Vol.2021-HPC-182 No.17
2021/12/7

超大規模並列計算時代における 通信機構「N-STDIO」のポスト処理利用

森江　善之[1,a)]　本田　宏明[2]　柴村　英智[3]　南里　豪志[4]

概要：ポスト処理を実行する際には異なるシステムを連結することから利用上のいくつかの問題が発生する．特にインターネットを介した解析計算機によるポスト処理を行うと現在の一般的な大規模並列計算機利用ではフロントエンドに大量のディスクアクセスが発生する．これらの問題を解決するために通信機構 N-STDIO を提案し，プロトタイプ実装を行った．
この通信機構 N-STDIO ではディスクアクセスを回避するため通信により直接データ転送を行う．また，接続に関してはサーバー/クライアントモデルを採用しているため，コネクションが必要ないプロセス間では接続処理を行わない．これにより少ない資源で通信路を作成できる．
そこで，N-STDIO のプロトタイプ実装を行い，これを用いた初期動作実験を実機で行った．この際，その動作に問題ないことを確認した．また，初期動作実験では ssh ポートフォワーディングなど既存の機能の利用も問題なくできた．これは他の通信機能との共存の一例を示せたと言える．

1. はじめに

現在，Society5.0 において IoT 化で全てのものがネットワークにつながり，サイバー空間とフィジカル空間を融合させたシステムが活用される社会の実現が目指されている．このようなシステムでは高速な計算機により処理・解析され，様々な課題解決がなされる．これらの課題解決の基幹として大規模並列計算が機能することが期待されている．

先に述べた大規模並列計算機の利用はそのユーザビリティ向上の観点からインターネットを通して解析計算機と繋がり，シームレスな解析処理が実行可能となることが求められる．

一般に，大規模並列計算機で計算した結果は最終的には利用者が手元の計算機で解析や可視化等の処理などにより意味を抽出することによって新たな知見の発見が行われれ，その成果が公開される．本稿では，このような意味を抽出する処理のことをポスト処理と呼ぶことにする．

大規模並列計算機の多くは計算を担当するバックエンドと呼ばれるシステムと利用者との間に，フロントエンドと呼ばれる対話的なシステムがあり，プログラム開発やバックエンドへのジョブ投入計算結果の収集等に用いられる．

従来のポスト処理は、いったんストレージに保存された主にバックエンドの計算結果をフロントエンドで収集し，それを利用者の手元の計算機に転送した後解析プログラムや可視化プログラムと入力するものであった．これらは，長時間の並列計算結果を最後にまとめてデータが出力される場合はそこでポスト処理始める．この流れでも大きな問題は無い．

しかし，近年の大規模並列計算の途中で計算結果を解析したり可視化したりすることでより効率的に研究を進めるという需要が増えつつある．このような利用では、ストレージへのファイルの読み書きに要する時間や，フロントエンドにファイルが生成されるたびに転送する手間が支障となり，円滑な解析ができない．

これらの問題は図 1 のよう一度フロントエンドのストレージにデータを出力することが原因となっている．そこで，本研究では、ストレージを介さず，直接バックエンド上の並列プログラムと利用者の手元の計算機上のプログラムを接続する通信機構を

これらの通信機構に求められる要件は以下の 3 件となる．

- 様々なユーザに幅広く利用されるために簡易で汎用的なインターフェースを持つこと．
- バックエンドで利用される標準の MPI ライブラリ [1][2] と共存できること．

[1] 帝京大学福岡医療技術学部
Teikyo University
[2] 有限会社 HREM
HREM research Inc.
[3] 合同会社つくるばい
Tsukurubai LLC
[4] 九州大学
Kyushu Univsersity
[a)] ymorie@fmt.teikyo-u.ac.jp

