MUST system applied to high level language approach in MYX project

Taisuke Boku

Deputy Director and HPC System Research Division Leader

Center for Computational Sciences, University of Tsukuba

(collaborated with Hitoshi Murai and Miwako Tsuji, R-CCS RIKEN)

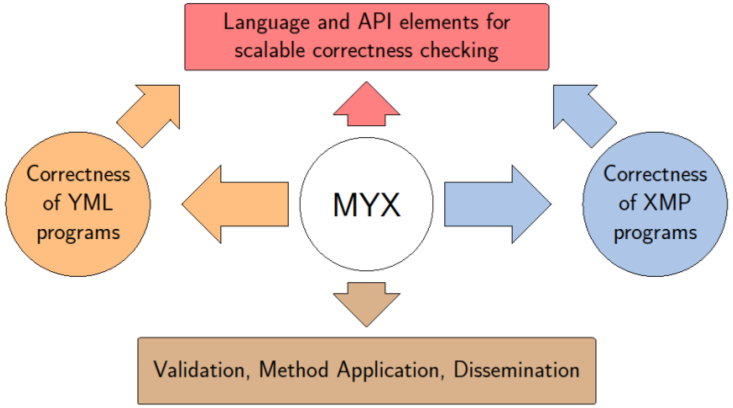
MYX Project Consortium

- MUST Correctness Checking for YML and XMP Programs.
- International collaboration among Germany (DFG), Japan (JST), and France (ANR).
- Part of the Priority Programme "Software for Exascale Computing" (SPPEXA) in German.

- Partner from Germany (project coordinator)
  - RWTH Aachen, IT Center and Institute for High Performance Computing
  - Prof. Matthias S. Mueller, Joachim Protze, Christian Terboven
- Partner from Japan
  - University of Tsukuba, Center for Computational Sciences, and Advanced Institute of Computational Science, RIKEN
  - Prof. Taisuke Boku, Hitoshi Murai, Miwako Tsuji
- Partner from France
  - Maison de la Simulation
  - Prof. Serge Petiton. Prof. Nahid Emad

MYX Project

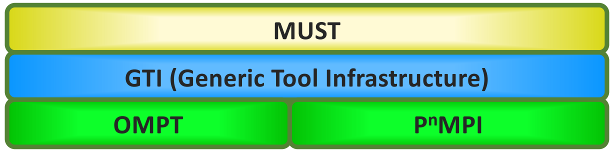
- Background
  - Errors in programs will increase in highly-parallel and complicated exascale computing.
  - Automatic correctness checking of programs is important.
- Goals
  - higher productivity by scalable correctness checking
  - targets: YML and/or XcalableMP (XMP)
- Components
  - MUST: a correctness checking tool
  - YML: a workflow language
  - XMP: a PGAS language

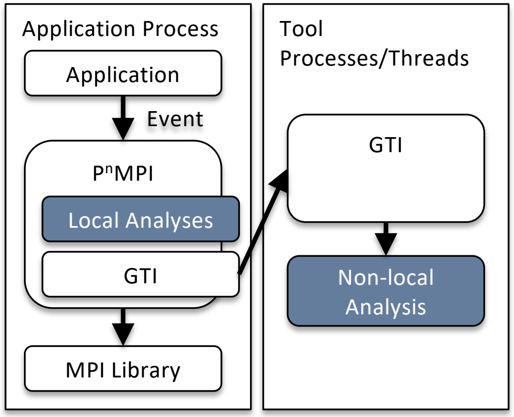

Language and API elements for scalable correctness checking
Correctness of YML programs
MYX
Correctness of XMP programs
Validation, Method Application, Dissemination

MUST

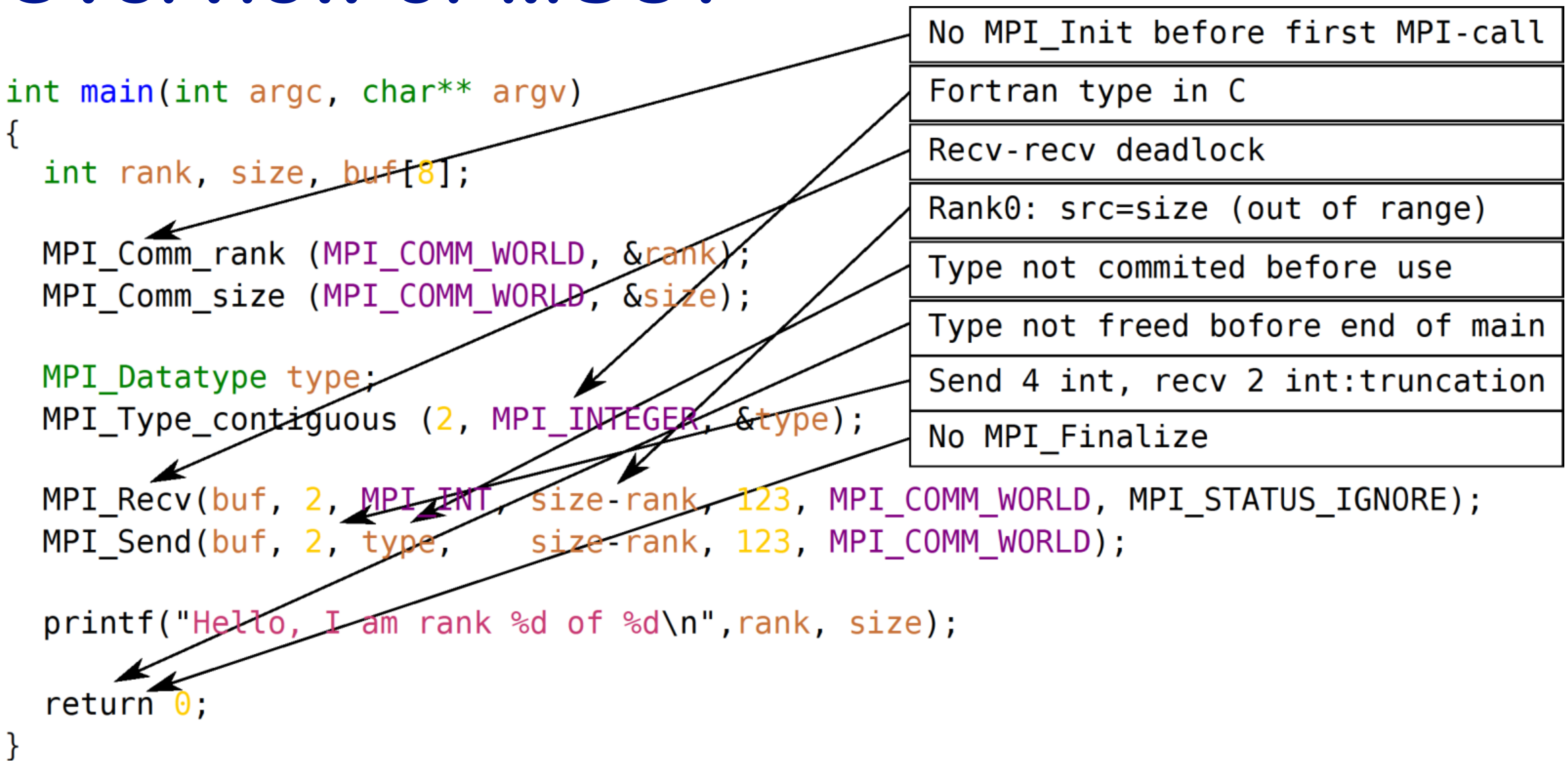
- Correctness checking tool developed by RWTH Aachen
  - can detect local and global errors in MPI/OpenMP programs.
- The latest version supports checking MPI one-sided comms.

