プロセスとスレッドの適切な割り当て ノードグループA編

この資料について

- 玄界の各計算ノードグループには複数のCPUやGPUやNICが搭載されている
- 最大性能を得るためには、それらの配置を考えたプログラムの実行が必要

- この資料ではプロセスやスレッドの割り当てを最適化するためのヒントについて説明する
  - ノードグループAにはGPUがないため、CPUを用いたOpenMPやMPIプログラムについてのみ

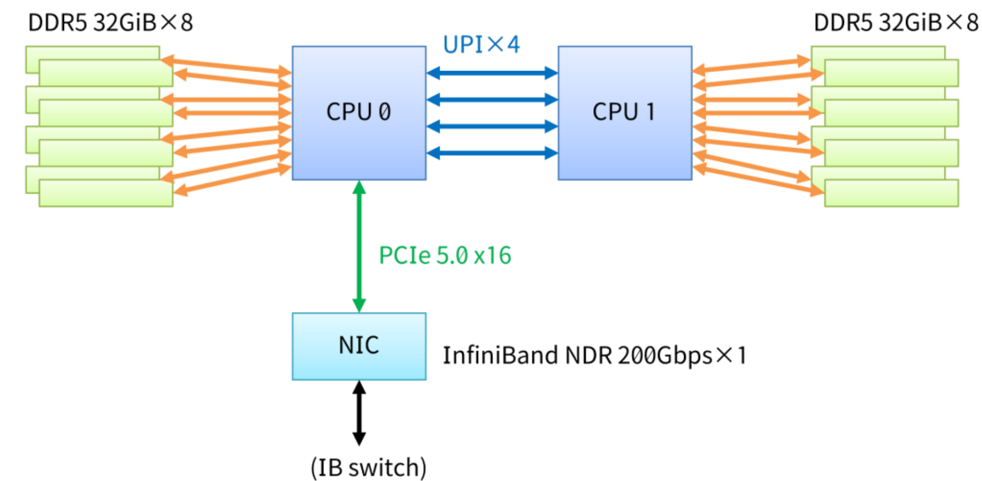
ハードウェア構成とNUMA構成の基礎知識

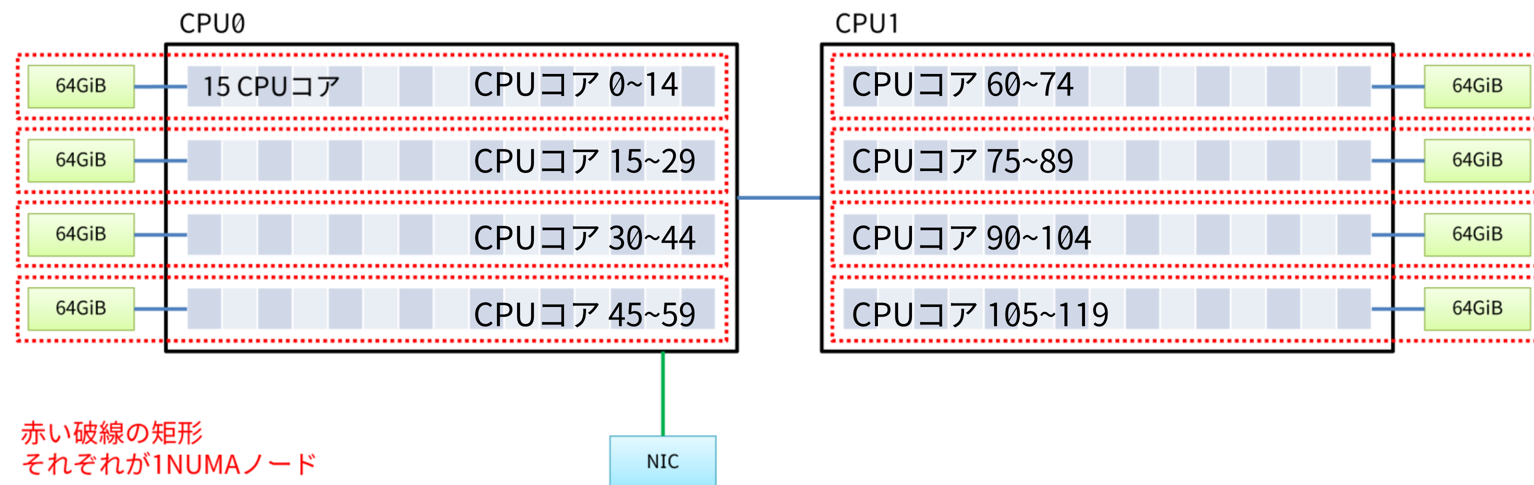
最適なプログラム実行のためには対象ハードウェアに関するある程度の知識が必要。

はじめに、計算ノードグループのハードウェア構成の概要と、ユーザ（プログラム）からどのように見えるのかについて説明する。

基本的な構成の理解

- 左図：ノードグループAのCPU・GPU・メモリ・NICの物理的な構成
  - メモリはそれぞれのCPUソケットにつながっているため、異なるCPUソケットにつながっているメモリにアクセスするには時間がかかる。
  - NICはCPU0にのみ接続されているため、ノード間通信はCPU1よりもCPU0が行った方が高速。
- 右図：CPUの内部は4つのグループ（NUMAノード）に分かれている
  - NUMAノードをまたぐよりもまたがない方がメモリアクセスが高速（ソケットをまたぐのと比べると小さな差）
- 実行時には `numactl -H`, `numactl -s`, `lstopo` などで確認可能





ユーザからはどう見える？（numactl、1ノード占有ジョブ実行時）

- CPU2ソケットで合計8つのNUMAノードが確認できる

```
[ku40000105@a0001 ~]$ numactl -H
available: 8 nodes (0-7)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
node 0 size: 63904 MB
node 0 free: 63005 MB
node 1 cpus: 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
node 1 size: 64508 MB
node 1 free: 64001 MB
node 2 cpus: 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
node 2 size: 64508 MB
node 2 free: 64019 MB
node 3 cpus: 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
node 3 size: 64464 MB
node 3 free: 63366 MB
node 4 cpus: 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
node 4 size: 64508 MB
node 4 free: 63649 MB
node 5 cpus: 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
node 5 size: 64508 MB
node 5 free: 63845 MB
node 6 cpus: 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104
node 6 size: 64508 MB
node 6 free: 64026 MB
node 7 cpus: 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
node 7 size: 64505 MB
node 7 free: 62155 MB
```

右上へ続く

0から3がCPU0、4から7がCPU1

```
node distances:
node   0   1   2   3   4   5   6   7
  0:  10  12  12  12  21  21  21  21
  1:  12  10  12  12  21  21  21  21
  2:  12  12  10  12  21  21  21  21
  3:  12  12  12  10  21  21  21  21
  4:  21  21  21  21  10  12  12  12
  5:  21  21  21  21  12  10  12  12
  6:  21  21  21  21  12  12  10  12
  7:  21  21  21  21  12  12  12  10

[ku40000105@a0001 ~]$ numactl -s
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
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102
103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
118 119
cpubind: 0 1 2 3 4 5 6 7
nodebind: 0 1 2 3 4 5 6 7
membind: 0 1 2 3 4 5 6 7
```

120コア、8NUMAノード

ユーザからはどう見える？（numactl、1CPUコアジョブ実行時）

- 確認方法によって見え方が違う

```
[ku40000105@a0803 ~]$ numactl -H
available: 8 nodes (0-7)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
node 0 size: 63904 MB
node 0 free: 59468 MB
node 1 cpus: 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
node 1 size: 64508 MB
node 1 free: 52288 MB
node 2 cpus: 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
node 2 size: 64508 MB
node 2 free: 56552 MB
node 3 cpus: 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
node 3 size: 64508 MB
node 3 free: 55971 MB
node 4 cpus: 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
node 4 size: 64508 MB
node 4 free: 56900 MB
node 5 cpus: 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
node 5 size: 64508 MB
node 5 free: 40637 MB
node 6 cpus: 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104
node 6 size: 64508 MB
node 6 free: 59927 MB
node 7 cpus: 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
node 7 size: 64461 MB
node 7 free: 53811 MB
```

右上へ続く

```
node distances:
node   0   1   2   3   4   5   6   7
  0:  10  12  12  12  21  21  21  21
  1:  12  10  12  12  21  21  21  21
  2:  12  12  10  12  21  21  21  21
  3:  12  12  12  10  21  21  21  21
  4:  21  21  21  21  10  12  12  12
  5:  21  21  21  21  12  10  12  12
  6:  21  21  21  21  12  12  10  12
  7:  21  21  21  21  12  12  12  10

[ku40000105@a0803 ~]$ numactl -s
policy: default
preferred node: current
physcpubind: 75
cpubind: 5
nodebind: 5
membind: 0 1 2 3 4 5 6 7
```

numactl -s では使える資源のみが見える
メモリノードだけはノード全体が見えてしまう（仕様）

numactl -Hではハードウェア情報が見えてしまうためノード占有ジョブと変化なし

何を考える必要がある？　1/2

- CPUとメモリを同じNUMAノードに配置すること
  - （プログラムによっては、あえてこれをやらない方が良いこともある）
- ノードごとにデータを集約してからノード間通信をする場合は、NICが接続されている側のCPUに集約すること
- プロセスやスレッドの配置順序
  - NUMAノード内で連続になるように配置するか、広く分散させるか

何を考える必要がある？　2/2

- ファーストタッチ（主にOpenMPプログラム）
  - 対象変数（配列）に対するキャッシュは、対象変数に最初にアクセスしたスレッドの存在する計算コアのキャッシュに割り当てられる
  - 並列計算ループ（主な計算処理）で行うのと同じ配列アクセスパターンの初期化ループを用意して実行しておくと、主な計算時に適切にキャッシュが使われて最大性能が得られる（低下を防げる）
  - 例

```
// 対象配列に対するアクセスが初めて生じる位置に追加する。
// 実際に計算をするループと同じスレッド割り当てが行われる必要があるため、
// schedule(dynamic)などを使うと効果が得られない。
#pragma omp parallel for
for(i=0; i<N; i++){
  dst[i] = srt[i] = 0;
}

// 本来のデータ初期化処理（省略）

// 実際に計算をするループ
#pragma omp parallel for
for(i=0; i<N; i++){
  dst[i] = src[i] * value;
}
```

考えられるプログラム実行パターン

1. ノード全体を占有するジョブ
   - node=ノード数（もしくはvnode-core=120）指定を行ってジョブを実行（pjsub）
   1. OpenMPのみ
      1. ノード全体に均等にスレッドを配置して実行
      2. CPUソケットやNUMAノードを限定し、その範囲内で均等にスレッドを配置して実行（※ノードを占有する意味があまりない。性能比較用などに。）
   2. MPIのみ
      1. ノード全体に均等にプロセスを配置して実行
   3. MPI + OpenMP
      1. NUMAノードごとにMPIプロセスを配置、NUMAノード内でOpenMP並列化
      2. CPUソケットごとにMPIプロセスを配置、CPUソケット内でOpenMP並列化
2. ノードの一部のみを使う
   - vnode-core=コア数（120未満） 指定を行ってジョブを実行（pjsub）
   - ノードの一部しか使えないため、細かい資源割り当ての最適化は不可能（本資料では最低限の説明のみ）

numactlによるプログラムの最適化（実行方法の調整）

- numactl
  - 対象プログラムをどのNUMAノードやコア上で実行させるかや、どのメモリを使わせるかなどを指定するプログラム
  - MPIプログラムやOpenMPプログラムにも影響し、挙動を細かく調整することができる

- numactlの使い方
  - numactl オプション 対象プログラム　の形式でプログラムを実行する
  - 主なオプション
    - --cpunodebind/-N　スレッドを配置するNUMAノード番号を指定
    - --physcpubind/-C　スレッドを配置するCPUコア番号を指定
    - --membind/-m　指定したNUMAノードに接続されたメモリを使う
    - --interleave/-i　指定した範囲のメモリを均等に使う
    - --localalloc/-l　計算コアに近いメモリのみを使う
  - 多くの場合は--localallocが高速と思われるが、大容量のメモリを使う場合などに--interleave=allなどを活用すると良い

numactlの利用例　1/2

- NUMAノード0上で実行、メモリもNUMAノード0に接続されたもののみ利用
  - NUMAノード単位で指定する例
    - `numactl -N 0 -l ./a.out`
    - `numactl --cpunodebind=0 --localalloc ./a.out`
  - CPUコア単位で指定する例
    - `numactl -C 0-14 -l ./a.out`
    - `numactl --physcpubind=0-14 --localalloc ./a.out`
- CPUソケット0上で実行、メモリもCPUソケット0に接続されたもののみ利用
  - numactlで指定できるのはコアとNUMAノードのみのため、ソケット単位の指定は対応するコアやNUMAノードの組み合わせで指定する
  - `numactl -N 0-3 -l ./a.out`
  - `numactl --cpunodebind=0-3 --localalloc ./a.out`
  - `numactl -C 0-59 -l ./a.out`
  - `numactl --physcpubind=0-59 --localalloc ./a.out`

numactlの利用例　2/2

- CPUソケット0上で実行するが、メモリはノード全体を使う
  - 多くの容量のメモリを使いたい場合に有効
  - numactl -N 0-3 -i all ./a.out
  - numactl --cpunodebind=0-3 --interleave=all ./a.out

- 対象プログラムとして「numactl -s」を実行すると割り当て情報を確認しやすい
  - numactl -N 0 -l numactl -s
  - 結果

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

- NUMAノード0のCPU情報のみが見えるようになった
- localalloc指定してもmembindの表示は変わらない点には注意（実際には0だけが使われるはずである）

OpenMPプログラムの具体的な実行方法

OpenMPプログラムの実行を考える

- 問題設定（想定する実行条件）
  - ノードを占有したプログラム実行を対象としているため、ある程度多くのスレッドを用いた実行を想定
  - メモリは numactl --localalloc を指定しCPUコアに近いものを使用（必要に応じて変更して良い）

- 考えること
  - スレッドをどのCPUコアに割り当てるか　および　割り当てる順序をどうするか

- どのような最適化が可能か（最適化に使える推奨の手段は何があるか）
  - OpneMPの仕様に定められた環境変数
  - コンパイラごとの環境変数
  - numactlコマンド（解説済み）

OpenMPプログラムの制御方法

- OpenMPの仕様に定められた環境変数
  - OMP_PROC_BIND、OMP_PLACES
    - OMP_PROC_BIND=CLOSE　スレッドを隣接するCPUコアに割り当てる
    - OMP_PROC_BIND=SPREAD　スレッドを全体的に均等に割り当てる
    - OMP_PLACES　コア番号を指定するなどして詳細に割り当てる

- 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
    - verboseを加えておくとデバッグに便利
      - 例：`export KMP_AFFINITY=verbose`
  - NVIDIA(PGI)：MP_BIND, MP_BLIST
    - 使いたいコアの番号を指定する

- ノードグループAではIntelコンパイラかGNUコンパイラを使うことが多いと思うため、これらを中心にいくつか例を示す

OpenMPプログラムの実行例を考える

- ノード全体でOpenMPプログラムを実行する場合、基本的には各NUMAノードに同程度の数のスレッドを配置して実行したいはずである
  - たくさんのメモリを使いたいためにノード単位でジョブを実行しているとしても、一部のNUMAノードだけ集中的に使う必要性は低いだろうと仮定
  - もしNUMAノードへの配置を全く気にしないのであれば、スレッド数のみ指定して実行すれば十分
  - この場合、どのNUMAノード・どのCPUコアから利用するかはシステム依存となる（制御不能）

```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=1
#PJM -L elapse=10:00
export OMP_NUM_THREADS=32
./a.out
```

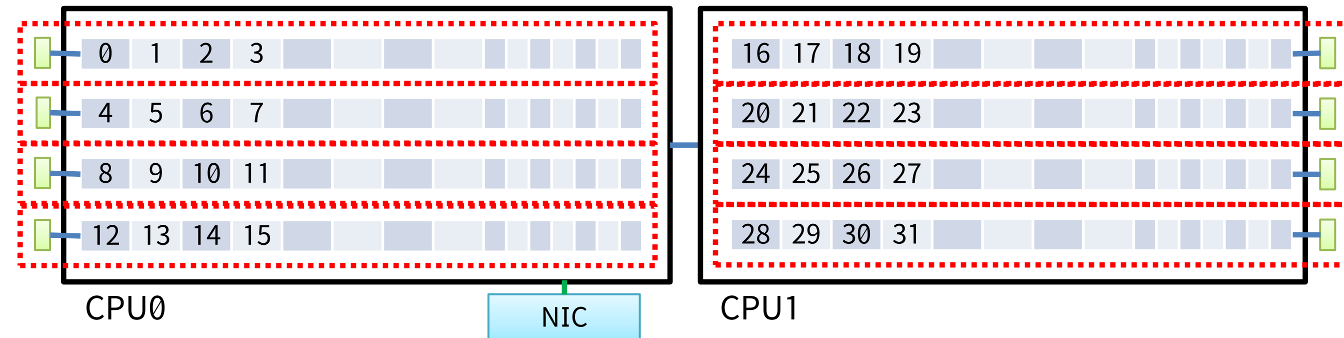
そもそもどのようなスレッド配置順が適切か

- 究極的には問題依存
- 一般的には……
  - スレッド数が少ない場合、メモリ性能を活かすためには全体に分散して配置、L3キャッシュを活かすには同一CPUソケットに配置するのが良い
  - スレッド数が多い場合、全体的に均等に配置するのが良い
    - 連続したスレッドIDを有するスレッドを近い位置に配置するか、連続したスレッドIDを有するスレッドを分散配置するか、の選択肢がある
- 近い位置に配置するか分散配置するかを考える際に、CPUソケットやNUMAノードを意識する必要がある

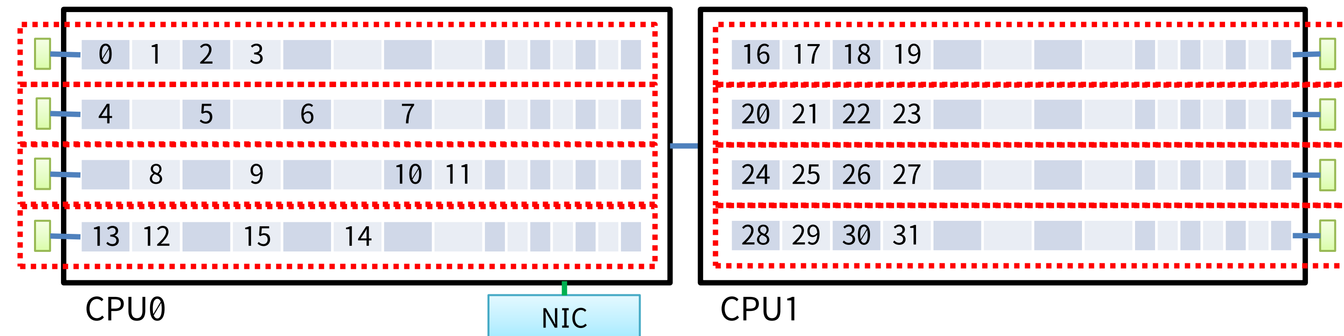
OpenMPプログラムの実行例：配置例1

- 配置例1
  - NUMAノードに必要数分のスレッドを配置してから、次のNUMAノードに配置する
  - 32スレッド実行を例に考える（全120コアに対して少なめだが、図で例示する手間の都合上）
    - 注意点：均等な距離にスレッドを配置するような指定ではうまくいかないことがある
      - export OMP_PROC_BIND=SPREAD　などではうまくいったりいかなかったりする
      - （120をスレッド数で割った際に余りが出ると配置がズレてしまうのが原因と思われる）

狙いたい配置イメージ→

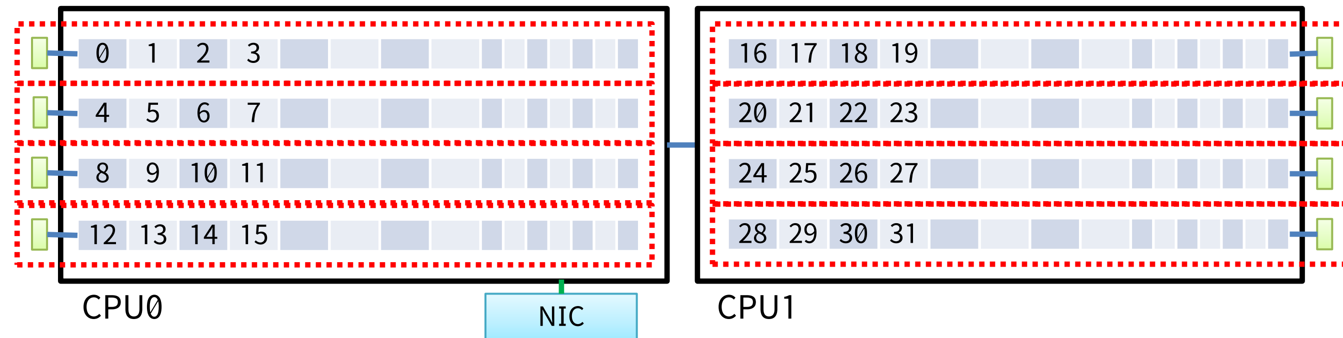


- NUMAノード内のスレッドの配置や順序は上図とある程度異なっていても良いし、NUMAノード毎に違っても構わない（有意な性能差は生じないことが多いはず）



OpenMPプログラムの実行例：配置例1　実行例1

- 実行例1：環境変数 OMP_PLACES で明示的に指定する
  - export OMP_PLACES="{0},{1},{2},{3},{15},{16},{17},{18},{30},{31},{32},{33},{45},{46},{47},{48},{60},{61},{62},{63},{75},{76},{77},{78},{90},{91},{92},{93},{105},{106},{107},{108}" ※改行せず一行で書く
  - Intelコンパイラ、GNUコンパイラ、NVIDIAコンパイラ共通で使える
  - {} で括ったコア番号にスレッドが1つずつ配置される
  - やや面倒だが確実で自由度も高い



