大阪大学 D3センター
利用者講習会

# スーパーコンピュータ バッチシステム入門／応用

2026年6月10日

日本電気株式会社
インフラ・テクノロジーサービス事業部門
コンピュート統括部

© NEC Corporation 2026

\Orchestrating a brighter world

NECは、安全・安心・公平・効率という社会価値を創造し、

誰もが人間性を十分に発揮できる持続可能な社会の実現を目指します。

## バッチシステム入門／応用

1. バッチシステムの概要
2. ジョブの投入方法
3. 各種計算資源の利用方法
4. ジョブスクリプトのテクニック
5. 高度なジョブ投入・実行方法
6. SX-Aurora TSUBASA 公開情報

# 1. バッチシステムの概要

# 1. バッチシステムの概要

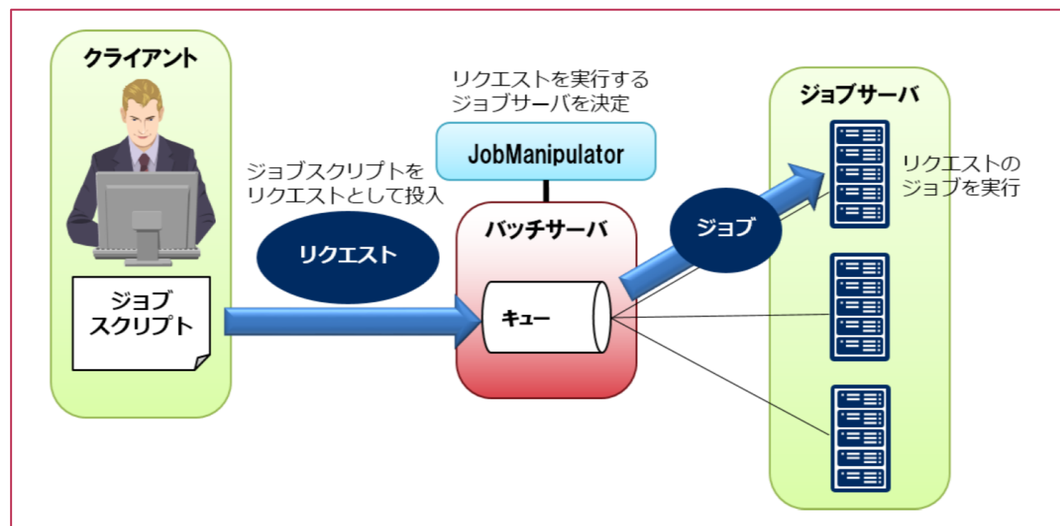
- スーパーコンピュータは、多くの利用者で共同利用するため、利用者の計算資源要求(ジョブ)を受け付け、計算資源が空いたタイミングでジョブを順次実行するバッチシステムを採用しています。
- バッチシステム内で、ジョブの実行を制御・管理するものが、ジョブ管理システムです。SQUIDでは「NEC NQSV」を採用しています。

※「NEC NQSV」における”リクエスト”と”ジョブ”
「NEC NQSV」の製品マニュアルでは以下の用語が使われます。
・**リクエスト**：ユーザがフロントエンドから要求するバッチ利用の単位
・**ジョブ**：個々の計算ノード上で実行される、リクエスト内の実行単位
本講習会では、他システムも含めた理解のしやすさを考慮し、"ジョブ"という用語を、
フロントエンドから要求するバッチ利用の単位(NQSVのリクエストに相当)の意味で使用します。


© NEC Corporation 2026

# 1. バッチシステムの概要

- 利用方法には、**「会話ジョブ」**と**「バッチジョブ」**の2種類があります。
- フロントエンドからジョブ管理システムにジョブ実行を要求（ジョブを投入）します。
- 要求されたジョブ要求は、他のジョブ要求の優先度や要求資源量、ジョブサーバの利用状況などがジョブ管理システムによって加味・判断され、実行順序が決定されます。





© NEC Corporation 2026

# 2. ジョブの投入方法

2.1. 会話ジョブ投入方法
2.2. バッチジョブ投入方法
2.3. ジョブ管理システムのコマンドの使い方

\Orchestrating a brighter world NEC

# 2.1. 会話ジョブ投入方法

- 会話ジョブは、会話的（インタラクティブ）に計算ノードを利用するジョブです。会話ジョブを使用する場合は**qloginコマンド**を使用します。
- 会話ジョブ投入コマンドのサンプル(汎用CPUノードで実行する例)を以下に記載します。

**$ qlogin -q INTC --group=G01234 [オプション]**
-q キュー名…会話キューを指定
--group=グループ名…課金対象にするグループ名を指定

- qloginコマンドを実行すると、リクエストIDが採番され、以下の例のように標準出力に表示されます。

(出力例)
Request **123.sqd** submitted to queue: INTC.
Waiting for **123.sqd** to start.


© NEC Corporation 2026

# 2.2. バッチジョブ投入方法(1)

## ◆ ジョブスクリプトファイルの作成

- ジョブスクリプトファイルは、バッチジョブ内での実行内容、キューの指定、計算資源の要求量やその他オプションを記述するファイルです。
- このジョブスクリプト内に、**計算を行うための実行コマンド**を書くとともに、ジョブ管理システムに対して**要求する資源量などをオプション(#PBS )として記述**していきます。なお、オプションを指定しない場合は**既定値が適用**されます。

・ジョブスクリプトのサンプル(汎用CPUノードでシリアル実行する例)を以下に記載します。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID                        → バッチジョブを投入するキュー名の指定
#PBS --group=G01234                  → 課金対象にするグループ名
#PBS -l elapstim_req=01:00:00        → ジョブの最大実行時間の要求値　1時間の例
#PBS -l cpunum_job=76                → 使用するCPUコア数の要求値
#PBS -m b                            → バッチジョブ実行開始時にメールを送信
#PBS -M user@hpc.cmc.osaka-u.ac.jp   → 送信先アドレス

#------- Program execution -----------
module load BaseCPU     → ベース環境をロードします

cd $PBS_O_WORKDIR                    → qsub実行時のカレントディレクトリへ移動
./a.out                              → プログラムの実行
```

要求する資源量などを記載

計算を行うための実行コマンドを記載


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 2.2. バッチジョブ投入方法(2)

## ◆ ジョブ管理システムへのバッチジョブ要求

### ■ qsub - バッチジョブの投入

```
$ qsub [オプション] [ジョブスクリプトファイル名]
```

※[オプション]については、頁11-14にて記載

qsubコマンドを実行すると、リクエストIDが採番され、以下の例のように標準出力に表示されます。

```
(出力例)
        Request 1182.sqd submitted to queue: SQUID.
```

・計算ノードのファイルシステム

計算ノードでは以下ファイルシステムを利用することが可能です。フロントエンドからもアクセス可能です。

- home領域(/sqfs/home/(利用者番号)) ※ジョブ実行開始時のカレントディレクトリ
- 拡張領域(/sqfs/work/(グループ番号)/(利用者番号)) ※5TB(購入により拡張可能)
- 高速領域(/sqfs/ssd/(グループ番号)/(利用者番号)) ※要購入

home領域の容量は10GBであるため、**拡張領域または高速領域の利用を推奨します。**

・環境変数PBS_O_WORKDIRの利用

ジョブスクリプトでは、環境変数PBS_O_WORKDIRにqsub実行時のカレントディレクトリのパスが自動的に代入されます。バッチスクリプト内で、以下のコマンドを実行すると、qsub実行時のカレントディレクトリへ簡単に移動できますので、**実行ディレクトリでジョブを投入することを推奨します。**

