SYCL Academy

CPPCON 2021 class: SYCL 2020 and the future

Michael Wong

Acknowledgements:
SYCL WG

SYCL 2021 Tutorial group
Rod Burns

Parallel Programming with Modern C++: from CPU to GPU [2018 Class Archive]

"Parallel Programming with Modern C++: from CPU to GPU" is a two-day training course with programming exercises taught by Gordon Brown and Michael Wong. It is offered at the Meydenbauer Conference Center from 9AM to 5PM on Saturday and Sunday, September 29th and 30th, 2018 (immediately after the conference). Lunch is included.

Course Description

Parallel programming can be used to take advance of multi-core and heterogeneous architectures and can significantly increase the performance of software. It has gained a reputation for being difficult, but is it really? Modern C++ has gone a long way to making parallel programming easier and more accessible; providing both high-level and low-level abstractions. C++11 introduced the C++ memory model and standard threading library which includes threads, futures, promises, mutexes, atomics and more. C++17 takes this further by providing high level parallel algorithms; parallel implementations of many standard algorithms; and much more is expected in C++20. The introduction of the parallel algorithms also opens C++ to supporting non-CPU architectures, such as GPU, FPGAs, APUs and other accelerators.

This course will teach you the fundamentals of parallelism; how to recognise when to use parallelism, how to make the best choices and common parallel patterns such as reduce, map and scan which can be used over and again. It will teach you how to make use of the C++ standard threading library, but it will take this further by teaching you how to extend parallelism to heterogeneous devices, using the SYCL programming model to implement these patterns on a GPU using standard C++.

Parallelism in Modern C++: From CPU to GPU [2019 Class Archive]

"Parallelism in Modern C++: From CPU to GPU" is a two-day training course with programming exercises taught by Gordon Brown and Michael Wong. It is offered at the Gaylord Rockies from 9AM to 5PM on Saturday and Sunday, September 21st and 22nd, 2019 (immediately following the conference). Lunch is included.

Course Description

Parallel programming can be used to take advance of multi-core and heterogeneous architectures and can significantly increase the performance of software. It has gained a reputation for being difficult, but is it really? Modern C++ has gone a long way to making parallel programming easier and more accessible; providing both high-level and low-level abstractions. C++11 introduced the C++ memory model and standard threading library which includes threads, futures, promises, mutexes, atomics and more. C++17 takes this further by providing high level parallel algorithms; parallel implementations of many standard algorithms; and much more is expected in C++20. The introduction of the parallel algorithms also opens C++ to supporting non-CPU architectures, such as GPU, FPGAs, APUs and other accelerators.

This course will teach you the fundamentals of parallelism; how to recognize when to use parallelism, how to make the best choices and common parallel patterns such as reduce, map and scan which can be used over and again. It will teach you how to make use of the C++ standard threading library, but it will take this further by teaching you how to extend parallelism to heterogeneous devices, using the SYCL programming model to implement these patterns on a GPU using standard C++.

GPU Programming in Modern C++

GPU Programming in Modern C++ is a three-day online training course with programming exercises taught by Gordon Brown and Michael Wong. It is offered online from 11AM to 5PM Eastern Time (EDT), Monday September 21st through Wednesday September 23rd, 2020 (after the conference).

Course Description

Parallel programming can be used to take advance of heterogeneous architectures such as GPUs to significantly increase the performance of applications. It has gained a reputation for being difficult, but is it really? Modern C++ has gone a long way to making parallel programming easier and more accessible, and the introduction of the SYCL programming model means heterogeneous programming is now more accessible than ever.

This course will teach you the fundamentals of parallelism; how to recognize when to use parallelism, how to make the best choices and common parallel patterns which can be used over and again. It will teach you how to make use of modern C++ and the SYCL programming model to create parallel algorithms for heterogeneous devices such as GPUs. Finally, it will teach you how to apply common GPU optimizations.

Heterogeneous Programming in Modern C++ with SYCL

Heterogeneous Programming in Modern C++ with SYCL is a three-day online training course with programming exercises taught by Aksel Alpay, Gordon Brown, James Reinders, Michael Wong, Peter Zuzek, Rod Burns, and Ronan Keryell. It is offered **online** from 09:00 to 15:00 Aurora time (MDT), 11:00 to 17:00 EDT, 17:00 to 23:00 CET, Monday, November 1st through Wednesday, November 3rd, 2021 (after the conference).

