linaroforge

Finding and fixing bugs on Hunter with Linaro DDT

Rudy Shand
Technical Services and Support



Agenda (Thursday, CEST)

10:00 - 11:00 Debugging with Linaro DDT (Lecture)

- 40 minutes of slides
- 20 minutes of demo

11:00 - 13:00 DDT hands-on (Hunter)

- 10 minutes setup
- 15 minutes saxpy GPU debugging example
- 10 minutes: Introduction to the CloverLeaf example
- 80 minutes CloverLeaf bugs, 4 bugs, 20 minutes each
- 5 minutes round-up

14:00 - 15:00 Apply DDT to your own code

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

Incorrect result

- Runs to the end, but the answer is wrong

linaroforge

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

GDB Under the hood

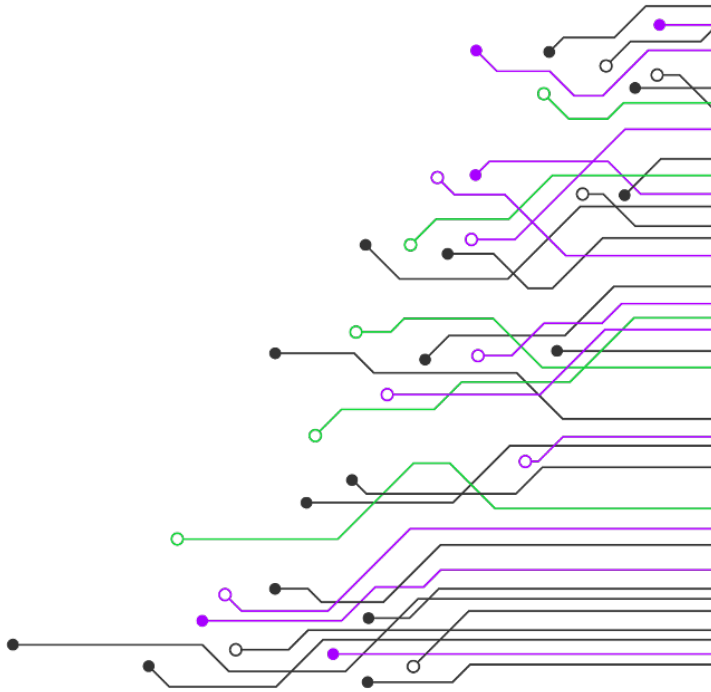
Leveraging the gdb community

GDB is the underlying DDT debugger

- Linaro upstreams patches to the GDB community
- Forge team raises and fixes GDB bugs

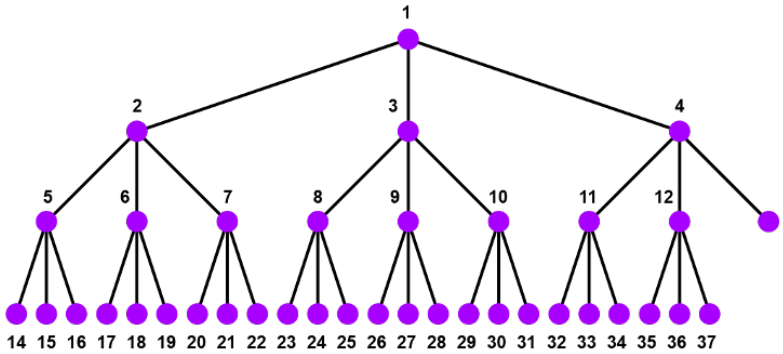
Nimble at supporting new technologies

- Rely on Hardware vendors to add GDB support
- Rely on Software consortiums to add GDB support
- Helps us to stay current with technologies
- Provides a state of the art debugger



