Intel® Distribution for GDB*

Sergey Kiselev

Technical Consulting Engineer
Intel Corporation

Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation. Learn more at intel.com or from the OEM or retailer.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804. https://software.intel.com/en-us/articles/optimization-notice

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. See backup for configuration details. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Agenda

- Intel® Distribution for GDB* Overview
- Debugging GPU Offloaded Code
  - Compilation flow and debug information
  - GPU debug model
  - Compute kernel debugging
  - SIMD lanes
  - Multi-device debugging
  - GDB commands relevant for GDB debugging
- GDB Demo
- Summary and References

Intel® Distribution for GDB* Overview

Intel® Distribution for GDB* Overview

- Intel® oneAPI
  - Set of toolkits that include software development tools and libraries
  - Advanced compilers, including C++ compiler with SYCL support
  - The goal is to enable developers to write code for heterogeneous and offload processors
- Intel® Distribution for GDB*
  - All standard GDB* features
    - Intel GDB team works with the GDB project to contribute back
  - 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
      - Automatically detect JIT-compiled, or dynamically loaded, kernel code
    - SIMD lanes: display lane information and switch among lanes

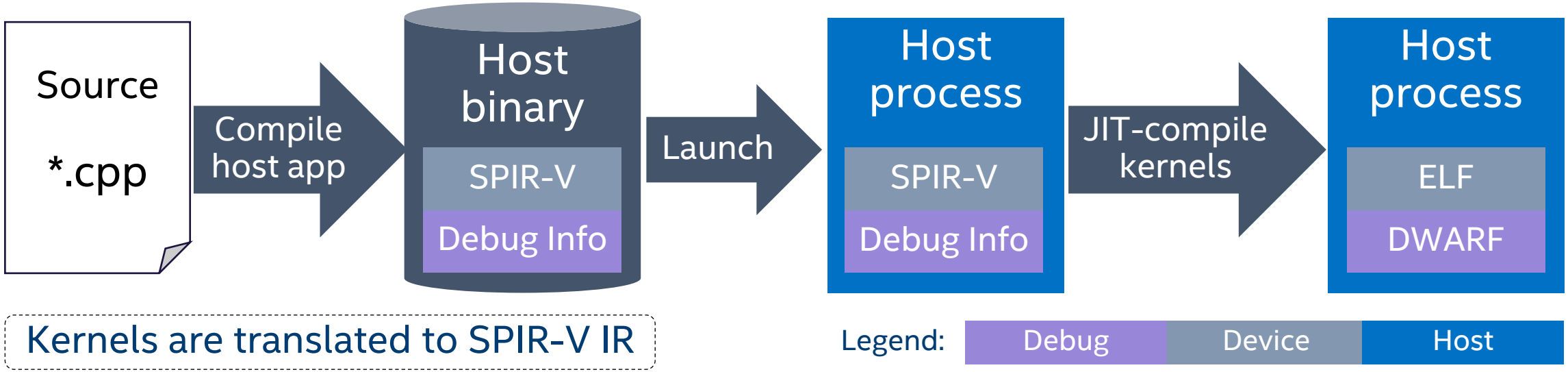
Fundamental GDB Commands

Command	Description
`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 specified line
`info break`	Show defined breakpoints
`delete <N>`	Remove Nth breakpoint
`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

