

# Linaro Tools

**Rudy Shand**  
Linaro Ltd



# HPC development Solutions from Linaro

Build reliable and optimized code on multiple Server and HPC architectures

Linaro Forge combines



## Linaro DDT

Market leading, simple to use  
HPC debugger for C/C++,  
Fortran and Python  
applications.



## Linaro MAP

Effortless performance  
analysis for experts and  
novices alike.



## Linaro Performance Reports

At a glance, single-page, application  
performance summary.

**Performance  
Engineering for any  
architecture, at any  
scale**



# gdb under the hood

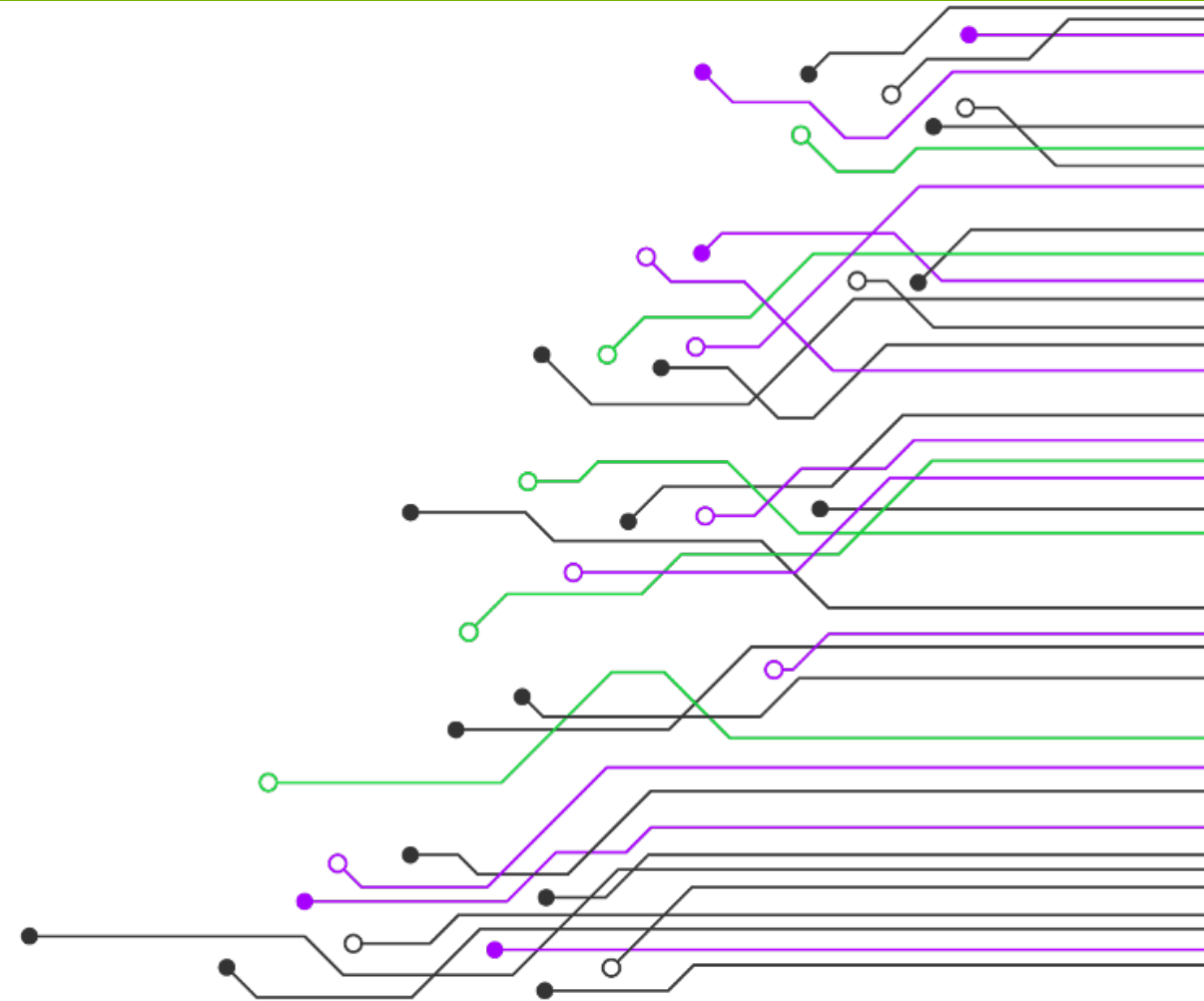
## Leveraging the gdb community

GDB is the underlying technology

- Linaro upstreams patches to the gdb community
- Forge team raises and fixes gdb bugs

Nimble at supporting new technologies

- Rely on hardware vendors to add gdb support
- Rely on software consortiums to add gdb support
- Helps us to stay current with technologies
- Provides a state of the art debugger

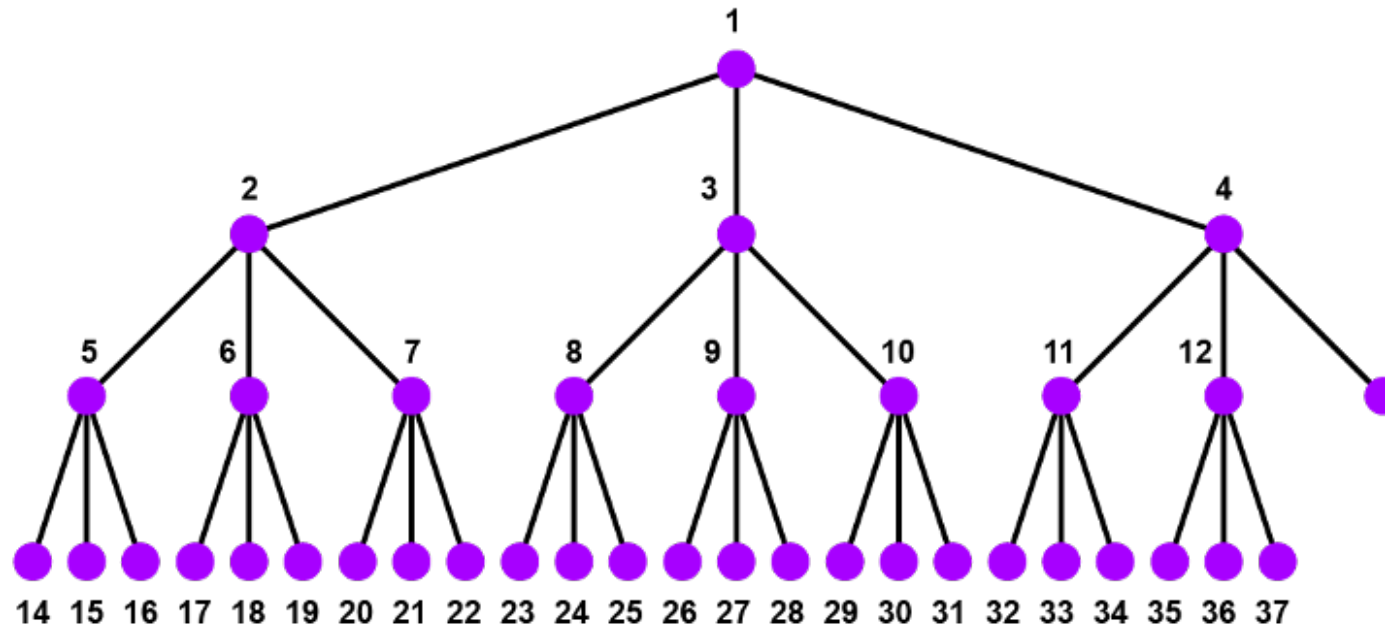




# Scaling gdb

## Tree network topology

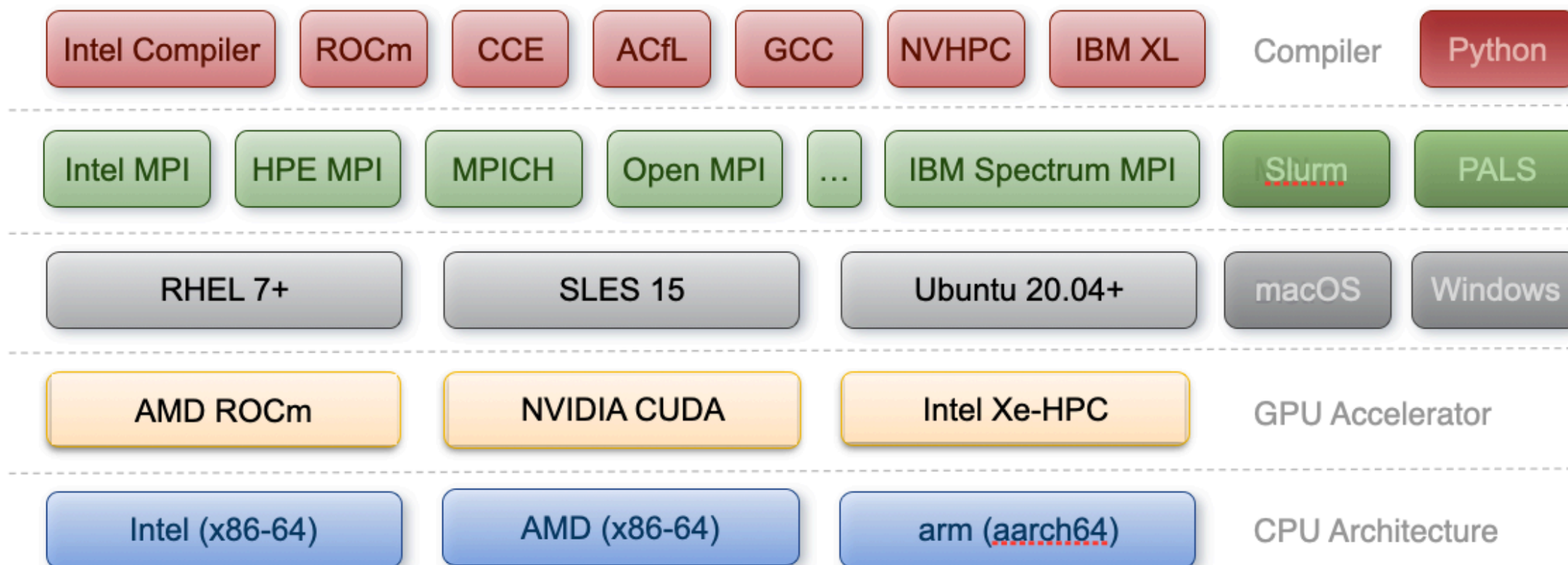
- Tree server design is how Forge is able to scale
- Send bulk commands and merge responses
- Aggregate the data instead of broadcasting thousands of responses





