Productivity and Performance of the HPC Challenge Benchmarks with the XcalableMP PGAS Language

Masahiro Nakao, Hitoshi Murai,
Takenori Shimosaka, Mitsuhisa Sato

Center for Computational Sciences, University of Tsukuba, Japan
RIKEN Advanced Institute for Computational Science, Japan

Overview of XcalableMP（XMP)

- Directive-based language extension of C99 and Fortran2008
  - The same directives are used in XMP/C and XMP/Fortran
  - Coarray syntax is available in XMP/C and XMP/Fortran

XMP/C

```
int array[16];
#pragma xmp nodes p(4)
#pragma xmp template t(0:15)
#pragma xmp distribute t(block) onto p
#pragma xmp align array[i] with t(i)

main(){
  ...
#pragma xmp loop on t(i)
  for(i = 0; i < 16; i++){
    array[i] = func(i);
  }
}
```

XMP/Fortran

```
integer array(16);
!$xmp nodes p(4)
!$xmp template t(1:16)
!$xmp distribute t(block) onto p
!$xmp align array(i) with t(i)

program main
  ...
!$xmp loop on t(i)
  do i=1,16
    array(i) = func(i)
  done
end program
```

Overview of XcalableMP（XMP)

- Directive-based language extension of C99 and Fortran2008
  - The same directives are used in XMP/C and XMP/Fortran
  - Coarray syntax is available in XMP/C and XMP/Fortran

XMP/C

```
int b[10]:[*];

if(me == 1){
  b[0:5]:[2] = b[0:5];  // Put
}
```

XMP/Fortran

```
integer b(10)[*]

if(me == 1) then
  b(1:5)[2] = b(1:5)  // Put
end if
```

XMP/Fortran is upward compatible with the Fortran2008

Objective

- Examine effectiveness of designs of the XMP PGAS language for improved **productivity** and **performance** of HPC systems
  - Evaluate the productivity and the performance of XMP through implementations of the HPC Challenge (HPCC) Benchmarks
  - Use 32,768 compute nodes at a maximum on the K computers (which consists of 88,128 compute nodes)

