

# Linaro Forge

Forge Training For Debugging and Profiling

Rudy Shand - Field Application Engineer

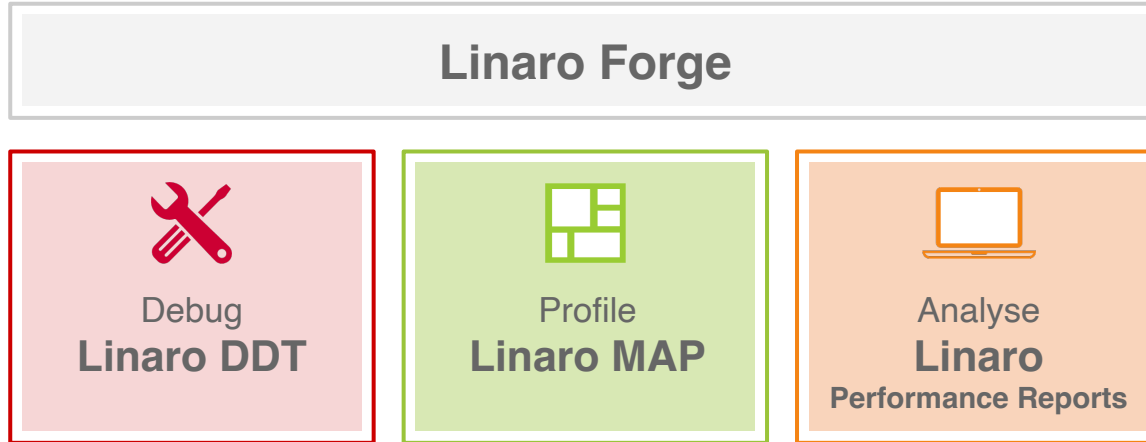
Linaro Forge

# 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

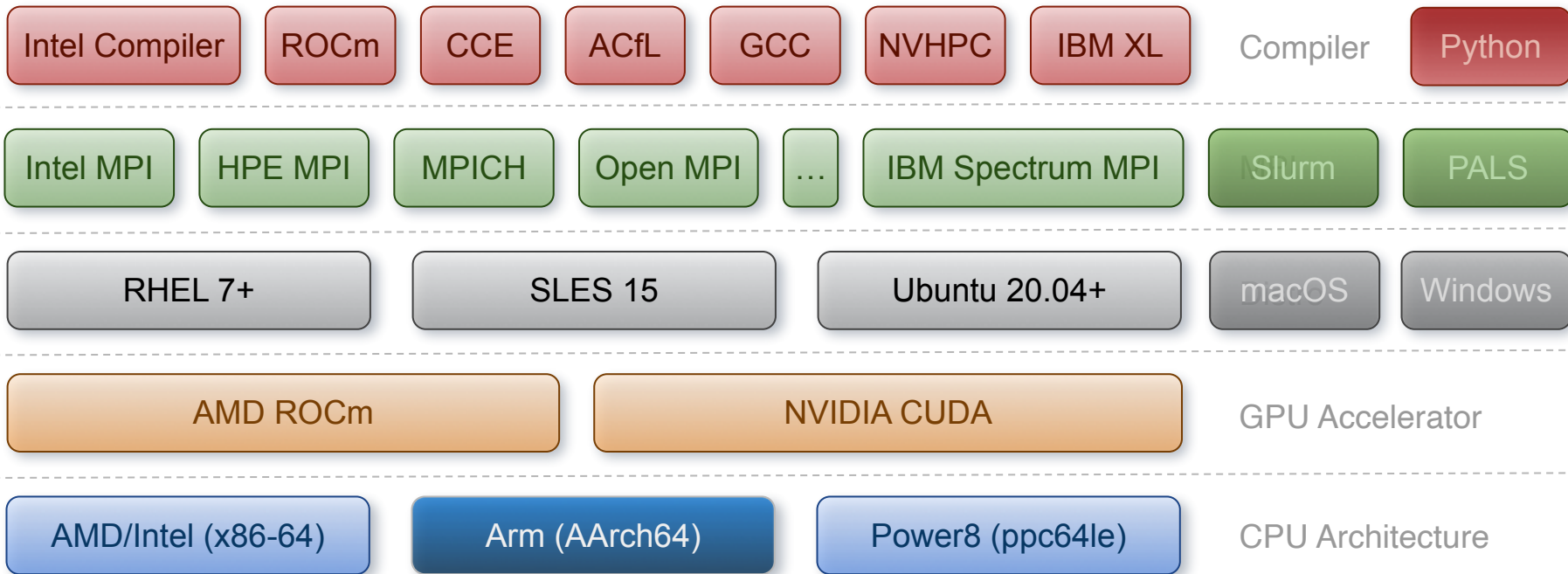
- Powerful and in-depth error detection mechanisms (including memory debugging)
- Sampling-based profiler to identify and understand bottlenecks
- Available at any scale (from serial to exascale applications)



