arm

Debugging and Profiling HPC Applications

ATPESC

August 7, 2018

Ryan Hulguin
ryan.hulguin@arm.com
HPC Applications Engineer

Agenda

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

Debugging

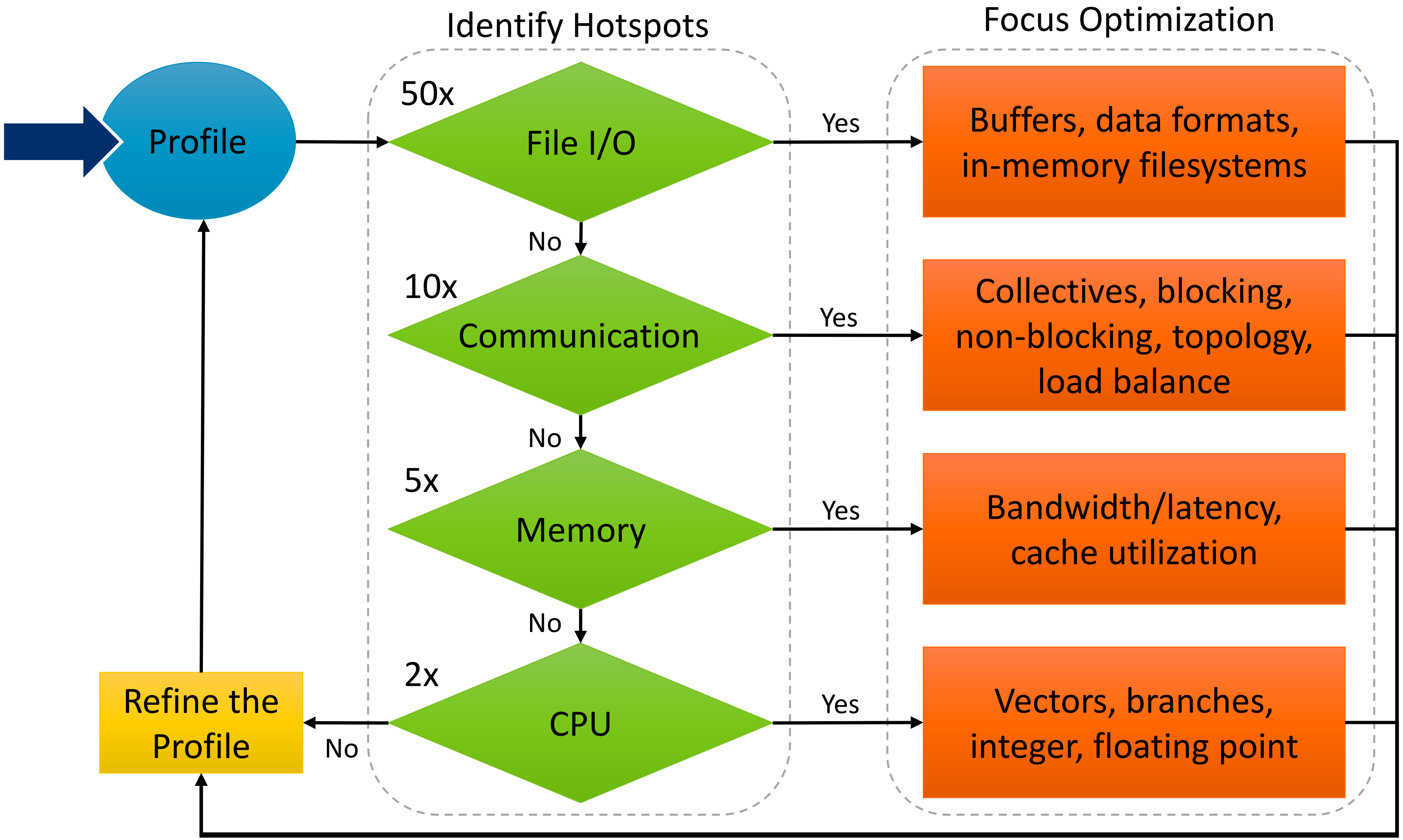
Transforming a broken program to a working one

How? TRAFFIC!

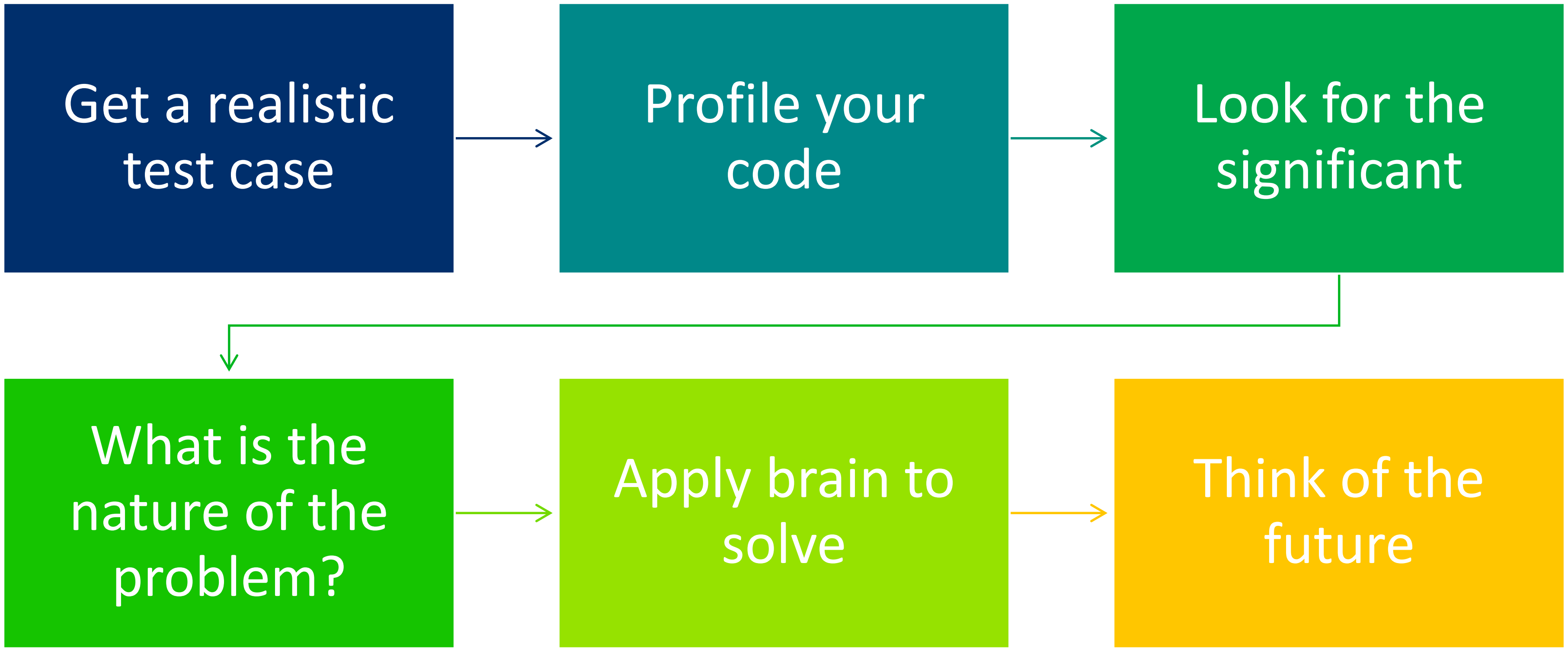
- **T**rack the problem
- **R**eproduce
- **A**utomate - (and simplify) the test case
- **F**ind origins – where could the “infection” be from?
- **F**ocus – examine the origins
- **I**solate – narrow down the origins
- **C**orrect – fix and verify the test case is successful

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

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

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)

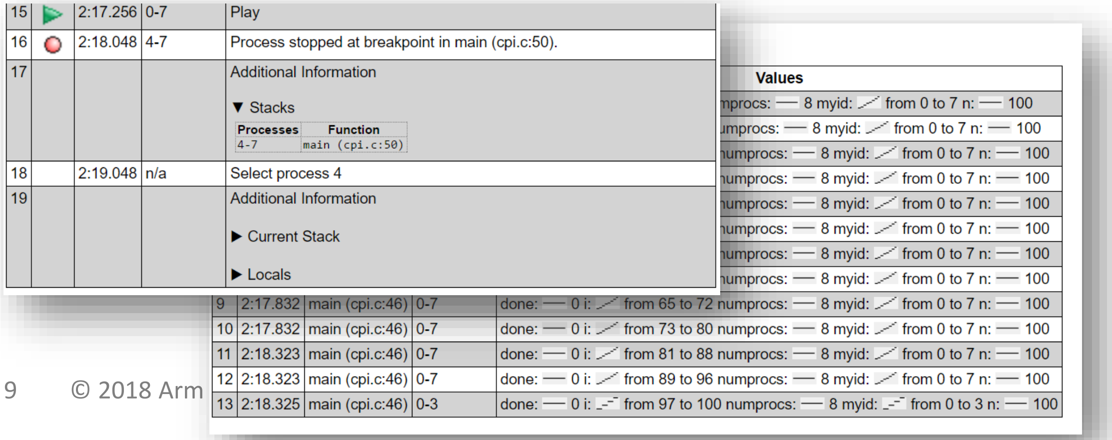
Run and ensure application correctness

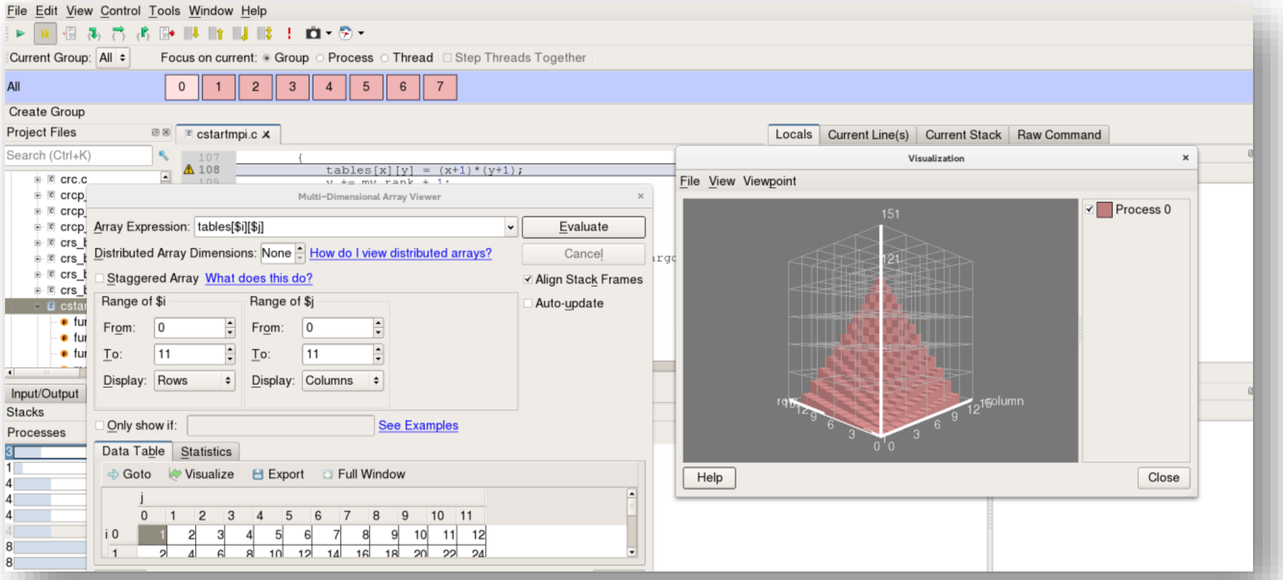
Combination of debugging and re-compilation

