GPU Performance Portability Using Standard C++ with SYCL

15th February 2023

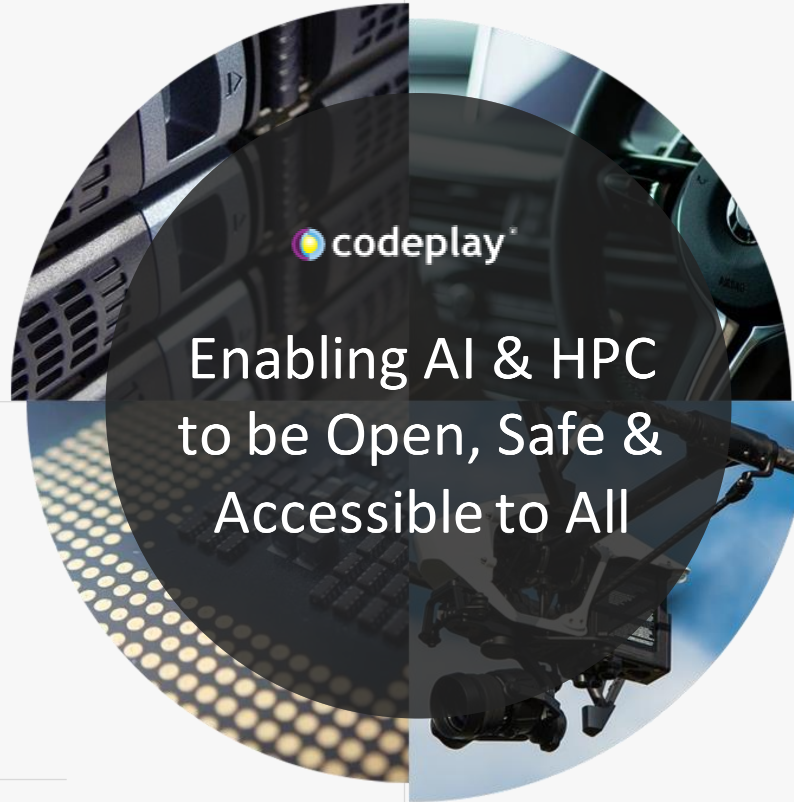
WAMTA 23 - Hugh Delaney

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.



Partners

intel
BROADCOM
SYNOPSYS
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

ComputeCpp™

C++ platform via the SYCL™ open standard, enabling vision & machine learning e.g. TensorFlow™

ComputeAorta™

The heart of Codeplay's compute technology enabling OpenCL™, SPIR-V™, HSA™ and Vulkan™

Markets

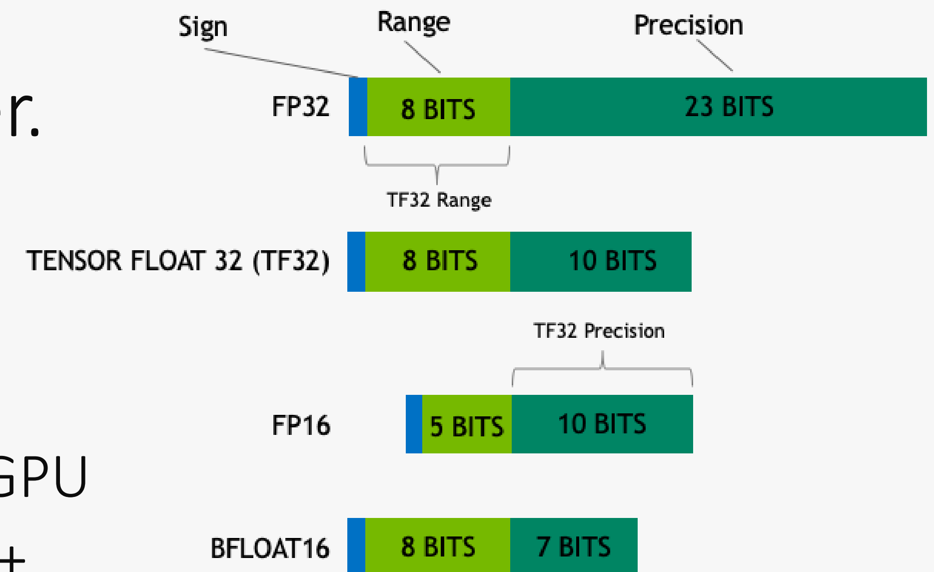
High Performance Compute (HPC)
Automotive ADAS, IoT, Cloud Compute
Smartphones & Tablets
Medical & Industrial

Technologies: Artificial Intelligence
Vision Processing
Machine Learning
Big Data Compute

Copyright Codeplay Software Ltd 2022

Hugh Delaney – Software Engineer

- Working on the HIP and CUDA backend of the DPC++ SYCL compiler.
- Recent work:
  - Add support for the CXX standard library for the CUDA backend.
  - Add support for native atomic ops for AMDGPU
  - Add support for tf32 data type for the DPC++ joint matrix API using CUDA tensor cores.



Copyright Codeplay Software Ltd 2022

What is SYCL?

- SYCL is a C++ interface enabling heterogeneous acceleration.
- SYCL is an open standard under the Khronos Group.
- SYCL can be used to run single source code on AMD, Nvidia, or Intel GPUs, and more.
- Performance of SYCL code is comparable to performance of "native" APIs.
- SYCL is the "native" heterogeneous API for Intel GPUs.

```
array_view<float> a, b, c;
<2> idx) restrict(amp) {
```

