PGAS言語XcalableMPを用いた HPC Challengeベンチマークの実装と評価

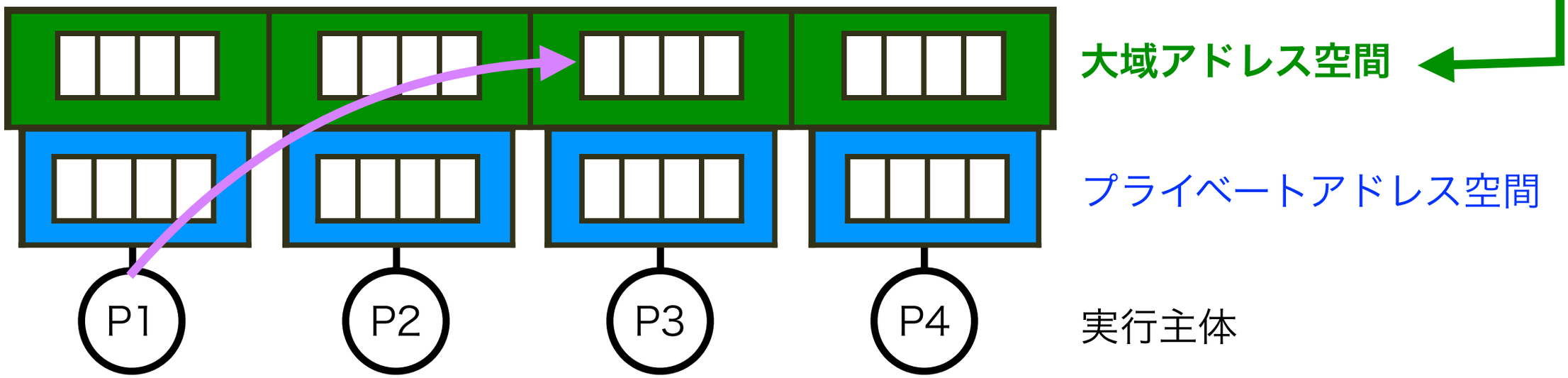
中尾 昌広,[1,2] 村井 均,[1] 岩下 英俊[1]
下坂 健則,[1] 朴 泰祐,[2,3] 佐藤 三久[1,2,3]

1. 理化学研究所 計算科学研究機構
2. 筑波大学 計算科学研究センター
3. 筑波大学大学院 システム情報工学研究科

第148回HPC研究会@別府温泉，2015年3月

背景：PGAS言語モデル

- Partitioned Global Address Space（PGAS）：分散メモリ環境向けの言語モデル
  - **XcalableMP（XMP）**，X10，Coarray Fortran，Unified Parallel C（UPC），UPC++，HabaneroUPC++，Chapel，PCJ，など
  - プロセスなどの実行主体同士は**透過的にアクセス可能な領域（大域アドレス空間）**を介してデータのread/writeを行う
    - **大域アドレス空間**は論理的に区切られており，各実行主体と対応している
  - データの局所性を意識しつつ，共有メモリに似たプログラミングが可能
  - 生産性の性能のバランスが良いモデル



目的

- 大規模環境におけるXMPの評価
  - HPC ChallengeベンチマークのXMPによる実装
  - スーパーコンピュータ「京」を用いて、生産性と性能を評価

- 上記の作業を通して，XMPを用いた並列化とそのチューニングの
ノウハウの蓄積

The K computer: 82,944 nodes

http://www.aics.riken.jp/jp/outreach/photogallery.html

- SPARC64 VIIIfx Chip, 128 GFlops
- DDR3 SDRAM 16GB, 64GB/s
- Tofu Interconnect
  - 6D mesh/torus network
  - 5GB/s x 4links x 2

発表の流れ

- XMPについて（グローバルビューとローカルビューのプログラム例）
- HPC Challengeベンチマークについて
- XMPを用いたHPC Challengeベンチマークの実装と性能評価
- まとめ

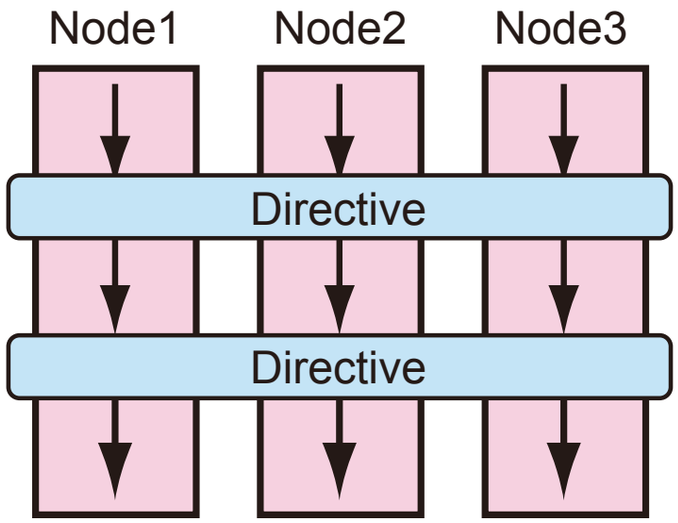
発表の流れ

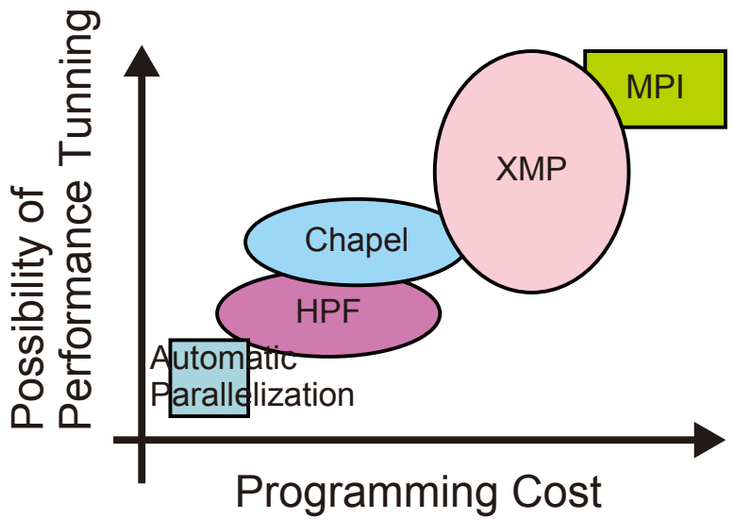
- XMPについて（グローバルビューとローカルビューのプログラム例）
- HPC Challengeベンチマークについて
- XMPを用いたHPC Challengeベンチマークの実装と性能評価
- まとめ

XMP

- 既存言語（C99とFortran95）を指示文と拡張構文で並列化
  - 記法のいくつかはCoarray FortranとHPFから継承
- 実行モデルはMPIと同じSPMD
  - 指示文がなければ、重複実行
  - 指示文とCoarray記法により通信・同期を実行
- 様々なアプリケーションに対応するための並列化ビュー
  - グローバルビュー：各実行主体が一部の名前空間を共有
    （指示文による典型的なデータ並列のサポート）
  - ローカルビュー：各実行主体が固有の名前空間を持つ
    （Coarrayによる片側通信）
- PCクラスタコンソーシアムにより仕様書の策定
- Omni XMP Compiler（http://omni-compiler.org）
  - 「京」，FX10，SX9，SR-16000，BG/Q，
    Cray XE6 など

XMPの実行主体は”**node**”と呼称





Code example (グローバルビュー)

```
int a[MAX];
#pragma xmp nodes p(4)
#pragma xmp template t(0:MAX-1)
#pragma xmp distribute t(block) on p
#pragma xmp align a[i] with t(i)

main(){
  int i, j, res = 0;

#pragma xmp loop on t(i) reduction(+:res)
  for(i = 0; i <MAX; i++){
    a[i] = func(i);
    res += array[i];
  }
```

