XcalableMP

# Preliminary Performance Report on programming in XcalableMP

Mitsuhisa Sato and Jinpil Lee

Center for Computational Sciences, University of Tsukuba, Japan

XcalableMP specification Working Group (XMP-WG)

hpcs lab

High Performance Computing System

# Agenda

- XcalableMP : directive-based language eXtension for Scalable and performance-aware Parallel Programming
  - Motivation and organization
  - Concept and model
  - Some examples
- Preliminary performance report of HPCC Class2 XcalableMP results

# XcalableMP Specification working group (XMP Spec WG)

- Objectives : it was "Petascale" Parallel language design working group
  - Making a draft on "petascale" parallel language for "standard" parallel programming
  - To propose the draft to "world-wide" community as "standard" (not research product)
- Members
  - Academia: M. Sato, T. Boku (compiler and system, U. Tsukuba), K. Nakajima (app. and programming, U. Tokyo), Nanri (system, Kyusyu U.), Okabe (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)
- More than 10 WG meetings have been held (Dec. 13/2007 for kick-off)
- Funding for development
  - E-science project : "Seamless and Highly-productive Parallel Programming Environment for High-performance computing" project funded by Ministry of Education, Culture, Sports, Science and Technology, JAPAN.
    - Project PI: Yutaka Ishiakwa, co-PI: Sato and Nakashima(Kyoto), PO: Prof. Oyanagi
    - Project Period: 2008/Oct to 2012/Mar (3.5 years)

# Background: HPF history in Japan

- Japanese supercomputer venders were interested in HPF and developed HPF compiler on their systems.
- NEC has been supporting HPF for Earth Simulator System.
- Many workshops: HPF Users Group Meeting (HUG from 1996-2000), HFP intl. workshop (in Japan, 2002 and 2005)
- Japan HPF promotion consortium was organized by NEC, Hitatchi, Fujitsu …
  - HPF/JA proposal
- Still survive in Japan, supported by Japan HPF promotion consortium
- Compiler Availability
  - HPF/ES (HPF+HPF/JA+some extension for Earth Simulator)
  - HPF/SX, HPF/VPP, HPF/ES for PC clusters, fhpf (free software distributed by HPF consortium)

# One Lesson learned from HPF

- No explicit mean for performance tuning .
  - Everything depends on compiler optimization.
  - Users can specify more detail directives, but no information how much performance improvement will be obtained by additional informations
    - INDEPENDENT for parallel loop
    - PROCESSOR + DISTRIBUTE
    - ON HOME
  - The performance is too much dependent on the compiler quality, resulting in “incompatibility” due to compilers.
- **Lesson :“*Specification must be clear. Programmers want to know what happens by giving directives*”**
  - The way for tuning performance should be provided.

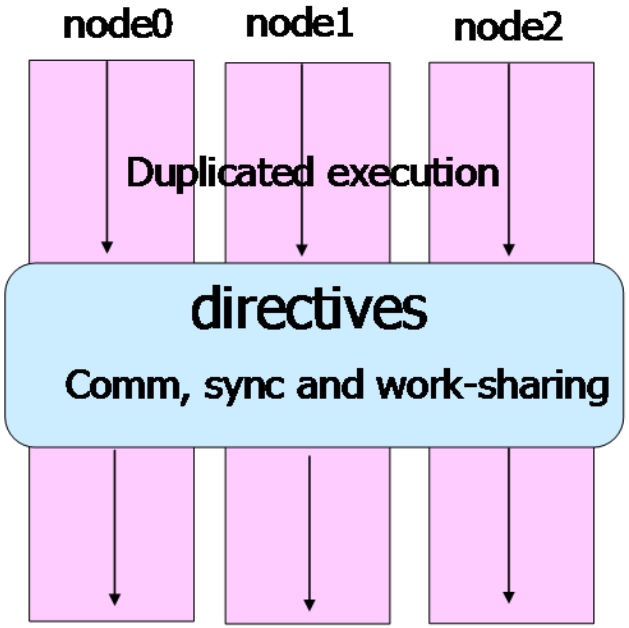
Performance-awareness: This is one of the most important lessons for the design of XcalableMP

