サブシステム AOBA-A の利用法

情報部デジタルサービス支援課

2020 年 10 月 1 日 初版
2025 年 5 月 22 日 第四版

目次

1 はじめに 2

2 システムの構成 2
2.1 サブシステム AOBA-A の特徴 3
2.2 プログラムの実行方法 4

3 プログラムのコンパイルと実行手順 5
3.1 フロントエンドサーバへのログイン 5
3.2 プログラムの作成とコンパイル 5
3.3 プログラムの実行 8

4 スーパーコンピュータ用数値演算ライブラリ 11
4.1 NLC (NEC Numeric Library Collection) 11

5 マニュアル 12
5.1 コンパイラマニュアル 13
5.2 性能情報取得についてのマニュアル 13
5.3 数値計算ライブラリ（NLC）マニュアル 13

6 おわりに 13

1　はじめに

本センターはスーパーコンピュータ AOBA のサブシステム AOBA-A の運用を 2020 年 10 月から開始しています。本稿では，サブシステム AOBA-A でのプログラミング利用ガイドとして，プログラムの作成からコンパイル，実行等の使い方についてご紹介します。

サイバーサイエンスセンターの大規模科学計算システムを利用するためには利用申請が必要です。利用申請について詳しくは「利用申請」(https://www.ss.cc.tohoku.ac.jp/apply-for-use/) ご参照ください。

2　システムの構成

本センターでは、スーパーコンピュータ AOBA として、サブシステム AOBA-S（SX-Aurora TSUBASA Type 30A）と、サブシステム AOBA-A（SX-Aurora TSUBASA Type 20B）、およびサブシステム AOBA-B（LX 406Rz-2）の 3 つの計算機システムをサービスしています（図 1）。また、AOBA-S 用のストレージとして 5.6PB、AOBA-A および AOBA-B 用のストレージとして 2PB のストレージシステムをサービスしています。

