How does PGAS “collaborate” with MPI+X?

Contributors: members of XMP-WG
Hitoshi Murai (RIKEN), Masahiro Nakao (RIKEN), Jinpil Lee (RIKEN), Tesuya Odajima (RIKEN), Hidetoshi Iwashita(Fujitsu), Hitoshi Sakagami (NIFS), Takeshi Nanri (Kyushu U), Atsushi Hori (RIKEN), Taisuke Boku (U. Tsukuba), Akihiro Tabuchi (U. Tsukuba), Keisuke Tsugane (U. Tsukuba)

Mitsuhisa Sato

Deputy Project Leader and Team Leader, Architecture Development Team, Flagship 2020 Project

Team Leader, Programming Environment Research Team

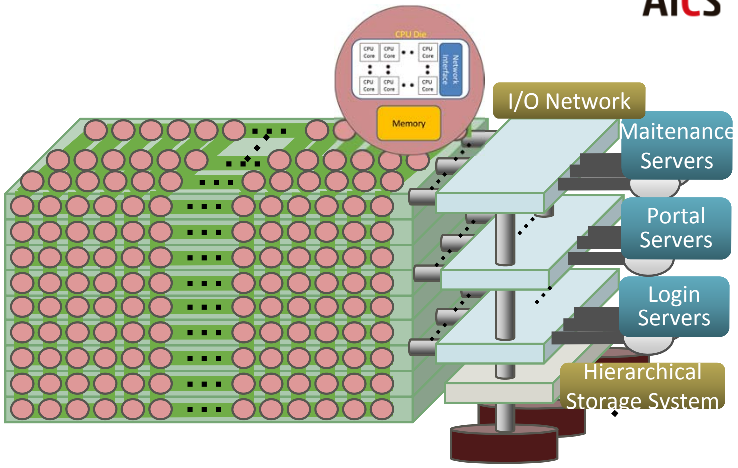
Advanced Institute for Computational Science, RIKEN

Outline of my talk

- **Background**
  - XcalableMP and FLAGSHIP 2020 project

- **Agenda: How does PGAS “collaborate” with MPI+X?**
  - Should everything be written in PGAS?
  - Can PGAS replace MPI? / Can PGAS be faster than MPI?
  - Is RMA good as a underlying comm. layer for PGAS?
  - “PGAS + X” ? : PGAS with multitasking
  - “Compiler-free” approaches for PGAS

FLAGSHIP2020 Project



□ Missions

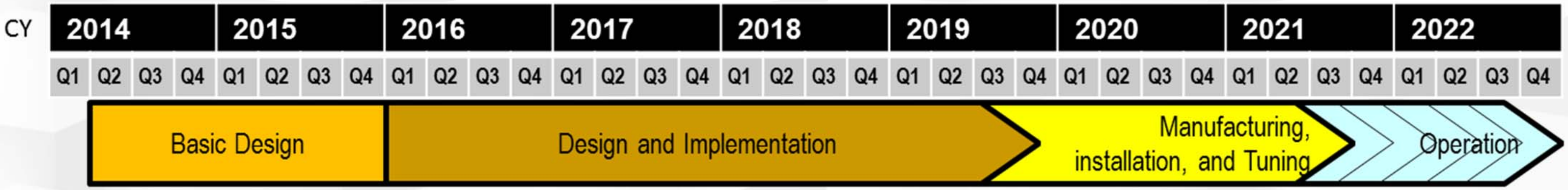
- Building the Japanese national flagship supercomputer, post K, and
- Developing wide range of HPC applications, running on post K, in order to solve social and science issues in Japan

□ Project organization

- Post K Computer development
  - RIKEN AICS is in charge of development
  - Fujitsu is vendor partner.
  - International collaborations: DOE, JLESC, ..
- Applications
  - The government selected 9 social & scientific priority issues and their R&D organizations.

□ Status and Update

- “Basic Design” was finalized and now in “Design and Implementation” phase.
- We have decided to choose ARM v8 with SVE as ISA for post-K manycore processor.
- We are working on detail evaluation by simulators and compilers

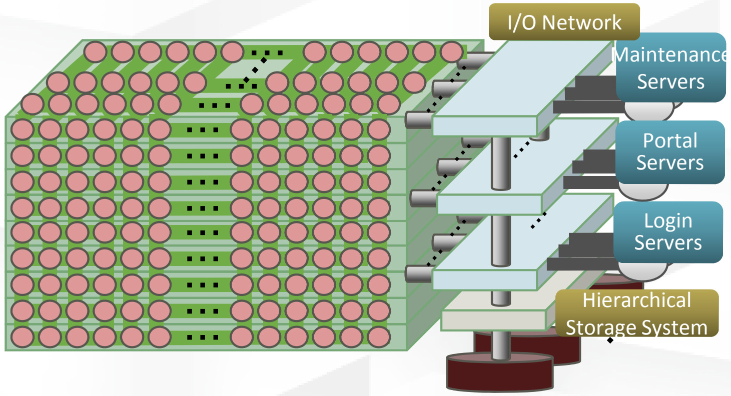


An Overview of post K

MC-kernel: a lightweight Kernel for manycore

Hardware

- Manycore architecture
  - Core: ARM v8 with SVE (512bits)
- 6D mesh/torus Interconnect
  - Tofu-2 interconnect, which support RDMA
- 3-level hierarchical storage system
  - Silicon Disk
  - Magnetic Disk
  - Storage for archive

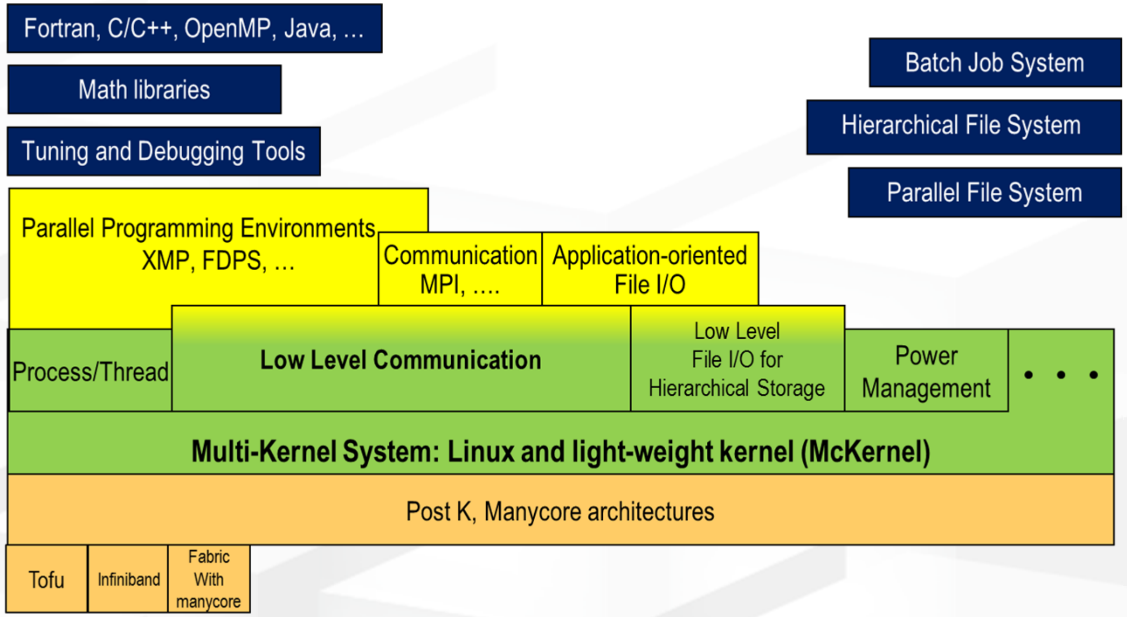


System Software

- Multi-Kernel: Linux with Light-weight Kernel
- File I/O middleware for 3-level hierarchical storage system and application
- Application-oriented file I/O middleware
- MPI+OpenMP programming environment
- **Highly productive programming language and libraries**

