October 29-31, 2024

ALCF Hands-on HPC Workshop

HPC Development Solutions from Linaro

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

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

DDT Supported Platforms

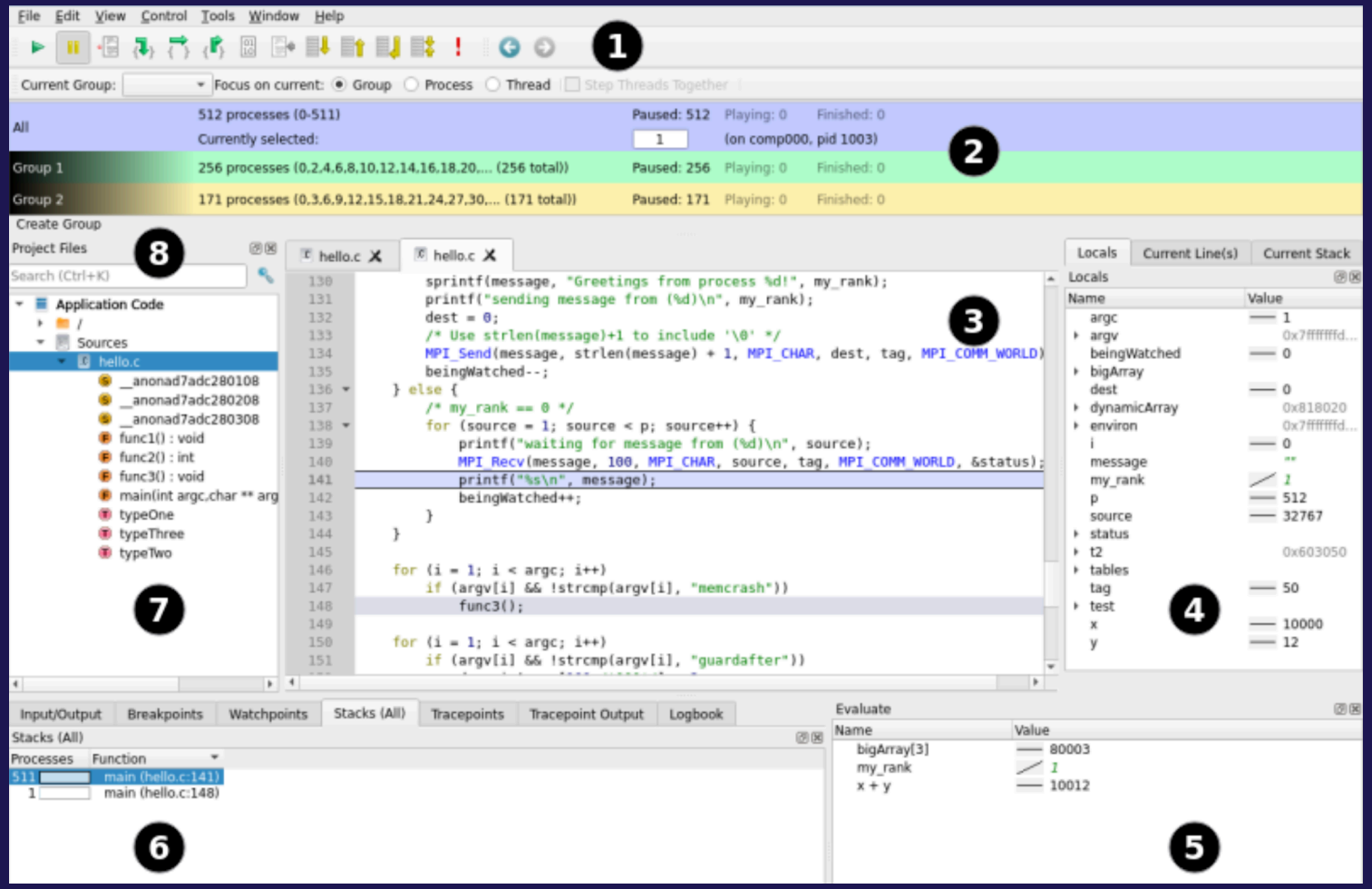
Works across hardware architectures and HPC technologies

Intel Compiler	ROCm	CCE	ACfL	GCC	NVHPC	IBM XL	Compiler	Python
Intel MPI	HPE MPI	MPICH	Open MPI	…	IBM Spectrum MPI	Slurm		PALS
RHEL 7+	SLES 15	Ubuntu 20.04+	macOS	Windows				
AMD ROCm	NVIDIA CUDA	Intel Xe-HPC	GPU Accelerator					
Intel (x86-64)	AMD (x86-64)	arm (aarch64)	CPU Architecture					

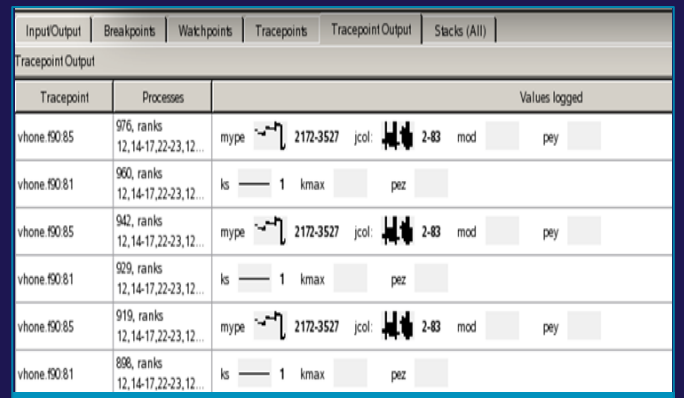
DDT UI

Intuitive and scalable user interface

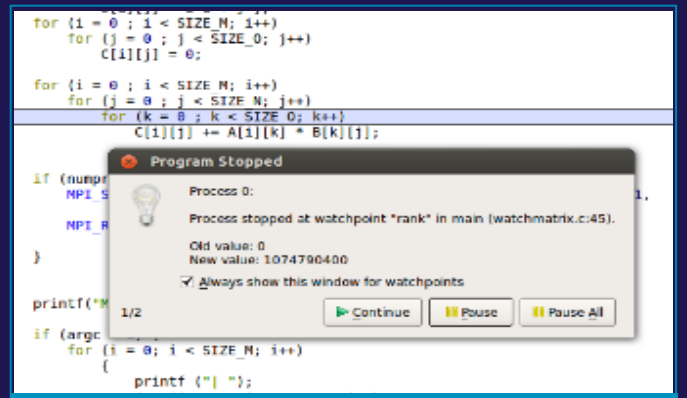
- 1 Process controls
- 2 Process groups
- 3 Source Code view
- 4 Variables
- 5 Evaluate window
- 6 Parallel Stack
- 7 Project files
- 8 Find a file or function