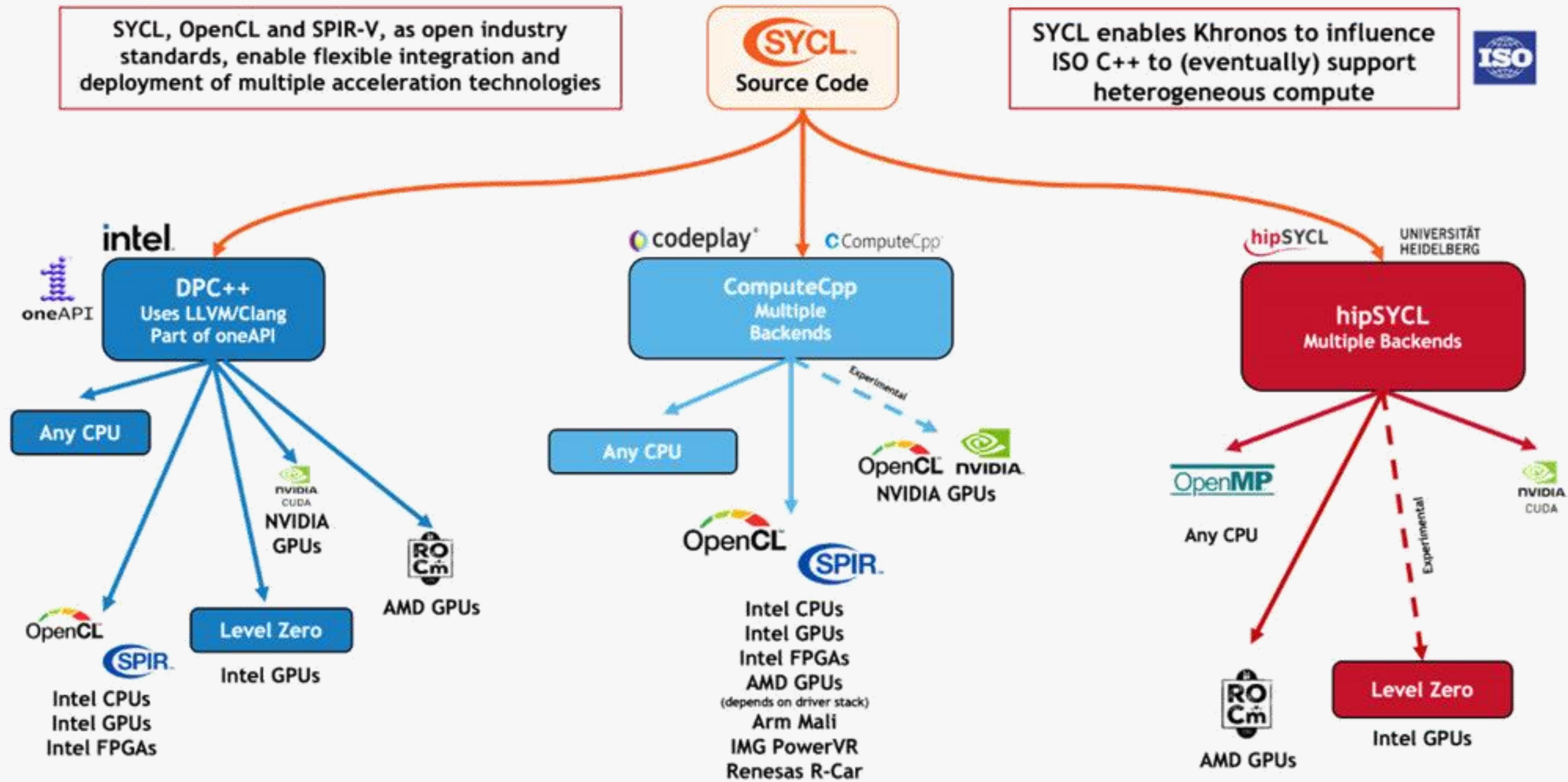
```
std::vector<float> a, b, c;

#pragma parallel_for
for(int i = 0; i < a.size(); i++) {
  c[i
}
```

```
__global__ vec_add(float *a, float *b, float *c) {
  return c[i] = a[i] + b[i];
}

float *a, *b, *c;
vec_add<<<range>>>(a, b, c);
```

```
cgh.parallel_for(range, [=](cl::sycl::id<2> idx) {
  c[idx] = a[idx] + b[idx];
});
```


Copyright Codeplay Software Ltd 2022

SYCL Implementations



Copyright Codeplay Software Ltd 2022

What Does SYCL Look Like?

```cpp
#include <CL/sycl.hpp>

using T = float;

constexpr size_t n = 1638400;

int main() {
  std::vector<T> A(n);
  std::vector<T> B(n);

  for (auto i = 0; i < n; ++i) {
    A[i] = i;
  }

  sycl::queue q{};
  std::cout << "Running on device: "
            << q.get_device().get_info<sycl::info::device::name>() << std::endl;

  try {
    sycl::buffer bufA{A};
    sycl::buffer bufB{B};

    q.submit([&](sycl::handler &cgh) {
      sycl::accessor accA{bufA, cgh};
      sycl::accessor accB{bufB, cgh};
      cgh.parallel_for(sycl::range{n},
                       [=](sycl::id<1> i) { accB[i] = accA[i] * accA[i]; });
    });
  } catch (sycl::exception &e) {
    std::cerr << "Caught exception: " << e.what();
  }

  std::cout << "Results: " << std::endl;
  std::for_each(B.begin(), B.begin() + 10, [](T &c) { std::cout << c << " "; });
  std::cout << "\n\n";
}
```

Device code

Executor x86-64 icx 2022.1.0 (C++, Editor #1)

x86-64 icx 2022.1.0 | -fsycl

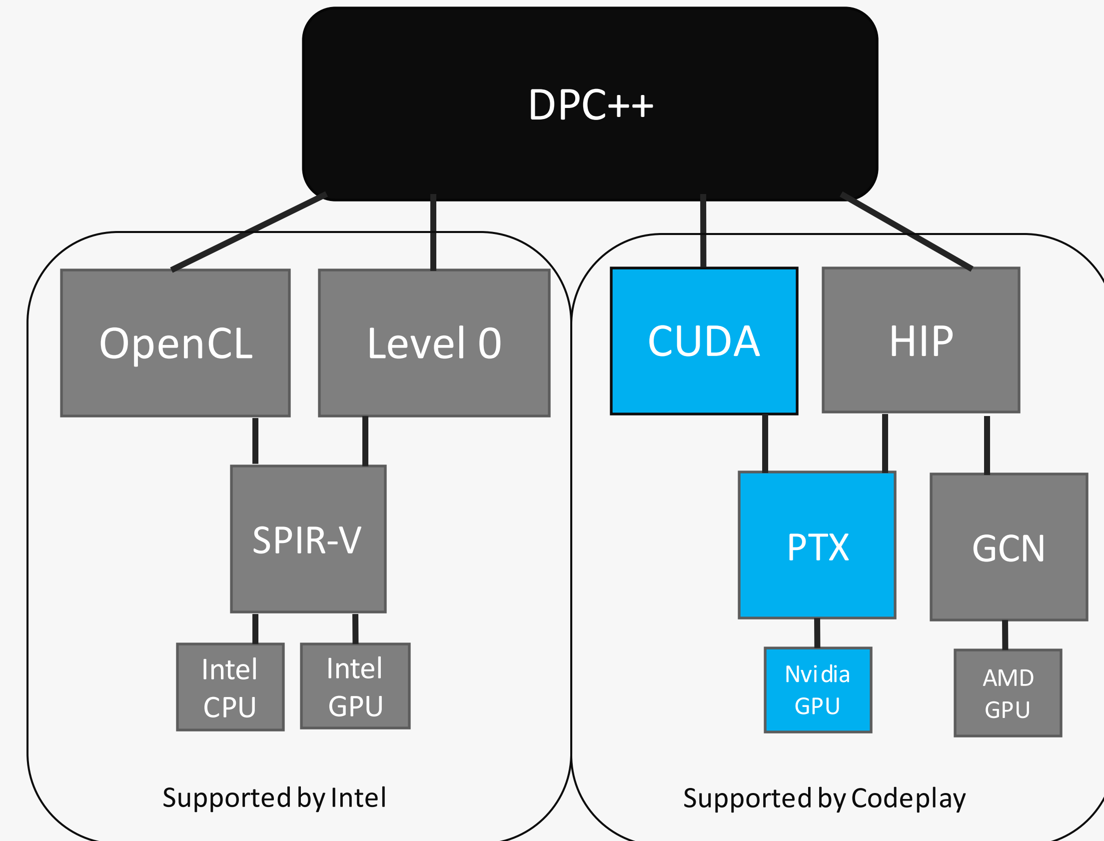
```
Program returned: 0
Program stdout
Running on device: AMD EPYC 7R32
Results:
0 1 4 9 16 25 36 49 64 81
```

- Modern C++
- Uses exceptions, lambdas, templates, CTAD.
- Aside: Library-only implementations are possible.
- Experiment on https://godbolt.org/z/6aon4Evsz

Copyright Codeplay Software Ltd 2022

What is DPC++?