XcalableMP PGAS language

FPDS DSL



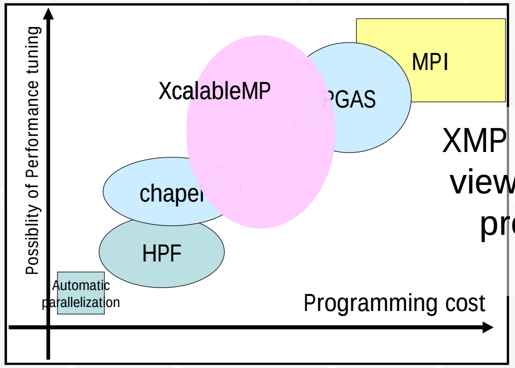
Fortran, C/C++, OpenMP, Java, ...
Math libraries
Tuning and Debugging Tools
Batch Job System
Hierarchical File System
Parallel File System
Parallel Programming Environments XMP, FDPS, ...
Communication MPI, ...
Application-oriented File I/O
Process/Thread
Low Level Communication
Low Level File I/O for Hierarchical Storage
Power Management
• • •
Multi-Kernel System: Linux and light-weight kernel (McKernel)
Post K, Manycore architectures
Tofu
Infiniband
Fabric With manycore

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 2016)**
  - XMP Spec **Version 1.2.1** is available at XMP site. new features: mixed OpenMP and OpenACC , libraries for collective communications.
  - Reference implementation by U. Tsukuba and Riken AICS: **Version 1.0 (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];
   }
}
```

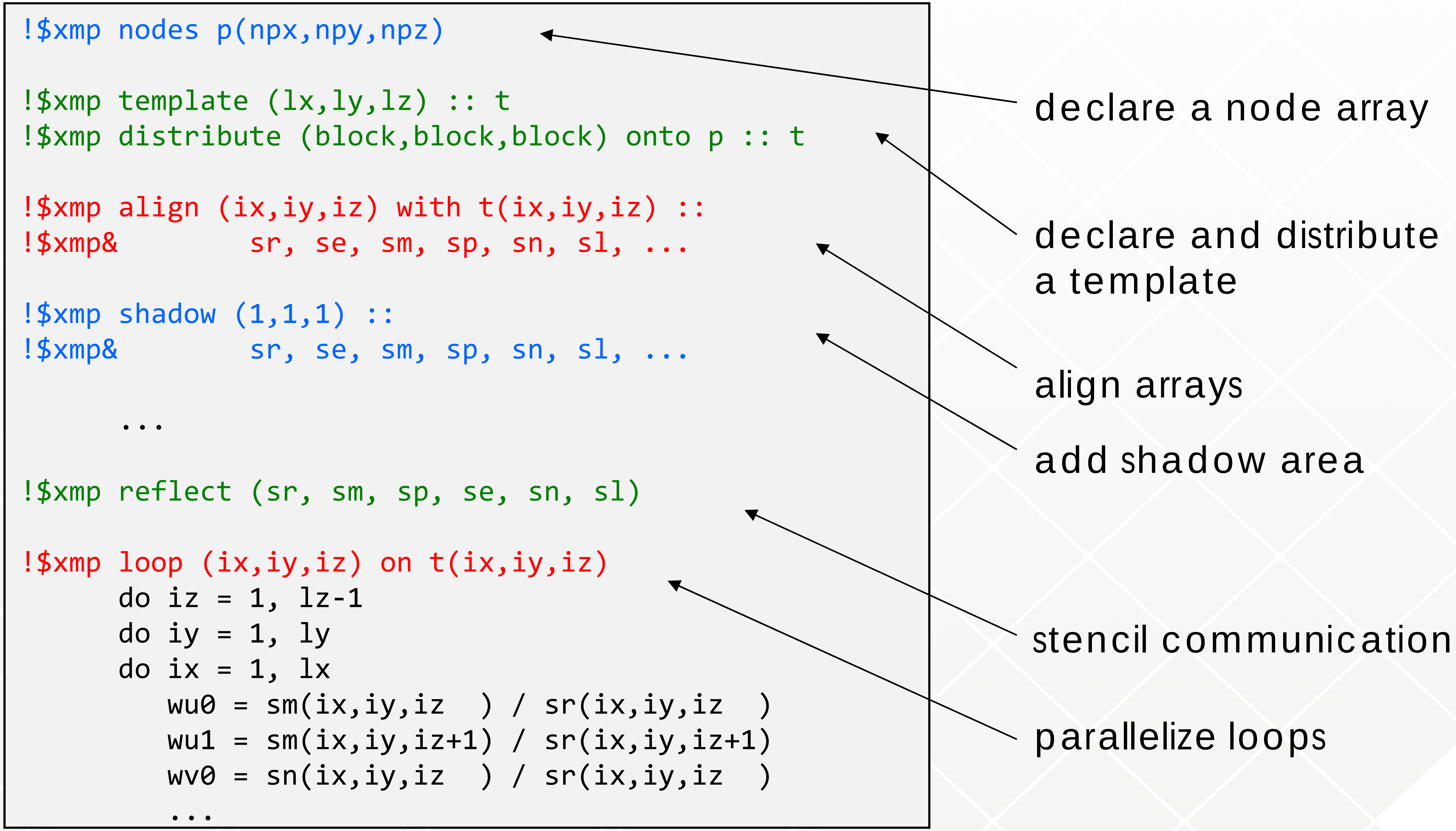
data distribution

add to the serial code : incremental parallelization

work sharing and data synchronization

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

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)
```

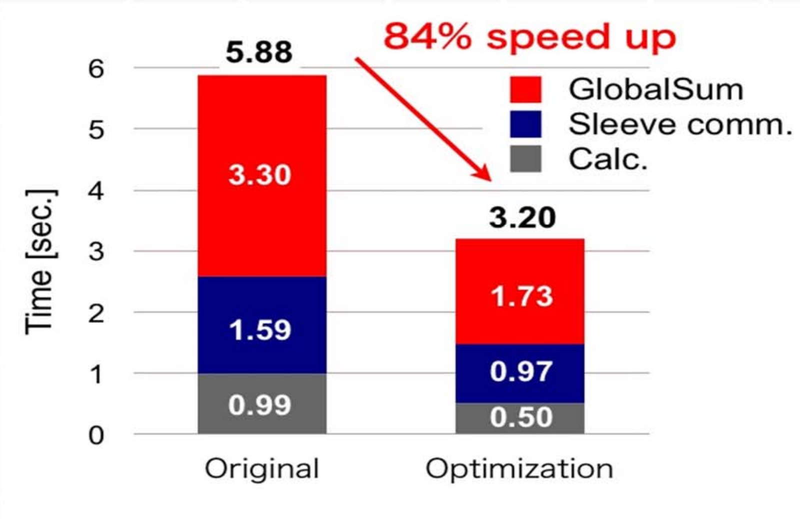
XcalableMP as evolutional approach

- **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)

