



arm

Debugging and Profiling HPC Applications

ATPESC

August 7, 2019

Agenda

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

Debugging

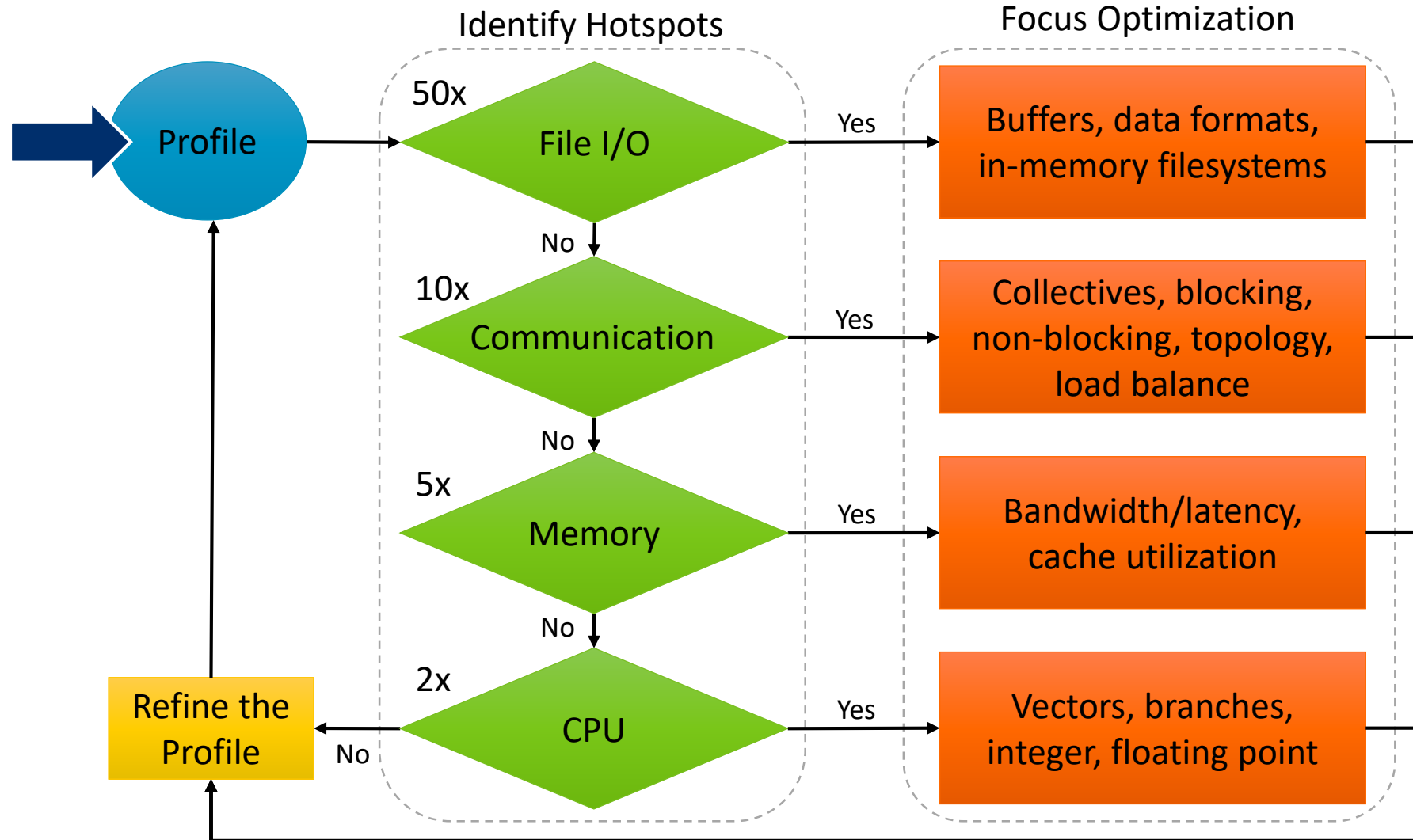
Transforming a broken program to a working one

How? TRAFFIC!

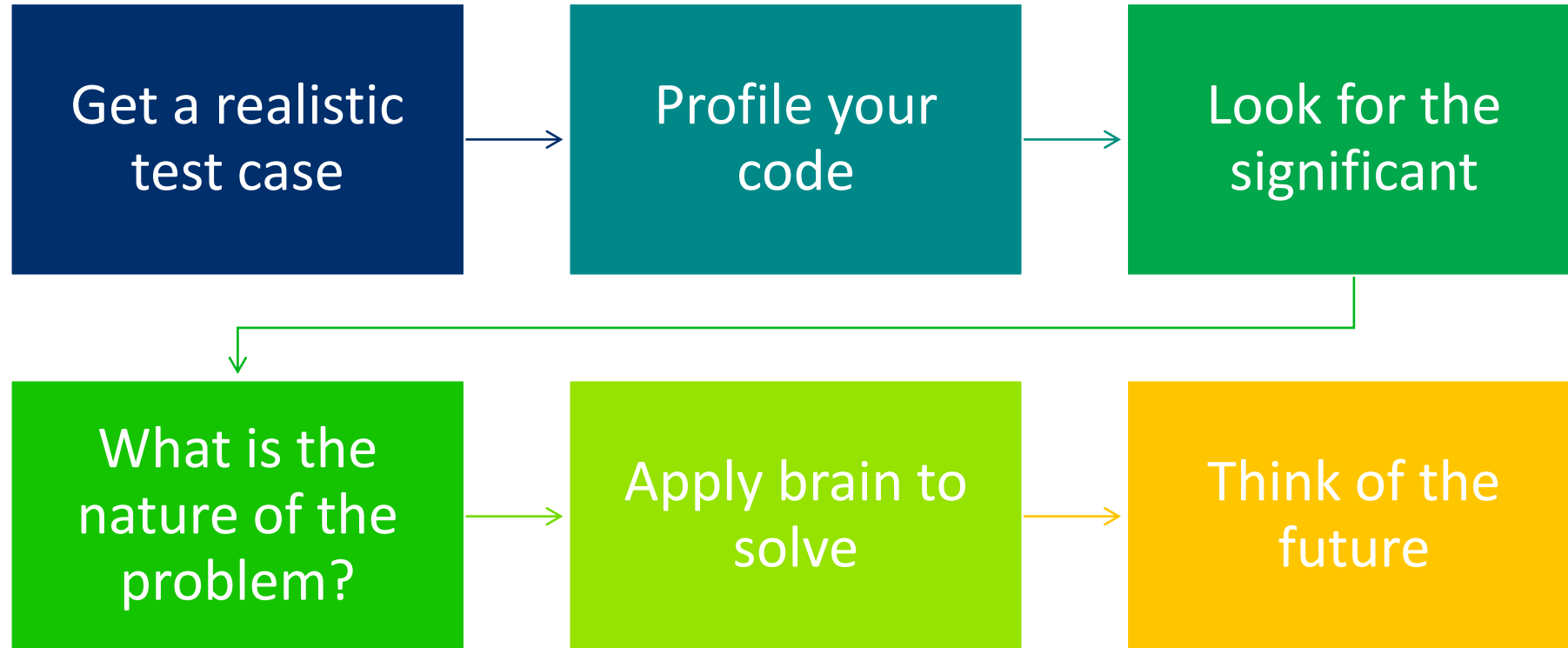
- **T**rack the problem
- **R**eproduce
- **A**utomate - (and simplify) the test case
- **F**ind origins – where could the “infection” be from?
- **F**ocus – examine the origins
- **I**solate – narrow down the origins
- **C**orrect – fix and verify the test case is successful

Profiling

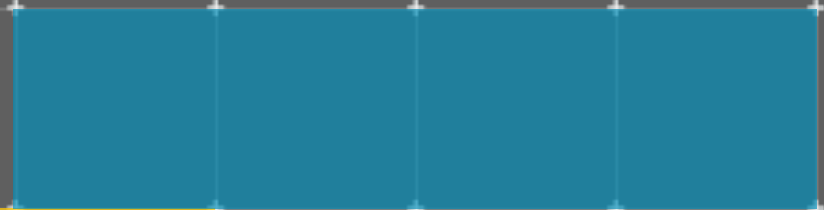
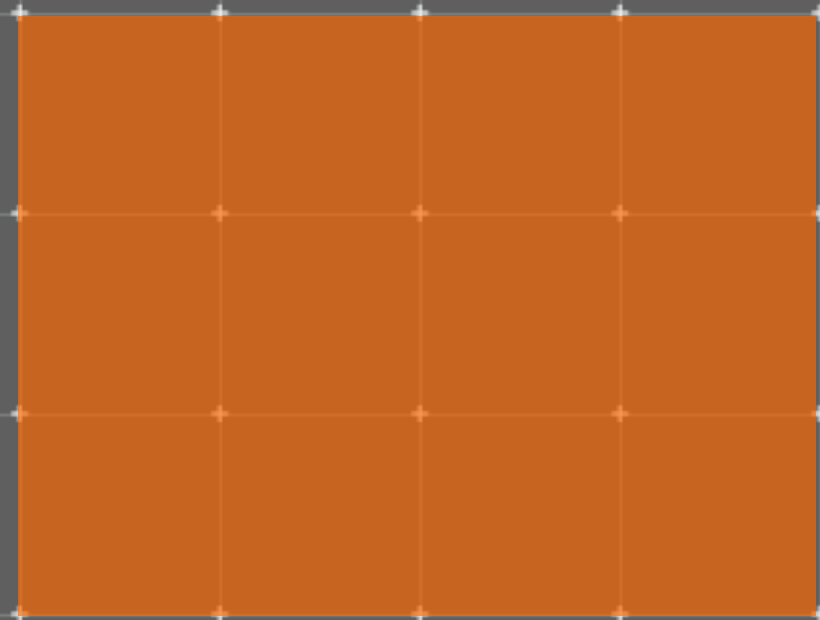
Profiling is central to understanding and improving application performance.