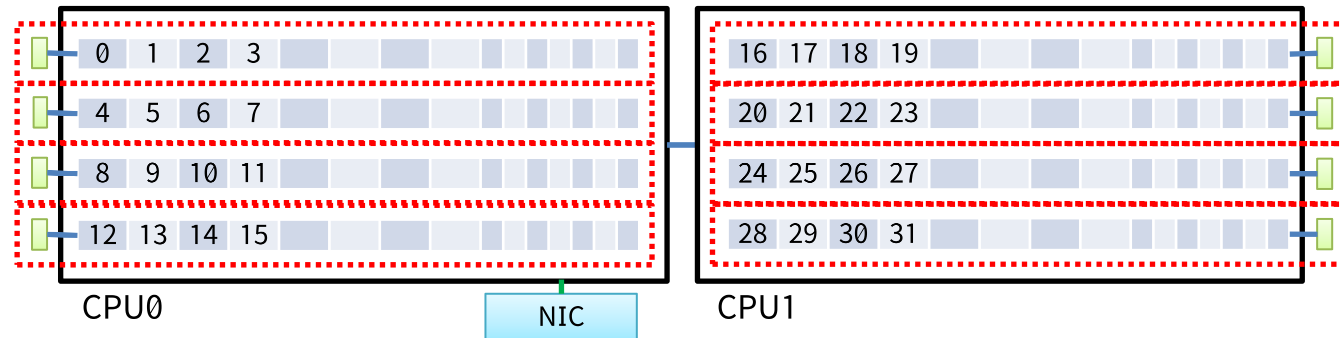

```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=1
#PJM -L elapse=10:00
必要に応じてmodule load
export OMP_NUM_THREADS=32
export OMP_PLACES="{0},{1},{2},{3},{15},{16},{17},{18},{30},{31},{32},{33},{45},{46},{47},{48},{60},{61},{62},{63},{75},{76},{77},{78},{90},{91},{92},{93},{105},{106},{107},{108}"
numactl --localalloc ./a.out
```

スレッド0から順に、OMP_PLACESで指定された番号のCPUコアに割り当てられる

スレッドID	0-3	4-7	8-11	12-15	16-19	20-23	24-27	28-31
CPUコアID	0,1,2,3	15,16,17,18	30,31,32,33	45,46,47,48	60,61,62,63	75,76,77,78	90,91,92,93	105,106,107,108
NUMAノード番号	0	1	2	3	4	5	6	7

OpenMPプログラムの実行例：配置例1　実行例2

- 実行例2：環境変数 GOMP_CPU_AFFINITY で明示的に指定する
  - export GOMP_CPU_AFFINITY="0-3,15-18,30-33,45-48,60-63,75-78,90-93,105-108"
    - 0,1,2,3,15,16,17,18,30,31,32,33,45,46,47,48,60,61,62,63,75,76,77,78,90,91,92,93,105,106,107,108 のようにべた書きしても同様
  - Intelコンパイラと GNUコンパイラで共通して使える
  - IntelコンパイラではKMP_AFFINITYで書いても良い（内部的にはGOMP~はKMP~のエイリアス）
  - export KMP_AFFINITY=verbose,granularity=fine,proclist=[0-3,15-18,30-33,45-48,60-63,75-78,90-93,105-108],explicit




```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=1
#PJM -L elapse=10:00
module load intel
export OMP_NUM_THREADS=32
export GOMP_CPU_AFFINITY="0-3,15-18,30-33,45-48,60-63,75-78,90-93,115-118"
export KMP_AFFINITY=verbose
numactl --localalloc ./a.out
```

スレッド0から順に、GOMP_CPU_AFFINITYで指定された番号のCPUコアに割り当てられる

スレッドID	0-3	4-7	8-11	12-15	16-19	20-23	24-27	28-31
CPUコアID	0,1,2,3	15,16,17,18	30,31,32,33	45,46,47,48	60,61,62,63	75,76,77,78	90,91,92,93	105,106,107,108
NUMAノード番号	0	1	2	3	4	5	6	7

KMP_AFFINITYにverboseを指定した場合の出力例 1

```
export OMP_NUM_THREADS=32
export OMP_PLACES="{0},{1},{2},{3},{15},{16},{17},{18},{30},{31},{32},{33},{45},{46},{47},{48},{60},{61},{62},{63},{75},{76},{77},{78},{90},{91},{92},{93},{105},{106},{107},{108}"
export KMP_AFFINITY=verbose
numactl --localalloc ./a.out
```

の出力例

```
OMP: Info #172: KMP_AFFINITY: OS proc 0 maps to socket 0 core 0 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 1 maps to socket 0 core 1 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 2 maps to socket 0 core 2 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 3 maps to socket 0 core 3 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 4 maps to socket 0 core 4 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 5 maps to socket 0 core 5 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 6 maps to socket 0 core 6 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 7 maps to socket 0 core 7 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 8 maps to socket 0 core 8 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 9 maps to socket 0 core 9 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 10 maps to socket 0 core 10 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 11 maps to socket 0 core 11 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 12 maps to socket 0 core 12 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 13 maps to socket 0 core 13 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 14 maps to socket 0 core 14 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 15 maps to socket 0 core 15 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 16 maps to socket 0 core 16 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 17 maps to socket 0 core 17 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 18 maps to socket 0 core 18 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 19 maps to socket 0 core 19 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 20 maps to socket 0 core 20 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 21 maps to socket 0 core 21 thread 0
……
```

「proc N」が「コア番号N」に対応しているというシンプルな関係性

```
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 72 thread 0 bound to OS proc set 0
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 73 thread 1 bound to OS proc set 1
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 74 thread 2 bound to OS proc set 2
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 75 thread 3 bound to OS proc set 3
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 76 thread 4 bound to OS proc set 15
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 77 thread 5 bound to OS proc set 16
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 78 thread 6 bound to OS proc set 17
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 79 thread 7 bound to OS proc set 18
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 80 thread 8 bound to OS proc set 30
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 81 thread 9 bound to OS proc set 31
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 82 thread 10 bound to OS proc set 32
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 83 thread 11 bound to OS proc set 33
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 84 thread 12 bound to OS proc set 45
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 85 thread 13 bound to OS proc set 46
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 86 thread 14 bound to OS proc set 47
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 87 thread 15 bound to OS proc set 48
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 88 thread 16 bound to OS proc set 60
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 89 thread 17 bound to OS proc set 61
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 90 thread 18 bound to OS proc set 62
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 91 thread 19 bound to OS proc set 63
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 92 thread 20 bound to OS proc set 75
OMP: Info #255: OMP_PROC_BIND: pid 72 tid 93 thread 21 bound to OS proc set 76
……
```

スレッド0がproc set 0 = コア番号0上で実行される
スレッド4がproc set 15 = コア番号15上で実行される　といった関係性がわかる

KMP_AFFINITYにverboseを指定した場合の出力例 2

```
export OMP_NUM_THREADS=32
export GOMP_CPU_AFFINITY="0-3,15-18,30-33,45-48,60-63,75-78,90-93,115-118"
export KMP_AFFINITY=verbose
numactl --localalloc ./a.out
```

の出力例

```
OMP: Info #172: KMP_AFFINITY: OS proc 0 maps to socket 0 core 0 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 1 maps to socket 0 core 1 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 2 maps to socket 0 core 2 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 3 maps to socket 0 core 3 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 4 maps to socket 0 core 4 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 5 maps to socket 0 core 5 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 6 maps to socket 0 core 6 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 7 maps to socket 0 core 7 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 8 maps to socket 0 core 8 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 9 maps to socket 0 core 9 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 10 maps to socket 0 core 10 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 11 maps to socket 0 core 11 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 12 maps to socket 0 core 12 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 13 maps to socket 0 core 13 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 14 maps to socket 0 core 14 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 15 maps to socket 0 core 15 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 16 maps to socket 0 core 16 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 17 maps to socket 0 core 17 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 18 maps to socket 0 core 18 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 19 maps to socket 0 core 19 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 20 maps to socket 0 core 20 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 21 maps to socket 0 core 21 thread 0
……
```

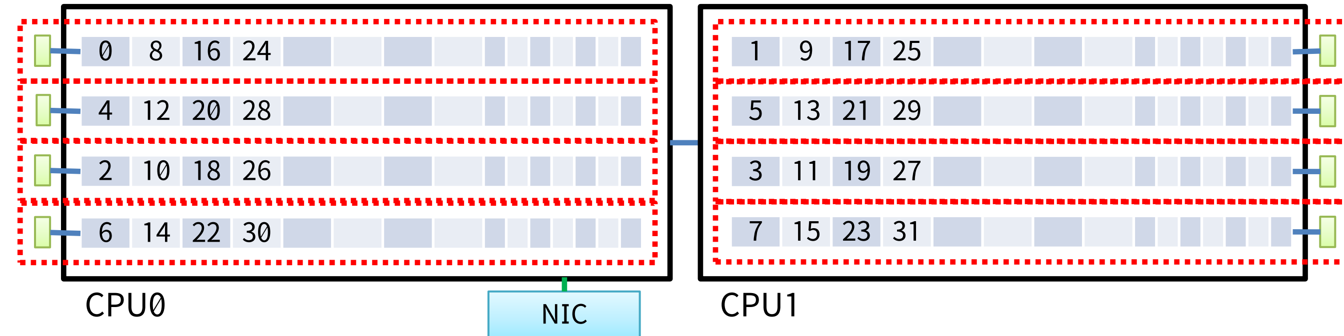
「proc N」が「コア番号N」に対応しているというシンプルな関係性

```
OMP: Info #255: KMP_AFFINITY: pid 74 tid 74 thread 0 bound to OS proc set 0
OMP: Info #255: KMP_AFFINITY: pid 74 tid 75 thread 1 bound to OS proc set 1
OMP: Info #255: KMP_AFFINITY: pid 74 tid 76 thread 2 bound to OS proc set 2
OMP: Info #255: KMP_AFFINITY: pid 74 tid 77 thread 3 bound to OS proc set 3
OMP: Info #255: KMP_AFFINITY: pid 74 tid 78 thread 4 bound to OS proc set 15
OMP: Info #255: KMP_AFFINITY: pid 74 tid 79 thread 5 bound to OS proc set 16
OMP: Info #255: KMP_AFFINITY: pid 74 tid 80 thread 6 bound to OS proc set 17
OMP: Info #255: KMP_AFFINITY: pid 74 tid 81 thread 7 bound to OS proc set 18
OMP: Info #255: KMP_AFFINITY: pid 74 tid 82 thread 8 bound to OS proc set 30
OMP: Info #255: KMP_AFFINITY: pid 74 tid 83 thread 9 bound to OS proc set 31
OMP: Info #255: KMP_AFFINITY: pid 74 tid 84 thread 10 bound to OS proc set 32
OMP: Info #255: KMP_AFFINITY: pid 74 tid 85 thread 11 bound to OS proc set 33
OMP: Info #255: KMP_AFFINITY: pid 74 tid 86 thread 12 bound to OS proc set 45
OMP: Info #255: KMP_AFFINITY: pid 74 tid 87 thread 13 bound to OS proc set 46
OMP: Info #255: KMP_AFFINITY: pid 74 tid 88 thread 14 bound to OS proc set 47
OMP: Info #255: KMP_AFFINITY: pid 74 tid 89 thread 15 bound to OS proc set 48
OMP: Info #255: KMP_AFFINITY: pid 74 tid 90 thread 16 bound to OS proc set 60
OMP: Info #255: KMP_AFFINITY: pid 74 tid 91 thread 17 bound to OS proc set 61
OMP: Info #255: KMP_AFFINITY: pid 74 tid 92 thread 18 bound to OS proc set 62
OMP: Info #255: KMP_AFFINITY: pid 74 tid 93 thread 19 bound to OS proc set 63
OMP: Info #255: KMP_AFFINITY: pid 74 tid 94 thread 20 bound to OS proc set 75
OMP: Info #255: KMP_AFFINITY: pid 74 tid 95 thread 21 bound to OS proc set 76
……
```

スレッド0がproc set 0 = コア番号0上で実行される
スレッド4がproc set 15 = コア番号15上で実行される　といった関係性がわかる

OpenMPプログラムの実行例：配置例2

- 実行例2
  - 各NUMAノードに1スレッドずつ配置し、配置し終えたら2スレッド目を配置する
  - やはり配置するコアを明示的に指定するのが確実




```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=1
#PJM -L elapse=10:00
必要に応じてmodule load
export OMP_NUM_THREADS=32

export
OMP_PLACES="{0},{60},{30},{90},{15},{75},{45},{105},{1},{61},{31},{91},{16},{76},{46},{106},{2},{62},{32},{92},{17},{77},{47},{
107},{3},{63},{33},{93},{18},{78},{48},{108}"

numactl --localalloc ./a.out
```

※ 改行せず一行で書く

OpenMPプログラムの実行例：補足1

- もっと簡単な指定方法はないのか？
  - 配置例1について
    - export OMP_PLACES="{0:4}:8:15" （32スレッドではない場合は4をNUMAあたり利用コア数（切り上げ）に変更）でも同様の配置ができるようだが、割りきれない数の場合などに気持ち悪い配置がされてしまうようなので利用時には注意（確認）が必要
  - 配置例2について
    - いまのところ不明
    - KMP_AFFINTYでscatterを使うなどが良さそうに思ったが、NUMAノード単位ではなくソケット単位で分散配置されてしまった
- 特定のNUMAノードやCPUソケット内にのみスレッドを配置したい場合は？
  - やはり OMP_PLACES や GOMP_CPU_AFFINITY でコア番号を明確に指定するのがシンプルで確実
  - numactlでNUMAノードやCPUコアを指定しても良い
    - CPUコアを指定すれば完全に同じ配置ができる（記述量もほぼ変わらない）
- Intelコンパイラ以外で（プログラム内から）配置を確認する方法は？
  - 資料末尾の参考情報を参照

OpenMPプログラムの実行例：補足2

- CPU内のNUMAの構成まで細かく考えなくて良いのであれば OMP_PROC_BINDでcloseやspread、KMP_AFFINITYでcompactやscatterを指定する程度でも良い

※ closeやspreadはgnuもintelも共通に利用可能。scatter的かつNUMAノードに均等に配置したい場合などはOMP_PLACESで細かく指定する必要があるようだ。

```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=1
#PJM -L elapse=10:00

export OMP_PROC_BIND=close
export OMP_PROC_BIND=spread

module load intel
export KMP_AFFINITY=verbose,granularity=fine,compact
export KMP_AFFINITY=verbose,granularity=fine,scatter
export KMP_AFFINITY=verbose,granularity=fine,balanced

export OMP_NUM_THREADS=60

numactl --localalloc ./a.out
```

- CPUコアを端から順に埋める
- CPUコアを端から全体的に均等に埋める（小さな番号のコアに小さな番号のスレッドが配置される）

- CPUコアを端から順に埋める
- scatterやbalancedではCPUソケット単位で均等な配置になってしまうようなので注意が必要（NUMAノード0と4に優先して配置されてしまう）

ノード共有実行の場合は？

- vnode-core<120 を指定して実行した、ノードの一部の資源のみが使えるジョブの場合
  - OpenMP実行においては単純にスレッド数だけ指定して実行するのが妥当
  - NUMAノードをまたいで計算資源が割り当たることもあるため、細かい指定についてはあまり考える必要がない（考えても仕方がない）と思われる

MPI（MPIのみ、またはMPI+OpenMP）プログラムの具体的な実行方法

MPIプログラムおよびMPI+OpenMPプログラムの実行を考える

- MPIプログラムの実行については利用するMPIの種類によって設定方法が異なるため、Intel MPIを使う場合とOpen MPIを使う場合とに分けて説明する

- 前提（問題設定）
  - 明確に狙いがある場合を除き、基本的にはCPUソケットやNUMAノードに対して均等にMPIプロセスを配置することが推奨されるため、本資料でもこれを対象問題設定とする

- 共通のTips
  - ノードグループAはCPU0側にのみNICが接続されているため、ノード毎にデータを集約してからノード間通信を行うようなプログラムを作成する場合はCPU0側にデータを集約するようなプロセス配置にした方が良い性能が期待できる
    - 本資料ではCPU0側のプロセスからMPIランク番号が始まるように意識するのみとする
  - 複数ノード実行の場合、ノードへのプロセス配置はMPIレベルでしか行えないため、MPIレベルでプロセスの配置を行った後で、必要に応じてnumactl等で調整すると良い
    - numactlではプロセスが実行されるCPUコアを調整できるが、ノードを変えることはできない

MPIプログラムの具体的な実行方法
1. Intel MPIを使う場合

Intel MPIを使う：基礎編　1/2

- 準備：module load intel impi
  - intel/2023.2　と　impi/2021.10.0
または
intel/2024.1　と　impi/2021.12
の組み合わせが利用可能（前者がdefault）
- MPIプログラムの実行はmpiexecで行う
- mpiexecへの引数やI_MPI_から始まる環境変数などを用いることで動作を制御可能
  - プロセス数は実行時オプション-n
  - ノードあたりプロセス数は実行時オプション-perhostや環境変数I_MPI_PERHOST
  - プロセスの配置方法は環境変数I_MPI_PIN_ORDERやI_MPI_PIN_DOMAIN、numactlの併用
- -print-rank-mapオプションを使うと簡易的なランク割り当て情報を確認可能
- 環境変数I_MPI_DEBUGを指定すれば詳細なランク割り当て情報を確認可能
  - 適切な割り当てができているかわからない際はI_MPI_DEBUG=5で実行することを推奨

Intel MPIを使う：基礎編　2/2

- pjsubのオプションでも動作を制御可能
  - 利用の手引きでは #PJM -L node=ノード数、#PJM --mpi proc=プロセス数 の指定と mpiexec -n $PJM_MPI_PROC を用いた実行例が示されているが、必須なのはノード数のみ
  - 実際に起動するプロセス数を左右するのはmpiexecの-nオプション
  - --mpi procによるプロセス数はノード数と乗算して$PJM_MPI_PROCを算出するために使われる
  - #PJM --mpi proc=プロセス数 や $PJM_MPI_PROC を使わずに自分でプロセス数を指定してもよい

Intel MPIを使う：出力ファイルの制御

- 標準出力・エラー出力をプロセスごとに別のファイルに書き出すには？
  - -outfile-pattern 引数と -errfile-pattern 引数で出力ファイルを変更可能
    - ファイル名そのものを指定する。%r でランク番号を使える。
  - 例：TCS（バッチジョブシステム）のジョブIDとMPIのランク番号を用いて重複しない名前のファイルに出力

```
mpiexec -print-rank-map ¥
 -outfile-pattern out_${PJM_JOBID}_%r.txt ¥
 -errfile-pattern err_${PJM_JOBID}_%r.txt ¥
 -n 4 numactl -l ./a.out
```

- ジョブスクリプト（bashスクリプト）は行末にバックスラッシュ（PowerPoint上では表示の都合で円記号）を書くことで行の折り返しが可能

Intel MPIによる1ノード内複数プロセス実行：基本的な考え方

- プロセスの配置方法は主にI_MPI_PIN_ORDER環境変数とI_MPI_PIN_DOMAIN環境変数で制御、numactlの併用も可能（環境変数で十分なはず）
  - I_MPI_PIN_ORDER：全体的なプロセス配置の方法をどのようにするかを指定
    - ノードの一部に寄せるのか、全体的に均等に配置するのか、など
  - I_MPI_PIN_DOMAIN：1プロセスあたり何コアに割り当てるのかを指定
    - MPIとOpenMPを用いたハイブリッド並列化を行う際はこれにプロセスあたりスレッド数を指定する
      - OMP_NUM_THREADSの値を自動的に使ってくれるompや、総コア数をプロセス数で除算した値を参照するautoも有用

