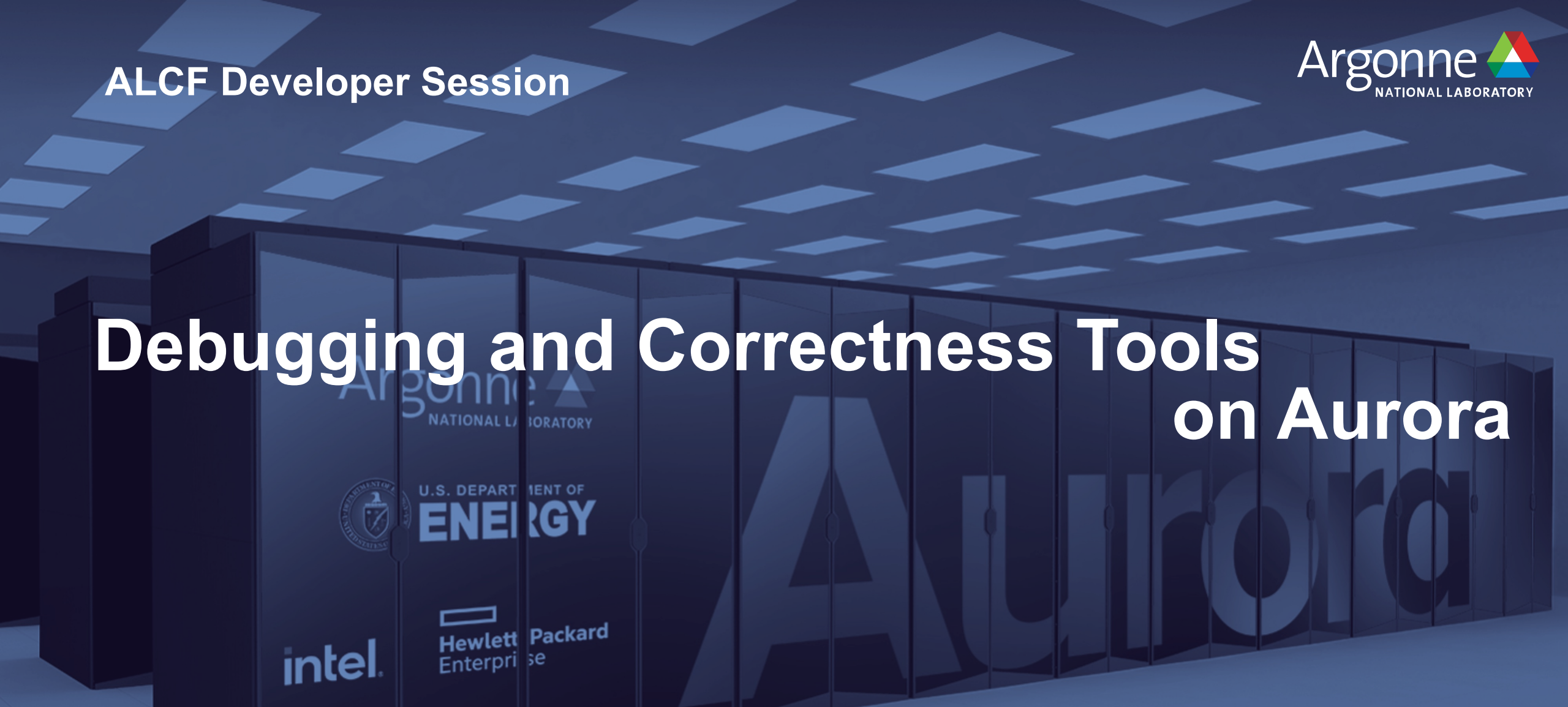
ALCF Developer Session

Argonne NATIONAL LABORATORY



Debugging and Correctness Tools on Aurora

JaeHyuk Kwack (jkwack@anl.gov)
ALCF Performance Engineering Group

10/1/2025

Agenda

- Debugging tools on Aurora
  - —Preliminaries for debugging on Aurora
  - —Intel gdb-oneapi
  - —Linaro DDT
- Correctness tool on Aurora
  - —Intel Sanitizer

Preliminaries for debugging on Aurora

- To use debuggers (e.g., `gdb-oneapi`, `ddt`), you need to compile and link your applications with `-g`. To get anywhere with GPU debugging, the current best practice is to compile and link with `-g -O0`.
- Before debugging your applications with either `gdb-oneapi` or `ddt` on Aurora, you must explicitly enable GPU debugging configurations on all the GPUs you are using. One way to do this is to create a script and execute it across all your compute nodes using mpiexec.
- On the next slide, there is an example script (`helper_toggle_eu_debug.sh`), which takes an argument 1 to enable debugging or 0 to disable it.

