並列プログラミング言語XcalableMP プロジェクトの概要

佐藤　三久

XcalableMP WG,

筑波大学　計算科学研究センター

もくじ

- XcalableMPプロジェクト について
- XcalableMPの仕様
  - グローバルビューとローカルビュー
  - directives
  - プログラミング例
- HPCC ベンチマークの性能
- まとめ

“Petascale” 並列プログラミング WG

XcalableMP

- 目的
  - “標準的な”並列プログラミングのためのペタスケールを目指した並列プログラミング言語の仕様を策定する
  - “標準化”を目指して、“world-wide” communityに提案する。
- Members
  - Academia: M. Sato, T. Boku (compiler and system, U. Tsukuba), K. Nakajima (app. and programming, U. Tokyo), Nanri (system, Kyusyu U.), Okabe , Yasugi(HPF, Kyoto U.)
  - Research Lab.: Watanabe and Yokokawa (RIKEN), Sakagami (app. and HPF, NIFS), Matsuo (app., JAXA), Uehara (app., JAMSTEC/ES)
  - Industries: Iwashita and Hotta (HPF and XPFortran, Fujitsu), Murai and Seo (HPF, NEC), Anzaki and Negishi (Hitachi)
- 2007年12月にkick-off, 現在、e-scienceプロジェクトの並列プログラミング検討委員会に移行
- メーカからのコメント・要望（活動開始時）
  - 科学技術アプリケーション向けだけでなく、組み込みのマルチコアでも使えるようなものにするべき。
  - 国内の標準化だけでなく、world-wideな標準を目指す戦略を持つべき
  - 新しいものをつくるのであれば、既存の並列言語（HPF やXPFortranなど）からの移行パスを考えてほしい

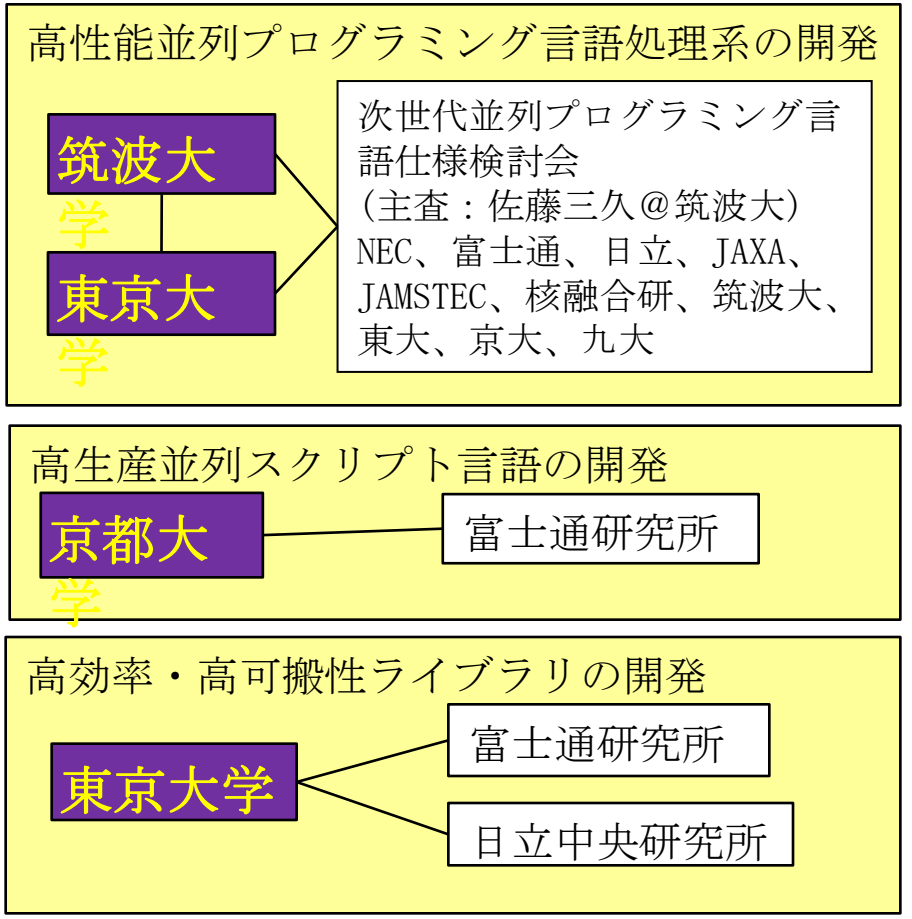
文部科学省次世代IT基盤構築のための研究開発「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」

「シームレス高生産・高性能プログラミング環境」(代表 東京大学 石川裕、H20-23, 3.5年)

「並列アプリケーション生産性拡大のための道具」の開発

PCクラスタから大学情報基盤センター等に設置されているスパコンまで、ユーザに対するシームレスなプログラミング環境を提供

- 高性能並列プログラミング言語処理系
  - 逐次プログラムからシームレスに並列化および高性能化を支援する並列実行モデルの確立とそれに基づく並列言語コンパイラの開発
- 高生産並列スクリプト言語
  - 最適パラメータ探索など粗粒度の大規模な階層的並列処理を、簡便かつ柔軟に記述可能で処理効率に優れたスクリプト言語とその処理系の開発
- 高効率・高可搬性ライブラリの開発
  - 自動チューニング(AT)機構を含む数値計算ライブラリの開発
  - PCクラスタでも基盤センタースパコン(1万規模CPU)でも単一実行時環境を提供するSingle Runtime Environment Image環境の提供



T2K Open Supercomputer Alliance

e-science XcalableMP プロジェクト

- 現状と課題
  - 並列プログラムの大半はMPI通信ライブラリによるプログラミング
    - 生産性が悪く、並列化のためのコストが高い。
  - 並列プログラミングの教育のための簡便で標準的な言語がない（MPIでの教育にとどまっている）
  - 研究室のPCクラスタから、センター、ペタコンまでに到るスケーラブルかつポータブルな並列プログラミング言語が求められている
- 目標
  - 既存言語を指示文により拡張し、これからの大規模並列システム（分散メモリシステムと共有メモリノード）でのプログラミングを助け、生産性を向上させる並列プログラミング言語を設計・開発する。
  - 標準化をすることを前提に、ユーザのわかりやすさを第一にどこでも使えるということを重視し、開発ならびに普及活動を進める。

Current Problem ?!

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

main(int argc, char**argv){
  int i,j,res,temp_res, dx,llimit,ulimit,size,rank;

  MPI_Init(argc, argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  dx = YMAX/size;
  llimit = rank * dx;
  if(rank != (size - 1)) ulimit = llimit + dx;
  else ulimit = YMAX;

  temp_res = 0;
  for(i = llimit; i < ulimit; i++)
    for(j = 0; j < 10; j++){
      array[i][j] = func(i, j);
      temp_res += array[i][j];
    }

  MPI_Allreduce(&temp_res, &res, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
  MPI_Finalize();
}
```

MPIしか、使えるものがない
MPIの並列プログラムはむずかしい… いっぱい書き換えないといけないし、時間がかかる、デバックもむずかしいし、…

We need better solutions!!

```
#pragma xmp template T[10]
#pragma xmp distributed T[block]

int array[10][10];
#pragma xmp aligned array[i][*] to T[i]

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

data d

add to
incremer

work sharing and data synchronization

いまのプログラムに指示文を加えるだけだから、簡単！性能チューニングも可能、…
どこでも使えるから安心、並列プログラミングも習得にもお勧め！

並列プログラミング言語：何が問題だったのか

XcalableMP

- HPFの教訓（by 坂上＠核融合研、村井@NEC)
  - 並列性とデータ分散を書いて、自動的に生成するという方針は理想的だったが、必ずしも性能は上がらなかった。期待が大きかった分、失望も大きかった。
  - ベース言語としたF90が未熟だった。Fortranだけだった。
  - 必要な情報をユーザで指示文で補ってもらうという方針だったが、どこをどうすれば最適なコードになるかが明らかでなかった。
  - 自動であるがために、通信がどこでおこっているのか、どうやってチューニングすればいいのか、ユーザに手段が与えられていなかった。
  - 完全性を求めるあまり不必要な仕様があり、実装の障害になっていた。
  - レファレンス実装が不在。教育が考慮されていない。

- 90年代の並列プログラミング言語
  - 多くはプログラミング言語の研究が主で、実際のアプリで使われることが少なかった。
  - 組織的な普及活動、標準化、教育活動がない。

“petascale” システムのプログラミング言語に要請される要素

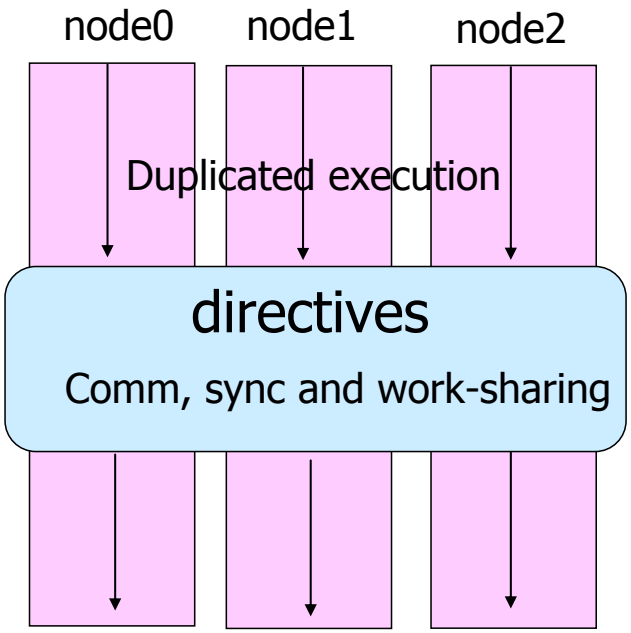
XcalableMP

- Performance
  - ユーザはMPIと同等の性能を引き出すことができること
  - MPIにはない要素も！ – one-sided communication (remote memory copy)
- Expressiveness
  - ユーザはMPIでのプログラミングと同等のことが、MPIよりも簡単に書けること。
  - 例えば、Task parallelism – for multi-physics
- Optimizability
  - コンパイラの解析や最適化のために構造的な記述を提供すること
  - ハードウエアのネットワークトポロジーにマッピングする機能
- Education cost
  - CSでないユーザに対して、必ずしも新しくなくてもいいので、実用的な機能を提供すること。

