
# VERIFYING MPI EXECUTION CONFORMANCE WITH MUST

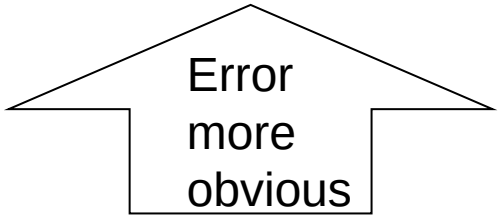
28.11.2018 | MICHAEL KNOBLOCH

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
Forschungszentrum

# MOTIVATION

- MPI programming is error prone
  - Complex API
- Bugs may manifest as
  - Crash
  - Application hanging
  - Application finishes (but gives wrong result)




- Questions
  - Why crash/hang?
  - Is my result correct?
  - Will my code also give correct results on another system?
- MUST helps to pin-point these bugs

# MUST

- Next generation MPI correctness and portability checker
- http://doc.itc.rwth-aachen.de/display/CCP/Project+MUST

- MUST reports
  - Errors: violations of the MPI-standard
  - Warnings: unusual behavior or possible problems
  - Notes:  harmless but remarkable behavior
  - Further: potential deadlock detection

- Stores output in html files for easy analysis

- Scalability of MUST depends strongly on the scalability of the attached application.

# MUST ERROR CLASSES

- Constants and integer values
- Communicator usage
- Datatype usage
- Group usage
- Operation usage
- Request usage
- Leak checks (MPI resources not freed before calling MPI Finalize)
- Type mis-matches
- Overlapping buffers passed to MPI
- Deadlocks resulting from MPI calls
- Basic checks for thread level usage (MPI_Init_thread)

# MUST: EXAMPLE

- Usage
  - Compile application with debug symbols (-g)
  - Run application under the control of mustrun
    - Requires one additional MPI process
  - See MUST_Output.html report

```
% module load Intel IntelMPI MUST
% mpiicc -g -o deadlock deadlock.c

############################
##  In the job script:  ##
############################


mustrun --must:mpiexec srun --must:np -n\
-n 4 ./deadlock
```

# MUST: EXAMPLE OUTPUT

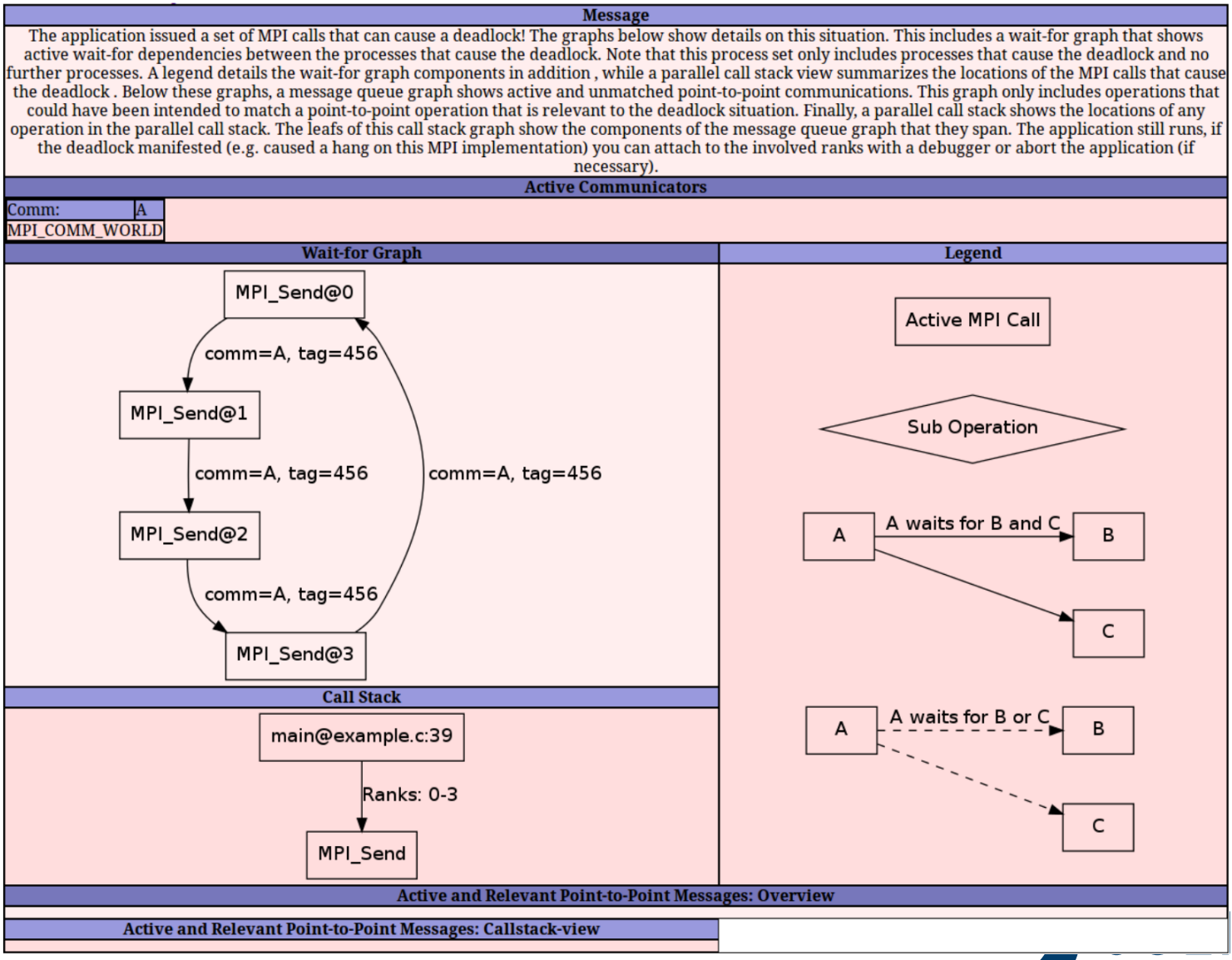
```
[MUST] MUST configuration ... centralized checks with fall-back application crash handling (very slow)
[MUST] Information: overwritting old intermediate data in directory "/path/to/application/must_temp"!
[MUST] Using prebuilt infrastructure at /usr/local/software/jureca/Stages/2016b/software/MUST/\
       1.5.0-iimpi-2016b-Python-2.7.12/modules//mode1-layer2
[MUST] Weaver ... success
[MUST] Generating P^nMPI configuration ... success
[MUST] Search for linked P^nMPI ... not found ... using LD_PRELOAD to load P^nMPI ... success
[MUST] Executing application:
…
Application output
…
============MUST===============
ERROR: MUST detected a deadlock, detailed information is available in the MUST output file.
You should either investigate details with a debugger or abort, the operation of MUST will stop from now.
==============================
[MUST] Execution finished, inspect "/path/to/application/MUST_Output.html"!
```