- On an interactive job mode, issue the following before starting debugging

```
export NNODES=`wc -l < $PBS_NODEFILE`
mpiexec -n $NNODES ./helper_toggle_eu_debug.sh 1
export ZET_ENABLE_PROGRAM_DEBUGGING=1
```

helper_toggle_eu_debug.sh

```
#!/usr/bin/env bash
export MY_RANK=${PMIX_RANK}
export MY_NODE=${PALS_NODEID}
export MY_LOCAL_RANK=${PALS_LOCAL_RANKID}

eu_debug_toggle() {
  for f in /sys/class/drm/card*/prelim_enable_eu_debug
  do
    echo $1 > $f
  done
  echo "INFO: EU debug state on rank-${MY_RANK}: $(cat /sys/class/drm/card*/prelim_enable_eu_debug | tr '\n' ' ')"
}

# One rank per node toggles eu debug:
if [ ${MY_LOCAL_RANK} -eq 0 ]; then
    eu_debug_toggle $1
fi
```

Intel `gdb-oneapi`

- The `gdb-oneapi` tool is a part of Intel's oneAPI software and is available via the default modules loaded on Aurora.
  - All standard GDB features
  - Support for C, C++, SYCL, Fortran, OpenMP for both C/C++ and Fortran
  - Multi-target/GPU: debug “host” and ”kernel” in the same session
    - Auto-attach: automatically create inferior to debug GPU
    - SIMD lanes: display lane information and switch among lanes

- Fundamental GDB commands

`help [cmd|cmd-class]`	Print help for the given command or command class
`run [arg1, … argN]`	Start the program, optionally with arguments
`break <file>:<line>`	Define a breakpoint at a specific line
`info break`	Show defined breakpoints
`delete <N>`	Remove Nth breakpoint class
`step / next`	Single-step a source line, stepping into/over function calls
`info args/locals`	Show the arguments / local variables of the current function
`print <exp>`	Print value of expression
`x/<format> <addr>`	Examine the memory at <addr>
`up, down`	Go one level up/down in the function call stack
`disassemble`	Disassemble the current function
`backtrace`	Show the function call stack

Inferiors

GDB uses *inferior* objects to represent states of program execution (usually a process)

- Debugger create inferior(s) that attaches to GPU(s) to receive events and control the GPU


```
$ icpx -fsycl -g -O0 array-transform.cpp -o array-transform
$ gdb-oneapi -q -args ./array-transform gpu
Reading symbols from ./array-transform...
(gdb) b 59
Breakpoint 1 at 0x406f34: file array-transform.cpp, line 59.
(gdb) r
…
intelgt: gdbserver-gt started for process 28986.
…
(gdb) info inferiors
 Num  Description   Connection                                        Executable
   1  process 9463  1 (native)                                        <path_to_program>
* 2  device 1       2 (extended-remote gdbserver-gt --multi --hostpid=9463 -)
```

GDB commands relevant for GPU debugging

- `info inferior`
  - Display information about the inferior. GPU debugging will display additional inferiors(s) (gdbserver-gt)
- `info threads <thread>`
  - Display information about threads, including their active SIMD lanes
- `thread <thread>:<lane>`
  - Switch context to the SIMD lane of the specific thread
- `thread apply <thread>:<lane> <cmd>`
  - Apply <cmd> to specified lane of the thread
- `set scheduler-locking on/step/off`
  - Lock the thread scheduler. Keep other threads stopped while current thread is stepping (step) or resume (on) to avoid interference. Default (off)
- `set nonstop on/off`
  - Enable/disable nonstop mode. Set before program starts.
  - (off): When a thread stops, all other threads stop. Default
  - (on): When a thread stops, other threads keep running.
- `print/t $emask`
  - Inspect the execution mask to show active lanes