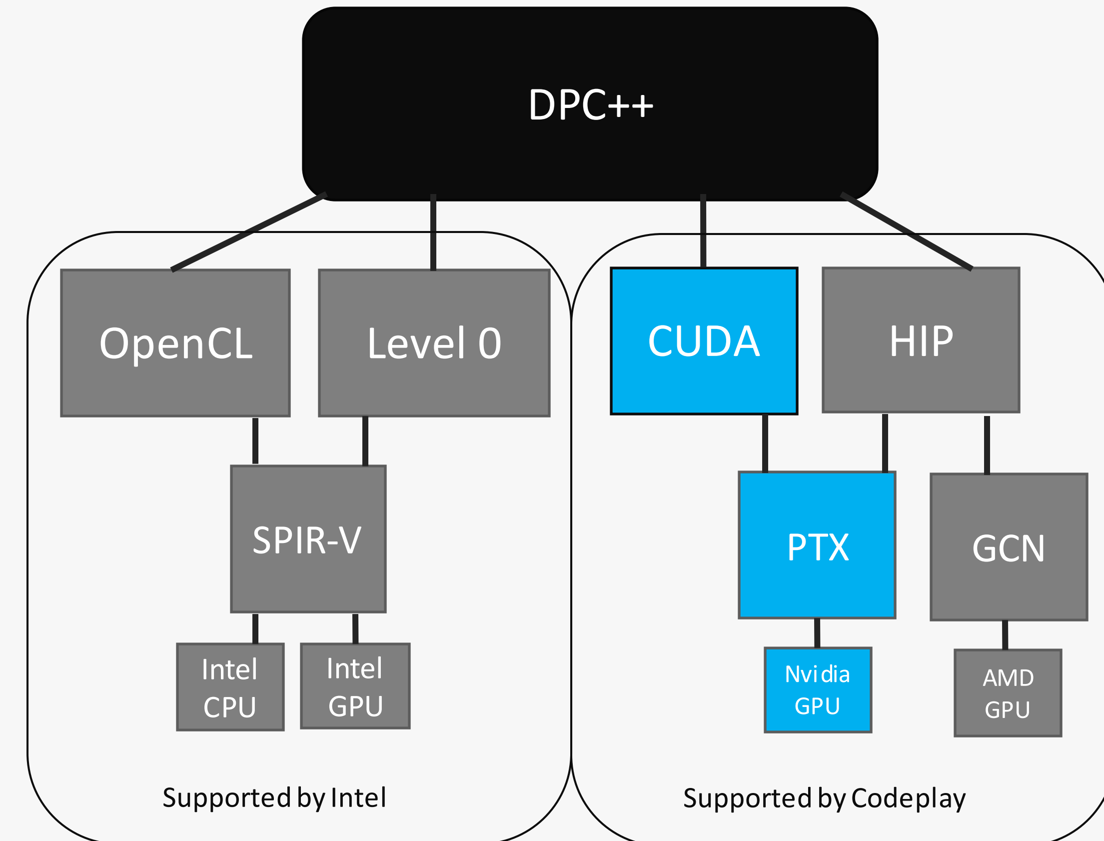
- DPC++ is the open source Intel-backed SYCL implementation.
- DPC++ is an LLVM fork.
  - Shared infrastructure with wider LLVM project
  - CUDA-clang / openMPTarget / AMDGPU



Copyright Codeplay Software Ltd 2022

How Does the DPC++ Toolchain Work?

- Kernel code is compiled for any platform specified to the compiler.
- Device code is bundled within the final binary.



Copyright Codeplay Software Ltd 2022

Demo

- Compilation in action!

Copyright Codeplay Software Ltd 2022

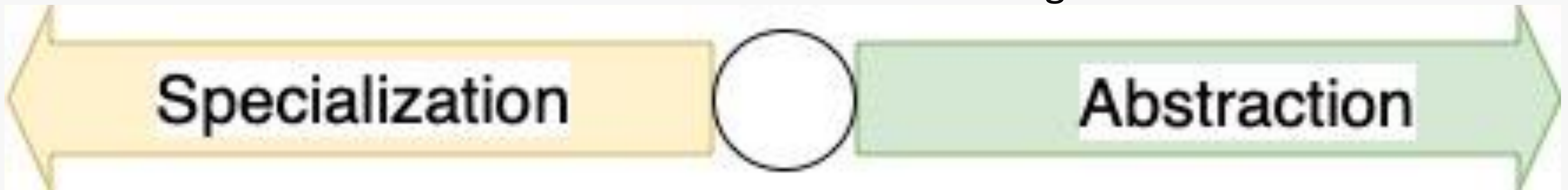
Portability

Examples of Specialization:

- Writing code for a specific architecture (i.e. GPUs)
- Writing code using vendor-specific APIs. (i.e. CUDA)
- Optimizing code for the specific features of a particular device.

Examples of Abstraction:

- Using APIs that map to different backends. (i.e. SYCL, openMP, Kokkos)
- Using libraries that support multiple backends. (i.e. oneDNN, oneMKL)
- Allowing an API to abstract away details of data transfer and heterogeneous execution.



Copyright Codeplay Software Ltd 2022

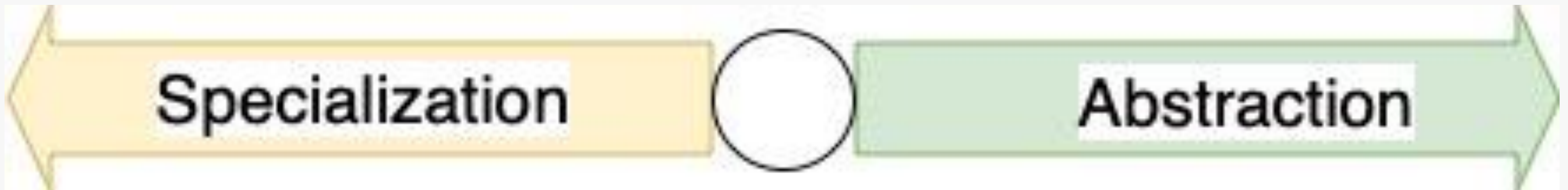
Portability

Benefits of Specialization:

- Optimization

Benefits of Abstraction:

- Code portability.
- High level APIs (usually) increase productivity.
- No need to learn platform-specific APIs.