XcalableMP

http://www.xcalablemp.org

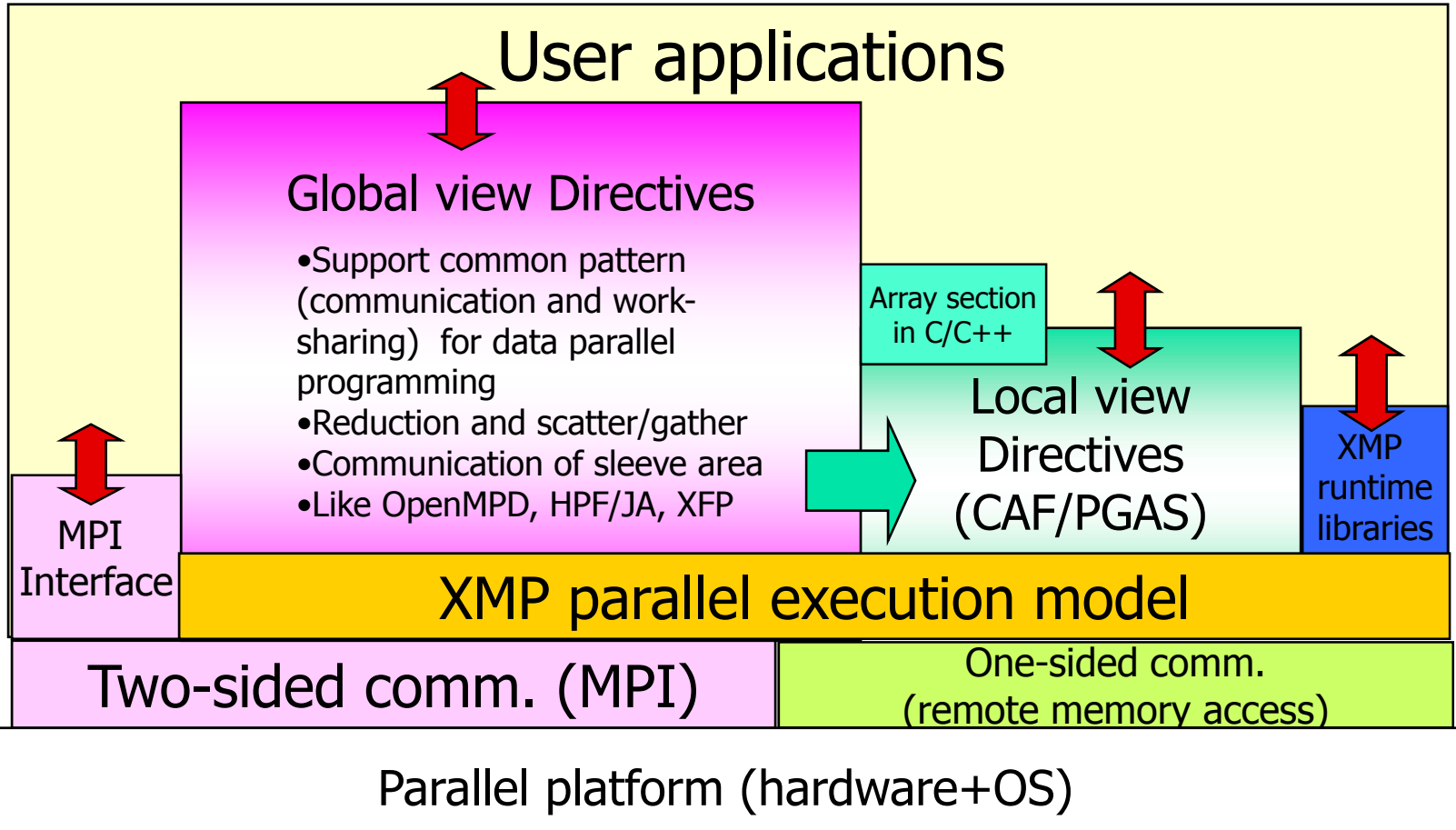
XcalableMP : directive-based language eXtension
for Scalable and performance-aware Parallel Programming

- **Directive-based language extensions** for familiar languages F90/C/C++
  - To reduce code-rewriting and educational costs.
- **"Scalable" for Distributed Memory Programming**
  - SPMD as a basic execution model
  - A thread starts execution in each node independently (as in MPI) .
  - Duplicated execution if no directive specified.
  - MIMD for Task parallelism
- **"performance-aware" for explicit communication and synchronization.**
  - Work-sharing and communication occurs when directives are encountered
  - All actions are taken by directives for being "easy-to-understand" in performance tuning (different from HPF)



# Overview of XcalableMP

- XMP supports typical parallelization based on the **data parallel paradigm** and work mapping under "**global view**“
  - An original sequential code can be parallelized with **directives**, like OpenMP.
- XMP also includes CAF-like PGAS (Partitioned Global Address Space) feature as "**local view**" programming.



# 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][*] 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 sharing and data synchronization

# The same code written in MPI

```
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();
}
```

# Nodes, templates and data/loop distributions

- Idea inherited from HPF
- Node is an abstraction of processor and memory in distributed memory environment, declared by node directive.

```
#pragma xmp nodes p(32)
#pragma xmp nodes p(*)
```

- Template is used as a dummy array distributed on nodes

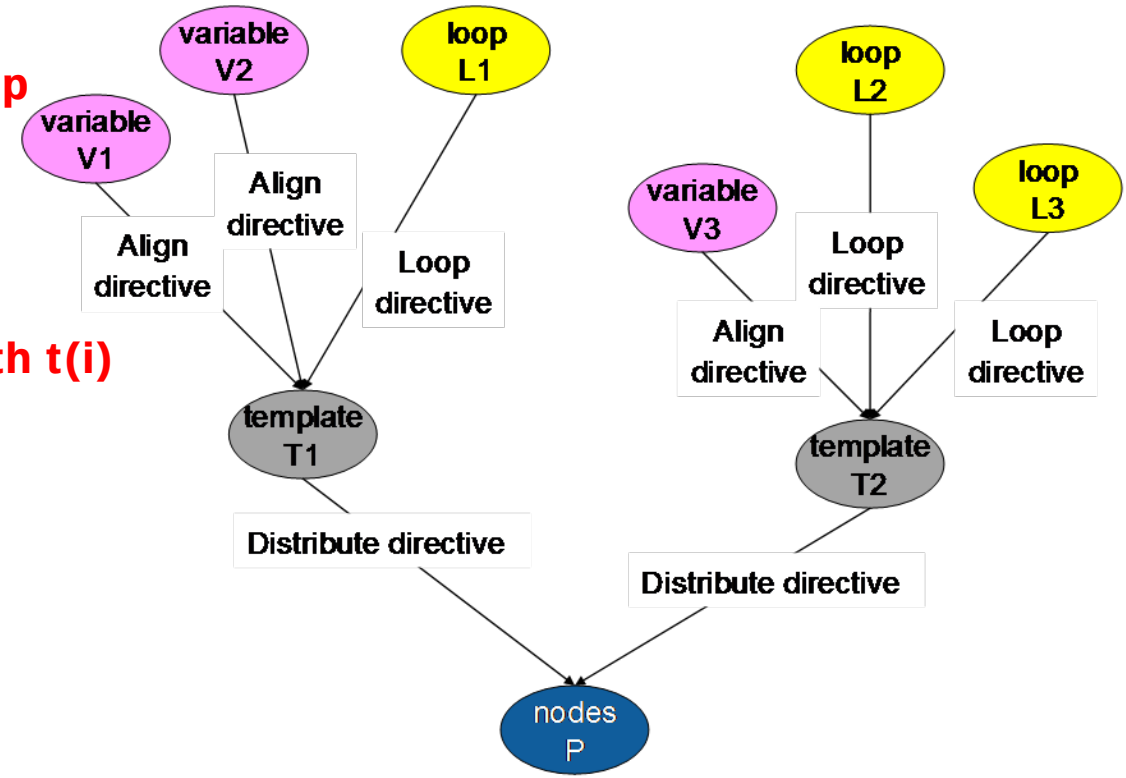
```
#pragma xmp template t(100)
#pragma distribute t(block) onto p
```

