



SYCL codesign with RISC-V as open standard programming for HPC, AI, and Datacenter

Michael Wong

Distinguished Engineer

SYCL WG Chair

ISO C++ Directions Group

Chair of C++ Machine Learning, Low latency, Games, Embedded, Finance

RISC-V Datacenter/Cloud Computing Chair

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.

Supported Solutions



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™



And many more!

Markets

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

Technologies: Artificial Intelligence
Vision Processing
Machine Learning
Big Data Compute

Distinguished Engineer

- Chair of SYCL Heterogeneous Programming Language
- RISC-V Datacenter/Cloud COmputign Chair
- ISO C++ Directions Group past Chair
- Past CEO OpenMP
- ISOCPP.org Director, VP
- <http://isocpp.org/wiki/faq/wg21#michael-wong>
- michael@codeplay.com
- fraggamuffin@gmail.com
- Head of Delegation for C++ Standard for Canada
- Chair of Programming Languages for Standards Council of Canada
- Chair of WG21 SG19 Machine Learning
- Chair of WG21 SG14 Games Dev/Low Latency/Financial Trading/Embedded
- Editor: C++ SG5 Transactional Memory Technical Specification
- Editor: C++ SG1 Concurrency Technical Specification
- MISRA C++ and AUTOSAR
- Chair of Standards Council Canada TC22/SC32 Electrical and electronic components (SOTIF)
- Chair of UL4600 Object Tracking
- <http://wongmichael.com/about>
- C++11 book in Chinese:
<https://www.amazon.cn/dp/B00ETOV2OQ>

Michael Wong

Argonne and Oak Ridge National Laboratories Award Codeplay® Software to Further Strengthen SYCL™ Support Extending the Open Standard Software for AMD GPUs

17 June 2021



LEMONT, IL, and OAK RIDGE, TN, and EDINBURGH, UK, June 17, 2021 - Argonne National Laboratory (ANL) in collaboration with Oak Ridge National Laboratory (ORNL), has awarded Codeplay a contract implementing the oneAPI DPC++ compiler, an implementation of the SYCL™ open standard software, to support AMD GPU-based high-performance compute (HPC) supercomputers.

NSITEXE, Kyoto Microcomputer and Codeplay Software are bringing open standards programming to RISC-V Vector processor for HPC and AI systems

29 October 2020



Implementing OpenCL™ and SYCL™ for the popular RISC-V processors will make it easier to port existing HPC and AI software for embedded systems

NERSC, ALCF, Codeplay Partner on SYCL for Next-generation Supercomputers

02 February 2021



The National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory (Berkeley Lab), in collaboration with the Argonne Leadership Computing Facility (ALCF) at Argonne National Laboratory, has signed a contract with Codeplay Software to enhance the LLVM SYCL™ GPU compiler capabilities for NVIDIA® A100 GPUs.

We build GPU compilers for some of the most powerful supercomputers in the world

Acknowledgement and Disclaimer



THIS WORK REPRESENTS THE VIEW OF THE AUTHOR AND DOES NOT NECESSARILY REPRESENT THE VIEW OF CODEPLAY.



OTHER COMPANY, PRODUCT, AND SERVICE NAMES MAY BE TRADEMARKS OR SERVICE MARKS OF OTHERS.

Nvidia is a registered trademark of NVIDIA Corporation.
AMD is a registered trademark of Advanced Micro Devices Corporation
SYCL, SPIR are trademarks of the Khronos Group Inc. OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.

Numerous people internal and external to the original C++/Khronos group/OpenMP, in industry and academia, have made contributions, influenced ideas, written part of this presentations, and offered feedbacks to form part of this talk. These include Bjarne Stroustrup, Joe Hummel, Botond Ballo, Simon McIntosh-Smith, Rod Burns, Ronan Keryell, Mike Kinsner, James Brodman, Colin Davidson, Fraser Cormack, Mehdi Goli, Aidan Dodds, SYCL WG, ISO C++, OpenMP, Codeplay Research as well as many others.

But I claim all credit for errors, and stupid mistakes. These are mine, all mine! You can't have it

Legal Disclaimer



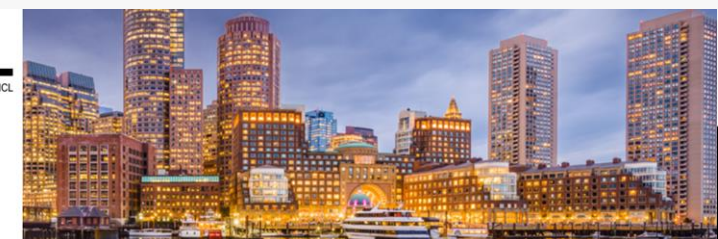
THIS WORK REPRESENTS THE VIEW OF THE AUTHOR AND DOES NOT NECESSARILY REPRESENT THE VIEW OF CODEPLAY.



OTHER COMPANY, PRODUCT, AND SERVICE NAMES MAY BE TRADEMARKS OR SERVICE MARKS OF OTHERS.

A Brief history of DHPCC++

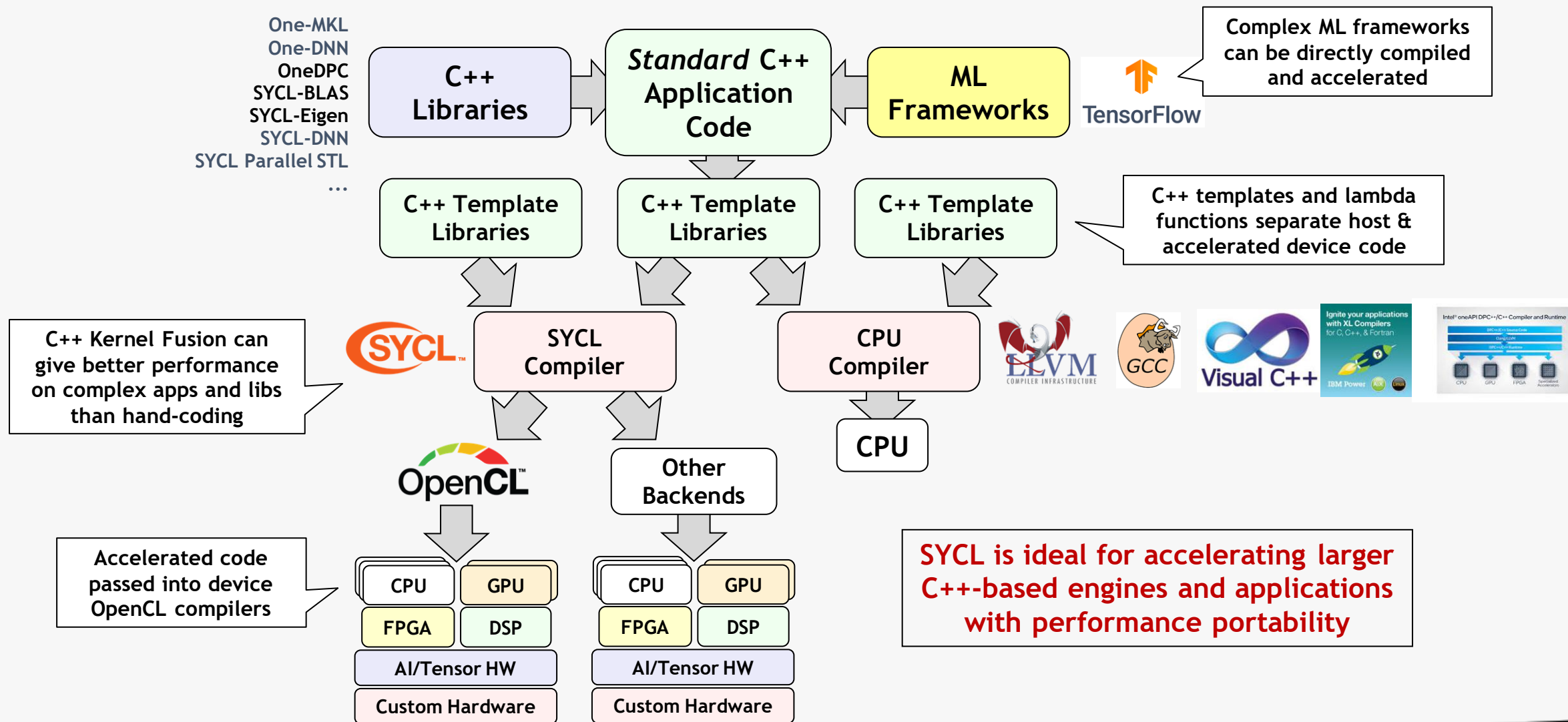
- A mouthful like RAI1 but what does it mean ...??
- First was 2017 with IWOCL 2017, Toronto, Canada by Rod and I
 - [Distributed and Heterogeneous Programming in C/C++ \(DHPCC++17\) - SYCL.tech](#)
 - Actually it was not the first, it was just the first called by that name which I deliberately targeted HPC in the middle of the name but it doesn't stand for HPC
 - First was
 - SYCL workshop at [PPoPP'16](#) as well as the [Berkeley Heterogeneous C++ Summit in 2016](#).
- Second was 2018 in Oxford, UK
 - [Heterogeneous Development at the DHPCC++ 2018 Workshop - Codeplay Software Ltd](#)
- Third was 2019 in Boston, USA
 - [DHPCC++ 2019 \(iwocl.org\)](#)
- Then we stopped ... why?
 - SYCLCon 2020 coincided with SYCL becoming a TUII WG
- Rebirth 2022 at EuroPar
 - [Distributed and Heterogeneous Programming in C++ \(DHPCC++22\) - SYCL.tech](#)
- More coming



What I'll Talk about

- The future of C++ with heterogeneous systems
- SYCL's role in this future
- Emergence of RISC-V
- The safety critical challenges

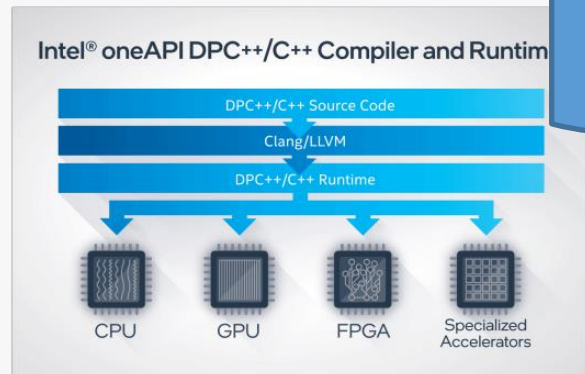
SYCL Single Source C++ Parallel Programming




oneAPI and SYCL

1
oneAPI

SYCL™

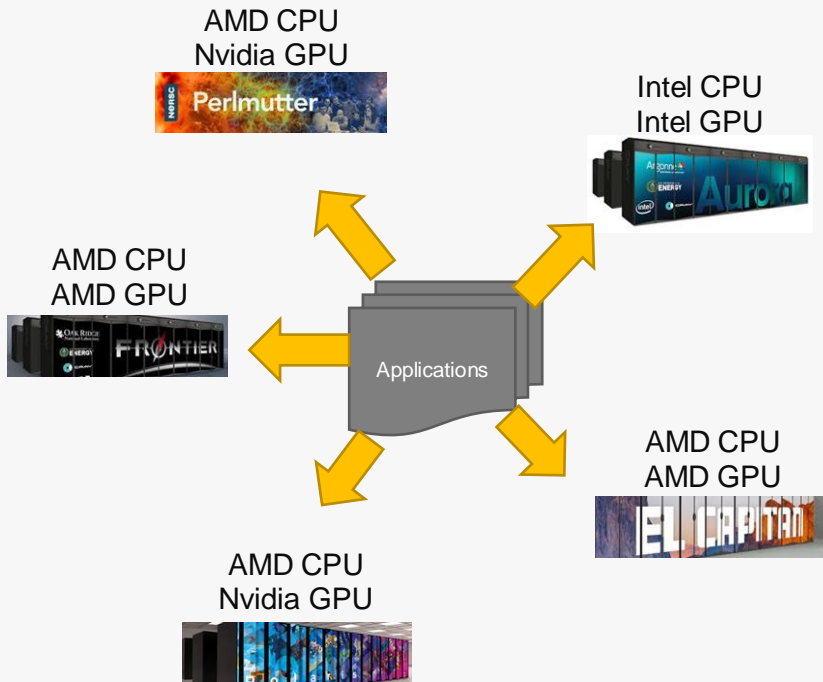


SYCL source code

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

Programming Models Must Persist

US National Laboratory Supercomputers 2021-2023



- HPC and now exascale computing requires programming models that endure for future workloads, > 20 years
- But Hardware changes frequently, constant improvement
- Programming models, have to be stable but also support latest HW,

Requires an open interface, across architectures with multiple implementations

SYCL 2020 Launched February 2021

Expressiveness and simplicity for heterogeneous programming in modern C++

Closer alignment and integration with ISO C++ to simplify porting of standard C++ applications

Improved programmability, smaller code size, faster performance

Based on C++17, backwards compatible with SYCL 1.2.1

Backend acceleration API independent

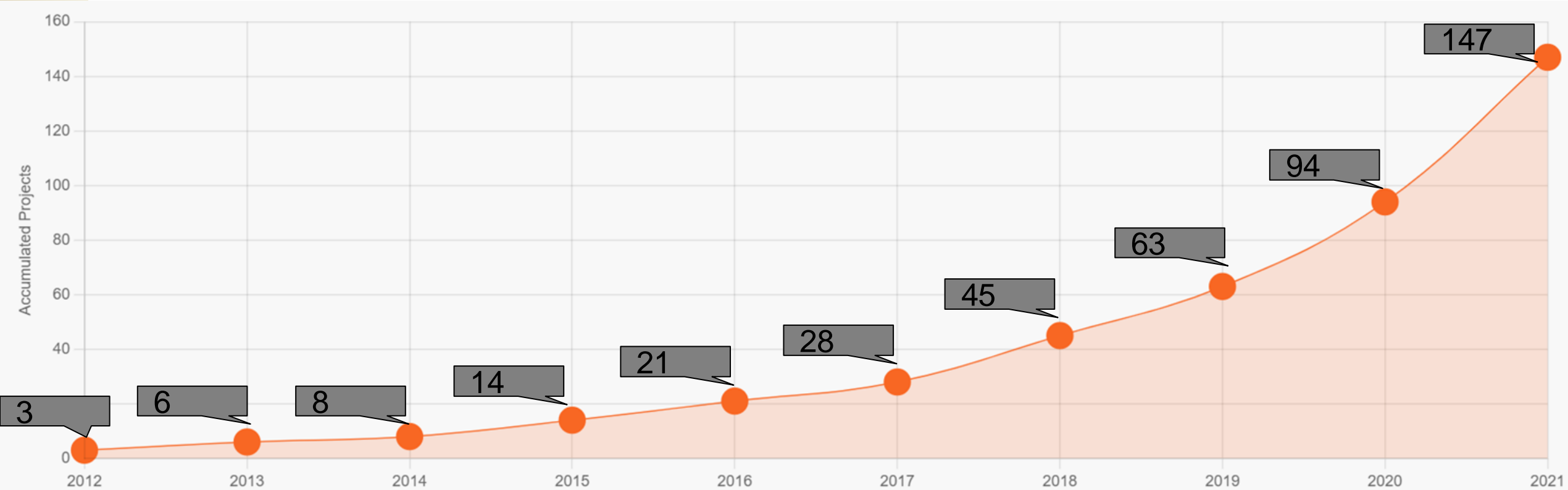
New Features

Unified Shared Memory | Parallel Reductions | Subgroup Operations | Class template Argument Deduction