Copyright Codeplay Software Ltd 2022

Portability

More programming effort
Lower productivity

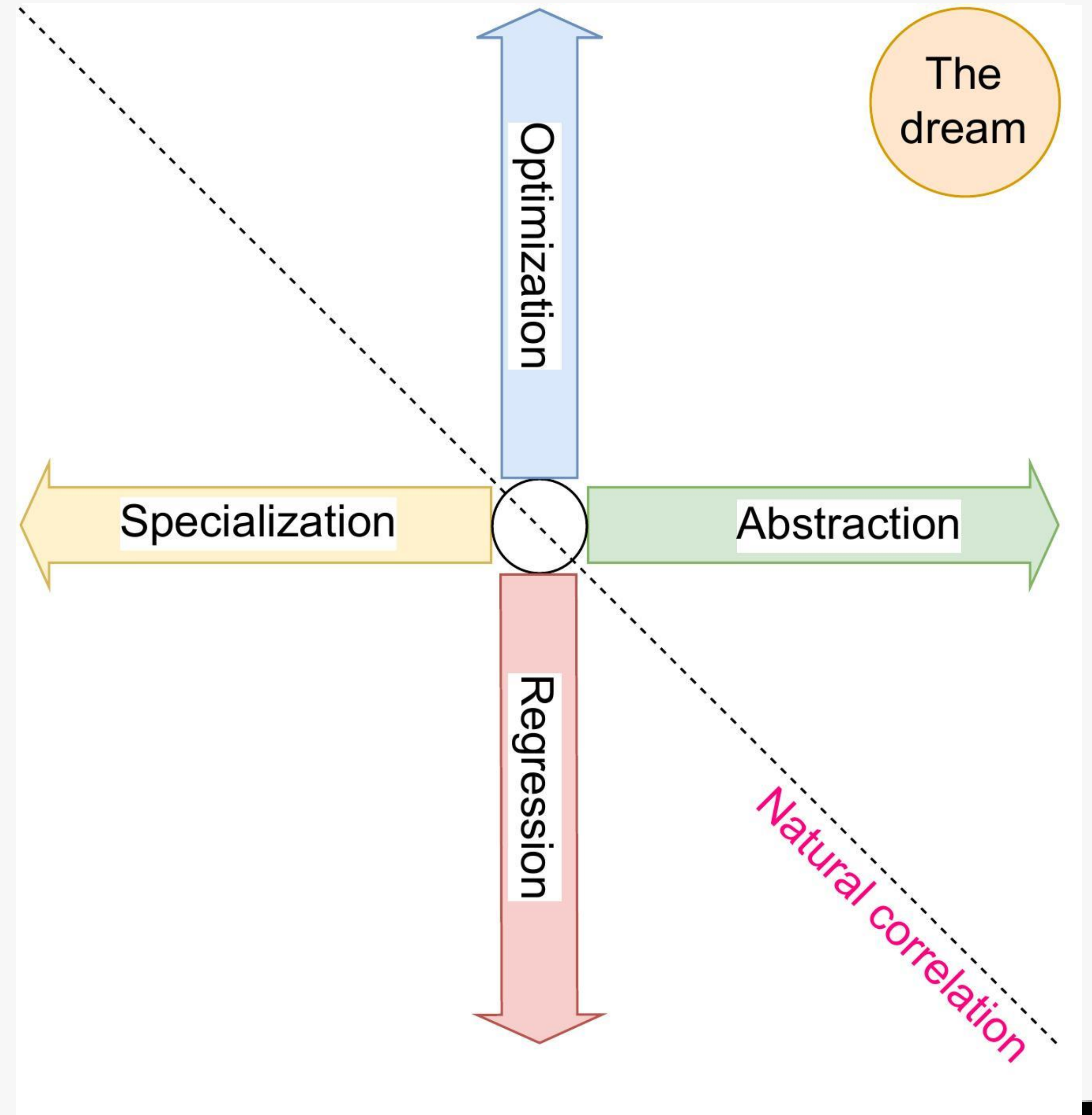
Less programming effort
Higher productivity

Copyright Codeplay Software Ltd 2022

Performance Portability

- Caveat: this is not a numerically precise scale. For a more rigorous presentation:
  - *"Navigating Performance, Portability, and Productivity", Pennycook, Deakin et al.*

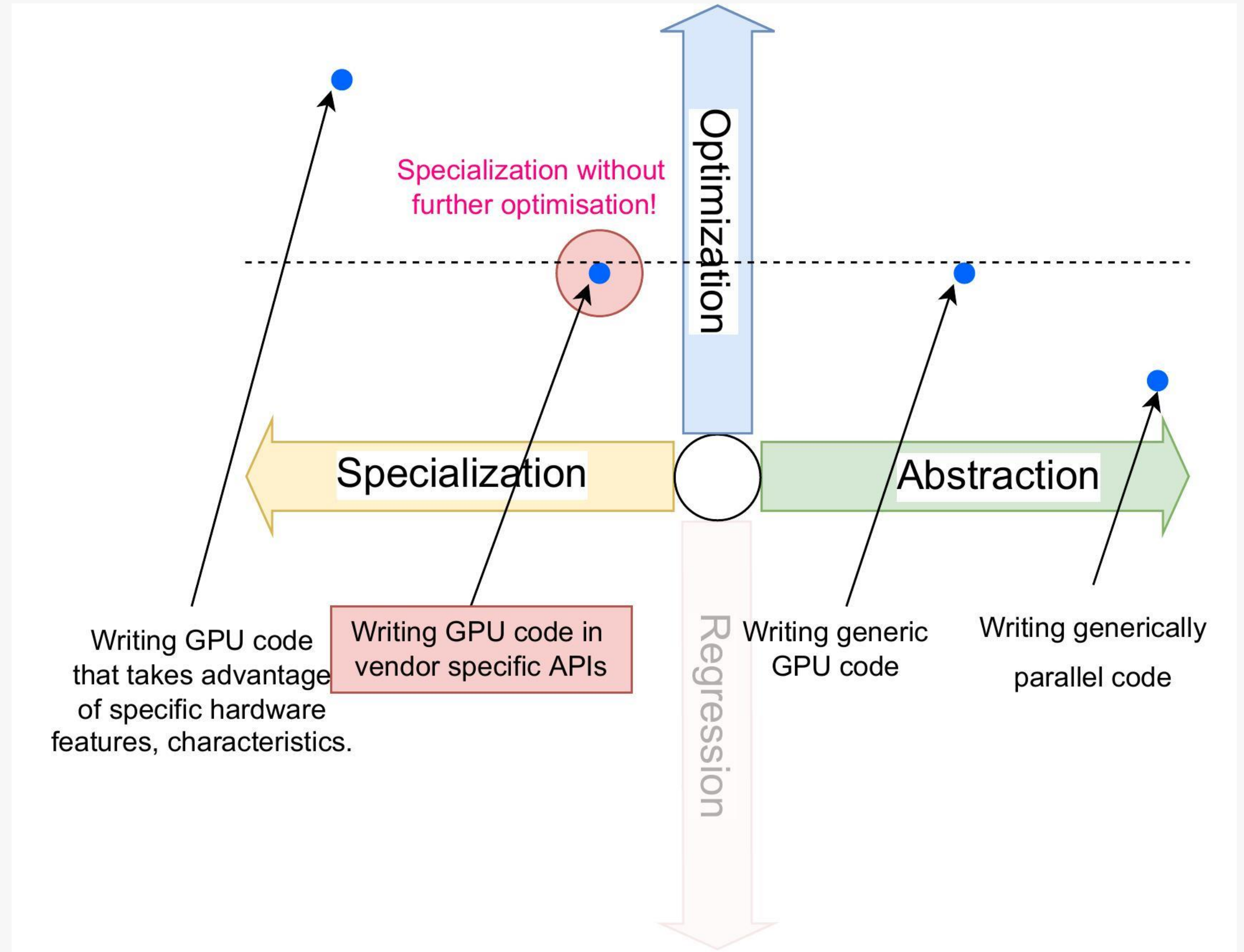
- In general, the more specialized our code, the more optimization we should expect.
- We should not accept greater specialization, unless it offers greater performance.