ranked 1st in the Top500 on June, 2011

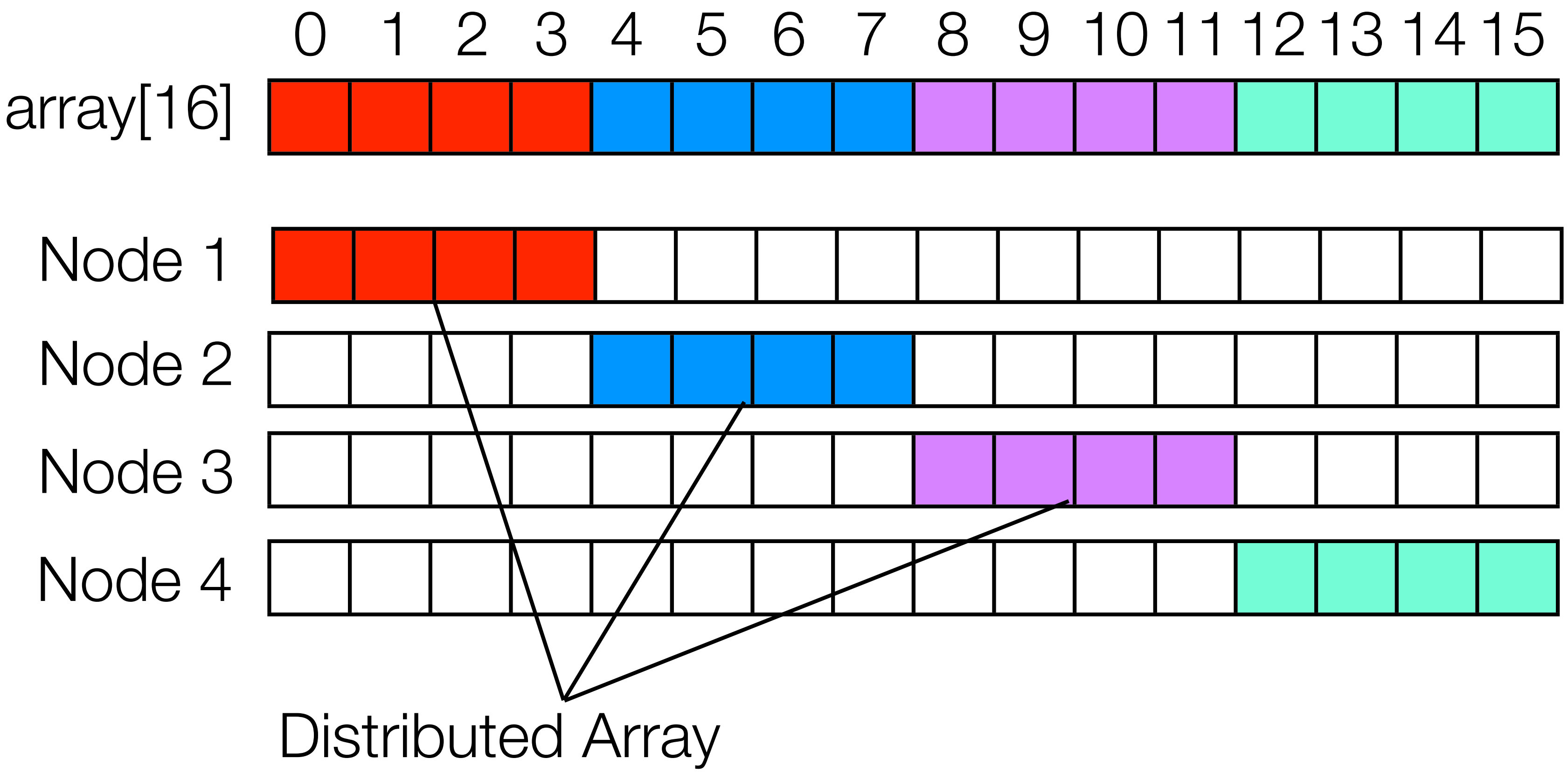
Agenda

1. Introduce XMP features
   - Global-view memory model with XMP directives
   - Local-view memory model with coarray syntax
   - Designs of XMP for HPC applications
2. Explain implementations of the HPCC Benchmarks and evaluate their productivity and performance
3. Discuss experimental results
4. Summarize our presentation

XMP Global-view model (1/3)

- The directives specify a data distribution among nodes

```
int array[16];
#pragma xmp nodes p(4)
#pragma xmp template t(0:15)
#pragma xmp distribute t(block) on p
#pragma xmp align array[i] with t(i)
```

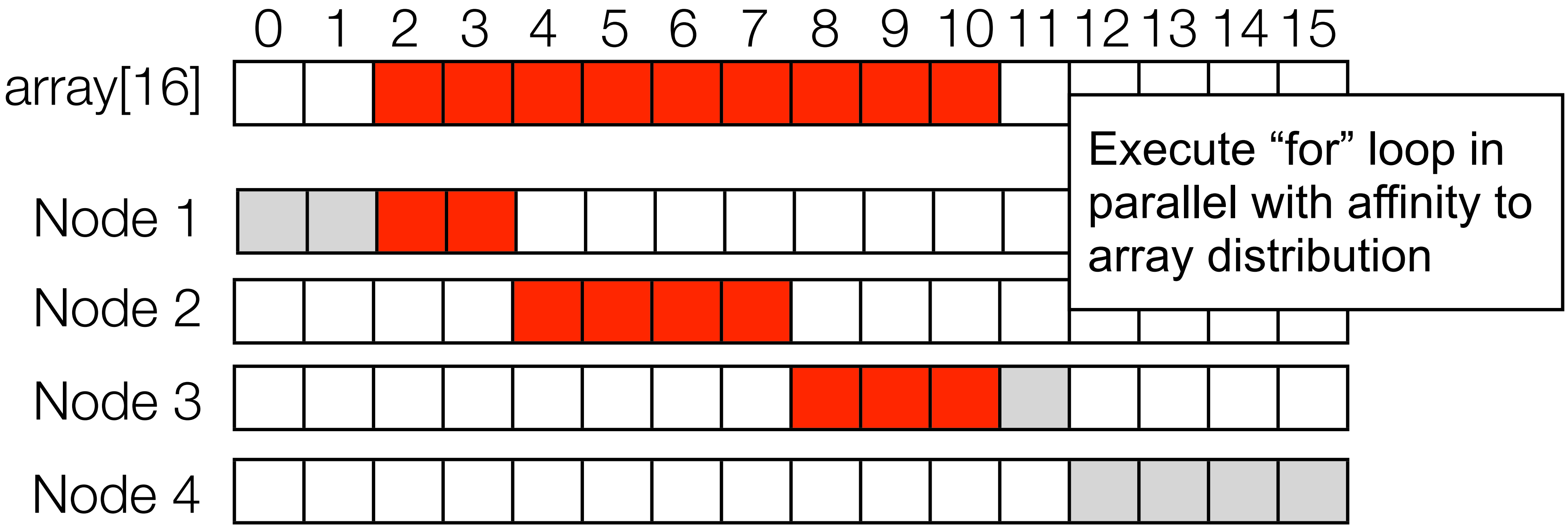


XMP Global-view model (2/3)

- Loop directive is to parallelize loop statement

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

```
int array[16];
#pragma xmp nodes p(4)
#pragma xmp template t(0:15)
#pragma xmp distribute t(block) on p
#pragma xmp align array[i] with t(i)
```



