サブシステム AOBA-B の利用法

情報部デジタルサービス支援課

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-B 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-B 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-A（SX-Aurora TSUBASA）とサブシステム AOBA-B（LX 406Rz-2）の 2 つの計算機システムをサービスしています（図 1）。また, 2PB のストレージシステムは AOBA-A，AOBA-B と InfiniBand で接続され，高速な I/O が可能です。

利用者は SINET5 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。

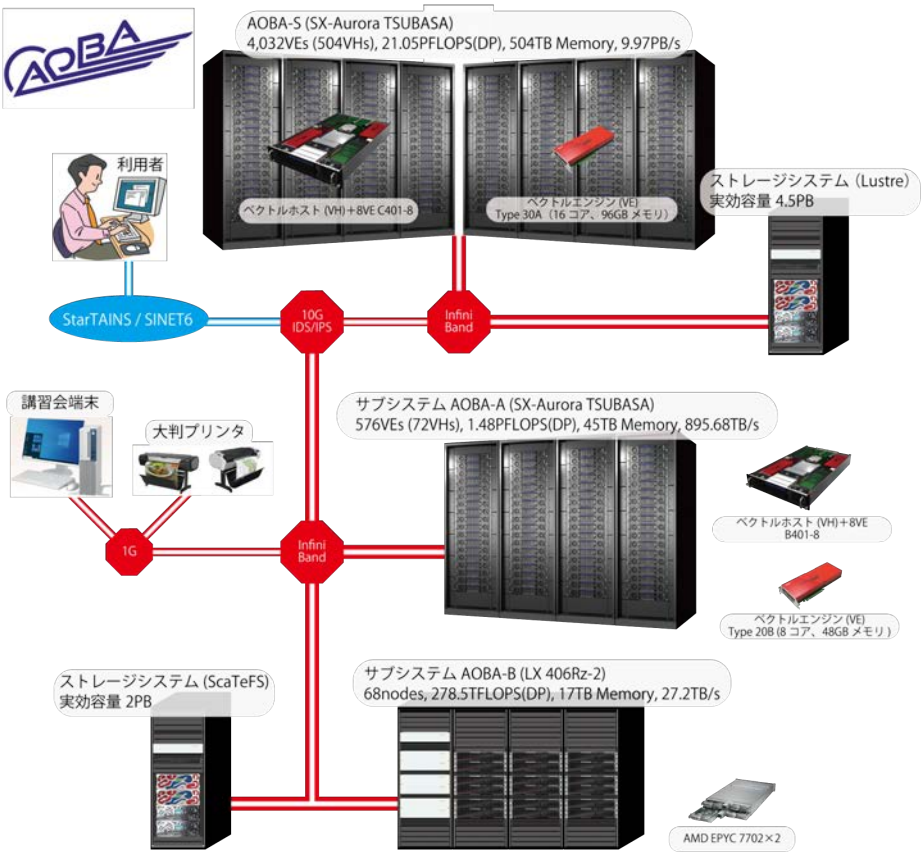

図 1　スーパーコンピュータ AOBA の構成

2.1 サブシステム AOBA-B の特徴

LX 406Rz-2 は，1 ノードに AMD EPYC プロセッサ 7702（64 コア）を 2 基と 256GB の主記憶装置を搭載し，合計 68 ノードで構成されます。OpenMP・MPI を利用したノード内の並列処理は 128 並列まで可能で，ノードあたりの最大演算性能は 4.096TFLOPS（倍精度），システム全体では 278.5 TFLOPS（倍精度）となります。複数のノードを使用した並列処理は，MPI の利用により最大 2,048 コア並列，4TB の主記憶を利用可能です。ベクトル演算に不向きなプログラムや，商用アプリケーションの高速な実行が可能です。

図 2　サブシステム AOBA-B（4 ラック）

- 1 ノード（64 コア× 2）あたり，4.096TFLOPS(DP) の理論演算性能
- 1 ノードあたり 256GB 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 409.6GB/s
- 一時領域として，アクセスが高速な SSD ストレージを搭載

2.1.1 ソフトウェア

フロントエンドサーバの Linux OS 環境上で，EPYC プロセッサ向けのプログラムを作成できます。一般的な Linux OS ですので，商用アプリケーションやオープンソフトウェアも実行が可能です。

サブシステム AOBA-B 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　Fortran/C/C++ コンパイラとして，AMD Optimizing C/C++ Compiler と GNU Compiler Collection が利用出来ます。OpenMP による共有メモリ並列実行と MPI ライブラリによる分散メモリ並列実行が可能です。また，前システムの並列コンピュータで実行していたソースコードの移行用として，Intel Compiler をライセンス数限定で利用できます。