```
$ cd $PBS_O_WORKDIR
```


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 2.2. バッチジョブ投入方法(3)

◆ ノード共通オプション(必須)

| オプション | 機能 |
| --- | --- |
| -q [バッチキュー名] | バッチジョブを投入するキューを指定します。<br>キューの詳細は「3. 各主計算資源環境の使い方」で説明します。 |
| --group=[グループ名] | 指定グループでジョブを実行します。<br>指定グループのポイントが消費されます。 |


© NEC Corporation 2026

# 2.2. バッチジョブ投入方法(4)

## ◆ 汎用CPUノード向けオプション

| オプション | 機能 |
|---|---|
| -b [ノード数] | ジョブを実行するノード数を指定します。 |
| -T [使用MPIライブラリ名] | MPI実行を行う場合に指定が必要です。<br>Intelコンパイラを利用している場合、IntelMPIが利用可能です。<br>-T intmpi と指定してください。<br>NVIDIA HPC SDKコンパイラを利用している場合、OpenMPIが利用可能です。-T openmpiを指定してください。 |

※MPIライブラリとモジュール、コンパイラの関係
　本システムではコンパイラ、ライブラリ、アプリケーションの環境変数設定を「Environment modules」で管理しています。MPIライブラリとモジュール、コンパイラの関係は以下のとおりです。

| 計算環境 | MPIライブラリ | コンパイラ | 推奨環境<br>モジュール名 |
|---|---|---|---|
| 汎用CPUノード | Intel MPI | Intel oneAPI | BaseCPU |
| ベクトルノード | NEC MPI | NEC SDK for VE | BaseVEC |
| GPUノード | Open MPI | NVIDIA HPC SDK<br>CUDA | BaseGPU |
| (なし) | Open MPI | GNU Compiler | BaseGCC |

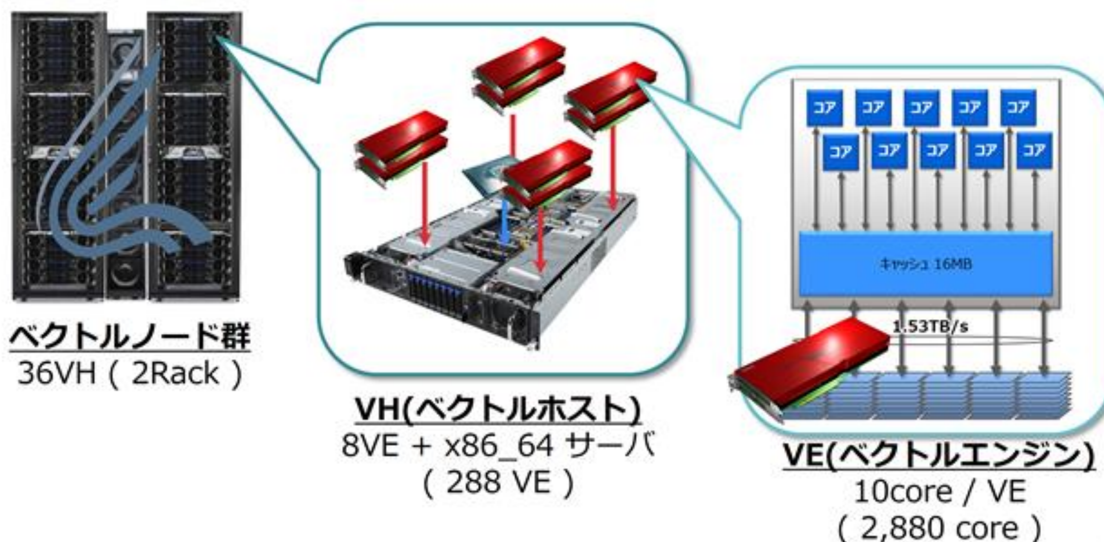

© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 2.2. バッチジョブ投入方法(5)

## ◆ ベクトルノード向けオプション

| オプション | 機能 |
|---|---|
| **--venode=[総VE数]** | **利用する総VE数を指定します。<br>ベクトルノードを使用する場合、必須オプションです。** |
| --venum-lhost=[論理ホストあたりのVE数] | 論理ホストを構成するVE数を指定します。<br>（大まかには、[1VH内で確保するVE数]とご理解ください） |
| -T necmpi | SX-Aurora TSUBASAにてMPI実行を行う場合に指定が必要です。 |

【参考】ベクトル計算ノード
x86_64サーバであるVH(ベクトルホスト)に8VEを搭載し、全36VHで構成されます。





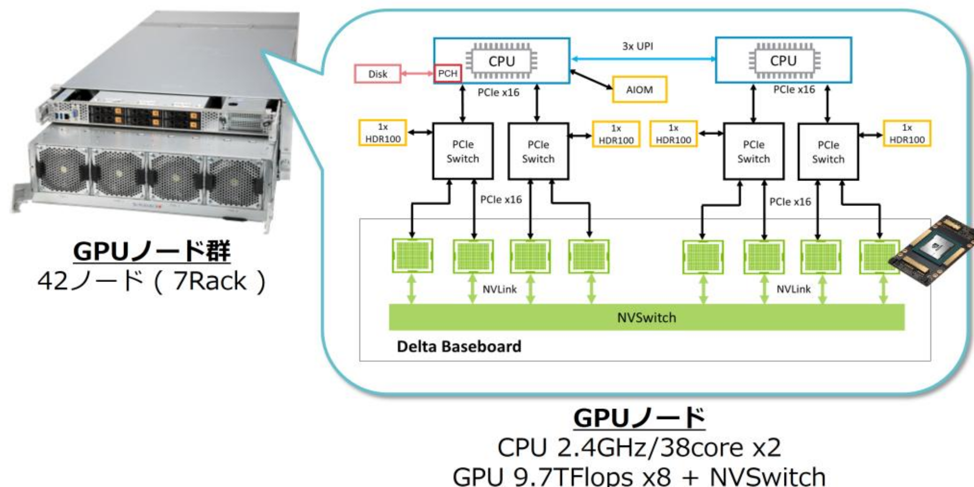
© NEC Corporation 2026

# 2.2. バッチジョブ投入方法(6)

## ◆ GPUノード向けオプション

| オプション | 機能 |
| --- | --- |
| **-l gpunum_job=[ノードあたりのGPU数]** | **ノードあたりの利用するGPU数を指定します。<br>GPUノードを使用する場合、必須オプションです。** |
| --gpunum-lhost=[ノードあたりのGPU数] | 上記「-l gpunum_job」と同様です。 |
| -b [ノード数] | ジョブを実行するノード数を指定します。 |
| -T [使用MPIライブラリ名] | MPI実行を行う場合に指定が必要です。<br>Intelコンパイラを利用している場合、IntelMPIが利用可能です。<br>-T intmpi と指定してください。<br>NVIDIA HPC SDKコンパイラを利用している場合、OpenMPIが利用可能です。-T openmpiを指定してください。 |

【参考】GPUノード
CPUとして「Intel Xeon Platinum 8368」、GPUとして「NVIDIA A100」を有し、4Uの筐体に8GPUを搭載しています。




**GPUノード群**
42ノード（7Rack）

**GPUノード**
CPU 2.4GHz/38core x2
GPU 9.7TFlops x8 + NVSwitch


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 2.3. ジョブ管理システムのコマンドの使い方(1)

- バッチジョブの確認
  - qstat - 投入したジョブの状況を確認

**$ qstat**
ユーザ自身が投入したジョブの一覧を表示

**$ qstat -l**
省略なく一覧を表示
上記のオプションはアルファベットのLの小文字です。
「--adjust-column」を同時に指定することで幅の最適化が行われます。

**$ qstat -f [リクエストID]**
特定ジョブの詳細を表示


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 2.3. ジョブ管理システムのコマンドの使い方(2)

- バッチジョブの確認
  - qstatgroup – ユーザが所属するグループが投入したジョブを確認

```
$ qstatgroup
  ユーザが所属するグループが投入したジョブの一覧を表示
```

  - sstat – 投入リクエストの実行開始時刻を確認

```
$ sstat
  投入リクエストの実行開始時刻を表示
  実行開始時刻が決まっていない場合は、時刻は表示されない。
```


© NEC Corporation 2026

# 2.3. ジョブ管理システムのコマンドの使い方(3)

- バッチジョブの標準出力/標準エラー出力表示
  - qcat – ジョブ実行中に、標準出力/標準エラー出力を表示

```
$ qcat -e [リクエストID]
  標準エラー出力を表示する場合
$ qcat -o [リクエストID]
  標準出力を表示する場合
```

以下のオプションを組み合わせることも可能です。

-f　　ファイルの内容が増え続けるとき、追加されたデータを出力します。
-n　　指定した行数分表示します。（無指定時は10行分です。）
-b　　ファイルの先頭から表示します。（無指定時は最終行から表示されます。）


© NEC Corporation 2026

# 2.3. ジョブ管理システムのコマンドの使い方(4)

- バッチジョブの保留・解除
  - qhold – 投入したジョブを保留

**$ qhold [リクエストID]**
ホールドすることでスケジューリング対象から外れ、ジョブを実行開始されない状態にする。

  - qrls – 保留にしたジョブを保留解除

**$ qrls [リクエストID]**
ホールド状態が解除され、再度スケジューリング対象の状態にする。


© NEC Corporation 2026

# 2.3. ジョブ管理システムのコマンドの使い方(5)

- ジョブ情報の表示
  - acstat – ユーザ自身が過去に投入したジョブ情報を表示

| | |
|---|---|
| **$ acstat** | コマンド実行時から24時間以内のジョブの情報を表示 |
| **$ acstat -A** | コマンド実行年度のジョブの情報を表示 |

  - acstatgroup – ユーザが所属するグループが過去に投入したジョブ情報を表示

| | |
|---|---|
| **$ acstatgroup** | コマンド実行時から24時間以内のジョブの情報を表示 |
| **$ acstatgroup -A** | コマンド実行年度のジョブの情報を表示 |

- バッチジョブの削除
  - qdel – 投入したリクエストを削除

**$ qdel [リクエストID]**
バッチジョブが実行状態(RUN)の場合は、最初にSIGTERMを送り、その後SIGKILLが送られます。


© NEC Corporation 2026

# 2.3. ジョブ管理システムのコマンドの使い方(6)

- キューの状態確認
  - qstat -Q – バッチジョブが実行されるキューの状態を表示

```
$ qstat -Q
```

```
[EXECUTION QUEUE] Batch Server Host: sqd
========================================
QueueName       SCH JSVs ENA STS   PRI  TOT ARR WAI QUE PRR RUN POR EXT HLD HOL RST SUS MIG STG CHK
--------------- --- ---- -------- ---- ------------------------------------------------------------
SC1               1 1518 ENA ACT    10  169   0   0   5   0 164   0   0   0   0   0   0   0   0   0