- A global data is aligned to the template

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

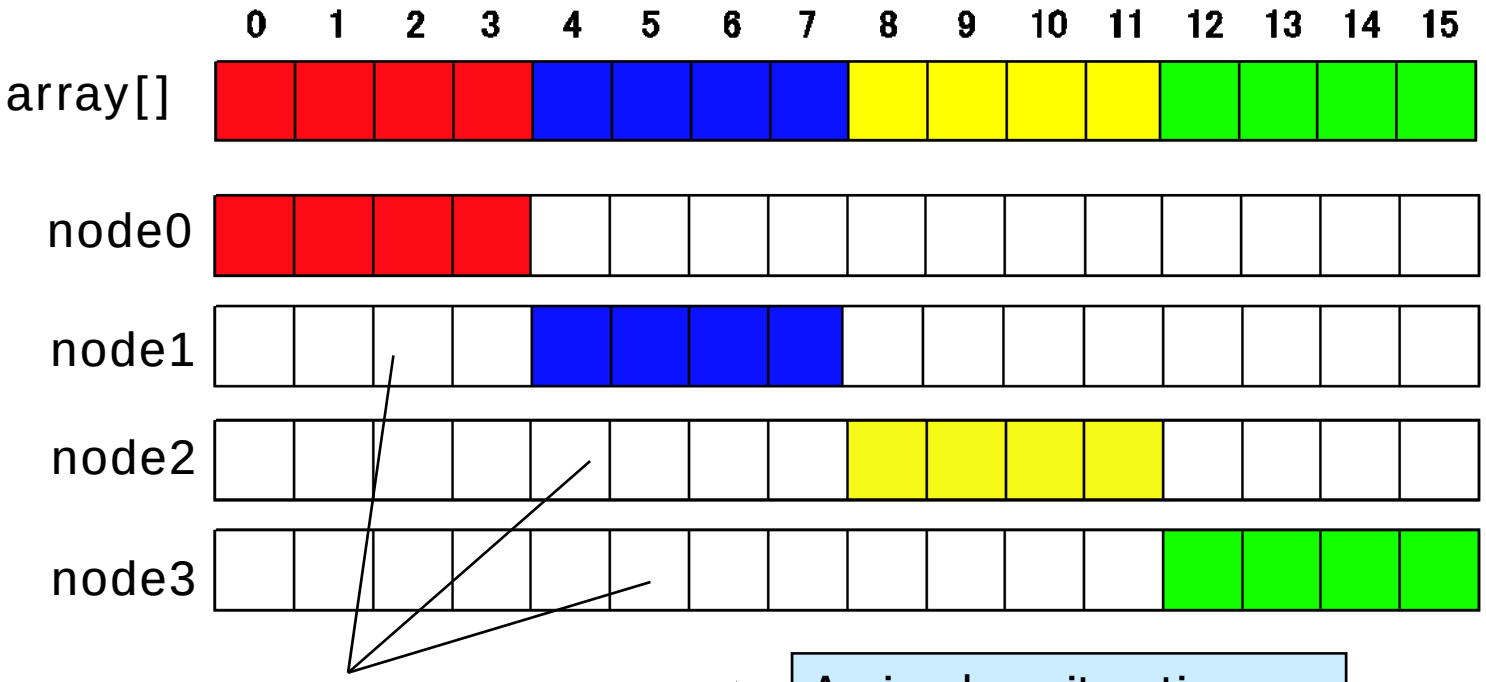
- Loop iteration must also be aligned to the template by on-clause.

```
#pragma xmp loop on t(i)
```



# Array data distribution

- The following directives specify a data distribution among nodes
  - #pragma xmp nodes p(*)
  - #pragma xmp template T(0:15)
  - #pragma xmp distribute T(block) on p
  - #pragma xmp align array[i] with T(i)