例：32プロセスフラットMPI

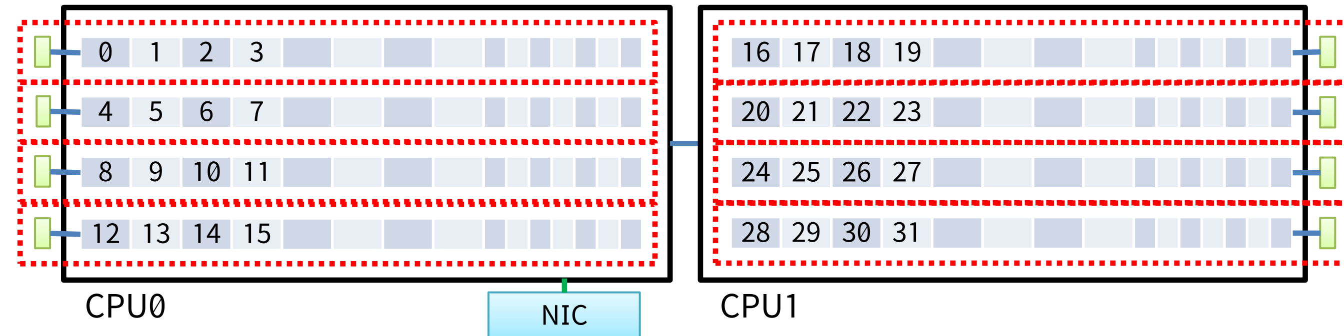
- 基本的に推奨：8つのNUMAノードに4プロセスずつ配置し、各プロセスがNUMAノード内のメモリにアクセスする
  - プロセス数がより少ない場合でも多い場合でも、全NUMAノードに均等に配置するのが基本
- 基本的には非推奨：片方のCPUソケットに全32プロセスを配置
  - I_MPI_PIN_ORDERにcompactを指定して32プロセス実行するとこうなる
  - プロセス同士が近いため通信性能に優れるが、そもそも1ノード全て使う意味がない
    - 多くのメモリ容量が欲しいためあえて1ノード占有する場合も、両ソケットに分散配置した方が全体的には高いメモリアクセス性能が期待しやすい

- 主なプロセス配置戦略
  - 1. NUMAノードあたり4プロセス配置してから次のNUMAノードへプロセスを配置する
    - 偏らない程度に端から詰めていくようなイメージ
  - 2. まず全NUMAノードに1プロセス配置し、次にまた全NUMAノードに1プロセス配置する
    - 全体的に均等に配置していくイメージ
  - どちらが良いかは、プログラム内のキャッシュの利用具合や、プロセス間の通信パターン次第

偏らない程度に端から配置する

- I_MPI_PIN_ORDERにspreadやbunchを指定すると、各NUMAノードに同数の連続したランクのプロセスが割り当てられ、NUMAノード内のプロセスは連続するCPUコアに配置される
  - 本環境においてはspreadとbunchは同じ挙動になる

```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=1
#PJM -L elapse=10:00
module load intel
module load impi
export I_MPI_DEBUG=5
export I_MPI_PIN_ORDER=spread
export I_MPI_PIN_DOMAIN=1
mpiexec -print-rank-map -n 32 numactl -l ./a.out
```



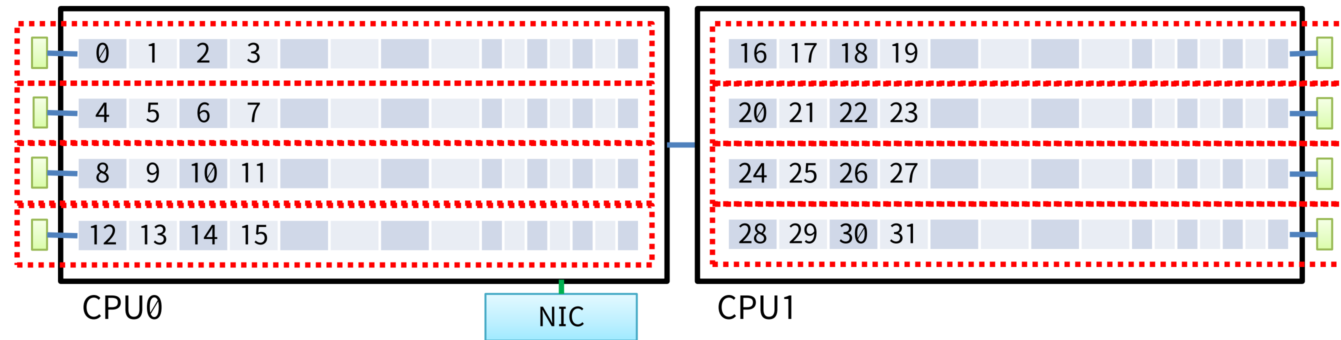
I_MPI_DEBUG=5 指定時に得られる出力（割り当て情報）の一部

```
[0] MPI startup(): Rank    Pid      Node name  Pin cpu
[0] MPI startup(): 0       124      a0001      {45}
[0] MPI startup(): 1       125      a0001      {46}
[0] MPI startup(): 2       126      a0001      {47}
[0] MPI startup(): 3       127      a0001      {48}
[0] MPI startup(): 4       128      a0001      {0}
[0] MPI startup(): 5       129      a0001      {1}
[0] MPI startup(): 6       130      a0001      {2}
[0] MPI startup(): 7       131      a0001      {3}
[0] MPI startup(): 8       132      a0001      {15}
[0] MPI startup(): 9       133      a0001      {16}
[0] MPI startup(): 10      134      a0001      {17}
[0] MPI startup(): 11      135      a0001      {18}
[0] MPI startup(): 12      136      a0001      {30}
[0] MPI startup(): 13      137      a0001      {31}
[0] MPI startup(): 14      138      a0001      {32}
[0] MPI startup(): 15      139      a0001      {33}
[0] MPI startup(): 16      140      a0001      {60}
[0] MPI startup(): 17      141      a0001      {61}
[0] MPI startup(): 18      142      a0001      {62}
[0] MPI startup(): 19      143      a0001      {63}
[0] MPI startup(): 20      144      a0001      {75}
[0] MPI startup(): 21      145      a0001      {76}
[0] MPI startup(): 22      146      a0001      {77}
[0] MPI startup(): 23      147      a0001      {78}
[0] MPI startup(): 24      148      a0001      {90}
[0] MPI startup(): 25      149      a0001      {91}
[0] MPI startup(): 26      150      a0001      {92}
[0] MPI startup(): 27      151      a0001      {93}
[0] MPI startup(): 28      152      a0001      {105}
[0] MPI startup(): 29      153      a0001      {106}
[0] MPI startup(): 30      154      a0001      {107}
[0] MPI startup(): 31      155      a0001      {108}
```

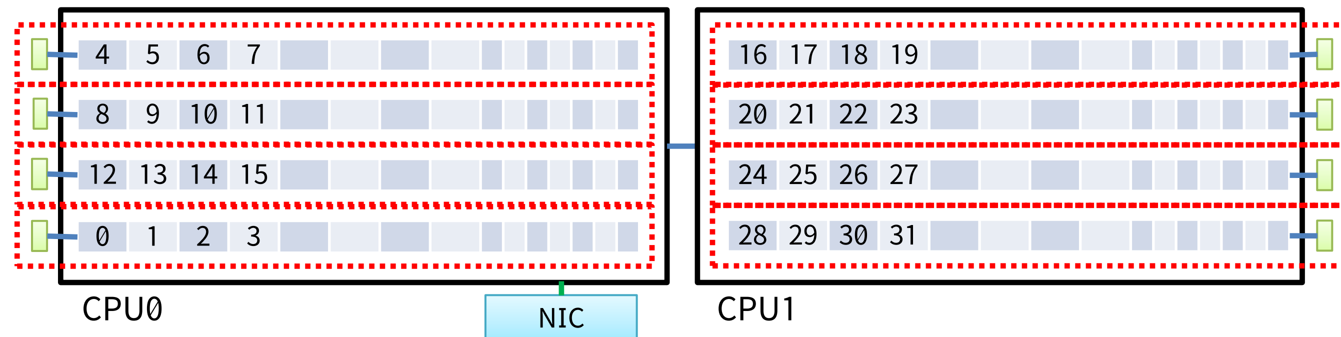
- 関係する情報を切り出してソートしたもの
- 各プロセスがどこに配置されたのかが一目で確認できる

- 何故かCPUソケット0側は0から始まらずに45から始まっているが、割り当てポリシーとしては問題ないため気にしないことにする

想定→



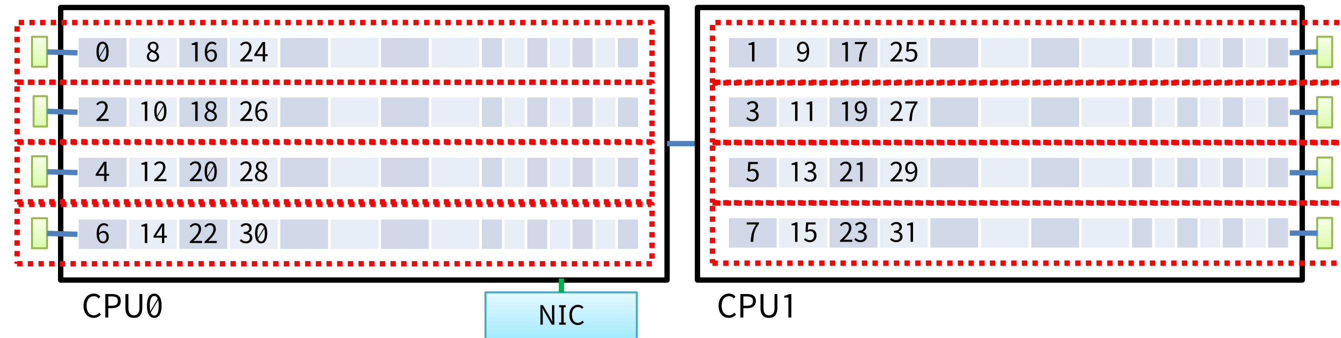
実際→



全体的に均等に配置する

- I_MPI_PIN_ORDERにscatterを指定すると、各NUMAノードに同数のプロセスが割り当てられるが、連続したランクのプロセスは異なるNUMAノードに配置される
  - 全体的に分散して配置：あるNUMAノードに1プロセスを配置したら次は別のNUMAノードへ
  - spread/bunchとscatterのどちらが良いかはプログラム内の通信次第

```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=1
#PJM -L elapse=10:00
module load intel
module load impi
export I_MPI_DEBUG=5
export I_MPI_PIN_ORDER=scatter
export I_MPI_PIN_DOMAIN=1
mpiexec -print-rank-map -n 32 numactl -l ./a.out
```



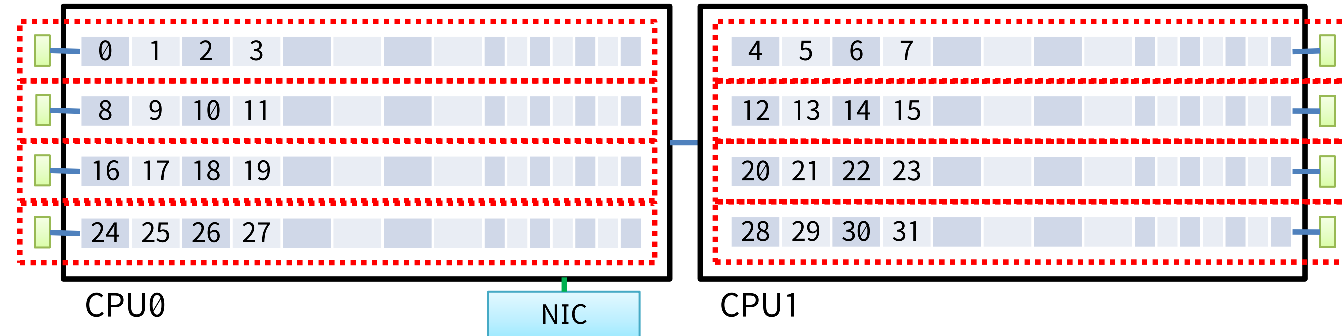
I_MPI_DEBUG=5 指定時に得られる出力（割り当て情報）の一部

```
[0] MPI startup(): Rank    Pid      Node name  Pin cpu
[0] MPI startup(): 0       124      a0003      {0}
[0] MPI startup(): 1       125      a0003      {60}
[0] MPI startup(): 2       126      a0003      {15}
[0] MPI startup(): 3       127      a0003      {75}
[0] MPI startup(): 4       128      a0003      {30}
[0] MPI startup(): 5       129      a0003      {90}
[0] MPI startup(): 6       130      a0003      {45}
[0] MPI startup(): 7       131      a0003      {105}
[0] MPI startup(): 8       132      a0003      {1}
[0] MPI startup(): 9       133      a0003      {61}
[0] MPI startup(): 10      134      a0003      {16}
[0] MPI startup(): 11      135      a0003      {76}
[0] MPI startup(): 12      136      a0003      {31}
[0] MPI startup(): 13      137      a0003      {91}
[0] MPI startup(): 14      138      a0003      {46}
[0] MPI startup(): 15      139      a0003      {106}
[0] MPI startup(): 16      140      a0003      {2}
[0] MPI startup(): 17      141      a0003      {62}
[0] MPI startup(): 18      142      a0003      {17}
[0] MPI startup(): 19      143      a0003      {77}
[0] MPI startup(): 20      144      a0003      {32}
[0] MPI startup(): 21      145      a0003      {92}
[0] MPI startup(): 22      146      a0003      {47}
[0] MPI startup(): 23      147      a0003      {107}
[0] MPI startup(): 24      148      a0003      {3}
[0] MPI startup(): 25      149      a0003      {63}
[0] MPI startup(): 26      150      a0003      {18}
[0] MPI startup(): 27      151      a0003      {78}
[0] MPI startup(): 28      152      a0003      {33}
[0] MPI startup(): 29      153      a0003      {93}
[0] MPI startup(): 30      154      a0003      {48}
[0] MPI startup(): 31      155      a0003      {108}
```

- 関係する情報を切り出してソートしたもの
- 各プロセスがどこに配置されたのかが一目で確認できる

その他の配置方法（完全に思い通りに配置するには？）

- I_MPI_PIN_DOMAIN等による指定が気に入らない場合はどうするべきか
  - 例：NUMA単位で均等に分散させるには？
  - I_MPI_PIN_PROCESSOR_LIST で指定すれば確実（やや面倒ではある）

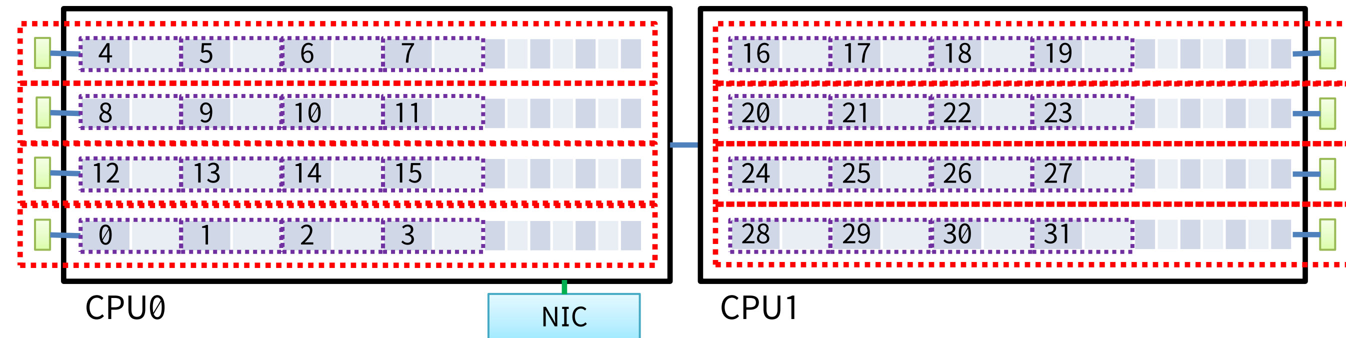



```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=1
#PJM -L elapse=10:00
module load intel
module load impi
export I_MPI_DEBUG=5
export I_MPI_PIN_PROCESSOR_LIST="0,1,2,3,15,16,17,18,30,31,32,33,45,46,47,48,60,61,62,63,75,76,77,78,90,91,92,93,105,106,107,108"
mpiexec -print-rank-map -n 32 numactl -l -s
```

MPI + OpenMP ハイブリッド並列化

- OpenMPスレッド並列化と組み合わせる場合、一般的には同一プロセス内のスレッドを連続したCPUコアに割り当てるのが良い
- OMP_NUM_THREADSとI_MPI_PIN_DOMAINの指定によりプロセスの配置方法を指定する
  - I_MPI_PIN_DOMAINに「OMP_NUM_THREADSの値」または「omp」を指定する
  - プロセス配置時に連続する複数コアが確保され、その中でOpenMP並列化が行われる

```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=1
#PJM -L elapse=10:00
module load intel
module load impi
export I_MPI_DEBUG=5
export I_MPI_PIN_ORDER=spread
export I_MPI_PIN_DOMAIN=omp
export OMP_NUM_THREADS=2
mpiexec -print-rank-map -n 32 numactl -l ./a.out
```



複数ノード実行

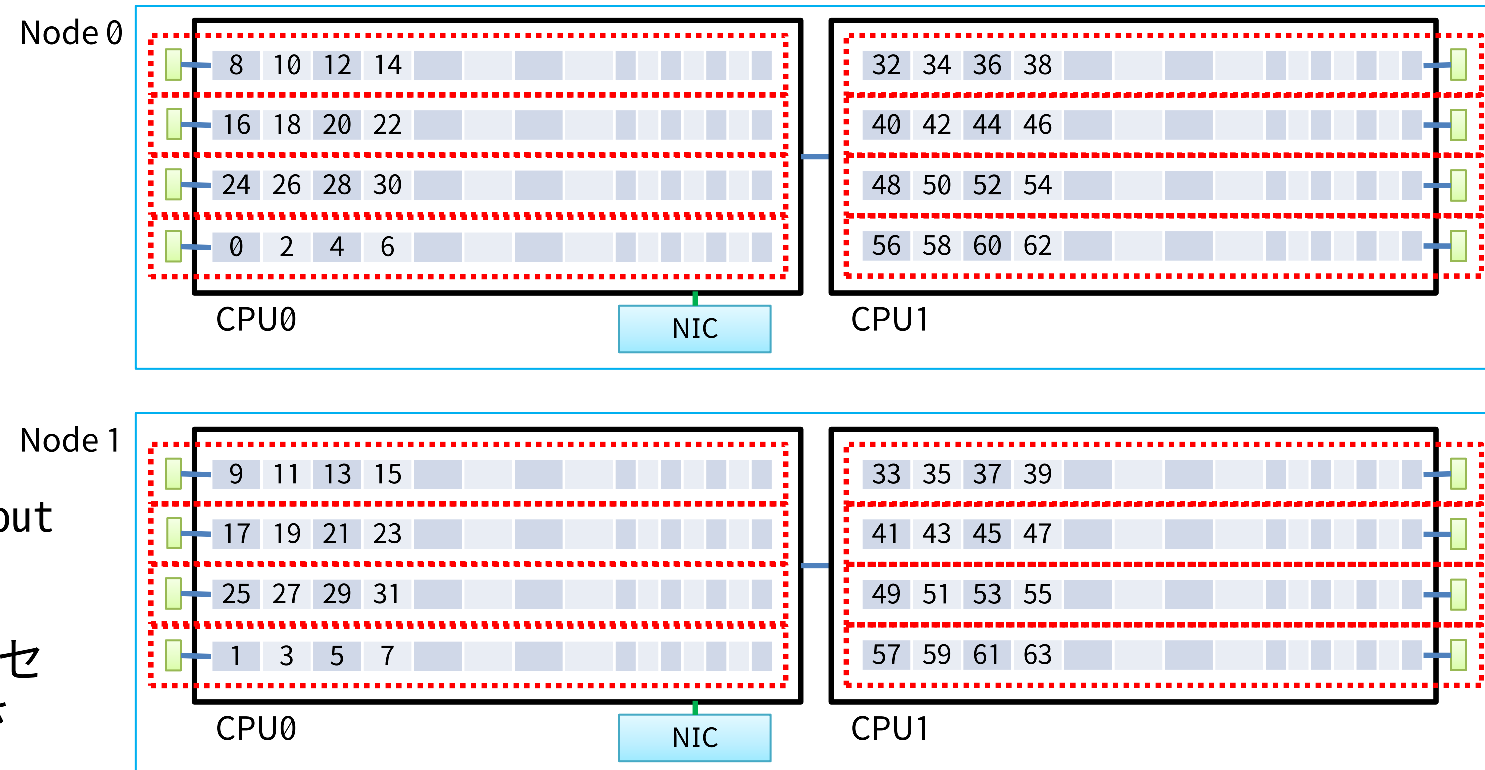
- #PJM –L node=ノード数 により利用ノード数を指定（必須）
- 引数-ppnや環境変数I_MPI_PERHOSTを指定すると、指定した数のプロセスを配置してから次のノードへプロセスを配置する、という挙動になる
- 様々な配置が可能であり、対象プログラムにとって適切な配置になるよう設定する必要がある

