Linaro Forge

Forge Training For Debugging and Profiling

Rudy Shand - Field Application Engineer

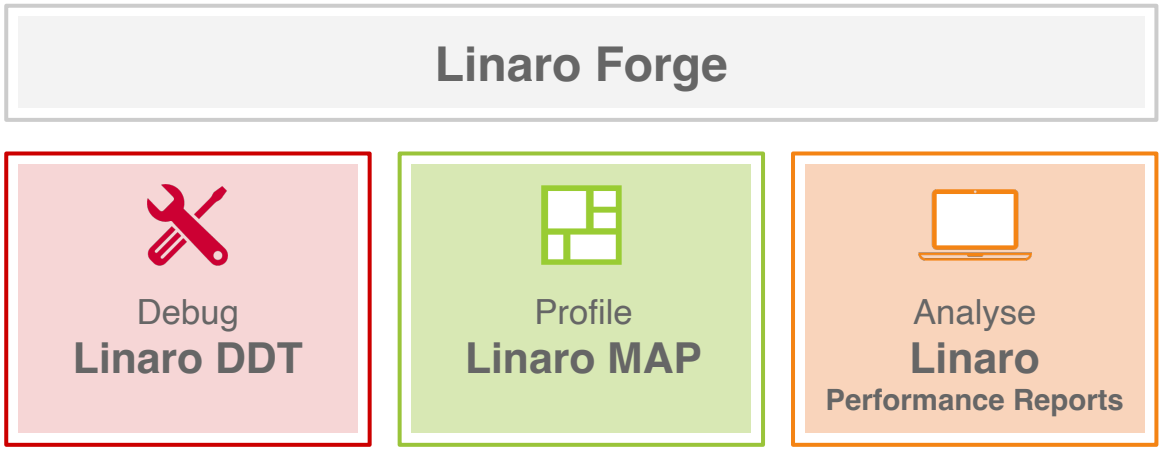
Linaro Forge

Agenda

- 09:00am Welcome
- 09:10am Ensuring Program Correctness with Linaro DDT
- 10:10am Break
- 10:20am Performance Engineering with Linaro Performance Tools
- 11:20am Wrap up

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

Linaro Forge

An interoperable toolkit for debugging and profiling

The de-facto standard for HPC development

- Most widely-used debugging and profiling suite in HPC
- Fully supported by Linaro on Intel, AMD, Arm, Nvidia, AMD GPUs, etc.

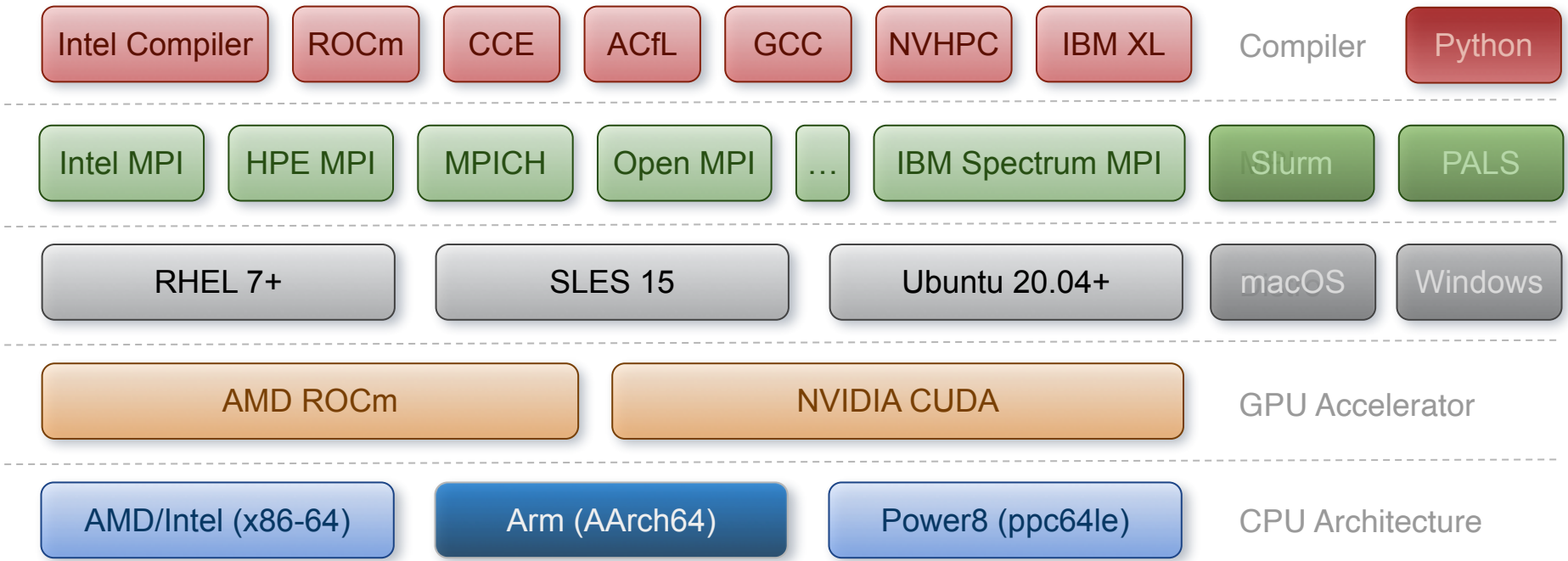
State-of-the art debugging and profiling capabilities

