

アクセラレータ間直接通信機構TCAと XcalableMP

朴 泰祐

筑波大学・計算科学研究センター
(システム情報系)

研究の背景

- ポストペタスケール～エクサスケールシステムの有力な候補として演算加速装置 (Accelerating Devices: AD) を多数用いた超並列システムを想定
 - AD: GPU, GRAPE-DR, MIC, FPGA, etc.
- 超並列AD環境におけるHPCの問題点
 - AD間の結合: AD性能の向上に伴う通信ボトルネックの顕在化
 - 記憶装置: 限られたメモリサイズ
 - ADとホストCPUの通信: バンド幅、協調計算モデル⇒複雑化
 - プログラミング: 直交する多くのパラダイム⇒生産性の低下
- エクサ時代の Strong Scaling 問題
 - コア当たりメモリ容量の縮小により weak scaling に依存した性能向上が限界に
 - strong scaling では通信レイテンシが本質的問題となる

研究の目的

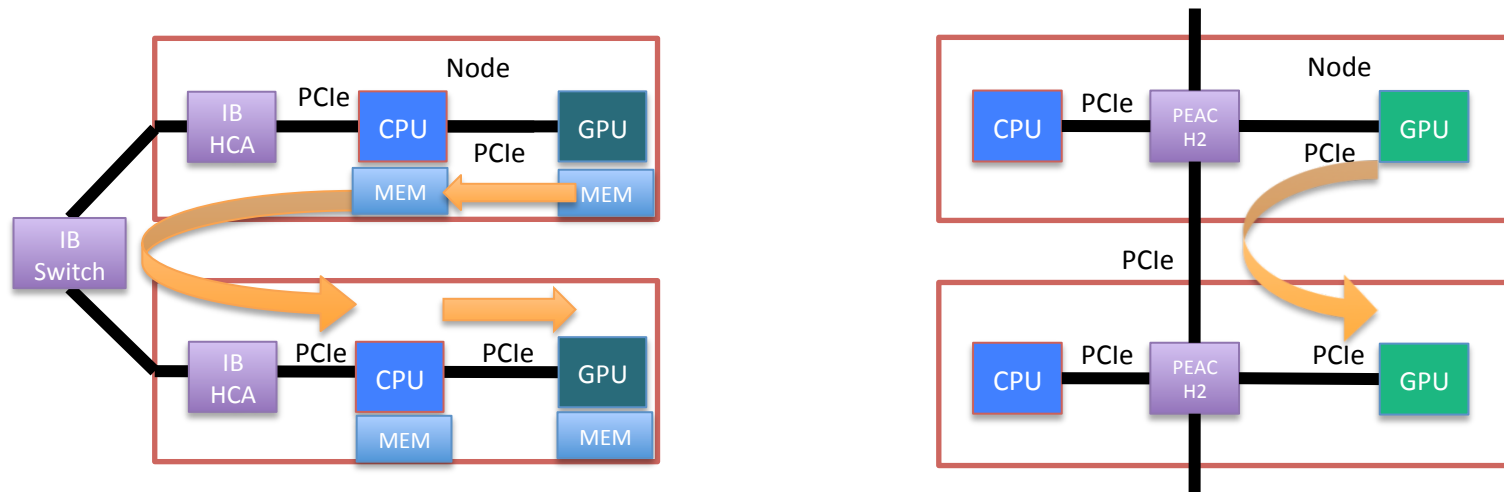
- Strong scalingを意識した低レイテンシAD間通信システムを構築し、AD演算性能を最大限に活かすシステムを構築
 - 並列化されたAD間の超高速(低レイテンシ)通信の実現
 - AD間直接通信をサポートするシステムソフトウェア(ドライバ、通信レイヤ)の開発
 - AD間直接通信をサポートする並列言語・コンパイラの開発
 - AD間直接通信環境を想定したアプリケーション開発
- 大規模並列GPUコンピューティング(代表的AD)の高生産プログラミング環境
 - MPI, OpenMP, CUDA (or OpenCL, OpenACC) の複合コード
 - アプリケーション開発の生産性が非常に低い
 - 今後のAD環境に対するアプリケーションの継続性が保てない
 - ⇒ 高レベルで洗練された開発環境が必要
- TCA(後述)のコンセプトをより高度なソフトウェア技術として確立し、システムソフトウェアからアプリケーションまでの開発を行う
 - ⇒ 実証実験、システムソフト開発、アルゴリズム開発

CREST研究体制

- 「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」領域、研究課題「ポストペタスケール時代に向けた演算加速機構・通信機構統合環境の研究開発」
- 研究チーム
 - TCAシステムソフトウェア開発グループ(朴:筑波大、代表)
 - FPGA開発グループ(天野:慶大)
 - 言語開発グループ(村井:理研)
 - アプリケーション開発グループ(梅村:筑波大)
- ドライバ、低レベル～高レベル通信ライブラリ、コンパイラ、アプリケーションを一体化して開発することが重要

基本コンセプト:TCA

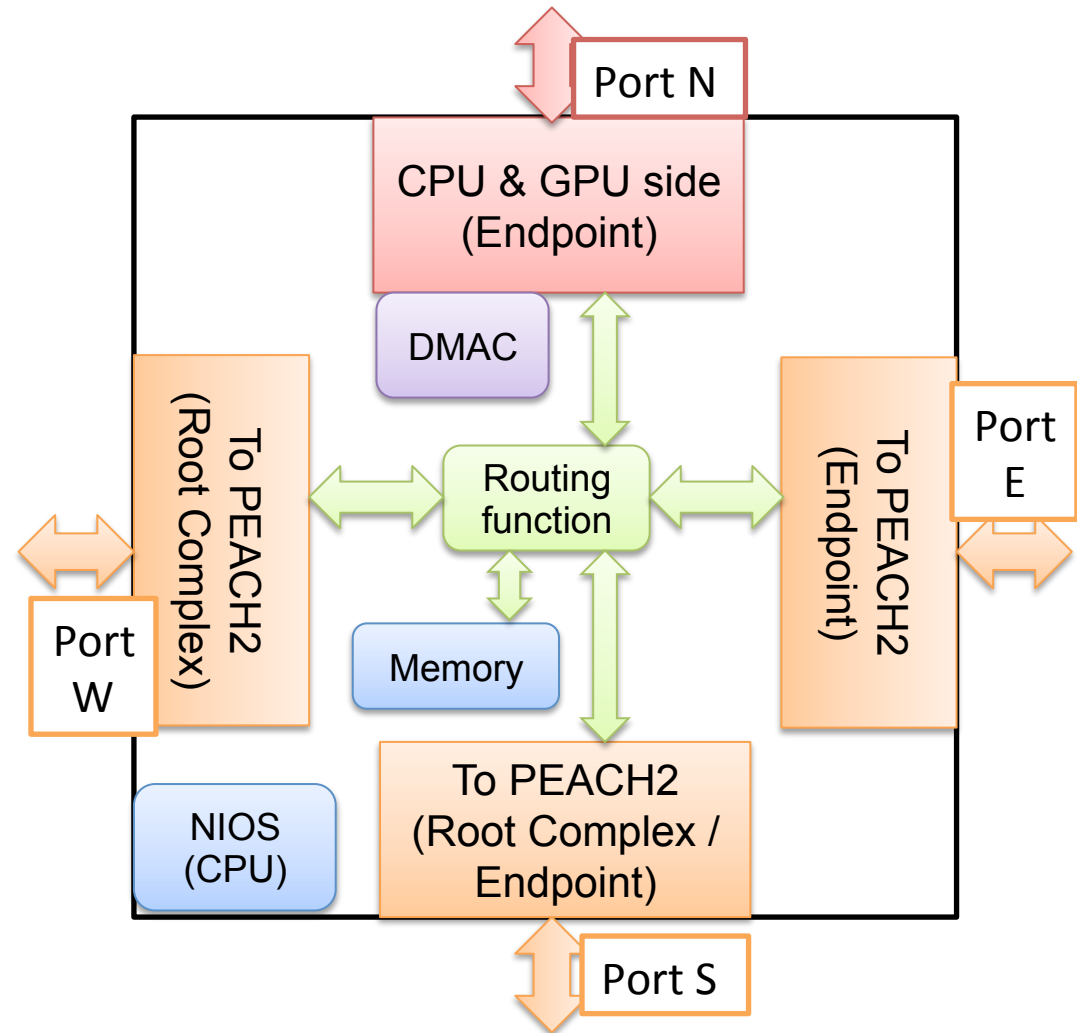
- TCA (Tightly Coupled Accelerators)
 - PCIe によるAD間直接通信(ノード内・ノード間)を実現
 - PEACH2チップ(FPGAによるプロトタイプ)によるインテリジェントなPCIeスイッチ+コントローラ
 - ホストCPU・メモリ・結合網に依存しないAD間直接通信



