Debugging oneAPI applications with The Intel® Distribution for GDB*

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.

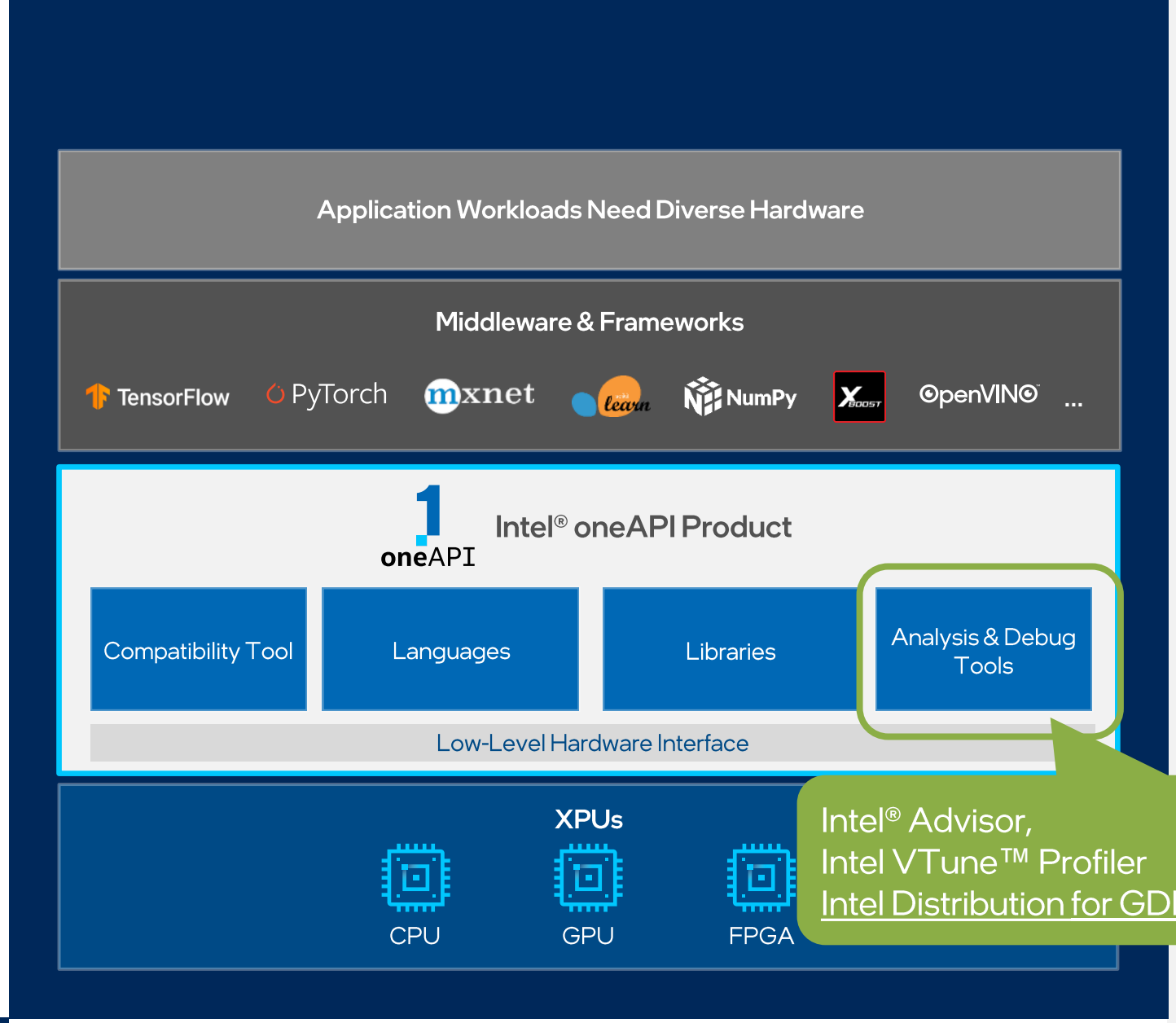
Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Intel® oneAPI Product

