

A man with a beard is looking at a tablet. The background is a blurred office setting with a grid overlay. The word 'arm' is written in white lowercase letters on the left side of the image.

arm

Debugging and Profiling with DDT and Map

SDL Workshop

October 3, 2018

Ryan Hulguin

ryan.hulguin@arm.com

Agenda

- General Debugging and Profiling Advice
- Arm Software for Debugging and Profiling
- Debugging with DDT
- Profiling with MAP
- Theta Specific Settings

Debugging

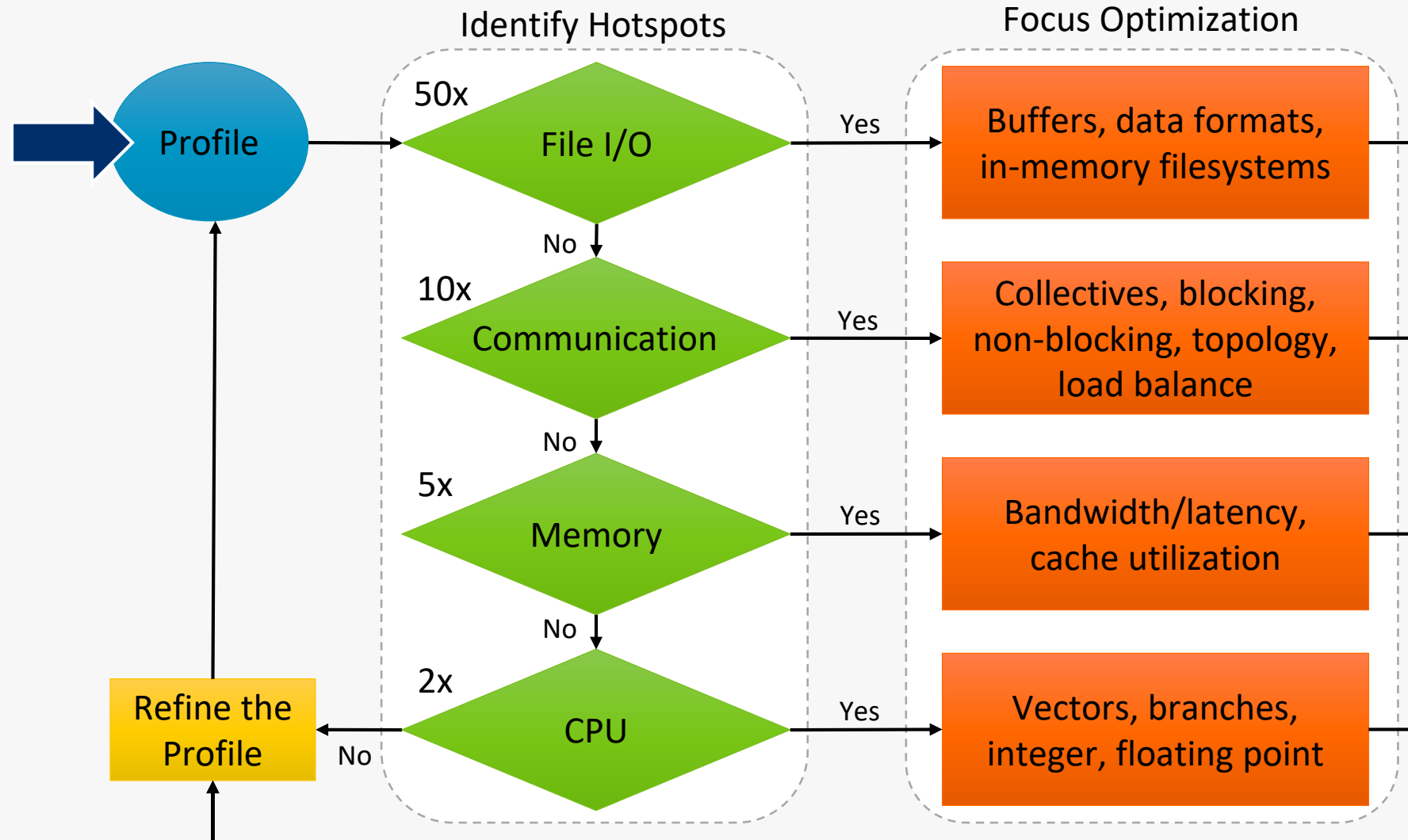
Transforming a broken program to a working one

How? TRAFFIC!

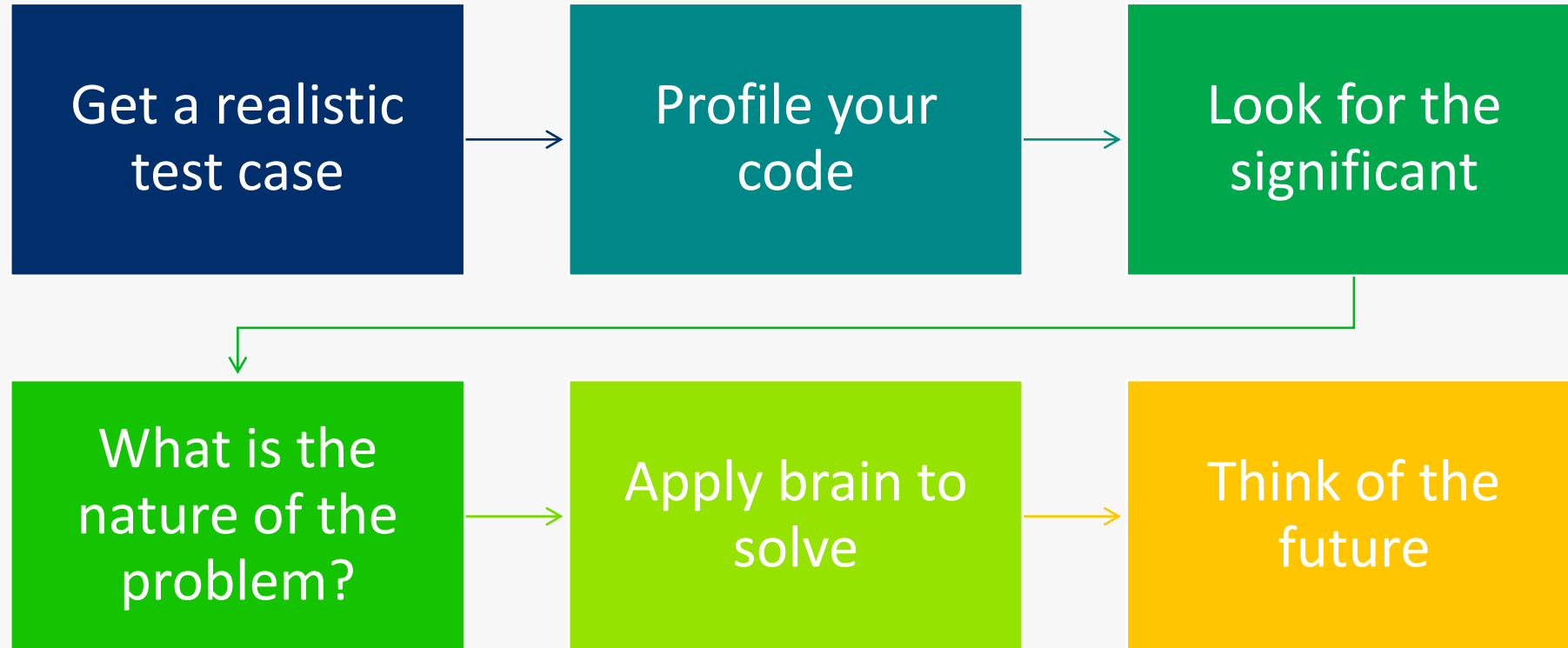
- Track the problem
- Reproduce
- Automate - (and simplify) the test case
- Find origins – where could the “infection” be from?
- Focus – examine the origins
- Isolate – narrow down the origins
- Correct – fix and verify the test case is successful

Profiling

Profiling is central to understanding and improving application performance.



Performance Improvement Workflow



Arm Software

Arm Forge

An interoperable toolkit for debugging and profiling



The de-facto standard for HPC development

- Available on the vast majority of the Top500 machines in the world
- Fully supported by Arm on x86, IBM Power, Nvidia GPUs, etc.



State-of-the art debugging and profiling capabilities

- Powerful and in-depth error detection mechanisms (including memory debugging)
- Sampling-based profiler to identify and understand bottlenecks
- Available at any scale (from serial to parallel applications running at petascale)



Easy to use by everyone

- Unique capabilities to simplify remote interactive sessions
- Innovative approach to present quintessential information to users

Arm Performance Reports

Characterize and understand the performance of HPC application runs



Commercially supported
by Arm



Accurate and astute
insight



Relevant advice
to avoid pitfalls

Gathers a rich set of data

- Analyses metrics around CPU, memory, IO, hardware counters, etc.
- Possibility for users to add their own metrics

Build a culture of application performance & efficiency awareness

- Analyses data and reports the information that matters to users
- Provides simple guidance to help improve workloads' efficiency

Adds value to typical users' workflows

- Define application behaviour and performance expectations
- Integrate outputs to various systems for validation (e.g. continuous integration)
- Can be automated completely (no user intervention)

Run and ensure application correctness

Combination of debugging and re-compilation

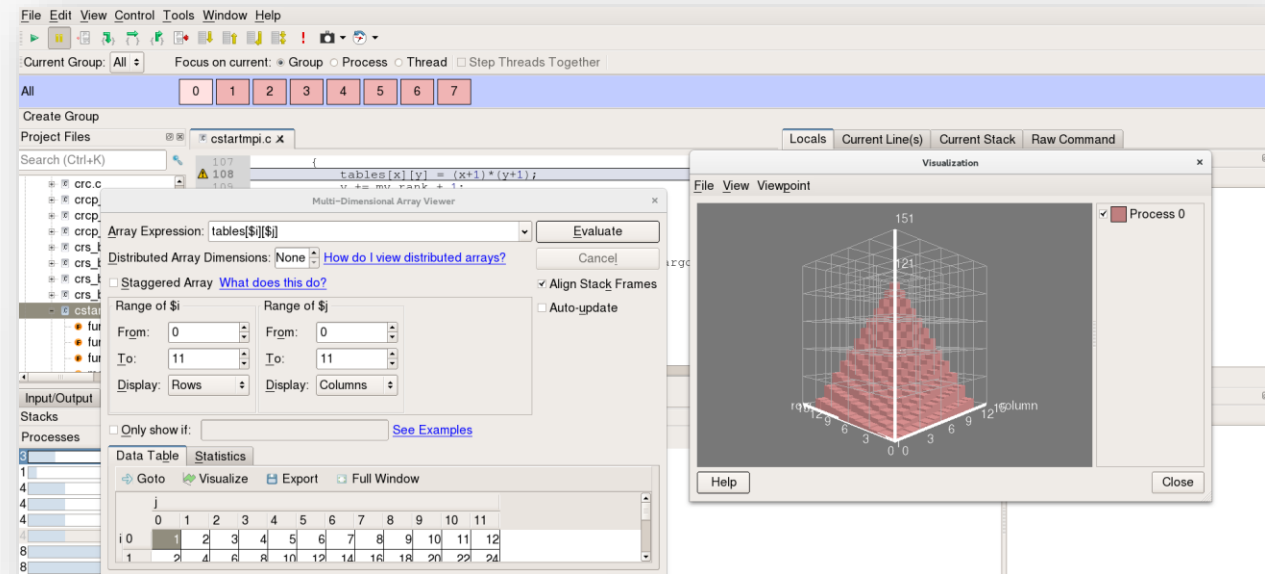
- Ensure application correctness with
- Integrate with continuous integration system.
- Use version control to track changes and leverage Forge's built-in VCS support.