- Ensure application correctness with **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
```





Understand application behaviour

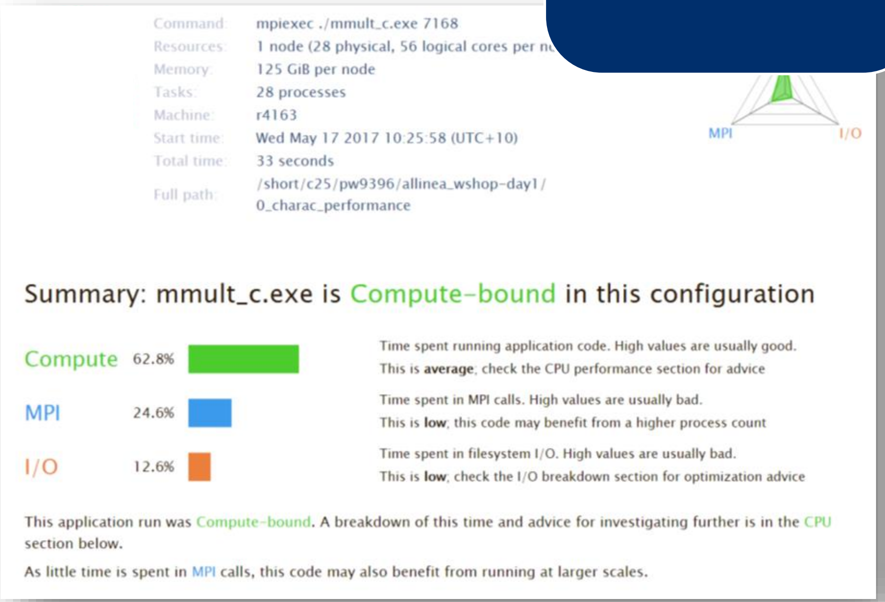
Set a reference for future work

Is it performant?

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

<u>Example:</u>

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



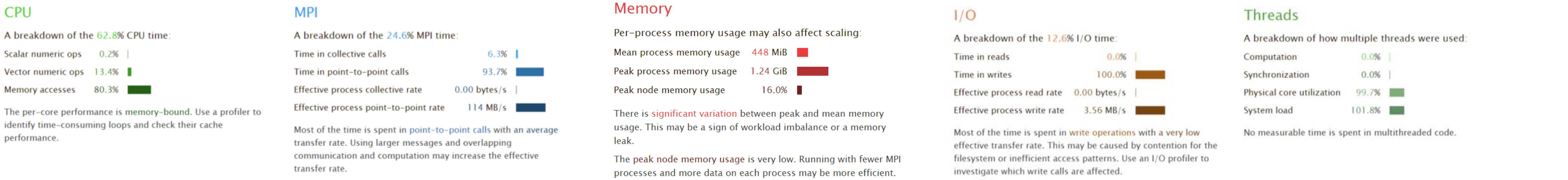
Command:	mpiexec ./mmult_c.exe 7168
Resources:	1 node (28 physical, 56 logical cores per n
Memory:	125 GiB per node
Tasks:	28 processes
Machine:	r4163
Start time:	Wed May 17 2017 10:25:58 (UTC+10)
Total time:	33 seconds
Full path:	/short/c25/pw9396/allinea_wshop-day1/0_charac_performance

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

Compute	62.8%	Time spent running application code. High values are usually good. This is **average**; check the CPU performance section for advice
MPI	24.6%	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%	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.

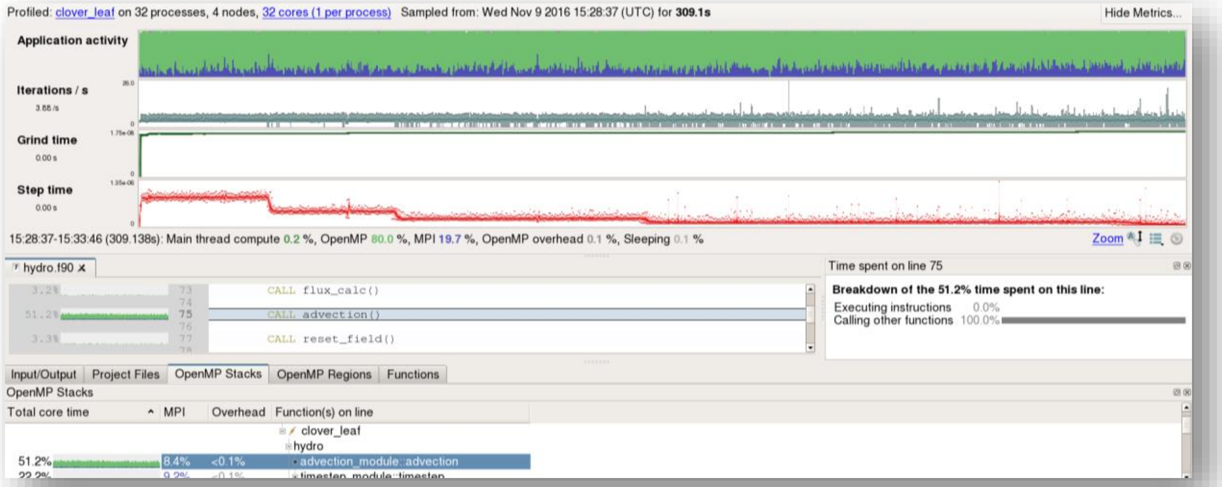
Optimize the application for Arm

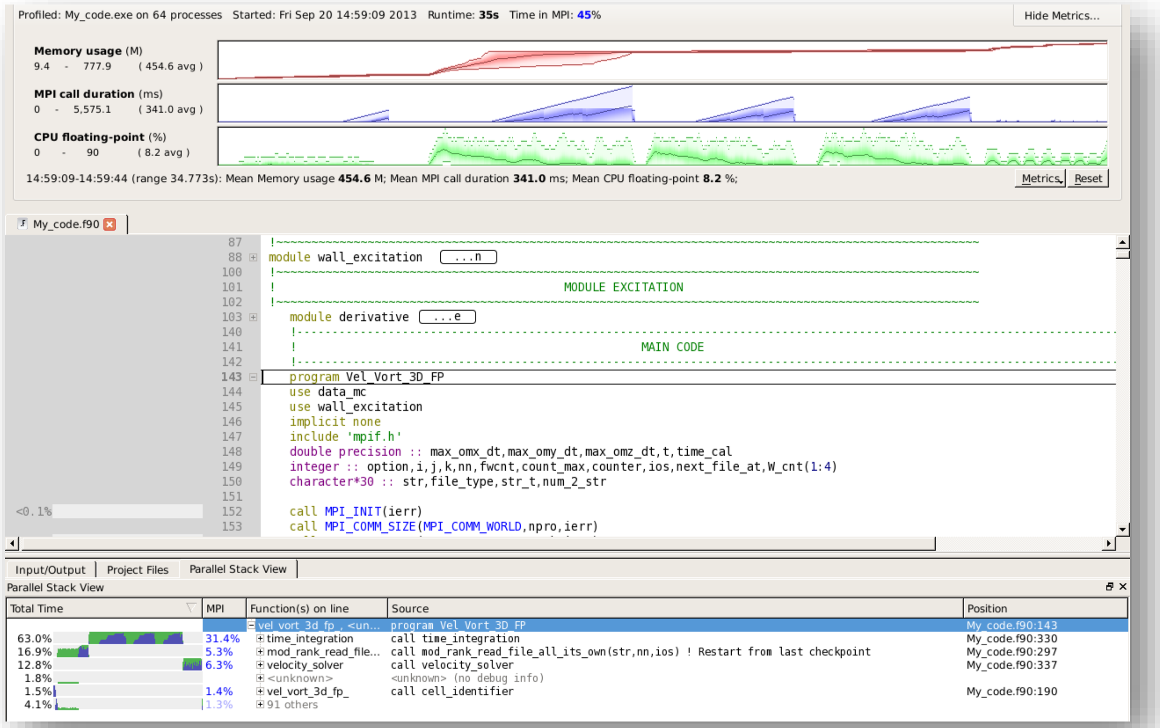
if not, use the Arm MAP profiler for optimization

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

Examples:

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





arm

Debugging with DDT

arm

Arm DDT – The Debugger

Who had a rogue behaviour ?

- Merges stacks from processes and threads

Where did it happen?

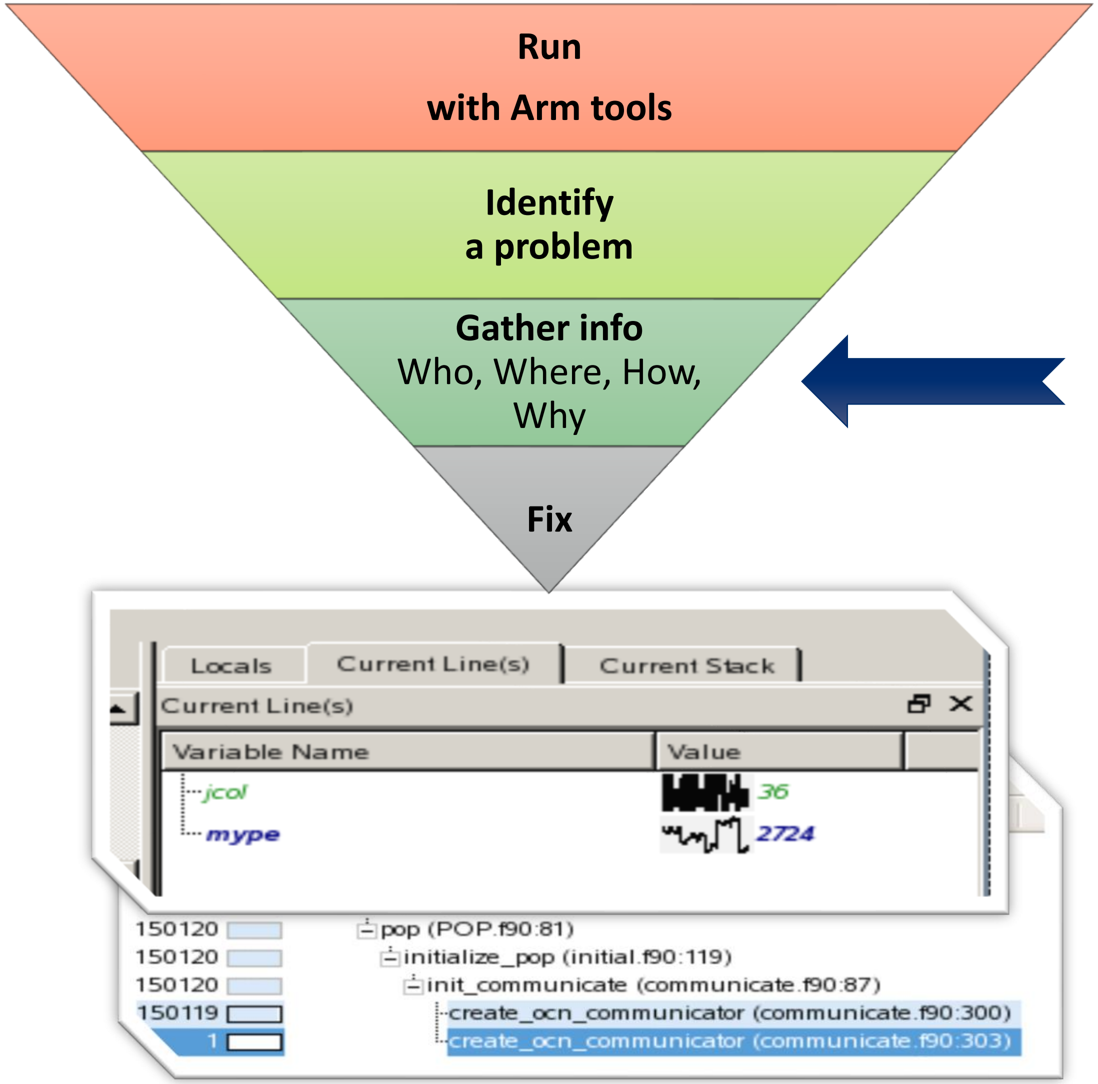
- leaps to source

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

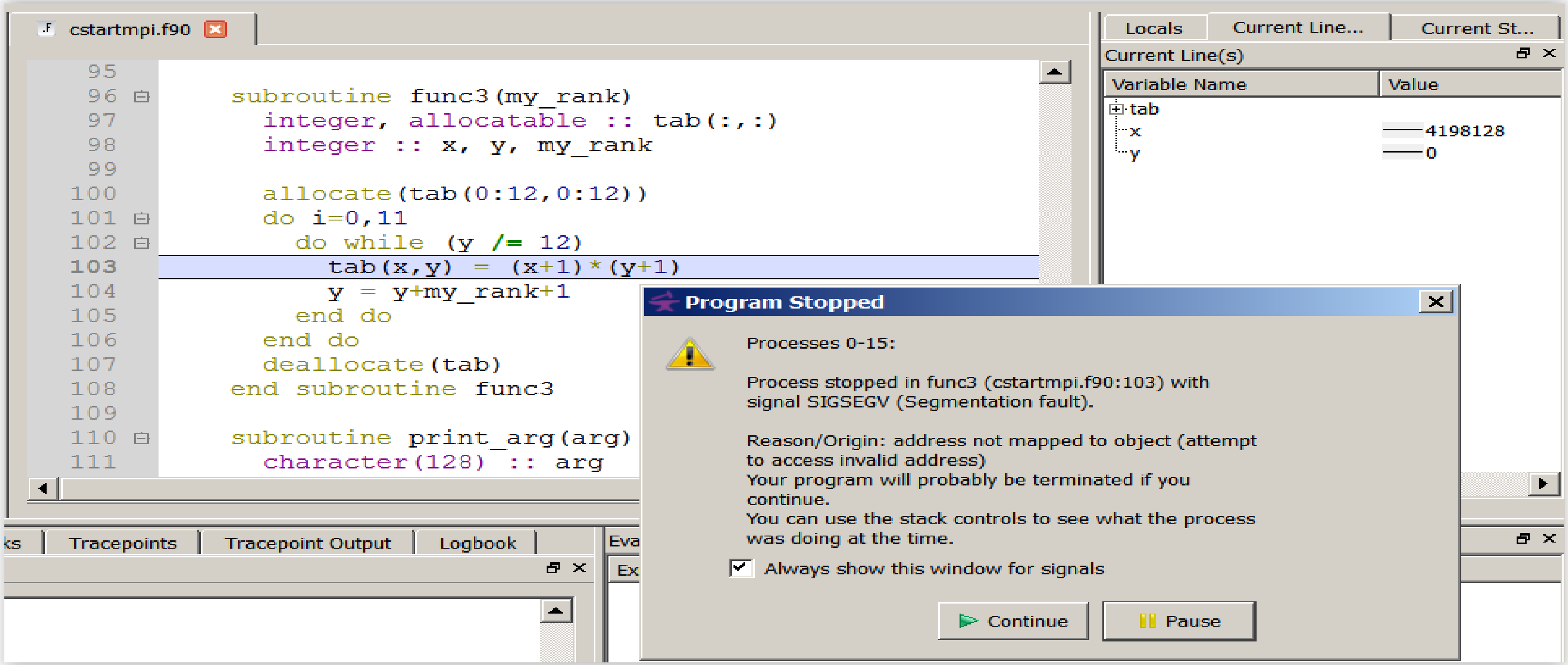
Segmentation Fault

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/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]$
```

