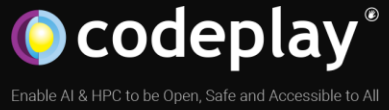


Targeting multiple accelerator architectures with oneAPI and SYCL

Rafal Bielski – Codeplay Software (ex ATLAS TDAQ)

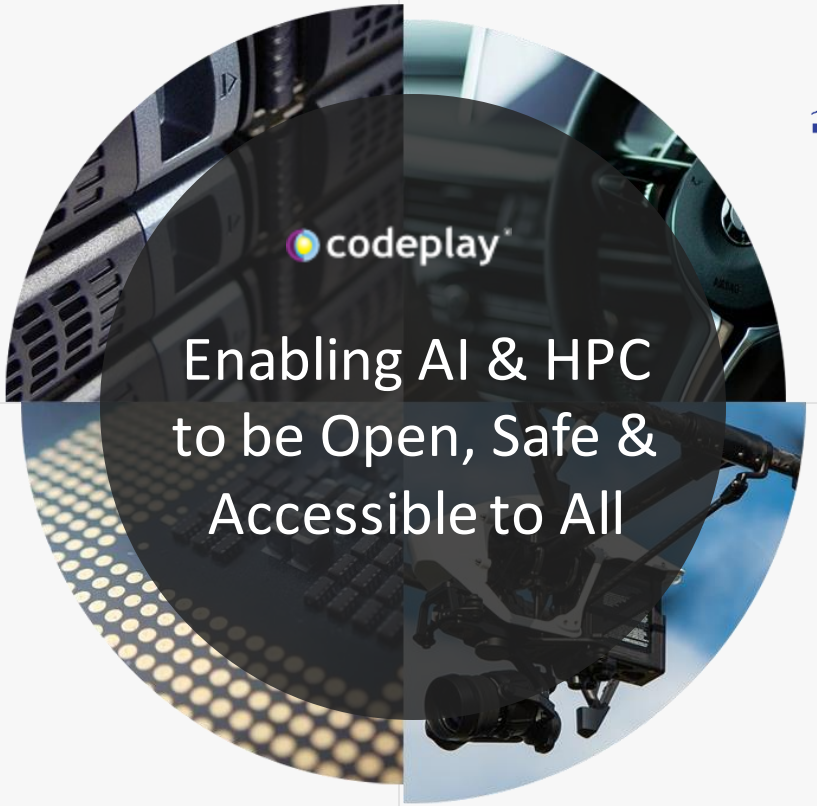
The 2023 international workshop on the Circular Electron Positron Collider, 3–6 July 2023, Edinburgh

Company

Leaders in enabling high-performance software solutions for new AI processing systems

Enabling the toughest processors with tools and middleware based on open standards

Established 2002 in Scotland, acquired by Intel in 2022 and now ~90 employees



Collaborations

SYNOPSYS
BROADCOM
CEVA
Imagination
RENESAS
KMC Kyoto Microcomputer Co., Ltd.
NSI-TEXE
BERKELEY LAB
OAK RIDGE National Laboratory
Argonne NATIONAL LABORATORY

And many more!

Supported Solutions

oneAPI

An open, cross-industry, SYCL based, unified, multiarchitecture, multi-vendor programming model that delivers a common developer experience across accelerator architectures

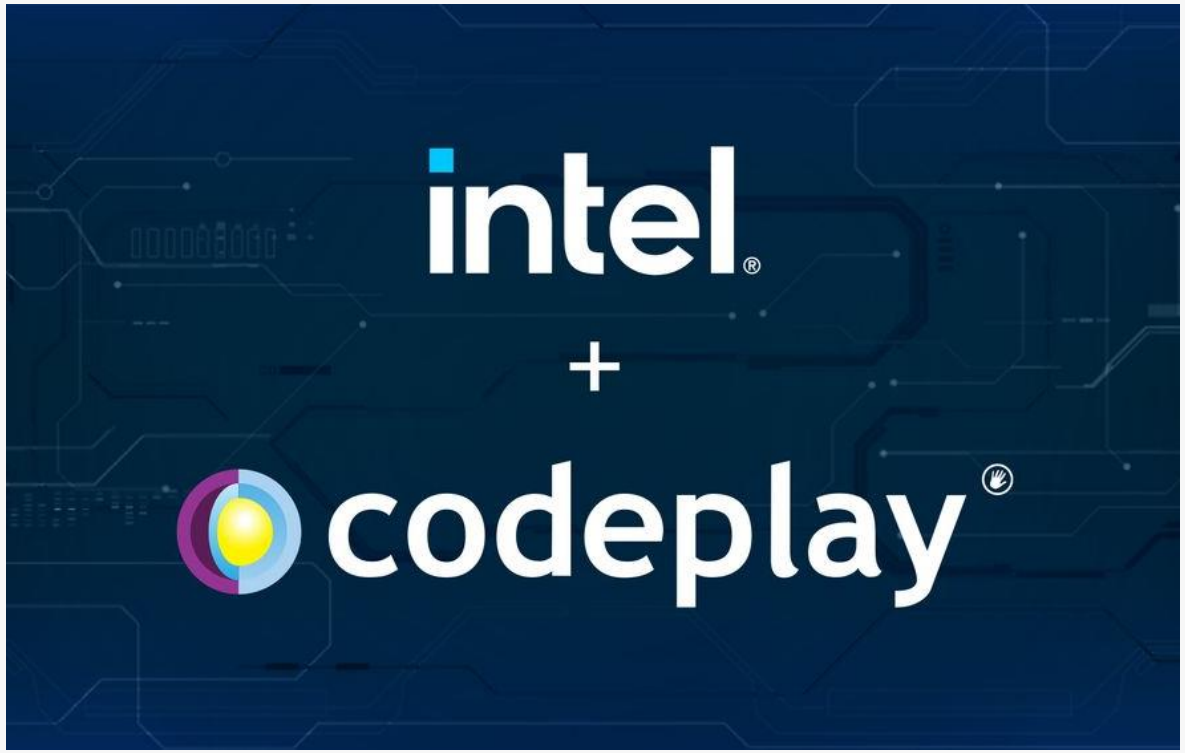
Markets

High Performance Compute (HPC)
Automotive ADAS, IoT, Cloud Compute
Smartphones & Tablets
Medical & Industrial

Technologies: Artificial Intelligence
Vision Processing
Machine Learning
Big Data Compute

Who we are

- Codeplay is a wholly owned subsidiary of Intel
- Focus on advancing and embracing SYCL and oneAPI



Outline

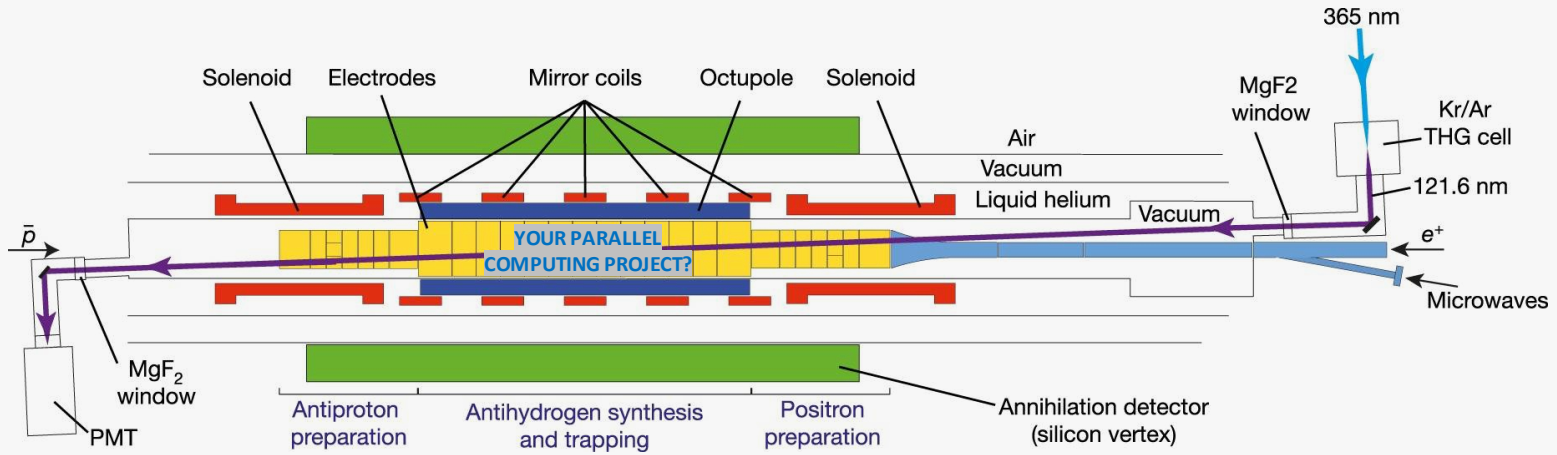
1. **Open standards** bring freedom and choice
2. One **SYCL** source to target them (accelerators) all
3. How SYCL addresses your **parallel programming** needs
4. **Performance portability** across all GPU vendors
5. How to use SYCL, oneAPI and **Codeplay NVIDIA/AMD plugins**
6. Example **successful projects** using SYCL