Copyright Codeplay Software Ltd 2022

Performance Portability

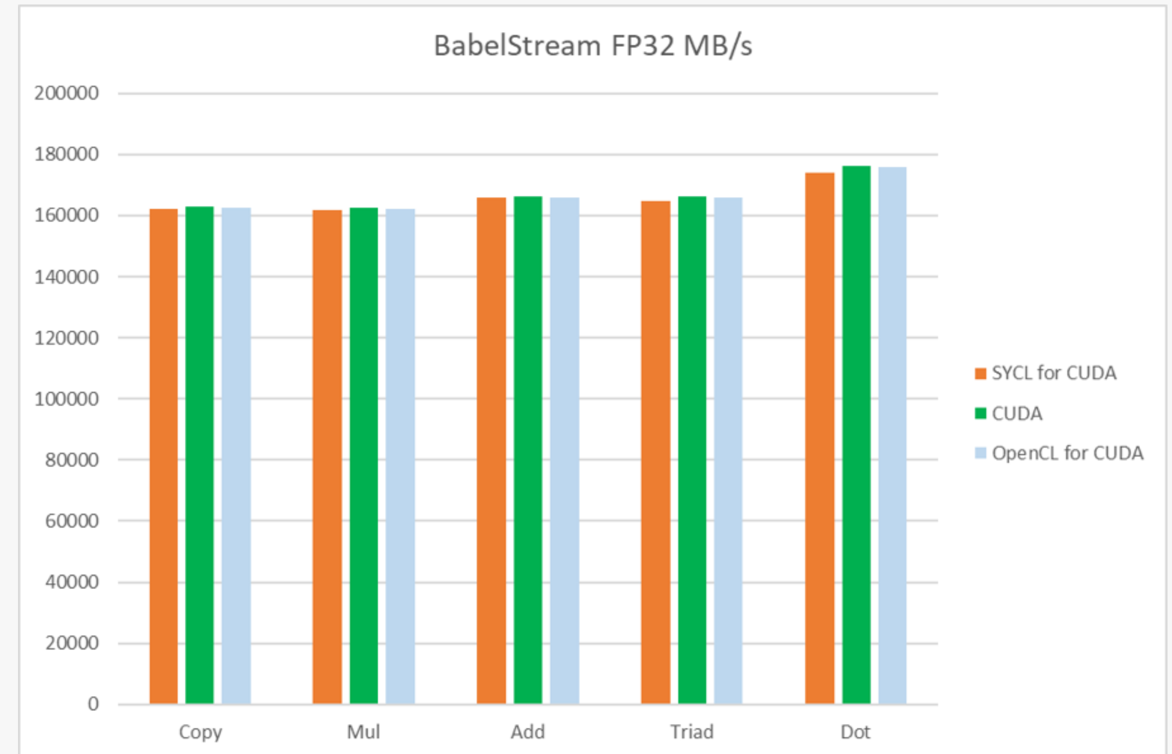
- Can GPU code written in SYCL really match the performance of native APIs?



Copyright Codeplay Software Ltd 2022

DPC++ performance on Nvidia GPUs

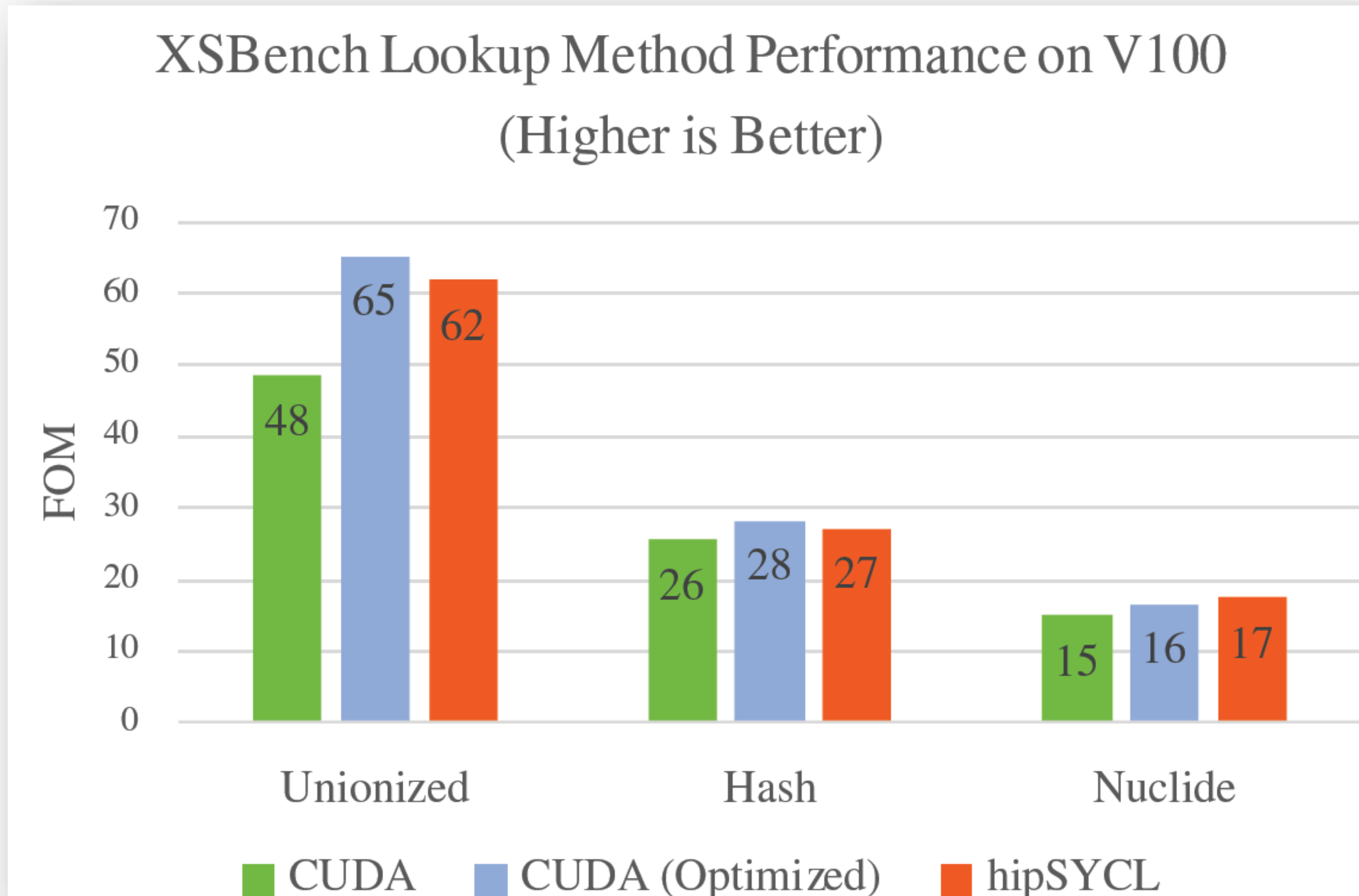
- This graph compares the BabelStream benchmarks results for:
  - Native CUDA code
  - OpenCL code
  - SYCL code using the CUDA backend
- Run on GeForce GTX 980 with CUDA 10.1
- Minimal SYCL overhead



