XcalableMP

PCクラスターコンソーシアム
第2回 XcalableMP ワークショップ

並列プログラミング言語 XcalableMP の課題と展望

佐藤　三久
筑波大学・計算科学研究センター
理化学研究所・計算科学研究機構

並列プログラミング言語XcalableMP(XMP)　規格部会

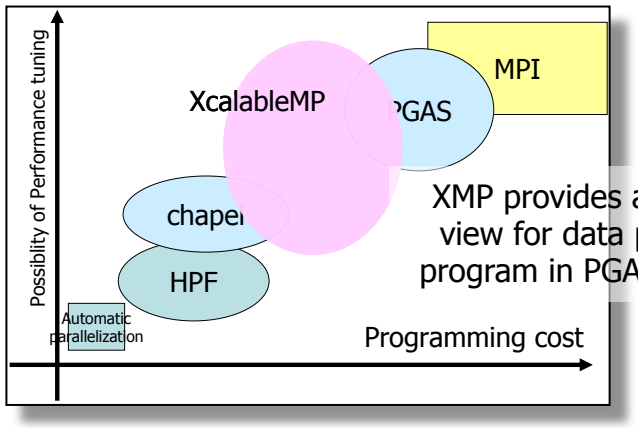
- 並列プログラミング言語XcalableMPとは
  - XcalableMP言語仕様検討委員会（大学、研究機関、メーカの有志がメンバ）で、仕様を検討した並列プログラミング言語
  - 分散メモリの並列システム（PCクラスタ）を対象、PGASモデル、HPFの経験を反映
  - ベース言語C/Fortranを指示文で拡張（既存プログラムの利用が容易）
  - 現在、仕様v1.0を決定、公開中　http://www.xcalablemp.org/
  - 「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」（代表：東大　石川裕、H23年度終了）において、筑波大を中心にレファレンス実装

- 部会の活動、計画概要
  - 部会長：佐藤三久、副部会長：岩下英俊　（富士通（株））、林 康晴（日本電気（株））
  - 趣旨：並列プログラミング言語XcalableMP(XMP)の仕様について、検討し、決定する。本部会では、アプリケーション開発者がさまざまな計算環境で並列プログラム言語を容易に利用することができるように、プログラミング言語に要求される機能を検討するとともに，計算機メーカ及びユーザ・コミュニティが合意できる標準的な言語仕様を定めることを目的とする。これにより、並列プログラミングの生産性を向上させるとともに、コミュニティからの支援により、さまざまなプラットフォームにおいて利用できる言語の普及を目指し活動を行う。
  - 1）並列プログラミング言語XcalableMPの言語仕様（v1.0以降）の検討・決定
  - 2）並列プログラミング言語XcalableMPの普及活動（講習会、ワークショップ、コンテストの開催等）
  - 3）PCクラスタ向け並列プログラミング言語の動向の調査

slide at SC13 PGAS BoF

XcalableMP(XMP) http://www.xcalablemp.org

- What's XcalableMP (XMP for short)?
  - A PGAS programming model and language for distributed memory , proposed by **XMP Spec WG**
  - XMP Spec WG is a special interest group to design and draft the specification of XcalableMP language. It is now organized under **PC Cluster Consortium**, Japan. Mainly active in Japan, but open for everybody.
- Project status (as of Nov. 2013)
  - XMP Spec **Version 1.2** is available at XMP site. new features: mixed OpenMP and OpenACC , libraries for collective communications.
  - Reference implementation by U. Tsukuba and Riken AICS: **Version 0.7 (C and Fortran90)** is available for PC clusters, Cray XT and K computer. Source-to-Source compiler to code with the runtime on top of MPI and GasNet.



- Language Features
  - Directive-based language extensions for Fortran and C for PGAS model
  - Global view programming with global-view distributed data structures for data parallelism
    - SPMD execution model as MPI
    - pragmas for data distribution of global array.
    - Work mapping constructs to map works and iteration with affinity to data explicitly.
    - Rich communication and sync directives such as "gmove" and "shadow".
    - Many concepts are inherited from HPF
  - Co-array feature of CAF is adopted as a part of the language spec for local view programming (also defined in C).

Code example

```
int array[YMAX][XMAX];

#pragma xmp nodes p(4)
#pragma xmp template t(YMAX)
#pragma xmp distribute t(block) on p
#pragma xmp align array[i][*] to t(i)

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

#pragma xmp loop on t(i)  reduction(+:res)
  for(i = 0; i < 10; i++)
    for(j = 0; j < 10; j++){
      array[i][j] = func(i, j);
      res += array[i][j];
    }
}
```

data distribution

add to the serial code : incremental parallelization

work sharing and data synchronization

XcalableMP Code Example (Fortran)

```
program xmp_example
Integer array(XMAX,MAX)
!$XMP nodes p(4)
!$XMP template t(YMAX)
!$XMP distribute t(block) onto p
!$XMP align array(*,j) with t(j)

  integer :: i, j, res
  res = 0

!$XMP loop on t(j)  reduction(+:res)
  do j = 1, 10
    do I = 1, 10
       array(I,j) = func(i, j)
       res += array(I,j)
  enddo
 enddo
```

data distribution

add to the serial code : incremental parallelization

work mapping and data synchronization

XcalableMP Code Example (C)

```
int array[YMAX][XMAX];

#pragma xmp nodes p(4)
#pragma xmp template t(YMAX)
#pragma xmp distribute t(block) on p
#pragma xmp align array[i][*] with t(i)

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

#pragma xmp loop on t(i)  reduction(+:res)
  for(i = 0; i < 10; i++)
    for(j = 0; j < 10; j++){
      array[i][j] = func(i, j);
      res += array[i][j];
    }
}
```

data distribution

add to the serial code : incremental parallelization

work mapping and data synchronization

MPIで、いいのか？

いつまで、MPIでプログラミングしなくてはならないのか？

XcalableMP as evolutional approach

- 現状のコードからの移行を重視
  - 既存の言語C/Fortranに指示文を加えて並列プログラミング
- 過去の経験を取り入れる
  - 我が国のHPFの歴史
- コミュニティで言語を策定: e-science プロジェクトで組織。
  - その後、PCクラスタコンソーシアムの部会として活動
- PGAS モデルをベースに言語を設計
- 難しいところはCoarray（from CAF）で。
- 演算加速機構への対応　XMP-dev　等、リサーチビークルとしても利用

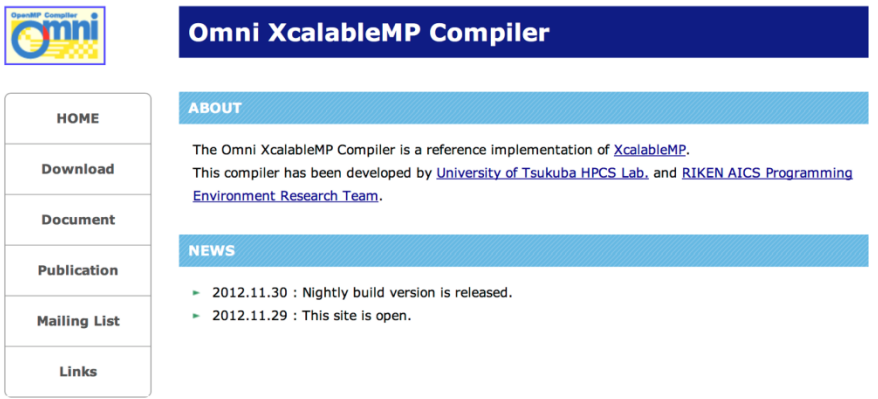
仕様書 v 1.2 での追加内容

- マルチコア対応
  - 現状
    - ほとんどのクラスタがいまや、マルチコアノード（SMPノード）
    - 小規模では格コアにMPIを走らせるflat MPIでいいが、大規模ではMPI数を減らすためにOpenMPとのハイブリッドになっている。
      - ハイブリッドにすると（時には）性能向上も。メモリ節約も。
    - しかし、ハイブリッドはプログラミングのコストが高い。
  - XMPとOpenMPをhybridに記述、その規則を定義

- collective communicationのライブラリを定義
  - sin,cosなどの算術関数の要素並列
  - transposeなどの形状変換関数

リファレンス実装について

- Omni Compiler プロジェクト
  - XcalableMPのリファレンス実装
  - 元々、OpenMPコンパイラを開発するために作ったもの
- Omni XMP Compiler v0.6.1をリリース（2013年3月）
  - Coarray機能のストライド通信の対応
  - 京コンピュータにおけるCoarray機能のサポート
- 2013年11月にOmni XMP Compiler v0.7をリリース
  - OpenMP指示文やOpenACC指示文との混在利用
  - 京コンピュータにおけるハードウェア サポートを利用した実装
  - XMP/Fortranの片側通信機能の サポートは、未
- リポジトリーをgitに移行
- 2014年　v0.9?