Open Standards

Locking into proprietary software and hardware stack may **trap** your project and impede your ability to:

- Choose the best hardware, **regardless of vendor**
- **Negotiate the best prices** for hardware

The remedy for lock-in is to **use products that conform to free, open standards**



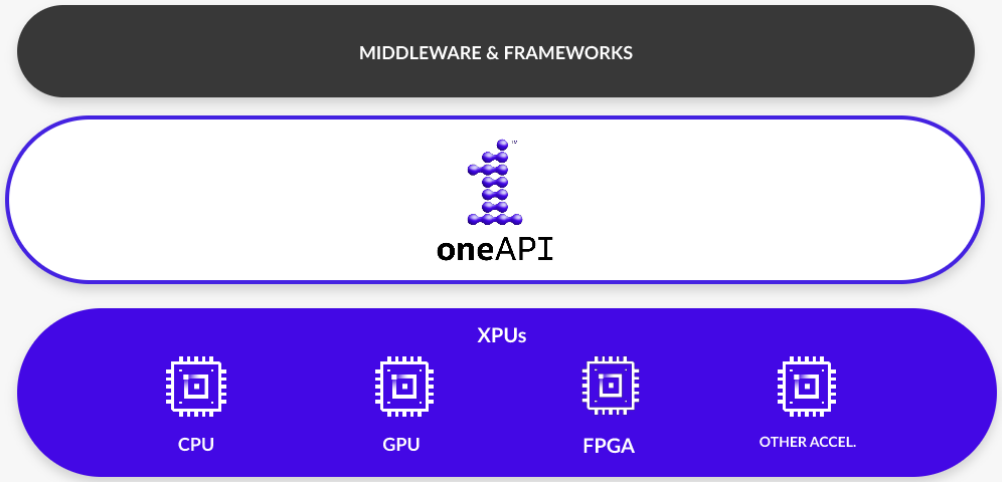
No need to create your project in a trap!

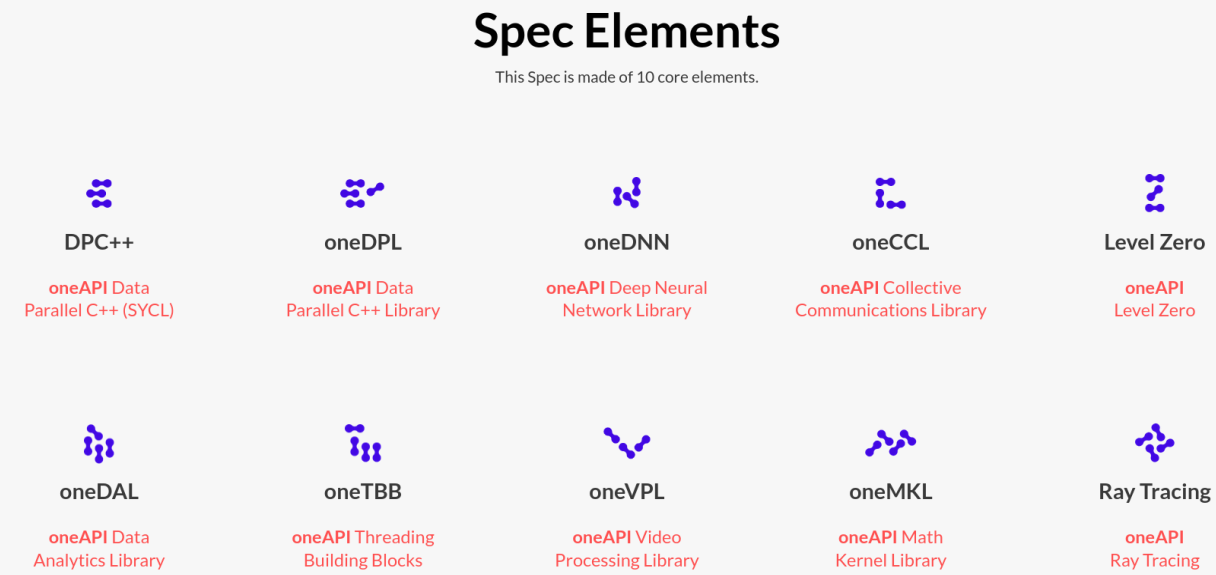
Image: ALPHA-2 central apparatus (**antihydrogen trap**) from Nature 578, 375–380 (2020)

Open Standards: oneAPI

oneAPI is a multi-architecture accelerator programming model joining together several open standards, with SYCL at its centre

- **Open:** join the community / special interest groups!
- **Multi-architecture:** target CPU, GPU, FPGA, specialized chips
- **Multi-vendor:** implementations allow the same code to target hardware from different vendors



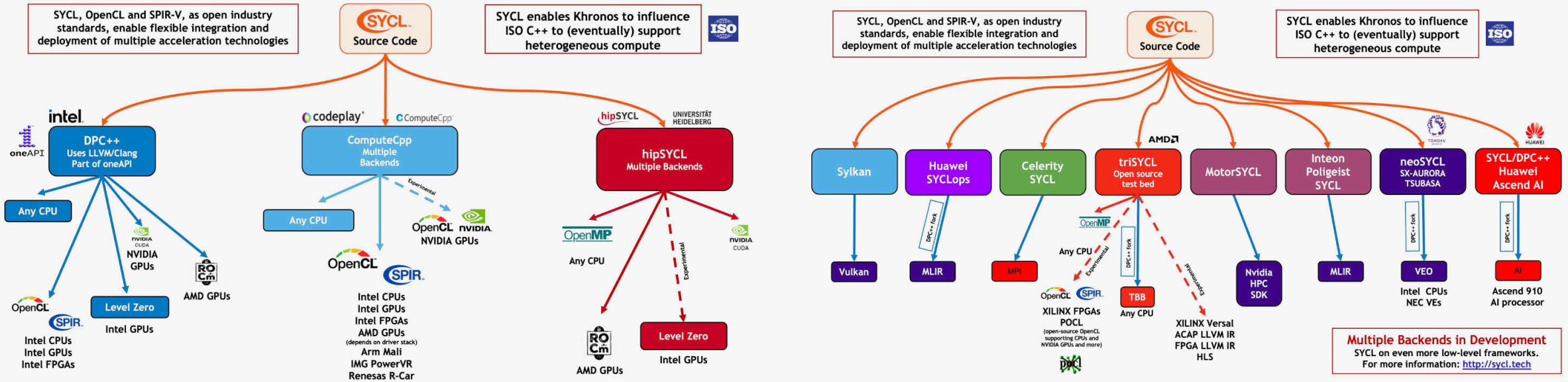


Open Standards: SYCL

- SYCL is a single-source, high-level, standard C++ programming model, that can target a range of heterogeneous platforms
- Open standard provided by the non-profit cross-industry **Khronos Group**
- Well-defined **concurrency and memory models** enable more optimisation and performance opportunities

Open Standards: SYCL

- **Multiple implementations** of SYCL exist with different target specialisations
- The same SYCL code can be compiled with any of them and provide **comparable performance** to other SYCL implementations and **to native APIs**
- Focusing today on the **DPC++** implementation (**open-source** fork of llvm) which is at the heart of the **Intel oneAPI** toolkits



