XcalableMP: Directive-based language extensions for PGAS programming model

Mitsuhisa Sato,
Team Leader of Programing environment Team
AICS RIKEN, Japan

Brief History of XcalableMP Project

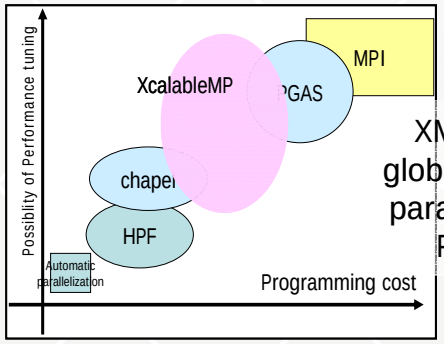
- **2007: XcalableMP Specification Group launched**
- **2008-2011: e-science project**
  - C-front Revised (“gcc” based parser, gnu-extension supported)
  - F95 parser
  - XcodeML for C and F95
  - XcalableMP Spec v 1.0 released
  - Omni XcalableMP compiler for C released
- **2012 - : supported by RIKEN AICS,　2014- : adopted for post-K project**
  - Specification Working Group was (re-) organized under PC cluster Consortium, Japan.
  - Omni XcalableMP compiler for F95 released
  - Omni OpenACC (by U. Tsukuba)
  - 2016: Omni XcalableMP compiler v 1.0

XcalableMP as evolutional approach

- **We focus on migration from existing codes.**
  - Directive-based approach to enable parallelization by adding directives/pragma.
  - Also, should be from MPI code. Coarray may replce MPI.
- **Learn from the past**
  - Global View for data-parallel apps. Japanese community had experience of HPF for Global-view model.
- **Specification designed by community**
  - Spec WG is organized under the PC Cluster Consortium, Japan
- **Design based on PGAS model and Coarray (From CAF)**
  - PGAS is an emerging programming model for exascale!
- **Used as a research vehicle for programming lang/model research.**
  - XMP 2.0 for multitasking.
  - Extension to accelerator (XACC)

XcalableMP(XMP) http://www.xcalablemp.org

- **What's XcalableMP (XMP for short)?**
  - A PGAS programming model and language for distributed memory , proposed by **XMP Spec WG**
  - XMP Spec WG is a special interest group to design and draft the specification of XcalableMP language. It is now organized under **PC Cluster Consortium**, Japan. Mainly active in Japan, but open for everybody.
- **Project status (as of June 2017)**
  - XMP Spec **Version 1.3.1** is available at XMP site. new features: mixed OpenMP and OpenACC , libraries for collective communications.
  - Reference implementation by U. Tsukuba and Riken AICS: **Version 1.3 (C and Fortran90)** is available for PC clusters, Cray XT and K computer. Source-to- Source compiler to code with the runtime on top of MPI and GasNet.
- **HPCC class 2 Winner 2013. 2014**



XMP provides a global view for data parallel program in PGAS model

- Language Features
  - Directive-based language extensions for Fortran and C for PGAS model
  - Global view programming with global-view distributed data structures for data parallelism
    - SPMD execution model as MPI
    - pragmas for data distribution of global array.
    - Work mapping constructs to map works and iteration with affinity to data explicitly.
    - Rich communication and sync directives such as "gmove" and "shadow".
    - Many concepts are inherited from HPF
  - Co-array feature of CAF is adopted as a part of the language spec for local view programming (also defined in C).

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

RIKEN AICS

Example of a Global-view XMP Program

- **Collaboration in Scale project (with Tomita's Climate Science Team)**
- **Typical Stencil Code**

```
!$xmp nodes p(npx,npy,npz)

!$xmp template (lx,ly,lz) :: t
!$xmp distribute (block,block,block) onto p :: t

!$xmp align (ix,iy,iz) with t(ix,iy,iz) ::
!$xmp&        sr, se, sm, sp, sn, sl, ...

!$xmp shadow (1,1,1) ::
!$xmp&        sr, se, sm, sp, sn, sl, ...

      ...

!$xmp reflect (sr, sm, sp, se, sn, sl)

!$xmp loop (ix,iy,iz) on t(ix,iy,iz)
      do iz = 1, lz-1
      do iy = 1, ly
      do ix = 1, lx
         wu0 = sm(ix,iy,iz  ) / sr(ix,iy,iz  )
         wu1 = sm(ix,iy,iz+1) / sr(ix,iy,iz+1)
         wv0 = sn(ix,iy,iz  ) / sr(ix,iy,iz  )
         ...
```

declare a node array

declare and distribute a template

align arrays

add shadow area

stencil communication

parallelize loops

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

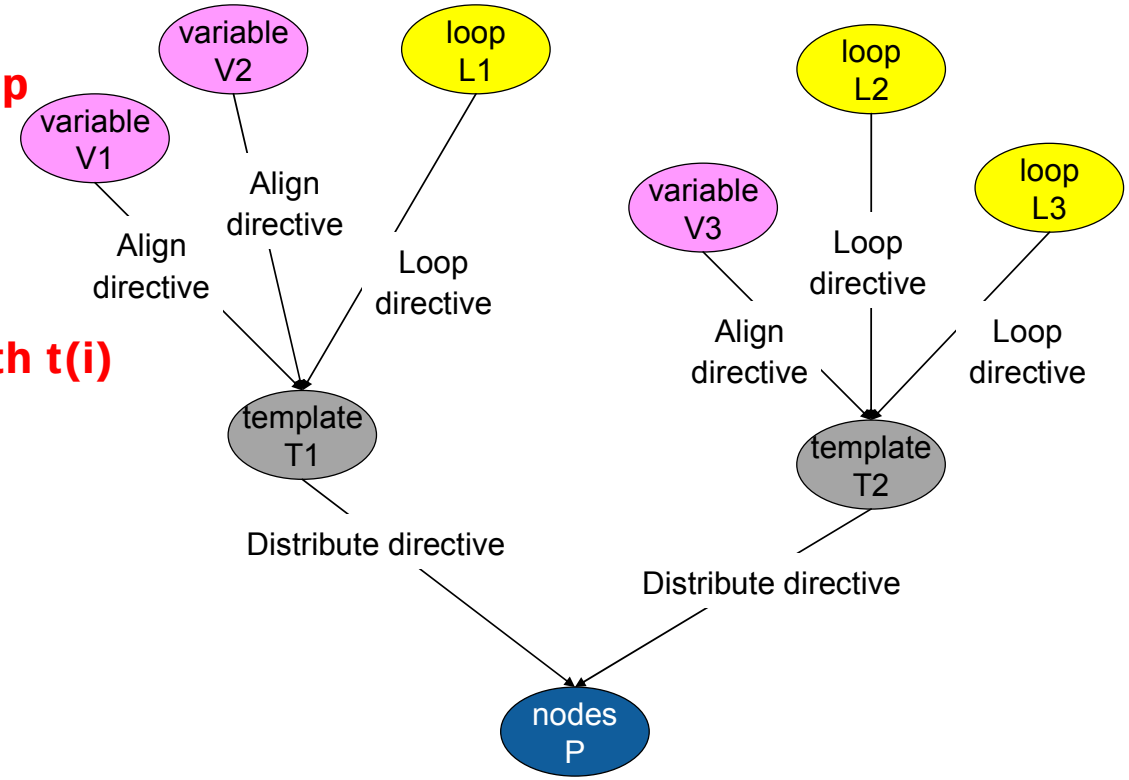
```
#pragma xmp template t(100)
#pragma distribute t(block) onto p
```