### 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

Tracepoint	Process	Values logged
inode RD05	0% nmls 12.16.17.23.23.12	mype 2170-3527 jol 3-48 mod pty
inode RD01	0% nmls 12.16.17.23.23.12	lg 1 knas pcr
inode RD05	0% nmls 12.16.17.23.23.12	mype 2170-3527 jol 3-48 mod pty
inode RD01	0% nmls 12.16.17.23.23.12	lg 1 knas pcr
inode RD05	0% nmls 12.16.17.23.23.12	mype 2170-3527 jol 3-48 mod pty
inode RD01	0% nmls 12.16.17.23.23.12	lg 1 knas pcr
inode RD05	0% nmls 12.16.17.23.23.12	mype 2170-3527 jol 3-48 mod pty
inode RD01	0% nmls 12.16.17.23.23.12	lg 1 knas pcr
inode RD05	0% nmls 12.16.17.23.23.12	mype 2170-3527 jol 3-48 mod pty
inode RD01	0% nmls 12.16.17.23.23.12	lg 1 knas pcr

The scalable print alternative

```

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

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

if (strcmp("NP1", "NP1"))
    NP1 = "NP1";
}

printf("1/2\n");

if (argc > 1)
    for (i = 0; i < SIZE; i++)
        print ("");
    
```

Stop on variable change

```

hello.c:43: warning: 'Y' is of type 'void *'. When using void pointers in calculations, the behaviour is undefined.
43     else
44     test=1;
45 }
46
47 void func1()
48 {
49     void* i = (void*) 1;
50     while(i++ || i)
51     free((void*)i);
52 }
53
54 typeThree test;
55 typeThree t2;
56 int i;
57
58
    
```

Static analysis warnings on code errors

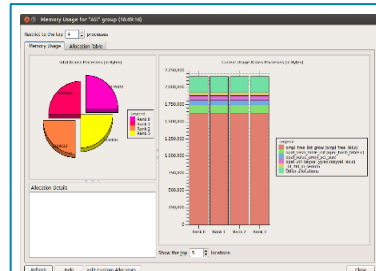
```

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

func1();
func2();
fprintf(stderr, "1\n");
beingWatched = 1;

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

Detect read/write beyond array bounds



Detect stale memory allocations

# GPU Debugging

The screenshot displays a development environment for GPU debugging. At the top, a menu bar includes File, Edit, View, Control, Tools, Window, and Help. Below it is a toolbar with various icons for execution and debugging. A 'Threads' panel shows three threads, with thread 'K4' selected. The main window shows a C++ code file named 'matrixMul.cpp' with the following code:

```
19 int i = blockIdx.y * blockDim.y + threadIdx.y;
20 int j = blockIdx.x * blockDim.x + threadIdx.x;
21
22 for( int k = 0; k < wA; k++)
23 {
24     temp += A[ i * wA + k ] * B[k* wB +
25 }
26 C[ i * wB + j ] = temp;
27
28 __syncthreads();
29 }
30
31 __global__ void MatrixMulHIPPShared(float *C
32 {
33     // Block row and column
34     int blockDim = blockDim.y;
```

On the right, a 'GPU Devices' panel shows the configuration for 'vega20' with 2 devices, 0-1 IDs, 2400 threads, and 240 cores. At the bottom, a 'Kernel Progress View' shows a progress bar for the 'MatrixMul...' kernel, which is currently scheduled (green bar). An 'Evaluate' panel shows the current values of variables: i=82, wA=128, wB=128, and temp=1.27999914. A legend at the bottom left indicates 'not scheduled' (white), 'scheduled' (green), and 'selected' (blue).

- 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

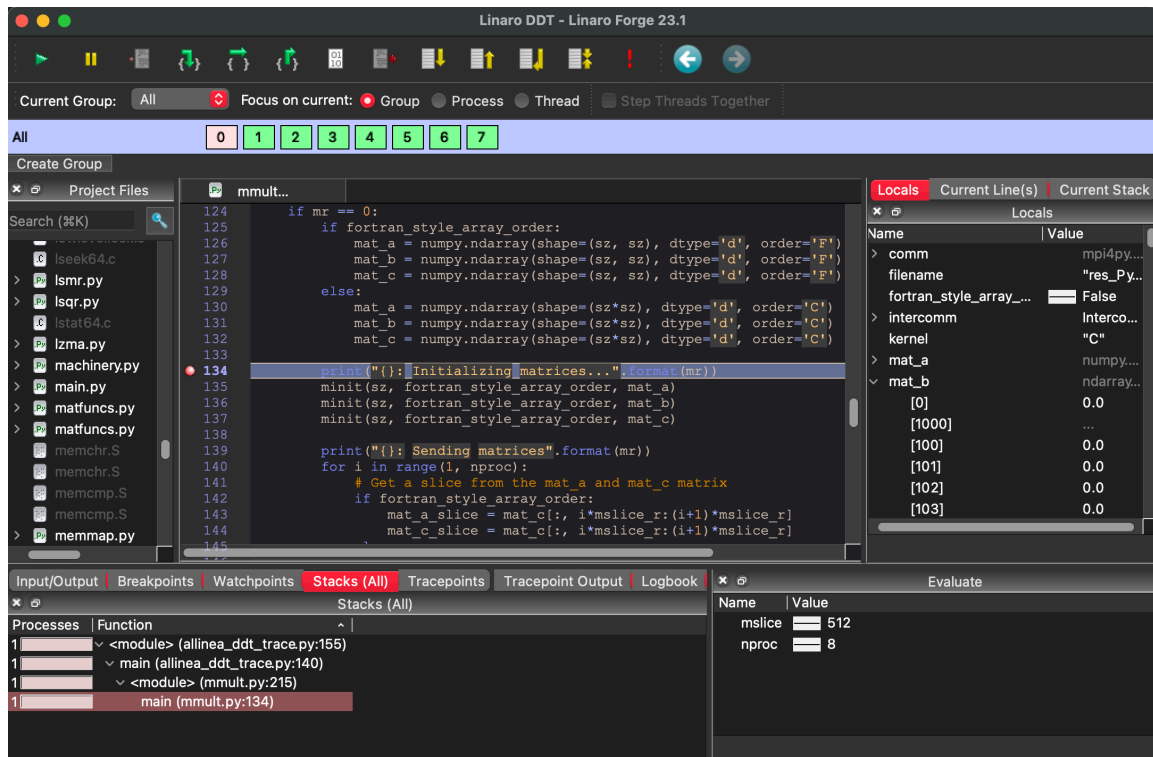
- Improved Evaluations:

- 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

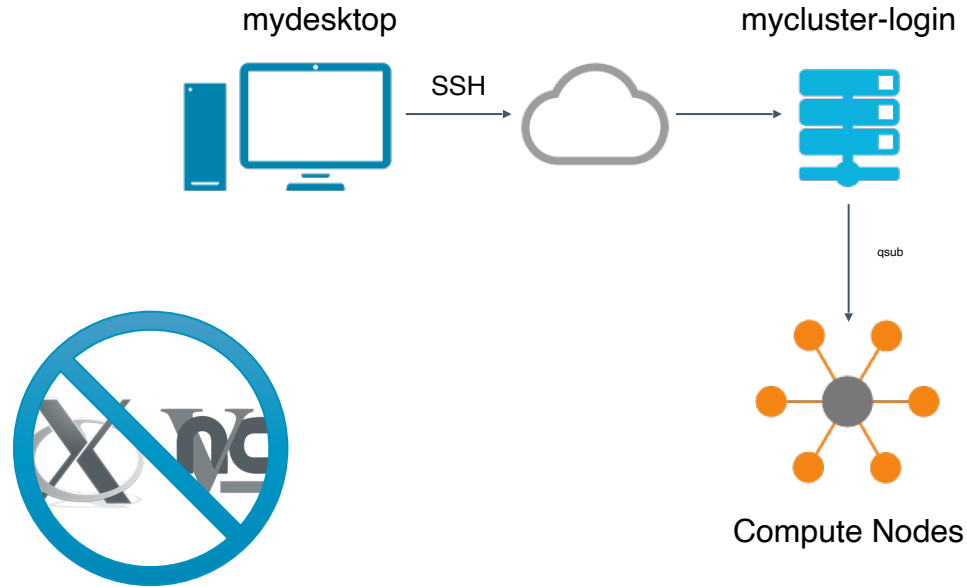
```
ddt --connect srun -n 8 python3  
%allinea_python_debug% ./mmult.py
```





# 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

The screenshot shows the Linaro Forge web interface with a 'Remote Launch Settings' dialog box open. The dialog box contains the following fields and options:

- Connection Name:
- Host Name:  (dropdown menu)
- Remote Installation Directory:
- Remote Script:
- Private Key:  (with folder icon)
- Options:
  - Always look for source files locally
  - Enable
- KeepAlive Packets:  Enable
- Interval:  (dropdown menu)
- Proxy through login node

Buttons: Help, Test Remote Launch, OK, Cancel

The 'Remote Launch: Configure...' button in the main interface is highlighted with a red box.

