π-VizStudio User Manual

神戸大学 計算科学教育センター　著

2018-07-26 版　　発行

確認事項

π-CAVE やπ-VizStudio 利用については以下のマニュアル (このマニュアルを含む) を参照してください。

利用者向け

- π-CAVE π-VizStudio User Manual
  - π-VizStudio をπ-CAVE で利用するためのマニュアル
- π-VizStudio User Manual
  - π-VizStudio をスーパーコンピュータとして利用するためのマニュアル
- π-VizStudio 遠隔可視化 Manual
  - π-VizStudio のリソースを用いて遠隔可視化を行うためのマニュアル

管理者／内部関係者向け

- π-CAVE π-VizStudio Admin Manual
  - π-CAVE およびπ-VizStudio の管理者向けマニュアル
- Multiverse Manual
  - 没入型 VR 可視化ソフトウェアでデモンストレーションを行う際のマニュアル
- 3D Movie Manual
  - 計算科学教育センターの各設備で立体視動画を上映するためのマニュアル

簡易マニュアル（コンパイル例と実行用シェルスクリプト例）

主な変更点（2018 年 7 月 26 日改訂）

MPI 並列を行う場合、コンパイルコマンド (ifort,icc 等) に-lmpi のオプションをつける方法を案内していましたが，アップデートに伴い mpif90（Fortran）, mpicc(C), mpicxx(C++) を使うこととなりました。このコマンドにより、2018 年 4 月より不可能であった vizfront でコンパイルした実行ファイルを、vizcore で実行することが再び可能になりました。以下では、コンパイル例と実行用シェルスクリプトの例を示しました。詳しくはマニュアル本文をご覧ください。

コンパイル例（Fortran, C, C++）

コンパイルはログインサーバーの vizfront 上で実行する。以下がその例である。

逐次プログラムコンパイル例

```
【Fortran】
$ ifort -O3 -xAVX prog.f

【C】
$ icc -O3 -xAVX prog.c

【C++】
$ icpc -O3 -xAVX prog.cpp
```

OpenMP プログラムコンパイル例

```
【Fortran】
$ ifort -O3 -xAVX -qopenmp prog_omp.f

【C】
```

```
$ icc -O3 -xAVX -qopenmp prog_omp.c

  【C++】
$ icpc -O3 -xAVX -qopenmp prog_omp.cpp
```

MPI 並列プログラムコンパイル例

```
【Fortran】
$ mpif90 -O3 -xAVX prog_mpi.f

【C】
$ mpicc -O3 -xAVX prog_mpi.c

【C++】
$ mpicxx -O3 -xAVX prog_mpi.cpp
```

MPI + OpenMP 実行 (ハイブリッド並列) コンパイル例

```
【Fortran】
$ mpif90 -O3 -xAVX -qopenmp prog_hyb.f

【C】
$ mpicc -O3 -xAVX -qopenmp prog_hyb.c

【C++】
$ mpicxx -O3 -xAVX -qopenmp prog_hyb.cpp
```

ジョブ実行について（実行用シェルスクリプト例）(Fortran,C,C++ 等共通)

ジョブ実行（投入）の基本

実行用シェルスクリプトを作成して vizfront 上で qsub コマンドを実行する。次の例はシェルスクリプト名が run_mpi.sh の場合である。

```
【ジョブを投入】
$ qsub run_mpi.sh
60.vizcore

【ジョブが実行中であることを確認】
$ qstat
```

```
Job id            Name             User             Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
60.vizcore        mpi              scuser2           00:19:12 R uv-large

【ジョブを削除】
$ qdel 60
```

以下では，様々な並列実行の場合の実行用シェルスクリプト例を示す。

逐次プログラム実行用シェルスクリプト

リスト 1: run_serial.sh

```
#!/bin/bash
#PBS -N serial            ←ジョブ名
#PBS -q uv-test            ←キュー名
#PBS -o stdout.log        ←標準出力ファイル
#PBS -e stderr.log          ←標準エラー出力ファイル
#PBS -l select=1:ncpus=2    ←リソース確保（2 コア）

cd ${PBS_O_WORKDIR}         ←作業ディレクトリへ移動

dplace -c1 ./a.out        ←実行
```

スレッド並列（OpenMP）実行用シェルスクリプト

12 スレッドを使用する OpenMP 並列ジョブ投入スクリプト例は以下の通りです。（スクリプト名: run_omp.sh）

リスト 2: run_omp.sh

```
#!/bin/bash
#PBS -N openmp                ←ジョブ名
#PBS -q uv-large                ←キュー名
#PBS -o stdout.log            ←標準出力ファイル
#PBS -e stderr.log            ←標準エラー出力ファイル
#PBS -l select=1:ncpus=12     ←リソース確保（12 コア）

cd ${PBS_O_WORKDIR}           ←作業ディレクトリへ移動

export KMP_AFFINITY=disabled  ← Intel の Affinity を disabled にする
export OMP_NUM_THREADS=12     ←スレッド並列数の設定（12 スレッド）

dplace -x2 -c0-11 ./a.out     ← 12 スレッドで OpenMP 実行
```

MPI 並列実行用シェルスクリプト

12 プロセスを使用する MPI 並列ジョブ投入スクリプト例は以下の通り（スクリプト名: run_mpi.sh）。

リスト 3: run_mpi.sh

```
#!/bin/bash
#PBS -N mpi                                  ←ジョブ名
#PBS -q uv-large                             ←キュー名
#PBS -o stdout.log                           ←標準出力ファイル
#PBS -e stderr.log                           ←標準エラー出力ファイル
#PBS -l select=1:ncpus=12:mpiprocs=12        ←リソース確保（12 コア）

cd ${PBS_O_WORKDIR}                          ←作業ディレクトリへ移動

mpiexec_mpt -np 12 dplace -s1 -c0-11 ./a.out   ← 12 プロセスで MPI 並列実行
```

MPI + OpenMP ハイブリッド並列プログラム実行用シェルスクリプト例

MPI が 4 プロセスと各プロセスから 12 スレッドを使用する MPI+OpenMP ハイブリッド並列ジョブ投入スクリプト例は以下の通りです。（スクリプト名: run_hybrid.sh）