# DDT Supported platforms

Works across hardware architectures and HPC technologies





# 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

The screenshot displays the DDT (Data Display Tool) interface, which is used for debugging and monitoring parallel applications. The interface is divided into several panels, each with a numbered callout:

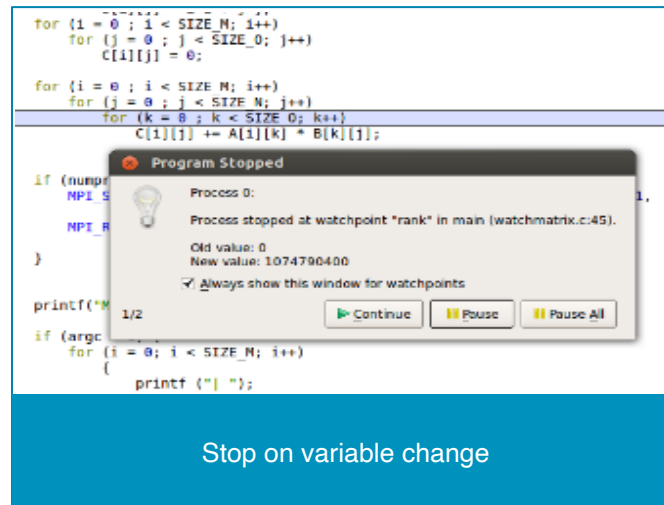
- 1 Process controls:** Located at the top, it includes a menu bar (File, Edit, View, Control, Tools, Window, Help) and a toolbar with icons for running, pausing, and stepping through code.
- 2 Process groups:** A table below the toolbar showing the status of different process groups. It includes columns for 'Current Group', 'Focus on current', 'Group', 'Process', 'Thread', and 'Step Threads Together'. The table lists 'All' (512 processes), 'Group 1' (256 processes), and 'Group 2' (171 processes).
- 3 Source Code view:** The central pane showing the source code of the application. It includes a search bar (Search (Ctrl+K)) and a list of project files on the left. The code is displayed in a syntax-highlighted format.
- 4 Variables:** A panel on the right side showing the current state of variables. It includes a table with columns for 'Name' and 'Value'. The table lists variables such as 'argc', 'argv', 'beingWatched', 'bigArray', 'dynamicArray', 'environ', 'i', 'message', 'my\_rank', 'p', 'source', 'status', 't2', 'tables', 'tag', 'test', 'x', and 'y'.
- 5 Evaluate window:** A panel at the bottom right used for evaluating expressions. It includes a table with columns for 'Name' and 'Value'. The table lists expressions such as 'bigArray[3]', 'my\_rank', and 'x + y'.
- 6 Parallel Stack:** A panel at the bottom left showing the stack of processes. It includes a table with columns for 'Processes' and 'Function'. The table lists processes such as '511' and '1'.
- 7 Project files:** A panel on the left side showing the project files. It includes a tree view of the project structure, with files such as 'hello.c' and 'main'.
- 8 Find a file or function:** A search bar located in the top left corner of the source code view, used for finding files and functions.



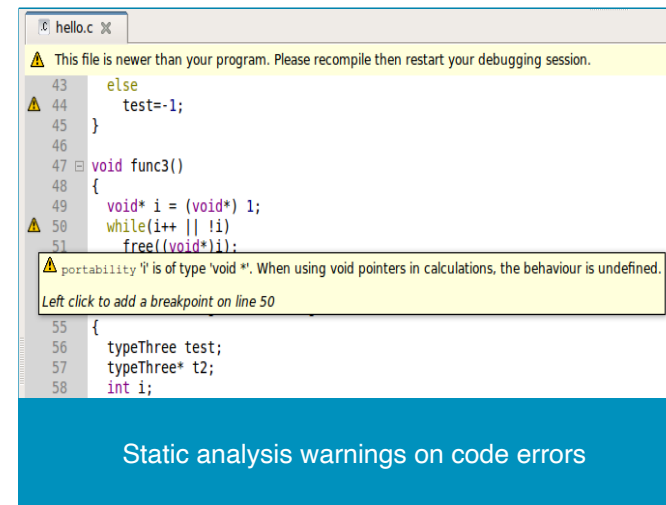
# DDT Highlights

Input/Output	Breakpoints	Watchpoints	Tracepoints	Tracepoint Output	Stacks (All)
Tracepoint Output					
Tracepoint	Processes	Values logged			
vhone:90:85	976, ranks 12,14-17,22-23,12...	mtype 2172-3527 jcol: 2-83 mod	pey		
vhone:90:81	960, ranks 12,14-17,22-23,12...	ls 1 kmax	pez		
vhone:90:85	942, ranks 12,14-17,22-23,12...	mtype 2172-3527 jcol: 2-83 mod	pey		
vhone:90:81	929, ranks 12,14-17,22-23,12...	ls 1 kmax	pez		
vhone:90:85	919, ranks 12,14-17,22-23,12...	mtype 2172-3527 jcol: 2-83 mod	pey		
vhone:90:81	898, ranks 12,14-17,22-23,12...	ls 1 kmax	pez		

The scalable print alternative



Stop on variable change



Static analysis warnings on code errors

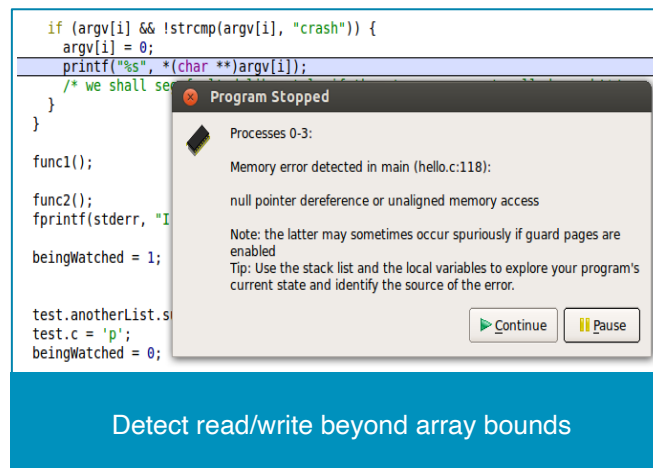
## Memory Debugging

# Enable reading of debug environment variables:

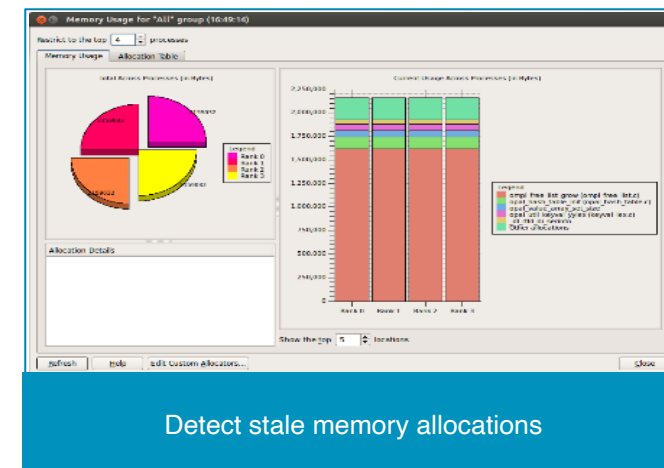
host:~/demo> export NEORadDebugKeys=1

# Disable RTLD\_DEEPBIND, so that malloc/free calls bind to  
# our memory debug library, instead of glibc:

host:~/demo> export NEODisableDeepBind=1



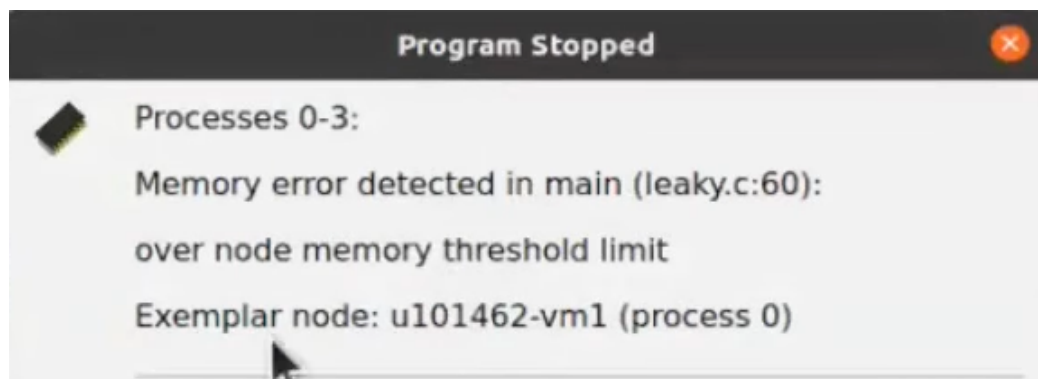
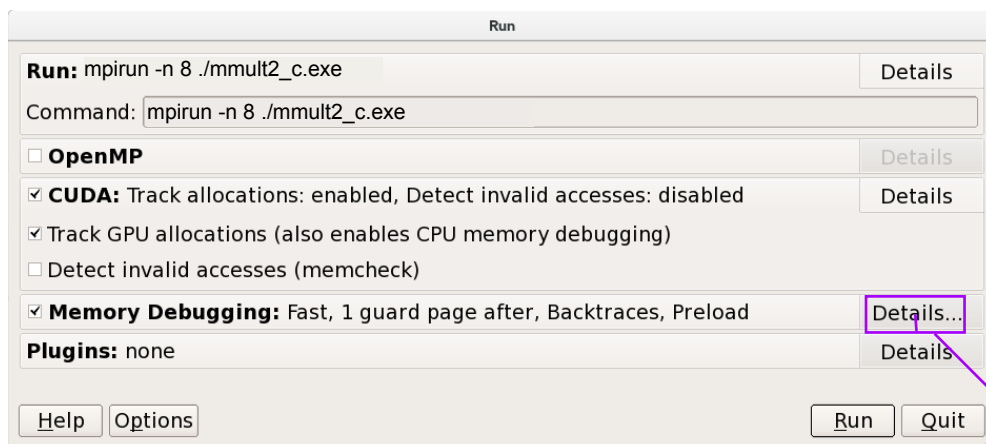
Detect read/write beyond array bounds



