XcalableMP

XcalableMP Implementation and Performance of NAS Parallel Benchmarks

Mitsuhisa Sato

Masahiro Nakao, Jinpil Lee and Taisuke Boku

University of Tsukuba, Japan

hpcs lab

High Performance Computing System

What's XcalableMP?

- XcalableMP (XMP for short) is:
  - A programming model and language for distributed memory , proposed by XMP WG
  - http://www.xcalablemp.org

- XcalableMP Specification Working Group (XMP WG)
  - XMP WG is a special interest group, which organized to make a draft on "petascale" parallel language.
  - Started from December 2007, the meeting is held about once in every month.
    - Mainly active in Japan, but open for everybody.

- XMP WG Members (the list of initial 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), (many HPF developers!)

- Funding for development
  - e-science project : "Seamless and Highly-productive Parallel Programming Environment for High-performance computing" project funded by MEXT,Japan
    - Project PI: Yutaka Ishiakwa, co-PI: Sato and Nakashima(Kyoto), PO: Prof. Oyanagi
    - Project Period: 2008/Oct to 2012/Mar (3.5 years)

Agenda

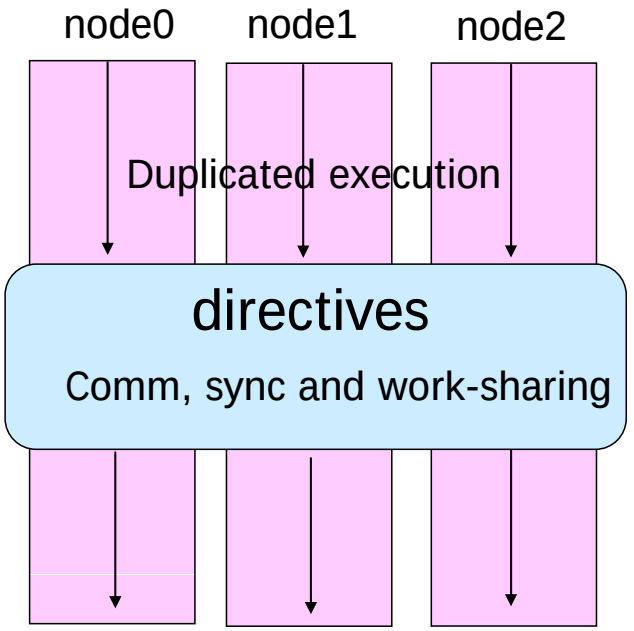
- XcalableMP : directive-based language eXtension for Scalable and performance-aware Parallel Programming
  - Concept and model
  - directives
  - Some examples
- XMP implementation of Nas Parallel Benchmark
  - ES, IS, CG (1-D, 2-D)
  - Preliminary performance reports

XcalableMP

http://www.xcalablemp.org

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

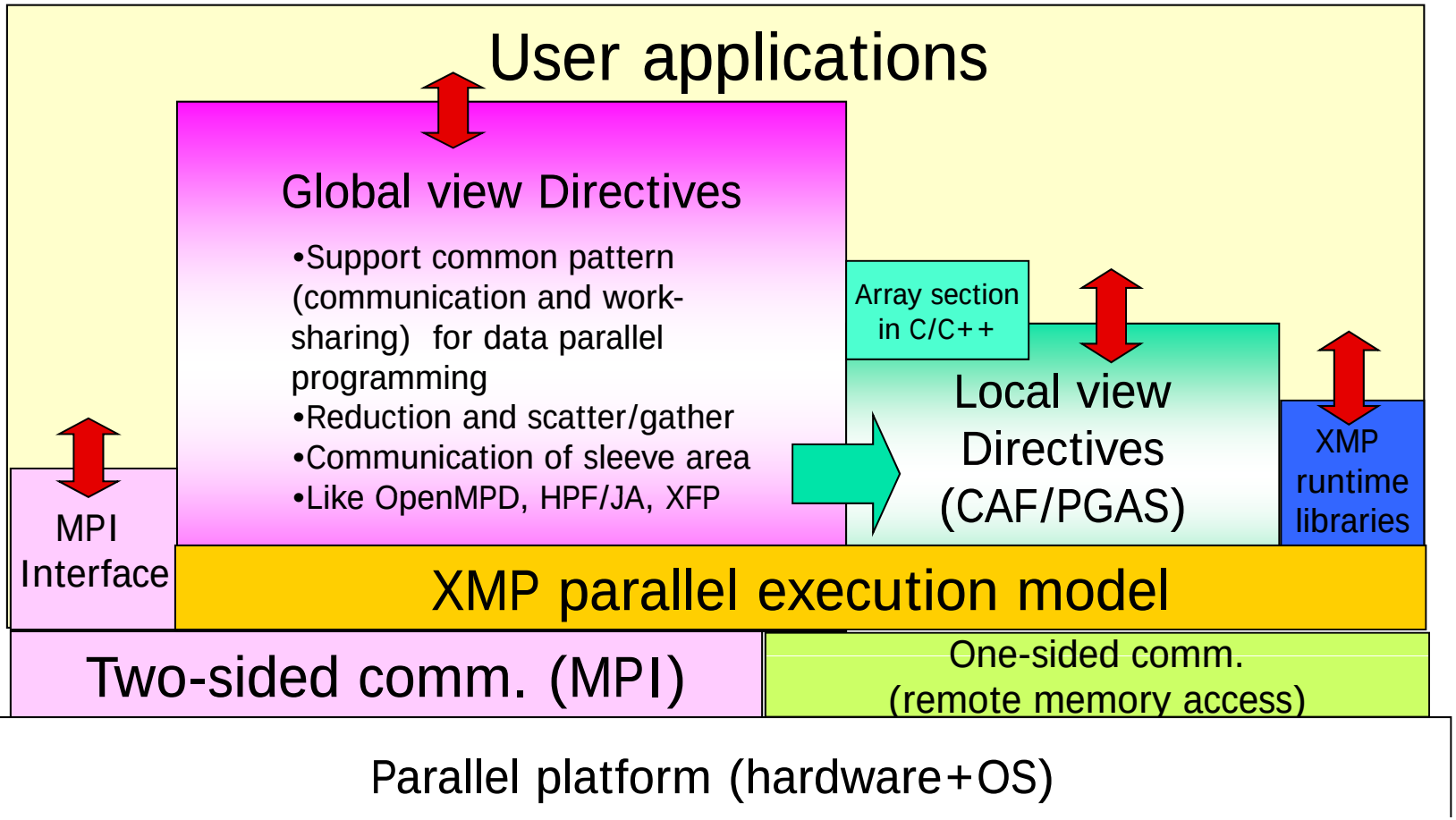
- **Directive-based language extensions** for familiar languages F90/C (C++)
  - To reduce code-rewriting and educational costs.



