



Target NVIDIA and AMD with oneAPI and SYCL

Rafal Bielski – Codeplay Software

14 September 2023



Established 2002 in
Edinburgh, Scotland.

Grown successfully to around
100 employees.

In 2022, we became a **wholly
owned subsidiary** of Intel.



Committed to expanding the
open ecosystem for
heterogeneous computing.

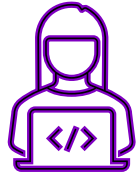
Through our involvement in
oneAPI and SYCL
governance, we help to
maintain and develop open
standards.



Developing at the forefront
of **cutting-edge research.**

Currently involved in two
research projects - **SYCLOPS**
and **AERO**, both funded by
the Horizon Europe Project.

Bringing oneAPI to NVIDIA and AMD GPUs



Write once.
Run anywhere.

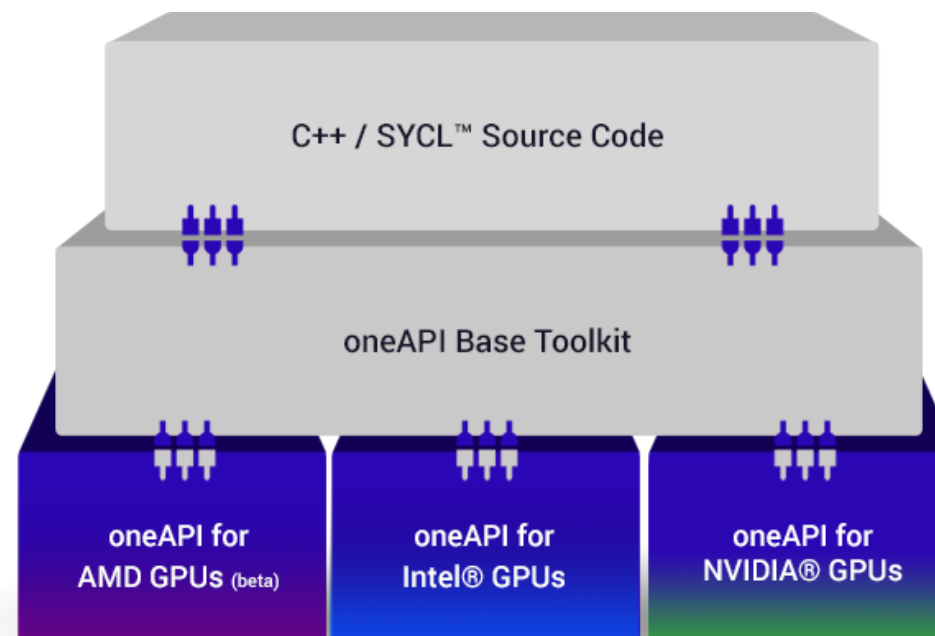
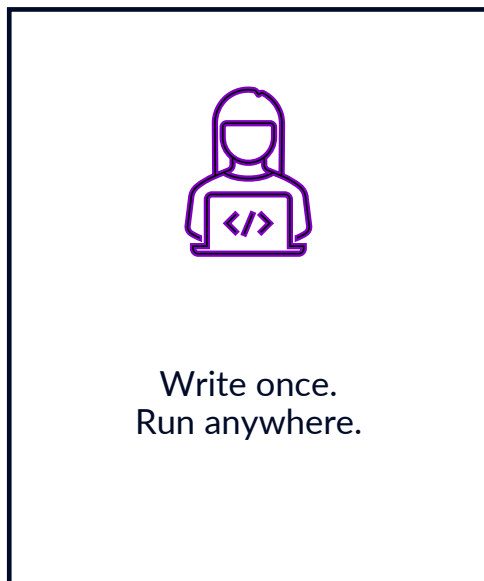


No compromises on
performance.



Open, cross-industry
collaboration on standards.