- 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 exascale applications)

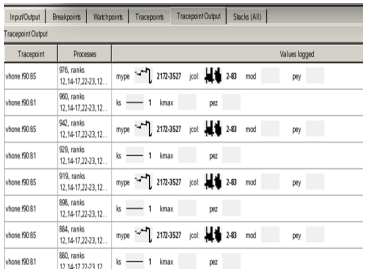
Easy to use by everyone

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

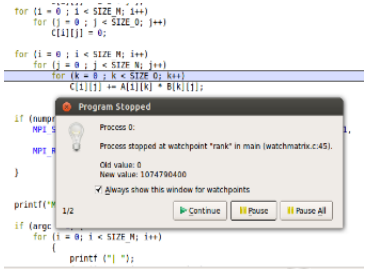
Supported Platforms



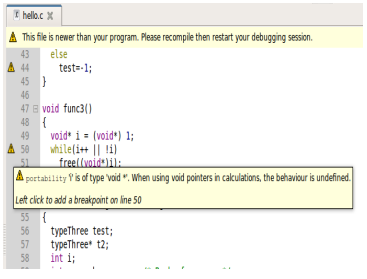
Linaro DDT Debugger Highlights



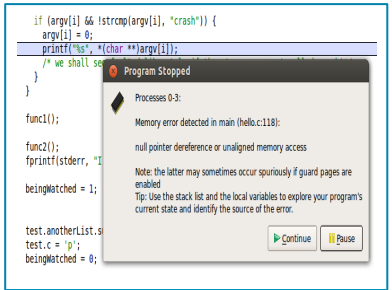
The scalable print alternative



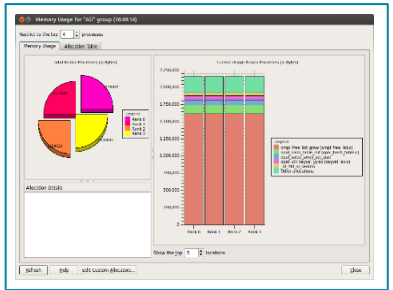
Stop on variable change



Static analysis warnings on code errors



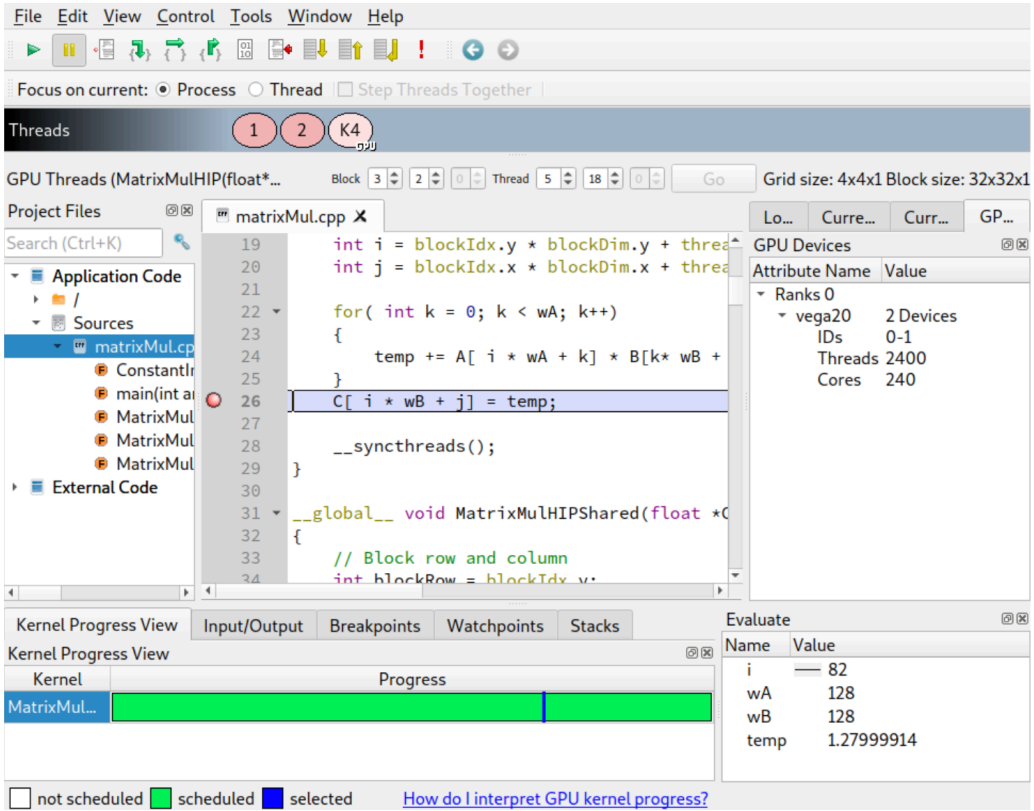
Detect read/write beyond array bounds



Detect stale memory allocations

Linaro Forge

GPU Debugging



- Support both AMD and Nvidia GPUs
- Debug simultaneously on GPU and CPU

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