データ分散の定義

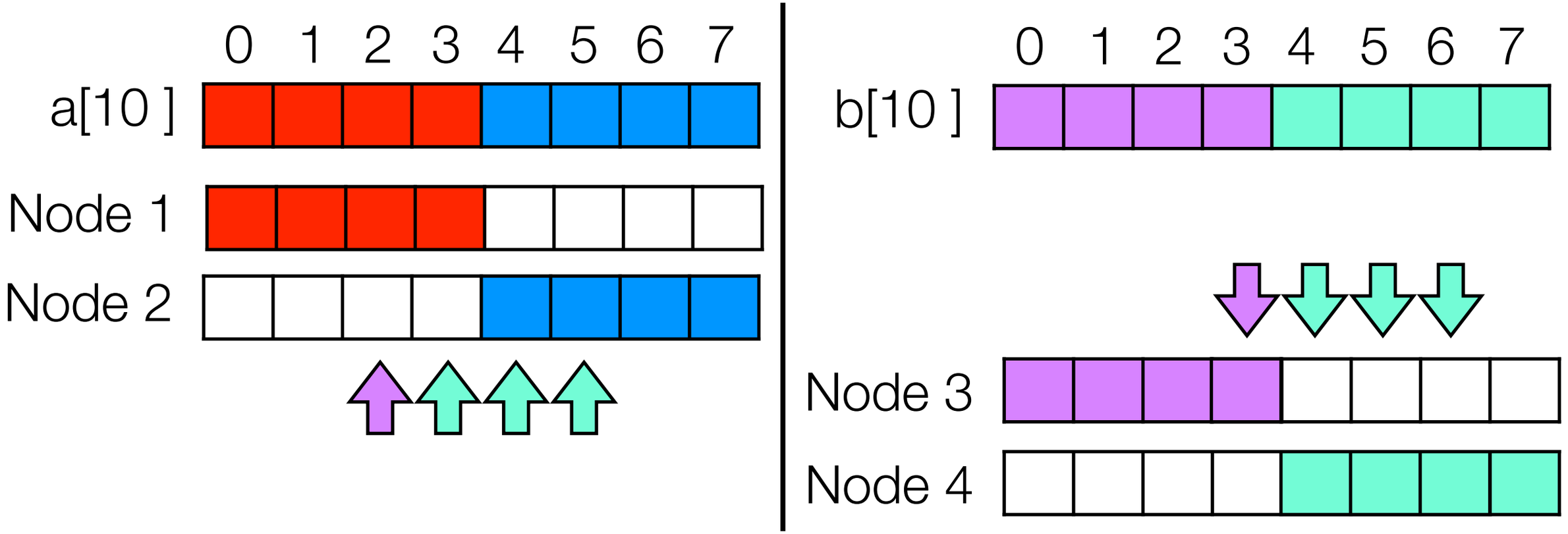
データ並列と集約演算

Code example (グローバルビュー)

- 通信指示文 : broadcast, reduction, gmove など
- gmove 指示文
  - グローバルなイメージを保ちつつ，通信を記述
  - array section notation

[start_index : length]

```
#pragma xmp gmove
a[2:4] = b[3:4];
```



Code example (ローカルビュー)

```
double a[100]:[*], b[100]:[*];
int me = xmp_node_num();

if(me == 2)
  a[:]:[1] = b[:];

if(me == 1)
  a[0:50] = b[0:50]:[2];
```

Coarrayの定義

Put通信

Get通信

Coarray synax in XMP/C

```
array_name[start_index:length]:[node_number];
```

XMP/FortranはFortran 2008のCoarray機能の上位互換

発表の流れ

- XMPについて（グローバルビューとローカルビューのプログラム例）
- HPC Challengeベンチマークについて
- XMPを用いたHPC Challengeベンチマークの実装と性能評価
- まとめ

HPC Challengeベンチマーク

- 7つからなる，HPCシステムを多角的に評価するためのベンチマークセット
  - MPIによるリファレンス実装（http://www.hpcchallenge.org）
- HPC Challenge Awards Competition@SC
  - In Class 1, only the performance of an HPC system is evaluated
  - In Class 2, the productivity and performance of a programming language are evaluated（コードのエレガントさを50%、性能を50%で評価）
- Class 2に提出するために，下記の4つのベンチマークを実装

Benchmark	説明
STREAM	実効メモリバンド幅を計測
HPL	密行列連立一次方程式を解く
RandomAccess	分散テーブルをランダムにアクセス
FFT	1次元離散複素フーリエ変換に対する速度

XMPの成果

- 2013年は，「京」を用いて評価し，HPC Challenge Class 2で受賞（日本初）
- 2014年は，2013年に作成した各ベンチマークに対してチューニング
  - 2年連続受賞

発表の流れ

- XMPについて（グローバルビューとローカルビューのプログラム例）
- HPC Challengeベンチマークについて
- XMPを用いたHPC Challengeベンチマークの実装と性能評価
- まとめ

HPC Challengeベンチマークの実装

Benchmark	説明
STREAM	実効メモリバンド幅を計測
HPL	密行列連立一次方程式を解く
RandomAccess	分散テーブルをランダムにアクセス
FFT	1次元離散複素フーリエ変換に対する速度

- 上記4つのXMPを用いた実装について説明
- 2013年と2014年の比較を中心に，”どんなチューニングをして，どのくらい性能が向上したか”について述べる（**結果：37〜94%の性能向上**）
  - ベンチマークのコードの改良：STREAM，HPL，FFT
  - Omni XMP Compilerの改良：RandomAccess，FFT
- 各XMPの結果を，Class 1に登録されている「京」の結果もしくはMPIによる結果との比較を行う

STREAMの実装

- 実効メモリバンド幅を計測する
- カーネル関数において通信が発生しない，非常に単純なベンチマーク

2013年

```
#pragma xmp nodes p(*)



#pragma omp parallel for
  for (i=0; i<N; i++)
    a[i] = b[i] + scalar*c[i];

#pragma xmp reduction(+:triadGBs)
```

