linaroforge

Debugging and Optimizing Parallel Codes with Linaro Forge

Rudy Shand
Field Application Engineer

linaroforge

Agenda

- 10:00 - 12:00   Session 1: Ensuring Program Correctness with Linaro DDT
- 13:00 - 15:00   Session 2: Performance Engineering with Linaro Performance Reports and Linaro MAP

Bug classification

- Crashes
  - One or more processes in application terminates
  - Most common and generally easiest to solve
- Hangs
  - Deadlocks - Stuck waiting for something that never happens
  - Livelocks - Making local progress, but no global progress
- Race conditions
  - One or more threads accessing the same data at the same time in non deterministic way
  - Shows up as incorrect answer or sometimes crashes

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

Linaro Forge: Where Most of Top Supercomputers turn for Performance Excellence

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.

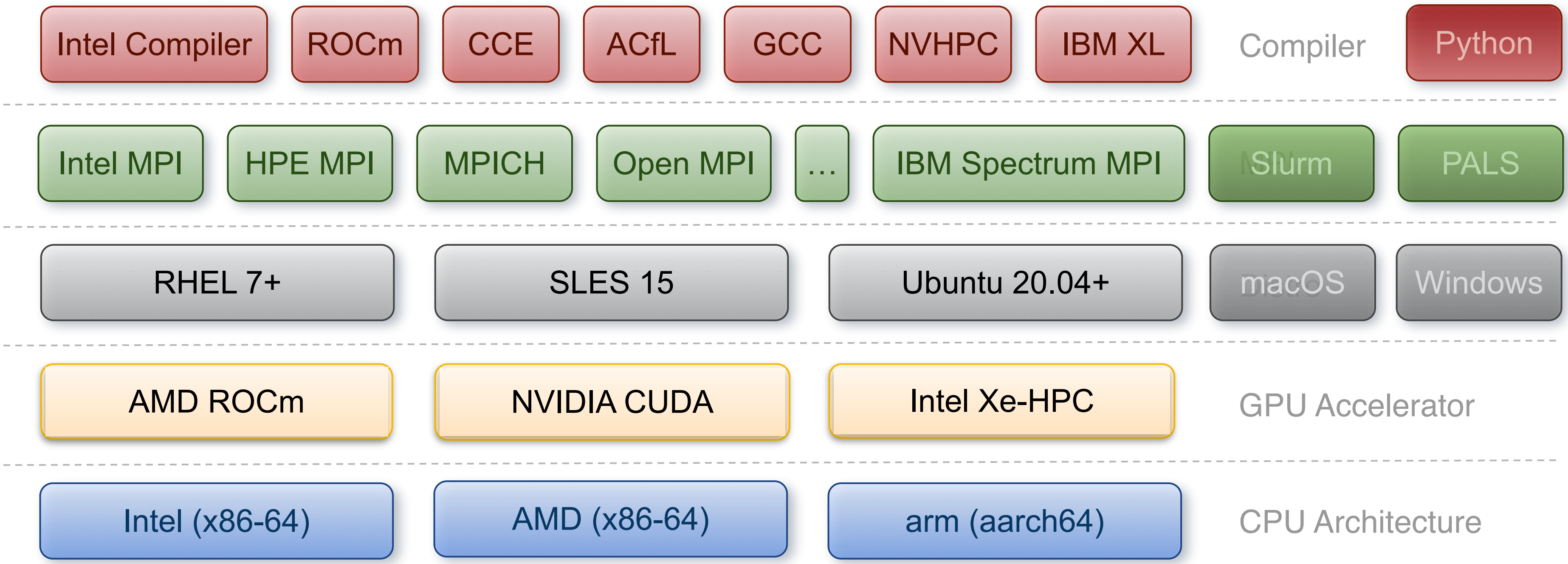
Linaro Performance Reports

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

Performance Engineering for any architecture, at any scale

linaroforge

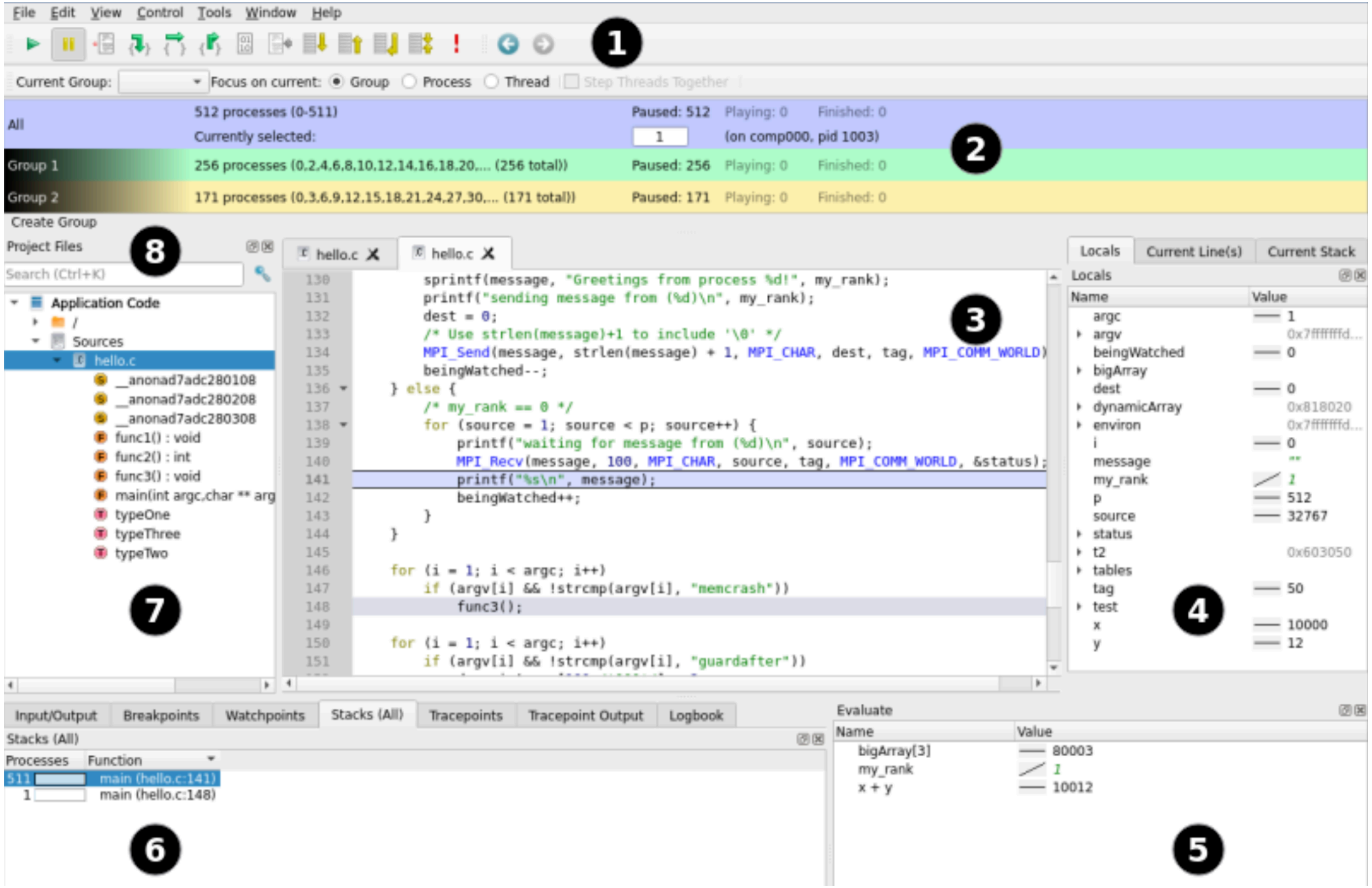
Supported Platforms



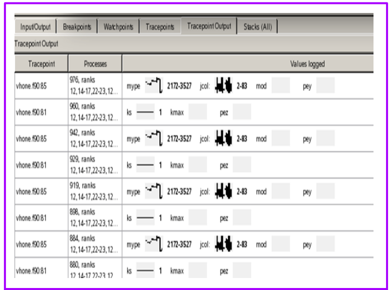
linaroforge

DDT UI

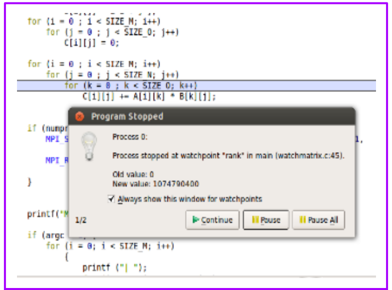
- 1 Process controls
- 2 Process groups
- 3 Source Code view
- 4 Variables
- 5 Evaluate window