http://uob-hpc.github.io/BabelStream

© 2021 Codeplay Software Ltd. - CONFIDENTIAL

Argonne National Labs Comparison



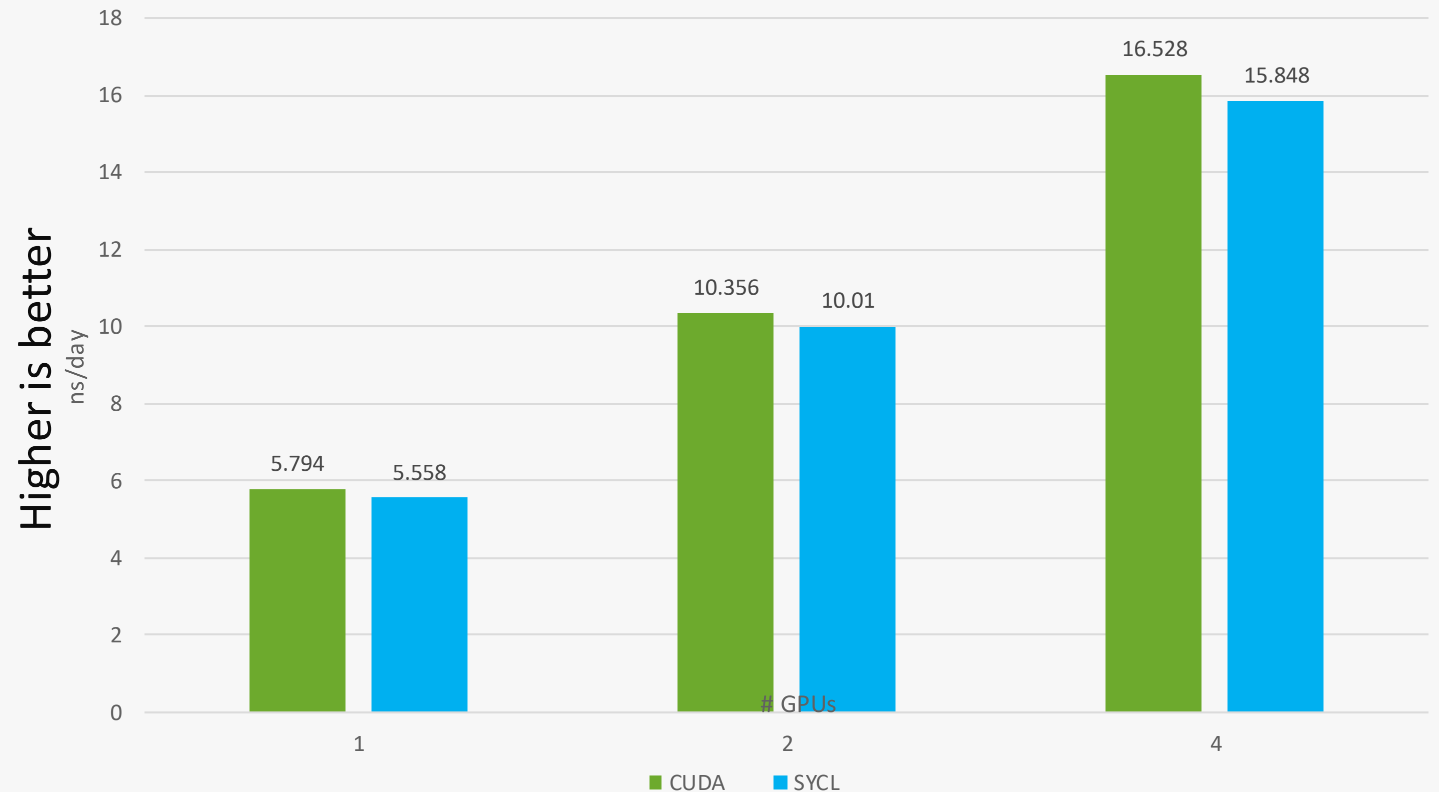
SYCL achieves equivalent performance to Optimized CUDA

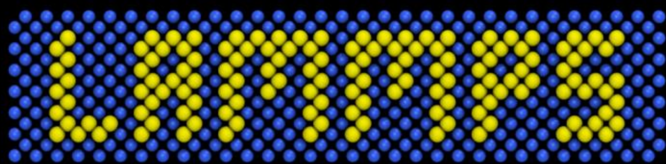
Presented at IWOCL, April'20

© 2021 Codeplay Software Ltd. - CONFIDENTIAL

LAMMPS on DPC++

- Molecular/Particle Simulator
- Distributed heterogenous computing over multiple GPUs.



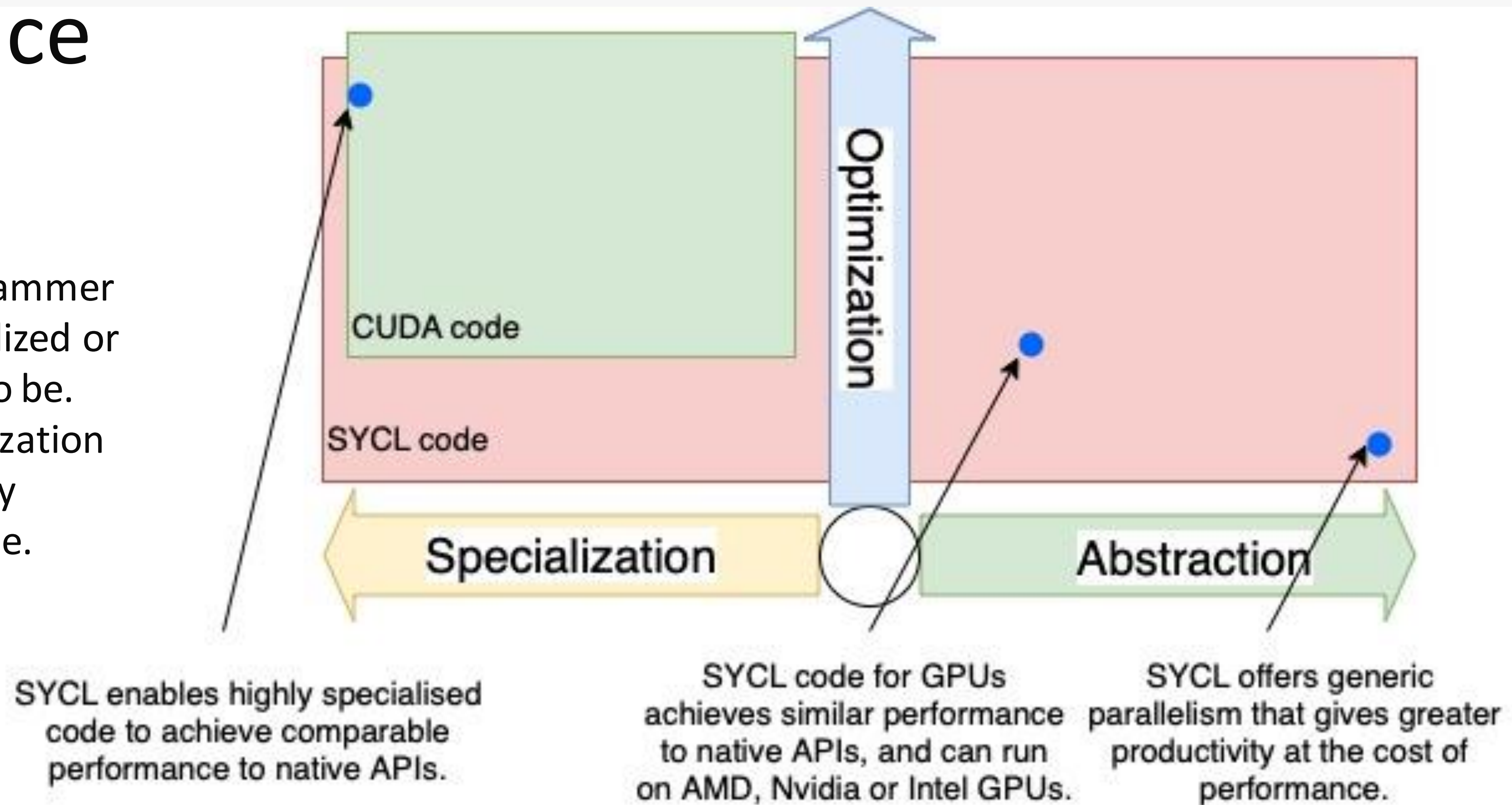


- Kokkos-CUDA vs Kokkos-SYCL for CUDA (same GPUs)