HPC Class2 Award

XcalableMP

- SC13@Denverでブース展示（11/18 - 21）

- 筑波大・理研AICS チームでHPC Challenge Class 2に応募

  ⇒ 日本初のawardを獲得！

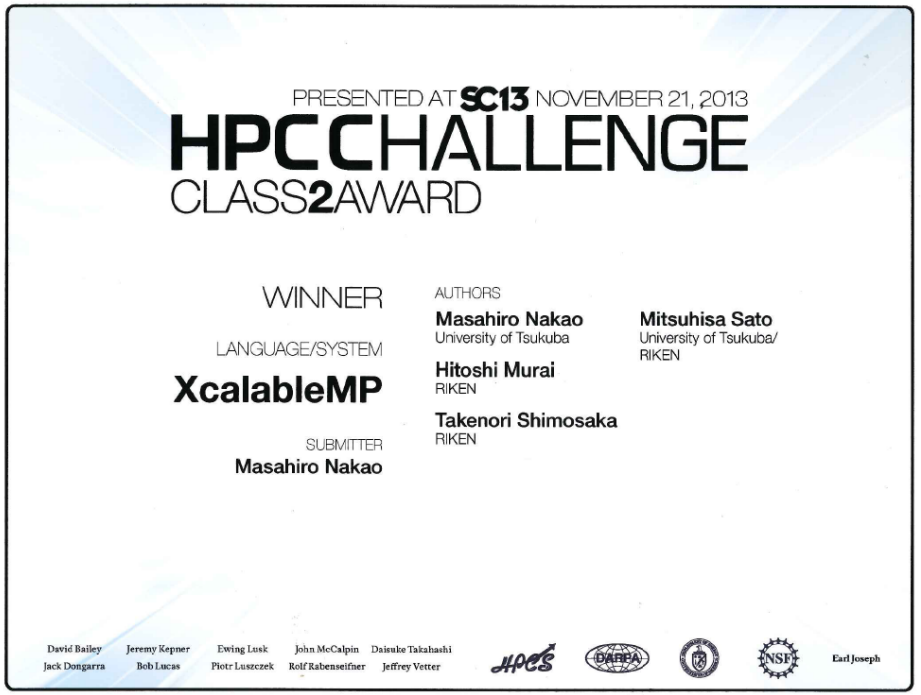
  - HPCシステムの性能を評価するためのベンチマーク集
    - HPL, FFT, RandomAccess, Stream
  - Class 1は性能のみを評価
  - Class 2は生産性

  （実装のエレガントさ）と性能の2つを統合的に評価
    - XMPでHPC Challenge

    ベンチマークを実装し，

    京コンピュータ上で計測
    - SC13のBoFで発表
    - SC13後にXMPのHPC Challenge

    ベンチマークを公開予定



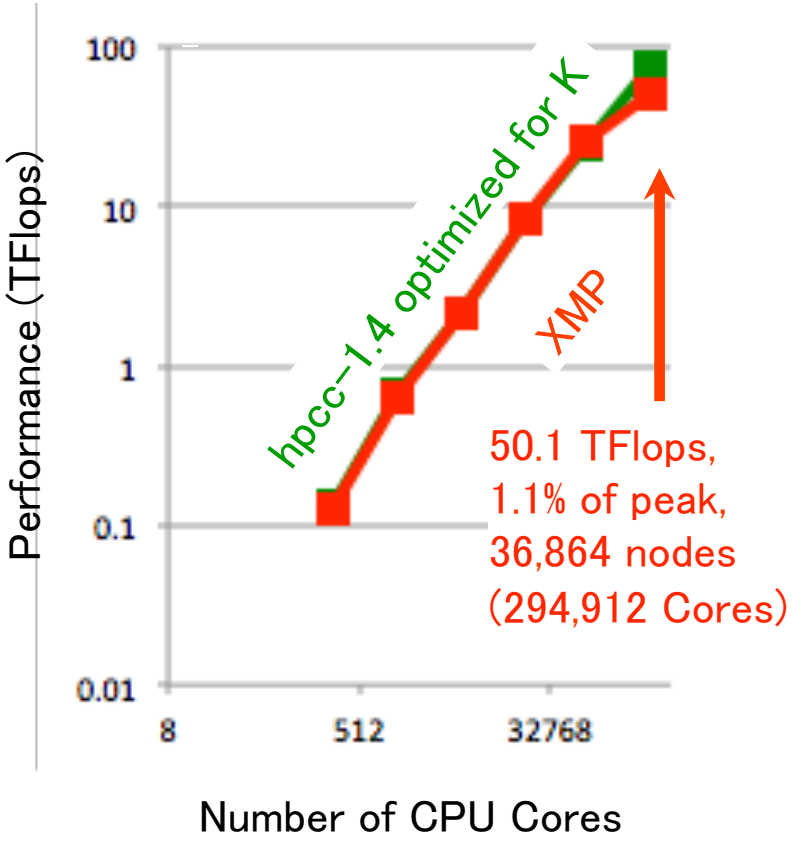
Summary

Benchmark	# Nodes	Performance	SLOC
HPL	16,384	933.8 TFlops (44.5% of peak)	306
RandomAccess	16,384	162.6 GUPs	250
FFT	36,864	50.1 TFlops (1.1% of peak)	239 + 283 + 1892
STREAM	16,384	481.8 TB/s	66
HIMENO	**82,944**	1.3 PFlops (12.7% of peak)	137

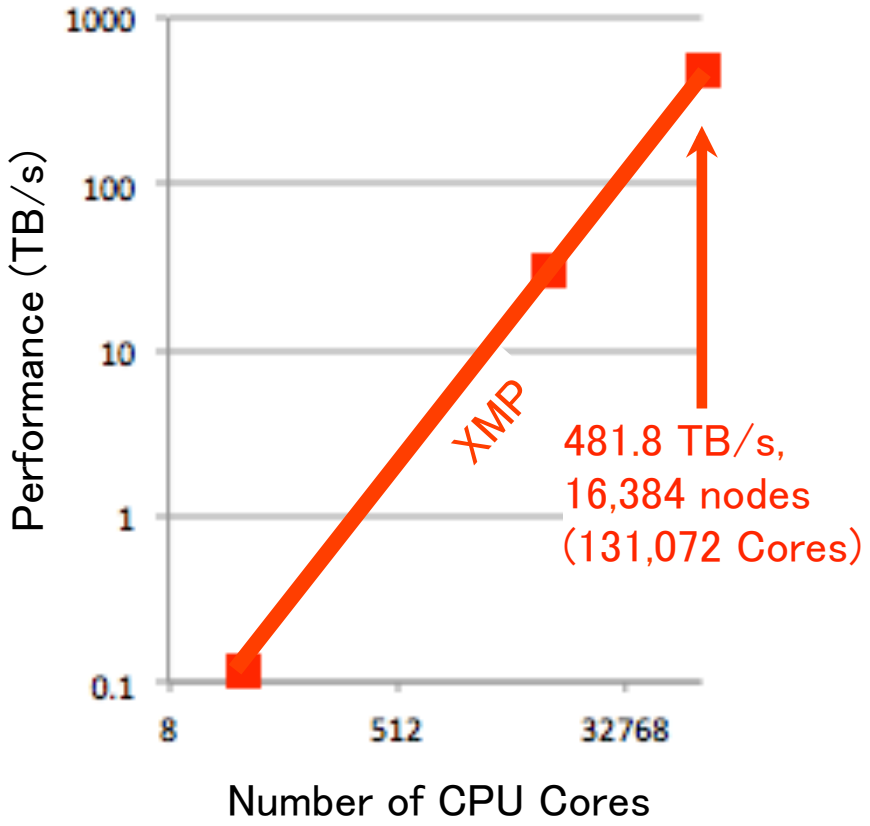
Full compute nodes

Results(2/3)

- FFT (1 process/node with 8 threads)



- STREAM (1 process/node with 8 threads)



現在の（研究）活動

- 開発は主にAICSで。
- NEC　SXやIBM BG/Qなどへの移植　（AICSの活動の一部）
  - そのうちに、調達の「仕様書」にいれてもらえるように
- XMPを使った並列プログラムのチュートリアルなどの人材育成活動　(AICS)
- GPUのサポート、XMP-devはあるが、OpenACCとの混在で再検討（朴CREST@筑波大、AICS)
- Total 社との共同研究
- 研究
  - Xeon Phiでの性能評価（筑波大）
  - XMP向けのshared memoryモデル、one-sided通信（筑波大）
  - 有限要素法などの不規則メッシュのための拡張（AICS）
  - XMP上の動的なタスク生成（AICS)