Left sidebar navigation:

- Linaro Forge
- Linaro DDT (with wrench icon)
- Linaro MAP (with grid icon)
- Get trial licence
- Support
- linaroforge.com
- Remote Client ?

Main content area:

- RUN: Run and debug a program.
- ATTACH: Attach to an already running program.
- OPEN CORE: Open a core file from a previous run.
- MANUAL LAUNCH (ADVANCED): Manually launch the backend yourself.

# 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

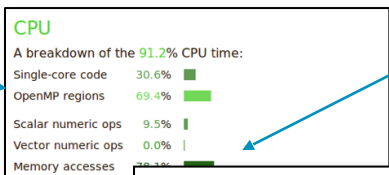
## Adds value to typical users' workflows

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

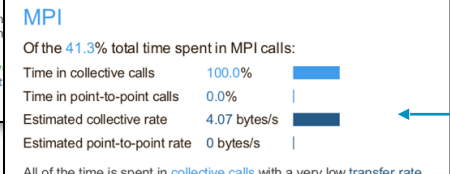
# Linaro Performance Reports Metrics

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

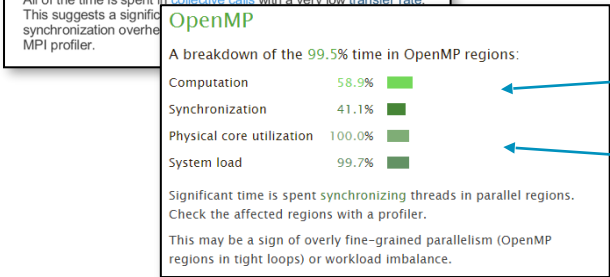
Multi-threaded parallelism



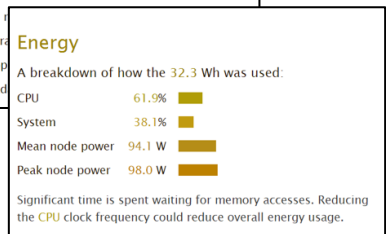
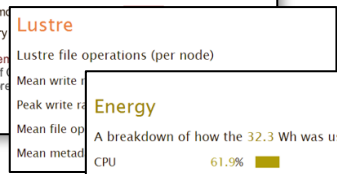
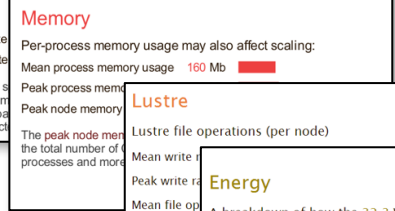
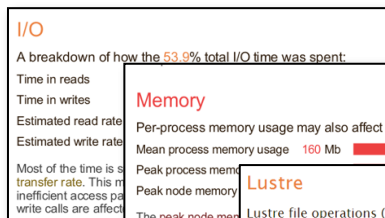
SIMD parallelism



Load imbalance



OMP efficiency  
System usage



# 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.

## 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.

## Workloads

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

## Communication

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

## Memory

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

## Vectorization

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

## Cores

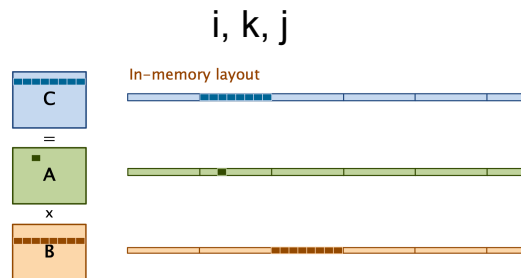
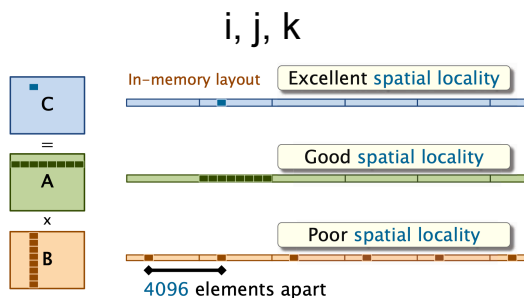
- Discover synchronization overhead and core utilization
- Synchronization-heavy code and implicit barriers are revealed

## Verification

- Validate corrections and optimal performance



# Performance Improvement



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];
        }
    }
}
```



