筑波大学計算科学研究センター
CCS HPCサマーセミナー
「T2Kの利用法」

建部修見
tatebe@cs.tsukuba.ac.jp
筑波大学大学院システム情報工学研究科
計算科学研究センター

T2K筑波システム

- 筑波大学が平成20年6月より運用を開始した、最新スーパーコンピュータシステム
- 筑波大学計算科学研究センターが管理・運用
- 95TFLOPSのピーク性能、800TBのユーザファイルシステム
- 東京大学情報基盤センター、京都大学学術情報メディアセンターとの連携：”T2K Open Supercomputer Alliance”の下、基本仕様の共同策定、運用開始後の運用、教育等の様々な点で連携を行う
  - 「T2K東大」：140TFLOPS
  - 「T2K京大」：61TFLOPS+SMPサーバ

T2K筑波システム全体

設置場所：筑波大学計算科学研究センター・スパコン別棟

クラスタ部とファイルサーバ部

計算ノード（70 racks）

648 node (quad-core x 4 socket / node)
Opteron “Barcelona” B8000 CPU
2.3 GHz x 4命令実行 x 4 core x 4 socket
= 147.2 GFLOPS / node
= 95.3 TFLOPS / system
20.7 TB memory / system

800 TB (physical 1PB) RAID-6
Luster cluster file system
Infiniband x 2 接続
Meta-Data Server, File Serverの全てが2重系⇒高度なfault tolerance

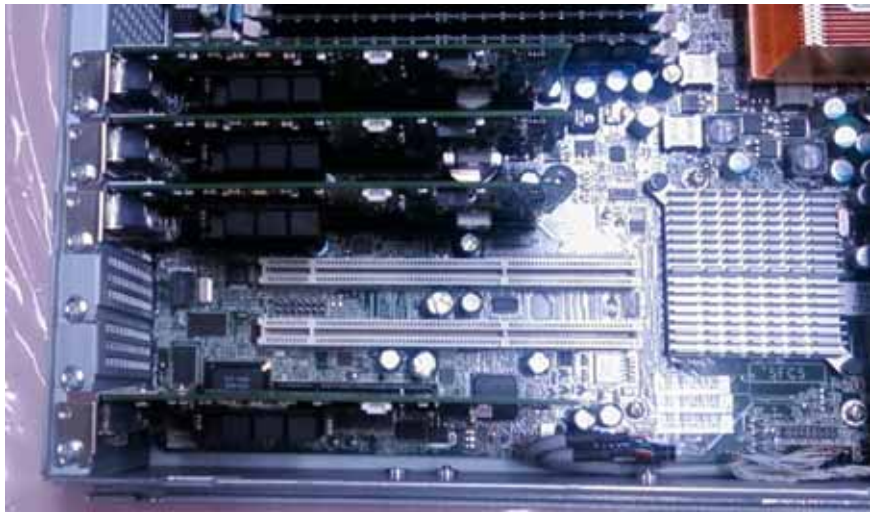
ファイルサーバ（ディスク部）

計算ノードの構成

- Multi-core & multi-socket構成による超高性能演算
  - AMD quad-core Opteron 8000シリーズ (Barcelona) の採用
  - 4 socket / node 構成⇒ ピーク演算性能 147 GFLOPS/node
  - OpteronのNUMAアーキテクチャにより、各プロセッサのメモリバンド幅を効率的に利用⇒ メモリバンド幅 40 GB/s/node
- Multi-rail ネットワークによる超高バンド幅ネットワークリンク
  - 4xDDR Infiniband (Mellanox ConnectX) x 4 / node
  - Quad-rail Infiniband をトランキングすることにより高バンド幅ネットワークリンクを実現⇒ ピーク通信性能 8 GB/s/node
- 柔軟なプロセス構成による最適化された並列処理
  - MVAPICH (modified by Appro) の multi-rail configuration により、MPIプロセス当たりの利用rail数を制御可能
    ⇒ノード上の並列プロセス数に応じてmulti-railを使い分け可能