Copyright Codeplay Software Ltd 2021

Performance Portability

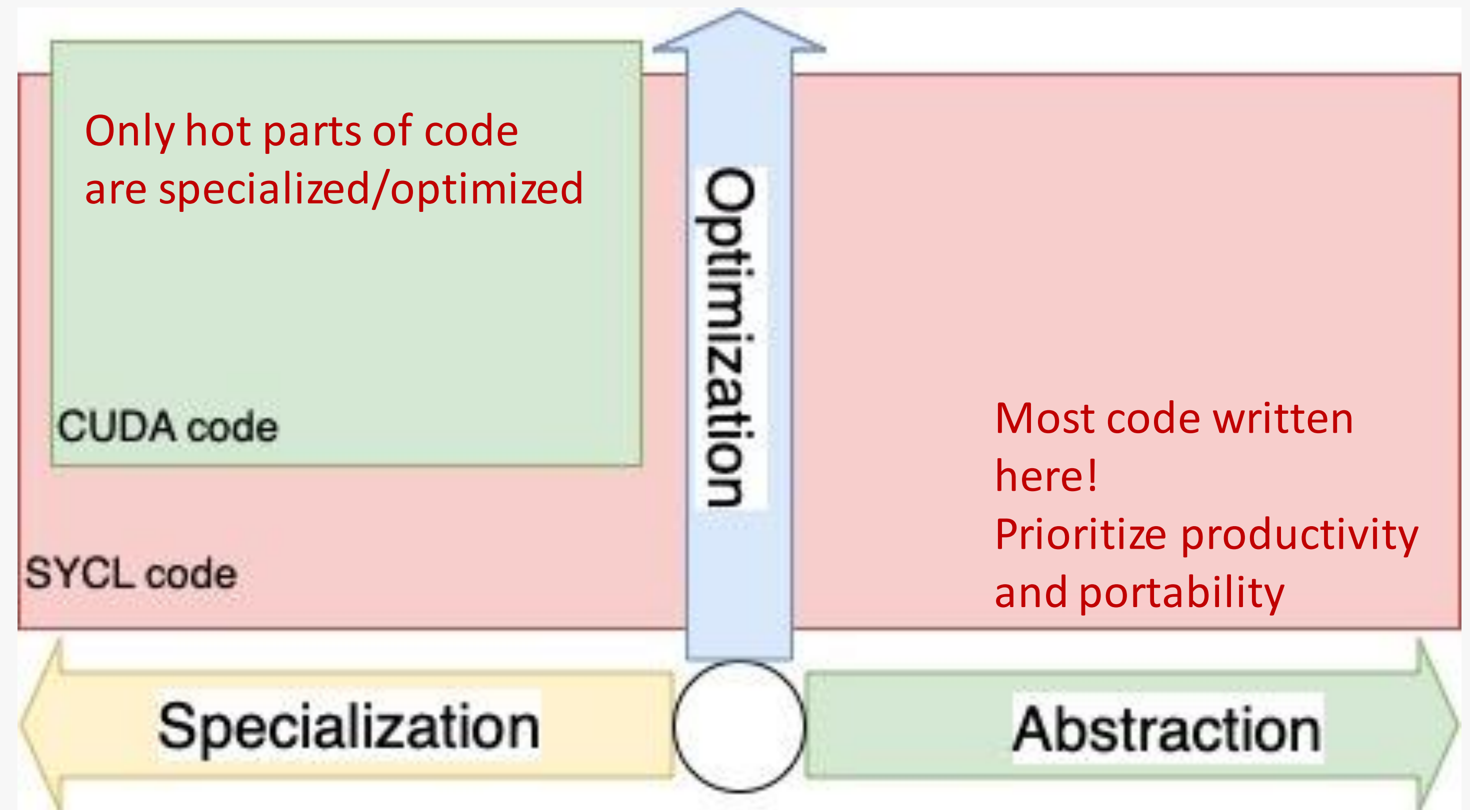
- SYCL allows the programmer to choose how specialized or abstract code ought to be.
- The degree of specialization or abstraction can vary throughout a codebase.



Copyright Codeplay Software Ltd 2022

Performance Portability

- In a given codebase, only the "hot" parts need be highly specialized.
- Non "hot" parts are written in generically parallel SYCL.
    - Improves code portability.
    - Improves programmer productivity.