主な項目の説明は以下のとおりです。
ENA : ENA → ジョブ投入可能、DIS → ジョブ投入不可
STS : ACT → 実行可能、INA → 実行不可
QUE : 実行待ちジョブ件数
RUN : 実行中ジョブ件数
```


© NEC Corporation 2026

# 3. 各種計算資源環境の使い方

3.1. SQUIDシステム
3.2. OCTOPUSシステム

\Orchestrating a brighter world NEC

# 3.1.1. 【SQUID】ジョブクラス

- それぞれのジョブクラスは、ジョブ管理システム上のキューに対応しており、利用者はキューにジョブを投入することで計算環境の利用が可能です。
- キューは、利用者がジョブを直接投入する投入キューと、実行順を待ち合わせる実行キューに分かれます。

・汎用CPUノード向けジョブクラス

| 利用方法 | ジョブクラス | 利用可能経過時間 | 利用可能最大Core数 | 利用可能メモリ | 同時利用可能ノード数 | 備考 |
|---|---|---|---|---|---|---|
| 共有利用 | SQUID | 120時間 | 38,912core (76c/ノード) | 124TiB (248GB/ノード) | 512 | |
| | SQUID-R | 120時間 | 38,912core (76c/ノード) | 124TiB (248GB/ノード) | 512 | ※1 |
| | SQUID-H | 120時間 | 38,912core (76c/ノード) | 124TiB (248GB/ノード) | 512 | ※2 |
| | SQUID-S | 120時間 | 38core | 124GiB | 1 | ※3 |
| | DBG | 10分 | 152core (76c/ノード) | 496GiB (248GB/ノード) | 2 | デバッグ用 |
| | INTC | 10分 | 152core (76c/ノード) | 496GiB (248GB/ノード) | 2 | 会話ジョブ利用 |
| 占有利用 | mySQUID | 無制限 | 76core × 占有ノード数 | 248GiB × 占有ノード数 | 占有数 | |

※1 NW帯域が狭い経路の利用を許容して、実行待ち時間を短縮されたい方向け
※2 ポイント消費を多くして高優先度ジョブを投入し、実行待ち時間を短縮されたい方向け
※3 他のジョブとのノード内の共有を許容して、ポイント消費を抑えたい方向け
※利用可能経過時間を指定しない場合、利用経過時間の規定値はDBGが10分、その他は1時間となります
※Core数を指定しない場合、利用Core数の規定値はSQUID-Sが38core、その他は76coreとなります
※メモリを指定しない場合、利用メモリの規定値はSQUID-Sが124GB、その他は248GBとなります


© NEC Corporation 2026

# 3.1.1. 【SQUID】ジョブクラス

・ベクトルノード向けジョブクラス

| 利用方法 | ジョブクラス | 利用可能経過時間 | 利用可能最大Core数 | 利用可能メモリ | 同時利用可能VE数 | 備考 |
|---|---|---|---|---|---|---|
| 共有利用 | SQUID | 120時間 | 2,560core (10c/VE) | 12TiB (48GB/VE) | 256 | |
| | SQUID-H | 120時間 | 2,560core (10c/VE) | 12TiB (48GB/VE) | 256 | ※1 |
| | SQUID-S | 120時間 | 40core (10c/VE) | 192GiB (48GB/VE) | 4 | ※2 |
| | DBG | 10分 | 40core (10c/VE) | 192GiB (48GB/VE) | 4 | デバッグ用 |
| | INTV | 10分 | 40core (10c/VE) | 192GiB (48GB/VE) | 4 | 会話ジョブ利用 |
| 占有利用 | mySQUID | 無制限 | 10core × 占有VE数 | 48GiB × 占有VE数 | 占有数 | |

※1 ポイント消費を多くして高優先度ジョブを投入し、実行待ち時間を短縮されたい方向け
※2 他のジョブとのノード内の共有を許容して、ポイント消費を抑えたい方向け
※利用可能経過時間を指定しない場合、利用経過時間の規定値はDBGが10分、その他は1時間となります


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 3.1.1. 【SQUID】ジョブクラス

・GPUノード向けジョブクラス

| 利用方法 | ジョブクラス | 利用可能経過時間 | 利用可能最大Core数 | 利用可能メモリ | 同時利用可能ノード数 | 備考 |
|---|---|---|---|---|---|---|
| 共有利用 | SQUID | 120時間 | 2,432core<br>(76c/ノード) | 15.75TiB<br>(504GB/ノード) | 32 | |
| | SQUID-H | 120時間 | 2,432core<br>(76c/ノード) | 15.75TiB<br>(504GB/ノード) | 32 | ※1 |
| | SQUID-S | 120時間 | 38core | 252GiB | 1 | ※2 |
| | DBG | 10分 | 152core<br>(76c/ノード) | 1,008GiB<br>(504GB/VE) | 2 | デバッグ用 |
| | INTG | 10分 | 152core<br>(76c/ノード) | 1,008GiB<br>(504GB/VE) | 2 | 会話ジョブ利用 |
| 占有利用 | mySQUID | 無制限 | 76core ×<br>占有ノード数 | 504GiB ×<br>占有ノード数 | 占有数 | |

※1 ポイント消費を多くして高優先度ジョブを投入し、実行待ち時間を短縮されたい方向け
※2 他のジョブとのノード内の共有を許容して、ポイント消費を抑えたい方向け
※利用経過時間を指定しない場合、利用経過時間の規定値はDBGが10分、その他は1時間となります
※Core数を指定しない場合、利用Core数の規定値はSQUID-Sが8core、その他は76coreとなります
※メモリを指定しない場合、利用メモリの規定値はSQUID-Sが252GB、その他は504GBとなります


© NEC Corporation 2026

# 3.1.2. 【SQUID】汎用CPUノードの使い方(1)

## ◆ シリアル実行利用方法

1ノード内でのプログラム実行を想定したジョブスクリプトのサンプルです。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00

#------- Program execution -----------
module load BaseCPU
module load xxx/xxx                 →  コンパイル時にmodule loadしていたものを記載

cd $PBS_O_WORKDIR
./a.out
```


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 3.1.2. 【SQUID】汎用CPUノードの使い方(2)

## ◆ スレッド並列利用方法

1ノード内でのスレッド並列プログラム実行を想定したジョブスクリプトのサンプルです。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS -v OMP_NUM_THREADS=76        → 並列実行数を指定

#------- Program execution -----------
module load BaseCPU
module load xxx/xxx               → コンパイル時にmodule loadしていたものを記載

cd $PBS_O_WORKDIR
./a.out
```

利用時の注意点
実行スクリプトにて「OMP_NUM_THREADS」を忘れずに指定してください。
「OMP_NUM_THREADS」を指定しない場合、あるいは間違った値を指定してしまった場合、
意図しない並列数での実行となる可能性があります。


© NEC Corporation 2026

# 3.1.2. 【SQUID】汎用CPUノードの使い方(3)

## ◆ MPI利用方法(IntelMPI)

4ノードで合計304プロセスを生成するIntelMPIのプログラム実行を想定したジョブスクリプトのサンプルです。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS -b 4                          → ノード数を指定
#PBS -T intmpi                     → IntelMPIを指定

#------- Program execution -----------
module load BaseCPU
module load xxx/xxx                → コンパイル時にmodule loadしていたものを記載

cd $PBS_O_WORKDIR
mpirun ${NQSV_MPIOPTS} -np 304 ./a.out  → mpirunの引数に${NQSV_MPIOPTS}を指定
```


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 3.1.2. 【SQUID】汎用CPUノードの使い方(4)

## ◆ MPI＋ノード内並列利用方法

4ノードで各ノード上に1プロセスを生成し、各プロセスで76個のスレッドを生成する
IntelMPIプログラム実行を想定したジョブスクリプトのサンプルです。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS -b 4                          →  ノード数を指定
#PBS -T intmpi                     →  IntelMPIを指定
#PBS -v OMP_NUM_THREADS=76         →  スレッド並列実行数を指定

#------- Program execution -----------
module load BaseCPU
module load xxx/xxx                →  コンパイル時にmodule loadしていたものを記載

cd $PBS_O_WORKDIR
mpirun ${NQSV_MPIOPTS} -np 4 ./a.out  →  mpirunの引数に${NQSV_MPIOPTS}を指定
                                        各ノードで1コアに1プロセスを割り当て、76スレッドを生成する
```

 © NEC Corporation 2026
\Orchestrating a brighter world NEC

# 3.1.3. 【SQUID】ベクトルノードの使い方(1)

## ◆ シリアル実行利用方法

1VE内でのプログラム実行を想定したジョブスクリプトのサンプルです。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID-S                     →  1～4VE使用する場合は、SQUID-Sキューを指定
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS --venode=1                     →  全体で1VE使用することを指定

#------- Program execution -----------
module load BaseVEC
module load xxx/xxx                 →  コンパイル時にmodule loadしていたものを記載

cd $PBS_O_WORKDIR
./a.out
```

 © NEC Corporation 2026

\Orchestrating a brighter world NEC

# 3.1.3. 【SQUID】ベクトルノードの使い方(2)

## ◆ スレッド並列利用方法

1VE内でのスレッド並列プログラム実行を想定したジョブスクリプトのサンプルです。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID-S                  →  1～4VE使用する場合は、SQUID-Sキューを指定
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS -v OMP_NUM_THREADS=10        →  並列実行数を指定
#PBS --venode=1                   →  全体で1VE使用することを指定

#------- Program execution -----------
module load BaseVEC
module load xxx/xxx               →  コンパイル時にmodule loadしていたものを記載

cd $PBS_O_WORKDIR
./a.out
```

利用時の注意点
実行スクリプトにて「OMP_NUM_THREADS」を忘れずに指定してください。
「OMP_NUM_THREADS」を指定しない場合、あるいは間違った値を指定してしまった場合、
意図しない並列数での実行となる可能性があります。


© NEC Corporation 2026

# 3.1.3. 【SQUID】ベクトルノードの使い方(3)

## ◆ MPI利用方法(NEC MPI)

40VEで合計400プロセスを生成するNEC MPIのプログラム実行を想定したジョブスクリプトのサンプルです。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS --venode=40                      → 全体で40VE使用することを指定
#PBS -T necmpi                        → NEC MPIを指定

#------- Program execution -----------
module load BaseVEC
module load xxx/xxx                   → コンパイル時にmodule loadしていたものを記載

cd $PBS_O_WORKDIR
mpirun -venode -np 400 ./a.out        → MPIプロセスのVHおよびVEへの割り当てはNQSVが自動で実施
```


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 3.1.3. 【SQUID】ベクトルノードの使い方(4)

## ◆ MPI＋ノード内並列利用方法

40VEでVEあたり1プロセスを生成し、各プロセスで10個のスレッドを生成するNEC MPIプログラム実行を想定したジョブスクリプトのサンプルです。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS --venode=40                 →  全体で40VE使用することを指定
#PBS -T necmpi                   →  NEC MPIを指定
#PBS -v OMP_NUM_THREADS=10       →  スレッド並列実行数を指定

#------- Program execution -----------
module load BaseVEC
module load xxx/xxx              →  コンパイル時にmodule loadしていたものを記載

cd $PBS_O_WORKDIR
mpirun -venode -np 40 ./a.out    →  MPIプロセスのVHおよびVEへの割り当てはNQSVが自動で実施
```

利用時の注意点
「OMP_NUM_THREADS」で指定するVEあたりのスレッド数は、コア数である10を超えないようにしてください。


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 3.1.4. 【SQUID】GPUノードの使い方(1)

## ◆ シリアル実行利用方法

1ノード内でのプログラム実行を想定したジョブスクリプトのサンプルです。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS -l gpunum_job=1              → ノードあたり1GPU使用することを指定

#------- Program execution -----------
module load BaseGPU
module load xxx/xxx               → コンパイル時にmodule loadしていたものを記載

cd $PBS_O_WORKDIR
./a.out
```


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 3.1.4. 【SQUID】GPUノードの使い方(2)

## ◆ スレッド並列利用方法

1ノード内でのスレッド並列プログラム実行を想定したジョブスクリプトのサンプルです。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS -l gpunum_job=8              →  ノードあたり8GPU使用することを指定
#PBS -v OMP_NUM_THREADS=76        →  並列実行数を指定

#------- Program execution -----------
module load BaseGPU
module load xxx/xxx               →  コンパイル時にmodule loadしていたものを記載

cd $PBS_O_WORKDIR
./a.out
```

利用時の注意点
実行スクリプトにて「OMP_NUM_THREADS」を忘れずに指定してください。
「OMP_NUM_THREADS」を指定しない場合、あるいは間違った値を指定してしまった場合、
意図しない並列数での実行となる可能性があります。


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 3.1.4. 【SQUID】GPUノードの使い方(3)

## ◆ MPI利用方法(OpenMPI)

2ノードで合計4プロセスを生成するOpenMPIのプログラム実行を想定したジョブスクリプトのサンプルです。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS -b 2                        →  ノード数を指定
#PBS -l gpunum_job=8             →  ノードあたり8GPU使用することを指定
#PBS -T openmpi                  →  OpenMPIを指定
#PBS -v NQSV_MPI_MODULE=BaseGPU  →  OpenMPIの起動に必要となるモジュール名を指定

#------- Program execution -----------
module load BaseGPU
module load xxx/xxx              →  コンパイル時にmodule loadしていたものを記載

cd $PBS_O_WORKDIR
mpirun ${NQSV_MPIOPTS} -np 4 -npernode 2  ${PBS_O_WORKDIR}/mpi_prog
                                 →  mpirunの引数に${NQSV_MPIOPTS}を指定
```