例：32プロセス×2ノード（合計64プロセス）、フラットMPI　1/3

- ノードあたりプロセス数を指定しない場合、プロセスごとに異なるノードにプロセスが配置される
- I_MPI_PIN_ORDERにspreadを指定した場合（前述の例を単純に2ノード化した場合）の例

```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load intel
module load impi
export I_MPI_DEBUG=5
export I_MPI_PIN_ORDER=spread
export I_MPI_PIN_DOMAIN=1
mpiexec -print-rank-map -n 64 numactl -l ./a.out
```

- I_MPI_PIN_ORDER=scatterの場合もプロセス単位でノード0とノード1に交互に配置される



-print-rank-map 指定時に得られる出力（割り当て情報）

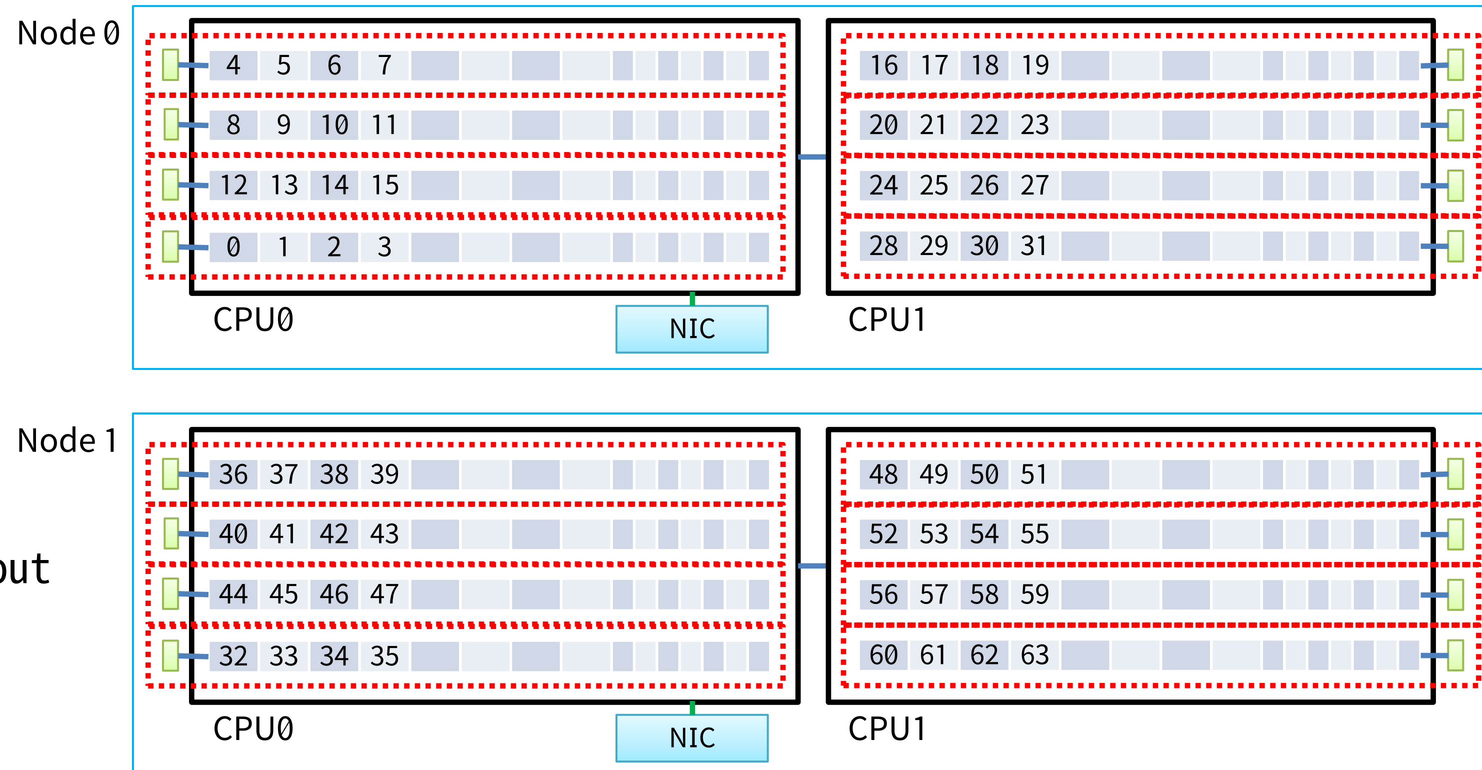
- 表示される情報がシンプル過ぎるため、あまり意味はないかもしれない
  - ホストのIPアドレスとそこに割り当てられたランク番号のみ

```
(172.16.1.1:0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62)
(172.16.1.2:1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55,57,59,61,63)
```

例：32プロセス×2ノード（合計64プロセス）、フラットMPI　2/3

- ノードあたりプロセス数として32を明示すると、ノード0に32プロセス配置し終えてからノード1に配置するようになる
  - （前述の例よりもこちらのほうが期待される挙動？）

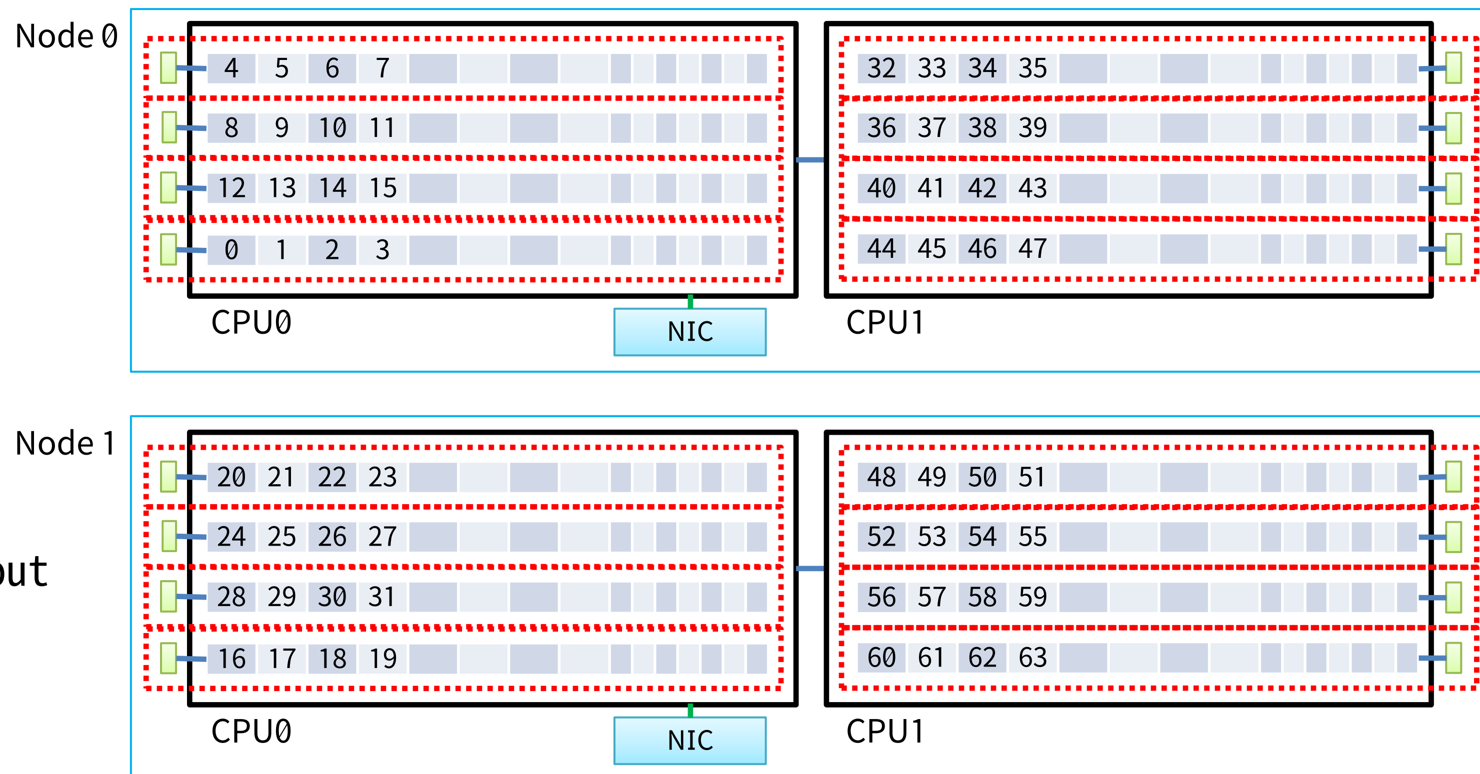
```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load intel
module load impi
export I_MPI_DEBUG=5
export I_MPI_PIN_ORDER=spread
export I_MPI_PIN_DOMAIN=1
export I_MPI_PERHOST=32
mpiexec -print-rank-map -n 64 numactl -l ./a.out
```



例：32プロセス×2ノード（合計64プロセス）、フラットMPI　3/3

- ノードあたりプロセス数を16に減らすと、ノード0に16プロセス配置し終えた時点でノード1に配置するようになる
  – 結果的にソケット単位でノードが変わる（これも期待される挙動と思われる）

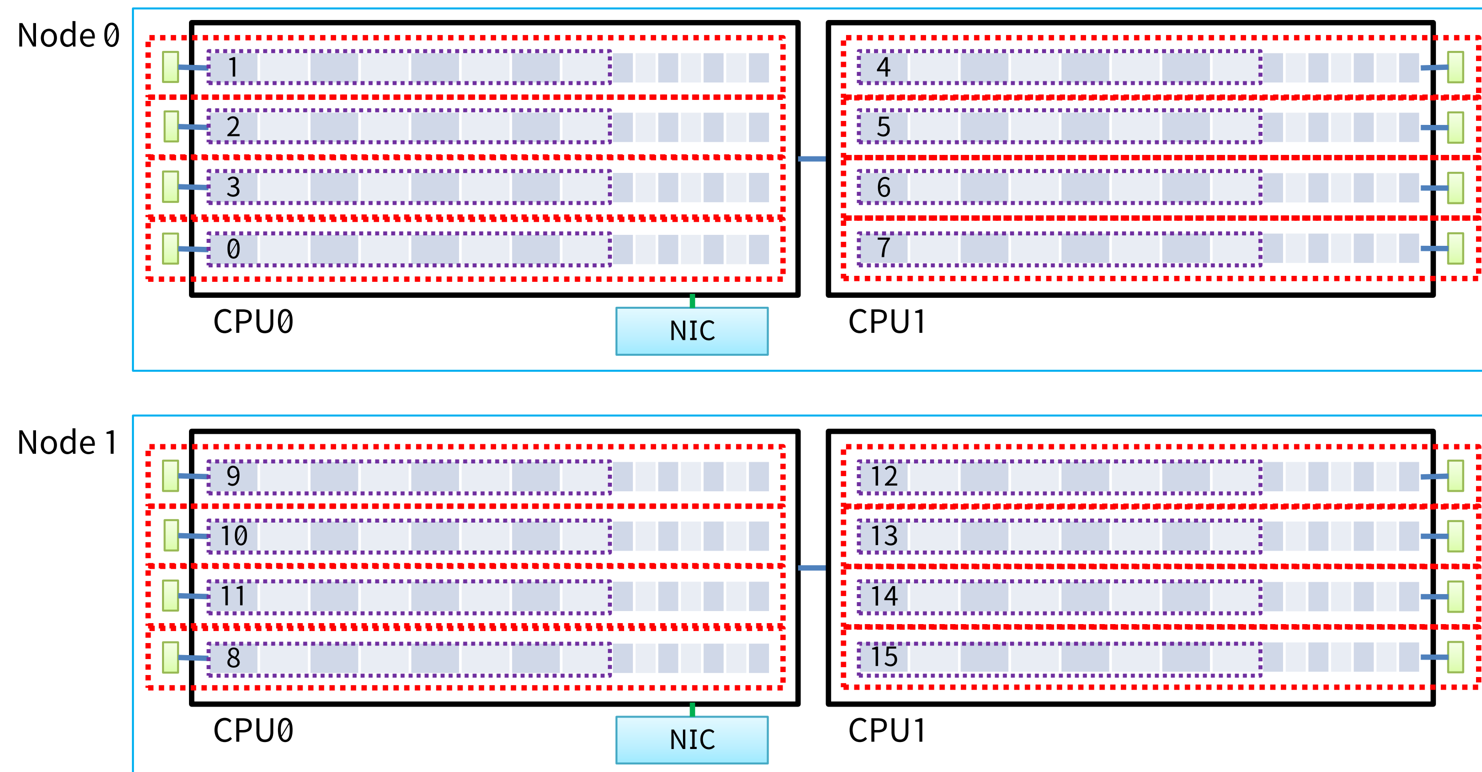
```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load intel
module load impi
export I_MPI_DEBUG=5
export I_MPI_PIN_ORDER=spread
export I_MPI_PIN_DOMAIN=1
export I_MPI_PERHOST=16
mpiexec -print-rank-map -n 64 numactl -l ./a.out
```



複数ノード、OpenMP+MPIハイブリッド並列化　1/3

- 単一ノードの場合と同様に、I_MPI_PIN_DOMAINを組み合わせて指定する
- 例：8プロセス×2ノード、プロセスあたり8スレッド並列化
  - NUMAノードあたり1プロセスを配置し、NUMAノード内で2のべき乗最大のスレッド数を実行（HW構成にあう自然な実行形態の1つと考えられる）

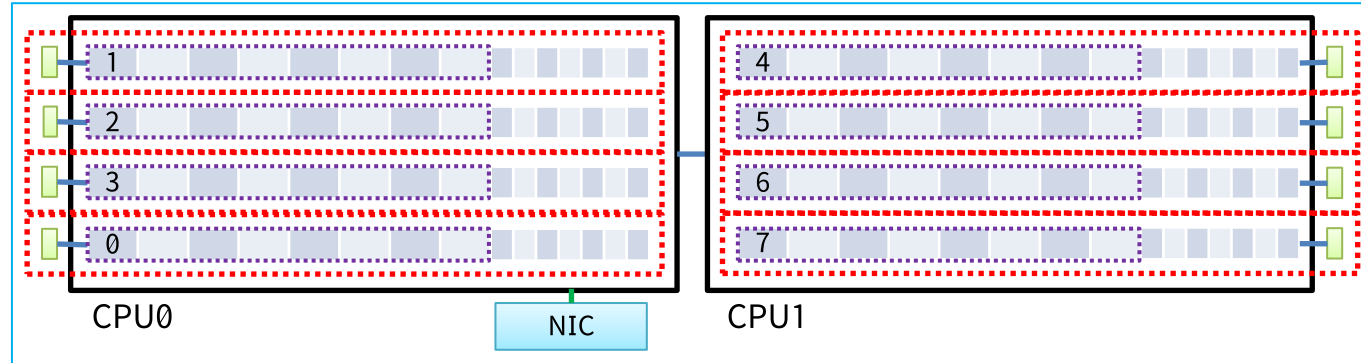
```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load intel
module load impi
export I_MPI_DEBUG=5
export I_MPI_PIN_ORDER=spread
export I_MPI_PIN_DOMAIN=omp
export I_MPI_PERHOST=8
export OMP_NUM_THREADS=8
export KMP_AFFINITY=verbose
mpiexec -print-rank-map -n 16 ./a.out
```



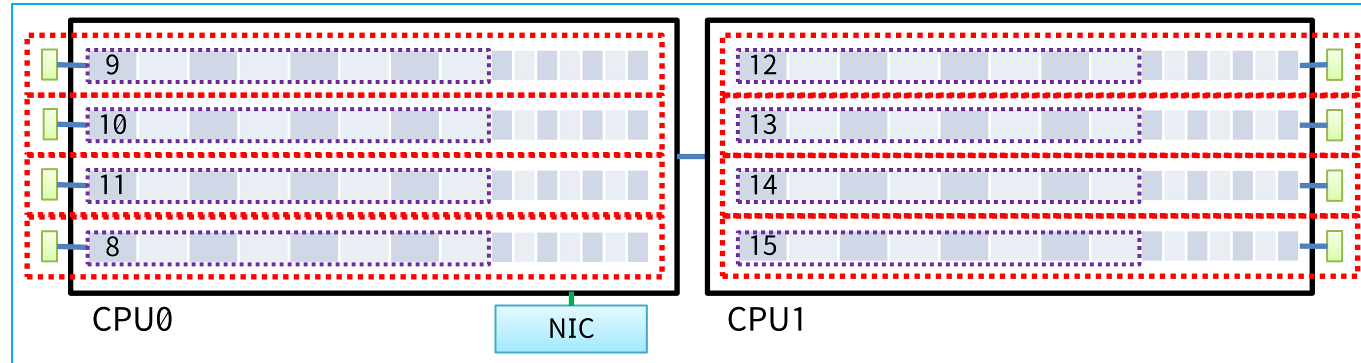
実行結果（I_MPI_DEBUG=5 情報）

```
[0] MPI startup(): Rank  Pid  Node name Pin cpu

[0] MPI startup(): 0   102  a0001  {45,46,47,48,49,50,51,52}
[0] MPI startup(): 1   103  a0001  {0,1,2,3,4,5,6,7}
[0] MPI startup(): 2   104  a0001  {15,16,17,18,19,20,21,22}
[0] MPI startup(): 3   105  a0001  {30,31,32,33,34,35,36,37}
[0] MPI startup(): 4   106  a0001  {60,61,62,63,64,65,66,67}
[0] MPI startup(): 5   107  a0001  {75,76,77,78,79,80,81,82}
[0] MPI startup(): 6   108  a0001  {90,91,92,93,94,95,96,97}
[0] MPI startup(): 7   109  a0001  {105,106,107,108,109,110,111,112}

[0] MPI startup(): 8   20   a0002  {45,46,47,48,49,50,51,52}
[0] MPI startup(): 9   21   a0002  {0,1,2,3,4,5,6,7}
[0] MPI startup(): 10  22   a0002  {15,16,17,18,19,20,21,22}
[0] MPI startup(): 11  23   a0002  {30,31,32,33,34,35,36,37}
[0] MPI startup(): 12  24   a0002  {60,61,62,63,64,65,66,67}
[0] MPI startup(): 13  25   a0002  {75,76,77,78,79,80,81,82}
[0] MPI startup(): 14  26   a0002  {90,91,92,93,94,95,96,97}
[0] MPI startup(): 15  27   a0002  {105,106,107,108,109,110,111,112}
```