リスト 4: run_hybrid.sh

```
#!/bin/bash
#PBS -N hybrid                    ←ジョブ名
#PBS -q uv-large                  ←キュー名
#PBS -o stdout.log                ←標準出力ファイル
#PBS -e stderr.log                ←標準エラー出力ファイル
#PBS -l select=1:ncpus=48:mpiprocs=4       ←リソース確保（48 コア）

cd ${PBS_O_WORKDIR}               ←作業ディレクトリへ移動

export KMP_AFFINITY=disabled   ← Intel の Affinity を disabled にする
export OMP_NUM_THREADS=12       ←スレッド並列数の設定（12 スレッド）

mpiexec_mpt -np 4 omplace -nt ${OMP_NUM_THREADS} -c 0-47 ./a.out   ← hybrid 実行
```

目次

第１章

システム構成

1.1　ハードウェア構成

π-VizStudio システムの全体構成図を示します。

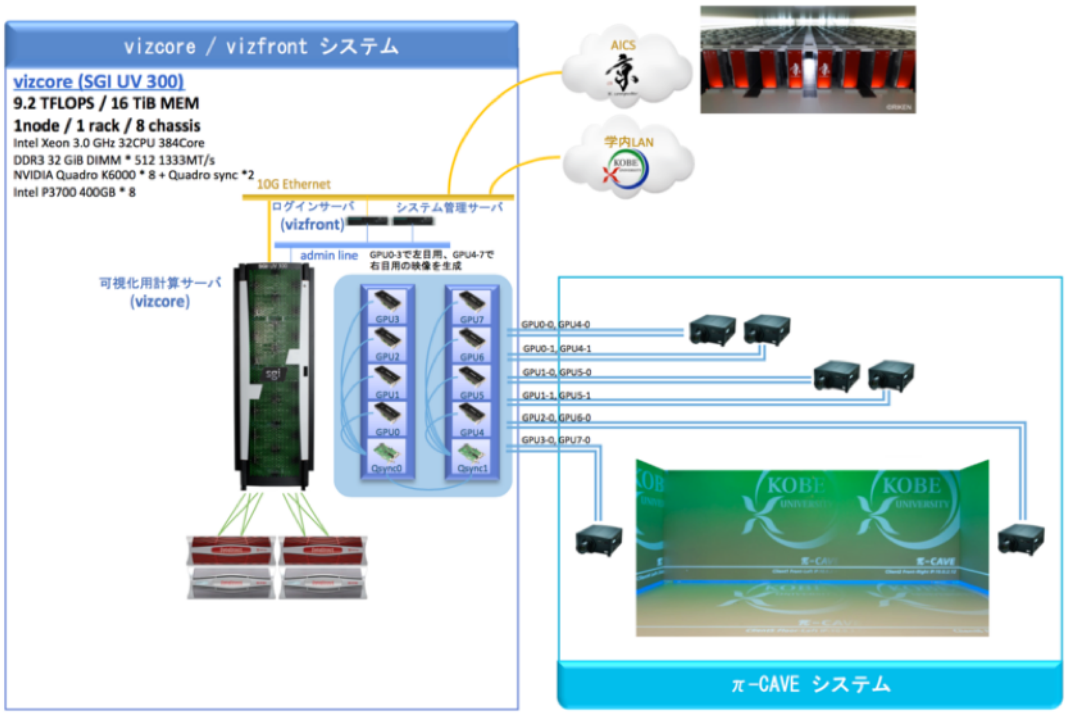

図 1.1: システム構成図

π-VizStudio システムは、ログインサーバの vizfront および中核となる可視化用計算サーバ vizcore、π-CAVE から構成されます。

主なハードウェアスペックを以下に示します。

	vizcore	vizfront
製品名	SGI UV 300	SGI C2112-GP2
CPU	Intel Xeon E7-8857 v2 (12 コア 3.0 GHz 30MB LLC) *32	Intel Xeon E5-2667 v3 (8 コア 3.2GHz 20MB LLC) *2
総 CPU **コア数**	384	16
メモリ	16 TiB (DDR3 32GiB * 512)	256 GiB(DDR4 32GiB * 8)
ディスク (home)	86TB (xfs)	86TB (nfs)
ディスク (data)	342TB (xfs)	342 TB (nfs)
ディスク (work)	3TB (xfs)	-
グラフィックス	NVIDIA Quadro K6000 * 8GPU	NVIDIA Quadro K5200 * 1GPU
ユーザネットワーク	10Gbps	10Gbps

1.2 ソフトウェア構成

主なソフトウェアスペックを以下に示します。

	vizcore	vizfront
OS	Red Hat Enterprise Linux 6.6	Red Hat Enterprise Linux 7.4
コンパイラ	Intel Parallel Studio XE Cluster Edition, PGI Fortran/C/C++ Server for Linux	Intel Parallel Studio XE Cluster Edition
MPI **通信ライブラリ**	SGI MPT (MPI Toolkit)	SGI MPT (MPI Toolkit), Intel MPI
数値計算ライブラリ	Intel MKL (Math Kernel Library)	Intel MKL (Math Kernel Library)
GPU **開発環境**	NVIDIA CUDA Toolkit 7.0	NVIDIA CUDA Toolkit 5.5
バッチ管理ソフトウェア	Altair PBS Professional 13.0	Altair PBS Professional 13.0
遠隔可視化ソフトウェア	NICE DCV	NICE DCV
汎用ポストプロセッサ	EnSight VR	-
汎用可視化ソフトウェア	AVS Express MPE	-
汎用可視化ソフトウェア	ParaView	ParaView
画像表示ソフトウェア	ImageMagick	ImageMagick
コーデックライブラリ	FFMpeg	FFMpeg
レイトレーサ	POV-Ray	POV-Ray

第 2 章

システム利用環境

2.1 ログイン環境

2.1.1 vizfront へのログイン／ログアウト方法

π-VizStudio システムのフロントエンドサーバ vizfront へは、ssh でログインします。公開鍵認証のみ有効化しており、パスワード認証は無効化しています。

フロントエンドサーバのアドレスは、

```
vizfront.eccse.kobe-u.ac.jp
```

です。

```
[wsuser@workstation ~]$ ssh userxxxx@vizfront.eccse.kobe-u.ac.jp
The authenticity of host 'vizfront.eccse.kobe-u.ac.jp (133.30.94.201)' can't be establishe
RSA key fingerprint is 24:fa:25:92:a0:57:18:89:f9:13:56:1c:c7:50:a7:4f.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'vizfront.eccse.kobe-u.ac.jp,133.30.94.201' (RSA) to the list o
Enter passphrase for key '/home/wsuser/.ssh/id_rsa':
Last login: Mon Aug 17 05:14:04 2015 from workstation
[scuser@vizfront ~]$
```

openssh の ssh クライアントを利用している場合、下記のようにワークステーションや PC 端末のホームディレクトリ配下に、

```
~/.ssh/config
```