Single C++ source for all architectures

```
1  #include <sycl/sycl.hpp>
2  #include <vector>
3
4  int main() {
5      constexpr static size_t N{10000};
6      std::vector<float> a(N, 1.0f);
7      std::vector<float> b(N, 2.0f);
8      std::vector<float> c(N, 0.0f);
9
10     sycl::queue q{};
11     {
12         sycl::buffer buf_a{a};
13         sycl::buffer buf_b{b};
14         sycl::buffer buf_c{c};
15         q.submit([&](sycl::handler& h){
16             sycl::accessor acc_a{buf_a, h, sycl::read_write};
17             sycl::accessor acc_b{buf_b, h, sycl::read_only};
18             sycl::accessor acc_c{buf_c, h, sycl::write_only, sycl::no_init};
19             h.parallel_for<class my_kernel>(N, [=](sycl::id<1> id){
20                 acc_a[id] += acc_b[id];
21                 acc_c[id] = 2.0f * acc_a[id];
22             });
23         }).wait();
24     }
25
26     for (float x : c) {std::cout << x << " ";}
27     std::cout << std::endl;
28
29     return 0;
30 }
```

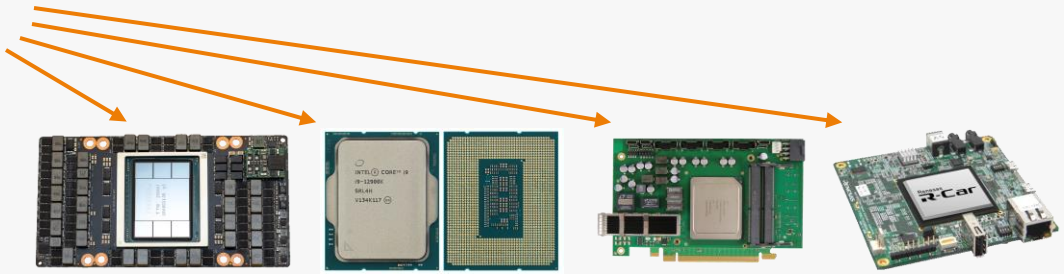
Device management with queues

Memory management with buffers

Submit a work unit to a queue

Execute device code

- Standard C++
  - SYCL 2020 based on ISO C++17
- Unlike in other parallel programming APIs, there are:
  - No pragmas or macros
  - No special attributes
  - No language extensions

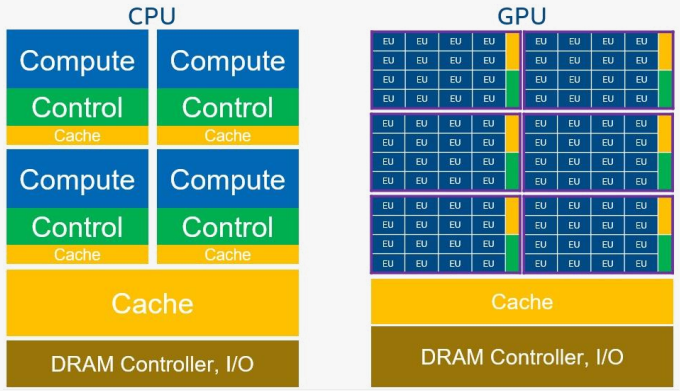


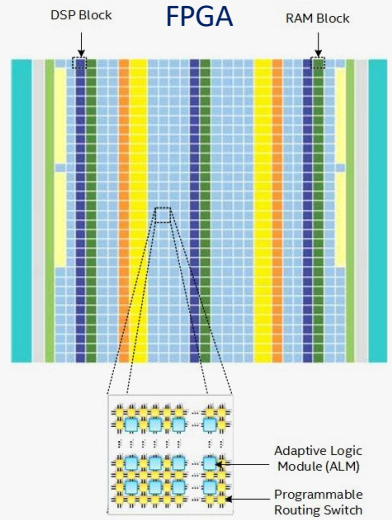
GPU CPU FPGA Specialised processors

Designing software for heterogeneous systems

While SYCL will make **writing and deploying code** on heterogeneous systems **easy for you**, it cannot change your algorithms

- CPU and SIMD processors require different **programming paradigms**
- Algorithms which are optimal for one type of processor may perform badly on another
- Truly high-performance software requires the developer to think about the **hardware architecture**
  - Memory access – e.g. efficient data structures, explicit use of local cache
  - Concurrency – e.g. avoiding thread divergence within a SIMD work group
- SYCL provides convenient interface to **leverage the hardware capabilities**
- Easy to dynamically dispatch different algorithms/tuning to different hardware with SYCL device management





See the online article *Compare Benefits of CPUs, GPUs, and FPGAs for Different oneAPI Compute Workloads*

Designing software for heterogeneous systems

> Bocci A, Innocente V, Kortelainen M, Pantaleo F and Rovere M (2020)
> *Heterogeneous Reconstruction of Tracks and Primary Vertices With the CMS Pixel Tracker*
> Front. Big Data 3:601728
>
> The development of a heterogeneous reconstruction faces several **fundamental challenges**:
> - the adoption of a different **programming paradigm**;
> - the experimental reconstruction framework and its **scheduling** must accommodate for heterogeneous processing;
> - the heterogeneous algorithms should achieve the same or better **physics performance and processing throughput** as their CPU siblings;
> - it must be possible to run and **validate on conventional machines**, without any dedicated resources.

Designing software for heterogeneous systems

Start your future TDAQ / reconstruction software project today with SYCL and get for free:

> Bocci A, Innocente V, Kortelainen M, Pantaleo F and Rovere M (2020)
> *Heterogeneous Reconstruction of Tracks and Primary Vertices With the CMS Pixel Tracker*
> Front. Big Data 3:601728
>
> The development of a heterogeneous reconstruction faces several **fundamental challenges**:
> - the adoption of a different **programming paradigm**;
> - the experimental reconstruction framework and its **scheduling** must accommodate for heterogeneous processing;
> - the heterogeneous algorithms should achieve the same or better **physics performance and processing throughput** as their CPU siblings;
> - it must be possible to run and **validate on conventional machines**, without any dedicated resources.

Designing software for heterogeneous systems

Start your future TDAQ / reconstruction software project today with SYCL and get for free:

- Full flexibility in **choosing hardware** vendors in the future

> Bocci A, Innocente V, Kortelainen M, Pantaleo F and Rovere M (2020)
> *Heterogeneous Reconstruction of Tracks and Primary Vertices With the CMS Pixel Tracker*
> Front. Big Data 3:601728
>
> The development of a heterogeneous reconstruction faces several **fundamental challenges**:
> - the adoption of a different **programming paradigm**;
> - the experimental reconstruction framework and its **scheduling** must accommodate for heterogeneous processing;
> - the heterogeneous algorithms should achieve the same or better **physics performance and processing throughput** as their CPU siblings;
> - it must be possible to run and **validate on conventional machines**, without any dedicated resources.

Designing software for heterogeneous systems

Start your future TDAQ / reconstruction software project today with SYCL and get for free:

- Full flexibility in **choosing hardware** vendors in the future
- Easy to get started and provides good transferable skills for your developers – writing **standard C++** software

> Bocci A, Innocente V, Kortelainen M, Pantaleo F and Rovere M (2020)
> *Heterogeneous Reconstruction of Tracks and Primary Vertices With the CMS Pixel Tracker*
> Front. Big Data 3:601728
>
> The development of a heterogeneous reconstruction faces several **fundamental challenges**:
> - the adoption of a different **programming paradigm**;
> - the experimental reconstruction framework and its **scheduling** must accommodate for heterogeneous processing;
> - the heterogeneous algorithms should achieve the same or better **physics performance and processing throughput** as their CPU siblings;
> - it must be possible to run and **validate on conventional machines**, without any dedicated resources.