利用者は SINET6 を利用したリモートアクセス接続により、全国から大規模科学計算システムを利用することが可能です。

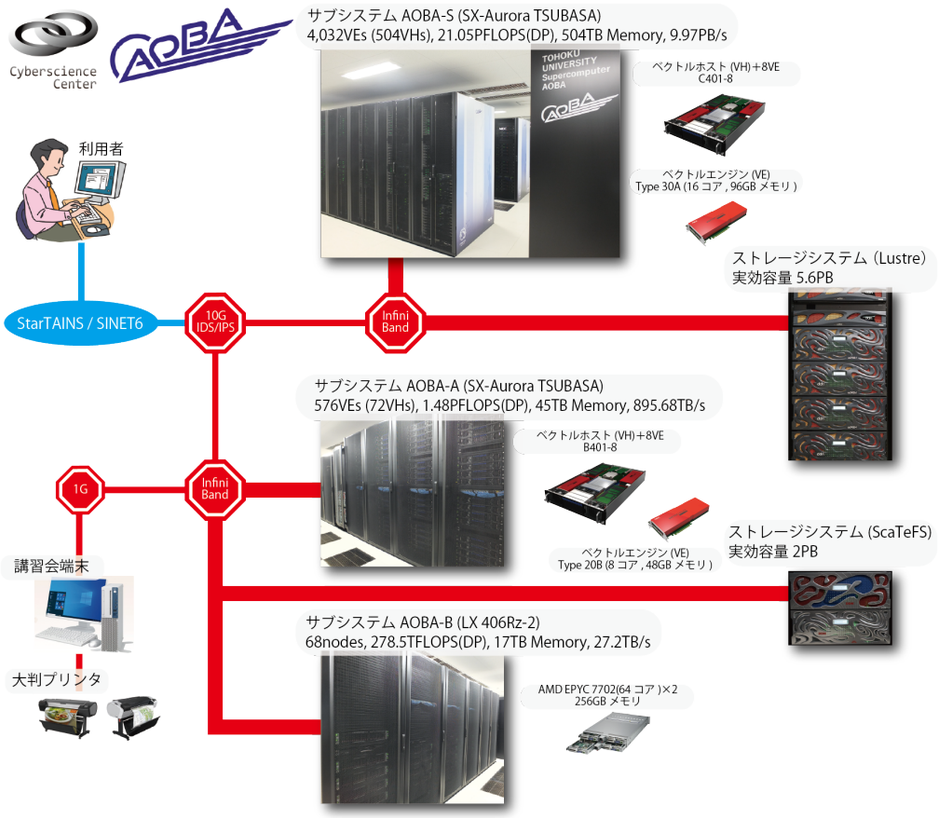

図 1　スーパーコンピュータ AOBA の構成

2.1　サブシステム AOBA-A の特徴

2.1.1　SX-Aurora TSUBASA アーキテクチャ

SX-Aurora TSUBASA（日本電気株式会社製）は，前システム SX-ACE と同じくベクトルアーキテクチャを継承しています。アプリケーション演算処理を行うベクトルエンジン（以下，VE）部と，主に OS 処理を行うベクトルホスト（以下，VH）部により構成されます。PCIe カードに搭載される VE 部はベクトルプロセッサ，及び高速メモリから構成され，x86/Linux である VH と PCIe 経由で接続されます。

1VE は理論最大演算性能 2,456GFLOPS(DP) となるマルチコア (8 コア) ベクトルプロセッサを 1 基，主記憶は 48GB を搭載し，1.53TB/s という高いメモリバンド幅でプロセッサと接続されることで，高い演算性能とメモリ性能の最適化を実現しています。

今回導入した SX-Aurora TSUBASA のシステムは，1VH と 8VE が構成単位となる B401-8 モデルを採用し，システム全体では 72 個の VH と 576 個の VE で構成されます。VE と VH を合わせたシステム全体の理論演算性能は，1.48PFLOPS(DP) ，主記憶は 45TB，総メモリバンド幅は 895.68TB/s となります。

図 2　サブシステム AOBA-A（4 ラック）

- マルチコアベクトルエンジン（8 コア）あたり，2,456GFLOPS(DP) の理論演算性能
- 1VE あたり 48GB の共有メモリを搭載し，メモリバンド幅（データ転送性能）は 1.53TB/s

2.1.2　ベクトルプロセッサ

ベクトルプロセッサとは，ベクトル演算を行う専用のハードウェアを持つコンピュータです。ベクトル演算は，ループ中で繰り返し処理されるような配列データの演算に対して一括して演算を実行するため，高速に演算することができます。

ベクトル計算機は科学技術用の数値計算に適しており，大量のデータを繰り返し処理するような大規模計算に向いていると言われています。本センターでは，流体解析や気象解析，電磁界解析をはじめとする大規模シミュレーションに利用されます。

2.1.3　ソフトウェア

Linux OS 環境上で，ベクトルエンジンの開発環境を利用できます。ベクトル処理，並列処理に対応した大規模プログラムの作成と実行が可能です。

サブシステム AOBA-A 向けにインストールされているアプリケーションについては，「アプリケーションサービス」(https://www.ss.cc.tohoku.ac.jp/software-service/) をご参照ください。

■プログラミング言語　GNU 互換環境を装備し，アプリケーションの実効性能を向上させる高度な自動ベクトル化・自動並列化機能を備えた Fortran/C/C++ コンパイラが利用出来ます。それぞれのコンパイラは自動ベクトル化機能，自動並列化機能を有していますので，既存のプログラムを修正することなくベクトル化，並列化することができます。自動並列化機能と OpenMP による共有メモリ並列実行と，システム構成に最適化された MPI ライブラリにより，分散メモリ並列実行も可能です。

■科学技術計算ライブラリ　業界標準の BLAS，FFTW，LAPACK，ScaLAPACK を含む，最適化された科学技術計算ライブラリを利用出来ます。NEC Numeric Library Collection（NLC）は，広範な分野の数値シミュレーションプログラムの作成を強力に支援する数学ライブラリのコレクションで，VE に対応しています。NLC を用いることにより，難解な数値計算アルゴリズムの詳細に煩わされることなく高度な科学技術計算プログラムを作成することができ，数値シミュレーションプログラム開発の生産性を大幅に改善することができます。NLC は，Fortran または C 言語プログラムから利用できます。

2.2　プログラムの実行方法

プログラムの実行の方法として表 1 の 4 種類が利用できます。並列実行には 3 種類があります。

■逐次実行　単一のコアで動作するプログラムです。VE 内には 8 コアがありますが，逐次実行では 1 コアのみが利用されます。メモリは 48GB まで利用出来ます。

■自動並列化　逐次処理プログラムを，コンパイラが並列化可能な箇所を自動的に判断して並列化します。プログラムを改めて書き直す必要はなく，プログラムをそのまま利用することができます。

共有メモリ並列実行が可能なので，VE 内の 8 コアまで，メモリは 48GB まで利用出来ます。

■OpenMP 並列　自動並列化と同じく逐次処理プログラムを並列化します。並列化の判断は自動ではなくユーザが明示的に行います。ソース中の並列化したい箇所に並列化指示行を追加します。