設定ファイルを作成すると、

```
ssh vizfront
```

を実行することにより vizfront へログインできます。

```
[wsuser@workstation ~]$ cat .ssh/config
Host vizfront
  HostName vizfront.eccse.kobe-u.ac.jp
  Port 22
  User scuser
  IdentityFile ~/.ssh/id_rsa
```

```
[wsuser@workstation ~]$ ssh vizfront
Enter passphrase for key '/home/sgi-remote/.ssh/id_rsa':
Last login: Mon Aug 17 09:12:17 2015 from workstation
[scuser@vizfront ~]$
```

logout もしくは exit コマンドを実行するとログアウトすることができます。

```
[scuser@vizfront ~]$ logout
Connection to vizfront.eccse.kobe-u.ac.jp closed.
```

2.1.2 vizcore へのログイン／ログアウト環境

vizcore を会話型で利用したい場合は、バッチ環境のインタラクティブモードを利用します。

```
[scuser@vizfront ~]$ qsub -I
qsub: waiting for job 13.vizcore to start
qsub: job 13.vizcore ready
[scuser@vizcore ~]$
```

logout もしくは exit コマンドを実行するとログアウトすることができます。

```
[scuser@vizcore ~]$ exit
logout
qsub: job 13.vizcore completed
```

```
[scuser@vizfront ~]$
```

2.2 インタラクティブ環境

2.2.1 シェル環境

vizfront

```
[scuser@vizfront ~]$ echo $SHELL
/bin/bash
```

vizcore

```
[scuser@vizcore ~]$ echo $SHELL
/bin/bash
```

2.2.2 言語環境

vizfront

```
[scuser@vizfront ~]$ echo $LANG
ja_JP.UTF-8
```

vizcore

```
[scuser@vizcore ~]$ echo $LANG
en_US.UTF-8
```

2.3 ファイルシステムの構成

π-VizStudio のファイルシステムを以下に示します。

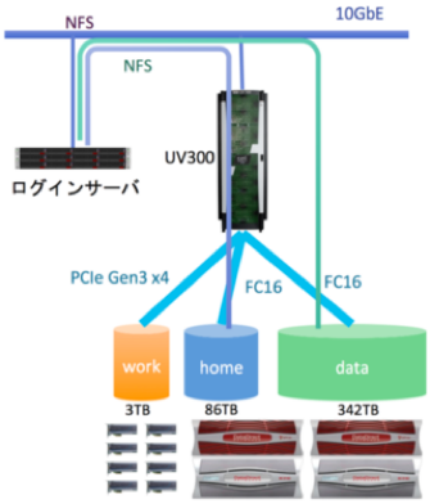

図 2.1: NFS 構成図

vizcore 及び vizfront で利用できる領域は下記の通りです。

マウントポイント	ファイルシステム	総容量	quota (soft)	quota (hard)	制限単位
/home/${GROUP}/${USER}	xfs	86TB	800GB	1TB	ユーザー
/data/${GROUP}/${USER}	xfs	342TB	16TB	20TB	グループ
/work	xfs	3TB	-	-	-

vizfront は vizcore の/home と/data を、NFS でマウントしています。/home のユーザ quota と/data のグループ quota でディスク使用状況の確認の仕方は下記の通りです。

$(GROUP) は申請プロジェクトの番号で、G の後に 5 桁の数字で表されています。$(USER) はあなたのアカウント名です。

2.3.1 /home の ユーザ quota

```
[viz@vizcore ~]$ /usr/sbin/xfs_quota -c 'quota -u viz' /home
Disk quotas for User viz (50002)
Filesystem Blocks Quota Limit Warn/Time Mounted on
/dev/xvm-1 78533664 838860800 1073741824 00 [--------] /home
```

2.3.2 /data の グループ quota

```
[viz@vizcore ~]$ /usr/sbin/xfs_quota -c 'quota -g eccse' /data
Disk quotas for Group eccse (10001)
Filesystem Blocks Quota Limit Warn/Time Mounted on
/dev/xvm-0 24071164 17179869184 21474836480 00 [--------] /data
```

2.3.3 各ストレージの用途

/home

設定ファイルやプログラム、小さめのデータを配置しておくことができます。

/data

設定ファイルやプログラム、大きめのデータを配置しておくことができます。/data/$(GROUP) に、グループ内でデータを共有するための shared フォルダがあります。この shared フォルダはグループメンバーであれば自由に読み書き可能なフォルダです。

```
/data/$(GROUP)/shared
```

/work

高速な SSD ストレージです。計算や可視化作業時の一時的なデータを配置することができます。容量の制限はかけていませんが、あくまで一時的なデータ置き場ですので、長期間、大容量のデータを放置しないようにお願いします。

■: /home と/data

各ユーザーのホームディレクトリには、初回ログイン時から data というディレクトリが配置されています。これは/data/$(GROUP)/$(USER) へのシンボリックリンクとなっています。

```
/home/GROUP/USER/data -> /data/GROUP/USER
```

2.4 データの転送

PC ／ワークステーションから vizfront へファイルを転送する場合、PC ／ワークステーションにおいて scp コマンドによるファイル転送を行います。

```
[wsuser@workstation ~]$ scp testfile scuser@vizfront.eccse.kobe-u.ac.jp:~/data
```

vizfront から PC ／ワークステーションへファイルを転送する場合も同様に、PC ／ワークステーションにおいて scp コマンドによるファイル転送を行います。

```
[wsuser@workstation ~]$ scp scuser@vizfront.eccse.kobe-u.ac.jp:~/data/testfile .
※最後のドットを忘れないようにしてください。
```

第 3 章

開発環境

π-VizStudio システムでは Intel コンパイラ、PGI コンパイラ、GNU コンパイラを利用できます。数値計算ライブラリは Intel Math Kernel Library (MKL)、MPI 通信ライブラリは SGI MPI Toolkit (MPT) が利用できます。

開発環境	vizfront	vizcore
intel **コンパイラ**	○	○
PGI **コンパイラ**	×	○
GNU **コンパイラ**	○	○
Intel MKL (**数値計算ライブラリ**)	○	○
SGI(HPE) MPT (MPI **通信ライブラリ**)	○	○
Intel MPI (MPI **通信ライブラリ**)		

3.1 Intel コンパイラの利用

3.1.1 コンパイル／リンクの方法

コンパイル／リンクの書式とコマンド一覧は、以下の通りです。

```
[scuser@vizfront ~]$ command [options] sourcefile [...]
```