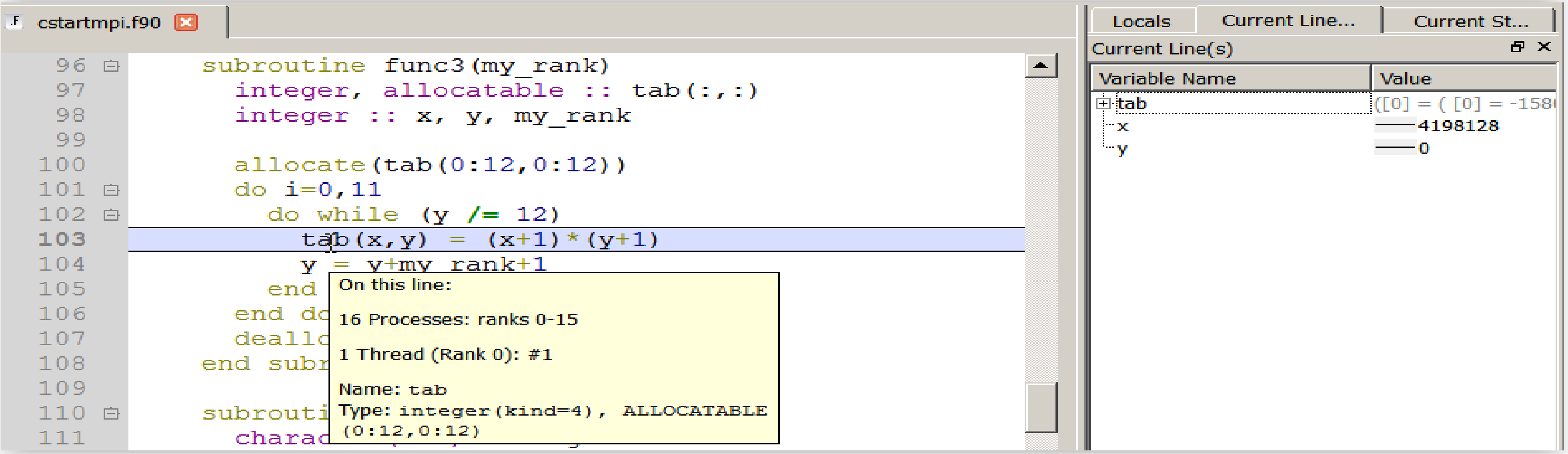
What happens when it runs under DDT?

Segmentation Fault in DDT



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

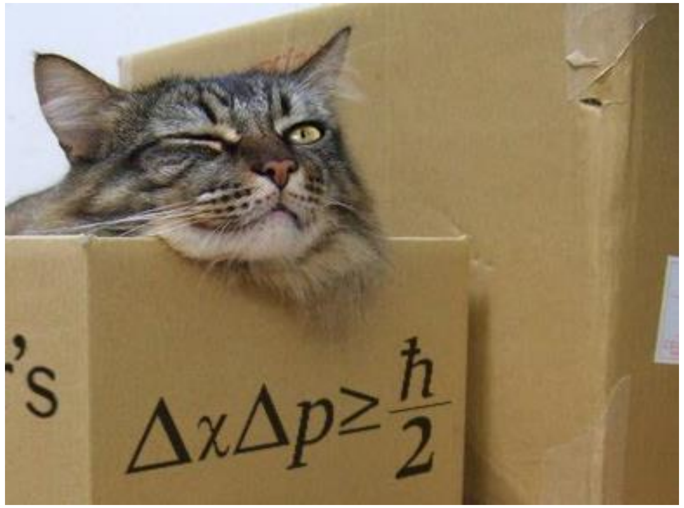
Invalid Memory Access



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



SCHRODIN BUG

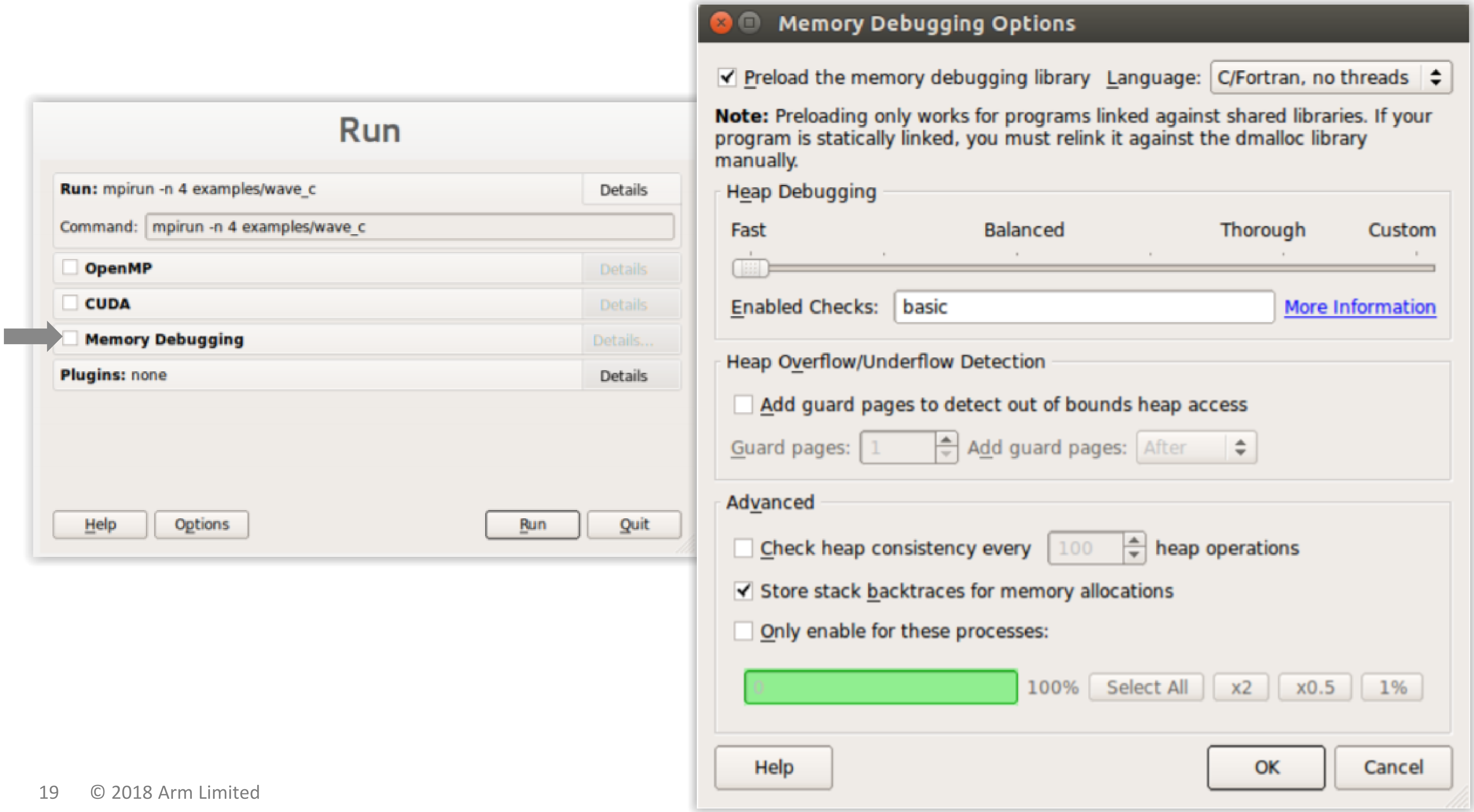
!

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

arm

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.

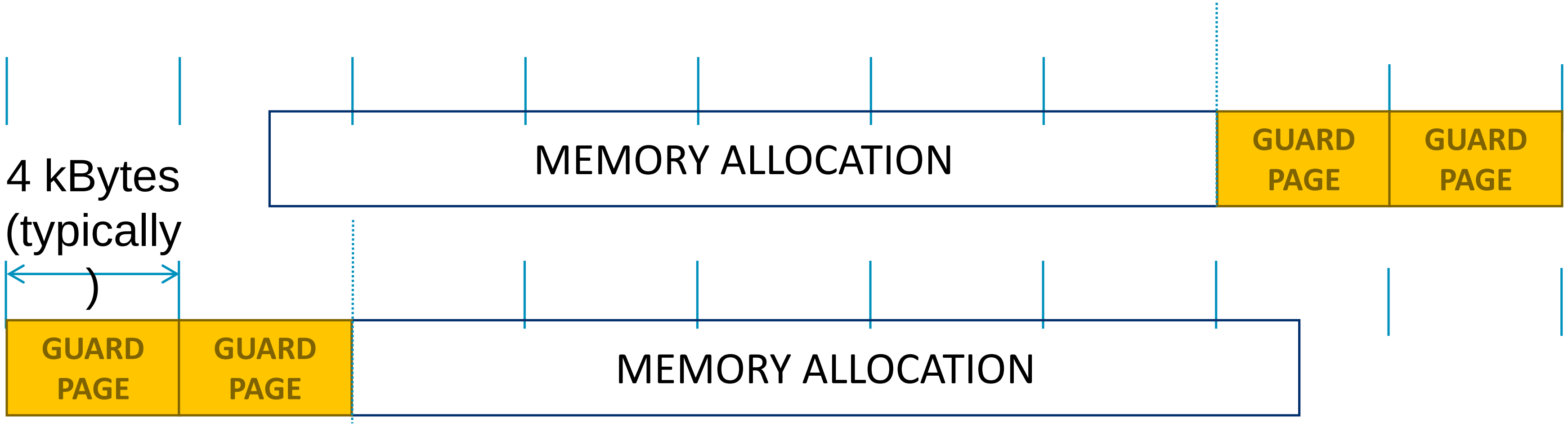
check-funcs

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

See user-guide:

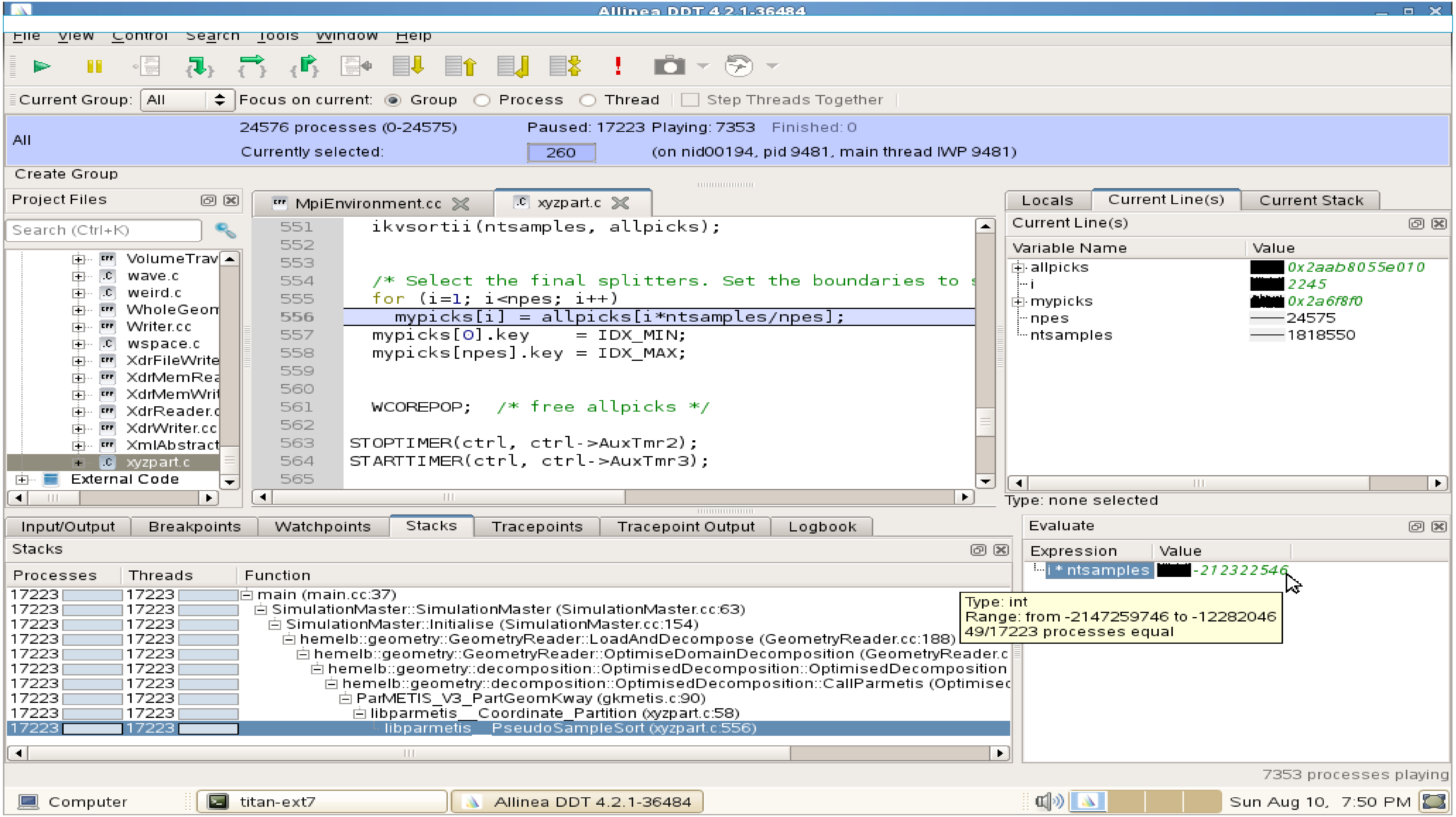
Chapter 12.3.2

Guard pages (aka “Electric Fences”)

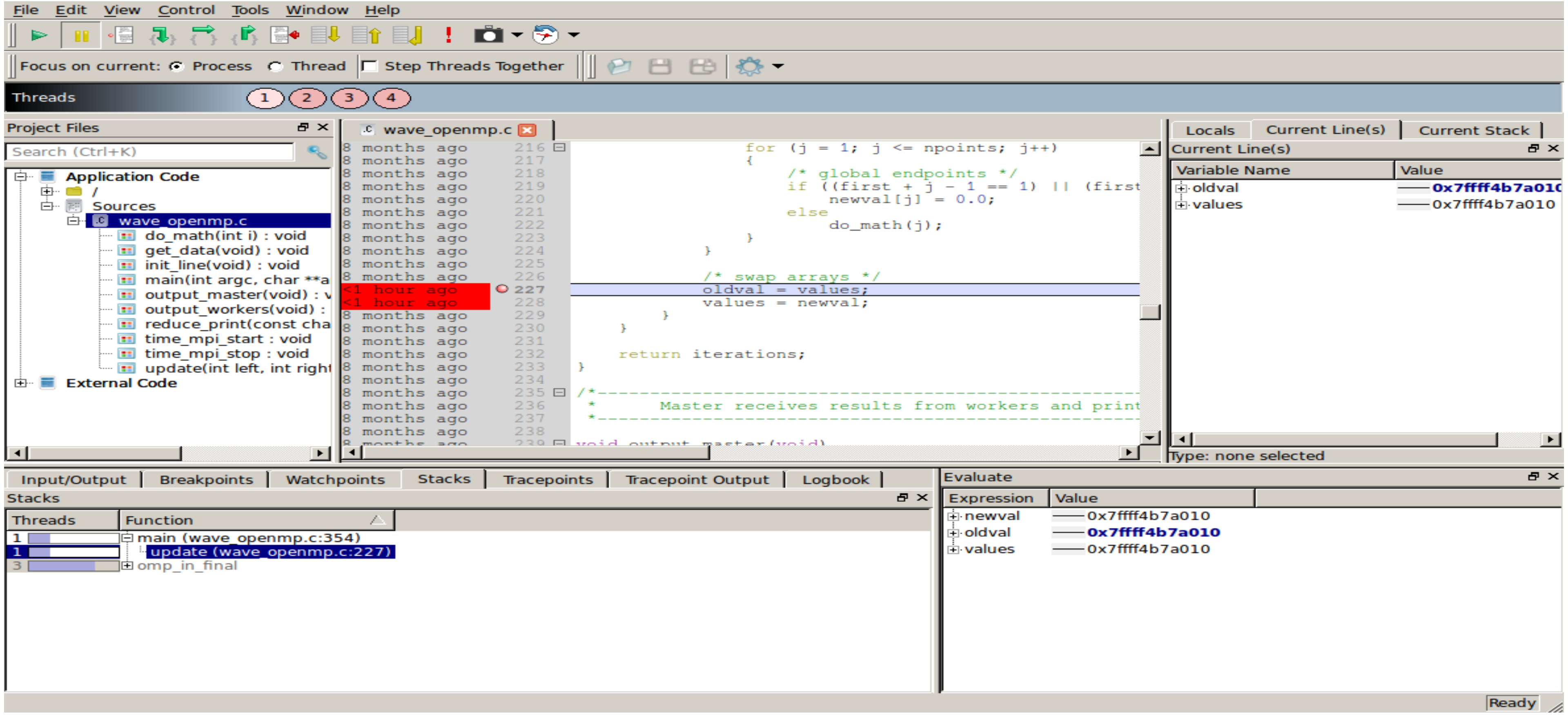


- **A powerful feature…:**
  - Forbids read/write on guard pages throughout the whole execution

    (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

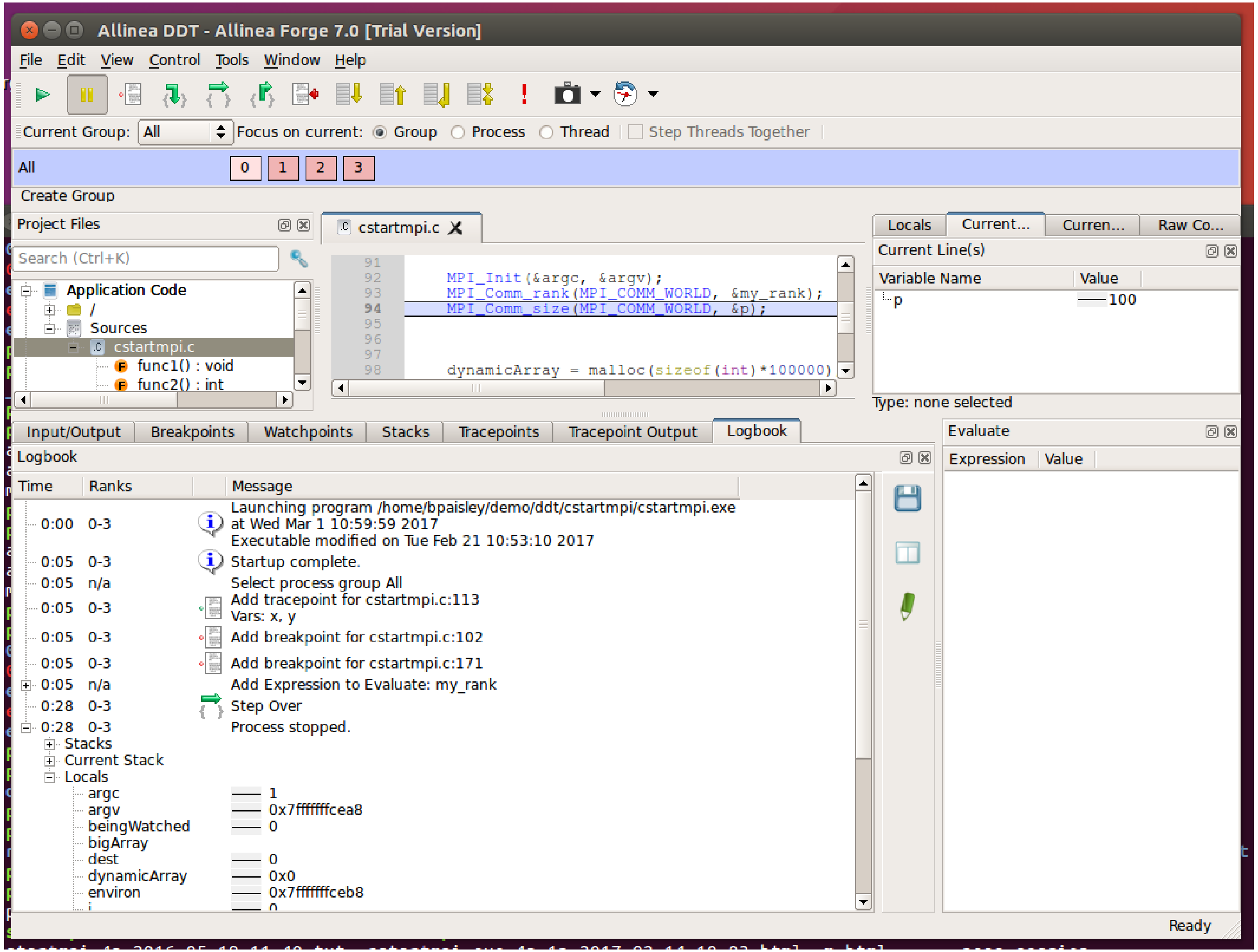

Allinea DDT 4.2.1-36484
File View Control Search Tools Window Help
Current Group: All
Focus on current: Group Process Thread
Step Threads Together
All
24576 processes (0-24575)
Paused: 17223 Playing: 7353 Finished: 0
Currently selected: 260 (on nid00194, pid 9481, main thread lWP 9481)
Create Group
Project 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);
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 (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