自動並列化と同じく，共有メモリ並列実行が可能なので，VE 内の 8 コアまで，メモリは 48GB まで利用出来ます。

■MPI 並列　MPI ライブラリによりプロセッサ間通信を行います。データの分割，処理方法等の並列処理手順を明示的に記述した MPI プログラムを作成する必要があります。

分散メモリ並列実行が可能なので，複数の VE を用いた実行が可能です。センターでは最大 256VE（2,048 コア），メモリは 12TB まで利用出来ます。

MPI 並列と自動並列化／ OpenMP 並列を同時に利用した並列実行も可能です。

表 1　実行の種類，特徴，最大並列数，最大メモリ量

実行の種類	並列化の方法	コードの改変量	最大並列数	最大メモリ量
逐次実行	-	-	1 コア	48GB
自動並列化	コンパイラによる自動並列化	なし	8 コア	48GB
OpenMP 並列	ユーザによる指示行挿入	少ない	8 コア	48GB
MPI 並列	MPI ライブラリを用いたプログラミング	多い	2,048 コア	12,288GB

3　プログラムのコンパイルと実行手順

本章ではプログラムのコンパイルと，サブシステム AOBA-A で実行する手順を説明します。

3.1　フロントエンドサーバへのログイン

サブシステム AOBA-A 用のコンパイルはフロントエンドサーバで行います。セキュリティ対策のため，フロントエンドサーバへはログインサーバを経由します。ストレージシステムとの大容量のデータ転送は，データ転送サーバを利用します。いずれのサーバにも公開鍵暗号方式による SSH 接続でログインを行います。

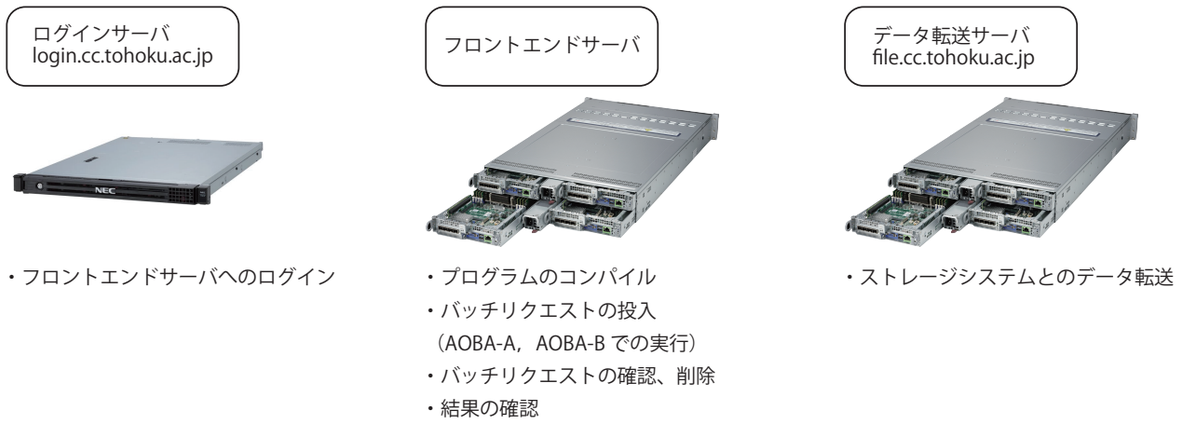

図 3　ログインサーバ，フロントエンドサーバ，データ転送サーバ

鍵の作成からログインサーバを経由してフロントエンドサーバーへログインする方法については，利用申請からログインまで (https://www.ss.cc.tohoku.ac.jp/first-use/) をご参照ください。

データ転送サーバの利用方法や，ストレージ容量の追加方法などについては，「データ転送（ストレージ）」(https://www.ss.cc.tohoku.ac.jp/storage/) をご参照ください。

3.2　プログラムの作成とコンパイル

3.2.1　ソースファイルの作成

フロントエンドサーバ上でソースファイルを作成するためのエディタは，emacs または vim が一般的です。研究室等のサーバやパソコンからソースファイルを転送する場合は，SSH 対応のファイル転送ソフトや scp コマンドで利用者のホームディレクトリに転送してください。その際，ソース（テキス

ト）ファイルは ASCII モードで転送します。Windows 環境で作成したソースコードの改行コードと文字コードを Linux 用にするためには，転送した後のソースコードに nkf コマンドを利用します（リスト 1）。

