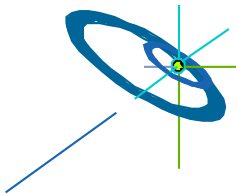
MPI Correctness Checking with MARMOT

Bettina Krammer
University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de

Matthias Müller
University of Dresden
Centre for Information Services and High Performance Computing (ZIH)
www.tu-dresden.de/zih

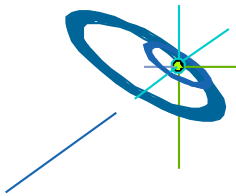


ZIH
Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

Overview

- Typical MPI Programming Errors
- What is Marmot?
- Examples
- Exercises

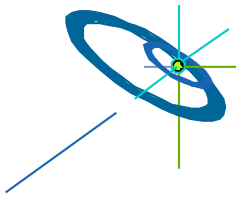


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

HLRS

Common MPI programming errors I – Collective Routines

- Argument mismatches (e.g. different send/recv-counts in Gather)
- Deadlocks: not all processes call the same collective routine
  - E.g. all procs call Gather, except for one that calls Allgather
  - E.g. all procs call Bcast, except for one that calls Send before Bcast, matching Recv is called after Bcast
  - E.g. all procs call Bcast, then Gather, except for one that calls Gather first and then Bcast



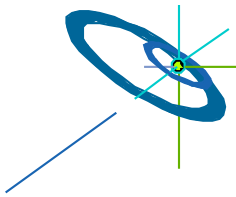
Common MPI programming errors II – Point-to-Point Routines

- Deadlocks: matching routine is not called, e.g.

  Proc0: MPI_Send(…)
  MPI_Recv(..)
  Proc1: MPI_Send(…)
  MPI_Recv(…)

- Argument mismatches
  - different **datatypes** in Send/Recv pairs, e.g.

    Proc0: MPI_Send(1, MPI_INT)
    Proc1: MPI_Recv(8, MPI_BYTE)
    Illegal!



ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

Common MPI programming errors III – Point-to-Point Routines

- especially tricky with user-defined datatypes, e.g.

  MPI_INT 

  MPI_DOUBLE 

  derived datatype 1: DER_1 

  derived datatype 2: DER_2 

  derived datatype 3: DER_3 

  MPI_Send(2, DER_1), MPI_Recv(1, DER_2) is legal

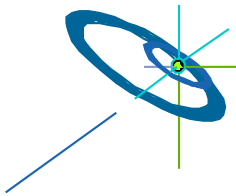
  MPI_Send(2, DER_1), MPI_Recv(1, DER_3) is illegal

- different **counts** in Send/Recv pairs are allowed as *Partial Receive*

  MPI_Send(1, DER_1), MPI_Recv(1, DER_2) is legal

  MPI_Send(1, DER_1), MPI_Recv(1, DER_3) is legal

  MPI_Send(1, DER_2), MPI_Recv(1, DER_1) is illegal



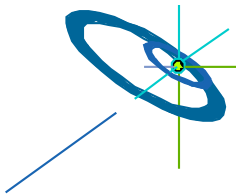
ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

Common MPI programming errors IV – Point-to-Point Routines

- Incorrect resource handling
  - **Non-blocking calls (e.g. Isend, Irecv) can complete without issuing test/wait call, BUT:**

    Number of available request handles is limited (and implementation defined)
  - **Free request handles before you reuse them (either with wait/successful test routine or MPI_Request_free)**

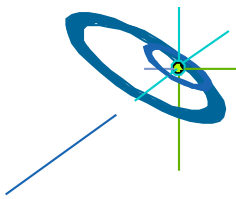


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

HLRS

Common MPI programming errors V – Others

- Incorrect resource handling
  - Incorrect creation or usage of resources such as communicators, datatypes, groups, etc.
  - Reusing an active request
  - Passing wrong number and/or types of parameters to MPI calls (often detected by compiler)
- Memory and other resource exhaustion
  - Read/write from/into buffer that is still in use, e.g. by an unfinished Send/Recv operation
  - Allocated communicators, derived datatypes, request handles, etc. were not freed
- Outstanding messages at Finalize
- MPI-standard 2: I/O errors etc.

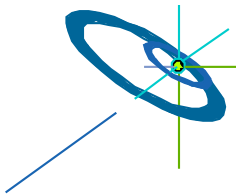


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

HLRS

Common MPI programming errors VI – Race conditions

- Irreproducibility
  - Results may sometimes be wrong
  - Deadlocks may occur sometimes
- Possible reasons:
  - Use of wild cards (MPI_ANY_TAG, MPI_ANY_SOURCE)
  - Use of random numbers etc.
  - Nodes do not behave exactly the same (background load, …)
  - No synchronization of processes
- Bugs can be very nasty to track down in this case!
- Bugs may never occur in the presence of a tool (so-called *Heisenbugs*)

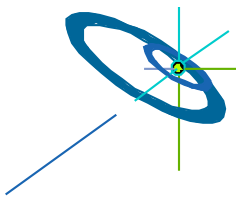


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

HLRS

Common MPI programming errors VII – Portability issues

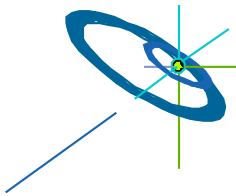
- MPI standard leaves some decisions to implementors, portability therefore not guaranteed!
  - "Opaque objects" (e.g. MPI groups, datatypes, communicators) are defined by implementation and are accessible via handles.
    - **For example, in mpich, MPI_Comm is an int**
    - **In lam-mpi, MPI_Comm is a pointer to a struct**
  - Message buffering implementation-dependent (e.g. for Send/Recv operations)
    - **Use Isend/Irecv**
    - **Bsend (usually slow, beware of buffer overflows)**
  - Synchronizing collective calls implementation-dependent
  - Thread safety not guaranteed



ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

HLRS

What is Marmot?

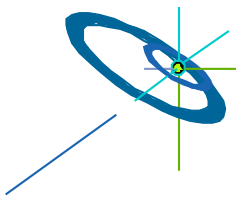


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

What is MARMOT?

