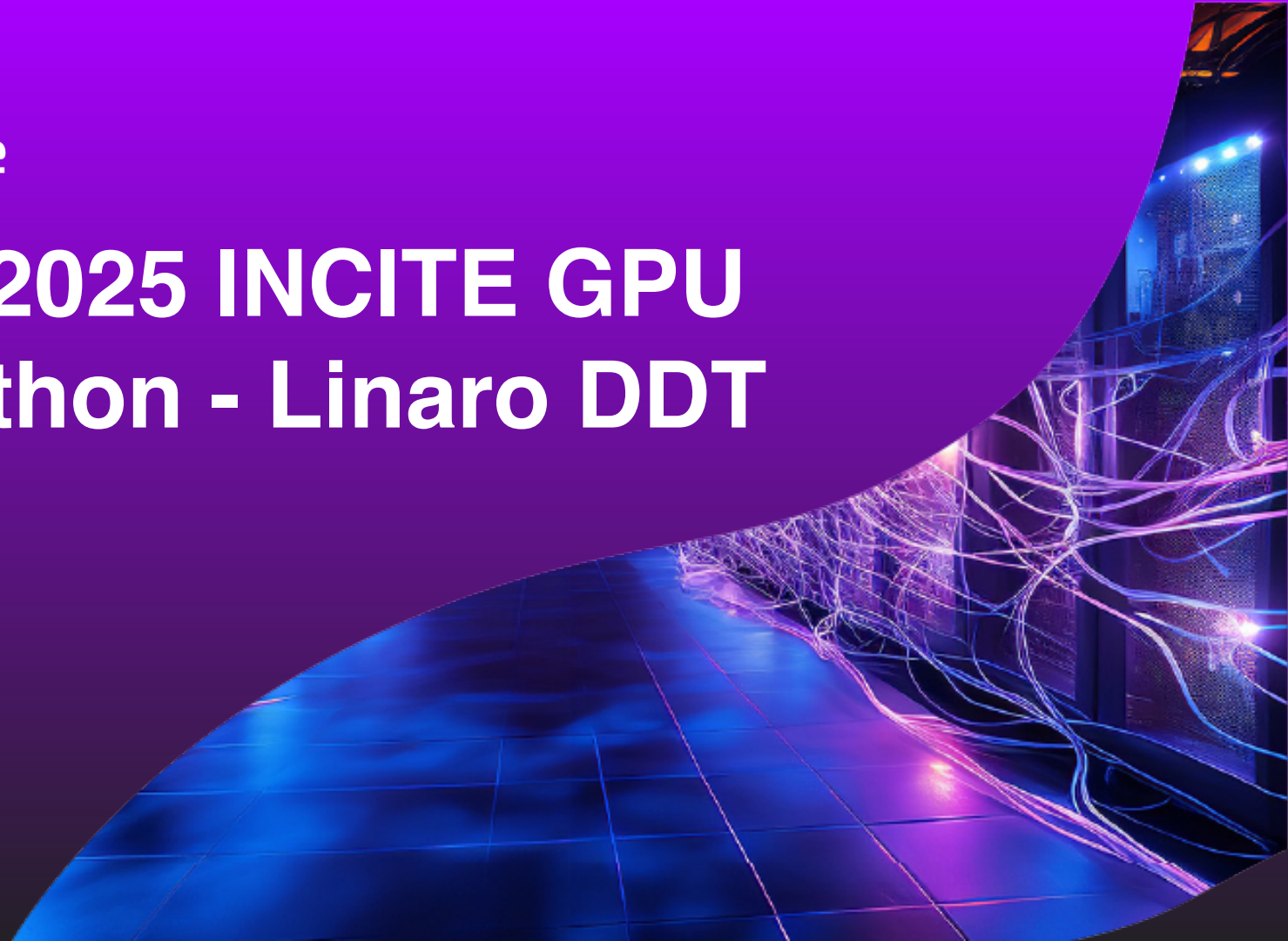


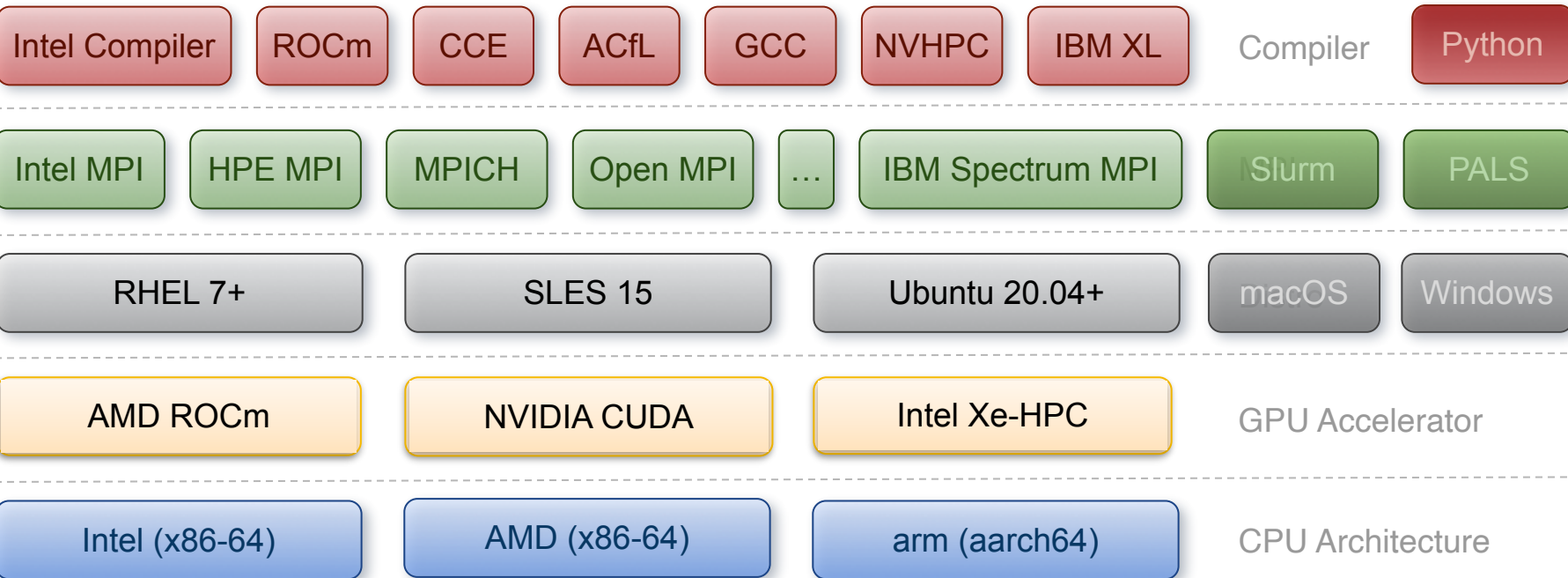
linaroforge

ALCF 2025 INCITE GPU Hackathon - Linaro DDT

Rudy Shand
Principal FAE



Supported Platforms



Linaro DDT Debugger Highlights

InputOutput	Breakpoint	Watchpoint	Tracepoint	TracepointOutput	Stacks (All)
Tracepoint	Process				Values logged
vhone RDE05	976, vars	12, 14, 17, 20, 23, 12	myse ~ 210-3527 jst	340 mod	py
vhone RDE01	960, vars	12, 14, 17, 20, 23, 12	ls — 1 lmax	per	
vhone RDE05	960, vars	12, 14, 17, 20, 23, 12	myse ~ 210-3527 jst	340 mod	py
vhone RDE01	960, vars	12, 14, 17, 20, 23, 12	ls — 1 lmax	per	
vhone RDE05	976, vars	12, 14, 17, 20, 23, 12	myse ~ 210-3527 jst	340 mod	py
vhone RDE01	960, vars	12, 14, 17, 20, 23, 12	ls — 1 lmax	per	
vhone RDE05	960, vars	12, 14, 17, 20, 23, 12	myse ~ 210-3527 jst	340 mod	py
vhone RDE01	960, vars	12, 14, 17, 20, 23, 12	ls — 1 lmax	per	