利用時の注意点
NQSV_MPI_MODULEに複数のモジュールを指定する場合は、コロン(:)区切りで指定することが可能です。
(例) BaseGPUとcuda/11.8を指定する
#PBS -v NQSV_MPI_MODULE=BaseGPU:cuda/11.8


© NEC Corporation 2026

# 3.1.4. 【SQUID】GPUノードの使い方(4)

## ◆ MPI+ノード内並列利用方法(OpenMPI)

2ノードで各ノード上に2プロセスを生成し、OpenMPにより各プロセスで12個のスレッドを生成するOpenMPIプログラム実行を想定したジョブスクリプトのサンプルです。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS -b 2                          →  ノード数を指定
#PBS -l gpunum_job=8               →  ノードあたり8GPU使用することを指定
#PBS -T openmpi                    →  OpenMPIを指定
#PBS -v OMP_NUM_THREADS=12         →  スレッド並列実行数を指定
#PBS -v NQSV_MPI_MODULE=BaseGPU  →  OpenMPIの起動に必要となるモジュール名を指定

#------- Program execution -----------
module load BaseGPU
module load xxx/xxx                →  コンパイル時にmodule loadしていたものを記載

cd $PBS_O_WORKDIR
mpirun ${NQSV_MPIOPTS} -np 4 -npernode 2 --bind-to socket ¥
                                     --report-bindings  ${PBS_O_WORKDIR}/mpi_prog
                                   →  各ノードで、各ソケットに1プロセスを割り当て
                                      12スレッドを生成するため、「--bind-to socket」等を指定
```

利用時の注意点
NQSV_MPI_MODULEに複数のモジュールを指定する場合は、コロン(:)区切りで指定することが可能です。
(例) BaseGPUとcuda/11.8を指定する
#PBS -v NQSV_MPI_MODULE=BaseGPU:cuda/11.8


© NEC Corporation 2026

# 3.2. 【OCTOPUS】ジョブクラス

OCTOPUSのジョブクラスは以下を参照ください。
https://www.hpc.cmc.osaka-u.ac.jp/system/manual/octopus2-use/jobclass/

OCTOPUSのジョブスクリプトファイルのサンプルは以下を参照ください。
https://www.hpc.cmc.osaka-u.ac.jp/system/manual/octopus2-use/jobscript/


© NEC Corporation 2026

# 4. ジョブスクリプトのテクニック

# 4.1. ジョブスクリプトの推奨オプション(1)

- 経過時間制限値の指定

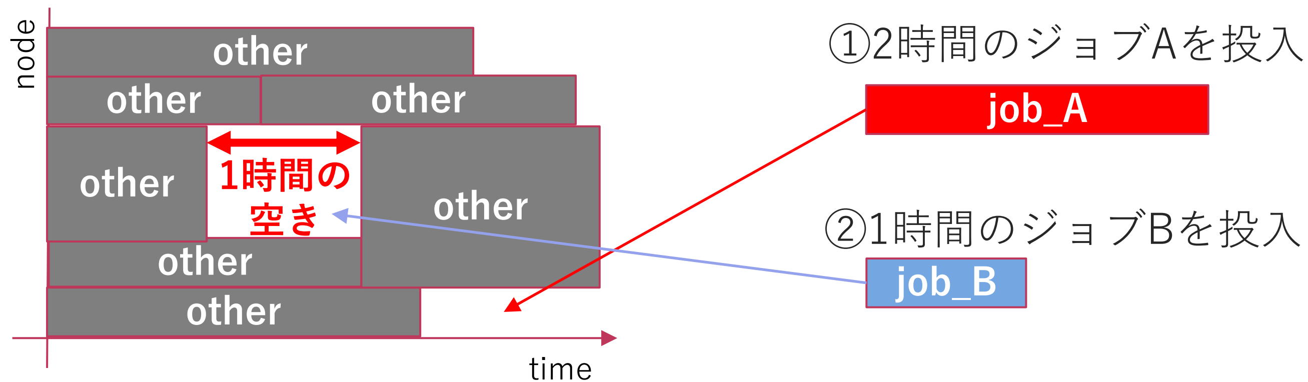
ジョブ管理システムでは要求資源に合わせて最短で実行可能なノードにジョブをアサインします。

ジョブスクリプトにて、

**経過時間制限値(#　PBS -l elapstim_req=[経過時間制限値])**に

**適切な時間(プログラムの実行時間＋α)**を指定することで、

アサインされやすくなり、実行開始時間が早くなる可能性があります。





© NEC Corporation 2026

# 4.1. ジョブスクリプトの推奨オプション(2)

## ◆ ノード共通おすすめオプション

必須ではありませんが、便利なオプションを紹介します。

| オプション | 機能 |
|---|---|
| -N [ジョブ名] | ジョブの名前を指定します。<br>指定がなければ、バッチスクリプト名がジョブ名になります。 |
| -o [標準出力ファイル名] | バッチジョブの標準出力の出力ファイル名を指定します。<br>指定がなければ、ジョブ投入時のディレクトリに「ジョブ名.oリクエストID」のファイル名で出力されます。 |
| -e [標準エラー出力ファイル名] | バッチジョブの標準エラー出力の出力ファイル名を指定します。<br>指定がなければ、リクエスト投入時のディレクトリに「ジョブ名.eリクエストID」のファイル名で出力されます。 |
| -j [o,e] | バッチジョブの標準出力と標準エラー出力をマージします。<br>o：マージした結果を標準出力に出力します。<br>e：マージした結果を標準エラー出力に出力します。<br>（-jとoもしくはeの間にはスペースが必要です） |


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 4.1. ジョブスクリプトの推奨オプション(3)

## ◆ ノード共通おすすめオプション

| オプション | 機能 |
|---|---|
| -M [メールアドレス] | メールの送信先を指定します。複数指定する場合は<br>　-M メールアドレス1,メールアドレス2<br>のように「,」で区切ってください。 |
| -m [b,e,a] | バッチジョブの状態の変化についてのメールを送ります。<br>b：ジョブが開始したときにメールを送信<br>e：ジョブが終了したときにメールを送信<br>a：ジョブが異常終了したときにメールを送信<br>（-mとbやeの間にはスペースが必要です）<br>複数を指定することができます。<br>例）開始時と終了時にメール通知<br>　-m be |
| -v 環境変数 | バッチジョブを実行するときに使用する環境変数を指定します。<br>複数ノードで実行する場合、全てのノードで指定した環境変数の値が設定されます。 |
| -r { y \| n } | バッチジョブのリラン可否を指定します。<br>　y：リラン可能<br>　n：リラン不可 |


© NEC Corporation 2026

# 4.2. 環境変数の活用(1)

◆ ジョブスクリプトの作成では、環境変数を活用すると便利です。
以下は、コア数、ノード数を環境変数として設定するジョブスクリプトのサンプルです。

2ノードで76並列(38コア/ノード)のプログラム実行を想定しています。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS -v NODE=2
#PBS -v CORE=38
#PBS -b ${NODE}                     →  ノード数を指定
#PBS -l cpunum_job=${CORE}          → ノード当たりのコア数を指定
#PBS -T intmpi

#------- Program execution -----------
module load BaseCPU
module load xxx/xxx

cd $PBS_O_WORKDIR                   → qsub実行時のカレントディレクトリへ移動

NP=`expr ${NODE} "*" ${CORE}`       → 総並列数を計算
mpirun ${NQSV_MPIOPTS} -np ${NP} -ppn ${CORE} ./a.out
```

実行するノード数を変更する場合、ジョブスクリプトの変更箇所を1箇所にすることができます。


© NEC Corporation 2026

# 4.2. 環境変数の活用(2)

## ◆ ジョブ管理システムの既定値として設定される環境変数

以下は、リクエスト実行時に既定値として設定される環境変数です。

| 環境変数名 | 説明 | -vオプションによる変更可否 |
|---|---|---|
| PBS_ENVIRONMENT | バッチリクエストがバッチ処理であることを示す、"PBS_BATCH"が設定 されます。 | × |
| PBS_JOBID | バッチリクエストのジョブIDが設定されます。 | × |
| PBS_SUBREQNO | サブリクエスト番号が設定されます（パラメトリックリクエストの場合）<br>パラメトリックリクエストは「5.1. パラメトリックリクエスト機能」で説明します。 | ○ |
| PBS_JOBNAME | バッチリクエスト名が設定されます。 | × |
| PBS_NODEFILE | リクエストを構成するジョブが実行されているホスト一覧が記載されたフ ァイルへのパスが設定されます。 | ○ |
| PBS_O_HOME | クライアントホスト上の環境変数 HOME が設定されます。 | ○ |
| PBS_O_HOST | クライアントホスト名が設定されます。 | ○ |
| PBS_O_LANG | クライアントホスト上の環境変数 LANG が設定されます。 | ○ |
| PBS_O_LOGNAME | クライアントホスト上の環境変数 LOGNAME が設定されます。 | ○ |
| PBS_O_MAIL | クライアントホスト上の環境変数 MAIL が設定されます。 | ○ |
| PBS_O_PATH | クライアントホスト上の環境変数 PATH が設定されます。 | ○ |
| PBS_O_SHELL | クライアントホスト上の環境変数 SHELL が設定されます。 | ○ |
| PBS_O_TZ | クライアントホスト上の環境変数 SHELL が設定されます。 | ○ |
| PBS_O_WORKDIR | クライアントホスト上の作業用ディレクトリを設定します。 | ○ |