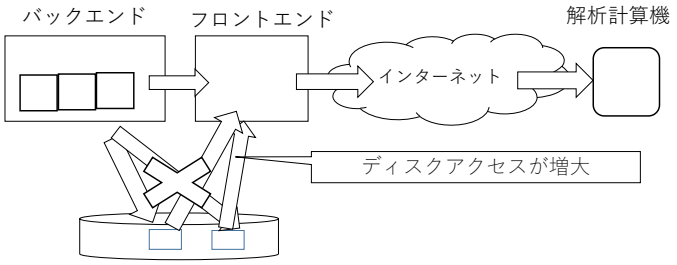

図 1 フロントエンドのストレージへのディスクアクセスが発生する例

- 継続的なデータ転送を可能とする信頼性を持つこと．

以降，上記の要件を実現する通信ライブラリ N-STDIO の開発について述べる．

本稿では，次節で N-STDIO の設計について述べる．また，第 3 節で，N-STDIO の実装について述べる．第 4 節では，プロトタイプ実装を行い初期動作実験について述べる．さらに関連研究を述べ，最後にまとめと今後の課題について述べる．

2. N-STIDO の設計

本節では，ポスト処理を行うために計算結果を解析計算機に直接転送する通信ライブラリである N-STDIO の設計について述べる．N-STDIO は前節で述べた 3 件の条件を満たすため以下の特徴を持っている．

- サーバ・クラインとモデルによる動的な接続
- 非同期な通信インフェース
- コネクションの切断の検出
- 簡易なインターフェース

これらの特徴を持つ通信ライブラリ N-STDIO のプロトタイプ実装を Github にて公開している [3].

2.1 サーバー・クライアントモデルによる動的な接続

N-STDIO は MPI のような多対多のメッセージパッシングによる通信モデルではなく，サーバークライアントモデルを採用している．これは，ヘテロな構成となる大規模計算機利用におけるポスト処理において計算資源間の接続の柔軟さが必要となる．

例えば，図 2 に典型的な利用例を示す．この例では，バックエンドからフロントエンド，インターネットを介して解析計算機へデータ転送をする構成となっている．バックエンドとフロントエンドは計算資源の管理が異なるため，初期化時に一斉に接続を行うことに向かない．特に，バックエンドを共有して利用する場合はその傾向が顕著となる．さらに，リアルタイム可視化を想定したフロントエンドの予約利用なども開始されている．このため，これらの計算資源の各要素間の接続は動的であることが最も相応しい．

また，図 3 のようにバックエンドで出力を縮約する場合は接続がバックエンドの一部のみと接続すれば良くなる構成も考えられる．この構成では前の例と必要な接続資源が変わる．このことからも柔軟な接続が可能なサーバー・クラインモデルによる動的な接続が N-STDIO には相応しい．