MUST software stack





Overview of MUST



```
int main(int argc, char** argv)
{
  int rank, size, buf[8];

  MPI_Comm_rank (MPI_COMM_WORLD, &rank);
  MPI_Comm_size (MPI_COMM_WORLD, &size);

  MPI_Datatype type;
  MPI_Type_contiguous (2, MPI_INTEGER, &type);

  MPI_Recv(buf, 2, MPI_INT, size-rank, 123, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  MPI_Send(buf, 2, type,    size-rank, 123, MPI_COMM_WORLD);

  printf("Hello, I am rank %d of %d\n",rank, size);

  return 0;
}
```

No MPI_Init before first MPI-call

Fortran type in C

Recv-recv deadlock

Rank0: src=size (out of range)

Type not commited before use

Type not freed bofore end of main

Send 4 int, recv 2 int:truncation

No MPI_Finalize

What's XcalableMP ?

www.xcalablemp.org

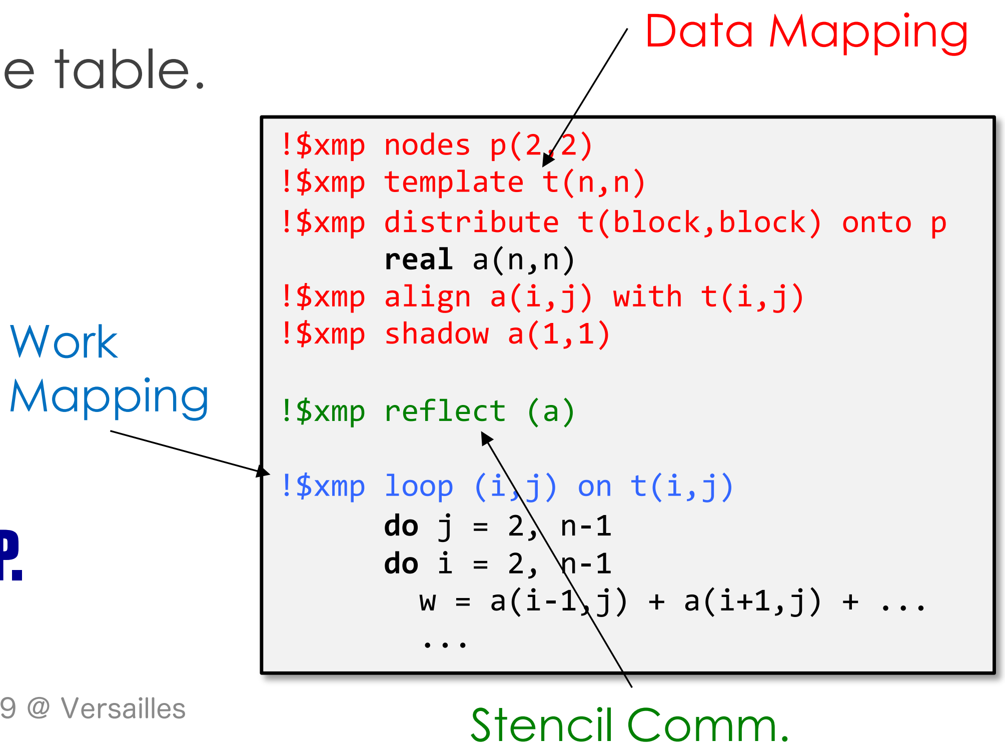
Directive-based PGAS extension for Fortran & C

- Proposed by *XMP Spec. WG* of PC Cluster Consortium.
- Ver. 1.4 spec. is available.
- Now ver. 2.0 (incl. C++ support) on the table.
- Adopted by Post-K Projects.

Supports two parallelization models:

- Global-view (based on HPF-like data/work mapping directives)
- Local-view (based on coarray)

Allows mixture with MPI and/or OpenMP.



