KHRONOS GROUP

SYCL State of the Union Keynote SYCLcon 2022

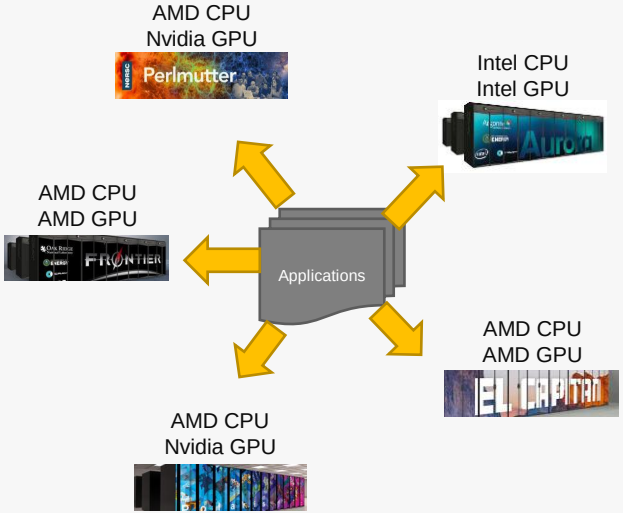
Specification Release

IWOCL SYCLcon

Michael Wong
SYCL V
Codeplay Distinguished
ISOCPP Foundation
ISO C++ Directio
michael@codeplay.com | wongmichael.co

