Overview of MYX
(MUST correctness checking for YML and XMP)

Taisuke Boku

Deputy Director, Center for Computational Sciences

University of Tsukuba

Project partner organizations: RWTH Aachen University, Germany
Maison de la Simulation, France
University of Tsukuba, Japan

Motivation

Exascale Systems consist of

tens of thousands of compute nodes + accelerators

Hierarchy of Compute and Data require

multi-level parallel programs, for instance MPI+X
important: user productivity in parallel programs

Opportunities for new Paradigms examples

Japan's Exascale Language: XMP
Workflow Language YML
} Correctness checking

Aspects of MYX

Correctness Checking of PGAS, distributed and shared memory
Guidance on the development of parallel programming languages

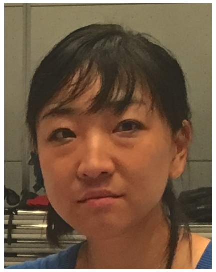
guide

Consortium

- MYX builds on successful preliminary work and collaboration:
  - FP3C: French-Japanese collaboration on YML and XMP for over 10 years
  - JST-CREST: Japanese Exascale research program supporting XMP
  - MUST: scalable correctness checking tool for MPI (and OpenMP)





Partner from Germany (project coordinator)

RWTH Aachen University
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

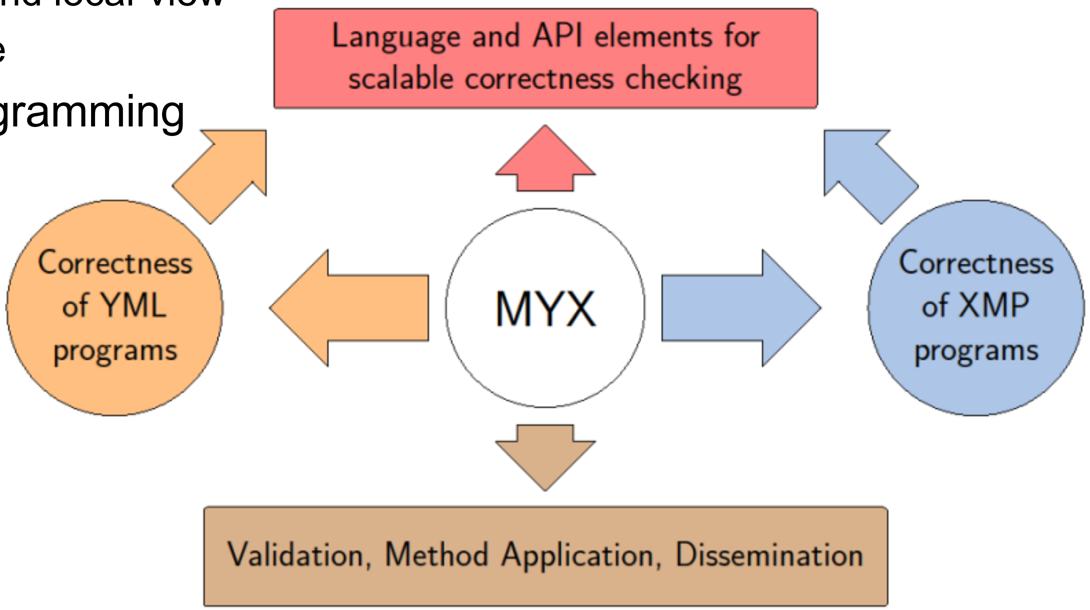
Maison de la Simulation
Prof. Serge Petiton. Prof. Nahid Emad

Research Challenges and Project Results

- The more parallelism expressed, the higher the chance of errors being made
- Time of programming error search and fix: productivity loss!
  - Automatic correctness checking may be used to avoid that

- MYX objectives are
  - enable productivity improvements by means of scalable correctness checking
  - of YML- and XMP-programs
    - XMP: PGAS, with both global-view and local-view
    - YML: graph of components language
  - guide the development of future programming models

- MYX will result in
  - a clear guidance how to limit the risk to introduce errors,
  - how to best express parallelism to catch errors at runtime,
  - extended and scalable correctness checking methods, including PGAS



Initial results for defect classification

- Degree of non-determinism:
  - Strict rules minimize “design issues“ → detection of issues
  - Loose rules provide more freedom in application / algorithm

- More constraints → issues can be detected at compile time/runtime
  - Classification of constraints as static, dynamic or global properties

  - Exercised for XMP:
    - Static constraints can be analyzed at compile time
    - Dynamic constraints can be analyzed at runtime
    - Global constraints can be analyzed at runtime with global knowledge

- Memory model:
  - How is synchronization defined?
  - What is the intended behavior for unsynchronized memory access?