Examples:

```
$> ddt --offline aprun -n 48 ./example
```

```
$> ddt --connect aprun -n 48 ./example
```

Step	Time	Process	Message
15	2:17.256	0-7	Play
16	2:18.048	4-7	Process stopped at breakpoint in main (cpi.c:50).
17			Additional Information Stacks Processes: Function 4-7 main (cpi.c:50)
18	2:19.048	n/a	Select process 4
19			Additional Information Current Stack Locals
9	2:17.832	main (cpi.c:46)	0-7 done: 0 i: from 65 to 72 numprocs: 8 myid: from 0 to 7 n: 100
10	2:17.832	main (cpi.c:46)	0-7 done: 0 i: from 73 to 80 numprocs: 8 myid: from 0 to 7 n: 100
11	2:18.323	main (cpi.c:46)	0-7 done: 0 i: from 81 to 88 numprocs: 8 myid: from 0 to 7 n: 100
12	2:18.323	main (cpi.c:46)	0-7 done: 0 i: from 89 to 96 numprocs: 8 myid: from 0 to 7 n: 100
13	2:18.325	main (cpi.c:46)	0-3 done: 0 i: from 97 to 100 numprocs: 8 myid: from 0 to 3 n: 100



Understand application behaviour

Is it performant?

Set a reference for future work

- Choose a representative test case with known behavior
- Analyse performance with **Arm Performance Reports**

Example:

```
$> perf-report aprun -n 16 mmult_c.exe
```

```
Command: mpiexec ./mmult_c.exe 7168
Resources: 1 node (28 physical, 56 logical cores per node)
Memory: 125 GiB per node
Tasks: 28 processes
Machine: r4163
Start time: Wed May 17 2017 10:25:58 (UTC+10)
Total time: 33 seconds
Full path: /short/c25/pw9396/allinea_wshop-day1/0_charac_performance
```

Summary: mmult_c.exe is **Compute-bound** in this configuration

Category	Percentage	Color	Notes
Compute	62.8%	Green	Time spent running application code. High values are usually good. This is average ; check the CPU performance section for advice.
MPI	24.6%	Blue	Time spent in MPI calls. High values are usually bad. This is low ; this code may benefit from a higher process count.
I/O	12.6%	Orange	Time spent in filesystem I/O. High values are usually bad. This is low ; check the I/O breakdown section for optimization advice.

This application run was **Compute-bound**. A breakdown of this time and advice for investigating further is in the **CPU** section below.

As little time is spent in **MPI** calls, this code may also benefit from running at larger scales.

CPU

A breakdown of the 62.8% CPU time:

Scalar numeric ops	0.2%
Vector numeric ops	13.4%
Memory accesses	80.3%

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

MPI

A breakdown of the 24.6% MPI time:

Time in collective calls	6.3%
Time in point-to-point calls	93.7%
Effective process collective rate	0.00 bytes/s
Effective process point-to-point rate	114 MB/s

Most of the time is spent in **point-to-point** calls with an average transfer rate. Using larger messages and overlapping communication and computation may increase the effective transfer rate.

Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	448 MiB
Peak process memory usage	1.24 GiB
Peak node memory usage	16.0%

There is **significant variation** between peak and mean memory usage. This may be a sign of workload imbalance or a memory leak.

The peak node memory usage is very low. Running with fewer MPI processes and more data on each process may be more efficient.

I/O

A breakdown of the 12.6% I/O time:

Time in reads	0.0%
Time in writes	100.0%
Effective process read rate	0.00 bytes/s
Effective process write rate	3.56 MB/s

Most of the time is spent in **write operations** with a very low effective transfer rate. This may be caused by contention for the filesystem or inefficient access patterns. Use an I/O profiler to investigate which write calls are affected.

Threads

A breakdown of how multiple threads were used:

Computation	0.0%
Synchronization	0.0%
Physical core utilization	99.7%
System load	101.8%

No measurable time is spent in multithreaded code.

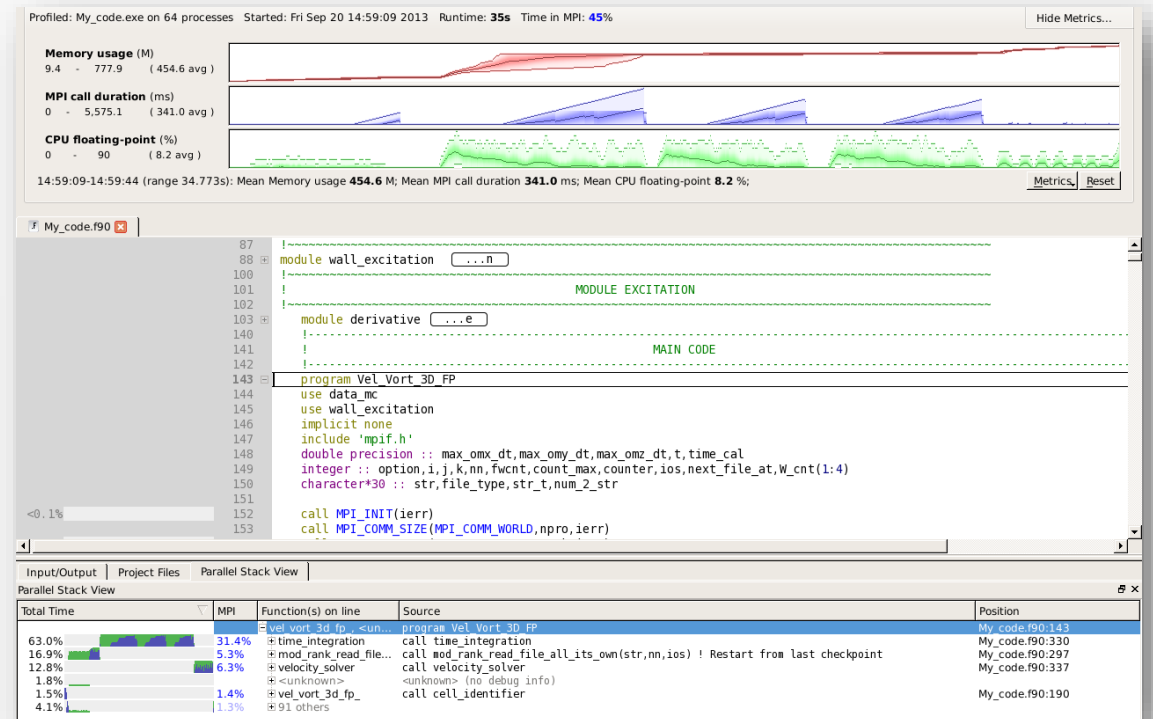
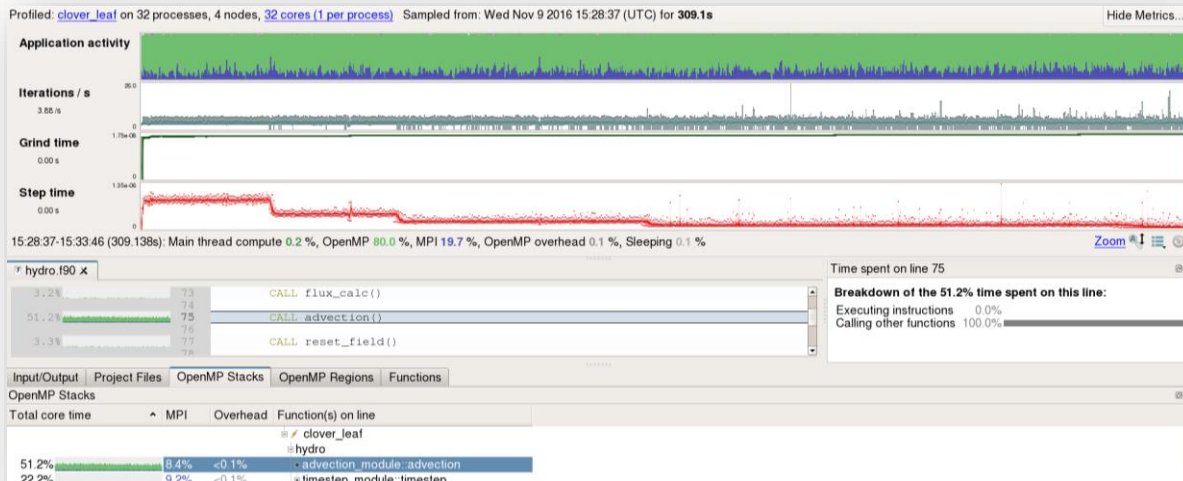
Optimize the application for Arm

if not, use the Arm MAP profiler for optimization

- Measure all performance aspects with
- Identify bottlenecks and rewrite some code for better performance

Examples:

```
$> map --profile aprun -n 48 ./example
```



Debugging with DDT

Arm DDT – The Debugger

Who had a rogue behaviour ?

- Merges stacks from processes and threads

Where did it happen?

- leaps to source

How did it happen?