**Reference to assigned to other nodes may causes error!!**

Assign loop iteration as to compute own data

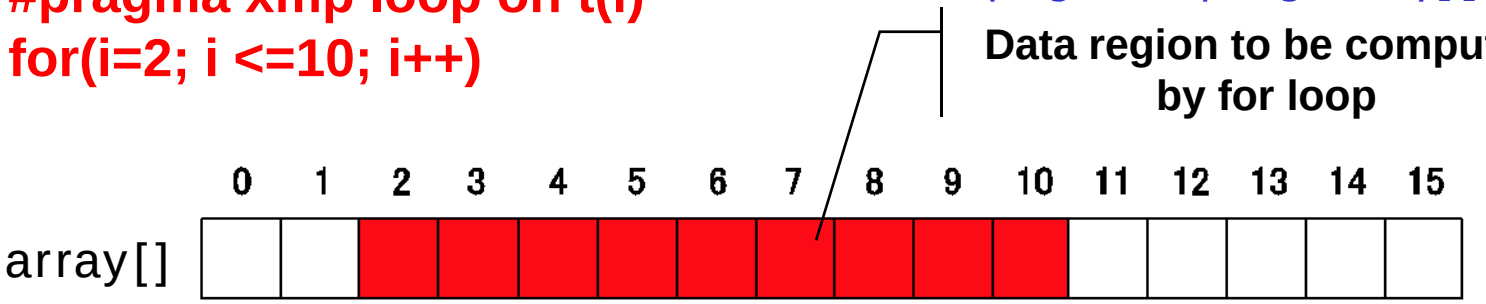
Communicate data between other nodes

# Parallel Execution of “for” loop

- Execute for loop to compute on array

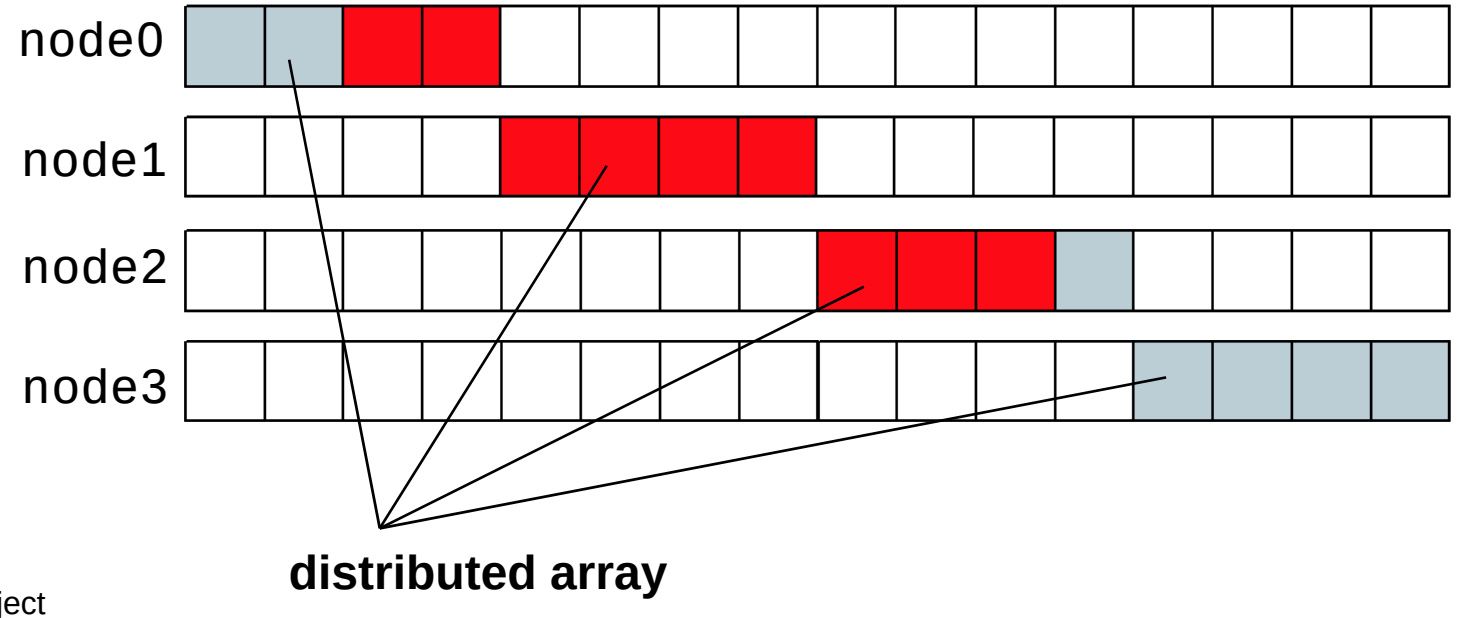
```
#pragma xmp nodes p(*)
#pragma xmp template T(0:15)
#pragma xmp distributed T(block) onto p
#pragma xmp align array[i] with T(i)
```

```
#pragma xmp loop on t(i)
for(i=2; i <=10; i++)
```




