Debugging and Profiling with DDT and Map

ALCF Simulation Data and Learning Workshop

October 2, 2019

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

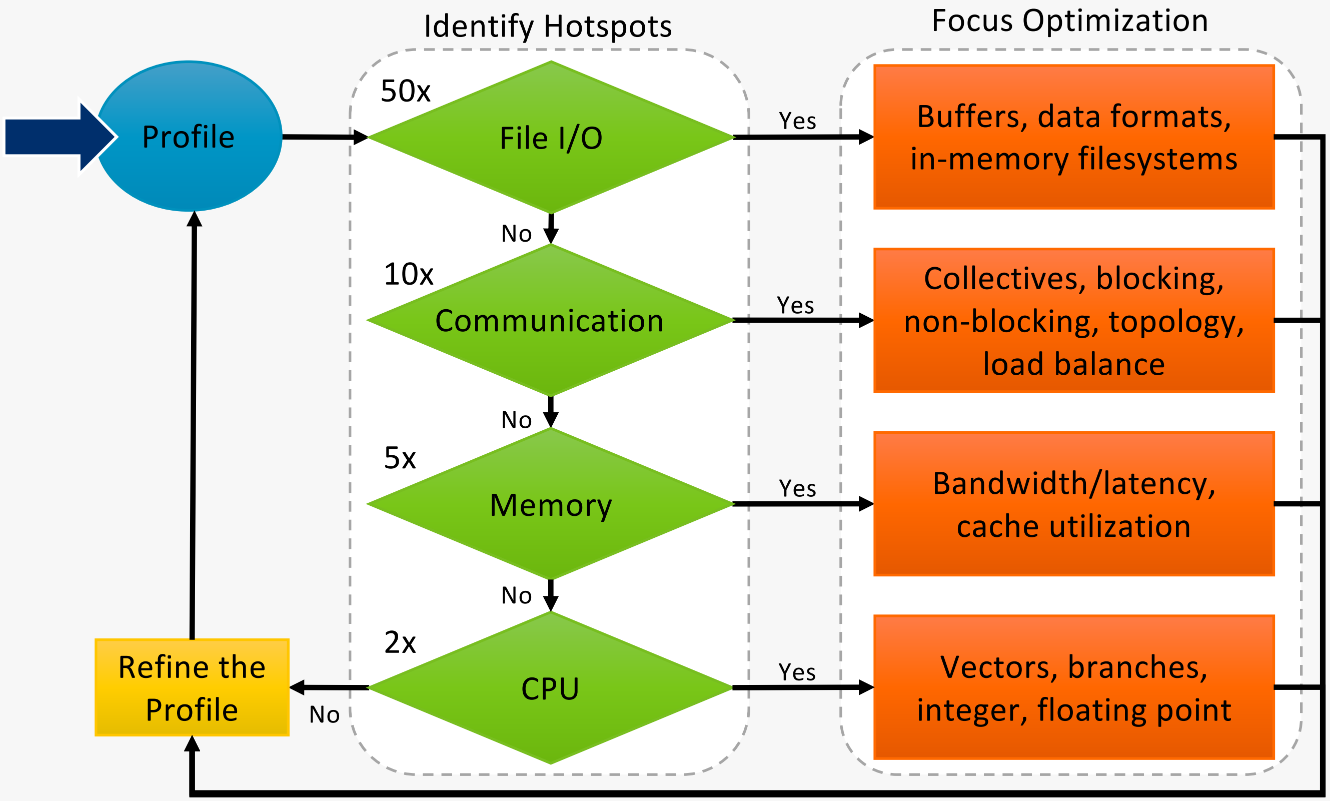
Transforming a broken program to a working one

How? TRAFFIC!

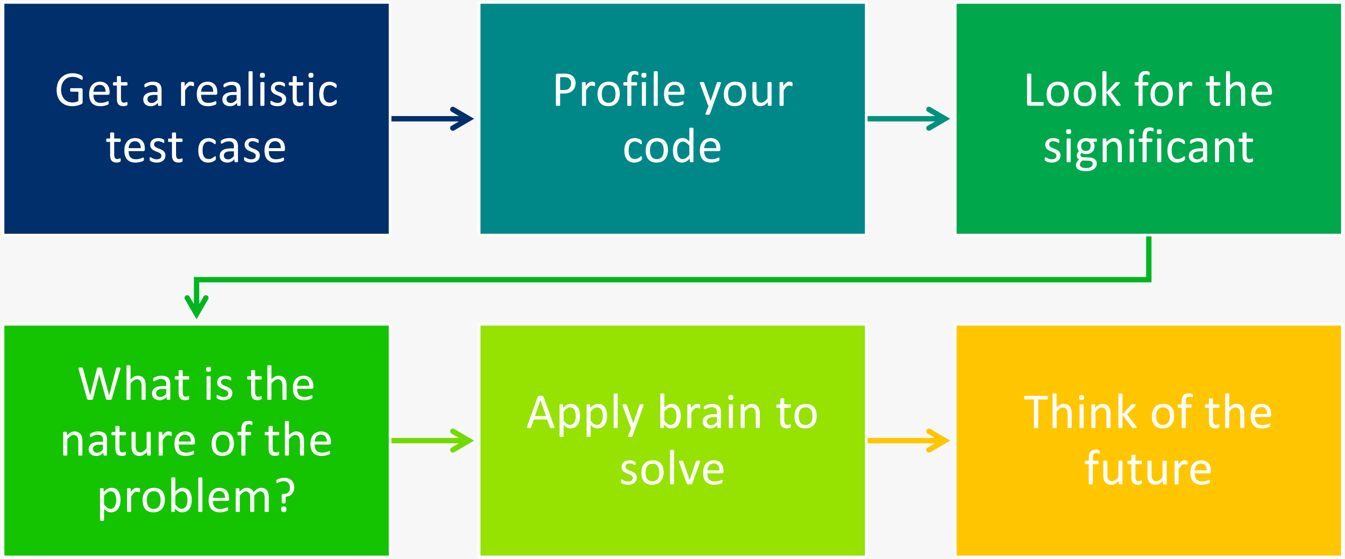
- **T**rack the problem
- **R**eproduce
- **A**utomate - (and simplify) the test case
- **F**ind origins – where could the “infection” be from?
- **F**ocus – examine the origins
- **I**solate – narrow down the origins
- **C**orrect – fix and verify the test case is successful

Profiling

Profiling is central to understanding and improving application performance.


Identify Hotspots
Focus Optimization
Profile
50x
File I/O
Yes
Buffers, data formats, in-memory filesystems
No
10x
Communication
Yes
Collectives, blocking, non-blocking, topology, load balance
No
5x
Memory
Yes
Bandwidth/latency, cache utilization
No
2x
CPU
Yes
Vectors, branches, integer, floating point
No
Refine the Profile

Performance Improvement Workflow



Get a realistic test case

Profile your code

Look for the significant

What is the nature of the problem?

Apply brain to solve

Think of the future

Arm Software

arm

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

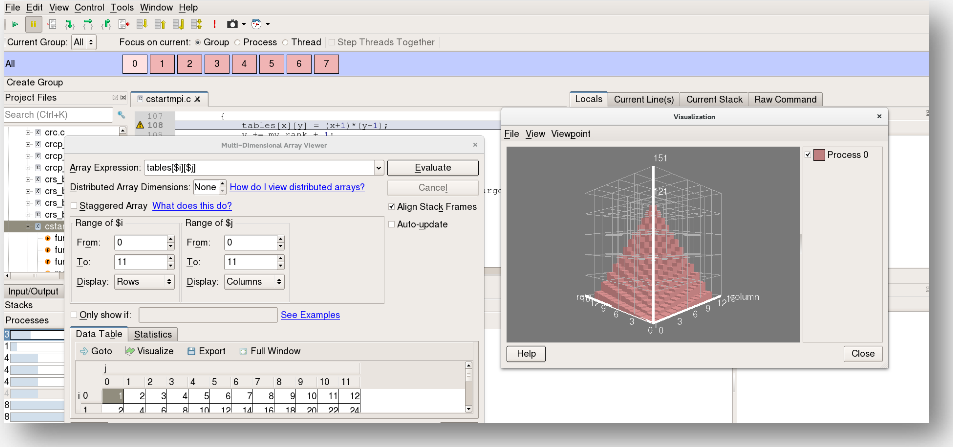
Run and ensure application correctness

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.

<u>Examples:</u>

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



File Edit View Control Tools Window Help

Current Group: All ⇕ Focus on current: ● Group ○ Process ○ Thread □ Step Threads Together

All 0 1 2 3 4 5 6 7

Create Group

Project Files | cstartmpi.c ✕

Search (Ctrl+K)

```
107          {
108              tables[x][y] = (x+1)*(y+1);
```

crc.c, crcp_, crcp_, crcp_, crs_b, crs_b, crs_b, crs_b, cstar

Input/Output, Stacks, Processes

Locals | Current Line(s) | Current Stack | Raw Command

Multi-Dimensional Array Viewer

Array Expression: tables[$i][$j] Evaluate

Distributed Array Dimensions: None How do I view distributed arrays? Cancel

□ Staggered Array What does this do? ☑ Align Stack Frames □ Auto-update

Range of $i		Range of $j	
From:	0	From:	0
To:	11	To:	11
Display:	Rows	Display:	Columns

□ Only show if: See Examples

Data Table | Statistics

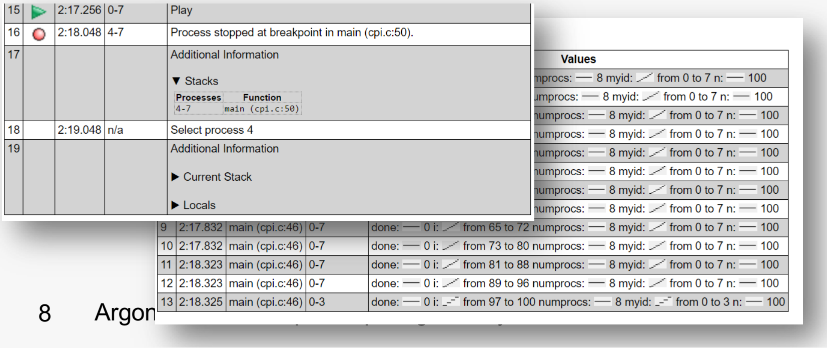
Goto Visualize Export Full Window

	j											
	0	1	2	3	4	5	6	7	8	9	10	11
i 0	1	2	3	4	5	6	7	8	9	10	11	12
1	2	4	6	8	10	12	14	16	18	20	22	24

Visualization

File View Viewpoint

☑ Process 0



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: 4-7 Function: main (cpi.c:50)
18		2:19.048	n/a	Select process 4
19				Additional Information ▶ Current Stack ▶ Locals

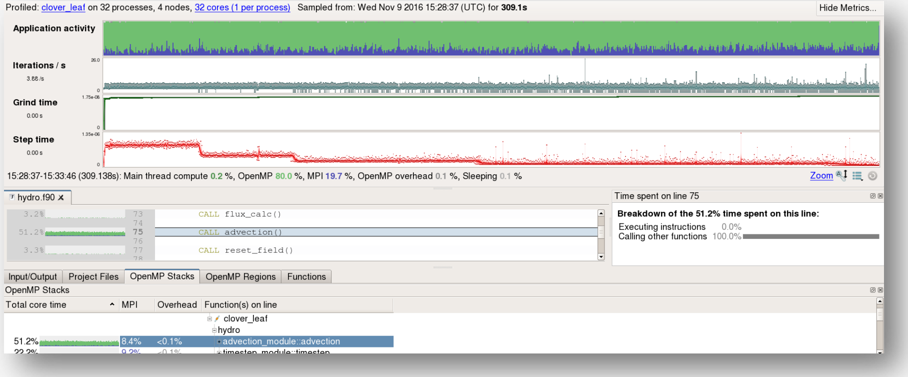
				Values
				mprocs: 8 myid: from 0 to 7 n: 100
				umprocs: 8 myid: from 0 to 7 n: 100
				numprocs: 8 myid: from 0 to 7 n: 100
				numprocs: 8 myid: from 0 to 7 n: 100
				numprocs: 8 myid: from 0 to 7 n: 100
				numprocs: 8 myid: from 0 to 7 n: 100
				numprocs: 8 myid: from 0 to 7 n: 100
				numprocs: 8 myid: from 0 to 7 n: 100
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
- Identify bottlenecks and rewrite some code for better performance

Examples:

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



Profiled: clover_leaf on 32 processes, 4 nodes, 32 cores (1 per process) Sampled from: Wed Nov 9 2016 15:28:37 (UTC) for **309.1s**

15:28:37-15:33:46 (309.138s): Main thread compute 0.2 %, OpenMP 80.0 %, MPI 19.7 %, OpenMP overhead 0.1 %, Sleeping 0.1 %

hydro.f90

```
3.2%       CALL flux_calc()

51.2%      CALL advection()

3.3%       CALL reset_field()
```