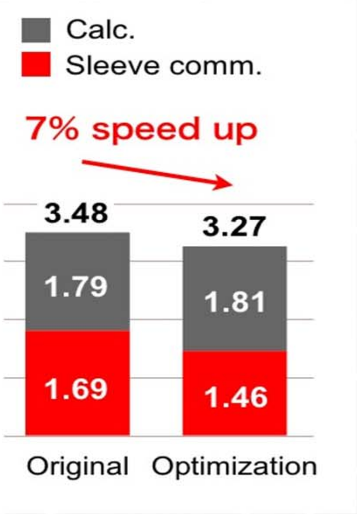
XcalableMP and Application Studies using PGAS model

- **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)

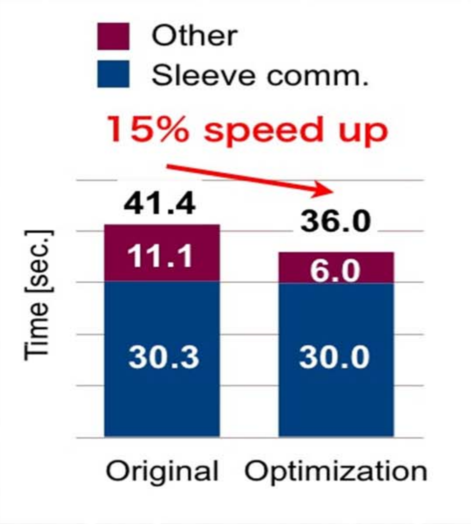
CGPOP : 7500 nodes



NICAM : 640 nodes



CICE : 1024 nodes



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
  - 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
```

“MPI+X” for exascale?

- **X is OpenMP!**
- **“MPI+Open” is now a standard programming for high-end systems.**
  - I’d like to celebrate that OpenMP became “standard” in HPC programming
- **Questions:**
  - “MPI+OpenMP” is still a main programming model for exa-scale?
  - How does PGAS “collaborate” with MPI+X?
  - Can PGAS replace(beat!?) “MPI” for exascale?

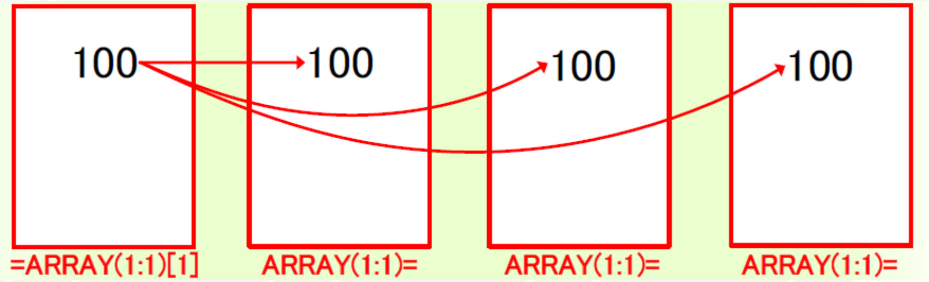
Should everything be written in PGAS?

- MPI broadcast operation can be written in CAF easily, …
- But, is it efficient?

```
REAL(8),DIMENSION(:),ALLOCATABLE :: ARRAY
・ ・ ・
CALL MPI_COMM_SIZE(MPI_COMM_WORLD,NIMG,IERR)
CALL MPI_COMM_RANK(MPI_COMM_WORLD,ID,IERR)
・ ・ ・
CALL MPI_BCAST(ARRAY, MSGSIZE, MPI_REAL8, 0,
MPI_COMM_WORLD,IERR)
```

- Should use "co_broadcast" in CAF.
- Sophisticated collective communication libraries of "matured" MPI are required
- Obviously, PGAS need to **collaborate** with MPI.