- TCAライクな通信システムはベンダーや多くの研究者が注目している
 - NVIDIA+Mellanox GPU direct
 - MVAPICH2 (OSU) の CUDA5.0 対応MPI
 - APE+ network (イタリア)

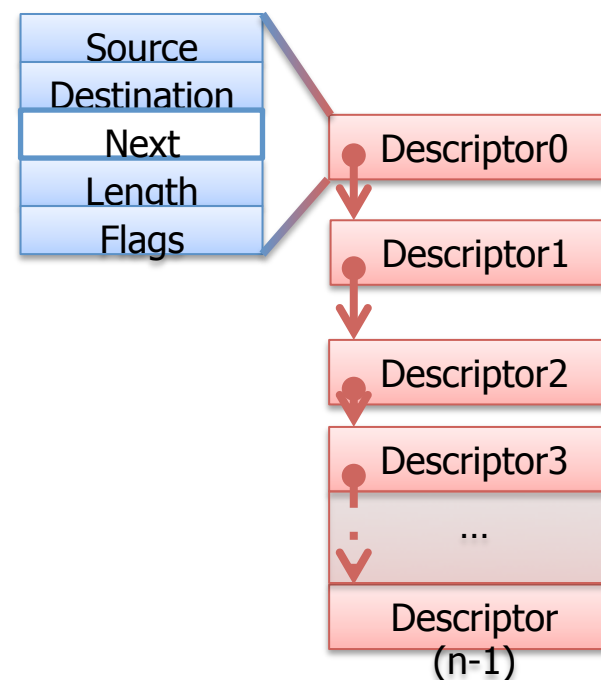
PEACH2チップ

- **PEACH2**: PCI Express Adaptive Communication Hub ver.2
- PCIe Gen2 準拠
- FPGA (Altair Stratix-IV) による実装
- Root Complex と EndPoint がペアにならないといけない
- **Port N**: ホストと他のGPU用
- **Port E and W**: リング結合用
- **Port S**: 別次元リング用
 - RCとEPを切り替え可能
- Port N以外は書き込み専用
 - リモート読み出しは“Proxy write”で実行



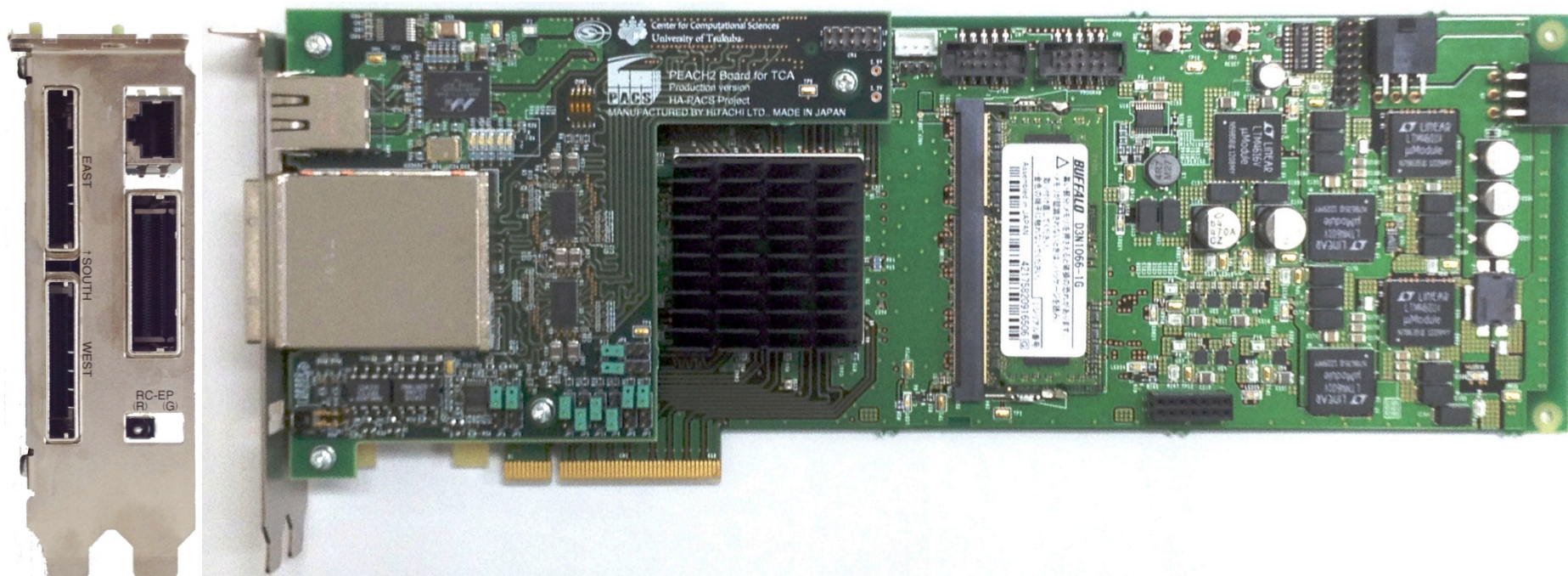
PEACH2による通信

- PIO
 - CPUはmmap関数を使ってリモートノードにデータ書き込みを実行可能
- DMA
 - Chaining mode
 - ホストまたはPEACH2上の内蔵メモリにDMA descriptorを用意し、これを数珠つなぎにしておく
 - Register mode
 - PEACH2内に16個のDMA request register
 - DMA chainingより立ち上げが速い
 - Block-stride転送が可能