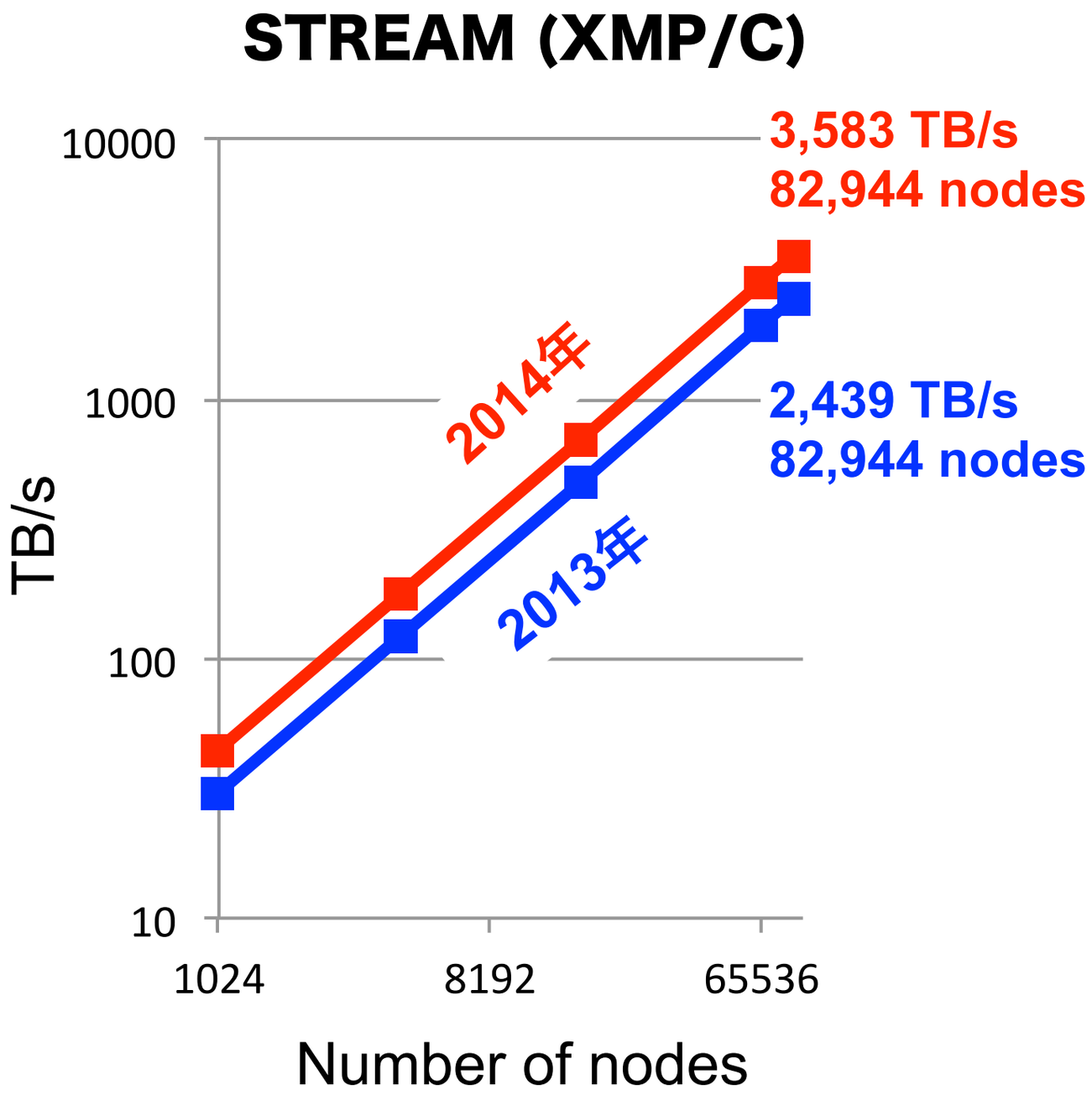
2014年

```
#pragma xmp nodes p(*)

#pragma loop xfill
#pragma loop noalias
#pragma omp parallel for
  for (i=0; i<N; i++)
    a[i] = b[i] + scalar*c[i];

#pragma xmp reduction(+:triadGBs)
```

富士通コンパイラの最適化指示文を追加

実装の考察：XMPの実行モデルが明確であるため、
XMP以外の部分に対する最適化と干渉しない

STREAMの結果



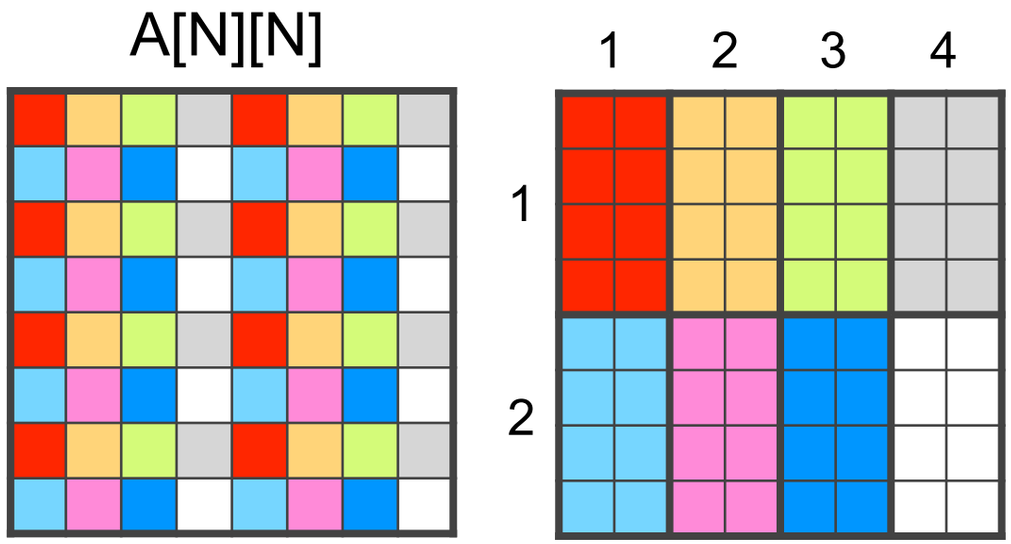
1.47倍の性能向上

Class 1の結果は
3,857 TB/s

HPL version 1

- 密行列連立一次方程式を解く
- ブロックサイクリック分散

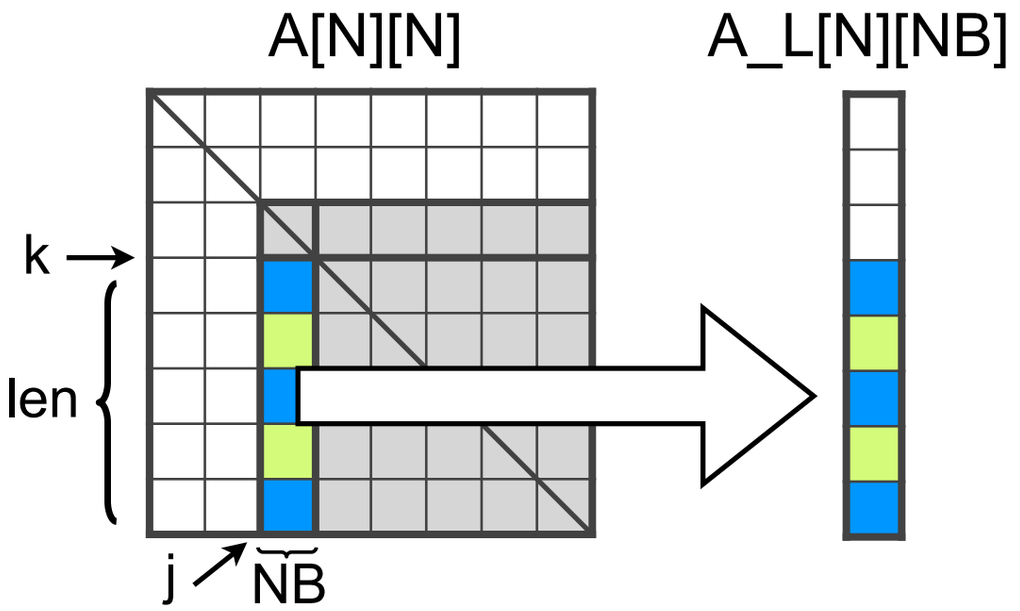
```
double A[N][N];
#pragma xmp nodes p(P,Q)
#pragma xmp template t(0:N-1, 0:N-1)
#pragma xmp distribute t(cyclic(NB), \
                         cyclic(NB)) onto p
#pragma xmp align A[i][j] with t(j,i)
```



行列積には「京」が提供しているBLASを利用

- **gmove**指示文を用いたパネル転送

```
double A_L[N][NB];
#pragma xmp align A_L[i][*] with t(*,i)
     :
#pragma xmp gmove
A_L[k:len][0:NB] = A[k:len][j:NB];
```