- A global data is aligned to the template

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

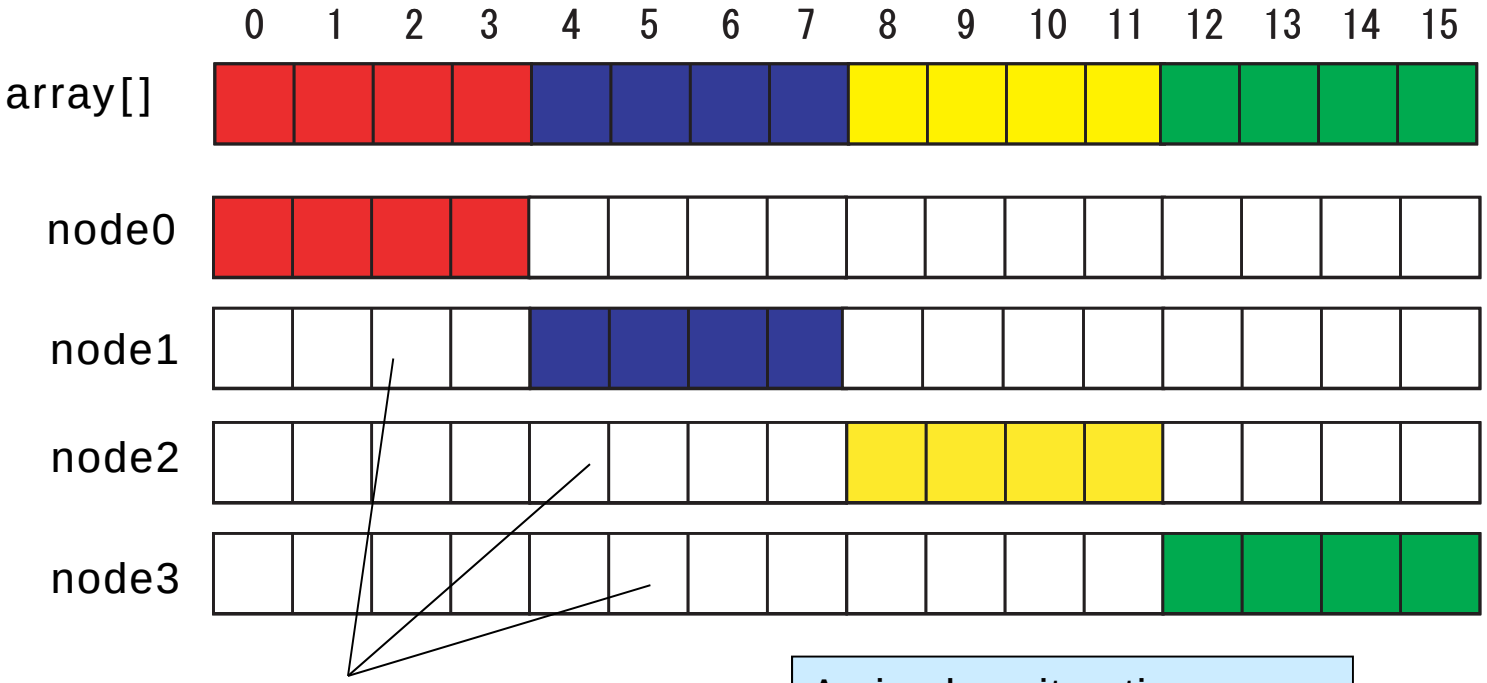
- Loop iteration must also be aligned to the template by on-clause.

```
#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

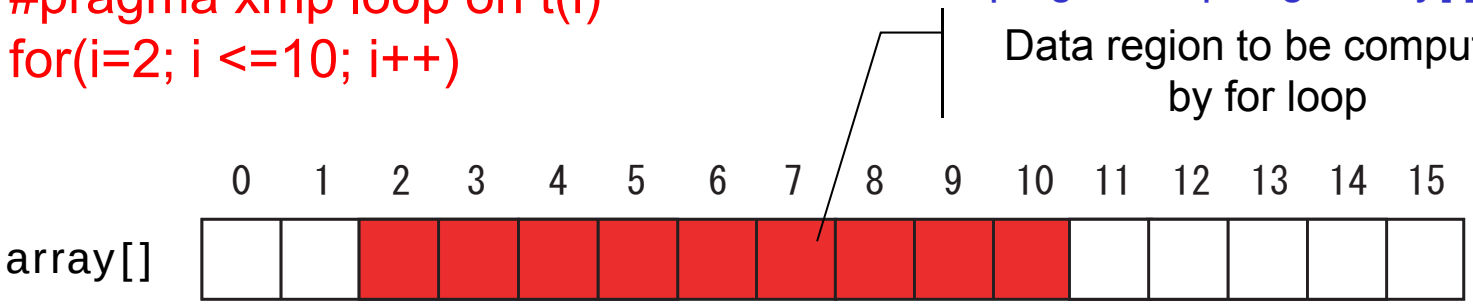
Communicate data between other nodes

Parallel Execution of “for” loop

- Execute for loop to compute on array

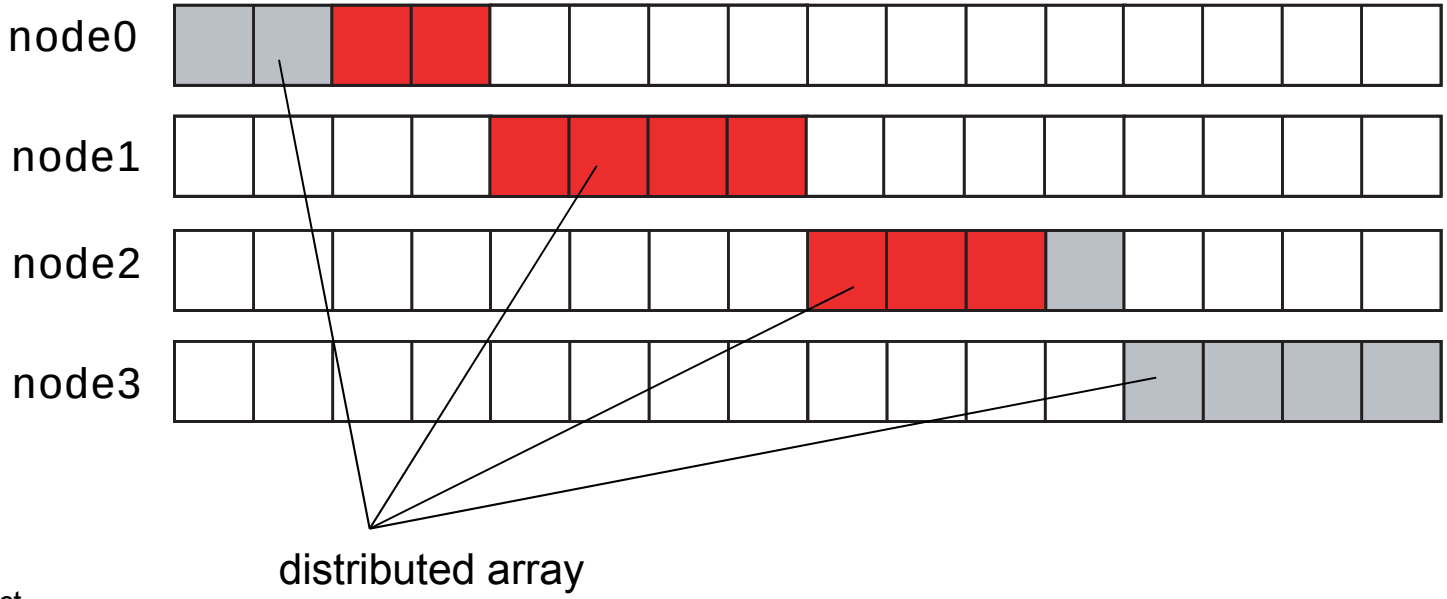
```
#pragma xmp nodes p(*)
#pragma xmp template T(0:15)
#pragma xmp distributed T(block) onto p
#pragma xmp align array[i] with T(i)
```

```
#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:

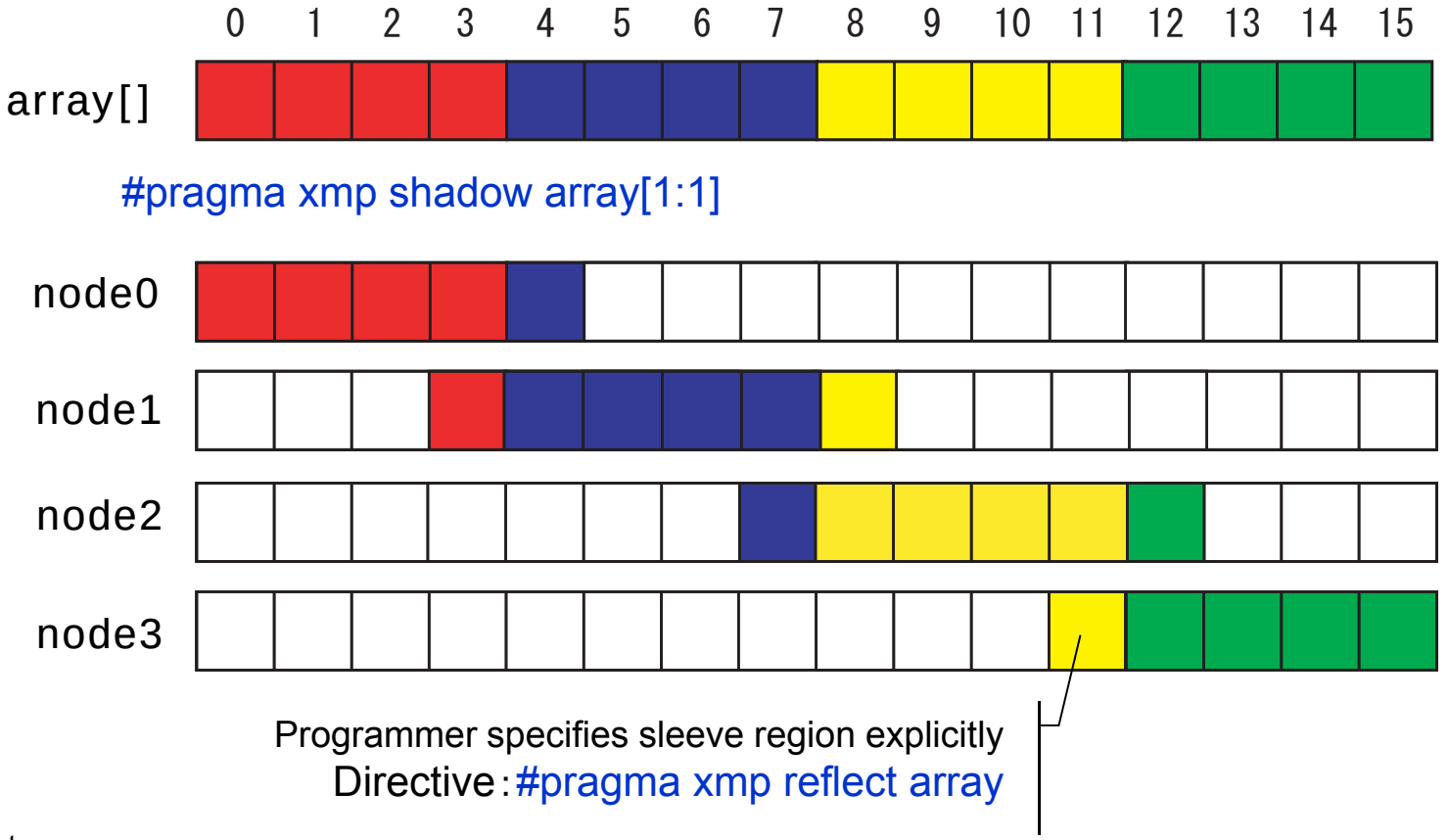
```
#pragma xmp loop on t(i)
```



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)



Local-view XMP program: Coarray

- **XMP includes the coarray feature imported from Fortran 2008 for the local-view programming.**
  - Basic idea: data declared as *coarray* can be accessed by remote nodes.
  - Coarray in XMP/Fortran is fully compatible with Fortran 2008.

```
real b(8)[*]

if (xmp_node_num() == 1) then
  a(:) = b(:)[2]
```

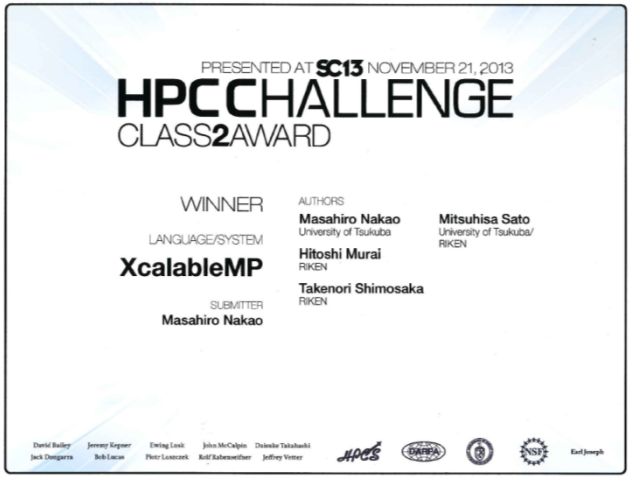
b is declared as a coarray.

Node 1 *gets* b from node 2.

- **Coarrays can be used in XMP/C.**
  - The subarray notation is also available as an extension.
    - Declaration: `float b[8]:[*];`
    - Put: `a[0:3]:[1] = b[3:3];` — *puts* b to node 1.
    - Get: `a[0:3] = b[3:3]:[2];` — *gets* b from node 2.
    - Synchronization: `void xmp_sync_all(int *status)`

Evaluation of productivity and performance of XcalableMP: HPCC Class 2

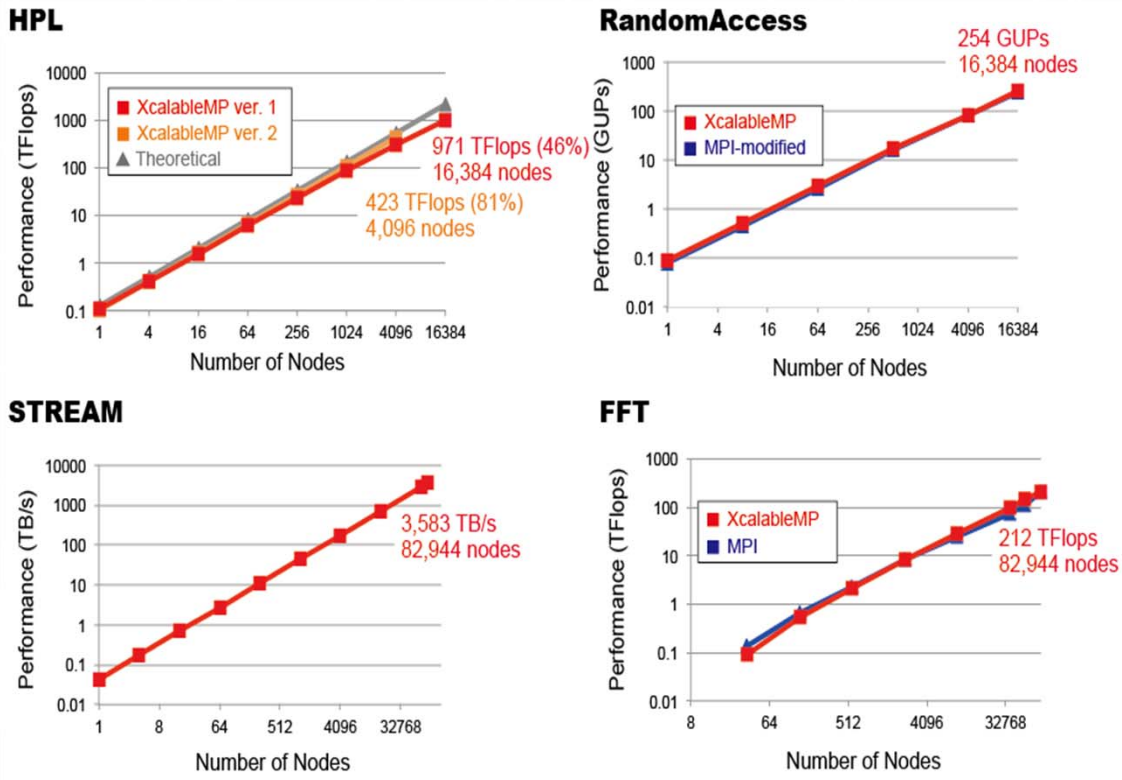
- **We have submitted the results of XcalableMP to the SC13 and SC14 HPC Challenge Benchmark Class2 Competition, and we have awarded the HPC Challenge Class 2 Award at SC13, and the HPC Challenge Class 2 Best Performance Award at SC14.**
  - HPCC Class2 is a competition of parallel programming languages for productivity and performance