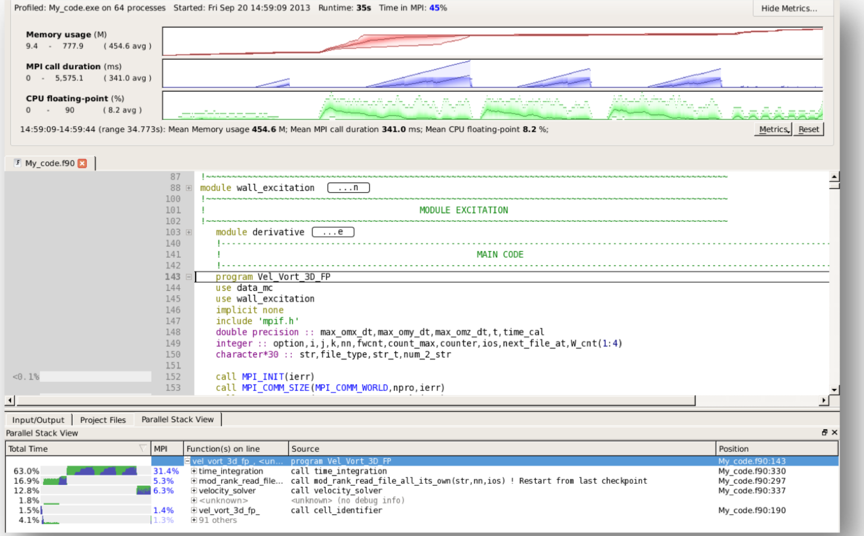
Time spent on line 75

Breakdown of the 51.2% time spent on this line:

Executing instructions 0.0%
Calling other functions 100.0%

OpenMP Stacks

Total core time	MPI	Overhead	Function(s) on line
			clover_leaf
			hydro
51.2%	8.4%	<0.1%	advection_module::advection
22.2%	9.2%	<0.1%	timestep_module::timestep



Profiled: My_code.exe on 64 processes Started: Fri Sep 20 14:59:09 2013 Runtime: **35s** Time in MPI: **45**%

14:59:09-14:59:44 (range 34.773s): Mean Memory usage **454.6** M; Mean MPI call duration **341.0** ms; Mean CPU floating-point **8.2** %;

My_code.f90

```
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
module wall_excitation  ...n
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
!                                    MODULE EXCITATION
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  module derivative  ...e
  !-----------------------------------------------------------------------------------------
  !                                          MAIN CODE
  !-----------------------------------------------------------------------------------------
  program Vel_Vort_3D_FP
  use data_mc
  use wall_excitation
  implicit none
  include 'mpif.h'
  double precision :: max_omx_dt,max_omy_dt,max_omz_dt,t,time_cal
  integer :: option,i,j,k,nn,fwcnt,count_max,counter,ios,next_file_at,W_cnt(1:4)
  character*30 :: str,file_type,str_t,num_2_str

<0.1%  call MPI_INIT(ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD,npro,ierr)
```

Parallel Stack View

Total Time	MPI	Function(s) on line	Source	Position
		vel_vort_3d_fp_, <un...	program Vel_Vort_3D_FP	My_code.f90:143
63.0%	31.4%	time_integration	call time_integration	My_code.f90:330
16.9%	5.3%	mod_rank_read_file...	call mod_rank_read_file_all_its_own(str,nn,ios) ! Restart from last checkpoint	My_code.f90:297
12.8%	6.3%	velocity_solver	call velocity_solver	My_code.f90:337
1.8%		<unknown>	<unknown> (no debug info)	
1.5%	1.4%	vel_vort_3d_fp_	call cell_identifier	My_code.f90:190
4.1%	1.3%	91 others		

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?

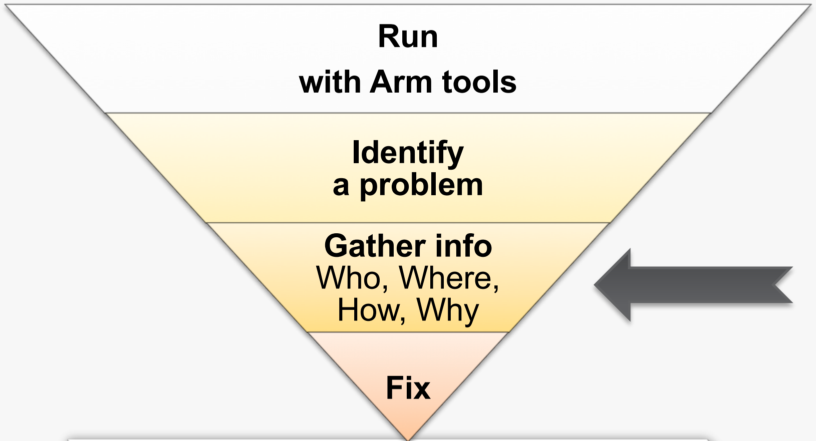
– Diagnostic messages

– Some faults evident instantly from source

Why did it happen?

– Unique “Smart Highlighting”

– Sparklines comparing data across processes

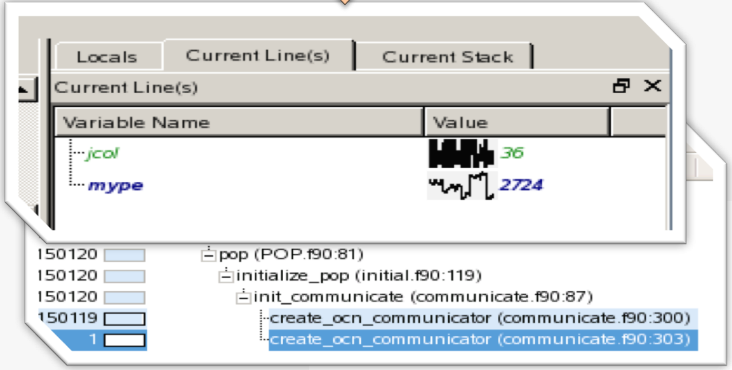


Run
with Arm tools

Identify
a problem

Gather info
Who, Where,
How, Why

Fix



Variable Name	Value
jcol	36
mype	2724

```
150120   pop (POP.f90:81)
150120     initialize_pop (initial.f90:119)
150120       init_communicate (communicate.f90:87)
150119         create_ocn_communicator (communicate.f90:300)
     1         create_ocn_communicator (communicate.f90:303)
```

Preparing Code for Use with DDT

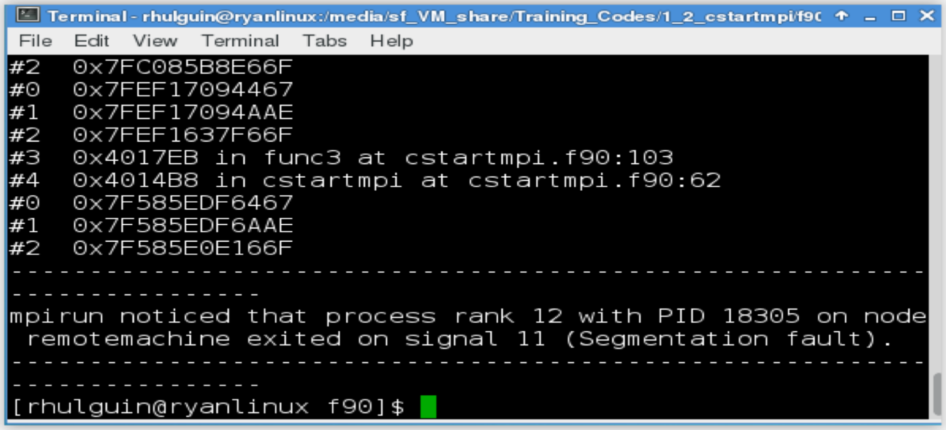
As with any debugger, code must be compiled with the debug flag typically `-g`

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

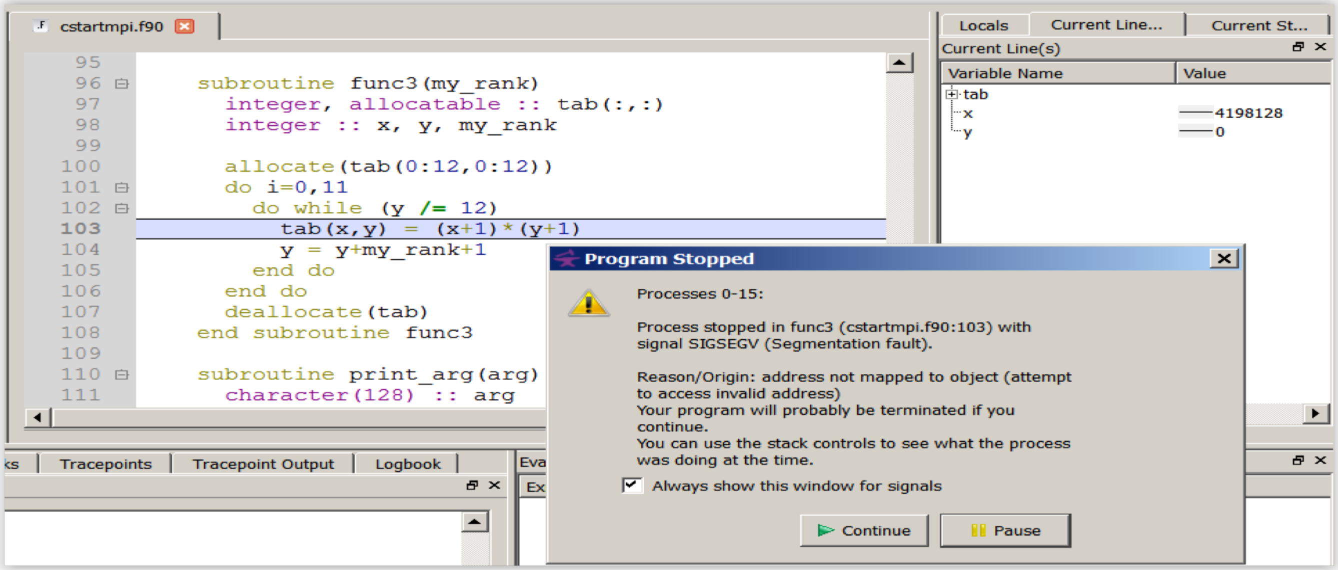
In this example, the application crashes with a segmentation error outside of DDT.


Terminal - rhulguin@ryanlinux:/media/sf_VM_share/Training_Codes/1_2_cstartmpi/f9(
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



```
 95
 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 func3
109
110       subroutine print_arg(arg)
111         character(128) :: arg
```

Current Line(s)

Variable Name	Value
tab	
x	4198128
y	0

Program Stopped

Processes 0-15:

Process stopped in func3 (cstartmpi.f90:103) with signal SIGSEGV (Segmentation fault).

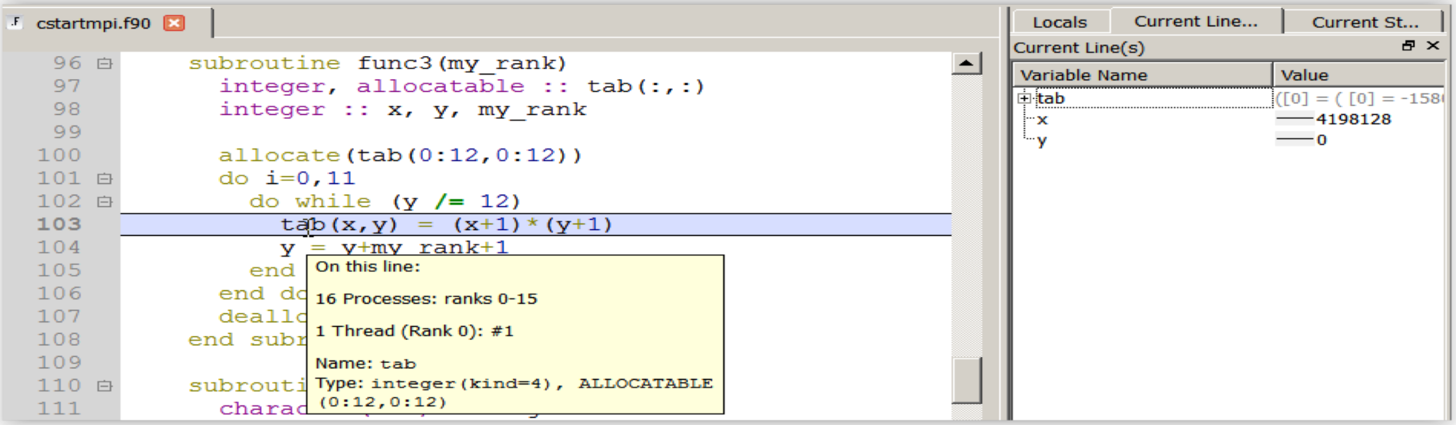
Reason/Origin: address not mapped to object (attempt to access invalid address)
Your program will probably be terminated if you continue.
You can use the stack controls to see what the process was doing at the time.