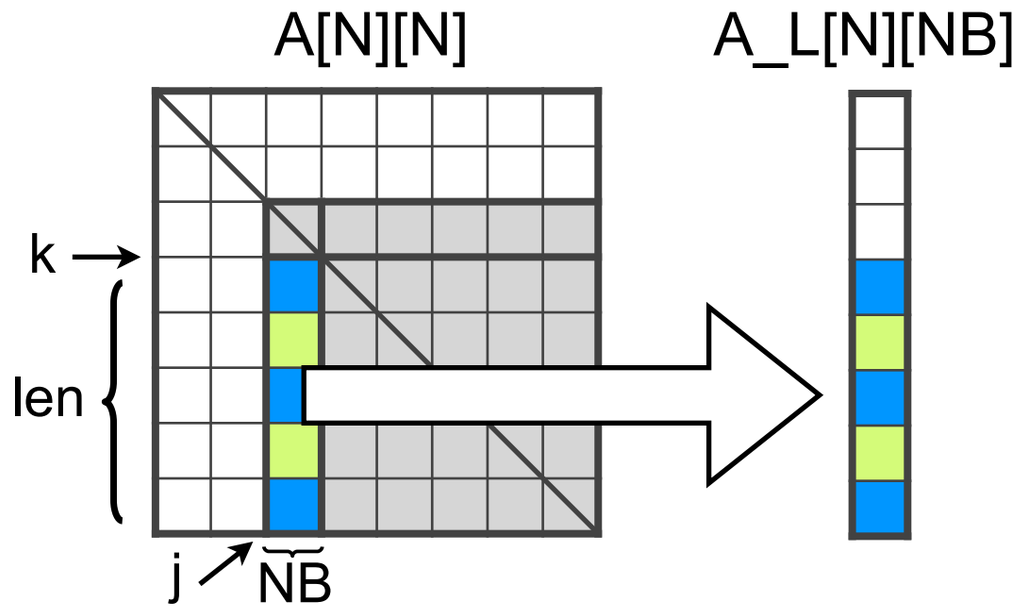


HPL version 2

- **gmove**指示文と**async**節を用いたLookaheadアルゴリズムの実装

  通信と計算のオーバラップ

```
double A_L[N][NB];
#pragma xmp align A_L[i][*] with t(*,i)
     :
#pragma xmp gmove async(1)
A_L[k:len][0:NB] = A[k:len][j:NB];
     :
for(m=j+NB;m<N;m+=NB){
  for(n=j+NB;n<N;n+=NB){
    cblas_dgemm(&A[m][n], ..);
    if(xmp_test_async(1)){
       // receive A[k:len][j:NB];
            :
```

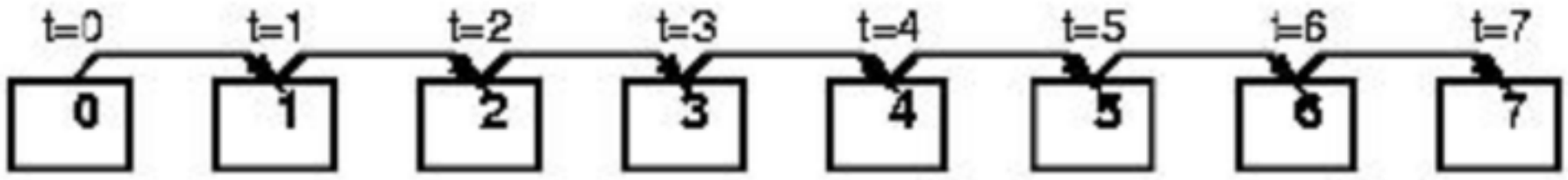


非同期Broadcast通信

非同期通信が来ているかどうかをチェック

非同期bcastの実装

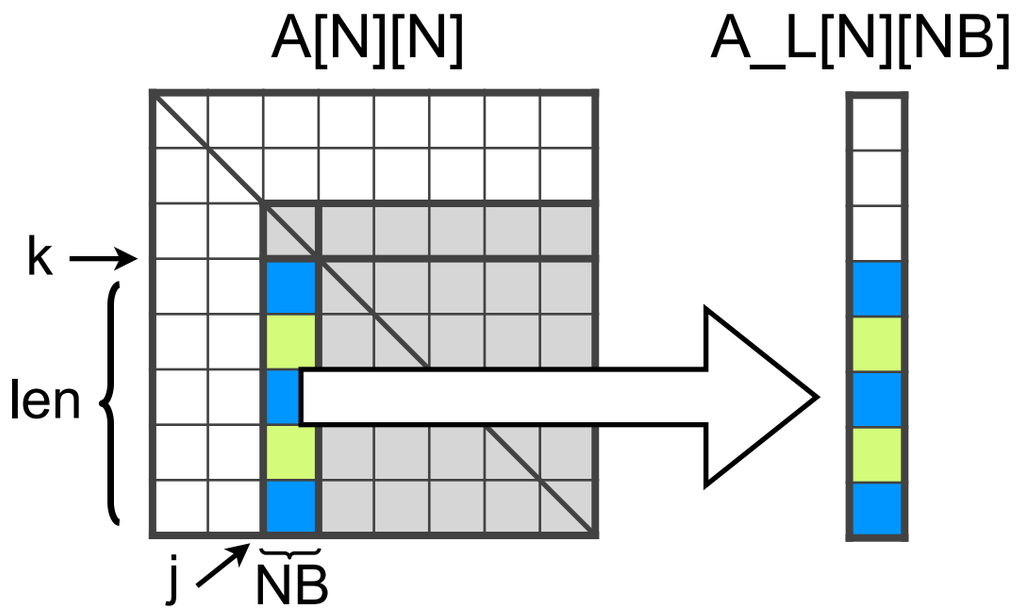
- 今回はIncreasing-ring



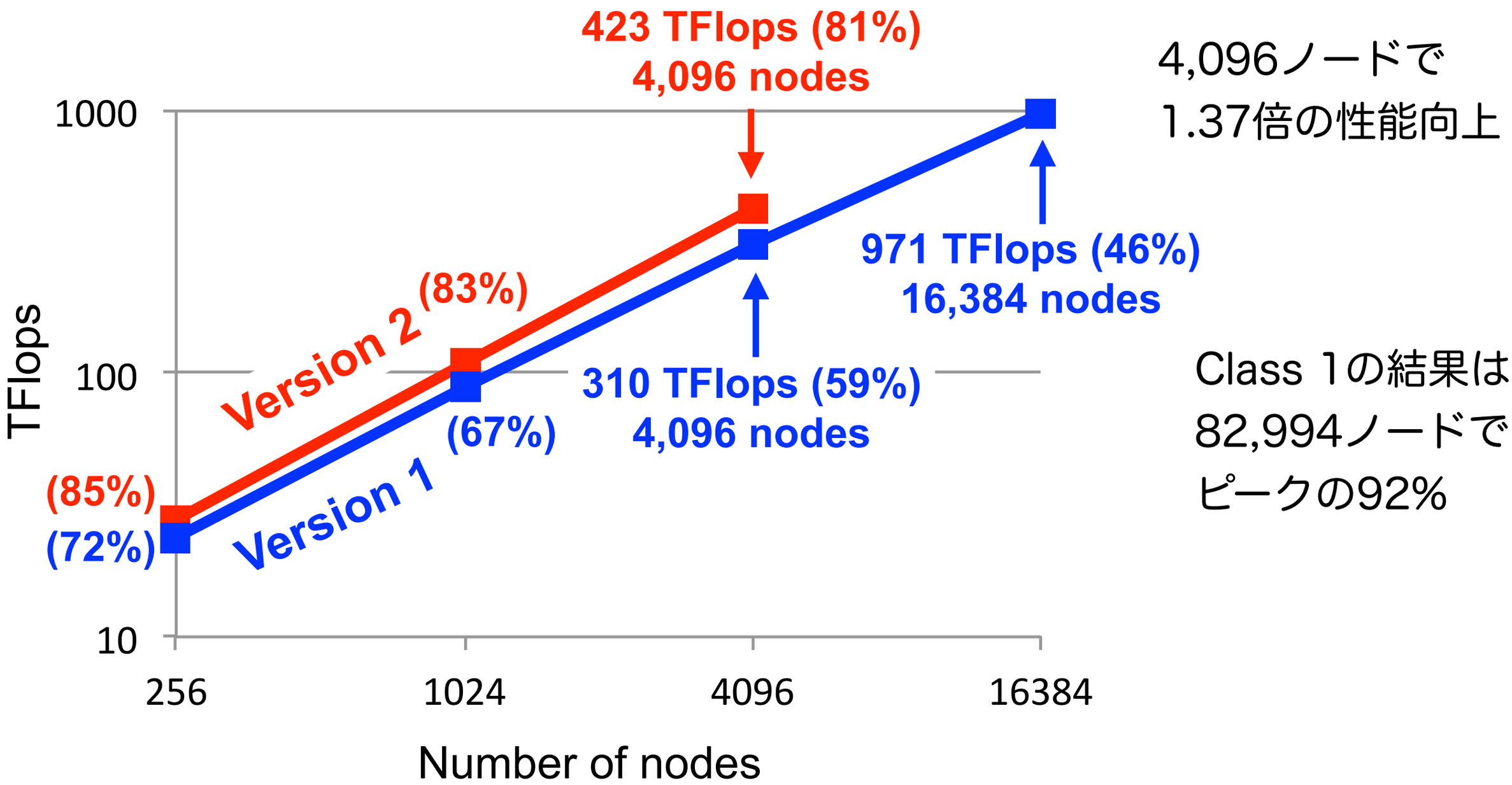
http://www.netlib.org/benchmark/hpl/algorithm.html