Significant SYCL adoption in Embedded, Desktop and HPC Markets

SYCL Projects cumulative growth

[Projects - SYCL.tech](https://www.sycl.tech)



Benchmarks



Frameworks

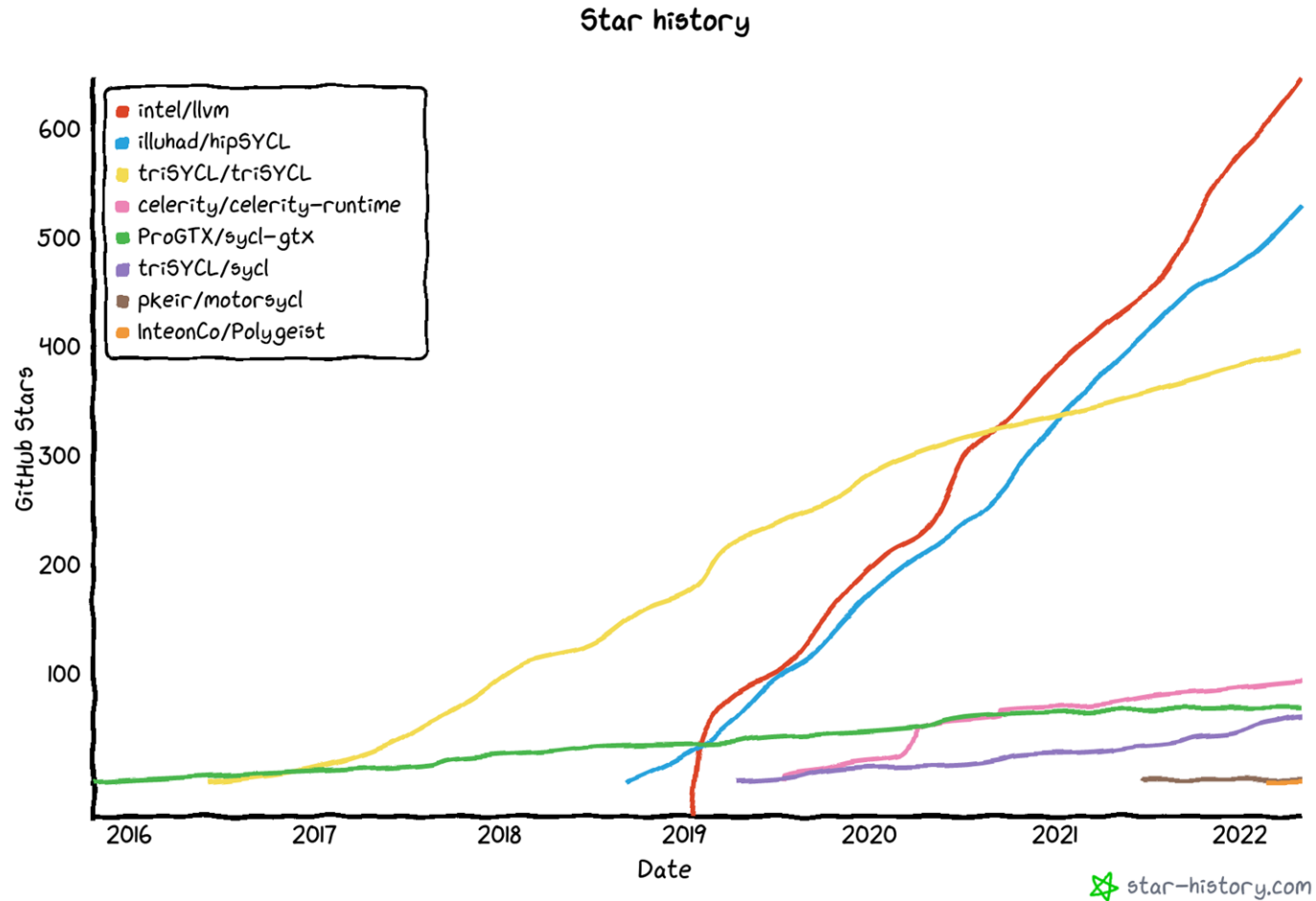


Libraries

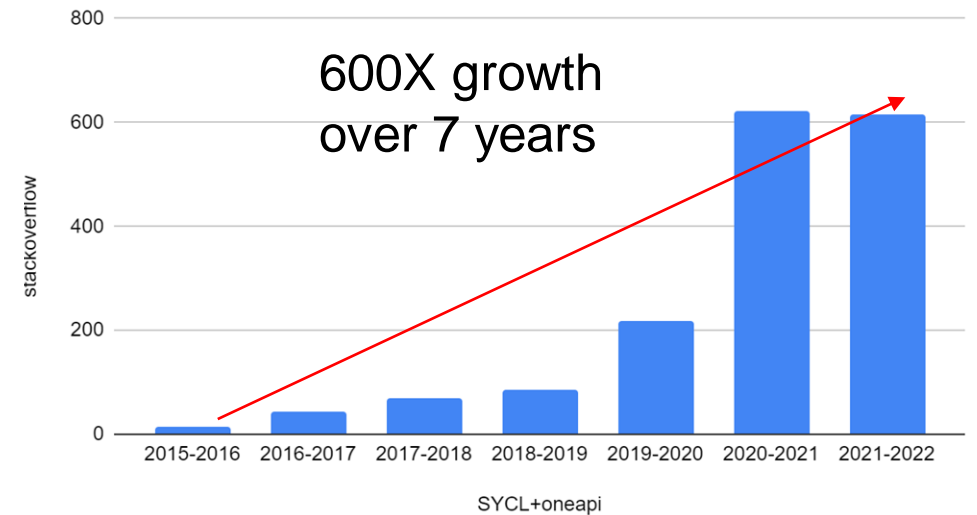


Scientific

SYCL user and developer Phenomenal Growth



stackoverflow questions annually on SYCL+oneapi



Some open-source SYCL implementations/prototypes on GitHub

Github SYCL source files is over 30000

"#include <CL/sycl.hpp>" / Pull requests Issues Marketplace Explore

Repositories	0
Code	11K
Commits	13
Issues	117
Discussions	3
Packages	0
Marketplace	0
Topics	0

21,446 code results

jfuentes/heterogeneous-computing-website
english/content/unit2/oneapi.md

```
18 {{< embed-pdf url="pdf/11-dpc++.pdf" >}}
19
20 **E>
21
22 *Exc
23
24
25 #inc
26 cons
27 usir
```

17.5k mentions of #include <CL/sycl.hpp> in Jan, 2022
21 K in April, 2022

"#include <SYCL/sycl.hpp>" / Pull requests Issues Marketplace Explore

Repositories	0
Code	1K
Commits	277
Issues	152
Discussions	3
Packages	0
Marketplace	0

8,804 code results

xfong/sycl
library/include/sycl_engine.hpp

```
1 #ifndef RUNTIME_INCLUDE_SYCL_SYCL_HPP_
2 #define RUNTIME_INCLUDE_SYCL_SYCL_HPP_
3 #include <CL/sycl.hpp>
4 namespace sycl = cl::sycl;
5
6 #endif // RUNTIME_INCLUDE_SYCL_SYCL_HPP_
7
8 #include "crossproduct.hpp"
```

8.2k mentions of #include <SYCL/sycl.hpp> in Jan, 2022

8.8 K in Apr, 2022

10k+ in August 2022



Market needs SYCL: easy to build SYCL on any device

2016



- 21 projects
- 2 implementations in work
- 3-4 platforms
- SYCL was a TSG of OpenCL
- 10 members attending TSG
- <10 GitHub stars
- <10 StackOverflow questions
- No Safety Critical
- No book, a few articles
- < 100 GitHub include code
- No Supercomputing presence
- No automotive
- No Clang/LLVM
- No common one-stop website
- No common teaching material
- No HPC

2022



- 147 projects
- >12 implementations in work
- 10+ platforms
- SYCL is independent WG of Khronos
- 20+ members attending WG
- >600 GitHub stars
- >600 StackOverflow questions
- Safety Critical Exploratory Group
- 1 book, many articles
- ~30000 GitHub include code
- SC BoF 5 years in a row
- Renesas R-Car
- Clang/LLVM DPC++ active branch
- [SYCL.tech](https://www.sycl.tech)
- SYCL Academy
- ~7 HPC systems

SYCL is mainstream

- Open Standards and Open Source implementations
- Open cross-company collaboration
- Co-design for all forms of extreme heterogeneity
- SYCL Survey published

Q* What language functionality would you like to see more broadly supported?

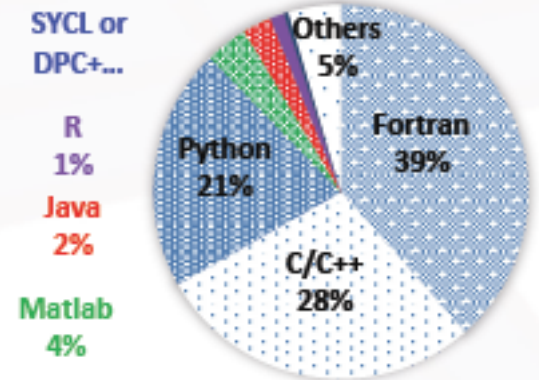
- C++20 as the core language
- C++23 as the core language
- Unified Shared Memory (USM)
- Unnamed kernel lambda functions
- Hierarchical Parallelism improvements
- None
- Other (Comment)

Market needs SYCL to Evolve (call to action)

- More workloads, NAMD, GROMACS, ROOTS
- Compile more ISO C++ as C++ advances
- **CTS, SDK, Compiler explorer**
- SYCL Graphs
- SPEC Accel/HPC Benchmarks
- Wikipedia, FAQ
- Safety Critical
- Educational videos
- Public CTS nodes
- More ecosystem, more libraries
- SYCL MLIR
- Public CTS does
- SYCL interpreter, Jupyter notebook
- More Vendor adoption
- Better tooling, profilers, debuggers, analyzers
- Data movement is still King, Will get worse with sparsity
- CUDA to SYCL conversion
- Parallelism Survey 2022 at NASA

SYCL Projects
2022

Programming Languages
(244 entries)



Others:

- Ruby (3 entries)
- Julia (2)
- CUDA/OpenMP (1)
- IDL (1)
- Tcl/tk (1)
- Shell scripting (1)
- Don't know (2)

- Fortran/C/C++ still dominate.
- Python is getting popular.
- SYCL/DPC++ is being explored (by FUN3D developers).

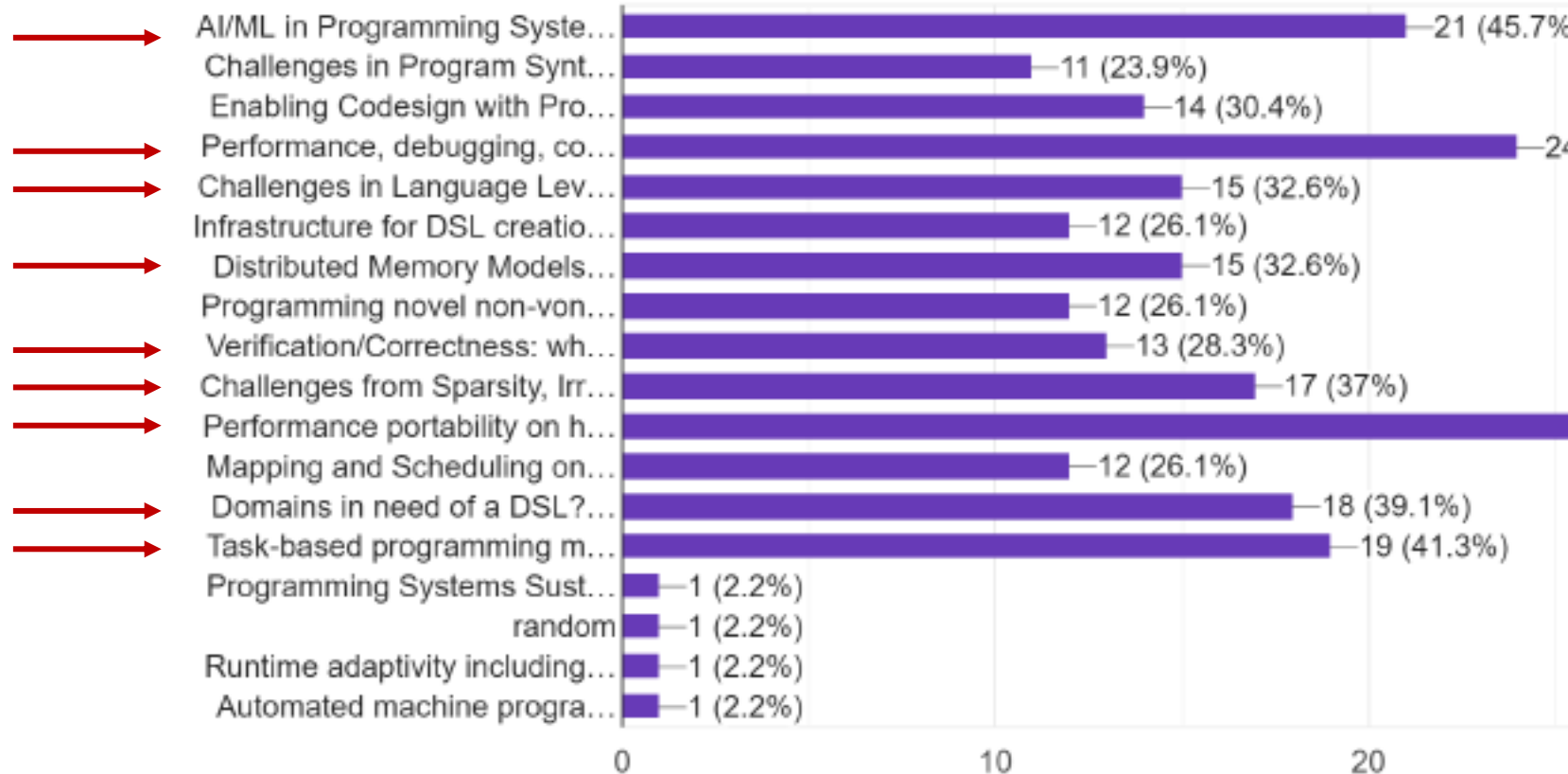
Market needs SYCL to succeed in democratizing

Q* Which of these provisional and vendor extensions would you like to see become Khronos extensions?