リスト 1　nkf コマンドの使用例

```
改行コードをLF，文字コードをUTF-8に変換し，別のファイルに書き出す方法
front$ nkf -Lu 変換前ソースファイル名 > 変換後ソースファイル名

改行コードをLF，文字コードをUTF-8に変換してソースファイルを上書き保存する方法
front$ nkf --overwrite -Lu ソースファイル名
```

3.2.2　コンパイラの仕様

SX-Aurora TSUBASA 用に最適化された Fortran および C/C++ コンパイラを利用できます。コンパイラの仕様は以下の通りです。

Fortran コンパイラ

- nfort，mpinfort コマンドでコンパイル
- 自動ベクトル化・自動並列化機能機能対応
- ISO/IEC 1539-1:2004 Programming languages - Fortran に準拠
- OpenMP Application Program Interface Version 4.5 に準拠
- ISO/IEC 1539-1:2010 Programming languages - Fortran の一部機能にも対応

C/C++ コンパイラ

- ncc，mpincc，nc++，mpinc++ コマンドでコンパイル
- 自動ベクトル化・自動並列化機能機能対応
- ISO/IEC 9899:2011 Programming languages - C に準拠
- ISO/IEC 14882:2014 Programming languages - C++ に準拠
- OpenMP Application Program Interface Version 4.5 に準拠

3.2.3　コンパイルを行う

ソースファイルはそれぞれの言語用コマンドでコンパイルします。単一コアで実行する逐次実行，自動並列化または OpenMP による共有メモリ並列実行および MPI ライブラリによる分散メモリ並列実行のコンパイル手順について解説します。

■Fortran プログラム　ソースコードは nfort コマンドまたは mpinfort コマンドでコンパイルします。利用したい機能があれば，表 2 に示したようなコンパイルオプションを同時に指定します。その他のオプションについてはマニュアル（5 章）をご参照ください。

ソースファイルの拡張子は，自由形式（フリーフォーマット）なら.f90 .f95 か.F90 .F95 を，固定形式（7 カラム目から記述）なら.f か.F を，2003 形式であれば.f03 か.F03 を付けます。（拡張子が大文字で始まるものは，コンパイルの前に fpp によるプリプロセス処理が行われます。）

コンパイルが成功すると，カレントディレクトリに実行オブジェクト a.out が作成されます。

表 2　Fotran,C,C++ コンパイラの主なオプション

コンパイルオプション	機能
-On	最適化レベルを指定する
n に指定できる値	
4	言語仕様を逸脱した副作用を伴う最大限の最適化・自動ベクトル化を適用する
3	副作用を伴う最適化・自動ベクトル化，および，多重ループの最適化を適用する
2	（既定値）副作用を伴う最適化・自動ベクトル化を適用する
1	副作用を伴わない最適化・自動ベクトル化を適用する
0	最適化，自動ベクトル化，並列化，インライン展開を適用しない （最適化レベルを高くすると，最適化の副作用により計算結果が変わることがありますのでご注意下さい）
-finline-functions	自動インライン展開機能を適用する
-mparallel	自動並列化機能を利用する
-fopenmp	OpenMP 並列化を利用する
-mno-parallel-omp-routine	OpenMP ディレクティブを含むルーチンを自動並列化しない （-mparallel と-fopenmp が同時に指定されたとき，ループが OpenMP の並列区間の外側にある場合は外側のループが自動並列化の対象となります）
-ftrace	性能解析 ftrace 機能用の実行ファイルを作成する
-report-diagnostics	ベクトル診断リストを出力する
-fdiag-vector=2	詳細なベクトル化診断メッセージを出力する（既定値は 1）
-report-format	ベクトル化，並列化などの最適化情報がソース行とともに出力された編集リストを出力する
-fbounds-check	バウンスチェック（配列の上下限のチェック）を行う

リスト 2　nfort コマンドの使用例

```
（逐次実行）
front$ nfort コンパイルオプション Fortranソースファイル名

（自動並列化実行）
front$ nfort -mparallel コンパイルオプション Fortranソースファイル名

（OpenMP並列実行）
front$ nfort -fopenmp コンパイルオプション Fortranソースファイル名
```

リスト 3　mpinfort コマンドの使用例

```
（MPI並列実行）
front$ mpinfort コンパイルオプション Fortranソースファイル名

（MPIと自動並列化の同時並列実行）
front$ mpinfort -mparallel コンパイルオプション Fortranソースファイル名

（MPIとOpenMPの同時並列実行）
front$ mpinfort -fopenmp コンパイルオプション Fortranソースファイル名
```