Codeplay provides oneAPI plugins enabling SYCL to run on NVIDIA and AMD GPUs

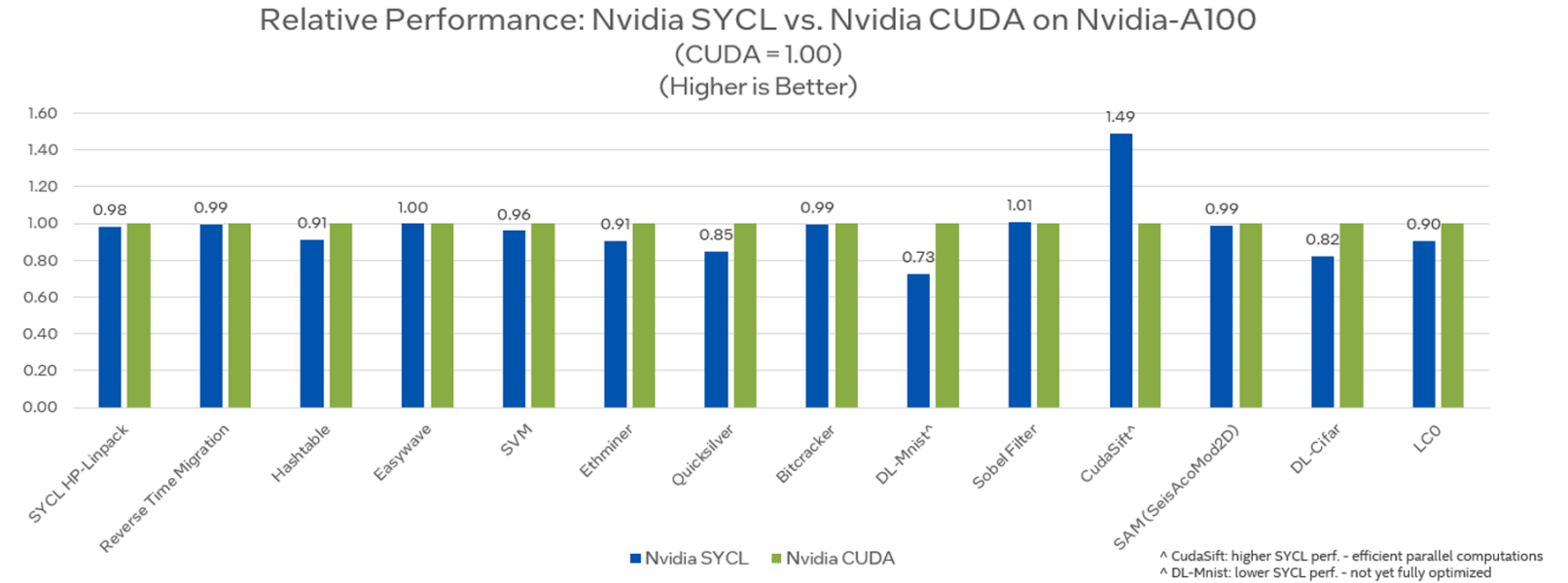


Write code using SYCL, and then run freely across Intel, NVIDIA and AMD GPUs



No compromises on performance.

On NVIDIA GPU – SYCL Provides Comparable Performance to CUDA



Testing Date: Performance results are based on testing by Intel as of April 15, 2023 and may not reflect all publicly available updates.

Configuration Details and Workload Setup: Intel® Xeon® Platinum 8360Y CPU @ 2.4GHz, 2 socket, Hyper Thread On, Turbo On, 256GB Hynix DDR4-3200, ucode 0xd000363. GPU: Nvidia A100 PCIe 80GB GPU memory. Software: SYCL open source/CLANG 17.0.0, CUDA SDK 12.0 with NVIDIA-NVCC 12.0.76, cuMath 12.0, cuDNN 12.0, Ubuntu 22.04.1. SYCL open source/CLANG compiler switches: -fscycl-targets=rvptx64-nvidia-cuda, NVIDIA NVCC compiler switches: -O3 -gencode arch=compute_80, code=sm_80. Represented workloads with Intel optimizations.

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.

Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex. Your costs and results may vary.