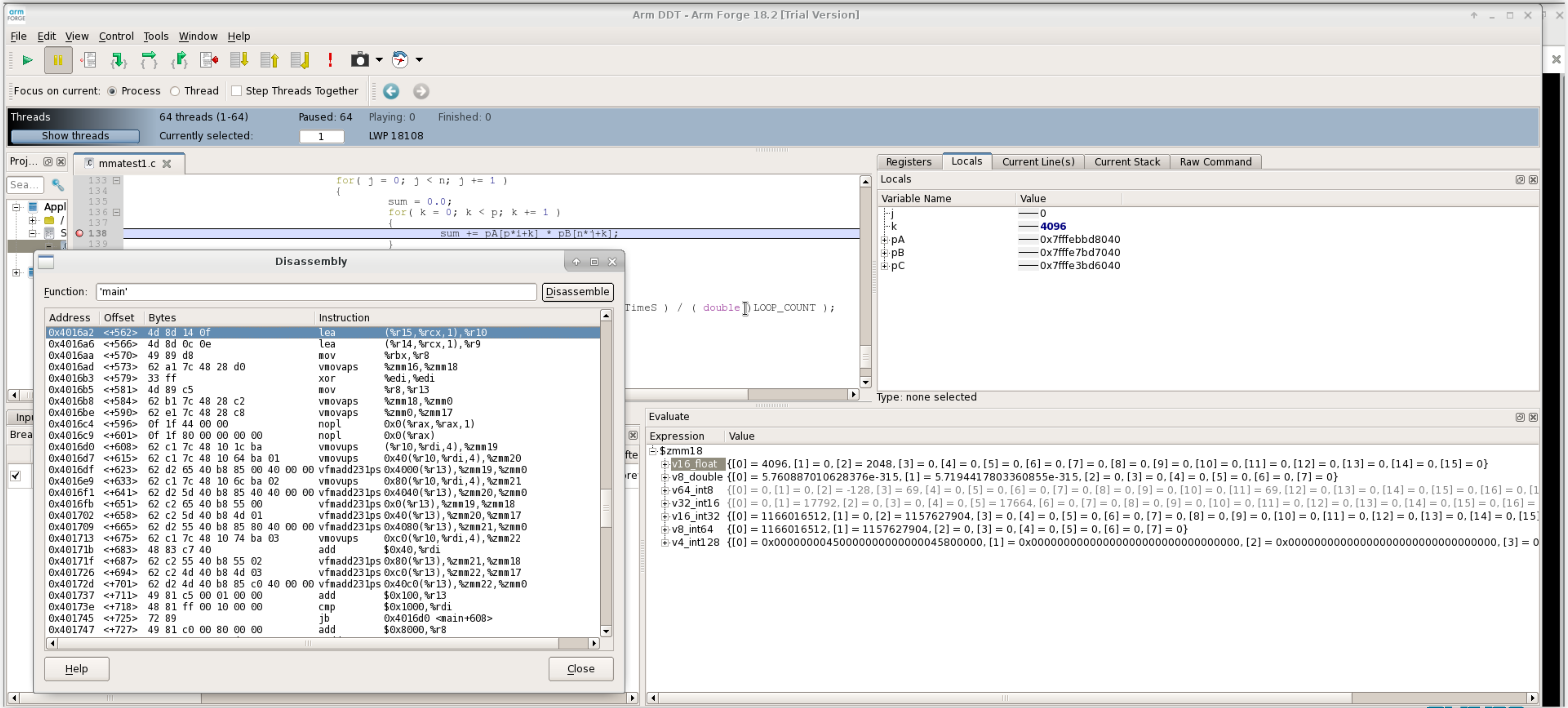


arm

Track Your Changes in a Logbook



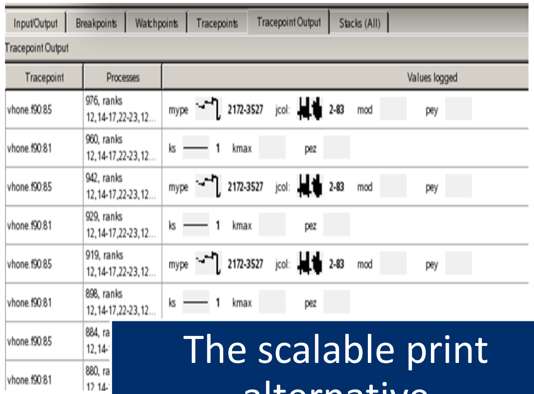
Inspect AVX Registers



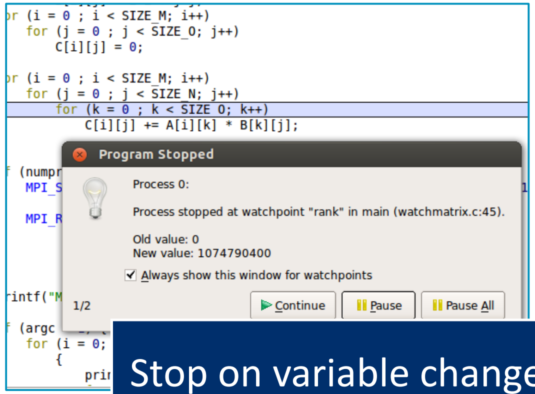
Arm DDT Demo

arm

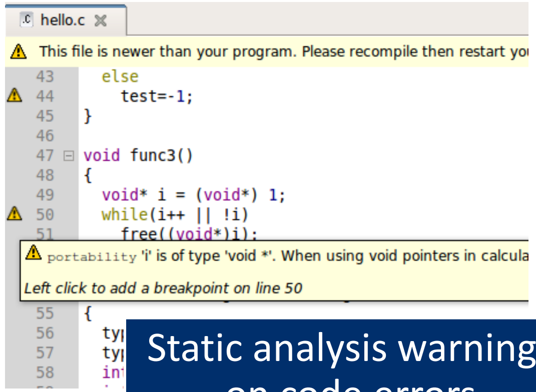
Five great things to try with Allinea DDT



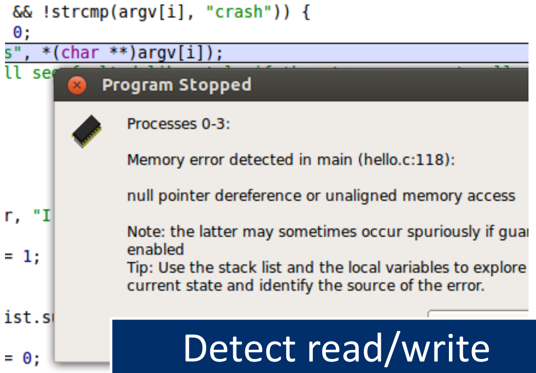
The scalable print alternative



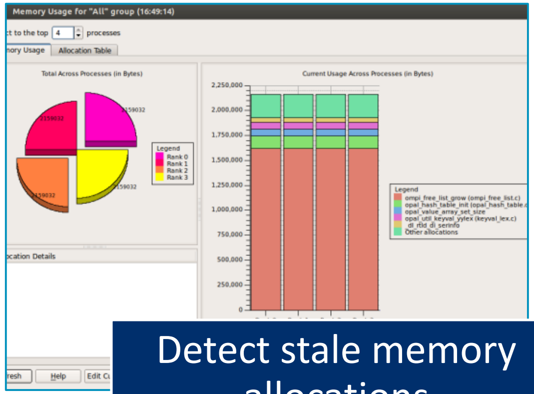
Stop on variable change



Static analysis warnings on code errors



Detect read/write beyond array bounds



Detect stale memory allocations

arm

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

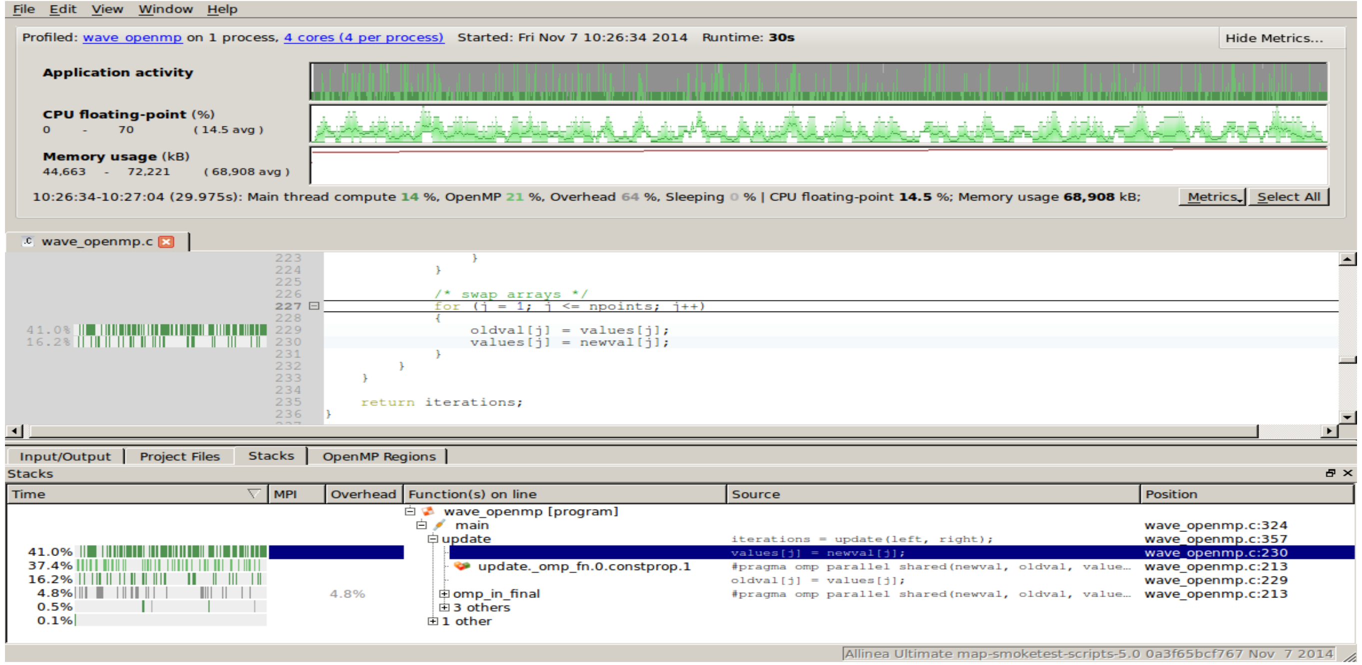
- On the login node:
  - $ ddt &
- (or use the remote client) **<- Preferred method**
- Then, edit the job script to run the following command and submit:
  - **ddt --connect** aprun -n 8 ./myapp.exe arg1 arg2

