linaroforge

Debugging and Optimising Parallel Codes with Linaro Forge

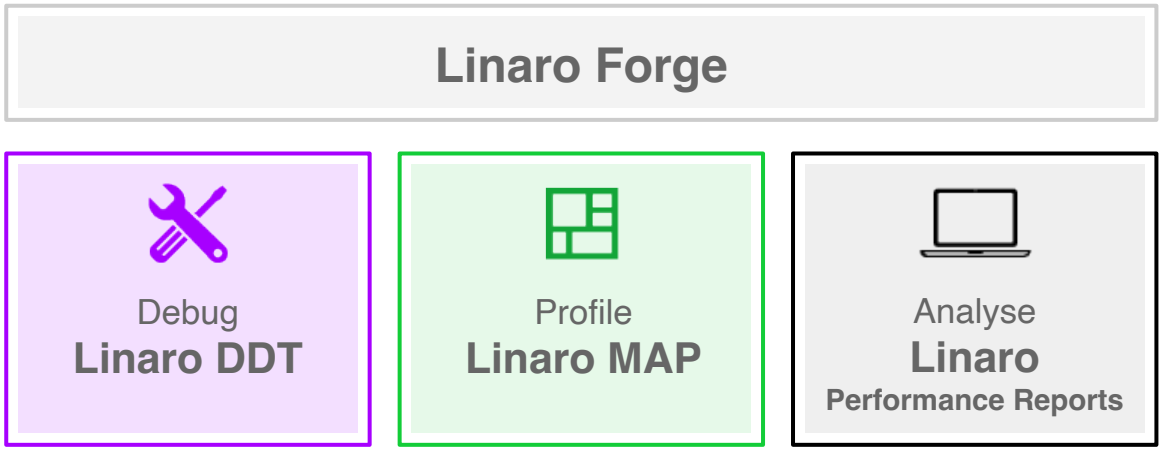
Rudy Shand
Principal FAE

Agenda

10:00 - 11:00 Lecture on Debugging with DDT
11:00 - 12:30 DDT Debugging Hands-on session
12:30 - 13:00 Break
13:00 - 14:00 Lecture on Profiling with MAP
14:00 - 15:30 MAP Profiling Hands-on session
15:30 - 16:00 Break
16:00 - 17:00 Try DDT / MAP with own codes

HPC Development Solutions from Linaro

Best in class commercially supported tools for Linux and high-performance computing (HPC)



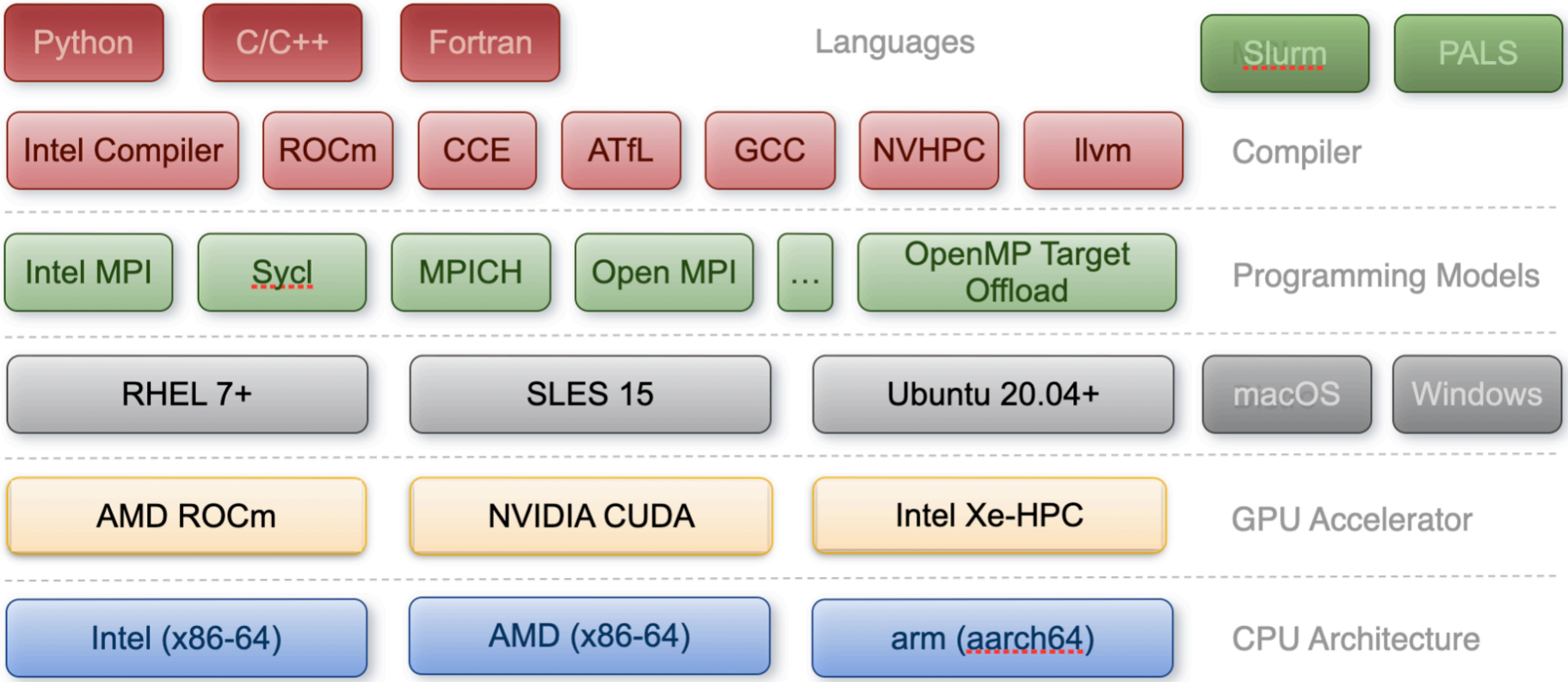
Performance Engineering for any architecture, at any scale

Bug classification

- Crashes
  - One or more processes in application terminates
  - Most common and generally easiest to solve
- Hangs
  - Deadlocks - Stuck waiting for something that never happens
  - Livelocks - Making local progress, but no global progress
- Race conditions
  - One or more threads accessing the same data at the same time in non deterministic way
  - Shows up as incorrect answer or sometimes crashes

linaroforge

Supported Platforms

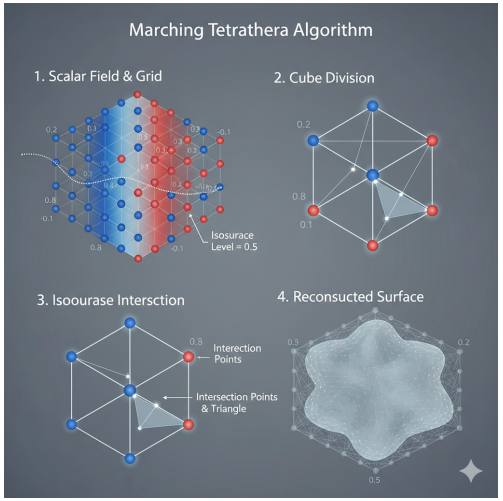


linaroforge

Case Study: Vienne Supercomputing Center (VSC)

Developing a GPU version of an already existing serial algorithm (Marching Tetrahedra). https://github.com/HliasGit/tetra

- The aim of the algorithm is to extract the approximation of an iso-surface from a scalar field.
- The scalar field (whose dimensions are DX,DY,DZ) is represented as a 3D tensor (of floats)
- Linearise (flatten) the scalar field to use it in a simpler way on GPU



Used three kernels to develop the GPU version:

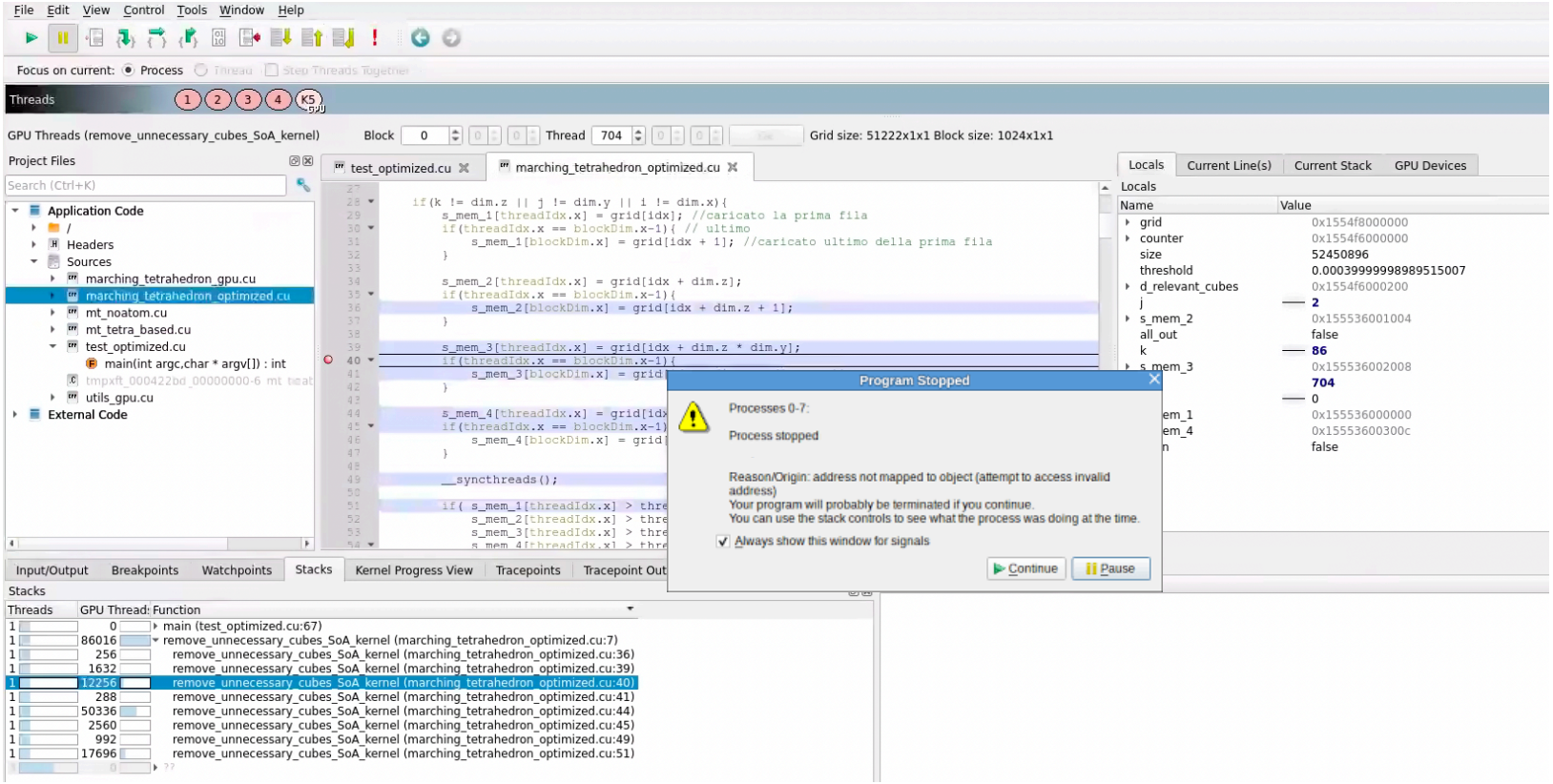
- The first one is a preprocessing step to reduce the computational complexity.
- Second and the third ones do the needed computation for the algorithm to work.

