Archer: Debugging with compiler-based feedback

RWTH AACHEN UNIVERSITY

Examples of Undefined Behavior as Defined in C/C++

- Read of **uninitialized variable**
- Out-of-bounds access to heap, stack and globals
  - **Use after free / return / scope**
- Bit shifts beyond type bounds
- Dereferencing a **NULL pointer**
- Signed integer overflow
- Conversion to floating point overflowing the destination
- Accessing a different union member than most recently written
- **Data race**
- More: see C/C++ standard

Undefined Behavior: What could go wrong?

- UB allows compilers any behavior

➢ Possible optimization: assume absence of UB

➢ Unexpected results

➢ Avoid UB in any case!

```
int check_for_32(int a) {
  int res;
  if (a == 32)
    res = a;
  return res;
}
```

clang 17:

```
check_for_32(int):
        mov     eax, edi  # return a
        ret
```

gcc 13:

```
check_for_32(int):
        mov     eax, 32   # return 32
        ret
```

- Program is only well-defined, if a=32!

➢ Both compilers generate valid code!

Sanitizers

- LLVM offers multiple sanitizers
  - MemorySanitizer
  - AddressSanitizer
  - UBSanitizer
  - ThreadSanitizer

- Activate sanitizers with `clang/icx -fsanitize=<name>`

Memory Sanitizer

Typical slowdown: 3x

- Detects read of uninitialized memory

```
$ icx -fsanitize=memory -fno-omit-frame-pointer -g -O2 2_msan/msan.c
```

```
int check_for_32(int *a) {
  int res;
  if (*a == 32)
    res = *a;
  return res;
}

int main(){
  int a;
  printf("check_for_32(%i) = %i\n", a,
check_for_32(&a) );
}
```

```
==145056==WARNING: MemorySanitizer: use-of-uninitialized-value
    #0 0x55dbd1ab8571 in check_for_32 msan.c:14:7
    #1 0x55dbd1ab8656 in main msan.c:23:40
    #2 0x7fa9ccacbd8f in __libc_start_call_main
csu/../sysdeps/nptl/libc_start_call_main.h:58:16
    #3 0x7fa9ccacbe3f in __libc_start_main csu/../csu/libc-
start.c:392:3
    #4 0x55dbd1a322a4 in _start (a.out+0x1e2a4) (BuildId:
763eb80b676294a02a65890f5181a8a697e4b2a5)

SUMMARY: MemorySanitizer: use-of-uninitialized-value msan.c:14:7
in check_for_32
Exiting
```

AddressSanitizer

Typical slowdown: 2x

```
$ icpx -fsanitize=address -fno-omit-frame-pointer -g -O2 3_asan/asan.cc
```

```
int main(int argc, char **argv)
{
  int *array = new int[100];
  delete [] array;
  return array[argc];  // BOOM
}
```

```
==176065==ERROR: AddressSanitizer: heap-use-after-free on address
0x614000000044 at pc 0x563c6909f597 bp 0x7ffe9aa80610 sp 0x7ffe9aa80608
SUMMARY: AddressSanitizer: heap-use-after-free asan.cc:4:10 in main
Shadow bytes around the buggy address:
  0x0c287fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c287fff8000: fa fa fa fa fa fa fa fa[fd]fd fd fd fd fd fd fd
  0x0c287fff8010: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c287fff8020: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c287fff8030: fd fd fd fd fd fd fd fd fd fd fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Heap left redzone:       fa
  Freed heap region:       fd
```

UB Sanitizer

Typical slowdown: 2x

```
$ icx -fsanitize=undefined 4_ubsan/ubsan.c
```

```
int main(int argc, char **argv) {
 int k = 0x7ffffffe;
 k += argc;
 return 0;
}
```

```
$ ./a.out
$ ./a.out 5
ubsan.c:3:5: runtime error: signed integer
overflow: 2147483646 + 2 cannot be
represented in type 'int'
SUMMARY: UndefinedBehaviorSanitizer:
undefined-behavior ubsan.c:3:5 in
```

Thread Sanitizer

C++

- The execution of a program contains a **data race** if it contains **two** potentially **concurrent** conflicting actions, at least one of which is **not atomic**, and **neither happens before** the other, [...]. Any such data race results in **undefined behavior**.

OpenMP

- **Multiple threads** access the same memory **unordered**, at least one thread writes. If a **data race** occurs then the result of the program is **unspecified**.