Node 0



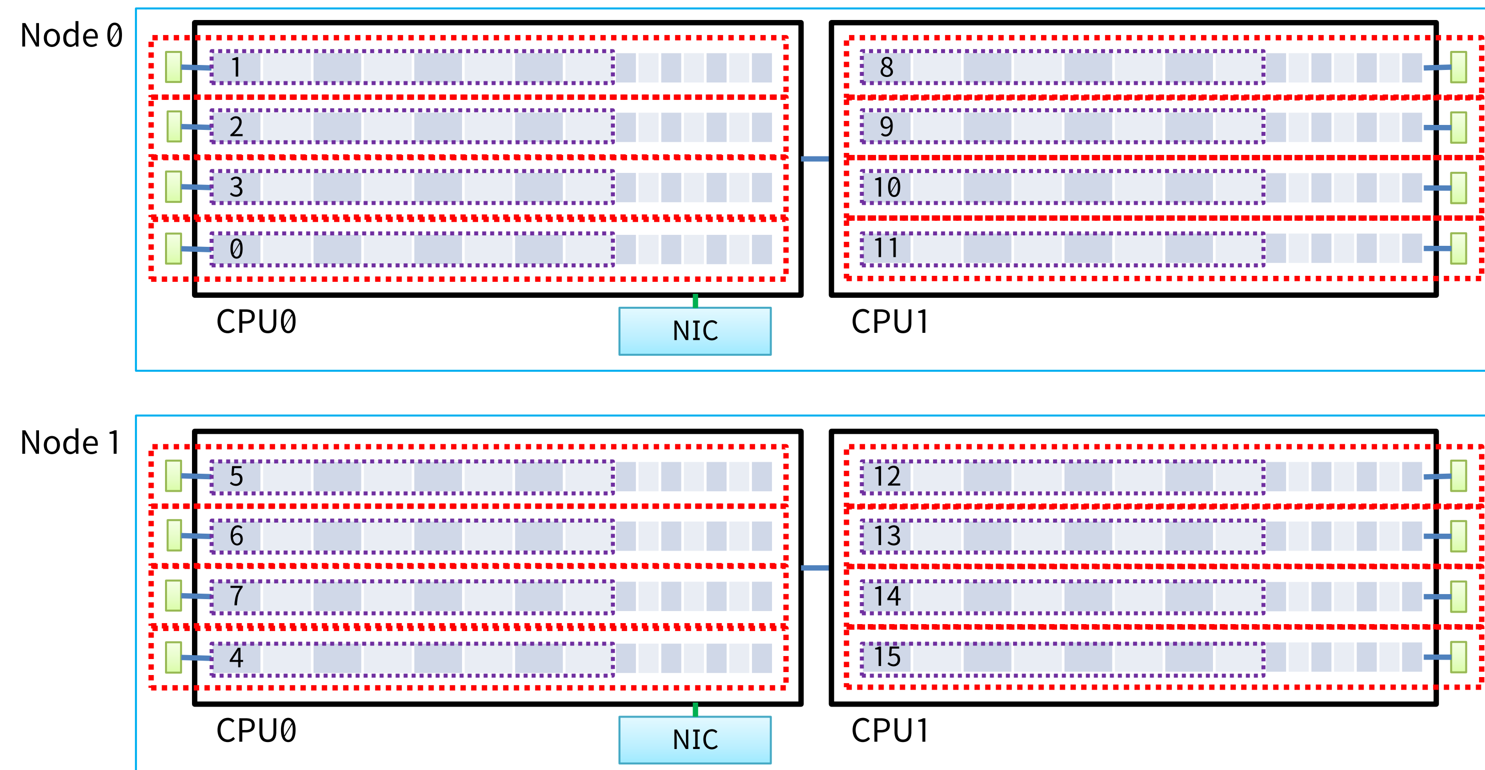
Node 1



複数ノード、OpenMP+MPIハイブリッド並列化　2/3

- I_MPI_PERHOST=4 にすればノード0のCPUソケット0の次にノード1のCPU0を使うようになる

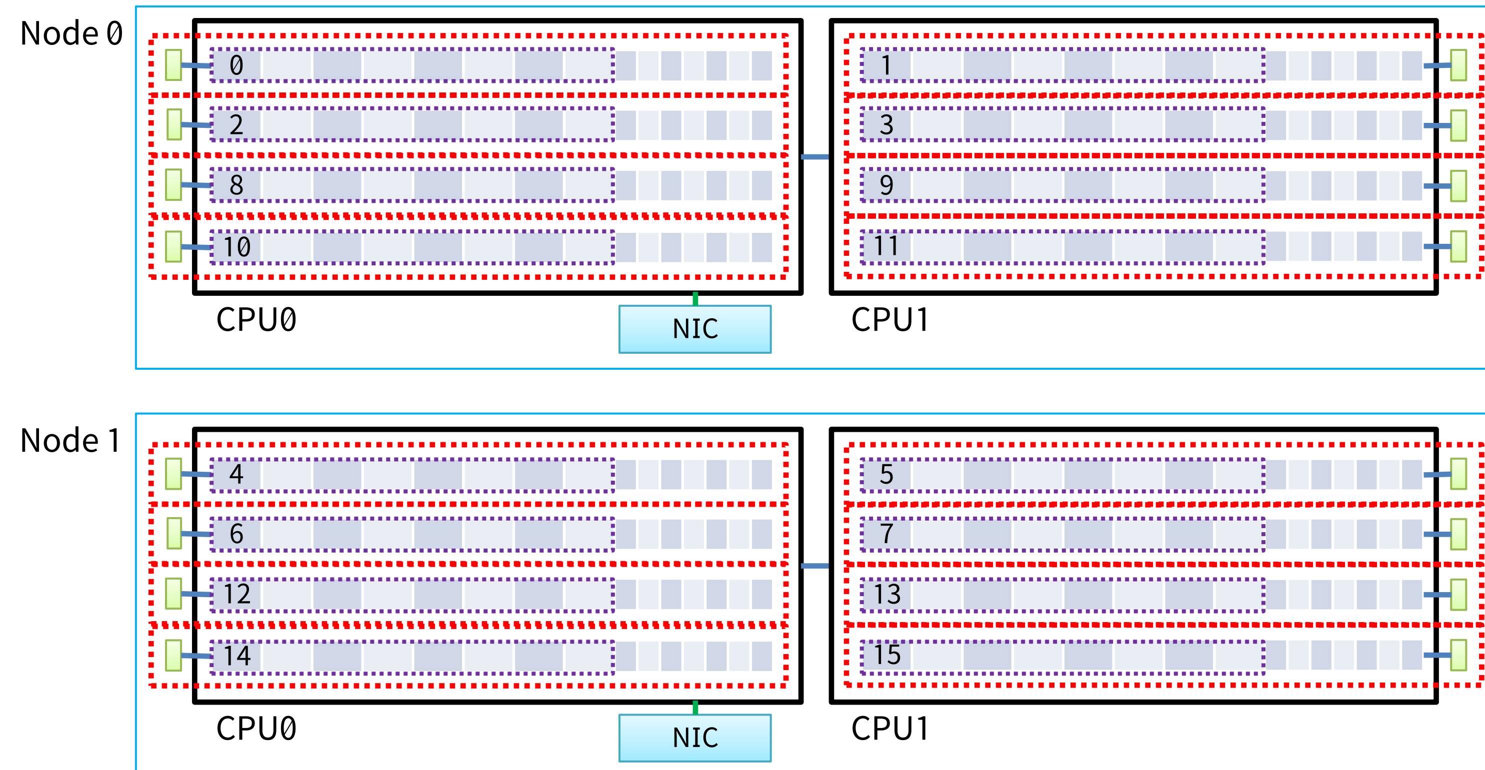
```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load intel
module load impi
export I_MPI_DEBUG=5
export I_MPI_PIN_ORDER=spread
export I_MPI_PIN_DOMAIN=omp
export I_MPI_PERHOST=4
export OMP_NUM_THREADS=8
export KMP_AFFINITY=verbose
mpiexec -print-rank-map -n 16 ./a.out
```



複数ノード、OpenMP+MPIハイブリッド並列化　3/3

- I_MPI_PERHOST=4 かつ I_MPI_PIN_ORDER=scatterの場合
  - ノード内ではscatterな分散配置
  - 4プロセス配置したら次のノードへ

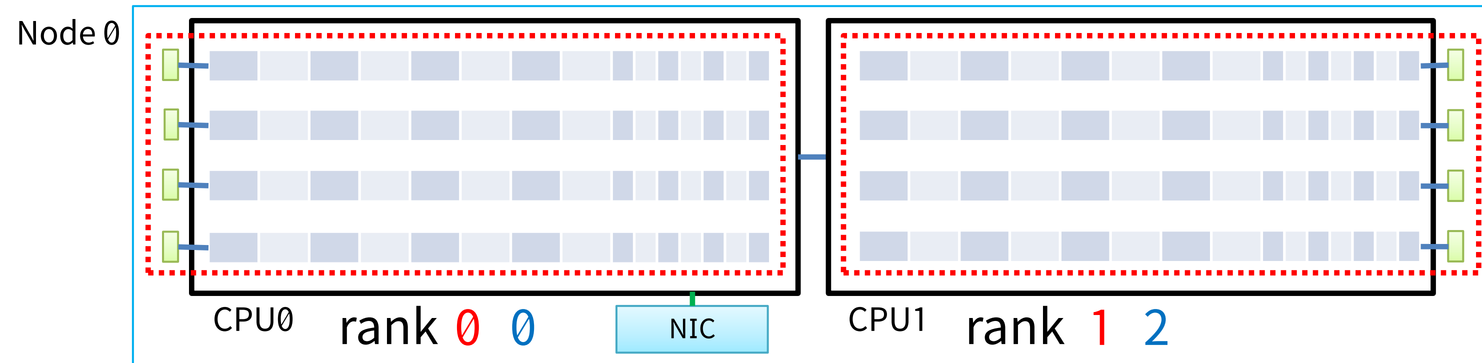
```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load intel
module load impi
export I_MPI_DEBUG=5
export I_MPI_PIN_ORDER=scatter
export I_MPI_PIN_DOMAIN=omp
export I_MPI_PERHOST=4
export OMP_NUM_THREADS=8
export KMP_AFFINITY=verbose
mpiexec -print-rank-map -n 16 ./a.out
```

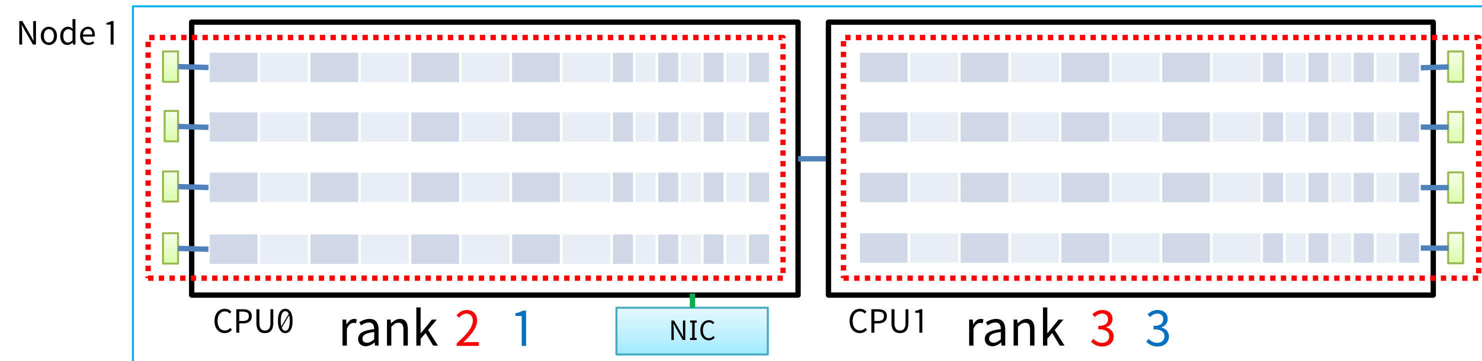


複数ノード、OpenMP+MPIハイブリッド並列化　B 1/2

- CPUソケットあたり1プロセス実行、1プロセスあたり60コア（に近い多スレッド）実行したい場合の例
  - I_MPI_PERHOSTの値によってランク番号の順序が変化（赤と青の文字色に対応）
  - I_MPI_PIN_ORDERはspreadでもscatterでも変わらず

```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load intel
module load impi
export I_MPI_DEBUG=5
export I_MPI_PIN_ORDER=spread
export I_MPI_PIN_DOMAIN=omp
export I_MPI_PERHOST=2
export I_MPI_PERHOST=1
export OMP_NUM_THREADS=60
export KMP_AFFINITY=verbose
mpiexec -print-rank-map -n 4 ./a.out
```





補足

- ジョブスクリプト内でプロセス番号（MPIランク番号）を使う方法
  - （もちろんMPIプログラム内ではMPI_Comm_rank関数でランク番号を取得できる）
  - ジョブスクリプト内で使いたい場合は2段階でスクリプトを実行する
  - mpiexecでスクリプトを呼び出すと、Intel MPIにより追加された環境変数をスクリプト内で使用可能
    - PMI_ で始まる変数などが追加される
  - 環境変数や実行時引数などに利用できる
  - numactlでより詳細なコア割り当てをしたい場合や、ノードグループB,CでGPU番号を指定する際に有用
    - MPIレベルでノード単位の割り当てだけ済ませておき、あとはnumactlで配置する、ということも可能

例

ジョブスクリプト

```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load intel
module load impi
export I_MPI_DEBUG=5
export I_MPI_PIN_ORDER=spread
export I_MPI_PIN_DOMAIN=omp
export I_MPI_PERHOST=8
export OMP_NUM_THREADS=8
export KMP_AFFINITY=verbose
mpiexec -print-rank-map -n 16 ./run2.sh
```

run2.sh （chmodで実行権を付与しておく）

```
#!/bin/bash
env
echo "RANK $PMI_RANK / $PMI_SIZE"
echo "LOCALRANK $MPI_LOCALRANKID / $MPI_LOCALNRANKS"
```

出力結果ファイルからRANKで始まるものとLOCALRANKで始まるものをそれぞれ抜き出してソートした結果 →

```
RANK 0 / 16        LOCALRANK 0 / 8
RANK 1 / 16        LOCALRANK 0 / 8
RANK 2 / 16        LOCALRANK 1 / 8
RANK 3 / 16        LOCALRANK 1 / 8
RANK 4 / 16        LOCALRANK 2 / 8
RANK 5 / 16        LOCALRANK 2 / 8
RANK 6 / 16        LOCALRANK 3 / 8
RANK 7 / 16        LOCALRANK 3 / 8
RANK 8 / 16        LOCALRANK 4 / 8
RANK 9 / 16        LOCALRANK 4 / 8
RANK 10 / 16       LOCALRANK 5 / 8
RANK 11 / 16       LOCALRANK 5 / 8
RANK 12 / 16       LOCALRANK 6 / 8
RANK 13 / 16       LOCALRANK 6 / 8
RANK 14 / 16       LOCALRANK 7 / 8
RANK 15 / 16       LOCALRANK 7 / 8
```

MPI_RANKとMPI_SIZEは総ランク数とID
MPI_LOCALRANKIDとMPI_LOCALRANKSは
ノード単位での総ランク数とID
（LOCALRANK情報はノード0とノード1で同一となるため重複している）

複数ノード、OpenMP+MPIハイブリッド並列化 B 2/2

- 同一プロセス内のスレッドがNUMAノードにまたがり配置・実行されるため、numactlでもそれにあわせたメモリ指定を行いたい

```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load intel
module load impi
export I_MPI_DEBUG=5
export I_MPI_PIN_ORDER=spread
export I_MPI_PIN_DOMAIN=omp
export OMP_NUM_THREADS=60
export KMP_AFFINITY=verbose
export I_MPI_PERHOST=2
mpiexec -print-rank-map -n 4 ./run2a.sh ./a.out
export I_MPI_PERHOST=1
mpiexec -print-rank-map -n 4 ./run2b.sh ./a.out
```

- 例：プロセス内ではCPUソケットに近いメモリノード全体をinterleaveアクセスするように指定（これが本当に有効かどうかはプログラム次第）

- PERHOST=2では偶数MPIランクがCPUソケット0側、奇数MPIランクがCPUソケット1側に配置される。MPIランクの偶数奇数で判定して配置した。

run2a.sh
```
#!/bin/bash
if [ $(( PMI_RANK % 2 )) -eq 0 ];then
    numactl --interleave=0-3 $@
else
    numactl --interleave=4-7 $@
fi
```

- PERHOST=1では前半MPIランクがCPUソケット0側、後半MPIランクがCPUソケット1側に配置される。ローカルランクで単純に判定して配置した。

run2b.sh
```
#!/bin/bash
if [ $MPI_LOCALRANKID -eq 0 ];then
    numactl --interleave=0-3 $@
else
    numactl --interleave=4-7 $@
fi
```

ノード共有実行の場合は？

- vnode-core<120 を指定して実行した、ノードの一部の資源のみが使えるジョブの場合
  - プロセス数をスレッド数を適切に指定し、I_MPI_PIN_DOMAIN=ompで実行すれば良い
  - 割り当たる資源の状況に大きく影響を受けるため、I_MPI_PIN_ORDERは指定してもあまり意味がないと思われる（spreadやscatterを指定しても、確保されているコアの状況によっては思ったようには割り当てられない）

MPIプログラムの具体的な実行方法
2. Open MPIを使う場合

Open MPIを使う：基礎編

- MPIプログラムの実行はmpirunまたはmpiexecで行う（どちらでも一緒）
- 主にmpirunへの引数で動作を制御する
  - プロセス数は実行時オプション-n
  - ノードあたりプロセス数は実行時オプション-npernode
  - プロセスの配置方法は実行時オプション--map-byや--bind-to
  - etc.
- -display-mapオプションや-display-devel-mapを使うと割り当て情報を確認可能

Open MPIを使う：出力ファイルの制御

- 標準出力・エラー出力をプロセスごとに別のファイルに書き出すには？
  - -output-filename 引数で出力ファイルを変更可能
    - 引数にはディレクトリ名を指定する。その下に
      1/rank.ランク番号/stdout　標準出力の出力先
      1/rank.ランク番号/stderr　標準エラー出力の出力先
      が作られる。
  - 例：TCS（バッチジョブシステム）のジョブIDを用いて重複しない名前のディレクトリに出力

```
mpirun -display-map -display-devel-map ¥
 -n 4 --map-by socket --bind-to socket ¥
 --rank-by node ¥
 -output-filename out_${PJM_JOBID} ./a.out
```

- ジョブスクリプト（bashスクリプト）は行末にバックスラッシュ（PowerPoint上では表示の都合で円記号）を書くことで行の折り返しが可能

Open MPIによる1ノード内複数プロセス実行：基本的な考え方

- プロセスの配置方法は主にmpirunの引数で制御
  - --map-by, --rank-by：どのような順序でプロセスを配置するか
    - --map-by numa：各NUMAノードに横断的にプロセスを配置する
    - --map-by ppr:X:numa：NUMAノードへXプロセスを配置してから次のNUMAノードへ
    - --rank-by socket, --rank-by node：MPIランク番号の振り方の指定
  - --bind-to：プロセスをどのような単位で配置するか
    - core, numa, socket などの単位でプロセスを割り付ける

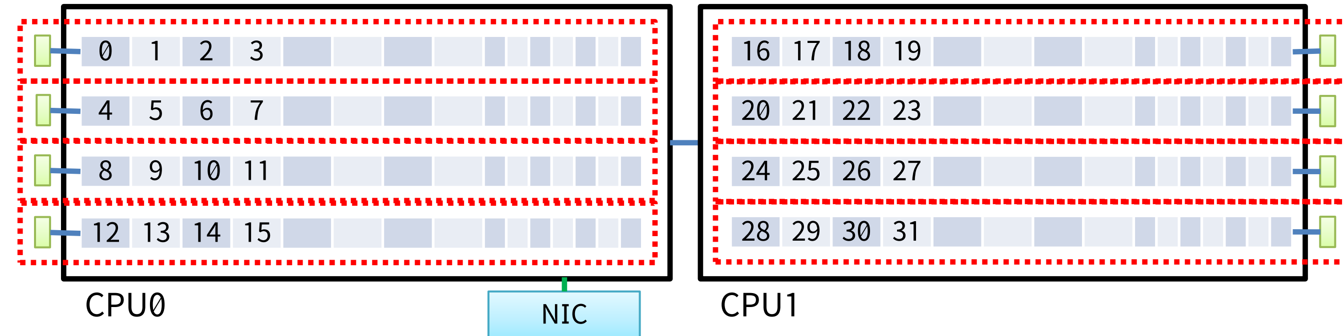
例：32プロセスフラットMPI

- 基本的に推奨：8つのNUMAノードに4プロセスずつ配置し、各プロセスがNUMAノード内のメモリにアクセスする
  - プロセス数がより少ない場合でも多い場合でも、全NUMAノードに均等に配置するのが基本
- 基本的には非推奨：片方のCPUソケットに全32プロセスを配置
  - プロセス同士が近いため通信性能に優れるが、そもそも1ノード全て使う意味がない
    - 多くのメモリ容量が欲しいためあえて1ノード占有する場合も、両ソケットに分散配置した方が全体的には高いメモリアクセス性能が期待しやすい

- 主なプロセス配置戦略
  - 1. NUMAノードあたり4プロセス配置してから次のNUMAノードへプロセスを配置する
    - 偏らない程度に端から詰めていくようなイメージ
  - 2. まず全NUMAノードに1プロセス配置し、次にまた全NUMAノードに1プロセス配置する
    - 全体的に均等に配置していくイメージ
  - どちらが良いかは、プログラム内のキャッシュの利用具合や、プロセス間の通信パターン次第

例1：偏らない程度に端から配置する

- --map-by ppr:X:numa --bind-to core オプションを指定すると、NUMAノードにX個の連続したランクのプロセスを割り当てたら次のNUMAノードへ、という挙動になり、NUMAノード内のプロセスは同じ連続するCPUコアに割り当てられる




```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=1
#PJM -L elapse=10:00
module load gcc
module load ompi
export OMP_NUM_THREADS=1
mpirun -display-map -display-devel-map -n 32 --map-by ppr:4:numa --bind-to core numactl -l ./a.out
```