- Tool for the development of MPI applications
- Automatic runtime analysis of the application:
  - Detect incorrect use of MPI
  - Detect non-portable constructs
  - Detect possible race conditions and deadlocks
- MARMOT does not require source code modifications, “just” relink and run with 1 additional process
- C and Fortran binding of MPI -1.2 is supported, also C++ and mixed C/Fortran code
- Development is still ongoing (not every possible functionality is implemented yet…)
- Tool makes use of the so-called *profiling interface*



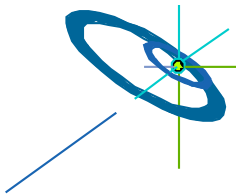
ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

What is the profiling interface?

- Defined in the MPI standard
- Every MPI routine can also be called as the nameshifted routine PMPI.
- This allows users to replace MPI routines by there own routines.
- Example (MARMOT): redefine the MPI calls

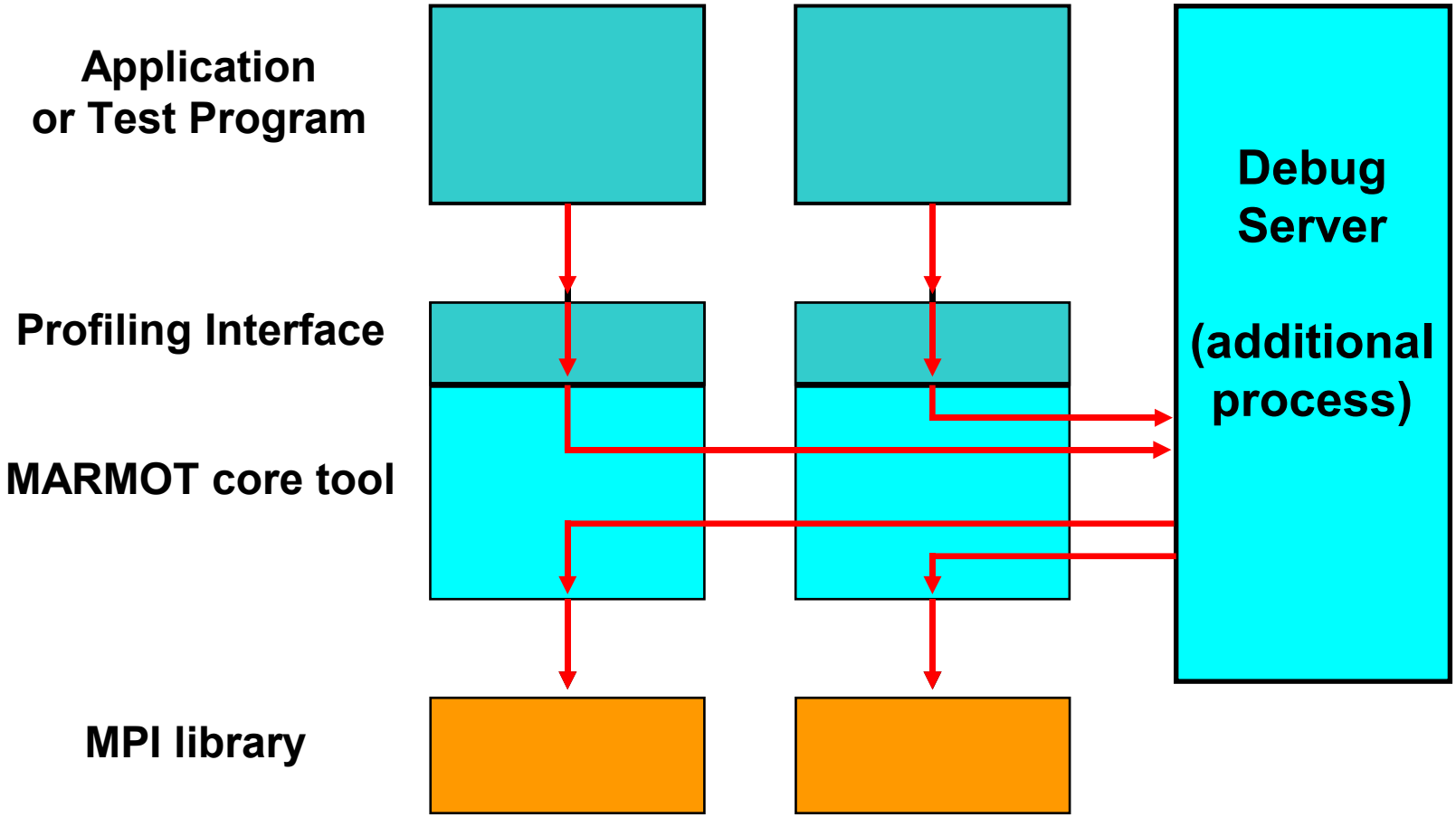
```
MPI_Send {
    doSomeChecks();
    PMPI_Send(…);
 }
```



ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

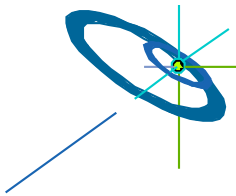
Design of MARMOT



Availability of MARMOT

- Tested on different platforms, using different compilers (Intel, GNU,…) and MPI implementations (mpich, lam, Open MPI, vendor MPIs,…), e.g.
  - IA32/IA64 clusters
  - Opteron clusters
  - Xeon EM64T clusters
  - IBM
  - NEC SX5,…, SX8

