DEBUGGING SYCL PROGRAMS ON HETEROGENEOUS INTEL® ARCHITECTURES

Natalia Saiapova

Intel GmbH, Munich, Germany
Several people contributed to various parts of the debugger presented in this talk

<table>
<thead>
<tr>
<th>Arik Adler</th>
<th>Markus Metzger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sanimir Agovic</td>
<td>Ofer Rubin</td>
</tr>
<tr>
<td>Tankut Baris Aktemur</td>
<td>Natalia Saiapova</td>
</tr>
<tr>
<td>Albertano Caruso</td>
<td>Fabian Schnell</td>
</tr>
<tr>
<td>Ofir Cohen</td>
<td>Mihails Strasuns</td>
</tr>
<tr>
<td>Mircea Gherzan</td>
<td>Kai Trojahner</td>
</tr>
</tbody>
</table>
Talk plan

- oneAPI programming model
- How does a debugger work?
- Architecture
- Demo
- Challenges
oneAPI programming model

- **Problem:**
  - Modern problems imply workload diversity
  - Variety of hardware (CPU, **GPU**, accelerators) and programming languages, APIs, tools, libraries needed to achieve best performance

- **Aim:** a unified programming model to deliver uncompromised performance for diverse workloads across multiple architectures

Beta release of Intel® oneAPI products made in November

[https://software.intel.com/oneapi](https://software.intel.com/oneapi)
How does a debugger work?

- Debugger – separate process
- Translates between source-level and machine-level worlds
- OS provides debugger with:
  - Permission to control another process
  - Access to debuggee’s memory and its threads’ register state
  - Means to alter execution of debuggee
- Exceptions from the debuggee are delivered to the debugger
SYCL application compilation

SYCL source code *.cpp

Compile host app

Host binary

Host process

Launch

Jit-compile kernels

Host process

Kernels are translated to SPIR-V

Standard GDB can debug the host part.

Kernels offloaded to GPU device are transparent to the debugger!

Legend:
- Debug
- Device
- Host

Refer to [software.intel.com/articles/optimization-notice](https://software.intel.com/articles/optimization-notice) for more information regarding performance & optimization choices in Intel software products.

Copyright ©, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.
Detailed view on architecture

Multi-target GDB* Intel enhancements

Host process
- SYCL RT
- L0 driver

Kernel | ptrace
- CPU

Gdbserver-gt Intel-gt target L0 debug driver

i915 | dbg
- GPU

Legend:
- Intel
- 3rd party
- Hardware

Refer to software.intel.com/articles/optimization-notice for more information regarding performance & optimization choices in Intel software products.

Copyright ©, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Detailed view on architecture

- multi-target GDB*
  - Intel enhancements
- Host process
  - SYCL RT
  - L0 driver
- kernel | ptrace
- gdbserver-gt
  - Intel-gt target
  - L0 debug driver

Legend:
- Intel
- 3rd party
- Hardware

Refer to software.intel.com/articles/optimization-notice for more information regarding performance & optimization choices in Intel software products.
Copyright ©, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Detailed view on architecture

We represent Intel GT devices as separate inferiors with a remote connection to retargeted gdbserver

Legend:
- Intel
- 3rd party
- Hardware

Refer to software.intel.com/articles/optimization-notice for more information regarding performance & optimization choices in Intel software products.

Copyright ©, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Detailed view on architecture

Published as part of level-zero spec to enable 3rd party tool development: https://spec.oneapi.com/oneL0/

Legend: Intel 3rd party Hardware
Detailed view on architecture

multi-target GDB*
Intel enhancements

Host process
SYCL RT
L0 driver

For GPU it uses NEO as its backend. IGC for JIT compilation.

kernel | ptrace

CPU | CPU | CPU | CPU

i915 | dbg

GPU | GPU | GPU

Legend: Intel | 3rd party | Hardware

Refer to software.intel.com/articles/optimization-notice for more information regarding performance & optimization choices in Intel software products.
Copyright ©, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
#include <CL/sycl.hpp>
using namespace cl::sycl;

void compute(int input[], int output[]) {
    queue device_queue; // picks default device
    range<1> range{64};
    buffer<int, 1> buffer_in{input, range};
    buffer<int, 1> buffer_out{output, range};

    device_queue.submit([&](handler & cgh) {
        auto in = buffer_in.get_access<access::mode::read>(cgh);
        auto out = buffer_out.get_access<access::mode::write>(cgh);

        cgh.parallel_for<class kernel>(range, [=](id<1> index) {
            int element = in[index];
            if (index % 2 == 0)
                element = element + 1000; // then-branch (line #17)
            else
                element = -1; // else-branch (line #19)
            out[index] = element;
        });
    });
}
#include <CL/sycl.hpp>
using namespace cl::sycl;

daemon compute(int input[], int output[]) {
  queue device_queue; // picks default device
  range<1> range{64};
  buffer<int, 1> buffer_in{input, range};
  buffer<int, 1> buffer_out{output, range};

device_queue.submit([&](handler& cgh) {
  auto in = buffer_in.get_access<access::mode::read>(cgh);
  auto out = buffer_out.get_access<access::mode::write>(cgh);

  cgh.parallel_for<class kernel>(range, [=](id<1> index) {
    int element = in[index];
    if (index % 2 == 0)
      element = element + 1000; // then-branch (line #17)
    else
      element = -1;          // else-branch (line #19)
    out[index] = element;
  });
});