- **"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 sharing 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

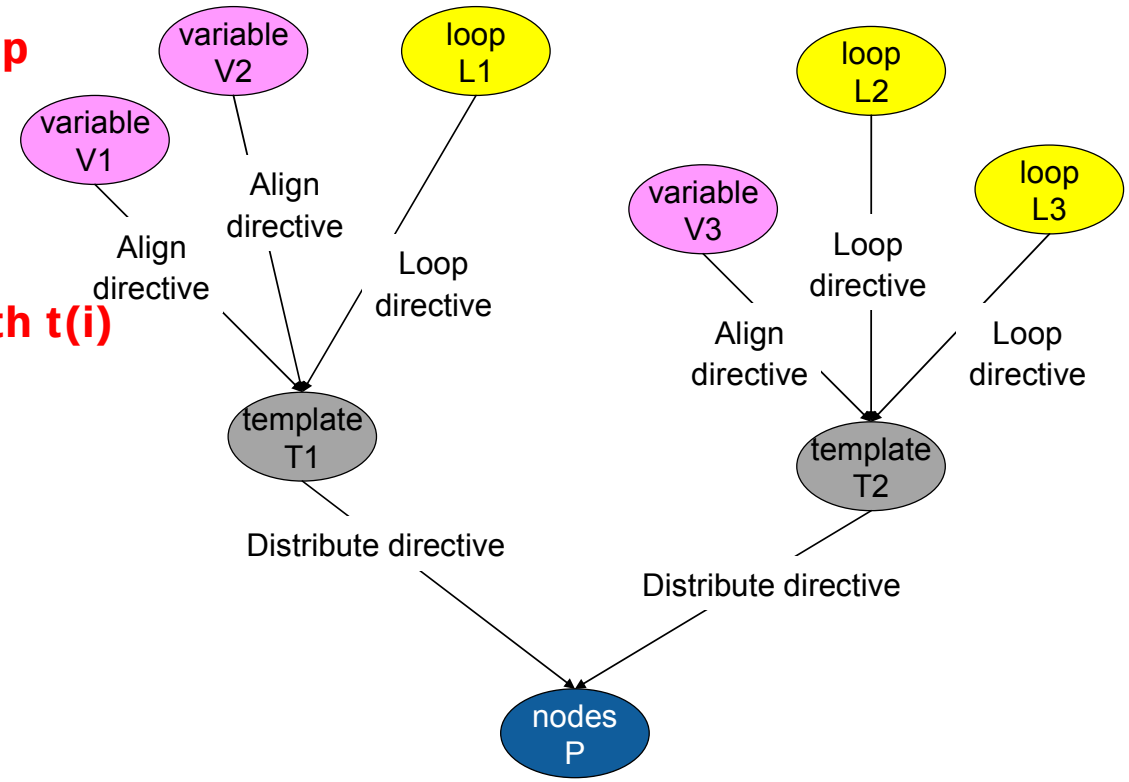
```
#pragma xmp template t(100)
#pragma distribute t(block) onto p
```

- A global data is aligned to the template

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

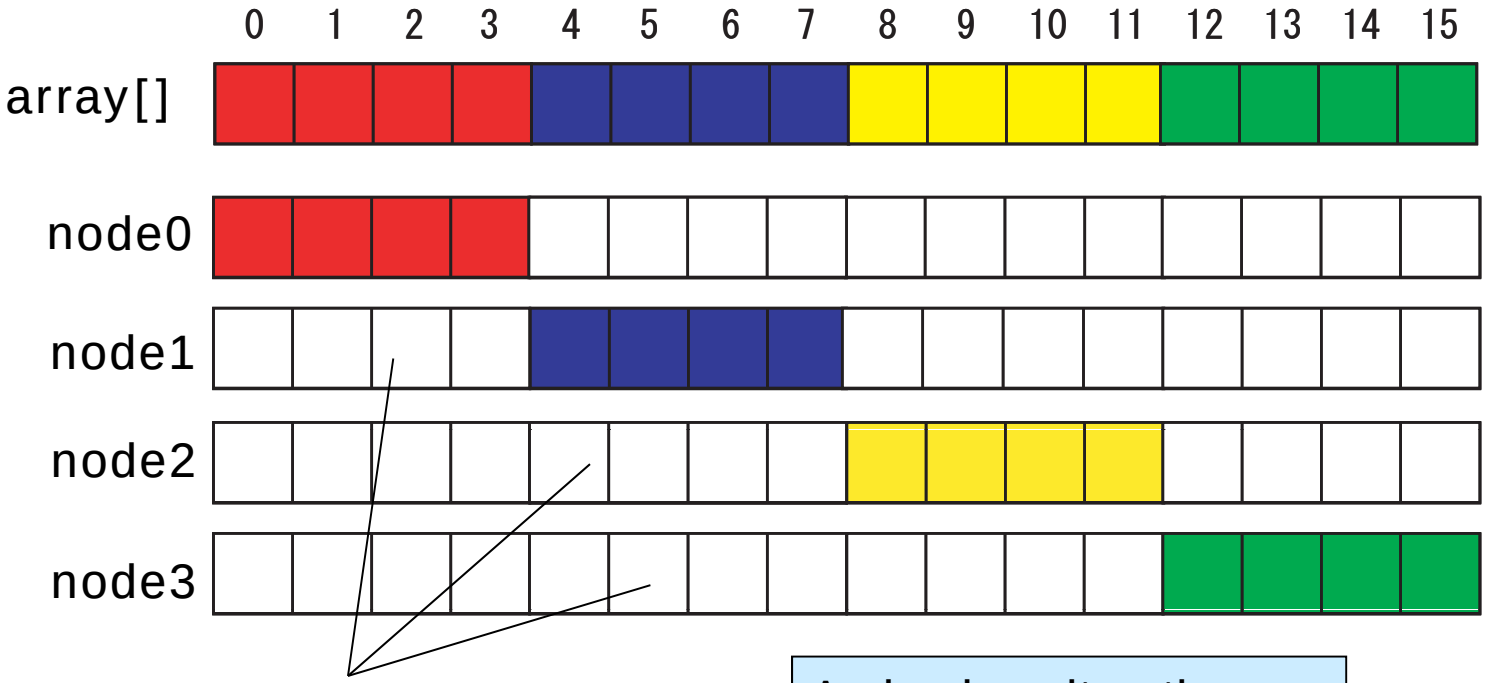
- Loop iteration must also be aligned to the template by on-clause.

```
#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

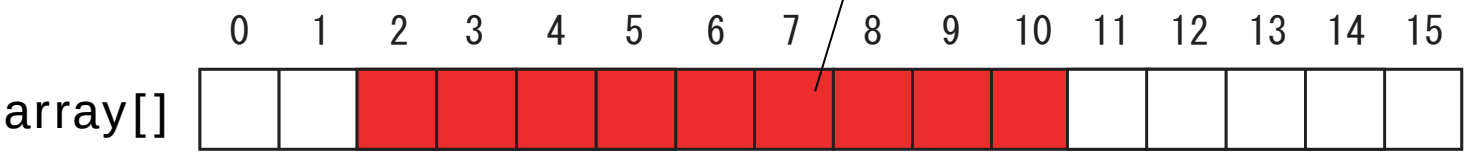
Parallel Execution of “for” loop

- 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)
```

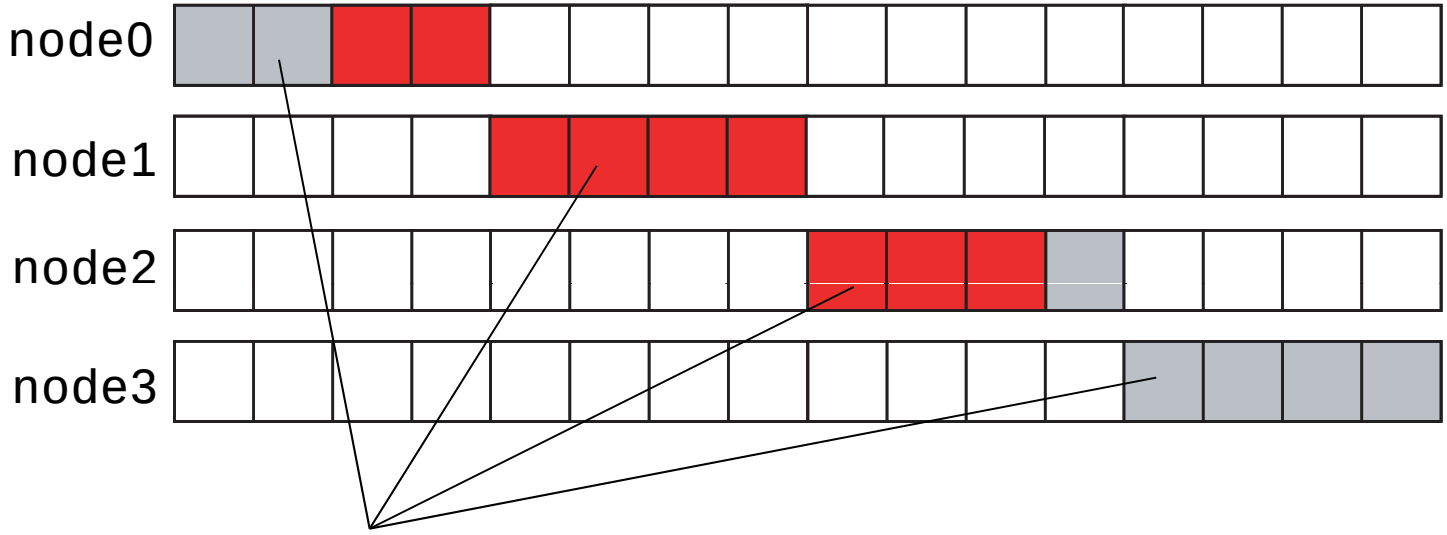
```
#pragma xmp loop on t(i)
for(i=2; i <=10; i++)
```

