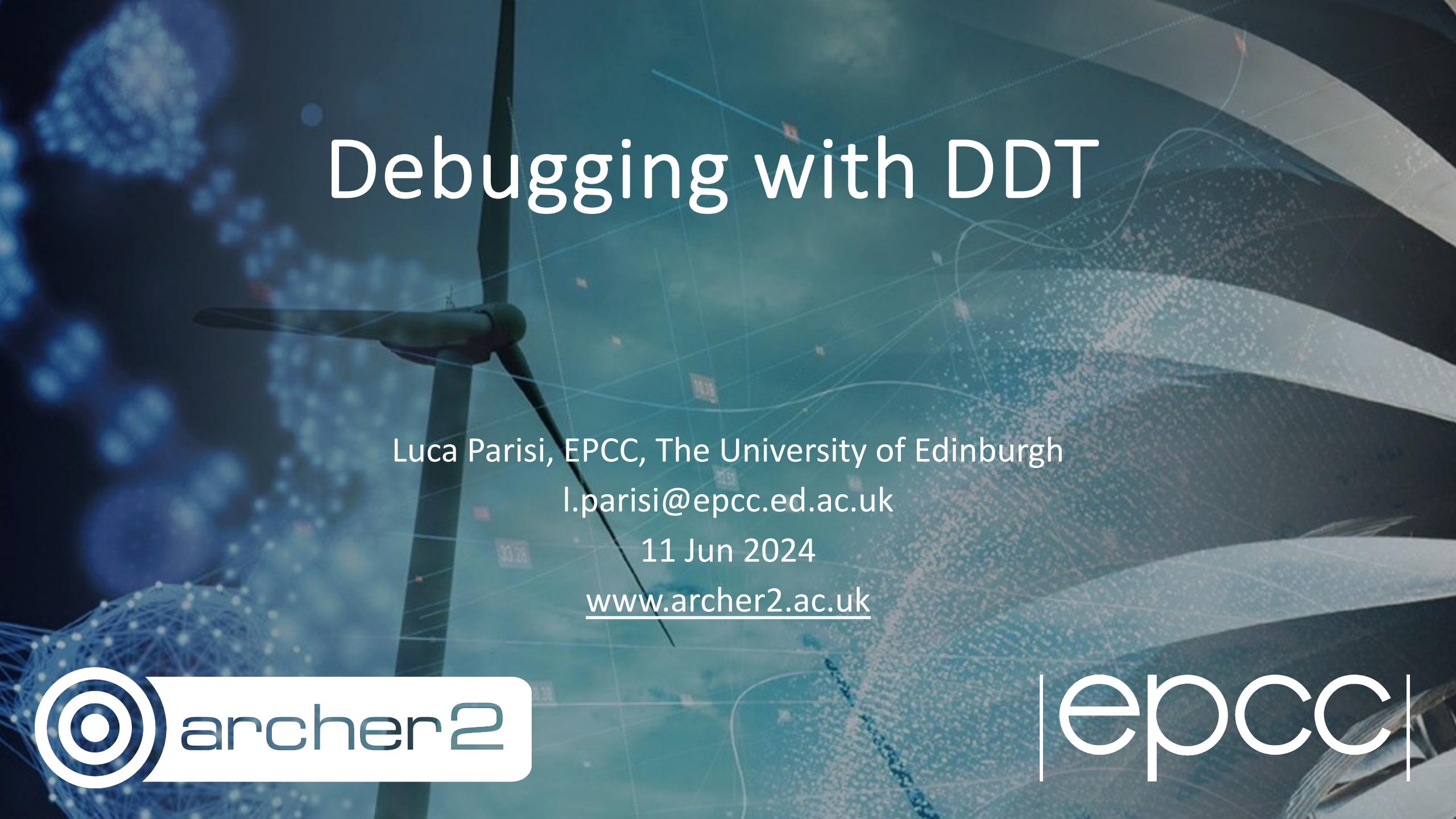


Debugging with DDT

Luca Parisi, EPCC, The University of Edinburgh
l.parisi@epcc.ed.ac.uk

11 Jun 2024

www.archer2.ac.uk

archer2

epcc

Outline

- Introduction to Debuggers
- Stepping through a parallel program
- Memory errors
- Memory management
- Offline Debugging

Introduction to debugging

What is a debugger ?

- Tool to inspect and test your program
- Allows you to step trough each line of your program : move to next line, step into functions, step over functions, break at a certain line
- Allows you to inspect the state of your program (print variable values, check memory usage etc..)

Serial debugging

GDB: The GNU Project Debugger

[bugs] [maintainers] [contributing] [current git] [documentation] [download] [home] [irc] [links] [mailing lists] [news] [schedule] [song] [wiki]

GDB: The GNU Project Debugger

What is GDB?

GDB, the GNU Project debugger, allows you to see what is going on `inside' another program while it executes -- or what another program was doing at the moment it crashed.

GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

Those programs might be executing on the same machine as GDB (native), on another machine (remote), or on a simulator. GDB can run on most popular UNIX and Microsoft Windows variants, as well as on macOS.

What Languages does GDB Support?

GDB supports the following languages (in alphabetical order):

- Ada
- Assembly
- C
- C++
- D
- Fortran
- Go
- Objective-C
- OpenCL
- Modula-2
- Pascal
- Rust

- GDB: one of the most common debuggers
- A optimized version of GDB is bundled inside DDT
- Command line interface

Parallel debugging

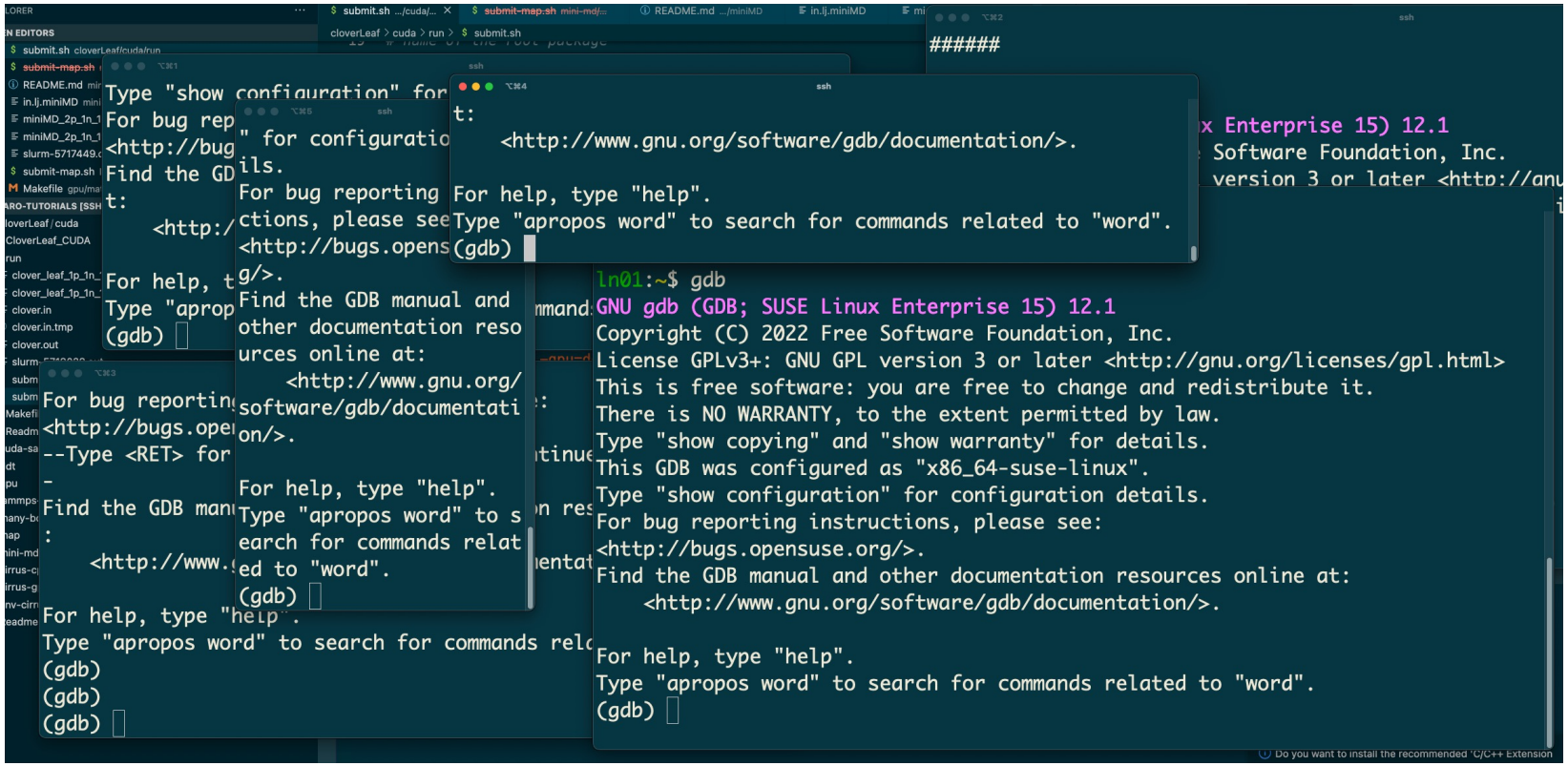
PROBLEM: What about debugging a parallel program ?