Profiling with MAP

arm

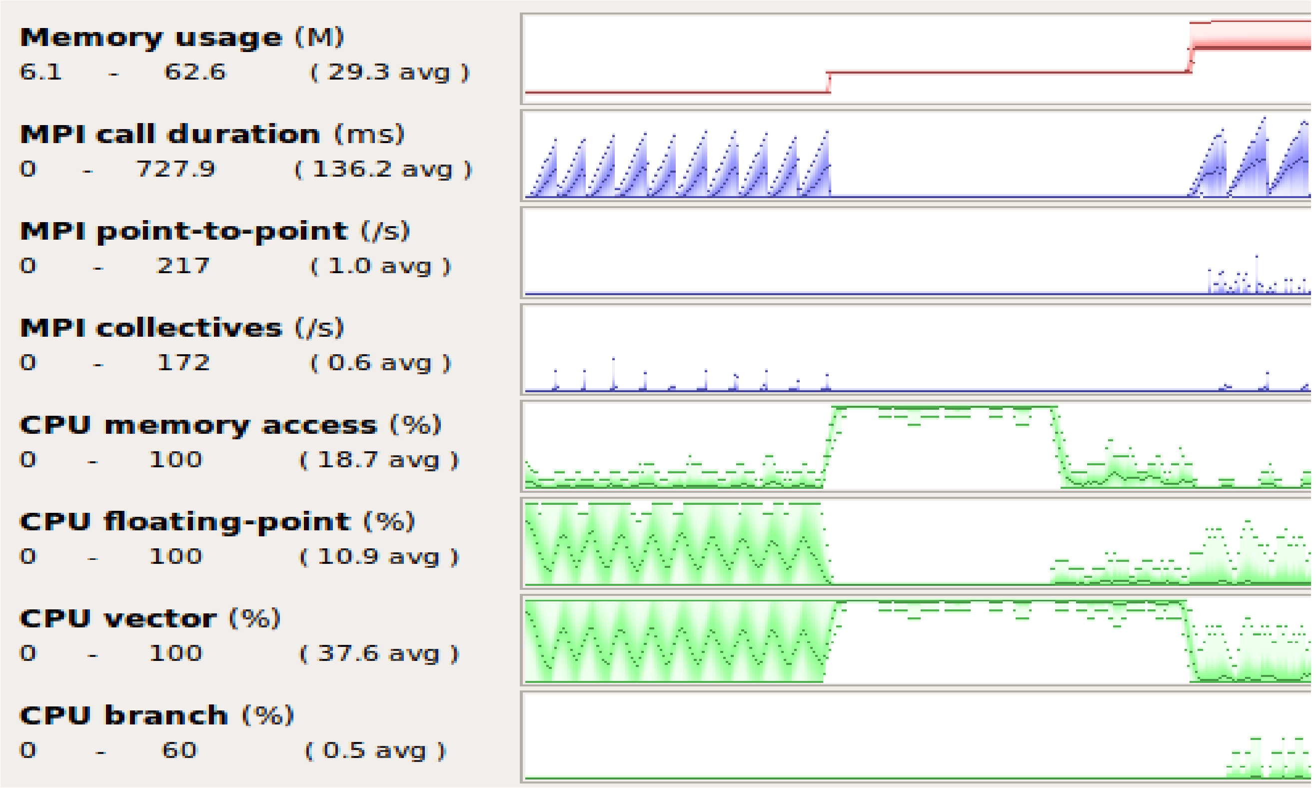
Arm MAP – The Profiler

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



arm

Glean Deep Insight from our Source-Level Profiler



Track memory usage across the entire application over time

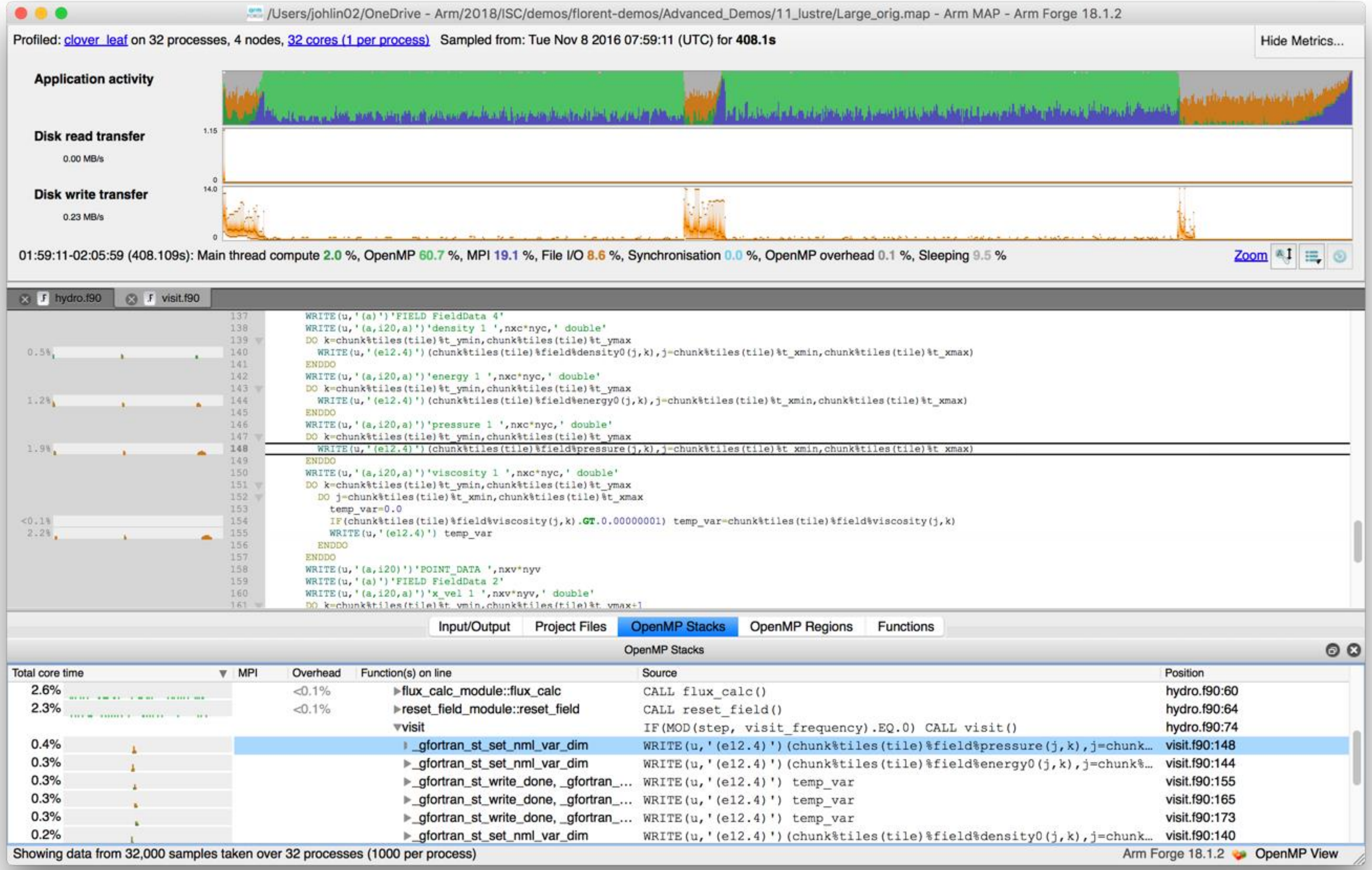
Spot MPI and OpenMP imbalance and overhead

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

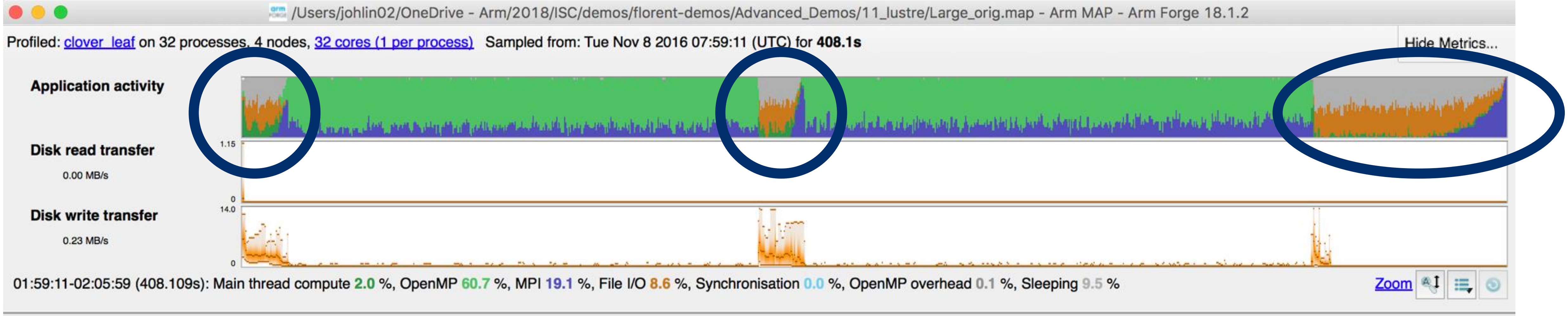
Each I/O operation should take about the same time, but it's not the case.



arm

Symptoms and causes of the I/O issues

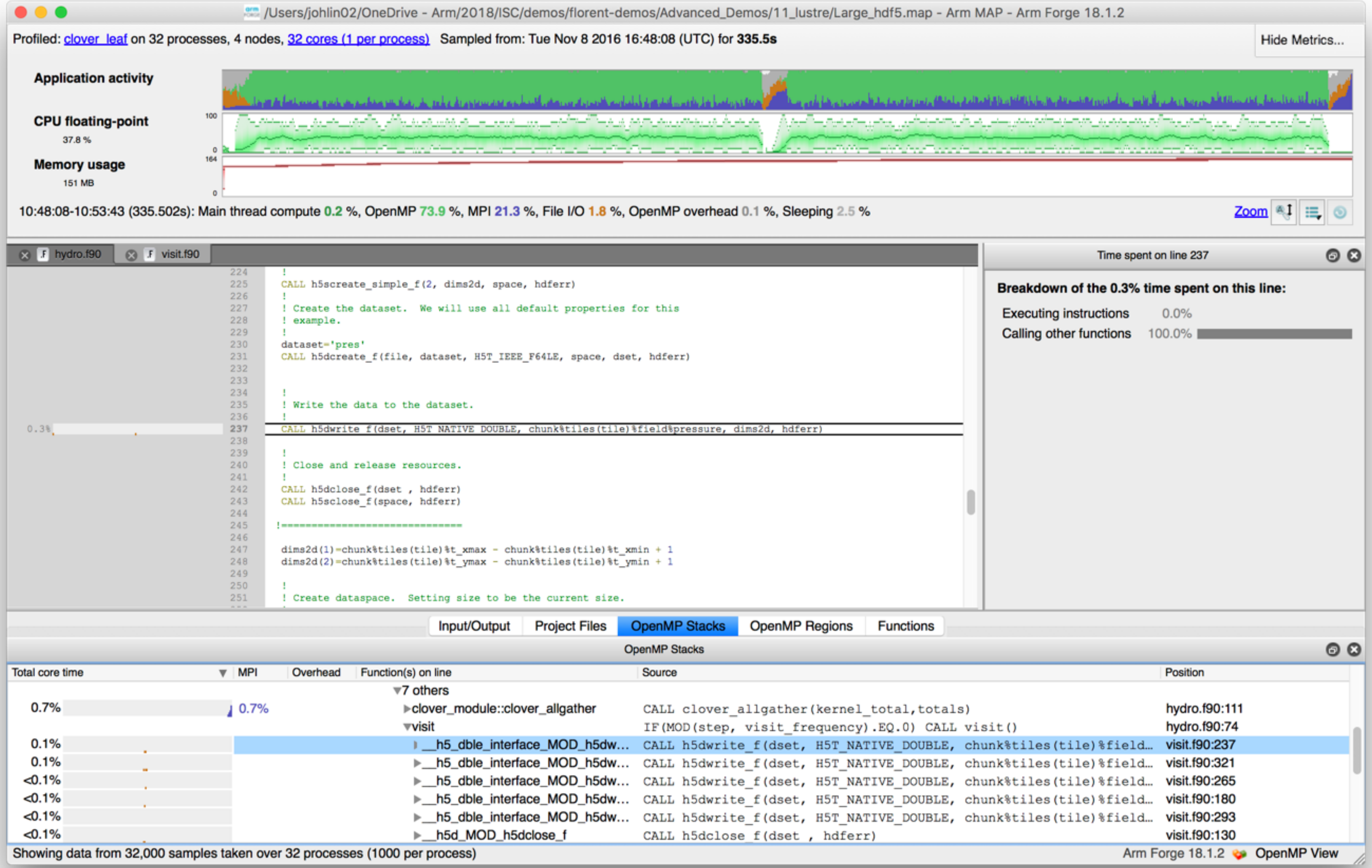
Sub-optimal file format and surprise buffering.



- 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.

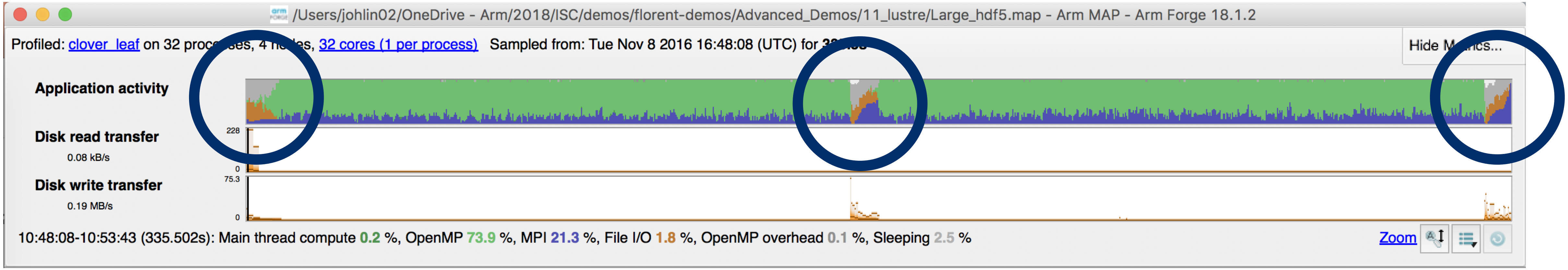
Solution: use HDF5 to write binary files

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

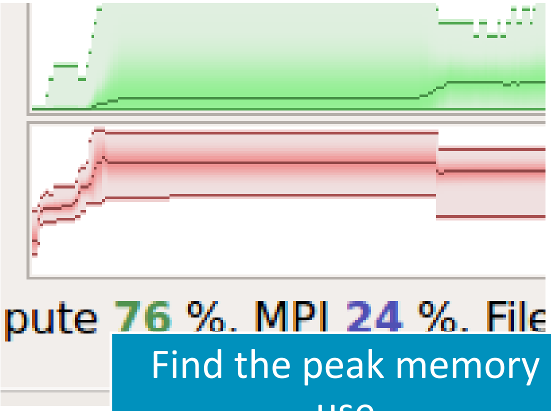
- $ cc -O3 **-g** myapp.c -o myapp.exe **-Wl,@/path/to/profiler_wrapper_libraries/allinea-profiler.ld**
- Edit the job script to run Arm MAP in "profile" mode
- $ **map --profile** aprun -n 8 ./myapp.exe arg1 arg2