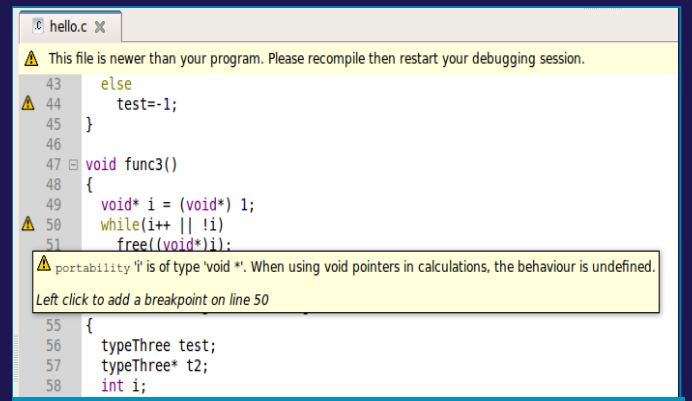
Linaro DDT Debugger Highlights



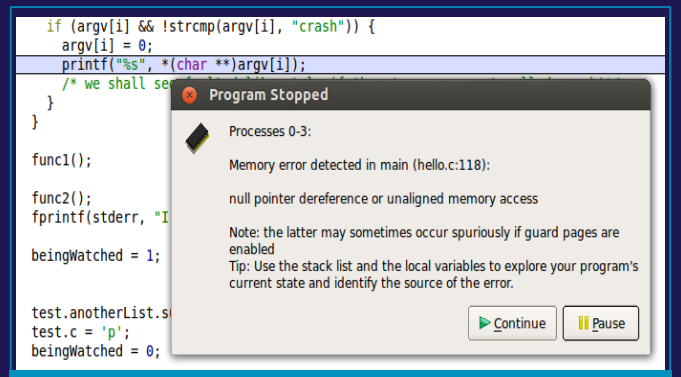
The scalable print alternative



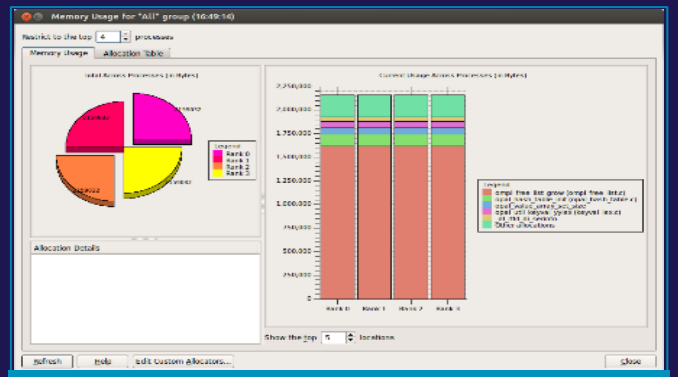
Stop on variable change



Static analysis warnings on code errors

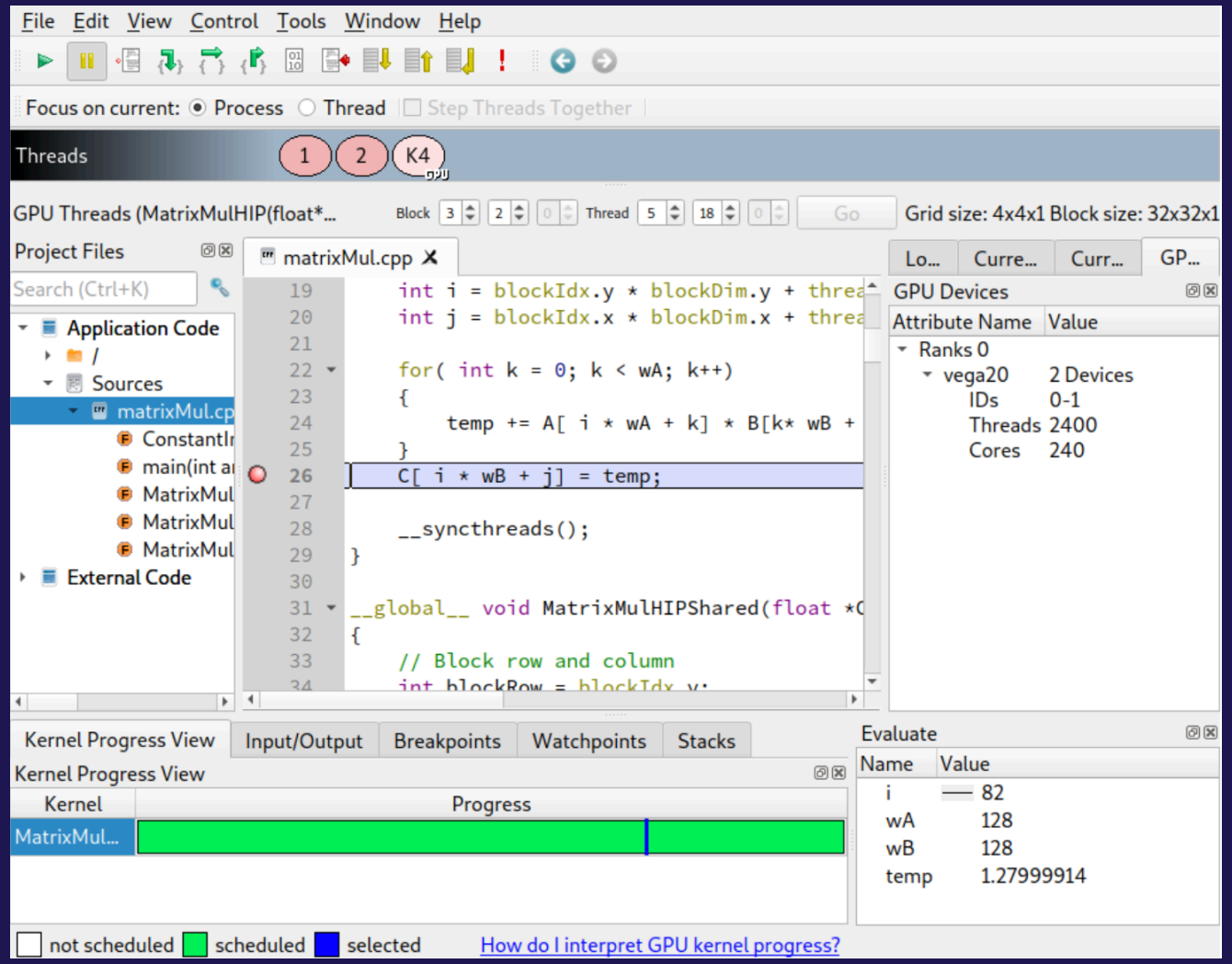


Detect read/write beyond array bounds



Detect stale memory allocations

GPU Debugging



- Support AMD, Intel and Nvidia GPUs
- Debug simultaneously on GPU and CPU

- Look and feel exactly the same
- Main Features work in GPU

