



MPI/OPENMP COURSE – MUST

AUGUST 15, 2024 | MICHAEL KNOBLOCH | M.KNOBLOCH@FZ-JUELICH.DE

Thread State	TID	Location
Breakpoint	1.1	tensorflow::SoftmaxXentWith...
Stopped	1.2	pthread_cond_wait
Stopped	1.3	pthread_cond_wait
Stopped	1.4	pthread_cond_wait

Name	Type	Value
_	int	0x0000000000000000...
nstar	int	0x000000006 (6)
grap...	int	0x000000015 (21)
[Add...]		


```

601 // Target nodes
602 const char** c_target_oper_names, int ntargets,
603 TF_Buffer* run_metadata, TF_Status* status) {
604 TF_Run_Setup(noutputs, c_outputs, status);
605 std::vector<std::pair<tensorflow::string, Tensor>> input_pairs(n
606 if (!TF_Run_Inputs(c_inputs, &input_pairs, status)) return;
607 for (int i = 0; i < ninputs; ++i) {
608   input_pairs[i].first = c_input_names[i];
609 }
610 std::vector<tensorflow::string> output_names(noutputs);
611 for (int i = 0; i < noutputs; ++i) {
612   output_names[i] = c_output_names[i];
613 }
614 std::vector<tensorflow::string> target_oper_names(ntargets);
615 for (int i = 0; i < ntargets; ++i) {
616   target_oper_names[i] = c_target_oper_names[i];
617 }
618 TF_Run_Helper(s->session, nullptr, run_options, input_pairs, outp
619 c_outputs, target_oper_names, run_metadata, status);
620 }
621
622 void TF_PRunSetup(TF_DeprecatedSession* s,
623 // Input names
624 const char** c_input_names, int ninputs,
625 // Output names
626 const char** c_output_names, int noutputs,
627 // Target nodes
628 const char** c_target_oper_names, int ntargets,
629 const char** handle, TF_Status* status) {
630   status->status = Status::OK();
631
632   std::vector<tensorflow::string> input_names(ninputs);
633   std::vector<tensorflow::string> output_names(noutputs);
634   std::vector<tensorflow::string> target_oper_names(ntargets);

```


Language	Function Name
C++	tensorflow::FunctionLibraryRunti...
C++	tensorflow::DirectSession::GetOr...
C++	std::_Function_handler<tensorflo...
C++	std::function<tensorflow::Status (...
C++	tensorflow::Sunamed_namespa...
C++	tensorflow::NewLocalExecutor
C++	tensorflow::DirectSession::GetOr...
C++	tensorflow::DirectSession::Run
C++	TF_Run_Helper
C++	TF_Run
C++	tensorflow::TF_Run_wrapper_hel...
C++	tensorflow::TF_Run_wrapper
Py	_run_fn
C	ext_do_call
Py	_do_call
Py	_do_run

MUST – MPI CORRECTNESS CHECKER

HOW MANY ISSUES CAN YOU SPOT?

```
#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 example!

MOTIVATION

- MPI programming is error prone
- Portability errors
(just on some systems, runs, configurations)
- Bugs may manifest as
 - Crash
 - Application hanging
 - Application finishes
- Questions
 - Why crash/hang?
 - Is the result correct?
 - Will the code produce the correct result on another system?
- Tools help to pin-point these issues



TYPES OF ERRORS

- Common syntactic errors:

- Incorrect arguments
- Resource usage
- Lost/Dropped Requests
- Buffer usage
- Type-matching
- Deadlocks

Tool to use:
MUST,
Static analysis tool,
(Debugger)

- Semantic errors that are correct in terms of MPI standard, but do not match the programmer's intention:

- Displacement/Size/Count errors

Tool to use:
Debugger

- Next generation MPI correctness and portability checker
- <https://www.i12.rwth-aachen.de/go/id/nrbe>
- MUST reports
 - Errors: violations of the MPI-standard
 - Warnings: unusual behavior or possible problems
 - Notes: harmless but remarkable behavior
 - Potential deadlock detection
- Usage
 - Compile with debug information (i.e. use the -g flag)
 - Run application under the control of `mustrun` (requires (at least) one additional MPI process)
 - E.g. on JUSUF: `mustrun --must:mpiexec srun --must:np -n -n 4 ./app`
 - Open output html report (might need to copy it to your local machine)

MUST 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;
}
```

