

# TSUBAME3.0利用の手引き

---

## Table of contents

---

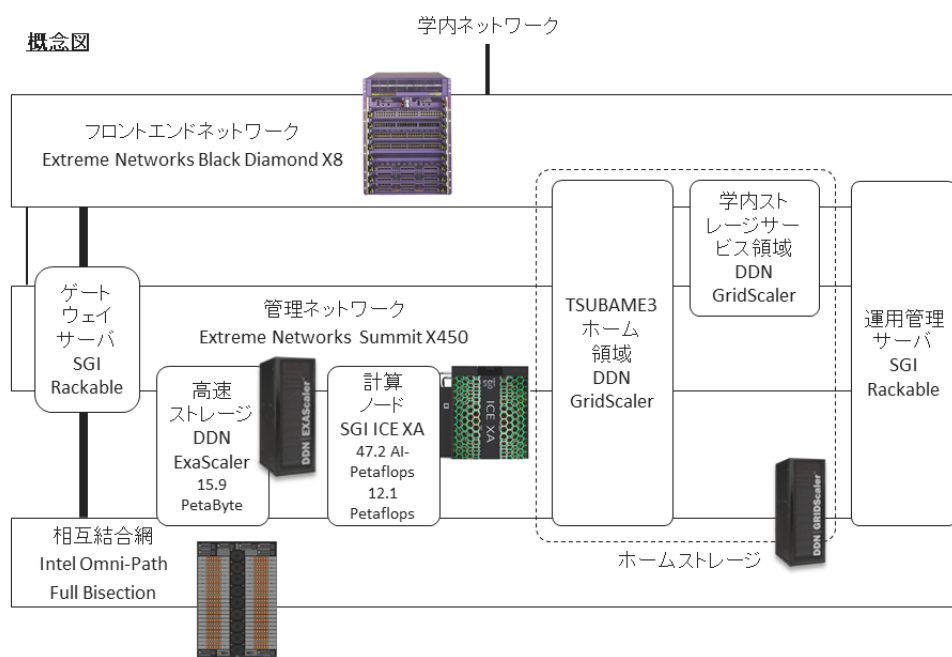
1. TSUBAME3.0概要	4
1.1. システム概念	4
1.2. 計算ノードの構成	4
1.3. ソフトウェア構成	5
1.4. ストレージ構成	6
2. 利用開始にあたって	7
2.1. アカウントの取得	7
2.2. ログイン方法	7
2.3. パスワード管理	8
2.4. ログインシェルの変更	9
2.5. TSUBAMEポイントの確認	9
3. ストレージ環境	10
3.1. Homeディレクトリ	10
3.2. 高速ストレージ領域	10
3.3. 学内からのCIFSによるアクセス	10
4. ソフトウェア環境	12
4.1. 利用環境の切り換え方法	12
4.2. バッチスクリプト内での利用	13
4.3. Intelコンパイラ	13
4.4. PGIコンパイラ	15
4.5. 並列化	15
4.6. GPU環境	16
5. ジョブスケジューリングシステム	20
5.1. 計算ノードの種類	20
5.2. ジョブの投入	21
5.3. 計算ノードの予約	29
5.4. インタラクティブジョブの投入	30
5.5. 計算ノードへのSSHログイン	32
5.6. 計算ノード上のストレージの利用	33
6. ISVアプリケーション	34
6.1. ANSYS	35
6.2. Fluent	36
6.3. ABAQUS	38
6.4. ABAQUS CAE	39
6.5. Marc & Mentat / Dytran	39

6.6. Nastran	41
6.7. Patran	41
6.8. Gaussian	41
6.9. GaussView	43
6.10. AMBER	44
6.11. Materials Studio	46
6.12. Discovery Studio	48
6.13. Mathematica	50
6.14. Maple	50
6.15. AVS/Express	51
6.16. AVS/Express PCE	52
6.17. LS-DYNA	53
6.18. LS-PrePost	55
6.19. COMSOL	56
6.20. Schrodinger	57
6.21. MATLAB	58
6.22. Arm Forge	59
7. フリーウェア	61
7.1. 量子化学/MD関連ソフトウェア	62
7.2. CFD関連ソフトウェア	64
7.3. GPU用数値計算ライブラリ	64
7.4. 機械学習、ビッグデータ解析関連ソフトウェア	65
7.5. 可視化関連ソフトウェア	69
7.6. その他フリーウェア	73
改訂履歴	83

# 1. TSUBAME3.0概要

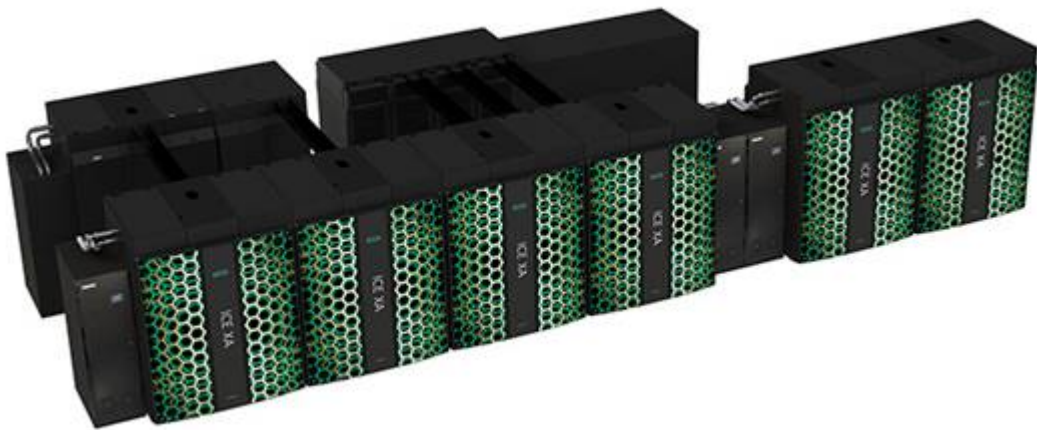
## 1.1. システム概念

本システムは、東京工業大学における種々の研究・開発部門から利用可能な共有計算機です。本システムの倍精度総理論演算性能は12.15PFLOPS、総主記憶容量は135TiB、磁気ディスク容量は15.9PBです。各計算ノード及びストレージシステムはOmni-Pathによる高速ネットワークに接続され、また現在は20Gbpsの速度でインターネットに接続、将来的にはSINET5を経由し100Gbpsの速度でインターネットに接続される予定です。2019年5月時点。TSUBAME3.0の全体概念を以下に示します。



## 1.2. 計算ノードの構成

本システムの計算ノードはSGI ICE XA 540ノードで構成されたブレード型大規模クラスタシステムです。1台の計算ノードには、Intel Xeon E5-2680 v4(2.4GHz、14core)を2基搭載し、総コア数は15,120コアとなります。また、主記憶容量は計算ノードあたり256GiBを搭載し、総主記憶容量は、135TiBとなります。各計算ノードは、Intel Omni-Pathインタフェースを4ポート有しており、Omni-Pathスイッチによりファットツリーで接続されます。



TSUBAME3.0のマシンの基本スペックは次の通りです。

演算部名		計算ノード 540台
ノード構成	台あたり	
CPU		Intel Xeon E5-2680 v4 2.4GHz× 2CPU
コア数/スレッド		14コア / 28スレッド×2CPU
メモリ		256GiB
GPU		NVIDIA TESLA P100 for NVlink-Optimized Servers ×4
SSD		2TB
インターコネクト		Intel Omni-Path HFI 100Gbps? ×4

### 1.3. ソフトウェア構成

本システムのオペレーティングシステム(OS)は、下記の環境を有しています。

- SUSE Linux Enterprise Server 12 SP2

OS構成は、サービス実行形態に応じて動的に変更されます。

また、本システムで利用可能なアプリケーションソフトウェアに関しては、[ISVアプリケーション](#)、[フリーウェア](#)を参照ください。

### 1.4. ストレージ構成

本システムでは、様々なシミュレーション結果を保存するための高速・大容量のストレージを備えています。計算ノードでは高速ストレージ領域としてLustreファイルシステムにより、HomeディレクトリはGPFS+cNFSによりファイル共有されています。また、各計算ノードにローカルクラッチ領域として2TBのSSDが搭載されています。本システムで利用可能な、各ファイルシステムの一覧を以下に示します。

用途	マウント	容量	ファイルシステム
Homeディレクトリ	/home	40TB	GPFS+cNFS
共有アプリケーション配備	/apps		
高速ストレージ領域1	/gs/hs0	4.8PB	Lustre
高速ストレージ領域2	/gs/hs1	4.8PB	Lustre
高速ストレージ領域3	/gs/hs2	4.8PB	Lustre
ローカルクラッチ領域	/scr	各ノード1.9TB	xfs SSD)

## 2. 利用開始にあたって

### 2.1. アカウントの取得

本システムを利用するには、予め利用申請を行い、ユーザIDを取得する必要があります。

利用者区分に応じて必要な操作・手続きが異なりますので、詳細は[アカウント取得方法](#)をご参照ください。

### 2.2. ログイン方法

ログインノードにアクセスするためには、ログインに使うSSH公開鍵をアップロードする必要があります。公開鍵の登録の操作は、[TSUBAME3.0ポータル利用の手引き SSH公開鍵の登録](#)を参照ください。

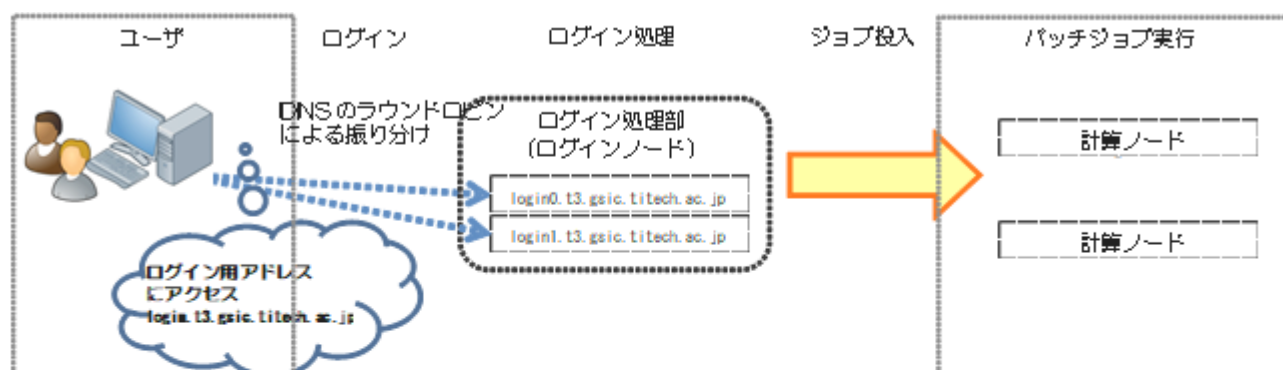
本システムを利用するには、まずログインノードにログインする必要があります。ログインノードへのログインは、DNSのラウンドロビンによる自動振り分けが行われます。



Warning

ログインノードは複数のユーザで共用されているため、[負荷のかかる操作は行わない](#)でください。

利用イメージを以下に示します。



ログイン先には、SSHで接続します。また、ファイル転送はSFTPで接続します。

```
login.t3.gsic.titech.ac.jp
```

任意のログインノードにログインしたい場合は、以下のホスト名(FQDN)を指定してください。

```
login0.t3.gsic.titech.ac.jp
login1.t3.gsic.titech.ac.jp
```

Linux/Mac/Windows(Cygwin)からX転送オプションを有効にして接続するには以下の例のようになります。

例)アカウント名が `gsic_user`、秘密鍵が `~/ssh/t3-key` の場合

```
$ ssh gsic_user@login.t3.gsic.titech.ac.jp -i ~/ssh/t3-key -YC
```



鍵ペアの格納場所を標準のパス・ファイル名とした場合には `-i` オプションは指定不要です。

最初にログインする際、クライアントの設定によっては以下のようなメッセージが出る場合があります。その場合は `yes` と入力してください。

```
The authenticity of host 'login0.t3.gsic.titech.ac.jp (131.112.3.21)' can't be established.  
ECDSA key fingerprint is SHA256:RImxLoC4tBjIYQljwIImCKshjef4w7Pshjef4wtBj  
Are you sure you want to continue connecting (yes/no)?
```

### 2.2.1. ログインノードにおける高負荷プログラムの実行制限について

ログインノード(login, login0, login1)は多数のユーザが同時に利用しているため、CPUを占有するプログラムを実行しないでください。並列計算、長時間な計算は計算ノードを利用してください(qsub/qrshコマンドを利用する)。以下に判断の目安を例示します。ここで許可されている、もしくは触れていない項目についても、他ユーザの利用の妨げとなっているプログラムについては、システム管理者の判断で予告なく停止・削除させていただきます。

基本的には問題ないこと

- ・ファイルの転送・展開 (scp, sftp, rsync, tarなど)
- ・プログラムのコンパイル(ただし並列コンパイルなど多数の資源を一度に使う場合は計算ノードをご利用ください)

行わないで欲しいこと

- ・ISVアプリケーション、フリーソフトウェアおよび自作プログラムによる計算の実行
- ・10分を超えるプログラムの実行(ファイル転送を除く)
- ・並列処理を行うプログラム(pythonによるものやMPIを含む)の実行
- ・メモリを大量に消費するプログラムの実行
- ・多数のプロセスの同時実行(並列コンパイルなど)
- ・常駐するプログラムや、停止時に方法の如何に関わらず自動で再実行されるプログラム(VSCode ServerやJupyter Notebookなど)
- ・その他、CPUに長時間負荷がかかる作業

ログインノードでは、1プロセスあたり4GBのメモリ制限を課しています。また、システムに負荷を与えているプログラムはシステム管理者によって予告なく停止させていただきますのでご注意ください。

ログインノードが高負荷で作業しづらい時や、負荷のかかる作業を行うときは、ジョブスケジューラ経由で[インタラクティブジョブ](#)として実行してください。

## 2.3. パスワード管理

本システムのユーザアカウントはLDAPサーバで管理され、システム内の認証はSSHの鍵認証で行っています。このため、計算ノードの利用にあたってパスワードを意識する必要はありませんが、学内から高速ストレージへのアクセスなどパスワードが必要になるケースがあります。

パスワードの変更が必要になる場合は、TSUBAME3.0利用ポータルから行ってください。パスワードのルールについては、TSUBAME3.0利用ポータルのパスワード設定のページをご覧ください。



## 2.4. ログインシェルの変更

ユーザ登録の時点で各ユーザアカウントのログインシェルはbashとなっています。デフォルトのログインシェルを変更するにはchshコマンドを利用ください。利用可能なログインシェルはbash, csh, ksh, tcsh, zshとなります。引数なしのchshコマンドで利用可能なログインシェルを確認することができます。

```
$ chsh
Usage: chsh shell(/bin/bash /bin/csh /bin/sh /bin/ksh /bin/tcsh /bin/zsh).
```

以下は、ログインシェルをtcshに変更する例です。

```
$ chsh /bin/tcsh
Please input Web Portal Password(not SSH Passphrase)
Enter LDAP Password: xxxxxx      ← パスワードを入力してください
Changing shell succeeded!!
```

## 2.5. TSUBAMEポイントの確認

コマンドでのTSUBAMEポイントの確認は `t3-user-info group point` コマンドにて確認できます。以下は、TESTGROUPのTSUBAMEポイントを確認する例です。

```
$ t3-user-info group point -g TESTGROUP
gid      group_name      deposit      balance
-----
xxxx     TESTGROUP            5000        800000000
```

参加しているTESTGROUPの仮ポイントが5000ポイント、残TSUBAMEポイントが800000000ポイントである状況が確認できます。

## 3. ストレージ環境

本システムでは、Homeディレクトリ以外にも 高速ストレージ領域のLustreファイルシステム、ローカルクラッチ領域のSSD領域、SSDをまとめて作成する共有クラッチ領域のBeeGFS On Demandといった様々な並列ファイルシステムを利用することができます。

### 3.1. Homeディレクトリ

HOMEディレクトリはユーザあたり 25GiBを利用できます。

使用容量は `t3-user-info disk home` コマンドにて確認できます。以下は、TESTUSERのHOMEディレクトリの容量を確認する例です。

```
$ t3-user-info disk home
uid name          b_size (GB) b_quota (GB) i_files i_quota
-----
2011 TESTUSER      7           25    101446 2000000
```

25GBのクォータ制限のうち、7GB利用し、inode制限については、200万のクォータ制限のうち、約10万利用している状況が確認できます。

クォータ制限を超過した場合、新規の書き込みができなくなりますのでご注意ください。クォータ制限を下回るように容量を削減すれば再度書き込みが可能になります。

稀に容量を削減してもクォータ制限を超過したままの状態が維持される場合があります。その際は最大 日程度待機する事でクォータの再計算処理が行われ、正常な値に戻ります。

### 3.2. 高速ストレージ領域

高速ストレージ領域はLustreファイルシステムで構成され、グループディスクとして購入することで利用することができます。グループディスクの購入方法は「TSUBAME3.0ポータル利用説明書」をご参照ください。

グループディスクの使用容量は `t3-user-info disk group` コマンドにて確認できます。以下は、TESTGROUPのグループディスクの容量を確認する例です。

```
$ t3-user-info disk group -g TESTGROUP
gid group_name      size (TB) quota (TB) file (M) quota (M) size (TB) quota (TB) file (M) quota (M) size (TB) quota (TB) file (M) quota (M)
-----
xxxx TESTGROUP      0.00      0      0.00      0      59.78      100      7.50      200      0.00      0      0.00      0
```

指定したTESTGROUPグループでは、/gs/hs1のみ購入し、100TBのクォータ制限のうち、約60TB利用し、inode制限については、2億のクォータ制限のうち、750万利用している状況が確認できます。

### 3.3. 学内からのCIFSによるアクセス

TSUBAME3.0では、高速ストレージ領域に対して学内のWindows/Mac端末からCIFSによるアクセスが可能です。以下のアドレスでアクセスすることができます。

```
\\gshs.t3.gsic.titech.ac.jp
```

アカウントはTSUBAME3.0のアカウント、パスワードはポータルで設定したパスワードになります。Windowsからアクセスする際には、以下のようTSUBAMEドメインを指定してください。

ユーザー名	TSUBAME\ (TSUBAME3.0アカウント名)
-------	-----------------------------

パスワード	(TSUBAME3.0アカウントのパスワード)
-------	-------------------------

/gs/hs0、/gs/hs1、/gs/hs2に対応して、T3\_HS0、T3\_HS1、T3\_HS2となっています。グループディスクとして購入したディレクトリへアクセスしてください。

## 4. ソフトウェア環境

---

### 4.1. 利用環境の切換え方法

---

本システムでは、moduleコマンドを使用することでコンパイラやアプリケーション利用環境の切り替えを行うことができます。

#### 4.1.1. 利用可能なmodule環境の表示

利用可能なmodule環境はmodule availまたはmodule avaで確認できます。

```
$ module avail
```

読み込めるバージョンについてはTSUBAME計算サービスWebページの[ソフトウェア構成](#)をご確認下さい。

#### 4.1.2. module環境の設定情報表示

module環境の設定情報を確認したい場合、「module whatisモジュール名」を実行します。

```
$ module whatis intel/17.0.4.196
intel/17.0.4.196      : Intel Compiler version 17.0.4.196 (parallel_studio_xe_2017) and MKL
```

#### 4.1.3. module環境のロード

module環境をロードしたい場合、「module load モジュール名」を実行します。

```
$ module load intel/17.0.4.196
```

バッチスクリプトにおいてロードするmoduleは、コンパイル時と同様のものをロードしてください。

#### 4.1.4. module環境の表示

現在使用しているmodule環境を確認したい場合、「module list」を実行します。

```
$ module list
Currently Loaded Modulefiles:
  1) intel/17.0.4.196  2) cuda/8.0.61
```

#### 4.1.5. module環境のアンロード

ロードしたmodule環境をアンロードしたい場合「module unload モジュール名」を実行します。

```
$ module list
Currently Loaded Modulefiles:
  1) intel/17.0.4.196  2) cuda/8.0.61
$ module unload cuda
$ module list
Currently Loaded Modulefiles:
  1) intel/17.0.4.196
```

#### 4.1.6. module環境の初期化

ロードしたmodule環境を初期化したい場合、「module purge」を実行します。

```
$ module list
Currently Loaded Modulefiles:
  1) intel/17.0.4.196  2) cuda/8.0.61
$ module purge
$ module list
No Modulefiles Currently Loaded.
```

## 4.2. バッチスクリプト内での利用

バッチスクリプト内でmoduleコマンドを実行する場合、以下のとおり、バッチスクリプト内でmoduleコマンドの初期設定を行う必要があります。

【実行シェルがsh, bashの場合】

```
. /etc/profile.d/modules.sh  
module load intel/17.0.4.196
```

【実行シェルがcsh, tcshの場合】

```
source /etc/profile.d/modules.csh  
module load intel/17.0.4.196
```

## 4.3. Intelコンパイラ

本システムではコンパイラとして、Intelコンパイラ、PGIコンパイラおよびGNUコンパイラが利用できます。Intelコンパイラの各コマンドは以下のとおりです。

コマンド	言語	コマンド形式
ifort	Fortran 77/90/95	<code>\$ ifort [オプション] source_file</code>
icc	C	<code>\$ icc [オプション] source_file</code>
icpc	C++	<code>\$ icpc [オプション] source_file</code>

利用する際は、moduleコマンドでintelを読み込んでください。-helpオプションを指定して頂くとコンパイラオプションの一覧が表示されます。

### 4.3.1. コンパイルの主なオプション

コンパイルの最適化オプションを以下に示します。

オプション	説明
<code>-O0</code>	すべての最適化を無効にします。
<code>-O1</code>	最適化を有効にします。コードサイズを大きくするだけで高速化に影響を与えるような一部の最適化を無効にします。
<code>-O2</code>	最適化を有効にします。一般的に推奨される最適化レベルです。 ベクトル化は O2 以上のレベルで有効になります。-O オプションを指定しない場合、デフォルトでこちらが指定されます。
<code>-O3</code>	O2 よりも積極的に最適化を行い、融合、アンロールとジャムのブロック、IF 文の折りたたみなど、より強力なループ変換を有効にします。
<code>-xCORE-AVX2</code>	Intel プロセッサ向けのIntel アドバンスド・ベクトル・エクステンション 2 (Intel AVX2)、Intel AVX、SSE4.2、SSE4.1、SSSE3、SSE3、SSE2、SSE 命令を生成します。Intel AVX2 命令セット対応のIntel プロセッサ向けに最適化します。
<code>-xSSE4.2</code>	Intel プロセッサ向けのIntel SSE4 高効率および高速な文字列処理命令、Intel SSE4 ベクトル化コンパイラ命令およびメディア・アクセラレーター命令、およびIntel SSSE3、SSE3、SSE2、SSE 命令を生成します。Intel SSE4.2 命令セット対応のIntel プロセッサ向けに最適化します。
<code>-xSSSE3</code>	Intel プロセッサ向けのIntel SSSE3、SSE3、SSE2、SSE 命令を生成します。Intel SSSE3 命令セット対応のIntel プロセッサ向けに最適化します。x オプションを指定しない場合、デフォルトでこちらが指定されます。
<code>-qopt-report=n</code>	最適化レポートを生成します。デフォルトでは、レポートは.optprt 拡張子を持つファイルに出力されます。n には、0 (レポートなし) から5 (最も詳しい) の詳細レベルを指定します。デフォルトは2 です。
<code>-fp-model precise</code>	浮動小数点演算のセマンティクスを制御します。浮動小数点データの精度に影響する最適化を無効にし、中間結果をソースで定義された精度まで丸めます。
<code>-g</code>	-g オプションはオブジェクト・ファイルのサイズを大きくするシンボリック・デバッグ情報をオブジェクト・ファイルに生成するようにコンパイラに指示します。
<code>-traceback</code>	このオプションは、ランタイム時に致命的なエラーが発生したとき、ソースファイルのトレースバック情報を表示できるように、オブジェクト・ファイル内に補足情報を生成するようにコンパイラに指示します。 致命的なエラーが発生すると、コールスタックの 16 進アドレス (プログラム・カウンタ・トレース) とともに、ソースファイル、ルーチン名、および行番号の関連情報が表示されます。 マップファイルとエラーが発生したときに表示されるスタックの 16 進アドレスを使用することで、エラーの原因を特定できます。 このオプションを指定すると、実行プログラムのサイズが増えます。

### 4.3.2. コンパイルの推奨最適化オプション

コンパイルの推奨最適化オプションを以下に示します。本システムに搭載しているIntel Xeon E5-2680 v4は、Intel AVX2命令セットに対応していますので、-xCORE-AVX2オプションを指定することができます。-xCORE-AVX2を指定すると、コンパイラがソースコードを解析し、最適なAVX2、AVX、SSE命令を生成します。推奨最適化オプションは積極的な最適化を行い、かつ安全なオプションです。最適化のために計算の順序を変更する可能性があり、結果に誤差が生じる場合があります。

オプション	説明
<code>-O3</code>	O2 最適化を行い、融合、アンロールとジャムのブロック、IF 文の折りたたみなど、より強力なループ変換を有効にします。
<code>-xCORE-AVX2</code>	Intel プロセッサ向けのIntel アドバンスド・ベクトル・エクステンション 2 (Intel AVX2)、Intel AVX、SSE4.2、SSE4.1、SSSE3、SSE3、SSE2、SSE 命令を生成します。Intel AVX2 命令セット対応のIntel プロセッサ向けに最適化します。

上記のオプションを使用することにより、プログラムの性能が悪化した場合、最適化のレベルを-O2に下げるかベクトル化のオプションを変更してください。また、結果が一致していない場合、浮動小数点のオプションも試してみてください。

### 4.3.3. Intel 64アーキテクチャーのメモリモデル指定

次のいずれかのメモリモデルを使用して実行バイナリを作成します。

メモリモデル	説明
<code>small ( -mcmmodel=small )</code>	コードとデータのすべてのアクセスが、命令ポインター (IP) 相対アドレス指定で行われるように、コードとデータはアドレス空間の最初の 2GB までに制限されます。 -mcmmodelオプションを指定しない場合、デフォルトでこちらが指定されます。
<code>medium ( -mcmmodel=medium )</code>	コードはアドレス空間の最初の 2GB までに制限されますが、データは制限されません。コードは IP 相対アドレス指定でアクセスできますが、データのアクセスは絶対アドレス指定を使用する必要があります。
<code>large ( -mcmmodel=large )</code>	コードもデータも制限されません。コードもデータもアクセスは絶対アドレス指定を使用します。

IP 相対アドレス指定は 32 ビットのみ必要ですが、絶対アドレス指定は 64 ビット必要です。これは、コードサイズとパフォーマンスに影響します。(IP 相対アドレス指定の方が多少速くアクセスできます。)

プログラム内の共通ブロック、グローバルデータ、静的データの合計が2GBを越えるとき、リンク時に次のエラーメッセージが出力されます。

```
<some lib.a library>(some .o): In Function <function>:
: relocation truncated to fit: R_X86_64_PC32 <some symbol>
.....
: relocation truncated to fit: R_X86_64_PC32 <some symbol>
```

この場合は、`-mcmmodel=medium` と `-shared-intel` を指定してコンパイル/リンクして下さい。medium メモリモデルまたは large メモリモデルを指定した場合、Intel のランタイム・ライブラリの適切なダイナミック・バージョンが使用されるように、`-shared-intel` コンパイラ・オプションも指定する必要があります。

## 4.4. PGIコンパイラ

PGIコンパイラの各コマンドは以下のとおりです。

コマンド	言語	コマンド形式
<code>pgfortran</code>	Fortran 77/90/95	<code>\$ pgfortran [オプション] source_file</code>
<code>pgcc</code>	C	<code>\$ pgcc [オプション] source_file</code>
<code>pgc++</code>	C++	<code>\$ pgc++ [オプション] source_file</code>

PGIコンパイラにはLLVM版とno LLVM版の2種類があります。

LLVM版を使うには以下を実行します。

```
module load pgi
```

no LLVM版を使うには以下を実行します。

```
module load pgi-nollvm pgi
```

各コマンドの詳細は `$ man pgcc` 等でご確認下さい。

## 4.5. 並列化

### 4.5.1. スレッド並列(OpenMPと自動並列化)

OpenMP、自動並列化によるスレッド並列によりプログラムの高速化ができます。

OpenMP、自動並列化を使用する場合のコマンド形式を以下に示します。

言語	コマンド形式
OpenMP	
Fortran 77/90/95	<code>\$ ifort -qopenmp [オプション] source_file</code>
C	<code>\$ icc -qopenmp [オプション] source_file</code>
C++	<code>\$ icpc -qopenmp [オプション] source_file</code>
自動並列化	
Fortran 77/90/95	<code>\$ ifort -parallel [オプション] source_file</code>
C	<code>\$ icc -parallel [オプション] source_file</code>
C++	<code>\$ icpc -parallel [オプション] source_file</code>