XcalableMP http://www.xcalablemp.org XcalableMP

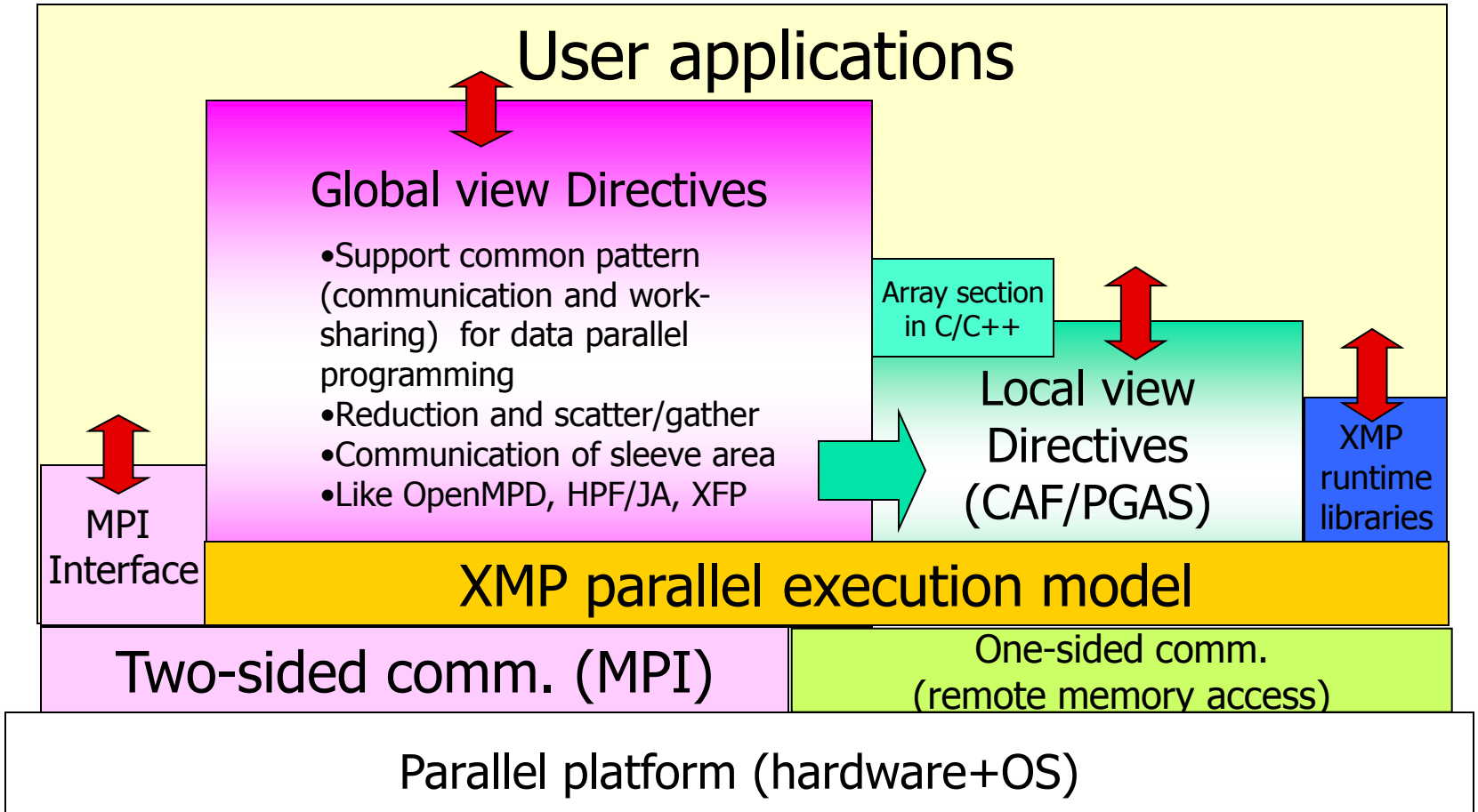
XcalableMP : directive-based language eXtension for Scalable and performance-tunable Parallel Programming

- **Directive-based language extensions** for familiar languages F90/C/C++
  - コードの書き換えや教育のコストを抑えること
- **“Scalable” for Distributed Memory Programming**
  - SPMDが基本的な実行モデル。
  - MPIのように、各ノードでスレッドが独立に実行を開始する。
  - 指示文(directive)がなければ、重複実行
  - タスク並列のためのMIMD実行も
- **“performance tunable” for explicit communication and synchronization.**
  - 指示文を実行するときに、Work-sharing や通信・同期がおきる。
  - すべての同期・通信操作は、指示文によって起きる。HPFと異なり、パフォーマンスのチューニングがわかりやすくなる。



Overview of XcalableMP

- XMP は、グローバルビューのデータ並列とwork sharingによって、典型的な並列化をサポート
  - もとの逐次コードは、OpenMPのように指示文で並列化ができる。
- これに加えて、ローカルビューとして、CAF-like PGAS (Partitioned Global Address Space) 機能を提供



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

The same code written in MPI

XcalableMP

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

main(int argc, char**argv){
  int i,j,res,temp_res, dx,llimit,ulimit,size,rank;

  MPI_Init(argc, argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  dx = YMAX/size;
  llimit = rank * dx;
  if(rank != (size - 1)) ulimit = llimit + dx;
  else ulimit = YMAX;

  temp_res = 0;
  for(i = llimit; i < ulimit; i++)
    for(j = 0; j < 10; j++){
      array[i][j] = func(i, j);
      temp_res += array[i][j];
    }

