arm

Advanced Arm Forge for MPI Performance Engineering

- John Linford <john.Linford@arm.com>

- 6 August 2018

Tutorial Schedule

- [15] Performance Engineering: Methodology and Tools
- [15] Arm Forge Quick Start: DDT, MAP, and Performance Reports
- [30] Exercises
  - Interactive debugging
  - Profiling from the command line
  - Detect memory leaks
  - Debug invalid memory access
- [30] Break
- [20] Exercises and Examples
  - Explore I/O imbalance with MAP and performance reports
  - Real-world success story
  - Custom metrics for Lustre profiling
- [10] Q&A

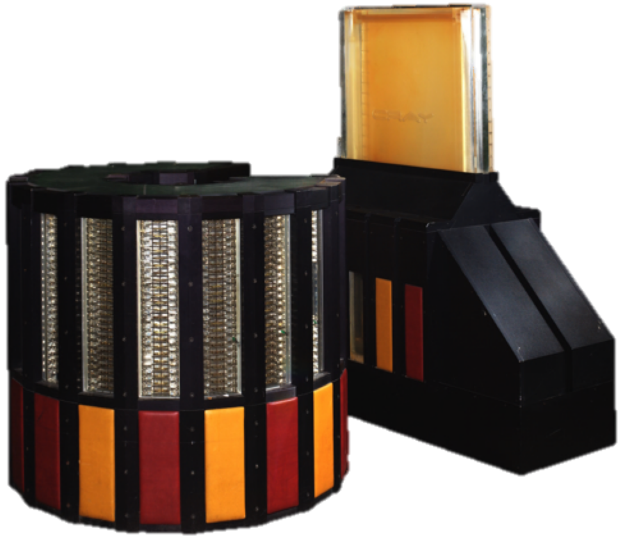
Performance Engineering

Methodology and Tools

Welcome to the age of machine-scale computing

It's dangerous to go alone! Take this.

30 years ago: human-scale computing



Cray 2:

- 4 vector processors
- 1.9 gigaflops (9.5 mflops/Watt)

Today: machine-scale computing

Summit:

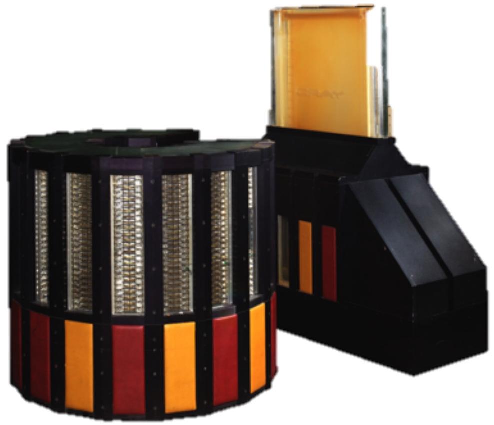
- 2,282,544 cores
- 2,000,000 gigaflops (154 mflops/Watt)

arm

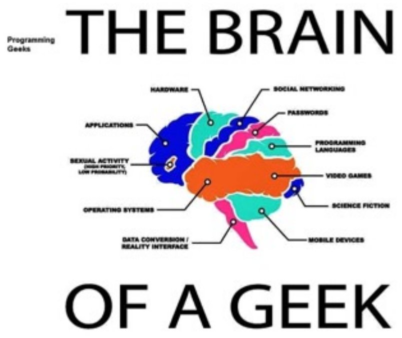
Your brain is no longer enough

No way around it, you need tools to achieve maximum performance.

- Supercomputers are now incomprehensibly complex.
- Naïve optimization may harm performance.
- **Performance engineering tools are essential** for realizing performance at scale.



<



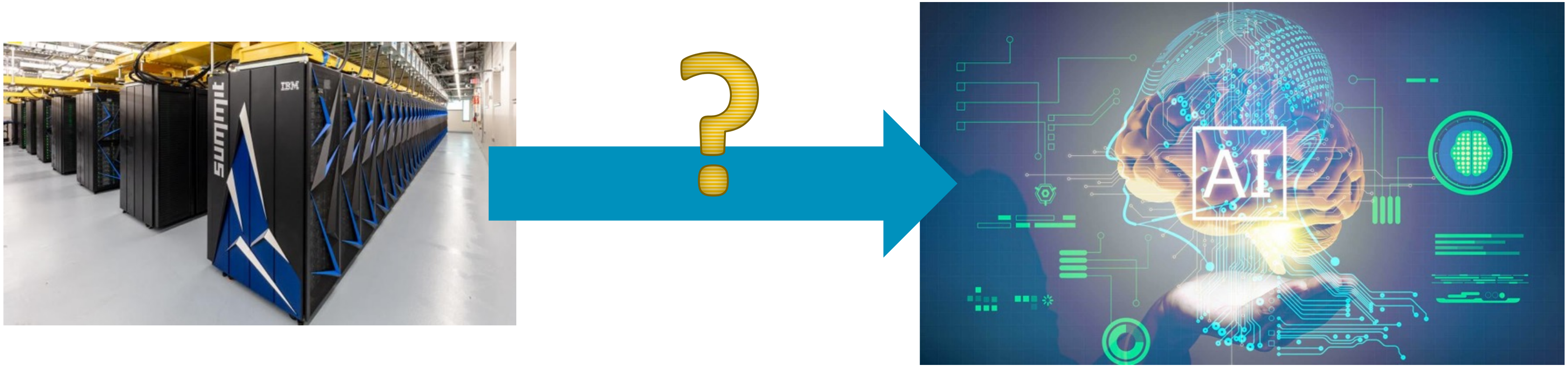
<

arm

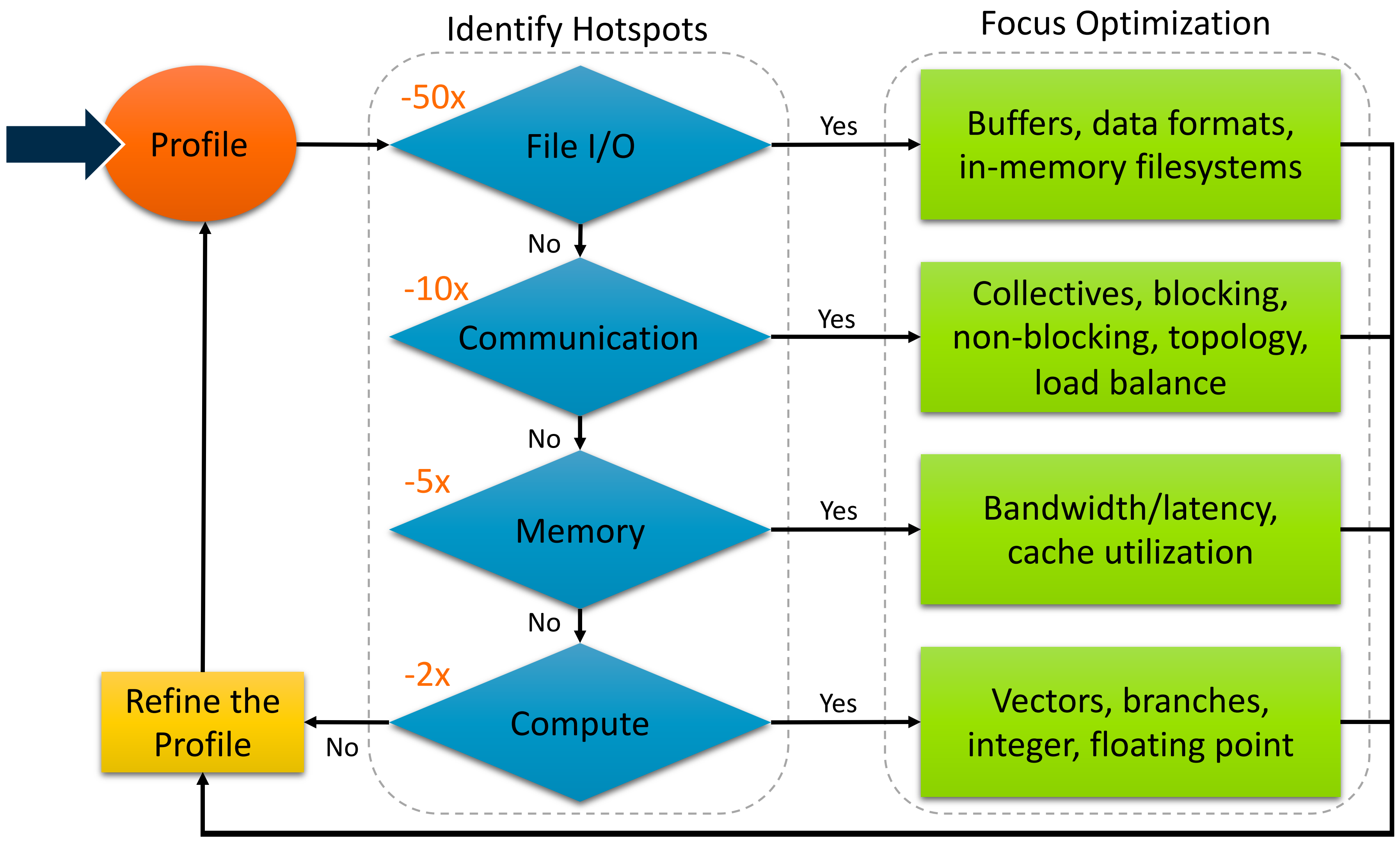
Your brain is no longer enough

No way around it, you need tools to achieve maximum performance.

- Supercomputers are now incomprehensibly complex.
- Naïve optimization may harm performance.
- **Performance engineering tools are essential** for realizing performance at scale.



Identifying and resolving performance issues



Arm's solution for *any* architecture, at *any* scale

Commercial tools for aarch64, x86_64, ppc64 and accelerators

Cross-platform Tools

arm FORGE

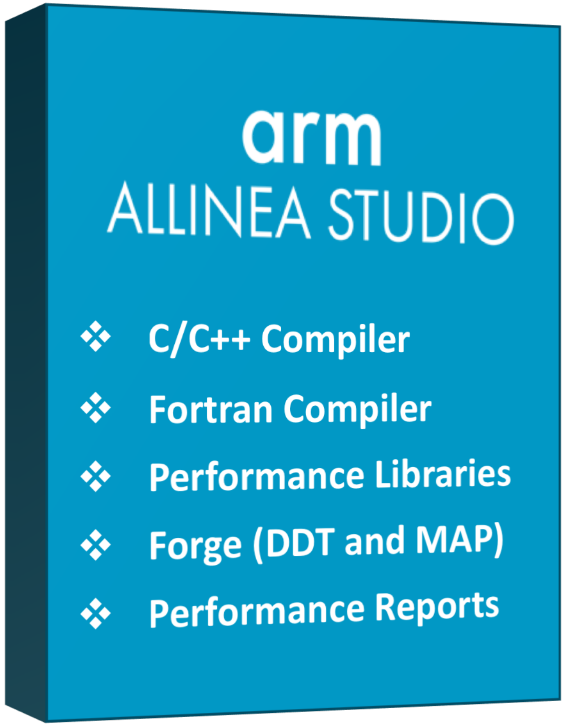
- DDT
- MAP

arm PERFORMANCE REPORTS

Arm Architecture Tools

arm C/C++ & FORTRAN COMPILER

arm PERFORMANCE LIBRARIES



Arm's solution for *any* architecture, at *any* scale

Commercial tools for aarch64, x86_64, ppc64 and accelerators

Cross-platform Tools

arm FORGE

DDT | MAP

arm PERFORMANCE REPORTS

+

Arm Architecture Tools

arm C/C++ & FORTRAN COMPILER

arm PERFORMANCE LIBRARIES

=

arm ALLINEA STUDIO

- C/C++ Compiler
- Fortran Compiler
- Performance Libraries
- Forge (DDT and MAP)
- Performance Reports

Arm Forge = DDT + MAP

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 petaflopic applications)

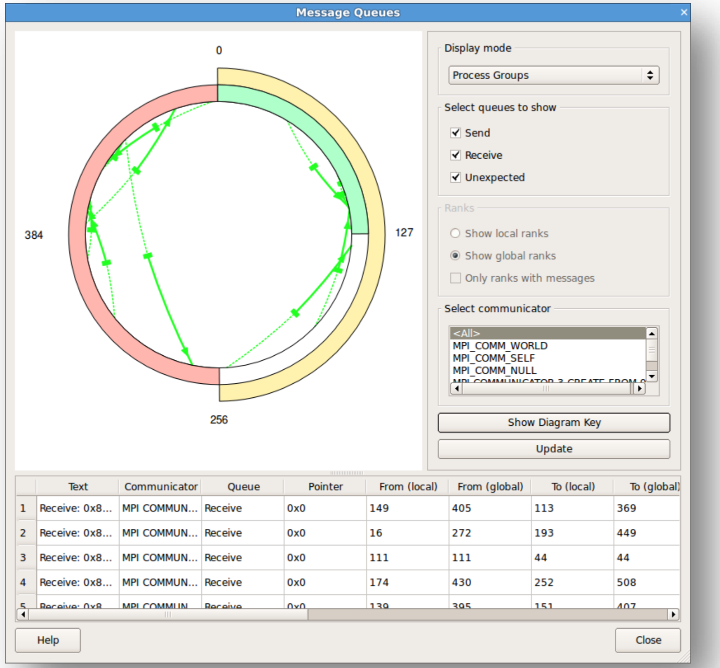
Easy to use by everyone

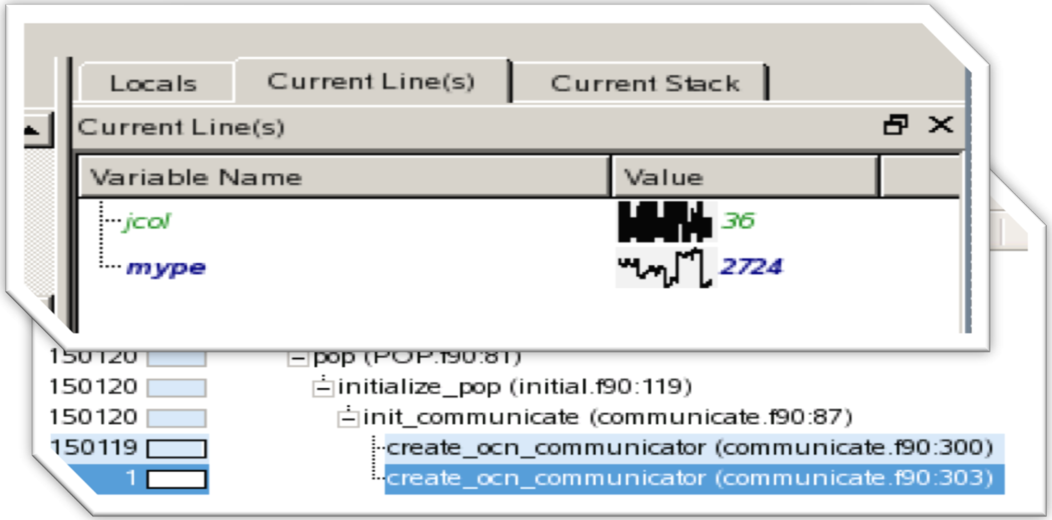
- Unique capabilities to simplify remote interactive sessions
- Innovative approach to present quintessential information to users

DDT: Production-scale debugging

Isolate and investigate faults at scale

- Which MPI rank misbehaved?
  - Merge stacks from processes and threads
  - Sparklines comparing data across processes
- What source locations are related to the problem?
  - Integrated source code editor
  - Dynamic data structure visualization
- How did it happen?
  - Parse diagnostic messages
  - Trace variables through execution
- Why did it happen?
  - Unique “Smart Highlighting”
  - Experiment with variable values

