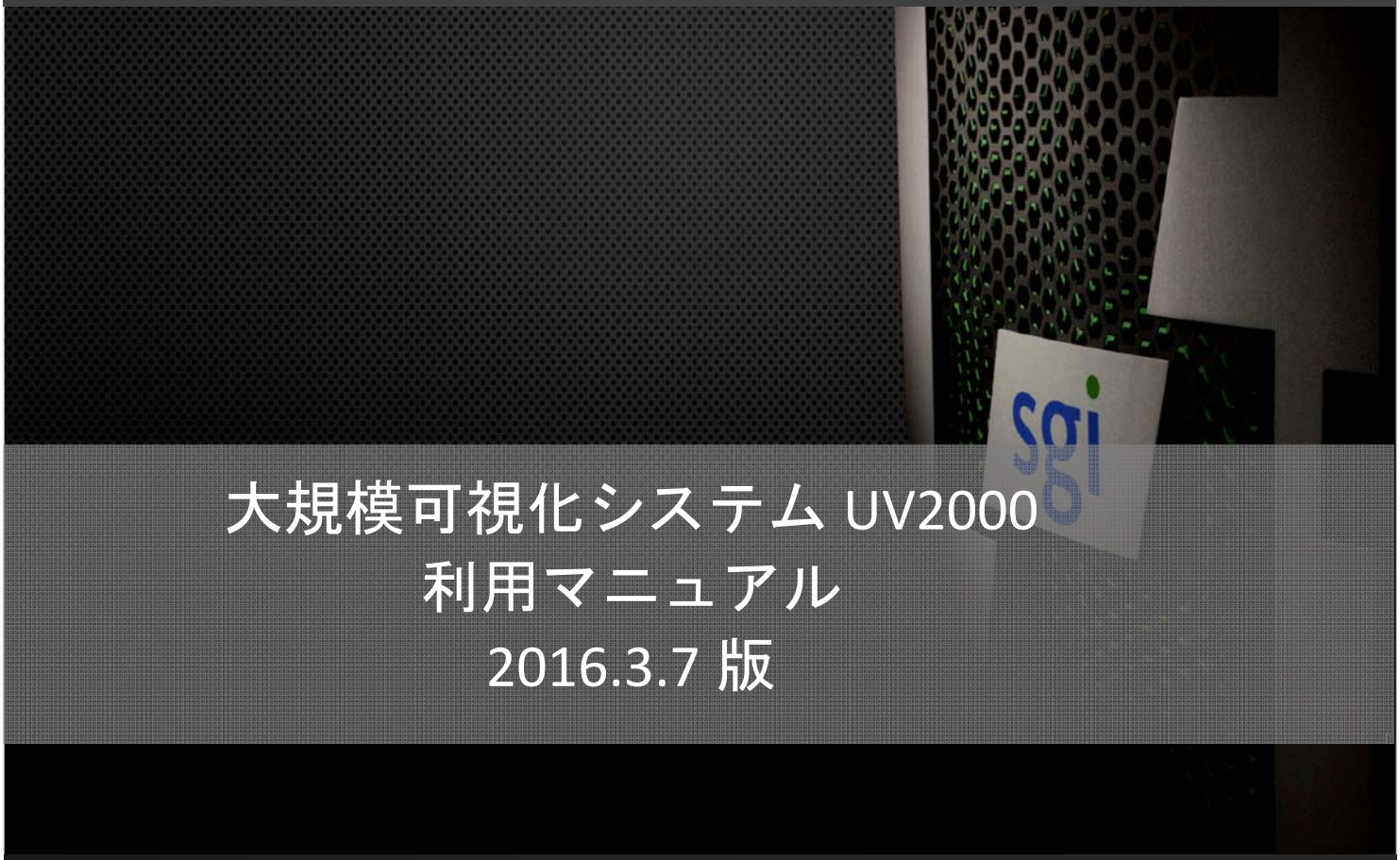


大規模可視化システム UV2000
利用マニュアル
2016.3.7 版

名古屋大学　情報基盤センター

はじめに

本書『大規模可視化システム UV2000 利用者マニュアル』は、名古屋大学情報基盤センターの利用マニュアルです。

ご不明な点やご質問がございましたら、次の連絡先にお問い合わせください。

【問い合わせ先】
名古屋大学　情報連携統括本部　情報推進部
情報基盤共同利用担当（情報基盤センター）
メール：kyodo@itc.nagoya-u.ac.jp

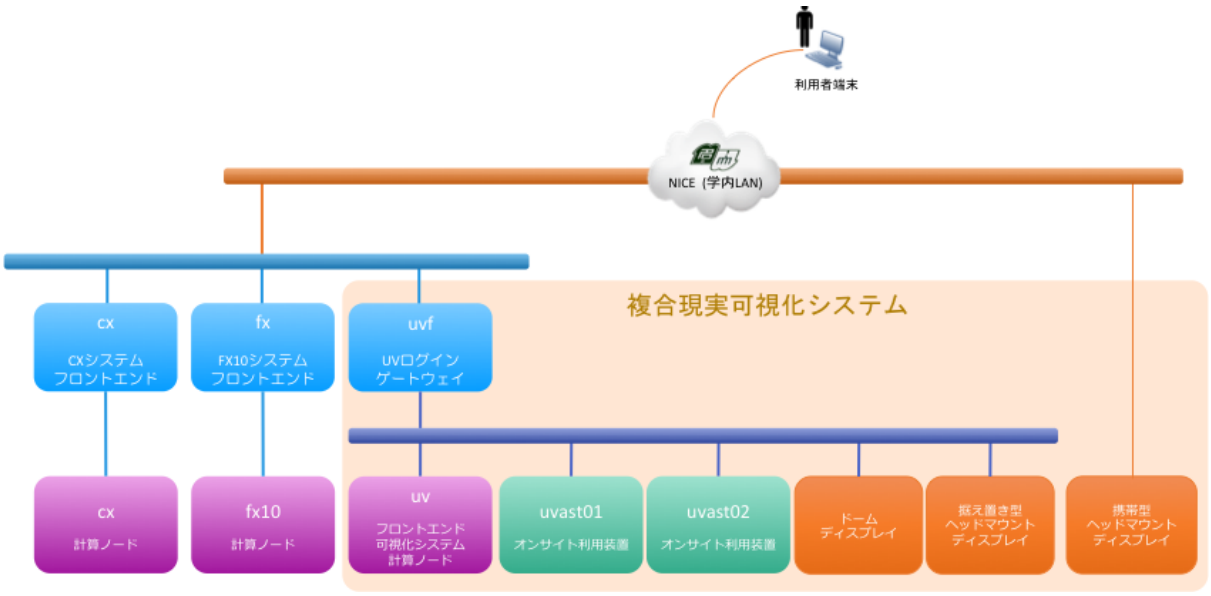
目次

1. 大規模可視化システム UV2000 構成概要

1.1. システム構成

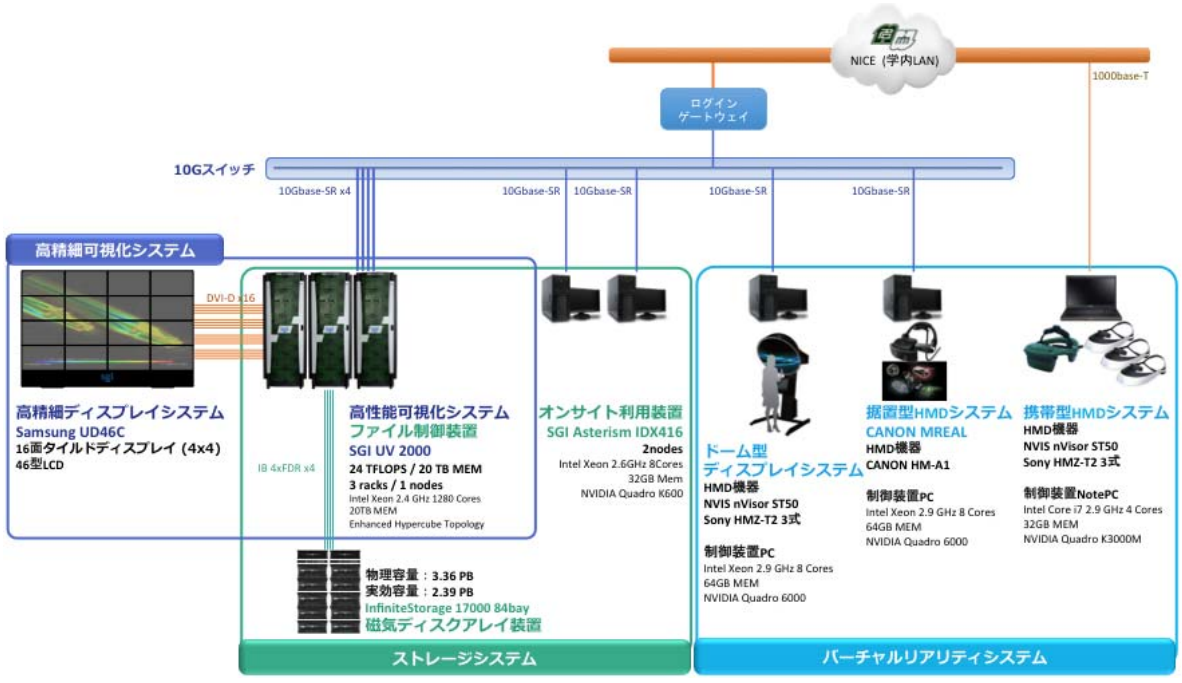
本システムは、スーパーコンピュータから得られる大規模シミュレーションデータの後処理・可視化・コンテンツ生成を行うためのシステムで、大規模メモリを使った短時間の計算にも利用できます。

FX システム、CX システムとのシステムの関係を下記に示します。



本システムは、高精細可視化システム、ストレージシステムおよびバーチャルリアリティシステムから構成されます。また、スパコンシステムで導入した円偏光立体視システム（150 インチスクリーン,プロジェクター）を使ったと組み合わせて利用することもできます。

各システム構成の概要を下記に示します。



1.2. 高精細可視化システム

1.2.1. ハードウェア構成

① ハードウェア仕様

TSS 環境、バッチ環境および可視化環境を提供する SGI UV 2000 システムの大きな特徴は、