For each point in the scalar field, it needs to form a "cube" by accessing its 8 neighbours. Due to the Z-major linearization, only the neighbor in the Z-direction (index + 1) is adjacent in memory. Accessing neighbors in the X-direction (e.g., index + DZ*DY) or Y-direction (e.g., index + DZ) requires calculated memory offsets.

This approach led to "CUDA illegal memory access" errors. The problem was a failure to correctly handle boundary conditions. When a thread processed a point on the "edge" of the 3D domain (like at x = DX-1 or y = DY-1), these offset calculations were trying to access memory outside the allocated array. This boundary-checking logic was not intuitive, used Linaro DDT debugger to pinpoint these out-of-bounds access.

linaroforge

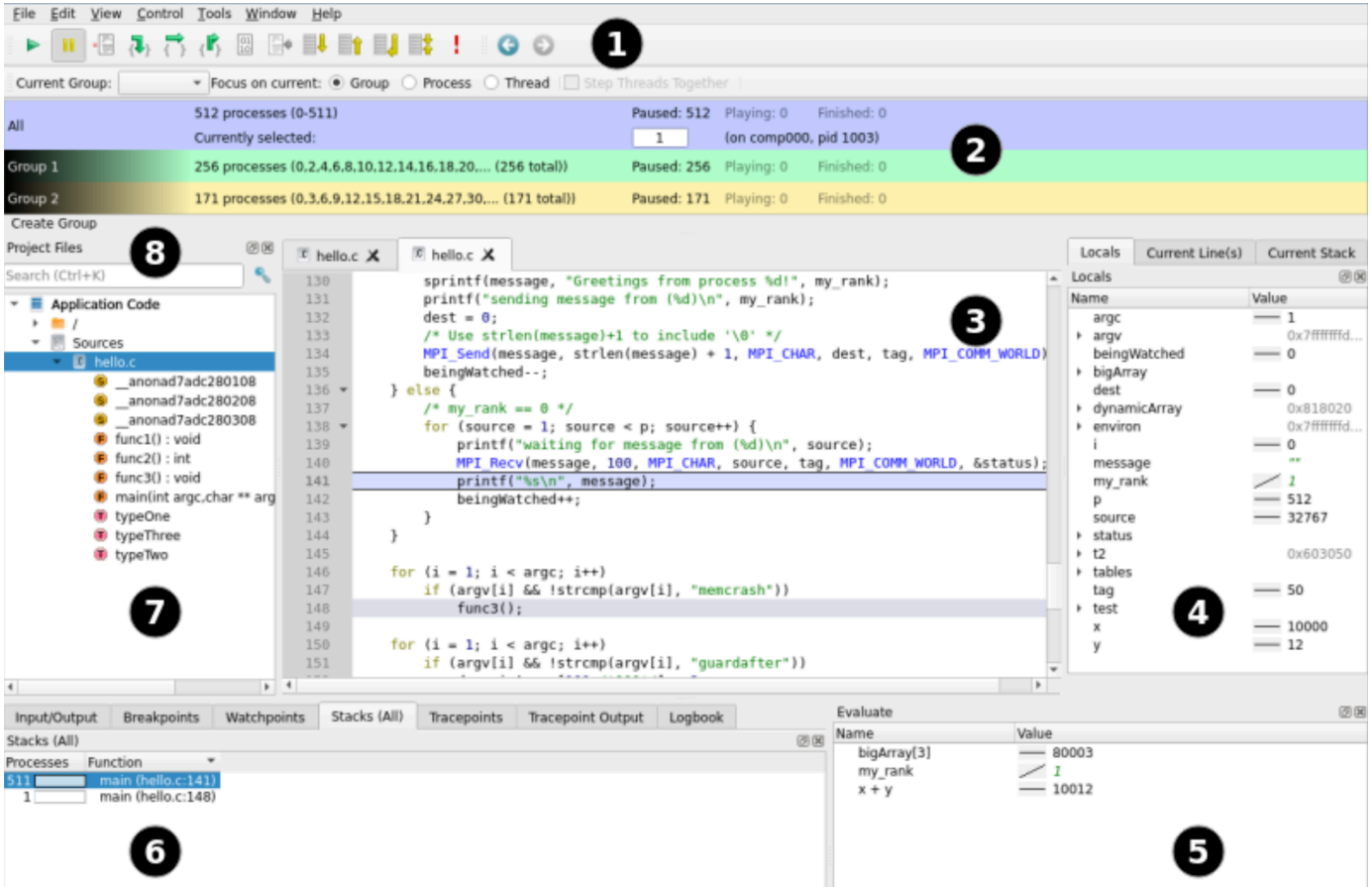
Spotting illegal memory access with Linaro DDT



linaroforge

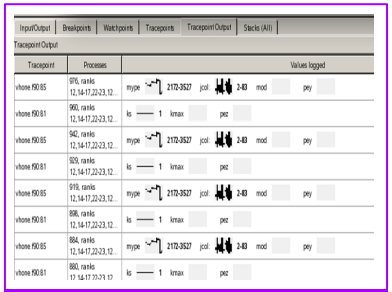
DDT UI

- 1 Process controls
- 2 Process groups
- 3 Source Code view
- 4 Variables
- 5 Evaluate window
- 6 Parallel Stack
- 7 Project files
- 8 Find a file or function