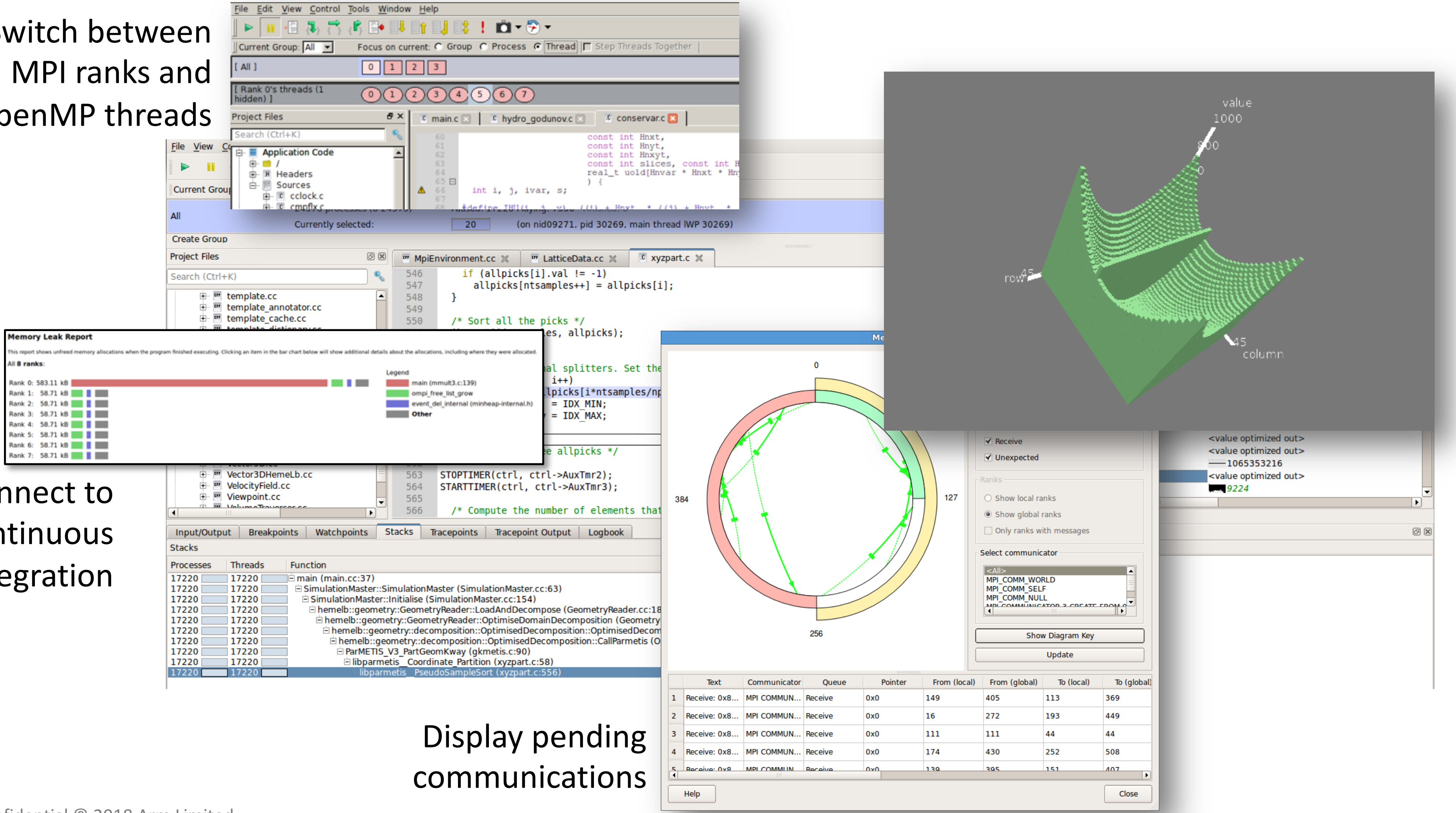




DDT: Feature Highlights

Switch between MPI ranks and OpenMP threads

Visualise data structures



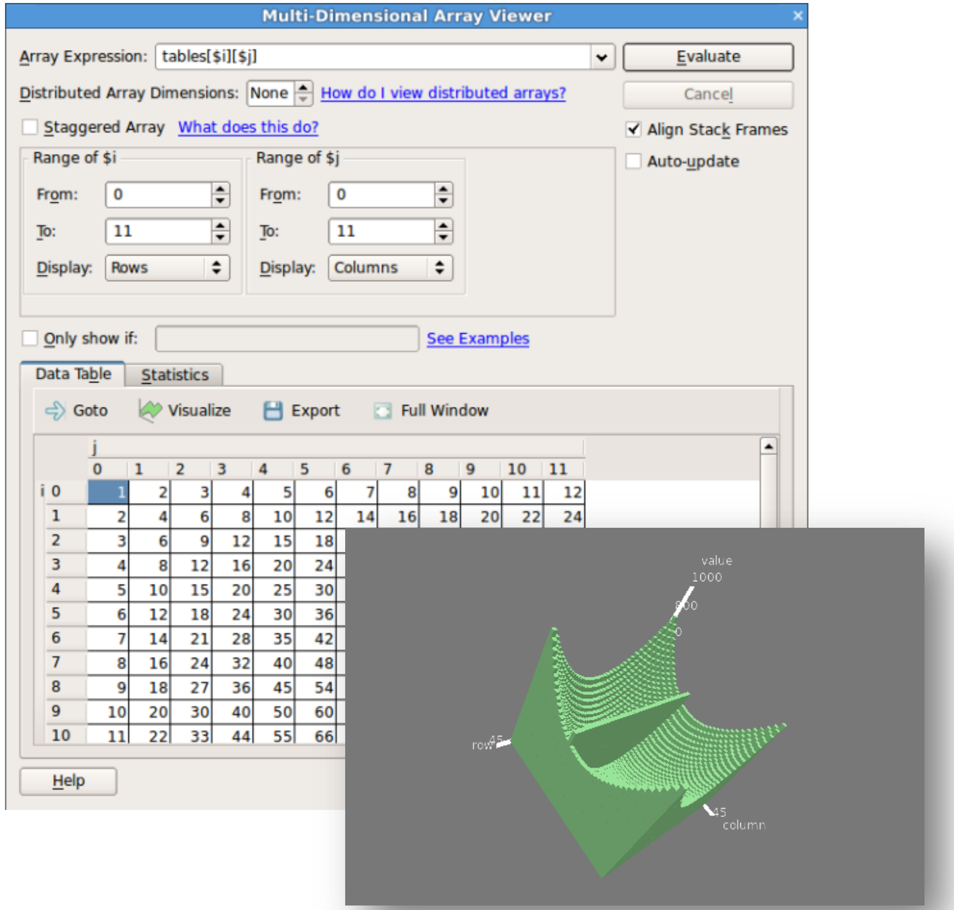
Connect to continuous integration

Display pending communications

Multi-dimensional Array Viewer

What does your data look like at runtime?

- View arrays
  - On a single process
  - Or distributed on many ranks
- Use metavariables to browse the array
  - Example: $i and $j
  - Metavariables are unrelated to the variables in your program.
  - The bounds to view can be specified
  - Visualise draws a 3D representation of the array
- Data can also be filtered
  - “Only show if”: $value > 0 for example $value being a specific element of the array



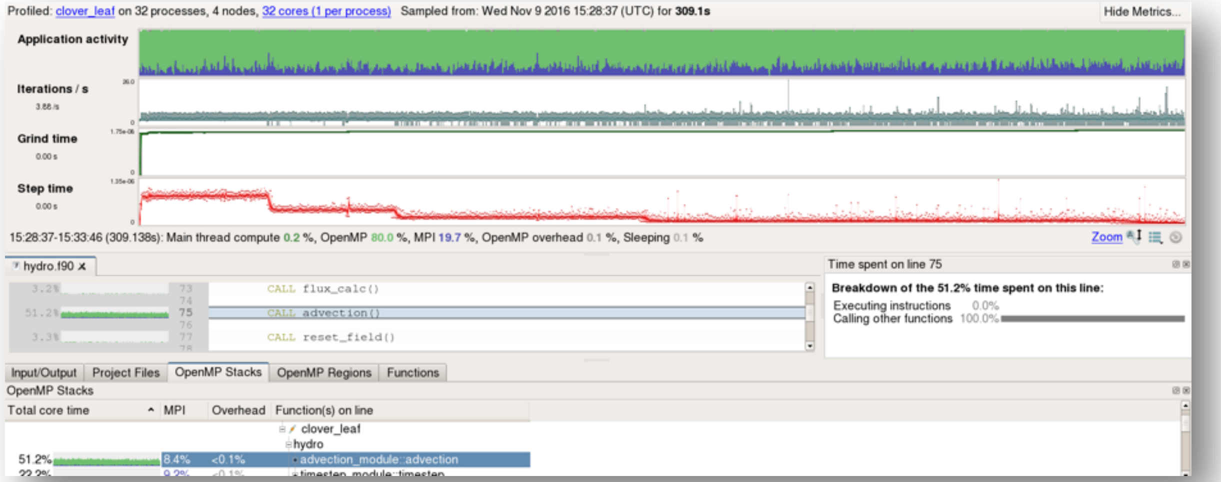
MAP: Production-scale application profiling

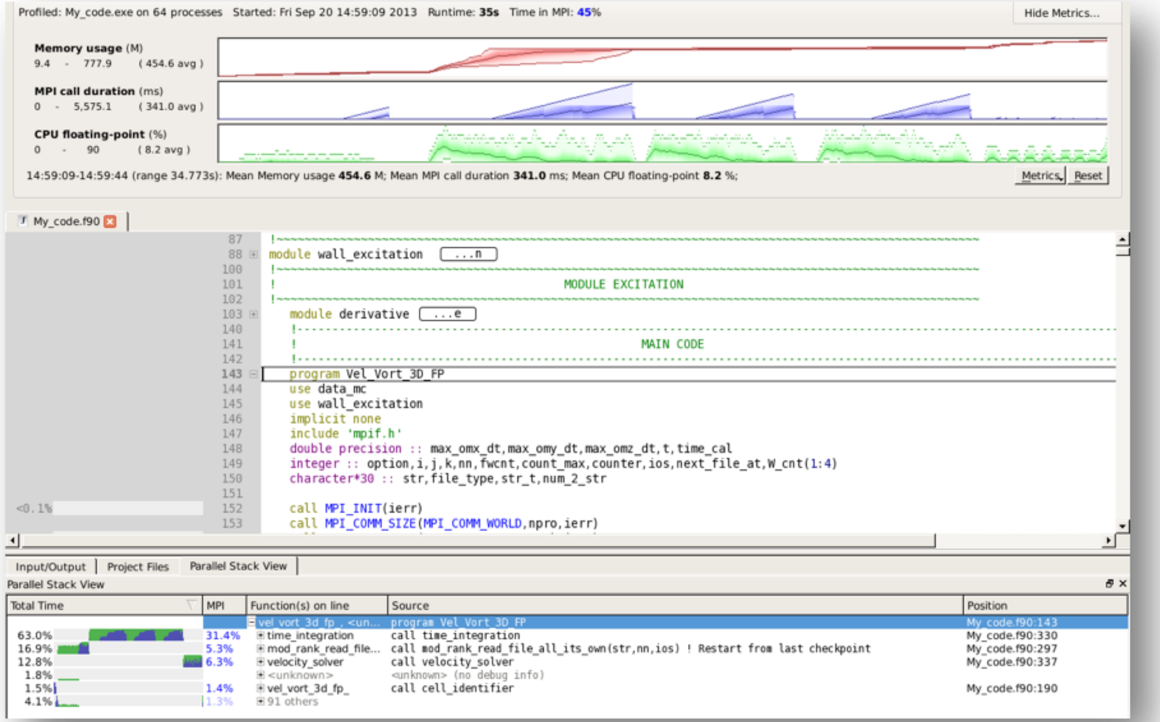
Identify bottlenecks and rewrite code for better performance

- Run with the representative workload you started with
- Measure all performance aspects with Arm Forge Professional

Examples:

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





arm

How MAP is different

MAP’s flagship feature is lightweight, highly scalable performance profiling

Adaptive sampling	Sample frequency decreases over time	Data never grows too much	**Run for as long as you want**
Scalable	Same scalable infrastructure as Allinea DDT	Merges sample data at end of job	**Handles very high core counts, fast**
Instruction analysis	Categorizes instructions sampled	Knows where processor spends time	**Shows vectorization and memory bandwidth**
Thread profiling	Core-time not thread-time profiling	Identifies lost compute time	**Detects OpenMP issues**
Integrated	Part of Forge tool suite	Zoom and drill into profile	**Profiling within your code**

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)

Arm Performance Reports

A high-level view of application performance with “plain English” insights

arm
PERFORMANCE
REPORTS

Command:	mpiexec.hydra -host node-1,node-2 -map-by socket -n 16 -ppn 8 ./Bin/low_freq/../../Src//hydro -i ./Bin/low_freq/../../../../Input/input_250x125_corner.nml
Resources:	2 nodes (8 physical, 8 logical cores per node)
Memory:	15 GiB per node
Tasks:	16 processes, OMP_NUM_THREADS was 1
Machine:	node-1
Start time:	Thu Jul 9 2015 10:32:13
Total time:	165 seconds (about 3 minutes)
Full path:	Bin/../Src

Summary: hydro is MPI-bound in this configuration

Compute	20.6%	Time spent running application code. High values are usually good. This is **very low**; focus on improving MPI or I/O performance first
MPI	63.2%	Time spent in MPI calls. High values are usually bad. This is **high**; check the MPI breakdown for advice on reducing it
I/O	16.2%	Time spent in filesystem I/O. High values are usually bad. This is **average**; check the I/O breakdown section for optimization advice

I/O

A breakdown of the 16.2% I/O time:

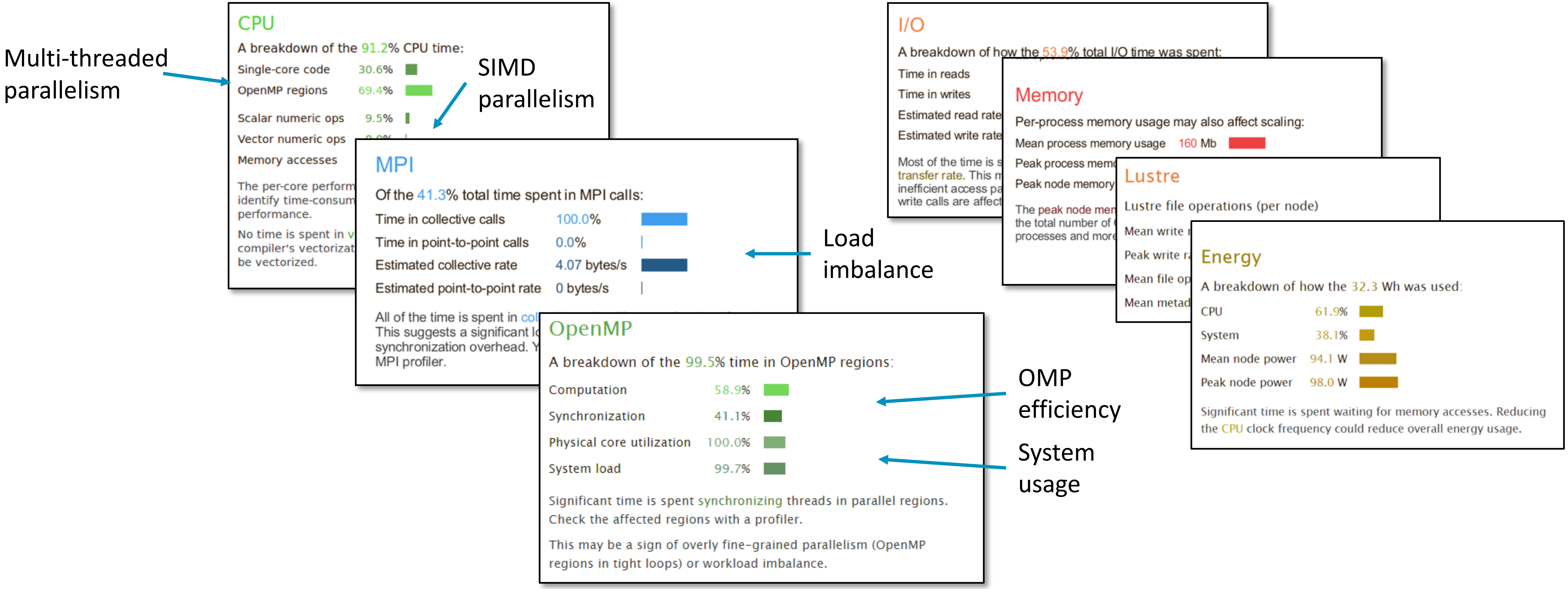
Time in reads	0.0%
Time in writes	100.0%
Effective process read rate	0.00 bytes/s
Effective process write rate	1.38 MB/s

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.

arm

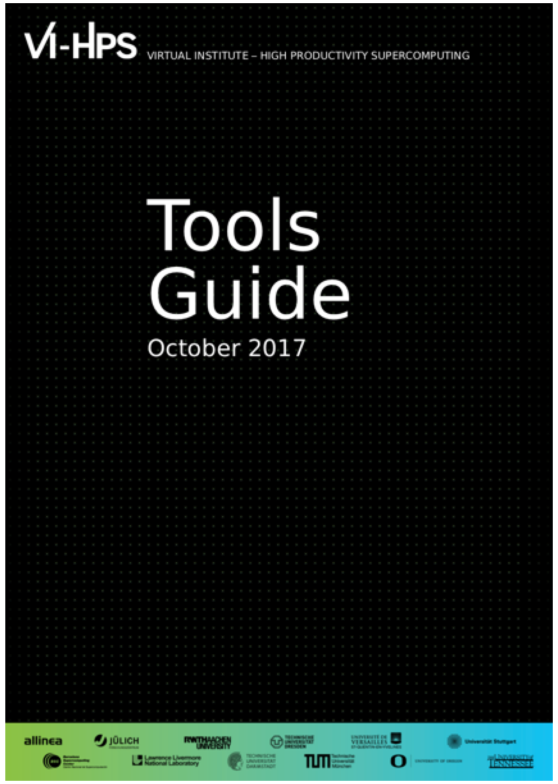
Arm Performance Reports Metrics

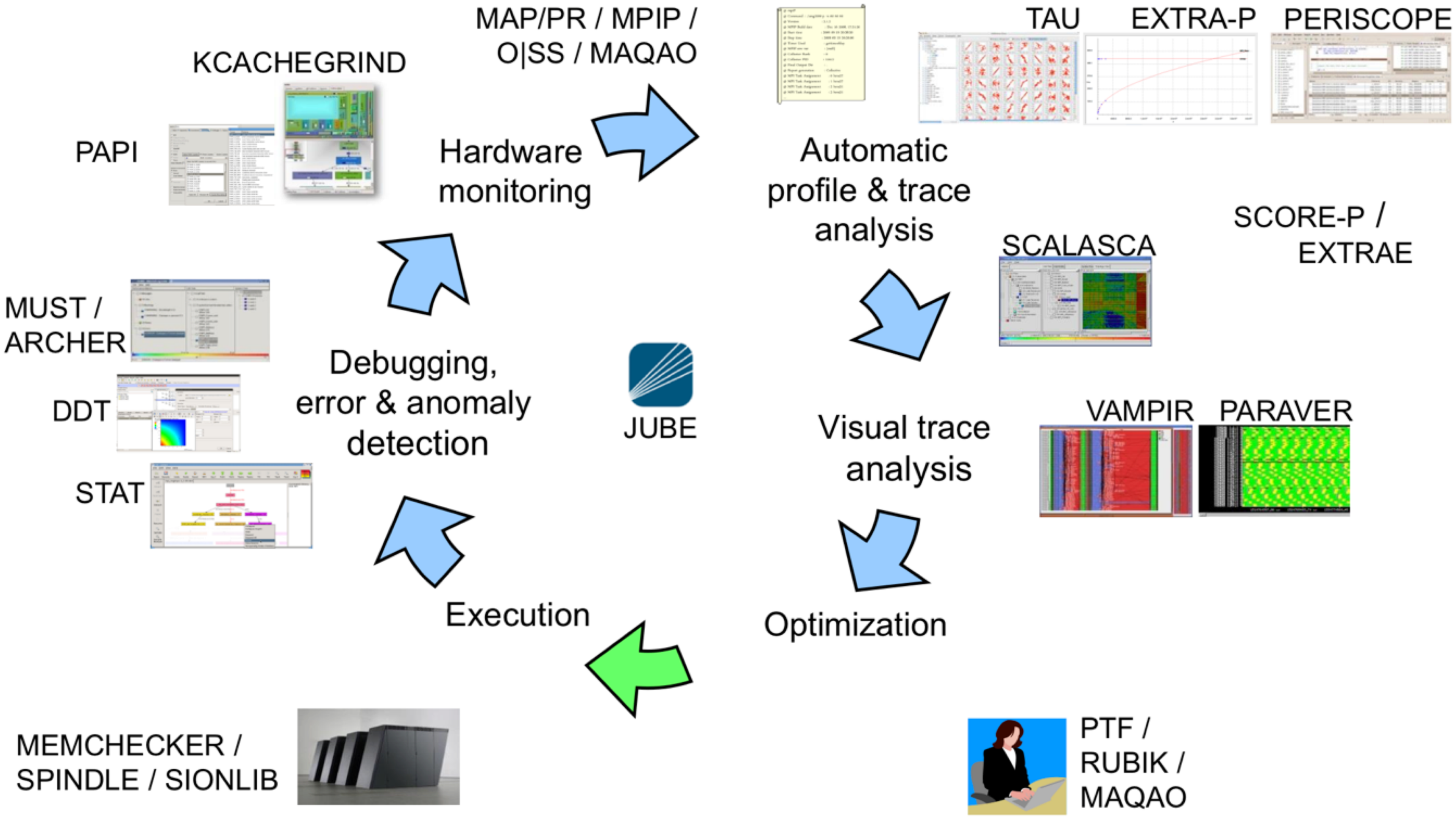
Lowers expertise requirements by explaining everything in detail right in the report.