**Execute “for” loop in parallel with affinity to array distribution by on-clause:**

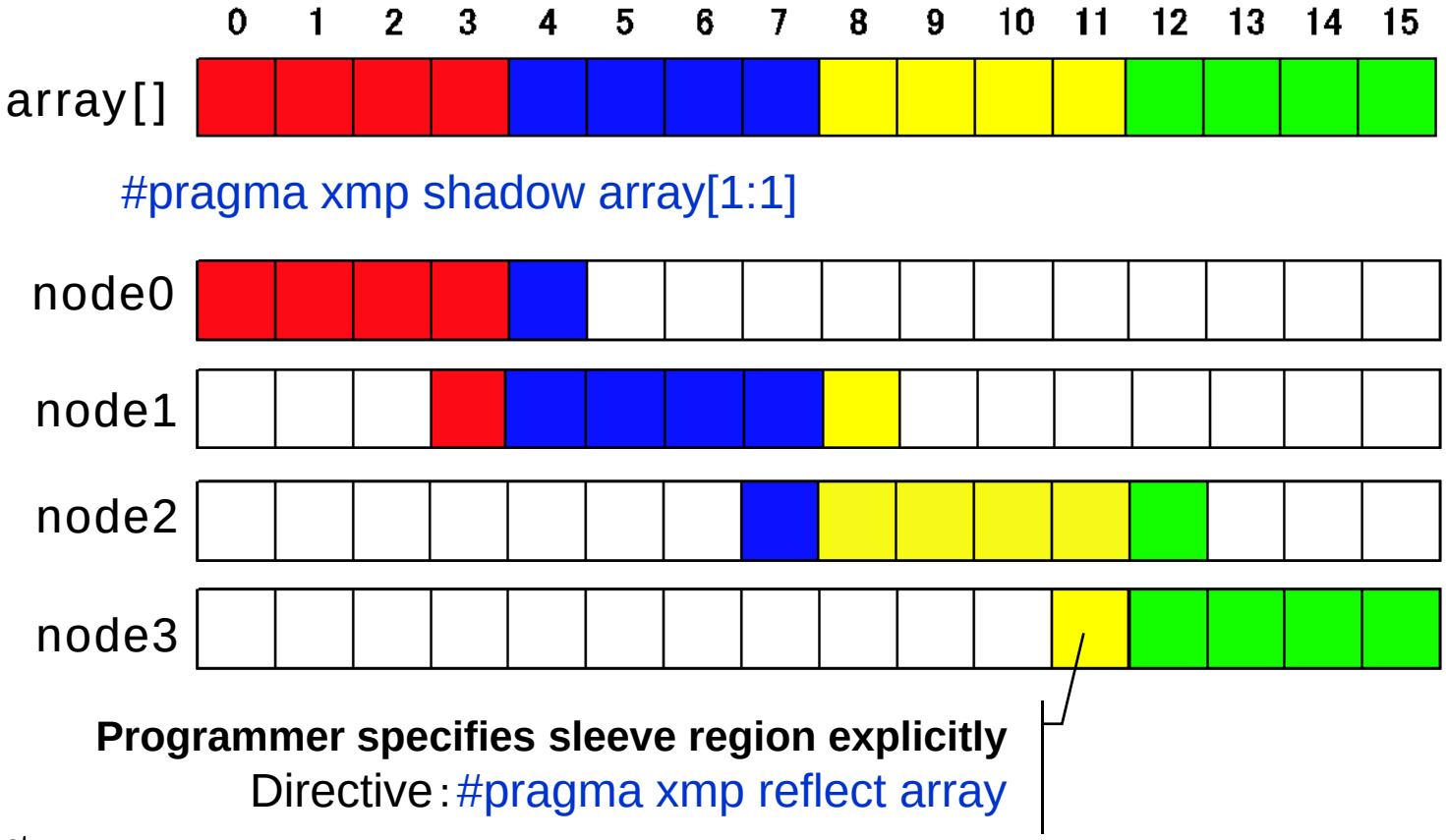
```
#pragma xmp loop on t(i)
```



# 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)

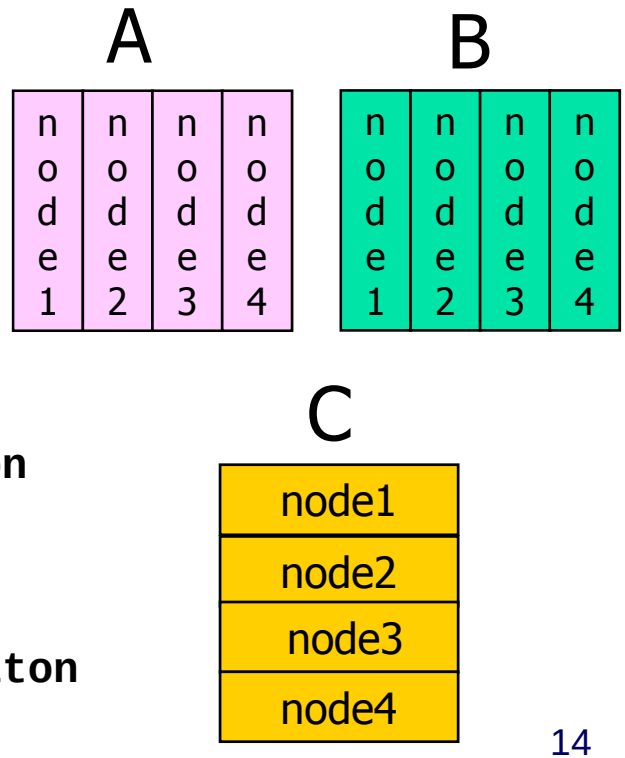


# gmove directive

- 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)
!$xmp distributed t(block) to p
real A(N,N),B(N,N),C(N,N)
!$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 operaiton
```



# XcalableMP global view directives

- Task parallelism
  - #pragma xmp task on *node-set*
  - Execution only master node
    - #pragma xmp task on master
- Barrier/Reduction
  - #pragma xmp reduction (*op: var*) *[on … ]*
  - #pragma xmp barrier *[on … ]*
- Broadcast from master node
  - #pragma xmp bcast *var [on … ]*

# XcalableMP Local view directives

- XcalableMP also includes CAF-like PGAS (Partitioned Global Address Space) feature as "**local view**" programming.
  - The basic execution model of XcalableMP is SPMD
    - Each node executes the program independently on local data if no directive
  - We adopt Co-Array as our PGAS feature.
  - In C language, we propose array section construct.
  - Can be useful to optimize the communication
- Support alias Global view to Local view

**Array section in C**

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

A[5:9] = B[0:4];
```

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

# Performance of HPCC Benchmark

- HPC Challenge Benchmark Class2
  - Class 2: Most Productivity
    - Class1:Best Performance
  - Most "elegant" implementation of four or more of the HPC Challenge benchmarks with special emphasis being placed on:
    - STREAM
    - Random Access
    - HPL
    - FFT
  - in SC09, Awards were given to IBM(X10 and UPC) on performance, to Cary (Chapel) on code elegance.