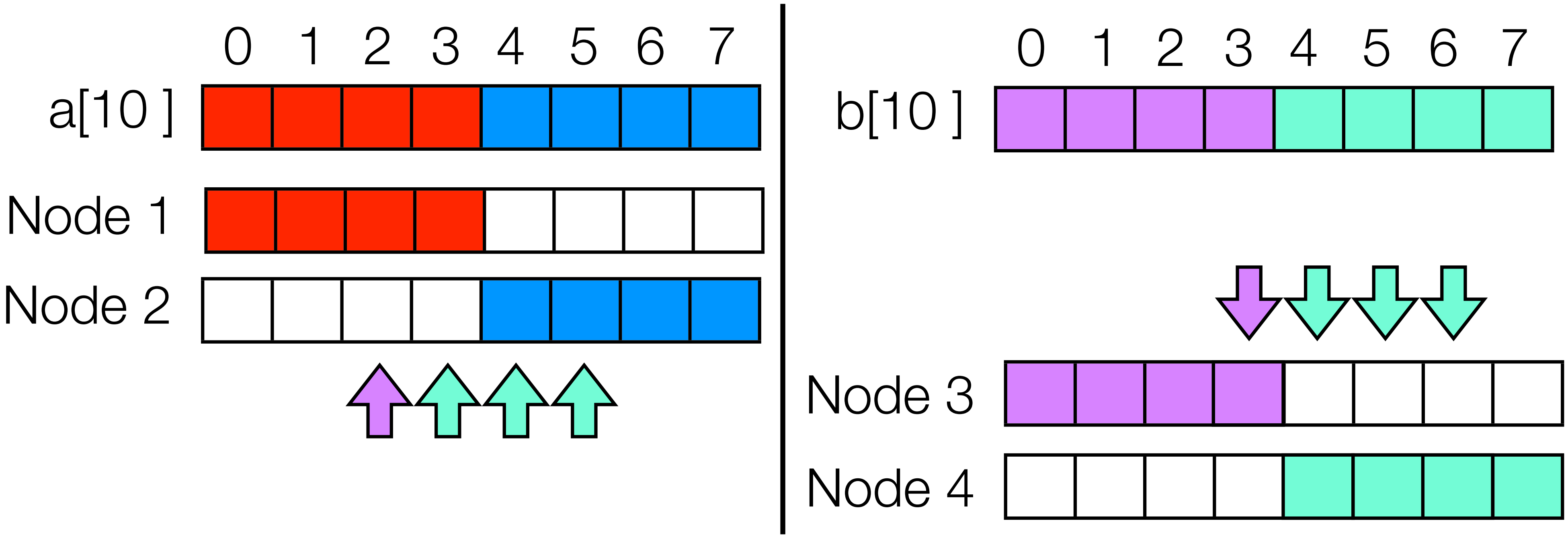
Each node computes Red elements in parallel

XMP Global-view model (3/3)

- Data communication directives : broadcast, reduction, gmove
- gmove directive
  - Transfer data while keeping the global image by using "array section notation"

[start_index : length]

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



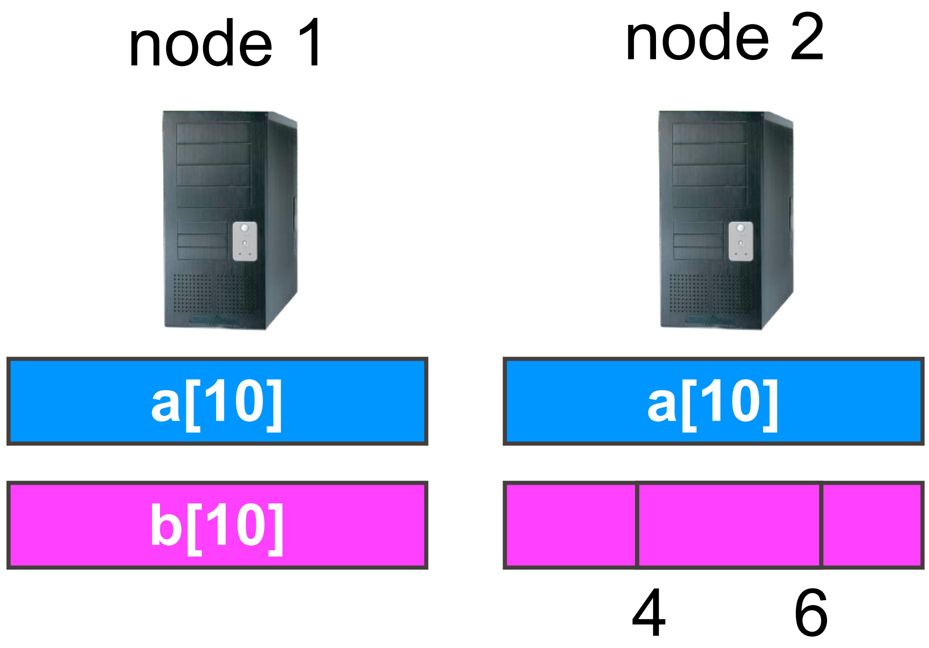
XMP Local-view model

- Support coarray syntax in XMP/C and XMP/Fortran
  - XMP/Fortran is upward compatible with the Fortran 2008
  - XMP/C also uses **array section notation** in coarray syntax

XMP/C

```
int b[10]:[*];                // Declare
      :
if(me == 1)
  a[0:3] = b[4:3]:[2];        // Get
```

The node1 gets b[4:3] in the node 2



It is easy to express one-sided communication for local data (Put/Get).
Can mix XMP global-view directives with coarray-syntax.

Designs of XMP for HPC applications

- PGAS programming language must have both **high productivity** and **high performance**
  - The productivity of HPC applications consists of **programming cost, educational cost, porting cost, and tuning cost**
- Designs of XMP for HPC applications (1/3)
  - Easy writing of various parallel applications **<programming cost↓>**
    - [Global-view] Enable parallelization of an original sequential code using minimal modification with simple directives
    - [Local-view] Easy to express one-sided comm. with coarray-syntax
  - Easy learning **<educational cost↓>**
    - Extension of C and Fortran

Designs of XMP for HPC applications

- Design of XMP for HPC applications (2/3)
  - Numerical libraries (BLAS etc.) & MPI library can be invoked from XMP program <**porting↓, tuning cost↓, performance↑**>

```
int array[16];
#pragma xmp nodes p(4)
#pragma xmp template t(0:15)
#pragma xmp distribute t(block) onto p
#pragma xmp align array[i] with t(i)

main(){
    ...
    cblas_dgemm(.., &array[k], ...);
}
```

This is a code example where a **global array** is used in BLAS library.

a pointer of a global array indicates a local pointer on the node to which it is distributed

XMP inquiry functions obtain local memory information from a global array. For example, `xmp_array_lead_dim()` obtains a local leading dimension of a global array.