- GDB is designed for serial processes

SOLUTION: Run a GDB instance for each process in the application ?

- Run GDB in a different window for each process
- Works fine for a handful of processes

Parallel debugging



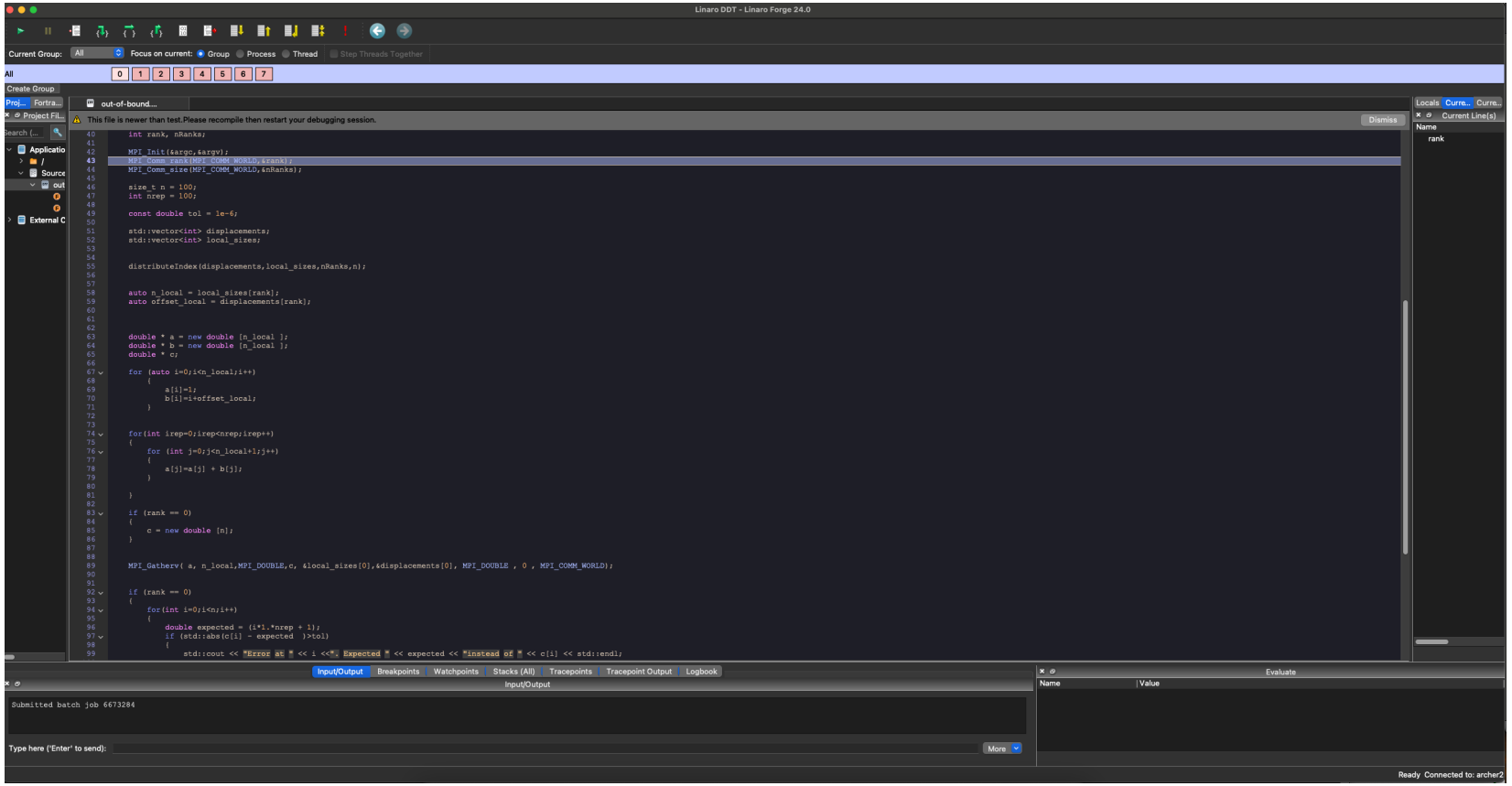
PROBLEM: Does not scale !!!

Parallel debuggers

- Gdb4hpc : Comes with the cray programming environment. Command line interface only
- TotalView: Contains a parallel debugger. Commercial product.
- Linaro Forge: Contains DDT, a parallel debugger . Commercial product.

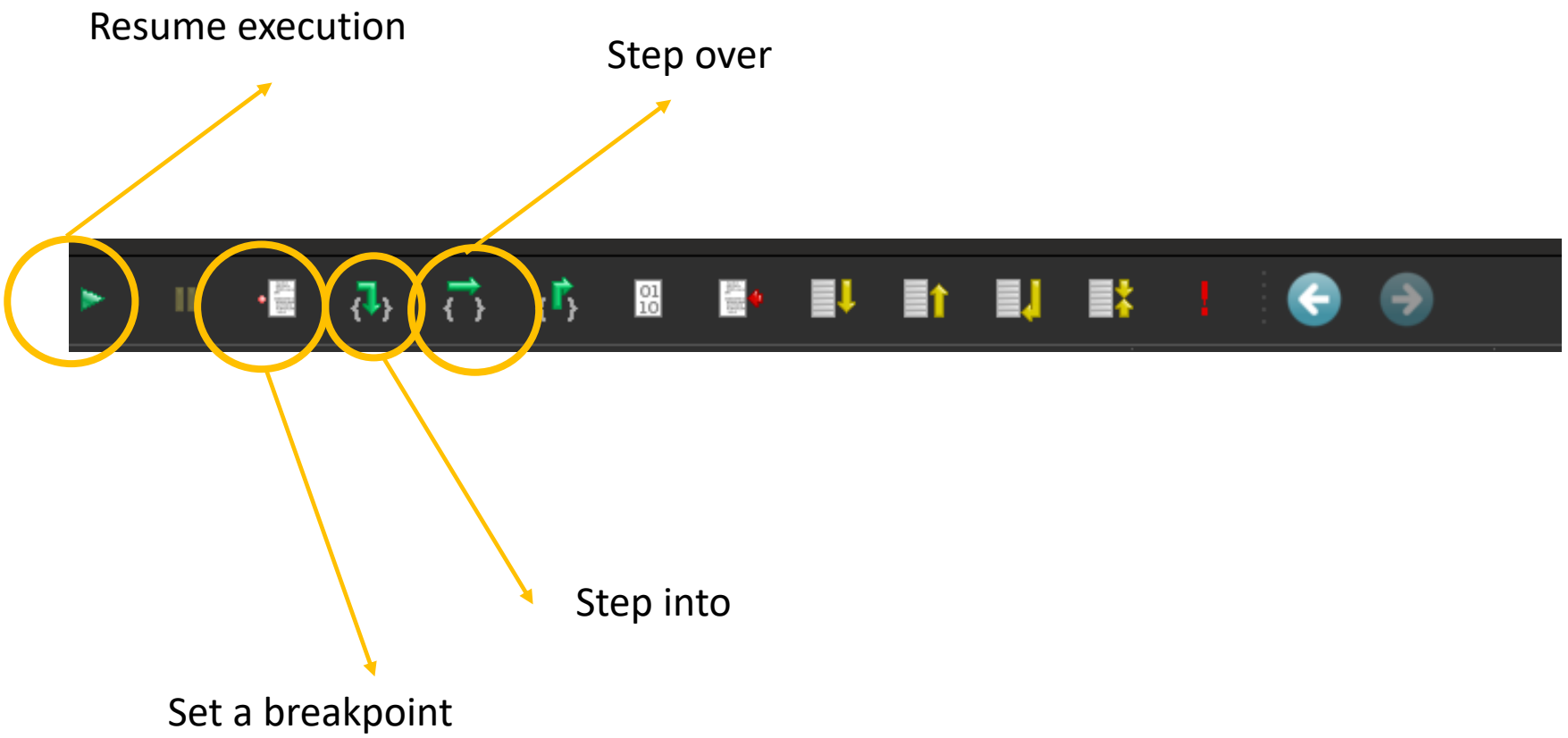
Linaro DDT

- Has a graphical interface
- A single interface to inspect all processes and threads