VI-HPS and the tools ecosystem

See the http://www.vi-hps.org/tools/ for an excellent view of the tools ecosystem.





Arm Forge Quick Start

Tool cheat sheets

arm

Arm DDT cheat sheet

Start DDT interactively, remotely, or from a batch script.

- Load the environment module:
  - $ module load **forge**

- Prepare the code:
  - $ mpicc **-O0 -g** myapp.c -o myapp.exe
  - $ mpfort **-O0 -g** myapp.f -o myapp.exe

- Start DDT in interactive mode:
  - $ **ddt** mpirun -n 8 ./myapp.exe arg1 arg2 …

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

Run DDT in offline mode

Run the application under DDT and halt or report when a failure occurs.

- You can run the debugger in non-interactive mode
  - For long-running jobs
  - For automated testing, continuous integration…

- To do so, use the following arguments:
  - $ ddt **--offline --output=report.html** mpirun ./jacobi_omp_mpi_gnu.exe
    - **--offline** enable non-interactive debugging
    - **--output** specifies the name and output of the non-interactive debugging session
      - Html
      - Txt
    - Add **--mem-debug** to enable memory debugging **and memory leak detection**

DDT command line options

```
$ ddt --help
Arm Forge 18.2.1 - Arm DDT

Usage: ddt [OPTION...] [PROGRAM [PROGRAM_ARGS]]
       ddt [OPTION...] (mpirun|mpiexec|aprun|...) [MPI_ARGS] PROGRAM [PROGRAM_ARGS]

  --connect                                              Reverse Connect (launch as a server and wait)
  --attach=[host1:]pid1,[host2:]pid2... [PROGRAM]        attach to PROGRAM being run by list of host:pid
  --attach-mpi=MPI_PID [--subset=rank1,rank2,rank3,...] [PROGRAM]   attach to processes in an MPI program.
  --break-at=LOCATION[,START:EVERY:STOP] [if CONDITION]  set a breakpoint at LOCATION
  --trace-at=LOCATION[,START:EVERY:STOP],VAR1,VAR2,...   set a tracepoint at LOCATION
  --cuda                                                 enable CUDA
  --mem-debug[=(fast|balanced|thorough|off)]             configure memory debugging (defaults to fast)
  --mpiargs=ARGUMENTS                                    command line arguments to pass to mpirun
  -n, --np, --processes=NUMPROCS                         specify the number of MPI processes
  --nodes=NUMNODES                                       configure the number of nodes for MPI jobs
  --procs-per-node=PROCS                                 configure the number of processes per node
  --offline                                              run through program without user interaction
  -s, --silent                                           don't write unnecessary output to the command line
```

Arm MAP cheat sheet

Generate profiles and view offline

- Load the environment module
  - $ module load **forge**

- Prepare the code
  - $ mpicc **-O0 -g** myapp.c -o myapp.exe
  - $ mpfort **-O0 -g** myapp.f -o myapp.exe

- Offline: edit the job script to run Arm MAP in “profile” mode
  - $ **map --profile** mpirun ./myapp.exe arg1 arg2

- View profile in MAP:
  - 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)

MAP command line options

```
$ map --help
Arm Forge 18.2.1 - Arm MAP

Usage: map [OPTION...] [PROGRAM [PROGRAM_ARGS]]
       map [OPTION...] (mpirun|mpiexec|aprun|...) [MPI_ARGS] PROGRAM [PROGRAM_ARGS]
       map [OPTION...] [MAP_FILE]

  --connect                        Reverse Connect (launch as a server and wait for the GUI to connect)
  --cuda-kernel-analysis           Analysis of the CUDA kernel source code lines
  --list-metrics                   Display metrics IDs which can be explicitly enabled or disabled.
  --disable-metrics=METRICS        Explicitly disable metrics specified by their metric IDs.
  --enable-metrics=METRICS         Explicitly enable metrics specified by their metric IDs.
  --export=FILE.json               Exports a specified .map file as JSON
  --export-functions=FILE          Export all the available columns in the functions view to a CSV file (use --profile)
  --select-ranks=RANKS             Select ranks to profile.
  --mpiargs=ARGUMENTS              command line arguments to pass to mpirun
  -n, --np, --processes=NUMPROCS   specify the number of MPI processes
  --nodes=NUMNODES                 configure the number of nodes for MPI jobs
  --procs-per-node=PROCS           configure the number of processes per node
  --profile                        run through program without user interaction
```


Arm Performance Reports cheat sheet

Generate text and HTML reports from application runs or MAP files

- Load the environment module:
  - $ module load **reports**

- Run the application:
  - **perf-report** mpirun -n 8 ./myapp.exe

- … or, if you already have a MAP file:
  - **perf-report** myapp_8p_1n_YYYY-MM-DD_HH:MM.txt

- Analyze the results
  - $ cat myapp_8p_1n_YYYY-MM-DD_HH:MM.txt
  - $ firefox myapp_8p_1n_YYYY-MM-DD_HH:MM.html

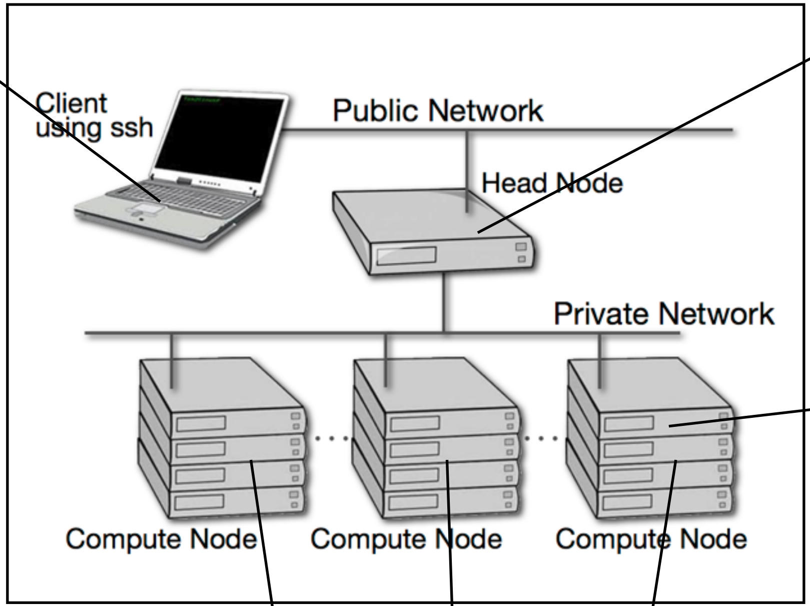
Performance Reports command line options

```
$ perf-report --help
Arm Performance Reports 18.2.1 - Arm Performance Reports

Usage: perf-report [OPTION...] PROGRAM [PROGRAM_ARGS]
       perf-report [OPTION...] (mpirun|mpiexec|aprun|...) [MPI_ARGS] PROGRAM [PROGRAM_ARGS]
       perf-report [OPTION...] MAP_FILE

  --list-metrics                    Display metrics IDs which can be explicitly enabled or disabled.
  --disable-metrics=METRICS         Explicitly disable metrics specified by their metric IDs.
  --enable-metrics=METRICS          Explicitly enable metrics specified by their metric IDs.
  --mpiargs=ARGUMENTS               command line arguments to pass to mpirun
  --nodes=NUMNODES                  configure the number of nodes for MPI jobs
  -o, --output=FILE                 writes the Performance Report to FILE instead of an auto-generated name.
  -n, --np, --processes=NUMPROCS    specify the number of MPI processes
  --procs-per-node=PROCS            configure the number of processes per node for MPI jobs
  --select-ranks=RANKS              Select ranks to profile.
```

The Forge GUI and where to run it

DDT and MAP provide powerful GUIs that can be run in a variety of configurations.

Remote client
(remote launch + reverse connect)



On the head node
(interactive mode + reverse connect)

On the compute node
(offline OR interactive mode)

Ultimately, that's where the tools will run.
But what about the GUI?

arm

Launching the Forge Remote Client

The remote client is a stand-alone application that runs on your local system

Install the Arm Remote Client (Linux, macOS, Windows)

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

Connect to the cluster with the remote client

- Open Forge Remote Client
- Create a new connection: Remote Launch ➔ Configure ➔ Add
  - Hostname: <username>@<hostname>
  - Remote installation directory: </path/to/arm-forge/X.Y/>
- Connect!

Arm Forge 18.1.2 and MVAPICH2

- To use DDT's memory debugging features, **set the environment variable MV2_ON_DEMAND_THRESHOLD to the maximum job size you expect.** This setting should ***not*** be a system wide default; it should be set as needed.
- To use mpirun_rsh with DDT, from *File* → *Options* go to the *System* page, check *Override default mpirun path* and enter mpirun_rsh. You should also add –hostfile <hosts>, where <hosts> is the name of your hosts file, within the *mpirun_rsh arguments* field in the *Run* window.
- To enable message Queue Support MVAPICH 2 must be compiled with the flags --enable-debug --enable-sharedlib. These are not set by default.
- MVAPICH2 MPI programs cannot be started using Express Launch syntax.
  - Do use: "ddt ./a.out" and configure MPI launch parameters in the GUI.
  - ~~Don't use: "ddt mpirun <mpi_args> ./a.out"~~

Interactive Debugging

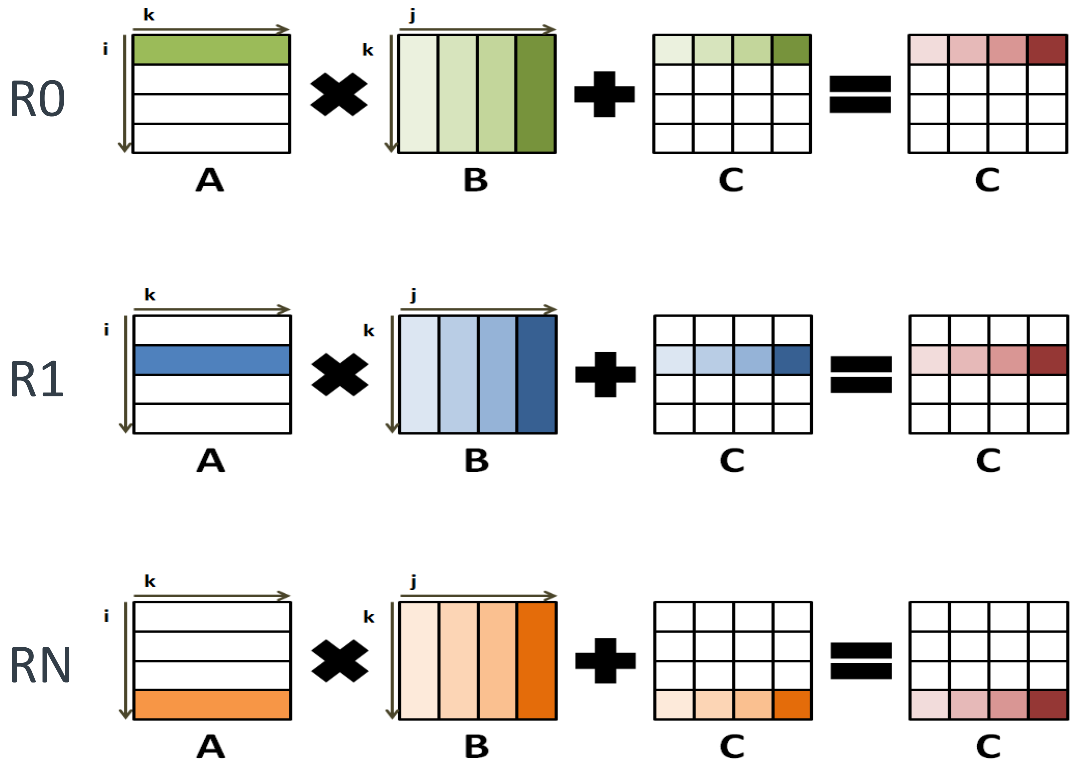
Crash and hang

C = A x B + C

Simply multiply and add two matrices

Algorithm

1. Rank 0 (R0) initialises matrices A, B & C
2. R0 slices the matrices A & C and sends them to Rank 1…N (R1+)
3. R0 and R1+ perform the multiplication
4. R1+ send their results back to R0
5. R0 writes the result matrix C to file



Fix a simple crash in a MPI code

Simple matrix multiply and add? No problem! Except that it crashes…

Exercise Outline

- **Objectives**
  - Discover Arm DDT's interface
  - Interactively debug a crash in a MPI application
- **Commands**

```
$ make
$ mpirun –np 4 ./mmult1_c.exe
# Observe crash
$ ddt ./mmult1_c.exe
# Observe cause of crash
```

Initial Result: Crash!

```
johlin02@johlin02-VM: ~/MUG18/01_walkthrough/1_crash
johlin02@johlin02-VM:~/MUG18/01_walkthrough/1_crash$ make
mpicc -g -ffast-math -O0 -DDEBUG -std=c99 mmult1.c -o mmult1_c.exe -lm
mpif90 -g -ffast-math -O0 -DDEBUG -cpp mmult1.f90 -o mmult1_f90.exe -lm
johlin02@johlin02-VM:~/MUG18/01_walkthrough/1_crash$ mpirun -np 4 ./mmult1_c.exe
0: Size of the matrices: 64x64
0: Initializing matrices...
0: Sending matrices...
0: Processing...
[johlin02-VM:mpi_rank_0][error_sighandler] Caught error: Segmentation fault (signal 11)
3: Receiving matrices...
2: Receiving matrices...
1: Receiving matrices...
2: Processing...
[johlin02-VM:mpi_rank_2][error_sighandler] Caught error: Segmentation fault (signal 11)
1: Processing...
[johlin02-VM:mpi_rank_1][error_sighandler] Caught error: Segmentation fault (signal 11)

===================================================================================
=   BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
=   PID 9160 RUNNING AT johlin02-VM
=   EXIT CODE: 139
=   CLEANING UP REMAINING PROCESSES
=   YOU CAN IGNORE THE BELOW CLEANUP MESSAGES
===================================================================================
YOUR APPLICATION TERMINATED WITH THE EXIT STRING: Segmentation fault (signal 11)
This typically refers to a problem with your application.
Please see the FAQ page for debugging suggestions
johlin02@johlin02-VM:~/MUG18/01_walkthrough/1_crash$ $
```

Answer: Fix incorrect limits on k-loop

Incorrect limits lead to invalid memory access

Before

```
do i=0,size/nslices-1
  do j=0,size-1
    res=0.0
    do k=size,size*size
     res=A(i*size+k)*B(k*size+j)+res
    end do
    C(i*size+j)=res+C(i*size+j)
  end do
end do
```

After

```
do i=0,size/nslices-1
  do j=0,size-1
    res=0.0
    do k=0,size-1
     res=A(i*size+k)*B(k*size+j)+res
    end do
    C(i*size+j)=res+C(i*size+j)
  end do
end do
```


Answer: Fix incorrect limits on i-loop

Incorrect limits on i-loop lead to unmatched MPI_Send

Before

```
do i=1,nproc-2
  call MPI_Send(mat_a(slice*i), slice, &
                MPI_DOUBLE, i, 100+i, &
                MPI_COMM_WORLD, ierr)
  call MPI_Send(mat_b, size*size, &
                MPI_DOUBLE, i, 200+i, &
                MPI_COMM_WORLD, ierr)
  call MPI_Send(mat_c(slice*i), slice, &
                MPI_DOUBLE, i, 300+i, &
                MPI_COMM_WORLD, ierr)
end do
```

After

```
do i=1,nproc-1
  call MPI_Send(mat_a(slice*i), slice, &
                MPI_DOUBLE, i, 100+i, &
                MPI_COMM_WORLD, ierr)
  call MPI_Send(mat_b, size*size, &
                MPI_DOUBLE, i, 200+i, &
                MPI_COMM_WORLD, ierr)
  call MPI_Send(mat_c(slice*i), slice, &
                MPI_DOUBLE, i, 300+i, &
                MPI_COMM_WORLD, ierr)
end do
```

Improve performance

Efficient memory access

Fix inefficient memory access pattern

It works! But wow it's slow.

