2021.04.16 更新

Type IIサブシステム向けのプロセス・スレッド配置方法
主にGPU+OpenMPIを使う際のポイント：2021年度版

注意

- コンパイラとMPIの組み合わせ、実行するプログラムの形態によっては変なプロセス・スレッド配置になってしまうことがあります。本資料でもカバーしきれていない組み合わせパターンがあると思います。
- うまくいかない例などあればご質問・お問い合わせください。

- 問い合わせ先
  - 情報基盤センター准教授 大島聡史
  - ohshima@cc.nagoya-u.ac.jp

更新履歴

- 2020年10月19日
  - HPC-Xを使わない方が良い性能が得られるケースも多いようなので、OpenMPIの使い方を追加
    - どちらが良いのかは確認中
- 2020年10月22日
  - UCXのmodule loadの書き間違いを修正
- 2021年3月5日
  - Intel MPIを使う場合についてはクラウドシステム向けの資料を参照するよう誘導
- 2021年4月16日
  - 2021年度システム構成にあわせて更新
- （2021年4月19日、微修正）

Type IIサブシステム向けのプロセス・スレッド配置方法

- Type IIサブシステムでGPU+MPIを使う場合はOpenMPIを利用する
- 用途に合わせてGNUコンパイラ、Intelコンパイラ、CUDA Toolkit、HPC SDKと組み合わせることになる
- 基本的にはこのあたりの組み合わせになる（はず）
  - CUDA：GNUコンパイラ＋CUDA Toolkit＋OpenMPI
  - OpenACC：HPC SDK(+OpenMPI)

- Intelコンパイラ + Intel MPIを使う場合の基本的な使い方についてはクラウドシステム向けの資料を参照
  - Intelコンパイラ + Intel MPIからGPUを使う場合も、CUDA_VISIBLE_DEVICESを適切に設定すれば利用するGPUを指定することができる

- Type IIサブシステムは2CPU, 4GPU, 2NICにより構成されており、用途に応じて様々な使い方が考えられる　⇒　特に需要が高いと思われる例に絞って妥当な設定を紹介する

前提（本資料が想定する利用者層）

- 想定：複数のGPUが相互に通信を繰り返すようなMPIプログラムを実行したい場合
  - 単体GPUを利用する場合やGPU間通信をほとんど行わない場合は、MPIについてあまり色々考える必要はない（細かく調整しても性能に差が出ない）

- 1GPUのみ利用するプログラムを実行したい場合の注意
  - cx-shareリソースグループを使用し、プロセスやスレッドの割り当ては特に行わないでください
    - CPU10コア（0.5ソケット分）、1GPU、96GBのメモリが割り当てられる
    - cx-shareだけが利用ポイントの消費が1GPU分だけになる
      - cx-share以外のジョブクラス（リソースグループ）はノード単位で資源を確保・占有する。そのため、たとえ実際に使っているのが1GPUだとしても、4GPUが占有されてしまうため、利用ポイントも4GPU分消費される。
    - cx-singleは1ノードを占有するジョブであるため、常に4GPU分のポイントが消費される
      - singleはGPU数ではなくノード数のこと、他のサブシステムと概念を統一している
    - MPI通信は想定されない（10コア内に複数プロセスを立てて通信すること自体は問題ない）
      - 仮にMPIを利用するにしても、GPU間通信について考える必要がないため難しい問題は生じにくい

プロセス・スレッドの配置状況の確認方法と基本方針

- 確認方法
  - Intelコンパイラ・Intel MPIを使う場合は、KMP_AFFINITYのverboseオプションや環境変数I_MPI_DEBUGを使えば配置情報が出力される
  - OpenMPIでは-report-bindingsや-display-devel-mapオプションを使えば配置情報が出力される
  - 実行方法を問わず、プログラム実行中に/proc/{pid}/task/{tid}/statを読めば、各プロセス・スレッドが割り当てられたコアIDが確認できる
    - 本資料の最終ページ「参考：プロセスの配置情報を確認する方法（プログラム側から）」を参照
- プロセス・スレッドの配置の基本
  - mpirunやmpiexecでジョブ全体のプロセスの配置を調整できる
  - numactlでノード内のプロセス・スレッドの配置を調整できる
  - ⇒　まずmpirun/mpiexecで基本的な割り当てを指定し、必要に応じてnumactlも併用する
    - 場合によってはMPIで調整しやすい配置指定とnumactlで調整しやすい配置指定を使い分ける。ただしMPIレベルで各ノードへの配置プロセス数を間違えているとnumactlで修正できないため、基本的にMPIレベルで制御可能なものは優先的にMPIで制御する。
  - ※MPIプロセスとNICの対応付けの必要性を考慮すると、本当はMPI側で調整した方が良いのかも知れないが、その差は確認できていない（確認できた場合は資料を更新する）

OpenMPIの利用準備

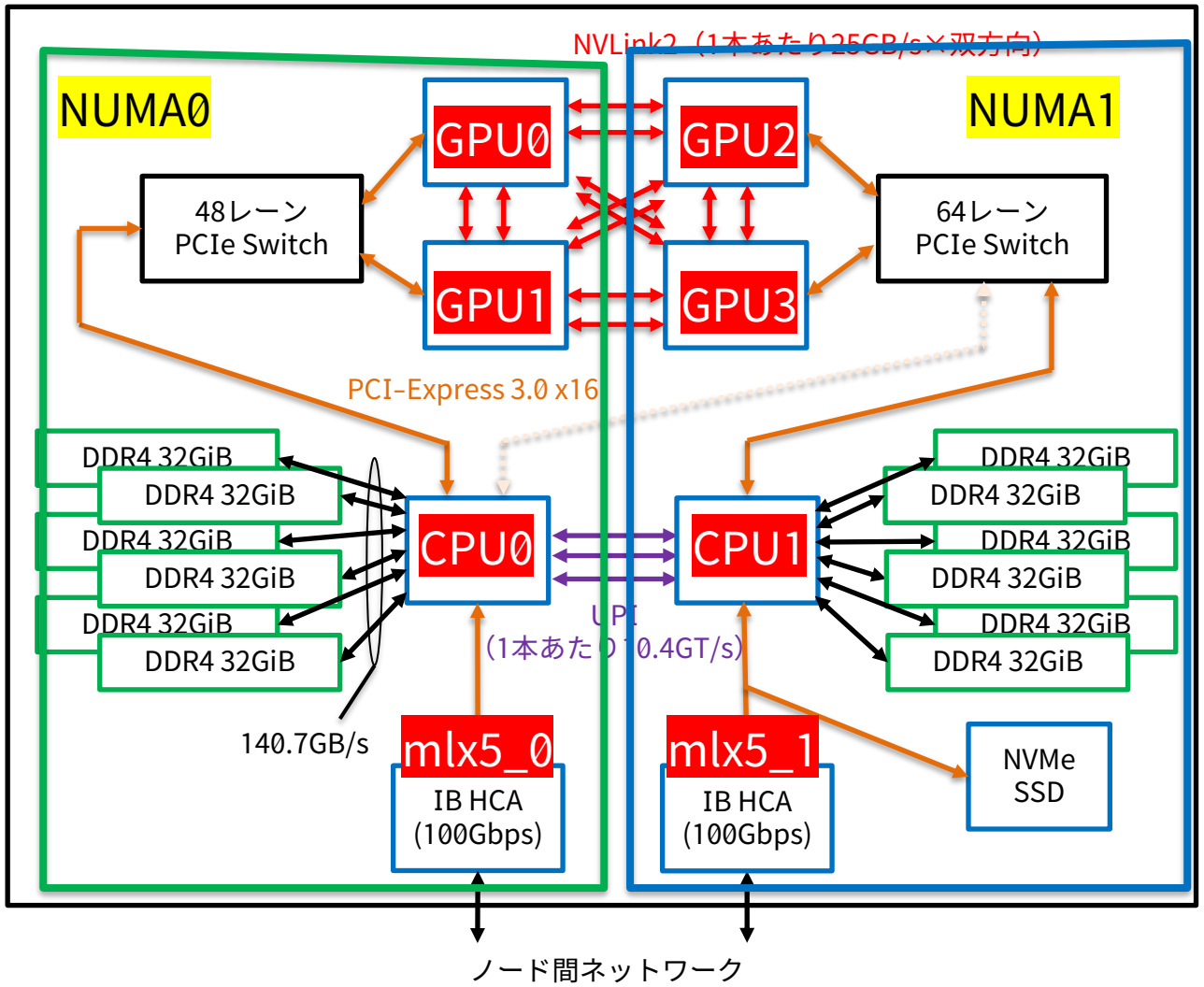
- GPUとMPIを使う場合はOpenMPIの利用を推奨
- 2021年4月現在、Type IIサブシステムにはいくつかのCUDAバージョンとOpenMPIバージョンが導入されている
  - 特に制約がない場合は最新のCUDA 11.2.1 + OpenMPI 4.0.5を推奨
  - 利用時には以下の通りmoduleコマンドを実行する

```
module load cuda/11.2.1
module load openmpi_cuda/4.0.5
module load ucx_cuda/1.10.0                    ←必須ではない
module load singularity/3.7.1                  ←必須ではない
```

- 1ノード実行の場合
  - MPIを使わない場合
  - MPIも使う場合
- 複数ノード実行の場合

Type IIサブシステムを1ノードだけ使う場合：基本的な考え方

- CPUやGPUの構成はnumactl -H, numactl -sやnvidia-smi topo -mコマンドで確認可能
- CPU, GPU, IBは右下図のような番号で認識されている
- CPU0とGPU0,1、CPU1とGPU2,3を組み合わせて使うことを基本とする
  - CPU-GPU間の通信をあまりしないのであれば、影響は小さいためあまり気にしなくても良い
- CPU1プロセス（1スレッド）で4GPUの面倒を見る場合は、CPUから見ると、どう設定しても、2つのGPUは距離が近く、残りの2つのGPUは距離が遠くなる



利用するGPUを指定する方法

- 環境変数CUDA_VISIBLE_DEVICESで指定する
  - 指定した順にGPUが見えるようになる
  - CUDAだけではなくOpenACCでも同様に有効
  - OpenMPやMPIを使う場合も常に効果がある環境変数、GPUを指定・制限する際は必須
    - 各プロセスに適切なCUDA_VISIBLE_DEVICESが見えている状態になるようにする

    - 以下、基本的な例

      - `export CUDA_VISIBLE_DEVICES=0,1`
        → プロセスに見えるGPUとその順番は GPU0, GPU1

      - `export CUDA_VISIBLE_DEVICES=3,2,1,0`
        → プロセスに見えるGPUとその順番は GPU3, GPU2, GPU1, GPU0

    - 適切なCUDA_VISIBLE_DEVICESが渡されるようなバッチジョブスクリプトを書くことを考える
    - 何も設定しなかった場合は0,1,2,3が指定されているのと同様

CPUとスレッドやプロセスの対応付け：OpenMPの基本

- OpenMP
  - numactl
    - -NでCPUソケット番号、-Cでコア番号
    - -l（--localallocでも同じ）はCPUコアに近いメモリを使ってくれる指定。多くの場合はこのオプションを指定するのが良い（実行時間が短くなる）。
  - OpenMPの仕様に定められた環境変数
    - OMP_PROC_BIND、OMP_PLACESなど
    - よく利用する設定の例
      - OMP_PROC_BIND=CLOSE　を指定するとスレッドが一方のCPUソケットからコンパクトに割り当てられる
      - OMP_PROC_BIND=SPREAD　を指定するとスレッドがソケットにまたがって均等に配置される
  - コンパイラごとの主要な環境変数
    - GNU：GOMP_CPU_AFFINITY
      - 使いたいコアの番号を指定する
    - Intel：KMP_AFFINITY
      - 割りあての方針を指定する
    - PGI：MP_BIND, MP_BLIST
      - 使いたいコアの番号を指定する
        - » はずなのだが、試してもうまく行かなかった

CPUとスレッドやプロセスの対応付け：MPIの基本

- MPI
  - 実装ごとに異なるが、引数や環境変数で指定が可能
  - 各ノードにプロセスを割り当てることはMPIにしかできない（numactlを使ってもノードを変えることはできない）が、ノード内のプロセス配置はMPIが行ったあとでnumactlにより変更することが可能
    → MPIでノード単位の配置を指定、ノード内の配置はnumactlで調整することが多い
  - もしかしたら通信性能に影響するかも知れないため、本資料ではMPIレベルで指定可能な部分はMPIで指定することにする
- OpenMPI
  - -may-byや-bind-to引数を使うと様々な割り当てが実現可能
    - 本資料で紹介
- IntelMPI
  - I_MPI_で始まる環境変数が利用できる
  - I_MPI_PIN_DOMAINとI_MPI_PIN_ORDERを使うと「よくありそうな割り当て方法」の多くが実現できる
    - クラウドシステム向けの資料を参照

1ノード内複数GPU実行：逐次実行版

- 逐次プログラム内でcudaSetDeviceやaccSetDeviceNumを使って使うGPUを指定する想定

- CPU0とGPU0,1を使うプログラム実行の例、および、CPU1とGPU2,3を使う実行の例

```
export CUDA_VISIBLE_DEVICES=0,1
numactl -N 0 -l ./a.out
```

```
export CUDA_VISIBLE_DEVICES=2,3
numactl -N 1 -l ./a.out
```