- 6 Parallel Stack
- 7 Project files
- 8 Find a file or function



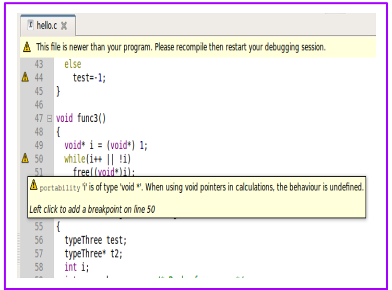
Linaro DDT Debugger Highlights



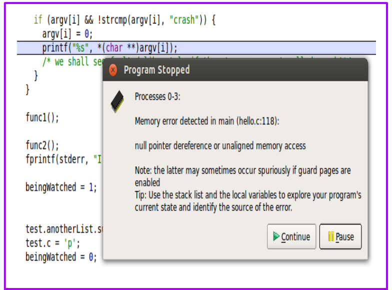
The scalable print alternative



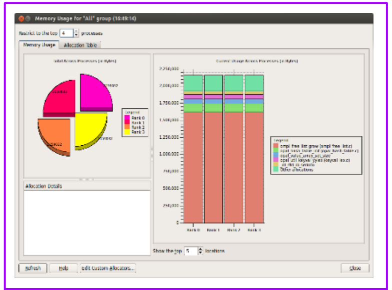
Stop on variable change



Static analysis warnings on code errors



Detect read/write beyond array bounds



Detect stale memory allocations

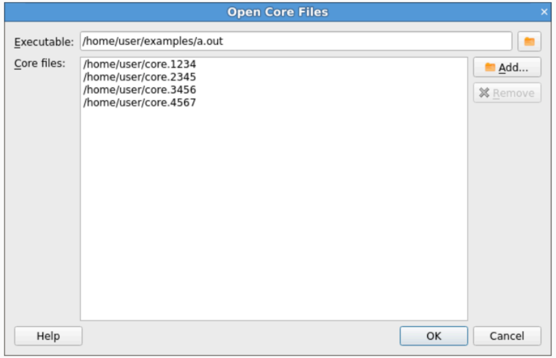
linaroforge

Core files

You can open and debug one or more core files generated by your application.

Procedure

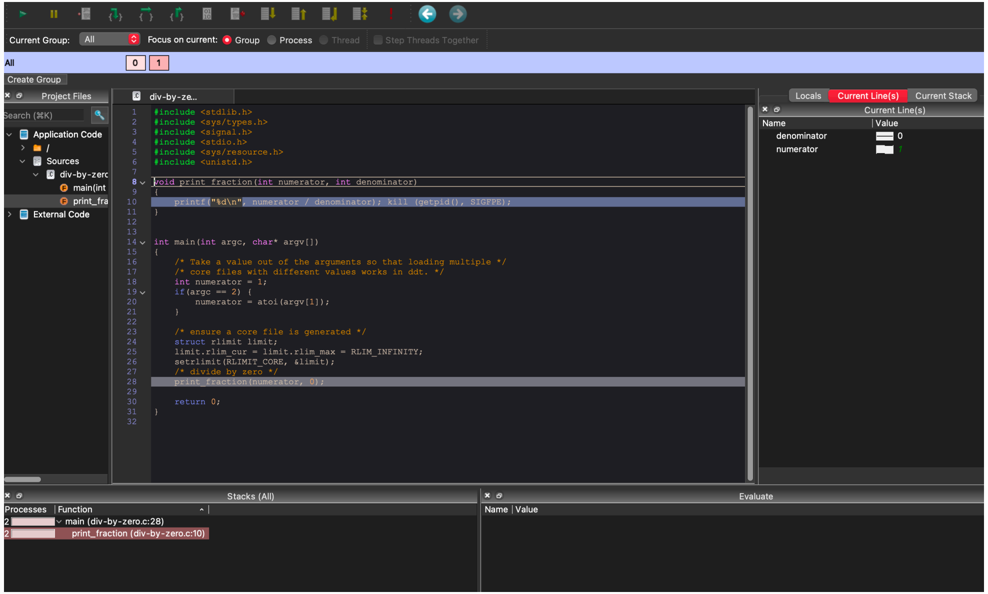
1. On the Welcome page click **Open Core Files**. The **Open Core Files** window opens.



2. Select an executable and a set of core files, then click **OK** to open the core files and start debugging them.

Note

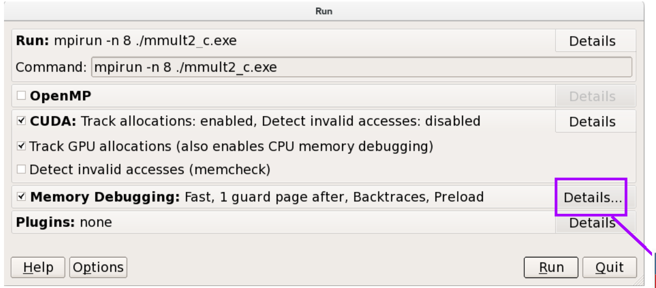
While Linaro DDT is in this mode, you cannot play, pause, or step, because there is no process active. You are, however, able to evaluate expressions and browse the variables and stack frames saved in the core files.



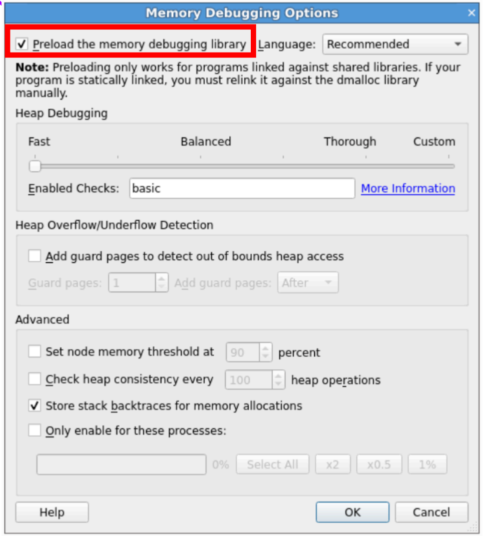
- View core files for CPU’s
- View core files for GPU’s

linaroforge

Memory debugging menu in Linaro DDT



When manual linking is used, untick “Preload” box



linaroforge

Multi-dimensional Array Viewer

What does your data look like at runtime?

