ConcurrencyInES: SYCL for Embedded

Joe Todd, Víctor Pérez

HiPEAC–Jan 16, 2023

Copyright Codeplay Software Ltd 2023

Overview

- Why SYCL for Embedded Systems?
  - Concurrency Model
  - Heterogeneous Memory Model
  - Scheduling
  - Synchronisation
- Intro to SYCL programming model
- A small SYCL application (Vector Addition)
- Case Studies
  - Automotive
  - Drones
  - Medical Imaging

Copyright Codeplay Software Ltd 2021

Why SYCL

- *End of the Free Lunch* implies heterogeneous hardware
- SYCL provides standardized C++ API for portable software acceleration across different types of hardware
- SYCL code is single-source!

© 2023 Codeplay Software Ltd.

Why targeting SYCL to embedded platforms

- Artificial Intelligence (AI) operations (training and inference) are increasingly performed "at the Edge" by embedded devices that are typically less powerful and have more constraints/restrictions than HPC hardware
  - Limited memory, processing power and energy consumption
  - Specialized hardware (e.g. ASIC, DSP, FPGA etc; IoT devices)
- SYCL as an open standard with a growing open-source software ecosystem enables reusing existing AI acceleration written for HPC on a wide range of diverse embedded platforms.

© 2023 Codeplay Software Ltd.

What SYCL can do for you

- Heterogeneous hardware
  - GPU, CPU, FPGA, ASIC, custom silicon!
- Pure C++ code
- Well defined:
  - Concurrency model
  - Memory model
- Scheduling
- Performance Portability

© 2023 Codeplay Software Ltd.

Concurrency Model

© 2023 Codeplay Software Ltd.

Concurrency Model

Typical multi-core

- Several threads
- Homogeneous hardware
- Shared memory

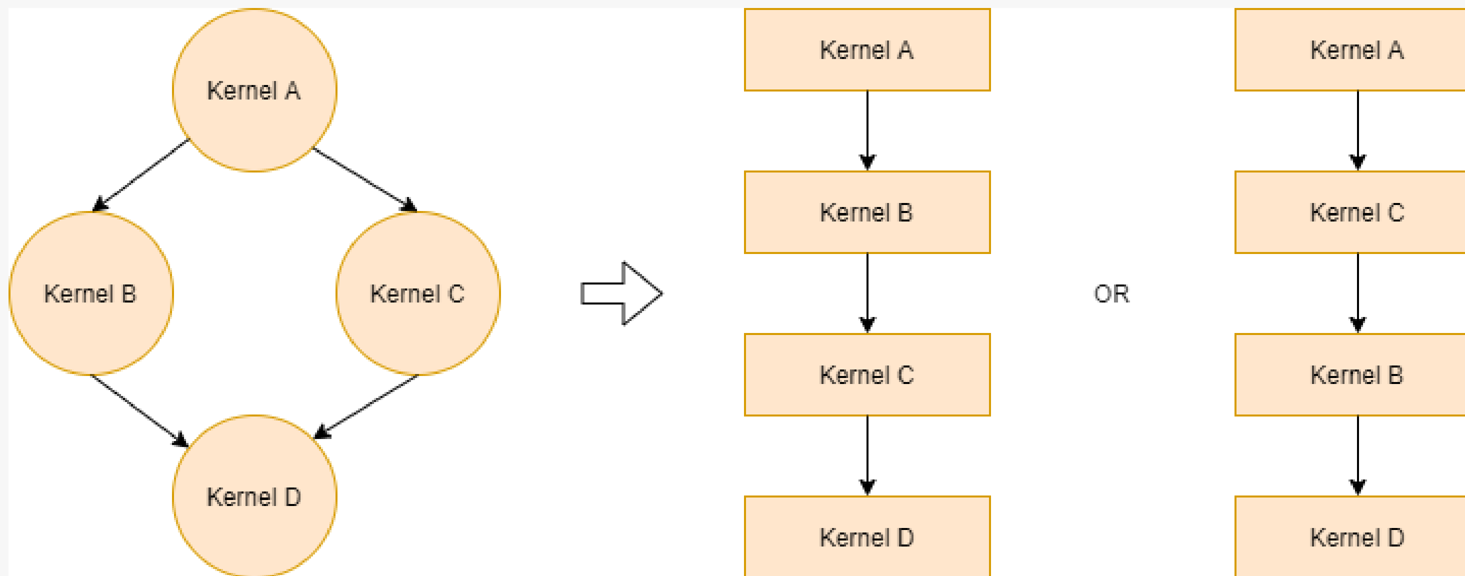
SYCL

- Thousands of threads
- Host & Accelerator(s)
- Separate memory spaces
- Host is in charge:
  - Schedules kernels & data flow
- Accelerator does the work
- Queue per accelerator

© 2023 Codeplay Software Ltd.

Concurrency & Scheduling

- Host & 1 or more accelerators
  - Host can also use itself as an accelerator (e.g. via CPU OpenCL runtime)
- Work split into discrete kernels, submitted to device's queue
- SYCL provides both implicit & explicit task graph generation



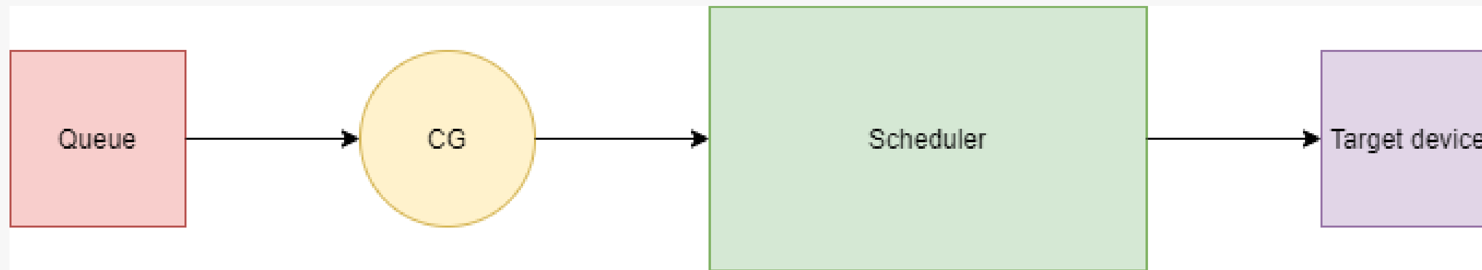
© 2023 Codeplay Software Ltd.

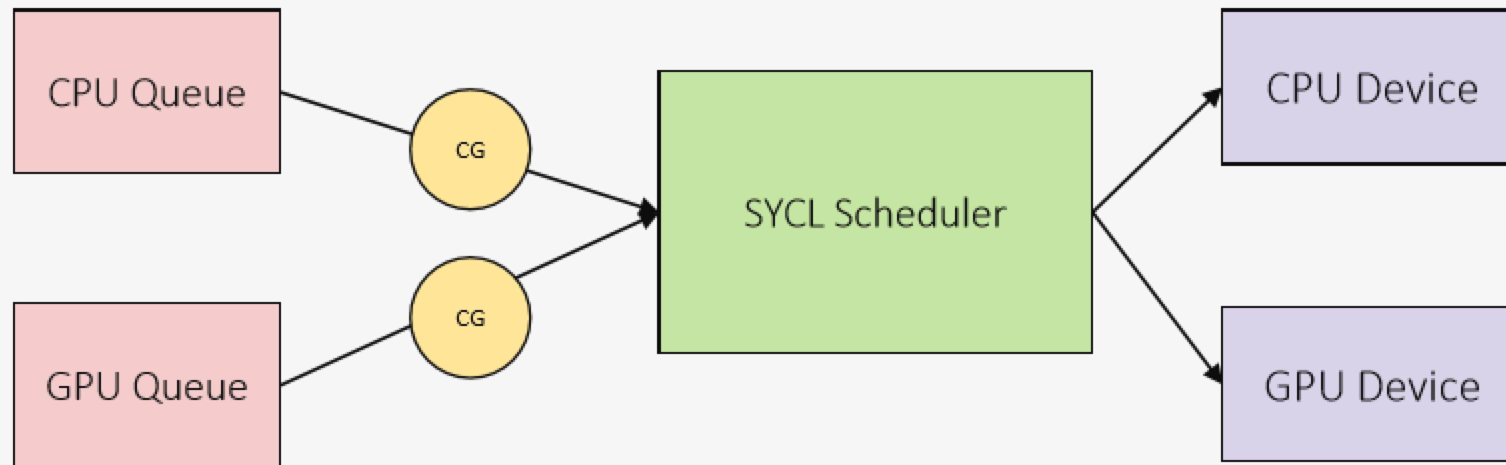
Scheduling

- Queues & events allow the SYCL runtime to handle scheduling
- Host is free to:
  - Get on with other work
  - Schedule tasks on other devices
  - Synthesise results
- host_task allows us to effectively write host callbacks in the task graph

© 2023 Codeplay Software Ltd.

Scheduling





© 2023 Codeplay Software Ltd.

SCHEDULING ON MULTICORE PROCESSORS

- Utilisation
  - For m resources (cores) and n tasks, how to schedule tasks so to avoid underutilisation of resources? How to avoid idle resources? (without using static scheduling), while at the same time
    - Minimise pre-emption
    - Minimise spinning
- Deadlines
  - No optimal on-line scheduler can exist for a set of jobs with two or more distinct deadlines on any ($m > 1$) multiprocessor system. Theorem [Hong, Leung: RTSS 1988, IEEE TCO 1992]

Memory Model

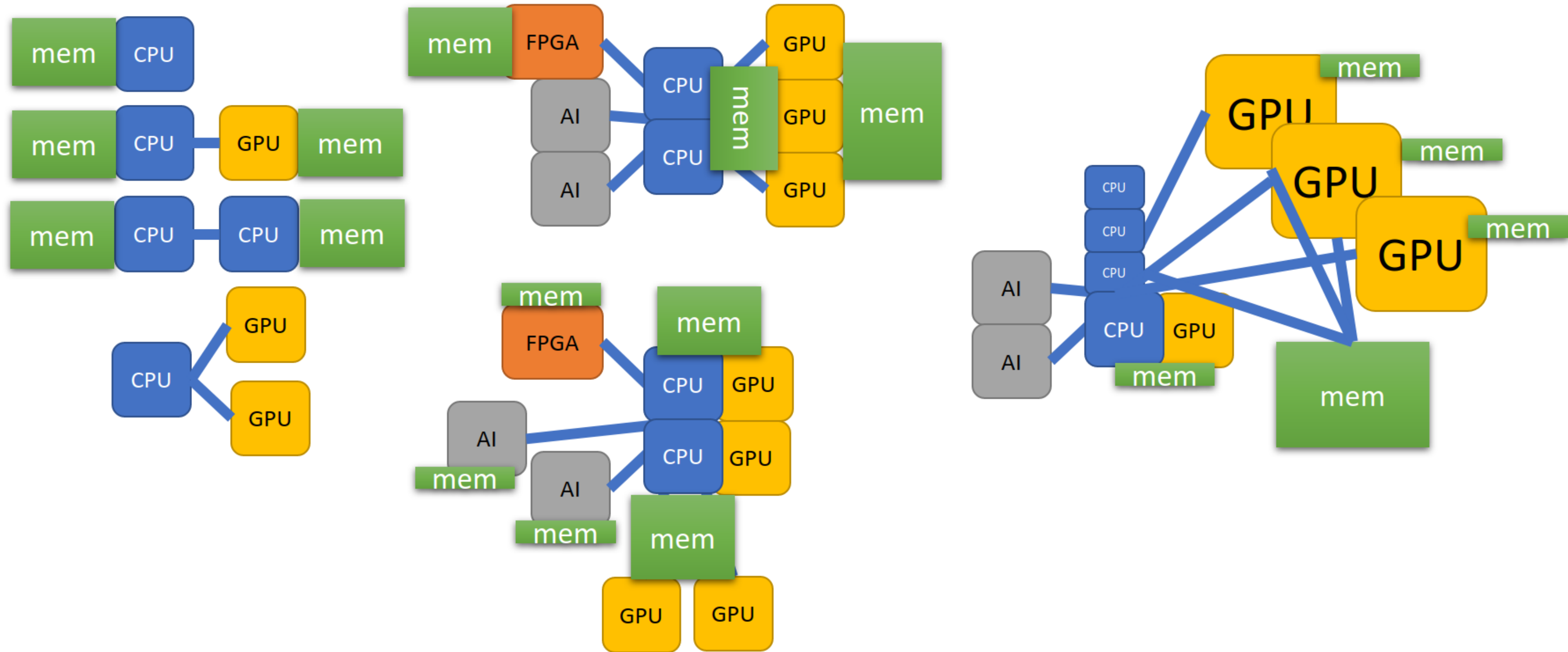
© 2023 Codeplay Software Ltd.

Memory Model

- Heterogeneous systems have complex memory architecture

© 2023 Codeplay Software Ltd.

Memory all over the place!



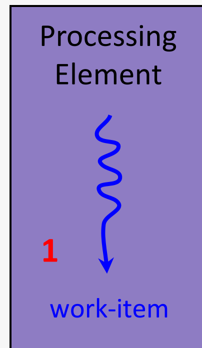
© 2023 Codeplay Software Ltd.

Memory Model

- Heterogeneous systems have complex memory architecture

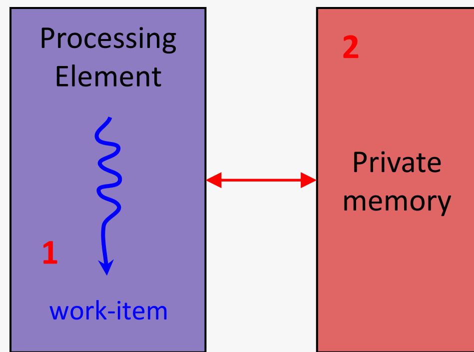
- SYCL makes sense of this by:
    - Defining memory hierarchy (C++ doesn't know about this!)

© 2023 Codeplay Software Ltd.