Running applications with `gdb-oneapi` on Aurora

Debugging on a single GPU

```
$ gdb-oneapi -q ./{your_application}
```

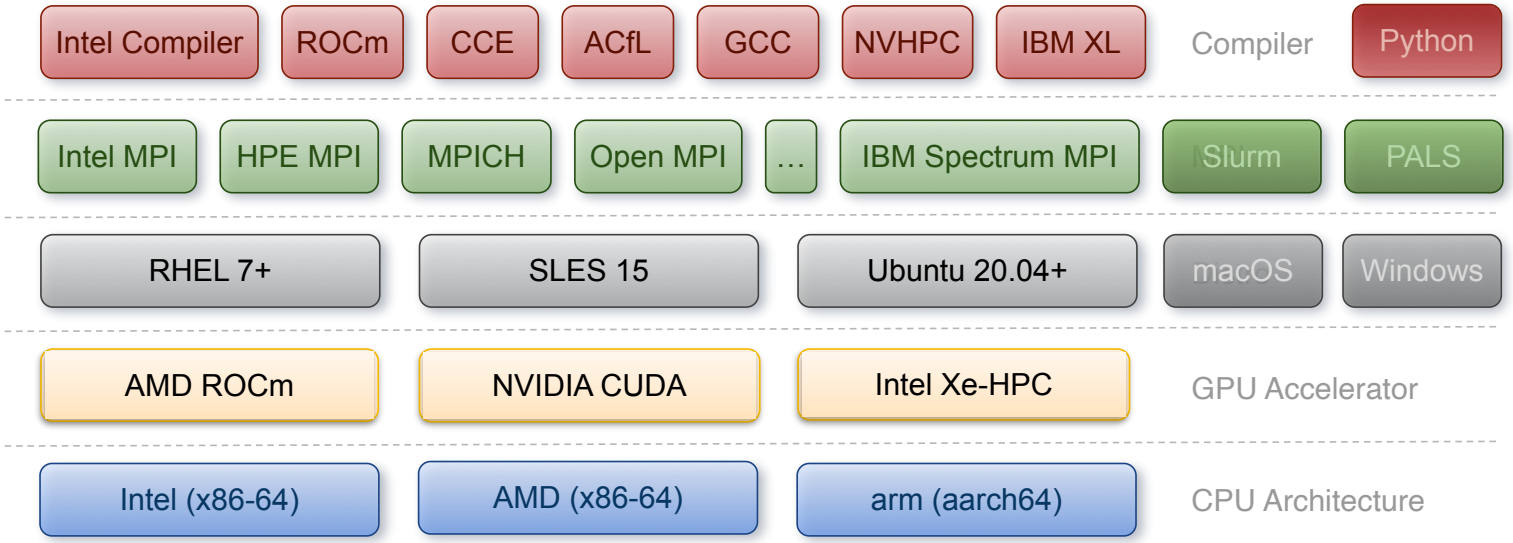
Debugging an MPI rank out of multiple MPI ranks

To debug an MPI application with `gdb-oneapi`, xterm can be used to display the output from multiple processes launched in separate xterm windows. For that, X11 forwarding should be established from user's local system to Aurora login node, and then to the compute node.

```
$ mpirun -n 1 ./{your_application} : -n 1 xterm -e
gdb-oneapi -q ./{your_application} : -n 10
./{your_application}
```