Data Mapping

Work Mapping

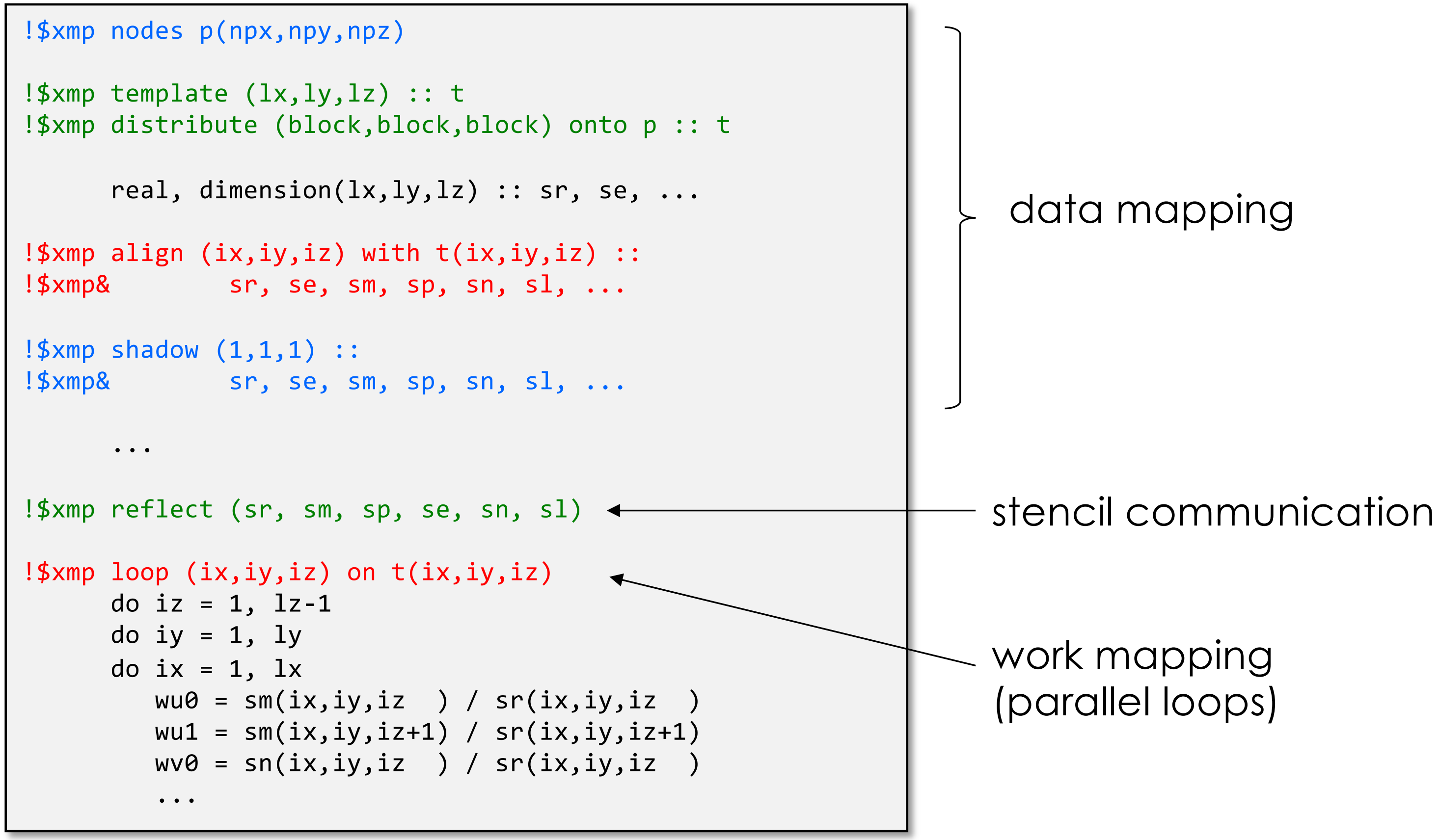
Stencil Comm.

```
!$xmp nodes p(2,2)
!$xmp template t(n,n)
!$xmp distribute t(block,block) onto p
      real a(n,n)
!$xmp align a(i,j) with t(i,j)
!$xmp shadow a(1,1)

!$xmp reflect (a)

!$xmp loop (i,j) on t(i,j)
      do j = 2, n-1
      do i = 2, n-1
        w = a(i-1,j) + a(i+1,j) + ...
        ...
```

Example of a Global-view XMP Program

```
real, dimension(lx,ly,lz) :: sr, se, ...







...




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  )
   ...
```

Example of a Global-view XMP Program



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

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

      real, dimension(lx,ly,lz) :: sr, se, ...

!$xmp align (ix,iy,iz) with t(ix,iy,iz) ::
!$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  )
         ...
```

data mapping

stencil communication

work mapping
(parallel loops)

Local-view Programming in XMP

- *Coarray*, a PGAS feature of Fortran 2008, is available in XMP/C as well as in XMP/Fortran.
- Basic idea: data declared as *coarray* can be accessed by remote nodes.

XMP/Fortran