- http://www.hlrs.de/organization/amt/projects/marmot/



ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

Future Directions

Functionality

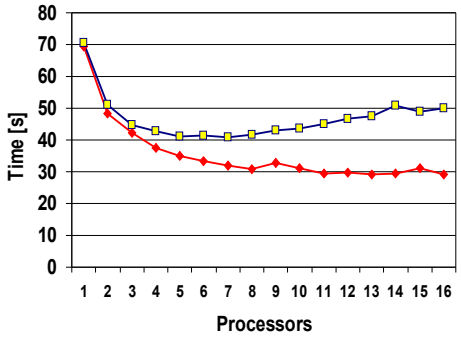
-More checks

-MPI-2

-OpenMP/MPI

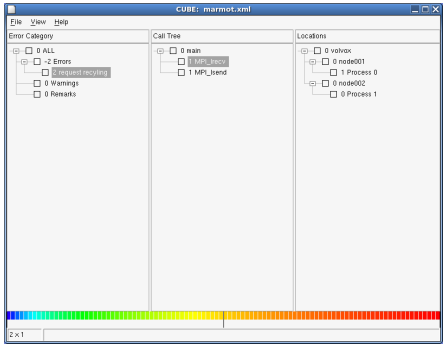
Performance

- Scalability



Usability

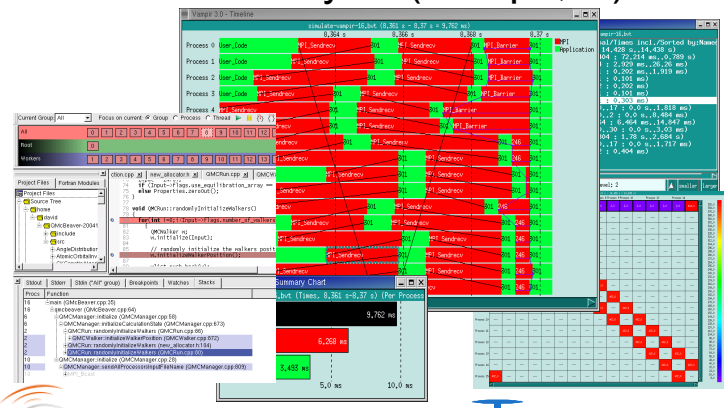
- GUI



Combination with other tools

-Debugger (DDT)

-Performance analysis (Vampir,…)



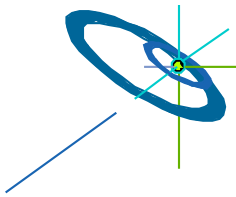
ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

HLRS

Examples

Example 1: request-reuse (source code)

```
#include <stdio.h>
#include <assert.h>
#include "mpi.h"
#include "enhancempicalls"

int main( int argc, char **argv ) {
  int size   = -1;
  int rank   = -1;
  int value  = -1;
  int value2 = -1;
  MPI_Status  send_status, recv_status;
  MPI_Request send_request, recv_request;

  printf( "We call Irecv and Isend with non-freed requests.\n" );
  MPI_Init( &argc, &argv );
  MPI_Comm_size( MPI_COMM_WORLD, &size );
  MPI_Comm_rank( MPI_COMM_WORLD, &rank );
  printf( " I am rank %d of %d PEs\n", rank, size );
```

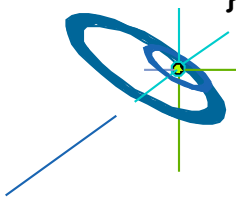


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

Example 1: request-reuse (source code continued)

```
if( rank == 0 ){
   /* get the request used */
   MPI_Irecv( &value, 1, MPI_INT, 1, 18, MPI_COMM_WORLD, &recv_request );
   /* receive the message and reuse a non-freed request */
   MPI_Irecv( &value, 1, MPI_INT, 1, 17, MPI_COMM_WORLD, &recv_request );
   MPI_Wait( &recv_request, &recv_status );
   assert( value = 19 );
 }
 if( rank == 1 ){
   value2 = 19;
   /* use the request */
   MPI_Isend( &value, 1, MPI_INT, 0, 18, MPI_COMM_WORLD, &send_request );
   /* send the message */
   MPI_Isend( &value2, 1, MPI_INT, 0, 17, MPI_COMM_WORLD,
  &send_request );
   MPI_Wait( &send_request, &send_status );
 }
 MPI_Finalize();
 return 0;
}
```

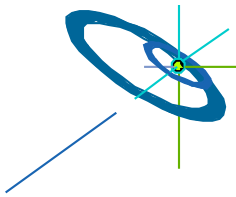


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

HLRS

Example 1: request-reuse (output log – old, mpich)

```
   We call Irecv and Isend with non-freed requests.
1 rank 0 performs MPI_Init
2 rank 1 performs MPI_Init
3 rank 0 performs MPI_Comm_size
4 rank 1 performs MPI_Comm_size
5 rank 0 performs MPI_Comm_rank
6 rank 1 performs MPI_Comm_rank
   I am rank 0 of 2 PEs
7 rank 0 performs MPI_Irecv
   I am rank 1 of 2 PEs
8 rank 1 performs MPI_Isend
9 rank 0 performs MPI_Irecv
10 rank 1 performs MPI_Isend
   ERROR: MPI_Irecv Request is still in use !!
11 rank 0 performs MPI_Wait
   ERROR: MPI_Isend Request is still in use !!
12 rank 1 performs MPI_Wait
13 rank 0 performs MPI_Finalize
14 rank 1 performs MPI_Finalize
```

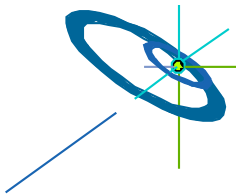


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

HLRS

Example 1: request-reuse (output log – new, Open MPI)

```
...
10: Note from rank 1 with  Text: performing On Call: MPI_Isend
 From: request-reuse2.c line: 73
10: Error from rank 0 with  Text: ERROR: MPI_Irecv Request is still in use
!!
Argument: request

Information for Resource of type MPI_Request:
created at request-reuse2.c line: 55
not yet freed.

On Call: MPI_Irecv From: request-reuse2.c line: 59
for MPI-Standard information
see:/opt/marmot/marmot_icc_openmpi/share/doc/cg-wp2
.2-marmot-cvs2007070613/MPI-STANDARD/marmot_err/node92.html

10: Error from rank 1 with  Text: ERROR: MPI_Isend Request is still in use
!!

....
```

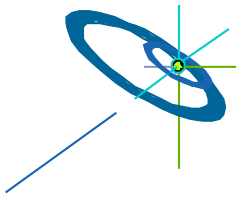


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

HLRS

Example 1: request-reuse (output log – new, Open MPI)

```
From: request-reuse2.c line: 78
13: Warning from rank 1 with  Text: WARNING: MPI_Finalize: There are
   still pending messages!

On Call: MPI_Finalize From: request-reuse2.c line: 78
for MPI-Standard information
   see:/opt/marmot/marmot_icc_openmpi/share/doc/cg-wp2
.2-marmot-cvs2007070613/MPI-STANDARD/marmot_err/node73.html

13: Error global message with  Text: WARNING: all clients are pending!
Last calls (max. 10) on node 0:
 timestamp 1: MPI_Init(*argc, ***argv)
 timestamp 3: MPI_Comm_size(comm = MPI_COMM_NULL, *size)
 timestamp 5: MPI_Comm_rank(comm = MPI_COMM_NULL, *rank)
 timestamp 7: MPI_Isend(*buf, count = 1, datatype = MPI_DATATYPE_NULL,
   dest = 1,
 tag = 16, comm = MPI_COMM_NULL, *request)
 timestamp 9: MPI_Irecv(*buf, count = 1, datatype = MPI_DATATYPE_NULL,
   source =
1, tag = 17, comm = MPI_COMM_NULL, *request)
 timestamp 11: MPI_Wait(*request, *status)
```