Stepping through the code

Stepping through the code



Stepping through the code

Green: running

Red: stopped

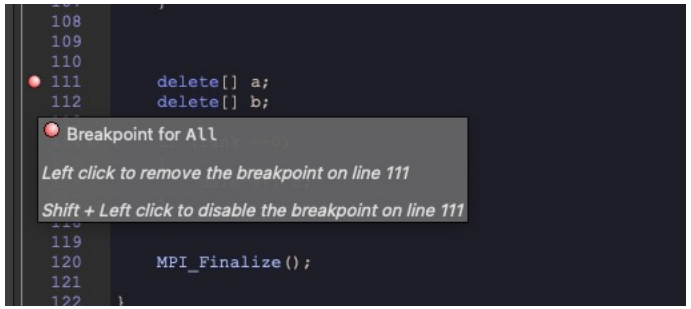
Focus on a single process, subset of processes or a single thread

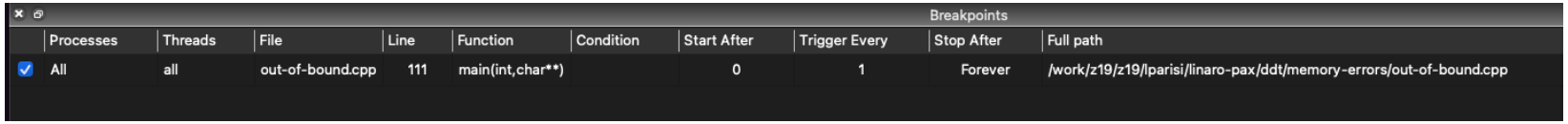
Option to run threads in lock step or advance one thread at a time

- Information is present for the current focus
- Can step a single process or all processes in lock step

Stepping through the code

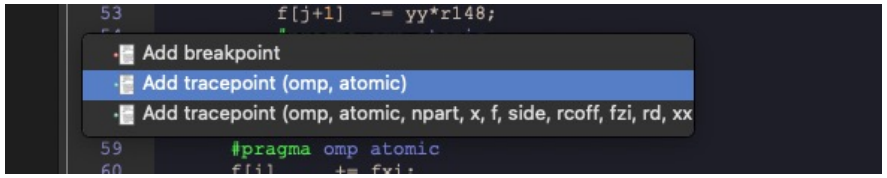
Breakpoints: stop at a certain line in the code

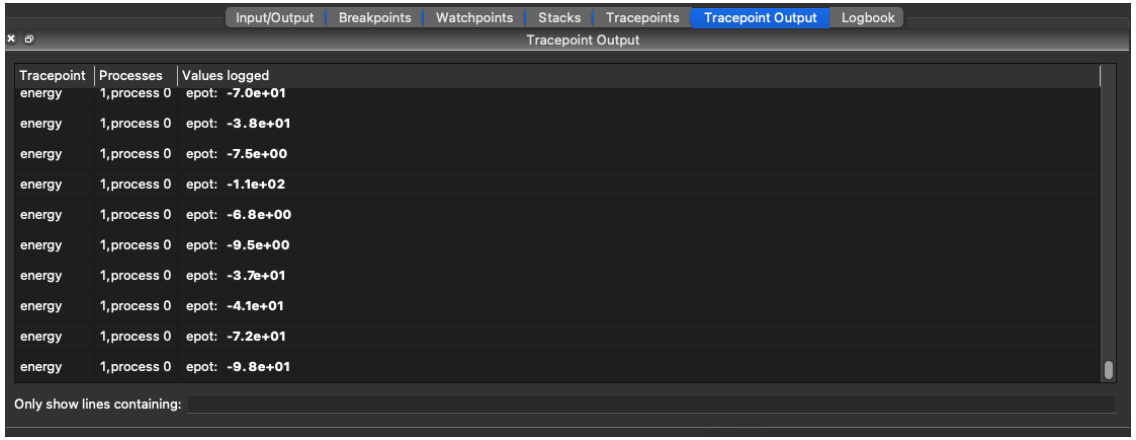


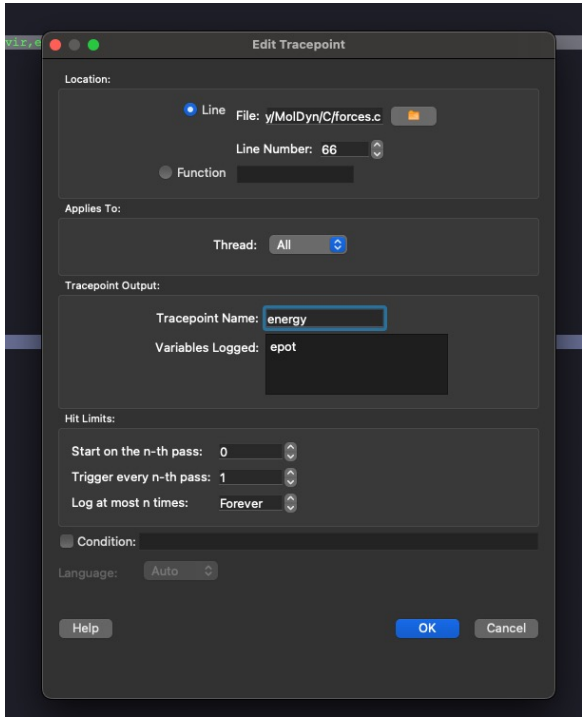


Stepping through the code

- **Tracepoints**: prints the value of an expression without stopping the program

