Highly productive parallel programming language

XcalableMP

Masahiro NAKAO
Center for Computational Sciences
University of Tsukuba

Background

- MPI is widely used as a parallel programming model on distributed memory systems
  - However, writing MPI programs is often a time-consuming and a complicated process
- Another programming model is needed !!
  - High performance
  - Easy to program

Development of XcalableMP

What's XcalableMP?

- XcalableMP(XMP) is a new programming model and language for distributed memory systems
- XMP is proposed by XcalableMP Specification Working Group(XMP WG)
  - XMP WG is a special interest group, which is organized to make a draft on "petascale" parallel language
  - XMP WG consists of members from academia (U. Tsukuba, U. Tokyo, Kyoto U. and Kyusyu U.), research labs(RIKEN, NIFS, JAXA, JAMSTEC/ES) and industries(Fujitsu, NEC, Hitachi)
- XMP Specification version 1.0 released !!
  - http://www.xcalablemp.org/

Agenda

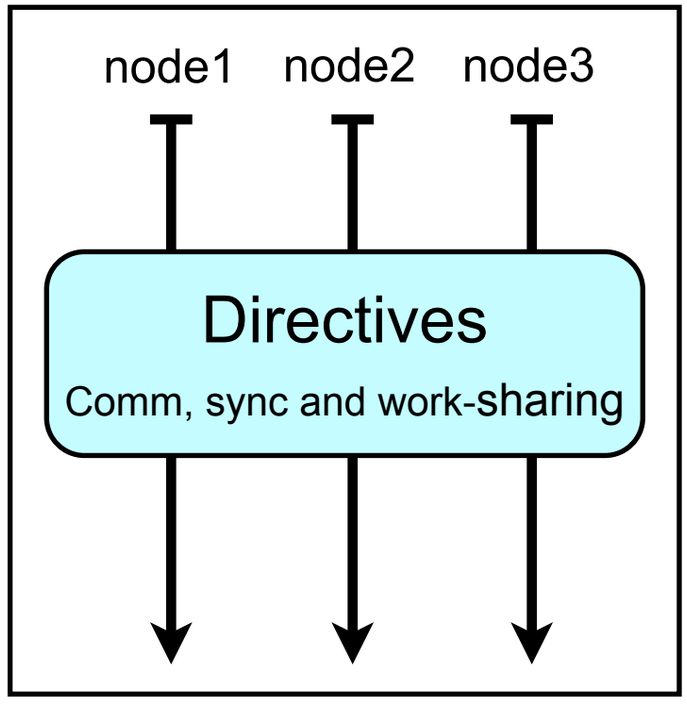
- Overview of XMP
- XMP Programming Model
- XMP Directives

Agenda

- Overview of XMP
- XMP Programming Model
- XMP Directives

Overview of XMP

- XMP is a directive-based language extension like OpenMP and HPF based on C and Fortran95
  - To reduce code-writing and educational costs
- “Scalable” for Distributed Memory Programming
  - A thread starts execution in each node independently (as in MPI)
- “Performance-aware” for explicit communication, sync. and work-sharing
  - All actions occur when directives are encountered
  - All actions are taken by directives for being “easy-to-understand” in performance tuning (different from HPF)



XMP Code Example

```
int array[100];
#pragma xmp nodes p(*)
#pragma xmp template t(0:99)
#pragma xmp distribute t(block) onto p
#pragma xmp align array[i] with t(i)

main(){
#pragma xmp loop on t(i) reduction(+:res)
    for(i = 0; i < 100; i++){
        array[i] = func(i);
        res += array[i];
    }
}
```

data distribution

work mapping & reduction

Programmer adds XMP directives to serial code

<serial incremental parallelization>

The same code written in MPI

```
int array[100];

main(int argc, char **argv){
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    dx = 100/size;
    llimit = rank * dx;
    if(rank != (size -1)) ulimit = llimit + dx;
    else ulimit = 100;

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

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

Agenda

- Overview of XMP
- Programming Model
- XMP Directives

Programming Model

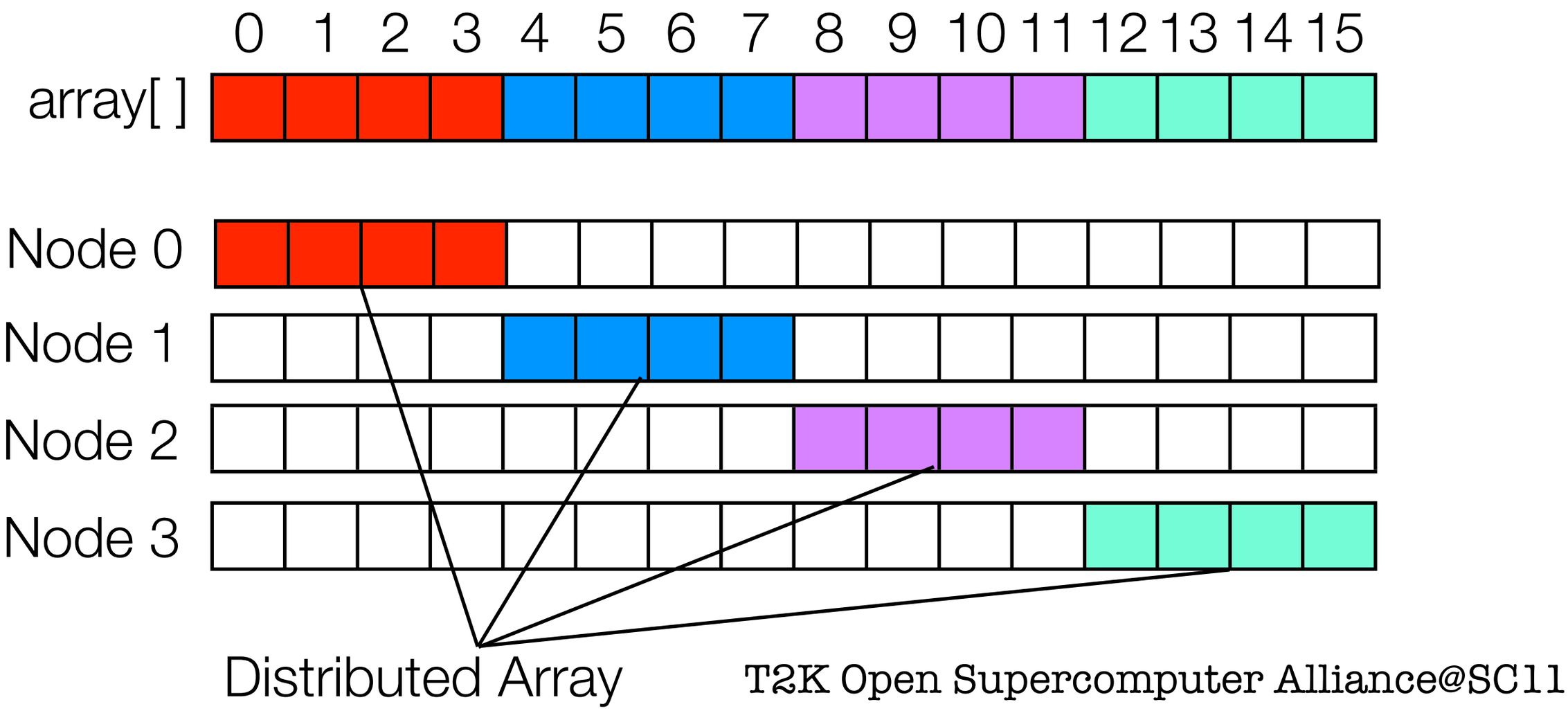
XcalableMP

- Global View Model (like as HPF)
  - programmer describes data distribution, work mapping and inter-node comm. with adding directives to serial code
    - ・support typical techniques for work-mapping and data-mapping
    - ・rich communication and sync. directives, such as “shadow”, “reflect” and “gmove”
- Local View Model (like as Co-array Fortran)
  - enable programmer to communicate with node index
  - one-sided communication using language extension (also defined C)

Data Distribution

- The directives specify a data distribution among nodes