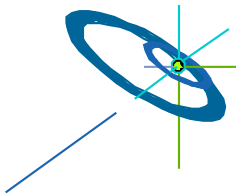


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

Example 1: request-reuse (output log – new, Open MPI)

```
Last calls (max. 10) on node 1:
 timestamp 2: MPI_Init(*argc, ***argv)

 timestamp 4: MPI_Comm_size(comm = MPI_COMM_NULL, *size)

 timestamp 6: MPI_Comm_rank(comm = MPI_COMM_NULL, *rank)

 timestamp 8: MPI_Isend(*buf, count = 1, datatype = MPI_DATATYPE_NULL,
   dest = 0,
 tag = 18, comm = MPI_COMM_NULL, *request)

 timestamp 10: MPI_Isend(*buf, count = 1, datatype = MPI_DATATYPE_NULL,
   dest = 1
, tag = 17, comm = MPI_COMM_NULL, *request)

 timestamp 12: MPI_Wait(*request, *status)

 timestamp 13: MPI_Finalize()
```

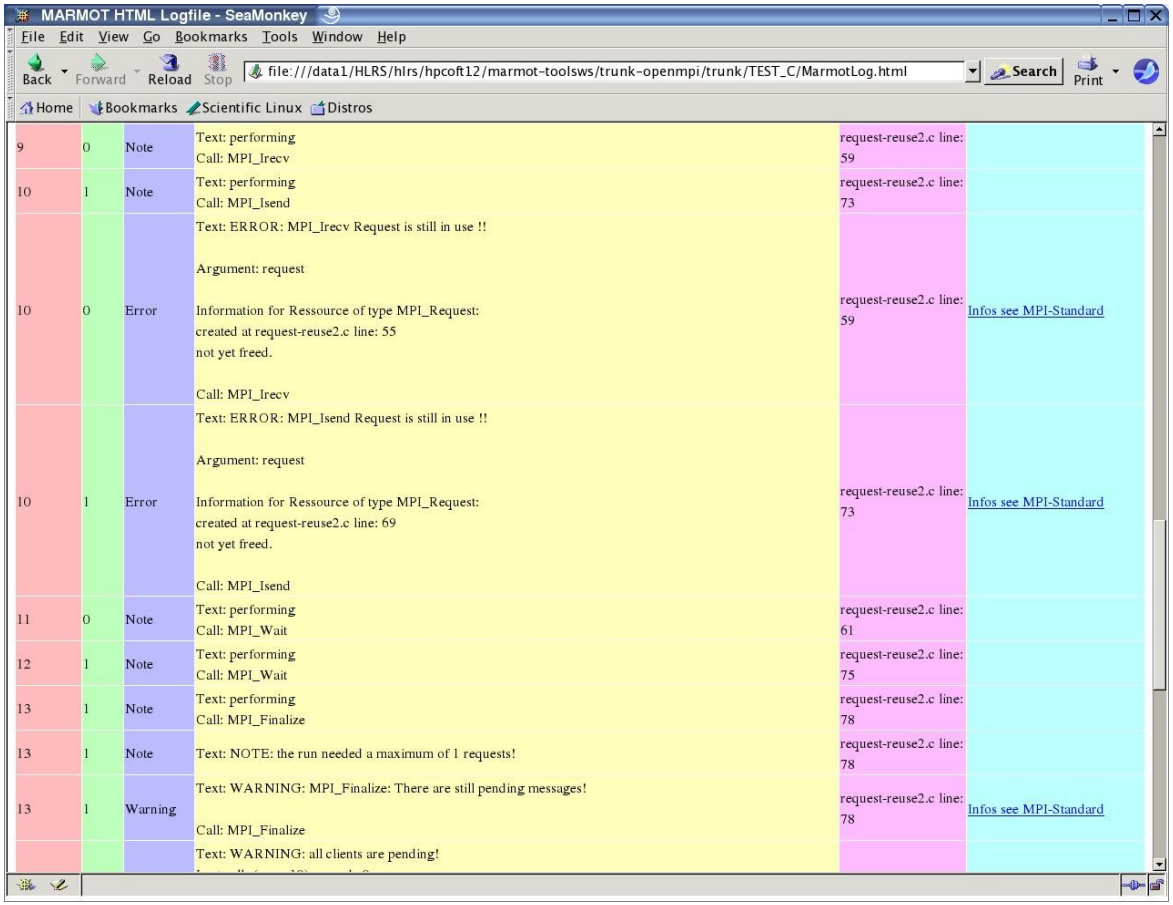


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

Example 1: request-reuse (output html – new, Open MPI)