2013.12.27部会の議題

XcalableMP

- HPCC Class2 Awardの報告
- 開発進捗・報告
- これからの進め方の議論
  - 一応、議論できるところはした感がある。
  - これからは、普及のための活動
    - 神戸では、講習会をやっているので、これを拡大
  - 追加する機能はあるのか？
    - 不規則問題は、議論したが不調
    - タスク並列？（現在の仕様ではなくて、もっとダイナミックのもの）
  - エクサに向けて、なにかやる？
  - XcodeMLのようなものの標準化？

XMPのこれからの課題

- これからのプラットフォームは大規模なメニーコアクラスタ

- メニーコアへの対応
  - 局所性を高めるプログラミング
- 大規模化への対応
  - （1） 効率的な通信レイヤーの利用
    - RDMA one-sided （ノード外）, 共有メモリ（ノード内）
  - （2） 動的な並列性への対応
    - 同期
    - 負荷分散

- MPIとの混在
- 最適化支援

XcalableMP on Xeon Phi

- 現在、筑波大のHPCS研究室において、XcalableMP処理系のXeon Phi (KNC)への設計・実装を進めている。

- XcalableMPの利点
  - メニーコアでは、ノードのコアが多いため(Xeon Phiの場合は60)、MPI +OpenMPのHybridプログラミングが必須になり、２重のプログラミングのコストが高くなる。XcalalableMPにより、１つのプログラミングモデルで、カバーできる。
  - ランタイムを効率的にデータにアクセス・通信できるように設計することにより、単一のモデルでスケーラブルな実装が可能。
  - 各コアでの局所性を高めるプログラミングモデルの提供

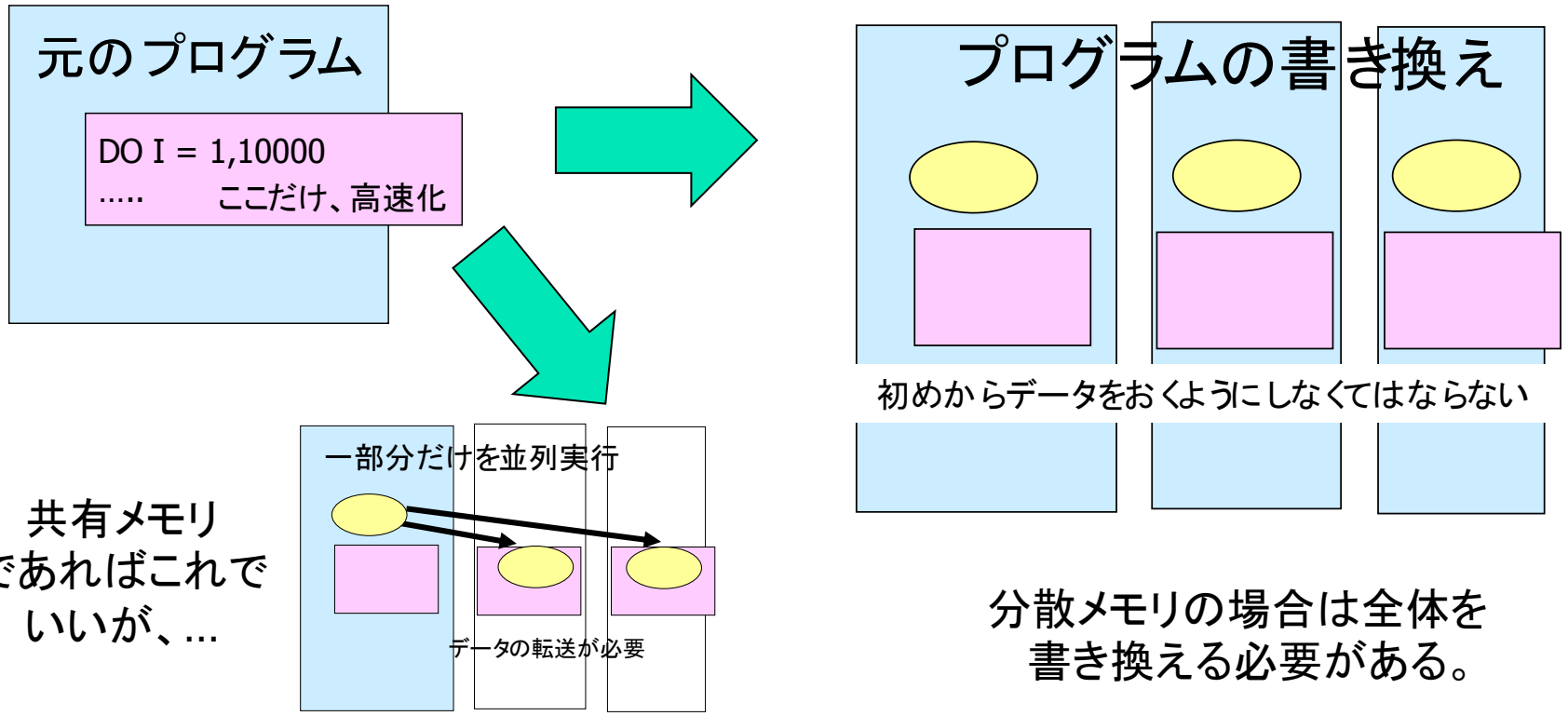
- 科学技術計算で典型的な計算パターンであるステンシル計算を対象に、評価実験
  - CCGrid2014で、発表 （by 池井さん@筑波大＆Intel）

並列プログラミングは何が難しいか

- All about memory! キーポイントは、メモリモデル
- 第一の波： 逐次プログラム ⇒ ベクトルプロセッサ
  - 並列性記述、データのレイアウトを変える
- 第2の波： ベクトルプロセッサ ⇒ 分散メモリ, MPP
  - MPI、プログラムの構造を変えることが必要
- 第3の波： 分散メモリ