Exercise Outline

- **Objectives**
  - Discover Arm MAP's interface
  - Gather initial profiles of a MVAPICH2 application
- **Commands**

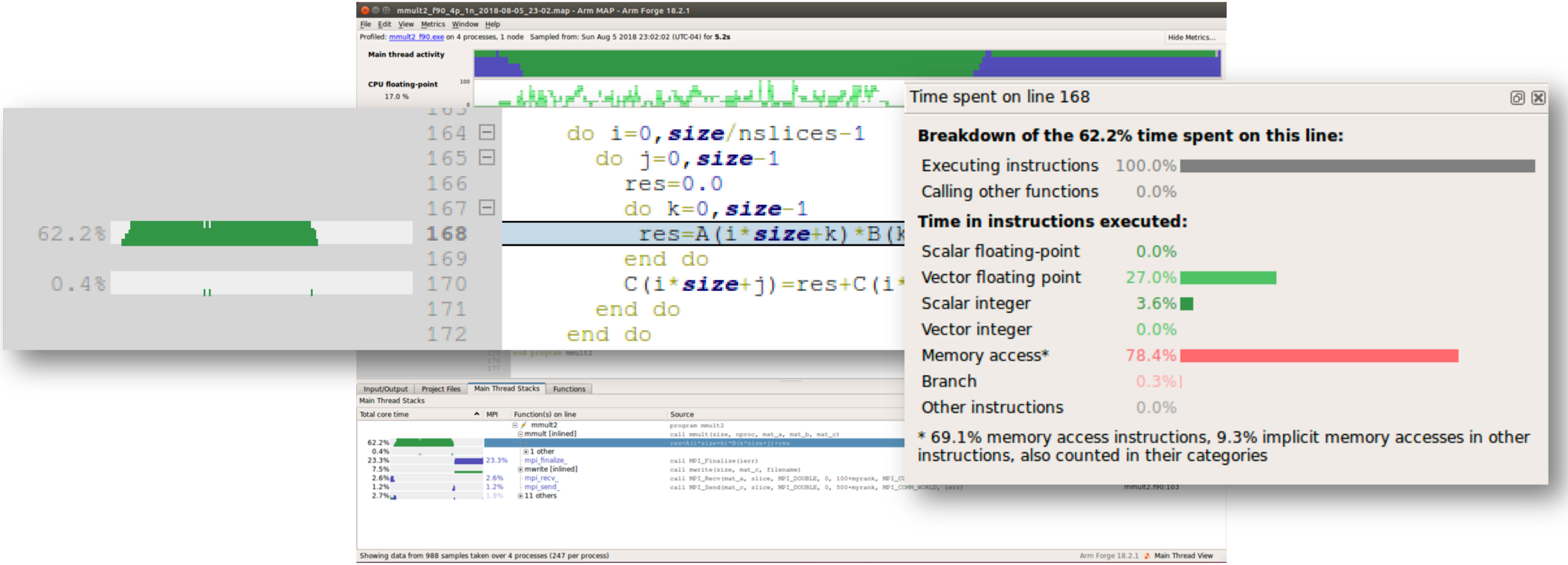
```
$ make
$ map --profile -n 4 \
    ./mmult2_f90.exe
$ map mmult2_f90_4p*.map
# Observe profile
```

Initial Result: SLOW

```
johlin02@johlin02-VM: ~/MUG18/01_walkthrough/2_memory_accesses
johlin02@johlin02-VM:~/MUG18/01_walkthrough/2_memory_accesses$ map --profile -n 4  ./mmult2_f90.exe
Arm Forge 18.2.1 - Arm MAP

Profiling             : /home/johlin02/MUG18/01_walkthrough/2_memory_accesses/mmult2_f90.exe
Allinea sampler       : not preloading
MPI implementation    : Auto-Detect (MVAPICH 2)
* number of processes : 4
* number of nodes     : 1
* Allinea MPI wrapper : not preloading

           1 : Receiving matrices...
           0 : Size of the matrices:         1024 x        1024
           2 : Receiving matrices...
           3 : Receiving matrices...
           0 : Initializing matrices...
           0 : Sending matrices...
           1 : Processing...
           2 : Processing...
           0 : Processing...
           3 : Processing...
           2 : Sending result matrix...
           1 : Sending result matrix...
           3 : Sending result matrix...
           0 : Receiving result matrix...
           0 : Writing results...
           0 : Done.

MAP analysing program...
MAP gathering samples...
MAP generated /home/johlin02/MUG18/01_walkthrough/2_memory_accesses/mmult2_f90_4p_1n_2018-08-05_23-0
2.map
johlin02@johlin02-VM:~/MUG18/01_walkthrough/2_memory_accesses$
```


Initial profile

Find the hotspot: look for the line with the highest core time.



Memory access patterns

- Data locality
  - Temporal locality: use of data within a short time of its last use
  - Spatial locality: use memory references close to memory already referenced

Temporal locality example

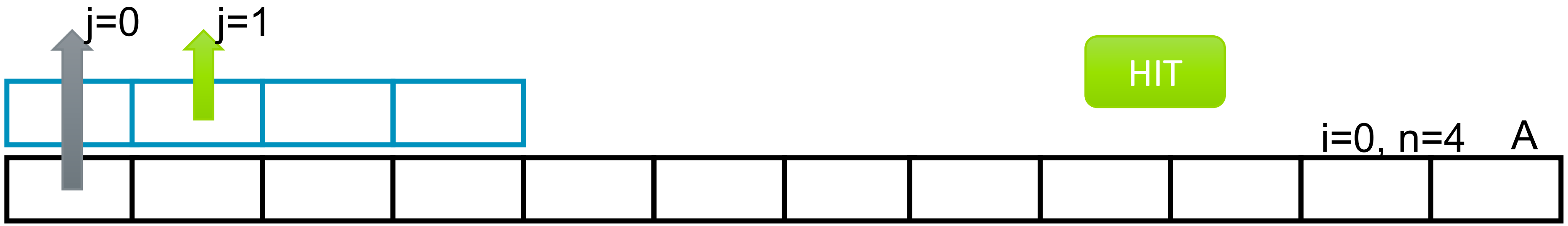
```
for (i=0 ; i < N; i++) {
    for (loop=0; loop < 10; loop++) {
        … = … x[i] …
    }
}
```

Spatial locality example

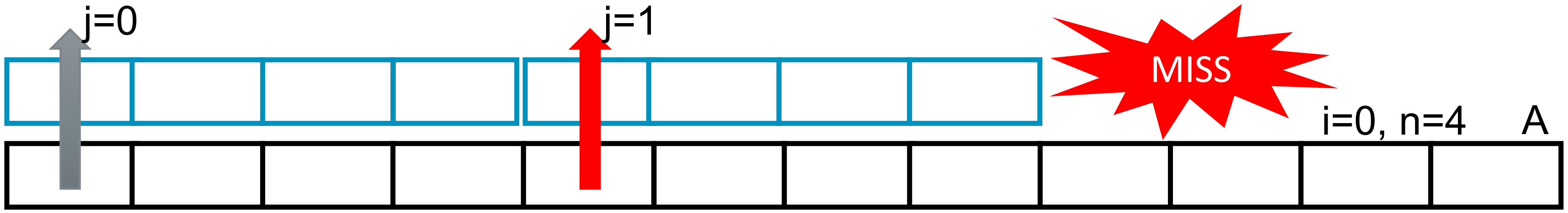
```
for (i=0 ; i < N*s; i+=s) {
    … = … x[i] …
}
```

Memory Accesses and Cache Misses

```
for(i=0; i<n; i++) {
  for(j=0; j<n; j++) {
    A[i*n+j]=…
  }
}
```




```
for(i=0; i<n; i++) {
  for(j=0; j<n; j++) {
    A[j*n+i]=…
  }
}
```



Answer: Transpose matrix and interchange loops

Transposing the matrix improves locality → performance

Before

```
do i=0,size/nslices-1
  do j=0,size-1
    res=0.0
    do k=0,size-1
     res=A(i*size+k)*B(k*size+j)+res
    end do
    C(i*size+j)=res+C(i*size+j)
  end do
end do
```

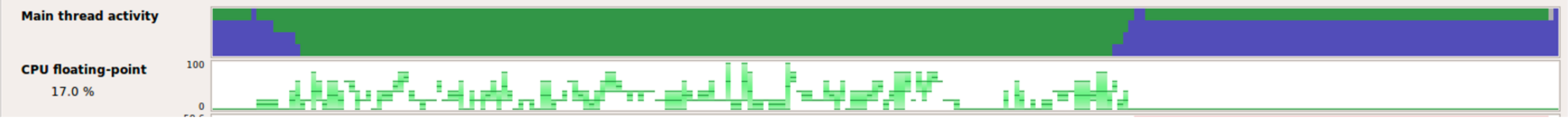
After

```
do i=0,size/nslices-1
  do j=0,size-1
    res=0.0
    do k=0,size-1
     res=A(i*size+k)*transB(j*size+k)+res
    end do
    C(i*size+j)=res+C(i*size+j)
  end do
end do
```

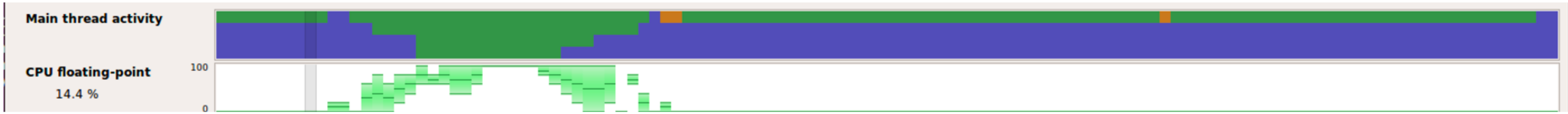
Final profile

About 3x faster

Before



After



Leak Detection

… and DDT in Offline Mode

arm

Possible memory leak

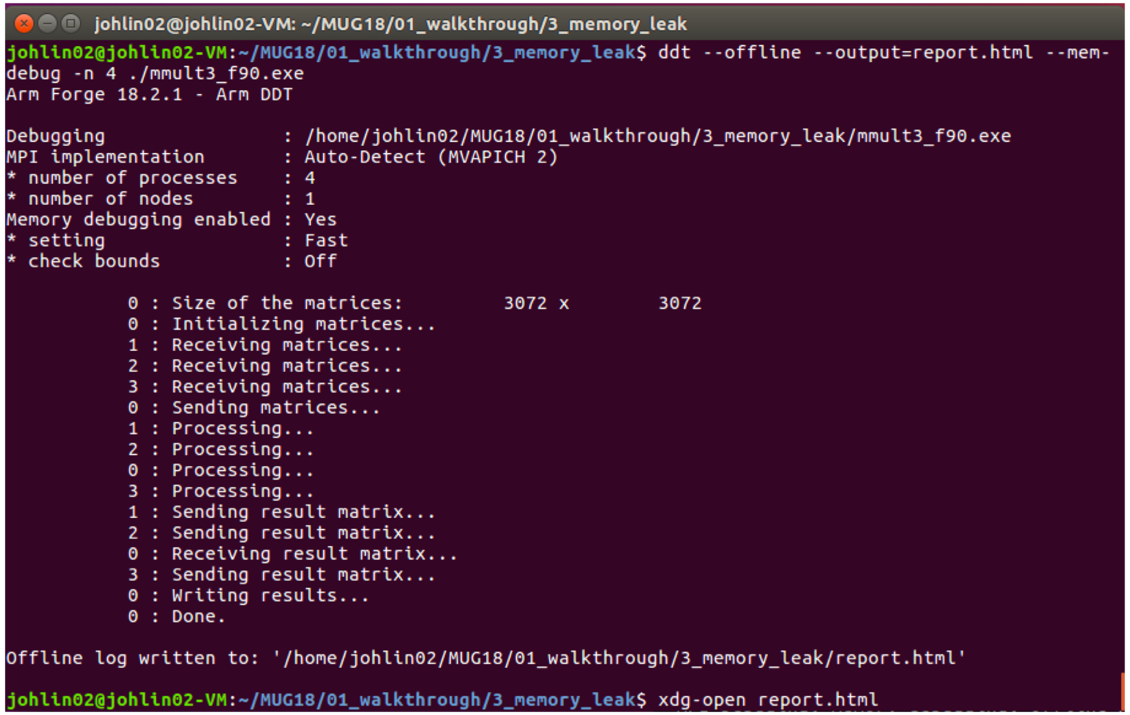
Transpose is working great, but sometimes I run out of memory?

Exercise Outline

- **Objectives**
  - Use DDT in offline mode
  - Explore DDT's report logbook
- **Commands**

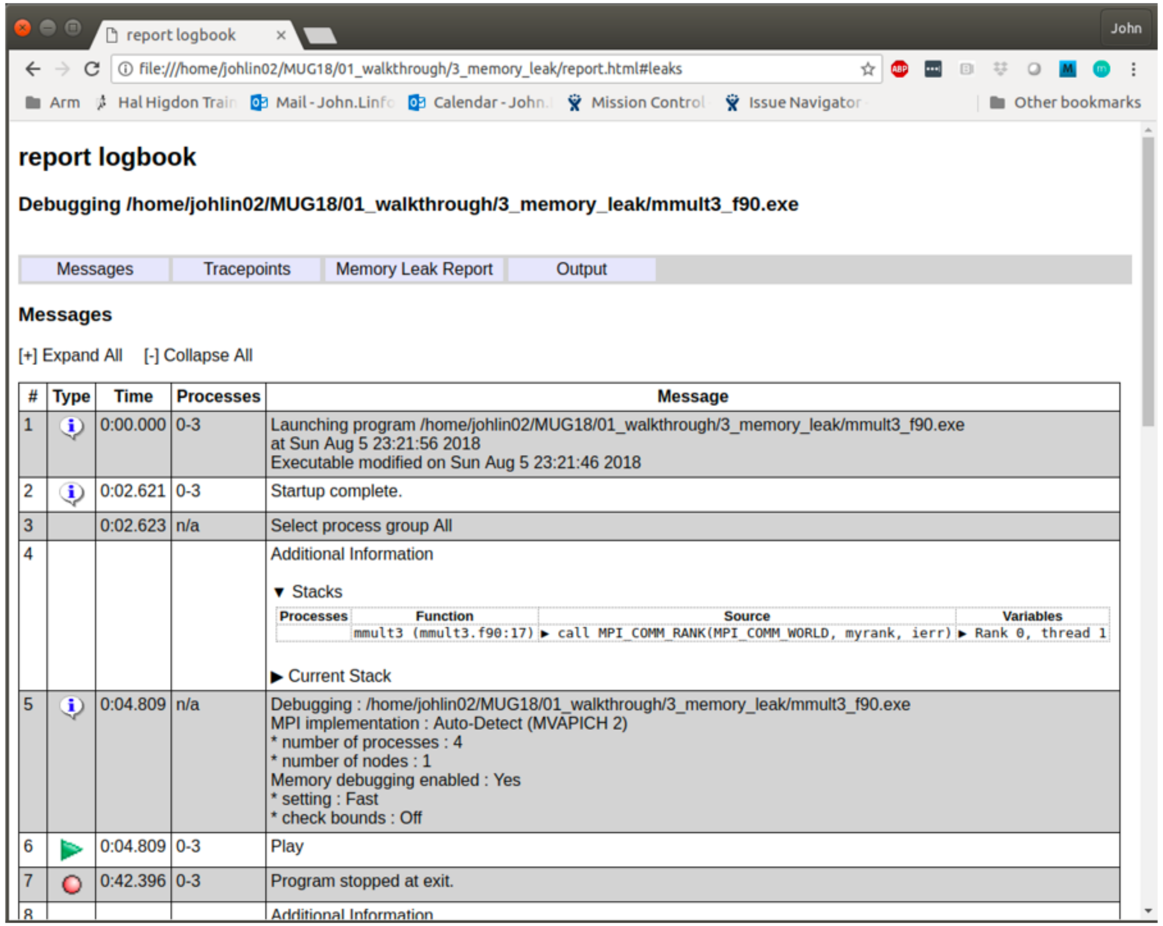
```
$ make
$ ddt --offline \
      --output=report.html \
      -n 4 \
    ./mmult3_f90.exe
$ xdg-open report.html
# Observe report
```

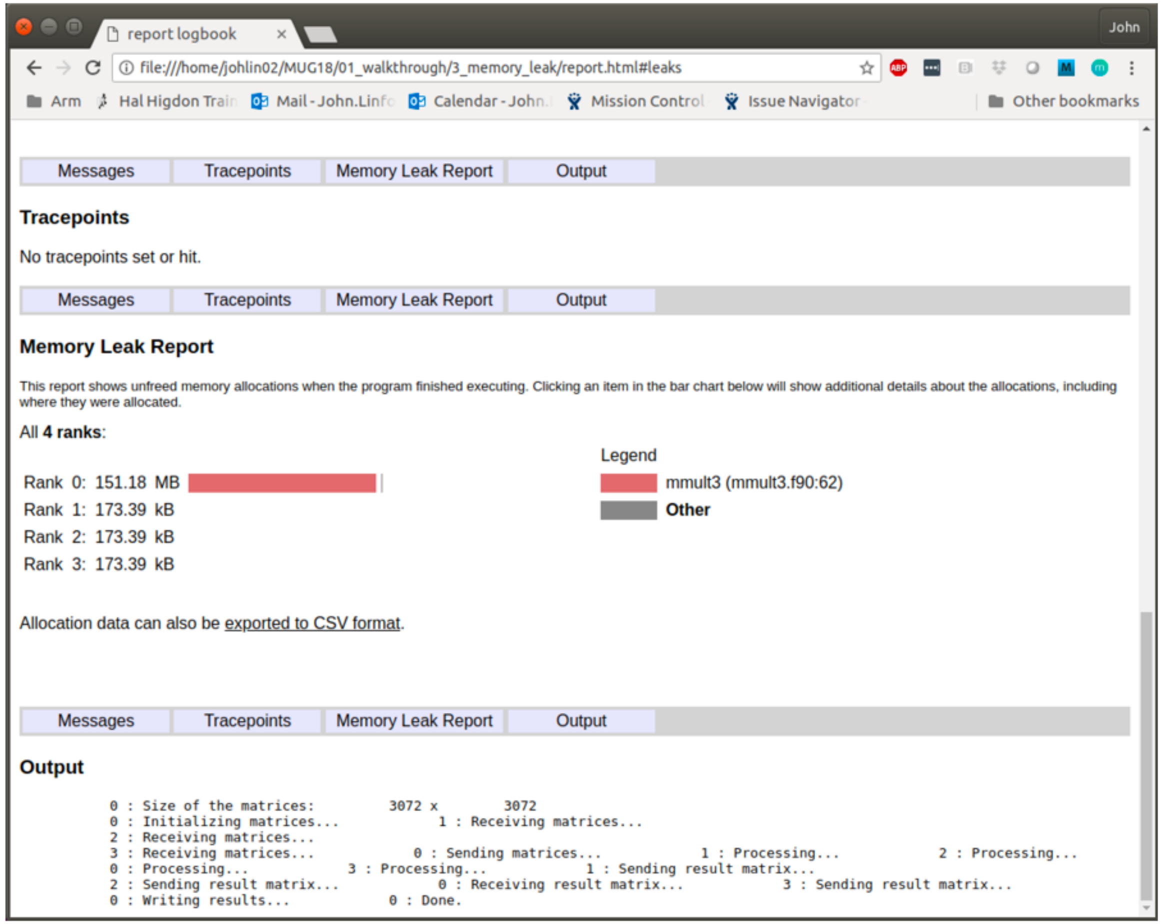
DDT in offline mode (--offline)