```
1 real a(1024)[*], b(1024)
2 a(512:1024)[1] = b(1:512)
3 sync all
```

XMP/C

```
1 float a[1024]:[*], b[1024];
2 a[512:512]:[0] = b[0:512];
3 xmp_sync_all(NULL);
```

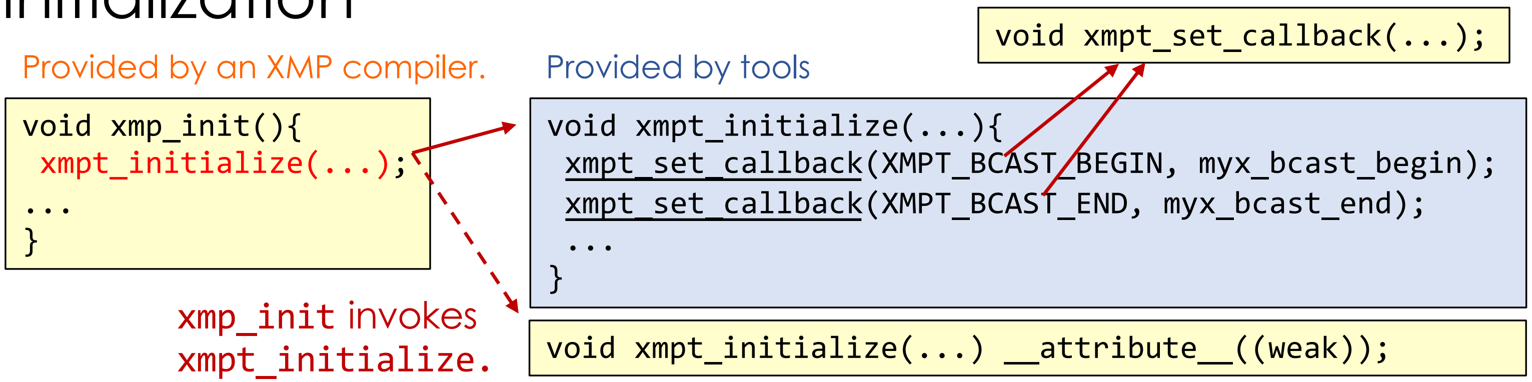
1. An array `a` is declared as a coarray.
2. A local array section `b(1:512)` is put to a remote array section `a(512:1024)` on image 1.
3. A memory fence and barrier synchronization is performed.

XMPT Tool Interface

- ... is a tool API of XMP.
- Objective:
  - providing a more generic tool API of XMP.
- Basic ideas inspired by OMPT
  - event- and callback-based
- Planned targets:
  - *Score-P* / *Scalasca* (JSC)
  - *Extrae* (BSC)
  - *MUST* correctness checking tool (this project)
  - etc.

Basic Design of XMPT

- At initialization



Callbacks are registered through `xmpt_set_callback.`

```
void xmpt_set_callback(...);
```

Provided by an XMP compiler.

```
void xmp_init(){
 xmpt_initialize(...);
...
}
```

xmp_init invokes xmpt_initialize.

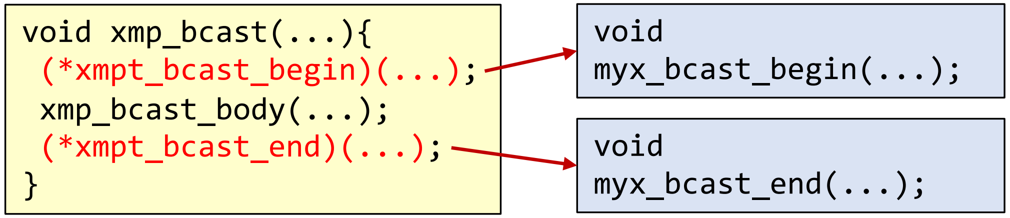
Provided by tools

```
void xmpt_initialize(...){
 xmpt_set_callback(XMPT_BCAST_BEGIN, myx_bcast_begin);
 xmpt_set_callback(XMPT_BCAST_END, myx_bcast_end);
 ...
}
```

```
void xmpt_initialize(...) __attribute__((weak));
```

- At each event

The registered callbacks are invoked.



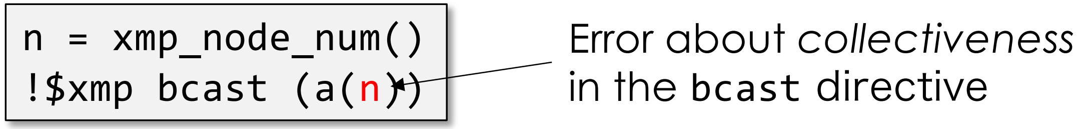
```
void xmp_bcast(...){
 (*xmpt_bcast_begin)(...);
 xmp_bcast_body(...);
 (*xmpt_bcast_end)(...);
}
```

```
void
myx_bcast_begin(...);
```

```
void
myx_bcast_end(...);
```

Correctness Checking of XMP Programs

- Errors in global directives



