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!

–**T**rack the problem

–**R**eproduce

–**A**utomate - (and simplify) the test case

–**F**ind origins – where could the “infection” be from?

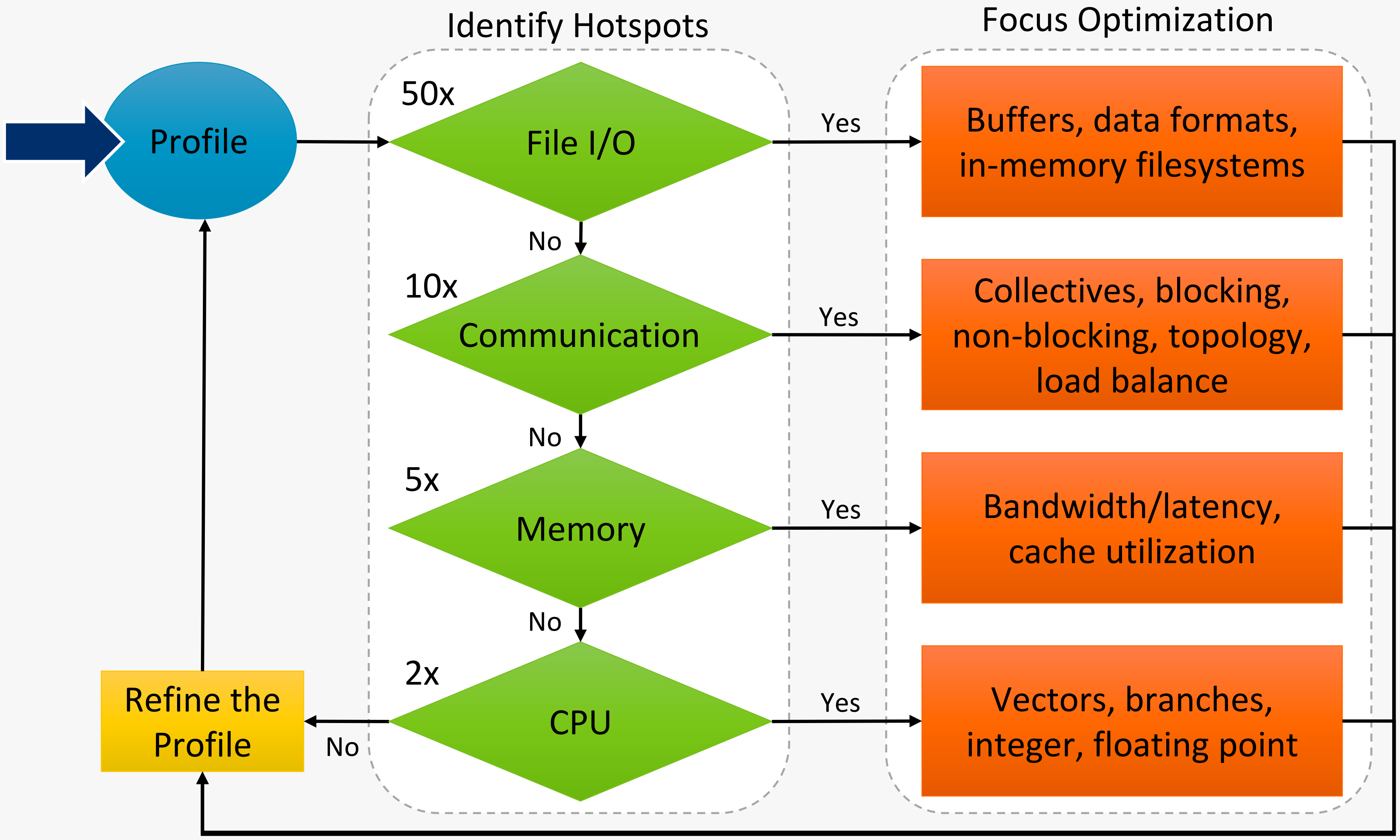
–**F**ocus – examine the origins

–**I**solate – narrow down the origins

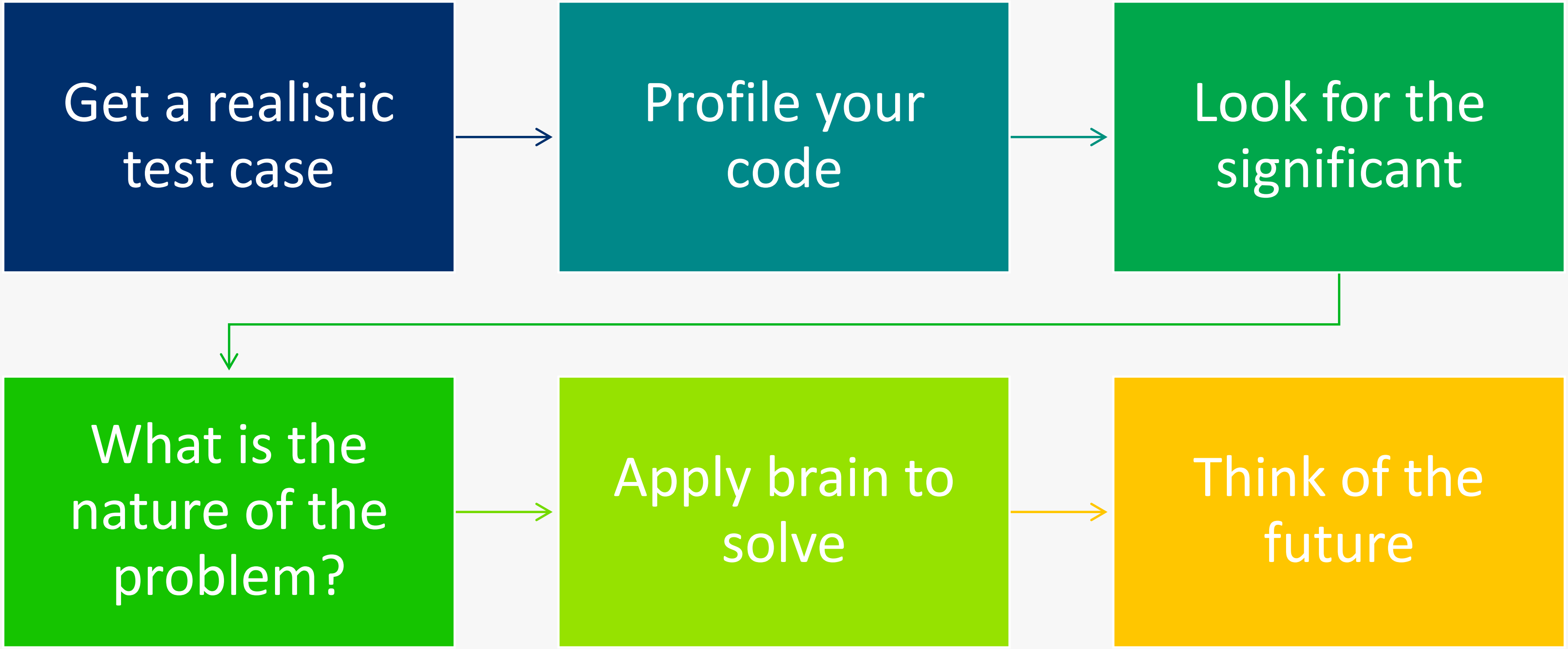
–**C**orrect – fix and verify the test case is successful

Profiling

Profiling is central to understanding and improving application performance.



Performance Improvement Workflow



Arm Software

arm

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)

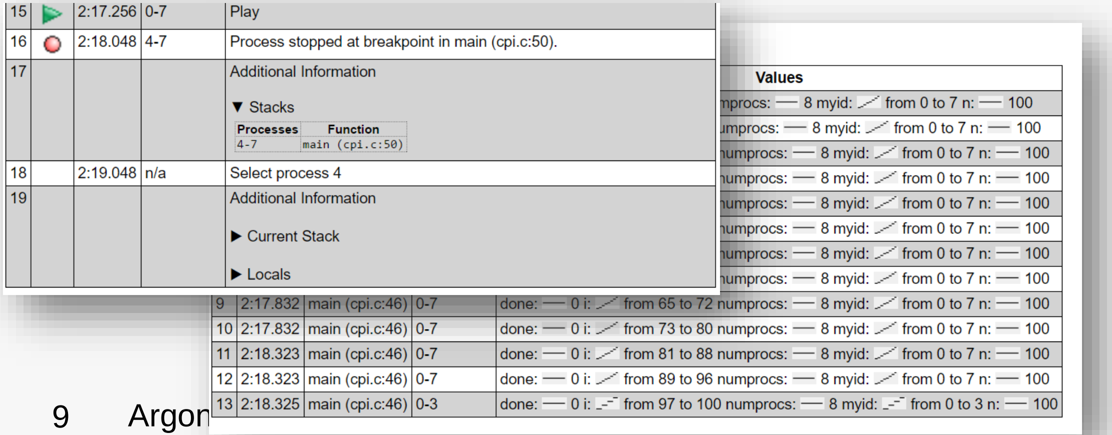
Run and ensure application correctness

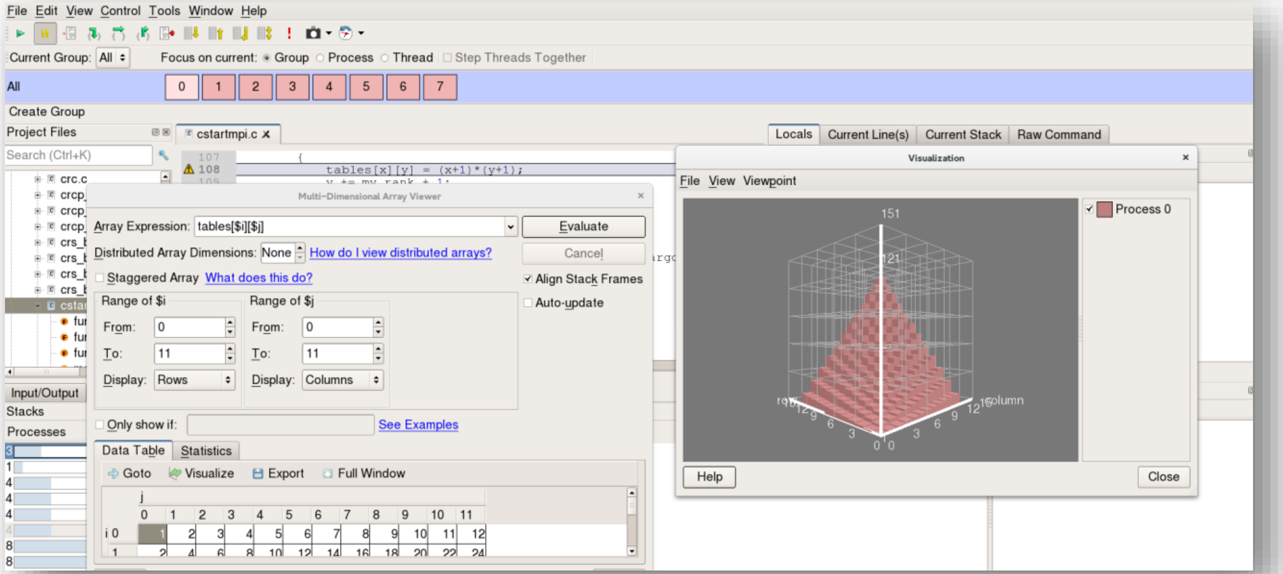
Combination of debugging and re-compilation

