

Debugging with DDT

Luca Parisi, EPCC, The University of Edinburgh

l.parisi@epcc.ed.ac.uk

11 Jun 2024

www.archer2.ac.uk



Outline

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



Introduction to debugging



|epcc|

What is a debugger ?

- Tool to inspect and test your program
- Allows you to step through 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



The screenshot shows the official GDB website. At the top, there's a navigation bar with links like [bugs], [maintainers], [contributing], [current git], [documentation], [download], [home], [fix], [links], [mailing lists], [news], [schedule], [song], and [wiki]. Below the navigation is a large title "GDB: The GNU Project Debugger" with a small cartoon fish icon to its right. Under the title, there's a section titled "What is GDB?" which defines it as a debugger for seeing what's happening inside a program. It also lists four main uses: starting programs, stopping them at specific conditions, examining what happened when they stopped, and changing things in the program to experiment with bug effects. A note says GDB runs on various platforms including native, remote, and simulator environments. Another section, "What Languages does GDB Support?", lists the languages it can debug, including Ada, Assembly, C, C++, D, Fortran, Go, Objective-C, OpenCL, Modula-2, Pascal, and 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

A screenshot of a terminal window with multiple tabs open. The active tab shows the GDB documentation. The text in the terminal is:

```
lin01:~$ gdb
GNU gdb (GDB; SUSE Linux Enterprise 15) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-suse-linux".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://bugs.opensuse.org/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb)
```

The terminal window has a dark background and light-colored text. A tooltip is visible in the upper right corner of the terminal window, providing information about the GDB command "apropos".

PROBLEM: Does not scale !!!

- [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.

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

The screenshot shows the Linaro DDT interface running on a Mac OS X system. The main window title is "Linaro DDT - Linaro Forge 24.0". The left sidebar shows a project structure with "Application", "Source", and "External C". The "Source" section contains a file named "test.c" which has a warning: "This file is newer than test. Please recompile then restart your debugging session." The code in "test.c" is as follows:

```
1  int rank, nRanks;
2
3  MPI_Init(&argc,&argv);
4  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
5  MPI_Comm_size(MPI_COMM_WORLD, &nRanks);
6
7  size_t n = 100;
8  int nrep = 100;
9
10 const double tol = 1e-6;
11
12 std::vector<int> displacements;
13 std::vector<int> local_sizes;
14
15 distributeIndex(displacements,local_sizes,nRanks,n);
16
17 auto n_local = local_sizes[rank];
18 auto offset_local = displacements[rank];
19
20
21 double * a = new double [n_local];
22 double * b = new double [n_local];
23 double * c;
24
25 for (auto i=0;i<n_local;i++)
26 {
27     a[i]=1;
28     b[i]=i+offset_local;
29 }
30
31
32 for(int irep=0;irep<nrep;irep++)
33 {
34     for (int j=0;j<n_local;j++)
35     {
36         a[j]=a[j] + b[j];
37     }
38 }
39
40 if (rank == 0)
41 {
42     c = new double [n];
43 }
44
45 MPI_Gatherv( a, n_local,MPI_DOUBLE,c, &local_sizes[0],&displacements[0], 0 , MPI_COMM_WORLD);
46
47
48 if (rank == 0)
49 {
50     for(int i=0;i<n;i++)
51     {
52         double expected = (i*1.*nrep + 1);
53         if (std::abs(c[i] - expected) > tol)
54             std::cout << "Error at " << i << ". Expected " << expected << " instead of " << c[i] << std::endl;
55     }
56 }
```

The right side of the interface shows a "Locals" panel with a variable "rank" set to 0. At the bottom, there is a "Logbook" tab showing the message "Submitted batch job 6673284" and a "Type here ('Enter' to send):" input field.