Demo: sample kernel
```cpp
#include <CL/sycl.hpp>
using namespace cl::sycl;

void compute(int input[], int output[]) {
    queue device_queue; // picks default device
    range<1> range{64};
    buffer<int, 1> buffer_in{input, range};
    buffer<int, 1> buffer_out{output, range};

    device_queue.submit([&](handler& cgh) {
        auto in = buffer_in.get_access<access::mode::read>(cgh);
        auto out = buffer_out.get_access<access::mode::write>(cgh);

        cgh.parallel_for<class kernel>(range, [=](id<1> index) {
            int element = in[index];
            if (index % 2 == 0)
                element = element + 1000; // then-branch (line #17)
            else
                element = -1; // else-branch (line #19)
            out[index] = element;
        });
    });
}
```cpp
#include <CL/sycl.hpp>
using namespace cl::sycl;

void compute(int input[], int output[]) {
    queue device_queue; // picks default device
    range<1> range{64};
    buffer<int, 1> buffer_in{input, range};
    buffer<int, 1> buffer_out{output, range};

    device_queue.submit([&](handler& cgh) {
        auto in = buffer_in.get_access<access::mode::read>(cgh);
        auto out = buffer_out.get_access<access::mode::write>(cgh);

        cgh.parallel_for<class kernel>(range, [=](id<1> index) {
            int element = in[index];
            if (index % 2 == 0)
                element = element + 1000; // then-branch (line #17)
            else
                element = -1; // else-branch (line #19)
            out[index] = element;
        });
    });
}
```

Demo: sample kernel
int main() {
    int input[64];
    int output[64];

    // Initialize the input
    for (unsigned int i = 0; i < 64; i++)
        input[i] = i + 100;

    compute(input, output);

    return 0; // end of main (line #35)
}
Challenges

✓ Implicit Pass-By-Reference Arguments
✓ C++ Functions with Template Parameters

✓ Thread View with SIMD Lanes
  o Modeling Device Threads
  o Conditional Breakpoints
C++ challenges: pass-by-value arguments

- GDB was not able to detect implicit pass-by-reference arguments

In cl::sycl::accessor class:

```cpp
dataT &operator[](id<dimensions> index) const;
dataT operator[](id<dimensions> index) const;
```

(gdb) print in[index]

✓ We presented the solution and fixed the function call mechanism in GDB
✓ We requested an addendum to the OpenCL Debug Information Spec: FlagTypePassByValue and FlagTypePassByReference
C++ challenges: functions with template parameters

- SYCL specification relies on templated C++ classes and functions
- If not used in the source code, the compiler does not emit the symbol
- Debugger does not know how templates are instantiated

Debugger cannot infer to which instance refers the expression f(5):

\[ f<int>(5) \text{ or } f<char>(5) \]

✓ Our mitigation: Xmethods for critical SYCL operators
✓ That allows us to simulate some inferior calls on GPU, where all functions are inlined, and inferior calls are not supported

(gdb) print index + 5
Could not find operator+.
GDB Python API: Xmethods feature

Additional methods or replacements for existing methods of a C++ class. Useful when a method is unavailable to GDB (e.g. optimized or inlined).

[Xmethods In-Python.html](https://sourceware.org/gdb/onlinedocs/gdb/Xmethods-In-Python.html)

**Xmethod matcher**
- Checks the class: `cl::sycl::accessor<.+>`
- Checks method name: `operator[]`

**Xmethod worker**
- GDB performs overload resolution among workers based on their argument lists

**Xmethod worker**
- GDB looks for the best match among available C++ methods

**best C++ method**
- GDB compares the best Xmethod with the best C++ method. When a tie – Xmethod wins.

Final best method
GPU challenges: threads view with SIMD lanes

- SYCL programs are written with a focus to a single data element
- An Intel GT thread processes several work items at once (Single Instruction Multiple Data)

**Problem**: provide a user with means to debug a single SIMD lane

✔ We extended GDB to support SIMD debugging:
  - Added a current lane field to the thread representation
  - `info threads`, `thread`, `thread apply`, `break`, `commands`
  - Condition of a breakpoint is checked for all enabled SIMD lanes
Overview of SIMD lanes support

- We display only enabled SIMD lanes
- 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
- IGC support required

Other names and brands may be claimed as the property of others.
Refer to software.intel.com/articles/optimization-notice for more information regarding performance & optimization choices in Intel software products.
GPU unresolved challenges: modelling threads

- HW dispatches available threads to active kernels:
  - HW thread can switch between different kernels or become idle

Current state

- The debugger knows only about threads that report an event
- We stop a kernel at initial breakpoint (BP) to place user-defined BPs

Different model under evaluation (minimize intrusion)

- New thread state: unavailable
- Place BPs when kernel gets loaded
- No thread entry/exit events
GPU unresolved challenges: conditional BPs

(gdb) break source.cpp:35 if id == 5

Scalability concerns: handshaking between GDB, gdbserver, and debug interface

Goal: move condition evaluation closer to device

- Evaluate the condition in gdbserver-gt
- Generate device code for the condition evaluation and inject to the kernel code
- For a specific class of conditions, we can evaluate the condition in the system routine

E.g. breakpoint on a specific work item
Summary

- Debug offloaded kernels on Linux* and Windows*, for GPU, CPU, FPGA (emu)
- Thread SIMD view
- GDB*
  - Fixed C++ function calls with call-by-value parameters
  - Used XMethods to replace calls to known SYCL template functions
- Scalability is expected to be a major challenge

Thank you!
Notices & Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SY斯mark 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. For more complete information visit www.intel.com/benchmarks.

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Copyright ©, Intel Corporation. All rights reserved. Intel, the Intel logo, Pentium, Xeon, Core, VTune, OpenVINO, Cilk, are trademarks of Intel Corporation or its subsidiaries in the U.S. and other countries.

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

Refer to software.intel.com/articles/optimization-notice for more information regarding performance & optimization choices in Intel software products.
Copyright ©, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.