Performance Improvement Workflow



Arm Software



Arm Forge

An interoperable toolkit for debugging and profiling



Commercially supported
by Arm



Fully Scalable



Very user-friendly

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

Run and ensure application correctness

Combination of debugging and re-compilation

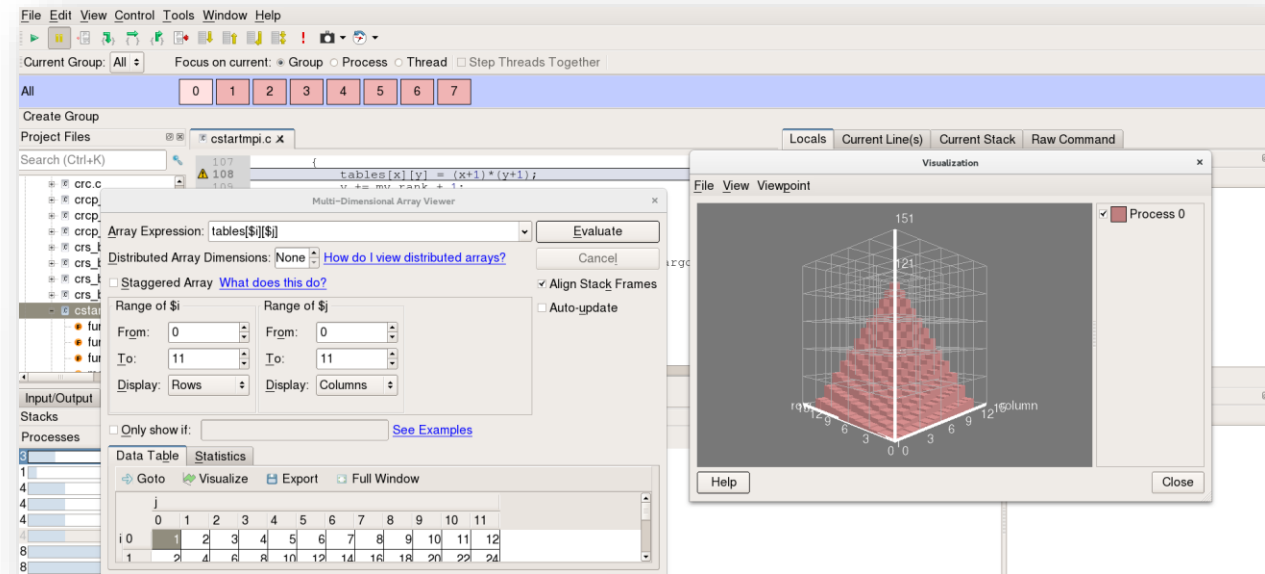
- Ensure application correctness with **Arm DDT scalable debugger**
- Integrate with continuous integration system.
- Use version control to track changes and leverage Forge's built-in VCS support.

Examples:

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

```
$> ddt mpirun -n 48 ./example
```

Step	Time	Process	Action
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
18	2:19.048	n/a	Select process 4
19			Additional Information
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

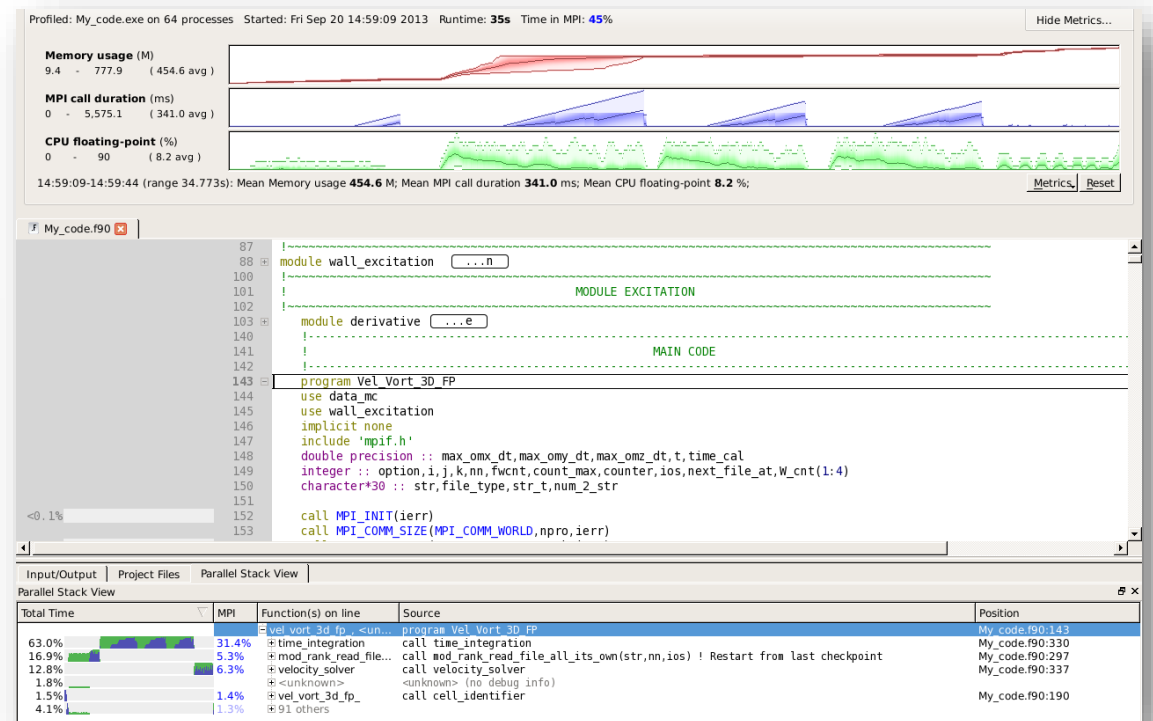
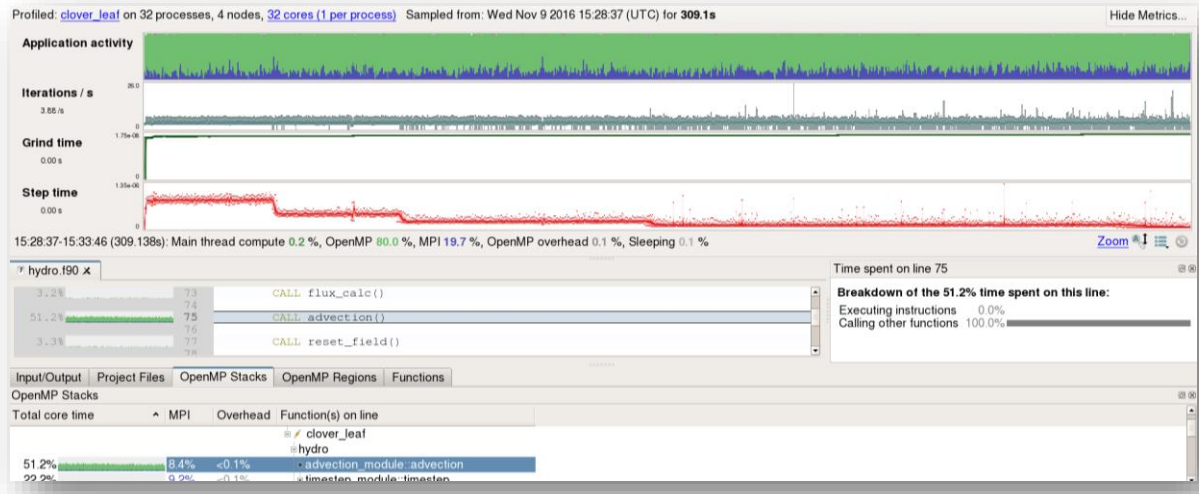


Visualize the performance of your application

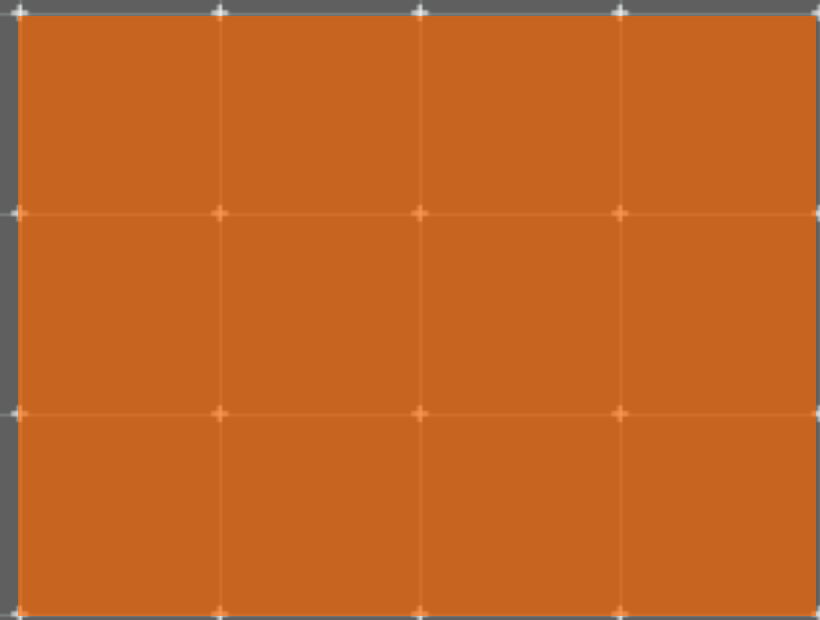
- Measure all performance aspects with **Arm MAP parallel profiler**
- Identify bottlenecks and rewrite some code for better performance