Designing software for heterogeneous systems

Start your future TDAQ / reconstruction software project today with SYCL and get for free:

- Full flexibility in **choosing hardware** vendors in the future
- Easy to get started and provides good transferable skills for your developers – writing **standard C++** software
- Can share **code between the CPU and GPU** implementations

> Bocci A, Innocente V, Kortelainen M, Pantaleo F and Rovere M (2020)
> *Heterogeneous Reconstruction of Tracks and Primary Vertices With the CMS Pixel Tracker*
> Front. Big Data 3:601728
>
> The development of a heterogeneous reconstruction faces several **fundamental challenges**:
> - the adoption of a different **programming paradigm**;
> - the experimental reconstruction framework and its **scheduling** must accommodate for heterogeneous processing;
> - the heterogeneous algorithms should achieve the same or better **physics performance and processing throughput** as their CPU siblings;
> - it must be possible to run and **validate on conventional machines**, without any dedicated resources.

```
auto cpuOptimisedKernel(Params p) {
    return [p](sycl::id<1> id) {
        auto myArray = getArray(id, p.a, p.b); // same function called on CPU and GPU
        p.c[id] = std::any_of(myArray.begin(),myArray.end(),[](auto elem){return myCondition(elem);});
        // any_of early exit causes thread divergence on GPU, but works faster on CPU
    };
}

auto gpuOptimisedKernel(Params p) {
    return [p](sycl::id<1> id) {
        auto myArray = getArray(id, p.a, p.b); // same function called on CPU and GPU
        p.c[id] = false;
        for (auto elem : myArray) {p.c[id] |= myCondition(elem);}
        // redundant loop over full range works out faster on GPU than any_of on CPU
    };
}
```

```
sycl::event event = q.submit([&](sycl::handler& h){
    Params params{
        sycl::accessor{buf_a, h, sycl::read_write},
        sycl::accessor{buf_b, h, sycl::read_only},
        sycl::accessor{buf_c, h, sycl::write_only, sycl::no_init}
    };
    if (q.get_device().is_gpu() || validation) {
        h.parallel_for(N, gpuOptimisedKernel(params));
    } else if (q.get_device().is_cpu()) {
        h.parallel_for(N, cpuOptimisedKernel(params));
    }
});
// some asynchronous code here, followed by a wait on the event
event.wait_and_throw();
// sync point may be in another scope, just copy the event and wait there
myScheduler.pushEvent(event);
```

Designing software for heterogeneous systems

Start your future TDAQ / reconstruction software project today with SYCL and get for free:

- Full flexibility in **choosing hardware** vendors in the future
- Easy to get started and provides good transferable skills for your developers – writing **standard C++** software
- Can share **code between the CPU and GPU** implementations
- Portable **high performance** with **flexibility to tune** further for specific devices

> Bocci A, Innocente V, Kortelainen M, Pantaleo F and Rovere M (2020)
> *Heterogeneous Reconstruction of Tracks and Primary Vertices With the CMS Pixel Tracker*
> Front. Big Data 3:601728
>
> The development of a heterogeneous reconstruction faces several **fundamental challenges**:
> - the adoption of a different **programming paradigm**;
> - the experimental reconstruction framework and its **scheduling** must accommodate for heterogeneous processing;
> - the heterogeneous algorithms should achieve the same or better **physics performance and processing throughput** as their CPU siblings;
> - it must be possible to run and **validate on conventional machines**, without any dedicated resources.

```
auto cpuOptimisedKernel(Params p) {
    return [p](sycl::id<1> id) {
        auto myArray = getArray(id, p.a, p.b); // same function called on CPU and GPU
        p.c[id] = std::any_of(myArray.begin(),myArray.end(),[](auto elem){return myCondition(elem);});
        // any_of early exit causes thread divergence on GPU, but works faster on CPU
    };
}

auto gpuOptimisedKernel(Params p) {
    return [p](sycl::id<1> id) {
        auto myArray = getArray(id, p.a, p.b); // same function called on CPU and GPU
        p.c[id] = false;
        for (auto elem : myArray) {p.c[id] |= myCondition(elem);}
        // redundant loop over full range works out faster on GPU than any_of on CPU
    };
}
```

```
sycl::event event = q.submit([&](sycl::handler& h){
    Params params{
        sycl::accessor{buf_a, h, sycl::read_write},
        sycl::accessor{buf_b, h, sycl::read_only},
        sycl::accessor{buf_c, h, sycl::write_only, sycl::no_init}
    };
    if (q.get_device().is_gpu() || validation) {
        h.parallel_for(N, gpuOptimisedKernel(params));
    } else if (q.get_device().is_cpu()) {
        h.parallel_for(N, cpuOptimisedKernel(params));
    }
});
// some asynchronous code here, followed by a wait on the event
event.wait_and_throw();
// sync point may be in another scope, just copy the event and wait there
myScheduler.pushEvent(event);
```

Designing software for heterogeneous systems

Start your future TDAQ / reconstruction software project today with SYCL and get for free:

- Full flexibility in **choosing hardware** vendors in the future
- Easy to get started and provides good transferable skills for your developers – writing **standard C++** software
- Can share **code between the CPU and GPU** implementations
- Portable **high performance** with **flexibility to tune** further for specific devices
- Flexibility in validation – possible to run GPU kernels on CPU

> Bocci A, Innocente V, Kortelainen M, Pantaleo F and Rovere M (2020)
> *Heterogeneous Reconstruction of Tracks and Primary Vertices With the CMS Pixel Tracker*
> Front. Big Data 3:601728
>
> The development of a heterogeneous reconstruction faces several **fundamental challenges**:
> - the adoption of a different **programming paradigm**;
> - the experimental reconstruction framework and its **scheduling** must accommodate for heterogeneous processing;
> - the heterogeneous algorithms should achieve the same or better **physics performance and processing throughput** as their CPU siblings;
> - it must be possible to run and **validate on conventional machines**, without any dedicated resources.

```
auto cpuOptimisedKernel(Params p) {
    return [p](sycl::id<1> id) {
        auto myArray = getArray(id, p.a, p.b); // same function called on CPU and GPU
        p.c[id] = std::any_of(myArray.begin(),myArray.end(),[](auto elem){return myCondition(elem);});
        // any_of early exit causes thread divergence on GPU, but works faster on CPU
    };
}

auto gpuOptimisedKernel(Params p) {
    return [p](sycl::id<1> id) {
        auto myArray = getArray(id, p.a, p.b); // same function called on CPU and GPU
        p.c[id] = false;
        for (auto elem : myArray) {p.c[id] |= myCondition(elem);}
        // redundant loop over full range works out faster on GPU than any_of on CPU
    };
}
```

```
sycl::event event = q.submit([&](sycl::handler& h){
    Params params{
        sycl::accessor{buf_a, h, sycl::read_write},
        sycl::accessor{buf_b, h, sycl::read_only},
        sycl::accessor{buf_c, h, sycl::write_only, sycl::no_init}
    };
    if (q.get_device().is_gpu() || validation) {
        h.parallel_for(N, gpuOptimisedKernel(params));
    } else if (q.get_device().is_cpu()) {
        h.parallel_for(N, cpuOptimisedKernel(params));
    }
});
// some asynchronous code here, followed by a wait on the event
event.wait_and_throw();
// sync point may be in another scope, just copy the event and wait there
myScheduler.pushEvent(event);
```

Designing software for heterogeneous systems

Start your future TDAQ / reconstruction software project today with SYCL and get for free:

- Full flexibility in **choosing hardware** vendors in the future
- Easy to get started and provides good transferable skills for your developers – writing **standard C++** software
- Can share **code between the CPU and GPU** implementations
- Portable **high performance** with **flexibility to tune** further for specific devices
- Flexibility in validation – possible to run GPU kernels on CPU
- Convenient scheduling interface

> Bocci A, Innocente V, Kortelainen M, Pantaleo F and Rovere M (2020)
> *Heterogeneous Reconstruction of Tracks and Primary Vertices With the CMS Pixel Tracker*
> Front. Big Data 3:601728
>
> The development of a heterogeneous reconstruction faces several **fundamental challenges**:
> - the adoption of a different **programming paradigm**;
> - the experimental reconstruction framework and its **scheduling** must accommodate for heterogeneous processing;
> - the heterogeneous algorithms should achieve the same or better **physics performance and processing throughput** as their CPU siblings;
> - it must be possible to run and **validate on conventional machines**, without any dedicated resources.

```
auto cpuOptimisedKernel(Params p) {
    return [p](sycl::id<1> id) {
        auto myArray = getArray(id, p.a, p.b); // same function called on CPU and GPU
        p.c[id] = std::any_of(myArray.begin(),myArray.end(),[](auto elem){return myCondition(elem);});
        // any_of early exit causes thread divergence on GPU, but works faster on CPU
    };
}

auto gpuOptimisedKernel(Params p) {
    return [p](sycl::id<1> id) {
        auto myArray = getArray(id, p.a, p.b); // same function called on CPU and GPU
        p.c[id] = false;
        for (auto elem : myArray) {p.c[id] |= myCondition(elem);}
        // redundant loop over full range works out faster on GPU than any_of on CPU
    };
}
```

```
sycl::event event = q.submit([&](sycl::handler& h){
    Params params{
        sycl::accessor{buf_a, h, sycl::read_write},
        sycl::accessor{buf_b, h, sycl::read_only},
        sycl::accessor{buf_c, h, sycl::write_only, sycl::no_init}
    };
    if (q.get_device().is_gpu() || validation) {
        h.parallel_for(N, gpuOptimisedKernel(params));
    } else if (q.get_device().is_cpu()) {
        h.parallel_for(N, cpuOptimisedKernel(params));
    }
});
// some asynchronous code here, followed by a wait on the event
event.wait_and_throw();
// sync point may be in another scope, just copy the event and wait there
myScheduler.pushEvent(event);
```