Argonne NATIONAL LABORATORY

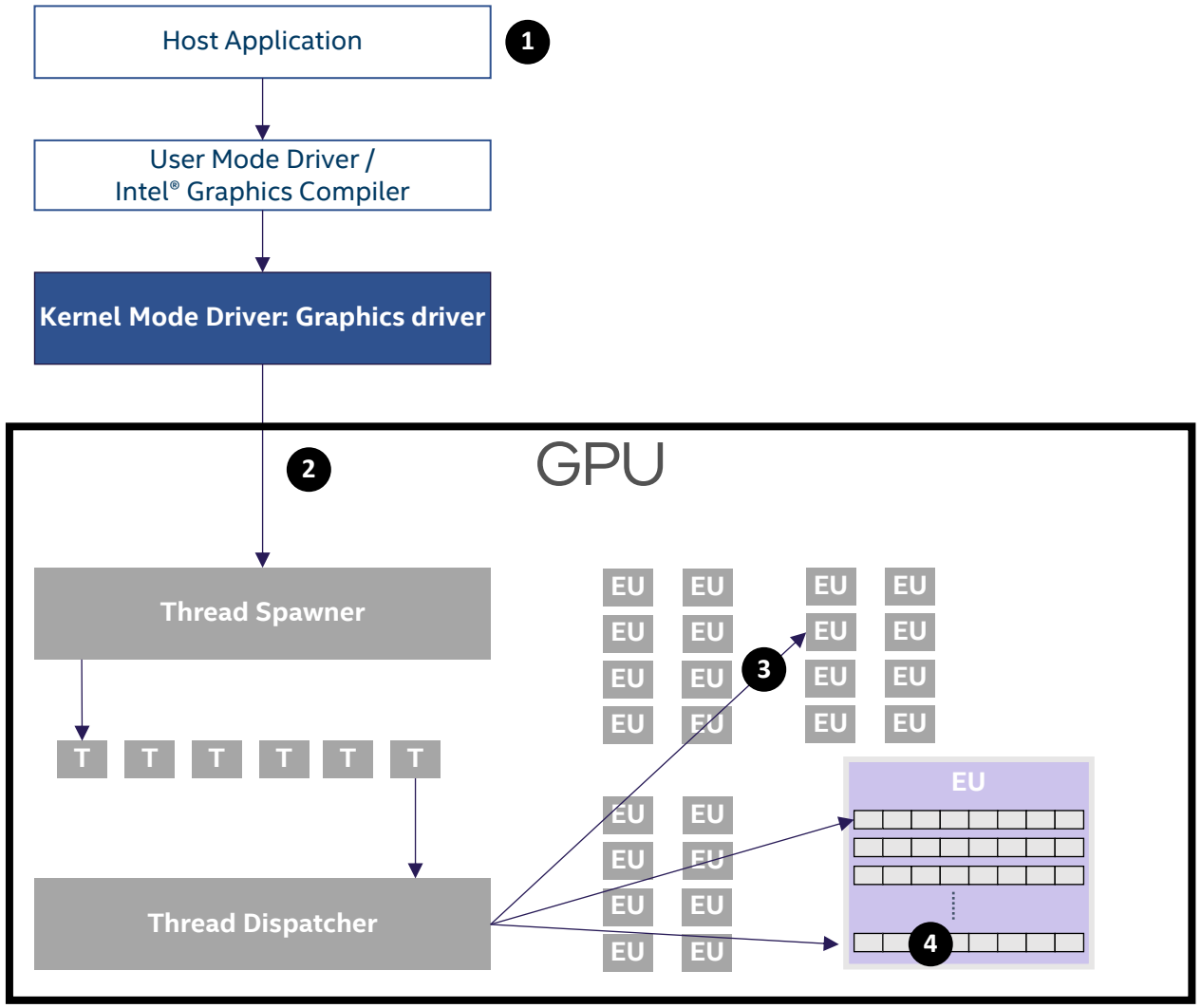
Debugging GPU Offloaded Code

Application Compilation (JIT)



- Compile with -g (generate debug information) and -O0 (disable optimizations) to debug.
  - May use –O2 to debug at assembly level
  - Use same optimization level when linking
- Debug also works with ahead-of-time (AOT) compilation

GPU Debug Model



1. Host inferior*
2. Device inferior*, one per Device / Tile
3. Device thread, one per EU thread
4. SIMD lanes, depending on SIMD width (1/8/16/32)

- *inferior ≈ debuggee process

Kernel Debugging, CPU vs GPU

- Behavior and commands are very similar to standard GDB*.
- CPU and GPU debugging experience is similar except:
- SIMD Lane Support
  - CPU: Cannot switch context to non-default SIMD lane.
  - GPU: May switch to a particular SIMD lane during debugging.
- Debugger calls to kernel functions
  - CPU: Supported
  - GPU: Not supported