Examples:

```
$> map --profile mpirun -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?

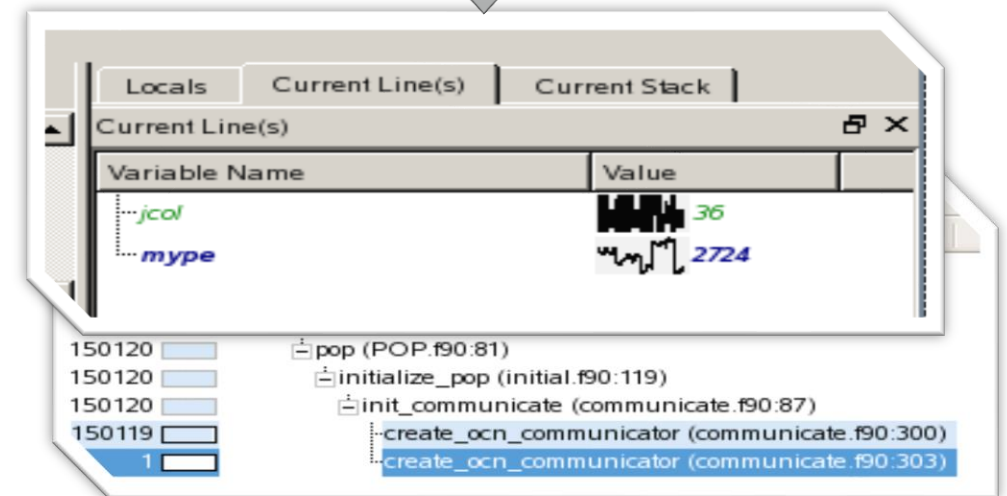
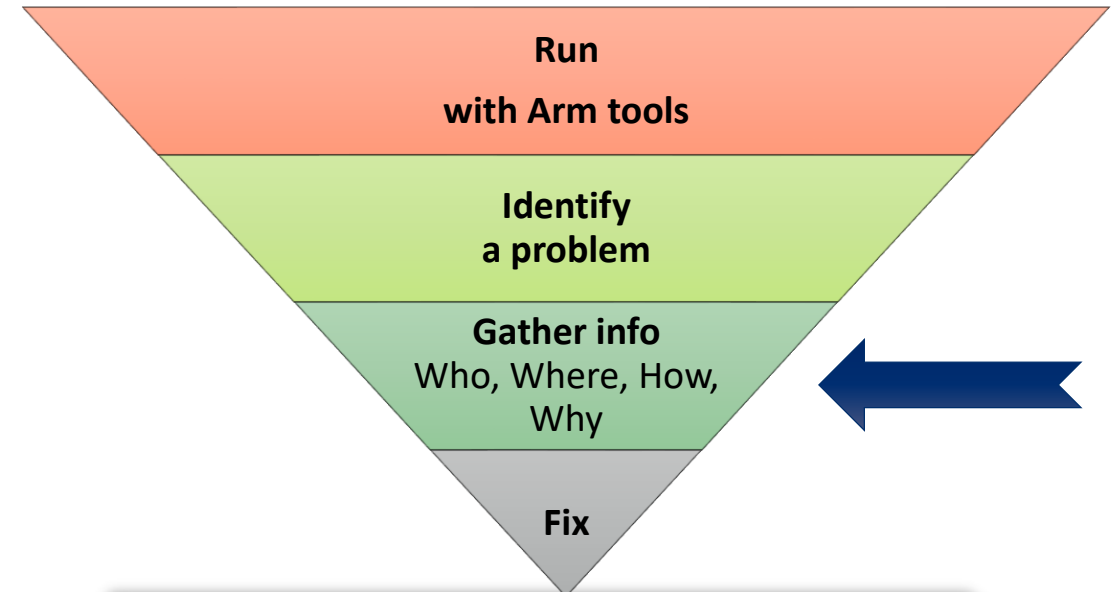
- leaps to source

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

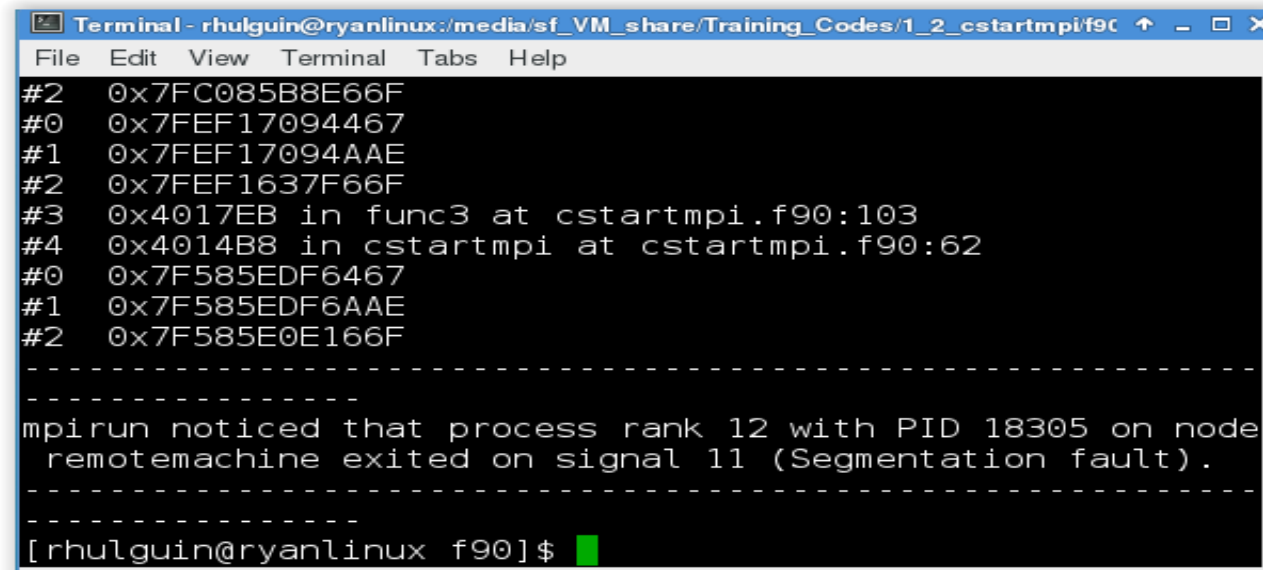
As with any debugger, code must be compiled with the debug flag typically `-g`

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

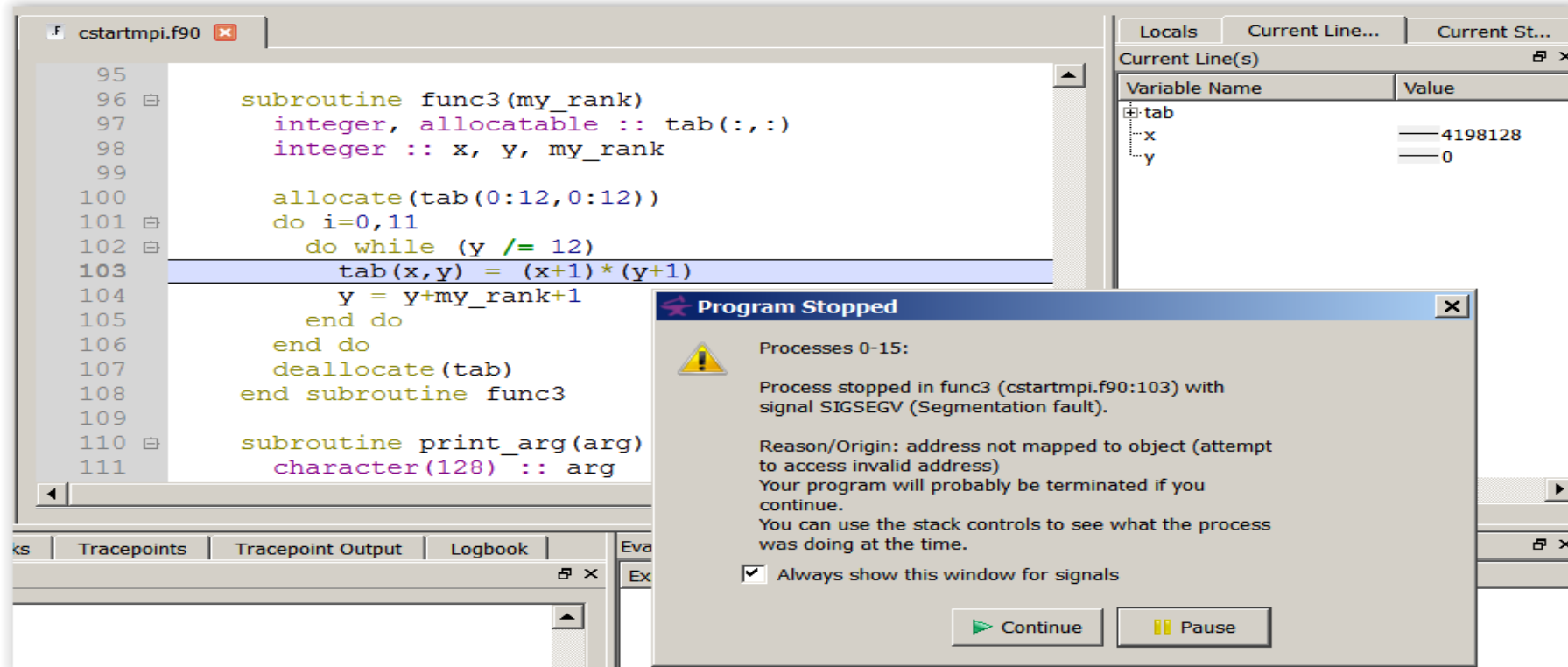
In this example, the application crashes with a segmentation error outside of DDT.



```
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]$
```

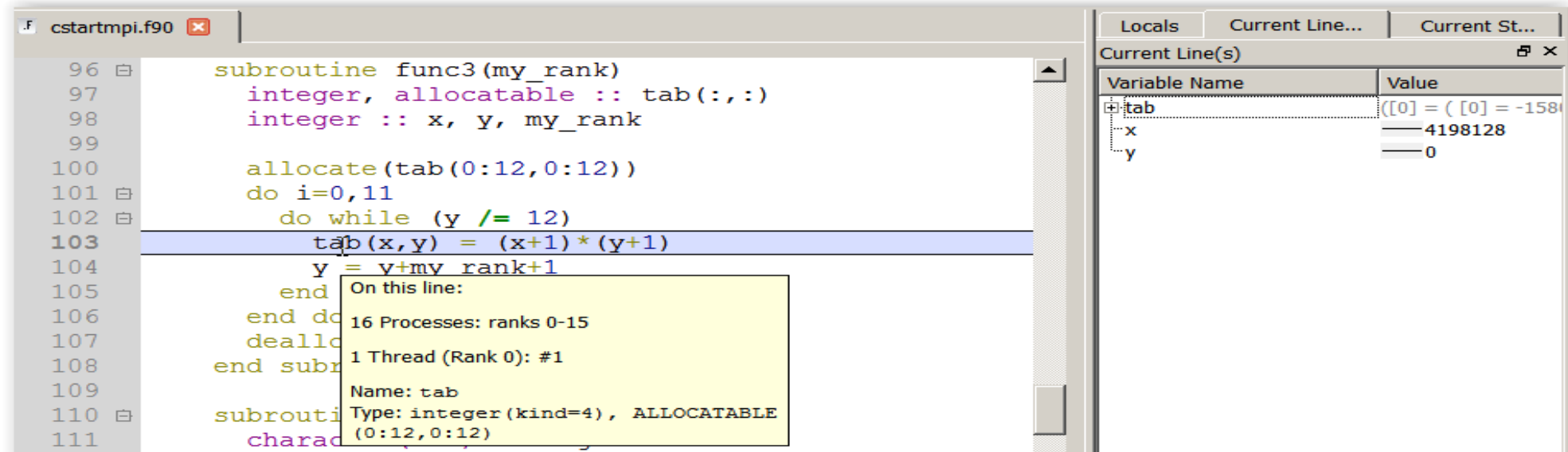
What happens when it runs under DDT?

Segmentation Fault in DDT



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

Invalid Memory Access