② 1 システムで 1,280 コア、20TB メモリを有する共有メモリ機
③ 高精細グラフィックス NVIDIA K5000 を 8 基搭載

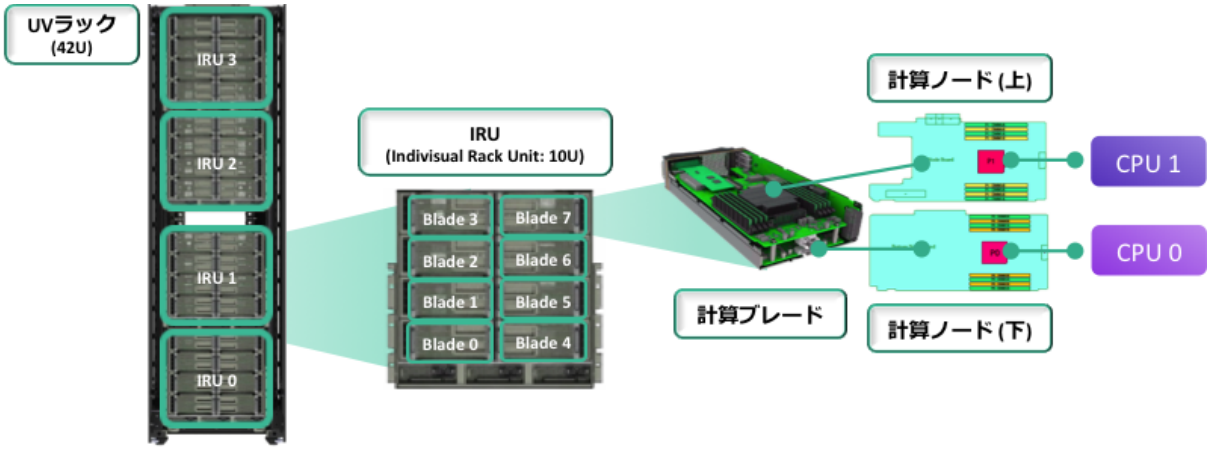
であることです。

主なハードウェア仕様は、下記の通りです。

システム名		SGI UV 2000	備考
ノード数		1	
CPU	TFLOPS	24 TFLOPS	
	CPU タイプ	Intel Xeon E5 4650 2.4GHz 8Cores	
	コア数	1,280	8 コア ＊ 160 ソケット
	L3 キャッシュ	20MB	8 コアで共有
MEM	MEM タイプ	DDR3-1600 16GiB ＊ 1,280	8 Banks ＊ 160 ソケット
	主記憶容量	20TiB	
	転送速度	51.2 GB/s	1 ソケットあたり
インターコネクト	ネットワーク	NUMALink 6 ネットワーク	1 リンクあたり双方向 6.7 GB/s
	トポロジー	Enhanced Hypercube topology	
グラフィックス	名称	NVIDIA Quadro K5000 ＊ 8 基	8K スクリーン用途に 4GPU 遠隔可視化用途に 4GPU
	CUDA コア	1,536	1GPU あたり
	VRAM 容量	GDDR 4GB	1GPU あたり
ストレージ	装置名	SGI InfiniteStorage 17000	
	RAID レベル	RAID6	
	ディスク総容量	2.4PB	
	インターフェース	InfiniBand 4xFDR	IB による DAS 接続
ネットワーク		10GbE NIC dual port ＊ 2	10GbE ＊ 4 による bonding 接続