The scalable print alternative

```

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

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

if (strcmp("NPI",
NPI) != 0)
    continue;
}

printf("1/2\n");

if (argc > 1)
    if (arg == 1)
        print ("1");

```

Stop on variable change

```

hello.c:13
This file is newer than your program. Please recompile then restart your debugging session.
13 else
14 test=1;
15 }
16
17 void func3()
18 {
19     void* i = (void*) 1;
20     while(i != 1)
21         free(i);
22 }
23
24 portability Y is of type 'void *'. When using void pointers in calculations, the behaviour is undefined.
25 Left click to add a breakpoint on line 50
26
27 {
28     typeThree test;
29     typeThree* t2;
30     int i;
31 }

```

Static analysis warnings on code errors

Memory Debugging

Enable reading of debug environment variables:

host:~/demo> export NEORReadDebugKeys=1

Disable RTLD_DEEPBIND, so that malloc/free calls bind to

our memory debug library, instead of glibc:

host:~/demo> export NEODisableDeepBind=1

```

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

func1();

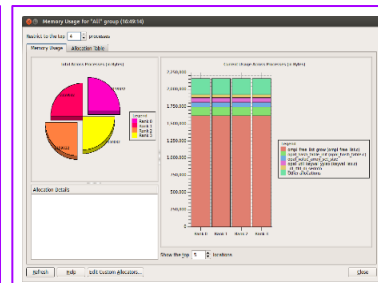
func2();
fprintf(stderr, "1\n");

beingMatched = 1;

test_anotherList()
test.c = 'p';
beingMatched = 0;

```

Detect read/write beyond array bounds



Detect stale memory allocations

Terminology

	NVIDIA GPU	AMD GPU	Intel GPU
Name	CUDA	ROCm	Xe
Language	CUDA C/C++ and Fortran	HIP	SYCL
Execution Unit	Warp	Wavefront	Sub-group
EU Size	32 Threads	64 Threads	8, 16 or 32 Threads
EU GDB	...	GDB Thread	GDB Thread
EU Thread	...	Lane	Lane (work-item)
Forge GPU Thread	Lane	Lane	Lane

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

The screenshot displays the DDT UI with the following components and annotations:

- 1**: Process controls toolbar (run, pause, etc.)
- 2**: Process groups summary (All, Group 1, Group 2)
- 3**: Source code view (hello.c)
- 4**: Variables panel (Locals, Current Line(s), Current Stack)
- 5**: Evaluate window (Name, Value)
- 6**: Parallel Stack (Processes, Function)
- 7**: Project files (Application Code, Sources)
- 8**: Find a file or function (Search bar)

Process Groups Summary:

Group	Processes	Paused	Playing	Finished
All	512 processes (0-511)	512	0	0
Group 1	256 processes (0.2.4.6.8.10.12.14.16.18.20.... (256 total))	256	0	0
Group 2	171 processes (0.3.6.9.12.15.18.21.24.27.30.... (171 total))	171	0	0

Stacks (All):

Processes	Function
511	main (hello.c:141)
1	main (hello.c:148)

Evaluate:

Name	Value
bigArray[3]	80003
my_rank	1
x + y	10012

Debugging Intel Xe GPUs

Using Linaro DDT

Debug code simultaneously on the GPU and the CPU

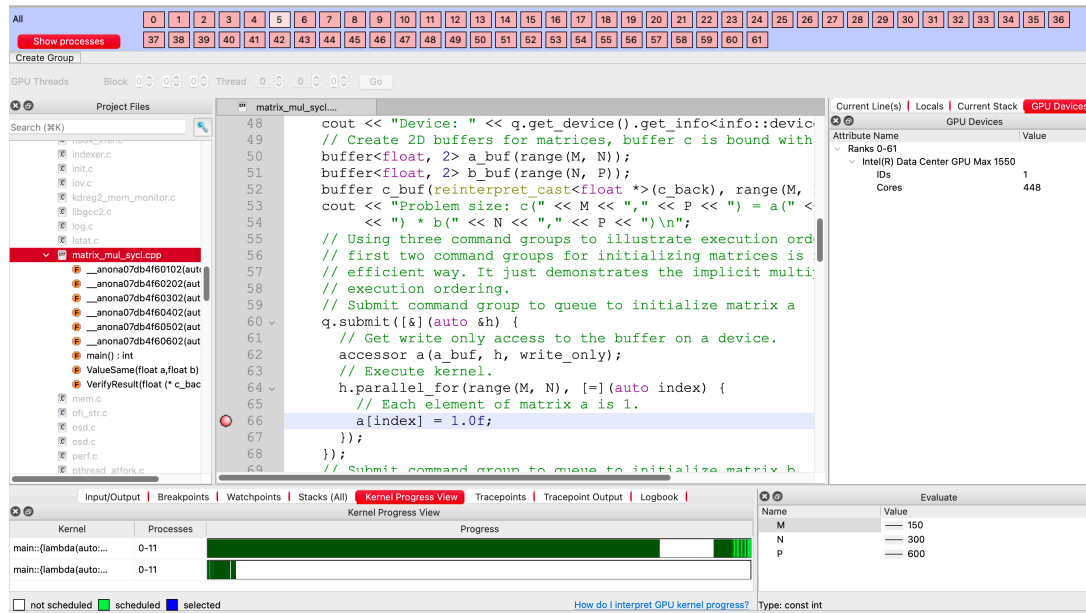
Controlling the GPU execution:

- All active threads in a Sub-group will execute in lockstep. Therefore, DDT will step 16 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

Kernels must be compiled with the -g and -O0 flags



Parallel Stack View

Threads	GPU Thread	Function
1	39734	main::{lambda(auto:1&)#1}::operator()<sycl::_V1::handler>(sycl::_V1::handler&)
1	7680	main::{lambda(auto:1&)#2}::operator()<sycl::_V1::handler>(sycl::_V1::handler&) cons
1	16	main::{lambda(auto:1&)#2}::operator()<sycl::_V1::handler>(sycl::_V1::handler&) cor
1	7664	<truncated>
1	7664	main::{/home/rshand/examples/matrix_mul_sycl.cpp:76 _V1::handler&) c
1	0	> main (mat

Kernel 2: 16 GPU threads

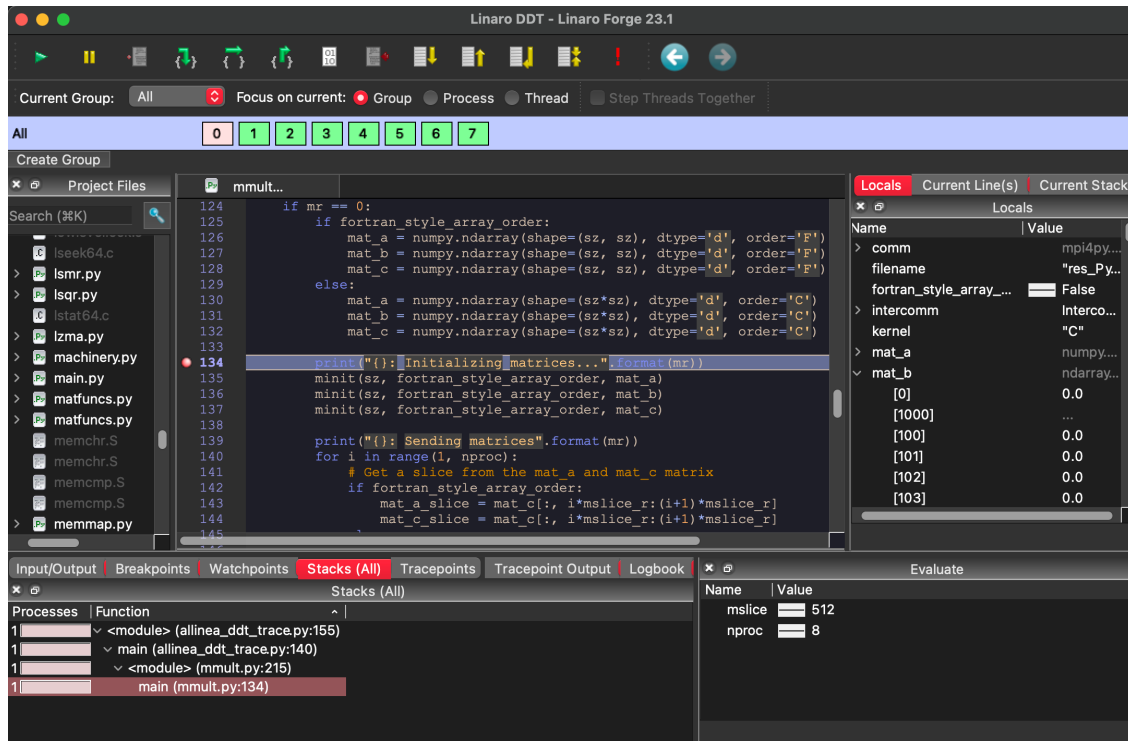
<<<(0,0,0), (128,0,0)>>> ... <<<(0,0,0), (135,0,0)>>> (8 threads)