```
REAL(8),DIMENSION(:),CODIMENSION[:],ALLOCATABLE :: ARRAY
・ ・ ・
NIMG= NUM_IMAGES()
ID= THIS_IMAGE()
・ ・ ・
SYNC ALL
IF(ID /= 1) THEN
ARRAY(1:MSGSIZE) = ARRAY(1:MSGSIZE)[1]
END IF
SYNC ALL
```



PGAS and remote memory access (RMA)/one-sided comm.

- PGAS is a programming model relating to distributed memory system with a shared address space that distinguishes between local (cheap) and remote (expensive) memory access.
  - Easy and intuitive to describe remote data access, for not only one side-comm, but also stride comm.
- RMA is a mechanism (operation) to access data in remote memory by giving address in (shared) address space.
  - RDMA is a mechanism to directly access data in remote memory without involving the CPU or OS at the destination node.
  - Recent networks such as Cray and Fujitsu Tofu support remote DMA operation which strongly support efficient one-sided communication.
- PGAS is implemented by RMA providing light-weight one-sided communication and low overhead synchronization semantics.
- For programmers, both PGAS and RMA are programming interfaces and offer several constructs such as remote read/write and synchronizations.
  - MPI3 provides several RMA (one-sided comm.) APIs as library interface.

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.

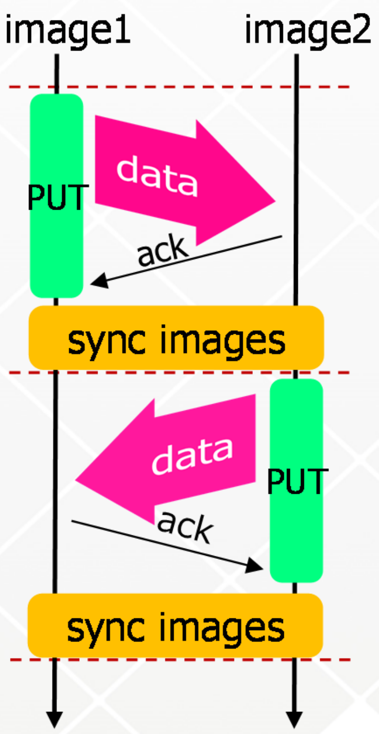
Case study: “Ping-pong” in CAF and RDMA

- Ping-pong: single data transfer with synchronization operation
- EPCC Fortran Coarray micro-benchmark

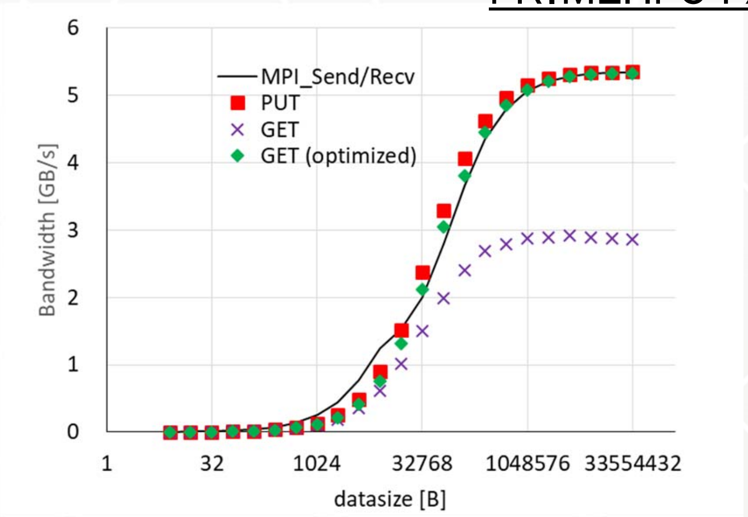
	image1	image2
Ping phase	x(1:ndata)[image2] = x(1:ndata) sync images	sync images
Pong phase	sync images	x(1:ndata)[image1] = x(1:ndata) sync images

- Bandwidth of PUT-based achieves almost the same performance of send/recv. But, latency of small messages is bad.
- PUT exec time = data transfer + wait ack + sync_image
  - Sync_images exchange sync between nodes. It can be optimized by one-direction sync: post/wait (XMP), notify/query (Rice U, CAF2.0), event post/event wait (Fortran 2015)

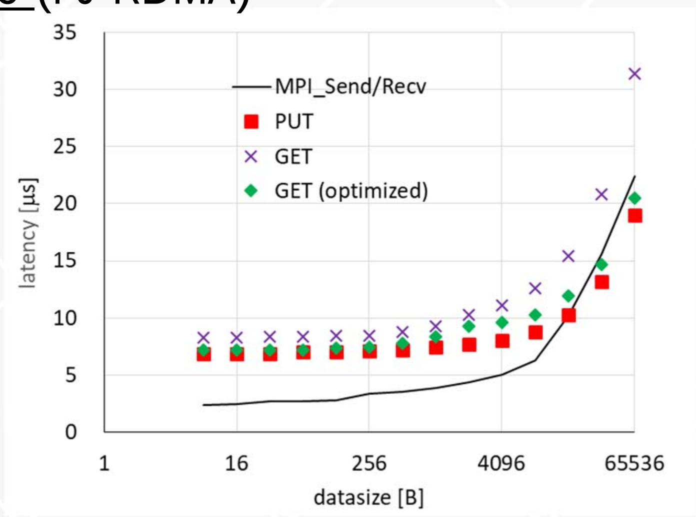
PUT-based



PRIMEHPC FX100 (FJ-RDMA)



bandwidth

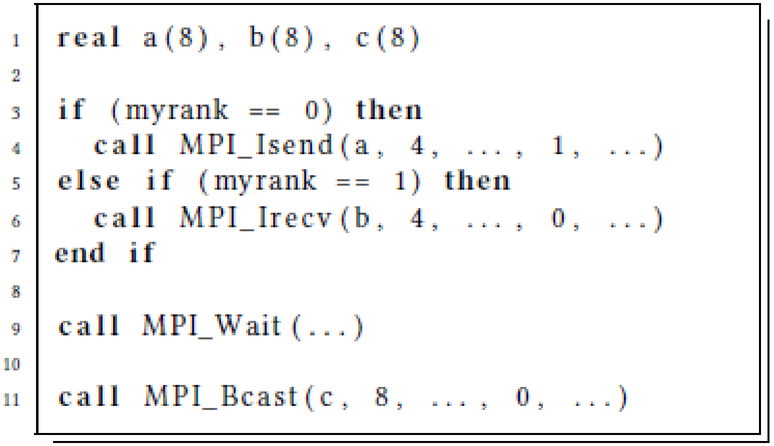


latency

Fiber min-apps in CAF/XMP

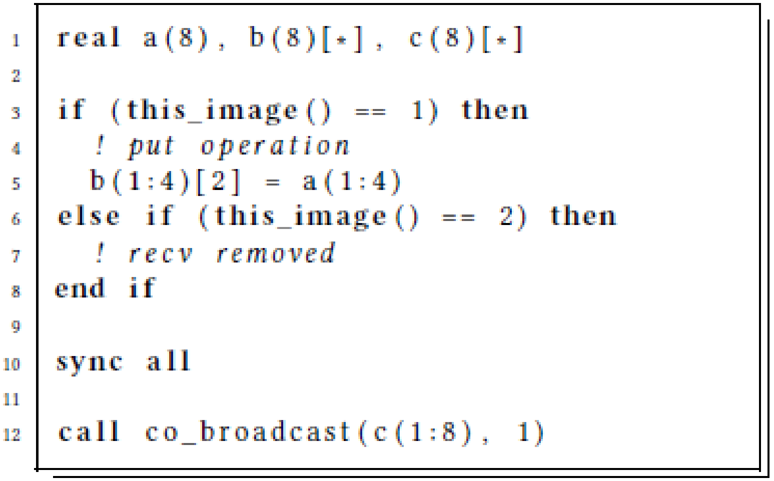
- Hitoshi Murai, Masahiro Nakao, Hidetoshi Iwashita and Mitsuhisa Sato, "Preliminary Performance Evaluation of Coarray-based Implementation of Fiber Miniapp Suite using XcalableMP PGAS Language“, PAW2017 (in morning)
- Replace of MPI with Coarray operation by simple rewriting rule.
- Our coarray-based implementations of three (NICAM-DC, NTChem-MINI, and FFB-MINI) of the five miniapps were comparable to their original MPI implementations.
- However, for the remaining two miniapps (CCS QCD and MODYLAS-MINI) due to communication buffer management

Simple re-writing rule



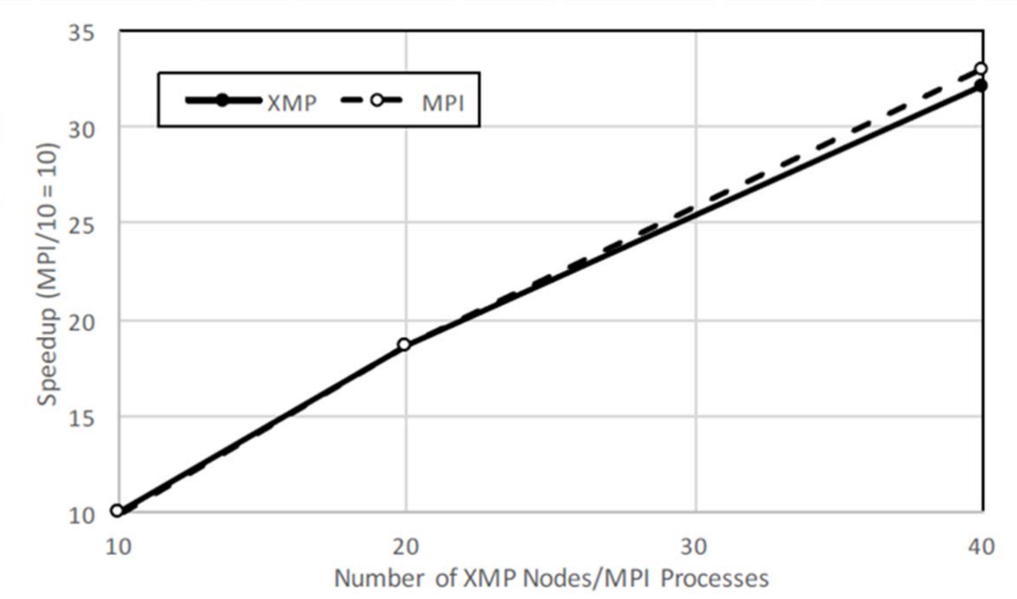
```
real a(8), b(8), c(8)

if (myrank == 0) then
  call MPI_Isend(a, 4, ..., 1, ...)
else if (myrank == 1) then
  call MPI_Irecv(b, 4, ..., 0, ...)
end if

call MPI_Wait(...)