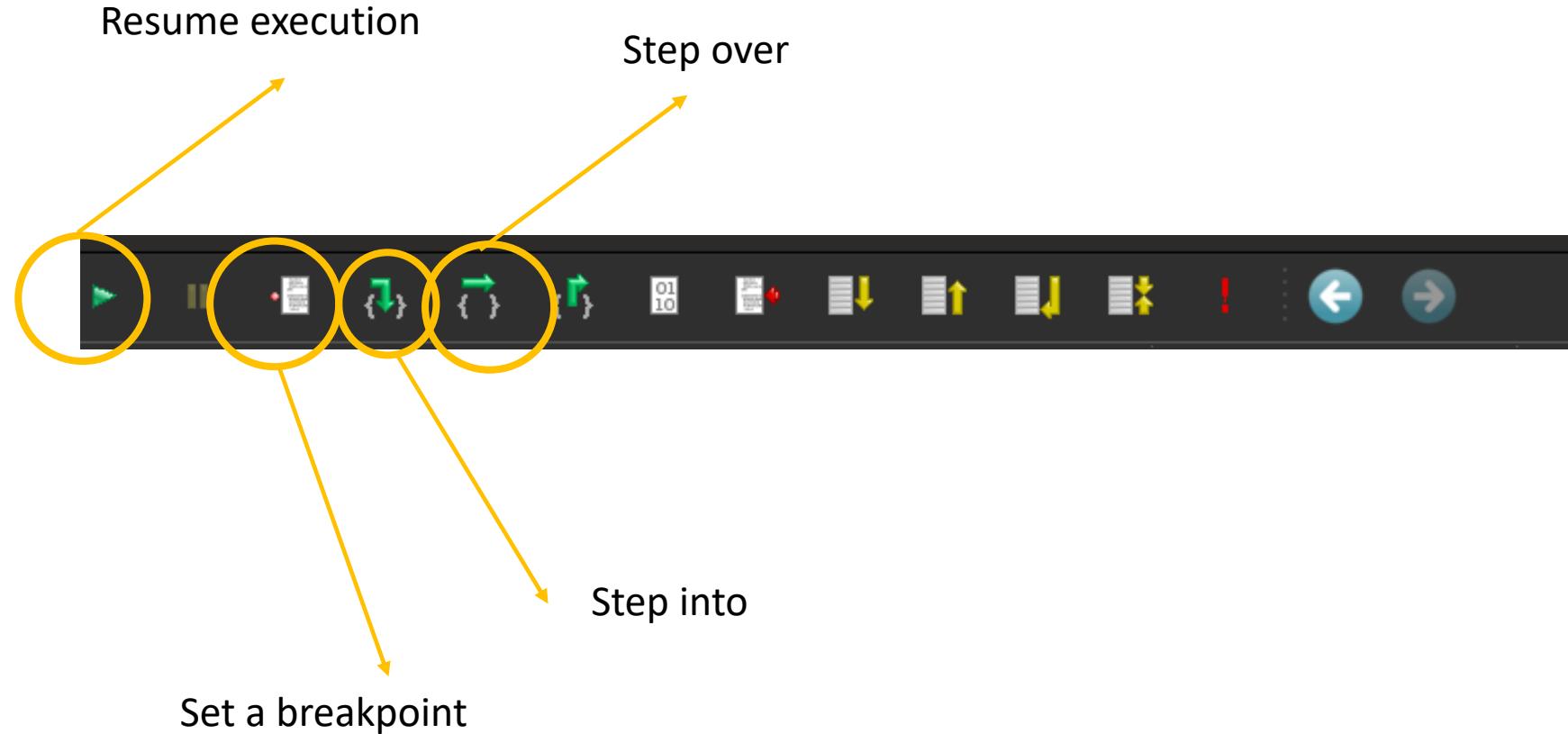


Stepping through the code



|epcc|

Stepping through the code



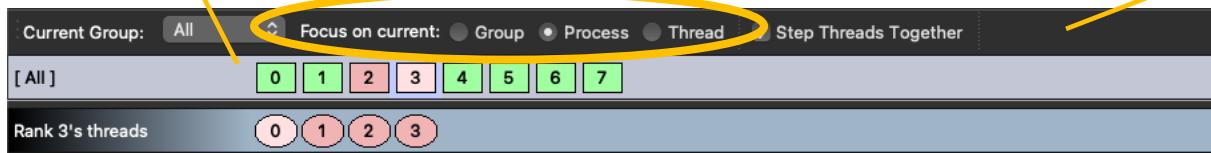
Stepping through the code

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

A screenshot of a terminal window showing a C++ code editor. The code is as follows:

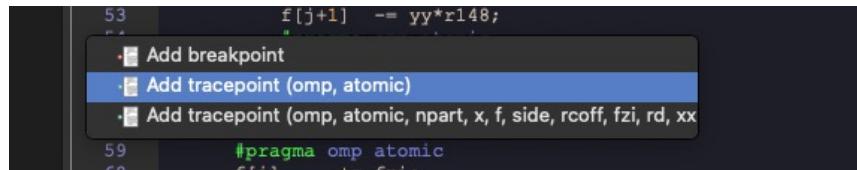
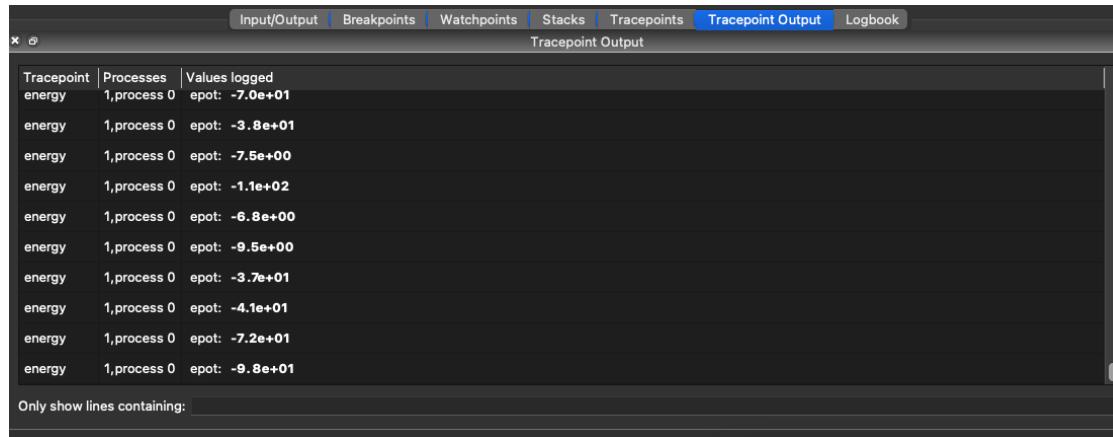
```
107
108
109
110
111     delete[] a;
112     delete[] b;
113
114     Breakpoint for All
115
116     Left click to remove the breakpoint on line 111
117
118     Shift + Left click to disable the breakpoint on line 111
119
120     MPI_Finalize();
121
122 }
```

The line 111 is highlighted with a red dot, indicating it is a breakpoint. A tooltip "Breakpoint for All" is visible above the line. Below the code, there are instructions: "Left click to remove the breakpoint on line 111" and "Shift + Left click to disable the breakpoint on line 111".