PEACH2 board

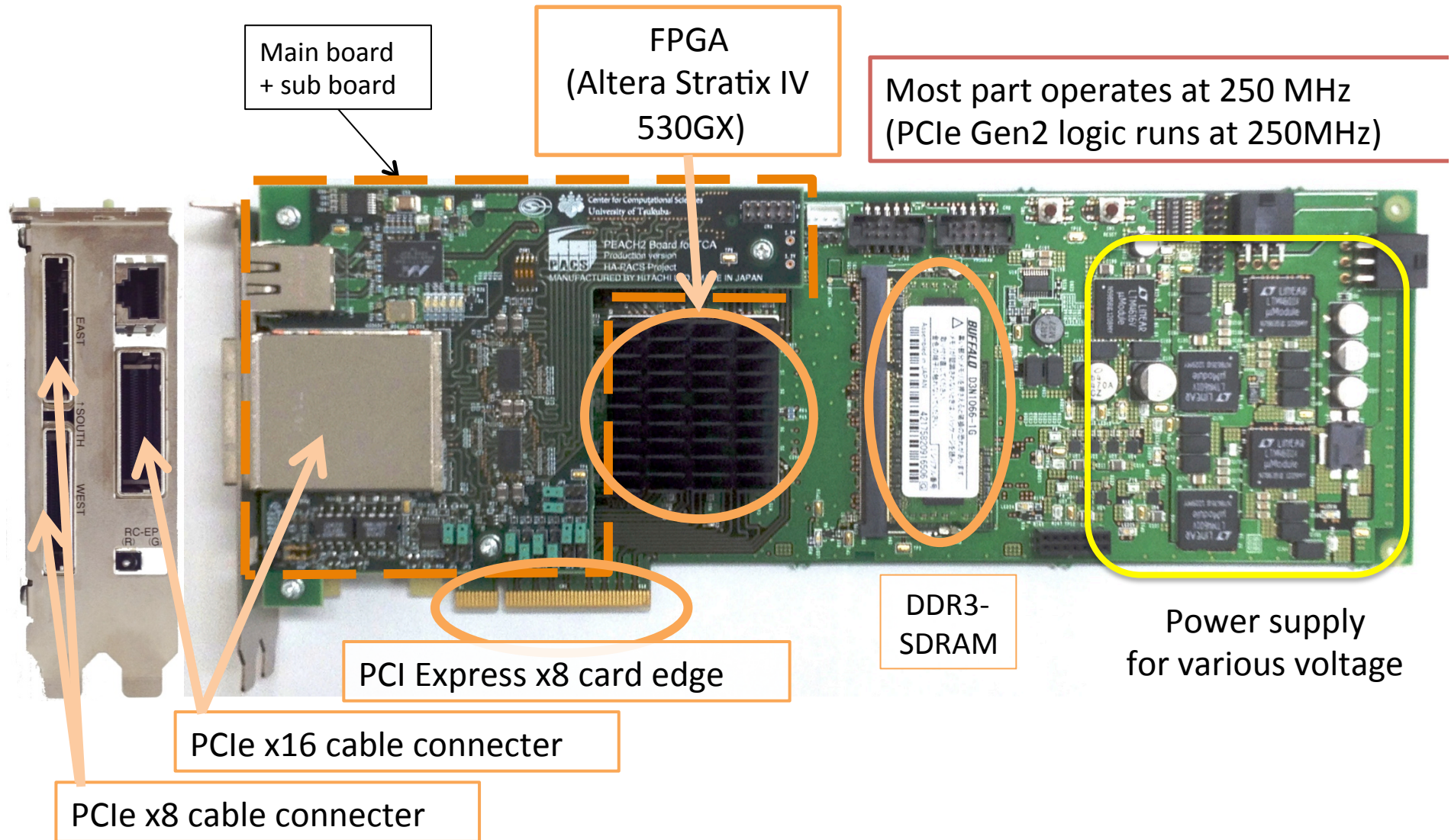
- PCI Express Gen2 x8 peripheral board
 - Compatible with PCIe Spec.



Side View

Top View

PEACH2 board



HA-PACS/TCA

- TCAアーキテクチャのコンセプトを実現し、実証実験を行うためのプラットフォーム
- 全てのノードに4台のGPUとPEACH2ボードを実装
HA-PACS (Highly Accelerated Parallel Advanced System for Computational Sciences) プロジェクト
 - 2011-2013でハード実装と基本部分は終了
 - JST-CRESTプロジェクトで研究を継続
 - PEACH2機能なしの Base Cluster 部
 - PEACH2装備の TCA 部
- Base Cluster 部 = 268 nodes
 - Intel SandyBridge CPU x 2 + NVIDIA M2090 (Fermi) x 4
 - dual rail InfiniBand QDR x2
- TCA 部 = 64 nodes
 - Intel IvyBridge CPU x 2 + NVIDIA K20X (Kepler) x 4
 - PEACH2 + InfiniBand QDR x2

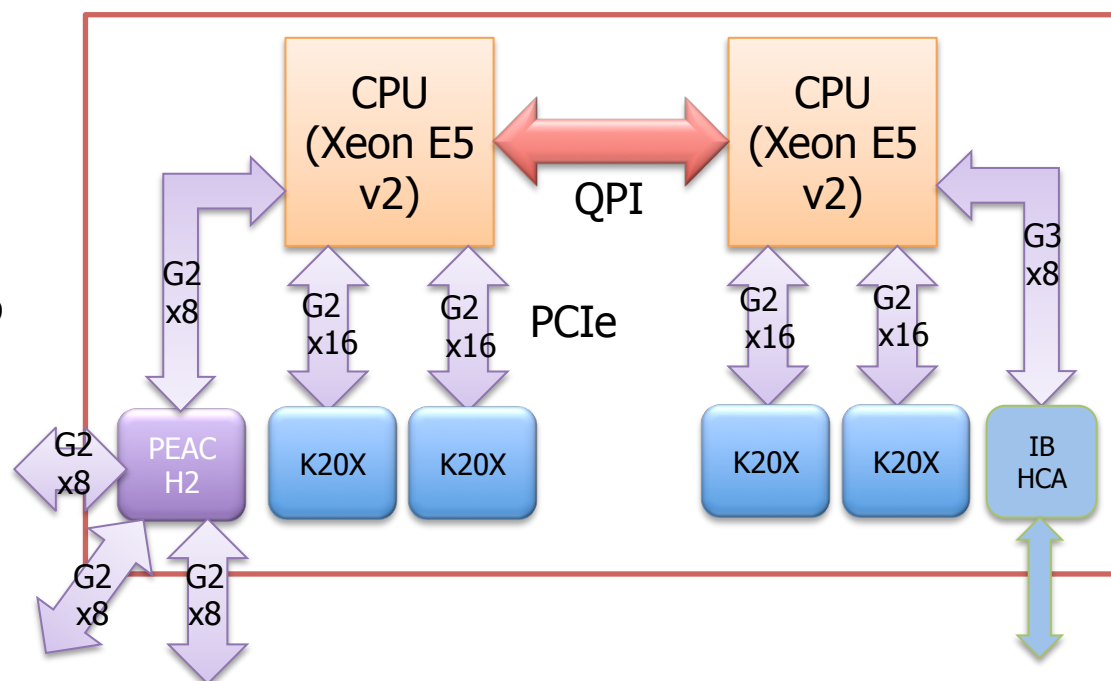
HA-PACS Base Cluster + TCA (Base Cluster: 2012/2 TCA: 2013/11)



- HA-PACS Base Cluster = 2.99 TFlops x 268 node = 802 TFlops
- HA-PACS/TCA = 5.69 TFlops x 64 node = 364 TFlops
- TOTAL: 1.166 PFlops
- TCA part (individually) ranked as #3 in Green500, Nov. 2013