  - CUDA_VISIBLE_DEVICESで利用するGPU番号を指定
  - numactl -Nで利用するCPUソケットを指定
  - -l はlocalalloc、近くのメモリだけを使うという指定（この資料では常に付けているが、プログラムによっては付けない/別のオプションの方が良いこともある）
  - 後者（右例）の実行では、プログラム内でcudaSetDeviceやaccSetDeviceNumでデバイス番号0を指定すると実際はGPU2が、同様にデバイス番号1を指定すると実際はGPU3が使われる
  - コンパイラを問わず共通して使える基本的な手順

- なにも指定せずにプログラムを実行すれば4GPUが順番に見える状態になっている
  - 正しく動けば良いだけであれば、気にせず実行すれば良い

1ノード内複数GPU実行：OpenMPスレッド並列版

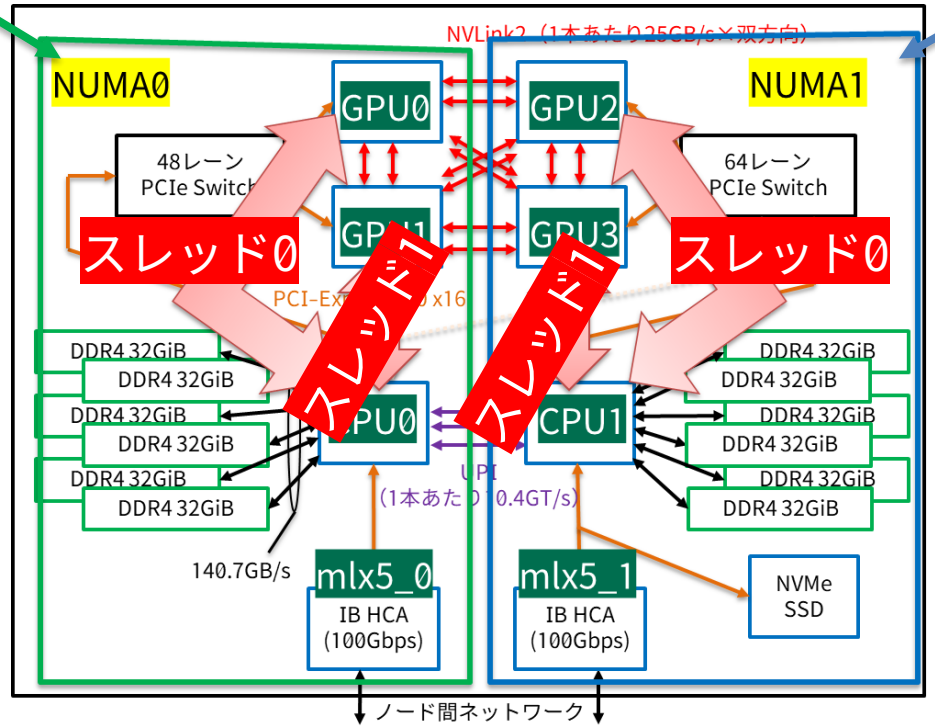
- numactlだけで簡単にできる指定の例
  - 片方のソケットで2スレッド実行、各スレッドはソケットから近い2GPUを1つずつ担当

```
export OMP_NUM_THREADS=2
export CUDA_VISIBLE_DEVICES=0,1
numactl -N 0 -l ./a.out
```

左半分が使われる

```
export OMP_NUM_THREADS=2
export CUDA_VISIBLE_DEVICES=2,3
numactl -N 1 -l ./a.out
```

右半分が使われる



1ノード内複数GPU実行：OpenMPスレッド並列版（つづき）

- numactlだけで簡単にできる指定の例
  - 片方のソケットで2スレッド実行、各スレッドはソケットから近い2GPUを1つずつ担当

```
export OMP_NUM_THREADS=2
export CUDA_VISIBLE_DEVICES=0,1
numactl -N 0 -l ./a.out
```

```
export OMP_NUM_THREADS=2
export CUDA_VISIBLE_DEVICES=2,3
numactl -N 1 -l ./a.out
```

    - スレッドは-Nで指定されたソケット側でのみ実行される、ソケットから近いGPUのみが利用対象として指定されている
      → 結果として意図した通りの担当になる

  - （とにかく4GPU使えれば良いだけであれば、OMP_NUM_THREADS=4だけ指定してプログラムを実行すれば十分。）

- numactlだけでは難しい指定の例
  - 4スレッド実行、各スレッドにソケットから近いGPUを担当させる
    - numactlによる指定はプロセス全体に対する指定になってしまうため、各スレッドの配置を細かに指定することは難しい
    - スレッド0,1はCPU0上で実行、スレッド2,3はCPU1上で実行、としたくても、numactlだけではこのスレッド配置が指定できない
    - -C 0,10,20,30 などと指定した場合、指定した4つのコアが使われるが、割り当たる順番が保証されない
      - 結果的に割り当たったコア番号を見てGPUを選ぶという方法もなくはないが、手間がかかる上に実行前に確定させられないのは不便

1ノード内複数GPU実行：OpenMPスレッド並列版（つづき）

- OpenMPの仕様に基づいて指定
  - OpenMPの仕様なので、全コンパイラで同様になることが期待される
    - 従わないコンパイラがある可能性は否定できないので注意（確認）すること
  - スレッド0,1はCPU0上で実行してGPU0と1を担当、スレッド2,3はCPU1上で実行してGPU2と3を担当　→　SPREADで実現可能

```
export OMP_NUM_THREADS=4
export CUDA_VISIBLE_DEVICES=0,1,2,3
export OMP_PROC_BIND=SPREAD
./a.out
```

  - spreadを指定するとscatter的ではなくbalanced的な割り当てになる
  - スレッド0,2はCPU0上で実行してGPU0と1を担当、スレッド1,3はCPU1上で実行してGPU2と3を担当　→　実現不可能
  - OMP_PLACESでコア番号を直接記述することにより実現可能

```
export OMP_NUM_THREADS=4
export CUDA_VISIBLE_DEVICES=0,2,1,3
export OMP_PLACES="{0},{20},{10},{30}"
./a.out
```

  - {} で囲まれたIDのコアに1スレッドずる割り当てられる
  - Type IIサブシステムは20コア×2のためこのような数になる

1ノード内複数GPU実行：OpenMPスレッド並列版（つづき）

- Affinity指定版：GNUコンパイラで作成したプログラムを実行する場合
  - GOMP_CPU_AFFINITYで各スレッドが利用するCPUコアを明示できる

```
export OMP_NUM_THREADS=4
export CUDA_VISIBLE_DEVICES=0,1,2,3
export GOMP_CPU_AFFINITY=0,10,20,30
./a.out
```

  - 起動するスレッドは順番に0,10,20,30番のコアに割り当てられる
    - 0-19はソケット0上、20-39はソケット1上のCPUコアに対応
    - あとはcudaSetDeviceやacc_set_device_numにOpenMPのスレッドIDを入れれば順番にGPU0, 1, 2, 3が割り当たり、近いCPUとGPUが組になる
  - 適切なソケット内のCPUコア番号を指定さえすれば、具体的な番号はなんでもよい
    - 例えば左図のGOMP_CPU_AFFINITYを2,4,22,33にしても近いCPU-GPUが組になる
  - 「scatter的」な割り当ても同様に行える

```
export OMP_NUM_THREADS=4
export CUDA_VISIBLE_DEVICES=0,2,1,3
export GOMP_CPU_AFFINITY=0,20,10,30
./a.out
```

    - スレッド0はソケット0上のCPUコア0に割り当てられてGPU0を担当、
      スレッド1はソケット1上のCPUコア20に割り当てられてGPU2を担当、
      スレッド2はソケット0上のCPUコア10に割り当てられてGPU1を担当、
      スレッド3はソケット1上のCPUコア30に割り当てられてGPU3を担当

1ノード内複数GPU実行：OpenMPスレッド並列版（つづき）

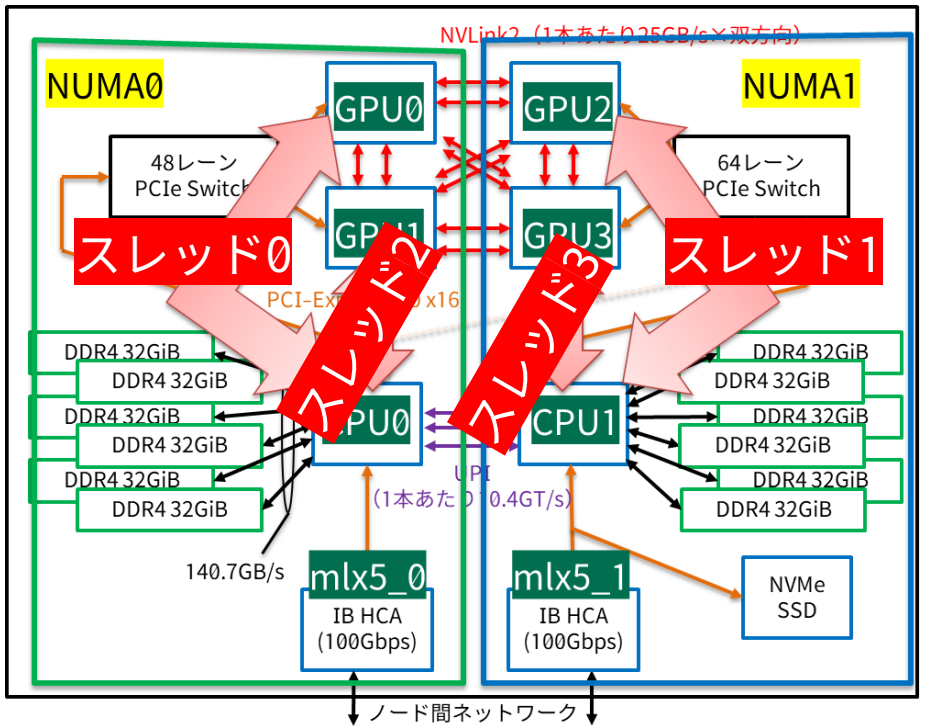
- Affinity指定版：Intelコンパイラで作成したプログラムを実行する場合
  - KMP_AFFINITYで割り当て方法を指定する
  - スレッド0,1はCPU0上で実行してGPU0と1を担当、スレッド2,3はCPU1上で実行してGPU2と3を担当　→　balancedに対応

```
export OMP_NUM_THREADS=4
export CUDA_VISIBLE_DEVICES=0,1,2,3
export KMP_AFFINITY=granularity=fine,balanced
./a.out
```

  - スレッド0,2はCPU0上で実行してGPU0と1を担当、スレッド1,3はCPU1上で実行してGPU2と3を担当　→　scatterに対応

```
export OMP_NUM_THREADS=4
export CUDA_VISIBLE_DEVICES=0,2,1,3
export KMP_AFFINITY=granularity=fine,scatter
./a.out
```

- Affinity指定版：PGIコンパイラで作成したプログラムを実行する場合
  - 現時点では不明
    - MP_BIND, MP_BLIST環境変数でコア指定ができるはずなのだが、実際に試してみたら思ったような挙動にならなかった



MPIを用いたプロセス配置の基本

- MPIのオプション（引数、環境変数）を使うことで様々な配置が可能
- ノード内のプロセス配置はnumactlでも可能なため、使い分けることも必要
  - MPI側で細かく配置してしまうとnumactl側で調整できなくなる点には注意（numactlで配置を変更できるのはMPI側で制限した範囲においてのみ）
  - ノードへの配置のみMPIで行い、あとはnumactlでやってしまうというのも一つの方法

MPIを用いたプロセス配置の基本：2段階のジョブスクリプト実行

- mpirun・mpiexecでノードに配置されることではじめて利用可能になる環境変数もあるため、2段階のバッチジョブスクリプトを使うことも多い
  - 典型的な例　→

run1.sh

```
mpirun -n 2 ./run2.sh ./a.out
```

run2.sh

```
#!/bin/bash
if [ ${OMPI_COMM_WORLD_RANK} -eq 0 ]; then
  CUDA_VISIBLE_DEVICES=0,1
  numactl -N 0 -l $1
else
  CUDA_VISIBLE_DEVICES=2,3
  numactl -N 1 -l $1
fi
```

$1は実行時に与えられた1つ目の引数（今回は./a.out）を意味する。

  - pjsub run1.shを実行するとrun2.shが2つ起動する
  - OpenMPIの1ノード実行の場合、特に何も指定しなければrun2.shはOpenMPIのプロセス配置により計算ノードのソケット0に2プロセス配置される。このとき各プロセスはOMPI_COMM_WORLD_RANKという通し番号を環境変数として与えられた状態で実行されるため、これを利用して各プロセスの実行するコマンドを調整している。
  - run2.sh内でenv関数を実行するとMPIによって与えられた環境変数も出力されるため参考になる

  - 実はこのmpirunの実行方法だと2プロセスともソケット0に配置されてしまいnumactlにはソケット0しか見えない状態になるのだが、このrun2.shを実行するとプロセス1はソケット1で実行されることが確認できている（気持ちが悪いが）