View arrays

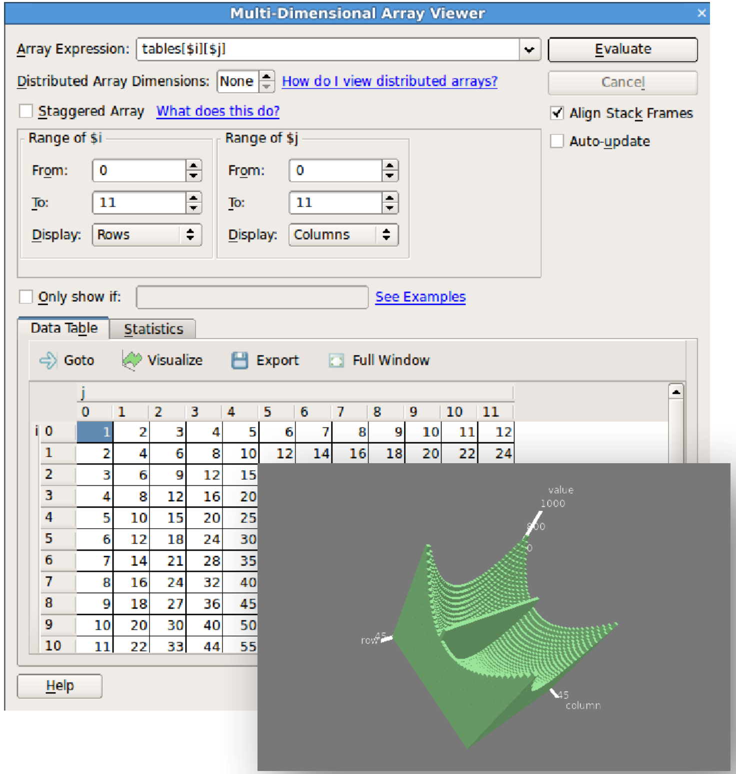
- On a single process
- Or distributed on many ranks

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



DDT: Production-scale debugging

Isolate and investigate faults at scale

Who misbehaved?

- Merge stacks from processes and threads
- Sparklines comparing data across processes
- Which MPI rank

Where is the problem?

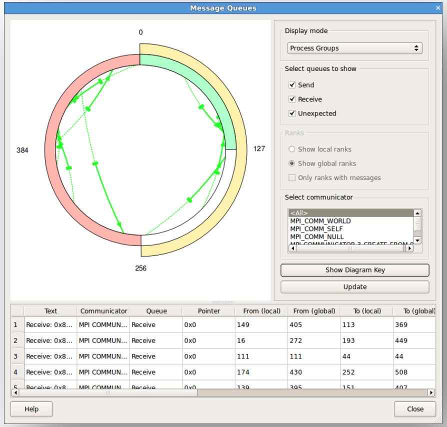
- Integrated source code editor
- Dynamic data structure visualization

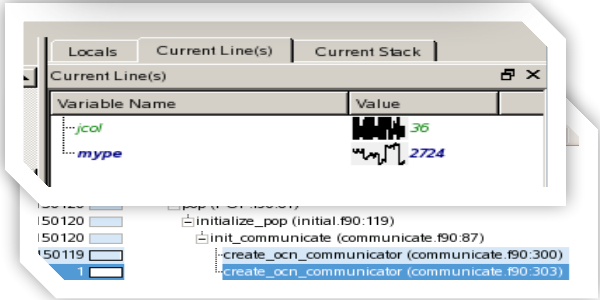
How did it happen?

- Parse diagnostic messages
- Trace variables through execution

Why did it happen?

- Unique “Smart Highlighting”
- Experiment with variable values

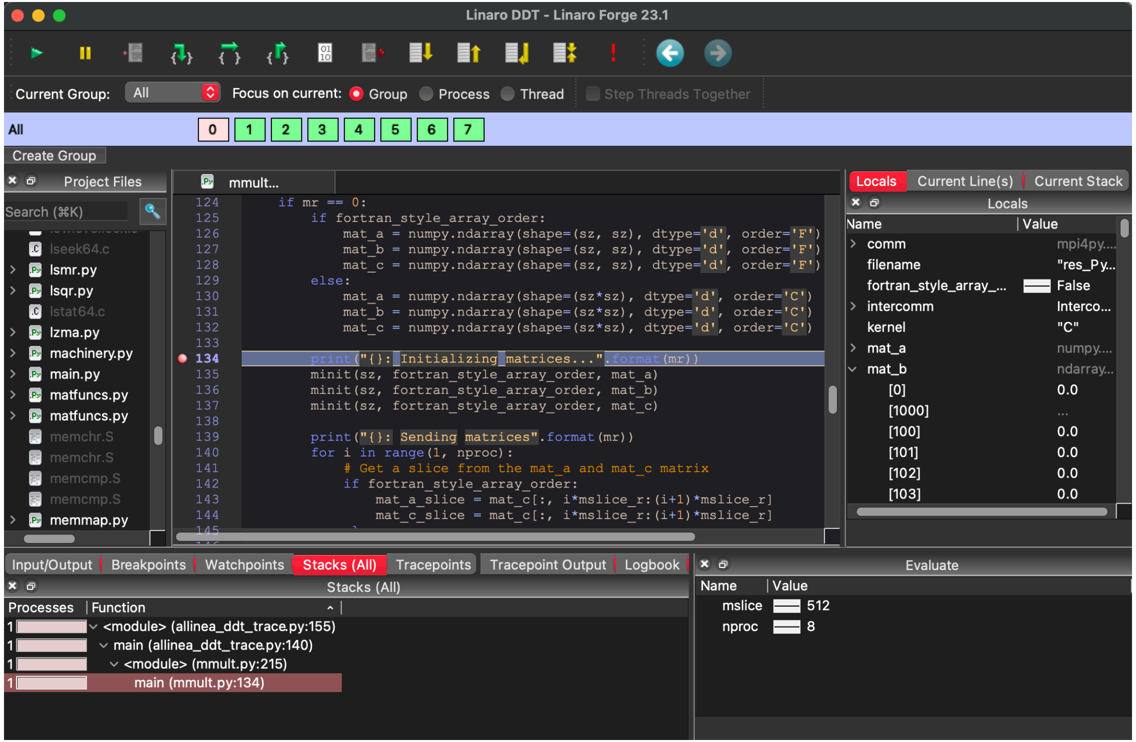




linaroforge

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 mpirun -n 8 python3 **%allinea_python_debug%** ./mmult.py

linaroforge

Debugging Nvidia GPUs

Using Linaro DDT

Debug code simultaneously on the A100 GPU and the CPU

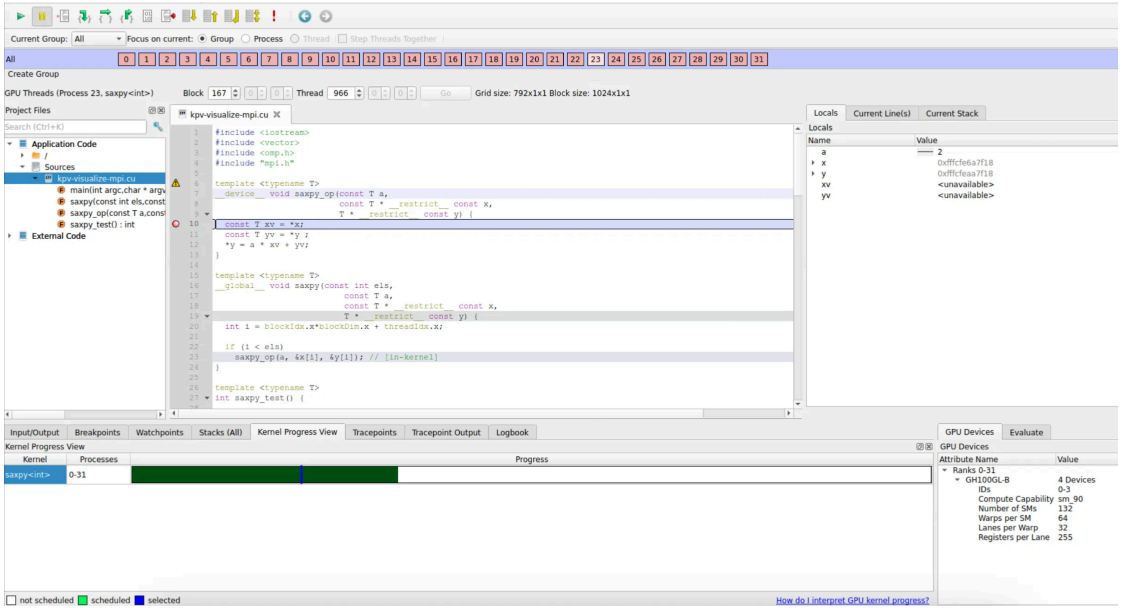
Controlling the GPU execution:

- 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



Run 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
    - 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
```

Report output

12		0:08.188	0-3	Process stopped at breakpoint in update (wave.c:216).
13				Additional Information

▼ Stacks

Processes	Threads	Function	Source	Variables
0-3	4	main (wave.c:334)	▶ iterations = update(left, right);	▶ Rank 0, thread 1
0-3	4	update (wave.c:216)	▶ values[j] = newval[j];	▼ Rank 0, thread 1
0-3	8	progress_engine		
0-3	8	opal_libevent2022_event_base_loop (event.c:1630)		▶ Rank 0, thread 2
0-3	4	poll_dispatch (poll.c:165)		▶ Rank 0, thread 2
0-3	4	poll		
0-3	4	epoll_dispatch (epoll.c:407)		▶ Rank 0, thread 3
0-3	4	epoll_wait		

Name	Value
i	0
iterations	1
j	101
left	*-2 (from -2 to 2)*
now	<aggregate value>
right	*1 (from -2 to 3)*
stop	0

▶ Current Stack

▼ Evaluate

Name	Value
3*j*j	30603
j	101

14		0:11.009	0-3	Play

linaroforge

9 Step Guide

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

Cores

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

Vectorization

- 🟠 Understand numerical intensity and vectorization level.
- 🔵 Hot loops, unvectorized code and GPU performance reveleaed

Verification

- 🟠🔵 Validate corrections and optimal performance

Key : 🔵 Linaro Forge
🟠 Linaro Performance Reports

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)

linaroforge

Linaro Performance Reports

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

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.

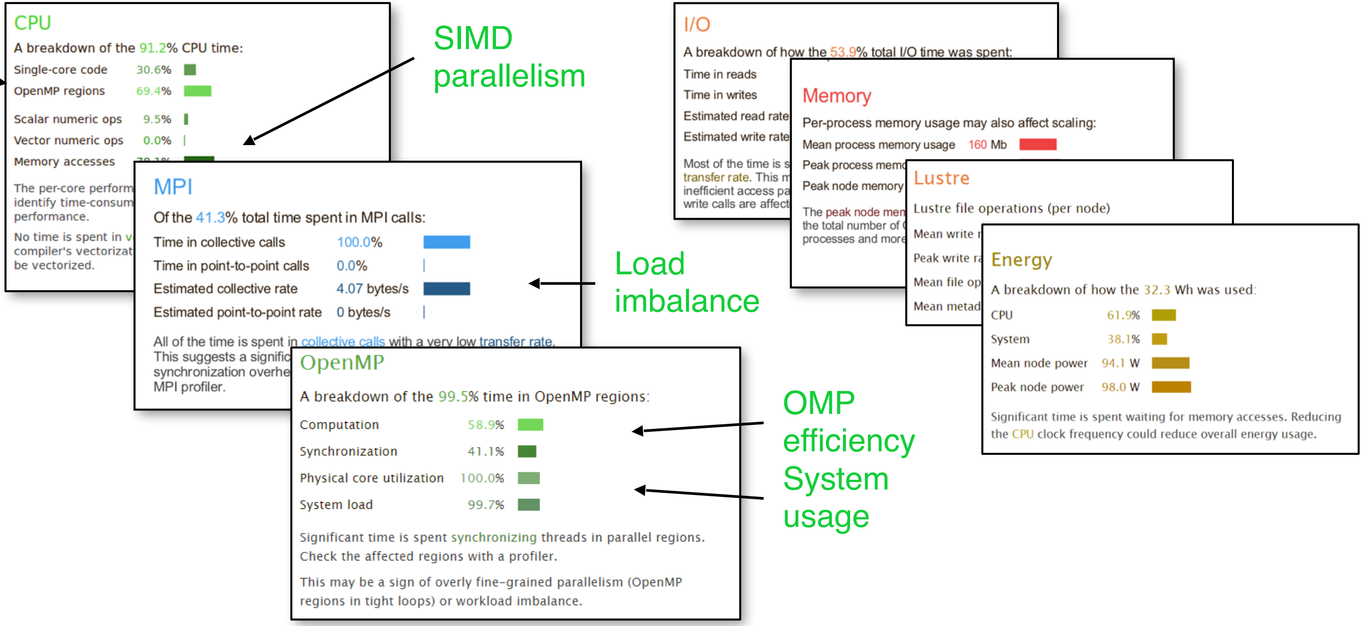
linaroforge

Linaro Performance Reports Metrics

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

Multi-threaded parallelism

SIMD parallelism



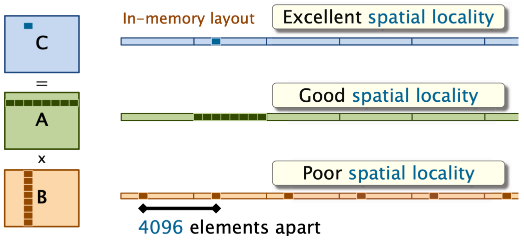
Load imbalance

OMP efficiency

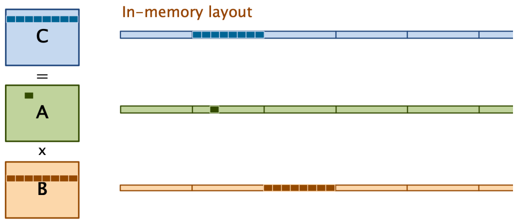
System usage

Performance Improvement

i, j, k



i, k, j



Think, code, 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

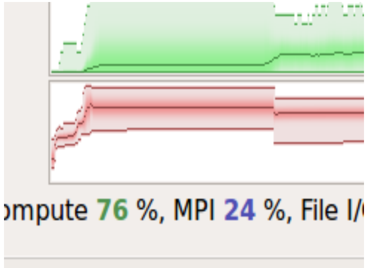
i, j, k

```
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];
    }
  }
}
```

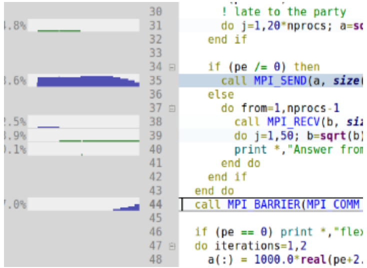
i, k, j