☑ Always show this window for signals

Continue | Pause

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

Invalid Memory Access



```
subroutine func3(my_rank)
  integer, allocatable :: tab(:,:)
  integer :: x, y, my_rank

  allocate(tab(0:12,0:12))
  do i=0,11
    do while (y /= 12)
      tab(x,y) = (x+1)*(y+1)
      y = y+my_rank+1
    end
  end dc
  deallc
end subr

subrouti
  charac
```

On this line:

16 Processes: ranks 0-15

1 Thread (Rank 0): #1

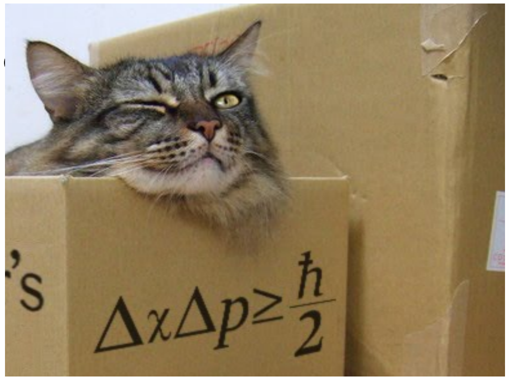
Name: tab
Type: integer(kind=4), ALLOCATABLE (0:12,0:12)

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



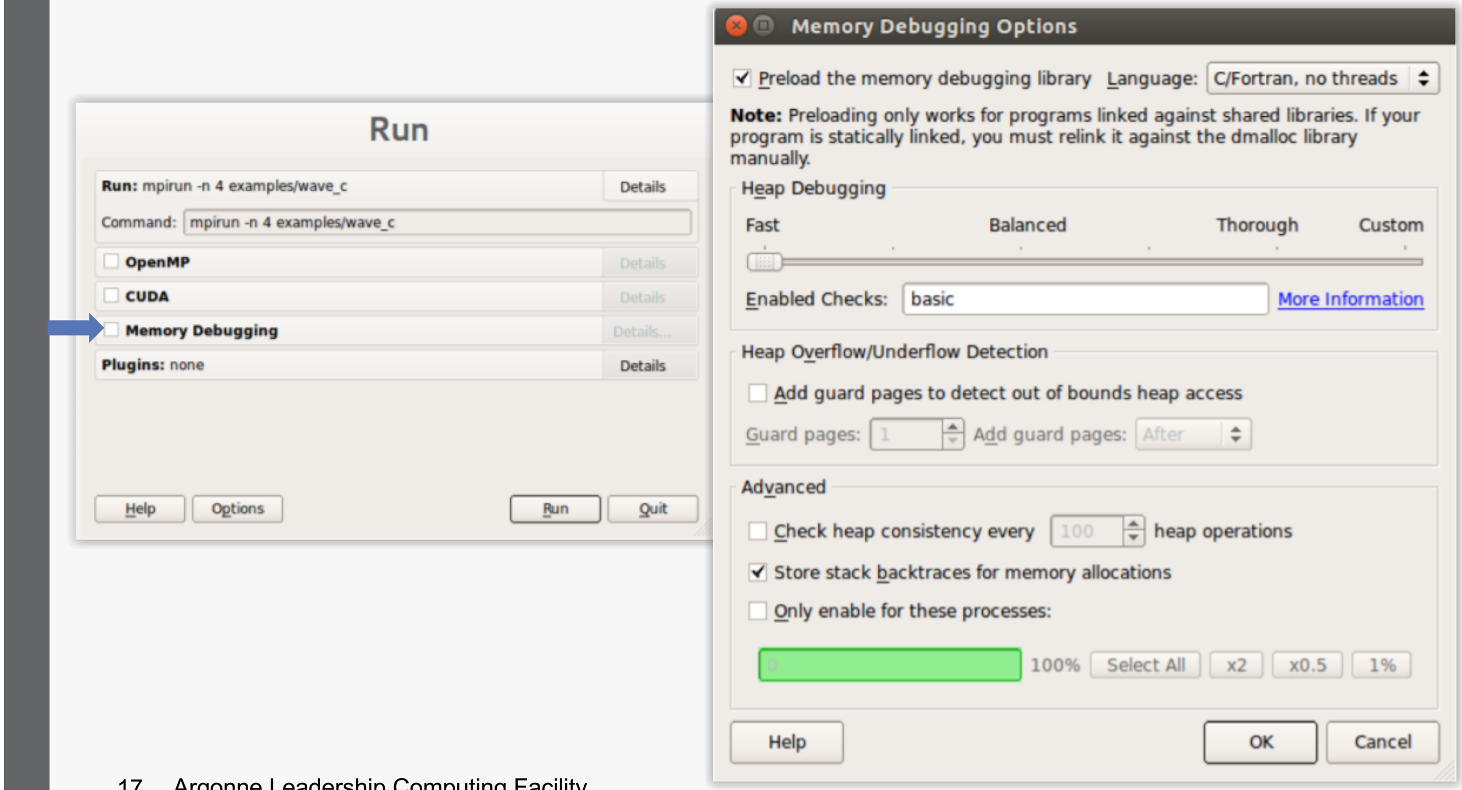
SCHRODIN
BUG

!

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



Run

Run: mpirun -n 4 examples/wave_c — Details

Command: mpirun -n 4 examples/wave_c

- □ **OpenMP** — Details
- □ **CUDA** — Details
- □ **Memory Debugging** — Details...

Plugins: none — Details

Memory Debugging Options

☑ Preload the memory debugging library Language: C/Fortran, no threads

Note: Preloading only works for programs linked against shared libraries. If your program is statically linked, you must relink it against the dmalloc library manually.

Heap Debugging

Fast — Balanced — Thorough — Custom

Enabled Checks: basic — More Information

Heap Overflow/Underflow Detection

□ Add guard pages to detect out of bounds heap access

Guard pages: 1 Add guard pages: After

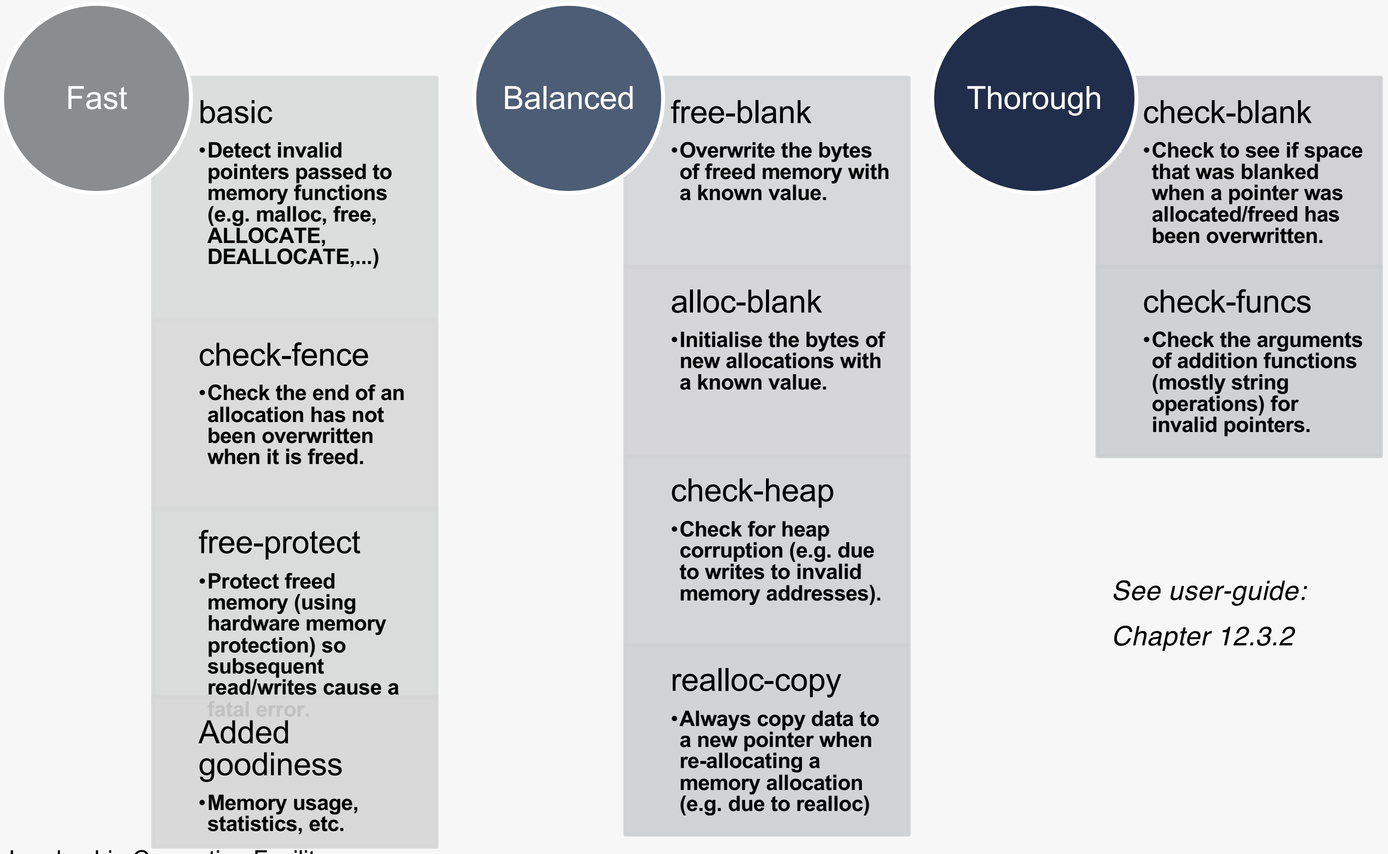
Advanced

□ Check heap consistency every 100 heap operations

☑ Store stack backtraces for memory allocations

□ Only enable for these processes:

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 goodiness

- 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.

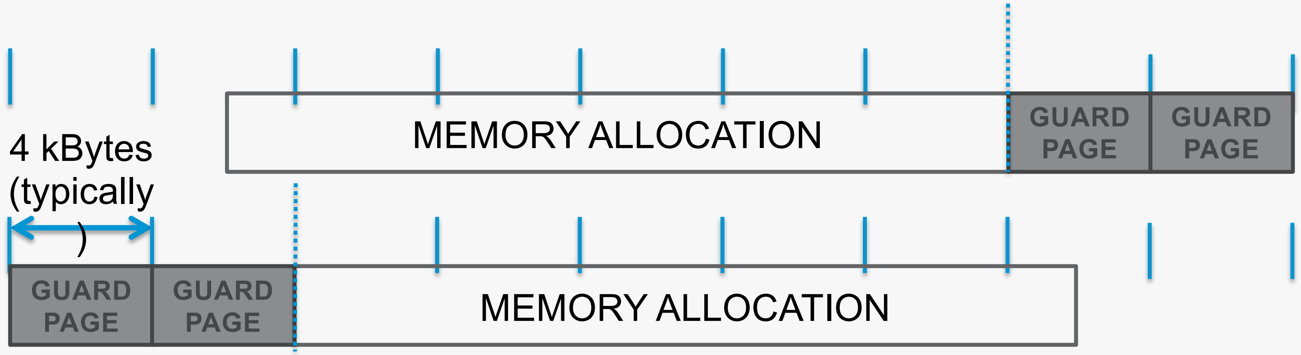
check-funcs

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

See user-guide:

Chapter 12.3.2

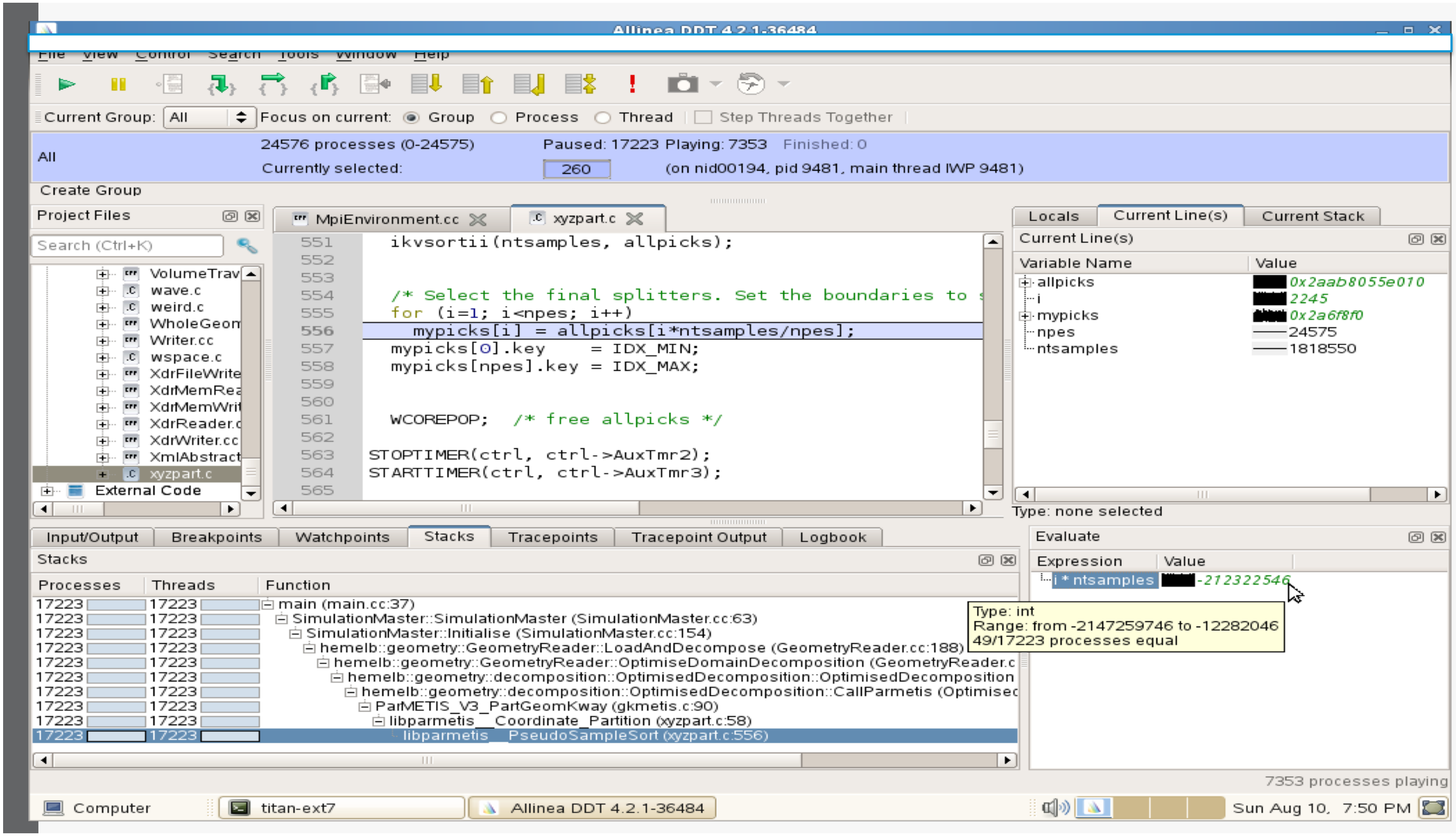
Guard pages (aka “Electric Fences”)