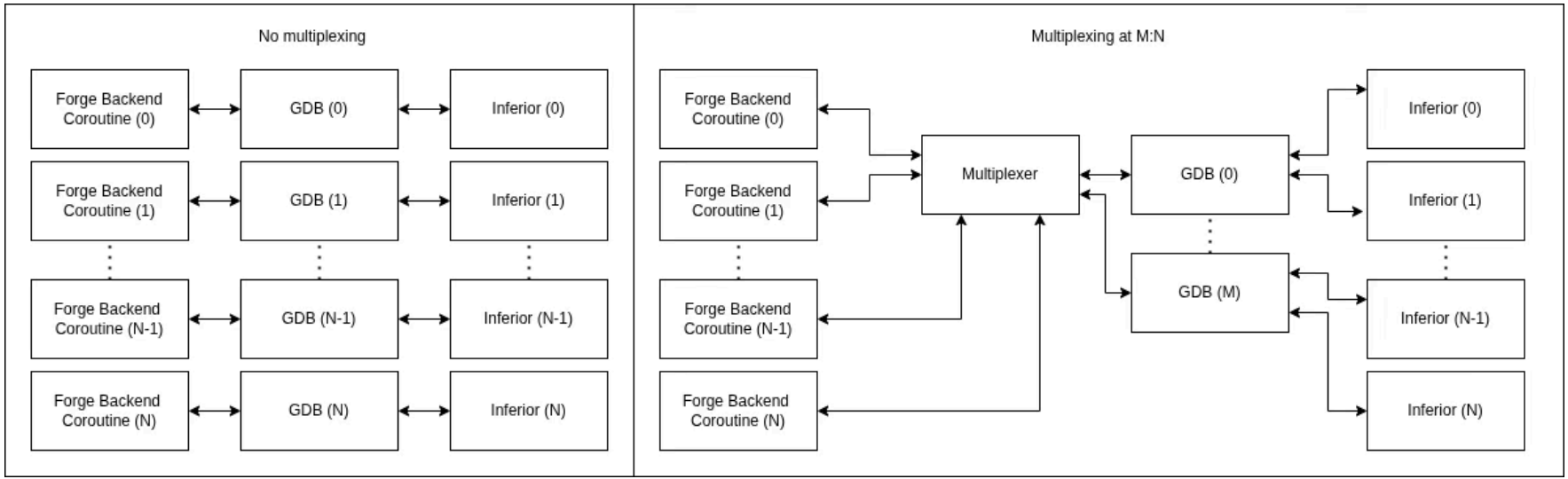
linaroforge

Scaling gdb



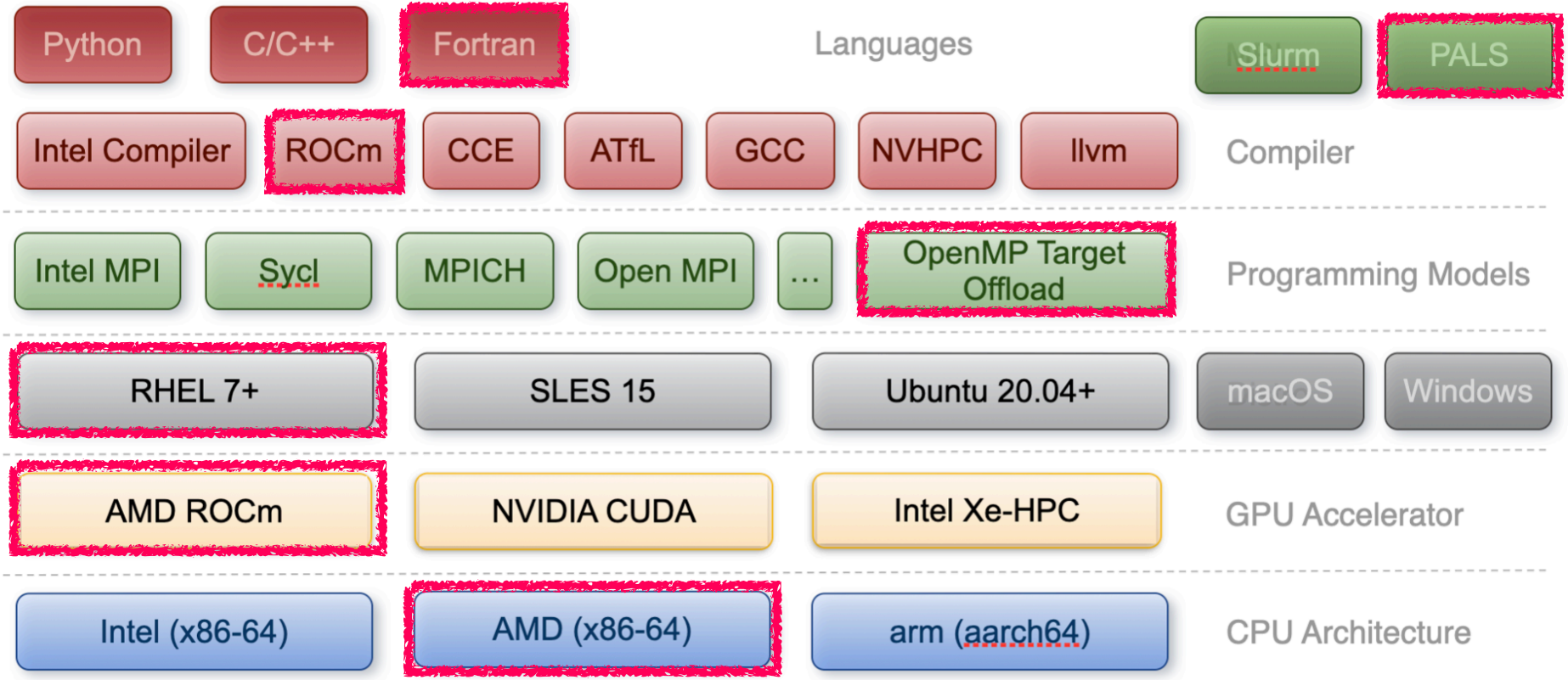
Tree network topology

- Tree server design is how Forge is able to scale
- Send bulk commands and merge responses
- Aggregate the data instead of broadcasting thousands of responses
- Configurable gdb inferior multiplexing



linaroforge

Supported Platforms



linaroforge

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?

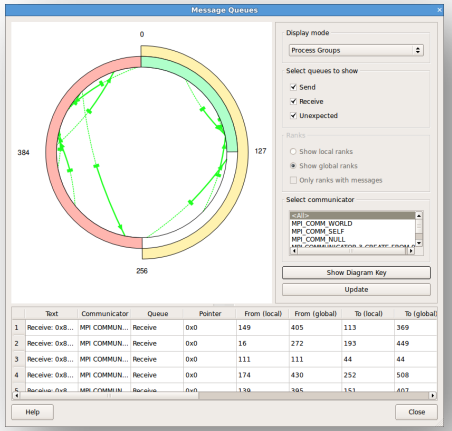
- Integrated source code editor
- Dynamic data structure visualization

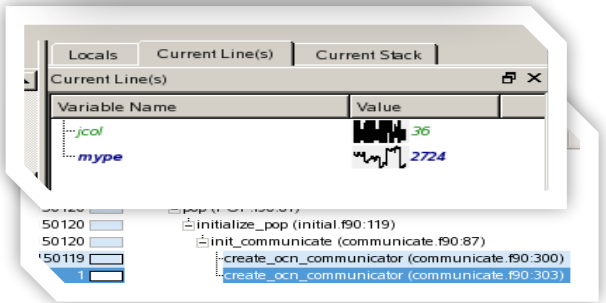
How did it happen?

- Parse diagnostic messages
- Trace variables through execution

Why did it happen?