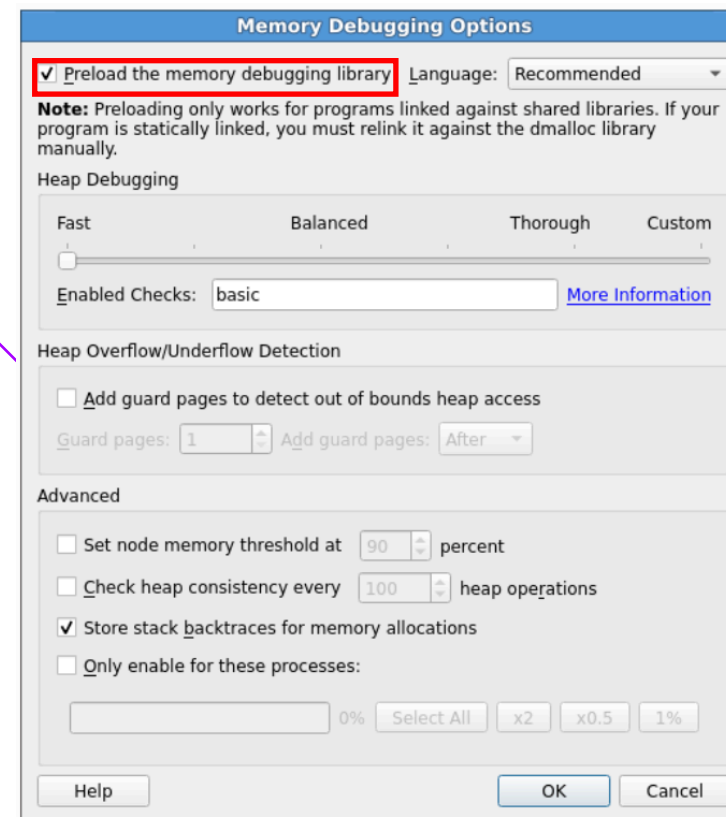
Detect stale memory allocations



# Memory Debugging



When manual linking is used,  
untick "Preload" box





# MDA Viewer

## What does your data look like at runtime?

### View arrays

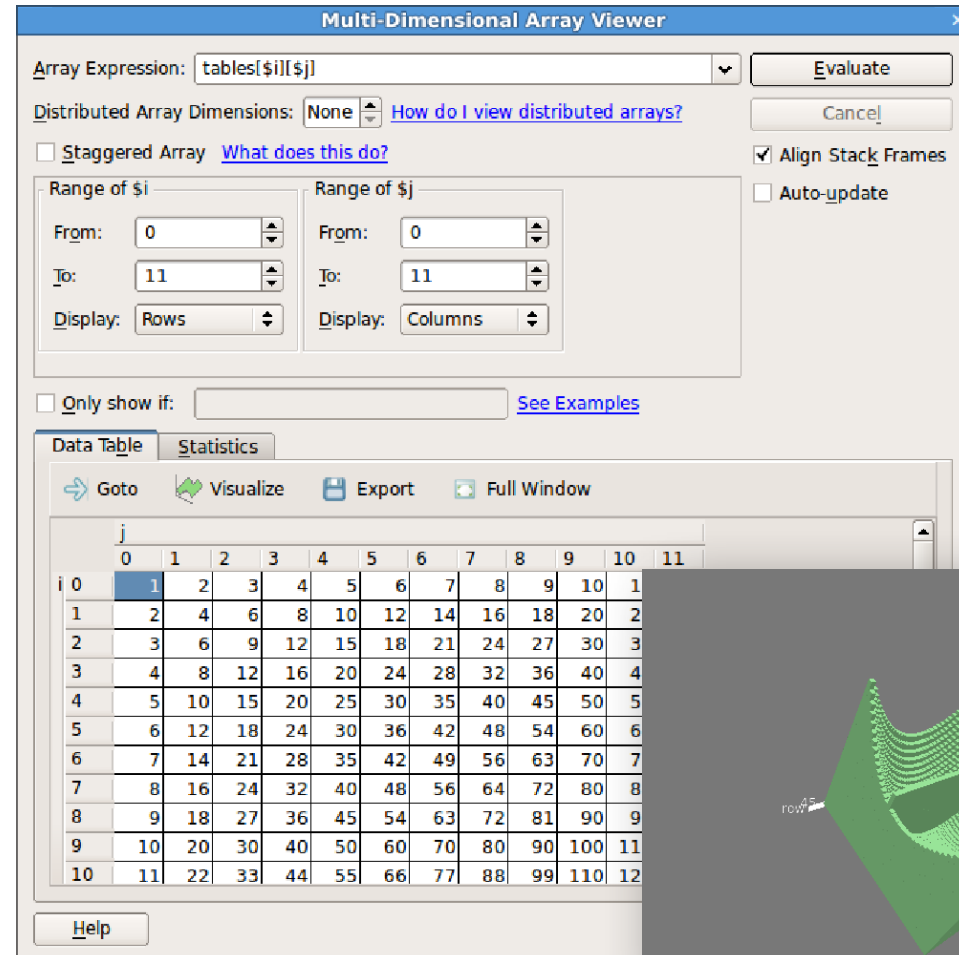
- On a single process
- Or distributed on many ranks

### 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





# Terminology

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



# 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 step 16 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

The screenshot displays the Linaro DDT interface for debugging Intel Xe GPUs. The top panel shows a grid of 64 GPU threads, with threads 0-11 highlighted in red. The main editor shows the source code of a C++ program, `matrix_mul_sycl.cpp`, which demonstrates GPU acceleration using SYCL. The code includes comments and function calls like `q.submit` and `h.parallel_for`. The bottom panel features the 'Kernel Progress View', which shows the progress of the kernel execution across different processes (0-11). The right sidebar displays the 'GPU Devices' pane, showing the configuration of the Intel(R) Data Center GPU Max 1550, including 1 ID and 448 Cores. The bottom right corner shows an 'Evaluate' pane with variables M, N, and P, and their corresponding values (150, 300, and 600).



# Debugging Nvidia GPUs

## Using Linaro DDT

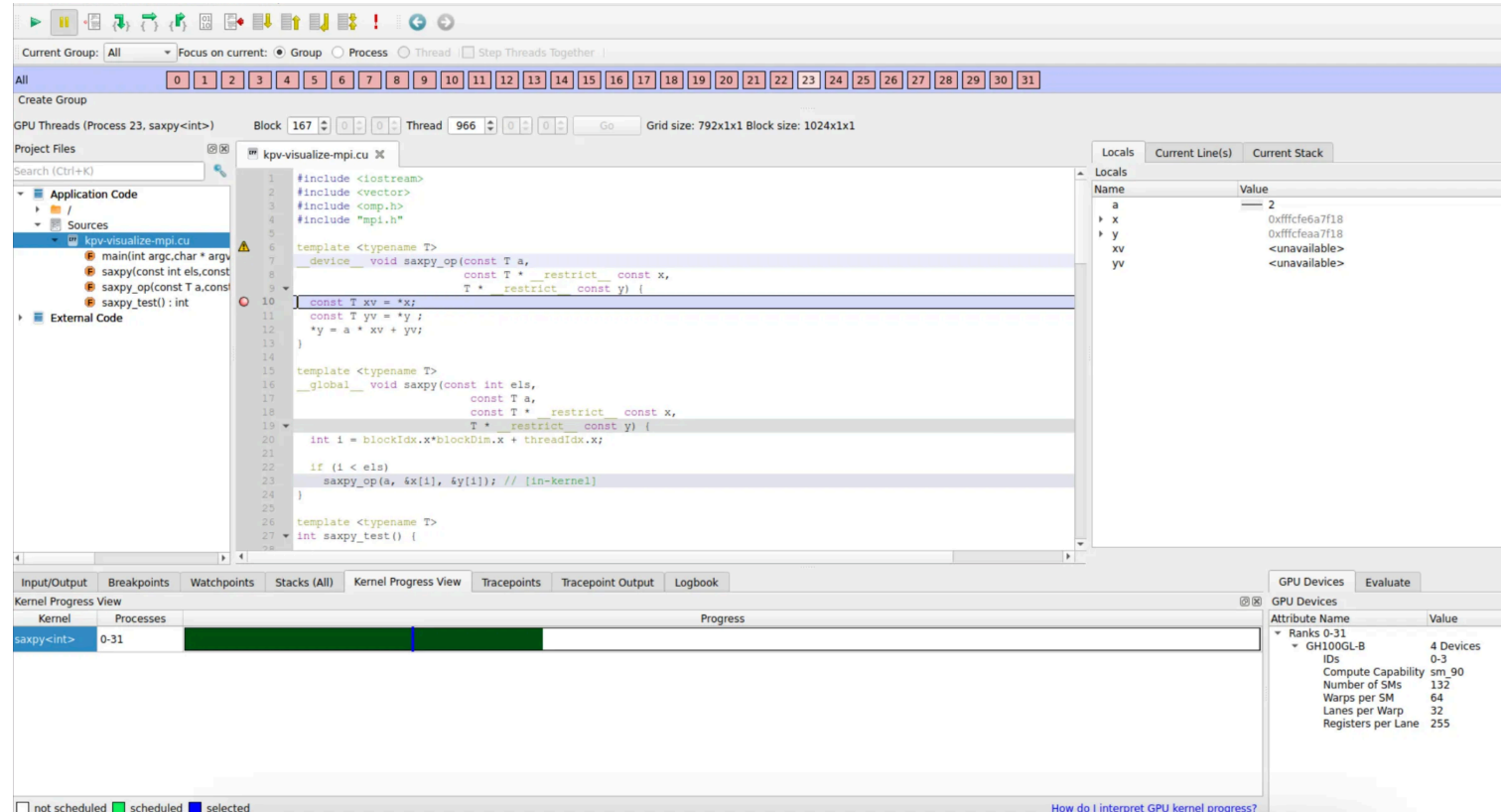
Controlling the GPU execution:

- All active threads in a warp will execute in lockstep  
Therefore, DDT will step 32 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

For NVIDIA's nvcc compiler, kernels must be compiled with the -g and -G flags





# Parallel Stack view

## Display location and number of threads

Input/Out... | Breakpo... | Watchpoi... | **Stacks (Proces...** | Kernel Progress V... | Tracepo... | Tracepoint Out... | Logb...