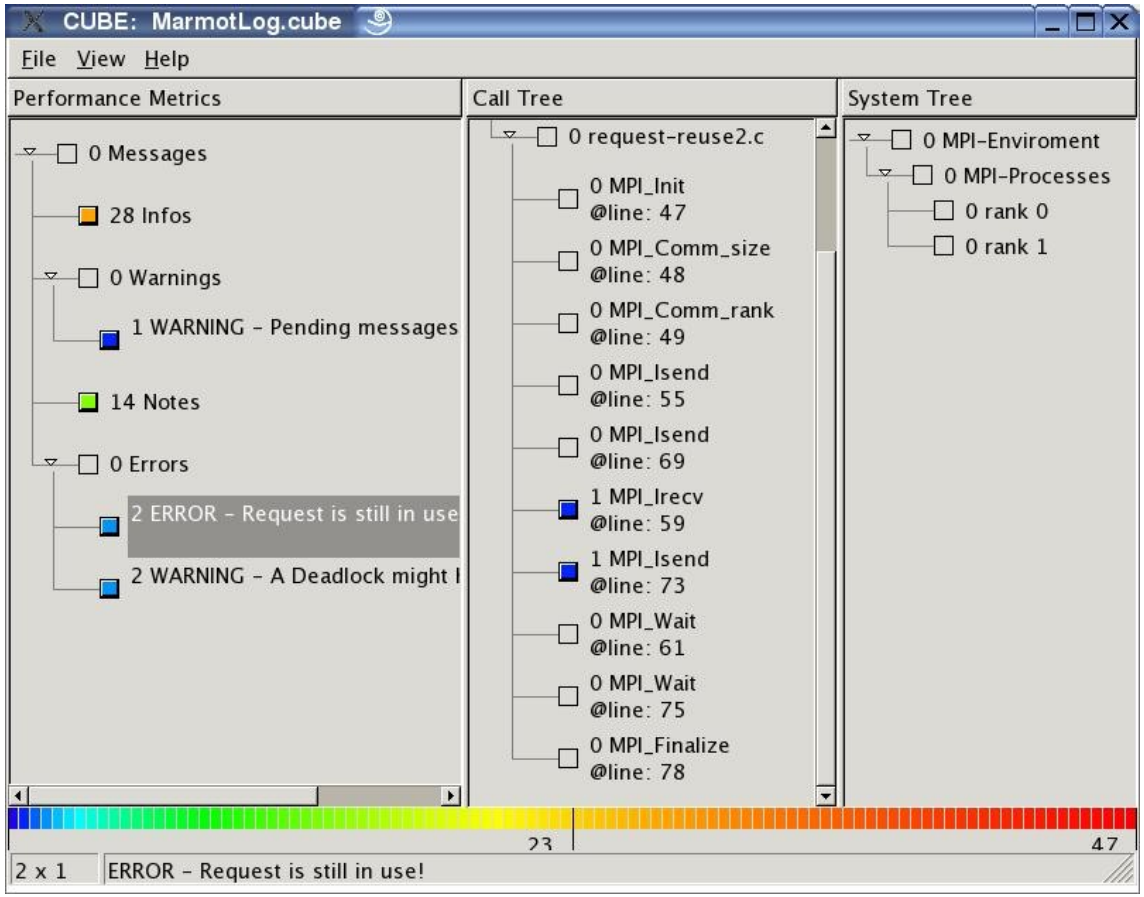
```
export MARMOT_LOGFILE_TYPE=1
```



9	0	Note	Text: performing Call: MPI_Irecv	request-reuse2.c line: 59	
10	1	Note	Text: performing Call: MPI_Isend	request-reuse2.c line: 73	
10	0	Error	Text: ERROR: MPI_Irecv Request is still in use !! Argument: request Information for Ressource of type MPI_Request: created at request-reuse2.c line: 55 not yet freed. Call: MPI_Irecv	request-reuse2.c line: 59	Infos see MPI-Standard
10	1	Error	Text: ERROR: MPI_Isend Request is still in use !! Argument: request Information for Ressource of type MPI_Request: created at request-reuse2.c line: 69 not yet freed. Call: MPI_Isend	request-reuse2.c line: 73	Infos see MPI-Standard
11	0	Note	Text: performing Call: MPI_Wait	request-reuse2.c line: 61	
12	1	Note	Text: performing Call: MPI_Wait	request-reuse2.c line: 75	
13	1	Note	Text: performing Call: MPI_Finalize	request-reuse2.c line: 78	
13	1	Note	Text: NOTE: the run needed a maximum of 1 requests!	request-reuse2.c line: 78	
13	1	Warning	Text: WARNING: MPI_Finalize: There are still pending messages! Call: MPI_Finalize	request-reuse2.c line: 78	Infos see MPI-Standard
			Text: WARNING: all clients are pending!		

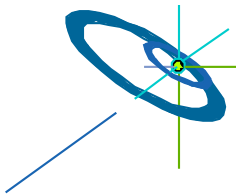
Example 1: request-reuse (output cube – new, Open MPI)

```
export MARMOT_LOGFILE_TYPE=2
```



Example: deadlock (source code)

```
/* This program produces a deadlock.
** At least 2 nodes are required to run the program.
**
** Rank 0 recv a message from Rank 1.
** Rank 1 recv a message from Rank 0.
**
** AFTERWARDS:
** Rank 0 sends a message to Rank 1.
** Rank 1 sends a message to Rank 0.
*/

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

int main( int argc, char** argv ){
  int rank  = 0;
  int size  = 0;
  int dummy = 0;
  MPI_Status status;
```

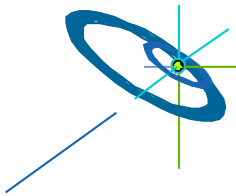


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

Example: deadlock (source code continued)

```
  MPI_Init( &argc, &argv );
  MPI_Comm_rank( MPI_COMM_WORLD, &rank );
  MPI_Comm_size( MPI_COMM_WORLD, &size );
  if( size < 2 ){
    fprintf( stderr," This program needs at least 2 PEs!\n" );
  }
  else {
    if ( rank == 0 ){
      MPI_Recv( &dummy, 1, MPI_INT, 1, 17, MPI_COMM_WORLD, &status );
      MPI_Send( &dummy, 1, MPI_INT, 1, 18, MPI_COMM_WORLD );
    }
    if( rank == 1 ){
      MPI_Recv( &dummy, 1, MPI_INT, 0, 18, MPI_COMM_WORLD, &status );
      MPI_Send( &dummy, 1, MPI_INT, 0, 17, MPI_COMM_WORLD );
    }
  }
  MPI_Finalize();
}
```

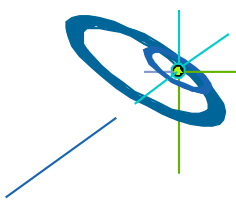


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

HLRS

Example: deadlock (output log)

```
$ mpirun -np 3 deadlock1

1 rank 0 performs MPI_Init
2 rank 1 performs MPI_Init
3 rank 0 performs MPI_Comm_rank
4 rank 1 performs MPI_Comm_rank
5 rank 0 performs MPI_Comm_size
6 rank 1 performs MPI_Comm_size
7 rank 0 performs MPI_Recv
8 rank 1 performs MPI_Recv
8 Rank 0 is pending!
8 Rank 1 is pending!
   WARNING: deadlock detected, all clients are
   pending
```

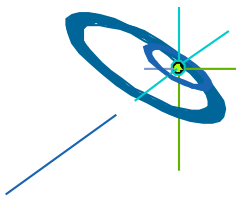


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

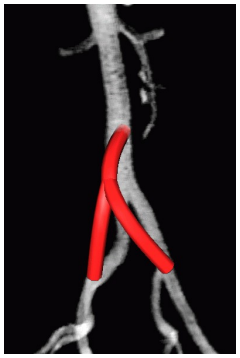
Example: deadlock (output log continued)