HA-PACS/TCAノード構成

- PEACH2 から全てのGPUをアクセス可能
 - Kepler architecture + CUDA 5.0 “GPUDirect Support for RDMA”
 - QPIを越えたPCIe通信が極めて低速
=> 実験では2台のGPUを使う
- 他の3ノードと接続



HA-PACS/TCA計算ノード内部

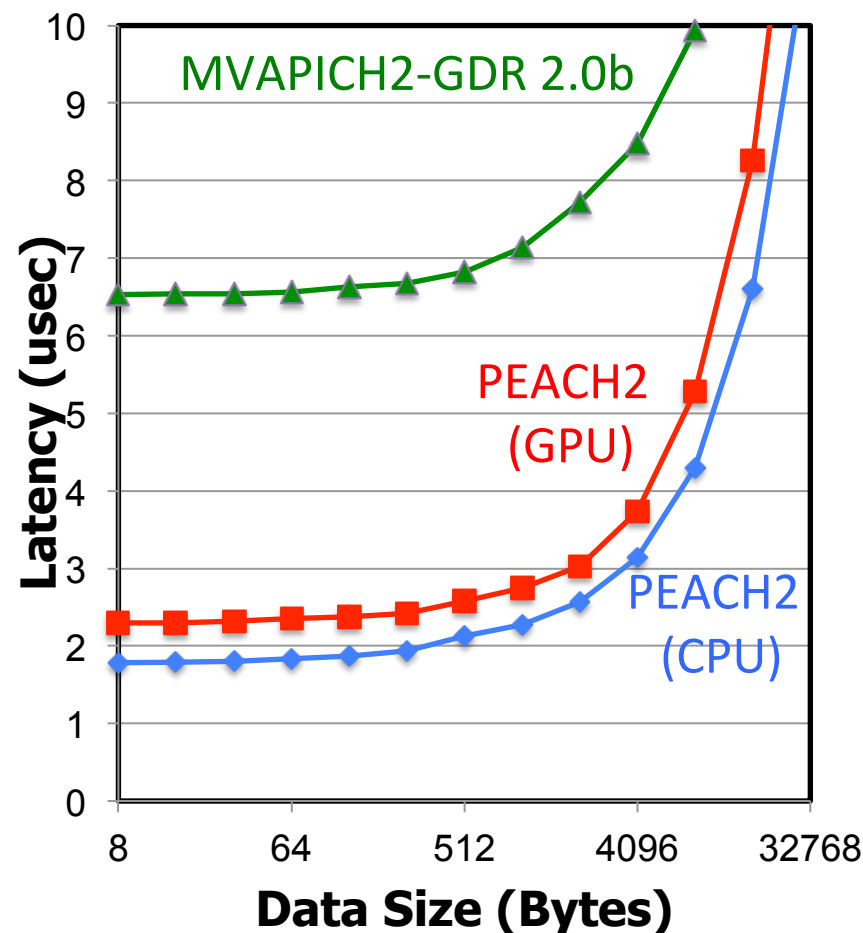


Ping-pong 通信のレイテンシ

最小レイテンシ(nearest neighbor comm.)

- PIO: CPU to CPU: 0.8 us
- DMA: CPU to CPU: 1.8 us
GPU to GPU: 2.3 us

cf. MV2-GDR 2.0b: 6.5 us (w/ GDR),
17 us (w/o GDR)