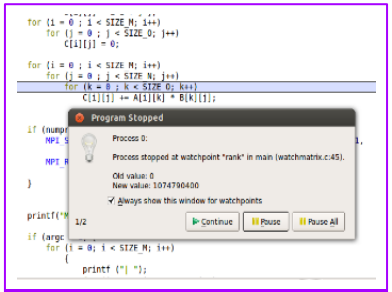


linaroforge

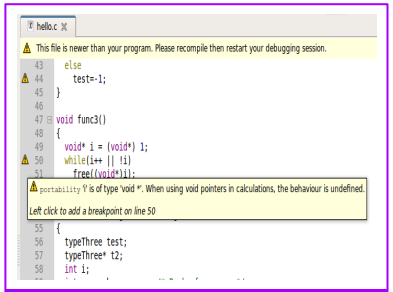
Linaro DDT Debugger Highlights



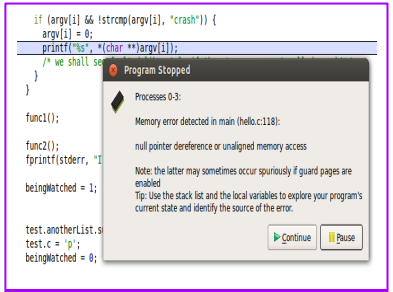
The scalable print alternative



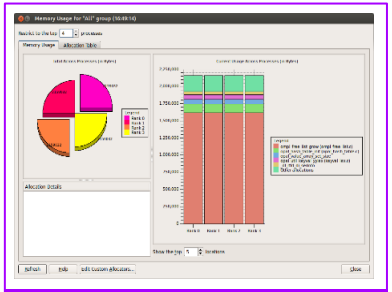
Stop on variable change



Static analysis warnings on code errors



Detect read/write beyond array bounds



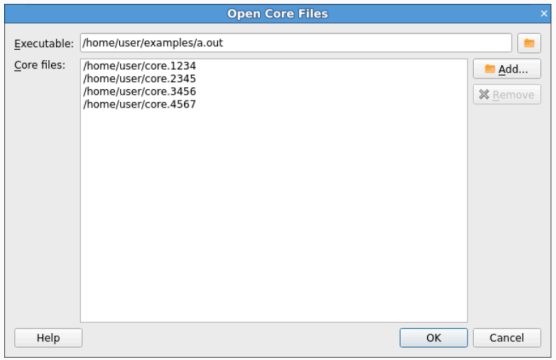
Detect stale memory allocations

Core files

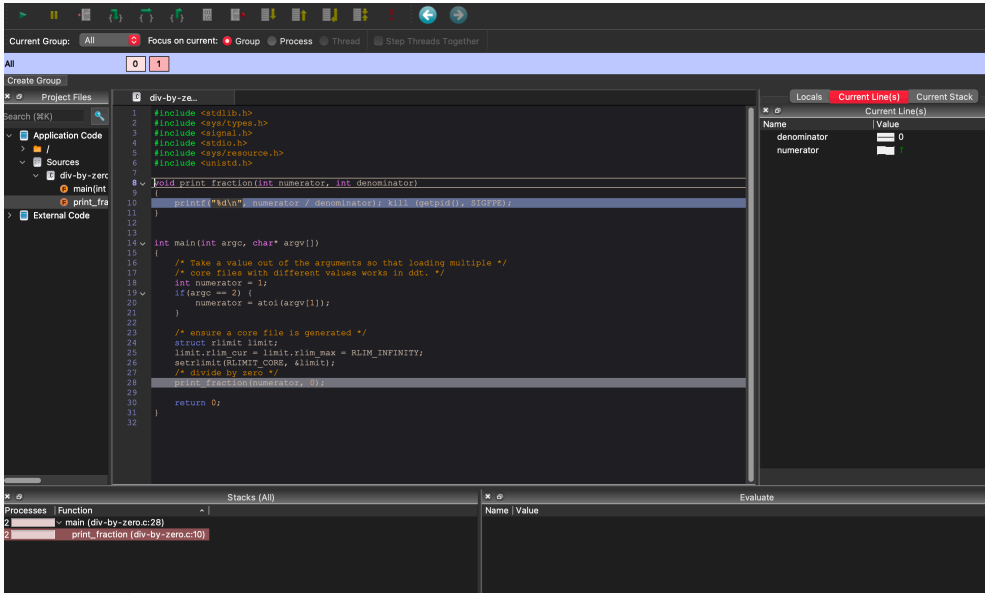
You can open and debug one or more core files generated by your application.

Procedure

1. On the Welcome page click **Open Core Files**. The **Open Core Files** window opens.



2. Select an executable and a set of core files, then click **OK** to open the core files and start debugging them.



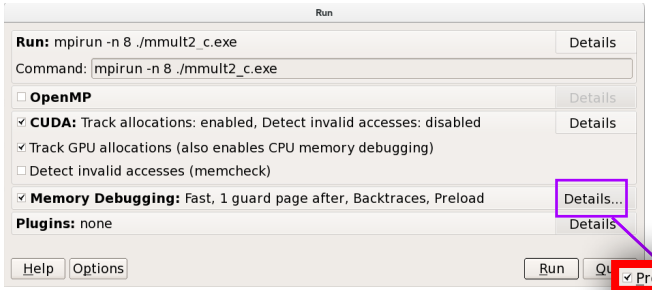
Note

While Linaro DDT is in this mode, you cannot play, pause, or step, because there is no process active. You are, however, able to evaluate expressions and browse the variables and stack frames saved in the core files.

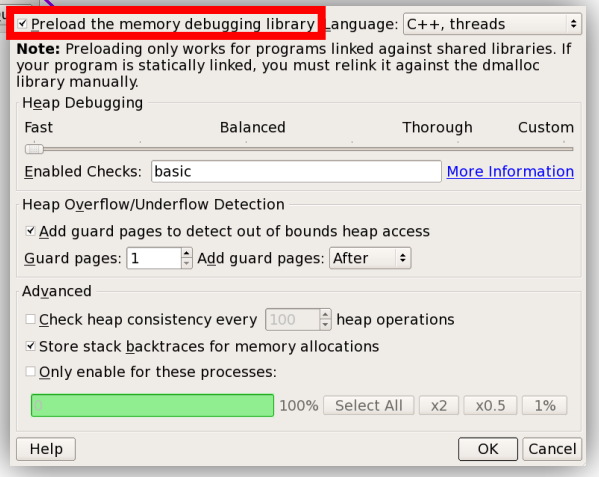
- View core files for CPU's
- View core files for GPU's

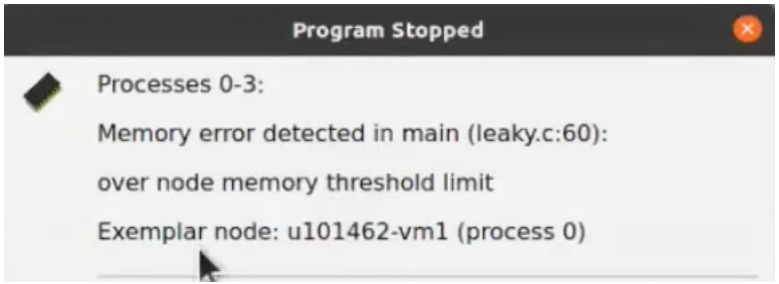
linaroforge

Memory debugging menu in Linaro DDT



When manual linking is used, untick “Preload” box





linaroforge

Multi-dimensional Array Viewer

What does your data look like at runtime?

View arrays

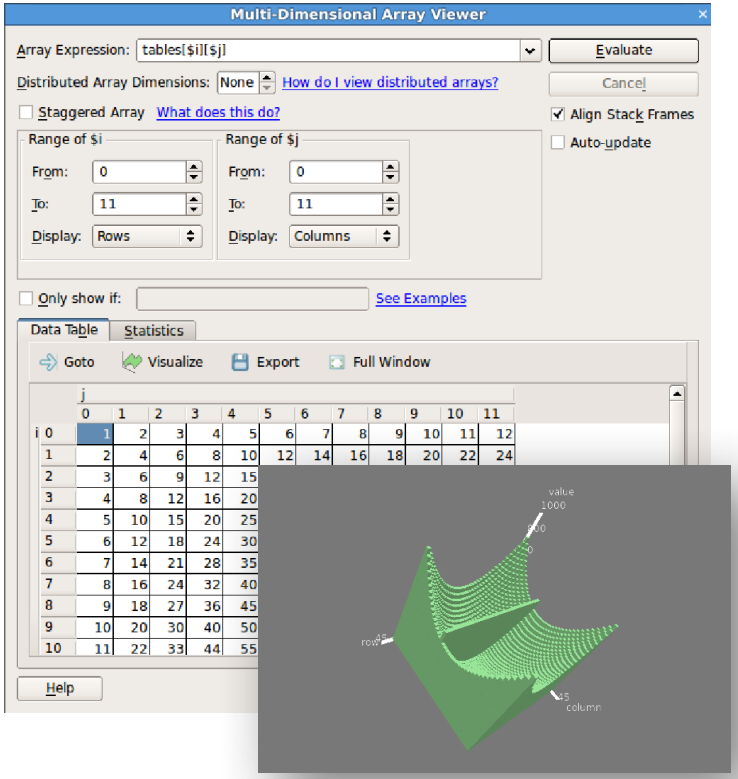
- On a single process
- Or distributed on many ranks

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



linaroforge

DDT: Production-scale debugging

Isolate and investigate faults at scale

Who misbehaved?

- Merge stacks from processes and threads
- Sparklines comparing data across processes
- Which MPI rank

Where is the problem?

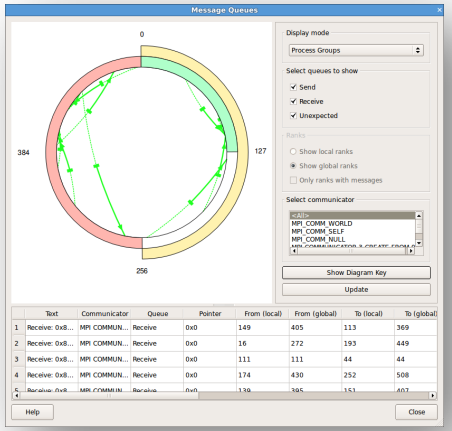
- Integrated source code editor
- Dynamic data structure visualization

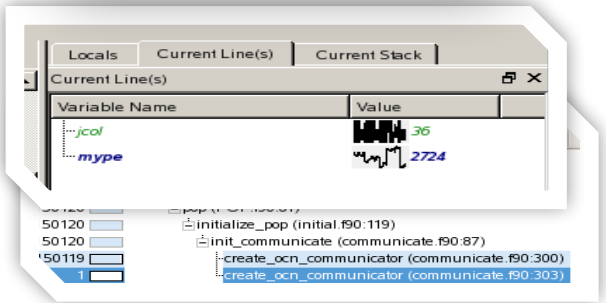
How did it happen?

- Parse diagnostic messages
- Trace variables through execution

Why did it happen?