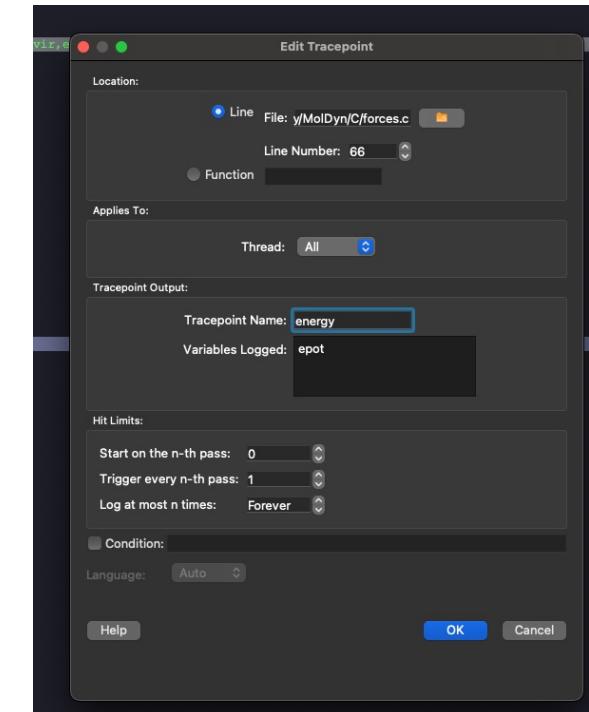
Breakpoints									
Processes	Threads	File	Line	Function	Condition	Start After	Trigger Every	Stop After	Full path
All	all	out-of-bound.cpp	111	main(int,char**)		0	1	Forever	/work/z19/z19/lparisi/linaro-pax/ddt/memory-errors/out-of-bound.cpp

Stepping through the code

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

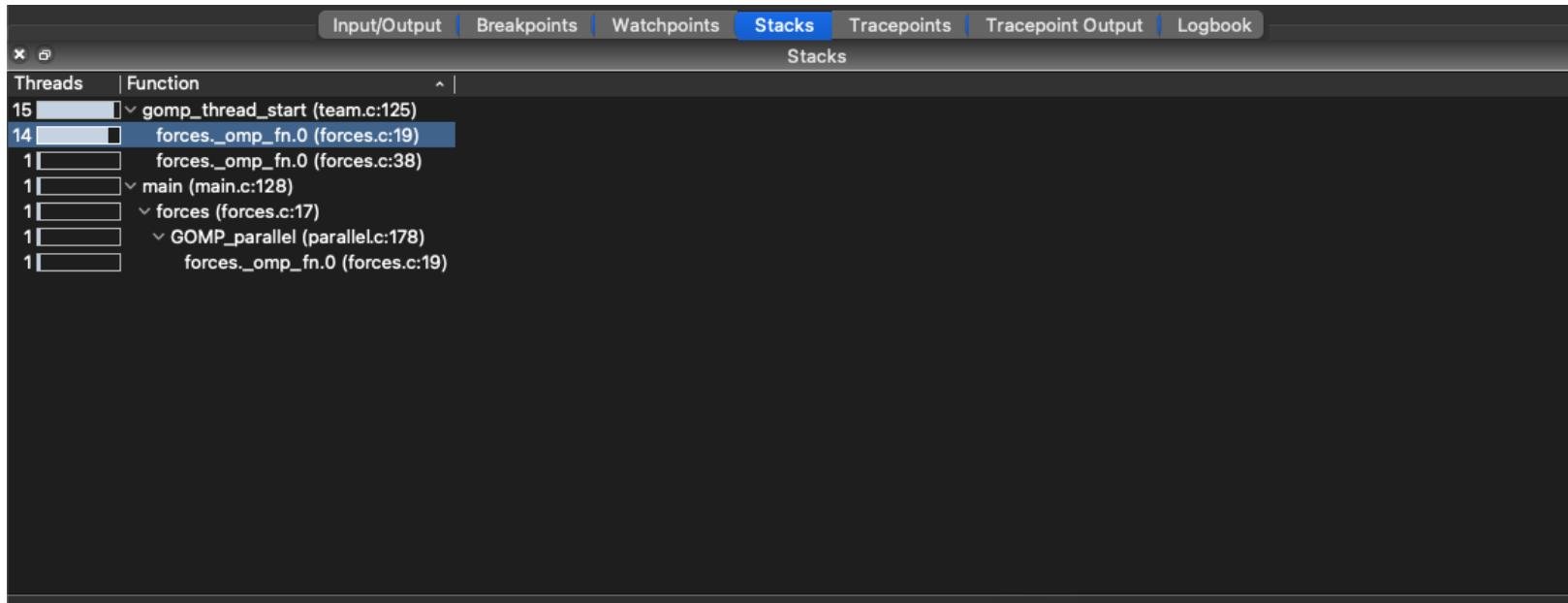



Tracepoint	Processes	Values logged
energy	1,process 0	epot: -7.0e+01
energy	1,process 0	epot: -3.8e+01
energy	1,process 0	epot: -7.5e+00
energy	1,process 0	epot: -1.1e+02
energy	1,process 0	epot: -6.8e+00
energy	1,process 0	epot: -9.5e+00
energy	1,process 0	epot: -3.7e+01
energy	1,process 0	epot: -4.1e+01
energy	1,process 0	epot: -7.2e+01
energy	1,process 0	epot: -9.8e+01