■科学技術計算ライブラリ　EPYC プロセッサに最適化された, AMD Optimizing CPU Libraries (AOCL) を利用出来ます。

- AOCL-Sparse
- BLIS
- FFTW (Fastest Fourier Transform in the West)
- AMD Math Library (LibM)
- libFLAME
- ScaLAPACK
- AMD Random Number Generator Library

AOCL の詳細については https://developer.amd.com/tools-and-sdks/をご参照ください。

また，Intel Compiler からは MKL ライブラリが利用出来ます。MKL についての詳細は https://www.xlsoft.com/jp/products/intel/perflib/mkl/index.html をご参照ください。

■アプリケーション性能解析ツール　AMD μ Prof が利用できます。実行したプログラムの性能およびシステム性能分析機能が利用出来ます。

AMD μ Prof の詳細については https://developer.amd.com/tools-and-sdks/をご参照ください。

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 3 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。1 ノード内には 64 コアの CPU が 2 つありますが，逐次実行では 1 コアのみが利用されます。メモリは 256GB まで利用出来ます。

■OpenMP 並列　逐次処理プログラムをユーザが明示的に並列化します。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。

共有メモリ並列実行が可能なので，1 ノード内の 128 コア（2CPU）を利用出来ます。メモリは 256GB 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，複数のノードを用いた実行が可能です。サブシステム AOBA-B では最大 16 ノード（2,048 コア），メモリは 4,096GB まで利用出来ます。

MPI 並列と 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	-	-	128 コア	256GB
OpenMP 並列	ユーザによる指示行挿入	少ない	128 コア	256GB
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	4,096GB

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-B で実行する手順を説明します。コンパイラについては AOCC を使用した場合について説明します。

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-B 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。

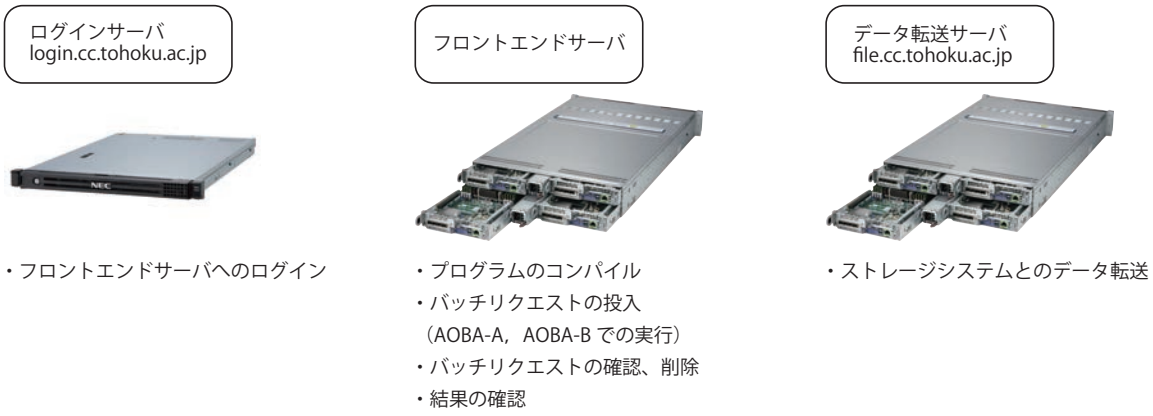

図 3 ログインサーバ，フロントエンドサーバ，データ転送サーバ

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。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 利用可能なコンパイラの種類

AOCC コンパイラとして，以下の Fortran および C/C++ コンパイラを利用できます。

AOCC

- flang：Fortran コンパイラ
- mpifort：MPI プログラム用 Fortran コンパイラ
- clang：C コンパイラ
- mpicc：MPI プログラム用 C コンパイラ
- clang++：C++ コンパイラ
- mpic++：MPI プログラム用 C++ コンパイラ

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，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 flang コマンドまたは mpifort 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（4 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 cpp 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，標準では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　Fortran,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
-march=znver2	EPYC CPU（Rome）用にコンパイルすることを指定する
-On	最適化レベルを指定する
n に指定できる値	
fast	最大限の最適化を適用する
3	積極的に最適化を適用する
2	（既定値）標準的な最適化を適用する
1	最低限の最適化を適用する
0	すべての最適化を無効化する
-fopenmp	OpenMP 並列化を利用する

