Runtime Correctness Checking

Joachim Protze, RWTH Aachen University

EU H2020 Centre of Excellence (CoE)

1 December 2018 – 30 November 2021

Grant Agreement No 824080

Runtime Correctness Checking

- Tools to detect correctness issues in parallel applications
- Can only detect issues observable during the execution
  - Static analysis tool can detect issues in complete code base
- Use representative input data
- Optimized to yield as few false positive reports as possible

MUST

Scalable correctness checking for MPI

Typical MPI errors

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

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

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

    MPI_Datatype type;
    MPI_Type_contiguous (2, MPI_INTEGER, &type);

    MPI_Recv (buf, 2, MPI_INT, size - rank, 123, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

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

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

    return 0;
}
```

No MPI_Init before first MPI-call
Fortran type in C
Recv-recv deadlock
Rank0: src=size (out of range)
Type not committed before use
Type not freed before end of main
Send 4 int, recv 2 int: truncation
No MPI_Finalize before end of main

Note: MUST needs MPI_Init and MPI_Finalize to detect start and end of the MPI application!

Running MUST with the example

```
~/must-example $ mpigcc -g example.c
~/must-example $ mustrun --must:mpiexec mpiexec -hosts localhost -n 2 a.out
[MUST] MUST configuration ... centralized checks with fall-back application crash handling
(very slow)
[MUST] Information: overwritting old intermediate data in directory "~/must-example/must_temp"!
[MUST] Using prebuilt infrastructure at ~/MUST/modules/mode1-layer2
[MUST] Search for linked P^nMPI ... not found ... using LD_PRELOAD to load P^nMPI ... success
[MUST] Executing application:
============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.
============================
^C
[MUST] Execution finished, inspect "~/must-example/MUST_Output.html"!
~/must-example $
```

Press Ctrl + C

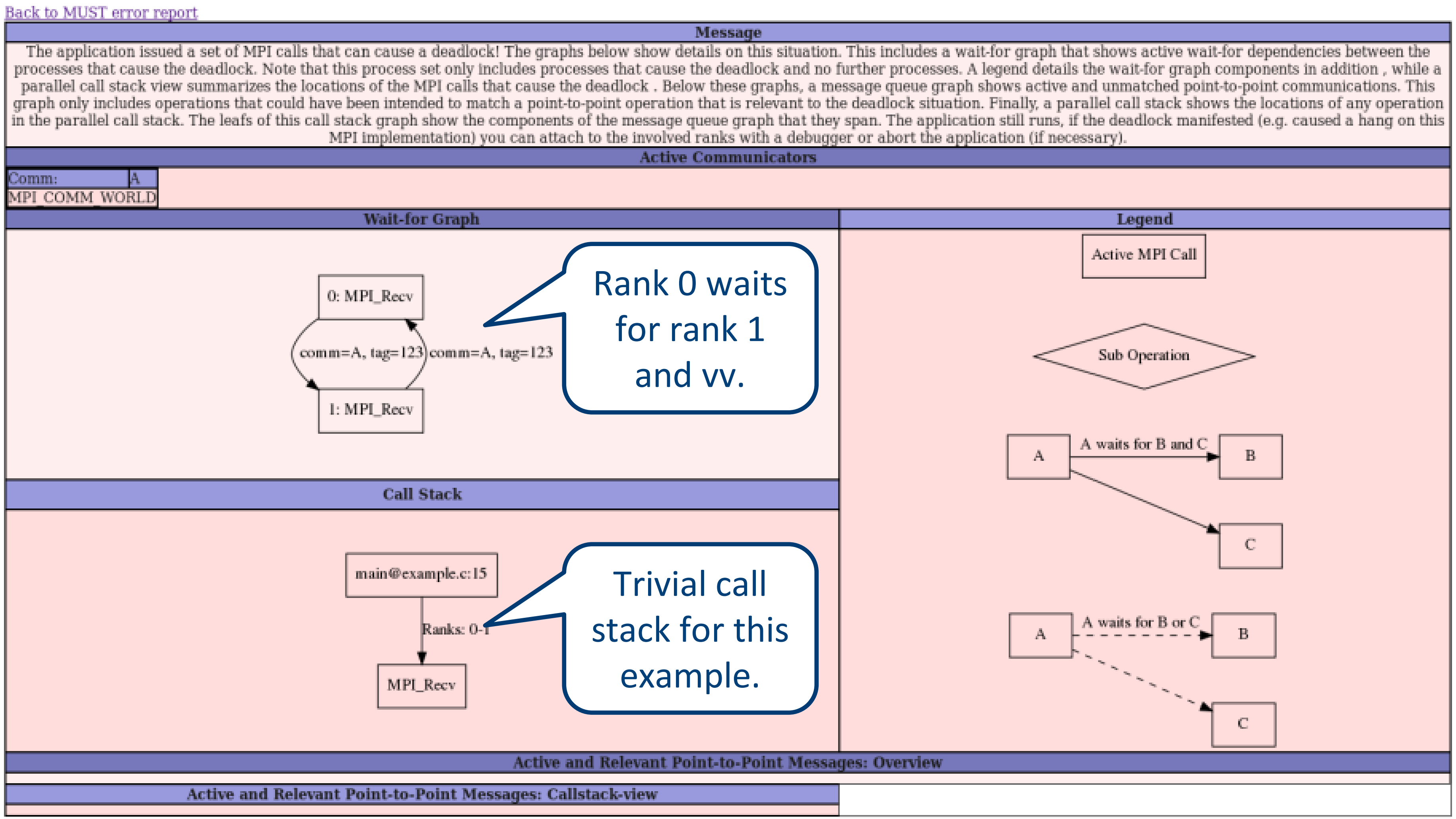
MUST Output: Html-file

Who? What? Where? Details

Rank(s)	Type	Message	From	Reference
2	**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 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, commited at reference 4, based on the following type(s): { MPI_INTEGER}Typemap = {(MPI_INTEGER, 0), (MPI_INTEGER, 4)}) (Information on receive of count 2 with type:MPI_INT)	Representative location: **MPI_Send** (1st occurrence) called from: #0 main@example-fix1.c:19	References of a representative process: reference 1 rank 2: **MPI_Send** (1st occurrence) called from: #0 main@example-fix1.c:19 reference 2 rank 1: **MPI_Irecv** (1st occurrence) called from: #0 main@example-fix1.c:17 reference 3 rank 2: **MPI_Type_contiguous** (1st occurrence) called from: #0 main@example-fix1.c:13 reference 4 rank 2: **MPI_Type_commit** (1st occurrence) called from: #0 main@example-fix1.c:14

In case of an available link:
Click for graphical representation of the detected deadlock situation or buffer-related issues.

Graphical representation of deadlocks



Best case: no error is detected

MUST Outputfile

file:/// /example/MUST_Output.html

MUST Output, starting date: Thu Nov 28 13:56:03 2013.

Rank(s)	Type	Message	From	References
	Information	MUST detected no MPI usag[illegible] nor any suspicious behavior during this application run.		

MUST has completed successfully, end date: Thu Nov 28 13:56:0[illegible] 2013.

No further error detected

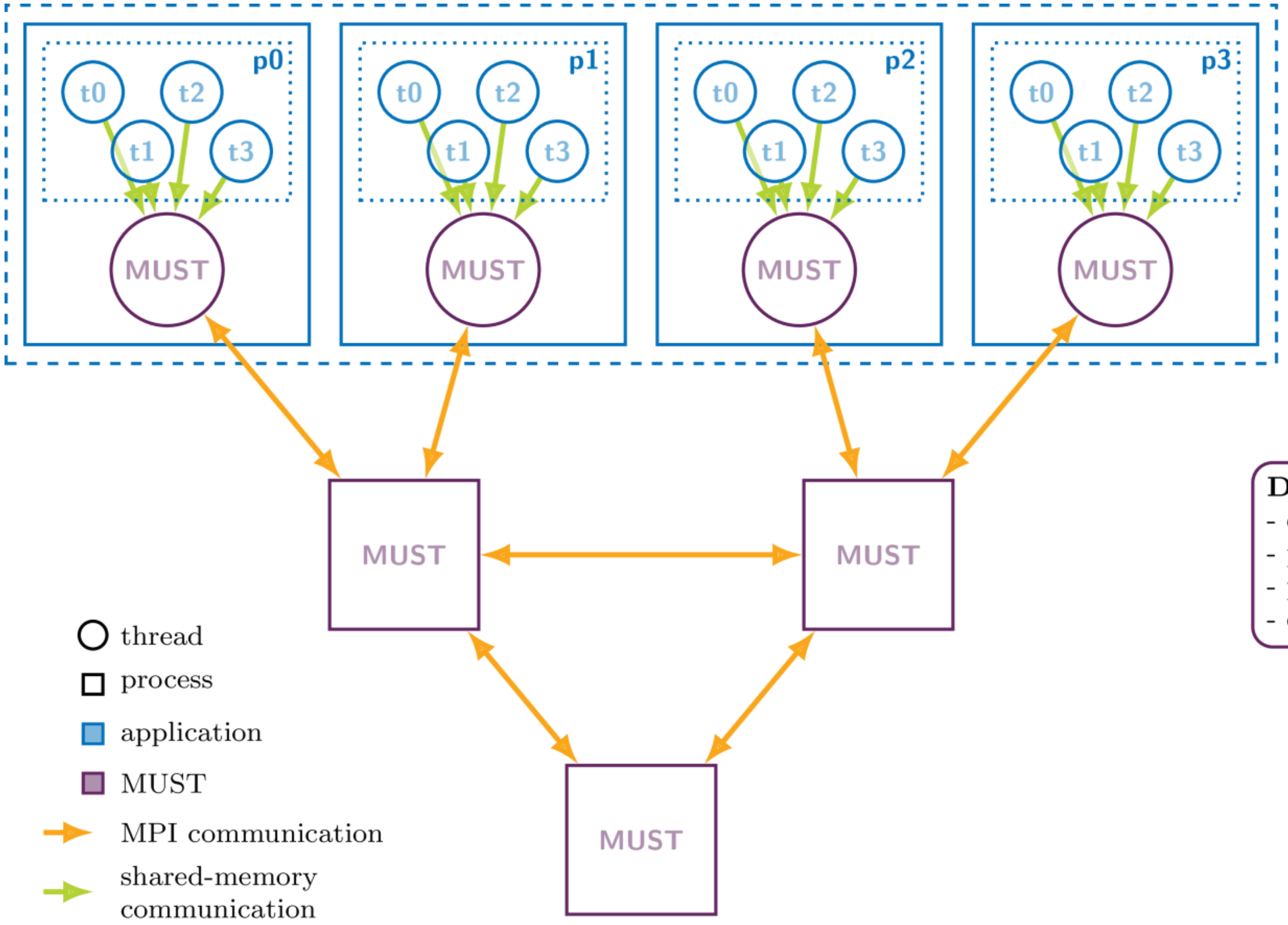
Hopefully this message applies to many applications

MUST Usage

1. Compile and link application as usual
   - Link against the shared version of the MPI lib (Usually default)