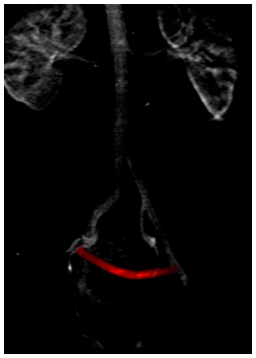
```
Last calls (max. 10) on node 0:
 timestamp = 1: MPI_Init( *argc, ***argv )
 timestamp = 3: MPI_Comm_rank( comm, *rank )
 timestamp = 5: MPI_Comm_size( comm, *size )
 timestamp = 7: MPI_Recv( *buf, count = -1,
  datatype = non-predefined datatype, source =
  -1, tag = -1, comm, *status)

Last calls (max. 10) on node 1:
 timestamp = 2: MPI_Init( *argc, ***argv )
 timestamp = 4: MPI_Comm_rank( comm, *rank )
 timestamp = 6: MPI_Comm_size( comm, *size )
 timestamp = 8: MPI_Recv( *buf, count = -1,
  datatype = non-predefined datatype, source =
  -1, tag = -1, comm, *status)
```

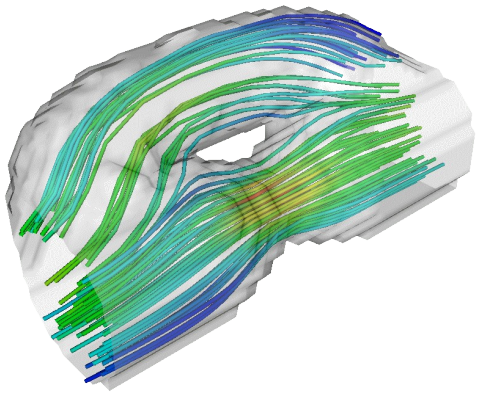


Real World Application

Example - Medical Application B_Stream







- Calculation of blood flow with Lattice-Boltzmann method
- 16 different MPI calls:
  - MPI_Init, MPI_Comm_rank, MPI_Comm_size, MPI_Pack, MPI_Bcast, MPI_Unpack, MPI_Cart_create, MPI_Cart_shift, MPI_Cart_rank, MPI_Send, MPI_Recv, MPI_Barrier, MPI_Reduce, MPI_Sendrecv, MPI_Wtime, MPI_Finalize
- We use different input files that describe the geometry of the artery: tube, tube-stenosis, bifurcation

Example: B_Stream

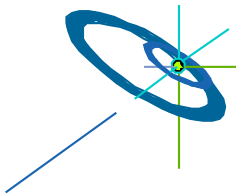
- Running the application

```
mpirun -np <np> B_Stream <Reynolds> <geometry-file>
```

  - With `10 <= Reynolds <= 500`
  - `geometry-file` = `tube` or `tube-stenosis` or `bifurcation`

- For example

```
mpirun -np 3 B_Stream 500. tube
```



ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

HLRS

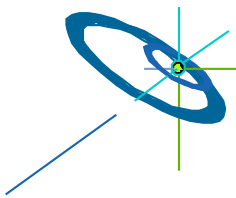
Example: B_Stream (blood flow simulation, tube)

- Tube geometry: simplest case, just a tube with about the same radius everywhere
- Running the application without/with MARMOT:

  mpirun -np 3 B_Stream 500. tube

  mpirun -np 4 B_Stream_marmot 500. tube

- Application seems to run without problems



ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

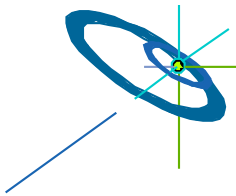
Example: B_Stream (blood flow simulation, tube-stenosis)

- Tube-stenosis geometry: just a tube with varying radius
- Without MARMOT:

  mpirun -np 3 B_Stream 500. tube-stenosis
- Application is hanging

- With MARMOT:

  mpirun -np 4 B_Stream_marmot 500. tube-stenosis
- Deadlock found



ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

HLRS

Example: B_Stream (blood flow simulation, tube-stenosis)

```
9310 rank 1 performs MPI_Sendrecv
9311 rank 2 performs MPI_Sendrecv
9312 rank 0 performs MPI_Barrier
9313 rank 1 performs MPI_Barrier
9314 rank 2 performs MPI_Barrier
9315 rank 1 performs MPI_Sendrecv
9316 rank 2 performs MPI_Sendrecv
9317 rank 0 performs MPI_Sendrecv
9318 rank 1 performs MPI_Sendrecv
9319 rank 0 performs MPI_Sendrecv
9320 rank 2 performs MPI_Sendrecv
9321 rank 0 performs MPI_Barrier
9322 rank 1 performs MPI_Barrier
9323 rank 2 performs MPI_Barrier
9324 rank 1 performs MPI_Comm_rank
9325 rank 1 performs MPI_Bcast
9326 rank 2 performs MPI_Comm_rank
9327 rank 2 performs MPI_Bcast
9328 rank 0 performs MPI_Sendrecv
```

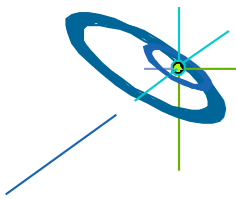
Iteration step: Calculate and exchange results with neighbors

Communicate results among all procs

WARNING: all clients are pending!

Example: B_Stream (blood flow simulation, tube-stenosis) deadlock: traceback on node 0

```
timestamp= 9304: MPI_Barrier(comm = MPI_COMM_WORLD)
timestamp= 9307: MPI_Sendrecv(*sendbuf, sendcount = 7220,
   sendtype = MPI_DOUBLE, dest = 2, sendtag = 1, *recvbuf,
   recvcount = 7220, recvtype = MPI_DOUBLE, source = 1, recvtag
   = 1, comm = self-defined communicator, *status)
timestamp= 9309: MPI_Sendrecv(*sendbuf, sendcount = 7220,
   sendtype = MPI_DOUBLE, dest = 1, sendtag = 1, *recvbuf,
   recvcount = 7220, recvtype = MPI_DOUBLE, source = 2, recvtag
   = 1, comm = self-defined communicator, *status)
timestamp= 9312: MPI_Barrier(comm = MPI_COMM_WORLD)
timestamp= 9317: MPI_Sendrecv(*sendbuf, sendcount = 7220,
   sendtype = MPI_DOUBLE, dest = 2, sendtag = 1, *recvbuf,
   recvcount = 7220, recvtype = MPI_DOUBLE, source = 1, recvtag
   = 1, comm = self-defined communicator, *status)
timestamp= 9319: MPI_Sendrecv(*sendbuf, sendcount = 7220,
   sendtype = MPI_DOUBLE, dest = 1, sendtag = 1, *recvbuf,
   recvcount = 7220, recvtype = MPI_DOUBLE, source = 2, recvtag
   = 1, comm = self-defined communicator, *status)
timestamp= 9321: MPI_Barrier(comm = MPI_COMM_WORLD)
timestamp= 9328: MPI_Sendrecv(*sendbuf, sendcount = 7220,
   sendtype = MPI_DOUBLE, dest = 2, sendtag = 1, *recvbuf,
   recvcount = 7220, recvtype = MPI_DOUBLE, source = 1, recvtag
   = 1, comm = self-defined communicator, *status)
```