`-qopt-report-phase=openmp` オプションを使用することでOpenMP最適化フェーズのレポートを作成することができます。

`-qopt-report-phase=par` オプションを使用することで自動並列化フェーズのレポートを作成することができます。

## 4.5.2. プロセス並列(MPI)

Fortran/C/C++ プログラムに MPI ライブラリをリンクし、プロセス並列プログラムを作成/実行することができます。MPIを使用する場合のコマンド形式を以下に示します。利用の際は、`module` コマンドで各MPIを読み込んでください。

MPIライブラリ	言語	コマンド形式
Intel MPI	Fortran 77/90/95	<code>\$ mpiifort [オプション] source_file</code>
	C	<code>\$ mpiicc [オプション] source_file</code>
	C++	<code>\$ mpiicpc [オプション] source_file</code>
Open MPI	Fortran 77/90/95	<code>\$ mpifort [オプション] source_file</code>
	C	<code>\$ mpicc [オプション] source_file</code>
	C++	<code>\$ mpicxx [オプション] source_file</code>
SGI MPT	Fortran 77/90/95	<code>\$ mpif90 [オプション] source_file</code>
	C	<code>\$ mpicc [オプション] source_file</code>
	C++	<code>\$ mpicxx [オプション] source_file</code>

## 4.6. GPU環境

本システムではGPU NVIDIA TESLA P100 の利用環境を提供しております。

### 4.6.1. インタラクティブジョブの実行・デバッグ

ログインノード `login`, `login0`, `login1` には、GPUを搭載しておらず、コンパイル、リンクのみ実行可能です。また、ログインノードにおける高負荷プログラムの実行は制限されています。

インタラクティブでの実行、デバッグについては、バッチシステムを使用して実行可能です。詳細については、[インタラクティブジョブの投入](#)を参照ください。



## 4.6.2. 対応アプリケーション

現在のGPU対応アプリケーションは次の通りです。(2017.12.18現在)

- ABAQUS 2017 --- ABAQUS利用の手引(別冊) を参照ください。
- NASTRAN 2017.1 --- NASTRAN利用の手引(別冊) を参照ください。
- ANSYS 18 --- ANSYS 利用の手引(別冊) を参照ください。
- AMBER 16 --- AMBER利用の手引(別冊) を参照ください。
- Maple 2016 --- Maple利用の手引(別冊) を参照ください。
- Mathematica 11.2 --- Mathematica利用の手引(別冊) を参照ください。
- MATLAB --- MATLAB利用の手引(別冊) を参照ください。
- Allinea Forge --- Allinea Forge利用の手引(別冊) を参照ください。
- PGI Compiler --- PGI コンパイラ 利用の手引(別冊) を参照ください。

他のアプリケーションにつきまても、順次展開してまいります。

## 4.6.3. CUDA 対応のMPI

CUDA版に対応したMPI環境を用意しております。

### OpenMPI + gcc環境

```
# CUDA, Open MPI環境の読み込み (gccは、デフォルトで設定されています。)
module load cuda openmpi
```

### OpenMPI + pgi環境

```
# CUDA, PGI環境の読み込み (最初にコンパイラ環境を読み込みます。)
module load cuda pgi
# Open MPI環境の読み込み (コンパイラに応じたOpenMPIの環境が設定されます。)
module load openmpi
```



以前記載されておりましたPGIバンドル版の `openmpi/2.1.2-pgi2017` のバージョン指定は現在は不要となっております。

### OpenMPI + Intel環境

```
# CUDA, Intel環境の読み込み (最初にコンパイラ環境を読み込みます。)
module load cuda intel
# Open MPI環境の読み込み (コンパイラに応じたOpenMPIの環境が設定されます。)
module load openmpi
```

## 4.6.4. NVIDIA GPUDirect

現在、NVIDIA GPUDirect GPUDIRECT FAMILY としては、4つの機能 GPUDIRECT SHARED GPU SYMMEM、GPUDIRECT P2P、GPUDIRECT RDMA、GPUDIRECT ASYNC があります。(2017.12.18現在)

このうち、TSUBAME3.0 では、GPUDIRECT SHARED GPU SYMMEM、GPUDIRECT P2P、GPUDIRECT RDMA をサポートしております。- GPUDIRECT SHARED GPU SYMMEM Version1

MPIの送受信バッファにCUDA pinnedメモリやデバイスメモリのアドレスを直接指定することができる機能です。デバイスメモリのアドレスを指定した場合には実際にはデータがホストメモリ上のバッファを経由して転送されます。

- GPUDIRECT P2P Version2

PCI-Express、NVLinkを経由したGPU間の直接データ転送(P2P)の機能です。TSUBAME 3.0では、ノードあたり、4GPUを搭載しておりますが、1つのCPUあたり、PLX switch を介して 2つのGPUに接続しております。4GPU間は、高速なNVLinkで接続されています。

- GPUDIRECT RDMA Version3

ホストメモリを介することなくGPUとインターコネクト間 TSUBAME3.0では、Intel Omni-Path で直接データ転送(RDMA)をすることにより異なるノードのGPU間の高速なデータ転送を実現する機能です。

- GPUDIRECT ASYNC

ホストメモリを介することなくGPUとインターコネクト間で非同期通信する機能です。現在、TSUBAME3.0の Intel Omni-Pathでは、未対応です。

参考 <http://on-demand.gputechconf.com/gtc/2017/presentation/s7128-davide-rossetti-how-to-enable.pdf>

GPUDirectについては、以下のURLも参照ください。

- <https://developer.nvidia.com/gpudirect>
- <http://docs.nvidia.com/cuda/gpudirect-rdma>

#### 4.6.5. GPUDirect RDMA

OPA10.9環境下でGPUDirect RDMAを実行したい場合、MPI\_Init()の前にcudaSetDevice()を呼ぶ必要があります。 [https://www.intel.com/content/dam/support/us/en/documents/network-and-i-o/fabric-products/Intel\\_PSM2\\_PG\\_H76473\\_v12\\_0.pdf](https://www.intel.com/content/dam/support/us/en/documents/network-and-i-o/fabric-products/Intel_PSM2_PG_H76473_v12_0.pdf) p.15

CUDA support is limited to using a single GPU per process.

You set up the CUDA runtime and pre-select a GPU card (through the use of cudaSetDevice() or a similar CUDA API) prior to calling psm2\_init() or MPI\_Init(), if using MPI.

While systems with a single GPU may not have this requirement, systems with multiple GPU may see non-deterministic results without proper initialization.

Therefore, it is strongly recommended that you initialize the CUDA runtime before the psm2\_init() or MPI\_Init() call.

ご自身のコード内で上記修正を行うか、上記をopenmpi内で行うように修正されたopenmpi/3.1.4-opa10.10-t3がTSUBAMEにはインストールされております。 `module load cuda openmpi/3.1.4-opa10.10-t3` でご利用が可能です。

OpenMPIでのGPUDirect RDMAの実行方法を以下に示します。以下、2ノード、MPI×2での実行例になります。

```
# module load cuda openmpi/3.1.4-opa10.10-t3
# mpirun -np 2 -npernode 1 -x PSM2_CUDA=1 -x PSM2_GPUDIRECT=1 -x LD_LIBRARY_PATH -x PATH [プログラム]
```

- PSM2\_CUDA --- Omni-PathでのCUDA有効
- PSM2\_GPUDIRECT --- GPUDirect RDMA有効

#### 4.6.6. GPUのCOMPUTE MODEの変更

資源タイプFのf\_nodeを利用した場合、GPUのCOMPUTE MODEを変更することが出来ます。GPUのCOMPUTE MODEを変更するには、f\_nodeを指定した上で、ジョブスクリプトの中で、 `#$ -v GPU_COMPUTE_MODE=<利用するモード>` を指定してください。利用可能なモードは以下の3つです。

モード	説明
0	DEFAULTモード 1つのGPUを複数のプロセスから同時に利用できる。
1	EXCLUSIVE_PROCESSモード 1つのGPUを1プロセスのみが利用できる。1プロセスから複数スレッドの利用は可能。
2	PROHIBITEDモード GPUへのプロセス割り当てを禁止する。

以下はスクリプトの例となります。

```
#!/bin/sh
#$ -cwd
#$ -l f_node=1
#$ -l h_rt=1:00:00
#$ -N gpumode
#$ -v GPU_COMPUTE_MODE=1
/usr/bin/nvidia-smi
```

インタラクティブで利用する場合、qrshは以下のような形となります。

```
$ qrsh -g [TSUBAMEグループ] -l f_node=1 -l h_rt=0:10:00 -pty yes -v TERM -v GPU_COMPUTE_MODE=1 /bin/bash
```

## 5. ジョブスケジューリングシステム

本システムのジョブスケジューリングには、シングルジョブ・並列ジョブを優先度や必要なリソースに従い効率的にスケジューリングする、「UNIVA Grid Engine」を採用しています。

### 5.1. 計算ノードの種類

#### 5.1.1. ベアメタル環境

##### 5.1.1.1. 利用可能な資源タイプ

本システムでは計算ノードを論理的に分割した資源タイプを利用して、システムリソースを確保します。

ジョブ投入の際には、資源タイプをいくつ使うかを指定します(例 `-l f_node=2`)。利用できる資源タイプの一覧を以下に示します。

資源タイプ	資源タイプ名	使用物理CPUコア数	メモリ (GB)	GPU 数
F	f_node	28	235	4
H	h_node	14	120	2
Q	q_node	7	60	1
C1	s_core	1	7.5	0
C4	q_core	4	30	0
G1	s_gpu	2	15	1

- ・「使用物理CPUコア数」、「メモリ(GB)」、「GPU数」は、各資源タイプ1つあたりの使用可能な量です。
- ・「資源タイプ名」=[回数]で同じ資源タイプを複数指定できます。資源タイプの組み合わせはできません。
- ・実行可能時間の最大値は24時間です。
- ・TSUBAME3では「同時に実行可能なジョブ数」や「実行可能な総スロット数」など各種制限値があります。(スロット=資源タイプ毎に設定されている物理CPUコア数x利用ノード数(qstatコマンドのslotsと同等))

現在の制限値の一覧は以下のURLで確認できます。

<https://www.t3.gsic.titech.ac.jp/resource-limit>

利用状況に応じて随時変更する可能性がありますのでご注意ください。

#### 5.1.2. コンテナ環境

本システムでは、ソフトウェアの依存関係によりホストOSで動作させることが困難なアプリケーションを利用可能とするために、Dockerを利用したシステムコンテナとSingularityを利用したアプリケーションコンテナを提供しています。ここでは、Dockerを利用したシステムコンテナのジョブの利用方法を記載します。Singularityについてはフリーウェアの章を参照ください。本節ではコンテナを使わずにノードを直接利用する実行方法を「ノード利用」と記載します。

### 5.1.2.1. 利用可能な資源タイプ

コンテナ利用のジョブで利用できる資源タイプは以下のとおりです。バッチスクリプトで利用する際は末尾に.mpiをつけることで複数ノードの利用ができます。インタラクティブジョブでは.mpi付きの資源タイプは指定できません。

資源タイプ	ノード利用	コンテナ利用
F	f_node	t3_d_f_node / t3_d_f_node.mpi
H	h_node	t3_d_h_node / t3_d_h_node.mpi
Q	q_node	t3_d_q_node / t3_d_q_node.mpi
C1	s_core	t3_d_s_core (下記参照)
C4	q_core	t3_d_q_core / t3_d_q_core.mpi
G1	s_gpu	t3_d_s_gpu / t3_d_s_gpu.mpi

資源タイプC1の t3\_d\_s\_coreでは外部への通信は可能ですが、コンテナ間通信をサポートしていません。このため、MPIやマルチコンテナの通信を行う場合は他のコンテナ資源をご指定ください。



-t 1-1 のように1コンテナしか起動しない場合には .mpi なしの資源タイプを指定してください。 .mpi 付きの資源タイプで -t 1-1 とすると正しく動作しません。

ノード利用時とコンテナ利用時のqsubコマンドのオプションを以下に示します。

	ノード利用	コンテナ利用
イメージ指定		-ac d=[イメージ名]
資源タイプ指定	-l [資源タイプ名]=[個数]	-jc [コンテナ資源名] -t 1-[個数]
Walltime指定	-l h_rt=[経過時間]	-adds l_hard h_rt [経過時間]

利用可能なイメージは本システムで公開しているイメージのみとなります。利用可能なイメージについてはTSUBAME計算サービスWebページの[システムソフトウェア](#)をご確認下さい。

## 5.2. ジョブの投入

本システムでジョブを実行するには、ログインノードへログインしてqsubコマンドを実行します。

### 5.2.1. バッチジョブの流れ

ジョブを投入するためにはジョブスクリプトを作成し投入します。または、コマンドラインにキュー名などを指定してジョブを投入することもできます。投入コマンドは"qsub"です。

- ・ジョブスクリプトの作成
- ・qsubを使用しジョブを投入
- ・qstatなどを使用しジョブの状態確認
- ・必要に応じてqdelを使用しジョブのキャンセル
- ・ジョブの結果確認

qsubコマンドは、課金情報(TSUBAME3ポイント)を確認し、ジョブを受け付けます。

## 5.2.2. ジョブスクリプト

ジョブスクリプトの記述方法を以下に示します。

```
#!/bin/sh
#$ -cwd
#$ -l [資源タイプ名] =[個数]
#$ -l h_rt=[経過時間]
#$ -p [プライオリティ]

[moduleの初期化]

[プログラミング環境のロード]

[プログラム実行]
```



### Warning

shebang(#!/bin/shの箇所)は必ずジョブスクリプトの先頭に来るようにして下さい。

#### • [moduleの初期化]

以下を実行し、moduleコマンドの初期化を行います。

```
. /etc/profile.d/modules.sh
```

#### • [プログラミング環境のロード]

moduleコマンドを用い、必要な環境のロードを行います。  
intelコンパイラをロードする場合の例は以下となります。

```
module load intel
```

#### • [プログラム実行]

プログラムの実行を行います。  
バイナリがa.outの場合の例は以下となります。

```
./a.out
```

資源タイプの指定などはコマンドラインで指定するか、またはスクリプトファイルの最初のコメントブロック(##\$)に記述することで有効になります。資源タイプ、実行時間は必須項目になるため必ず指定するようにしてください。

qsubコマンドの主なオプションを以下に示します。

オプション	説明
-l [資源タイプ名]=[個数] (必須)	資源タイプおよびその個数を指定します。
-l h_rt=[経過時間] (必須)	Wall time(経過時間)を指定します。[[HH:]MM:]SSで指定することができます。HH:MM:SやMM:SSやSSのように指定することができます。
-N	ジョブ名を指定します。(指定しない場合はスクリプトファイル名)
-o	標準出力ファイル名を指定します。
-e	標準エラー出力ファイル名を指定します。
-j y	標準エラー出力を標準出力ファイルに統合します。
-m	ジョブについての情報をメールで送信する条件を指定します。 a バッチシステムによりジョブが中止された場合 b ジョブの実行が開始された場合 e ジョブの実行が終了した場合 abeのように組み合わせることも可能です。 メールオプションをつけて大量のジョブを投入すると、大量のメールによってメールサーバーに負荷が掛かり、攻撃と検知され他の利用者もまとめて東工大からのメールを遮断される可能性があります。そのようなジョブを流す必要がある場合は、メールオプションを外すか一度のジョブで実行できるようスクリプトの見直しを行ってください。
-M	送信先メールアドレスを指定します。
-p プレミアムオプション)	ジョブの実行優先度を指定します。-3,-4を指定すると通常よりも高い課金係数が適用されます。設定値の-5,-4,-3は課金規則の優先度0,1,2に対応します。 -5: 標準の実行優先度です。(デフォルト) -4: 実行優先度は-5より高く,-3より低くなります。 -3: 最高の実行優先度となります。 優先度の値は全て負の数です。マイナス記号を含めて指定してください。
-t	タスクIDの範囲を指定します。 開始番号-終了番号[:ステップサイズ] で指定することができます。
-hold_jid	依存関係にあるジョブIDを指定します。 指定された依存ジョブが終了しなければ、発行ジョブは実行されません。
-ar	予約ノードを利用する際に 予約AR IDを指定します。

本システムではジョブ投入環境の環境変数渡し-Vオプションは利用できません。ご注意ください。

## 5.2.3. ジョブスクリプトの記述例

### 5.2.3.1. シングルジョブ/GPUジョブ

シングルジョブ(並列化されていないジョブ)を実行する時に作成するバッチスクリプトの例を以下に示します。GPUを使用するジョブの場合は `-l s_core=1` を `-l s_gpu=1` に変更し、GPUで利用するmoduleの読み込み以外はシングルジョブと同様になります。

```
#!/bin/sh
# カレントディレクトリでジョブを実行する場合に指定
#$ -cwd

#$ -l s_core=1
# 実行時間を指定
#$ -l h_rt=1:00:00
#$ -N serial

# Moduleコマンドの初期化
. /etc/profile.d/modules.sh
# CUDA環境の読込
module load cuda
# Intel Compiler環境の読込
```

```
module load intel
./a.out
```

### 5.2.3.2. SMP並列

SMP並列ジョブを実行する時に作成するバッチスクリプトの例を以下に示します。計算ノードはハイパースレッディングが有効になっています。使用するスレッド数につきましては、明示的に指定してください。

```
#!/bin/sh
#$ -cwd
# 資源タイプF 1ノードを使用
#$ -l f_node=1
#$ -l h_rt=1:00:00
#$ -N openmp
. /etc/profile.d/modules.sh
module load cuda
module load intel
# ノード内に28スレッドを配置
export OMP_NUM_THREADS=28
./a.out
```

### 5.2.3.3. MPI並列

MPI並列ジョブを実行する時に作成するバッチスクリプトの例を以下に示します。使用するMPI環境により使い分けをお願いします。OpenMPIでスレーブノードにライブラリ環境変数を渡すには、`-x LD_LIBRARY_PATH` を利用する必要があります。

#### Intel MPI環境

```
#!/bin/sh
#$ -cwd
# 資源タイプF 4ノードを使用
#$ -l f_node=4
#$ -l h_rt=1:00:00
#$ -N flatmpi
. /etc/profile.d/modules.sh
module load cuda
module load intel
# Intel MPI環境の読み込み
module load intel-mpi
# ノードあたり8プロセスMPI全32 プロセスを使用
mpirun -np 32 ./a.out
```

#### OpenMPI環境

```
#!/bin/sh
#$ -cwd
# 資源タイプF 4ノードを使用
#$ -l f_node=4
#$ -l h_rt=1:00:00
#$ -N flatmpi
. /etc/profile.d/modules.sh
module load cuda
module load intel
# Open MPI環境の読み込み
module load openmpi
# ノードあたり8プロセスMPI全32 プロセスを使用
mpirun -np 32 -x LD_LIBRARY_PATH ./a.out
```

#### SGI MPT環境

```
#!/bin/sh
#$ -cwd
# 資源タイプF 4ノードを使用
#$ -l f_node=4
#$ -l h_rt=1:00:00
#$ -N flatmpi
. /etc/profile.d/modules.sh
module load cuda
module load intel
# SGI MPT環境の読み込み
module load mpt
# ノードあたり8プロセスMPI全32 プロセスを使用
mpirun -np 32 ./a.out
```





SGI MPTのmpixexec\_mptではa.outへのパスが通っていない場合にはフルパスを指定する必要があります。パスが通っている場合にはmpixexec\_mpt ... a.outで実行できます。

※ 投入したジョブに対して割り当てられているノードリストは、`PE_HOSTFILE` 変数で参照できます。

```
$ echo $PE_HOSTFILE
/var/spool/uge/r6i0n4/active_jobs/4564.1/pe_hostfile
$ cat /var/spool/uge/r6i0n4/active_jobs/4564.1/pe_hostfile
r6i0n4 28 all.q@r6i0n4 <NULL>
r6i3n5 28 all.q@r6i3n5 <NULL>
```

#### 5.2.3.4. プロセス並列/スレッド並列(ハイブリッド, MPI+OpenMP)

プロセス並列/スレッド並列(ハイブリッド, MPI+OpenMP)のジョブを実行する時に作成するバッチスクリプトの例を以下に示します。使用するMPI環境により使い分けをお願いします。OpenMPIでスレーブノードにライブラリ環境変数を渡すには、`-x LD_LIBRARY_PATH` を利用する必要があります。

##### Intel MPI環境

```
#!/bin/sh
#$ -cwd
# 資源タイプ 4ノードを使用
#$ -l f_node=4
#$ -l h_rt=1:00:00
#$ -N hybrid
. /etc/profile.d/modules.sh
module load cuda
module load intel
module load intel-mpi
# ノード内に28スレッドを配置
export OMP_NUM_THREADS=28
# ノードあたりMPI 1プロセス、全4 プロセスを使用
mpixexec.hydra -ppn 1 -n 4 ./a.out
```

##### OpenMPI環境

```
#!/bin/sh
#$ -cwd
# 資源タイプ 4ノードを使用
#$ -l f_node=4
#$ -l h_rt=1:00:00
#$ -N hybrid
. /etc/profile.d/modules.sh
module load cuda
module load intel
module load openmpi
# ノード内に28スレッドを配置
export OMP_NUM_THREADS=28
# ノードあたりMPI 1プロセス、全4 プロセスを使用
mpirun -npnnode 1 -n 4 -x LD_LIBRARY_PATH ./a.out
```

#### 5.2.3.5. コンテナの利用

ジョブスクリプトの記述方法を以下に示します。資源タイプや経過時間の指定方法が通常の利用の場合と異なりますので、ご注意ください。

```
#!/bin/sh
#$ -cwd
#$ -ac [コンテナイメージ名]
#$ -jc [コンテナ資源名]
#$ -t 1-[個数]
#$ -adds l_hard h_rt=[経過時間]
[moduleの初期化]
[プログラミング環境のロード]
[プログラム実行]
```

コンテナジョブを実行する時に作成するバッチスクリプトの例を以下に示します。GPUの使用やMPI並列ジョブの利用方法は通常の利用方法と同様です。

```
#!/bin/sh
#$ -cwd

#$ -ac d=sles12sp2-latest          # コンテナイメージとしてSLES12SP2を指定
#$ -jc t3_d_q_node.mpi            # 資源タイプQ を指定
#$ -t 1-4                          # コンテナ数 4を指定
#$ -adds l_hard h_rt 0:10:00      # 実行時間を指定

. /etc/profile.d/modules.sh
module load cuda
module load intel
module load openmpi
mpirun -npnode 6 -n 24 -hostfile $SGE_JOB_SPOOL_DIR/ompi_hostfile -x LD_LIBRARY_PATH ./a.out
```



`-t 1-1` のように1コンテナしか起動しない場合には `.mpi` なしの資源タイプを指定してください。 `.mpi` 付きの資源タイプで `-t 1-1` とすると正しく動作しません。

## 5.2.4. ジョブの投入

ジョブを実行するために、バッチリクエストを事前に作成する必要があります。qsubコマンドにジョブ投入スクリプトを指定することで、ジョブがキューイングされ実行されます。qsubコマンドを使用してジョブを投入する例を以下に示します。

```
$ qsub -g [TSUBAMEグループ] スクリプト名
```

オプション	説明
-g	TSUBAMEグループ名を指定します。 スクリプトの中ではなくqsubコマンドのオプションとしてつけてください。

### 5.2.4.1. お試し実行



本機能は既にアカウントをお持ちの方(おもに自由にアカウント作成ができる学内利用者)向けの機能です。

TSUBAMEを自分の研究に利用できるか不安のある利用者がポイント購入をする前に動作確認できるよう、TSUBAMEではポイントを消費することなくプログラムの動作確認を行うことができるお試し実行機能が用意されています。

ジョブ投入時に `-g` オプションでグループを指定しないことで、ジョブをお試し実行として投入することができます。この際、2並列以下、実行時間10分以下、優先度-5(最低) という制限がかかります。



お試し実行は課金前のプログラムの動作確認を目的とした利用に限り、実際の研究や計測を目的とした実行は行わないでください。  
制限内でも無償で無制限に利用してよいというわけではありません。

お試し実行機能はTSUBAMEを自分の研究に利用できるか不安のある利用者がポイント購入をする前に動作確認できるように用意されたものですので、これらの目的から大きく逸脱する、直接研究成果につながるような計算はお試し実行機能で行わないようお願いします。  
授業において、教育目的の小規模な計算を実行したい場合は[インタラクティブキュー](#)をご利用することもご検討ください。

お試し実行の場合、資源量に以下の制限が適用されます。

利用可能な最大ノード数(資源数)(*1)	2
利用最長時間	10分
同時実行数	1
資源タイプ	制限なし

(\*1): [Dockerコンテナ利用](#)の場合は1

また、お試し実行には「TSUBAMEグループ」を指定しないで実行する必要があります。  
TSUBAMEグループを指定した場合=-gオプションを利用した場合はポイントが消費されますのでご注意ください。

5.2.5. ジョブの状態確認

qstatコマンドはジョブ状態表示コマンドです。

```
$ qstat [オプション]
```

qstatコマンドの主なオプションを以下に示します。

表 5-5 qstatコマンドのオプション

オプション	説明
-r	ジョブのリソース情報を表示します。
-j [ジョブID]	ジョブに関する追加情報を表示します。

qstatコマンドの実行結果の例を以下に示します。

```
$ qstat
job-IDprior  nameuser  statesubmit/start at  queuejclass  slotsja-task-ID
-----
307 0.55500 sample.sh testuser r 02/12/2015 17:48:10 all.q@r8i6n1A.default32
(以下省略)
```

qstat コマンドの表示内容を以下に示します。

表示項目	説明
Job-ID	ジョブIDを表示します。
prior	優先度を表示します。
name	ジョブ名を表示します。
user	ジョブのオーナーを表示します。
state	ジョブのステータスを表示します。 r 実行中 qw 待機中 h ホールド中 d 削除中 t 移動中 s サスペンド状態、一時停止 S サスペンド状態、キューのサスペンド状態 T サスペンド状態、制限超過によるサスペンド E エラー状態 Rq 再スケジューリングされ待機中のジョブ Rr 再スケジューリングされ実行中のジョブ
submit/start at	投入/開始日時を表示します。
queue	キュー名を表示します。
jclass	ジョブクラス名を表示します。
slots	利用しているスロット数を表示します。 (スロット=資源タイプ毎に設定されている物理CPUコア数x利用ノード数)
ja-task-ID	アレイジョブに関してタスクIDを表示します。

## 5.2.6. ジョブの削除

バッチジョブを削除する場合には、`qdel` コマンドを使用します。

```
$ qdel [ジョブID]
```

以下にジョブをqdelした結果を示します。

```
$ qstat
job-IDprior  nameuser      statesubmit/start at  queuejclass  slotsja-task-ID
-----
307 0.55500 sample.sh testuser r 02/12/2015 17:48:10 all.q@r8i6n1A.default32

$ qdel 307
testuser has registered the job 307 for deletion

$ qstat
job-IDprior  nameuser      statesubmit/start at  queuejclass  slotsja-task-ID
-----
```

## 5.2.7. ジョブの結果確認

UGEのジョブの標準出力はジョブを実行したディレクトリの「スクリプトファイル名.oジョブID」というファイルに保管されます。また、標準エラー出力は「スクリプトファイル名.eジョブID」です。

### 5.2.8. アレイジョブ

ジョブスクリプト内に含まれる操作をパラメータ化して繰り返し実行する機能としてアレイジョブ機能があります。アレイジョブで実行される各ジョブをタスクと呼び、タスクIDによって管理されます。またタスクIDを指定しないジョブIDは、タスクID全部を範囲とします。



アレイジョブの各タスクはそれぞれ別のジョブとしてスケジュールされるため、タスクの数に比例したスケジュールの待ち時間が発生します。各タスクの処理が短い場合や、タスク数が多い場合は、複数のタスクをまとめてタスクの数を減らすことを強くお勧めいたします。  
例: 10000タスクを、それぞれ100タスク分の処理をするタスク100個にまとめる

タスク番号の指定は、qsubコマンドのオプションもしくはジョブスクリプト内で定義します。投入オプションは `-t (開始番号)-(終了番号):(ステップサイズ)` として指定します。ステップサイズが1の場合は省略可能です。以下に例を示します。

```
# ジョブスクリプト内にて以下を指定
#$ -t 2-10:2
```

上記例 `2-10:2` では、開始番号 2、終了番号 10、ステップサイズ2 (1つ飛ばしのインデックス)が指定され、タスク番号 2、4、6、8、10 の5つのタスクによって構成されます。

各タスクのタスク番号は `$SGE_TASK_ID` という環境変数に設定されるため、この環境変数をジョブスクリプト内で利用することで、パラメータスタディが可能となります。結果ファイルはジョブ名の後ろにタスクIDが付いた形で出力されます。