Built on Intel's Rich Heritage of CPU Tools Expanded to XPUs

A complete set of advanced compilers, libraries, and porting, analysis and debugger tools

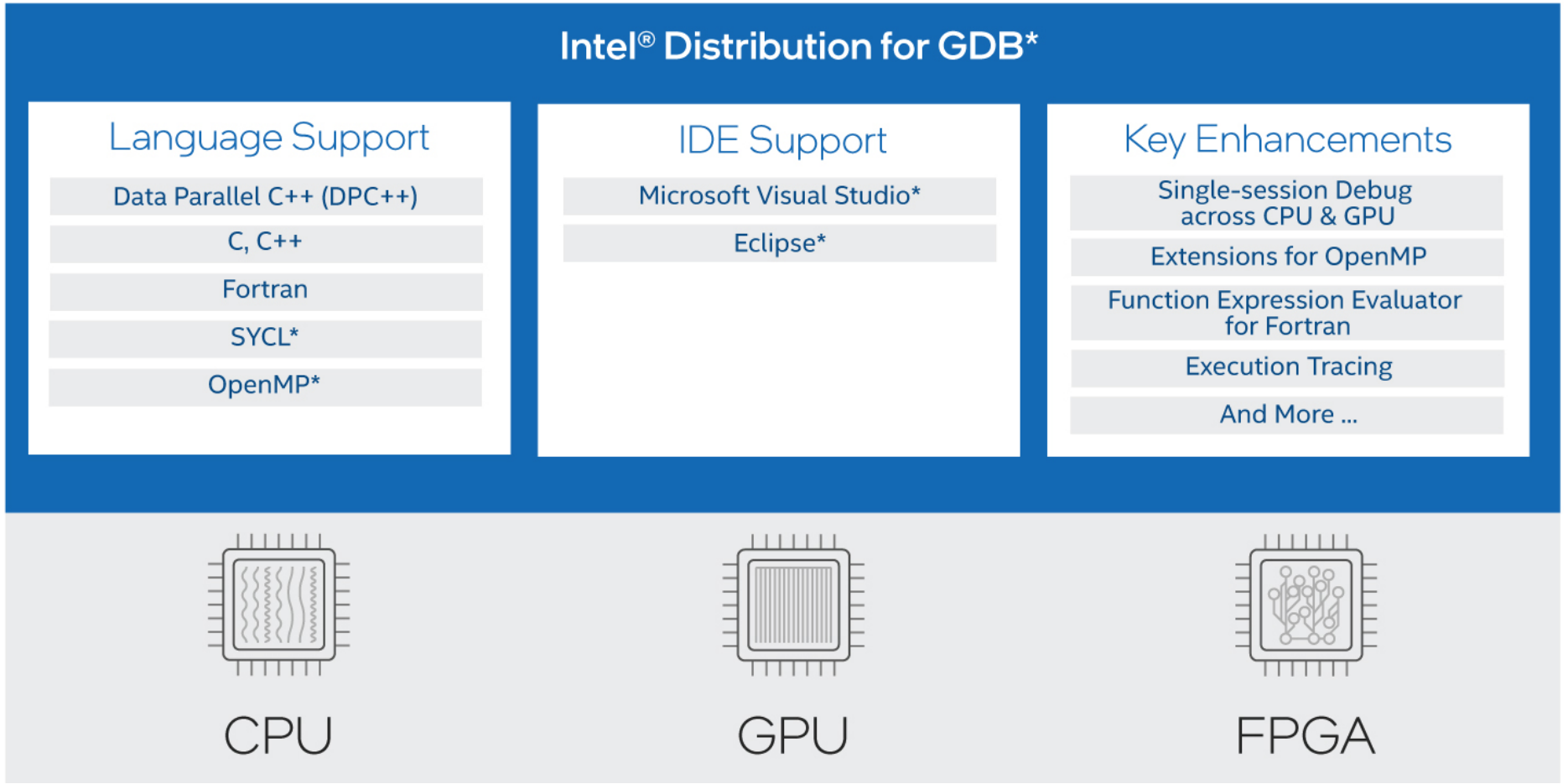
- Accelerates compute by exploiting cutting-edge hardware features
- Interoperable with existing programming models and code bases (C++, Fortran, Python, OpenMP, etc.), developers can be confident that existing applications work seamlessly with oneAPI
- Eases transitions to new systems and accelerators—using a single code base frees developers to invest more time on innovation