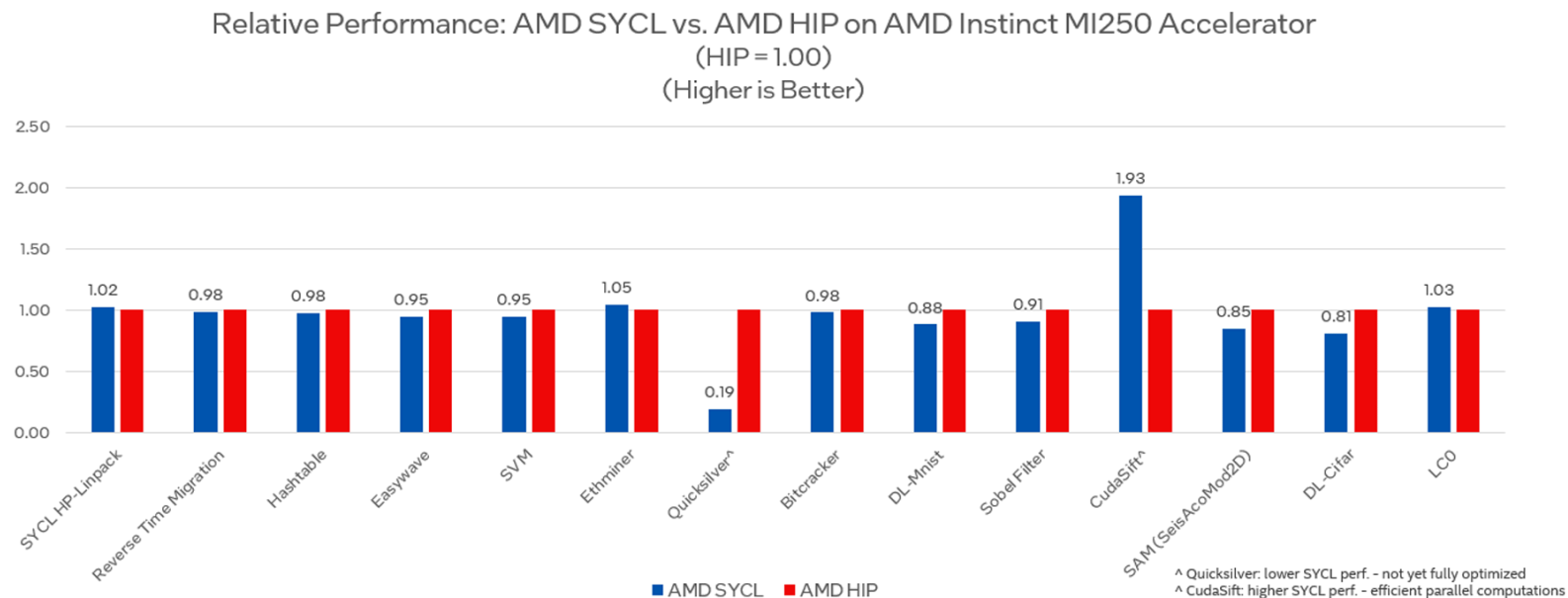
SYCL performance on NVIDIA/AMD GPUs matches native CUDA/HIP for diverse workloads

See [our blog post](#) for more details on these benchmark results



No compromises on performance.

On AMD GPU – SYCL Provides Comparable Performance to HIP



Testing Date: Performance results are based on testing by Intel as of April 15, 2023 and may not reflect all publicly available updates.

Configuration Details and Workload Setup: AMD EPYC 7313 CPU @ 3.0GHz, 2 socket, AMD Simultaneous Multi-Threading Off, AMD Precision Boost Enabled, 512GB DDR4, ucode 0xa001144. GPU: AMD Instinct MI250 OAM, 128GB GPU memory. Software: SYCL open source/CLANG 17.0.0, AMD RoCm 5.3.0 with roc-5.3.0 22362, hipSolver 5.3.0, rocBLAS 5.3.0, Ubuntu 20.04.4, SYCL open source/CLANG compiler switches: -O3 -fsycl -fsycl-targets=amdgc-n-amd-amdhsa -Xsycl-target-backend=offload-arch=gfx90a, AMD-ROCm compiler switches: -O3. Represented workloads with Intel optimizations.