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, 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];
        }
    }
}
```

# MAP Capabilities

MAP is a sampling based scalable profiler

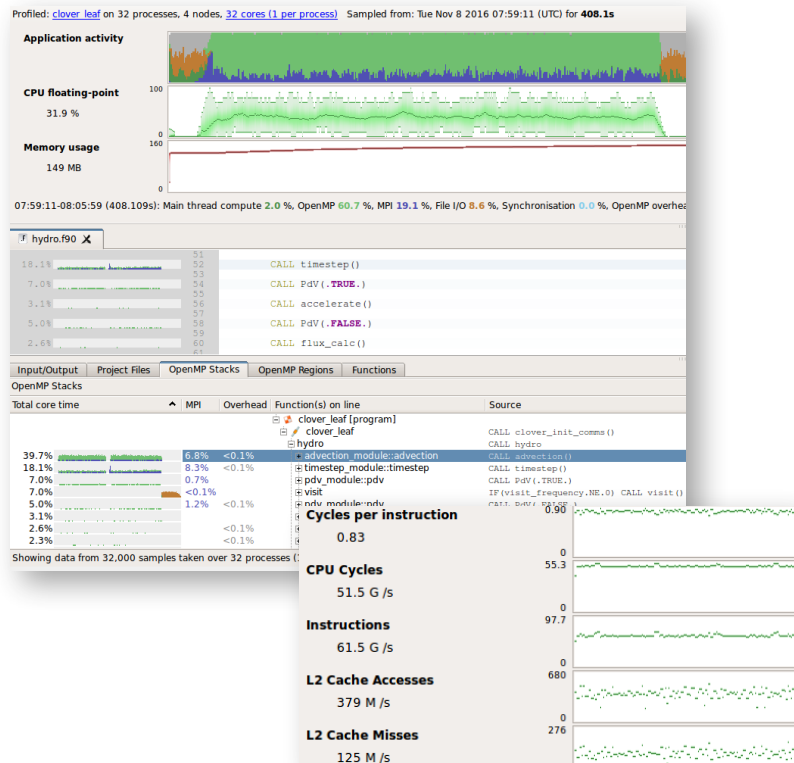
- Built on same framework as DDT
- Parallel support for MPI, OpenMP, CUDA
- Designed for C/C++/Fortran

Designed for 'hot-spot' analysis

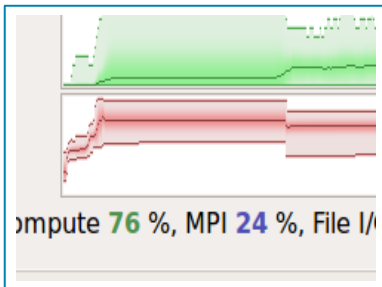
- Stack traces
- Augmented with performance metrics

Adaptive sampling rate