  MPI_Allreduce(&temp_res, &res, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
  MPI_Finalize();
}
```

ノード、テンプレート、データとループの分散

- HPFから、取り入れたアイデア
- ノードは、分散メモリ環境のプロセッサ（複数）とメモリのabstraction.

#pragma xmp nodes p(32)

- テンプレートとは、ノード上に分散配置されたダミー配列

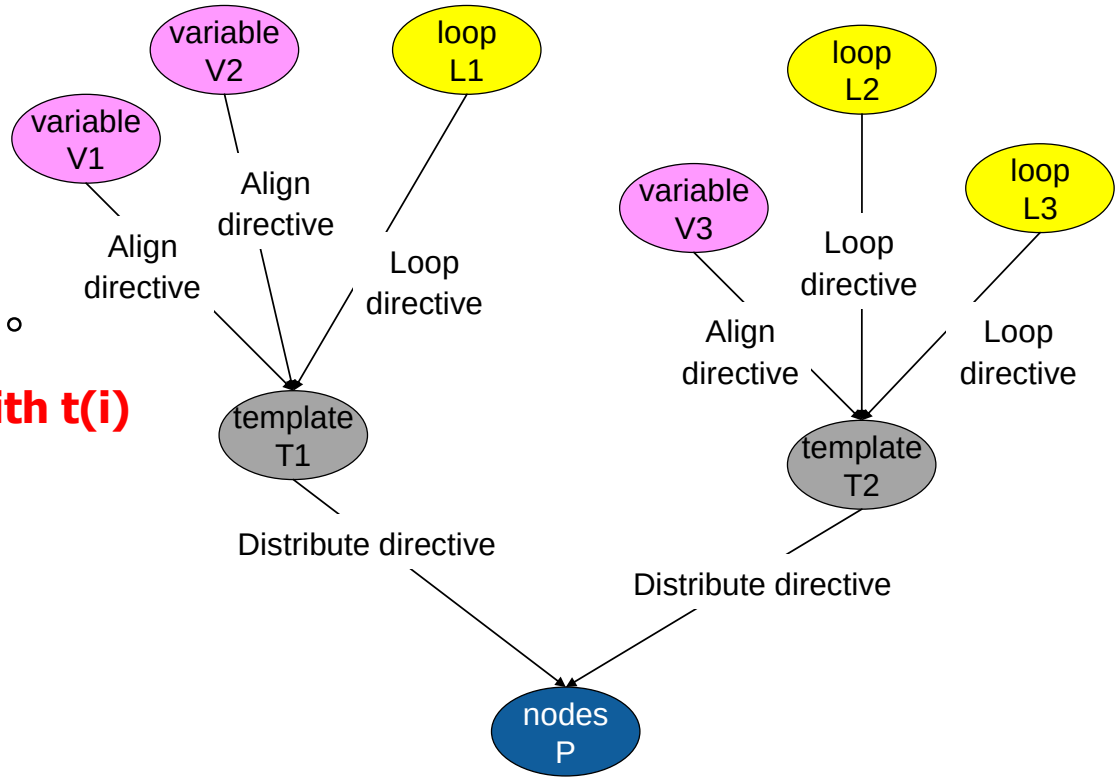
#pragma xmp template t(100)
#pragma distribute t(block) on p

- 分散されるデータは、
  テンプレートにalign（整列）する。

#pragma xmp align array[i][*] with t(i)

- ループのiterationも、on節に
  よって、テンプレートにalignする。

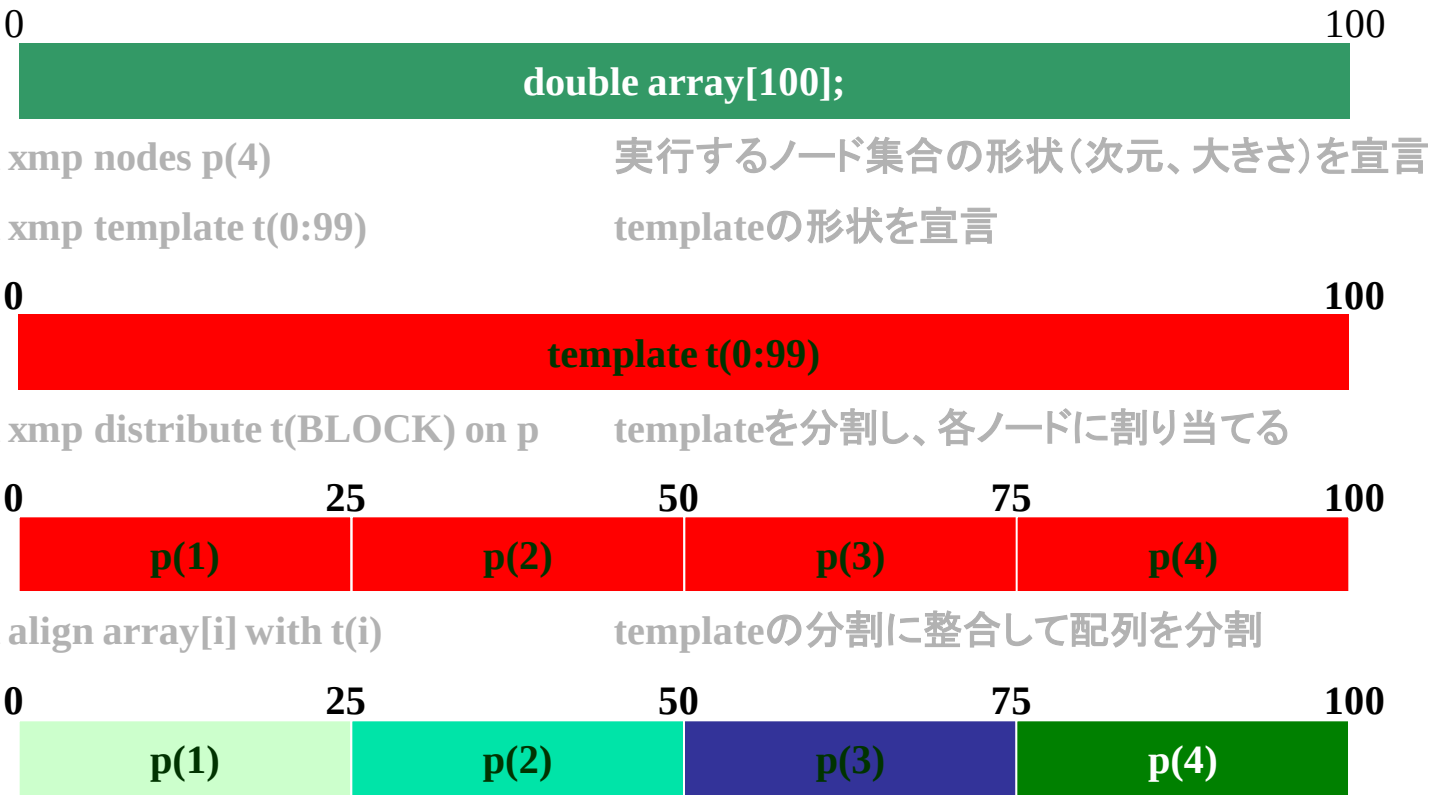
#pragma xmp loop on t(i)



templateを用いたindex空間の分割

- template
  - index空間を表す仮想的な配列
  - 配列の分割・ループ文の並列実行に用いる

templateを用いた配列の分割

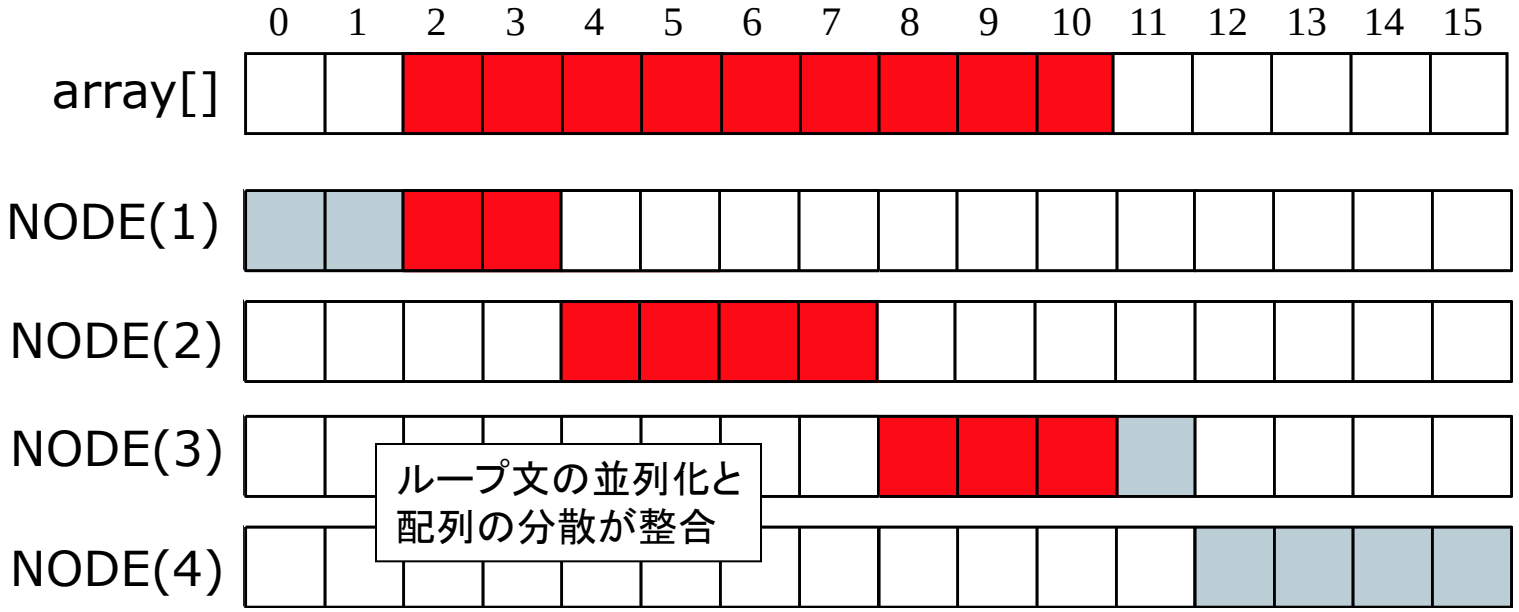


ループ文とタスクの並列実行

- **#pragma xmp loop on *template***
  - ループ文の並列実行をtemplateで指定
  - 配列の分割と整合しなければならない

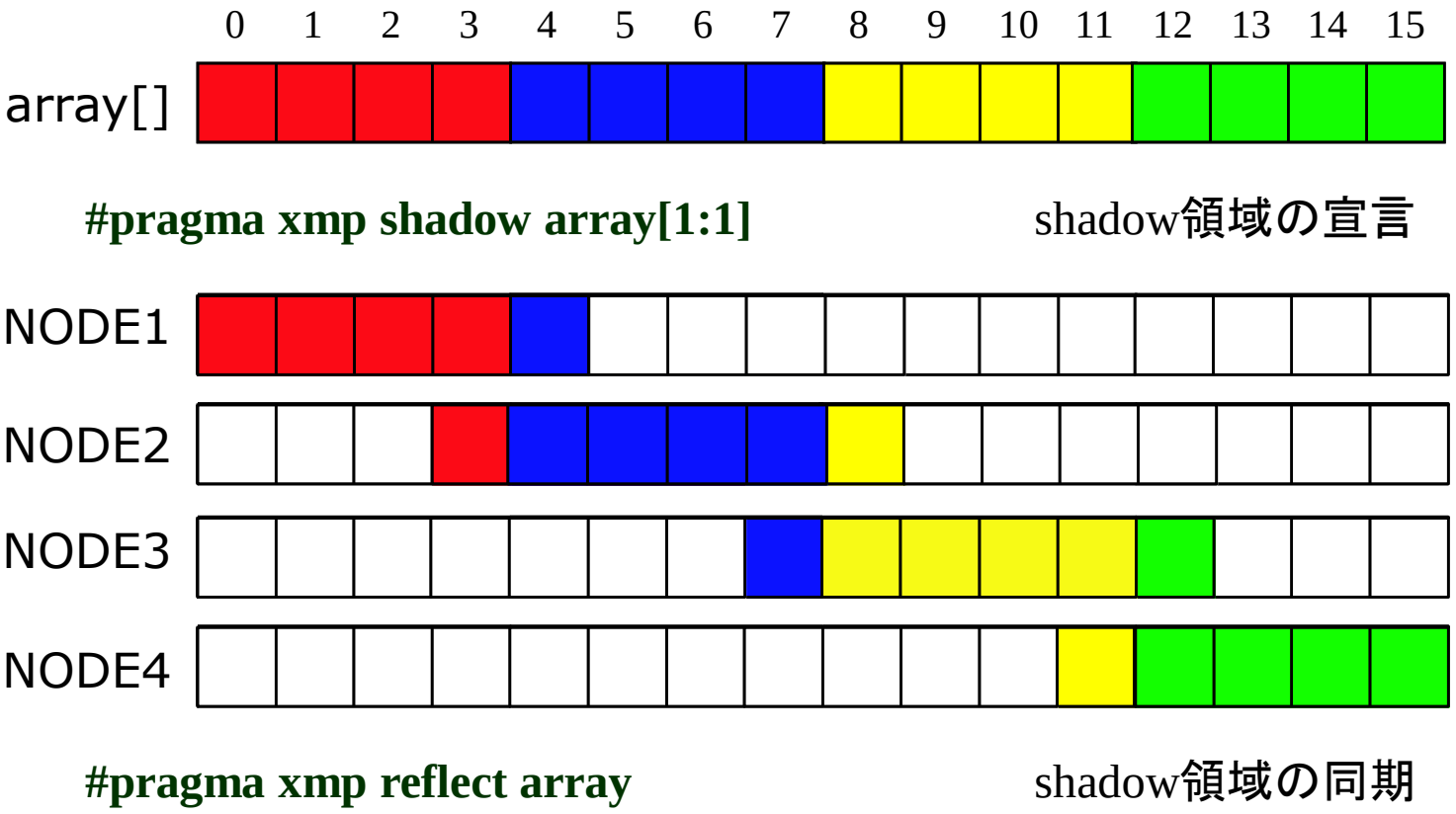
  例）　**#pragma xmp loop on t(i)**

  for(i = 2; i <= 10; i++) array[i] = . . .



配列の重複宣言と同期

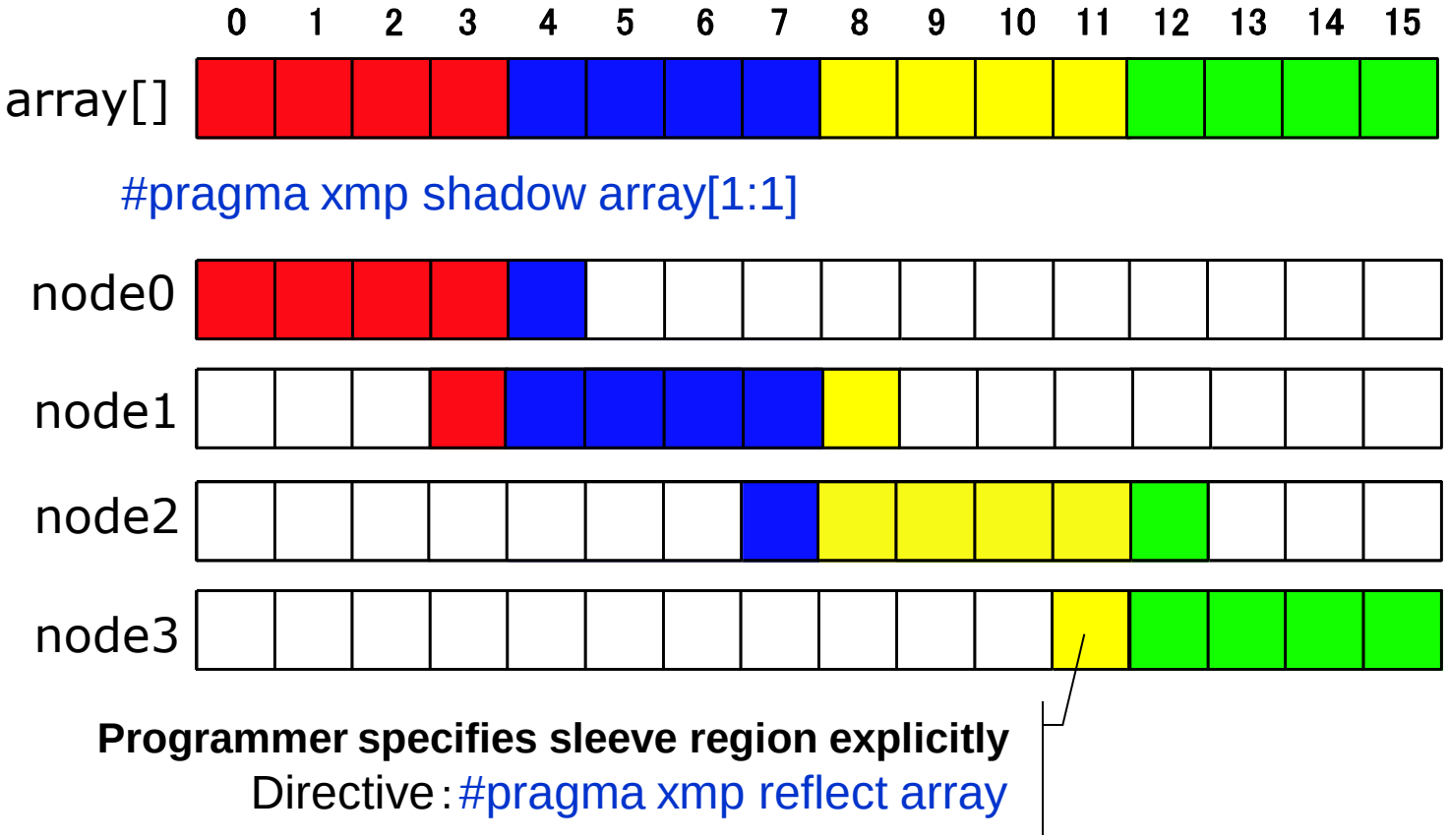
- 他のノードに割り当てられた要素を参照
  - XMPではメモリアクセスで常にローカルメモリを参照
  - 配列の重複宣言と同期：**shadow**, **reflect**指示文



Data synchronization of array (shadow)

- Exchange data only on "shadow" (sleeve) region
  - If neighbor data is required to communicate, then only sleeve area can be considered.
  - example：b[i] = array[i-1] + array[i+1]

#pragma xmp align array[i] with t(i)



XcalableMP コード例 (laplace, global view)