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.

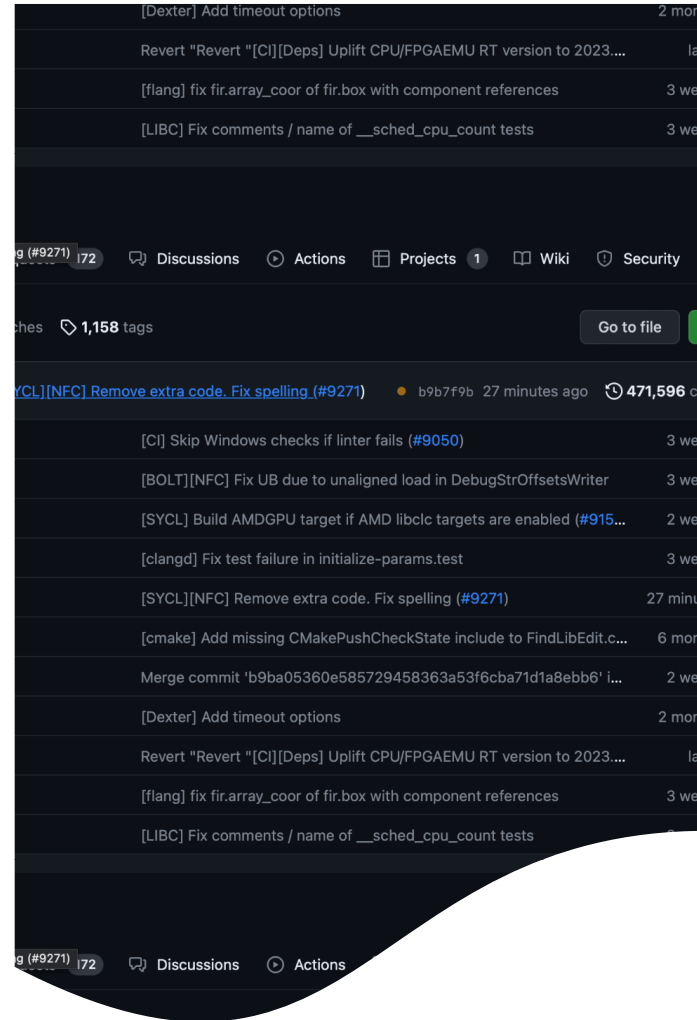
Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex. Your costs and results may vary.

SYCL performance on NVIDIA/AMD GPUs matches native CUDA/HIP for diverse workloads

See [our blog post](#) for more details on these benchmark results



Open, cross-industry
collaboration on standards.



The code is entirely open source

Available as a free plugin download
at developer.codeplay.com
for selected platforms

Available to build from source
for other platforms

Getting started – live demo

Get started



Live demo: installing the plugins

- Using the following setup:
 - A machine with an NVIDIA GPU and an Intel iGPU
 - Another machine with an AMD GPU
 - Both with Ubuntu 22.04, [Intel oneAPI Base Toolkit 2023.2.1](#), drivers and toolkits (CUDA/ROCm) for the GPUs installed
- We'll show:
 - Checking the available GPUs with `lshw -c video`
 - Checking GPU and driver details with `nvidia-smi / rocm-smi`
 - Using `sycl-ls` to find supported SYCL backends
 - Installing [NVIDIA](#) and [AMD](#) plugins for oneAPI

Compiling

Basic compilation:

```
icpx -fsycl -fsycl-targets=nvptx64-nvidia-cuda sycl-app.cpp -o sycl-app
```

Use the SYCL compiler

Compile for NVIDIA

The source file

The binary

Run the app with optional environment variables switching SYCL runtime options:

```
ONEAPI_DEVICE_SELECTOR=cuda:gpu SYCL_PI_TRACE=1 ./sycl-app
```