- MPI3が使える環境では（将来的には）MPI_Ibcastを利用

```
double A_L[N][NB];
#pragma xmp align A_L[i][*] with t(*,i)
     :
#pragma xmp gmove
A_L[k:len][0:NB] = A[k:len][j:NB];
```



HPLの結果



4,096ノードで
1.37倍の性能向上

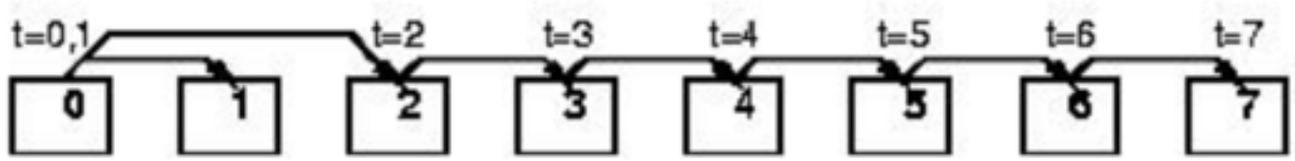
Class 1の結果は
82,994ノードで
ピークの92%

XMP-HPL Version 2の方がスケーラビリティが高い

HPLの実装の考察

- 生産性について
  - 係数行列をブロックサイクリック分散の簡易化
  - XMPが定義した分散配列に対するBLASの適用
  - パネルブロードキャストの簡易化
    - 非同期通信も可能
    - Version 1の行数は313に対して，Version 2の行数は426

- 性能について
  - ノード内部のチューニングに余地（Top500の「京」の結果はピークの93%）
  - パネル転送の性能向上？（例えば，modified-Increasing-ring）



Coarrayを用いると可能だが，
生産性は下がる

http://www.netlib.org/benchmark/hpl/algorithm.html

RandomAccessの実装

- ノードに分散されたテーブルの要素に対して排他的論理和をとりつつ，
  各ノードにアクセス（リファレンス実装では，チャンク毎に2分木で通信）
- XMP/C coarrayを用いたローカルビュー
- 基本的なアルゴリズムはリファレンス実装と同じ
  - リファレンス実装はMPI_Send/Recvなので，それを片側通信で記述
- Post指示文とWait指示文を用いたP2Pバリア

```
u64Int recv[LOGPROCS][RCHUNK+1]:[*];
...
for (j = 0; j < logNumProcs; j++) {
  recv[j][0:num]:[i_partner] = send[i][0:num];

#pragma xmp sync_memory
#pragma xmp post(p(i_partner), 0)
    :
#pragma xmp wait(p(j_partner))
}
```

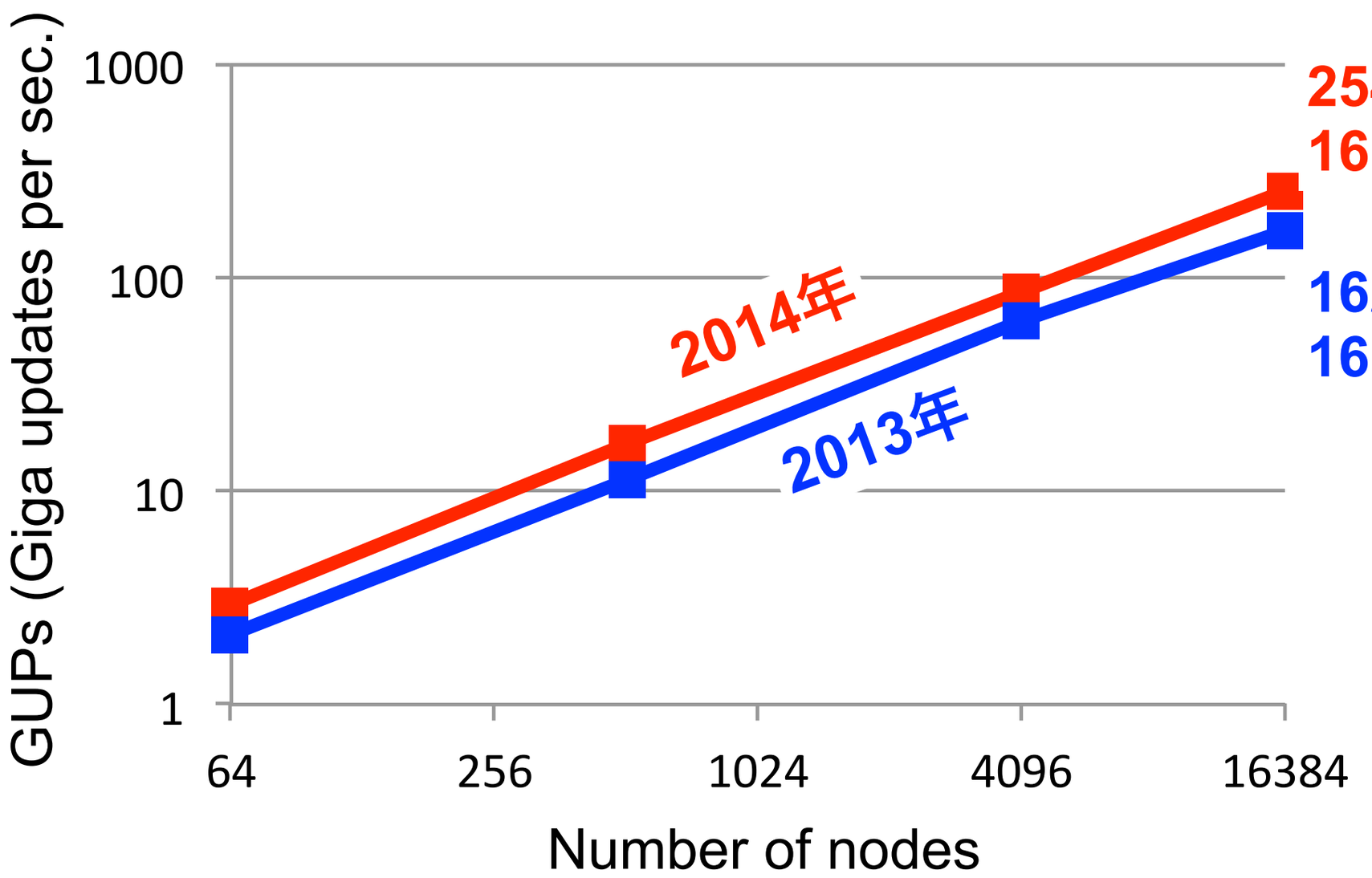
Coarrayの定義

Put operation

post/wait指示文を用いたP2Pバリア

RandomAccessの評価

2013年は post/wait指示文を MPI_Send/Recvを用いて実装.
2014年は post/wait指示文を「京」のRDMAを用いて実装.