Available Now

Visit software.intel.com/oneapi for more details
Some capabilities may differ per architecture and custom-tuning will still be required. Other accelerators to be supported in the future.

Overview of Intel® Distribution of GDB



- Focus of this workshop is on debugging DPC++ GPU kernels

Setup for GPU (Linux)

Install the debug companion driver (DCD)

```
$ tree /opt/intel/oneapi/debugger/latest/
├── dep
│   └── lib
├── documentation
│   ├── gdb.pdf
│   ├── info
│   └── man
├── env
│   └── vars.sh
├── gdb
│   └── intel64
├── igfxdcd-1.8.0-Linux.deb
├── igfxdcd-1.8.0-Linux.rpm
├── …
```

User Manual

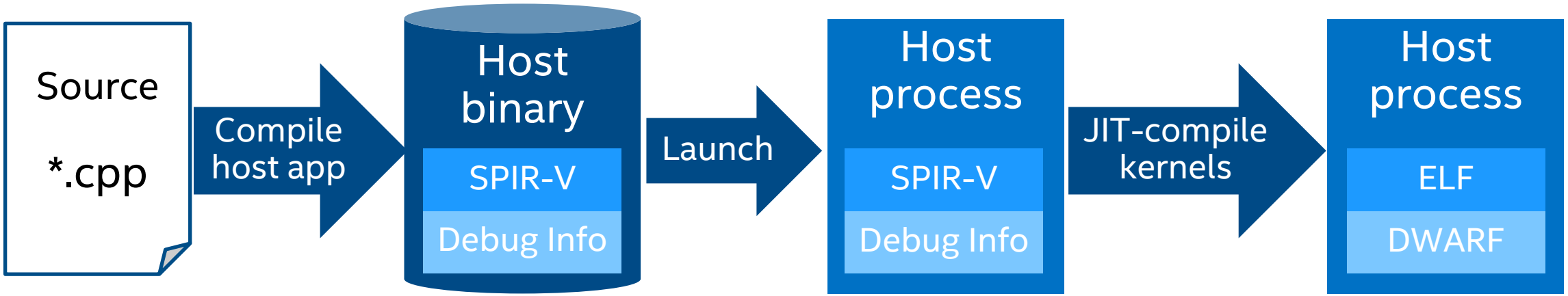
Debug driver

```
$ sudo dpkg -i path/to/igfxdcd-1.8.0-Linux.deb
$ sudo modprobe igfxdcd
```

Additional setup steps maybe necessary, see Get Started guide

Application Compilation (JIT)