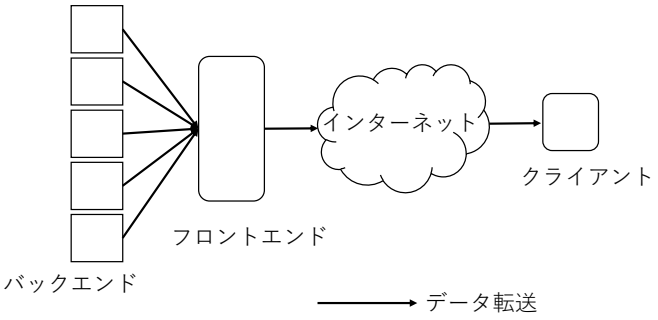

図 2 バックエンドからフロントエンド，インターネットを介して解析計算機へのデータ転送する構成

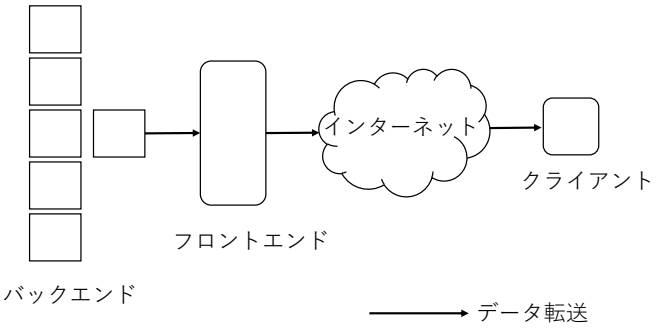

図 3 バックエンドで出力を縮約する場合のデータ転送

2.2 非同期な通信インターフェース

N-STDIO ではバックエンドの計算とデータ解析を行うポスト処理を並行して実行することを支援する通信機能の提供が求められる．提供する通信機能が同期的に実行されれば，解析計算機へデータ転送をしている間，バックエンドなどの計算が通信待ちで継続できなくなってしまう．このため，N-STDIO では非同期な通信インターフェースを採用した．

2.3 コネクションの切断の検出

ポスト処理は大規模なデータ処理を行う際には長時間の解析処理の実行が必要となる．特にインターネット等のネットワークを介したデータ転送では信頼性が重要となる．このような要件を考える場合，その信頼性能の指標としてポスト処理の継続時間が評価項目となると考えられる．

不意のコネクションの切断を防ぐことは簡単ではないので，まずはコネクションの切断を検出することが重要となる．コネクションの切断が検出されれば，そこで未送信のデータを確定することできる．さらにコネクションを再度接続し，そのデータを再計算・再転送することが可能と

なる．

N-STDIO は，信頼性を高めるため，異常検出は通信切断の発生を持って行う．相手側で通信接続が発生したら，正常切断の時以外は異常として検出する．通信において相手側の状況を知ることはできないので，正常切断以外は異常として検出する．例えば，タイムアウトが発生した場合は相手側で何らかの発生したと考え接続異常として検出する．

2.4 簡易なインターフェース

N-STDIO のように新たに開発する通信ライブラリは MPI などの標準的な通信インターフェースではと異なり，その利用に対する障壁が高いため，そのインターフェースは出来るだけ簡易である必要がある．

たとえば，ソケットプログラミングを行えば，特殊なインターコネクト以外の全ての通信に関することがが実行可能である．しかし，ソケットプログラミングは習得までの時間が多大となる．通信の”Hello, world” プログラムにあたるソケットプログラミングではさまざまなネットワークに関する情報の初期化を行うことが必要となる．ポスト処理のためだけに習得することは容易ではないと考えられる．

一方で N-STDIO は IP アドレスとポート番号のみが必須で，タイムアウト時間を追加的に設定するのみである．習得にかかるコストが低くなると考える．通信の専門家でないシミュレーションのエンジニアの利用に関してもその敷居は高くないと考える．

図 4，5 で”hello, world” プログラムにあたるソースコードとその実行結果を示している．

クライアント: cl.c

```c
#include <stdio.h>
#include "nstdio.h"

int main(int argc, char **argv){
    NET *nt;
    ND *nd;
    NHDL *hdl;

    nt = setnet(argv[1], argv[2], NTCP);
    nd = nopen(nt, "c");
    hdl = nwrite(nd, argv[3], 256);
    while (nquery(hdl));
    nclose(nd);
    freenet(nt);

    return 0;
}
```

サーバ: sv.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "nstdio.h"

int main(int argc, char **argv){
    NET *nt;
    ND *nd;
    NHDL *hdl;
    char *str;

    str = (char *)malloc(
                sizeof(char) * 256);
    nt = setnet(NULL, argv[1], NTCP);
    nd = nopen(nt, "s");
    hdl = nread(nd, str, 256);
    while(nquery(hdl));
    printf("sv: get data  [%s]¥n", str);
    nclose(nd);
    freenet(nt);