Filter devices to only use NVIDIA GPU

Set runtime verbosity level

Documentation of the runtime environment variables:

<https://intel.github.io/llvm-docs/EnvironmentVariables.html>

Compiling

- Some features are not yet fully supported by the `icpx` compiler driver
- See developer.codeplay.com/products/oneapi/nvidia/2023.2.0/guides/troubleshooting
- If you wish to use them, it is possible to use the `clang++` driver
- Set up the environment with:

```
source /opt/intel/oneapi/setvars.sh --include-intel-llvm
```
- Use `clang++` in the same way as `icpx` – the aforementioned CLI options are identical between the two
- `icpx` provides Intel's extra proprietary optimisations

Live demo: compiling for a single target

- Example code:

- github.com/rafbiels/HeCBench/tree/nbody-sycl-demo/nbody-sycl
- See upstream repo for more details: github.com/zjin-lcf/HeCBench

- Compile for NVIDIA GPU:

```
icpx -fsycl -fsycl-targets=nvptx64-nvidia-cuda \  
-Xsycl-target-backend --offload-arch=sm_86 \  
-O3 -o main main.cpp GSimulation.cpp
```

- Compile for AMD GPU:

```
icpx -fsycl -fsycl-targets=amdgcN-amd-amdhsa \  
-Xsycl-target-backend --offload-arch=gfx1010 \  
-O3 -o main main.cpp GSimulation.cpp
```

- Run:

```
SYCL_PI_TRACE=1 ./main
```

Optionally print debug info
about device selection

1

Enable SYCL compilation

2

Set the SYCL target to either `nvptx64-nvidia-cuda`
or `amdgcN-amd-amdhsa`

3

Optionally set the GPU architecture to make the most out of
your hardware. See these pages for the right values:

- <https://developer.nvidia.com/cuda-gpus>
- <https://llvm.org/docs/AMDGPUUsage.html>

4

Add any other flags, e.g. the optimisation level

Multi-Target Compilation

- Use a comma-separated list of targets for `-fsycl-targets`
- Specify a target for the backend options with `-Xsycl-target-backend=<target>`

```
icpx -fsycl -fsycl-targets=amdgcN-AMD-amdhsa,nvptx64-nvidia-cuda,spir64 \  
-Xsycl-target-backend=amdgcN-AMD-amdhsa --offload-arch=gfx1010 \  
-Xsycl-target-backend=nvptx64-nvidia-cuda --offload-arch=sm_86 \  
-o sycl-app sycl-app.cpp
```

Compile for AMD

Compile for NVIDIA

Compile for Intel

Set NVIDIA architecture

Set AMD architecture

Multi-Target Compilation

Executing the same binary on different target hardware

Tells the runtime
to use NVIDIA
GPU

```
ONEAPI_DEVICE_SELECTOR=cuda:gpu SYCL_PI_TRACE=1 ./sycl-app
```

```
ONEAPI_DEVICE_SELECTOR=hip:gpu SYCL_PI_TRACE=1 ./sycl-app
```

Tells the runtime
to use AMD GPU

Select Intel GPU with Level Zero or OpenCL backend:

```
ONEAPI_DEVICE_SELECTOR=level_zero:gpu SYCL_PI_TRACE=1 ./sycl-app
```

```
ONEAPI_DEVICE_SELECTOR=opencl:gpu SYCL_PI_TRACE=1 ./sycl-app
```

This can also be
done through
device selectors
in code

Live demo: multi-target compilation

- Compile for three targets:

```
icpx -fsycl -fsycl-targets=amdgcN-amd-amdhsa,nvptx64-nvidia-cuda,spir64 \  
-Xsycl-target-backend=amdgcN-amd-amdhsa --offload-arch=gfx1010 \  
-Xsycl-target-backend=nvptx64-nvidia-cuda --offload-arch=sm_86 \  
-O3 -o main main.cpp GSimulation.cpp
```

- `objdump -h` shows the binary contains an offload section for each target
- The default selector chooses an available backend
- Choose a specific backend with `ONEAPI_DEVICE_SELECTOR` or with custom selectors in code