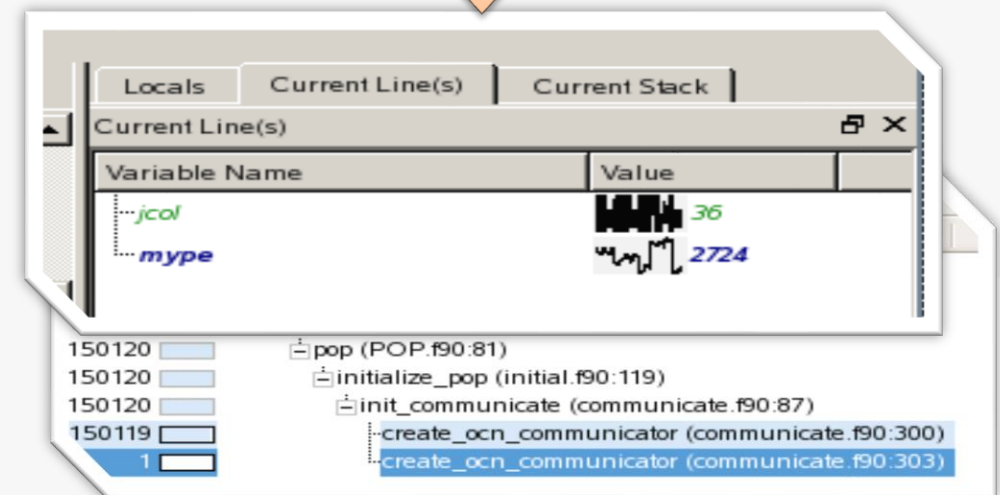
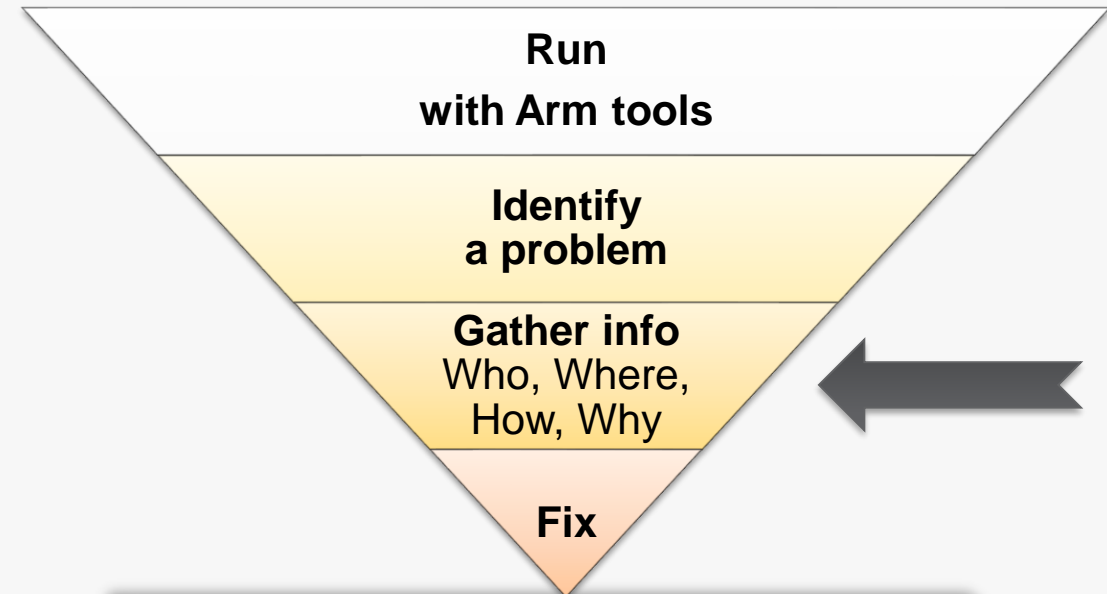
- Diagnostic messages

- Some faults evident instantly from source

Why did it happen?

- Unique “Smart Highlighting”

- Sparklines comparing data across processes



Preparing Code for Use with DDT

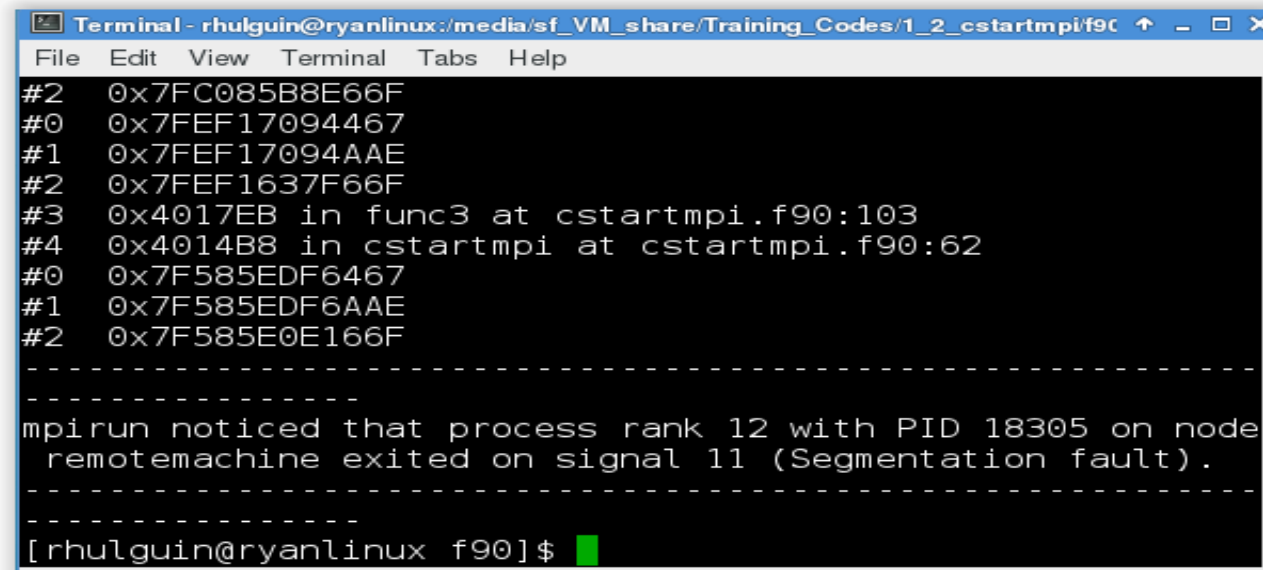
As with any debugger, code must be compiled with the debug flag typically `-g`

It is recommended to turn off optimization flags i.e. `-O0`

Leaving optimizations turned on can cause the compiler to *optimize out* some variables and even functions making it more difficult to debug

Segmentation Fault

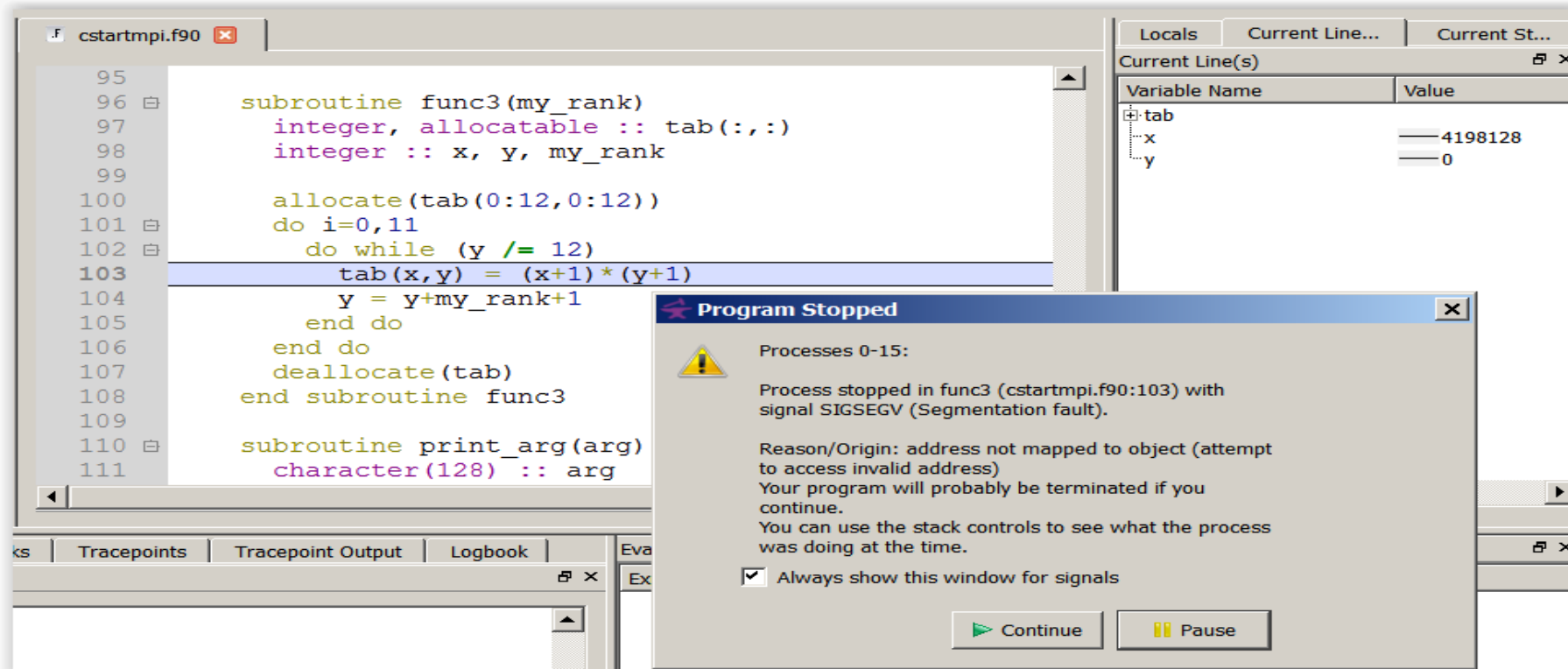
In this example, the application crashes with a segmentation error outside of DDT.



```
Terminal - rhulguin@ryanlinux:/media/sf_VM_share/Training_Codes/1_2_cstartmpi/f90
File Edit View Terminal Tabs Help
#2 0x7FC085B8E66F
#0 0x7FEF17094467
#1 0x7FEF17094AAE
#2 0x7FEF1637F66F
#3 0x4017EB in func3 at cstartmpi.f90:103
#4 0x4014B8 in cstartmpi at cstartmpi.f90:62
#0 0x7F585EDF6467
#1 0x7F585EDF6AAE
#2 0x7F585E0E166F
-----
mpirun noticed that process rank 12 with PID 18305 on node
remotemachine exited on signal 11 (Segmentation fault).
-----
[rhulguin@ryanlinux f90]$
```

What happens when it runs under DDT?