    return 0;
}
```

図 4 N-STDIO を用いたサンプルコード

```
[host1]./sv 44444
[host2]./cl host1 44444 hello_world.
[host1]sv: get data [hello_world.]
```

図 5 N-STDIO を用いたサンプルコードの実行結果

MPI も簡易なプログラムであるが，メッセージパッシングモデルのため，動的な接続とは馴染まない．また，バックエンドとフロントエンド，フロントエンドと解析用計算機の接続に関して同様に利用可能な通信ライブラリはソケットプログラミングの他に適切なものがない．

3. N-STDIO のプロトタイプ実装

今回，N-STDIO プロトタイプは TCP/IP の通信関数を利用して実装した．バックエンドとフロントエンド間は InfiniBand 接続がされている場合で IPoIB 等で TCP/IP での通信ができることが一般的である．また，インターネットを介して通信ではそのまま TCP/IP の通信が必要になる．フロントエンドでは，バックエンドとの通信，解析計算機との通信が異なるインターフェースを使うことは問題が発生しやすいと考える．

一方で，TCP/IP では性能が足りないことが懸念される．その際は ibverbs 等のインターコネクトのネイティブのインターフェースを利用することを再度検討する．

3.1 N-STDIO の基本構造

N-STDIO の基本構造は，図 6 のようにプロセスはメインスレッドと通信スレッド，送信および受信ハンドルキューで構成される．メインスレッドは関数呼び出しのみを実施し，通信処理の実行は通信スレッドが実行するようにする．これにより計算プログラムから見た場合，N-STDIO は完全な非同期動作が可能となる．

また，サーバー側で切断を検出した場合に通信スレッドによる接続待ち状態に遷移させることができる．クライアント側の接続要求は通信スレッドが待受するため，バックエンドで動作している計算プログラムは接続作業を明示する必要はない．解析計算機などのクライアント側は自身のタイミングで接続要求を発行することで接続待ちプロセスと再接続することができる．

次に送信側，受信側プロセスにおける基本動作を図 6 に示す．N-STDIO は非同期通信を発行する．このため，通信到達をハンドルを用いて確認する．ハンドルの管理は各プロセスで管理している通信ハンドルキューで行われている．

まず，送信側について述べる．送信側のメインスレッドは関数が呼び出された時に送信ハンドルを生成し，送信ハンドルキューへそのハンドルの挿入を行う．また，送信側の通信スレッドは送信ハンドルキューを監視し，送信ハンドルキューに送信ハンドルを確認すると送信ハンドルの情報に従い送信の実態の処理を行う．送信バッファへのコピーが済んだら送信処理の完了フラグを立てる．

一方，受信側についてであるが，送信側のメインスレッドと同様の動作する．受信側のメインスレッドは通信関数が呼び出された時に受信ハンドルを生成し，受信ハンドルにそのハンドルの挿入を行う．また，送信側同様，受信側

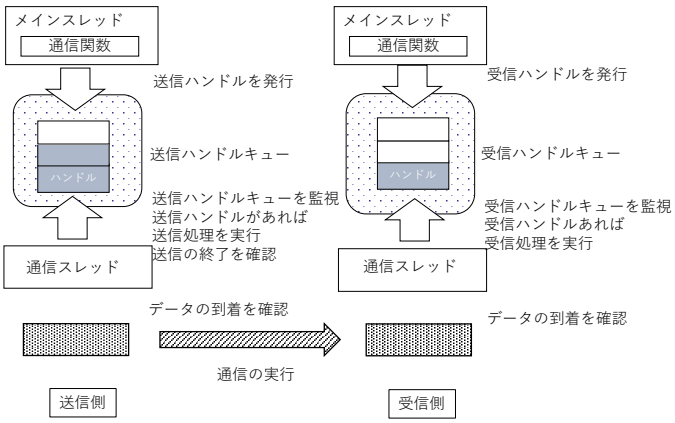

図 6 N-STDIO の基本構造