FIX 0: ADD MPI_INIT/MPI_FINALIZE

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

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

```
int main(int argc, char** argv) {
```

```
    int rank, size, buf[8];
```

```
    MPI_Init(&argc, &argv);
```

```
    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 - 1, 123, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

```
    MPI_Send(buf, 2, type, size - rank - 1, 123, MPI_COMM_WORLD);
```

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

```
    MPI_Finalize();
```

```
    return 0;
```

```
}
```


MUST DETECTS DEADLOCKS

MUST Output, starting date: Fri Mar 24 11:59:41 20...

Rank(s)	Type	Message
	Error	The application issued a set of MPI calls that can cause a deadlock! A graphical representation of this situation is available in a detailed de...

Details:

Message	From	References
The application issued a set of MPI calls that can cause a deadlock! A graphical representation of this situation is available in a detailed deadlock view (MUST_Output-files/MUST_Deadlock.html) . References 1-2 list the involved calls (limited to the first 5 calls, further calls may be involved). The application still runs if the deadlock manifested (e.g. caused a hang on this MPI implementation) you can attach to the involved ranks with a debugger or abort the application (if necessary).		References of a representative process: reference 1 rank 0: MPI_Recv (1st occurrence) called from: #0 main@example.c:15 reference 2 rank 3: MPI_Recv (1st occurrence) called from: #0 main@example.c:15

Who?

What?

Where?

Details

Click for graphical representation of the detected deadlock situation.

MUST DETECTS DEADLOCKS

Message	
<p>The application issued a set of MPI calls that can cause a deadlock! The graphs below show details on this situation. This includes a wait-for graph that shows active wait-for dependencies between the processes that cause the deadlock. Note that this process set only includes processes that cause the deadlock and no further processes. A legend details the wait-for graph components in addition, while a parallel call stack view summarizes the locations of the MPI calls that cause the deadlock. Below these graphs, a message queue graph shows active and unmatched point-to-point communications. This graph only includes operations that could have been intended to match a point-to-point operation that is relevant to the deadlock situation. Finally, a parallel call stack shows the locations of any operation in the parallel call stack. The leaves of this call stack graph show the components of the message queue graph that they span. The application still runs, if the deadlock manifested (e.g. caused a hang on this MPI implementation) you can attach to the involved ranks with a debugger or abort the application (if necessary).</p>	
Active Communicators	
Comm:	A
MPI COMM WORLD	
Wait-for Graph	Legend
<p>Rank 0 waits for rank 1 and v.</p>	<p>Active MPI Call</p> <p>Sub Operation</p> <p>A A waits for B and C B</p> <p>C</p> <p>A A waits for B or C B</p> <p>C</p>
Call Stack	
<p>main@/rwthfs/rz/cluster/home/pj416018/must-example/VI-HPS/example.c:15</p> <p>Ranks:</p> <p>MPI_Recv</p> <p>Simple call stack for this example.</p>	
Active and Relevant Point-to-Point Messages: Overview	
Message queue	
Active and Relevant Point-to-Point Messages: Callstack-view	
stack	

FIX 1: USE ASYNC RECV

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

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

```
    MPI_Init(&argc, &argv);
```

```
    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_Request request;
    MPI_Irecv(buf, 2, MPI_INT, size - rank - 1, 123, MPI_COMM_WORLD, &request);
```

```
    MPI_Send (buf, 2, type, size - rank - 1, 123, MPI_COMM_WORLD);
```

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