1.56倍の
性能向上

254 GUPs
16,384 nodes

162 GUPs
16,384 nodes

MPIによるRandom Accessの結果は16,384ノードで243 GUPs

RandomAccessの実装の考察

- 生産性について
  - MPI関数をCoarray syntaxに置き換えただけ
  - MPI関数よりも直感的にデータの送信を行えていると言えるが・・・

- 性能について
  - 「京」におけるCoarray syntaxは，「京」が提供しているRDMA関数に置き換わる
  - pingpongでは，MPIよりもRDMA-PUTの方がレイテンシは良い

Coarray Syntaxを用いることで，「京」のRDMAを簡易に利用できる

「京」が提供するRDMA関数の例

```
#include <mpi.h>
#include <mpi-ext.h>

#define FLAG_NIC (FJMPI_RDMA_LOCAL_NIC0 | FJMPI_RDMA_REMOTE_NIC1 | FJMPI_RDMA_IMMEDIATE_RETURN)
#define SEND_NIC FJMPI_RDMA_LOCAL_NIC0
struct FJMPI_Rdma_cq cq;
 :
MPI_Init(&argc, &argv);
FJMPI_Rdma_init();
MPI_Comm_rank(MPI_COMM_WORLD, &me);
target = 1 - me;

laddr = FJMPI_Rdma_reg_mem(0, buf, sizeof(char)*MAX_SIZE);
while((raddr = FJMPI_Rdma_get_remote_addr(target, 0)) == FJMPI_RDMA_ERROR);

if(me == 0){
  FJMPI_Rdma_put(target, 0, raddr, laddr, size, FLAG_NIC);
  while(buf[0] == '0' || buf[size-1] == ‘0')
    FJMPI_Rdma_poll_cq(SEND_NIC, &cq);

  buf[0] = buf[size-1] = '0';
     :
FJMPI_Rdma_finalize();
MPI_Finalize();
```

領域登録のために必要なIDやキューを意識したプログラミング

FFTの実装

- 基本的なアルゴリズムはリファレンス実装と同じ
  - 1次元離散複素FFTを six-step FFTアルゴリズムで解く
  - 2次元配列表現 + 転置（All-to-all通信）+ FFTE 6.0

- 2013年と2014年の違い
  - ベンチマークコードの変更
    - 測定区間内にあった，測定区間外に置くことができる指示文を移動
    - コードの全体的なクリーンナップ
  - XMP処理系の変更（不必要なMPI_Comm_split()の呼び出しの削除）
    - 参照：XcalableMPによる効率的なFFTの実装方法，2014-HPC-143(34),1-8
  - その他
    - 1プロセスがスレッド化したFFTEライブラリを呼ぶように変更

FFTの実装

```
complex*16 a(nx,ny), b(ny,nx)
!$xmp template tx(nx)
!$xmp template ty(ny)
!$xmp distribute tx(block) onto p
!$xmp distribute ty(block) onto p
!$xmp align a(*,i) with ty(i)
!$xmp align b(*,i) with tx(i)
 :
!$xmp loop on tx(i)
do i=1,nx
  call zfft1d(b(1,i),ny,-1,cy)
end do

!$xmp loop on tx(i)
!$omp parallel do
do i=1,nx
  do j=1,ny
    b(j,i)=b(j,i)*w(j,i)
  end do
end do

call xmp_transpose(a,b,1)
```

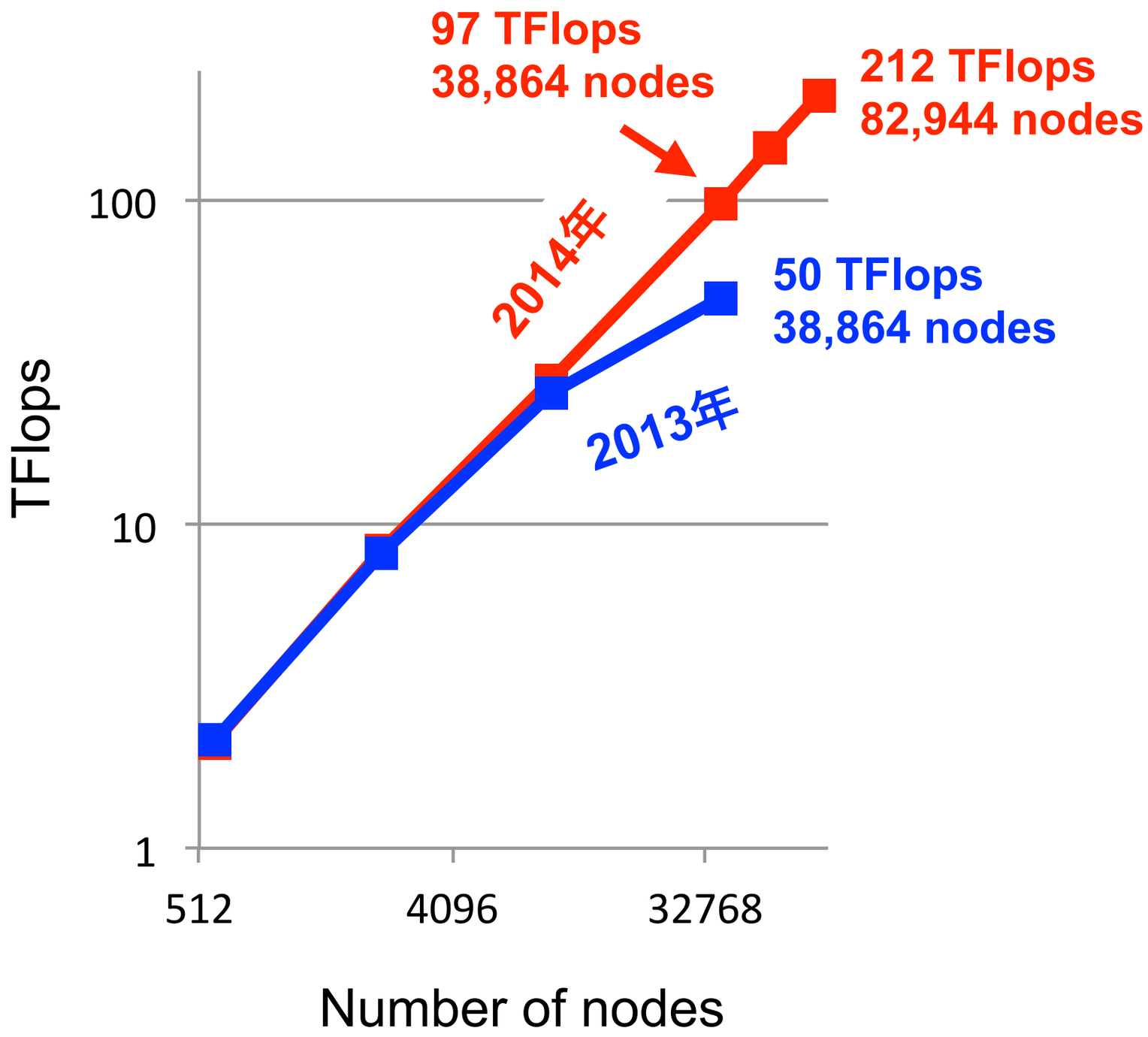
2次元分散配列の定義

FFTE関数の呼び出し

ひねり係数の乗算

XMPの転置関数

FFTの結果



38.864ノード時に,
1.94倍の性能向上

Class 1の結果は
82,944ノードで
206 TFlops

FFTの実装の考察

- 生産性について
  - グローバルビューにより，2次元配列を直接操作
  - FFTEを直接呼び出し
  - 分散配列用転置関数xmp_transpose()の利用

