oneAPI

LRZ Workshop

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

Alina Shadrina

alina.shadrina@intel.com

intel

Agenda

- System Requirements Overview
- Key features
- Troubleshooting
- DPC++ Linux* Demo
- 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 2017*
- Microsoft Visual Studio 2019*
- Microsoft Visual Studio 2022*
- Visual Studio Code *

OS Support

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

GPUs

- Intel® HD Graphics Gen9
- Intel® Iris® Xe Graphics

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* 18.x, 20.04
- CentOS* 7
- Fedora* 34
- SLES 15

GPUs

- Intel® HD Graphics Gen9
- Intel® Iris® Xe Graphics

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`

Troubleshooting

- Companion driver not installed properly:
  - Incorrect behavior:

```
$ gdbserver-gt --attach --hostpid=999 :1234 1
no device '1' found, there are 0 devices
Exiting
```

  - Expected behavior :

```
$ gdbserver-gt --attach --hostpid=999 :1234 1
intelgt: attached to device 1 of 1; id 0x5927 (Gen9)
Attached; pid = 1
Listening on port 1234
```

  - **Solution:** review the GPU installation and configuration instructions to ensure that you set up the device correctly.

DPC++ Linux* Demo (Command Line)

Jacobi Sample

- Prerequisites:
  - Get Started Guide to configure the debugger
  - array-transform sample

- Clone **oneAPI-samples**/Tools/ApplicationDebugger/**jacobi/**
- `source /opt/intel/oneapi/setvars.sh`

Jacobi Sample

```
          A                  x   =   b

[5 1 1 0 0     ... 0 0 0 0]  [1]     [7]
[1 5 1 1 0 0 0 ... 0 0 0 0]  [1]     [8]
[1 1 5 1 1 0 0 ... 0 0 0 0]  [1]     [9]
[0 1 1 5 1 1 0 0 ... 0 0 0]  [1]     [9]
[0 0 1 1 5 1 1 0 0 ... 0 0]  [1]  =  [9]
[...]                       [...]   [...]
[0 0 0 0 ... 0 1 1 5 1 1 0]  [1]     [9]
[0 0 0 0 ... 0 0 1 1 5 1 1]  [1]     [8]
[0 0 0 0 ... 0 0 0 0 1 1 5]  [1]     [7]
```

linear system of equations

$$Ax=b$$

Where:

- A: n x n
- b: n x 1
- x: n x 1 – solution vector

Jacobi Sample on CPU

- Build `dpcpp -g -O0 jacobi-bugged.cpp -o jacobi-bugged.exe`
- Run `./jacobi-bugged.exe cpu`
- Check output. It indicates some bugs

```
fail; Bug 1. Fix this on CPU: components of x_k are not close to 1.0.
Hint: figure out which elements are farthest from 1.0.
```

- Open sources
- Run under the debugger:

```
gdb-oneapi --args ./jacobi-bugged.exe cpu
```

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 OpenMP* Offload (C++)

Matmul build and run

- Build :
  - `icx -O0 -g -fiopenmp -fopenmp-targets=spir64 matmul_offload.cpp -o matmul_debug`
- Disable device optimizations:
  - `export LIBOMPTARGET_OPENCL_COMPILATION_OPTIONS="-g -cl-opt-disable"`
  - `export LIBOMPTARGET_LEVEL0_COMPILATION_OPTIONS="-g -cl-opt-disable”`
- Set up offloading:
  - `export OMP_TARGET_OFFLOAD="MANDATORY"`
- Debug :
  - `gdb-oneapi ./matmul_debug`

Debugging OpenMP offload for Fortran is not supported yet!