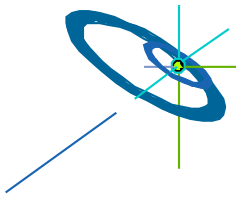


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

Example: B_Stream (blood flow simulation, tube-stenosis) deadlock: traceback on node 1

```
timestamp= 9306: MPI_Sendrecv(*sendbuf, sendcount = 7220,
   sendtype = MPI_DOUBLE, dest = 0, sendtag = 1, *recvbuf,
   recvcount = 7220, recvtype = MPI_DOUBLE, source = 2, recvtag
   = 1, comm = self-defined communicator, *status)
timestamp= 9310: MPI_Sendrecv(*sendbuf, sendcount = 7220,
   sendtype = MPI_DOUBLE, dest = 2, sendtag = 1, *recvbuf,
   recvcount = 7220, recvtype = MPI_DOUBLE, source = 0, recvtag
   = 1, comm = self-defined communicator, *status)
timestamp= 9313: MPI_Barrier(comm = MPI_COMM_WORLD)
timestamp= 9315: MPI_Sendrecv(*sendbuf, sendcount = 7220,
   sendtype = MPI_DOUBLE, dest = 0, sendtag = 1, *recvbuf,
   recvcount = 7220, recvtype = MPI_DOUBLE, source = 2, recvtag
   = 1, comm = self-defined communicator, *status)
timestamp= 9318: MPI_Sendrecv(*sendbuf, sendcount = 7220,
   sendtype = MPI_DOUBLE, dest = 2, sendtag = 1, *recvbuf,
   recvcount = 7220, recvtype = MPI_DOUBLE, source = 0, recvtag
   = 1, comm = self-defined communicator, *status)
timestamp= 9322: MPI_Barrier(comm = MPI_COMM_WORLD)
timestamp= 9324: MPI_Comm_rank(comm = MPI_COMM_WORLD, *rank)
timestamp= 9325: MPI_Bcast(*buffer, count = 3, datatype =
   MPI_DOUBLE, root = 0, comm = MPI_COMM_WORLD)
```

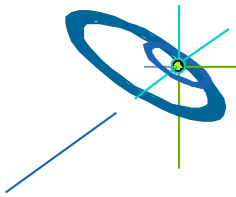


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

Example: B_Stream (blood flow simulation, tube-stenosis) deadlock: traceback on node 2

```
timestamp= 9308: MPI_Sendrecv(*sendbuf, sendcount = 7220,
   sendtype = MPI_DOUBLE, dest = 1, sendtag = 1, *recvbuf,
   recvcount = 7220, recvtype = MPI_DOUBLE, source = 0, recvtag
   = 1, comm = self-defined communicator, *status)
timestamp= 9311: MPI_Sendrecv(*sendbuf, sendcount = 7220,
   sendtype = MPI_DOUBLE, dest = 0, sendtag = 1, *recvbuf,
   recvcount = 7220, recvtype = MPI_DOUBLE, source = 1, recvtag
   = 1, comm = self-defined communicator, *status)
timestamp= 9314: MPI_Barrier(comm = MPI_COMM_WORLD)
timestamp= 9316: MPI_Sendrecv(*sendbuf, sendcount = 7220,
   sendtype = MPI_DOUBLE, dest = 1, sendtag = 1, *recvbuf,
   recvcount = 7220, recvtype = MPI_DOUBLE, source = 0, recvtag
   = 1, comm = self-defined communicator, *status)
timestamp= 9320: MPI_Sendrecv(*sendbuf, sendcount = 7220,
   sendtype = MPI_DOUBLE, dest = 0, sendtag = 1, *recvbuf,
   recvcount = 7220, recvtype = MPI_DOUBLE, source = 1, recvtag
   = 1, comm = self-defined communicator, *status)
timestamp= 9323: MPI_Barrier(comm = MPI_COMM_WORLD)
timestamp= 9326: MPI_Comm_rank(comm = MPI_COMM_WORLD, *rank)
timestamp= 9327: MPI_Bcast(*buffer, count = 3, datatype =
   MPI_DOUBLE, root = 0, comm = MPI_COMM_WORLD)
```



ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

Example: B_Stream (blood flow simulation, tube-stenosis) – Code Analysis

```
main {
    …
    num_iter = calculate_number_of_iterations();
    for (i=0; i < num_iter; i++) {
        computeBloodflow();
    }
        writeResults();
…. MPI_Bcast(…);
}
```

```
if (radius < x) num_iter = 50;
if (radius >= x) num_iter = 200;
// ERROR: it is not ensured here that all
// procs do the same number of iterations
```

```
CalculateSomething();
// exchange results with neighbors
MPI_Sendrecv(…);
```

```
// communicate results with neighbors
MPI_Bcast(…);
```

Be careful if you call functions with hidden MPI calls!