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

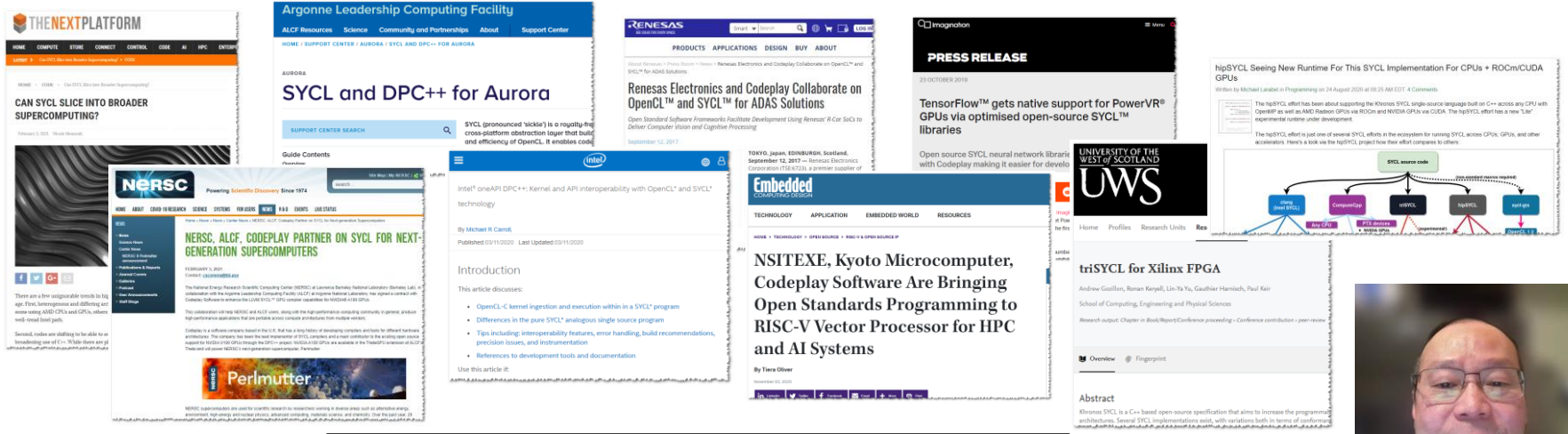
SYCL 2020 Launched February 2021

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

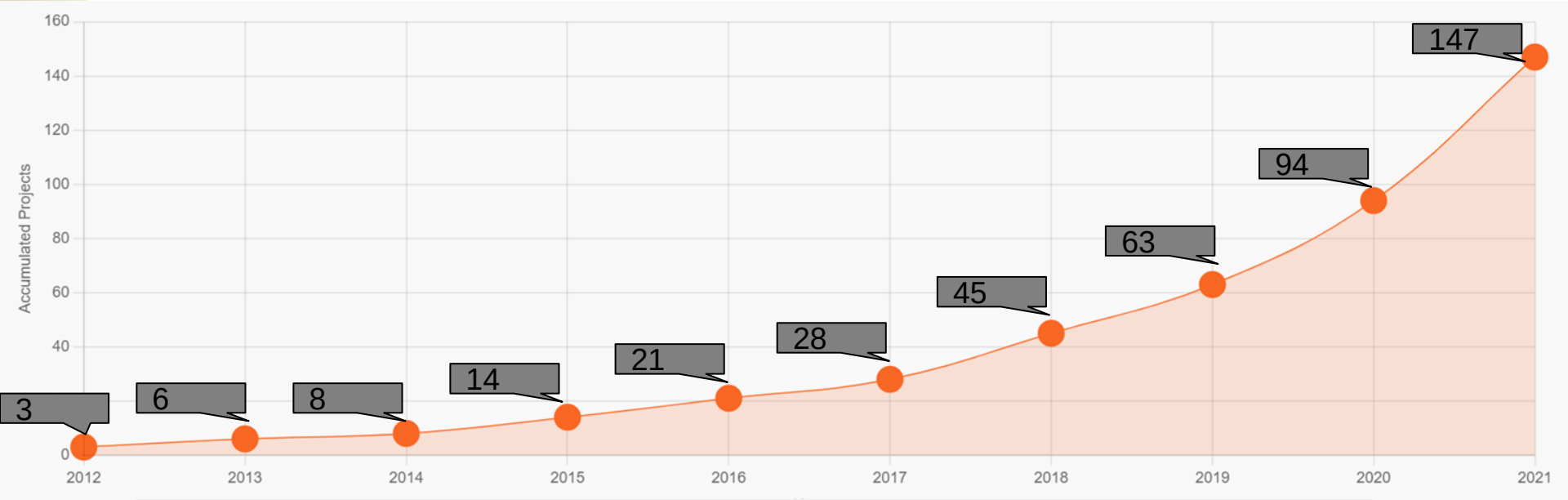
Unified Shared Memory | Parallel Reductions | Subgroup Operations | Class template Argument Deduction