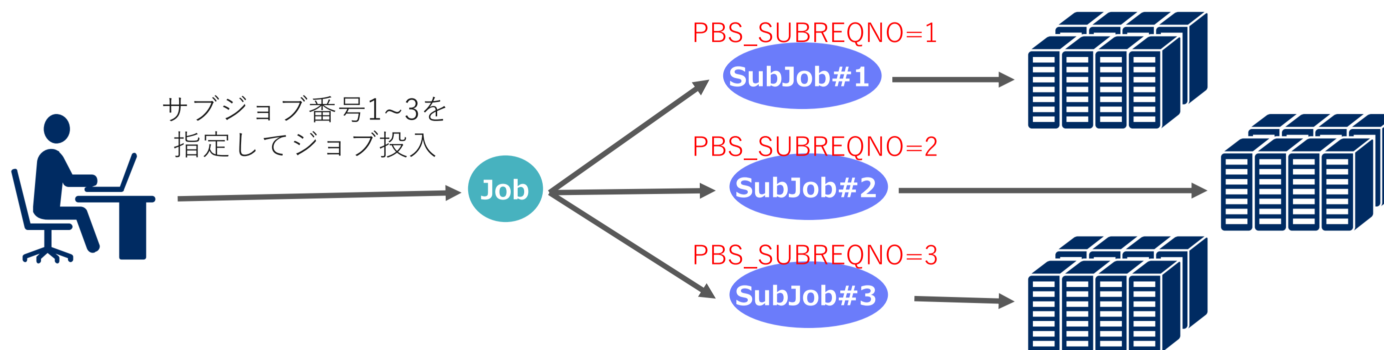

© NEC Corporation 2026

# 5. 高度なジョブ投入・実行方法

# 5.1. パラメトリックリクエスト機能(1)

## ◆ パラメトリックリクエストとは

入力となるパラメータや入力ファイルを変えながら、同一のジョブスクリプトをサブジョブとして複数回実行するジョブです。




### ■ 利用シーン

同一ソルバーを用い、複数の入力パターンのあるアンサンブル実行


© NEC Corporation 2026

# 5.1. パラメトリックリクエスト機能(2)

◆ パラメトリックリクエストの投入方法

サブジョブ番号を指定したジョブスクリプトを作成し、qsubコマンドにて投入します。

ジョブスクリプトのサンプル

```
#!/bin/bash
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=01:00:00
#PBS -t 1,2,3
# ↑ パラメトリックリクエストのサブジョブ番号として1から3を指定(-t 1-3 と指定することもできます)

echo $PBS_SUBREQNO
# ↑ 環境変数PBS_SUBREQNOにサブジョブごとのサブジョブ番号が設定されています
```


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 5.1. パラメトリックリクエスト機能(3)

◆ パラメトリックリクエストの状態確認

投入したパラメトリックリクエストの状態は、qstatコマンドで確認することができます。

```
$ qstat
RequestID       ReqName  UserName Queue     Pri STT S   Memory      CPU    Elapse R H M Jobs
--------------- -------- -------- -------- ---- --- - -------- -------- -------- - - - ----
1234[].sqd      qsub.sh  user     SC1        0 QUE -        -        -        - Y Y Y    1
  ↑qstatコマンドのオプションなしの場合、投入したジョブの情報が表示されます

$ qstat -s
RequestID       ReqName  UserName Queue     Pri STT S   Memory      CPU    Elapse R H M Jobs
--------------- -------- -------- -------- ---- --- - -------- -------- -------- - - - ----
1234[1].sqd     qsub.sh  user     SC1        0 QUE -    0.00B     0.00        0 Y Y Y    1
1234[2].sqd     qsub.sh  user     SC1        0 QUE -    0.00B     0.00        0 Y Y Y    1
1234[3].sqd     qsub.sh  user     SC1        0 QUE -    0.00B     0.00        0 Y Y Y    1
  ↑qstatコマンドに-sオプションを指定した場合、サブジョブの情報が表示されます
    []内の数字はサブジョブ番号です

$ qstat -R
RequestID       ReqName  UserName Queue     Pri STT R H M Jobs  TOTAL ACTIVE   DONE
--------------- -------- -------- -------- ---- --- - - - ---- ------ ------ ------
1234[].sqd      qsub.sh  user     SC1        0 QUE Y Y Y    1      3      3      0
  ↑qstatコマンドに-Rオプションを指定した場合、サマリが表示されます
    TOTAL に、全サブジョブ数、ACTIVE に、バッチサーバ上で保持しているサブジョブ数、
    DONE に、終了したサブジョブ数(削除された数を含む)を表示します
```


© NEC Corporation 2026

# 5.1. パラメトリックリクエスト機能(4)

## ◆ パラメトリックリクエストの削除

パラメトリックリクエストは全サブジョブを含む親ジョブに対する削除と、サブスクリプト単位での削除を行うことができます。

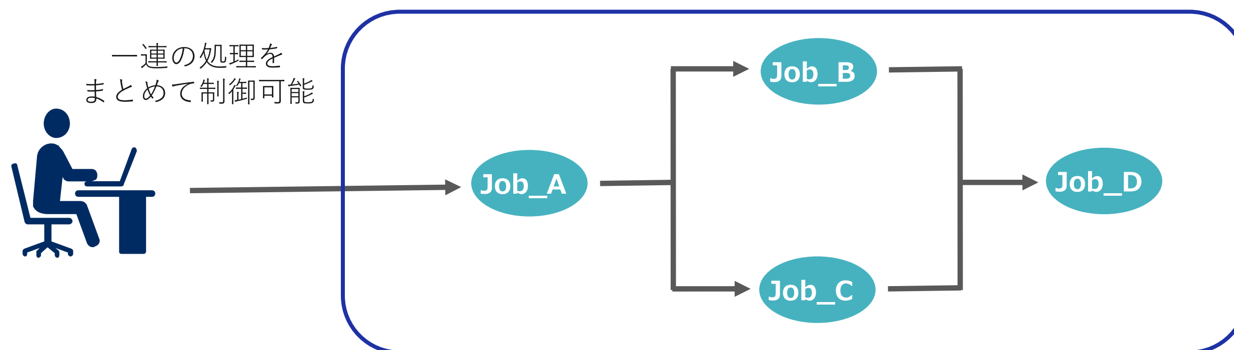
```
$ qdel 1234[].sqd
Request 1234[].sqd was deleted.
  ↑全サブジョブを削除します

$ qdel 1234[2].sqd
Request 1234[2].sqd was deleted.
  ↑指定されたサブジョブのみ削除します
```


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 5.2. ワークフロー機能(1)

## ◆ ワークフローとは

ジョブの投入・実行を含む一連の処理をシェルスクリプトとして記述し、ジョブの実行順序等を制御する機能です。




### ■ 利用シーン

(例1) 最初に前処理用の1本のジョブを実行
その後、前処理で生成したファイルを利用する複数のジョブを実行

(例2) ジョブA、B、…、Zを順次実行
途中で正常終了しないジョブがあった場合、以降の処理をすべて自動削除

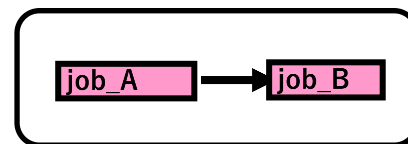

© NEC Corporation 2026

# 5.2. ワークフロー機能(2)

## ◆ 指定可能な依存関係、制御

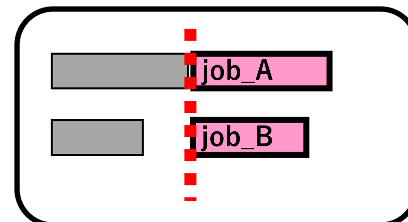
### ■ 順次実行（--afterオプション）

「--after job_A ./job_B」のように指定することで、
job_Aの実行終了後、job_Bの実行が開始されます。



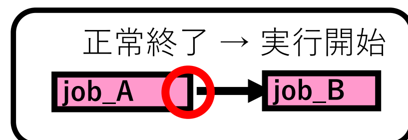

### ■ 同時実行（--parallelオプション）

「--parallel ./job_A ./job_B」のように指定することで、
job_Aとjob_Bが同時に実行開始されます。

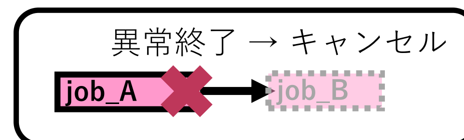



### ■ エラー発生時の後続ジョブのキャンセル（--cancel-afterオプション）

「--cancel-after ./job_A」のように指定した上で、
「--after job_A ./job_B」のように指定することで、
job_Aが異常終了した場合、job_Bはキャンセル(qdel)されます。




ジョブAが正常終了した場合、
ジョブBの実行が開始される




ジョブAが異常終了した場合、
ジョブBはキャンセルされる


© NEC Corporation 2026

# 5.2. ワークフロー機能(3)

◆ ワークフローのジョブ実行制御を使うメリット

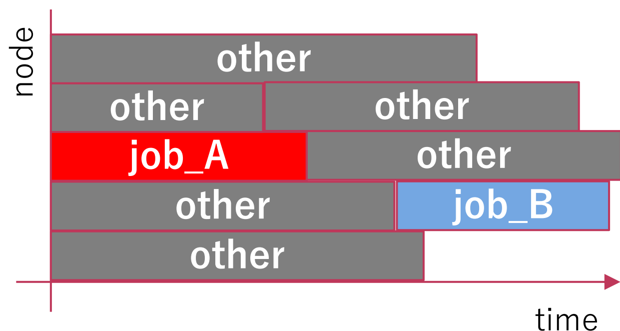
■ ワークフローを使わずに順次実行制御

(例:qwaitの利用)

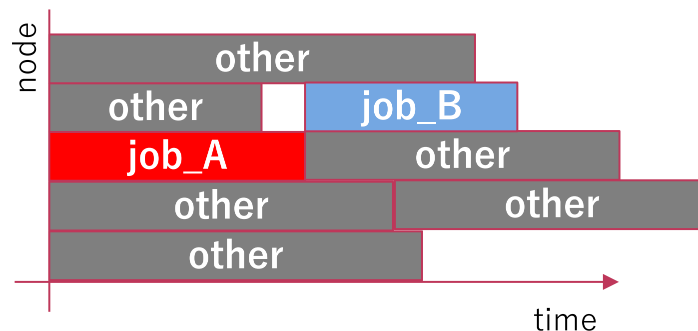
フロントエンドサーバ上でqwaitを利用し、1本目のジョブが終了したことを確認したら次のジョブをqsubする。

```
#!/bin/bash
qwait  <reqID_A>    ←1本目のジョブが終了するまでここで待つ
qsub ./B_req        ←2本目のジョブを投入
```

■ ワークフローを使うと隙間なくスケジューリングされる




qwaitを利用する場合
(job_Aの実行終了後、job_Bを投入)




ワークフローを利用する場合
(job_Aの実行終了後、job_Bを投入)