④　システムコンポーネント

システムを構成するコンポーネントは下記の通りです。



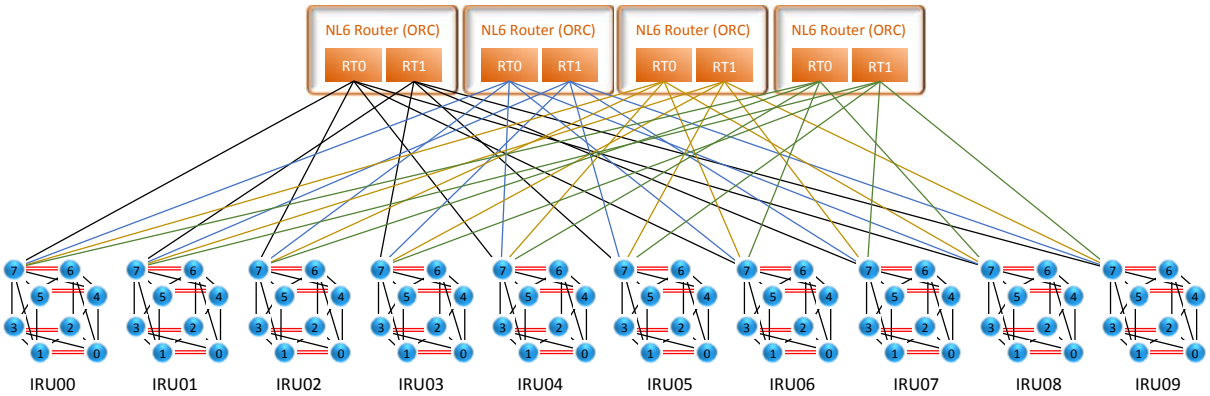
UV 2000 は 42 U のトールラック 3 ラックより構成されます。このシステムには 10 基の IRU というユニットを搭載しています。

1IRU は 8 計算ブレードが入り、1 計算ブレードには 1CPU が搭載された計算ノードが上下で 2 枚搭載されています。したがって、1IRU には合計 16 計算ノード、16CPU を搭載され、これが 10IRU では 160CPU になります。

⑤　NUMALink 6 インターコネクト

計算ノード間をつなぐインターコネクトは NUMALink 6 という独自のケーブルで接続し、全てのプロセッサが全てのメモリに直接アクセスが可能です。1 リンクあたりの双方向通信性能は 6.7GB/s です。 インターコネクトのトポロジーは Enhanced Hypercube です。

10IRU における Enhanced Hypercube のトポロジーを下記に示します。



⑥　ストレージ装置

外部ストレージ装置として、総容量 2.4PB を有します。この装置は InfiniBand 4xFDR で DAS 接続し、高速な IO 性能を有します。

UV 2000 と外部ストレージ装置のストレージ接続構成を下記に示します。

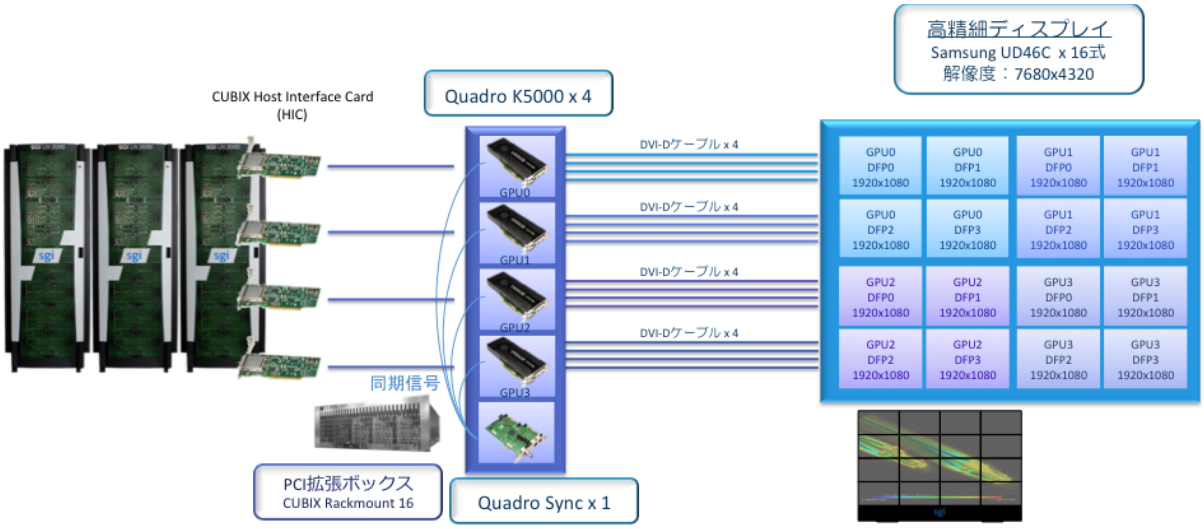
⑦　16 面タイルドディスプレイ

UV 2000 のディスプレイとして、縦 4 面、横 4 面の合計 16 台のタイルドディスプレイを有します。16 台の合計解像度は 7680 x 4320 画素 の 8K 解像度になります。

UV 2000 は NVIDIA K5000 を 8 枚有し、そのうち GPU 4 枚を使って 8K 解像度の映像を生成することができます。残りの GPU 4 枚は、遠隔可視化システムで利用します。

UV 2000 は、新館 3F に設置され、16 面タイルドディスプレイは新館 1F に設置されています。映像信号及びキーボード、マウスは光ケーブルを敷設し延長しています。

映像信号の概要図を下記に示します。



1.2.2. ソフトウェア構成

UV システムのソフトウェア構成を下記に示します。

システム名	SGI UV 2000
OS	SuSE Linux Enterprise Server 11
開発環境	Intel Composer XE 14.0.1
数値計算ライブラリ	Intel MKL 11.1
MPI 通信ライブラリ	SGI MPT 2.08
バッチシステム	PBS Professional 12.2
汎用可視化ソフトウェア	AVS Express Developer AVS Express PCE Enshight DR Paraview IDL ENVI ffmpeg ffplay osgviewer vmd(Visual Molecular Dynamics)
遠隔可視化ソフトウェア	SGI NICE DCV 2013.0
レイトレーシングソフトウェア	POV-Ray (SMP 対応版) 3.7.0
フーリエ変換ライブラリ	FFTW 3.3
入出力ライブラリ	HDF5 1.8.2 netcdf 4.2.1

2. システム利用環境

2.1. ログイン環境

2.1.1. UV システムへのログイン／ログアウト方法

UV システムは NICE (学内ネットワーク) には直接接続していないため、いったんログインのゲートウェイになっている フロントエンドサーバ (uvf) へ SSH でログイン後、uv へ ssh で接続します。

uvf へのログインは、スパコンの公開鍵認証でスパコンのログインノートと同様に PuTTY や Tera Term などのターミナルソフトを使ってログインします。

ホスト名 (FQDN)	サービス	ssh 認証方式	アクセス用途
uvf.cc.nagoya-u.ac.jp	ssh	公開鍵認証	NICE から UV へのログインゲートウェイ
uv	ssh	パスワード認証	uvf から uv への ssh ログイン

uvf から uv への ssh ログインはパスワード認証で、HPC ポータルにログインする際に使用するパスワードを入力します。

```
[wsuser@workstation ~]$ ssh uvf.cc.nagoya-u.ac.jp   ←  uvf に ssh でログイン
Enter passphrase for key  '/home/wsuser/.ssh/id_rsa' :      ←  公開鍵パスフレーズを入力
[scuser@uvf ~]$ ssh uv                                      ←  uv に ssh でログイン
Password:                                                   ←  パスワード認証 (passphrase でない)
Last login: Thu May  8 17:55:09 2014 from uvf.cc.nagoya-u.ac.jp


        Nagoya University Mixed Reality Large-scale Visualization System

                                    u v

                SGI UV 2000 160CPU/1280core 20TB Memory



[scuser@uv ~]$
```