Significant SYCL adoption in Embedded, Desktop and HPC Markets

SYCL Projects cumulative growth

Projects - SYCL.tech



Benchmarks

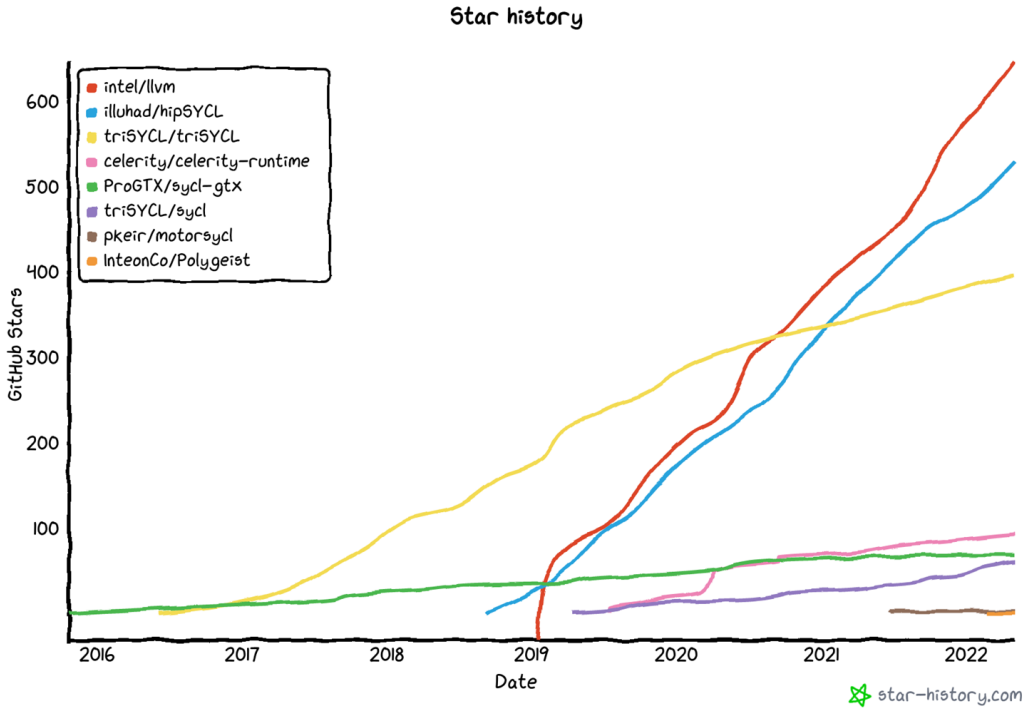
Frameworks

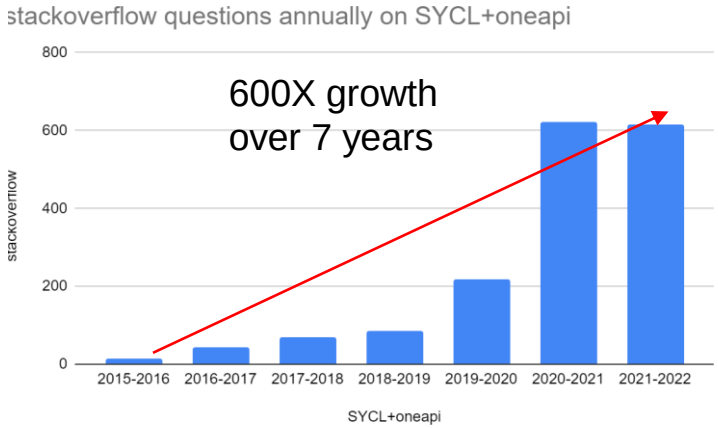
Libraries

Scientific

This

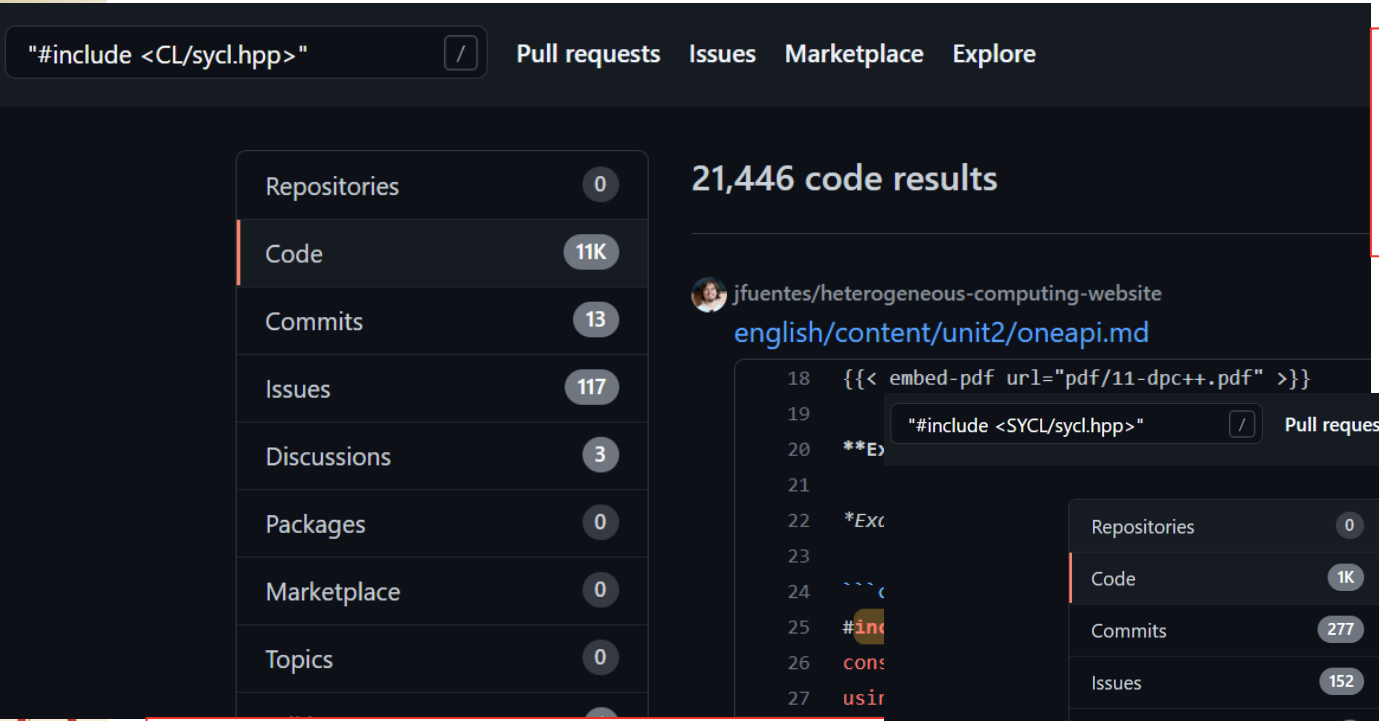
SYCL user and developer Phenomenal Growth





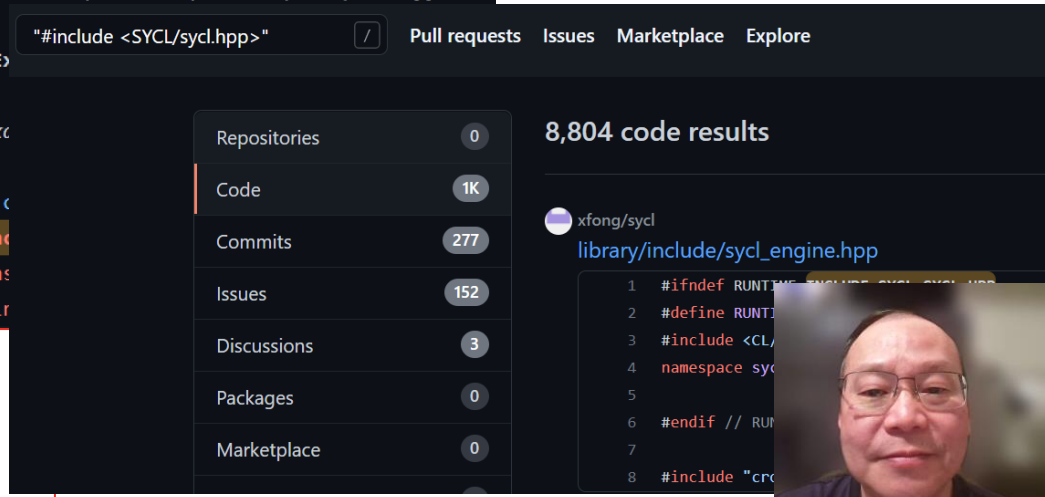
Some open-source SYCL implementations/prototypes on GitHub