- Scoped Parallelism: [View via hipSYCL](#)
- Multi-device queue: [View via hipSYCL](#)
- Command group properties: [View via hipSYCL](#)
- Buffer-USM interop: [View via hipSYCL](#)
- Pipes: [View via Intel](#)
- Accessor restrict property: [View via Intel](#)
- Accessor properties: [View via Intel](#)
- Enqueue barrier: [View via Intel](#)
- Bfloat16: [View via Intel](#)
- Properties: [View via Intel](#)
- None

From the recent Programming Systems Research Forum at DOE Feb 2022

Please rate your interest in participating in each of these potential breakout group topics. Pick at most six (6) topics. We are saving two breakout groups free to suggest a topic in the future. 46 responses



- Compilers with AI/ML
- Performance debugging
- Language Parallelism
- Distributed Memory
- Verification Correctness
- Sparsity Irregularity
- Performance Portability
- Domains needing DSL
- Task Based Programming

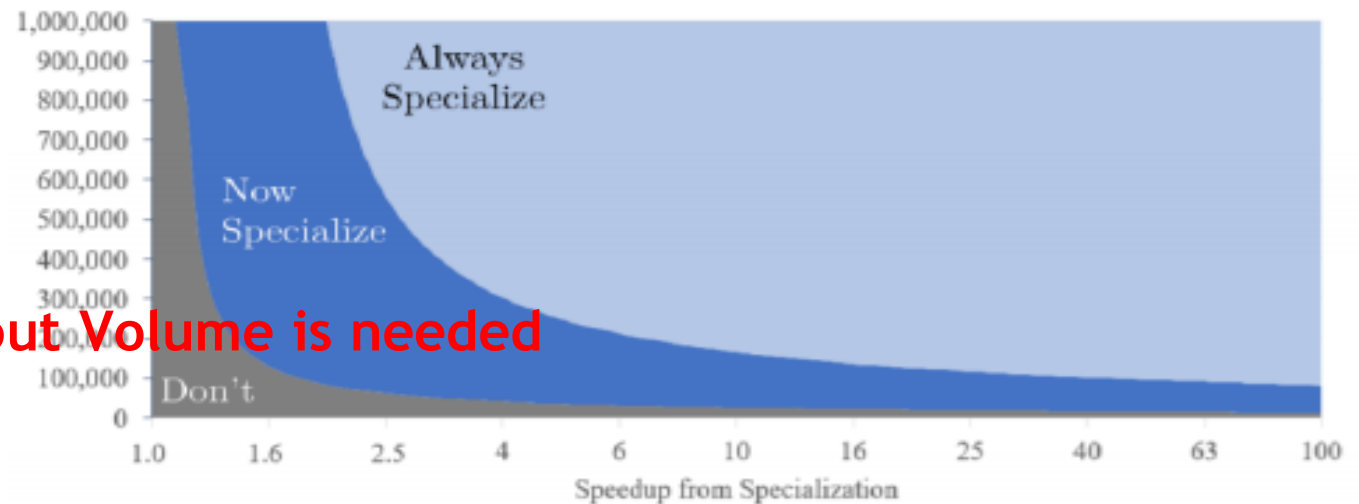
From ASCR Workshop Mar 2021

ASCR Workshop on Reimagining Codesign

<https://www.orau.gov/ASCR-CoDesign/>

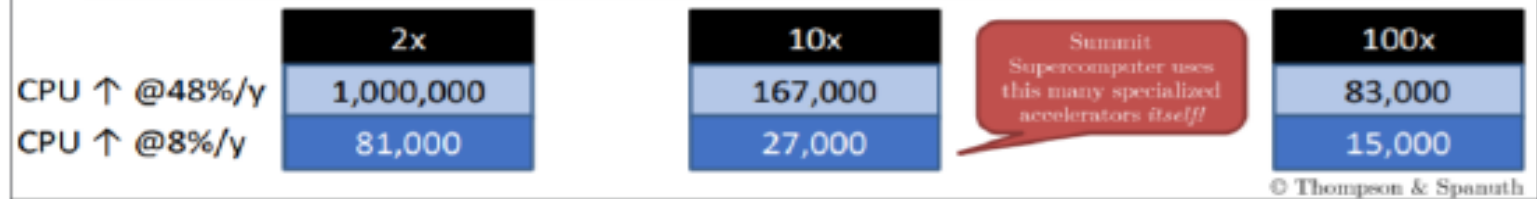
Specialization is more attractive now
that CPUs are improving slowly

Volume needed to be worth designing a specialized chip

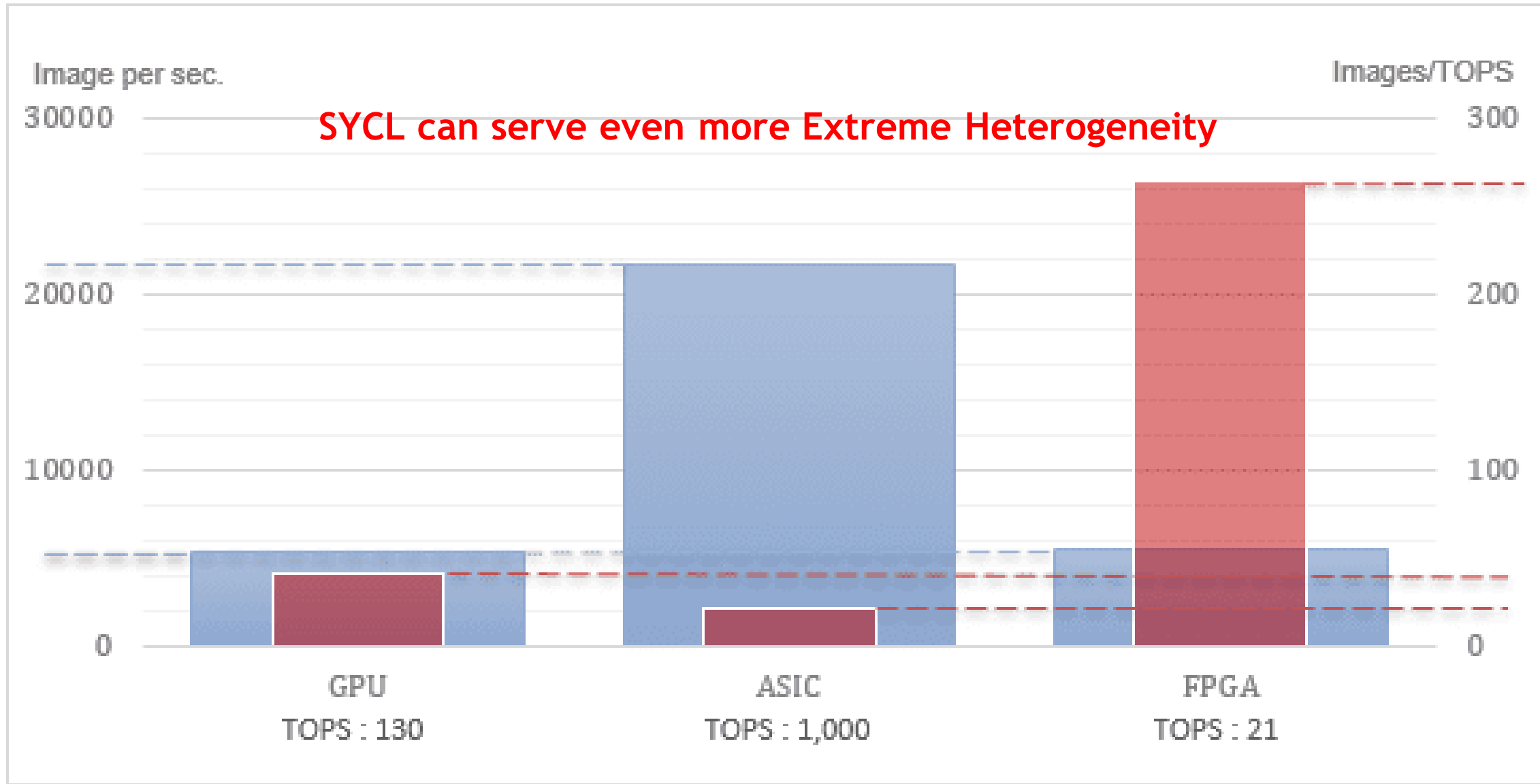


From Neil Thompson's keynote:

Specialization is more attractive now but Volume is needed



From Flops to TOPS in ML



<https://www.kdnuggets.com/2020/05/tops-just-hype-dark-ai-silicon-disguise.html>

From Torsten Hoefler talk: Stop counting and start moving (data) Data movement is all you need

DATA MOVEMENT IS ALL YOU NEED: A CASE STUDY ON OPTIMIZING TRANSFORMERS

Andrei Ivanov^{1*}, Nikoli Dryden^{1*}, Tal Ben-Nun¹, Shigang Li¹, Torsten Hoefler¹

ABSTRACT

Transformers are one of the most important machine learning workloads today. Training one is a very compute-intensive task, often taking days or weeks, and significant attention has been given to optimizing transformers. Despite this, existing implementations do not efficiently utilize GPUs. We find that data movement is the key bottleneck when training. Due to Amdahl's Law and massive improvements in compute performance, training has now become memory-bound. Further, existing frameworks use suboptimal data layouts. Using these insights, we present a recipe for globally optimizing data movement in transformers. We reduce data movement by up to 22.91% and overall achieve a 1.30x performance improvement over state-of-the-art frameworks when training a BERT encoder layer and 1.19x for the entire BERT. Our approach is applicable more broadly to optimizing deep neural networks, and offers insight into how to tackle emerging performance bottlenecks.

1 INTRODUCTION

Transformers (Vaswani et al., 2017) are a class of deep neural network architecture for sequence transduction (Graves, 2012), with similar applicability as RNNs (Rumelhart et al., 1986) and LSTMs (Hochreiter & Schmidhuber, 1997). They have recently had a major impact on natural language processing (NLP), including language modeling (Radford et al., 2018; Wang et al., 2018, 2019), question-answering (Rajpurban et al., 2018), and text summarization (Goyal et al., 2017). Microsoft, 2020b), number of training samples (Raffel et al., 2019; Liu et al., 2019), and total iterations (Liu et al., 2019; Kaplan et al., 2020). These all significantly increase compute requirements. Indeed, transformers are becoming the dominant task for machine learning compute where training a model can cost tens of thousands to millions of dollars, even cause environmental concerns (Strubell et al., 2019). These trends will only accelerate with pushes toward models with tens of billions to trillions of parameters (N



072v3 [cs.LG] 8 Nov 2021

02.00554v1 [cs.LG] 31 Jan 2021

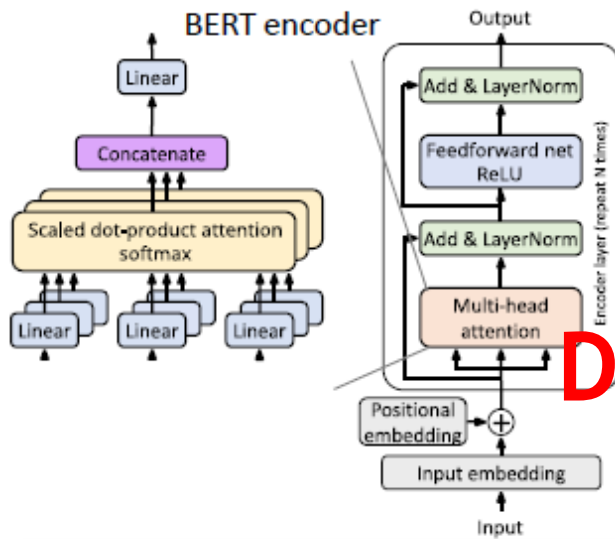
Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks

TORSTEN HOEFLER, ETH Zürich, Switzerland
 DAN ALISTARH, IST Austria, Austria
 TAL BEN-NUN, ETH Zürich, Switzerland
 NIKOLI DRYDEN, ETH Zürich, Switzerland
 ALEXANDRA PESTE, IST Austria, Austria

The growing energy and performance costs of deep learning have driven the community to reduce the size of neural networks by selectively pruning components. Similarly to their biological counterparts, sparse networks generalize just as well, if not better than, the original dense networks. Sparsity can reduce the memory footprint of regular networks to fit mobile devices, as well as shorten training time for ever growing networks. In this paper, we survey prior work on sparsity in deep learning and provide an extensive tutorial of sparsification for both inference and training. We describe approaches to remove and add elements of neural networks, different training strategies to achieve model sparsity, and mechanisms to exploit sparsity in practice. Our work distills ideas from more than 300 research papers and provides guidance to practitioners who wish to utilize sparsity today, as well as to researchers whose goal is to push the frontier forward. We include the necessary background on mathematical methods in sparsification, describe phenomena such as early structure adaptation, the intricate relations between sparsity and the training process, and show techniques for achieving acceleration on real hardware. We also define a metric of pruned parameter efficiency that could serve as a baseline for comparison of different sparse networks. We close by speculating on how sparsity can improve future workloads and outline major open problems in the field.