How many errors can you spot in this tiny example?

```
#include <mpi.h>
#include <stdio.h>

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;
}
```

At least 8 issues in this code example

How many errors can you spot in this tiny example?

```c
#include <mpi.h>
#include <stdio.h>

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 committed before use
- Type not freed before end of main
- Send 4 int, recv 2 int: truncation
- No MPI_Finalize before end of main

Overview of defect classification from tiny MPI example

	Static	Dynamic
Local	Fortran Type in C	MPI_Init before first MPI call Rank out of range Type not committed before use Type not freed before finalize No call to MPI_Finalize
Global		Recv-Recv Deadlock Type matching in messages

Examples of defect classification for XMP

	Static	Dynamic
Local	• *async-id* must be of type default integer • *array-name* must be declared before align directive	• The node set specified in the on clause must be a subset of the parent node set. • The source node specified by the from clause must belong to the node set specified by the on clause of bcast.
Global		• Collective consistency • Deadlock • Co-array data race

Abstract data / controlflow for correctness tools

Parallel language domain:

- Synchronization
- Parallel constructs
- Data access

Abstraction layer:
Event notification

Transformation →

Parallel tool domain:

- Synchronization
- Concurrency
- Dependencies
- Memory access

Analysis ↓

Parallel tool domain :

- Report of data races
- Report of dead locks
- Report of performance issue

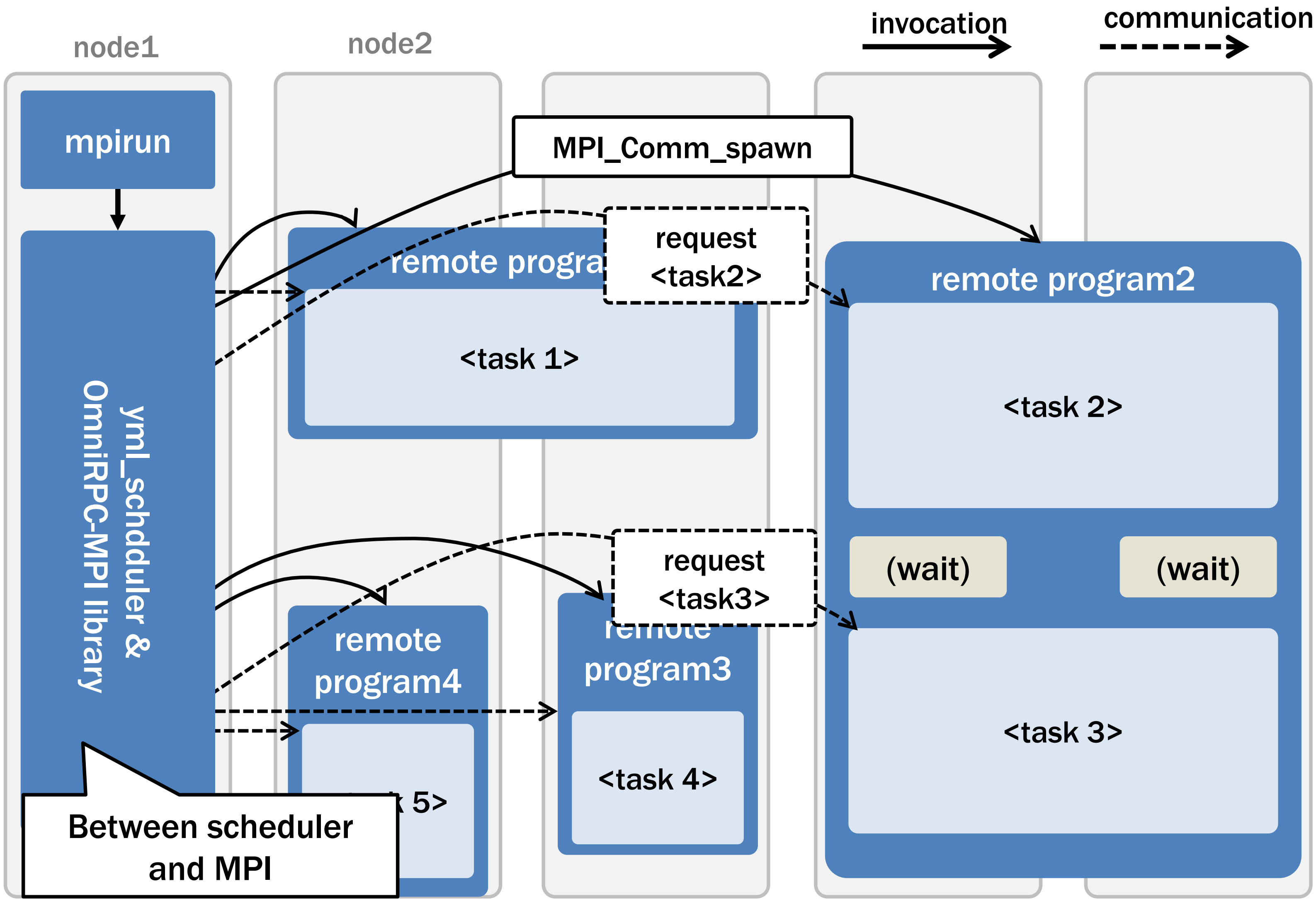
Abstraction layer:
Source information

← Transformation

Parallel language domain:

- Report of data races
- Report of dead locks
- Report of performance issue

Application Execution with XMP/YML



Outline of XcalableMP (XMP) language

- Execution model: SPMD (=MPI)
- Two programming model on data view
  - Global View (PGAS): based on data parallel concept, directives similar to OpenMP is used for data and task distribution (easy programming)
  - Local View (Coarray): based on local data and explicit communication (easy performance tuning)
- OpenMP-like directives
  - Incremental parallelization from original sequential code
  - Low cost for parallelization -> high productivity
- Not "fully automatic parallelization", but user must do:
  - Each node processes the local data on that node
  - User can clearly imagine the data distribution and parallelization for easiness of tuning
  - Communication target of variables (arrays) and partitions can be simply specified
  - Communication point is specified by user, in easy manner
- Running on many platforms: K Computer, PC clusters, Fujitsu MPP
  ➞ planned to run also on Post-K Computer

XMPT Tool I/F

- A tool API of XMP (including XACC)
- Objective:
  - providing a more generic tool API of XMP.
- Basic ideas inspired by OMPT (OpenMP Tools API)
  - event- and callback-based
- Planned targets:
  - *MUST* correctness checking tool (SPPEXA)
  - *Score-P* / *Scalasca* (JSC)
  - *Extrae* (BSC)
  - 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(...);
...
}
```