- 性能について
  - Class 1と同等

発表の流れ

- XMPについて（グローバルビューとローカルビューのプログラム例）
- HPC Challengeベンチマークについて
- XMPを用いたHPC Challengeベンチマークの実装と性能評価
- まとめ

まとめ

- 目的
  - HPCチャレンジベンチマークによる大規模環境におけるXMPの評価
  - XMPを用いた並列化とそのチューニングのノウハウの蓄積

- 達成したこと
  - 生産性
    - XMPが提供するグローバルビューもしくはローカルビューにより，簡易にベンチマークを実装可能
    - CやFortranのチューニング技法や既存ライブラリをそのまま利用可能
  - 性能
    - STREAM，RandomAccess，FFTについてはMPIとほぼ同じ
    - HPLはローカルチューニングの余地があるが，スケーラビリティは良い

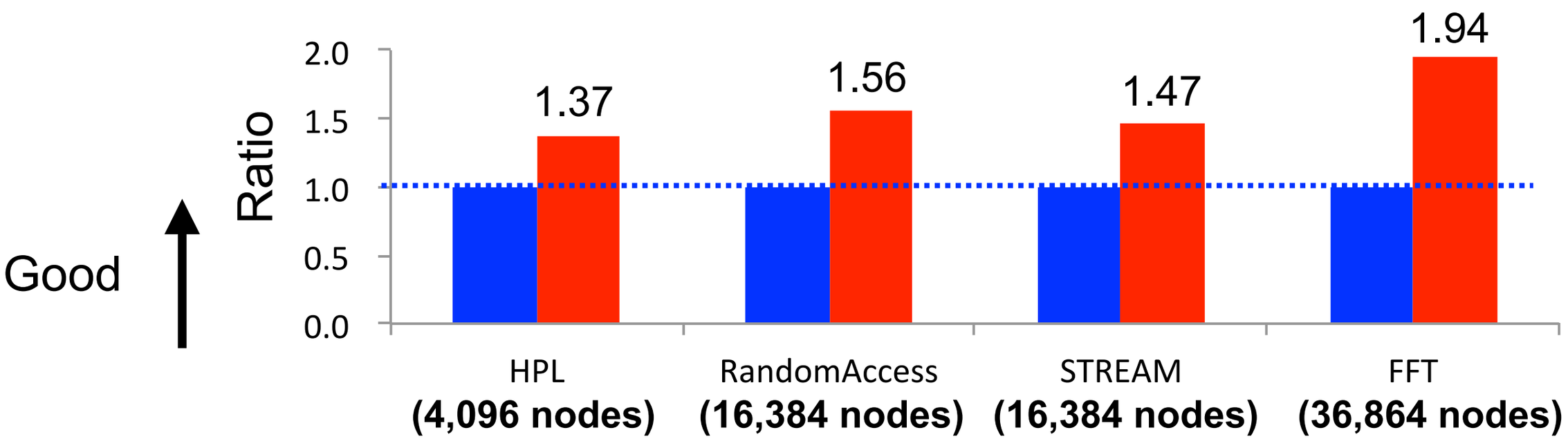
今後の課題

- 実アプリケーション and/or ミニアプリを用いた性能評価
- XMPの機能拡張
  - アクセラレータ対応（XcalableMP + OpenACC = XcalableACC[1]）
  - タスク並列機能

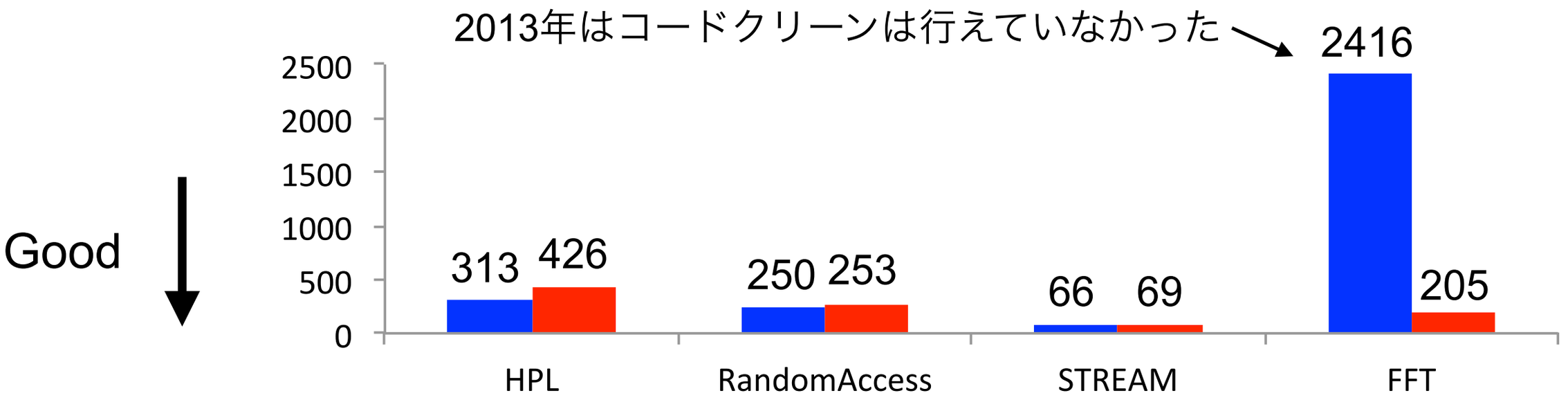
[1] Masahiro Nakao, et al. ``XcalableACC: Extension of XcalableMP PGAS Language using OpenACC for Accelerator Clusters,'' Workshop on accelerator programming using directives (WACCPD), New Orleans, LA, USA, Nov., 2014