Data region to be computed by for loop



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

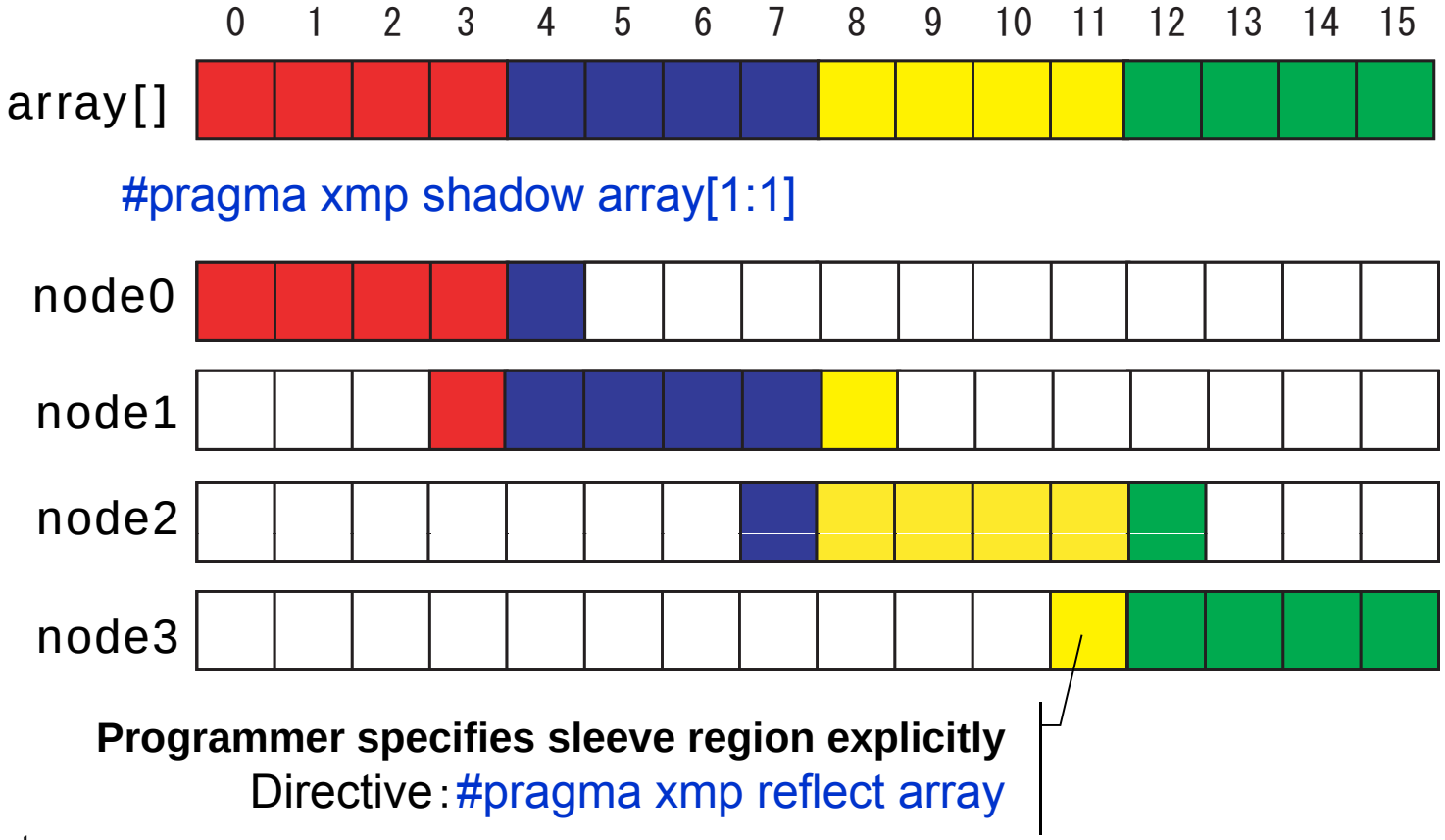
```
#pragma xmp loop on t(i)
```



Data synchronization of array (shadow)

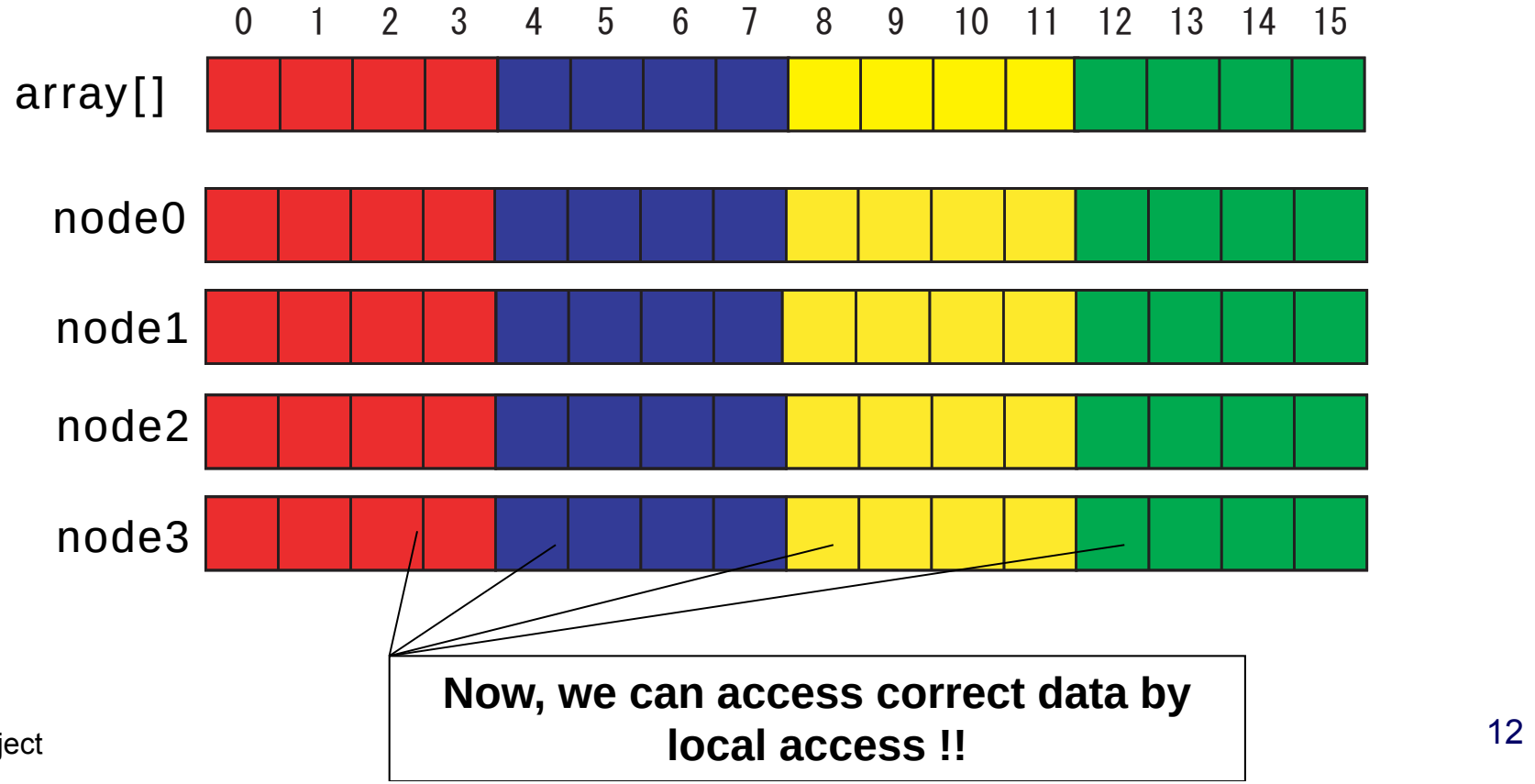
- Exchange data only on “shadow” (sleeve) region
  - If neighbor data is required to communicate, then only sleeve area can be considered.
  - example：b[i] = array[i-1] + array[i+1]