計算ノードの内部

ノードシャーシ内部



Infiniband ConnectX x 4
(各PCI-Express x 8 lane)

T2K筑波システムのプログラミング

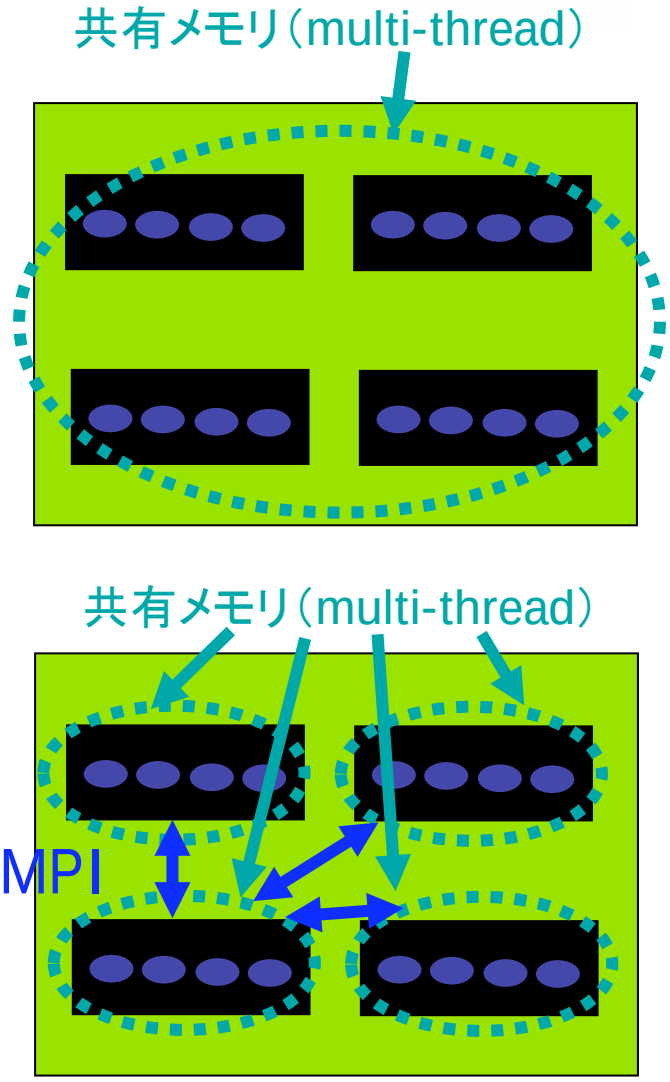
- ソフトウェア環境
  - OS: Red Hat Enterprise Linux v.5 WS (Linux kernel 2.6)
  - 使用可能言語：F90, C, C++, Java
  - コンパイラ：PGI (Portrand Group), Intel
  - MPIライブラリ：MVAPICH（Approによる修正版）
  - 数値計算ライブラリ：IMSL（一部ノード群）, ACML, SCALAPACK
  - チューニング環境：PGPROFR, PAPI
- プログラミング
  - 逐次プログラミング：コンパイラによる最適化
  - 並列プログラミング
    - コンパイラによる共有メモリ並列化：ノード内
    - ユーザによるOpenMP並列化：ノード内
    - ユーザによるMPI並列化：ノード間、ノード内

様々な並列化

- ノード内自動並列化
  - 最大16スレッド（16コア）による共有メモリ並列化
  - コンパイラによる解析に依存するため、単純ループ構造が対象
- ノード内OpenMP並列化
  - 最大16スレッド（16コア）による共有メモリ並列化
  - 標準的なOpenMP directiveによる並列化指示
- ノード間MPI並列化
  - 標準的なMPI (Message Passing Interface)による明示的な並列プログラミング
- ハイブリッド並列化
  - ノード内の共有メモリ並列化（コンパイラ自動 or OpenMP）とMPI並列化を組み合わせる