DDT Debugging Report

Use DDT’s reporting feature to debug long-running applications

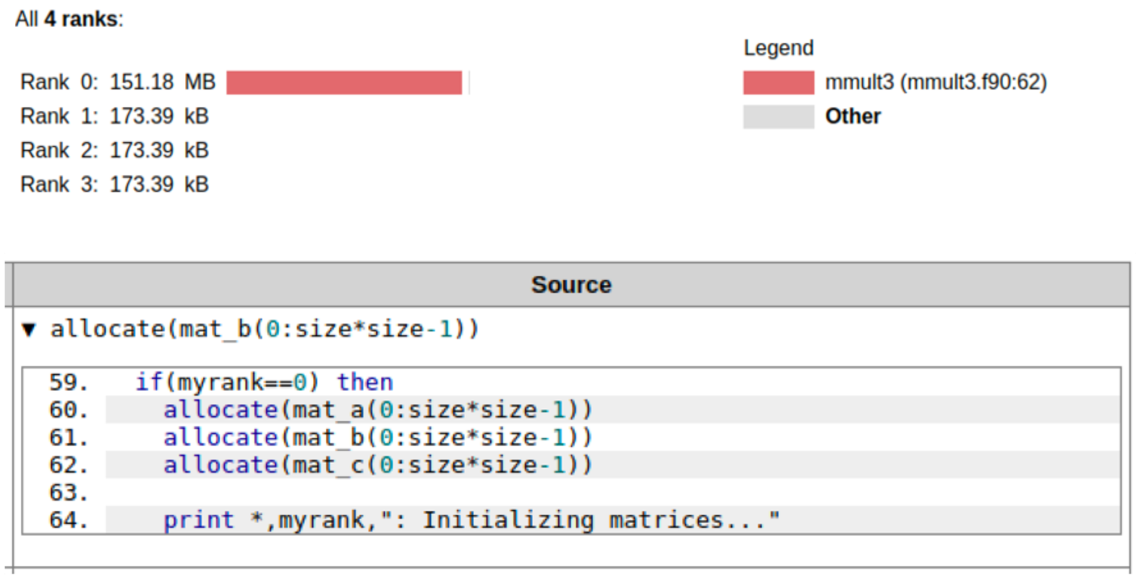




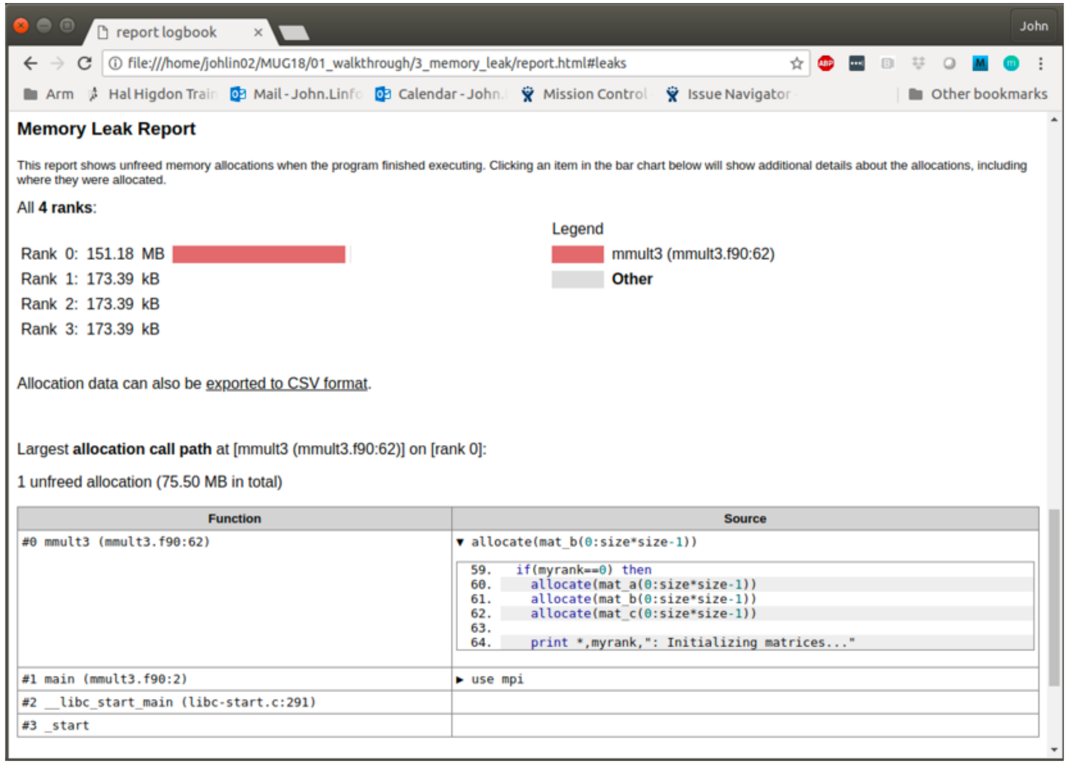
View the memory leak report to see unfreed allocations

Allocations that are not freed when the program exits *could* be leaks

Click allocation to see function source



Review source code to verify leak



arm

Memory Debugging

Allocation tracking and guard pages

Three levels of heap debugging overhead

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

arm

Tri-diagonal solve: segmentation fault

Crashing with invalid memory reference. Sounds like a job for a memory debugger!

Exercise Outline

- **Objectives**
  - Use DDT’s memory debugging features
  - Use guard pages to find out-of-bounds access
- **Commands**

```
$ make
$ ddt -n 4 ./trisol.exe
# Enable fast memory debugging
# Do not enable guard pages
```

Invalid memory access

```
johlin02@johlin02-VM: ~/MUG18/03_mem_debugging
johlin02@johlin02-VM:~/MUG18/03_mem_debugging$ make
mpif90 -c -g -O3 -o mod_trisol.o mod_trisol.f90
mpif90 -c -g -O3 -o trisol.o trisol.f90
mpif90 -c -g -O3 -o gendata.o gendata.f90
mpif90 -c -g -O3 -o solve.o solve.f90
mpif90 -c -g -O3 -o check.o check.f90
mpif90 -c -g -O3 -o matnrm.o matnrm.f90
mpif77 -c -g -O3 -o numroc.o numroc.f
mpif77 -c -g -O3 -o dlaruv.o dlaruv.f
mpif90 -g  -o trisol.exe mod_trisol.o trisol.o gendata.o solve.o check.o matnrm.o numroc.o dlaruv.o
johlin02@johlin02-VM:~/MUG18/03_mem_debugging$ mpirun -n 4 ./trisol.exe

Program received signal SIGSEGV: Segmentation fault - invalid memory reference.

Backtrace for this error:

Program received signal SIGSEGV: Segmentation fault - invalid memory reference.

Backtrace for this error:

Program received signal SIGSEGV: Segmentation fault - invalid memory reference.

Backtrace for this error:

Program received signal SIGSEGV: Segmentation fault - invalid memory reference.

Backtrace for this error:
^C[mpiexec@johlin02-VM] Sending Ctrl-C to processes as requested
[mpiexec@johlin02-VM] Press Ctrl-C again to force abort
johlin02@johlin02-VM:~/MUG18/03_mem_debugging$
```


arm

DDT's heap memory debugging framework

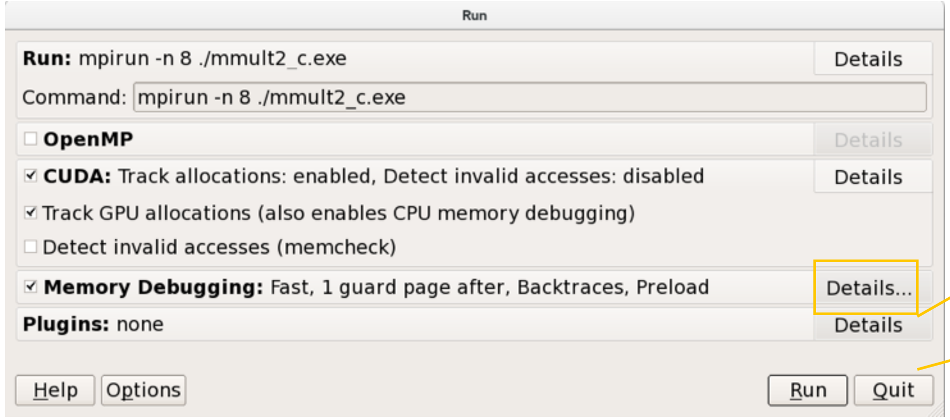
Dynamically linked binaries

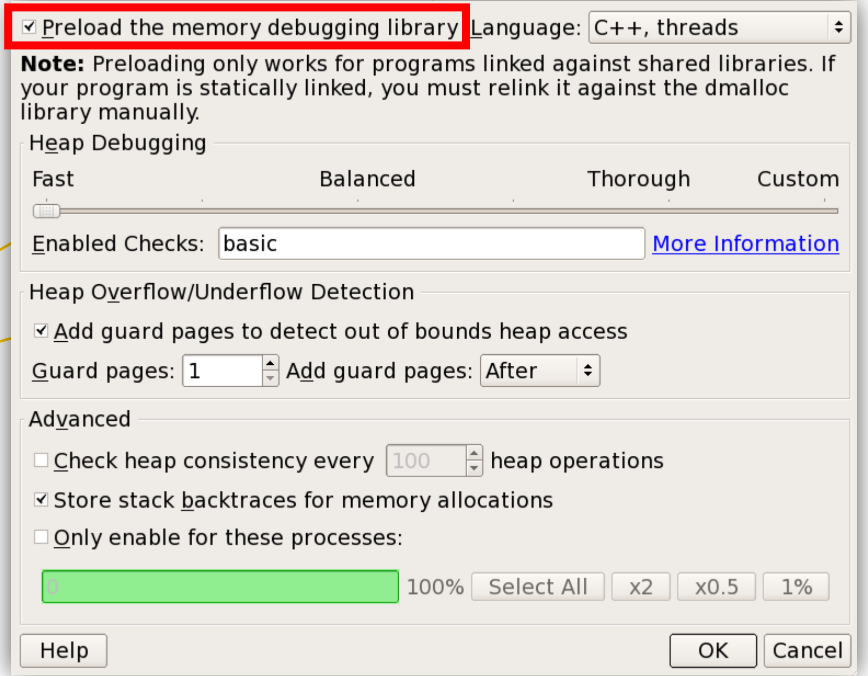
- LD_PRELOAD is usually used automatically
- Not on static binaries, not on all Crays or old SLURMs

Statically linked binaries

- If not, manual linking is required

```
LFLAGS = -dynamic -L/path/to/forge/lib/64/ -zmuldefs -Wl,--undefined=malloc -ldmalloc
```





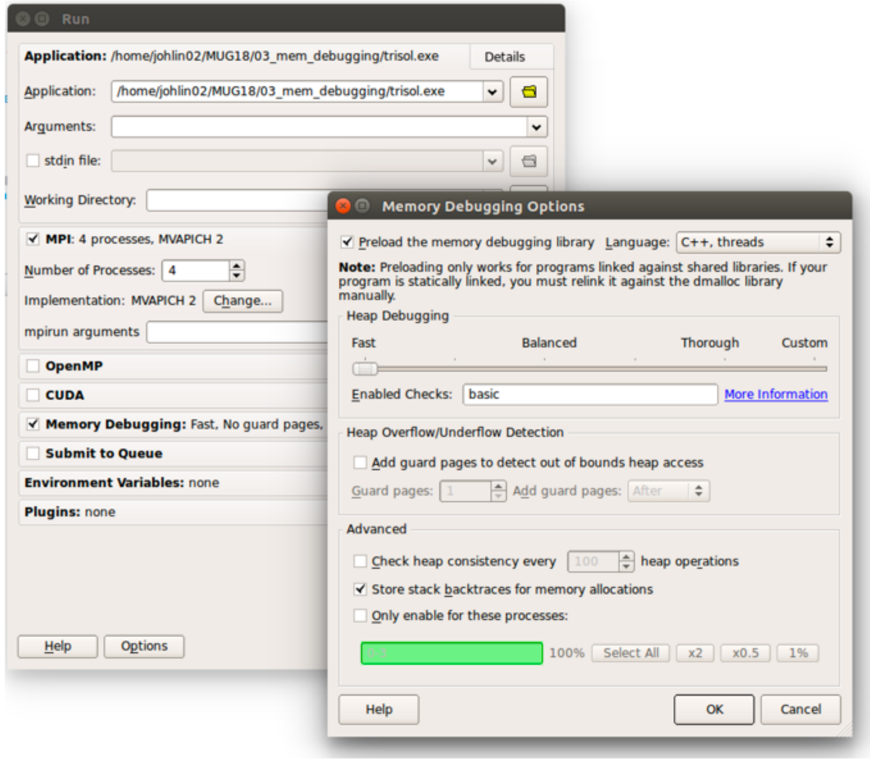
When manual linking is used, untick "Preload" box

arm

It works in DDT?????

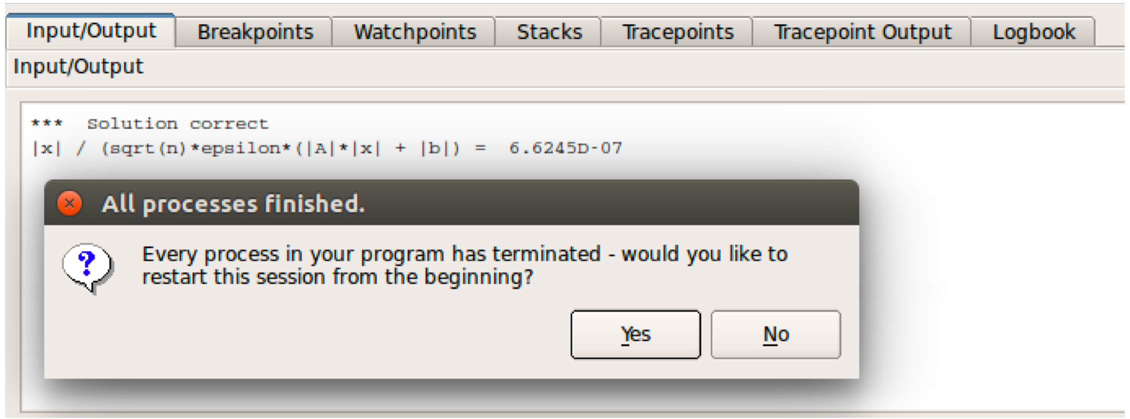
The code appears to run fine when launched from the debugger! Why?

DDT launch configuration



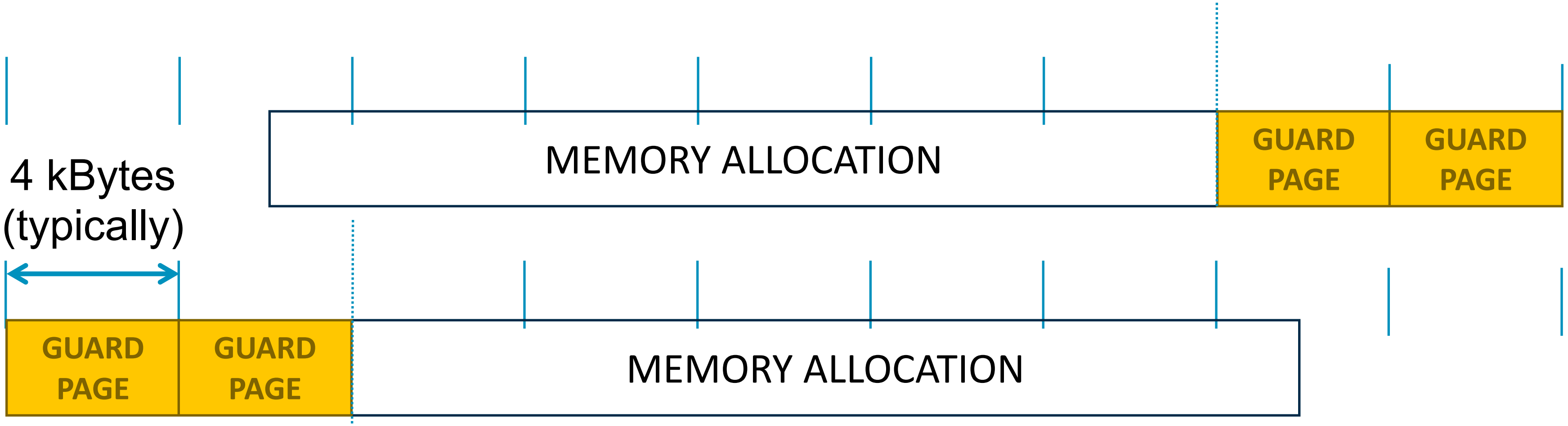
Uh oh, program output looks great

It should have crashed! What changed?



arm

Guard pages (aka “electric fences”)



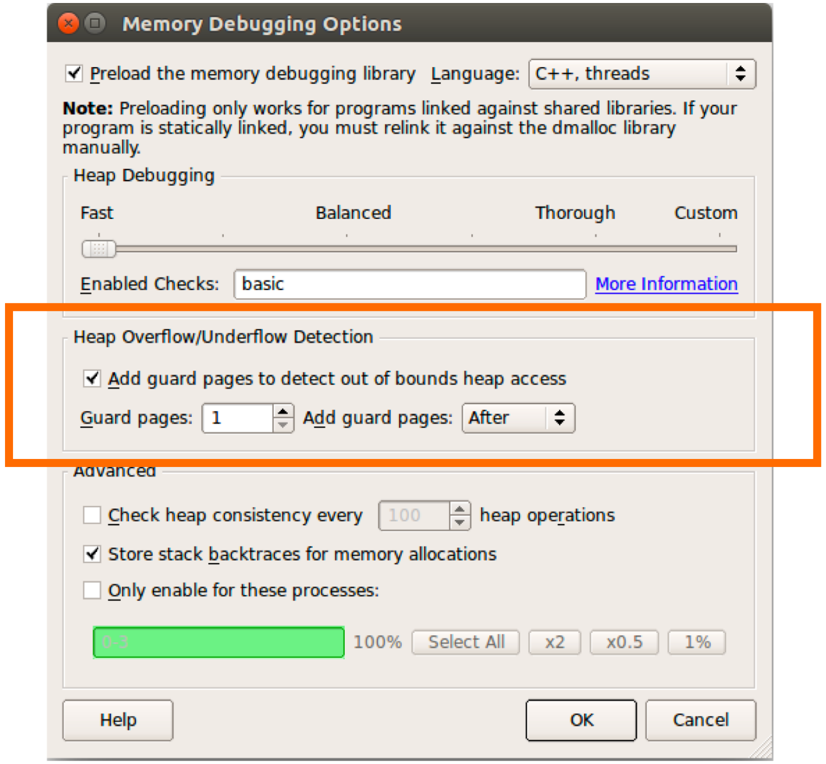
- **A powerful feature…:**
  - Forbids read/write on guard pages throughout the whole execution

    (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

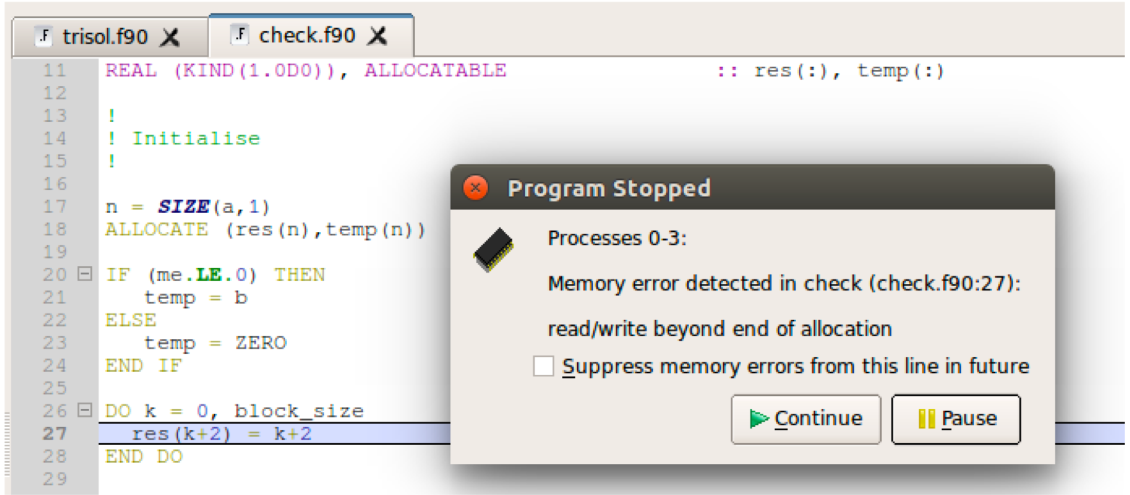
OK, this time enable guard pages

The code appears to run fine when launched from the debugger! Why?