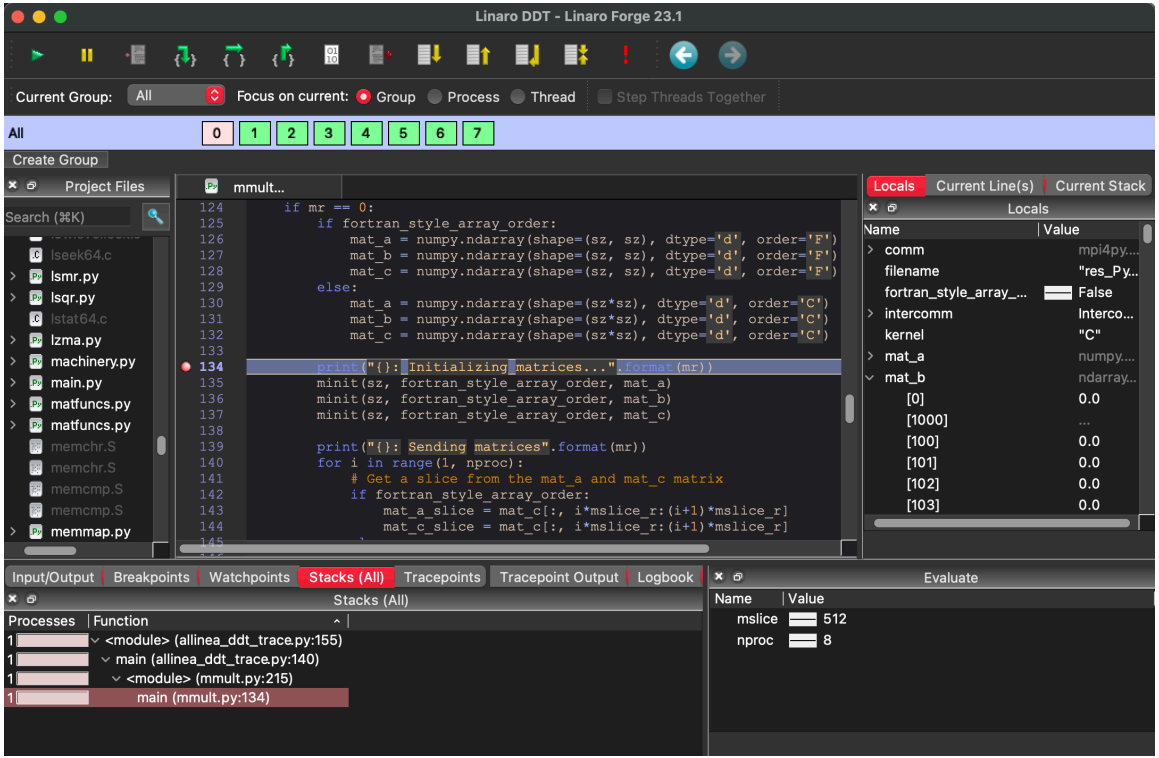
- Key (additional) GPU features:
  - Kernel Progress View
  - GPU thread in parallel stack view
  - GPU Thread Selector
  - GPU Device Pane

- For NVIDIA's nvcc compiler, kernels must be compiled with the -g -G flags
- Module load PrgEnv-nvidia
- Run GPU examples in a GPU batch job

Python Debugging

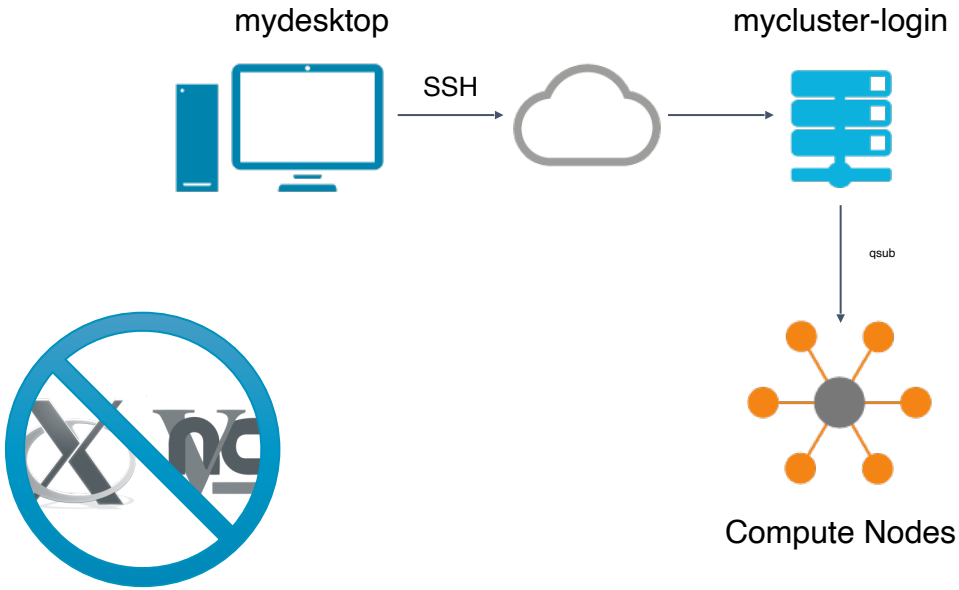
- Debug Features
  - Sparklines for Python variables
  - Tracepoints
  - MDA viewer
  - Mixed language support
- Improved Evaluations:
  - Matrix objects
  - Array objects
  - Pandas DataFrame
  - Series objects
- Python Specific:
  - Stop on uncaught Python exception
  - Show F-string variables in “Current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