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。

リスト 2　flang コマンドの使用例

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

リスト 3　mpifort コマンドの使用例

```
（MPI並列実行）
front$ mpifort 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$ mpifort -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 clang または mpicc コマンドで C プログラムを，clang++ または mpic++ 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（4 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　clang/clang++ コマンドの使用例

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

リスト 5　mpicc/mpic++ コマンドの使用例

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

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 4 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。

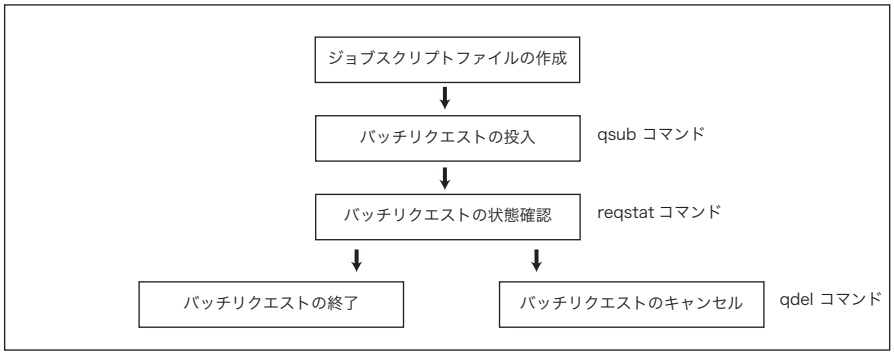

図 4　バッチリクエストの概念図

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
- 利用するノード数
- 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-B で実行する場合の，必須オプションを表 3 に示します。

表 3　サブシステム AOBA-B 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	lx	AOBA-B を利用することを指定
-b	1〜16 の整数値	利用するノード数を指定
-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

3.3.2　ジョブスクリプトの例

■逐次実行の場合　リスト 6 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 1CPU 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.csh の記述内容）
#!/bin/csh
#PBS -q lx #AOBA-Bを使用する
#PBS -b 1 #1ノードを使う
#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
```

■OpenMP 並列実行の場合　リスト 7 に OpenMP 並列実行の場合のジョブスクリプトの例を示します。OpenMP 並列実行の場合，プログラムは 1 ノード内の 2〜128 コアで実行されます。実行コア数の指定は，環境変数 OMP_NUM_THREADS で行います。

リスト 7　ジョブスクリプトの例（OpenMP 並列実行）

```
（run.csh の記述内容）
#!/bin/csh
#PBS -q lx #AOBA-Bを使用する
#PBS -b 1 #1ノードを使う
#PBS -l elapstim_req=2:00:00 #最大経過時間を2時間に指定
#PBS -v OMP_NUM_THREADS=128 #128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 8 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。mpirun コマンドの後に，-b オプションで指定した使用ノード数に応じて，バッチリクエストのノード割当てを自動的に設定するための環境変数$NQSV_MPIOPT を指定します。（逐次実行や 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，1 ノード内でしか実行されません。）

MPI プロセス数が確保したノードの物理コア数（ノード数× 128）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 8　ジョブスクリプトの例（MPI 並列実行）

```
（run.csh の記述内容）
#!/bin/csh
#PBS -T openmpi # MPI実行環境を指定，AOCCではopenmpiを使用する
#PBS -q lx #AOBA-Bを使用する
#PBS -b 1 #1ノードを使う
#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 128 $NQSV_MPIOPTS ./a.out #カレントディレクトリの a.out を128プロセス並列で実行
```

■MPI，OpenMP 並列実行同時利用の場合　リスト 9 に MPI，OpenMP 並列実行同時利用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）×（ノード内並列数）が確保したノードの物理コア数（ノード数× 128）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，OpenMP 同時並列実行）

```
（run.csh の記述内容）
#!/bin/csh
#PBS -T openmpi # MPI実行環境を指定，AOCCではopenmpiを使用する
#PBS -q lx #AOBA-Bを使用する
#PBS -b 2 #2ノードを使う
#PBS -l elapstim_req=2:00:00 #最大経過時間を2時間に指定
#PBS -v OMP_NUM_THREADS=64 #ノード内は64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 4 $NQSV_MPIOPTS ./a.out #カレントディレクトリの a.out を4プロセス×64コア並列で実行
```

4　マニュアル

4.1　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。

- AMD Optimizing C/C++ Compiler：https://developer.amd.com/amd-aocc/
- Intel Compiler：https://www.xlsoft.com/jp/products/intel/studio_xe/

4.2　数値計算ライブラリマニュアル

並列コンピュータ用数値計算ライブラリについては以下のマニュアルをご参照下さい。

- AMD Optimizing CPU Libraries（AOCL）：https://developer.amd.com/amd-aocl/
- Intel MKL：https://www.xlsoft.com/jp/products/intel/perflib/mkl/

5　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-B 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。