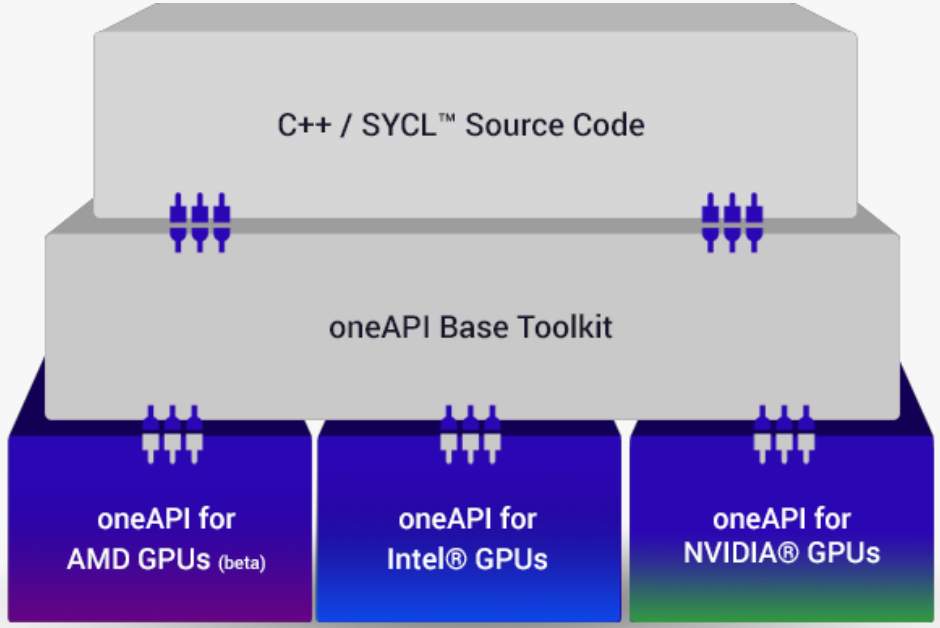
Target any GPU with oneAPI and SYCL

Codeplay's flagship product:

Plugins to the Intel oneAPI toolkit adding support for NVIDIA and AMD GPUs

Simply **download from** developer.codeplay.com, install and you're ready to compile SYCL for NVIDIA/AMD GPUs

- Currently only for Ubuntu 22.04 and compatible Linux systems
- Open source – build from source for other platforms

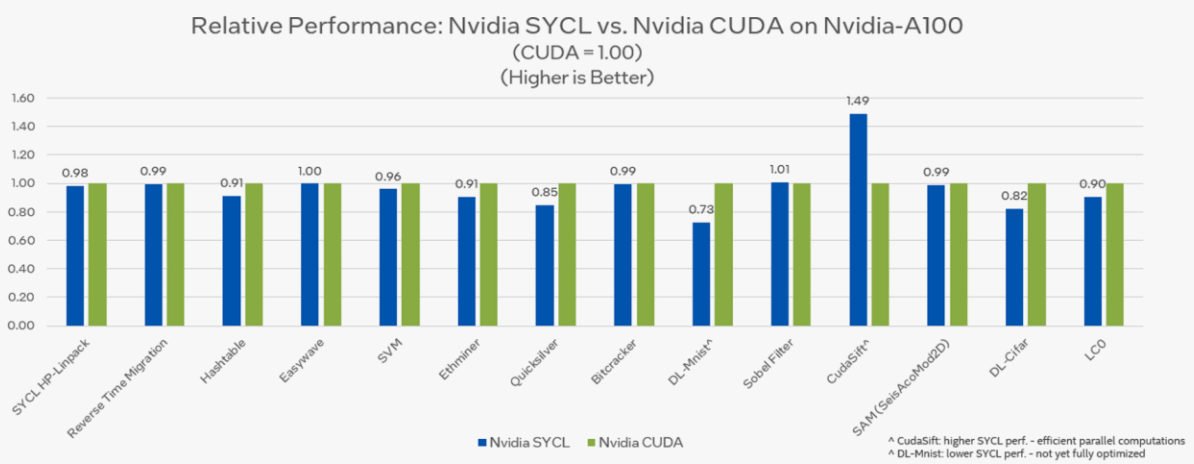


Prerequisites:

- Intel oneAPI base toolkit
- GPU vendor drivers and libraries (CUDA/ROCm)

SYCL performance matches native CUDA/HIP

On NVIDIA GPU – SYCL Provides Comparable Performance to CUDA



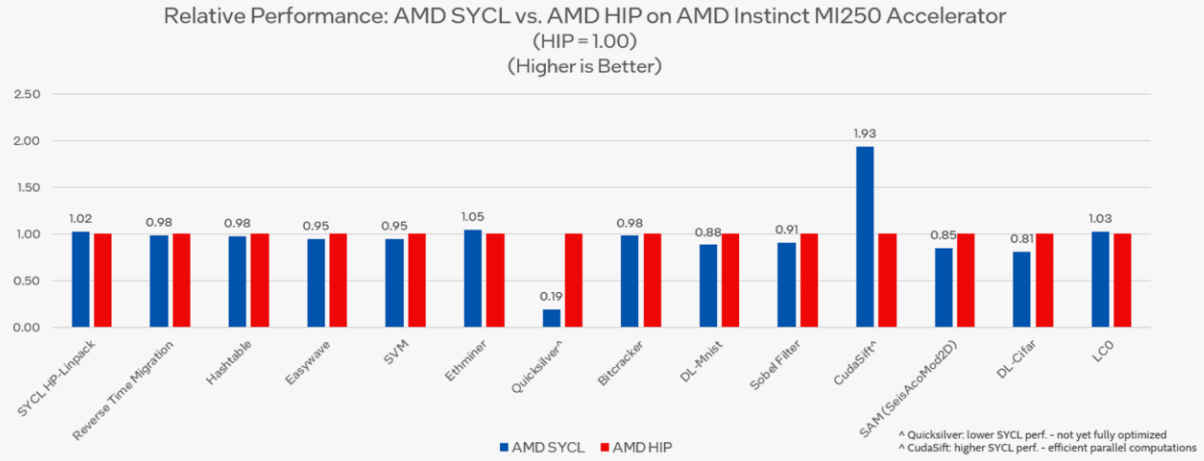
Testing Date: Performance results are **based on testing by Intel as of April 15, 2023** and may not reflect all publicly available updates.

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: -fsycl-targets=nvptx64-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.

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 0xa00144. 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=amdgcn-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.

See our blog post for more details on these benchmark results

Compile your SYCL code for multiple targets

Basic compilation:

```
clang++ -fsycl -fsycl-targets=nvptx64-nvidia-cuda –O3 sycl-app.cpp -o sycl-app
```

Use the SYCL compiler

Compile for Nvidia

Other flags

The source file

The binary

Multi-target compilation for specific archs:

Compile for AMD

Compile for NVIDIA

Compile for Intel

```
clang++ -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 sycl-app sycl-app.cpp
```

Set NVIDIA architecture

Set AMD architecture

Run your multi-target SYCL application

Executing the same binary on different target hardware

Filter devices to only 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
```

Filter devices to only use AMD GPU

Set runtime verbosity level

This can also be done through device selectors in code

Select Intel GPU with the 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
```

See the documentation of the environment variables

Run on any x86_64 CPU with the OpenCL backend:

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

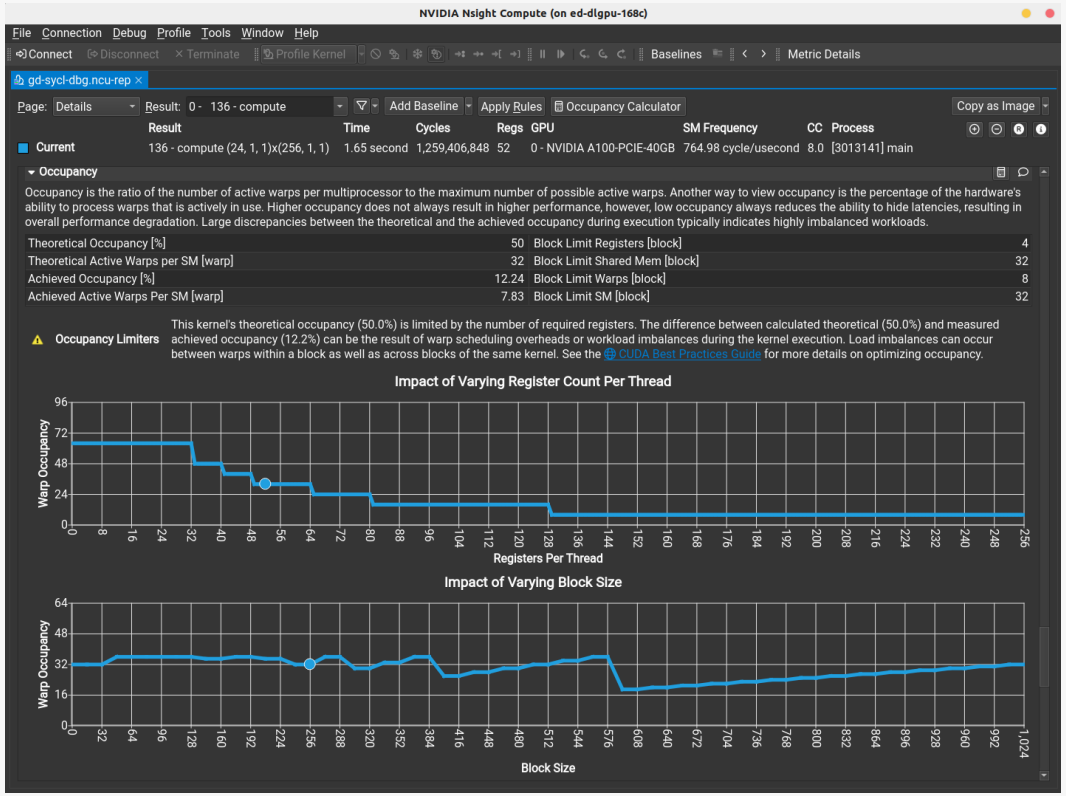
Debugging

- 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
- Debuggers can be integrated with your favourite IDE just like the regular **gdb** or **lldb**

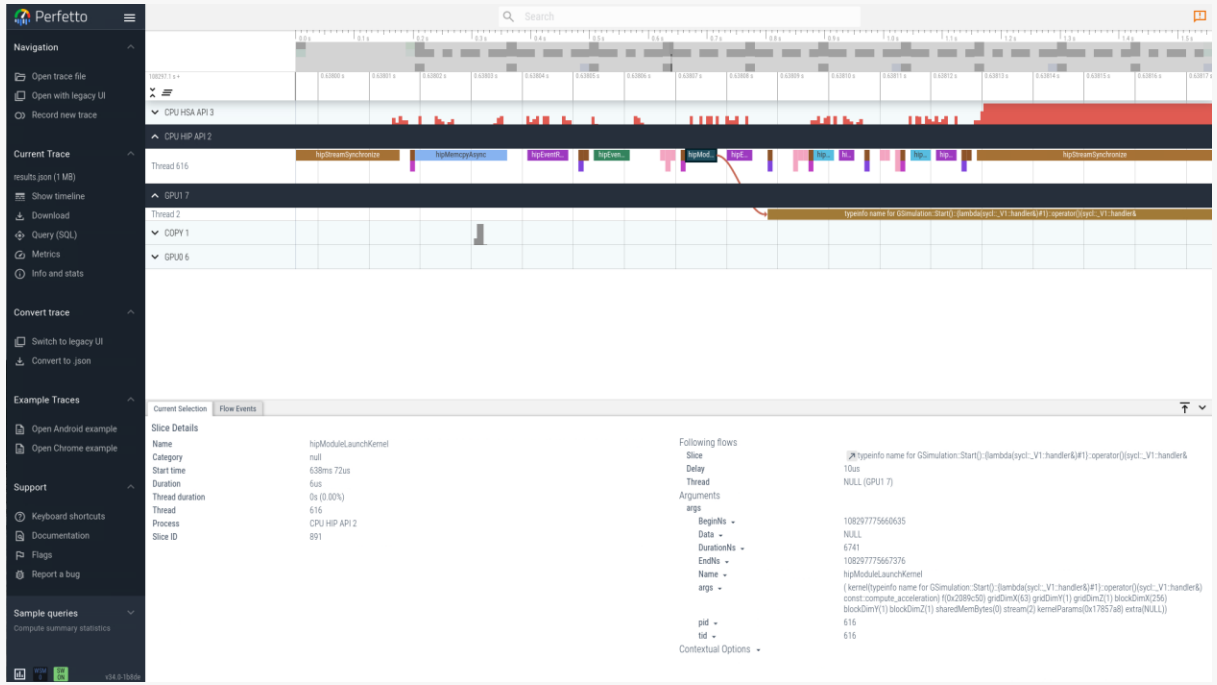
```
[Switching focus to CUDA kernel 1, grid 4, block (5,0,0), thread (32,0,0), device 0, sm 10, warp 0, lane 0]

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 0x00000000000000000555555555555 (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)
```

Profiling

Use NVIDIA NSight Systems and NSight Compute to profile SYCL code



Use AMD ROCProfiler to profile SYCL code



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

See also:

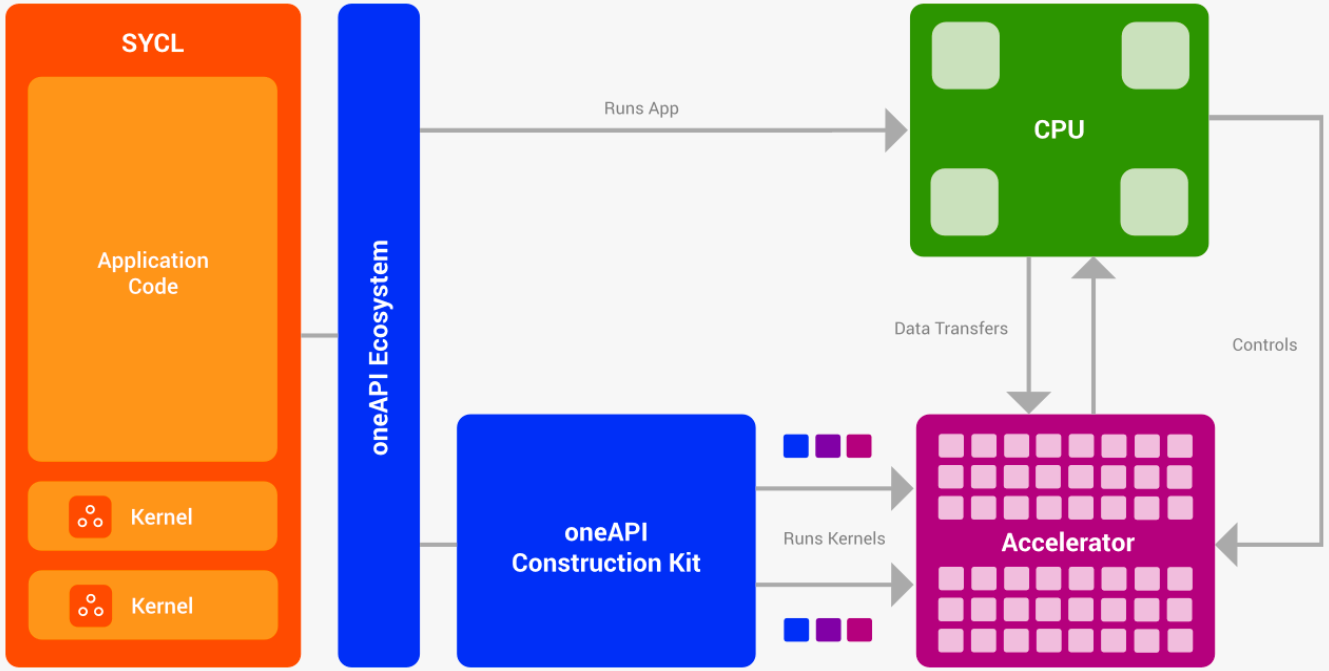
- Instructions, performance guides and blog posts at developer.codeplay.com
- My recent live demo at the oneAPI DevSummit workshop

oneAPI Construction Kit

The **oneAPI Construction Kit** enables the CPU to offload compute-intensive kernels to **custom accelerators**