Stacks (Process 0)

Threads	GPU Thread	Function
1	39734	main::{lambda(auto:1&)#1}::operator()<sycl::_V1::handler>(sycl::_V1::handler&) co
1	7680	main::{lambda(auto:1&)#2}::operator()<sycl::_V1::handler>(sycl::_V1::handler&) cons
1	16	main::{lambda(auto:1&)#2}::operator()<sycl::_V1::handler>(sycl::_V1::handler&) cor
1	7664	<truncated>
1	7664	main::{/home/rshand/examples/matrix_mul_sycl.cpp:76 _V1::handler&) co
1	0	> main (mat

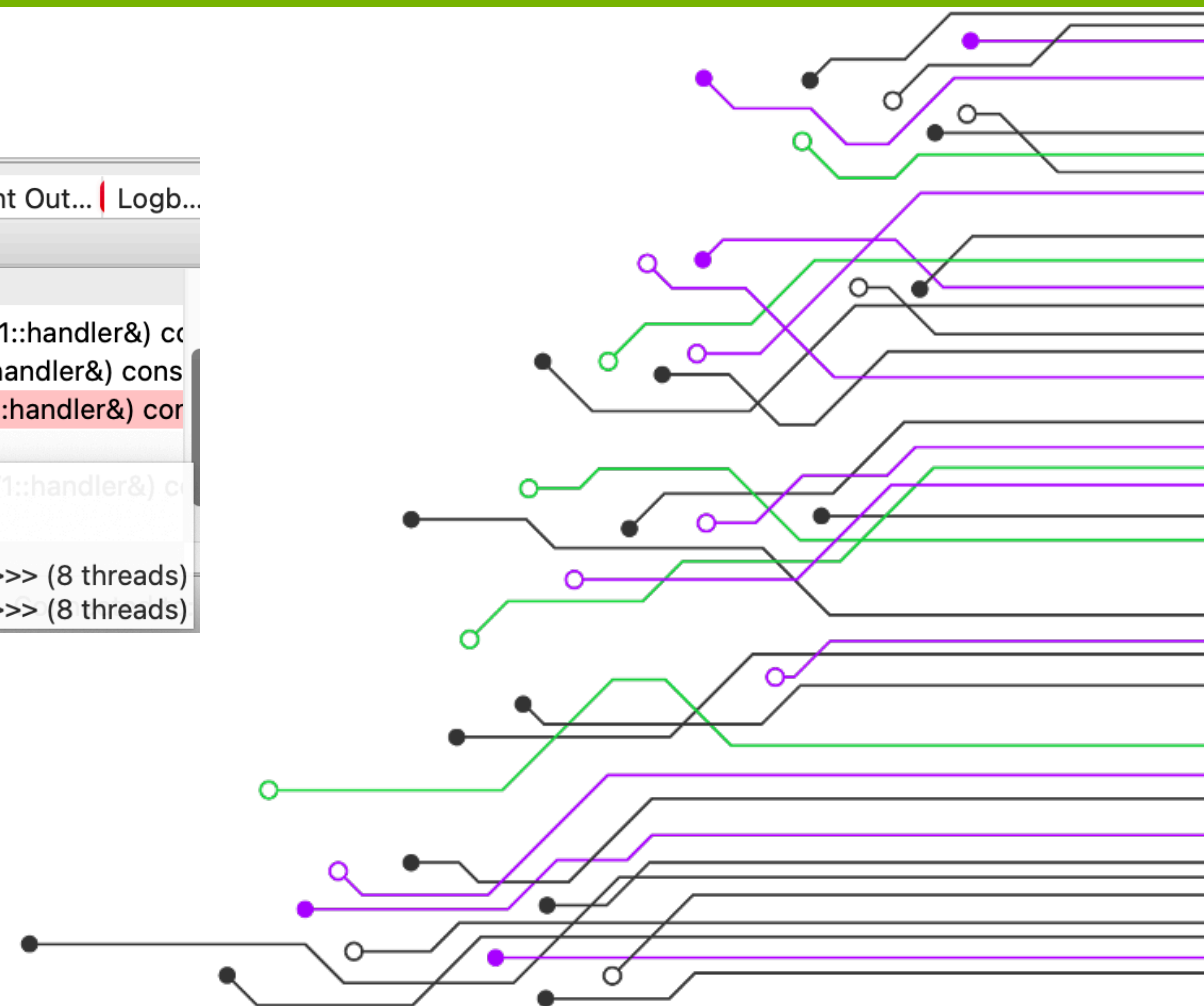
Kernel 2: 16 GPU threads  
<<<(0,0,0),(128,0,0)>>> ... <<<(0,0,0),(135,0,0)>>> (8 threads)  
<<<(0,0,0),(128,1,0)>>> ... <<<(0,0,0),(135,1,0)>>> (8 threads)

### Click stack item

- Select GPU Thread
- Update variable display
- Move Source Code Viewer

### Tooltip displays

- GPU Thread Ranges
- Size of each range





# Python Debugging

- Debug Features

- Sparklines for Python variables
- Tracepoints
- MDA viewer
- Mixed language support

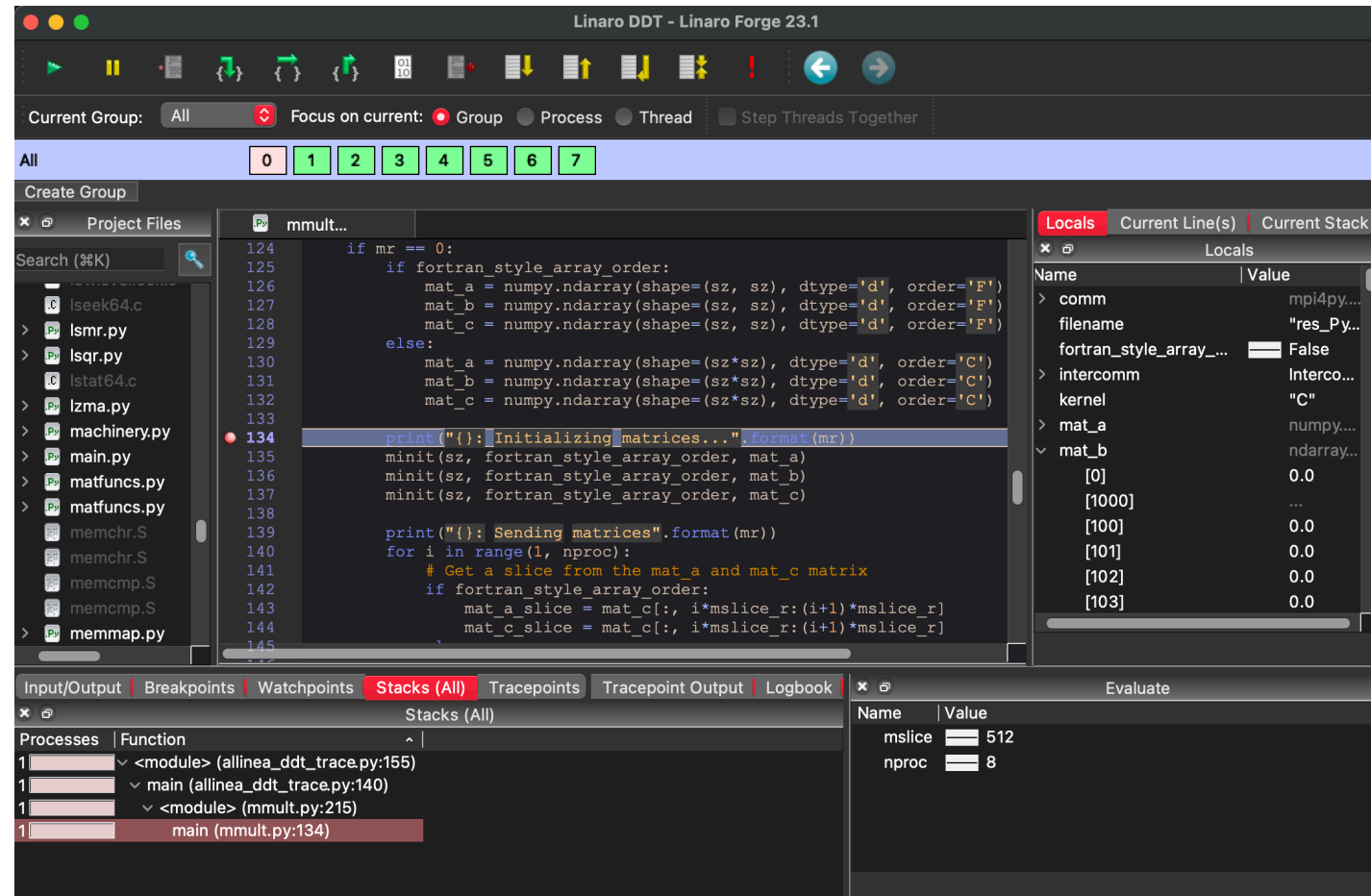
- Improved Evaluations:

- Matrix objects
- Array objects
- Pandas DataFrame
- Series objects

- Python Specific:

- Stop on uncaught Python exception
- Show F-string variables in “Current Line” display
- Mpi4py, NumPy, SciPy