Segmentation Fault in DDT



DDT takes you to the exact line where Segmentation fault occurred, and you can pause and investigate

Invalid Memory Access

```
196  subroutine func3(my_rank)
197      integer, allocatable :: tab(:, :)
198      integer :: x, y, my_rank
199
200      allocate(tab(0:12, 0:12))
201      do i=0, 11
202          do while (y /= 12)
203              tab(x, y) = (x+1) * (y+1)
204              y = y + my_rank + 1
205          end do
206      end do
207      deallocate(tab)
208  end subroutine func3
209
210  subroutine main
211      character(12) :: name
```

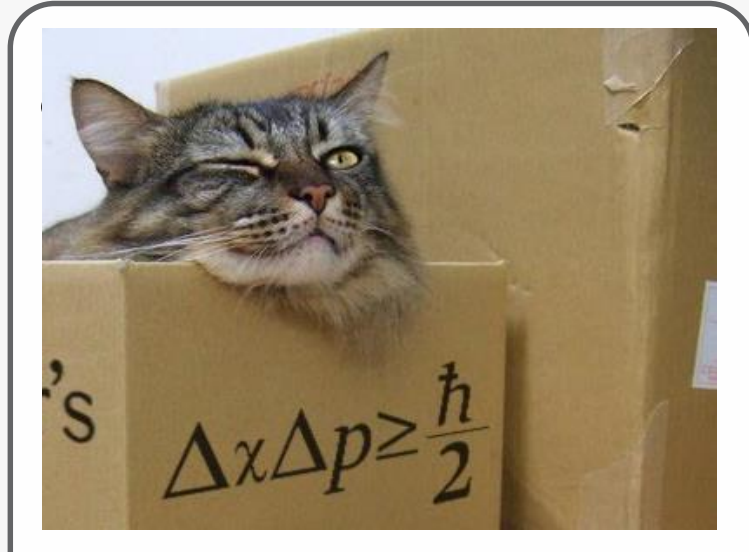
On this line:
16 Processes: ranks 0-15
1 Thread (Rank 0): #1
Name: tab
Type: integer(kind=4), ALLOCATABLE
(0:12, 0:12)

Variable Name	Value
tab	[[0] = ([0] = -158
x	4198128
y	0

The array `tab` is a 13x13 array, but the application is trying to write a value to `tab(4198128,0)` which causes the segmentation fault.

`i` is not used, and `x` and `y` are not initialized

It works... Well, most of the time



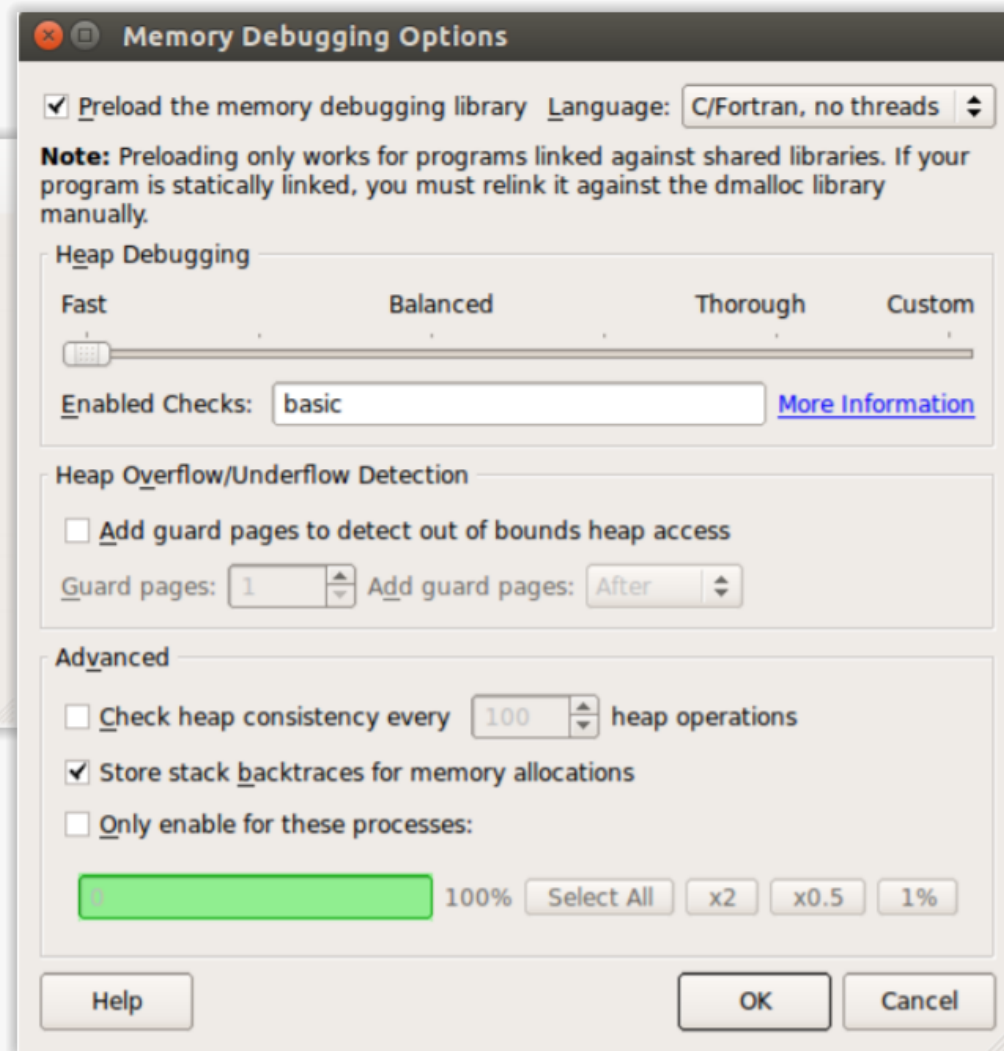
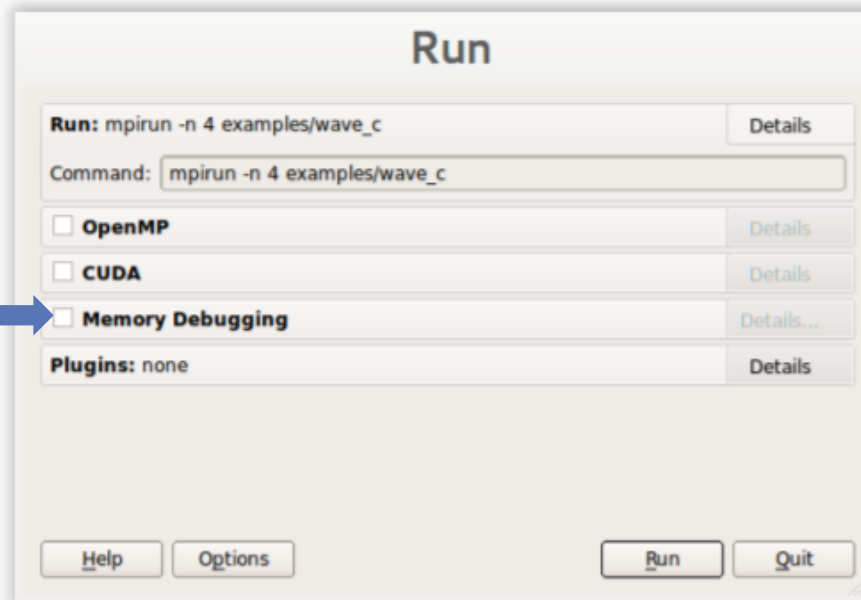
**SCHRODIN
BUG**



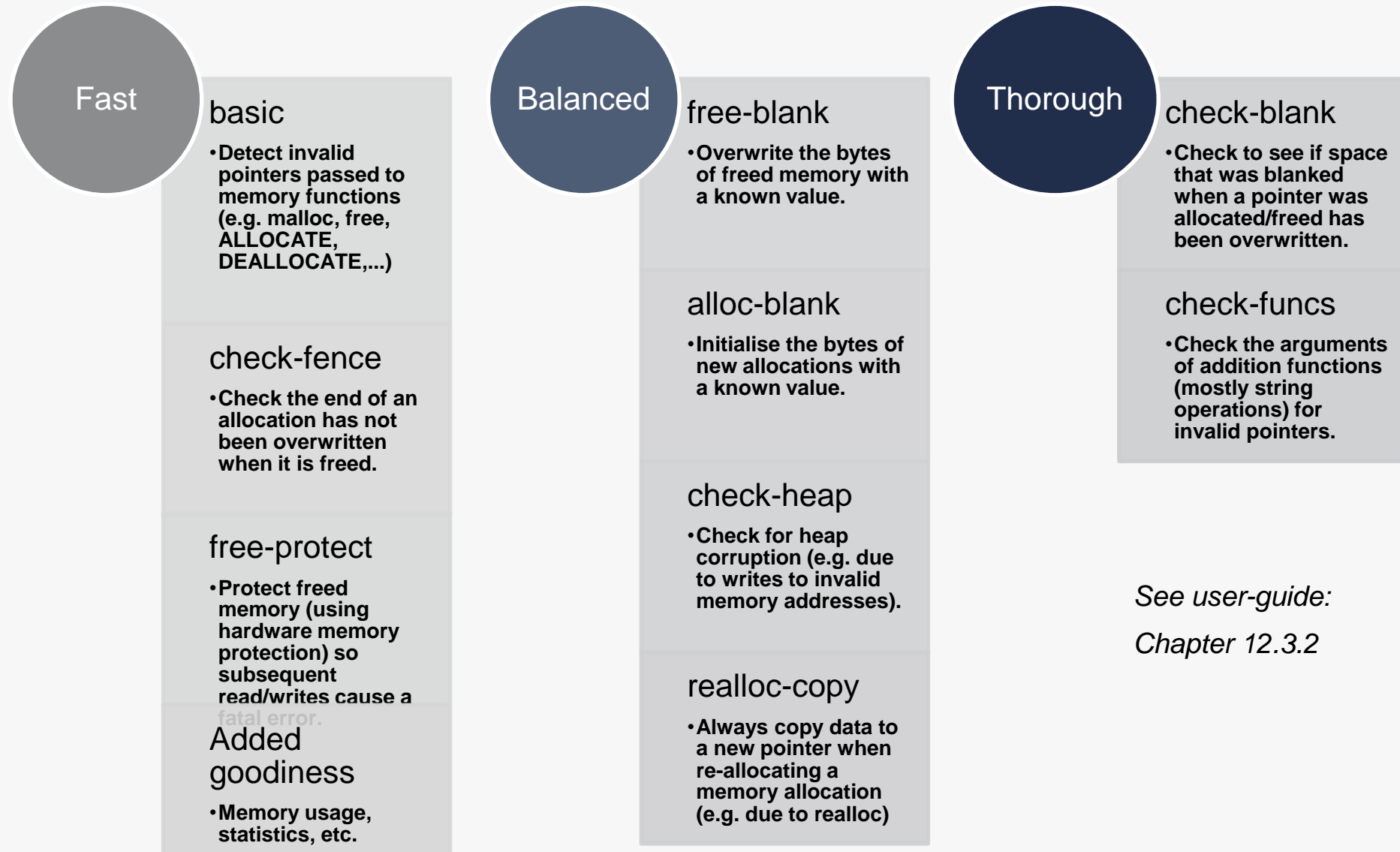
A strange behaviour where the application “sometimes” crashes is a typical sign of a memory bug