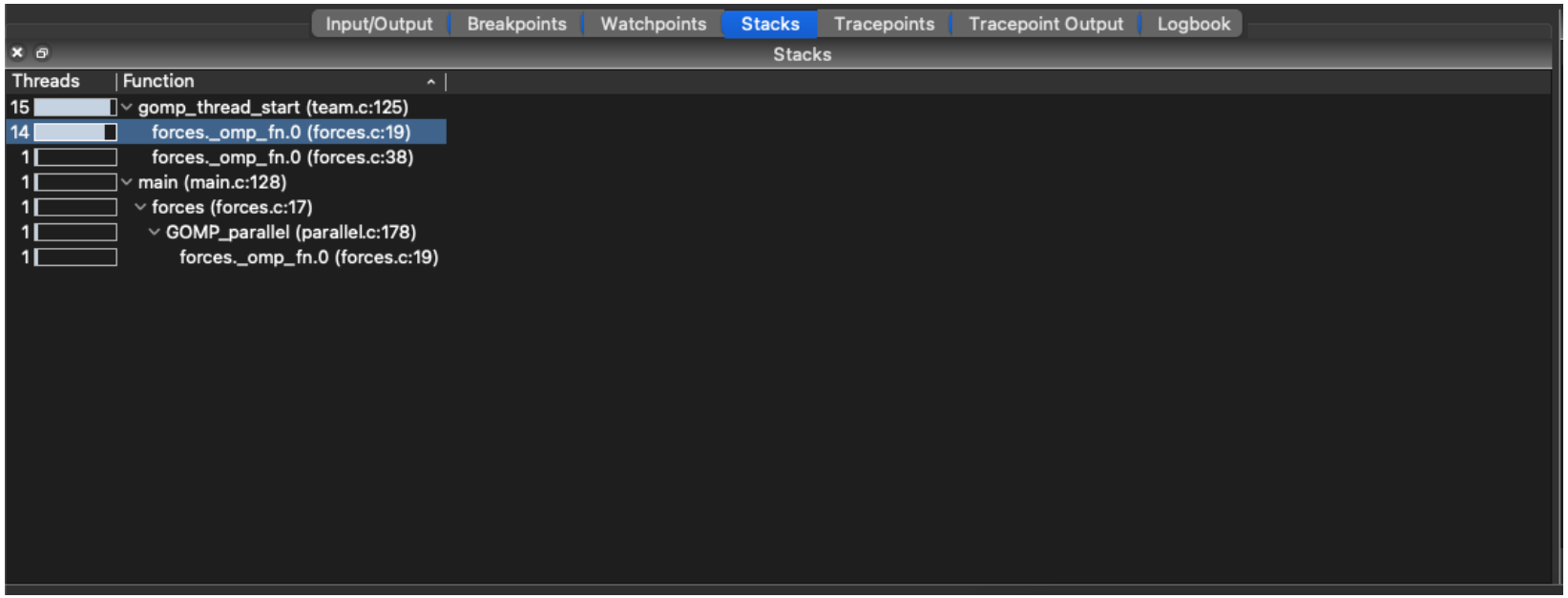






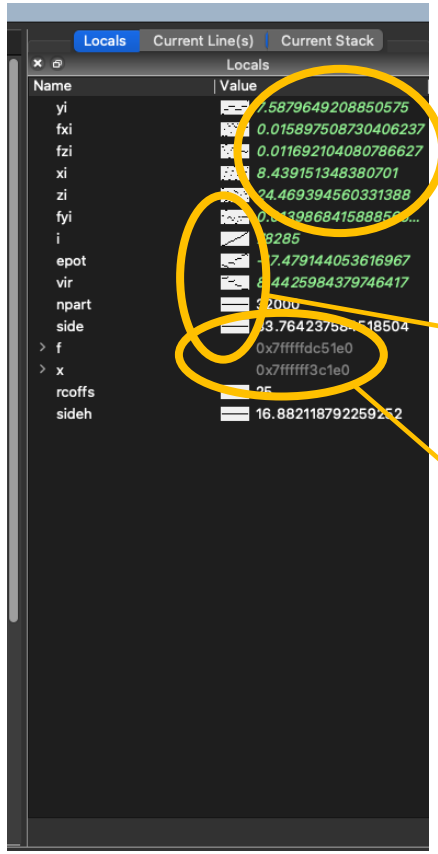
Inspecting the stack trace

The stack trace: see which line each process/thread is executing at a certain time