Debugging

- Use standard tooling for debugging
- Vendor-specific gdb extensions facilitate debugging device code: **cuda-gdb** for NVIDIA GPU and **gdb-oneapi** for Intel GPU
- rocgdb for AMD GPU currently not supported, discussion ongoing on upstreaming AMD support for device debug information to llvm
- GPU debuggers can be integrated with your favourite IDE just like the regular gdb or lldb

```
[Switching focus to CUDA kernel 1, grid 4, block (5,0,0), thread (32,0,0), device 0, sm 10, warp 0, lane 0]

Thread 1 "main" hit Breakpoint 1, main::{lambda(sycl::_V1::handler&)#5}::operator()(sycl::_V1::handler& const
::{lambda(sycl::_V1::nd_item<1>)#1}::operator()(sycl::_V1::nd_item<1>) const (this=0x7fffa3fffb68, item=...) a
t main.cpp:115
115     float v = sycl::log(1+sycl::exp(-1*A_y_label[i]*xp));
(cuda-gdb) info cuda kernels
Kernel Parent Dev Grid Status          SMS Mask  GridDim  BlockDim  Invocation
*      1      -   0   4 Active 0x00000000000000000555555555555555 (24,1,1) (256,1,1) typeinfo name for main::{lam
bda(sycl::_V1::handler&)#5}::operator()(sycl::_V1::handler& const)::compute()
(cuda-gdb) list
110     for( int j = A_row_ptr[i]; j < A_row_ptr[i+1]; ++j){
111         xp += A_value[j] * x[A_col_index[j]];
112     }
113
114     // compute objective
115     float v = sycl::log(1+sycl::exp(-1*A_y_label[i]*xp));
116     auto atomic_obj_ref = atomic_ref<float,
117         memory_order::relaxed, memory_scope::device,
118         access::address_space::global_space> (total_obj_val[0]);
119     atomic_obj_ref.fetch_add(v);
(cuda-gdb) print i
$1 = 1312
(cuda-gdb) print xp
$2 = 0.0494509786
(cuda-gdb) next
118     access::address_space::global_space> (total_obj_val[0]);
(cuda-gdb) print v
$3 = 0.668727338
(cuda-gdb) continue
Continuing.
[Switching focus to CUDA kernel 1, grid 4, block (0,0,0), thread (0,0,0), device 0, sm 0, warp 3, lane 0]

Thread 1 "main" hit Breakpoint 1, main::{lambda(sycl::_V1::handler&)#5}::operator()(sycl::_V1::handler& const
::{lambda(sycl::_V1::nd_item<1>)#1}::operator()(sycl::_V1::nd_item<1>) const (this=0x7fffa3fffb68, item=...) a
t main.cpp:115
115     float v = sycl::log(1+sycl::exp(-1*A_y_label[i]*xp));
(cuda-gdb) print xp
$4 = 0.0241964087
(cuda-gdb) □
```

Debugging

- `cuda-gdb` might have some issues with certain calls generated by LLVM
- If you run into errors while debugging SYCL code, try starting `cuda-gdb` with `CUDBG_USE_LEGACY_DEBUGGER=1`