Arm DDT is able to force the crash to happen

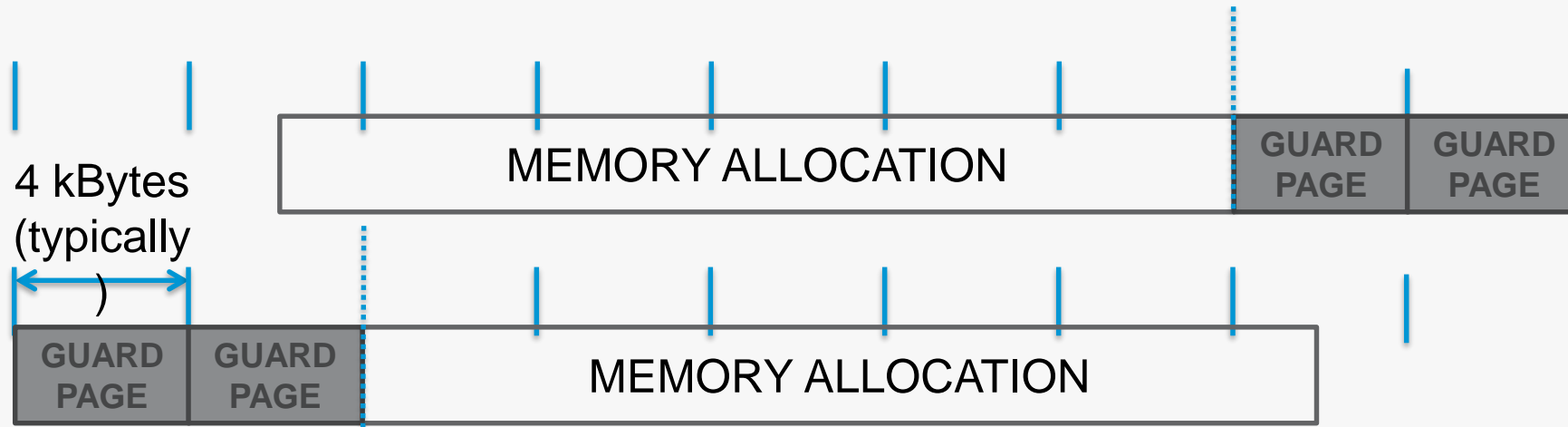
Advanced Memory Debugging



Heap debugging options available

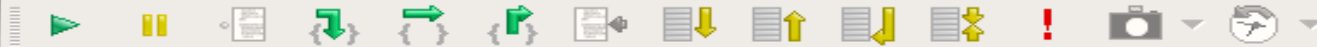


Guard pages (aka “Electric Fences”)



- **A powerful feature....:**
 - Forbids read/write on guard pages throughout the whole execution
(because it overrides C Standard Memory Management library)
- **... to be used carefully:**
 - Kernel limitation: up to 32k guard pages max (“mprotect fails” error)
 - Beware the additional memory usage cost

File View Control Search Tools Window Help

Current Group: All Focus on current: Group Process Thread Step Threads TogetherAll 24576 processes (0-24575) Paused: 17223 Playing: 7353 Finished: 0
Currently selected: 260 (on nid00194, pid 9481, main thread IWP 9481)

Create Group

Project Files

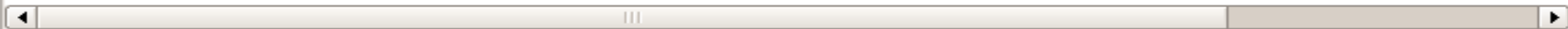
Search (Ctrl+K)

- VolumeTrav
- wave.c
- weird.c
- WholeGeom
- Writer.cc
- wspace.c
- XdrFileWrite
- XdrMemRea
- XdrMemWrit
- XdrReader.c
- XdrWriter.cc
- XmlAbstract
- xyzpart.c
- External Code



Input/Output Breakpoints Watchpoints Stacks Tracepoints Tracepoint Output Logbook

Processes	Threads	Function
17223	17223	main (main.cc:37)
17223	17223	SimulationMaster::SimulationMaster (SimulationMaster.cc:63)
17223	17223	SimulationMaster::Initialise (SimulationMaster.cc:154)
17223	17223	hemelb::geometry::GeometryReader::LoadAndDecompose (GeometryReader.cc:188)
17223	17223	hemelb::geometry::GeometryReader::OptimiseDomainDecomposition (GeometryReader.c
17223	17223	hemelb::geometry::decomposition::OptimisedDecomposition::OptimisedDecomposition
17223	17223	hemelb::geometry::decomposition::OptimisedDecomposition::CallParmetis (Optimise
17223	17223	ParMETIS_V3_PartGeomKway (gkmetis.c:90)
17223	17223	libparmetis_Coordinate_Partition (xyzpart.c:58)
17223	17223	libparmetis_PseudoSampleSort (xyzpart.c:556)



Computer

titan-ext7

Alinea DDT 4.2.1-36484



Sun Aug 10, 7:50 PM

```
551 ikvsortii(ntsamples, allpicks);
552
553
554 /* Select the final splitters. Set the boundaries to s
555 for (i=1; i<npes; i++)
556     mypicks[i] = allpicks[i*ntsamples/npes];
557 mypicks[0].key = IDX_MIN;
558 mypicks[npes].key = IDX_MAX;
559
560
561 WCOREPOP; /* free allpicks */
562
563 STOPTIMER(ctrl, ctrl->AuxTmr2);
564 STARTTIMER(ctrl, ctrl->AuxTmr3);
565
```

Locals Current Line(s) Current Stack

Current Line(s)

Variable Name	Value
allpicks	0x2aab8055e010
i	2245
mypicks	0x2a6f8f0
npes	24575
ntsamples	1818550

Type: none selected

Evaluate

Expression	Value
i * ntsamples	-212322546

Type: int
Range: from -2147259746 to -12282046
49/17223 processes equal

7353 processes playing

New Bugs from Latest Changes

The screenshot shows a debugger interface for a C program named `wave_openmp.c`. The main window displays the source code with a red highlight on line 227, which contains the assignment `oldval = values;`. The 'Locals' panel on the right shows the current state of local variables: `oldval` at memory address `0x7ffff4b7a010` and `values` at `0x7ffff4b7a010`. The 'Stacks' panel at the bottom left shows the call stack with the following entries:

Threads	Function
1	main (wave_openmp.c:354)
1	update (wave_openmp.c:227)
3	omp_in_final

The 'Evaluate' panel at the bottom right shows the values of the variables `newval`, `oldval`, and `values`, all pointing to the same memory address `0x7ffff4b7a010`.

Track Your Changes in a Logbook

The screenshot displays the Allinea DDT - Allinea Forge 7.0 [Trial Version] interface. The main window shows the source code for `cstartmpi.c` with the following lines highlighted:

```
91 MPI_Init(&argc, &argv);
92 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
93 MPI_Comm_size(MPI_COMM_WORLD, &p);
94
95
96
97
98 dynamicArray = malloc(sizeof(int)*100000);
```

The Logbook window is open, showing a list of events:

Time	Ranks	Message
0:00	0-3	Launching program /home/bpaisley/demo/ddt/cstartmpi/cstartmpi.exe at Wed Mar 1 10:59:59 2017 Executable modified on Tue Feb 21 10:53:10 2017
0:05	0-3	Startup complete.
0:05	n/a	Select process group All
0:05	0-3	Add tracepoint for cstartmpi.c:113 Vars: x, y
0:05	0-3	Add breakpoint for cstartmpi.c:102
0:05	0-3	Add breakpoint for cstartmpi.c:171
0:05	n/a	Add Expression to Evaluate: my_rank
0:28	0-3	Step Over
0:28	0-3	Process stopped.

The Stack window shows the following variables:

Variable	Value
argc	1
argv	0x7ffffffcea8
beingWatched	0
bigArray	
dest	0
dynamicArray	0x0
environ	0x7ffffffceb8
i	0

The Evaluate window is empty.

caption

Inspect AVX Registers

The screenshot displays the Arm DDT - Arm Forge 18.2 [Trial Version] interface. The main window shows the source code for `mmatest1.c` with a `for` loop. The disassembly window shows the corresponding assembly instructions, including `leaq`, `vmovaps`, `xor`, `mov`, `vfmadd231ps`, `add`, `cmp`, and `jb`. The locals window shows the current state of local variables: `j` is 0, `k` is 4096, `pA` is 0x7fffebbd8040, `pB` is 0x7fffe7bd7040, and `pC` is 0x7fffe3bd6040. The evaluate window shows the value of `$xmm18` as a `v16_float` vector: `{[0] = 4096, [1] = 0, [2] = 2048, [3] = 0, [4] = 0, [5] = 0, [6] = 0, [7] = 0, [8] = 0, [9] = 0, [10] = 0, [11] = 0, [12] = 0, [13] = 0, [14] = 0, [15] = 0}`.

```
for( j = 0; j < n; j += 1 )
{
    sum = 0.0;
    for( k = 0; k < p; k += 1 )
    {
        sum += pA[p*i+k] * pB[n*j+k];
    }
}
```

Disassembly:

Address	Offset	Bytes	Instruction
0x4016a2	<-+562>	4d 8d 14 0f	leaq (%r15,%rcx,1),%r10
0x4016a6	<-+566>	4d 8d 0c 0e	leaq (%r14,%rcx,1),%r9
0x4016aa	<-+570>	49 89 d8	mov %rbx,%r8
0x4016ad	<-+573>	62 a1 7c 48 28 d0	vmovaps %xmm16,%xmm18
0x4016b3	<-+579>	33 ff	xor %edi,%edi
0x4016b5	<-+581>	4d 89 c5	mov %r8,%r13
0x4016b8	<-+584>	62 b1 7c 48 28 c2	vmovaps %xmm18,%xmm0
0x4016be	<-+590>	62 e1 7c 48 28 c8	vmovaps %xmm0,%xmm17
0x4016c4	<-+596>	0f 1f 44 00 00	nopl 0x0(%rax,%rax,1)
0x4016c9	<-+601>	0f 1f 80 00 00 00 00	nopl 0x0(%rax)
0x4016d0	<-+608>	62 c1 7c 48 10 1c ba	vmovups (%r10,%rdi,4),%xmm19
0x4016d7	<-+615>	62 c1 7c 48 10 64 ba 01	vmovups 0x40(%r10,%rdi,4),%xmm20
0x4016df	<-+623>	62 d2 65 40 b8 85 00 40 00 00	vfmadd231ps 0x4000(%r13),%xmm19,%xmm0
0x4016e9	<-+633>	62 c1 7c 48 10 6c ba 02	vmovups 0x80(%r10,%rdi,4),%xmm21
0x4016f1	<-+641>	62 d2 5d 40 b8 85 40 40 00 00	vfmadd231ps 0x4040(%r13),%xmm20,%xmm0
0x4016fb	<-+651>	62 c2 65 40 b8 55 00 00	vfmadd231ps 0x0(%r13),%xmm19,%xmm18
0x401702	<-+658>	62 c2 5d 40 b8 4d 01 00 00	vfmadd231ps 0x40(%r13),%xmm20,%xmm17
0x401709	<-+665>	62 d2 55 40 b8 85 80 40 00 00	vfmadd231ps 0x4080(%r13),%xmm21,%xmm0
0x401713	<-+675>	62 c1 7c 48 10 74 ba 03	vmovups 0xc0(%r10,%rdi,4),%xmm22
0x40171b	<-+683>	48 83 c7 40	add \$0x40,%rdi
0x40171f	<-+687>	62 c2 55 40 b8 55 02 00 00	vfmadd231ps 0x80(%r13),%xmm21,%xmm18
0x401726	<-+694>	62 c2 4d 40 b8 4d 03 00 00	vfmadd231ps 0xc0(%r13),%xmm22,%xmm17
0x40172d	<-+701>	62 d2 4d 40 b8 85 c0 40 00 00	vfmadd231ps 0x40c0(%r13),%xmm22,%xmm0
0x401737	<-+711>	49 81 c5 00 01 00 00 00	add \$0x100,%r13
0x40173e	<-+718>	48 81 ff 00 10 00 00 00	cmp \$0x1000,%rdi
0x401745	<-+725>	72 89	jb 0x4016d0<main+608>
0x401747	<-+727>	49 81 c0 00 80 00 00 00	add \$0x8000,%r8