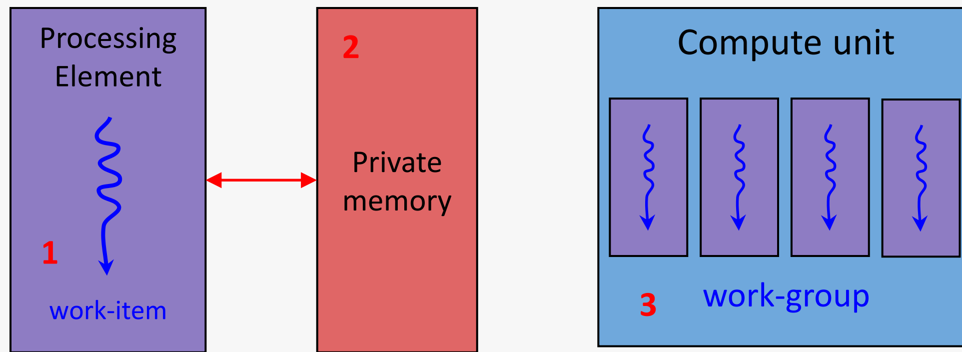
1. A processing element executes a single work-item

codeplay®
© 2019 Codeplay Software Ltd.



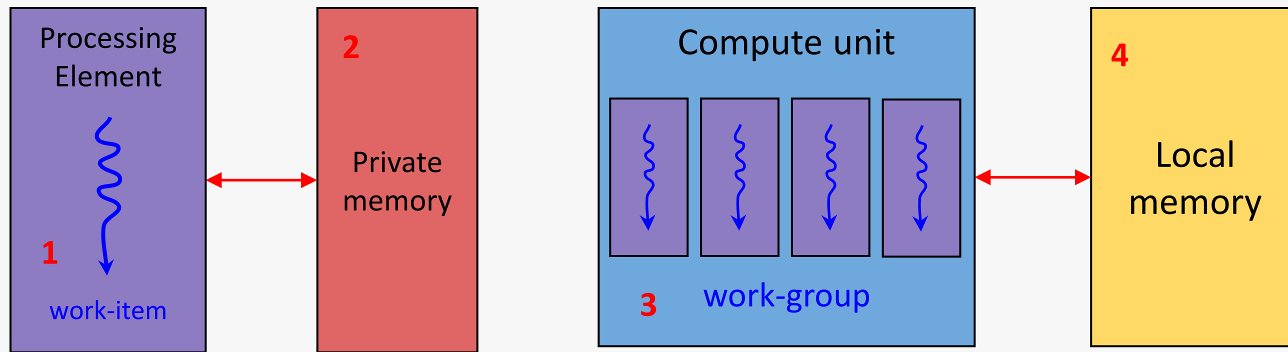
1. A processing element executes a single work-item
2. Each work-item can access private memory, a dedicated memory region for each processing element

© 2019 Codeplay Software Ltd.



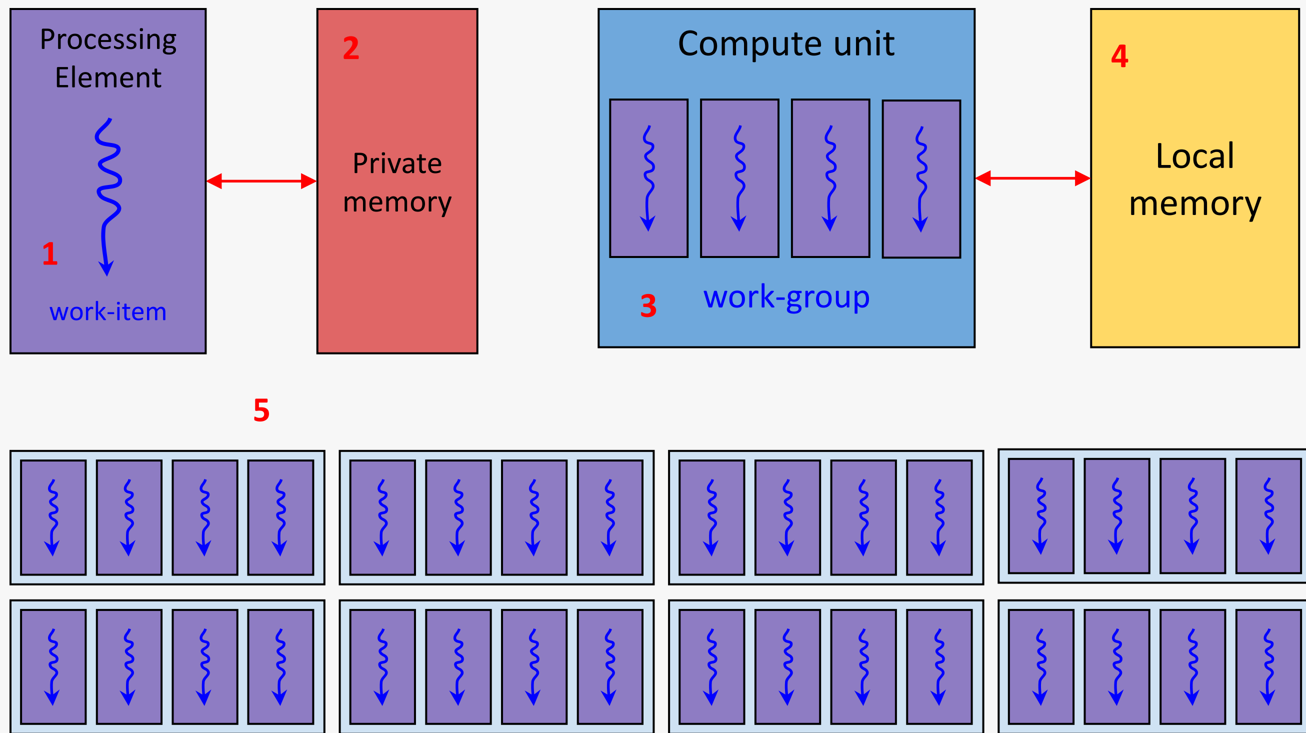
1. A processing element executes a single work-item
2. Each work-item can access private memory, a dedicated memory region for each processing element
3. A compute unit executes a work-group, composed of multiple work-items, one for each processing element in the compute unit

© 2019 Codeplay Software Ltd.



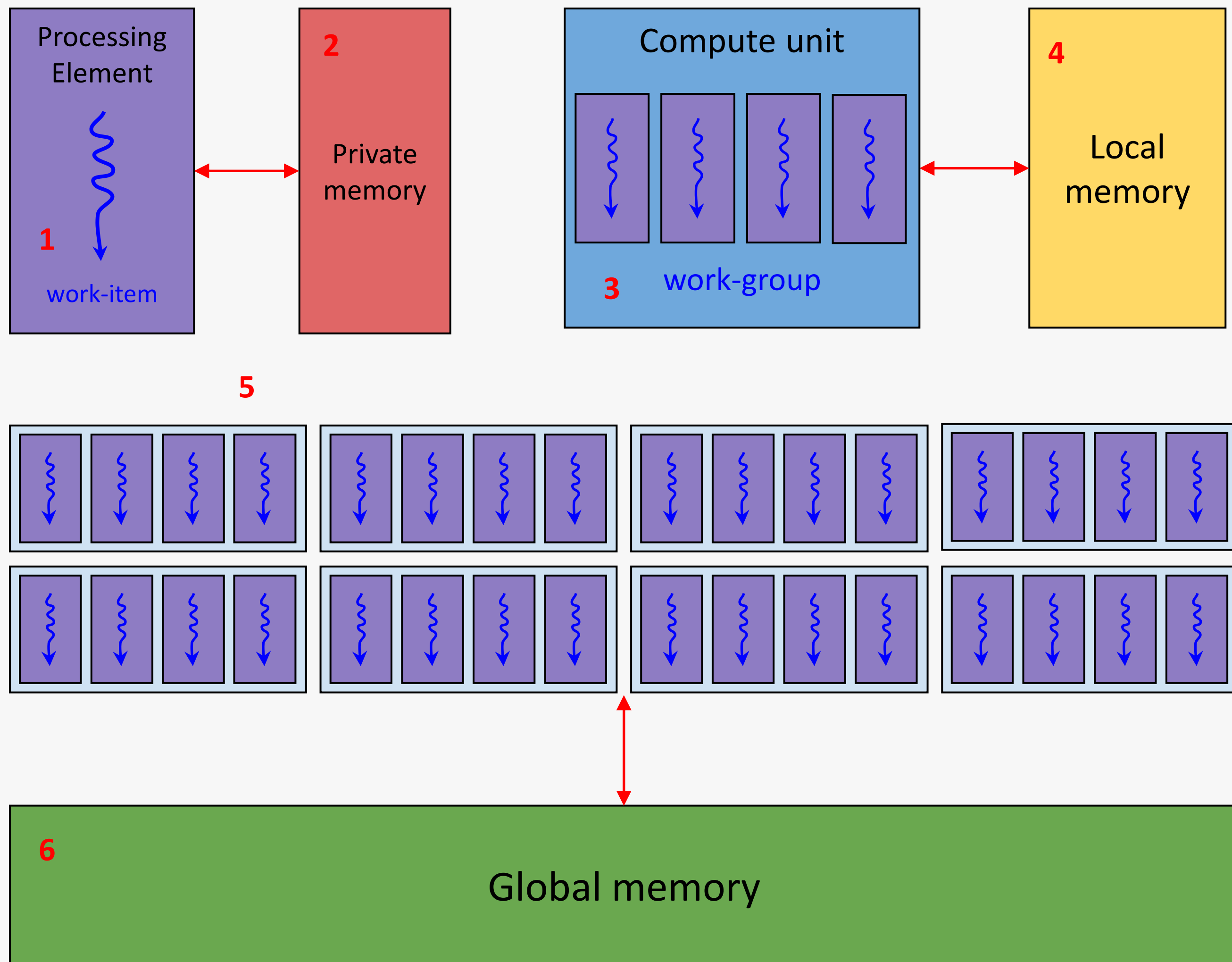
1. A processing element executes a single work-item
2. Each work-item can access private memory, a dedicated memory region for each processing element
3. A compute unit executes a work-group, composed of multiple work-items, one for each processing element in the compute unit
4. Each work-item can access local memory, a dedicated memory region for each compute unit

© 2019 Codeplay Software Ltd.



1. A processing element executes a single work-item
2. Each work-item can access private memory, a dedicated memory region for each processing element
3. A compute unit executes a work-group, composed of multiple work-items, one for each processing element in the compute unit
4. Each work-item can access local memory, a dedicated memory region for each compute unit
5. A device can execute multiple work-groups

© 2019 Codeplay Software Ltd.



1. A processing element executes a single work-item
2. Each work-item can access private memory, a dedicated memory region for each processing element
3. A compute unit executes a work-group, composed of multiple work-items, one for each processing element in the compute unit
4. Each work-item can access local memory, a dedicated memory region for each compute unit
5. A device can execute multiple work-groups
6. Each work-item can access global memory, a single memory region available to all processing elements

© 2019 Codeplay Software Ltd.

Memory Model

- Heterogeneous systems have complex memory architecture

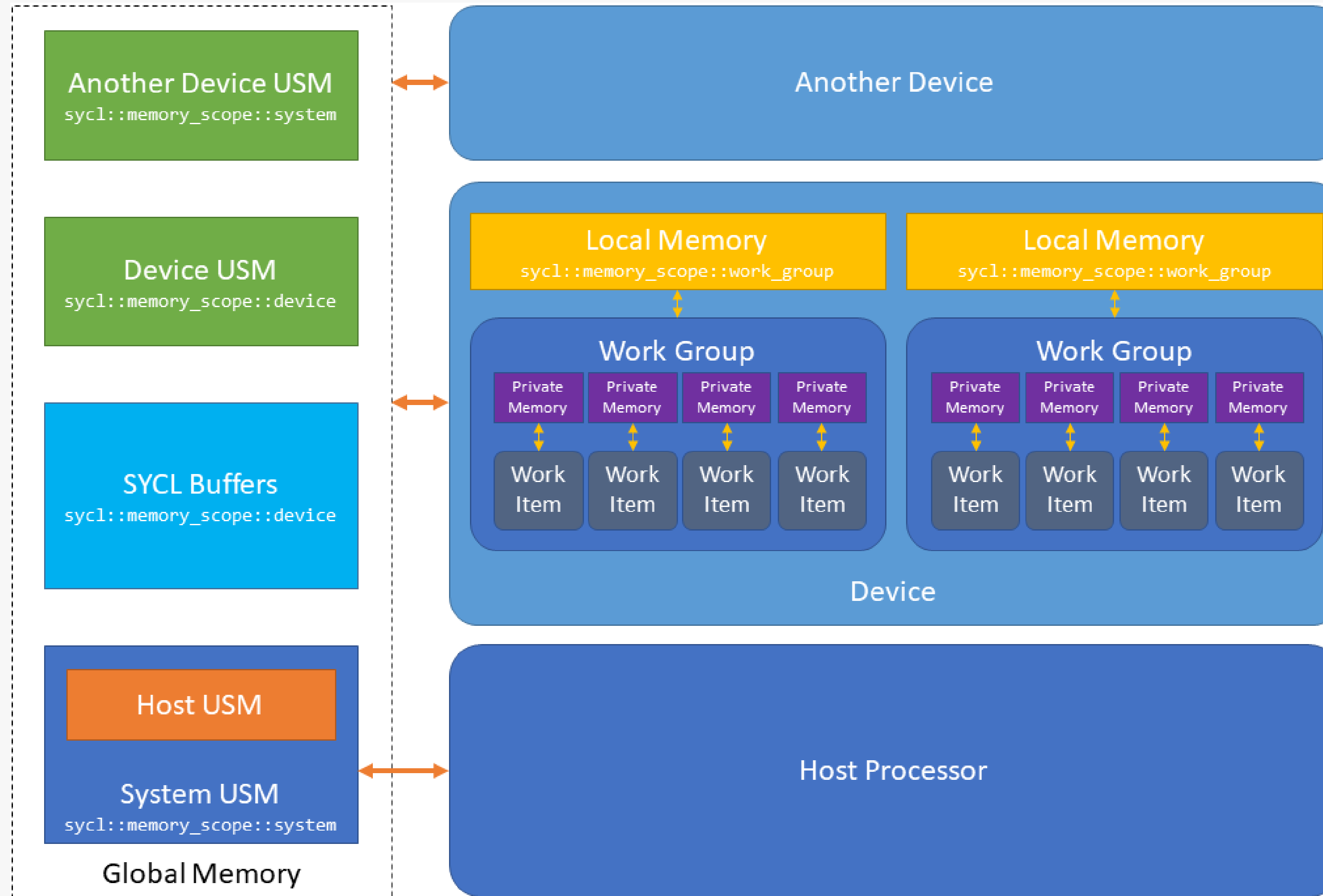
- SYCL makes sense of this by:

  - Defining memory hierarchy (C++ doesn't know about this!)