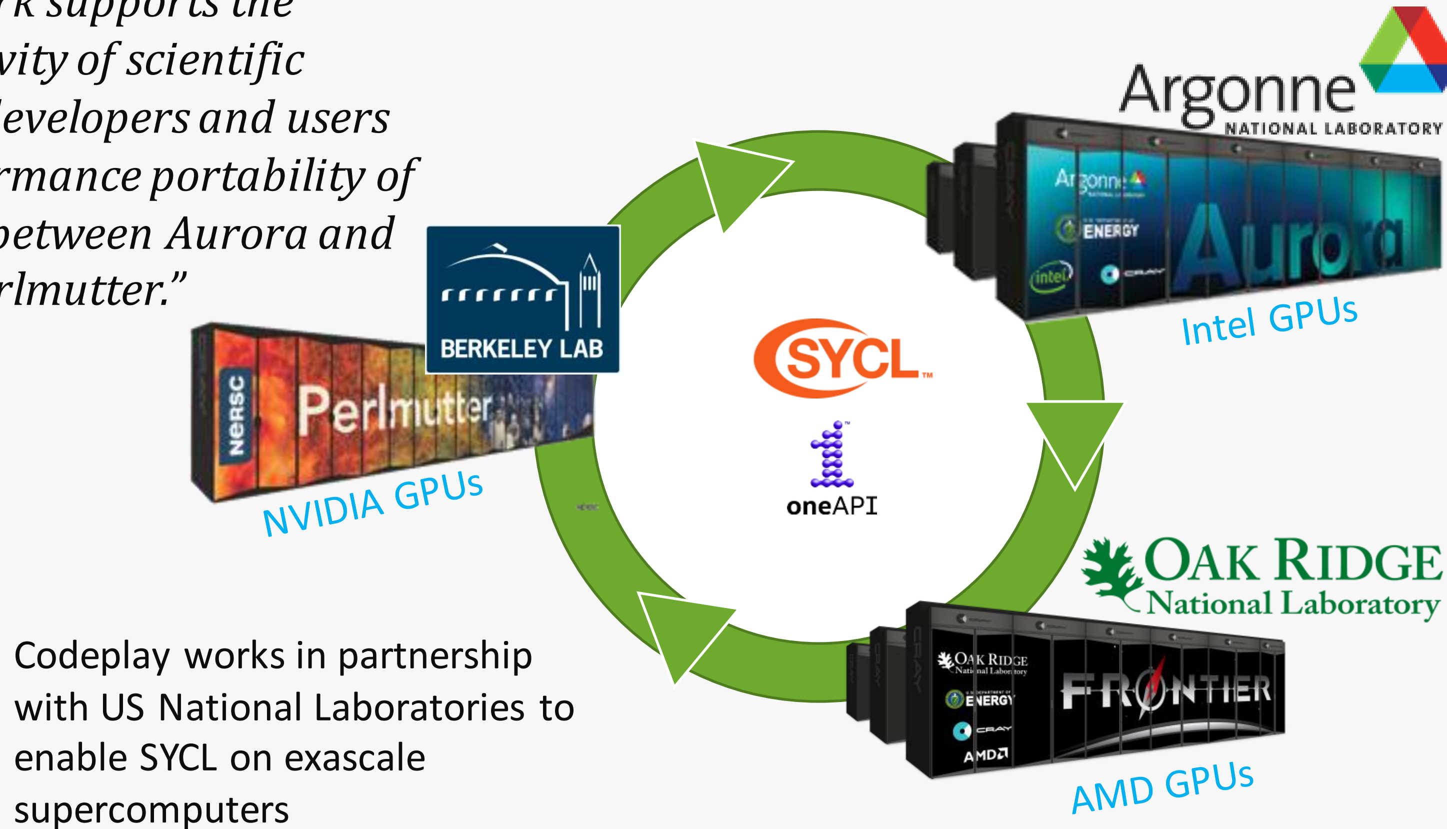


Copyright Codeplay Software Ltd 2022

SYCL Enables Supercomputers

NERSC National Energy Research Scientific Computing Center

"this work supports the productivity of scientific application developers and users through performance portability of applications between Aurora and Perlmutter."



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

Enables a broad range of software frameworks and applications

Copyright Codeplay Software Ltd 2022

Takeaway

With SYCL you can:

- Use modern C++.
- Write code that can target multiple architectures/platforms.
- Attain comparable performance to native APIs.
- Choose how generic or specialized your code ought to be.

Copyright Codeplay Software Ltd 2022

Questions

COMPILER EXPLORER Add... More Templates Support Compiler Explorer on Patreon! Sponsors Backtrace intel Share Policies

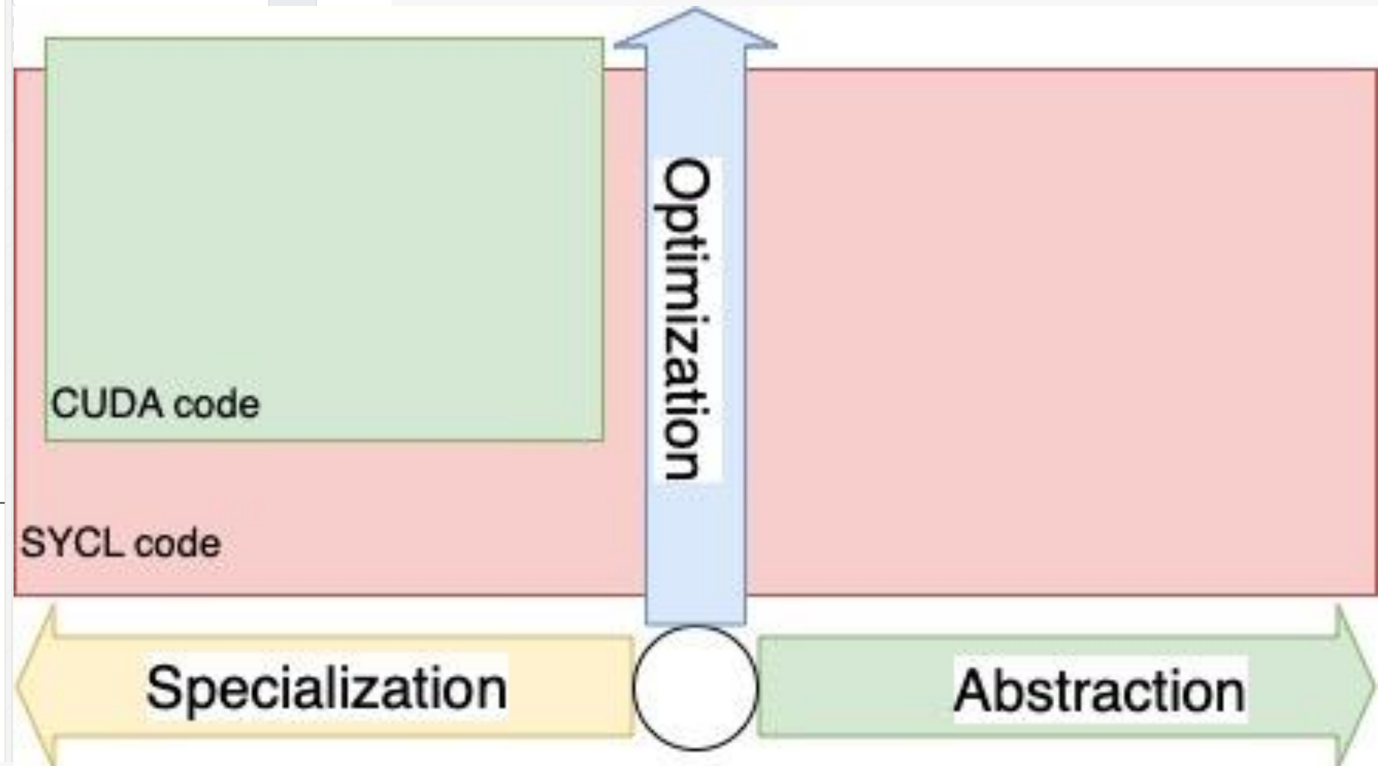
C++ source #1 | A Save/Load Add new... Vim CppInsights Quick-bench C++

```cpp
#include <CL/sycl.hpp>

using T = float;

constexpr size_t n = 1638400;

int main() {
  std::vector<T> A(n);
  std::vector<T> B(n);

  for (auto i = 0; i < n; ++i) {
    A[i] = i;
  }

  sycl::queue q{};
  std::cout << "Running on device: "
            << q.get_device().get_info<sycl::info::device::name>() << std::endl;

  try {
    sycl::buffer bufA{A};
    sycl::buffer bufB{B};

    q.submit([&](sycl::handler &cgh) {
      sycl::accessor accA{bufA, cgh};
      sycl::accessor accB{bufB, cgh};
      cgh.parallel_for(sycl::range{n},
                       [=](sycl::id<1> i) { accB[i] = accA[i] * accA[i]; });
    });
  } catch (sycl::exception &e) {
    std::cerr << "Caught exception: " << e.what();
  }

  std::cout << "Results: " << std::endl;
  std::for_each(B.begin(), B.begin() + 10, [](T &c) { std::cout << c << " "; });
  std::cout << "\n\n";
}
```

Device code

Executor x86-64 icx 2022.1.0 (C++, Editor #1) | A Wrap lines | x86-64 icx 2022.1.0 -fsycl

https://godbolt.org/z/E33oTG8vE



Copyright Codeplay Software Ltd 2022