Viewing variables

The alternative tf

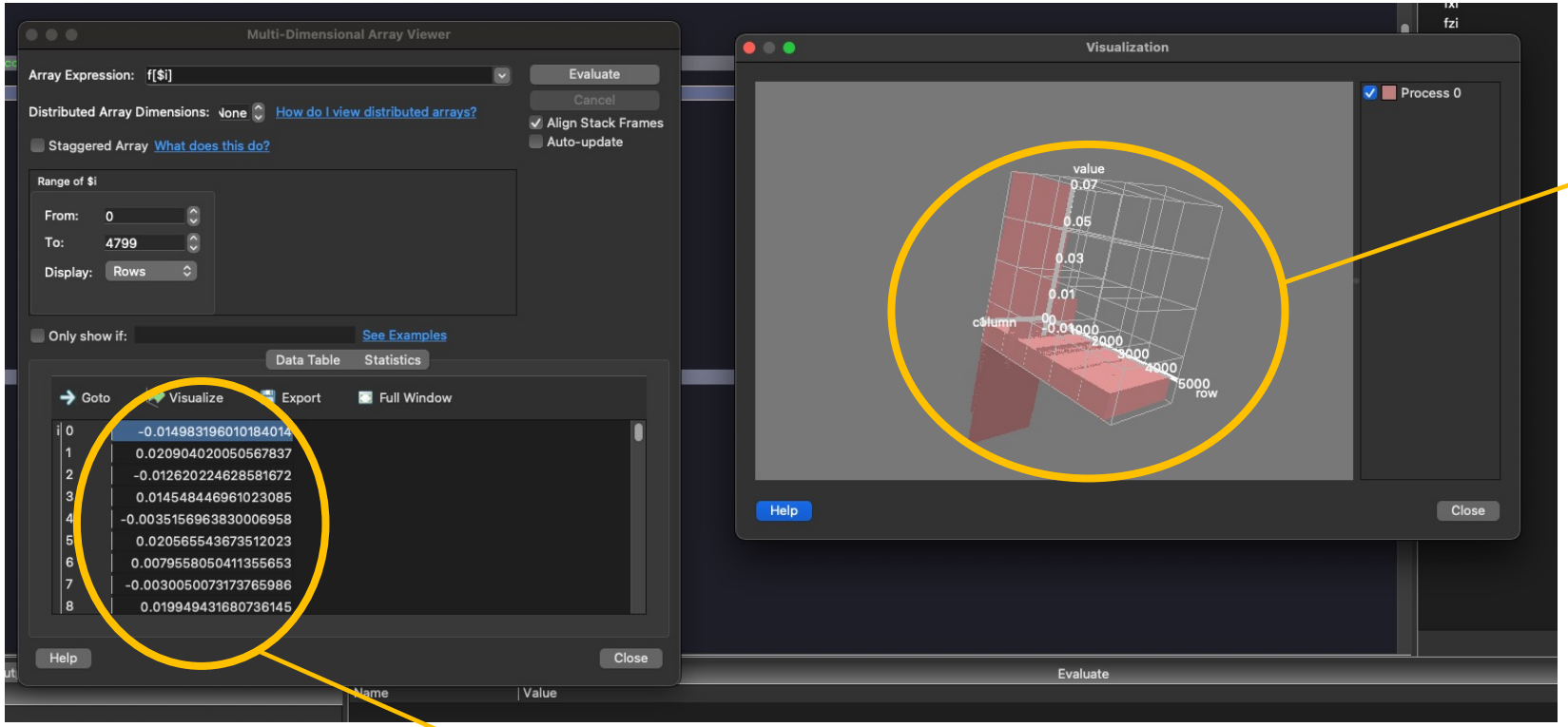


Value of variables in current frame on selected thread and process

Sparkline show values across all threads and processes

Arrays show up as an address

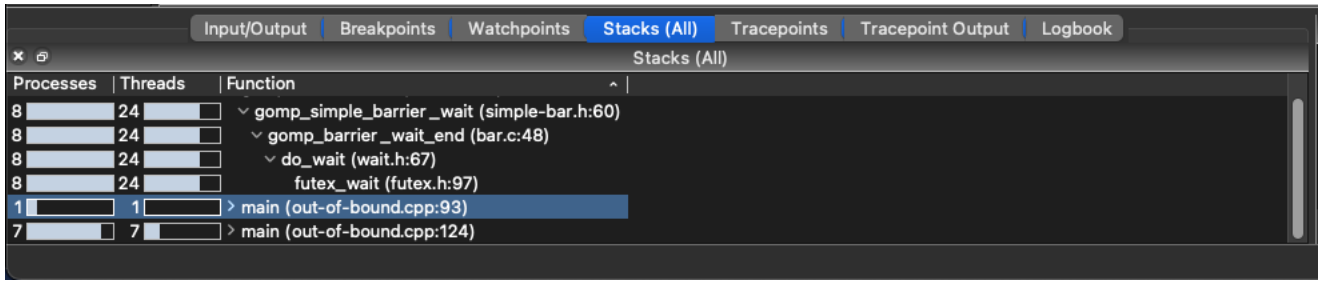
Multidimensional array viewer



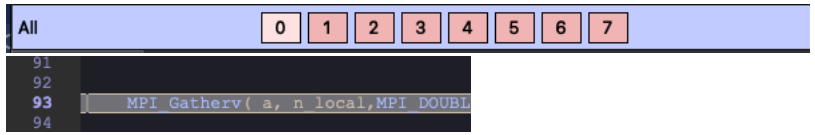
3D plot of numerical values

Display numerical values

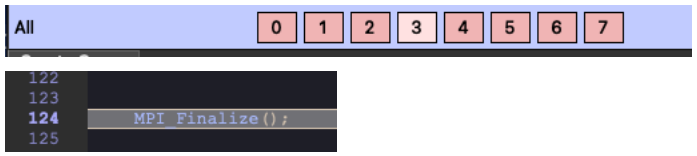
Deadlocks



1 process is at line 93



Process 0 is in a collective at line 93



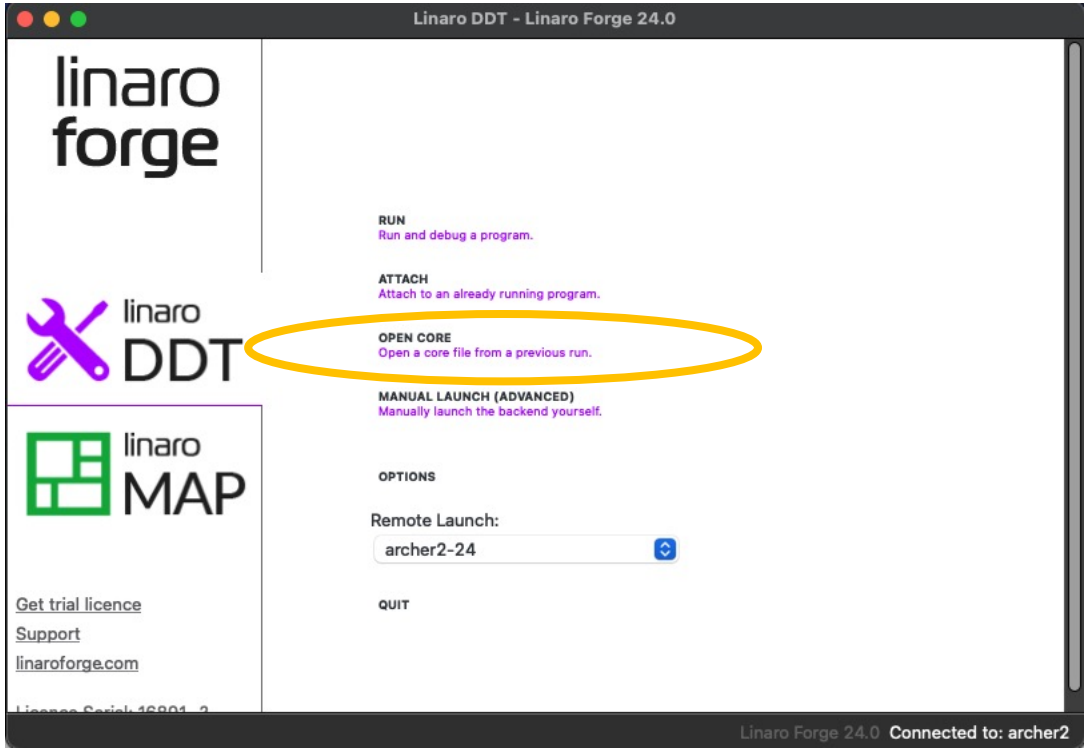
Other processes have skipped the collective to the end of the program

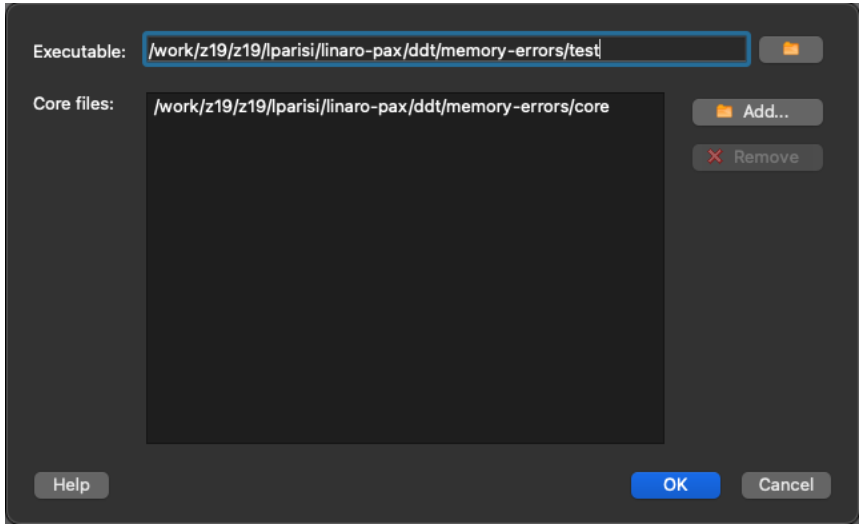
Deadlock !