  - Defining the accessibility of memory from different devices

© 2023 Codeplay Software Ltd.

Memory all over the place!



© 2023 Codeplay Software Ltd.

Buffer/Accessor Model

- Alternative to 'raw pointer' memory management
- Abstraction around data
- Keeps track of which kernels are using data
- Data migration & kernel scheduling handled automatically!
- No possibility of unsafe concurrent access by different kernels

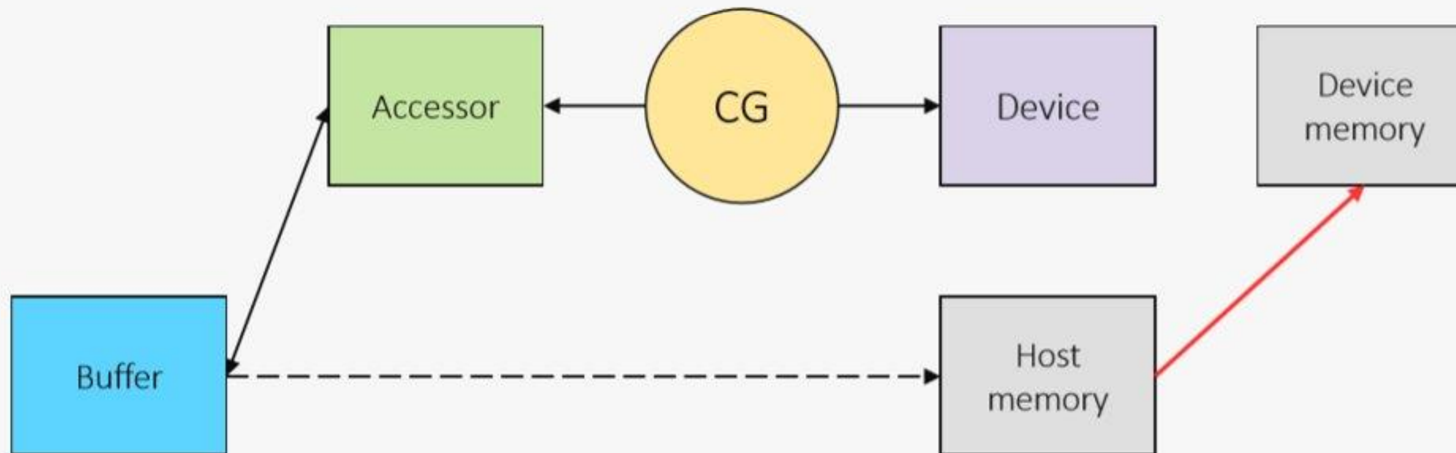
© 2023 Codeplay Software Ltd.

Buffers & Accessors

- Buffers
  - owning memory container
  - manage data migration and coherence across host and devices
  - users can specify "properties" to inform the runtime of extra information
- Accessors
  - give access to a buffer memory in a kernel
  - express data dependency of kernels

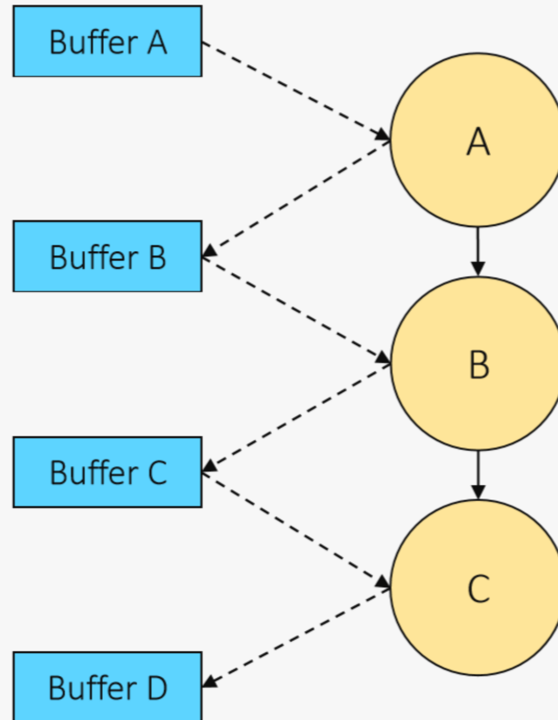
© 2023 Codeplay Software Ltd.

Buffers & Accessors



© 2023 Codeplay Software Ltd.

Buffers & Accessors



© 2023 Codeplay Software Ltd.

Communication & Synchronization

- Host-device synchronization
  - Queues provide host-device synchronization
  - Buffer destructor blocks until data safely back on host
- Thread-thread sync:
  - Threads can wait on each other & read/write shared memory
  - Hierarchical parallelism defines when this is possible
- SYCL defines synchronous & asynchronous exceptions
  - & the ability to provide custom exception handlers

© 2023 Codeplay Software Ltd.

Performance Portability

- SYCL targets embedded through to exascale computing
- Avoid writing different code for variants of your embedded system
- Hardware flexibility & future proofing

© 2023 Codeplay Software Ltd.

Aspects & Info

- Not all hardware supports *all* SYCL features
- How can we target these devices with SYCL?
- Using aspects:
  - aspect::atomic64
  - aspect::cpu
  - aspect::host_debuggable
- And device_info:
  - info::device::max_compute_units
  - info::device::preferred_vector_width_int
  - info::device::max_clock_frequency

© 2023 Codeplay Software Ltd.

SYCL for Embedded

- General purpose computing increasingly resource limited
  - Motivates the development of SYCL
  - Good news for the embedded world!

- SYCL:
  - defines a programming model for heterogeneous systems
  - formalises multiple memory spaces/scopes
  - provides powerful abstractions for data flow

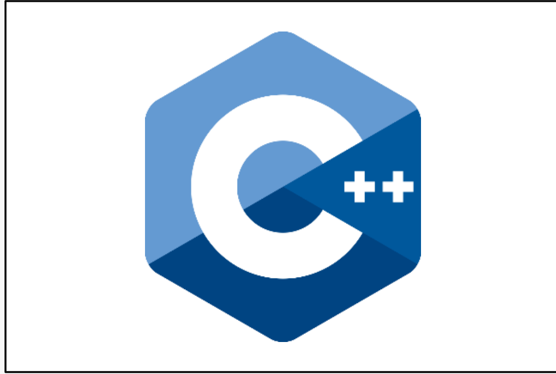
© 2023 Codeplay Software Ltd.

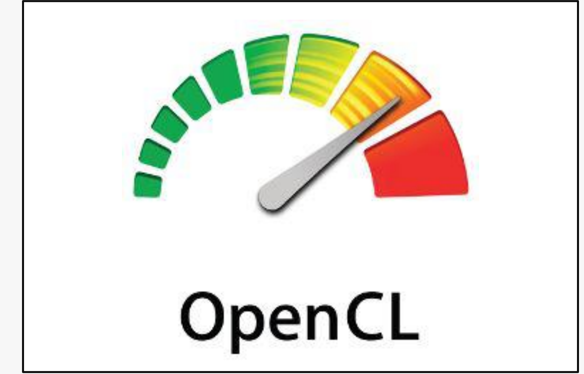
What is SYCL?

© 2023 Codeplay Software Ltd.

- Learning objectives:
  - Learn about the SYCL 2020 specification and its implementations
  - Learn about the major features that SYCL provides
  - Learn about the components of a SYCL implementation
  - Learn about the anatomy of a typical SYCL application
  - Learn where to find useful resources for SYCL

© 2023 Codeplay Software Ltd.





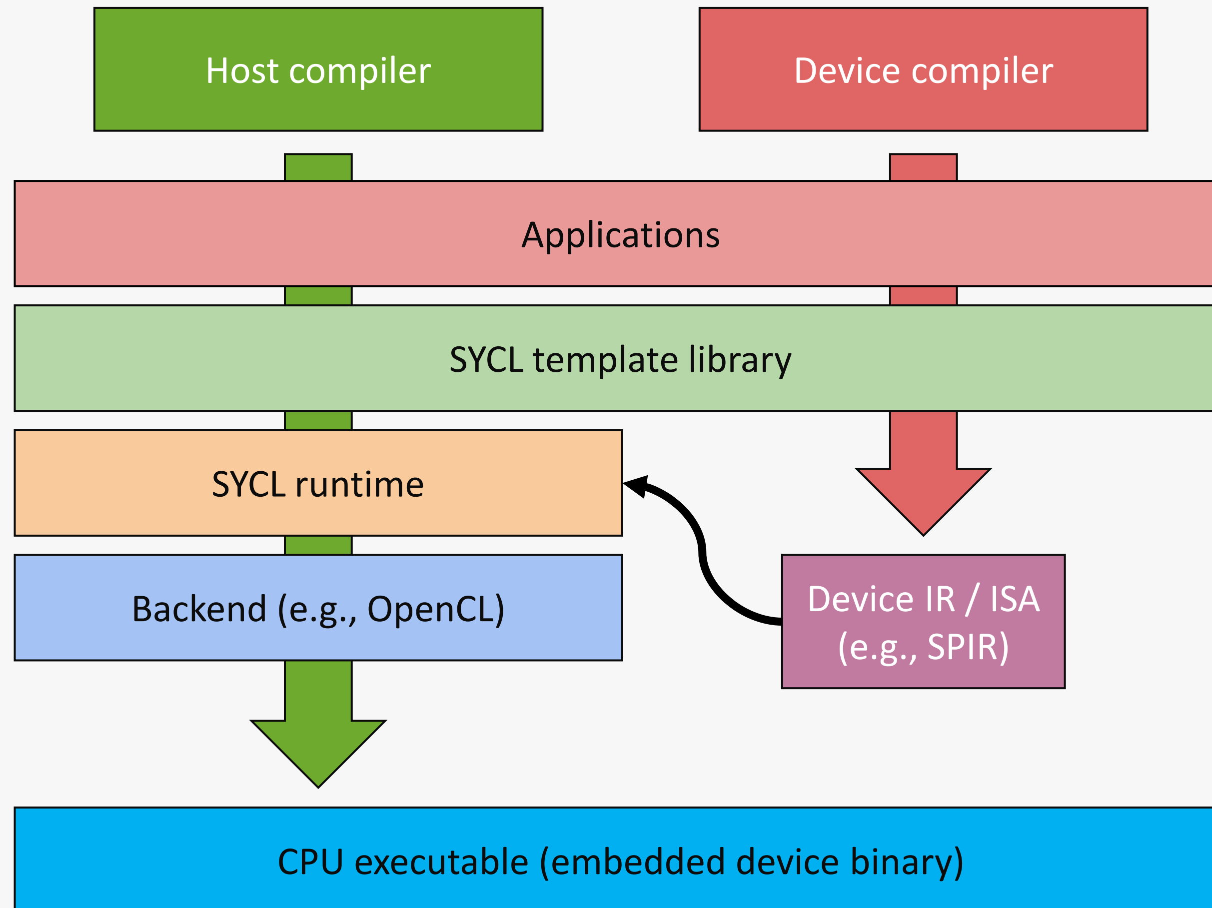
SYCL is a single-source, high-level, standard C++ programming model, that can target a range of heterogeneous platforms

© 2023 Codeplay Software Ltd.

SYCL is a single-source, high-level, standard C++ programming model, that can target a range of heterogeneous platforms

© 2023 Codeplay Software Ltd.

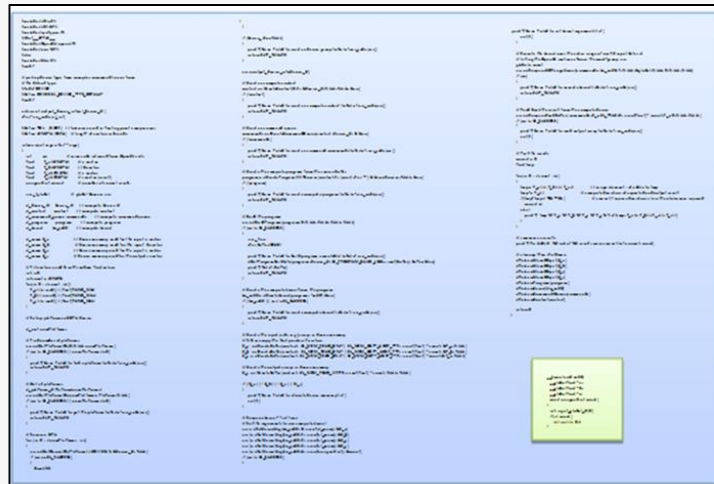
SYCL is a single-source, high-level, standard C++ programming model, that can target a range of heterogeneous platforms



- SYCL allows you write both host CPU and device code in the same C++ source file
  - This is usually implemented in two compilation passes; one for the host code and one for the device code

© 2023 Codeplay Software Ltd.

SYCL is a single-source, high-level, standard C++ programming model, that can target a range of heterogeneous platforms


Typical OpenCL hello world application

Typical SYCL hello world application

- SYCL provides high-level abstractions over common boiler-plate code
  - Platform/device selection
  - Buffer creation
  - Kernel compilation
  - Dependency management and scheduling

© 2023 Codeplay Software Ltd.

