FX1000における
インターコネクトの利用について

2023.9.6

富士通株式会社
ジャパン・グローバルゲートウェイ
Public & Social ITS Division
笠井良浩

目次

1. はじめに

FUJITSU

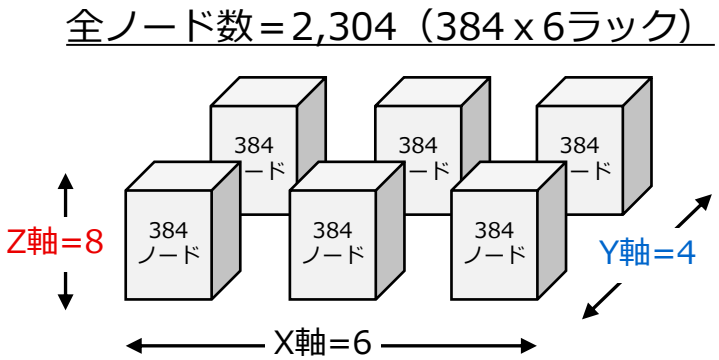
- FX1000での並列ジョブ実行は、指定方法によって性能差が生じる場合があります。特にMPI通信については、プロセスの配置が通信性能に大きく影響するため、通信性能の改善を図る場合は、プロセスの配置について把握することが重要になります。今回は通信性能に影響するノードの形状やプロセスの配置について説明をします。

  - ノード形状によるプロセスの配置について
  - プロセスがどのように配置されたかを確認する方法

2. FX1000のインターコネクトの構成（1/2）

FUJITSU

- FX1000のインターコネクトの構成
  - 不老システムTypeIサブシステム
    TypeIサブシステムはFX1000を2,304ノード／6ラックで構成されている



  - TofuインターコネクトDの形状
    全6ラックでのTofuインターコネクトD形状は（X,Y,Z,a,b,c）＝（6,4,8,2,3,2）となる
    それぞれの次元だと以下の形状になる
    - 3次元の場合=12x12x16　　（ X*a, Y*b, Z*c ）
    - 2次元の場合=36x64　　（ X*a*b, Y*Z*c ）
    - 1次元の場合=2304　　（ X*a*Y*b*Z*c ）

2. FX1000のインターコネクトの構成（2/2）

- FX1000のインターコネクトの構成
  - 不老システムTypeIサブシステムのリソースグループの確認
    - pjstatコマンドによる確認

```
[username@flow-fx01 ~]$ pjstat --rsc

RSCUNIT                                 RSCUNIT_SIZE  RSCGRP                              RSCGRP_SIZE
fx[ENABLE,                              6x4x8         fx-large[ENABLE,START]              8x12x16
fx[ENABLE,                              6x4x8         fx-xlarge[ENABLE,START]             8x12x16
fx[ENABLE,                              6x4x8         fx-bmt[ENABLE,START]                6x12x16
fx[ENABLE,START]                        6x4x8         fx-special[ENABLE,STOP]             12x12x16
fx[ENABLE,START]                        6x4x8         fx-middle2[ENABLE,START]            8x12x16
fx[ENABLE,START]                        6x4x8         fx-interactive[ENABLE,START]        4x12x16
fx[ENABLE,START]                        6x4x8         fx-small[ENABLE,START]              4x12x16
fx[ENABLE,START]                        6x4x8         fx-debug[ENABLE,START]              4x12x16
fx[ENABLE,START]                        6x4x8         fx-middle[ENABLE,START]             8x12x16
[username@flow-fx01 ~]$
```

Tofuの X軸、Y軸、Z軸のサイズを表す。

用意されているリソースグループ

各リソースグループのX軸、Y軸、Z軸のノード数を表す。プログラムの実行で必要なノード数は、このサイズ内で指定する。

    - pjstat2コマンドによる確認

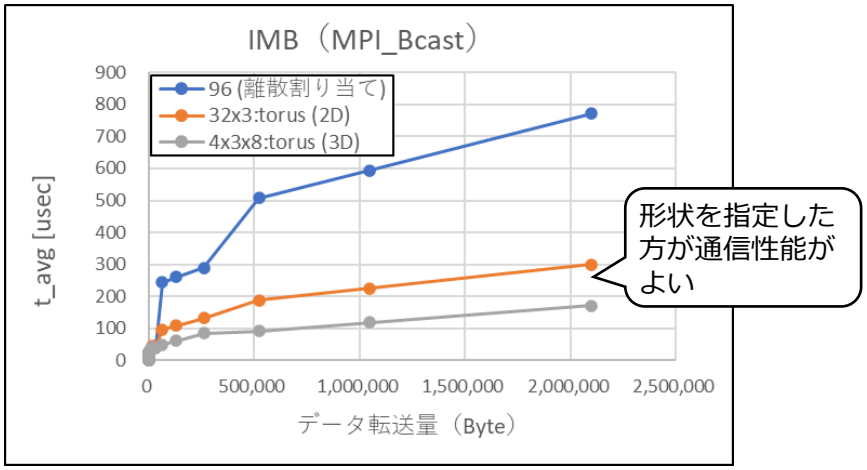
```
[username@flow-fx01 ~]$ pjstat2 --rsc
RSCGRP              STATUS              NODE
fx-interactive      [ENABLE,START]      768:4x12x16
fx-small            [ENABLE,START]      768:4x12x16
fx-debug            [ENABLE,START]      768:4x12x16
fx-middle           [ENABLE,START]      1536:8x12x16
fx-middle2          [ENABLE,START]      1536:8x12x16
fx-large            [ENABLE,START]      1536:8x12x16
fx-xlarge           [ENABLE,START]      1536:8x12x16
fx-special          [ENABLE,STOP]       2304:12x12x16
[username@flow-fx01 ~]$
```

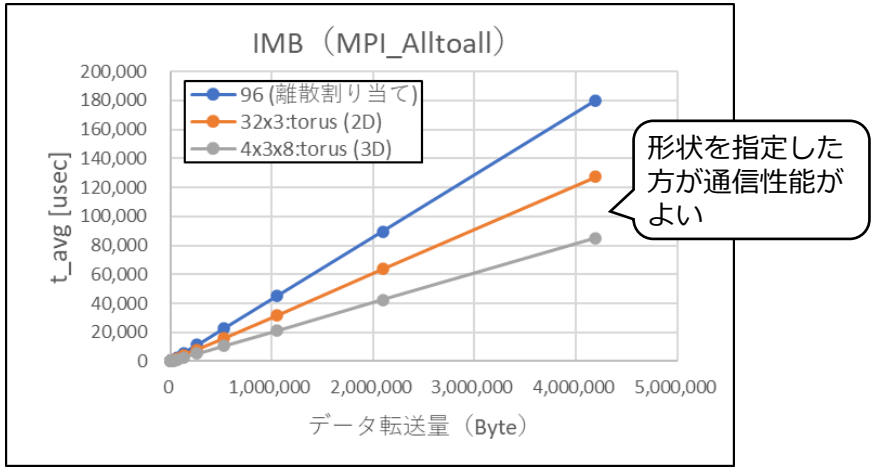
用意されているリソースグループ

各リソースグループのX軸、Y軸、Z軸のノード数を表す。プログラムの実行で必要なノード数は、このサイズ内で指定する。

3. 通信性能の確認（1/6）

- 通信性能の確認
  - 実行条件
    - 使用プログラム：IMBのMPI_Bcast, MPI_Alltoall
    - 96プロセス実行（96ノード使用）
    - 1次元、2次元、3次元による性能の確認
      - 1次元　node=96　← デフォルトは離散割り当て（noncont）
      - 2次元　node=32x3:torus
      - 3次元　node=4x3x8:torus





MPI通信の性能を上げるには、ノードの形状指定が有効であることが分かる

3. 通信性能の確認（2/6）

- MPI通信のコストの調査方法
  - 詳細プロファイラ(fapp)で確認する

    詳細プロファイラの出力結果にMPIのプロファイル結果がある。プログラムで使用されているMPI関数の種別やコスト情報等が確認できる。

```
MPI profile

    ********************************************************************************
        Application
    ********************************************************************************
      Kind   Elapsed(s)       Wait(s)          Byte          Call (        0-4K        4K-64K      64K-1024K     1024KByte-)
    --------------------------------------------------------------------------------------------------------------------------------
           10231.7422     2462.3977          ----     125513560      99447368      15782712       7828920       2454560   all 0
    --------------------------------------------------------------------------------------------------------------------------------
       AVG     0.0093        0.0000    18969.4664     1675.0208     1213.4375      350.1667      105.1667        6.2500   MPI_Send
       MAX     0.2035        0.0000    19863.3741       38468         27390          8404          2524           150
       MIN     0.0001        0.0000        4.0000          77            77             0             0             0
       ----------------------------------------------------------------------------------------------------------------------------
       AVG     1.0980        0.0000    19855.0201   179233.3958   127536.0625    39218.6680    11778.6670      700.0000   MPI_Recv
       MAX     3.0537        0.0000    19863.3741       541093        386001         117656         35336          2100
       MIN     0.4225        0.0000    19730.5484        76782         54626          16808          5048           300
       ----------------------------------------------------------------------------------------------------------------------------
       AVG     0.6574        0.0000    39726.7482    87979.3750    57777.5000    19259.1660    10211.6670      731.0417   MPI_Sendrecv

                                        <省略>

    ********************************************************************************
        Process 0
    ********************************************************************************
           Elapsed(s)       Wait(s)          Byte          Call (        0-4K        4K-64K      64K-1024K     1024KByte-)
    --------------------------------------------------------------------------------------------------------------------------------
            198.4422        6.4302          ----       7013809       5428663        890824        548944        145378   all 0
    --------------------------------------------------------------------------------------------------------------------------------
              0.2035        0.0000    19863.3741         38391         27313          8404          2524           150   MPI_Send
              3.0537        0.0000    19730.5484        541093        386001        117656         35336          2100   MPI_Recv
              1.7642        0.0000    39726.7482        230346        151272         50424         26736          1914   MPI_Sendrecv

                                        <省略>
```

詳細プロファイラのデータ採取方法、結果出力の方法は次頁を参照

3. 通信性能の確認（3/6）

FUJITSU

- MPI通信のコストの調査方法
  - 詳細プロファイラの確認方法

    詳細プロファイラの結果を確認するには、プロファイルデータの採取と結果出力を行う必要がある。

    - 詳細プロファイルデータの採取方法

      プログラムの実行時にfappコマンドを付けると、プロファイルデータが採取される。

実行ジョブスクリプト例

```
#!/bin/bash
#------ pjsub option --------#
#PJM -L rscunit=fx
#PJM -L rscgrp=fx-xlarge
#PJM -L elapse=1:00:00
#PJM -L node=256
#PJM --mpi proc=256
#PJM -s

#------- Program execution -------#
fapp -C -d ./profile_data mpiexec ./a.out
```

詳細プロファイラの詳細については「プロファイラ使用手引書」を参照のこと
プロファイラ使用手引書はHPC Portalより入手することが可能
HPC Portal(https://portal.cc.nagoya-u.ac.jp/cgi-bin/hpcportal.ja/index.cgi)の
「言語製品」
→「FX1000 言語製品」
→「開発支援ツール」
→「プロファイラ使用手引書」