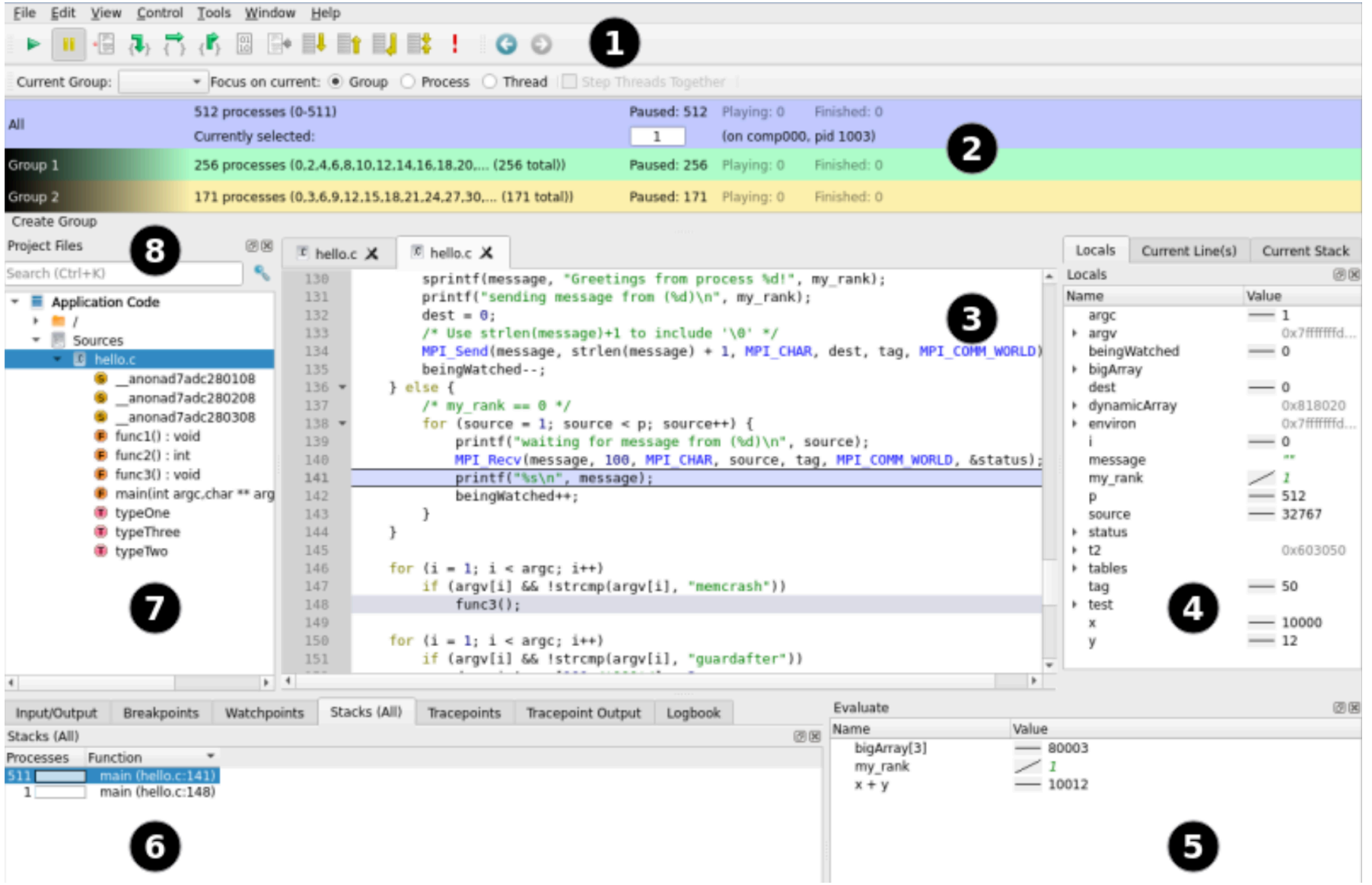
Linaro DDT

- Linara DDT is widely used for debugging parallel Message Passing Interface (MPI) and threaded applications on HPC systems.
- Its graphical user interface (GUI) provides a simplifying, easy-to-use method of debugging applications at scale on Aurora.
- For the best experience, download and install the Linaro Forge client 24.1. This is available for Linux, macOS, and Windows systems. You can configure the remote client following instruction here.
- It supports various platforms