```
96  subroutine func3(my_rank)
97      integer, allocatable :: tab(:, :)
98      integer :: x, y, my_rank
99
100     allocate(tab(0:12, 0:12))
101     do i=0, 11
102     do while (y /= 12)
103     tab(x, y) = (x+1) * (y+1)
104     y = y+my_rank+1
105     end do
106     end do
107     deallocate(tab)
108     end subroutine
109
110     subroutine charac
111     character
```

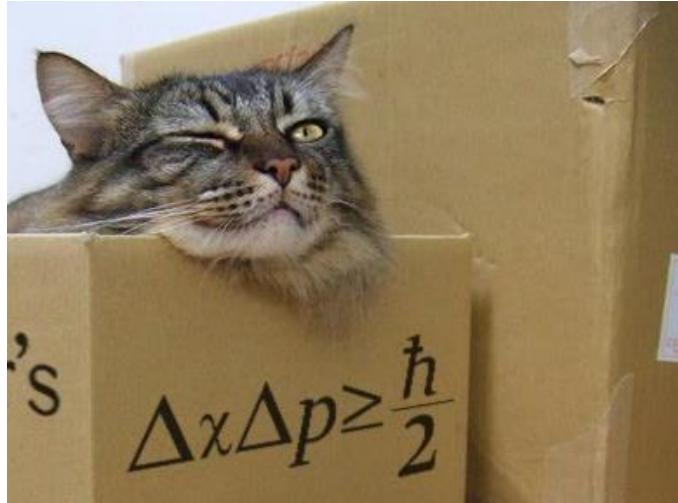
On this line:
16 Processes: ranks 0-15
1 Thread (Rank 0): #1
Name: tab
Type: integer(kind=4), ALLOCATABLE
(0:12, 0:12)

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



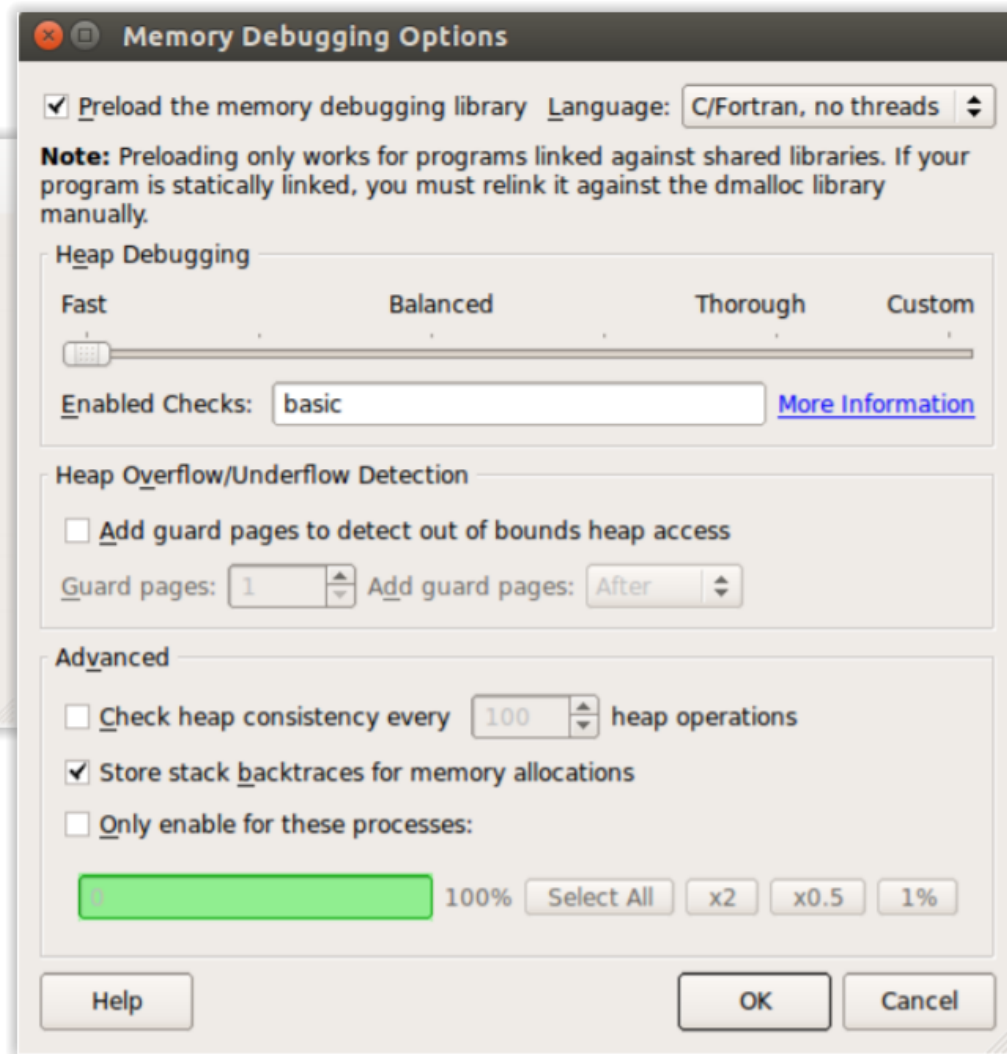
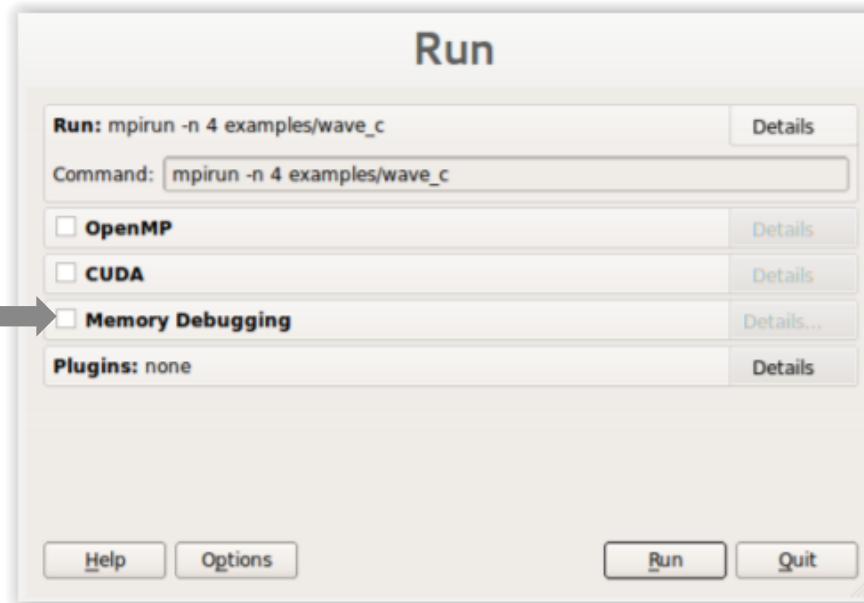
**SCHRODIN
BUG**



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

Fast

basic

- Detect invalid pointers passed to memory functions (e.g. malloc, free, ALLOCATE, DEALLOCATE,...)

check-fence

- Check the end of an allocation has not been overwritten when it is freed.

free-protect

- Protect freed memory (using hardware memory protection) so subsequent read/writes cause a fatal error.

Added goodness

- Memory usage, statistics, etc.

Balanced

free-blank

- Overwrite the bytes of freed memory with a known value.

alloc-blank

- Initialise the bytes of new allocations with a known value.

check-heap

- Check for heap corruption (e.g. due to writes to invalid memory addresses).

realloc-copy

- Always copy data to a new pointer when re-allocating a memory allocation (e.g. due to realloc)

Thorough

check-blank

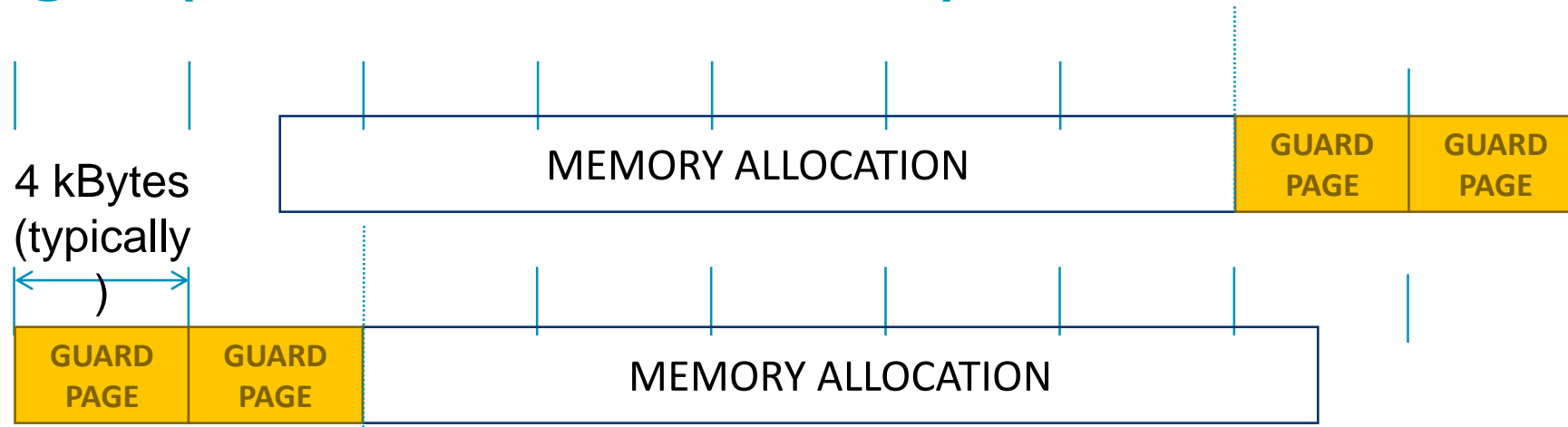
- Check to see if space that was blanked when a pointer was allocated/freed has been overwritten.

check-funcs

- Check the arguments of addition functions (mostly string operations) for invalid pointers.

*See user-guide:
Chapter 12.3.2*

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



Current Group: All Focus on current: Group Process Thread Step Threads Together

All 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