Course Description

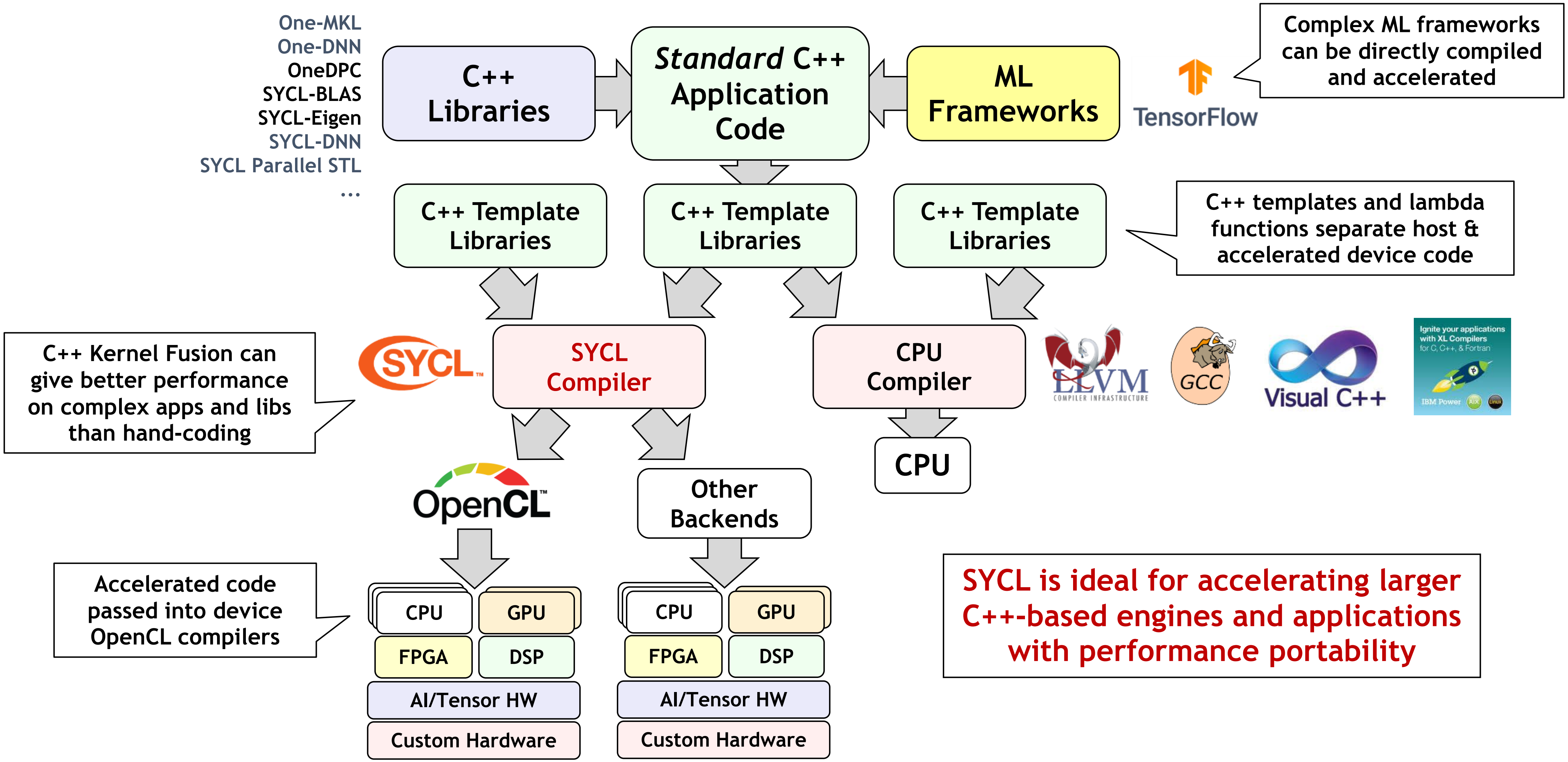
Parallel programming can be used to take advantage of heterogeneous architectures such as GPUs, FPGAs, ASICs, XPUs, IPUs, TPUs or special units on CPUs, to significantly increase the performance of applications. It has gained a reputation for being difficult, but is it really? Modern C++ has gone a long way to making parallel programming easier and more accessible, and the introduction of the SYCL programming model means heterogeneous programming is now more accessible than ever.

This course will teach you the fundamentals of heterogeneous parallelism; how to recognize when to use parallelism, how to make the best choices and common parallel patterns which can be used over and again. It will teach you how to make use of modern C++ and the SYCL programming model to create parallel algorithms for heterogeneous devices. Most of the programming focus will be on GPUs, but some time will be spent applying the techniques to simple FPGA examples. The course will teach you how to apply common GPU optimizations.

The challenges and general approaches for heterogeneous programming are well covered in this tutorial. Heterogeneous programming is an incredibly important topic, adding important dimensions to parallel programming that gained widespread usage after multicore (c. 2006) and the rise of GPU compute. The future is heterogeneous programming, with numerous device types from many vendors. This course will give attendees a deep appreciation of the challenge and a solid understanding of the programming techniques available to meet the challenge.

SYCL Academy

SYCL Single Source C++ Parallel Programming



SYCL 2020 is here!

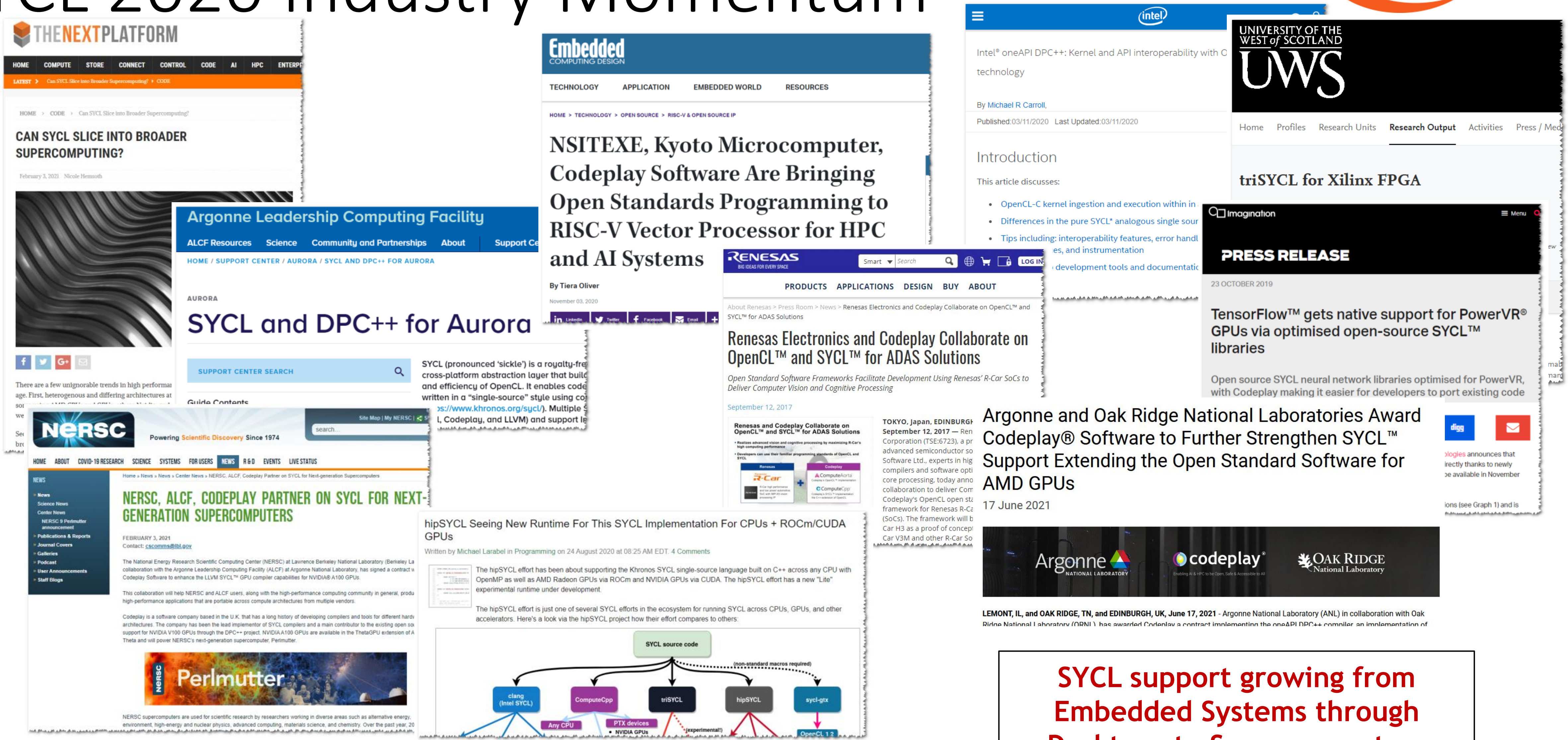
Open Standard for Single Source C++ Parallel Heterogeneous Programming