Inferiors

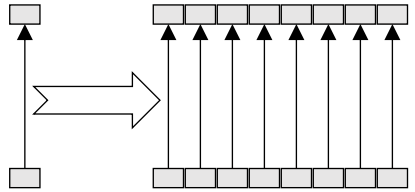
GDB uses *inferior* objects to represent states of program execution (usually a process)

- 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 -)
```

Debugging Threaded GPU SIMD Code

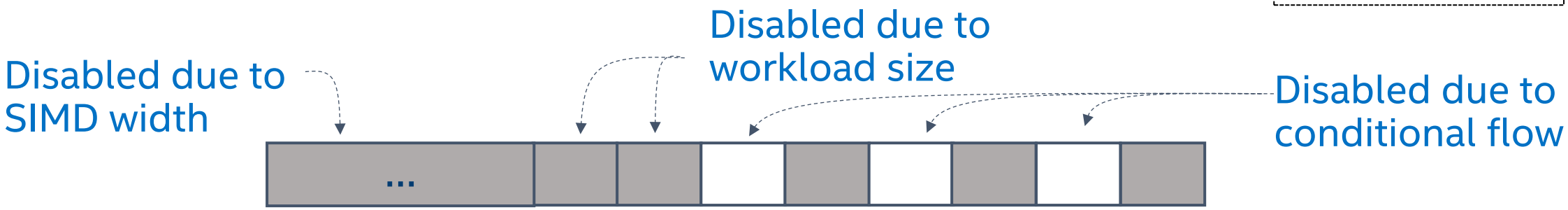
- Kernel code written for single work-item
- Code implicitly threaded and widened to vectors of work-items
- Variable locations expressed as functions of the SIMD lane
  - Lane field added to thread representation <inferior>.<thread>:<lane>
  - Applies to `info threads`, `thread`, `thread apply` …



```
(gdb) info thread 2.*
  Id           Target Id           Frame
* 2.1:0        Thread 1.1073741824 main::$_1::operator()<cl::sycl::handler>(cl::sycl::handler&) const::{lambc
    index=cl::sycl::id<1> = {...}) at array-transform.cpp:59
  2.1:[2 4 6]  Thread 1.1073741824 main::$_1::operator()<cl::sycl::handler>(cl::sycl::handler&) const::{lambc
    index=cl::sycl::id<1> = {...}) at array-transform.cpp:59
  2.2:[0 2 4 6] Thread 1.1073742400 main::$_1::operator()<cl::sycl::handler>(cl::sycl::handler&) const::{lamb
    index=cl::sycl::id<1> = {...}) at array-transform.cpp:59
  2.3:[0 2 4 6] Thread 1.1073742336 main::$_1::operator()<cl::sycl::handler>(cl::sycl::handler&) const::{lamb
    index=cl::sycl::id<1> = {...}) at array-transform.cpp:59
  2.4:[0 2 4 6] Thread 1.1073742144 main::$_1::operator()<cl::sycl::handler>(cl::sycl::handler&) const::{lamb
    index=cl::sycl::id<1> = {...}) at array-transform.cpp:59
```

SIMD Lanes Support

```
if (index % 2 == 0)
  out[index] = value;
else
  out[index] = -value;
```



- Only enabled SIMD lanes displayed
  - `info threads` to see active threads and lanes
- SIMD width is not fixed
- A user can switch only between enabled SIMD lanes
  - `thread <inferior>:<thread>.<lane>`
- After a stop, GDB switches to an enabled SIMD lane
- `print/t $emask` to see the execution mask

A thread might switch between different kernels with different SIMD widths

Multi-Device Debugging

- A program can offload a kernel to all or subset of GPU devices
- Intel® Distribution for GDB* can debug the CPU and GPUs in the same debug session
- User can switch to the context of a thread on a GPU or the CPU
- Each GPU device appears as a separate inferior (i.e. process)
- Inferior for a device does not appear if not used
- Threads of the GPUs can be independently resumed; thread state can be examined.

Multi-Device Debugging