ジョブスクリプト例

run1.sh

```
#!/bin/bash -x
#PJM -L rscgrp=cx-small
#PJM -L node=1
#PJM -L elapse=00:05:00
#PJM -j
#PJM -S

module load cuda/11.2.1
module load openmpi_cuda/4.0.5
env
mpirun -n 2 -display-devel-map ./run2.sh ./a.out
```

run2.sh

```
#!/bin/bash
env
numactl -s
if [ ${OMPI_COMM_WORLD_RANK} -eq 0 ]; then
  numactl -N 0 -l $1
else
  numactl -N 1 -l $1
fi
```

- run1.shのenvとrun2.shの最初のenvとnumactlは参考情報出力用
- run2.shはmpirunによって起動されるため、run2.sh内のenvはrun1.sh内のenvと比べてMPI実行時に関する環境変数が追加されている
- run2.shにはchmod u+xなどで実行権限を与えておく必要がある

1ノード内複数GPU実行：MPI（OpenMPI）版、2プロセス×2GPU

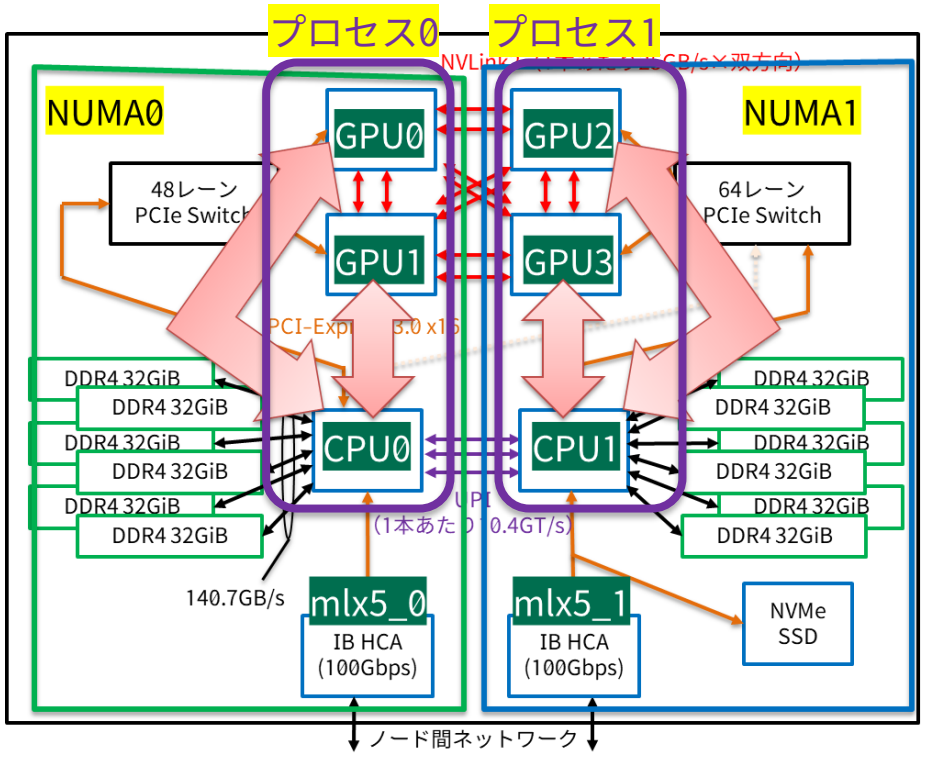
- 2プロセス実行、
  プロセス0がCPU0上でGPU0,1を、
  プロセス1がCPU1上でGPU2,3を担当

```
mpirun -n 2 -map-by socket -bind-to socket ./run2.sh ./a.out
```

run2.sh

```
#!/bin/bash
if [ ${OMPI_COMM_WORLD_RANK} -eq 0 ];
then
    export CUDA_VISIBLE_DEVICES=0,1
    numactl -l $1 2
else
    export CUDA_VISIBLE_DEVICES=2,3
    numactl -l $1 2
fi
```

-map-by socket -bind-to socketを指定すると、プロセス0（ランク0）はCPUソケット0、プロセス1（ランク1）はCPUソケット1に配置されるため、ランク情報で分岐させて適切なGPU番号を指定した。



同じ値の指定さえできていればrun2.shの書き方は問わない。
例えば、以下のように環境変数を計算して使っても良い。

```
#!/bin/bash
GPU1=$(( ${OMPI_COMM_WORLD_RANK} * 2 ))
GPU2=$(( ${GPU1} + 1 ))
export CUDA_VISIBLE_DEVICES=${GPU1},${GPU2}
numactl -l $1
```

コンパイル例、ジョブスクリプト例

mpi_cuda.cuにGPUカーネルが、mpi_cuda_main.cに通信を含むmain関数が書かれている想定

```
$ nvcc -O3 -arch=sm_70 -c mpi_cuda.cu
$ mpicc -O3 -march=native -o ./a.out mpi_cuda_main.c mpi_cuda.o -L${CUDA_HOME}/lib64 -lcudart
```

run2.sh　　※envとnumactl -sの行は参考情報出力用

```
#!/bin/bash
env > log_${PJM_JOBID}_${OMPI_COMM_WORLD_RANK}.txt
numactl -s >> log_${PJM_JOBID}_${OMPI_COMM_WORLD_RANK}.txt
if [ ${OMPI_COMM_WORLD_RANK} -eq 0 ]; then
    export CUDA_VISIBLE_DEVICES=0,1
    numactl -l $1
else
    export CUDA_VISIBLE_DEVICES=2,3
    numactl -l $1
fi
```

```
#!/bin/bash -x
#PJM -L rscgrp=cx-single
#PJM -L elapse=00:01:00
#PJM -j
#PJM -S

module load cuda/11.2.1
module load openmpi_cuda/4.0.5
mpirun -n 2 -report-bindings -display-devel-map -map-by socket -bind-to socket ./run2.sh ./a.out
```

実行例

```
Data for node: cx174   State: 3        Flags: 11
       Daemon: [[9739,0],0]    Daemon launched: True
       Num slots: 40   Slots in use: 2 Oversubscribed: FALSE
       Num slots allocated: 40 Max slots: 0
       Num procs: 2    Next node_rank: 2
       Data for proc: [[9739,1],0]
              Pid: 0  Local rank: 0   Node rank: 0    App rank: 0
              State: INITIALIZED      App_context: 0
              Locale:  [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././.]
              Binding: [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././.]
       Data for proc: [[9739,1],1]
              Pid: 0  Local rank: 1   Node rank: 1    App rank: 1
              State: INITIALIZED      App_context: 0
              Locale:  [./././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
              Binding: [./././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
```

ランク0はソケット0、ランク1はソケット1上に配置されていることを示している

←–display-devel-map で出力される情報

↓–report-bindings で出力される情報

```
[cx174:00061] MCW rank 0 bound to socket 0[core 0[hwt 0]], socket 0[core 1[hwt 0]], socket 0[core 2[hwt 0]], socket 0[core 3[hwt 0]], socket 0[core
4[hwt 0]], socket 0[core 5[hwt 0]], socket 0[core 6[hwt 0]], socket 0[core 7[hwt 0]], socket 0[core 8[hwt 0]], socket 0[core 9[hwt 0]], socket
0[core 10[hwt 0]], socket 0[core 11[hwt 0]], socket 0[core 12[hwt 0]], socket 0[core 13[hwt 0]], socket 0[core 14[hwt 0]], socket 0[core 15[hwt 0]],
socket 0[core 16[hwt 0]], socket 0[core 17[hwt 0]], socket 0[core 18[hwt 0]], socket 0[core 19[hwt 0]]:
[B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././.]
[cx174:00061] MCW rank 1 bound to socket 1[core 20[hwt 0]], socket 1[core 21[hwt 0]], socket 1[core 22[hwt 0]], socket 1[core 23[hwt 0]], socket
1[core 24[hwt 0]], socket 1[core 25[hwt 0]], socket 1[core 26[hwt 0]], socket 1[core 27[hwt 0]], socket 1[core 28[hwt 0]], socket 1[core 29[hwt 0]],
socket 1[core 30[hwt 0]], socket 1[core 31[hwt 0]], socket 1[core 32[hwt 0]], socket 1[core 33[hwt 0]], socket 1[core 34[hwt 0]], socket 1[core
35[hwt 0]], socket 1[core 36[hwt 0]], socket 1[core 37[hwt 0]], socket 1[core 38[hwt 0]], socket 1[core 39[hwt 0]]:
[./././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
```

1ノード内複数GPU実行：MPI（OpenMPI）版、4プロセス×1GPU

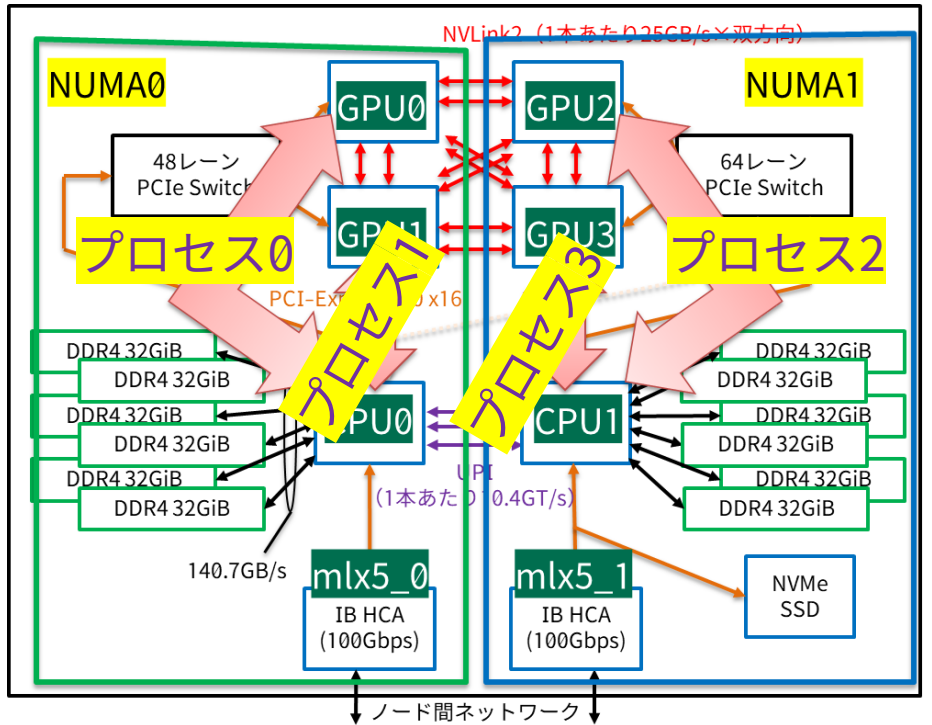
- 4プロセス実行、各MPIプロセスが1つずつGPUを担当
  - プロセス0,1をCPU0上に配置してGPU0,1を担当
  - プロセス2,3をCPU1上に配置してGPU2,3を担当

```
mpirun -n 4 -npersocket 2 -bind-to socket ./run2.sh ./a.out
```

run2.sh

```
#!/bin/bash
export CUDA_VISIBLE_DEVICES=${OMPI_COMM_WORLD_RANK}
numactl -l $1
```

-npersocket 2 -bind-to socketを指定すると、まずソケット0に2プロセス配置し、その後でソケット1に2プロセス配置する。MPIランク番号0,1のプロセスにはCPUソケット0側でGPU0,1を、MPIランク番号2,3のプロセスにはCPUソケット1側でGPU2,3を担当させるなら、このようにGPUデバイス番号とMPIランク番号を同じにするだけで良い。



ジョブスクリプト例

```
#!/bin/bash -x
#PJM -L rscgrp=cx-single
#PJM -L elapse=00:01:00
#PJM -j
#PJM -S

module load cuda/11.2.1
module load openmpi_cuda/4.0.5
mpirun -n 4 -report-bindings -display-devel-map -npersocket 2 -bind-to socket ./run2.sh
```

run2.sh

```
#!/bin/bash
env > log_${PJM_JOBID}_${OMPI_COMM_WORLD_RANK}.txt
numactl -s >> log_${PJM_JOBID}_${OMPI_COMM_WORLD_RANK}.txt
export CUDA_VISIBLE_DEVICES=${OMPI_COMM_WORLD_RANK}
numactl -l $1
```

プロセス0, 1から見えるnumactl -s情報

```
policy: default
preferred node: current
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
cpubind: 0
nodebind: 0
membind: 0 1
```

プロセス2, 3から見えるnumactl -s情報

```
policy: default
preferred node: current
physcpubind: 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
cpubind: 1
nodebind: 1
membind: 0 1
```

実行例

```
Data for node: cx174   State: 3        Flags: 11
       Daemon: [[9739,0],0]    Daemon launched: True
       Num slots: 40   Slots in use: 4 Oversubscribed: FALSE
       Num slots allocated: 40 Max slots: 0
       Num procs: 4    Next node_rank: 4
       Data for proc: [[9739,1],0]
              Pid: 0  Local rank: 0   Node rank: 0    App rank: 0
              State: INITIALIZED      App_context: 0
              Locale:  [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././.]
              Binding: [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././.]
       Data for proc: [[9739,1],1]
              Pid: 0  Local rank: 1   Node rank: 1    App rank: 1
              State: INITIALIZED      App_context: 0
              Locale:  [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././.]
              Binding: [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././.]
       Data for proc: [[9739,1],2]
              Pid: 0  Local rank: 2   Node rank: 2    App rank: 2
              State: INITIALIZED      App_context: 0
              Locale:  [./././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
              Binding: [./././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
       Data for proc: [[9739,1],3]
              Pid: 0  Local rank: 3   Node rank: 3    App rank: 3
              State: INITIALIZED      App_context: 0
              Locale:  [./././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
              Binding: [./././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
```