Selected as a finalist
of SC09 HPCC Class2

# Submission

- XMP/C: a prototype compiler was implemented
  - supports basic functions for data parallelism
  - (Some parts ware compiled by hand)

- In this submission, we focus on programmability of XcalableMP
  - STREAM, RandomAccess, HPL, FFT are parallelized by XMP

T2K OpenSupercomputer – Tsukuba System (2to32nodes)

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

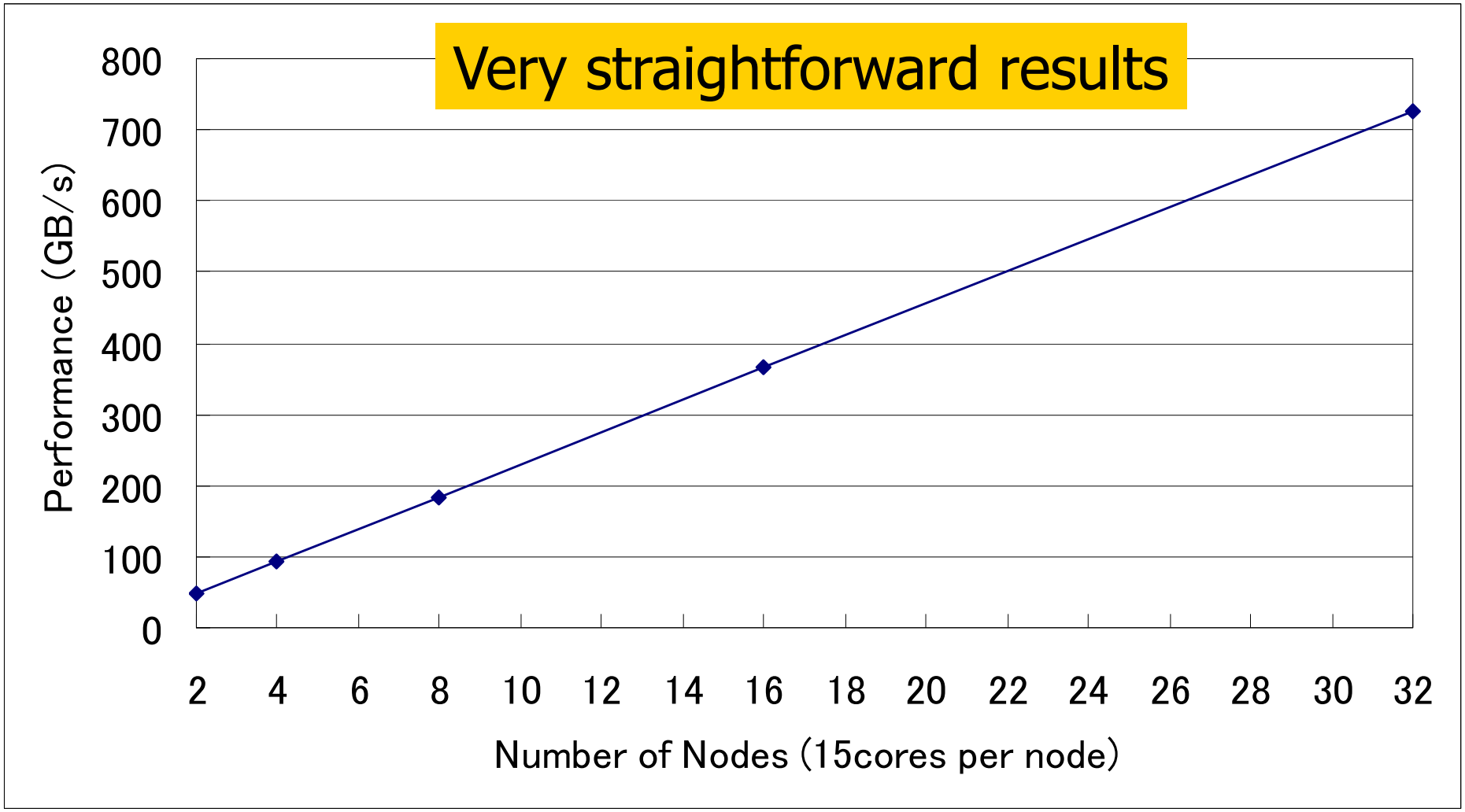
# HPCC Benchmark1: STREAM

- This benchmark is used to measure the memory bandwidth
- 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