Source lines of code of HPCC benchmarks (SC14)

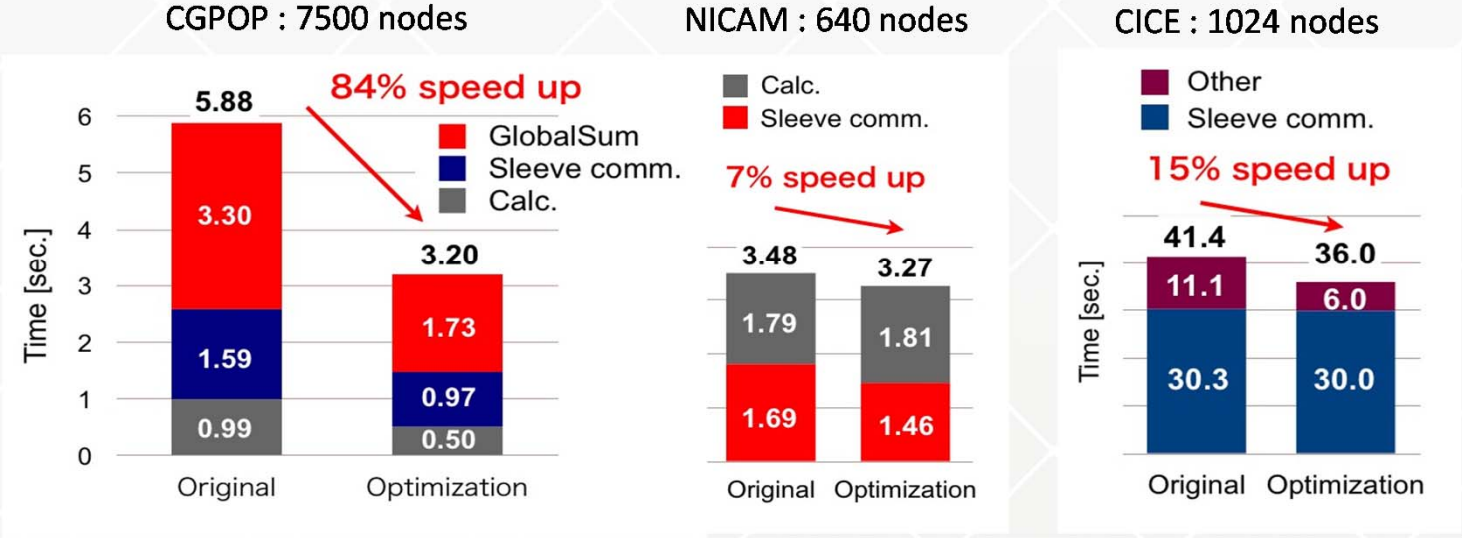
	HPL	FFT	STREAM	RandomAccess
XcalableMP	313	204	69	253
Reference	8,800	787	329	938

Performance results of HPCC benchmarks (SC14)



XcalableMP and Application Studies using PGAS model

- **XMP applications Experience**
  - IMPACT-3D: 3D Eulerian fluid code, which performs compressible and inviscid fluid computation to simulate converging asymmetric flows related to laser fusion (NIFS)
  - RTM code: Reverse-time Migration Method for Remote Sensing applications (Total, France)
  - SCALE-LES: Next-generation Climate Code developed by AICS Tomita's Team
  - GTC-P: Gyrokinetic Toroidal Code , which is a 3D PIC code to study the micro turbulence phenomenon in magnetically connected fusion plasma (Princeton Univ. and Univ. Tsukuba)
- **Case study: G8 ECS : Enabling Climate Simulations at Extreme Scale**
  - CGPOP(Sea) 、NICAM (Cloud) 、CICE (Sea-Ice) on the K computer
    - Usage of RDMA for sync. of sleeve region ⇒ Co-array (in XMP)
    - Optimization of collective comm., thread parallelization (CGPOP)



Current status of XcalableMP project

- **XcalableACC**
  - An “orthogonal” integration of XcalableMP and OpenACC for GPU cluster
- **XcalableMP 2.0**
  - Multitasking model integrated with PGAS model
- **Design of low-level communication layer for PGAS**

XcalableACC(ACC) = XcalableMP+OpenACC+α

- **Extension of XcalableMP for GPU**
  - A part of JST-CREST project "Research and Development on Unified Environment of Accelerated Computing and Interconnection for Post-Petascale" of U. Tsukuba leaded by Prof. Taisuke Boku
  - An "orthogonal" integration of XcalableMP and OpenACC
    - Data distribution for both host and GPU by XcalableMP
    - Offloading computations in a set of nodes by OpenACC
    - "+α" is notation for direct communication between GPUs
  - Proposed as unified parallel programming model for many-core architecture & accelerator
    - GPU, Intel Xeon Phi
    - OpenACC supports many architectures

Source Code Example: NPB CG

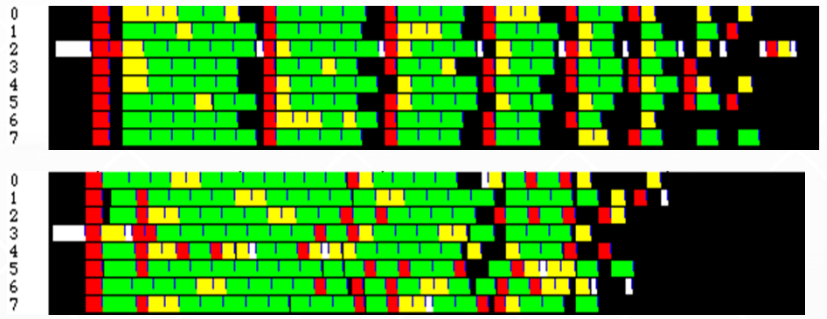
```
#pragma xmp nodes p(NUM_COLS, NUM_ROWS)
#pragma xmp template t(0:NA-1,0:NA-1)
#pragma xmp distribute t(block, block) onto p
#pragma xmp align w[i] with t(*,i)
#pragma xmp align q[i] with t(i,*)
double a[NZ];
int rowstr[NA+1], colidx[NZ];
…
#pragma acc data copy(p,q,r,w,rowstr[0:NA+1]¥
                                , a[0:NZ], colidx[0:NZ])
{
  …
#pragma xmp loop on t(*,j)
#pragma acc parallel loop gang
  for(j=0; j < NA; j++){
    double sum = 0.0;
#pragma acc loop vector reduction(+:sum)
    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(:,*) acc
#pragma xmp gmove acc
  q[:] = w[:];
  …
} //end acc data
```

Towards “XcalableMP 2.0”