ログアウトをする場合は、”exit” もしくは “logout”コマンドを実行します。

```
[scuser@uv ~]$ logout
[scuser@uvf ~]$ logout
[wsuser@workstation ~]$
```

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、uv への口グインを直接行うことができます。

【ワークステーション、PC 端末における “~/.ssh/config” ファイルの内容】

```
Host uv
    HostName uv
    HostKeyAlias nagoya-uv
    Port 22
    User scuser
    ProxyCommand ssh uvf nc %h %p
Host uvf
    HostName uvf.cc.nagoya-u.ac.jp
    Port 22
    User scuser
    IdentityFile ~/.ssh/id_rsa
```

上記設定ファイルを作成しておくと、ワークステーションや PC 端末より”ssh uv” で口グインすることが可能です。

```
[wsuser@workstation ~]$ ssh uv
Enter passphrase for key  ‘/home/wsuser/.ssh/id_rsa’ ;        ←  公開鍵パスフレーズを入力
Password:                                                     ←  パスワード認証 (passphrase でない)
Last login: Thu May  8 17:55:09 2014 from uvf.cc.nagoya-u.ac.jp


        Nagoya University Mixed Reality Large-scale Visualization System

                                    u v

                SGI UV 2000 160CPU/1280core 20TB Memory



[scuser@uv ~]$
```

ログアウトは “logout”もしくは”exit”コマンドを実行し、抜けてください。

```
[scuser@uv ~]$ logout
connection to uv closed.
Killed by signal 1.
[wsuser@workstation ~]$
```

2.2. TSS 環境

⑧　シェル環境

デフォルトのシェルは bash です。

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

⑨　言語環境

言語環境は C に設定されています。

```
[scuser@uv ~]$ echo $LANG
C
```

日本語環境に変更するには

```
[scuser@uv ~]$ export LANG=ja_JP.UTF-8
[scuser@uv ~]$ echo $LANG
ja_JP.UTF-8
```

⑩　プロセス制限

UV システムにログインした際のプロセス制限は下記の通りです。
ソフトリミットは ulimit コマンドで確認できます。

```
[scuser@uv ~]$ ulimit -a
core file size          (blocks, -c) 1
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 160279708
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) 17438498592
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) 43200
max user processes              (-u) 160279708
virtual memory          (kbytes, -v) 16520076240
file locks                      (-x) unlimited
```

ハードリミットは “ulimit -H”で確認できます。

```
[scuser@uv ~]$ ulimit -aH
core file size          (blocks, -c) unlimited
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 160279708
max locked memory       (kbytes, -l) 256
max memory size         (kbytes, -m) unlimited
open files                      (-n) 8192
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) unlimited
cpu time               (seconds, -t) unlimited
max user processes              (-u) 160279708
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

2.3. ファイルシステムの利用

UV システムで利用可能なファイルシステムとして、FX システムや CX システムと共通の home 領域、large 領域および UV 専用の領域として data 領域を有します。

ユーザが利用可能と割り当てられたファイルシステムを下記に示します。

ファイルシステム		総容量	備考
home	NFS		スパコンシステム（システム共通） ホーム・ファイルシステム
large	NFS		スパコンシステム 大容量ファイルシステム
data	xfs	2.4PB	UV 専用ファイルシステム /data/usr/GROUP/USER を用意

UV の専用領域である data 領域で利用可能なディレクトリは下記コマンドを実行することで確認することができます。

```
[scuser@uv:~]$ ls -ld /data/usr/`id -gn`/`id -un`
drwxr-xr-x 4 scuser 35 May 18 22:01 /data/usr/scgroup/scuser/
```

data 領域のディスク使用状況を確認する方法は、下記コマンドを実行します。

```
[scuser@uv:~]$ /usr/sbin/xfs_quota -c 'quota -u scuser' /data
Disk quotas for User scuser (XXXXX)
Filesystem             Blocks      Quota      Limit  Warn/Time      Mounted on
/dev/lxvm/data            176          0          0   00 [--------] /data
```

UV システムからスパコンのファイルシステム (/home および/large) はアクセスできますが、アクセス速度が遅いため高速に利用する場合は、ログインノード (uvf) 上で/data ファイルシステムにファイル転送してからご利用ください。

2.4. ファイル転送の利用

2.4.1. PC/ワークステーションから UV へのファイル転送方法

PC やワークステーションから uv へのファイル転送する場合、

⑪ PC/WS から uvf へ scp によるファイル転送
⑫ uvf から uv へ scp によるファイル転送
⑬ uvf に格納した中間ファイルの削除

という手続きを行ってください。

```
【1. PC/WS から uvf へ ファイル名”testfile”をファイル転送】
[wsuser@workstation ~]$ scp testfile scuser@uvf.cc.nagoya-u.ac.jp:/tmp
Enter passphrase for key ‘/home/wsuser/.ssh/id_rsa’ ;        ← 公開鍵パスフレーズを入力
testfile                                    100%   14KB  14.5KB/s   00:00

【2. uvf から uv へ ファイル名”testfile”をファイル転送】
[wsuser@workstation ~]$ ssh scuser@uvf.cc.nagoya-u.ac.jp
Enter passphrase for key ‘/home/wsuser/.ssh/id_rsa’ ;        ← 公開鍵パスフレーズを入力
 [scuser@uvf:~]$ scp /tmp/testfile uv:/tmp
Password:                                                    ← パスワード認証（passphrase でない）
testfile                                    100%   14KB  14.5KB/s   00:00

【3. uvf に格納したファイル名”testfile”を削除】
[scuser@uvf:~]$ rm /tmp/testfile
```

2.4.2. uv から PC/ワークステーションへのファイル転送方法

uv から PC/WS へのファイル転送する場合、

⑭ uv から uvf へ scp によるファイル転送
⑮ uvf から PC/WS へ scp によるファイル転送
⑯ uvf に格納した中間ファイルの削除

という手続きを行ってください。

```
【1. uv から uvf へ ファイル名”testfile”をファイル転送しログアウト】
[scuser@uvf:~]$ scp uv:/tmp/testfile /tmp
Password:                                          ←  パスワード認証（passphrase でない）
testfile                              100%   14KB  14.5KB/s   00:00
[scuser@uvf:~]$ logout
Connection to uvf.cc.nagoya-u.ac.jp closed.