■C/C++ プログラム　ソースコードは ncc または mpincc コマンドで C プログラムを，nc++ または mpinc++ コマンドで C++ プログラムをコンパイルします。利用したい機能があれば Fortran コンパイラと同じく，表 2 に示したようなコンパイルオプションを同時に指定します。その他のオプションについてはマニュアル（5 章）をご参照ください。

ソースファイルの拡張子は，C 言語であれば.c を，C++ であれば.C .cc .cpp .cp .cxx .c++ のいずれかを付けます。

コンパイルが成功すると，カレントディレクトリに実行オブジェクト a.out が作成されます。

リスト 4　ncc/nc++ コマンドの使用例

```
（逐次実行）
front$ ncc コンパイルオプション Cソースファイル名
front$ nc++ コンパイルオプション C++ソースファイル名

（自動並列化実行）
front$ ncc -mparallel コンパイルオプション Cソースファイル名
front$ nc++ -mparallel コンパイルオプション C++ソースファイル名

（OpenMP並列実行）
front$ ncc -fopenmp コンパイルオプション Cソースファイル名
front$ nc++ -fopenmp コンパイルオプション C++ソースファイル名
```

リスト 5　mpincc/mpinc++ コマンドの使用例

```
（MPI並列実行）
front$ mpincc コンパイルオプション Cソースファイル名
front$ mpinc++ コンパイルオプション C++ソースファイル名

（MPIと自動並列化の同時並列実行）
front$ mpincc -mparallel コンパイルオプション Cソースファイル名
front$ mpinc++ -mparallel コンパイルオプション C++ソースファイル名

（MPIとOpenMPの同時並列実行）
front$ mpincc -fopenmp コンパイルオプション Cソースファイル名
front$ mpinc++ -fopenmp コンパイルオプション C++ソースファイル名
```

■性能解析情報（FTRACE）を確認する　コンパイル時に-ftrace オプションを指定すると，実行後にディレクトリに性能解析情報の結果ファイルが書き出されます。MPI 並列実行時の性能情報も取得可能です。

結果の確認は，ftrace コマンドでテキスト形式で確認するか，ftraceviewer コマンドで GUI で確認することが出来ます。図 4 は Ftrace Viewer の表示例です。

FTRACE と Ftrace Viewer についての詳細は，5 章に記載のマニュアルをご参照ください。

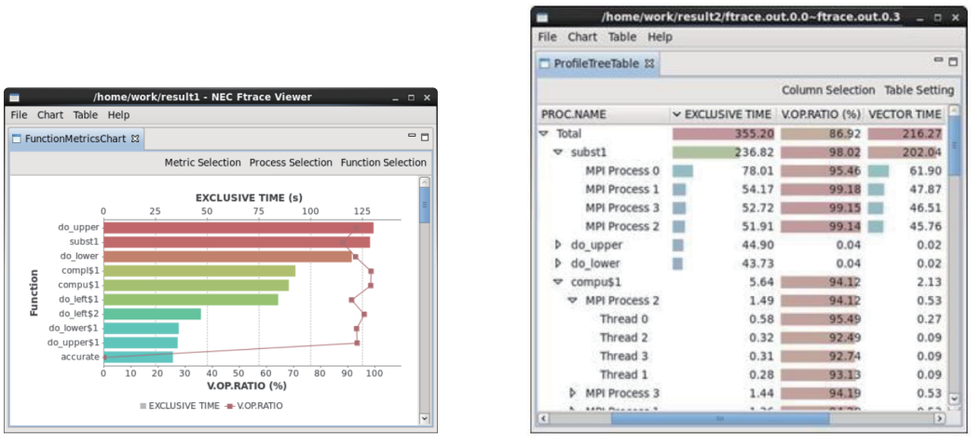

図 4　Ftrace Viewer の表示例

3.3　プログラムの実行

フロントエンドサーバでコンパイル作業を行って作成したプログラム（ジョブ）の実行は，バッチリクエストと呼ばれる方法で計算機に実行を依頼します。本センターではバッチリクエストの処理に NEC Network Queuing System V（以下，NQSV）を採用しています。図 5 にバッチリクエストの概念図を示します。