プロセス0,1はCPUソケット0上に配置

プロセス2,3はCPUソケット1上に配置

1ノード内複数GPU実行：MPI（OpenMPI）版、4プロセス×1GPU

numactl側で配置を制御する版

- プロセス数の指定さえ正しければnumactlでどうにかできる

- 2プロセス実行、以下略

```
mpirun -n 2 ./run2.sh ./a.out
```

run2.sh

```
#!/bin/bash
if [ ${OMPI_COMM_WORLD_RANK} -eq 0 ];
then
  CUDA_VISIBLE_DEVICES=0,1
  numactl -N 0 -l $1
else
  CUDA_VISIBLE_DEVICES=2,3
  numactl -N 1 -l $1
fi
```

OpenMPIのランク情報を使って配置に必要な情報を選択。-NでCPUソケット番号を指定。

- 4プロセス実行、以下略

```
mpirun -n 4 ./run2.sh ./a.out
```

run2.sh

```
#!/bin/bash
case ${OMPI_COMM_WORLD_RANK} in
0 )
  export CUDA_VISIBLE_DEVICES=0
  numactl -N 0 -l $1;;
1 )
  export CUDA_VISIBLE_DEVICES=1
  numactl -N 0 -l $1;;
2 )
  export CUDA_VISIBLE_DEVICES=2
  numactl -N 1 -l $1;;
3 )
  export CUDA_VISIBLE_DEVICES=3
  numactl -N 1 -l $1;;
esac
```

run2.sh 別の書き方の一例

```
#!/bin/bash
GID=${OMPI_COMM_WORLD_RANK}
CID=$(( ${GID} / 2 ))
export CUDA_VISIBLE_DEVICES=${GID}
numactl -N ${CID} -l $1
```

もちろん、2プロセス実行の例のようにif文で書いても構わない。

ジョブスクリプト例

```
#!/bin/bash -x
#PJM -L rscgrp=cx-single
#PJM -L elapse=00:01:00
#PJM -j
#PJM -S

module load cuda/11.2.1
module load openmpi_cuda/4.0.5

mpirun -n 4 -report-bindings -display-devel-map ./run2.sh ./a.out
```

```
#!/bin/bash
env > log_${PJM_JOBID}_${OMPI_COMM_WORLD_RANK}.txt
numactl -s >> log_${PJM_JOBID}_${OMPI_COMM_WORLD_RANK}.txt
case ${OMPI_COMM_WORLD_RANK} in
0 )
  export CUDA_VISIBLE_DEVICES=0
  numactl -N 0 -l $1;;
1 )
  export CUDA_VISIBLE_DEVICES=1
  numactl -N 0 -l $1;;
2 )
  export CUDA_VISIBLE_DEVICES=2
  numactl -N 1 -l $1;;
3 )
  export CUDA_VISIBLE_DEVICES=3
  numactl -N 1 -l $1;;
esac
```

実行例

```
Data for node: cx174   State: 3        Flags: 11
       Daemon: [[9739,0],0]    Daemon launched: True
       Num slots: 40   Slots in use: 4 Oversubscribed: FALSE
       Num slots allocated: 40 Max slots: 0
       Num procs: 4    Next node_rank: 4
       Data for proc: [[9739,1],0]
               Pid: 0  Local rank: 0   Node rank: 0    App rank: 0
               State: INITIALIZED      App_context: 0
               Locale:  [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][././././././././././././././././././.]
               Binding: [B/./././././././././././././././././.][././././././././././././././././././.]
       Data for proc: [[9739,1],1]
               Pid: 0  Local rank: 1   Node rank: 1    App rank: 1
               State: INITIALIZED      App_context: 0
               Locale:  [././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
               Binding: [././././././././././././././././././.][B/./././././././././././././././././.]
       Data for proc: [[9739,1],2]
               Pid: 0  Local rank: 2   Node rank: 2    App rank: 2
               State: INITIALIZED      App_context: 0
               Locale:  [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][././././././././././././././././././.]
               Binding: [./B/././././././././././././././././.][././././././././././././././././././.]
       Data for proc: [[9739,1],3]
               Pid: 0  Local rank: 3   Node rank: 3    App rank: 3
               State: INITIALIZED      App_context: 0
               Locale:  [././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
               Binding: [././././././././././././././././././.][./B/././././././././././././././././.]
```

←–display-devel-map で出力された情報

MPIレベルではランク（プロセス）0,2がソケット0に、
ランク（プロセス）1,3がソケット1に配置されたが、
numactlの設定により実際にプロセスが実行されたコアの番号は
0,3,34,23の順番になった。
（コア番号0–19がソケット0，コア番号20–39がソケット1。）

↓プログラム中で配置を確認した結果

```
hostname=cx174 rank=0 thread-id=75 omp-tid= 0 core-id=0
hostname=cx174 rank=1 thread-id=77 omp-tid= 0 core-id=3
hostname=cx174 rank=2 thread-id=78 omp-tid= 0 core-id=34
hostname=cx174 rank=3 thread-id=80 omp-tid= 0 core-id=23
```

1ノード内複数GPU実行：MPI（Intel MPI）版

- Intel MPIではPMI_RANKにランク番号が設定されるため、これでOpenMPI版のOMPI_COMM_WORLD_RANKを置き換えれば良い（総ランク数はPMI_SIZE）
  - その他のプロセス配置情報はクラウドシステム向け資料を参照
- 2プロセス、プロセス0がCPU0上でGPU0,1を、プロセス1がCPU1上でGPU2,3を担当

```
export I_MPI_PIN_DOMAIN=1
export I_MPI_PIN_ORDER=scatter
mpiexec -n 2 ./run2.sh ./a.out
```

run2.sh

```
#!/bin/bash
GPU1=$(( ${PMI_RANK} * 2 ))
GPU2=$(( ${GPU1} + 1 ))
export CUDA_VISIBLE_DEVICES=${GPU1},${GPU2}
numactl -l $1
```

- 4プロセス実行、各MPIプロセスが1つずつGPUを担当
  - プロセス0,1＝CPU0上、GPU0,1を担当
  - プロセス2,3＝CPU1上、GPU2,3を担当

```
export I_MPI_PIN_DOMAIN=10
export I_MPI_PIN_ORDER=compact
mpiexec -n 4 ./run2.sh ./a.out
```

run2.sh

```
#!/bin/bash
export CUDA_VISIBLE_DEVICES=${PMI_RANK}
numactl -l $1
```

- とにかくプロセスとソケットの配置が正しければ良いので、これら以外にもうまくいく設定は色々考えられる

実行例

- 2プロセス×2GPU
  - I_MPI_DEBUG=5で表示される情報

```
[0] MPI startup(): Rank    Pid      Node name  Pin cpu
[0] MPI startup(): 0       99       cx174      {0}
[0] MPI startup(): 1       100      cx174      {20}
```

ランク（プロセス）0はコア0に、ランク（プロセス）1はコア20に配置された。

  - コア番号

```
hostname=cx174 rank=0 thread-id= 99 omp-tid= 0 core-id=0
hostname=cx174 rank=1 thread-id=100 omp-tid= 0 core-id=20
```

- 4プロセス×1GPU
  - I_MPI_DEBUG=5で表示される情報

```
[0] MPI startup(): Rank    Pid      Node name  Pin cpu
[0] MPI startup(): 0       114      cx174      {0,1,2,3,4,5,6,7,8,9}
[0] MPI startup(): 1       115      cx174      {10,11,12,13,14,15,16,17,18,19}
[0] MPI startup(): 2       111      cx174      {20,21,22,23,24,25,26,27,28,29}
[0] MPI startup(): 3       116      cx174      {30,31,32,33,34,35,36,37,38,39}
```

  - コア番号

```
hostname=cx174 rank=0 thread-id=114 omp-tid= 0 core-id=0
hostname=cx174 rank=1 thread-id=115 omp-tid= 0 core-id=10
hostname=cx174 rank=2 thread-id=111 omp-tid= 0 core-id=20
hostname=cx174 rank=3 thread-id=116 omp-tid= 0 core-id=33
```

ランク（プロセス）0はコア0-9に、ランク（プロセス）1はコア10-29に、以下省略。
コアが1つに絞られていないが、望んだ実行形態にはなっている。

1ノード内複数GPU実行：MPI（Intel MPI）版
numactl側で配置を制御する版

- OpenMPI版OMPI_COMM_WORLD_RANKをPMI_RANKで置き換えただけ版
  - MPIによる差が小さいという点で使いやすい

- 2プロセス、プロセス0がCPU0上でGPU0,1を、プロセス1がCPU1上でGPU2,3を担当

```
mpiexec -n 2 ./run2.sh ./a.out
```

run2.sh

```
#!/bin/bash
if [ ${PMI_RANK} -eq 0 ]; then
  CUDA_VISIBLE_DEVICES=0,1
  numactl -N 0 -l $1
else
  CUDA_VISIBLE_DEVICES=2,3
  numactl -N 1 -l $1
fi
```

- 4プロセス実行、各MPIプロセスが1つずつGPUを担当
  - プロセス0,1＝CPU0上、GPU0,1を担当
  - プロセス2,3＝CPU1上、GPU2,3を担当

```
mpiexec -n 4 ./run2.sh ./a.out
```

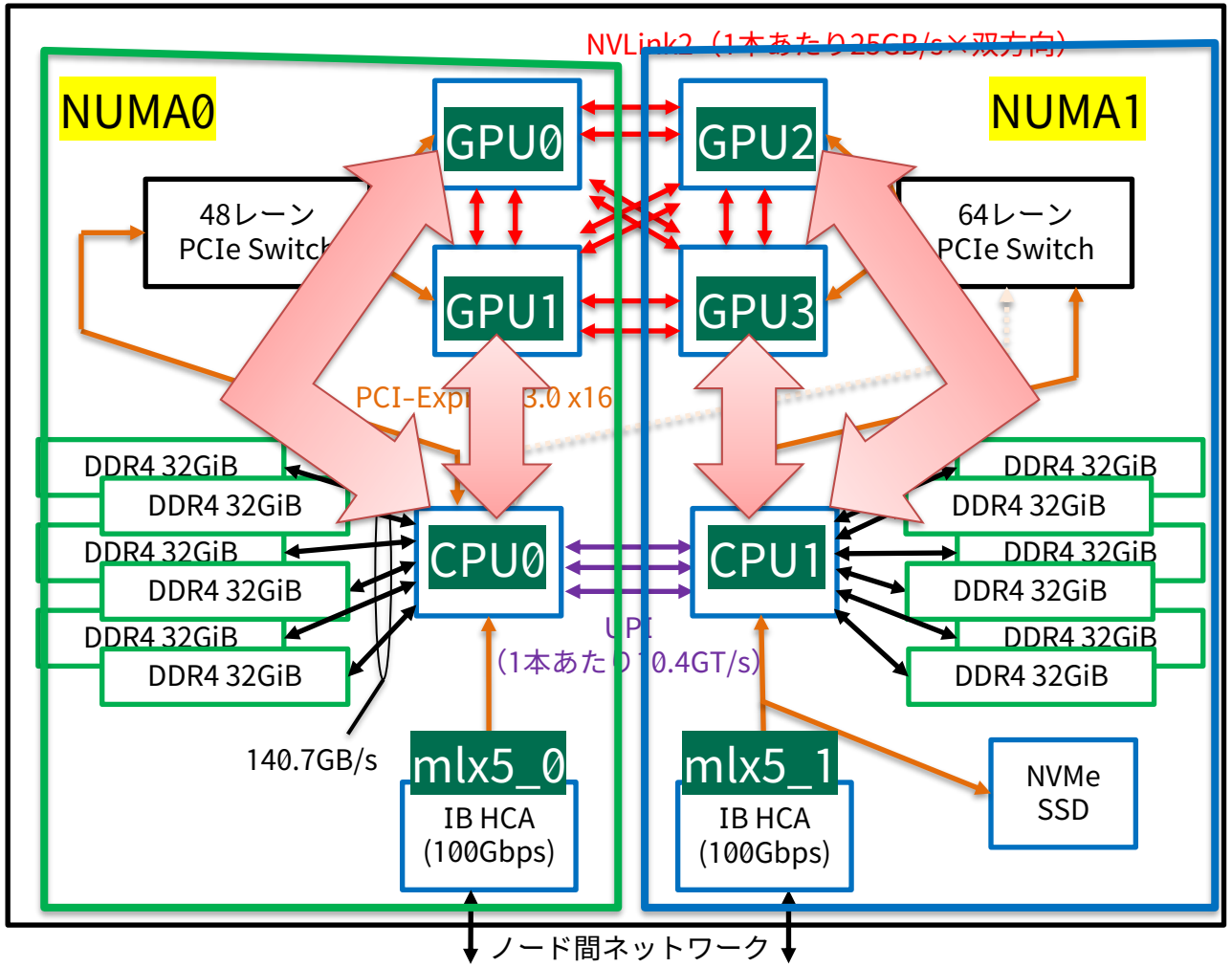
run2.sh

```
#!/bin/bash
case ${PMI_RANK} in
0 )
  export CUDA_VISIBLE_DEVICES=0
  numactl -N 0 -l $1;;
1 )
  export CUDA_VISIBLE_DEVICES=1
  numactl -N 0 -l $1;;
以下省略
```