- **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 lWP 9481)

Create Group

Project Files

Search (Ctrl+K)

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

MpiEnvironment.cc   xyzpart.c

```
  ikvsortii(ntsamples, allpicks);


  /* Select the final splitters. Set the boundaries to s
  for (i=1; i<npes; i++)
    mypicks[i] = allpicks[i*ntsamples/npes];
  mypicks[0].key     = IDX_MIN;
  mypicks[npes].key = IDX_MAX;


  WCOREPOP;   /* free allpicks */

STOPTIMER(ctrl, ctrl->AuxTmr2);
STARTTIMER(ctrl, ctrl->AuxTmr3);
```

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 (Optimisec
17223	17223	ParMETIS_V3_PartGeomKway (gkmetis.c:90)
17223	17223	libparmetis__Coordinate_Partition (xyzpart.c:58)
17223	17223	libparmetis__PseudoSampleSort (xyzpart.c:556)

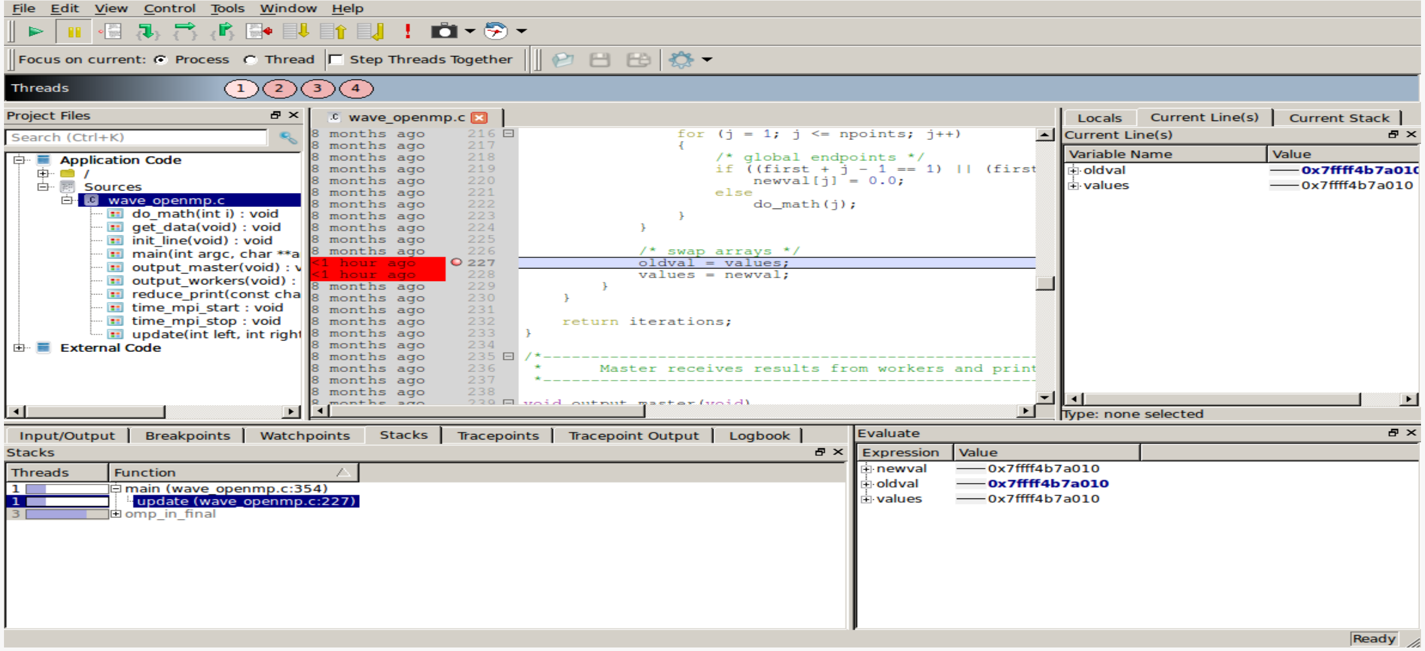
Evaluate

Expression	Value
i * ntsamples	-212322546

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

7353 processes playing

New Bugs from Latest Changes



```
8 months ago              for (j = 1; j <= npoints; j++)
8 months ago              {
8 months ago                  /* global endpoints */
8 months ago                  if ((first + j - 1 == 1) || (first
8 months ago                      newval[j] = 0.0;
8 months ago                  else
8 months ago                      do_math(j);
8 months ago              }
8 months ago          }
8 months ago
8 months ago          /* swap arrays */
<1 hour ago           oldval = values;
<1 hour ago           values = newval;
8 months ago      }
8 months ago  }
8 months ago
8 months ago  return iterations;
8 months ago  }
8 months ago
8 months ago  /*------------------------------------------------------------
8 months ago   *       Master receives results from workers and print
8 months ago   *------------------------------------------------------------
8 months ago
8 months ago  void output_master(void)
```

Current Line(s)

Variable Name	Value
oldval	**0x7ffff4b7a010**
values	0x7ffff4b7a010

Type: none selected

Stacks

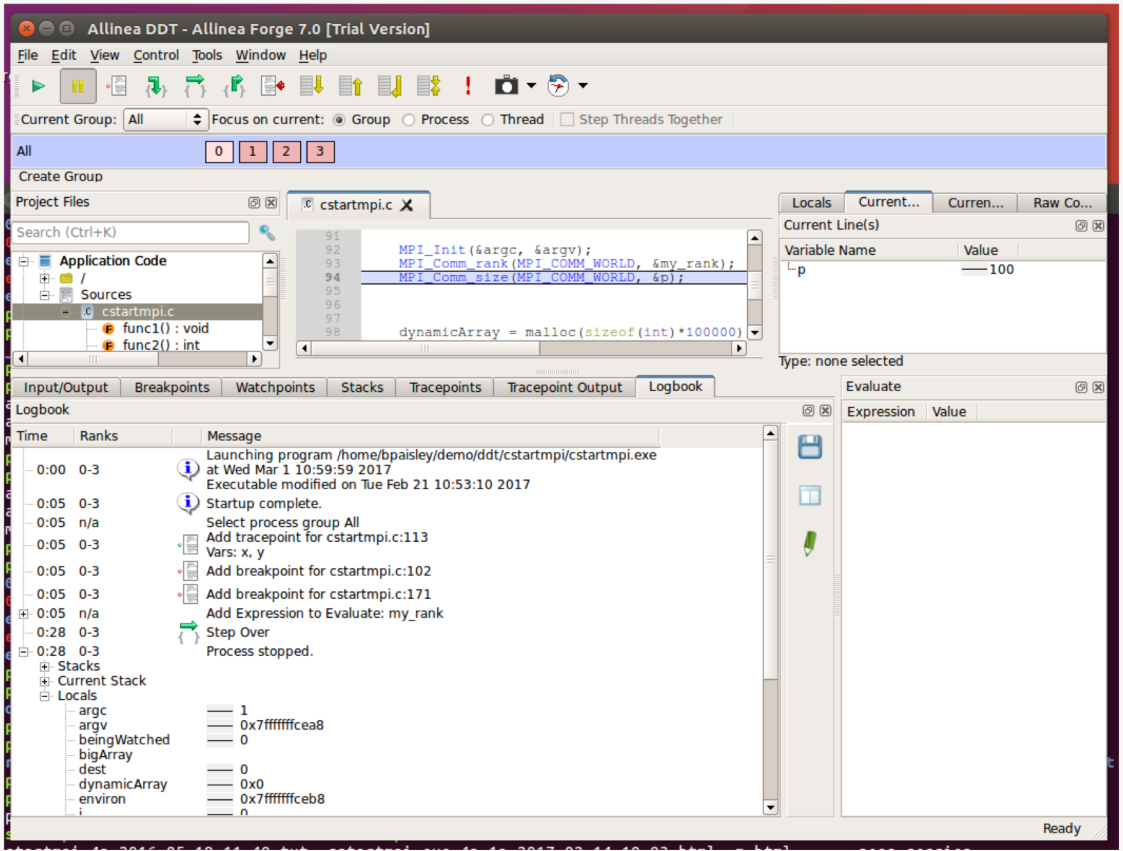
Threads	Function
1	main (wave_openmp.c:354)
1	update (wave_openmp.c:227)
3	omp_in_final

Evaluate

Expression	Value
newval	0x7ffff4b7a010
oldval	**0x7ffff4b7a010**
values	0x7ffff4b7a010

Track Your Changes in a Logbook

caption



```
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &p);



    dynamicArray = malloc(sizeof(int)*100000)
```

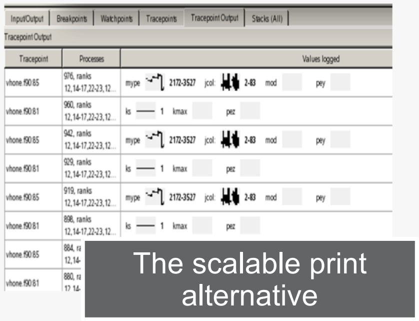
Variable Name	Value
p	100

Logbook

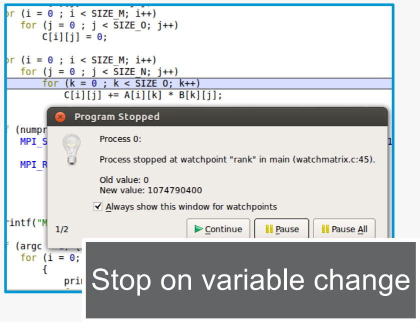
```
Time  Ranks   Message
                Launching program /home/bpaisley/demo/ddt/cstartmpi/cstartmpi.exe
0:00  0-3       at Wed Mar 1 10:59:59 2017
                Executable modified on Tue Feb 21 10:53:10 2017
0:05  0-3       Startup complete.
0:05  n/a       Select process group All
0:05  0-3       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.
  Stacks
  Current Stack
  Locals
    argc            1
    argv            0x7fffffffcea8
    beingWatched    0
    bigArray
    dest            0
    dynamicArray    0x0
    environ         0x7fffffffceb8
    i               0
```

Arm DDT Demo

Five great things to try with Allinea DDT



The scalable print alternative



```
or (i = 0 ; i < SIZE_M; i++)
   for (j = 0 ; j < SIZE_O; j++)
      C[i][j] = 0;

or (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];
```

Program Stopped

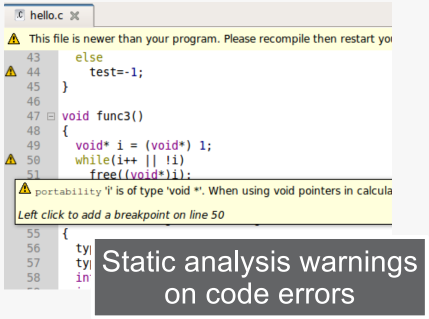
Process 0:

Process stopped at watchpoint "rank" in main (watchmatrix.c:45).