コンパイルにはプログラム言語に応じて、command の部分を以下のコマンドに置き換えて利用します。

言語	コマンド名	実行例
Fortran77, 90, 95, 2003	ifort	$ ifort [options] program.f
ISO **標準** C	icc	$ icc [options] program.c
ISO **標準** C++	icpc	$ icpc [options] program.cpp

[options] には、最適化オプション、並列化オプション、ライブラリのリンクオプション、その他のコンパイラオプションが入ります。主なコンパイラオプションは、「3.7 Intel コンパイラのオプション」を参照ください。Intel コンパイラおよび数値計算ライブラリ Intel MKL（Math Kernel Library）環境は、

```
/opt/intel
```

配下にインストールされています。

3.1.2 サンプルプログラム

Intel コンパイラには、サンプルプログラムが付属しています。

項目	場所
Fortran / C / C++	/opt/intel/composerxe/Samples
Intel MKL	/opt/intel/composerxe/mkl/examples

3.1.3 環境設定

vizfront および vizcore では、デフォルトで Intel コンパイラと SGI MPT が利用できるように設定されています。

3.2 Fortran プログラムのコンパイル／リンク／実行方法

3.2.1 逐次 Fortran プログラム

```
【コンパイル】
$ ifort -O3 -xAVX prog.f

【実行】
$ dplace ./a.out
```

3.2.2 スレッド並列プログラム（OpenMP）

```
【コンパイル】
$ ifort -O3 -xAVX -qopenmp -qopt-report -qopt-report-phase=openmp \
  -qopt-report-file=stdout prog_omp.f

【環境設定】
$ export KMP_AFFINITY=disabled（Intel コンパイラのバインド機能の無効化）
$ export OMP_NUM_THREADS=4（スレッド数指定）

【実行】
$ dplace -x2 ./a.out
```

3.2.3 MPI 並列プログラム

```
【コンパイル】
$ mpif90 -O3 -xAVX prog_mpi.f
【実行】
$ mpirun -np 4 dplace -s1 ./a.out
```

3.2.4 MPI + OpenMP ハイブリッド並列プログラム

```
【コンパイル】
$ mpif90 -O3 -xAVX -qopenmp -qopt-report -qopt-report-phase=openmp -qopt-report-file=stdou
prog_hyb.f -lmpi

【環境設定】
$ export KMP_AFFINITY=disabled（Intel コンパイラのバインド機能の無効化）
$ export OMP_NUM_THREADS=4 （スレッド数指定）

【実行】
$ mpirun -np 4 omplace -nt ${OMP_NUM_THREADS} ./a.out
```

3.3 C コンパイル／リンク／実行方法

3.3.1 逐次 C プログラム

```
【コンパイル】
$ icc -O3 -xAVX prog.c

【実行】
$ dplace ./a.out
```

3.3.2 スレッド並列プログラム（OpenMP）

```
【コンパイル】
$ icc -O3 -xAVX -qopenmp -oqpt-report -qopt-report-phase=openmp -qopt-report-file=stdout \
prog_omp.c

【環境設定】
$ export KMP_AFFINITY=disabled（Intel コンパイラのバインド機能の無効化）
$ export OMP_NUM_THREADS=4     （スレッド数指定）
```

```
【実行】
$ dplace -x2 ./a.out
```

3.3.3 MPI 並列プログラム

```
【コンパイル】
$ mpicc -O3 -xAVX prog_mpi.c

【実行】
$ mpirun -np 4 dplace -s1 ./a.out
```

3.3.4 MPI + OpenMP ハイブリッド並列プログラム

```
【コンパイル】
$ mpicc -O3 -xAVX -qopenmp -qopt-report -qopt-report-phase=openmp \
-qopt-report-file=stdout prog_hyb.c

【環境設定】
$ export KMP_AFFINITY=disabled（Intel コンパイラのバインド機能の無効化）
$ export OMP_NUM_THREADS=4     （スレッド数指定）

【実行】
$ mpirun -np 4 omplace -nt ${OMP_NUM_THREADS} ./a.out
```

3.4 C++ コンパイル／リンク／実行方法

3.4.1 逐次 C++ プログラム

```
【コンパイル】
$ icpc -O3 -xAVX prog.cpp

【実行】
$ dplace ./a.out
```

3.4.2 スレッド並列プログラム (OpenMP)

```
【コンパイル】
```

```
$ icpc -O3 -xAVX -qopenmp -qopt-report -qopt-report-phase=openmp \
-qopt-report-file=stdout prog_omp.cpp

【環境設定】
$ export KMP_AFFINITY=disabled（Intel コンパイラのバインド機能の無効化）
$ export OMP_NUM_THREADS=4     （スレッド数指定）

【実行】
$ dplace -x2 ./a.out
```

3.4.3　MPI 並列プログラム

```
【コンパイル】
$ mpicxx -O3 -xAVX prog_mpi.cpp

【実行】
$ mpirun -np 4 dplace -s1 ./a.out
```

3.4.4　MPI + OpenMP ハイブリッド並列プログラム

```
【コンパイル】
$ mpicxx -O3 -xAVX -qopenmp -qopt-report -qopt-report-phase=openmp \
-qopt-report-file=stdout prog_hyb.cpp

【環境設定】
$ export KMP_AFFINITY=disabled（Intel コンパイラのバインド機能の無効化）
$ export OMP_NUM_THREADS=4    （スレッド数指定）

【実行】
$ mpirun -np 4 omplace -nt ${OMP_NUM_THREADS} ./a.out
```

3.5　dplace / omplace コマンドの利用

OS のプロセススケジュールにより、引き起こされる CPU 間でのプロセスの移動を避けるためにプログラム実行時に dplace や omplace コマンドを使用します。omplace コマンドは「3.5.4 MPI + OpenMP ハイブリッド並列プログラム」の際、スレッド並列を最適に CPU 配置する場合に指定します。特に何も指定しない場合、OS はプロセスが実行される CPU を含むノードのローカルメモリにデータを配置します。もし、プロセスが CPU 間を移動してしまった場合、いったん配置されたデータは移動しないためデータ参照はリモートメモリへの参照となってしまいます。リモートメモリの参照はローカルメモリのデータ参照と比較して時間がかかるため、プログラムの性能を低下させる可能性があります。このような現象を防ぐためにプログラム実行時には dplace や omplace コマンドを指定してください。

dplace や omplace コマンドの利用時に、以下で説明する-c オプション用いて CPU 番号を明示的に指定すると、プロセスは指定された CPU へバインドされます。該当 CPU で既に他のプロセスが dplace や ompleace コマンドによりバインドされていた場合には、1 つの CPU を 2 つ以上のプロセスが共有することになってしまうため、プロセスが割り当てられていない、空いている CPU 番号を指定してバインドする必要があります。バインドされた CPU の状態を確認するためのコマンドとして、ndstat を vizcore 上に用意しています。ndstat を利用するためには、「2.1.2 vizcore へのログイン／ログアウト環境」で説明したインタラクティブモードでの vizcore へのログインが必要です。以下に CPU 番号を明示的に指定する場合の例を示しますが、プロセス配置を細かく指定する必要が特に無い場合には、-c オプション無しでの dplace や omplace の利用が推奨されます。