- 1ノード実行の場合
  - MPIを使わない場合
  - MPIも使う場合
- 複数ノード実行の場合

複数ノード実行（MPI）の場合

- 多く利用されるのは「1MPIプロセスあたり1GPU」×nプロセスという実行形態だと思われるため、この形態について説明する
  - ノード内のCPU-GPUの担当関係は右図の赤い矢印のとおり
  - 複数ノード実行の場合は
    MPIランク0-3: ノード0
    MPIランク4-7: ノード1
    のようにランクが近いものを同一ノードに配置するcompactな配置が基本と思われる
  - もちろんscatterな配置も可能
  - （CPUとIBがノードあたり2つあるため
    1MPIプロセスあたり2GPU、
    もしくは、ノード単位で処理をまとめて
    1MPIプロセスあたり4GPU、
    というパターンも需要はあるのだろうか？）



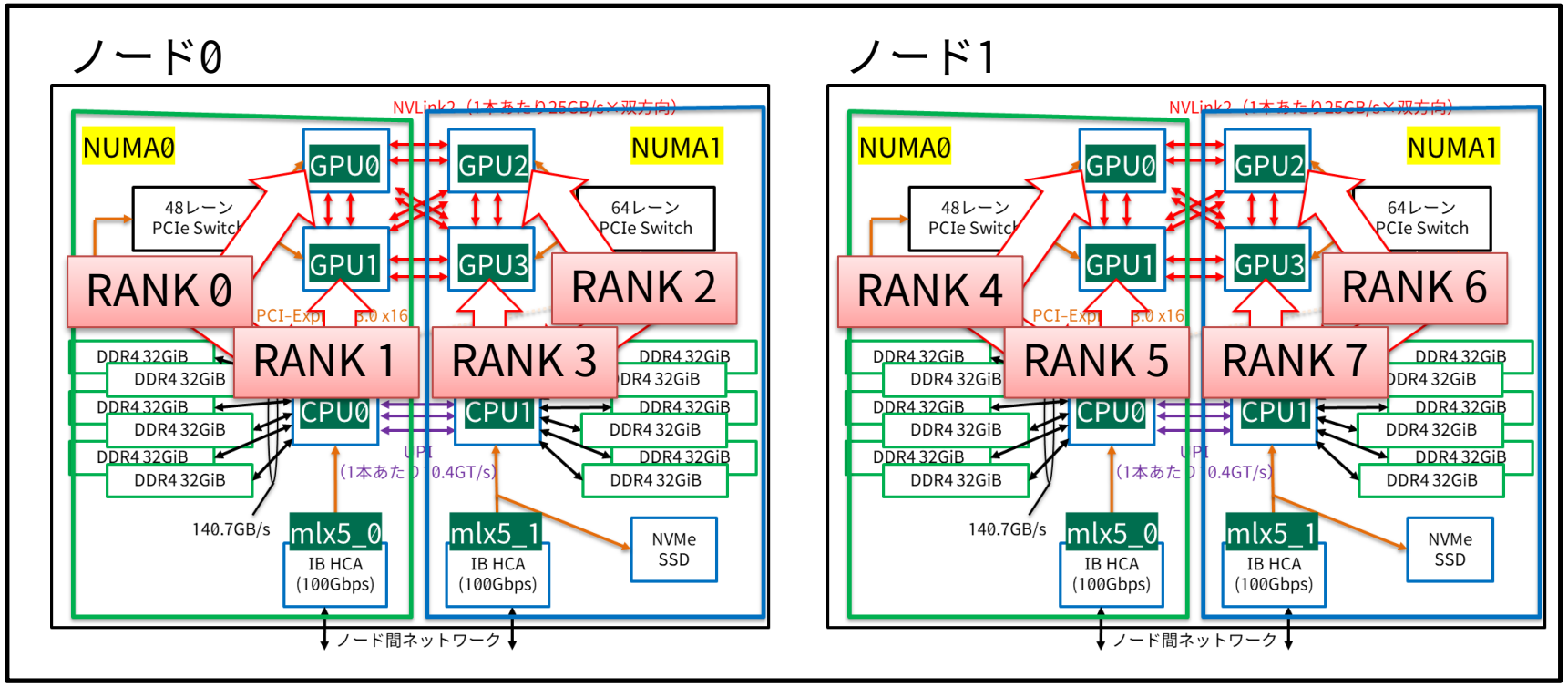
※他の実行形態を考える余地について

- ノード間通信を減らすという意味では、1ノードあたり4MPIプロセスではなく、1ノードあたり2MPIプロセスか1MPIプロセスに減らした方が全体として高性能になる可能性がある
- ただし、そのためにはノード内でいったんデータを集約してからノード間通信を行うなどの対応が必要になり、プログラムの作りが複雑になったり通信回数が増えたりする可能性がある
- どのような作りにするのが最良であるかはプログラムによる
- そのため、ここでは単純で多くの環境に適用可能な
  「1MPIプロセスあたり1GPU」×ｎプロセス
  のみに絞って解説する

OpenMPI、複数ノード、フラットMPI、利用例1

- compactな配置
  - ノード内の全GPU分のMPIランクを配置したら次のノードへ

配置イメージ→



- -map-by ppr:2:socketを指定すると、ソケットあたり2プロセス配置したら次のソケットへ、という挙動になる

```
$ mpirun -n 8 -machinefile $PJM_O_NODEINF -report-bindings -display-devel-map ¥
-map-by ppr:2:socket ./run2.sh ./a.out
```

run2.sh

```
#!/bin/bash
export CUDA_VISIBLE_DEVICES=${OMPI_COMM_WORLD_LOCAL_RANK}
numactl -l $1
```

※行末の￥は継続行、つまり本来は一行で書かれるべきものを改行する場合に使う

- OMPI_COMM_WORLD_LOCAL_RANKはその名の通りローカル、つまりノードごとに0から始まるランク番号。

スクリプトと実行例

```
#!/bin/bash -x
#PJM -L rscgrp=cx-small
#PJM -L node=2
#PJM -L elapse=00:10:00
#PJM -j
#PJM -S

module load cuda/11.2.1
module load openmpi_cuda/4.0.5
mpirun -n 8 -machinefile $PJM_O_NODEINF -report-bindings ¥
-map-by ppr:2:socket ./run2.sh ./a.out
```

run2.sh

```
#!/bin/bash
export CUDA_VISIBLE_DEVICES=${OMPI_COMM_WORLD_LOCAL_RANK}
numactl -l $1
```

コア配置情報

```
hostname=cx174 rank=0 thread-id= 78 omp-tid= 0 core-id=0
hostname=cx174 rank=1 thread-id= 77 omp-tid= 0 core-id=1
hostname=cx174 rank=2 thread-id= 79 omp-tid= 0 core-id=20
hostname=cx174 rank=3 thread-id= 81 omp-tid= 0 core-id=21
hostname=cx175 rank=4 thread-id= 24 omp-tid= 0 core-id=0
hostname=cx175 rank=5 thread-id= 23 omp-tid= 0 core-id=1
hostname=cx175 rank=6 thread-id= 25 omp-tid= 0 core-id=20
hostname=cx175 rank=7 thread-id= 27 omp-tid= 0 core-id=21
```

※配置ルールの都合上、rankでソートすると必ずcore-idが20未満, 20未満, 20以上, 20以上, 20未満, 20未満, 20以上, 20以上という順番になるが、具体的な数字は固定ではない

-report-bindingsの出力

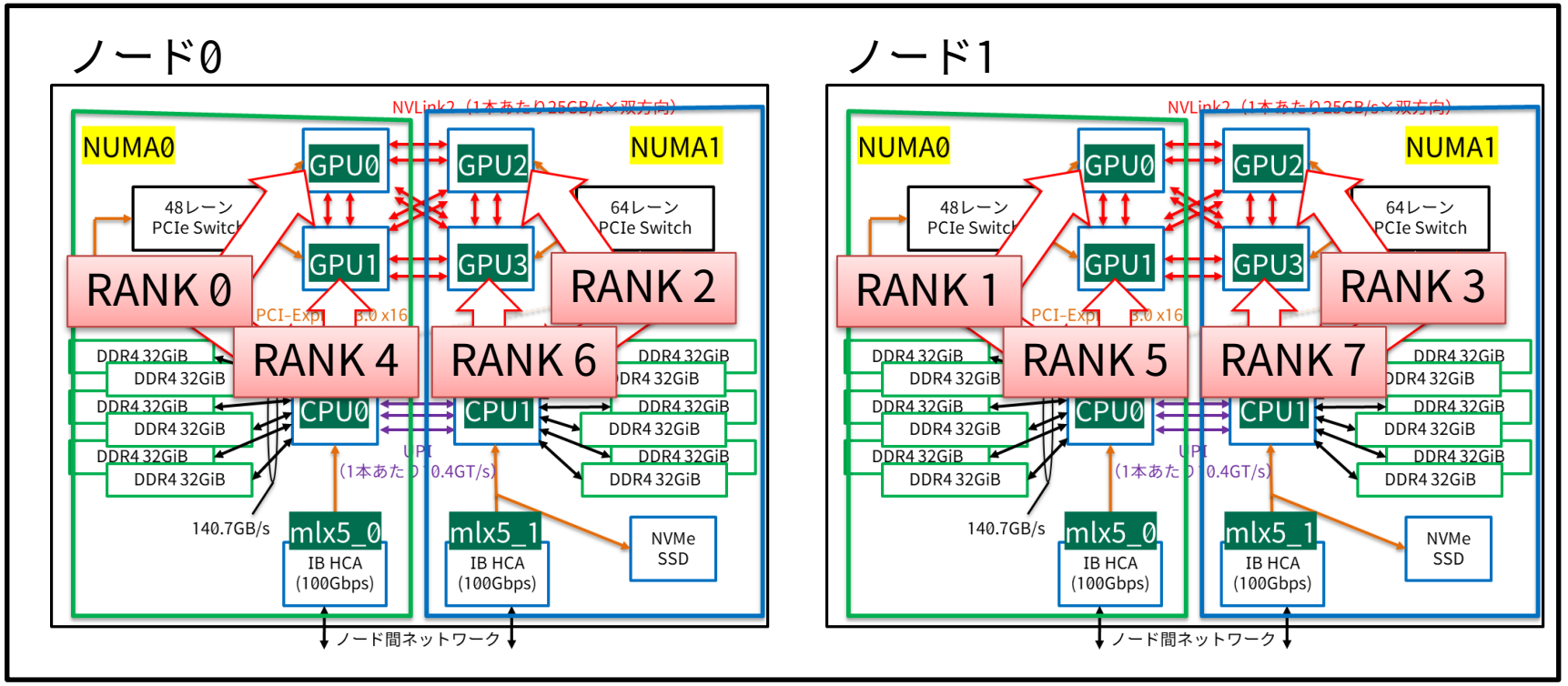
```
[cx174:00061] MCW rank 0 bound to socket 0[core 0[hwt 0]]: [B/./././././././././././././././.][./././././././././././././.]
[cx174:00061] MCW rank 1 bound to socket 0[core 1[hwt 0]]: [./B/./././././././././././././.][./././././././././././././.]
[cx174:00061] MCW rank 2 bound to socket 1[core 20[hwt 0]]: [./././././././././././././.][B/./././././././././././././.]
[cx174:00061] MCW rank 3 bound to socket 1[core 21[hwt 0]]: [./././././././././././././.][./B/./././././././././././.]
[cx175:00009] MCW rank 4 bound to socket 0[core 0[hwt 0]]: [B/./././././././././././././.][./././././././././././././.]
[cx175:00009] MCW rank 5 bound to socket 0[core 1[hwt 0]]: [./B/./././././././././././././.][./././././././././././././.]
[cx175:00009] MCW rank 6 bound to socket 1[core 20[hwt 0]]: [./././././././././././././.][B/./././././././././././././.]
[cx175:00009] MCW rank 7 bound to socket 1[core 21[hwt 0]]: [./././././././././././././.][./B/./././././././././././.]
```

想定したとおりのコア配置（ソケット配置順序）になっていることが確認できる

OpenMPI、複数ノード、フラットMPI、利用例2

- scatterな配置
  - ノード内のGPUに1つMPIランクを配置したら次のノードへ
  - なるべく分散させる

配置イメージ→




```
$ mpirun -n 8 -machinefile $PJM_O_NODEINF -report-bindings ¥
-map-by node -bind-to socket ./run2.sh ./a.out
```

run2.sh

```
#!/bin/bash
export CUDA_VISIBLE_DEVICES=${OMPI_COMM_WORLD_LOCAL_RANK}
numactl -l $1
```