詳しくは「ジョブの実行方法」(https://www.ss.cc.tohoku.ac.jp/nqs/) をご参照ください。

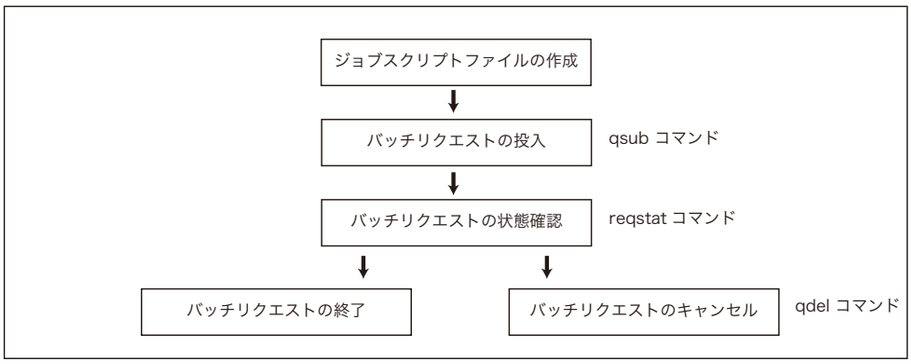

図 5　バッチリクエストの概念図

3.3.1　ジョブスクリプト

ジョブスクリプトに指定が必要となるのは,

- 投入キュー名
- 利用する VE 数
- 最大経過時間
- 作業ディレクトリへの移動コマンド
- プログラムの実行コマンド

です。他に環境変数の指定，ファイルの操作コマンド等があれば適切な箇所に手続きを記述します。オプション，環境変数は ジョブスクリプト中では#PBS の後に記述します。

サブシステム AOBA-A で実行する場合の，必須オプションを表 3 に，利用形態を表 4 に示します。

表 3　サブシステム AOBA-A で実行する場合の必須オプション

オプション	指定するもの	機能
-q	sx sxf のどちらか 1 つ	AOBA-A の利用形態を指定
--venode	1〜256 の整数値	利用する VE 数を指定
-l elapstim_req	最大経過時間を hh:mm:ss 形式で	計算機を利用する時間を指定

表 4　サブシステム AOBA-A で実行する場合の利用形態

投入キュー名	利用可能 VE 数	リクエストの実行形態
sxf	1	無料の 1VE リクエスト（VH を共用する）
sx	1	1VE リクエスト（VH を共用する）
sx	2〜256	8VE 単位で確保（VH を共用しない）

VH を共用しない：　投入したリクエストが他のリクエストと VH を共用しないで実行されます。2〜7 個の VE を使うと指定をしたリクエストも，8 個の VE と 1 個の VH を確保します。他のリクエストと VH を共用しないため，演算時間のバラツキが少なくなります。

VH を共用する：　投入したリクエストは他のリクエストと VH を共用して実行されます。2 個の VE を使うと指定したリクエストは，他 6 個の VE で別のリクエストが実行されることがあります。指定し

た数の VE（2〜8 個）を確保しやすく，混雑時にも待ち時間が短くなります。

3.3.2　ジョブスクリプトの例

■逐次実行の場合　リスト 6 とリスト 7 に逐次実行の場合のジョブスクリプトの例を示します。逐次実行の場合，プログラムは 1VE 中の 1 コアで実行されます。

リスト 6　ジョブスクリプトの例（逐次実行）

```
（run.sh の記述内容）
#!/bin/sh
#PBS -q sx #AOBA-Aを使用する
#PBS --venode 1 #VEを1個使う
#PBS -l elapstim_req=2:00:00 #最大経過時間を2時間に指定

cd $PBS_O_WORKDIR #qsubを実行したディレクトリに移動
./a.out #カレントディレクトリの a.out を実行
```

リスト 7　ジョブスクリプトの例（逐次実行，無料キュー）

```
（run.sh の記述内容）
#!/bin/sh
#PBS -q sxf #AOBA-Aの無料キューを使用する
#PBS --venode 1 #VEを1個使う
#PBS -l elapstim_req=0:30:00 #最大経過時間を30分に指定

cd $PBS_O_WORKDIR #qsubを実行したディレクトリに移動
./a.out #カレントディレクトリの a.out を実行
```

■自動並列化/OpenMP 並列実行の場合　リスト 8 に自動並列化/OpenMP 並列実行の場合のジョブスクリプトの例を示します。自動並列化/OpenMP 並列実行の場合，プログラムは 1VE 中の 2〜8 コアで実行されます。実行コア数の指定は，環境変数 VE_OMP_NUM_THREADS で行います。逐次実行と同様に，sxf も指定できます。