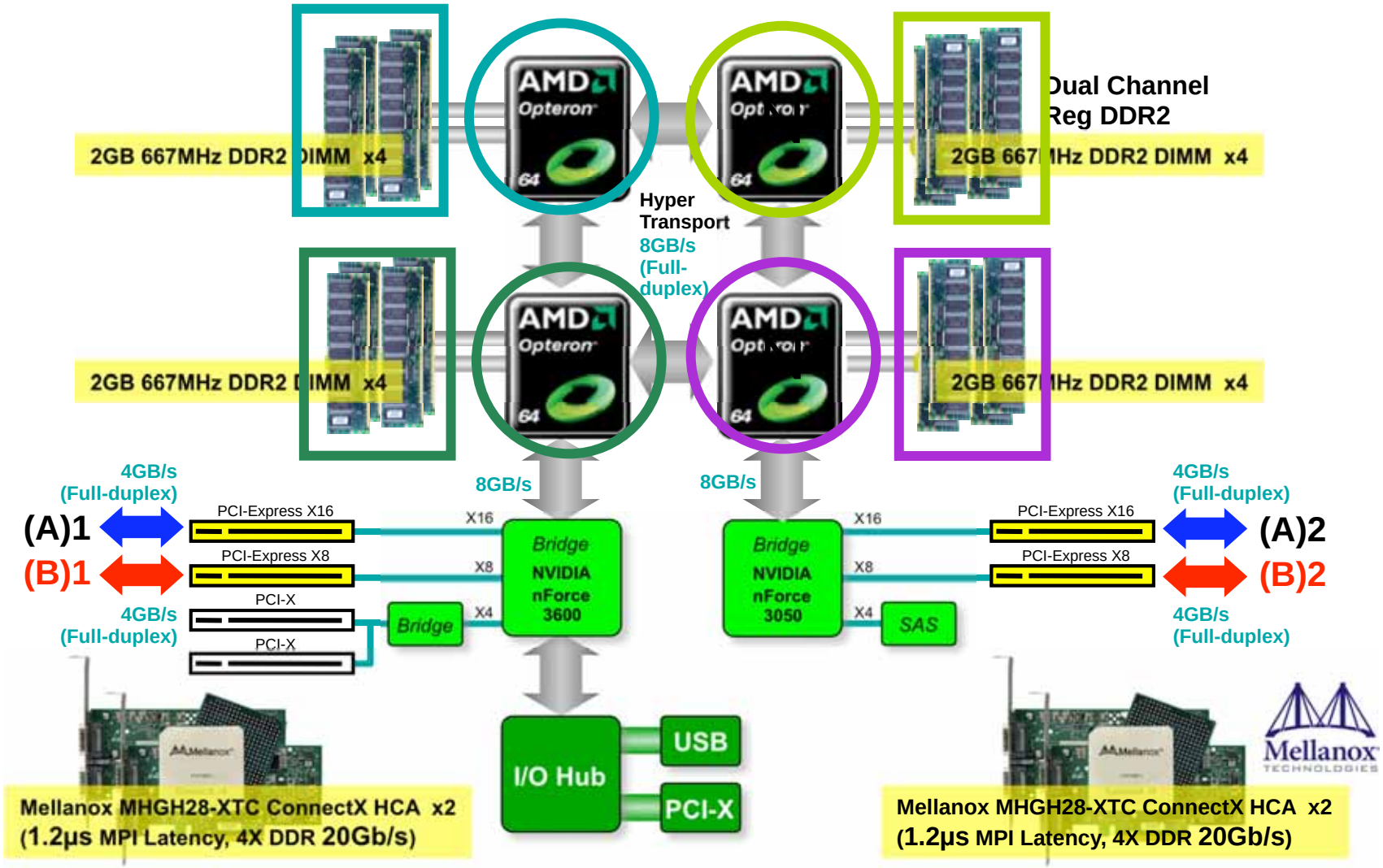
高度なハイブリッド並列化

- 共有メモリ並列(スレッド並列:自動またはOpenMP)の対象はノード内全コアとは限らない
  - 各CPUコアから見た他のコアとのメモリ共有レベル
    - L1及びL2キャッシュはコア独立
    - L3キャッシュは(同一ソケット内)コア間共有
    - オフチップメモリはソケット間・コア間共有
    - ⇒全16コアによる共有メモリプログラミングは性能を最大限に生かせない可能性がある
  - ソケット(4コア)内で共有メモリ並列化を行い、ノード内といえどもソケット間はMPIによるメッセージ通信という使い方も可能
    - プログラムのループ構造やデータサイズ(ワーキングセット)により最適化が可能