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



The Forge GUI and where to run it

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



Hands on Setup

Remote System

Host perlmutter
    Hostname perlmutter.nersc.gov
    user <username>

linaro-forge-training.tar.gz

module load forge

Local Machine

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

Forge userguide

Hands on session

System Info

https://docs.nersc.gov/systems/perlmutter

Perlmutter:

- AMD EPYC 7763 CPUs
- NVIDIA A100 GPUs

https://docs.nersc.gov/systems/perlmutter/running-jobs/

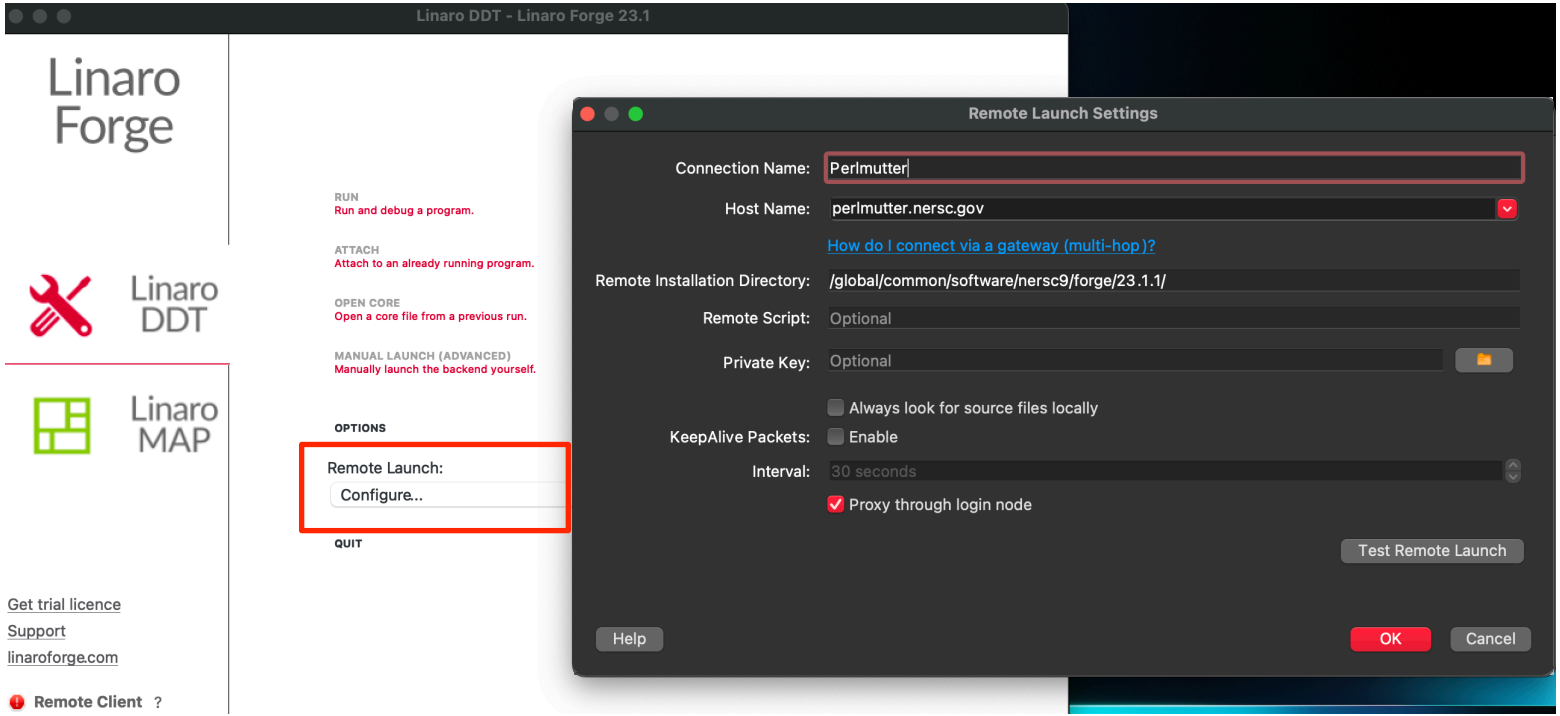
Interactive Session:

- salloc --nodes 1 --qos interactive --time 00:30:00 --constraint cpu --account=ntrain7 --reservation=forge_cpu
- salloc --nodes 1 --qos interactive --time 00:30:00 --constraint gpu --account=ntrain7 --reservation=forge_gpu

Scripting:

- <linaro-forge-training>/scripts

Remote connection to Perlmutter



Hands on session

Build and run debug examples

```
# Use default Perlmutter modules

# build deadlock, simple and split programs
cd <linaro-forge-training>/correctness/debug
make

# run simple example with ddt
ddt --connect srun -n 4 ./simple

# offline-debugging
sbatch <linaro-forge-training>/scripts/submit-job.sh
```

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

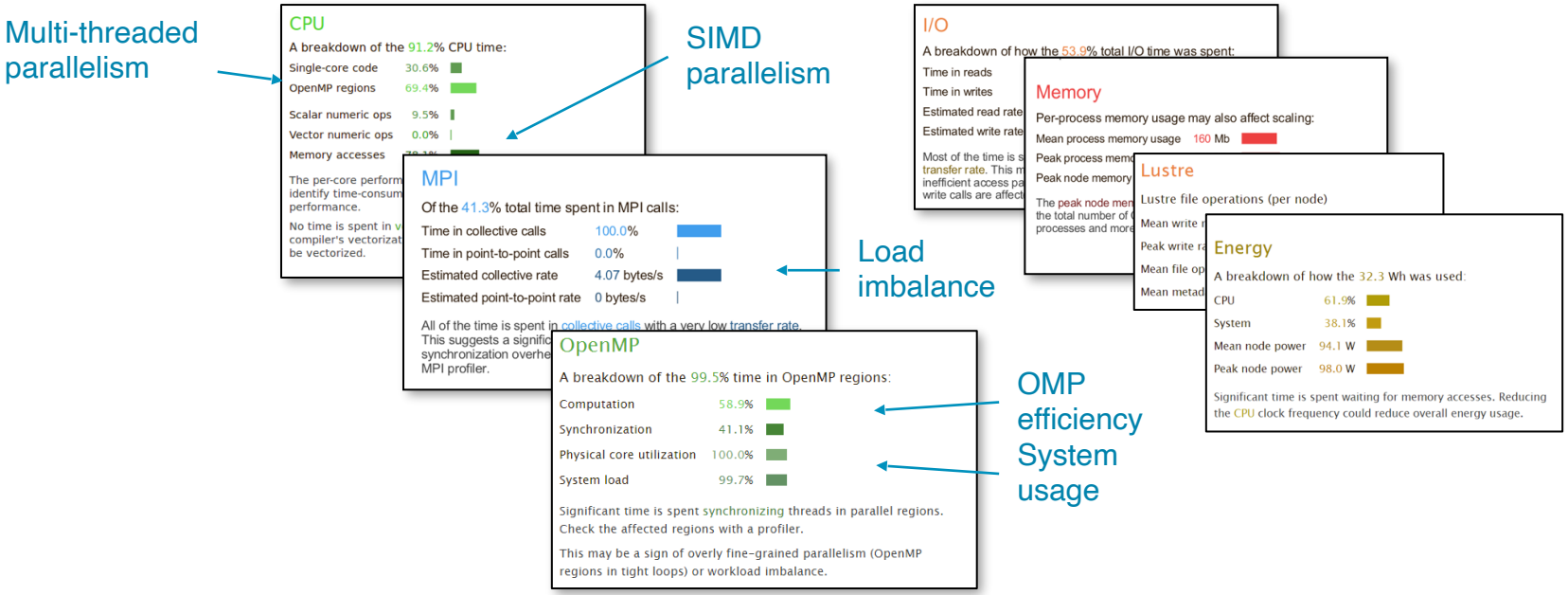
Relevant advice to avoid pitfalls

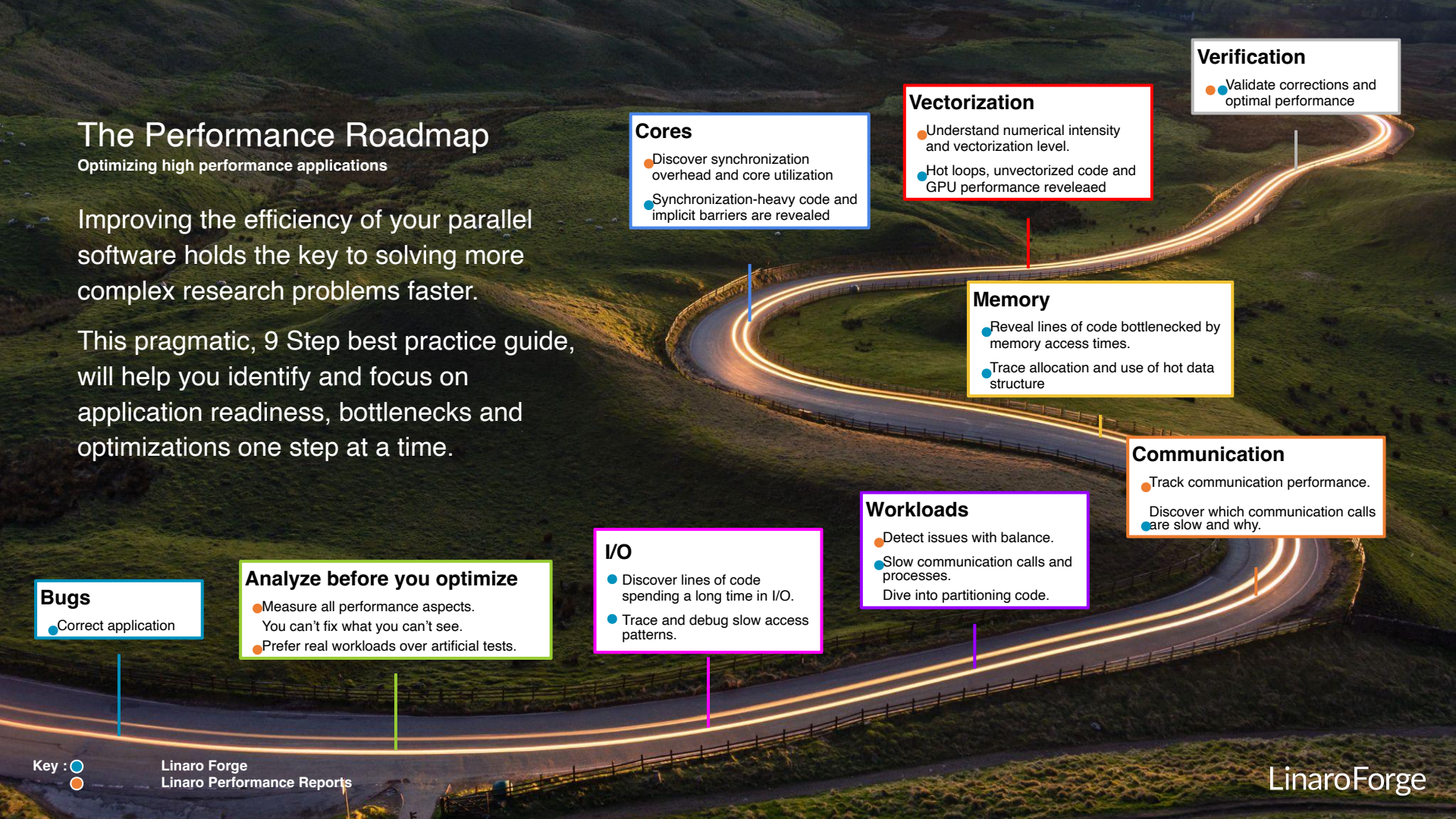
Adds value to typical users' workflows

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

Linaro Performance Reports Metrics

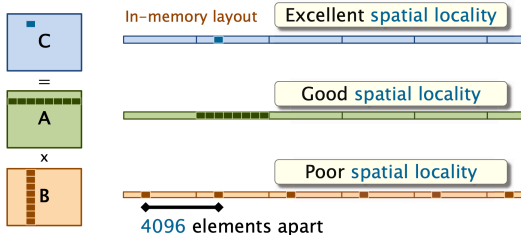
Lowers expertise requirements by explaining everything in detail right in the report



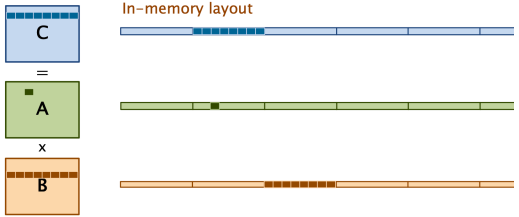

The Performance Roadmap
Optimizing high performance applications
Improving the efficiency of your parallel software holds the key to solving more complex research problems faster.
This pragmatic, 9 Step best practice guide, will help you identify and focus on application readiness, bottlenecks and optimizations one step at a time.
Verification
Validate corrections and optimal performance
Vectorization
Understand numerical intensity and vectorization level.
Hot loops, unvectorized code and GPU performance reveleaed
Cores
Discover synchronization overhead and core utilization
Synchronization-heavy code and implicit barriers are revealed
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.
Workloads
Detect issues with balance.
Slow communication calls and processes.
Dive into partitioning code.
I/O
Discover lines of code spending a long time in I/O.
Trace and debug slow access patterns.
Analyze before you optimize
Measure all performance aspects.
You can't fix what you can't see.
Prefer real workloads over artificial tests.
Bugs
Correct application
Key :
Linaro Forge
Linaro Performance Reports
LinaroForge