```
    MPI_Finalize();
```

```
    return 0;
}
```

Use asynchronous
receive (MPI_Irecv)

MUST DETECTS BUFFER ERRORS

Rank(s)	Type	Message	From	References
2(28793)	Error	A receive operation uses a (datatype, count) pair that can not hold the data transferred by the send it matches! The first element of the send...		
Details:				
		<p>A receive operation uses a (datatype, count) pair that can not hold the data transferred by the send it matches! The first element of the send that did not fit into the receive operation is at (contiguous)[0](MPI_INTEGER) in the send type (consult the MUST manual for a detailed description of datatype positions). The send operation was started at reference 1, the receive operation was started at reference 2. (Information on communicator: MPI_COMM_WORLD) (Information on send of count 2 with type:Datatype created at reference 3 is for Fortran, based on the following type(s): { MPI_INTEGER}) (Information on receive of count 2 with type:MPI_INT)</p>	<p>Representative location: MPI_Send (1st occurrence) called from: #0 main@example-fix1.c:18</p>	<p>References of a representative process:</p> <p>reference 1 rank 2: MPI_Send (1st occurrence) called from: #0 main@example-fix1.c:18</p> <p>reference 2 rank 1: MPI_Irecv (1st occurrence) called from: #0 main@example-fix1.c:16</p> <p>reference 3 rank 2: MPI_Type_contiguous (1st occurrence) called from: #0 main@example-fix1.c:13</p>
1(28792)	Error	A receive operation uses a (datatype, count) pair that can not hold the data transferred by the send it matches! The first element of the send...		
0-3	Error	Argument 3 (datatype) is not committed for transfer, call MPI Type commit before using the type for transfer!(Information on datatypeData...		
2(28793)	Error	The memory regions to be transferred by this send operation overlap with regions spanned by a pending non-blocking receive operation!(In...		
1(28792)	Error	The memory regions to be transferred by this send operation overlap with regions spanned by a pending non-blocking receive operation!(In...		
3(28795)	Error	The memory regions to be transferred by this send operation overlap with regions spanned by a pending non-blocking receive operation!(In...		
3(28795)	Error	A receive operation uses a (datatype, count) pair that can not hold the data transferred by the send it matches! The first element of the send...		
0(28794)	Error	The memory regions to be transferred by this send operation overlap with regions spanned by a pending non-blocking receive operation!(In...		
0(28794)	Error	A receive operation uses a (datatype, count) pair that can not hold the data transferred by the send it matches! The first element of the send...		

Size of sent message larger than receive buffer

All detected errors are collapsed for overview - click to expand

FIX 2: SAME MESSAGE SIZE FOR SEND/RECV

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

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

```
    MPI_Init(&argc, &argv);
```

```
    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_Request request;
    MPI_Irecv(buf, 2, MPI_INT, size - rank - 1, 123, MPI_COMM_WORLD, &request);
```

```
    MPI_Send (buf, 1, type, size - rank - 1, 123, MPI_COMM_WORLD);
```

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