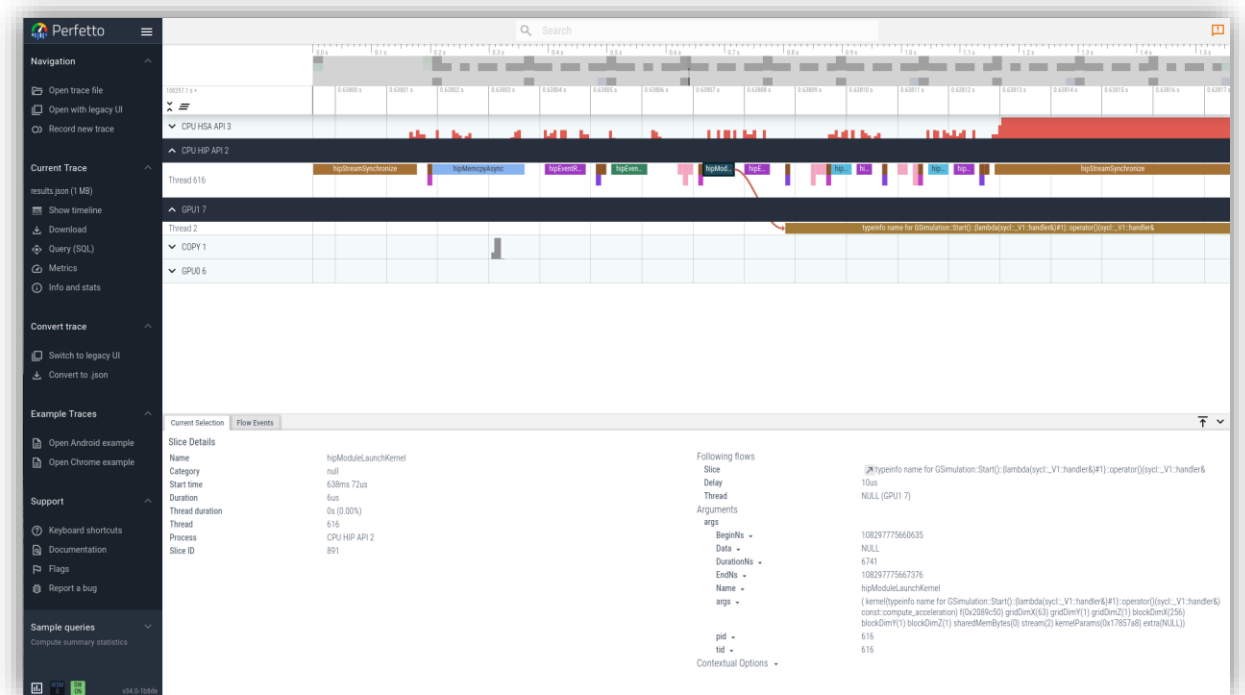
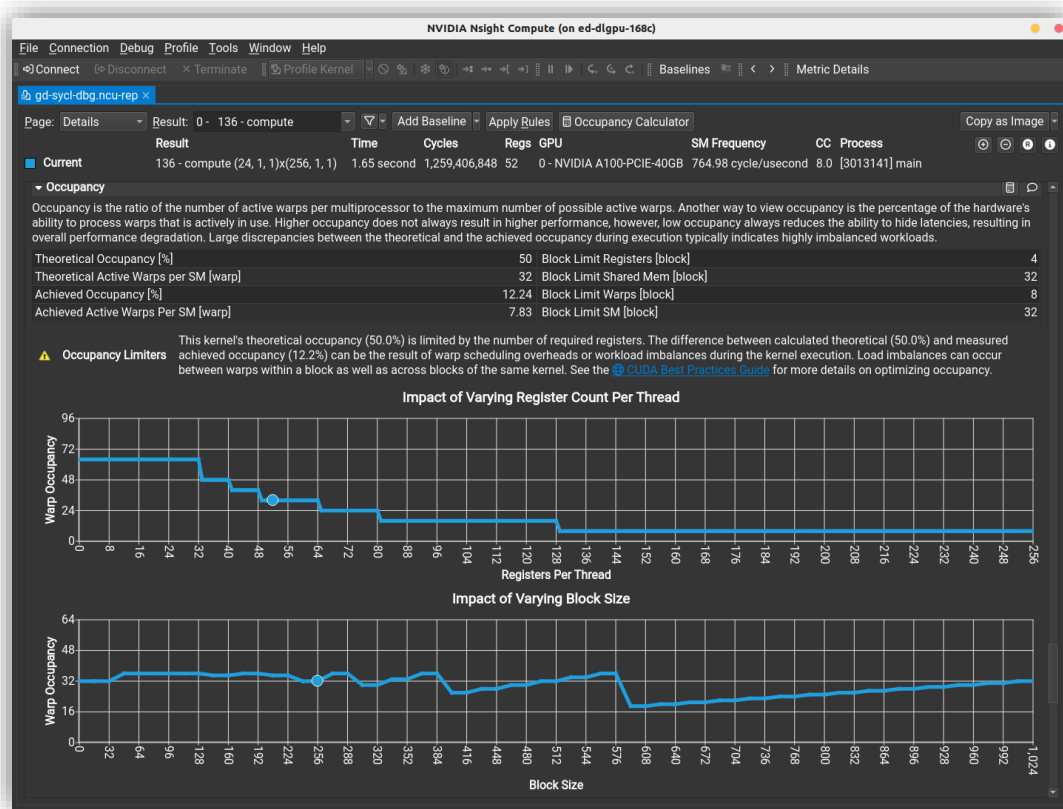
Live demo: debugging