**⇒ job_Bの実行開始時刻が早い**


© NEC Corporation 2026

# 5.2. ワークフロー機能(4)

## ◆ ワークフロースクリプトの作成

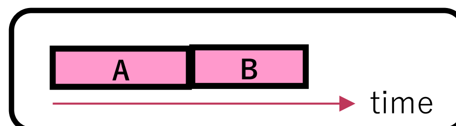
### ■ ワークフロースクリプトとは？

- ワークフローの一連の処理を記述したシェルスクリプト
- 後述するwstartコマンドを利用して、ワークフロースクリプトを投入します。

### ■ スクリプト例(Aの終了後、Bの実行開始)

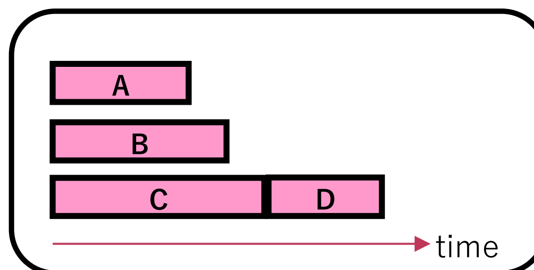
```
#!/bin/bash
qsub -N A A.sh
qsub --after A -N B B.sh
```

※-Nオプションによりジョブに
　名前を付けることができる。




### ■ スクリプト例(A、B、Cのすべてが終了後、Dの実行開始)

```
#!/bin/bash
qsub -N A A.sh
qsub -N B B.sh
qsub -N C C.sh
qsub --after A,B,C -N D D.sh
```



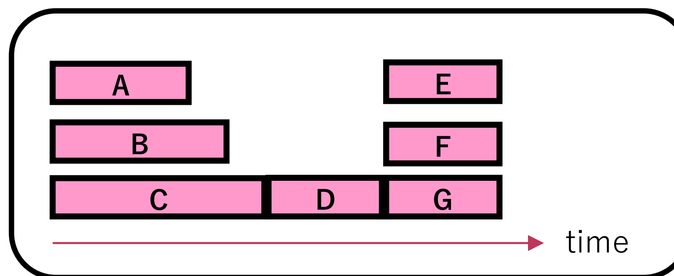


© NEC Corporation 2026

# 5.2. ワークフロー機能(5)

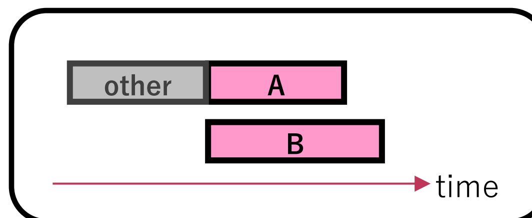
- スクリプト例(A、B、Cのすべてが終了後、Dの実行開始、Dの終了後、E、F、Gの開始)

```
#!/bin/bash
qsub -N A A.sh
qsub -N B B.sh
qsub -N C C.sh
qsub --after A,B,C -N D D.sh
qsub --after D -N E E.sh
qsub --after D -N F F.sh
qsub --after D -N G G.sh
```




- スクリプト例(A、Bを実行開始)

```
#!/bin/bash
qsub --parallel A.sh B.sh
```



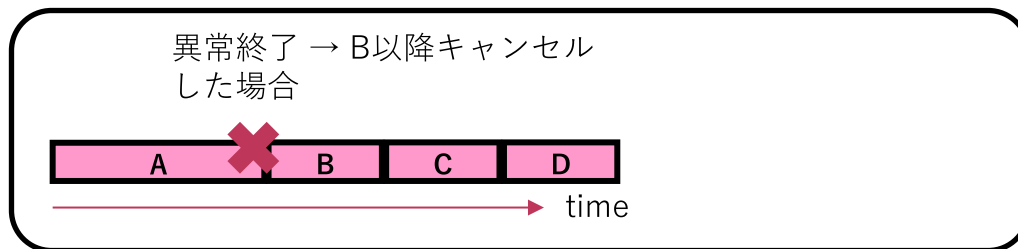


© NEC Corporation 2026

# 5.2. ワークフロー機能(6)

■ スクリプト例(Aが異常終了した場合、B以降をすべてキャンセル)

```
#!/bin/bash

qsub --cancel-after -N A A.sh
qsub --after A -N B B.sh
qsub --after B -N C C.sh
qsub --after B -N D D.sh
```




※以下の場合が異常終了とみなされます
- 終了ステータスが0以外
- qdelで削除された
- システム障害によって終了した


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 5.2. ワークフロー機能(7)

## ◆ ワークフローの投入

wstartコマンドの引数にワークフロースクリプトを指定します。

```
$ wstart wfl.sh
```

※ワークフロースクリプトの名前をwfl.shとしています。

## ◆ ワークフローの状態確認

wstatコマンドを利用します。引数にワークフローIDを指定してください。

```
$ wstat 123
```

※ワークフローIDを123としています。
　ワークフローIDは投入時に表示されます。もしくは、引数なしでwstatコマンドを実行することで確認可能です。

＜出力例＞

```
$ wstat 123
WFL-ID        RequestID       RequestName  Stat  Exit
------------- --------------- ------------ ----  ----
123_sqd       12345.sqd       A            RUN    -
123_sqd       12346.sqd       B            QUE    -
```


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 5.2. ワークフロー機能(8)

## ◆ ワークフローの削除

wdelコマンドの引数にワークフローIDを指定します。

```
$ wdel 123
```

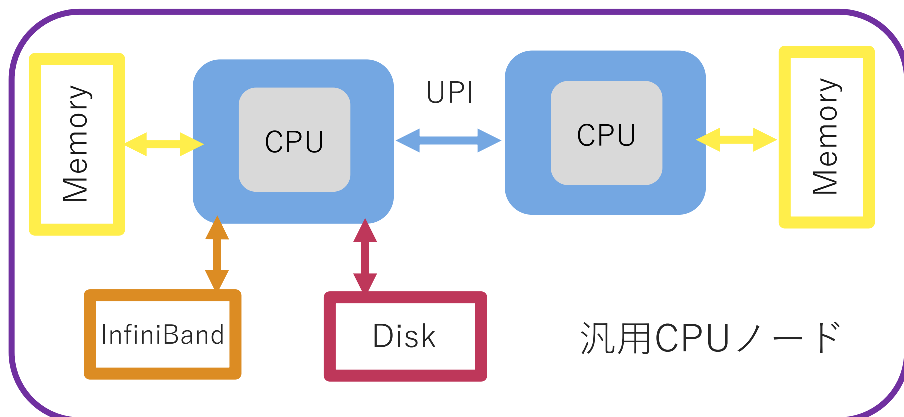
これにより、ワークフロー内で投入されたジョブがすべて削除されます。

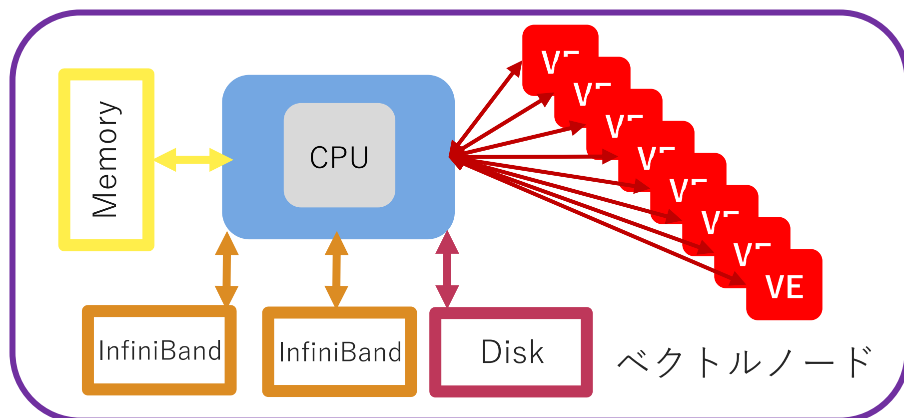

© NEC Corporation 2026
\Orchestrating a brighter world NEC

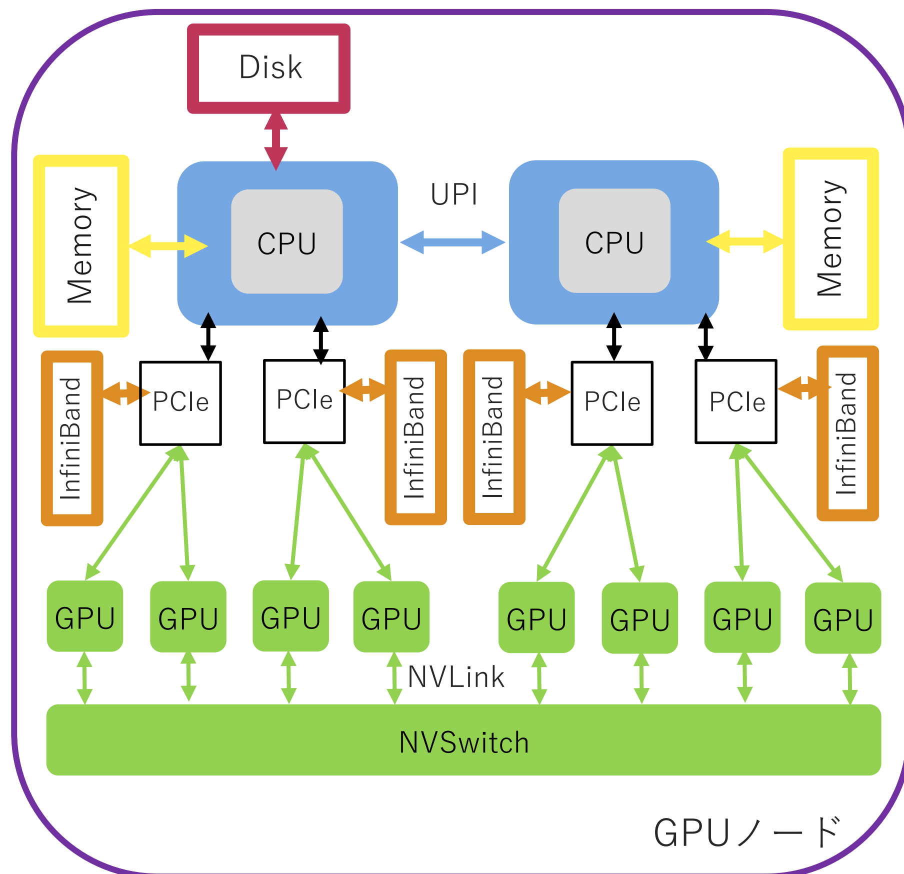
# 5.3. 高度なMPIの実行[アフィニティ] (1)