このジョブスクリプトを実行すると
profile_dataというディレクトリに詳
細プロファイルのデータが採取される

    - 詳細プロファイラの結果出力方法

      fapppxコマンドを使用して結果を出力する

```
[username@flow-fx01 ~]$ fapppx -A -d ./profile_data -o ./prof_result.txt
[username@flow-fx01 ~]$
```

profile_dataディレクトリの詳細プロファ
イルデータからprofile_result.txtというテ
キストファイルに結果が出力される

3. 通信性能の確認（4/6）

- MPI通信の通信内容の確認方法
  - MPI統計情報を取得する

    実行時のmcaオプションでMPIの統計情報を取ることが可能。MPIに関する統計情報が確認できる。

```
================================================================
/****************** MPI Statistical Information ******************/
================================================================

------------------------ MPI Information ------------------------
Dimension                         1
Shape                            48

---------------------- MPI Memory Usage (MiB) ---------------------
Estimated_Memory_Size        110.46

------------------- Per-peer Communication Count ------------------
In_Node                           0
Neighbor                          0
Not_Neighbor                      0
Total_Count                       0
Connection                       47
Max_Hop                           6
Average_Hop                    3.23

----------------- Per-peer Transmission Size (MiB) ----------------
In_Node                        0.00
Neighbor                       0.00
Not_Neighbor                   0.00
Total_Size                     0.00

----------------- Per-protocol Communication Count ----------------
Eager                             0
Rendezvous                        0
Persistent_Extended_IF            0
```

```
Unexpected_Message                1

------------------- Barrier Communication Count -------------------
Tofu                         192625
Soft                          38583

----------- Tofu Barrier Collective Communication Count -----------
Bcast                           122
Reduce                            0
Allreduce                       504

--------- 6D-Tofu-specific Collective Communication Count ---------
Alltoall                          0

----------- Tofu-specific Collective Communication Count ----------
Bcast                             0
Reduce                            0
Gather                            0
Gatherv                           0
Allreduce                         0
Alltoall                     116457
Alltoallv                         0
Allgather                         0
Allgatherv                        0

--------- Non-Tofu-specific Collective Communication Count --------
Bcast                            25
Reduce                            0
Gather                          144

                          ＜省略＞
```

MPI統計情報の出力方法は次頁を参照

3. 通信性能の確認（5/6）

- MPI通信の通信内容の確認方法
  - MPI統計情報を取得する
    - 実行時のmcaオプションでMPIの統計情報を取ることが可能
    - 実行プログラムの再コンパイルの必要はない
    - 標準エラーに出力。ジョブ実行の場合は .errファイルに出力される。

実行ジョブスクリプト例

```
#!/bin/bash
#------ pjsub option --------#
#PJM -L rscunit=fx
#PJM -L rscgrp=fx-xlarge
#PJM -L elapse=1:00:00
#PJM -L node=256
#PJM --mpi proc=256
#PJM -s

#------- Program execution -------#

mpiexec -mca mpi_print_stats 2 ./a.out
```

MPI統計情報については「MPI使用手引書」6.15 MPI統計情報 を参照のこと
MPI使用手引書はHPC Portalより入手することが可能
HPC Portal(https://portal.cc.nagoya-u.ac.jp/cgi-bin/hpcportal.ja/index.cgi)の
「言語製品」
→「FX1000 言語製品」
→「MPI」
→「MPI使用手引書」

3. 通信性能の確認（6/6）

FUJITSU

- ノードの形状に関係する部分（本日の説明の範囲）

TypeIのバッチジョブスクリプト例

```
#!/bin/bash

#------ pjsub option --------#
#PJM -L rscunit=fx
#PJM -L rscgrp=fx-xlarge
#PJM -L elapse=1:00:00
#PJM -L node=4x4x4:torus
#PJM --mpi proc=256
#PJM --mpi rank-map-hostfile=rankmap.txt
#PJM --mpi rank-map-bynode
#PJM -j
#PJM -s

#------- Program execution -------#
mpiexec ./a.out
```

ノード形状の指定

ノード形状を指定するとプロセスがどのように配置されるのか

ランクマップの指定

ランクマップを使用したプロセス配置（上級者向き）

プロセスの配置方法の指定

1ノードに複数プロセスの場合プロセスの配置がどうなるか

ジョブの統計情報出力

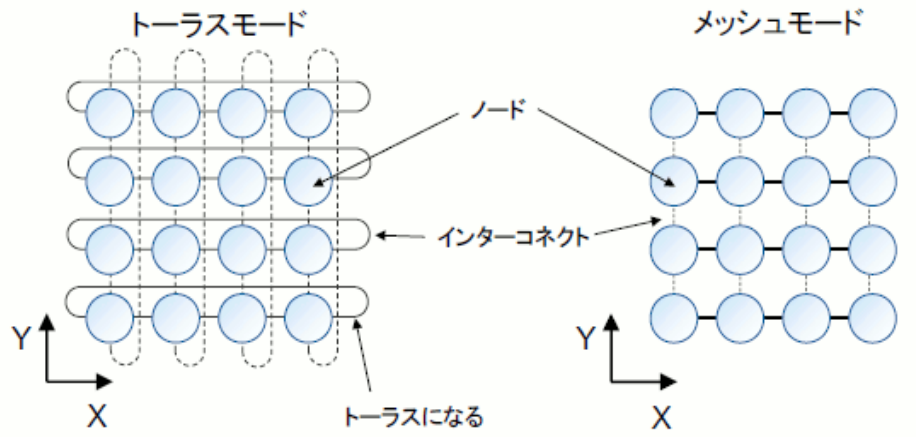
ジョブの統計情報を使ったノード座標、Tofu座標の確認方法

4. ノード形状の指定方法（1/2）

FUJITSU

- ノードの配置方法

ノードの配置方法	説明
メッシュモード (mesh)	最少割り当て単位は1ノード Tofu座標上で隣接するノードが選択
トーラスモード (torus)	最少割り当て単位はTofu単位(12ノード) Tofu座標上で隣接するノードが選択
離散割り当て (noncont) デフォルト	最少割り当て単位は1ノード できるだけTofu座標上で隣接するように選択 以下の場合は隣接するノードが選択されない ・隣接する空きノードが無い場合 ・隣接しないノードを選択することでジョブの実行開始が早められる場合



4. ノード形状の指定方法（2/2）

FUJITSU

- rank-map-bynode
  - 計算ノードに1プロセスを生成すると、次の計算ノードに移動して、1プロセスを生成する
- rank-map-bychip(デフォルト)
  - 計算ノードにnプロセスを生成すると、次の計算ノードに移動して、nプロセスを生成する
- rank-map-hostfile
  - ホストファイルに指定された座標を元に、ランクを割り当てる

5. MPIランクの割り当て（1/6）

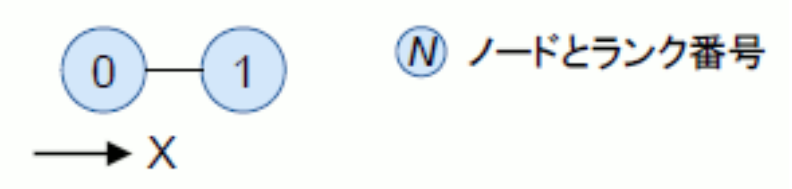
FUJITSU

- ランクの割り当て方法（１次元）
  - 座標の原点をランク0とし、ノード割り当てはランク0のノードから隣接するノードに移動する。
  - 1ノードに1つのランク(プロセス)を生成する場合は、 rank-map-bychip, rank-map-bynodeどちらを指定してもノードの割り当て方は同じ。
  - どちらも指定しない場合は、デフォルトのrank-map-bychipとなる

```
--mpi rank-map-bynode
--mpi rank-map-bychip
```

  - 例

```
#PJM -L  node=2
#PJM --mpi proc=2
#PJM --mpi rank-map-bynode
```



5. MPIランクの割り当て（2/6）

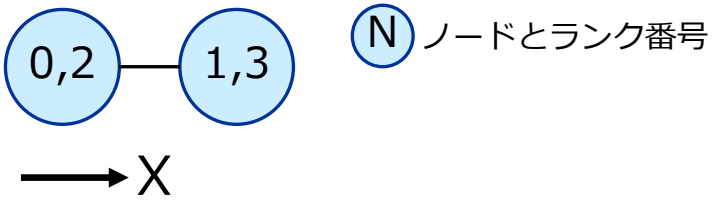
FUJITSU

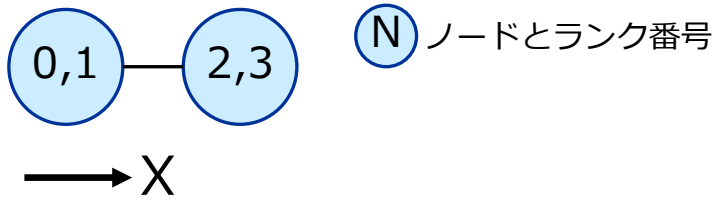
- ランクの割り当て方法（１次元）

割り当てノードの順番	割り当てるランク	
	rank-map-bynodeの場合	rank-map-bychipの場合
0番	ランク0と2	ランク0と1
1番	ランク1と3	ランク2と3

```
#PJM -L  node=2
#PJM --mpi proc=4
#PJM --mpi rank-map-bynode
```

```
#PJM -L  node=2
#PJM --mpi proc=4
#PJM --mpi rank-map-bychip
```





5. MPIランクの割り当て（3/6）

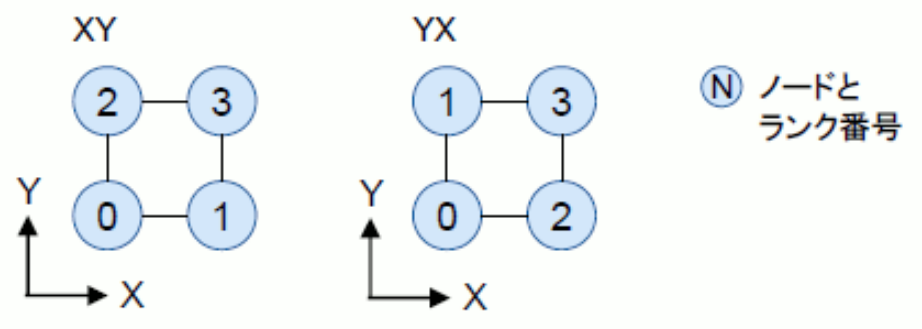
FUJITSU

- ランクの割り当て方法（２次元）
  - 形状が2次元の場合は、ランクの割り当て方法を2種類(XY|YX)から選択できる。
  - (XY|YX)の指定がない場合はXYとなる。

```
--mpi rank-map-bynode[={XY|YX}]
--mpi rank-map-bychip[={XY|YX}]
```

  - 例

```
#PJM -L  node=2x2:torus
#PJM --mpi proc=4
#PJM --mpi rank-map-bynode=XY
```



5. MPIランクの割り当て（4/6）

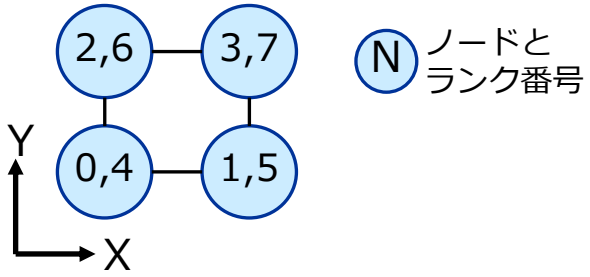
FUJITSU

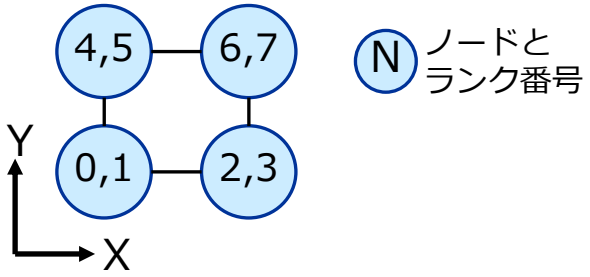
- ランクの割り当て方法（２次元）

割り当てノードの順番	割り当てるランク	
	rank-map-bynodeの場合	rank-map-bychipの場合
0番	ランク0と4	ランク0と1
1番	ランク1と5	ランク2と3
2番	ランク2と6	ランク4と5
3番	ランク3と7	ランク6と7

```
#PJM -L  node=2x2:torus
#PJM --mpi proc=8
#PJM --mpi rank-map-bynode
```

```
#PJM -L  node=2x2:torus
#PJM --mpi proc=8
#PJM --mpi rank-map-bychip
```