ddt --connect mpirun -n 8 python3  
%allinea\_python\_debug% ./mmult.py





# 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:

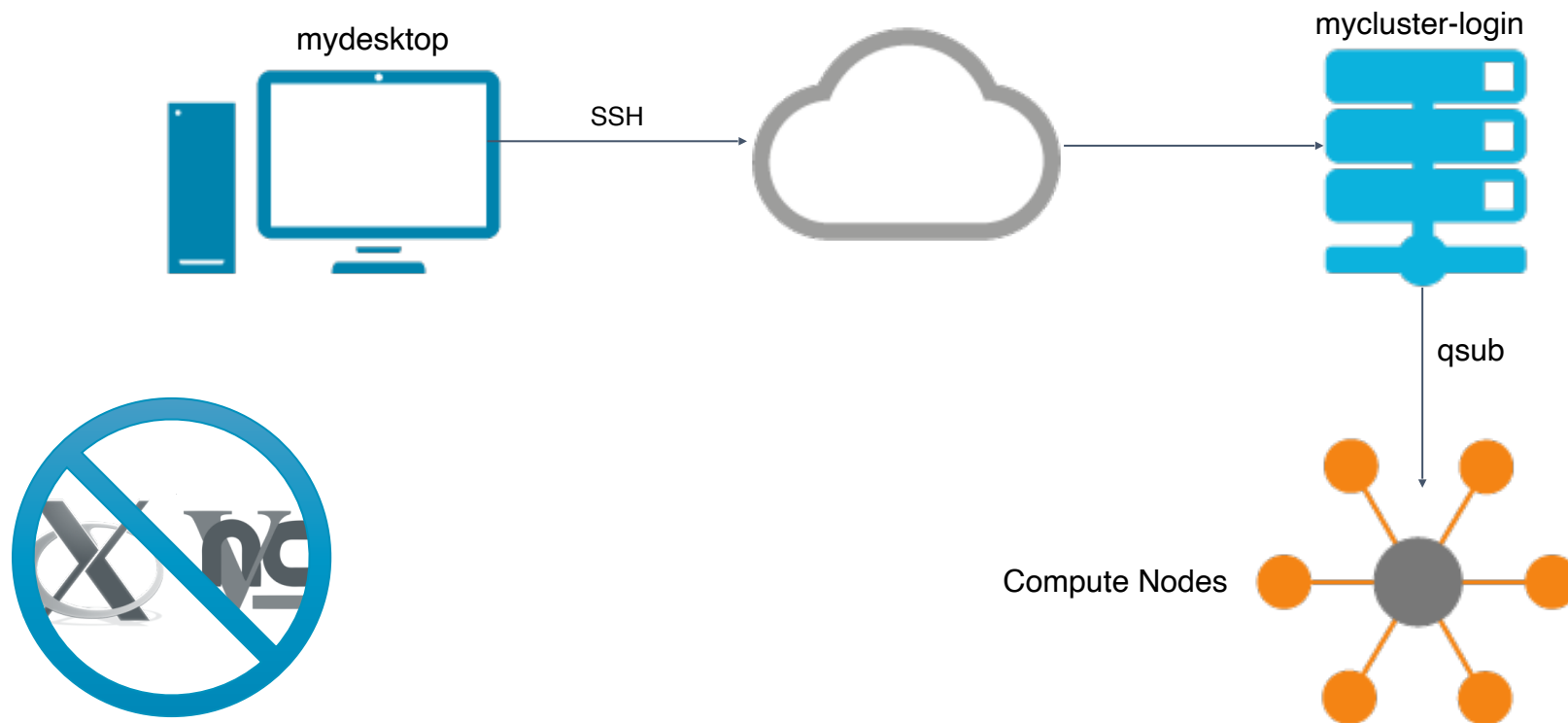
- `$ ddt --offline --output=report.html mpirun ./jacobi_omp_mpi_gnu.exe`
  - **--offline** enable non-interactive debugging
  - **--output** specifies the name and output of the non-interactive debugging session (HTML or Txt)
  - Add **--mem-debug** to enable memory debugging **and memory leak detection**

```
ddt --offline -o jacobi_omp_mpi_gnu_debug.txt \  
    --trace-at _jacobi.F90:83,residual \  
    mpirun ./jacobi_omp_mpi_gnu.exe
```



# The Forge GUI and where to run it

Forge provides a powerful GUIs that can be run in a variety of configurations





# Remote connection to Aurora

The image shows the Linaro Forge 23.1 application window. On the left is a sidebar with the Linaro Forge logo, Linaro DDT icon, Linaro MAP icon, and links for 'Get trial licence', 'Support', and 'linaroforge.com'. The main area contains sections for 'RUN', 'ATTACH', 'OPEN CORE', 'MANUAL LAUNCH (ADVANCED)', 'OPTIONS', and 'QUIT'. The 'Remote Launch:' option under 'OPTIONS' is highlighted with a red box, and its 'Configure...' button is also highlighted. Overlaid on top of the main window is a 'Remote Launch Settings' dialog box. This dialog contains the following fields and options:

- Connection Name:** aurora
- Host Name:** rshand@login.aurora.alcf.anl.gov (with a dropdown arrow and a link 'How do I connect via a gateway (multi-hop)?')
- Remote Installation Directory:** /lus/flare/projects/Tools/jkwack/Linaro/forge/25.0.1
- Remote Script:** Optional
- Private Key:** Optional (with a file selection icon)
- KeepAlive Packets:** ☐ Always look for source files locally
- Interval:** 30 seconds (with a slider and a dropdown arrow)
- ☒ Proxy through login node
- Buttons:** 'Test Remote Launch', 'Help', 'OK', and 'Cancel'.



# Cheat sheet

## ***Training material***

### *1. Getting the examples*

```
cp /flare/ATPESC2025/EXAMPLES/track6-tools/linaro-forge/linaro-forge-training.tar.gz  
tar -xf linaro-forge-training.tar.gz
```

### *2. Set the path to the forge training folder*

```
export FORGE_TRAINING=<path_to_training_folder>
```

## ***Forge Client (On local machine)***

Install Forge client <https://www.linaroforge.com/downloadForge>

## ***Running with a batch script***

```
qsub $FORGE_TRAINING/submit-polaris.sh
```

## ***Interactive Session***

```
qsub -l -l select=1 -l filesystems=home:flare -l  
walltime=1:00:00 -q ATPESC -A ATPESC2025
```

## ***Forge commands***

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

## ***Guides***

[Forge userguide](#)

[Debugging on Aurora](#)



# Debugging with DDT on Aurora

1. *build deadlock, simple, memory\_debugging and split examples*

```
cd $FORGE_TRAINING/correctness/debug  
make
```

2. *Get an interactive session*

3. *export FORGE\_LICENSE\_FILE=/pe/licenses/arm\_forge/Licence*

4. *split*

```
/lus/flare/projects/Tools/jkwack/Linaro/forg/25.0.1/bin/ddt --np=12 --connect --mpi=generic --mpiargs='--ppn 12 --envall' ./split
```

5. *deadlock*

```
/lus/flare/projects/Tools/jkwack/Linaro/forg/25.0.1/bin/ddt --np=12 --connect --mpi=generic --mpiargs='--ppn 12 --envall' ./deadlock
```

6. *Memory\_debugging*

```
/lus/flare/projects/Tools/jkwack/Linaro/forg/25.0.1/bin/ddt --np=1 --connect --mpi=generic --mpiargs='--ppn 1 --envall' ./memory_debugging
```

```
/lus/flare/projects/Tools/jkwack/Linaro/forg/25.0.1/bin/ddt --np=4 --connect --mpi=generic --mpiargs='--ppn 4 --envall' ./memory_debugging
```



# Debugging Intel Xe GPUs using DDT on Aurora

```
cd $FORGE_TRAINING/correctness/gpu-intel-mmult
```

```
mpicxx -fsycl -g -O0 matrix_mul_sycl.cpp -o matrixmul
```

```
qsub -l select=2 -l walltime=1:00:00 -l filesystems=home:flare -A ATPESC2025 -q ATPESC -I
```

```
./soft/compilers/oneapi/2025.1.0/debugger/2025.1/env/vars.sh
```

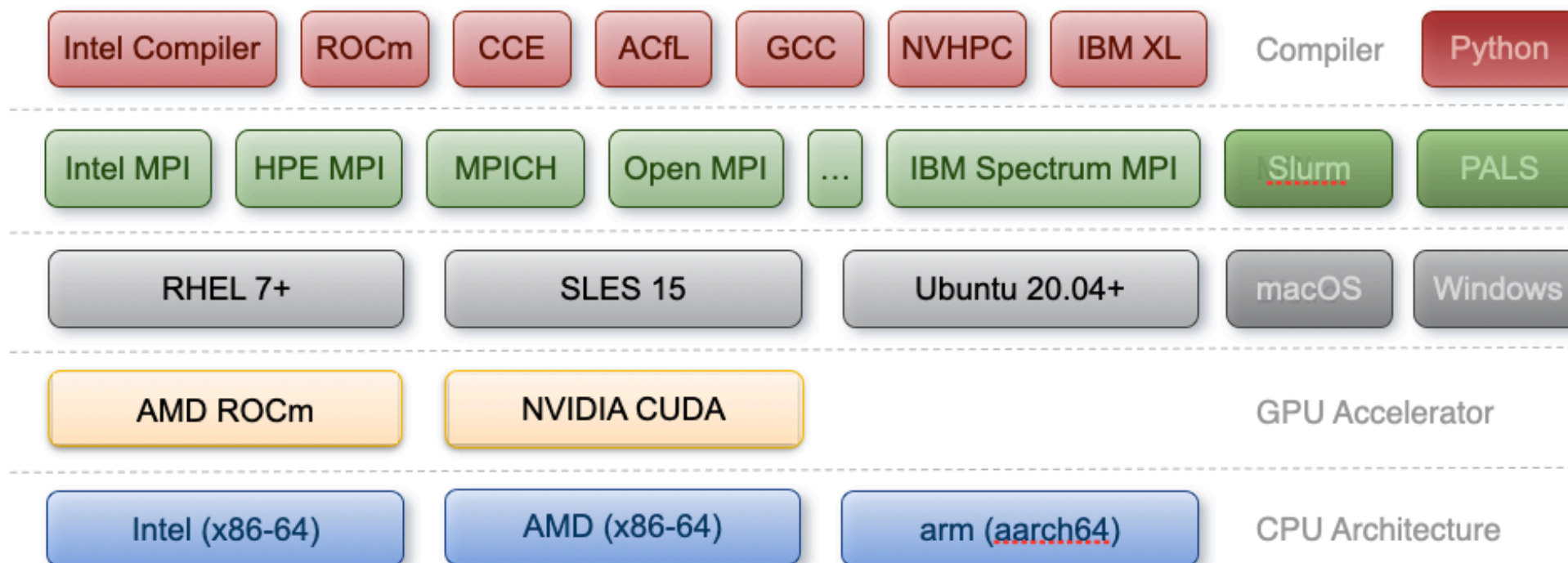
```
https://docs.alcf.anl.gov/aurora/debugging/ddt-aurora/#invoking-the-ddt-server-from-aurora
```

```
ddt --np=24 --connect --mpi=generic --mpiargs="--ppn 12 --envall" ./matrixmul
```