Designs of XMP for HPC applications

- Design of XMP for HPC applications (3/3)
  - "OpenMP-safe", except for comm. directives <**performance↑**>
    - Programmer can use **OpenMP directives** in XMP

XMP/C

```
int array[16];
#pragma xmp nodes p(4)
#pragma xmp template t(0:15)
#pragma xmp distribute t(block) onto p
#pragma xmp align array[i] with t(i)

main(){
   ...
#pragma xmp loop on t(i)
#pragma omp parallel for
   for(i = 0; i < 16; i++){
      array[i] = func(i);
   }
}
```

XMP/Fortran

```
integer array(16);
!$xmp nodes p(4)
!$xmp template t(1:16)
!$xmp distribute t(block) onto p
!$xmp align array(i) with t(i)

program main
   ...
!$xmp loop on t(i)
!$omp parallel do
   do i=1,16
      array(i) = func(i)
   done
end program
```

Agenda

1. Introduce XMP features
   - Global-view memory model with XMP directives
   - Local-view memory model with coarray syntax
   - Designs of XMP for HPC applications
2. Explain implementations of the HPCC Benchmarks and evaluate their productivity and performance
3. Discuss experimental results
4. Summarize our presentation

HPC Challenge(HPCC) Benchmarks

- The HPCC Benchmarks are a set of benchmarks to evaluate multiple attributes on an HPC system
- The HPCC Benchmarks are also used at HPCC Award Competition at Supercomputer Conference
  - 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
    - RandomAccess
    - High Performance Linpack (HPL)
    - Fast Fourier Transform (FFT)
    - STREAM

- based on hpcc-1.4 written in C + Fortran + MPI which is released by the HPCC community (http://icl.cs.utk.edu/hpcc/software/)
- weak scaling

Evaluation

- Omni XMP Compiler version 0.7-alpha
  - Reference Implementation
  - Open Source http://www.hpcs.cs.tsukuba.ac.jp/omni-compiler/xcalablemp/
  - Optimized for the K computer
    - "./configure --target=Kcomputer-linux-gnu"
    - To use high-speed one-sided communication on the K computer, the coarray syntax is translated into calling the extended RDMA
  - This Compiler will be released in Nov. 2013

Environment

	The K computer	HA-PACS
CPU	SPARC64 VIIIfx 2.0GHz 8Cores, **128GFlops**	Xeon E5-2670 2.6GHz x2 8Cores x2, **332.8GFlops**
Memory	DDR3 SDRAM **16GB** **64GB/s/Socket**	DDR3 SDRAM **128GB** **51.4GB/s/Socket**
Network	Torus fusion six-dimensional mesh/torus network, **5GB/s**	Infiniband QDRx2rails Fat-tree network, **4GB/s**

HA-PACS has GPUs
as an accelerator.
But we used only CPU.

To measure the performance, we used **32,768 nodes** at a maximum of the K computer and **64 nodes** at a maximum of HA-PACS

RandomAccess

- The RandomAccess benchmark measures the performance of random integer updates of memory via interconnect
  - Each process randomly updates table of other processes
  - It is suitable to use coarray syntax
  - To reduce communication times, our algorithm is iterated over sets of CHUNK updates on each node
    - Our algorithm is almost the same as the hpcc-1.4 RandomAccess