```

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

Input/Output Breakpoints Watchpoints Stacks Tracepoints Tracepoint Output Logbook

Stacks

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)

Evaluate

Expression	Value
i * ntsamples	-212322546

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

New Bugs from Latest Changes

The screenshot shows a debugger interface with the following components:

- Project Files:** A tree view showing the source code structure, including 'wave_openmp.c'.
- Code Editor:** Displays the source code for 'wave_openmp.c'. The current line is 227, which is highlighted in red. The code snippet is:

```
for (j = 1; j <= npoints; j++)  
{  
    /* global endpoints */  
    if ((first + j - 1 == 1) || (first  
        newval[j] = 0.0;  
    else  
        do_math(j);  
}  
  
/* swap arrays */  
oldval = values;  
values = newval;  
}
```
- Locals:** A table showing the current line's local variables:

Variable Name	Value
oldval	0x7fff4b7a010
values	0x7fff4b7a010
- Evaluate:** A table showing the evaluation of expressions:

Expression	Value
newval	0x7fff4b7a010
oldval	0x7fff4b7a010
values	0x7fff4b7a010
- Stacks:** A table showing the current stack frame:

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

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:

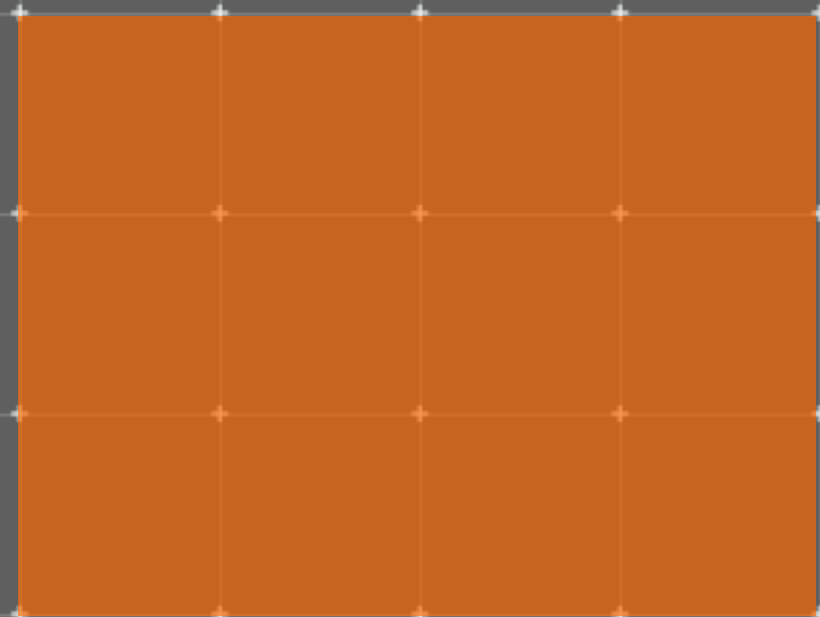
Time	Ranks	Message
0:00	0-3	Launching program /home/bpaisley/demo/ddt/cstartmpi/cstartmpi.exe at Wed Mar 1 10:59:59 2017 Executable modified on Tue Feb 21 10:53:10 2017
0:05	0-3	Startup complete.
0:05	n/a	Select process group All
0:05	0-3	Add tracepoint for cstartmpi.c:113 Vars: x, y
0:05	0-3	Add breakpoint 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 Locals window shows the following variables and values:

Variable Name	Value
...p	100

The Evaluate window is empty.

Arm DDT Demo



Five great things to try with Alinea DDT

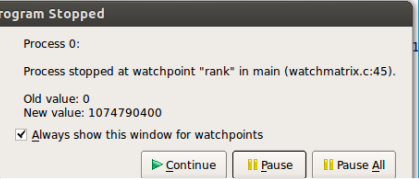
Tracepoint	Processes	Values logged
vhone #90 85	976, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol 2-43 mod pey
vhone #90 81	960, ranks 12,14-17,22-23,12...	ks 1 kmax pec
vhone #90 85	942, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol 2-43 mod pey
vhone #90 81	920, ranks 12,14-17,22-23,12...	ks 1 kmax pec
vhone #90 85	919, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol 2-43 mod pey
vhone #90 81	898, ranks 12,14-17,22-23,12...	ks 1 kmax pec
vhone #90 85	884, ra 12,14-	
vhone #90 81	880, ra 17 14-	

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 O; k++)
      C[i][j] += A[i][k] * B[k][j];
    
```



Stop on variable change

```

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

portability 'i' is of type 'void *'. When using void pointers in calcula
Left click to add a breakpoint on line 50