SYCL 2020 is released after 3 years of intense work
Significant adoption in Embedded, Desktop and HPC markets
Improved programmability, smaller code size, faster performance
Based on C++17, backwards compatible with SYCL 1.2.1
Simplify porting of standard C++ applications to SYCL
Closer alignment and integration with ISO C++
Multiple Backend acceleration and API independent

SYCL 2020 increases expressiveness and simplicity
for modern C++ heterogeneous programming

SYCL Academy

SYCL 2020 Industry Momentum



SYCL support growing from Embedded Systems through Desktops to Supercomputers

https://www.alcf.anl.gov/support-center/aurora/sycl-and-dpc-aurora
https://www.embeddedcomputing.com/technology/open-source/risc-v-open-source-ip/nsitexe-kyoto-microcomputer-and-codeplay-software-are-bringing-open-standards-programming-to-risc-v-vector-processor-for-hpc-and-ai-systems
https://www.nextplatform.com/2021/02/03/can-sycl-slice-into-broader-supercomputing/
https://www.phoronix.com/scan.php?page=news_item&px=hipSYCL-New-Lite-Runtime
https://software.intel.com/content/www/us/en/develop/articles/interoperability-dpcpp-sycl-opencl.html
https://www.renesas.com/br/en/about/press-room/renesas-electronics-and-codeplay-collaborate-opencl-and-sycl-adas-solutions
https://www.nersc.gov/news-publications/nersc-news/nersc-center-news/2021/nersc-alcf-codeplay-partner-on-sycl-for-next-generation-supercomputers/
https://research-portal.uws.ac.uk/en/publications/trisycl-for-xilinx-fpga
https://www.imaginationtech.com/news/press-release/tensorflow-gets-native-support-for-powervr-gpus-via-optimised-open-source-sycl-libraries/

SYCL 2020 Major Features

- Unified Shared Memory (USM)
  - Code with pointers can work naturally without buffers or accessors
  - Simplifies porting from most code (e.g. CUDA, C++)
- Parallel Reductions
  - Added built-in reduction operation to avoid boilerplate code and achieve maximum performance on hardware with built-in reduction operation acceleration.
- Work group and subgroup algorithms
  - Efficient parallel operations between work items
- Class template argument deduction (CTAD) and template deduction guides
  - Simplified class template instantiation
- Simplified use of Accessors with a built-in reduction operation
  - Reduces boilerplate code and streamlines the use of C++ software design patterns
- Expanded interoperability
  - Efficient acceleration by diverse backend acceleration APIs
- SYCL atomic operations are now more closely aligned to standard C++ atomics
  - Enhances parallel programming freedom