   - Add debugging flag for source line information
2. Replace "mpiexec" with "mustrun"
   - E.g.: mustrun -np 4 myApp.exe input.txt output.txt
   - Or: mustrun --must:mpiexec srun -np 4 myApp.exe input.txt output.txt
3. Inspect "MUST_Output.html" in run directory
   - "MUST_Output/MUST_Deadlock.dot" exists in case of deadlock
   - Visualize with: dot -Tps MUST_Deadlock.dot -o deadlock.ps

- The mustrun script will use an extra process for non-local checks (Invisible to application)
- I.e.: "mustrun -np 4 …" will issue a "mpirun -np 5 …"
- Make sure to allocate the extra task in batch jobs

Distributed Agent-based Correctness Analysis



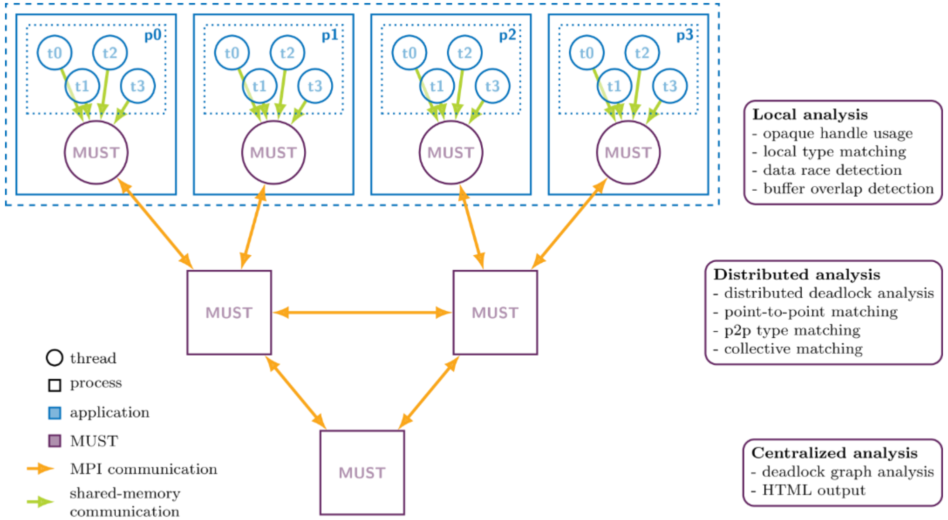
Local analysis
- opaque handle usage
- local type matching
- data race detection
- buffer overlap detection

Distributed analysis
- distributed deadlock analysis
- point-to-point matching
- p2p type matching
- collective matching

Centralized analysis
- deadlock graph analysis
- HTML output

Important `mustrun` flags

`--must:mpiexec <C>`	Sets C as the mpiexec command (e.g., srun)
`--must:nocrash`	Assert that application will not crash
`--must:info`	Print information on necessary processes
`--must:distributed`	Request multi-level analysis tree (fan-in=16)
`--must:fanin <N>`	Request a branching factor of N
`--must:hybrid`	Enables analysis of multi-threaded applications



ARCHER

Data race detection for OpenMP

What is ARCHER?

- Data race detection based on ThreadSanitizer (LLVM/GNU)
  - Runtime overhead 2-20x
- Archer runtime
  - Provides OpenMP synchronization semantics to the analysis based on OMPT
  - Delivered as part of LLVM since release 10.0
- Archer static analysis
  - Clang compiler pass
  - Exclude race-free code from runtime analysis → reduce overhead
  - Development stalled and not compatible with latest clang

Running ARCHER with an example