- There is **no benign** data race in C/C++. It is always **UB**!

Data Race

```
program hello
#ifdef _OPENMP
       use omp_lib
#endif
       implicit none
       integer:: nthreads, threadid
!$omp parallel
#ifdef _OPENMP
       nthreads = OMP_GET_NUM_THREADS()
       threadid = OMP_GET_THREAD_NUM()
       if(threadid.eq.0) then
           write(*,*) "Open-MP version with threads = ", nthreads
       endif
#else
       write(*,*) "Serial version "
#endif
!$omp end parallel
end program
```

Typical slowdown: 5-15x

Thread Sanitizer

- Detects data race
- Detects lock order inversion (potential deadlocks, but no actual deadlock)
- Many supported OS/architectures:
  - Android aarch64, x86_64
  - Darwin arm64, x86_64
  - FreeBSD
  - Linux aarch64, x86_64, powerpc64, powerpc64le
  - NetBSD
- New and vectorized runtime library introduced with LLVM 15 (halved space and time overhead)

Thread Sanitizer Example

```
$ icx -fsanitize=thread -g -O2 5_tsan/tsan.c
```

```
include <pthread.h>
int Global;
void *Thread1(void *x) {
  Global = 42;
  return x;
}
int main() {
  pthread_t t;
  pthread_create(&t, NULL, Thread1, NULL);
  Global = 43;
  pthread_join(t, NULL);
  return Global;
}
```

```
==================
WARNING: ThreadSanitizer: data race (pid=134056)
  Write of size 4 at 0x564d0188c258 by main thread:
    #0 main tsan.c:10:9 (a.out+0xe78e2)

  Previous write of size 4 at 0x564d0188c258 by thread T1:
    #0 Thread1 tsan.c:4:9 (a.out+0xe7899)

  Location is global 'Global' of size 4 at 0x564d0188c258 (a.out+0x1493258)

  Thread T1 (tid=134058, finished) created by main thread at:
    #0 pthread_create tsan_interceptors_posix.cpp:1022:234 (a.out+0x60a8d)
    #1 main tsan.c:9:2 (a.out+0xe78d3)

SUMMARY: ThreadSanitizer: data race tsan.c:10:9 in main
==================
ThreadSanitizer: reported 1 warnings
```

ThreadSanitizer + Archer = OpenMP Data Race Detection

- Developed in cooperation with UoUtah, LLNL since 2014
- Injects OpenMP synchronization (and concurrency) into Tsan
- Avoids false positive reports
- Shipped with LLVM since 10.0
  - oneAPI DPC++/C++ Compiler since version 2024.0.
- Covers latest OpenMP semantics (when using latest llvm ;)
- Verify that Archer is active: `ARCHER_OPTIONS=verbose=1`
  - Known issue with Ubuntu: the packaging bricks Archer
    - `export OMP_TOOL_LIBRARIES=/usr/lib/llvm-*/lib/libarcher.so`
  - Known issue of TSan with certain HPC applications: increased overhead for 5+ threads

Archer: Runtime Options

- export as `ARCHER_OPTIONS=`
  - `verbose={0|1}` for information about archer status
  - `enable={0|1}` for disabling archer
  - `ignore_serial={0|1}` to disable analysis for serial code
  - `all_memory={0|1}` to enable analysis of `all_memory` dependences

- Strongly encouraged: `TSAN_OPTIONS=ignore_noninstrumented_modules=1`

Using Sanitizers with Fortran code

- Latest versions of the Intel Fortran compiler (ifx) support sanitizers
  - `$ ifx -fopenmp -fsanitize=thread test.f90`
- GCC supports most sanitizers (other than MSan)
- Compile Fortran codes with `gfortran` + sanitizer flags
- For Archer support, make sure to link the LLVM OpenMP runtime, not GNU runtime:
  - `$ gfortran -lomp -fopenmp -fsanitize=thread test.f90`
  - `$ gfortran -fopenmp -c -fsanitize=thread test.f90`

    ```
    $ clang -fopenmp -fsanitize=thread --gcc-install-dir=<path-to-gfortran> -lgfortran test.o
    ```
  - The latter links the LLVM TSan runtime which has significantly lower runtime overhead

Hands-on: Archer

- Get the Hands-on code: /gpfs/scratch/nct_362/exercises/ARCHER/prime_omp.c

- Load intel compilers >= 2024 (or LLVM > 10)

```
$ module purge
$ module load intel/2025.2
```

- Compile with LLVM based compiler (e.g. `icx`) and run

```
$ icx -g -qopenmp -O2 -lm prime_omp.c
$ OMP_NUM_THREADS=2 ./a.out
```

- Compile and run with Tsan

```
$ icx -g -fsanitize=thread -fopenmp -O2 -lm prime_omp.c
$ OMP_NUM_THREADS=2 ARCHER_OPTIONS=enable=1 OMP_TOOL_LIBRARIES=libarcher.so ./a.out
```

MPI + OpenMP Correctness Checking with MUST

RWTH AACHEN UNIVERSITY

How many issues 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!

