

October 29-31, 2024

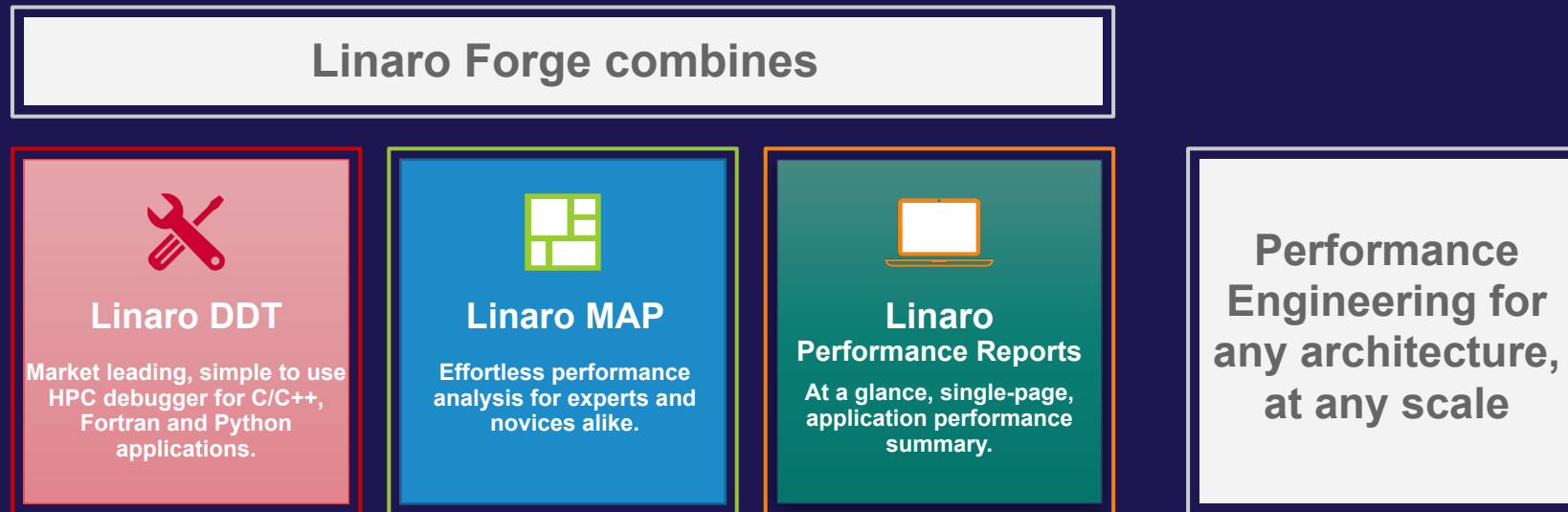
---



# ALCF Hands-on HPC Workshop

# HPC Development Solutions from Linaro

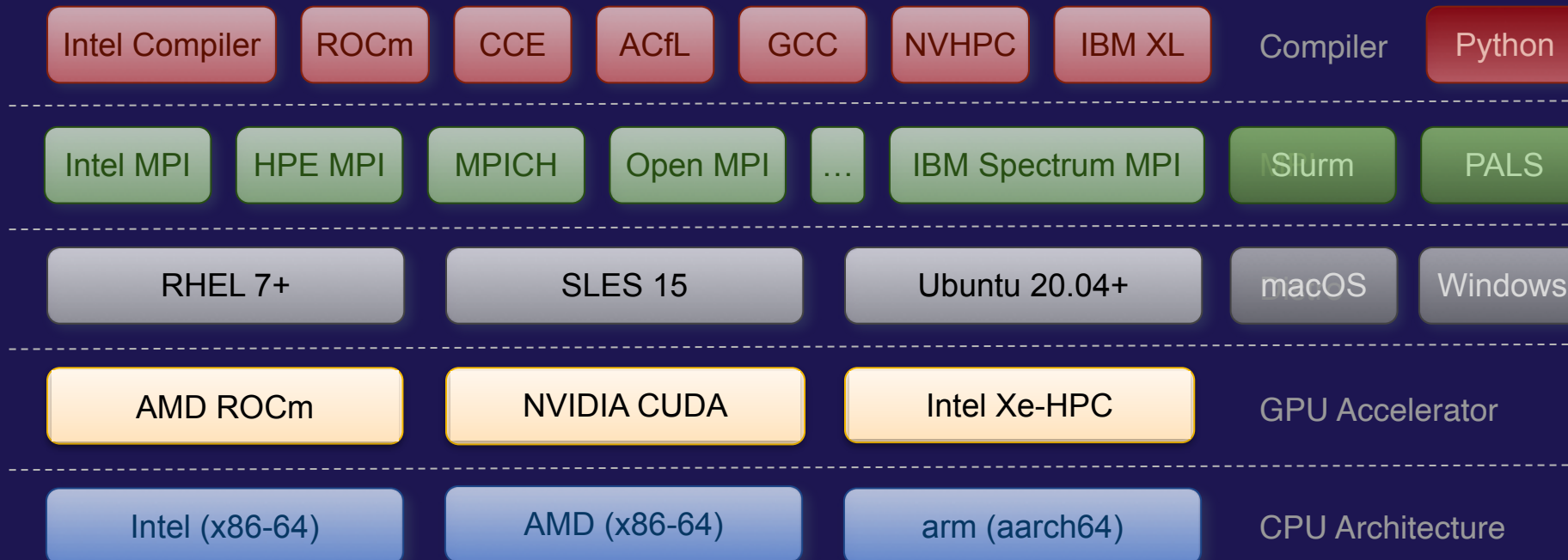
Best in class commercially supported tools for Linux and high-performance computing (HPC)





# DDT Supported Platforms

Works across hardware architectures and HPC technologies



# 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

The screenshot displays the DDT GUI with several key components highlighted by numbered callouts:

- 1**: The top toolbar containing execution and control icons.
- 2**: The process control area showing a tree view of process groups (All, Group 1, Group 2) with status indicators (Paused, Playing, Finished).
- 3**: The source code editor displaying C code with MPI-related functions like `MPI_Send` and `MPI_Recv`.
- 4**: The Locals window showing a list of variables and their values, such as `my_rank` with value 1.
- 5**: The Evaluate window showing the result of an expression, such as `x + y` resulting in 10012.
- 6**: The Stacks (All) window showing the call stack for the current process, including `main (hello.c:148)`.
- 7**: The Project Files tree view on the left side of the interface.
- 8**: The search bar at the top of the Project Files view.