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



Running applications with `ddt` on Aurora

Loading a module for `ddt` on Aurora

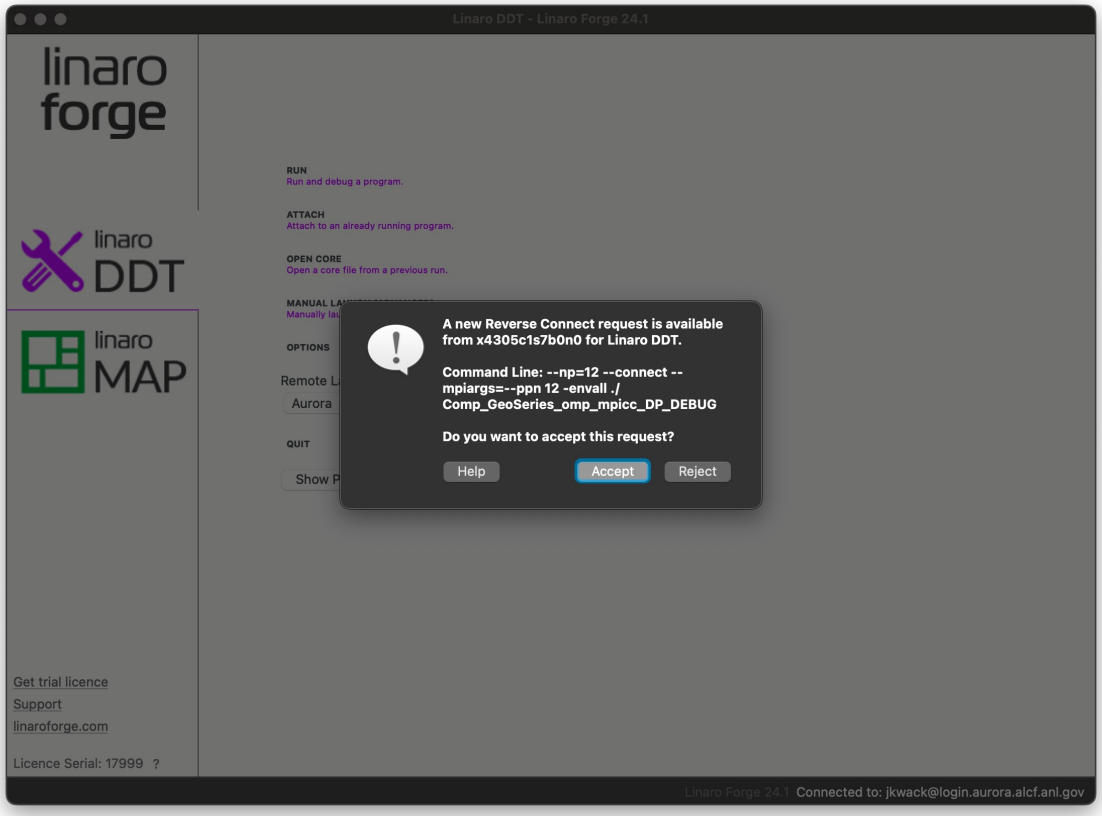
```
$ module load forge
```

Running applications with `ddt` and connecting it to your client

- Connect the Linaro client on your local system to Aurora from the Remote Launch pull-down:
- On an interactive job mode or a batch job, run `ddt --connect` as follows (e.g., 12 MPI ranks):

```
$ ddt --np=12 --connect --mpi=generic --
mpiargs="--ppn 12 -envall" ./{your_application}
```

- On your client, a window with your `ddt` command pops up for a reverse connection, and click Accept to connect your client to the `ddt` run on the compute node:



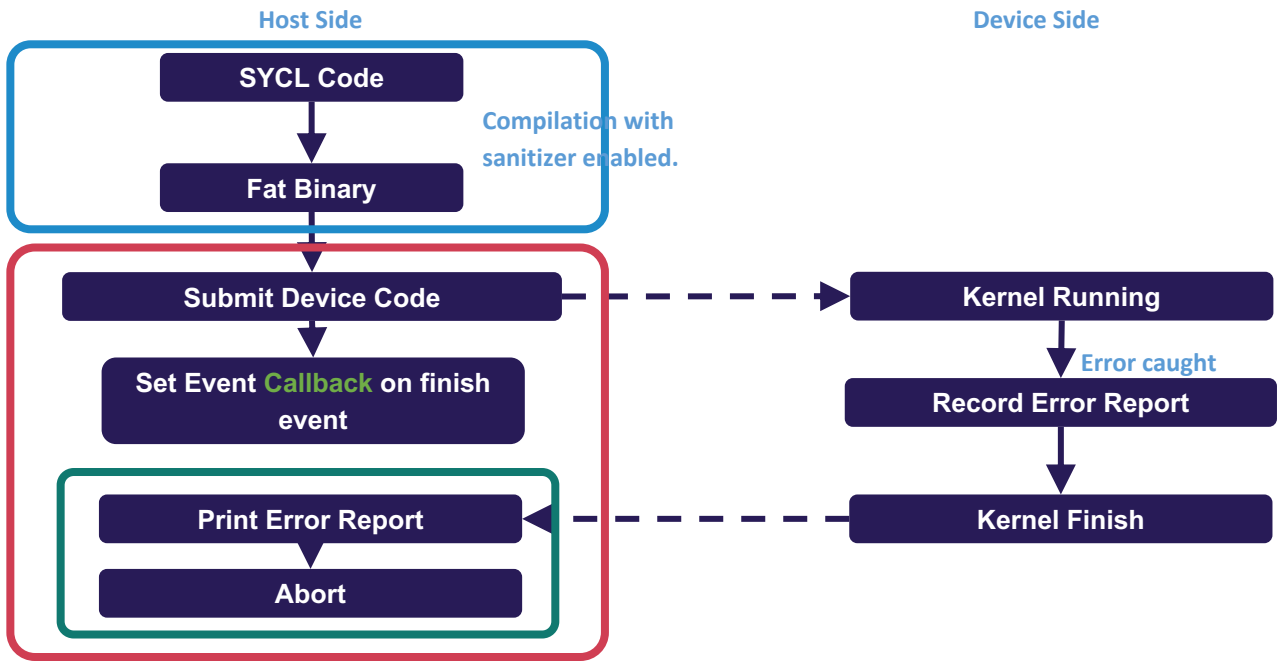
Demo: `gdb-oneapi` & `ddt` on Aurora

- https://github.com/argonne-lcf/ALCFBeginnersGuide/blob/master/aurora/02_a_debugger.md#a-quick-example

Intel Sanitizer

Intel Sanitizer is a correctness tool to detect addressability issues, memory leaks, data races and deadlocks, and use of uninitialized memory.