```
#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)
```

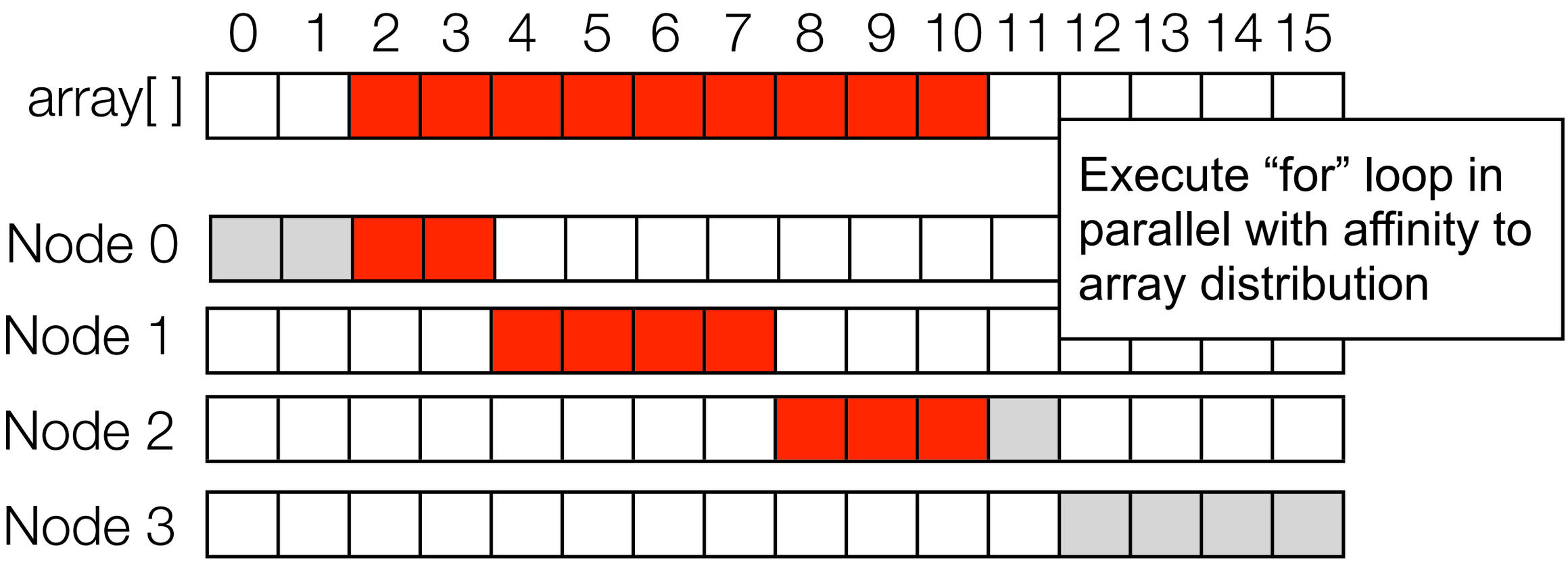


Parallel Execution of loop

- Loop directive is inserted before loop statement

```
#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)
```

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



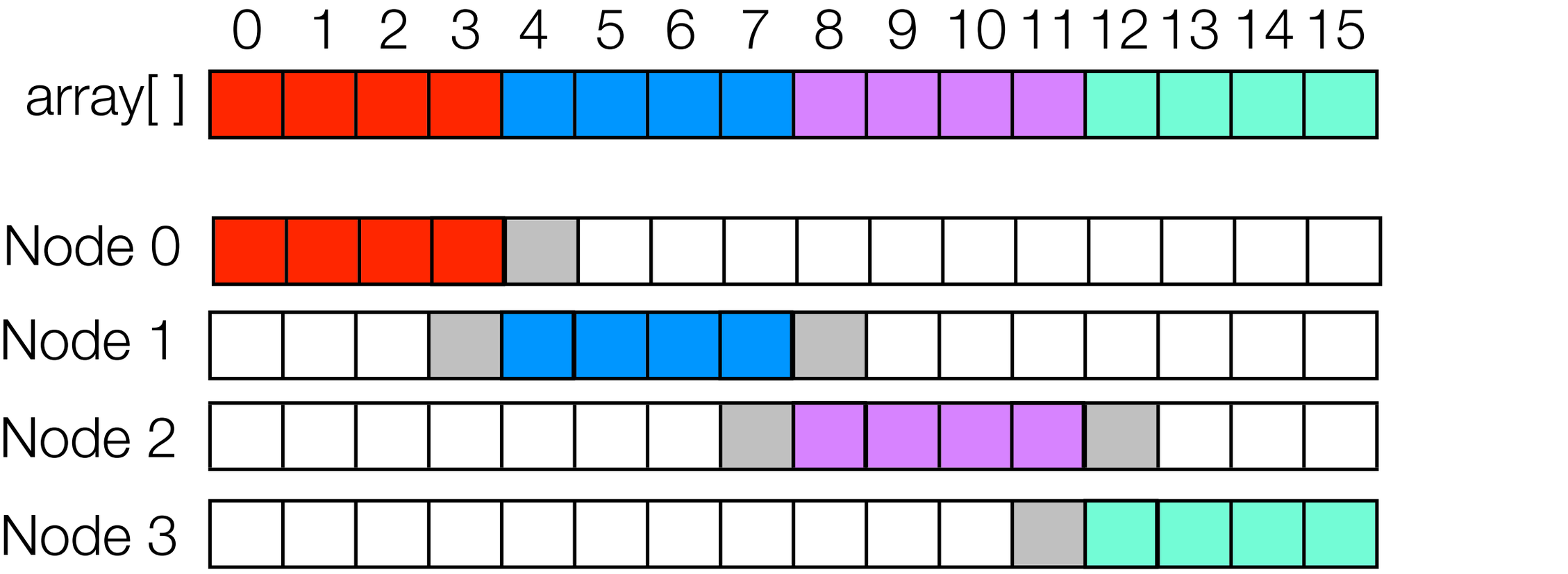
Each node computes Red elements in parallel

shadow/reflect directives

- Synchronize data only on shadow region
  - If neighbor data is required to communicate, then only shadow area can be considered