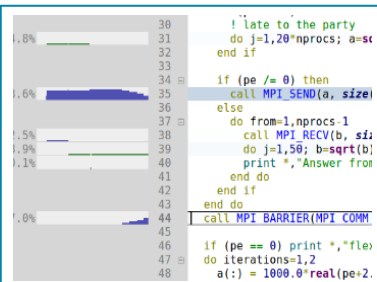
- Throws data away - 1,000 samples per process
- Low overhead, scalable and small file size



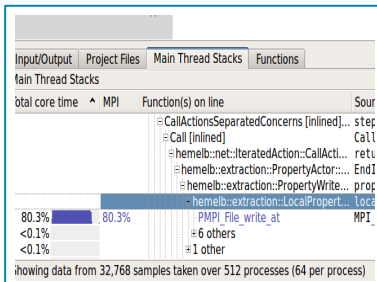
# Linaro MAP Source Code Profiler Highlights



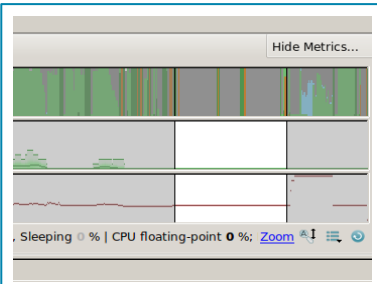
Find the peak memory use



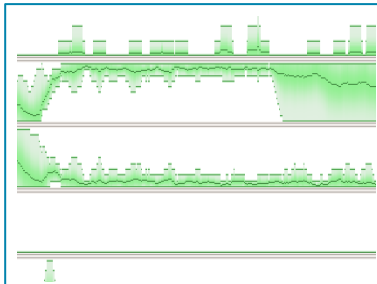
Fix an MPI imbalance



Remove I/O bottleneck



Make sure OpenMP regions make sense



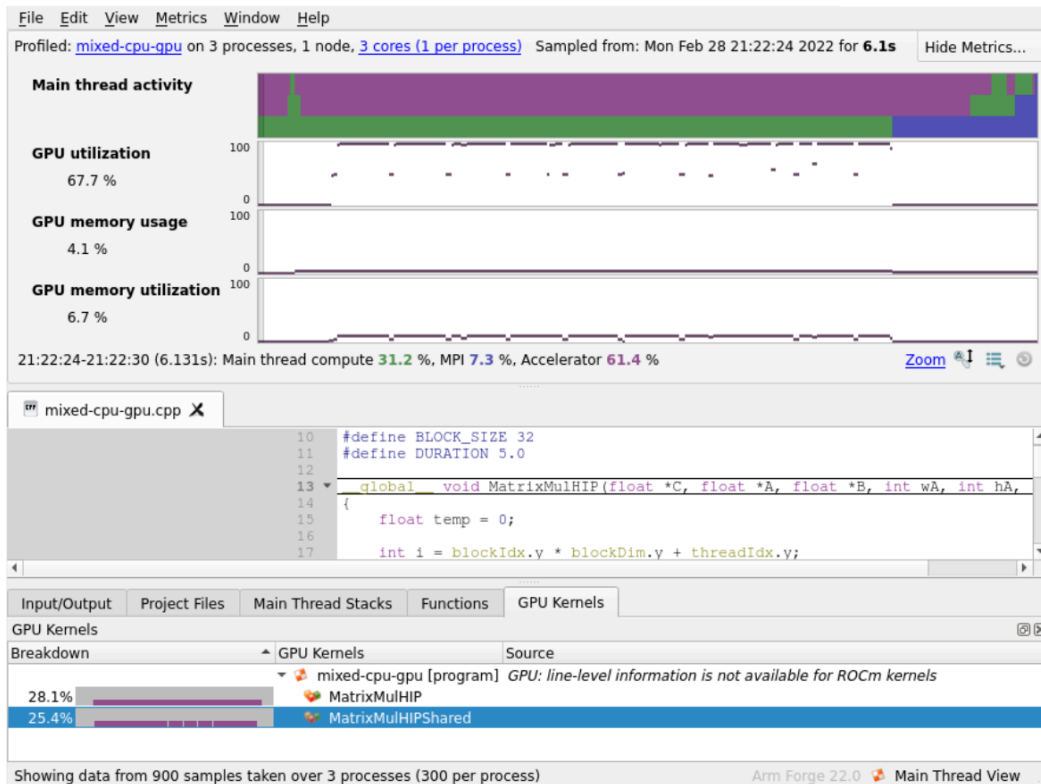
Improve memory access