-display-devel-map 指定時に得られる出力（割り当て情報）の一部

各プロセスがどこに配置されたのかが一目で確認できる

再確

```
Data for proc: [[37622,1],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/././././././././././././././././././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
        Binding: [B/./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
Data for proc: [[37622,1],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/././././././././././././././././././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
        Binding: [./B/././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
Data for proc: [[37622,1],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/././././././././././././././././././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
        Binding: [././B/./././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
Data for proc: [[37622,1],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/././././././././././././././././././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
        Binding: [./././B/././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
Data for proc: [[37622,1],4]
        Pid: 0  Local rank: 4   Node rank: 4    App rank: 4
        State: INITIALIZED      App_context: 0
        Locale:  [./././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/./././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
        Binding: [./././././././././././././././B/./././././././././././././././././././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
Data for proc: [[37622,1],5]
        Pid: 0  Local rank: 5   Node rank: 5    App rank: 5
        State: INITIALIZED      App_context: 0
        Locale:  [./././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/./././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
        Binding: [././././././././././././././././B/././././././././././././././././././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
Data for proc: [[37622,1],6]
        Pid: 0  Local rank: 6   Node rank: 6    App rank: 6
        State: INITIALIZED      App_context: 0
        Locale:  [./././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/./././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
        Binding: [./././././././././././././././././B/./././././././././././././././././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
Data for proc: [[37622,1],7]
        Pid: 0  Local rank: 7   Node rank: 7    App rank: 7
        State: INITIALIZED      App_context: 0
        Locale:  [./././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/./././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
        Binding: [././././././././././././././././././B/././././././././././././././././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
Data for proc: [[37622,1],8]
        Pid: 0  Local rank: 8   Node rank: 8    App rank: 8
        State: INITIALIZED      App_context: 0
        Locale:  [././././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/./././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
        Binding: [././././././././././././././././././././././././././././././B/./././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
```

CPUソケット0

CPUソケット1

プロセス0（ランク0）＝コア0

プロセス1（ランク1）＝コア1

プロセス2（ランク2）＝コア2

プロセス3（ランク3）＝コア3

プロセス4（ランク4）＝コア15

プロセス5（ランク5）＝コア16

プロセス6（ランク6）＝コア17

プロセス7（ランク7）＝コア18

プロセス8（ランク8）＝コア30

NUMA 0 | NUMA 1 | NUMA 2 | NUMA 3 | NUMA 4 | NUMA 5 | NUMA 6 | NUMA 7

-display-map 指定時に得られる出力（割り当て情報）の一部

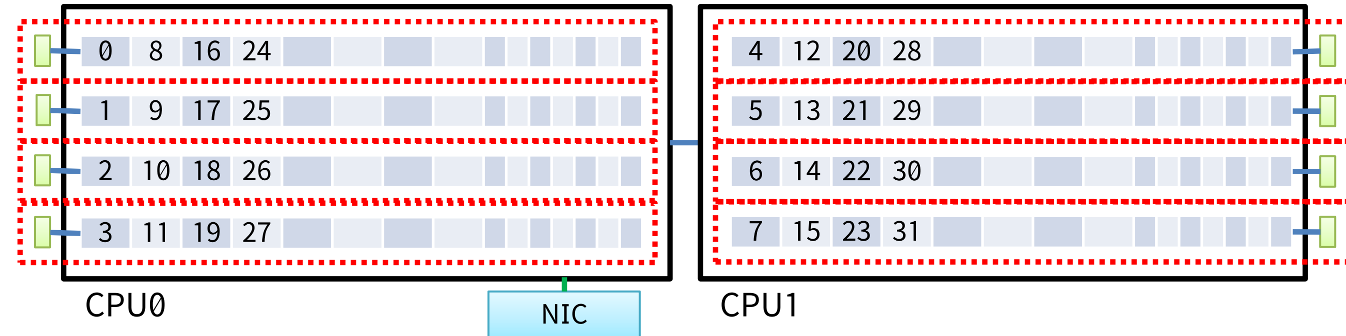
各プロセスがどこに配置されたのかが一目で確認できる
……のだが、コア数が多い環境では読みにくい

```
[B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
[B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
[B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
[B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
[././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
[././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
[././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
[././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
[./././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
[./././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
[./././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
[./././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././././././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././././././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././././././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././././././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][./././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././.]
[././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
[././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
[././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
[././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
```

この部分の見方は -display-devel-map と同様

例2：全体的に均等に配置する

- --map-by numa --bind-to core オプションを指定すると、NUMAノードに1プロセスを配置したら次のNUMAノードへ、という挙動になり、先の例よりも全体的に均等にプロセスが配置される
  - どちらが良いかはプログラム次第




```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=1
#PJM -L elapse=10:00
module load gcc
module load ompi
export OMP_NUM_THREADS=1
mpirun -display-map -display-devel-map -n 32 --map-by numa --bind-to core numactl -l ./a.out
```

（Intel MPIのscatter配置とはまた異なるようだ）

-display-devel-map 指定時に得られる出力（割り当て情報）の一部

各プロセスがどこに配置されたのかが一目で確認できる

```
Data for proc: [[8274,1],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/././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
        Binding: [B/././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
Data for proc: [[8274,1],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/././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
        Binding: [././././././././././././././././B/././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
Data for proc: [[8274,1],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/././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
        Binding: [././././././././././././././././././././././././././././././B/././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
Data for proc: [[8274,1],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][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
        Binding: [././././././././././././././././././././././././././././././././././././././././././././B/./././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
Data for proc: [[8274,1],4]
        Pid: 0  Local rank: 4   Node rank: 4    App rank: 4
        State: INITIALIZED      App_context: 0
        Locale:  [././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
        Binding: [././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][B/././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
Data for proc: [[8274,1],5]
        Pid: 0  Local rank: 5   Node rank: 5    App rank: 5
        State: INITIALIZED      App_context: 0
        Locale:  [././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
        Binding: [././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././B/././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
Data for proc: [[8274,1],6]
        Pid: 0  Local rank: 6   Node rank: 6    App rank: 6
        State: INITIALIZED      App_context: 0
        Locale:  [././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././.]
        Binding: [././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././B/././././././././././././././././././././././././././././././././././././././././././.]
Data for proc: [[8274,1],7]
        Pid: 0  Local rank: 7   Node rank: 7    App rank: 7
        State: INITIALIZED      App_context: 0
        Locale:  [././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
        Binding: [././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././B/./././././././././././././.]
Data for proc: [[8274,1],8]
        Pid: 0  Local rank: 8   Node rank: 8    App rank: 8
        State: INITIALIZED      App_context: 0
        Locale:  [B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
        Binding: [./B/./././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.][././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
```

CPUソケット0
CPUソケット1

プロセス0（ランク0）＝コア0
プロセス1（ランク1）＝コア15
プロセス2（ランク2）＝コア30
プロセス3（ランク3）＝コア45
プロセス4（ランク4）＝コア60
プロセス5（ランク5）＝コア75
プロセス6（ランク6）＝コア90
プロセス7（ランク7）＝コア115
プロセス8（ランク8）＝コア1

NUMA 0 | NUMA 1 | NUMA 2 | NUMA 3 | NUMA 4 | NUMA 5 | NUMA 6 | NUMA 7

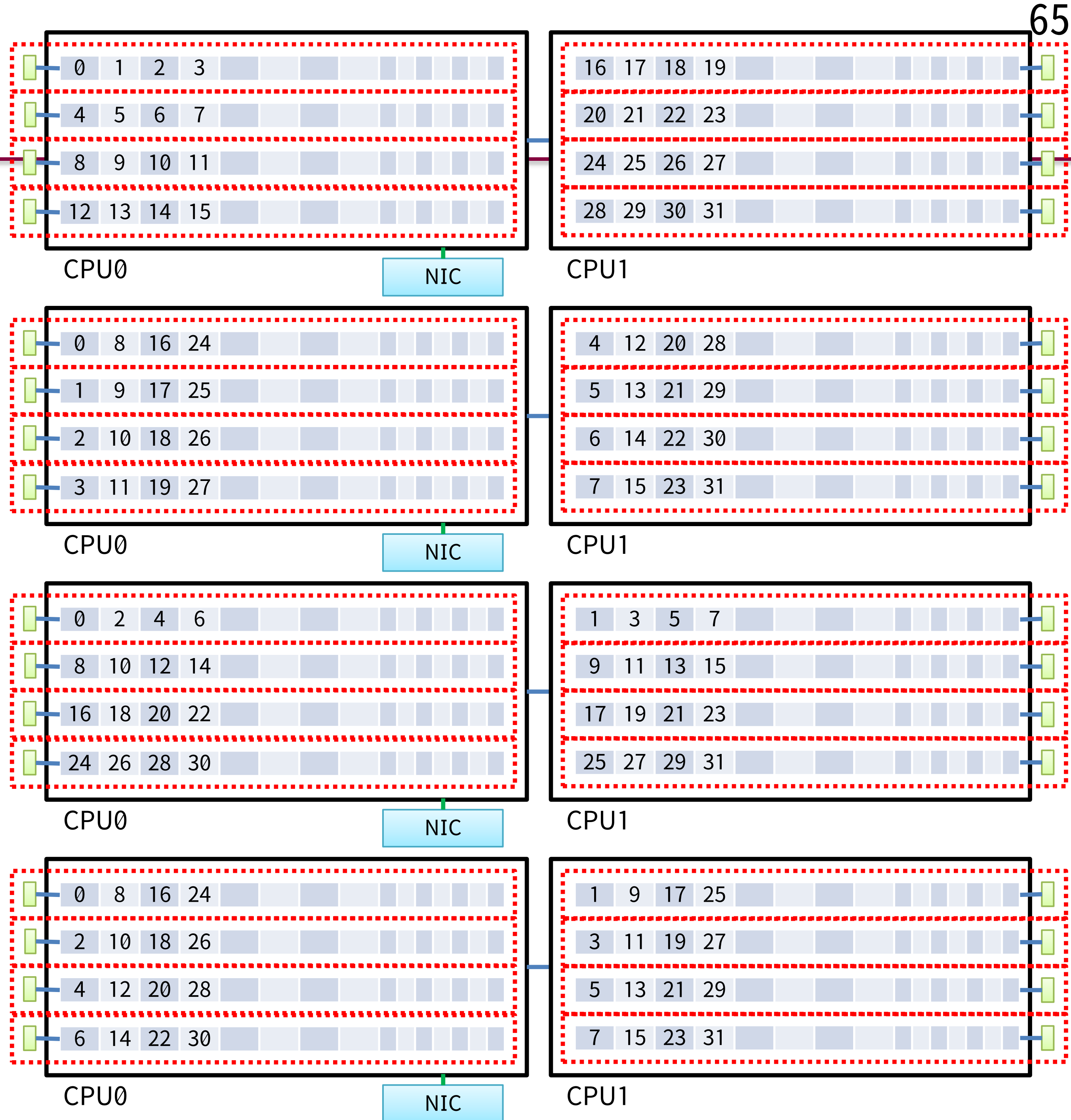
まとめ+α

例1
--map-by ppr:4:numa --bind-to core
(--rank-by numa)

例2
--map-by numa --bind-to core
(--rank-by numa)

例3
--map-by ppr:4:numa --bind-to core
--rank-by socket

例4
--map-by numa --bind-to core
--rank-by socket



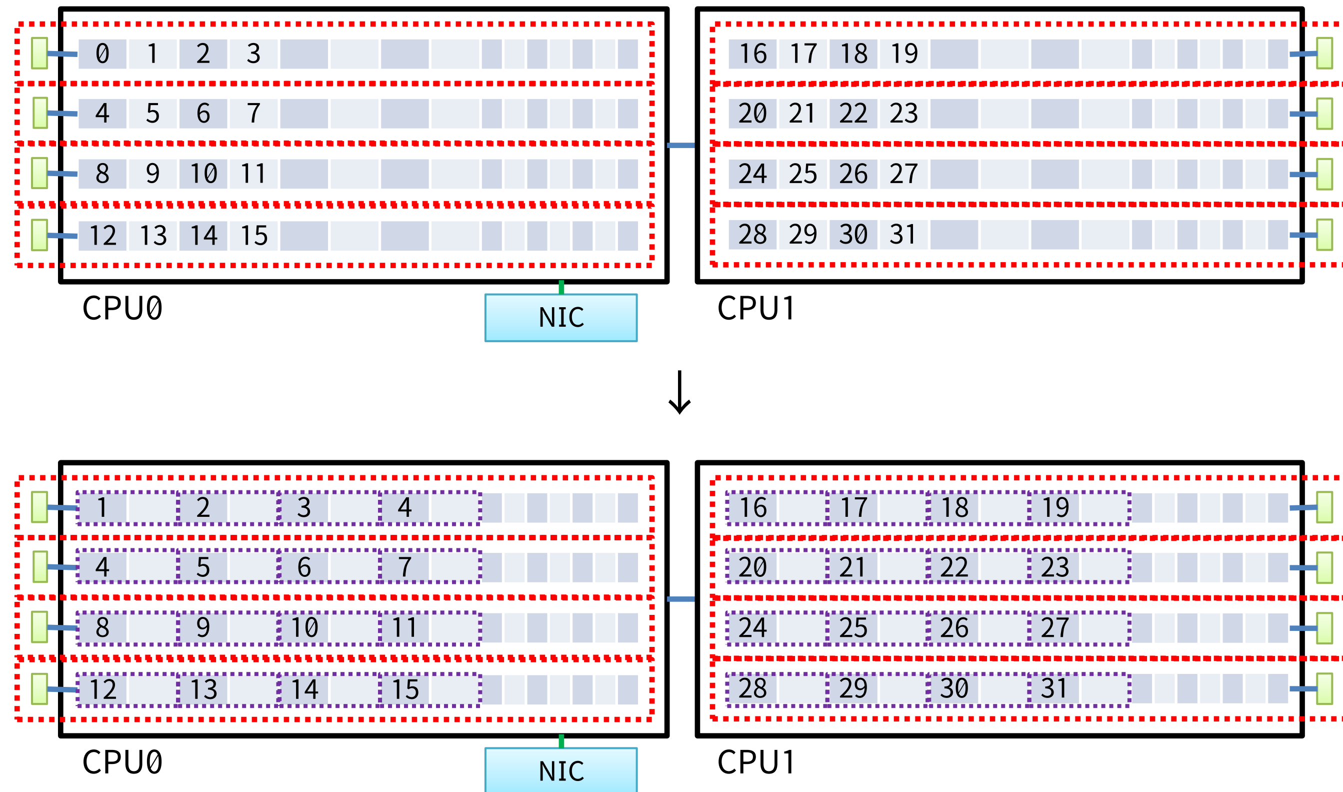
MPI + OpenMP ハイブリッド並列化　1/2

- OpenMPスレッド並列化と組み合わせる場合、一般的には同一プロセス内のスレッドを連続したCPUコアに割り当てたいはずである
- 本設定は --map-by オプションの末尾に :PE=スレッド数 を組み合わせるだけで良い
  - プロセスを配置する際に指定した数分だけのCPUコアをまとめて確保してくれるイメージ

```
export OMP_NUM_THREADS=1
mpirun -n 32 --map-by ppr:4:numa （改行なし）
 --bind-to core numactl -l ./a.out
```

↓

```
export OMP_NUM_THREADS=2
mpirun -n 32 --map-by ppr:4:numa:PE=2 （改行なし）
 --bind-to core numactl -l ./a.out
```

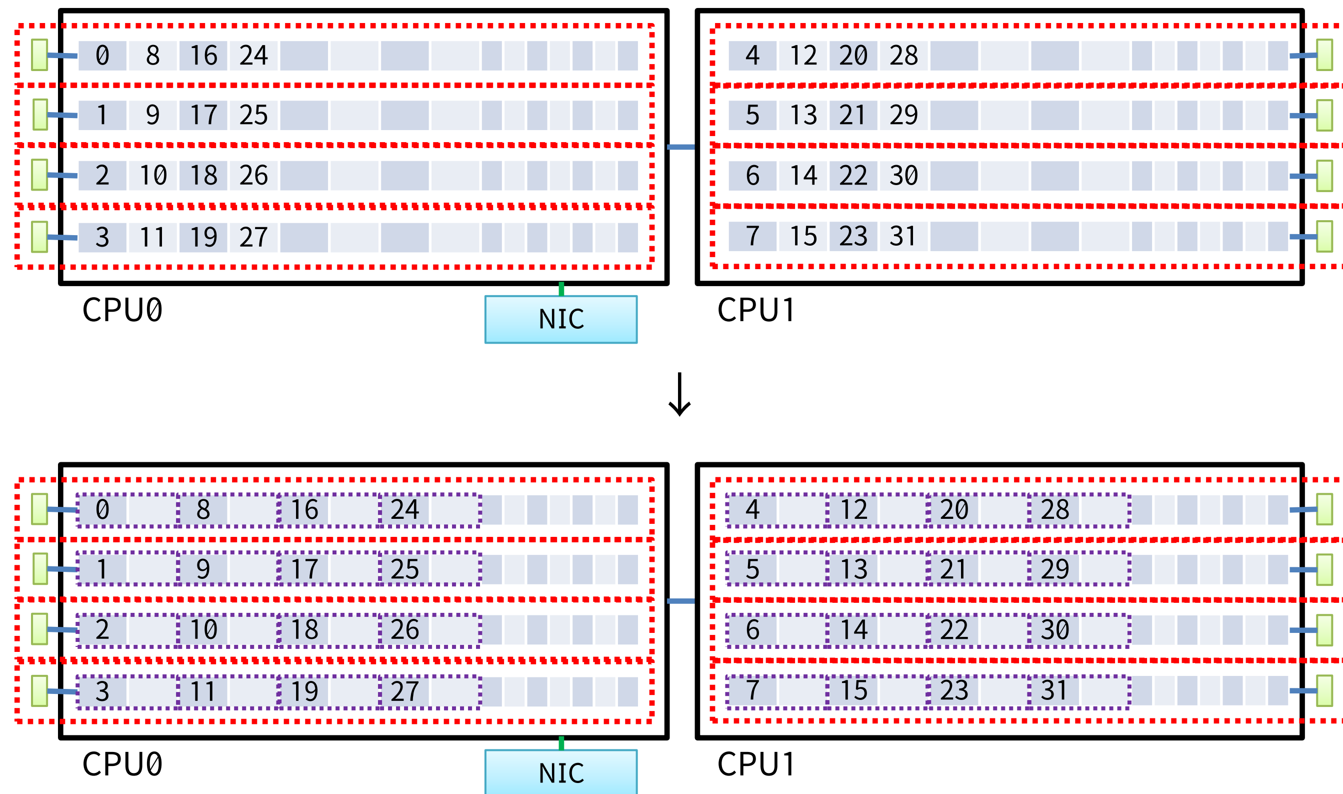


MPI + OpenMP ハイブリッド並列化　2/2

```
export OMP_NUM_THREADS=1
mpirun -n 32 --map-by numa （改行なし）
 --bind-to core numactl -l ./a.out
```

↓

```
export OMP_NUM_THREADS=2
mpirun -n 32 --map-by numa:PE=2 （改行なし）
 --bind-to core numactl -l ./a.out
```



複数ノード実行

- #PJM -L node=ノード数 により利用ノード数を指定（必須）
- ノードあたりプロセス数を指定するオプションもあるが、NUMAノード単位での配置設定と衝突することもあるため、必要に応じて使い分ける必要がある

- 先の例を単純に複数ノード化すると、ノード0への配置を終えてから、同様にノード1も埋める、という振る舞いになる

```
例1  --map-by ppr:4:numa --bind-to core (--rank-by numa)
例2  --map-by numa --bind-to core (--rank-by numa)
例3  --map-by ppr:4:numa --bind-to core --rank-by socket
例4  --map-by numa --bind-to core --rank-by socket
```