【2. uvf から PC/WS へ ファイル名”testfile”をファイル転送】
[wsuser@workstation ~]$ scp scuser@uf.cc.nagoya-u.ac.jp:/tmp/testfile /tmp
Enter passphrase for key ‘/home/wsuser/.ssh/id_rsa’ ;      ←  公開鍵パスフレーズを入力
testfile                              100%   14KB  14.5KB/s   00:00

【3. uvf に再度ログインし、格納したファイル名”testfile”を削除しログアウト】
[wsuser@workstation ~]$ ssh scuser@uvf.cc.nagoya-u.ac.jp
Enter passphrase for key ‘/home/wsuser/.ssh/id_rsa’ ;      ←  公開鍵パスフレーズを入力
 [scuser@uvf:~]$ rm /tmp/testfile
[scuser@uvf:~]$ logout
[wsuser@workstation ~]$
```

2.4.3. openssh の多段 SSH を使用したファイル転送方法

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、uvf を介さずファイル転送を直接行うことができます。

【ワークステーション、PC 端末における “~/.ssh/config” ファイルの内容】

```
Host uv
    HostName uv
    HostKeyAlias nagoya-uv
    Port 22
    User scuser
    ProxyCommand ssh uvf nc %h %p
Host uvf
    HostName uvf.cc.nagoya-u.ac.jp
    Port 22
    User scuser
    IdentityFile ~/.ssh/id_rsa
```

上記設定ファイルを作成しておくと、ワークステーションや PC 端末より uvf に中間ファイルを作成せずにファイル転送することが可能です。

PC やワークステーションから uv へファイル転送を行う場合、下記のような操作になります。

```
[wsuser@workstation ~]$ scp testfile uv:/tmp
Enter passphrase for key '/home/wsuser/.ssh/id_rsa';        ← 公開鍵パスフレーズを入力
Password:                                                   ← パスワード認証（passphrase でない）
testfile                                  100%   14KB  14.5KB/s   00:00
Killed by signal 1.
```

また、uv から PC やワークステーションへファイル転送を行う場合、下記のような操作になります。

```
[wsuser@workstation ~]$ scp uv:/tmp/testfile /tmp
Enter passphrase for key '/home/wsuser/.ssh/id_rsa';        ← 公開鍵パスフレーズを入力
Password:                                                   ← パスワード認証（passphrase でない）
testfile                                  100%   14KB  14.5KB/s   00:00
Killed by signal 1.
```

3. 開発環境

UV システムで利用できる開発環境として、コンパイラは Intel コンパイラ、GNU コンパイラ、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		uvf	uv
コンパイラ	intel コンパイラ	×	○
	富士通コンパイラ	×	×
	GNU コンパイラ	○	○
数値計算ライブラリ	Intel MKL	×	○
	富士通 SSL II	×	×
MPI 通信ライブラリ	SGI MPT	×	○
	Intel MPI	×	×

3.1. Intel コンパイラの利用

3.1.1. コンパイル／リンクの概要

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@uv:~]$ command [options] sourcefile […]
```

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。

	言語処理系	コマンド名	実行例
Fortran	Fortran 77, 90, 95, 2003	ifort	$ ifort [options] program.f
C	ISO 標準 C	icc	$ icc [options] program.c
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、『3.6 Intel コンパイラのオプション』を参考ください。

Intel コンパイラおよび数値計算ライブラリ Intel MKL (Math Kernel Library)環境は下記にインストールされています。

⑰ uv:/opt/intel 配下

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。

⑱ Fortran/C/C++
⑲ /opt/intel/composerxe/Samples
⑳ Intel MKL
21 /opt/intel/composerxe/mkl/examples

3.1.3. 環境設定

UV にログインすると、Intel コンパイラの環境が設定されています。
設定されている環境設定は、以下の通りです。

```
【Intel コンパイラ及び数値計算ライブラリ MKL 環境設定】
[scuser@uv:~]$ cat /etc/profile.d/intel_compiler.csh
source /opt/intel/bin/compilervars.csh intel64
[scuser@uv:~]$ cat /etc/profile.d/intel_compiler.sh
source /opt/intel/bin/compilervars.sh intel64

【MPI 通信ライブラリ MPT 環境設定】
[scuser@uv:~]$ cat /etc/profile.d/mpivars.csh
module use /usr/share/modules/modulefiles
module load mpt/2.08
[scuser@uv:~]$ cat /etc/profile.d/mpivars.sh
module use /usr/share/modules/modulefiles
module load mpt/2.08
```

3.2. Fortran コンパイル / リンク / 実行方法

3.2.1. 逐次 Fortran プログラム (シリアル)

```
【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 -openmp -openmp-report1 prog_omp.f

【環境設定】
$ export KMP_AFFINITY=disabled   (Intel compiler 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
```

```
【コンパイル】
$ ifort -O3 -xAVX prog_mpi.f -lmpi   (-lmpi で SGI MPT をリンク)

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

3.2.3. MPI 並列プログラム

3.2.4. MPI +OpenMP ハイブリッド並列プログラム

```
【コンパイル】
$ ifort -O3 -xAVX -openmp -openmp-report1 prog_hyb.f -lmpi

【環境設定】
$ export KMP_AFFINITY=disabled   (Intel compiler 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 (シリアル)

```
【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 -openmp -openmp-report1 prog_omp.c

【環境設定】
$ export KMP_AFFINITY=disabled   (Intel compiler 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】
$ icc -O3 -xAVX prog_mpi.c -lmpi   (-lmpi で SGI MPT をリンク)

【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】
$ icc -O3 -xAVX -openmp -openmp-report1 prog_hyb.c -lmpi

【環境設定】
$ export KMP_AFFINITY=disabled   (Intel compiler 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 (シリアル)

```
【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 -openmp -openmp-report1 prog_omp.cpp

【環境設定】
$ export KMP_AFFINITY=disabled   (Intel compiler 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】
$ icpc -O3 -xAVX prog_mpi.cpp -lmpi   (-lmpi で SGI MPT をリンク)

【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】
$ icpc -O3 -xAVX -openmp -openmp-report1 prog_hyb.cpp -lmpi

【環境設定】
$ export KMP_AFFINITY=disabled   (Intel compiler 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 dplace や omplace コマンドを指定します。

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 に CPU 番号を指定すると、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、dplace や omplace コマンドはプロセスが割り当てられていない、空いている CPU へバインドします。

3.5.1. 逐次プログラム（シリアル）

```
【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 版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 版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 や omplace コマンドで配置するコアの指定方法

```
22  dplace コマンド
23  -c <cpulist>
```

24 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

25 -x2
26 OpenMP の管理スレッドを配置するスレッドから除外する
27 -s1
28 MPI の管理プロセスを配置するプロセスから除外する
29 omplace コマンド
30 -nt ${OMP_NUM_THREADS}
31 スレッド数を指定する
32 -c <cpulist>
33 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=2	16,17,20,21,24,25,28,39

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) を提供します。
富士通 SSL2 は使用できませんので、ご注意ください。

MKL の特徴は、

34 科学技術計算向け
35 インテルプロセッサにチューニング
36 マルチスレッド対応
37 スレッド並列化
38 スレッドセーフ
39 自動ランタイム・プロセッサ検出機能
40 C および Fortran のインターフェース

を有しています。

Intel MKL は数値計算ライブラリとして、下記の主な機能が含まれます。

41 BLAS
42 BLACS
43 LAPACK
44 ScaLAPACK
45 PBLAS
46 Sparse Solver
47 Vector Math Library (VML)
48 Vector Statistial Library (VSL)
49 Conventional DFTs and Cluster DFTs
50 FFTW interface

3.6.2. 逐次 C++プログラム (シリアル)

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】
$ ifort -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】
$ ifort -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】
$ ifort -lmkl_scalapack_lp64 -lmkl_blacs_sgimpt_lp64 -lmkl_intel_lp64 \
-lmkl_sequential -lmkl_core test.f -lmpi

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

3.6.3. スレッド並列プログラム (OpenMP)

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】
$ ifrot -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】
$ intel -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】
$ ifort -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】
$ ifort -lmkl_scalapack_lp64 -lmkl_blacs_sgimpt_lp64 -mkl=parallel \
test.f -lmpi
```

実行方法は下記の通り。

```
【環境設定】
$ export KMP_AFFINITY disabled   (Intel compiler 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
```