call MPI_Bcast(c, 8, ..., 0, ...)
```

(a) MPI Program



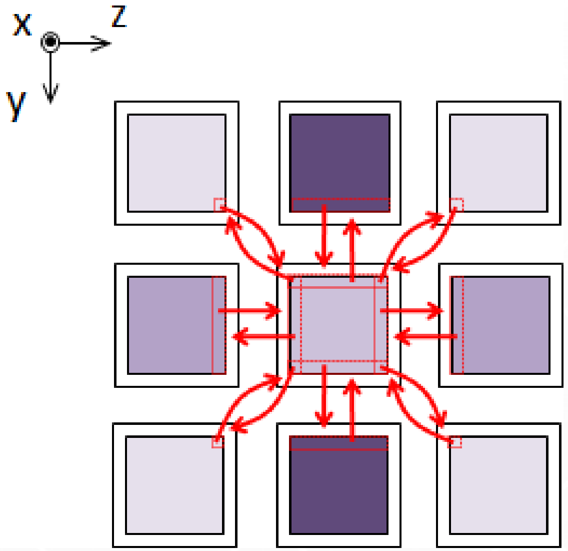
```
real a(8), b(8)[*], c(8)[*]

if (this_image() == 1) then
  ! put operation
  b(1:4)[2] = a(1:4)
else if (this_image() == 2) then
  ! recv removed
end if

sync all

call co_broadcast(c(1:8), 1)
```

Result of NICAM-DC

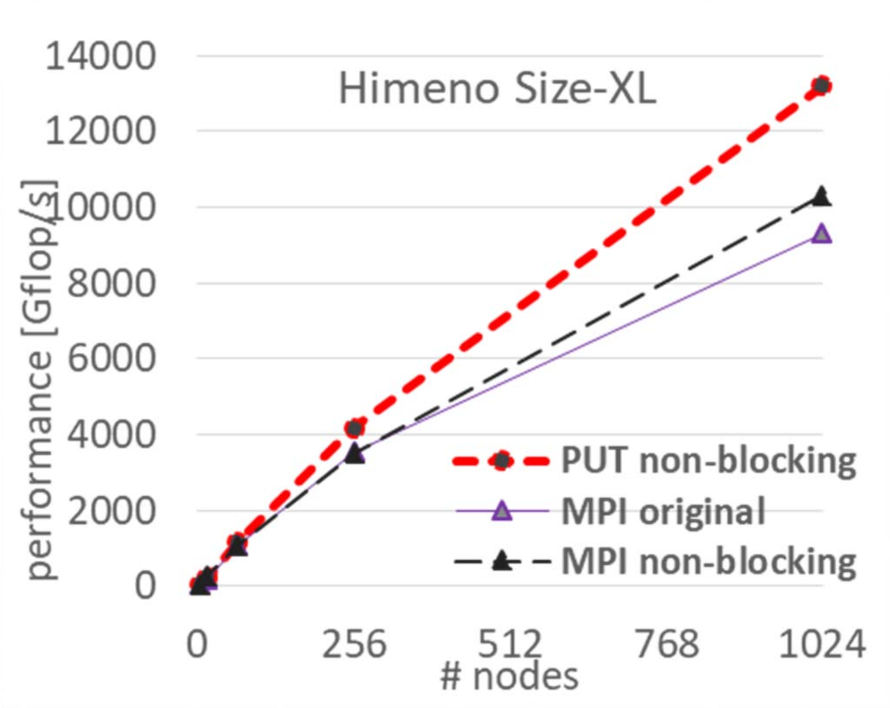


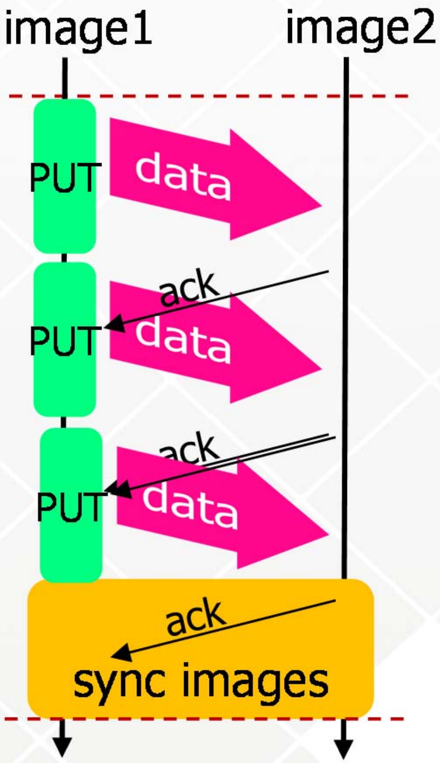
Case study(2): stencil communication

- 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 to be 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”





Support communication patterns in PGAS

- When the communication pattern is expressed in some defined forms, PGAS comm. can be optimized.
  - Coarray assignment by array section syntax. Runtime select better stride comm, pack/unack or direct RDMA.

```
A(1, 1:10)[1] = B(1:10)    // put from image2 to image1
```

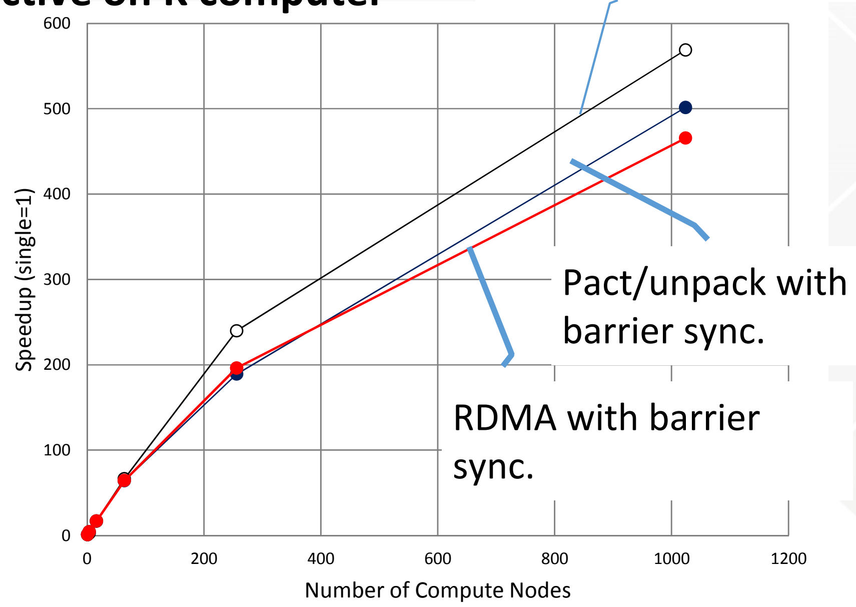
  - Reflect operation in XMP global view model (stencil neighbor comm)

- Irregular Communication Patterns:
  - If communication *pattern* is not known *a priori*, but the data locations are known, send-recv program needs an extra step to make pairs of send-recv. BUT, RMA can handle it easily because only the source or destination process needs to perform the put or get call