Example: B_Stream (blood flow simulation, bifurcation)

- Bifurcation geometry: forked artery
- Without MARMOT:

  mpirun -np 3 B_Stream 500. bifurcation

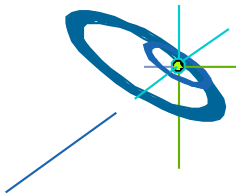
  …

  Segmentation fault

  (platform dependent if the code breaks here or not)

- With MARMOT:

  mpirun -np 4 B_Stream_marmot 500. bifurcation
- Problem found at collective call MPI_Gather

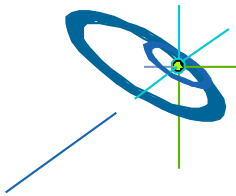


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

Example: B_Stream (blood flow simulation, bifurcation)

```
9319 rank 2 performs MPI_Sendrecv
9320 rank 1 performs MPI_Sendrecv
9321 rank 1 performs MPI_Barrier
9322 rank 2 performs MPI_Barrier
9323 rank 0 performs MPI_Barrier
9324 rank 0 performs MPI_Comm_rank
9325 rank 1 performs MPI_Comm_rank
9326 rank 2 performs MPI_Comm_rank
9327 rank 0 performs MPI_Bcast
9328 rank 1 performs MPI_Bcast
9329 rank 2 performs MPI_Bcast
9330 rank 0 performs MPI_Bcast
9331 rank 1 performs MPI_Bcast
```

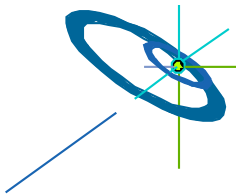


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

Example: B_Stream (blood flow simulation, bifurcation)

```
9332 rank 2 performs MPI_Bcast
9333 rank 0 performs MPI_Gather
9334 rank 1 performs MPI_Gather
9335 rank 2 performs MPI_Gather
/usr/local/mpich-1.2.5.2/ch_shmem/bin/mpirun: line
   1: 10163
Segmentation fault
   /home/rusbetti/B_Stream/bin/B_Stream_marmot
"500." "bifurcation"
   9336 rank 1 performs MPI_Sendrecv
9337 rank 2 performs MPI_Sendrecv
9338 rank 1 performs MPI_Sendrecv
WARNING: all clients are pending!
```

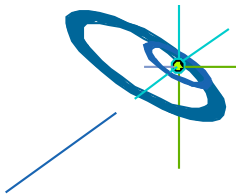


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

Example: B_Stream (blood flow simulation, bifurcation)

```
Last calls  on node 0:
   timestamp= 9327: MPI_Bcast(*buffer, count = 3, datatype =
   MPI_DOUBLE, root = 0, comm = MPI_COMM_WORLD)
   timestamp= 9330: MPI_Bcast(*buffer, count = 3, datatype =
   MPI_DOUBLE, root = 0, comm = MPI_COMM_WORLD)
   timestamp= 9333: MPI_Gather(*sendbuf, sendcount = 266409,
   sendtype = MPI_DOUBLE, *recvbuf, recvcount = 266409,
   recvtype = MPI_DOUBLE, root = 0, comm = MPI_COMM_WORLD)
Last calls on node 1:
   timestamp= 9334: MPI_Gather(*sendbuf, sendcount = 258336,
   sendtype = MPI_DOUBLE, *recvbuf, recvcount = 258336,
   recvtype = MPI_DOUBLE, root = 0, comm = MPI_COMM_WORLD)
   timestamp= 9336: MPI_Sendrecv(*sendbuf, sendcount = 13455,
   sendtype = MPI_DOUBLE, dest = 0, sendtag = 1, *recvbuf,
   recvcount = 13455, recvtype = MPI_DOUBLE, source = 2,
   recvtag = 1, comm = self-defined communicator, *status)
   timestamp= 9338: MPI_Sendrecv(*sendbuf, sendcount = 13455,
   sendtype = MPI_DOUBLE, dest = 2, sendtag = 1, *recvbuf,
   recvcount = 13455, recvtype = MPI_DOUBLE, source = 0,
   recvtag = 1, comm = self-defined communicator, *status)
```

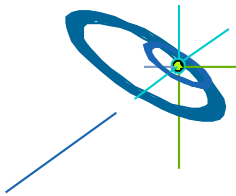


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

Example: B_Stream (blood flow simulation, bifurcation)

```
Last calls on node 2:
   timestamp= 9332: MPI_Bcast(*buffer, count = 3, datatype
   = MPI_DOUBLE, root = 0, comm = MPI_COMM_WORLD)
   timestamp= 9335: MPI_Gather(*sendbuf, sendcount =
   258336, sendtype = MPI_DOUBLE, *recvbuf, recvcount =
   258336, recvtype = MPI_DOUBLE, root = 0, comm =
   MPI_COMM_WORLD)
   timestamp= 9337: MPI_Sendrecv(*sendbuf, sendcount =
   13455, sendtype = MPI_DOUBLE, dest = 1, sendtag = 1,
   *recvbuf, recvcount = 13455, recvtype = MPI_DOUBLE,
   source = 0, recvtag = 1, comm = self-defined
   communicator, *status)
```

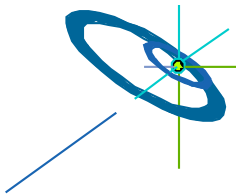


ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

HLRS

Example: BStream – summary of problems

- Different errors occur on different platforms (different compilers, different MPi implementations,…)
- Different errors occur with different input files
- Not all errors can be found with tools



ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

H L R S

Exercises

Marmot on dgrid

Initialization

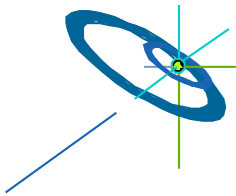
- **module load marmot/marmot_icc_openmpi compiler/intel mpi/openmpi**
- **qsub –I –V –lnodes=2:dgrid,walltime=00:30:00**
  (interactive batch environment on 2 nodes for 30 min.)
- **cd MARMOT/openmpi**

Compilation

- Compiling the application:
  marmotcc –g –o *my_prog my_prog*.**c**
  or
  marmotf77 –g –o *my_prog my_prog*.**f** or **.f90**
- **(add #include "enhancempicalls.h" to get source info)**

Execution

- **mpirun –np [n+1]** ./my_prog



ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

HLRS

Marmot on dgrid

Environment

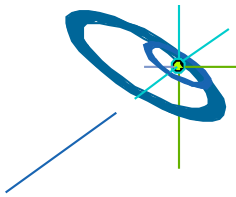
- **export MARMOT_LOGFILE_TYPE=2**

  cube output – run **cube MarmotLog.cube** to visualize your output
- **export MARMOT_LOGFILE_TYPE=1**

  html output
- **export MARMOT_LOGFILE_TYPE=0**

  txt output (default)

- (there's more environment variables for configuring Marmot...)



ZIH Zentrum für Informationsdienste und Hochleistungsrechnen

HLRS