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 compiler 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、下記の注意が必要です。

実行方法	設定方法
逐次(シリアル)で実行	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。

オプションの種類	オプション	オプションのレベル
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ないます。
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。

オプションの種類	オプション	オプションのレベル
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 Xeon E5-2600 番台および 4600 番台の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

オプション	内容
-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	51 グローバルな最適化を有効化 52 組み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。 -axSSE4.2 および-xSSE4.2 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	-xHOST -O3 -ipo -no-prec-div -static 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

オプション	内容
-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。
-vec-report	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。
-vec-report-file=name	ベクタライザーからのレポートを name というファイルに出力します。
-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。

オプション	内容
-opt-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。
-opt-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。
-opt-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
-opt-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
-opt-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。

最適化フェーズ	最適化の内容	関連するオプション
ipo	Interprocedural Optimizer	-ipo, -ip
hlo	High-level Language Optimaizer	-O3 (Loop Unrolling)
ilo	Intermediate Language Scalar Optimizer	
hpo	High Performance Optimizer	
pgo	Profile Guided Optimizer	-prof_gen, -prof_use

3.7.3. 特定のプロセッサ向けの最適化オプション

オプション	内容
-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。

プロセッサ	プロセッサに対する最適化内容
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	SandyBridge(Intel Xeon E5-2600 番台および 4600 番台)向けの最適化を行い、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。
SSSE3	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

オプション	内容
-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

オプション	内容
-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】

オプション	内容
-openmp	プログラム中の OpenMP 指示行を有効にします。
-openmp-report [n]	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。(デフォルト) n=2: 上記に加え、MASTER, SINGLE, CRITICAL 指示句などを表示 -openmp-report オプションを設定しない場合、-openmp 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】

オプション	内容
-parallel	自動並列化を行います。
-par-report [n]	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。(デフォルト) 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: 並列化不可要因 (依存関係) も表示します。 -par-report 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

オプション	内容
-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。

3.7.9. その他のオプション

オプション	内容
-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。

4. ジョブ実行

4.1. ジョブシステム概要

UV 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 という）を導入しています。ユーザは、ジョブ開始時に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ています。

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

PBS Pro のキュー構成は下記の通りです。

キュー名	キューへの割当		1ユーザ		並列数		メモリ		経過時間		備考
	コア数	メモリ	実行数	投入数	標準値	制限値	標準値	制限値	標準値	制限値	
uv-middle	512	7.2TB	1	4	64	128	900GB	1.8TB	8h	72h	デフォルト
uv-large	512	7.2TB	1	4	256	512	3.6TB	7.2TB	8h	8h	

4.3. ジョブ投入方法

4.3.1. 逐次プログラム (シリアル)

ジョブ投入スクリプトを作成します。(スクリプト名: run_serial.sh)

```
【run_serial.sh スクリプト】
#!/bin/bash
#PBS -N serial              ←ジョブ名
#PBS -q uv-middle           ←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    ←リソース確保（8 コア）

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

dplace -c7 ./a.out          ←実行
```

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
331.uv
```

4.3.2. スレッド並列プログラム (OpenMP)

16 スレッドを使用する OpenMP 並列ジョブ投入スクリプトを作成します。(スクリプト名: run_omp.sh)

```
【run_omp.sh スクリプト】
#!/bin/bash
#PBS -N openmp                  ←ジョブ名
#PBS -q uv-middle               ←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=16       ←リソース確保（16 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=16       ←スレッド並列数の設定（16 スレッド）

dplace -x2 -c0-15 ./a.out       ←16 スレッドで OpenMP 実行
```

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_omp.sh
332.uv
```

4.3.3. MPI 並列プログラム

16 プロセスを使用する MPI 並列ジョブ投入スクリプトを作成します。(スクリプト名: run_mpi.sh)

```
【run_mpi.sh スクリプト】
#!/bin/bash
#PBS -N mpi                                  ←ジョブ名
#PBS -q uv-middle                            ←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=16                    ←リソース確保（16 コア）

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

mpirun -np 16 dplace -s1 -c0-15 ./a.out      ←16 プロセスで MPI 並列実行
```

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_mpi.sh
333.uv
```

4.3.4. MPI+OpenMP ハイブリッド並列プログラム

MPI が 4 プロセスと各プロセスから 4 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を作成します。(スクリプト名: run_hybrid.sh)

```
【run_hybrid.sh スクリプト】
#!/bin/bash
#PBS -N hybrid                    ←ジョブ名
#PBS -q uv-middle                 ←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=16         ←リソース確保（16 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=4          ←スレッド並列数の設定（4 スレッド）

mpirun -np 4 omplace -nt ${OMP_NUM_THREADS} -c0-15 ./a.out   ←hybrid 実行
```

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_hybrid.sh
334.uv
```

4.3.5. 大規模メモリプログラム (シリアル)

ジョブ投入スクリプトを作成します。(スクリプト名: run_mem.sh)

```
【run_mem.sh スクリプト】
#!/bin/bash
#PBS -N largemem                      ←ジョブ名
#PBS -q uv-middle                     ←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
```

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_mem.sh
335.uv
```

4.3.6. 会話型ジョブ

“qsub -I” でインタラクティブにジョブを投入することができます。

```
[scuser@uv:/data/usr/scgroup/scuser/serial]$ qsub -q uv-middle -l select=1:ncpus=16 -I
qsub: waiting for job 336.uv to start
qsub: job 336.uv ready

[scuser@uv:~]$ cd ${PBS_O_WORKDIR}
[scuser@uv:/data/usr/scgroup/scuser/serial]$ dplace -c7 ./a.out

    Number of    | Computed Integral |
 Interior Points |                   |
-----------------------------------------
             4   |   3.1415927E+00   |
-----------------------------------------
             8   |   3.7922378E+00   |
-----------------------------------------
...
-----------------------------------------
       8388608   |   4.0000000E+00   |
-----------------------------------------
      16777216   |   4.0000000E+00   |
-----------------------------------------
      33554432   |   4.0000000E+00   |
-----------------------------------------
      67108864   |   4.0000000E+00   |
-----------------------------------------

 CPU Time =   0.7640470      seconds
[scuser@uv:/data/usr/scgorup/scuser/serial]$ exit

logout
qsub: job 336.uv completed
```

4.3.7. ジョブ投入が受け付けられない場合

53　キュー設定以上のリソースを確保しようとした場合

uv-middle で 512CPU を確保しようとします。

【runmpi.sh の内容】

```
#!/bin/sh
#PBS -N mpi-genlte
#PBS -q uv-middle
#PBS -o stdout_middle_512p.log
#PBS -e stderr_middle_512p.log
#PBS -l select=1:ncpus=512
cd ${PBS_O_WORKDIR}
mpirun -np 512 dplace -s1 ./a.out
```

キューの制限で 128 までとなっているため、上記ジョブスクリプトを投入すると弾かれます。

```
$ qsub runmpi.sh
qsub: Job exceeds queue and/or server resource limits
```

54　ジョブの投入本数を超えた場合

同一ユーザで 4 本ジョブを投入しており、5 本目を投入するとはじかれます。