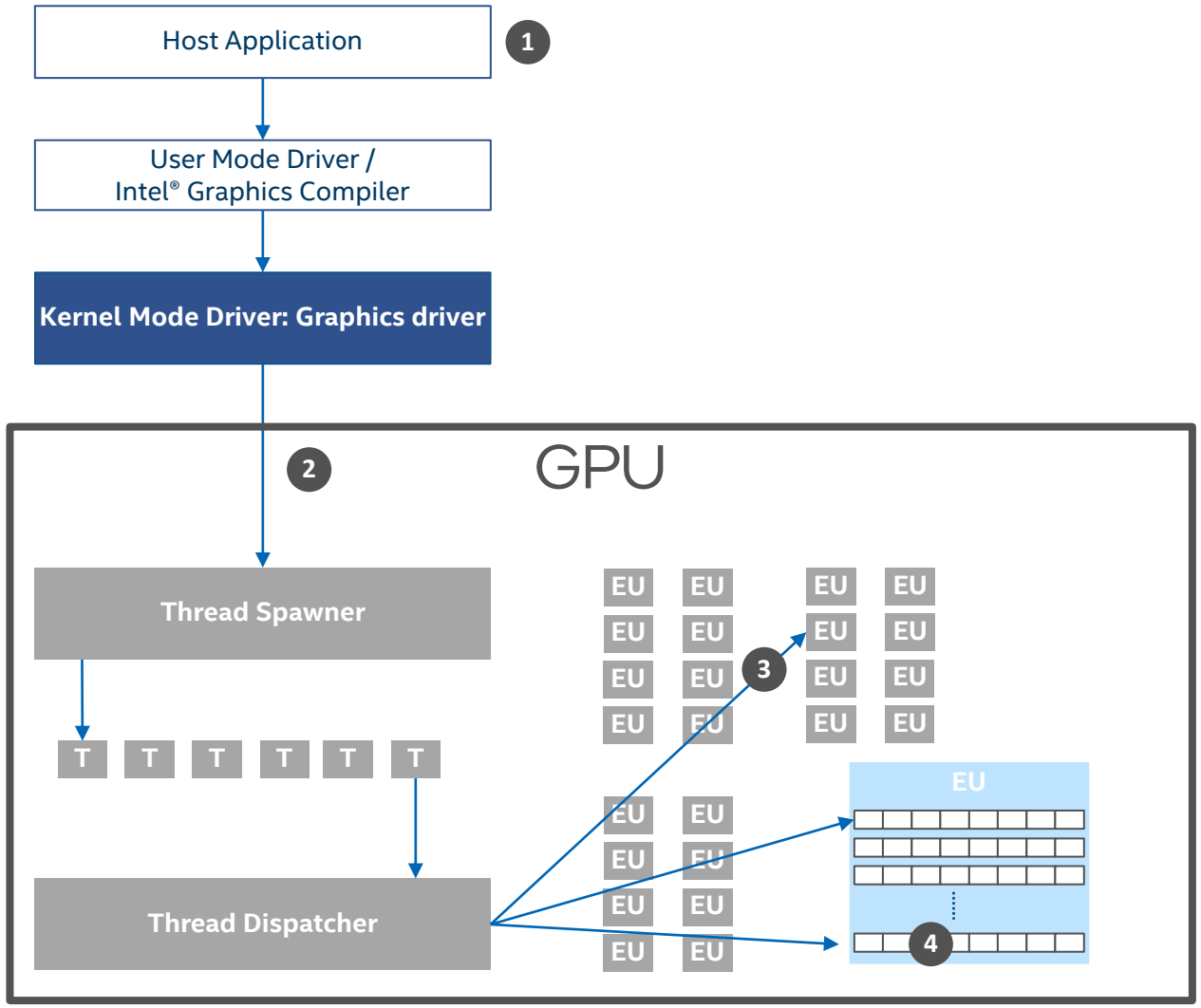
Legend: Debug | Device | Host



Kernels are translated to SPIR-V IR

- Compile with -g (generate debug information) and -O0 (disable optimization) 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

Fundamental GDB Commands

Command	Description
`help <cmd>`	Print help info
`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