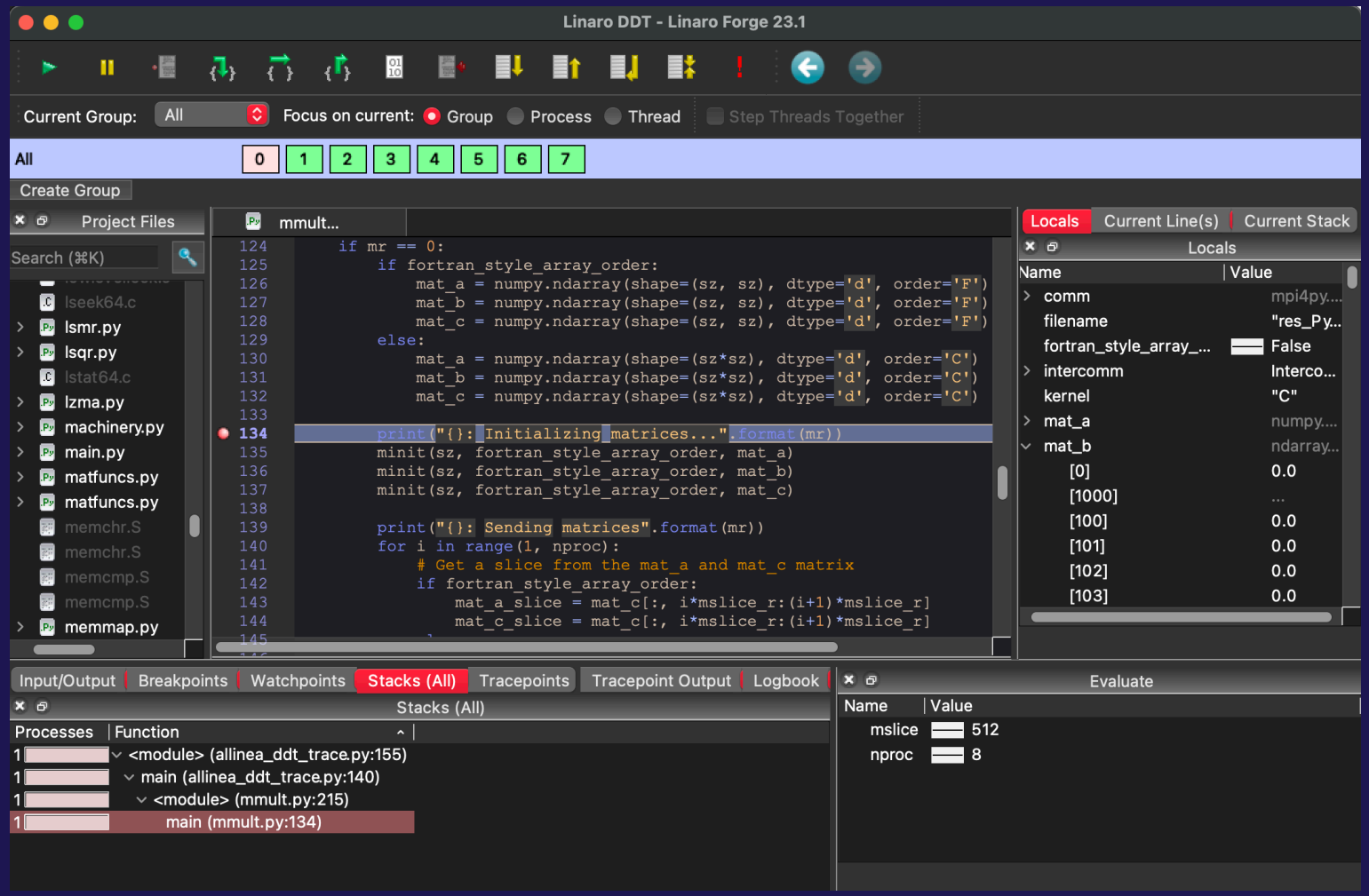
- Key (additional) GPU features:
  - Kernel Progress View
  - GPU thread in parallel stack view
  - GPU Thread Selector
  - GPU Device Pane

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

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
```



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…

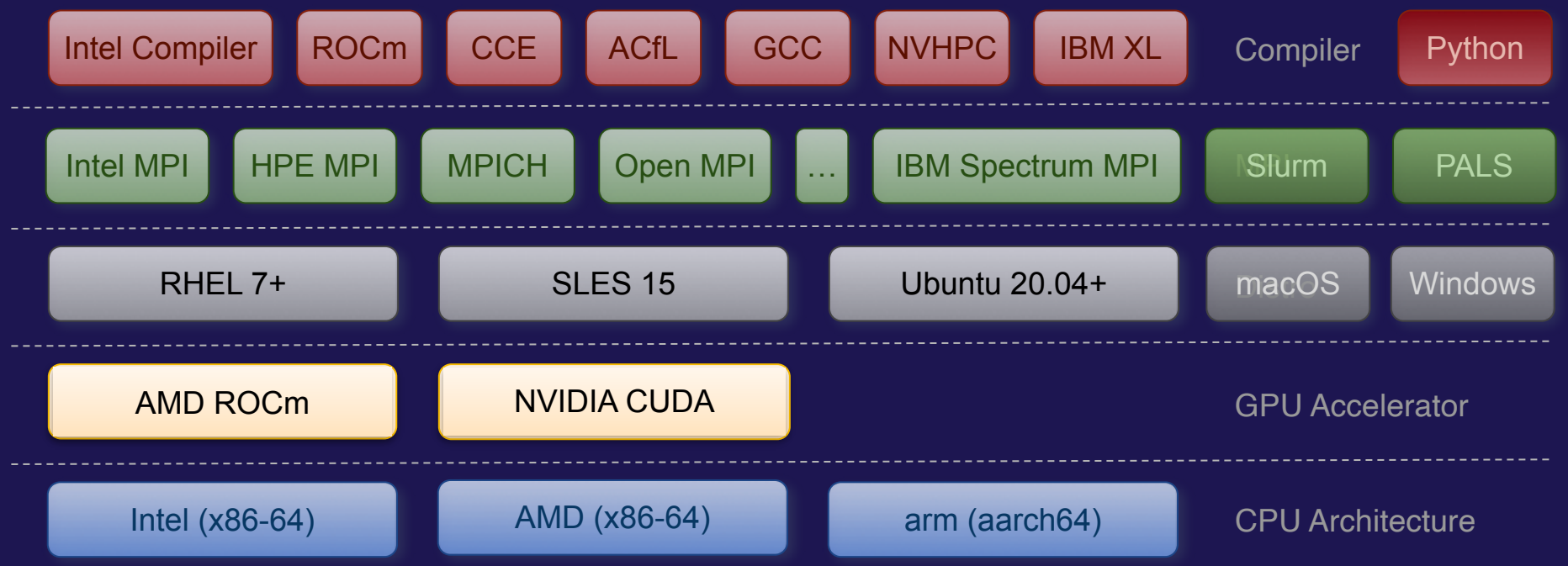
To do so, use following arguments:

- $ 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 \
                                mpiexec ./jacobi_omp_mpi_gnu.exe
```