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

<u>Examples:</u>

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





Argonne NATIONAL LABORATORY

Understand application behaviour

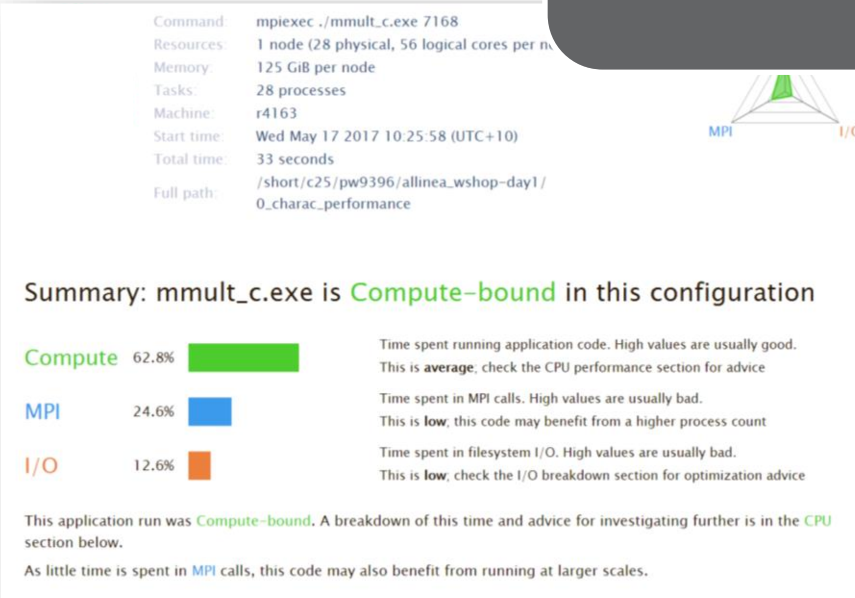
Is it performant?

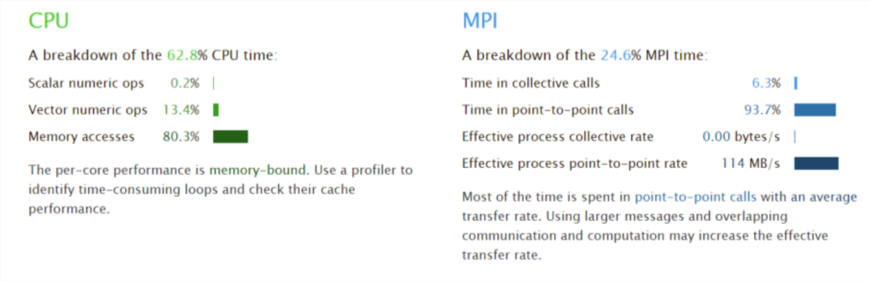
Set a reference for future work

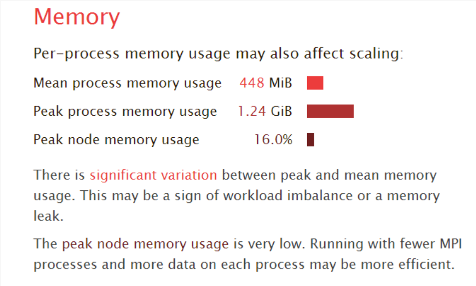
- Choose a representative test case with known behavior
- Analyse performance with **Arm Performance Reports**

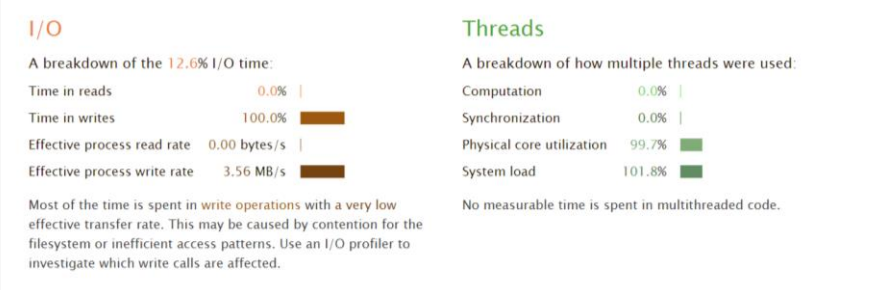
Example:

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









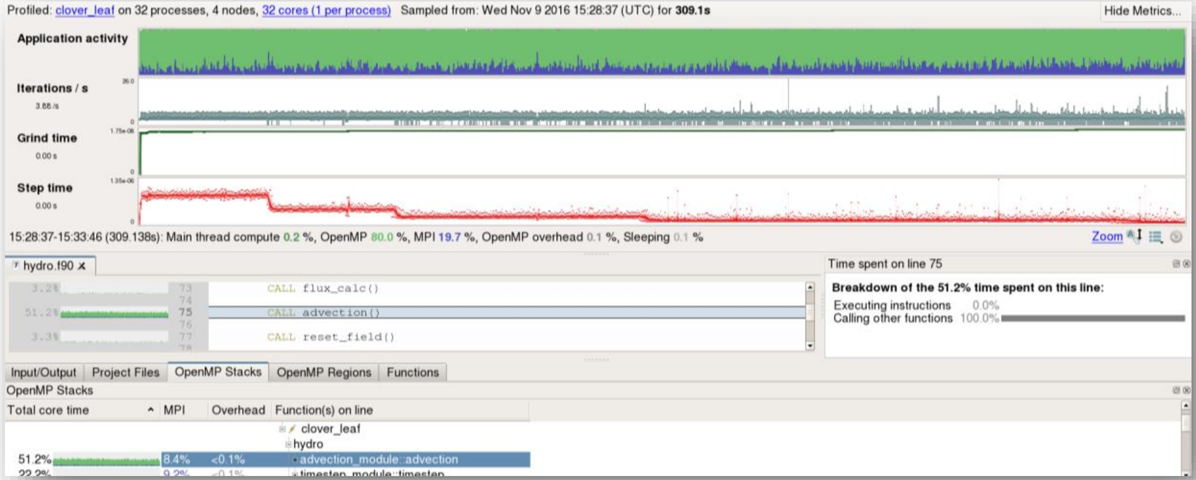
Optimize the application for Arm

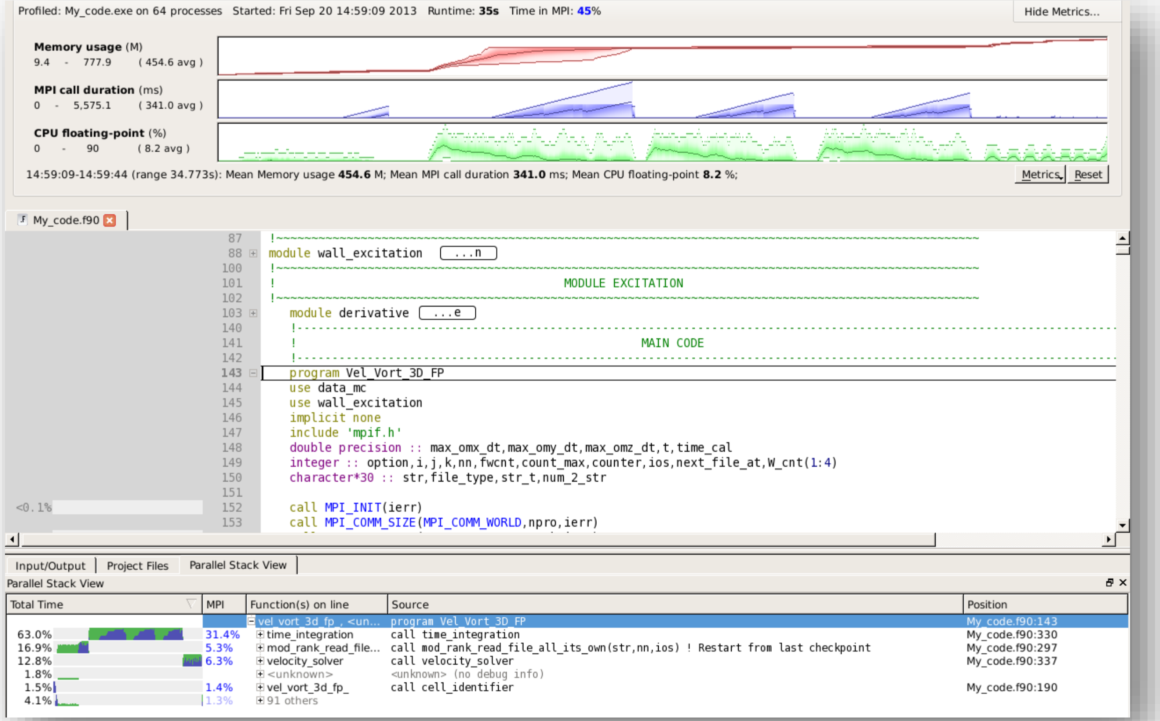
if not, use the Arm MAP profiler for optimization

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

Examples:

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

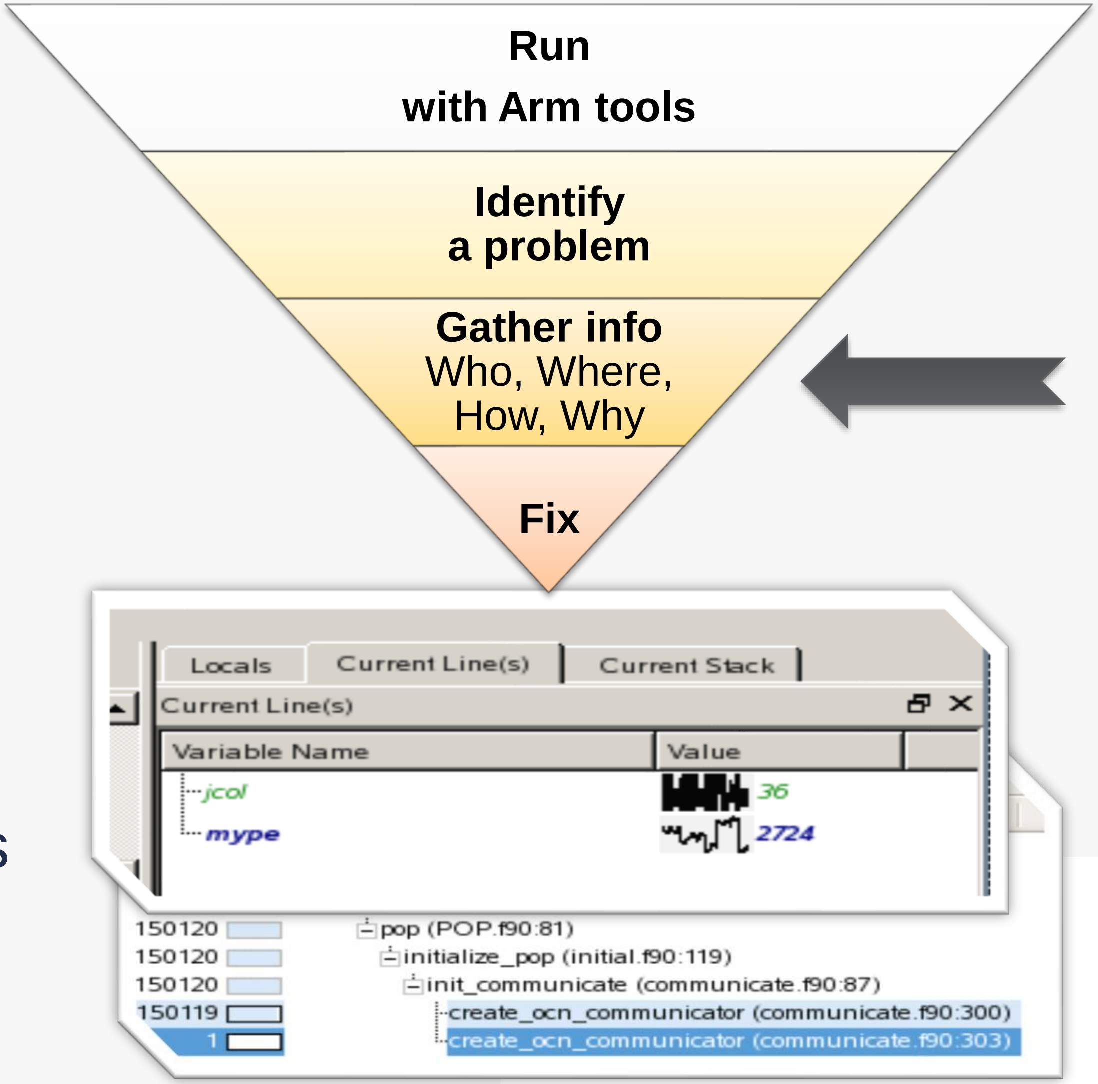




Debugging with DDT

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

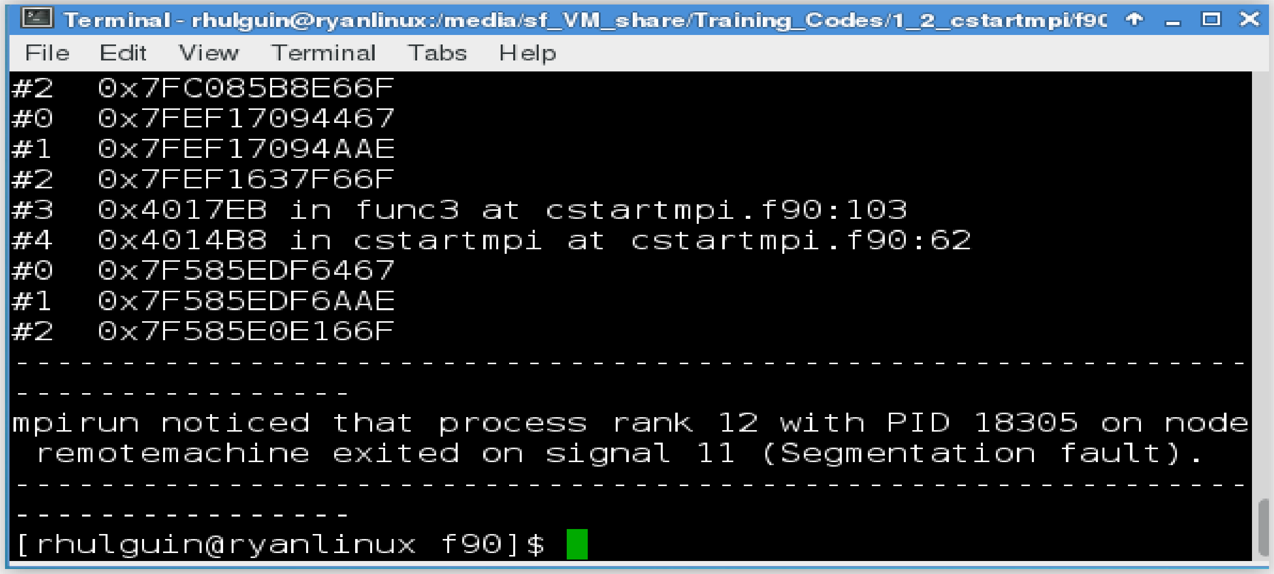
As with any debugger, code must be compiled with the debug flag typically **-g**

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

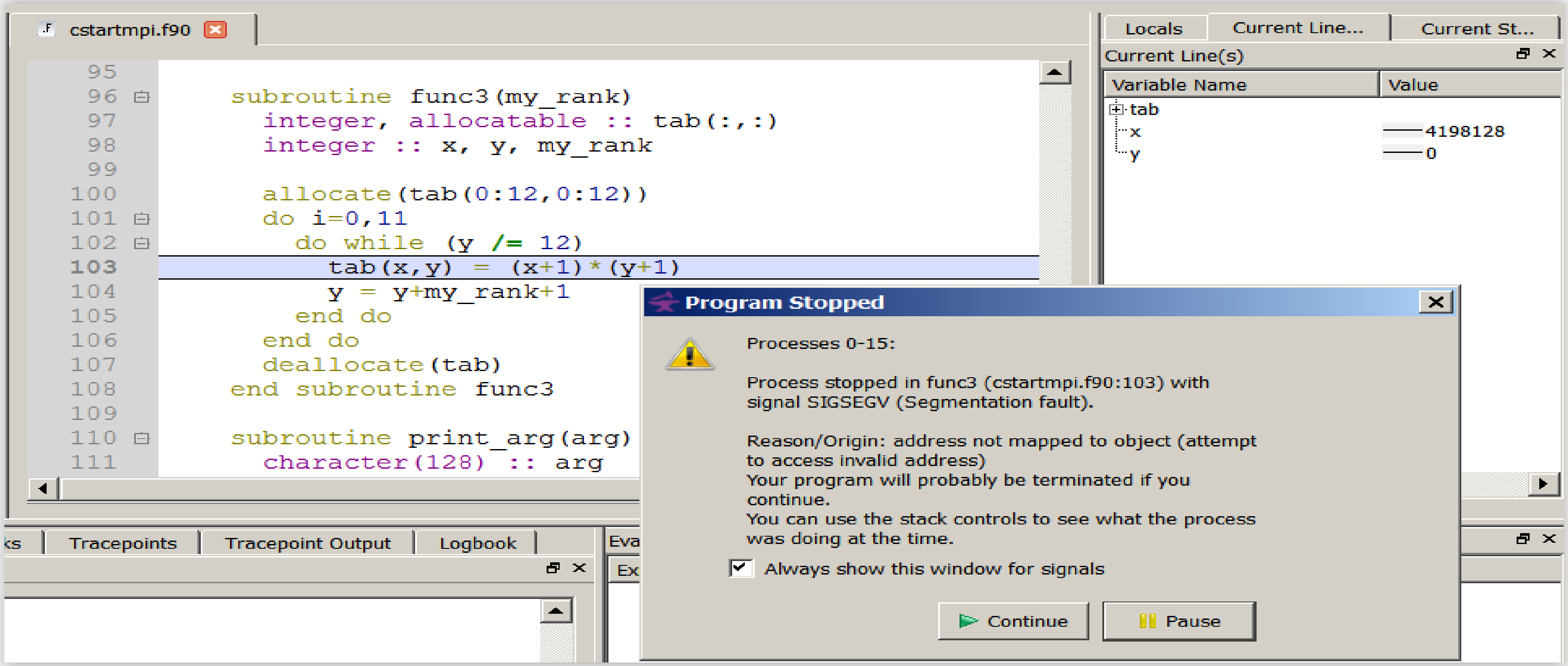
Segmentation Fault

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



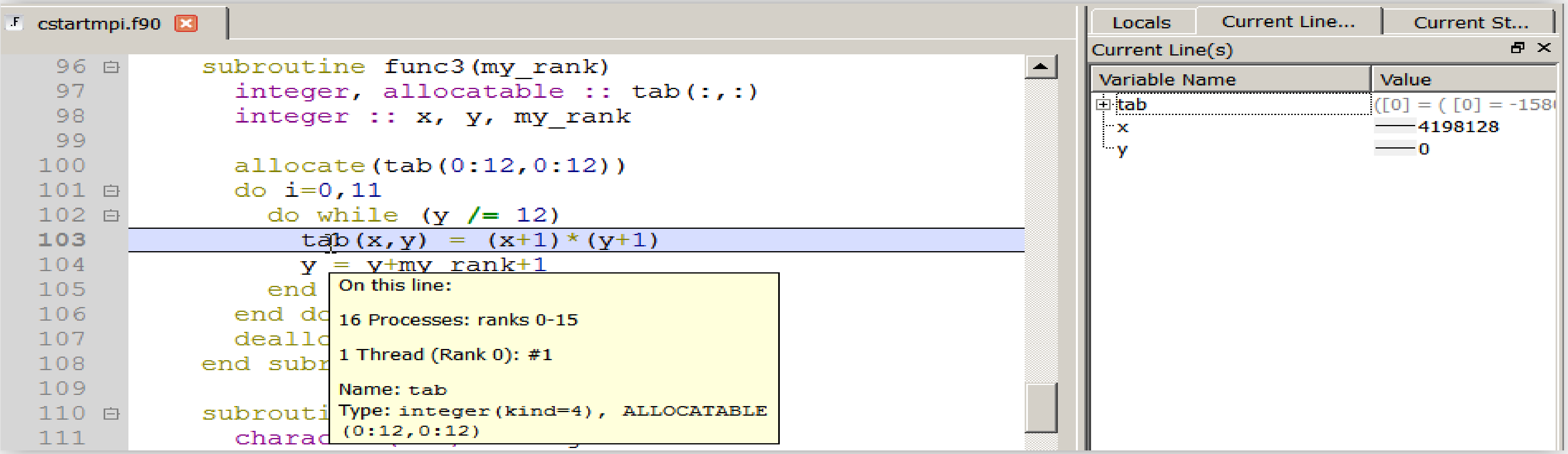
What happens when it runs under DDT?