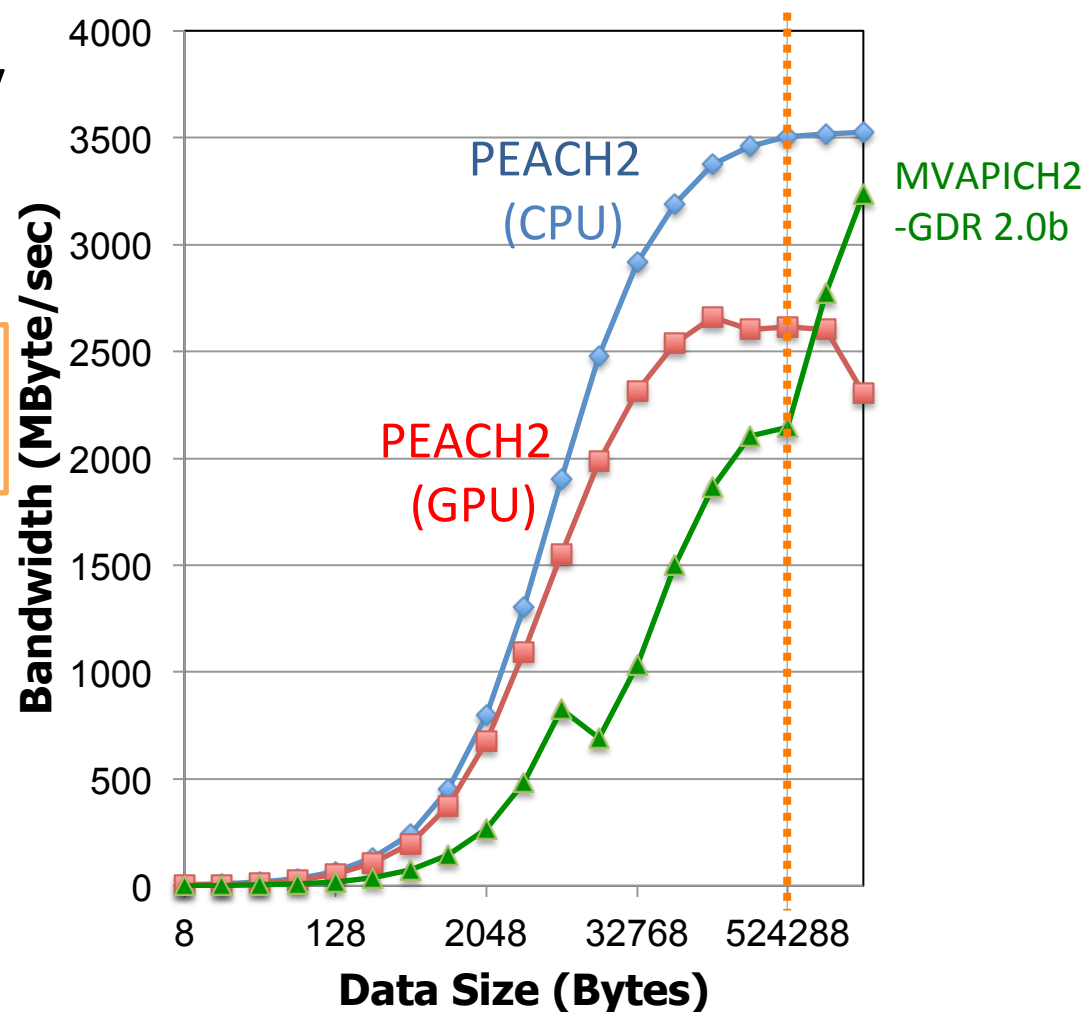


Ping-pong 通信のバンド幅

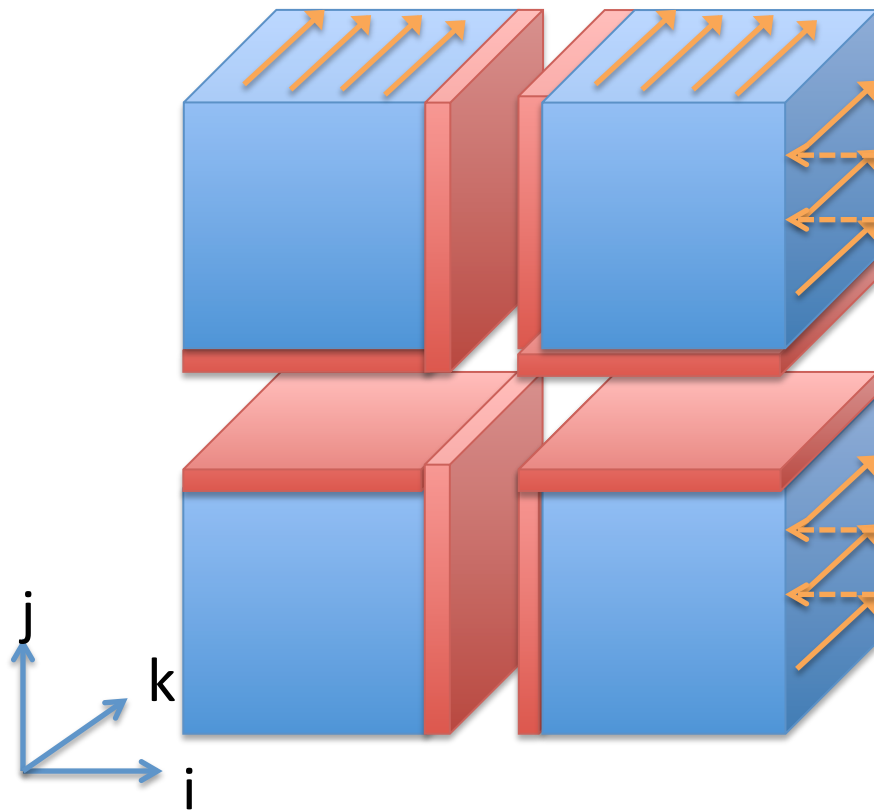
- CPU通信は max 3.5 GByte/sec
 - 理論ピークの95%

Max Payload Size = 256byte
 Theoretical peak (detailed):
 $4\text{GB/sec} \times 256 / (256 + 24) = \underline{3.66} \text{ GB/s}$

- GPU - GPU DMA 通信では max 2.6 GByte/sec.
 - 512KBまではMVAPICH2より高速
 - Over QPI: 360MB/s まで



3次元ステンシル計算の例



3-D decomposition in 1x2x2 (4 nodes)

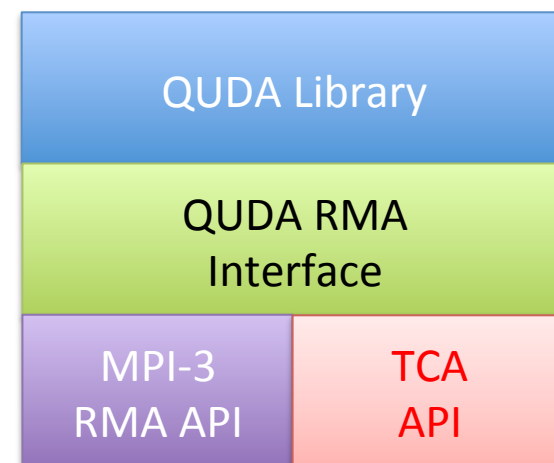
- 3次元データを2次元分割、2つの次元の面で通信(黄色矢印)
- i-k平面は連続転送、j-k平面はblock-stride転送
- 毎時間発展ループでは固定パターンの通信



PEACH2 のDMA chainingが有効に利用可能、block-strideとの組み合わせも可能

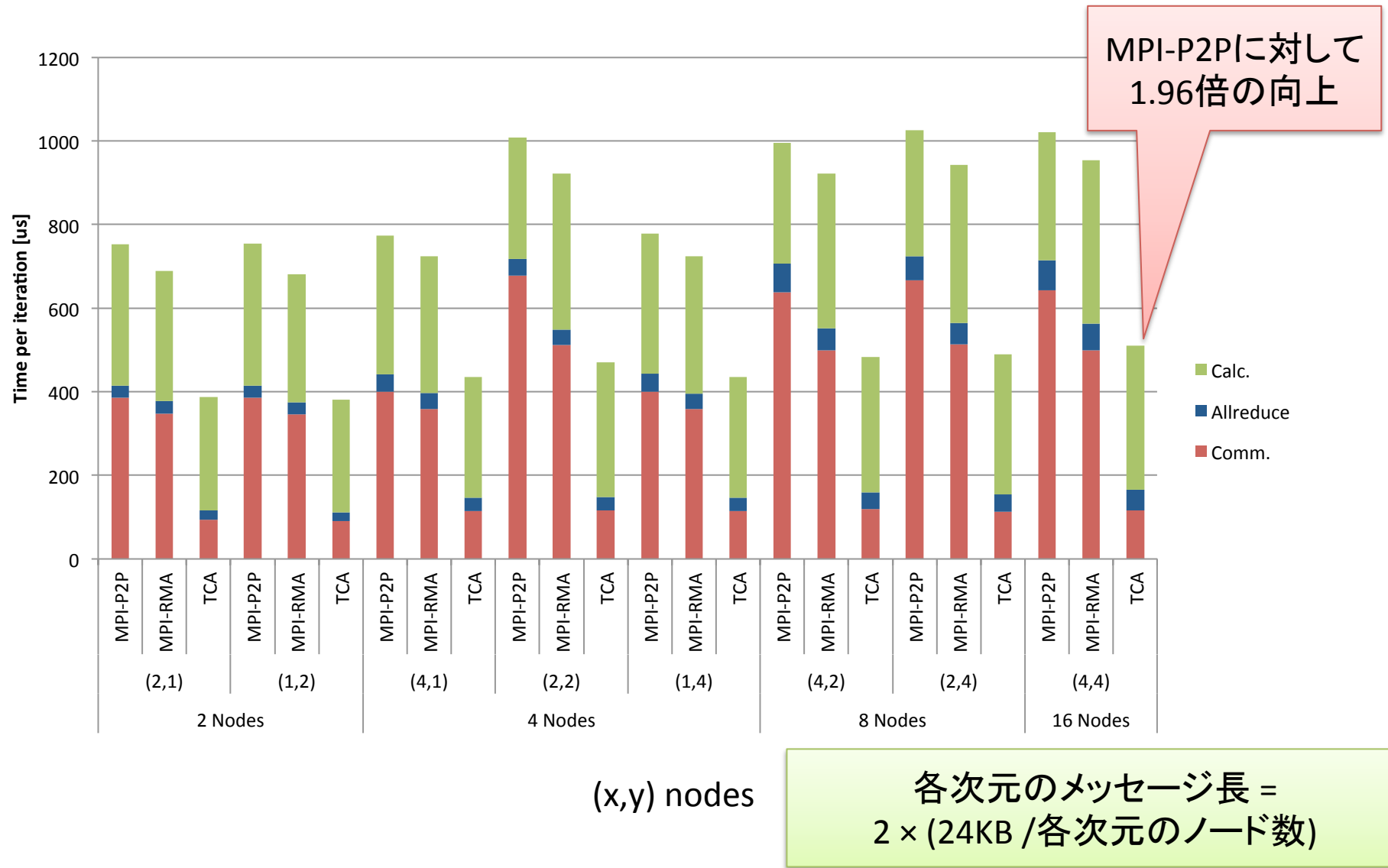
QUDA QCD Library のTCA向け実装

- QUDA: The open source Lattice QCD library
 - NVIDIA GPU向けのLQCDライブラリ
 - 全ての計算をNVIDIA GPU上で行う
- MPI実装のみで、そのままではTCAを適用できない
 - TCAはRDMA Writeで通信を行う
 - RMAをサポートするようにQUDAを拡張する
 - QUDAの通信ライブラリを再構築し、RMA通信ベースで設計
 - MPI-3 RMAでの実装
 - TCAでの実装
- NVIDIA社、Mike Clarkとの共同研究