Locals:

Variable Name	Value
j	0
k	4096
pA	0x7fffebbd8040
pB	0x7fffe7bd7040
pC	0x7fffe3bd6040

Evaluate:

Expression	Value
\$xmm18	{[0] = 4096, [1] = 0, [2] = 2048, [3] = 0, [4] = 0, [5] = 0, [6] = 0, [7] = 0, [8] = 0, [9] = 0, [10] = 0, [11] = 0, [12] = 0, [13] = 0, [14] = 0, [15] = 0}
v8_double	{[0] = 5.760887010628376e-315, [1] = 5.7194417803360855e-315, [2] = 0, [3] = 0, [4] = 0, [5] = 0, [6] = 0, [7] = 0}
v64_int8	{[0] = 0, [1] = 0, [2] = -128, [3] = 69, [4] = 0, [5] = 0, [6] = 0, [7] = 0, [8] = 0, [9] = 0, [10] = 0, [11] = 69, [12] = 0, [13] = 0, [14] = 0, [15] = 0, [16] = 0, [17] = 0, [18] = 0, [19] = 0, [20] = 0, [21] = 0, [22] = 0, [23] = 0, [24] = 0, [25] = 0, [26] = 0, [27] = 0, [28] = 0, [29] = 0, [30] = 0, [31] = 0, [32] = 0, [33] = 0, [34] = 0, [35] = 0, [36] = 0, [37] = 0, [38] = 0, [39] = 0, [40] = 0, [41] = 0, [42] = 0, [43] = 0, [44] = 0, [45] = 0, [46] = 0, [47] = 0, [48] = 0, [49] = 0, [50] = 0, [51] = 0, [52] = 0, [53] = 0, [54] = 0, [55] = 0, [56] = 0, [57] = 0, [58] = 0, [59] = 0, [60] = 0, [61] = 0, [62] = 0, [63] = 0}
v32_int16	{[0] = 0, [1] = 17792, [2] = 0, [3] = 0, [4] = 0, [5] = 17664, [6] = 0, [7] = 0, [8] = 0, [9] = 0, [10] = 0, [11] = 0, [12] = 0, [13] = 0, [14] = 0, [15] = 0, [16] = 0, [17] = 0, [18] = 0, [19] = 0, [20] = 0, [21] = 0, [22] = 0, [23] = 0, [24] = 0, [25] = 0, [26] = 0, [27] = 0, [28] = 0, [29] = 0, [30] = 0, [31] = 0, [32] = 0, [33] = 0, [34] = 0, [35] = 0, [36] = 0, [37] = 0, [38] = 0, [39] = 0, [40] = 0, [41] = 0, [42] = 0, [43] = 0, [44] = 0, [45] = 0, [46] = 0, [47] = 0, [48] = 0, [49] = 0, [50] = 0, [51] = 0, [52] = 0, [53] = 0, [54] = 0, [55] = 0, [56] = 0, [57] = 0, [58] = 0, [59] = 0, [60] = 0, [61] = 0, [62] = 0, [63] = 0}
v16_int32	{[0] = 1166016512, [1] = 0, [2] = 1157627904, [3] = 0, [4] = 0, [5] = 0, [6] = 0, [7] = 0, [8] = 0, [9] = 0, [10] = 0, [11] = 0, [12] = 0, [13] = 0, [14] = 0, [15] = 0}
v8_int64	{[0] = 1166016512, [1] = 1157627904, [2] = 0, [3] = 0, [4] = 0, [5] = 0, [6] = 0, [7] = 0}
v4_int128	{[0] = 0x000000004500, [1] = 0x00, [2] = 0x00, [3] = 0}

Arm DDT Demo

Five great things to try with Allinea DDT

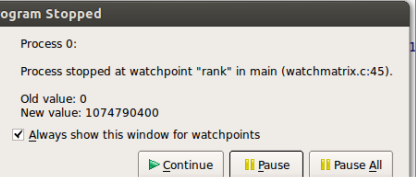
Tracepoint	Processes	Values logged
vhone #90 85	976, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol 2-83 mod pey
vhone #90 81	960, ranks 12,14-17,22-23,12...	ks 1 kmax pec
vhone #90 85	942, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol 2-83 mod pey
vhone #90 81	920, ranks 12,14-17,22-23,12...	ks 1 kmax pec
vhone #90 85	919, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol 2-83 mod pey
vhone #90 81	898, ranks 12,14-17,22-23,12...	ks 1 kmax pec
vhone #90 85	884, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol 2-83 mod pey
vhone #90 81	880, ranks 12,14-17,22-23,12...	ks 1 kmax pec

The scalable print alternative

```

for (i = 0 ; i < SIZE M; i++)
  for (j = 0 ; j < SIZE N; j++)
    C[i][j] = 0;

for (i = 0 ; i < SIZE M; i++)
  for (j = 0 ; j < SIZE N; j++)
    for (k = 0 ; k < SIZE 0; k++)
      C[i][j] += A[i][k] * B[k][j];
    
```



Stop on variable change

```

43
44     else
45     test=-1;
46   }
47 void func3()
48 {
49     void* i = (void*) 1;
50     while(i++ || !i)
51     free((void*)i);
    
```

portability 'i' is of type 'void *'. When using void pointers in calculations, the pointer must be cast to the appropriate type.

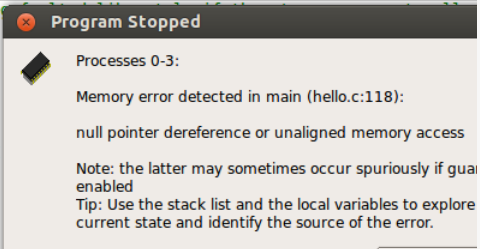
Static analysis warnings on code errors

```

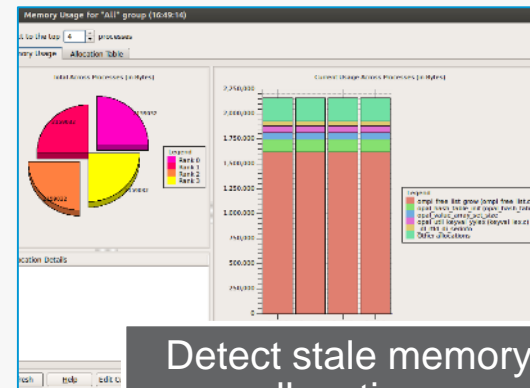
&& !strcmp(argv[i], "crash")) {
0;
s", *(char **)argv[i]);
ll se

r, "I
= 1;

ist.s
= 0;
    
```



Detect read/write beyond array bounds



Detect stale memory allocations

Arm DDT cheat sheet

Load the environment module

- \$ module load **forge/18.2.1**

Prepare the code

- \$ cc **-O0 -g** myapp.c -o myapp.exe

Start Arm DDT in interactive mode





- \$ **ddt** aprun -n 8 ./myapp.exe arg1 arg2

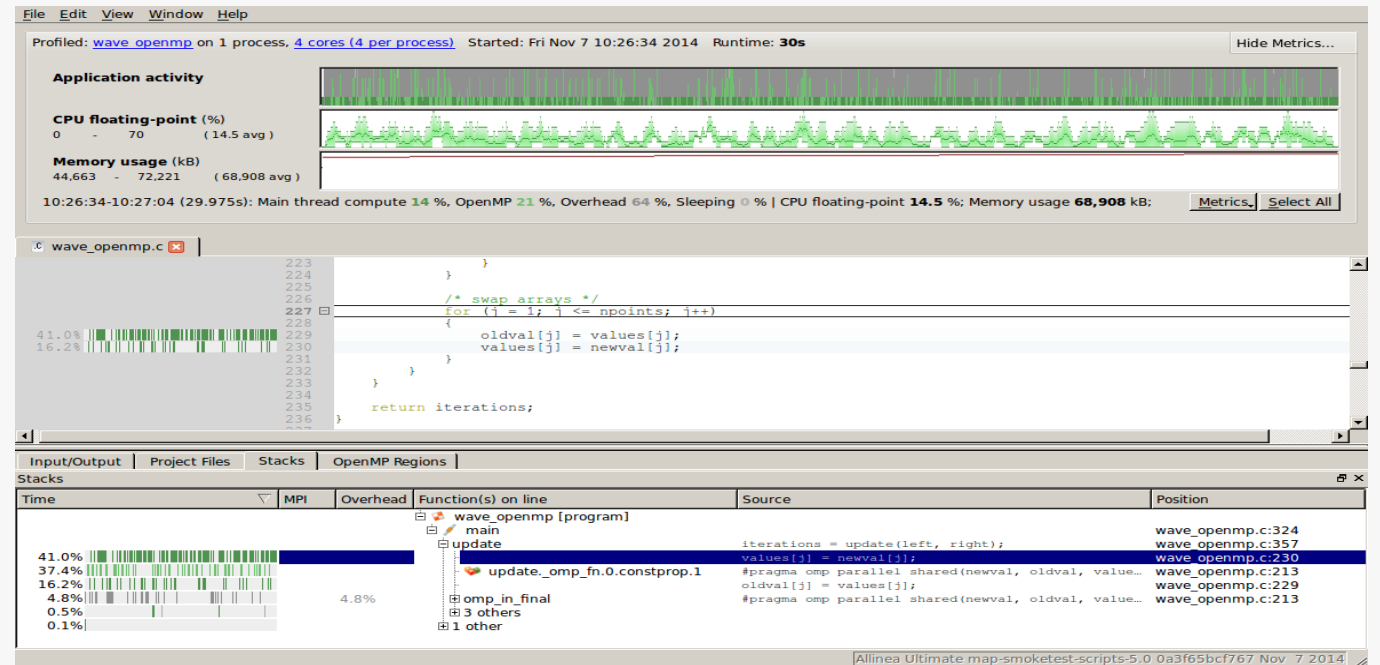
Or use the reverse connect mechanism

- On the login node:
 - \$ ddt &
- (or use the remote client) <- **Preferred method**
- Then, edit the job script to run the following command and submit:
 - **ddt --connect** aprun -n 8 ./myapp.exe arg1 arg2