- **Specification v 1.2, v1.3:**
  - Support for Multicore: hybrid XMP and OpenMP is defined.
  - Dynamic allocation of distributed array
  - ・・・
- **A set of spec in version 1 is now “converged”. New functions should be discussed for version 2.**
- **Main topics for XcalableMP 2.0: Support for manycore**
  - Multitasking with integrations of PGAS model
  - Synchronization models for dataflow/multitasking executions
  - Proposal: tasklet directive
    - Similar to OpenMP task directive
    - Including inter-node communication on PGAS

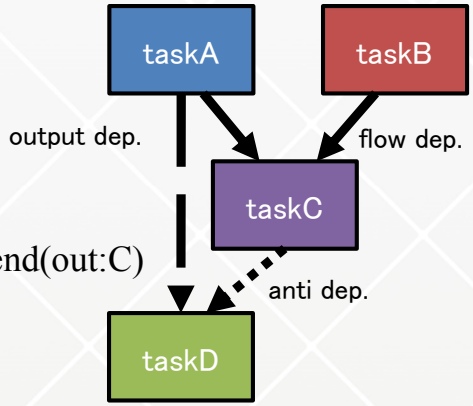
Task in OpenMP 4.0

- Task directive in OpenMP4.0 creates a task with dependency specified “depend” clause
  - The task dependency depends on the order of reading and writing to data based on the sequential execution.
- Can remove barrier which is costly in large scale manycore system.



Execution results of Cholesky Factorization in OpenMP 4.0 task

```
#pragma omp parallel
#pragma omp single
{
   int A, B, C;
#pragma omp task depend(out:A)
   A = 1;         /* taskA */
#pragma omp task depend(out:B)
   B = 2;         /* taskB */
#pragma omp task depend(in:A, B) depend(out:C)
   C = A + B;  /* taskC */
#pragma omp task depend(out:A)
   A = 3;         /* taskD */
}
```



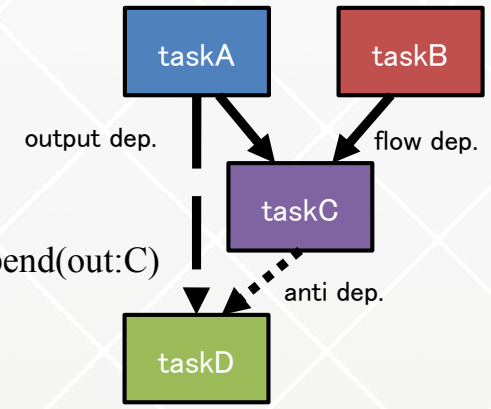
Extension of Multitasking model to between nodes

- Multitasking/Multithreaded execution between nodes : many “tasks” are generated/executed and communicates with each others by data dependency.
  - The task dependency depends on the order of reading and writing to data based on the sequential execution.
    - OpenMP multi-tasking model cannot directly be applied to tasks running in different nodes since threads of each nodes are running in parallel.
  - In OmpSs, interactions between nodes are described through the MPI task that is executing MPI communications (MPI task). OmpSs Cluster model offers other programming model.
  - Otherwise, run the same task program in all nodes to resolve dep. (StarPU)
- Advantages
  - Thread-to-thread synchronization /communications rather than barrier
  - Remove barrier which is costly in large scale manycore system.
  - Overlap of computations and computation is done naturally.
- A Proposal of Tasklet directive in XMP 2.0
  - Integration with Multitasking and PGAS

Task in OpenMP and extension to between nodes

- Task directive in OpenMP4.0 creates a task with dependency specified “depend” clause
- The task dependency depends on the order of reading and writing to data based on the sequential execution.
  - OpenMP multi-tasking model cannot be applied to tasks running in different nodes since threads of each nodes are running in parallel.
  - In OmpSs, interactions between nodes are described through the MPI task that is executing MPI communications (MPI task).
  - Otherwise, run the same task program in all nodes to resolve dep. (StarPU)

```
#pragma omp parallel
#pragma omp single
{
   int A, B, C;
#pragma omp task depend(out:A)
   A = 1;          /* taskA */
#pragma omp task depend(out:B)
   B = 2;          /* taskB */
#pragma omp task depend(in:A, B) depend(out:C)
   C = A + B;  /* taskC */
#pragma omp task depend(out:A)
   A = 3;          /* taskD */
}
```



- Flow dependency: The flow dependency occurs between dependence-type out and in with same variables, similar to read after write (RAW) consistency. It is shown in between taskA and taskC with variable *A*, or taskB and taskC with variable *B*.
- Anti dependency: The anti dependency occurs between dependence-type in and out with same variables, similar to write after read (WAR) consistency. It is shown in between taskC and taskD with variable *A*.
- Output dependency: The output dependency occurs between dependence-type out and out with same variables, similar to write after write (WAW) consistency. It is shown in between taskA and taskC with variable *A*.

Multitasking in XMP: our proposal

- **For Intra-node**
  - Tasklet directive creates a task with dependency specified "in/out/inout" clause in sequential order
  - Same as in OpenMP4.0 and OMPss
- **For Inter-node**
  - Describe communications by PGAS operations (Coarray and gmove)
  - Annotated by get/put and get_ready/put_ready clauses to specify dependency.

`#pragma xmp tasklet` *tasklet-clause[, tasklet-clause[, ...]]* on {*node-ref* | *template-ref*}
(structured-block)

where *tasklet-clause* is :
{`in` | `out` | `inout`} (*variable[, variable, ...]*)
or
{`put` | `get`} (*tag*)
or
{`put_ready` | `get_ready`} (*variable*, {*node-ref* | *template-ref*}, *tag*)

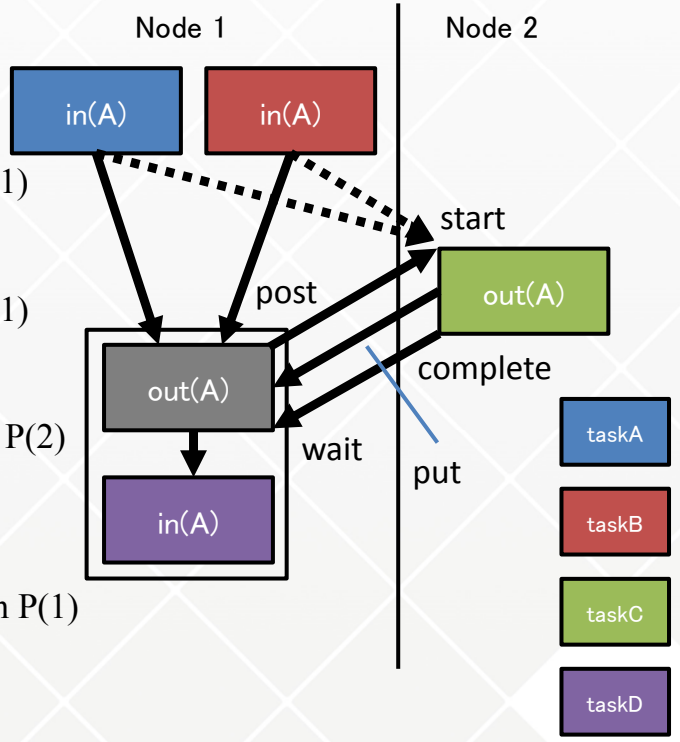
`#pragma xmp taskletwait` *[on* {*node-ref* | *template-ref*}*]*

Put operation in tasklet

- put_ready clause: indicates that the specified data may be written by the associated PUT operation
  - This clause has the dependence-type out for the specified data on a node since its values are overwritten by the remote node.
- put clause: indicates that the PUT operation may be performed in the associated structured block.
  - At the beginning of the block, the task waits to receive the post notification with the tag by the put_ready clause to indicates that the data is exposed in the target node for the PUT operations.