- Lines Of Code: 98



# HPCC Benchmark2: Random Access

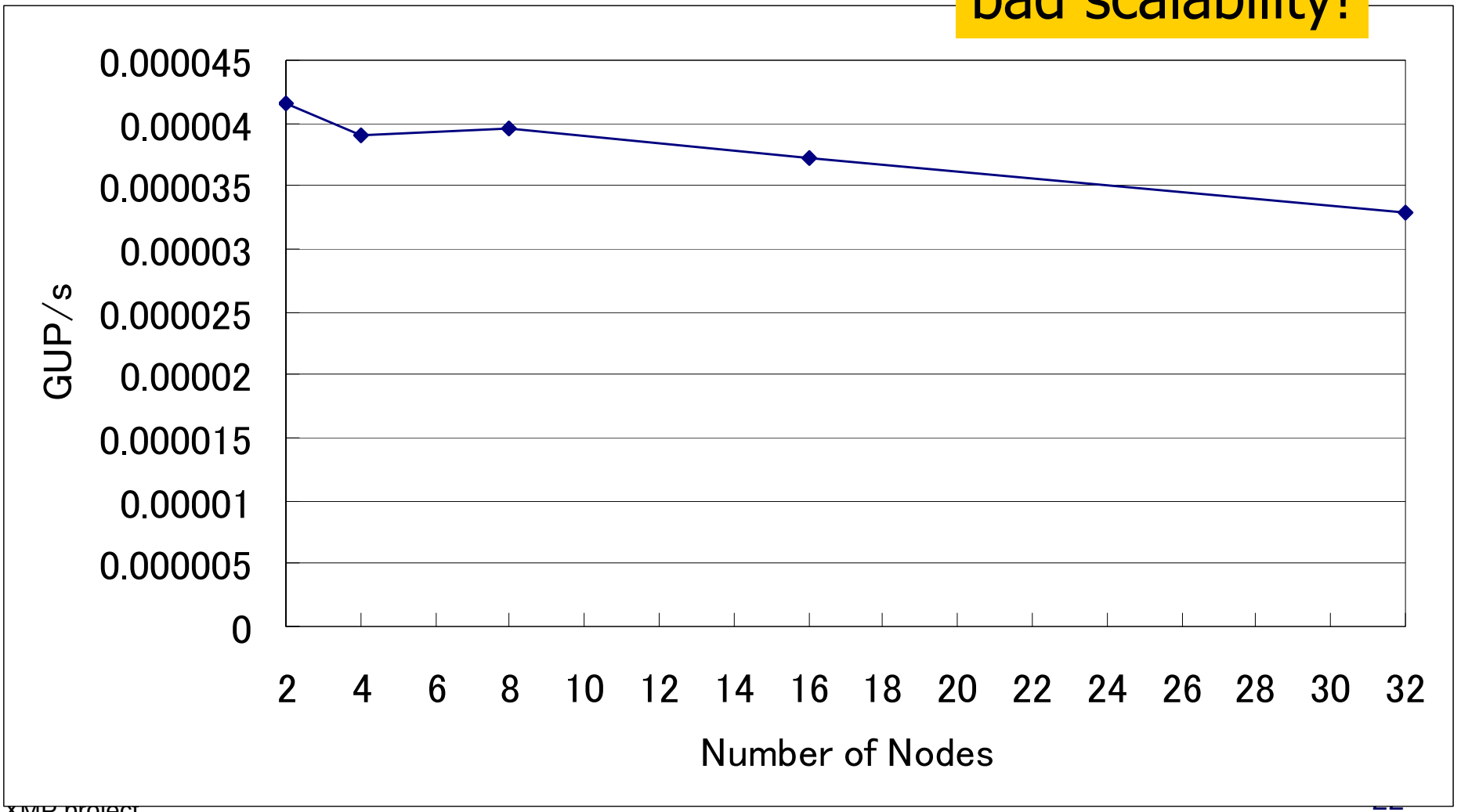
- Local view programming with co-array

```
#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

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

bad scalability!



# HPCC Benchmark3: HPL

- Parallelized in global view
- Matrix/vectors are distributed in cyclic manner in one dimension. No two dimensional data distribution
- Using **gmove** to exchange columns for pivot exchange

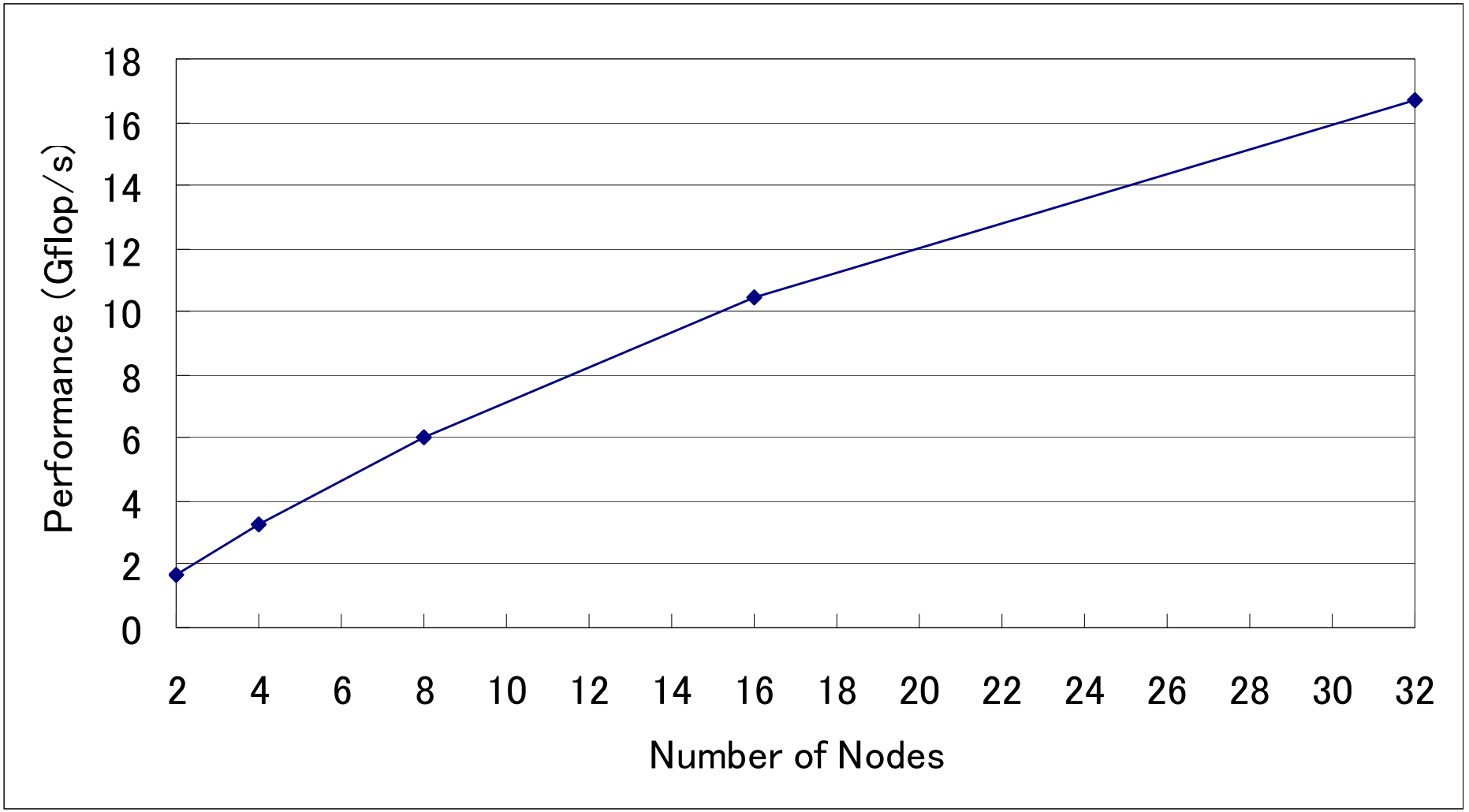
*dgefa function:*