Memory debugging

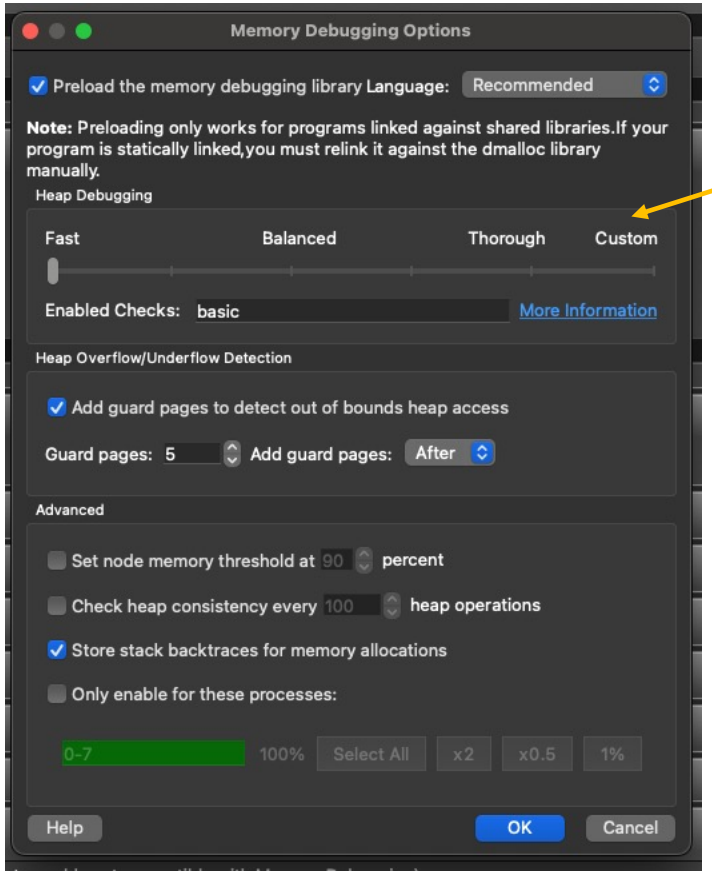
Core files

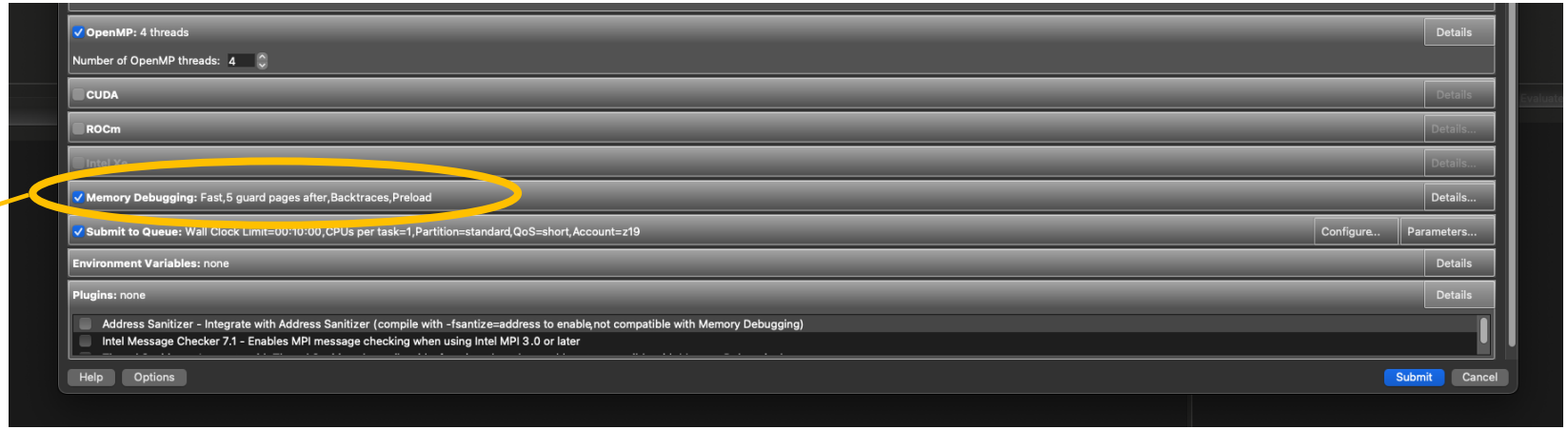
```
ERROR: SEGMENTATION FAULT
```





Memory debugging



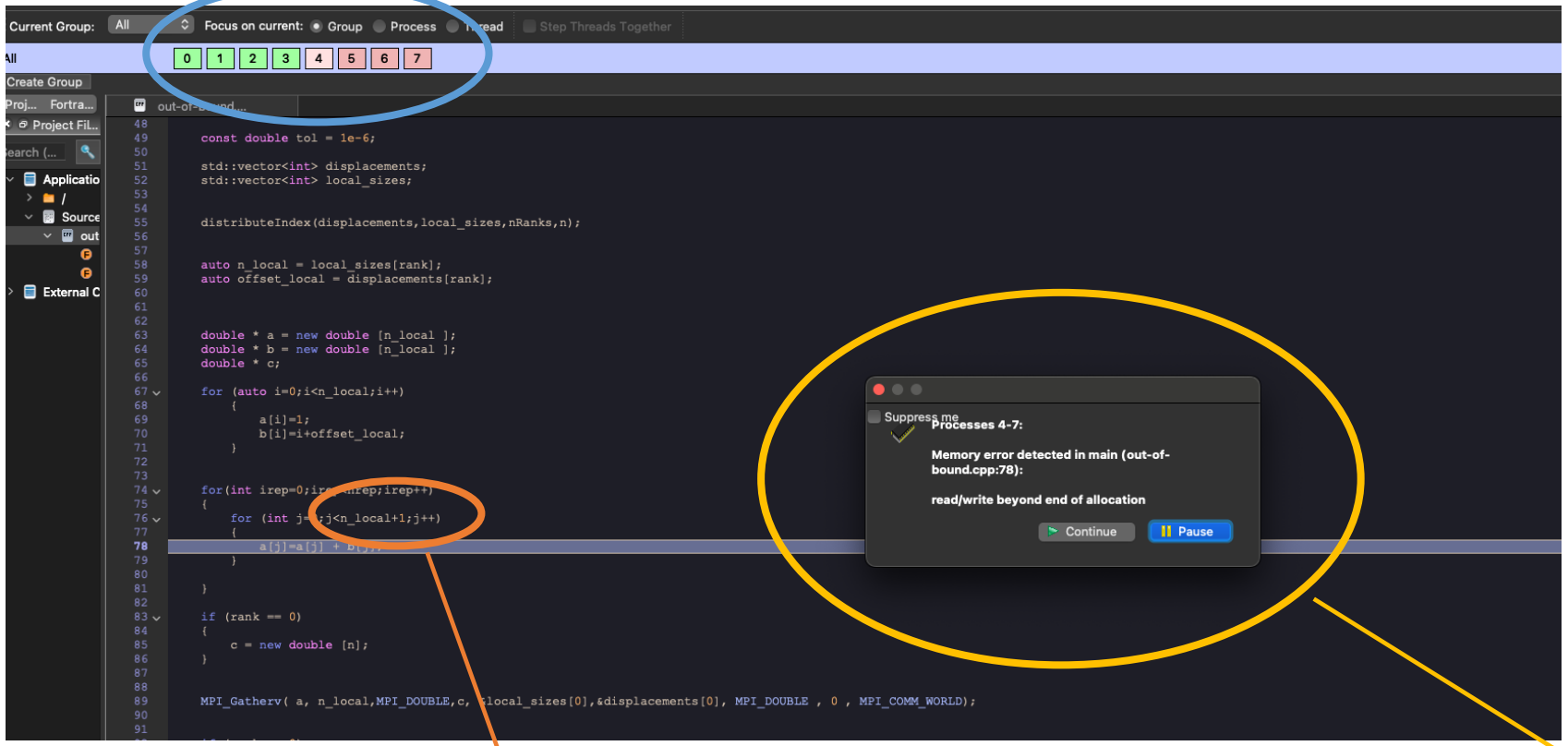


Enable various memory checks:

- Check for invalid pointers
- Check for memory corruption
- Check for accesses to previously freed memory
- Explicitly initialize memory on allocation

Out of bound access

Shows on which ranks the error was encountered



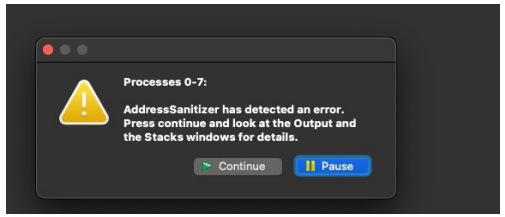
The guilty line

Warning, out of bound access

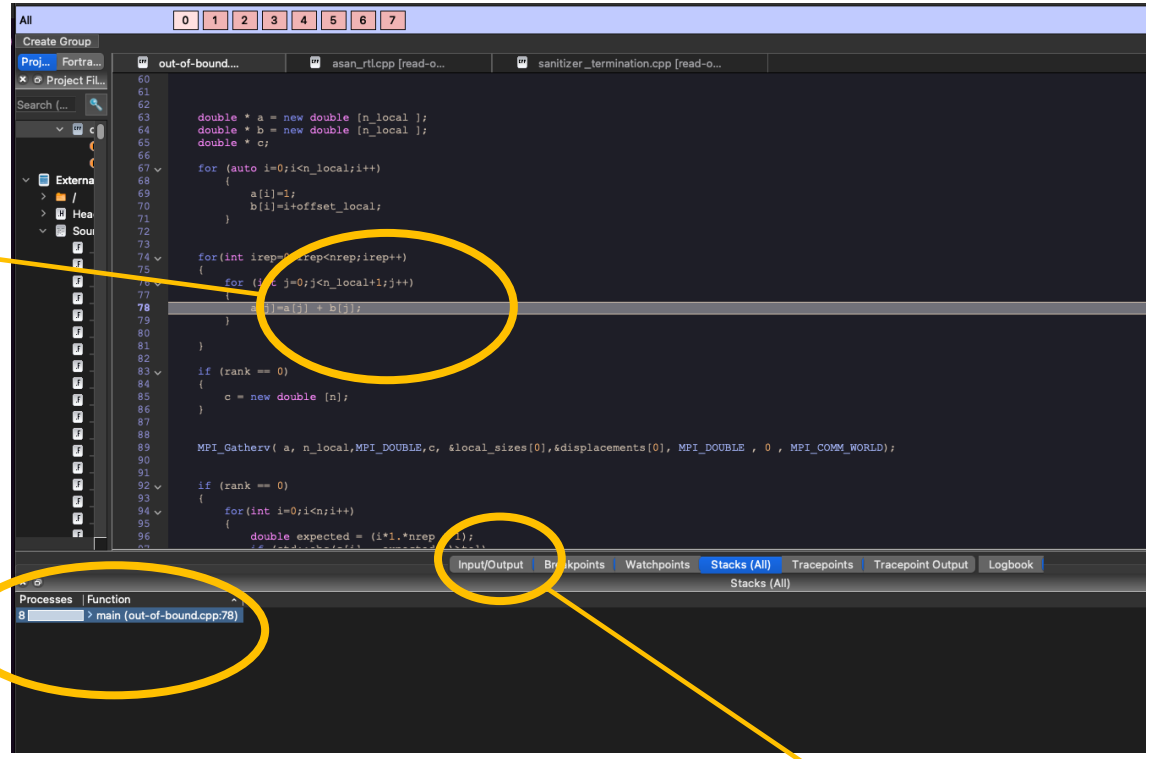
Address sanitizer

- Compile with `-fsanitize=address`
- Supported by many compilers built on LLVM (Clang, Cray)
- DDT has a plugin for address sanitizer. Not compatible with memory debugging
- Reasonable overhead