- Unique “Smart Highlighting”
- Experiment with variable values

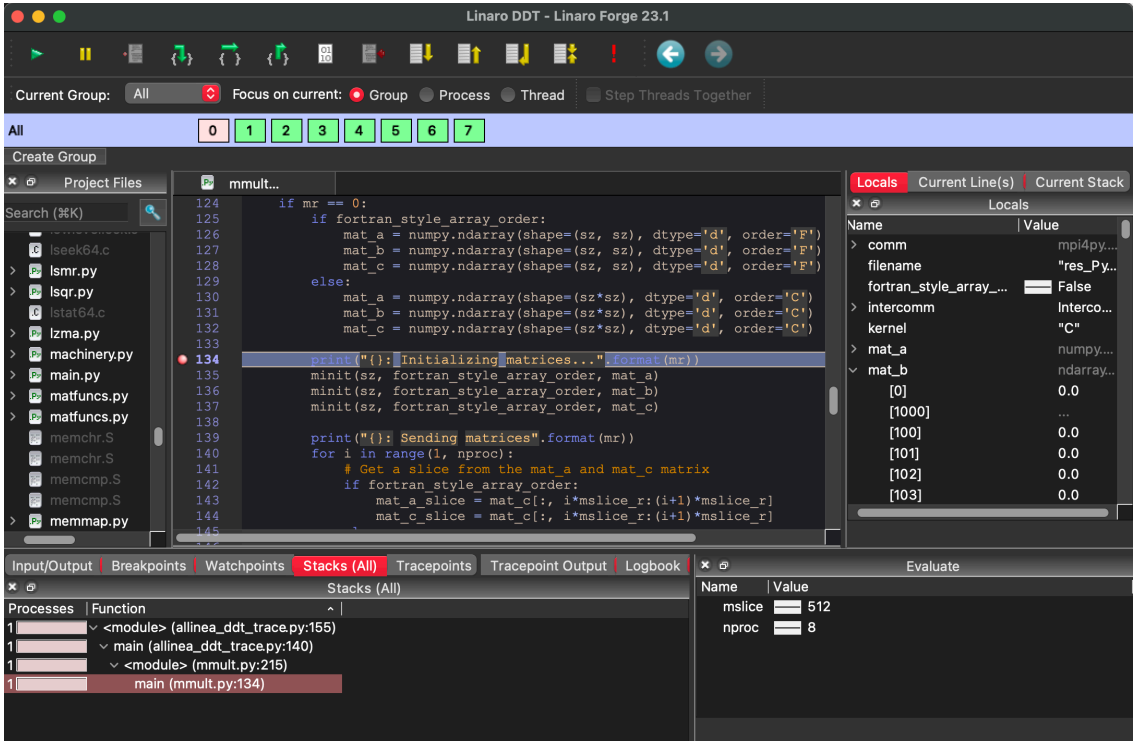




linaroforge

Python Debugging

- Debug Features
  - Sparklines for Python variables
  - Tracepoints
  - MDA viewer
  - Mixed language support
- Improved Evaluations:
  - Matrix objects
  - Array objects
  - Pandas DataFrame
  - Series objects
- Python Specific:
  - Stop on uncaught Python exception
  - Show F-string variables
  - Mpi4py, NumPy, SciPy



ddt --connect mpiexec -n 8 python3
%allinea_python_debug% ./mmult.py

Terminology

	NVIDIA GPU	AMD GPU	Intel GPU
Name	CUDA	ROCm	Xe
Language	CUDA C/C++ and Fortran	HIP	SYCL
Execution Unit	Warp	Wavefront	Sub-group
EU Size	32 Threads	64 Threads	8, 16 or 32 Threads
EU GDB	…	GDB Thread	GDB Thread
EU Thread	…	Lane	Lane (work-item)
Forge GPU Thread	Lane	Lane	Lane

linaroforge

Debugging Intel Xe GPUs

Using Linaro DDT

Debug code simultaneously on the GPU and the CPU

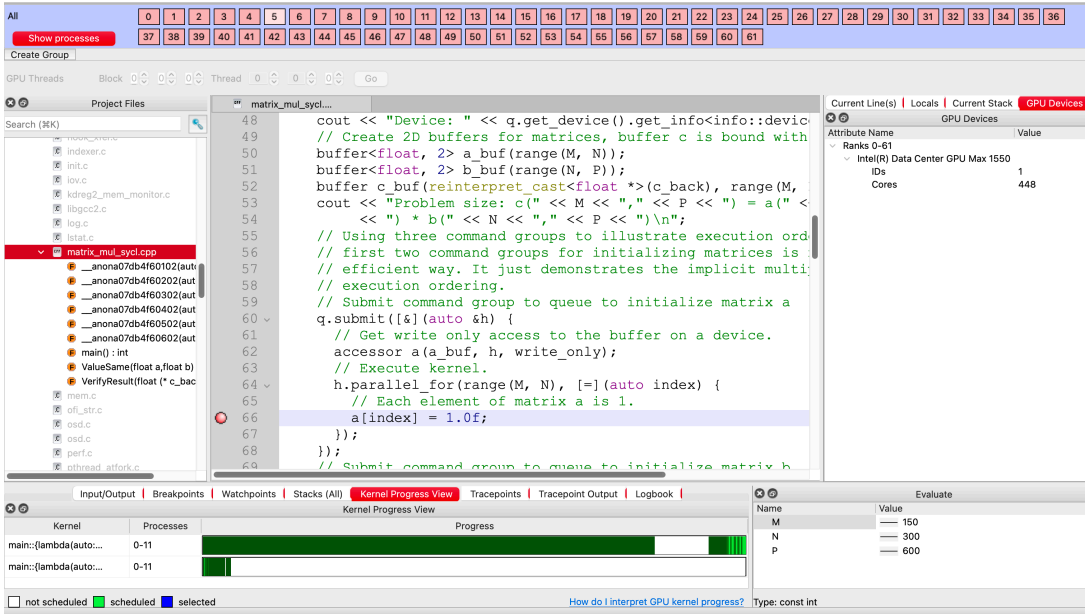
Controlling the GPU execution:

- All active threads in a Sub-group will execute in lockstep. Therefore, DDT will step16 threads at a time.
- Play/Continue runs all GPU threads
- Pause will pause a running kernel

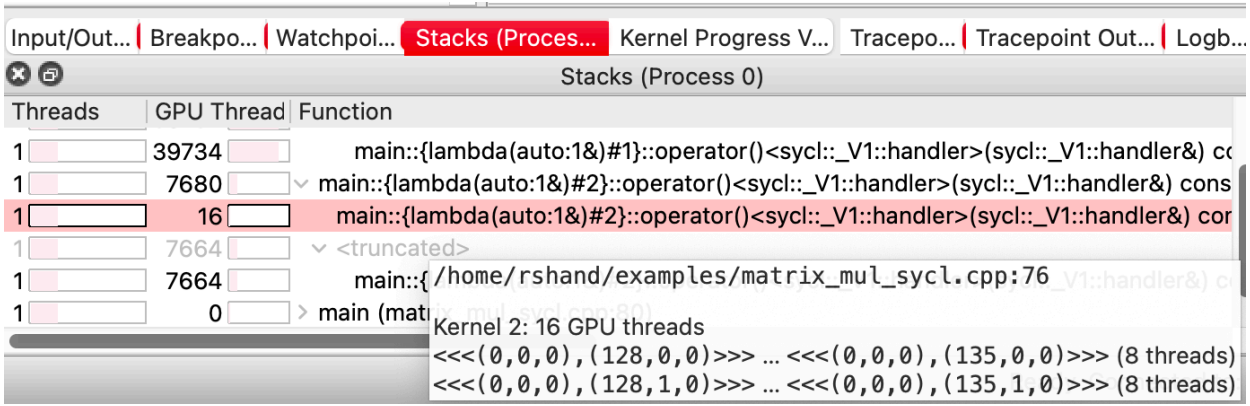
Key (additional) GPU features:

- Kernel Progress View
- GPU thread in parallel stack view
- GPU Thread Selector
- GPU Device Pane

Kernels must be compiled with the -g and -O0 flags



Parallel Stack View



- Display location and number of threads
- Click item:
  - Select GPU Thread.
  - Update variable display.
  - Move Source Code Viewer.
- Tooltip displays:
  - GPU Thread Ranges.
  - Size of each range.

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

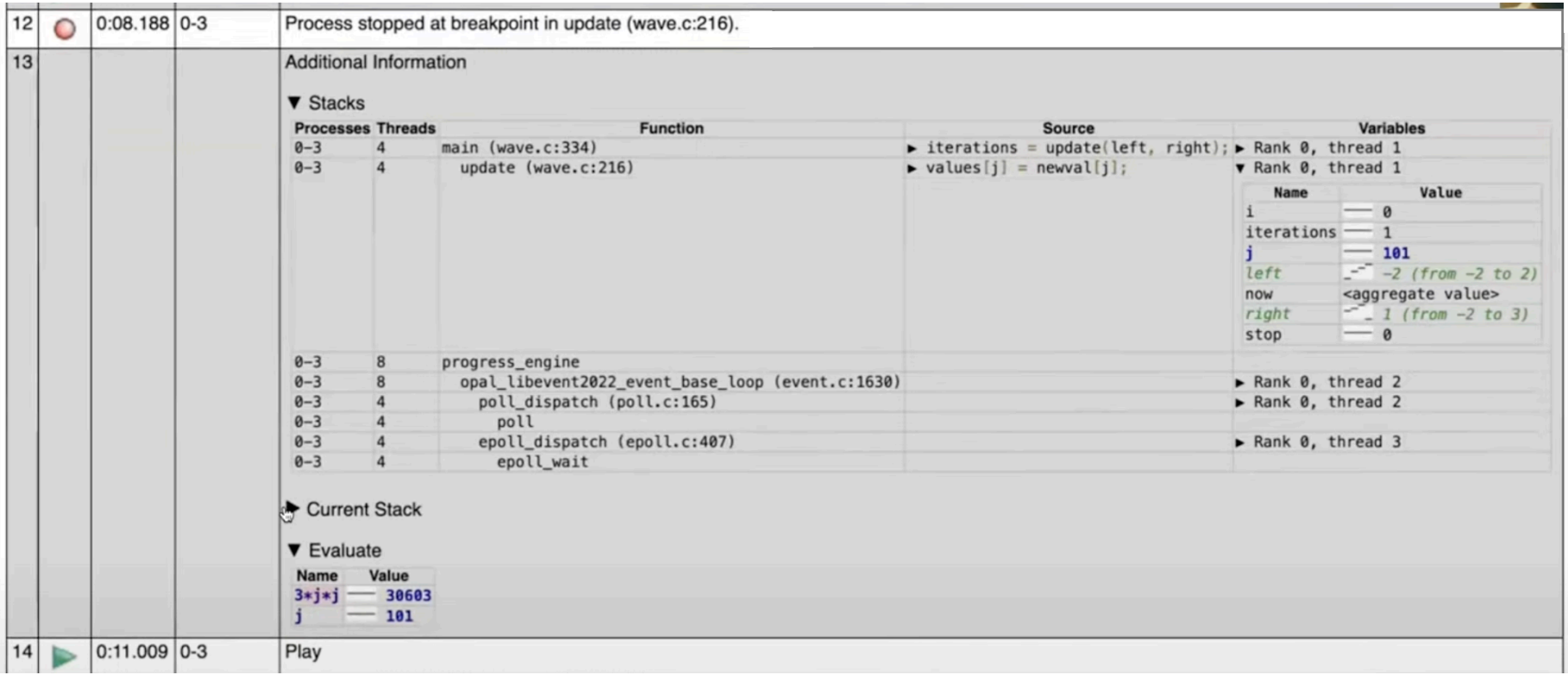
- For long-running jobs / debugging at very high scale
- For automated testing, continuous integration…