Old value: 0
New value: 1074790400

☑ Always show this window for watchpoints

Stop on variable change



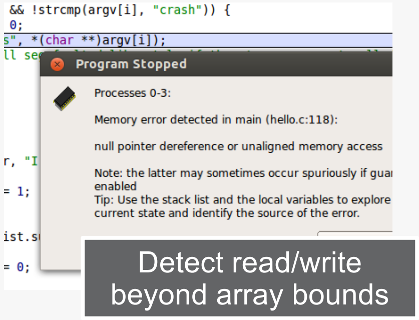
This file is newer than your program. Please recompile then restart yo

```
43     else
44       test=-1;
45   }
46
47   void func3()
48   {
49     void* i = (void*) 1;
50     while(i++ || !i)
51       free((void*)i);
55   {
56     ty
57     ty
58     in
```

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;
```

Program Stopped

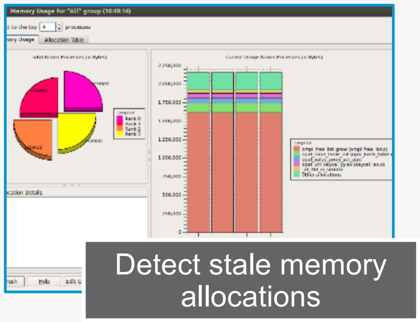
Processes 0-3:

Memory error detected in main (hello.c:118):

null pointer dereference or unaligned memory access

Note: the latter may sometimes occur spuriously if guar enabled
Tip: Use the stack list and the local variables to explore current state and identify the source of the error.

Detect read/write beyond array bounds



Detect stale memory allocations

Arm DDT cheat sheet

Load the environment module

- $ module load **forge/19.1.2**
- $ module unload **xalt**

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

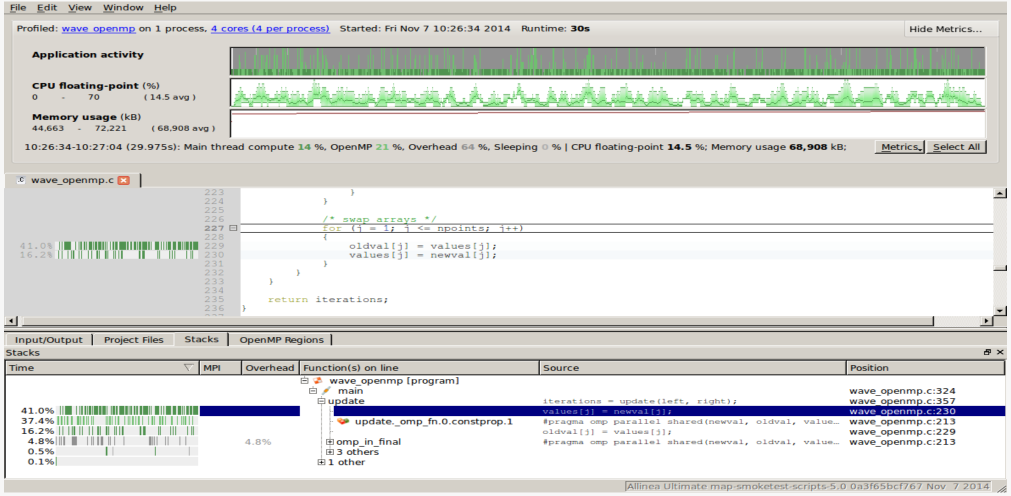
- On the login node:
  - $ ddt &
- (or use the remote client) **<- Preferred method**
- Then, edit the job script to run the following command and submit:
  - **ddt --connect** aprun -n 8 ./myapp.exe arg1 arg2

Profiling with MAP

arm

Arm MAP – The Profiler

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



Profiled: wave_openmp on 1 process, 4 cores (4 per process) Started: Fri Nov 7 10:26:34 2014 Runtime: 30s

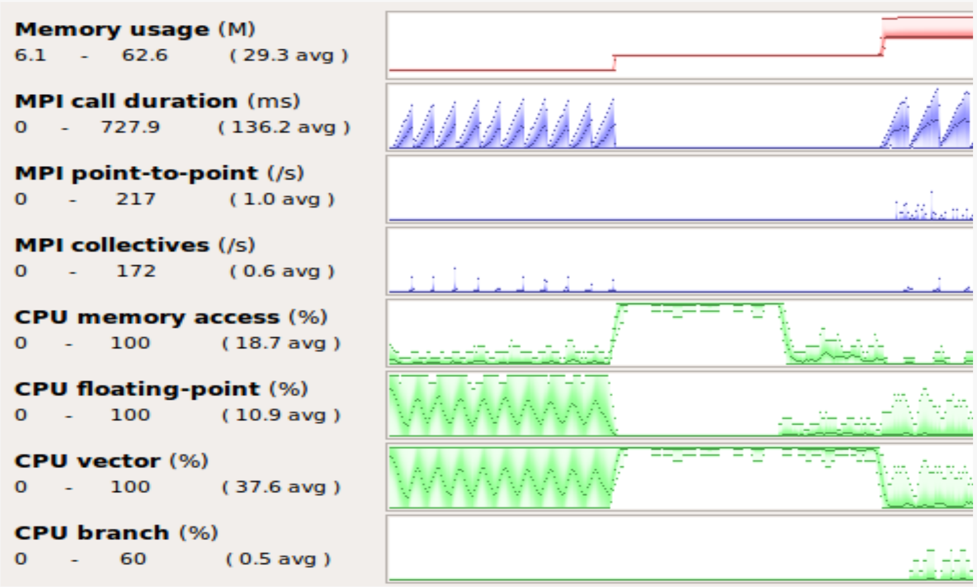
10:26:34-10:27:04 (29.975s): Main thread compute 14 %, OpenMP 21 %, Overhead 64 %, Sleeping 0 % | CPU floating-point 14.5 %; Memory usage 68,908 kB;

```
                        }
                }

                /* swap arrays */
                for (j = 1; j <= npoints; j++)
                {
41.0%                   oldval[j] = values[j];
16.2%                   values[j] = newval[j];
                }
            }
        }

        return iterations;
}
```

Time	MPI	Overhead	Function(s) on line	Source	Position
			wave_openmp [program]		
			main		wave_openmp.c:324
			update	iterations = update(left, right);	wave_openmp.c:357
41.0%				values[j] = newval[j];	wave_openmp.c:230
37.4%			update._omp_fn.0.constprop.1	#pragma omp parallel shared(newval, oldval, value…	wave_openmp.c:213
16.2%				oldval[j] = values[j];	wave_openmp.c:229
4.8%		4.8%	omp_in_final	#pragma omp parallel shared(newval, oldval, value…	wave_openmp.c:213
0.5%			3 others		
0.1%			1 other		

Allinea Ultimate map-smoketest-scripts-5.0 0a3f65bcf767 Nov 7 2014

Glean Deep Insight from our Source-Level Profiler



```
Memory usage (M)
6.1    -    62.6       ( 29.3 avg )
MPI call duration (ms)
0    -    727.9       ( 136.2 avg )
MPI point-to-point (/s)
0    -    217         ( 1.0 avg )
MPI collectives (/s)
0    -    172         ( 0.6 avg )
CPU memory access (%)
0    -    100         ( 18.7 avg )
CPU floating-point (%)
0    -    100         ( 10.9 avg )
CPU vector (%)
0    -    100         ( 37.6 avg )
CPU branch (%)
0    -    60          ( 0.5 avg )
```

Track memory usage across the entire application over time

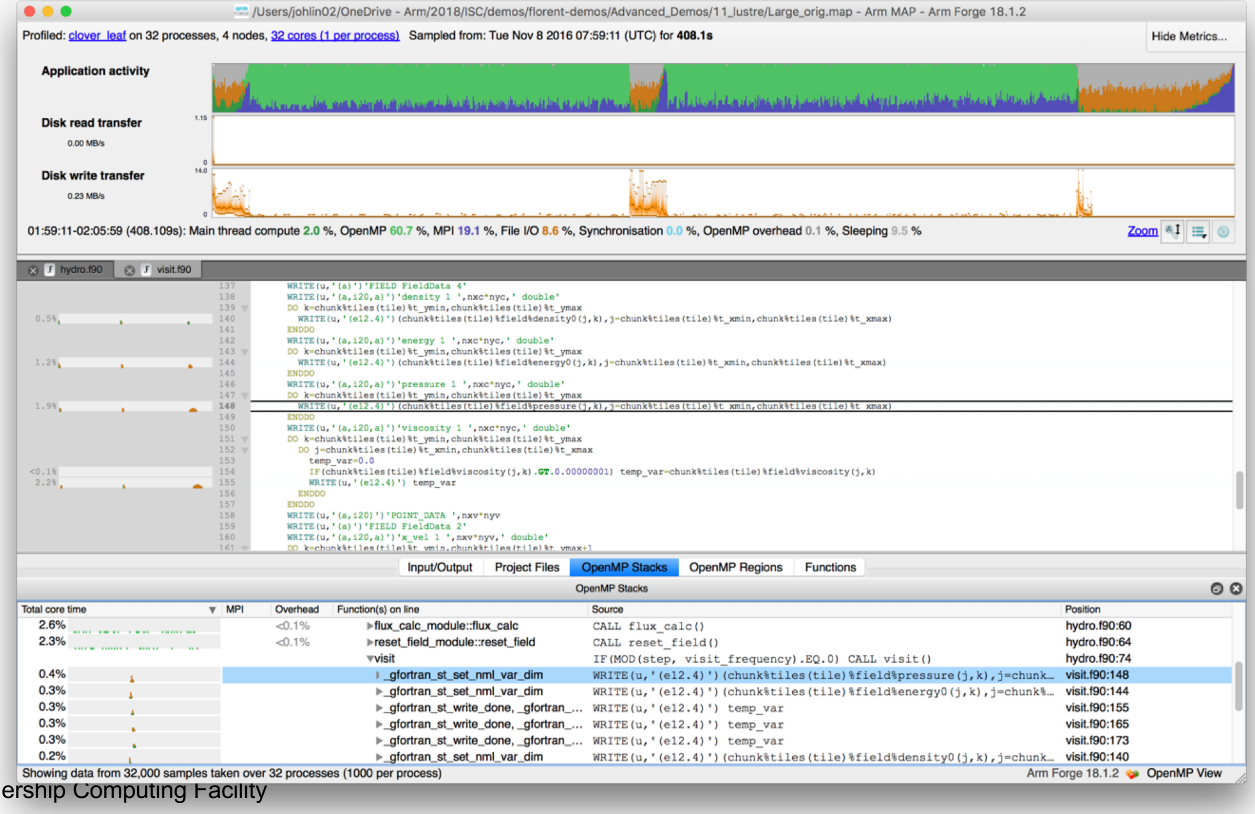
Spot MPI and OpenMP imbalance and overhead

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.



Profiled: clover_leaf on 32 processes, 4 nodes, 32 cores (1 per process) Sampled from: Tue Nov 8 2016 07:59:11 (UTC) for **408.1s**

01:59:11-02:05:59 (408.109s): Main thread compute 2.0 %, OpenMP 60.7 %, MPI 19.1 %, File I/O 8.6 %, Synchronisation 0.0 %, OpenMP overhead 0.1 %, Sleeping 9.5 %

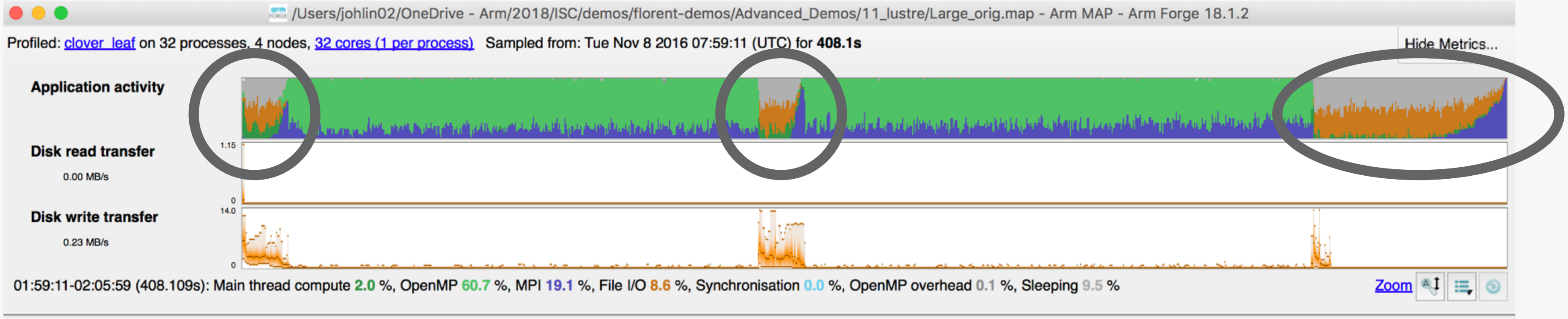
```
       137       WRITE(u,'(a)')'FIELD FieldData 4'
       138       WRITE(u,'(a,i20,a)')'density 1 ',nxc*nyc,' double'
       139       DO k=chunk%tiles(tile)%t_ymin,chunk%tiles(tile)%t_ymax
0.5%   140         WRITE(u,'(e12.4)')(chunk%tiles(tile)%field%density0(j,k),j=chunk%tiles(tile)%t_xmin,chunk%tiles(tile)%t_xmax)
       141       ENDDO
       142       WRITE(u,'(a,i20,a)')'energy 1 ',nxc*nyc,' double'
       143       DO k=chunk%tiles(tile)%t_ymin,chunk%tiles(tile)%t_ymax
1.2%   144         WRITE(u,'(e12.4)')(chunk%tiles(tile)%field%energy0(j,k),j=chunk%tiles(tile)%t_xmin,chunk%tiles(tile)%t_xmax)
       145       ENDDO
       146       WRITE(u,'(a,i20,a)')'pressure 1 ',nxc*nyc,' double'
       147       DO k=chunk%tiles(tile)%t_ymin,chunk%tiles(tile)%t_ymax
1.9%   148         WRITE(u,'(e12.4)')(chunk%tiles(tile)%field%pressure(j,k),j=chunk%tiles(tile)%t_xmin,chunk%tiles(tile)%t_xmax)
       149       ENDDO
       150       WRITE(u,'(a,i20,a)')'viscosity 1 ',nxc*nyc,' double'
       151       DO k=chunk%tiles(tile)%t_ymin,chunk%tiles(tile)%t_ymax
       152         DO j=chunk%tiles(tile)%t_xmin,chunk%tiles(tile)%t_xmax
       153           temp_var=0.0
<0.1%  154           IF(chunk%tiles(tile)%field%viscosity(j,k).GT.0.00000001) temp_var=chunk%tiles(tile)%field%viscosity(j,k)
2.2%   155           WRITE(u,'(e12.4)') temp_var
       156         ENDDO
       157       ENDDO
       158       WRITE(u,'(a,i20)')'POINT_DATA ',nxv*nyv
       159       WRITE(u,'(a)')'FIELD FieldData 2'
       160       WRITE(u,'(a,i20,a)')'x_vel 1 ',nxv*nyv,' double'
       161       DO k=chunk%tiles(tile)%t_ymin,chunk%tiles(tile)%t_ymax+1
```

Total core time	MPI	Overhead	Function(s) on line	Source	Position
2.6%		<0.1%	▶flux_calc_module::flux_calc	`CALL flux_calc()`	hydro.f90:60
2.3%		<0.1%	▶reset_field_module::reset_field	`CALL reset_field()`	hydro.f90:64
			▼visit	`IF(MOD(step, visit_frequency).EQ.0) CALL visit()`	hydro.f90:74
0.4%			_gfortran_st_set_nml_var_dim	`WRITE(u,'(e12.4)')(chunk%tiles(tile)%field%pressure(j,k),j=chunk…`	visit.f90:148
0.3%			▶_gfortran_st_set_nml_var_dim	`WRITE(u,'(e12.4)')(chunk%tiles(tile)%field%energy0(j,k),j=chunk%…`	visit.f90:144
0.3%			▶_gfortran_st_write_done, _gfortran_…	`WRITE(u,'(e12.4)') temp_var`	visit.f90:155
0.3%			▶_gfortran_st_write_done, _gfortran_…	`WRITE(u,'(e12.4)') temp_var`	visit.f90:165
0.3%			▶_gfortran_st_write_done, _gfortran_…	`WRITE(u,'(e12.4)') temp_var`	visit.f90:173
0.2%			▶_gfortran_st_set_nml_var_dim	`WRITE(u,'(e12.4)')(chunk%tiles(tile)%field%density0(j,k),j=chunk…`	visit.f90:140

Showing data from 32,000 samples taken over 32 processes (1000 per process)

Symptoms and causes of the I/O issues

Sub-optimal file format and surprise buffering.



/Users/johlin02/OneDrive - Arm/2018/ISC/demos/florent-demos/Advanced_Demos/11_lustre/Large_orig.map - Arm MAP - Arm Forge 18.1.2

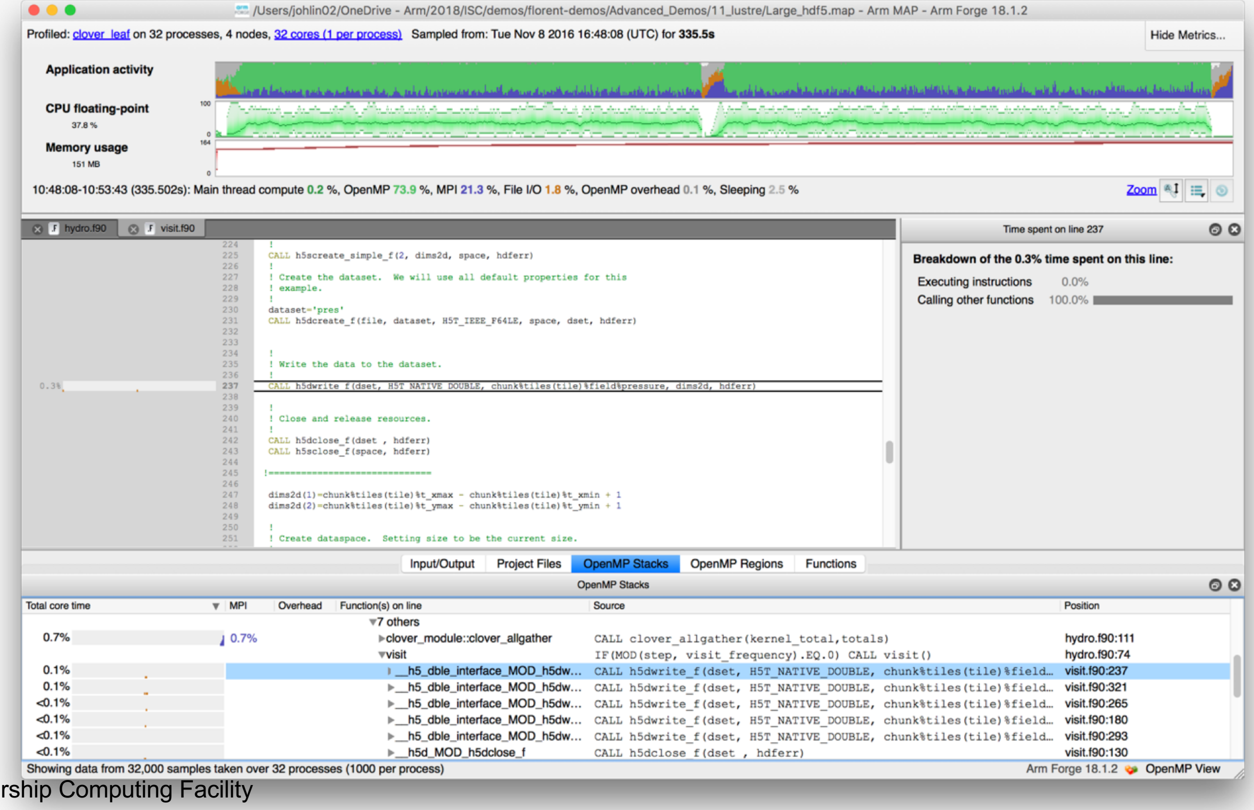
Profiled: clover_leaf on 32 processes, 4 nodes, 32 cores (1 per process) Sampled from: Tue Nov 8 2016 07:59:11 (UTC) for 408.1s

01:59:11-02:05:59 (408.109s): Main thread compute 2.0 %, OpenMP 60.7 %, MPI 19.1 %, File I/O 8.6 %, Synchronisation 0.0 %, OpenMP overhead 0.1 %, Sleeping 9.5 %

- 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.



/Users/johlin02/OneDrive - Arm/2018/ISC/demos/florent-demos/Advanced_Demos/11_lustre/Large_hdf5.map - Arm MAP - Arm Forge 18.1.2

Profiled: clover_leaf on 32 processes, 4 nodes, 32 cores (1 per process) Sampled from: Tue Nov 8 2016 16:48:08 (UTC) for **335.5s**

10:48:08-10:53:43 (335.502s): Main thread compute 0.2 %, OpenMP 73.9 %, MPI 21.3 %, File I/O 1.8 %, OpenMP overhead 0.1 %, Sleeping 2.5 %