```
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];
    }
  }
}
```

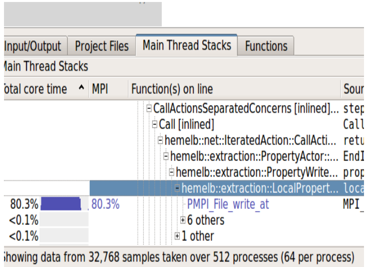
Linaro MAP Source Code Profiler Highlights



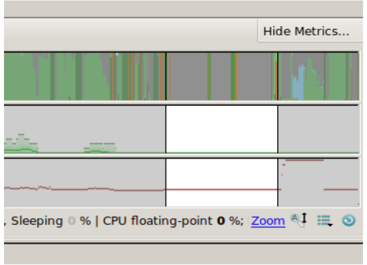
Find the peak memory use



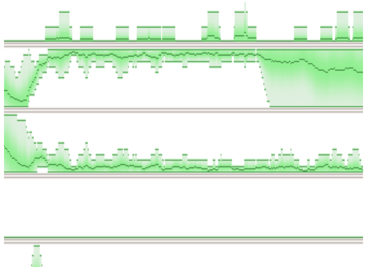
Fix an MPI imbalance



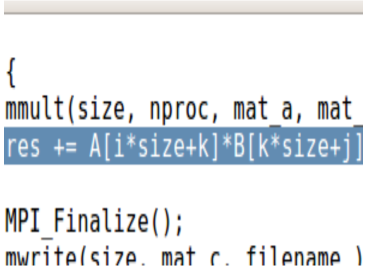
Remove I?O bottleneck



Make sure OpenMP regions make sense



Improve memory access



Restructure for vectorization

linaroforge

MAP Capabilities

MAP is a sampling based scalable profiler

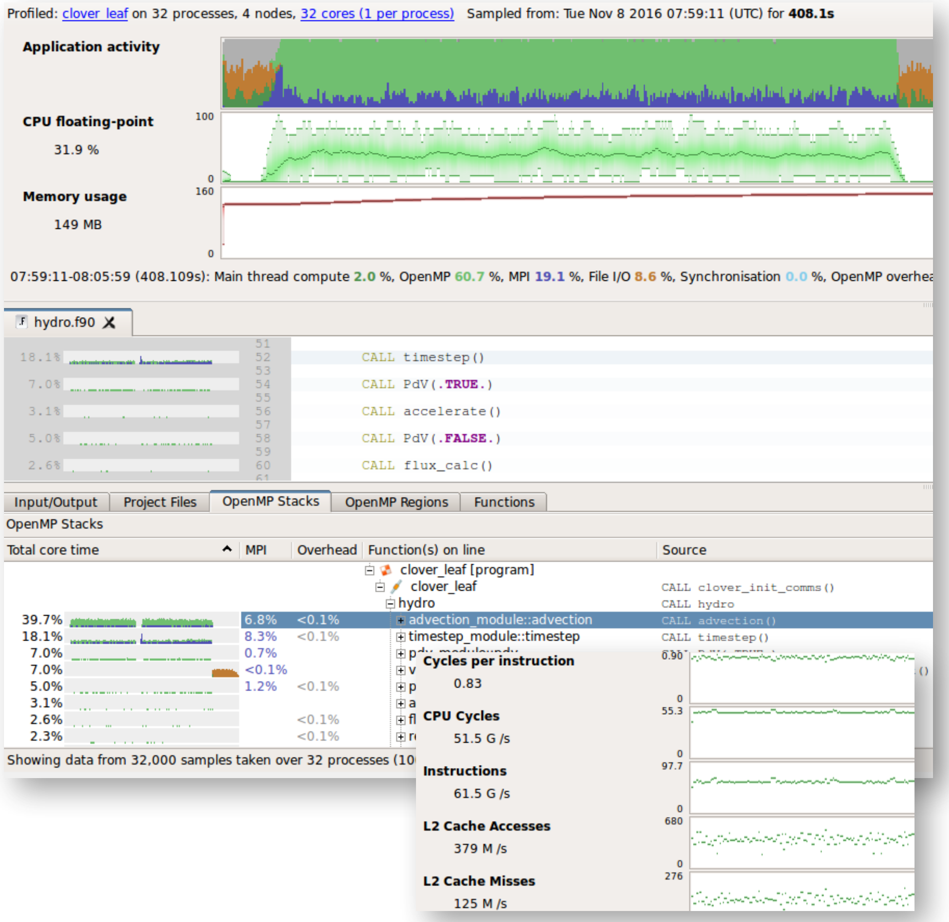
- Built on same framework as DDT
- Parallel support for MPI, OpenMP, CUDA