To do so, use following arguments:

- $ ddt **--offline --output=report.html** mpiexec ./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 --offline -o jacobi_omp_mpi_gnu_debug.txt \
                                 --trace-at _jacobi.F90:83,residual \
                                 mpiexec ./jacobi_omp_mpi_gnu.exe
```

Report output



12		0:08.188	0-3	Process stopped at breakpoint in update (wave.c:216).
13				Additional Information
14		0:11.009	0-3	Play

▼ Stacks

Processes	Threads	Function	Source	Variables
0-3	4	main (wave.c:334)	▶ iterations = update(left, right);	▶ Rank 0, thread 1
0-3	4	update (wave.c:216)	▶ values[j] = newval[j];	▼ Rank 0, thread 1
0-3	8	progress_engine		
0-3	8	opal_libevent2022_event_base_loop (event.c:1630)		▶ Rank 0, thread 2
0-3	4	poll_dispatch (poll.c:165)		▶ Rank 0, thread 2
0-3	4	poll		
0-3	4	epoll_dispatch (epoll.c:407)		▶ Rank 0, thread 3
0-3	4	epoll_wait		

Name	Value
i	0
iterations	1
j	101
left	-2 (from -2 to 2)
now	<aggregate value>
right	1 (from -2 to 3)
stop	0

▶ Current Stack

▼ Evaluate

Name	Value
3*j*j	30603
j	101

linaroforge

9 Step Guide

Optimizing high performance applications

Improving the efficiency of your parallel software holds the key to solving more complex research problems faster.

This pragmatic, 9 Step best practice guide, will help you identify and focus on application readiness, bottlenecks and optimizations one step at a time.

Bugs
- Correct application

Analyze before you optimize
- Measure all performance aspects.
  You can't fix what you can't see.
- Prefer real workloads over artificial tests.

I/O
- Discover lines of code spending a long time in I/O.
- Trace and debug slow access patterns.

Cores
- Discover synchronization overhead and core utilization
- Synchronization-heavy code and implicit barriers are revealed

Workloads
- Detect issues with balance.
- Slow communication calls and processes.
  Dive into partitioning code.

Vectorization
- Understand numerical intensity and vectorization level.
- Hot loops, unvectorized code and GPU performance reveleaed

Memory
- Reveal lines of code bottlenecked by memory access times.
- Trace allocation and use of hot data structure

Communication
- Track communication performance.
- Discover which communication calls are slow and why.

Verification
- Validate corrections and optimal performance

Key : Linaro Forge
Linaro Performance Reports

linaroforge

Linaro Performance Reports

Characterize and understand the performance of HPC application runs

Commercially supported by Linaro

Gather a rich set of data

- Analyses metric around CPU, memory, IO, hardware counters, etc.
- Possibility for users to add their own metrics

Accurate and Astute insight

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

Relevant advice to avoid pitfalls

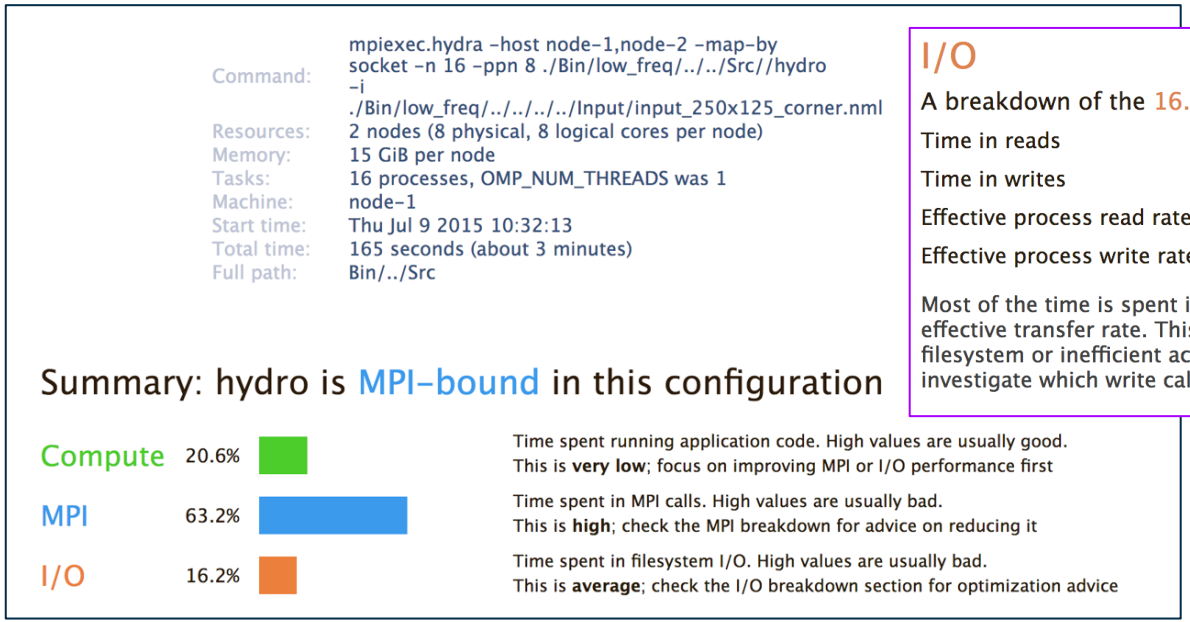
Adds value to typical users' workflows

- Define application behaviour and performance expectations
- Integrate outputs to various systems for validation (eg. continuous integration)
- Can be automated completely (no user intervention)

linaroforge

Linaro Performance Reports

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



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:

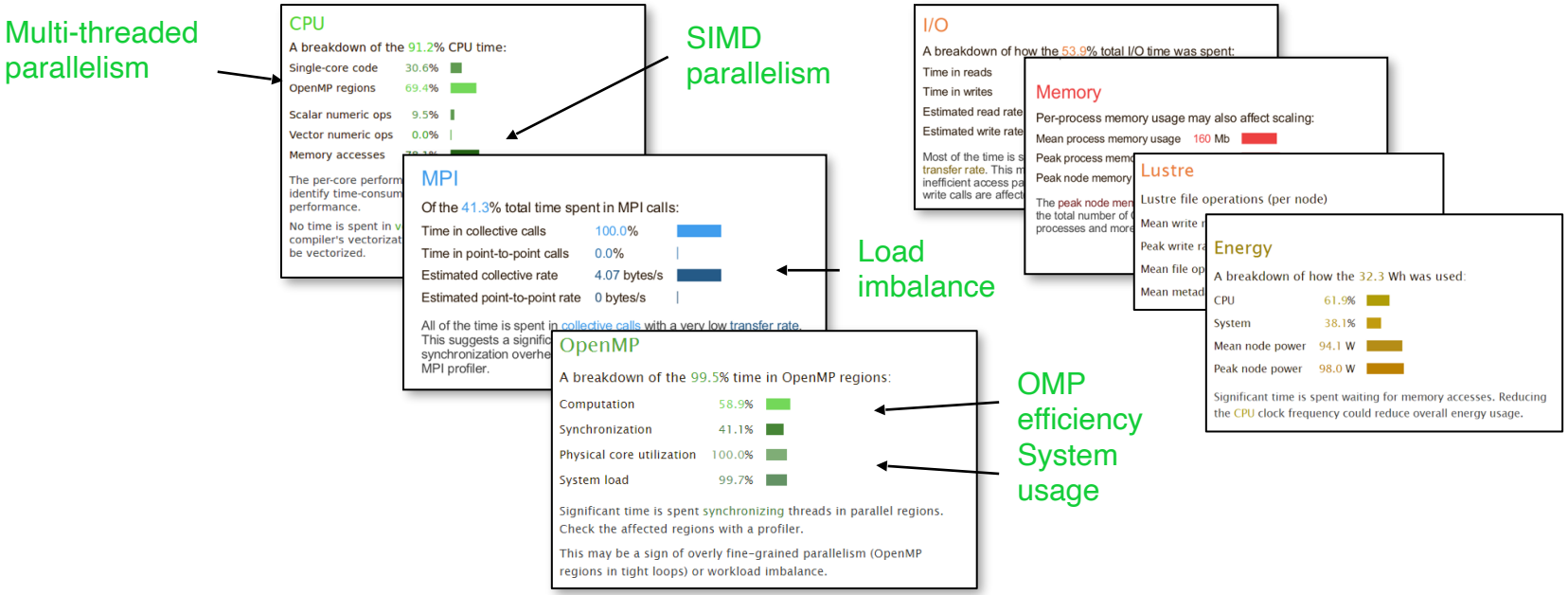
Time in reads	0.0%
Time in writes	100.0%
Effective process read rate	0.00 bytes/s
Effective process write rate	1.38 MB/s

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.

linaroforge

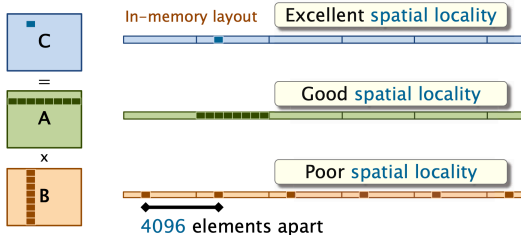
Linaro Performance Reports Metrics

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

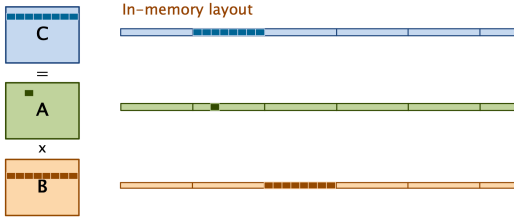


Performance Improvement

i, j, k



i, k, j



Think,

code,

run, run, run…

…to test and measure many different implementations

Loop order (outer to inner)	Running time (s)
i, j, k	1155.77
i, k, j	177.68
j, i, k	1080.61
j, k, i	3056.63
k, i, j	179.21
k, j, i	3032.82

i, j, k

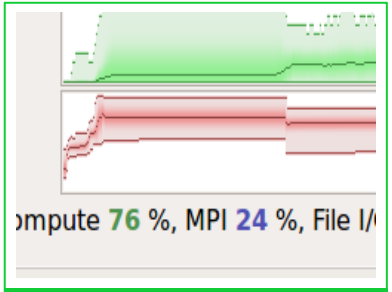
```
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < n; ++j) {
    for (int k = 0; k < n; ++k) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