```
!$xmp shadow a(2:2, 1:1)
!$xmp reflect (a) width (/periodic/1:1, 0:0) async (0)
!$xmp wait_async (0)
```

- **shadow: declares width of shadow**
- **reflect: executes stencil comm.**
  - width: specifies shadow width to be updated. /periodic/: updates shadow periodically. async: asynchronous comm.
- **wait_async: completes async. reflects**

Optimized stencil communication by reflect directive on K computer



Optimized by RDMA with pt-to-pt sync.

Pact/unpack with barrier sync.

RDMA with barrier sync.

Target: a prototype dynamical core of a climate model SCALE-LES

MPI RMA as a underlying comm. layer for PGAS?

- **“MPI is too low and too high API for communication”. (Prof. Marc Snir, JLESC 7th WS)**
  - MPI RMA APIs offer their PGAS model rather than “primitives” for other PGAS.

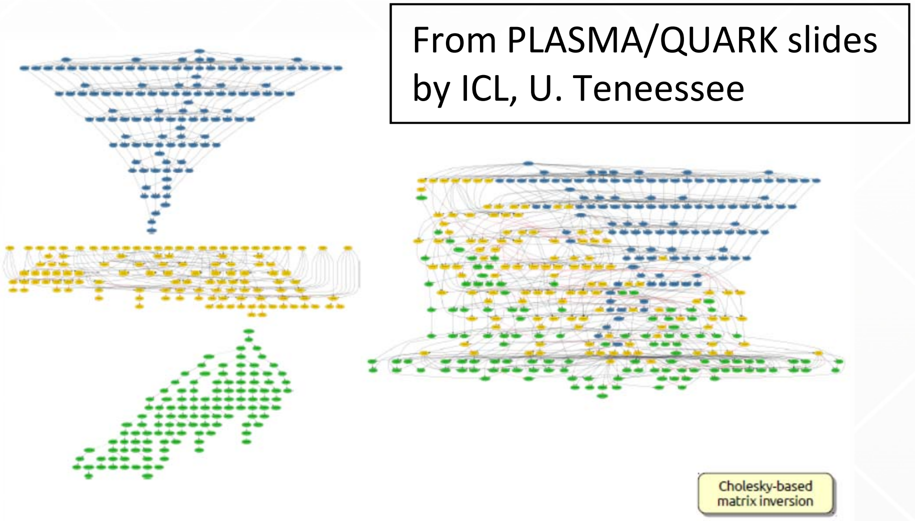
- **In case of our XMP Coarray implementation:**
  - Using “passive target”
  - MPI flush operation and synchronization do not match to implement “sync_images”.
  - Complex “window” management to expose the memory as a coarray.
  - (We need more study for better usage of MPI RMA)
  - Fujitsu RDMA interface is much faster in K-computer.

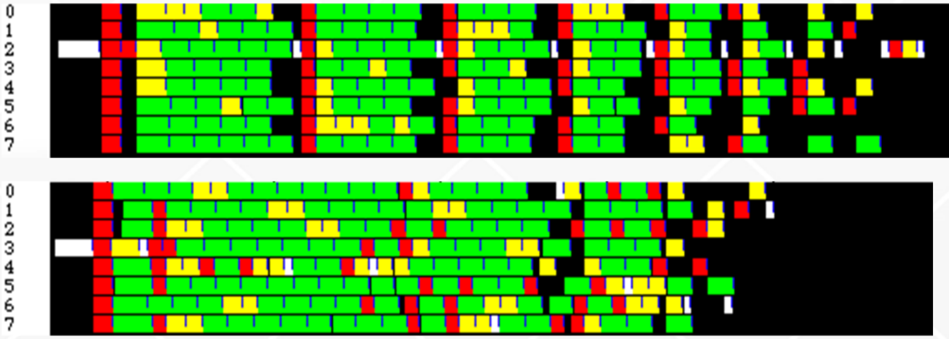
“PGAS + X” ? : PGAS with multitasking

- **X is OpenMP or multi-tasking.**
  - Question: How does PGAS can combine task-model?

- **Multitasking/Multithreaded execution: many “tasks” are generated/executed and communicates with each others by data dependency.**
  - OpenMP task directive, OmpSS, PLASMA/QUARK, StarPU, ..
  - Thread-to-thread synchronization /communications rather than barrier
- **Advantages**
  - 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**
  - The detail spec of the directive is under discussion in spec-WG

From PLASMA/QUARK slides by ICL, U. Teneessee



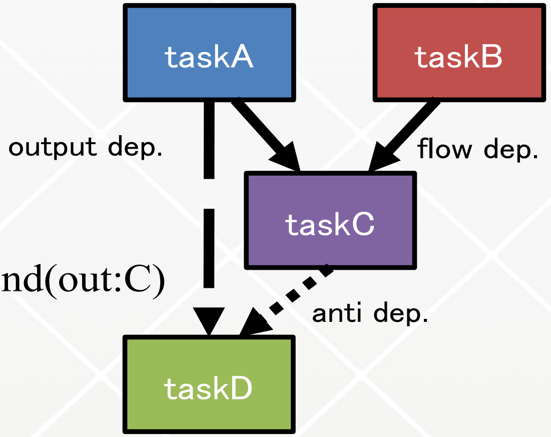


Execution results of Cholesky Factorization in OpenMP 4.0 task

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.

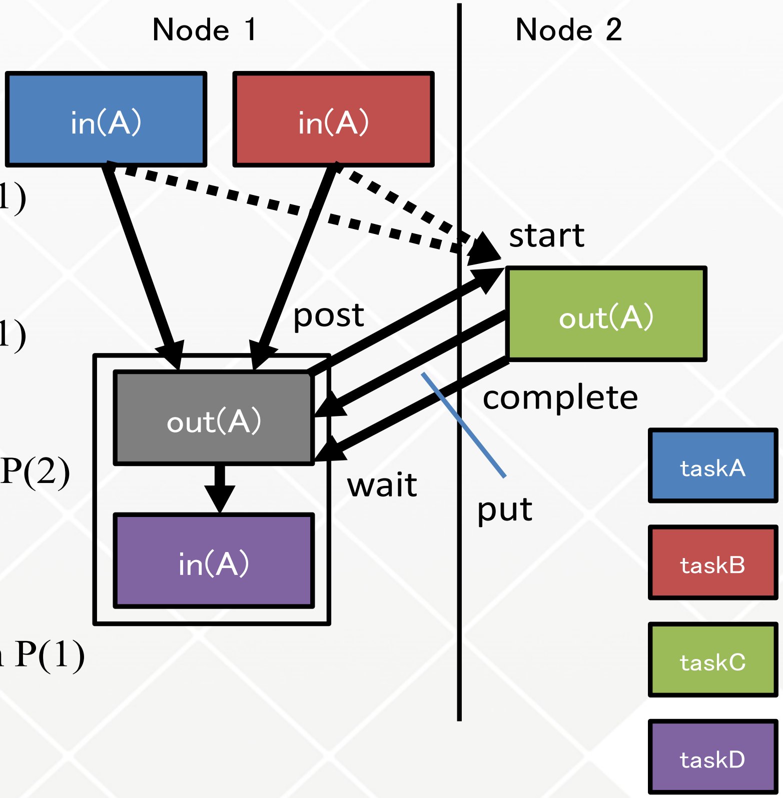
```
#pragma xmp tasklet tasklet-clause[, tasklet-clause[, ...]] on {node-ref | template-ref}
 (structured-block)

 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 */
```

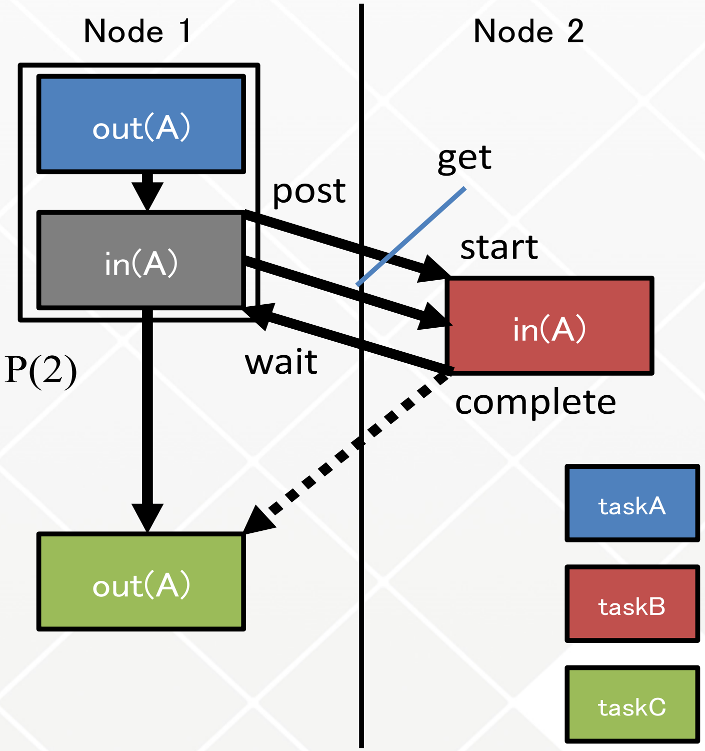


Get operation in tasklet

- 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 */
```