```
#pragma xmp nodes p[NPROCS]
#pragma xmp template t[1:N]
#pragma xmp distribute t[block] on p

double u[XSIZE+2][YSIZE+2],
       uu[XSIZE+2][YSIZE+2];
#pragma xmp aligned u[i][*] to t[i]
#pragma xmp aligned uu[i][*] to t[i]
#pragma xmp shadow uu[1:1]

lap_main()
{
 int x,y,k;
 double sum;
    …
```

```
for(k = 0; k < NITER; k++){
        /* old <- new */
#pragma xmp loop on t[x]
        for(x = 1; x <= XSIZE; x++)
          for(y = 1; y <= YSIZE; y++)
            uu[x][y] = u[x][y];
#pragma xmp reflect uu
#pragma xmp loop on t[x]
        for(x = 1; x <= XSIZE; x++)
          for(y = 1; y <= YSIZE; y++)
            u[x][y] = (uu[x-1][y] + uu[x+1][y
                 uu[x][y-1] + uu[x][y+1])/4.0
    }
    /* check sum */
    sum = 0.0;
#pragma xmp loop on t[x] reduction(+:sum)
    for(x = 1; x <= XSIZE; x++)
        for(y = 1; y <= YSIZE; y++)
          sum += (uu[x][y]-u[x][y]);
#pragma xmp block on master
    printf("sum = %g¥n",sum);
}
```

ノードの形状の定義

Templateの定義と
データ分散を定義

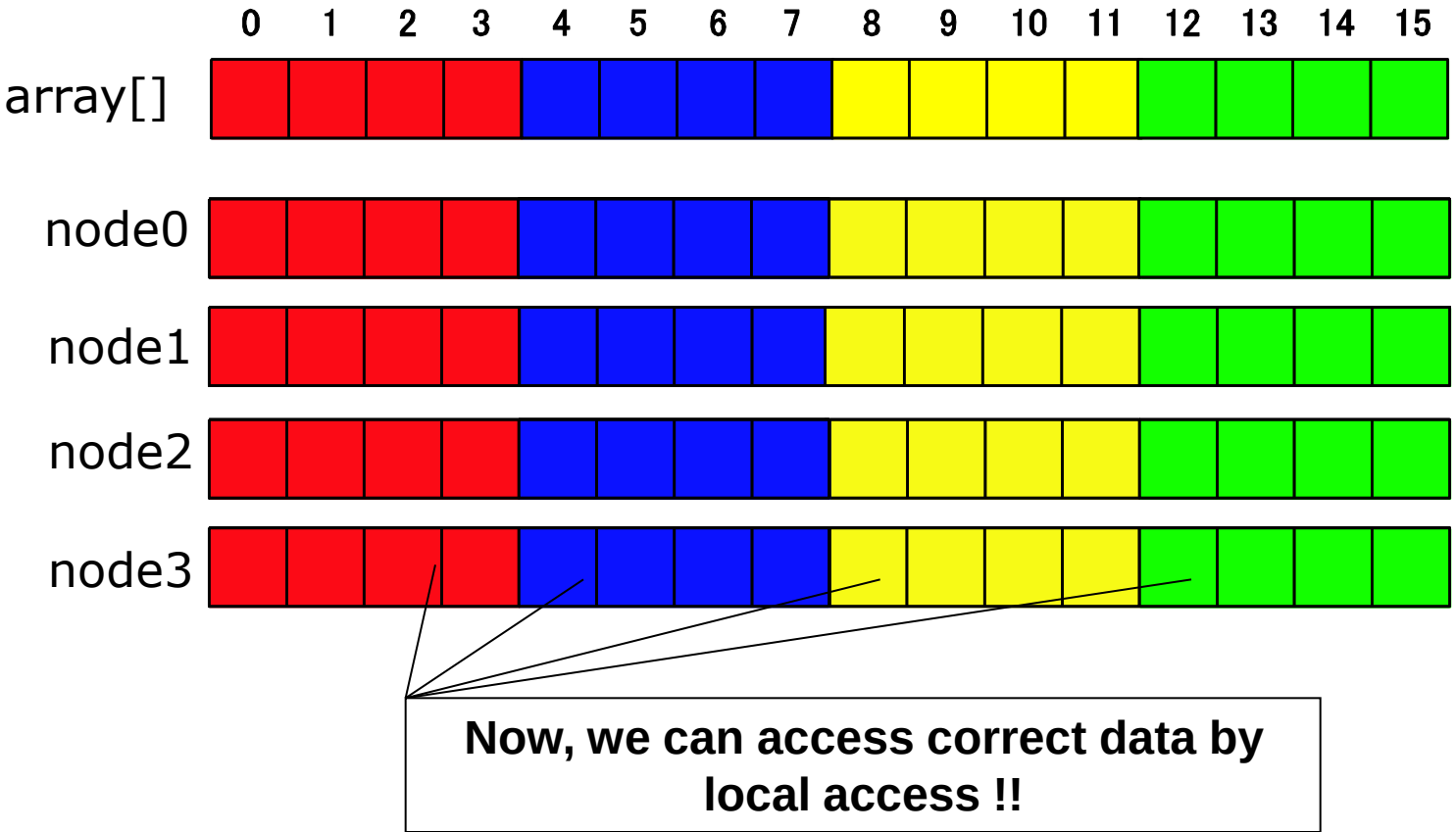
データの分散は、
templateにalign
データの同期のための
shadowを定義、この場
合はshadowは袖領域

Work sharing
ループの分散

データの同期

Data synchronization of array (full shadow)

- Full shadow specifies whole data replicated in all nodes
  - #pragma xmp shadow array[*]
- reflect operation to distribute data to every nodes
  - #pragma reflect array
  - Execute communication to get data assigned to other nodes
  - Most easy way to synchronize → But, communication is expensive!



XcalableMP コード例 (NPB CG, global view)