また、実行前/実行中に特定のタスクIDを削除したい場合には、以下のように `qdel` の `-t` オプションを使用します。

```
$ qdel [ジョブID] -t [タスクID]
```

## 5.3. 計算ノードの予約

計算ノードを予約することにより、24時間および72ノードを越えるジョブの実行が可能です。予約実行の流れは以下のようになります。

- TSUBAMEポータルから予約の実行
- TSUBAMEポータルから予約状況の確認、キャンセル
- 予約ノードに対してqsubを使用しジョブを投入
- 必要に応じてqdelを使用しジョブのキャンセル
- ジョブの結果確認
- コマンドラインからの予約状況およびAR IDの確認

ポータルからの予約の実行、予約状況の確認、予約のキャンセルに関して[TSUBAMEポータル利用の手引き 計算ノードの予約](#)をご参照ください。

予約時間になりましたら、予約グループのアカウントでジョブの実行ができるようになります。予約IDであるAR IDを指定したジョブ投入の例を以下に示します。

- qsubで予約ノードにジョブを投入する場合

```
$ qsub -g [TSUBAMEグループ] -ar [AR ID] スクリプト名
```

- qrshで予約ノードにインタラクティブジョブを投入する場合

```
$ qrsh -g [TSUBAMEグループ] -l [資源タイプ]=[個数] -l h_rt=[時間] -ar [AR ID]
```

予約実行で利用できる資源タイプはf\_node,h\_node,q\_nodeになります。q\_core,s\_core,s\_gpuは利用できません。

ジョブ投入後のジョブの状態確認はqstatコマンド、ジョブの削除はqdelコマンドを使用します。

また、スクリプトの書式は通常実行時のものと同じになります。

コマンドラインから予約状況及びAR IDを確認するためには `t3-user-info compute ar` を使用します。

xxxxx@login0:~> t3-user-info compute ar											
ar_id	uid	user_name	gid	group_name	state	start_date	end_date	time_hour	node_count	point	return_point
1320	2005	A2901247	2015	tga-red000	r	2018-01-29 12:00:00	2018-01-29 13:00:00	1	1	18000	0
1321	2005	A2901247	2015	tga-red000	r	2018-01-29 13:00:00	2018-01-29 14:00:00	1	1	18000	0
1322	2005	A2901247	2015	tga-red000	w	2018-01-29 14:00:00	2018-02-02 14:00:00	96	1	1728000	1728000
1323	2005	A2901247	2015	tga-red000	r	2018-01-29 14:00:00	2018-02-02 14:00:00	96	1	1728000	1728000
1324	2005	A2901247	2015	tga-red000	r	2018-01-29 15:00:00	2018-01-29 16:00:00	1	17	306000	0
1341	2005	A2901247	2015	tga-red000	w	2018-02-25 12:00:00	2018-02-25 13:00:00	1	18	162000	162000
3112	2004	A2901239	2349	tgz-training	r	2018-04-24 12:00:00	2018-04-24 18:00:00	6	20	540000	0
3113	2004	A2901239	2349	tgz-training	r	2018-04-25 12:00:00	2018-04-25 18:00:00	6	20	540000	0
3116	2005	A2901247	2015	tga-red000	r	2018-04-18 17:00:00	2018-04-25 16:00:00	167	1	3006000	0
3122	2005	A2901247	2014	tga-blue000	r	2018-04-25 08:00:00	2018-05-02 08:00:00	168	5	15120000	0
3123	2005	A2901247	2014	tga-blue000	r	2018-05-02 08:00:00	2018-05-09 08:00:00	168	5	3780000	0
3301	2005	A2901247	2015	tga-red000	r	2018-08-30 14:00:00	2018-08-31 18:00:00	28	1	504000	0
3302	2005	A2901247	2009	tga-green000	r	2018-08-30 14:00:00	2018-08-31 18:00:00	28	1	504000	0
3304	2005	A2901247	2014	tga-blue000	r	2018-09-03 10:00:00	2018-09-04 10:00:00	24	1	432000	0
3470	2005	A2901247	2014	tga-blue000	w	2018-11-11 22:00:00	2018-11-11 23:00:00	1	1	4500	4500
4148	2004	A2901239	2007	tga-hpe_group00	w	2019-04-12 17:00:00	2019-04-12 18:00:00	1	1	4500	4500
4149	2005	A2901247	2015	tga-red000	w	2019-04-12 17:00:00	2019-04-13 17:00:00	24	1	108000	108000
4150	2004	A2901239	2007	tga-hpe_group00	w	2019-04-12 17:00:00	2019-04-12 18:00:00	1	1	4500	4500
total :								818	97	28507500	3739500

コマンドラインから当月の予約の空き状況を確認するには、 `t3-user-info compute ars` を使用します。

## 5.4. インタラクティブジョブの投入

本システムのジョブスケジューラでは、インタラクティブにプログラムやシェルスクリプトを実行する機能を有しています。

インタラクティブジョブを実行するためには、`qrsh`コマンドを使用し、`-l`で資源タイプ、経過時間を指定します。

`qrsh`でジョブ投入後、ジョブがディスパッチされるとコマンドプロンプトが返ってきます。

インタラクティブジョブの使用方法的流れ以下に示します。

```
#!/bash
$ qrsh -g [TSUBAMEグループ] -l [資源タイプ]=[個数] -l h_rt=[経過時間]
Directory: /home/N/username
(ジョブ開始時刻)
username@rXiXnX:~> [計算ノードで実行したいコマンド]
username@rXiXnX:~> exit
```

`-g`オプションのグループ指定が未指定の場合は資源タイプ2つまで、経過時間10分間まで、優先度-5の「お試し実行」となります。

資源タイプF、1ノード、経過時間 10分を指定した例

```
#!/bash
$ qrsh -g [TSUBAMEグループ] -l f_node=1 -l h_rt=0:10:00
Directory: /home/N/username
(ジョブ開始時刻)
username@rXiXnX:~> [計算ノードで実行したいコマンド]
username@rXiXnX:~> exit
```

プロンプトに`exit`と入力することでインタラクティブジョブを終了します。

インタラクティブジョブでコンテナを利用する使用方法を以下に示します。インタラクティブジョブでは複数コンテナの指定はできません。

```
$ qrsh -g [TSUBAMEグループ] -jc [コンテナ資源名] -adds l_hard h_rt [経過時間] -ac [イメージ名]
```

資源タイプQ、実行時間10分でSLES12SP2のコンテナを指定する場合以下ようになります。

```
$ qrsh -g tga-hpe_group00 -jc t3_d_q_node -adds l_hard h_rt 0:10:00 -ac d=sles12sp2-latest
```

### 5.4.1. インタラクティブノードを利用したX転送

qrshで接続したノードから直接X転送を行う場合は、下記の手順にて接続ください。

1. X転送を有効にしてログインノードにssh
2. qrshコマンドの実行



2020年4月に実施したスケジューラの更新に伴い、qrsh 実行時に `-pty yes -display "$DISPLAY" -v TERM /bin/bash` を明示的に指定する必要がなくなりました。

#### コマンド実行例

例では資源タイプs\_core、1ノードで2時間のジョブを実行しています。

割り当てノードはコマンド実行時に空いているノードですので、明示的にノードを指定することはできません。

```
# qrshの実行
$ qrsh -g [TSUBAMEグループ] -l s_core=1 -l h_rt=2:00:00
username@rXiXnX:-> module load [読み込みたいアプリケーション]
username@rXiXnX:-> [実行したいアプリケーションの実行コマンド]
username@rXiXnX:-> exit
```

コンテナ資源タイプt3\_d\_s\_coreを使用したインタラクティブジョブのX転送の例は以下になります。

```
$ qrsh -g [TSUBAMEグループ] -jc t3_d_s_core -adds l_hard h_rt 0:10:00 -ac d=sles12sp2-latest
```

### 5.4.2. ネットワーク系アプリケーションへの接続

コンテナによるインタラクティブジョブにおいて、Webブラウザ等でアプリケーションを操作する必要がある場合、SSHポートフォワードを用いて手元のWebブラウザからアクセスすることが可能です。

- (1) qrshで接続したインタラクティブノードのホスト名の取得

```
$ qrsh -g tga-hpe_group00 -jc t3_d_q_node -adds l_hard h_rt 0:10:00 -ac d=sles12sp2-latest
$ hostname
r7i7n7-cnnode00
$ [Webブラウザ等からのアクセスが必要なプログラムの実行]
```

qrshでインタラクティブジョブを起動後、そのマシンのホスト名を取得します。上記の例では、ホスト名としてr7i7n7-cnnode00がホスト名になります。この、コンソールでの作業はおわりですが、アプリケーションによる作業が終了するまで、そのまま維持してください。

- (2) ssh接続元のコンソールよりSSHのポートフォワードを有効にして接続する。(ログインノードやインタラクティブジョブ上ではありません)

```
ssh -l username -L 8888:r7i7n7-cnnode00:<接続するアプリケーションのネットワークポート> login.t3.gsic.titech.ac.jp
```

接続するアプリケーションのネットワークポートは、アプリケーションごとに異なります。詳しくは、各アプリケーションの説明書もしくは、アプリケーションの起動メッセージをご確認ください。



TSUBAME3にSSHするコンソールによっては、SSHのポートフォワードの設定が異なります。詳しくは、各SSHコンソールの説明をご確認いただくか、FAQをご参照ください。

- (3) Webブラウザでアプリケーションに接続する。手元のコンソール上でWebブラウザ (Microsoft Edge, Firefox, Safari等) を立ち上げ、`http://localhost:8888/` にアクセスしてください。

### 5.4.3. インタラクティブキュー

インタラクティブキューは、同じリソースをユーザ間で共有することで、混雑時にもノード確保に失敗しにくく、可視化や対話的なプログラムを素早く開始できるように用意された環境です。

インタラクティブキューへのジョブの投入方法は以下のとおりです。



学内ユーザ(tgz-edu)とアクセスカードユーザ(ログイン名がAxxxのユーザ)に限り、グループの指定を省略して無償で実行できます。

```
iqcrsh -g [TSUBAMEグループ] -l h_rt=<時間>
```

※インタラクティブキューではCPU/GPUのオーバーコミットが許可されていることにご注意下さい。

インタラクティブキューの制限値一覧は[こちら](#)をご確認下さい。

## 5.5. 計算ノードへのSSHログイン

資源タイプf\_nodeでジョブを行ったノードには直接sshでログインできます。

確保したノードは以下の手順により、確認することができます。

```
$ qstat -j 1463
=====
job_number:          1463
(途中省略)
exec_host_list       1:   r8i6n3:28, r8i6n4:28      ← 確保したノード  r8i6n3、r8i6n4
(以降省略)
```

確保したコンテナには直接sshでログインできます。確保したコンテナのホスト名は下記に例示する手順により、確認することができます。

```
$ qstat -j (ジョブID)
(中略)
binding              3:   r7i7n7=1,0:1,1
binding              4:   r7i7n7=1,7:1,8
resource map         1:   hostipv4=r7i7n7=(r7i7n7-cnnode00), s_gpu=r7i7n7=(0)
resource map         2:   hostipv4=r7i7n7=(r7i7n7-cnnode01), s_gpu=r7i7n7=(1)
resource map         3:   hostipv4=r7i7n7=(r7i7n7-cnnode02), s_gpu=r7i7n7=(2)
resource map         4:   hostipv4=r7i7n7=(r7i7n7-cnnode03), s_gpu=r7i7n7=(3)
scheduling info:      (Collecting of scheduler job information is turned off)
```

上記例の場合、resource map行の括弧内にある r7i7n7-cnnode00, r7i7n7-cnnode01, r7i7n7-cnnode02, r7i7n7-cnnode03 が確保されたコンテナのホスト名となります。



計算ノードにsshした際はsshしたプロセスのGIDがtsubame-users(2000)となっている為、お試し実行の場合を除いて、ssh直後の状態では実行している自分のジョブのプロセスが見えず、gdbでアタッチもできません。

見えるようにするにはssh後にジョブを実行したグループ名で以下を実行して下さい。

```
newgrp <グループ名>
```

又は

```
sg <グループ名>
```



## 5.6. 計算ノード上のストレージの利用

### 5.6.1. ローカルスクラッチ領域

SSDをローカルスクラッチ領域として使用することができます。利用する際には、\$TMPDIR および \$T3TMPDIR にローカルスクラッチ領域のパスが設定されます。ジョブスクリプトの中で、作業領域のパスを指定することにより参照可能です。

ローカルスクラッチ領域は各計算ノードの個別領域となり共有されていないため、ジョブスクリプト内からの入力ファイルと出力ファイルをローカルホストにステージングする必要があります。下記の例では、使用する計算ノードが1ノードの場合に、ホームディレクトリからローカルスクラッチ領域にインプットデータセットをコピーし、結果をホームディレクトリに返します。(複数ノードは対応していません) \$TMPDIRは各MPIプロセス終了時に削除されるため、複数のMPIプロセスを1ジョブで利用し、ローカルスクラッチの内容を引き継ぎたい場合は \$T3TMPDIR をご利用ください。

```
#!/bin/sh
# 計算に必要な入力ファイルのコピー
cp -rp $HOME/datasets $TMPDIR/
# 入力、出力を指定する計算プログラムの実行
./a.out $TMPDIR/datasets $TMPDIR/results
# 必要な結果ファイルのコピー
cp -rp $TMPDIR/results $HOME/results
```

### 5.6.2. 共有スクラッチ領域

資源タイプF(`f_node`)を利用したバッチスクリプトの場合のみ、確保した複数の計算ノードのSSDをオンデマンドに共有ファイルシステムとして作成するBeeGFS On Demand(BeeOND)を利用できます。BeeONDを有効にするには、ジョブスクリプトの中で、`f_node` を指定した上で、`#$ -v USE_BEEOND=1` を指定してください。BeeONDは計算ノード上の/beeondにマウントされます。以下はスクリプトの例となります。

```
#!/bin/sh
#$ -cwd
#$ -l f_node=4
#$ -l h_rt=1:00:00
#$ -N flatmpi
#$ -v USE_BEEOND=1
. /etc/profile.d/modules.sh
module load cuda
module load intel
module load intel-mpi
mpiexec.hydra -ppn 8 -n 32 ./a.out
```

インタラクティブで利用する場合、qrshは以下のような形となります。利用しない場合と比べ、ディスクのマウント処理に少し時間を要します。

```
$ qrsh -g [TSUBAMEグループ] -l f_node=2 -l h_rt=0:10:00 -pty yes -v TERM -v USE_BEEOND=1 /bin/bash
```

BeeOND共有スクラッチ領域はジョブで確保されたタイミングで作成されるため、ジョブスクリプト内からの入力ファイルと出力ファイルを/beeondにステージングする必要があります。下記の例では、ホームディレクトリからBeeOND共有スクラッチ領域にインプットデータセットをコピーし、結果をホームディレクトリに返します。

```
#!/bin/sh
# 計算に必要な入力ファイルのコピー
cp -rp $HOME/datasets /beeond/
# 入力、出力を指定する計算プログラムの実行
./a.out $TMPDIR/datasets /beeond/results
# 必要な結果ファイルのコピー
cp -rp /beeond/results $HOME/results
```

## 6. ISVアプリケーション

ライセンス契約上、ISVアプリケーションを利用できる利用者は限られます。東工大に所属する「1.学生証・職員証」以外の利用者は以下のISVアプリケーションのみ利用できます。

- Gaussian/Gauss View
- AMBER 学術機関に所属する利用者に限る
- Intel Compiler
- PGI Compiler
- Arm Forge

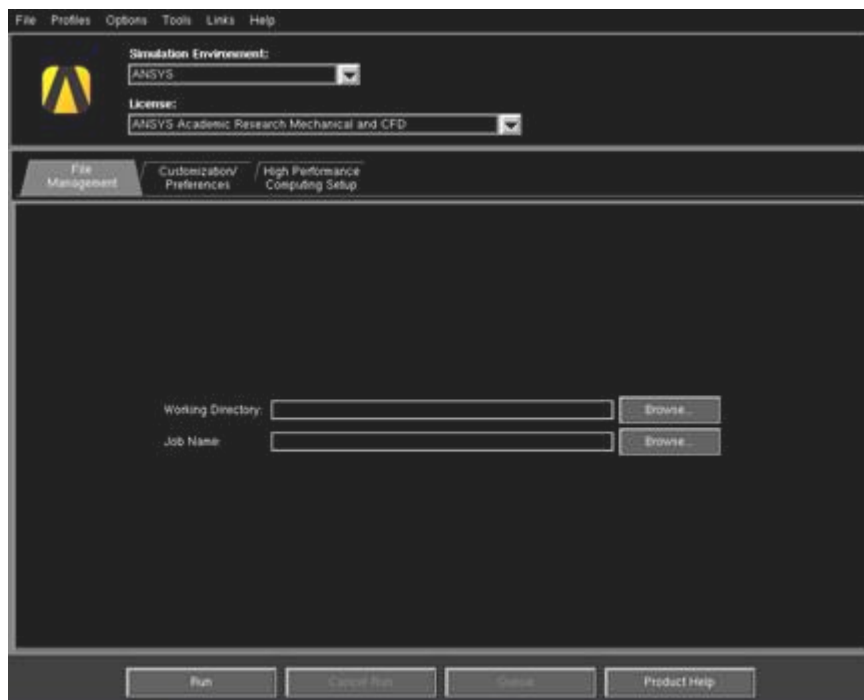
ISVアプリケーションの一覧表を以下に示します。

ソフトウェア名	概要
ANSYS	解析ソフトウェア
Fluent	解析ソフトウェア
ABAQUS	解析ソフトウェア
ABACUS CAE	解析ソフトウェア
Marc & Mentant / Dytran	解析ソフトウェア
Nastran	解析ソフトウェア
Patran	解析ソフトウェア
Gaussian	量子化学計算プログラム
GaussView	量子化学計算プログラム プリポストツール
AMBER	分子動力学計算プログラム
Materials Studio	化学シミュレーションソフトウェア
Discovery Studio	化学シミュレーションソフトウェア
Mathematica	数式処理ソフトウェア
Maple	数式処理ソフトウェア
AVS/Express	可視化ソフトウェア
AVS/Express PCE	可視化ソフトウェア
LS-DYNA	解析ソフトウェア
LS-PrePost	解析ソフトウェア プリポストツール
COMSOL	解析ソフトウェア
Schrodinger	化学シミュレーションソフトウェア
MATLAB	数値計算ソフトウェア
Arm Forge	デバッグ
Intel Compiler	コンパイラ
PGI Compiler	コンパイラ

## 6.1. ANSYS

GUIでの利用手順を以下に示します。

```
$ module load ansys
$ launcher
```



CLIでの利用手順を以下に示します。

```
$ module load ansys
$ mapdl
```

mapdlコマンドの代わりに以下のコマンドも使用できます。

ANSYS18.2の場合。バージョンによって異なります。

```
$ ansys182
```

`exit` と入力すると終了します。

入力ファイルを指定すると非対話的に実行されます。

```
実行例1
$ mapdl [options] < inputfile > outputfile
実行例2
$ mapdl [options] -i inputfile -o outputfile
```

バッチキューシステムを使用する場合は、シェルスクリプトを作成しCLIで以下のように実行します。

```
sample.shを使用する場合
$ qsub sample.sh
```

スクリプト例 MPI並列処理

```
#!/bin/bash
#$ -cwd
#$ -V
#$ -l f_node=2
#$ -l h_rt=0:10:0

. /etc/profile.d/modules.sh
module load ansys
```

```
mapdl -b -dis -np 56 < inputfile > outputfile
```

#### スクリプト例 GPU使用

```
#!/bin/bash
#$ -cwd
#$ -V
#$ -l f_node=1
#$ -l h_rt=0:10:0

. /etc/profile.d/modules.sh
module load ansys

mapdl -b -dis -np 28 -acc nvidia -na 4 < inputfile > outputfile
```

ANSYSのライセンス利用状況を以下のコマンドで確認できます。

```
$ lmsutil lmstat -S ansyslmd -c 27001@lice0:27001@remote:27001@t3ldapl
```

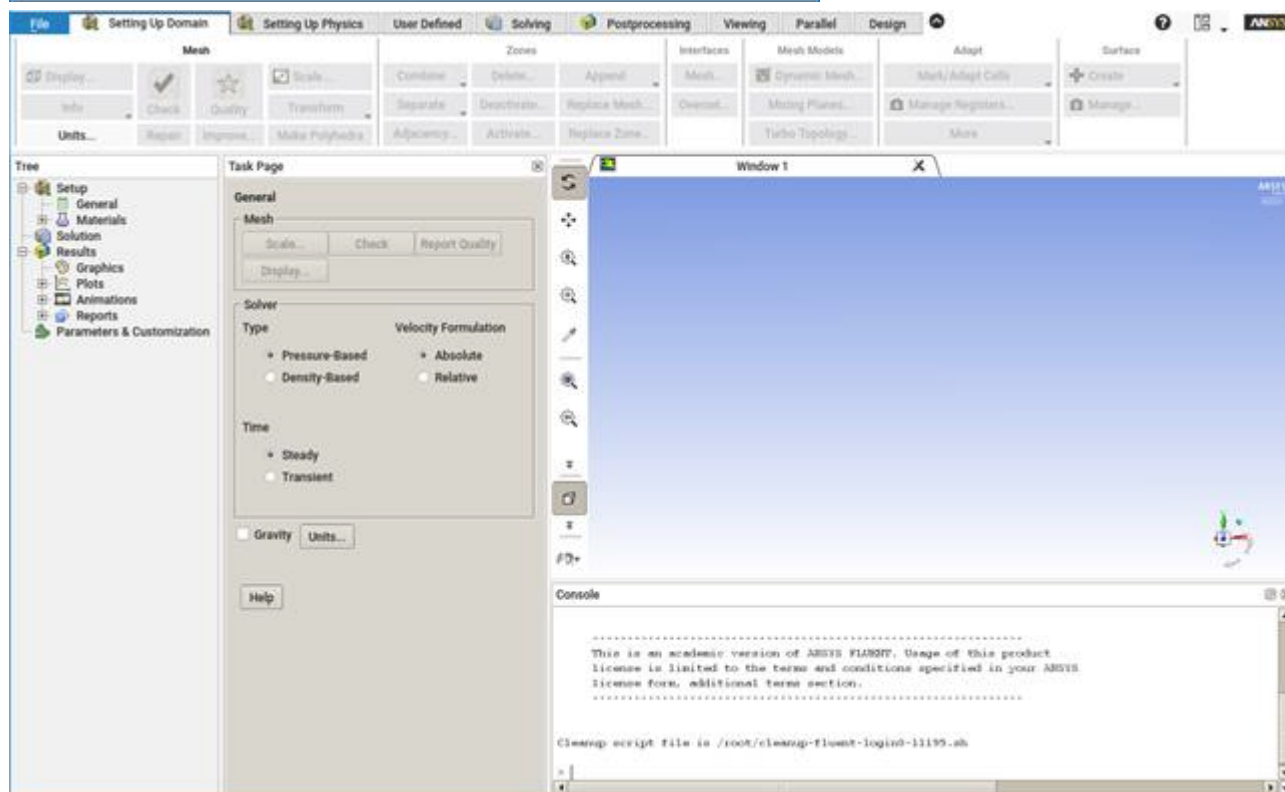
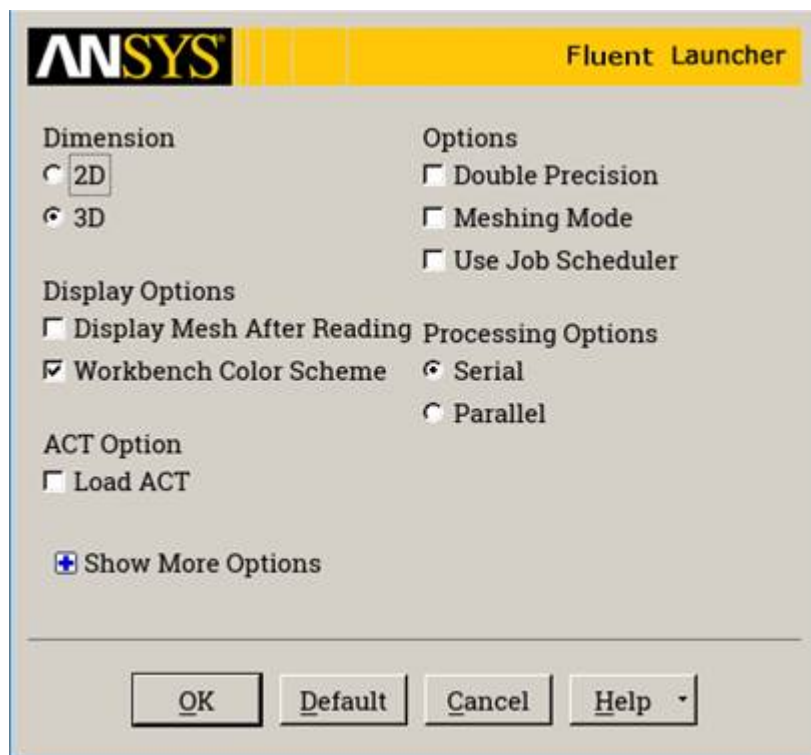
## 6.2. Fluent

---

Fluentは熱流体解析アプリケーションです。利用手順を以下に示します。

GUIでの起動手順を以下に示します。

```
$ module load ansys
$ fluent
```



CLIでの起動手順を以下に示します。

```
$ module load ansys
$ fluent -g
```

exitと入力すると終了します。

journalファイルを使用してインタラクティブに実行する場合は以下のようにコマンドを実行します。

```
journalファイル名がfluentbench.jou、3Dの場合
$fluent 3d -g -i fluentbench.jou
```

バッチキューシステムを使用する場合は、シェルスクリプトを作成しCLIで以下のように実行します。

```
sample.shを利用する場合
$ qsub sample.sh
```

#### スクリプト例 MPI並列処理(f\_node利用時)

```
#!/bin/bash
#$ -cwd
#$ -V
#$ -l f_node=2
#$ -l h_rt=0:10:0

. /etc/profile.d/modules.sh

module load ansys

JOURNAL=journalfile
OUTPUT=outputfile
VERSION=3d

fluent -mpi=intel -g ${VERSION} -cnf=${PE_HOSTFILE} -i ${JOURNAL} > ${OUTPUT} 2>&1
```

#### スクリプト例 MPI並列処理(h\_node利用時)

```
#!/bin/bash
#$ -cwd
#$ -V
#$ -l h_node=1
#$ -l h_rt=0:10:0

. /etc/profile.d/modules.sh
module load ansys

JOURNAL=journalfile
OUTPUT=outputfile
VERSION=3d

fluent -ncheck -mpi=intel -g ${VERSION} -cnf=${PE_HOSTFILE} -i ${JOURNAL} > ${OUTPUT} 2>&1
```

f\_node以外の利用では資源をまたぐ設定ができないため、`#$ -l {資源名}=1` (例えば `h_node` では `#$ -l h_node=1`) とし、コマンド中に `-ncheck` オプションを入れてください。

Fluentのライセンス利用状況を以下のコマンドで確認できます。

```
$ lmtutil lmstat -S ansyslmd -c 27001@lice0:27001@remote:27001@t3ldapl
```

## 6.3. ABAQUS

インタラクティブでの利用手順を以下に示します。

```
$ module load abaqus
$ abaqus job=inputfile [options]
```

バッチキューシステムを使用する場合は、シェルスクリプトを作成しCLIで以下のように実行します。

```
sample.shを利用する場合
$ qsub sample.sh
```

#### スクリプト例 MPI並列処理

```
#!/bin/bash
#$ -cwd
#$ -V
#$ -l q_core=1
#$ -l h_rt=0:10:0

. /etc/profile.d/modules.sh
module load abaqus

# ABAQUS settings.
INPUT=s2a
```

```

ABAQUS_VER=2017
ABAQUS_CMD=abq$(ABAQUS_VER)
SCRATCH=$(TMPDIR)
NCPUS=4

$(ABAQUS_CMD) interactive \
job=$(INPUT) \
cpus=$(NCPUS) \
scratch=$(SCRATCH) \
mp_mode=mpi > ${INPUT}.'date '+%Y%m%d%H%M%S'`.log 2>&1