The supreme goal of all theory is to make the irreducible basic elements as simple and as few as possible without having to surrender the adequate representation of a single datum of experience -
 Albert Einstein, 1933

Even Deep Learning will be limited by data movement (arXiv:2007.00072)



Operator class	% flop	% Runtime
Tensor contraction	99.80	61.0
Statistical normalization	0.17	25.5
Element-wise	0.03	13.5

Data Movement is still King

Our performance improvement for BERT-large

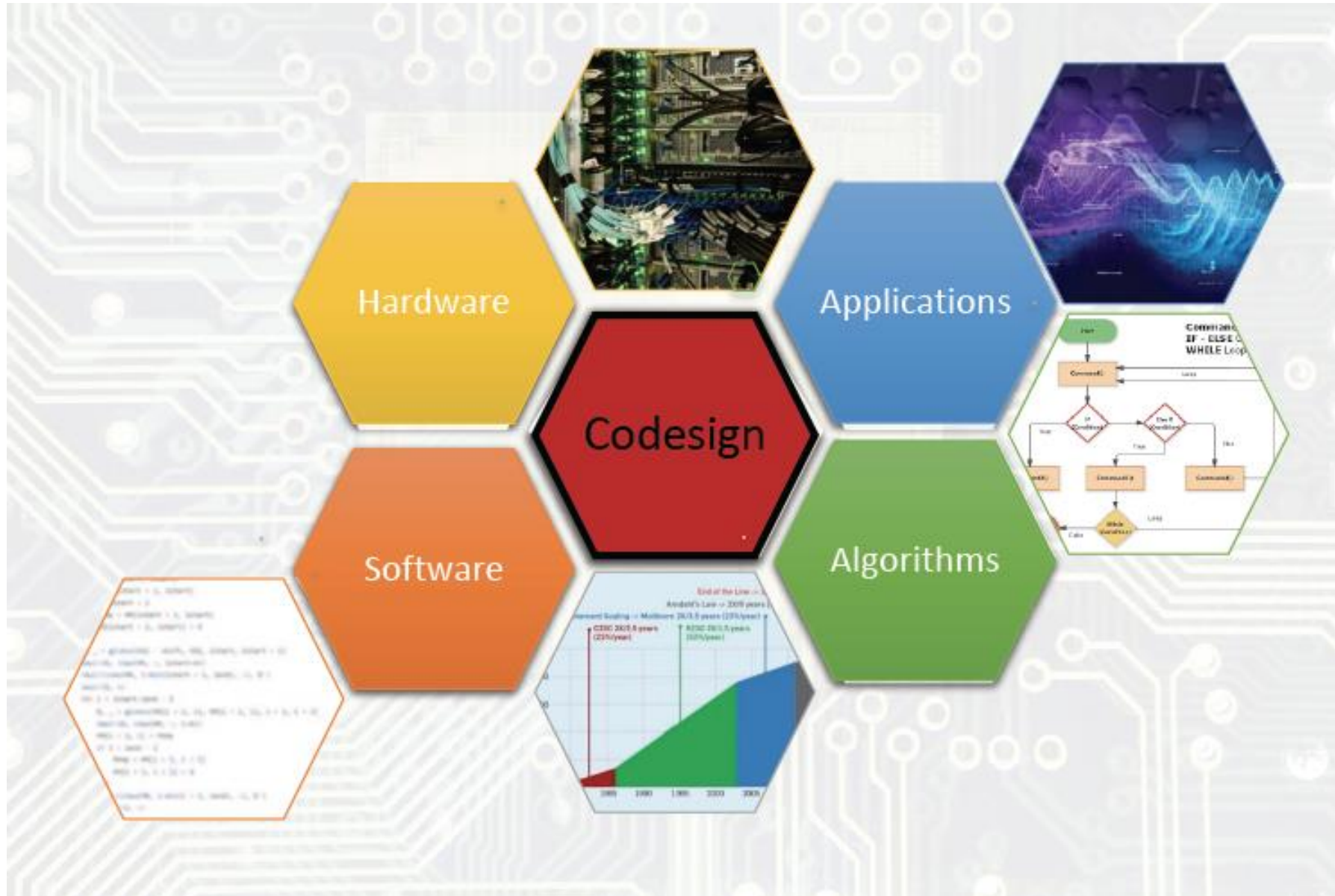
- 30% over PyTorch
- 20% over Tensorflow + XLA
- 8% over DeepSpeed

est. savings on AWS over PyTorch:
\$85k for BERT, \$3.6M GPT-3

OpenAI booth at NeurIPS 2019 in Vancouver, Canada
 Image Credit: Khan Johnson / VentureBeat

Last week, OpenAI published a paper detailing GPT-3, a machine learning model that achieves strong results on a number of natural language benchmarks. A **175 billion parameters** where a parameter affects data's prominence in an overall prediction, it's the largest of its kind. And with a memory size exceeding 350GB, it's one of the priciest, costing an estimated **\$12 million to train.**

We need Codesign with extreme Heterogeneity



Future SYCL: Emerging transformative technologies



- Chiplets and Superchips
- Licensable IP for Server-class processors (ARM)
- Open Source Hardware and Open Silicon Compilers (RISC-V)
- Photonic Resource Disaggregation (Ayar Labs TeraPhy, ARPA-E ENLITENED, and DARPA PIPES)
- Standardized Accelerator Interfaces (CCIX, Coherent PCIe, CXL, UClle, HSA, Intel Level-0 & DSA)
- Advanced Hardware description Languages and Hardware generator
- New Accelerators (Ex: AMD/Xilinx Versal, SambaNova, Graphcore, Cerebras...)
- Advanced Packaging Technologies for Heterogeneous Integration (HIR) <https://eps.ieee.org/technology/heterogeneous-integration-roadmap/2019-edition.html>
- Open Source and Extensible Compiler Frameworks (LLVM, MLIR)
- AI integrated applications
- Programming Abstractions, Frameworks, and Languages (SYCL, [Reimagining Codesign 2021 Position Papers \(ossl.gos\)](#))

Parallel Industry Initiatives



C++11



C++14



C++17



C++20



C++23



SYCL 1.2
C++11 Single source
programming



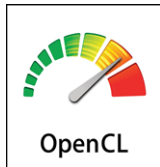
SYCL 1.2.1
C++11 Single source
programming



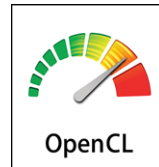
SYCL 2020
C++17 Single source
programming
Many backend options



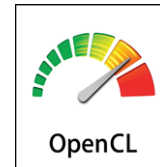
SYCL 202X
C++20 Single source
programming
Many backend options



OpenCL 1.2
OpenCL C Kernel
Language



OpenCL 2.1
SPIR-V in Core



OpenCL 2.2



OpenCL 3.0



2011

2015

2017

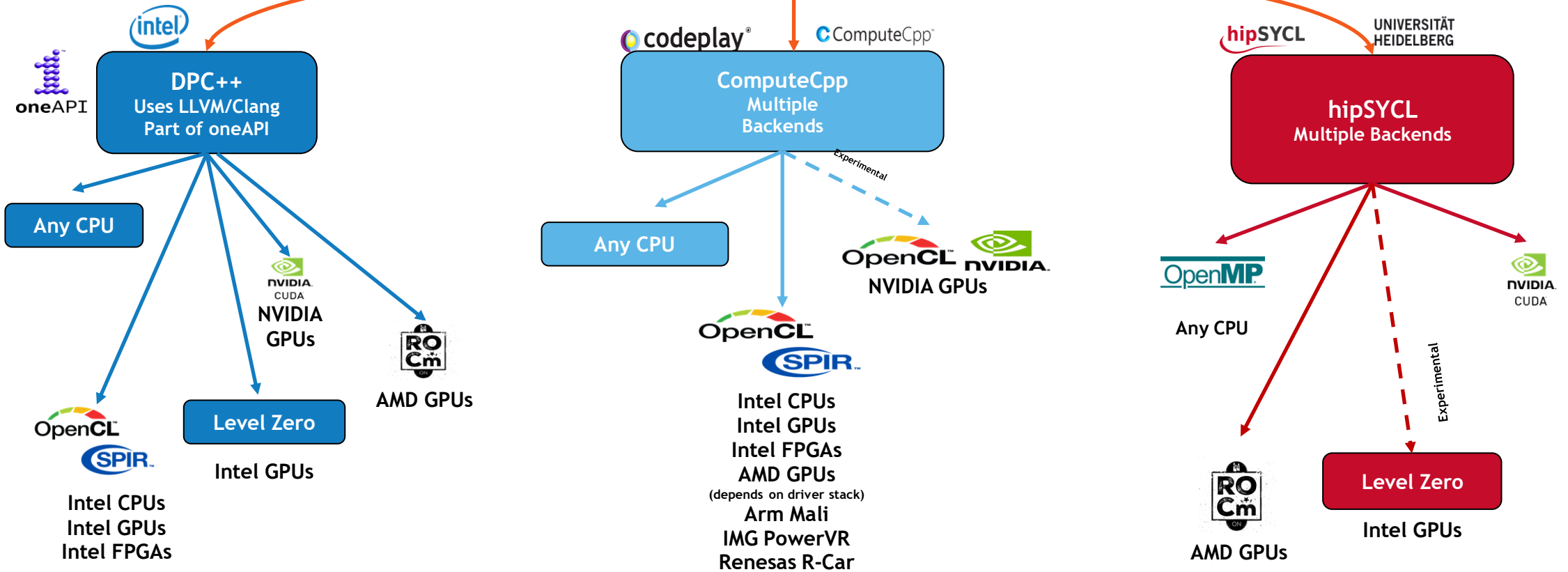
2020

202X

SYCL Implementations in Development (2022/05/01)

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

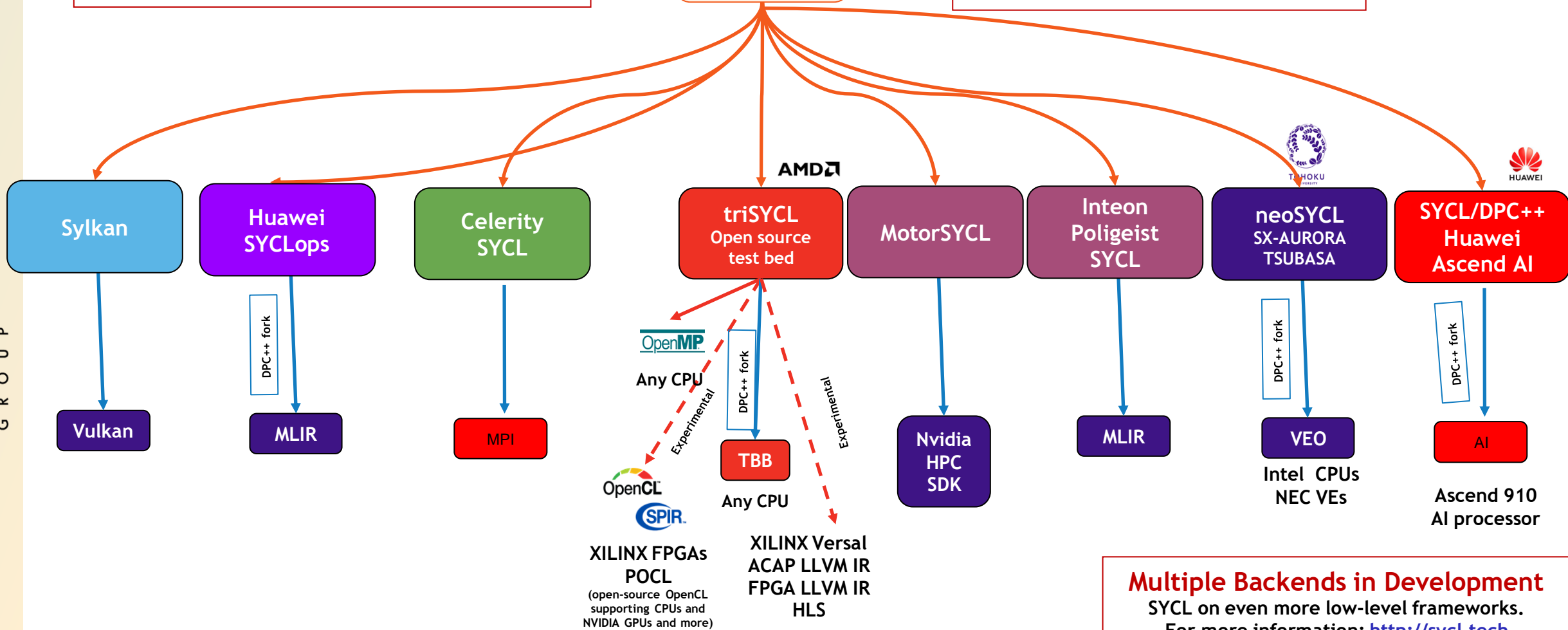
SYCL enables Khronos to influence ISO C++ to (eventually) support heterogeneous compute



SYCL Experimental Development (2022/05/01)

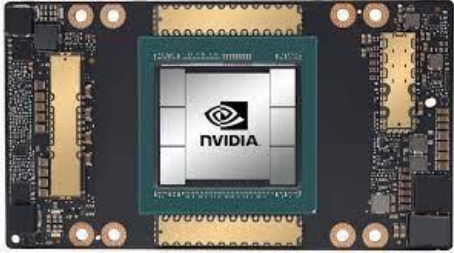
SYCL, OpenCL and SPIR-V, as open industry standards, enable flexible integration and deployment of multiple acceleration technologies

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



Multiple Backends in Development
 SYCL on even more low-level frameworks.
 For more information: <http://sycl.tech>

Migration advice by Platform (2022/05/05): For a device kind, which SYCL compiler?



ALTERA



XILINX

- NVIDIA GPUs
 - DPC++ supported today by Codeplay, open-source, optimized
- AMD GPUs
 - hipSYCL open-source today is better and is supporting European Processor initiative in LUMI & Karolina
 - DPC++ experimental support today by Codeplay, open-source
- Intel GPUs
 - DPC++ supported by Intel
- New hardware:
 - Codeplay supports a range of new hardware with ComputeAorta + ComputeCpp, including RISC-V custom hardware and GPU IP from Imagination Technologies and ARM, as well as Renesas R-Car for automotive
 - Also can do it on their own (DIY)
- FPGA
 - DPC++ compiler for Intel FPGAs
 - AMD/Xilinx has triSYCL prototype for FPGA & CGRA
- AI/ML/NN
 - All the major implementations/companies can customize support or you can build it DIY
- RISC-V
 - Codeplay has developed ACORAN stack for RISC-V as an accelerator over X86 CPU
- CPU
 - All major SYCL implementations

SYCL Ecosystem, Research and Benchmarks 2022/05/01

Implementations

neoSYCL
SX-AURORA TSUBASA

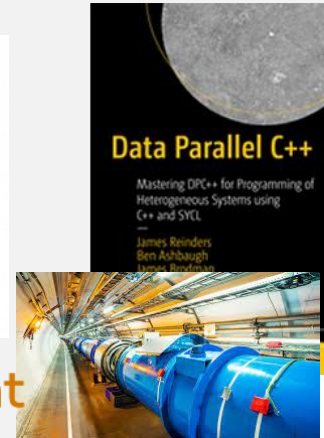


Research

C Celerity
High-level C++ for Accelerator Clusters

ECIP
EXASCALE COMPUTING PROJECT

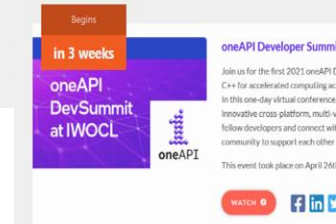
Tutorials/Benchmarks/ Books



Future Workshops



Distributed and Heterogeneous Programming in C++ (DHPCC++22)



oneAPI Developer Summit @ SYCLcon 2022

Recent Workshops



Portable Heterogeneous Programming with SYCL (PHPS22)