MAP and Performance Reports Supported Platforms

Works across hardware architectures and HPC technologies



Linaro Performance tools

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

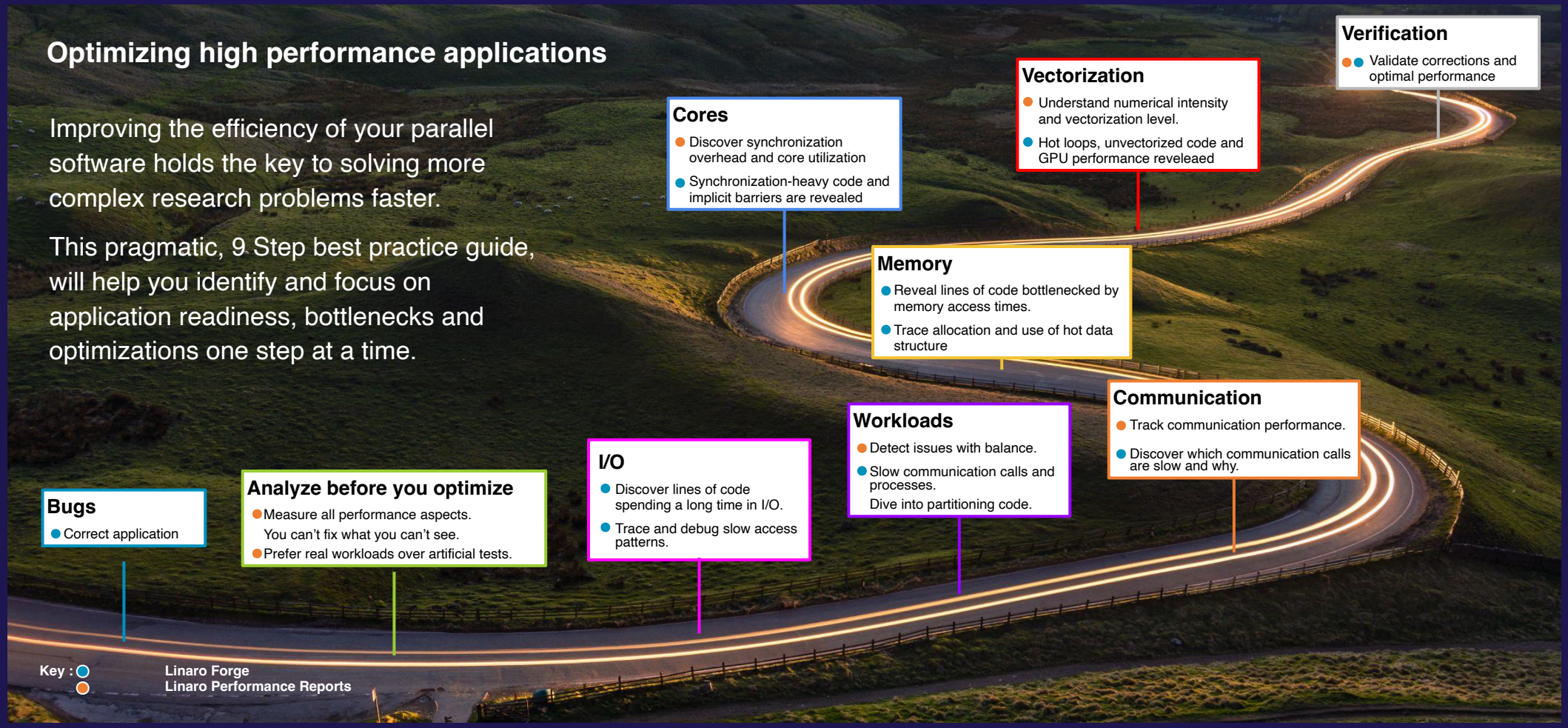
- Analyses data and reports the information that matters to users
- Provides simple guidance to help improve workloads' efficiency

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)

9 Step guide for optimising code



Linaro Performance Reports

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

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.

Linaro Performance Reports Metrics

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

Multi-threaded parallelism

SIMD parallelism

Load imbalance

OMP efficiency

System usage

CPU

A breakdown of the 91.2% CPU time:

Single-core code	30.6%
OpenMP regions	69.4%
Scalar numeric ops	9.5%
Vector numeric ops	0.0%
Memory accesses	78.1%

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

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

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%

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:

Memory

Per-process memory usage may also affect scaling:

Lustre

Lustre file operations (per node)

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.