```
n = xmp_node_num()
!$xmp bcast (a(n))
```

Error about *collectiveness* in the `bcast` directive

- Data race of coarrays
  - XMPT events are defined for coarray accesses & syncs. as well as XMP directives.
  - MYX could detect it.

A data race may occur when a coarray is accessed in unordered segments in different images.

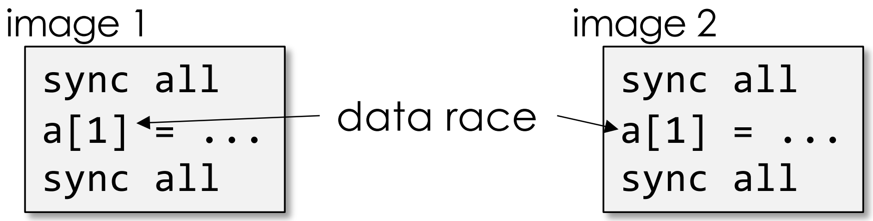


image 1

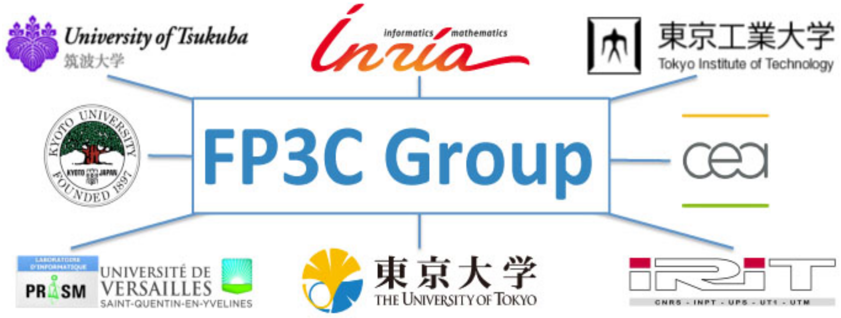
```
sync all
a[1] = ...
sync all
```

image 2

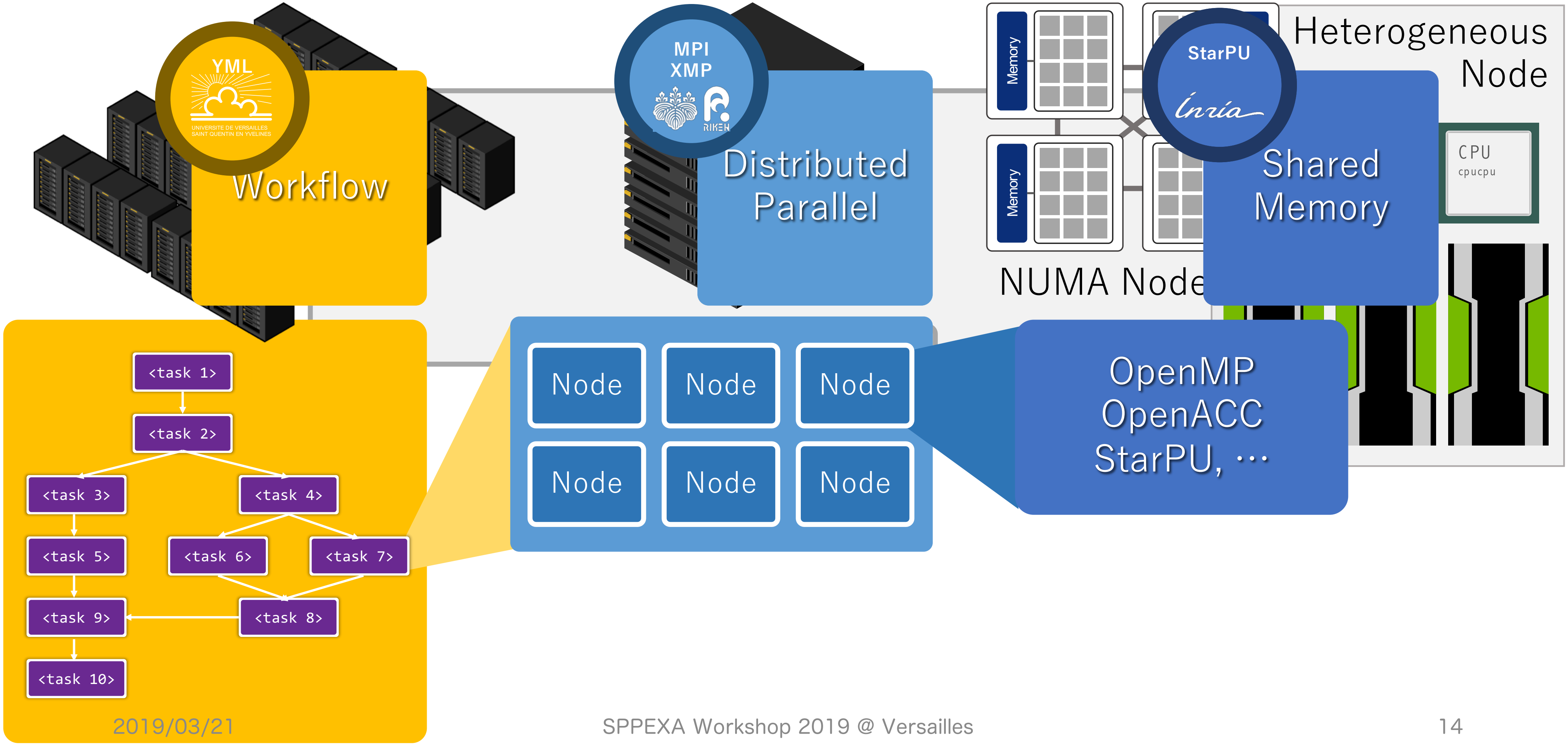
```
sync all
a[1] = ...
sync all
```

XMP+YML and FP3C project

- FP3C: Framework and Programming for Post Petascale Computing
  - a collaborative project between Japan and France
  - September. 2010 – March. 2014
- Various research fields and their integration
  - Programming model and programming language design
  - Runtime libraries
  - Accelerator
  - Algorithm and mathematical libraries
  - etc…

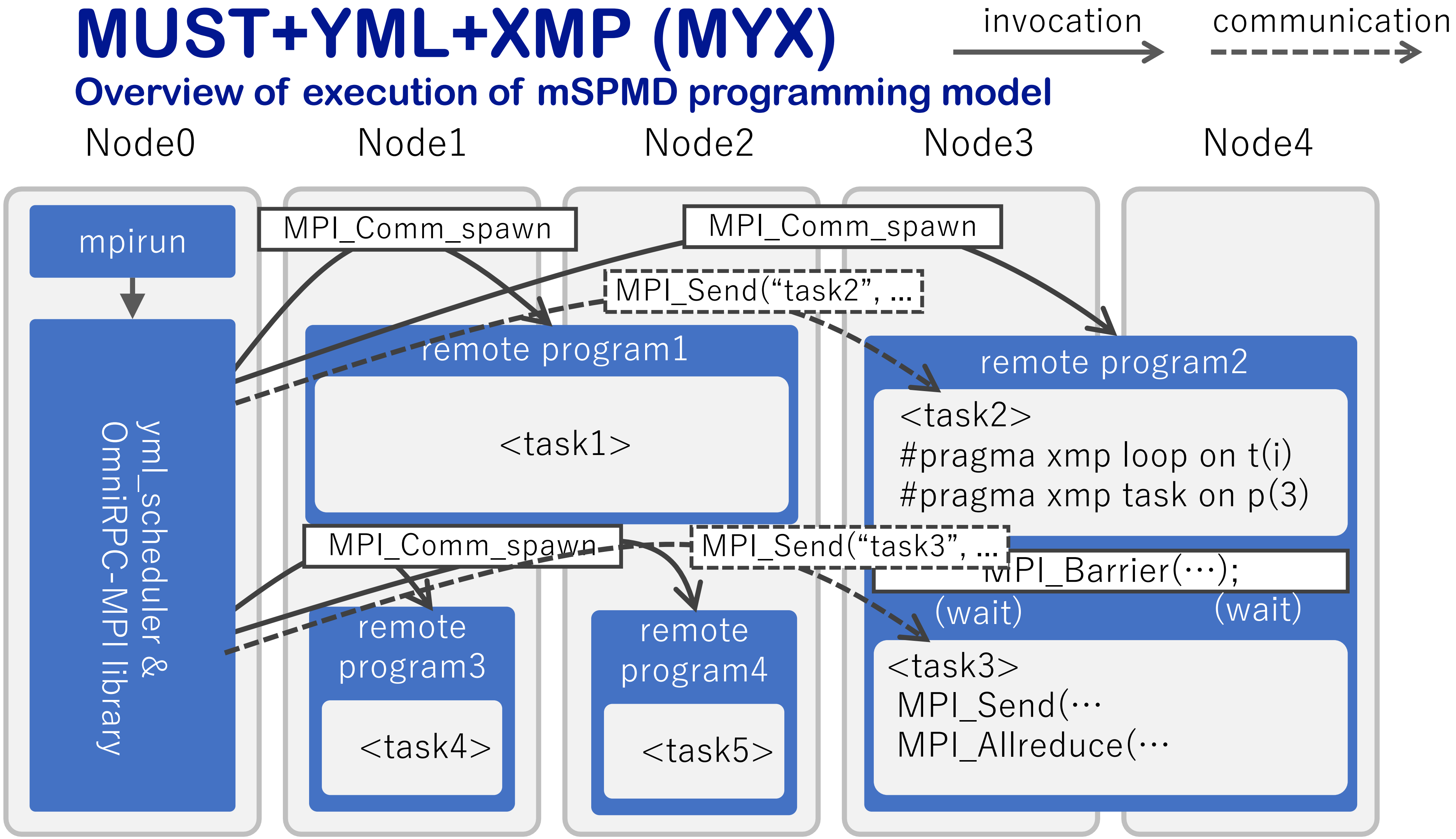


Multi SPMD (mSPMD) Programming Model



MUST+YML+XMP (MYX)

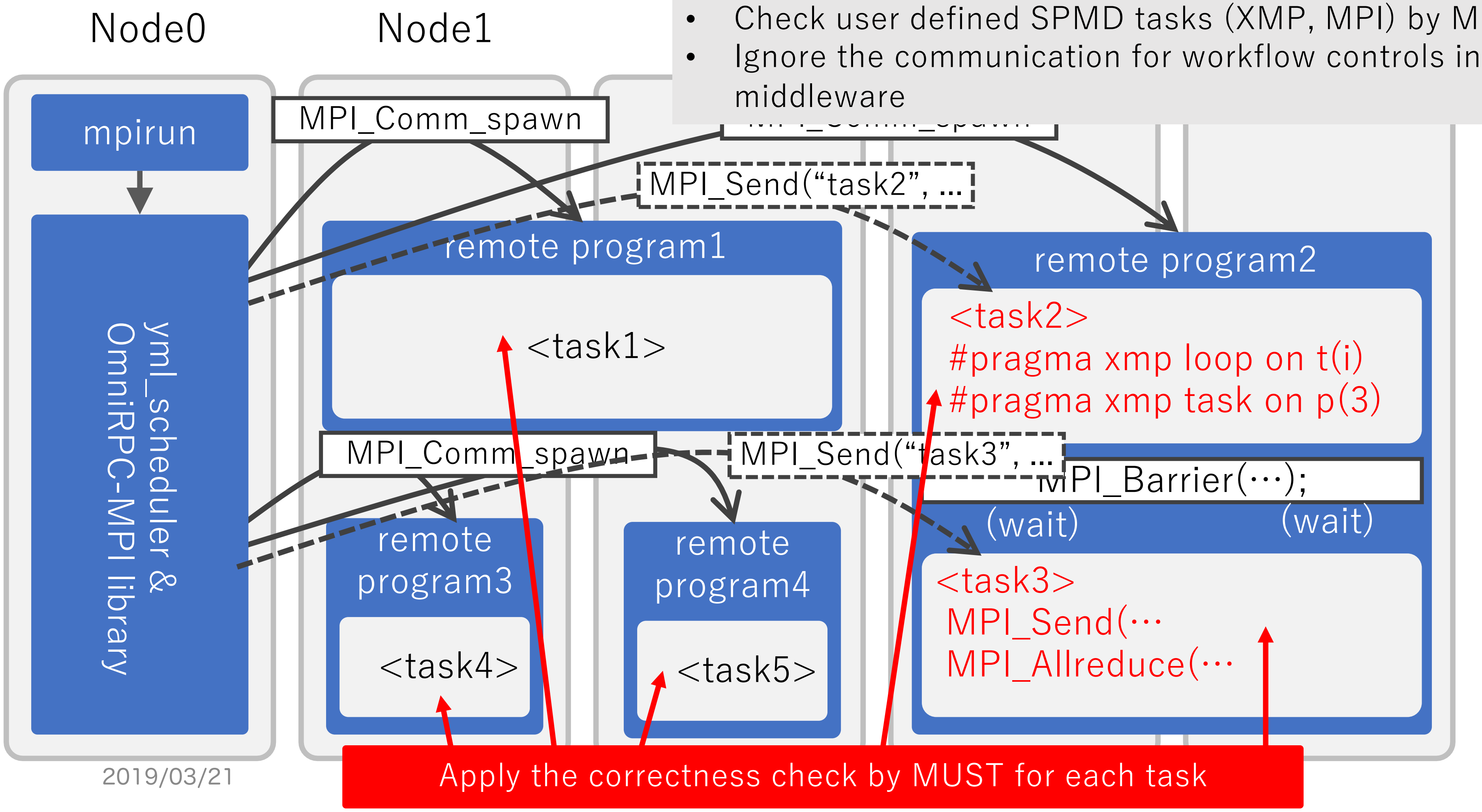
Overview of execution of mSPMD programming model



MUST+YML+XMP (MYX)

Target of correctness check in execution of mSPMD programming model

- Check user defined SPMD tasks (XMP, MPI) by MUST
- Ignore the communication for workflow controls in the middleware