```
#pragma xmp shadow array[1:1]
```

  - for(...) b[i] = array[i-1] + array[i+1];



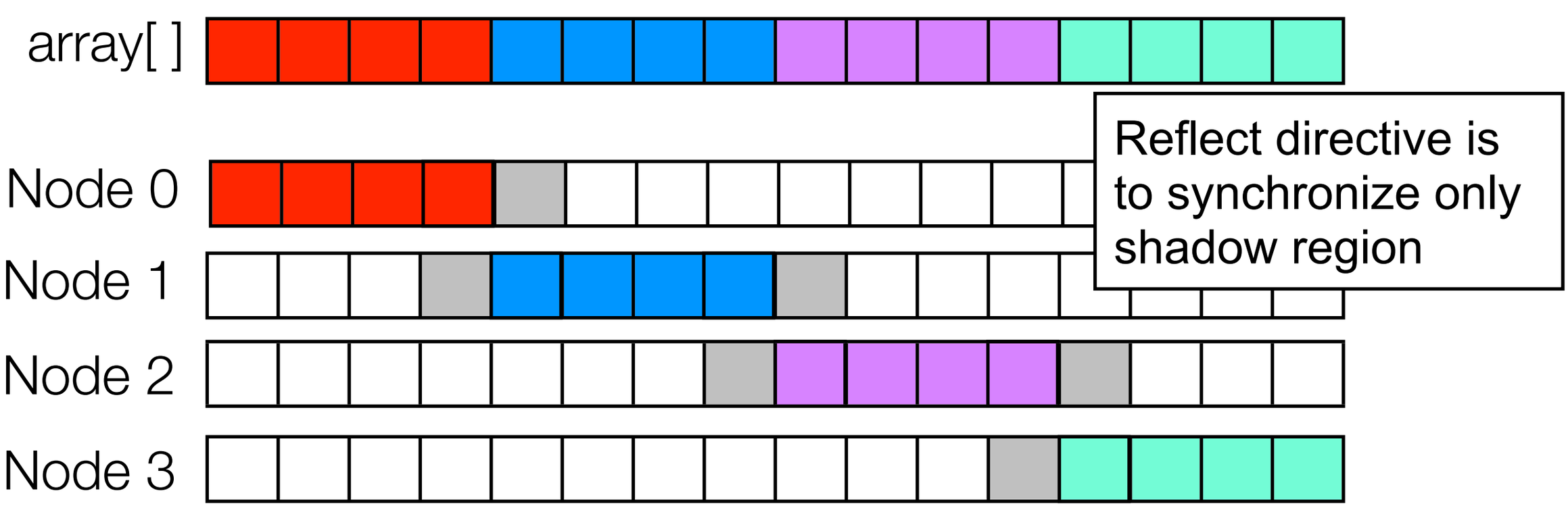

```
#pragma xmp reflect array
```

shadow/reflect directives

- Synchronize data only on shadow region
  - If neighbor data is required to communicate, then only shadow area can be considered

```
#pragma xmp shadow array[1:1]
```

  - for(...) b[i] = array[i-1] + array[i+1];



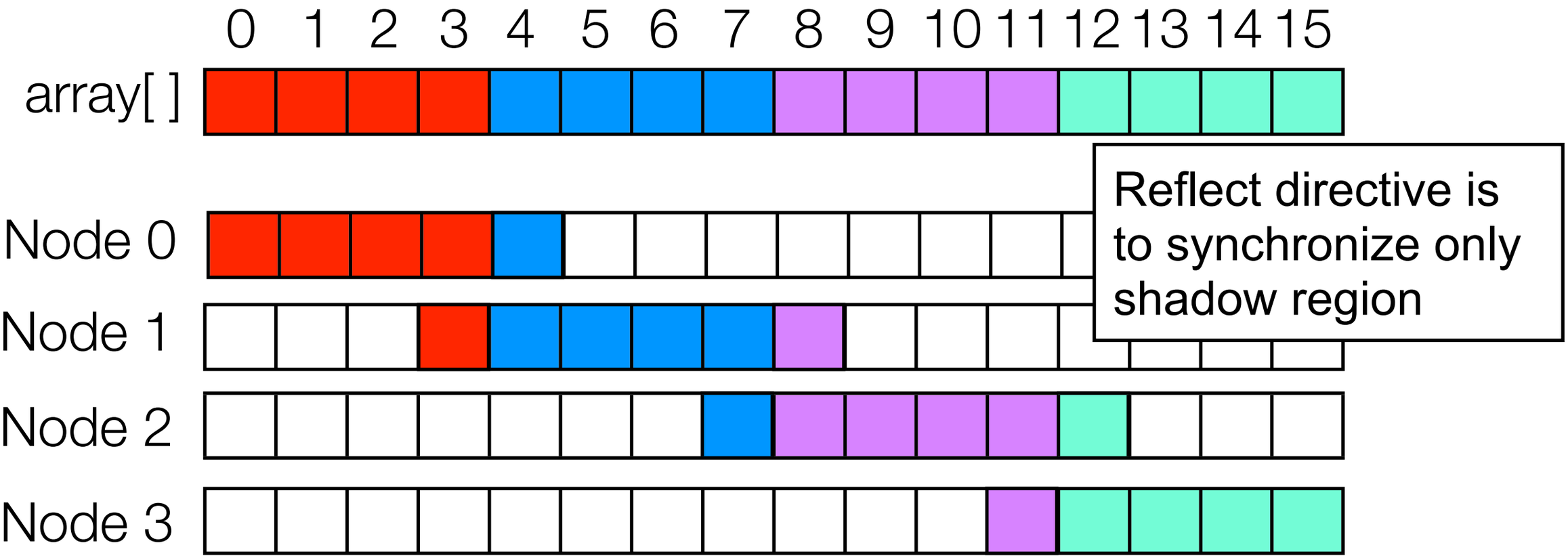

```
#pragma xmp reflect array
```

shadow/reflect directives

- Synchronize data only on shadow region
  - If neighbor data is required to communicate, then only shadow area can be considered