Segmentation Fault in DDT



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

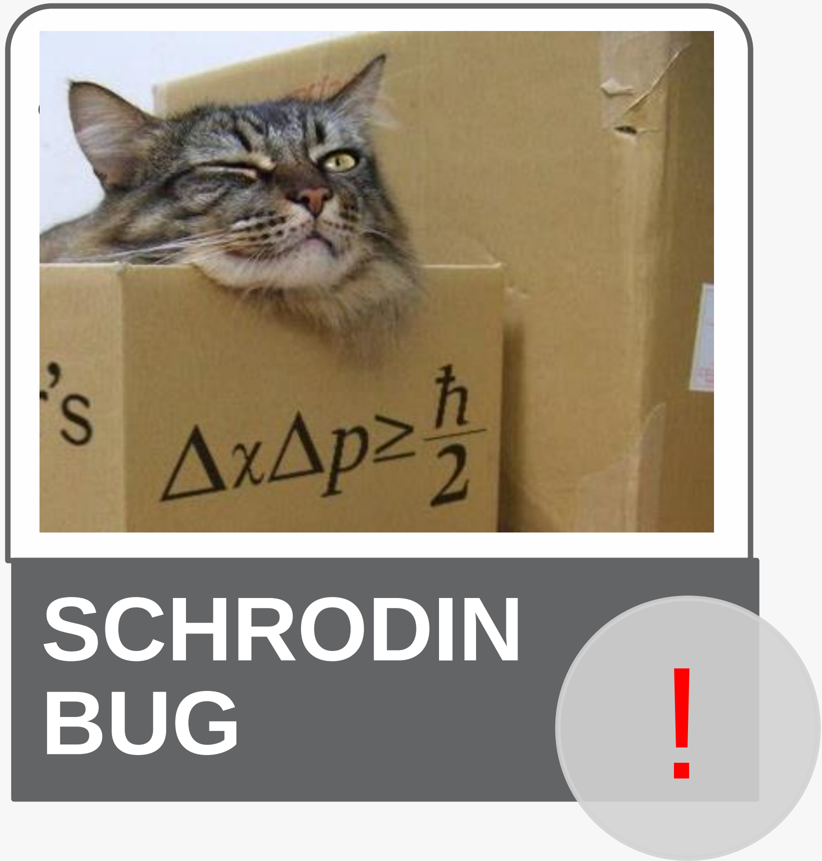
Invalid Memory Access



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

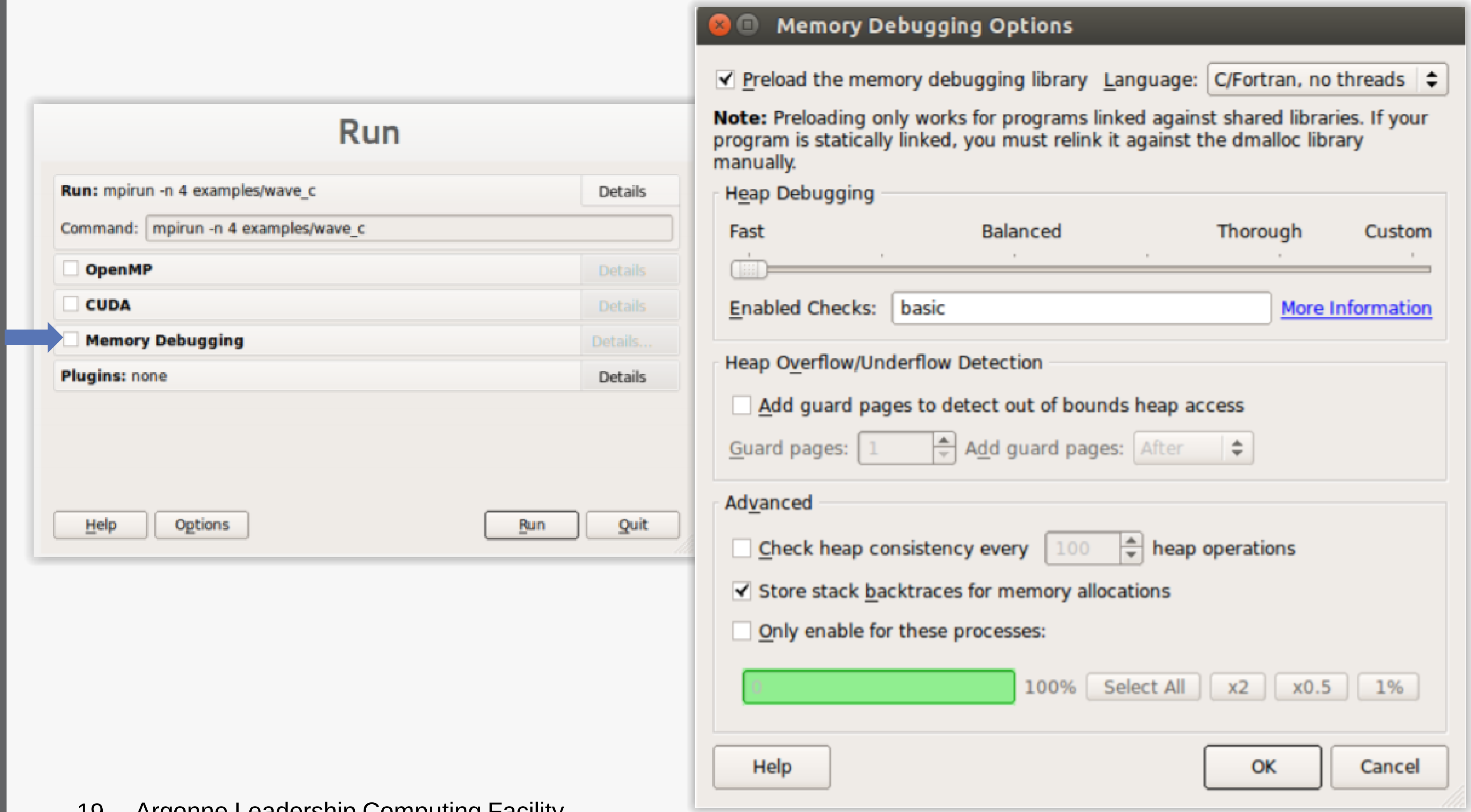
It works… Well, most of the time



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 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

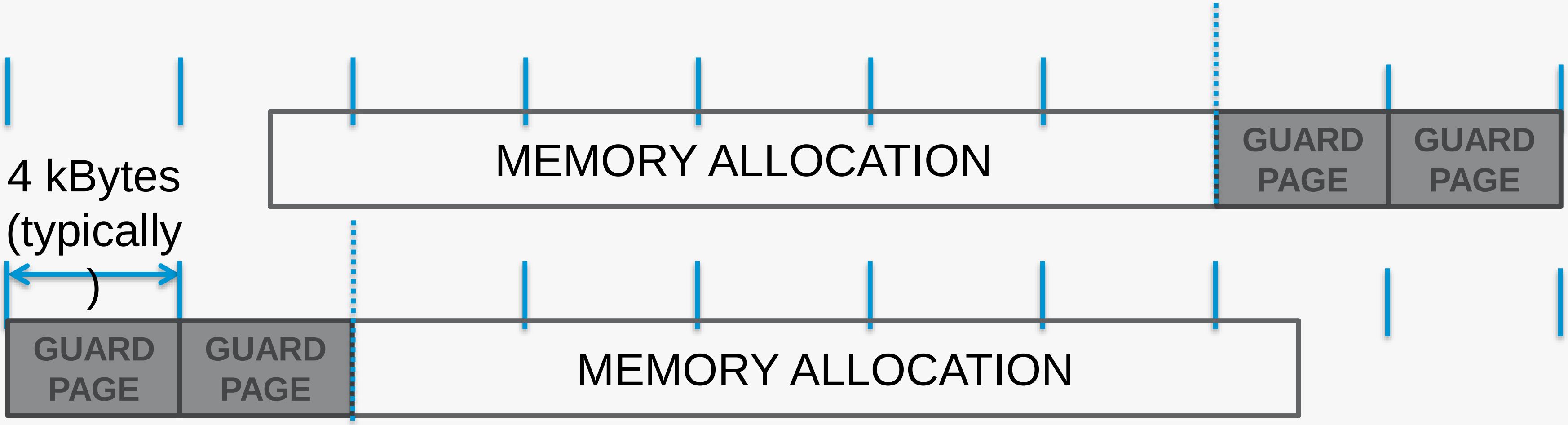
check-blank
- **Check to see if space that was blanked when a pointer was allocated/freed has been overwritten.**

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

See user-guide:

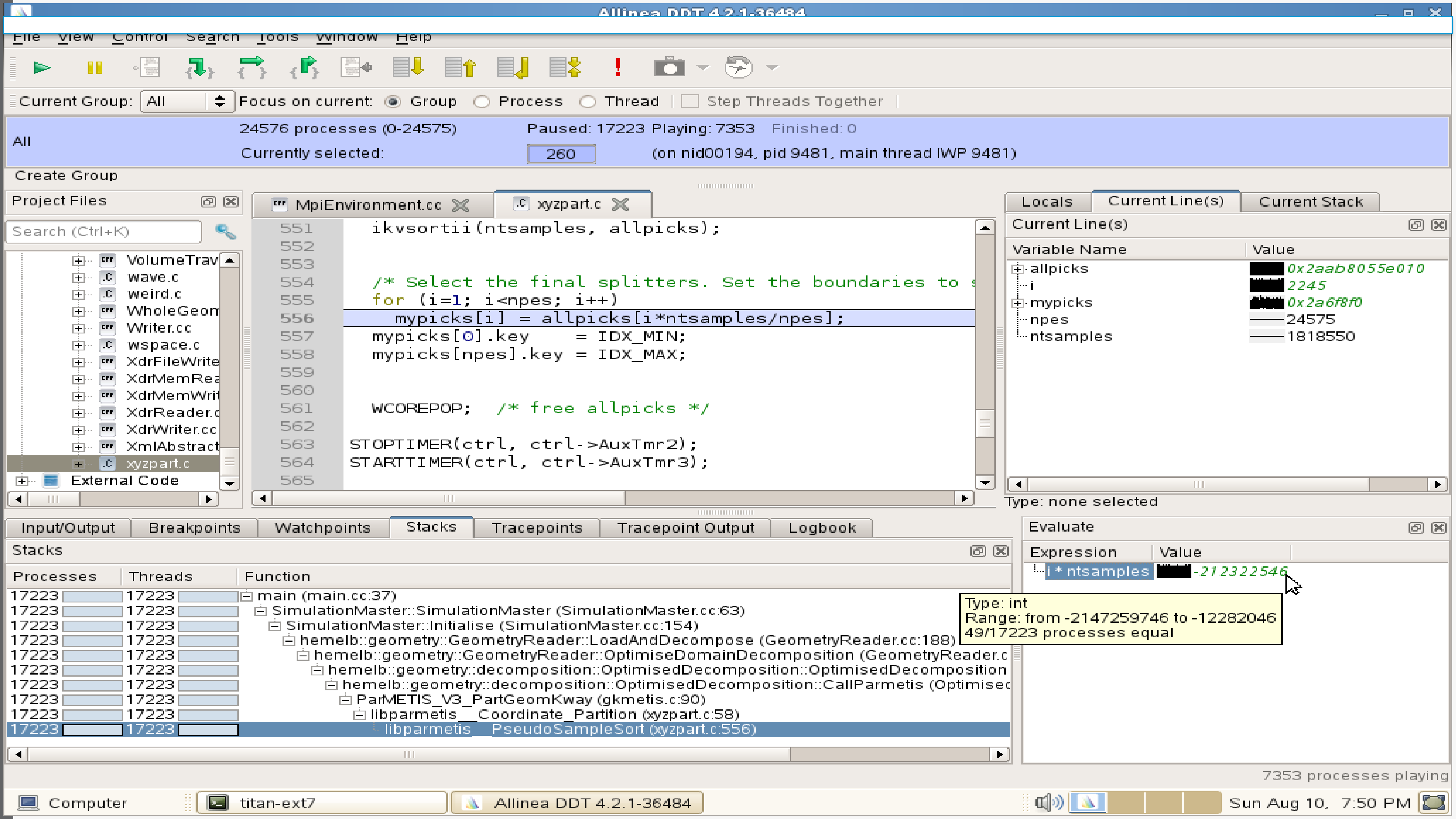
Chapter 12.3.2

Guard pages (aka “Electric Fences”)

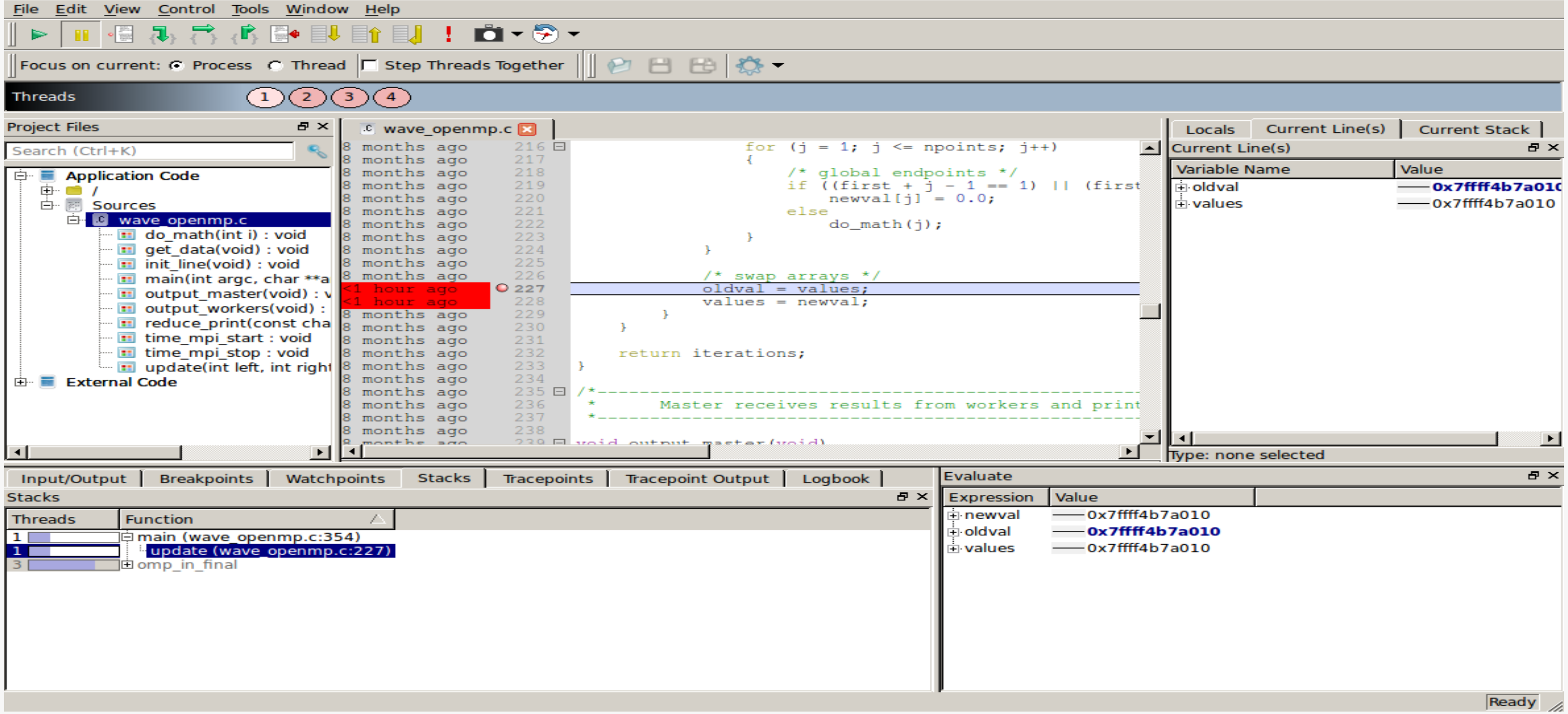


- **A powerful feature…:**
  - Forbids read/write on guard pages throughout the whole execution

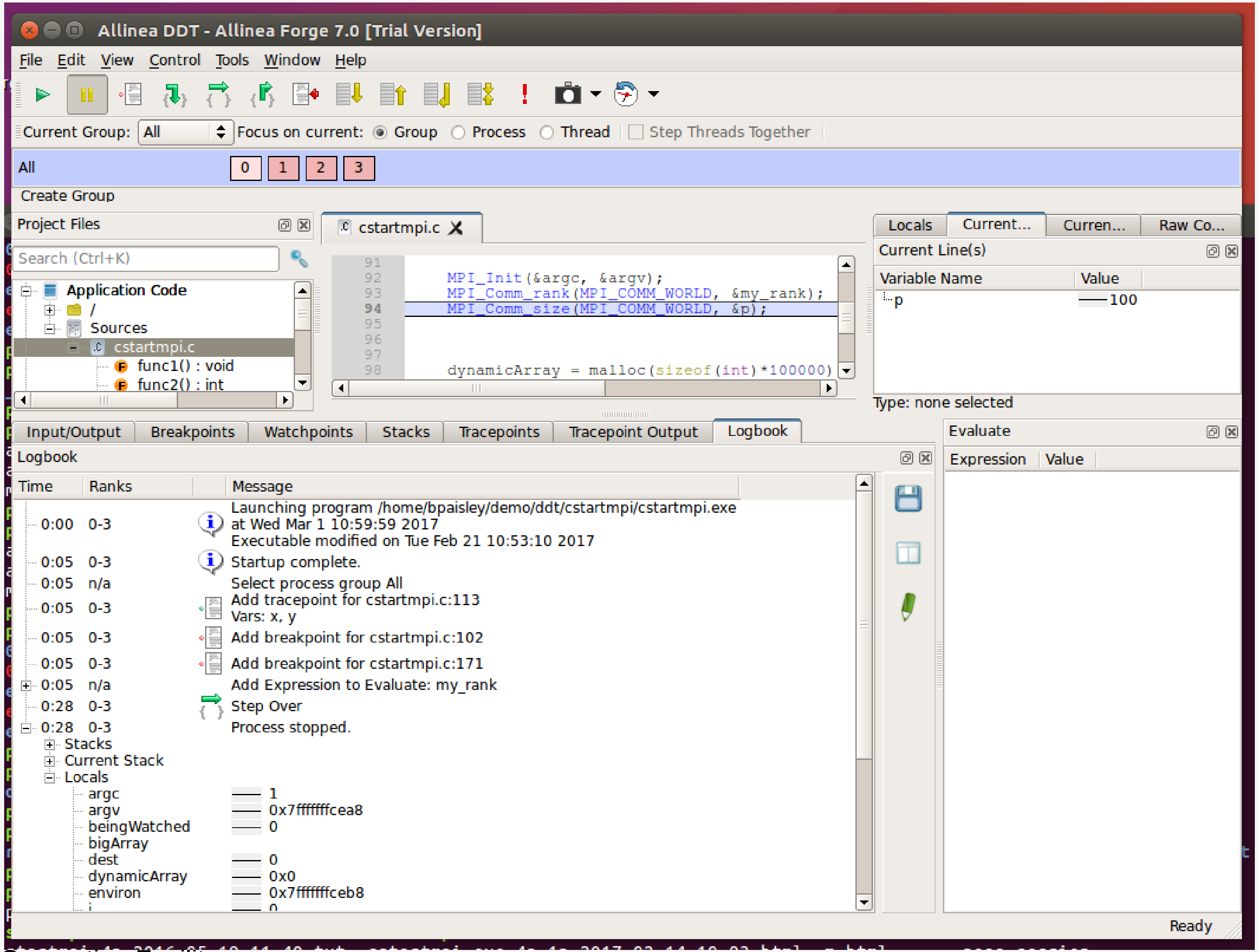
    (because it overrides C Standard Memory Management library)
- **… to be used carefully:**
  - Kernel limitation: up to 32k guard pages max (“mprotect fails” error)
  - Beware the additional memory usage cost