- The **compilation** builds sanitized device images based on compiler driver flags.
- The **runtime** checks for the existence of sanitized images. If they exist, the runtime enables a specific sanitizer layer in the unified runtime module to support sanitizer functionality.



Address/Memory/Thread Sanitizer

	Address Sanitizer (ASan)	Memory Sanitizer (MSan)	Thread Sanitizer (TSan)
Features	• Out-of-bound check • Use-after-free • Misaligned access • Mismatched deallocation • Misusing host & device pointers • Invalid kernel argument • Mismatched queue	• Use of uninitialized memory	• Deadlock • Data race
oneAPI version	2024.2 or newer	2025.0 or newer	2025.3

Address Sanitizer (ASan) Instruction

- Enable Device-side ASan on SYCL kernel:

```
$ icpx -fsycl -Xarch_device -fsanitize=address -g demo.cpp
$ ONEAPI_DEVICE_SELECTOR=level_zero:gpu ./demo
```

- Enable Device-side ASan on OpenMP Device code (w/ `icpx` or `ifx`):

```
$ icpx -fiopenmp -Xarch_device -fsanitize=address -fopenmp-targets=spir64 -g demo.cpp
$ LIBOMPTARGET_PLUGIN=unified_runtime UR_ENABLE_LAYERS=UR_LAYER_ASAN ./demo
```

- Error message example (e.g. OpenMP Fortran example)

```
==== DeviceSanitizer: ASAN
…
====ERROR: DeviceSanitizer: out-of-bounds-access on Device USM (0xff00fffff4a00280)
WRITE of size 8 at kernel <__omp_offloading_10301_140c146_gpu_rhf_j01_ssss__l85> LID(163, 0, 0)
GID(12707, 0, 0)
#0 OMP TARGET outline from gpu_rhf_j01_ssss_ line 85 /tmp/debug-
bench/applications/GAMESS/build/Release/object/tgpu_ompmod_sp.F90:106
```

Example: ASan for SYCL USM case

```
#include <sycl/sycl.hpp>

 int main() {
  sycl::queue Q;
  constexpr std::size_t N = 1024;
  auto *array = sycl::malloc_device<int>(N, Q);

   Q.submit([&](sycl::handler &h) {
    h.parallel_for<class MyKernel>(
        sycl::nd_range<1>(N, 1),
        [=](sycl::nd_item<1> item) { ++array[item.get_global_id(0)
+ 1]; });
  });

  Q.wait();

   return 0;
}
```

```
$ icpx –fsycl –Xarch_device –fsanitize=address  -g demo.cpp

$ ONEAPI_DEVICE_SELECTOR=level_zero:gpu ./demo

====ERROR: DeviceSanitizer: out-of-bounds-access on
Device USM
READ of size 4 at kernel <typeinfo name for
main::{lambda(sycl::_V1::handler&)#1}::operator()(sycl::_V1::ha
ndler&) const::MyKernel> LID(0, 0, 0) GID(1023, 0, 0)
  #0 typeinfo name for
main::'lambda'(sycl::_V1::handler&)::operator()(sycl::_V1::handl
er&) const::MyKernel
/root/sycl_workspace/examples/demo.cpp:11
```

Example: ASan for OpenMP Fortran

- Example code:

```
program map__out_of_bound
 use omp_lib
 implicit none

 integer, parameter :: LENGTH = 64
 integer, allocatable :: A(:)
 integer :: i

 allocate( A(1 : LENGTH) )

 !$omp target data map(tofrom: A)

 !$omp target teams distribute parallel do
 do i=1, LENGTH+1  ! <== OOB here
  A(i) = 42
 end do
 !$omp end target teams distribute parallel do

 !$omp end target data
 deallocate(A)

end program map__out_of_bound
```

Command line:

```
$ ifx -fpp -free -fiopenmp -fopenmp-targets=spir64 -fsanitize=address -
O0 -g map_out-of-bound.f90 -o map_out-of-bound.f90.exe
$ LIBOMPTARGET_PLUGIN=unified_runtime
UR_ENABLE_LAYERS="UR_LAYER_ASAN" ./map_out-of-bound.f90.exe
```

Error message:

```
====ERROR: DeviceSanitizer: out-of-bounds-access on
Device USM

WRITE of size 4 at kernel
<__omp_offloading_802_ea5ce_MAIN___l13> LID(0, 0, 0)
GID(64, 0, 0)

 #0 OMP TARGET outline from MAIN__ line 13
/localdisk2/user/ws/xm-devsan-fort/tc-0912/map_out-of-
bound.f90:15
```