```
~/archer-example$ clang -fopenmp prime_omp.c -lm
~/archer-example$ OMP_NUM_THREADS=1 ./a.out
Number of prime numbers between 2 and 1000000: 78498
~/archer-example$ OMP_NUM_THREADS=4 ./a.out
Number of prime numbers between 2 and 1000000: 78409
~/archer-example$ clang -fsanitize=thread -g -fopenmp prime_omp.c –lm
~/archer-example$ OMP_NUM_THREADS=4 ./a.out
==================
WARNING: ThreadSanitizer: data race (pid=6999)
  Write of size 4 at 0x000001124ca8 by thread T3:
    #0 .omp_outlined._debug__ ~/archer-example/prime_omp.c:44:29 (a.out+0x4b6022)
    #1 .omp_outlined. ~/archer-example/prime_omp.c:40:5 (a.out+0x4b60b5)

  Previous write of size 4 at 0x000001124ca8 by thread T1:
    #0 .omp_outlined._debug__ ~/archer-example/prime_omp.c:44:29 (a.out+0x4b6022)
    #1 .omp_outlined. ~/archer-example/prime_omp.c:40:5 (a.out+0x4b60b5)

SUMMARY: ThreadSanitizer: data race ~/archer-example/prime_omp.c:44:29 in .omp_outlined._debug__
==================
```

When can ARCHER detect a data race?

- A data race is when **two threads** access the same data **without synchronization** and at least one thread **writes**.

```
41 #pragma omp parallel for
42     for (i = 2; i < N; i++) {
43         if ( is_prime(i) ) {
44             primes[total++] = i;
45         }
46     }
```

- Data race on `total` in line 44 is detected, if two different threads enter the body of the if-statement.
- Data race on `primes` in line 44 is detected, if above data race on `total++` manifests in missing an increment.
- There is no benign race in C, C++, or OpenMP! Any data race is UB!

How to use ARCHER?

```
clang -g -fopenmp -fsanitize=thread app.c
```

Fortran:

```
gfortran -g -c -fopenmp -fsanitize=thread app.c
clang -fopenmp -fsanitize=thread app.o -lgfortran \
  --gcc-toolchain=$(dirname $(dirname $(which gcc)))
```

- Sanitize flag is needed in compile and link step!
- Application must compile with LLVM or GNU compilers
- gfortran can be replaced with flang (more picky on Fortran standard!)

Important ARCHER flags

- Avoid false-positive reports from uninstrumented runtime libraries:

```
$ export TSAN_OPTIONS="ignore_noninstrumented_modules=1"
```

- Disable analysis for sequential part of a pure OpenMP program (LLVM/12):

```
$ export ARCHER_OPTIONS="ignore_serial=1"
```

Make sure OpenMP is initialized early!
Add omp_get_max_threads() at the beginning of main

- Disable loading of ARCHER at runtime:

```
$ export ARCHER_OPTIONS="enable=0"
```

- More flags:
  - https://github.com/llvm/llvm-project/tree/main/openmp/tools/archer

Limitations of ARCHER

- OpenMP tasks:
  - Dependencies: currently ARCHER sees synchronization for non-sibling tasks (will be fixed in LLVM/13 *)
  - Concurrency of tasks: currently ARCHER analyses on a threading level (task-level support to come in LLVM/13 *)
- OpenMP target regions:
  - No TSan analysis possible on GPUs, ARCHER can be used with x86 offloading
  - Might yield false positives, as ARCHER has no support yet for target constructs
- OpenMP reductions:
  - Memory accesses to implement reductions are ignored – might leed to omission of races. (runtime flag to toggle reduction handling in LLVM/13)

*) Preview: https://github.com/jprotze/llvm-project/tree/archer-task-fibers

MUST + ARCHER

Hybrid analysis of MPI + OpenMP

Features in *testing* state

For testing download
MUST 1.8 feature preview

- Analysis of data race in MPI:
  - Race between MPI buffer access and other thread
  - Race in MPI non-blocking communication
  - Race in MPI one-sided communication
  - Output onto commandline (or redirect into file using TSAN_OPTIONS)
- Type matching for application view on buffers
  - Compile-time information necessary for the analysis
  - Implemented as clang optimizer pass
- Both features rely on LLVM compiler / OpenMP runtime

Performance Optimisation and Productivity

A Centre of Excellence in HPC

Contact:

https://www.pop-coe.eu

mailto:pop@bsc.es

@POP_HPC

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 676553 and 824080.