5. MPIランクの割り当て（5/6）

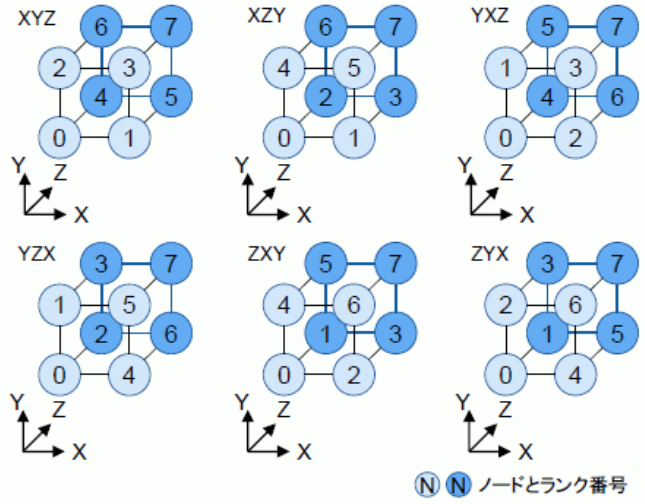
FUJITSU

- ランクの割り当て方法（３次元）
  - 形状が3次元の場合は、ランクの割り当て方法を6種類から選択できる。
  - (XYZ|XZY|YXZ|YZX|ZXY|ZYX)の指定がない場合はXYZとなる。

```
--mpi rank-map-bynode[={XYZ|XZY|YXZ|YZX|ZXY|ZYX}]
--mpi rank-map-bychip[={XYZ|XZY|YXZ|YZX|ZXY|ZYX}]
```

  - 例

```
#PJM -L  node=2x2x2:torus
#PJM --mpi proc=8
#PJM --mpi rank-map-bynode=XYZ
```



5. MPIランクの割り当て（6/6）

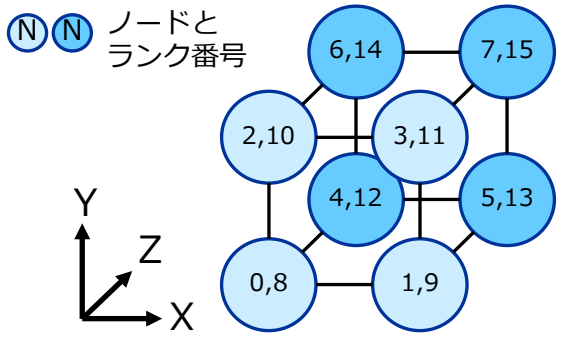
FUJITSU

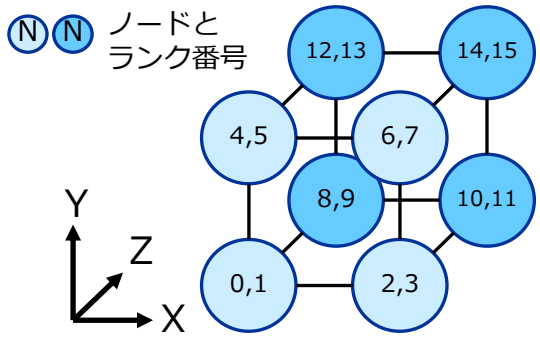
- ランクの割り当て方法（３次元）

割り当てノードの順番	割り当てるランク	
	rank-map-bynodeの場合	rank-map-bychipの場合
0番	ランク0と8	ランク0と1
1番	ランク1と9	ランク2と3
2番	ランク2と10	ランク4と5
3番	ランク3と11	ランク6と7
4番	ランク4と12	ランク8と9
5番	ランク5と13	ランク10と11
6番	ランク6と14	ランク12と13
7番	ランク7と15	ランク14と15

```
#PJM -L  node=2x2x2:torus
#PJM --mpi proc=16
#PJM --mpi rank-map-bynode
```

```
#PJM -L  node=2x2x2:torus
#PJM --mpi proc=16
#PJM --mpi rank-map-bychip
```





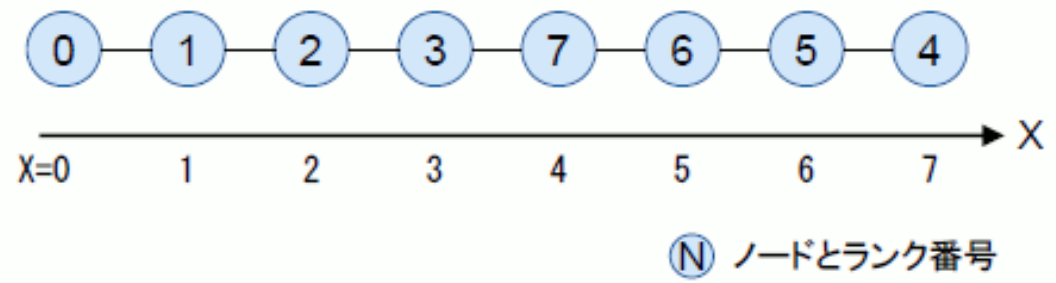
6. ランクマップファイル（1/3）

FUJITSU

- ランクマップファイル（１次元）
  - 形状が１次元の場合のランクマップファイルの例

```
[username@flow-fx01 ~]$ cat rankmapfile-1
(0)     <---- ランク0
(1)     <---- ランク1
(2)     <---- ランク2
(3)     <---- ランク3
(7)     <---- ランク4
(6)     <---- ランク5
(5)     <---- ランク6
(4)     <---- ランク7
```

```
#PJM -L  node=8:torus
#PJM --mpi proc=8
#PJM --mpi rank-map-hostfile=rankmapfile-1
```



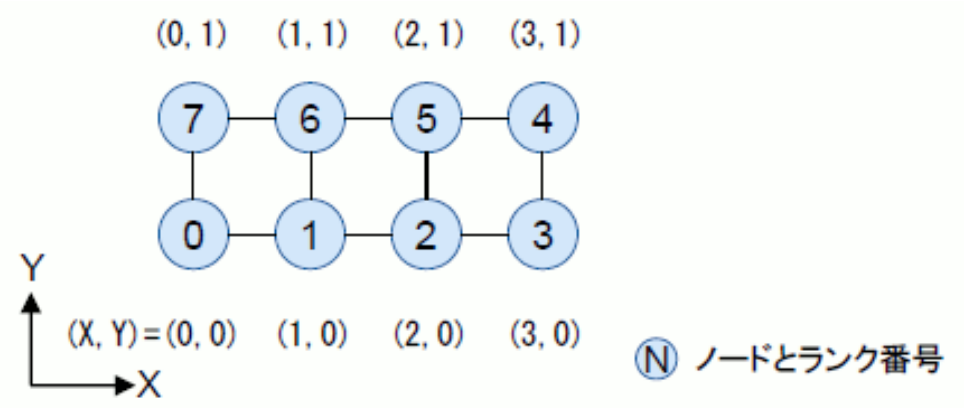
6. ランクマップファイル（2/3）

FUJITSU

- ランクマップファイル（２次元）
  - 形状が２次元の場合のランクマップファイルの例

```
[username@flow-fx01 ~]$ cat rankmapfile-2
(0,0)     <----- ランク0
(1,0)     <----- ランク1
(2,0)     <----- ランク2
(3,0)     <----- ランク3
(3,1)     <----- ランク4
(2,1)     <----- ランク5
(1,1)     <----- ランク6
(0,1)     <----- ランク7
```

```
#PJM -L  node=4x2:torus
#PJM --mpi proc=8
#PJM --mpi rank-map-hostfile=rankmapfile-2
```



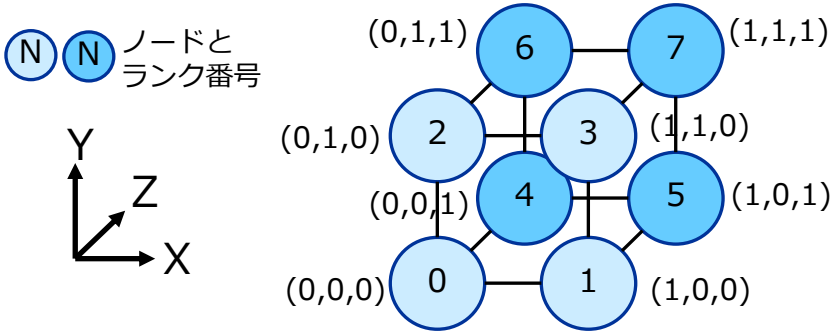
6. ランクマップファイル（3/3）

FUJITSU

- ランクマップファイル（３次元）
  - 形状が３次元の場合のランクマップファイルの例

```
[username@flow-fx01 ~]$ cat rankmapfile-3
(0,0,0)     <----- ランク0
(1,0,0)     <----- ランク1
(0,1,0)     <----- ランク2
(1,1,0)     <----- ランク3
(0,0,1)     <----- ランク4
(1,0,1)     <----- ランク5
(0,1,1)     <----- ランク6
(1,1,1)     <----- ランク7
```

```
#PJM -L  node=2x2x2:torus
#PJM --mpi proc=8
#PJM --mpi rank-map-hostfile=rankmapfile-3
```



7. ノード形状の確認方法（1/4）

FUJITSU

- ジョブの統計情報を取得する
  - ジョブの統計情報の取得方法（ジョブスクリプト例）

```
#!/bin/bash
#PJM -L rscunit=fx
#PJM -L rscgrp=fx-small
#PJM -L elapse=0:10:00
#PJM -L node=2x3x2:torus
#PJM --mpi proc=12
#PJM -s

mpiexec ./a.out
```

ジョブ統計情報を出力するオプション

  - pjsub コマンドの -s または -S オプションで、ジョブの実行結果としてジョブ統計情報を出力。-S オプションは -s オプションで出力される統計情報に加えてノード毎の統計情報も出力する。
  - ジョブ実行後にファイル拡張子statsのテキストファイルが生成される