```
    MPI_Finalize();
```

```
    return 0;
```

```
}
```

Reduce the
message size

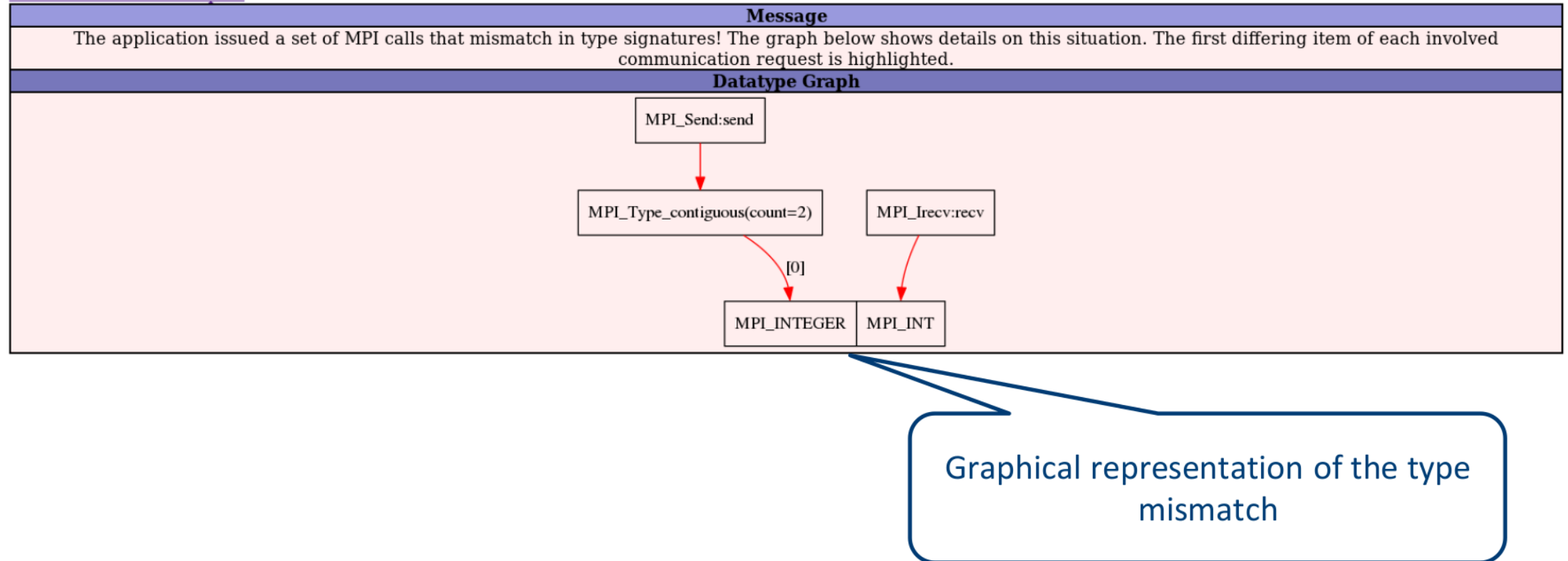
MUST DETECTS DATATYPE ERRORS

Rank(s)	Type	Message						
2(17250)	Error	A send and a receive operation use datatypes that do not match! Mismatch occurs at (contiguous)[0](MPI_INTEGER) in the send type and a...						
Details:								
		<table border="1"> <thead> <tr> <th>Message</th> <th>From</th> <th>References</th> </tr> </thead> <tbody> <tr> <td>A send and a receive operation use datatypes that do not match! Mismatch occurs at (contiguous)[0](MPI_INTEGER) in the send type and at (MPI_INT) in the receive type (consult the MUST manual for a detailed description of datatype positions). A graphical representation of this situation is available in a detailed type mismatch view (MUST Output-files/MUST_Typemismatch_74088185856002.html). The send operation was started at reference 1, the receive operation was started at reference 2. (Information on communicator: MPI_COMM_WORLD) (Information on send of count 1 with type:Datatype created at reference 3 is for Fortran, based on the following type(s): { MPI_INTEGER}) (Information on receive of count 2 with type:MPI_INT)</td> <td>Representative location: MPI_Send (1st occurrence) called from: #0 main@example-fix2.c:18</td> <td>References of a representative process: reference 1 rank 2: MPI_Send (1st occurrence) called from: #0 main@example-fix2.c:18 reference 2 rank 1: MPI_Irecv (1st occurrence) called from: #0 main@example-fix2.c:16 reference 3 rank 2: MPI_Type_contiguous (1st occurrence) called from: #0 main@example-fix2.c:13</td> </tr> </tbody> </table>	Message	From	References	A send and a receive operation use datatypes that do not match! Mismatch occurs at (contiguous)[0](MPI_INTEGER) in the send type and at (MPI_INT) in the receive type (consult the MUST manual for a detailed description of datatype positions). A graphical representation of this situation is available in a detailed type mismatch view (MUST Output-files/MUST_Typemismatch_74088185856002.html) . The send operation was started at reference 1, the receive operation was started at reference 2. (Information on communicator: MPI_COMM_WORLD) (Information on send of count 1 with type:Datatype created at reference 3 is for Fortran, based on the following type(s): { MPI_INTEGER}) (Information on receive of count 2 with type:MPI_INT)	Representative location: MPI_Send (1st occurrence) called from: #0 main@example-fix2.c:18	References of a representative process: reference 1 rank 2: MPI_Send (1st occurrence) called from: #0 main@example-fix2.c:18 reference 2 rank 1: MPI_Irecv (1st occurrence) called from: #0 main@example-fix2.c:16 reference 3 rank 2: MPI_Type_contiguous (1st occurrence) called from: #0 main@example-fix2.c:13
Message	From	References						