- Designed for C/C++/Fortran

Designed for ‘hot-spot’ analysis

- Stack traces
- Augmented with performance metrics

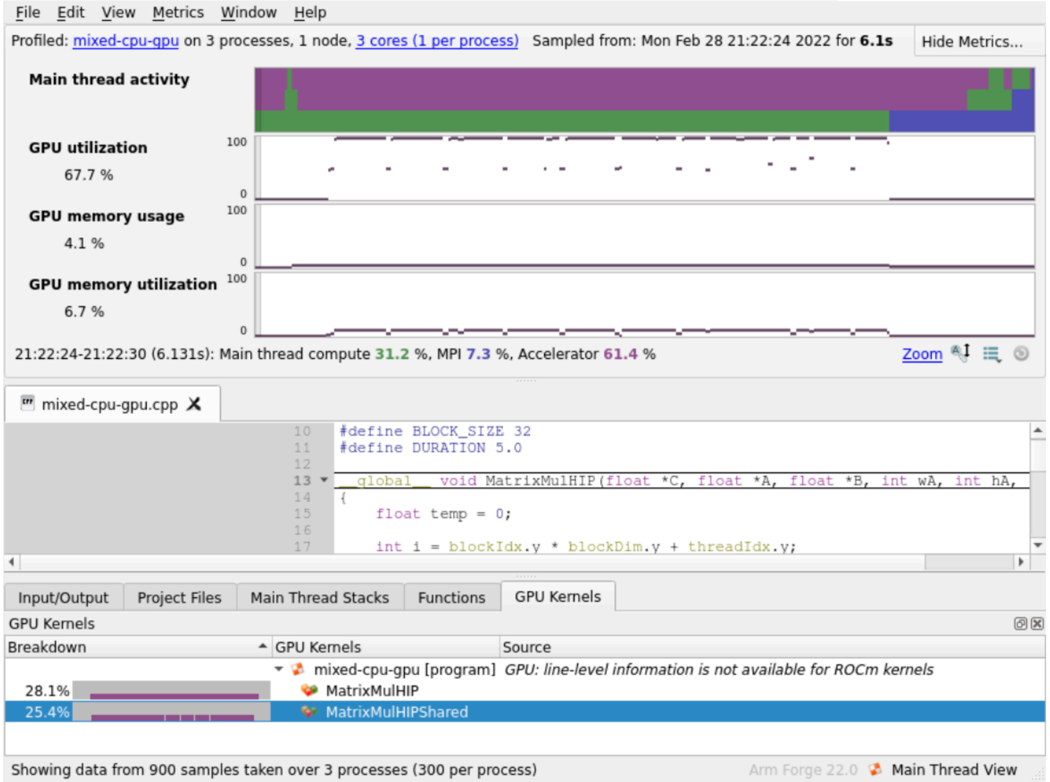
Adaptive sampling rate

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



linaroforge

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

linaroforge

Python Profiling

19.0 adds support for Python

- Call stacks
- Time in interpreter

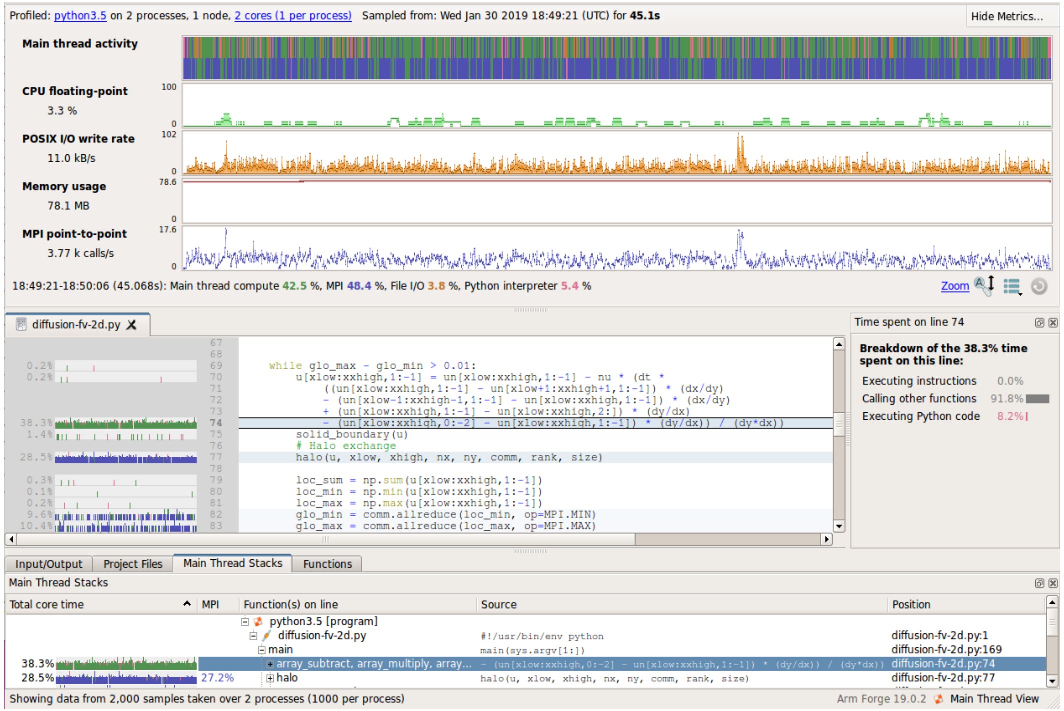
Works with MPI4PY

- Usual MAP metrics

Source code view

- Mixed language support

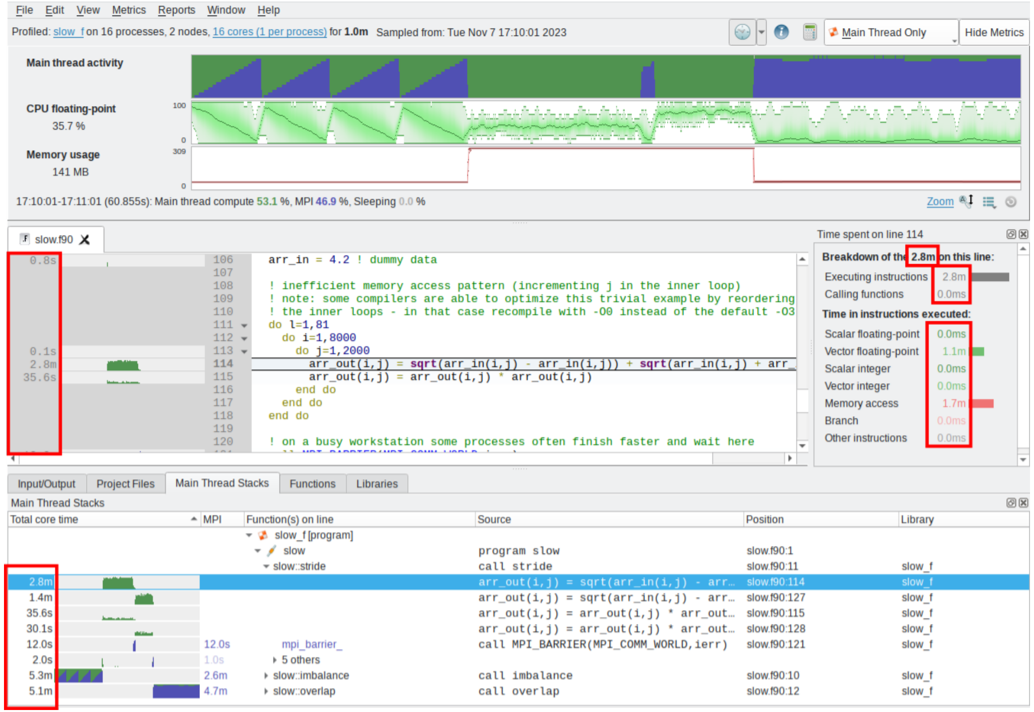
Note: Green as operation is on numpy array, so backed by C routine, not Python (which would be pink)