- Unique “Smart Highlighting”
- Experiment with variable values

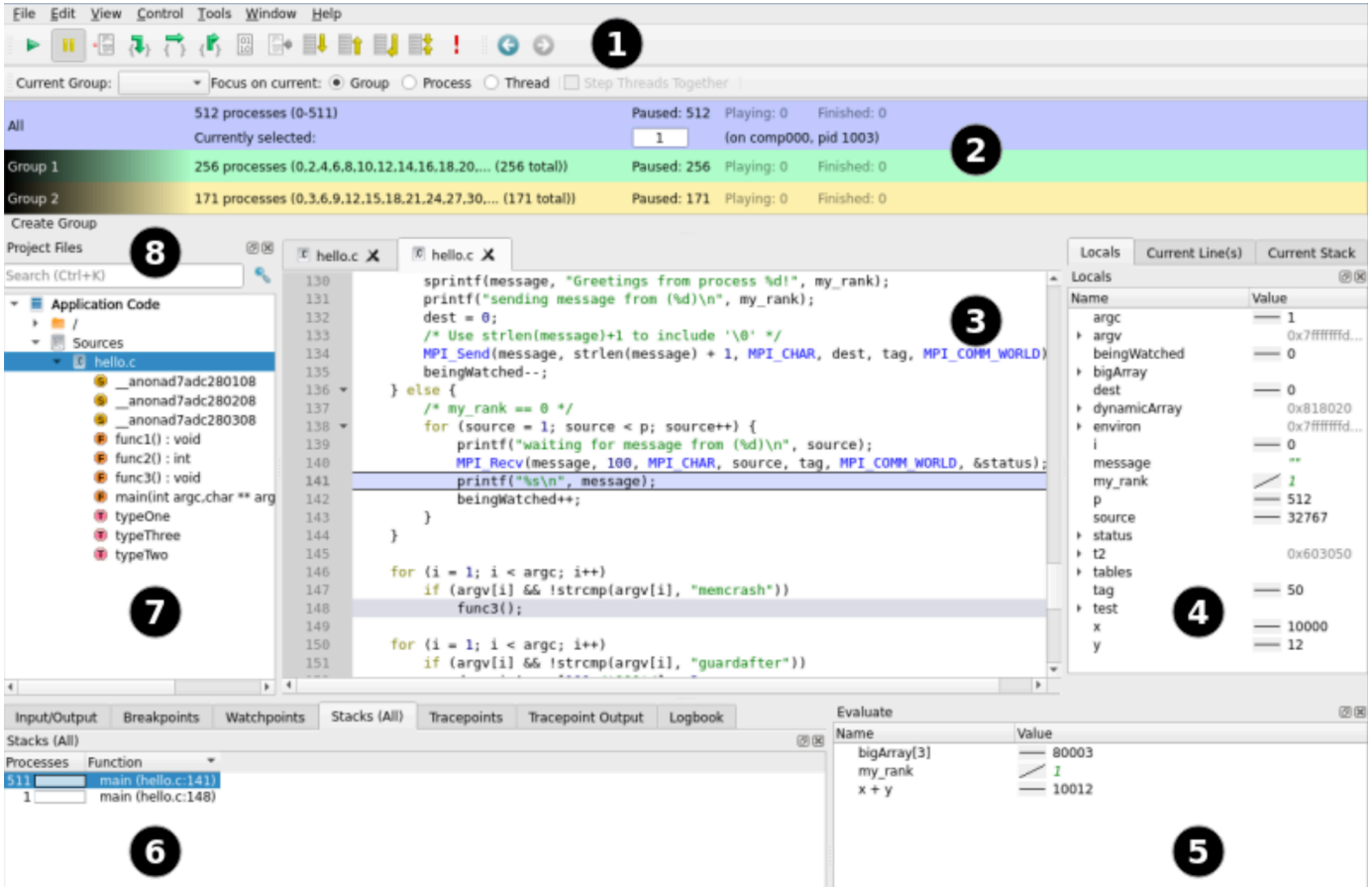




linaroforge

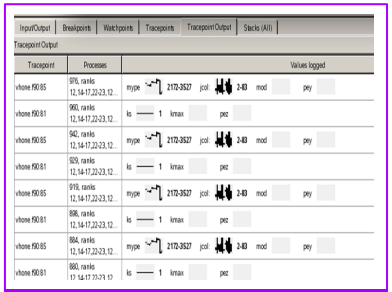
DDT UI

- 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

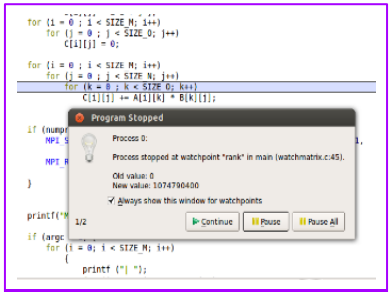


linaroforge

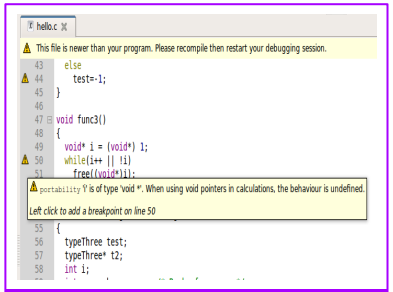
Linaro DDT Debugger Highlights



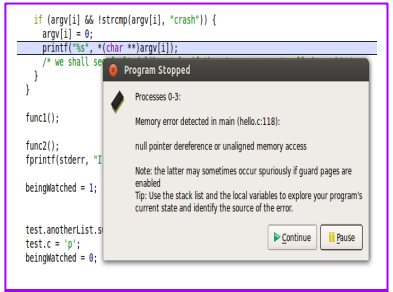
The scalable print alternative



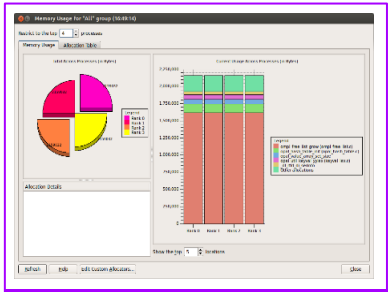
Stop on variable change



Static analysis warnings on code errors



Detect read/write beyond array bounds



Detect stale memory allocations

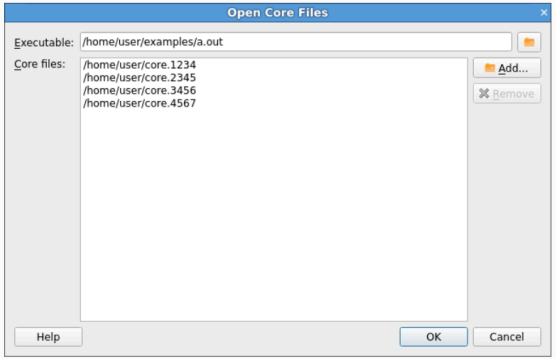
linaroforge

Core files

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

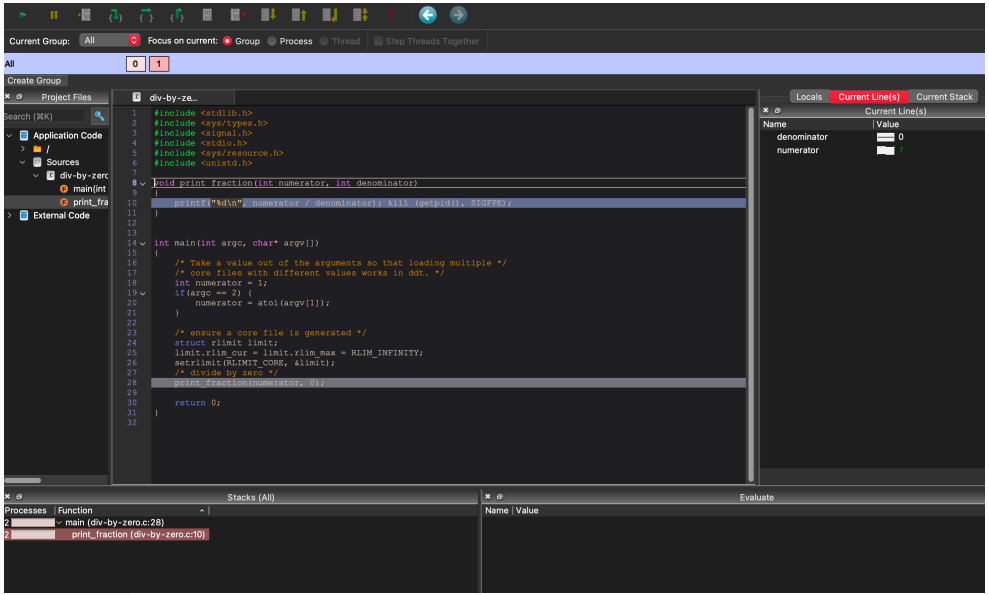
Procedure

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.