- -map-by node -bind-to socketにするとscatterなノード割り当てになる（1ノードに1プロセス置いたら次のノードへ、となる）

スクリプトと実行例

```
#!/bin/bash -x
#PJM -L rscgrp=cx-small
#PJM -L node=2
#PJM -L elapse=00:10:00
#PJM -j
#PJM -S

module load cuda/11.2.1
module load openmpi_cuda/4.0.5
mpirun -n 8 -machinefile $PJM_O_NODEINF -report-bindings ¥
-map-by node -bind-to socket ./run2.sh ./a.out
```

run2.sh

```
#!/bin/bash
export CUDA_VISIBLE_DEVICES=${OMPI_COMM_WORLD_LOCAL_RANK}
numactl -l $1
```

コア配置情報

```
hostname=cx174 rank=0 thread-id=119 omp-tid= 0 core-id=5
hostname=cx175 rank=1 thread-id= 64 omp-tid= 0 core-id=0
hostname=cx174 rank=2 thread-id=123 omp-tid= 0 core-id=20
hostname=cx175 rank=3 thread-id= 69 omp-tid= 0 core-id=20
hostname=cx174 rank=4 thread-id=125 omp-tid= 0 core-id=0
hostname=cx175 rank=5 thread-id= 70 omp-tid= 0 core-id=1
hostname=cx174 rank=6 thread-id=126 omp-tid= 0 core-id=34
hostname=cx175 rank=7 thread-id= 72 omp-tid= 0 core-id=37
```

※配置ルールの都合上、rankでソートすると必ずcore-idが20未満, 20未満, 20以上, 20以上, 20未満, 20未満, 20以上, 20以上という順番になるが、具体的な数字は固定ではない

-report-bindingsの出力

```
[cx174:00061] MCW rank 0 bound to 中略 : [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././.]
[cx175:00010] MCW rank 1 bound to 中略 : [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././.]
[cx174:00061] MCW rank 2 bound to 中略 : [./././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
[cx175:00010] MCW rank 3 bound to 中略 : [./././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
[cx174:00061] MCW rank 4 bound to 中略 : [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././.]
[cx175:00010] MCW rank 5 bound to 中略 : [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././.]
[cx174:00061] MCW rank 6 bound to 中略 : [./././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
[cx175:00010] MCW rank 7 bound to 中略 : [./././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
```

想定したとおりのコア配置（ソケット配置順序）になっていることが確認できる（ノードが交互なのも重要）

複数ノード実行（MPI+OpenMPハイブリッド）の場合

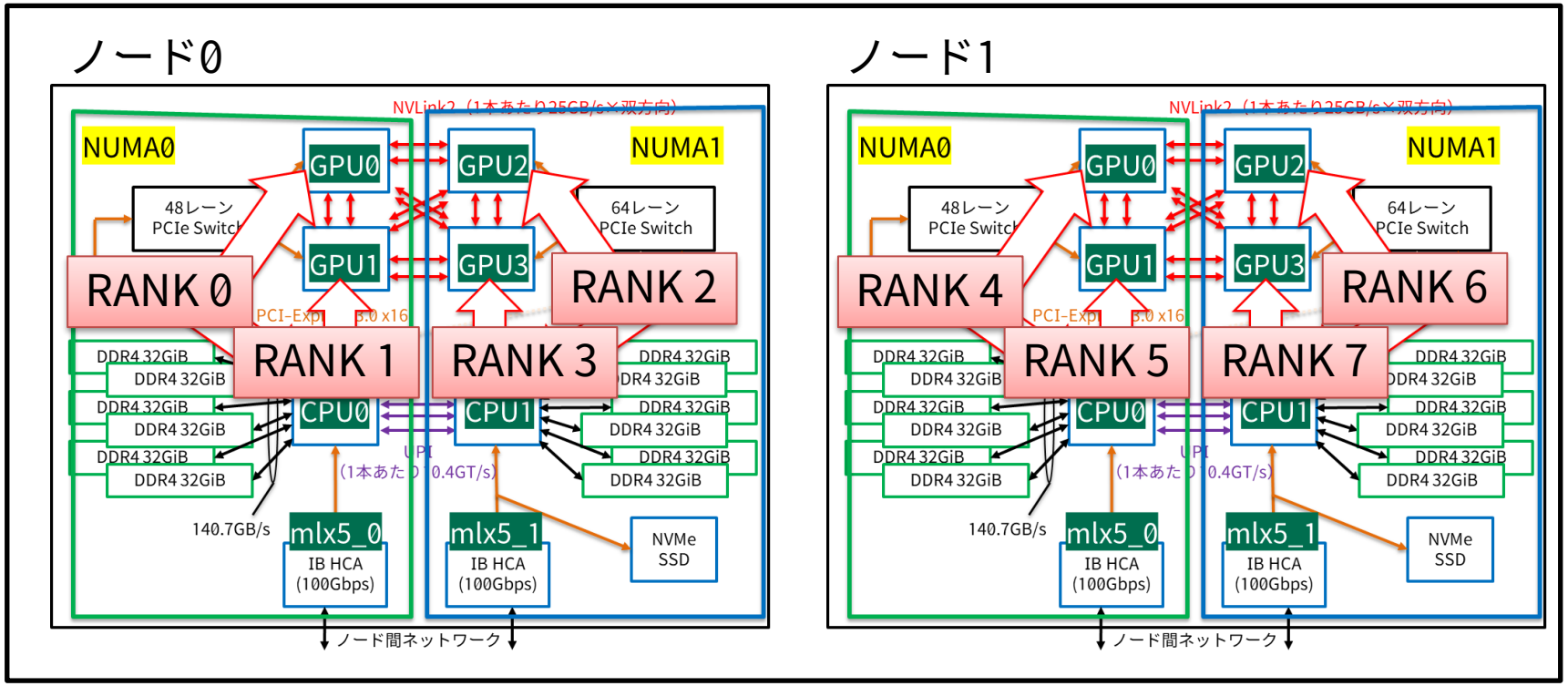
- さらにCPU側ではOpenMPも利用したい場合、プロセスだけではなくスレッドの配置も考慮せねばならない
- GPUが4枚、CPUソケットが2つなのは分かっているため、現実的に需要のある割りあて方法は限られる
  - 「10スレッドOpenMP + 1GPU」を各CPUソケットに2プロセスずつ配置する以外は考えにくい
  - バリエーションがあるとしたらプロセス内のスレッド数（全コアを使うか、2の倍数をとるか等）

- プロセスを10コアに対して配置、というのは実は難しいため、MPIレベルではソケット単位の配置指定のみ行い、あとはnumactlで調整することにする
  - Intel MPIではI_MPI_PIN_DOMAINで一発なのだが

OpenMPI、複数ノード、MPI+OpenMP、利用例1

- compactな配置
  - ノード内の全GPU分のMPIランクを配置したら次のノードへ

配置イメージ→



- MPI版のcompact(-map-by ppr:2:socket)に-bind-to socketを追加すると、コア単位ではなくソケット単位で配置される（デフォルトはコア単位）

```
mpirun -n 8 -machinefile $PJM_O_NODEINF -report-bindings -display-devel-map ¥
-map-by ppr:2:socket -bind-to socket ./run2.sh ./a.out
```

run2.sh

```
#!/bin/bash
export CUDA_VISIBLE_DEVICES=${OMPI_COMM_WORLD_LOCAL_RANK}
CPU1=$(( ${OMPI_COMM_WORLD_LOCAL_RANK} * 10 ))
CPU2=$(( ${CPU1} + 9 ))
numactl -C ${CPU1}-${CPU2} -l $1
```

- さらにnumactlで範囲指定することで配置を調整することでようやく望み通りの配置が確定する

スクリプト

```
#!/bin/bash -x
#PJM -L rscgrp=cx-small
#PJM -L node=2
#PJM -L elapse=00:10:00
#PJM -j
#PJM -S

module load cuda/11.2.1
module load openmpi_cuda/4.0.5

export OMP_NUM_THREADS=10
export OMP_PROC_BIND=CLOSE


mpirun -n 8 -machinefile $PJM_O_NODEINF -report-bindings ¥
-map-by ppr:2:socket -bind-to socket ./run2.sh ./a.out
```

run2.sh

```
#!/bin/bash
env > log_${PJM_JOBID}_${OMPI_COMM_WORLD_RANK}.txt
numactl -s >> log_${PJM_JOBID}_${OMPI_COMM_WORLD_RANK}.txt
export CUDA_VISIBLE_DEVICES=${OMPI_COMM_WORLD_LOCAL_RANK}
CPU1=$(( ${OMPI_COMM_WORLD_LOCAL_RANK} * 10 ))
CPU2=$(( ${CPU1} + 9 ))
numactl -C ${CPU1}-${CPU2} -l $1
```

OMP_NUM_THREADSとOMP_PROC_BINDの指定も必要。
OMP_PROC_BINDをTRUEやCLOSEにしておかないと同じ計算コアに複数のスレッドが配置されることがある。
（実行例のように綺麗にコア番号が並ばなくなってしまう。）

実行例（-report-bindingsの出力）

```
[cx174:00223] MCW rank 0 bound to socket 0[core 0[hwt 0]],  中略 : [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././.]
[cx174:00223] MCW rank 1 bound to socket 0[core 0[hwt 0]],  中略 : [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././.]
[cx174:00223] MCW rank 2 bound to socket 1[core 20[hwt 0]], 中略 : [./././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
[cx174:00223] MCW rank 3 bound to socket 1[core 20[hwt 0]], 中略 : [./././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
[cx175:00170] MCW rank 4 bound to socket 0[core 0[hwt 0]],  中略 : [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././.]
[cx175:00170] MCW rank 5 bound to socket 0[core 0[hwt 0]],  中略 : [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././.]
[cx175:00170] MCW rank 6 bound to socket 1[core 20[hwt 0]], 中略 : [./././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
[cx175:00170] MCW rank 7 bound to socket 1[core 20[hwt 0]], 中略 : [./././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
```

想定通りの挙動となっている

1. ソケット0に2プロセス配置したらソケット1へ
2. 次のノードへ
3. ソケット0に2プロセス配置したらソケット1へ

実行例（コア配置情報）

```
hostname=cx174 rank=0 thread-id= 79 omp-tid= 0 core-id=0
hostname=cx174 rank=0 thread-id=104 omp-tid= 1 core-id=1
hostname=cx174 rank=0 thread-id=108 omp-tid= 2 core-id=2
hostname=cx174 rank=0 thread-id=112 omp-tid= 3 core-id=3
hostname=cx174 rank=0 thread-id=116 omp-tid= 4 core-id=4
hostname=cx174 rank=0 thread-id=121 omp-tid= 5 core-id=5
hostname=cx174 rank=0 thread-id=125 omp-tid= 6 core-id=6
hostname=cx174 rank=0 thread-id=130 omp-tid= 7 core-id=7
hostname=cx174 rank=0 thread-id=135 omp-tid= 8 core-id=8
hostname=cx174 rank=0 thread-id=137 omp-tid= 9 core-id=9
hostname=cx174 rank=1 thread-id= 77 omp-tid= 0 core-id=10
hostname=cx174 rank=1 thread-id=105 omp-tid= 1 core-id=11
hostname=cx174 rank=1 thread-id=109 omp-tid= 2 core-id=12
hostname=cx174 rank=1 thread-id=113 omp-tid= 3 core-id=13
hostname=cx174 rank=1 thread-id=117 omp-tid= 4 core-id=14
hostname=cx174 rank=1 thread-id=120 omp-tid= 5 core-id=15
hostname=cx174 rank=1 thread-id=124 omp-tid= 6 core-id=16
hostname=cx174 rank=1 thread-id=129 omp-tid= 7 core-id=17
hostname=cx174 rank=1 thread-id=134 omp-tid= 8 core-id=18
hostname=cx174 rank=1 thread-id=136 omp-tid= 9 core-id=19
hostname=cx174 rank=2 thread-id= 80 omp-tid= 0 core-id=20
hostname=cx174 rank=2 thread-id=103 omp-tid= 1 core-id=21
hostname=cx174 rank=2 thread-id=107 omp-tid= 2 core-id=22
hostname=cx174 rank=2 thread-id=111 omp-tid= 3 core-id=23
hostname=cx174 rank=2 thread-id=115 omp-tid= 4 core-id=24
hostname=cx174 rank=2 thread-id=119 omp-tid= 5 core-id=25
hostname=cx174 rank=2 thread-id=123 omp-tid= 6 core-id=26
hostname=cx174 rank=2 thread-id=127 omp-tid= 7 core-id=27
hostname=cx174 rank=2 thread-id=128 omp-tid= 8 core-id=28
hostname=cx174 rank=2 thread-id=132 omp-tid= 9 core-id=29
右上へ
```

```
hostname=cx174 rank=3 thread-id= 81 omp-tid= 0 core-id=30
hostname=cx174 rank=3 thread-id=102 omp-tid= 1 core-id=31
hostname=cx174 rank=3 thread-id=106 omp-tid= 2 core-id=32
hostname=cx174 rank=3 thread-id=110 omp-tid= 3 core-id=33
hostname=cx174 rank=3 thread-id=114 omp-tid= 4 core-id=34
hostname=cx174 rank=3 thread-id=118 omp-tid= 5 core-id=35
hostname=cx174 rank=3 thread-id=122 omp-tid= 6 core-id=36
hostname=cx174 rank=3 thread-id=126 omp-tid= 7 core-id=37
hostname=cx174 rank=3 thread-id=131 omp-tid= 8 core-id=38
hostname=cx174 rank=3 thread-id=133 omp-tid= 9 core-id=39
hostname=cx175 rank=4 thread-id= 23 omp-tid= 0 core-id=0
hostname=cx175 rank=4 thread-id= 49 omp-tid= 1 core-id=1
hostname=cx175 rank=4 thread-id= 51 omp-tid= 2 core-id=2
hostname=cx175 rank=4 thread-id= 53 omp-tid= 3 core-id=3
hostname=cx175 rank=4 thread-id= 56 omp-tid= 4 core-id=4
hostname=cx175 rank=4 thread-id= 59 omp-tid= 5 core-id=5
hostname=cx175 rank=4 thread-id= 61 omp-tid= 6 core-id=6
hostname=cx175 rank=4 thread-id= 63 omp-tid= 7 core-id=7
hostname=cx175 rank=4 thread-id= 65 omp-tid= 8 core-id=8
hostname=cx175 rank=4 thread-id= 69 omp-tid= 9 core-id=9
hostname=cx175 rank=5 thread-id= 24 omp-tid= 0 core-id=10
hostname=cx175 rank=5 thread-id= 48 omp-tid= 1 core-id=11
hostname=cx175 rank=5 thread-id= 50 omp-tid= 2 core-id=12
hostname=cx175 rank=5 thread-id= 52 omp-tid= 3 core-id=13
hostname=cx175 rank=5 thread-id= 54 omp-tid= 4 core-id=14
hostname=cx175 rank=5 thread-id= 58 omp-tid= 5 core-id=15
hostname=cx175 rank=5 thread-id= 60 omp-tid= 6 core-id=16
hostname=cx175 rank=5 thread-id= 62 omp-tid= 7 core-id=17
hostname=cx175 rank=5 thread-id= 64 omp-tid= 8 core-id=18
hostname=cx175 rank=5 thread-id= 68 omp-tid= 9 core-id=19
右上へ
```