の通信スレッドも受信ハンドルを監視し，受信ハンドルキューに受信ハンドルを確認すると受信ハンドルに従い受信の実態の処理を行う．受信は実際にデータ届いているか調べ，データが届いていたら，そのデータ数を数え，データを完全に受信できたところで受信処理の完了フラグを立てる．

さらに，通信ハンドルにハンドルが挿入されていない場合は通信スレッドは一定時間スリープ状態に入る．これにより通信スレッドによるプロセッサコアの占領時間を削減することができる．

また，これらの構造をもつことでポスト処理のための通信機構のみならず，副次的に簡易なデーモン開発を容易にするライブラリともなる．

3.2 N-STDIO のインターフェース

表 1 に N-STDIO のインターフェースとその機能概要を示す．前節で説明した通り，C 言語標準入出力 (stdio) を参考し，親しまれたインターフェースに近くなるように意識して作成した．

個別の関数に関しては次のようになる．接続・切断関数として nopen, nclose の 2 個，通信関数として nwrite, nread, nquery, nsync の 4 個，その他，ネットワークの設定に関する関数として setenet, freenet, settimeout, の 3 個，計 9 個の関数を提供している．簡易であることが重要であるため，関数は利便性に影響しない範囲で最小限のものとした．

3.3 コネクションの切断検出の実装

まず，N-STDIO におけるタイムアウトについて述べる．N-STDIO のタイムアウトは通信スレッドが計測して判定する．通信スレッドの通信完了確認時および通信発行時に時間計測をし，通信スレッドが動作している間，相手側からの通信処理が一定時間発生していないことがわかったら，タイムアウトとする．

次に，切断の判定と再接続に関する動作を説明する．まず，サーバー側ではコネクションの切断検出は Fin パケット，タイムアウトの確認により行う．Fin パケットが届くか「タイムアウト」時間に至るとコネクションが切断したと判断する．コネクションが切断したと判断するとサーバーは接続待ち状態へ遷移する．これによりクライアント側が再度接続要求をすることが可能となる．

一方，クライアント側がコネクションの切断を発行を発行した際は，サーバー側に Fin パケットを送信し，自身はコネクションを切断する．サーバー側から ack がなく，タイムアウトした場合はサーバー側が切断したと判断してクライアント側でコネクションを切断する．この時，クライアント側で再度接続要求できるように現在のコネクションの初期化作業を行う．

表 1 N-STDIO のインターフェース

インターフェース	機能概要
NET *setnet(char *hostname, char *servname, uint32_t Dflag)	接続プロセスの ネットワーク情報生成
void *freenet(NET nt)	接続先プロセスの ネットワーク情報の解放
ND *nopen(NET nt, char *mode)	ネットワークへの 接続を開く
void nclose(ND *nd)	ネットワークへの 接続を閉じる
NHDL *nread(ND *nd, void *addr, size_t size)	指定コネクション間の 受信処理発行