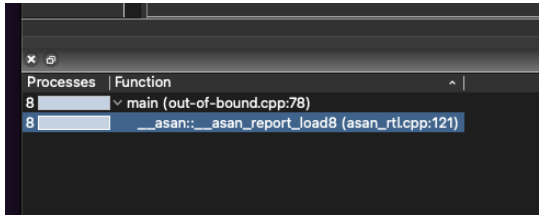
Address sanitizer

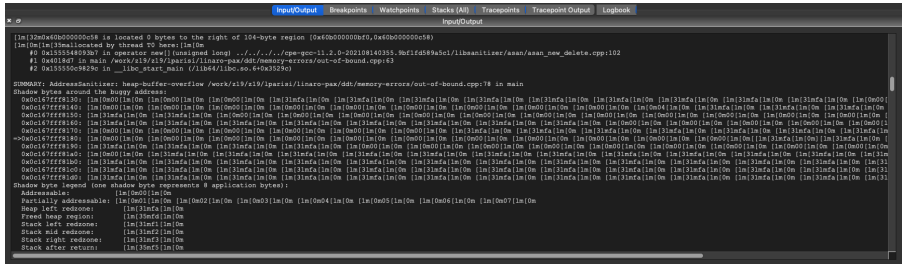


Troublesome lines



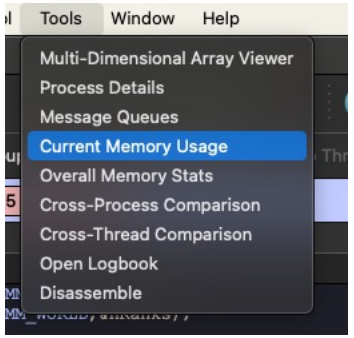
Stack when address sanitizer registered an issue





Details of the error

Memory management



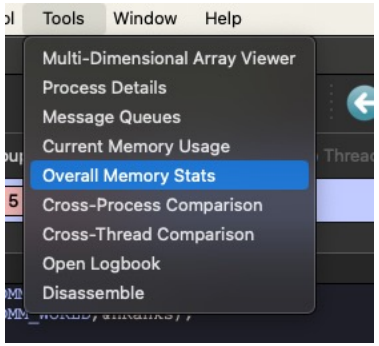
Different color for allocations from different functions, including MPI internal allocations

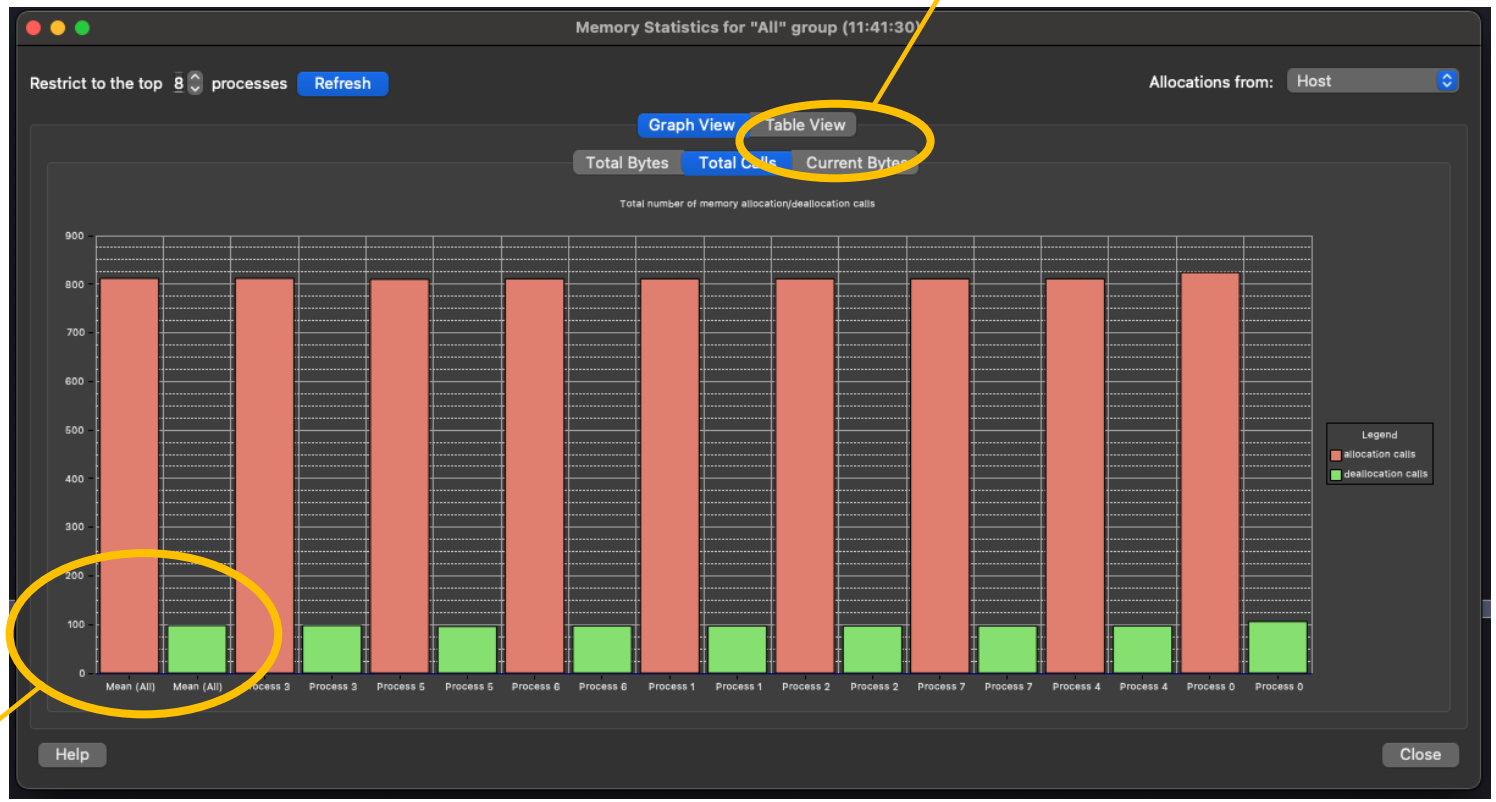
Information about memory allocations

Histogram of memory usage across ranks

Memory management

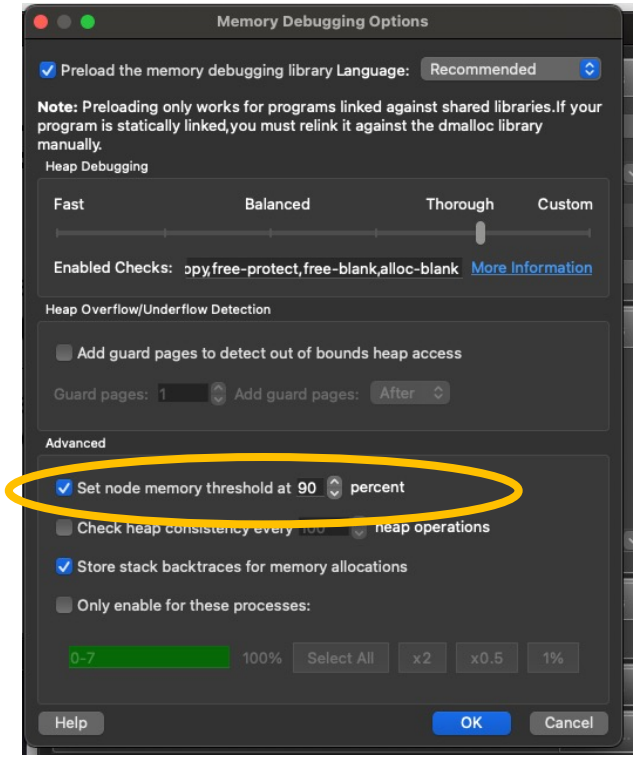
How many and how big allocations and deallocations done per rank ?



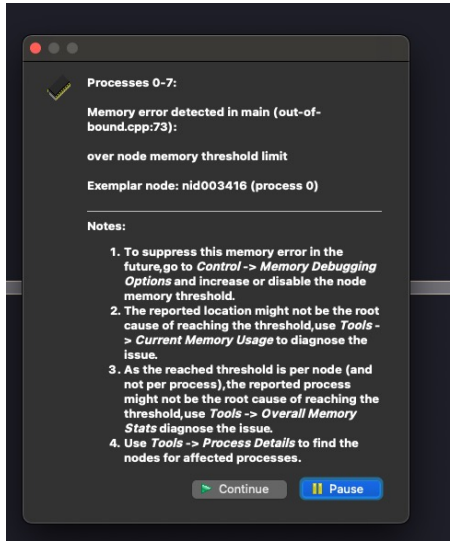


Growing number of memory allocations, but not deallocations suggests a memory leak

Out Of Memory



Breaks when memory allocations go above a certain threshold



Inspect the last allocation call and the memory allocations of the program

```
69
70
71      for(int irep=0;irep<nrep;irep++)
72      {
73          b = new double [ n_local ];
74          for (auto i=0;i<n_local;i++)
75          {
76              b[i]=i+offset_local;
77          }
78
79  #pragma omp parallel for
80          for (int j=0;j<n_local;j++)
81          {
82              a[j]=a[j] + b[j];
83          }
84
85      }
```