```
#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

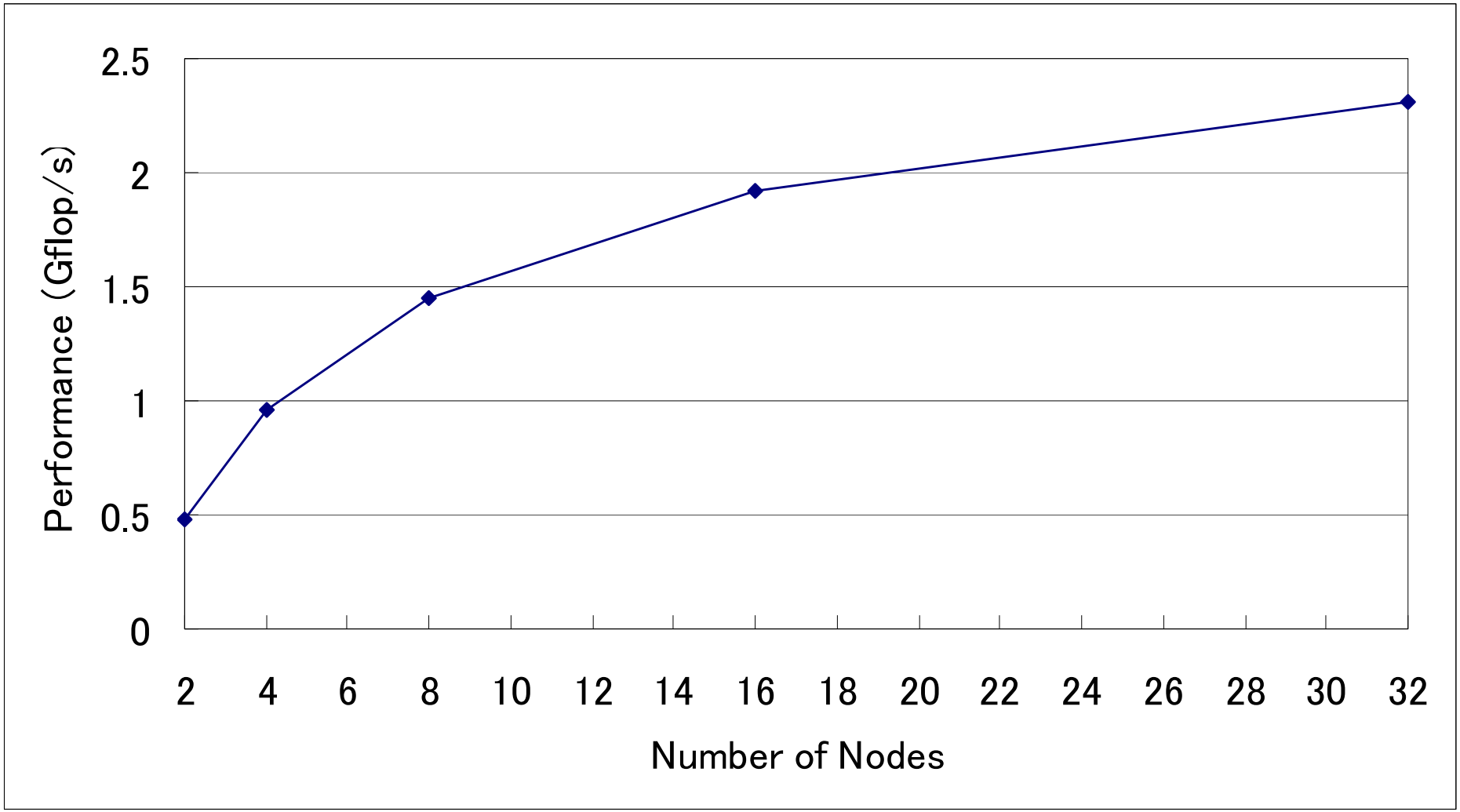
- Parallelized in global view
- Using six-step FFT algorithm
  - Matrix transpose is a key operation.
- Matrix transpose using **gmove**

```
#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



# Short Summary

http://www.xcalablemp.org

- Preliminary performance report of HPCC Class2 XcalableMP results
  - We found XMP can be a good solution to describe HPCC benchmark.
  - Performance looks reasonable, but much performance tuning is required
    - One-sided communication for Random Access, 2-D block distribution in HPL
  - More experience is needed such as NPB, ….
- XcalableMP project: status and schedule
  - A draft of XcalableMP specification 0.7
    - http://www.xcalablemp.org/xmp-spec-0.7.pdf
  - 2Q?/10 α release, C language version compiler
  - Fortran version compiler before SC10
- Issues for the next
  - Multicores (SMP) Cluster and Hybrid programming with OpenMP
  - IO
  - GPGPU?, Manycore?, Fault tolerant?, Others …