A send and a receive operation use datatypes that do not match! Mismatch occurs at (contiguous)[0](MPI_INTEGER) in the send type and at (MPI_INT) in the receive type (consult the MUST manual for a detailed description of datatype positions). A graphical representation of this situation is available in a detailed type mismatch view (MUST Output-files/MUST_Typemismatch_74088185856002.html) . The send operation was started at reference 1, the receive operation was started at reference 2. (Information on communicator: MPI_COMM_WORLD) (Information on send of count 1 with type:Datatype created at reference 3 is for Fortran, based on the following type(s): { MPI_INTEGER}) (Information on receive of count 2 with type:MPI_INT)	Representative location: MPI_Send (1st occurrence) called from: #0 main@example-fix2.c:18	References of a representative process: reference 1 rank 2: MPI_Send (1st occurrence) called from: #0 main@example-fix2.c:18 reference 2 rank 1: MPI_Irecv (1st occurrence) called from: #0 main@example-fix2.c:16 reference 3 rank 2: MPI_Type_contiguous (1st occurrence) called from: #0 main@example-fix2.c:13						
0(17249)	Error	that do not m						
1(17248)	Error	that do not m						
3(17251)	Error	that do not						
0-3	Error	transfer, call						
Details:								
		<table border="1"> <thead> <tr> <th>Message</th> <th>From</th> <th>References</th> </tr> </thead> <tbody> <tr> <td>Argument 3 (datatype) is not committed for transfer, call MPI_Type_commit before using the type for transfer! (Information on datatypeDatatype created at reference 1 is for Fortran, based on the following type(s): { MPI_INTEGER})</td> <td>Representative location: MPI_Send (1st occurrence) called from: #0 main@example-fix2.c:18</td> <td>References of a representative process: reference 1 rank 2: MPI_Type_contiguous (1st occurrence) called from: #0 main@example-fix2.c:13</td> </tr> </tbody> </table>	Message	From	References	Argument 3 (datatype) is not committed for transfer, call MPI_Type_commit before using the type for transfer! (Information on datatypeDatatype created at reference 1 is for Fortran, based on the following type(s): { MPI_INTEGER})	Representative location: MPI_Send (1st occurrence) called from: #0 main@example-fix2.c:18	References of a representative process: reference 1 rank 2: MPI_Type_contiguous (1st occurrence) called from: #0 main@example-fix2.c:13
Message	From	References						
Argument 3 (datatype) is not committed for transfer, call MPI_Type_commit before using the type for transfer! (Information on datatypeDatatype created at reference 1 is for Fortran, based on the following type(s): { MPI_INTEGER})	Representative location: MPI_Send (1st occurrence) called from: #0 main@example-fix2.c:18	References of a representative process: reference 1 rank 2: MPI_Type_contiguous (1st occurrence) called from: #0 main@example-fix2.c:13						

Use of Fortran type in C, datatype mismatch between sender and receiver

Use of uncommitted datatype: type

MUST DETECTS DATATYPE ERRORS



FIX 3+4: C INT TYPE & USE TYPE_COMMIT

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

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

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