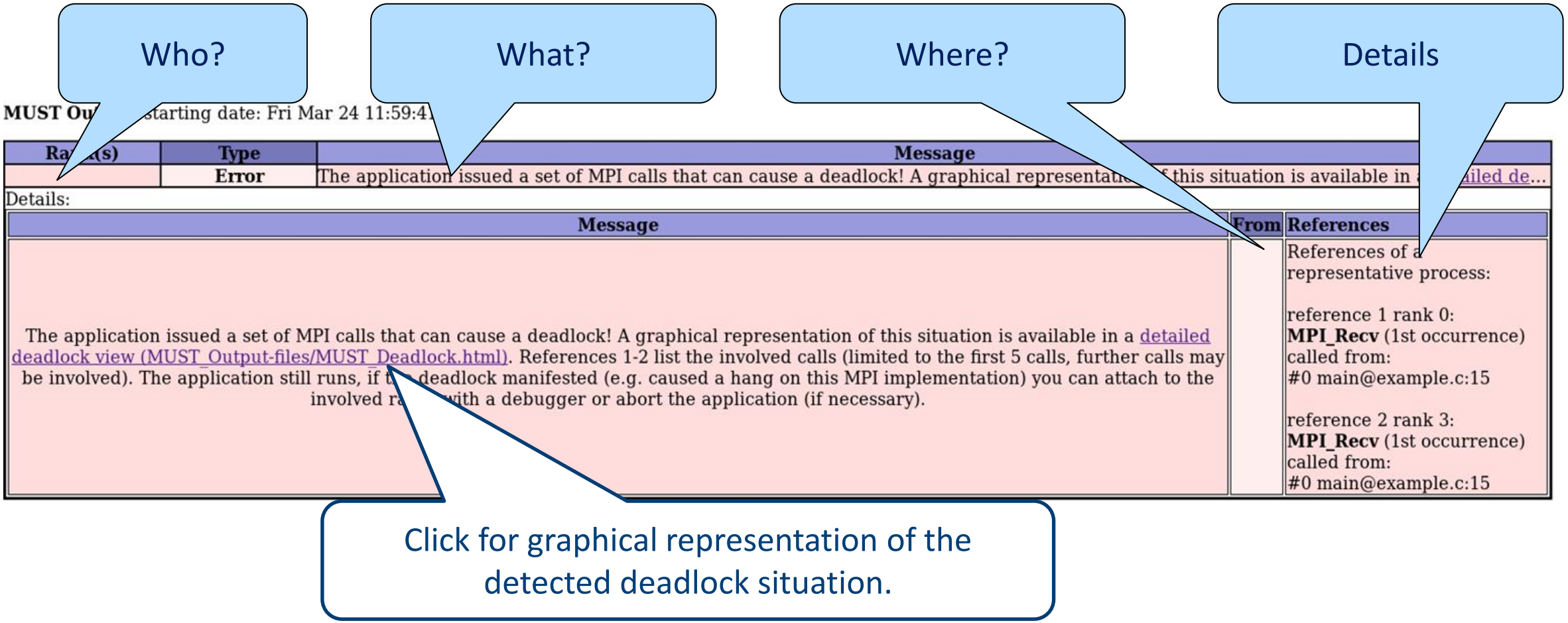
Motivation

- MPI programming is error prone
- Portability errors (just on some systems, just for some runs)
- Bugs may manifest as:
  - Crash
  - Application hanging
  - Finishes