# MAP Supported platforms

Works across hardware architectures and HPC technologies





# 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)



# Linaro Performance Reports

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

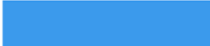
**arm**  
PERFORMANCE  
REPORTS

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%** | 

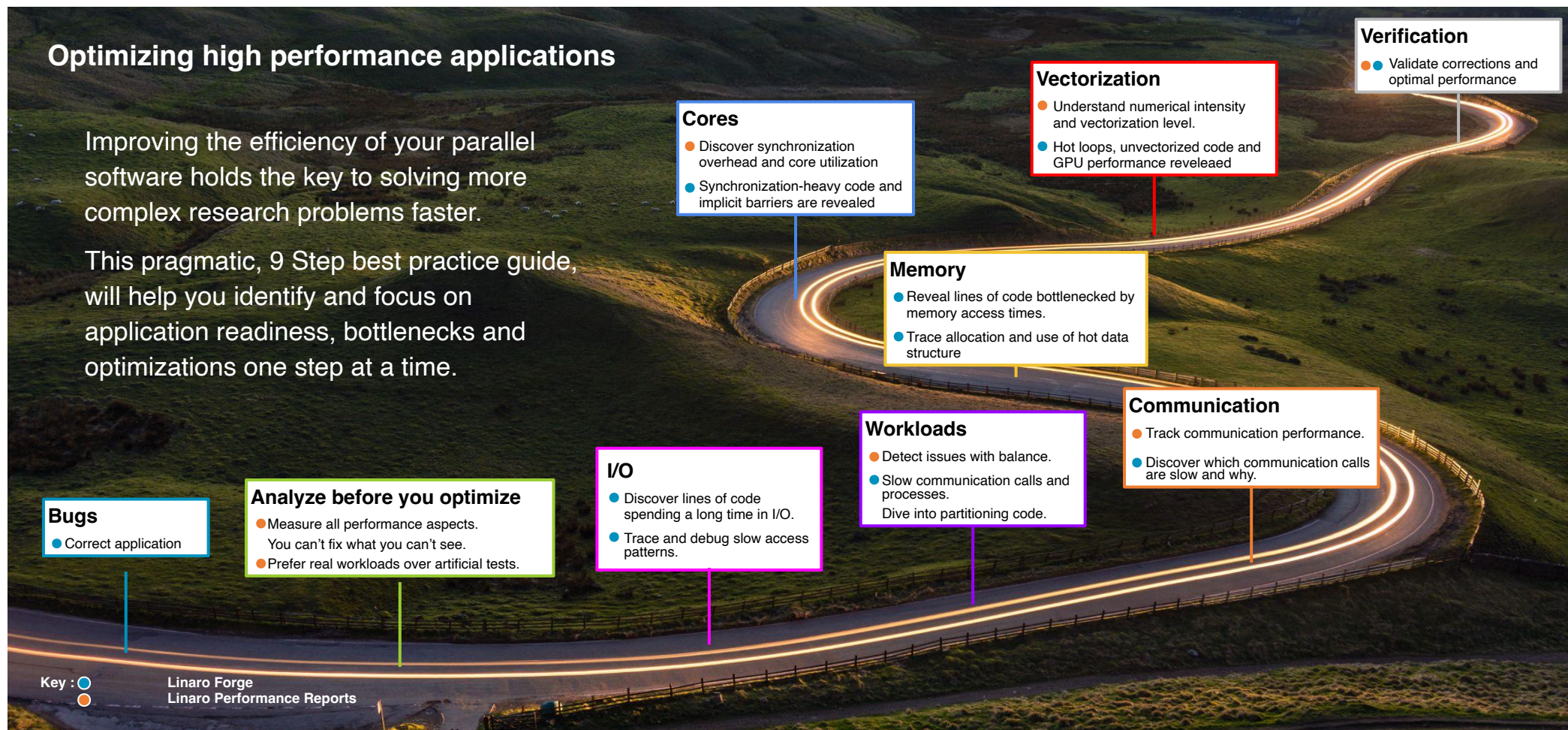
Effective process read rate **0.00 bytes/s** | 

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.



# Performance Roadmap

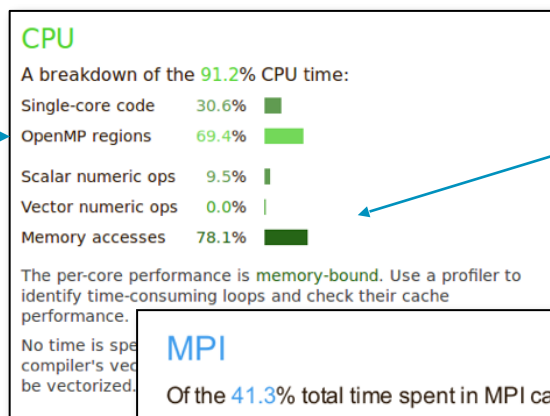




# Linaro Performance Reports Metrics

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

Multi-threaded  
parallelism



SIMD  
parallelism

## MPI

Of the 41.3% total time spent in MPI calls:

Time in collective calls	100.0%	■
Time in point-to-point calls	0.0%	■
Estimated collective rate	4.07 bytes/s	■
Estimated point-to-point rate	0 bytes/s	■

Load  
imbalance

## OpenMP

A breakdown of the 99.5% time in OpenMP regions:

Computation	58.9%	■
Synchronization	41.1%	■
Physical core utilization	100.0%	■
System load	99.7%	■

OMP  
efficiency

Significant time is spent **synchronizing** threads in parallel regions. Check the affected regions with a profiler.

This may be a sign of overly fine-grained parallelism (OpenMP regions in tight loops) or workload imbalance.

## I/O

A breakdown of how the 53.9% total I/O time was spent:

Time in reads

Time in writes

Estimated

Estimated

Most of the

transfer rate

inefficient

write calls

## Memory

Per-process memory usage may also affect scaling:

Mean process

Peak process

Peak node

The peak node

the total number

processes at

## Lustre

Lustre file operations (per node)

Mean write

Peak write

Mean file d

Mean meta

## Energy

A breakdown of how the 32.3 Wh was used:

CPU	61.9%	■
System	38.1%	■
Mean node power	94.1 W	■
Peak node power	98.0 W	■

Significant time is spent waiting for memory accesses. Reducing the **CPU** clock frequency could reduce overall energy usage.

System  
usage



# 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

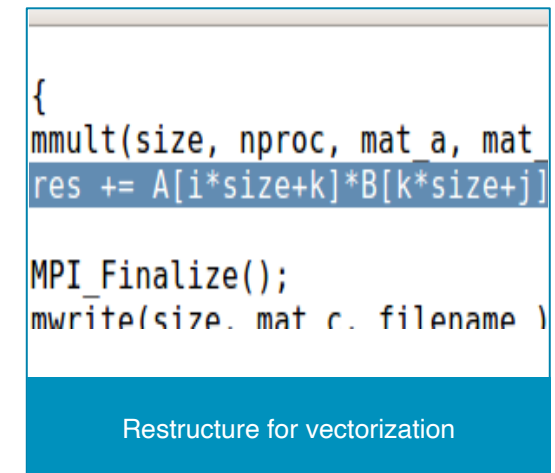
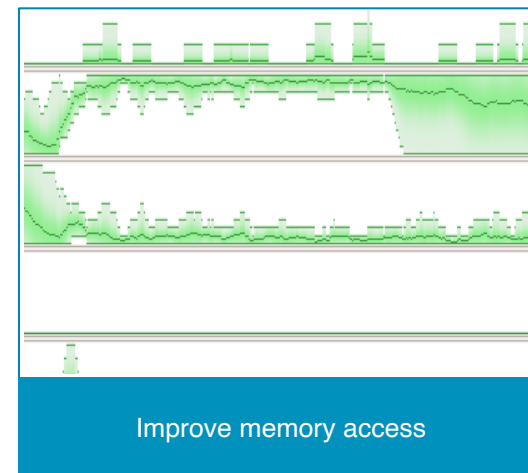
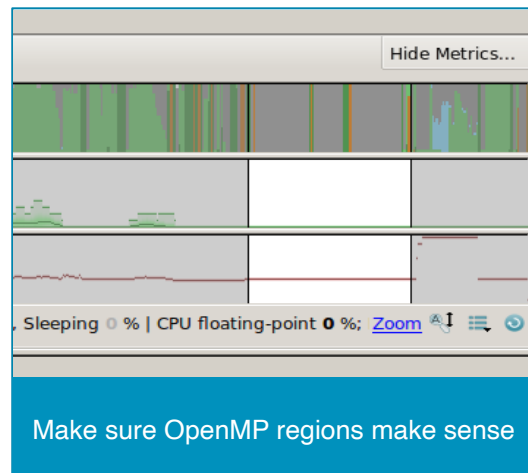
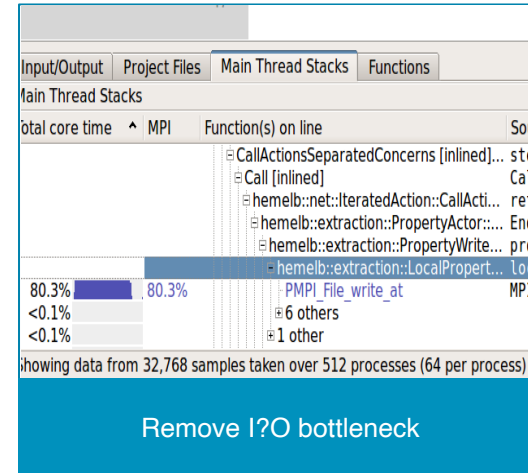
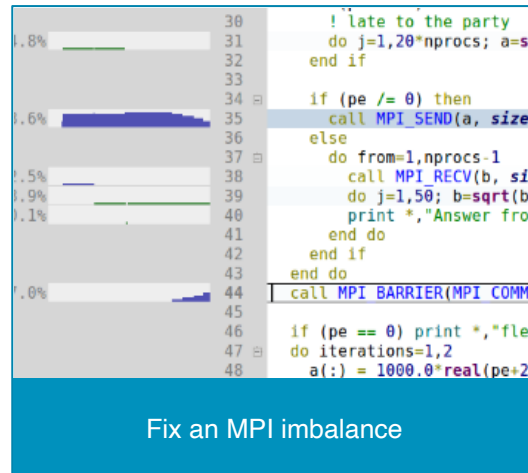
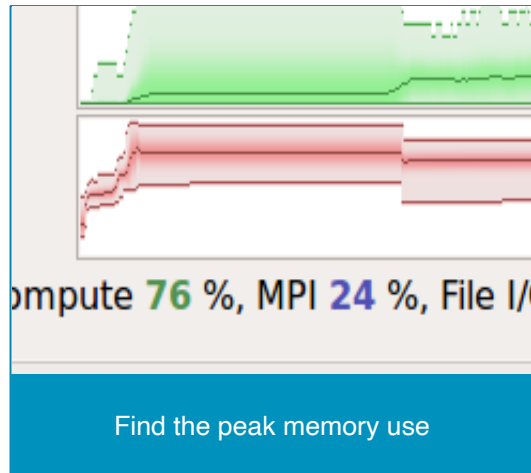
Adaptive sampling rate