Add one guard page after every allocation



Gotcha! Write OOB at res(k+2)



Debugging Imbalance

MPI I/O

Can we improve I/O performance?

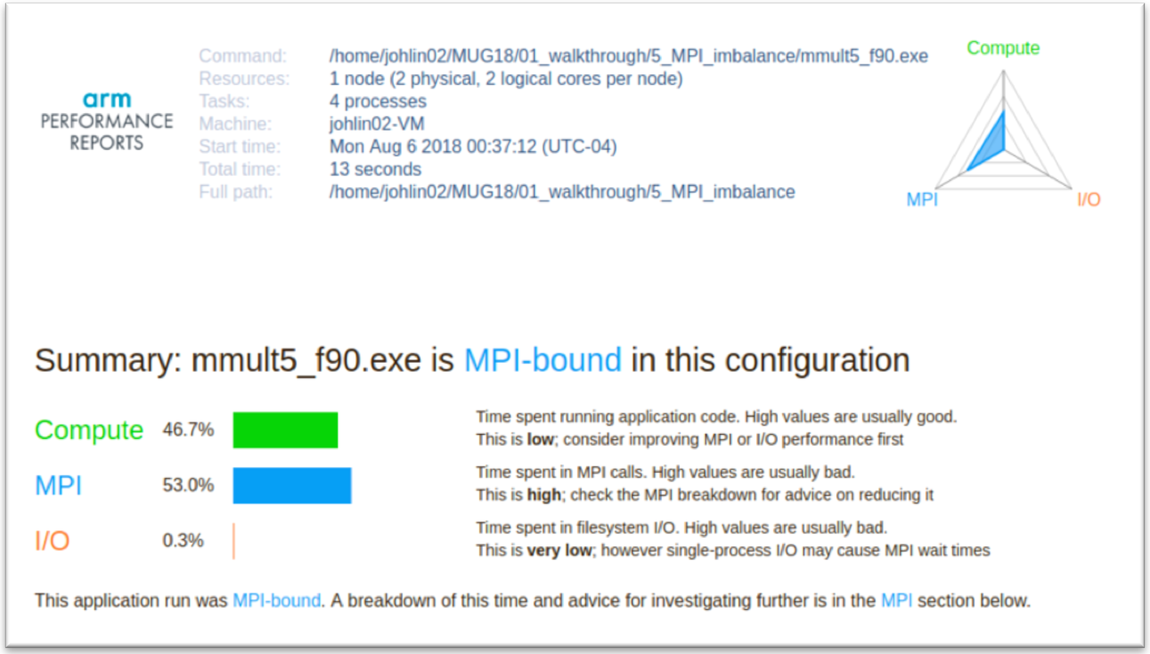
R0 responsible for all file I/O after R1+ return results. Surely we can do better?

Exercise Outline

- **Objectives**
  - Use MAP’s I/O profiling features
  - Use performance reports to quantify speedup
- **Commands**

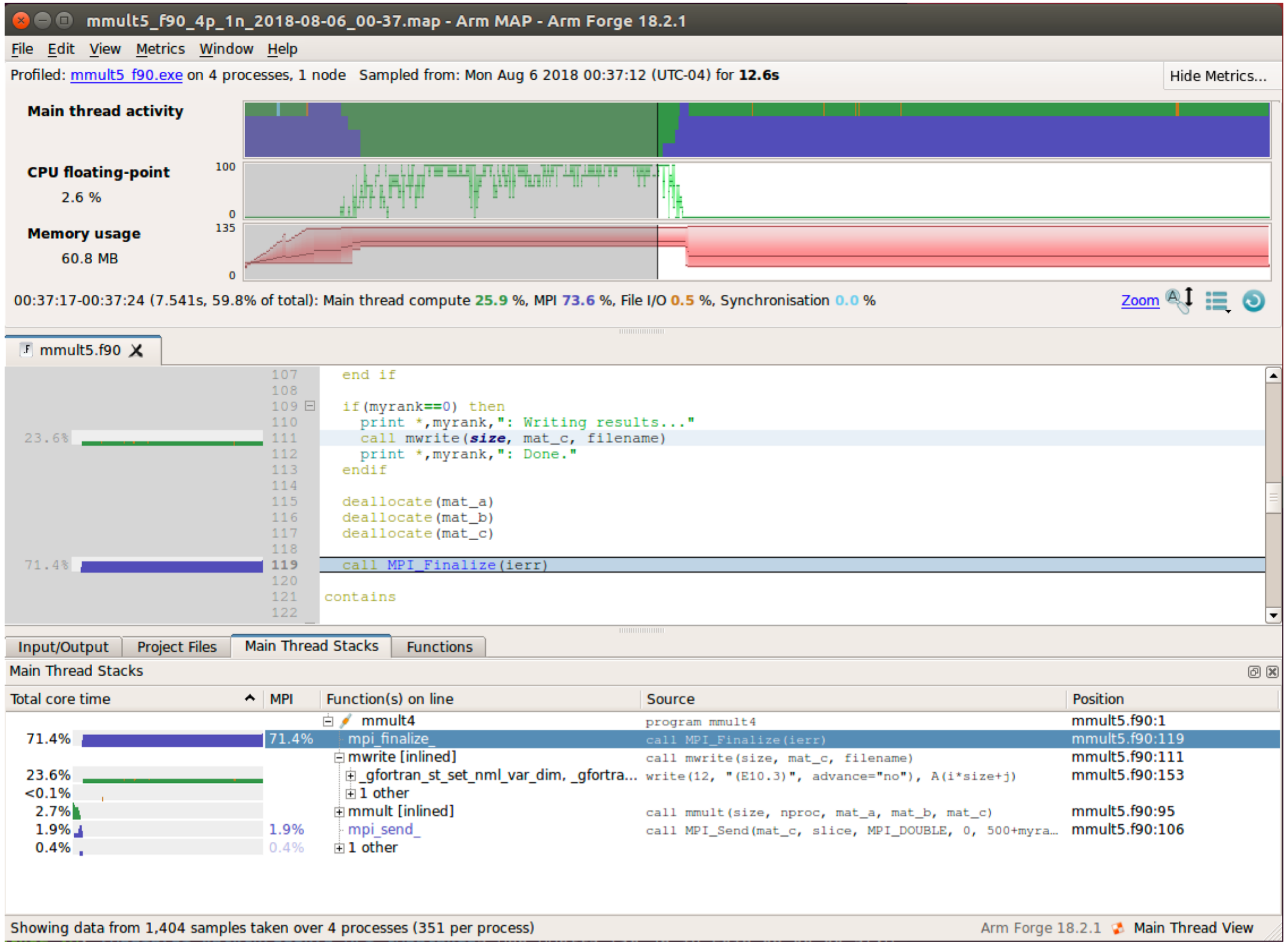
```
$ make
$ map --profile -n 4 \
    ./mmult5_f90.exe
$ perf-report mmult5_f90_4p*.map
$ xdg-open mmult5_f90_4p*.html
```

Performance report shows MPI bound



Initial profile shows MPI_Finalize dominates

Time spent in MPI_Finalize is due to load imbalance in file I/O



Answer: improve scalability of I/O routines

Use MPI-IO to let all MPI ranks write their results to file simultaneously.

Before

```
if(myrank==0) then
  do i=1,nproc-1
    call MPI_Recv(mat_c(slice*i), slice, &
                  MPI_DOUBLE, &i, 500+i, &
                  MPI_COMM_WORLD, st, ierr)
  end do
else
  call MPI_Send(mat_c, slice, MPI_DOUBLE, &
                0, 500+myrank, &
                MPI_COMM_WORLD, ierr)
end if
if(myrank==0) then
  call mwrite(size, mat_c, filename)
endif
```

After

```
call MPI_FILE_OPEN(MPI_COMM_WORLD, &
        filename, &
        MPI_MODE_CREATE+MPI_MODE_WRONLY, &
        MPI_INFO_NULL, fh, ierr)
call MPI_FILE_SET_VIEW(fh, &
        0_MPI_OFFSET_KIND, MPI_DOUBLE, &
        MPI_DOUBLE, 'native', &
        MPI_INFO_NULL, ierr)
call MPI_FILE_WRITE_AT(fh, disp, mat_c, &
        slice, MPI_DOUBLE, st, ierr)
call MPI_BARRIER(MPI_COMM_WORLD, ierr)
call MPI_FILE_CLOSE(fh, ierr)
```

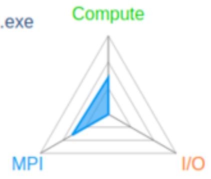

New approach: use MPI-IO for file output

Each MPI rank writes its results to it's own part of the output file

Before: runtime 13 seconds

arm PERFORMANCE REPORTS

Command:	/home/johlin02/MUG18/01_walkthrough/5_MPI_imbalance/mmult5_f90.exe
Resources:	1 node (2 physical, 2 logical cores per node)
Tasks:	4 processes
Machine:	johlin02-VM
Start time:	Mon Aug 6 2018 00:37:12 (UTC-04)
Total time:	13 seconds
Full path:	/home/johlin02/MUG18/01_walkthrough/5_MPI_imbalance



Summary: mmult5_f90.exe is MPI-bound in this configuration

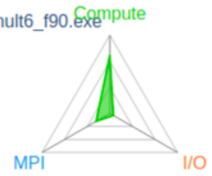
Compute	46.7%	Time spent running application code. High values are usually good. This is **low**; consider improving MPI or I/O performance first
MPI	53.0%	Time spent in MPI calls. High values are usually bad. This is **high**; check the MPI breakdown for advice on reducing it
I/O	0.3%	Time spent in filesystem I/O. High values are usually bad. This is **very low**; however single-process I/O may cause MPI wait times

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

After: runtime 5 seconds (2.6x speedup)

arm PERFORMANCE REPORTS

Command:	/home/johlin02/MUG18/01_walkthrough/5_MPI_imbalance/solution/mmult6_f90.exe
Resources:	1 node (2 physical, 2 logical cores per node)
Tasks:	4 processes
Machine:	johlin02-VM
Start time:	Mon Aug 6 2018 00:34:17 (UTC-04)
Total time:	5 seconds
Full path:	/home/johlin02/MUG18/01_walkthrough/5_MPI_imbalance/ solution



Summary: mmult6_f90.exe is Compute-bound in this configuration

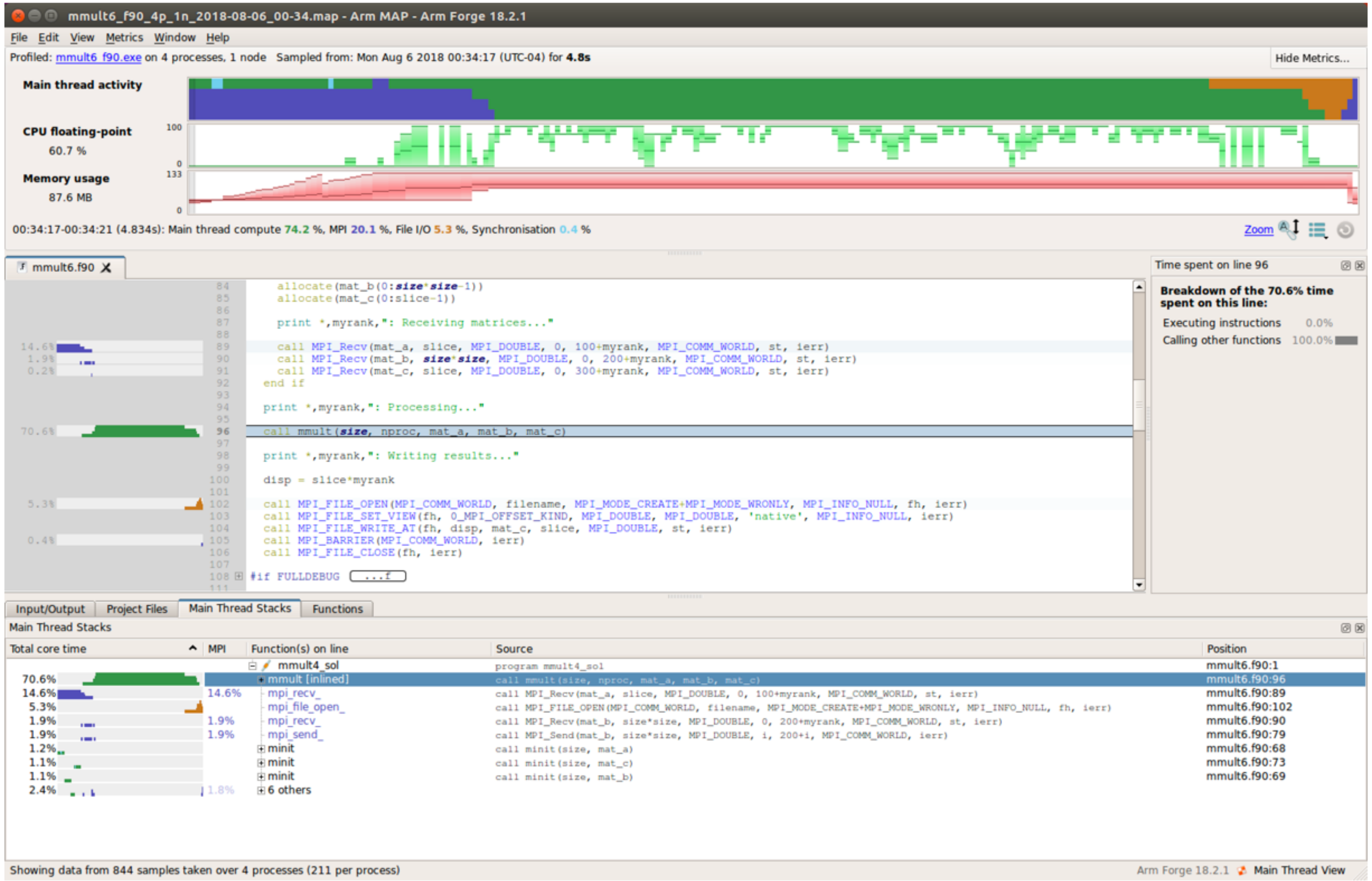
Compute	74.5%	Time spent running application code. High values are usually good. This is **high**; check the CPU performance section for advice
MPI	20.1%	Time spent in MPI calls. High values are usually bad. This is **low**; this code may benefit from a higher process count
I/O	5.3%	Time spent in filesystem 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.

arm

Final profile shows balanced I/O and compute dominates

New approach is about 3x faster



arm

Success at Scale

Curtin Quantum Collisions

arm

CCC and the ORNL GPU Hackathon @ Pawsey

Quantum collisions in atomic and molecular physics

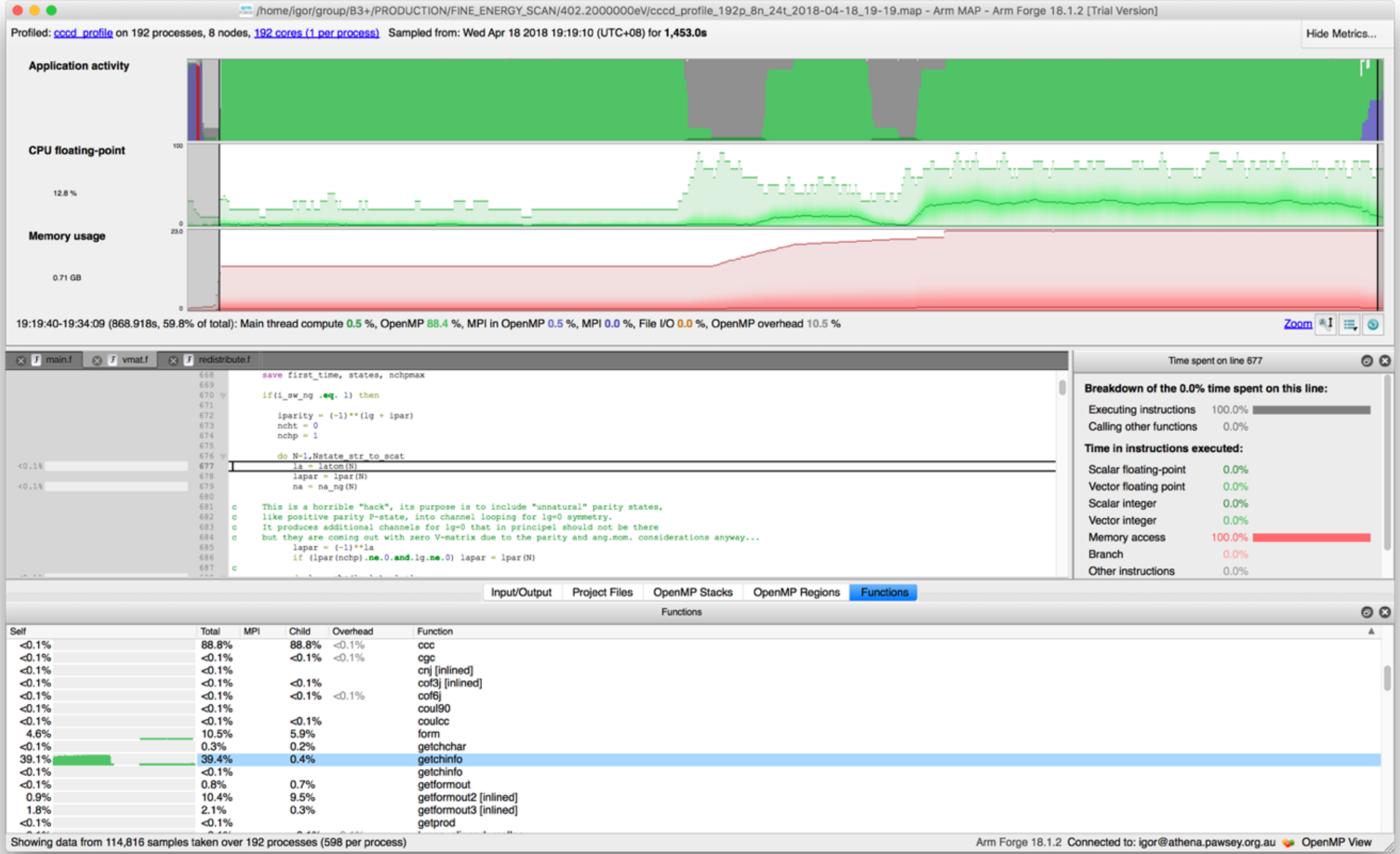
- CCC: Quantum mechanics
  - Fusion energy
  - Laser science
  - Lighting industry
  - Medical imaging / therapy
  - Astrophysics

- Igor Bray, Head of Physics and Astronomy, and the Theoretical Physics Group, in the Faculty of Science and Engineering, at Curtin University

Curtin University

arm

Initial profile at production scale



arm

Load balancer is imbalanced?