7. ノード形状の確認方法（2/4）

- ジョブの統計情報の出力内容を取得する
  - ジョブの統計情報の出力例

```
Job Statistical Information

 JOB ID                    : 1005143
 SUB JOB NUM               : -
 START BULKNO              : -
 END BULKNO                : -
 HOST NAME                 : flow-fx01
 JOB NAME                  : go_4x3x4.sh
 JOB TYPE                  : BATCH
 JOB MODEL                 : NM
 USER                      : fj-se
 GROUP                     : fj
 RESOURCE UNIT             : fx
 RESOURCE GROUP            : fx-middle
  〈略〉
 NODE NUM (REQUIRE)        : 48:4x3x4
 NODE NUM (ALLOC)          : 48:4x3x4
 NODE NUM (USE)            : 32
 NODE NUM (UNUSED)         : 0
 NODE ID (USE)             : 0xFF2B0001 0xFF2E0005 0xFF2E0001 0xFF2B0007 0xF                FF300001 0xFF2D0007
                             0xFF2C0001 0xFF2F0005 0xFF2F0001 0xFF2C0007 0xF                FF2E0008 0xFF2B0008
                             0xFF2D0005 0xFF300006 0xFF300008 0xFF2D0008 0xF                FF2F0008 0xFF2C0008
                             0xFF2B000D 0xFF2E000E 0xFF2E0010 0xFF2B0010 0xFF2D000D 0xFF30000E 0xFF300010 0xFF2D0010
 TOFU COORDINATE (USE)     : (2,1,6) (3,1,6) (3,1,6) (2,1,6) (2,1,6)                1,6) (3,1,6) (3,1,6)
                             (2,1,6) (2,1,7) (3,1,7) (3,1,7) (2,1,7) (2,1,7)                1,7) (2,1,7) (3,1,7)
                             (3,1,7) (2,1,7) (2,1,7) (3,1,7) (3,1,7) (2,1,7)                1,7) (2,1,7)
 NODE ID (RANK)            : 0xFF2B0001(0) 0xFF2B0001(1) 0xFF2E0005(2) 0xFF2E0005(3) 0xFF2E0001(4) 0xFF2E0001(5)
                             0xFF2B0007(6) 0xFF2B0007(7) 0xFF2D0001(8) 0xFF2D0001(9) 0xFF300005(10) 0xFF300005(11)
                             0xFF300001(12) 0xFF300001(13) 0xFF2D0007(14) 0xFF2D0007(15) 0xFF2C0001(16) 0xFF2C0001(17)
                             0xFF2F0005(18) 0xFF2F0005(19)                7(22) 0xFF2C0007(23)
                             0xFF2B0005(24) 0xFF2B0005(25) 0x                8(28) 0xFF2E0008(29)
                             0xFF2B0008(30) 0xFF2B0008(31) 0x                6(34) 0xFF300006(35)
                             0xFF300008(36) 0xFF300008(37) 0xFF2D0008(38) 0xFF2D0008(39) 0xFF2C0005(40) 0xFF2C0005(41)
                             0xFF2F0006(42) 0xFF2F0006(43) 0xFF2F0008(44) 0xFF2F0008(45) 0xFF2C0008(46) 0xFF2C0008(47)
                             0xFF2B000D(48) 0xFF2B000D(49) 0xFF2E000E(50) 0xFF2E000E(51) 0xFF2E0010(52) 0xFF2E0010(53)
                             0xFF2B0010(54) 0xFF2B0010(55) 0xFF2D000D(56) 0xFF2D000D(57) 0xFF30000E(58) 0xFF30000E(59)
                             0xFF300010(60) 0xFF300010(61) 0xFF2D0010(62) 0xFF2D0010(63)
```

要求ノードと形状

割り当てノードと形状

使用ノード数

使用されたノードのノードIDリスト

使用されたノードのTofu座標リスト

割り当てられたノードのノードIDとカッコ内はそれに対応するランク番号

7. ノード形状の確認方法（3/4）

FUJITSU

- ノード形状指定
  - リソースグループの情報の表示

```
[username@flow-fx01 ~]$ pjstat --rsc

RSCUNIT                          RSCUNIT_SIZE  RSCGRP                          RSCGRP_SIZE
fx[ENABLE,START]                 6x4x8         fx-large[ENABLE,START]          8x12x16
fx[ENABLE,START]                 6x4x8         fx-xlarge[ENABLE,START]         8x12x16
fx[ENABLE,START]                 6x4x8         fx-bmt[ENABLE,START]            6x12x16
fx[ENABLE,START]                 6x4x8         fx-special[ENABLE,STOP]         12x12x16
fx[ENABLE,START]                 6x4x8         fx-middle2[ENABLE,START]        8x12x16
fx[ENABLE,START]                 6x4x8         fx-interactive[ENABLE,START]    4x12x16
fx[ENABLE,START]                 6x4x8         fx-small[ENABLE,START]          4x12x16
fx[ENABLE,START]                 6x4x8         fx-debug[ENABLE,START]          4x12x16
fx[ENABLE,START]                 6x4x8         fx-middle[ENABLE,START]         8x12x16
$
```

指定できるサイズ

  - ノードの利用状況により、指定した形状が割り当てられない場合は、形状を回転してノードを割り当てる。

ジョブの統計情報

```
Job Statistical Information

 〈略〉
 NODE NUM (REQUIRE)          : 96:8x3x4
 NODE NUM (ALLOC)            : 96:4x3x8
 NODE NUM (USE)              : 96
 NODE NUM (UNUSED)           : 0
 NODE ID (USE)               : 0xFF1F0004 0xFF1F0003
```

形状を回転してノードを割り当てる

7. ノード形状の確認方法（4/4）

FUJITSU

- strictについて
  - 形状を回転させない方法

```
#!/bin/bash
#PJM -L rscunit=fx
#PJM -L rscgrp=fx-large
#PJM -L elapse=00:01:00
#PJM -L node=8x3x4:torus:strict
#PJM --mpi proc=96
#PJM -s

mpiexec ./a.out
```

回転させたくない場合はstrictを付ける

  - リソースグループの設定値より大きい形状を指定するとエラーメッセージが出力される。（例：12x3x4の形状指定でstrictを指定した場合）

```
[username@flow-fx01 ~]$ cat go12x3x4_torus_strict.sh
#!/bin/bash
#PJM -L rscunit=fx
#PJM -L rscgrp=fx-large
#PJM -L elapse=00:01:00
#PJM -L node=12x3x4:torus:strict
#PJM --mpi proc=144
#PJM -s

mpiexec ./a.out

[username@flow-fx01 ~]$
[username@flow-fx01 ~]$ pjsub go12x3x4_torus_strict.sh
[ERR.] PJM 0057 pjsub node=12x3x4 is greater than the upper limit (8x12x16).
$
```

strictを付けているため回転しない

エラーメッセージが出力される

8. MPIランクの割り当て状況（1/4）

FUJITSU

- ランクマップファイルを使った実行（３次元：トーラス）
  - 形状が4x3x4(torus)の場合の状況

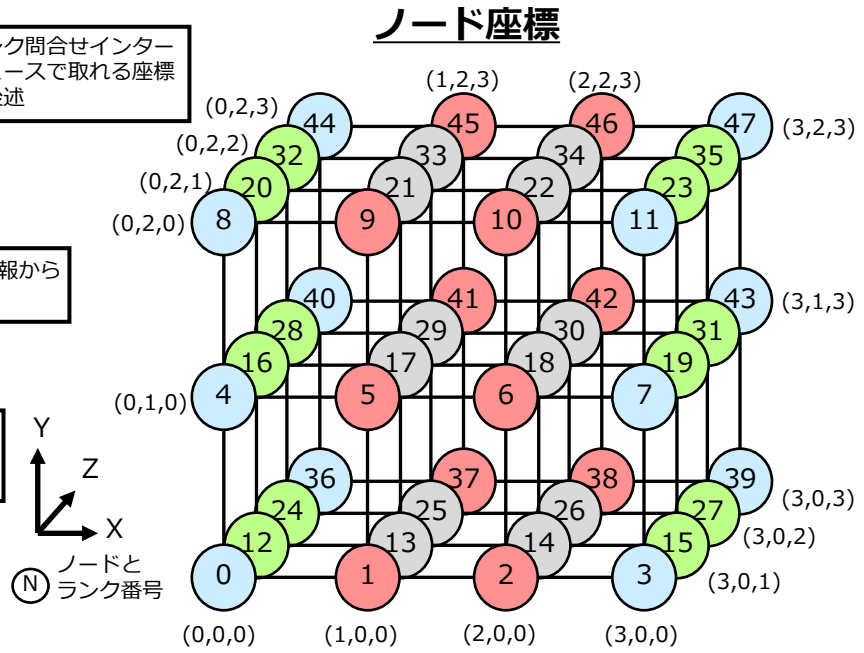
RANK NO	rankmap	NODE COORDINATE	TOFU COORDINATE	TOFU 6次元 (x,y,z,a,b,c)
0	(0,0,0)	(0,0,0)	(2,1,6)	(2,1,6,0,0,0)
1	(1,0,0)	(1,0,0)	(3,1,6)	(3,1,6,0,0,0)
2	(2,0,0)	(2,0,0)	(3,1,6)	(3,1,6,1,0,0)
3	(3,0,0)	(3,0,0)	(2,1,6)	(2,1,6,1,0,0)
4	(0,1,0)	(0,1,0)	(2,1,6)	(2,1,6,0,1,0)
5	(1,1,0)	(1,1,0)	(3,1,6)	(3,1,6,0,2,0)
6	(2,1,0)	(2,1,0)	(3,1,6)	(3,1,6,1,2,0)
7	(3,1,0)	(3,1,0)	(2,1,6)	(2,1,6,1,2,0)
8	(0,2,0)	(0,2,0)	(2,1,6)	(2,1,6,0,0,1)
9	(1,2,0)	(1,2,0)	(3,1,6)	(3,1,6,0,0,1)
10	(2,2,0)	(2,2,0)	(3,1,6)	(3,1,6,1,1,0)
11	(3,2,0)	(3,2,0)	(2,1,6)	(2,1,6,1,1,0)
12	(0,0,1)	(0,0,1)	(2,1,7)	(2,1,7,0,0,0)
13	(1,0,1)	(1,0,1)	(3,1,7)	(3,1,7,0,0,0)
14	(2,0,1)	(2,0,1)	(3,1,7)	(3,1,7,1,0,0)
15	(3,0,1)	(3,0,1)	(2,1,7)	(2,1,7,1,0,0)
16	(0,1,1)	(0,1,1)	(2,1,7)	(2,1,7,0,2,0)
17	(1,1,1)	(1,1,1)	(3,1,7)	(3,1,7,0,2,0)
:	:	:	:	:
44	(0,2,3)	(0,2,3)	(2,1,6)	(2,1,6,0,1,1)
45	(1,2,3)	(1,2,3)	(3,1,6)	(3,1,6,0,1,1)
46	(2,2,3)	(2,2,3)	(3,1,6)	(3,1,6,1,1,1)
47	(3,2,3)	(3,2,3)	(2,1,6)	(2,1,6,1,1,1)

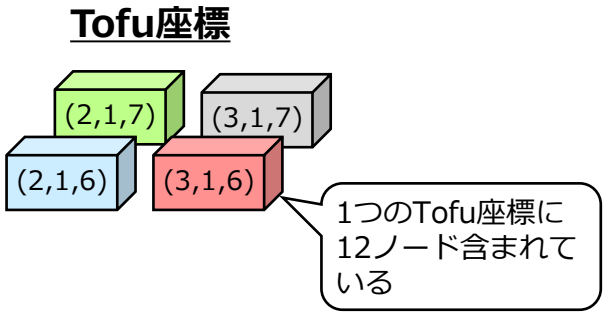
ランク問合せインターフェースで取れる座標 ※後述

ジョブ統計情報から取れる情報

ランクマップファイルで指定した座標

同一TOFU単位で色分け





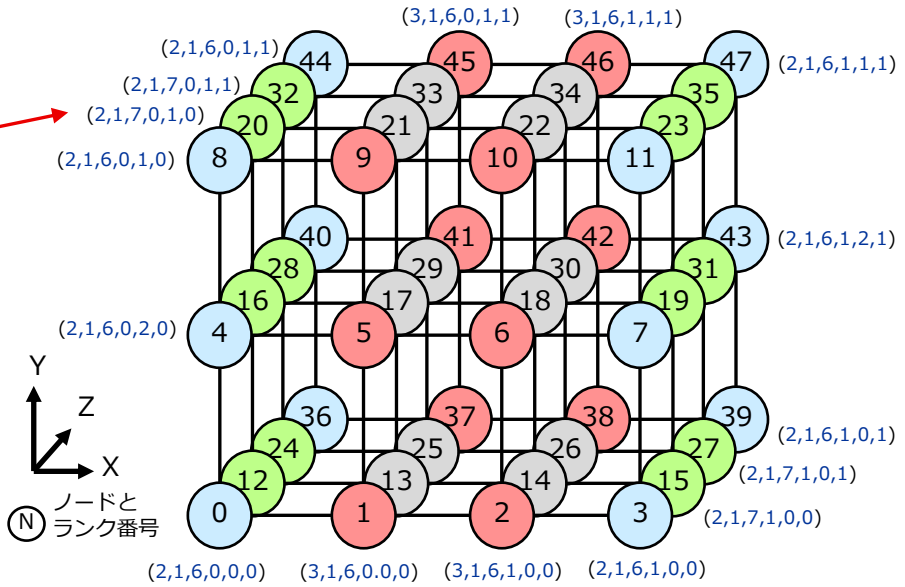
8. MPIランクの割り当て状況（2/4）

FUJITSU

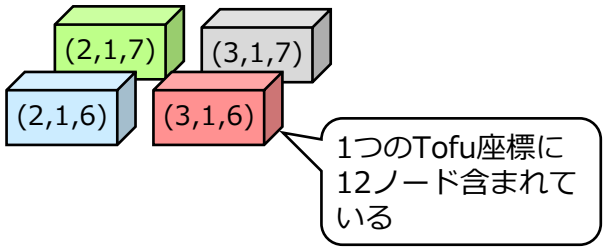
- ランクマップファイルを使った実行（３次元：トーラス）
  - 形状が4x3x4(torus)の場合の状況