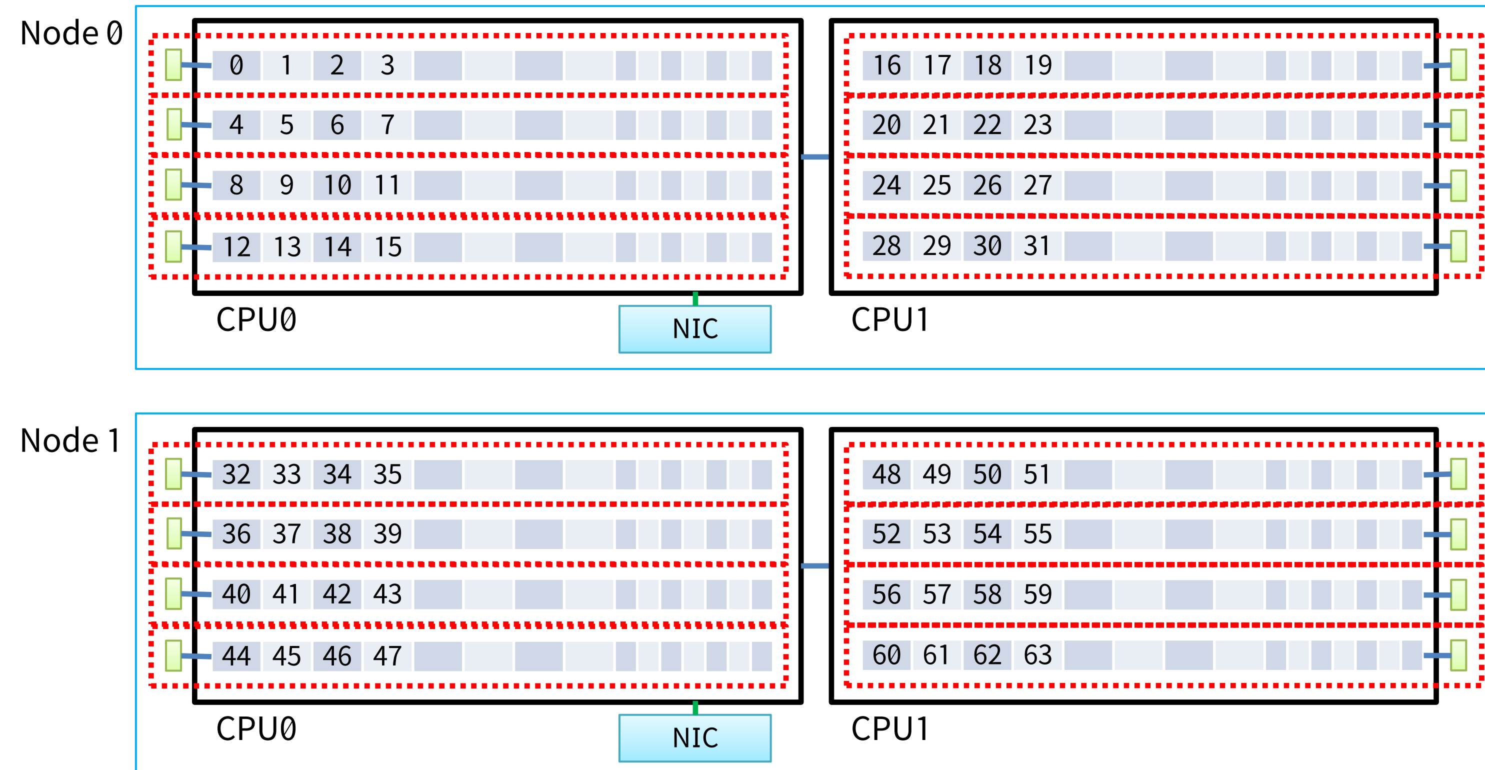
- --rank-by node を指定すると、連続するランクのプロセスが別のノードに配置される

```
例5  --map-by ppr:4:numa --bind-to core --rank-by node
例6  --map-by numa --bind-to core --rank-by node
```

例：32プロセス×2ノード（合計64プロセス）、フラットMPI　1/

- 「偏らない程度に端から配置する」例を2ノードに拡張
- ノード数と総プロセス数を変更
- ノード0への配置を終えてから、同様にノード1も埋める、という振る舞い

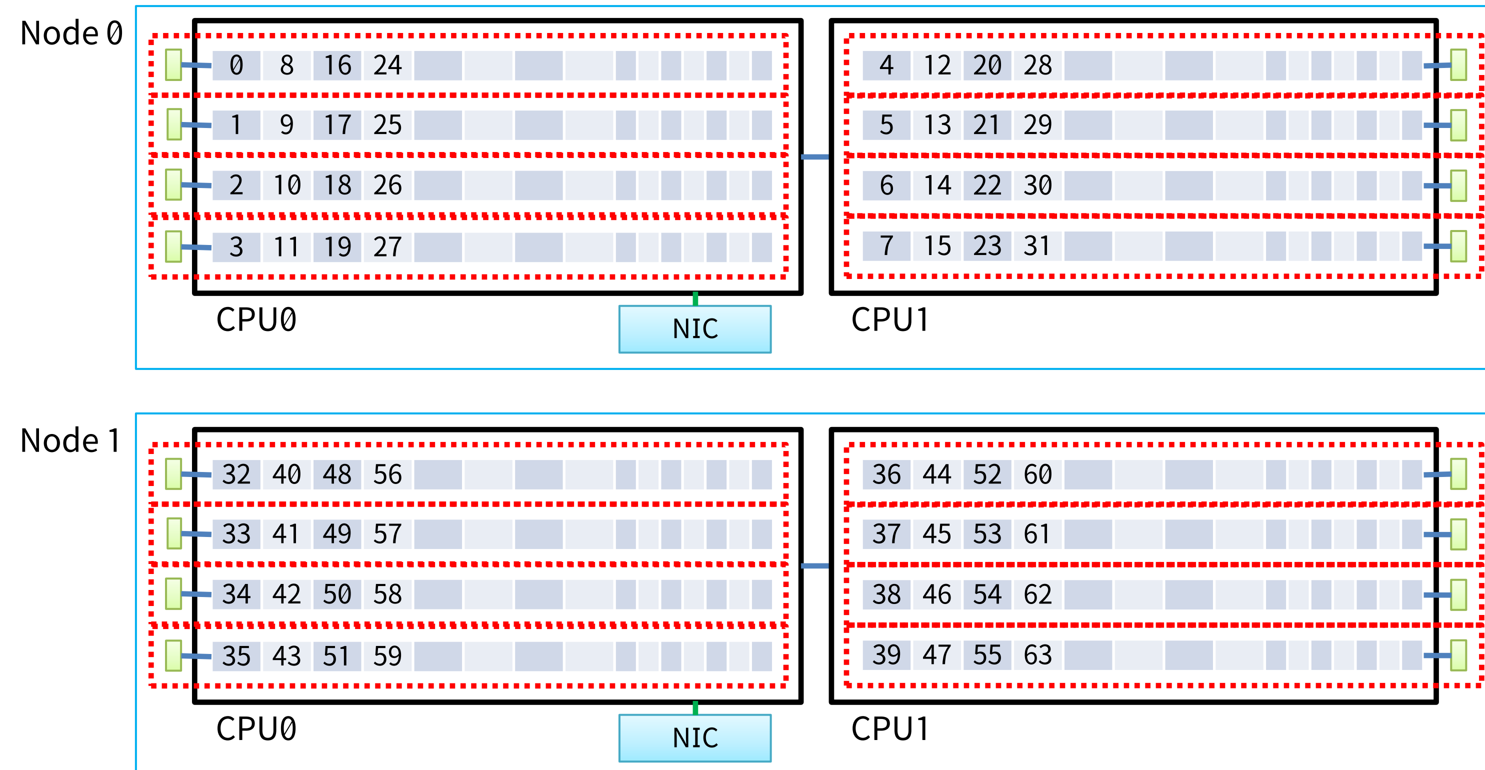
```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load gcc
module load ompi
export OMP_NUM_THREADS=1
mpirun -display-map -display-devel-map ¥
 -n 64 --map-by ppr:4:numa --bind-to core ¥
 numactl -l ./a.out
```



例：32プロセス×2ノード（合計64プロセス）、フラットMPI　2/

- 「全体的に均等に配置する」例を2ノードに拡張
- ノード数と総プロセス数を変更
- ノード0への配置を終えてから、同様にノード1も埋める、という振る舞い

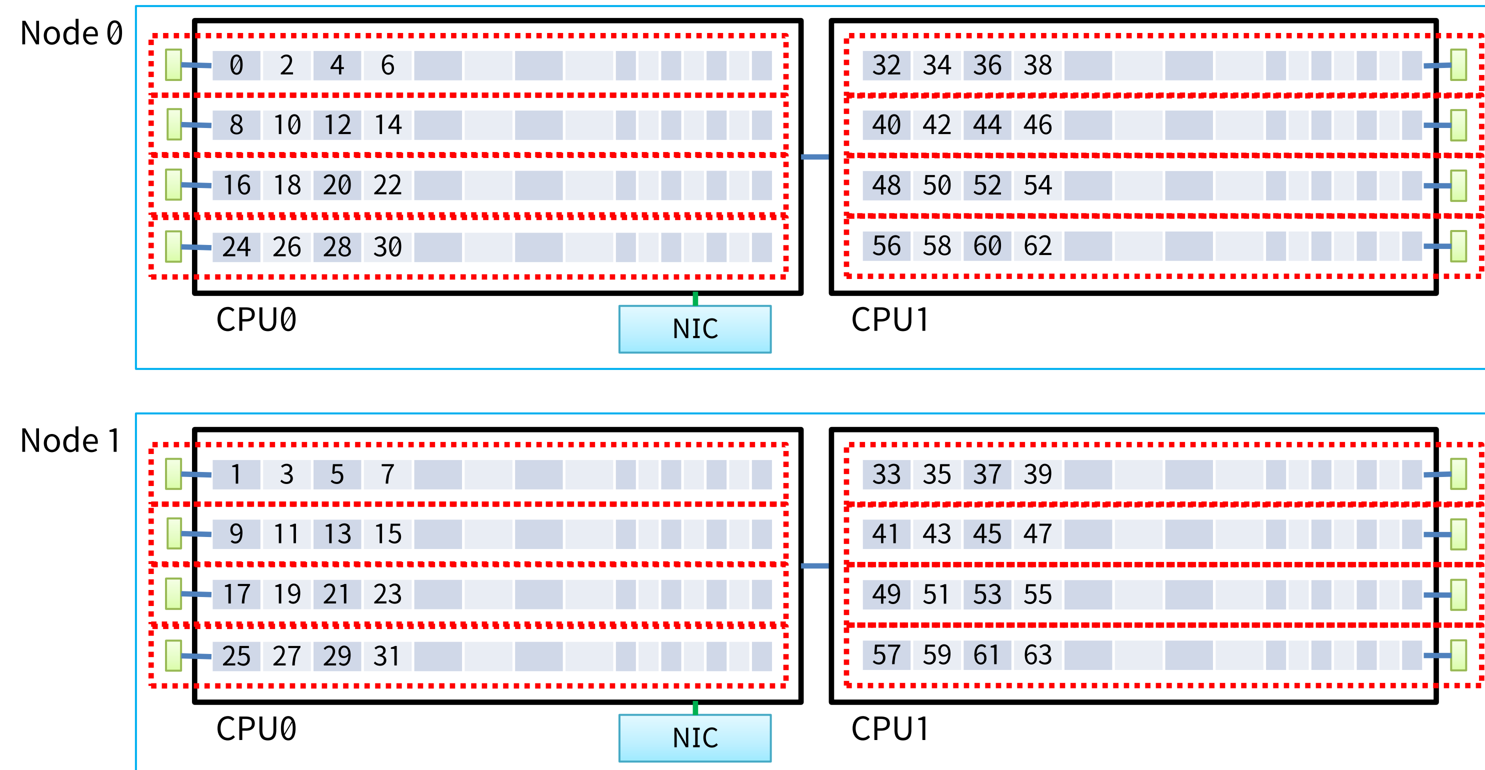
```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load gcc
module load ompi
export OMP_NUM_THREADS=1
mpirun -display-map -display-devel-map ¥
 -n 64 --map-by numa --bind-to core ¥
 numactl -l ./a.out
```



例：32プロセス×2ノード（合計64プロセス）、フラットMPI　3/

- 「偏らない程度に端から配置する」例を2ノードに拡張
- ノード数と総プロセス数を変更
- ノード0とノード1へ互い違いに配置（連続したランクは異なるノードへ配置）

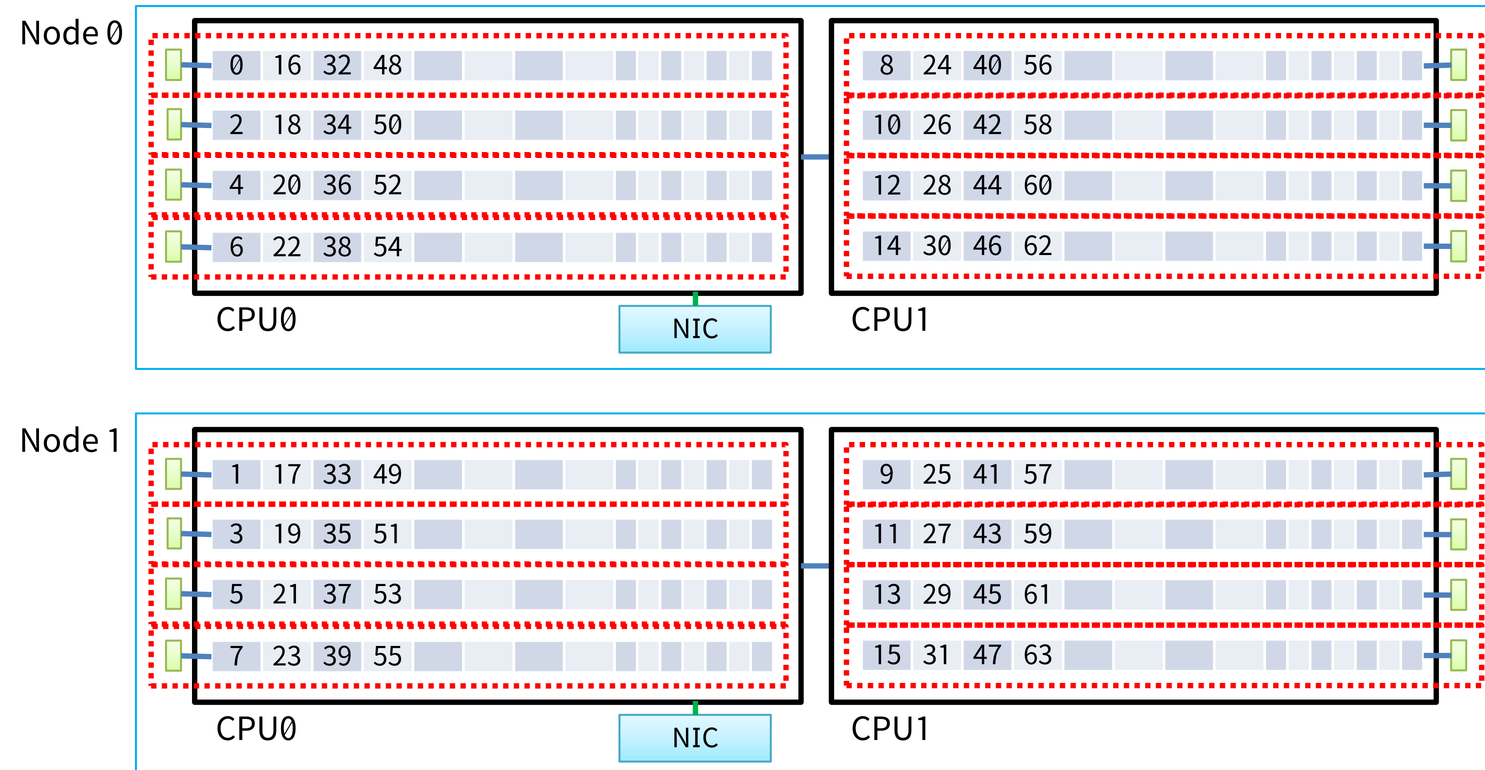
```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load gcc
module load ompi
export OMP_NUM_THREADS=1
mpirun -display-map -display-devel-map ¥
 -n 64 --map-by ppr:4:numa --bind-to core ¥
 --rank-by node numactl -l ./a.out
```



例：32プロセス×2ノード（合計64プロセス）、フラットMPI 4/

- 「全体的に均等に配置する」例を2ノードに拡張
- ノード数と総プロセス数を変更
- ノード0とノード1へ互い違いに配置（連続したランクは異なるノードへ配置）

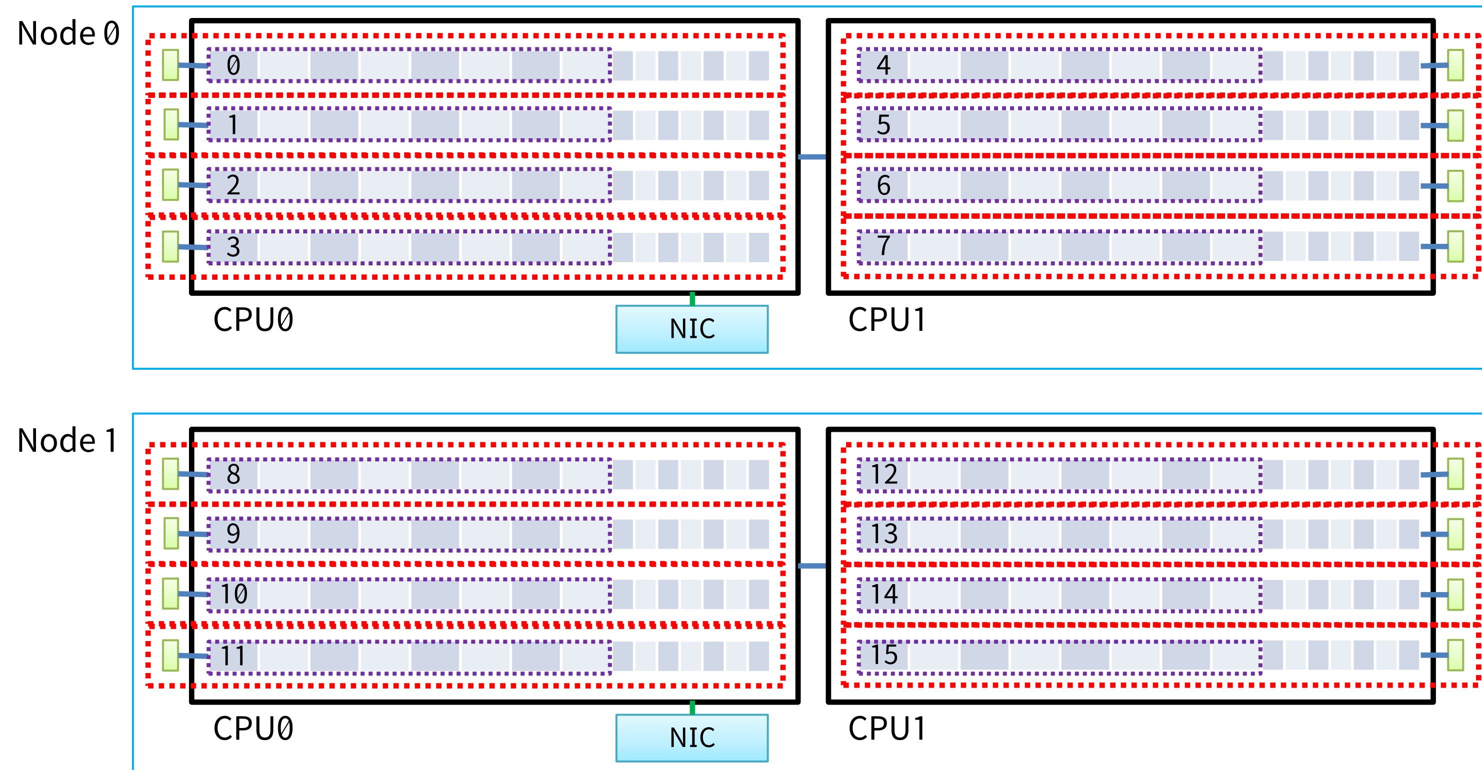
```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load gcc
module load ompi
export OMP_NUM_THREADS=1
mpirun -display-map -display-devel-map ¥
 -n 64 --map-by numa --bind-to core ¥
 --rank-by node numactl -l ./a.out
```



複数ノード、OpenMP+MPIハイブリッド並列化　A 1/2

- 単一ノードの場合と同様に、--map-by オプションの末尾に :PE=スレッド数 を組み合わせれば複数コア単位での割り当てになる
- 例： 8プロセス×2ノード、プロセスあたり8スレッド並列化
  - NUMAノードあたり1プロセスを配置し、NUMAノード内で2のべき乗最大のスレッド数を実行（HW構成にあう自然な実行形態の1つと考えられる）