Github SYCL source files is over 30000



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

21 K in April, 2022



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

8.8 K in Apr, 2022

Market needs SYCL: easy to build SYCL on any device

2016 	2022 
- 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	- 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 role - Renesas R-Car - Clang/LLVM DPC++ activ - 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 coming

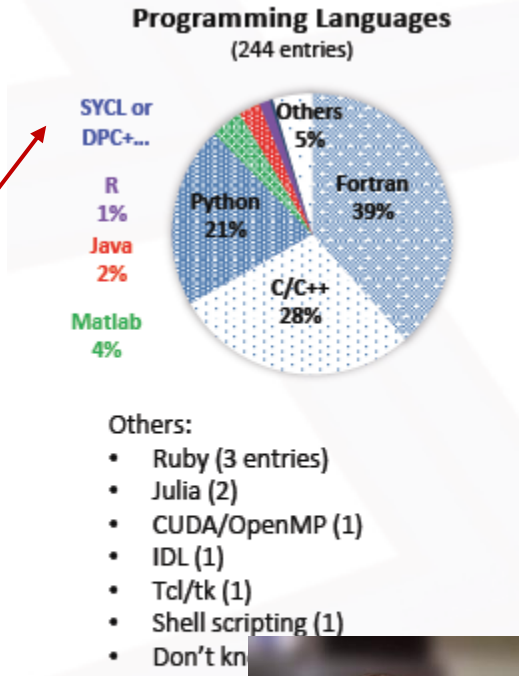
Q* What language functionality would you like to see more broadly supported?