MAP Capabilities

MAP is a sampling based scalable profiler

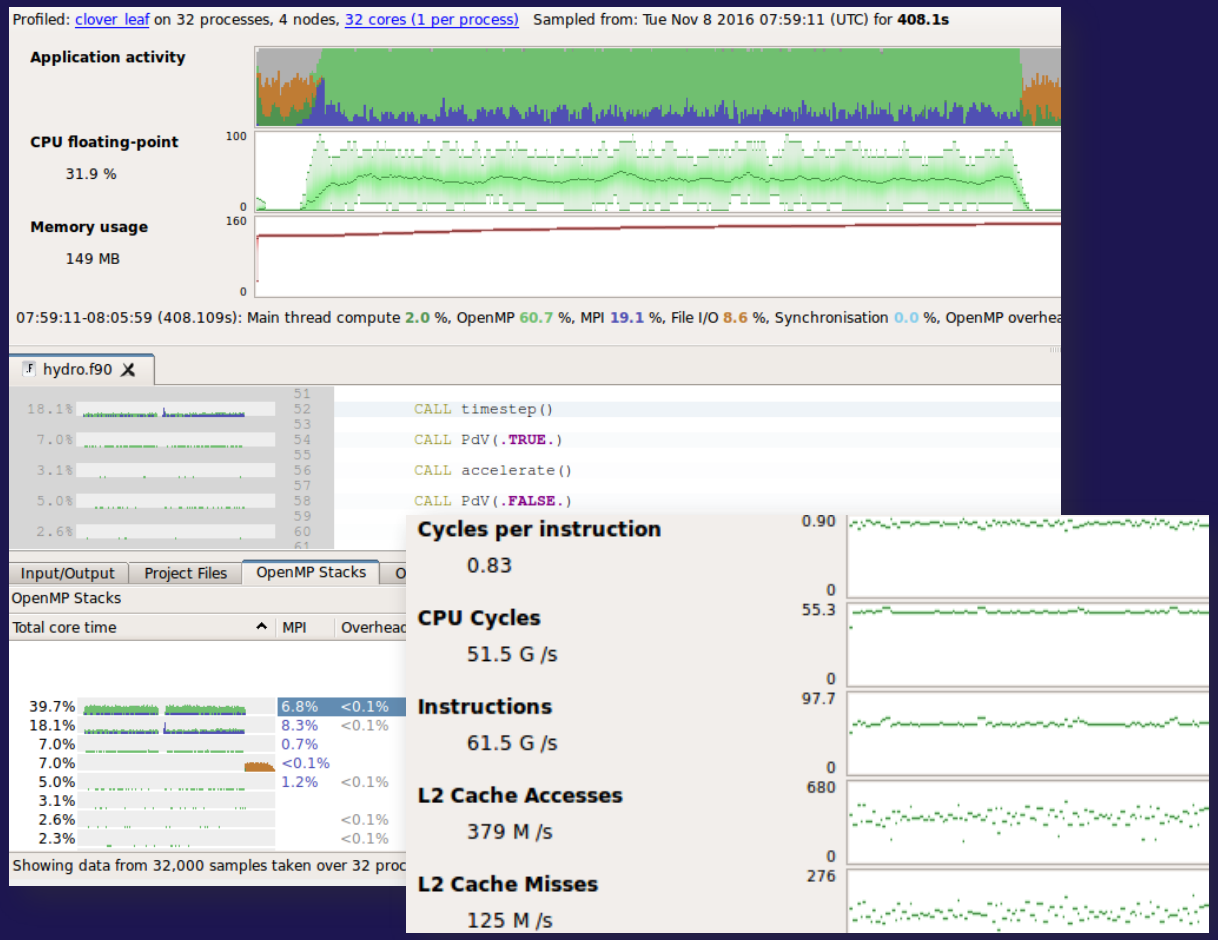
- Built on same framework as DDT
- Parallel support for MPI, OpenMP, CUDA
- Designed for C/C++/Fortran

Designed for ‘hot-spot’ analysis

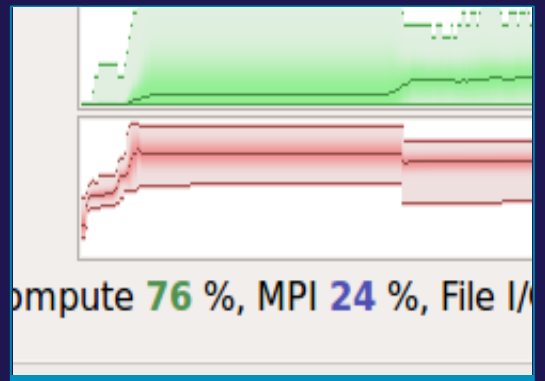
- Stack traces
- Augmented with performance metrics

Adaptive sampling rate

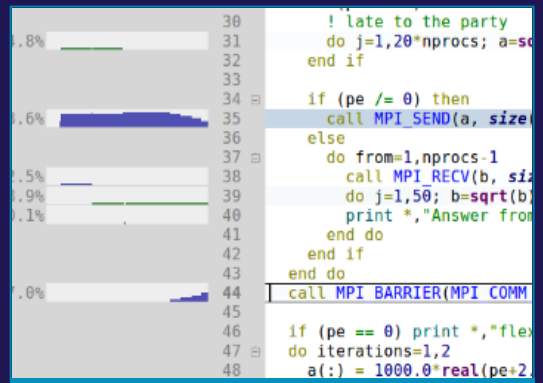
- Throws data away - 1,000 samples per process
- Low overhead, scalable and small file size