- Compile for the NVIDIA target with `-O0` and `-g`
- Launch the program in `cuda-gdb`
- List code, set a breakpoint, run the program
- Investigate the device state at breakpoint
- Print kernel stack variables and step through instructions

Profiling

Use NVIDIA NSight Systems and NSight Compute to profile SYCL code

Use AMD ROCProfiler to profile SYCL code




Live demo: profiling

- Record standard NSight Systems profile: `nsys profile ./main`
- Analyse the profile in the GUI: `nsys-ui report1.nsys-rep`
- Record a selection of NSight Compute metrics:
`ncu --section ComputeworkloadAnalysis --section LaunchStats
--section MemoryWorkloadAnalysis --section Occupancy
--section SchedulerStats --section SpeedOfLight
--section SpeedOfLight_RooflineChart -o ncu-profile ./main 16000 1`
- Analyse the profile in the GUI: `ncu-ui ncu-profile.ncu-rep`
- Record ROCProfiler metrics: `rocprof --sys-trace ./main`
- Print the output CSV files, analyse the trace (results.json) in the web UI at <https://ui.perfetto.dev>

Common Optimisations

- Choose an optimal work group size
 - Read more in our [blog post](#)
- Follow recommended indexing of multi-dimensional arrays
 - In SYCL the right-most index should vary the fastest
- Consider forcing the inlining for some functions
- Apply loop unrolling
- Avoid code path divergence between threads
- Ensure coalesced memory access



Our website contains
guides for performance
developer.codeplay.com

Live demo: common optimisations

- Example project:
 - github.com/rafbiels/sycl-crowd-simulation/tree/demo-optimisation
 - Crowd simulation with live display, but comes also with a headless `-DPROFILING_MODE=ON`
- Check for optimisation potential in NSight Systems
- Analyse detailed metrics for the "hot" kernel in NSight Compute
- Showcase an optimisation reducing branching and thread divergence

Support for our plugins



Enterprise Support (currently NVIDIA only)

Our highest level of support,
for large teams.

Direct access to Codeplay's engineers
and expertise via scheduled calls.

A custom support plan tailored
to your requirements.



Priority Support (currently NVIDIA only)

Suited to small teams and
individuals.

Access to a ticketed support desk.

Accelerated response time for
questions and requests.



Forum Support (NVIDIA and AMD)

A public forum moderated by
Codeplay engineers.

Available for free.

Engage with the oneAPI community
and our engineers.

<https://codeplay.com/company/contact/>

<https://support.codeplay.com>

Summary

- Using open standard oneAPI and SYCL brings you choice
- Possible to achieve performance portability with oneAPI and SYCL
- You can use Codeplay's plugins to target NVIDIA and AMD GPUs today
- Possible to debug and profile SYCL code with NVIDIA and AMD tools



oneAPI Plugins for NVIDIA/AMD

Scan QR code or visit developer.codeplay.com





Social Media

Don't forget to follow us for the latest updates!



@codeplaysoft



@codeplaysoftware



codeplay-software



codeplay-software



Disclaimers

A wee bit of legal

Performance varies by use, configuration and other factors.

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.

© Codeplay Software Ltd.. Codeplay, Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.