SYCL Workshop with ENCCS for the Karolina Supercomputer

oneAPI



PYTORCH



ATLAS
EXPERIMENT

Flashlight



HUAWEI ComputeCpp™

hipSYCL



STREAM HPC

Argonne NATIONAL LABORATORY

arm

QUALCOMM

AMD

SYCL



intel



codeplay

SAMSUNG



University of BRISTOL

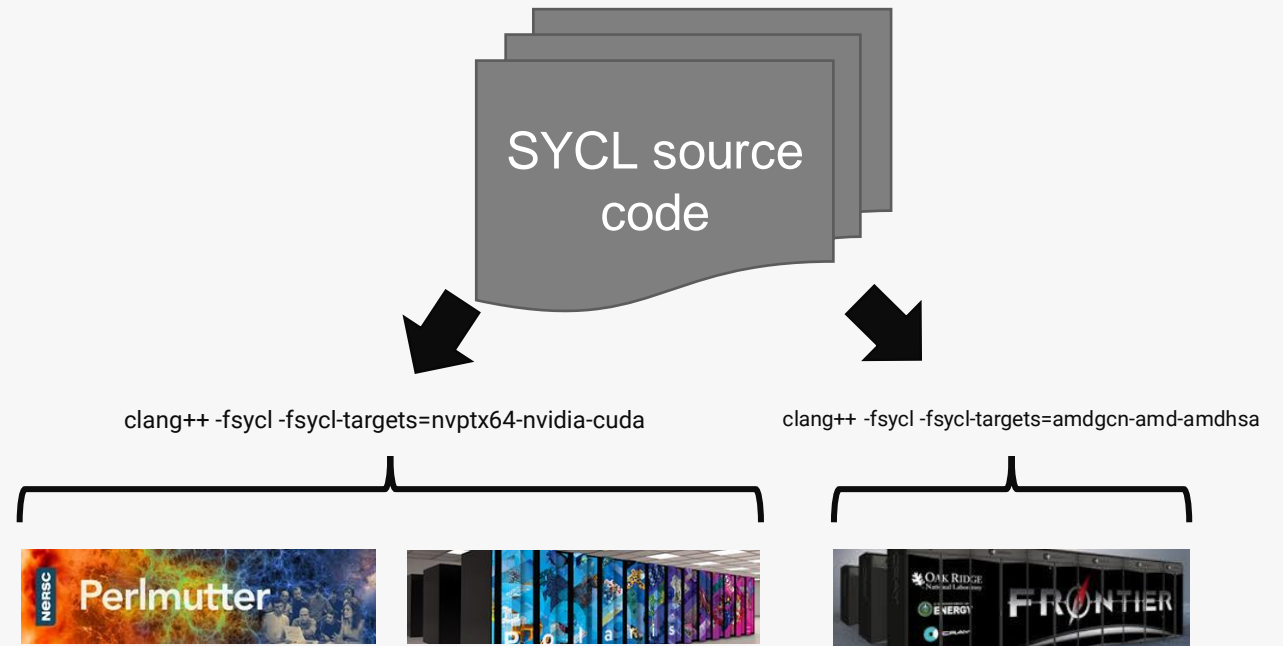
Working Group Members

Creative Commons Attribution 4.0 International License

Nvidia and AMD Support in DPC++

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

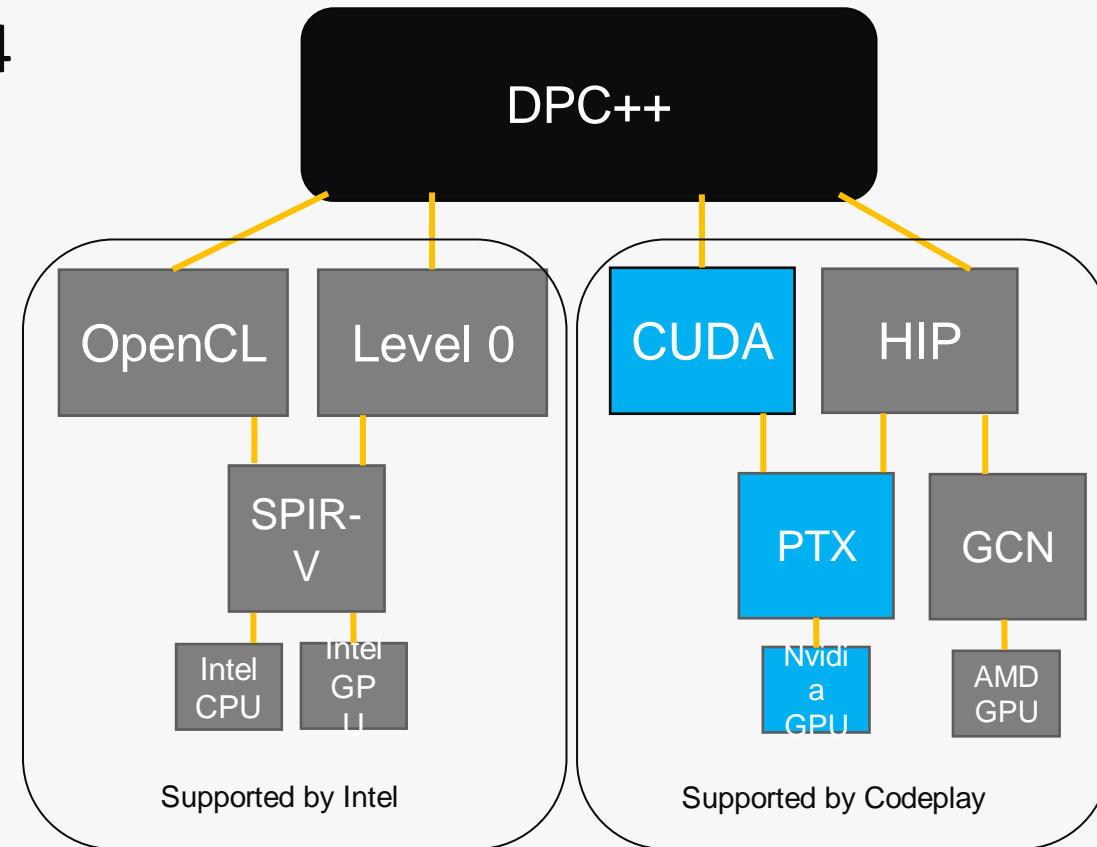
Different targets using a simple compiler flag



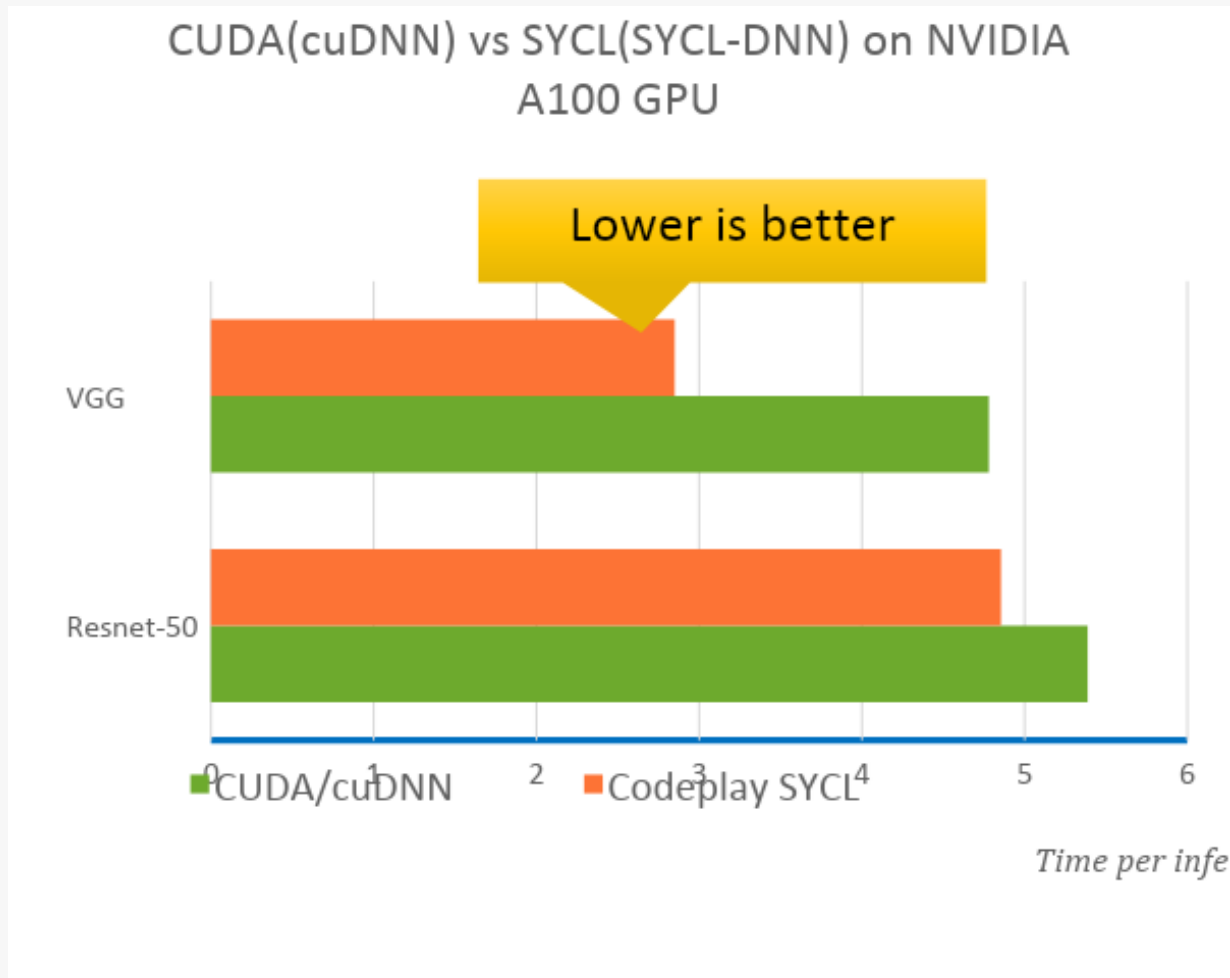
<https://www.codeplay.com/oneapiforcuda>
Resources for AMD coming soon

CUDA as a backend

- The DPC++ runtime currently implements 4 SYCL backends:
 - OpenCL
 - Level Zero
 - CUDA
 - HIP
- Moving up a level of abstraction, you untether your codebase from specific hardware
- Developers have flexibility to prioritize performance

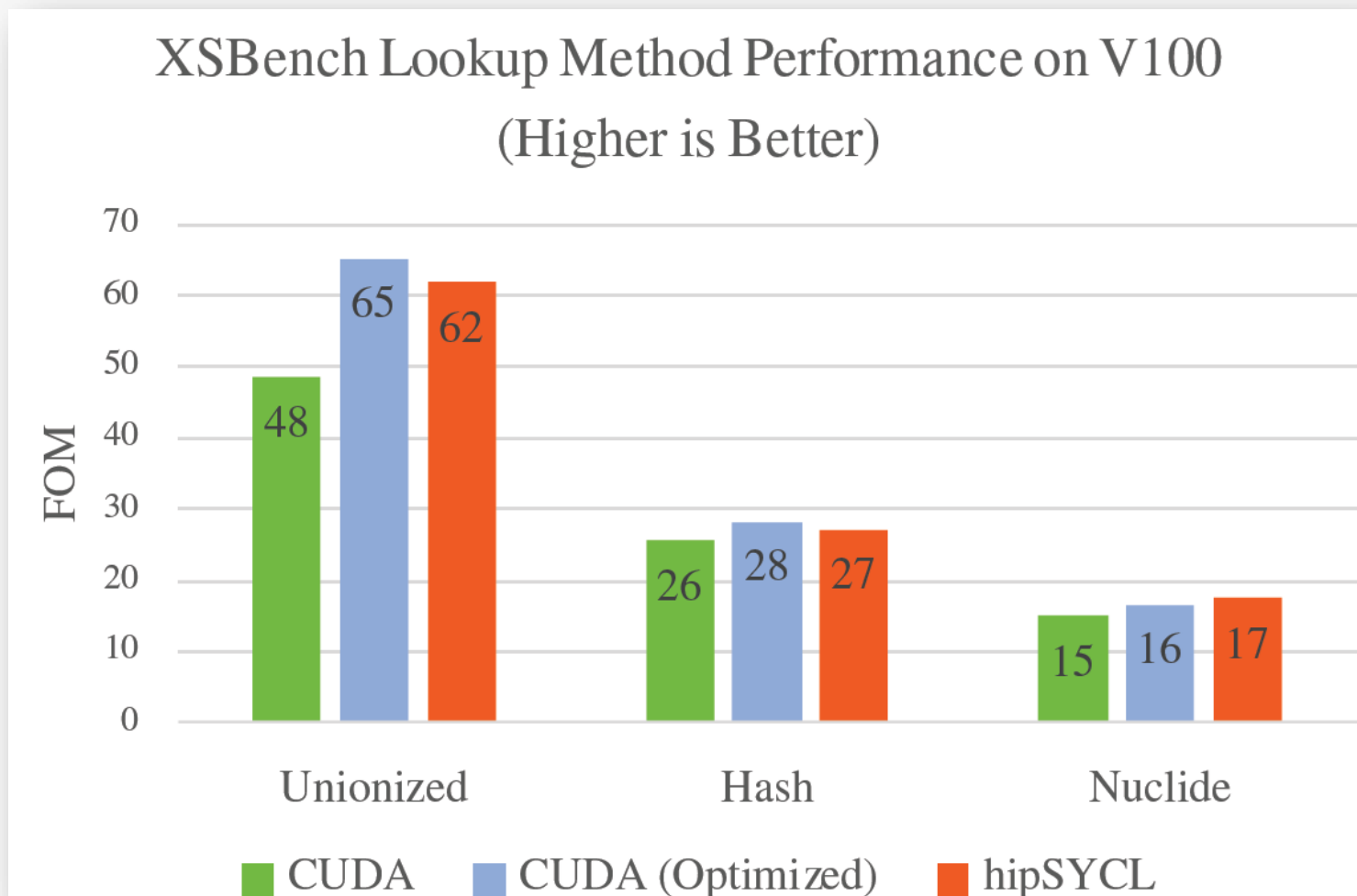


Performance comparison



- SYCL uses the same C++ performance model as CUDA, so it achieves very similar performance for the same code
- SYCL makes it easy to write *parameterizable* code that adapts the algorithms to underlying hardware: this enables automatic optimizations that increase performance