SYCL is a single-source, high-level, standard C++ programming model, that can target a range of heterogeneous platforms

```
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
}
```

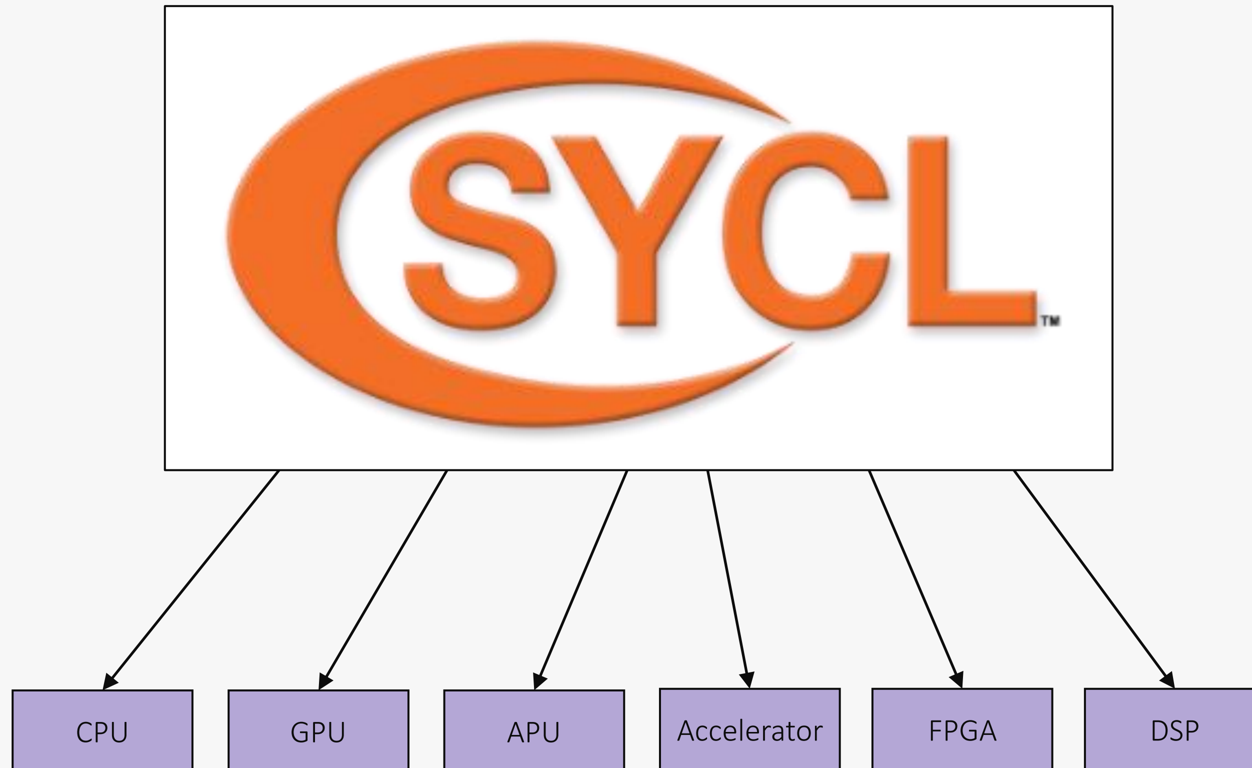
```
__global__ vec_add(float *a, float *b, float *c) {
  return c[i] = a[i] + b[i];
}

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

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

- SYCL allows you to write standard C++
  - No language extensions
  - No pragmas

© 2023 Codeplay Software Ltd.

SYCL is a single-source, high-level, standard C++ programming model, that can **target a range of heterogeneous platforms**



- SYCL can target any device supported by its backend
- SYCL can target a number of different backends

© 2023 Codeplay Software Ltd.

Who is implementing SYCL?

© 2023 Codeplay Software Ltd.

SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies

SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute

intel
oneAPI
DPC++
Uses LLVM/Clang
Part of oneAPI

Any CPU

OpenCL
SPIR
Intel CPUs
Intel GPUs
Intel FPGAs

Level Zero
Intel GPUs

NVIDIA CUDA
NVIDIA GPUs

ROCm
AMD GPUs

codeplay
ComputeCpp
ComputeCpp
Multiple
Backends

Any CPU

OpenCL
SPIR
Intel CPUs
Intel GPUs
Intel FPGAs
AMD GPUs
(depends on driver stack)
Arm Mali
IMG PowerVR
Renesas R-Car

Experimental
OpenCL NVIDIA
NVIDIA GPUs

hipSYCL
UNIVERSITÄT HEIDELBERG
hipSYCL
Multiple Backends

OpenMP
Any CPU

ROCm
AMD GPUs

Experimental
Level Zero
Intel GPUs

NVIDIA CUDA

© 2023 Codeplay Software Ltd.

SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies

SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute

Sylkan → Vulkan

Huawei SYCLops → MLIR (DPC++ fork)

Celerity SYCL → MPI

AMD

triSYCL Open source test bed → OpenMP; Any CPU; Experimental → OpenCL, SPIR, XILINX FPGAs, POCL (open-source OpenCL supporting CPUs and NVIDIA GPUs and more) pocl; DPC++ fork → TBB Any CPU; Experimental → XILINX Versal ACAP LLVM IR FPGA LLVM IR HLS

MotorSYCL → Nvidia HPC SDK

Inteon Poligeist SYCL → MLIR

TOHOKU

neoSYCL SX-AURORA TSUBASA → VEO (DPC++ fork) Intel CPUs NEC VEs

HUAWEI

SYCL/DPC++ Huawei Ascend AI → AI (DPC++ fork) Ascend 910 AI processor

Multiple Backends in Development

SYCL on even more low-level frameworks.

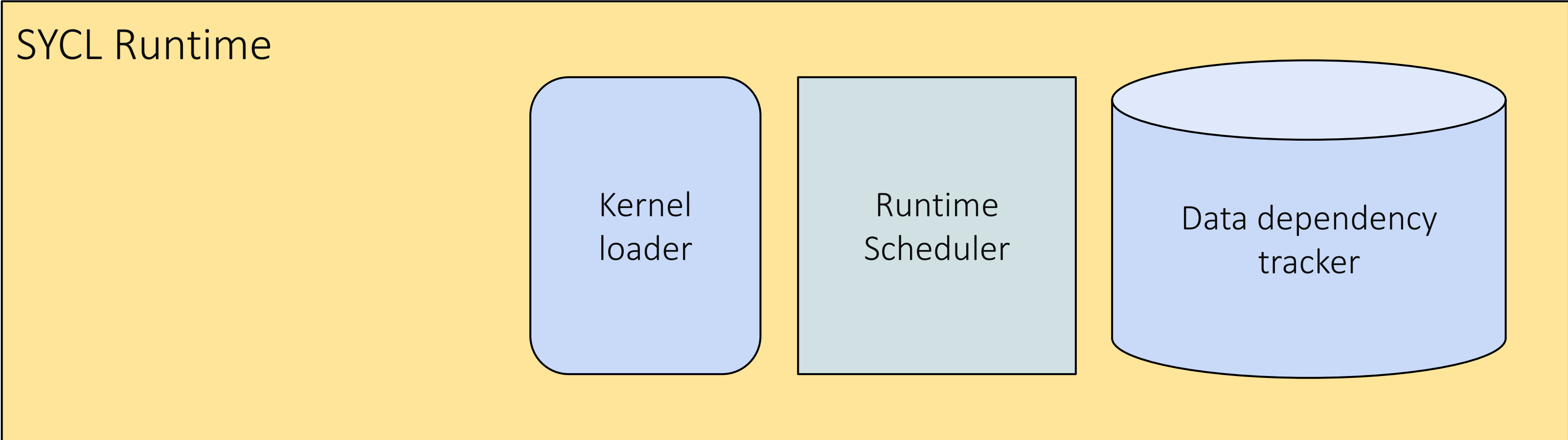
For more information: http://sycl.tech

© 2023 Codeplay Software Ltd.

What is in a SYCL implementation?

© 2023 Codeplay Software Ltd.

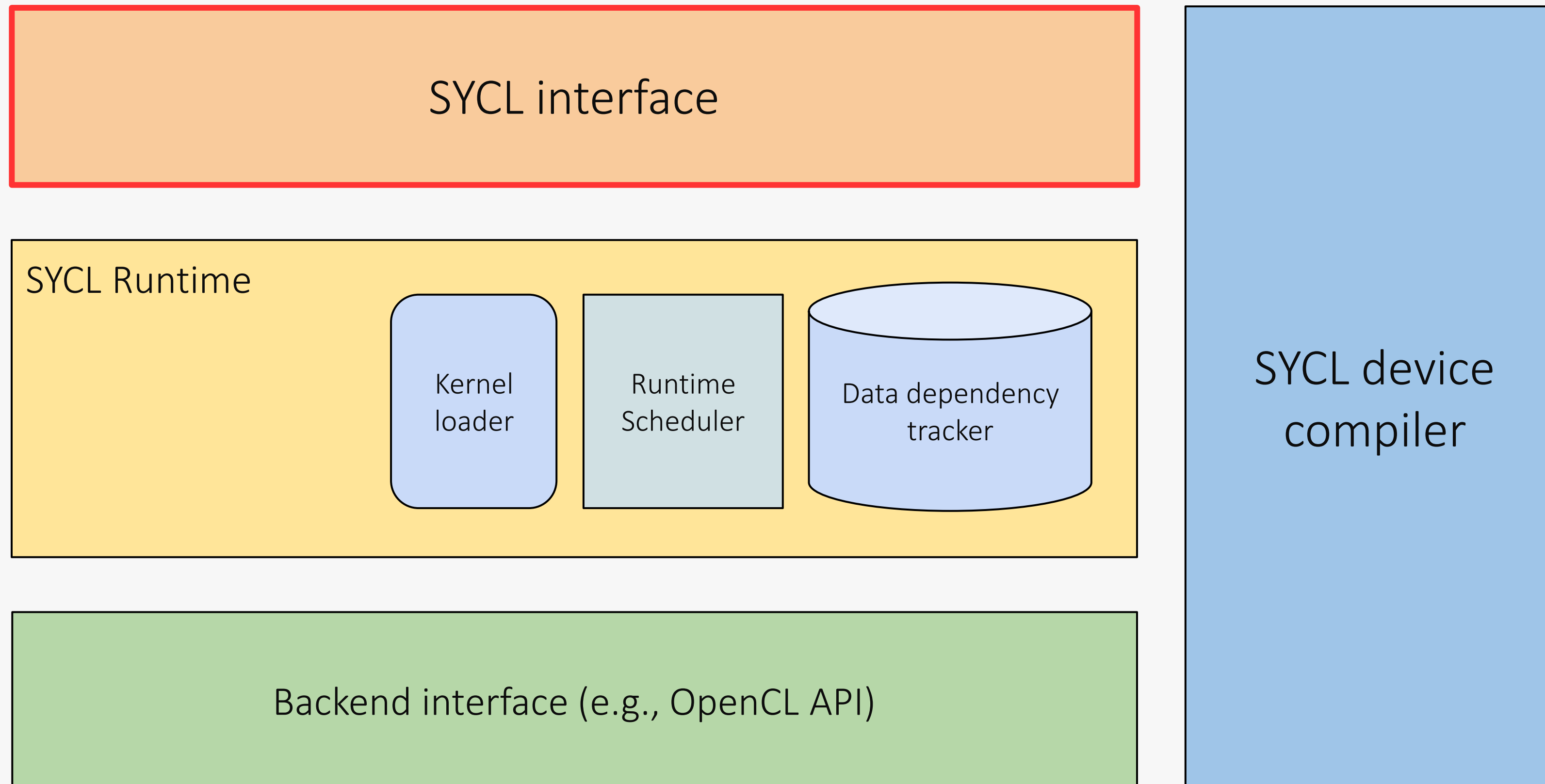
SYCL interface



SYCL device compiler

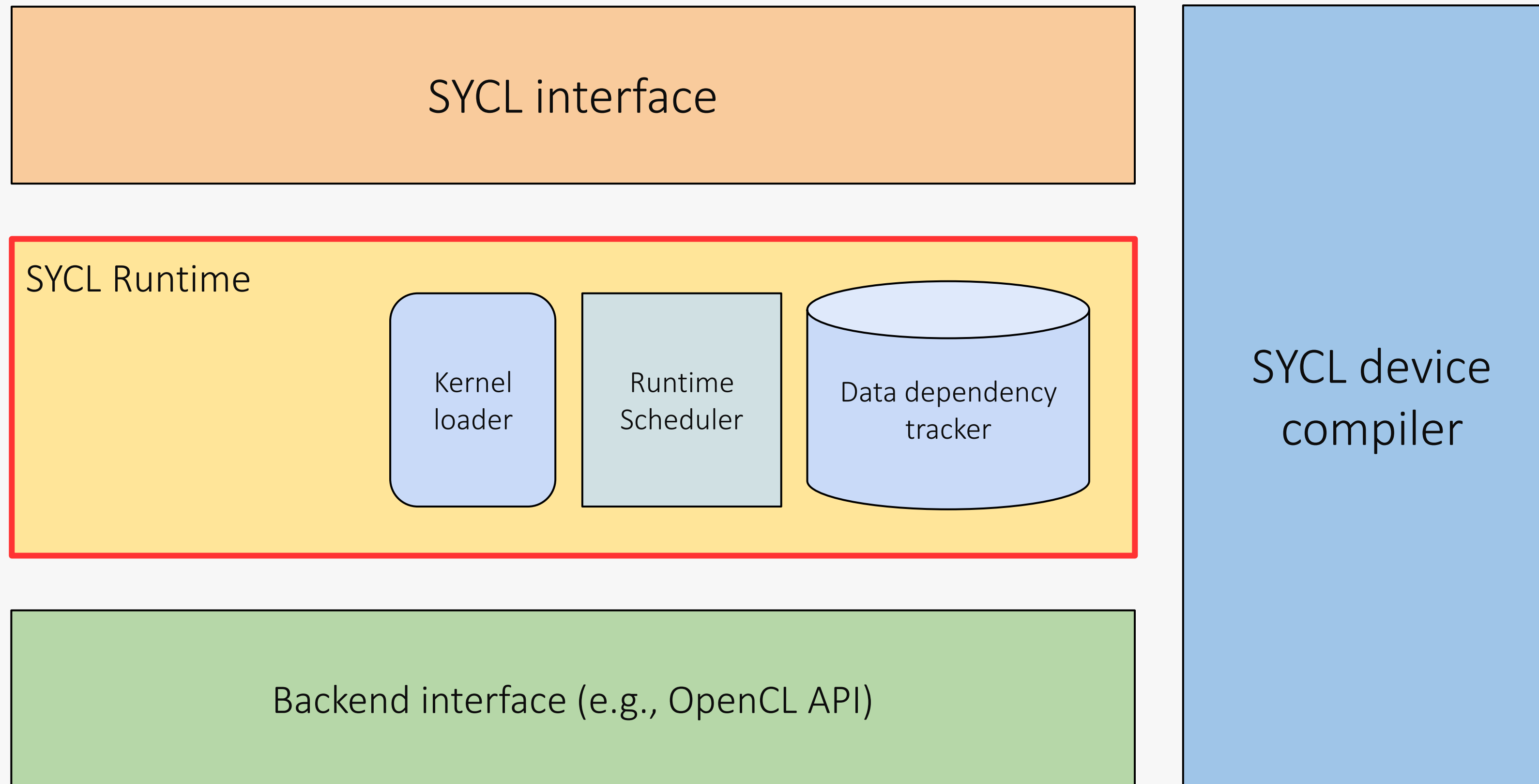
Backend interface (e.g. OpenCL API)

© 2023 Codeplay Software Ltd.



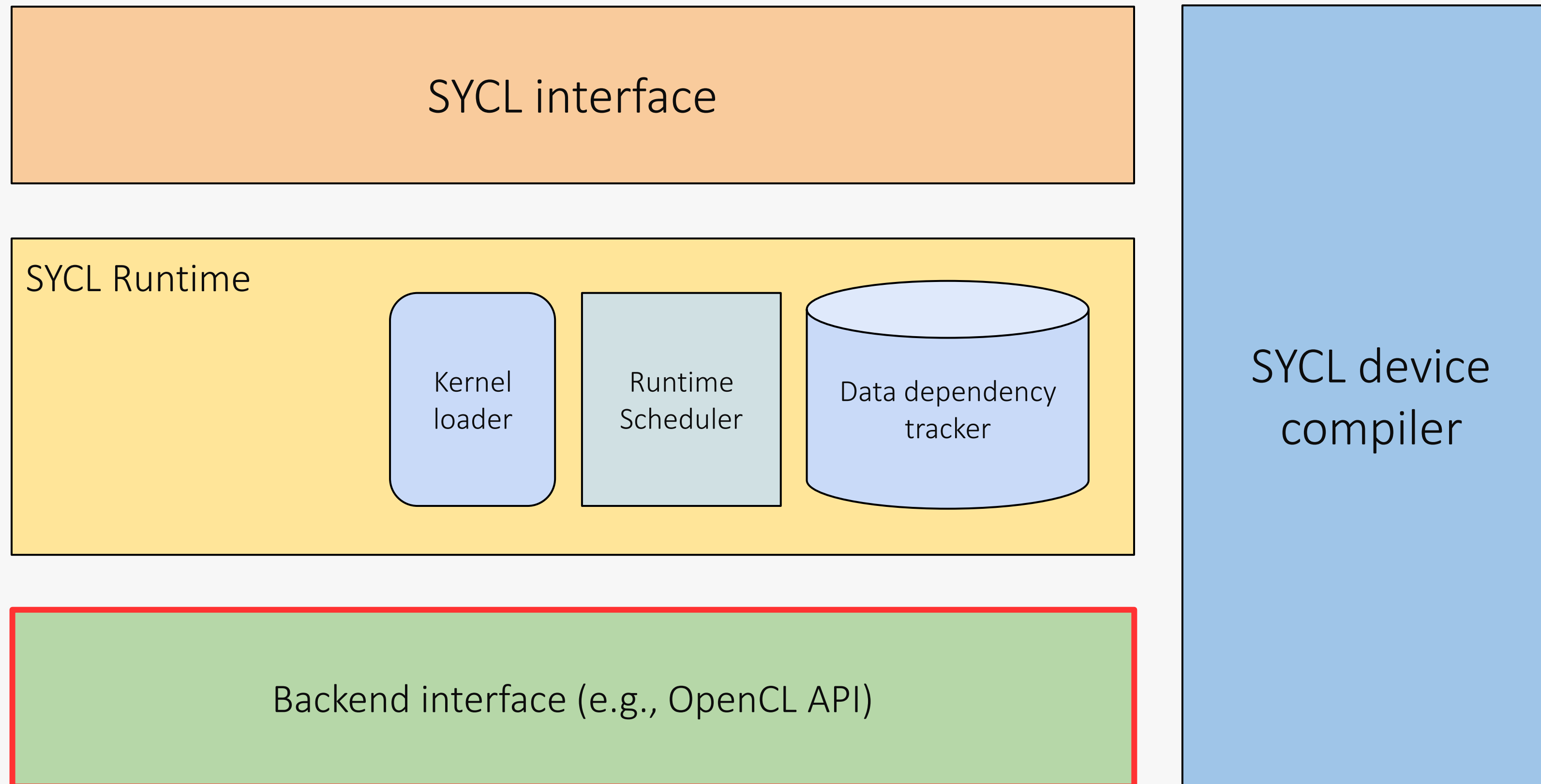
- The SYCL interface is a C++ template library that users and library developers program to
  - The same interface is used for both the host and device code

© 2023 Codeplay Software Ltd.



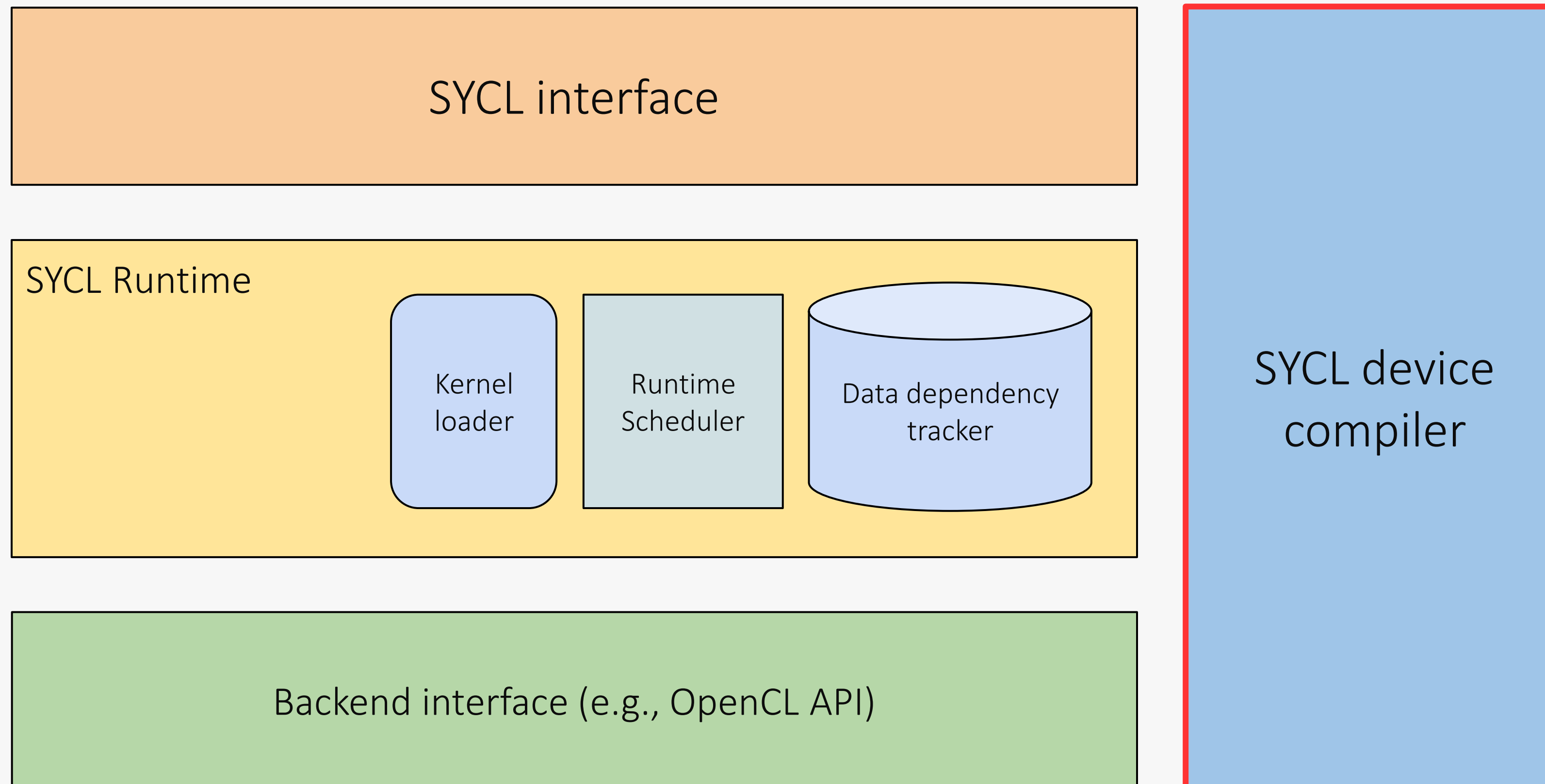
- The SYCL runtime is a library that schedules and executes work
  - It loads kernels, tracks data dependencies and schedules commands

© 2023 Codeplay Software Ltd.



- The backend interface is where the SYCL runtime calls down into a backend in order to execute on a particular device
  - The standard backend is OpenCL but some implementations have supported others

© 2023 Codeplay Software Ltd.



- The SYCL device compiler is a C++ compiler which can identify SYCL kernels and compile them down to an IR or ISA
  - This can be SPIR, SPIR-V, GCN, PTX or any proprietary vendor ISA

© 2023 Codeplay Software Ltd.

What does a SYCL application look like?

© 2023 Codeplay Software Ltd.

