MYX

MUST Correctness Checking for XMP Programs

SPPEXA Workshop in Japan 2017

Presenter: Joachim Protze

IT Center | RWTH AACHEN UNIVERSITY

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;
}
```

At least 8 issues in this code example

How many errors can you spot in this tiny example?

No MPI_Init before first MPI-call

```
#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;
}
```

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;
}
```

Fortran type in C

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;
}
```

Recv-recv deadlock

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;
}
```

Rank0: src=size (out of range)

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;
}
```

Type not committed before use

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;
}
```

Type not freed before end of main

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;
}
```

Send 4 int, recv 2 int: truncation

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

Outline

- Possible issues in XMP programs
  - To analyse with static analysis
  - To analyse with dynamic, local analysis
  - To analyse with dynamic, global analysis
- How to analyse with MUST

Static Analysis Class: Restrictions (1)

- Name for newly defined entity must not be used before

```
int p

#pragma xmp nodes p(*)

#pragma xmp nodes p(4)
```

- No recursive definition of new entity, e.g. nodes

```
#pragma xmp nodes p(*)=p(0:3)
```

- Reference must be *[int-expr :] int-expr* or *:*

```
#pragma xmp align a on t(0:)
```

- *loop-index* must be a control variable of a loop in the associated loop nest:

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

  for(j=0; j<100; j++)
```

Static Analysis Class: Restrictions (2)

- *async-id* must be of type default integer

```
float i

#pragma xmp bcast a async(i)
```

- *array-name* must be declared before align directive:

```
#pragma xmp align a(i,j) with t(i,j)

float a[64][64]
```

- A shadow directive must preceed any usage of the array:

```
#pragma xmp loop on t(i,j)
  for(i=0; i<64; i++)
    for(j=0; j<64; j++)
      a[i][j]=i*3.0+j;

#pragma xmp shadow a(1,1)
```

Dynamic Local Analysis Class: Restrictions (1)

- If no nodes-spec is “*”, then the product of all nodes-spec must be equal to the total size of the entire node set in the first form, the executing node set in the second form, or the referenced node set in the third form.

```
#pragma xmp nodes p(4,4) = **

#pragma xmp nodes q(8) = p
```

- The node set specified in the on clause must be a subset of the parent node set.

```
#pragma xmp task on q(2:5)

  #pragma xmp task on p(1,:)

    #pragma xmp barrier on q(3:6)
```

- The source node specified by the from clause must belong to the node set specified by the on clause of bcast.

```
#pragma xmp bcast a from p(0) on p(4:8)
```

Dynamic Local Analysis Class: Restrictions (2)

- Several issues with gblock (non-negative, size, sum)

```
int garray = {1,-2,3,4,5};

#pragma xmp nodes p(4)

#pragma xmp template t(15)

#pragma xmp distribute t(gblock(garray))
```

  - Length of garray (5) != size of dimension in p (4)
  - Second entry of garray is negative integer
  - Sum of entries in garray (11) is different from size in t

- The value specified by *shadow-width* must be a non-negative integer

```
int i = xmp_num_nodes() - 6;

#pragma xmp shadow a(i)
```

Dynamic Local Analysis Class: Data Race

- Modify a value that is used in an async send/receive:

```
#pragma xmp bcast a on p async(10)

a++;

#pragma xmp wait_async(10)
```

- For XMP+OpenMP programs, also another thread might modify the value:

```
#pragma omp master

{

  #pragma xmp bcast a on p async(10)

  #pragma xmp wait_async(10)

}

#pragma omp single

  a++;
```

- Also race without async:

```
#pragma omp master
{
  #pragma xmp bcast a on p
}
#pragma omp single
  a++;
```

Dynamic Global Analysis Class: Restrictions

- Collective consistency:
  - All executing nodes need execute global directives
  - All provided variables must have same values across all nodes

```
#pragma xmp nodes p(*)

if (xmp_node_num()!=0){

  #pragma xmp loop (i) on p(i)

    for(i=1; i<xmp_node_num(); i++){

      // do something

    }

}
```

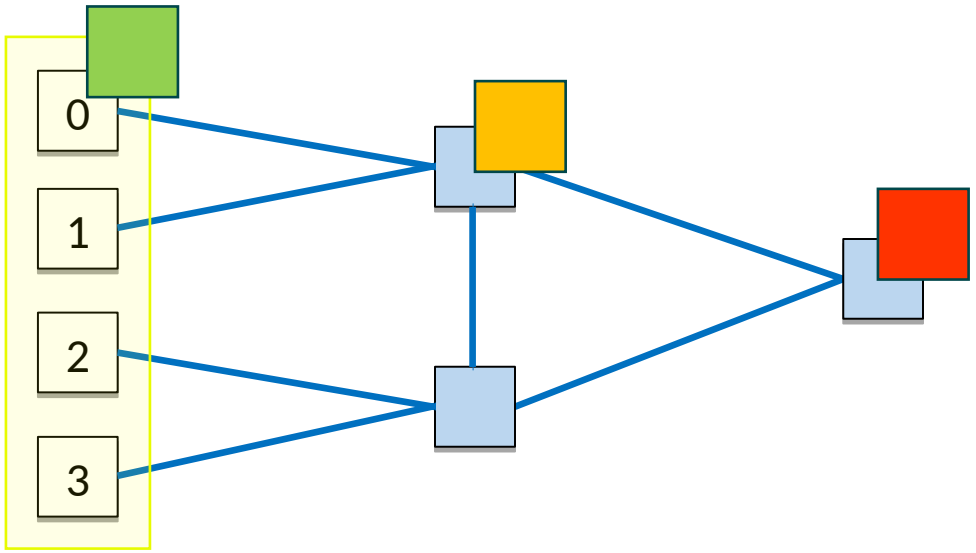
- Similar for other global constructs like array, bcast and gmove.

Dynamic Global Analysis Class: Deadlock

- Global `wait` – Deadlock:

```
#pragma xmp nodes p(4)

#pragma xmp wait
```



- P2P `wait` – Deadlock:

```
#pragma xmp nodes p(4)

#pragma xmp wait p(xmp_num_nodes() - xmp_node_num() + 1)
```

Dynamic Global Analysis Class: Data Race on Co-Array

- Data-race on Co-Array access:

```
#pragma nodes node(8)=**

double array[*];

#pragma xmp coarray on node :: array

#pragma xmp task on p(1:4)

array[5]=xmp_node_num();
```

XMPT – XMP Tools Interface

- Designed in cooperation with Hitoshi Murai
- Enables dynamic correctness analysis
- Follows the spirit of OMPT
- More details in next talk

Conclusions

- Restrictions of XMP are well-designed
- We identified 3 classes of analyses for programing issues
- MUST should be able to analyse most of the dynamic issues
- Tools interface designed to enable analysis of the issues

Vielen Dank für ihre Aufmerksamkeit