Offline debugging

Offline debugging

Can set breakpoints, tracepoints etc.

```
module load forge/24
ddt --mem-debug -o memory_errors.html --offline --break-at=out-of-
bound.cppmiscellanea:73 srun --hint=nomultithread --distribution=block:block
./test
```

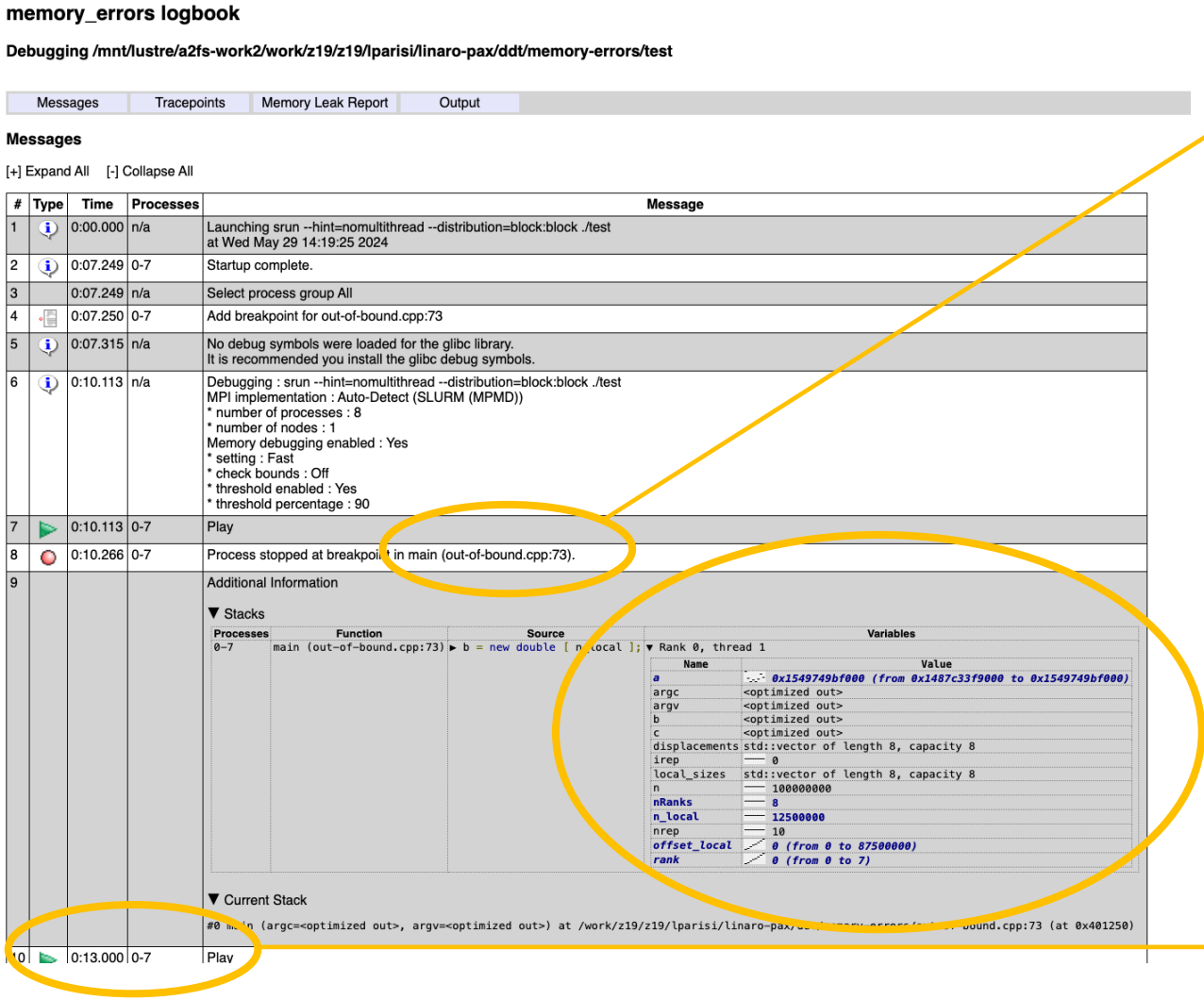
Output html report

Do not use the GUI interface

```
ddt --help
```

To get a list of all accepted flags

Offline debugging



memory_errors logbook

Debugging /mnt/lustre/a2fs-work2/work/z19/z19/lparisi/linaro-pax/ddt/memory-errors/test

Messages | Tracepoints | Memory Leak Report | Output

Messages

[+] Expand All [-] Collapse All

#	Type	Time	Processes	Message
1		0:00.000	n/a	Launching srun --hint=nomultithread --distribution=block:block ./test at Wed May 29 14:19:25 2024
2		0:07.249	0-7	Startup complete.
3		0:07.249	n/a	Select process group All
4		0:07.250	0-7	Add breakpoint for out-of-bound.cpp:73
5		0:07.315	n/a	No debug symbols were loaded for the glibc library. It is recommended you install the glibc debug symbols.
6		0:10.113	n/a	Debugging : srun --hint=nomultithread --distribution=block:block ./test MPI implementation : Auto-Detect (SLURM (MPMD)) * number of processes : 8 * number of nodes : 1 Memory debugging enabled : Yes * setting : Fast * check bounds : Off * threshold enabled : Yes * threshold percentage : 90
7		0:10.113	0-7	Play
8		0:10.266	0-7	Process stopped at breakpoi in main (out-of-bound.cpp:73).
9				Additional Information ▼ Stacks Processes: 0-7 | Function: main (out-of-bound.cpp:73) | Source: ▶ b = new double [n_local];
10		0:13.000	0-7	Play

▼ Rank 0, thread 1

Name	Value
a	0x1549749bf000 (from 0x1487c33f9000 to 0x1549749bf000)
argc	<optimized out>
argv	<optimized out>
b	<optimized out>
c	<optimized out>
displacements	std::vector of length 8, capacity 8
irep	0
local_sizes	std::vector of length 8, capacity 8
n	100000000
nRanks	8
n_local	12500000
nrep	10
offset_local	0 (from 0 to 87500000)
rank	0 (from 0 to 7)

▼ Current Stack

#0 m...n (argc=<optimized out>, argv=<optimized out>) at /work/z19/z19/lparisi/linaro-pax/d... ...ound.cpp:73 (at 0x401250)

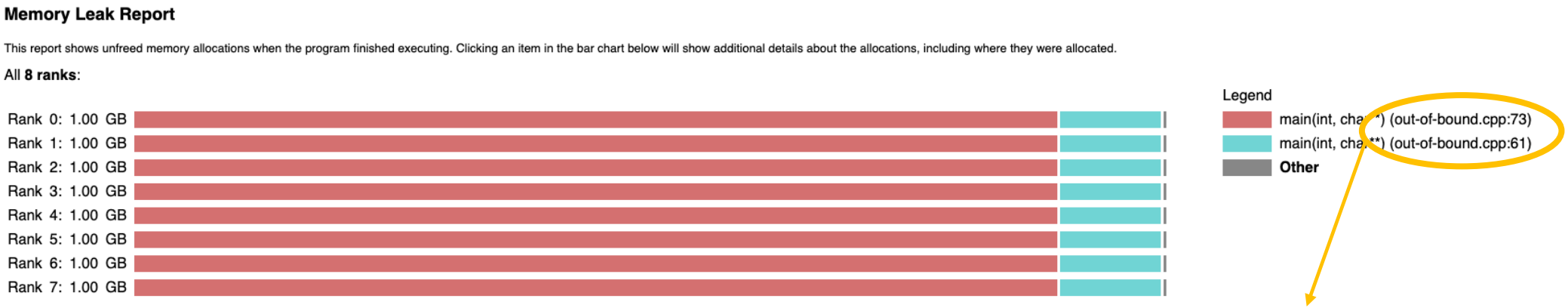
Debugger encountered a breakpoint

Stack trace when breakpoint was reached

Resume execution

Memory leaks -- offline

Offline report contains memory leaks information captured at the end of the program



Memory at these lines was allocated but never freed

Conclusions

- Use a debugger when **printf** is not enough
- Track memory errors: corruption, out of bound access etc..
- Find out why memory is leaking or why you are getting OOM errors