RandomAccess

Source lines of code (SLOC) **is 258**, written in XMP/C

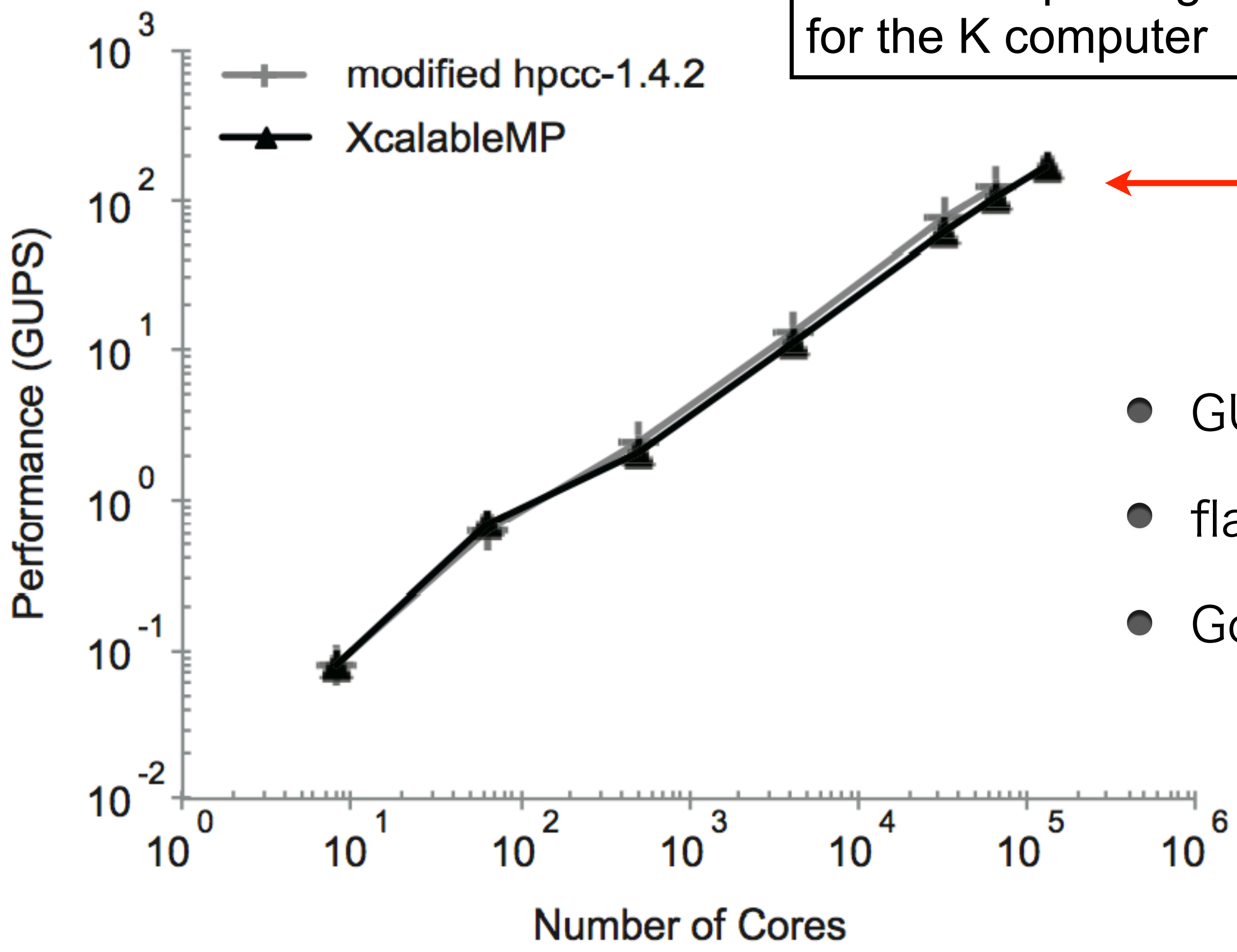
```
u64Int recv[MAXLOGPROCS][RCHUNK+1]:[*];

for(...){
  ...
  send[isend][0] = nsend;  // set "number of transfer elements"
  recv[j][0:nsend+1]:[send_target] = send[isend][0:nsend+1];
#pragma xmp sync_memory
#pragma xmp post(p(send_target), 0)
  ...
#pragma xmp wait(p(recv_target))
#pragma xmp sync_memory
 nrecv = recv[j-1][0];
 sort_data(&recv[j-1][1], nrecv, ..);
 ...
}
```

Declare coarray

PUT

Ensure to finish PUT operation

Performance of RandomAccess



The modified hpcc-1.4 RandomAccess, for which the functions updating the table are specifically optimized for the K computer

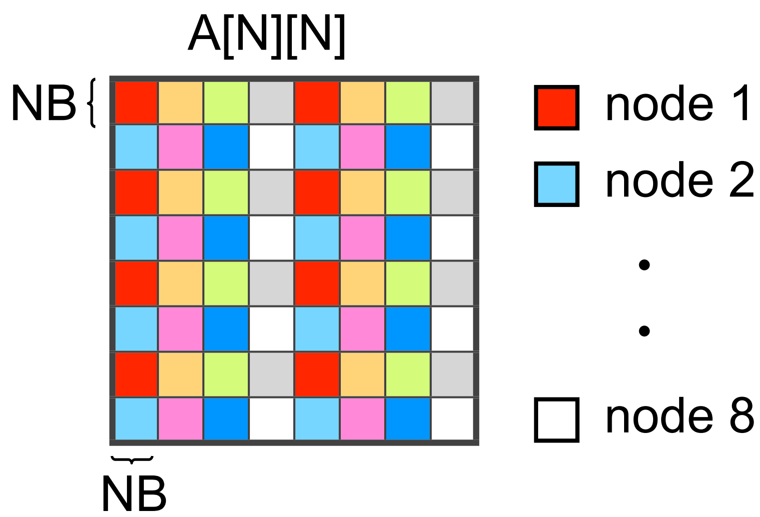
163GUPS in 16,384 nodes (131,072 CPU Cores)

- GUPS (Giga UPdates per Second)
- flat-MPI (8 Process/Node)
- Good Performance !!

High Performance Linpack (HPL)

- HPL measures the floating point rate of execution to solve a dense system of linear equations using LU factorization
  - In our implementation, the coefficient matrix is distributed in block-cyclic manner like hpcc-1.4 HPL
    - This distribution is useful to perform good load balance
  - BLAS Library is used

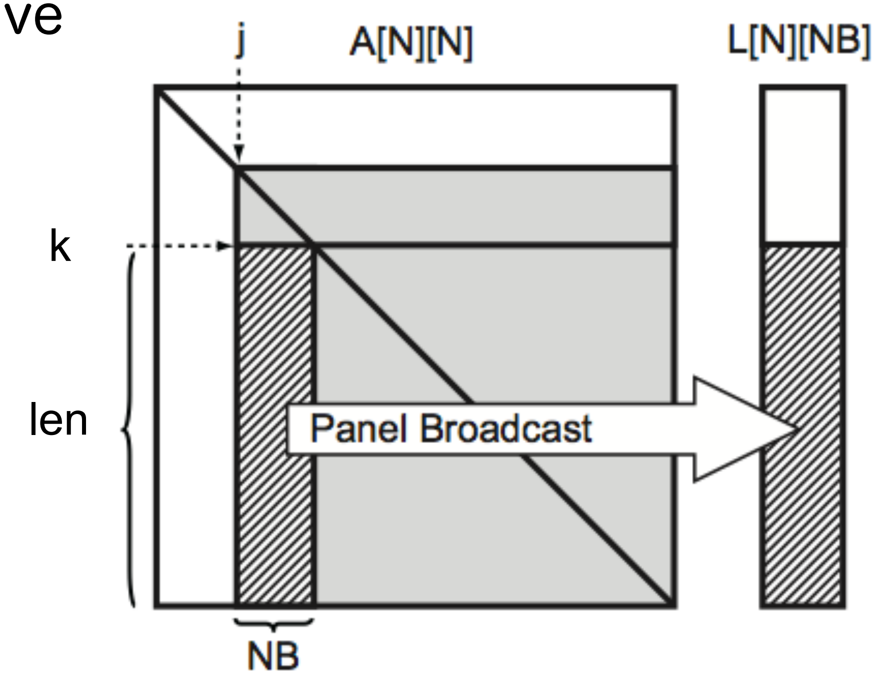
```
double A[N][N];
#pragma xmp nodes p(4,2)
#pragma xmp template t(0:N-1, 0:N-1)
#pragma xmp distribute t(cyclic(NB), \
                   cyclic(NB)) onto p
#pragma xmp align A[i][j] with t(j,i)
```



