LRZ Workshop

Intel® Distribution for GDB* A Cross-Architecture Application Debugger

Alina Shadrina

alina.shadrina@intel.com

Agenda

- System Requirements Overview
- Key features
- DPC++ Linux* Demo
- Debugging Multi-Tile GPU
- C++: Debugging OpenMP* offload
- Other Debug Capabilities

System Requirements Overview

Windows*

Language Support

- Data Parallel C++ (DPC++)
- C \ C++
- Fortran
- OpenMP

IDE Support

- Microsoft Visual Studio 2022*
- Visual Studio Code *

OS Support

- Windows* 10, 64-bit
- Windows* 11, 64-bit

GPUs

- Intel® Arc™ Series

CPUs

- Intel® Core™ Processor family
- Intel® Xeon® Processor family
- Intel® Xeon® Scalable Performance processors

FPGA

- Emulation device only

Intel® Distribution for GDB* Release Notes
Intel® Distribution for GDB* System Requirements

Linux*

Language Support

- Data Parallel C++ (DPC++)
- C \ C++
- Fortran
- OpenMP

IDE Support

- Eclipse *
- Visual Studio Code *

OS Support

- Ubuntu* 20.04, 22.04
- SLES* 15
- RHEL* 8, 9

GPUs

- Intel® Arc™ Series
- Intel® Data Center GPU Flex Series
- Intel® Data Center GPU Max

CPUs

- Intel® Core™ Processor family
- Intel® Xeon® Processor family
- Intel® Xeon® Scalable Performance processors

FPGA

- Emulation device only

Intel® Distribution for GDB* Release Notes
Intel® Distribution for GDB* System Requirements

Key features

- Command line debugging on the same machine: `gdb-oneapi`
- IDE Integration
  - 2 machines required: CPU host and GPU target
- Device support:

Multi-node debugging	MPI applications	Not supported
Multi-thread debugging	On the same GPU	Supported
Multi-user debugging	On the same GPU	Not supported; GPU is blocked by the debugger
Multi-target debugging	debug GPU and CPU code in the same session	Supported

CPU and GPU Debugging:
Major Differences

Aspect	Description	CPU	GPU
Threads and single instruction, multiple data (SIMD) lanes	When the code is vectorized, threads process vectors of data elements in parallel	Not supported	Context switch supported
Inferior calls	Inferior calls are calls to kernel functions from inside the debugger as part of expression evaluation	Inferior calls are supported.	Inferior calls are not supported.

CPU and GPU Debugging:
Commands Differences

Command	Description	GPU Modification	Example
`disassemble`	Disassemble the current function.	GEN instructions and registers are shown.	`N/A`
`step`	Single-step a source line, stepping into function calls.	SIMD lanes are supported, and SIMD lane switches can occur.	`next` `[Switching to SIMD lane0]`
`stepi`	Single-step a machine instruction.		
`next`	Single-step a source line, stepping over function calls.		
`thread`	Switch context to the SIMD lane of the specified thread.	SIMD lanes are supported.	`thread 2.5:1`
`thread apply`	Apply a command to the specified SIMD lane of the thread.	SIMD lanes are supported.	`thread apply 2.3:* print element`

CPU and GPU Debugging:
Commands Differences

Command	Description	GPU Modification	Example
`info threads`	Display information about threads with ID, including their active SIMD lanes.	SIMD lanes are supported.	N/A

`commands`	Specify a list of commands to execute when your program stops due to a particular breakpoint.	**`/a`** modifier - breakpoint actions apply to all SIMD lanes that match the condition of the specified breakpoint.	`commands /a`
			`print element`
			`end`

`break`	Create a breakpoint at a specified line.	Create a breakpoint at a special SIMD lane 3 of thread 2	`break 56 thread 2:3`
		Specify a breakpoint for a particular inferior 2	`break 56 inferior 2`

DPC++ Linux* Demo (Command Line)

Array Transform Sample

- Prerequisites:
  - Get Started Guide to configure the debugger

- Clone **oneAPI-samples**:

```
git clone https://github.com/oneapi-src/oneAPI-samples.git
cd oneAPI-samples/Tools/ApplicationDebugger/array-transform
```

- Set oneAPI environment:

```
source /opt/intel/oneapi/setvars.sh
```

Array Transform Sample

- Enable i915 debug support in kernel persistently:
  - Requires sudo!
  - `cat /etc/default/grub`
  - Make sure your `GRUB_CMDLINE_LINUX_DEFAULT` contains:

```
 i915.debug_eu=1 drm.debug=0xa i915.enable_hangcheck=0
i915.debugger_timeout_ms=0
```

- Enable i915 debug support in Kernel:
  - `cat /sys/class/drm/card*/prelim_enable_eu_debug`
  - Make sure the output is `1`