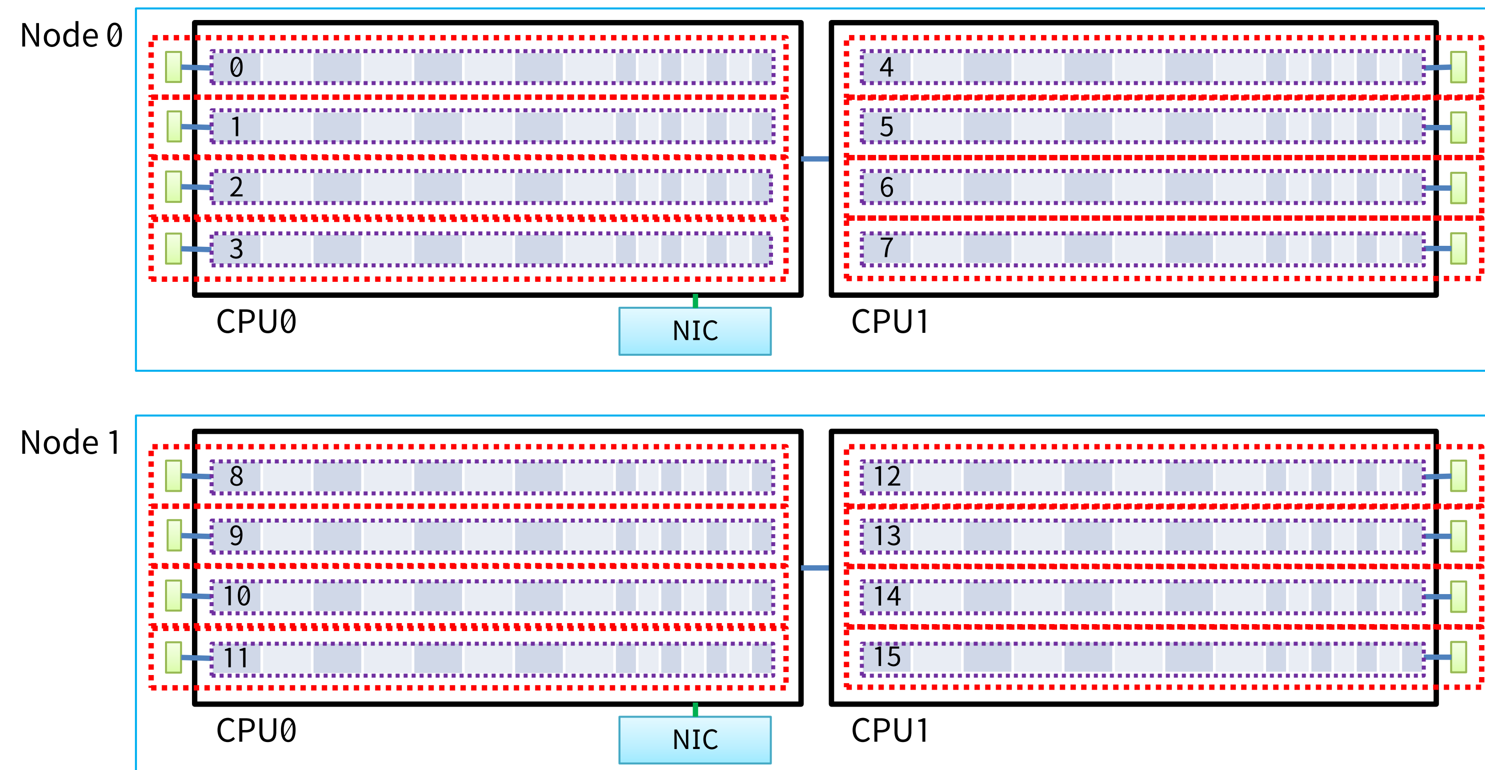
```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load gcc
module load ompi
export OMP_NUM_THREADS=8
mpirun -display-map -display-devel-map ¥
 -n 16 --map-by numa:PE=8 --bind-to core ¥
 numactl -l ./a.out
```



複数ノード、OpenMP+MPIハイブリッド並列化　A 2/2

- この実行形態については --may-by numa --bind-to numa などでも実質的に同じ
  - MPI的にはnuma単位でプロセスに割り当て、その中で8スレッドOpenMP実行

```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load gcc
module load ompi
export OMP_NUM_THREADS=8
mpirun -display-map -display-devel-map ¥
 -n 16 --map-by numa --bind-to numa ¥
 numactl -l ./a.out
```

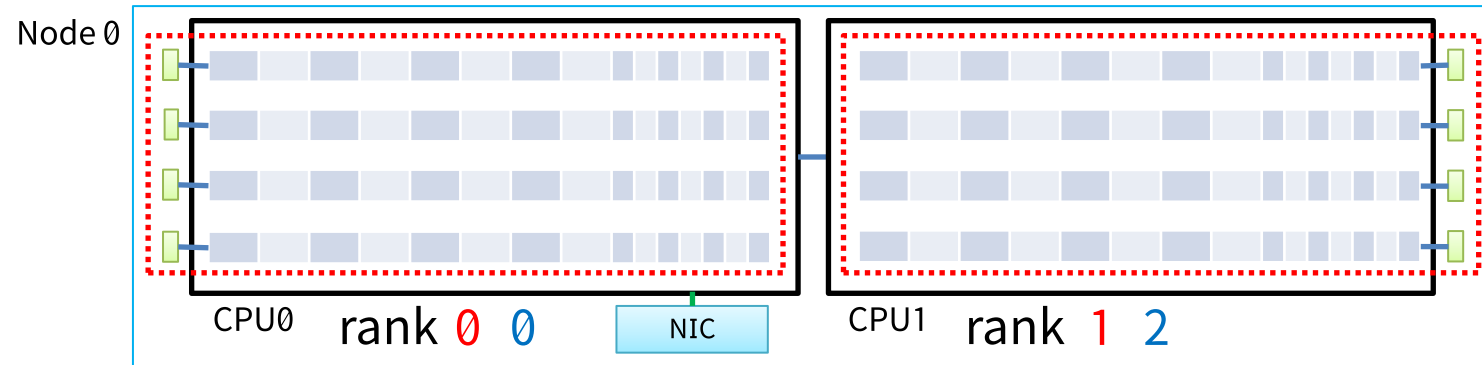


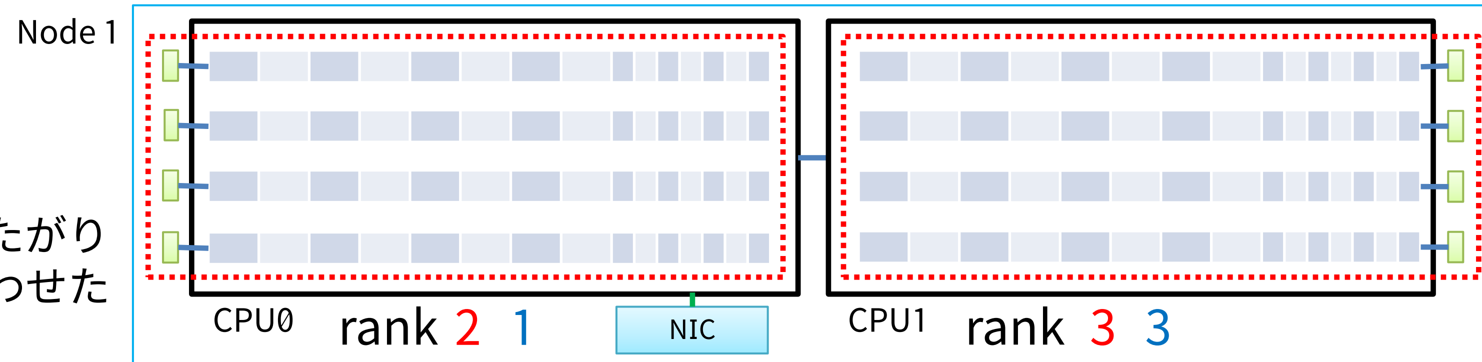
複数ノード、OpenMP+MPIハイブリッド並列化　B 1/2

- CPUソケットあたり1プロセス実行、1プロセスあたり60コア（に近い多スレッド）実行したい場合は --map-by socket --bind-to socket を指定すると良い
  - --rank-byの指定によってランク番号の順序が変化（赤と青の文字色に対応）

```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load gcc
module load ompi
export OMP_NUM_THREADS=60
mpirun -display-map -display-devel-map ¥
 -n 4 --map-by socket --bind-to socket ¥
 --rank-by socket ./a.out
mpirun -display-map -display-devel-map ¥
 -n 4 --map-by socket --bind-to socket ¥
 --rank-by node ./a.out
```

- 同一プロセス内のスレッドがNUMAノードにまたがり配置・実行されるため、numactlでもそれにあわせたメモリ指定を行うべきかもしれない（後述）





-display-devel-map 指定時に得られる出力（割り当て情報）の比較

--rank-by socket の場合

CPUソケット0　CPUソケット1

```
Data for node: a0001    State: 3        Flags: 11
        Daemon: [[37622,0],0]   Daemon launched: True
        Num slots: 120  Slots in use: 2 Oversubscribed: FALSE
        Num slots allocated: 120        Max slots: 0
        Num procs: 2    Next node_rank: 2
        Data for proc: [[37622,1],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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/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: [[37622,1],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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]

Data for node: 172.16.1.2       State: 1        Flags: 13
        Daemon: [[37622,0],1]   Daemon launched: True
        Num slots: 120  Slots in use: 2 Oversubscribed: FALSE
        Num slots allocated: 120        Max slots: 0
        Num procs: 2    Next node_rank: 2
        Data for proc: [[37622,1],2]
                Pid: 0  Local rank: 0   Node rank: 0    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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
                Binding: UNBOUND
        Data for proc: [[37622,1],3]
                Pid: 0  Local rank: 1   Node rank: 1    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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
                Binding: UNBOUND
Data for JOB [37622,1] offset 0 Total slots allocated 240

Mapper requested: NULL  Last mapper: round_robin  Mapping policy: BYSOCKET:NOOVERSUBSCRIBE  Ranking policy: SOCKET
Binding policy: SOCKET  Cpu set: NULL  PPR: NULL  Cpus-per-rank: 0
        Num new daemons: 0      New daemon starting vpid INVALID
        Num nodes: 2

Data for node: a0001    State: 1        Flags: 10
        Daemon: [[37622,0],0]   Daemon launched: False
        Num slots: 120  Slots in use: 0 Oversubscribed: FALSE
        Num slots allocated: 120        Max slots: 0
        Num procs: 2    Next node_rank: 2
        Data for proc: [[37622,1],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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
                Binding: UNBOUND
        Data for proc: [[37622,1],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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
                Binding: UNBOUND

Data for node: 172.16.1.2       State: 1        Flags: 10
        Daemon: [[37622,0],1]   Daemon launched: False
        Num slots: 120  Slots in use: 0 Oversubscribed: FALSE
        Num slots allocated: 120        Max slots: 0
        Num procs: 2    Next node_rank: 2
        Data for proc: [[37622,1],2]
                Pid: 0  Local rank: 0   Node rank: 0    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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/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: [[37622,1],3]
                Pid: 0  Local rank: 1   Node rank: 1    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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
```

App rank: 0 → プロセス0（ランク0）
App rank: 1 → プロセス1（ランク1）
App rank: 2 → プロセス2（ランク2）
App rank: 3 → プロセス3（ランク3）

BindingがBとなっているところ（のいずれか）にプロセスが配置される
＝1プロセス60スレッドが
　ノード0 ソケット0、ノード0 ソケット1、ノード1 ソケット0、ノード1 ソケット1
　の順に配置されたことが確認できる

（ノード数の2乗分だけ表示されるため、実行ノード数が増えると表示が増えてしまう）

-display-devel-map 指定時に得られる出力（割り当て情報）の比較

--rank-by node の場合

CPUソケット0　　CPUソケット1

```
Data for node: a0003	State: 3	Flags: 11
	Daemon: [[30227,0],0]	Daemon launched: True
	Num slots: 120	Slots in use: 2	Oversubscribed: FALSE
	Num slots allocated: 120	Max slots: 0
	Num procs: 2	Next node_rank: 2
	Data for proc: [[30227,1],0]
		Pid: 0	Local rank: 0	Node rank: 0	App rank: 0    プロセス0（ランク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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/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: [[30227,1],2]
		Pid: 0	Local rank: 1	Node rank: 1	App rank: 2    プロセス2（ランク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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]

Data for node: 172.16.1.4	State: 1	Flags: 13
	Daemon: [[30227,0],1]	Daemon launched: True
	Num slots: 120	Slots in use: 2	Oversubscribed: FALSE
	Num slots allocated: 120	Max slots: 0
	Num procs: 2	Next node_rank: 2
	Data for proc: [[30227,1],1]
		Pid: 0	Local rank: 0	Node rank: 0	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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
		Binding: UNBOUND
	Data for proc: [[30227,1],3]
		Pid: 0	Local rank: 1	Node rank: 1	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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
		Binding: UNBOUND
Data for JOB [30227,1] offset 0 Total slots allocated 240

Mapper requested: NULL  Last mapper: round_robin  Mapping policy: BYSOCKET:NOOVERSUBSCRIBE  Ranking policy: NODE
Binding policy: SOCKET  Cpu set: NULL  PPR: NULL  Cpus-per-rank: 0
	Num new daemons: 0	New daemon starting vpid INVALID
	Num nodes: 2

Data for node: a0003	State: 1	Flags: 10
	Daemon: [[30227,0],0]	Daemon launched: False
	Num slots: 120	Slots in use: 0	Oversubscribed: FALSE
	Num slots allocated: 120	Max slots: 0
	Num procs: 2	Next node_rank: 2
	Data for proc: [[30227,1],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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B][./././././././././././././././././././././././././././././././././././././././././././././././././././././././.]
		Binding: UNBOUND
	Data for proc: [[30227,1],2]
		Pid: 0	Local rank: 1	Node rank: 1	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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
		Binding: UNBOUND

Data for node: 172.16.1.4	State: 1	Flags: 10
	Daemon: [[30227,0],1]	Daemon launched: False
	Num slots: 120	Slots in use: 0	Oversubscribed: FALSE
	Num slots allocated: 120	Max slots: 0
	Num procs: 2	Next node_rank: 2
	Data for proc: [[30227,1],1]
		Pid: 0	Local rank: 0	Node rank: 0	App rank: 1    プロセス1（ランク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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/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: [[30227,1],3]
		Pid: 0	Local rank: 1	Node rank: 1	App rank: 3    プロセス3（ランク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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/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/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B/B]
```

BindingがBとなっているところ（のいずれか）にプロセスが配置される
＝1プロセス60スレッドが
　ノード0 ソケット0、ノード0 ソケット1、ノード1 ソケット0、ノード1 ソケット1
　の順に配置されたことが確認できる

（ノード数の2乗分だけ表示されるため、実行ノード数が増えると表示が増えてしまう）

補足

- ジョブスクリプト内でプロセス番号（MPIランク番号）を使う方法
  - （もちろんMPIプログラム内ではMPI_Comm_rank関数でランク番号を取得できる）
  - ジョブスクリプト内で使いたい場合は2段階でスクリプトを実行する
  - mpiexecでスクリプトを呼び出すと、Open MPIにより追加された環境変数をスクリプト内で使用可能
    - OMPI_ で始まる変数などが追加される
  - 環境変数や実行時引数などに利用できる
  - numactlでより詳細なコア割り当てをしたい場合や、ノードグループB,CでGPU番号を指定する際に有用
    - MPIレベルでノード単位の割り当てだけ済ませておき、あとはnumactlで配置する、ということも可能

例

ジョブスクリプト

```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load gcc
module load ompi
export OMP_NUM_THREADS=8
mpiexec -display-map -display-devel-map -n 16 --map-by ppr:4:numa --bind-to core ./run2.sh
```

run2.sh （chmodで実行権を付与しておく）

```
#!/bin/bash
env
echo "RANK $OMPI_COMM_WORLD_RANK / $OMPI_COMM_WORLD_SIZE"
echo "LOCALRANK $OMPI_COMM_WORLD_LOCAL_RANK / $OMPI_COMM_WORLD_LOCAL_SIZE"
```

出力結果ファイルからRANKで始まるものと
LOCALRANKで始まるものを
それぞれ抜き出してソートした結果 →

OMPI_COMM_WORLD_RANKとOMPI_COMM_WORLD_SIZEは総ランク数とID
OMPI_COMM_WORLD_LOCAL_RANKとOMPI_COMM_WORLD_LOCAL_SIZEは
ノード単位での総ランク数とID
（OMPI_COMM_WORLD_LOCAL_RANK情報はノード0とノード1で
同一となるため重複している）

```
RANK 0 / 16        LOCALRANK 0 / 8
RANK 1 / 16        LOCALRANK 0 / 8
RANK 2 / 16        LOCALRANK 1 / 8
RANK 3 / 16        LOCALRANK 1 / 8
RANK 4 / 16        LOCALRANK 2 / 8
RANK 5 / 16        LOCALRANK 2 / 8
RANK 6 / 16        LOCALRANK 3 / 8
RANK 7 / 16        LOCALRANK 3 / 8
RANK 8 / 16        LOCALRANK 4 / 8
RANK 9 / 16        LOCALRANK 4 / 8
RANK 10 / 16       LOCALRANK 5 / 8
RANK 11 / 16       LOCALRANK 5 / 8
RANK 12 / 16       LOCALRANK 6 / 8
RANK 13 / 16       LOCALRANK 6 / 8
RANK 14 / 16       LOCALRANK 7 / 8
RANK 15 / 16       LOCALRANK 7 / 8
```

複数ノード、OpenMP+MPIハイブリッド並列化　B 2/2

- 同一プロセス内のスレッドがNUMAノードにまたがり配置・実行されるため、numactlでもそれにあわせたメモリ指定を行いたい

```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L node=2
#PJM -L elapse=10:00
module load gcc
module load ompi
export OMP_NUM_THREADS=60
mpirun -display-map -display-devel-map ¥
 -n 4 --map-by socket --bind-to socket ¥
 --rank-by socket ./run2a.sh ./a.out
mpirun -display-map -display-devel-map ¥
 -n 4 --map-by socket --bind-to socket ¥
 --rank-by node ./run2b.sh ./a.out
```

- 例：プロセス内ではCPUソケットに近いメモリノード全体をinterleaveアクセスするように指定（これが本当に有効かどうかはプログラム次第）

- --rank-by socketでは偶数MPIランクがCPUソケット0側、奇数MPIランクがCPUソケット1側に配置される。MPIランクの偶数奇数で判定して配置した。

```
run2a.sh
#!/bin/bash
if [ $(( OMPI_COMM_WORLD_RANK % 2 )) -eq 0 ];then
    numactl --interleave=0-3 $@
else
    numactl --interleave=4-7 $@
fi
```

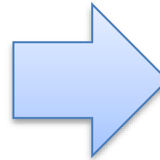
- --rank-by nodeでは前半MPIランクがCPUソケット0側、後半MPIランクがCPUソケット1側に配置される。ローカルランクで単純に判定して配置した。

```
run2b.sh
#!/bin/bash
if [ $OMPI_COMM_WORLD_LOCAL_RANK -eq 0 ];then
    numactl --interleave=0-3 $@
else
    numactl --interleave=4-7 $@
fi
```

ノード共有実行の場合は？

- vnode-core<120 を指定して実行した、ノードの一部の資源のみが使えるジョブの場合
- 計算コアがどのように提供されてくるかわからないが、コアをどのように配置するかをまったく指定しないで実行するとエラーしてしまう
  - 例

```
#!/bin/bash
#PJM -L rscgrp=a-batch
#PJM -L vnode-core=4
#PJM -L elapse=10:00
module load gcc
module load ompi
export OMP_NUM_THREADS=2
mpirun -display-map -display-devel-map -n 2 ./a.out
```



```
Your job has requested more processes than the ppr for
this topology can support:

  App: ./a.out
  Number of procs:  2
  PPR: 1:node

Please revise the conflict and try again.
```

- --map-by ppr:プロセス数:node:PE=プロセスあたりスレッド数 --bind-to core が妥当
  - 例

```
#PJM -L vnode-core=4
export OMP_NUM_THREADS=2
mpirun -display-map -display-devel-map -n 2 --map-by ppr:2:node:PE=2 --bind-to core ./a.out
```

※vnode-core = プロセス数×プロセスあたりスレッド数

その他・補足

参考：プログラム側から実行されているコアを確認する方法

- /proc/プロセスID/task/スレッドID/stat を確認する
  - 38番目のエントリがコア番号
- sched_getaffinity で得られる情報を確認する

/proc/プロセスID/task/スレッドID/stat を確認する例

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/syscall.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 %d¥n",
           hostname, rank, tid, ompid, buf2, atoi(buf2)/15);
  }
```

右上へ続く

```
    FILE *fout;
    char fname[0xff];
    sprintf(buf, "/proc/%d/task/%d/stat", pid, tid);
    snprintf(fname, 0xff, "stat_%d_%d.txt", pid, tid);
    if ((fout = fopen(fname, "w")) != NULL) {
      if ((fp = fopen(buf, "r")) != NULL) {
        while(fgets(buf, 0xff, fp)!=NULL){
          fprintf(fout, "%s", buf);
        }
        fclose(fp);
      }
      fclose(fout);
    }
  }

int main()
{
  #pragma omp parallel
  check_core(0);
}
```

sched_getaffinity で得られる情報を確認する例

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/syscall.h>
#include <omp.h>
#include <sched.h>

void check_core(int rank){
  int pid, tid, ompid;
  pid = getpid();
  tid = (pid_t) syscall(SYS_gettid);
  ompid = omp_get_thread_num();

  cpu_set_t cpuset;
  CPU_ZERO(&cpuset);
#ifdef _OPENMP
  int ret = sched_getaffinity(tid, sizeof(cpuset), &cpuset);
#else
  int ret = sched_getaffinity(pid, sizeof(cpuset), &cpuset);
#endif
  char affinity[0xfff];
  affinity[0] = '¥0';
  int ncore = sysconf(_SC_NPROCESSORS_CONF);
  for (int i = 0; i < ncore; i++) {
    if (CPU_ISSET(i, &cpuset) == 1) {
      snprintf(affinity, 0xfff, "%s %d", affinity, i);
    }
  }
  printf("getaffinity rank=%d omp-tid=%2d core-id=%s¥n", rank, ompid, affinity);
}
```

```
int main()
{
  #pragma omp parallel
  check_core(0);
}
```

右上へ続く