通信抽象化レイヤーによって複数の通信APIをサポートしている

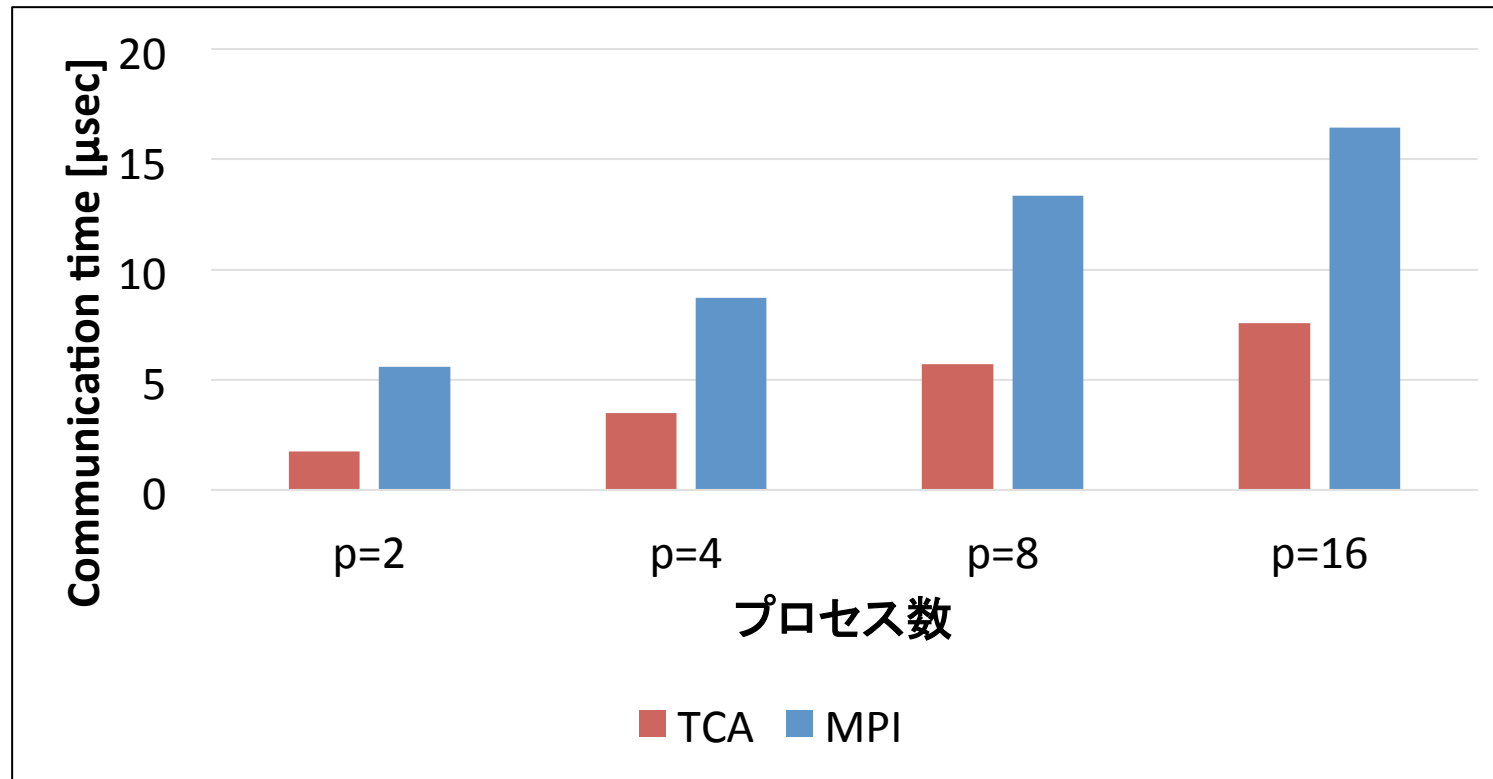
QUDA Small Model (8^4)



Allreduceの性能

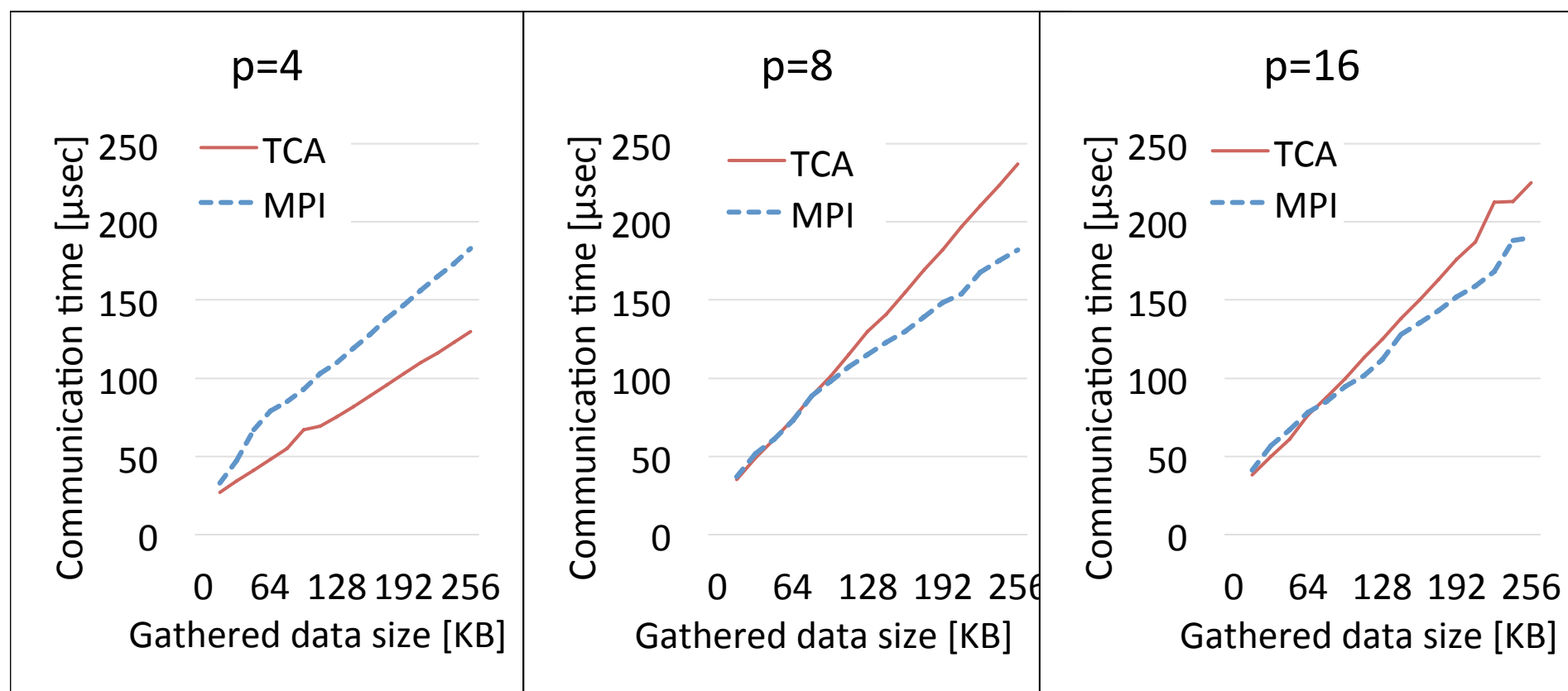
8 Bytesのスカラー値をAllreduceする通信時間(50回の平均)