メモリマップとプロセスマップ

プロセス（コア）と参照データを近接メモリにマッピング可能（*numactl* 機能）

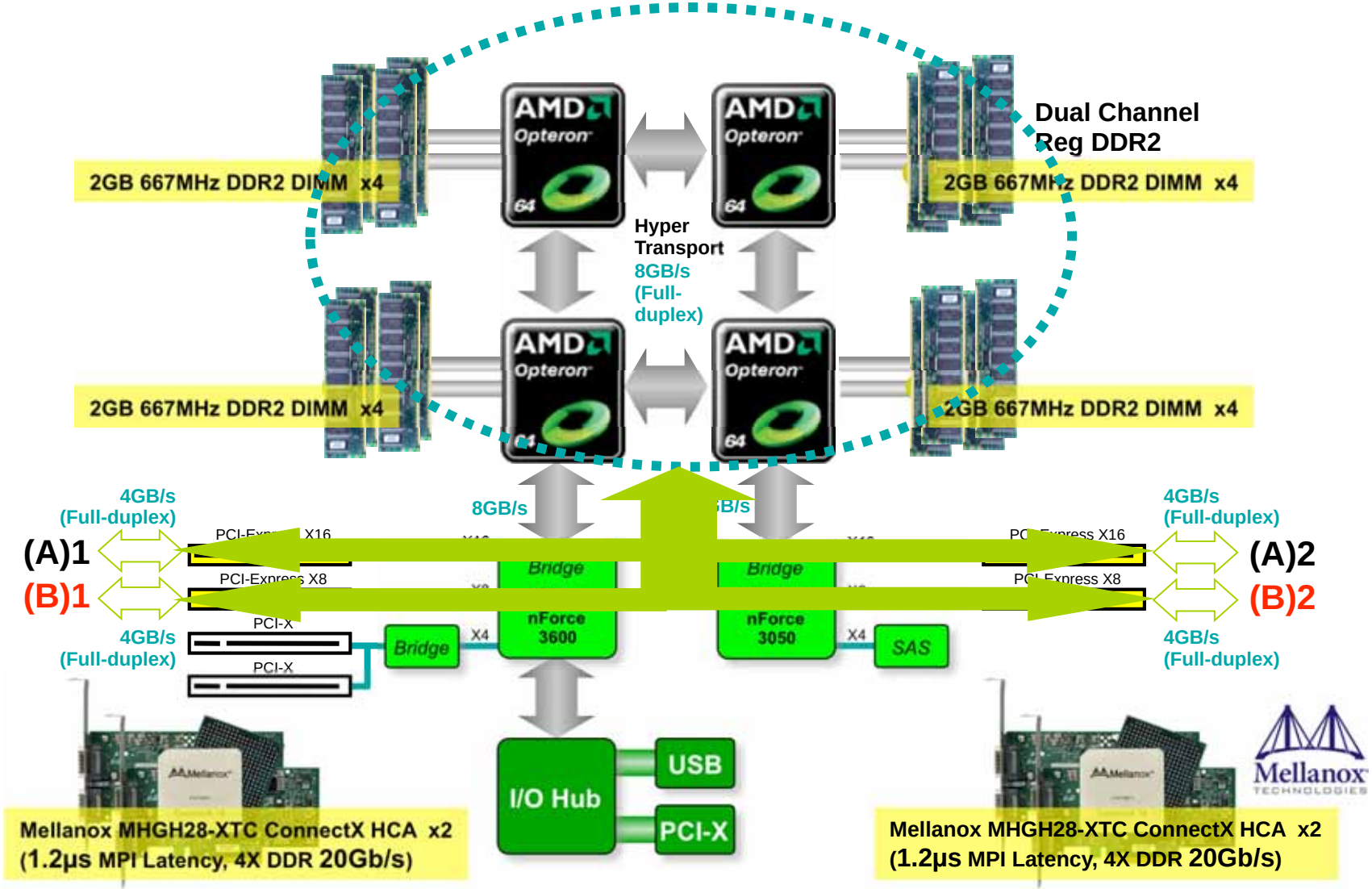


MPI通信におけるマルチリンクの利用

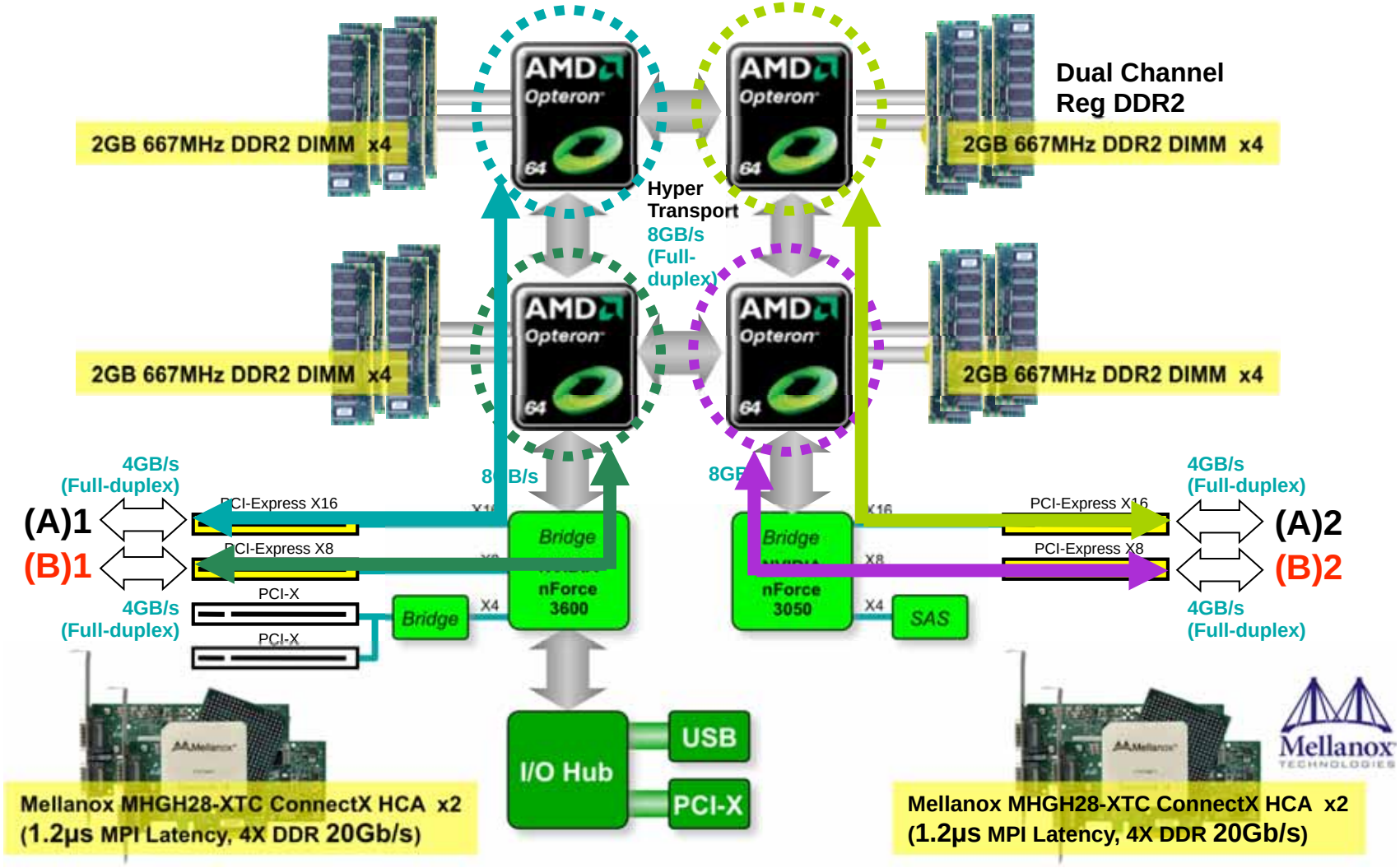
- Multi-rail の利用方法⇒様々な構成が可能
  例:
  - 4 rail / 1 MPI process (x 16 thread)
  - 4 rail / 4 MPI process (x 4 thread)
  - 1 rail / 1 MPI process (x 4 thread)
  - 4 rail / 16 MPI process (x 1 thread)