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



Data synchronization of array (full shadow)

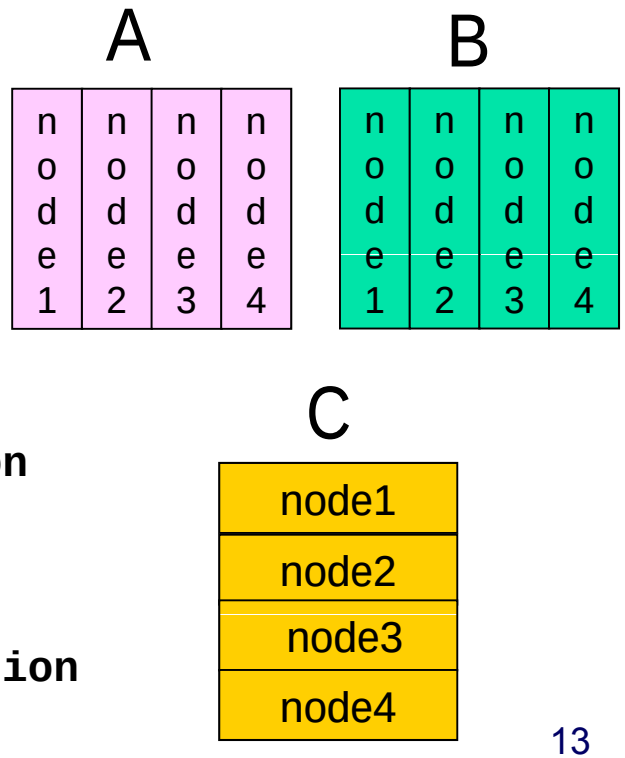
- Full shadow specifies whole data replicated in all nodes
  - #pragma xmp shadow array[*]
- reflect operation to distribute data to every nodes
  - #pragma reflect array
  - Execute communication to get data assigned to other nodes
  - Most easy way to synchronize → But, communication is expensive!



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 Global view directives

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

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];
```

Co-array notation in C

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

Experience with NPB in XcalableMP

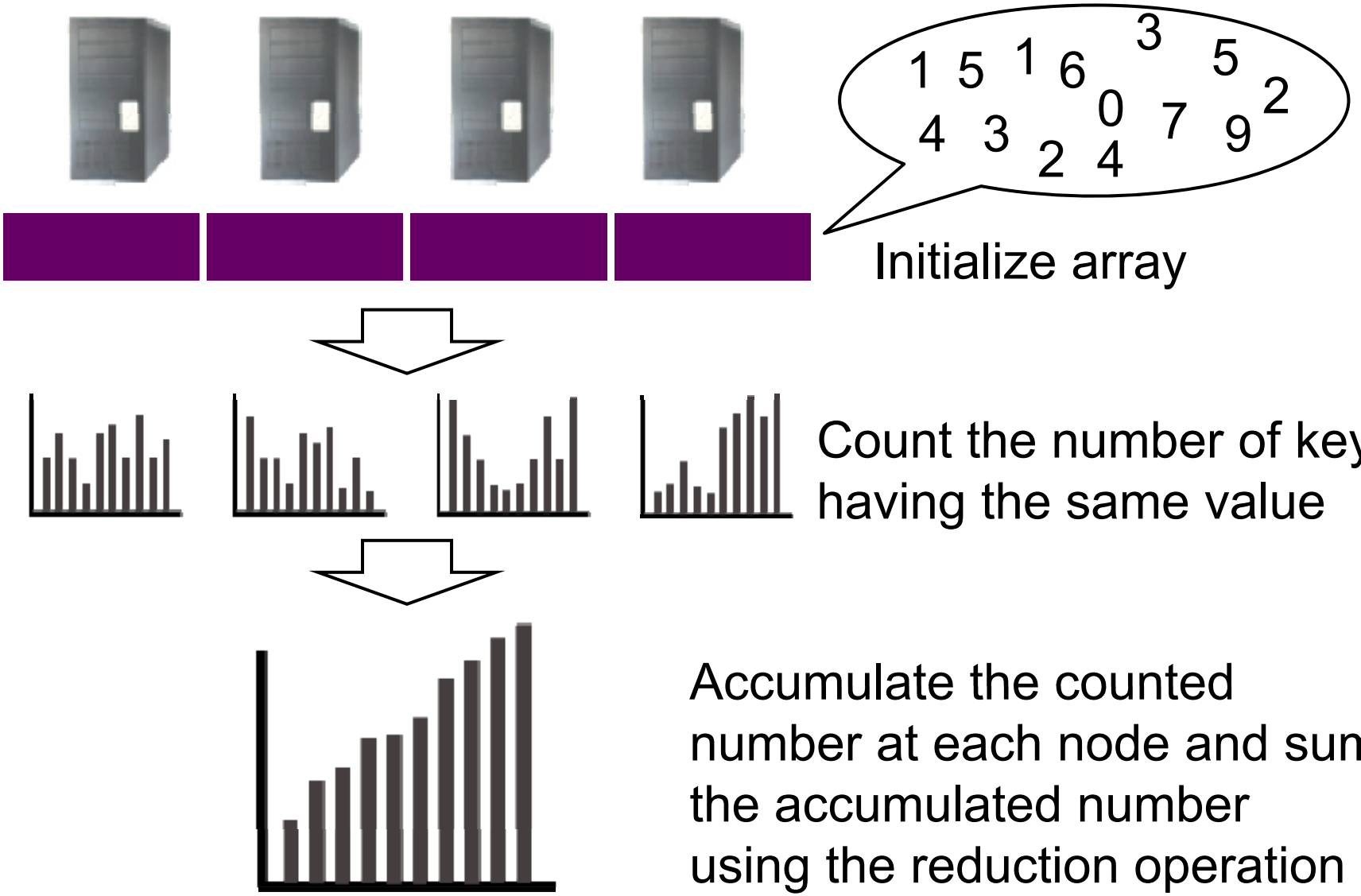
- The following three benchmarks are selected for the XMP benchmark
  - EP
  - IS
    - with a histogram (buckets)
    - without a histogram (buckets)
  - CG
    - one-dimensional parallelization
    - two-dimensional parallelization
- Check
  - Programmability/Expressiveness (How to write programs)
  - Performance (How fast the written programs run)

NPB-EP