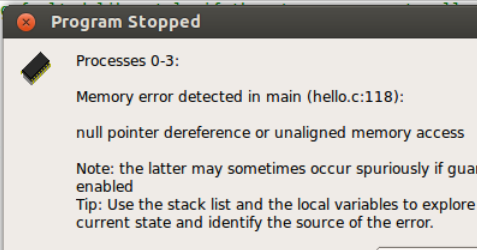
Static analysis warnings on code errors

```

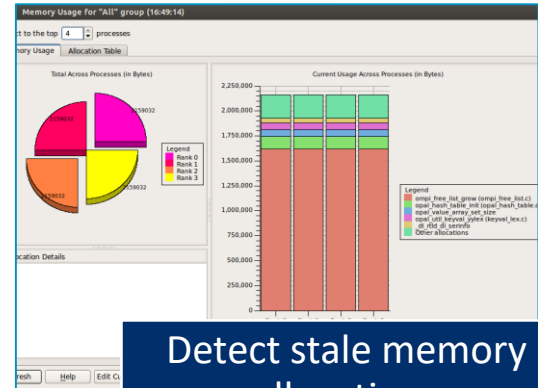
&& !strcmp(argv[i], "crash")) {
0;
s", *(char **)argv[i]);
ll se

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/19.0.2`

Prepare the code

- `$ cc -O0 -g myapp.c -o myapp.exe`

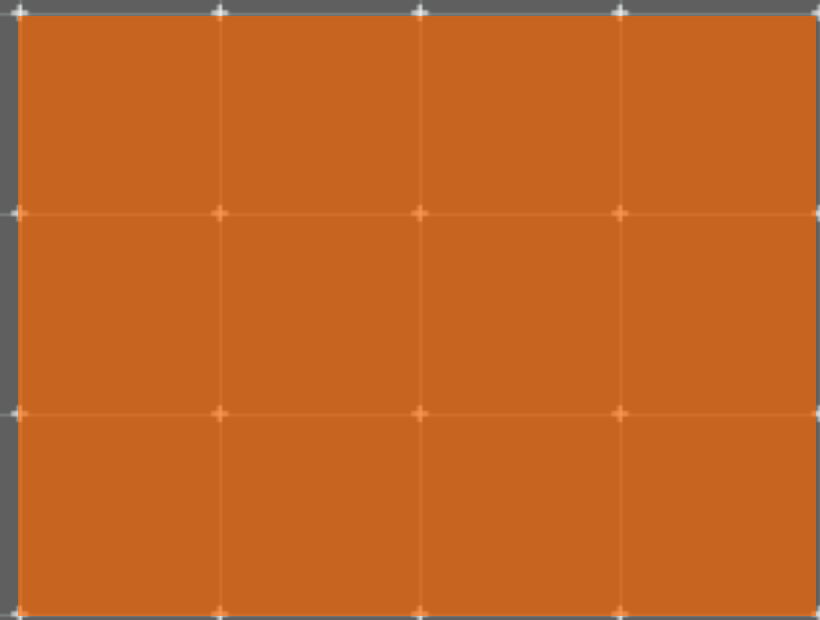
Start Arm DDT in interactive mode

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

Or use the reverse connect mechanism

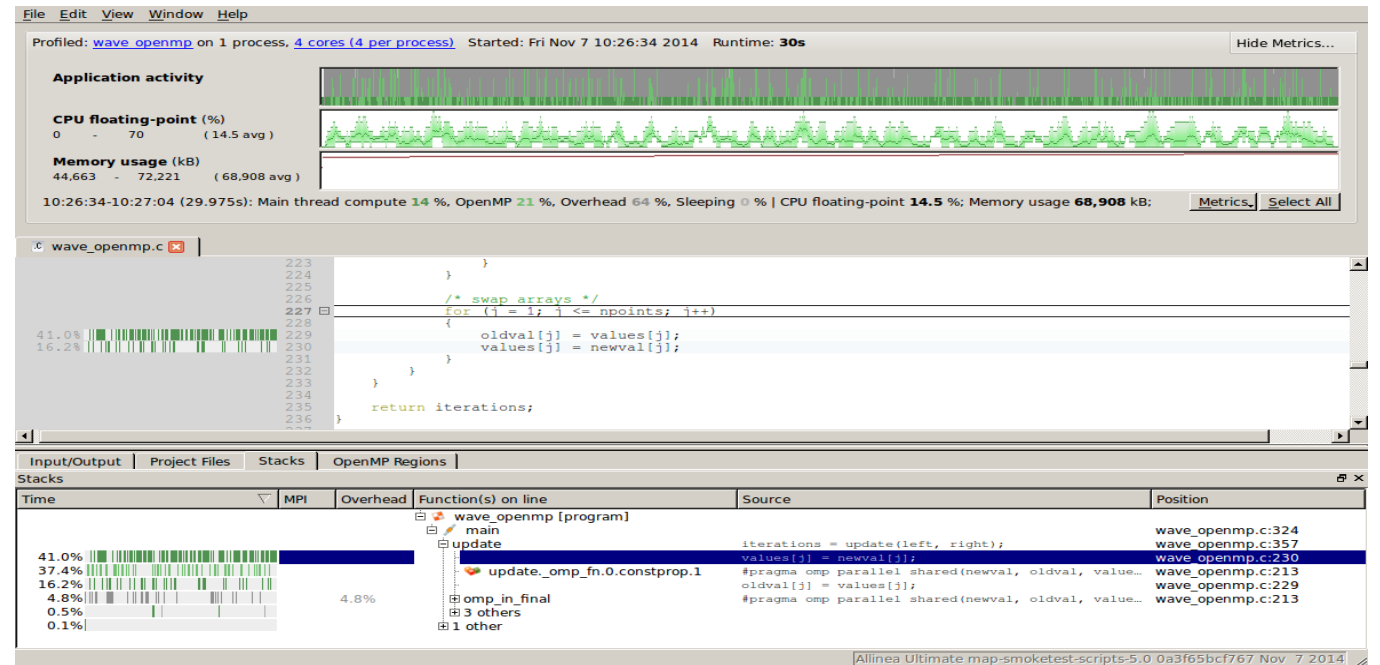
- On the login node:
 - `$ ddt &`
- (or use the remote client) <- **Preferred method**
- Then, edit the job script to run the following command and submit:
 - `ddt --connect aprun -n 8 ./myapp.exe arg1 arg2`

Profiling with MAP

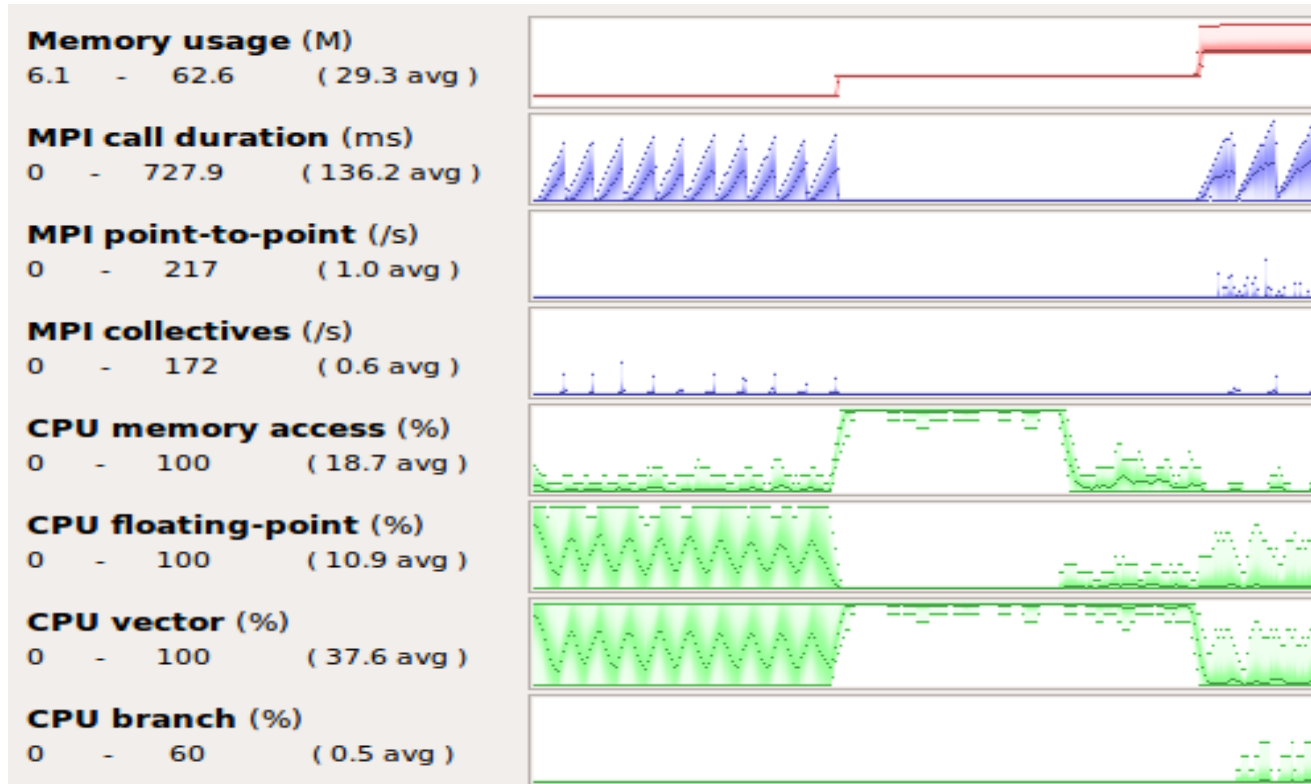


Arm MAP – The Profiler

- ✓ 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

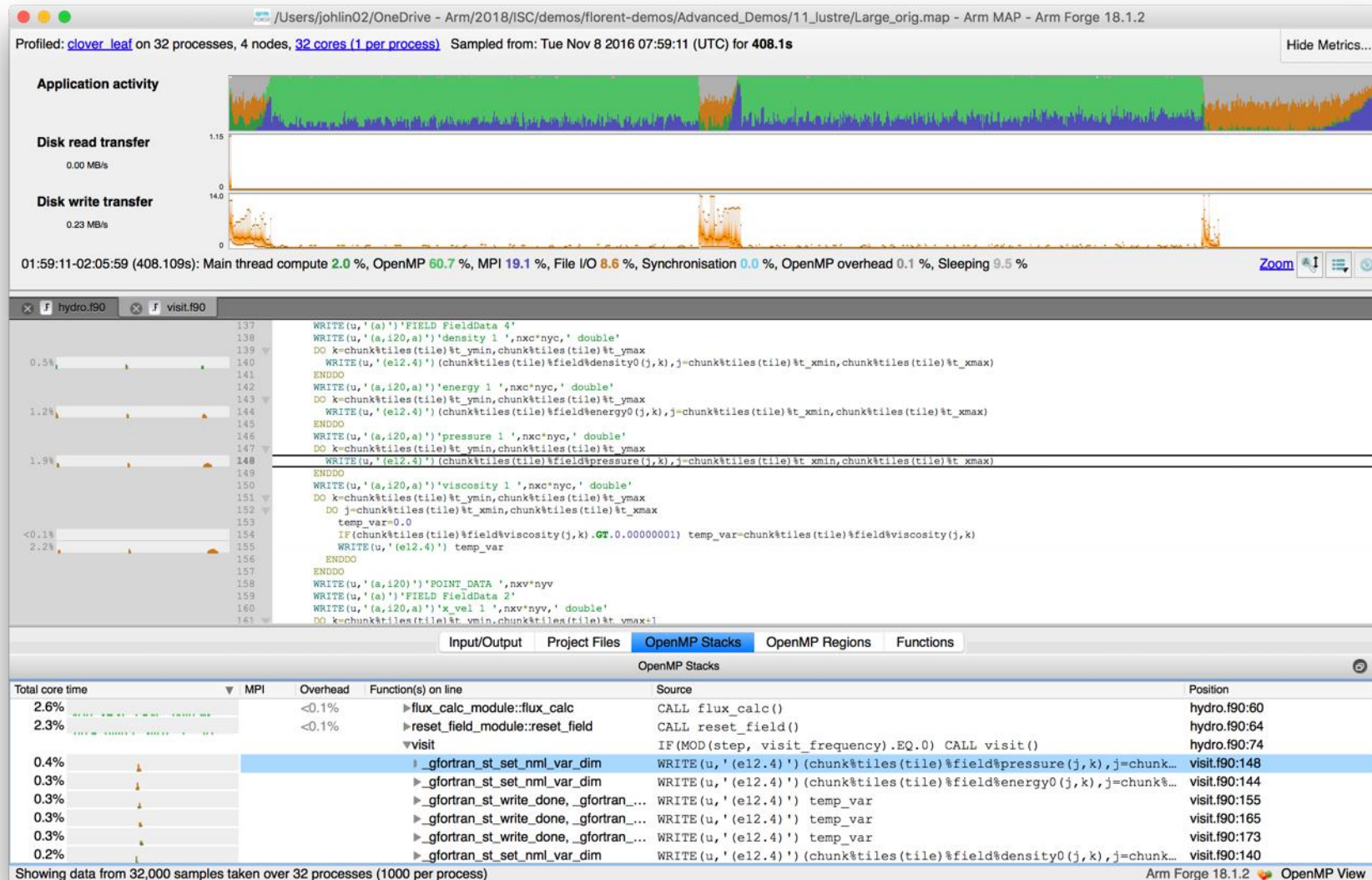
Spot MPI and OpenMP imbalance and overhead