Parallel Industry Initiatives

C++11

C++14

C++17

C++20

C++23

SYCL 1.2
C++11 Single source programming

SYCL 1.2.1
C++11 Single source programming

SYCL 2020
C++17 Single source programming
Many backend options

SYCL 202X
C++20 Single source programming
Many backend options

OpenCL 1.2
OpenCL C Kernel Language

OpenCL 2.1
SPIR-V in Core

OpenCL 2.2

OpenCL 3.0

2011

2015

2017

2020

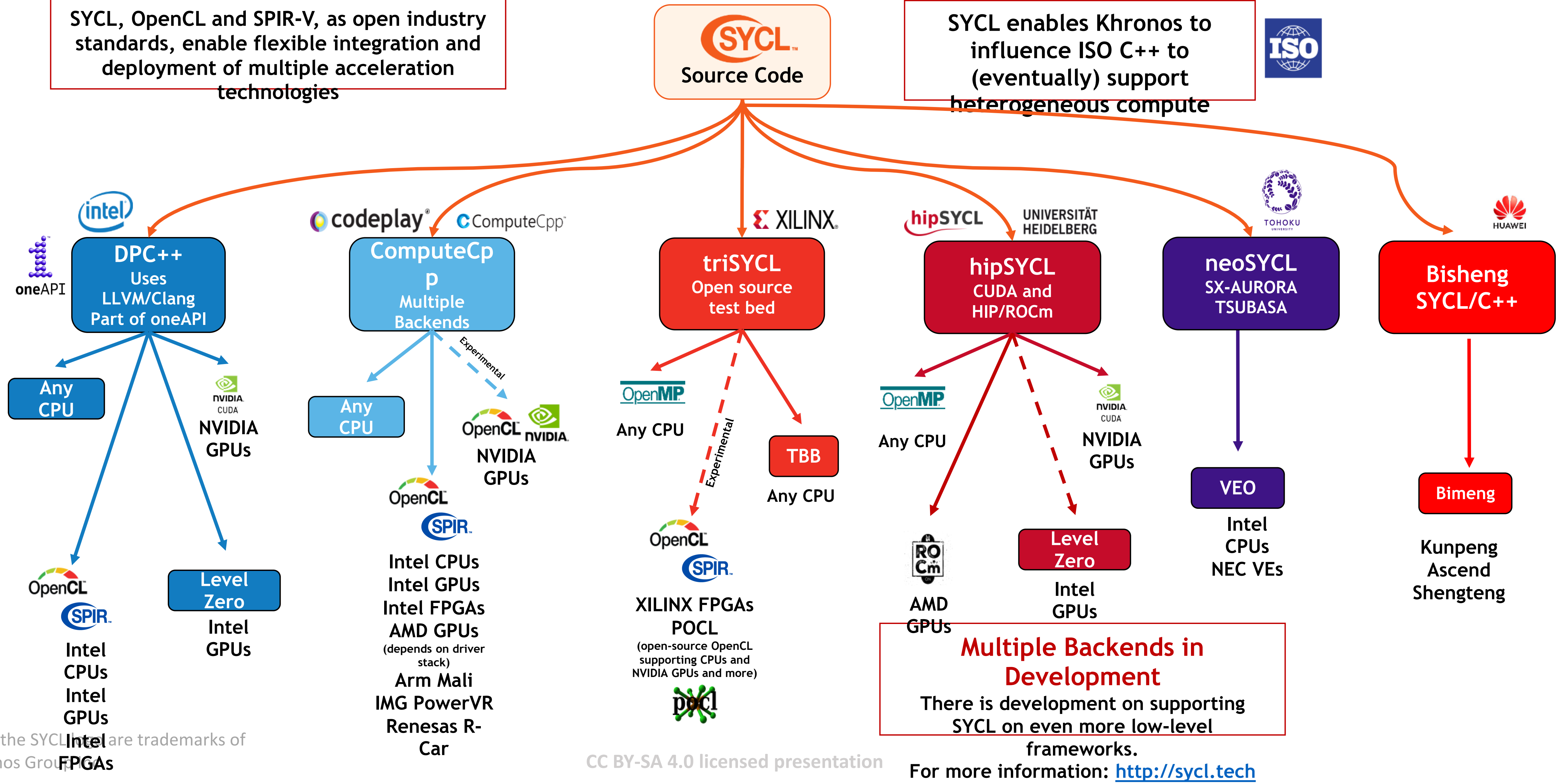
202X

SYCL Academy

SYCL Implementations in Development

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



Multiple Backends in Development

There is development on supporting SYCL on even more low-level frameworks.

For more information: http://sycl.tech

SYCL Ecosystem, Research and Benchmarks

Implementations

triSYCL

neoSYCL SX-AURORA TSUBASA

oneAPI

DATA PARALLEL C++
Performant, Productive Programming for Heterogeneous Architecture

ComputeCpp

hipSYCL

Celerity
High-level C++ for Accelerator Clusters

ECP EXASCALE COMPUTING PROJECT

kokkos

Taskflow: A General-purpose Parallel and Heterogeneous Task Programming System using Modern C++
Dr. Tsung-Wei (TW) Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT
https://taskflow.github.io/

RAJA

Grid: A High-Performance and Portable Code for Quantum Chromodynamics. [E32016]

SYCL 2020 Programming

ATLAS EXPERIMENT

GROMACS
FAST. FLEXIBLE. FREE.

alpaka

Benchmarks/Books

Direct Programming Benchmark

Data Parallel C++
Mastering DPC++ for Programming of Heterogeneous Systems using C++ and SYCL
James Reinders
Ben Ashbaugh
James Brodman
Michael Kinsner
John Pennycook
Xinmin Tian

FREE eBook

360k downloads

SYCL-Bench

Linear Algebra Libraries

Machine Learning Libraries and Parallel Acceleration Frameworks

BLAS	FFT	Math	RAND
SYCLBLAS oneMKL	oneMKL	oneMKL	oneMKL
SOLVER	**SPARSE**	**TENSOR**	**STL**
oneMKL	oneMKL	SYCL-DNN Eigen oneDNN TensorFlow	SYCL Parallel STL oneDPL

STREAM HPC

Argonne NATIONAL LABORATORY

QUALCOMM

University of BRISTOL

arm

AMD

SYCL

Working Group Members

codeplay

XILINX

universität innsbruck

intel

TOHOKU UNIVERSITY

SYCL in Embedded Systems, Automotive, and AI

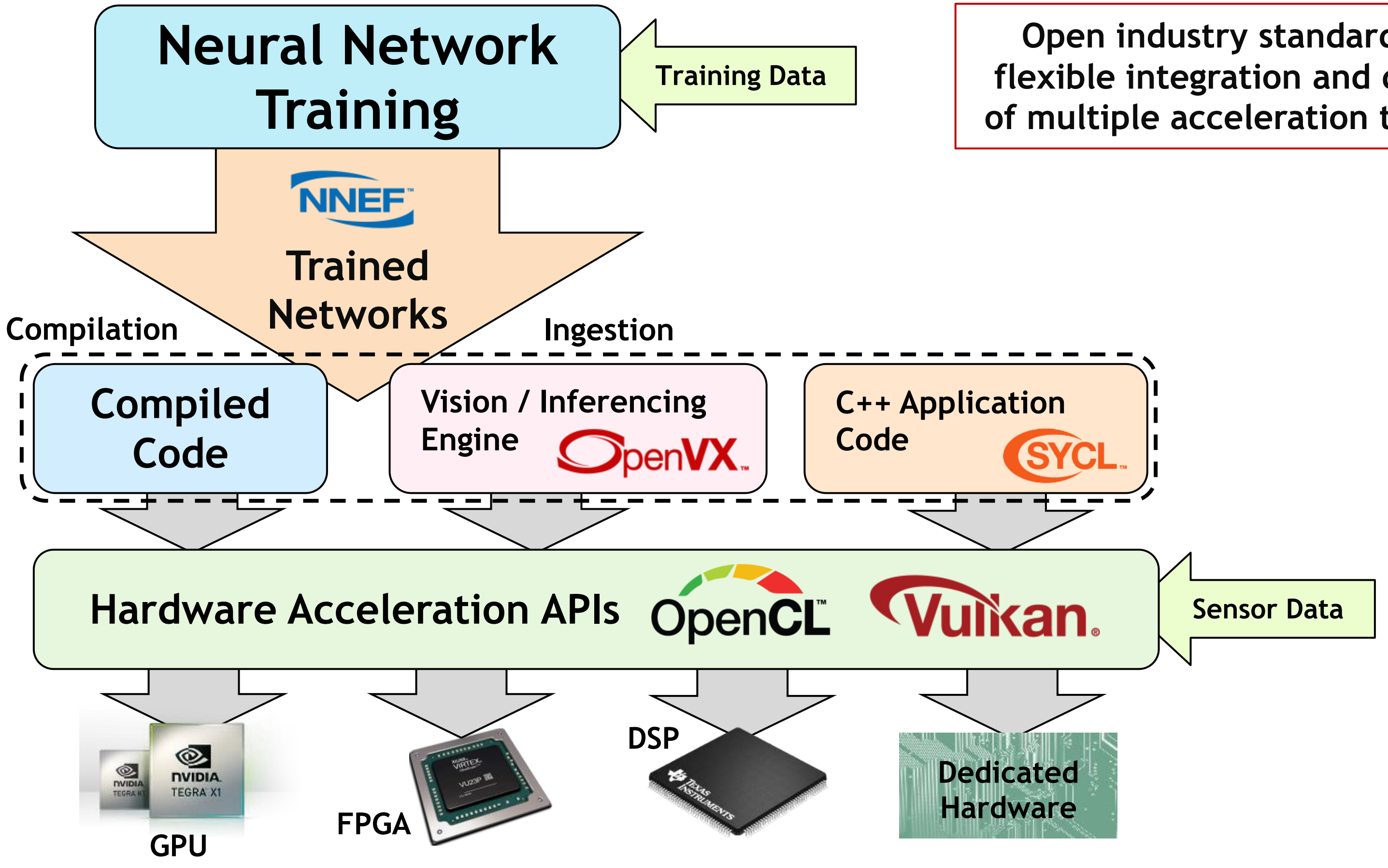
Open industry standards, enable flexible integration and deployment of multiple acceleration technologies