Memory Sanitizer (MSan) Instruction

- Enable Device-side MSan on SYCL kernel:

```
$ icpx -fsycl -Xarch_device -fsanitize=memory -g demo.cpp
$ ONEAPI_DEVICE_SELECTOR=level_zero:gpu ./demo
```

- Enable Device-side MSan on OpenMP Device code (w/ `icpx` or `ifx`):

```
$ icpx -fiopenmp -Xarch_device -fsanitize=memory -fopenmp-targets=spir64 -g demo.cpp
$ LIBOMPTARGET_PLUGIN=unified_runtime UR_ENABLE_LAYERS=UR_LAYER_MSAN ./demo
```

- Error message example (e.g. OpenMP C++ example)

```
==== DeviceSanitizer: MSAN
…
====WARNING: DeviceSanitizer: use-of-uninitialized-value
use of size 1 at kernel <__omp_offloading_10301_12070fb__ZN6openmc19process_init_eventsEi_l178>
LID(0, 0, 0) GID(3648, 0, 0)
#0 OMP TARGET outline from openmc::process_init_events(int) line 178
/localdisk2/maosuzha/ics_workspace/debug-bench/applications/OpenMC/src/src/event.cpp:180
```

Example: MSan to detect local memory

```
1  #include <sycl/sycl.hpp>
2
3   constexpr std::size_t global_size = 4;
4   constexpr std::size_t local_size = 1;
5
6   __attribute__((noinline)) int check(int data) { return data; }
7
8  int main() {
9  sycl::queue Q;
10
11 Q.submit([&](sycl::handler &cgh) {
12    auto acc = sycl::local_accessor<int>(local_size, cgh);
13    cgh.parallel_for<class MyKernel>(
14       sycl::nd_range<1>(global_size, local_size),
15        [=](sycl::nd_item<1> item) { check(acc[item.get_local_id()]); });
16 });
17 Q.wait();
18
19 return 0;
20 }
```

```
icpx -fsycl -Xarch_device -fsanitize=memory -g -O2 local_accessor.cpp

==== DeviceSanitizer: MSAN
====WARNING: DeviceSanitizer: use-of-uninitialized-value)
use of size 4 at kernel <typeinfo name for main::{lambda(sycl::_V1::handler&)#1}::operator()(sycl::_V1::handler&)
const::MyKernel> LID(0, 0, 0) GID(3, 0, 0)
#0 typeinfo name for main::'lambda'(sycl::_V1::handler&)::operator()(sycl::_V1::handler&) const::MyKernel
/localdisk2/work/sycl_ws/llvm/sycl/test-e2e/MemorySanitizer/local/local_accessor.cpp:15
```

Example: MSan for Fortran offload

```
program msan_fortran
  use omp_lib
  implicit none
  integer, parameter :: LENGTH = 64
  integer, allocatable :: A(:)
  integer :: i


  !$omp allocators
allocate(allocator(omp_target_device_mem_alloc): A)
  allocate(A(1 : LENGTH))


  !$omp target teams distribute parallel do has_device_addr(A)
  do i = 1, 10
    ! Trying to use uninitialized device USM A(i) here, resulted
in MSAN error
    if (A(i) == i) then
      A(i) = 42
    end if
  end do
  !$omp end target teams distribute parallel do
  deallocate(A)
end program msan_fortran
```

```
$> ifx -fpp -free -fiopenmp -Xarch_device -
fsanitize=memory  -fopenmp-targets=spir64  -
O0  -g 1.f90 -o 1.exe



$> LIBOMPTARGET_PLUGIN=unified_runtime
UR_ENABLE_LAYERS=UR_LAYER_MSAN ./1.exe
==== DeviceSanitizer: MSAN
====WARNING: DeviceSanitizer: use-of-
uninitialized-value
use of size 1 at kernel
<__omp_offloading_802_2406b9_MAIN___l16>
LID(0, 0, 0) GID(3, 0, 0)
  #0 OMP TARGET outline from MAIN line 16
/tmp/test/1.f90:18
```

Thread Sanitizer (TSan) Instruction

- Enable Device-side TSan on SYCL kernel:

```
$ icpx -fsycl -Xarch_device -fsanitize=thread -g demo.cpp
$ ONEAPI_DEVICE_SELECTOR=level_zero:gpu ./demo
```

- Enable Device-side TSan on OpenMP Device code (w/ `icpx` or `ifx`):

```
$ icpx -fiopenmp -Xarch_device -fsanitize=thread -fopenmp-targets=spir64 -g demo.cpp
$ LIBOMPTARGET_PLUGIN=unified_runtime UR_ENABLE_LAYERS=UR_LAYER_TSAN ./demo
```