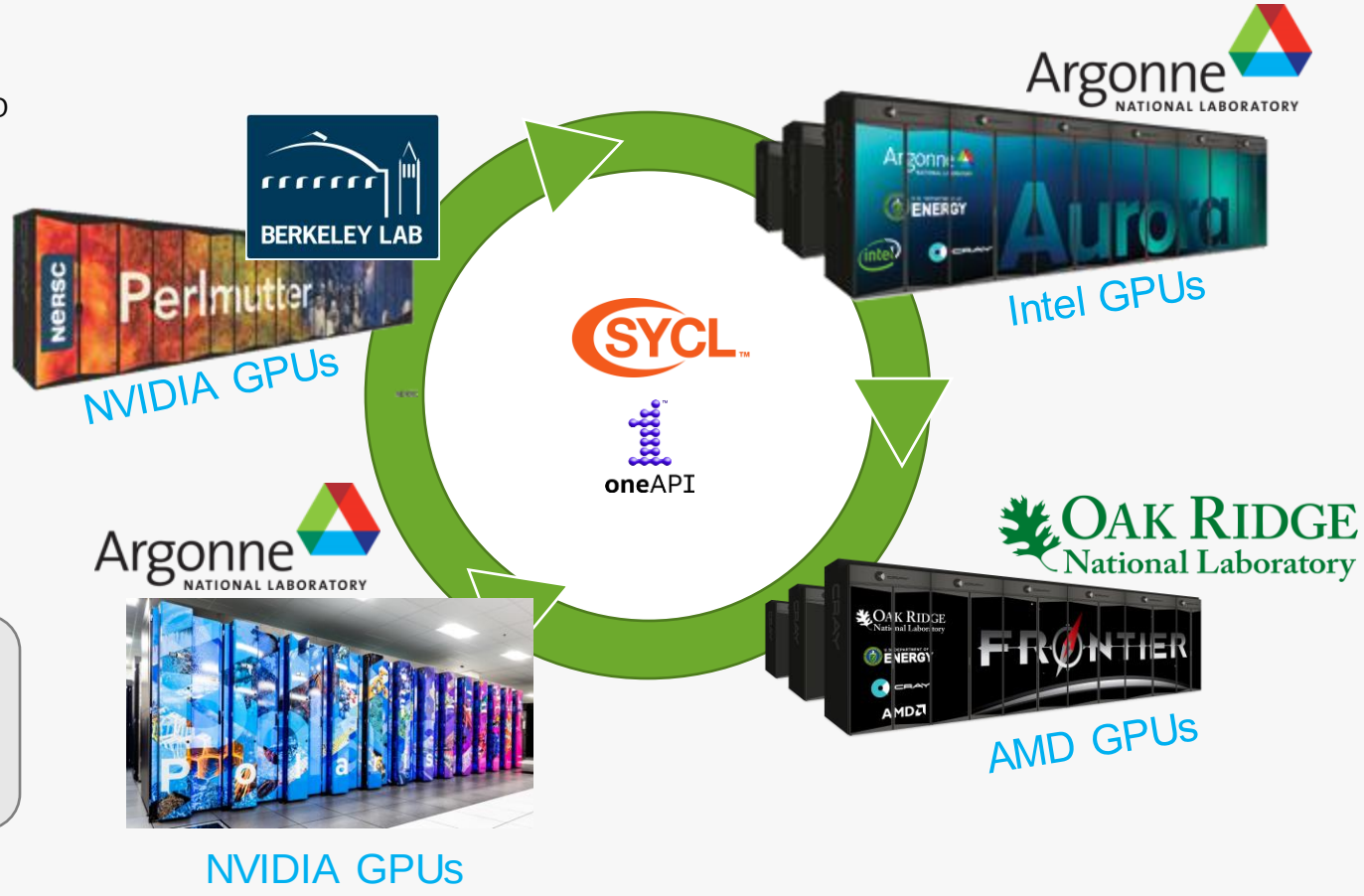
Find more details at:

- https://developer.codeplay.com/products/oneapi/construction-kit/home/
- https://github.com/codeplaysoftware/oneapi-construction-kit



SYCL Enables Supercomputers

Codeplay works in partnership with **US National Laboratories** to enable SYCL on **exascale supercomputers**

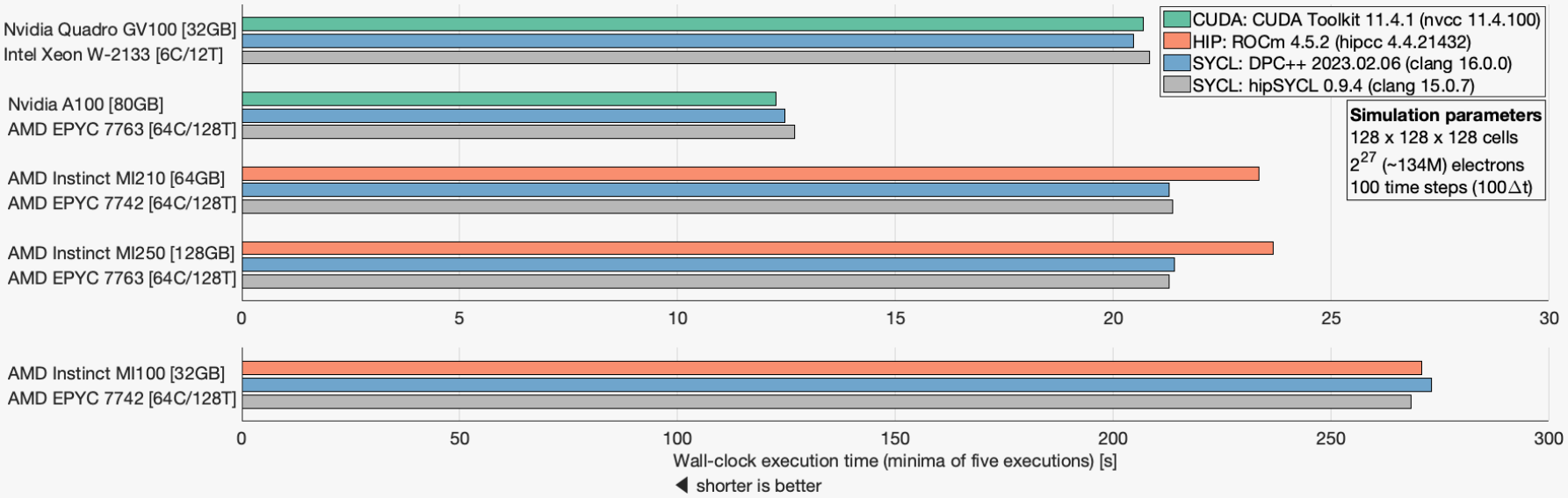


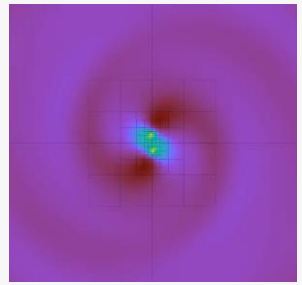
> *This work supports the productivity of scientific application developers and users through performance portability of applications between Aurora and Perlmutter.*

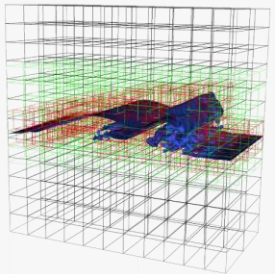
SYCL enables a broad range of software frameworks and applications

Example successful SYCL projects

- The adaptive mesh refinement framework **AMReX** adapted SYCL and demonstrated **performance matching native CUDA/HIP implementations**
- Nobre N, Grant A, Chockalingam K, Guo X (2023) farscape-project/amrex-sycl (v1.0.1). Zenodo. https://doi.org/10.5281/zenodo.8020802, https://github.com/farscape-project/amrex-sycl
- The AMReX Development Team (2023) AMReX-Codes/amrex: AMReX 23.06 (23.06). Zenodo. https://doi.org/10.5281/zenodo.7995865