# Linaro DDT Debugger Highlights

Tracepoint	Processes	Values logged
vhone:50:85	976, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol 2-83 mod pey
vhone:50:81	960, ranks 12,14-17,22-23,12...	ls 1 kmax pez
vhone:50:85	942, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol 2-83 mod pey
vhone:50:81	929, ranks 12,14-17,22-23,12...	ls 1 kmax pez
vhone:50:85	919, ranks 12,14-17,22-23,12...	mype 2172-3527 jcol 2-83 mod pey
vhone:50:81	898, ranks 12,14-17,22-23,12...	ls 1 kmax pez

The scalable print alternative

```

for (i = 0 ; i < SIZE_M; i++)
  for (j = 0 ; j < SIZE_O; j++)
    C[i][j] = 0;

for (i = 0 ; i < SIZE_M; i++)
  for (j = 0 ; j < SIZE_N; j++)
    for (k = 0 ; k < SIZE_O; k++)
      C[i][j] += A[i][k] * B[k][j];

if (numprocs > 1)
  MPI_Scatter(
    MPI_Ranks,
    MPI_Ranks,
    MPI_Ranks,
    MPI_Ranks,
    MPI_COMM_WORLD,
    MPI_BYTE,
    MPI_BYTE,
    MPI_COMM_WORLD,
    MPI_STATIC,
    MPI_INFO_NULL,
    0);

printf("M

if (argc > 1)
  for (i = 0 ; i < SIZE_M; i++)
  {
    printf("%i ");
  }
  
```

Stop on variable change

```

hello.c
This file is newer than your program. Please recompile then restart your debugging session.

43 else
44     test=-1;
45 }
46
47 void func3()
48 {
49     void* i = (void*) 1;
50     while(i++ || i)
51         free((void*)i);
52 }
53
54 portability 'i' is of type 'void *'. When using void pointers in calculations, the behaviour is undefined.
Left click to add a breakpoint on line 50

55 {
56     typeThree test;
57     typeThree* t2;
58     int i;
  
```

Static analysis warnings on code errors

```

if (argv[i] && !strcmp(argv[i], "crash")) {
  argv[i] = 0;
  printf("%s", *(char **)argv[i]);
  /* we shall see
}
}

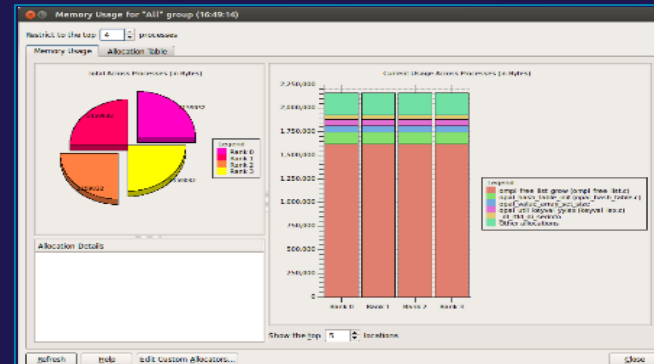
func1();

func2();
fprintf(stderr, "I

beingWatched = 1;

test.anotherList.s
test.c = 'p';
beingWatched = 0;
  
```

Detect read/write beyond array bounds



Detect stale memory allocations

# GPU Debugging

The screenshot displays a GPU debugging interface with the following components:

- Threads Panel:** Shows three threads labeled 1, 2, and K4. Thread K4 is selected.
- GPU Threads Panel:** Shows configuration for 'MatrixMulHIP(float\*...)' with Block size 3x2x0 and Thread size 5x18x0. Grid size is 4x4x1 and Block size is 32x32x1.
- Code Editor:** Shows C++ code for a matrix multiplication kernel. The line `C[ i * wB + j ] = temp;` is highlighted.
- GPU Devices Panel:** Shows 'Ranks 0' with 'vega20' (2 Devices, 0-1 IDs, 2400 Threads, 240 Cores).
- Kernel Progress View:** Shows a progress bar for 'MatrixMul...' which is mostly green (scheduled).
- Evaluate Panel:** Shows variable values: `i = 82`, `wA = 128`, `wB = 128`, and `temp = 1.27999914`.

- 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

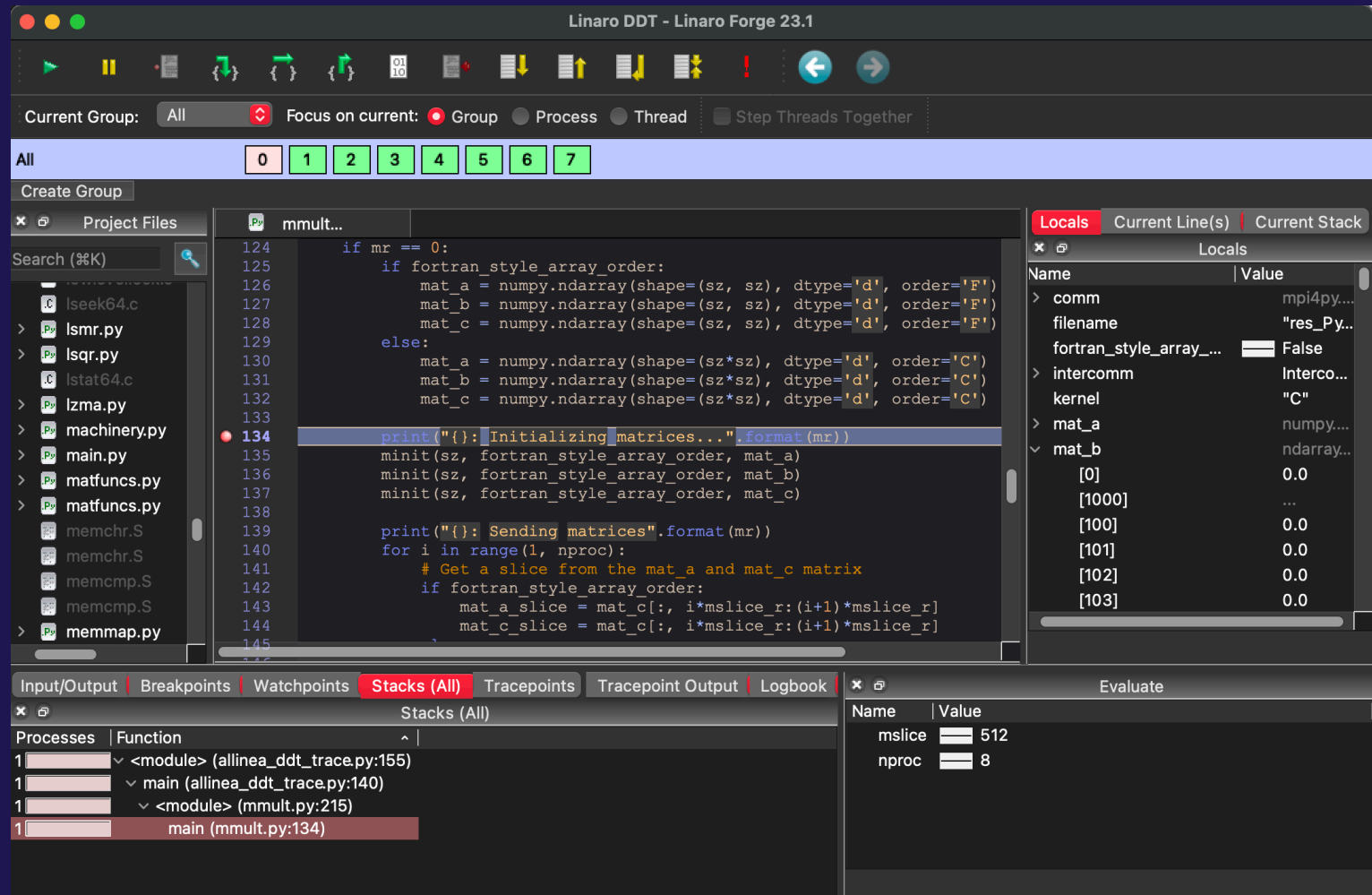
- Improved Evaluations:

- 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...

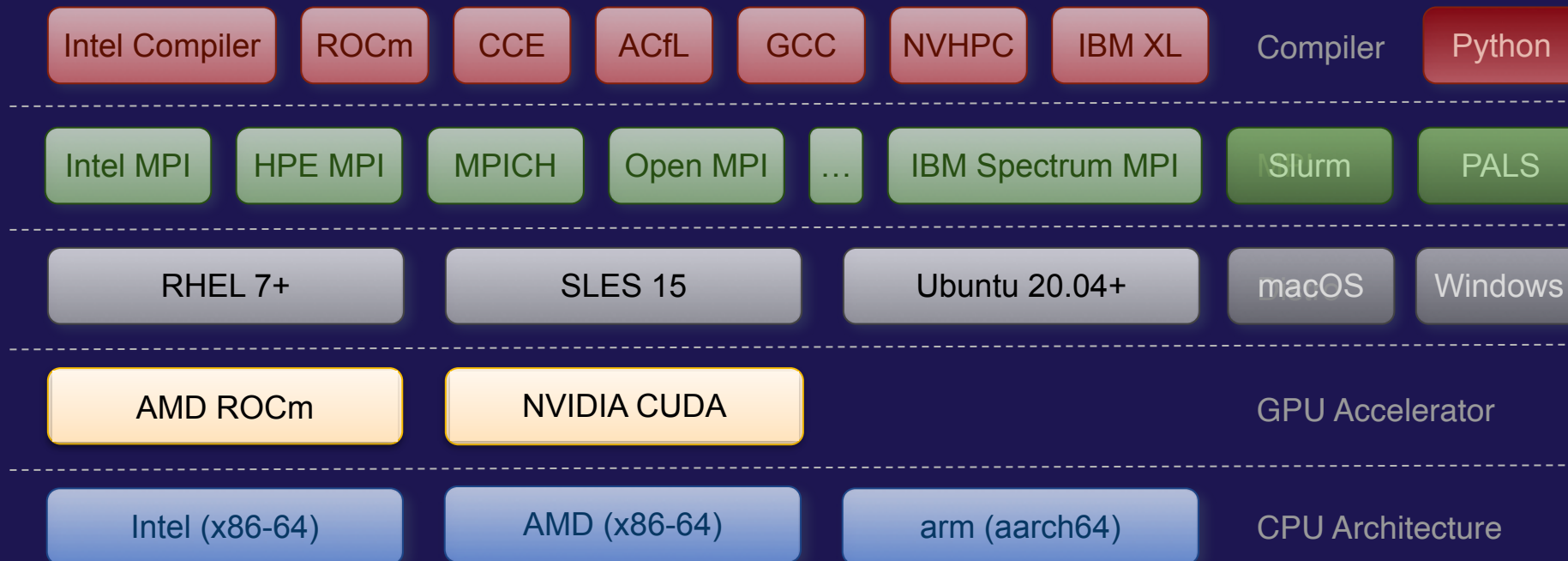
To do so, use following arguments:

- `$ 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)