リスト 8　ジョブスクリプトの例（自動並列化/OpenMP 並列実行）

```
（run.sh の記述内容）
#!/bin/sh
#PBS -q sx #AOBA-Aを使用する
#PBS --venode 1 #VEを1個使う
#PBS -l elapstim_req=2:00:00 #最大経過時間を2時間に指定

export VE_OMP_NUM_THREADS=8 #VE内は8コア並列で実行

cd $PBS_O_WORKDIR #qsubを実行したディレクトリに移動
./a.out #カレントディレクトリの a.out を実行
```

■MPI 並列実行の場合　リスト 9 に MPI 並列実行の場合のジョブスクリプトの例を示します。MPI 並列実行の場合は，mpirun コマンドでプログラムの実行を行います。（逐次実行や自動並列化/OpenMP 並列実行用にコンパイルされたプログラムは，MPI 並列実行を行っても 1VE 中の 8 コア並列でしか実行されません。）

MPI プロセス数が確保した VE の物理コア数（VE 数× 8）を超えると，演算性能が著しく低下しますのでご注意下さい。

リスト 9　ジョブスクリプトの例（MPI 並列実行）

```
（run.sh の記述内容）
#!/bin/sh
#PBS -q sx #AOBA-Aを使用する
#PBS --venode 8 #VEを8個使う
#PBS -l elapstim_req=2:00:00 #最大経過時間を2時間に指定

cd $PBS_O_WORKDIR #qsubを実行したディレクトリに移動
mpirun -np 64 ./a.out #カレントディレクトリの a.out を64プロセス並列で実行
```

■MPI と自動並列化／ OpenMP 同時並列実行の場合　リスト 10 に MPI と自動並列化／ OpenMP 同時並列実行の場合のジョブスクリプトの例を示します。

（MPI プロセス数）×（VE 内並列数）が確保した VE の物理コア数（VE 数× 8）を超えると，演算性能が著しく低下しますのでご注意下さい。

リスト 10　ジョブスクリプトの例（MPI と自動並列化／ OpenMP 同時並列実行）

```
（run.sh の記述内容）
#!/bin/sh
#PBS -q sx #AOBA-Aを使用する
#PBS --venode 8 #VEを8個使う
#PBS -l elapstim_req=2:00:00 #最大経過時間を2時間に指定

export VE_OMP_NUM_THREADS=8 #VE内は8コア並列で実行

cd $PBS_O_WORKDIR #qsubを実行したディレクトリに移動
mpirun -np 8 ./a.out #カレントディレクトリの a.out を8プロセス×8コア並列で実行
```

標準出力，標準エラーファイルをプロセス毎に分割：　標準出力および標準エラー出力は，1 つのファイルに各プロセスからの出力が混在して書き出されます。このとき，システムで用意されたシェルスクリプト mpisep.sh を利用すると，同一ファイルにプロセス毎の出力が入り混じって出力されないように，MPI プロセスごとにファイルを分けて出力することができます。リスト 11 のようにシェルスクリプト/opt/nec/ve/bin/mpisep.sh を実行形式ファイル a.out の前に記述し，環境変数 NMPI_SEPSELECT に 1 から 4 の値を指定することで，表 5 に示した動作を選択します。

リスト 11　出力をプロセス毎に分割する方法

```
（run.sh の記述内容）
#!/bin/sh
#PBS -q sx #AOBA-Aを使用する
#PBS --venode 8 #VEを8個使う
#PBS -l elapstim_req=2:00:00 #最大経過時間を2時間に指定
#PBS -v NMPI_SEPSELECT=3 #出力形式に3を指定

cd $PBS_O_WORKDIR #qsubを実行したディレクトリに移動
mpirun -np 64 /opt/nec/ve/bin/mpisep.sh ./a.out #カレントディレクトリの a.out を64プロセス並列で実行
```

表 5　NMPI_SEPSELECT の引数と動作

NMPI_SEPSELECT の引数	動作
1	各 MPI プロセスの標準出力のみを stdout.uuu:rrr へ格納
2	各 MPI プロセスの標準エラーのみを stderr.uuu:rrr へ格納 (既定値)
3	各 MPI プロセスの標準出力と標準エラーをそれぞれ stdout.uuu:rrr および stderr.uuu:rrr へ格納
4	各 MPI プロセスの標準出力と標準エラーを同一ファイル stdout.uuu:rrr へ格納