Open the results

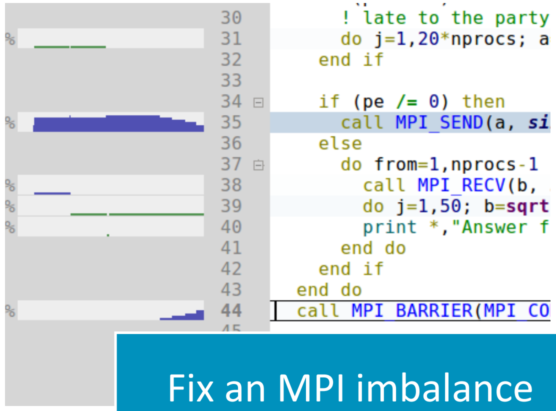
- On the login node:
    - $ map myapp_Xp_Yn_YYYY-MM-DD_HH-MM.map
- (or load the corresponding file using the remote client connected to the remote system or locally)

arm

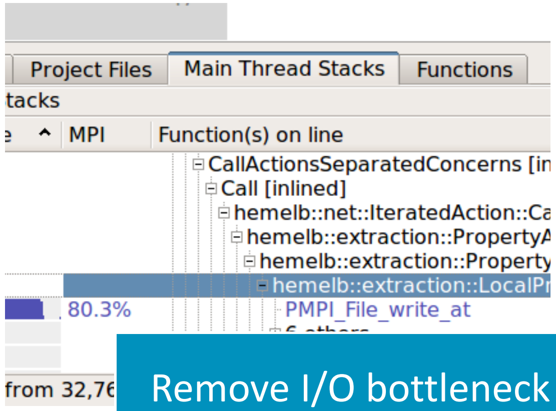
Six Great Things to Try with Allinea MAP



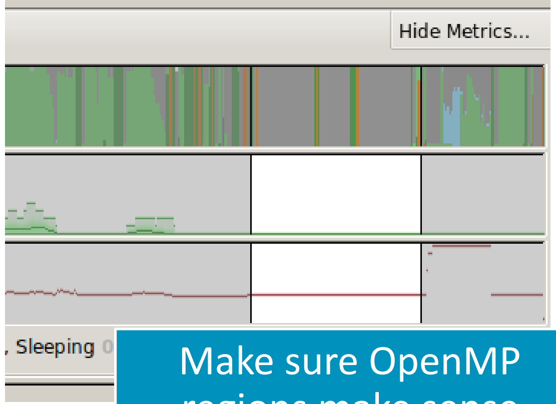
Find the peak memory use



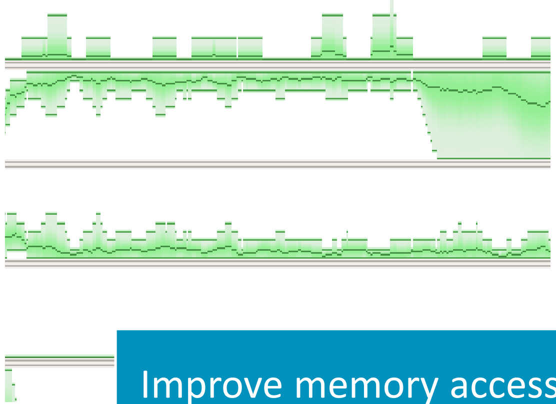
Fix an MPI imbalance



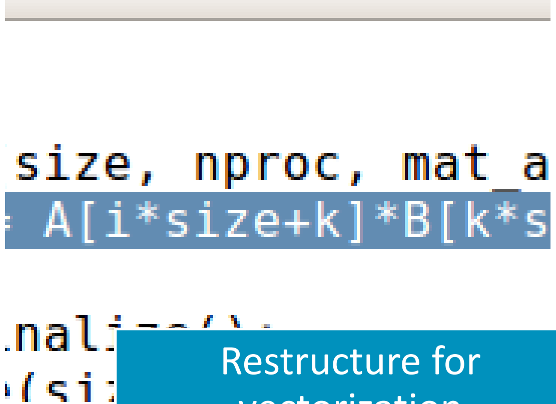
Remove I/O bottleneck



Make sure OpenMP regions make sense



Improve memory access



Restructure for vectorization

Theta Specific Settings

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/ddt`
- 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

Sample usage Commands

Theta

```
rpn=64
ddt aprun -n $((COBALT_JOBSIZE*rpn)) -N $rpn -d $depth -j 1 -cc depth ./myProgram.exe

map aprun -n $((COBALT_JOBSIZE*rpn)) -N $rpn -d $depth -j 1 -cc depth ./myProgram.exe
```

Questions?

arm

Thank You!

Danke!

Merci!

谢谢!

ありがとう!

Gracias!

Kiitos!

감사합니다

धन्यवाद

Arm Forge Hands-on Examples

Hands-on files

The files for the examples that follow can be obtained on theta at the following location

```
/projects/Comp_Perf_Workshop/allinea/allinea_handson.tgz
```

This extracts 2 directories: demonstrations and allinea_examples

The demonstrations are there for you to play with and ask questions

The examples are more like guided exercises

Launch Remote client

Be sure to launch the remote client first

Using a remote launch on your local machine is preferred

Alternatively you can forward X11 when connecting to the login node of theta, and launch it there

```
module load forge/18.2.1
forge &
```

If you accidentally close this window (easy to do), you will have to start it again

Hands-on Examples

These examples are meant to be run on Theta in an interactive session

```
qsub -I -q training -t 120 -n 1 --proccount 64
```

Once a session has been allocated, load the Forge module

```
module load forge/18.2.1
```


Before Generating MAP profiles

Static profiler libraries need to be created before MAP profiles can be generated

Go to the allinea_examples/wrapper directory

Run

```
make-profiler-libraries --lib-type=static
```

The Makefiles for the examples have already been modified to look for the profiler libraries in this directory

Go to exercise 1 – [Bug] Solver is not converging

- **Exercise objectives**
  - Familiarize with DDT user interface
  - Inspect values of u using multidimensional array viewer
  - Set watchpoint for diffnorm_global
  - Set breakpoint at line 89
- **Typical run commands to use:**

```
$> cd allinea_examples/1_debug/
$> make
```

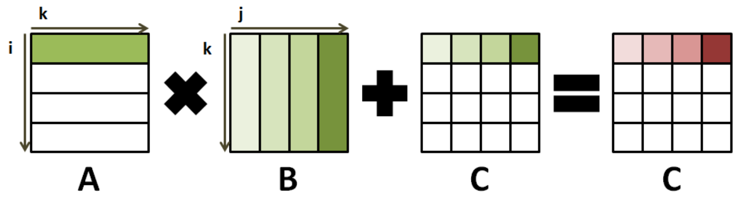
- **Key DDT commands**

```
On the login node:
$> forge &
In a submission file/interactive job:
$> ddt --connect –n 4 ./jacobi.exe
```