<<<(0,0,0), (128,1,0)>>> ... <<<(0,0,0), (135,1,0)>>> (8 threads)

- 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.

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
```

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...

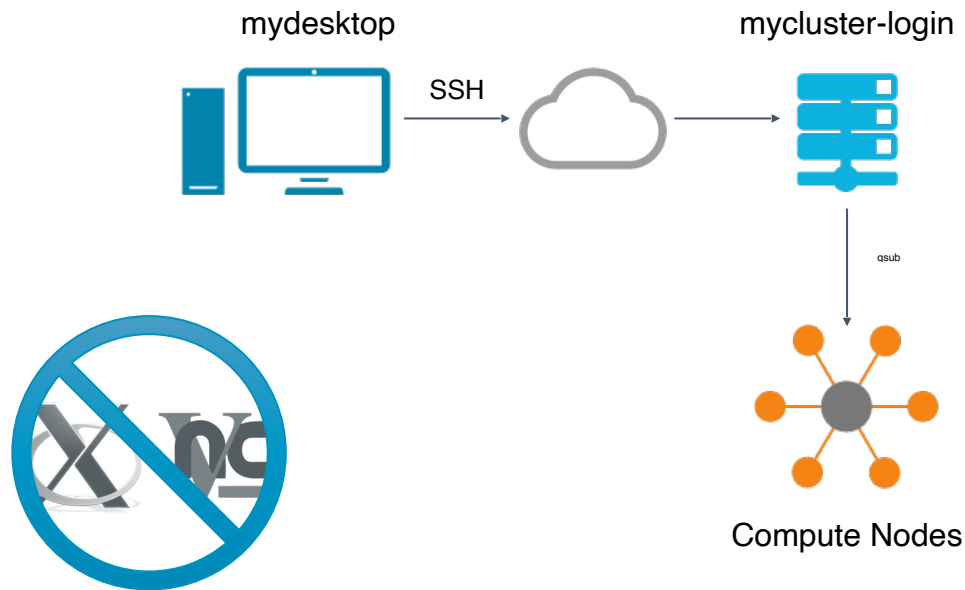
To do so, use following arguments:

- `$ 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**

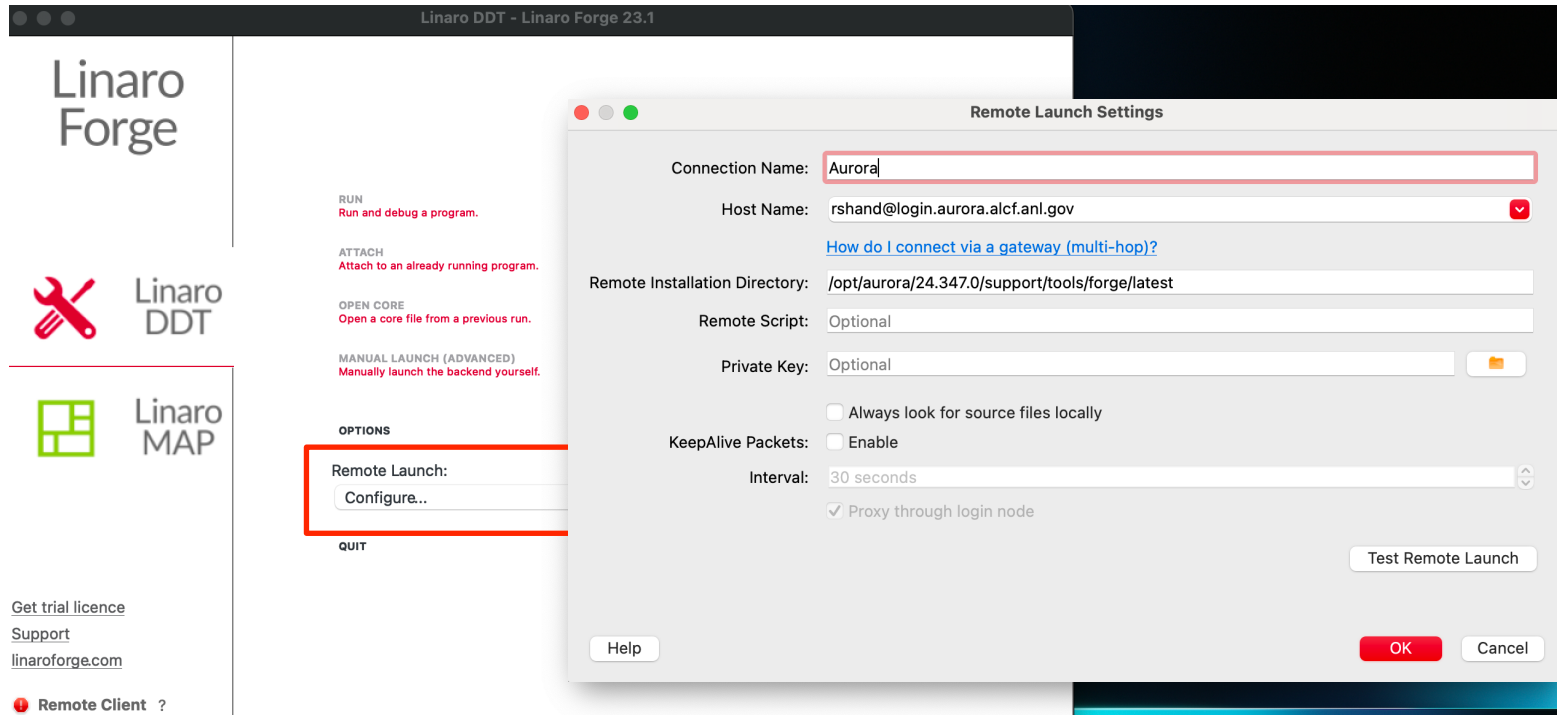
```
ddt --offline -o jacobi_omp_mpi_gnu_debug.txt \  
--trace-at _jacobi.F90:83,residual \  
mpiexec ./jacobi omp mpi gnu.exe
```

The Forge GUI and where to run it

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



Remote connection to AWS



Debug with DDT on Aurora

```
mpicxx -fsycl -g -O0 matrix_mul_sycl.cpp -o matrixmul
```

```
qsub -l select=2 -l walltime=30:00 -l filesystems=flare -A gpu_hack -q gpu_hack_prio -I
```

```
./soft/compilers/oneapi/2025.1.0/debugger/2025.1/env/vars.sh
```

<https://docs.alcf.anl.gov/aurora/debugging/ddt-aurora/#invoking-the-ddt-server-from-aurora>

```
ddt --np=24 --connect --mpi=generic --mpiargs="--ppn 12 --envall" ./matrixmul
```

Thank you

rudy.shand@linaro.org

support@forge.linaro.com

Go to www.linaroforge.com