Get Started with OpenMP* Offload to GPU for the Intel® oneAPI DPC/C++ Compiler and Intel® Fortran Compiler

Other Debug Capabilities

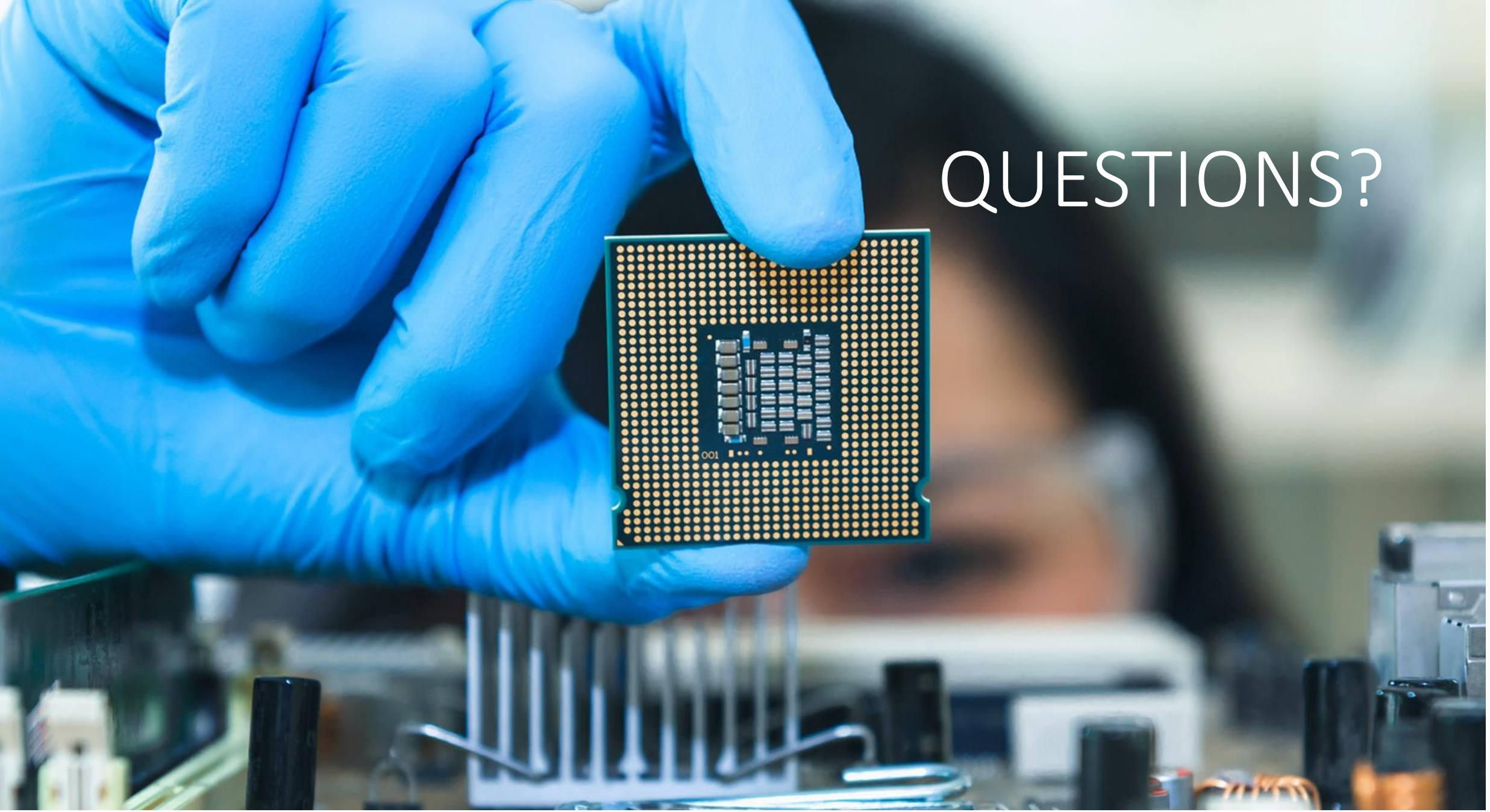
oneAPI Debug Tools and Variables

- 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`

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