```
!
CALL h5screate_simple_f(2, dims2d, space, hdferr)
!
! Create the dataset.  We will use all default properties for this
! example.
!
dataset='pres'
CALL h5dcreate_f(file, dataset, H5T_IEEE_F64LE, space, dset, hdferr)


!
! Write the data to the dataset.
!
CALL h5dwrite_f(dset, H5T_NATIVE_DOUBLE, chunk%tiles(tile)%field%pressure, dims2d, hdferr)

!
! Close and release resources.
!
CALL h5dclose_f(dset , hdferr)
CALL h5sclose_f(space, hdferr)

!=============================

dims2d(1)=chunk%tiles(tile)%t_xmax - chunk%tiles(tile)%t_xmin + 1
dims2d(2)=chunk%tiles(tile)%t_ymax - chunk%tiles(tile)%t_ymin + 1

!
! Create dataspace.  Setting size to be the current size.
```

Time spent on line 237

Breakdown of the 0.3% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%

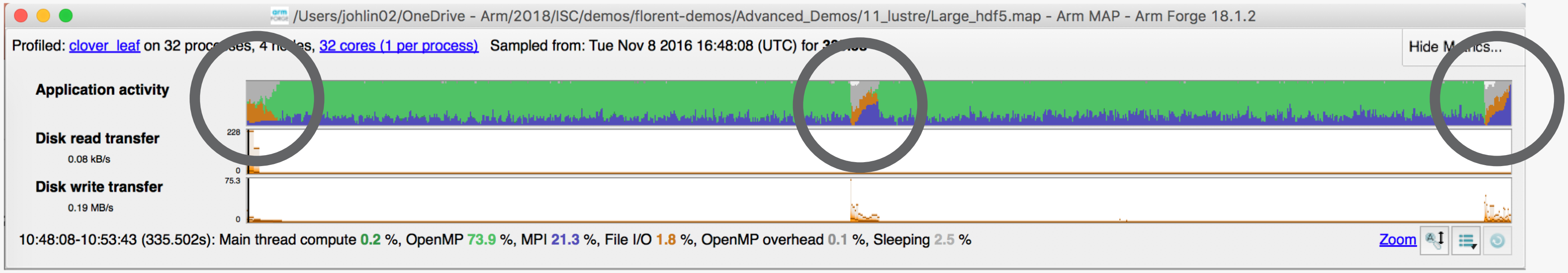
OpenMP Stacks

Total core time	MPI	Overhead	Function(s) on line	Source	Position
			▼7 others		
0.7%	0.7%		▶clover_module::clover_allgather	CALL clover_allgather(kernel_total,totals)	hydro.f90:111
			▼visit	IF(MOD(step, visit_frequency).EQ.0) CALL visit()	hydro.f90:74
0.1%			__h5_dble_interface_MOD_h5dw...	CALL h5dwrite_f(dset, H5T_NATIVE_DOUBLE, chunk%tiles(tile)%field...	visit.f90:237
0.1%			▶__h5_dble_interface_MOD_h5dw...	CALL h5dwrite_f(dset, H5T_NATIVE_DOUBLE, chunk%tiles(tile)%field...	visit.f90:321
<0.1%			▶__h5_dble_interface_MOD_h5dw...	CALL h5dwrite_f(dset, H5T_NATIVE_DOUBLE, chunk%tiles(tile)%field...	visit.f90:265
<0.1%			▶__h5_dble_interface_MOD_h5dw...	CALL h5dwrite_f(dset, H5T_NATIVE_DOUBLE, chunk%tiles(tile)%field...	visit.f90:180
<0.1%			▶__h5_dble_interface_MOD_h5dw...	CALL h5dwrite_f(dset, H5T_NATIVE_DOUBLE, chunk%tiles(tile)%field...	visit.f90:293
<0.1%			▶__h5d_MOD_h5dclose_f	CALL h5dclose_f(dset , hdferr)	visit.f90:130

Showing data from 32,000 samples taken over 32 processes (1000 per process)

Solution: use HDF5 to write binary files

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



/Users/johlin02/OneDrive - Arm/2018/ISC/demos/florent-demos/Advanced_Demos/11_lustre/Large_hdf5.map - Arm MAP - Arm Forge 18.1.2

Profiled: clover_leaf on 32 processes, 4 nodes, 32 cores (1 per process) Sampled from: Tue Nov 8 2016 16:48:08 (UTC) for 3[illegible]

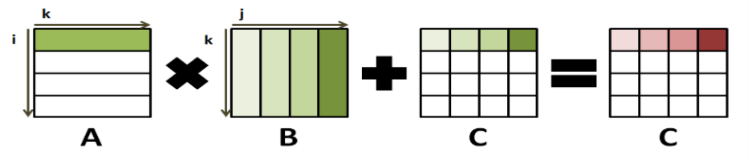
10:48:08-10:53:43 (335.502s): Main thread compute 0.2 %, OpenMP 73.9 %, MPI 21.3 %, File I/O 1.8 %, OpenMP overhead 0.1 %, Sleeping 2.5 %

- 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.

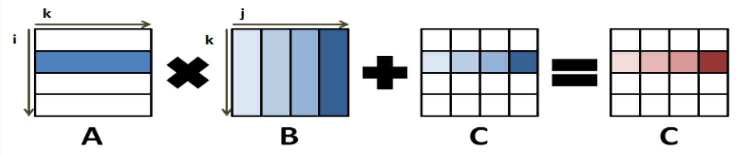
Matrix Multiplication Example