```
multi-gpu.cpp
   31            cl::sycl::device device = devices[device_index];
   32            print_device ("Found", device);
   33
   34            return cl::sycl::queue {device}; /* return-sycl-queue */
   35          }
   36
   37          static void
   38          compute (cl::sycl::id<1> index)
   39          {
   40            int point = index[0];
B+ 41            int a = 33; /* kernel-line-1 */
   42            int b = 44;
B+>43            int c = 55; /* kernel-line-3 */
   44          }
   45
   46          static void
```

```
extended-r Thread 3.1073741824 In: _ZTSZZL3runRN2cl4sycl5queueEENKUlRNS0_7handlerEE54_24clES4_EUlNS0_2idILi1EEEE* L43   PC: 0xffde3640
(gdb) info threads
  Id           Target Id                                        Frame
  1.1          Thread 0x7ffff6e37000 (LWP 401277) "multi-gpu" 0x00007ffff725f89b in sched_yield ()
   from /lib/x86_64-linux-gnu/libc.so.6
  1.2          Thread 0x5ffff37c5700 (LWP 401281) "multi-gpu" 0x00007ffff727150b in ioctl () from /lib/x86_64-linux-gnu/libc.so.6
  2.1:[0-7]    Thread 2.1073741824                              compute (index=cl::sycl::id<1> = {...}) at multi-gpu.cpp:43
  2.2:[0-7]    Thread 2.1073741888                              compute (index=cl::sycl::id<1> = {...}) at multi-gpu.cpp:43
* 3.1:0        Thread 3.1073741824                              compute (index=cl::sycl::id<1> = {...}) at multi-gpu.cpp:43
  3.1:[1-7]    Thread 3.1073741824                              compute (index=cl::sycl::id<1> = {...}) at multi-gpu.cpp:43
  3.2:[0-7]    Thread 3.1073741888                              compute (index=cl::sycl::id<1> = {...}) at multi-gpu.cpp:43
(gdb)
```

- Second GPU's threads
- First GPU's threads
- Host application threads (CPU)

GDB Commands Relevant for GPU Debugging

Command	Description
`info inferiors`	Display information about the inferiors. GPU debugging will display additional inferior(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 specified 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 resumed (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

Commands with Different Behavior – CPU vs. GPU

Command	Description	GPU Behavior
`disassemble`	Disassemble code	GEN instructions and registers shown
`step, stepi, next`	Single-step source line into function calls, single-step machine instruction, single-step source line over function calls	SIMD lanes are supported, SIMD lane switches can occur

`break`	Create breakpoint	May create break point at a specific SIMD lane.
		`break 56 thread 2:3`
		May specify breakpoint for a particular inferior
		`break 56 inferior 2`

`commands`	Specify a list of commands for given breakpoints	With the `/a` modifier, breakpoint actions apply to all SIMD lanes

Enabling GPU Debug

- GPU Debugging is disabled by default for performance reasons
- Enable debugging in the Kernel Mode Driver (KMD):

```
echo 1 > /sys/class/drm/card0/prelim_enable_eu_debug
…
echo 1 > /sys/class/drm/cardN/prelim_enable_eu_debug
```

- Set ZET_ENABLE_PROGRAM_DEBUGGING=1 to enable debugging in the GPU Runtime (UMD)

```
export ZET_ENABLE_PROGRAM_DEBUGGING=1
```

- After finishing the debug session, disable debugging in the KMD:

```
echo 0 > /sys/class/drm/card0/prelim_enable_eu_debug
…
echo 0 > /sys/class/drm/cardN/prelim_enable_eu_debug
```

- Argonne Aurora specific:
  - ***helper_toggle_eu_debug.sh*** script to enable and disable KMD is provided here:
    - https://github.com/argonne-lcf/ALCFBeginnersGuide/blob/master/aurora/02_a_debugger.md

GPU Debug Demo

Sample GDB Session