i, k, j

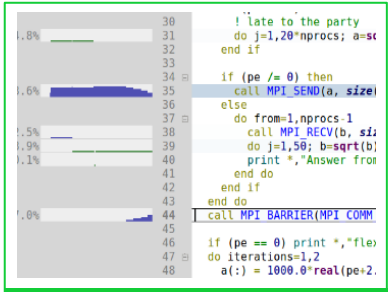
```
for (int i = 0; i < n; ++i) {
    for (int k = 0; k < n; ++k) {
  for (int j = 0; j < n; ++j) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

linaroforge

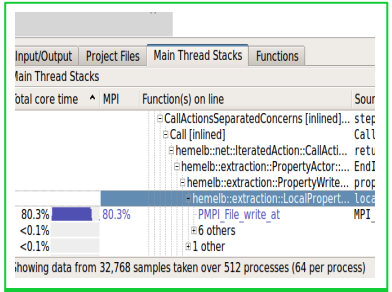
Linaro MAP Source Code Profiler Highlights



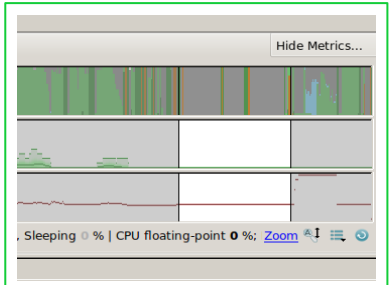
Find the peak memory use



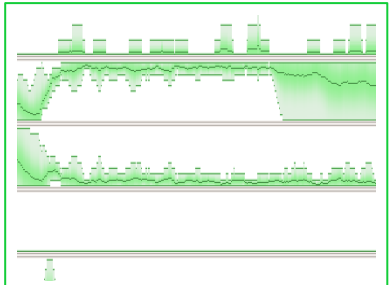
Fix an MPI imbalance



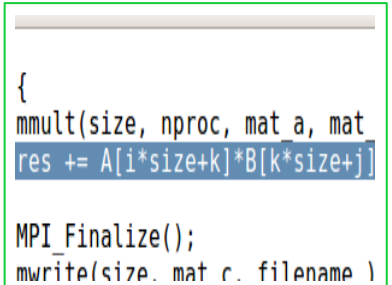
Remove I?O bottleneck



Make sure OpenMP regions make sense



Improve memory access



Restructure for vectorization

Case Study: Fire Dynamics Simulator (FDS)

Issue that we encountered with AWS when porting an app called FDS: https://github.com/firemodels/fds.

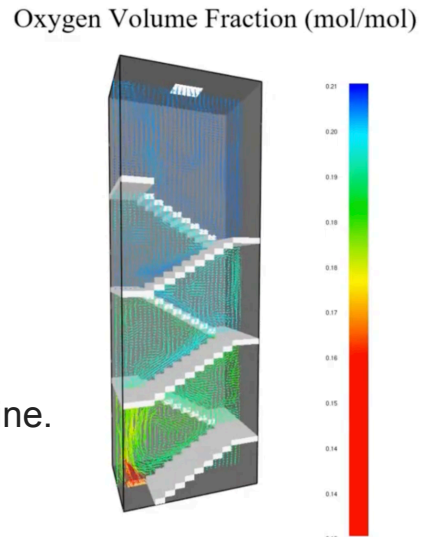
Software for fire & smoke modelling

Collaboration between Arm, AWS and NIST

Porting the application from x86_64 to Arm Graviton.

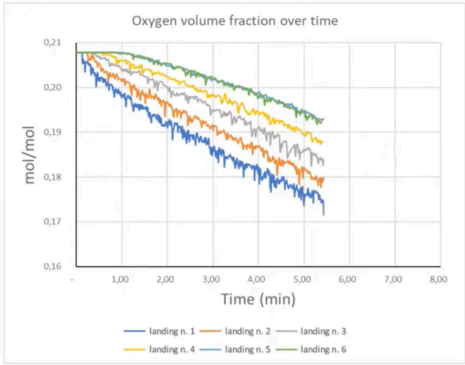
Spotted an inefficient MPI call routine using MAP

Fair amount of time that was spent on this “inquire” line.



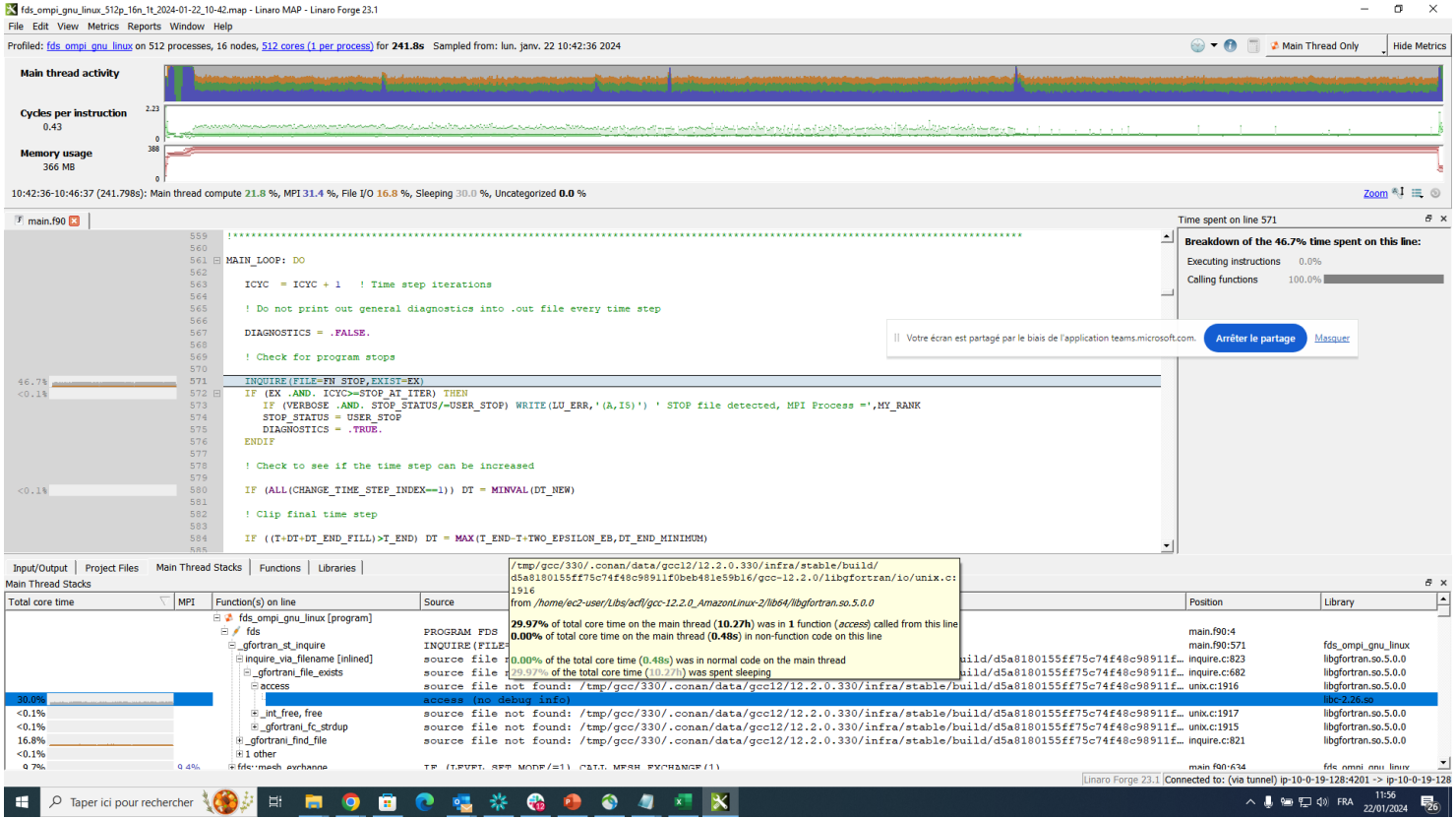
Simulation inputs:

- Soot yield = 0,03 kg/kg
- Carbon monoxide yield = 0,07 kg/kg
- Heat Release Rate = 120 kW



linaroforge

Spotting Bottleneck with Linaro MAP



MAP Capabilities

MAP is a sampling based scalable profiler

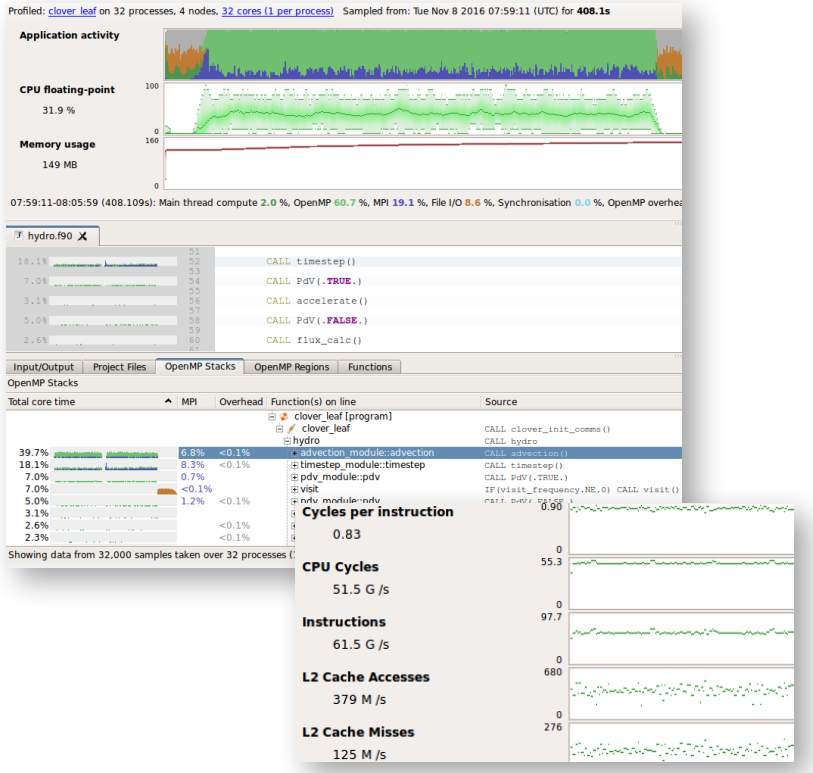
- Built on same framework as DDT
- Parallel support for MPI, OpenMP, CUDA