Diagnostics Utility

- For the default oneAPI installation:
  - `python3 /opt/intel/oneapi/diagnostics/latest/diagnostics.py --filter debugger_sys_check -force`
- Expected output:

```
Checks results:

================================================================================
Check name: debugger_sys_check
Description: This check verifies if the environment is ready to use gdb (Intel(R) Distri
bution for GDB*).
Result status: PASS
Debugger found.
libipt found.
libiga found.
i915 debug is enabled.
Environmental variables correct.
================================================================================

1 CHECK: 1 PASS, 0 FAIL, 0 WARNINGS, 0 ERRORS
```

Array Transform Sample on CPU

- Build:

```
icpx -fsycl -g -O0 array-transform.cpp -o array-transform.exe
```

- Run:

```
ONEAPI_DEVICE_SELECTOR=*:cpu ./array-transform.exe cpu
```

- Run under the debugger:

```
ONEAPI_DEVICE_SELECTOR=*:cpu gdb-oneapi --args ./ array-transform.exe cpu
```

Array Transform Sample on GPU

- Build:

```
icpx -fsycl -g -O0 jacobi.cpp -o array-transform.exe
```

- Run:

```
ONEAPI_DEVICE_SELECTOR=level_zero:gpu gdb-oneapi ./ array-transform.exe gpu
```

- Enable debugging:

```
export ZET_ENABLE_PROGRAM_DEBUGGING=1
export IGC_EnableGTLocationDebugging=1
```

- Run under the debugger:

```
ONEAPI_DEVICE_SELECTOR=level_zero:gpu gdb-oneapi --args ./ array-transform.exe gpu
```

Debugging on GPU

- `info inferiors` - make sure you are on GPU now
- `info threads` - inspect threads
- `thread 2.<Thread_number>:<SIMD_lane>` - switching between threads
- `info locals` - print local threads variables
- `disassemble` - see disassemble
- `set scheduler-locking step` - step to the next

Debugging Multi-Tile GPU

ZE_AFFINITY_MASK

Value	Behavior
0, 1	all devices and sub-devices are reported (same as default)
0	only device 0 is reported; with all its sub-devices
1	only device 1 is reported as device 0; with all its sub-devices
0.0	only device 0, sub-device 0 is reported as device 0
1.1	only device 1 is reported as device 0; with its sub-devices 1 and 2 reported as sub-devices 0 and 1, respectively
0.2, 1.3, 1.0, 0.3	both device 0 and 1 are reported; device 0 reports sub-devices 2 and 3 as sub-devices 0 and 1, respectively; device 1 reports sub-devices 0 and 3 as sub-devices 0 and 1, respectively; the order is unchanged.

Selecting Different Devices

- `$ gdb-oneapi --args ./array-transform.exe gpu`

```
(gdb) info devices
  Location   Sub-device   Vendor Id   Target Id   Cores   Device Name
  [3a:00.0]  -            0x8086      0x0bd5      1024    Intel(R) Graphics [0x0bd5]
* [9a:00.0]  -            0x8086      0x0bd5      1024    Intel(R) Graphics [0x0bd5]
```

- `$ ZE_AFFINITY_MASK=0.0 gdb-oneapi --args ./array-transform.exe gpu`

```
(gdb) info devices
  Location   Sub-device   Vendor Id   Target Id   Cores   Device Name
* [9a:00.0]  -            0x8086      0x0bd5      512     Intel(R) Graphics [0x0bd5]
```

- `$ ZE_AFFINITY_MASK=1.0 gdb-oneapi --args ./array-transform.exe gpu`

```
(gdb) info devices
  Location   Sub-device   Vendor Id   Target Id   Cores   Device Name
* [3a:00.0]  -            0x8086      0x0bd5      512     Intel(R) Graphics [0x0bd5]
```

Debugging OpenMP* Offload (C++)

Matmul build and run

- Build :
  - `icpx -O0 -g -fiopenmp -fopenmp-targets=spir64 matmul_offload.cpp -o matmul_debug`
- Disable device optimizations:

```
export ZET_ENABLE_PROGRAM_DEBUGGING=1
export IGC_EnableGTLocationDebugging=1
```

- Set up offloading:
  - `export OMP_TARGET_OFFLOAD="MANDATORY"`
- Debug :
  - `gdb-oneapi ./matmul_debug`

Expect full support for OpenMP offload for Fortran soon!