Allinea DDT 4.2.1-36484
File View Control Search Tools Window Help
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
MpiEnvironment.cc
xyzpart.c
551 ikvsortii(ntsamples, allpicks);
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;
561 WCOREPOP; /* free allpicks */
563 STOPTIMER(ctrl, ctrl->AuxTmr2);
564 STARTTIMER(ctrl, ctrl->AuxTmr3);
Locals
Current Line(s)
Current Stack
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
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 (Optimised
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
Computer
titan-ext7
Allinea DDT 4.2.1-36484
Sun Aug 10, 7:50 PM

New Bugs from Latest Changes

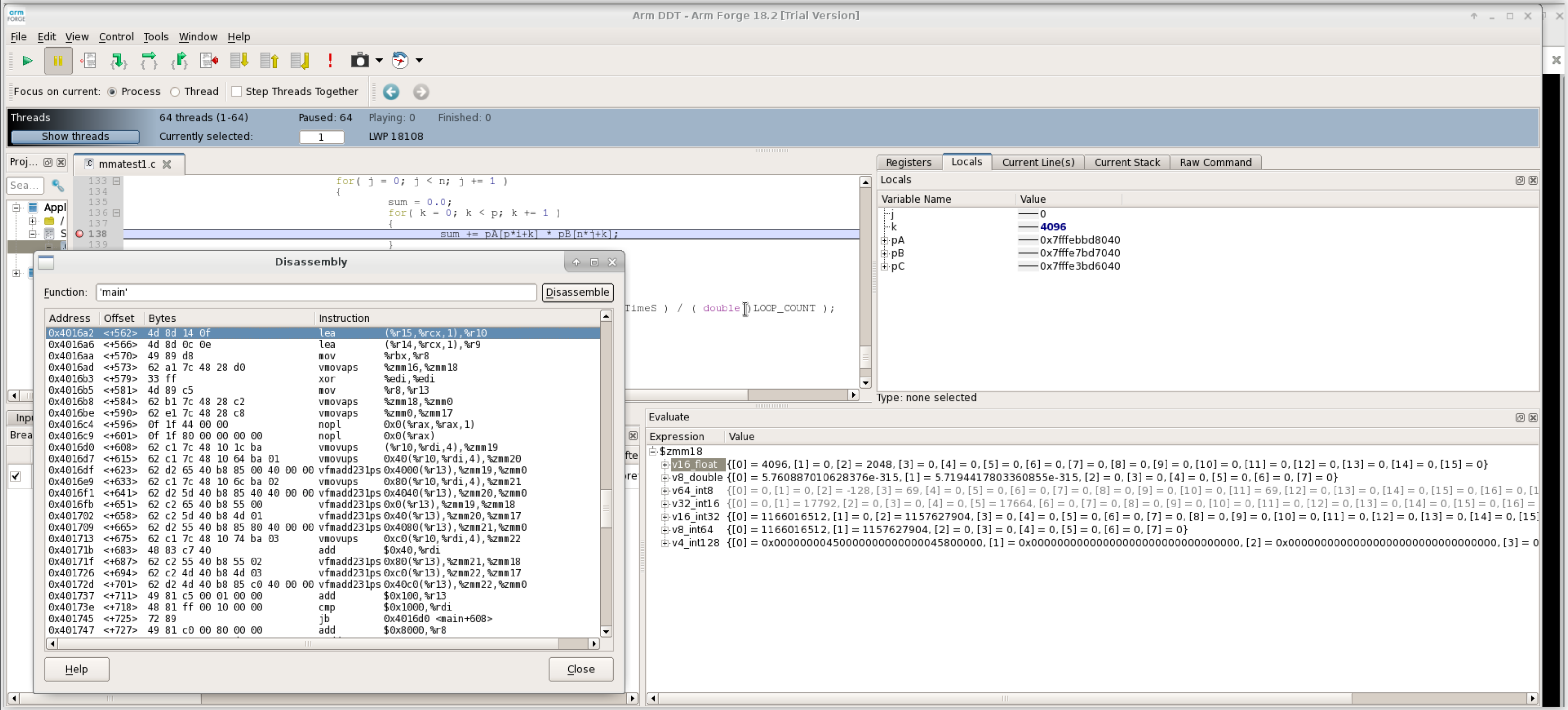


Track Your Changes in a Logbook



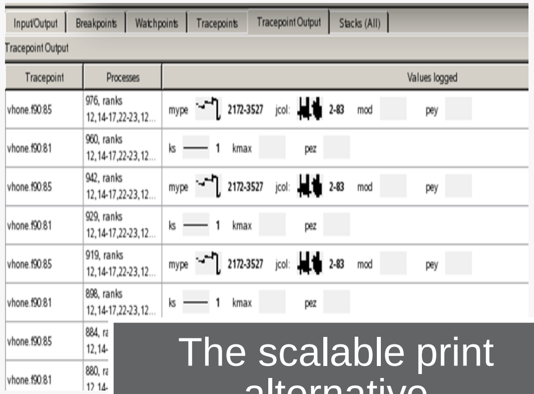
Argonne NATIONAL LABORATORY

Inspect AVX Registers

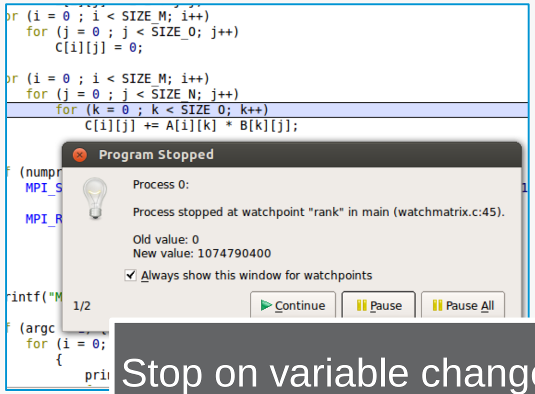


Arm DDT Demo

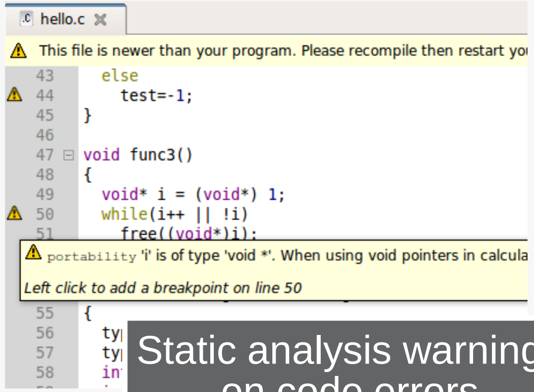
Five great things to try with Allinea DDT