3.5.1 逐次プログラム

```
【CPU 7 番のコアにプロセスを配置する場合】
$ dplace -c7 ./a.out
```

3.5.2 スレッド並列プログラム（OpenMP）

```
【CPU 0〜7 番のコアにスレッドを配置し、管理スレッドは配置するスレッドから除外】
$ dplace -x2 -c0-7 ./a.out
```

3.5.3 MPI 並列プログラム

```
【CPU 0〜7 番のコアにプロセスを配置し、管理プロセスは配置するプロセスから除外】
$ mpirun -np 8 dplace -s1 -c0-7 ./a.out
```

3.5.4 MPI + OpenMP ハイブリッド並列プログラム

```
【MPI プロセスが 4、OpenMP のスレッド数が 4 と設定されたハイブリッドジョブ】
$ export OMP_NUM_THREADS=4
$ mpirun -np 4 omplace -nt ${OMP_NUM_THREADS} -c0-15 ./a.out
```

ここでは、0 から 15 番のコアを指定しており、MPI プロセスが 4、OpenMP のスレッド数が 4 と設定されています。この場合、MPI プロセスは 0, 4, 8, 12 番のコアに配置され、OpenMP スレッドはそれぞれの MPI プロセスと隣り合うコア（MPI ランク 0 から生成される OpenMP スレッド

は 0〜3 番のコア）に配置されます。

3.5.5 dplace コマンドで配置するコアの指定方法

- -c <cpulist>
  - CPU コア番号の指定方法

cpulist	配置 (コア番号)
0-3	0,1,2,3
0-7:2	0,2,4,6
0-1,4-5	0,1,4,5
0-3:2,8-9	0,2,8,9

- -x2
  - OpenMP の管理スレッドを配置するスレッドから除外する

- -s1
  - MPI の管理プロセスを配置するプロセスから除外する

3.5.6 omplace コマンドで配置するコアの指定方法

- -nt ${OMP_NUM_THREADS}
  - スレッド数を指定する

- -c <cpulist>
  - CPU コア番号の指定方法

cpulist	配置 (コア番号)
0-N	0,1,2,3 … N (最後のコア番号)
1:st=2	1,3,5,7, … (全ての奇数番号のコア)
0,1,1-4	0,1,1,2,3,4 (1 番のコアに 2 つのプロセスを配置)
0-6:st=2, 1-7:st=2	0,2,4,6,1,3,5,7
16-31:bs=2+st=4	16,17,20,21,24,25,28,29

3.6 数値計算ライブラリの利用

3.6.1 Intel Math Kernel Library（MKL） 概要

数値演算ライブラリとして、Intel Math Kernel Library （MKL）が用意されています。MKL は、次の特徴を有しています。

- 科学技術計算向け
- インテルプロセッサにチューニング
- マルチスレッド対応スレッド並列化スレッドセーフ
- 自動ランタイム・プロセッサ検出機能

- C および Fortran のインターフェース

Intel MKL は数値計算ライブラリとして、下記の機能を含んでいます。

- BLAS
- BLACS
- LAPACK
- ScaLAPACK
- PBLAS
- Sparse Solver
- Vector Math Library (VML)
- Vector Statistical Library (VSL)
- Conventional DFTs and Cluster DFTs
- FFTW interface

3.6.2 シリアル版のリンク

シリアル版 MKL を利用する場合、下記のようにコンパイルします。

```
【MKL をリンク】
$ mpif90 -lmkl_intel_lp64 -lmkl_sequential -lmkl_core test.f

【Intel コンパイラのオプションによる指定】
$ mpif90 -mkl=sequential test.f
```

BLACS および ScaLAPACK を利用する場合、下記のようにコンパイルします。

```
【MKL をリンク】
$ mpif90 -lmkl_scalapack_lp64 -lmkl_blacs_sgimpt_lp64 -lmkl_intel_lp64 \
-lmkl_sequential -lmkl_core test.f

【Intel コンパイラのオプションによる指定】
$ mpif90 -lmkl_scalapack_lp64 -lmkl_blacs_sgimpt_lp64 -mkl=sequential test.f
```

3.6.3 スレッド並列版のリンク

スレッド並列版 MKL を利用する場合、下記のようにコンパイルします。

```
【MKL をリンク】
```

```
$ mpif90 -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5 test.f

【Intel コンパイラのオプションによる指定】
$ mpif90 -mkl=parallel test.f
```

BLACS および ScaLAPACK を利用する場合、下記のようにコンパイルします。

```
【MKL をリンク】
$ mpif90 -lmkl_scalapack_lp64 -lmkl_blacs_sgimpt_lp64 -lmkl_intel_lp64 \
-lmkl_intel_thread -lmkl_core -liomp5 test.f -lmpi

【Intel コンパイラのオプションによる指定】
$ mpif90 -lmkl_scalapack_lp64 -lmkl_blacs_sgimpt_lp64 -mkl=parallel \
test.f -lmpi
```

実行方法は下記の通りです。

```
【環境設定】
$ export KMP_AFFINITY=disabled  （Intel コンパイラのバインド機能の無効化）
$ export OMP_NUM_THREADS=4      （スレッド数指定）

【実行】
$ dplace -x2 ./a.out
```

MKL の関数を OpenMP で指定するスレッド数と異なるスレッド数で実行したい場合、下記のように MKL_NUM_THREADS 環境変数を別途設定します。

```
【環境設定】
$ export KMP_AFFINITY=disabled  （Intel コンパイラのバインド機能の無効化）
$ export OMP_NUM_THREADS=8      （OpenMP スレッド数指定）
$ export MKL_NUM_THREADS=4      （MKL スレッド数指定）

【実行】
$ dplace -x2 ./a.out
```