Networks trained on high-end desktop and cloud systems

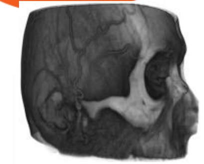
Applications link to compiled inferencing code or call vision/inferencing API

Diverse Embedded Hardware

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



Safety Critical API Evolution



New Generation Safety Critical APIs for Graphics, Compute and Display

OpenCL and SYCL SC work will minimize API surface area , reduce ambiguity, UB, increase determinism

OpenCL

SYCL™

Rendering Compute Display

Industry Need for GPU Acceleration APIs designed to ease system safety certification is increasing

ISO 26262 / ASIL-D

UNECE WP.29

SCSC FOR EVERYONE WORKING IN SYSTEM SAFETY WWW.SCSC.UK

ISO International Organization for Standardization

ISO 26262

ISO/PAS 21448

UL 4600

MISRA

SAE INTERNATIONAL

ISO/IEC JTC 1/SC 42
Artificial intelligence

RISC-V

SYCL Academy

SYCL in HPC/Supercomputers

Simulation	Data	Learning
HPC Languages Solver Libraries, Parallel RT	Productivity Languages Big Data Stack, Stats Lib, Databases	Productivity Languages Deep Learning, Linear Alg, ML

Three Pillars of Science Problem

OpenMP — OpenMP for C and Fortran

OpenCL — CUDA/pthreads/ OpenACC/OpenCL

SYCL — C++ Application uses SYCL, Kokkos, Raja

Math, ML, Data Libraries; C++ Std, C, Python Libraries

2021 — 2021 — 2022 — 2022 — 2023

SYCL

Need Languages that allow control of these Data Issues
Set Data affinity, Data Layout, Data movement, Data Locality, highly Parameterized Code and dynamically compose the algorithms (C++ templates, parallel STL, inlining and fusion, abstractions)

Libraries augment compiler optimizations for Performance Portable programs

Use open standards to run Performance Portable code on new generation, or different vendor's, hardware with compiler optimization, explicit parametrization and dynamically composed algorithm

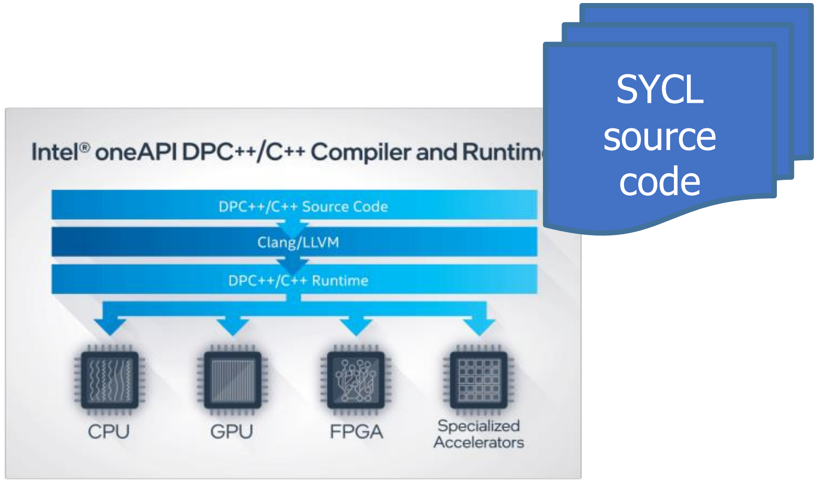
Today's Supercomputing Development Workflow needs knowledge of system architecture and tools that control data

Choose Algorithm for target

Implement and Test Algorithm

Optimize Algorithm

oneAPI and SYCL

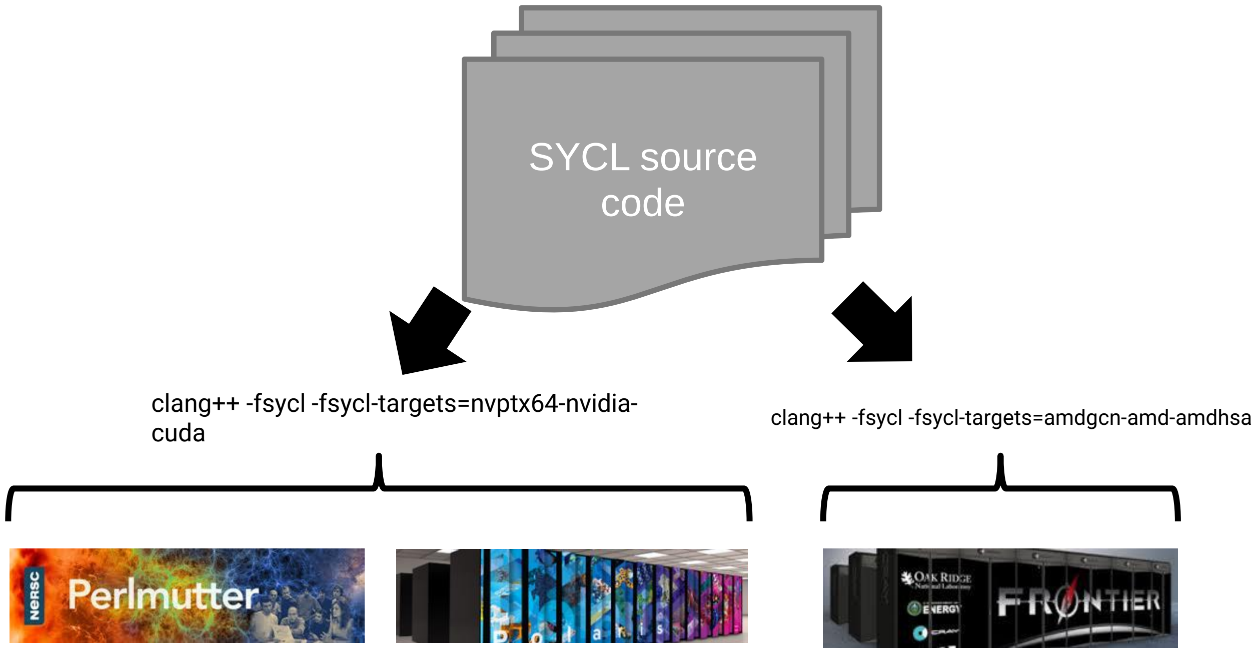


- SYCL sits at the heart of oneAPI
- Provides an open standard interface for developers
- Defined by the industry