Customized load balancing algorithm wasn't delivering expected results

```
   0    8    0 -10     199     329     492    1.21   13530        0      89   -1   91%
LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
   1    8    0  -7     591     573     872    1.97   45150        0     350    0   80%
LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
   2    8    0 -16     894     762    1153    2.28   77028        0     607    1   86%
LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
   3    8    0 -24     916     886    1331    2.05   99681        0     766    2   91%
LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
```

“That makes no sense!”

Computing one grid point takes as much time as computing the entire grid

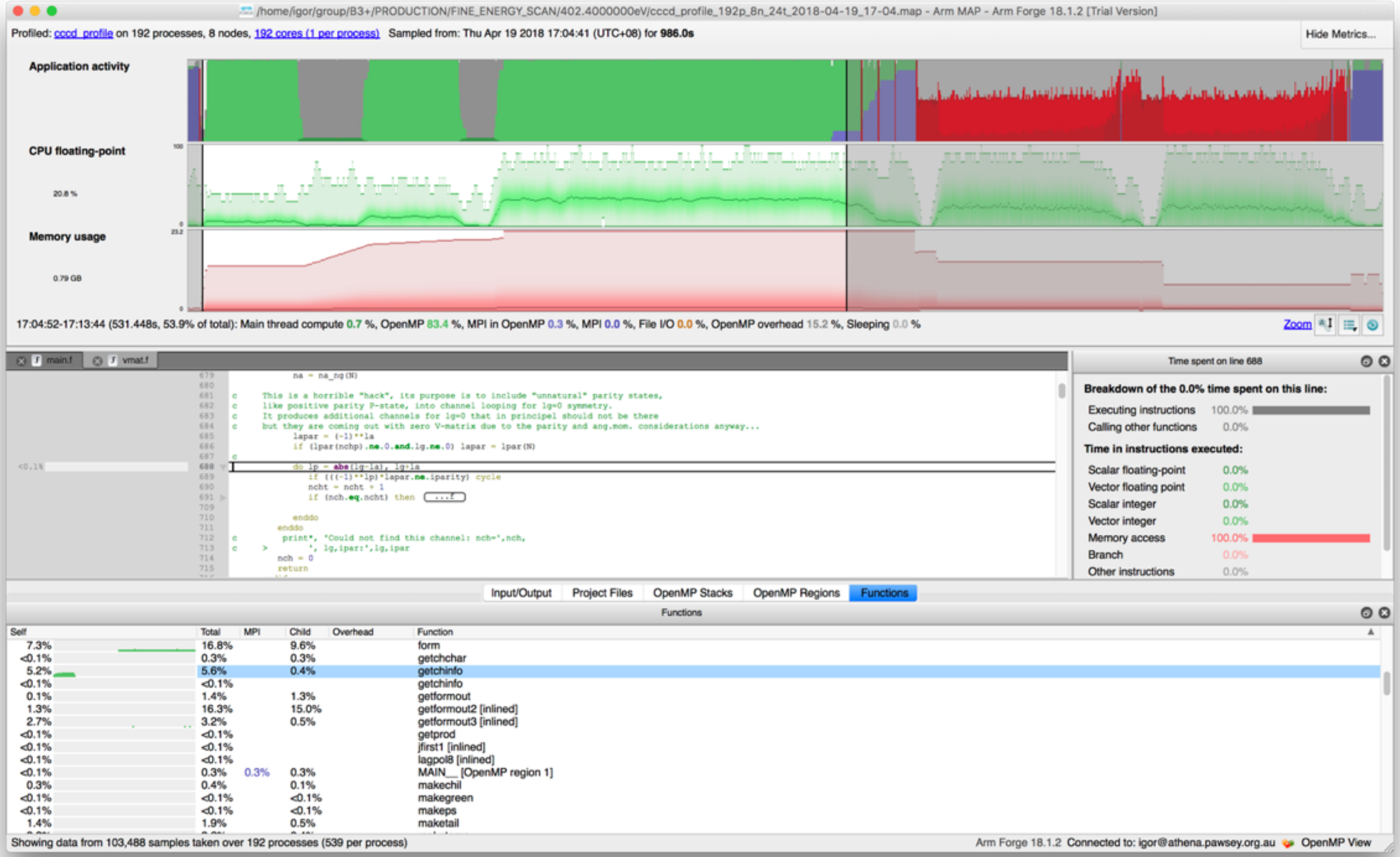
Self	Total	MPI	Child	Overhead	Function
<0.1%	88.8%		88.8%	<0.1%	ccc
<0.1%	<0.1%		<0.1%	<0.1%	cgc
<0.1%	<0.1%				cnj [inlined]
<0.1%	<0.1%		<0.1%		cof3j [inlined]
<0.1%	<0.1%		<0.1%	<0.1%	cof6j
<0.1%	<0.1%				coul90
<0.1%	<0.1%		<0.1%		coulcc
4.6%	10.5%		5.9%		form
<0.1%	0.3%		0.2%		getchchar
39.1%	39.4%		0.4%		getchinfo
<0.1%	<0.1%				getchinfo
<0.1%	0.8%		0.7%		getformout
0.9%	10.4%		9.5%		getformout2 [inlined]
1.8%	2.1%		0.3%		getformout3 [inlined]
<0.1%	<0.1%				getprod

Showing data from 114,816 samples taken over 192 processes (598 per process)

Surprise! Didn’t expect that.

Final profile, again at production scale

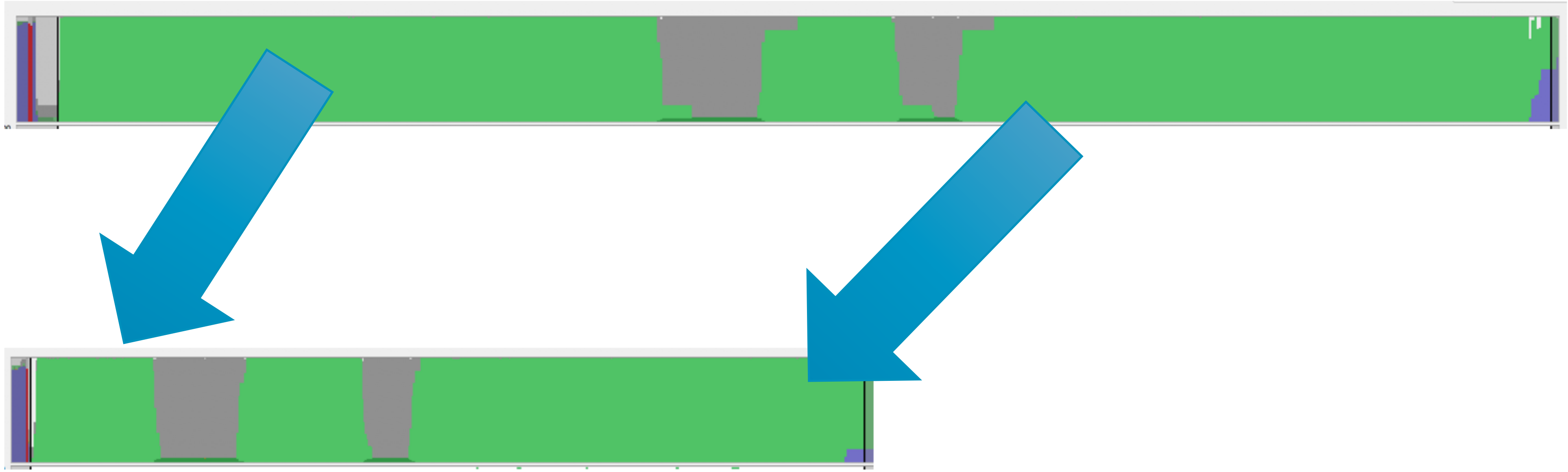
Found an unbounded array copy `a(:)` that should have been `a(1:N)`



arm

Before and after

4.6%	10.5%	5.9%	form
<0.1%	0.3%	0.2%	getchchar
39.1%	39.4%	0.4%	getchinfo
<0.1%	<0.1%		getchinfo
<0.1%	0.8%	0.7%	getformout



Self	Total	MPI	Child	Overhead	Function
7.3%	16.8%		9.6%		form
<0.1%	0.3%		0.3%		getchchar
5.2%	5.6%		0.4%		getchinfo
<0.1%	<0.1%				getchinfo
0.1%	1.4%		1.3%		getformout

Balanced the load balancer

Load can be balanced mow that work blocks are of expected sizes

Before:

```
0    8    0 -10    199    329     492    1.21   13530        0      89   -1   91% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
1    8    0  -7    591    573     872    1.97   45150        0     350    0   80% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
2    8    0 -16    894    762    1153    2.28   77028        0     607    1   86% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
3    8    0 -24    916    886    1331    2.05   99681        0     766    2   91% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
```

After:

```
0    8    0 -10    174    329     492    1.06   13530        0      85   -1   93% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
1    8    0 -11    415    577     872    1.40   43956        0     340    0   97% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
2    8    0 -11    616    757    1153    1.55   79003        0     592    1   97% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
3    8    0 -12    667    874    1331    1.46  105111        0     734    2   96% LG,node,ipar,inc,vt,i1,i2,tperi,nch,naps,mt,prev LG,eff
```

Custom metrics for Lustre

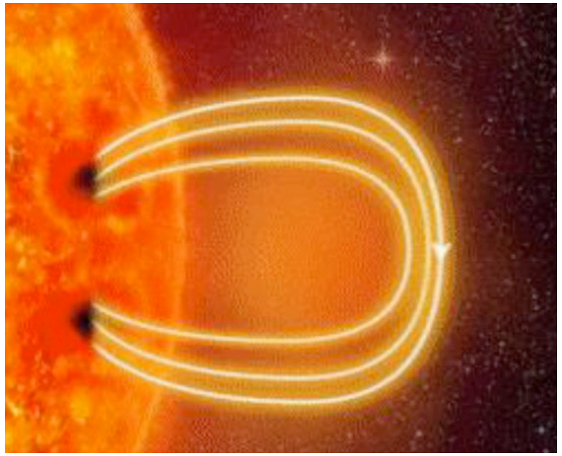
Combine I/O performance data from system and application

arm

Advanced I/O investigation of Lustre on Archer

Simultaneously view system-level and application-level performance.

- Show data from Lustre client logs along with application data
- iPIC3D: kinetic simulation of plasma
  - Fully 3D implicit particle-in-cell (PIC)
  - C++ and MPI
  - Intermediate simulation results saved in VTK binary files, single file per quantity
  - Checkpointing done through HDF5 to individual files per process
  - Field values saved using collective MPI-IO to single file



Available performance data

Use MAP's ability to measure filesystem performance at the system and application levels

System level performance data

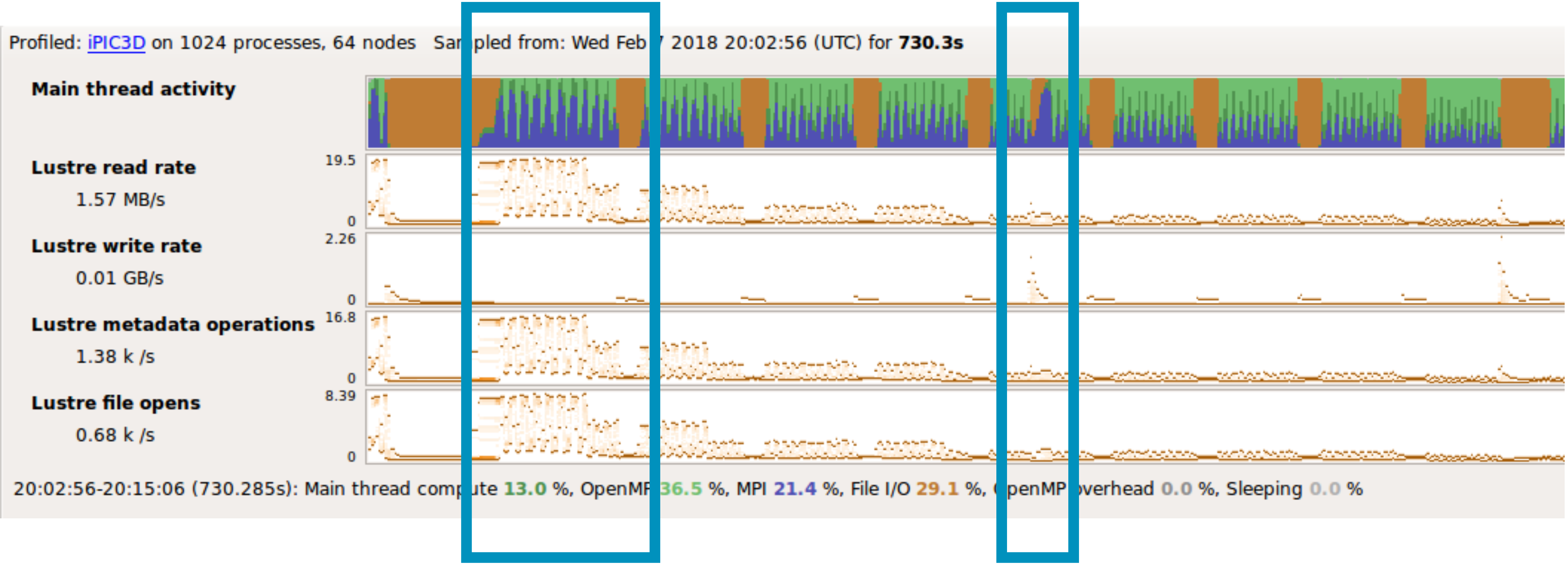
- Lustre logs: each read, write, or metadata operation recorded from each Lustre client.
- Aggregate I/O data for precise bandwidth figures for read/write at any moment in time.
- Max/min/mean bandwidth.
- Scheduler logs: application run start and end time and assigned nodes.

Application level performance data

- Approximate I/O bandwidth in a timeline.
- Approximate classification of I/O instructions (methods).
- In block-synchronous approach, it is possible to identify different I/O phases.

MAP aligns the system timeline with the application timeline

Lustre data is read from the lustre client's log files, while application data is read directly.

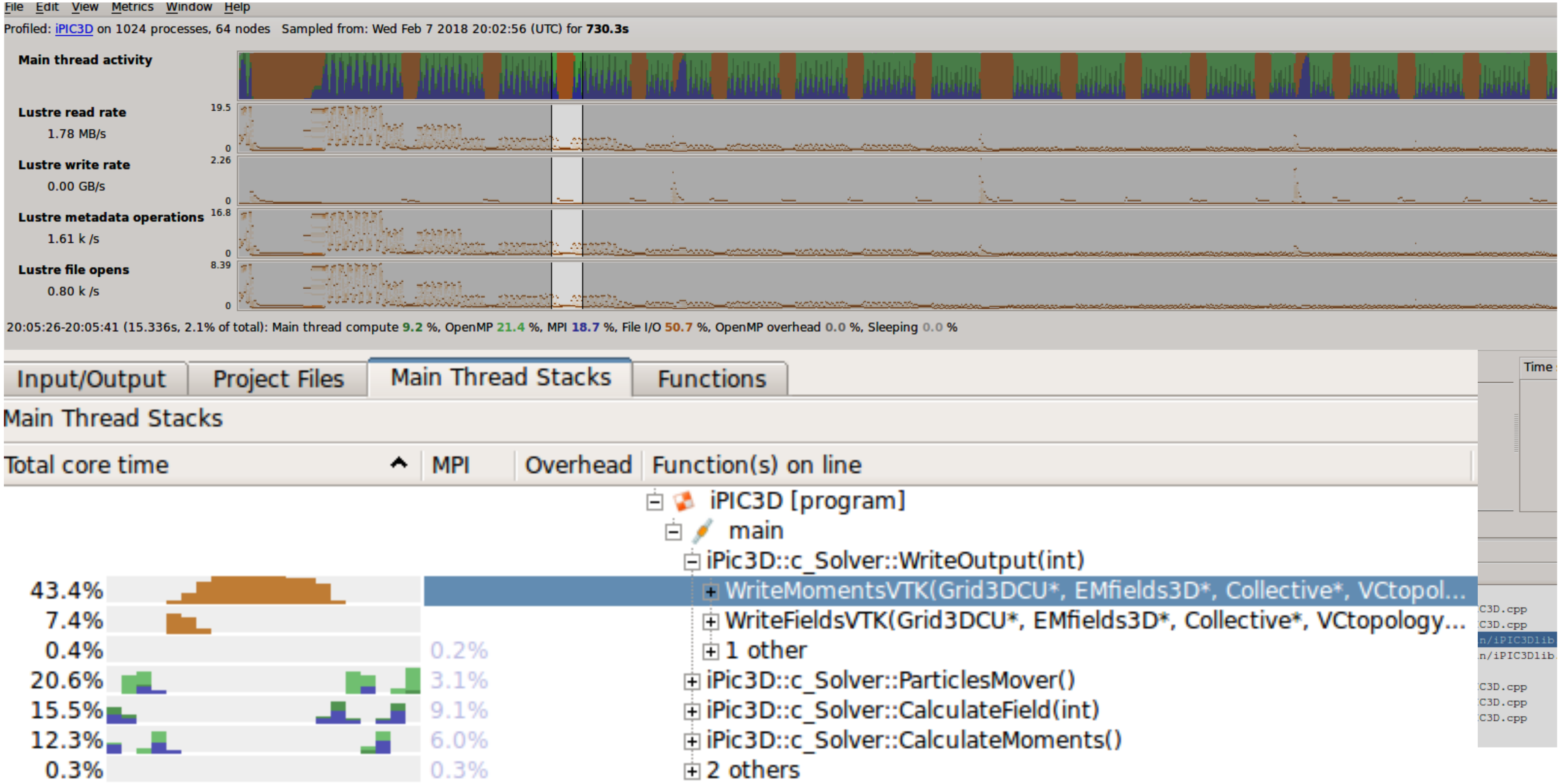


N-N file read shows spike in file open/read operations.