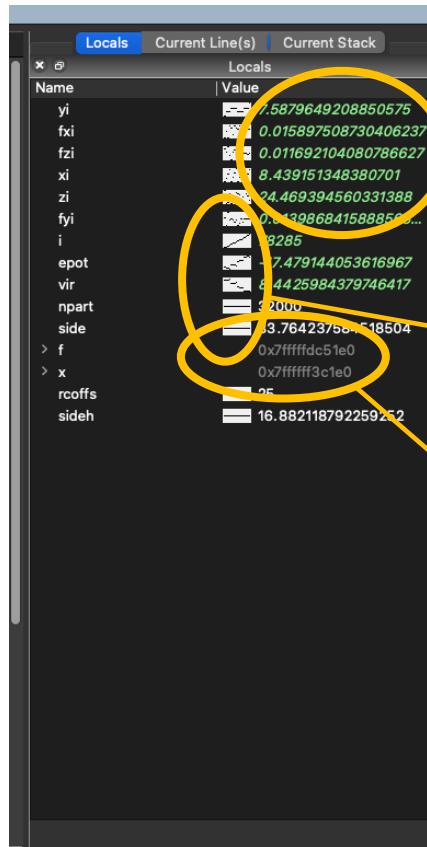
Inspecting the stack trace

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



Viewing variables

The alternative tf

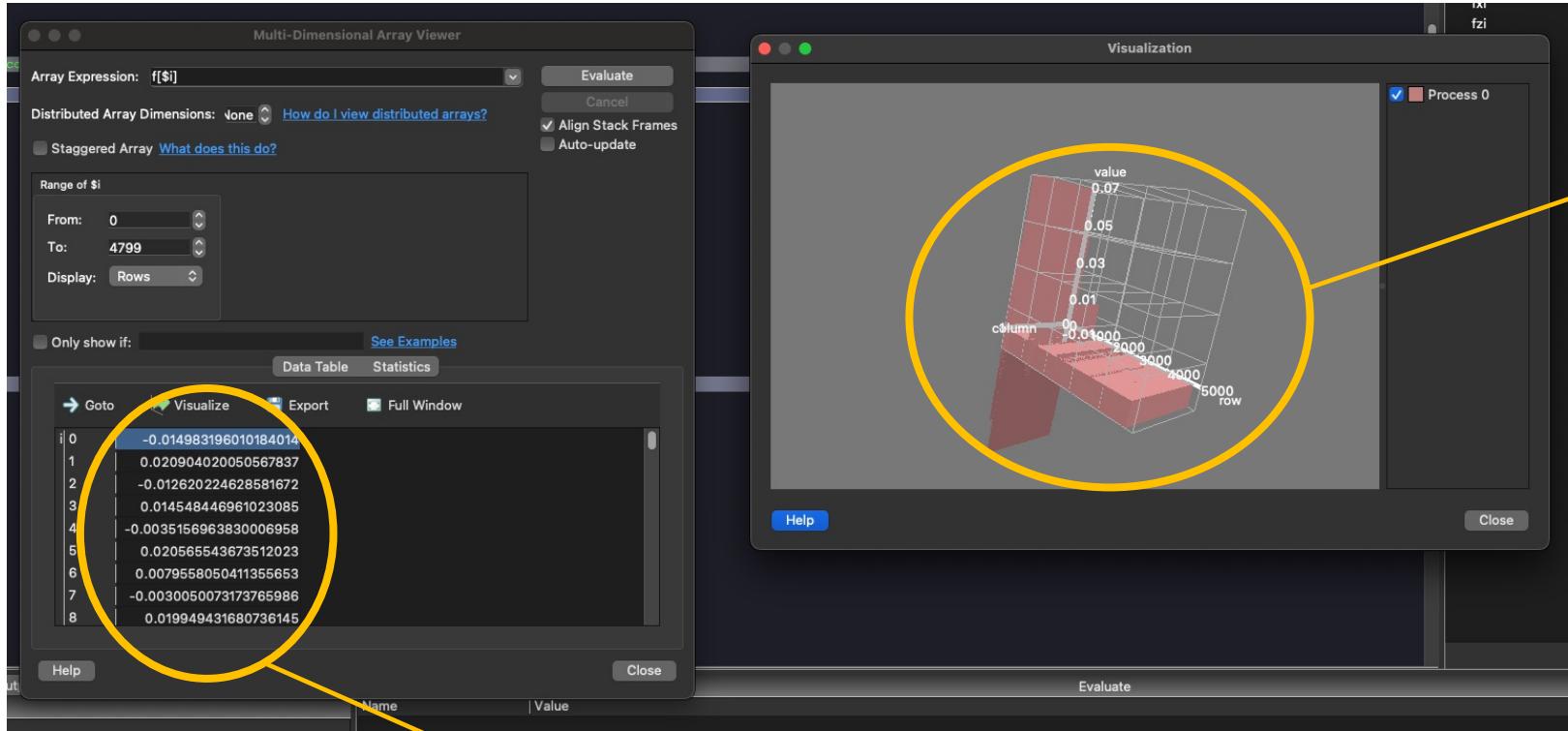


Value of variables in current frame on selected thread and process

Sparkline show values across all threads and processes

Arrays show up as an address

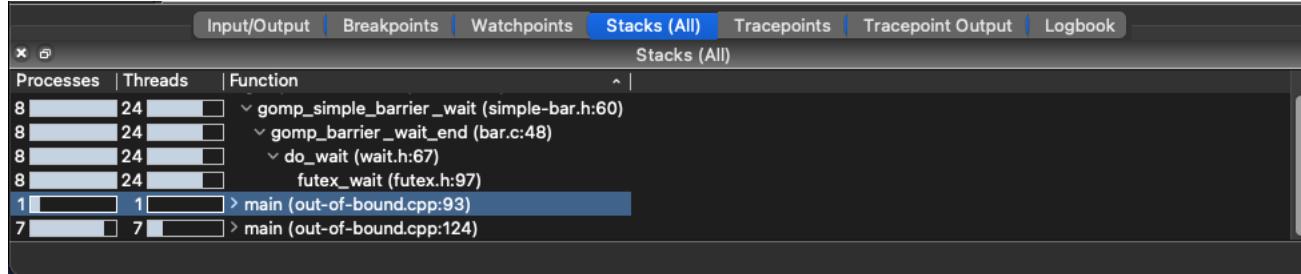
Multidimensional array viewer