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));
```

`xmp_init` invokes `xmpt_initialize`.

At an 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(...);
```

List of XMPT Events

- **xmpt_event_task_begin**
- **xmpt_event_task_end**
- **xmpt_event_tasks_begin**
- **xmpt_event_tasks_end**
- **xmpt_event_loop_begin**
- **xmpt_event_loop_end**
- **xmpt_event_array_begin**
- **xmpt_event_array_end**
- **xmpt_event_reflect_begin**
- **xmpt_event_reflect_begin_async**
- **xmpt_event_reflect_end**
- **xmpt_event_gmove_begin**
- **xmpt_event_gmove_begin_async**
- **xmpt_event_gmove_end**
- **xmpt_event_barrier_begin**

- xmpt_event_barrier_end
- xmpt_event_reduction_begin
- xmpt_event_reduction_begin_async
- xmpt_event_reduction_end
- xmpt_event_bcast_begin
- xmpt_event_bcast_begin_async
- xmpt_event_bcast_end
- xmpt_event_wait_async_begin
- xmpt_event_wait_async_end
- xmpt_event_coarray_remote_write
- xmpt_event_coarray_remote_read
- xmpt_event_coarray_local_write
- xmpt_event_coarray_local_read
- xmpt_event_sync_memory_begin
- xmpt_event_sync_memory_end

- xmpt_event_sync_all_begin
- xmpt_event_sync_all_end
- xmpt_event_sync_image_begin
- xmpt_event_sync_image_end
- xmpt_event_sync_images_all_begin
- xmpt_event_sync_images_all_end
- xmpt_event_sync_images_begin
- xmpt_event_sync_images_end

coarray events

Correctness Checking of XMP Programs Using XMPT

- Errors in the XMP directives

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

An error in *collectiveness* of the `bcast` directve

- <u>Data races of coarrays</u>
  - MUST could detect data races of coarrays using additional XMPT events on coarray accesses and *image control statements*.

Accesses of a coarray on multiple images in unordered segments could causes data race.

image 1

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

data race

image 2

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

Summary and outlook

- Improved programming models and environments are important for Exascale and beyond.
- Project goals and achievements of MYX
  - Extend correctness checking to XMP and YML
  - Improve productivity of XMP based codes toward Post-K Computer and many platforms
  - Improve existing parallel programming paradigms based on MPI
  - Develop high level abstractions to express parallelism based on YML scheduler/dispatcher with XMP