```
    MPI_Init(&argc, &argv);
```

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

```
    MPI_Datatype type;  
    MPI_Type_contiguous(2, MPI_INT, &type);  
    MPI_Type_commit(&type);
```

```
    MPI_Request request;  
    MPI_Irecv(buf, 2, MPI_INT, size - rank - 1, 123, MPI_COMM_WORLD, &request);
```

```
    MPI_Send(buf, 1, type, size - rank - 1, 123, MPI_COMM_WORLD);
```

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

```
    MPI_Finalize();
```

```
    return 0;
```

```
}
```

Use integer datatype
intended for C

Commit datatype
before usage

MUST DETECTS DATARACES IN ASYNC COMM

Data race between send and asynchronous receive operation

Rank(s)	Type		
0(1605)	Error	The memory regions to be transferred by this	receive operation!(In...
Details:			
Message		From	References
<p>The memory regions to be transferred by this send operation overlap with regions spanned by a pending non-blocking receive operation!</p> <p>(Information on the request associated with the other communication: Point-to-point request activated at reference 1)</p> <p>(Information on the datatype associated with the other communication: MPI_INT)</p> <p>The other communication overlaps with this communication at position:(MPI_INT)</p> <p>(Information on the datatype associated with this communication: Datatype created at reference 2 is for C, committed at reference 3, based on the following type(s): { MPI_INT})</p> <p>This communication overlaps with the other communication at position:(contiguous) [0](MPI_INT)</p> <p>A graphical representation of this situation is available in a detailed overlap view (MUST Output-files/MUST_Overlap_6893422510080_0.html).</p>		<p>Representative location: MPI_Send (1st occurrence) called from: #0 main@example-fix3.c:19</p>	<p>References of a representative process:</p> <p>reference 1 rank 0: MPI_Irecv (1st occurrence) called from: #0 main@example-fix3.c:17</p> <p>reference 2 rank 0: MPI_Type_contiguous (1st occurrence) called from: #0 main@example-fix3.c:13</p> <p>reference 3 rank 0: MPI_Type_commit (1st occurrence) called from: #0 main@example-fix3.c:14</p>
3(1610)	Error	The memory regions to be transferred by this send operation overlap with regions spanned by a pending non-blocking receive operation!(In...	
2(1608)	Error	The memory regions to be transferred by this send operation overlap with regions spanned by a pending non-blocking receive operation!(In...	
1(1606)	Error	The memory regions to be transferred by this send operation overlap with regions spanned by a pending non-blocking receive operation!(In...	
0-3	Error	There are 1 datatypes that are not freed when MPI Finalize was issued, a quality application should free all MPI resources before calling ...	
0-3	Error	There are 1 requests that are not freed when MPI Finalize was issued, a quality application should free all MPI resources before calling M...	

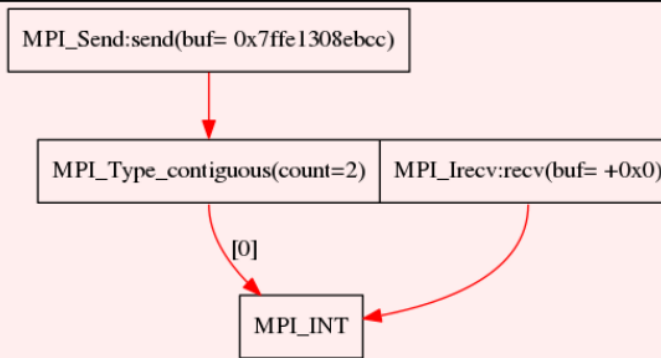
MUST DETECTS DATARACES IN ASYNC COMM

Graphical representation of the data race location

Message

The application issued a set of MPI calls that overlap in communication buffers! The graph below shows details on this situation. The first colliding item of each involved communication request is highlighted.

Datatype Graph



FIX 5: USE INDEPENDENT MEMORY REGIONS

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

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