NHDL *nwrite(ND *nd, void *addr, size_t size)	接続コネクション間の 送信処理発行
int nquery(NHDL *hdl)	通信ハンドルに 対応する通信の完了確認
void *nsync(ND *nd)	接続ネットワーク間の同期
void settimout(ND *nd, double timeout)	指定コネクションの タイムアウト時間を設定.

4. プロトタイプ実装の初期動作実験

九州大学に設置された並列計算機で N-STDIO の初期動作実験を実施した．

初期動作実験の中で図 7 のようにバックエンドとフロントエンド間の接続，さらにフロントエンドと解析計算機の接続を行う．フロントエンドと解析計算機のデータ転送はインターネットを介して行われる．これらの通信では暗号化が必要となる．そこで，このコネクションでは ssh ポートフォワーディングにより暗号処理を行った．

図 8 は実験における 3 つの動作を確認するウィンドウを並べている．左側がバッチジョブの発行を確認するウィンドウ，真ん中がフロントエンドでの実行結果の出力となる．右側がインターネットを介した解析プログラムの出力表示となる．図 8 ではいずれのプログラムも発行されてい

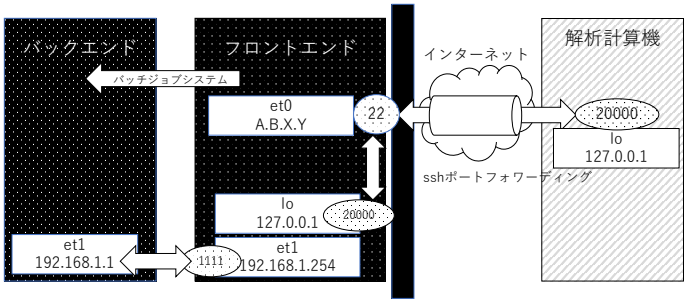

図 7 バックエンドで出力を縮約する場合のデータ転送

ない．

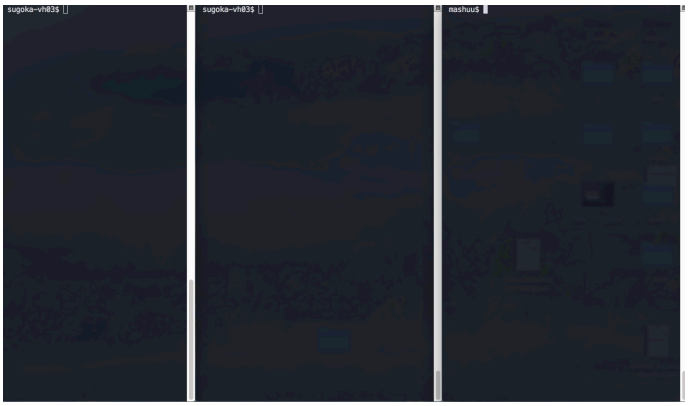

図 8 初期化状態

図 9 においてフロントエンドの出力を表示するプログラムと解析計算機での解析プログラムを実行する．このとき，解析プログラムはフロントエンドと接続が行われている．また，ジョブが投入されておらず，バックエンド上のタスクが開始されていないためフロントエンド/バックエンド間は接続待ち状態となっている．

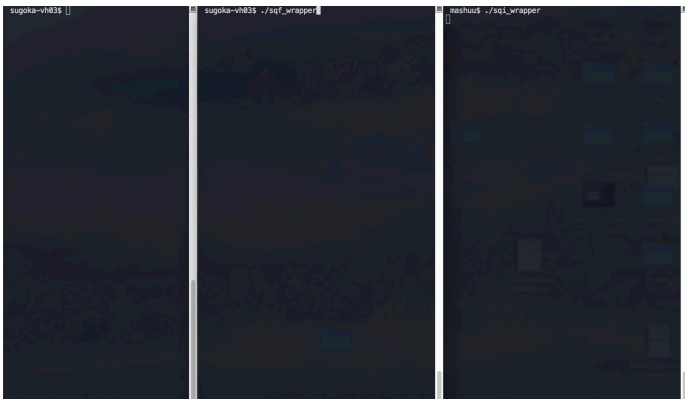

図 9 フロントエンドおよび解析計算機で解析プログラムを開始

図 10 で qsub コマンド用いてジョブの発行を行った．この際，ジョブはまだ完了していないのでフロントエンド，解析計算機の出力には変化はなく，待ち状態のままである．

図 11 で発行したタスクが終了したので，その出力が真

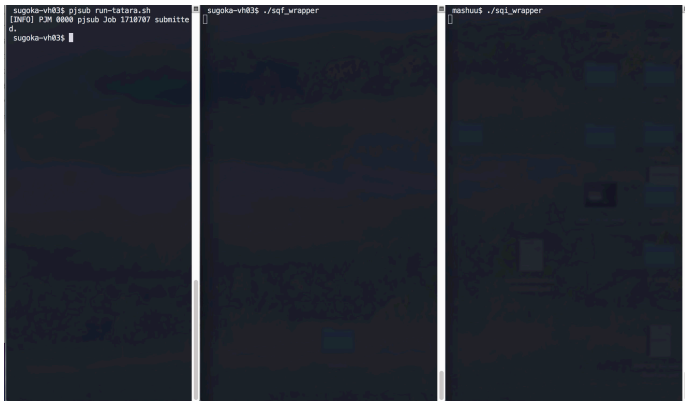

図 10 バックエンドのジョブを発行

ん中のウィンドウで表示されている．さらにそのデータを受信した解析計算機の出力も問題なく表示されている．こ

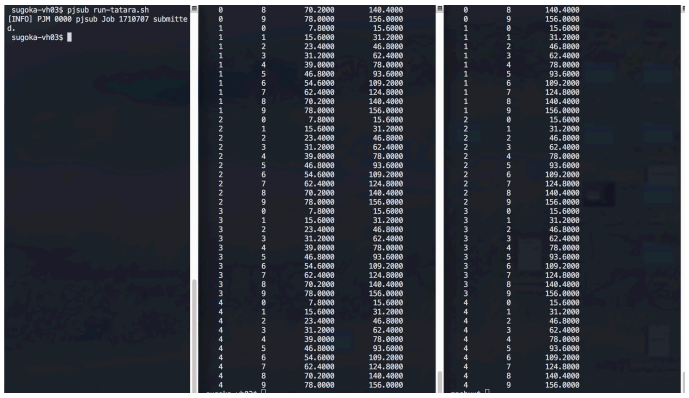

図 11 バックエンドのジョブ完了後

れにより N-STDIO を用いることで実機でポスト処理を含む一連のデータ転送が問題なくできたことが確認できた．初期動作の確認ができたことから，以降は信頼性を含めた動作確認を行うことが求められる．

5. 関連研究

小野らは，時間発展シミュレーションを実行する In situ フレームワークを紹介した．OpAS ライブラリにより提供されたバッファは外部プロセスからシミュレーション側の公開されたメモリバッファ領域へ通信が可能となり，さまざまな In situ アプローチが実行できる [4]．