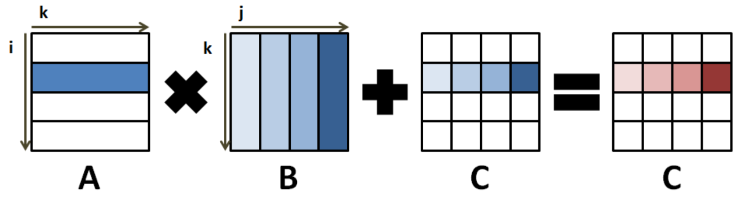
Matrix Multiplication Example

`C = A x B + C`

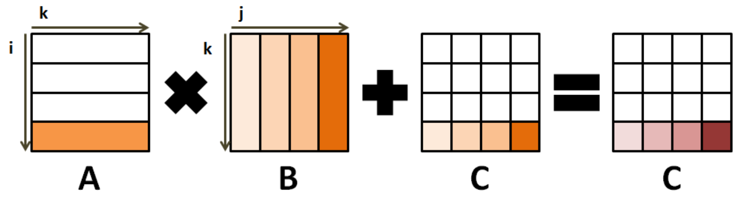
Master process



Slave process 1



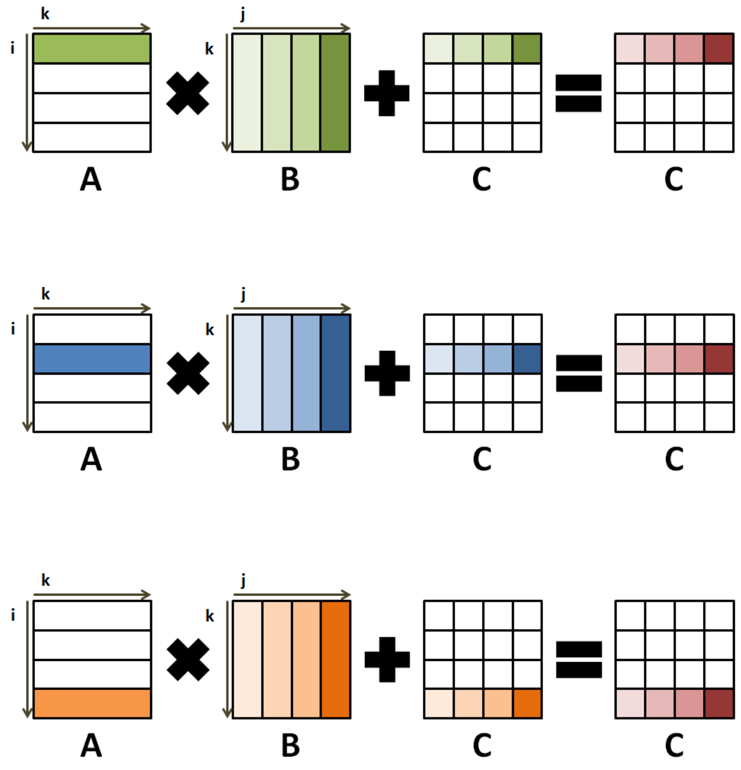
Slave process n-1



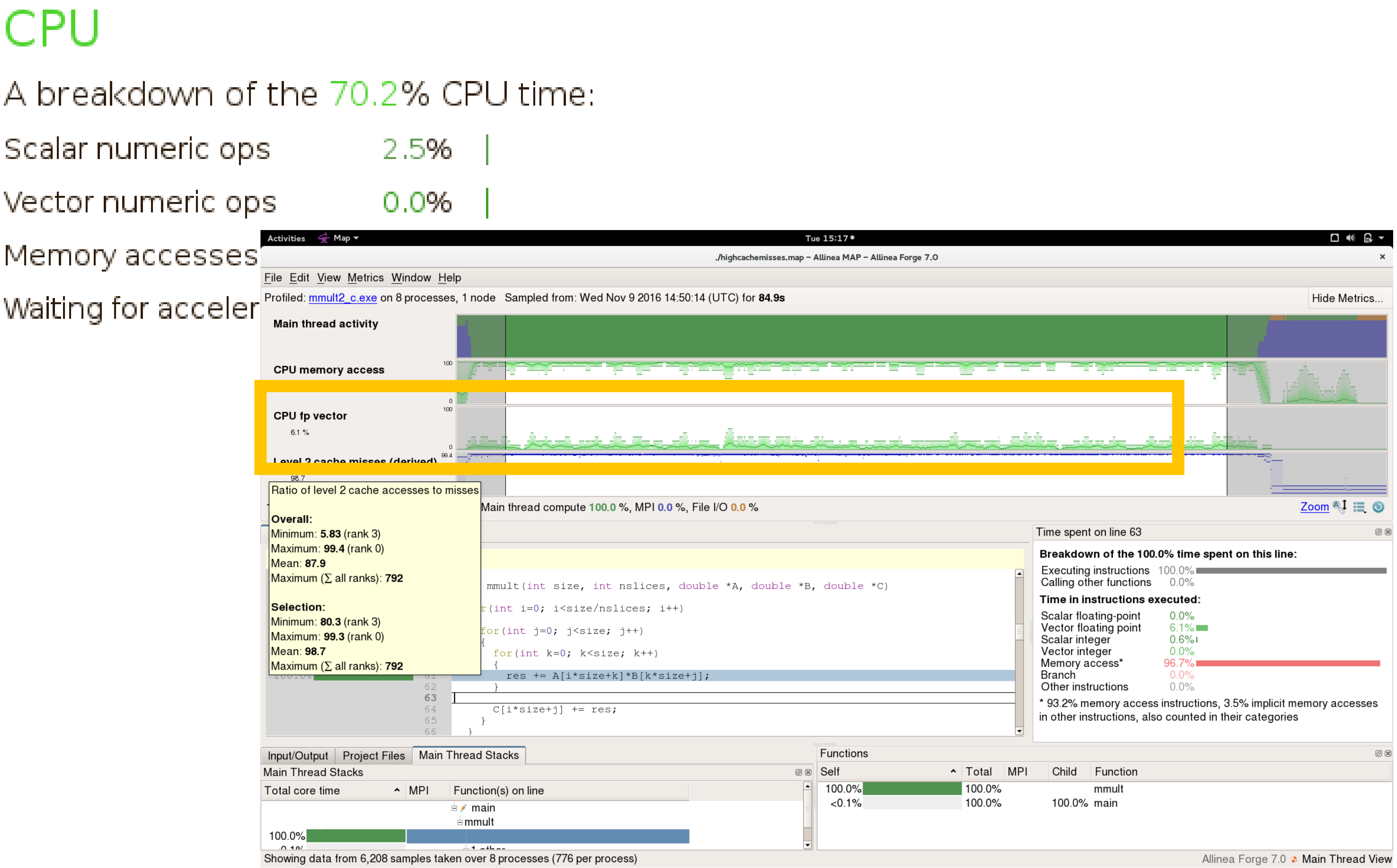
Matrix Multiplication Example Continued

C = A x B + C

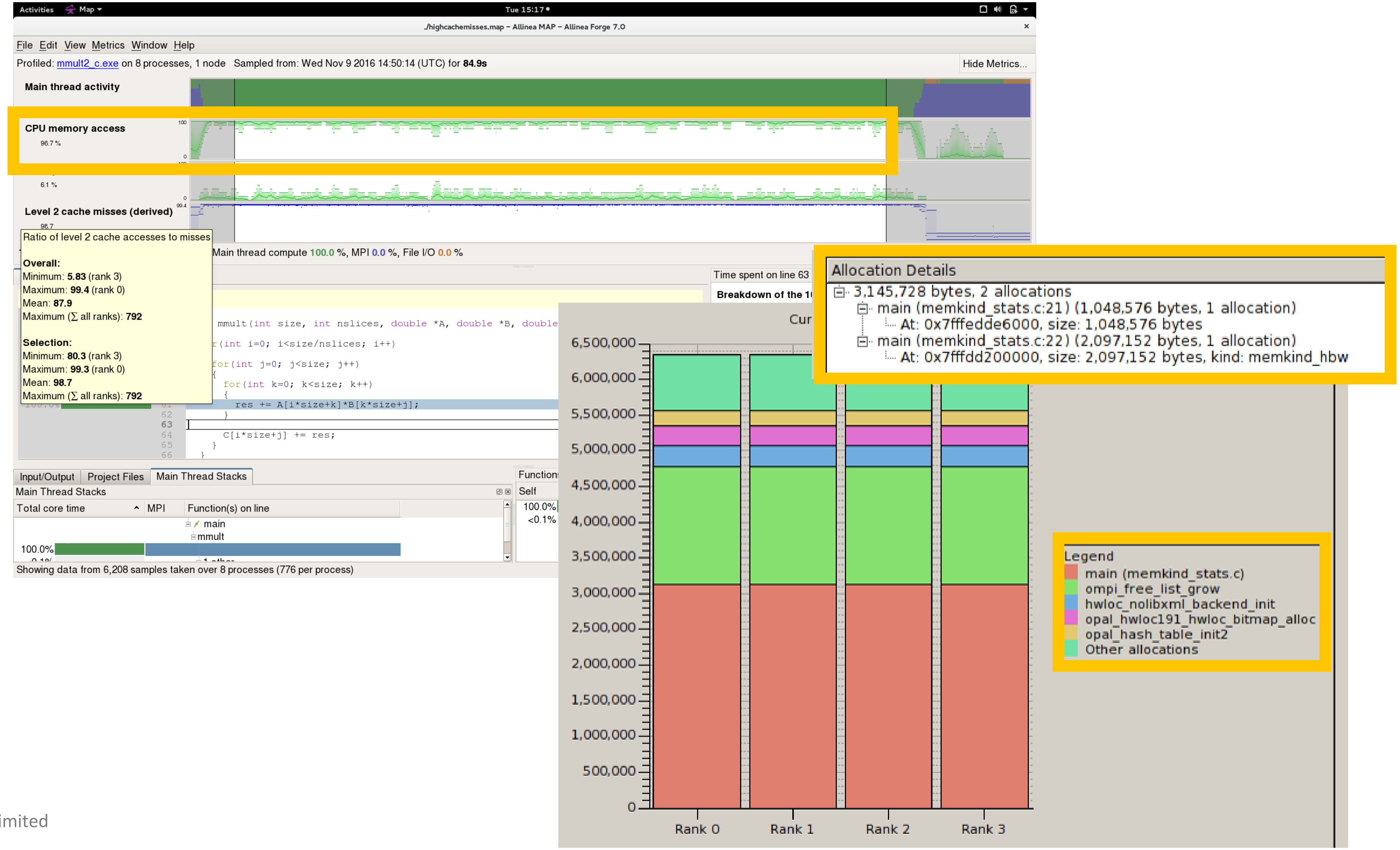
- The "Master" process initializes matrices A, B and C
- The "Master" process sends the whole matrix B along with slices of A and C to the "Slave" processes
- The "Master" and "Slave" processes perform the matrix multiplication function on the domain that has been given to them and everyone computes a slice of C
- The "Master" process retrieves all slices of C and puts the result matrix C together



Use Allinea Forge to vectorize your codes



Use Forge to optimize memory access



Go to exercise 2

- **Exercise objectives**
  - Generate initial baseline profile
  - Ensure the matrices are stored in the MCDRAM (if applicable)
  - Fix the inefficient memory access issues
  - Further enable vectorization with Intel compiler flag **`-xMIC-AVX512`**
  - Generate profile with MAP after applying changes
- **Typical run commands to use:**

```
$> cd allinea_examples/2_memory_accesses/
$> make
```

- **Key Map commands**

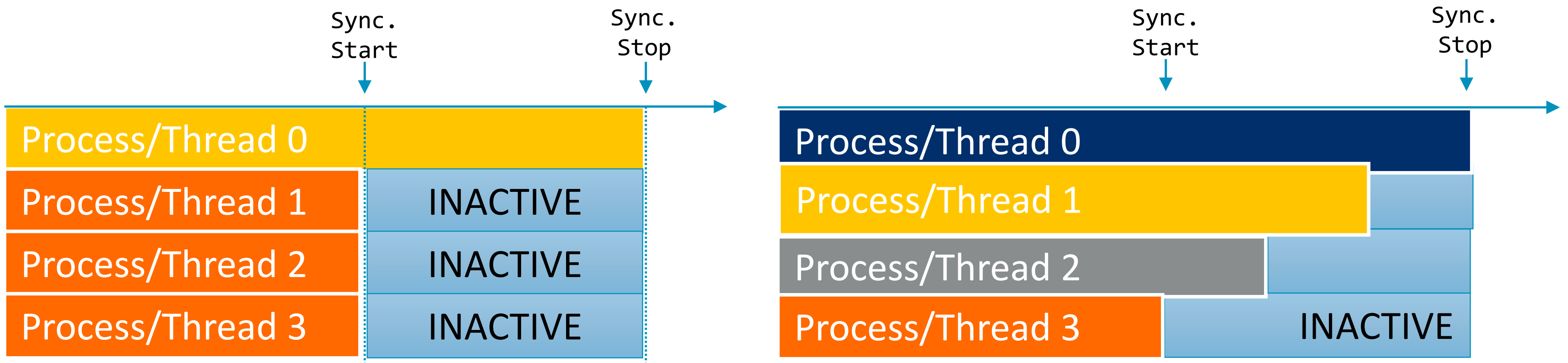
```
On the login node:
$> forge &
In a submission file/interactive job:
$> map --profile -n 64 ./mmult2_c.exe
$> map --connect ./mmult2_c_*.map
```


How to identify load balancing issues?

Problem: "one or some process(es) have too much work"

Clues visible in synchronization

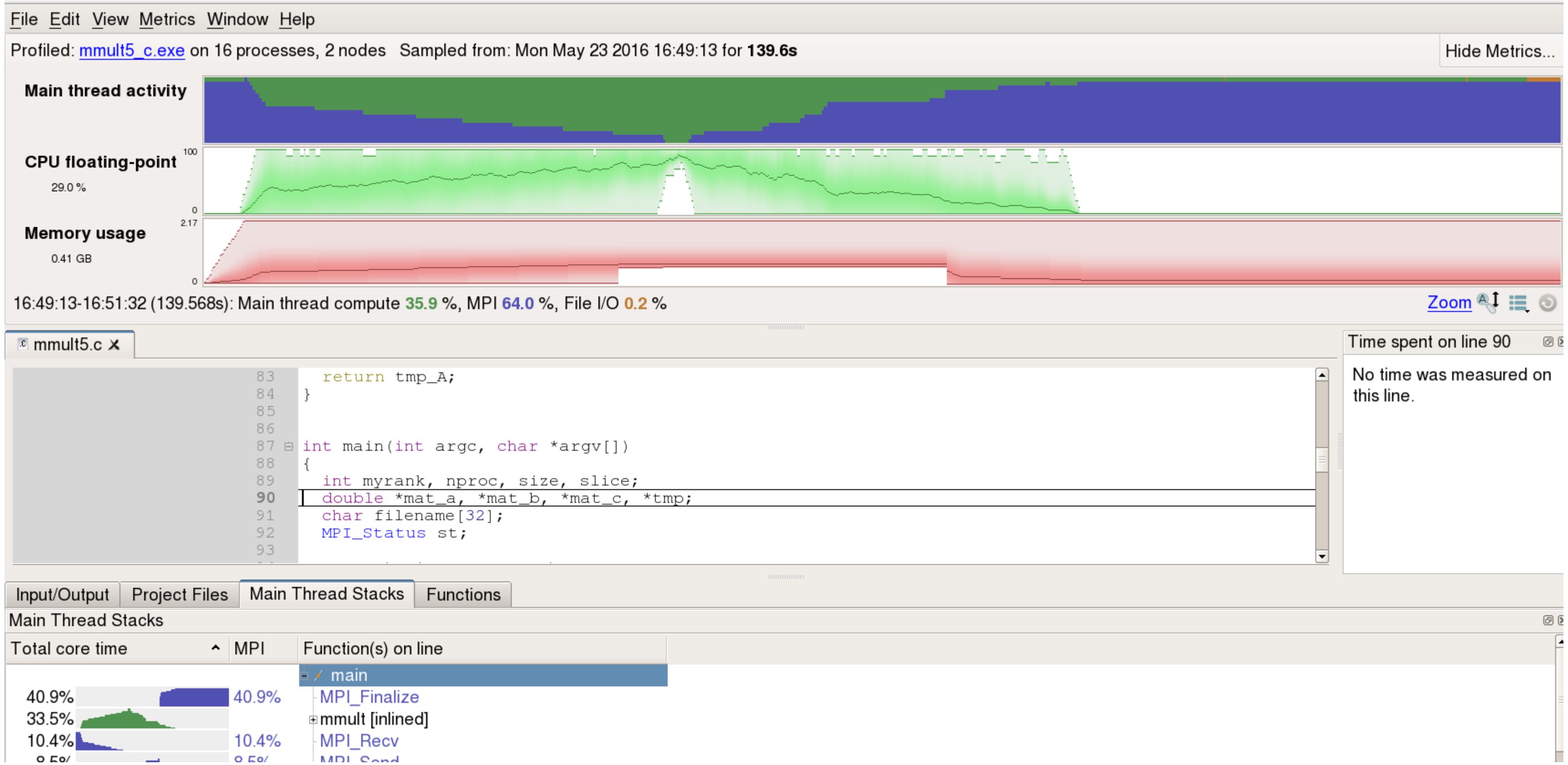
- MPI Collective calls (MPI_Barrier, _Broadcast, etc.) with no actual data transfer
- Idle cores where threads are stuck in locks/mutexes




```
Total runtime: 100 sec
Total CPU time available: 400 sec
Total CPU time actually used: 250 sec
Efficiency: 62.5% of the machine time
```

```
Total runtime: 100 sec
Total CPU time available: 400 sec
Total CPU time actually used: 300 sec
Efficiency: 75% of the machine time
```

Use Allinea MAP to balance your workloads



Go to exercise 3

- **Exercise objectives**
  - Expose workload imbalance issues in the code (preferably on 2 nodes)
  - Make suggestions to fix the problem
- **Typical run commands to use:**

```
$> cd allinea_examples/3_imbalance/
$> make
```

- **Key Map commands**

```
$> map --profile -n 64 ./mmult4_c.exe
$> map --connect mmult4_c_*.map
```

Go to exercise 4

Sometimes optimizations introduce bugs of their own

- **Exercise objectives**
  - Use ddt in offline mode to detect memory leaks
  - Examine the debug_report.txt file
  - Fix the leak
  - Generate new report
- **Typical run commands to use:**

```
$> cd allinea_examples/4_memory_leak/
$> make
```

- **Key ddt commands**

```
$> ddt --offline --mem-debug --output=debug_report.txt -n 64 ./mmult6_c.exe
```

Solutions to exercises

Solutions to these exercises can be found in the `.solution` directory in each of the exercises