```

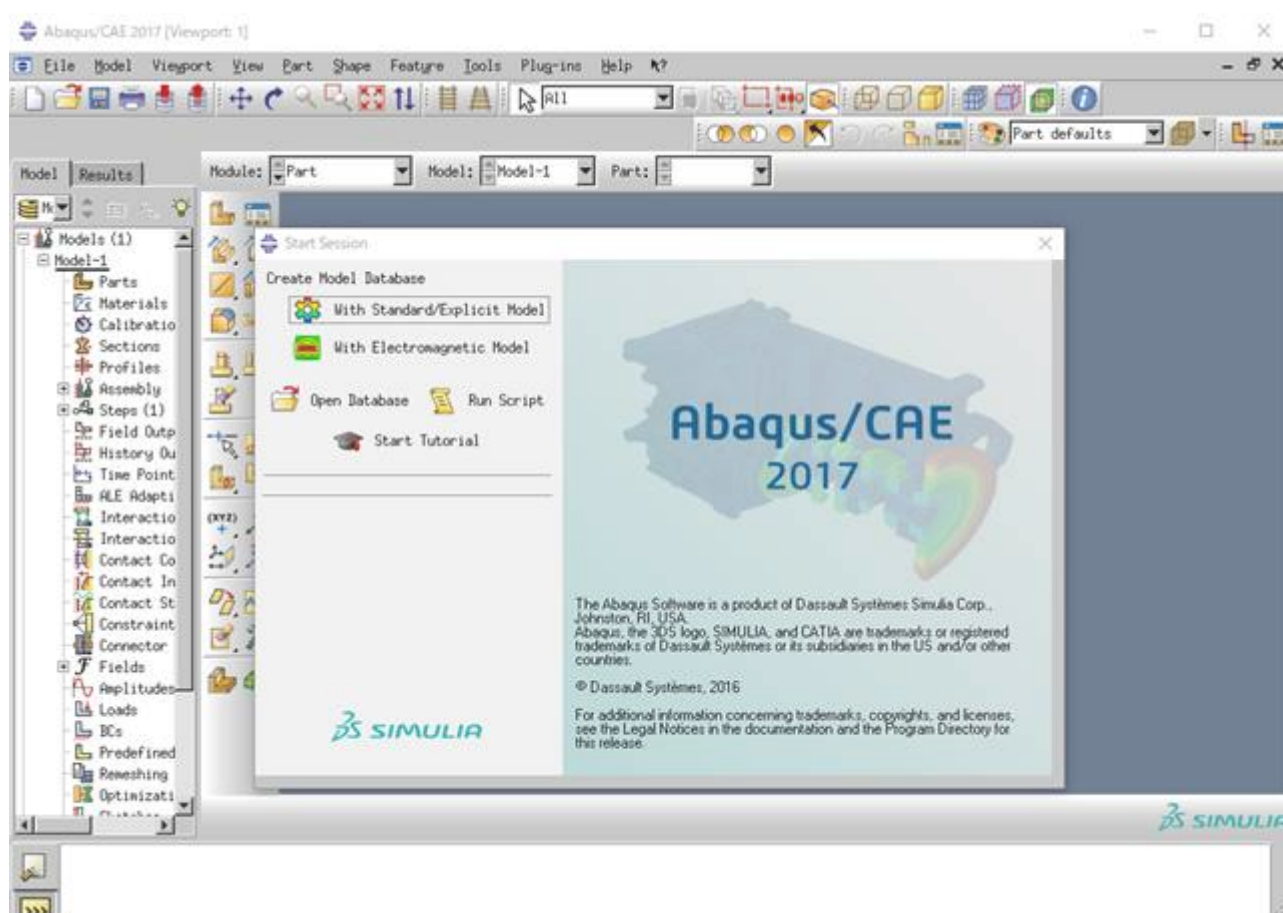
## 6.4. ABAQUS CAE

ABAQUS CAEの利用手順を以下に示します。

```

$ module load abaqus
$ abaqus cae

```



メニューバーの File > Exit をクリックすると終了します。

## 6.5. Marc & Mentat / Dytran

### 6.5.1. Marc & Mentat / Dytranの概要

各製品の概要はエムエスシーソフトウェア株式会社のWebサイトをご参照ください。

- Marc: <http://www.mscsoftware.com/ja/product/marc>
- Dytran: <http://www.mscsoftware.com/ja/product/dytran>

## 6.5.2. Marc & Mentat / Dytranのマニュアル

下記ドキュメントをご参照ください。

- [Marc & Mentat Docs](#) mscsoftware.com
- [Dytran Docs](#) mscsoftware.com

## 6.5.3. Marcの使用方法

インタラクティブでの利用手順を以下に示します。

使用したいバージョンに適宜読み替えてご実行ください。

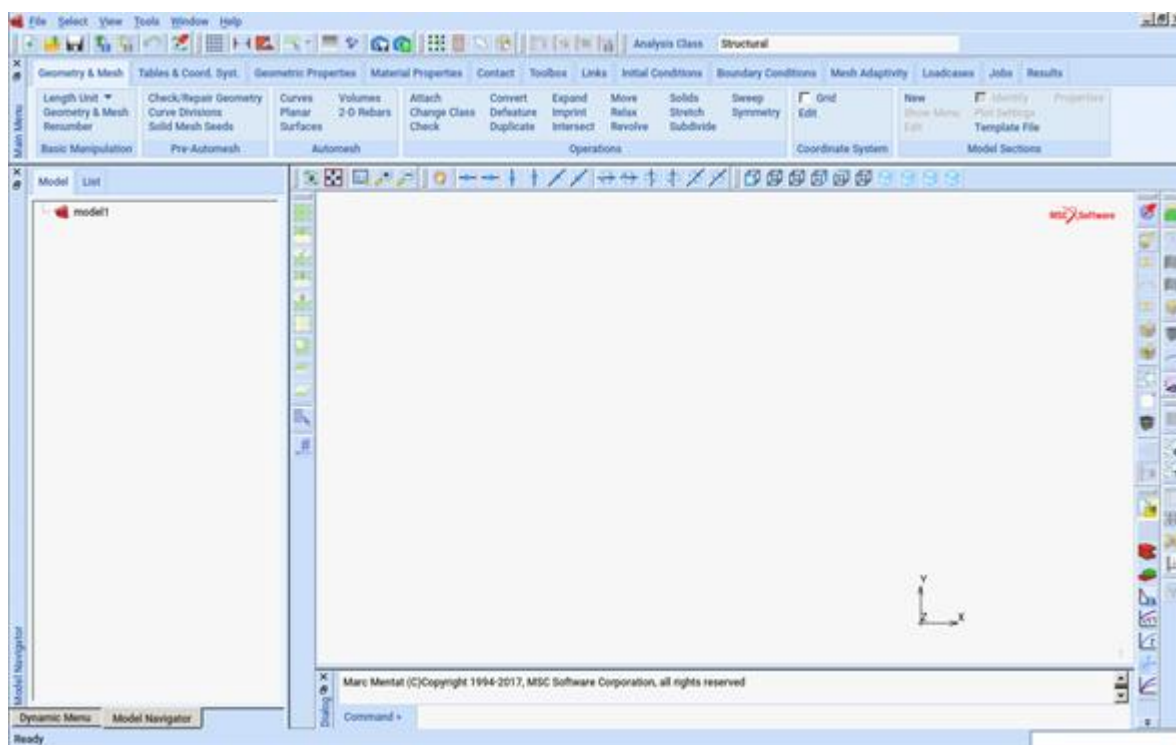
```
$ module load intel intel-mpi cuda marc_mentat/2017
サンプルファイル (e2x1.dat) の場合
$ cp /apps/t3/sles12sp2/ismv/msc/marc/marc2017/demo/e2x1.dat ./
$ marc -jid e2x1
```

## 6.5.4. Mentatの使用方法

Mentatの起動手順を以下に示します。

使用したいバージョンに適宜読み替えてご実行ください。

```
$ module load intel intel-mpi cuda marc_mentat/2017
$ mentat
```



メニューバーの File > Exit をクリックすると終了します。

Mentatのライセンス利用状況を以下のコマンドで確認できます。

```
$ lmutil lmstat -S MSC -c 27004@lice0:27004@remote:27004@t31dap1
```



## 6.6. Nastran

使用したいバージョンに適宜読み替えてご実行ください。

Nastranの起動手順を以下に示します。

```
$ module load nastran/2017.1
サンプルファイル (um24.dat) の場合
$ cp /apps/t3/sles12sp2/iscv/msc/MSC_Nastran/20171/msc20171/nast/demo/um24.dat ./
$ nast20171 um24
```

Nastranのバッチ投入手順を以下に示します。

```
サンプルファイル (parallel.sh) の場合
$ qsub parallel.sh
```

スクリプト例 CPU並列処理

```
#!/bin/bash
#$ -cwd
#$ -l q_core=1
#$ -l h_rt=0:10:00
#$ -V

export NSLOTS=4

. /etc/profile.d/modules.sh
module load cuda openmpi nastran/2017.1

mpirun -np $NSLOTS \
nast20171 parallel=$NSLOTS um24
```

Nastranのライセンス利用状況を以下のコマンドで確認できます。

```
$ lutil lmstat -S MSC -c 27004@lice0:27004@remote:27004@t3ldapl
```

## 6.7. Patran

Patranの起動手順を以下に示します。

使用したいバージョンに適宜読み替えてご実行ください。

```
$ module load patran/2017.0.2
$ pat2017
```



終了する際はFile>EXIT

Patranのライセンス利用状況を以下のコマンドで確認できます。

```
$ lutil lmstat -S MSC -c 27004@lice0:27004@remote:27004@t3ldapl
```

## 6.8. Gaussian

インタラクティブな利用手順を以下に示します。

GPUを利用するモジュールを読み込む場合 環境変数GAUSS\_CDEF及びGAUSS\_GDEFを自動設定します

```
$ module load gaussian16/revision_gpu
$ g16 inputfile
```

revisionには使用するリビジョンを指定してください。Gaussian16 Rev.B01の場合は以下の通りです。

```
$ module load gaussian16/B01_gpu
```

GPUを利用しないモジュールを読み込む場合 環境変数GAUSS\_CDEF/GAUSS\_GDEFは設定されません

```
$ module load gaussian16/revision
$ g16 inputfile
```

Linda並列版モジュールを読み込む場合

```
$ module load gaussian16_linda
$ g16 inputfile
```

バッチキューシステムを使用する場合は、シェルスクリプトを作成しCLIで以下のように実行します。

```
sample.shを使用する場合
$ qsub sample.sh
```

スクリプト例 ノード内並列処理

Glycineの構造最適化および振動解析(IR+ラマン強度)を計算する場合のサンプルスクリプトです。

下記のglycine.sh、glycine.gjfを同一ディレクトリ上に配置し、下記コマンドを実行することで計算ができます。計算後にglycine.log、glycine.chkが生成されます。

解析結果の確認についてはGaussViewにてご説明します。

```
$ qsub glycine.sh
```

glycine.sh

```
#!/bin/bash
#$ -cwd
#$ -l f_node=1
#$ -l h_rt=0:10:0
#$ -V

. /etc/profile.d/modules.sh
module load gaussian16

g16 glycine.gjf
```

glycine.gjf

```
chk=glycine.chk
cpu=0-27      +環境変数GAUSS_CDEF/GAUSS_GDEFを自動設定するモジュールを読み込んだ場合は不要
gpuscpu=0-3=0,1,2,3  +GPUを使用しない場合や環境変数を自動設定するモジュールを読み込んだ場合は不要
mem=120GB

P opt=(calcf,c,tight,rfo) freq=(raman)

glycine Test Job

  2
N      0   -2.15739574   -1.69517043   -0.01896033  H
H      0   -1.15783574   -1.72483643   -0.01896033  H
C      0   -2.84434974   -0.41935843   -0.01896033  H
C      0   -1.83982674    0.72406557   -0.01896033  H
H      0   -3.46918274   -0.34255543   -0.90878333  H
H      0   -3.46918274   -0.34255543    0.87086267  H
O      0   -0.63259574    0.49377357   -0.01896033  H
O      0   -2.22368674    1.89158057   -0.01896033  H
H      0   -2.68286796   -2.54598119   -0.01896033  H

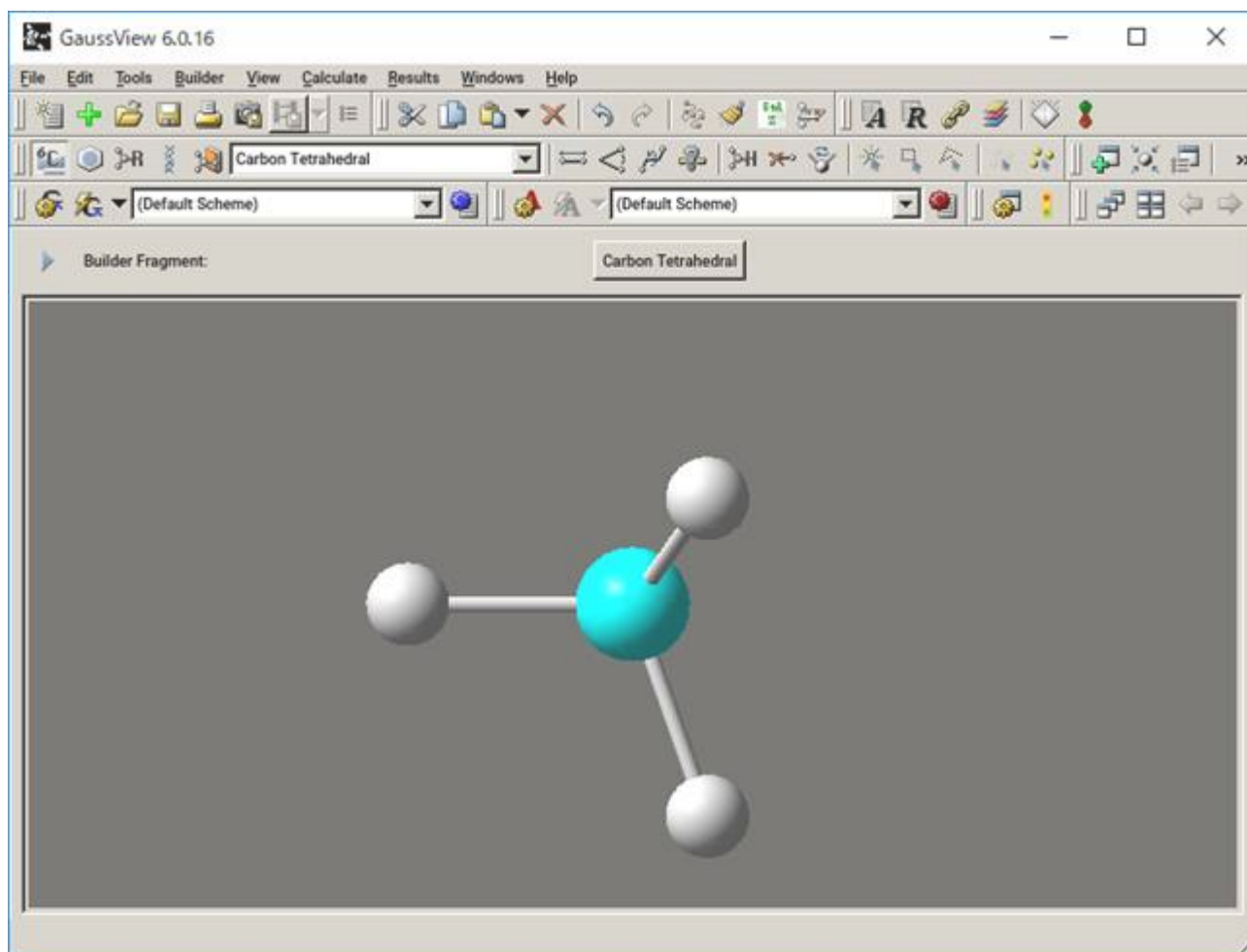
1 2 1.0 3 1.0 9 1.0
2
3 4 1.0 5 1.0 6 1.0
4 7 1.5 8 1.5
5
6
7
8
9
```

## 6.9. GaussView

GaussViewはGaussianの結果を可視化するアプリケーションです。

GaussViewの利用手順を以下に示します。

```
$ module load gaussian16 gaussview  
$ gview.exe
```



メニューバーから File > Exit をクリックすると終了します。

解析例 glycine.log

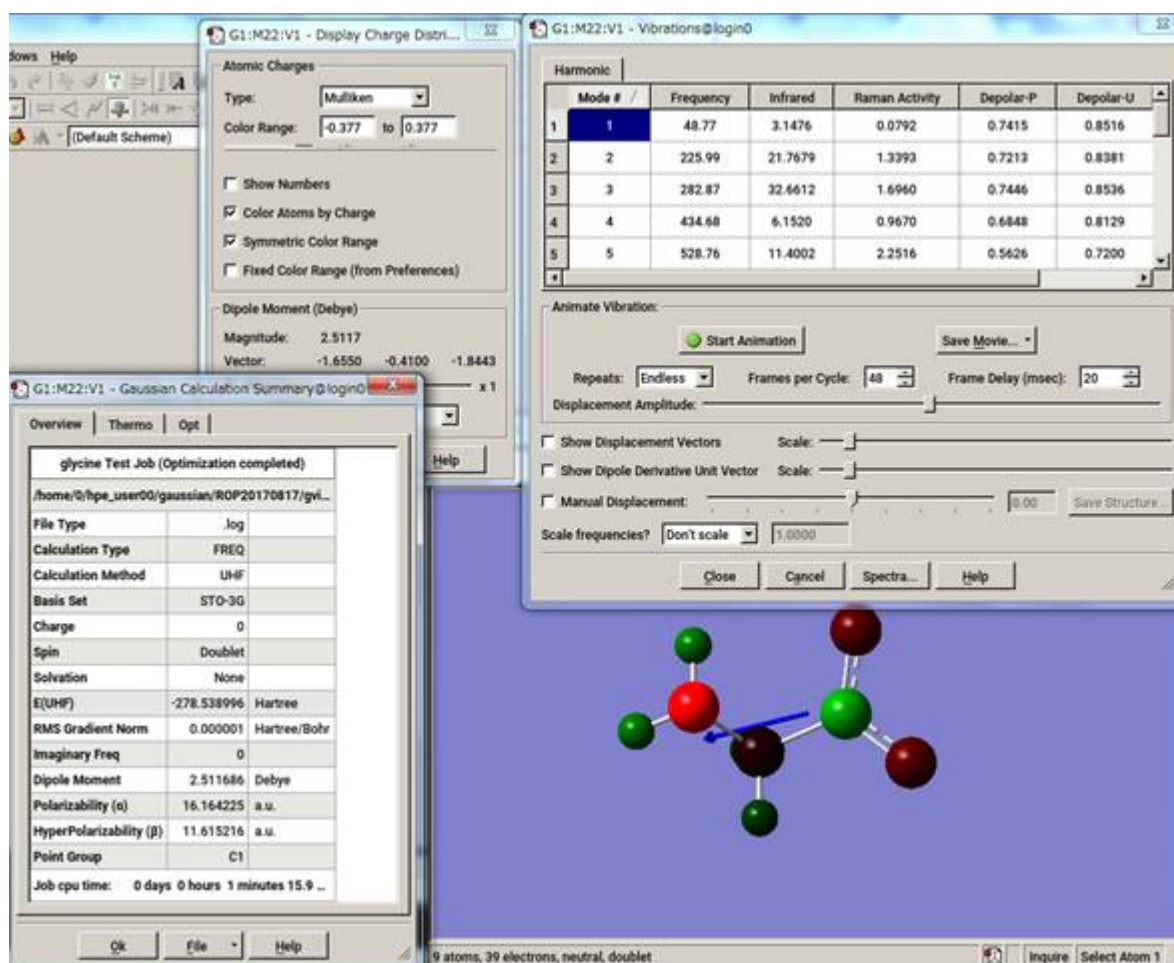
Gaussianの項にてサンプルとして例示しているスクリプトを実行した結果ファイルの解析を例にご説明します。

```
$ module load gaussian16 gaussview  
$ gview.exe glycine.log
```

Resultから解析結果の確認が可能です。

Result>Summaryにて計算の概要、Result>ChageDistribution...で電荷情報、Vibration...から振動解析の結果を確認できます。

サンプルでは振動解析を行っているので、VibrationダイアログのStartAnimationから振動の様子を確認できます。



## 6.10. AMBER

AMBERは本来タンパク質・核酸の分子動力学計算のために開発されたプログラムですが、最近では糖用のパラメータも開発され、化学・生物系の研究のために益々有用なツールとなってきました。ご自分の研究で利用する場合は、マニュアルや関連する論文等の使用例をよく調べて、AMBERが採用しているモデルや理論の限界、応用範囲等を把握しておくことが必要です。現在、AMBERはソースコードを無制限にコピーすることはできませんが、東工大内部で利用することは可能なので、これを基にさらに発展した手法を取り込むことも可能です。

下記について、使用したいバージョンに適宜読み替えてご実行ください。

インタラクティブでの逐次処理の場合の利用手順を以下に示します。

```
$ module load amber/16
$ sander [-O]A -i mdin -o mdout -p prmtop -c inpcrd -r restrt
```

インタラクティブでの並列処理(sander.MPI)の場合の利用手順を以下に示します。

```
$ module load amber/16
$ mpirun -np -[並列数] sander.MPI [-O]A -i mdin -o mdout -p prmtop -c inpcrd -r restrt
```

インタラクティブでのGPU逐次処理(pmemd.cuda)の場合の利用手順を以下に示します。

```
$ module load amber/16_cuda
$ pmemd.cuda [-O] -i mdin -o mdout -p prmtop -c inpcrd -r restrt
```

インタラクティブでのGPU並列処理(pmemd.cuda.MPI)の場合の利用手順を以下に示します。

```
$ module load amber/16_cuda
$ mpirun -np -[並列数] pmemd.cuda.MPI [-O] -i mdin -o mdout -p prmtop -c inpcrd -r restrt
```

バッチキューシステムの場合の利用手順を以下に示します。

```
parallel.shを利用する場合
$ qsub parallel.sh
```

### スクリプト例 CPU並列処理

```
#!/bin/bash
#$ -cwd
#$ -l f_node=2
#$ -l h_rt=0:10:00
#$ -V
export NSLOTS=56

in=./mdin
out=./mdout_para
inpcrd=./inpcrd
top=./top

cat <<eof > $in
Relaxtion of trip cage using
&cntrl
  imin=1,maxcyc=5000,irest=0, ntx=1,
  nstlim=10, dt=0.001,
  ntc=1, ntf=1, ioutfm=1
  ntt=9, tautp=0.5,
  tempi=298.0, temp0=298.0,
  ntpr=1, ntwx=20,
  ntb=0, igb=8,
  nkija=3, gamma_ln=0.01,
  cut=999.0, rgbmax=999.0,
  idistr=0
/
eof

. /etc/profile.d/modules.sh
module load amber/16

mpirun -np $NSLOTS \
sander.MPI -O -i $in -c $inpcrd -p $top -o $out < /dev/null

/bin/rm -f $in restrt
```

### スクリプト例 GPU並列処理

```
#!/bin/bash
#$ -cwd
#$ -l f_node=2
#$ -l h_rt=0:10:0
#$ -V

export NSLOTS=56

in=./mdin
out=./mdout
inpcrd=./inpcrd
top=./top

cat <<eof > $in
FIX (active) full dynamics ( constraint dynamics: constant volume)
&cntrl
  ntx = 7,      irest = 1,
  ntpr = 100,   ntwx = 0,   ntwr = 0,
  ntf = 2,      ntc = 2,    tol = 0.000001,
  cut = 8.0,
  nstlim = 500, dt = 0.00150,
  nscm = 250,
  ntt = 0,
  lastist = 4000000,
  lastrst = 6000000,
/
eof

. /etc/profile.d/modules.sh
module load amber/16_cuda

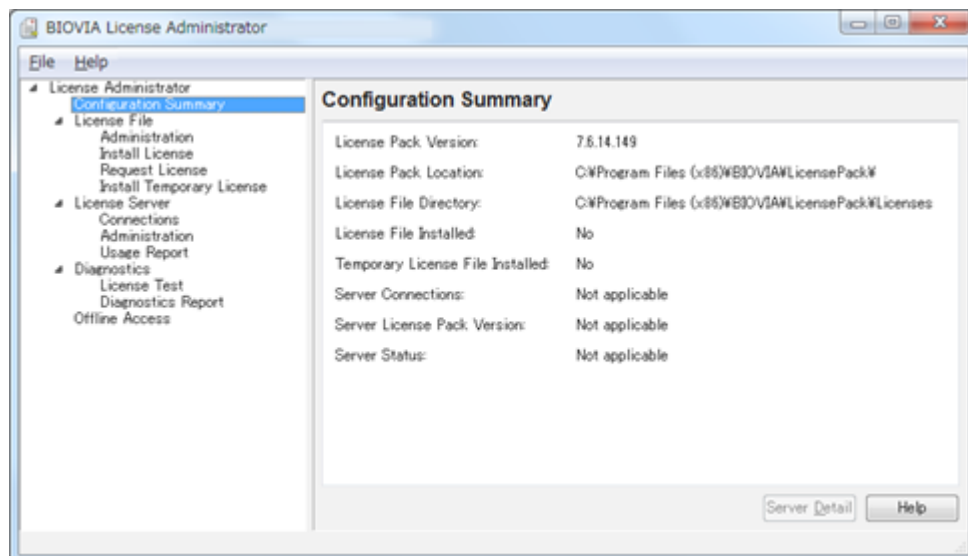
mpirun -np $NSLOTS \
pmemd.cuda.MPI -O -i $in -c $inpcrd -p $top -o $out < /dev/null

/bin/rm -f $in restrt
```

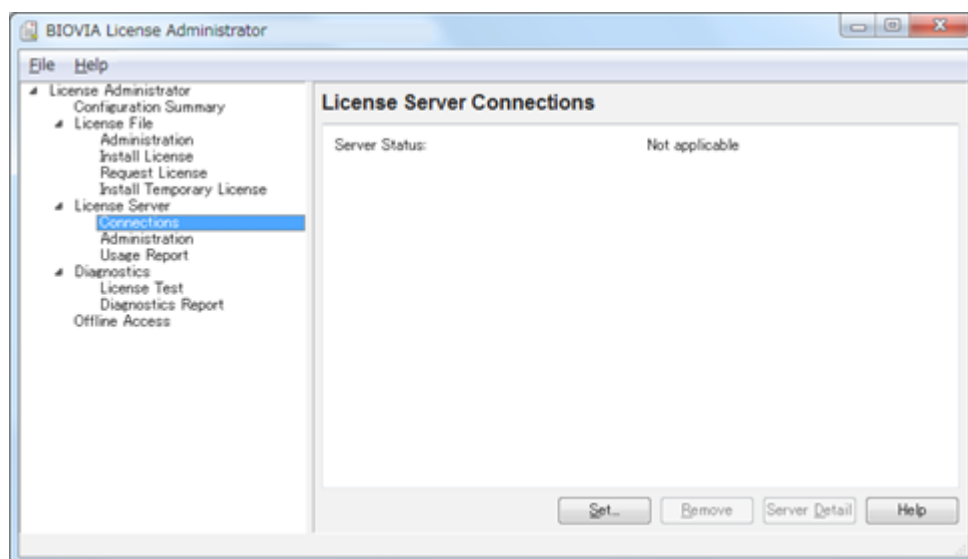
## 6.11. Materials Studio

### 6.11.1. ライセンス接続設定方法

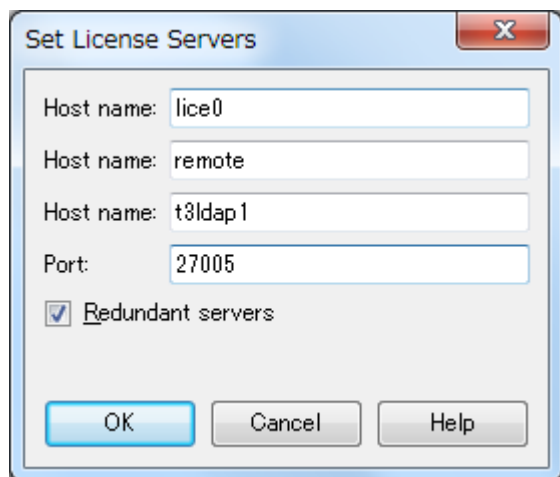
スタートメニューから すべてのプログラム > BIOVIA > Licensing > License Administrator 7.6.14 を管理者として実行します。