- When output dependencies for the data are satisfied before executing the block, the clause exposes the data for the PUT operation from the specified set of nodes by sending the post notifications to these nodes, starting the PUT operations eventually in remote nodes. Then, it waits until remote operations are done. When the task receives the completion notification of the PUT operation, the block is immediately scheduled.
- When the post notification is received, the task is scheduled to execute the calculation and PUT operation in the block. When the execution of the block is finished, the data written by the PUT operation is flushed and the completion notification is sent to the node matched by the tag.

```
#pragma xmp nodes P(2)
   int A:[*], B, C, D, tag;

#pragma xmp tasklet in(A) out(B) on P(1)
   B = A;        /* taskA */

#pragma xmp tasklet in(A) out(C) on P(1)
   C = A;        /* taskB */

#pragma xmp tasklet out(A) put(tag) on P(2)
   A:[1] = 1;    /* taskC */

#pragma xmp tasklet in(A) out(D) ¥
                 put_ready(A, P(1), tag) on P(1)
   D = A;        /* taskD */
```

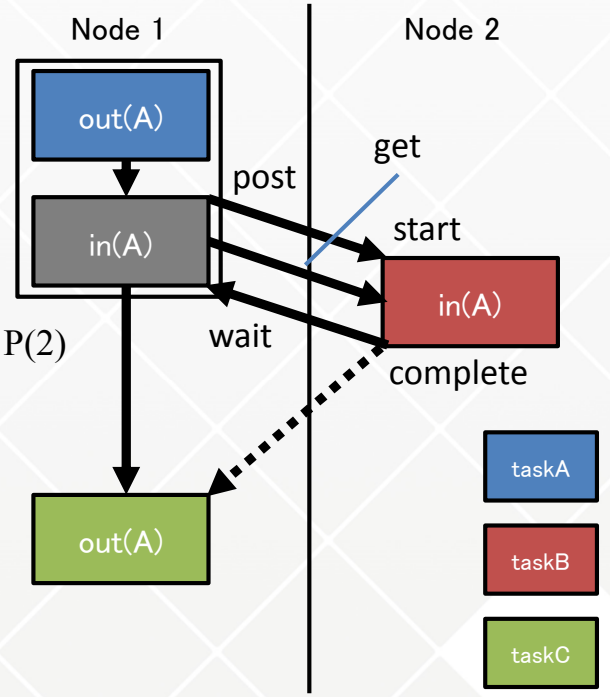


Get operation in tasklet

- get_ready clause: indicates that the specified data may be read by the associated GET operation.
  - When the execution of the block is finished, the post notification is sent to the specified set of nodes to indicate that the value of the data is ready to be accessed, starting the GET operations eventually in remote nodes.
- get clause: indicates that the GET operation may be performed in the associated structured block.
  - Before executing the block, the execution is postponed until the post notification by the get_ready clause from matching task by the tag is received.

- This clause has the dependence-type in for the specified data in a node since its values are to be read by the remote node. Before executing the blocks which have anti-dependency to the data, all completion notifications from remote nodes executing GET operations must be received. Otherwise, the dependent tasks are blocked until all completion notifications are received.
- When the notification is received, the task is scheduled to execute the calculation and GET operation in the block. When the execution of the block is finished, the completion notification is sent to the node matched by the tag.

```
#pragma xmp nodes P(2)
   int A:[*], B, tag;

#pragma xmp tasklet out(A) ¥
                get_ready(A, P(2), tag) on P(1)
   A = 0;          /* taskA */

#pragma xmp tasklet in(A) out(B) get(tag) on P(2)
   B = A:[1];    /* taskB */

#pragma xmp tasklet out(A) on P(1)
   A = 1;          /* taskC */
```



Another directive: tasklet gmove

- **Tasklet gmove: Combined directive with tasklet and gmove**
  - Enable to generate communications with send/recv rather than put/get.
  - Compiled into send/recv
- **Tasklet reflect: Combined directive with tasklet and reflect**

#pragma xmp tasklet gmove *[clause[, clause] ...] [on { node-ref | template-ref }]*
(an assignment statement)

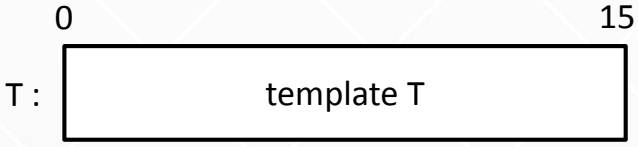
#pragma xmp tasklet reflect *(array-name[, array-name] ...)*
[chunksize (refect-chunksize[, refect-chunksize] ...)]

where clause is :
{in | out | inout} *(variable[, variable] ...])*

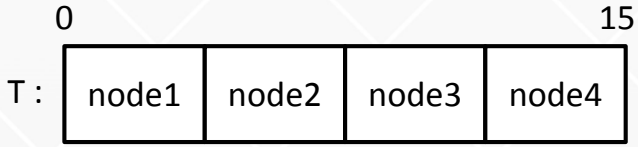
Gmove directive in XMP

- **Copy data in global address space**
  - Collective, but one-sided if in/out specified

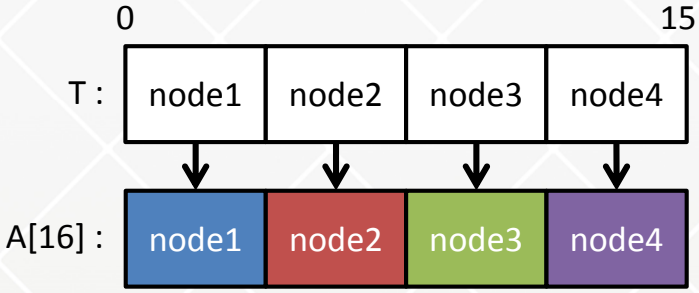
```
int A[16], res;
#pragma xmp nodes P(4)
#pragma xmp template T(0:15)
```




```
#pragma xmp distribute T(block) onto P
```

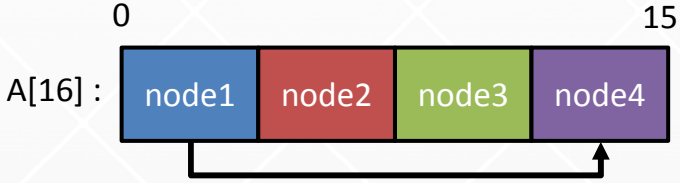



```
#pragma xmp align A[i] with T(i)
```

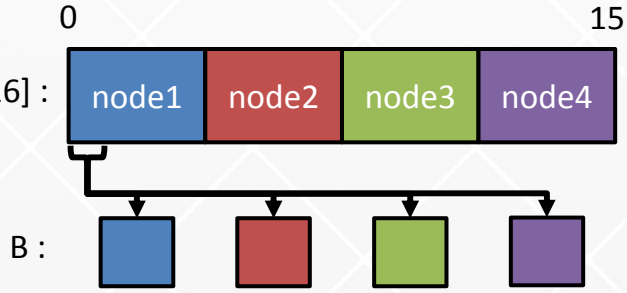



```
#pragma xmp loop(i) on T(i) reduction(+:res)
for (int i = 0; i < 16; i++) {
  A[i] = func(i);
  res += A[i];
```

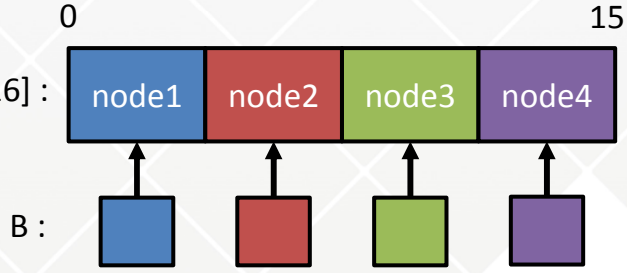
```
int B;  /* B is a local variable */

/* send-recv */
#pragma xmp gmove
  A[12:4] = A[0:4];
```




```
/* broadcast */
#pragma xmp gmove
  B = A[0:1];
```




```
/* local copy */
#pragma xmp gmove
  A[0:16] = B;
```



How to translate XMP2.0 multitasking directives

- **Two designs to be evaluated**
  - Translation to code using OpenMP
  - Translation to code calling the runtime using Argobots.