3D plot of numerical values

Display numerical values

Deadlocks



1 process is at line 93

All 0 1 2 3 4 5 6 7

```
91  
92  
93 MPI_Gatherv( a, n local,MPI_DOUBL  
94
```

Process 0 is in a collective at line 93

All 0 1 2 3 4 5 6 7

```
122  
123  
124 MPI_Finalize();  
125
```

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



Deadlock !



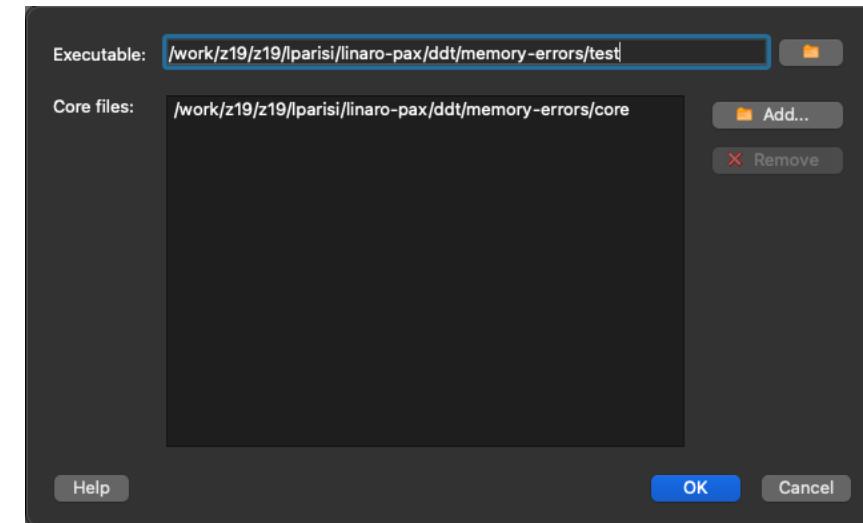
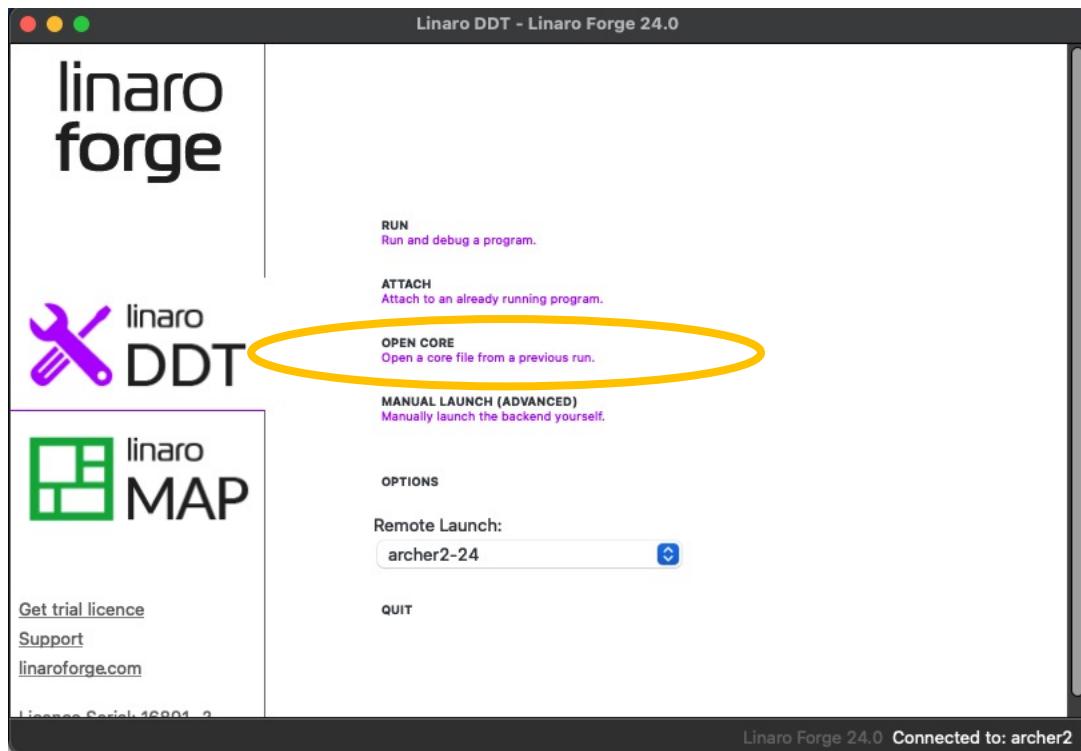
Memory debugging