```
int main(int argc, char *argv[]) {

}
```


© 2023 Codeplay Software Ltd.

```
#include <sycl/sycl.hpp>
using namespace sycl;


int main(int argc, char *argv[]) {

}
```

First we include the SYCL header which contains the runtime API

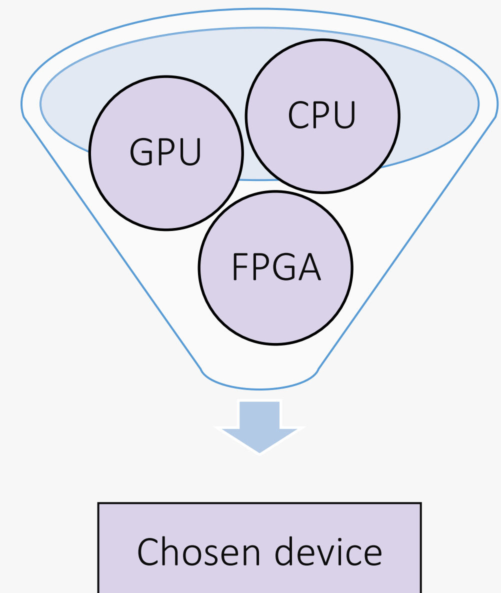
We also import the sycl namespace here for the purposes of presenting

© 2023 Codeplay Software Ltd.

```
#include <sycl/sycl.hpp>
using namespace sycl;


int main(int argc, char *argv[]) {



  queue q{default_selector_v};
















}
```

Device selectors allow you to choose a device based on a custom configuration

The queue default constructor uses the default_selector_v, which allows the runtime to select a device for you



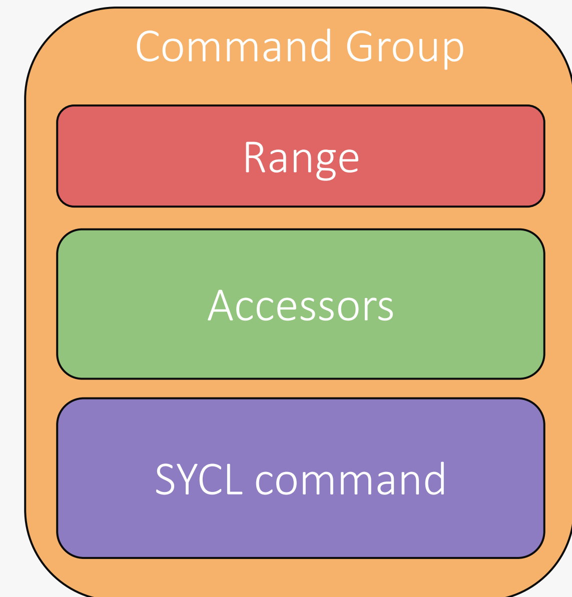
© 2023 Codeplay Software Ltd.

```
#include <sycl/sycl.hpp>
using namespace sycl;

int main(int argc, char *argv[]) {

  queue q{default_selector_v};

  q.submit([&](handler &cgh){

  });

}
```

With a queue we can submit a command group

A command group contains:

- A SYCL command (e.g. a SYCL kernel function)
- Execution range
- Accessors



© 2023 Codeplay Software Ltd.

```
#include <sycl/sycl.hpp>
using namespace sycl;


int main(int argc, char *argv[]) {
  std::vector<float> dA{ … }, dB{ … }, dO{ … };


  queue q{default_selector_v};





  q.submit([&](handler &cgh){







  });


}
```

We initialize three vectors, two inputs and an output

© 2023 Codeplay Software Ltd.

```
#include <sycl/sycl.hpp>
using namespace sycl;


int main(int argc, char *argv[]) {
  std::vector<float> dA{ … }, dB{ … }, dO{ … };


  queue q{default_selector_v};

  buffer<float> bufA{dA};
  buffer<float> bufB{dB};
  buffer<float> bufO{dO};

  q.submit([&](handler &cgh){





  });


}
```

We create a buffer for each vector to manage the data across host and device

© 2023 Codeplay Software Ltd.