```
$ gdb-oneapi -q ./array-transform
Reading symbols from ./array-transform...
(gdb) set print thread-events off
(gdb) break 61
Breakpoint 1 at 0x406362: file array-transform.cpp, line 61.
(gdb) run gpu
Starting program: /home/intel/oneAPI-samples/Tools/ApplicationDebugger/array-transform/src/array-transform gpu
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
intelgt: gdbserver-gt started for process 124229.
Will listen for an attached process
[SYCL] Using device: [Intel(R) Iris(R) Plus Graphics 655 [0x3ea5]] from [Intel(R) Level-Zero]
intelgt: attached to device 1 of 1; id 0x3ea5 (Gen9)
[New inferior 2]
[Switching to Thread 1.32768 lane 1]

Thread 2.1 hit Breakpoint 1, with SIMD lanes [1 3 5 7],
main::{lambda(auto:1&)#1}::operator()<cl::sycl::handler>(cl::sycl::handler&)
const::{lambda(cl::sycl::id<1>)#1}::operator()(cl::sycl::id<1>) const (this=0x1c49310, index=cl::sycl::id<1> = {...}) at
array-transform.cpp:61
61                result = -1;  // else-branch
(gdb) list
56              int element = in[index];  // breakpoint-here
57              int result = element + 50;
58              if (id0 % 2 == 0) {
59                result = result + 50;  // then-branch
60              } else {
61                result = -1;  // else-branch
62              }
63              out[index] = result;
64            });
65            // kernel-end
(gdb) print index
$1 = cl::sycl::id<1> = {1}
(gdb)
```

GPU Debugging Limitations

- If a bug occurs on both CPU and GPU, debug on the CPU
- Breakpoint must be set inside kernel to debug GPU
  - —Unable to step into the kernel, separate inferiors
- Debug process state in GPU hardware (not on CPU)
  - —Restricts GPU to single context (unable to perform other tasks)
  - —Display interruption for rendering GPU
- CPU polls status of debug process state through MMIO
  - —Increases load on host
- Inspecting shared local memory (SLM) is not supported
- See Release Notes for complete list

Additional Debug Scenarios

- Core dump Analysis
  - Enable core dumps:
    ```
    ulimit -c unlimited
    mpiexec -ppn 1 --rlimits CORE pwd
    ```
  - Run the application and generate core dump file:
    ```
    mpiexec –n N debugged_application <options>
    ```
  - Load application in GDB and print the backtrace:
    ```
    gdb-oneapi debugged_application corefile
    thread apply all bt
    ```
- MPI Debugging
  - Debug a single rank by running that rank under the GDB, for example:
    ```
    mpirun --env ZET_ENABLE_PROGRAM_DEBUGGING=1 -n 1 gdb-oneapi <mpi_app>: -n 1 <mpi_app>
    ```
  - Debug several ranks, using several GDB instances, each one in its own xterm:
    ```
    mpirun --env ZET_ENABLE_PROGRAM_DEBUGGING=1 -n 2 xterm –e gdb-oneapi <mpi_app>
    ```
  - Non-interactive debug:
    ```
    gdb-oneapi -batch -ex <cmd1> -ex <cmd2> … -ex <cmdN> --args <mpi_app> <app_args>
    ```
    - Wrapper script: https://docs.alcf.anl.gov/aurora/debugging/gdb-oneapi/#noninteractive-debugging
- Python Debugging
  - Mainly intended for debugging C API extensions and CPython Internals
  - Use ***`python-gdb.py`*** extension
  - Documentation: https://docs.python.org/3/howto/gdb_helpers.html

    source `/usr/share/gdb/auto-load/usr/lib64/libpython3.6m.so.1.0-gdb.py` in GDB

Summary and References

- Intel® Distribution for GDB* can be used to debug host and device for oneAPI applications written in various languages
- Traditional GDB commands have been extended to accommodate GPU execution mode
- References:
  - ALCF Beginners Guide - Debugging on Aurora
  - Debugging on Aurora with gdb-oneapi
  - Intel® Distribution for GDB* Get Started Guide
    - Linux, Windows
  - Debugging with Intel® Distribution for GDB Tutorial
    - Linux, Windows
  - Intel® Distribution for GDB* Release Notes
  - Intel® Distribution for GDB* Reference Sheet
  - Debugger Samples on GitHub

What questions do you have?