Apply the correctness check by MUST for each task

MUST+YML+XMP (MYX): Implementation

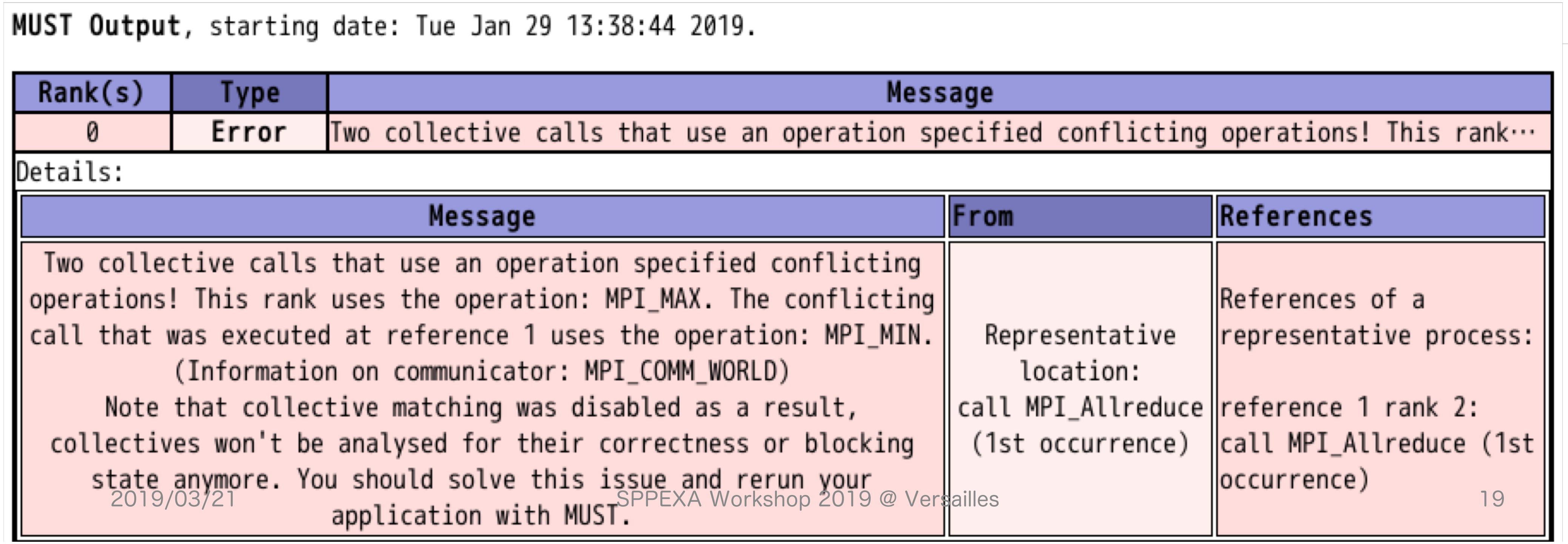
- MUST+MPI / MUST+XMP : to check a single SPMD program
  - **mustrun** –np *n* application.exe
  - prepare a dedicated dynamic library for the application.exe, set the environmental variables
  - mpirun –np (*n*+*1*) application.exe: 1 process should be kept for the MUST analysis
- MUST+YML+MPI/XMP: to check multiple SPMD program
  - Instead of mustrun (mpirun), MPI_Comm_spawn is used to invoke remote SPMD programs in mSPMD
  - extend the middleware of workflow scheduler and the remote program generator in mSPMD
    - MPI_* functions in the workflow control are replaced with PMPI_* functions
    - MPI_Comm_spwan("prog", **n**, …) → PMPI_Comm_spwan("prog", **n+1**, …)
  - preparation steps performed within the mustrun script before mpirun should be performed before starting a workflow
    - set the environmental variables required by MUST manually (Originally, they are set by the mustrun scprit)
    - prepare a dedicated dynamic library to analyze each remote program

Experiments

- Repeat simple communications w/ a nd w/o error s in each task of themSPMD Programming Model
  - investigate the results when MUST is applied, or when MUST is not applied