```
#include <sycl/sycl.hpp>
using namespace sycl;


int main(int argc, char *argv[]) {
  std::vector<float> dA{ … }, dB{ … }, dO{ … };


  queue q{default_selector_v};
  {
    buffer<float> bufA{dA};
    buffer<float> bufB{dB};
    buffer<float> bufO{dO};

    q.submit([&](handler &cgh){






    });
  }

}
```

Buffers synchronize on destruction via RAII

So adding this scope means that all kernels writing to the buffers will be waited on and the data will be copied back to the vectors on leaving this scope

© 2023 Codeplay Software Ltd.

```
#include <sycl/sycl.hpp>
using namespace sycl;


int main(int argc, char *argv[]) {
  std::vector<float> dA{ … }, dB{ … }, dO{ … };


  queue q{default_selector_v};
  {
    buffer<float> bufA{dA};
    buffer<float> bufB{dB};
    buffer<float> bufO{dO};

    q.submit([&](handler &cgh){

      accessor inA{bufA, cgh, read_only};
      accessor inB{bufB, cgh, read_only};
      accessor out{bufO, cgh, write_only, no_init};



    });
  }

}
```

We create an accessor for each of the buffers

Read access for the two input buffers and write access for the output buffer. An additional property is passed to the output accessor to specify that the previous data will not be used.

© 2023 Codeplay Software Ltd.

```
#include <sycl/sycl.hpp>
using namespace sycl;
class add;

int main(int argc, char *argv[]) {
  std::vector<float> dA{ … }, dB{ … }, dO{ … };


  queue q{default_selector_v};
  {
    buffer<float> bufA{dA};
    buffer<float> bufB{dB};
    buffer<float> bufO{dO};

    q.submit([&](handler &cgh){

      accessor inA{bufA, cgh, read_only};
      accessor inB{bufB, cgh, read_only};
      accessor out{bufO, cgh, write_only, no_init};

      cgh.parallel_for<add>(dA.size(),
          [=](id<1> i) { out[i] = inA[i] + inB[i]; });
    });
  }

}
```

We define a SYCL kernel function for the command group using the parallel_for API

The first argument here is cast to a range, specifying the iteration space

The second argument is a lambda function that represents the entry point for the SYCL kernel

This lambda takes an id parameter that describes the current iteration being executed

© 2023 Codeplay Software Ltd.

```
#include <sycl/sycl.hpp>
using namespace sycl;
class add;

int main(int argc, char *argv[]) {
  std::vector<float> dA{ … }, dB{ … }, dO{ … };


  queue q{default_selector_v};
  {
    buffer<float> bufA{dA};
    buffer<float> bufB{dB};
    buffer<float> bufO{dO};

    q.submit([&](handler &cgh){

      accessor inA{bufA, cgh, read_only};
      accessor inB{bufB, cgh, read_only};
      accessor out{bufO, cgh, write_only, no_init};

      cgh.parallel_for<add>(dA.size(),
          [=](id<1> i) { out[i] = inA[i] + inB[i]; });
    });
  }

}
```

The template parameter to parallel_for is used to name the lambda

The reason for this is that C++ does not have a standard ABI for lambdas so they are represented differently across the host and device compiler

SYCL kernel functions follow C++ ODR rules, which means that if a SYCL kernel is in a template context, the kernel name needs to reflect that context, so must contain the same template arguments

© 2023 Codeplay Software Ltd.

```cpp
#include <sycl/sycl.hpp>
using namespace sycl;
class add;

int main(int argc, char *argv[]) {
  std::vector<float> dA{ … }, dB{ … }, dO{ … };

  try {
    queue q{default_selector_v, async_handler{}};
    {
      buffer<float> bufA{dA};
      buffer<float> bufB{dB};
      buffer<float> bufO{dO};

      q.submit([&](handler &cgh){

        accessor inA{bufA, cgh, read_only};
        accessor inB{bufB, cgh, read_only};
        accessor out{bufO, cgh, write_only, no_init};