Nvidia and AMD Support in oneAPI

- Extending DPC++ to target Nvidia and AMD GPUs
- Supporting Perlmutter, Polaris and Frontier supercomputers
- Open source and available to everyone

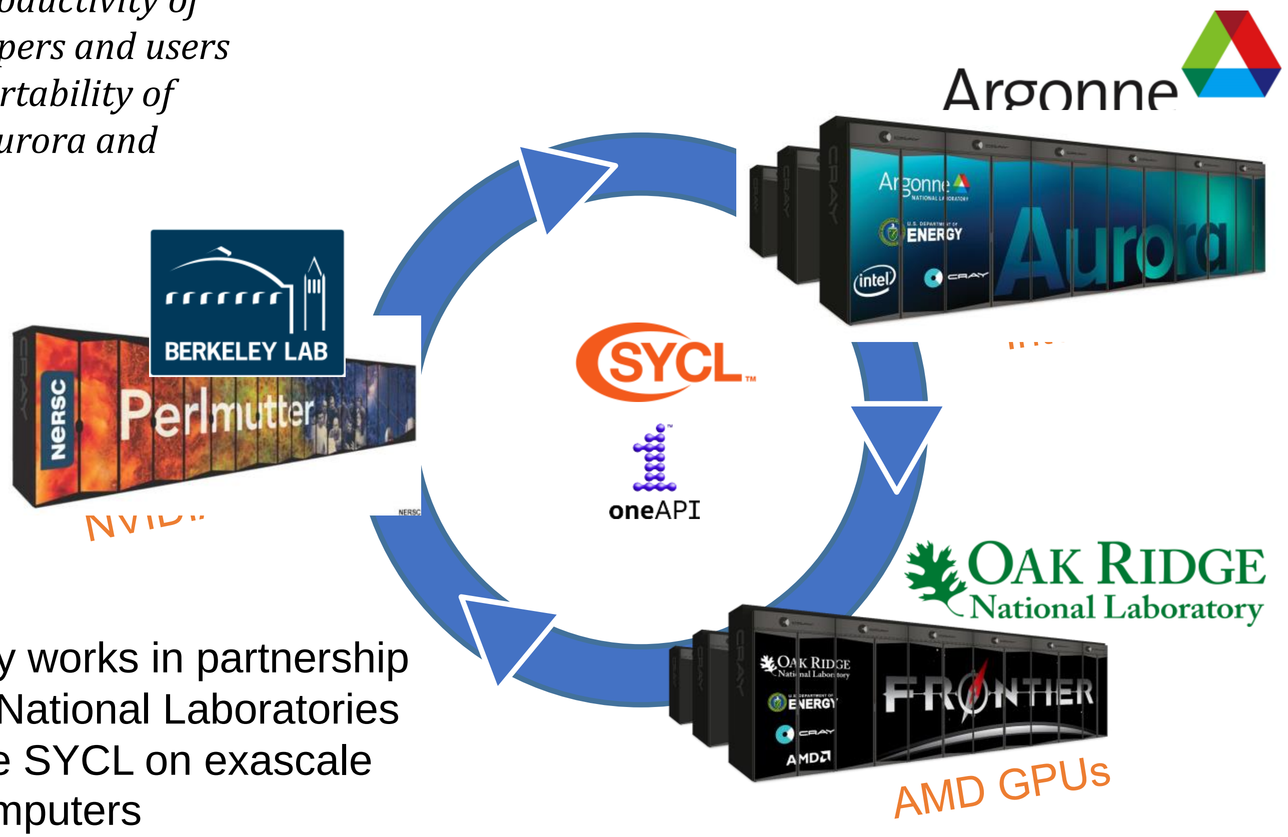
Different targets using a simple compiler flag



https://www.codeplay.com/oneapiforcuda
Resources for AMD coming soon

SYCL Enables Supercomputers

“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

SYCL Future Evolution

SYCL 2020 compared with SYCL 1.2.1

- Easier to integrate with C++17 (CTAD, Deduction Guides...)
- Less verbose, smaller code size, simplify patterns
- Backend independent
- Multiple object archives aka modules simplify interoperability
- Ease porting C++ applications to SYCL
- Enable capabilities to improve programmability
- Backwards compatible but minor API break based on user feedback

SYCL Future Roadmap (MAY CHANGE)

SYCL 2020

NEXT

Integration of successful Extensions plus new Core functionality

Over 40 Selected Features for SYCL 2020

Unified Shared Memory)
Parallel Reductions adds a built in reduction operation
Work-group and sub-group algorithms
Improvements to atomic operations
Class template argument deduction (CTAD) and deduction guides
Simplification of accessors
Expanded interoperability with different backends
Extension mechanism
Address spaces
Vector rework
Specialization Constants
...

Improving Software Ecosystem

Books, Tutorials, Tool, libraries, GitHub

Expanding Implementations

DPC++
ComputeCpp
triSYCL
hipSYCL
neoSYCL

Regular Maintenance Updates

Spec clarifications, formatting and bug fixes
https://www.khronos.org/registry/SYCL/

Conformance Tests

Working on Implementations

Future SYCL NEXT Proposals

...

Converge SYCL with ISO C++ and continue to support OpenCL to deploy on more devices

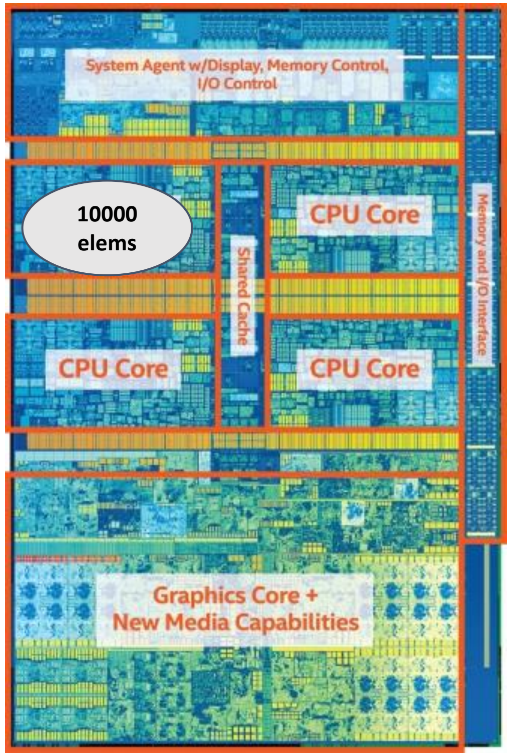
CPU
GPU
FPGA
AI processors
Custom Processors

Repeat The Cycle every 1.5-3 years

SYCL Academy

A Demo with C++ Parallel STL

What can I do with a Parallel For Each?