```
#pragma xmp shadow array[1:1]
```

  - for(...) b[i] = array[i-1] + array[i+1];

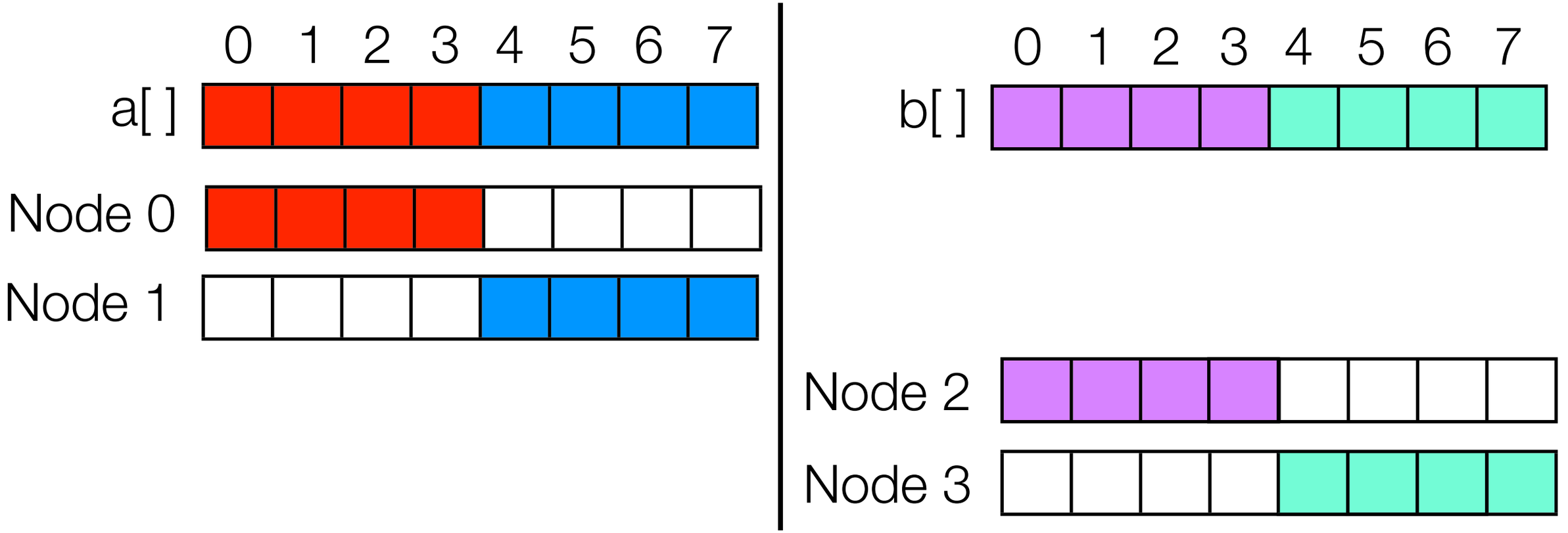