- Multi-rail Infinibandによるバックアップ運用
  - 平常時は全リンクを均等利用
  - リンクに故障が生じた場合、そのリンクをシステム構成から外して運用することが可能（自動fail-overはまだ未対応）
  - リンク間のロードバランスはInfinibandのSubnet Management機能によって提供

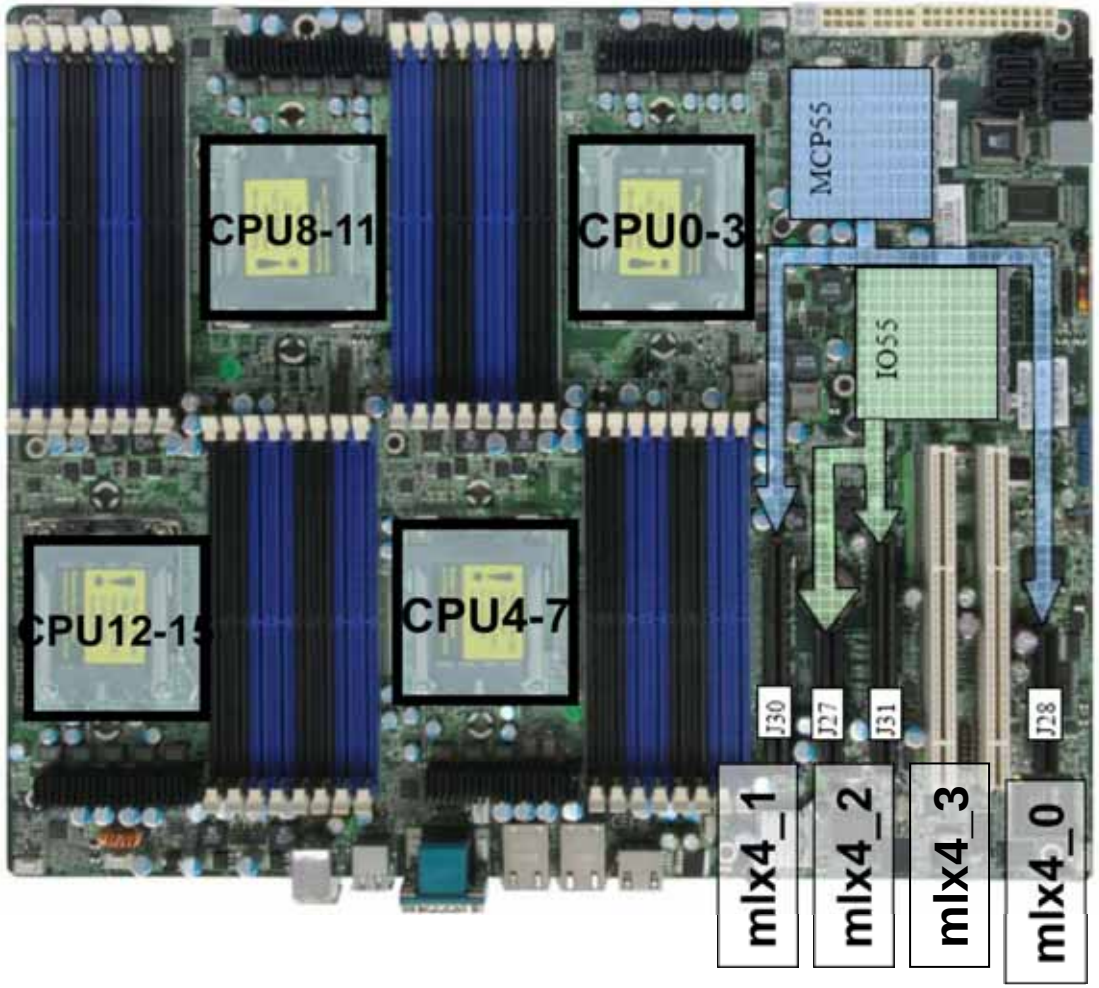
4 link / 1 MPI process (x 16 thread) の例



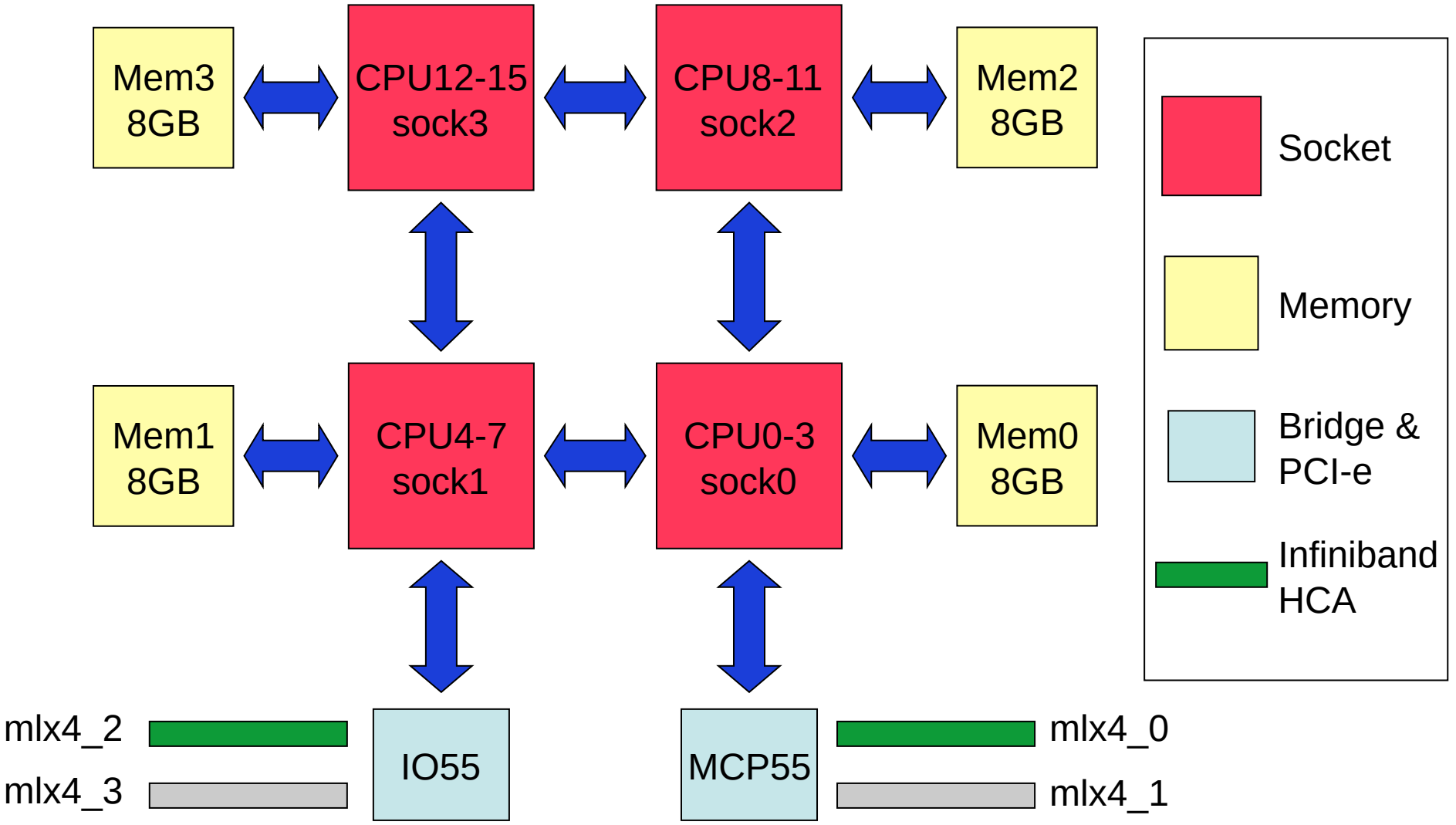
1 link / 1 MPI process (x4 thread) の例



T2K筑波ノード内部


CPU8-11
CPU0-3
MCP55
IO55
CPU12-15
CPU4-7
J30
J27
J31
J28
mlx4_1
mlx4_2
mlx4_3
mlx4_0

T2K筑波ノードの構成



CPUとメモリの配置（1）

- numactl – プロセス，メモリ配置の制御

```
$ numactl --show
policy: default
preferred node: current
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
cpubind: 0 1 2 3
nodebind: 0 1 2 3
membind: 0 1 2 3
```

```
$ numactl --hardware
available: 4 nodes (0-3)
node 0 size: 8062 MB
node 0 free: 112 MB