実行時，stdout.*や stderr.*の同名ファイルが存在する場合は，上書きでなく追記します。

4　スーパーコンピュータ用数値演算ライブラリ

4.1　NLC (NEC Numeric Library Collection)

NEC Numeric Library Collection は，広範な分野の数値シミュレーションプログラムの作成を強力に支援する数学ライブラリのコレクションであり，Vector Engine に対応しています。NEC Numeric Library Collection を用いることにより，難解な数値計算アルゴリズムの詳細に煩わされることなく高度な科学技術計算プログラムを作成することができ，数値シミュレーションプログラム開発の生産性を大幅に改善することができます。

NEC Numeric Library Collection は，Fortran または C 言語プログラムから利用できます。

4.1.1　Fortran プログラムから利用する場合

Fortran プログラムから利用する場合，表 6 に示すライブラリが使用できます。

表 6　Fortran 用 NLC ライブラリと機能概要

ライブラリ名	機能概要
ASL ネイティブインタフェース	数値計算・統計計算のための各種アルゴリズムを備えた 科学技術計算ライブラリ
ASL 統合インタフェース	フーリエ変換，乱数，ソート
ASL FFTW3 インタフェース	FFTW (version 3.x) の API で ASL のフーリエ変換を 利用するためのインタフェースライブラリ
BLAS	ベクトル，行列の基本演算
LAPACK	連立 1 次方程式，固有値方程式，特異値分解
ScaLAPACK	連立 1 次方程式，固有値方程式，特異値分解 (分散メモリ並列用)
BLACS	ベクトル，行列の基本演算のためのメッセージパッシングライブラリ (分散メモリ並列用)
SBLAS	スパース行列の基本演算
HeteroSolver	連立 1 次方程式 (スパース行列用の直接法ソルバ)
Stencil Code Accelerator	ステンシル計算の加速

4.1.2　C プログラムから利用する場合

C プログラムから利用する場合，表 7 に示すライブラリが使用できます。

表 7　C 言語用 NLC ライブラリと機能概要

ライブラリ名	機能概要
ASL ネイティブインタフェース	数値計算・統計計算のための各種アルゴリズムを備えた 科学技術計算ライブラリ
ASL 統合インタフェース	フーリエ変換，乱数，ソート
ASL FFTW3 インタフェース	FFTW (version 3.x) の API で ASL のフーリエ変換を 利用するためのインタフェースライブラリ
CBLAS	BLAS C 言語インタフェース
SBLAS	スパース行列の基本演算
HeteroSolver	連立 1 次方程式 (スパース行列用の直接法ソルバ)
Stencil Code Accelerator	ステンシル計算の加速

NLC ライブラリの詳細およびコンパイルとリンク方法については 5 章の NLC (NEC Numeric Library Collection) ユーザーズガイドをご参照ください。

5　マニュアル

サブシステム AOBA-A についてのマニュアルは，NEC Aurora Forum の NEC SX-Aurora TSUBASA Documentation をご参照下さい。英語版，日本語版ともに提供されています。

https://www.hpc.nec/documentation

当センター独自の運用方法により，マニュアルに記載の事項が動作しない場合もあります。

5.1　コンパイラマニュアル

コンパイラについては以下のマニュアルをご参照下さい。

- C/C++ Compiler ユーザーズガイド, C/C++ Compiler User's Guide
- Fortran Compiler ユーザーズガイド, Fortran Compiler User's Guide
- NEC MPI ユーザーズガイド, NEC MPI User's Guide

5.2　性能情報取得についてのマニュアル

実行時の性能情報取得については以下のマニュアルをご参照下さい。

- PROGINF/FTRACE ユーザーズガイド, PROGINF/FTRACE User's Guide
- NEC Ftrace Viewer ユーザーズガイド, NEC Ftrace Viewer User's Guide

5.3　数値計算ライブラリ（NLC）マニュアル

数値計算ライブラリ（NLC）については以下のマニュアルをご参照下さい。

- NLC (NEC Numeric Library Collection) ユーザーズガイド,
  NLC (NEC Numeric Library Collection) User's Guide

6　おわりに

本稿では，スーパーコンピュータ AOBA のサブシステム AOBA-A のプログラミング利用ガイドとして基本的な手順を紹介しました。研究室のサーバでは実現できなかったプログラムやアイデアを，ぜひ最新鋭のスーパーコンピュータでお試しください。研究の強力なツールとしてご活用いただければ幸いです。

ご不明な点，ご質問等ございましたら，お気軽にセンターまでお問い合わせください。問い合わせについては「利用相談」(https://www.ss.cc.tohoku.ac.jp/consultation/) をご参照下さい。