  ＋ 加速機構オフロード or メニーコア
  - 必要なデータをオフロード
  - コアあたりの局所性を高める

分散メモリと並列プログラミング



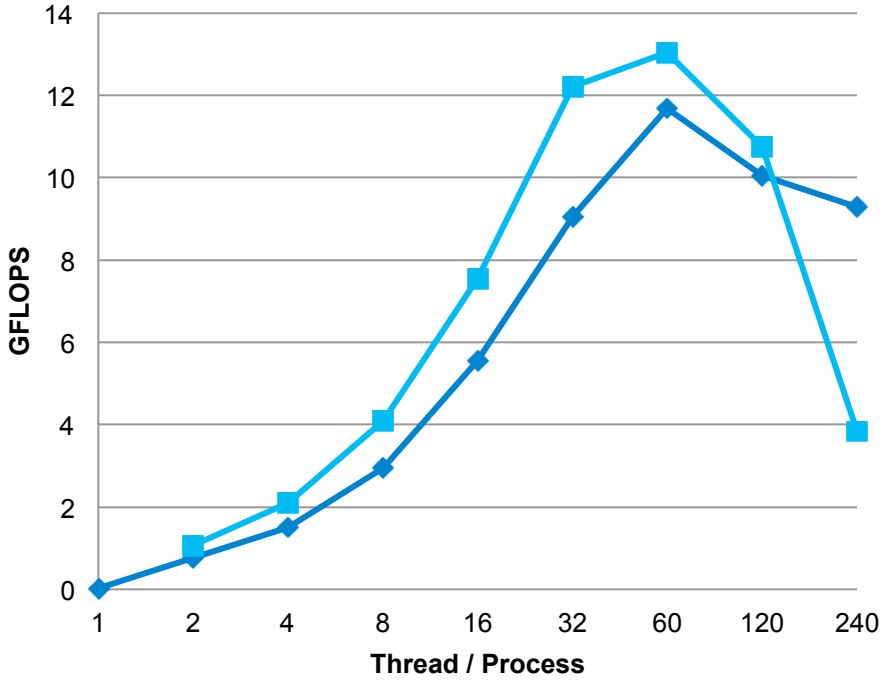
共有メモリ
であればこれで
いいが、…

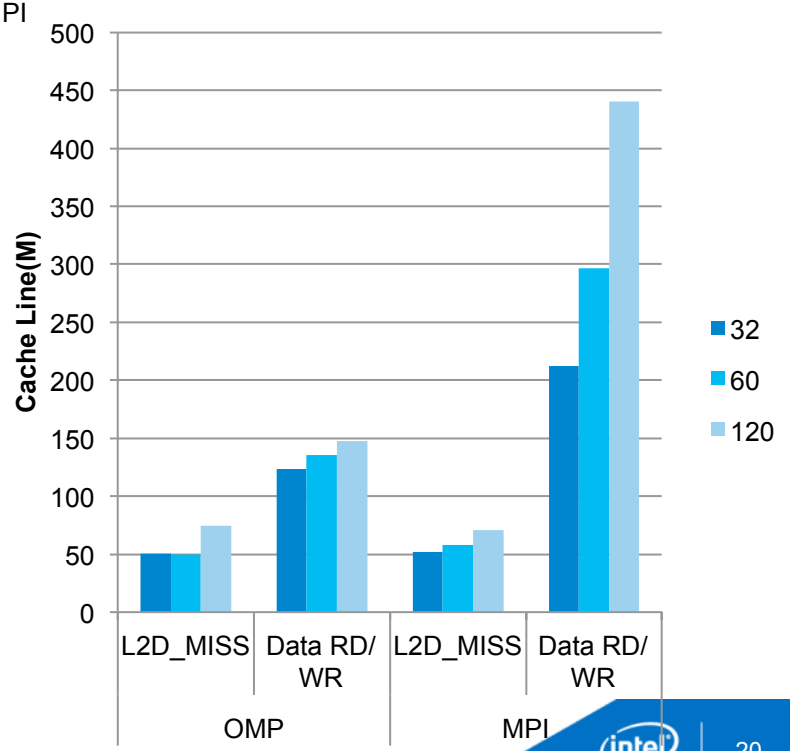
分散メモリの場合は全体を
書き換える必要がある。

OpenMP Laplace Benchmark

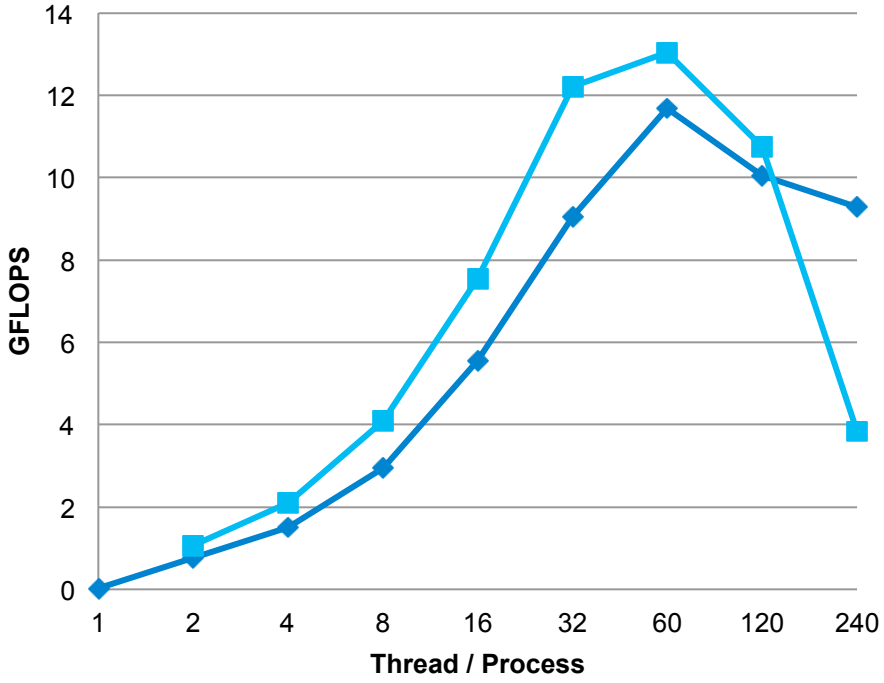
```
 48
 49 #pragma omp parallel private(k,x,y)
 50 {
 51     for(k = 0; k < NITER; k++){
 52         /* old <- new */
 53 #pragma omp for
 54         for(x = 1; x <= XSIZE; x++)
 55           for(y = 1; y <= YSIZE; y++)
 56             uu[x][y] = u[x][y];
 57         /* update */
 58 #pragma omp for
 59         for(x = 1; x <= XSIZE; x++)
 60           for(y = 1; y <= YSIZE; y++)
 61             u[x][y] = (uu[x-1][y] + uu[x+1][y]
+ uu[x][y-1] + uu[x][y+1])/4.0;
 62     }
 63 }
```

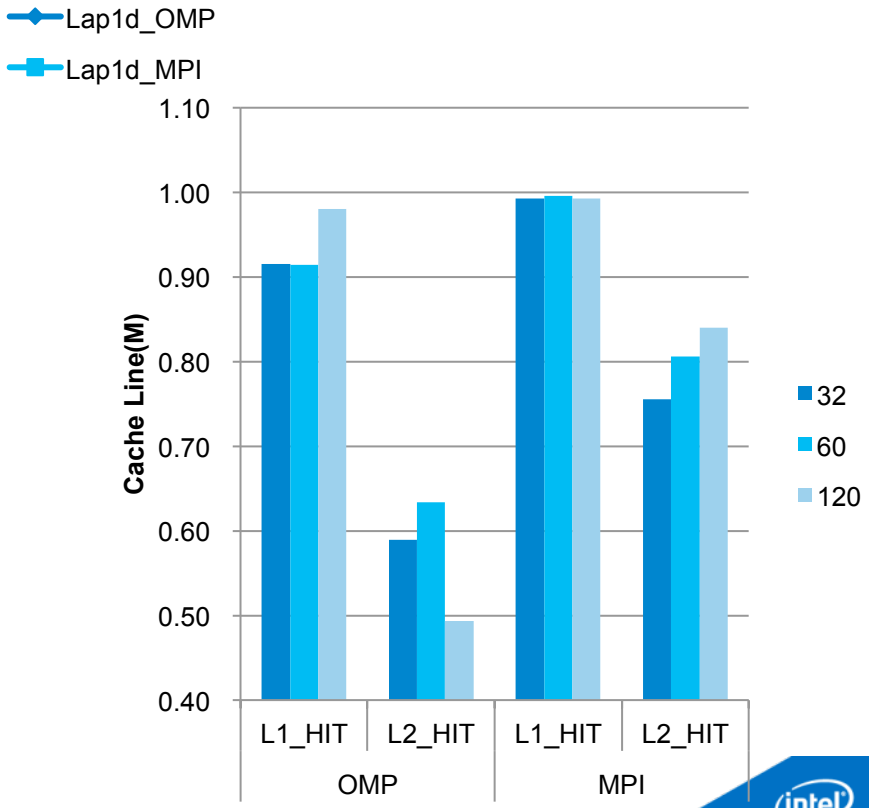
Laplace Result OpenMP vs MPI on Phi





Laplace Result OpenMP vs MPI on Phi





Xeon Phi向けXcalableMPの実装

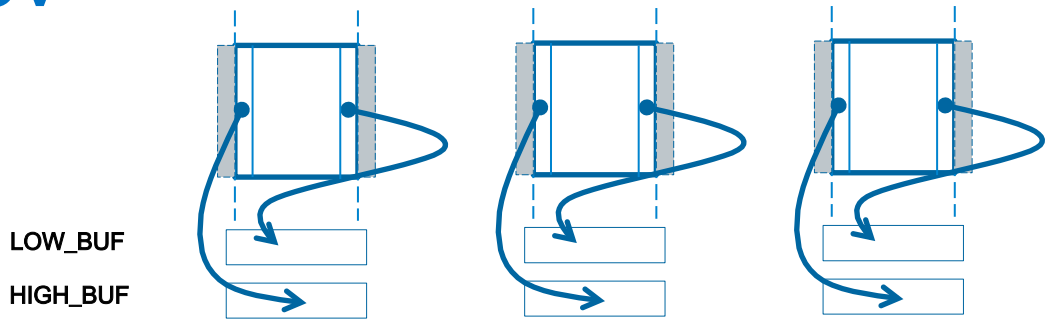
- Xeon Phiのノード内での実装の選択肢
  - MPIを使う（現状、何もしない）
  - mmapによる共有メモリを確保を行い、これを通して通信（←今回）
  - プログラム変換によるスレッドモデルへのトランスレート（UPCでやっている。現在、実装計画中）
  - PVASによる実装（理研AICS グループと共同研究）