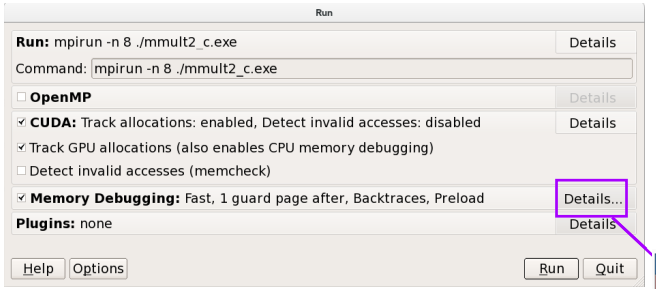
> **Note**
>
> While Linaro DDT is in this mode, you cannot play, pause, or step, because there is no process active. You are, however, able to evaluate expressions and browse the variables and stack frames saved in the core files.



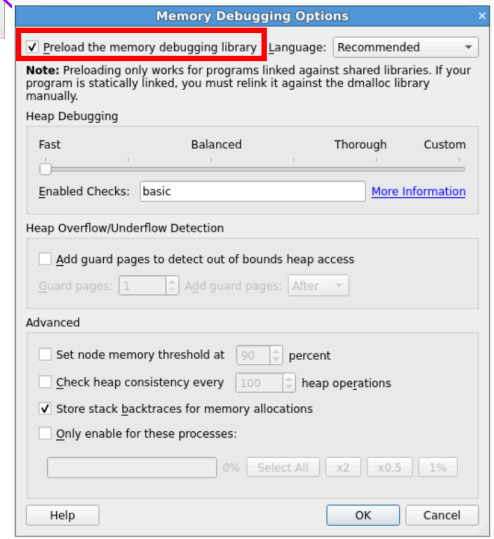
- View core files for CPU's
- View core files for GPU's

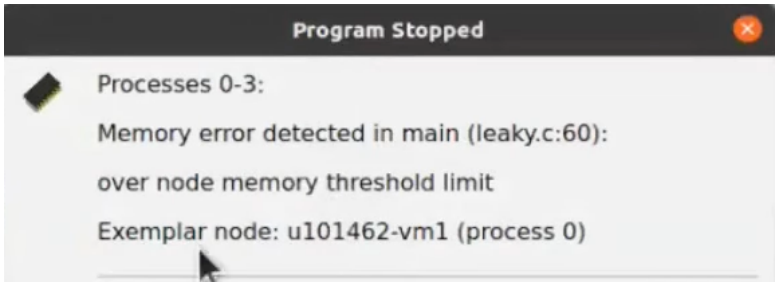
linaroforge

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

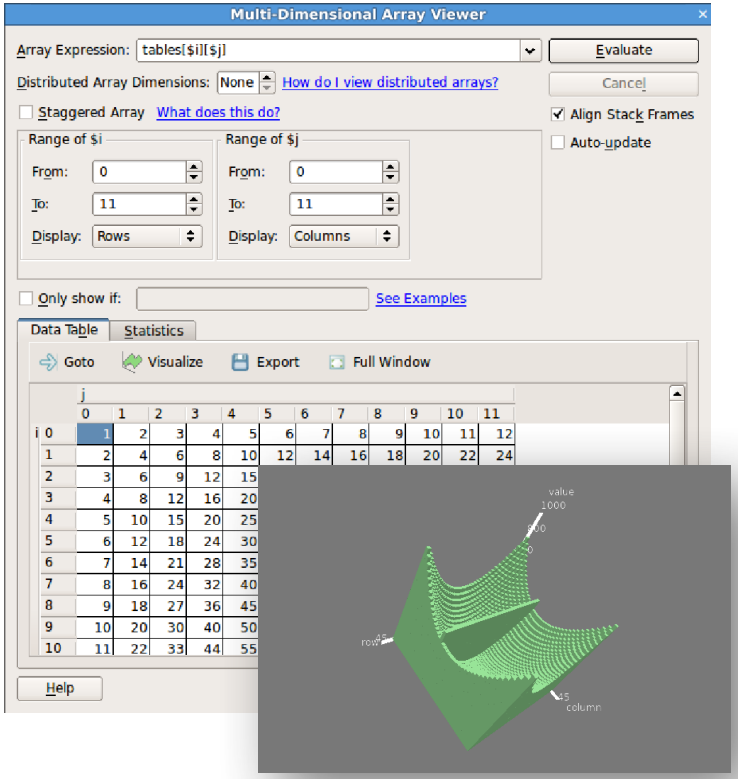
- On a single process
- Or distributed on many ranks