High Performance Linpack (HPL)

- Panel Broadcast by using **gmove** directive

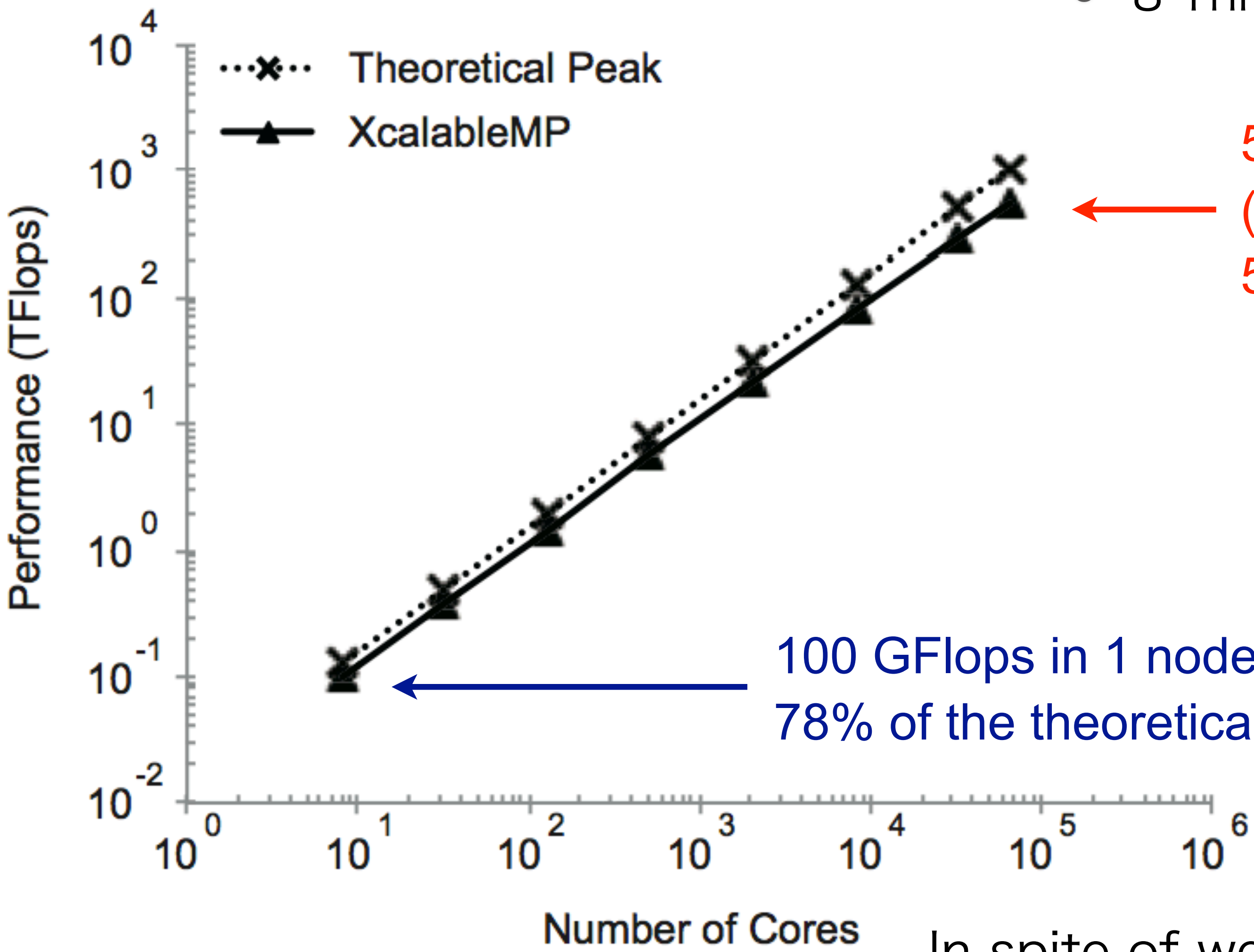
```
double A_L[N][NB];
#pragma xmp align L[i][*] with t(*,i)
    :
#pragma xmp gmove
L[k:len][0:NB] = A[k:len][j:NB];
```



SLOC is **288**, written in XMP/C

Performance of HPL

- 8 Threads/Process on 1 node



543 TFlops in 8,192 nodes (65,536 CPU Cores), 53% of the theoretical peak

100 GFlops in 1 node, 8 CPU Cores, 78% of the theoretical peak

In spite of weak scaling, the parallel efficiency is not very good.