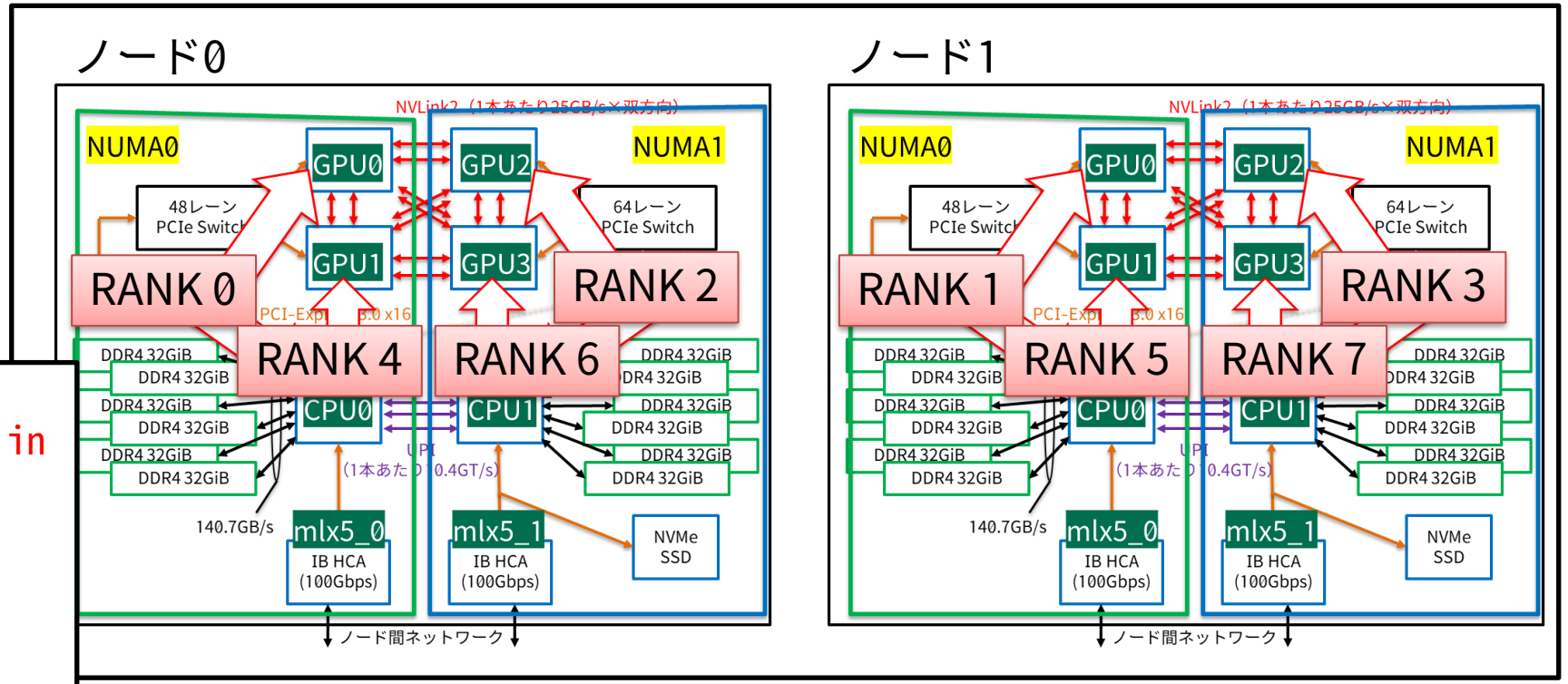
```
hostname=cx175 rank=6 thread-id= 26 omp-tid= 0 core-id=20
hostname=cx175 rank=6 thread-id= 55 omp-tid= 1 core-id=21
hostname=cx175 rank=6 thread-id= 66 omp-tid= 2 core-id=22
hostname=cx175 rank=6 thread-id= 71 omp-tid= 3 core-id=23
hostname=cx175 rank=6 thread-id= 73 omp-tid= 4 core-id=24
hostname=cx175 rank=6 thread-id= 75 omp-tid= 5 core-id=25
hostname=cx175 rank=6 thread-id= 77 omp-tid= 6 core-id=26
hostname=cx175 rank=6 thread-id= 78 omp-tid= 7 core-id=27
hostname=cx175 rank=6 thread-id= 80 omp-tid= 8 core-id=28
hostname=cx175 rank=6 thread-id= 83 omp-tid= 9 core-id=29
hostname=cx175 rank=7 thread-id= 27 omp-tid= 0 core-id=30
hostname=cx175 rank=7 thread-id= 57 omp-tid= 1 core-id=31
hostname=cx175 rank=7 thread-id= 67 omp-tid= 2 core-id=32
hostname=cx175 rank=7 thread-id= 70 omp-tid= 3 core-id=33
hostname=cx175 rank=7 thread-id= 72 omp-tid= 4 core-id=34
hostname=cx175 rank=7 thread-id= 74 omp-tid= 5 core-id=35
hostname=cx175 rank=7 thread-id= 76 omp-tid= 6 core-id=36
hostname=cx175 rank=7 thread-id= 79 omp-tid= 7 core-id=37
hostname=cx175 rank=7 thread-id= 81 omp-tid= 8 core-id=38
hostname=cx175 rank=7 thread-id= 82 omp-tid= 9 core-id=39
```

※ちなみに2のべき乗スレッド数を使いたい場合は、
`export OMP_NUM_THREADS=8`
に変更するだけで、コア番号
0-7, 10-17, 20-27, 30-37だけが使われるようになる。

rank0, 1は1つめのホストのソケット0（コアID 0-19），rank2, 3は1つめのホストのソケット1（コアID 20-39），
rank4, 5は2つめのホストのソケット0（コアID 0-19），rank6, 7は2つめのホストのソケット1（コアID 20-39）

OpenMPI、複数ノード、MPI+OpenMP、利用例2

- scatterな配置
  - ノード内のGPUに1つMPIランクを配置したら次のノードへ
  - なるべく分散させる




```
#!/bin/bash
case ${OMPI_COMM_WORLD_LOCAL_RANK} in
0) export CUDA_VISIBLE_DEVICES=0
numactl -C 0-9 -l $1
;;
1) export CUDA_VISIBLE_DEVICES=2
numactl -C 20-29 -l $1
;;
2) export CUDA_VISIBLE_DEVICES=1
numactl -C 10-19 -l $1
;;
3) export CUDA_VISIBLE_DEVICES=3
numactl -C 30-39 -l $1
;;
esac
```

```
mpirun -n 8 -machinefile $PJM_O_NODEINF -report-bindings -display-devel-map ¥
-map-by node -bind-to socket ./run2.sh ./a.out
```

- mpirun側はフラットMPIの場合と同じ
- run2.sh側のnumactlによる細かい配置を変更した

スクリプト

```
#!/bin/bash -x
#PJM -L rscgrp=cx-small
#PJM -L node=2
#PJM -L elapse=00:10:00
#PJM -j
#PJM -S

module load cuda/11.2.1
module load openmpi_cuda/4.0.5

export OMP_NUM_THREADS=10
export OMP_PROC_BIND=CLOSE

mpirun -n 8 -machinefile $PJM_O_NODEINF -report-bindings ¥
-map-by node -bind-to socket ./run2.sh ./a.out
```

run2.sh

```
#!/bin/bash
env > log_${PJM_JOBID}_${OMPI_COMM_WORLD_RANK}.txt
numactl -s >> log_${PJM_JOBID}_${OMPI_COMM_WORLD_RANK}.txt

case ${OMPI_COMM_WORLD_LOCAL_RANK} in
0) export CUDA_VISIBLE_DEVICES=0
numactl -C 0-9 -l $1
;;
1) export CUDA_VISIBLE_DEVICES=2
numactl -C 20-29 -l $1
;;
2) export CUDA_VISIBLE_DEVICES=1
numactl -C 10-19 -l $1
;;
3) export CUDA_VISIBLE_DEVICES=3
numactl -C 30-39 -l $1
;;
esac
```

実行例（-report-bindingsの出力）

```
[cx174:00061] MCW rank 0 bound to socket 0[core 0[hwt 0]],  中略 : [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././.]
[cx174:00061] MCW rank 2 bound to socket 1[core 20[hwt 0]], 中略 : [./././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
[cx174:00061] MCW rank 4 bound to socket 0[core 0[hwt 0]],  中略 : [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././.]
[cx174:00061] MCW rank 6 bound to socket 1[core 20[hwt 0]], 中略 : [./././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
[cx175:00009] MCW rank 1 bound to socket 0[core 0[hwt 0]],  中略 : [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././.]
[cx175:00009] MCW rank 3 bound to socket 1[core 20[hwt 0]], 中略 : [./././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
[cx175:00009] MCW rank 5 bound to socket 0[core 0[hwt 0]],  中略 : [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././.]
[cx175:00009] MCW rank 7 bound to socket 1[core 20[hwt 0]], 中略 : [./././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
```