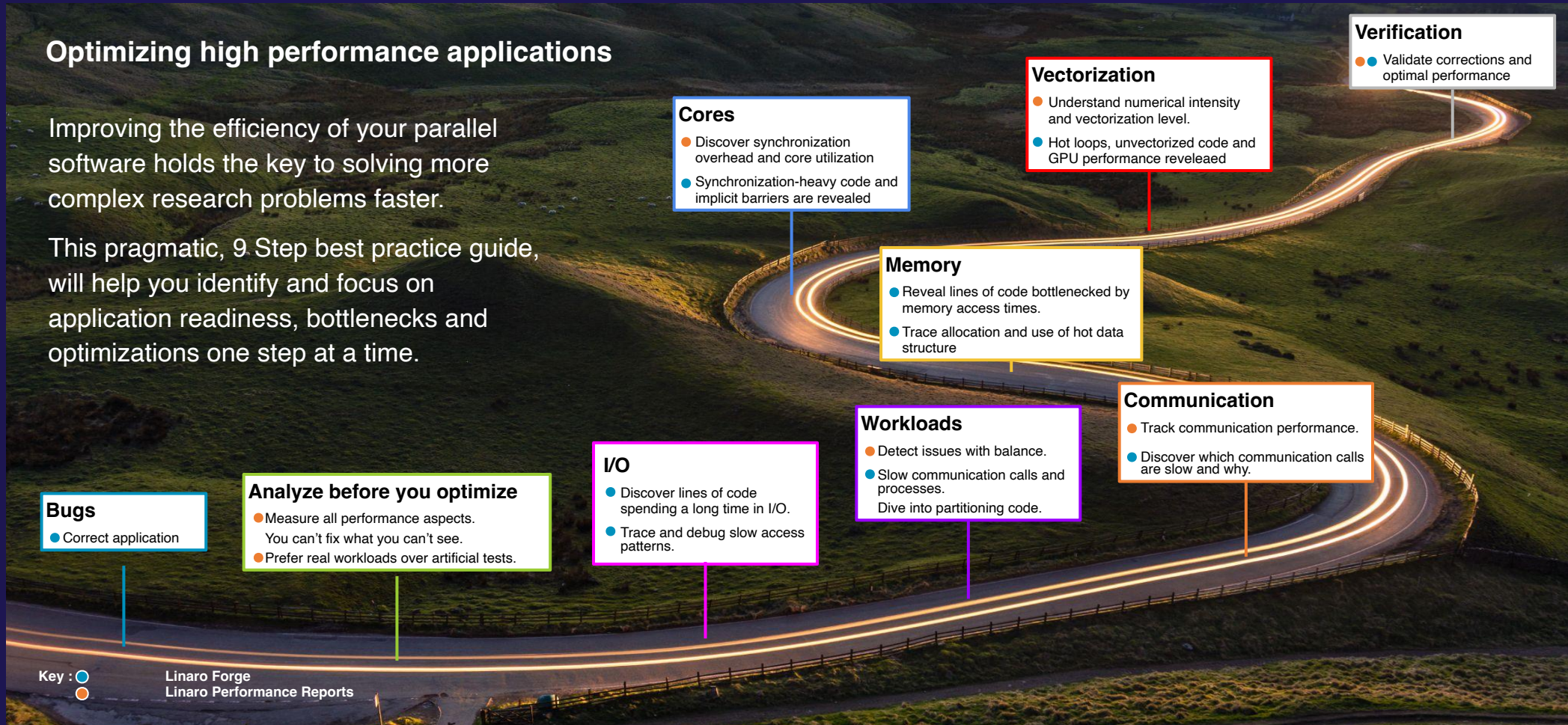


# 9 Step guide for optimising code

## 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.



# 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/../../Src//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

## 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.

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

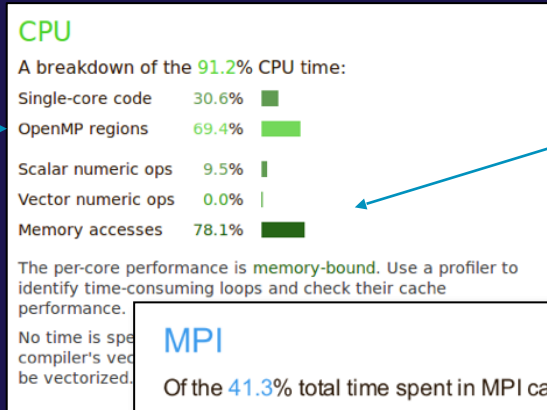
**I/O** 16.2% █

Time spent in filesystem I/O. High values are usually bad. This is **average**; check the I/O breakdown section for optimization advice