```
$ ndstat
JOBID    USER     QUEUE    JOBNAME    NODE CPUS Memory REQTIME  STAT  ELAPSE  START_TIME
-------- -------- -------- ---------- ---- ---- ------ -------- ----- -------  ------------
352      scuser4  uv-middl mpi-genlte    1   16  900gb   08:00     R   00:00  Jun 07 23:31
353      scuser4  uv-middl mpi-genlte    1   16  900gb   08:00     Q      --  --
354      scuser4  uv-middl mpi-genlte    1   16  900gb   08:00     Q      --  --
355      scuser4  uv-middl mpi-genlte    1   16  900gb   08:00     Q      --  --
```

表示されるエラーメッセージは次の通りです。

```
$ qsub runmpi.sh
qsub: would exceed queue generic's per-user limit
```

4.4. ジョブ状態表示

4.4.1. qstat コマンドによるジョブの状態表示

ジョブの状態は qstat コマンドで表示できます。

```
$ qstat
Job id            Name             User             Time Use S Queue
----------------  ---------------- ---------------  -------- - -----
404.uv            mpi-genlte       scuser1          28:28:46 R uv-middle
405.uv            mpi-genlte       scuser5          28:24:32 R uv-middle
406.uv            mpi-genlte       scuser3          28:18:08 R uv-middle
407.uv            mpi-genlte       scuser4          28:13:52 R uv-middle
408.uv            mpi-genlte       scuser2          111:23:0 R uv-large
409.uv            mpi-genlte       scuser5                 0 Q uv-middle
410.uv            mpi-genlte       scuser1                 0 Q uv-middle
411.uv            mpi-genlte       scuser1                 0 Q uv-large
```

全ての情報を表示する場合には、下記のオプションをつけて実行します。

```
$ qstat -asn
uv:
                                                                  Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
404.uv          scuser1  uv-middl mpi-genlte 264367   1 128  900gb 08:00 R 00:00
   uv/0*0
   Job run at Sun Jun 08 at 07:37 on (uv[32]:ncpus=8+uv[33]:ncpus=8+uv[34]...
405.uv          scuser5  uv-middl mpi-genlte 264626   1 128  900gb 08:00 R 00:00
   uv/1*0
   Job run at Sun Jun 08 at 07:37 on (uv[50]:ncpus=8+uv[51]:ncpus=8+uv[52]...
406.uv          scuser3  uv-middl mpi-genlte 264978   1 128  900gb 08:00 R 00:00
   uv/2*0
   Job run at Sun Jun 08 at 07:37 on (uv[70]:ncpus=8+uv[71]:ncpus=8+uv[64]...
407.uv          scuser4  uv-middl mpi-genlte 265255   1 128  900gb 08:00 R 00:00
   uv/3*0
   Job run at Sun Jun 08 at 07:37 on (uv[80]:ncpus=8+uv[81]:ncpus=8+uv[82]...
408.uv          scuser2  uv-large mpi-genlte 265711   1 512 3600gb 08:00 R 00:00
   uv/4*0
   Job run at Sun Jun 08 at 07:37 on (uv[96]:ncpus=8+uv[97]:ncpus=8+uv[98]...
409.uv          scuser5  uv-middl mpi-genlte    --    1 128  900gb 08:00 Q   --
    --
   Not Running: User has reached queue uv-middle running job limit.
410.uv          scuser1  uv-middl mpi-genlte    --    1 128  900gb 08:00 Q   --
    --
   Not Running: User has reached queue uv-middle running job limit.
411.uv          scuser1  uv-large mpi-genlte    --    1 256 3600gb 08:00 Q   --
    --
   Not Running: Insufficient amount of resource ncpus (R: 256 A: 255 T: 1279)
```

4.4.2. ndstat コマンドによるジョブの状態表示

ndstat コマンドを使用して、ジョブが UV システムのどの場所で実行されているかを表示します。

ndstat で表示される上段はジョブの実行状況を、下段は UV のレイアウトを表示しています。

uv-middle は IRU02～IRU05 を、uv-large は IRU06～IRU09 を使用して、ジョブが実行されていることが分かります。「アスタリスク（ * ）+3 桁の数字」で表示される文字列は、ジョブ ID の下 3 桁を表示しています。

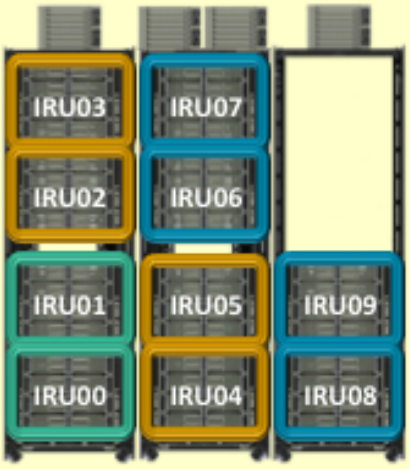
```
$ ndstat
JOBID    USER     QUEUE    JOBNAME    NODE CPUS Memory REQTIME  STAT  ELAPSE  START_TIME
-------- -------- -------- ---------- ---- ---- ------ -------- ----- -------  ------------
404      scuser1  uv-middl mpi-genlte    1  128  900gb    08:00     R   00:00  Jun 08 07:37
405      scuser5  uv-middl mpi-genlte    1  128  900gb    08:00     R   00:00  Jun 08 07:37
406      scuser3  uv-middl mpi-genlte    1  128  900gb    08:00     R   00:00  Jun 08 07:37
407      scuser4  uv-middl mpi-genlte    1  128  900gb    08:00     R   00:00  Jun 08 07:37
408      scuser2  uv-large mpi-genlte    1  512 3600gb    08:00     R   00:00  Jun 08 07:37
409      scuser5  uv-middl mpi-genlte    1  128  900gb    08:00     Q      --  --
410      scuser1  uv-middl mpi-genlte    1  128  900gb    08:00     Q      --  --
411      scuser1  uv-large mpi-genlte    1  256 3600gb    08:00     Q      --  --

 IRU  |  0     1     2     3     4     5     6     7
------|-----------------------------------------------
 000  |  --    --    --    --    --    --    --    --
 000  |  --    --    --    --    --    --    --    --
 001  |  --    --    --    --    --    --    --    --
 001  |  --    --    --    --    --    --    --    --
 002  | *404  *404  *404  *404  *404  *404  *404  *404
 002  | *404  *404  *404  *404  *404  *404  *404  *404
 003  | *405  *405  *405  *405  *405  *405  *405  *405
 003  | *405  *405  *405  *405  *405  *405  *405  *405
 004  | *406  *406  *406  *406  *406  *406  *406  *406
 004  | *406  *406  *406  *406  *406  *406  *406  *406
 005  | *407  *407  *407  *407  *407  *407  *407  *407
 005  | *407  *407  *407  *407  *407  *407  *407  *407
 006  | *408  *408  *408  *408  *408  *408  *408  *408
 006  | *408  *408  *408  *408  *408  *408  *408  *408
 007  | *408  *408  *408  *408  *408  *408  *408  *408
 007  | *408  *408  *408  *408  *408  *408  *408  *408
 008  | *408  *408  *408  *408  *408  *408  *408  *408
 008  | *408  *408  *408  *408  *408  *408  *408  *408
 009  | *408  *408  *408  *408  *408  *408  *408  *408
 009  | *408  *408  *408  *408  *408  *408  *408  *408
```

TSS

uv-middle

uv-large



4.5. ジョブキャンセル方法

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…]]
```