```
#pragma xmp nodes p[NPROCS]
#pragma xmp template t[N]
#pragma xmp distributed t[block] on p
...
#pragma xmp aligned [i] to t[i] :: x,z,p,q,r,w
#pragma xmp shadow [*] :: x,z,p,q,r,w
...
```

ノードの形状の定義

Templateの定義とデータ分散を定義

データの分散は、templateにalignデータの同期のためのshadowを定義、この場合はfull shadow

Work sharing ループの分散

データの同期

```
/* code fragment from conj_grad in NPB CG */
sum = 0.0;
#pragma xmp loop on t[j] reduction(+:sum)
    for (j = 1; j <= lastcol-firstcol+1; j++) {
        sum = sum + r[j]*r[j];
    }
    rho = sum;
for (cgit = 1; cgit <= cgitmax; cgit++) {
#pragma xmp reflect p
#pragma xmp loop on t[j]
        for (j = 1; j <= lastrow-firstrow+1; j++) {
            sum = 0.0;
            for (k = rowstr[j]; k <= rowstr[j+1]-1; k+
                sum = sum + a[k]*p[colidx[k]];
            }
            w[j] = sum;
        }
#pragma xmp loop on t[j]
        for (j = 1; j <= lastcol-firstcol+1; j++) {
            q[j] = w[j];
        }
```

通信・同期の操作

以下のような通信を指示文で記述することが可能

- **#pragma xmp bcast *var* on *node***
  - データのブロードキャスト
- **#pragma xmp barrier**
  - バリア同期
- **#pragma xmp reduction (*var*:*op*)**
  - リダクション操作（総和、最大値の計算など）
- **#pragma xmp gmove**
  - 直後の代入文がローカル領域ではなく、データが割り当てられたノードの値を参照するように通信を生成

  例）　**#pragma xmp gmove**

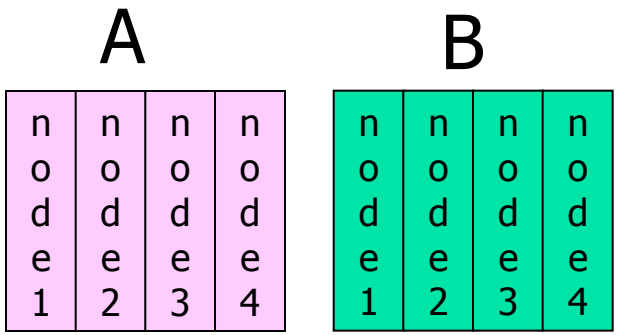
  x = array[100];

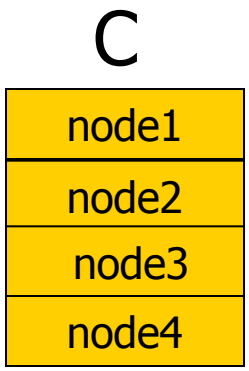
  (array[100]が割り当てられたノードからデータを転送する)

gmove directive

- The "gmove" construct copies data of distributed arrays in global-view.
  - When no option is specified, the copy operation is performed *collectively* by all nodes in the executing node set.
  - If an "in" or "out" clause is specified, the copy operation should be done by one-side communication ("get" and "put") for remote memory access.

```
!$xmp nodes p(*)
!$xmp template t(N)
!$xmp distribute t(block) to p
real A(N,N),B(N,N),C(N,N)
!$xmp align A(i,*), B(i,*),C(*,i) with t(i)

    A(1) = B(20)        // it may cause error
!$xmp gmove
    A(1:N-2,:) = B(2:N-1,:) // shift operation
!$xmp gmove
    C(:,:) = A(:,:)      //  all-to-all
!$xmp gmove out
    X(1:10) = B(1:10,1) // done by put operation
```





XcalableMP (local view)

- Co-Array Fortran
  - 代入文の形式でノード間通信を記述

    例）　real dimension a(100)**[*]**　（Co-array宣言）

    . . .

    b(:) = a(:)[1]　（ノード1からデータを転送）

    Co-array次元

- XcalableMPでは、何も指示をしなければ単なるSPMDのプログラム
  - Local viewでは、ノード内のオペレーションを中心に操作。PGAS (Partitioned Global Address Space) 機能により、他ノードのデータを参照できるようにして最適化を支援

- XcalableMPのローカルビュー
  - CAF相当の機能を提供
  - XMP-Fortran：CAF互換

```
int A[10]:
int B[5];

A[4:9] = B[0:4];
```

Array sectionの導入

  - XMP-C：**coarray**指示文 ＋ 構文拡張(array section: 部分配列記述)

  - 片側通信の記述
    - remote memory access機能

    （one-sided通信）をサポート
  - より自由な並列化が可能

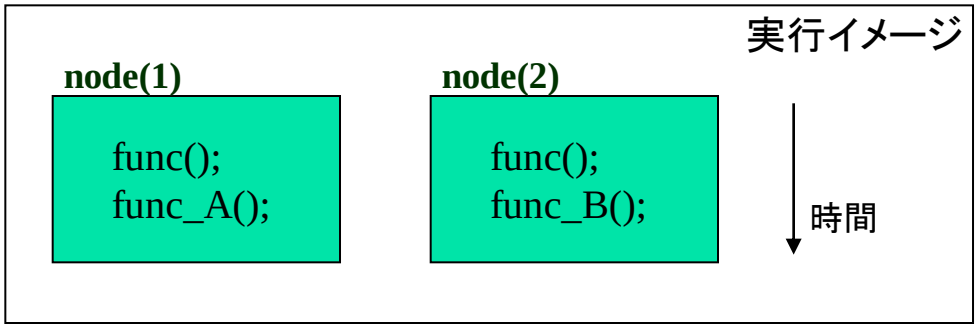
```
int A[10], B[10];
#pragma xmp coarray [*]: A, B
…
A[:] = B[:]:[2];
```

タスクの並列実行

- **#pragma xmp task on *node***
  - 直後のブロック文を実行するノードを指定

例）