```
map --profile mpirun -n 2 python ./diffusion-fv-2d.py
```

linaroforge

Toggle percentage-time and core-time in MAP



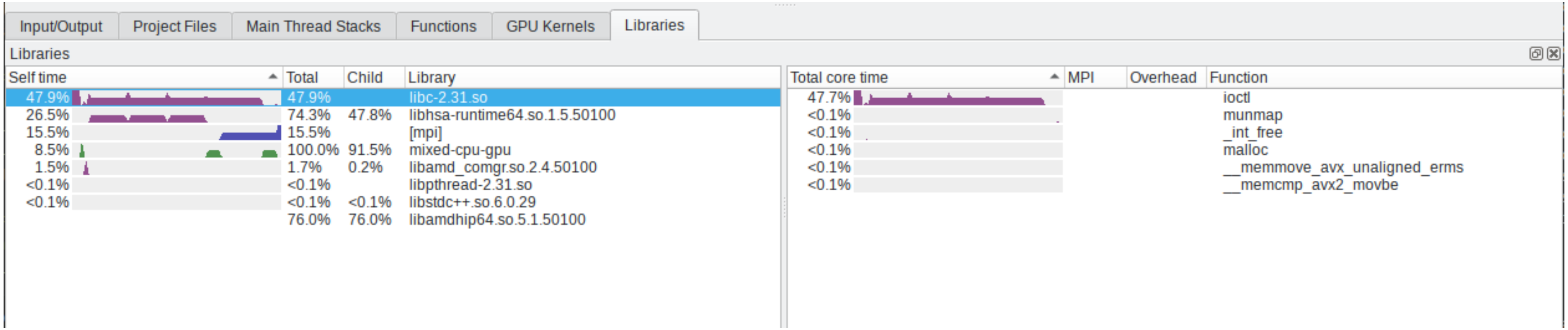
Use for direct comparisons between runs at the same scale (process/core counts).

- Easily determine if a change has made a portion of code faster, slower, or largely unchanged.
- Performance report automatically includes both percentage-time and core time
- Core-time is an estimation, but should be very close to the application run time

linaroforge

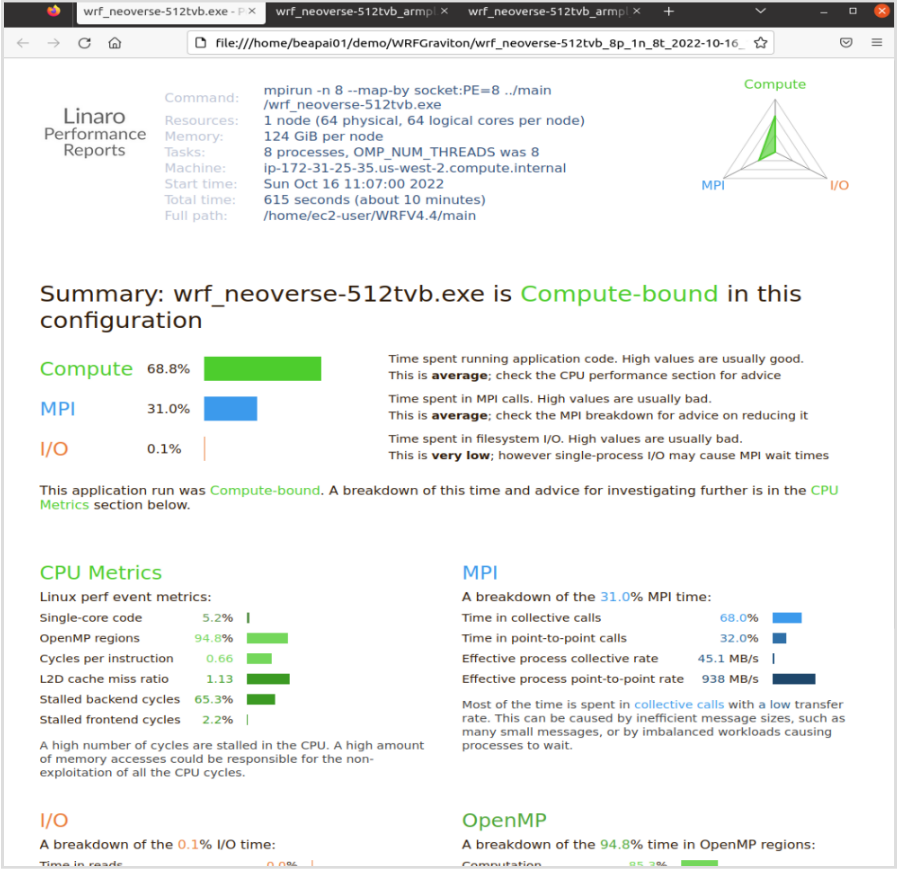
Libraries tab in MAP

- List time spent in shared libraries (left)
- List entry point functions into the selected library (right)



Use to identify the libraries that would benefit the most from optimisation or replacement (e.g. alternative maths library or memory management implementation).

WRF build without enhancements


wrf_neoverse-512tvb.exe - P × | wrf_neoverse-512tvb_armpl × | wrf_neoverse-512tvb_armpl × | +

file:///home/beapai01/demo/WRFGraviton/wrf_neoverse-512tvb_8p_1n_8t_2022-10-16_

Linaro Performance Reports

Command:	mpirun -n 8 --map-by socket:PE=8 ../main /wrf_neoverse-512tvb.exe
Resources:	1 node (64 physical, 64 logical cores per node)
Memory:	124 GiB per node
Tasks:	8 processes, OMP_NUM_THREADS was 8
Machine:	ip-172-31-25-35.us-west-2.compute.internal
Start time:	Sun Oct 16 11:07:00 2022
Total time:	615 seconds (about 10 minutes)
Full path:	/home/ec2-user/WRFV4.4/main

Summary: wrf_neoverse-512tvb.exe is Compute-bound in this configuration

Compute	68.8%	Time spent running application code. High values are usually good. This is **average**; check the CPU performance section for advice
MPI	31.0%	Time spent in MPI calls. High values are usually bad. This is **average**; check the MPI breakdown for advice on reducing it
I/O	0.1%	Time spent in filesystem I/O. High values are usually bad. This is **very low**; however single-process I/O may cause MPI wait times

This application run was Compute-bound. A breakdown of this time and advice for investigating further is in the CPU Metrics section below.

CPU Metrics

Linux perf event metrics:

Single-core code	5.2%
OpenMP regions	94.8%
Cycles per instruction	0.66
L2D cache miss ratio	1.13
Stalled backend cycles	65.3%
Stalled frontend cycles	2.2%

A high number of cycles are stalled in the CPU. A high amount of memory accesses could be responsible for the non-exploitation of all the CPU cycles.

MPI

A breakdown of the 31.0% MPI time:

Time in collective calls	68.0%
Time in point-to-point calls	32.0%
Effective process collective rate	45.1 MB/s
Effective process point-to-point rate	938 MB/s

Most of the time is spent in collective calls with a low transfer rate. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait.

I/O

A breakdown of the 0.1% I/O time:

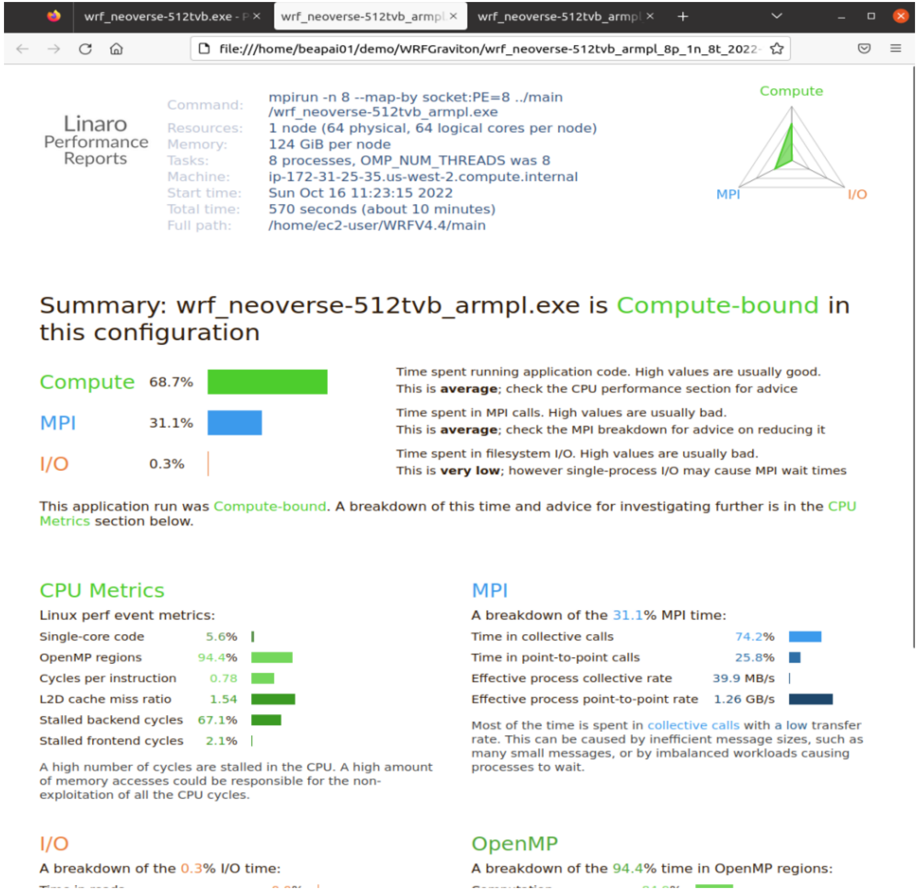