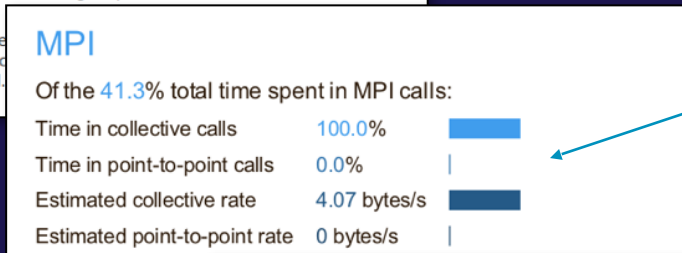
# Linaro Performance Reports Metrics

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

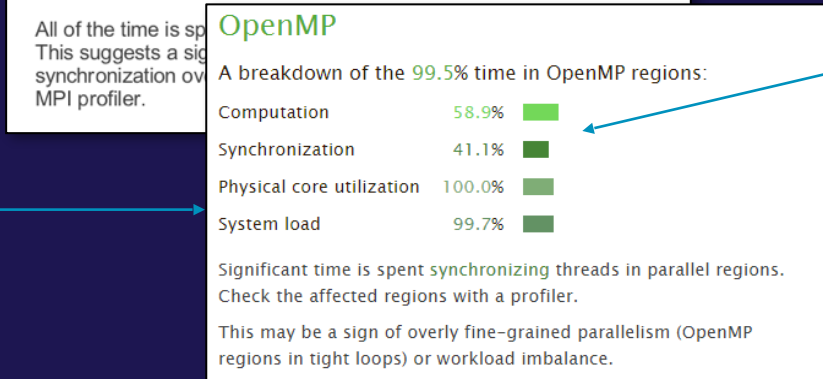
Multi-threaded parallelism



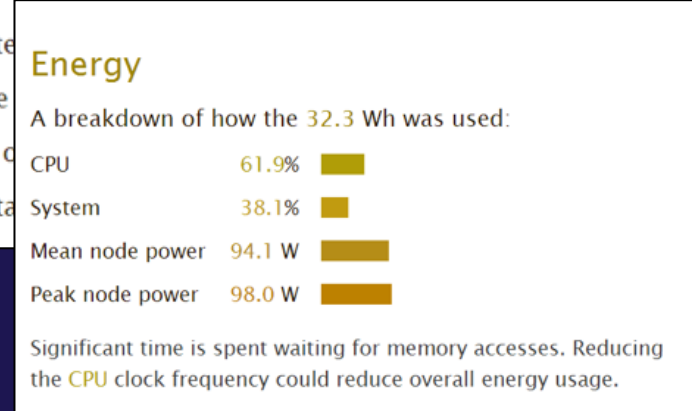
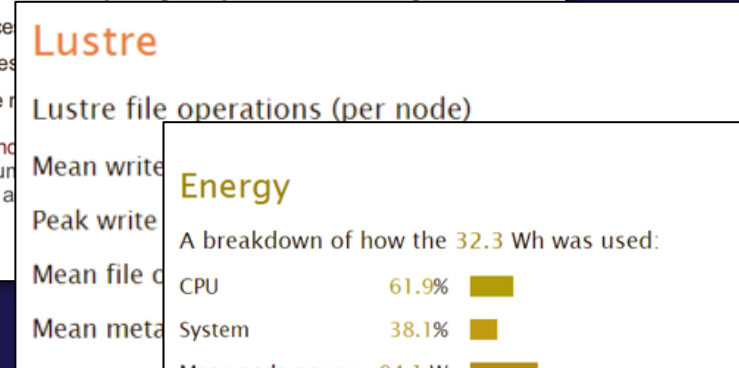
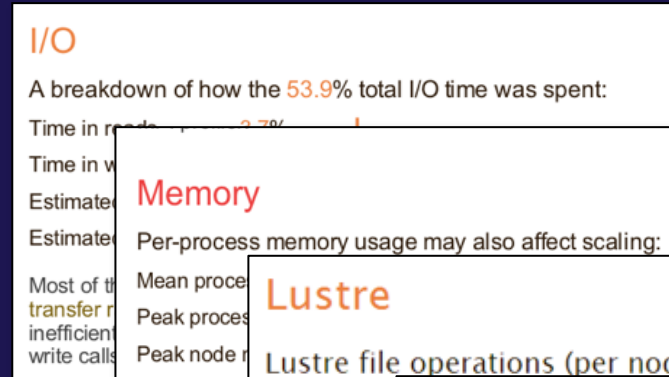
SIMD parallelism



Load imbalance



System usage





# MAP Capabilities

MAP is a sampling based scalable profiler

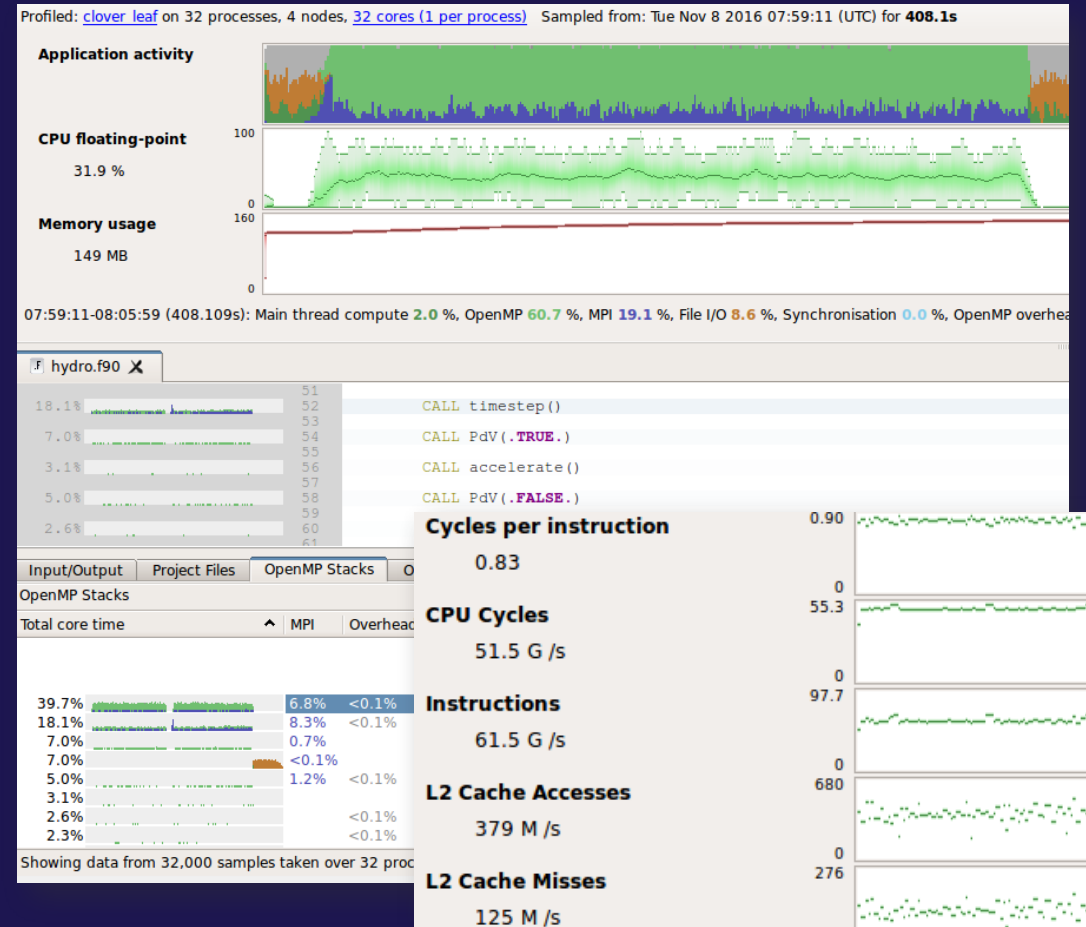
- Built on same framework as DDT
- Parallel support for MPI, OpenMP, CUDA
- Designed for C/C++/Fortran

Designed for 'hot-spot' analysis

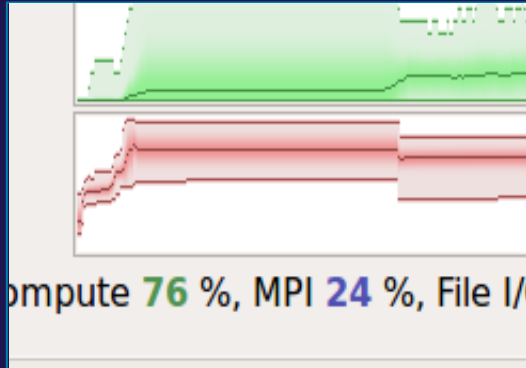
- Stack traces
- Augmented with performance metrics

Adaptive sampling rate

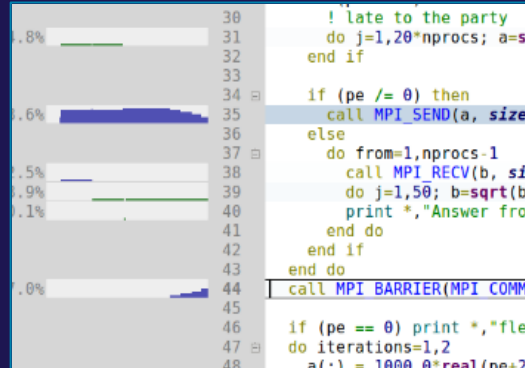
- Throws data away - 1,000 samples per process
- Low overhead, scalable and small file size