RANK NO	rankmap	NODE COORDINATE	TOFU COORDINATE	TOFU 6次元 (x,y,z,a,b,c)
0	(0,0,0)	(0,0,0)	(2,1,6)	(2,1,6,0,0,0)
1	(1,0,0)	(1,0,0)	(3,1,6)	(3,1,6,0,0,0)
2	(2,0,0)	(2,0,0)	(3,1,6)	(3,1,6,1,0,0)
3	(3,0,0)	(3,0,0)	(2,1,6)	(2,1,6,1,0,0)
4	(0,1,0)	(0,1,0)	(2,1,6)	(2,1,6,0,2,0)
5	(1,1,0)	(1,1,0)	(3,1,6)	(3,1,6,0,2,0)
6	(2,1,0)	(2,1,0)	(3,1,6)	(3,1,6,1,2,0)
7	(3,1,0)	(3,1,0)	(2,1,6)	(2,1,6,1,2,0)
8	(0,2,0)	(0,2,0)	(2,1,6)	(2,1,6,0,1,0)
9	(1,2,0)	(1,2,0)	(3,1,6)	(3,1,6,0,1,0)
10	(2,2,0)	(2,2,0)	(3,1,6)	(3,1,6,1,1,0)
11	(3,2,0)	(3,2,0)	(2,1,6)	(2,1,6,1,1,0)
12	(0,0,1)	(0,0,1)	(2,1,7)	(2,1,7,0,0,0)
13	(1,0,1)	(1,0,1)	(3,1,7)	(3,1,7,0,0,0)
14	(2,0,1)	(2,0,1)	(3,1,7)	(3,1,7,1,0,0)
15	(3,0,1)	(3,0,1)	(2,1,7)	(2,1,7,1,0,0)
16	(0,1,1)	(0,1,1)	(2,1,7)	(2,1,7,0,2,0)
17	(1,1,1)	(1,1,1)	(3,1,7)	(3,1,7,0,2,0)
:	:	:	:	:
44	(0,2,3)	(0,2,3)	(2,1,6)	(2,1,6,0,1,1)
45	(1,2,3)	(1,2,3)	(3,1,6)	(3,1,6,0,1,1)
46	(2,2,3)	(2,2,3)	(3,1,6)	(3,1,6,1,1,1)
47	(3,2,3)	(3,2,3)	(2,1,6)	(2,1,6,1,1,1)

ノード座標（６次元：x,y,z,a,b,c）

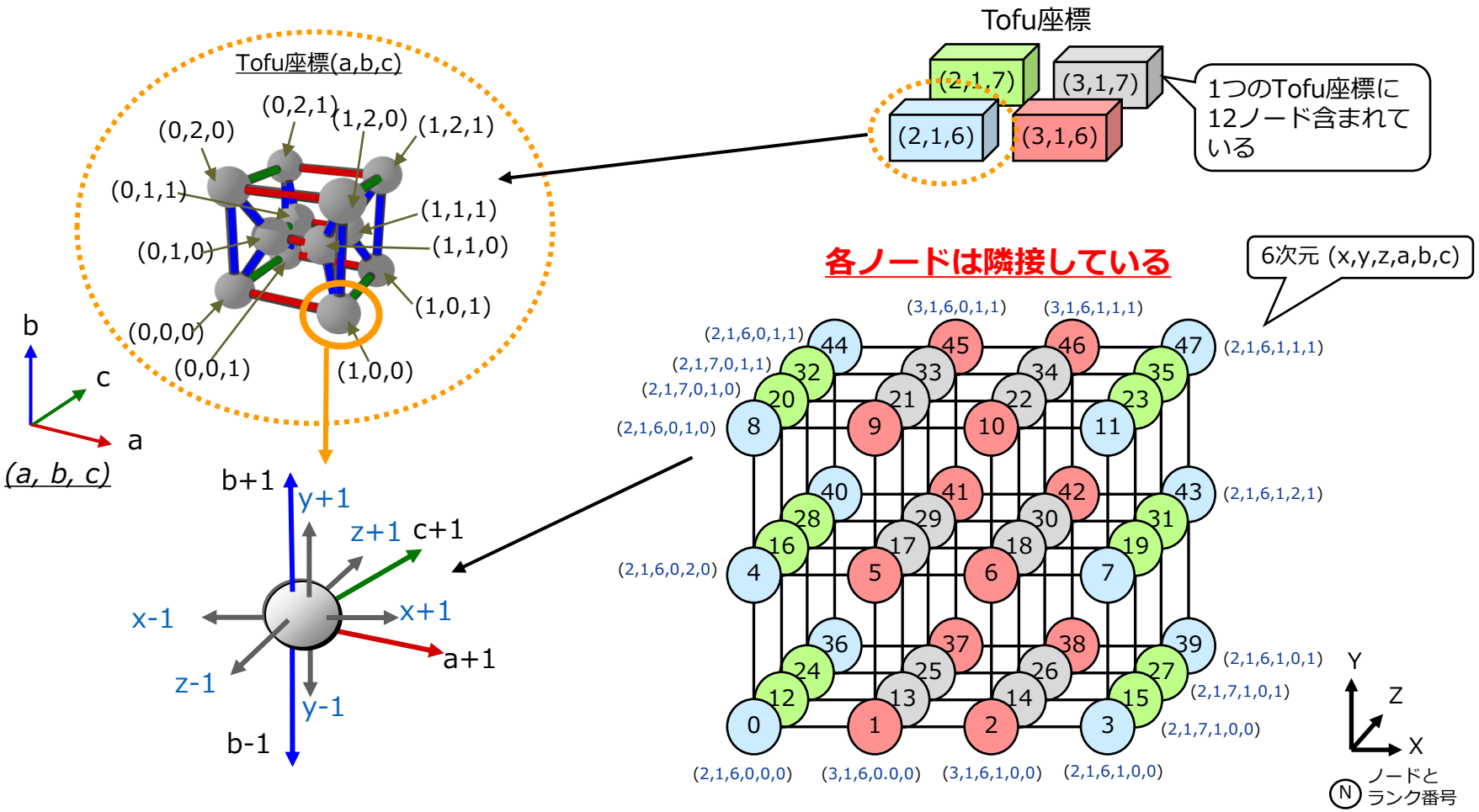


Tofu座標



8. MPIランクの割り当て状況（3/4）

- ランクマップファイルを使った実行（３次元：トーラス）
  - 形状が4x3x4(torus)の場合の状況



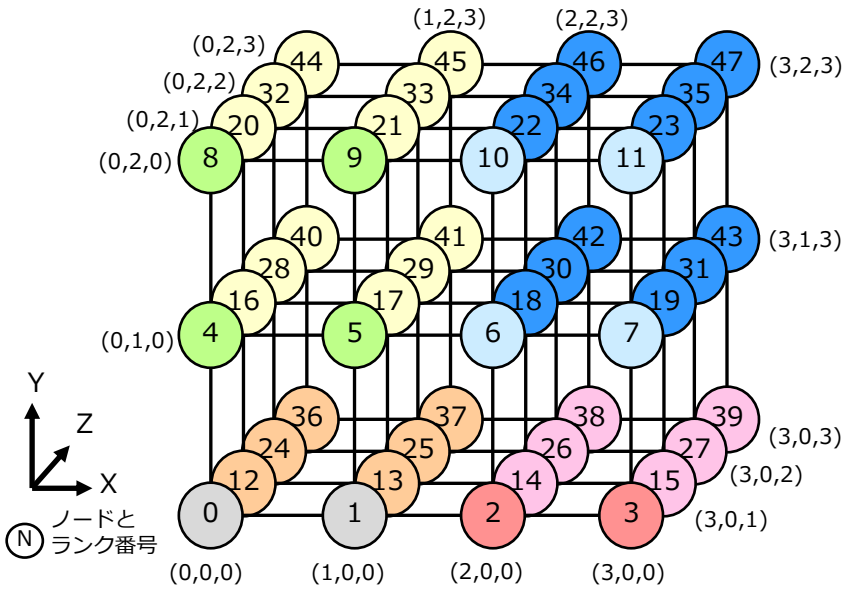
8. MPIランクの割り当て状況（4/4）

FUJITSU

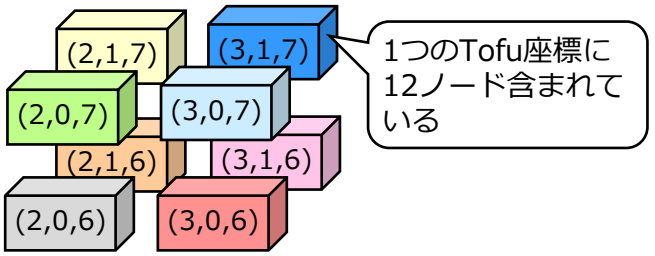
- ランクマップファイルを使った実行（３次元：メッシュ）
  - 形状が4x3x4(mesh)の場合の状況

RANK NO	rankmap	NODE COORDINATE	TOFU COORDINATE	TOFU 6次元 (x,y,z,a,b,c)
0	(0,0,0)	(0,0,0)	(2,0,6)	(2,0,6,0,2,1)
1	(1,0,0)	(1,0,0)	(2,0,6)	(2,0,6,1,2,1)
2	(2,0,0)	(2,0,0)	(3,0,6)	(3,0,6,1,2,1)
3	(3,0,0)	(3,0,0)	(3,0,6)	(3,0,6,0,2,1)
4	(0,1,0)	(0,1,0)	(2,0,7)	(2,0,7,0,2,1)
5	(1,1,0)	(1,1,0)	(2,0,7)	(2,0,7,1,2,1)
6	(2,1,0)	(2,1,0)	(3,0,7)	(3,0,7,1,2,1)
7	(3,1,0)	(3,1,0)	(3,0,7)	(3,0,7,0,2,1)
8	(0,2,0)	(0,2,0)	(2,0,7)	(2,0,7,0,2,0)
9	(1,2,0)	(1,2,0)	(2,0,7)	(2,0,7,1,2,0)
10	(2,2,0)	(2,2,0)	(3,0,7)	(3,0,7,1,2,0)
11	(3,2,0)	(3,2,0)	(3,0,7)	(3,0,7,0,2,0)
12	(0,0,1)	(0,0,1)	(2,1,6)	(2,1,6,0,2,1)
13	(1,0,1)	(1,0,1)	(2,1,6)	(2,1,6,1,2,1)
14	(2,0,1)	(2,0,1)	(3,1,6)	(3,1,6,1,2,1)
15	(3,0,1)	(3,0,1)	(3,1,6)	(3,1,6,0,2,1)
16	(0,1,1)	(0,1,1)	(2,1,7)	(2,1,7,0,2,1)
17	(1,1,1)	(1,1,1)	(2,1,7)	(2,1,7,1,2,1)
:	:	:	:	:
44	(0,2,3)	(0,2,3)	(2,1,7)	(2,1,7,0,0,0)
45	(1,2,3)	(1,2,3)	(2,1,7)	(2,1,7,1,0,0)
46	(2,2,3)	(2,2,3)	(3,1,7)	(3,1,7,1,0,0)
47	(3,2,3)	(3,2,3)	(3,1,7)	(3,1,7,0,0,0)