- Designed for C/C++/Fortran

Designed for ‘hot-spot’ analysis

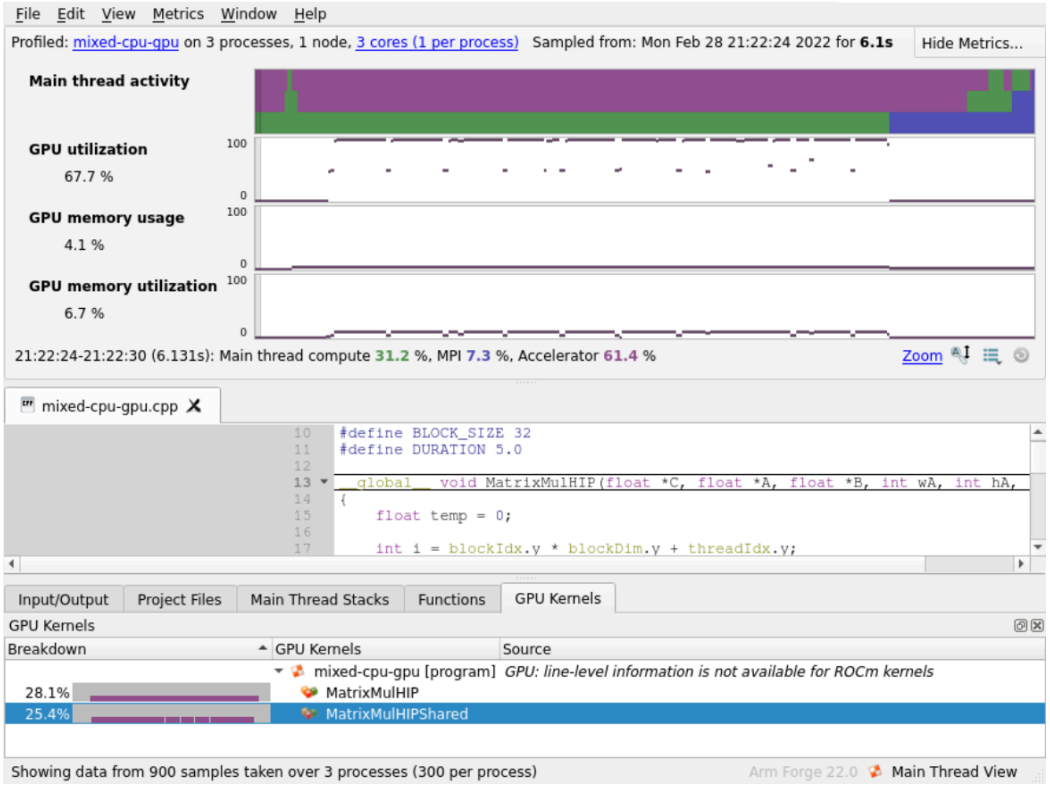
- Stack traces
- Augmented with performance metrics

Adaptive sampling rate

- Throws data away - 1,000 samples per process
- Low overhead, scalable and small file size



GPU Profiling



Profile

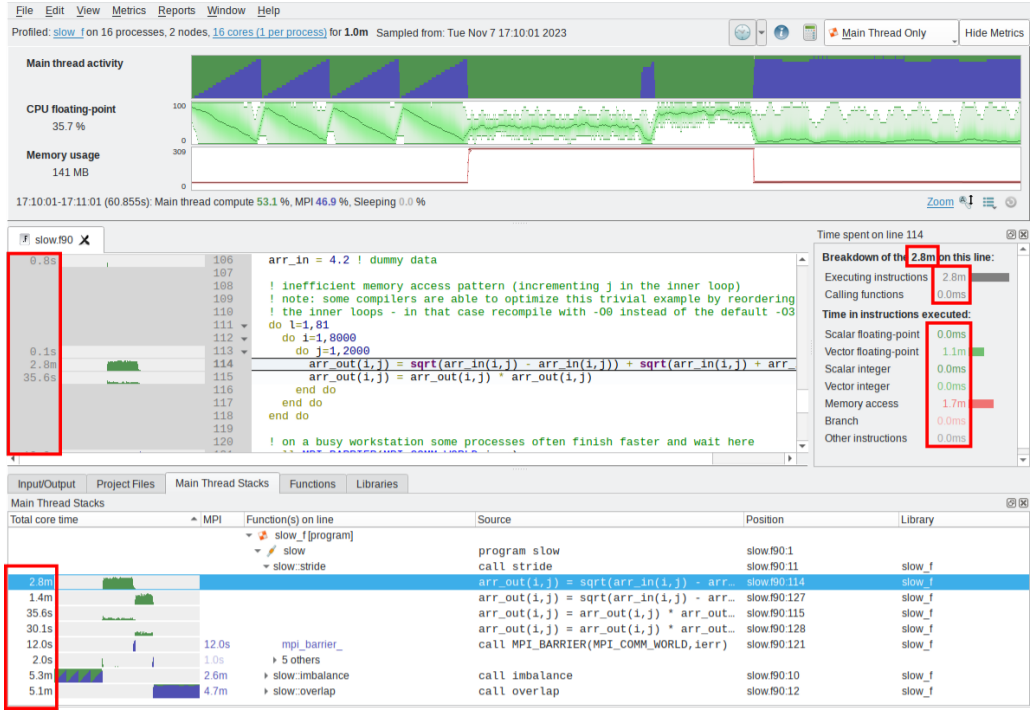
- Supports both AMD and Nvidia GPUs
- Able to bring up metadata of the profile
- Mixed CPU [green] / GPU [purple] application
- CPU time waiting for GPU Kernels [purple]
- GPU Kernels graph indicating Kernel activity

GUI information

- GUI is consistent across platforms
- Zoom into main thread activity
- Ranked by highest contributors to app time

linaroforge

Toggle percentage-time and core-time in MAP



Use for direct comparisons between runs at the same scale (process/core counts).

- Easily determine if a change has made a portion of code faster, slower, or largely unchanged.
- Performance report automatically includes both percentage-time and core time
- Core-time is an estimation, but should be very close to the application run time

linaroforge

Python Profiling

19.0 adds support for Python

- Call stacks
- Time in interpreter

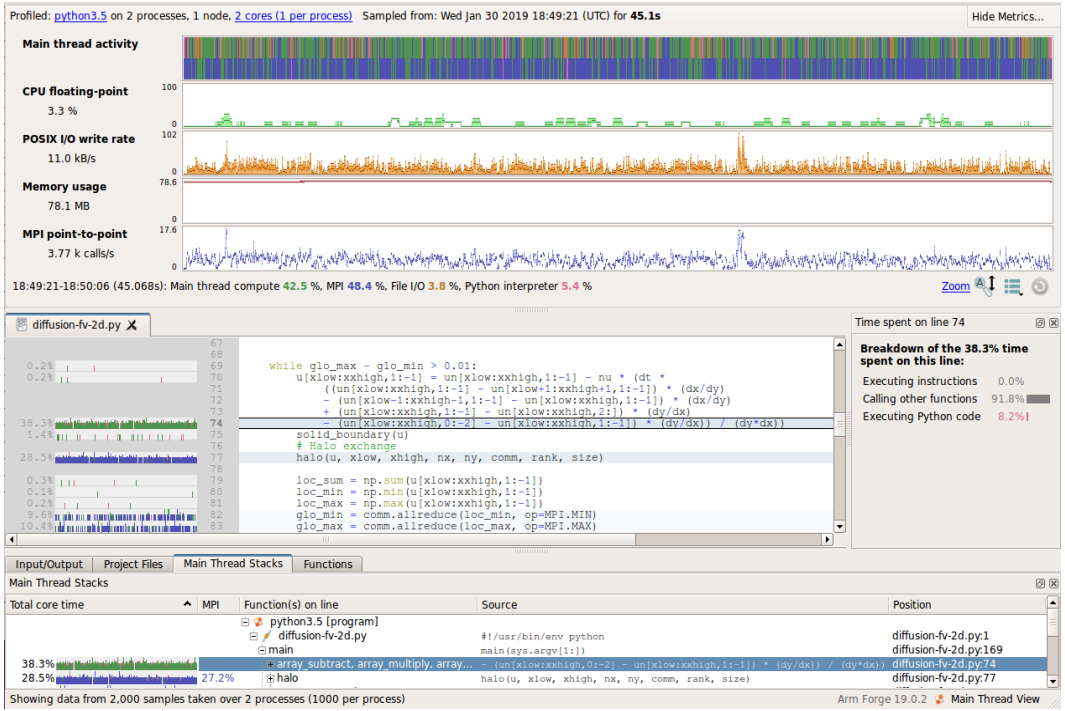
Works with MPI4PY

- Usual MAP metrics

Source code view

- Mixed language support

Note: Green as operation is on numpy array, so backed by C routine, not Python (which would be pink)

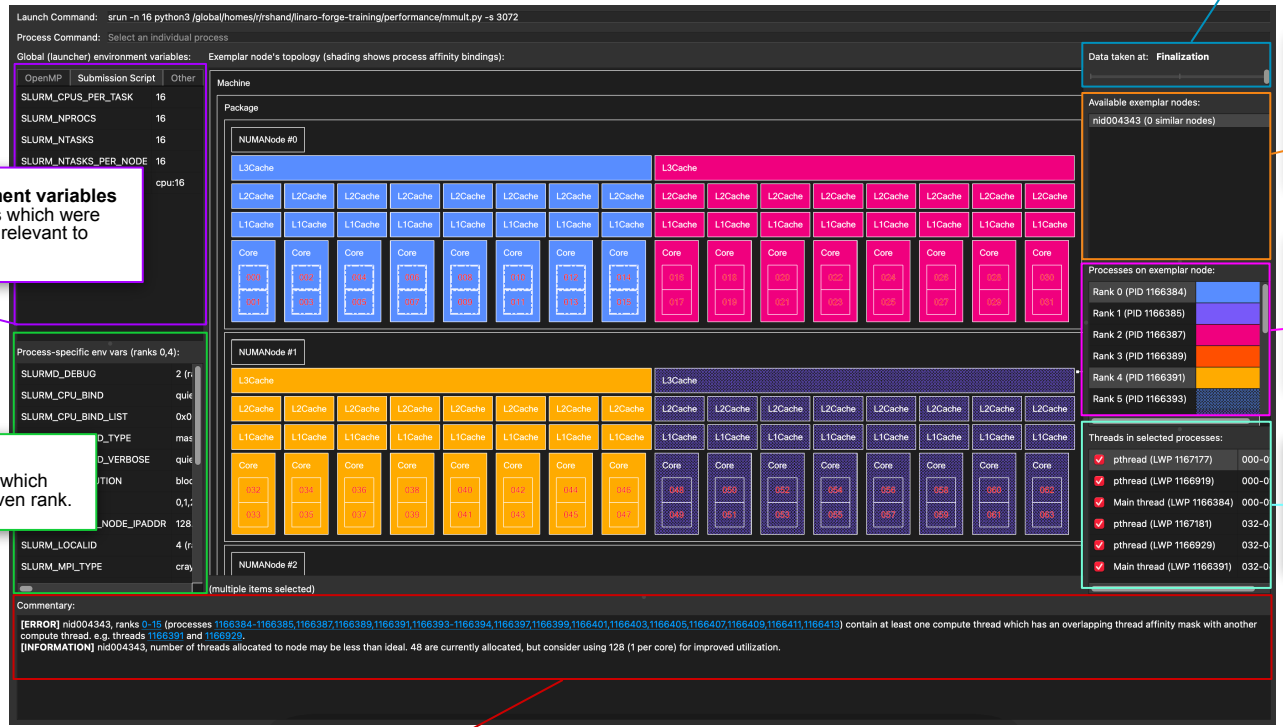