実行例（コア配置情報）

```
hostname=cx174 rank=0 thread-id= 76 omp-tid= 0 core-id=0
hostname=cx174 rank=0 thread-id=102 omp-tid= 1 core-id=1
hostname=cx174 rank=0 thread-id=105 omp-tid= 2 core-id=2
hostname=cx174 rank=0 thread-id=109 omp-tid= 3 core-id=3
hostname=cx174 rank=0 thread-id=112 omp-tid= 4 core-id=4
hostname=cx174 rank=0 thread-id=115 omp-tid= 5 core-id=5
hostname=cx174 rank=0 thread-id=119 omp-tid= 6 core-id=6
hostname=cx174 rank=0 thread-id=124 omp-tid= 7 core-id=7
hostname=cx174 rank=0 thread-id=128 omp-tid= 8 core-id=8
hostname=cx174 rank=0 thread-id=132 omp-tid= 9 core-id=9
hostname=cx174 rank=2 thread-id= 78 omp-tid= 0 core-id=20
hostname=cx174 rank=2 thread-id=107 omp-tid= 1 core-id=21
hostname=cx174 rank=2 thread-id=110 omp-tid= 2 core-id=22
hostname=cx174 rank=2 thread-id=116 omp-tid= 3 core-id=23
hostname=cx174 rank=2 thread-id=121 omp-tid= 4 core-id=24
hostname=cx174 rank=2 thread-id=125 omp-tid= 5 core-id=25
hostname=cx174 rank=2 thread-id=130 omp-tid= 6 core-id=26
hostname=cx174 rank=2 thread-id=133 omp-tid= 7 core-id=27
hostname=cx174 rank=2 thread-id=136 omp-tid= 8 core-id=28
hostname=cx174 rank=2 thread-id=137 omp-tid= 9 core-id=29
hostname=cx174 rank=4 thread-id= 80 omp-tid= 0 core-id=10
hostname=cx174 rank=4 thread-id=103 omp-tid= 1 core-id=11
hostname=cx174 rank=4 thread-id=104 omp-tid= 2 core-id=12
hostname=cx174 rank=4 thread-id=108 omp-tid= 3 core-id=13
hostname=cx174 rank=4 thread-id=113 omp-tid= 4 core-id=14
hostname=cx174 rank=4 thread-id=114 omp-tid= 5 core-id=15
hostname=cx174 rank=4 thread-id=120 omp-tid= 6 core-id=16
hostname=cx174 rank=4 thread-id=123 omp-tid= 7 core-id=17
hostname=cx174 rank=4 thread-id=127 omp-tid= 8 core-id=18
hostname=cx174 rank=4 thread-id=131 omp-tid= 9 core-id=19
右上へ
```

```
hostname=cx174 rank=6 thread-id= 81 omp-tid= 0 core-id=30
hostname=cx174 rank=6 thread-id=106 omp-tid= 1 core-id=31
hostname=cx174 rank=6 thread-id=111 omp-tid= 2 core-id=32
hostname=cx174 rank=6 thread-id=117 omp-tid= 3 core-id=33
hostname=cx174 rank=6 thread-id=118 omp-tid= 4 core-id=34
hostname=cx174 rank=6 thread-id=122 omp-tid= 5 core-id=35
hostname=cx174 rank=6 thread-id=126 omp-tid= 6 core-id=36
hostname=cx174 rank=6 thread-id=129 omp-tid= 7 core-id=37
hostname=cx174 rank=6 thread-id=134 omp-tid= 8 core-id=38
hostname=cx174 rank=6 thread-id=135 omp-tid= 9 core-id=39
hostname=cx175 rank=1 thread-id= 25 omp-tid= 0 core-id=0
hostname=cx175 rank=1 thread-id= 53 omp-tid= 1 core-id=1
hostname=cx175 rank=1 thread-id= 63 omp-tid= 2 core-id=2
hostname=cx175 rank=1 thread-id= 70 omp-tid= 3 core-id=3
hostname=cx175 rank=1 thread-id= 72 omp-tid= 4 core-id=4
hostname=cx175 rank=1 thread-id= 74 omp-tid= 5 core-id=5
hostname=cx175 rank=1 thread-id= 77 omp-tid= 6 core-id=6
hostname=cx175 rank=1 thread-id= 79 omp-tid= 7 core-id=7
hostname=cx175 rank=1 thread-id= 80 omp-tid= 8 core-id=8
hostname=cx175 rank=1 thread-id= 82 omp-tid= 9 core-id=9
hostname=cx175 rank=3 thread-id= 26 omp-tid= 0 core-id=20
hostname=cx175 rank=3 thread-id= 49 omp-tid= 1 core-id=21
hostname=cx175 rank=3 thread-id= 51 omp-tid= 2 core-id=22
hostname=cx175 rank=3 thread-id= 54 omp-tid= 3 core-id=23
hostname=cx175 rank=3 thread-id= 57 omp-tid= 4 core-id=24
hostname=cx175 rank=3 thread-id= 59 omp-tid= 5 core-id=25
hostname=cx175 rank=3 thread-id= 61 omp-tid= 6 core-id=26
hostname=cx175 rank=3 thread-id= 65 omp-tid= 7 core-id=27
hostname=cx175 rank=3 thread-id= 67 omp-tid= 8 core-id=28
hostname=cx175 rank=3 thread-id= 69 omp-tid= 9 core-id=29
右上へ
```

```
hostname=cx175 rank=5 thread-id= 24 omp-tid= 0 core-id=10
hostname=cx175 rank=5 thread-id= 55 omp-tid= 1 core-id=11
hostname=cx175 rank=5 thread-id= 64 omp-tid= 2 core-id=12
hostname=cx175 rank=5 thread-id= 71 omp-tid= 3 core-id=13
hostname=cx175 rank=5 thread-id= 73 omp-tid= 4 core-id=14
hostname=cx175 rank=5 thread-id= 75 omp-tid= 5 core-id=15
hostname=cx175 rank=5 thread-id= 76 omp-tid= 6 core-id=16
hostname=cx175 rank=5 thread-id= 78 omp-tid= 7 core-id=17
hostname=cx175 rank=5 thread-id= 81 omp-tid= 8 core-id=18
hostname=cx175 rank=5 thread-id= 83 omp-tid= 9 core-id=19
hostname=cx175 rank=7 thread-id= 27 omp-tid= 0 core-id=30
hostname=cx175 rank=7 thread-id= 48 omp-tid= 1 core-id=31
hostname=cx175 rank=7 thread-id= 50 omp-tid= 2 core-id=32
hostname=cx175 rank=7 thread-id= 52 omp-tid= 3 core-id=33
hostname=cx175 rank=7 thread-id= 56 omp-tid= 4 core-id=34
hostname=cx175 rank=7 thread-id= 58 omp-tid= 5 core-id=35
hostname=cx175 rank=7 thread-id= 60 omp-tid= 6 core-id=36
hostname=cx175 rank=7 thread-id= 62 omp-tid= 7 core-id=37
hostname=cx175 rank=7 thread-id= 66 omp-tid= 8 core-id=38
hostname=cx175 rank=7 thread-id= 68 omp-tid= 9 core-id=39
```

rank0, 1は各ホストのソケット0（コアID 0-9），rank2, 3は各ホストのソケット1（コアID 20-29），
rank4, 5は各ホストのソケット0（コアID 10-19），rank6, 7は各ホストのソケット1（コアID 30-39）

OpenMPI、複数ノード、MPI+OpenACC

- hpc_sdk/12.1を使えばMPI＋OpenACCによる複数GPUプログラムの作成・実行が可能
- cudaMemcpyを使わずに直接MPI通信も可能
- ソースコード例は次ページ
- コンパイル例

```
mpicc -fast -mp -acc -tp=skylake -gpu=cc70 source.c
```

※ /home/center/opt/x86_64/cores/hpc_sdk/Linux_x86_64/21.2/comm_libs/mpi/bin/mpicc が実行され、内部ではnvcコンパイラが呼び出される

- 実行例

```
#!/bin/bash -x
#PJM -L rscgrp=cx-small
#PJM -L node=2
#PJM -j
#PJM -S

module load hpc_sdk/21.2

mpirun -n 2 -machinefile $PJM_O_NODEINF -map-by node -bind-to socket -report-bindings ./a.out
```

1ノードに1プロセス配置したら次のノードへ

左側：update hostで明示的にCPU-GPU間データ転送する場合

```
    double *data, *data2;
    int i, r, ret;
    data = (double*)malloc(sizeof(double)*10);
    data2 = (double*)malloc(sizeof(double)*10);
    for(i=0; i<10; i++)data[i]=(double)rank + (double)(i)*0.01;
    for(r=0; r<size; r++){
      if(r==rank){
        printf("rank(before) %d:", rank);
        for(i=0; i<10; i++)printf(" %f", data[i]);
        printf("￥n");
      }
      MPI_Barrier(MPI_COMM_WORLD);
    }
```

初期データの準備

```
#pragma acc data copy(data[10],data2[10])
    {
#pragma acc kernels
      for(i=0; i<10; i++){
        data[i] += 0.001;
      }
#pragma acc update host(data[10])
      ret = MPI_Allreduce(data, data2, 10, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
#pragma acc update device(data2[10])
#pragma acc kernels
      for(i=0; i<10; i++){
        data2[i] += 0.0001;
      }
    }
```

GPUからCPUへのデータ転送

ホストメモリ間のMPI通信

CPUからGPUへのデータ転送

GPU上での処理

```
    for(r=0; r<size; r++){
      if(r==rank){
        printf("rank(after) %d:", rank);
        for(i=0; i<10; i++)printf(" %f", data2[i]);
        printf("￥n");
      }
      MPI_Barrier(MPI_COMM_WORLD);
    }
    free(data);
    free(data2);
```

結果の出力など

右側：GPUメモリを直接通信する場合

```
    double *data, *data2;
    int i, r, ret;
    data = (double*)malloc(sizeof(double)*10);
    data2 = (double*)malloc(sizeof(double)*10);
    for(i=0; i<10; i++)data[i]=(double)rank + (double)(i)*0.01;
    for(r=0; r<size; r++){
      if(r==rank){
        printf("rank(before) %d:", rank);
        for(i=0; i<10; i++)printf(" %f", data[i]);
        printf("￥n");
      }
      MPI_Barrier(MPI_COMM_WORLD);
    }
```

```
#pragma acc data copy(data[10],data2[10])
    {
#pragma acc kernels
      for(i=0; i<10; i++){
        data[i] += 0.001;
      }
#pragma acc host_data use_device(data[10],data2[10])
      ret = MPI_Allreduce(data, data2, 10, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);

#pragma acc kernels
      for(i=0; i<10; i++){
        data2[i] += 0.0001;
      }
    }
```

デバイスメモリ間のMPI通信

```
    for(r=0; r<size; r++){
      if(r==rank){
        printf("rank(after) %d:", rank);
        for(i=0; i<10; i++)printf(" %f", data2[i]);
        printf("￥n");
      }
      MPI_Barrier(MPI_COMM_WORLD);
    }
    free(data);
    free(data2);
```

補足事項

- インストールされているOpenMPIではなく自分で入手したOpenMPIを使って複数ノードMPI通信を行う場合は以下の環境変数を設定してください

```
export OMPI_MCA_plm_rsh_agent=/bin/pjrsh
```

※mpirunに -mca plm_rsh_agent /bin/pjrsh という引数を追加しても同じ効果が得られる

- 同様に右のWARNINGが出る場合は以下を設定してください

```
export OMPI_MCA_btl_openib_warn_default_gid_prefix=0
```

- OpenMPIやHPC_SDKのmodulefileを用いた場合は自動的に設定されるようにしてあります

```
--------------------------------------------------------------------------
WARNING: There are more than one active ports on host 'cx120', but the
default subnet GID prefix was detected on more than one of these
ports.  If these ports are connected to different physical IB
networks, this configuration will fail in Open MPI.  This version of
Open MPI requires that every physically separate IB subnet that is
used between connected MPI processes must have different subnet ID
values.

Please see this FAQ entry for more details:

  http://www.open-mpi.org/faq/?category=openfabrics#ofa-default-subnet-gid

NOTE: You can turn off this warning by setting the MCA parameter
      btl_openib_warn_default_gid_prefix to 0.
--------------------------------------------------------------------------
```

Intel MPIを使う場合

- Intel MPIの場合はクラウドシステム利用時と基本的に同じであるため、「クラウドシステム向けのプロセス・スレッド配置方法」を参照してください。
- 使用するGPUを明示的に指定したい場合はCUDA_VISIBLE_DEVICES環境変数に番号を与えてください。

参考：プロセスの配置情報を確認する方法（プログラム側から）

- /proc/{PID}/task/{TID}/stat
  の39列目（processor情報）
  にコア番号が入っている
- 確認用テストプログラムの例　→
- サブシステムを問わず利用可能、
  以下はType Iにおける
  4プロセス×4スレッドでの実行例（ソート済み）

```
hostname=fx2271 rank=0 thread-id=63 omp-tid= 0 core-id=12
hostname=fx2271 rank=0 thread-id=77 omp-tid= 1 core-id=13
hostname=fx2271 rank=0 thread-id=80 omp-tid= 2 core-id=14
hostname=fx2271 rank=0 thread-id=84 omp-tid= 3 core-id=15
hostname=fx2271 rank=1 thread-id=65 omp-tid= 0 core-id=24
hostname=fx2271 rank=1 thread-id=75 omp-tid= 1 core-id=25
hostname=fx2271 rank=1 thread-id=79 omp-tid= 2 core-id=26
hostname=fx2271 rank=1 thread-id=83 omp-tid= 3 core-id=27
hostname=fx2271 rank=2 thread-id=64 omp-tid= 0 core-id=36
hostname=fx2271 rank=2 thread-id=78 omp-tid= 1 core-id=37
hostname=fx2271 rank=2 thread-id=82 omp-tid= 2 core-id=38
hostname=fx2271 rank=2 thread-id=86 omp-tid= 3 core-id=39
hostname=fx2271 rank=3 thread-id=66 omp-tid= 0 core-id=48
hostname=fx2271 rank=3 thread-id=76 omp-tid= 1 core-id=49
hostname=fx2271 rank=3 thread-id=81 omp-tid= 2 core-id=50
hostname=fx2271 rank=3 thread-id=85 omp-tid= 3 core-id=51
```

```c
#include <unistd.h>
#include <sys/types.h>
#include <sys/syscall.h>
#include <mpi.h>
#include <omp.h>

#define min(a,b) a<b?a:b
void check_core(int rank){
  char buf[0xff], buf2[4], hostname[0xff];
  FILE *fp;
  int pid, tid, ompid, count, c1, c2;
  pid = getpid();
  tid = (pid_t) syscall(SYS_gettid);
  ompid = omp_get_thread_num();
  sprintf(buf, "/proc/%d/task/%d/stat", pid, tid);
  if ((fp = fopen(buf, "r")) != NULL) {
    fgets(buf, 0xff, fp);
    fclose(fp);
    count = 0;
    c1 = c2 = 0;
    while(buf[c1]!='￥0'){
      if(buf[c1]==' ')count++;
      c1++;
      if(count==38)break;
    }
    c2 = c1;
    while(buf[c2]!='￥0'){
      if(buf[c2]==' ')break;
      c2++;
    }
    strncpy(buf2, &buf[c1], min(c2-c1, 4));
    buf2[min(c2-c1, 4)] = '￥0';
    gethostname(hostname, 0xff);
    printf("hostname=%s rank=%d thread-id=%2d omp-tid=%2d core-id=%s￥n",
        hostname, rank, tid, ompid, buf2);
  }
}
```

※strtokで分解したらNULLを喰らうことが度々あったため、手動で分割している