリッジ，可視化の 3 つのプロセスの位置関係とそれらが実行される各サブシステム間の転送速度の総ピーク値を Fig. 8 に示す．GIO を 8 ノード確保したことにより，計算プロセスとブリッジプロセス間は Tofu で 100 GB/s, ブリッジプロセス(GIO)と infiniband のスイッチ(IB-SW)間は 54.4 GB/s, そして IB-SW と PP 間は 13.6 GB/s である．一方，GIO では合計 30 GB のデータを 128 プロセスに分割し，プロセス辺りのファイル長は概ね 230 MB である．そこで，各サブシステム間を単一の接続で 256 MB のデータ転送速度を実測した結果を Table 1 に示す．RDMA の方が 10 倍以上高速であるが，今回は技術上の問題から IP 通信しかできなかった．そこで，Table 1 下段に示した IP 通信の実測値を用いて要求実行速度を求め，データ転送の総ピーク値と比較した結果を Table 2 に示す．この結果，通信のボトルネックは PP の要求実行速度(4 GB/s)であることが分かる．従って，PP のプロセス数を 16 より更に増やすことで，データ通信性能の向上が見込める．以上のように，ブリッジプロセス，可視化プロセスに割当てる資源量やプロセス数は，メモリ使用量だけではなく，データ通信性能(データ転送の総ピーク値と要求実行速度)を考慮して決めなければならない. 最後に, 今回は通信エンジンに SST を利用していることから，可視化プロセスはブリッジプロセスに対して何度も Attach/Detach が可能であることも確認できた．

次に，in situ によるインタラクティブ可視化を実施した．前述の通り, VisIt/Libsim を利用してインタラクティブ可視化を行う場合，ユーザ端末で起動する VisIt Client と VisIt サーバの先頭ランクとがソケット通信する必要であることから，少なくとも 1 プロセスを IO ノードに割り付けた．計算・可視化プロセスの実行を確認したのち，VisIt Client をユーザ端末で起動し，IO ノードに割り付けられた可視化プロセス Rank 0 と接続が確立するまで，約 40 秒を要した．ADIOS2 を使った in transit 可視化と比べると，接続確立までに要する時間は早かった．一旦接続が確立すれば，VisIt Client から最新のタイムステップの計算結果を要求して結果が表示されるまで，同じく 40 秒程度を要し，こちらについては in transit 可視化とほぼ同程度であった．

in transit 可視化のブリッジプロセスに, 十分なメモリ容量と通信性能を割り当てることが必要であるが，JSS2 では IO ノード数が限られ，さらに特定ノードにジョブを割り振ることが JSS2 では現状できないことから，IO ノードの確保は大きな課題であった．一方, 可視化プロセスを PP に割り付けることでハードウェアレンダリングが可能となるため，インタラクティブ可視化を行う上では大きなメリットである．更に，可視化アプリが異常終了しても，計算プロセスに影響を及ぼさない安心感は大きい．使い勝手の上では in situ の方が平易だが，Libsim/VisIt では，可視化プロセスと計算プロセスは同期しており，インタラクティブ可視化している間は計算プロセスが止まってしまう．in situ でも，計算プロセスと可視化プロセスとは非同期である必要がある．

5．3 in situ/in transit 可視化の HPC 実装に関する検討事項

in transit アプローチでインタラクティブ可視化を行う場合, JSS2 ではブリッジプロセスに IO ノードを割当てる必要があるが, 上述の通り，ジョブスケジューラの問題や IO ノード数の制約があり，多くの IO ノード数を必要とするような規模の可視化は, 現状では困難である．例えば，IO ノードをルータに仕立て，IP パケットをルーティングすれば GbE，InfiniBand，Tofu 相互に End to End で通信できるようになることから，IO ノードにブリッジプロセスを起動させることは不要になる．このような仕組みは，HPC のシステム設計時から考慮すべきである．そのため，HPC にて in situ/in transit 可視化を利用するためには, 予めこれらの技術を利用することを想定したシステム設計が求められる. in transit アプローチでインタラクティブ可視化を行う場合，起動するジョブは対話ジョブであることから，必要な際にすぐに資源が確保できること，経過時間を気にせず利用できることなど，従来のバッチジョブとは異なる利用体系を考える必要がある．

今回，in situ 可視化ライブラリとして採用した VisIt/Libsim は，インタラクティブ可視化，及びバッチ可視化を概ね実現することが可能であった．ただし，VisIt/Libsim は計算プロセスと可視化プロセスが同期しており，インタラクティブ可視化中は計算プロセスが停止してしまった．実利用を考えた場合，計算プロセスと可視化プロセスは非同期である必要がある．一方，in transit によるインタラクティブ可視化のブリッジプロセスとして利用した adios_reorganize は，通信データ容量の約 6 倍ものメモリ量がバッファとして必要であった．上述の通り，IO ノード数は限られることから，ブリッジプロセスにおける省メモリ化は必須である．また，今回 CFD プログラムとして利用した upacs-mc-LES のデータ構造に ADIOS2 は完全に対応しておらず，利用できる機能に制約があった．アプリケーションプログラムと in situ/in transit ライブラ

リとの間で co-design の重要性が重要である.