```
map --profile mpiexec -n 2 python ./diffusion-fv-2d.py
```

linaroforge

MAP Thread Affinity Advisor

Snapshot Selector
Change at which point of a run the Affinity data is shown (*Library Load, Initialisation, Finalization*).



Global (launcher) environment variables
List of Environment Variables which were set at launch which might be relevant to how threads are distributed.

Process-specific env vars
List of Environment Variables which might affect the affinity of a given rank.

Exemplar Nodes
Selectable list of exemplars, allowing ability to switch data between nodes of a run. Nodes with similar affinity/structures are merged.

Processes List
List of processes (by MPI rank) of the selected exemplar. Shows the key for the node topology diagram and selecting one shows all threads for the process.

Threads List
List of all threads for the selected process. Selecting threads highlights which cores they are bound to in the topology view.

Commentary
A list of commentary, providing information and advice on Memory Imbalance, Core Utilization etc.

linaroforge

Forge hands-on

Rudy Shand
Field Application Engineer

linaroforge

Cheat Sheet

Training material

Slides are in google-drive

https://gitlab.com/Linaro/training/forge

Training content

linaro-forge-training.pdf (presenter slides)

run-job.sh (Job script for system)

Forge Client (On local machine)

Install Forge client *https://www.linaroforge.com/downloadForge*

Running with a batch script

sbatch run-job.sh

Forge commands

```
ddt --connect     # Reverse connect
ddt --offline     # Run DDT without GUI
map --profile     # Profile without GUI
perf-report       # Generate Performance Report
```

slurm *queue commands*

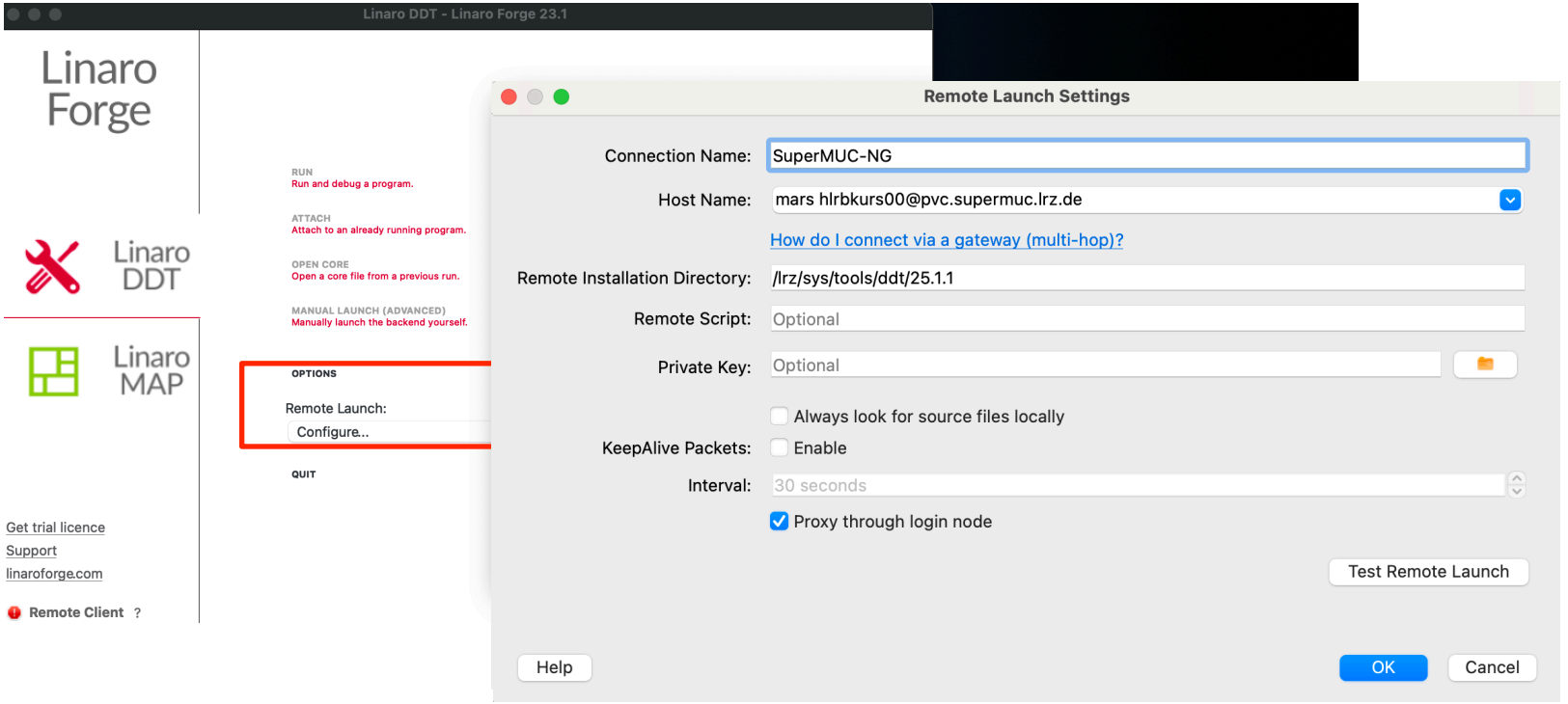
```
sbatch <jobscript>  # Submit a job to queue
squeue              # Check job queue
scancel  <job-id>   # Remove job from queue
```

Guides

Forge userguide

HPC System information

Remote connection to SuperMUC-NG



linaroforge

Thank you

Go to www.linaroforge.com