Argonne National Labs Comparison

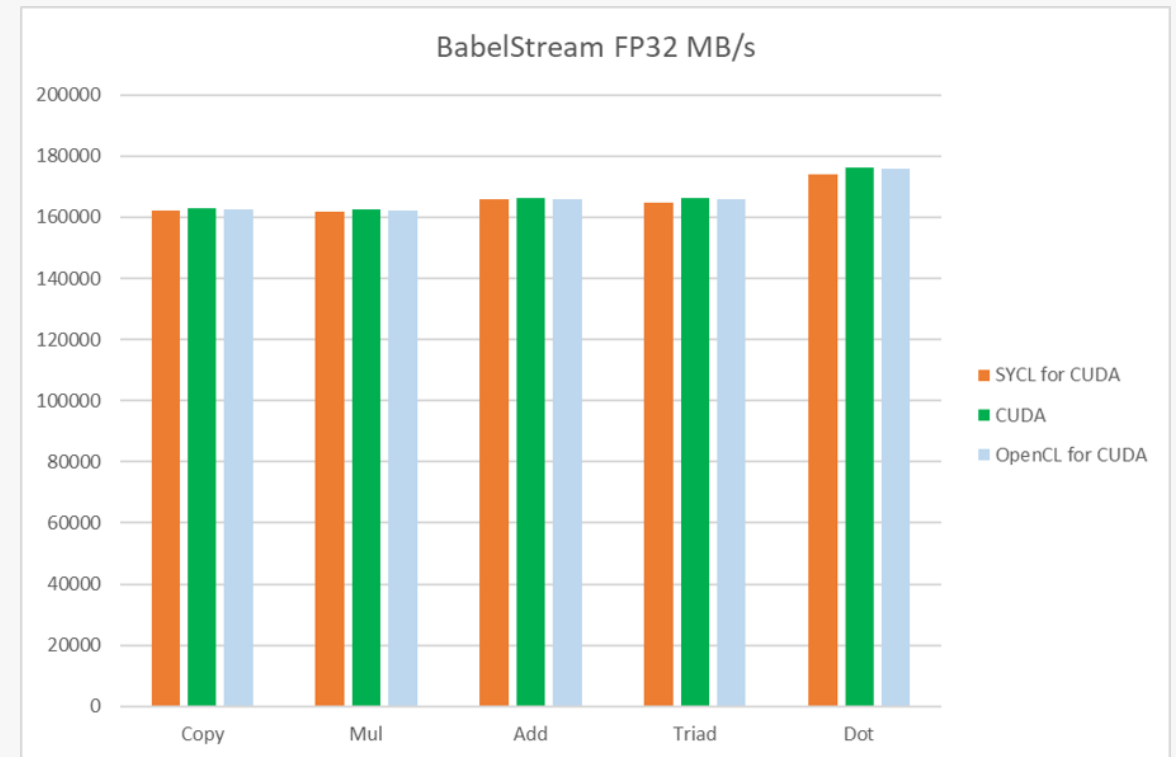


SYCL achieves
equivalent
performance to
Optimized CUDA

[Presented at IWOCCL, April'20](#)

DPC++ performance on Nvidia GPUs

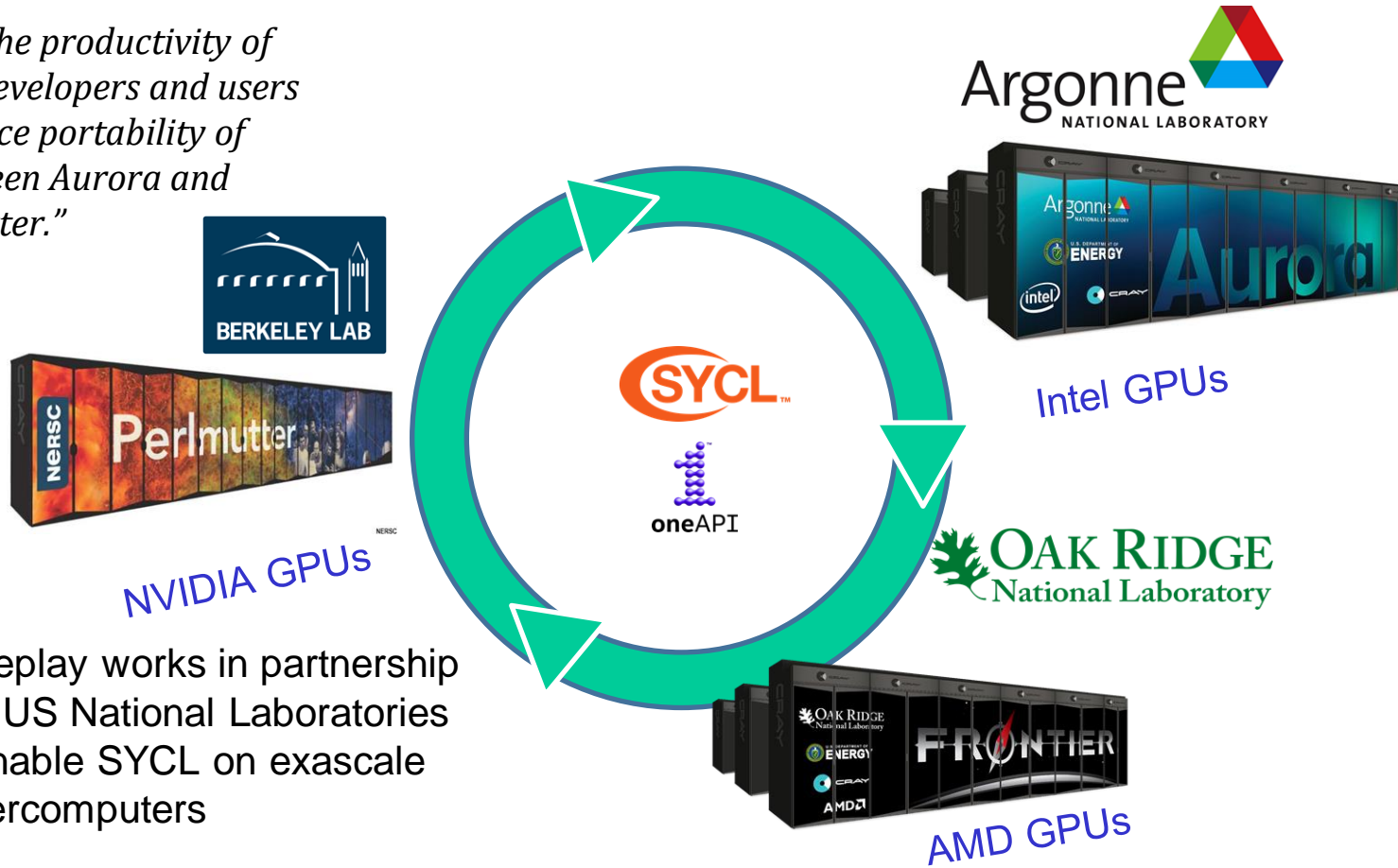
- This graph compares the BabelStream benchmarks results for:
 - Native CUDA code
 - OpenCL code
 - SYCL code using the CUDA backend
- Run on GeForce GTX 980 with CUDA 10.1
- Minimal SYCL overhead



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

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



Towards a Portable Pipeline in Drug Discovery

- The LIGATE project aims at building a portable drug discovery pipeline
 - Funded from the European High-Performance Computing Joint Undertaking Joint Undertaking (JU)
- HPC is heading toward specialization and extreme heterogeneity
- SYCL enables to write platform independent code, while keeping native-comparable performance



<https://www.ligateproject.eu/>

SYCL as a universal programming model for HPC

Starting with US National Labs

Across Europe, Asia are many Petascale and pre-exascale systems

- With many variety of CPUs GPUs FPGAs, custom devices
- Often with interconnected usage agreements
- Europe EPI: **hipSYCL in Leonardo, Lumi and Karolina**



Easier, Short Path to Heterogeneous Programming

Open Source CUDA* to SYCL* Code Migration Tool - Project SYCLomatic, DPCT

SYCL* with oneAPI open, cross-architecture, standards-based programming

Allows developers to expand the value of their investments across architectures

Provides choice in hardware & freedom from proprietary, single-vendor lock-in

Intel is providing a CUDA* to SYCL* migration tool: Project SYCLomatic

Enables developers a productive path to create single-source, portable code for hardware targets regardless of vendor

Simplifies development while delivering performance, reduces time & costs for code maintenance

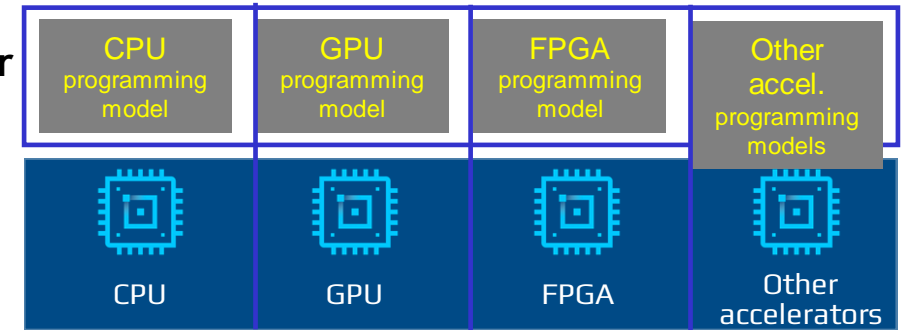
A community to share, collaborate & contribute software technologies

Available on GitHub now

github.com/oneapi-src/SYCLomatic <https://github.com/oneapi-src/SYCLomatic>

Use the tool, please provide feedback!

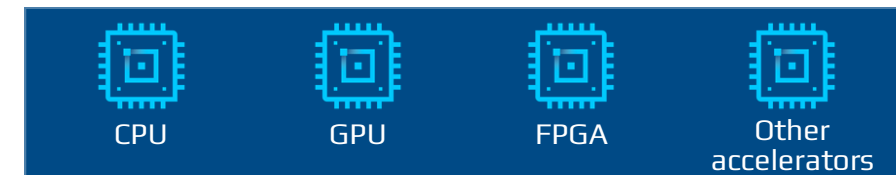
Simplify Heterogeneous Development
From Diverse Programming Approaches



To Productive, Performant
Cross-architecture, Cross-vendor Programming



Project SYCLomatic: CUDA to SYCL Code Migration Tool



SYCLomatic Tool Usage Workflow

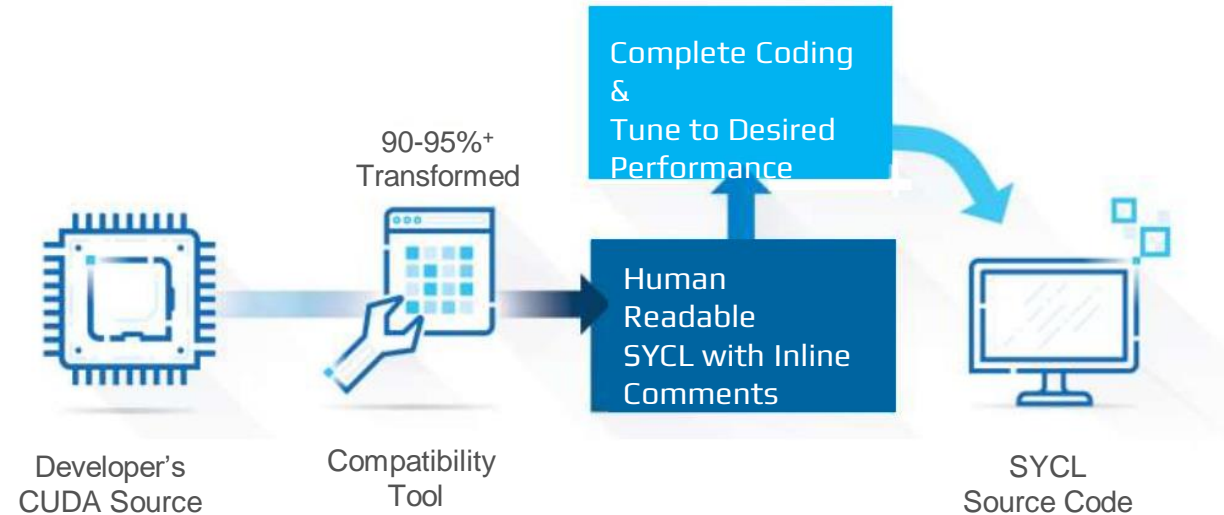
(newly open sourced by the end of May)

Collect compilation options of the Developer's CUDA* source from project build scripts, eg. Makefile, vcxproj file

Assist developers migrating code written in CUDA to SYCL by generating SYCL code wherever possible

Typically, 90%-95%+ of CUDA code automatically migrates to SYCL code

Inline comments are provided to help developer complete and tune the code



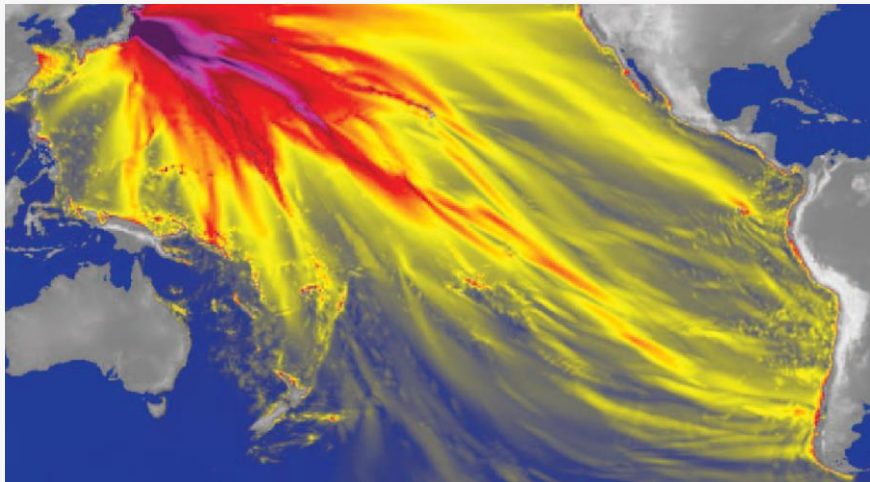
* All estimates as of September 2021. Based on measurements on a set of 70 HPC benchmarks and samples, with examples like Rodas, ScaLAPACK, and others. Results may vary.

*Other names and brands may be claimed as the property of others.