Performance Improvement

i, j, k



i, k, j



Think,

code,

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];
    }
  }
}
```

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

MAP Capabilities

MAP is a sampling based scalable profiler

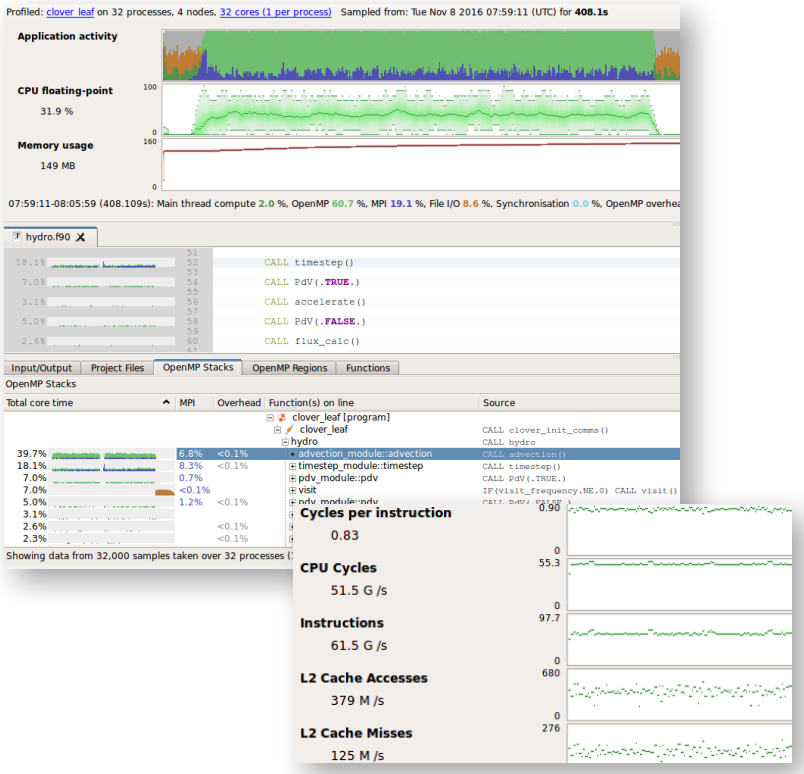
- Built on same framework as DDT
- Parallel support for MPI, OpenMP, CUDA
- Designed for C/C++/Fortran

Designed for ‘hot-spot’ analysis

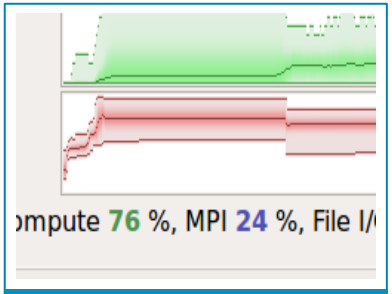
- Stack traces
- Augmented with performance metrics

Adaptive sampling rate

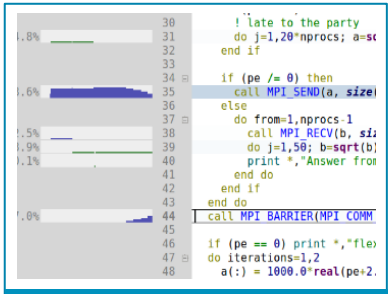
- Throws data away - 1,000 samples per process
- Low overhead, scalable and small file size



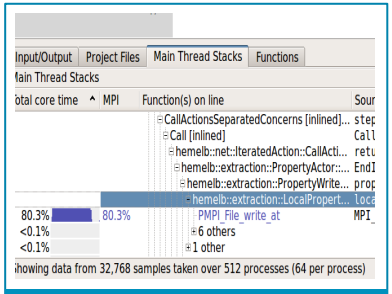
Linaro MAP Source Code Profiler Highlights



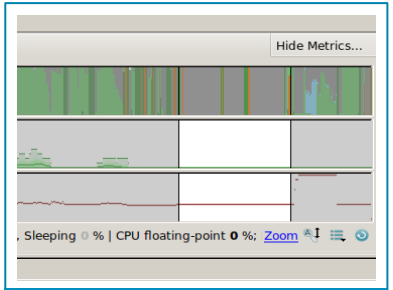
Find the peak memory use



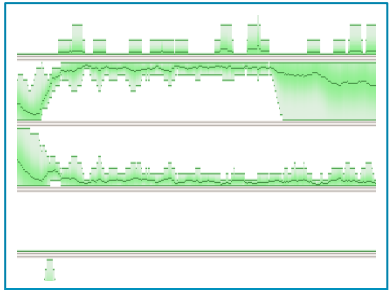
Fix an MPI imbalance



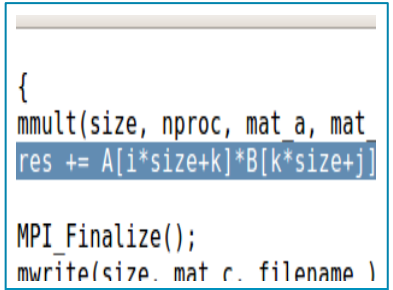