[Connections] を開き、[Set] をクリックして Set License Server ダイアログを開きます。



Redundant servers にチェックを入れ、ホスト名とポート番号を下図のように入力し、[OK] をクリックします。



Server Status が Connected と表示されれば設定完了です。

※Materials Studioを利用するためには、2ホスト以上のライセンスサーバーへの接続が確立している必要があります。

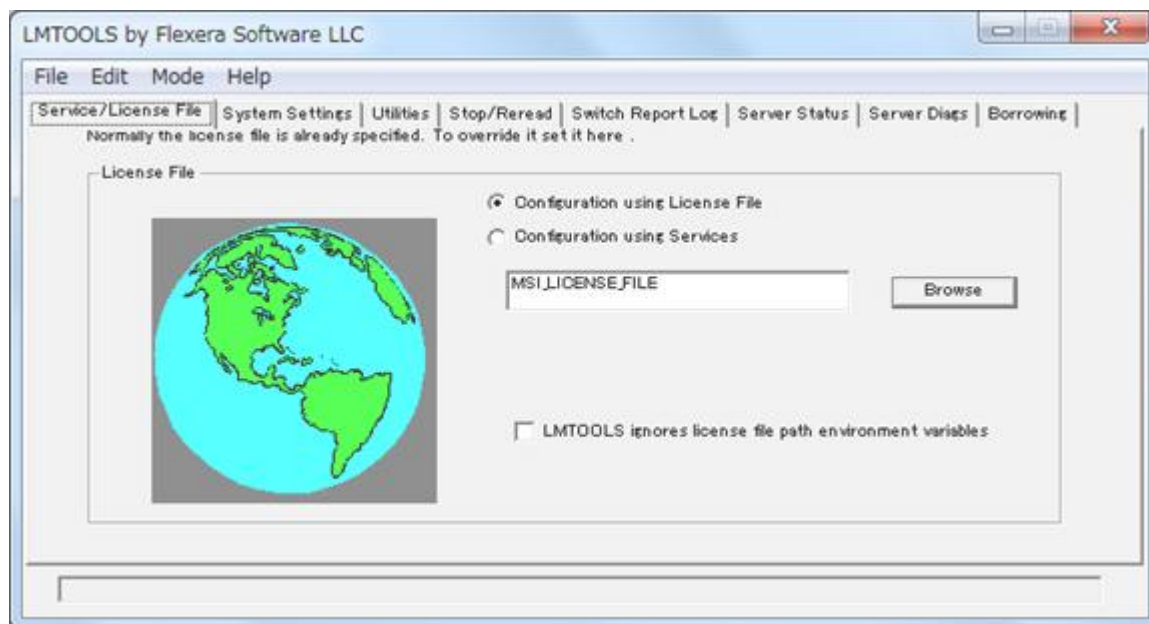
## 6.11.2. ライセンス利用状況の確認方法

### 6.11.2.1. Windowsでの確認方法

スタートメニューから すべてのプログラム > BIOVIA > Licensing > License Administrator 7.6.14 > Utilities (FLEXlm LMTTOOLS) を実行します。

[Service/License File] タブを開き、[Configuration using License File] を選択します。

MSI\_LICENSE\_FILE と表示されていることを確認します。



[Server Status] タブを開き、[Perform Status Enquiry] をクリックすると、ライセンスの利用状況が表示されます。

特定のライセンスのみを表示したい場合は、[Individual Feature] に表示したいライセンス名を入力して [Perform Status Enquiry] を実行します。

### 6.11.2.2. ログインノード上での確認方法

以下のコマンドを実行すると、利用状況が表示されます。

```
$ lmtutil lmstat -S msi -c 27005@lice0,27005@remote,27005@t3ldap1
```

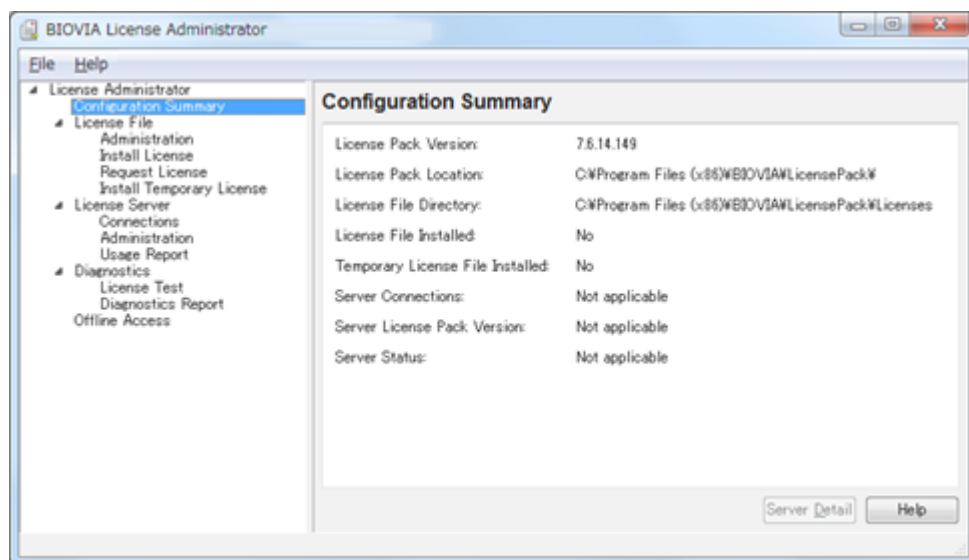
### 6.11.3. Materials Studioの起動方法

Materials StudioがインストールされたWindows環境においてスタートメニューを表示し、BIOVIA > Materials Studio 2017 R2 をクリックして起動します。

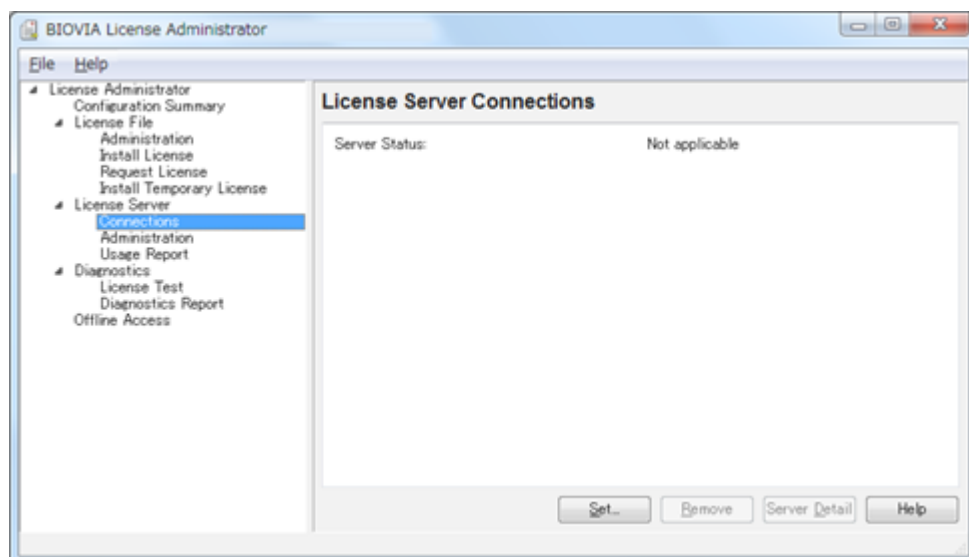
## 6.12. Discovery Studio

### 6.12.1. ライセンス接続設定方法

スタートメニューから すべてのプログラム > BIOVIA > Licensing > License Administrator 7.6.14 を管理者として実行します。

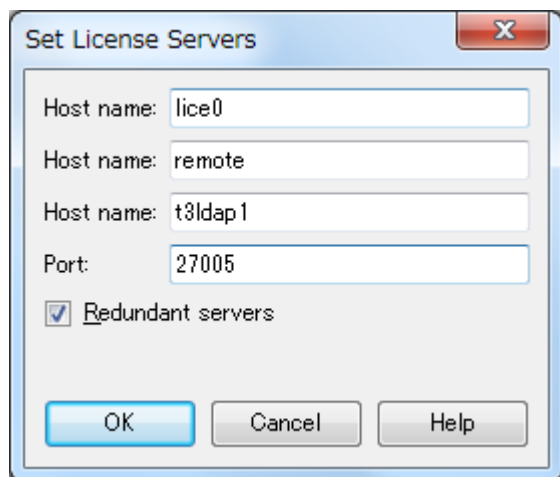


[Connections] を開き、[Set] をクリックして Set License Server ダイアログを開きます。



Redundant servers にチェックを入れ、ホスト名とポート番号を下図のように入力し、[OK] をクリックします。





Server Status が Connected と表示されれば設定完了です。

※Discovery Studioを利用するためには、2ホスト以上のライセンスサーバーへの接続が確立している必要があります。

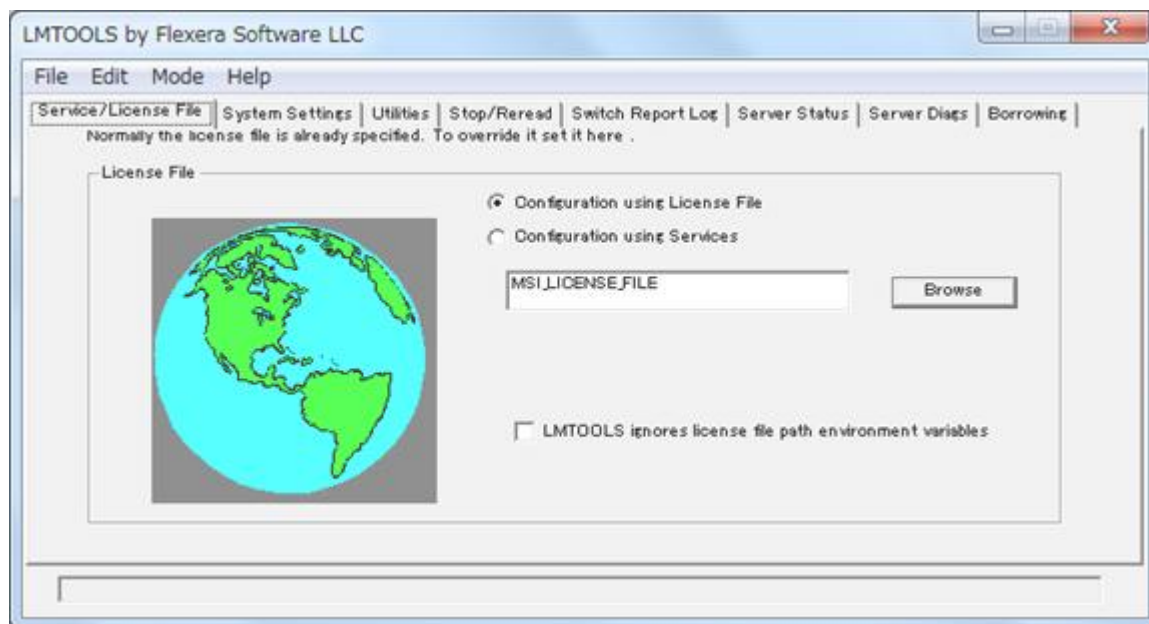
## 6.12.2. ライセンス利用状況の確認方法

### 6.12.2.1. Windowsでの確認方法

スタートメニューから すべてのプログラム > BIOVIA > Licensing > License Administrator 7.6.14 > Utilities (FLEXlm LMTTOOLS) を実行します。

[Service/License File] タブを開き、[Configuration using License File] を選択します。

MSI\_LICENSE\_FILE と表示されていることを確認します。



[Server Status] タブを開き、[Perform Status Enquiry] をクリックすると、ライセンスの利用状況が表示されます。

特定のライセンスのみを表示したい場合は、[Individual Feature] に表示したいライセンス名を入力して [Perform Status Enquiry] を実行します。

### 6.12.2.2. ログインノード上での確認方法

以下のコマンドを実行すると、利用状況が表示されます。

```
$ lmtutil lmstat -S msi -c 27005@lice0,27005@remote,27005@t3ldap1
```

### 6.12.3. Discovery Studioの起動方法

Discovery StudioがインストールされたWindows環境においてスタートメニューを表示し、BIOVIA > Discovery Studio 2017 R2 64-bit Client をクリックして起動します。

## 6.13. Mathematica

CLIでの起動手順を示します。

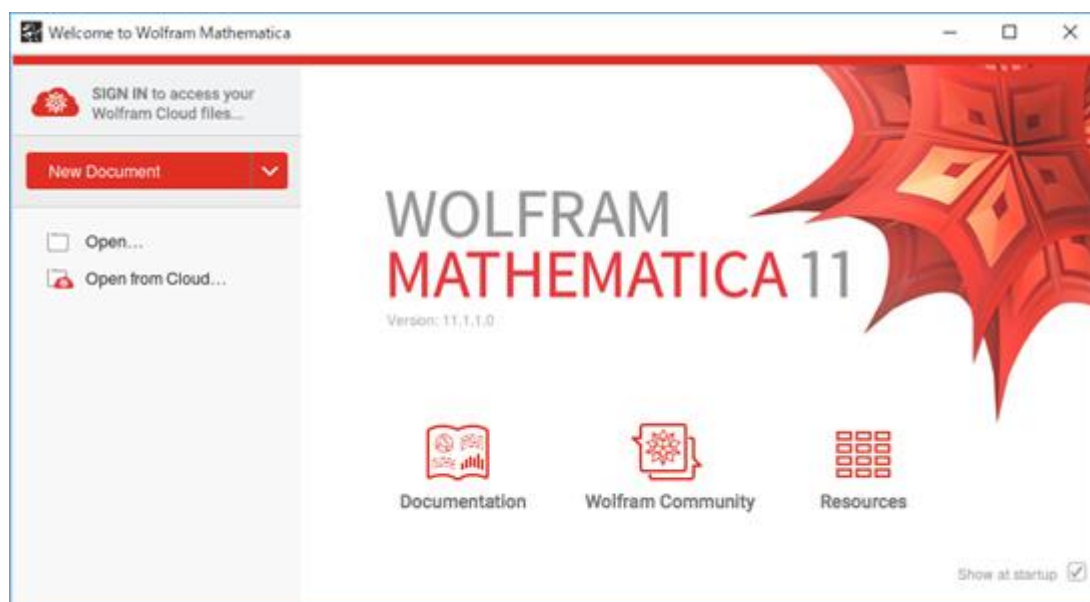
```
$ module load mathematica
$ math
Mathematica 11.1.1 Kernel for Linux x86 (64-bit)
Copyright 1988-2017 Wolfram Research, Inc.

In[1]:=
```

Quitと入力すると終了します。

GUIでの起動手順を以下に示します。

```
$ module load mathematica
$ Mathematica
```



終了する場合は、ノートブックのメニューバーから [File] を選択し「Exit」をクリックします。

## 6.14. Maple

Mapleは数式処理、数値計算アプリケーションです。

Mapleの利用方法の例を以下に示します。

使用したいバージョンに適宜読み替えてご実行ください。

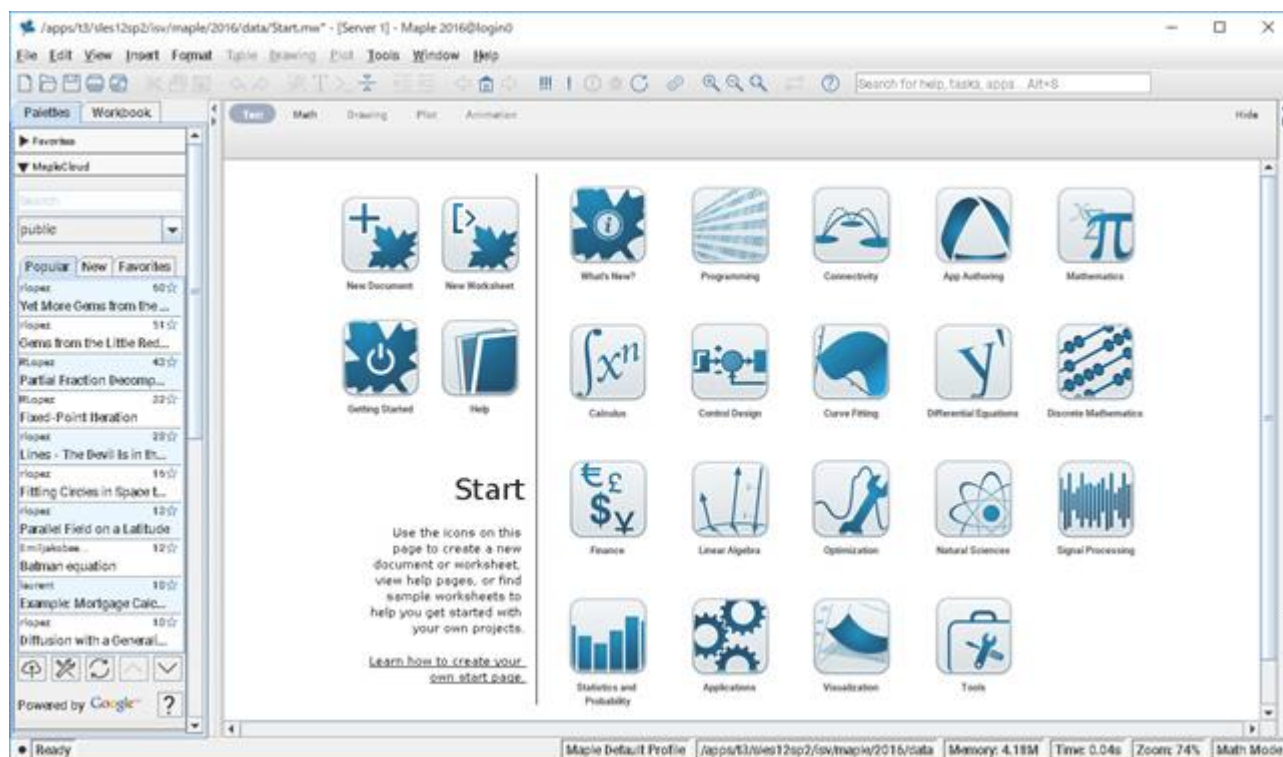
CLIでの起動手順を以下に示します。

```
$ module load maple/2016.2
$ maple
| ^ |      Maple 2016 (X86 64 LINUX)
_|_|_|_|_|. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2018
\ MAPLE /   All rights reserved. Maple is a trademark of
<__>      Waterloo Maple Inc.
|          Type ? for help.
>
```

quitと入力すると終了します。

GUIでの起動手順を以下に示します。

```
$ module load maple/2016.2
$ xmaple
```



メニューバーの File > Exit をクリックすると終了します。

Mapleのライセンス利用状況を以下のコマンドで確認できます。

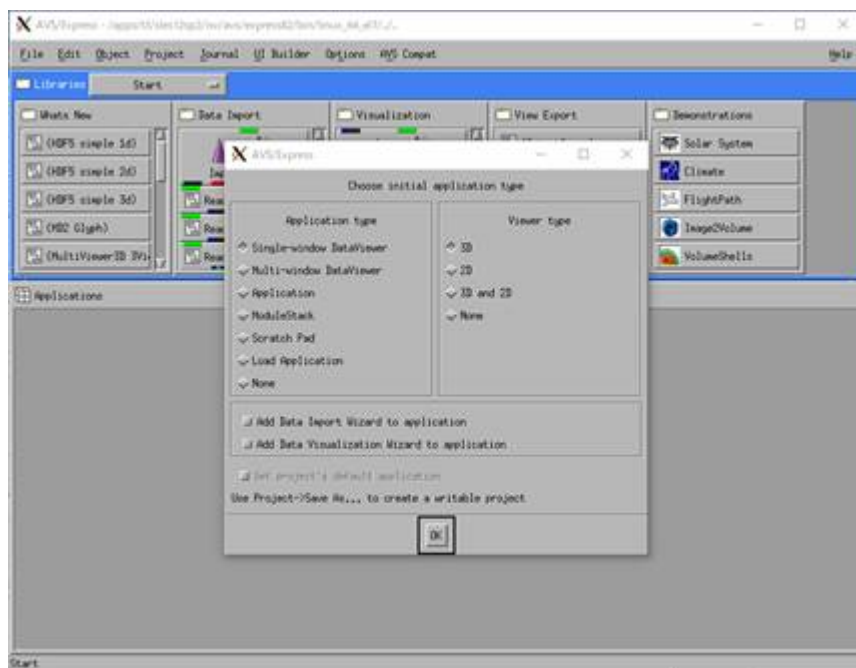
```
$ lmtutil lmstat -S maplelmg -c 27007@lice0:27007@remote:27007@t31dap1
```

## 6.15. AVS/Express

AVS/Expressの利用手順を以下に示します。

```
$ module load avs/8.4
$ xp
```

ハードウェアアクセラレーションを無効にして起動する場合は-nohwオプションを付けて実行してください。



メニューバーの File > Exit をクリックすると終了します。

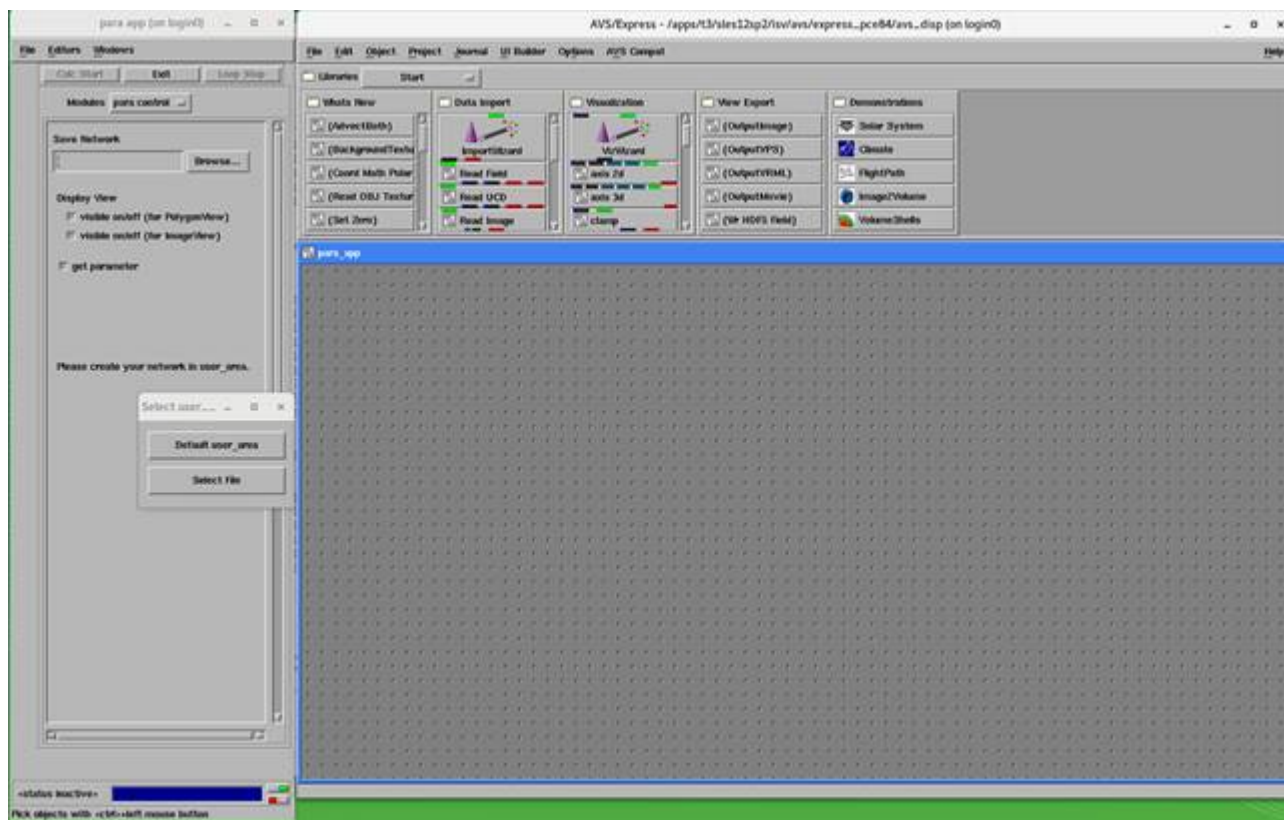
以下のコマンドでライセンス利用状況を確認できます。

```
$ w3m http://lice0:33333/STATUS
```

## 6.16. AVS/Express PCE

AVS/Express PCEの利用手順を以下に示します。

```
$ module load avs/8.4
$ para_start
```



メニューバーの File > Exit をクリックすると終了します。

以下のコマンドでライセンス利用状況を確認できます。

```
$ w3m http://lice0:33333/STATUS
```

## 6.17. LS-DYNA

### 6.17.1. LS-DYNAの概要

LS-DYNAは、陽解法により構造物の大変形挙動を時刻履歴で解析するプログラムで、衝突 衝撃解析、落下解析、塑性加工解析、貫通 亀裂 破壊解析などに威力を発揮し、これらの分野では世界有数の導入実績を誇る信頼性の高いプログラムです。

### 6.17.2. LS-DYNAの使用方法

LS-DYNAはバッチジョブで利用します。バッチスクリプトの例を以下に示します。

使用したいバージョンに適宜読み替えてご実行ください。

スクリプト例 SMP単精度版

```
#!/bin/bash
#$ -cwd
#$ -V
#$ -l h_node=1
#$ -l h_rt=0:10:0

. /etc/profile.d/modules.sh
module load cuda/8.0.44
module load lsdyna/R9.1.0

export exe=smpdynas

export NCPUS=4
export OMP_NUM_THREADS=${NCPUS}
export INPUT=airbag_deploy.k
```

```

${exe} i=${INPUT} ncpus=${NCPUS}

```

### スクリプト例 SMP倍精度版

```

#!/bin/bash
#$ -cwd
#$ -V
#$ -l h_node=1
#$ -l h_rt=0:10:0

. /etc/profile.d/modules.sh
module load cuda/8.0.44
module load lsdyna/R9.1.0

export exe=smpdynad

export NCPUS=4
export OMP_NUM_THREADS=${NCPUS}
export INPUT=airbag_deploy.k

${exe} i=${INPUT} ncpus=${NCPUS}

```

### スクリプト例 MPP単精度版

```

#!/bin/bash
#$ -cwd
#$ -V
#$ -l h_node=1
#$ -l h_rt=0:10:0

. /etc/profile.d/modules.sh
module load cuda/8.0.44
module load lsdyna/R9.1.0 mpt/2.16

export exe=mppdynas_avx2
export dbo=l2as_avx2

export NCPUS=4
export OMP_NUM_THREADS=1
export INPUT=airbag_deploy.k

export MPI_BUFS_PER_PROC=512
export MPI_REMSH=ssh

mpiexec_mpt -v -np 4 dplace -s1 ${exe} i=${INPUT} ncpus=${NCPUS}
${dbo} binout*

```



lsdyna モジュールでロードされるもの以外に、ANSYSに含まれるLS-DYNAも利用可能です。  
利用の際には下記スクリプト例をご参照ください。

```

#!/bin/bash
#$ -cwd
#$ -V
#$ -l h_node=1
#$ -l h_rt=5:00:0

. /etc/profile.d/modules.sh

module load ansys intel-mpi

export dynadir=/apps/t3/sles12sp2/isv/ansys_inc/v231/ansys/bin/linux64/
export exe=$dynadir/lsdyna_sp_mpp.e
export dbo=$dynadir/ls12a_sp.e

export LSTC_LICENSE_SERVER='(27008@lice0 27008@remote 27008@t3ldapl)'
export NCPUS=4
export INPUT=$base_dir/sample/airbag_deploy.k

mpiexec -np ${NCPUS} ${exe} i=${INPUT}

${dbo} binout*

```

### スクリプト例 MPP倍精度版

```

#!/bin/bash
#$ -cwd
#$ -V
#$ -l h_node=1
#$ -l h_rt=0:30:0

```

```
. /etc/profile.d/modules.sh
module load cuda/8.0.44
module load lsdyna/R9.1.0 mpt/2.16

export exe=mppdynad_avx2
export dbo=l2ad_avx2

export NCPUS=4
export OMP_NUM_THREADS=1
export INPUT=airbag_deploy.k

export MPI_BUFS_PER_PROC=512
export MPI_REMSH=ssh

mpiexec_mpt -v -np 4 dplace -s1 ${exe} i=${INPUT} ncpus=${NCPUS}
${dbo} binout*
```



lsdyna モジュールでロードされるもの以外に、ANSYSに含まれるLS-DYNAも利用可能です。  
利用の際には下記スクリプト例をご参照ください。

```
#!/bin/bash
#$ -cwd
#$ -V
#$ -l h_node=1
#$ -l h_rt=5:00:0

. /etc/profile.d/modules.sh

module load ansys intel-mpi

export dynadir=/apps/t3/sles12sp2/ismv/ansys_inc/v231/ansys/bin/linux64/
export exe=$dynadir/lsdyna_dp_mpp.e
export dbo=$dynadir/ls12a_dp.e

export LSTC_LICENSE_SERVER='(27008@lice0 27008@remote 27008@t31dap1)'
export NCPUS=4
export INPUT=$base_dir/sample/airbag_deploy.k

mpiexec -np ${NCPUS} ${exe} i=${INPUT}

${dbo} binout*
```

スクリプトは、利用者の環境に合わせて変更してください。上記スクリプト例では、インプットファイルはシェルスクリプト内でINPUT=inputfileとして指定しています。

LS-DYNAのライセンス利用状況は以下のコマンドで確認できます。

```
$ lstc_qrun
```

## 6.18. LS-PrePost

### 6.18.1. LS-PrePostの概要

LS-PrePostはLS-DYNAと合わせて無償提供されている先進的なプリポストツールになります。ユーザインターフェースは効率的かつ直感的に扱えるようにデザインされています。LS-PrePostは高速なレンダリングとXYプロットを実現するためにOpenGLグラフィックスを利用します。