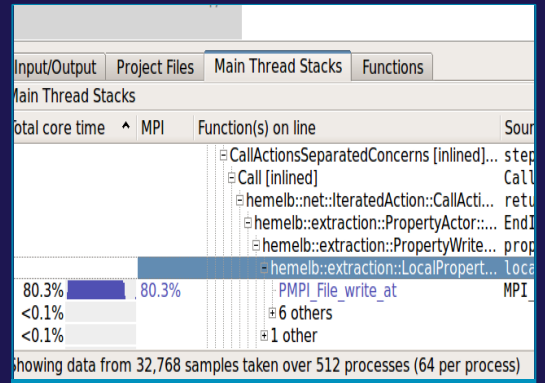
MAP Highlights



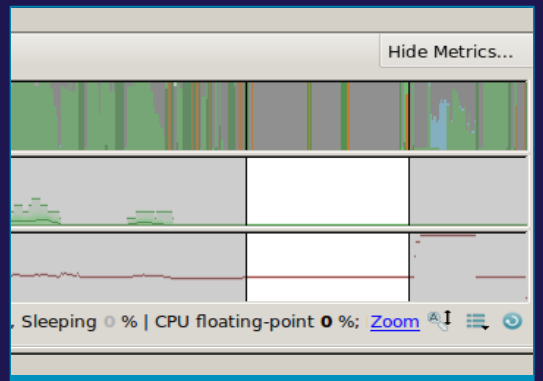
Find the peak memory use



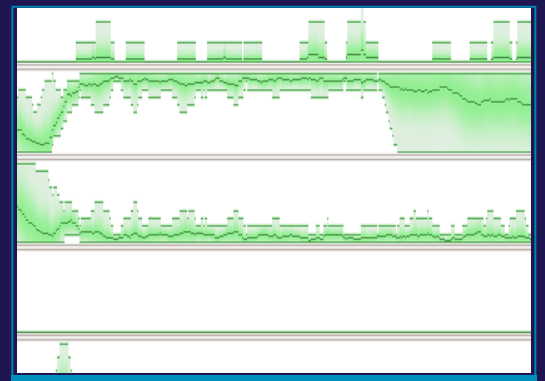
Fix an MPI imbalance



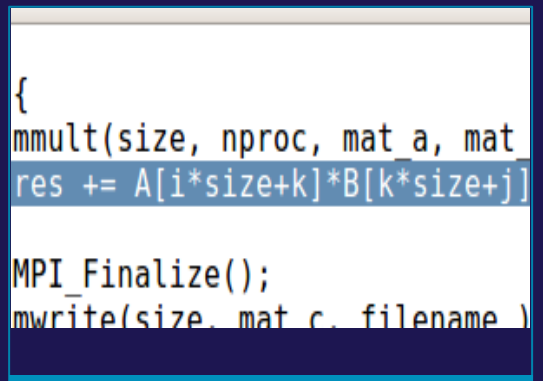
Remove I?O bottleneck



Make sure OpenMP regions make sense

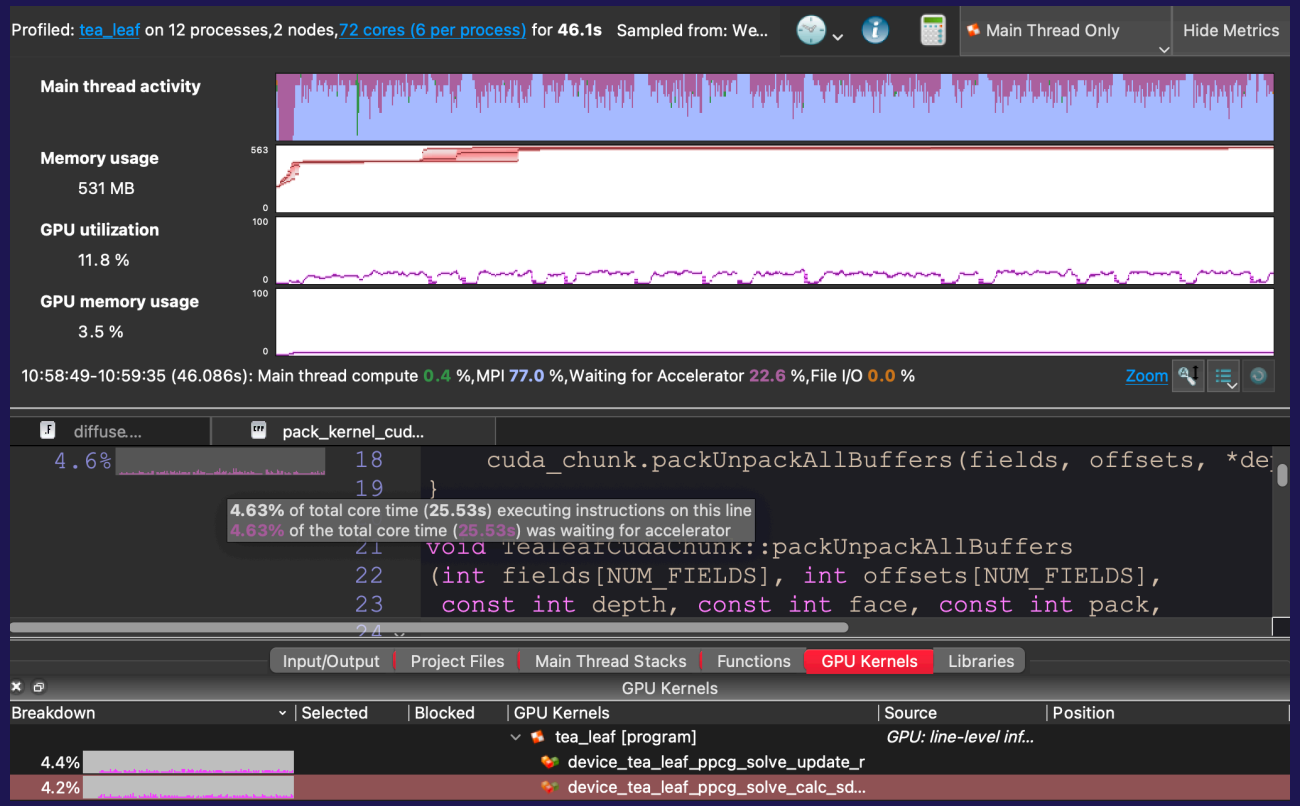


Improve memory access



Restructure for vectorization

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

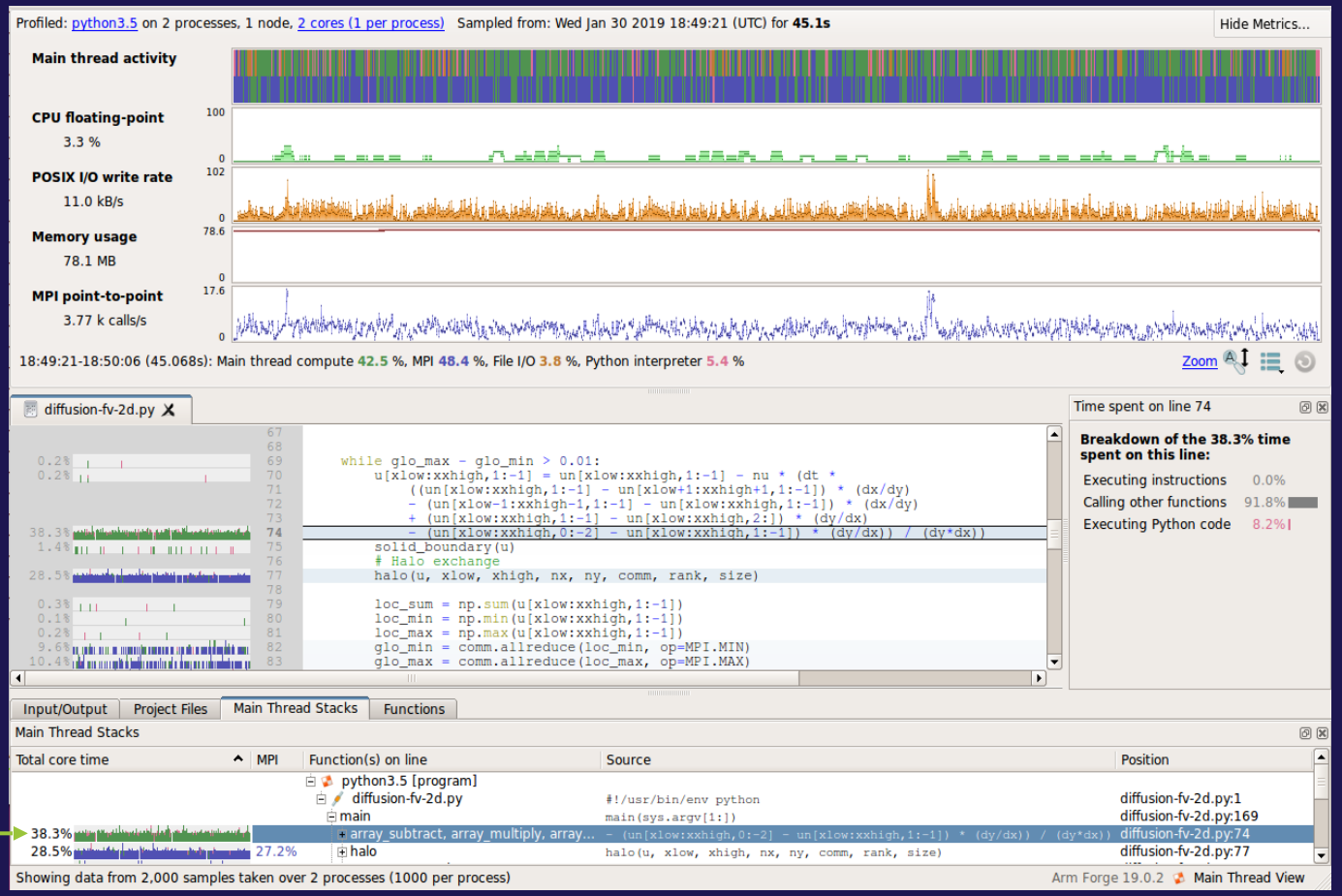
- Call stacks
- Time in interpreter

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

Compiler Remarks

Annotates source code with compiler remarks

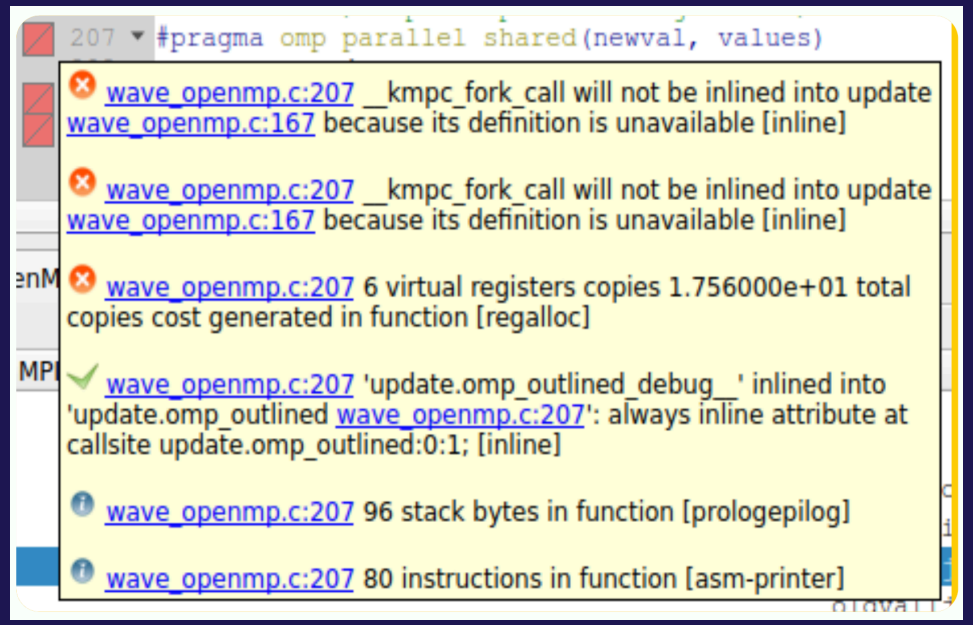
- Remarks are extracted from the compiler optimisation report
- Compiler remarks are displayed as annotations next to your source code