```
#pragma xmp reflect array
```

Gmove (Global View Communication)

- communication for distributed array
  - We extend “array section notation” in C
  - Programmer doesn’t need to know where each data is distributed

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



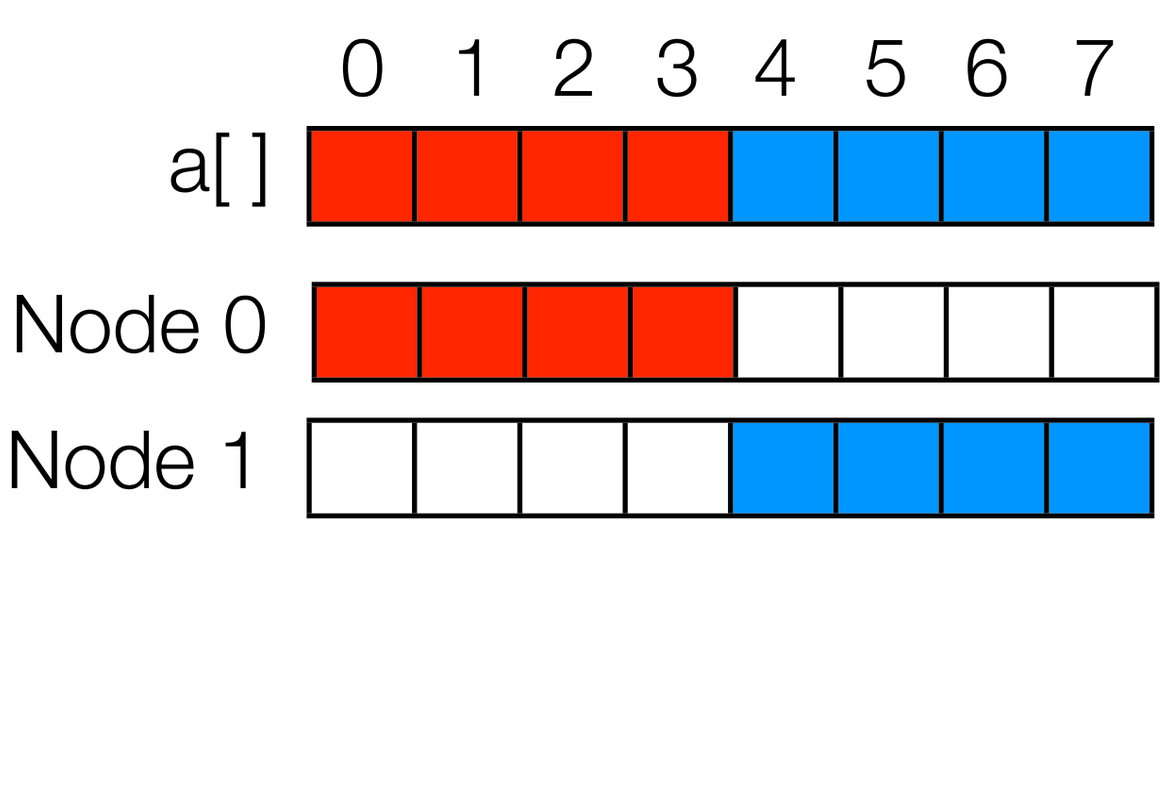
Gmove (Global View Communication)

- communication for distributed array
  - We extend “array section notation” in C
  - Programmer doesn’t need to know where each data is distributed

base

length

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



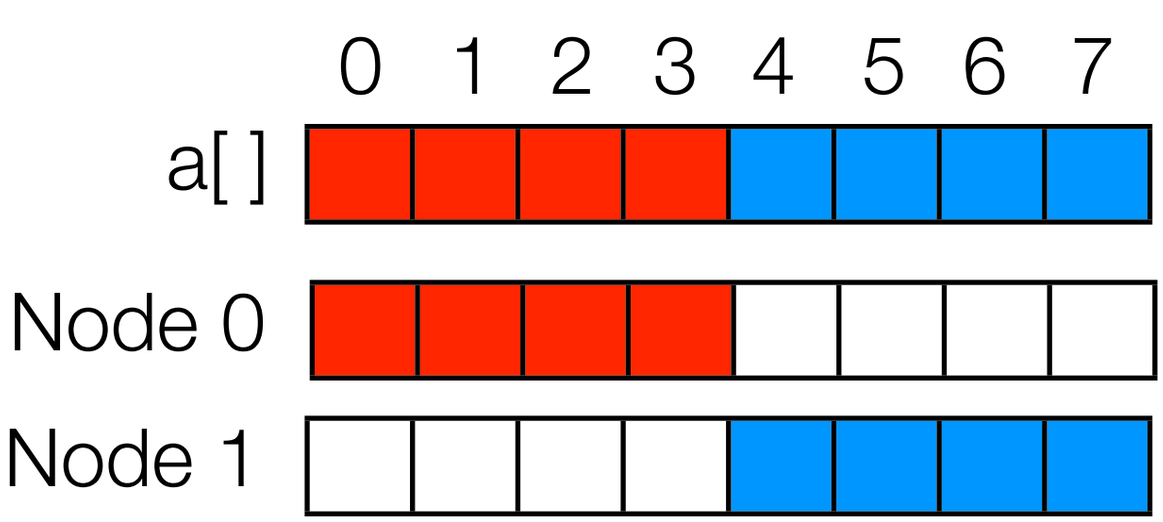
Gmove (Global View Communication)

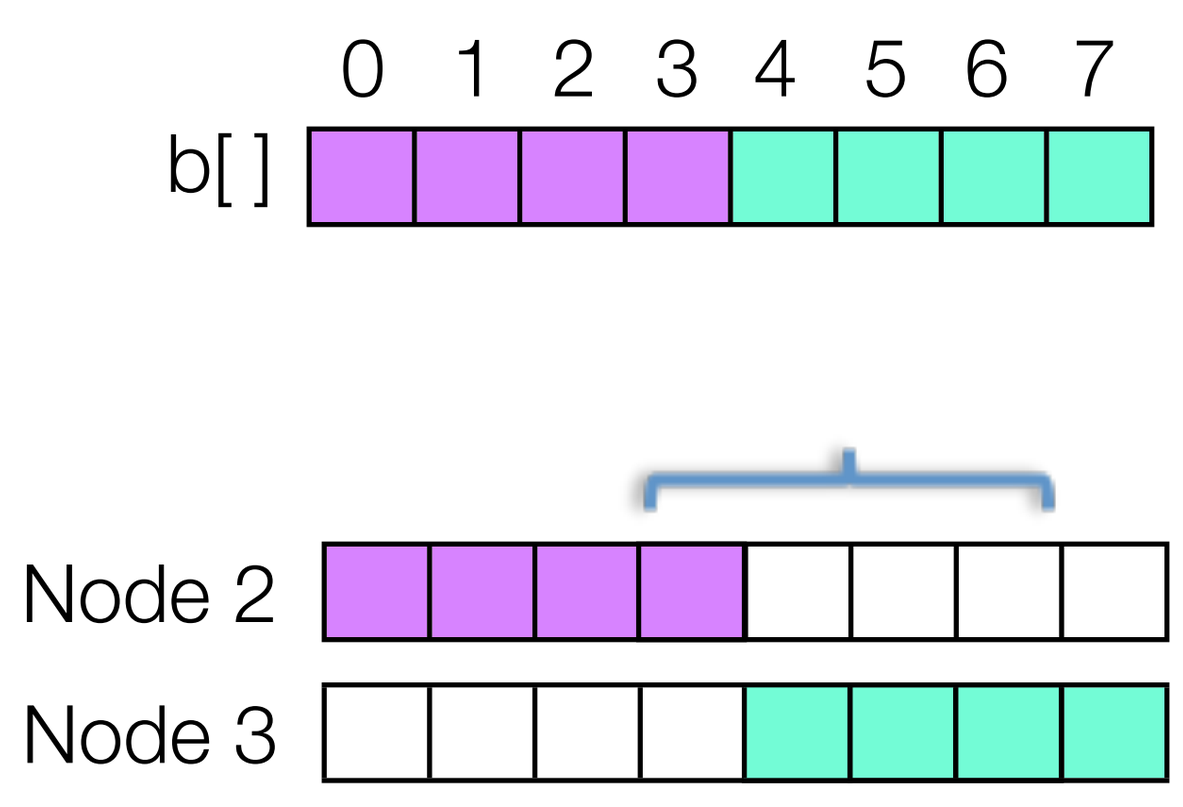
- communication for distributed array
  - We extend “array section notation” in C
  - Programmer doesn’t need to know where each data is distributed

base

length

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





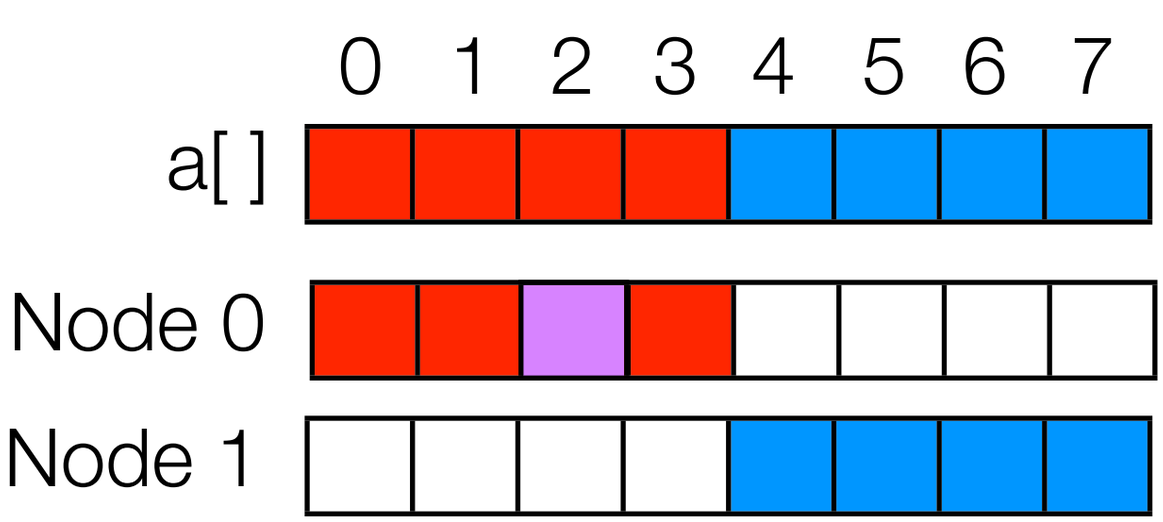
Gmove (Global View Communication)

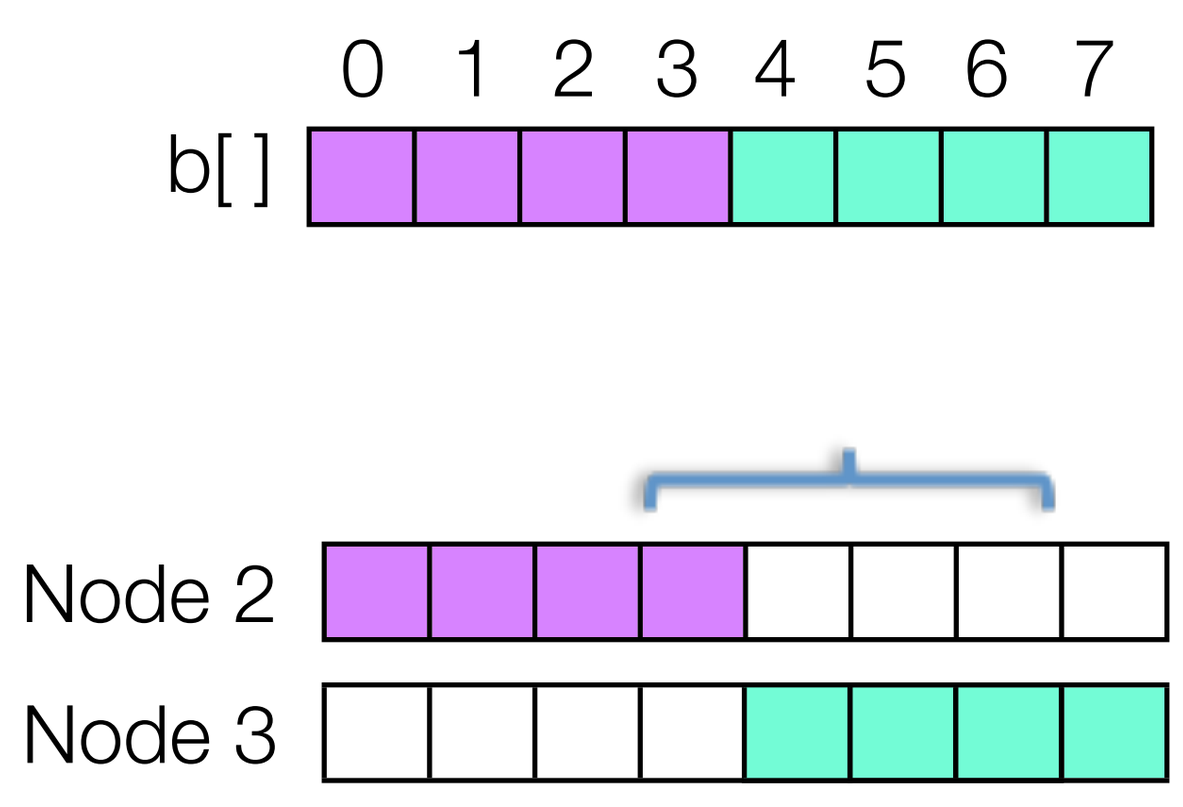
- communication for distributed array
  - We extend “array section notation” in C
  - Programmer doesn’t need to know where each data is distributed

base

length

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





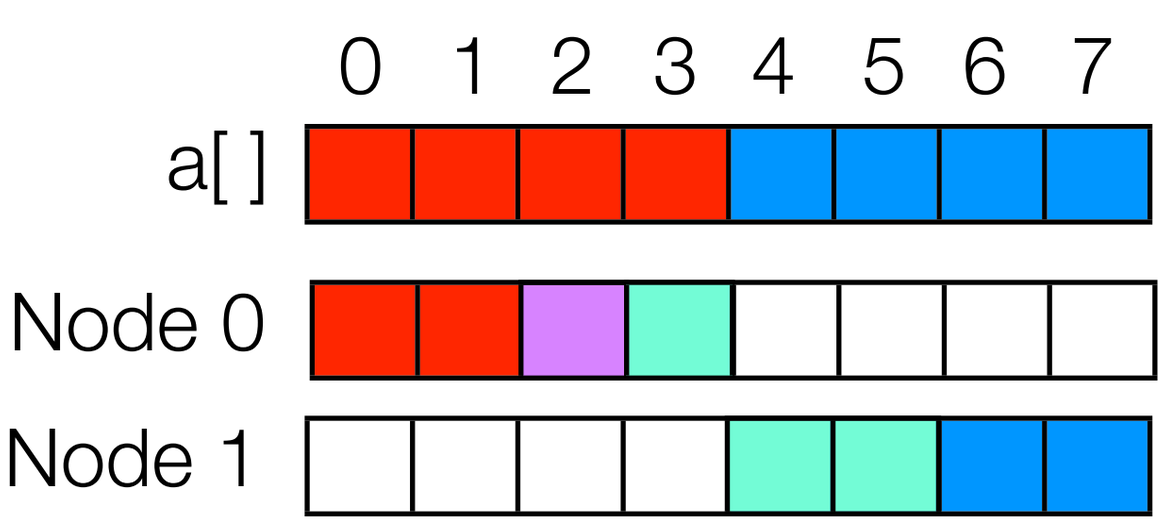
Gmove (Global View Communication)

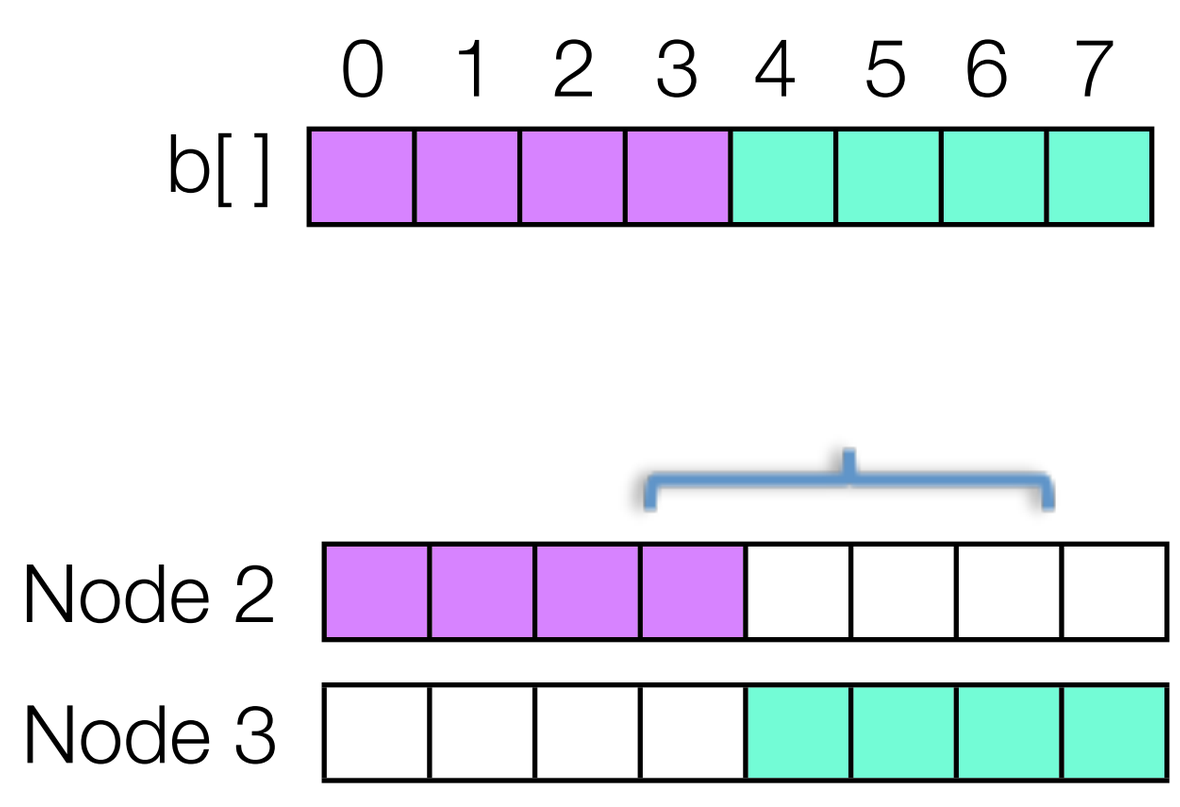
- communication for distributed array
  - We extend “array section notation” in C
  - Programmer doesn’t need to know where each data is distributed

base

length

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





XMP Global view directives

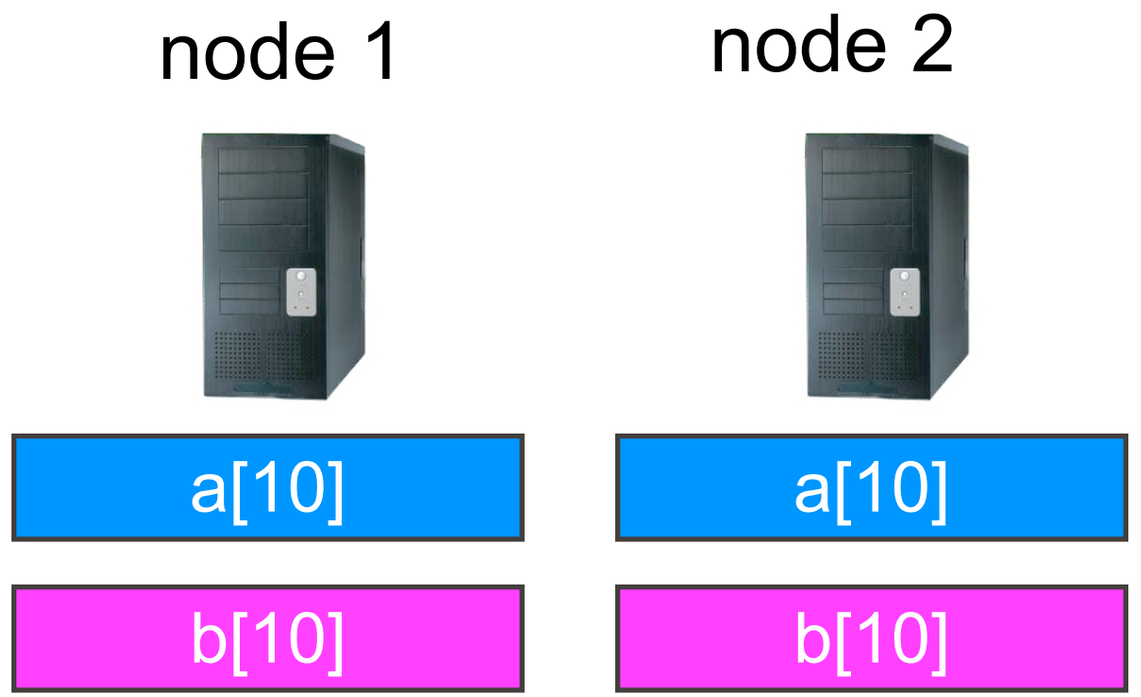
- Broadcast
  - #pragma xmp bcast (var)
- Reduction
  - #pragma xmp reduction (op: var)
- Barrier
  - #pragma xmp barrier

Local View Model

- We adopt Co-Array as PGAS (Partitioned Global Address Space) feature
- One-sided communication for local data(Put/Get)
- High interoperability with MPI