Optimize CPU memory and vectorization in loops

Detect and diagnose I/O bottlenecks at real scale

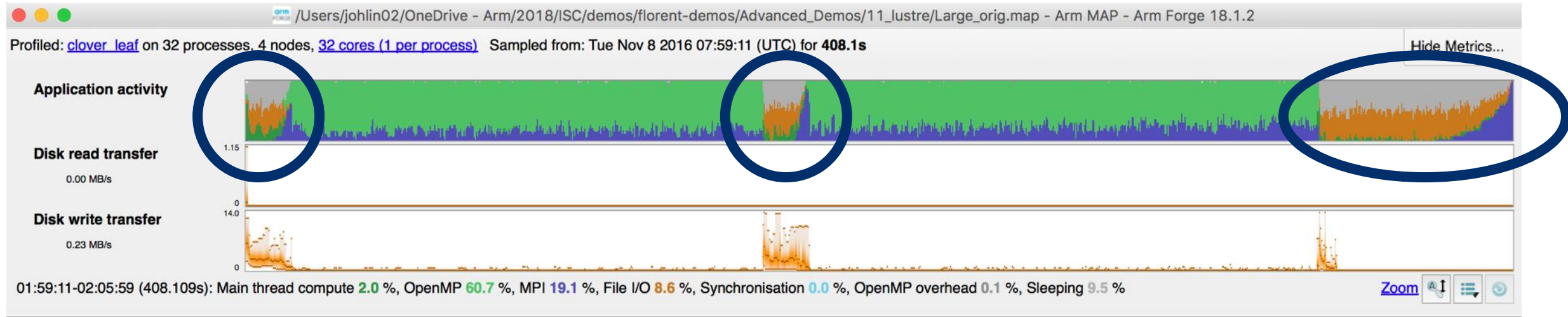
Initial profile of CloverLeaf shows surprisingly unequal I/O

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



Symptoms and causes of the I/O issues

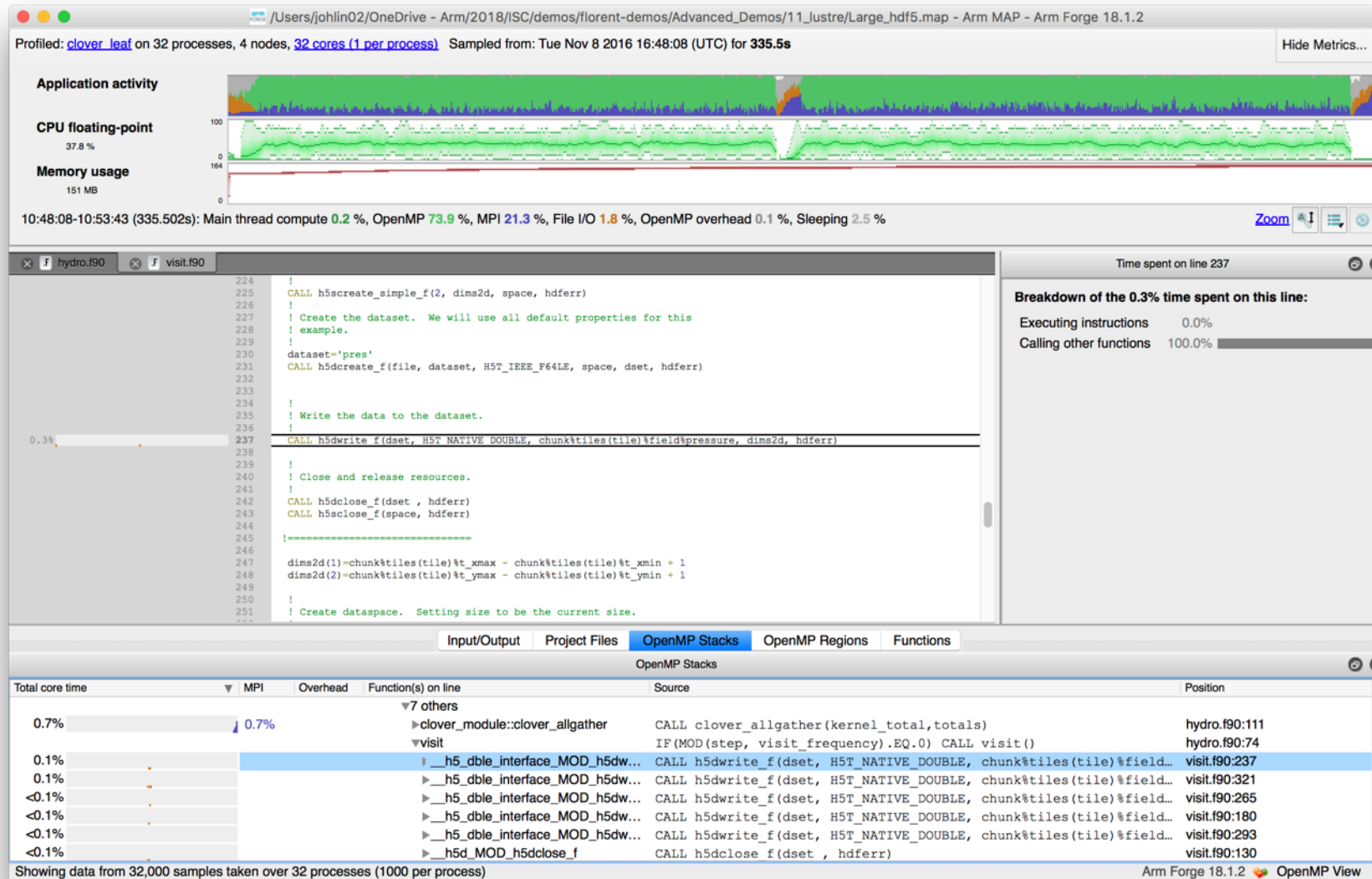
Sub-optimal file format and surprise buffering.



- 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.

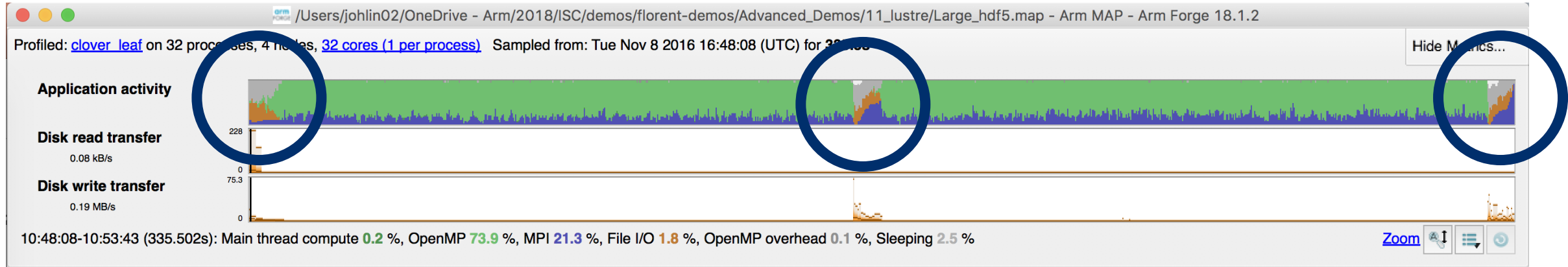
Solution: use HDF5 to write binary files

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.

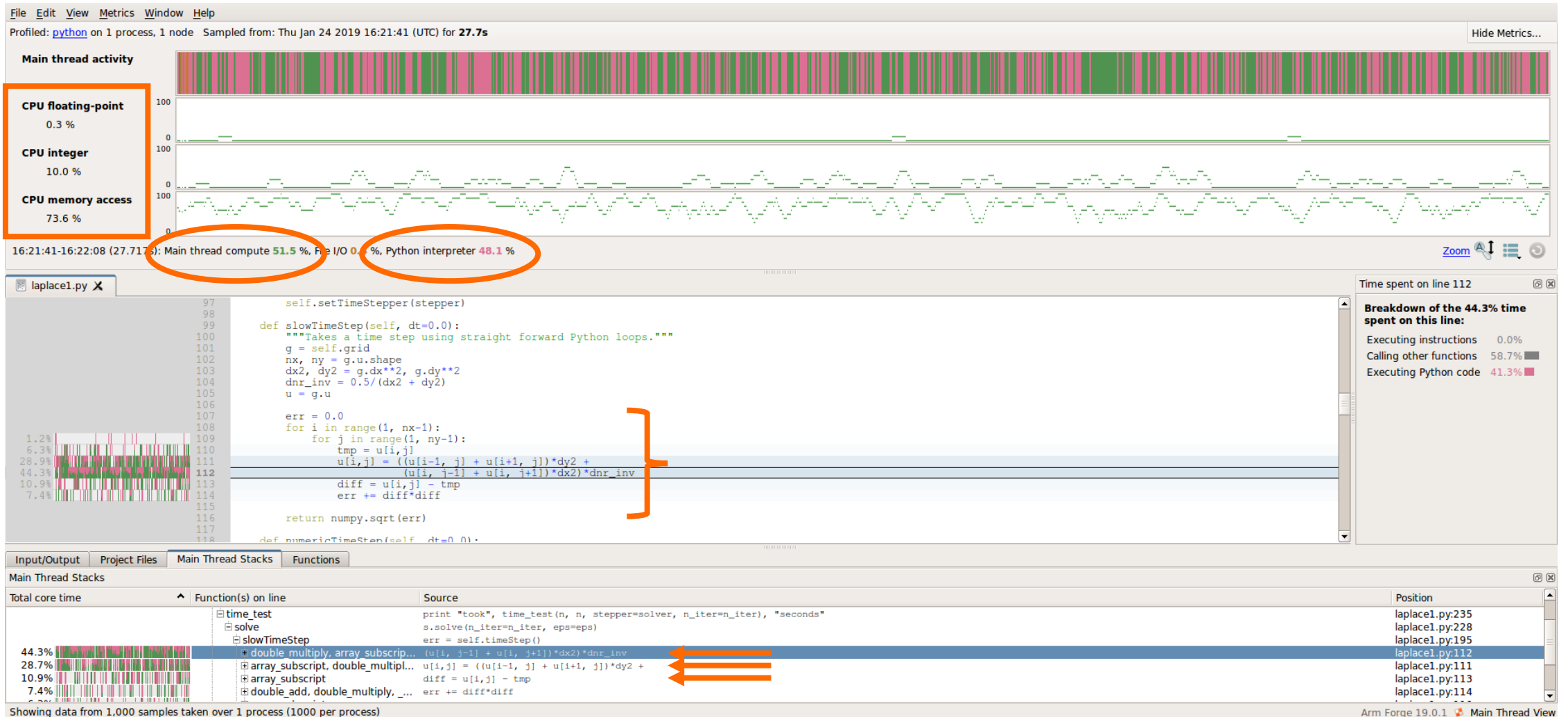
Arm MAP: Python profiling

- Launch command
 - \$ **python** ./laplace1.py slow 100 100
- Profiling command
 - \$ **map --profile python** ./laplace1.py slow 100 100
 - --profile: non-interactive mode
 - --output: name of output file
- Display profiling results
 - \$ **map** laplace1.map

Laplace1.py