```
#pragma xmp nodes p(*)
#pragma xmp template t(1:NN)
#pragma xmp distribute t(brock) onto p
…
#pragma xmp loop on t(k)
for(k=1; k<=NN; k++){

      /* pseudorandom number generation*/

}
```

Parallelized by "for statement"

NPB-IS without a histogram



NPB-IS without a histogram

key_array[] is a distributed array.
Prv_buff1 is local

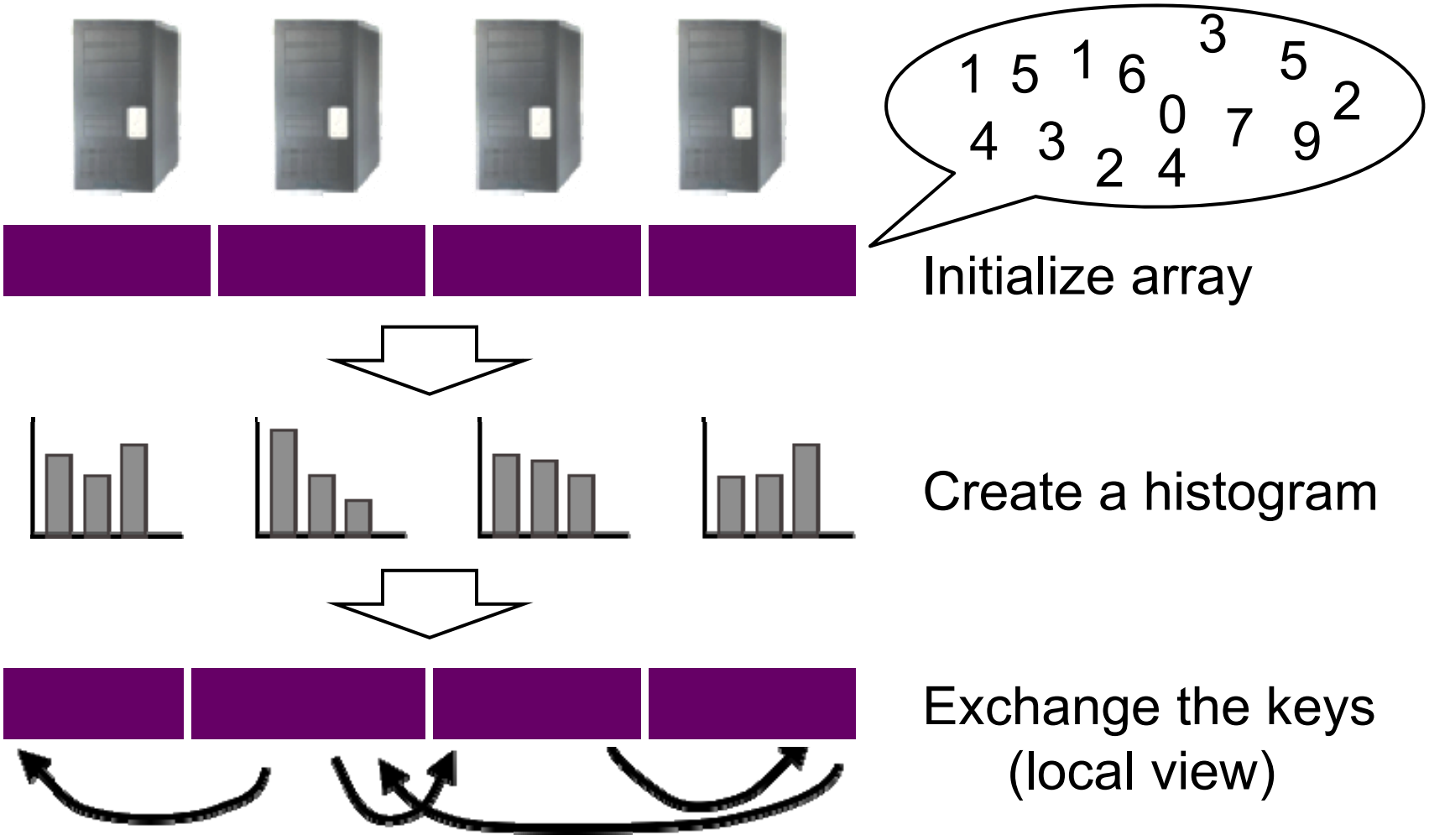
```
#pragma xmp loop on t(i)
for( i=0; i<NUM_KEYS; i++ ){
    key_buff2[i] = key_array[i];
    prv_buff1[key_buff2[i]]++;
}

for( i=0; i<MAX_KEY-1; i++ )
    prv_buff1[i+1] += prv_buff1[i];

#pragma xmp reduction(+:prv_buff1)
```

Count the number of keys having the same value

accumulate

reduction

NPB-IS with a histogram



If the value of the key is small,
the key is moved to left node.

NPB-IS with a histogram

key_array[] is distributed array.

```
#pragma xmp coarray key_buff2
…
#pragma xmp loop on t(i)
for(i=0; i<NUM_KEYS; i++)
    bucket_size[key_array[i] >> shift]++;
…
for(i=0;i<NUM_PROCS;i++)
    key_buff2[a[i]:b[i]]:[i] = key_buff1[c[i]:d[i]];
```

declare coarray

create a histogram

communication with coarray

Copy the range from c[i] to d[i] of key_buf2 to the range a[i] to b[i] of key_buff2 in proc [i] (equivalent to MPI_allgather_v)

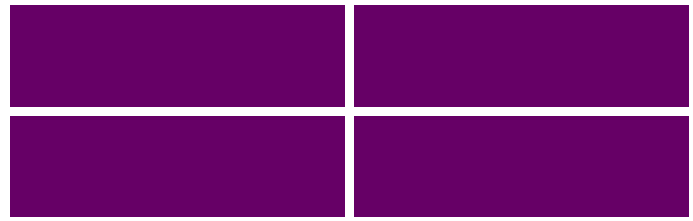
Implementation of NPB-CG

two-dimensional
sparse matrix

a[ ]

How do arrays are
distributed to each node ?

Conjugate Gradient



two-dimensional
parallelization

template(1:N, 1:N)

one-dimensional
parallelization

template(1:N)

NPB-CG One-Dimensional Parallelization

XcalableMP

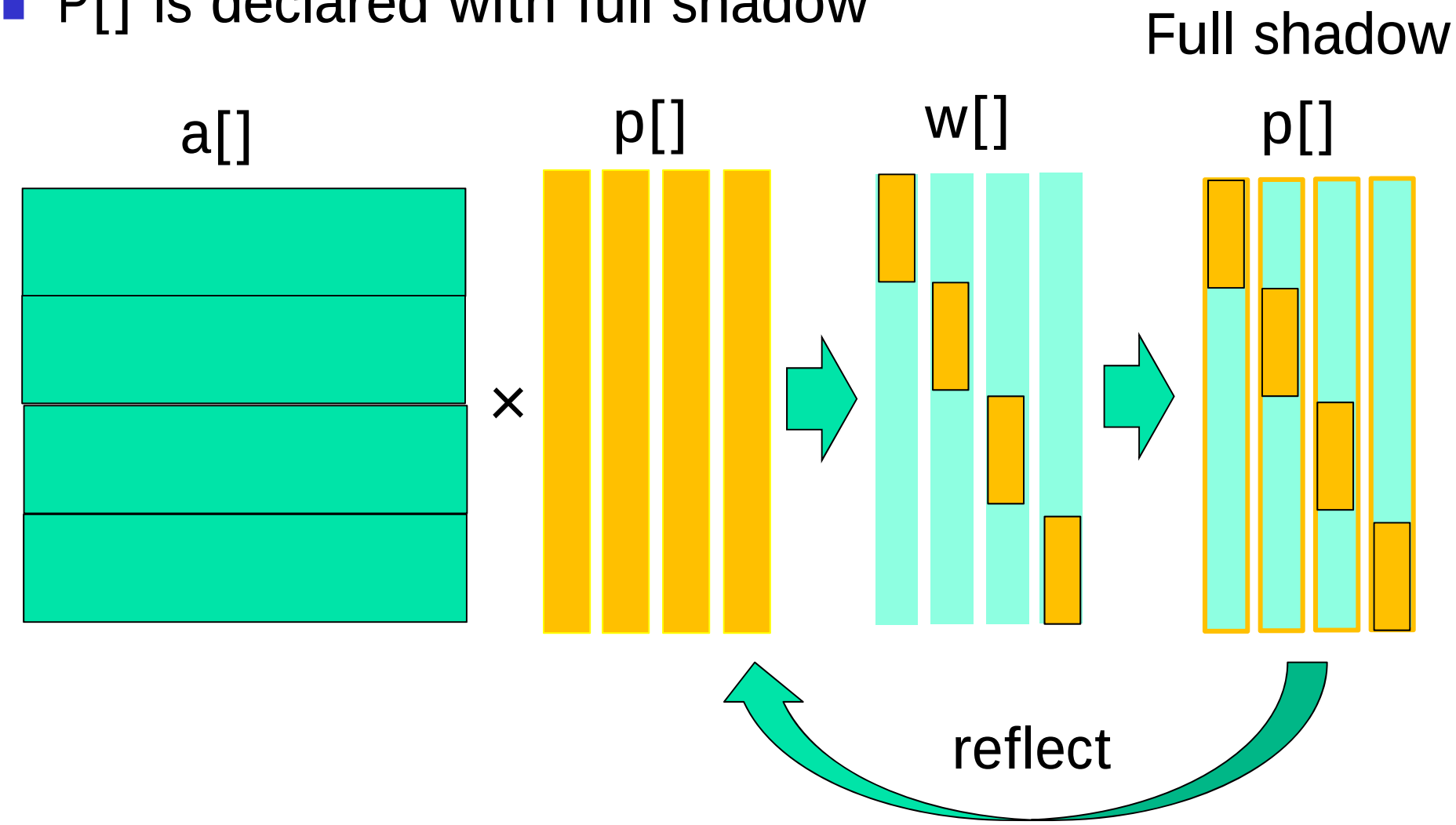
p[], q[], and w[] are distributed arrays.