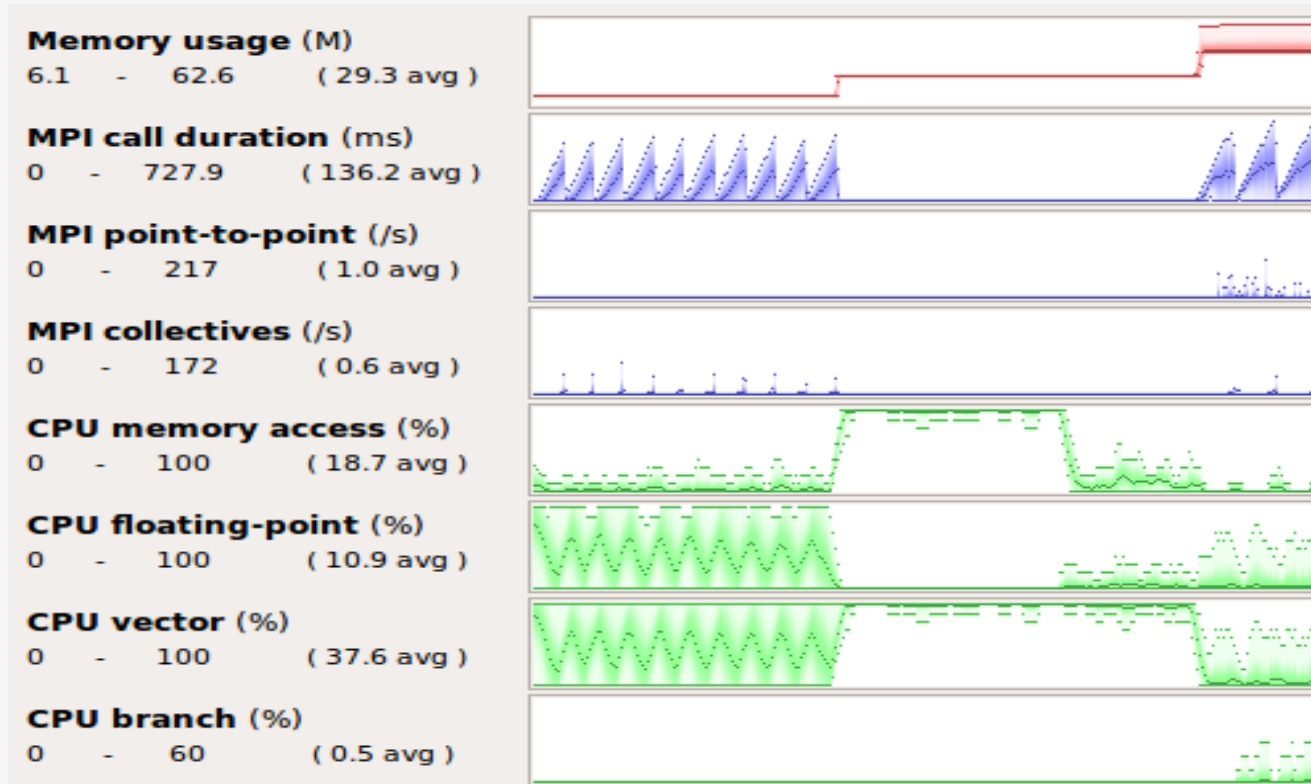
Profiling with MAP

Arm MAP – The Profiler

-  Small data files
-  <5% slowdown
-  No instrumentation
-  No recompilation



Glean Deep Insight from our Source-Level Profiler



Track memory usage across the entire application over time

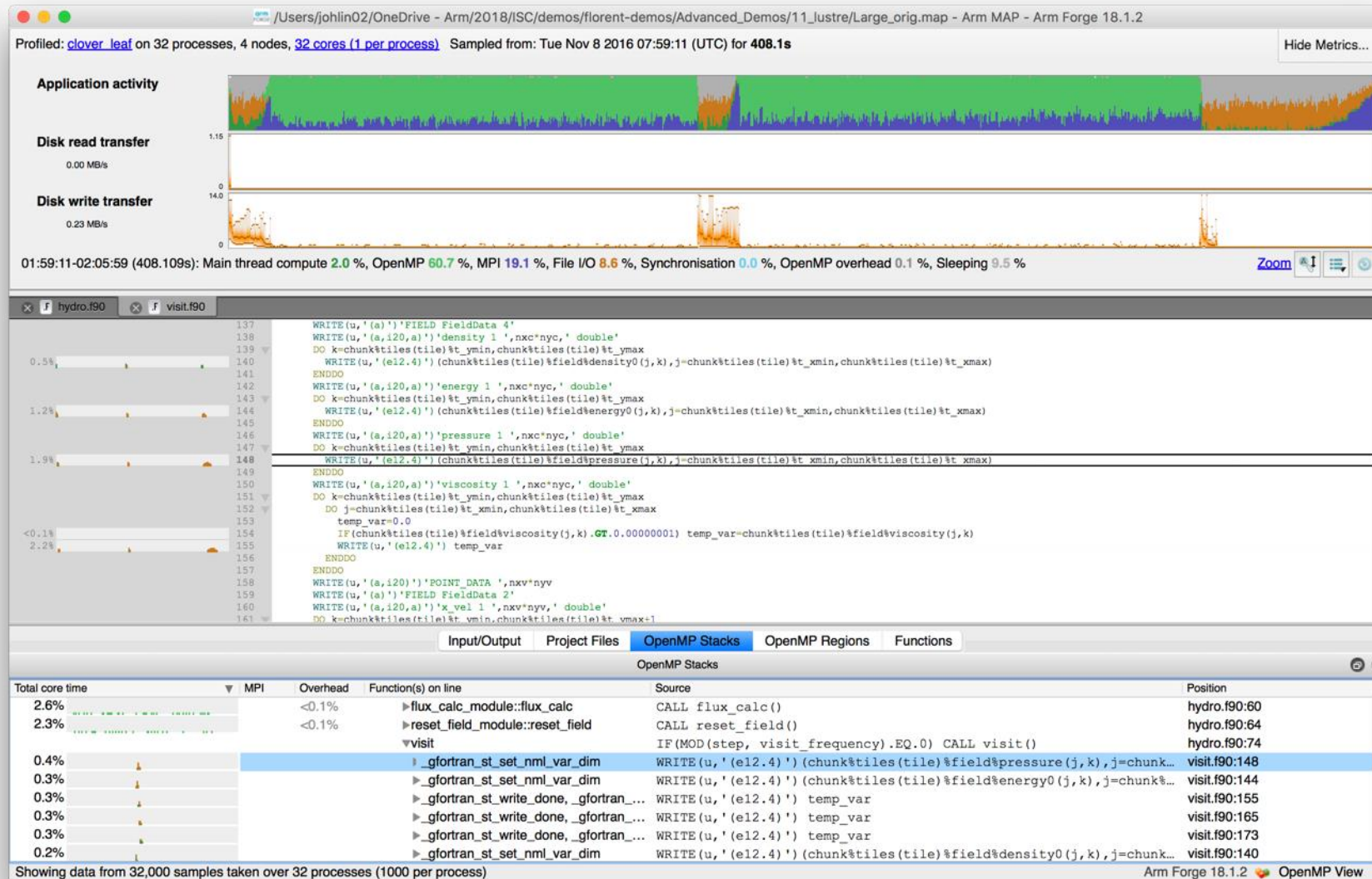
Spot MPI and OpenMP imbalance and overhead

Optimize CPU memory and vectorization in loops

Detect and diagnose I/O bottlenecks at real scale

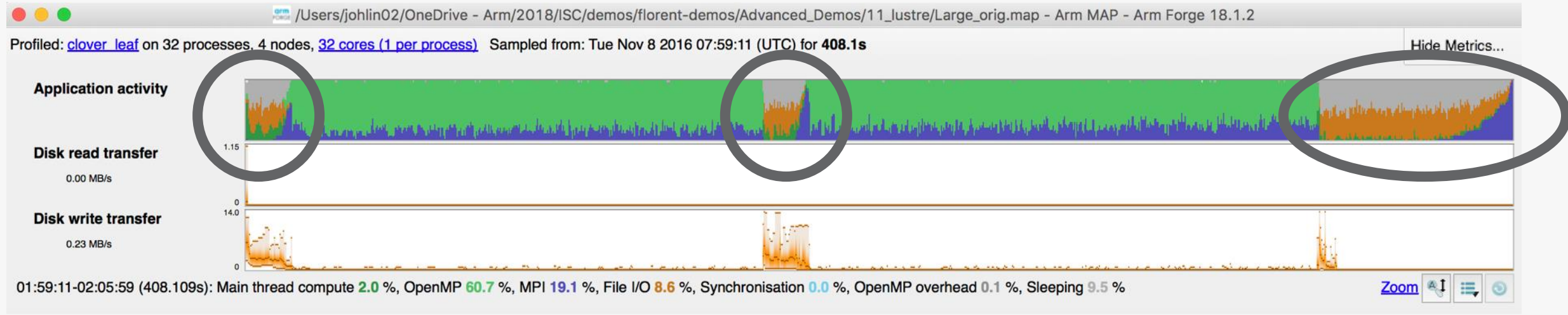
Initial profile of CloverLeaf shows surprisingly unequal I/O

Each I/O operation should take about the same time, but it's not the case.