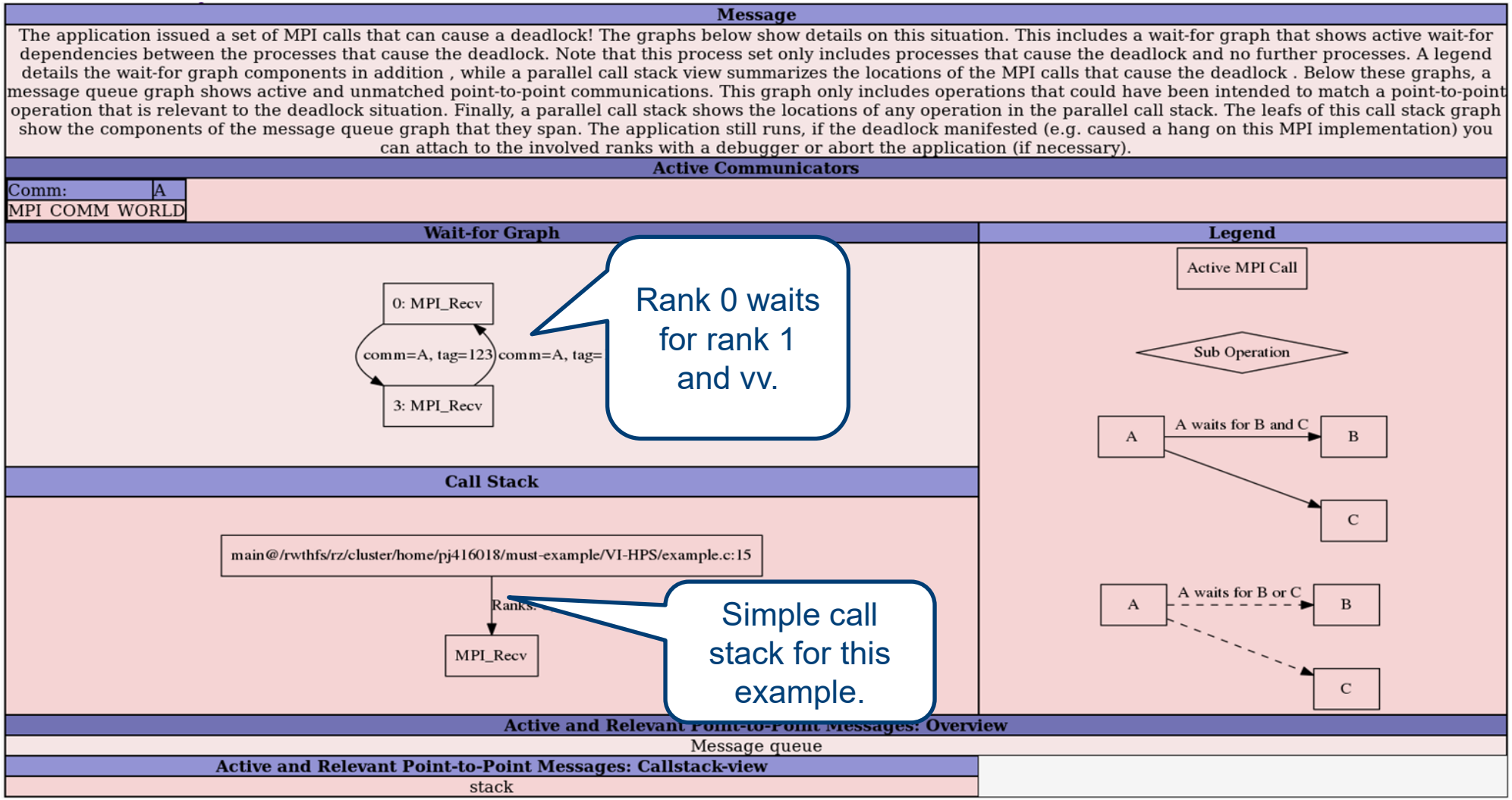
- Questions:
  - Why crashing/hanging?
  - Is my result correct?
  - Will my code also give correct results on another system?

- Tools help to pin-point these bugs

MUST detects deadlocks



Visualization of deadlock situation



MUST detects errors in transfer buffer sizes / types

Rank(s)	Type	
2(28793)	**Error**	A receive operation uses a (datatype…y the send it matches! The first element of the send…

Details:

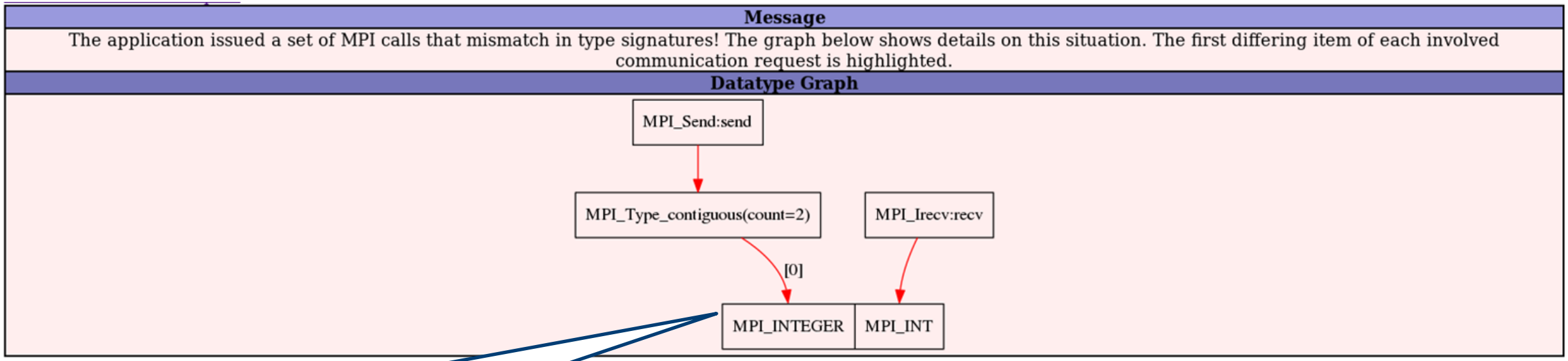
Message	From	References
A receive operation uses a (datatype,count) pair that can not hold the data transfered by the send it matches! The first element of the send that did not fit into the receive operation is at (contiguous)[0](MPI_INTEGER) in the send type (consult the MUST manual for a detailed 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)	Representative location: **MPI_Send** (1st occurrence) called from: #0 main@example-fix1.c:18	References of a representative process: reference 1 rank 2: **MPI_Send** (1st occurrence) called from: #0 main@example-fix1.c:18 reference 2 rank 1: **MPI_Irecv** (1st occurrence) called from: #0 main@example-fix1.c:16 reference 3 rank 2: **MPI_Type_contiguous** (1st occurrence) called from: #0 main@example-fix1.c:13

1(28792)	**Error**	A receive operation uses a (datatype,count) pair that can not hold the data transfered by the send it matches! The first element of the send…
0-3	**Error**	Argument 3 (datatype) is not commited for transfer, call MPI_Type_commit before using the type for transfer!(Information on datatypeData…
2(28793)	**Error**	The memory regions to be transfered by this send operation overlap with regions spanned by a pending non-blocking receive operation!(In…
1(28792)	**Error**	The memory regions to be transfered by this send operation overlap…receive operation!(In…
3(28795)	**Error**	The memory regions to be transfered by this send…receive operation!(In…
3(28795)	**Error**	A receive operation uses a (datatype,count) pair that can not…t element of the send…
0(28794)	**Error**	The memory regions to be transfered by this send operation overla…receive operation!(In…
0(28794)	**Error**	A receive operation uses a (datatype,count) pair that can not hold t…t element of the send…

Size of sent message larger than receive buffer

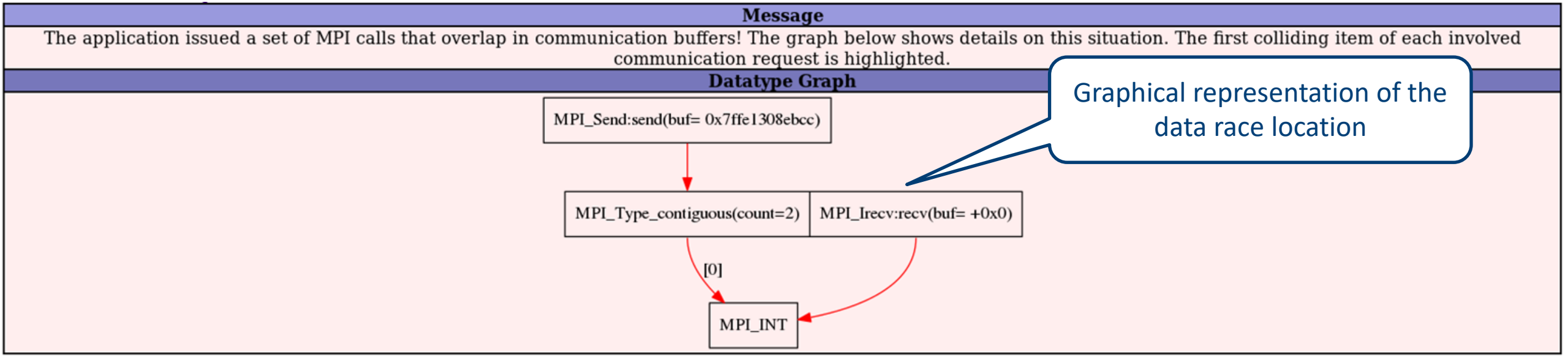
All detected errors are collapsed for overview - click to expand