```
    MPI_Init(&argc, &argv);
```

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

```
    MPI_Datatype type;
    MPI_Type_contiguous(2, MPI_INT, &type);
    MPI_Type_commit(&type);
```

```
    MPI_Request request;
    MPI_Irecv(buf, 2, MPI_INT, size - rank - 1, 123, MPI_COMM_WORLD, &request);
```

```
    MPI_Send (buf + 2, 1, type, size - rank - 1, 123, MPI_COMM_WORLD);
```

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

```
    MPI_Finalize();
```

```
    return 0;
}
```

Offset points to
independent memory

MUST DETECTS LEAKS OF USER-DEFINED OBJECTS

Rank(s)	Type	Message	
0-3	Error	There are 1 datatypes that are not freed when MPI Finalize was issued, a quality application should free all MPI resources before calling ...	
Details:			
Message		From	References
There are 1 datatypes that are not freed when MPI_Finalize was issued, a quality application should free all MPI resources before calling MPI_Finalize. Listing information for these datatypes: -Datatype 1: Datatype created at reference 1 is for C, committed at reference 2, based on the following type(s): { MPI_INT}		Representative location: MPI_Type_contiguous (1st occurrence) called from: #0 main@example-fix4.c:13	References of a representative process: reference 1 rank 1: MPI_Type_contiguous (1st occurrence) called from: #0 main@example-fix4.c:13 reference 2 rank 1: MPI_Type_commit (1st occurrence) called from: #0 main@example-fix4.c:14
0-3	Error	There are 1 requests that are not freed when MPI Finalize was issued, a quality application should free all MPI resources before calling M...	
Details:			
Message		From	References
There are 1 requests that are not freed when MPI_Finalize was issued, a quality application should free all MPI resources before calling MPI_Finalize. Listing information for these requests: -Request 1: Point-to-point request activated at reference 1		Representative location: MPI_irecv (1st occurrence) called from: #0 main@example-fix4.c:17	References of a representative process: reference 1 rank 1: MPI_irecv (1st occurrence) called from: #0 main@example-fix4.c:17

- User defined objects include
 - MPI_Comms (even by MPI_Comm_dup)
 - MPI_Datatypes
 - MPI_Groups

Unfinished non-blocking receive is resource leak and missing synchronization

Leak of user defined datatype object

FIX 6+7: USE MPI_WAIT & FREE DATATYPE

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

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

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

```
    MPI_Init(&argc, &argv);
```

```
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
    MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```
    MPI_Datatype type;
```

```
    MPI_Type_contiguous(2, MPI_INT, &type);
```

```
    MPI_Type_commit(&type);
```

```
    MPI_Request request;
```

```
    MPI_Irecv(buf, 2, MPI_INT, size - rank - 1, 123, MPI_COMM_WORLD, &request);
```

```
    MPI_Send(buf + 2, 1, type, size - rank - 1, 123, MPI_COMM_WORLD);
```

```
    MPI_Wait(&request, MPI_STATUS_IGNORE);
```

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