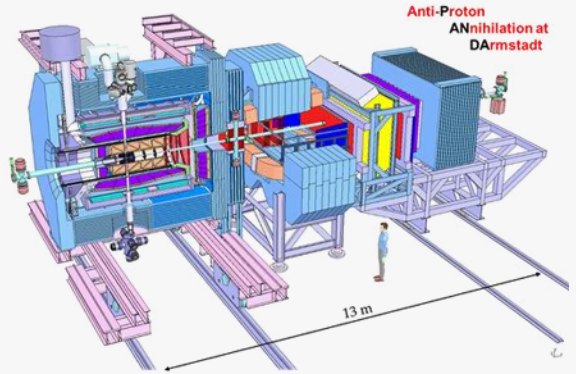


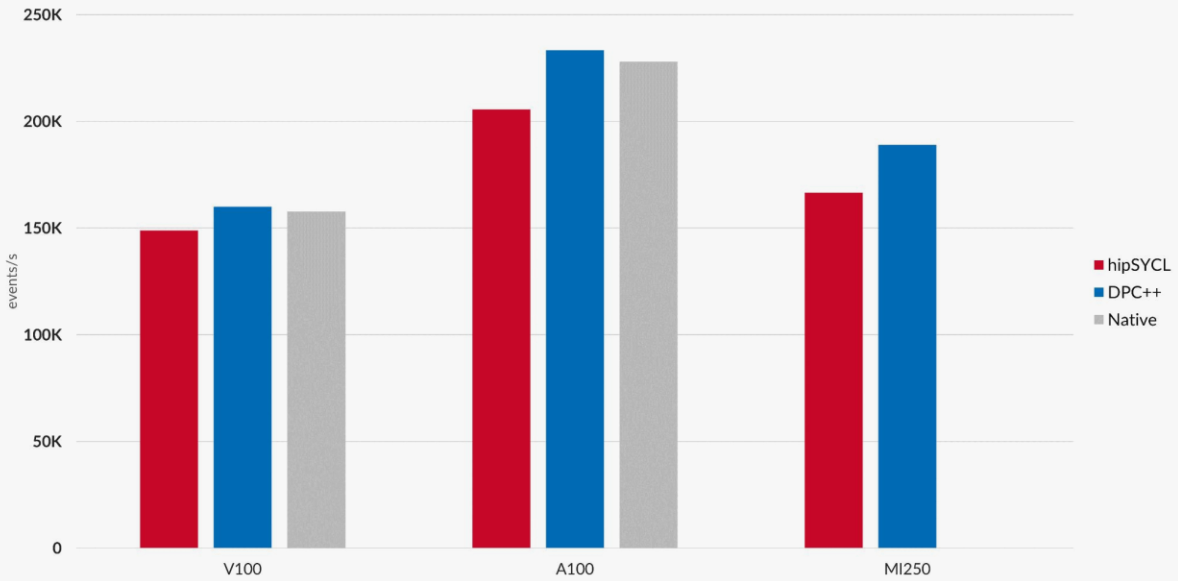
* Results in this chart are published in this GitHub project developed by N Nobre, a working repository adding Nvidia and AMD targets for AMReX using SYCL https://github.com/farscape-project/amrex-sycl

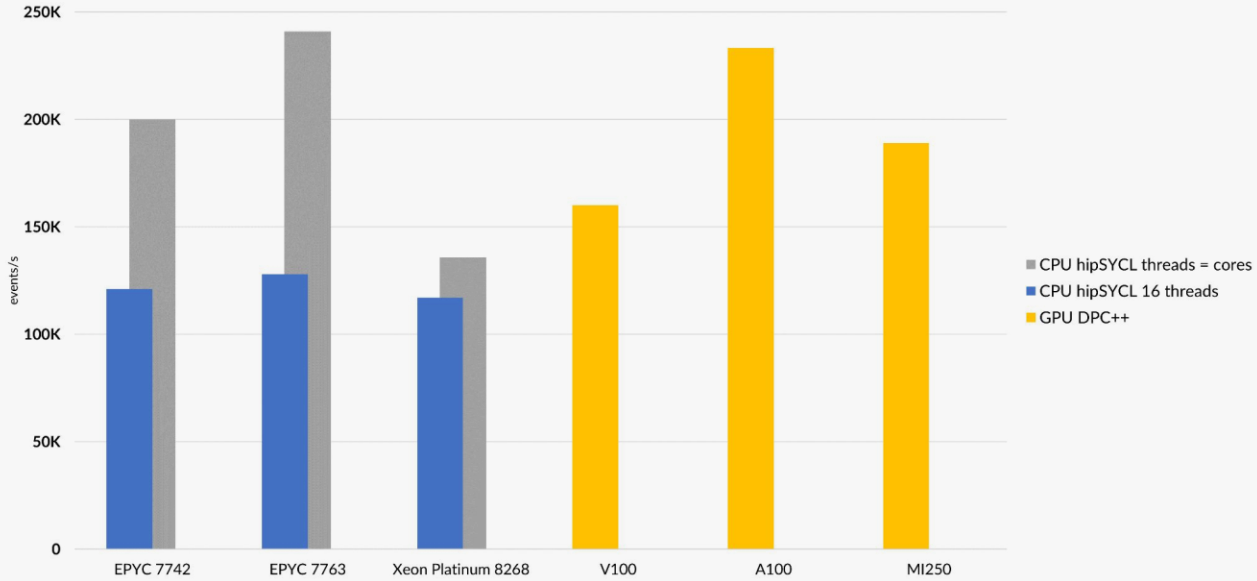
Example successful SYCL projects

- **PANDA experiment** at FAIR, GSI evaluated SYCL for their **tracking** software
- Found **excellent performance on GPU and CPU**, planning FPGA optimisation in the future
- Sobol B, Korcyl G (2023) Particle track reconstruction on heterogeneous platforms with SYCL, presentation at IWOCL 2023

> *We believe SYCL is a promising option for the use case*
> - *For prototyping, evaluating performance over platforms AND production use*
> - *Can provide satisfying and competitive performance with portability*





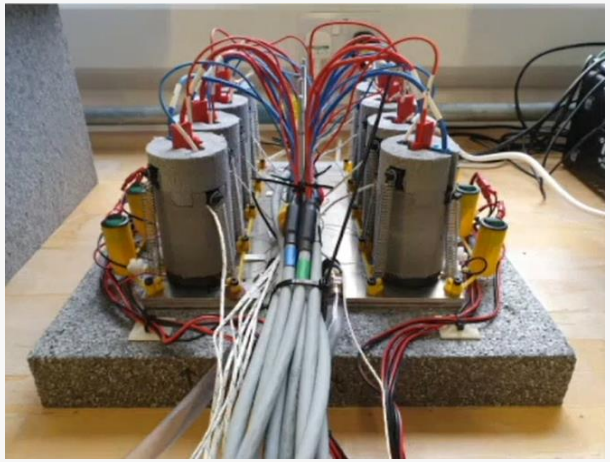


* Charts taken from this presentation: https://www.iwocl.org/wp-content/uploads/iwocl-2023-Bartosz-Sobol-1533.pdf

Example successful SYCL projects

WiZer Batteries

- Project to build novel High-Performance Hybrid Batteries for Electric Vehicles
- **Collaboration led by Williams Advanced Engineering.**
- **Codeplay's role:** Accelerating Battery Models run by Battery Management System via SYCL.



Experimental Battery Test rig at Imperial

- Quad Core Arm Cortex A53 processor
- Dual Core Arm Cortex R5 real-time processors
- Arm Mali GPU 400
- FPGA

Embedded MPSoC platform running the BMS on the Battery

Project consortium:

WILLIAMS ADVANCED ENGINEERING

Imperial College London

codeplay

Silver Power Systems

https://www.imperial.ac.uk/news/186707/building-better-batteries-your-future-electric/

Summary

- SYCL brings you **performance portability** across different architectures and vendors
- Use **standard C++** to write parallel computing projects
- Intel oneAPI toolkit with Codeplay plugins bring **NVIDIA/AMD** GPU performance for SYCL **matching the native APIs** (CUDA/HIP)
- Find guides and support in Codeplay's developer website
- Contact us about your SYCL projects

codeplay

Enable AI & HPC to be Open, Safe and Accessible to All

Notices & Disclaimers

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.

@codeplaysoft

info@codeplay.com

codeplay.com