XMP version of Laplace Benchmark

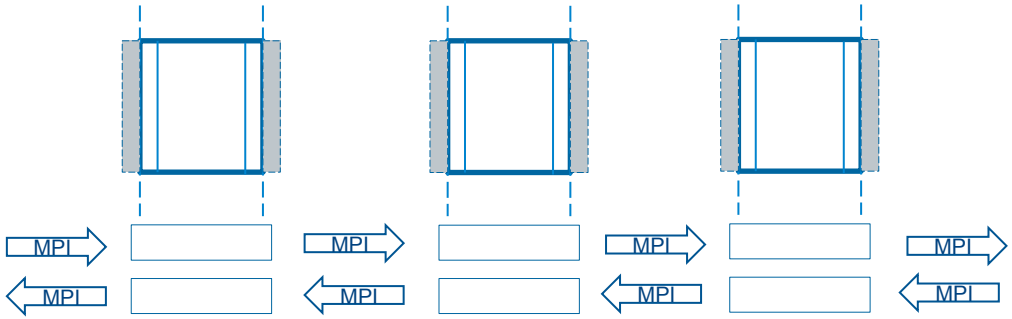
```
16 double u[XSIZE][YSIZE], uu[XSIZE][YSIZE];

19 #pragma xmp nodes p(*)
20 #pragma xmp template t(0:(10000)-1)
21 #pragma xmp distribute t(block) onto p
22 #pragma xmp align u[j][*] with t(j)
23 #pragma xmp align uu[j][*] with t(j)
24 #pragma xmp shadow uu[1][0]

...

86     /* old <- new */
87 #pragma xmp loop (x) on t(x)
88     for(x = 1; x < XSIZE-1; x++)
89       for(y = 1; y < YSIZE-1; y++)
90         uu[x][y] = u[x][y];
91
92 #pragma xmp reflect (uu)
93
94     /* update */
95 #pragma xmp loop (x) on t(x)
96     for(x = 1; x < XSIZE-1; x++)
97       for(y = 1; y < YSIZE-1; y++)
98         u[x][y] = (uu[x-1][y] + uu[x+1][y] + uu[x][y-1] + uu[x][y+1])/4.0;
```