Case Study: Zuse Institute Berlin

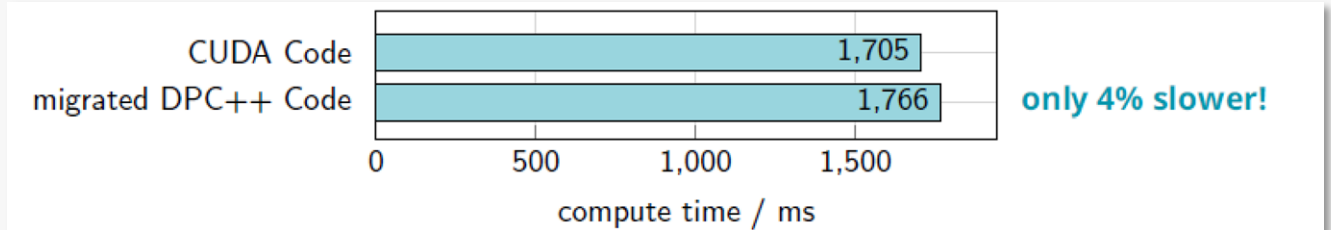


From CUDA to DPC++ and back to Nvidia GPUs... and FPGAs
A oneAPI case study with the tsunami simulation easyWave



- Originally written for NVIDIA GPUs with CUDA
- Auto-converted to SYCL (DPC++) with Intel's Compatibility Tool
- Resulting SYCL code runs on: NVIDIA GPU, Intel GPU, Intel CPU, Intel FPGA [+ any new processor with SYCL support]

Even on NVIDIA GPU, SYCL code is only 4% slower (with Codeplay developed open-source compiler)

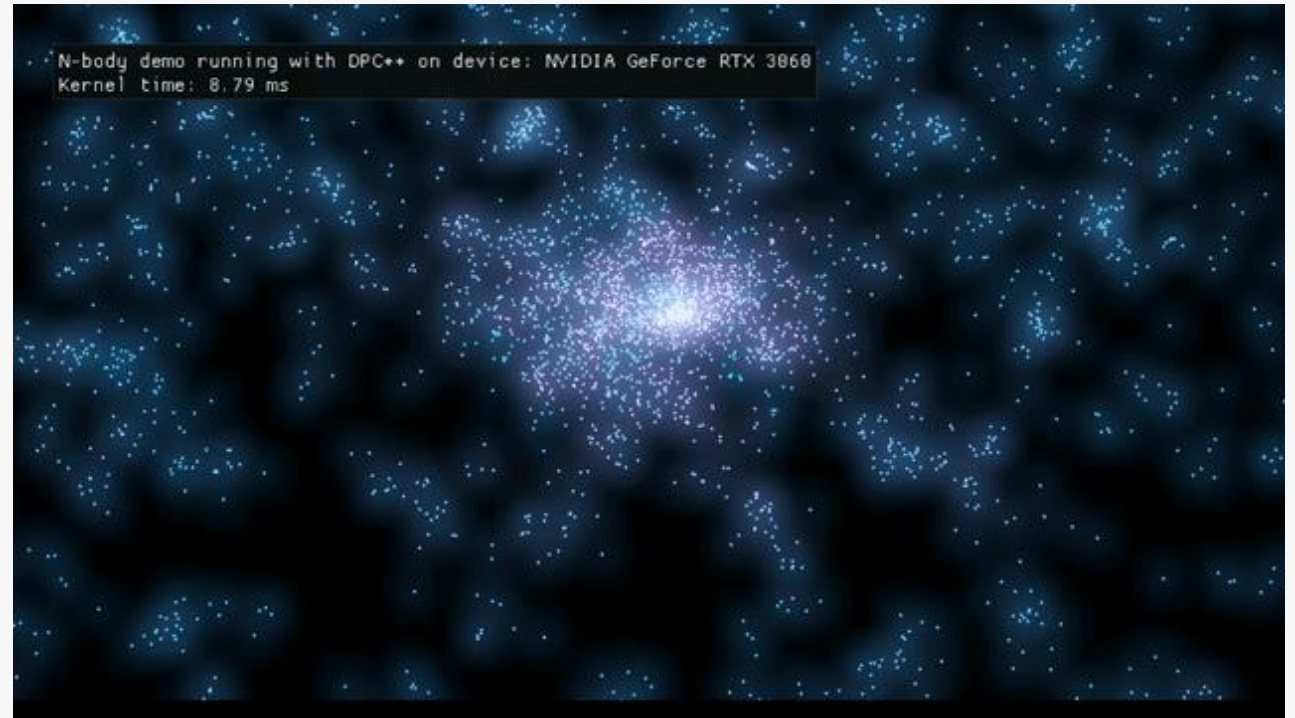


N-Body

- Simulates gravitational interaction in a fictional galaxy

$$\vec{F}_i = - \sum_{i \neq j} G \frac{(\vec{r}_i - \vec{r}_j)}{|\vec{r}_i - \vec{r}_j|^3}$$

<https://github.com/codeplaysoftware/cuda-to-sycl-nbody>



Familiar Standard C++ Code

CUDA

```
void DiskGalaxySimulator::stepSim() {  
    // Compute updated positions  
    constexpr int wg_size = 256;  
    int nblocks = ((getNumParticles() - 1) / wg_size) + 1;  
  
    // Profiling info - rather than using the CUDA event recording  
    // approach, we are instead measuring the time from before kernel  
    // submission until host synchronization. This is more portable via  
    // dpct.  
    auto start = std::chrono::steady_clock::now();  
    for (size_t i = 0; i < params.maxIterationsPerFrame; i++) {  
        particle_interaction<<<nblocks, wg_size>>>(pos_d, pos_next_d, vel_d,  
                                                    params);  
        std::swap(pos_d, pos_next_d);  
    }  
    gpuErrchk(cudaDeviceSynchronize());  
    auto stop = std::chrono::steady_clock::now();  
    lastStepTime =  
        std::chrono::duration<float, std::milli>(stop - start)  
            .count();  
  
    // Sync data  
    recvFromDevice();  
}
```

Code changes
are minimal

SYCL

```
void DiskGalaxySimulator::stepSim() {  
    // Compute updated positions  
    constexpr int wg_size = 256;  
    int nblocks = ((getNumParticles() - 1) / wg_size) + 1;  
  
    auto start = std::chrono::steady_clock::now();  
    for (size_t i = 0; i < params.maxIterationsPerFrame; i++) {  
        dpct::get_default_queue().submit([&](sycl::handler &cgh) {  
            auto pos_d_ct0 = pos_d;  
            auto pos_next_d_ct1 = pos_next_d;  
            auto vel_d_ct2 = vel_d;  
            auto params_ct3 = params;  
  
            cgh.parallel_for(sycl::nd_range<1>(sycl::range<1>(nblocks) *  
                                              sycl::range<1>(wg_size),  
                                              sycl::range<1>(wg_size)),  
                            [=](sycl::nd_item<1> item_ct1) {  
                                particle_interaction(pos_d_ct0, pos_next_d_ct1,  
                                                    vel_d_ct2, params_ct3,  
                                                    item_ct1);  
                            });  
        });  
        std::swap(pos_d, pos_next_d);  
    }  
    gpuErrchk((dpct::get_current_device()).queues_wait_and_throw(), 0);  
    auto stop = std::chrono::steady_clock::now();  
    lastStepTime =  
        std::chrono::duration<float, std::milli>(stop - start)  
            .count();  
  
    // Sync data  
    recvFromDevice();  
}
```

Performance Achieved

```
N-body demo running with CUDA on device: NVIDIA GeForce RTX 3060  
Kernel time: 6.47 ms
```

```
N-body demo running with DPC++ on device: NVIDIA GeForce RTX 3060  
Kernel time: 6.45 ms
```

www.codeplay.com/oneapiforcuda/



RISC-V

The Emergence of RISC-V

Semico Research's New Report Predicts There Will Be 25 Billion RISC-V-Based AI SoCs By 2027 | Rich Wawrzyniak, Semico Research Corporation

RISC-V

- What is RISC-V?
 - FOSS RISC Instruction Set Architecture
 - No license required
 - Simple base ISA (~40 instructions)
 - Multiple optional extensions
- Flexible Systems
 - Hard IP Blocks
 - Multiple RISC-V Cores



<https://riscv.org/>

The RISC-V Ecosystem

- RISC-V's place in the ecosystem
 - Growing ecosystem around RISC-V from different vendors
 - More single ISA solutions e.g. RISC-V host, RISC-V accelerator
 - Often used with bespoke components e.g. Convolution hardware
- RISC-V is growing rapidly
 - Over 1000 RISC-V members
 - Multiple large companies working with it
 - Likely to be used for a lot of new growth areas such as AI

RISC-V Vector Extension Driving AI and More

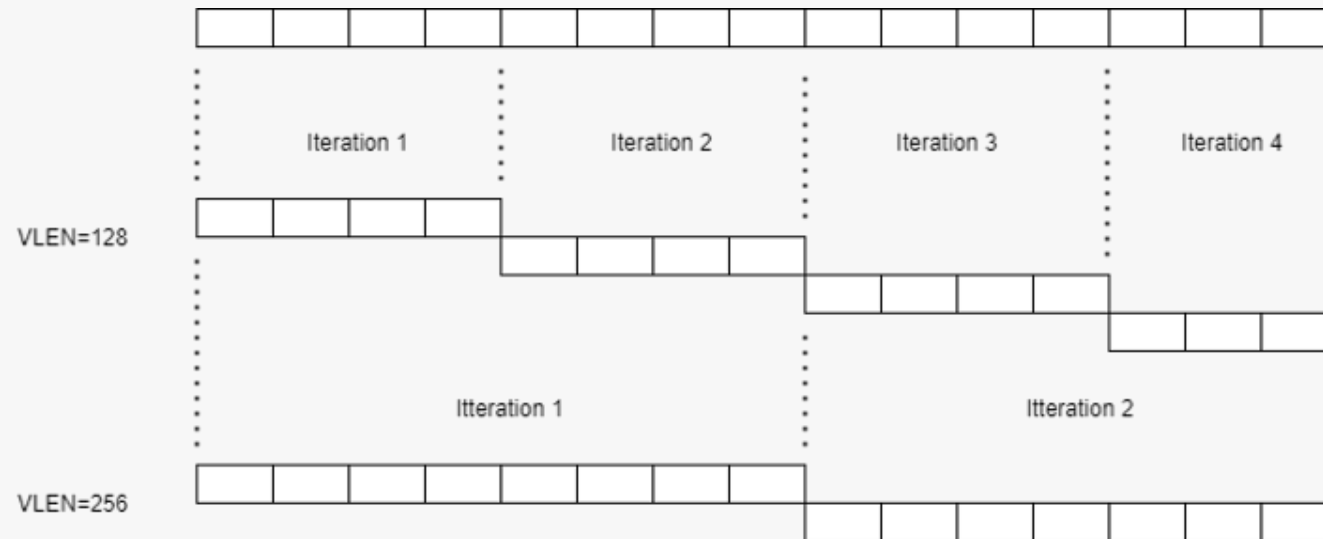
Proposed RISC-V vector instructions crank up computing power on small devices | Agam Shah, The Register

RISC-V Vector Extension: Scalable Vectors

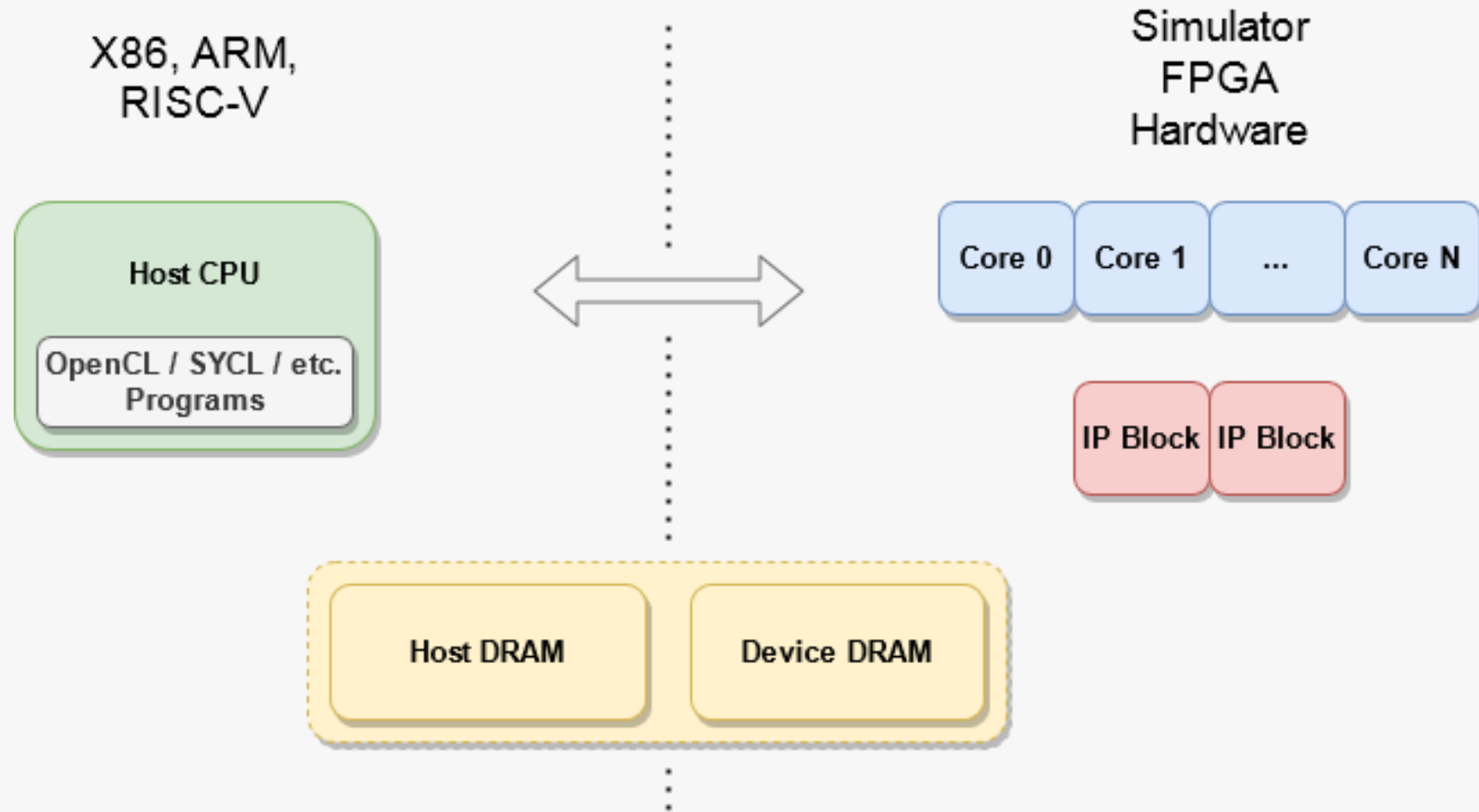
- Introduces vector support for different datatypes
 - half, float, double, etc
- Vector width operations configurable at runtime
 - The RISC-V V-extension uses scalable vectors
 - Not int2 or int4 but more like intN
 - Vector instructions will act on chunks within 'N'
 - Up to the maximum hardware-supported vector length (bits)
 - Config instructions dictate how vector registers will be used
 - e.g. element size, total required
 - returns active vector length used in following instructions

RISC-V Vector Example

- Example float operations for 15 items for different VLEN
- Configure for 15 total elements, 32 bits element size