```
#pragma xmp coarray b
 . . .
a[0:4] = b[3:4]:[2];
```

indicate node number

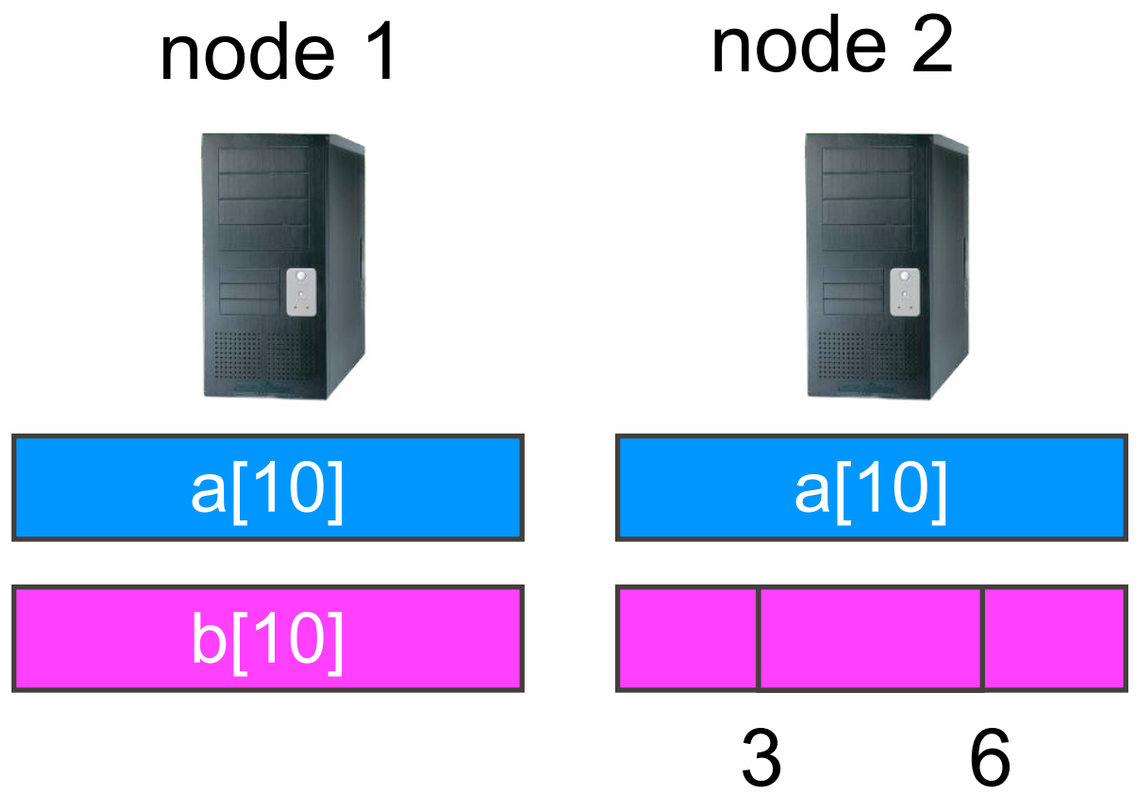


Local View Model

- We adopt Co-Array as PGAS (Partitioned Global Address Space) feature
- One-sided communication for local data(Put/Get)
- High interoperability with MPI