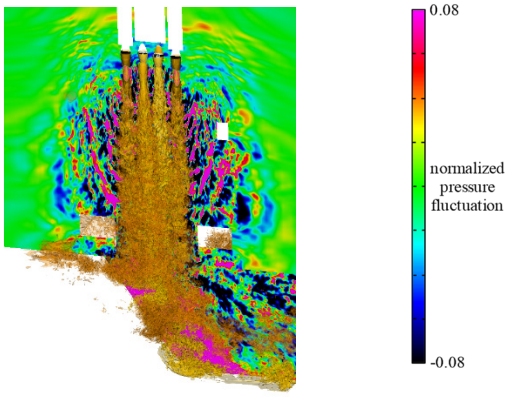

Fig. 7 Computational result obtained by interactive visualization with in transit approach. Acoustic wave on cross-section and exhaust jet are visualized by normalized pressure fluctuation and iso-surface of temperature, respectively.

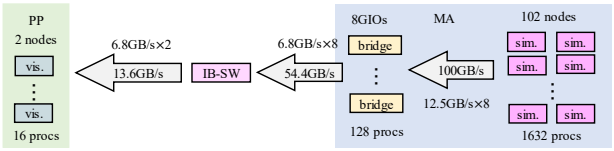

Fig.8 Total peak transfer rate in the present implementation..

Table 1 Effective transfer rate per single connection of 256 MB file in JSS2.

	Sim. to GIOs	GIOs to IB-SW	IB-SW to PPs
RDMA	12.5 GB/s (ideal)	2.8 GB/s	3 GB/s
IP	1 GB/s	0.25 GB/s	0.25 GB/s

Table 2 Comparison of total transfer rate, and required effective transfer rate enclosed in brackets.

	Sim. to GIOs	GIOs to IB-SW	IB-SW to PPs
RDMA	100 GB/s (ideal)	54.4 GB/s (2.8 GB/s×128 =358 GB/s)	13.6 GB/s (3 GB/s×16 =48 GB/s)
IP	100 GB/s (1 GB×128 =128 GB/s)	32.0 GB/s (0.25 GB/s×128 =32 GB/s)	13.6 GB/s (0.25 GB/s×16 =4 GB/s)

6. まとめ

本研究では，VisIt/Libsim を使った in situ 可視化と，ADIOS2 を使った in transit 可視化をインハウスの CFD プログラムに実装し，ヘテロジーニアス HPC として典型的な JAXA スーパーコンピュータシステム(JSS2)において試行した．そして，CFD のポスト処理として行うインタラクティブ可視化やバッチ可視化を概ね実現可能であることが分かった．バッチ可視化では，in situ，in transit いずれのアプローチでも，HPC での利用方法は従来と変わらず，いったん CFD プログラムに組み込んでしまえば，平易に使えることが分かった．実行性能を比較すると，可視化プロセスのノード数を十分確保すれば，in transit アプローチでは通信のオーバーヘッド等の影響がほぼ無視でき，in situ よりも高速に実行することが可能であった．インタラクティブ可視化を in transit アプローチで実施する場合，ブリッジプロセスには十分なメモリ容量と通信性能を確保するため多数の IO ノードを割り当てる必要があるが，JSS2 では IO ノード数が限られる上，ジョブスケジューラが特定のノードを指定してジョブ投入できなかった．そのため，より大規模な計算結果へ適用は，現状の JSS2 システムでは困難であった．一方，in transit アプローチでは GPU を持つシステムに可視化プロセスを割当てることができるなど，ヘテロジーニアスな HPC システムの資源を柔軟に利用できることがメリットである．更に，可視化プロセスが仮に異常停止した際にも，計算プロセスは独立して動き続けるという安心感がある．in situ アプローチも in transit アプローチもそれぞれに一長一短があり，計算規模や利用シーンに応じて，自由に使い分けることができることが望ましい．最後に，本研究を通して得られた知見をもとに，in situ/in transit 可視化をヘテロジーニアスな HPC システムで利用していくために考慮すべき検討事項を整理した.

本研究はポスト処理の中でも可視化に着目して議論を行ったが，in transit アプローチは可視化のみならず，ポスト処理全般にも利用可能である．機械学習を利用したようなより複雑で計算コストの高いポスト処理作業を，計算プロセスに与える影響を最小限に抑えつつ，またデータをすべてディスク領域に書き出すことなく実施できるため，より大規模な数値解析においては有効な手法である.