GPU-Relevant GDB Commands

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

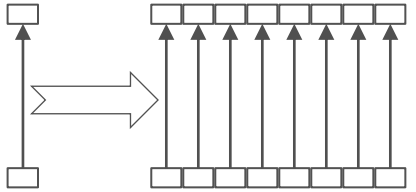
Inferiors

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


```
$ dpcpp -g -O0 array-transform.cpp -o array-transform
$ gdb-oneaopi -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

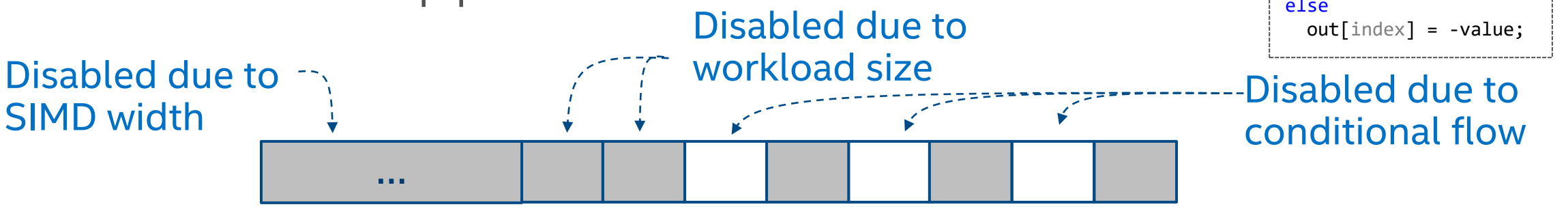


- Kernel code written for single work-item
- Code implicitly threaded and widened to vectors of work-items
- Variable locations expressed as functions of the SIMD lane
  - Lane field added to thread representation <inferior>.<thread>:<lane>
  - Applies to `info threads`, `thread`, `thread apply` …

```
(gdb) info thread 2.*
  Id           Target Id           Frame
* 2.1:0        Thread 1.1073741824 main::$_1::operator()<cl::sycl::handler>(cl::sycl::handler&) const::{lambo
    index=cl::sycl::id<1> = {...}) at array-transform.cpp:59
  2.1:[2 4 6]  Thread 1.1073741824 main::$_1::operator()<cl::sycl::handler>(cl::sycl::handler&) const::{lambo
    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
- SIMD width is not fixed
- A user can switch only between enabled SIMD lanes
- After a stop, GDB switches to an enabled SIMD lane
- If target architecture does not support SIMD or thread SIMD width is 1, GDB behavior is unchanged

A thread might switch between different kernels with different SIMD widths

A Sample GDB Session

```
$ gdb -q sycl-if
Reading symbols from sycl-if...
(gdb) set print thread-events off
(gdb) break 21
Breakpoint 1 at 0x40497b: file src/sycl-if.cpp, line 21.
(gdb) ignore 1 41
Will ignore next 41 crossings of breakpoint 1.
(gdb) run
Starting program: .../sycl-if
[...]
intelgt: attached to device with id 0x3ea5 (Gen9)
intelgt: inferior 2 (gdbserver-gt) created for process 1555133.
[Switching to Thread 1073746465 lane 1]

Thread 2.43 hit Breakpoint 1,  with SIMD lanes [1 3 5 7], compute(int*, int*)::$_0::operator()(cl::sycl::handler&)
const::{lambda(cl::sycl::id<1>)#1}::operator()(cl::sycl::id<1>) const (this=0x2dfff00, index=...) at sycl-if.cpp:21
21          out[index] = -value;
(gdb) list
16      cgh.parallel_for<class kernel>(range, [=](sycl::id<1> index) {
17        int value = in[index];
18        if (index % 2 == 0)
19          out[index] = value;
20        else
21          out[index] = -value;
22      });
23    });
24  }
25
(gdb) print index
$1 = {<cl::sycl::detail::array<1>> = {common_array = {617}}, <No data fields>}
(gdb)
```

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)

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 on 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
- See Release Notes for complete list

Case Study: Using Intel® Distribution of GDB* to Debug SYCL ZFP Compression Library

- ZFP: open-source library for compressed floating-point and integer arrays
  - Supports high-throughput random access read and write to individual elements
- Intel® DPC++ Compatibility Tool used to assist the migration of the CUDA source to DPC++
  - Unit test failures for 1D double, 2D float, 2D double, and all 3D cases.
- Intel® Distribution for GDB* used to identify the bug

Debugging ZFP

1. Identify indices of the array that was failing
   - Incorrect encoding due to overflow was happening for certain indices
2. Start debugger
3. Set conditional breakpoint in kernel
   - `b encode1.dp.hpp:64 if block_idx==65534`
4. After stepping through kernel code, an unnecessary typecast from double to float was discovered, causing overflow

References

- 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

Summary

- 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 model


intel