```
#pragma xmp template t(0:N-1)
#pragma xmp distribute t(block) on proc
#pragma xmp align p[i],q[i],w[i]  with t(i)
#pragma xmp shadow p[*]
…
for( ….){
#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]; k++) {
          sum = sum + a[k]*p[colidx[k]];
     }
     w[j] = sum;
}
#pragma xmp loop on t(j)
for (j = 1; j <= lastrow-firstrow+1; j++)
     q[j] = w[j];
… update p with q and w ….
}
```

declare full shadow

Synchronization by reflect

- P[] is declared with full shadow

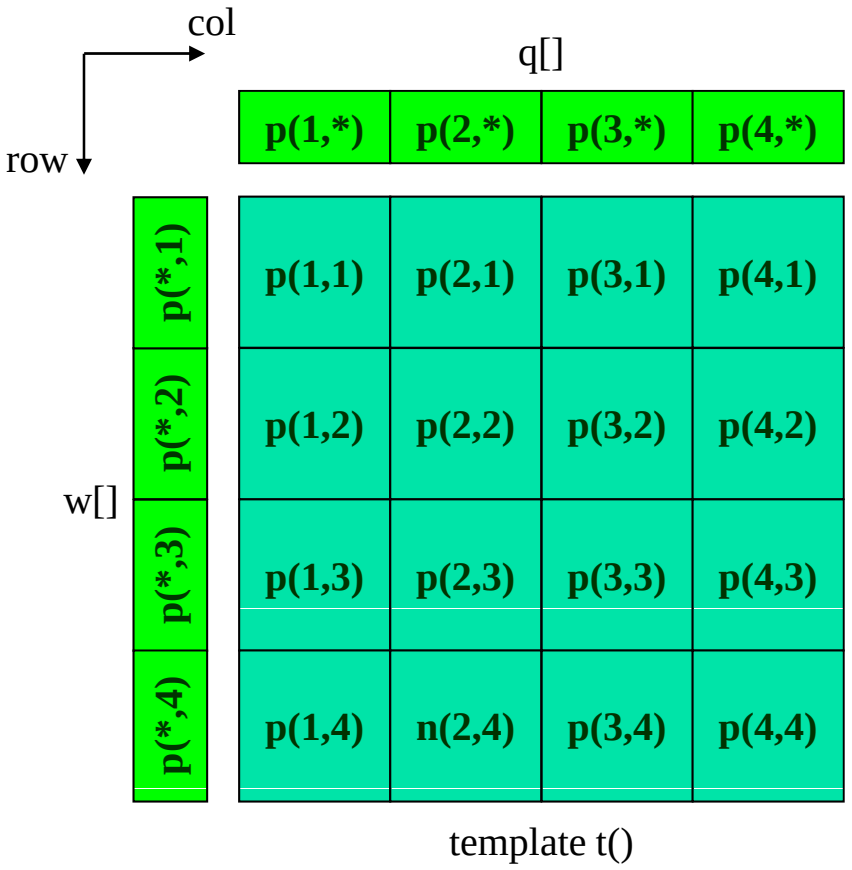


2D-Parallelization of NPB-CG (data distribution)

- Declaration of replicated Arrays

```
#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], z[na], p[na],
          q[na], r[na], w[na];

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

w is replicated at the first dimension of t, and distributed for the second dimension in block distribution.



- Matrix data **a[], rowstr[], colidx[]**
  1. **Declared as local arrays**
  2. **Arranged as to access each element locally.**