### 6.18.2. LS-PrePostの使用方法

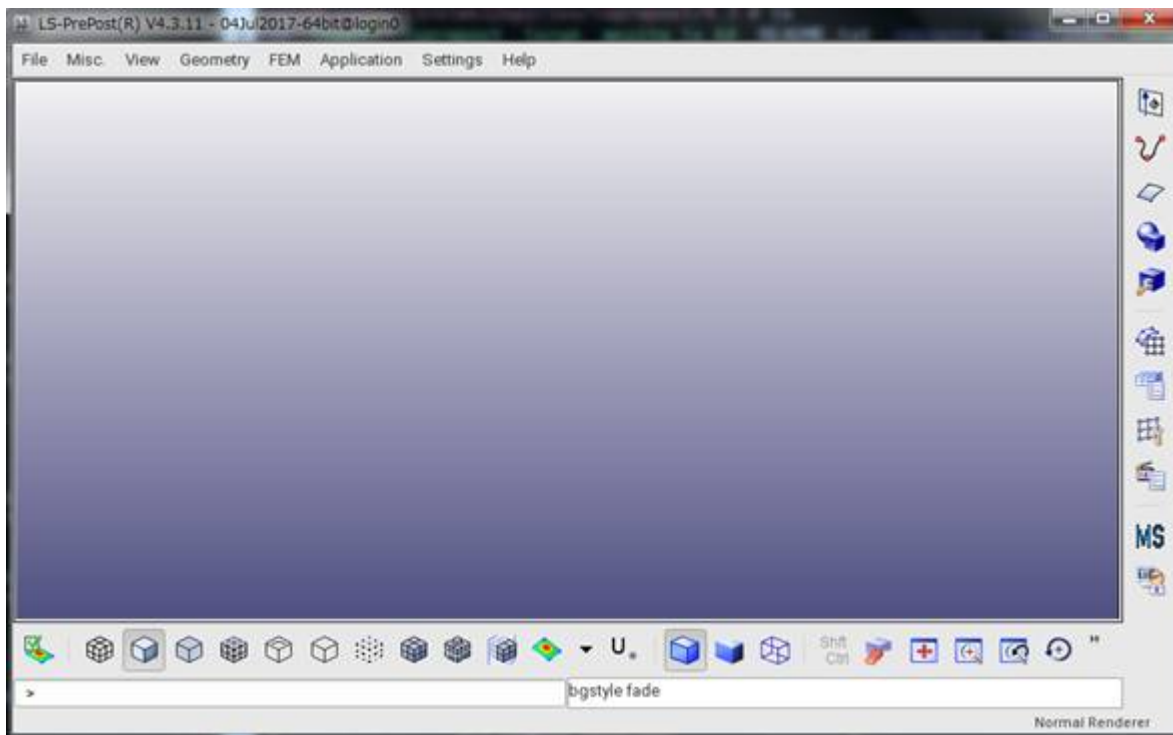
LS-PrePostの起動手順を以下に示します。

```
$ module load lsprepost/4.3
$ lsprepost
```

```
|
|  Livermore Software Technology Corporation
|
|      L S - P R E P O S T
|
|  Advanced Pre- and Post-Processor for LS-DYNA
|
|      LS-PrePost (R) V4.3.11 - 04Jul2017
|
```

```
LSTC Copyright (C) 1999-2014  
All Rights Reserved
```

```
OpenGL version 3.0 Mesa 11.2.1
```



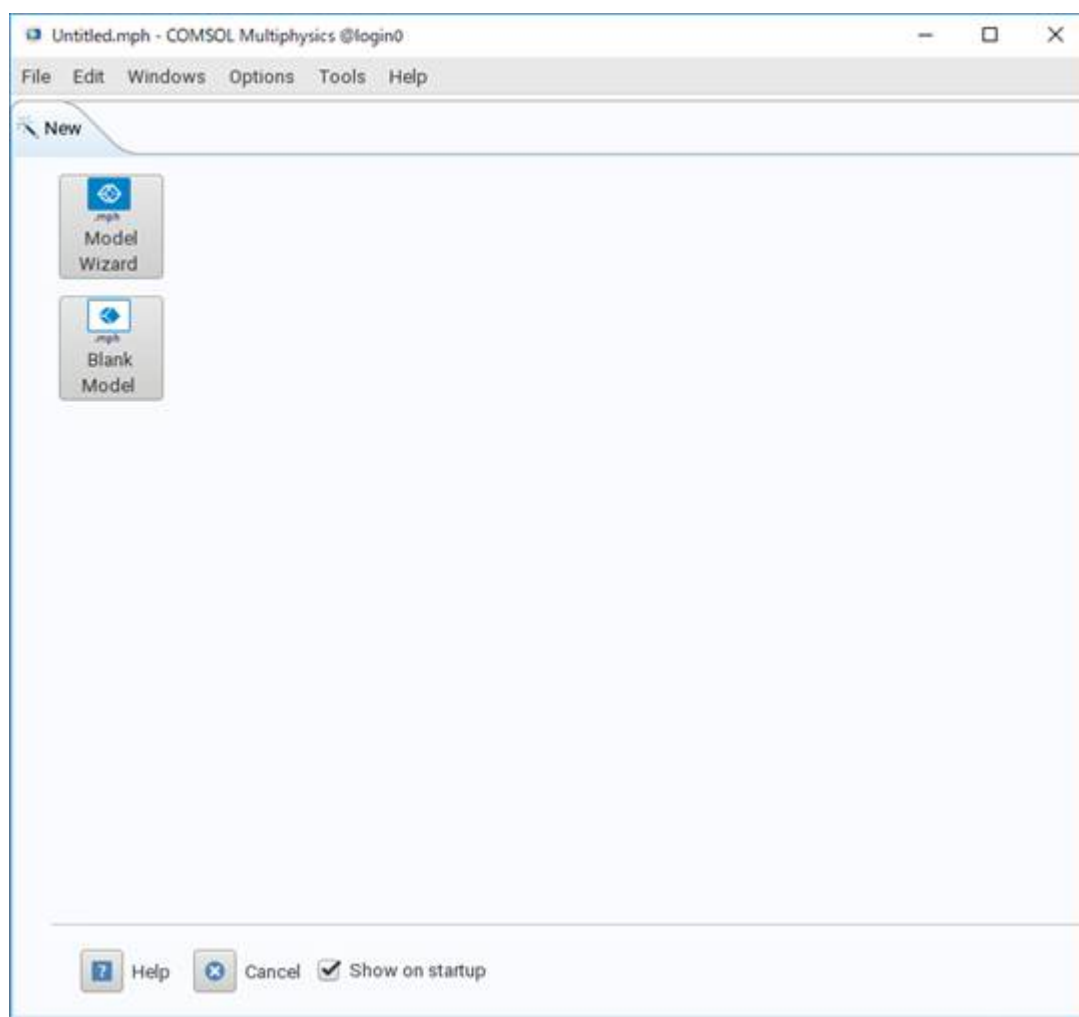
メニューバーのFile>Exit をクリックすると終了します。

## 6.19. COMSOL

COMSOLの利用手順を以下に示します。

```
$ module load comsol  
$ comsol
```





メニューバーの File > Exit をクリックすると終了します。

COMSOLのライセンス利用状況を以下のコマンドで確認できます。

```
$ lmtutil lmstat -S LMCOMSOL -c 27009@lice0:27009@remote:27009@t3ldapl
```

## 6.20. Schrodinger

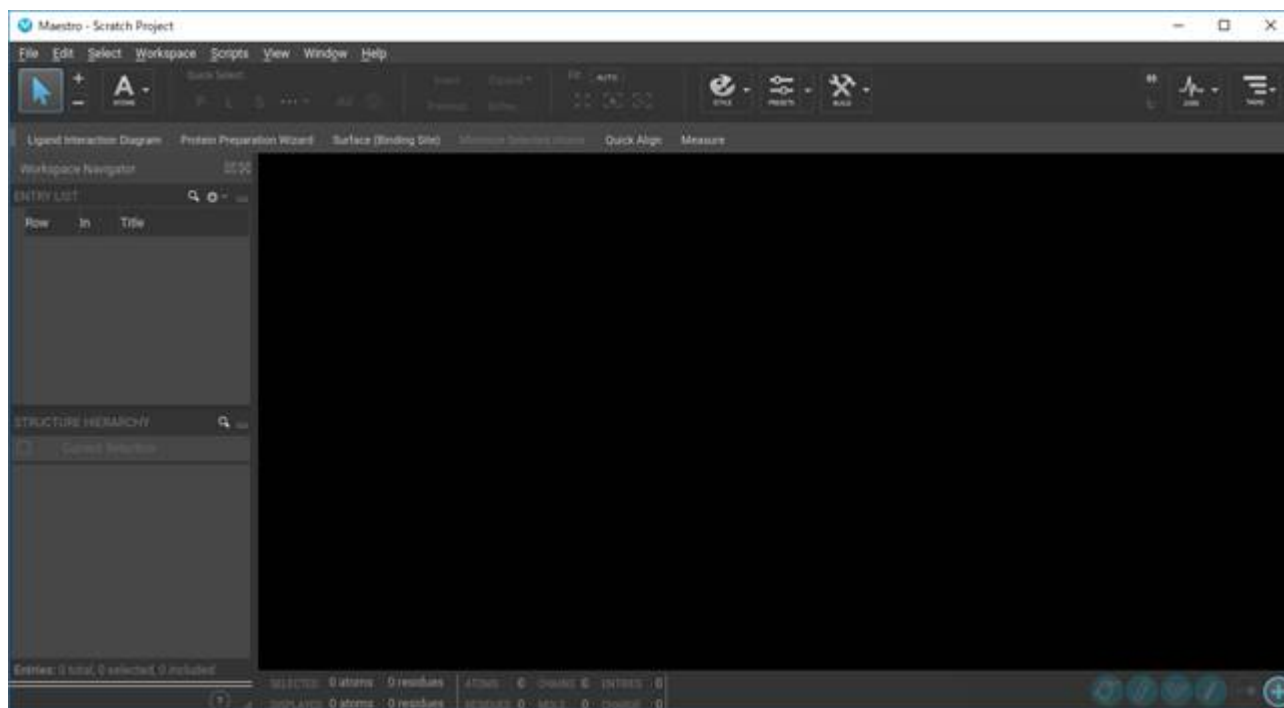
Schrodingerの利用手順を以下に示します。

LigprepのCLI実行例

```
$ module load schrodinger/Feb-17
SMILES形式の入力ファイルを使用し、MAE形式で出力する場合
$ ligprep -ismiinputfile -omaeoutputfile
```

GUIで利用する場合は、Maestroを起動します。

```
$ module load schrodinger/Feb-17
$ maestro
```



メニューバーのFile > Exit をクリックすると終了します。

Schrodingerのライセンス利用状況を以下のコマンドで確認できます。

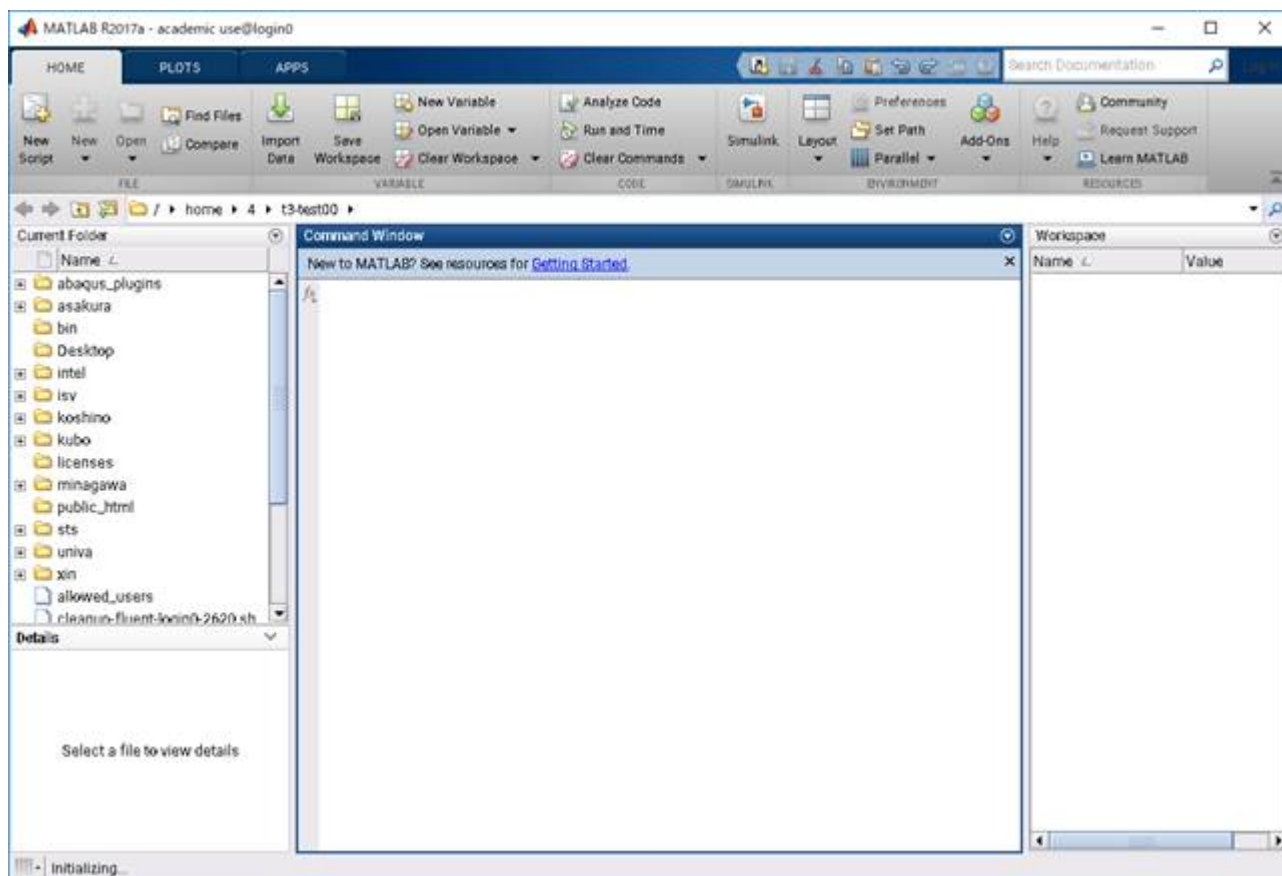
```
$ lmutil lmstat -S SCHROD -c 27010@lice0:27010@remote:27010@t3ldap1
```

## 6.21. MATLAB

MATLABは行列計算などの数値計算やデータの可視化をすることのできるアプリケーションです。

MATLABの利用方法の例を以下に示します。

```
$ module load matlab  
$ matlab
```



CLIでの使用手順について

```
$ module load matlab  
$ matlab -nodisplay
```

終了するにはexitを入力します。

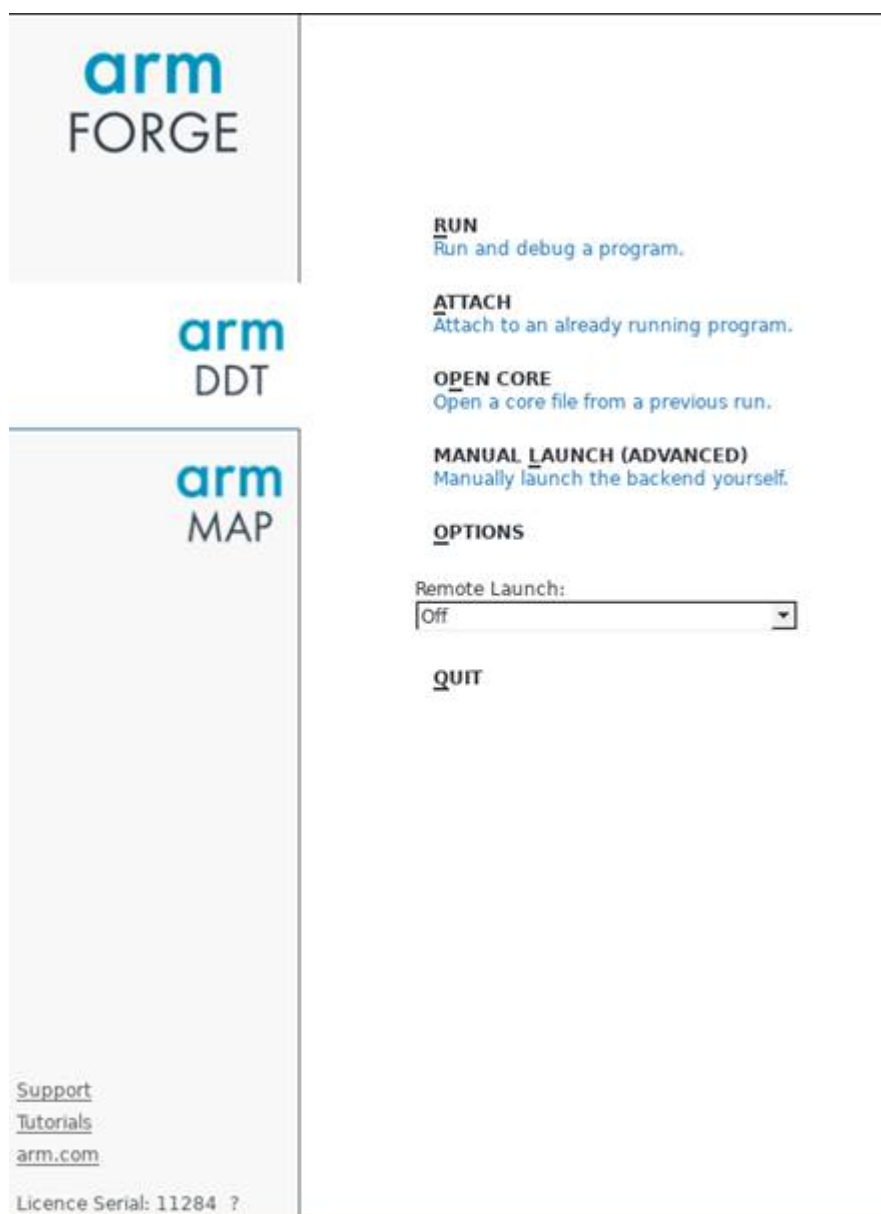
MATLABのライセンス利用状況を以下のコマンドで確認できます。

```
$ lmtutil lmstat -S MLM -c 27014@lice0:27014@remote:27014@t3ldapl
```

## 6.22. Arm Forge

Arm Forgeの利用方法を以下に示します。

```
$ module load forge  
$ forge
```



メニューバーの File > Exit をクリックすると終了します。

## 7. フリーウェア

フリーウェアの一覧表を以下に示します。

ソフトウェア名	概要
GAMESS	ソルバ・シミュレータ
Tinker	ソルバ・シミュレータ
GROMACS	ソルバ・シミュレータ
LAMMPS	ソルバ・シミュレータ
NAMMD	ソルバ・シミュレータ
QUANTUM ESPRESSO	ソルバ・シミュレータ
CP2K	ソルバ・シミュレータ
OpenFOAM	ソルバ・シミュレータ、可視化
CuDNN	GPUライブラリ
NCCL	GPUライブラリ
Caffe	DeepLearningフレームワーク
Chainer	DeepLearningフレームワーク
TensorFlow	DeepLearningフレームワーク
DeePMD-kit	MD用DeepLearningフレームワーク
R	インタプリタ(Rmpi,rpudに対応)
clang	コンパイラ
Apache Hadoop	分散データ処理ツール
POV-Ray	可視化
ParaView	可視化
VisIt	可視化
turbovnc	リモートGUI(X11) 表示
gnuplot	データ可視化
Tgif	画像表示・編集
GIMP	画像表示・編集
ImageMagick	画像表示・編集
TeX Live	TeX ディストリビューション
Java SDK	開発環境
PETSc	リニアシステムソルバ、ライブラリ
FFTW	高速フーリエ変換ライブラリ
DMTCP	チェックポイント・リスタート
Singularity	Linux container for HPC

## 7.1. 量子化学/MD関連ソフトウェア

### 7.1.1. GAMESS

GAMESSはオープンソースの第一原理分子量子化学計算アプリケーションです。

バッチキューシステムを利用したGAMESSの利用方法の例を以下に示します。

```
#!/bin/bash
#$ -cwd
#$ -l f_node=1
#$ -l h_rt=0:10:0
#$ -N gamess
. /etc/profile.d/modules.sh

module load intel intel-mpi gamess
cat $PE_HOSTFILE | awk '{print $1}' > $TMPDIR/machines
cd $GAMESS_DIR
./rungms exam08 mpi 4 4
```

詳細な説明は以下に記載されています。

<http://www.msg.ameslab.gov/gamess/index.html>

### 7.1.2. Tinker

Tinkerはバイオポリマーの為の特別な機能を備えた、分子動力学の為のモデリングソフトウェアです。

バッチキューシステムを利用したTinkerの利用方法の例を以下に示します。

```
#!/bin/bash
#$ -cwd
#$ -l f_node=1
#$ -l h_rt=0:10:0
#$ -N tinker
. /etc/profile.d/modules.sh

module load intel tinker
cp -rp $TINKER_DIR/example $TMPDIR
cd $TMPDIR/example
dynamic waterbox.xyz -k waterbox.key 100 1 1 2 300
cp -rp $TMPDIR/example $HOME
```

詳細な説明は以下に記載されています。

<https://dasher.wustl.edu/tinker/>

### 7.1.3. GROMACS

GROMACSは分子動力学シミュレーションとエネルギー最小化を行う為のエンジンです。

バッチキューシステムを利用したGROMACSの利用方法の例を以下に示します。

```
#!/bin/bash
#$ -cwd
#$ -l f_node=1
#$ -l h_rt=0:10:0
#$ -N gromacs
. /etc/profile.d/modules.sh

module load cuda/11.2.146 intel-mpi python/3.11.2 gcc/10.2.0 gromacs
cp -rp $GROMACS_DIR/examples/water_GMX50_bare.tar.gz $TMPDIR
cd $TMPDIR
tar xf water_GMX50_bare.tar.gz
cd water-cut1.0_GMX50_bare/3072
gmx_mpi grompp -f pme.mdp
OMP_NUM_THREADS=2 mpiexec.hydra -np 4 gmx_mpi mdrun
cp -rp $TMPDIR/water-cut1.0_GMX50_bare $HOME
```

詳細な説明は以下に記載されています。

<http://www.gromacs.org/>

### 7.1.4. LAMMPS

LAMMPSは液状、固体状、気体状の粒子の集団をモデル化する古典分子動力学コードです。

バッチキューシステムを利用したLAMMPSの利用方法の例を以下に記します。

```
#!/bin/bash
#$ -cwd
#$ -l f_node=1
#$ -l h_rt=0:10:0
#$ -N lammps
. /etc/profile.d/modules.sh

module load intel cuda openmpi/3.1.4-opa10.10-t3 ffmpeg python/3.11.2 lammps
cp -rp $LAMMPS_DIR/examples/VISCOSITY $TMPDIR
cd $TMPDIR/VISCOSITY
mpirun -x PATH -x LD_LIBRARY_PATH -x PSM2_CUDA=1 -np 4 lmp -pk gpu 0 -in in.gk.2d
cp -rp $TMPDIR/VISCOSITY $HOME
```

詳細な説明は以下に記載されています。

<http://lammps.sandia.gov/>

### 7.1.5. NAMD

NAMDは、大規模な生体分子システムの高性能シミュレーション用にデザインされたオブジェクト指向の並列分子動力学コードです。

バッチキューシステムを利用したNAMDの利用方法の例を以下に記します。

```
#!/bin/bash
#$ -cwd
#$ -l f_node=1
#$ -l h_rt=0:10:0
#$ -N namd
. /etc/profile.d/modules.sh

module load cuda intel namd
cp -rp $NAMD_DIR/examples/stmv.tar.gz $TMPDIR
cd $TMPDIR
tar xf stmv.tar.gz
cd stmv
namd3 +idlepoll +p4 +devices 0,1,2,3 stmv.namd
cp -rp $TMPDIR/stmv $HOME
```



バージョン2以前ではコマンド名が `namd2` となっておりますので、古いバージョンを使う場合には `namd3` を `namd2` に置き換えて下さい。

詳細な説明は以下に記載されています。

<https://www.ks.uiuc.edu/Research/namd/3.0/ug/>

### 7.1.6. CP2K

CP2Kは固体、液体、分子、周期的、物質、結晶、生物系の原子シミュレーションを実行できる量子化学、固体物理ソフトウェアパッケージです。

バッチキューシステムを利用したCP2Kの利用方法の例を以下に記します。

```
#!/bin/bash
#$ -cwd
#$ -l f_node=1
#$ -l h_rt=0:10:0
#$ -N cp2k
. /etc/profile.d/modules.sh

module load cuda gcc openmpi/3.1.4-opa10.10-t3 cp2k
cp -rp $CP2K_DIR/benchmarks/QS $TMPDIR
cd $TMPDIR/QS
export OMP_NUM_THREADS=1
mpirun -x PATH -x LD_LIBRARY_PATH -x PSM2_CUDA=1 -np 4 cp2k.psm -i H2O-32.inp -o H2O-32.out
cp -rp $TMPDIR/QS $HOME
```

詳細な説明は、以下に記載されています。

<https://www.cp2k.org/>

### 7.1.7. QUANTUM ESPRESSO

QUANTUM ESPRESSOは第一原理電子構造計算と材料モデリングのためのスイートです。  
バッチキューシステムを利用したQUANTUM ESPRESSOの利用方法の例を以下に記します。

```
#!/bin/sh
#$ -cwd
#$ -l h_rt=00:10:00
#$ -l f_node=1
#$ -N q-e
. /etc/profile.d/modules.sh

module purge
module load cuda/10.2.89 pgc openmpi/3.1.4-opa10.10-t3 quantumespresso

cp -p $QUANTUMESPRESSO_DIR/test-suite/pw_scf/scf.in .
cp -p $QUANTUMESPRESSO_DIR/example/Si.pz-vbc.UPF .

mpirun -x ESPRESSO_PSEUDO=$PWD -x PATH -x LD_LIBRARY_PATH -x PSM2_CUDA=1 -x PSM2_GPUDIRECT=1 -np 4 pw.x < scf.in
```

詳細な説明は、以下に記載されています。

<https://www.quantum-espresso.org/>

## 7.2. CFD関連ソフトウェア

### 7.2.1. OpenFOAM

OpenFOAMはオープンソースの流体/連続体シミュレーションコードです。  
Foundation版(openfoam)とESI版(openfoam-esi)の2種類がインストールされています。  
バッチキューシステムを利用したOpenFOAMの利用方法の例を以下に記します。

```
#!/bin/bash
#$ -cwd
#$ -l f_node=1
#$ -l h_rt=0:10:0
#$ -N openform
. /etc/profile.d/modules.sh

module load cuda openmpi openfoam
mkdir -p $TMPDIR/$FOAM_RUN
cd $TMPDIR/$FOAM_RUN
cp -rp $FOAM_TUTORIALS .
cd tutorials/incompressible/icoFoam/cavity/cavity
blockMesh
icoFoam
paraFoam
```

ESI版OpenFOAMをご利用の場合は上記の `module load` の箇所を `module load cuda openmpi openfoam-esi` として下さい。

詳細な説明は以下に記載されています。

<https://openfoam.org/resources/>

<http://www.openfoam.com/documentation/>

## 7.3. GPU用数値計算ライブラリ

### 7.3.1. cuBLAS

cuBLASはGPUで動作するBLAS Basic Linear Algebra Subprograms ライブラリです。

利用方法

```
$ module load cuda
$ nvcc -gencode arch=compute_60,code=sm_60 -o sample sample.cu -lcublas
```



通常のC言語のプログラム中で、cuBLASを呼び出す場合、コンパイル時に-I、-L、-lで指定する必要があります。

```
$ module load cuda
$ gcc -o blas blas.c -I${CUDA_HOME}/include -L${CUDA_HOME}/lib64 -lcublas
```

### 7.3.2. cuSPARSE

cuSPARSEはNVIDIA GPU上で疎行列計算を行うためのライブラリです。

利用方法

```
$ module load cuda
$ nvcc -gencode arch=compute_60,code=sm_60 sample.cu -lcusparse -o sample
```

通常のC言語のプログラム中で、cuSPARSEを呼び出す場合、コンパイル時に-I、-L、-lで指定する必要があります。

```
$ module load cuda
$ g++ sample.c -lcusparse_static -I${CUDA_HOME}/include -L${CUDA_HOME}/lib64 -lcublas -lcudart_static -lpthread -ldl -o sample
```

### 7.3.3. cuFFT

cuFFTはNVIDIA GPU上で並列FFT(高速フーリエ変換)を行うためのライブラリです。

利用方法

```
$ module load cuda
$ nvcc -gencode arch=compute_60,code=sm_60 -o sample sample.cu -lcufft
```

通常のC言語のプログラム中で、cuFFTを呼び出す場合、コンパイル時に-I、-L、-lで指定する必要があります。

```
$ module load cuda
$ gcc -o blas blas.c -I${CUDA_HOME}/include -L${CUDA_HOME}/lib64 -lcufft
```