Other Debug Capabilities

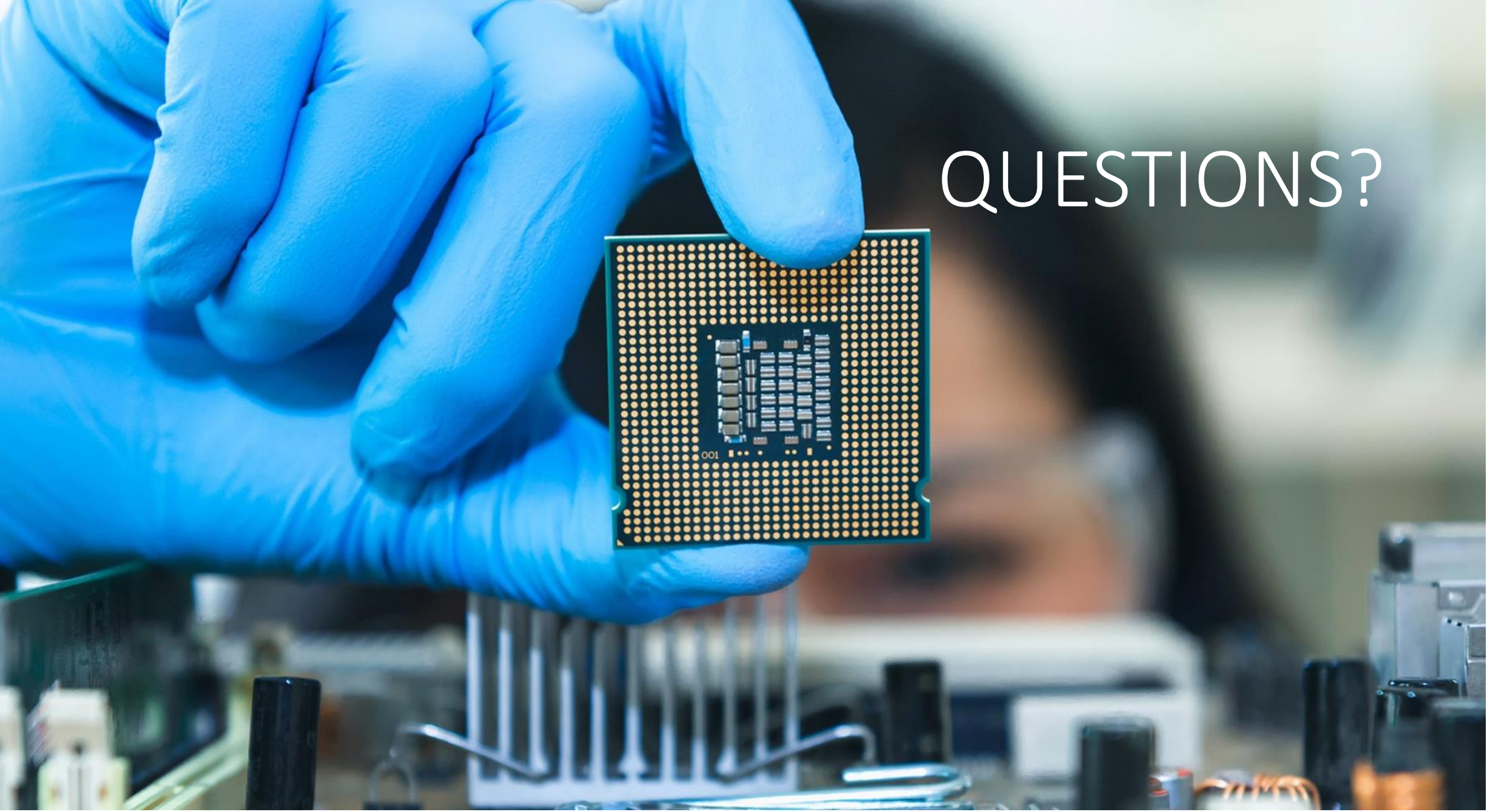
oneAPI Debug Tools and Variables

- Specified level of tracing for SYCL Plugin Interface:
  - `SYCL_PI_TRACE={1,2,-1}`
- GPU backends:
  - Profiling Tools Interfaces for GPU (PTI GPU) - Level Zero Tracer ze_tracer
  - Intercept Layer for OpenCL - How to Use the Intercept Layer for OpenCL™ Applications
- OpenMP Offload: `LIBOMPTARGET_DEBUG`
- Clang Sanitizers

Intel LLVM Compiler: Environment Variables docs

Useful Links

- Basic:
  - Documentation & Code Samples
  - Intel® Distribution for GDB* Release Notes
  - Intel® Distribution for GDB* System Requirements
- Advanced:
  - oneAPI Debug Tools at Intel® oneAPI Programming Guide
  - Get Started with OpenMP* Offload to GPU for the Intel® oneAPI DPC/C++ Compiler and Intel® Fortran Compiler


QUESTIONS?

Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

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

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

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


intel