Ghost region update using MPI sendrecv

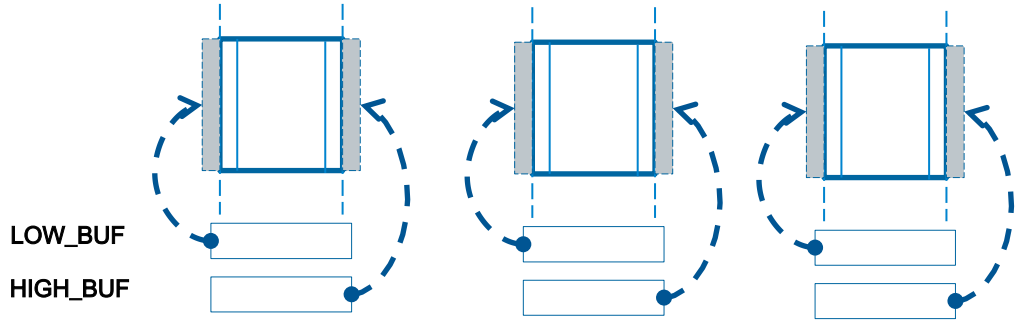
(1) Pack



(2) Exchange

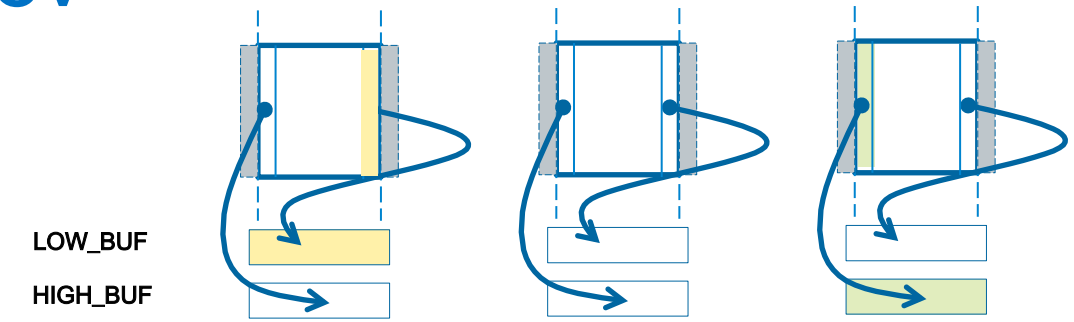


(3) Unpack

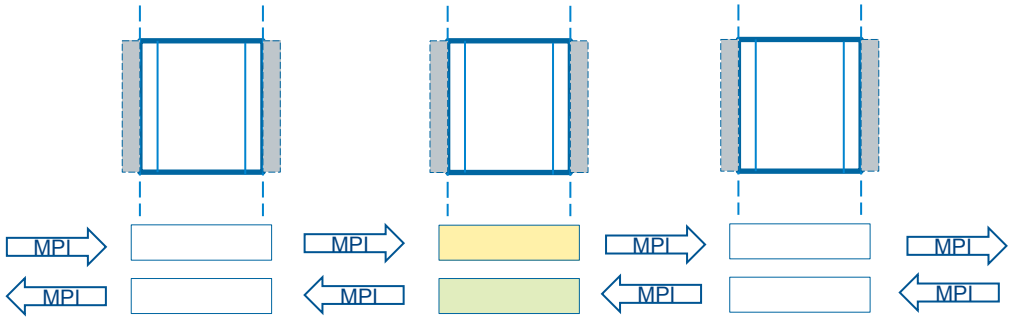


Ghost region update using MPI sendrecv

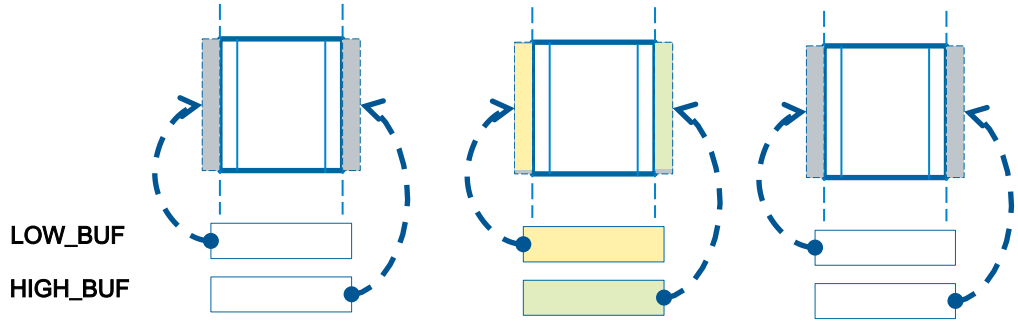
(1) Pack



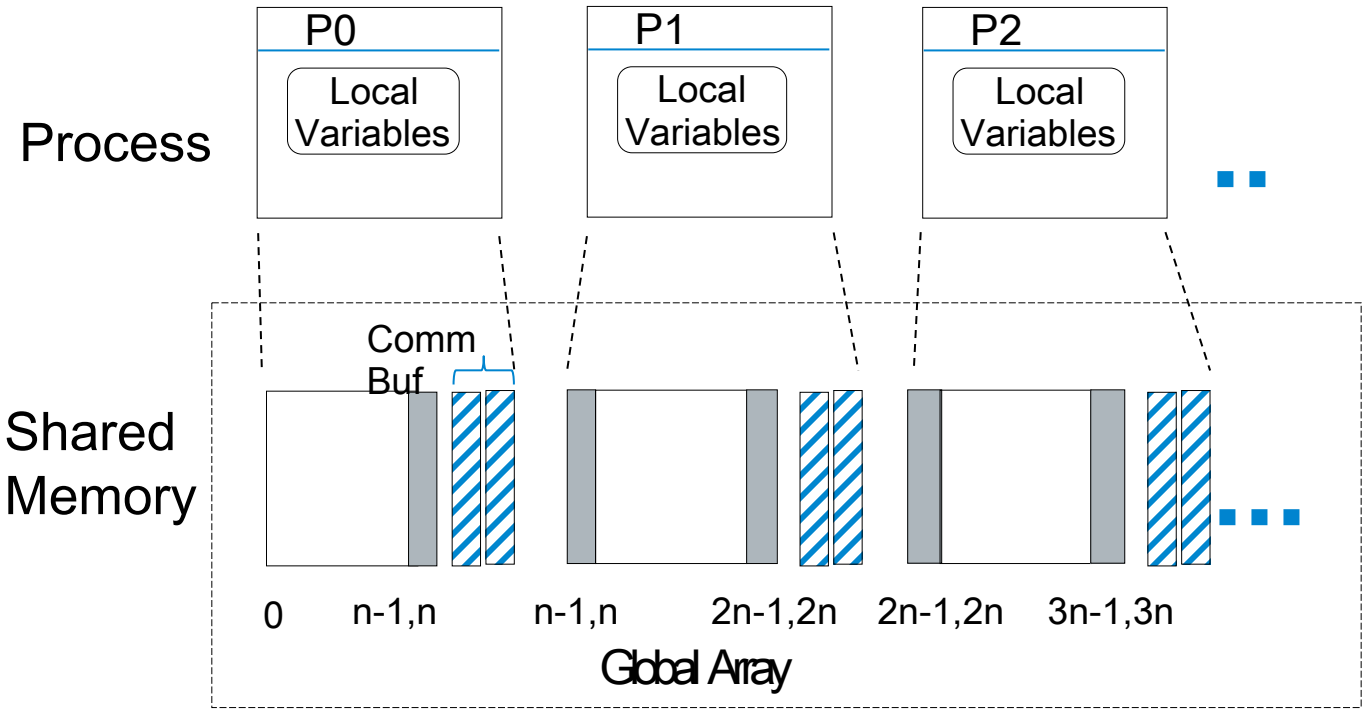
(2) Exchange



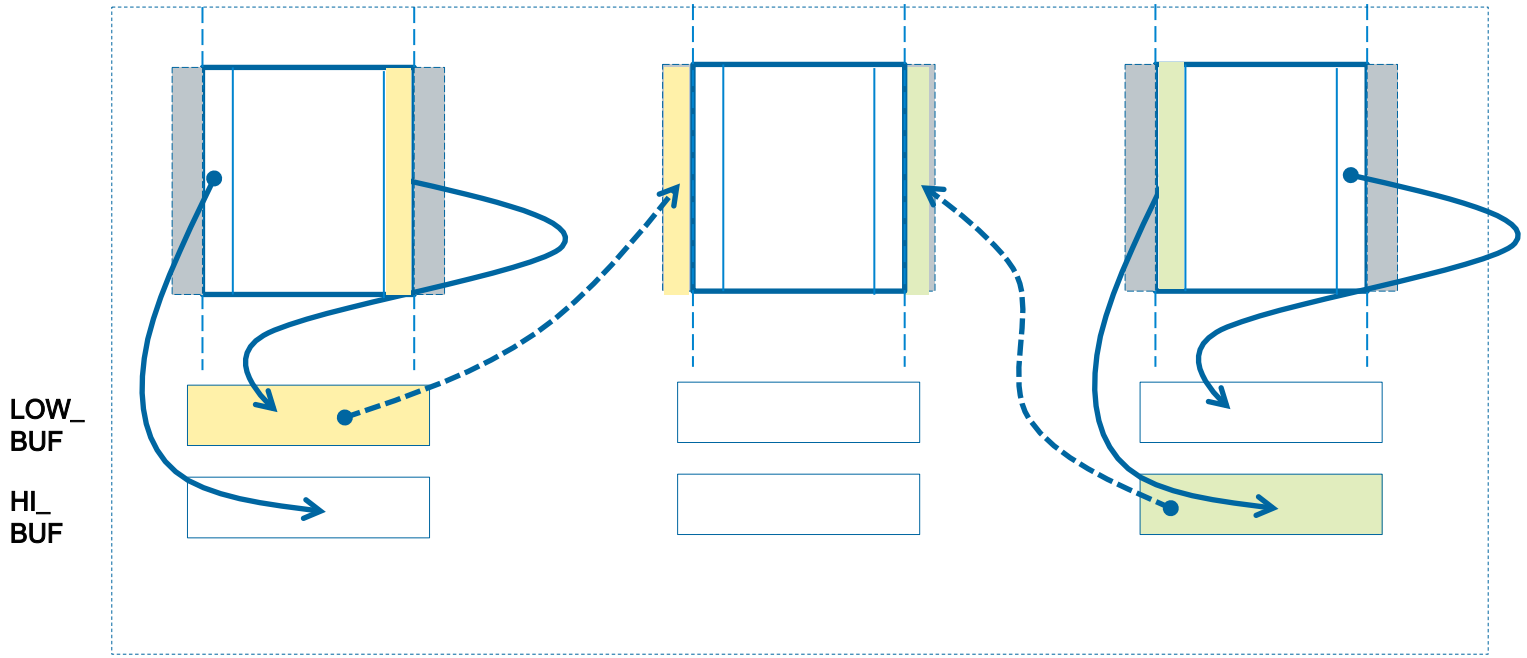
(3) Unpack



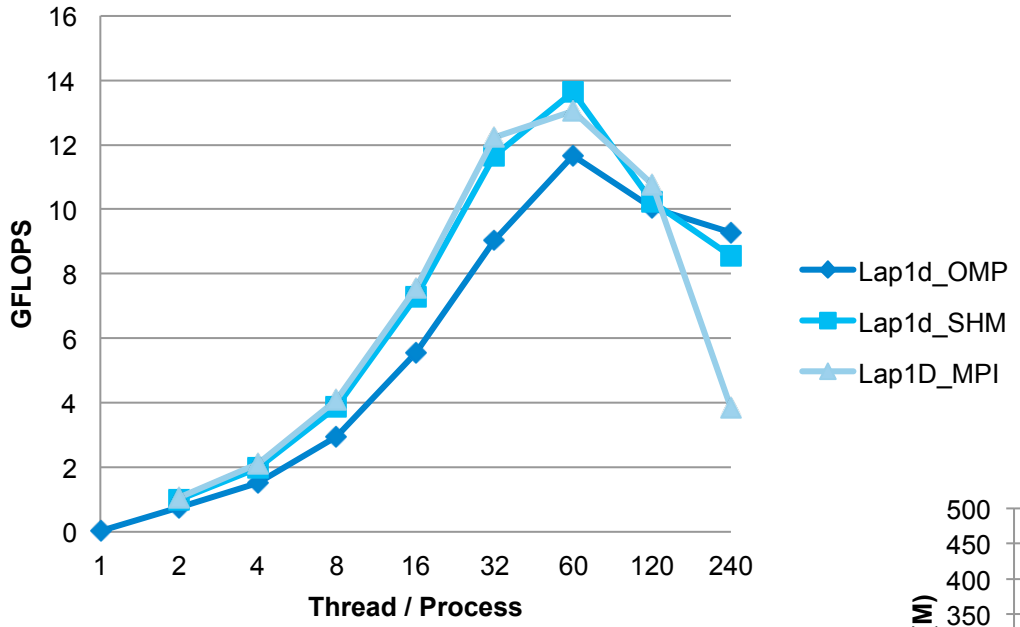
XMP Global Array Allocation

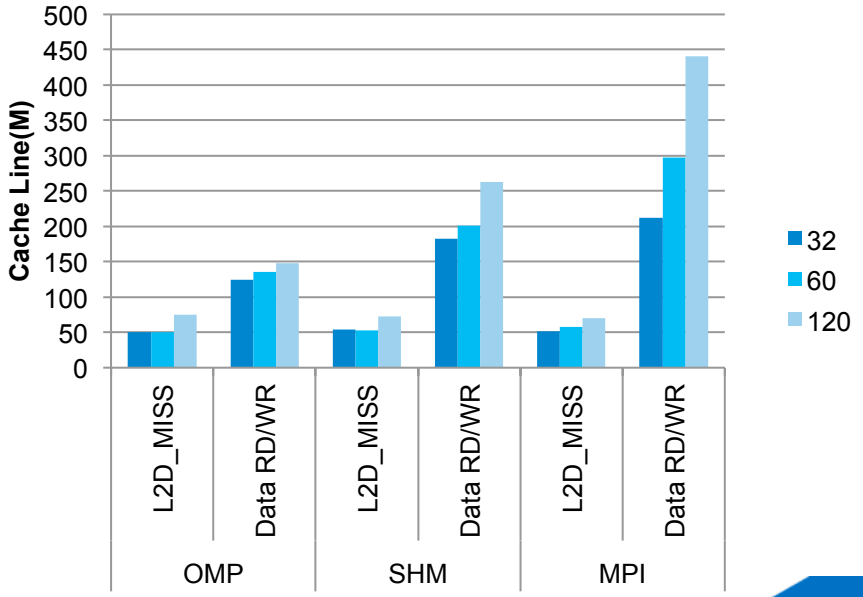


Ghost region update on SHM



Laplace 1D Benchmark Results

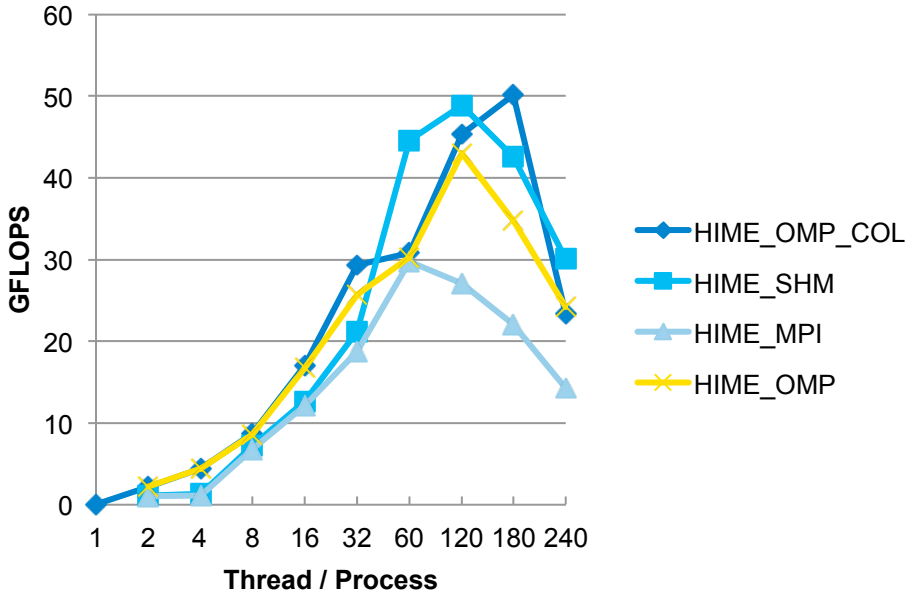


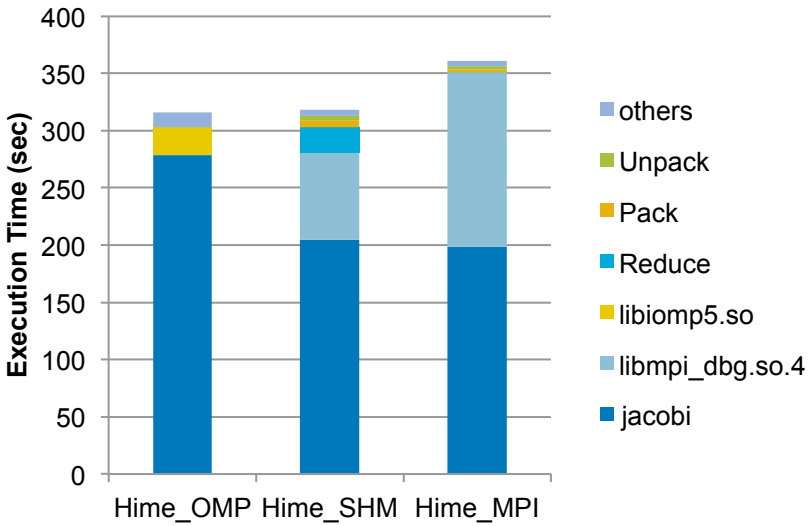


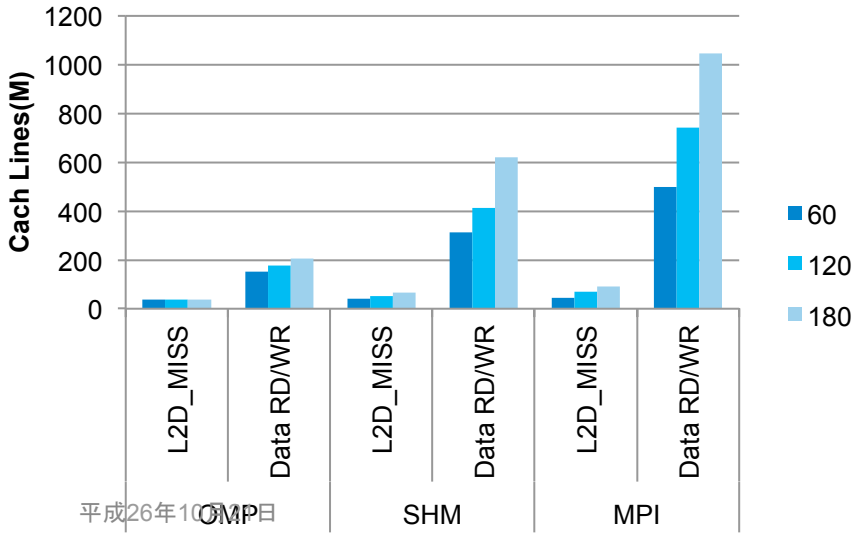
2D Distribution Himeno Benchmark

```
82 static float  p[257][257][513];
83 static float  a[4][257][257][513], b[3][257][257][513],
84              c[3][257][257][513];
85 static float  bnd[257][257][513];
86 static float  wrk1[257][257][513], wrk2[257][257][513];
87
88 #pragma xmp nodes n(6,10)