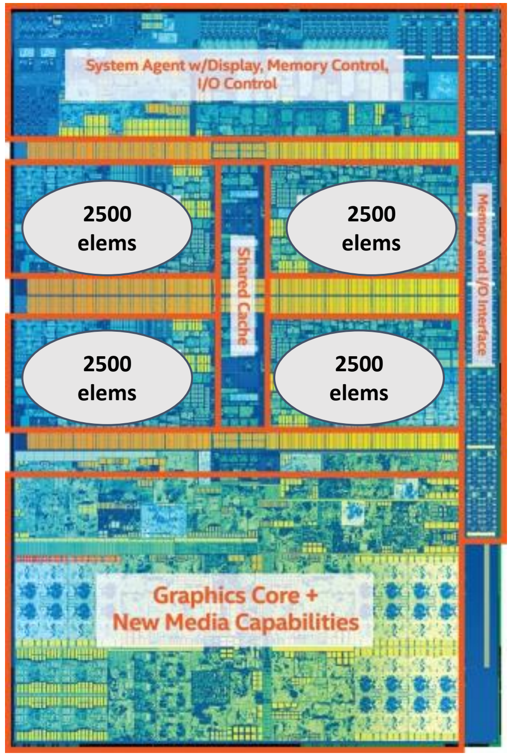


Intel Core i7 7th generation

```
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(std::begin(v1), nElems, 1);

std::for_each(std::begin(v), std::end(v),
                            [=](float f) { f * f + f });
```

Traditional for each uses only one core, rest of the die is unutilized!

What can I do with a Parallel For Each?



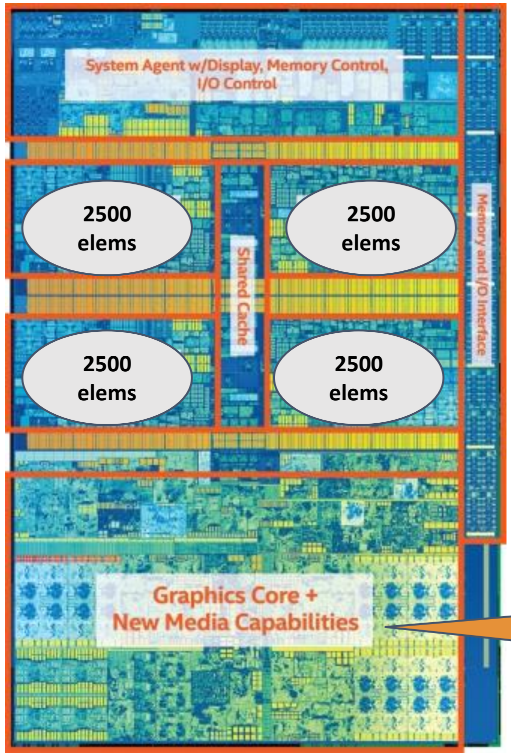
Intel Core i7 7th generation

```
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(std::execution_policy::par,
            std::begin(v1), nElems, 1);

std::for_each(std::execution_policy::par,
              std::begin(v), std::end(v),
              [=](float f) { f * f + f });
```

Workload is distributed across cores!

(mileage may vary, implementation-specific behaviour)

What can I do with a Parallel For Each?



Intel Core i7 7th generation

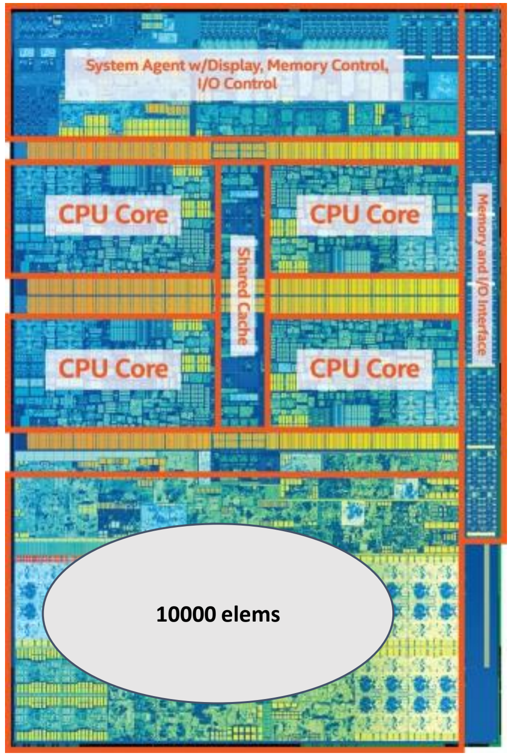
```
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(std::execution_policy::par,
            std::begin(v1), nElems, 1);

std::for_each(std::execution_policy::par,
              std::begin(v), std::end(v),
              [=](float f) { f * f + f });
```

What about this part?

Workload is distributed across cores!

(mileage may vary, implementation-specific behaviour)

What can I do with a Parallel For Each?



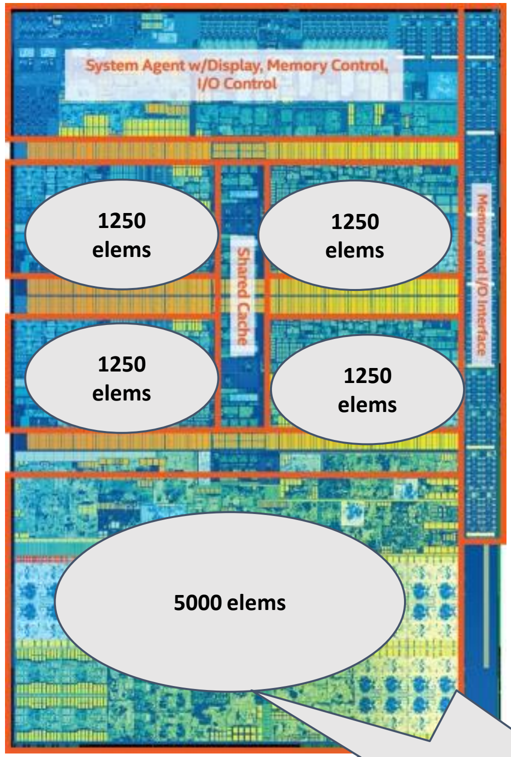
Intel Core i7 7th generation

```
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(sycl_policy,
            std::begin(v1), nElems, 1);

std::for_each(sycl_named_policy
              <class KernelName>,
              std::begin(v), std::end(v),
              [=](float f) { f * f + f });
```

Workload is distributed on the GPU cores

(mileage may vary, implementation-specific behaviour)

What can I do with a Parallel For Each?