参考文献

(1) A.C. Bauer, H. Abbasi, J. Ahrens, H. Childs, B. Geveci, S. Klasky, K. Moreland, P. O'Leary, V. Vishwanath, B. Whitlock and E.W. Bethel, “In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms, Computer Graphics Forum,” 35 (3), 2016, pp.577–597. DOI: http://dx.doi.org/10.1111/cgf.12930

(2) JAXA (Japan Aerospace Exploration Agency). JAXA Supercomputer System Generation2 (JSS2), https://www.jss.jaxa.jp/en/.

(3) M. Murayama, K. Yamamoto, and K. Kobayashi, “Validation of Computations Around High-Lift Configurations by Structured- and Unstructured-Mesh,” *Journal of Aircraft*, 43(9), 2006, pp.393-406. DOI: http://dx.doi.org/10.2514/1.15445.

(4) T. Imamura, S. Enomoto, Y. Yokokawa, and K. Yamamoto, “Three-Dimensional Unsteady Flow Computations Around a Conventional Slat of High-Lift Devices,” *AIAA Journal*, 46(5), 2008, pp.1045-1053. DOI: http://dx.doi.org/10.2514/1.25660.

(5) S. Tsutsumi, T. Ishii, K. Ui, S. Tokudome, and K. Wada, ”Study on Acoustic Prediction and Reduction of Epsilon Launch Vehicle at Liftoff,” *Journal of Spacecraft and Rockets*, 52(2), 2015, pp.350-361. DOI: http://dx.doi.org/10.2514/1.A33010.

(6) S. Tsutsumi, and K. Terashima, “Validation and Verification of Numerical Prediction Method for Lift-off Acoustics of Launch Vehicles,” *Transactions of the Japan Society for Aeronautical and Space Sciences, Aerospace Technology Japan*, 15 (2017), pp.7-12. DOI: http://dx.doi.org/10.2322/tastj.15.7.

(7) LLNL (Lawrence Livermore National Laboratory). VisIt. http://wci.llnl.gov/simulation/computer-codes/visit.

(8) U. Ayachit, B. Whitlock, M. Wolf, B. Loring, B. Geveci, D. Lonie, and E.W. Bethel, “The SENSEI Generic In Situ Interface,” *In proceedings of In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization (ISAV'16)*, (2016). DOI: https://doi.org/10.1109/ISAV.

(9) J. Nonaka, M. Matsuda, N. Sakamoto, M. Fujita, K. Onishi, E.C. Inacio, S. Ito, F. Shoji, and K. Ono, "A Study on Open Source Software for Large-Scale Data Visualization on SPARC64fx based HPC Systems," *In proceedings of the International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2018)*, 28-31 (2018).
DOI: http://dx.doi.org/10.1145/3149457.3155323.

(10) K. Moreland, R. Oldfield, P. Marion, S. Jourdain, N. Podhorszki, V. Vishwanath, N. Fabian, C. Docan, M. Parashar, M. Hereld, M.E. Papka, and S. Klasky, "Examples of in transit visualization," *In Proceedings of the 2nd International Workshop on Petascal Data Analytics: Challenges and Opportunities (PDAC '11)*, 1-6 (2011).
DOI: http://dx.doi.org/10.1145/2110205.2110207.

(11) V. Vishwanath, M. Hereld, V. Morozov, and M.E. Papka "Topology-aware data movement and staging for I/O acceleration on Blue Gene/P supercomputing systems," *In proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, (2011). DOI: http://dx.doi.org/10.1145/2063384.2063409.

(12) Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Choi, S. Klasky, R. Tchousa, J. Lofstead, R. Oldfield, et al., "Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks," *Concurrency and Computation: Practice and Experience*, 26(7), 2014, 1453-1473.
DOI: https://doi.org/10.1002/cpe.3125.

(13) ORNL (Oak Ridge National Laboratory). AIDOS2. https://adios2.readthedocs.io/en/latest/.

(14) W. Usher, S. Rizzi, I. Wald, J. Amstutz, J. Insley, V. Vishwanath, N. Ferrier, M.E. Papka, and V. Pascucci, "libIS: A Lightweight Library for Flexible In Transit Visualization," *In proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV'18)*, 33-38, (2018).
DOI: http://dx.doi.org/10.1145/3281464.3281466.

(15) J. Kress, M. Larsen, J. Choi, M. Kim, M. Wolf, N. Podhorszki, S. Klasky, H. Childs, and D. Pugmire, "Comparing the Efficiency of In Situ Visualization Paradigms at Scale," *High Performance Computing*, 99-117 (2019).
DOI: http://doi.org/10.1007/978-3-030-20656-7_6