NPB-CG Two-Dimensional Parallelization

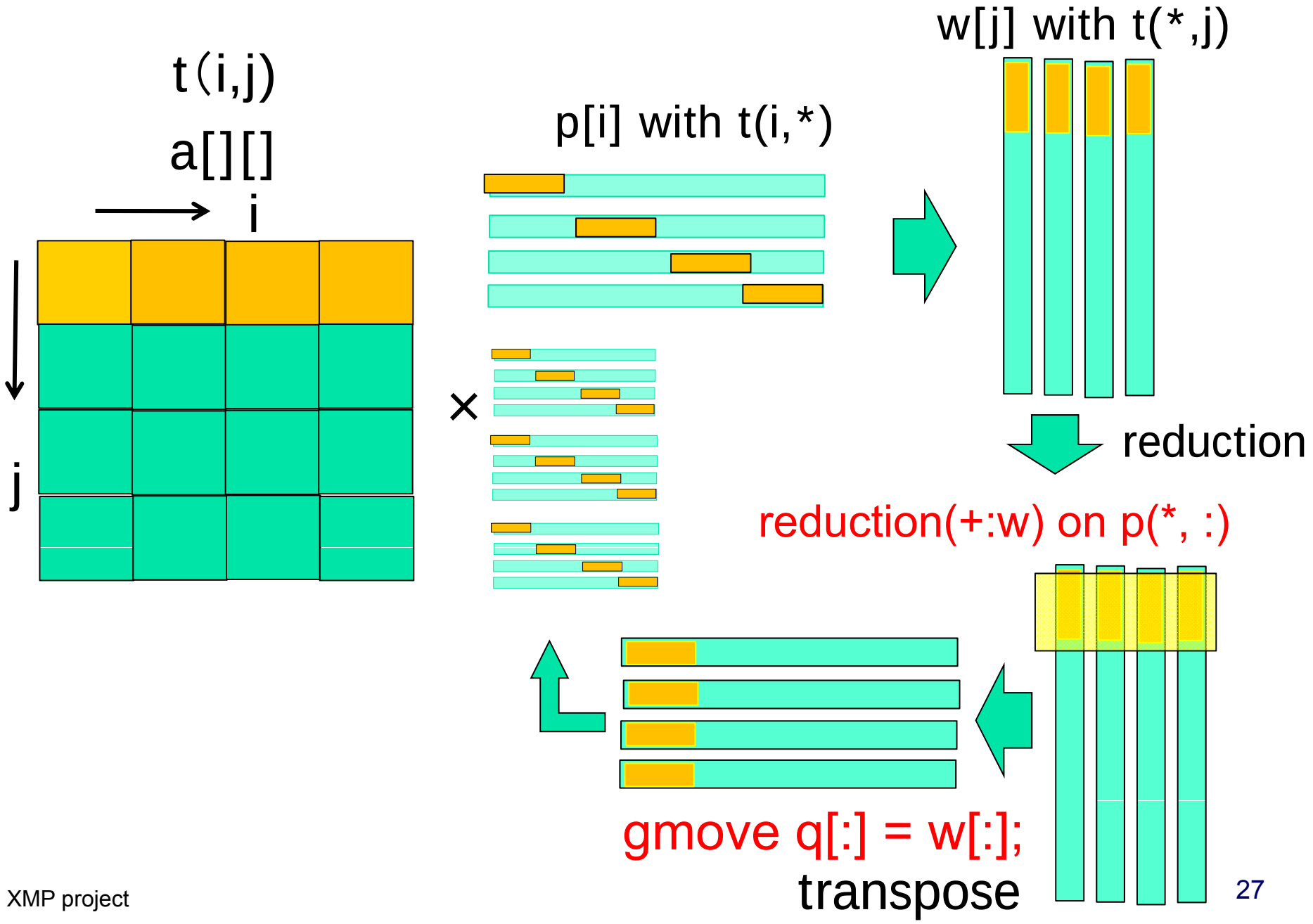
```
#pragma xmp template t(0:N-1,0:N-1)
#pragma xmp distribute t(block, block) on p
#pragma xmp align A[j][i] to t(i,j)
#pragma xmp align p[i] to t(i,*)
#pragma xmp align w[j] to t(*,j)
….
for(){
…
#pragma xmp loop on t(*, j)
for (j = 1; j <= lastrow-firstrow+1; j++) {
    sum = 0.0;
    for (k = rowstr[j]; k < rowstr[j+1]; k++) {
        sum = sum + a[k]*p[colidx[k]];
    }
    w[j] = sum;
}
#pragma xmp reduction(+:w) on p(*, :)
#pragma xmp gmove
q[:] = w[:];
….
….. Update p with q ….
}
```

p[], q[], and w[] are distributed arrays.

p[i], q[i] with t(i, *)
w[i] with t(*, i)

Reduction operation on replicated array

copy arrays with different distributions


t(i,j)
a[][]
i
j
p[i] with t(i,*)
×
w[j] with t(*,j)
reduction
reduction(+:w) on p(*, :)
gmove q[:] = w[:];
transpose

Performance Evaluation

XcalableMP

T2K Tsukuba System

AMD Opteron Quad 2.3GHz

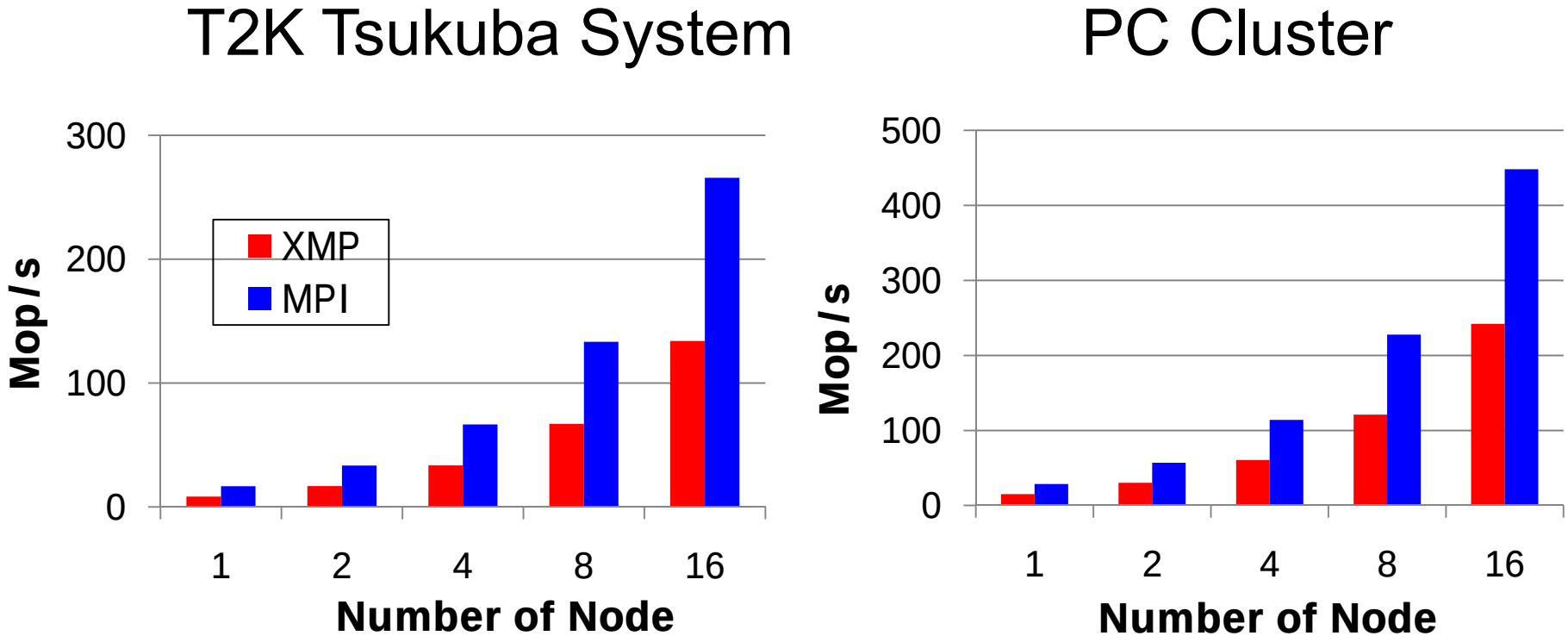
Infiniband DDR 4rails (8GB/s)