89 #pragma xmp template t(0:512,0:256,0:256)
90 #pragma xmp distribute t(*,block, block) onto n
91 #pragma xmp align [i][j][*] with t(*, j, i) :: p,bnd,wrk1,wrk2
92 #pragma xmp align [*][i][j][*] with t(*, j, i) :: a,b,c
93 #pragma xmp shadow p[1][1][0]
...
231 #pragma xmp reflect (p)
232 #pragma xmp loop(i,j) on t(*,j,i) reduction (+:gosa)
233     for(i=1 ; i<imax-1 ; i++)
234       for(j=1 ; j<jmax-1 ; j++)
235         for(k=1 ; k<kmax-1 ; k++){
236           s0 = a[0][i][j][k] * p[i+1][j  ][k  ]
237              + a[1][i][j][k] * p[i  ][j+1][k  ]
238              + a[2][i][j][k] * p[i  ][j  ][k+1]
239
             + b[0][i][j][k] * ...
```

Himeno Benchmark Results







コメント・これからの計画

XcalableMP

- Our overall performance advantage against OpenMP is 16.8 % for Lap1D and 13.8 % for Hime2D.

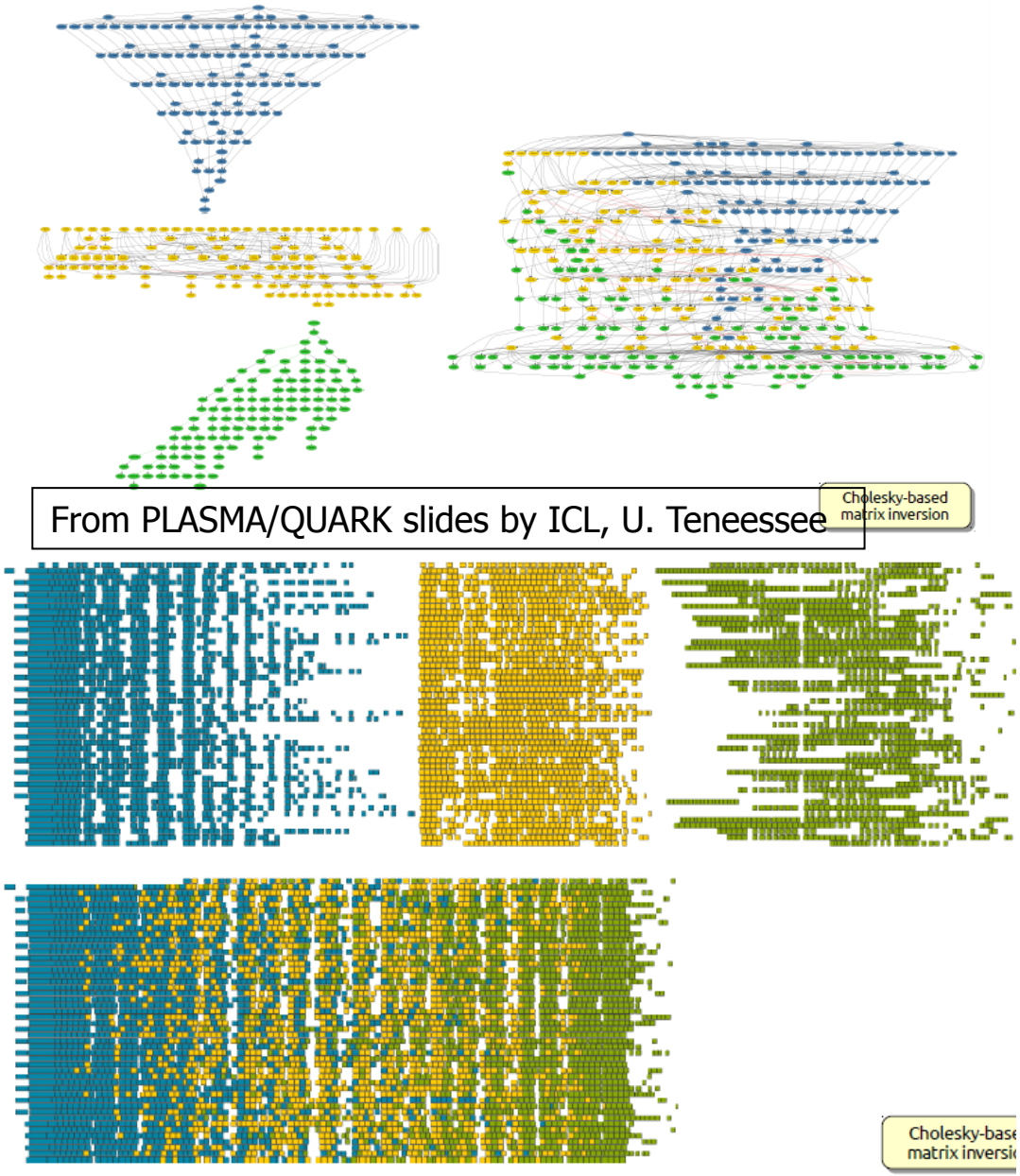
  ① **Blocking Effect of Global Arrays**: By dividing global arrays into sub-arrays detached each others, we could reduce 22.2 % and 27.3 % of CPU calculation time on Laplace and Himeno respectively.