スレッド並列版の MKL を使用する場合、下記の設定が必要です。

- 逐次 (シリアル) 実行
  - 環境変数 OMP_NUM_THREADS を 1 に設定します。または、シリアル版 MKL をリンクします。
- スレッド並列で実行
  - 環境変数 OMP_NUM_THREADS を並列実行数に設定します。OpenMP のプログ

ラム中で MKL を使う場合、OMP_NUM_THREADS で設定されたスレッド数で実行されます。また、OpenMP のスレッド数とは違うスレッド数で実行したい場合は OMP_NUM_THREADS 以外に MKL_NUM_THREADS を設定します。OpenMP で並列化されたループ内で MKL のスレッド並列化された関数を用いる場合、デフォルトでは OpenMP のネストが無効になっているため、MKL のスレッド並列は無効です。環境変数 OMP_NESTED を ”yes ”とすることにより、MKL のスレッド並列を有効にすることが可能です。

- MPI 並列で実行
  - MPI のみで並列実行する場合、MKL がスレッド並列で動作しないように環境変数 OMP_NUM_THREADS を 1 に設定します。または、シリアル版 MKL をリンクします。
- ハイブリッド並列で実行
  - MPI とスレッド並列のハイブリッドでの実行をする場合、MKL のスレッド数を OMP_NUM_THREADS または MKL_NUM_THREADS で設定します。

3.7 Intel コンパイラのオプション

Intel コンパイラのデフォルトで設定されている主なオプションは下記の通りです。

- 最適化レベル
  - -O2
  - パフォーマンス向上のための最適化を行います
- 特定のプロセッサ向けの最適化
  - -msse2
  - インテルプロセッサ向けに SSE2 および SSE 命令を生成し、SSE2 対応のインテル Xeon プロセッサ向けの最適化をします。

推奨するオプションは下記の通りです。

- 最適化レベル
  - -O3
  - O2 に加えプリフェッチ、スカラー置換、ループ変換、およびメモリアクセス変換などのより強力な最適を有効にします。
- 特定のプロセッサ向けの最適化
  - -xAVX
  - インテル AVX ベクトル化命令および、SSE4.2、SSSE3, SSE3, SSE2, SSE 命令を生成し、インテル AVX 命令をサポートするプロセッサ向けに最適化をします。

3.7.1 最適化レベルオプション

- -O0

  - 全ての最適化を無効とします。主にデバッグ時に利用。
- -O1
  - グローバルな最適化を有効化・組込み関数の認識と組込み関数のインライン展開の無効この最適化レベルでは、分岐が多く、実行時間の多くがループではないコードの性能向上が見込めます。
- -O2
  - デフォルトの最適化レベル。最適化レベルを指定しない場合、この最適化レベルが適用されます。この最適化レベルでは次の最適化を行います。
- -O3
  - -O2 オプションに加えて、プリフェッチ、スカラー置換、キャッシュ・ブロッキング、ループ変換、メモリアクセス変換などの最適化を行います。浮動小数点演算の多いループや大きなデータセットを処理するコードで性能向上が見込めます。-axAVX および-xAVX オプションとの組み合わせでより詳細なデータ依存性解析をします。
- -fast
  - -ipo、-O3、-no-prec-div、-static、-fp-model fast=2、-xHost を有効にするマクロオプションです。-fast オプションには-static オプションが含まれるため、ダイナミック・ライブラリしか提供されていないライブラリ (SGI MPT 等) を利用する場合、-Bdynamic オプションでそのライブラリを指定する必要があります。

3.7.2 最適化に関するオプション

- -vec
  - ベクトル化を有効/無効にします。デフォルトは有効です。
- -qopt-report=n -qopt-report-phase=vec -qopt-report-file=stdout
  - ベクタライザーからのメッセージをコントロールします。デフォルトではベクタライザーからのメッセージは出力されません。ベクタライザーからのメッセージを出力するためには、このオプションを有効にしてください。
- -no-prec-div
  - IEEE 準拠の除算よりも多少精度が低くなる場合がありますが、最適化を試みます。
- -no-prec-sqrt
  - 平方根計算については多少精度が低くなる場合がありますが、高速な計算を行います。

最適化レポートに関するオプションは下記の通りです。

- -qopt-report [n]
  - 最適化レポートを標準エラー出力に表示します。n=0: 最適化レポートを表示しません。n=1: 簡易なレポートを表示します。n=2: 標準的なレポートを表示します。（デフォルト）n=3: 詳細なレポートを表示します。
- -qopt-report-file=name
  - 最適化レポートを name というファイルに出力します。
- -qopt-report-routine=name

  - name で指定されたサブルーチンのレポートのみを出力
- -qopt-report-phase=name
  - name で指定された最適化フェーズのレポートを出力
- -qopt-report-help
  - 最適化レポート生成可能な最適化フェーズを表示

最適化のフェーズは下記の通りです。

- cg
  - コード生成フェーズ
- ipo
  - プロシージャー間の最適化 (ipo) フェーズ
- loop
  - ループの入れ子構造の最適化フェーズ
- openmp
  - OpenMP フェーズ
- par
  - 自動並列化フェーズ
- pgo
  - プロファイルに基づく最適化 (PGO) フェーズ
- tcollect
  - トレーズ収集フェーズ
- vec
  - ベクトル化フェーズ
- all
  - 全ての最適化フェーズ。Name を指定しない場合のデフォルト設定。

3.7.3 特定のプロセッサ向けの最適化オプション

- -x プロセッサ
  - プロセッサで指定した特定のプロセッサ向けのバイナリを生成します。
- -ax プロセッサ
  - プロセッサで指定した特定のプロセッサ向けのバイナリと一般的な IA32 アーキテクチャ向けのバイナリを一つのバイナリで生成します。

指定可能なプロセッサ名は下記の通りです。

- HOST
  - コンパイルをしたプロセッサで利用可能な、最も高いレベルの命令を生成し、そのプロセッサ向けの最適化を行います。
- AVX
  - IvyBridge 世代 (Intel Xeon E7-8800v2 等) 以降の最適化を行い、AVX 命令を生成しま

す。さらに、SSE4.2、SSE4、SSSE3、SSE3、SSE2、SSE 命令を生成し、インテル AVX 命令をサポートするプロセッサ向けの最適化を行います。

- SSE4.2
  - Nehalem-EP(Intel Xeon 5500 番台および 5600 番台）向けの最適化を行い、SSE4.2 命令を生成します。さらに、SSE4 のベクトル化コンパイル命令、メディア・アクセラレター、SSSE3, SSE3, SSE2, SSE 命令を生成し、インテル Core プロセッサ向け最適化を行います。
- SSE4.1
  - SSE4 のベクトル化コンパイル命令、メディア・アクセラレター、SSSE3, SSE3, SSE2, SSE 命令を生成し、45nm プロセスルール世代のインテル Core プロセッサ (Intel Xeon 5200 番台、5400 番台）向け最適化を行います。
- SSE3
  - SSSE3, SSE3, SSE2, SSE 命令を生成し、インテル Core2 Duo プロセッサ (Intel Xeon 5100 番台、5300 番台）向け最適化を行います。