- ☐ 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

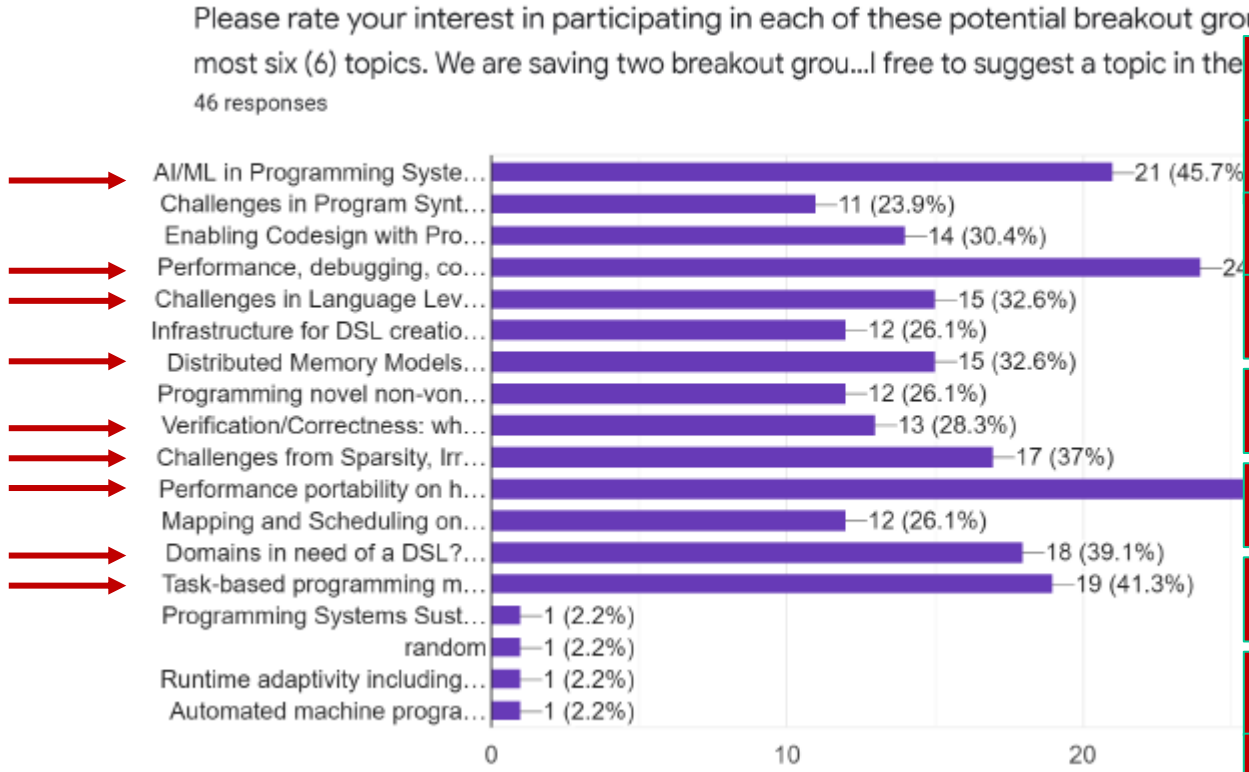


Market needs SYCL to succeed in democratizing

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



Compilers with AI/ML

Performance debugging

Language Parallelism

DIstributed Memory

Verification Correctness

Sparsity Irregularity

Performanc

Domains ne

Task Based

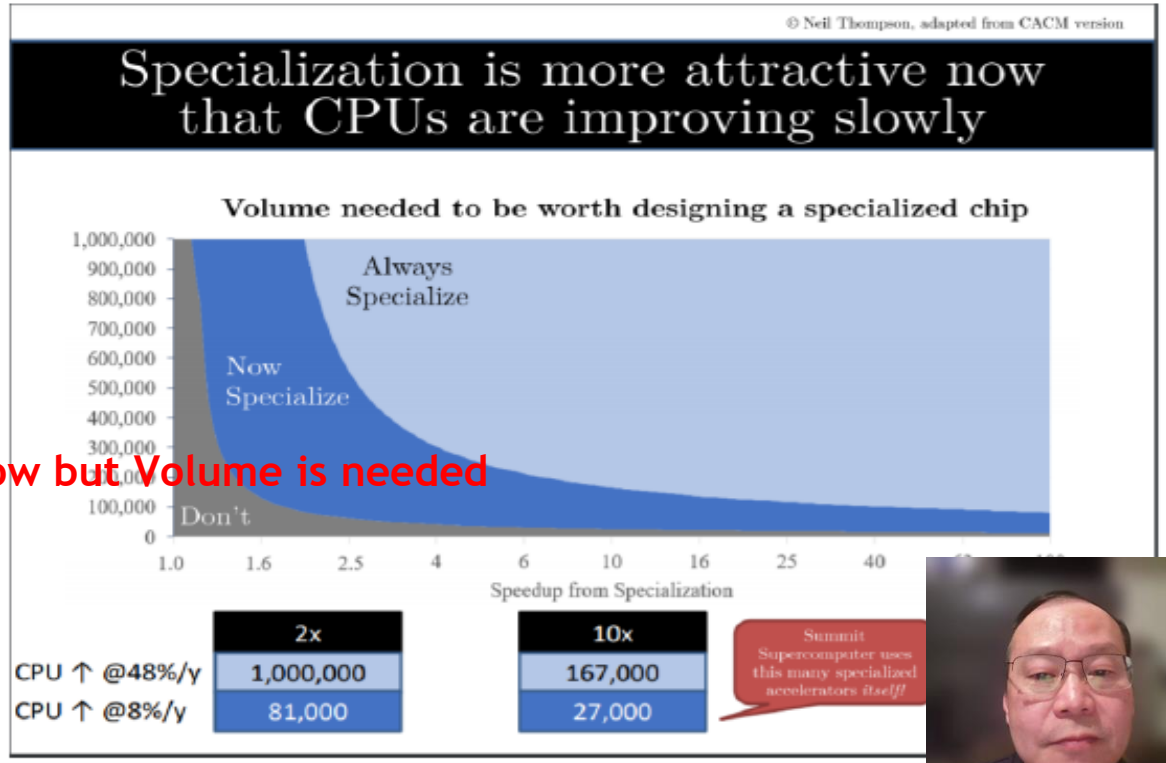
From ASCR Workshop Mar 2021

ASCR Workshop on Reimagining Codesign

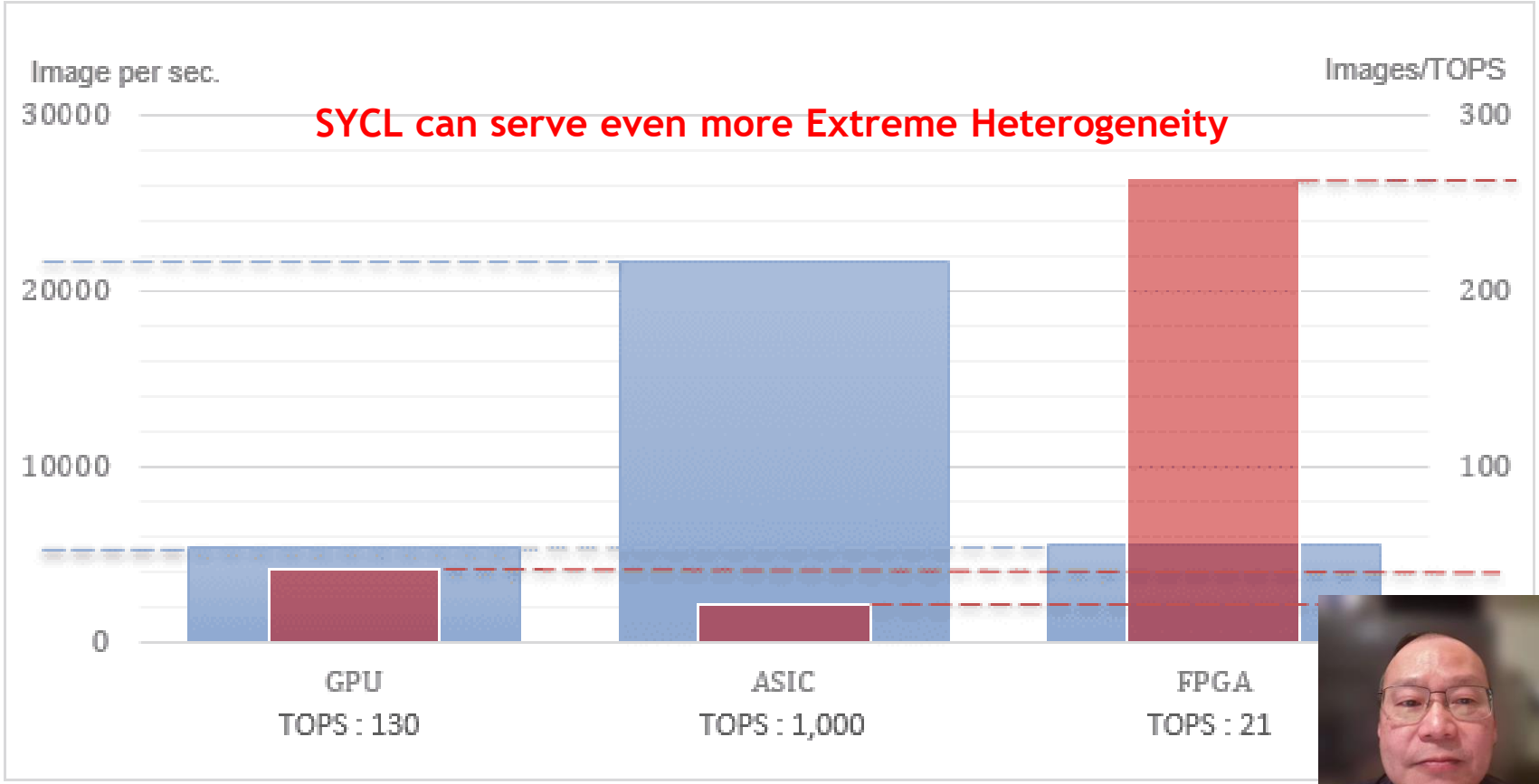
https://www.orau.gov/ASCR-CoDesign/

From Neil Thompson's keynote:

Specialization is more attractive now but Volume is needed



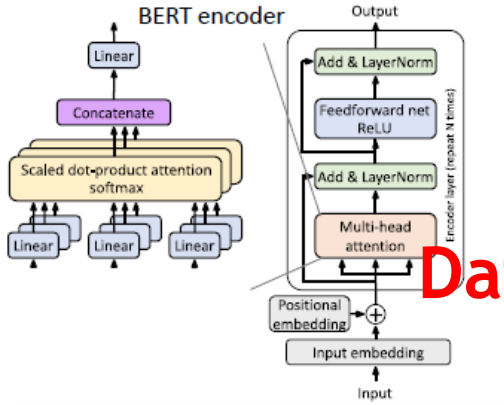
From Flops to TOPS in ML



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

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

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



highly optimized

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

0.2% 39%

Data Movement is still King

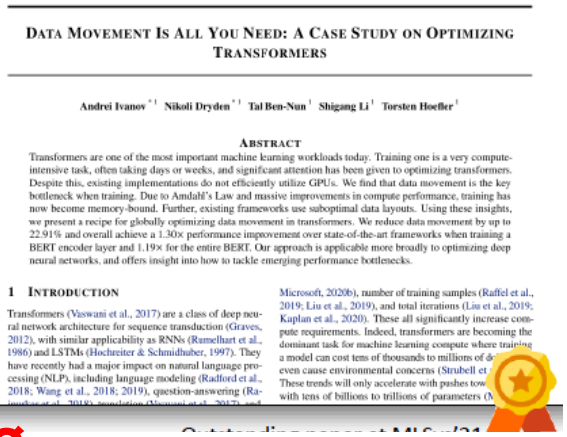
Our performance improvement for BERT-large

- 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: Khari 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.