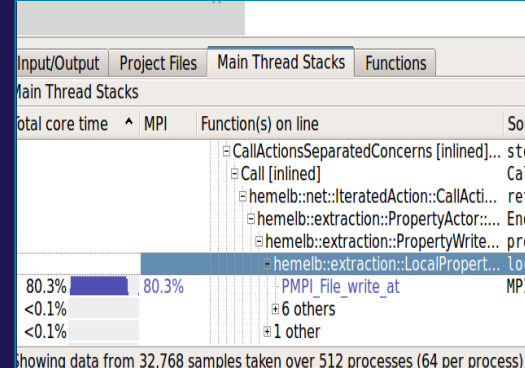
# MAP Highlights



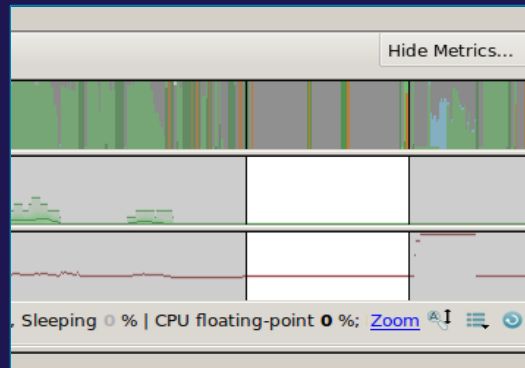
Find the peak memory use



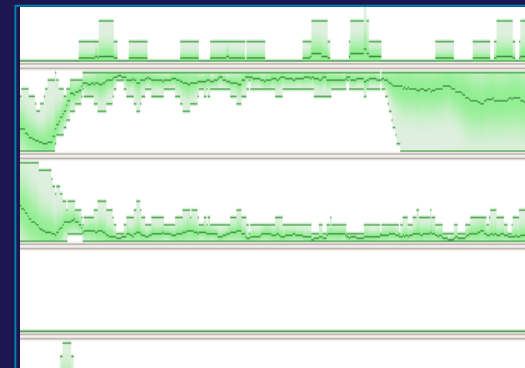
Fix an MPI imbalance



Remove I/O bottleneck



Make sure OpenMP regions make sense

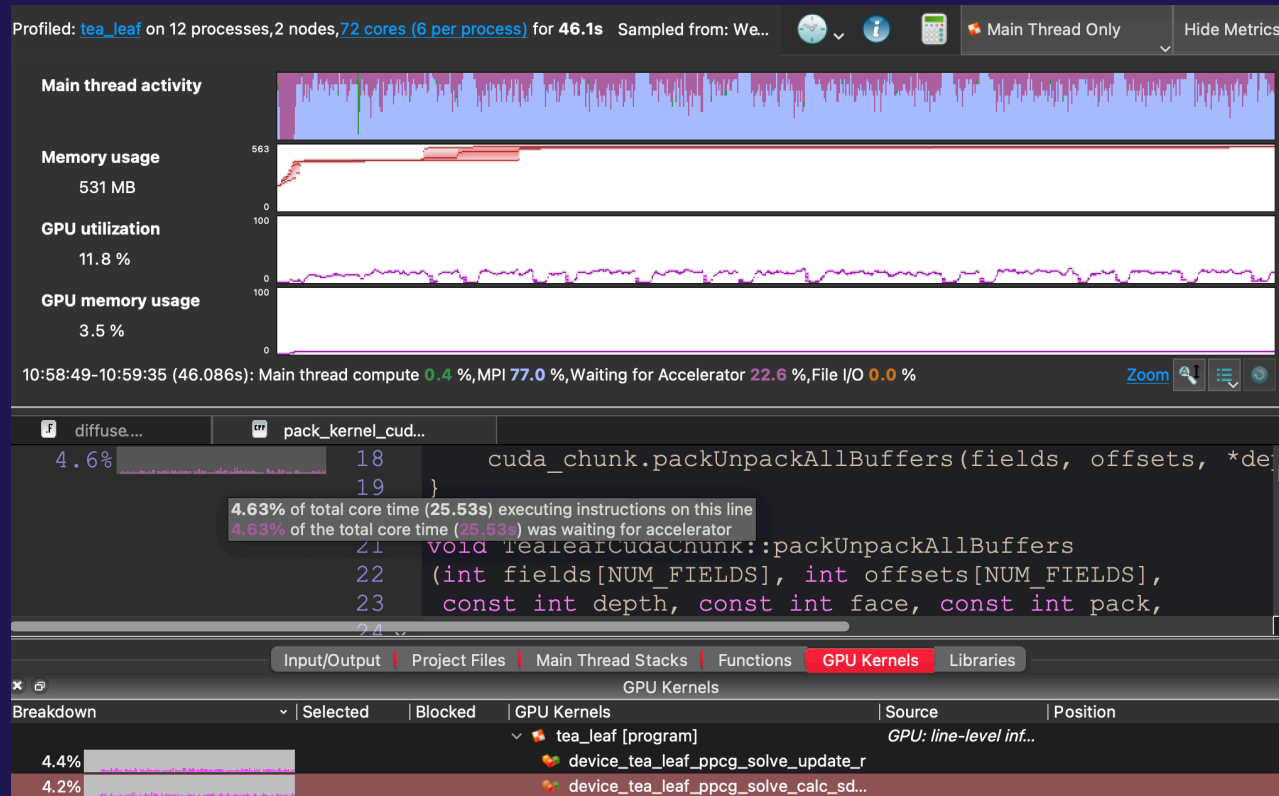


Improve memory access

```
{
mmult(size, nproc, mat a, mat
res += A[i*size+k]*B[k*size+j]
MPI_Finalize();
mwrite(size, mat c, filename)
```