3.7.4 プロシージャ間解析の最適化

- -ip
  - 1 つのソースファイルにあるプロシージャ間の解析、最適化を行います。
- -ipo
  - 複数のソースファイルにあるプロシージャ間の解析、最適化を行います。リンク時にもオプションとして指定してください。

3.7.5 浮動小数点演算に関するオプション

- -ftz
  - アンダーフローが発生したときに値をゼロに置き換えます。デフォルトでは、このオプションが有効になっています。このオプションが数値動作で好ましくない結果を出力した場合、-no-ftz オプションでアンダーフローが発生したときに値をゼロにフラッシュしなくなります。
- -fltconsistency
  - 浮動小数点の一貫性を向上させ、IEEE754 規格に則った浮動小数点演算コードを生成します。

3.7.6 スレッド並列化オプション 【OpenMP】

- -qopenmp
  - プログラム中の OpenMP 指示行を有効にします。
- -qopt-report=n -qopt-report-phase=omp -qopt-report-file=stdout
  - OpenMP 診断メッセージを出力します。n=0: メッセージを表示しません。n=1:

並列化されたループ、領域を表示します。n=2: 上記に加え、MASTER, SINGLE, CRITICAL 指示句などを表示。これらのオプションを設定しない場合、-qopenmp オプションをつけてコンパイルしても、OpenMP による並列化についてのメッセージは出力されません。

3.7.7 スレッド並列化オプション 【自動並列化】

- -parallel
  - 自動並列化を行います。
- -qopt-report=n -qopt-report-phase=vec -qopt-report-file=stdout
  - 自動並列化メッセージを出力します。n=0: メッセージを表示しません。n=1: 並列化されたループを表示します。n=2: 並列化されたループとされていないループを表示 n=3: 2 の出力に加えて、private、shared、reduction としての分類されているメモリ位置を出力します。n=4: 3 と同じ。n=5: 並列化不可要因（依存関係）も表示します。これらのオプションを設定しない場合、-parallel オプションをつけてコンパイルしても、自動並列化についてのメッセージは出力されません。
- -par-threshold [n]
  - 自動並列化のしきい値（確信度）を設定します。（n は 0〜100）n=0: ループの計算量に関わらず、常に自動並列化をします。n=100: 性能向上が見込める場合のみ、自動並列化をします。

3.7.8 MPI 並列化オプション

- -lmpi
  - mpi ライブラリにリンクし、コンパイルします。全てのオプションの最後に記述して下さい。SGI MPT は静的ライブラリがありません。-static オプション（static オプションを含む -fast オプション）をつけてコンパイルした場合、-Bdynamic -lmpi として SGI MPT の動的ライブラリをリンクしてください。
  - なお、mpif90 等は実際には mpifort にオプションをつけたものが実体です。mpif90 -show などで確認できます。

3.7.9 その他のオプション

- -V
  - コンパイラのバージョンを表示します。
- -help
  - オプション一覧を表示します。

3.8 PGI コンパイラの利用

PGI コンパイラは vizcore のみで利用可能です。PBS のインタラクティブモードで vizcore にログインして使用します。コンパイルには言語に応じて、以下のコマンドを利用します

言語	コマンド名
Fortran77, 90, 95, 2003	pgfortran
ISO 標準 C	pgcc
ISO 標準 C++	pgc++

PGI コンパイラには、サンプルプログラムが付属しています。

項目	場所
Fortran / C / C++	/opt/pgi/linux86-64/2015/examples

3.8.1 vizcore へのログイン

qsub -I でログインします。

```
[scuser@vizfront ~]$ qsub -I
qsub: waiting for job 17.vizcore to start
qsub: job 17.vizcore ready
```

3.8.2 モジュールのロード

PGI コンパイラを利用するため、モジュールをロードします。

```
[scuser@vizcore ~]$ module load pgi64/15.7
```

3.8.3 サンプルモジュールのコンパイル／実行

AutoPar モジュールをコンパイル、実行します。

```
[scuser@vizcore ~]$ cp -r /opt/pgi/linux86-64/2015/examples/AutoPar .
[scuser@vizcore ~]$ cd AutoPar
[scuser@vizcore AutoPar]$ make NTHREADS=4 all
```

標準出力で、"Test PASSED" の表示がされていれば、PGI コンパイラのビルドおよびビルドしたモジュールの実行が正常に動作したことが分かります。

3.8.4 vizcore からのログアウト

exit コマンドでジョブを終了します。

```
[scuser@vizcore AutoPar]$ exit
logout

qsub: job 17.vizcore completed
```

第 4 章

ジョブ実行

4.1 ジョブシステム概要

π-VizStudio システムでは、効率的なジョブ運用を実現するため、PBS Professional （以下、PBS Pro と記述） を導入しています。ユーザは、ジョブ投入時に vizcore の必要なリソース情報を指定し、PBS Pro に対してジョブ実行を指示します。バッチジョブには、CPU やメモリなどの計算に必要なリソースが排他的に割り当てられます。ユーザがジョブ操作に用いるコマンドは以下の通りです。

機能	コマンド名
ジョブ投入	qsub
会話型ジョブ投入	qsub -I
ジョブ参照	qstat
ジョブ削除	qdel

4.2 キュー構成

vizcore のリソースは、次のようにキューに割り当てられています

キュー名		uv-pv	uv-test	uv-large
キューへの割当	コア数	48	12	324
-	メモリ	1.8TB	450GB	12.15TB
1 ユーザ	実行数	4	2	2
-	投入数	4	2	4
並列数	デフォルト値	12	1	24
-	上限値	24	4	96
メモリ	デフォルト値	450GB	37.5GB	900GB
-	上限値	1.8TB	150GB	3.6TB
経過時間	標準値	2h	1h	24h
-	制限値	6h	1h	24h

バッチ処理によるジョブ実行を行う場合には、uv-test キューまたは uv-large キューを用います。uv-pv は vizcore の計算リソースを使用して、ParaView による可視化を行うためのキューです。優先的に GPU が搭載されたソケットでジョブ（ParaView）が実行されます。