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

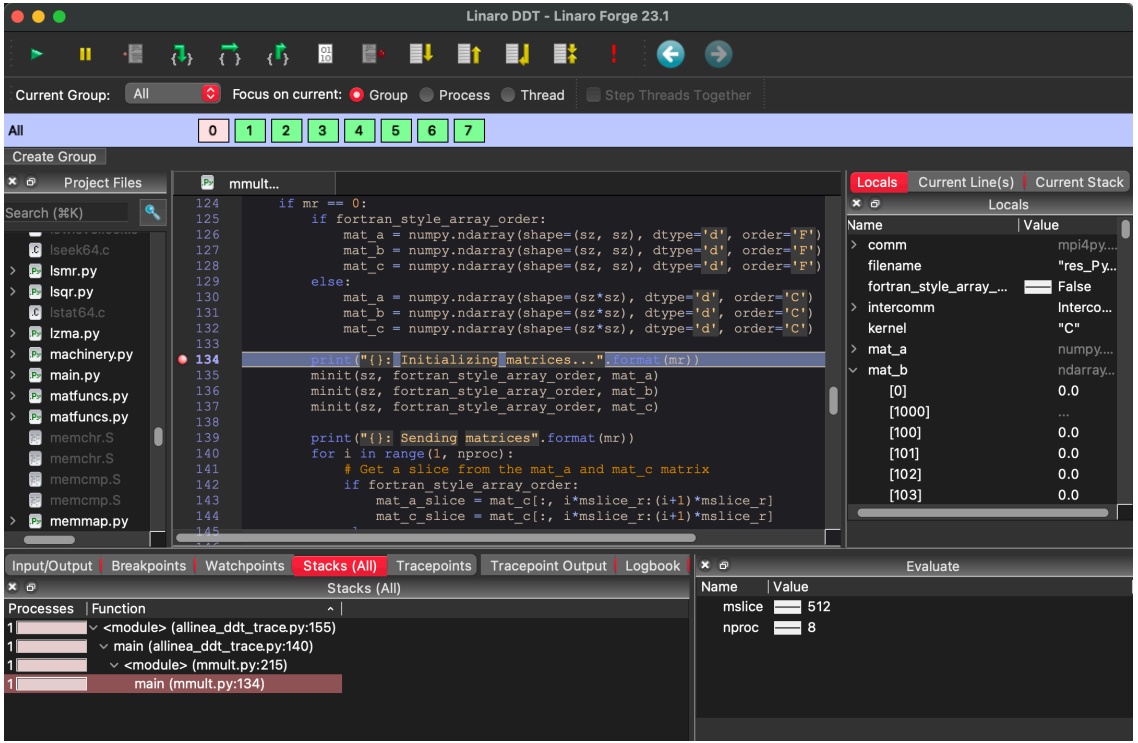
- “Only show if”: $value>0 for example $value being a specific element of the array



linaroforge

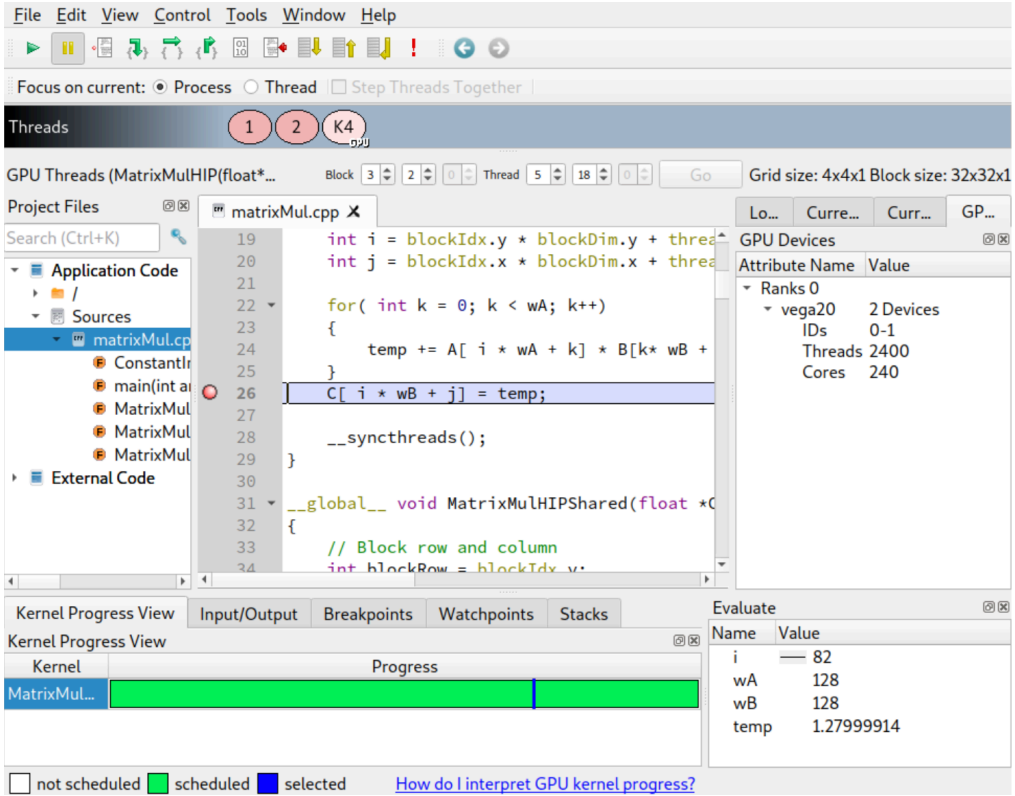
Python Debugging

- 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 mpiexec -n 8 python3
%allinea_python_debug% ./mmult.py

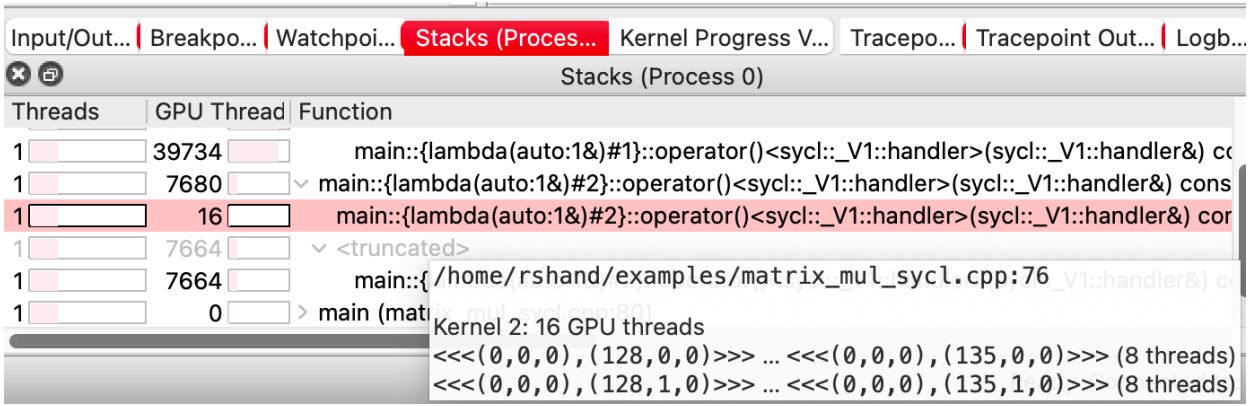
Debugging AMD GPUs



- 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
- Compile with -g (on Hunter: --offload-arch=gfx942)
- rocgdb must be in your environment (ROCm)
- DDT steps a whole wavefront (64 threads) at a time

linaroforge

Parallel Stack View



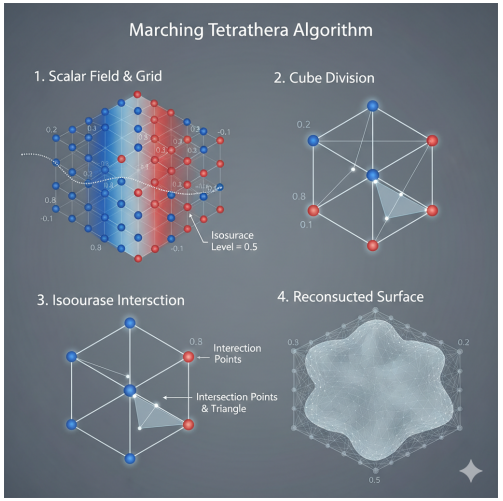
- Display location and number of threads
- Click item:
  - Select GPU Thread.
  - Update variable display.
  - Move Source Code Viewer.
- Tooltip displays:
  - GPU Thread Ranges.
  - Size of each range.