node 1 size: 8080 MB
node 1 free: 327 MB
node 2 size: 8080 MB
node 2 free: 274 MB
node 3 size: 8080 MB
node 3 free: 354 MB
node distances:
node   0   1   2   3
  0:  10  20  20  20
  1:  20  10  20  20
  2:  20  20  10  20
  3:  20  20  20  10
```

CPUとメモリの配置（2）

```
usage: numactl [--interleave=nodes] [--preferred=node]
               [--physcpubind=cpus] [--cpunodebind=nodes]
               [--membind=nodes] [--localalloc] command args ...
```

- --interleave
  - all, 0-2など複数のメモリブロックを指定
  - 複数のメモリブロックをインターリーブで利用
  - 確保できない場合は空いているメモリブロックを利用
- --preferred
  - なるべく利用したいメモリブロックを指定
  - 確保できない場合は空いているメモリブロックを利用
- --physcpubind
  - 実行したいCPUコア番号を指定
- --cpunodebind
  - 実行したいソケット番号を指定
- --membind
  - 利用するメモリブロックを指定
  - 確保できない場合はフェイル
- --localalloc
  - 自メモリブロックを利用

MPIプログラムのコンパイル

- MPI環境のセットアップ
  - % module load pgi mvapich2/pgi　　# PGI
  - % module load intel mvapich2/intel　# Intel
  - % module load mvapich2/gnu　　# GCC
- コンパイラドライバ
  - mpicc, mpiCC, mpif77, mpif90

16MPIプロセス／ノード

16MPIプロセス／ノード

mpirun_rsh –np 16 numactl –localalloc ./a.out

numactlにより近接メモリブロックを利用

4MPIプロセス／ノード

```
SOCKET=$(( $MPIRUN_RANK % 4 ))
/opt/sge/mpi/make_hostfile.sh 4 > ${JOB_ID}_host
mpirun_rsh -np 4 -hostfile ${JOB_ID}_host
    numactl –cpunodebind=$SOCKET –localalloc ./a.out
rm ${JOB_ID}_host
```

numactlによりソケット単位でのプロセス配置と近接メモリブロック利用の指定

マルチレールの利用

- T2K筑波では各ノードはIB4本で接続
- MVAPICH2で利用可能
- プロセス当たり利用するレールの数は環境変数MV2_NUM_HCASで指定
  - mpirun_rsh … MV2_NUM_HCAS=4 ./a.out
    ＃1プロセスで4レール利用する場合
- 詳細はT2K-Tsukubaシステム利用ガイドを参照