4.3 ジョブ投入方法

4.3.1 逐次プログラム（シリアル）実行用シェルスクリプト例

逐次プログラムのジョブ投入スクリプト例は以下の通りです。(スクリプト名: run_serial.sh)

リスト 4.1: run_serial.sh

```
#!/bin/bash
#PBS -N serial             ←ジョブ名
#PBS -q uv-test             ←キュー名
#PBS -o stdout.log         ←標準出力ファイル
#PBS -e stderr.log           ←標準エラー出力ファイル
#PBS -l select=1:ncpus=2    ←リソース確保（2 コア）

cd ${PBS_O_WORKDIR}          ←作業ディレクトリへ移動

dplace -c1 ./a.out         ←実行
```

ジョブスクリプトにジョブ投入オプションを指定せずにコマンド実行時に指定することも可能です。

- qsub [option] <JOB_SCRIPT>
  - -N
    - ジョブ名の指定
  - -q
    - ジョブを投入するキューの指定
  - -o
    - 標準出力ファイルの PATH の指定
  - -e
    - 標準エラー出力ファイルの PATH の指定
  - -l
    - ジョブ実行に必要なリソースの要求
    - 主なリソース
      - select=(使用ホスト数) → π-VizStudio ではホスト数は常に 1 を指定する
      - ncpus=(CPU コア数の指定)
      - mem=(最大物理メモリ容量)
      - walltime=(ジョブを実行できる実際の経過時間)

ジョブ投入スクリプトを使用して、ジョブを投入します。

```
$ qsub run_serial.sh
41.vizcore
```

4.4 スレッド並列（OpenMP）実行用シェルスクリプト

12 スレッドを使用する OpenMP 並列ジョブ投入スクリプト例は以下の通りです。（スクリプト名: run_omp.sh）

リスト 4.2: run_omp.sh

```
#!/bin/bash
#PBS -N openmp                ←ジョブ名
#PBS -q uv-large                ←キュー名
#PBS -o stdout.log            ←標準出力ファイル
#PBS -e stderr.log            ←標準エラー出力ファイル
#PBS -l select=1:ncpus=12     ←リソース確保（12 コア）

cd ${PBS_O_WORKDIR}           ←作業ディレクトリへ移動

export KMP_AFFINITY=disabled  ← Intel の Affinity を disabled にする
export OMP_NUM_THREADS=12     ←スレッド並列数の設定（12 スレッド）

dplace -x2 -c0-11 ./a.out     ← 12 スレッドで OpenMP 実行
```

4.5 MPI 並列実行用シェルスクリプト

12 プロセスを使用する MPI 並列ジョブ投入スクリプト例は以下の通り（スクリプト名: run_mpi.sh）。

リスト 4.3: run_mpi.sh

```
#!/bin/bash
#PBS -N mpi                                ←ジョブ名
#PBS -q uv-large                           ←キュー名
#PBS -o stdout.log                         ←標準出力ファイル
#PBS -e stderr.log                         ←標準エラー出力ファイル
#PBS -l select=1:ncpus=12:mpiprocs=12      ←リソース確保（12 コア）

cd ${PBS_O_WORKDIR}                        ←作業ディレクトリへ移動

mpiexec_mpt -np 12 dplace -s1 -c0-11 ./a.out   ← 12 プロセスで MPI 並列実行
```

4.6 MPI + OpenMP ハイブリッド並列プログラム実行用シェルスクリプト例

MPI が 4 プロセスと各プロセスから 12 スレッドを使用する MPI+OpenMP ハイブリッド並列ジョブ投入スクリプト例は以下の通りです。（スクリプト名: run_hybrid.sh）

リスト 4.4: run_hybrid.sh

```
#!/bin/bash
#PBS -N hybrid                  ←ジョブ名
#PBS -q uv-large                ←キュー名
#PBS -o stdout.log              ←標準出力ファイル
#PBS -e stderr.log              ←標準エラー出力ファイル
#PBS -l select=1:ncpus=48:mpiprocs=4      ←リソース確保（48 コア）

cd ${PBS_O_WORKDIR}             ←作業ディレクトリへ移動

export KMP_AFFINITY=disabled    ← Intel の Affinity を disabled にする
export OMP_NUM_THREADS=12        ←スレッド並列数の設定（12 スレッド）

mpiexec_mpt -np 4 omplace -nt ${OMP_NUM_THREADS} -c 0-47 ./a.out   ← hybrid 実行
```

4.6.1 大規模メモリプログラム （シリアル）実行用シェルスクリプト例

大きなメモリを確保するためのジョブ投入スクリプト例は以下の通り（スクリプト名: run_mem.sh）。

リスト 4.5: run_mem.sh

```
#!/bin/bash
#PBS -N largemem                    ←ジョブ名
#PBS -q uv-large                    ←キュー名
#PBS -o stdout.log                  ←標準出力ファイル
#PBS -e stderr.log                  ←標準エラー出力ファイル
#PBS -l select=1:ncpus=8:mem=1800gb  ←リソース確保（8 コア、1800GB）

cd ${PBS_O_WORKDIR}                 ←作業ディレクトリへ移動

dplace -c7 ./a.out                  ←実行
```

4.7 ジョブの状態確認方法

ジョブの状態は qstat コマンドで確認できます。

```
$ qstat
Job id            Name             User             Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
50.vizcore        mpi-gentle       scuser1          28:28:46 R uv-large
51.vizcore        mpi-gentle       scuser2          28:24:32 R uv-large
```

4.8 ジョブのキャンセル方法

ジョブをキャンセルするには qdel コマンドを実行します。

```
$ qdel [JOBID [JOBID…]]
```

ジョブの投入、実行状態の確認、ジョブのキャンセル例を下記に示します。

```
【ジョブを投入】
$ qsub runmpi.sh
60.vizcore

【ジョブが実行中であることを確認】
$ qstat
Job id            Name             User             Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
60.vizcore        mpi-gentle       scuser2          00:19:12 R uv-large

【ジョブを削除】
$ qdel 60

【実行中のジョブがなくなっていることを確認】
$ qstat
```

π-VizStudio User Manual

2016 年 2 月 29 日　初版第 1 刷　発行
2017 年 7 月 4 日　第 2 版第 1 刷　発行
2017 年 11 月 1 日　第 3 版第 1 刷　発行
2018 年 4 月 2 日　第 4 版第 1 刷　発行
2018 年 7 月 26 日　第 5 版第 1 刷　発行
著　者　神戸大学 計算科学教育センター
印刷所　神戸大学 計算科学教育センター