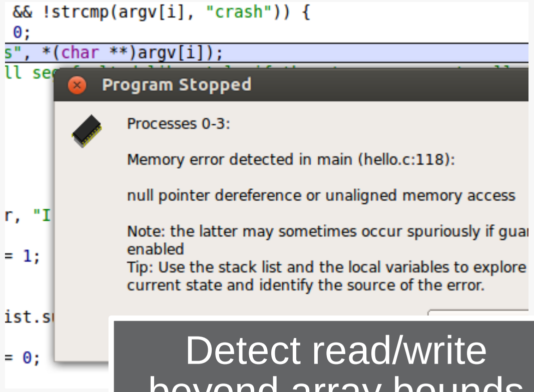
The scalable print alternative



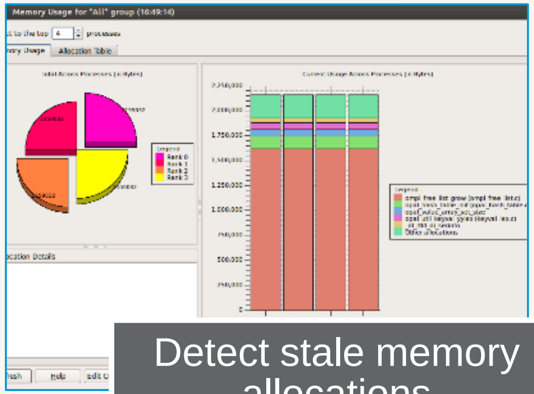
Stop on variable change



Static analysis warnings on code errors



Detect read/write beyond array bounds



Detect stale memory allocations

Argonne NATIONAL LABORATORY

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

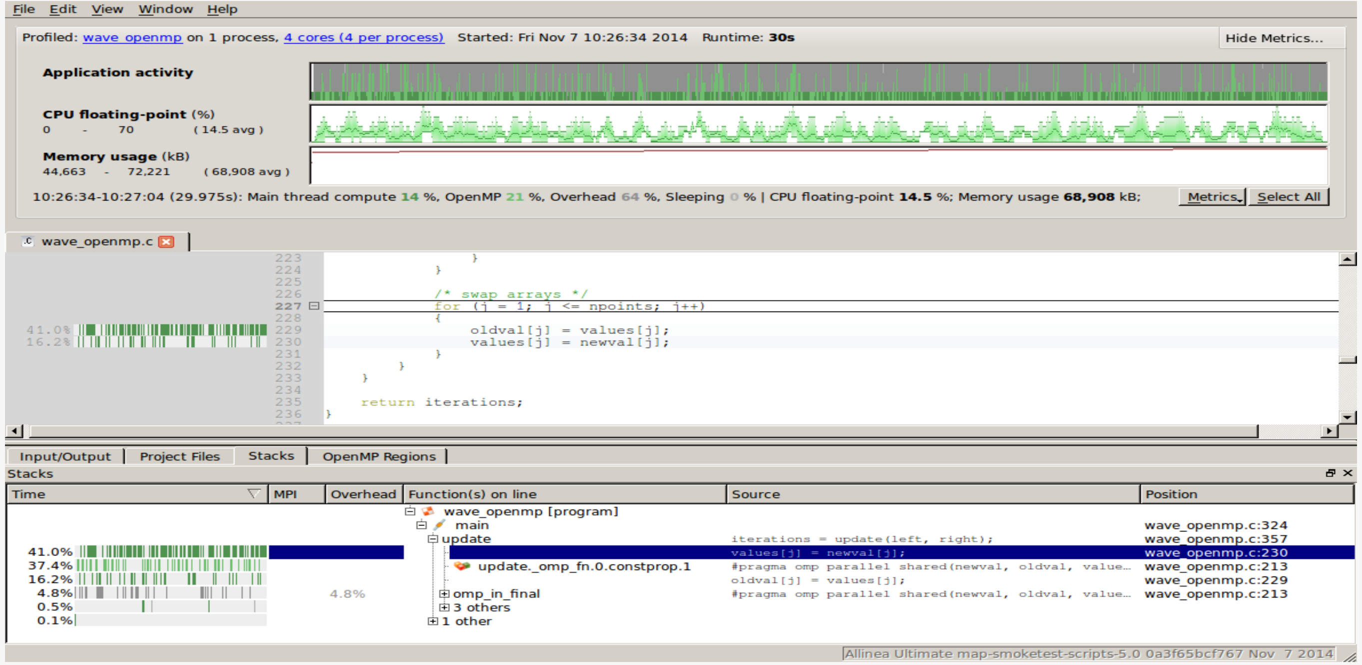
Or use the reverse connect mechanism

- 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

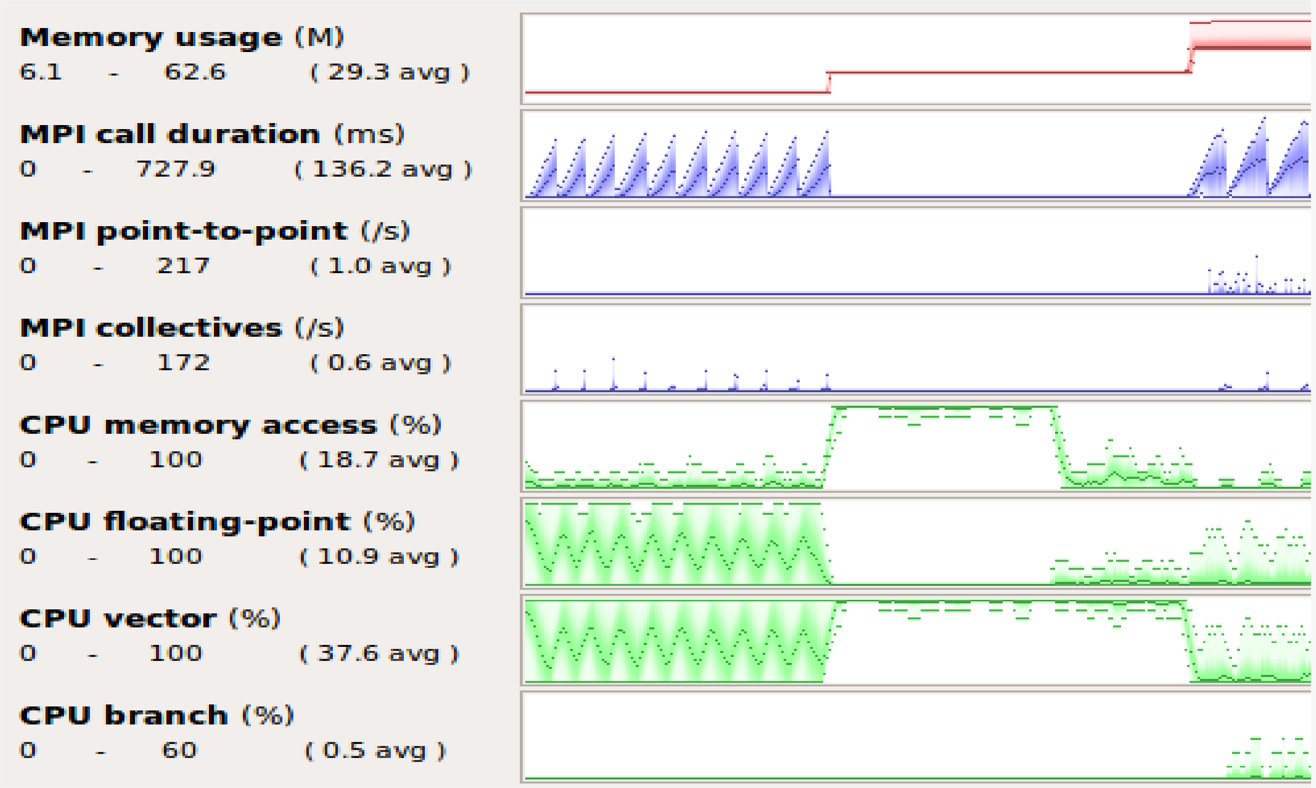
Arm MAP – The Profiler

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



Argonne NATIONAL LABORATORY

Glean Deep Insight from our Source-Level Profiler



Track memory usage across the entire application over time

Spot MPI and OpenMP imbalance and overhead

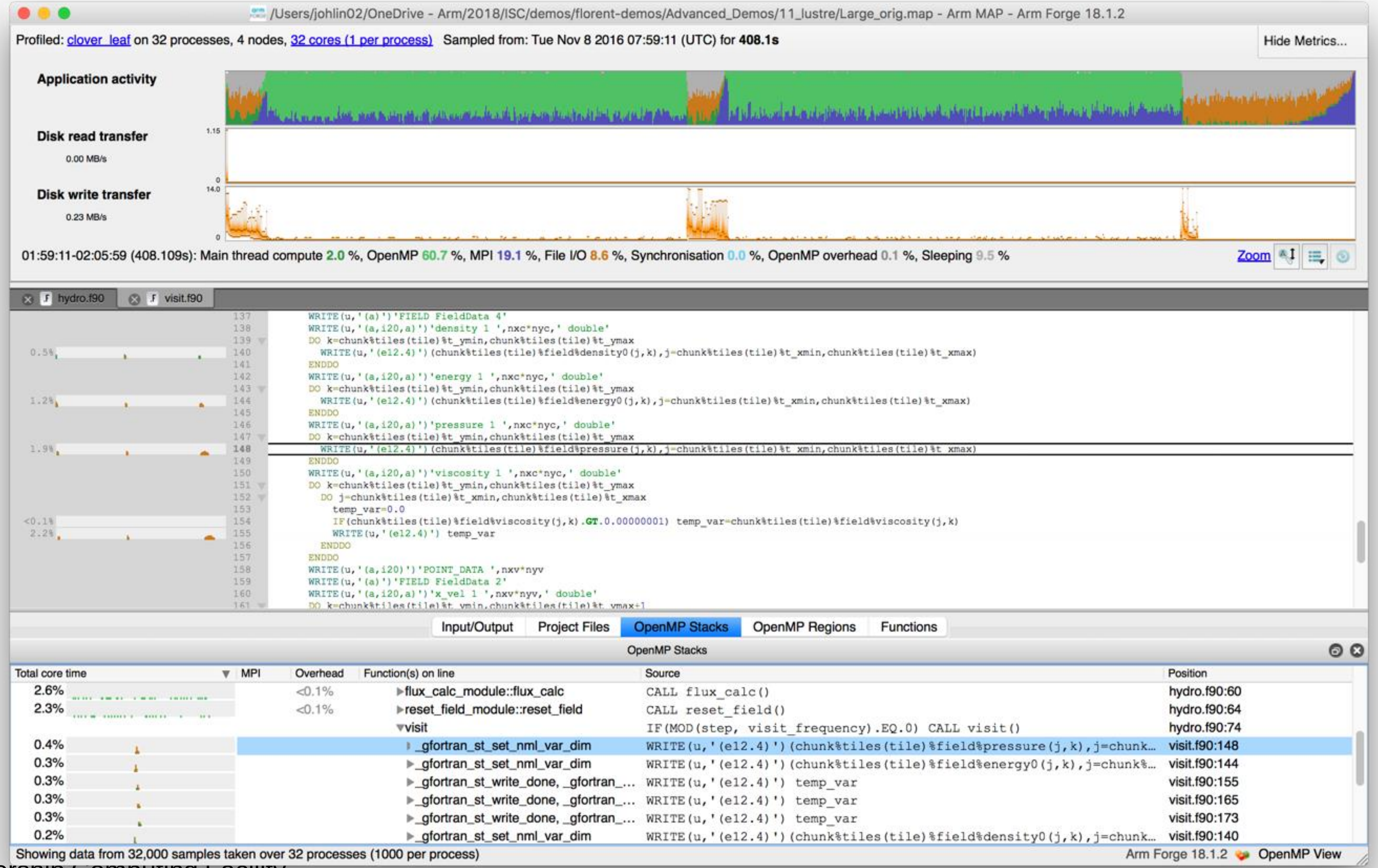
Optimize CPU memory and vectorization in loops

Detect and diagnose I/O bottlenecks at real scale

Argonne NATIONAL LABORATORY

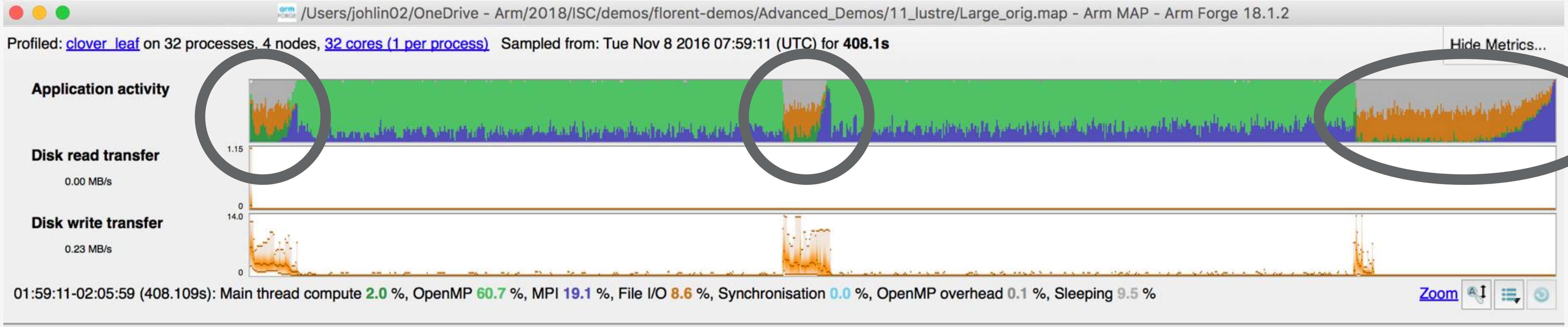
Initial profile of CloverLeaf shows surprisingly unequal I/O