linaroforge

Case Study: Austrian Scientific Computing (ASC)

Developing a GPU version of an already existing serial algorithm (Marching Tetrahedra). https://github.com/HliasGit/tetra

- The aim of the algorithm is to extract the approximation of an iso-surface from a scalar field.
- The scalar field (whose dimensions are DX,DY,DZ) is represented as a 3D tensor (of floats)
- Linearise (flatten) the scalar field to use it in a simpler way on GPU



Used three kernels to develop the GPU version:

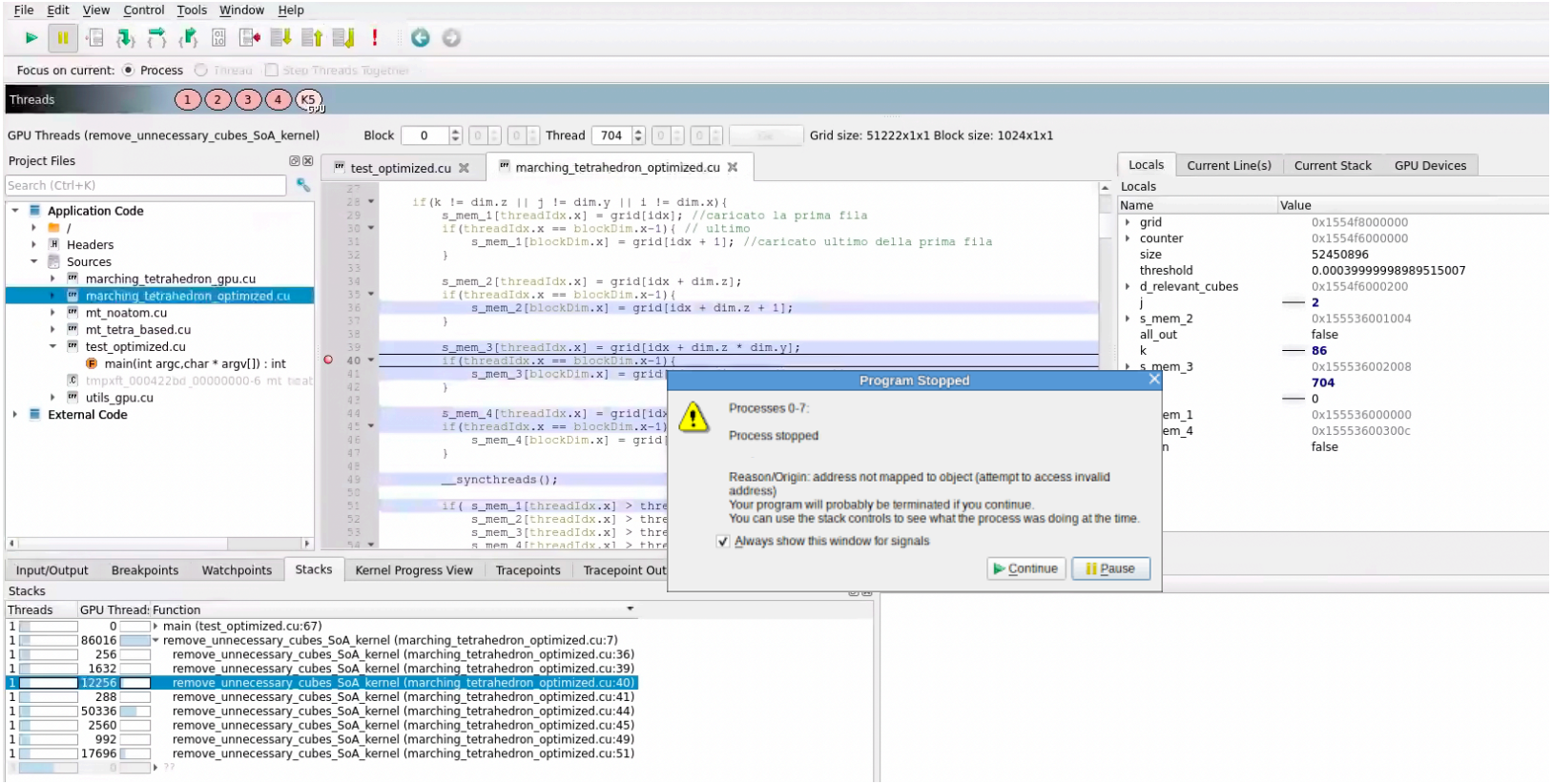
- The first one is a preprocessing step to reduce the computational complexity.
- Second and the third ones do the needed computation for the algorithm to work.

For each point in the scalar field, it needs to form a "cube" by accessing its 8 neighbours. Due to the Z-major linearization, only the neighbor in the Z-direction (index + 1) is adjacent in memory. Accessing neighbors in the X-direction (e.g., index + DZ*DY) or Y-direction (e.g., index + DZ) requires calculated memory offsets.