| 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 deadlock view (MUST_Output-files/MUST_Deadlock.ht… |

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 1-4 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:<br><br>reference 1 rank 0: call MPI_Send (1st occurrence)<br><br>reference 2 rank 1: call MPI_Send (1st occurrence)<br><br>reference 3 rank 2: call MPI_Send (1st occurrence)<br><br>reference 4 rank 3: call MPI_Send (1st occurrence) |

# MUST DEADLOCK DETECTION



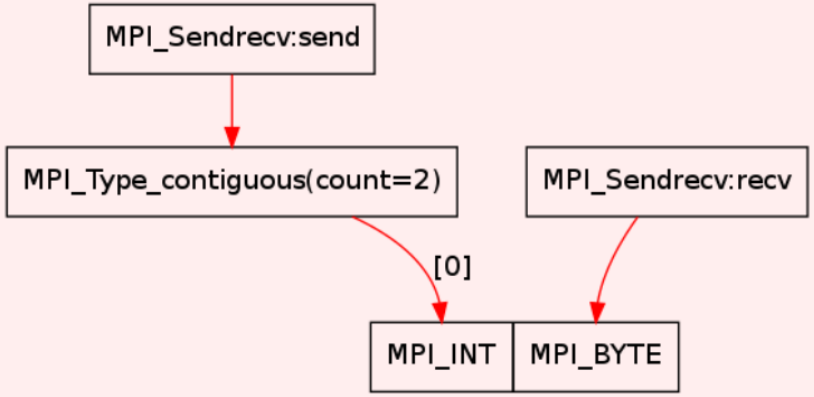
# MUST DATATYPE MISMATCH

| Rank | Type | Message | From | References |
|---|---|---|---|---|
| 0 | Error | A send and a receive operation use datatypes that do not match! Mismatch occurs at (contiguous) [0](MPI_INT) in the send type and at (MPI_BYTE) 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_0.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 C, commited at reference 4, based on the following type(s): { MPI_INT}Typemap = {(MPI_INT, 0), (MPI_INT, 4)}) (Information on receive of count 8 with type:MPI_BYTE) | **MPI_Sendrecv** called from: #0 main@example.c:33 | reference 1 rank 0: **MPI_Sendrecv** called from: #0 main@example.c:33<br><br>reference 2 rank 1: **MPI_Sendrecv** called from: #0 main@example.c:33<br><br>reference 3 rank 0: **MPI_Type_contiguous** called from: #0 main@example.c:29<br><br>reference 4 rank 0: **MPI_Type_commit** called from: #0 main@example.c:30 |

**Message**

The application issued a set of MPI calls that mismatch in type signatures! The graph below shows details on this situation. The first differing item of each involved communication request is highlighted.

**Datatype Graph**



# HANDS-ON: QUIZ

- How many issues are in the following MPI program?

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

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

  return 0;
}
```

# FIX 1: INIT AND FINALIZE MPI

```c
#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'm rank %d of %d.\n", rank, size);

  MPI_Finalize ();

  return 0;
}
```

# FIX 2: USE ASYNCHRONOUS RECEIVE

```c
#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'm rank %d of %d.\n", rank, size);

  MPI_Finalize ();

  return 0;
}
```

# FIX 3: SAME MESSAGE SIZE FOR SEND/RECEIVE

```c
#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'm rank %d of %d.\n", rank, size);

  MPI_Finalize ();

  return 0;
}
```

# FIX 4: USE MPI_TYPE_COMMIT AND MPI_INT

```c
#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'm rank %d of %d.\n", rank, size);

  MPI_Finalize ();

  return 0;
}
```

# FIX 5: USE INDEPENDEND MEMORY REGIONS

```c
#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 + 4, 1, type, size - rank - 1, 123, MPI_COMM_WORLD);

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

  MPI_Finalize ();

  return 0;
}
```

# FIX 6: FREE DATATYPE AND USE MPI_WAIT

```c
#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 + 4, 1, type, size - rank - 1, 123, MPI_COMM_WORLD);
  MPI_Wait (&request, MPI_STATUS_IGNORE);

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

  MPI_Type_free (&type);
  MPI_Finalize ();

  return 0;
}
```