- MPIを用いた実装の**半分ほどの通信時間**で済む
- TCAの低レイテンシという特徴が有効に働いている

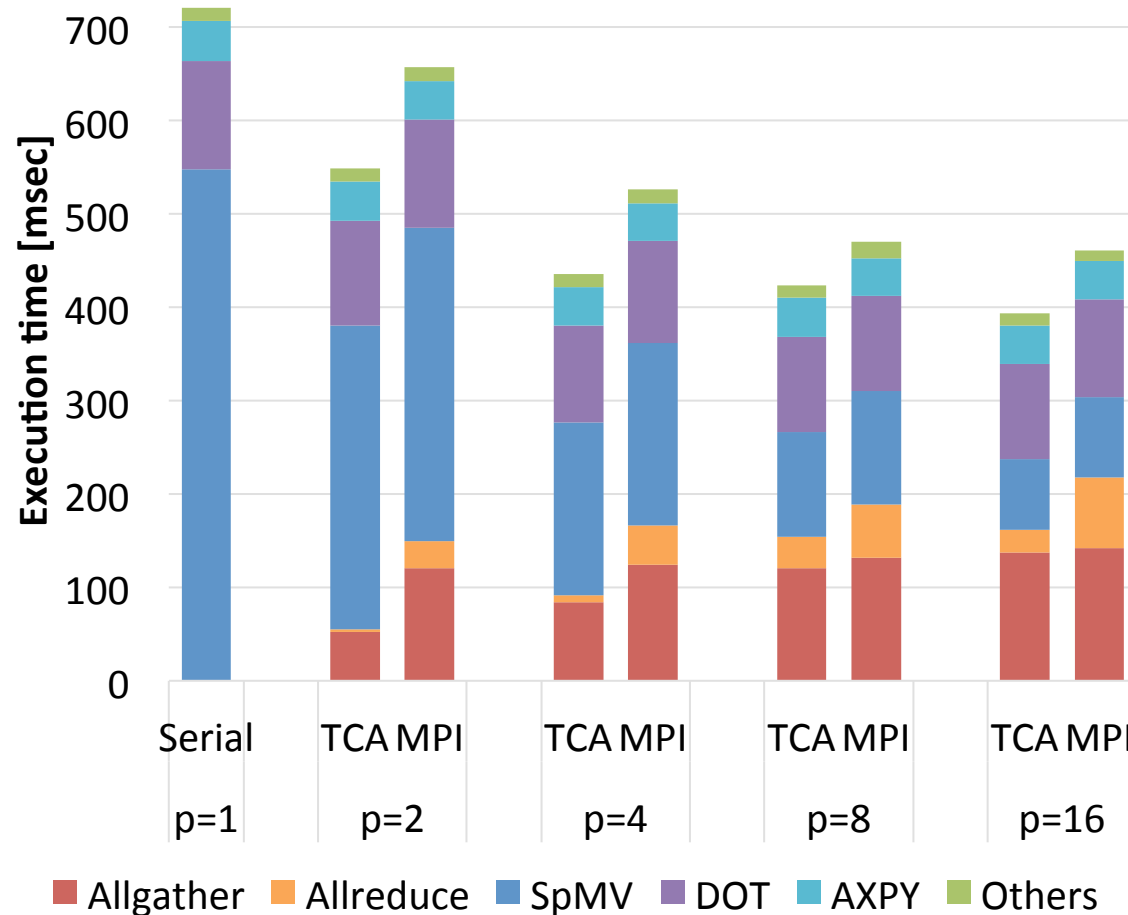


Allgatherの性能:

- HA-PACS/TCAの1サブクラスタ(最大16ノード)における測定結果
- プロセス数 p が増えるとMPI (MVAPICH 2 GDR 2.0b) との性能差が小さくなる(TCAでは通信経路の衝突が起こるため)



CG法の性能



- 1,000回反復を行った時の実行時間
 - GPUは1ノード辺り1GPUを使用
 - プロセスランク0の内訳
- 疎行列名: **nd6k**
 - 行数: 18,000
 - 非零要素数: 6,897,316
 - 先程の倍の大きさの行列
 - Univ. Florida Sparse Matrix Collectionより取得

TCAアーキテクチャのユーザ利用

- 基本APIを使ったプログラミング
 - TCAの初期化、GPUメモリのマッピング、DMA descriptor準備、PIO及びDMAによる通信起動、データ着信確認
 - 基本APIを使ったcollective通信ライブラリ (Allreduce, Allgather, Barrier)
 - TCA/PEACH2を低オーバーヘッドで直接利用可能
 - MPI等との互換性がなく、write-only protocolを用いたアルゴリズムレベルでの書き換えが必要
- XcalableACCによるプログラミング
 - XcalableMP をベースとした演算加速装置向け言語 XcalableACC
 - MPI環境でもTCA環境でも利用可能
 - ユーザアプリケーションを容易に記述可能 (MPIとの互換性を保ちつつ)

XcalableMP/XcalableACCによるプログラミング

- 分散メモリアーキテクチャ向けPGAS言語
- global view & local view の dual memory model
- directive base で他の言語システムとの直交性が(基本的に)ある
 - OpenMP
 - OpenACC
- XcalableMPのアクセラレータ対応
 - XMP-dev (XcalableMP for devices)
 - XcalableACC (XcalableMP + OpenACC)
- 分散メモリ上の通信系を下位レイヤで実装
 - MPI
 - native communication library (京)
 - TCA

XcalableACCのプログラミング例

```
double u[XSIZE][YSIZE], uu[XSIZE][YSIZE];

...

{
  for(k=0; k<MAX_ITER; k++){

    for(x=1; x<XSIZE-1; x++)
      for(y=1; y<YSIZE-1; y++)
        uu[x][y] = u[x][y];

    for(x=1; x<XSIZE-1; x++)
      for(y=1; y<YSIZE-1; y++)
        u[x][y] = (uu[x-1][y]+uu[x+1][y]+
                   uu[x][y-1]+uu[x][y+1])/4.0;
  } // end k
}
```

2次元ラプラス方程式

XcalableACCのプログラミング例

```
double u[XSIZE][YSIZE], uu[XSIZE][YSIZE];
#pragma xmp nodes p(x, y)
#pragma xmp template t(0:YSIZE-1, 0:XSIZE-1)
#pragma xmp distribute t(block, block) onto p
#pragma xmp align [j][i] with t(i,j) :: u, uu
#pragma xmp shadow uu[1:1][1:1]
...

for(k=0; k<MAX_ITER; k++){
#pragma xmp loop (y,x) on t(y,x)

    for(x=1; x<XSIZE-1; x++)
        for(y=1; y<YSIZE-1; y++)
            uu[x][y] = u[x][y];

#pragma xmp reflect (uu)

#pragma xmp loop (y,x) on t(y,x)

    for(x=1; x<XSIZE-1; x++)
        for(y=1; y<YSIZE-1; y++)
            u[x][y] = (uu[x-1][y]+uu[x+1][y]+
                      uu[x][y-1]+uu[x][y+1])/4.0;
} // end k
```