  - investigate the overhead
- Experimental environment
  - Intel Xeon CUP E5-2680 v3 @ 2.5GHz (24 core)
  - DDR4-2133 Reg ECC (2GBx6)
  - flat-MPI (up to 24 processes)
- Configurations:
  - each task runs on 4 processes, 4 tasks are executed simultaneously
  - each task runs on 10 processes, 2 tasks are executed simultaneously

Result

	mSPMD w/ MUST		mSPMD wo MUST
Reduction・correct	complete		complete
Reduction・incorrect	terminated	reported	terminated
Pingpong・correct	complete		complete
Pingpong・incorrect	complete	reported	complete



MUST Output, starting date: Tue Jan 29 13:38:44 2019.

Rank(s)	Type	Message
0	Error	Two collective calls that use an operation specified conflicting operations! This rank…

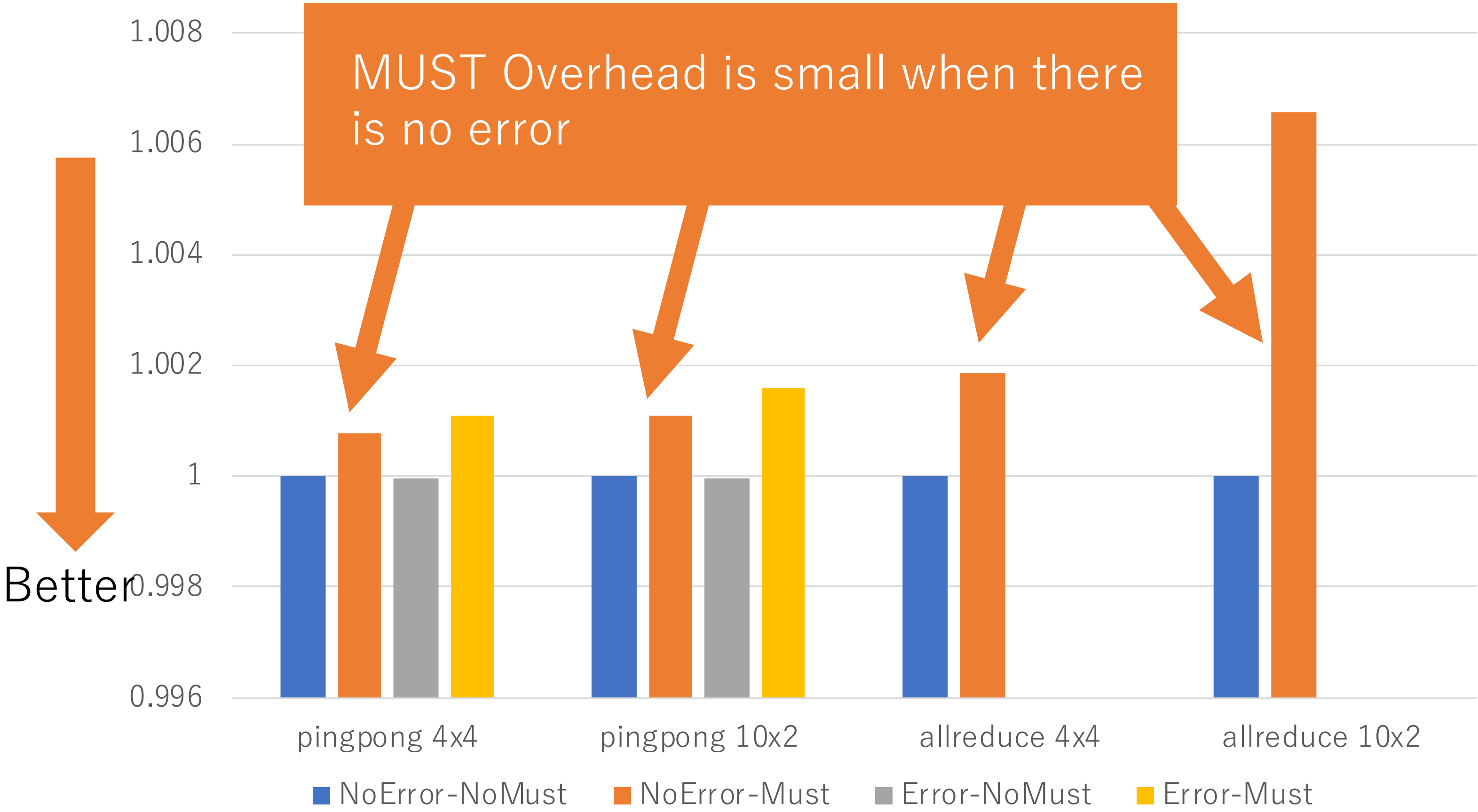
Details:

Message	From	References
Two collective calls that use an operation specified conflicting operations! This rank uses the operation: MPI_MAX. The conflicting call that was executed at reference 1 uses the operation: MPI_MIN. (Information on communicator: MPI_COMM_WORLD) Note that collective matching was disabled as a result, collectives won't be analysed for their correctness or blocking state anymore. You should solve this issue and rerun your application with MUST.	Representative location: call MPI_Allreduce (1st occurrence)	References of a representative process: reference 1 rank 2: call MPI_Allreduce (1st occurrence)

Experiments (overhead)

- MPI-pingpong w/ and w/o an error, w/ and w/o MUST
- MPI-allreduce w/ and w/o an error, w/ and w/o MUST
- Relative execution time

based on the case that is w/o error, w/o MUST


MUST Overhead is small when there is no error

Conclusion

- MYX: an international collaborative project for higher productivity in exascale computing. Runtime correctness check by MUST for multi SPMD Programming Model by YML+XMP
  - MUST is a correctness checking tool.
  - YML is a workflow language (to be presented by Miwako)
  - XMP is a directive-based PGAS extension for Fortran & C supporting the global- and local-view programming.
- XMP+MUST
  - XMP provides an interfere, XMPT, for performance tools
  - MUST uses the XMPT and check the correctness of XMP
- XMP+YML
  - Tasks written in XMP of a workflow managed by YML
- MUST+YML+XMP
  - The task generator and middleware in mSPMD have been extended

  ⇒ Scalable, reliable programming model with high productively

  Scalable : Combination of multiple-SPMDs by YML and XMP

  Reliable : Fault-detection and recovery are supported

  High Productively : XMP, YML are easier than C+MPI

  MUST and XMPT provide a debug tool for SPMD