Symptoms and causes of the I/O issues

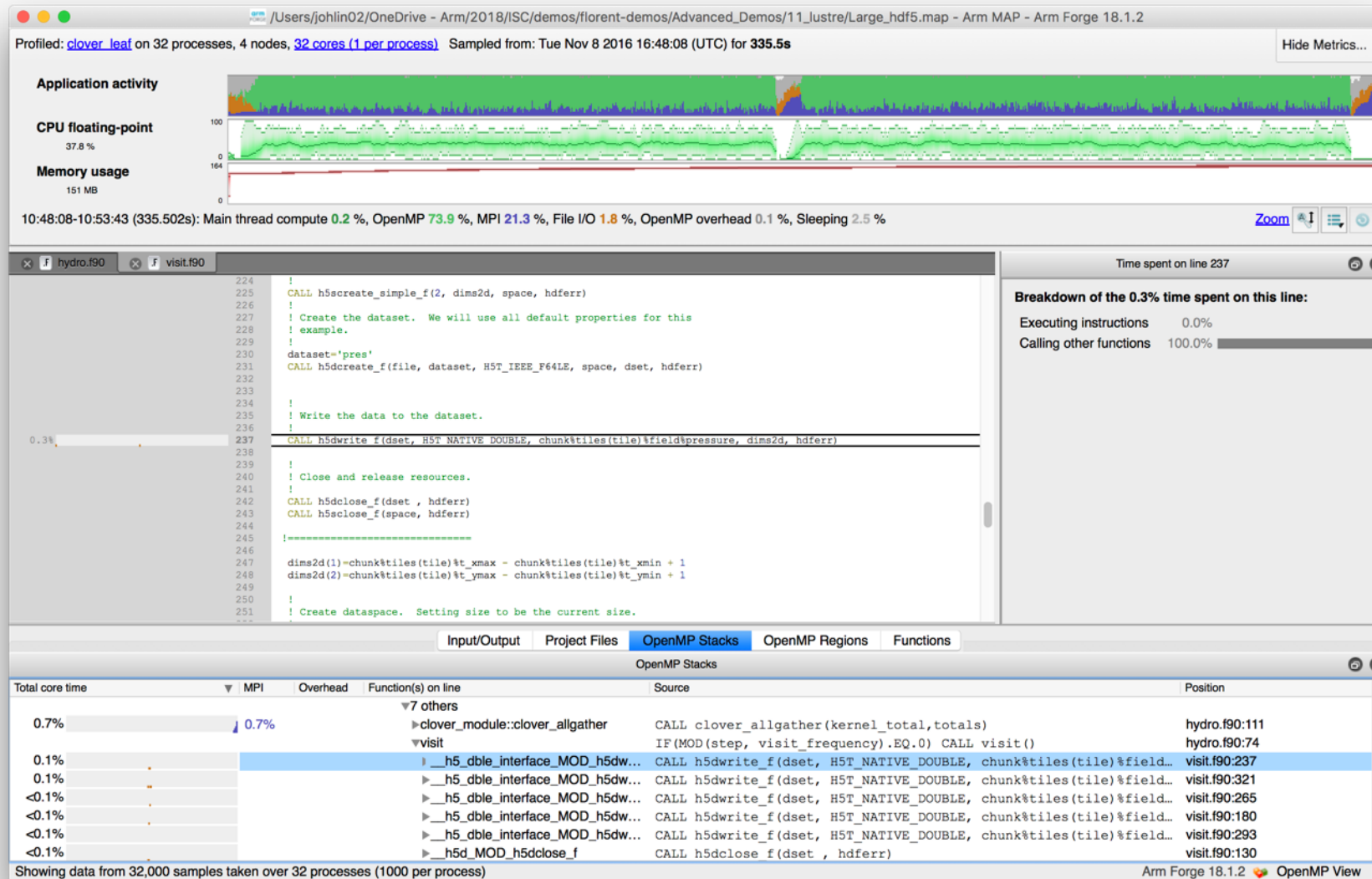
Sub-optimal file format and surprise buffering.



- Write rate is less than 14MB/s.
- Writing an ASCII output file.
- Writes not being flushed until buffer is full.
 - Some ranks have much less buffered data than others.
 - Ranks with small buffers wait in barrier for other ranks to finish flushing their buffers.

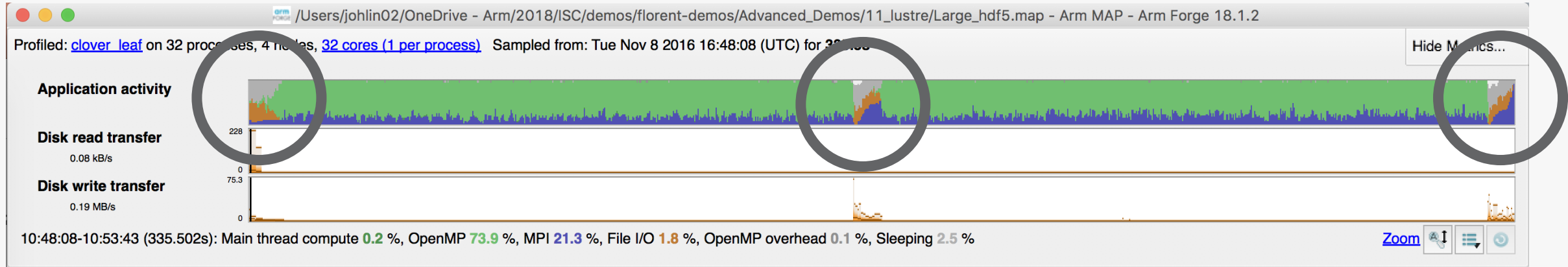
Solution: use HDF5 to write binary files

Using a library optimized for HPC I/O improves performance and portability.



Solution: use HDF5 to write binary files

Using a library optimized for HPC I/O improves performance and portability.



- Replace Fortran write statements with HDF5 library calls.
 - Binary format reduces write volume and can improve data precision.
 - Maximum transfer rate now 75.3 MB/s, over 5x faster.
- Note MPI costs (blue) in the I/O region, so room for improvement.

Arm MAP cheat sheet

Load the environment module (manually specify version)

- \$ module load **forge/18.2.1**

Generate the wrapper libraries (static is default on Theta)

- \$ make-profiler-libraries --lib-type=static

Unload Darshan module (It wraps MPI calls which cannot be used with MAP)

- \$ module unload darshan

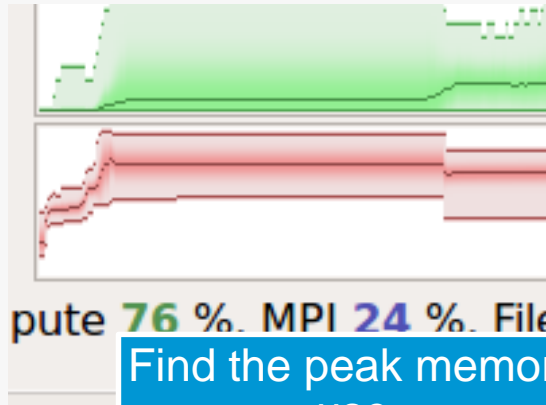
Follow the instructions displayed to prepare the code

- \$ cc -O3 -g myapp.c -o myapp.exe -WI,@/path/to/profiler_wrapper_libraries/allinea-profiler.ld
- Edit the job script to run Arm MAP in “profile” mode
- \$ **map --profile** aprun -n 8 ./myapp.exe arg1 arg2

Open the results

- On the login node:
 - \$ map myapp_Xp_Yn_YYYY-MM-DD_HH-MM.map
- (or load the corresponding file using the remote client connected to the remote system or locally)

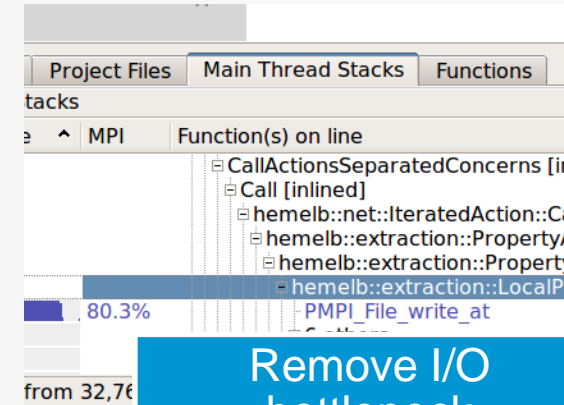
Six Great Things to Try with Allinea MAP



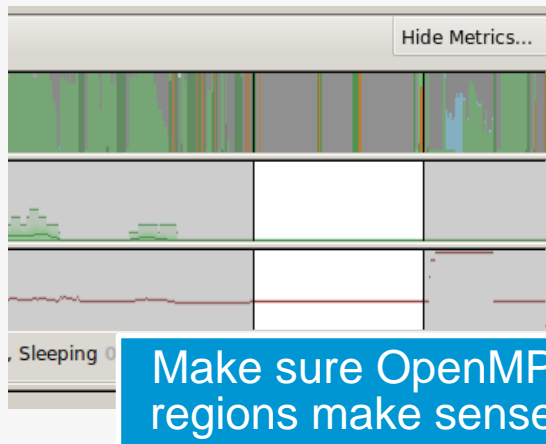
Find the peak memory use

```
30 ! late to the party
31 do j=1,20*nprocs; a
32 end if
33
34 if (pe /= 0) then
35 call MPI_SEND(a, si
36 else
37 do from=1,nprocs-1
38 call MPI_RECV(b,
39 do j=1,50; b=sqrt
40 print *, "Answer f
41 end do
42 end if
43 end do
44 call MPI_BARRIER(MPI CO
```

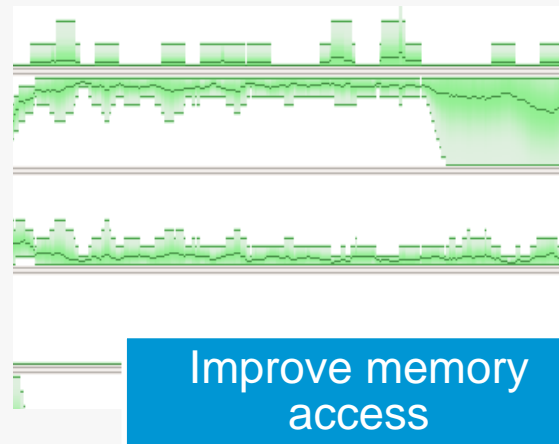
Fix an MPI imbalance



Remove I/O bottleneck



Make sure OpenMP regions make sense



Improve memory access

```
size, nproc, mat a
A[i*size+k]*B[k*s
```

```
nal:--(
(s:
```

Restructure for vectorization

Theta Specific Settings

Configure the remote client

Install the Arm Remote Client

- Go to : <https://developer.arm.com/products/software-development-tools/hpc/downloads/download-arm-forge>

Connect to the cluster with the remote client

- Open your Remote Client
- Create a new connection: Remote Launch → Configure → Add
 - Hostname: <username>@theta.alcf.anl.gov
 - Remote installation directory:
`/soft/debuggers/forge-18.2.1-2018-08-07`
- ALCF Documentation available at
<https://tinyurl.com/debugging-cpw-2018-05>

Static Linking Extra Steps

To enable advanced memory debugging features, you must link explicitly against our memory libraries

Simply add the link flags to your Makefile, or however appropriate

```
lflags = -L/soft/debuggers/ddt/lib/64 -Wl,--undefined=malloc -ldmalloc -Wl,--allow-multiple-definition
```

In order to profile, static profiler libraries must be created with the command `make-profiler-libraries --lib-type=static`

Instructions to link the libraries will be provided after running the above command

Questions?

Thank You!

Danke!

Merci!

谢谢!

ありがとう!

Gracias!

Kiitos!

감사합니다

धन्यवाद

arm