        cgh.parallel_for<add>(dA.size(),
            [=](id<1> i) { out[i] = inA[i] + inB[i]; });
      });
    }
    q.throw_asynchronous();
  } catch (...) { /* handle errors */ }
}
```

In SYCL errors are handled using exception handling, so you should always wrap SYCL code in a try-catch block

Some exceptions are thrown synchronously at the point of using a SYCL API

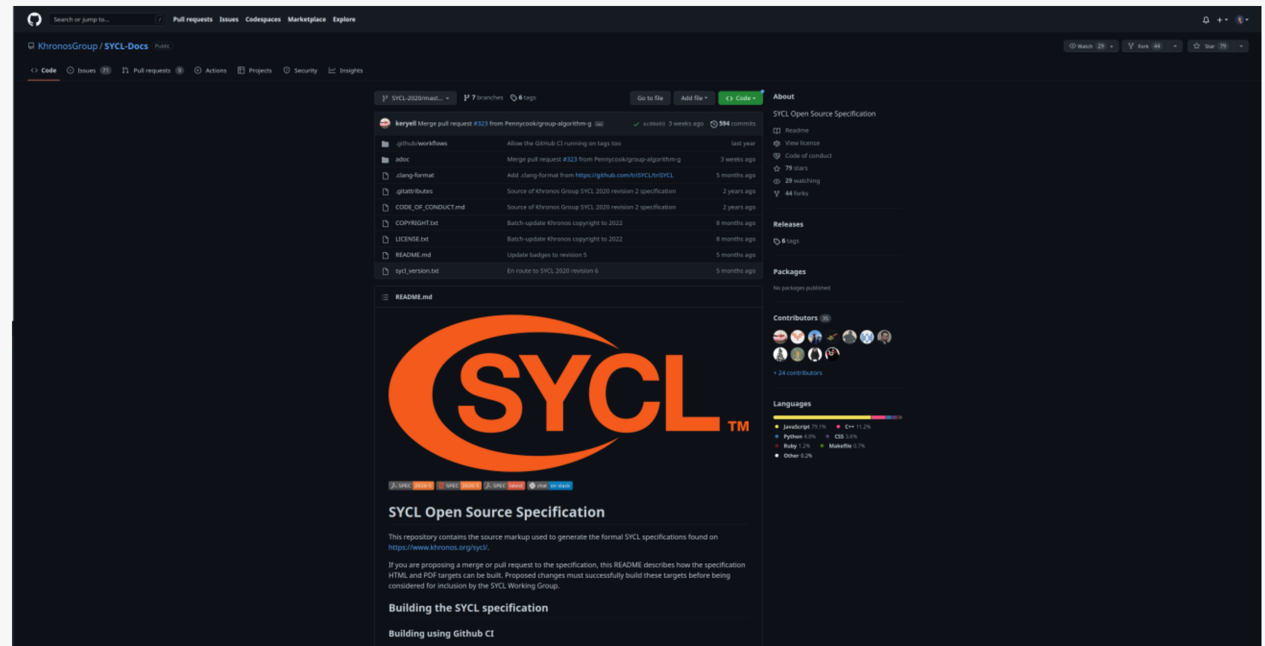
Other exceptions are asynchronous and are stored by the runtime and passed to an **async handler** when the queue is told to throw

© 2023 Codeplay Software Ltd.

Useful SYCL resources

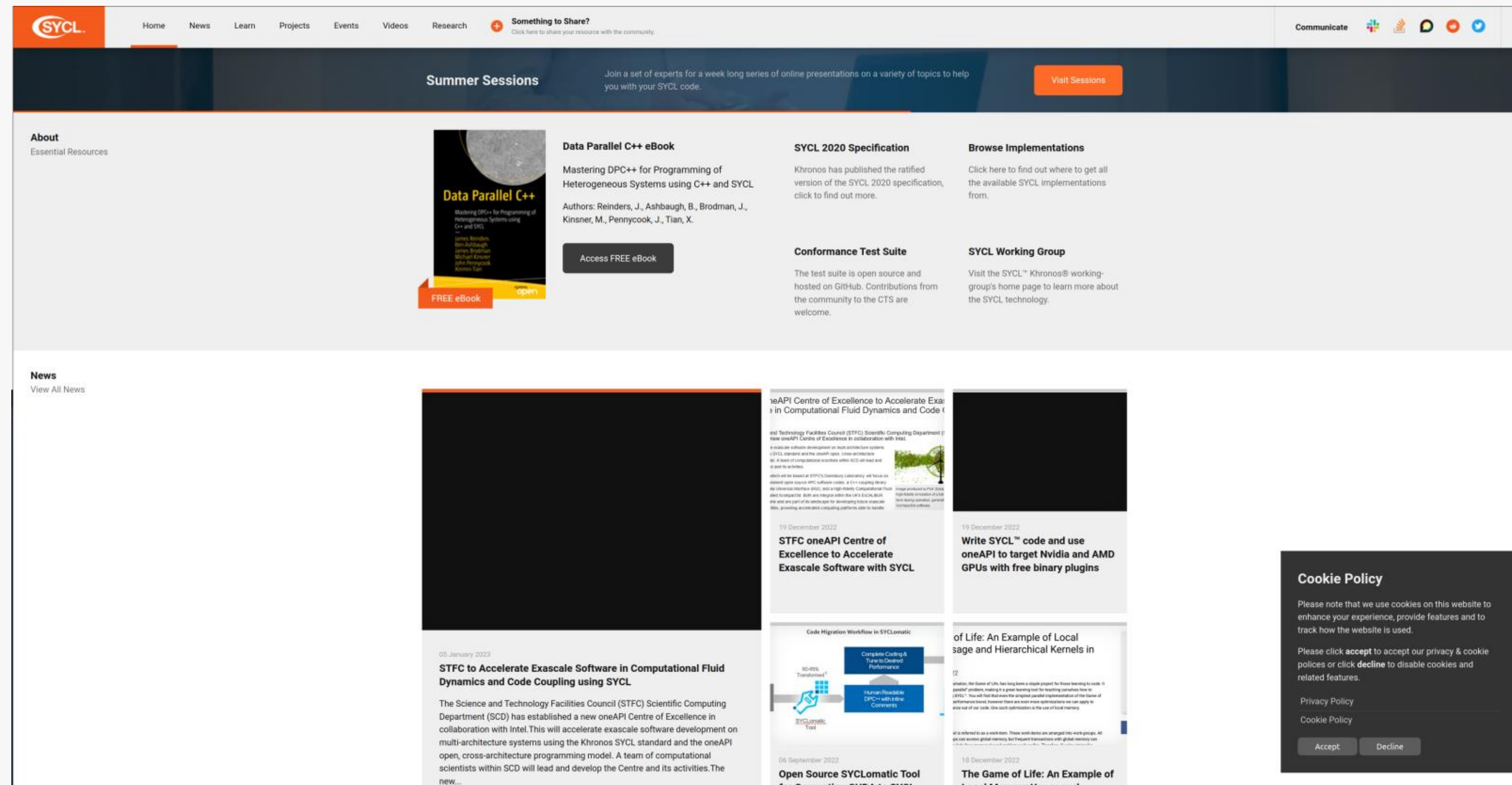
© 2023 Codeplay Software Ltd.

- The latest SYCL specification is SYCL 2020
  - Available at:
    - https://registry.khronos.org/SYCL/specs/sycl-2020/html/sycl-2020.html
- The specification is open source
  - Github project: https://github.com/KhronosGroup/SYCL-Docs



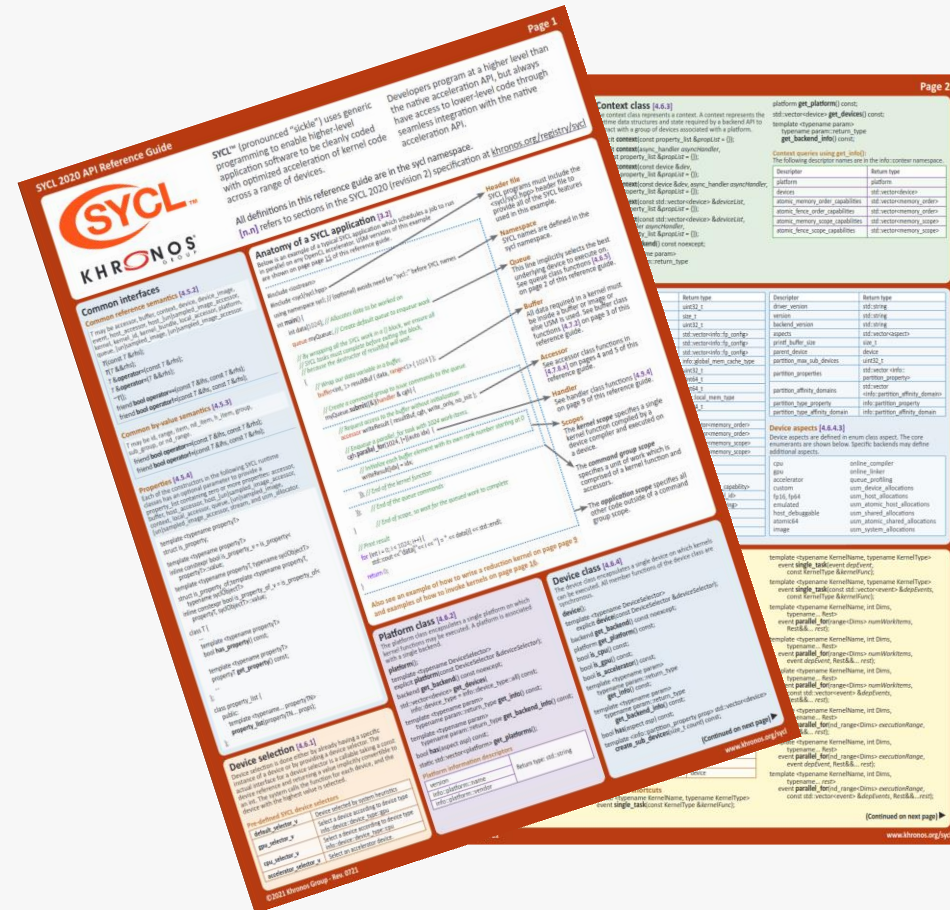
© 2023 Codeplay Software Ltd.

- There is a Khronos backed website for collecting SYCL related news and articles
  - Available at: http://sycl.tech/



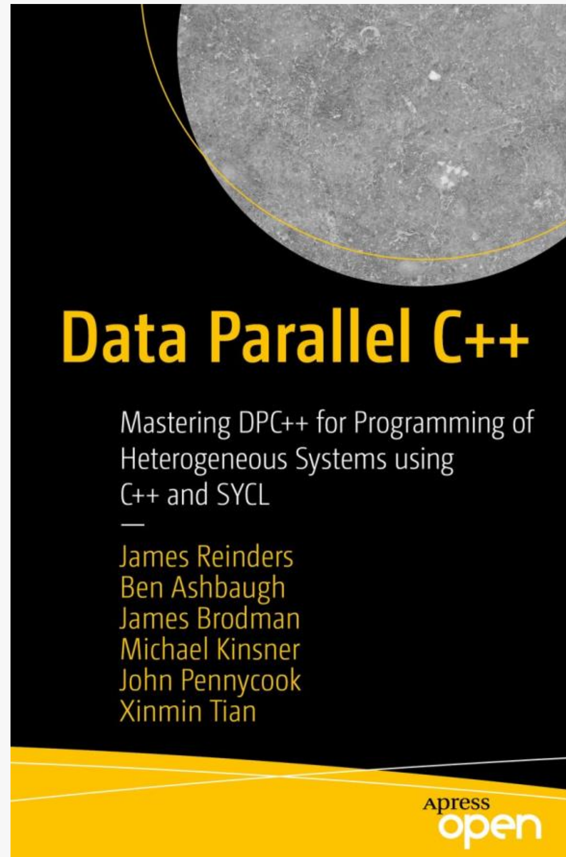
© 2023 Codeplay Software Ltd.

- There are Khronos produced SYCL 2020 reference cards
  - Available at: https://www.khronos.org/files/sycl/sycl-2020-reference-guide.pdf



© 2023 Codeplay Software Ltd.

- The free Data Parallel C++ book:
  - Available at: https://link.springer.com/book/10.1007/978-1-4842-5574-2



© 2023 Codeplay Software Ltd.

Embedded SYCL today

© 2023 Codeplay Software Ltd.

SYCL Tools for Embedded

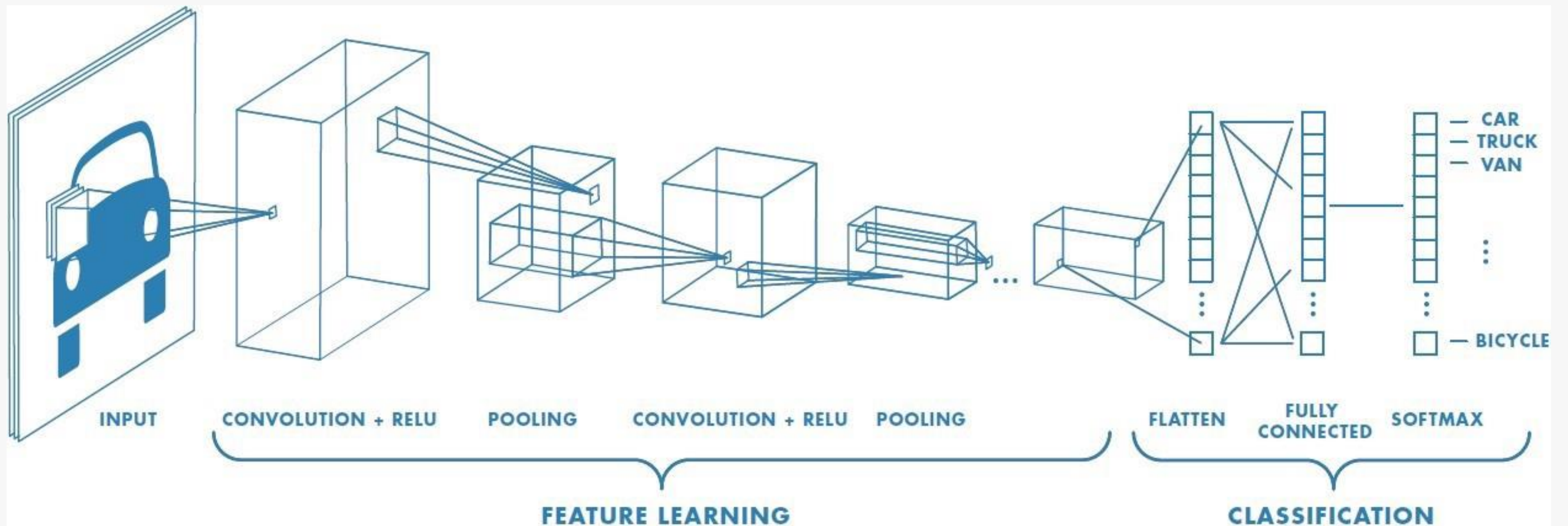
- ComputeCPP/ComputeAorta
  - Any CPU or CPU-like processor
  - Renesas R-Car, IMG PowerVR, ARM Mali, Xilinx FPGA (embedded)
  - Embedded platforms supporting OpenCL/SPIR/SPIR-V (ComputeAorta can provide this support).
  - Some undisclosed customer embedded platforms
- DPC++
  - Emerging support for MLIR
  - Huawei Ascent processor (for Autonomous Driving)
- TriSYCL
  - Xilinx FPGAs (including embedded)
- Sylkan
  - SYCL on top of Vulkan

© 2023 Codeplay Software Ltd.

Current SYCL Applications for Embedded

- Automotive
  - Autonomous driving & ADAS,
  - sensor fusion (fusing lidar and radar data)
  - battery management systems
- Autonomous Unmanned Arial Vehicles
  - Improve automatic detection and avoidance capabilities of drones.
- Medical imaging
  - E.g. skin cancer detection using mobile medical devices/instruments

© 2023 Codeplay Software Ltd.



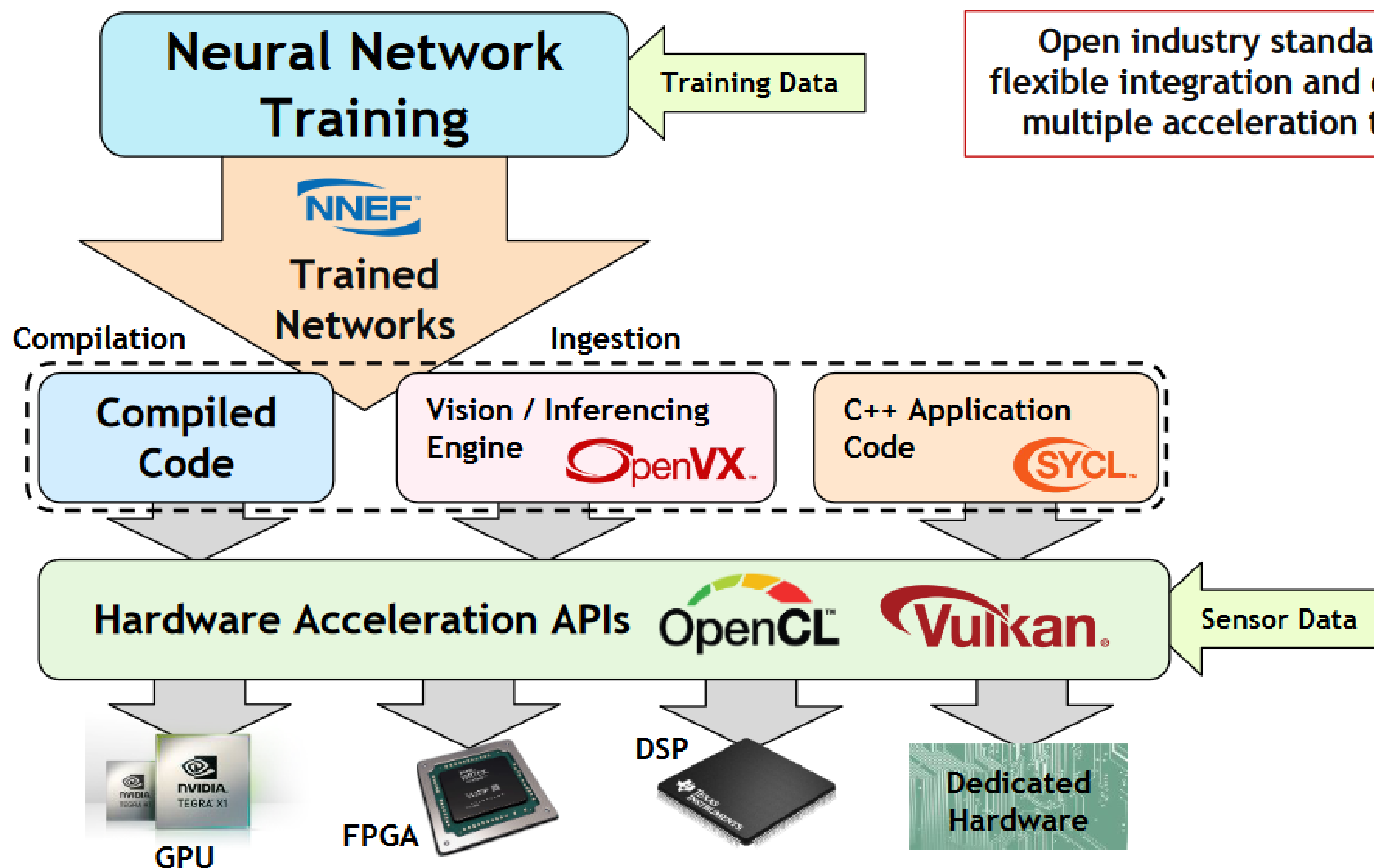
Copyright Codeplay Software Ltd 2021

SYCL in Embedded Systems, Automotive, and AI

Networks trained on high-end desktop and cloud systems

Applications link to compiled inferencing code or call vision/inferencing API

Diverse Embedded Hardware
Multi-core CPUs, GPUs
DSPs, FPGAs, Tensor Cores
* Vulkan only runs on GPUs



This work is licensed under a Creative Commons Attribution 4.0 International License

Technologies for Automated Driving

- Renesas R-Car architecture
  - Embedded automotive platform
  - Optimized for Computer Vision and Machine Learning
  - Designed for
    - Low latency
    - Low power consumption
    - Low cost

Copyright Codeplay Software Ltd 2021

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.

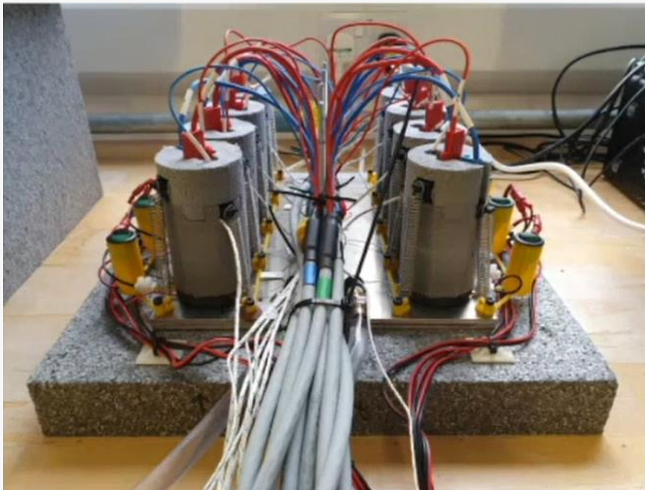
Project consortium:

WILLIAMS ADVANCED ENGINEERING

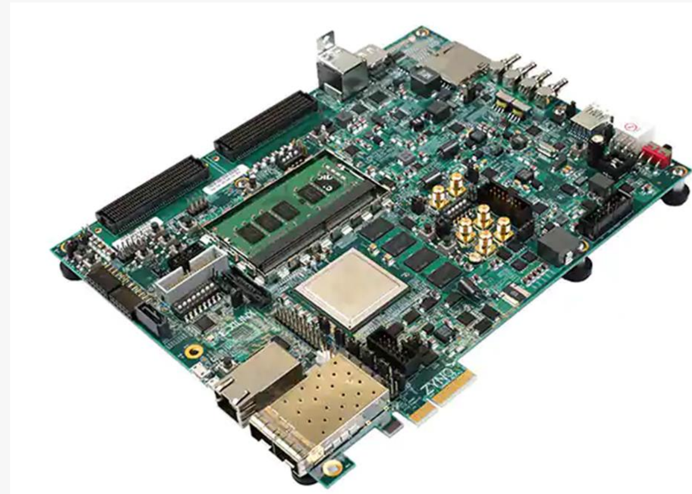
Imperial College London

codeplay

Silver Power Systems



Experimental Battery Test rig at Imperial.



Embedded MPSoC platform running the BMS on the Battery.

© 2023 Codeplay Software Ltd.

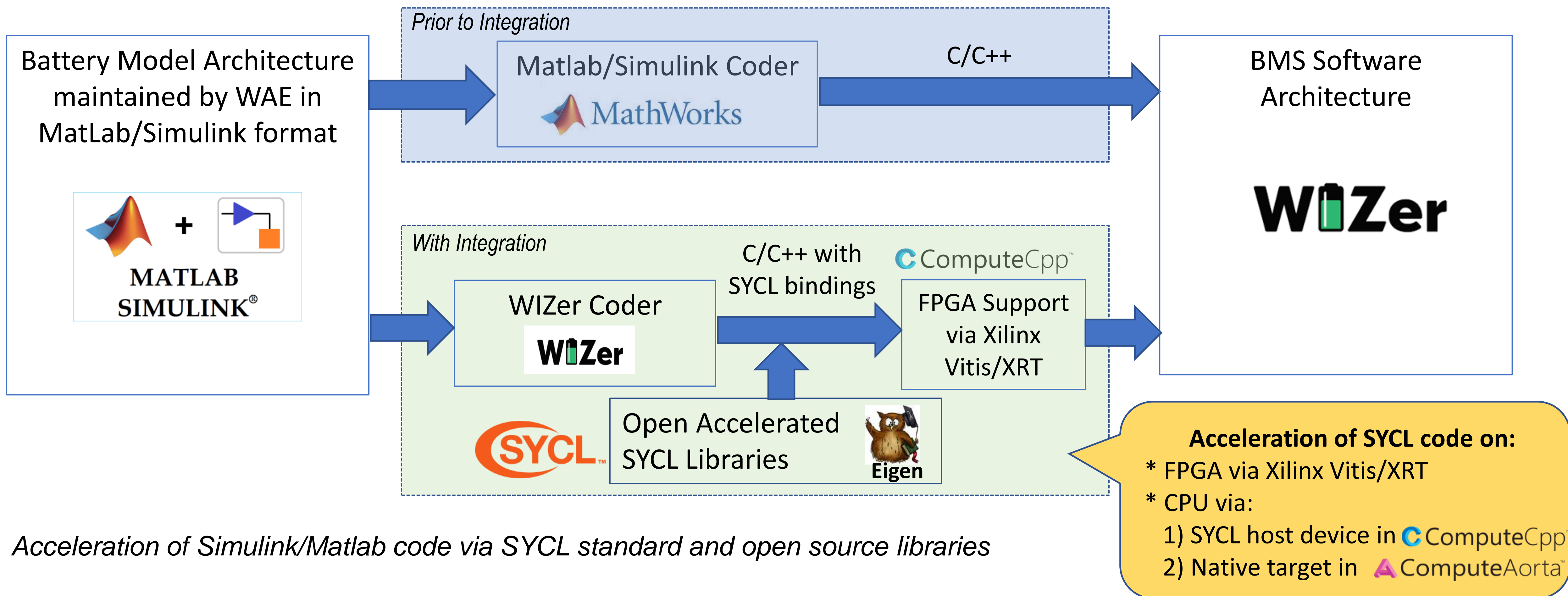
Zynq® UltraScale+™ MPSoC ZCU106 Evaluation Kit.

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

Targets low-power embedded applications, e.g. Advanced driver-assistance systems (ADAS), Battery Management Systems (BMS).

© 2023 Codeplay Software Ltd.

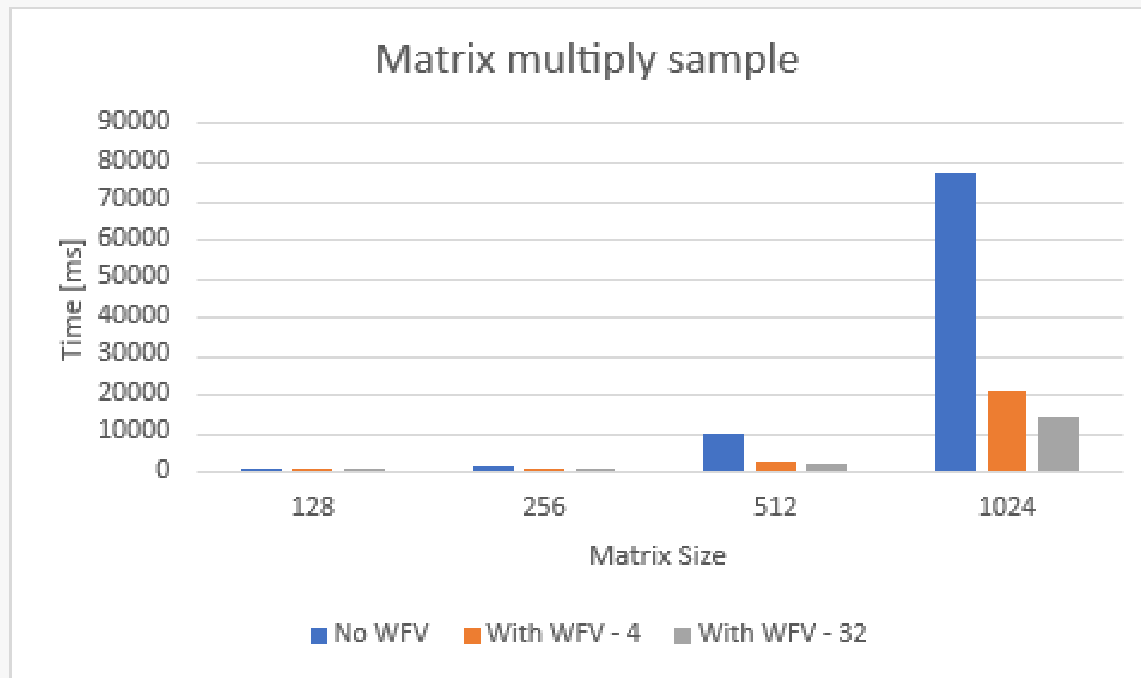
Software Acceleration – Integration Overview



Acceleration of Simulink/Matlab code via SYCL standard and open source libraries

Impact of compiler optimizations on Matrix Multiply running on Xilinx FPGA

ComputeCPP performs Whole Function Vectorization + single_task conversion (via ComputeAorta).



Best results with vector width of 32.

© 2023 Codeplay Software Ltd.

SYCL-BLAS General Matrix Multiplication running on embedded Xilinx zcu106 board with FPGA

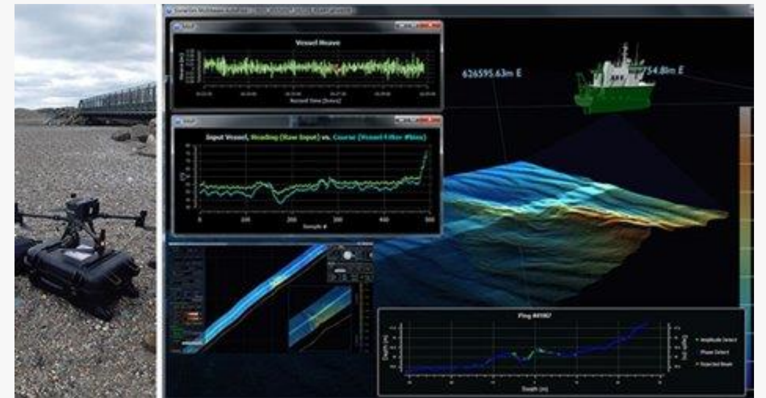
```
root@xilinx-zcu106-2020_2:~/gemm_test# ./run_gemm_test.sh
Time taken to load BIN is 192.000000 Milli Seconds
BIN FILE loaded through FPGA manager successfully
[==========] Running 296 tests from 6 test suites.
[----------] Global test environment set-up.
[----------] 72 tests from Gemm/GemmSmallBetaNonZeroLDMatchFloat
[ RUN      ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/0
XRT build version: 2.8.0
Build hash: f19a872233fbfe2eb933f25fa3d9a780ced774e5
Build date: 2021-03-30 19:53:41
Git branch: 2020.2
PID: 1928
UID: 0
[Fri Oct 21 08:13:28 2022 GMT]
HOST: xilinx-zcu106-2020_2
EXE: /home/root/gemm_test/blas3_gemm_test
[XRT] ERROR: Not implemented
Device vendor: Xilinx
Device name: zcu106_base
Device type: accelerator
[XRT] ERROR: Not implemented
[       OK ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/0 (600 ms)
[ RUN      ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/1
[XRT] ERROR: Not implemented
[       OK ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/1 (6 ms)
[ RUN      ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/2
[XRT] ERROR: Not implemented
[       OK ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/2 (6 ms)
[ RUN      ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/3
[XRT] ERROR: Not implemented
[       OK ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/3 (6 ms)
[ RUN      ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/4
[       OK ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/4 (3 ms)
[ RUN      ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/5
[       OK ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/5 (3 ms)
[ RUN      ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/6
[       OK ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/6 (3 ms)
[ RUN      ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/7
[       OK ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/7 (3 ms)
[ RUN      ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/8
[       OK ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/8 (3 ms)
[ RUN      ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/9
[       OK ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/9 (3 ms)
[ RUN      ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/10
[       OK ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/10 (3 ms)
[ RUN      ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/11
[       OK ] Gemm/GemmSmallBetaNonZeroLDMatchFloat.test/11 (3 ms)
```