◆ プロセスのコアへの割り当て

各計算ノードは、以下のような構成をしています。計算ノードの構成に合わせて、MPIプログラムの各プロセスをコアへ割り当てることで、性能向上する場合があります。





© NEC Corporation 2026

# 5.3. 高度なMPIの実行[アフィニティ] (2)

◆ プロセス割り当ての確認

各MPIライブラリは、実行時に割り当てたCPU(VE)を出力することが可能です。
採取方法は以下のとおりです。

■ Intel MPIの場合

実行時に「I_MPI_DEBUG=4」を指定します。
標準出力に以下のように出力されます。

```
[0] MPI startup(): Rank    Pid      Node name  Pin cpu
[0] MPI startup(): 0       446143   cpu0101    0  ←MPIのRank#0プロセスは
[0] MPI startup(): 1       446144   cpu0101    1    cpu0101ノードの#0コアに割り当てられた
[0] MPI startup(): 2       446145   cpu0101    2
[0] MPI startup(): 3       446146   cpu0101    3
```


© NEC Corporation 2026

# 5.3. 高度なMPIの実行[アフィニティ] (3)

## ■ NECMPIの場合

mpirunのオプションに「-v」オプションを指定します。

```
mpirun -v -venode -np 3 ./a.out
```

標準出力に以下のように出力されます。

```
/opt/nec/ve/mpi/libexec/mpid: Creating 1 processes of './a.out' on VE 0 of local host vec0101
/opt/nec/ve/mpi/libexec/mpid: Creating 1 processes of './a.out' on VE 1 of local host vec0101
/opt/nec/ve/mpi/libexec/mpid: Creating 1 processes of './a.out' on VE 2 of local host vec0101
```

↑ MPIの1プロセスがvec0101ノードのVE#2に割り当てられた

## ■ OpenMPIの場合

mpirunのオプションに「-display-map」オプションを指定します。

```
mpirun ${NQSV_MPIOPTS} -display-map -np 2 ./a.out
```

標準出力に以下のように出力されます。

```
Data for node: gpu0101 Num slots: 76   Max slots: 0    Num procs: 2
        Process OMPI jobid: [123,1] App: 0 Process rank: 0 Bound: socket 0[core 0[hwt 0]]:[B/./././././././~./././././././.][./././././././././~./././././././.]
        Process OMPI jobid: [123,1] App: 0 Process rank: 1 Bound: socket 0[core 1[hwt 0]]:[./B/././././././~./././././././.][./././././././././~./././././././.]
```

↑ MPIの1プロセスがgpu0101ノードのsocket#0のcore#1に割り当てられた


© NEC Corporation 2026

# 5.3. 高度なMPIの実行[アフィニティ] (4)

◆ アフィニティ

各MPIライブラリは、実行時に割り当てるコアを制御することが可能です。
ここでは、ソケット分散、ソケット集中を例に説明します。

■ ソケット集中

- ソケットに詰めてプロセスを割り当てます。

  ソケット0のコア0→ソケット0のコア1→ソケット0のコア2→ソケット0のコア3→…
  →ソケット1のコア0→ソケット1のコア1→…

- 近接プロセスがソケットに集中し、同一メモリを使用するため、ソケット分散に比べてUPI間の転送時間が短縮されます。

■ ソケット分散

- ソケットに対して交互にプロセスを割り当てます。

  ソケット0のコア0→ソケット1のコア0→ソケット0のコア1→ソケット1のコア1
  →ソケット0のコア2→ソケット1のコア２→…

- コアを余らせて実行する場合、ソケット集中に比べ1コア当たりが使用するメモリバンド幅(メモリ転送速度)を確保できるため、メモリへのデータアクセスが高速になります。


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 5.3. 高度なMPIの実行[アフィニティ] (5)

## ◆ ソケット分散の指定方法

各MPIライブラリは、実行時に割り当てるコアを制御することが可能です。
ここでは、ソケット分散、ソケット集中を例に説明します。

### ■ ソケット分散

- Intel MPIの場合
  実行時に環境変数「I_MPI_PIN_PROCESSOR_LIST=“all:map=scatter”」を指定する。
- OpenMPIの場合
  mpirunのオプションに「--map-by socket」を指定する。

```
mpirun ${NQSV_MPIOPTS} -display-map -np 2 --map-by socket ./a.out
```

### ■ ソケット集中

- Intel MPIの場合
  実行時に環境変数「I_MPI_PIN_PROCESSOR_LIST="all:map=bunch"」を指定する。
- OpenMPIの場合
  mpirunのオプションに「--map-by core」を指定する。

```
mpirun ${NQSV_MPIOPTS} -display-map -np 2 --map-by core ./a.out
```


© NEC Corporation 2026

# 5.3. 高度なMPIの実行[アフィニティ] (6)

## ◆ Intel MPIにおけるノード内プロセス数のマニュアル指定

Intel MPIでは、-ppn、-rr、-perhostオプション(I_MPI_PERHOST 環境変数含む)を指定することで、ノード内のプロセスを限定することが可能です。ただし、-machinefile オプションを併用した場合、-machinefile オプションが優先されます。

SQUIDのジョブ管理システムでは、NQSV_MPIOPTS環境変数の中に、-machinefile オプションを含んでいるため、-ppn、-rr、-perhost オプションは無効となります。ノード内のプロセス数を限定する場合は、-l cpunum_job オプションに必要コア数を指定することで可能です。ただし、コアのピニング等を併用するなどより細かい指定をする場合は、PBS_NODEFILE環境変数を利用します。

以下は、PBS_NODEFILE環境変数を利用して、2ノードで128プロセス(ノードあたり64プロセス)を生成するジョブスクリプトのサンプルです。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS -b 2
#PBS -T intmpi

#------- Program execution -----------
module load BaseCPU
module load xxx/xxx

cd $PBS_O_WORKDIR
mpirun -hostfile ${PBS_NODEFILE} -np 128 -ppn 64 ./a.out
```


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 5.3. 高度なMPIの実行[アフィニティ] (7)

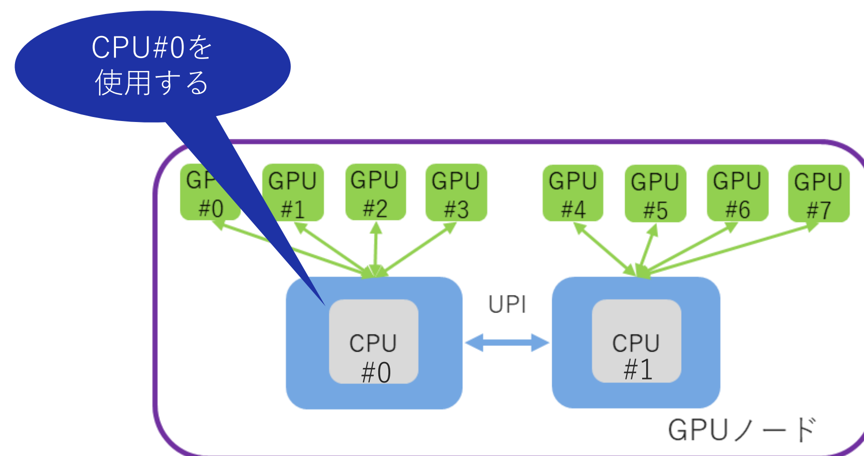
## ◆ GPUノードにおける使用CPU、GPUの指定方法

GPUノードにて使用するCPU、GPUは、NQSVでは制御しておりません。
コマンドや環境変数を利用することで、任意のCPU、GPUに割り当てることが可能です。

### ■ numactlコマンド

numactlコマンドを使用することでCPU番号を指定することができます。
以下は、CPU#0を使用することを指定しています。

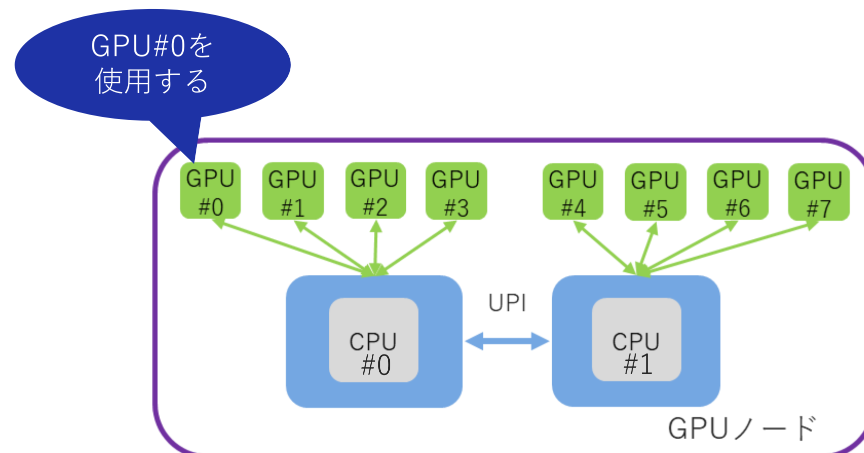
```
numactl -N 0 -l ./a.out
```




### ■ 環境変数CUDA_VISIBLE_DEVICES

使用するGPU番号を指定することができます。
以下は、GPU#0を使用することを指定しています。

```
export CUDA_VISIBLE_DEVICES=0
```





© NEC Corporation 2026

# 5.3. 高度なMPIの実行[アフィニティ] (8)

■ MPIランクごとに使用するGPUを指定する方法

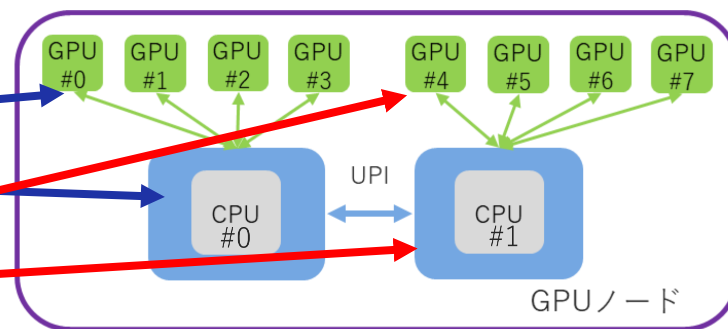
ラッパースクリプトを使用することで、MPIランクごとに使用するGPUを指定することができます。

以下は、1ノード内で2並列(1プロセス/ソケット)で実行し、それぞれのソケットにつながるGPUを使用すること想定した実行方法です。