Time in reads 0.0%

OpenMP

A breakdown of the 94.8% time in OpenMP regions:

Computation 85.3%

WRF with Arm Performance Libraries


wrf_neoverse-512tvb.exe - P | wrf_neoverse-512tvb_armpl | wrf_neoverse-512tvb_armpl

file:///home/beapai01/demo/WRFGraviton/wrf_neoverse-512tvb_armpl_8p_1n_8t_2022-

Linaro Performance Reports

Command:	mpirun -n 8 --map-by socket:PE=8 ../main /wrf_neoverse-512tvb_armpl.exe
Resources:	1 node (64 physical, 64 logical cores per node)
Memory:	124 GiB per node
Tasks:	8 processes, OMP_NUM_THREADS was 8
Machine:	ip-172-31-25-35.us-west-2.compute.internal
Start time:	Sun Oct 16 11:23:15 2022
Total time:	570 seconds (about 10 minutes)
Full path:	/home/ec2-user/WRFV4.4/main

Summary: wrf_neoverse-512tvb_armpl.exe is Compute-bound in this configuration

Compute	68.7%	Time spent running application code. High values are usually good. This is **average**; check the CPU performance section for advice
MPI	31.1%	Time spent in MPI calls. High values are usually bad. This is **average**; check the MPI breakdown for advice on reducing it
I/O	0.3%	Time spent in filesystem I/O. High values are usually bad. This is **very low**; however single-process I/O may cause MPI wait times

This application run was Compute-bound. A breakdown of this time and advice for investigating further is in the CPU Metrics section below.

CPU Metrics

Linux perf event metrics:

Single-core code	5.6%
OpenMP regions	94.4%
Cycles per instruction	0.78
L2D cache miss ratio	1.54
Stalled backend cycles	67.1%
Stalled frontend cycles	2.1%

A high number of cycles are stalled in the CPU. A high amount of memory accesses could be responsible for the non-exploitation of all the CPU cycles.

MPI

A breakdown of the 31.1% MPI time:

Time in collective calls	74.2%
Time in point-to-point calls	25.8%
Effective process collective rate	39.9 MB/s
Effective process point-to-point rate	1.26 GB/s

Most of the time is spent in collective calls with a low transfer rate. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait.

I/O

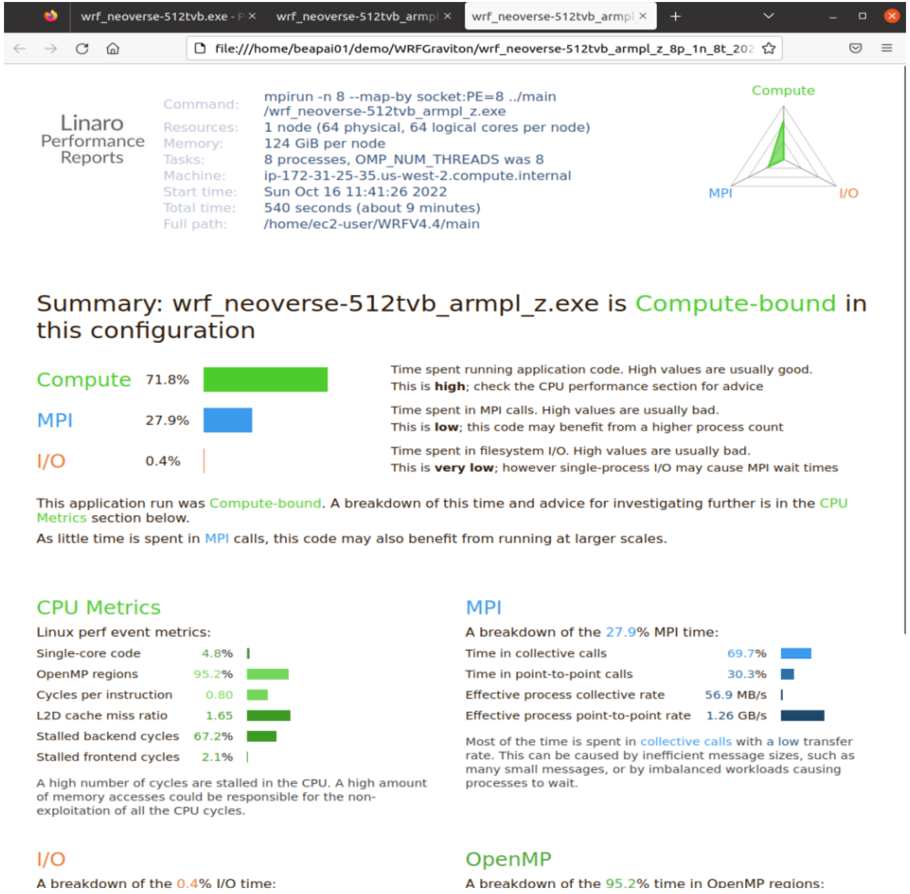
A breakdown of the 0.3% I/O time:

OpenMP

A breakdown of the 94.4% time in OpenMP regions:

Linaro Forge

WRF with Arm Performance Libraries and IO compression libraries



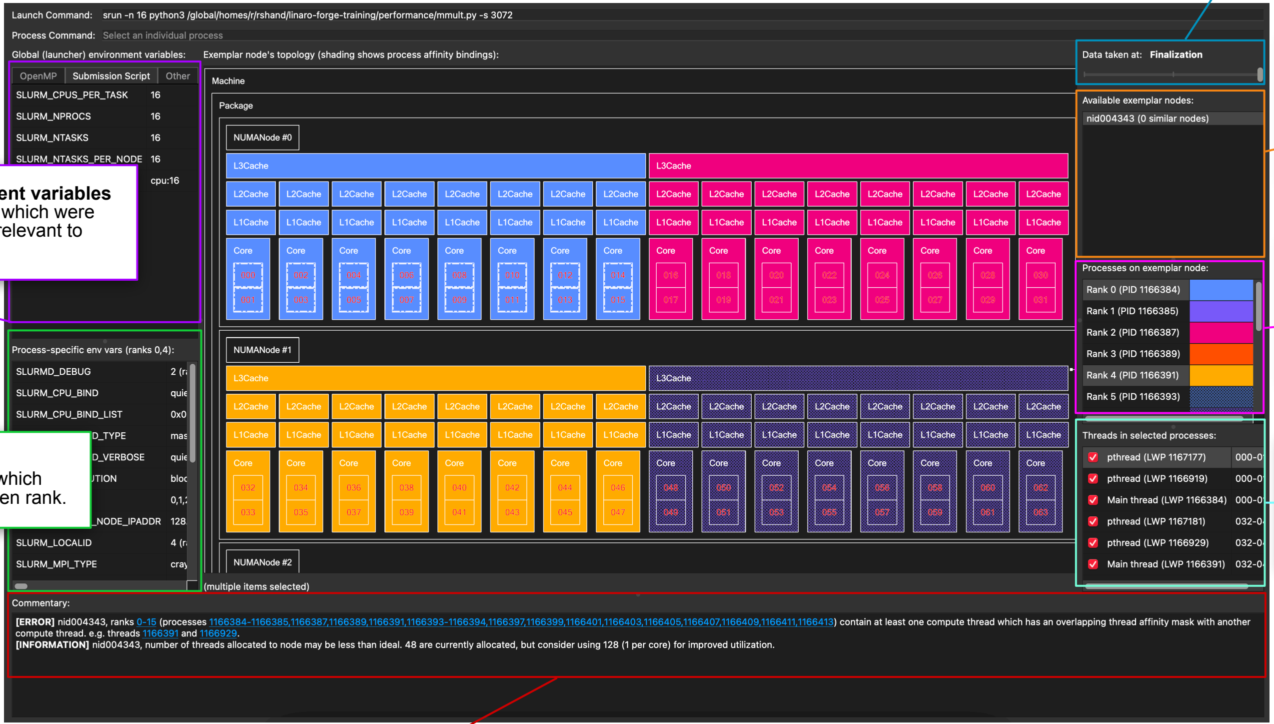
linaroforge

MAP Thread Affinity Advisor

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.

Global (launcher) environment variables
List of Environment Variables which were set at launch which might be relevant to how threads are distributed.



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.

Process-specific env vars
List of Environment Variables which might affect the affinity of a given rank.

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.

linaroforge

Thank you

rudy.shand@linaro.org

support@forge.linaro.com

Go to www.linaroforge.com