```
func();
#pragma xmp tasks
{
#pragma xmp task on node(1)
   func_A();
#pragma xmp task on node(2)
   func_B();
}
```



  - 異なるノードで実行することでタスク並列化を実現

ハイブリッドな並列化

XcalableMP

- グローバルビューとローカルビューの連携
  - 最初はグローバルビュー
  - 性能チューニングのためにローカルビューを導入

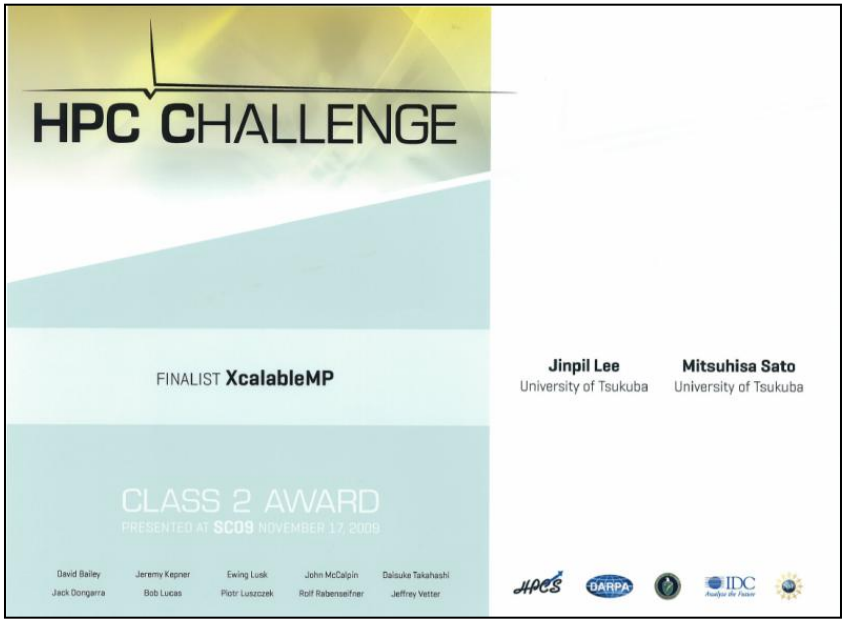
    → インクリメンタルな並列化
  - 連携のためのインターフェイス
    - グローバルビューとローカルビューではindexが異なる
    - 同じ配列に対して二つの名前を提供
    - index変換のための組み込み関数の提供

- OpenMP, MPIとの連携
  - 足りない機能を補う
  - 性能のチューニング

HPCCベンチマークのプログラミングと性能

- HPC Challenge Benchmark Class2
  - 新しい並列プログラミング言語での記述性と性能を競うカテゴリ
    - Class1はシステム性能
  - 4つのベンチマーク
    - STREAM
    - Random Access
    - HPL
    - FFT
  - 今年は、Awardは、性能はIBM(X10 and UPC), 記述性はCary (Chapel) になった

SC09 HPCC Class2 でFinalist!



性能評価環境

- XMP/C: C言語版のprototype compilerを実装
  - データ並列の基本的な機能のみを実装
  - (Some parts ware compiled by hand)

- 評価に際しては、主にprogrammabilityに焦点を当てた
  - STREAM, RandomAccess, HPL, FFT are parallelized by XMP

T2K OpenSupercomputer – Tsukuba System (2to32nodes)

CPU	AMD Opteron Quad-core 8000series 2.3Ghz x 4sockets (16 cores)
MEM	32GB
NETWORK	InfiniBand (x4 rails)
MPI lib	MVAPICH2 - 1.2

HPCC Benchmark1: STREAM

XcalableMP

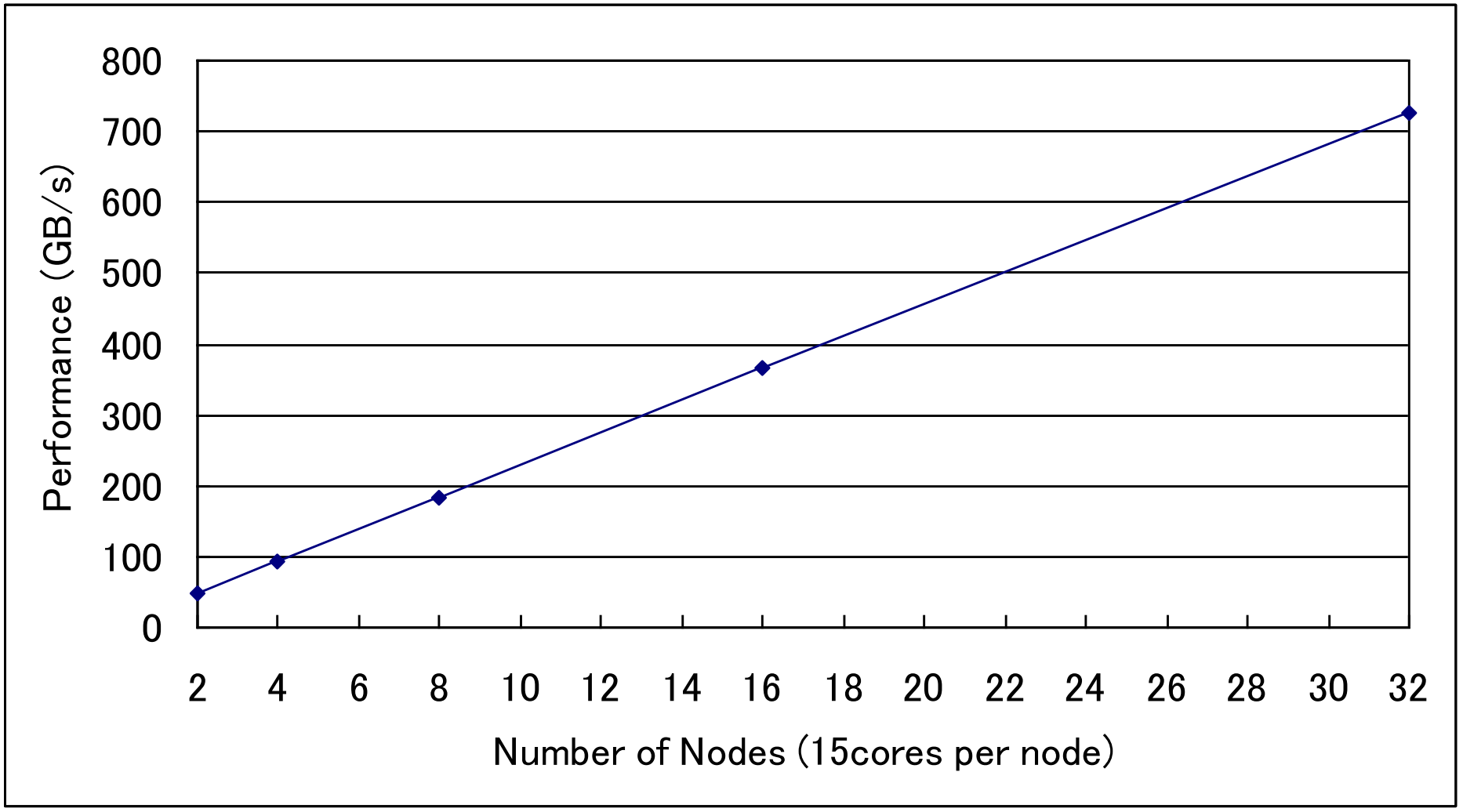
- Global view programming with directives
  - very straightforward to parallelize by a loop directive

```
double a[SIZE] , b[SIZE] , c[SIZE];
#pragma xmp nodes p(*)
#pragma xmp template t(0:SIZE−1)
#pragma xmp distribute t(block) onto p
#pragma xmp align [j] with t(j) :: a, b, c
. . .
# pragma xmp loop on t(j)
for (j = 0; j < SIZE; j++) a[j] = b[j] + scalar*c[j];
. . .
#pragma xmp reduction(+:triadGBs)
```

Performance of STREAM

XcalableMP

- Lines Of Code: 98



HPCC Benchmark2: Random Access

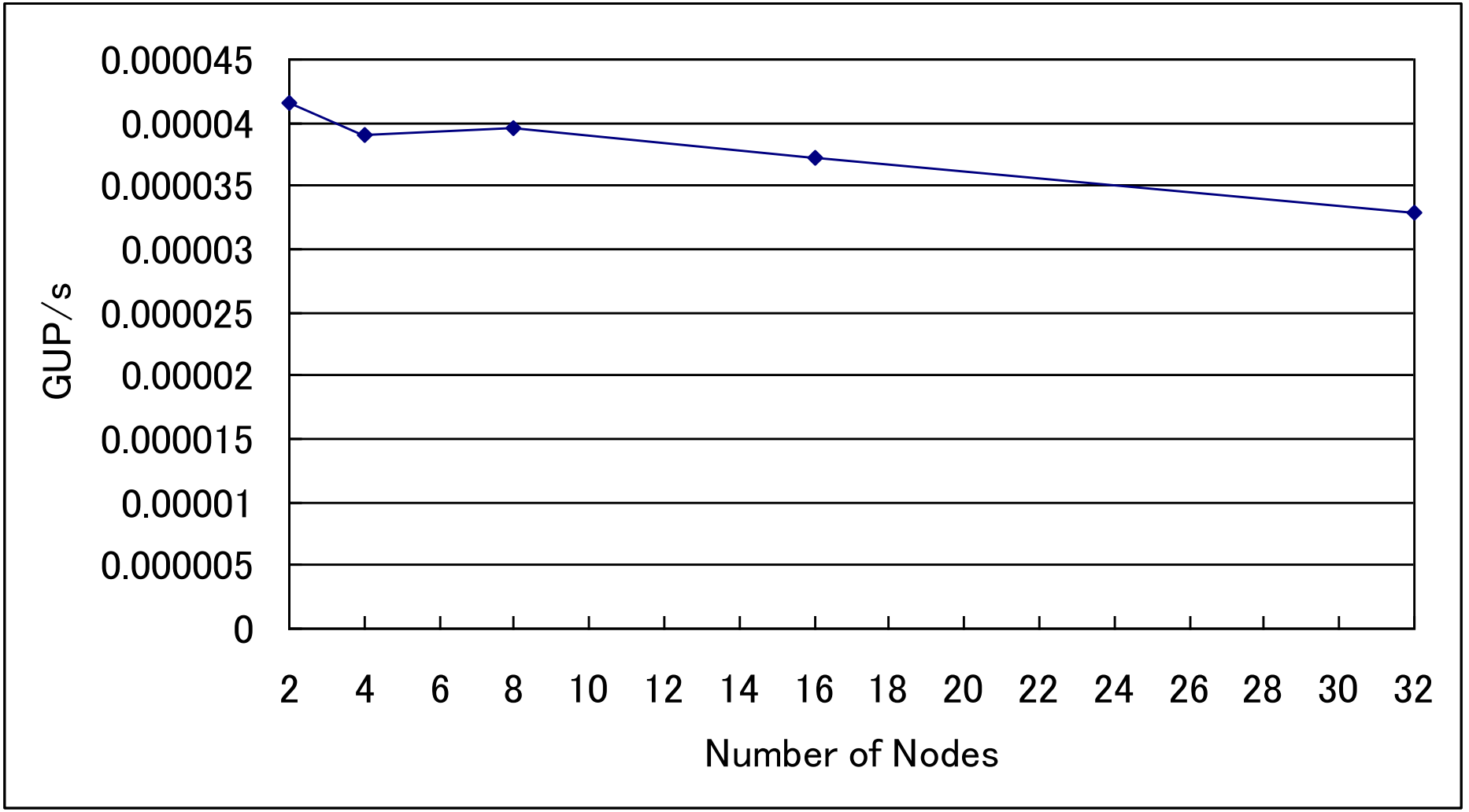
- Local view programming with co-array