Fast Fourier Transform (FFT)

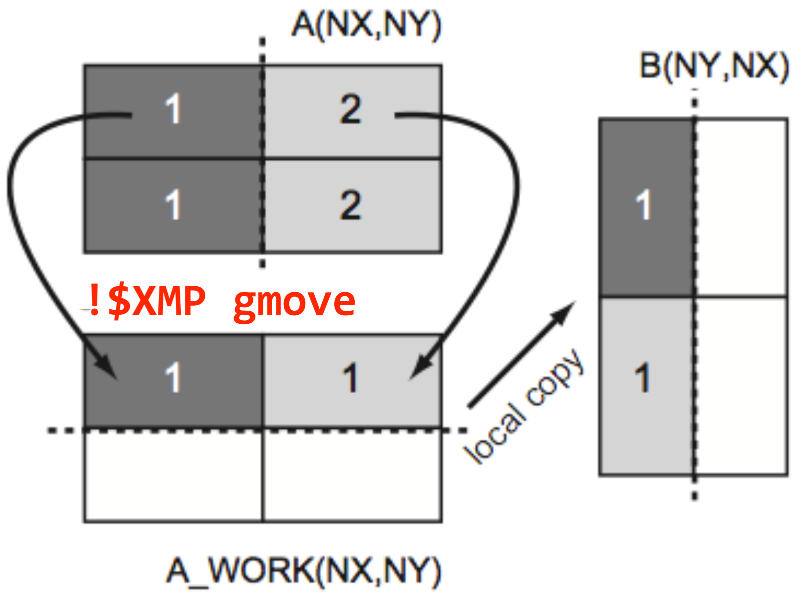
- FFT measures the floating point rate of execution for double-precision complex one-dimensional Discrete Fourier Transform
- We parallelized only a subroutine “PZFFT1D0”, which is the main kernel of the hpcc-1.4 FFT

Fast Fourier Transform (FFT)

- Matrix transposition is implemented by using **gmove** directive

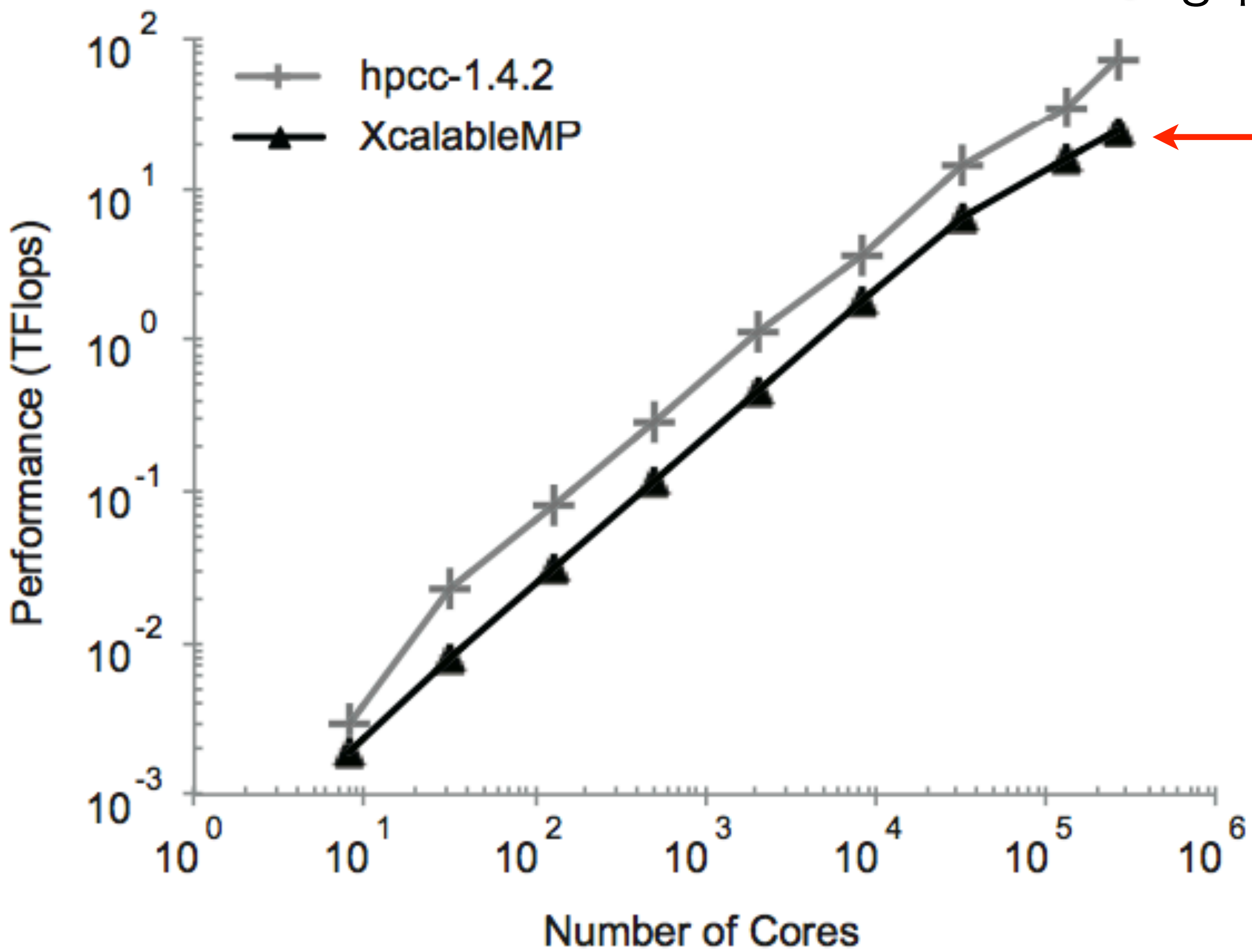
The SLOC of PZFFT1D0 is **65**, written in XMP/Fortran + OpenMP

```
!$XMP distribute tx(block) onto p
!$XMP distribute ty(block) onto p
!$XMP align A(*,i) with ty(i)
!$XMP align A_WORK(i,*) with tx(i)
!$XMP align B(*,i) with tx(i)
   :
!$XMP gmove
A_WORK(:,:) = A(:,:) ! all-to-all

!$XMP loop on tx(I)
!$OMP parallel do
DO 60 I=1,NX
  DO 70 J=1,NY
    B(J,I)=A_WORK(I,J)
  60 CONTINUE
70 CONTINUE
```