MUST detects errors in handling datatypes



Graphical representation of the type mismatch

Graphical representation of the race condition



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, commited 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 q lity 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 refere ce 1	Representative l n: **MPI_Irecv** (1st occurr called from: #0 main@example-fix4.c:17	References of a representative process: rence 1 rank 1: **MPI_Irecv** (1st ce) called from: #0 mple-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

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

MUST – Basic Usage

- Apply MUST as an mpiexec wrapper, that's it:

```
% mpicc source.c -o exe
% $MPIRUN -n 4 ./exe
```

```
% mpicc -g source.c -o exe
% mustrun --must:mpiexec $MPIRUN -n 4 ./exe
```

or simply

```
% mustrun -n 4 ./exe
```

- After run: inspect "MUST_Output.html"
- "mustrun" (default config.) uses an extra process:
  - I.e.: "mustrun -np 4 …" will use 5 processes
  - Allocate the extra resource in batch jobs!
  - Default configuration tolerates application crash; BUT is slower (details later)

Advanced Usage

MUST - At Scale (highly recommended for >10 processes)

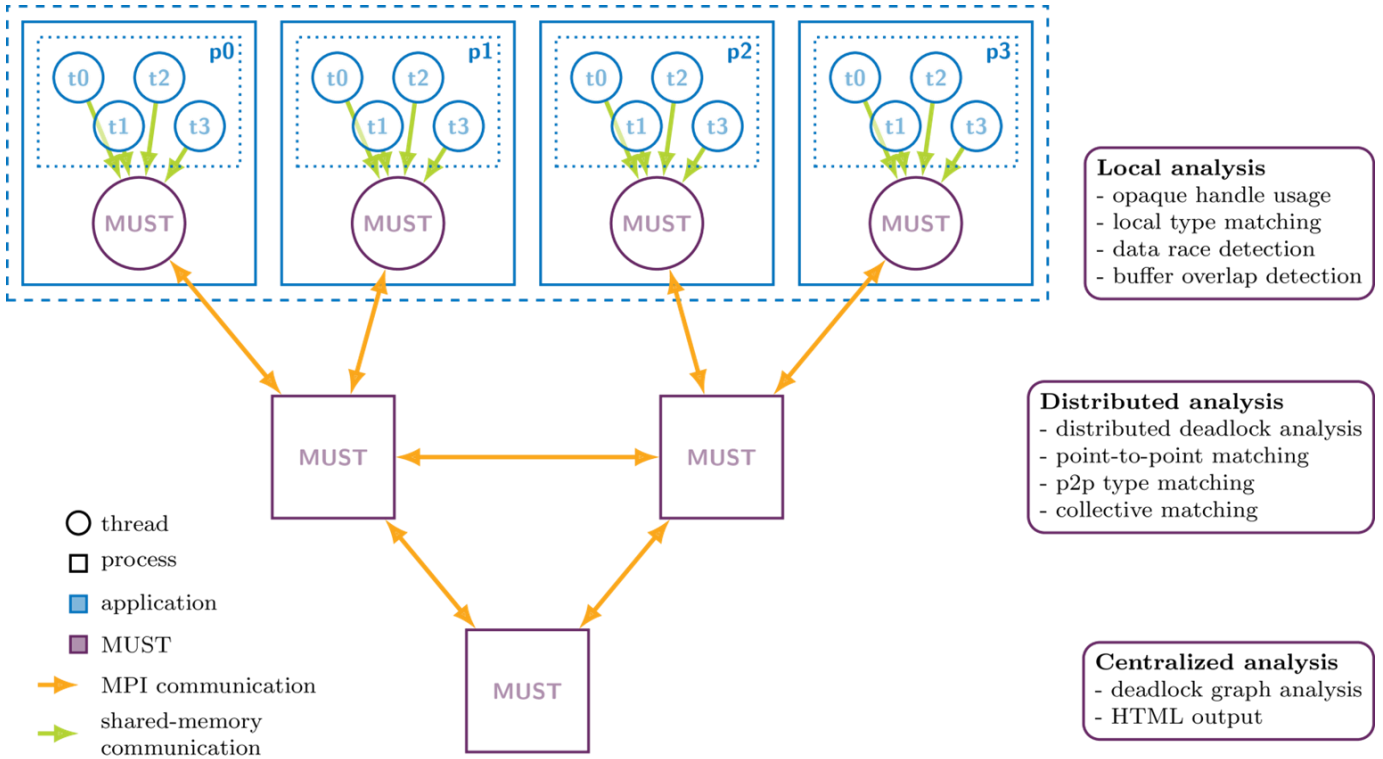
- Provide a branching factor (fan-in) for the tree infrastructure:

```
% mustrun -n 40 ./exe \
  --must:fanin 8
```