Remove I?O bottleneck



Make sure OpenMP regions make sense

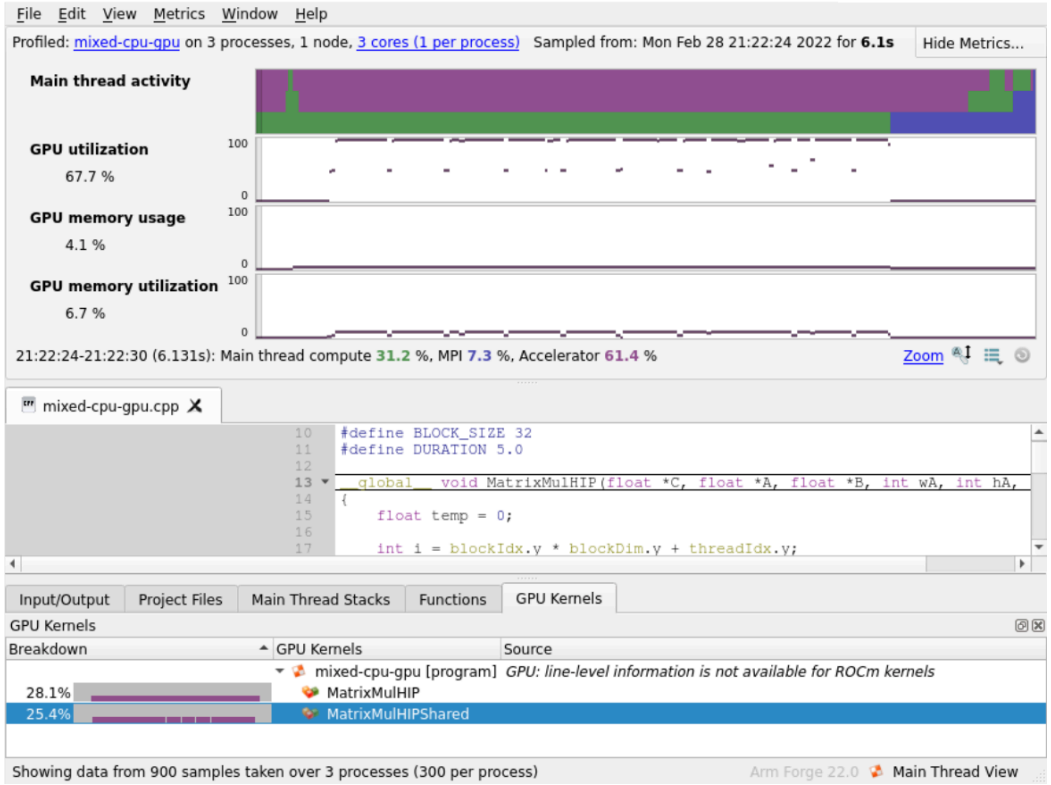


Improve memory access



Restructure for vectorization

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

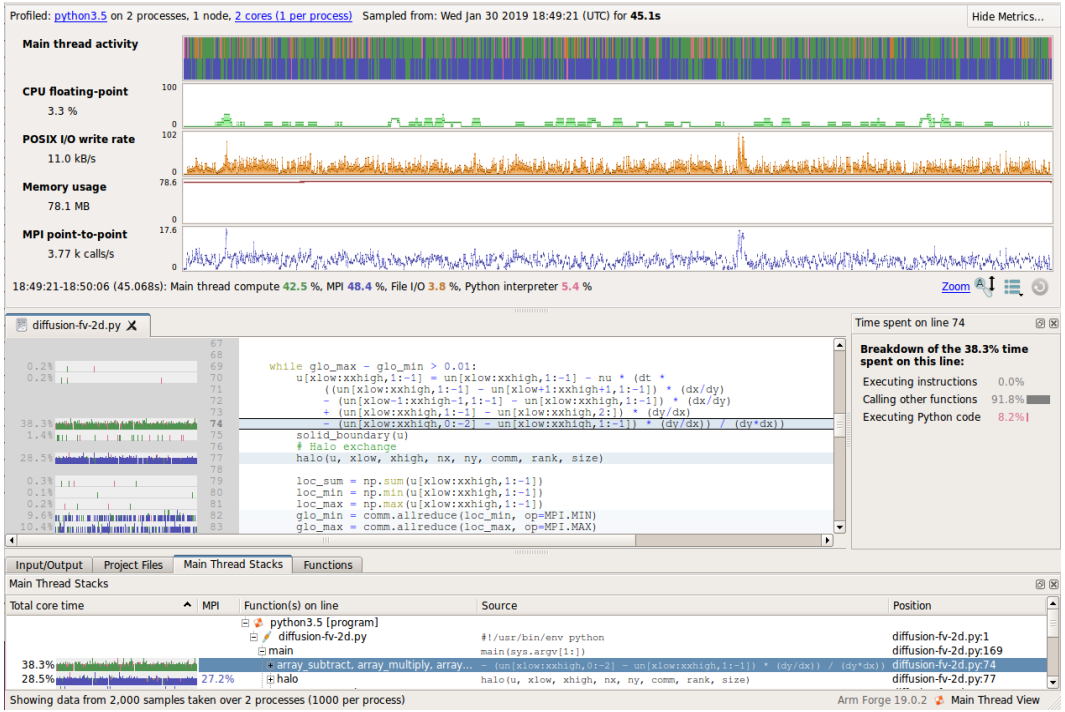
- Call stacks
- Time in interpreter

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 srun -n 2 python3 ./diffusion-fv-2d.py
```

Matrix Multiplication example

Build and run matrix multiplication example

https://docs.linaroforge.com/23.1.1/html/forge/worked_examples_appendix/mmult/analyze.html

```
# Build / Debug C and Fortran Examples
  make -f mmult.makefile DEBUG=1
  ddt --connect srun -n 8 ./mmult_c
  ddt --connect srun -n 8 ./mmult_f

# Build / Debug Python Examples
  module load python
  make -f mmult_py.makefile
  ddt --connect python3 %allinea_python_debug% ./mmult.py -s 3072

# Offline profile
  sbatch <linaro-forge-training>/scripts/submit-job.sh
```

Linaro Forge