2013年と2014年とのXMPの比較

- 同じノード数における性能向上率



- 行数

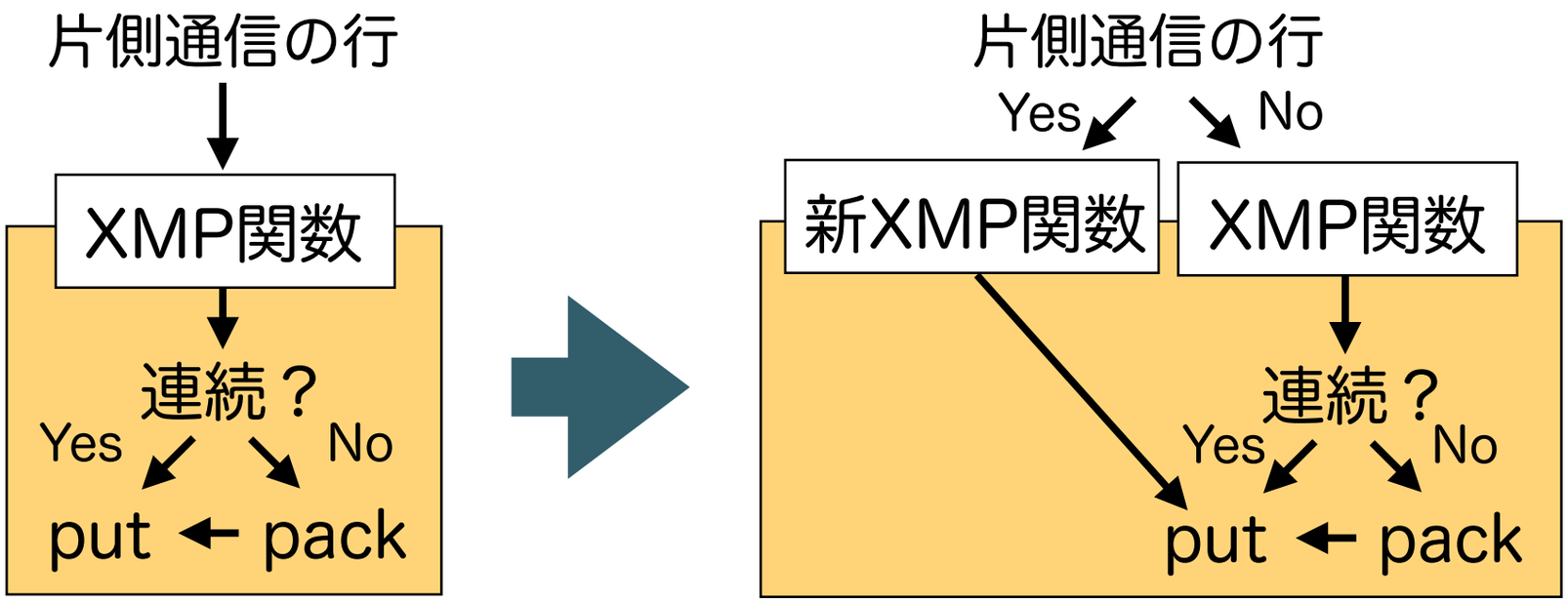


他のPGAS言語とMPIとの行数の比較

Lang.	HPL	RandomAccess	FFT	STREAM
XMP	313 (ver. 1) 426 (ver. 2)	253	205	69
PCJ	<N/A>	146	498	180
Coarray Fortran	786	409	450	63
Chapel	658	112	<N/A>	72
X10	708	143	236	60
MPI (hpcc-1.4)	8,800	787	1,904	329

片側通信機能

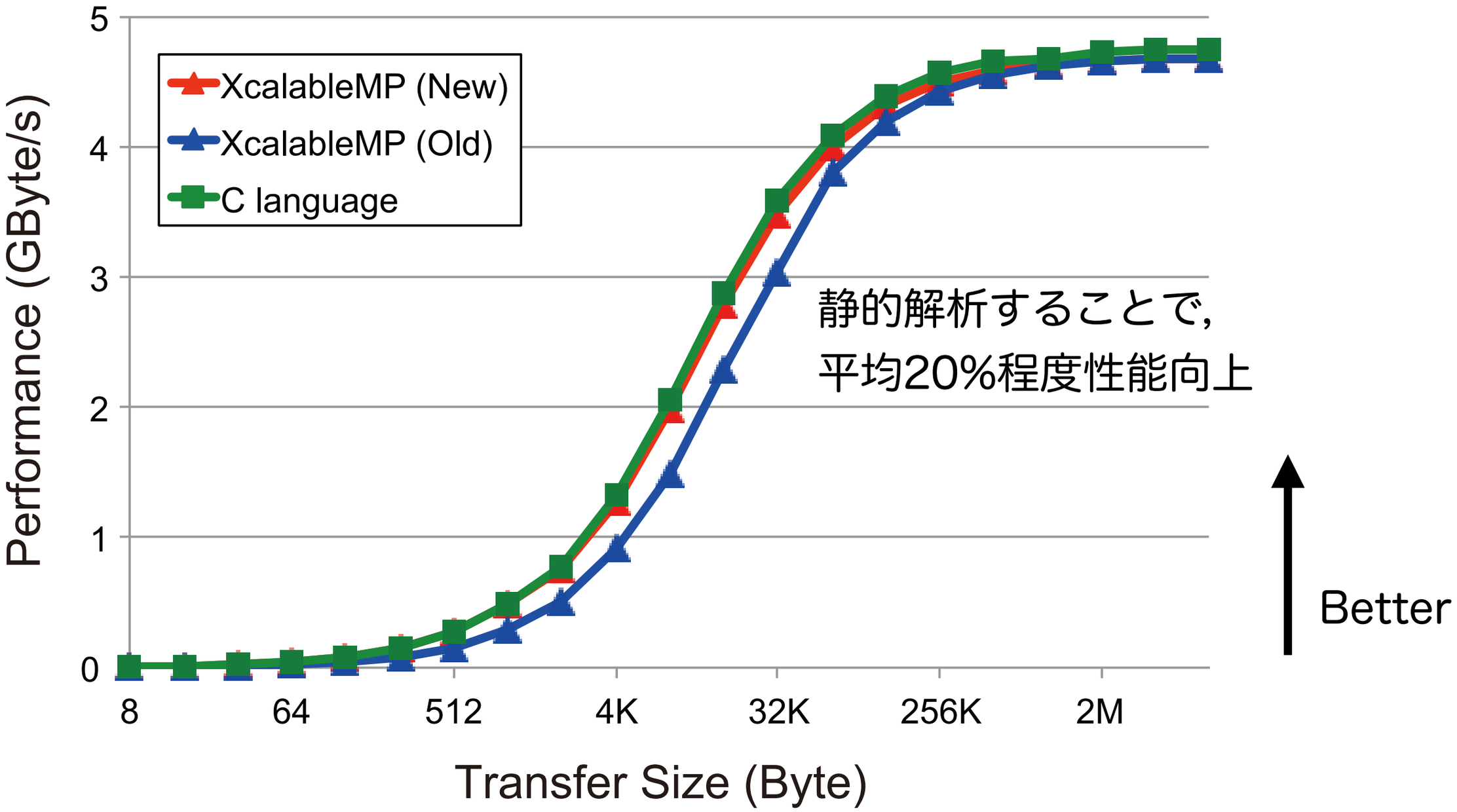
- 片側通信に関する機能（put/get/post-wait/lock-unlock/sync.）
- 基本的には，片側通信ライブラリGASNetを利用 (http://gasnet.lbl.gov)
- 「京」では，富士通RDMAを利用
- コードを静的に解析し，連続領域の転送の場合は，ランタイム内の
  いくつかの処理をショートカット



静的解析して，
連続領域の転送で
あれば新XMP関数を
呼ぶように置換する

片側通信機能

Putの性能比較：C言語から直接呼び出した場合とほぼ同じ性能



Results and Machine

Summary

Benchmark		# Nodes	Performance (/peak)	SLOC
HPL	Ver. 1	16,384	971 TFlops (46.3%)	313
	Ver. 2	4,096	423 TFlops (**80.7%**)	426
RandomAccess		16,384	254 GUPs	253
STREAM		**82,994**	3,583 TB/s (67.5%)	69
FFT		**82,944**	212 TFlops (2.0%)	205

The K computer: 82,944 nodes

http://www.aics.riken.jp/jp/outreach/photogallery.html

- SPARC64 VIIIfx Chip, 128 GFlops
- DDR3 SDRAM 16GB, 64GB/s
- Tofu Interconnect
  - 6D mesh/torus network
  - 5GB/s x 4links x 2

FFTの実装

```
!$xmp loop on tx(i)
do i=1,nx
  call zfft1d(b(1,i),ny,-1,cy)
end do

!$xmp loop on tx(i)
!$omp parallel do
do i=1,nx
  do j=1,ny
    b(j,i)=b(j,i)*w(j,i)
  end do
end do

call xmp_transpose(a,b,1)
```