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

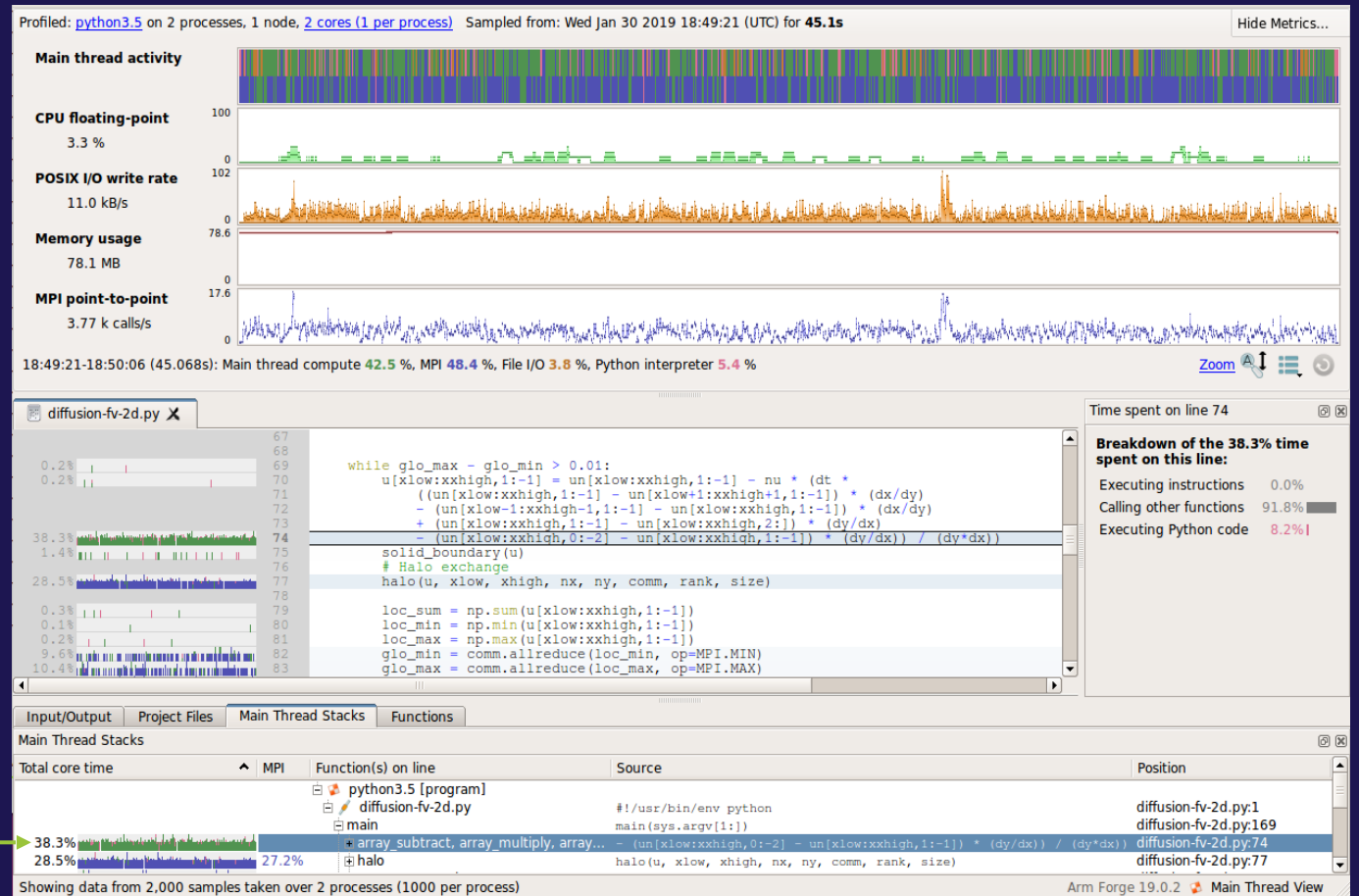
Works with MPI4PY

- 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

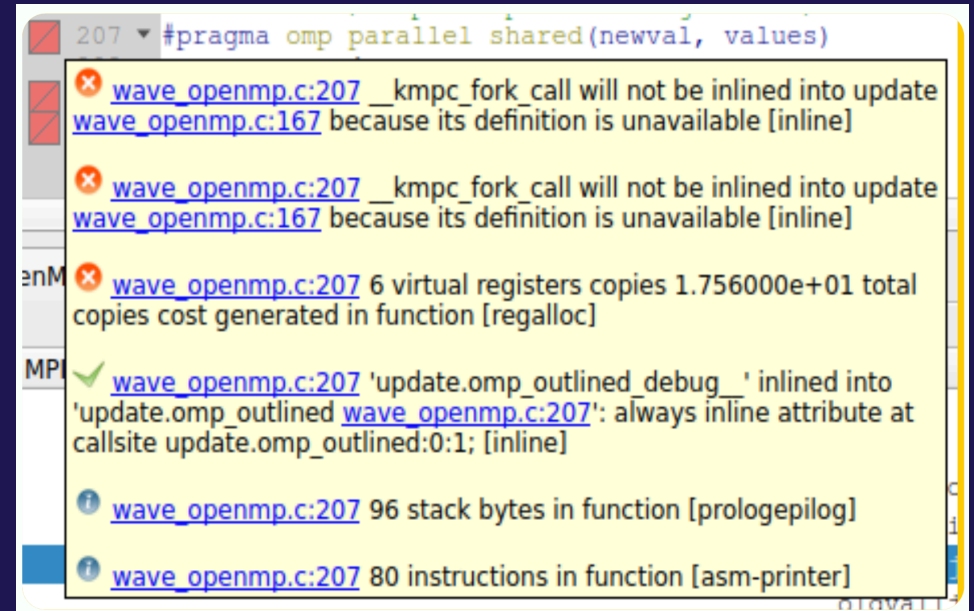
- Remarks are extracted from the compiler optimisation report
- Compiler remarks are displayed as annotations next to your source code

## 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

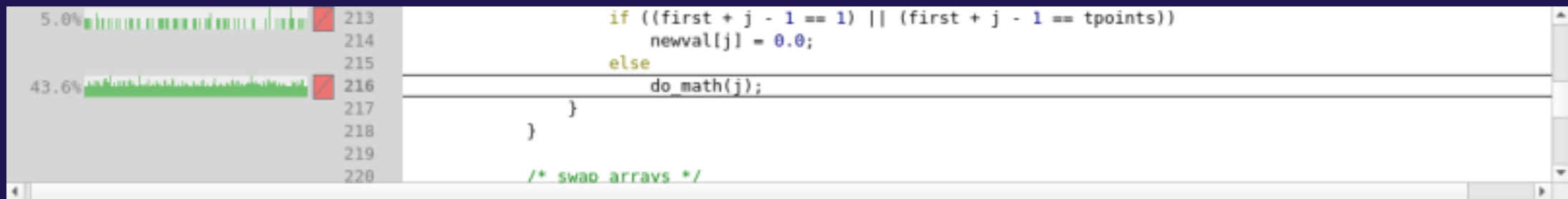
## Compiler Remarks menu.