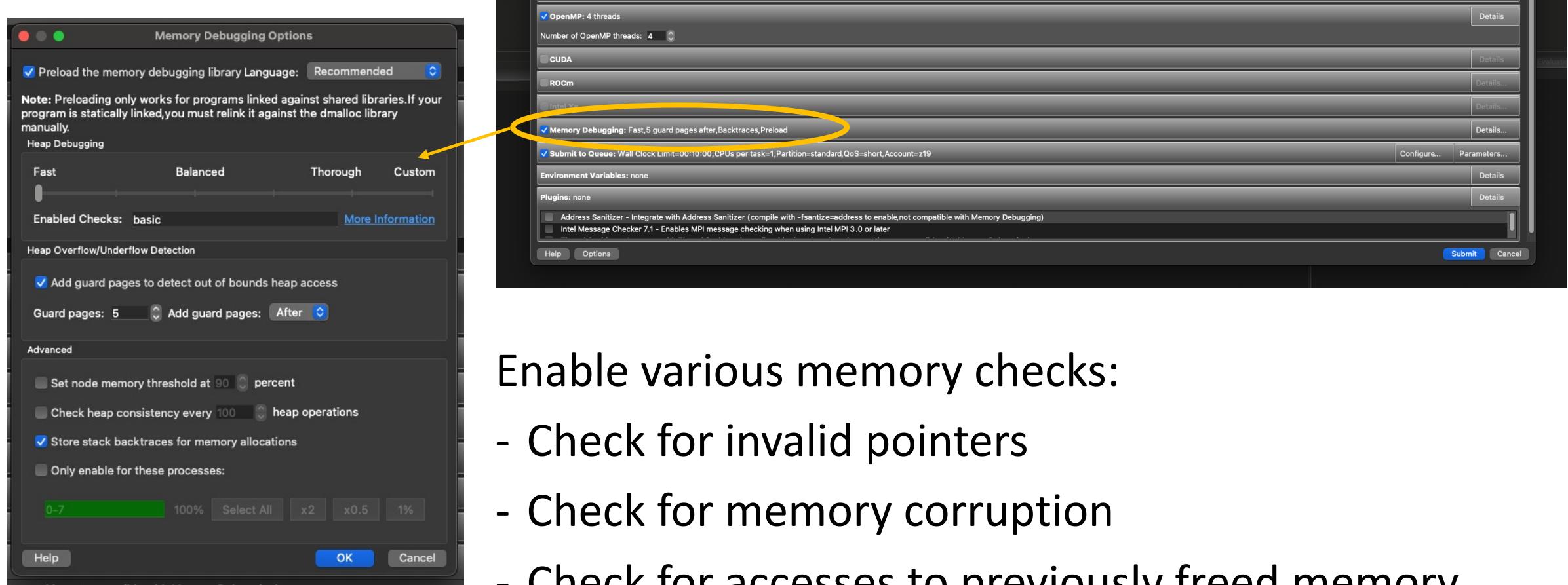
epcc

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 screenshot shows a debugger interface with several windows. At the top, there's a toolbar with buttons for 'All', 'Focus on current: Group', 'Process', 'Thread', and 'Step Threads Together'. Below the toolbar, a row of colored squares represents ranks: 0 (green), 1 (light green), 2 (light blue), 3 (light orange), 4 (pink), 5 (light pink), 6 (light red), and 7 (red). A blue oval highlights the rank 0 square. The main window displays a C++ code editor with code related to MPI operations. An orange oval highlights a specific line of code in the editor:

```
    a[j]=a[i] + b[i,j];
```

A yellow oval encloses a modal dialog box titled 'Processes 4-7' with the message: 'Memory error detected in main (out-of-bound.cpp:78): read/write beyond end of allocation'. It has 'Continue' and 'Pause' buttons.

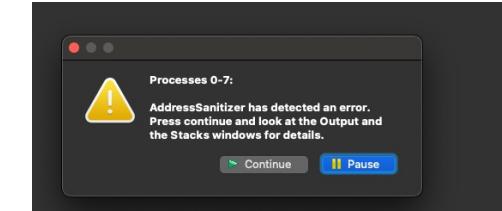
The guilty line

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

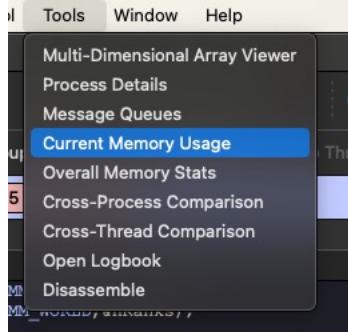
This screenshot shows the Clion IDE interface during an AddressSanitizer run. The code editor displays a C++ file with several loops and memory allocations. A yellow oval highlights a specific line of code: 'a[j] = a[j] + b[j];'. The stack browser (bottom left) shows a call stack starting with 'main (out-of-bound.cpp:78)'. The output browser (bottom right) shows the raw memory dump of the heap buffer overwriting.

This screenshot shows the Clion IDE's stack browser window. It displays a call stack for process 8, starting with 'main (out-of-bound.cpp:78)' and then '_asan::__asan_report_load8 (asan_rtl.cpp:121)'. This indicates that the application has triggered anasan's reporting mechanism.

This screenshot shows the Clion IDE's output browser window. It displays the raw memory dump of the heap buffer overwriting, showing the sequence of bytes being written to memory. The dump starts with '0x0132e0xb0000000c8' and continues with a series of hex values.