  ② **Efficient Ghost Region Exchange**: By changing MPI sendrecvs to direct memory copy on SHM, we could reduce memory traffics about 32.3% to 44.6% and get the maximum performance gain 62.9 %.

- KNCは、なかなか謎のところが多いので、KNLでの性能を期待。
- 現在、ノード間も含めたXcalableMPの性能評価中
- Xeon Phi向けの新しいRuntimeの設計・リリース
  - ベクトル化支援も含めて

動的な並列性への対応

- 現在のXMPのプログラミングモデル（データ並列）は、同期が「強すぎる」。
- 柔軟な同期（バリア）
  - データフロー的な記述？
- 負荷分散のサポート
  - タスクの定義とタスクマイグレーション



最適化支援

- XMPの並列性の記述の情報を使って、ベクトル化やブロッキングなどのコード変換が可能か
  - 現在のXMPの仕様では、ローカルには逐次実行することが前提となっている。
  - また、ローカルに並列化する場合にはOpenMPを書く。

```
#pragma xmp loop (i) on ...
for( ... i ...){
        x +=
        t = ...
        A(i) = t +1;
}
```

- 逆に、ローカルな最適化をしようとすると、グローバルビューはやりにくい
  - Local_aliasを用いる？

MPIとの混在

- すでにMPIのプログラムがある中で、XMPをどのようにつかっていくのか。
- ライブラリは
  考えたが,…

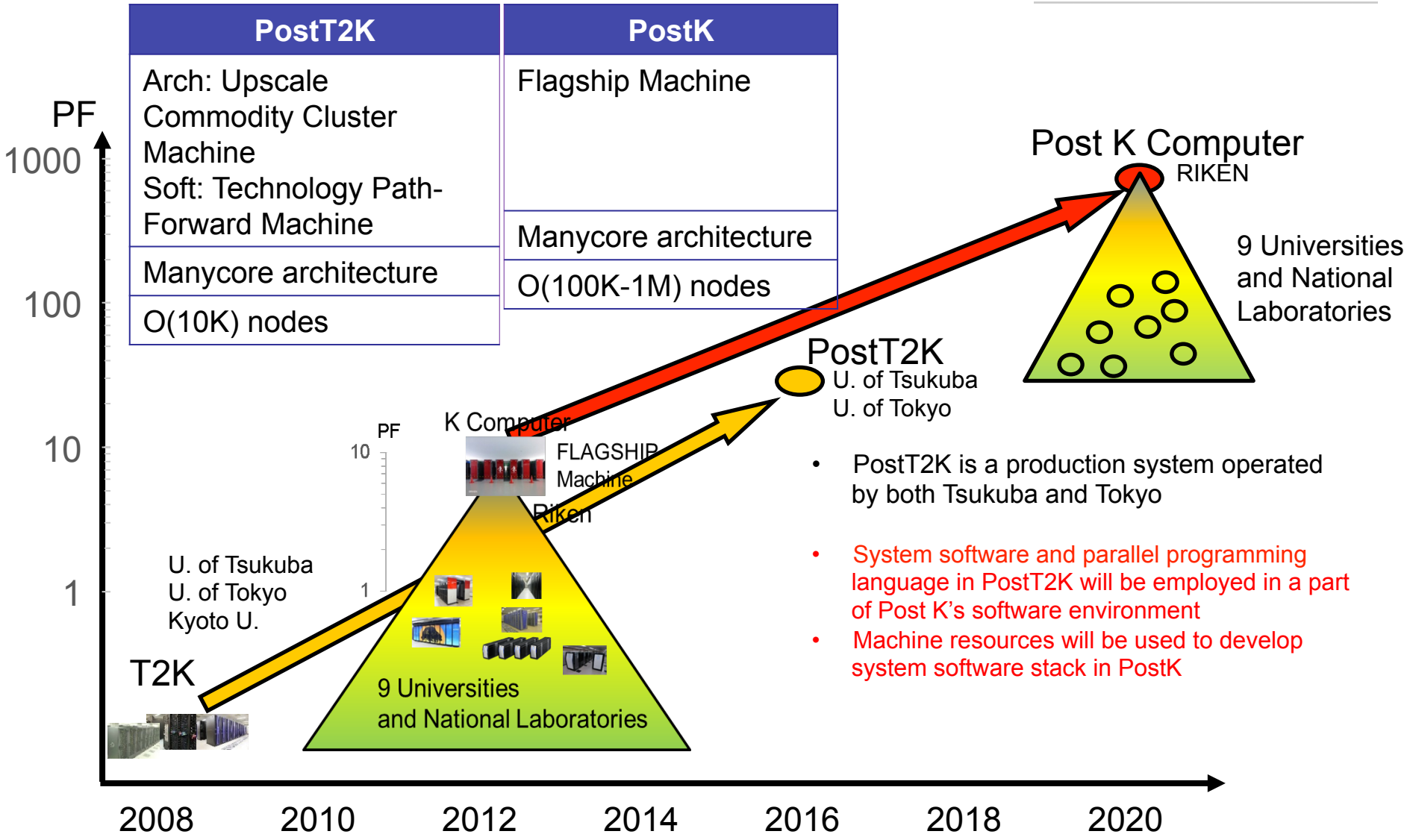
並列ライブラリインタフェース

- すべてをXMPで書くことは現実的ではない。他のプログラミングモデルとのインタフェースが重要
- MPIをXMPから呼び出すインタフェース
- (MPIからXMPを呼び出すインタフェース)
- XMPから、MPIで記述された並列ライブラリを呼び出す方法
  - ScalapackやMUMPSを対象
  - XMPの分散配列記述から、Scalapackのディスクリプタを作る
  - XMPで配列を設定、ライブラリを呼び出す
  - その場合、直によびだすか、wrapperをつくるか。

その他 …

- C++対応
- エラーメッセージ、デバック支援、性能チューニング環境
- ドキュメンテーション

Towards the Next Flagship Machine



XMPとpost T2K & Post K

XcalableMP

- XMPを、エクサスケールに向けた並列プログラミング言語として位置づけている。
  - メニーコアを使いこなすためのプログラミング環境
  - エクサスケールに向けたソフトウエア開発環境

並列プログラミング言語・モデルの研究・開発・普及

- まず、エクサに限った話ではない。今でも、十分に問題。
- 実用的な言語・モデルが一般に普及するには、（成功したとして）10年かかる。
  - OpenMPも10年、それにきっかけが必要
- アプリが言語で書かれるためには、標準化（デファクトでもいい）が必要
  - どこでも動くという保証が必要
  - 多くのプラットフォームで使えることが必要
- 普及には、教育が必要
  - 新しい言語の普及へのハードルは高い

おわりに

- 最近は、機能の議論についてはひと段落
- これからはしばらくは、アプリでの経験を積み重ねることが重要。
- PGASは、これからのプログラミングモデルとして注目されつつある。Coarrayは、Fortran 2008に規格に入り、メーカーのFortranコンパイラでもサポートされつつある。

- エクサスケールに向けた新たな機能の議論を始める時期

ご清聴ありがとうございました。