## 7.4. 機械学習、ビッグデータ解析関連ソフトウェア

### 7.4.1. CuDNN

CuDNNはGPUを用いたDeep Neural Networkの為のライブラリです。

CuDNNの利用方法を以下に記します。

```
$ module load cuda cudnn
```

### 7.4.2. NCCL

NCCLは複数GPUの為の集団通信ライブラリです。

NCCLの利用方法の例を以下に記します。

```
$ module load cuda nccl
```

### 7.4.3. Caffe

CaffeはオープンソースのDeepLearningフレームワークです。

Caffeの利用方法の例を以下に記します。

```
$ module load cuda nccl cudnn/6.0 intel caffe/1.0
```

詳細な説明は以下に記載されています。

<http://caffe.berkeleyvision.org/>

MKLを利用する際はコードの前半に `#define USE_MKL` を追記し、`$MKLROOT` 以下にある計算ライブラリを呼び出してください。

#### 7.4.4. Chainer

Chainerはフレキシブルなニューラルネットワークのフレームワークです。

Chainerの利用方法の例を以下に記します。

```
$ module load intel cuda nccl/2.2.13 cudnn/7.1 openmpi/2.1.2 chainer/4.3.0
```

詳細な説明は以下に記載されています。

<https://docs.chainer.org/en/stable/>

#### 7.4.5. TensorFlow

TensorFlowはデータフローグラフを用いた機械学習・AIのオープンソースのライブラリです。

TensorFlowの利用方法の例を以下に記します。

Python2.7の場合

```
$ module load python-extension
```

Python3.9.2の場合

```
$ module load python/3.9.2 cuda/11.2.146 cudnn/8.1 nccl/2.8.4 tensorflow
```

詳細な説明は以下に記載されています。

<https://www.tensorflow.org/>

#### 7.4.6. DeePMD-kit

DeePMD-kitはMD用機械学習フレームワークです。

DeePMD-kitのジョブスクリプトの例を以下に記します。

##### 7.4.6.1 DeePMD-kit + LAMMPS

###### 7.4.6.1.1. DEEPM-DKIT LAMMPS 1ノード

DeePMD-kit + LAMMPSのジョブスクリプト例(1ノード、4GPU)を以下に示します。

```
#!/bin/sh
#$ -l h_rt=6:00:00
#$ -l f_node=1
#$ -cwd

. /etc/profile.d/modules.sh
module purge
module load deepmd-kit/2.1.5 intel ffmpeg lammps/23jun2022_u2
module li 2>&1

# enable DeePMD-kit for lammps/23jun2022_u2
export LAMMPS_PLUGIN_PATH=$DEEPM-DKIT_DIR/lib/deepmd_lmp

# https://tutorials.deepmodeling.com/en/latest/Tutorials/DeePMD-kit/learnDoc/Handson-Tutorial%28v2.0.3%29.html

wget https://dp-public.oss-cn-beijing.aliyuncs.com/community/CH4.tar
tar xf CH4.tar

cd CH4/00.data
python3 <<EOF
import dpdata
import numpy as np
data = dpdata.LabeledSystem('OUTCAR', fmt = 'vasp/outcar')
print('# the data contains %d frames' % len(data))
# random choose 40 index for validation_data
```

```

index_validation = np.random.choice(200,size=40,replace=False)
# other indexes are training_data
index_training = list(set(range(200))-set(index_validation))
data_training = data.sub_system(index_training)
data_validation = data.sub_system(index_validation)
# all training data put into directory:"training_data"
data_training.to_deepmd_npy('training_data')
# all validation data put into directory:"validation_data"
data_validation.to_deepmd_npy('validation_data')
print('# the training data contains %d frames' % len(data_training))
print('# the validation data contains %d frames' % len(data_validation))
EOF

export PSM2_DEVICES="shm,self,hfi"

cd ../01.train
dp train input.json
dp freeze -o graph.pb
dp compress -i graph.pb -o graph-compress.pb
dp test -m graph-compress.pb -s ../00.data/validation_data -n 40 -d results

cd ../02.lmp
ln -s ../01.train/graph-compress.pb
lmp -i in.lammps

```

#### 7.4.6.1.2. DEEPM-D-KIT LAMMPS 2ノード

DeePMD-kit + LAMMPSのジョブスクリプト例(2ノード、8GPU)を以下に示します。

```

#!/bin/sh
#$ -l h_rt=12:00:00
#$ -l f_node=2
#$ -cwd

. /etc/profile.d/modules.sh
module purge
module load deepmd-kit/2.1.5 intel ffmpeg lammps/23jun2022_u2
module li 2>&1

# enable DeePMD-kit
export LAMMPS_PLUGIN_PATH=$DEEPM-D-KIT_DIR/lib/deepmd_lmp

# https://tutorials.deepmodeling.com/en/latest/Tutorials/DeePMD-kit/learnDoc/Handson-Tutorial%28v2.0.3%29.html

wget https://dp-public.oss-cn-beijing.aliyuncs.com/community/CH4.tar
tar xf CH4.tar

cd CH4/00.data
python3 <<EOF
import dpdata
import numpy as np
data = dpdata.LabeledSystem('OUTCAR', fmt = 'vasp/outcar')
print('# the data contains %d frames' % len(data))
# random choose 40 index for validation_data
index_validation = np.random.choice(200,size=40,replace=False)
# other indexes are training_data
index_training = list(set(range(200))-set(index_validation))
data_training = data.sub_system(index_training)
data_validation = data.sub_system(index_validation)
# all training data put into directory:"training_data"
data_training.to_deepmd_npy('training_data')
# all validation data put into directory:"validation_data"
data_validation.to_deepmd_npy('validation_data')
print('# the training data contains %d frames' % len(data_training))
print('# the validation data contains %d frames' % len(data_validation))
EOF

export PSM2_DEVICES="shm,self,hfi"

cd ../01.train
mpirun -x PATH -x LD_LIBRARY_PATH -x PYTHONPATH -x PSM2_CUDA=1 -x NCCL_BUFFSIZE=1048576 -npernode 4 -np 8 dp train input.json
dp freeze -o graph.pb
dp compress -i graph.pb -o graph-compress.pb
dp test -m graph-compress.pb -s ../00.data/validation_data -n 40 -d results

cd ../02.lmp
ln -s ../01.train/graph-compress.pb
mpirun -x PATH -x LD_LIBRARY_PATH -x PYTHONPATH -x LAMMPS_PLUGIN_PATH -x PSM2_CUDA=1 -npernode 4 -np 8 lmp -i in.lammps

```

#### 7.4.6.2 DeePMD-kit + GROMACS

DeePMD-kit + GROMACSのジョブスクリプト例(1ノード、4GPU)を以下に記します。

```

#!/bin/sh
#$ -l h_rt=8:00:00
#$ -l f_node=1
#$ -cwd

```

```

. /etc/profile.d/modules.sh
module purge
module load deepmd-kit/2.1.5 gromacs-deepmd/2020.2
module li 2>&1

export PSM2_DEVICES="shm,self,hfi"

cp -pr $DEEPMD_KIT_DIR/examples/examples/water .
cd water/se_e2_a

dp train input.json
dp freeze -o graph.pb
dp compress -i graph.pb -o graph-compress.pb
dp test -m graph-compress.pb -s ../data/data_3 -n 40 -d results

cd ../gmx
ln -s ../se_e2_a/graph-compress.pb frozen_model.pb
export GMX_DEEPMD_INPUT_JSON=input.json
gmx_mpi grompp -f md.mdp -c water.gro -p water.top -o md.tpr -maxwarn 3
gmx_mpi mdrun -deffnm md
gmx_mpi rdf -f md.tpr -s md.tpr -o md_rdf.xvg -ref "name OW" -sel "name OW"

```

詳細な説明は以下に記載されています。

<https://docs.deepmodeling.com/projects/deepmd/en/master/index.html>

### 7.4.7. R

Rはデータ解析とグラフィックスの為にインタプリタ型プログラミング言語です。

並列処理用にRmpi、GPU用にrpudがインストールされています。

Rの利用方法の例を以下に記します。

```

$ module load intel cuda openmpi r
$ mpirun -stdin all -np 2 R --slave --vanilla < test.R

```

### 7.4.8. clang

clangはLLVMバックエンドのC/C++コンパイラです。

clangでGPU offloadするバイナリを生成する例を以下に示します。

#### • Cの場合

```

$ module load cuda clang
$ clang -fopenmp -fopenmp-targets=nvptx64-nvidia-cuda --cuda-path=$CUDA_HOME -Xopenmp-target -march=sm_60 test.c

```

#### • C++の場合

```

$ module load cuda clang
$ clang++ -stdlib=libc++ -fopenmp -fopenmp-targets=nvptx64-nvidia-cuda --cuda-path=$CUDA_HOME -Xopenmp-target -march=sm_60 test.cxx -lc++abi

```

詳細な説明は以下に記載されています。

<https://clang.llvm.org/>

### 7.4.9. Apache Hadoop

Apache Hadoopソフトウェアライブラリは単純なプログラミングモデルを用いて大きなデータセットを分散処理する為のフレームワークです。

Apache Hadoopの利用方法の例を以下に記します。

```

$ module load jdk hadoop
$ mkdir input
$ cp -p $HADOOP_HOME/etc/hadoop/*.xml input
$ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.0.jar grep input output 'dfs[a-z.]+'
$ cat output/part-r-00000
1      dfsadmin

```

バッチキューシステムの場合の利用手順を以下に示します。

```

#!/bin/bash
#$ -cwd

```

```

#$ -l f_node=1
#$ -l h_rt=0:10:0
#$ -N hadoop
. /etc/profile.d/modules.sh

module load jdk hadoop
cd $TMPDIR
mkdir input
cp -p $HADOOP_HOME/etc/hadoop/*.xml input
hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.0.jar grep input output 'dfs[a-z.]+'
cp -rp output $HOME

```

## 7.5. 可視化関連ソフトウェア

### 7.5.1. POV-Ray

POV-Rayはフリーの光線追跡ソフトです。

POV-Rayの利用方法の例を以下に記します。

```

$ module load pov-ray
$ povray -benchmark

```

詳細な説明は以下に記載されています。

<http://www.povray.org/>

### 7.5.2. ParaView

ParaViewはオープンソース、マルチプラットフォームのデータ解析と可視化アプリケーションです。

ParaViewの利用方法の例を以下に記します。

```

$ module load cuda openmpi paraview
$ paraview

```

#### 7.5.2.1. 複数GPUを用いて可視化する場合

`paraview/5.10.0`、`paraview/5.10.0-egl`、を用いて複数ノードで複数GPUを用いて可視化することができます。

`paraview/5.10.0-egl` には `paraview` コマンドが含まれていないことにご注意ください。

以下は `f_node=2` で8GPUを使う例です。

#### • wrap.sh

```

#!/bin/sh

num_gpus_per_node=4
mod=$((OMPI_COMM_WORLD_RANK/num_gpus_per_node))

if [ $mod -eq 0 ];then
    export VTK_DEFAULT_EGL_DEVICE_INDEX=0
elif [ $mod -eq 1 ];then
    export VTK_DEFAULT_EGL_DEVICE_INDEX=1
elif [ $mod -eq 2 ];then
    export VTK_DEFAULT_EGL_DEVICE_INDEX=2
elif [ $mod -eq 3 ];then
    export VTK_DEFAULT_EGL_DEVICE_INDEX=3
fi

$*

```

#### • job.sh

```

#!/bin/sh
#$ -cwd
#$ -V
#$ -l h_rt=8:0:0
#$ -l f_node=2

. /etc/profile.d/modules.sh

module purge

```

```
module load cuda openmpi/3.1.4-openssl/1.1.1g-paraview/5.10.0-egl
mpirun -x PSM2_CUDA=1 -x PATH -x LD_LIBRARY_PATH -npernode 4 -np 8 ./wrap.sh pvserver
```

openmpi/3.1.4-openssl/1.1.1g-paraview/5.10.0-egl モジュールを使って `-x PSM2_CUDA=1` としないとGPUで可視化部分だけが実行されGPU部分では実行されないのをご確認ください。

`wrap.sh` に実行権限を忘れずに付与して下さい。( `chmod 755 wrap.sh` )

上記のジョブスクリプトで

```
qsub -g <グループ名> job.sh
```

を行い、ジョブを投入します。

`qstat` でジョブが流れているのを確認します。

```
yyyyyyyy@login0:-> qstat
job-ID      prior    name              user            state submit/start at   queue                          jclass                      slots ja-task-ID
-----
xxxxxxx    0.55354  job.sh            yyyyyyyy        r       05/31/2020 09:24:19  all.q@rXiYnZ                                     56
```

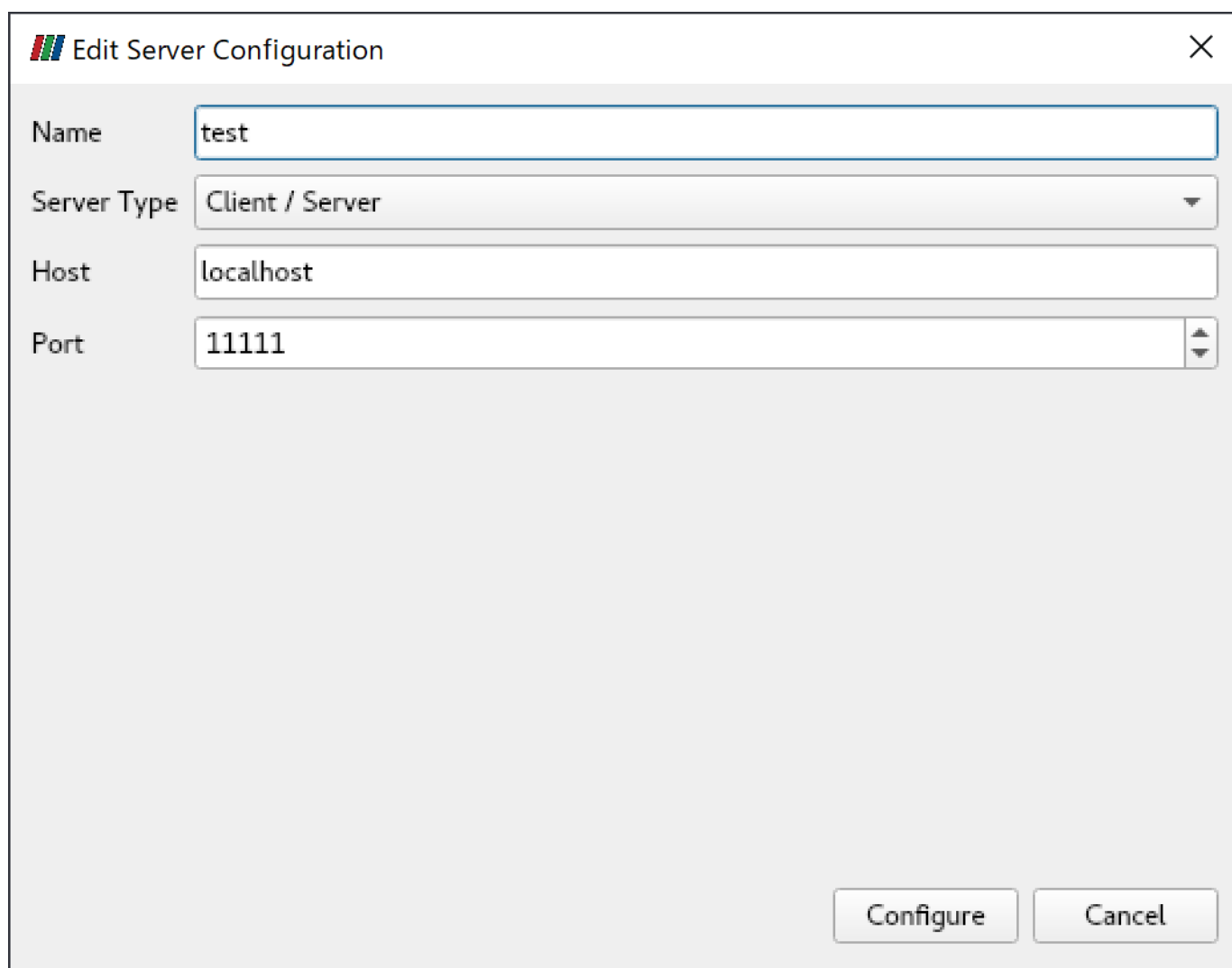
ジョブが流れているノードにX転送でsshし、paraviewを起動します。

```
yyyyyyyy@login0:-> ssh -CY rXiYnZ
yyyyyyyy@rXiYnZ:-> module load cuda openmpi paraview/5.10.0
paraview
```

※`turbovnc`を用いても可能です。

起動後、「File」->「Connect」をクリックし、「Add Server」をクリックします。

「Name」を適当に入力し(ここでは"test"とします)、「Configure」をクリックします。



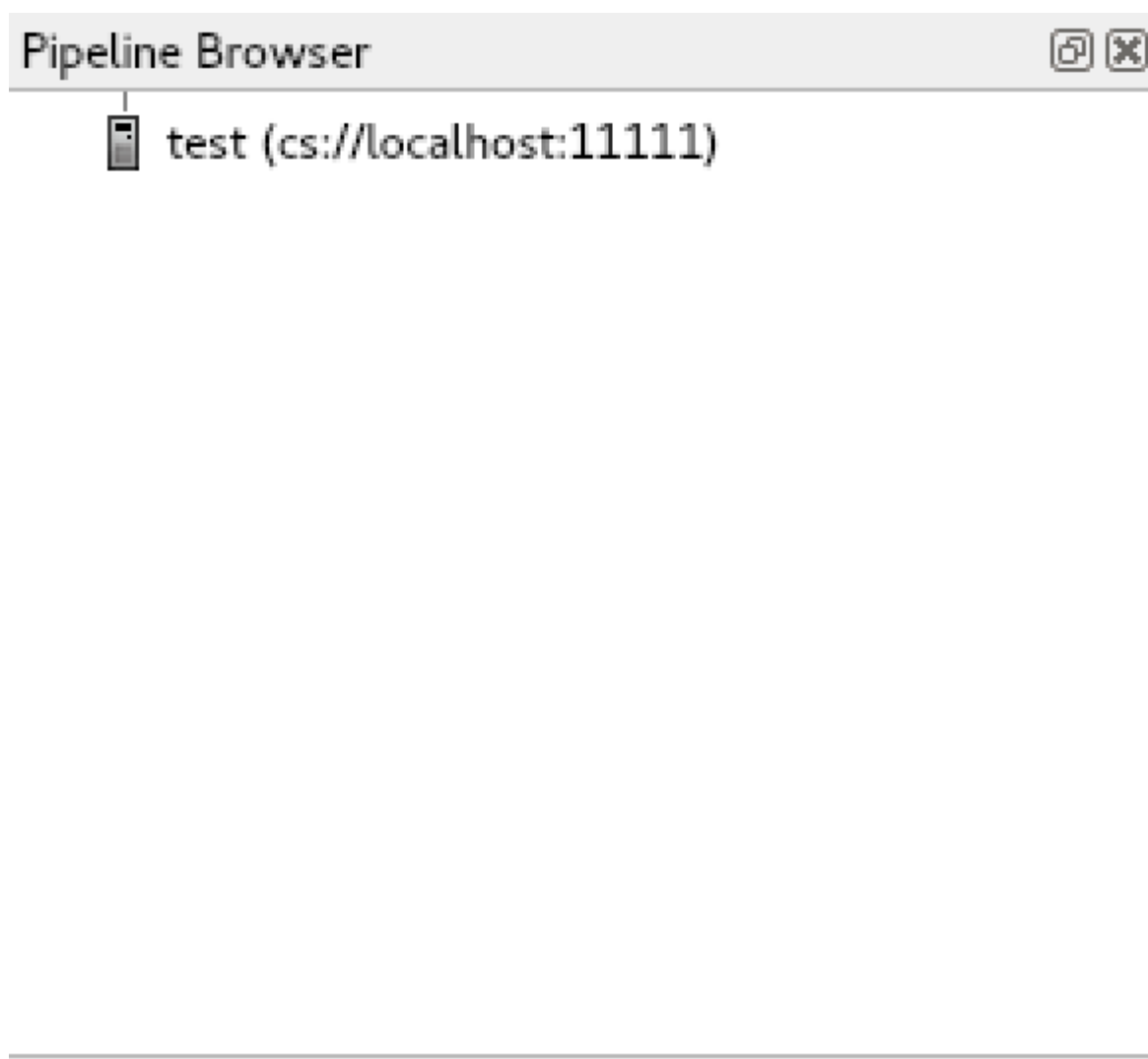
The image shows a dialog box titled "Edit Server Configuration" with a close button (X) in the top right corner. The dialog contains four input fields: "Name" with the value "test", "Server Type" with a dropdown menu showing "Client / Server", "Host" with the value "localhost", and "Port" with the value "11111". At the bottom right, there are two buttons: "Configure" and "Cancel".

Name	test
Server Type	Client / Server
Host	localhost
Port	11111

Configure Cancel

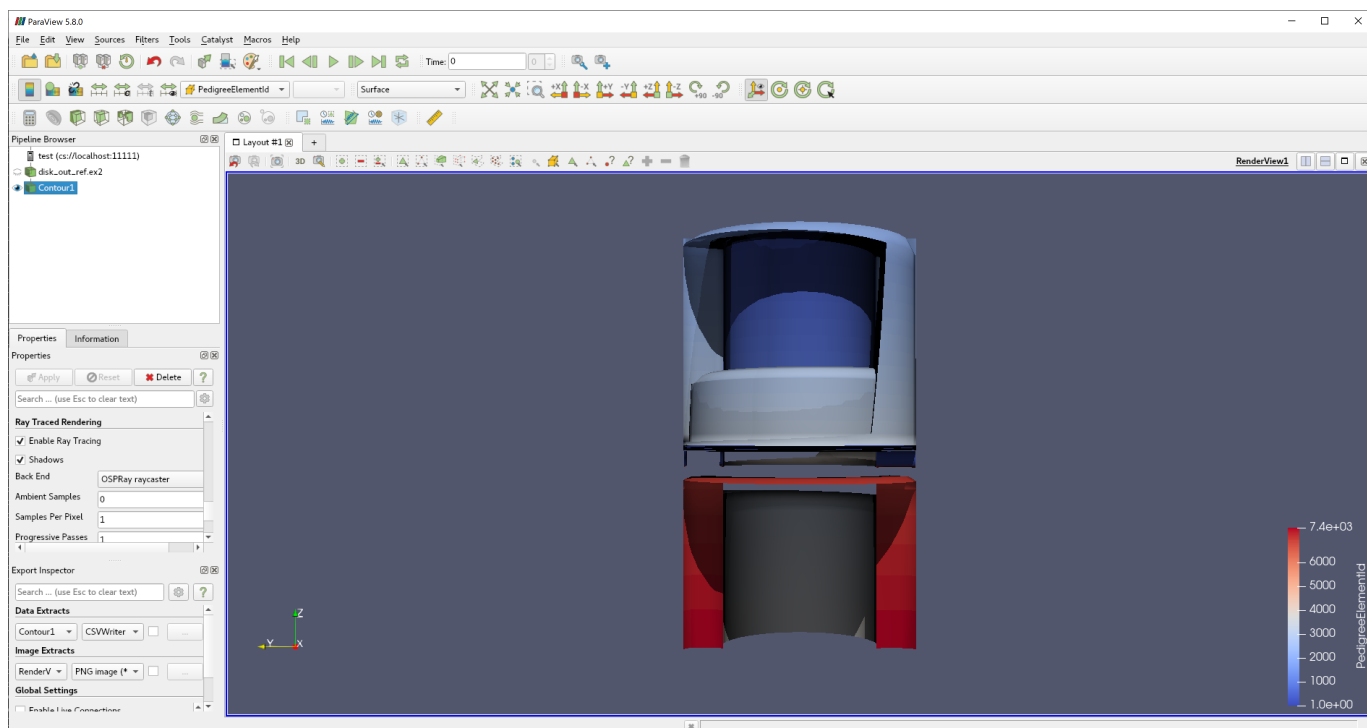
その後、「Connect」をクリックします。

接続されると、「Pipeline Browser」の項目に `test (cs://localhost:11111)` が表示されます。



paraviewのサンプルデータは[ここ](#)からダウンロードすることができます。





詳細な説明は以下に記載されています。

<https://www.paraview.org/>

### 7.5.3. VisIt

VisItはオープンソースの可視化アプリケーションです。

VisItの利用方法の例を以下に記します。

```
$ module load cuda openmpi vtk visit
$ visit
```

詳細な説明は以下に記載されています。

<https://wci.llnl.gov/simulation/computer-codes/visit/>

## 7.6. その他フリーウェア

### 7.6.1. turbovnc

turbovncはオープンソースのVNCソフトウェアです。

turbovncの使用法の例を以下に記します。 ※qrshで計算ノードを確保し計算ノード上で実行して下さい。

- ・計算ノードを確保する

```
$ qrsh -g <グループ名> -l <資源タイプ>=<個数> -l h_rt=<時間>
```

- ・確保した計算ノード上で以下を実行し、vncserverを起動する

```
$ module load turbovnc
$ vncserver

You will require a password to access your desktops.

Password: # <-- パスワードを聞かれるので設定する
Verify:
Would you like to enter a view-only password (y/n)? n
```

```
Desktop 'TurboVNC: rXiYnZ:1 ()' started on display rXiYnZ:1 # <-- このVNCのディスプレイ番号:1を覚えておく

Creating default startup script /home/n/xxxx/.vnc/xstartup.turbovnc
Starting applications specified in /home/n/xxxx/.vnc/xstartup.turbovnc
Log file is /home/n/xxxx/.vnc/rXiYnZ:1.log
```

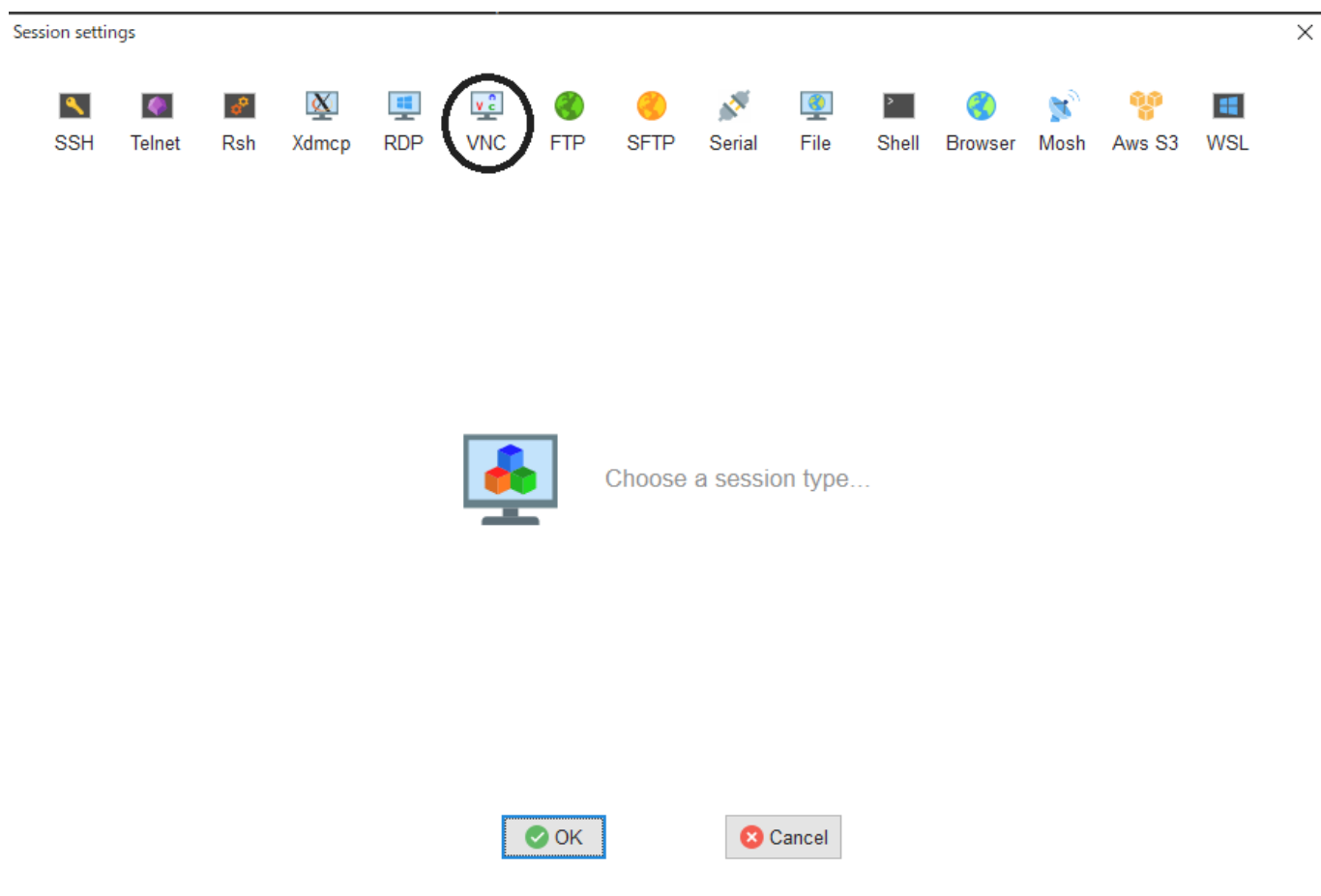
画面サイズを大きくしたい場合は `vncserver -geometry <WIDTH>x<HEIGHT>` としてサイズを指定します。