小林らは，大規模数値シミュレーション，特に過渡現象に新たなボトルネックの解消を目的とした開発を行った．このようなアプリケーションでは大量のストレージ I/O が発生することから九大センターの計算機にてポスト処理に 10 日程度かかることを指摘した [5]．

上記にあげた研究はともにポスト処理において通信を利用するもので N-STDIO の利用が考えられる。

6. おわりに

ポスト処理を実行する際には柔軟性がありかつ信頼性の

高い通信機構が必要であることを指摘した．この際，開発した N-STDIO の設計および実装方法について述べた．また，プロトタイプ実装における初期動作実験について述べ，運用計算機で動作することを確認した．

今後の課題としては，実アプリケーションを用いた際の動作確認を行う．この際，ディスクレスなポスト処理を行ったことによる性能向上を評価する．また，この時，実装通信ライブラリの問題点の洗い出しを行う．さらに今回は行わなかった通信性能改善を実施することがあげられる．

謝辞 本研究を進めるにあたりご議論頂いた株式会社秋仙の小林泰三氏に感謝いたします．

参考文献

[1] MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE (online), 入手先 ⟨http://mvapich.cse.ohio-state.edu/⟩
[2] Open MPI: Open Source High Performance Computing (online), 入手先 ⟨http://www.open-mpi.org/⟩.
[3] NSTDIO Library：入手先 ⟨https://github.com/y-morie/nstdio⟩ (2021.11.03).
[4] K. Ono, J. Nonaka, H. Yoshikawa, T. Nanri, Y. Morie, T. Kawanabe, F. Shoj.: Design of a Flexible In Situ Framework with a Temporal Buffer for Data Processing and Visualization of Time-Varying Datasets, High Performance Computing - ISC High Performance 2018 International Workshops, Frankfurt/Main, Germany, June 28, 2018, Revised Selected pp.243-257 (2018).
[5] T. Kobayashi, Y. Morie, H. Jistumoto, T. Takami, and M. Aoyagi.: A new bottleneck in large-scale numerical simulations of transient phenomena, and cooperation between simulations and the post-processes, Proc. of the 1st .Pan-American Congress on Computational Mechanics (PANACM 2015), p1, (2015).