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

Andrei Ivanov [*1] Nikoli Dryden [*1] Tal Ben-Nun [1] Shigang Li [1] Torsten Hoefler [1]

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 1.30× performance improvement over state-of-the-art frameworks when training a BERT encoder layer and 1.19× 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 (Ra-

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., These trends will only accelerate with pushes towards with tens of billions to trillions of parameters (N

[cs.LG] 8 Nov 2021

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

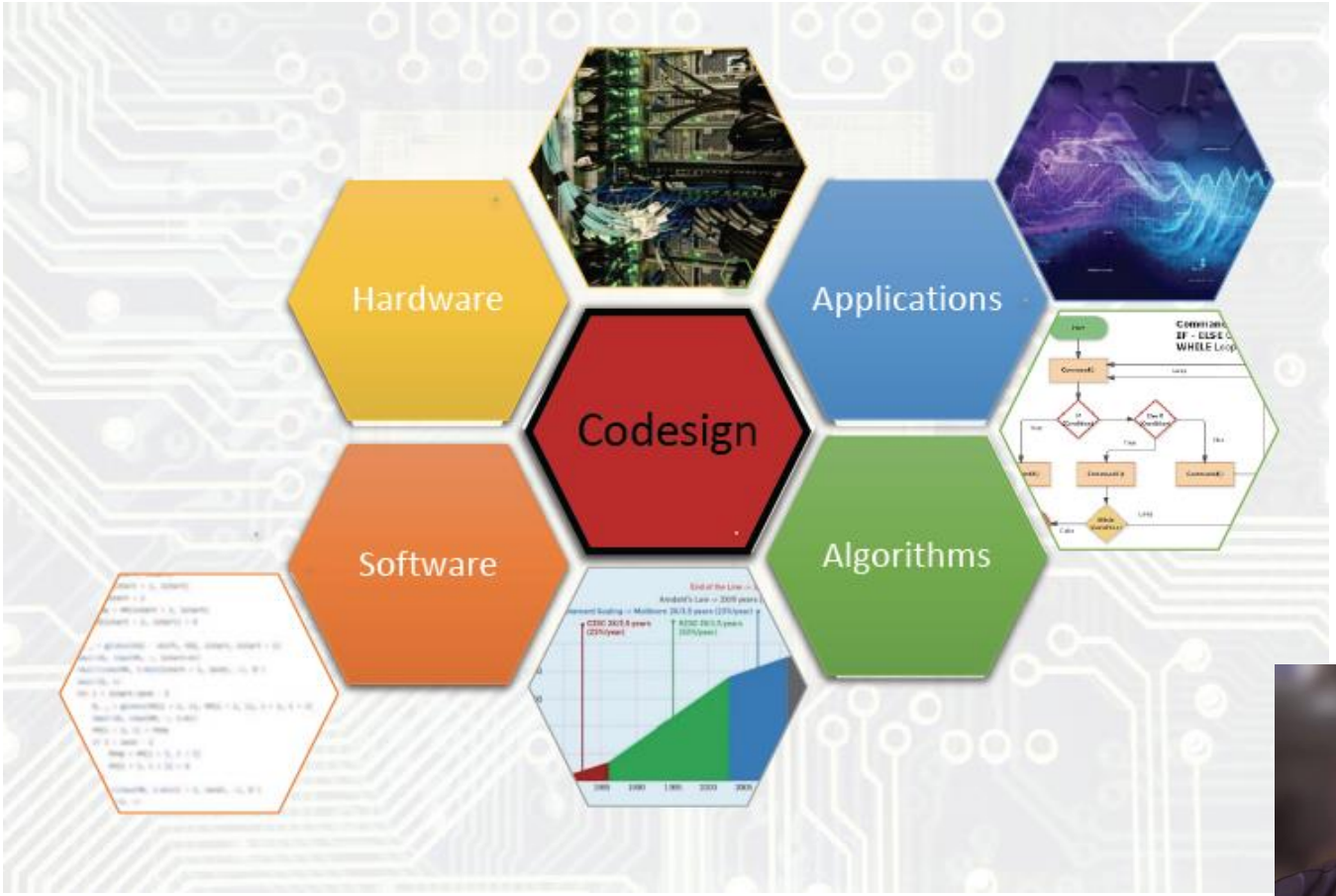
TORSTEN HOEFLER, ETH Zürich, Switzerland
DAN ALISTARH, IST Austria, Austria
TAL BEN-NUN, ETH Zürich, Switzerland
NIKOLI DRYDEN, ETH Zürich, Switzerland
ALEXANDRA PESTE, IST Austria, Austria

The growing energy and performance costs of deep learning have drive
of neural networks by selectively pruning components. Similarly to th
networks generalize just as well, if not better than, the original dense
memory footprint of regular networks to fit mobile devices, as well as sh
networks. In this paper, we survey prior work on sparsity in deep learni
of sparsification for both inference and training. We describe approac
neural networks, different training strategies to achieve model sparsity, ar
practice. Our work distills ideas from more than 300 research papers and
who wish to utilize sparsity today, as well as to researchers whose goal
include the necessary background on mathematical methods in sparsi
as early structure adaptation, the intricate relations between sparsity
techniques for achieving acceleration on real hardware. We also define a
that could serve as a baseline for comparison of different sparse networ
sparsity can improve future workloads and outline major open problems