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
```

Some comments

- **Our proposed directive can be applied to blocks containing PGAS communications:**
  - Gmove directive in XMP
  - Coarray operations (local-view of XMP)
  - Other one-sided communication such as MPI_put/MP_get, OpenShmem, GlobalArray …

- **This idea is similar to active target sync “PSCW” model in MPI3.**

Preliminary 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

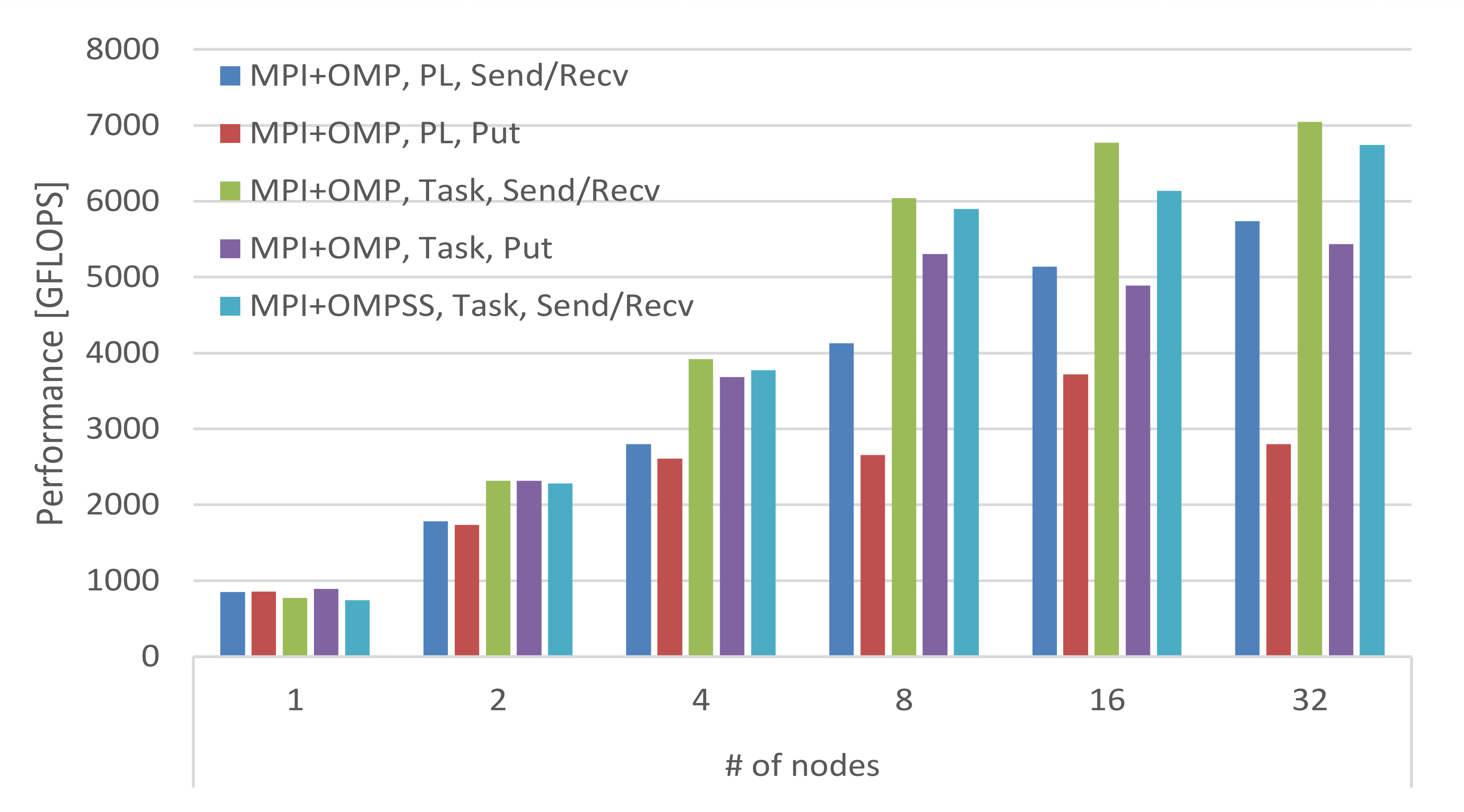
CPU	Intel Xeon Phi 7250 1.4 GHz 68 cores
Memory	16 GB (MCDRAM) + 96 GB (DDR 4)
Interconnect	Intel Omni-Path Architecture
Compiler	Intel Compiler 17.0.1
	Intel MPI Library 2017 Update 1
	Intel MKL 2017 Update 1
	OmpSs 16.06.3

- **Benchmark Program:**

  Blocked Cholesky Factorization

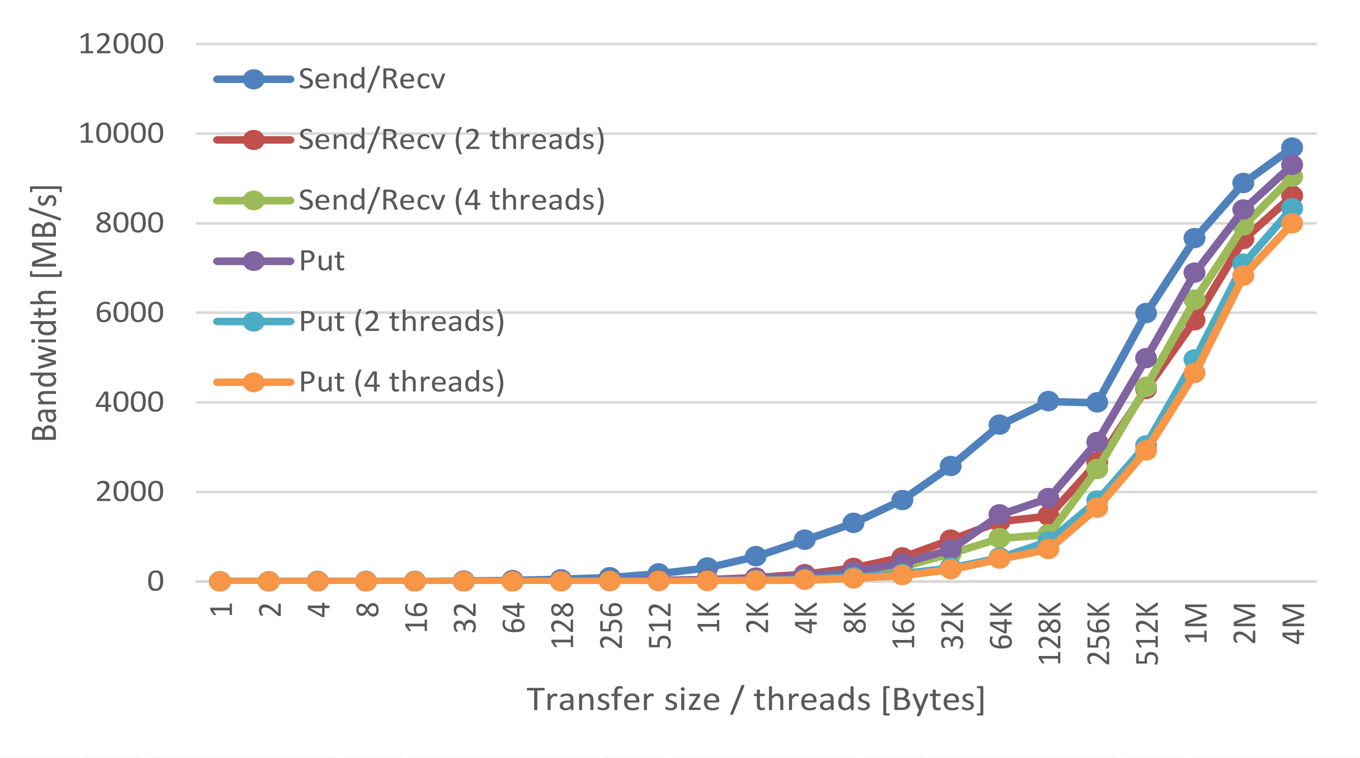
Results

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



Communication performance on KNL system

- **Aggregate comm. performance between multiple threads between nodes**
  - The performance using “Send/Recv” may be better than that using “Put”.
  - <u>the performance of MPI_THREAD_MULTIPLE is lower than that of the single-threaded communication.</u>

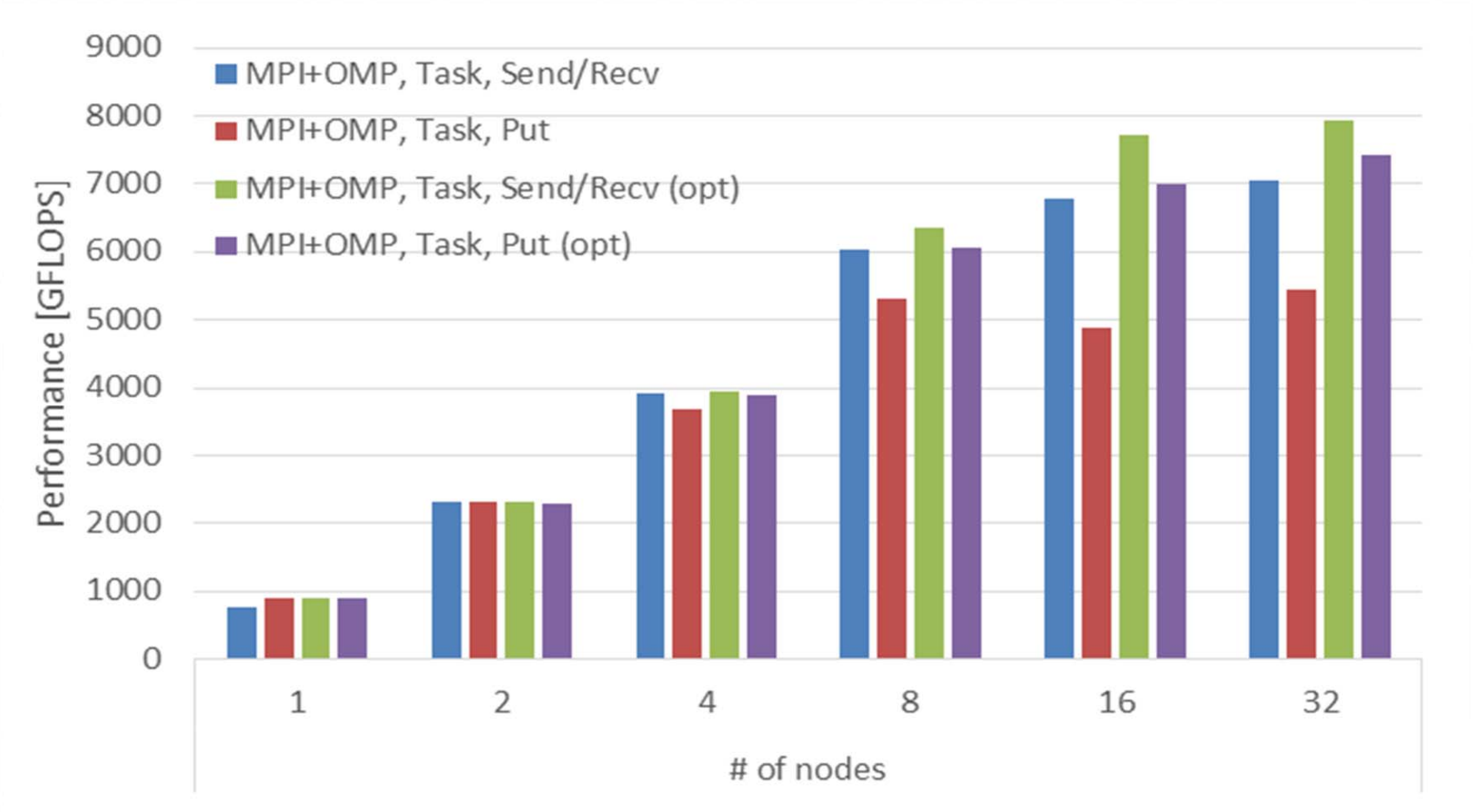


- **Why MPI_THREAD_MULTIPLE is so slow?**
  - “giant-lock” for message ordering and tag matching for wild-card.
  - New proposal is being discussed: “end-point” for thread.
- <u>**Can RDMA be another “light-weight” solution for communication in multithreaded execution of manycore?**</u>

Performance Improvements

- **Optimization**
  - The communication performance may be improved if all communications are to be delegated to the communication thread.
  - If communication pattern is simple enough to be analyzed by the compiler, some patterns of Put can be transformed into Send/Recv.