```
#pragma xmp coarray b
 . . .
a[0:4] = b[3:4]:[2];
```

indicate node number

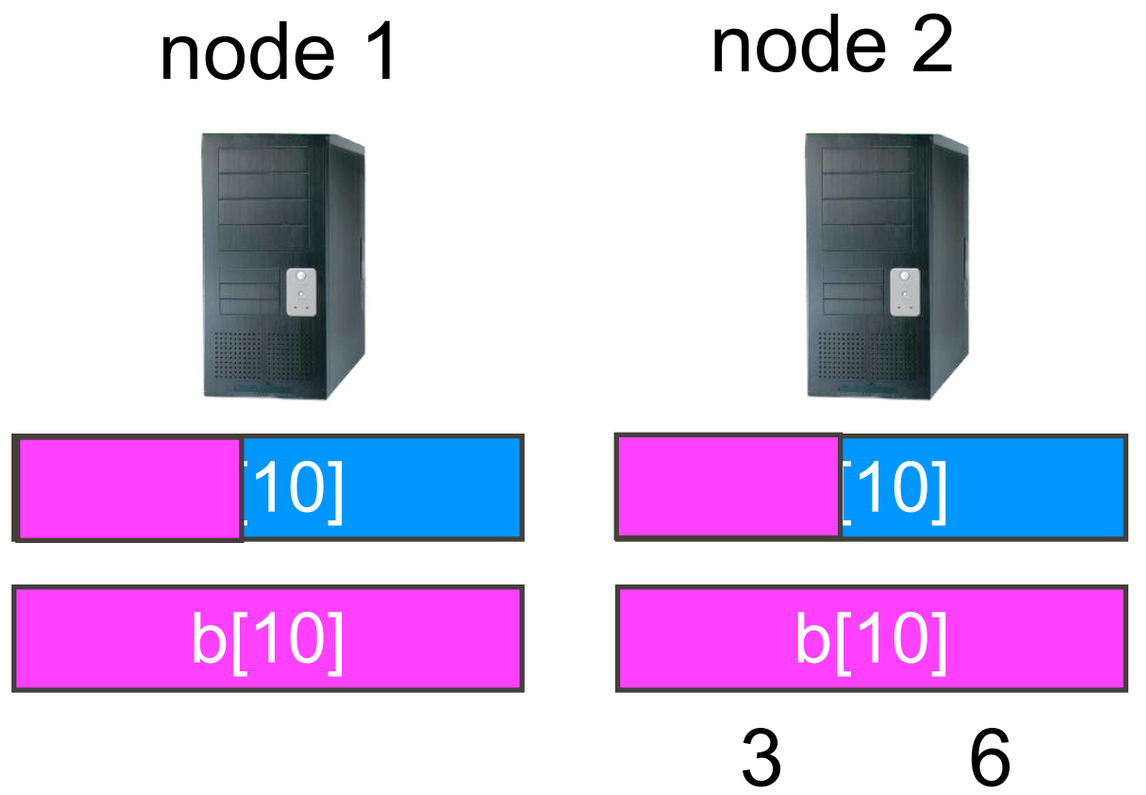


Local View Model

- We adopt Co-Array as PGAS (Partitioned Global Address Space) feature
- One-sided communication for local data(Put/Get)
- High interoperability with MPI