Example System







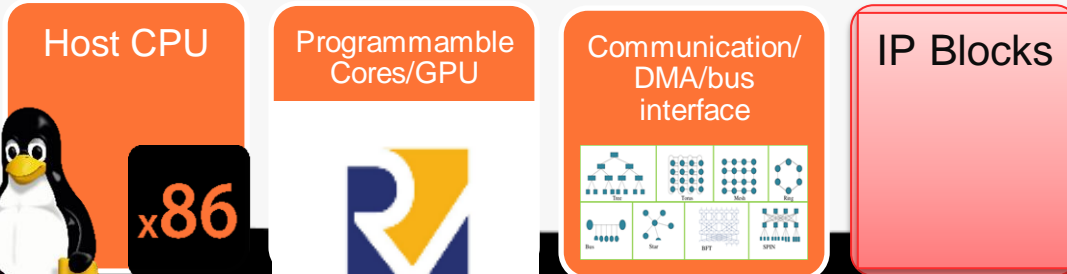
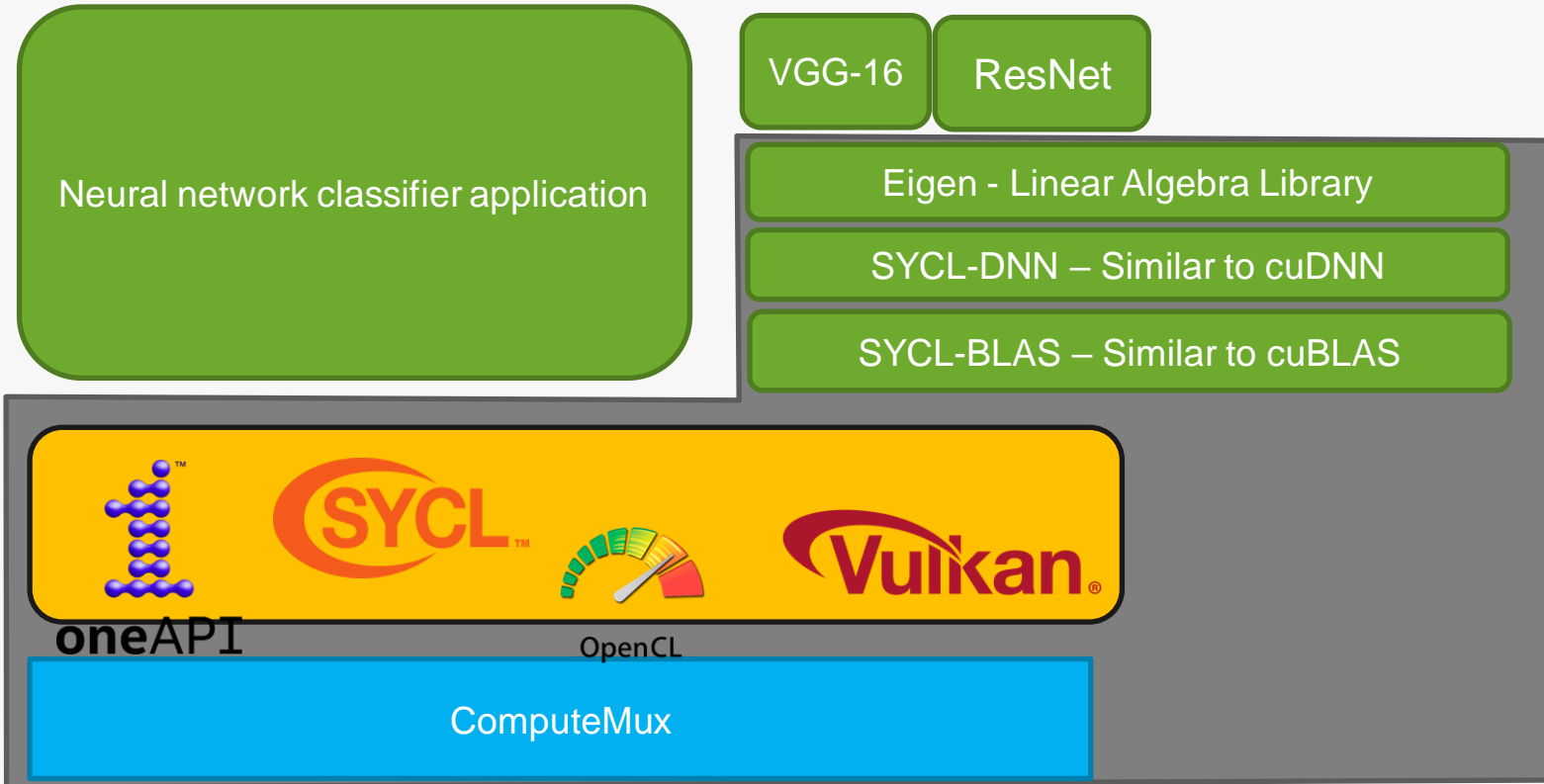
Implementing SYCL Support for RISC-V

- Codeplay is working with the community
- Code contributed to LLVM
- Our software stack enables RISC-V simulator

NSITEXE, Kyoto Microcomputer and Codeplay Software are bringing open standards programming to RISC-V Vector processor for HPC and AI systems

SYCL on RISC-V Architecture

-  C++ software
-  Compiler
-  Driver interface API
-  RISC-V hardware



This software stack provides all the supporting open source libraries and frameworks needed to build this



Next Steps

- Support additional hardware
 - Bespoke IP blocks
 - AI and image-specific hardware blocks
 - DMA
- Mapping OpenCL extensions to hardware
- Support new or custom RISC-V extensions
- Continue work on scalable vectors
- Tune for performance



Safety Critical C++

Safety Critical Heterogeneous Computing

- There are multiple industries interested in safety critical heterogeneous computing
- The automotive industry is moving towards these architectures
- They are essential for AI software execution

JOIN SYCL Safety-Critical Exploratory Forum Now

Exploring real-world industry requirements for *open* and *royalty-free* high-level compute APIs suitable for safety-critical markets

Proven Khronos Exploratory Process to ensure industry requirements are fully understood before starting standardization initiatives

More information and signup instructions
<https://www.khronos.org/syclsc>

Khronos SYCL Safety-Critical Exploratory Forum

KHRONOS GROUP **SYCL™**

Online discussion forum and weekly Zoom calls

No detailed design activity to protect participants IP

Explore if consensus can be built around an agreed **Scope of Work** document

Discuss what standardization activities can best execute actions in the Scope of Work

Any company is welcome to join

No cost or IP Licensing obligations

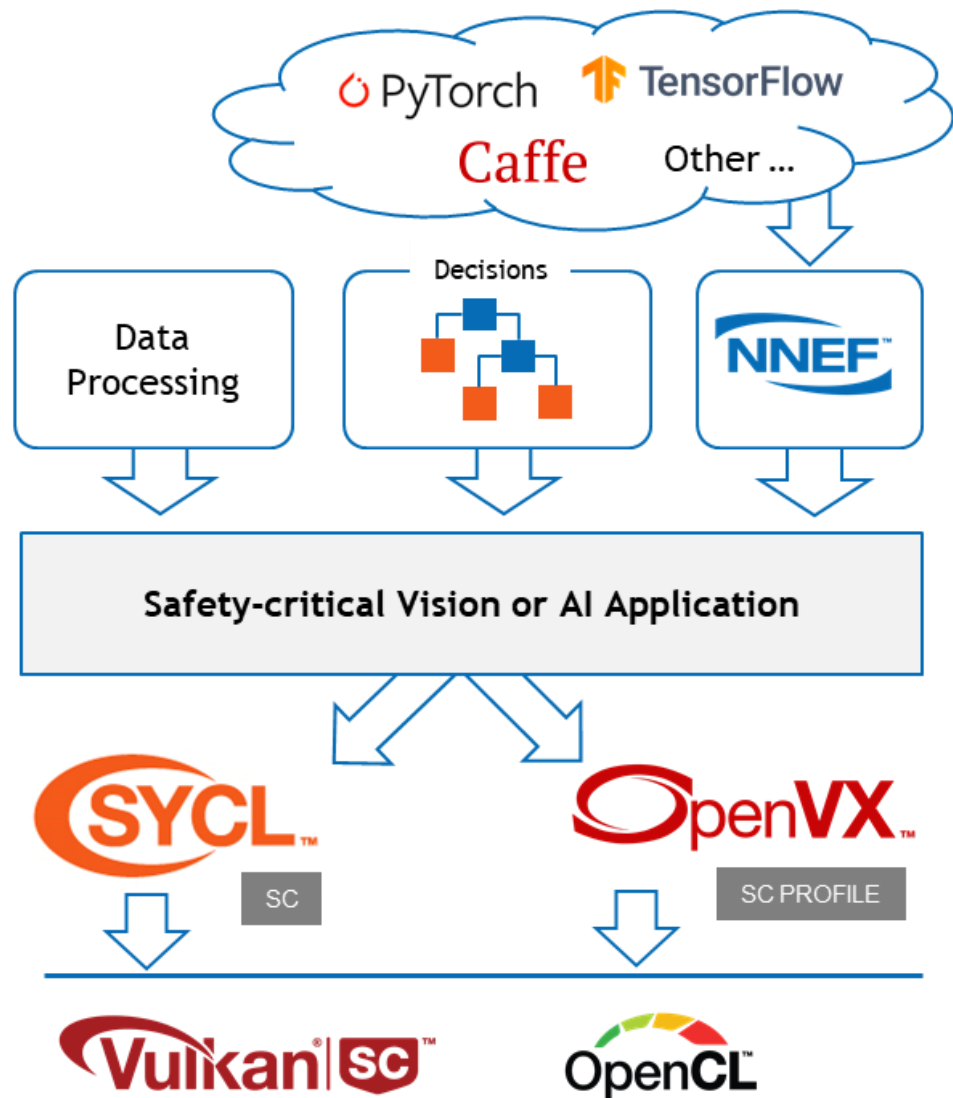
Project NDA to cover Exploratory Forum Discussions

Scope of Work Document

Agreed SOW document released from NDA and made public

Initiation of Khronos Working Group to execute the SOW

SYCL SC in the Khronos SC Ecosystem



Neural network models are trained in the cloud using a variety of platforms.

Once the model is trained it is exported and converted to NNEF before being passed to a safety-critical API for inferencing.

OpenVX provides high level APIs for Vision and AI with a safety-critical profile, enabling applications to quickly deploy trained NN models.

SYCL SC provides a general parallel programming API for accelerated compute at the C++ level. A typical AI application pipeline will combine the discreet functionality exposed by OpenVX with proprietary algorithms written using SYCL SC involving data pre-processing and post-processing, as well as complex decision making.

Vulkan SC or **OpenCL** are lower, execution-level APIs that could be used to accelerate higher-level APIs like SYCL SC & OpenVX

Khronos AUTOSAR Liaison What next?



CROSS-STANDARD LEAD WORKING GROUPS (FO, CP, AP)



Design guidelines for using parallel processing technologies on Adaptive Platform
AUTOSAR AP R19-11

Document Title	Design guidelines for using parallel processing technologies on Adaptive Platform
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	884
Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R19-11

3D Graphics
Desktop, Mobile and Web

3D Assets
Authoring and Delivery

Portable XR
Augmented and Virtual Reality

Parallel Computation
Vision, Inferencing, Machine Learning

SYCL/OpenCL
Safety Critical APIs

KHRONOS
SAFETY CRITICAL
ADVISORY FORUM





My Summary

SYCL Modern C++ Heterogeneous Model

- SYCL is becoming part of a standard programming model for HPC machines across the world
- SYCL is a global standards group with multiple company contributions
- SYCL moves with ISO C++, updates every 1.5-3 years
- SYCL is at the heart of oneAPI
- SYCL can adapt to the latest HPC hardware changes
- SYCL is being used in multiple top500 supercomputers
- SYCL, RISC-V, Vector Extension and Accelerators with LLVM Machine Learning

Enabling Industry Engagement (2022/05/14)

- SYCL working group values industry feedback
 - <https://community.khronos.org/c/sycl>
 - <https://sycl.tech>
- SYCL Academy
 - <https://github.com/codeplaysoftware/syclacademy>
- SYCL FAQ
 - <https://www.khronos.org/blog/sycl-2020-what-do-you-need-to-know>
- SYCL regular Survey

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**

• Regular meetings to give feedback on roadmap and draft specifications

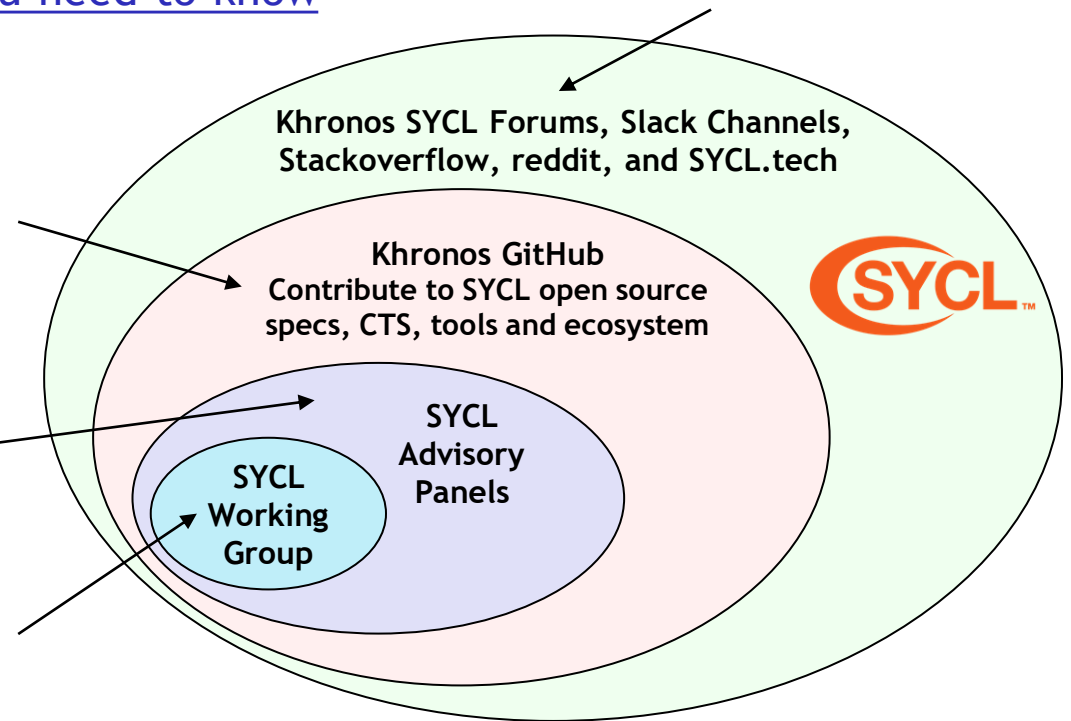
Public contributions to Specification, Conformance Tests and software
<https://github.com/KhronosGroup/SYCL-CTS>
<https://github.com/KhronosGroup/SYCL-Docs>
<https://github.com/KhronosGroup/SYCL-Shared>
<https://github.com/KhronosGroup/SYCL-Registry>
<https://github.com/KhronosGroup/SyclParallelSTL>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



We're
Hiring!

codeplay.com/careers/



Thanks



[@codeplaysoft](https://twitter.com/codeplaysoft)



info@codeplay.com



codeplay.com