ノード座標



Tofu座標



9. 付録

- ランク問合せインターフェース
- 12x3x4の形状指定例
- 12x12の形状指定例
- 6x6x4の形状指定例

ランク問合せインターフェース（1/2）

FUJITSU

- 座標の問合せ関数
  - 指定されたコミュニケータとそのコミュニケータにおけるプロセスのランクに対応する論理座標またはTofu座標の取得が可能
    - Fortran書式

```
USE MPI_EXT
FJMPI_TOPOLOGY_GET_COORDS(COMM, RANK, VIEW, MAXDIMS, COORDS, IERROR)
INTEGER COMM, RANK, VIEW, MAXDIMS, COORDS(*), IERROR
```

    - C言語書式

```
#include <mpi-ext.h>
int FJMPI_Topology_get_coords(MPI_Comm comm, int rank, int view, int maxdims, int coords[])
```

    - 引数説明

型	変数	説明	IN/OUT
MPI_Comm	comm	コミュニケータを指定	IN
int	rank	コミュニケータ内のランクを指定	IN
int	view	論理座標かTofu座標かを表すマクロを指定 FJMPI_LOGICAL: 論理座標 FJMPI_TOFU_SYS: Tofu座標(実際に割り当てられた座標) FJMPI_TOFU_REL: Tofu座標(引数commのランク0を基準とした相対座標)	IN
int	maxdim	取得する座標の次元数を指定 viewがFJMPI_LOGICALの場合: 1～3を指定 viewがFJMPI_LOGICAL以外の場合: 1～6を指定	IN
int[]	coords	コミュニケータとランクに対応する座標の配列	OUT

ランク問合せインターフェース（2/2）

FUJITSU

- 座標の問合せ関数の使用例
  - サンプルプログラムの実行条件
    - ① 4ノード4プロセス実行（1ノードあたり1プロセス）
    - ② 以下のTofu座標（6次元座標）を取得
      - 実際に割り当てられた座標
      - ランク０を基準とした相対座標

サンプルプログラム

```
program main
use MPI_EXT

integer myrank,ierr
integer coords(6)
integer coords2(6)

call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)

call FJMPI_TOPOLOGY_GET_COORDS(MPI_COMM_WORLD, myrank, &
                               FJMPI_TOFU_SYS, 6, coords, ierr)
call FJMPI_TOPOLOGY_GET_COORDS(MPI_COMM_WORLD, myrank, &
                               FJMPI_TOFU_REL, 6, coords2, ierr)
write(*,'(a,i4,a,6i2,a,6i2)') &
         "rank=",myrank,"  coords=",coords,"  coords2=",coords2

call MPI_FINALIZE(ierr)
end
```

実行結果

```
rank=    0  coords= 4 0 3 0 0 1  coords2= 0 0 0 0 0 0
rank=    1  coords= 4 0 3 0 1 1  coords2= 0 0 0 0 1 0
rank=    2  coords= 4 0 3 1 0 1  coords2= 0 0 0 1 0 0
rank=    3  coords= 4 0 3 1 1 1  coords2= 0 0 0 1 1 0
```

実際に割り当てられたTofu座標
6次元 (x,y,z,a,b,c)

ランク０を基準とした相対座標
6次元 (x,y,z,a,b,c)

付録：12x3x4の形状指定例 （1/3）

FUJITSU

- 12x3x4:torusの形状指定例
  - 実行条件
    - 形状は12x3x4:torus
    - strict指定は無し
    - プロセス数は144（1ノード1プロセス）
    - ランクマップファイルを使用

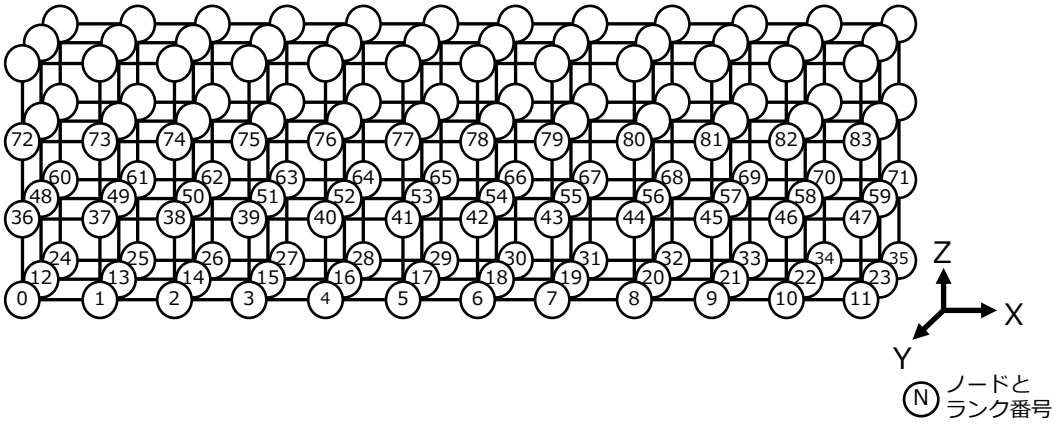
実行スクリプト

```
#!/bin/bash
#PJM -L rscunit=fx
#PJM -L rscgrp=fx-large
#PJM -L elapse=00:01:00
#PJM -L node=12x3x4:torus
#PJM --mpi proc=144
#PJM --mpi rank-map-hostfile=rankmap_12x3x4
#PJM -s

mpiexec ./a.out
```

ランクマップファイル(rankmap_12x3x4)

```
(  0,   0,   0)  ← ランク0
(  1,   0,   0)  ← ランク1
(  2,   0,   0)  ← ランク2
(  3,   0,   0)  ← ランク3
(  4,   0,   0)  ← ランク4
(  5,   0,   0)
(  6,   0,   0)
(  7,   0,   0)
(  8,   0,   0)
(  9,   0,   0)
( 10,   0,   0)
( 11,   0,   0)
(  0,   1,   0)
(  1,   1,   0)
(  2,   1,   0)
        :
        :
(  8,   2,   3)  ← ランク140
(  9,   2,   3)  ← ランク141
( 10,   2,   3)  ← ランク142
( 11,   2,   3)  ← ランク143
```

ランクマップファイルによるノード形状とプロセス配置



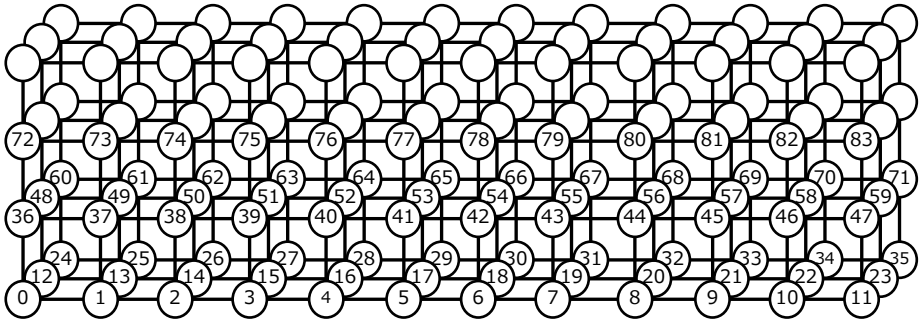
付録：12x3x4の形状指定例 （2/3）

FUJITSU

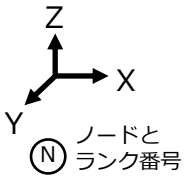
- 12x3x4:torusの形状指定例
  - ジョブの統計情報を確認

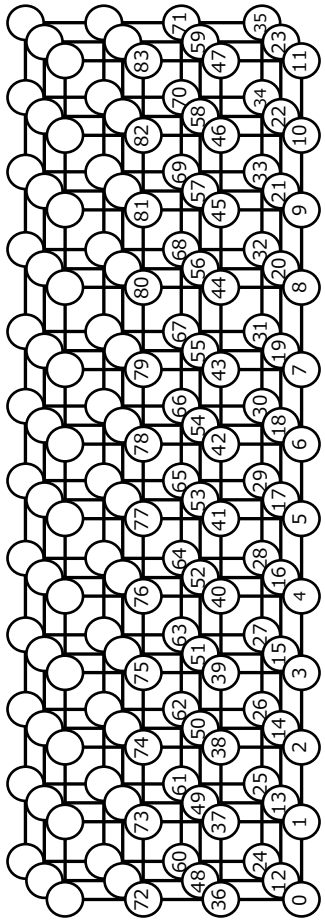
```
Job Statistical Information

 JOB ID                    : 1000889
 SUB JOB NUM               : -
 START BULKNO              : -
 END BULKNO                : -
 HOST NAME                 : flow-fx01
 JOB NAME                  : go12x3x4.sh
 JOB TYPE                  : BATCH
  〈略〉
 NODE NUM (REQUIRE)        : 144:12x3x4
 NODE NUM (ALLOC)          : 144:4x3x12
 NODE NUM (USE)            : 144
 NODE NUM (UNUSED)         : 0

  〈略〉
```