- Get info about the number of processes:

```
% mustrun -n 40 ./exe \
  --must:fanin 8 --must:info
```

- This will give you the number of processes needed with tool attached



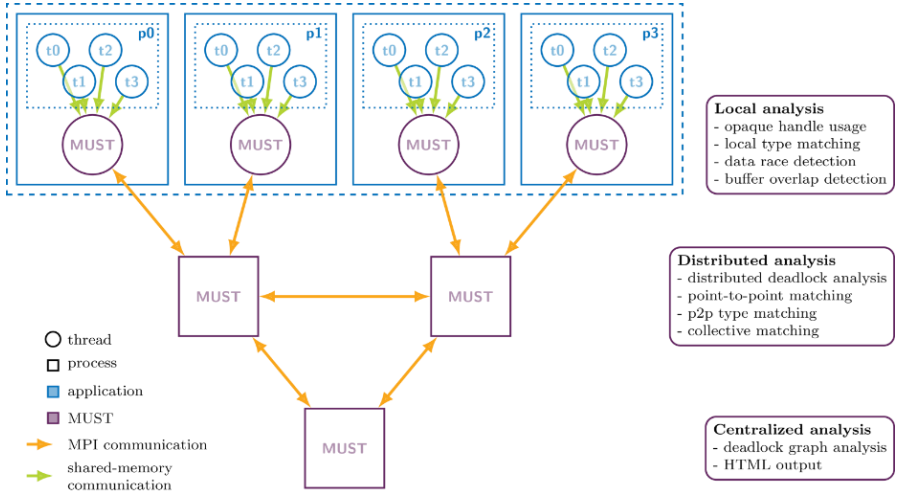
MUST – Execution Modes

	Application might crash	Application never crashes
Centralized analysis		`--must:nocrash`
	▪ 1 extra process ▪ Blocking communication	▪ 1 extra process ▪ Non-blocking communication
Distributed analysis	`--must:nodesize 8`	`--must:fanin 8`
	▪ 1 extra process per 7 application processes + tree ▪ Nodesize must be divisor of ranks sharing memory	▪ 1 extra process per 8 app processes + tree

MUST - Multithreading Support

- By default, MUST supports `MPI_THREAD_FUNNELED`
- For higher threading levels:

```
% mustrun -n 40 ./exe --must:hybrid
```



- This will raise the required level to `MPI_THREAD_MULTIPLE`!
- Some MPI might need env like: `MPICH_MAX_THREAD_SAFETY=multiple`
- Get info about the resources needed:

```
% mustrun -n 40 ./exe --must:hybrid --must:info
```

  - This will give you the number of processes needed with tool attached

Getting stacktraces

- We use Backward-cpp as an external lib for stacktraces
- Collecting stack traces can be costly. Select with
  `--must:stacktrace [backward|addr2line|none]`
- Supposed your application has no faults you won't need stacktraces ☺

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

From
Representative location: **MPI_Init_thread** (1st occurrence) called from: #0 MAIN__@bt.f:90 #1 main@bt.f:319
Representative location: **MPI_Comm_split** (1st occurrence) called from: #0 MAIN__@bt.f:90 #1 main@bt.f:319

MUST – Filter file

- Use filter files to selectively exclude error/warning messages (avoid cluttered output)
- Format: `messageType:MUST_MESSAGE_TYPE:source`
  - `MUST_MESSAGE_TYPE`: kind of message to ignore (e.g. `MUST_WARNING_COMM_NULL`)
  - `source`: specific file (`filename.c`), specific function (`function_name`) or all sources (`*`)