```
#define SIZE TABLE_SIZE/PROCS
u64Int Table[SIZE] ;
#pragma xmp nodes p(PROCS)
#pragma xmp coarray Table [PROCS]
. . .
for (i = 0; i < SIZE; i++) Table[i] = b + i ;
. . .
for (i = 0; i < NUPDATE; i++) {
  temp = (temp << 1) ^ ((s64Int)temp < 0 ? POLY : 0);
  Table[temp%SIZE]:[(temp%TABLE_SIZE)/SIZE] ^= temp;
}
#pragma xmp barrier
```

Performance of Random Access

XcalableMP

- Lines Of Code: 77
- complied into MPI2 one-sided functions



HPCC Benchmark3: HPL

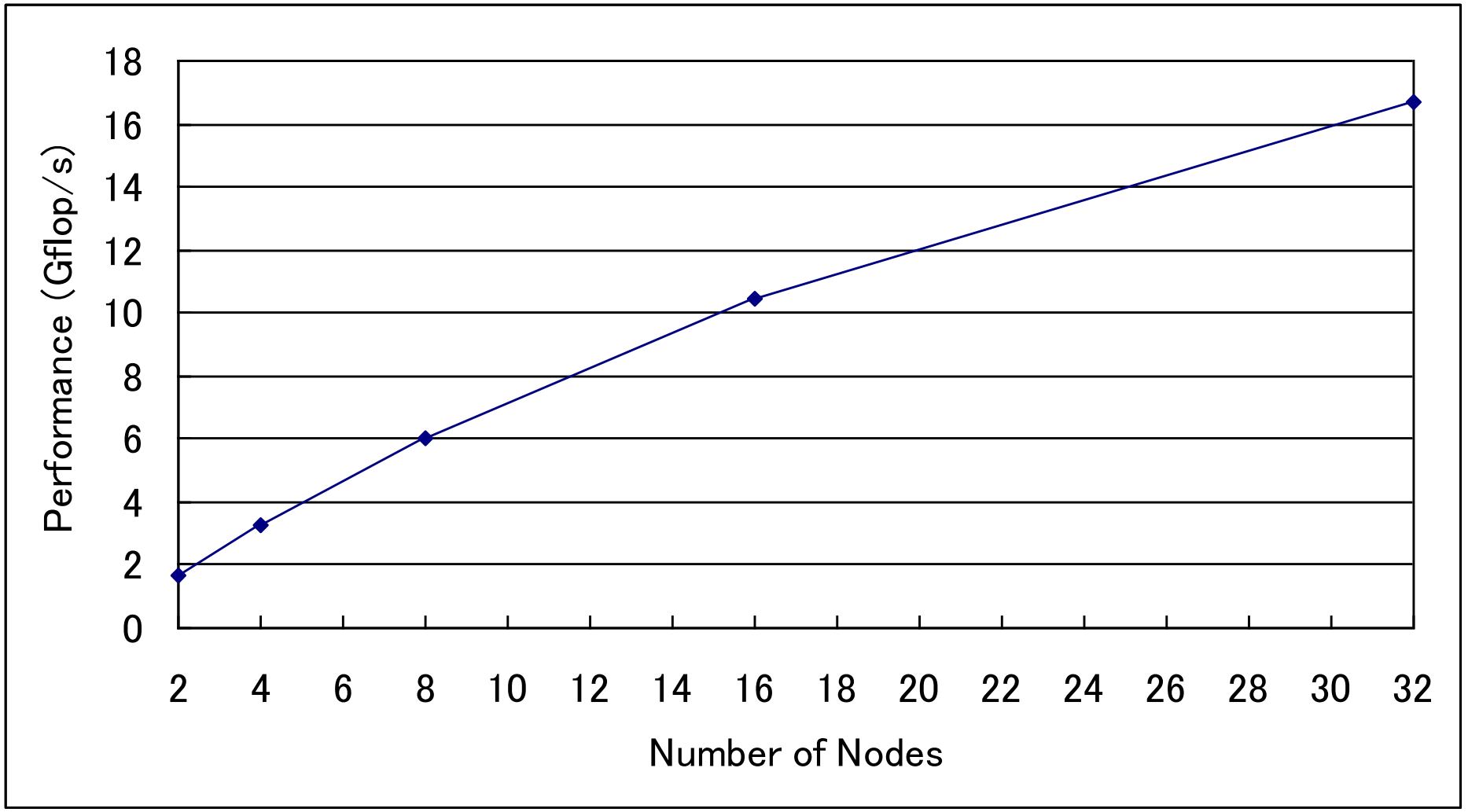
- Parallelized in global view
- Matrix/vectors are distributed in cyclic manner in one dimension.
- Using **gmove** to exchange columns for pivot exchange

dgefa function:

```
#pragma xmp gmove
    pvt_v[k:n-1] = a[k:n-1][l];

    if (l != k) {
#pragma xmp gmove
      a[k:n-1][l] = a[k:n-1][k];
#pragma xmp gmove
      a[k:n-1][k] = pvt_v[k:n-1];
    }
```

Performance of HPL

- Lines Of Code: 243



HPCC Benchmark4: FFT

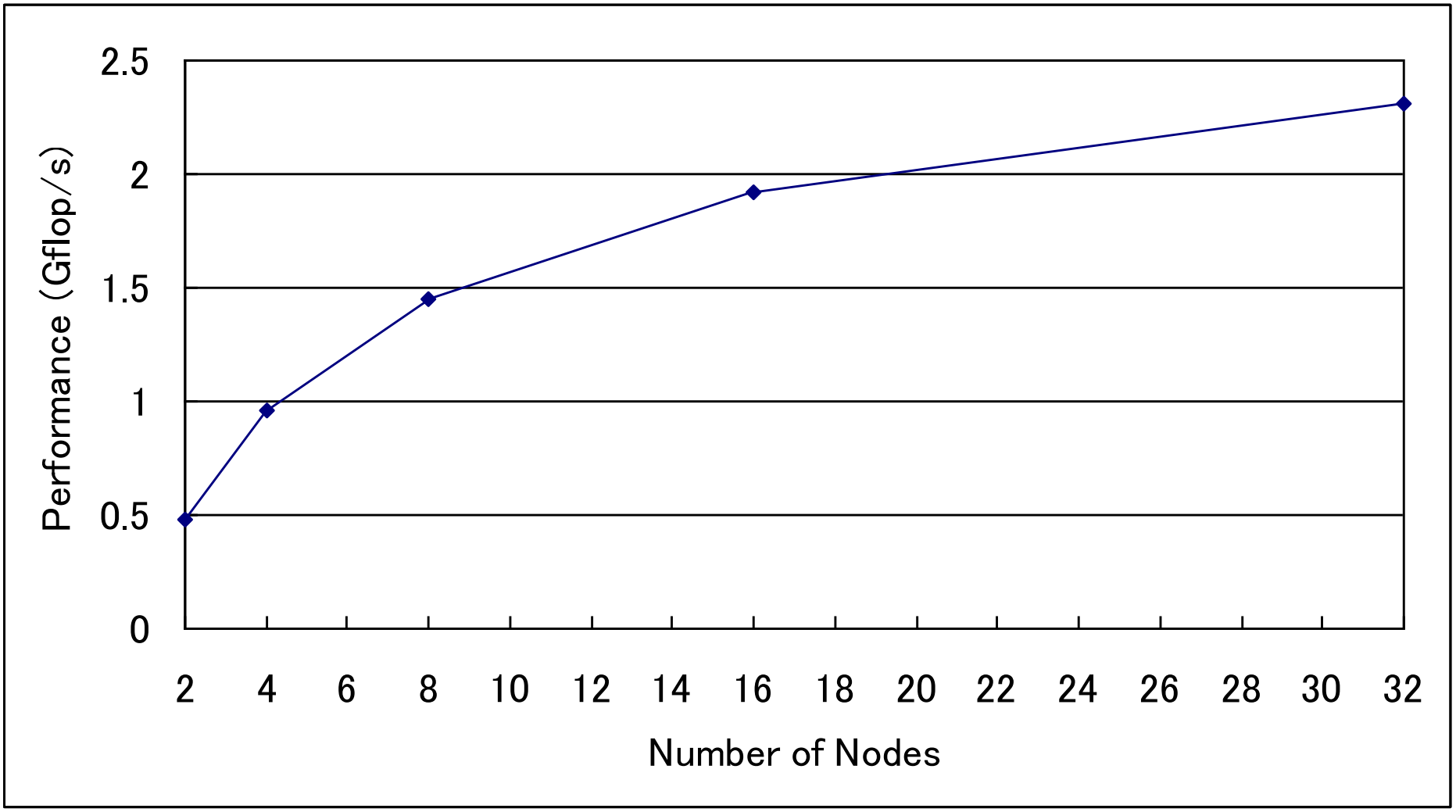
XcalableMP

- Parallelized in global view
- Using six-step FFT algorithm
  - Matrix transpose is a key operation.
- Matrix transpose using **gmove**

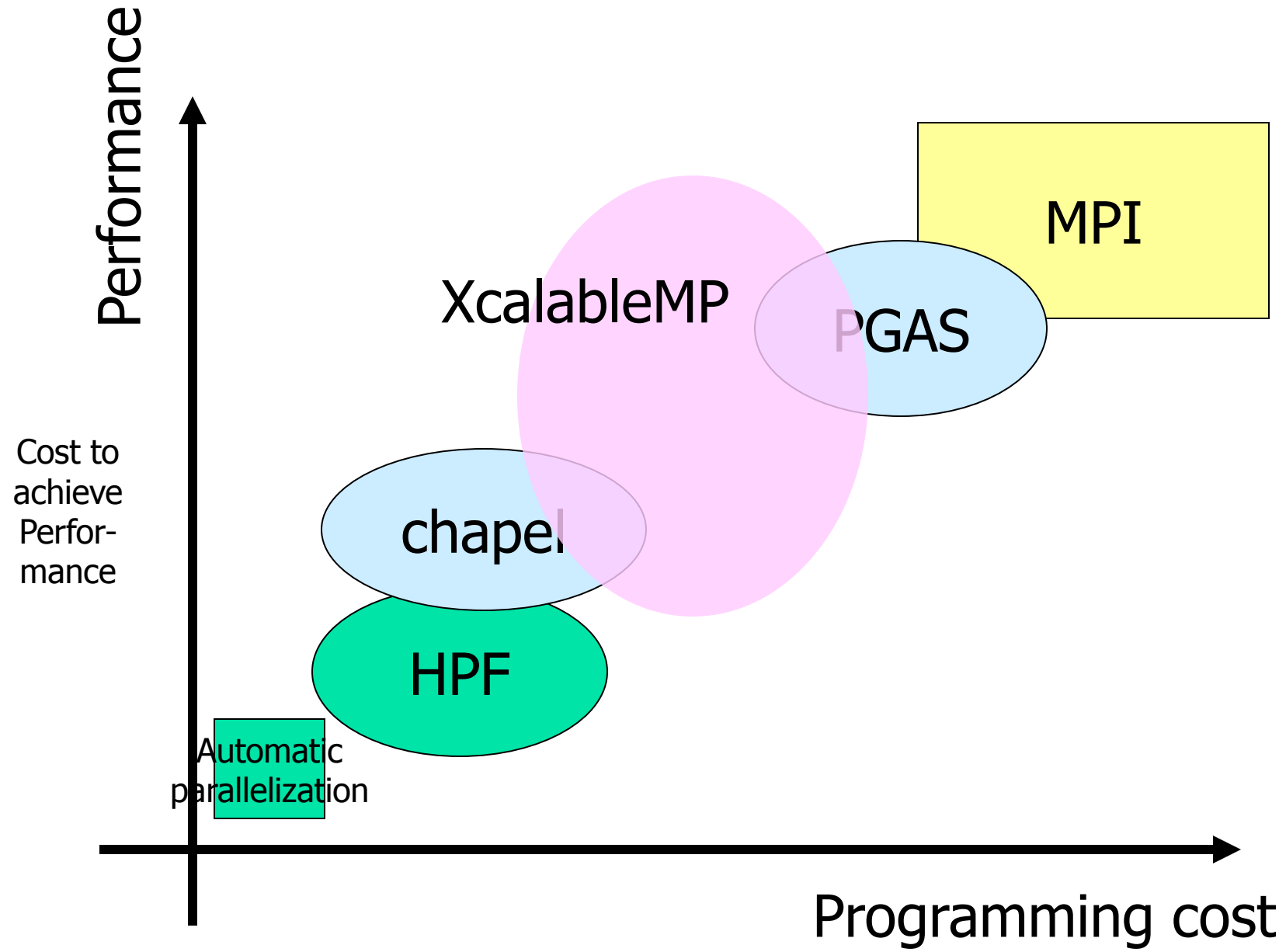
```
#pragma xmp align a_work[*][i] with t1(i)
#pragma xmp align a[i][*] with t2(i)
#pragma xmp align b[i][*] with t1(i)
. . .
#pragma xmp gmove
  a_work[:][:] = a[:][:];                    // all-to-all