ノード形状は4x3x12で割り当てられた。
これは要求した12x3x4は回転させないと
割り当てられない大きさのため。



形状を回転して
割り当てられた



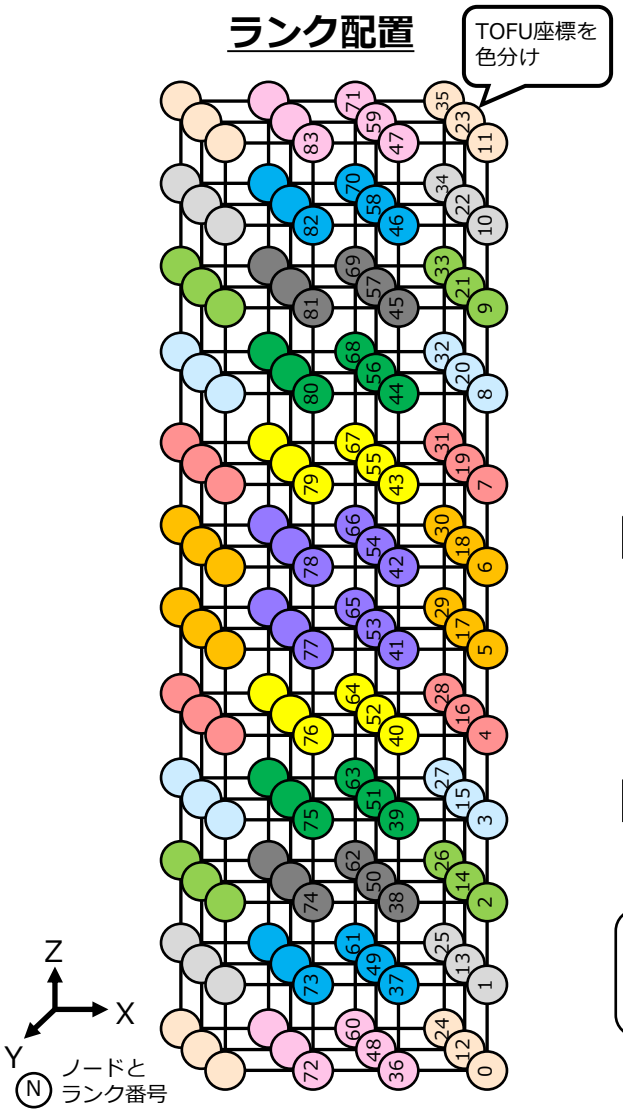


付録：12x3x4の形状指定例 （3/3）

FUJITSU

- 12x3x4:torusの形状指定例

RANK NO	rankmap	NODE COORDINATE	TOFU COORDINATE	6次元座標
0	(0,0,0)	(0,0,0)	(2,1,2)	(2,1,2,0,0,0)
1	(1,0,0)	(1,0,0)	(2,1,3)	(2,1,3,0,0,0)
2	(2,0,0)	(2,0,0)	(2,1,4)	(2,1,4,0,0,0)
3	(3,0,0)	(3,0,0)	(2,1,5)	(2,1,5,0,0,0)
4	(4,0,0)	(4,0,0)	(2,1,6)	(2,1,6,0,0,0)
5	(5,0,0)	(5,0,0)	(2,1,7)	(2,1,7,0,0,0)
6	(6,0,0)	(6,0,0)	(2,1,7)	(2,1,7,0,0,1)
7	(7,0,0)	(7,0,0)	(2,1,6)	(2,1,6,0,0,1)
8	(8,0,0)	(8,0,0)	(2,1,5)	(2,1,5,0,0,1)
9	(9,0,0)	(9,0,0)	(2,1,4)	(2,1,4,0,0,1)
10	(10,0,0)	(10,0,0)	(2,1,3)	(2,1,3,0,0,1)
11	(11,0,0)	(11,0,0)	(2,1,2)	(2,1,2,0,0,1)
12	(0,1,0)	(0,1,0)	(2,1,2)	(2,1,2,0,2,0)
13	(1,1,0)	(1,1,0)	(2,1,3)	(2,1,3,0,2,0)
:	:	:	:	:
36	(0,0,1)	(0,0,1)	(3,1,2)	(3,1,2,0,0,0)
37	(1,0,1)	(1,0,1)	(3,1,3)	(3,1,3,0,0,0)
38	(2,0,1)	(2,0,1)	(3,1,4)	(3,1,4,0,0,0)
39	(3,0,1)	(3,0,1)	(3,1,5)	(3,1,5,0,0,0)
40	(4,0,1)	(4,0,1)	(3,1,6)	(3,1,6,0,0,0)
41	(5,0,1)	(5,0,1)	(3,1,7)	(3,1,7,0,0,0)
42	(6,0,1)	(6,0,1)	(3,1,7)	(3,1,7,0,0,1)
43	(7,0,1)	(7,0,1)	(3,1,6)	(3,1,6,0,0,1)
44	(8,0,1)	(8,0,1)	(3,1,5)	(3,1,5,0,0,1)
45	(9,0,1)	(9,0,1)	(3,1,4)	(3,1,4,0,0,1)
46	(10,0,1)	(10,0,1)	(3,1,3)	(3,1,3,0,0,1)
47	(11,0,1)	(11,0,1)	(3,1,2)	(3,1,2,0,0,1)
48	(0,1,1)	(0,1,1)	(3,1,2)	(3,1,2,0,2,0)
:	:	:	:	:
138	(6,2,3)	(6,2,3)	(2,1,7)	(2,1,7,1,1,1)
139	(7,2,3)	(7,2,3)	(2,1,6)	(2,1,6,1,1,1)
140	(8,2,3)	(8,2,3)	(2,1,5)	(2,1,5,1,1,1)
141	(9,2,3)	(9,2,3)	(2,1,4)	(2,1,4,1,1,1)
142	(10,2,3)	(10,2,3)	(2,1,3)	(2,1,3,1,1,1)
143	(11,2,3)	(11,2,3)	(2,1,2)	(2,1,2,1,1,1)



付録：12x12の形状指定例 （1/2）

- 12x12:torusの形状指定例
  - 実行条件
    - 形状は12x12:torus
    - プロセス数は144（1ノード1プロセス）
    - ランクマップファイルを使用

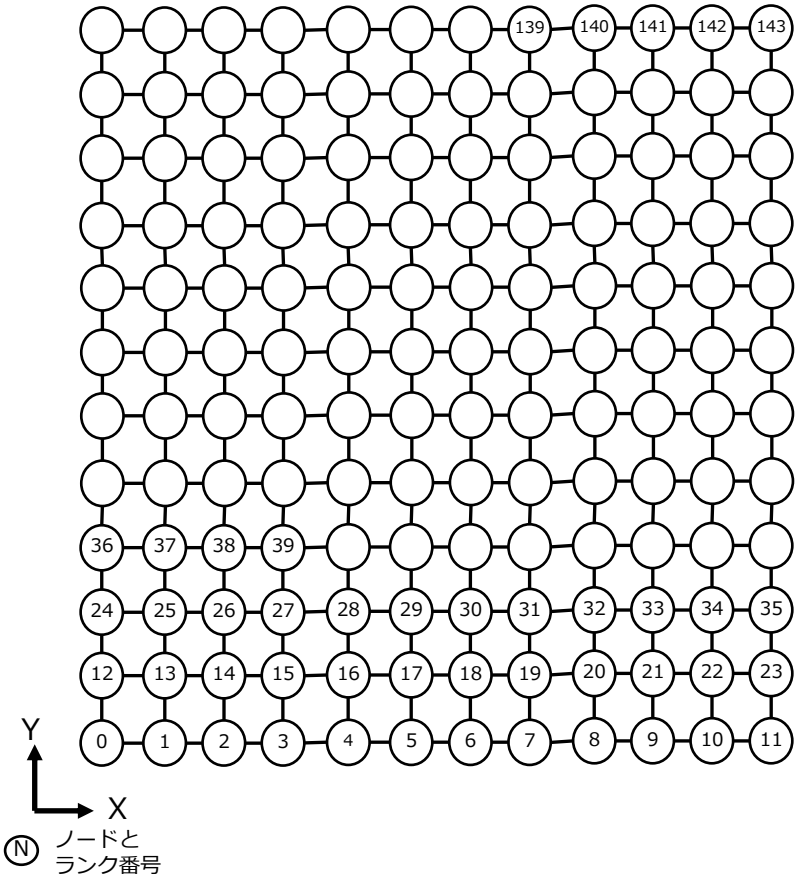
実行スクリプト

```
#!/bin/bash
#PJM -L rscunit=fx
#PJM -L rscgrp=fx-large
#PJM -L elapse=00:01:00
#PJM -L node=12x12:torus
#PJM --mpi proc=144
#PJM --mpi rank-map-hostfile=rankmap_12x12
#PJM -s

mpiexec ./a.out
```

ランクマップファイル
(rankmap_12x12)

```
(  0,   0) ← ランク0
(  1,   0) ← ランク1
(  2,   0) ← ランク2
(  3,   0) ← ランク3
(  4,   0) ← ランク4
(  5,   0)
(  6,   0)
(  7,   0)
(  8,   0)
(  9,   0)
( 10,   0)
( 11,   0)
(  0,   1)
(  1,   1)
(  2,   1)
      :
      :
(  8,  11) ← ランク140
(  9,  11) ← ランク141
( 10,  11) ← ランク142
( 11,  11) ← ランク143
```

ランクマップファイルによるプロセス配置

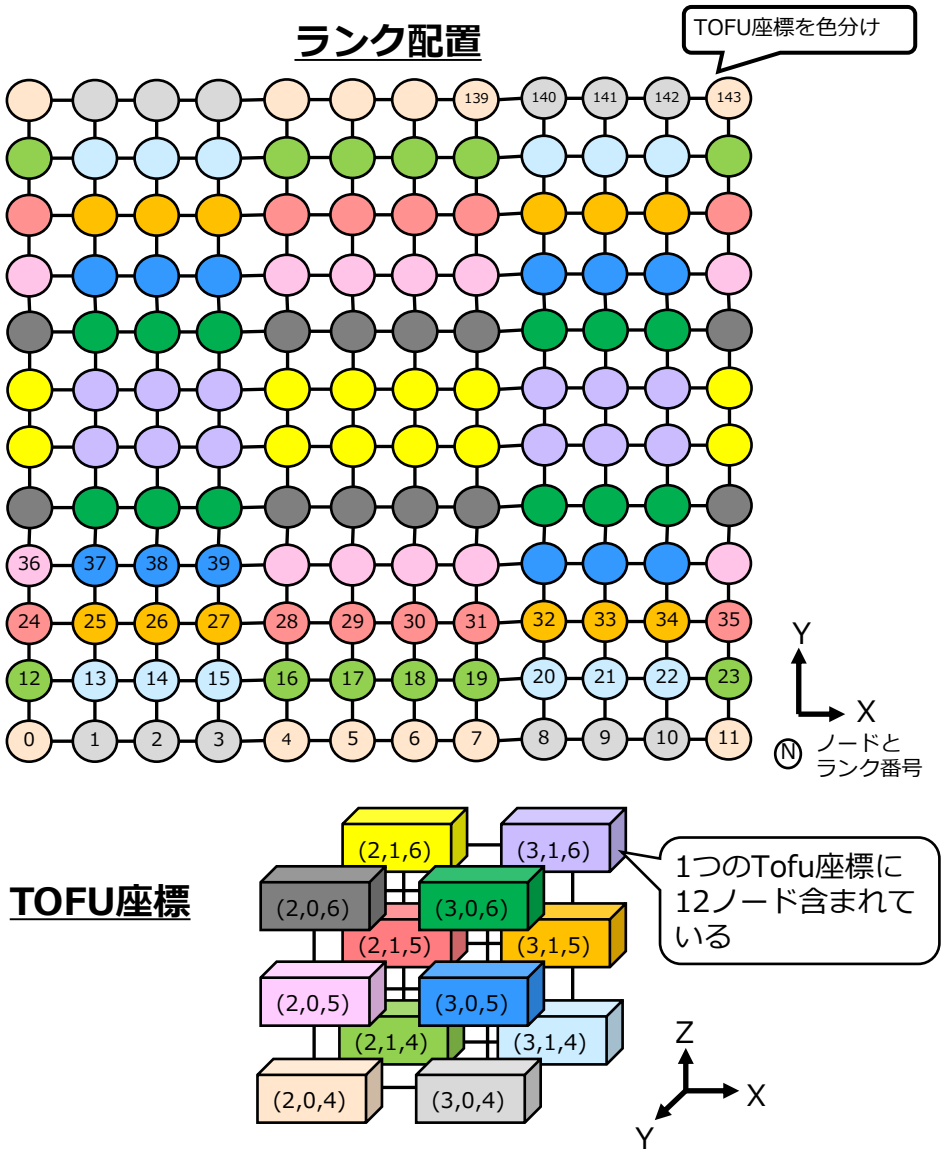


付録：12x12の形状指定例 （2/2）

FUJITSU

- 12x12:torusの形状指定例

RANK NO	rankmap	NODE COORDINATE	TOFU COORDINATE	6次元座標
0	(0,0)	(0,0,0)	(2,0,4)	(2,0,4,0,0,0)
1	(1,0)	(1,0,0)	(3,0,4)	(3,0,4,0,0,0)
2	(2,0)	(2,0,0)	(3,0,4)	(3,0,4,0,1,0)
3	(3,0)	(3,0,0)	(3,0,4)	(3,0,4,0,2,0)
4	(4,0)	(4,0,0)	(2,0,4)	(2,0,4,0,2,0)
5	(5,0)	(5,0,0)	(2,0,4)	(2,0,4,0,1,0)
6	(6,0)	(6,0,0)	(2,0,4)	(2,0,4,1,1,0)
7	(7,0)	(7,0,0)	(2,0,4)	(2,0,4,1,2,0)
8	(8,0)	(8,0,0)	(3,0,4)	(3,0,4,1,2,0)
9	(9,0)	(9,0,0)	(3,0,4)	(3,0,4,1,1,0)
10	(10,0)	(10,0,0)	(3,0,4)	(3,0,4,1,0,0)
11	(11,0)	(11,0,0)	(2,0,4)	(2,0,4,1,0,0)
12	(0,1)	(0,1,0)	(2,1,4)	(2,1,4,0,0,0)
13	(1,1)	(1,1,0)	(3,1,4)	(3,1,4,0,0,0)
:	:	:		
36	(0,3)	(0,3,0)	(2,0,5)	(2,0,5,0,0,0)
37	(1,3)	(1,3,0)	(3,0,5)	(3,0,5,0,0,0)
38	(2,3)	(2,3,0)	(3,0,5)	(3,0,5,0,1,0)
39	(3,3)	(3,3,0)	(3,0,5)	(3,0,5,0,2,0)
40	(4,3)	(4,3,0)	(2,0,5)	(2,0,5,0,2,0)
41	(5,3)	(5,3,0)	(2,0,5)	(2,0,5,0,1,0)
42	(6,3)	(6,3,0)	(2,0,5)	(2,0,5,1,1,0)
43	(7,3)	(7,3,0)	(2,0,5)	(2,0,5,1,2,0)
44	(8,3)	(8,3,0)	(3,0,5)	(3,0,5,1,2,0)
45	(9,3)	(9,3,0)	(3,0,5)	(3,0,5,1,1,0)
46	(10,3)	(10,3,0)	(3,0,5)	(3,0,5,1,0,0)
47	(11,3)	(11,3,0)	(2,0,5)	(2,0,5,1,0,0)
48	(0,4)	(0,4,0)	(2,0,6)	(2,0,6,0,0,0)
:	:	:		
138	(6,11)	(6,11,0)	(2,0,4)	(2,0,4,1,1,1)
139	(7,11)	(7,11,0)	(2,0,4)	(2,0,4,1,2,1)
140	(8,11)	(8,11,0)	(3,0,4)	(3,0,4,1,2,1)
141	(9,11)	(9,11,0)	(3,0,4)	(3,0,4,1,1,1)
142	(10,11)	(10,11,0)	(3,0,4)	(3,0,4,1,0,1)
143	(11,11)	(11,11,0)	(2,0,4)	(2,0,4,1,0,1)