```
{
  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

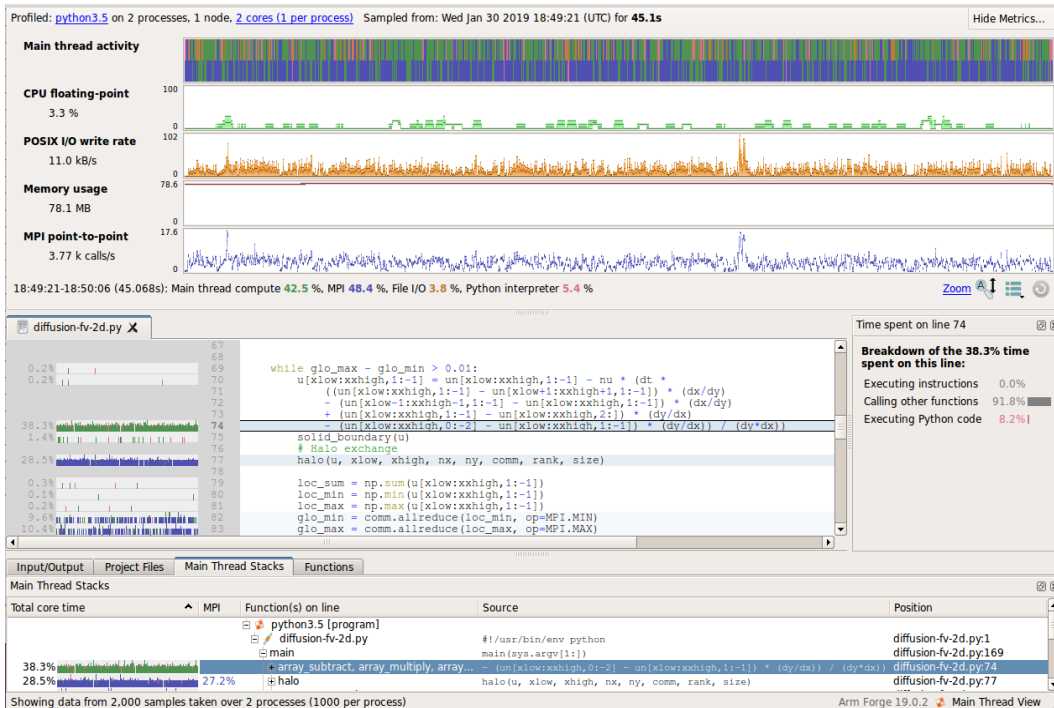
Works with MPI4PY

- 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 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](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
```



**Thank you**

[rudy.shand@linaro.org](mailto:rudy.shand@linaro.org)

Linaro Forge