#pragma xmp loop on t1(i)
  for(i = 0; i < N1; i++)
    for(j = 0; j < N2; j++)
      c_assgn(b[i][j], a_work[j][i]);
```

Performance of FFT

- Lines Of Code: 217



Position of XcalableMP



おわりに

- XcalableMPの目的・目標
  - 超並列マシンの並列プログラミングにはいろいろな課題はあるが、… 生産性(productivity)をあげることが重要
  - 「MPIよりもましなプログラミング環境を！」
- XcalableMP: これからの計画
  - 現在、XMP Spec は、version 0.9
    - http://www.xcalablemp.org で、公開中
  - C言語版・デモ版βリリースは2010/2Q (4月？)
  - 2010/3QにFortran版 (SC10前)
- 課題
  - マルチコア対応 (SMPノード)
  - ライブラリ、I/O

http://www.xcalablemp.org

お願い

- XMPは、HPFでの経験を重視している
- 普及については、これまでのHPF協議会のご経験に基づき、アドバイスいただきたい。

- 特に、メーカーがサポートしてくれるようにならないと、普及はしない。
- そのための戦略は？

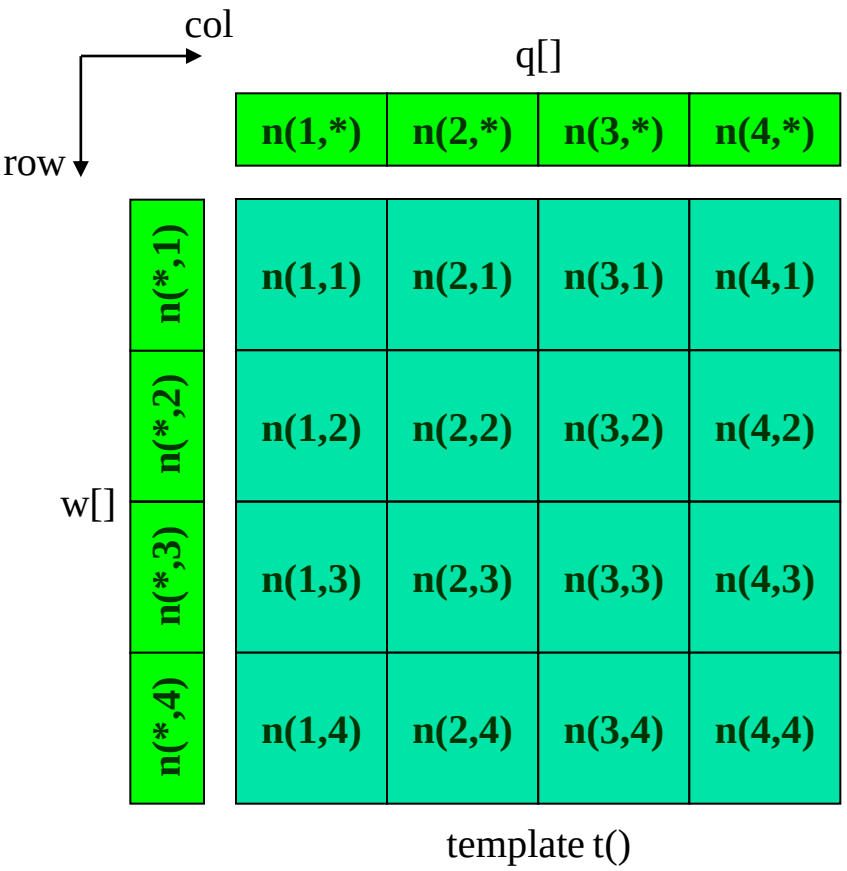
backup

NPB-CGの並列化（データ分割）

XcalableMP

- ベクトルデータの分割を指示文で宣言

```
#pragma xmp nodes on n(NPCOLS,NPROWS)
#pragma xmp template t(0:na+1,0:na+1)
#pragma xmp distribute t(BLOCK,BLOCK) on n

double    x[na+2], z[na+2], p[na+2],
          q[na+2], r[na+2], w[na+2];

#pragma xmp align [i] with t(i,*):: x,z,p,q,r
#pragma xmp align [i] with t(*,i):: w
```



- 行列データ **a[], rowstr[], colidx[]** の分割は手動で行う
  1. **ローカル配列として宣言**
  2. 行列要素のindexが割り当てられたtemplateの中

  → **ローカル配列a[]に収納し、index情報を記録**

  （MPIと同じ手法）

2次元分割できる！　OpenMPは1次元だけ。

NPB-CGの並列化（ループ並列化と通信の記述）

```
static void conj_grad()
{
  . . .
#pragma xmp loop on t(j,*)
  for(j = 0; j < lastcol-firstcol+1; j++) {
    x[j] = norm_temp12*z[j];                          （ベクトルの計算）
  }

#pragma xmp loop on t(*,j)
  for(j = 0; j < lastrow-firstrow+1; j++) {
    sum = 0.0;
    for(k = rowstr[j]; k <= rowstr[j+1]; k++) {       (手動並列化)
      sum = sum + a[k]*p[colidx[k]];
    }
    w[j] = sum;                                       （逐次コードでは q[j] = sum;）
  }

#pragma xmp reduction(+:w) on p(*,:)                  (ベクトルのリダクション操作)

#pragma xmp gmove                                     (ベクトル間のtranspose)
  q[:] = w[:];
  …
}
```

```
#pragma xmp nodes on p(NPCOL, NPROW)
#pragma xmp template t(n,n)
#pragma xmp distribute t(BLOCK,BLOCK) on p

double p[n],w[n];
double A[n][n];

#pragma xmp align A[j][i] to t(i,j)
#pragma xmp align p[i] to t(i,*)
#pragma xmp align w[j] to t(*,j)
```

```
conj_grad(...){
  …
  for(;;){

#pragma xmp loop j on t(:,j)
   for(j=0; j < n; j++){
      sum = 0;
#pragma xmp loop i on t(i,j)
      for(i = 0; i < n; n++)
          sum += a[j][i]*p[i];
     w[j] = sum;
   }
#pragma xmp reduction(+:w) on p(:,*)

#pragma xmp gmove
      p[:] = w[:];
  }
  ….
}
```