2次元ラプラス方程式

分散配列と袖の定義

配列uuの袖交換

XcalableACCのプログラミング例

```
double u[XSIZE][YSIZE], uu[XSIZE][YSIZE];
#pragma xmp nodes p(x, y)
#pragma xmp template t(0:YSIZE-1, 0:XSIZE-1)
#pragma xmp distribute t(block, block) onto p
#pragma xmp align [j][i] with t(i,j) :: u, uu
#pragma xmp shadow uu[1:1][1:1]
...
#pragma acc data copy(u) copyin(uu)
{
    for(k=0; k<MAX_ITER; k++){
        #pragma xmp loop (y,x) on t(y,x)
        #pragma acc parallel loop collapse(2)
        for(x=1; x<XSIZE-1; x++){
            for(y=1; y<YSIZE-1; y++){
                uu[x][y] = u[x][y];

        #pragma xmp reflect (uu) acc

        #pragma xmp loop (y,x) on t(y,x)
        #pragma acc parallel loop collapse(2)
        for(x=1; x<XSIZE-1; x++){
            for(y=1; y<YSIZE-1; y++){
                u[x][y] = (uu[x-1][y]+uu[x+1][y]+
                    uu[x][y-1]+uu[x][y+1])/4.0;
            } // end k
        } // end data
```

2次元ラプラス方程式

分散配列と袖の定義

分散配列をアクセラレータの
メモリに転送

XMP指示文で分散したループを
OpenACC指示文が分散して処理

配列uuの袖交換

acc節を指定すると、アクセラ
レータ上のデータが送信される

XcalableACCのプログラミング例

```
double u[XSIZE][YSIZE], uu[XSIZE][YSIZE];
```

```
...  
#pragma acc data copy(u) copyin(uu)  
{  
    for(k=0; k<MAX_ITER; k++){  
  
        #pragma acc parallel loop collapse(2)  
        for(x=1; x<XSIZE-1; x++)  
            for(y=1; y<YSIZE-1; y++)  
                uu[x][y] = u[x][y];  
  
        #pragma acc parallel loop collapse(2)  
        for(x=1; x<XSIZE-1; x++)  
            for(y=1; y<YSIZE-1; y++)  
                u[x][y] = (uu[x-1][y]+uu[x+1][y]+  
                        uu[x][y-1]+uu[x][y+1])/4.0;  
    } // end k  
} // end data
```

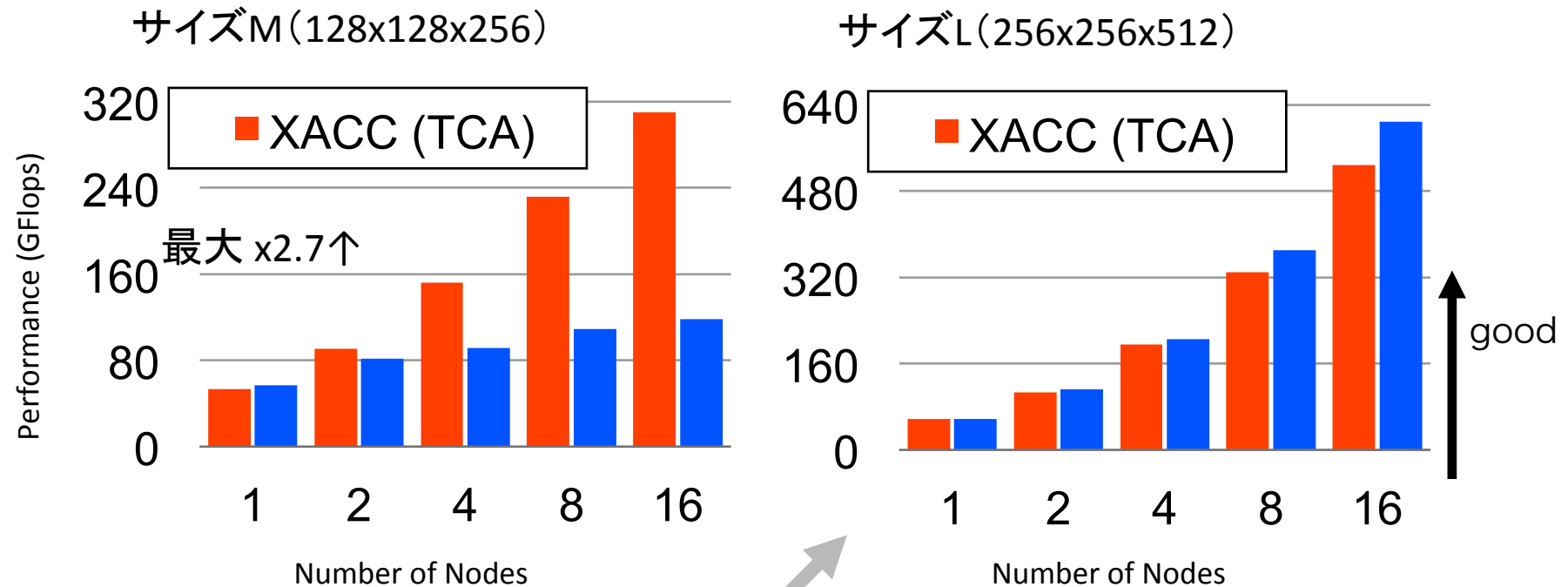
2次元ラプラス方程式

分散配列をアクセラレータの
メモリに転送

XMP指示文で分散したループを
OpenACC指示文が分散して処理

XcalableACCの性能評価一例

姫野ベンチマークのサイズS, M, L(サイズSの結果の傾向はサイズMと同じ)



サイズLの場合、袖領域の大きさは約520KByteであり、TCAとMVAPICH2-GDRの性能差は少ない。さらに、TCAはDMAの後にバリア同期が必要であるため、TCAの袖交換に要する時間はMVAPICH2-GDRの時間よりも少し長くなる

PEACH2 and InfiniBand

- TCA/PEACH2 では PCIe のアドレス空間制約と3-port実装 (FPGAの制約)により、トポロジ的なノード数が制限されている。
- HA-PACS/TCAは現時点で 16 node x 4 partition になっている。3-port なので2次元トーラスのうち1次元は一対一接続のみ。今後32 nodeぐらいまでは実験する予定。
- InfiniBandとの組み合わせを階層的に利用し、低階層の近接通信をTCAで、高階層の広域通信をInfiniBandで実現するシステム開発を開始。
- 2つの方法
 - XcalableACCの通信レイヤ上で実現
 - InfiniBand互換のVerbsライブラリをPEACH2に実装
⇒ MPI実装まで？

まとめ

- TCA architecture with PEACH2 による超低レイテンシGPU device間通信システムの実現
- JST-CRESTの枠組みの中でハードの発展開発、システムソフトウェア、ライブラリ、言語、アプリケーションまでを統一的に実現
- native API アプリケーション(ライブラリ)としてQUDA, CG法を開発中
- XcalableMP/XcalableACCの開発を推進
⇒ より user friendly なTCAの利用へ