- その後<https://sourceforge.net/projects/turbovnc/files/>から自分のPC用のインストーラをダウンロードし、turbovnc viewerをインストールする
- 計算ノードに接続したターミナルソフトからSSHポート転送の設定でローカルのポート5901を計算ノードのポート5901にポート転送するように設定する(もしディスプレイ番号がrXiYnZ:nだった場合、ポート転送のポート番号は5900+nに設定する)
- 自分のPCからturbovnc viewerを起動し、localhost:5901に接続して設定したパスワードを入力する

#### 7.6.1.1. MobaXtermからVNCクライアントを利用する方法

MobaXtermにはVNCクライアントが内蔵されておりますので、VNCクライアントをインストールしなくてもVNC接続がご利用できます。

- qrshでノードを確保後、MobaXtermから「Sessions」->「New session」->「VNC」を選択する。



- その後、「Basic Vnc settings」の「Remote hostname or IP address」に確保した計算ノードのホスト名、「Port」を5900+nを入力し、「Network settings」の「Connect through SSH gateway(jump host)」をクリックし「Gateway SSH server」にlogin.t3.gsic.titech.ac.jpを入力、「Port」は22のまま、「User」に自分のTSUBAMEのログイン名を入力、「Use private key」にチェックを入れ自分の秘密鍵を入力する。

Session settings

SSH Telnet Rsh Xdmcp RDP VNC FTP SFTP Serial File Shell Browser Mosh Aws S3 WSL

Basic Vnc settings

Remote hostname or IP address \* rXiYnZ Port 5901

Advanced Vnc settings Network settings Bookmark settings

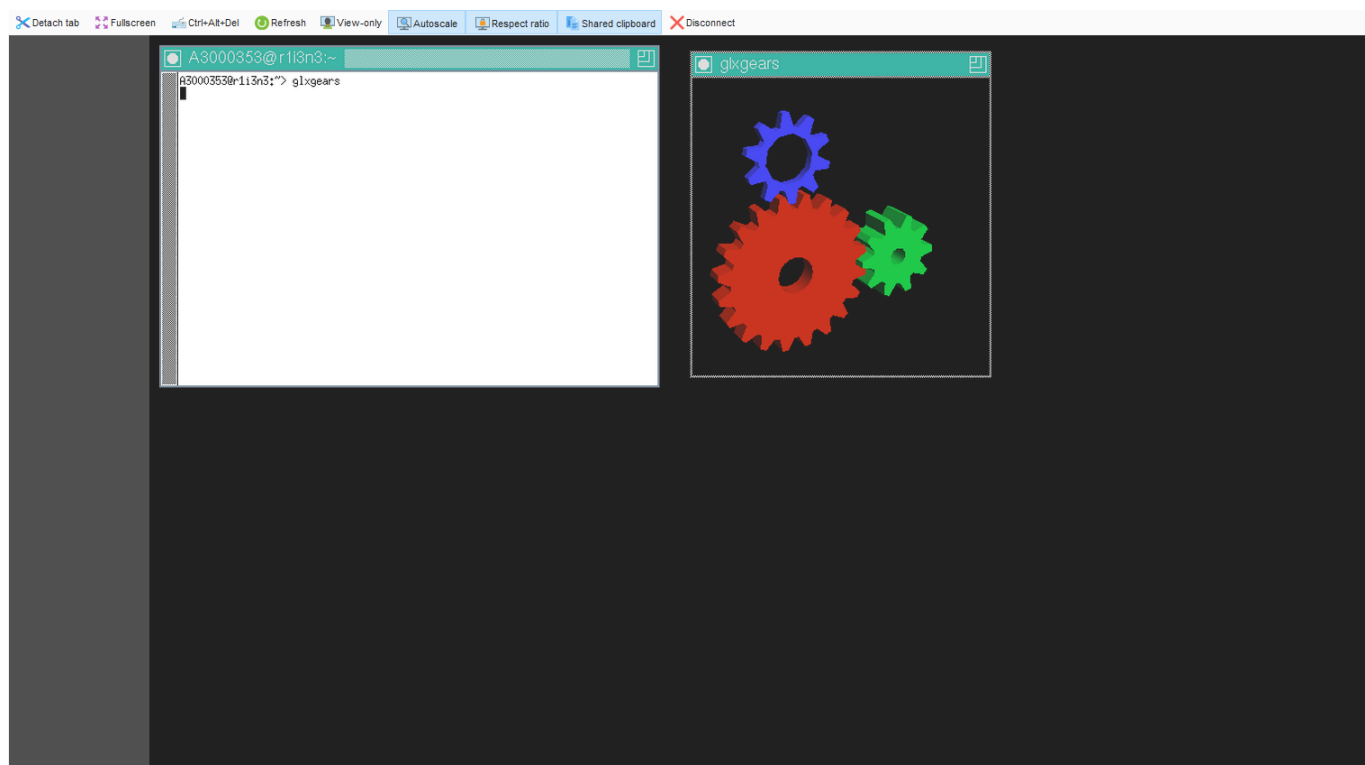
☒ Connect through SSH gateway (jump host)

Gateway SSH server login.t3.gsic.tite Port 22 User XXXXX

☒ Use private key

OK Cancel

OKをクリックするとVNCクライアントが起動します。



### 7.6.1.2. turbovnc + VirtualGL

turbovnc用時に、GPUを1つ以上確保する資源タイプ(s\_gpu, q\_node, h\_node, f\_node)を用いている場合、VirtualGLを使用してGPUを用いて可視化することができます。

一部のアプリケーションではX転送でも通常のVNCでも描画に失敗するものがあります(例えばGpuTestやUNIGINE等)が、そのようなアプリケーションを実行したい場合はVirtualGLをお試し下さい。

例として、s\_gpuの場合のVirtualGLの使用例を以下に示します。

```
$ qvnc ... -l s_gpu=1
$ . /etc/profile.d/modules.sh
$ module load turbovnc
$ Xorg -config xorg_vgl.conf :99 & # :99はVirtualGL用の任意のディスプレイ番号
$ vncserver
```



VirtualGLに使うディスプレイ番号はVNCで割り当てられたディスプレイ番号とは異なるので注意してください。

もし既に選択したディスプレイ番号が他のユーザに使われてしまっている場合、Xorgの実行で以下のようなエラーになります。

```
user@r7i7n7:~> Xorg -config xorg_vgl.conf :99 &
user@r7i7n7:~> (EE)
Fatal server error:
(E) Server is already active for display 99
      If this server is no longer running, remove /tmp/.X99-lock
      and start again.
(E)
(E)
Please consult the The X.Org Foundation support
      at http://wiki.x.org
      for help.
(E)
```

この場合は、:99を:100にするなどして他のユーザに使われていないディスプレイ番号を割り当てる用になってください。

- VNCクライアントで接続し、以下を実行

```
$ vglrun -d :99 <OpenGLアプリケーション> # :99は上のXorgで設定したVirtualGL用のディスプレイ番号
```

複数GPUを確保していて、2番目以降のGPUを使用したい場合はディスプレイ番号にスクリーン番号を追加します。

```
$ vglrun -d :99.1 <OpenGLアプリケーション> # :99は上のXorgで設定したVirtualGL用のディスプレイ番号、.1はスクリーン番号
```

上記でスクリーン番号を.2にするとGPU3番、.3にするとGPU4番が使われます。

- GpuTestをVirtualGLを用いて実行する例

```
vglrun -d :99 ./start_pixmark_piano_benchmark_fullscreen_1920x1080.sh
```



## 7.6.2. gnuplot

gnuplotはコマンドラインのインタラクティブなグラフ描画プログラムです。

標準機能に加え、X11、latex、PDFlib-lite、Qt4に対応するようにビルドされています。

gnuplotの利用方法の例を以下に記します。

```
$ module load gnuplot
$ gnuplot
```

## 7.6.3. Tgif

tgifはオープンソースの描画ツールです。

tgifの利用方法を以下に記します。

```
$ module load tgif
$ tgif
```

※Cannot open the Default(Msg) Font '*-courier-medium-r-normal--14-----\*iso8859-1*'.というエラーが出て起動しない場合は、`~/Xdefaults`に以下の行を追加して下さい。

```
Tgif.DefFixedWidthFont:      *-fixed-medium-r-semicondensed--13-*-*-*-*-*
Tgif.DefFixedWidthRulerFont: *-fixed-medium-r-semicondensed--13-*-*-*-*-*
Tgif.MenuFont:               *-fixed-medium-r-semicondensed--13-*-*-*-*-*
Tgif.BoldMsgFont:            *-fixed-medium-r-semicondensed--13-*-*-*-*-*
Tgif.MsgFont:                *-fixed-medium-r-semicondensed--13-*-*-*-*-*
```

### 7.6.4. GIMP

GIMPはオープンソースの画像操作プログラムです。

GIMPの利用方法の例を以下に記します。

```
$ module load gimp
$ gimp
```

### 7.6.5. ImageMagick

ImageMagickは画像処理ツールです。

標準機能に加え、X11、HDRI、libwmf、jpegに対応するようにビルドされています。

ImageMagickの利用方法の例を以下に記します。

```
$ module load imagemagick
$ convert -size 48x1024 -colorspace RGB 'gradient:#000000-#ffffff' -rotate 90 -gamma 0.5 -gamma 2.0 result.jpg
```

### 7.6.6. pLaTeX2e

pLaTeX2eは日本語化されたLaTeX2eの一つです。

pLaTeX2eの利用方法の例を以下に記します。

```
$ module load texlive
$ platex test.tex
$ dvipdfmx test.dvi
```

※pdfの作成にはdvipdfmxをご利用ください。dvipdfでは日本語が正常に変換されません。

### 7.6.7. Java SDK

Java SDKとして、Oracle JDK 1.8がインストールされています。

Java SDKの利用方法の例を以下に記します。

```
$ module load jdk
$ javac Test.java
$ java Test
```

### 7.6.8. PETSc

PETScはオープンソースの並列数値計算ライブラリです。線型方程式の求解等を行うことができます。

実数用、複素数用の2種類がインストールされています。

PETScの利用方法の例を以下に記します。

```
$ module load intel intel-mpi
$ module load petsc/3.7.6/real      ← 実数用
又は
$ module load petsc/3.7.6/complex  ← 複素数用
$ mpiifort test.F -lpetsc
```

### 7.6.9. FFTW

FFTWはオープンソースの高速フーリエ変換用ライブラリです。

FFTW 2x系列と3x系列は非互換な為、バージョン2系と3系の2種類がインストールされています。

FFTWの利用方法の例を以下に記します。

```
$ module load intel intel-mpi fftw      ← Intel MPIの場合
又は
$ module load intel cuda openmpi fftw   ← Open MPIの場合
$ ifort test.f90 -lfftw3
```

## 7.6.10. DMTCP

DMTCPはマルチノード・マルチスレッド対応のチェックポイントツールです。

DMTCPの利用方法の例を以下に記します。

### ・チェックポイントを作成する場合

```
#!/bin/sh
# 他の指定については記載を省略
module load dmtcp
export DMTCP_CHECKPOINT_DIR=<イメージの保存先>
export DMTCP_COORD_HOST=`hostname`
export DMTCP_CHECKPOINT_INTERVAL=<チェックポイント取得間隔>
dmtcp_coordinator --quiet --exit-on-last --daemon 2>&1 # DMTCPの実行
dmtcp_launch ./a.out # DMTCP経由でプログラムの実行
```

### ・作成されたチェックポイントからリスタートする場合

```
#!/bin/sh
# 他の指定については記載を省略
module load dmtcp
export DMTCP_CHECKPOINT_DIR=<イメージの保存先>
export DMTCP_COORD_HOST=`hostname`
export DMTCP_CHECKPOINT_INTERVAL=<チェックポイント取得間隔>
# DMTCP_CHECKPOINT_INTERVALの間隔でDMTCP_CHECKPOINT_DIRにチェックポイントが作成されます。
# dmtcp_restart_script.shスクリプトでチェックポイントからプログラムをリスタートできます。
$DMTCP_CHECKPOINT_DIR/dmtcp_restart_script.sh # イメージからのリスタート
```

DMTCPについては以下のページを参照ください。

<http://dmtcp.sourceforge.net/>

## 7.6.11. Singularity

SingularityはHPC向けLinuxコンテナです。

Singularityの使い方の例を以下に記します。

※qcrshでノードを確保した後に実行して下さい。

### シェルを起動する場合

```
$ module load singularity
$ cp -p $SINGULARITY_DIR/image_samples/centos/centos7.6-opa10.9.sif .
$ singularity shell --nv -B /gs -B /apps -B /scr centos7.6-opa10.9.sif
```

### コンテナ内のコマンドを実行する場合

```
$ module load singularity
$ cp -p $SINGULARITY_DIR/image_samples/centos/centos7.6-opa10.9.sif .
$ singularity exec --nv -B /gs -B /apps -B /scr centos7.6-opa10.9.sif <コマンド>
```

### MPIを実行する場合

```
$ module load singularity cuda openmpi
$ cp -p $SINGULARITY_DIR/image_samples/centos/centos7.6-opa10.9.sif .
$ mpirun -x LD_LIBRARY_PATH -x SINGULARITYENV_LD_LIBRARY_PATH=$LD_LIBRARY_PATH -x SINGULARITYENV_PATH=$PATH -x <環境変数> -np nnode <プロセス数/ノード> -np <プロセス数> singularity exec --nv -B /apps -B /gs -B /scr/ centos7.6-opa10.9.sif <MPI実行バイナリ>
```

singularity/3.4.2以降では、`fakeroot` オプションを使ってユーザ権限でコンテナを編集することができます。



Singularityがfakerootオプションに対応したのは3.3.0ですが、TSUBAMEの環境に起因する問題があり、3.4.1までのSingularityでは同オプションが正常に動作しません。fakerootオプションでイメージを編集するには計算ノードにて `$T3TMPDIR` 上で実行する必要があります。

以下にcentosのイメージにvimをインストールする例を記します。

```
$ cd $T3TMPDIR
$ module load singularity
$ singularity build -s centos/ docker://centos:latest
INFO: Starting build...
Getting image source signatures
...
$ singularity shell -f -w centos # -f がfakerootオプションです
Singularity> id
uid=0(root) gid=0(root) groups=0(root)
Singularity> unset TMPDIR # "Cannot create temporary file - mkstemp: No such file or directory"のエラーの回避策です
Singularity> yum install -y vim
Failed to set locale, defaulting to C.UTF-8
CentOS-8 - AppStream          6.6 MB/s | 5.8 MB    00:00
CentOS-8 - Base              5.0 MB/s | 2.2 MB    00:00
CentOS-8 - Extras
...
Installed:
  gpm-libs-1.20.7-15.el8.x86_64          vim-common-2:8.0.1763-13.el8.x86_64  vim-enhanced-2:8.0.1763-13.el8.x86_64
  vim-filesystem-2:8.0.1763-13.el8.noarch  which-2.21-12.el8.x86_64

Complete!
Singularity> which vim
/usr/bin/vim
Singularity> exit
$ singularity build -f centos.sif centos/
INFO: Starting build...
INFO: Creating SIF file...
INFO: Build complete: centos.sif
$ singularity shell centos.sif
Singularity> which vim
/usr/bin/vim # <--- vimがインストールされています
```

コンテナイメージにCUDA版OPAドライバライブラリをインストールする方法(centos7.5にCUDA版OPA10.9.0.1.2をインストールする場合)

※システムメンテナンスによってOPAが更新されることがございますので、新しいバージョンのOPAに更新された場合はは適時バージョンを変更して下さい。

OPAのバージョンは以下で確認が可能です。

```
$ rpm -qa |grep opaconfig
opaconfig-10.9.0.1-2.x86_64
```

[このリンク](#)から対応するOSのOPAのインストーラをダウンロードしてください

```
$ module load singularity/3.4.2
$ cp -p IntelOPA-IFS.RHEL75-x86_64.10.9.0.1.2.tgz ~
$ singularity build -s centos7.5/ docker://centos:centos7.5.1804
$ find centos7.5/usr/ -mindepth 1 -maxdepth 1 -perm 555 -print0 |xargs -0 chmod 755 # イメージ内の一部のファイル/ディレクトリに書き込み権限がないので追加します
$ singularity shell -f -w centos7.5
Singularity centos7.5:> tar xf IntelOPA-IFS.RHEL75-x86_64.10.9.0.1.2.tgz
Singularity centos7.5:> cd IntelOPA-IFS.RHEL75-x86_64.10.9.0.1.2/IntelOPA-OFA_DELTA.RHEL75-x86_64.10.9.0.1.2/RPMS/redhat-ES75/CUDA/
Singularity centos7.5:> yum install -y numactl-libs hwloc-libs libfabric libibverbs infinipath-psm
Singularity centos7.5:> rpm --force -ivh libpsm2-*.rpm
Singularity centos7.5:> exit
$ find centos7.5/usr/bin -perm 000 -print0 |xargs -0 chmod 755 # 上記yum install後、なぜかパーミッションが000のファイルが/usr/binにインストールされていますので変更します
$ singularity build centos7.5.sif centos7.5/
```

上記ではIFS版を用いていますが、BASIC版をダウンロードしても構いません。

詳細な説明は以下に記載されています。

<https://sylabs.io/docs/>

apptainerを使用する場合は `module load apptainer` とし、コマンドの `singularity` の部分を `apptainer` に変更してください。



## 7.6.12. Alphafold

Alphafoldは機械学習を用いたタンパク質構造予測プログラムです。  
Alphafoldを利用する例を以下に示します。

- ・初期設定 (ログインノードもしくは計算ノード)

```
module purge
module load cuda/11.0.3 alphafold/2.0.0

cp -pr $ALPHAFOLD_DIR .
cd alphafold
git pull # 最新版に更新
# 特定のバージョンを使用したい場合 (この場合はv2.0.1) は以下を追加してください。
# git checkout -b v2.0.1 v2.0.1
```

- ・実行時 (alphafold/2.0.0のジョブスクリプトの例)

```
#!/bin/sh
#$ -l h_rt=24:00:00
#$ -l f_node=1
#$ -cwd

. /etc/profile.d/modules.sh
module purge
module load cuda/11.0.3 alphafold/2.0.0
module li

cd alphafold
./run_alphafold.sh -a 0,1,2,3 -d $ALPHAFOLD_DATA_DIR -o dummy_test/ -m model_1 -f ./example/query.fasta -t 2020-05-14
```

- ・実行時 (alphafold/2.1.1のジョブスクリプトの例)

```
#!/bin/sh
#$ -l h_rt=24:00:00
#$ -l f_node=1
#$ -cwd

. /etc/profile.d/modules.sh
module purge
module load cuda/11.0.3 alphafold/2.1.1
module li

cd alphafold
./run_alphafold.sh -a 0,1,2,3 -d $ALPHAFOLD_DATA_DIR -o dummy_test/ -f ./example/query.fasta -t 2020-05-14
```

2.2.0 の場合は alphafold/2.1.1 の `module load cuda/11.0.3 alphafold/2.1.1` の部分を `module load cuda/11.0.3 alphafold/2.2.0` に変えて下さい。

データベースファイルの容量が大きいため、可能な限り個別にダウンロードすることはお避け下さい。

Alphafoldの詳細な説明は以下をご参照下さい。

<https://github.com/deepmind/alphafold>



## 改訂履歴

---

改定日付	内容
2024/01/23	「5.2.3.3. MPI並列」のMPTの例を修正
2023/10/18	「7.1.3. GROMACS」の例を修正
2023/08/18	「7.1.3. GROMACS」の例を修正
2023/07/10	「7.1.4. LAMMPS」の例を修正
2023/04/14	「7.1.5. NAMD」をNAMD 3.0b2用に修正
2023/04/06	「7.6.11. Singularity」にapptainerの使い方を追加
2023/03/15	「7.4.8. clang」にC++の場合を追加
2022/12/27	「7.4.6. DeePMD-kit」の例の修正
2022/12/22	「6. ISVアプリケーション」の修正
2022/12/16	「7.4.6.1.2. DeePMD-kit LAMMPS 2ノード」の追加
2022/12/13	「7.4.6. DeePMD-kit」の追加
2022/12/07	「2.2. ログイン方法」のロードバランサの記述を削除
2022/09/09	「7.1.4. LAMMPS」のジョブスクリプト例を修正
2022/08/23	「7.1.7. QUANTUM ESPRESSO」のジョブスクリプト例を修正
2022/05/31	「7.6.12. Alphafold」にalphafold/2.2.0の実行例を追加
2022/04/12	「7.1.5. NAMD」の実行例を修正
2022/04/05	バージョンアップによる雑多な修正
2022/01/26	「7.6.12. Alphafold」の非商用ライセンス関係の記載を削除
2022/01/25	「5.2. ジョブの投入」に補足事項を追記
2021/12/13	「7.6.12. Alphafold」に2.1.1を追加
2021/11/12	「7.1.7. QUANTUM ESPRESSO」を追加
2021/11/04	「7.6.12 Alphaold」の例にバージョン変更方法を追記
2021/09/29	「2.5. TSUBAMEポイントの確認」の更新
2021/08/10	「7.6.12. Alphafold」に <code>git pull</code> できる旨を追加
2021/08/18	「5.4.3. インタラクティブキュー」の利用者に関する記載を更新
2021/08/01	「7.6.12. Alphafold」を追加
2021/04/06	2021/04のメンテナンスのための各種修正 (「5.4.3. インタラクティブキュー」ほか)
2021/01/20	「5.2.3.5. コンテナの利用」に1コンテナの場合の注意書きを追記
2020/11/17	「7. フリーウェア」にsingularityを追加、「7.6.11. Singularity」の実行例の修正
2020/11/17	「7.6.1. turbovnc」にMobaXtermでの実行例を追加、VirtualGLの例を追加
2020/11/06	「5.2.5. ジョブの状態確認」にRqとRrの説明を追加
2020/10/09	「7.6.11. Singularity」のfakerootの実行例の修正
2020/09/24	「5.2.3.4. プロセス並列/スレッド並列(ハイブリッド, MPI+OpenMP)」のIntel MPI/OpenMPIの例を修正
2020/08/05	「2.2. ログイン方法」にデフォルトの鍵ペアの場合 <code>-i</code> オプションは省略可能な旨を追記
2020/06/01	「7.5.2. ParaVlew」の実行例に複数GPU版を追記

改定日付	内容
2020/05/25	「7.1.6. CP2K」の実行例を修正
2020/05/18	「7.1.4. LAMMPS」の実行例を修正
2020/04/30	「5.4.1. インタラクティブノードを利用したX転送」を修正
2020/04/22	「5.3. 計算ノードの予約」に <code>t3-user-info compute ars</code> を追記
2020/04/22	「4.6.5. GPUDirect RDMA」で <code>openmpi/2.1.2-opa10.9-t3</code> を <code>openmpi/3.1.4-opa10.10-t3</code> に変更
2020/04/09	「5.2.4.1. お試し実行」に <a href="https://www.t3.gsic.titech.ac.jp/node/129">https://www.t3.gsic.titech.ac.jp/node/129</a> をマージ
2020/04/08	「5.2.8. アレイジョブ」にタスク数をなるべく減らす旨を追記
2020/04/07	「5.1.1.1. 利用可能な資源タイプ」の <code>f_node</code> のメモリ使用量を変更
2020/02/28	「4.4. PGIコンパイラ」を追加
2020/02/05	「5.2.2. ジョブスクリプト」に <code>shebang</code> についての注意を追記
2020/01/17	「5.5. 計算ノードへのSSHログイン」に計算ノードへ <code>ssh</code> した際にプロセスを見る方法を追記
2019/12/16	「7.2.1. OpenFOAM」にFoundation版とESI版の2種類があることを記載
2019/12/06	「5.4.3. インタラクティブキュー」を修正
2019/12/03	「7.6.1.1. turbovc + VirtualGL」を追加
2019/11/29	「7.4.7. clang」を追加
2019/11/29	「7.6.1. turbovc」を追加
2019/11/12	「5.4.3. インタラクティブキュー」を追加
2019/10/21	「7.6.10. Singularity」の <code>fakeroot</code> オプションの例を修正
2019/10/04	「7.6.10. Singularity」に <code>fakeroot</code> オプションの例を追加
2019/09/27	「5.3. 計算ノードの予約」のリンクの修正
2019/08/02	「5. ジョブスケジューリングシステム」の章番号の見直し
2019/07/31	「5.2.3.1. シングルジョブ」の例を <code>f_node</code> から <code>s_core</code> に変更
2019/07/30	PGIコンパイラ利用の手引きから「GPU用数値計算ライブラリ」を7.3.に移行
2019/07/17	「5.4. チェックポイント」を削除、「7.5.9. DMTCPP」を追加
2019/07/10	「5.3. インタラクティブジョブの投入」ジョブ終了方法の記載を修正
2019/06/12	「5.8.5. ネットワーク系アプリケーションへの接続」を追加
2019/06/10	「4.5.3 CUDA対応のMPI」、「4.5.5 GPUDirect RDMA」の例をOPA10.9用に修正
2019/06/07	「5.2.2 ジョブスクリプト」の各セクションの説明を追加
2019/05/31	「2.2.1. ログインノードにおける高負荷プログラムの実行制限について」を記載
2019/05/30	ドキュメントシステムの変更 表現の微修正
2019/05/20	「5.5 予約実行」の実行制限について記述を修正
2019/05/15	「7.1.4 LAMMPS」、「7.1.5 NAMD」のリンクを修正
2019/03/29	「5.8 コンテナジョブ」の追加 「5.6.1 ローカルスラッシュ領域」に <code>\$T3TMPDIR</code> を追記

改定日付	内容
2019/01/25	章番号変更に伴う参照の修正
2019/01/25	「4.2.5 ジョブスクリプトの記述例MPI並列」「4.2.5 ジョブスクリプトの記述例プロセス並列/スレッド並列(ハイブリッド)」のOpenMPI項にライブラリの変数環境変数を加筆
2019/01/17	章の名称・順番・カテゴリーの変更
2019/01/16	「表4.3 qsubコマンドの主なオプション」の-mオプションに注意事項を追記 「4.6.1.Homeディレクトリ」にクォータ制限の注意事項を追記
2018/12/03	「3.5.6.GPUのCOMPUTE MODEの変更」を追記 「4.6.4.共有スクラッチ領域」を加筆
2018/10/26	「4.1. 利用可能な資源タイプ」の制限値についての記述を修正
2018/09/19	「5. ISVアプリケーション」を2018年9月時点の環境に合うように修正
2018/09/06	「6.5.9 singularity」を追加
2018/08/23	「6.3 Caffe,Chainer,Tensorflow」の利用手順を修正
2018/07/27	「5.8 Gaussian」「5.9 GaussView」の利用手順を修正
2018/05/25	「2.5 TSUBAMEポイントの確認」を追記 「4.6.1 Homeディレクトリ」,「4.6.2 高速ストレージ」の容量確認方法を加筆
2018/05/09	「3.1.1利用可能なmodule環境の表示」からモジュール一覧を削除、webページへのリンクを追加
2018/02/07	「2.4 ログインシェルの変更」を追記 「4.2.2 ジョブスクリプト」の優先度オプションを加筆
2018/01/13	「4.5 予約実行」を追記
2018/01/12	「2.2ログイン方法」を加筆 「4.3.1.インタラクティブノードを利用したX転送」を加筆
2017/12/28	「4.2ジョブ投入」にアレイジョブの項目を加筆 「4.4 シグナル通知/チェックポイント」を追記 「4.5 ストレージの利用」を加筆
2017/12/18	「3.1.1.利用可能なmodule環境の表示」を追記 「3.5 GPU環境」を加筆
2017/10/26	「5.12.4 ユーザー認証について」を削除
2017/10/05	「6.3.3.Caffe」にMKLの利用方法について追記
2017/09/25	「4.3.1.インタラクティブノードを利用したX転送」を追記
2017/09/14	「2.4ストレージサービス(CIFS)」にCIFSボリューム表示を加筆
2017/09/11	「2.2ログイン方法」にログインノードの制限を加筆 「2.4ストレージサービス(CIFS)」にCIFSボリューム表示を加筆 「4.4.4共有スクラッチ領域」のアクセス方法の追記
2017/09/06	改定第3版
2017/08/17	改定第2版
2017/07/31	初版