```
C = A x B + C
```

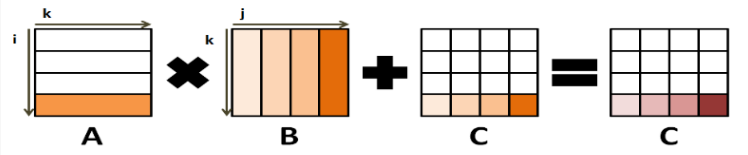
Master process



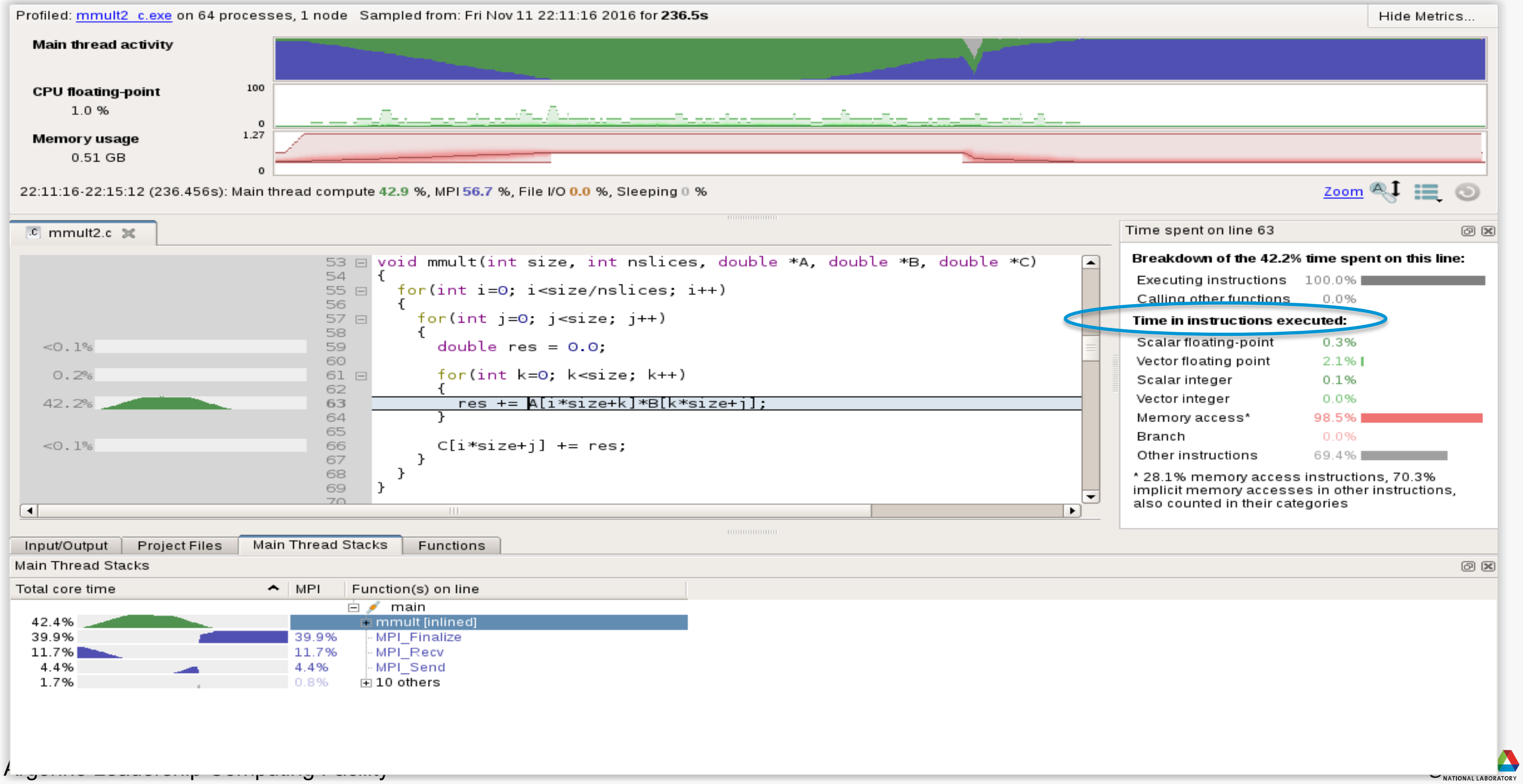
Slave process 1



Slave process n-1



Matrix Multiplication Profile



Profiled: mmult2_c.exe on 64 processes, 1 node   Sampled from: Fri Nov 11 22:11:16 2016 for **236.5s**

22:11:16-22:15:12 (236.456s): Main thread compute 42.9 %, MPI 56.7 %, File I/O 0.0 %, Sleeping 0 %

```
void mmult(int size, int nslices, double *A, double *B, double *C)
{
  for(int i=0; i<size/nslices; i++)
  {
    for(int j=0; j<size; j++)
    {
      double res = 0.0;

      for(int k=0; k<size; k++)
      {
        res += A[i*size+k]*B[k*size+j];
      }

      C[i*size+j] += res;
    }
  }
}
```

Time spent on line 63

Breakdown of the 42.2% time spent on this line:

Executing instructions	100.0%
Calling other functions	0.0%

Time in instructions executed:

Scalar floating-point	0.3%
Vector floating point	2.1%
Scalar integer	0.1%
Vector integer	0.0%
Memory access*	98.5%
Branch	0.0%
Other instructions	69.4%

* 28.1% memory access instructions, 70.3% implicit memory accesses in other instructions, also counted in their categories

Main Thread Stacks

Total core time	MPI	Function(s) on line
		main
42.4%		mmult [inlined]
39.9%	39.9%	MPI_Finalize
11.7%	11.7%	MPI_Recv
4.4%	4.4%	MPI_Send
1.7%	0.8%	10 others

Enabling Vectorization

The compiler is unable to vectorize efficiently because of the following line in C:

```
res += A[i*size+k]*B[k*size+j];
```

and in F90:

```
res=A(i*size+k)*B(k*size+j)+res
```

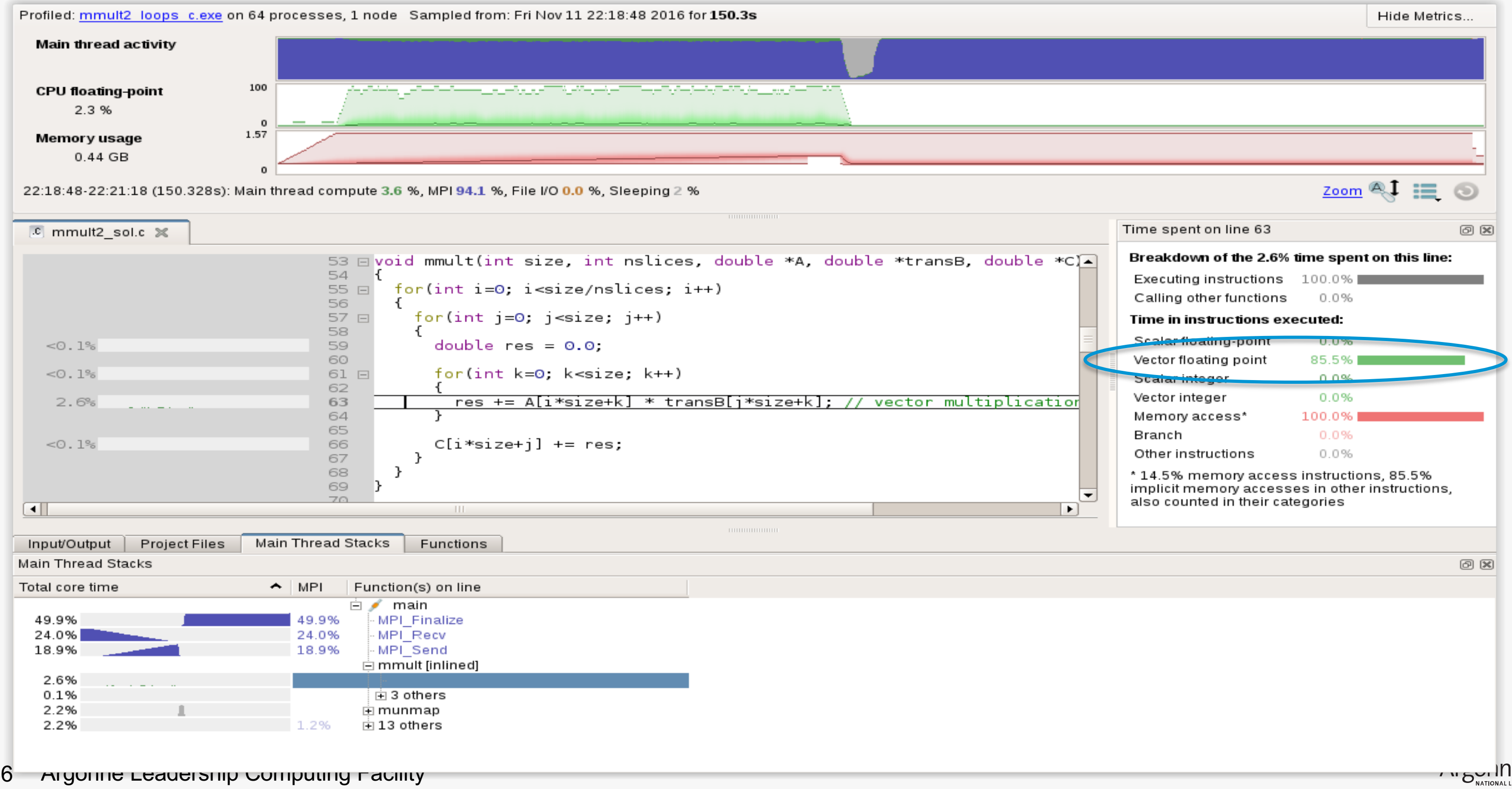
rewrite mmult to have
in C:

```
res += A[i*size+k]*transB[j*size+k];
```

and in F90:

```
res=A(i*size+k)*transB(j*size+k)+res
```

Improving Data Layout and Access Pattern



Profiled: mmult2_loops_c.exe on 64 processes, 1 node   Sampled from: Fri Nov 11 22:18:48 2016 for **150.3s**

22:18:48-22:21:18 (150.328s): Main thread compute 3.6 %, MPI 94.1 %, File I/O 0.0 %, Sleeping 2 %

```
          53 void mmult(int size, int nslices, double *A, double *transB, double *C)
          54 {
          55   for(int i=0; i<size/nslices; i++)
          56   {
          57     for(int j=0; j<size; j++)
          58     {
<0.1%     59       double res = 0.0;
          60
<0.1%     61       for(int k=0; k<size; k++)
          62       {
 2.6%     63         res += A[i*size+k] * transB[j*size+k]; // vector multiplicatio
          64       }
          65
<0.1%     66       C[i*size+j] += res;
          67     }
          68   }
          69 }
          70
```

Time spent on line 63

Breakdown of the 2.6% time spent on this line:

Executing instructions	100.0%
Calling other functions	0.0%

Time in instructions executed:

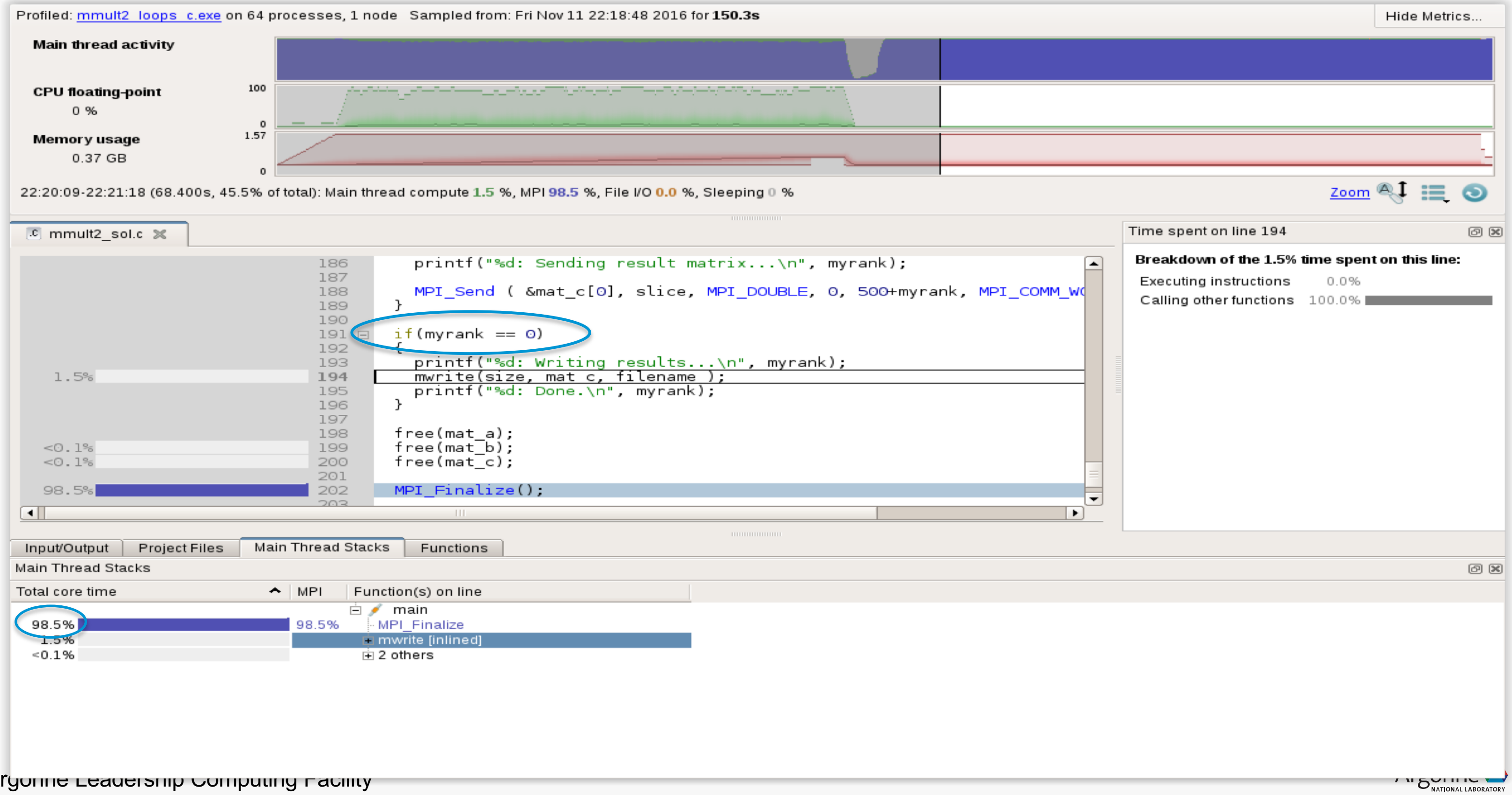
Scalar floating-point	0.0%
Vector floating point	85.5%
Scalar integer	0.0%
Vector integer	0.0%
Memory access*	100.0%
Branch	0.0%
Other instructions	0.0%

* 14.5% memory access instructions, 85.5% implicit memory accesses in other instructions, also counted in their categories

Main Thread Stacks