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



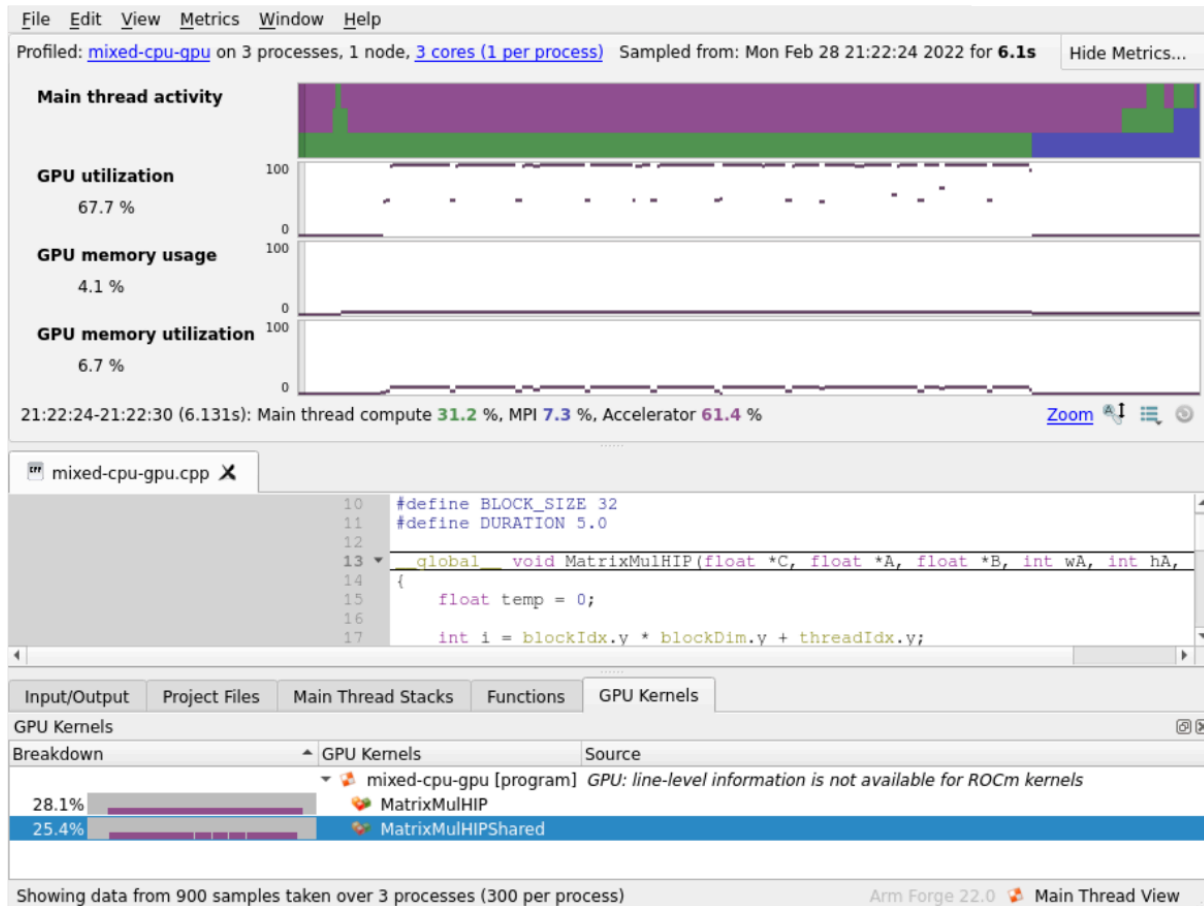


# MAP Highlights





# 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



# Python Profiling

19.0 adds support for Python

- Call stacks
- Time in interpreter

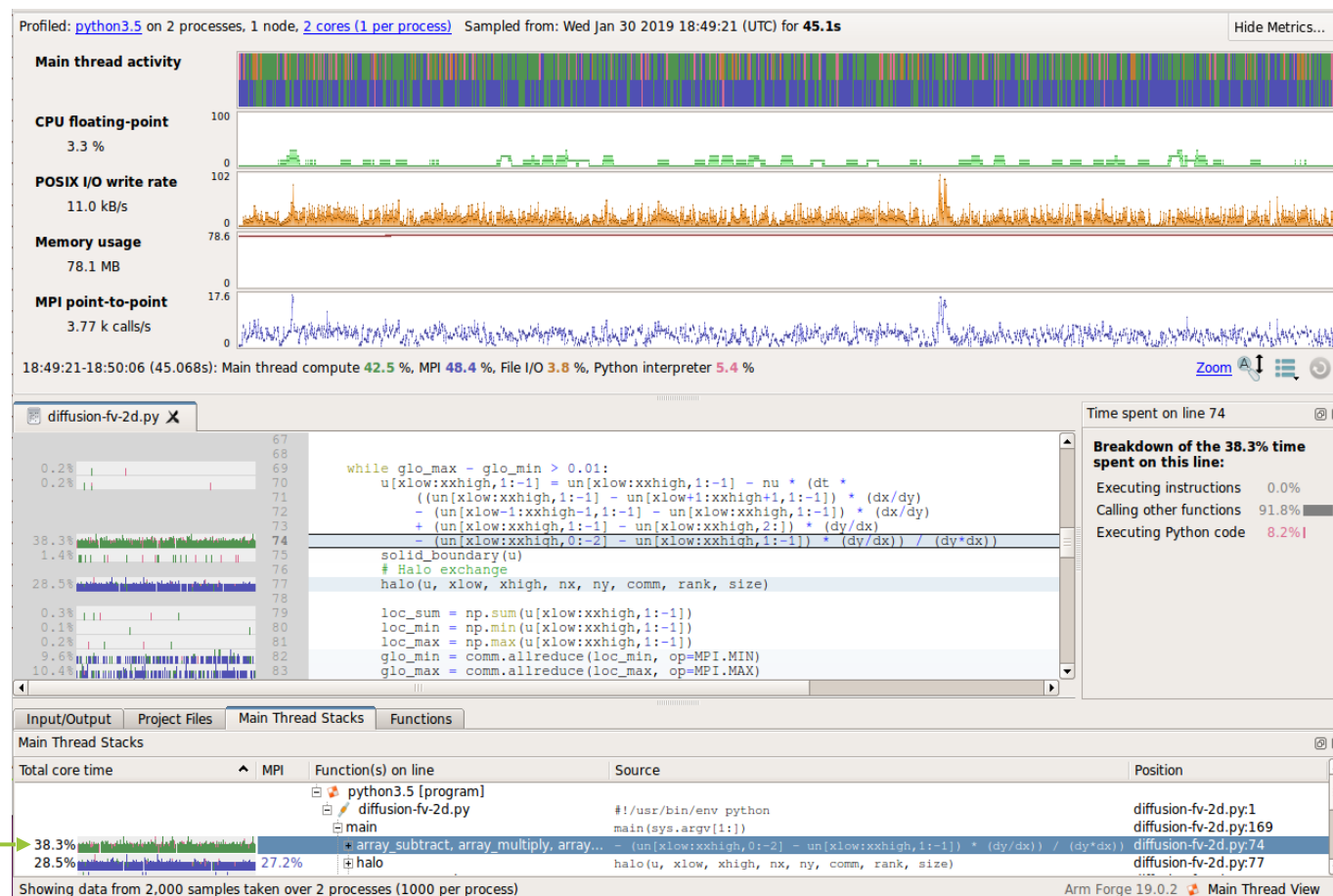
Works with MPI4PY

- 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 mpirun -n 2 python ./diffusion-fv-2d.py



# Compiler Remarks

## Annotates source code with compiler remarks

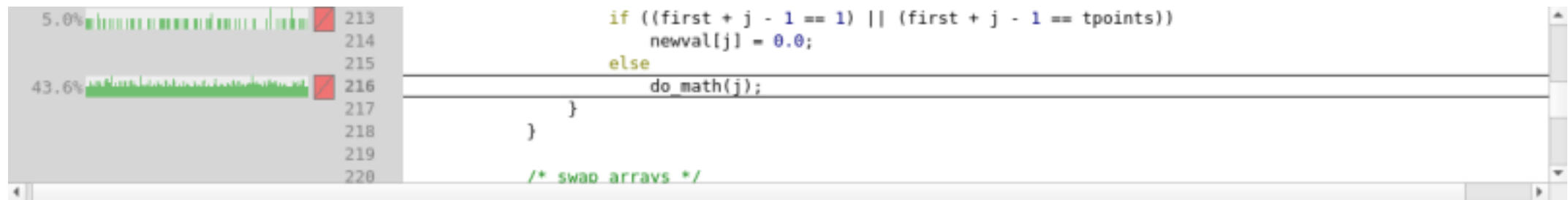
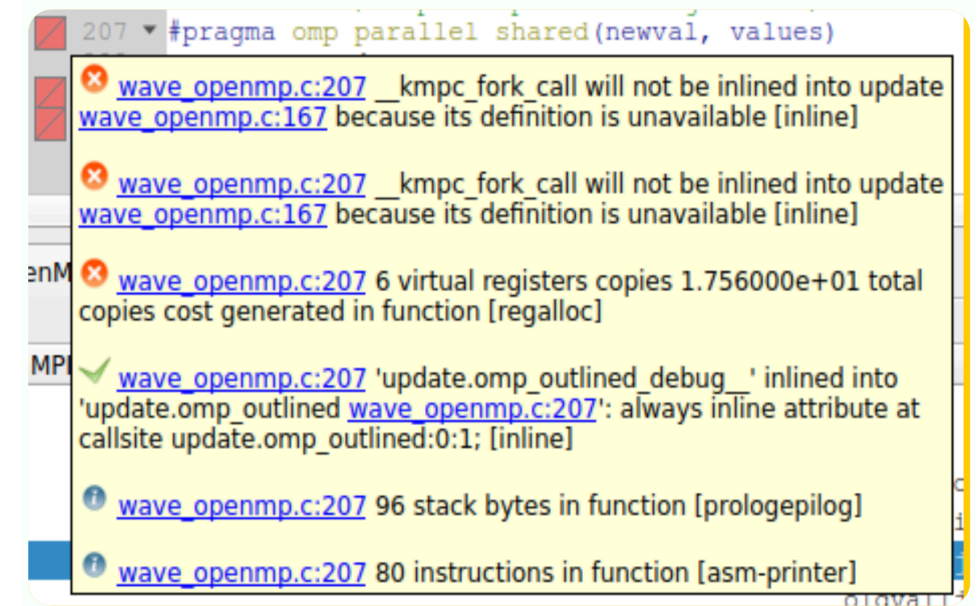
- Remarks are extracted from the compiler optimisation report
- Compiler remarks are displayed as annotations next to your source code