1. Node 2 transfers data to node 1 with packing it
2. Node 1 copies A_WORK() to B() by using XMP and OpenMP directives

Performance of FFT



- 8 Thread/process on 1 node

24TFlops in 32,768 nodes (262,144 CPU Cores)

50% of performance of the hpcc-1.4 FFT

Agenda

1. Introduce XMP features
    - Global-view memory model with XMP directives
    - Local-view memory model with coarray syntax
    - Designs of XMP for HPC applications
2. Explain implementations of the HPCC Benchmarks and evaluate their productivity and performance
3. Discuss experimental results
4. Summarize our presentation

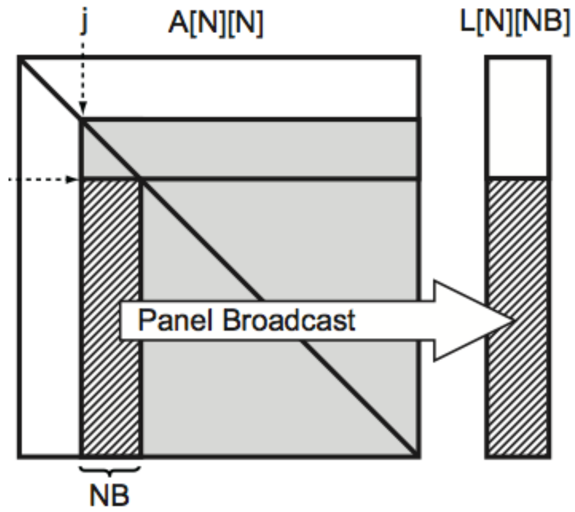
Comparison with hpcc-1.4 (MPI)

- Productivity
  - RandomAccess : SLOC : 938 -> 258 (hpcc-1.4 -> XMP)
    - coarray is a more convenient to express communications
  - HPL : SLOC : 8,800 -> 288
  - PZFFT1D0 of FFT : SLOC : 101 -> 65
    - XMP global view enables programmers to develop parallel applications easily
- Performance
  - RandomAccess : Good !
  - HPL and FFT : The performances of XMP implementations are worse than those of hpcc-1.4

Discussion (2/3)

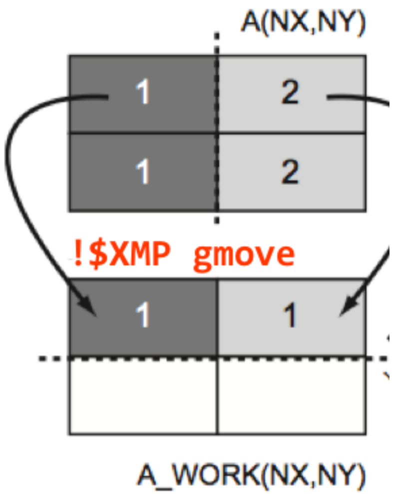
- Overhead of **gmove directive**
- [HPL] **Gmove directive** is a blocking operation. Communication and computation are not overlapped.

```
#pragma xmp gmove
A_L[k:len][0:NB] = A[k:len][j:NB];
```



- [FFT] In **gmove directive**, data pack/unpack operation is not executed with thread-parallelization

```
!$XMP gmove
A_WORK(:,:) = A(:,:)
```



Discussion (3/3)

- To improve performance
  - non-blocking gmove operation
  - data pack/unpack with threaded-parallelization in gmove

```
#pragma xmp gmove async(async-id)
A_L[k:len][0:NB] = A[k:len][j:NB];

(overlapped computation)
#pragma xmp wait_async async(async-id)
```

- Improving the performance of the gmove is important. **But**, …
  - While level of abstraction of the gmove is very high, the performance of the gmove remains unclarity
    - Gmove improves the productivity, but may become worse the performance
  - If the performance of the gmove has a problem, we recommend that programmer will be able to rewrite the communication with coarray-syntax or MPI library

Summary

- Examine the effectiveness of designs of XMP for improved the productivity and the performance of a HPC system
  - Global-view model and Local-view model
  - Can use Numerical Library with XMP inquiry functions
- Evaluate the productivity and the performance through implementations of HPCC Benchmarks on the K computer
  - Good productivity and performance in 32,768 nodes at a maximum
    - But the gmove directive has scope to continue to improve
- Future work
  - Support non-blocking operation and thread-parallelization
  - Retry to evaluate their performances for next HPCC Award at SC13