This approach led to "CUDA illegal memory access" errors. The problem was a failure to correctly handle boundary conditions. When a thread processed a point on the "edge" of the 3D domain (like at x = DX-1 or y = DY-1), these offset calculations were trying to access memory outside the allocated array. This boundary-checking logic was not intuitive, used Linaro DDT debugger to pinpoint these out-of-bounds access.

linaroforge

Spotting illegal memory access with Linaro DDT



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

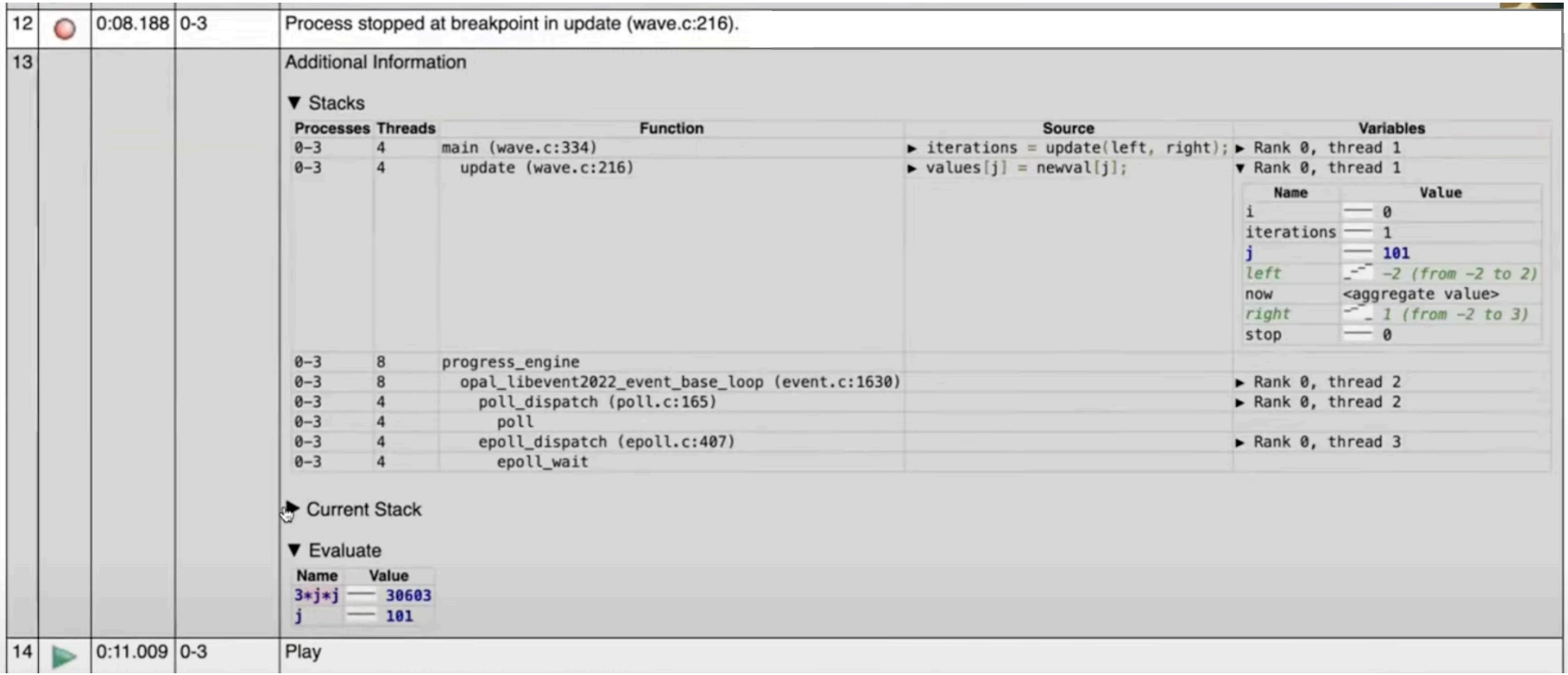
- For long-running jobs / debugging at very high scale
- For automated testing, continuous integration…

To do so, use following arguments:

- $ ddt **--offline --output=report.html** mpiexec -n 4 ./simple
  - **--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 offline-debugging.html \
    --break-at=simple.c:43 --evaluate="recvbuf; sendbuf" \
    mpiexec -n 4 ./simple
```

Report output



12		0:08.188	0-3	Process stopped at breakpoint in update (wave.c:216).
13				Additional Information
14		0:11.009	0-3	Play

▼ 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

linaroforge

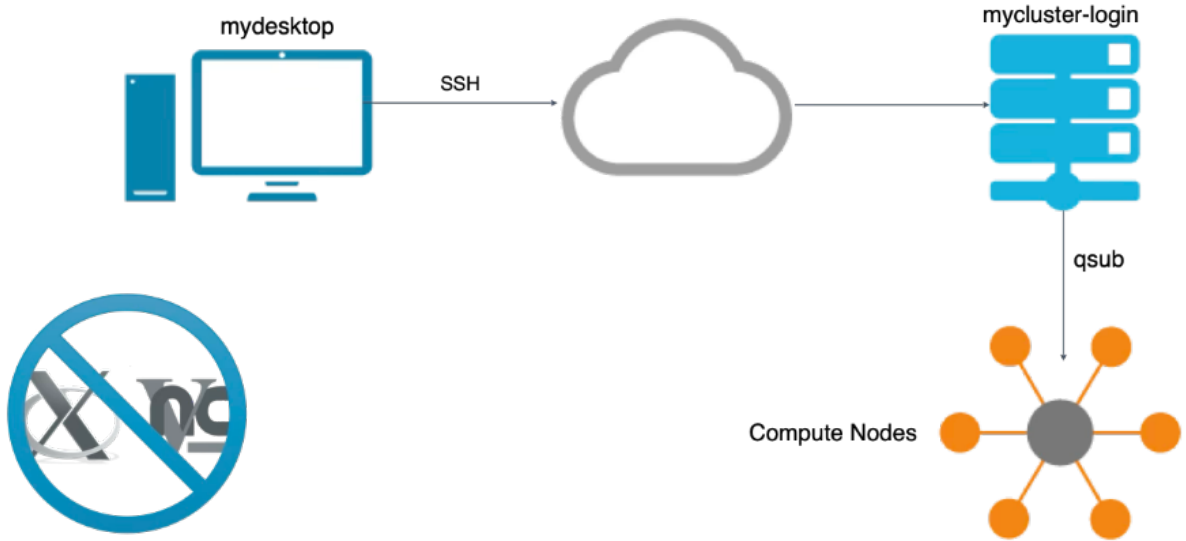


DDT Demo

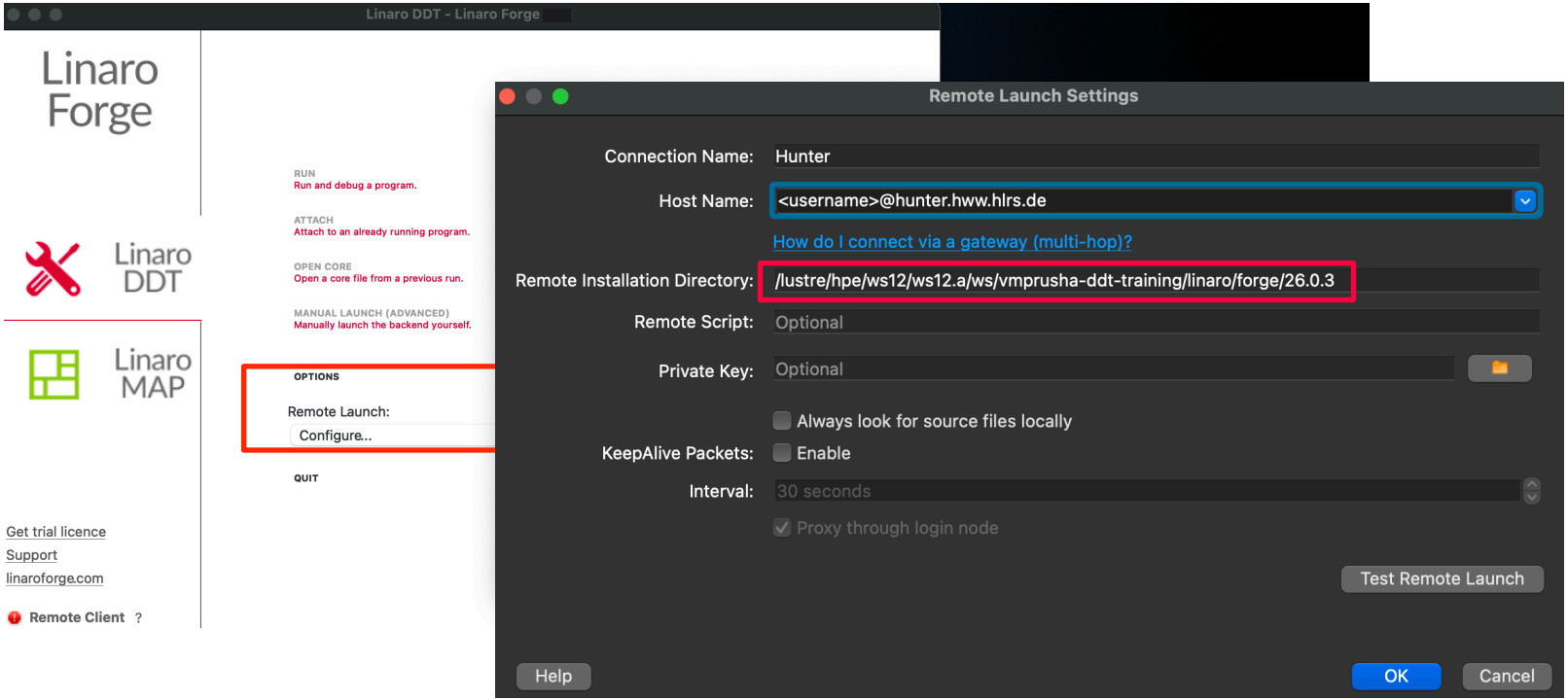
Running on a local machine, not Hunter: Hunter is a GPU system and these examples are CPU-only

The Forge GUI and where to run it

Forge provides a powerful GUI that can be run in a variety of configurations

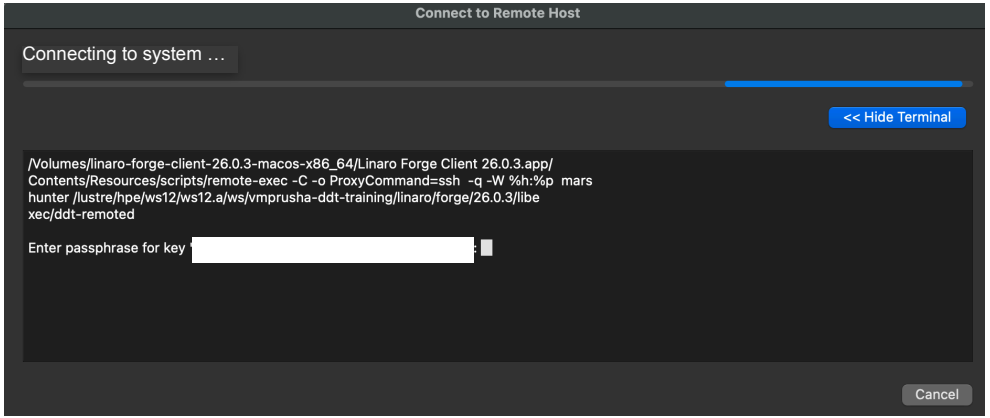


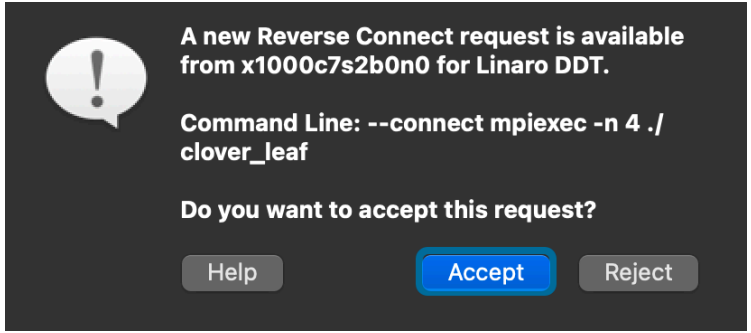
Remote connection to Hunter



Remote client and Reverse Connect

- Install the Linaro Forge 26.0.3 remote client on your laptop (macOS, Windows, Linux)
- Remote Launch: Provide your Hunter login, and the Forge install directory on Hunter
- You'll be asked for your password and your one-time code (two-factor)
- Your job starts `ddt --connect ...` and waits for the client to connect
- Accept the Reverse Connect request in the client





Running on Hunter

- The `ddt-training` path is `/lustre/hpe/ws12/ws12.a/ws/vmprusha-ddt-training`
- Copy the hands-on: `cp -r <ddt-training path>/hunter-handson <your workspace>`
- Build: `cd <your workspace>/hunter-handson; make gpu COMPILER=cray`
- Run: `qsub -v EXAMPLE=saxpy job.sh`
- Follow the README.txt guide

The hands-on

- saxpy: one wrong value on the GPU. Find it
- CloverLeaf: a GPU gas-flow simulation with 4 bugs. Fix them one after the other
- Follow the README.txt in hunter-handson
- Stuck for 10 minutes? Read the hint
- Is my answer right? Volume 20.0 and Mass 12.0 at every step and 535 steps

linaroforge



Thank you

rudy.shand@linaro.org
support@forge.linaro.com

Go to www.linaroforge.com