```
#pragma xmp coarray b
 . . .
a[0:4] = b[3:4]:[2];
```

indicate node number

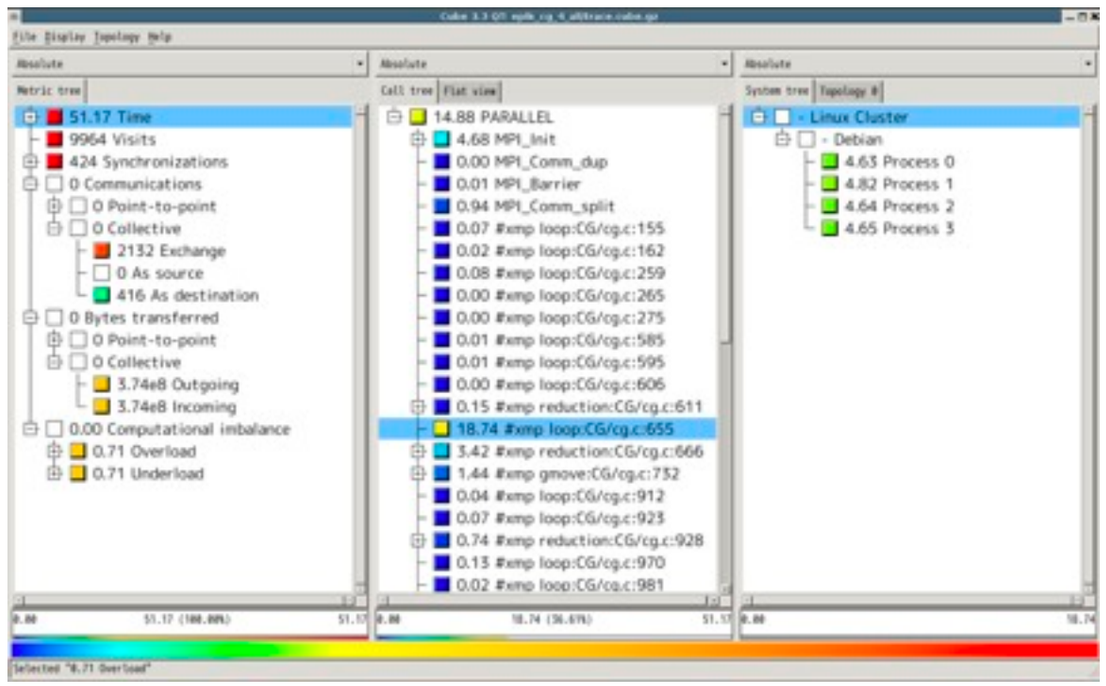


Other Functions

- Easy Hybrid Programming

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

- Interface of scalasca and tlog profiling tools




```
#pragma xmp gmove profile
...
#pragma xmp loop on t(i) profile
```

T2K Open Supercomputer Alliance@SC11

More Information

XcalableMP **Directive-based language eXtension for Scalable and performance-aware Parallel Programming**

日本語 [home] [publications] [download] [projects]

What's XcalableMP

Although MPI is the de facto standard for parallel programming on distributed memory systems, writing MPI programs is often a time-consuming and complicated process. XcalableMP is a directive-based language extension which allows users to develop parallel programs for distributed memory systems easily and to tune the performance by having minimal and simple notations.

The specification is being designed by XcalableMP Specification Working Group which consists of members from academia and research labs to industries in Japan.

The features of XcalableMP are summarized as follows:

- XcalableMP supports typical parallelization based on the data parallel paradigm and work mapping under "global view programming model", and enables parallelizing the original sequential code using minimal modification with simple directives, like OpenMP. Many ideas on "global-view" programming are inherited from HPF (High Performance Fortran).
- The important design principle of XcalableMP is"performance-awareness". All actions of communication and synchronization are taken by directives, different from automatic parallelizing compilers. The user should be aware of what happens by XcalableMP directives in the execution model on the distributed memory architecture.
- XcalableMP also includes a CAF-like PGAS (Partitioned Global Address Space) feature as "local view" programming.
- Extention of existing base languages with directives is useful to reduce rewriting and educational costs. XcalableMP APIs are defined on C and Fortran 95 as a base language.
- For flexibility and extensibility, the execution model allows to combine with explicit MPI coding for more complicated and tuned parallel codes and libraries.
- For multi-core and SMP clusters, OpenMP directives ca
  hybrid programming model.(Under discussion)

XcalableMP is being designed based on the experiences of HI

http://www.xcalablemp.org/

Summary & Future work

- XcalableMP was proposed as a new programming model to facilitate program parallel applications for distributed memory systems
- XcalableMP C language version compiler
  http://www.xcalablemp.org
- Future work
  - For accelerators(GPU, etc)
  - Parallel I/O
  - Interface of MPI library, and so on

Thank you for your attention!!!

Q & A?

Acknowledgements:
We would like to thank XMP-WG members for valuable discussions and comments

T2K Open Supercomputer Alliance@SC11