```
[...]  
err = 0.0  
for i in range(1, nx-1):  
    for j in range(1, ny-1):  
        tmp = u[i,j]  
        u[i,j] = ((u[i-1, j] + u[i+1, j])*dy2 +  
                (u[i, j-1] + u[i, j+1])*dx2)*dnr_inv  
        diff = u[i,j] - tmp  
        err += diff*diff  
return numpy.sqrt(err)  
[...]
```

Naïve Python loop (laplace1.py slow 100 1000)




Optimizing computation on NumPy arrays

Naïve Python loop

```
err = 0.0
for i in range(1, nx-1):
    for j in range(1, ny-1):
        tmp = u[i,j]
        u[i,j] = ((u[i-1, j] + u[i+1, j])*dy2 +
                 (u[i, j-1] + u[i, j+1])*dx2)*dnr_inv
        diff = u[i,j] - tmp
        err += diff*diff
return numpy.sqrt(err)
```

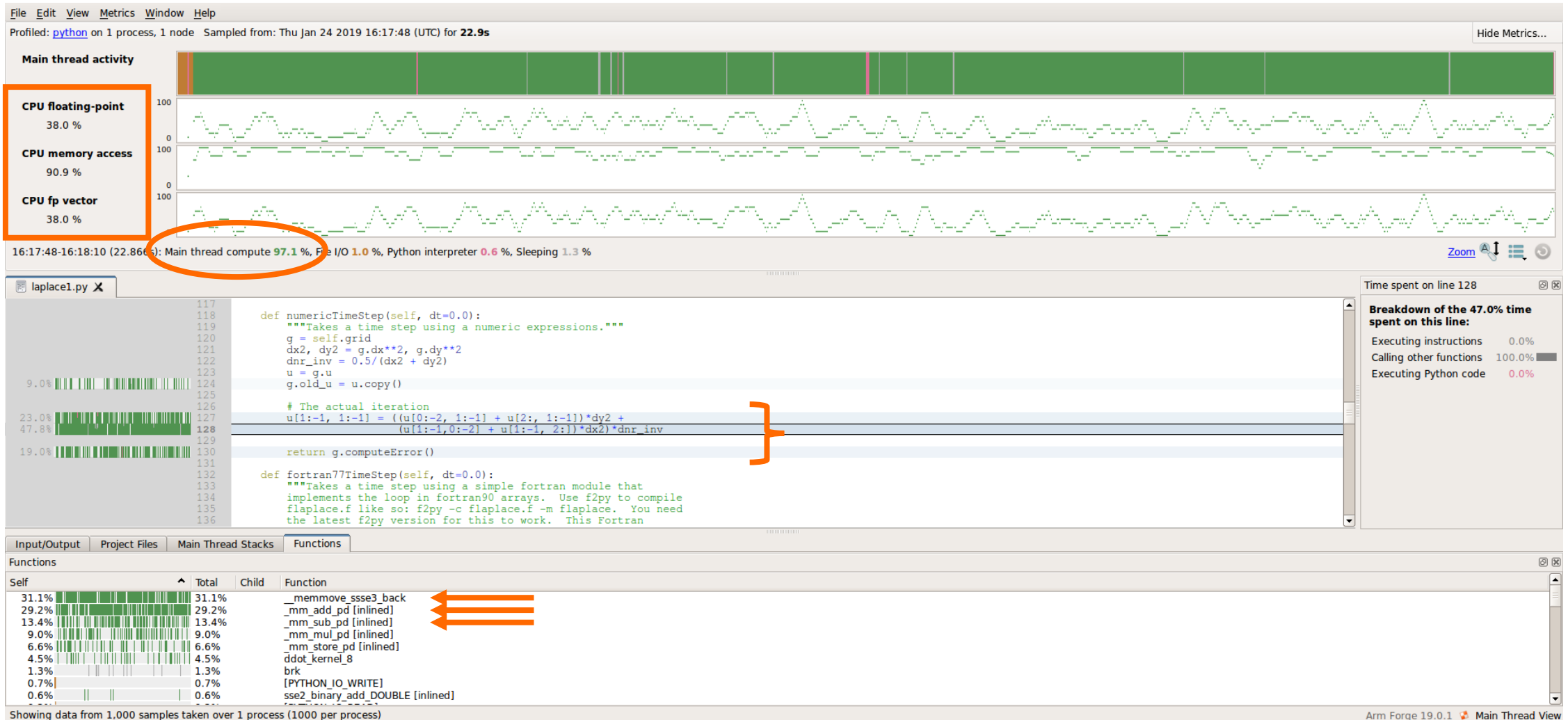
NumPy loop



```
u[1:-1, 1:-1] =
    ((u[0:-2, 1:-1] + u[2:, 1:-1])*dy2 +
     (u[1:-1, 0:-2] + u[1:-1, 2:])*dx2)*dnr_inv
return g.computeError()
```

NumPy array notation (laplace1.py numeric 1000 1000)

This is 10 times more iterations than was computed in the previous profile



Arm MAP cheat sheet

Load the environment module (manually specify version)

- `$ module load forge/19.0.2`

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`

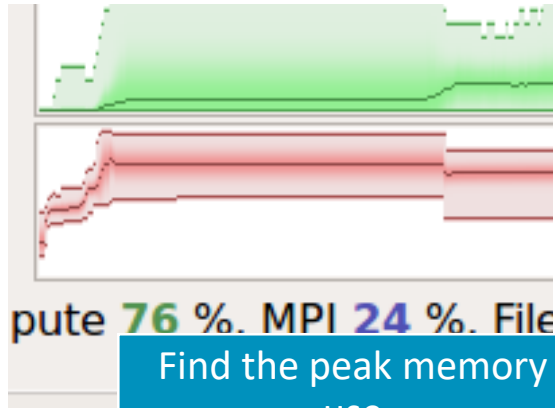
Follow the instructions displayed to prepare the code

- `$ 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 Alinea MAP



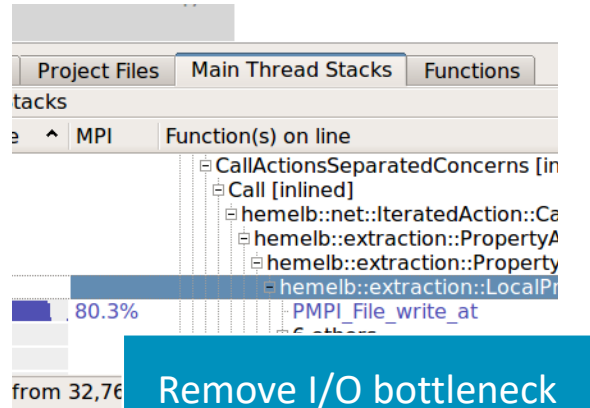
Find the peak memory use

```

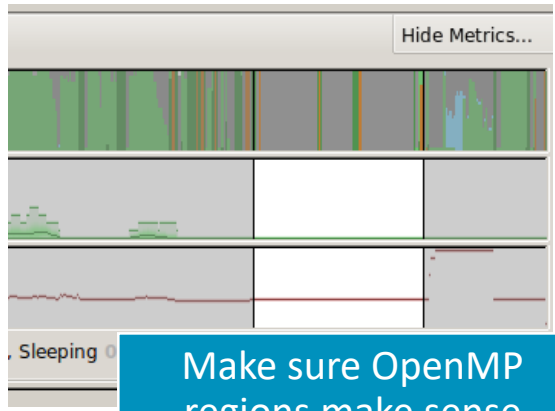
30 ! late to the party
31 do j=1,20*nprocs; a
32 end if
33
34 if (pe /= 0) then
35 call MPI_SEND(a, si
36 else
37 do from=1,nprocs-1
38 call MPI_RECV(b,
39 do j=1,50; b=sqrt
40 print *, "Answer f
41 end do
42 end if
43 end do
44 call MPI_BARRIER(MPI CO
45

```

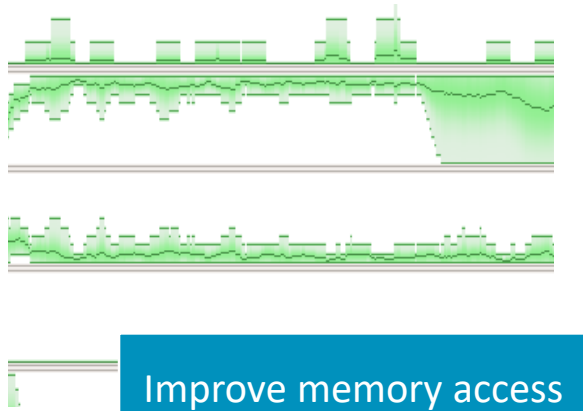
Fix an MPI imbalance



Remove I/O bottleneck



Make sure OpenMP regions make sense



Improve memory access

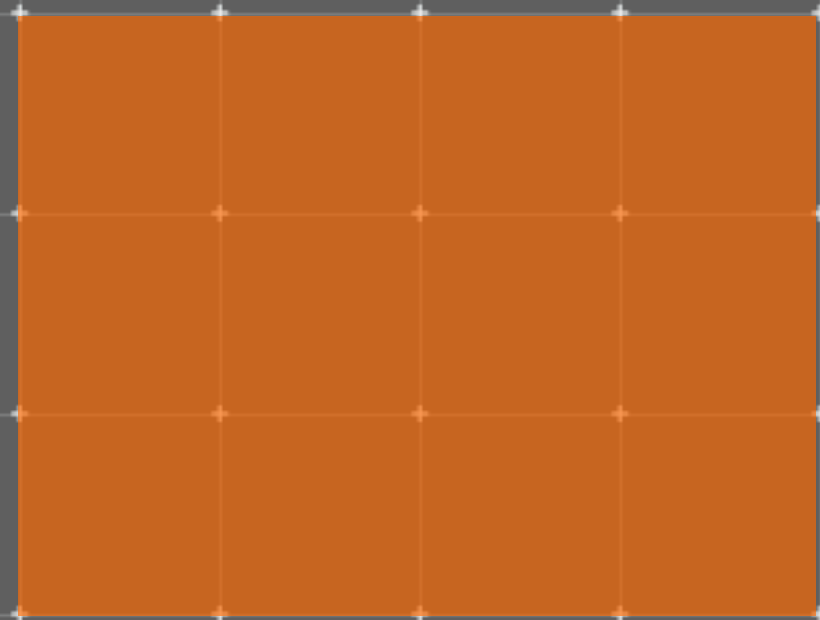
```

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

```

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
- 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

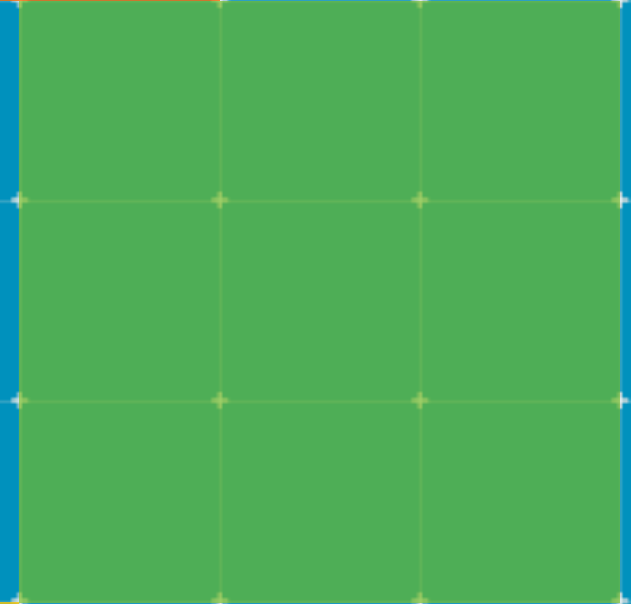
Simply add the link flags to your Makefile, or however appropriate

```
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