```
WRAP=mpiwrap-single.sh
mpirun ${NQSV_MPIOPTS} -mca pml ucx -np 2 -npernode 2 $WRAP ${EXECDIR}/a.out
```

上記で使用した、ラッパースクリプトmpiwrap-single.shの内容は以下のとおりです。

```
#!/bin/bash
echo LANK=${OMPI_COMM_WORLD_LOCAL_RANK}
if [ ${OMPI_COMM_WORLD_LOCAL_RANK} -eq 0 ] ; then
        export CUDA_VISIBLE_DEVICES=0
rank#0  numactl -N 0 -l $@
else
        export CUDA_VISIBLE_DEVICES=4
rank#1  numactl -N 1 -l $@
fi
```





© NEC Corporation 2026

# 5.3. 高度なMPIの実行[アフィニティ] (9)

◆ GPUノードにおけるNVLinkの利用方法

NVLinkとは、GPU間で直接データ転送を行う通信プロトコルです。CPUを介さないため、高速な通信を行うことが可能です。

シングルノードにて、MPI APIでNVLinkを利用し、転送レイヤにてUCXを使用する ( CUDA 対応 UCX + GPUDirect )場合に、指定する環境変数は以下のとおりです。

```
UCX_TLS=sm,cuda_copy,gdr_copy,cuda_ipc
```

※UCX(Unified Communication X)とは
HPC 用の通信 API フレームワークです。
InfiniBand経由のMPI通信のために最適化されています。


© NEC Corporation 2026
\Orchestrating a brighter world NEC

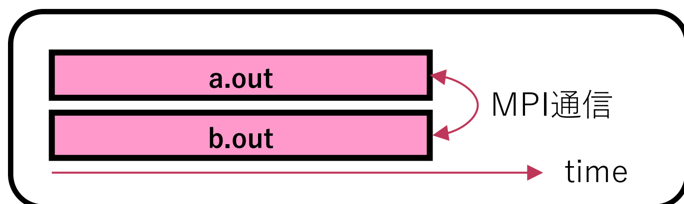
# 5.4. 高度なMPIの実行[MPMD] (1)

## ◆ MPMDジョブ

### ■ MPMDジョブとは

Multi Program Multi Dataの略です。

シミュレーション分野において、異なる2つ(もしくは3つ以上)の実行モジュールを同時に実行し、相互にMPI通信を行いながら連携実行するモデルをMPMDモデルと呼びます。




### ■ 利用可能なMPIライブラリ

以下3つのMPIライブラリは、いずれもMPMDに対応しています。

- Intel MPI
- NEC MPI
- OpenMPI


© NEC Corporation 2026

# 5.4. 高度なMPIの実行[MPMD] (2)

- MPMDジョブの実行方法

汎用CPUノードにてa.outをノード#1で76並列、b.outをノード#2,#3で152並列実行する場合

```
#!/bin/bash
#PBS -q SQUID
#PBS --group=G01234
#PBS -T intmpi
#PBS -b 3
   : （中略）

module load BaseCPU
export I_MPI_DEBUG=4
cd $PBS_O_WORKDIR

mpirun ${NQSV_MPIOPTS} -np 76  ./a.out : ¥
                       -np 152 ./b.out
```


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 5.5. コンテナの実行方法(1)

SQUIDシステムでは、コンテナを利用してジョブを実行することができます。コンテナを利用してジョブを実行する場合の、ジョブスクリプトについて説明します。

## ◆ コンテナ実行

コンテナの実行は、exec サブコマンドを指定して実施します。

(実行例)centos.sif コンテナイメージ内のhostname コマンドを実行する場合

```
$ singularity exec centos.sif hostname
```

コマンドの書式は以下のとおりです。

```
$ singularity exec <イメージファイル名> <コンテナ内の実行コマンド>
```

指定するコマンドは、**コンテナ内の実行コマンドとなっている点にご注意ください。** コマンドがパス指定なしであれば、コンテナ内のPATH環境変数で探索されたコマンドが実行されます。パス指定有りの場合も、絶対パスはコンテナ内のファイル構造に従います。


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 5.5. コンテナの実行方法(2)

- 環境変数

  **コンテナ外で定義した環境変数は、基本的にはコンテナ内にも引き継がれます。**
  ただし、build時などにコンテナ側で明示的に定義されている環境変数は、コンテナ側の定義に従います。

  コンテナ側で定義されている環境変数を上書きする場合には、--env オプションによる個別の指定や、--env-fileオプションによる一括の指定でコンテナ内に渡すことが可能です。

  - --env オプションによる個別指定

```
$ singularity exec --env MYVAR="My Value!" centos.sif myprog.exe
```

  - --env-file オプションによる一括指定

```
$ cat myenvfile
MYVAR="My Value!"
$ singularity exec --env-file myenvfile centos.sif myprog.exe
```

環境変数に関する詳細な内容は、下記の公式ドキュメントを参照ください。
https://sylabs.io/guides/3.7/user-guide/environment_and_metadata.html


© NEC Corporation 2026

# 5.5. コンテナの実行方法(3)

## ◆ ホストOSのマウント

コンテナ内からホストOSのファイルシステムのread/writeを行いたい場合には、ホストOSの特定のディレクトリをbindマウントすることで利用可能です。

### ■ ホストOSの標準マウントディレクトリ

下記のディレクトリは標準でマウントされており、コンテナ内でも同じパスで利用可能

- ホームディレクトリ：/sqfs/home/(利用者番号)
- テンポラリ領域：/tmp

(例)コンテナ内からホストOSのhomeディレクトリに置かれたプログラム
(/sqfs/home/(利用者番号)/a.out)を実行

```
$ singularity exec centos.sif ./a.out
```

上記例では、カレントディレクトリが、コンテナ内でhomeに移動しています。


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 5.5. コンテナの実行方法(4)

■ ホストOSの特定のディレクトリ

ホストOSの特定のディレクトリをマウントする場合には、--bindオプションを利用します。--bindオプションの書式は以下となります。

```
--bind <ホストOSのパス>:<コンテナ内のパス>:<モード>
```

コンテナ内のパス並びにモード(ro/rw)は省略可能です。省略した場合は、コンテナ内のパスは、ホストOSのパスと同じパスでread/write でマウントされます。

例(拡張領域上のディレクトリに置かれたプログラム(a.out)を実行)

```
$ cd /sqfs/work/(グループ名)/(利用者番号)
$ singularity exec --bind `pwd` centos.sif ./a.out
```

上記例では、コンテナ外での環境変数PWDが、コンテナ内へも引き継がれており、コンテナ内のカレントディレクトリが、拡張領域上のディレクトリとなっています。

ホストOSのマウントに関する詳細な内容は、下記の公式ドキュメントを参照ください。
https://sylabs.io/guides/3.7/user-guide/bind_paths_and_mounts.html


© NEC Corporation 2026

# 5.5. コンテナの実行方法(5)

## ◆ 汎用CPUノードでの実行方法

1ノード内でスレッド並列プログラム実行を想定したジョブスクリプトのサンプルです。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS -v OMP_NUM_THREADS=76         →  並列実行数を指定

#------- Program execution -----------
cd $PBS_O_WORKDIR
singularity exec --bind `pwd` image.sif ./a.out
                                   →  イメージファイルを指定して、singularity execコマンドを実行
```

利用時の注意点
ホストOSの特定のディレクトリをマウントする場合には、--bindオプションを利用してください。


© NEC Corporation 2026
\Orchestrating a brighter world NEC

# 5.5. コンテナの実行方法(6)

## ◆ GPUノードでの実行方法

1ノード内でスレッド並列プログラム実行を想定したジョブスクリプトのサンプルです。

```
#!/bin/bash
#------- qsub option -----------
#PBS -q SQUID
#PBS --group=G01234
#PBS -l elapstim_req=00:30:00
#PBS -l gpunum_job=8
#PBS -v OMP_NUM_THREADS=76  →  並列実行数を指定

#------- Program execution -----------
cd $PBS_O_WORKDIR
singularity exec --nv --bind `pwd` image.sif ./a.out
                                     →  イメージファイルを指定して、singularity execコマンドを実行します
                                        コンテナ内からGPGPUを利用可能とするため、--nvオプションを付加します
```

利用時の注意点
ホストOSの特定のディレクトリをマウントする場合には、--bindオプションを利用してください。


© NEC Corporation 2026

# 6. SX-Aurora TSUBASA 公開情報

# 6. SX-Aurora TSUBASA 公開情報

- NEC Aurora Forum
  - https://sxauroratsubasa.sakura.ne.jp/NEC_Aurora_Forum

    各種ドキュメント、コミュニティ

Create account Log in

Main page Discussion Read View source Search NEC Aurora Forum

## NEC Aurora Forum

Welcome to NEC Web Forum.

NEC aims to provide developers with an easy way to start working with the NEC SX-Aurora TSUBASA platform easily. We will use this website to publish interesting articles and experiences that help getting applications running at their best potential.

We want to help educate developers how to use the new platform but also learn from successful experiences of our users.

- Documentation [English|Japanese]
- Discussion Board
- FAQ

Top
Documentation
Discussion board
Tools
Special pages

クリック

Recently Updated

| Date | Contents |
|---|---|
| May 12th, 2023 | Updated Aurora SW : [Software Release/Updated] |

Page Discussion

## Documentation

[English|Japanese]

言語選択

Contents [hide]


1 Preliminary
2 VEOS
3 SDK
4 NEC MPI
5 NQSV
6 ScaTeFS


## NQSV

| Document Name | Revision |
|---|---|
| NQSV 利用の手引 導入編 | 12 |
| NQSV 利用の手引 管理編 | 16 |
| NQSV 利用の手引 操作編 | 18 |
| NQSV 利用の手引 リファレンス編 | 19 |
| NQSV 利用の手引 API編 | 8 |
| NQSV 利用の手引 アカウンティング・予算管理編 | 14 |
| NQSV 利用の手引 JobManipulator編 | 17 |
| NQSV 移行ガイド | - |
| NQSV リリースノート | - |
| NQSV Log Analysis Guide | - |

基本的な操作に関するマニュアルは以下の2点です

・NQSV 利用の手引き 操作編
　バッチリクエストの作成・投入・操作などの一連の操作に関するマニュアルです

・NQS利用の手引き リファレンス編
　バッチリクエスト操作関連のコマンドに関するリファレンスマニュアルです。


© NEC Corporation 2026

\Orchestrating a brighter world

NEC