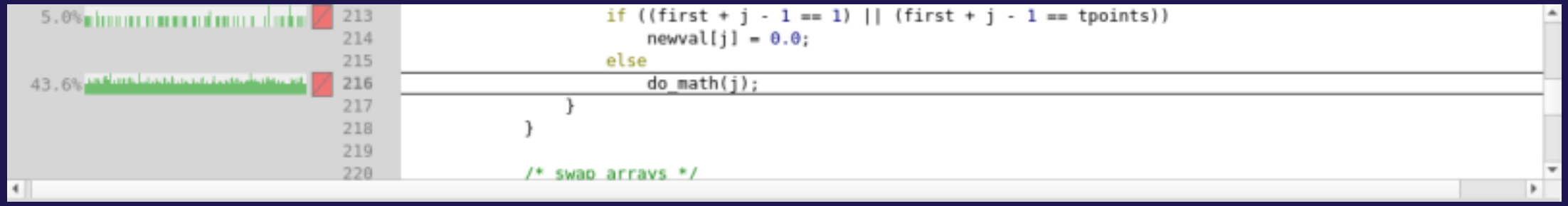
Colour coded

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

Compiler Remarks menu.

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





Cheat sheet

Job Script

```
#!/bin/bash -l

#PBS -l select=1
#PBS -l filesystems=home:eagle
#PBS -l walltime=0:30:00
#PBS -q HandsOnHPC
#PBS -A alcf_training

module use /soft/modulefiles
module load forge cray-cti

# Debug in reverse connect mode
ddt --connect mpiexec -n 4 ./simple

# Debug in offline mode
ddt --offline -o offline-debugging.html --break-at=simple.c:32 --break-at=simple.c:43 mpiexec -n 4 ./simple

# Profile a Python application
module load conda
conda activate base

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

Interactive Session

```
qsub -I -l select=1 -l filesystems=home:eagle -l
walltime=0:30:00 -q HandsOnHPC -A alcf_training