```
Total core time    MPI      Function(s) on line
                            ⊟ main
  49.9%            49.9%      MPI_Finalize
  24.0%            24.0%      MPI_Recv
  18.9%            18.9%      MPI_Send
                              ⊟ mmult [inlined]
   2.6%
   0.1%                         ⊞ 3 others
   2.2%                       ⊞ munmap
   2.2%             1.2%      ⊞ 13 others
```

Serial Bottleneck



Profiled: mmult2_loops_c.exe on 64 processes, 1 node   Sampled from: Fri Nov 11 22:18:48 2016 for **150.3s**

22:20:09-22:21:18 (68.400s, 45.5% of total): Main thread compute 1.5 %, MPI 98.5 %, File I/O 0.0 %, Sleeping 0 %

```
                  186      printf("%d: Sending result matrix...\n", myrank);
                  187
                  188      MPI_Send ( &mat_c[0], slice, MPI_DOUBLE, 0, 500+myrank, MPI_COMM_WO
                  189    }
                  190
                  191    if(myrank == 0)
                  192    {
                  193      printf("%d: Writing results...\n", myrank);
   1.5%           194      mwrite(size, mat_c, filename );
                  195      printf("%d: Done.\n", myrank);
                  196    }
                  197
                  198    free(mat_a);
  <0.1%           199    free(mat_b);
  <0.1%           200    free(mat_c);
                  201
  98.5%           202    MPI_Finalize();
```

Time spent on line 194

Breakdown of the 1.5% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%

Main Thread Stacks

Total core time	MPI	Function(s) on line
		main
98.5%	98.5%	MPI_Finalize
1.5%		mwrite [inlined]
<0.1%		2 others

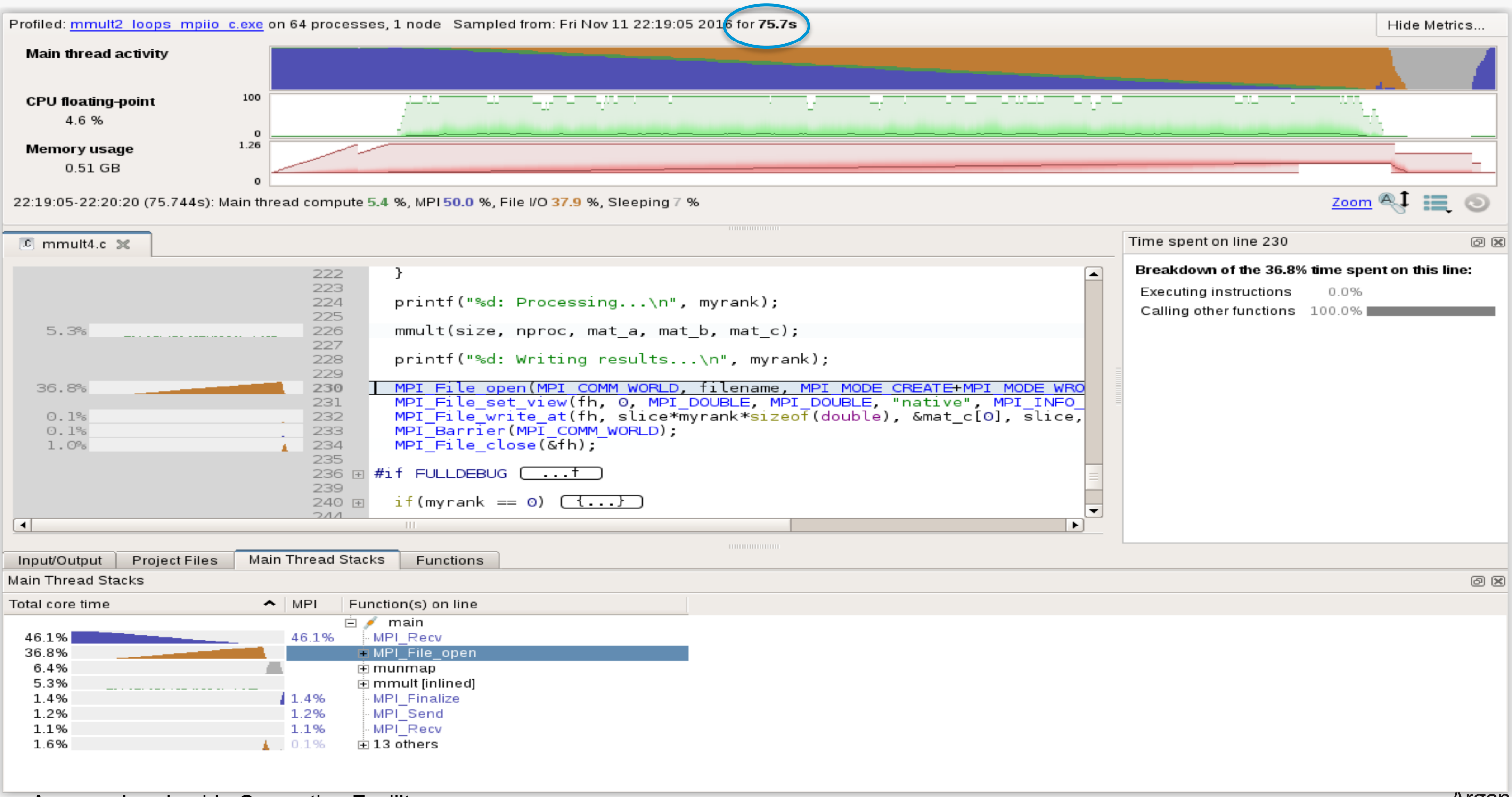
Inefficient I/O

```
if(myrank == 0)
{
  printf("%d: Receiving result matrix...\n", myrank);
  […]
}
else
{
  printf("%d: Sending result matrix...\n", myrank);
  […]
}

if(myrank == 0)
{
  printf("%d: Writing results...\n", myrank);
  mwrite(size, mat_c, filename);
}
```

Improve Scalability of I/O Routines

```
printf("%d: Writing results...\n", myrank);

MPI_File_open(MPI_COMM_WORLD, filename, MPI_MODE_CREATE+MPI_MODE_WRONLY, MPI_INFO_NULL,
&fh);
MPI_File_set_view(fh, 0, MPI_DOUBLE, MPI_DOUBLE, "native", MPI_INFO_NULL);
MPI_File_write_at(fh, slice*myrank*sizeof(double), &mat_c[0], slice, MPI_DOUBLE, &st);
MPI_Barrier(MPI_COMM_WORLD);
MPI_File_close(&fh);
```

3x Speedup from Original Code



Profiled: mmult2_loops_mpiio_c.exe on 64 processes, 1 node   Sampled from: Fri Nov 11 22:19:05 2016 for **75.7s**

22:19:05-22:20:20 (75.744s): Main thread compute 5.4 %, MPI 50.0 %, File I/O 37.9 %, Sleeping 7 %

```
        222    }
        223
        224    printf("%d: Processing...\n", myrank);
        225
 5.3%   226    mmult(size, nproc, mat_a, mat_b, mat_c);
        227
        228    printf("%d: Writing results...\n", myrank);
        229
36.8%   230    MPI_File_open(MPI_COMM_WORLD, filename, MPI_MODE_CREATE+MPI_MODE_WRO
        231    MPI_File_set_view(fh, 0, MPI_DOUBLE, MPI_DOUBLE, "native", MPI_INFO_
 0.1%   232    MPI_File_write_at(fh, slice*myrank*sizeof(double), &mat_c[0], slice,
 0.1%   233    MPI_Barrier(MPI_COMM_WORLD);
 1.0%   234    MPI_File_close(&fh);
        235
        236 ⊞ #if FULLDEBUG  ...
        239
        240 ⊞  if(myrank == 0)  {...}
```

Time spent on line 230

Breakdown of the 36.8% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%

Main Thread Stacks

Total core time	MPI	Function(s) on line
		⊟ main
46.1%	46.1%	MPI_Recv
36.8%		⊞ MPI_File_open
6.4%		⊞ munmap
5.3%		⊞ mmult [inlined]
1.4%	1.4%	MPI_Finalize
1.2%	1.2%	MPI_Send
1.1%	1.1%	MPI_Recv
1.6%	0.1%	⊞ 13 others

Arm MAP cheat sheet

Load the environment module (manually specify version)

- $ module load **forge/19.1.2**

Generate the wrapper libraries (static is default on Theta)

- $ make-profiler-libraries --platform=default --lib-type=static

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

- $ module unload darshan

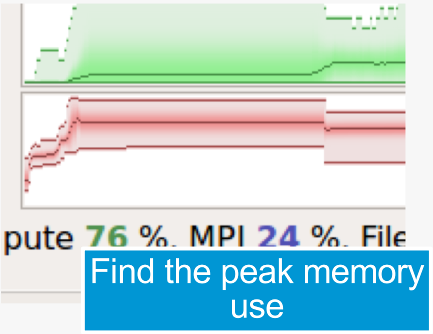
Follow the instructions displayed to prepare the code

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

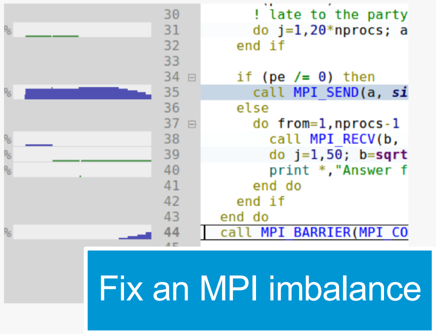
Open the results

- 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

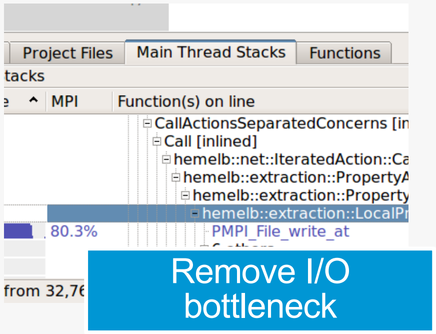


Find the peak memory use



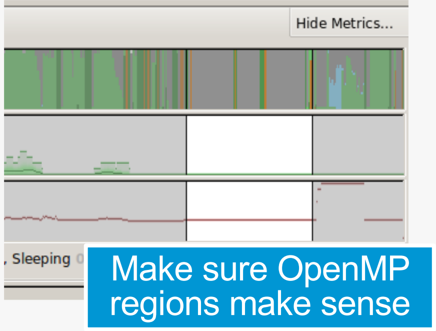
```
    ! late to the party
    do j=1,20*nprocs; a
  end if

  if (pe /= 0) then
    call MPI_SEND(a, si
  else
    do from=1,nprocs-1
      call MPI_RECV(b,
      do j=1,50; b=sqrt
      print *,"Answer f
    end do
  end if
end do
call MPI_BARRIER(MPI_CO
```

Fix an MPI imbalance

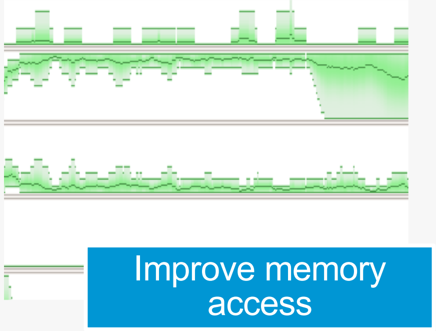


```
⊟CallActionsSeparatedConcerns [in
 ⊟Call [inlined]
  ⊟hemelb::net::IteratedAction::Ca
   ⊟hemelb::extraction::PropertyA
    ⊟hemelb::extraction::Property
     ⊟hemelb::extraction::LocalPr
80.3%   PMPI_File_write_at
from 32,76
```

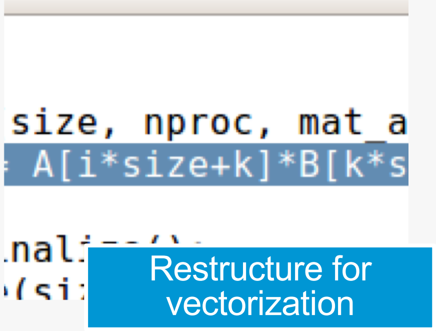
Remove I/O bottleneck



Make sure OpenMP regions make sense



Improve memory access



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

.nali
(si
```

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-19.1.2-2019-08-06
```

Static Linking Extra Steps

To enable advanced memory debugging features in DDT, 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 with MAP, 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