- **Translation to OpenMP+MPI**
  - Each tasklet directive is translated into OpenMP “task” directive.
  - Using XMP runtime calling MPI replaced by non-blocking operations.

```
MPI_Send(…);
```

or

```
MPI_Isend(…)
;
MPI_Wait(…);
```

```
MPI_Isend(…);
MPI_Test(&comp, …);
while (!comp) {
#pragma omp taskyield
    MPI_Test(&comp,
…);
}
```

Test the completion

Switch to other tasks

Performance Evaluation

- **Platform: Oakforest-PACS @ JCAHPC: A virtual Joint organization of U. Tsukuba and U. Tokyo**
  - KNL-based system (8208 nodes, 25PF peak)
  - Flat and Quadrant mode

CPU	Intel Xeon Phi 7250 1.4 GHz 68 cores
Memory	16 GB (MCDRAM) + 96 GB (DDR 4)
Interconnect	Intel Omni-Path Architecture
Compiler	Intel Compiler 17.0.1 Intel MPI Library 2017 Update 1 Intel MKL 2017 Update 1 OmpSs 16.06.3

- **Benchmark Program:**

  Blocked Cholesky Factorization
  - Problem size: 32k × 32k, block size : 512 × 512
  - 32 nodes(max), 64 threads/node

Example

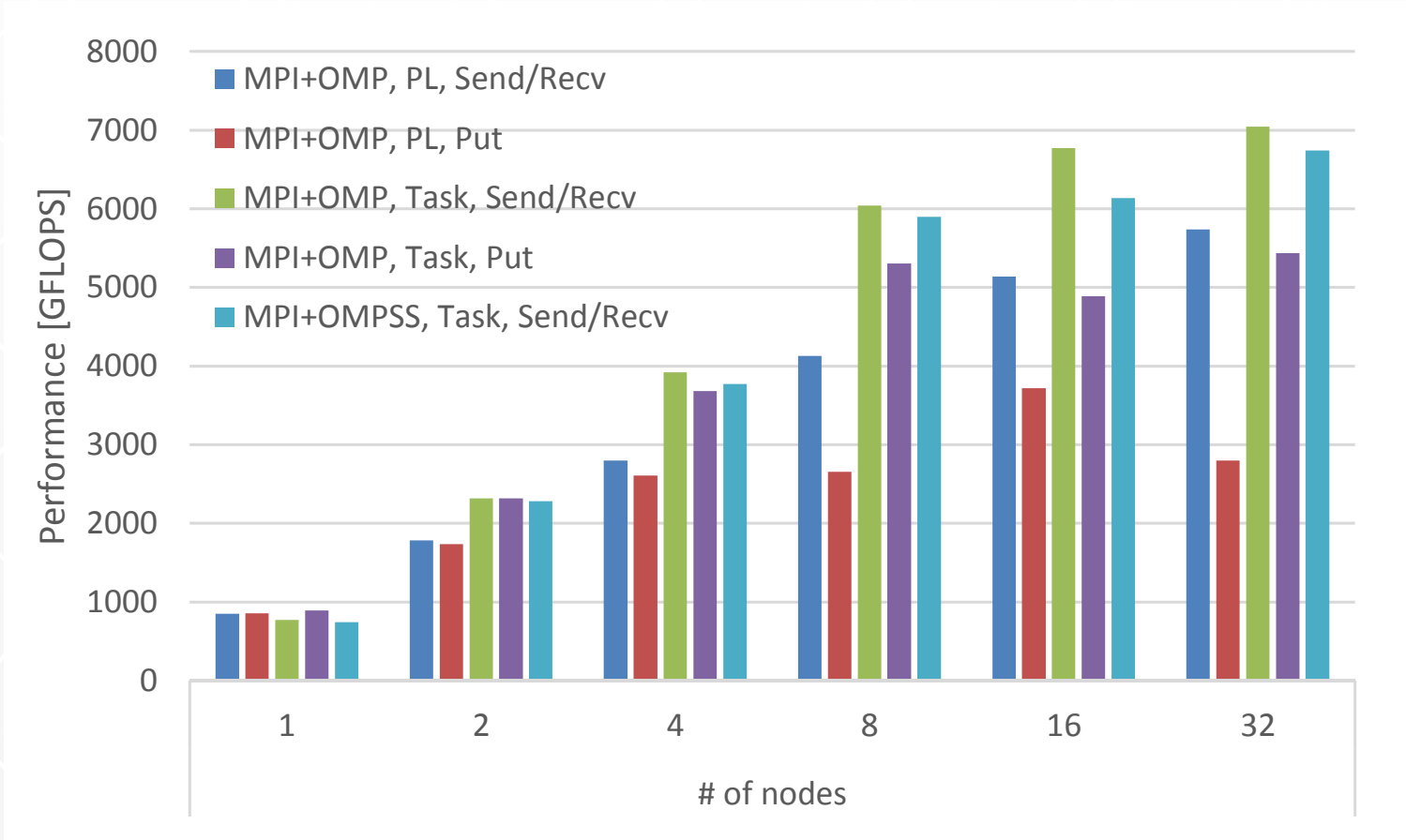
- **Block Cholesky Factorization**

```
double A[nt][nt][ts*ts], B[ts*ts], C[nt][ts*ts];
#pragma xmp nodes P(*)
#pragma xmp template T(0:nt-1)
#pragma xmp distribute T(cyclic) onto P
#pragma xmp align A[*][i][*] with T(i)
```

```
 for (int k = 0; k < nt; k++) {
#pragma xmp tasklet out(A[k][k]) ¥
                        get ready(A[k][k], T(k:), k*nt+k) on T(k)
   potrf(A[k][k]);

#pragma xmp tasklet in(A[k][k]) out(B) get(k*nt+k) on T(k:)
#pragma xmp gmove in
   B[:] = A[k][k][:];

   for (int i = k + 1; i < nt; i++) {
#pragma xmp tasklet in(B) out(A[k][i]) ¥
                    get ready(A[k][i], T(i:), k*nt+i) on T(i)
     trsm(A[k][k], A[k][i]);
   }
   for (int i = k + 1; i < nt; i++) {
#pragma xmp tasklet in(A[k][i]) out(C[i]) get(k*nt+i) on T(i:)
#pragma xmp gmove in
     C[i][:] = A[k][i][:];

     for (int j = k + 1; j < i; j++) {
#pragma xmp tasklet in(A[k][i], C[j]) out(A[j][i]) on T(j)
       gemm(A[k][i], C[j], A[j][i]);
     }
#pragma xmp tasklet in(A[k][i]) out(A[i][i]) on T(i)
     syrk(A[k][i], A[i][i]);
   }
 }
#pragma xmp taskletwait
#pragma xmp barrier
```

Results

- Comparison with “Parallel Loop” (PL) and Task-based
- “Put” and “Send/Recv”
- (OMPSS)