module use /soft/modulefiles
module load forge cray-cti
```

Forge commands

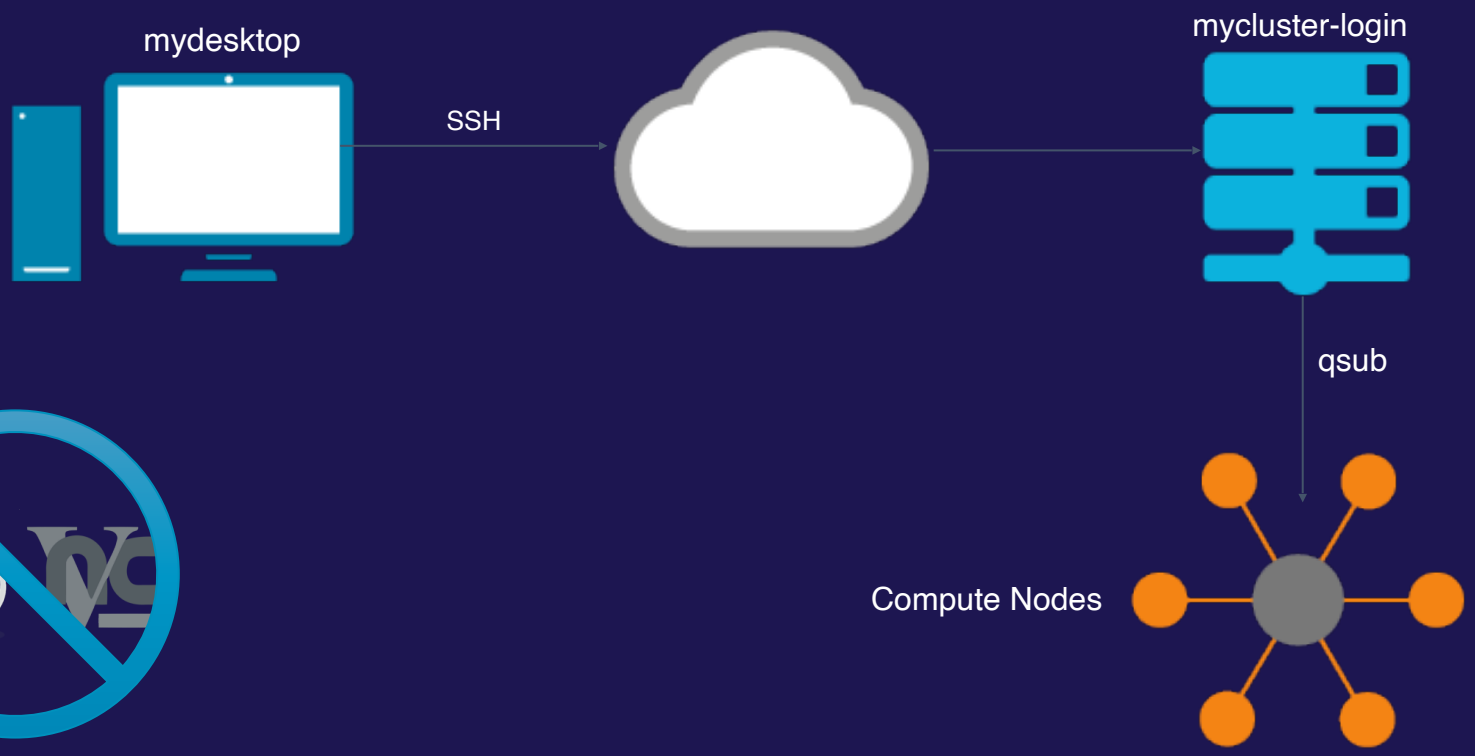
```
ddt --connect      # Reverse connect
ddt --offline      # Run DDT without GUI
map --profile      # Profile without GUI
perf-report        # Generate Performance Report
```

Guides

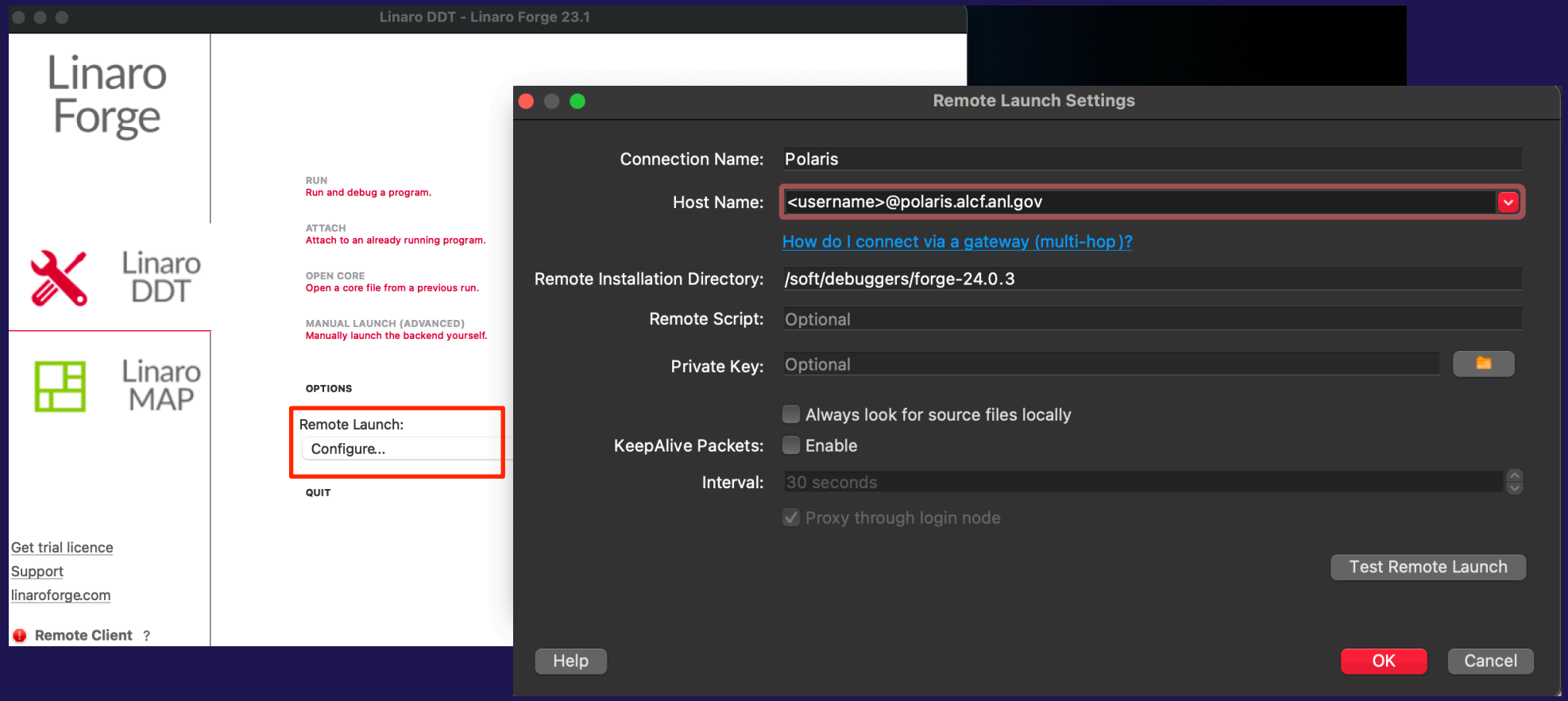
Forge userguide

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 Polaris



Thank you

- www.linaroforge.com
- support@forge.linaro.com
- https://docs.linaroforge.com/24.0.5/html/forge/index.html