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



Symptoms and causes of the I/O issues

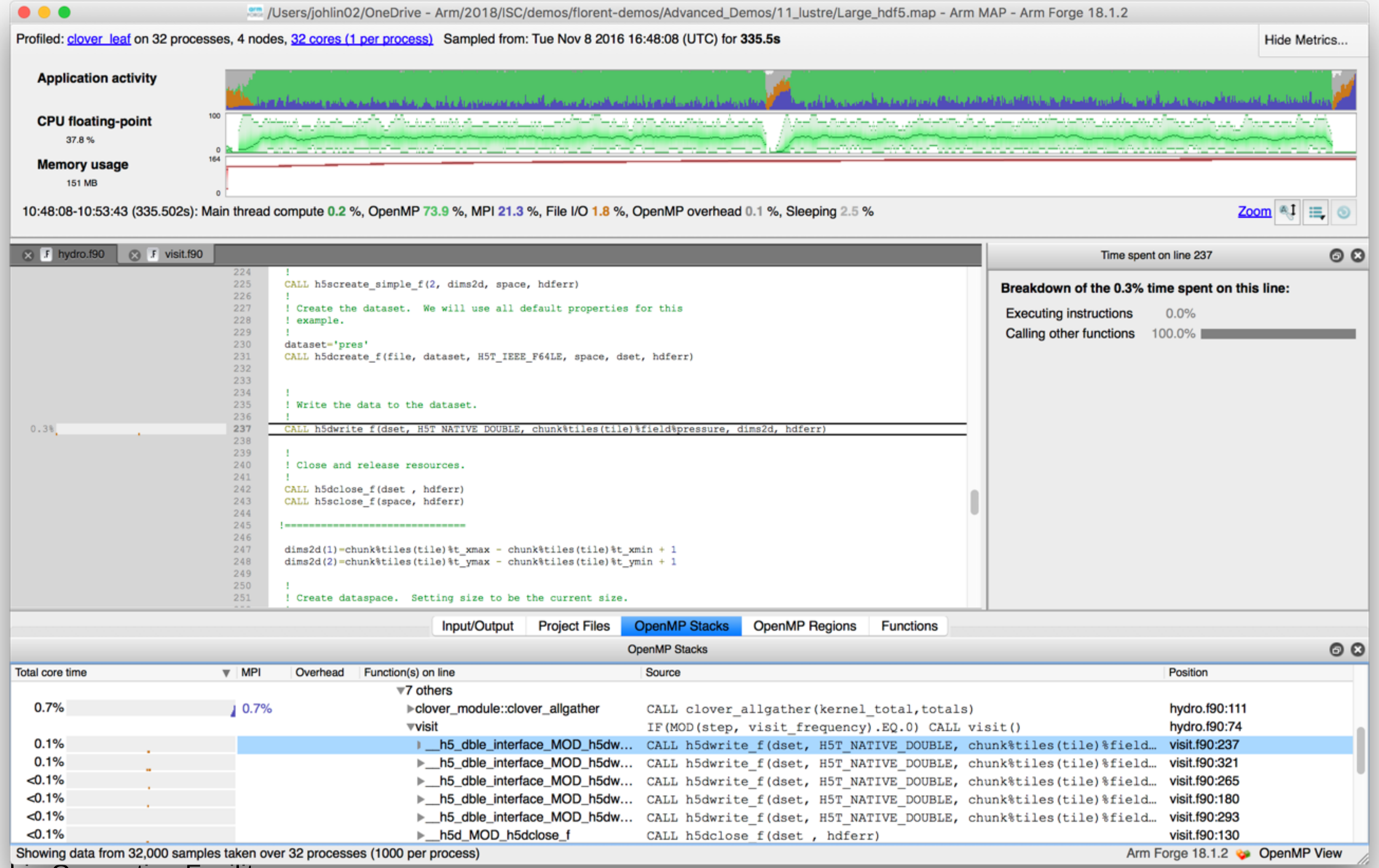
Sub-optimal file format and surprise buffering.



- 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.

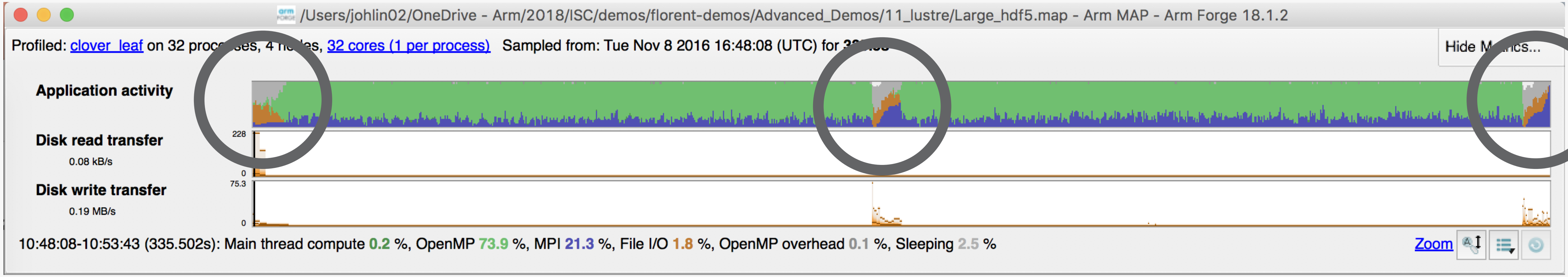
Solution: use HDF5 to write binary files

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.

Argonne NATIONAL LABORATORY

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

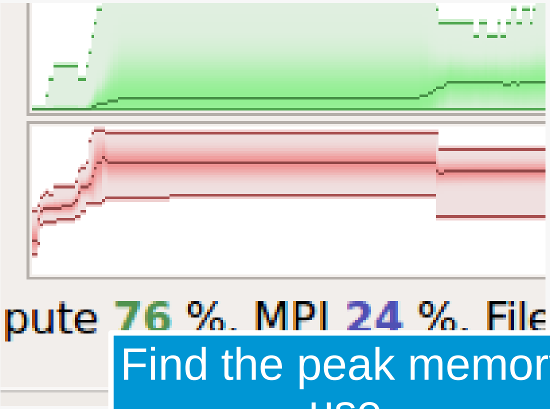
Follow the instructions displayed to prepare the code

- $ cc -O3 **-g** myapp.c -o myapp.exe **-Wl,@/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

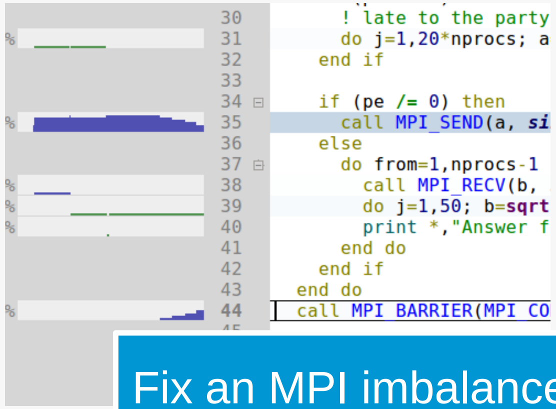
Open the results

- 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)

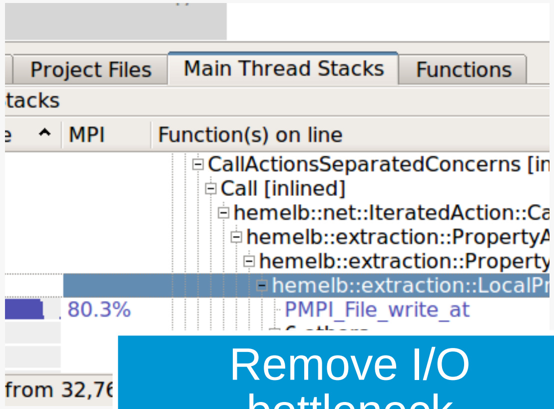
Six Great Things to Try with Allinea MAP



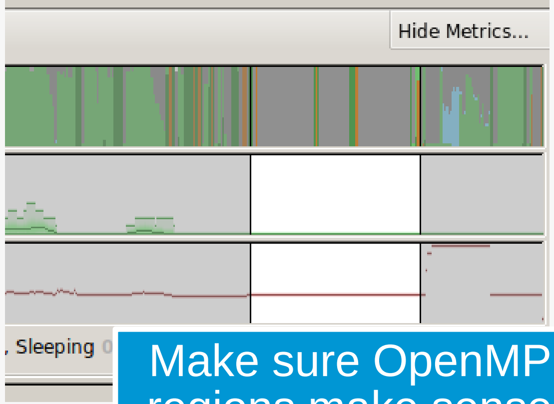
Find the peak memory use



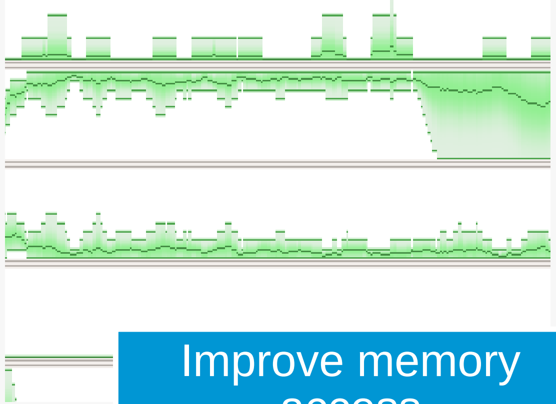
Fix an MPI imbalance



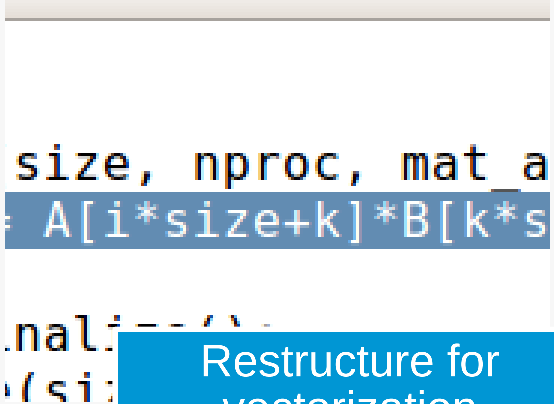
Remove I/O bottleneck



Make sure OpenMP regions make sense



Improve memory access



Restructure for vectorization

Theta Specific Settings

arm

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