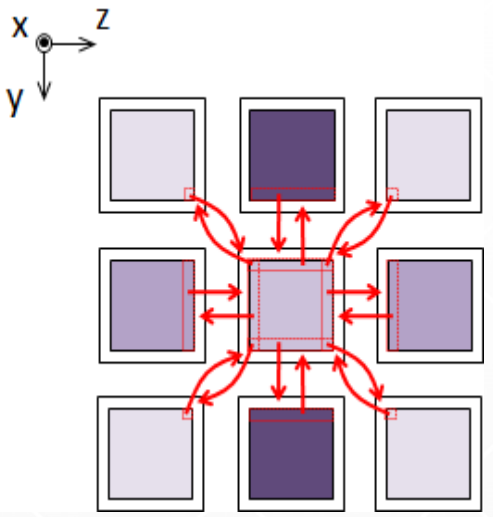


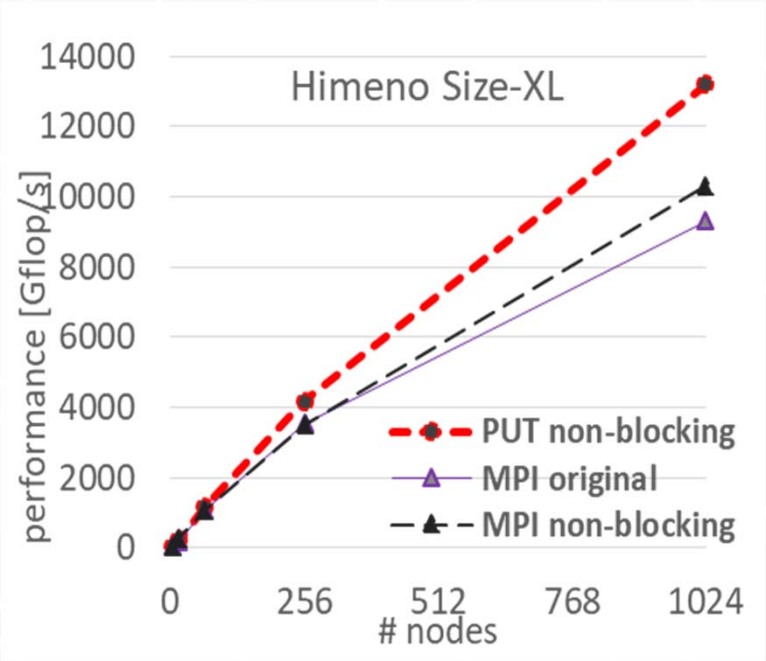
Can PGAS replace MPI? / Can PGAS be faster than MPI?

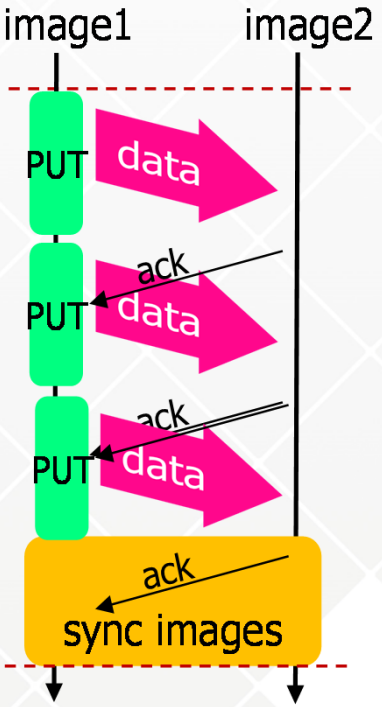
- Advantages of RMA/RDMA Operations
  - (Note: Assume MPI RMA is an API for PGAS)
  - multiple data transfers can be performed with a single synchronization operation
  - Some irregular communication patterns can be more economically expressed
  - Significantly faster than send/receive on systems with hardware support for remote memory access
    - Recently, many kinds of high-speed interconnect have hardware support for RDMA, including Infiniband, … as well as Cray and Fujitsu.

Case study: stencil communication

- Typical communication pattern in domain-decomposition.
- Advantage of PGAS: Multiple data transfers with a single synchronization operation at end
- PUT non-blocking outperforms MPI in Himeno Benchmark!
  - Don’t wait ack before sending the next data (by FJ-RDMA)





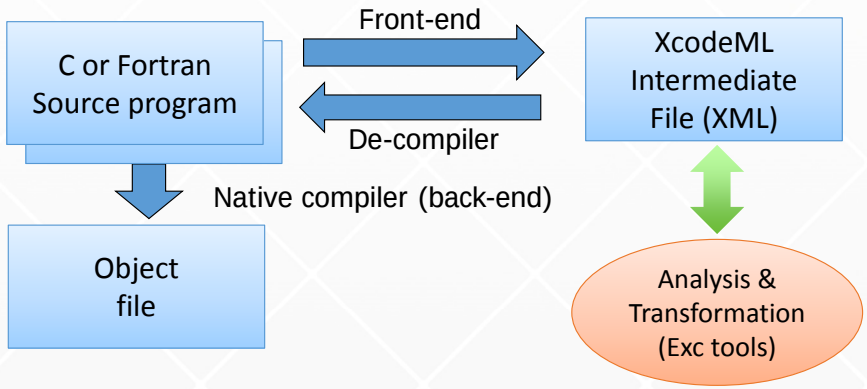


NOTE: The detail of this results is presented in HPCAisa 2018: Hidetoshi Iwashita, Masahiro Nakao, Hitoshi Murai, Mitsuhisa Sato, “A Source-to-Source Translation of Coarray Fortran with MPI for High Performance”

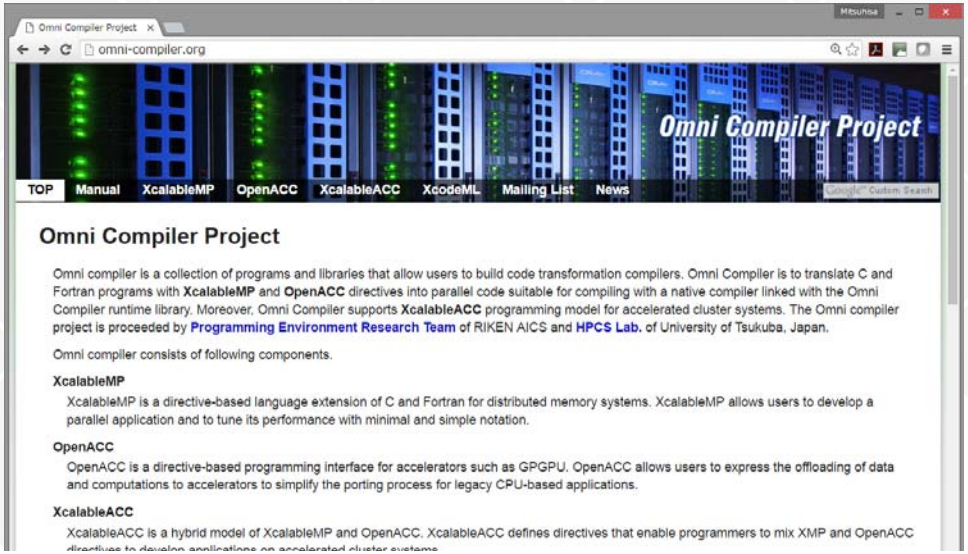
Omni Compiler infrastructure project

- **Omni compiler is a collection of programs and libraries that allow users to build code transformation compilers.**
- **Omni Compiler is to translate C and Fortran programs with directive-based extensions such as XcalableMP and OpenACC directives into codes compiled by a native back-end compiler linked with runtime libraries.**
- **Supported directives:**
  - OpenMP (C, F95)
    - OpenMP 2.x and a little bit old, but a new implementation of OpenMP 4.x planned.
  - OpenACC (C)
  - XcalableMP (C, F95)
  - XcalableACC (C)

- **The source programs in C and Fortran2003 (C++ planned) are translated into an XML intermediate form called XcodeML by the front-end programs, C-front and F-front.**



http://omni-compiler.org



RIKEN AICS

Current projects and Future plans

- **XcalableMP 2.0**
  - Multitasking model integrated with PGAS model
- **XcalableACC**
  - An “orthogonal” integration of XcalableMP and OpenACC for GPU cluster
- **Design of low-level communication layer for PGAS**

- **C++ front-end based on LLVM clang**
- **XMP API (“compiler-free” approach)**
- **METAX: Meta-programming for Fortran (and C) using Omni compiler**