付録：6x6x4の形状指定例 （1/2）

FUJITSU

- 6x6x4:torusの形状指定例
  - 実行条件
    - 形状は6x6x4:torus
    - プロセス数は144（1ノード1プロセス）
    - ランクマップファイルを使用

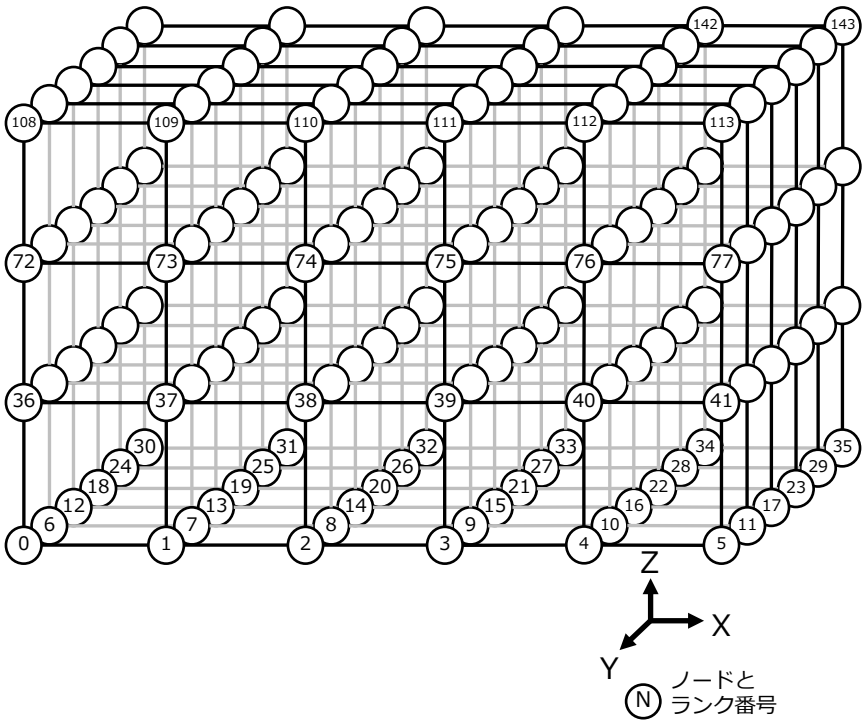
実行スクリプト

```
#!/bin/bash
#PJM -L rscunit=fx
#PJM -L rscgrp=fx-large
#PJM -L elapse=00:01:00
#PJM -L node=6x6x4:torus:strict
#PJM --mpi proc=144
#PJM --mpi rank-map-hostfile=rankmap_6x6x4
#PJM -s

mpiexec ./a.out
```

ランクマップファイル
(rankmap_6x6x4)

```
( 0, 0, 0) ← ランク0
( 1, 0, 0) ← ランク1
( 2, 0, 0) ← ランク2
( 3, 0, 0) ← ランク3
( 4, 0, 0) ← ランク4
( 5, 0, 0)
( 0, 1, 0)
( 1, 1, 0)
( 2, 1, 0)
( 3, 1, 0)
( 4, 1, 0)
( 5, 1, 0)
( 0, 2, 0)
( 1, 2, 0)
( 2, 2, 0)
      :
      :
( 2, 5, 3) ← ランク140
( 3, 5, 3) ← ランク141
( 4, 5, 3) ← ランク142
( 5, 5, 3) ← ランク143
```

ランクマップファイルによるプロセス配置



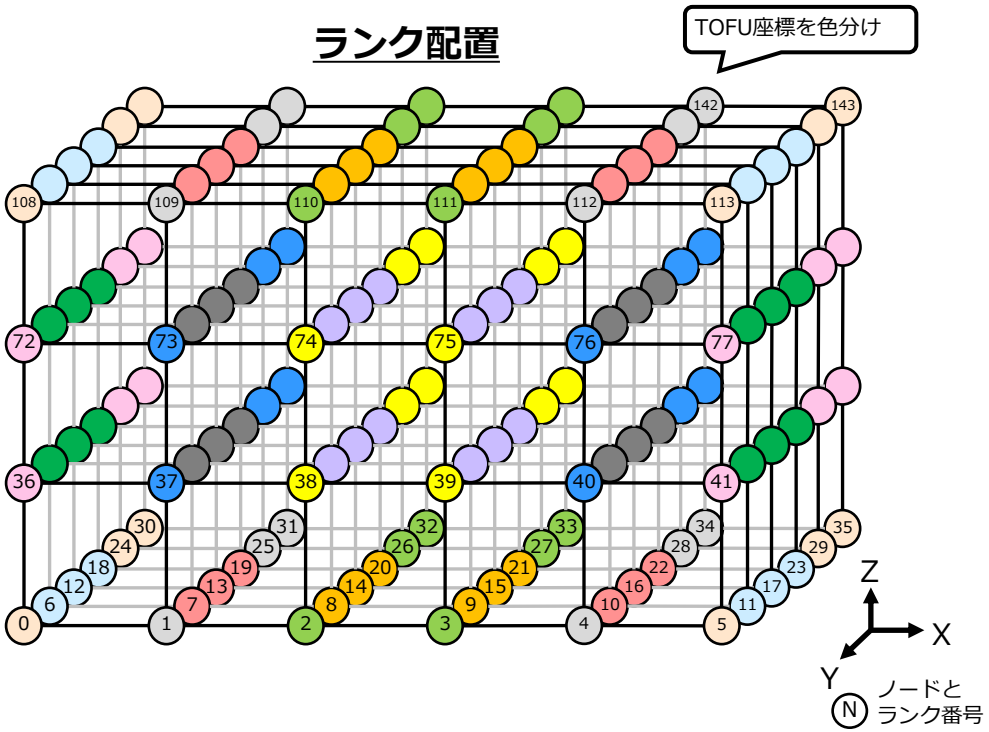
付録：6x6x4の形状指定例 （2/2）

FUJITSU

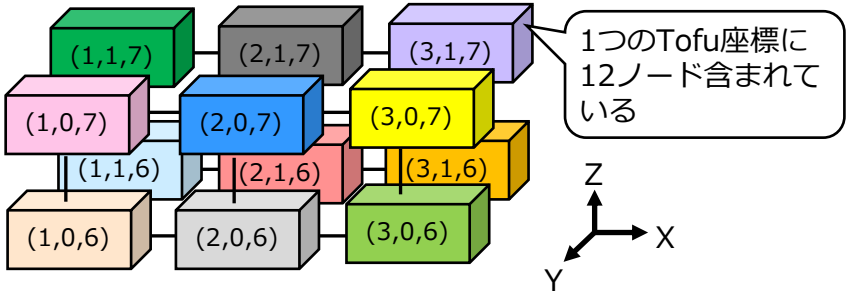
- 6x6x4:torusの形状指定例

RANK NO	rankmap	NODE COORDINATE	TOFU COORDINATE	6次元座標
0	(0,0,0)	(0,0,0)	(1,0,6)	(1,0,6,0,0,0)
1	(1,0,0)	(1,0,0)	(2,0,6)	(2,0,6,0,0,0)
2	(2,0,0)	(2,0,0)	(3,0,6)	(3,0,6,0,0,0)
3	(3,0,0)	(3,0,0)	(3,0,6)	(3,0,6,1,0,0)
4	(4,0,0)	(4,0,0)	(2,0,6)	(2,0,6,1,0,0)
5	(5,0,0)	(5,0,0)	(1,0,6)	(1,0,6,1,0,0)
6	(0,1,0)	(0,1,0)	(1,1,6)	(1,1,6,0,0,0)
7	(1,1,0)	(1,1,0)	(2,1,6)	(2,1,6,0,0,0)
8	(2,1,0)	(2,1,0)	(3,1,6)	(3,1,6,0,0,0)
9	(3,1,0)	(3,1,0)	(3,1,6)	(3,1,6,1,0,0)
10	(4,1,0)	(4,1,0)	(2,1,6)	(2,1,6,1,0,0)
11	(5,1,0)	(5,1,0)	(1,1,6)	(1,1,6,1,0,0)
12	(0,2,0)	(0,2,0)	(1,1,6)	(1,1,6,0,1,0)
13	(1,2,0)	(1,2,0)	(2,1,6)	(2,1,6,0,1,0)
:	:	:		
36	(0,0,1)	(0,0,1)	(1,0,7)	(1,0,7,0,0,0)
37	(1,0,1)	(1,0,1)	(2,0,7)	(2,0,7,0,0,0)
38	(2,0,1)	(2,0,1)	(3,0,7)	(3,0,7,0,0,0)
39	(3,0,1)	(3,0,1)	(3,0,7)	(3,0,7,1,0,0)
40	(4,0,1)	(4,0,1)	(2,0,7)	(2,0,7,1,0,0)
41	(5,0,1)	(5,0,1)	(1,0,7)	(1,0,7,1,0,0)
42	(0,1,1)	(0,1,1)	(1,1,7)	(1,1,7,0,0,0)
43	(1,1,1)	(1,1,1)	(2,1,7)	(2,1,7,0,0,0)
44	(2,1,1)	(2,1,1)	(3,1,7)	(3,1,7,0,0,0)
45	(3,1,1)	(3,1,1)	(3,1,7)	(3,1,7,1,0,0)
46	(4,1,1)	(4,1,1)	(2,1,7)	(2,1,7,1,0,0)
47	(5,1,1)	(5,1,1)	(1,1,7)	(1,1,7,1,0,0)
48	(0,2,1)	(0,2,1)	(1,1,7)	(1,1,7,0,1,0)
:	:	:		
138	(0,5,3)	(0,5,3)	(1,0,6)	(1,0,6,0,1,1)
139	(1,5,3)	(1,5,3)	(2,0,6)	(2,0,6,0,1,1)
140	(2,5,3)	(2,5,3)	(3,0,6)	(3,0,6,0,1,1)
141	(3,5,3)	(3,5,3)	(3,0,6)	(3,0,6,1,1,1)
142	(4,5,3)	(4,5,3)	(2,0,6)	(2,0,6,1,1,1)
143	(5,5,3)	(5,5,3)	(1,0,6)	(1,0,6,1,1,1)

ランク配置



TOFU座標



Thank you