- Example: Ignore `NULL` comm. warnings originating from `main.c` (needs stacktraces)

```
messageType:MUST_WARNING_COMM_NULL:src:main.c
```

- Example: Ignore all data type leak errors

```
messageType:MUST_ERROR_LEAK_DATATYPE:*
```

- Define and use a filter file:
  - `--must:filter-file <path-to-filter-file>`

MUST – more options

- Print help:
  - `--must:help`
- Select output format:
  - `--must:output {html|json|stdout}`
- Use with ddt:
  - Record error message information:
    - `--must:capture`
  - Replay under control of ddt:
    - `--must:reproduce --must:ddt`

Using MUST + Archer

- Compile the MPI(+OpenMP) application just like described for Archer
- Using clang to compile with OpenMPI/IntelMPI/MPICH:
  - `export OMPI_CC=clang; export I_MPI_CC=clang; export MPICH_CC=clang;`
  - `export OMPI_CXX/I_MPI_CXX/MPICH_CXX=clang++;`
- Run MUST with TSan support:
  - `--must:tsan`
  - Feeds all MPI buffer accesses to TSan

- For integration of TSan output into the MUST report, a helper-library must be linked into the application:
  - `-Wl,--whole-archive ${MUST_ROOT}/lib/libonReportLoader.a -Wl,--no-whole-archive`

TSan Output in MUST report

Rank(s)	Type	Message
0-7	MUST_WARNING_DATARACE	Data race between a read of size 8 at .omp_outlined._debug__.53@1 and a previous write of size 8 at .omp_outlined._debu…

Details:

Message	From	References
Data race between a read of size 8 at .omp_outlined._debug__.53@1 and a previous write of size 8 at .omp_outlined._debug__.53@2.	Representative location: **.omp_outlined._debug__.53** (0th occurrence) called from: #0 .omp_outlined._debug__.53@lulesh.cc:2258 #1 .omp_outlined..54@lulesh.cc:2240 #2 __kmp_invoke_microtask@libomp.so:0xbad72 #3 EvalEOSForElems(Domain&, double*, int, int*, int)@lulesh.cc:2240 #4 ApplyMaterialPropertiesForElems(Domain&)@lulesh.cc:2401 #5 LagrangeElements(Domain&, int)@lulesh.cc:2439 #6 LagrangeLeapFrog(Domain&)@lulesh.cc:2617 #7 main@lulesh.cc:2748	References of a representative process: reference 1 rank 1: **.omp_outlined._debug__.53** (0th occurrence) called from: #0 .omp_outlined._debug__.53@lulesh.cc:2258 #1 .omp_outlined..54@lulesh.cc:2240 #2 __kmp_invoke_microtask@libomp.so:0xbad72 #3 EvalEOSForElems(Domain&, double*, int, int*, int)@lulesh.cc:2240 #4 ApplyMaterialPropertiesForElems(Domain&)@lulesh.cc:2401 #5 LagrangeElements(Domain&, int)@lulesh.cc:2439 #6 LagrangeLeapFrog(Domain&)@lulesh.cc:2617 #7 main@lulesh.cc:2748 reference 2 rank 1: **.omp_outlined._debug__.53** (0th occurrence) called from: #0 .omp_outlined._debug__.53@lulesh.cc:2246 #1 .omp_outlined..54@lulesh.cc:2240 #2 __kmp_invoke_microtask@libomp.so:0xbad72 #3 EvalEOSForElems(Domain&, double*, int, int*, int)@lulesh.cc:2240 #4 ApplyMaterialPropertiesForElems(Domain&)@lulesh.cc:2401 #5 LagrangeElements(Domain&, int)@lulesh.cc:2439 #6 LagrangeLeapFrog(Domain&)@lulesh.cc:2617 #7 main@lulesh.cc:2748 #8 main@lulesh.cc:2715

Hands-on: MUST

- Get the Hands-on code from /gpfs/scratch/nct_362/exercises/MUST/example.c
- Load MUST module from /gpfs/scratch/nct_362/RWTH/modules
```
$ module use /gpfs/scratch/nct_362/RWTH/modules
$ module load MUST/1.11.3rc2-oneapi-2023.2
```
- Compile the code **with debug symbols** with the loaded runtime
```
$ mpicc -g example.c
```
- Run the code with MUST (remember that MUST requires an additional process)
```
$ mustrun -np 4 ./a.out
```
- Download the generated html file(s) and inspect with your browser
  - MUST_Output.html
  - MUST_Output_files (for visualizations)