Checkpoint I/O corresponds to spike in Lustre write rate

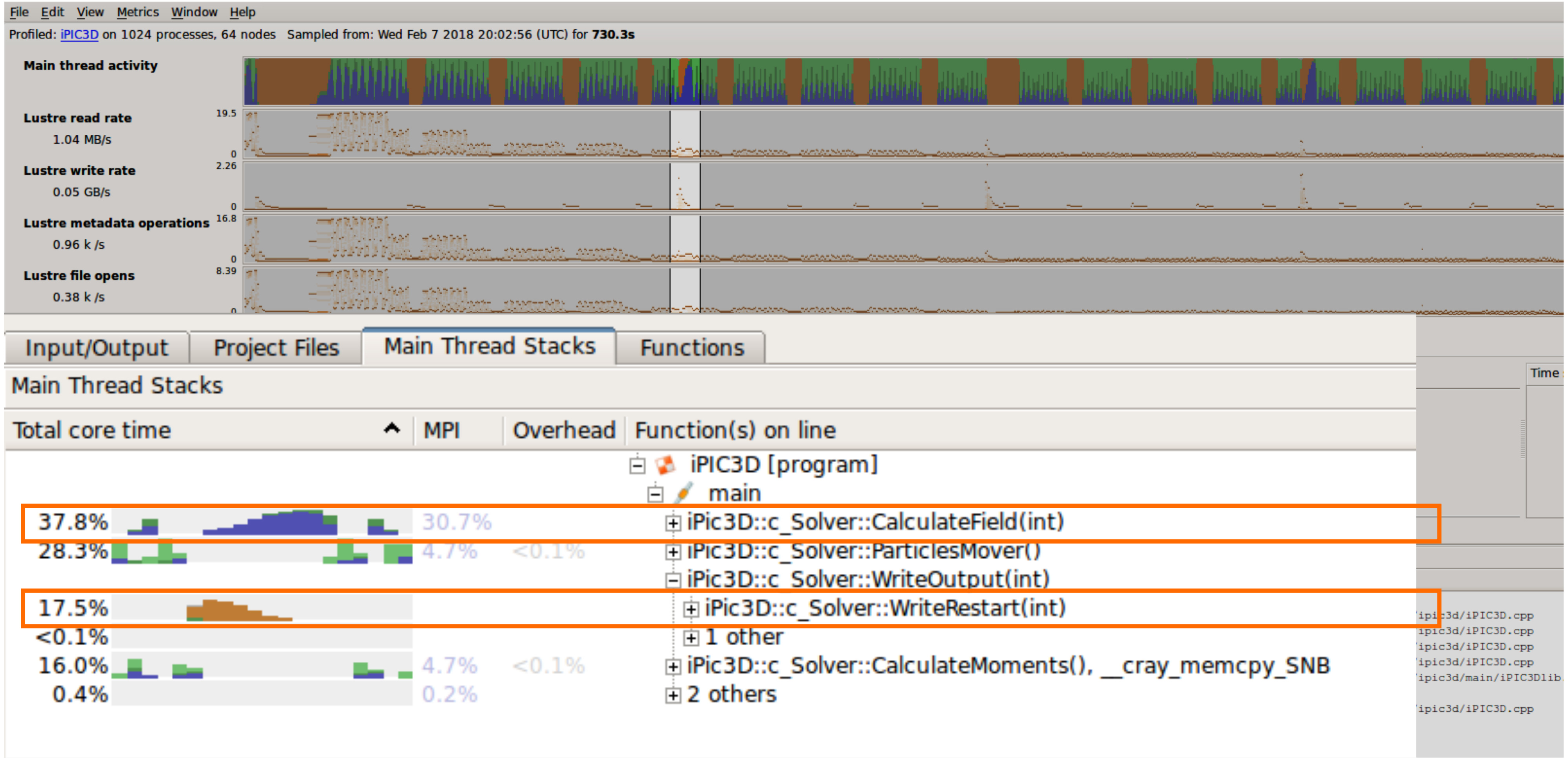
We can focus on each I/O operation individually

Select a portion of the application timeline to view the source code performing I/O.



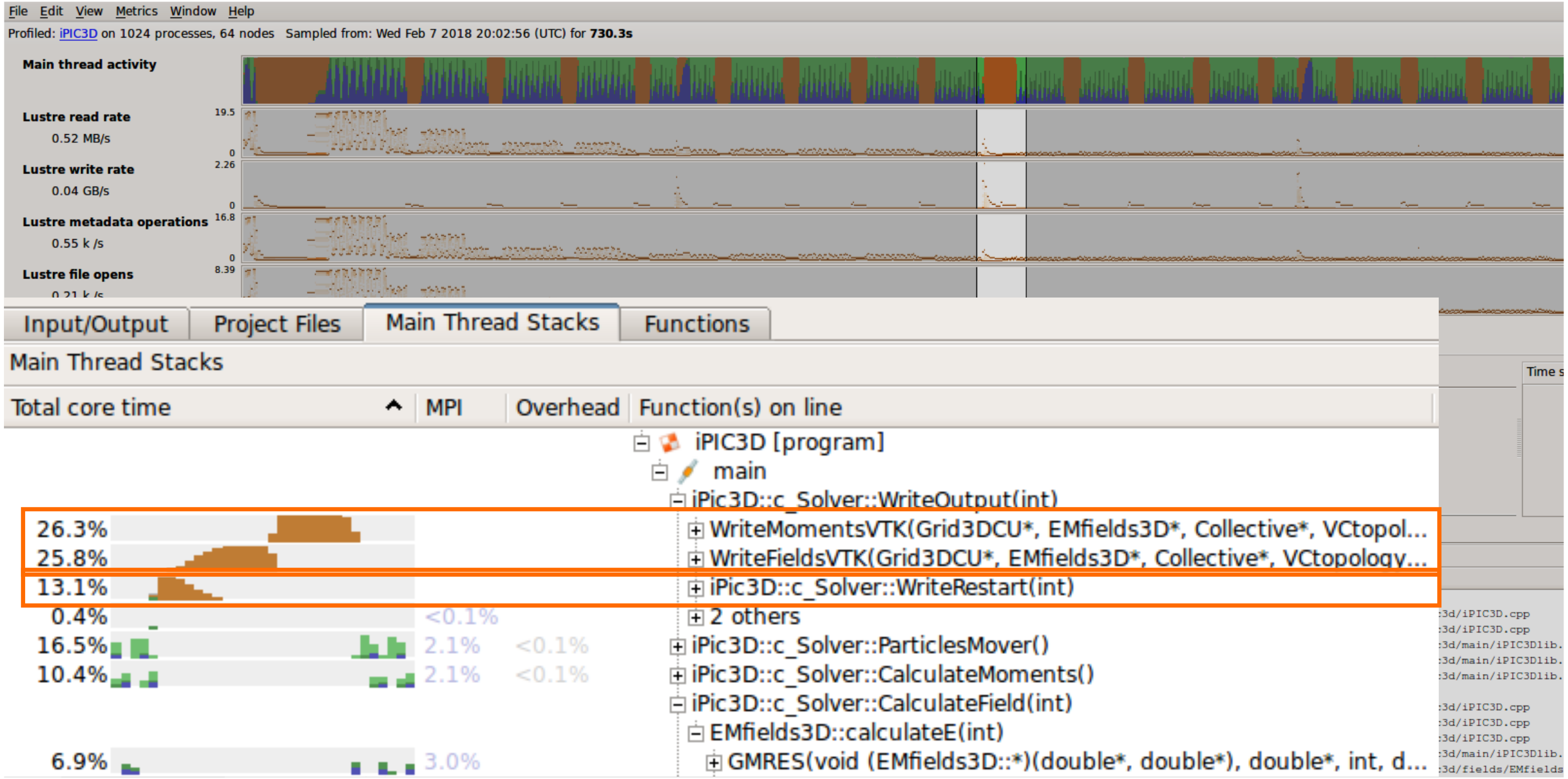
MAP's timeline shows I/O overlapping with communication

We see elevated Lustre write rate when writing checkpoint restart files in HDF5.



It's possible to overlap different I/O approaches

HDF5 and VTK I/O operations occur at the same time on different ranks.

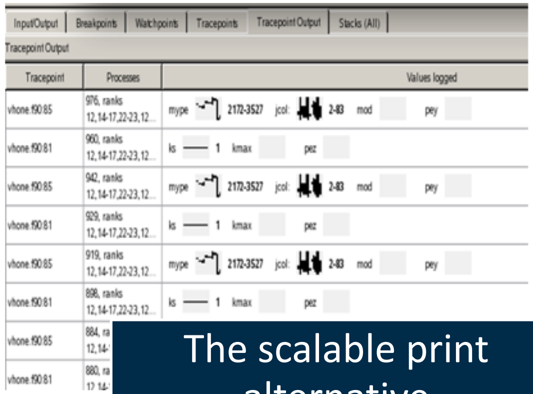


arm

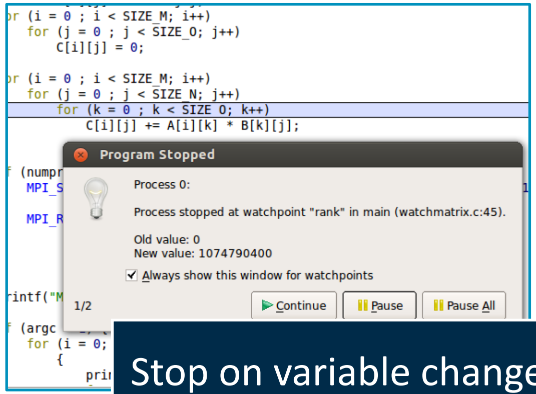
Wrap Up

arm

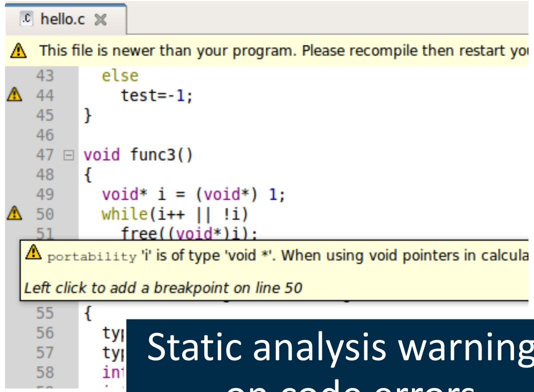
Five great things to try with Arm DDT



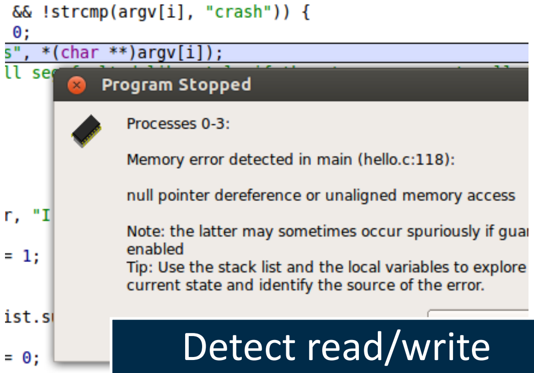
The scalable print alternative



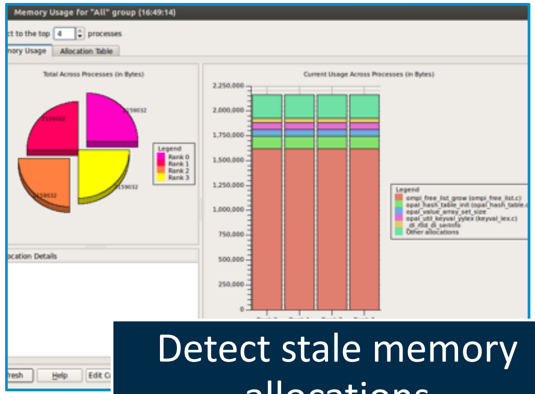
Stop on variable change



Static analysis warnings on code errors



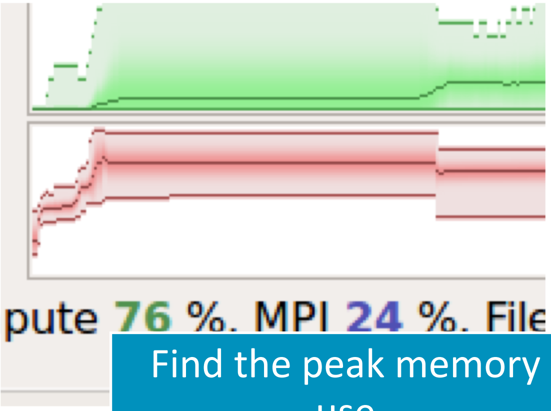
Detect read/write beyond array bounds



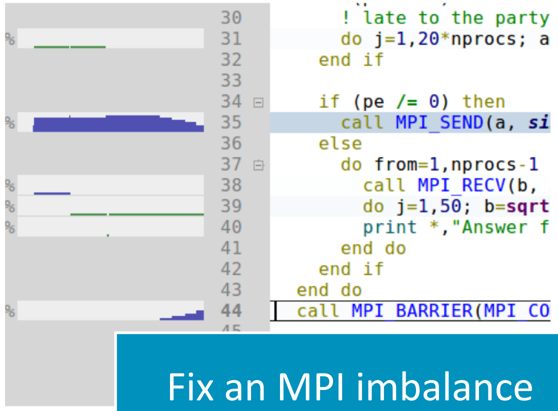
Detect stale memory allocations

arm

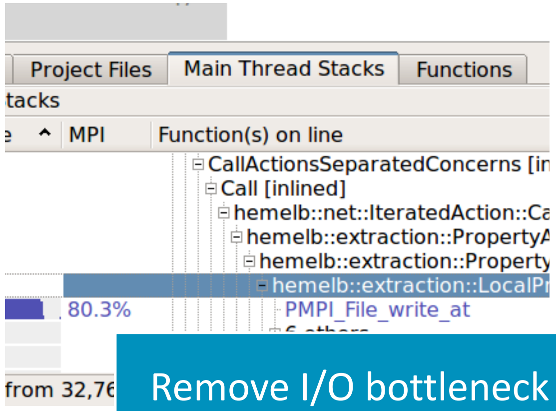
Six Great Things to Try with Arm MAP



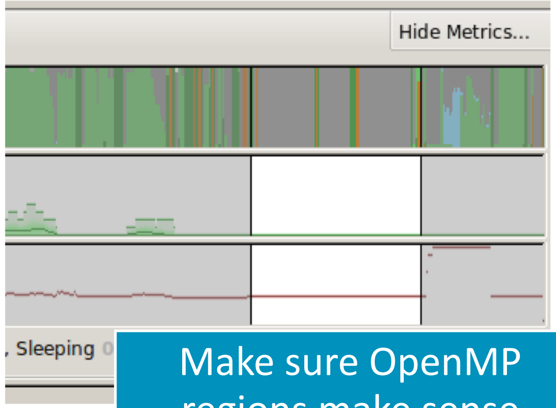
Find the peak memory use



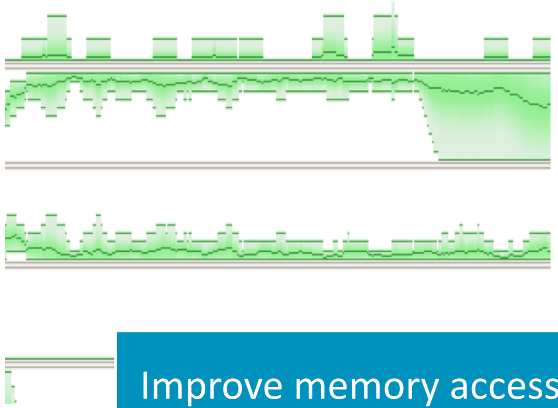
Fix an MPI imbalance



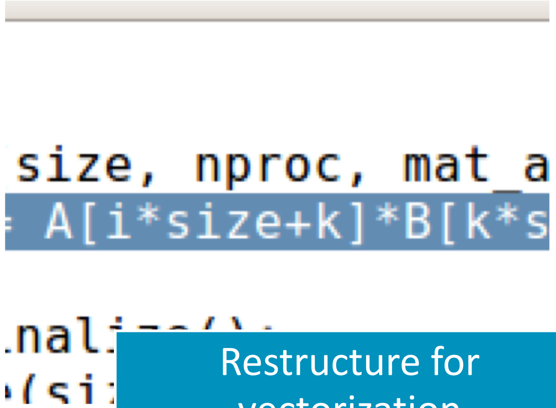
Remove I/O bottleneck



Make sure OpenMP regions make sense



Improve memory access

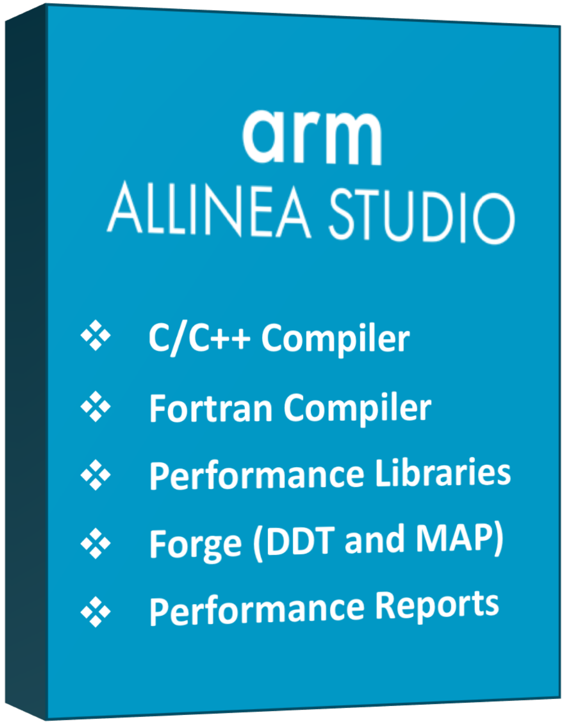


Restructure for vectorization

arm

Wrap Up

Visit arm.com/hpc to learn more about Arm Forge and download a free trial.



- Tools are a must-have when programming HPC systems
- Use a structured, profile-driven optimization methodology
- Arm DDT can help improve code correctness
- Arm MAP can help improve code performance
- Arm Forge = DDT + MAP is a great choice at scale

Download at arm.com/hpc

Thank You

Danke

Merci

谢谢

ありがとう

Gracias

Kiitos

감사합니다

धन्यवाद

תודה

arm