- (Opt): all communications are delegated to the communication thread
  - Still, Send/recv

  is better



“Compiler-free” approaches for PGAS

- Library approach: MPI3 RMA, OpenShmem, GlobalArray, …
- C++ Template approach: UPC++, DASH, …
- This approach may increase portability, clean separation from base compiler optimization, … but sometimes hard to debug in C++ template…
- But, approach by compiler will give:
  - New language, or language extension provides easy-to-use and intuitive feature resulting in better productivity.
  - Enable compiler analysis for further optimization: removal of redundant sync and selection of efficient communication, etc, …
  - But, in reality, compiler-approach is not easy to be accepted for deployment, and support many sites, …

Summary and Concluding remarks

- **Should everything be written in PGAS?**
  - No, PGAS needs sophisticated collective communication libraries of "matured" MPI.
- **Can PGAS replace MPI? / Can PGAS be faster than MPI?**
  - There are several communication patterns to take advantage of PGAS: multiple data transfers with a single synchronization operation, as well as irregular communication patterns.
- **Is RMA good as a underlying comm. layer for PGAS?**
  - Too high? Some mismatches to implement Co-array in our case.
- **"PGAS + X" ? : PGAS with multitasking**
  - Multitasking model in XMP 2.0 is proposed.
  - RDMA can take advantage in communication model for multithreading in manycore?
- **"Compiler-free" approaches for PGAS**
  - I like compiler-approach! ☺

- **We need to explore more applications to take advantage of PGAS models under <u>the fact that most apps are already written in MPI.</u>**