Details
of the
error

Memory management



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

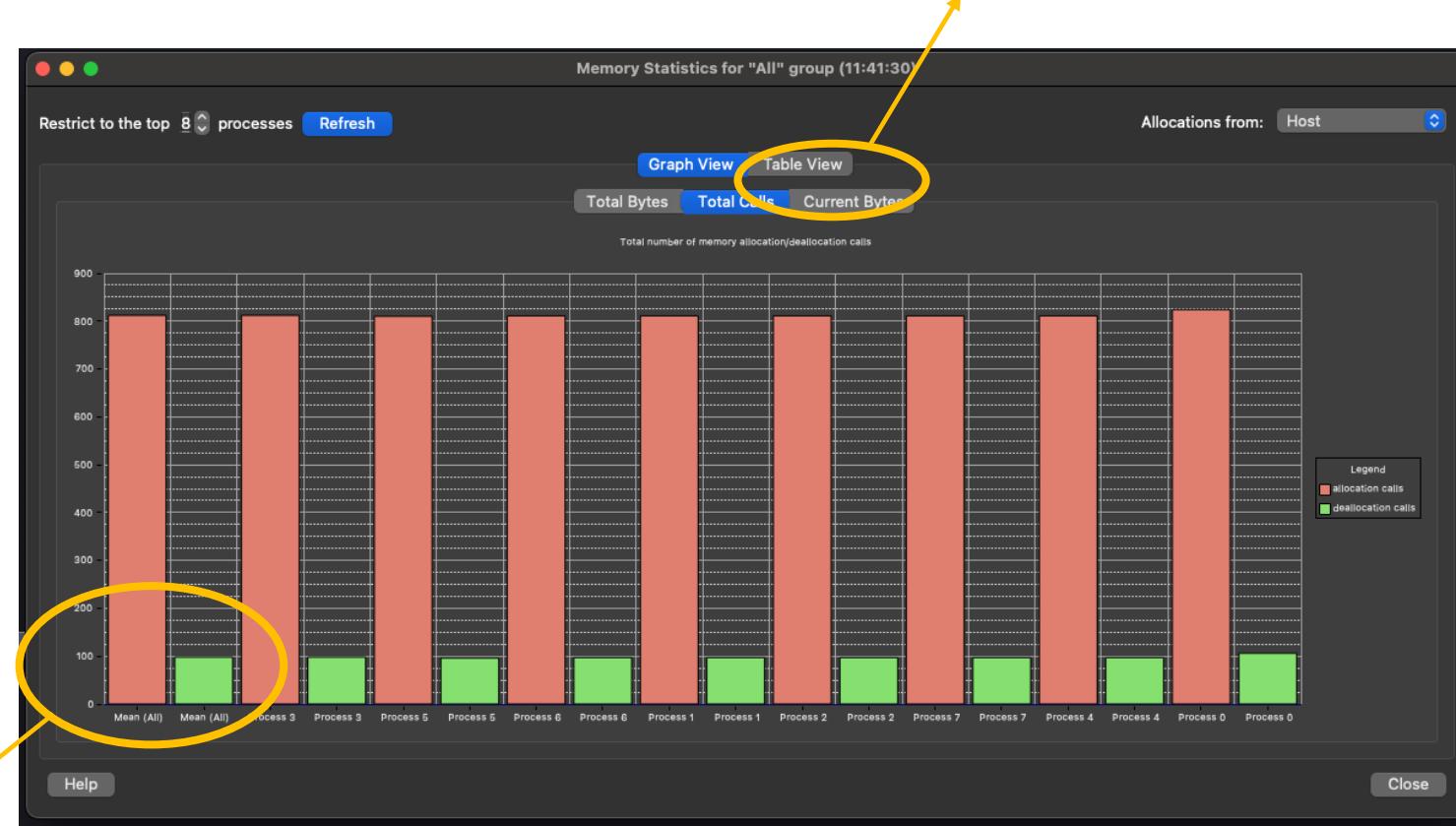
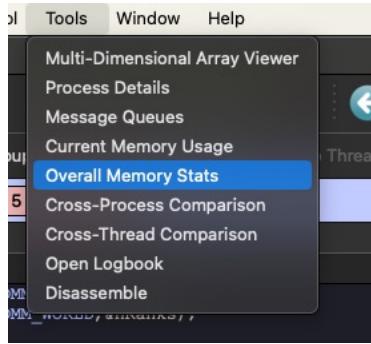


Information
about memory
allocations

Histogram of memory
usage across ranks

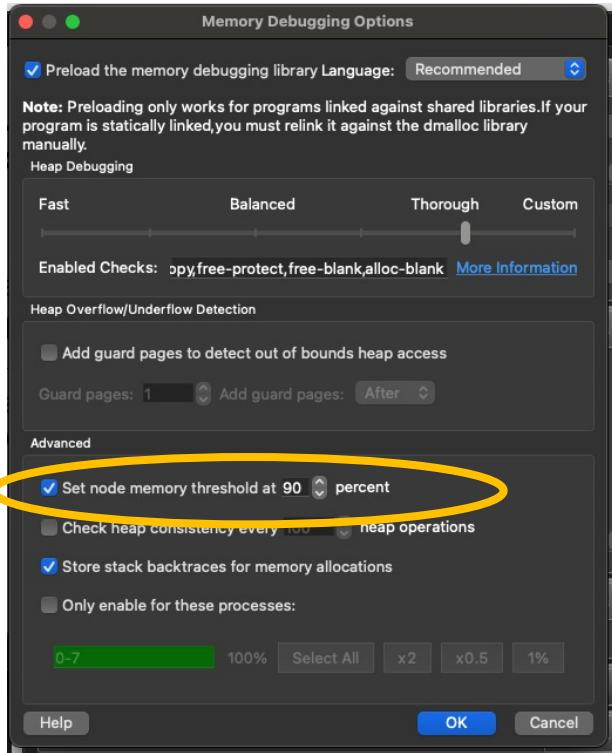
Memory management

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

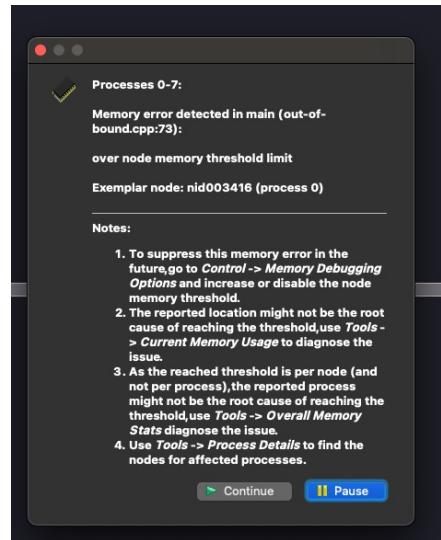


Growing number of memory allocations, but not deallocations suggests a memory leak