```
    MPI_Type_free(&type);
```

```
    MPI_Finalize();
```

```
    return 0;
```

Finish asynchronous
communication

Deallocate datatype

FINALLY

Rank(s)	Type	Message	
	Information	MUST detected no MPI usage errors nor any suspicious behavior during this application run.	
Details:			
	Message	From	References
	MUST detected no MPI usage errors nor any suspicious behavior during this application run.		

No further error detected

Hopefully this message applies to many applications

RUNNING MUST - CENTRALIZED

- **Slow, Centralized, Application may crash**
 - `mustrun --np X exe`
 - One additional analysis process
 - High overhead, < 32 processes
- **Fast, Centralized, Application may not crash**
 - `mustrun --np X --must:nocrash exe`
 - One extra analysis process
 - Limited scalability, < 100 processes
- **Fast, Centralized, Application may crash**
 - `mustrun --np X --must:nodesize Y exe [--must:show]`
 - $1 + \lceil X / (Y - 1) \rceil$ analysis processes
 - Limited scalability, < 100 processes
 - Requires shared memory communication

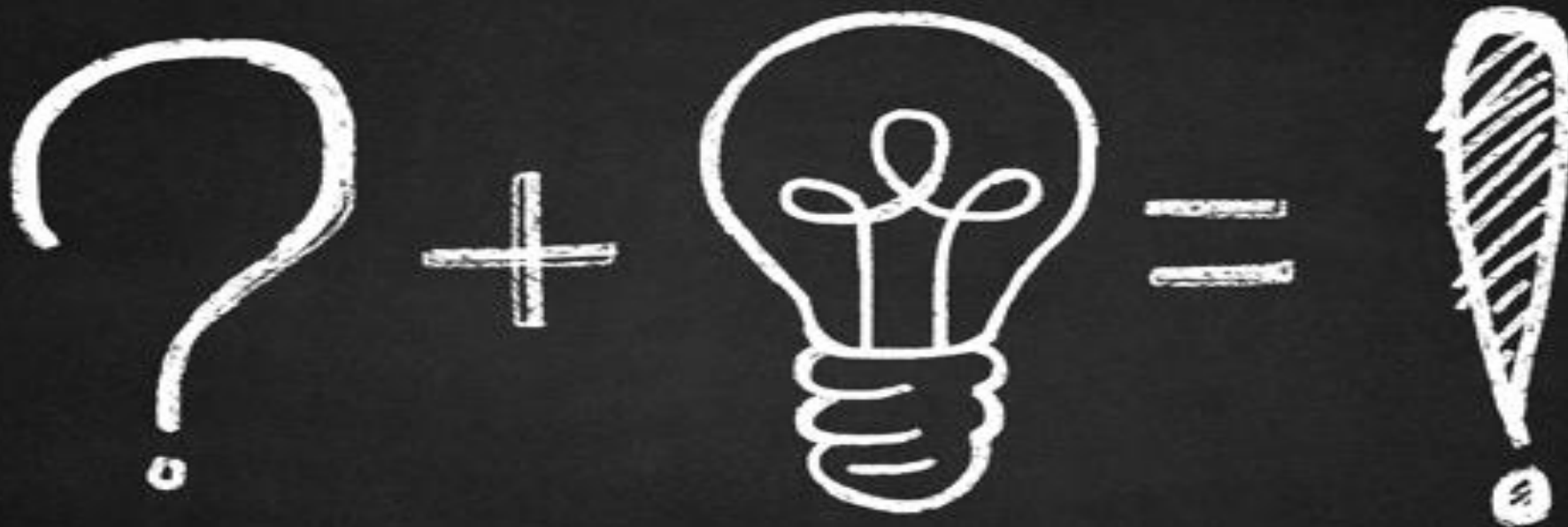
RUNNING MUST - DISTRIBUTED

- **Distributed, Application may not crash**

- `mustrun --np X --must:distributed [--must:fanin Z] exe`
- Layer 0: $A = [X/Z]$
- Layer 1: $B = [A/Z]$
- ...
- Layer k: 1
- Scalability > 10000 processes

- **Distributed, Application may crash**

- `mustrun --np X --must:distributed --must:nodesize Y [--must:fanin Z] exe`
- $A = [X/(Y - 1)] \rightarrow B = [A/Z] \rightarrow C = [B/Z] \rightarrow \dots \rightarrow 1$
- Scalability ~ 5000 processes
- Requires shared memory communication



QUESTIONS