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



```
207 #pragma omp parallel shared(newval, values)
```

- ✘ [wave\\_omp.c:207](#) \_\_kmpc\_fork\_call will not be inlined into update [wave\\_omp.c:167](#) because its definition is unavailable [inline]
- ✘ [wave\\_omp.c:207](#) \_\_kmpc\_fork\_call will not be inlined into update [wave\\_omp.c:167](#) because its definition is unavailable [inline]
- ✘ [wave\\_omp.c:207](#) 6 virtual registers copies 1.756000e+01 total copies cost generated in function [regalloc]
- ✔ [wave\\_omp.c:207](#) 'update.omp\_outlined\_debug\_' inlined into 'update.omp\_outlined [wave\\_omp.c:207](#)': always inline attribute at callsite update.omp\_outlined:0:1; [inline]
- ℹ [wave\\_omp.c:207](#) 96 stack bytes in function [prologepilog]
- ℹ [wave\\_omp.c:207](#) 80 instructions in function [asm-printer]



```
213 if ((first + j - 1 == 1) || (first + j - 1 == tpoints))
214     newval[j] = 0.0;
215 else
216     do_math(j);
217 }
218 }
219
220 /* swap arrays */
```

# 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 -l -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***

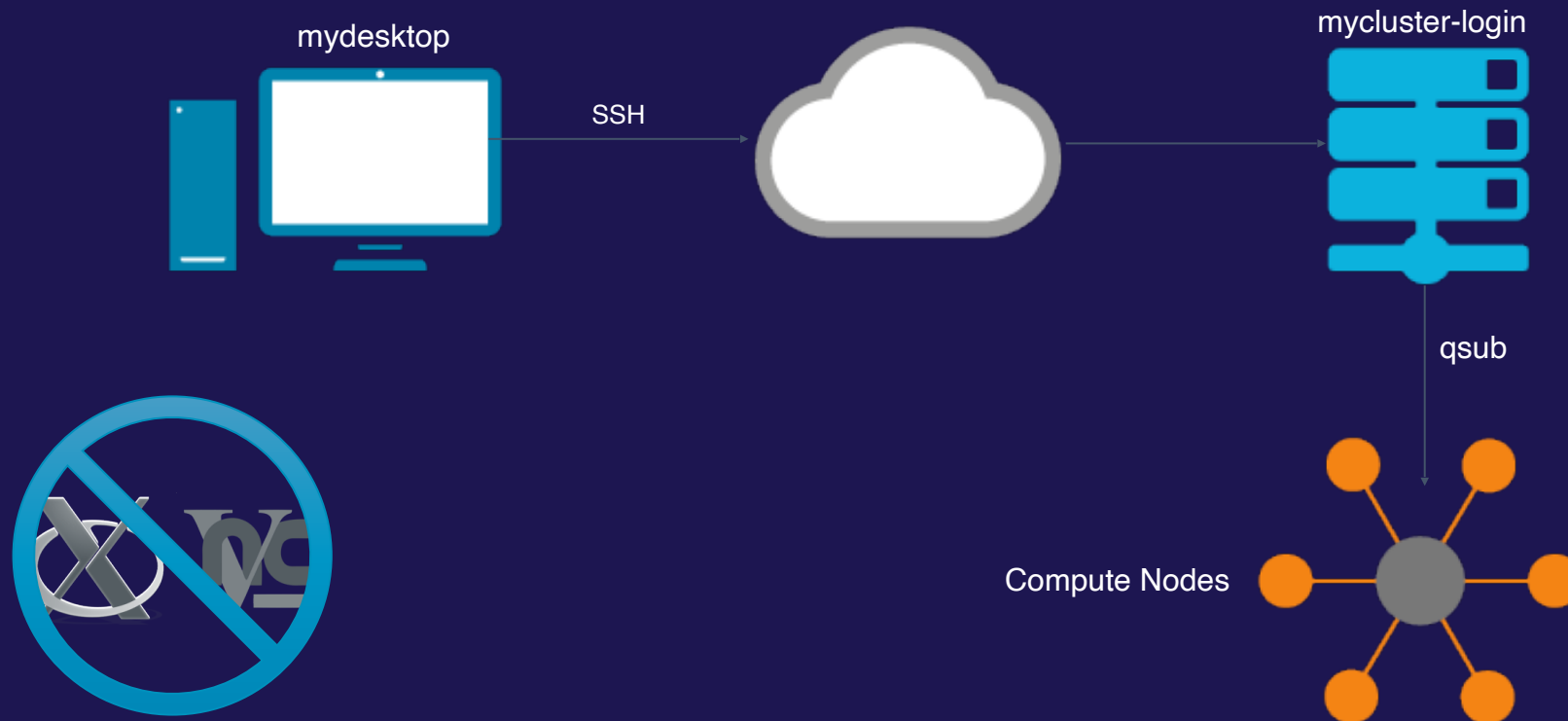
```
ddt --connect      # Reverse connect
ddt --offline      # Run DDT without GUI
map --profile      # Profile without GUI
perf-report        # Generate Performance Report
```

## ***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

The image shows a screenshot of the Linaro Forge application interface and a 'Remote Launch Settings' dialog box. The Linaro Forge interface includes the Linaro Forge logo, Linaro DDT logo, and Linaro MAP logo. The 'Remote Launch' option is highlighted with a red box. The 'Remote Launch Settings' dialog box contains the following fields and options:

- Connection Name: Polaris
- Host Name: <username>@polaris.alcf.anl.gov
- Remote Installation Directory: /soft/debuggers/forge-24.0.3
- Remote Script: Optional
- Private Key: Optional
- Always look for source files locally:
- KeepAlive Packets:  Enable
- Interval: 30 seconds
- Proxy through login node:

Buttons: Test Remote Launch, Help, OK, Cancel.

# Thank you

- [www.linaroforge.com](http://www.linaroforge.com)
- [support@forge.linaro.com](mailto:support@forge.linaro.com)
- <https://docs.linaroforge.com/24.0.5/html/forge/index.html>