Intel Core i7 7th

Experimental!

```
size_t nElems = 1000u;
std::vector<float> nums(nElems);

std::fill_n(sycl_heter_policy(cpu, gpu, 0.5),
            std::begin(v1), nElems, 1);

std::for_each(sycl_heter_policy<class kName>
              (cpu, gpu, 0.5),
               std::begin(v), std::end(v),
               [=](float f) { f * f + f });
```

Workload is distributed on all cores!

(mileage may vary, implementation-specific behaviour)


cppcon | 2017

Demo Results - Running std::sort
(Running on Intel i7 6600 CPU & Intel HD Graphics 520)

size	2^16	2^17	2^18	2^19
std::seq	0.27031s	0.620068s	0.669628s	1.48918s
std::par	0.259486s	0.478032s	0.444422s	1.83599s
std::par_unseq	0.24258s	0.413909s	0.456224s	1.01958s
sycl_execution_policy	0.273724s	0.269804s	0.277747s	0.399634s

SYCL 2020 is here!

Open Standard for Single Source C++ Parallel Heterogeneous Programming

SYCL 2020 is released after 3 years of intense work
Significant adoption in Embedded, Desktop and HPC markets
Improved programmability, smaller code size, faster performance
Based on C++17, backwards compatible with SYCL 1.2.1
Simplify porting of standard C++ applications to SYCL
Closer alignment and integration with ISO C++
Multiple Backend acceleration and API independent

SYCL 2020 increases expressiveness and simplicity
for modern C++ heterogeneous programming

Enabling Industry Engagement

- SYCL working group values industry feedback
  - https://community.khronos.org/c/sycl
  - https://sycl.tech
- SYCL FAQ
  - https://www.khronos.org/blog/sycl-2020-what-do-you-need-to-know
- What features would you like in future SYCL versions?

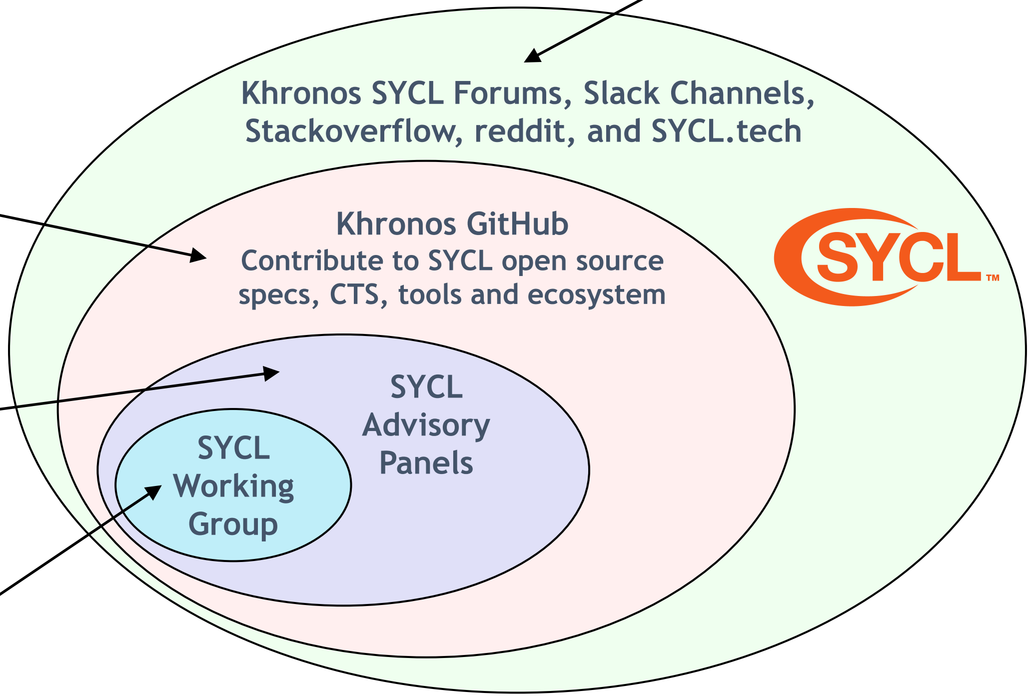
Open to all!
https://community.khronos.org/www.khr.io/slack
https://app.slack.com/client/TDMDFS87M/CE9UX4CHG
https://community.khronos.org/c/sycl/
https://stackoverflow.com/questions/tagged/sycl
https://www.reddit.com/r/sycl
https://github.com/codeplaysoftware/syclacademy
https://sycl.tech/

- Advisory Panel Chaired by Tom Deakin of U of Bristol
- Quarterly SYCL Advisory Panel
- Regular meetings to give feedback on roadmap and draft specifications

Public contributions to Specification, Conformance Tests and software
https://github.com/KhronosGroup/SYCL-CTS
https://github.com/KhronosGroup/SYCL-Docs
https://github.com/KhronosGroup/SYCL-Shared
https://github.com/KhronosGroup/SYCL-Registry
https://github.com/KhronosGroup/SyclParallelSTL
https://github.com/intel/llvm

Invited Experts
https://www.khronos.org/advisors/

Khronos members
https://www.khronos.org/members/
https://www.khronos.org/registry/SYCL/



Thank you to You, and all the contributors

- This is really a Special edition of the SYCL class, more time then any other tutorial, more experts then ever
  - Unlikely to ever repeat again
- Rod Burns for herding all the cats and made this such a great tutorial
- Gordon Brown who wrote most of the original tutorial
- James Reinders, Ben Odom, Aksel Alpay, Ronan Keryell, Ben Ashburgh, Peter Zuzek, Mike Kinsner, Susannah Martin, John Pennycook, Jason Sewall, James Brodman, Hugh Delaney
- And many others who contributed
- I am sure I forgot someone and it is unintentional, and all my fault