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

#	Type	Time	Processes	Message																														
1	info	0:00.000	n/a	Launching srun --hint=nomultithread --distribution=block:block ./test at Wed May 29 14:19:25 2024																														
2	info	0:07.249	0-7	Startup complete.																														
3	info	0:07.249	n/a	Select process group All																														
4	warn	0:07.250	0-7	Add breakpoint for out-of-bound.cpp:73																														
5	info	0:07.315	n/a	No debug symbols were loaded for the glibc library. It is recommended you install the glibc debug symbols.																														
6	info	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	play	0:10.113	0-7	Play																														
8	stop	0:10.266	0-7	Process stopped at breakpoint in main (out-of-bound.cpp:73).																														
9				Additional Information																														
				▼ Stacks																														
				<table border="1"><thead><tr><th>Processes</th><th>Function</th><th>Source</th></tr></thead><tbody><tr><td>0-7</td><td>main (out-of-bound.cpp:73)</td><td>b = new double [n_local];</td></tr></tbody></table>	Processes	Function	Source	0-7	main (out-of-bound.cpp:73)	b = new double [n_local];																								
Processes	Function	Source																																
0-7	main (out-of-bound.cpp:73)	b = new double [n_local];																																
				▼ Rank 0, thread 1																														
				<table border="1"><thead><tr><th>Name</th><th>Value</th></tr></thead><tbody><tr><td>a</td><td>0x1549749bf000 (from 0x1487c33f9000 to 0x1549749bf000)</td></tr><tr><td>argc</td><td><optimized out></td></tr><tr><td>argv</td><td><optimized out></td></tr><tr><td>b</td><td><optimized out></td></tr><tr><td>c</td><td><optimized out></td></tr><tr><td>displacements</td><td>std::vector of length 8, capacity 8</td></tr><tr><td>iRep</td><td>0</td></tr><tr><td>local_sizes</td><td>std::vector of length 8, capacity 8</td></tr><tr><td>n</td><td>10000000</td></tr><tr><td>nRanks</td><td>8</td></tr><tr><td>n_local</td><td>12500000</td></tr><tr><td>nrep</td><td>10</td></tr><tr><td>offset_local</td><td>0 (from 0 to 87500000)</td></tr><tr><td>rank</td><td>0 (from 0 to 7)</td></tr></tbody></table>	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	10000000	nRanks	8	n_local	12500000	nrep	10	offset_local	0 (from 0 to 87500000)	rank	0 (from 0 to 7)
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	10000000																																	
nRanks	8																																	
n_local	12500000																																	
nrep	10																																	
offset_local	0 (from 0 to 87500000)																																	
rank	0 (from 0 to 7)																																	
				▼ Current Stack																														
				#0 main (argc=<optimized out>, argv=<optimized out>) at /work/z19/z19/lparisi/linaro-pax/ddt/memory-errors/test/out-of-bound.cpp:73 (at 0x401250)																														
10	play	0:13.000	0-7	Play																														

Debugger encountered a breakpoint

Stack trace when breakpoint was reached

Resume execution

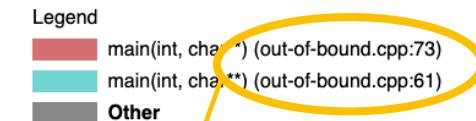
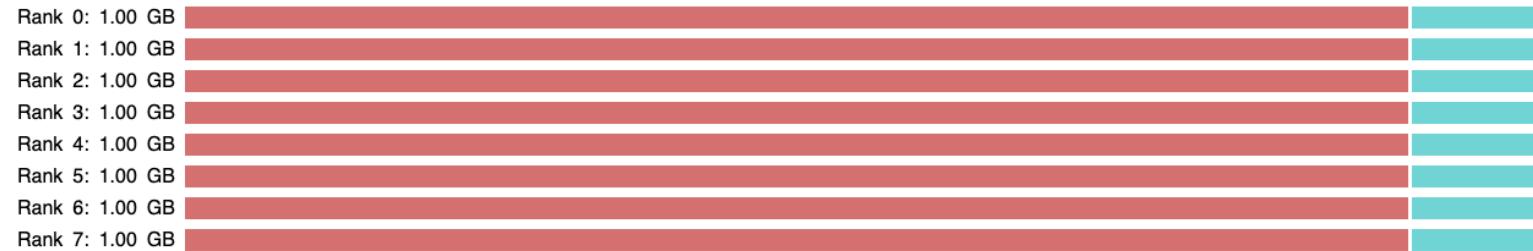
Memory leaks -- offline

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

Memory Leak Report

This report shows unfreed memory allocations when the program finished executing. Clicking an item in the bar chart below will show additional details about the allocations, including where they were allocated.

All 8 ranks:



Allocation data can also be [exported to CSV format](#).

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