【ジョブを投入】
```
$ qsub runmpi.sh
412.uv
```

【ジョブが実行中であることを確認】
```
$ qstat
Job id            Name             User             Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
412.uv            mpi-genlte       scuser4          00:19:12 R uv-middle
```

【ジョブを削除】
```
$ qdel 412
```

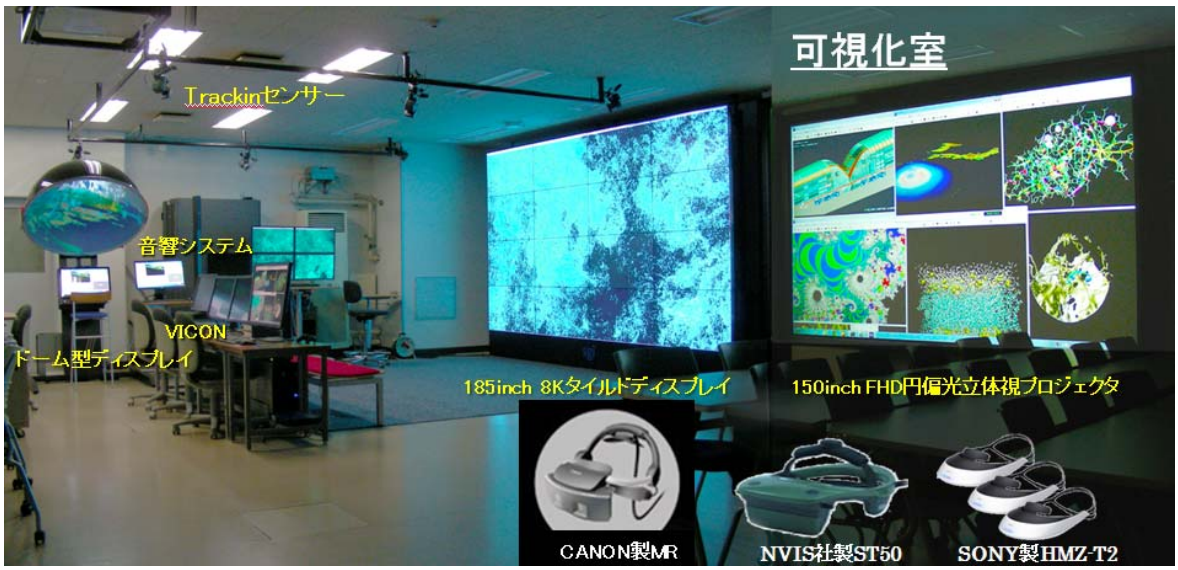
【実行中のジョブがなくなっていることを確認】
```
$ qstat
$
```

5. 可視化環境

ここでは、情報基盤センター1F 可視化室に設置されている可視化システムについて紹介します。各システムの利用方法は、操作マニュアルをご覧ください。

可視化システムは、次のシステムで構成されます。

- ８K タイルドディスプレイ・システム（高精細可視化システム）
- リモート可視化システム
- フル HD 円偏光立体視システム
- ドーム型ディスプレイ・システム
- 据え置き型 HMD ディスプレイ・システム
- 携帯型 HMD ディスプレイ・システム



5.1. ８K タイルドディスプレイ

８K タイルドディスプレイは 46 インチ液晶モニター16 面（4×4）で構成され、シミュレーションの可視化結果や医療画像および衛星画像等を高精細かつ大画面で可視化することができます。利用環境は、Linux と Windows が利用できます。Linux 環境は、大規模メモリと 1280CPU コア、NVIDIA K5000 グラフィックスカード 8 枚を搭載した「SGI UV2000 システム」を使って、大規模データを可視化することができます。操作は、８K タイルドディスプレイ横の４K モニターとキーボードとマウスを使用します。

5.2. リモート可視化システム

リモート可視化システムは、「SGI UV2000 システム」の利用環境を、リモートから利用するためのシステムです。NICE DCV を搭載した SGI VizServer ソフトウエアと NVIDIA K5000 グラフィックスカード 4 枚を使って動作し、すべての OpenGL アプリケーションは、NVIDIA グラフィックスカードを搭載したリモートの SGI VizServer システムでネイティブに実行されます。そ

して、作成された画像は圧縮･暗号化され、ネットワーク経由でローカルディスプレイに送信されます。

5.3. フル HD 円偏光立体視システム

フル HD 円偏光立体視システムは、NVIDIA Quadro FX6000 グラフィックスカードを搭載した WindowsPC，150 インチシルバースクリーン，円偏光フィルターを搭載した 2 台のフル HD DLP プロジェクタ，円偏光メガネ，ゲームコントローラから成るシステムです。大スクリーンにプロジェクタを使って立体投影し、円偏光メガネを使って可視化することができます。

5.4. ドーム型ディスプレイシステム

ドーム型ディスプレイシステムは、NVIDIA Quadro6000 グラフィックスカードを搭載した WindowsPC，直径 1.2m ドーム型スクリーン（ドーム視野角が 180 度の半球），ドームマスター（円周魚眼撮影）形式の画像および動画を投影する機能を有するフル HD ステレオ対応プロジェクタ，ゲームコントローラから成るシステムです。複数人が広視野立体映像を歪みを感じずに同時に見ることができ，空間共有体験を実現することができます。また、液晶シャッターメガネを使った立体視にも対応しています。

5.5. 据え置き型 HMD ディスプレイシステム

据え置き型ヘッドマウント･ディスプレイ MREAL(Mixed Reality)システムは、NVIDIA Quadro FX6000 グラフィックスカードを搭載した WindowsPC，Canon 社製 MR ヘッドマウント型ディスプレイシステムとジャイロセンサー、反射マーカー及び光学式の位置姿勢センサー（VICON）からなるシステムです。MREAL ヘッドマウント･ディスプレイは、カメラで撮影した 3 次元映像（現実空間画像）と 3 次元 CG 画像（仮想 CG 映像）を合成して立体視し、位置センサーを使って可視化空間内を移動して任意の方向から可視化データを閲覧することができます。

5.6. 据え置き型 HMD ディスプレイシステム

携帯型ヘッドマウント･ディスプレイシステムは、NVIDIA QuadroK3000M グラフィックスカードを搭載したノート型 WindowsPC， NVIS 社製 ST50 ヘッドマウント･ディスプレイ（クローズタイプとシースルータイプ兼用），SONY 製 HMZ-T2 ヘッドマウント･ディスプレイ（クローズタイプ），ゲームコントローラから成るシステムです。

6. 可視化アプリケーション一覧

UV システムで利用可能な可視化アプリケーションは、次のとおりです。
括弧内は、起動コマンドです。

AVS/Express Developer (XP)
AVS/Express PCE (スクリプト作成)
Enshight (スクリプト作成)
ENVI (envi)　……　合成開口レーダ SARscape 対応
ffmpeg (ffmpeg)
ffplay (ffplay)
IDL (/opt/ENVI/idl/bin/idl)
Mplayer (mplayer)
Osgviewer (psgviewer)
Paraview (/opt/sw/paraview/bin/paraview)
POV-Ray (/opt/POV-Ray/bin/povray)
3D AVS player (avsplayer)