PC Cluster

Intel Core2 Quad 3GHz

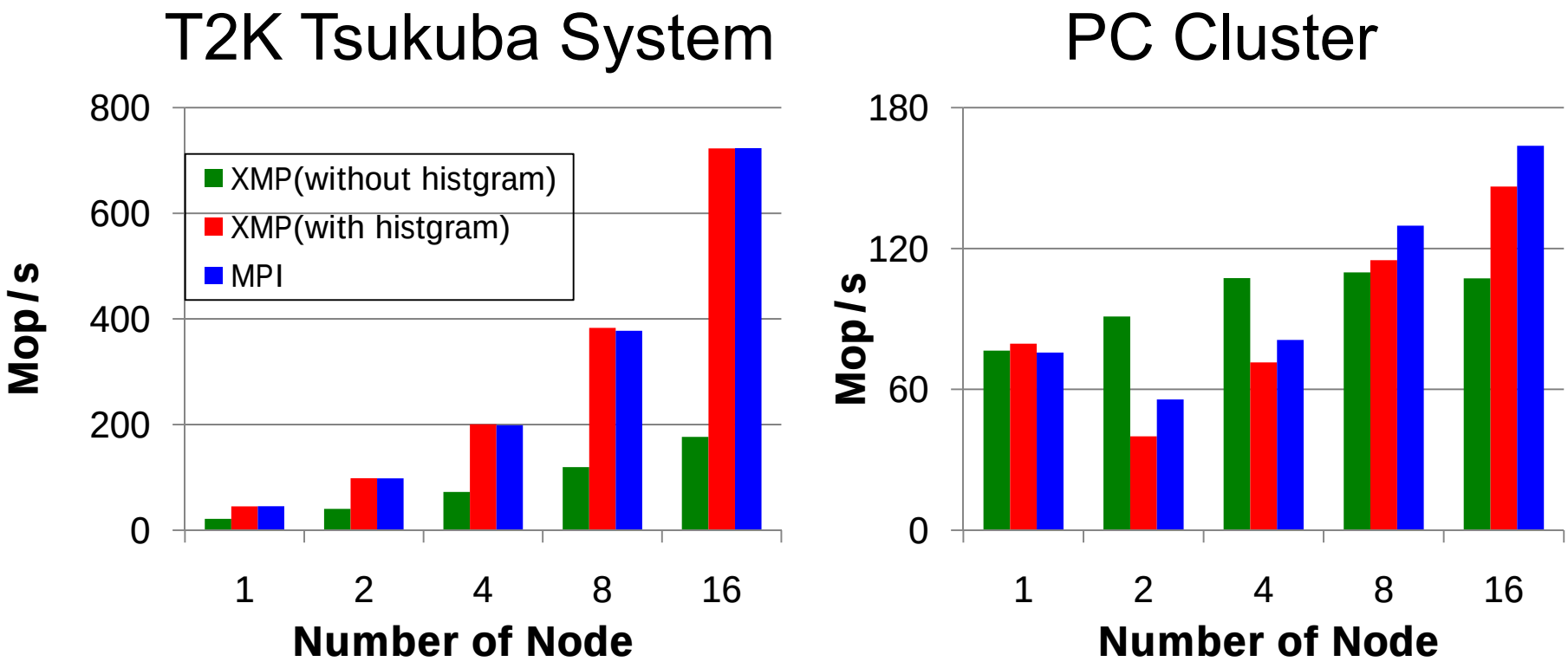
Gigabit Ethernet

Performance Results : NPB-EP



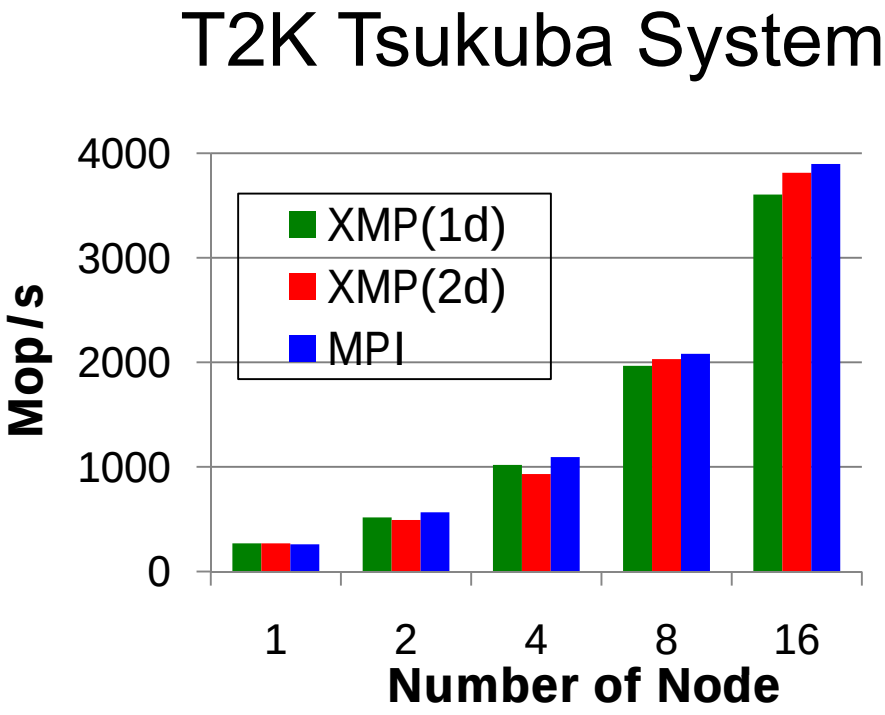
The difference in performance at 1 node is because the performance of the C compiler is poor than that of Fortran.

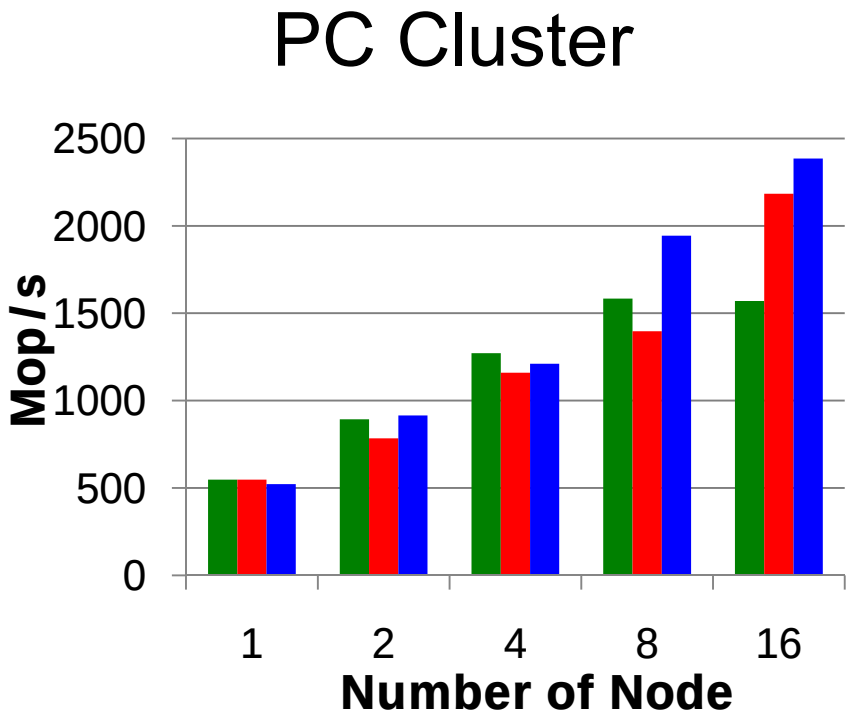
Evaluation Results : NPB-IS



The results indicate that the performance of XMP with a histogram is comparable to that of MPI.

Performance Results : NPB-CG





The results for CG indicate that the performance of 2D. parallelization in XMP is comparable to that of MPI.

Short Summary

http://www.xcalablemp.org

- Preliminary performance report NPB results
  - We found XMP can be a good solution to describe these benchmarks.
  - Performance looks reasonable, but much performance tuning is required
  - More experience is needed in real apps.

- XcalableMP project: status and schedule
  - A draft of XcalableMP specification 0.7
    - http://www.xcalablemp.org/xmp-spec-0.7.pdf
  - 3Q/10 beta release, C language version compiler (at SC10)
  - Fortran version compiler after SC10

- Issues under discussion
  - Multicores (SMP) Cluster and Hybrid programming with OpenMP
  - Parallel IO
  - Extension to GPGPU, Manycore, Fault tolerant?, Others …

Thank you for your attention!!!

Q & A?

http://www.xcalablemp.org/

Acknowledgements:

We would like to thank XMP-WG members for Valuable discussions and comments