```
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/8 (3718 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/9
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/9 (3458 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/10
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/10 (3704 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/11
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/11 (3712 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/12
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/12 (7068 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/13
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/13 (6782 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/14
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/14 (7040 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/15
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/15 (7046 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/16
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/16 (3782 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/17
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/17 (3492 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/18
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/18 (3759 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/19
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/19 (3763 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/20
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/20 (7148 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/21
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/21 (6864 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/22
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/22 (7145 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/23
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/23 (7147 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/24
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/24 (7441 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/25
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/25 (7033 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/26
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/26 (7444 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/27
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/27 (7451 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/28
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/28 (14150 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/29
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/29 (13890 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/30
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/30 (14176 ms)
[ RUN      ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/31
[       OK ] Gemm/GemmLargeBetaNonZeroLDMatchFloat.test/31 (14183 ms)
[----------] 32 tests from Gemm/GemmLargeBetaNonZeroLDMatchFloat (192982 ms total)

[----------] Global test environment tear-down
[==========] 296 tests from 6 test suites ran. (196042 ms total)
[  PASSED  ] 296 tests.
```


© 2023 Codeplay Software Ltd.

Unmanned Arial Vehicles (UAV)

- Autonomous UAVs for inspection of:
  - Ports
  - Bridges
  - Other structures
- Collision avoidance is critical
- AI 'at the edge'



codeplay®

Copyright Codeplay Software Ltd 2021

Unmanned Arial Vehicles (UAV)

- Intelligent radar perception processed by embedded device
- Improves automatic detection and avoidance capabilities of UAVs
- Collaboration with UWS, DataLab and Codeplay
- Uses deep learning models to perform classification tasks on the drone
- Employs ONNX runtime to perform model operations accelerated in SYCL

Copyright Codeplay Software Ltd 2021

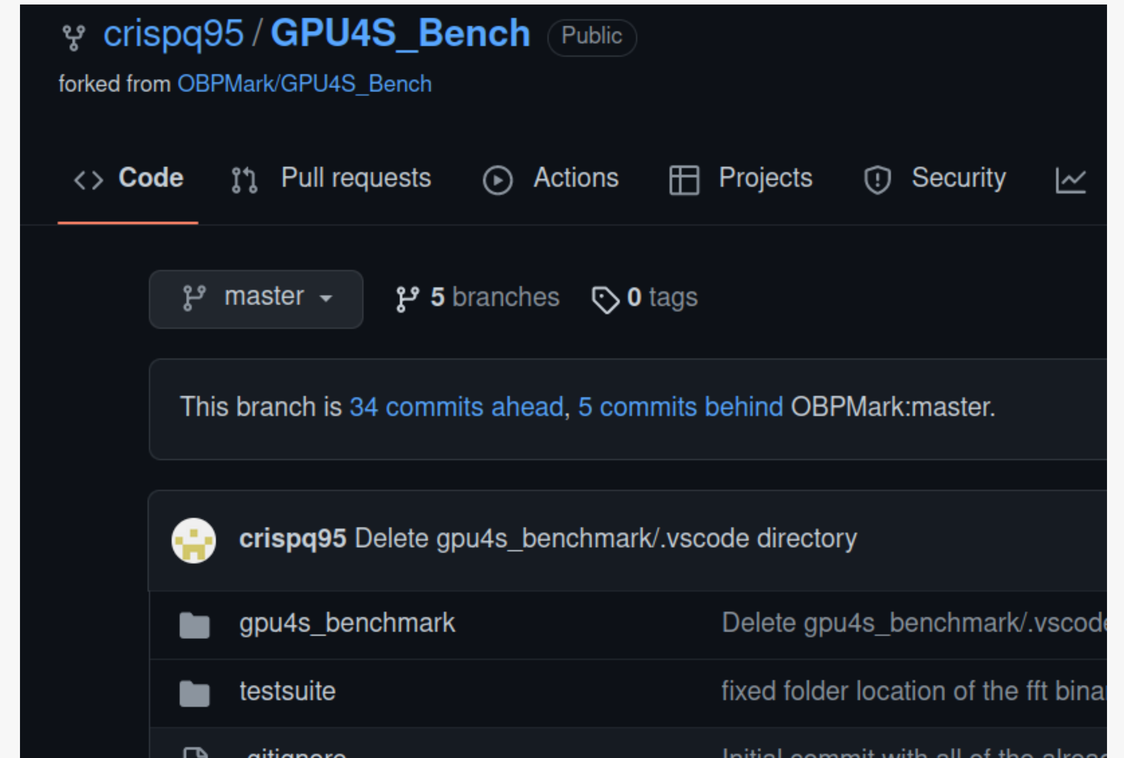
Medical Imaging on embedded devices

- Recently started collaboration with Napier University Edinburgh
- Developing new advanced AI- powered Computer Vision Algorithms for cancer diagnosis
- Enabling these algorithms to run on mobile medical devices/instruments – enabling these devices to perform more reliable image classification than SOTA.
- Employing SYCL/ComputeCPP to accelerate the new AI/Vision algorithms running on these embedded devices.

Copyright Codeplay Software Ltd 2021

GPUs for Space (GPU4S)

- Embedded GPU for spacecraft
- Benchmarking various kernels
- Ported to SYCL



codeplay
© 2023 Codeplay Software Ltd.

Safety-Critical requirements

- Applications in **automotive, avionics, healthcare/medical, energy, robotics** and other industries require functional safety and reliability guarantees – Application, tool chains and used APIs need to be certified to certain SC standards.

UL 4600

- Impacts on SYCL as many of these are embedded applications
- Various SC initiatives for SYCL in progress:
  - SYCL Safety-Critical Exploratory Forum https://www.khronos.org/syclsc
    - Building on experience with existing SC specifications such as Vulkan SC
  - Khronos and AUTOSAR collaborate on standardization in Automotive and Intelligent Mobility

© 2023 Codeplay Software Ltd.

Wrap

- SYCL targeting embedded is becoming increasingly important to provide a standards-based acceleration of artificial intelligence in **automotive, avionics, healthcare/medical, energy, robotics and other industries**
- Many embedded platforms already support SYCL (Arm, Xilinx, R-Car, Huawei) with more being added frequently.
- Embedded platforms take advantage of the existing software ecosystem (currently used by HPC).
- Applications in Automotive/ADAS
- Standardization of Safety-critical features in progress (Khronos + AUTOSAR)

Copyright Codeplay Software Ltd 2021

We're Hiring!
codeplay.com/careers/

Thank you!

@codeplaysoft

info@codeplay.com

codeplay.com