- Error message example (e.g. SYCL example)

```
==== DeviceSanitizer: TSAN
…
====WARNING: DeviceSanitizer: data race
When write of size 1 at 0xff00ffffffffe0000 in kernel <typeinfo name for
main::{lambda(sycl::_V1::handler&)#1}::operator()(sycl::_V1::handler&) const::Test> LID(7, 0, 0)
GID(23, 0, 0)
  #0 typeinfo name for main::'lambda'(sycl::_V1::handler&)::operator()(sycl::_V1::handler&)
const::Test /tmp/demo.cpp:10
```

Example: TSan to detect data race for USM

```
1  #include "sycl/sycl.hpp"
2
3  int main() {
4    sycl::queue Q;
5    auto *array = sycl::malloc_device<char>(1, Q);
6
7    Q.submit([&](sycl::handler &h) {
8       h.parallel_for<class Test>(sycl::nd_range<1>(32, 1),
9                                  [=](sycl::nd_item<1>) { array[0]++; });
10    }).wait();
11
12   sycl::free(array, Q);
13   return 0;
14 }
```

```
icpx -fsycl -Xarch_device -fsanitize=thread -O2 -g check_usm.cpp -o check_usm
==== DeviceSanitizer: TSAN
====WARNING: DeviceSanitizer: data race
When write of size 1 at 0xff00ffffffffe0000 in kernel LID(0, 0, 0) GID(29, 0, 0)
#0 typeinfo name for main::'lambda'(sycl::_V1::handler&)::operator()(sycl::_V1::handler&) const::Test /tmp/check_usm.cpp:9
```

Example: TSan to detect data race for OpenMP code

```
  1 #include <omp.h>
  2 #include <stdio.h>
  3 #include <stdlib.h>
  4 #define length     64
  5 int main(void) {
  6   int device_id = omp_get_default_device();
  7   size_t bytes = length * sizeof(double);
  8   double *__restrict A = (double *)omp_target_alloc_shared(bytes,
device_id);
  9   if (A == NULL) {
 10       printf(" ERROR: Cannot allocate space for A using
omp_target_alloc().\n");
 11     exit(1);
 12   }
 13   for (size_t idx = 0; idx < length; ++idx) A[idx] = idx + 0.5;
 14   double sum = 0.0;
 15 #pragma omp target teams distribute parallel for is_device_ptr(A)
 16   for (size_t i = 0; i < length; i++) {
 17     A[0] += 2.0;
 18   }
 19   omp_target_free(A, device_id);
 20   return 0;
 21 }
```

```
icx -fiopenmp -fopenmp-targets=spir64 -Xarch_device -
fsanitize=thread -O2 -g check_omp.c -o check_omp
==== DeviceSanitizer: TSAN ====WARNING:
DeviceSanitizer: data race When write of size 8 at
0x7fc1f2600000 in kernel
<__omp_offloading_802_36012d__Z4main_l13> LID(0, 0,
0) GID(57, 0, 0) #0 OMP TARGET outline from main line
13 /tmp/check_omp.c:17
```

More references

- Intel Distribution for GDB at ALCF INCITE GPU Hackathon 2025 by Sergey Kiselev (Intel)
  - https://www.youtube.com/watch?v=iEiGnPsVPck
  - https://www.alcf.anl.gov/support-center/training/intel-distribution-gdb
- Linaro DDT at ALCF INCITE GPU Hackathon 2025 by Rudy Shand (Linaro)
  - https://www.alcf.anl.gov/support-center/training/linaro-ddt
  - https://www.alcf.anl.gov/support-center/training/ddt-aurora
- Intel Distribution for GDB documentation
  - https://www.intel.com/content/www/us/en/developer/tools/oneapi/distribution-for-gdb-documentation.html
- Intel Sanitizer documentation
  - https://www.intel.com/content/www/us/en/developer/articles/technical/find-bugs-quickly-using-sanitizers-with-oneapi-compiler.html
- Linaro DDT user-guide
  - https://docs.linaroforge.com/25.0.4/html/forge/ddt/index.html
- ALCF user-guide
  - https://docs.alcf.anl.gov/aurora/debugging/gdb-oneapi/
  - https://docs.alcf.anl.gov/aurora/debugging/ddt-aurora/
- ALCF beginners guide for debuggers on Aurora
  - https://github.com/argonne-lcf/ALCFBeginnersGuide/blob/master/aurora/02_a_debugger.md


Argonne
NATIONAL LABORATORY
Thanks