## Colour coded

- Their colour indicates the type of remark present in the following priority order:
  1. Red: failed or missed optimisations
  2. Green: successful or passed optimisations
  3. White: information or analysis notes

## Compiler Remarks menu.

- Specify build directories for non-trivial build systems
- Filter out remarks





# MAP Thread Affinity Advisor

**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.

Launch Command: `srun -n 16 python3 /global/homes/r/rshand/linaro-forge-training/performance/mmult.py -s 3072`

Process Command: `Select an individual process`

Global (launcher) environment variables:

OpenMP

Submission Script

Other

SLURM\_CPUS\_PER\_TASK

16

SLURM\_NPROCS

16

SLURM\_NTASKS

16

SLURM\_NTASKS\_PER\_NODE

16

SLURM\_TRES\_PER\_TASK

cpu:16

Exemplar node's topology (shading shows process affinity bindings):

Machine

Package

NUMANode #0

L3Cache

L2Cache

L2Cache

L2Cache

L2Cache

L2Cache

L2Cache

L2Cache

L1Cache

L1Cache

L1Cache

L1Cache

L1Cache

L1Cache

L1Cache

L1Cache

Core

000

001

002

003

004

005

006

007

008

009

010

011

012

013

014

015

L3Cache

L2Cache

L2Cache

L2Cache

L2Cache

L2Cache

L2Cache

L2Cache

L2Cache

L1Cache

L1Cache

L1Cache

L1Cache

L1Cache

L1Cache

L1Cache

L1Cache

Core

016

017

018

019

020

021

022

023

024

025

026

027

028

029

030

031

NUMANode #1

L3Cache

L2Cache

L2Cache

L2Cache

L2Cache

L2Cache

L2Cache

L2Cache

L2Cache

L1Cache

L1Cache

L1Cache

L1Cache

L1Cache

L1Cache

L1Cache

L1Cache

Core

032

033

034

035

036

037

038

039

040

041

042

043

044

045

046

047

L3Cache

L2Cache

L2Cache

L2Cache

L2Cache

L2Cache

L2Cache

L2Cache

L2Cache

L1Cache

L1Cache

L1Cache

L1Cache

L1Cache

L1Cache

L1Cache

L1Cache

Core

048

049

050

051

052

053

054

055

056

057

058

059

060

061

062

063

NUMANode #2

(multiple items selected)

Data taken at: Finalization

Available exemplar nodes:

nid004343 (0 similar nodes)

Processes on exemplar node:

Rank 0 (PID 1166384)

Rank 1 (PID 1166385)

Rank 2 (PID 1166387)

Rank 3 (PID 1166389)

Rank 4 (PID 1166391)

Rank 5 (PID 1166393)

Threads in selected processes:

☒ pthread (LWP 1167177) 000-0

☒ pthread (LWP 1166919) 000-0

☒ Main thread (LWP 1166384) 000-0

☒ pthread (LWP 1167181) 032-0

☒ pthread (LWP 1166929) 032-0

☒ Main thread (LWP 1166391) 032-0

Commentary:

[ERROR]

nid004343, ranks 0-15 (processes 1166384-1166385,1166387,1166389,1166393-1166394,1166397,1166399,1166401,1166403,1166405,1166407,1166409,1166411,1166413) contain at least one compute thread which has an overlapping thread affinity mask with another compute thread, e.g. threads 1166391 and 1166929.

[INFORMATION]

nid004343, number of threads allocated to node may be less than ideal. 48 are currently allocated, but consider using 128 (1 per core) for improved utilization.

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

**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.

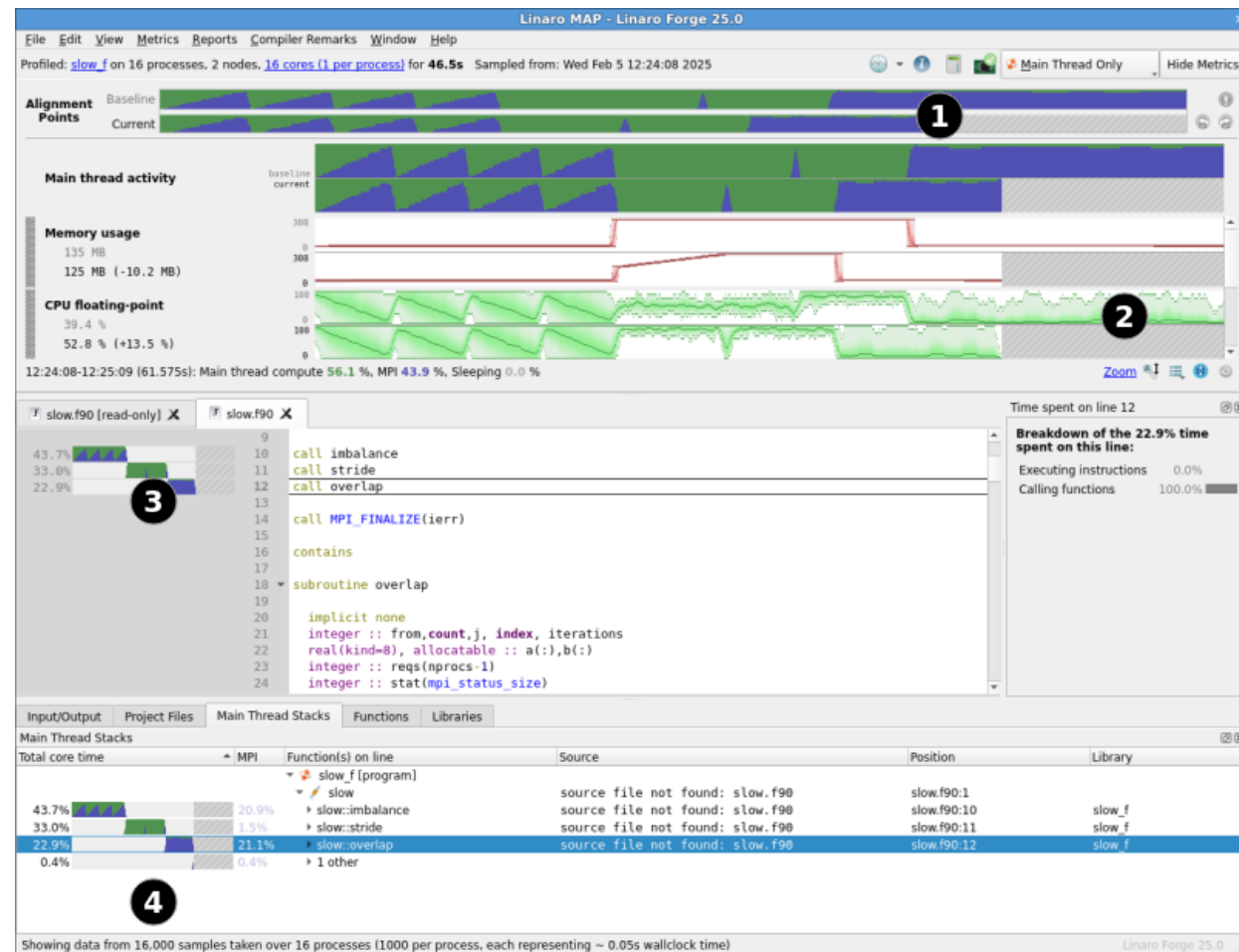
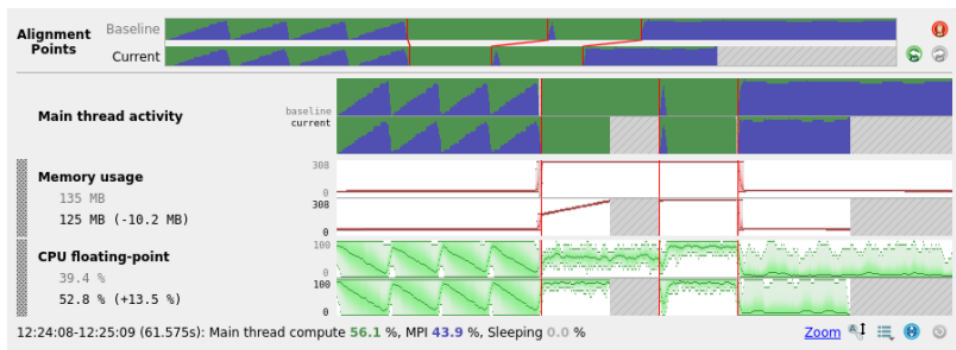
**Commentary**  
A list of commentary, providing information and advice on Memory Imbalance, Core Utilization etc.



# Differences between two profiles

## MAP Diff (--baseline support)

- MAP Diff allows comparisons of two MAP profiles, useful for identifying performance changes between different parameters, compilers, libraries and systems.
- Use the alignment points view (1) to line up phases of execution.
- Compare metric graphs of the two profiles (2), including metric summaries for each.
- See gaps in activity in the source code viewer (3) and stacks views (4), including OpenMP Regions View, Functions View, Library view and GPU Kernels/Memory Transfer View.





# Profiling with MAP on Polaris

Worked Example: [https://docs.linaroforge.com/23.1.1/html/forgeworked\\_examples\\_appendix/mmult/analyze.html](https://docs.linaroforge.com/23.1.1/html/forgeworked_examples_appendix/mmult/analyze.html)

## 1. Setup the environment

```
qsub -I -l select=1 -l filesystems=home:eagle -l walltime=0:30:00 -q ATPESC -A ATPESC2025
module use /soft/modulefiles
module load forge cray-cti
module load conda/2024-04-29
conda activate base
```

## 2. Build the Python example

```
cd $FORGE_TRAINING/performance
make -f mmult_py.makefile
```

## 3. Run the Python example

```
MPICH_GPU_SUPPORT_ENABLED=0 map --profile mpiexec -n 8 python ./mmult.py -s 3072
```

## 4. Offline profile

```
qsub $FORGE_TRAINING/scripts/submit-polaris.sh
```





## ARGONNE TRAINING PROGRAM ON EXTREME-SCALE COMPUTING

Produced by Argonne National Laboratory, a U.S. Department of Energy Laboratory managed by UChicagoArgonne, LLC under contract DE-AC02-06CH11357.

Special thanks to the National Energy Research Scientific Computing Center (NERSC) and Oak Ridge Leadership Computing Facility (OLCF) for the use of their resources during the training event.

The U.S. Government retains for itself and others acting on its behalf a nonexclusive, royalty-free license in this video, with the rights to reproduce, to prepare derivative works, and to display publicly.