The supreme goal of all theory is to make the irreducible basic
possible without having to surrender the adequate representation of

02.00554v1 [cs.LG] 31 Jan 2021

We need Codesign with extreme Heterogeneity



https://doi.org/10.2172/1822198

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, UCIe, 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 Framewor
  MLIR)
- AI integrated applications
- Programming Abstractions, Frameworks, and La

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

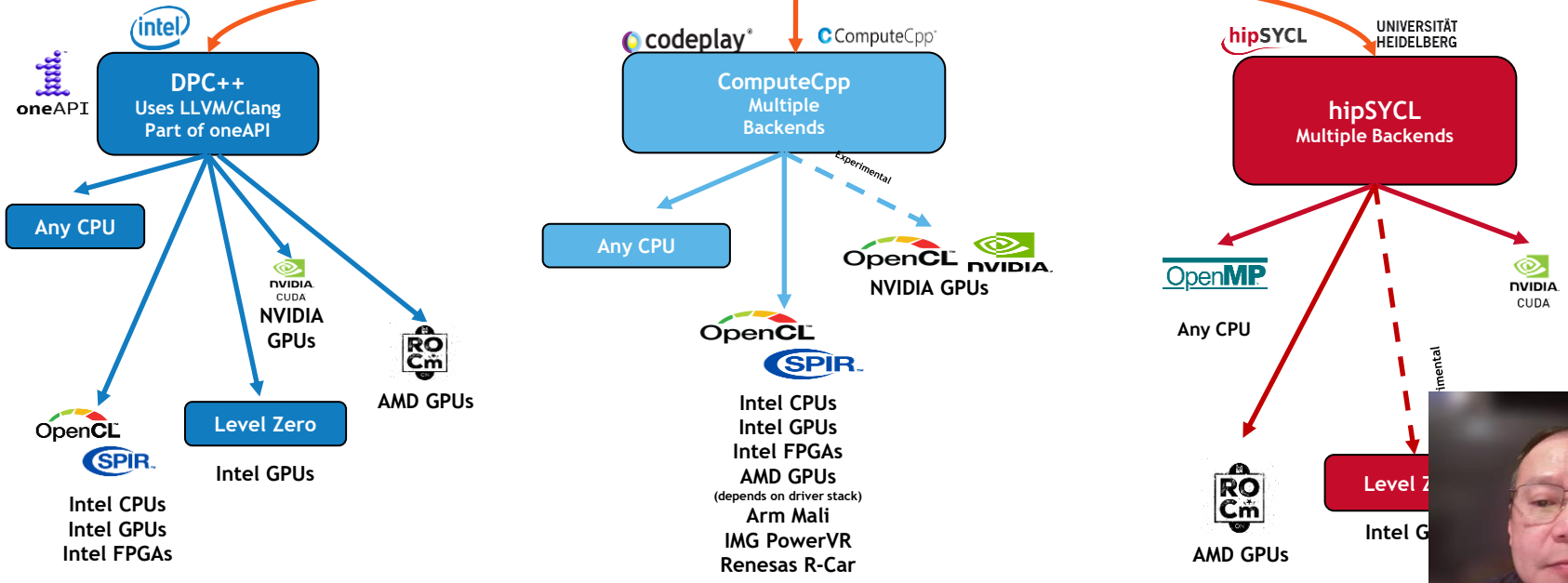
2020

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 Source Code

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

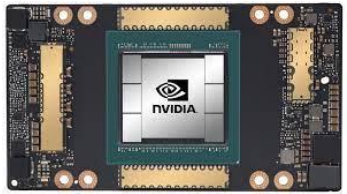
SYCL Source Code

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

- Sylkan → Vulkan
- Huawei SYCLops → DPC++ fork → MLIR
- Celerity SYCL → MPI
- triSYCL (Open source test bed) (AMD) → OpenMP → Any CPU; DPC++ fork → TBB → Any CPU; Experimental → OpenCL / SPIR → XILINX FPGAs POCL (open-source OpenCL supporting CPUs and NVIDIA GPUs and more); Experimental → XILINX Versal ACAP LLVM IR FPGA LLVM IR HLS
- MotorSYCL → Nvidia HPC SDK
- Inteon Poligeist SYCL → MLIR
- neoSYCL SX-AURORA TSUBASA (TOHOKU) → DPC++ fork → VEO → Intel CPU NEC VEs
- SYCL/DPC++ Huawei Ascend AI (HUAWEI) → DPC++ fork → AI

Multiple Backends
SYCL on even more lo
For more informatio

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



ALTERA

XILINX

RISC-V®

NVIDIA AMD habana Cerebras

Google groq GRAPHCORE

HUAWEI Tenstorrent SambaNova

- **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
- **CPU**
  - All major SYCL implementations

SYCL Ecosystem, Research and Benchmarks 2022/05/01

Implementations

neoSYCL
SX-AURORA TSUBASA

oneAPI
oneAPI

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

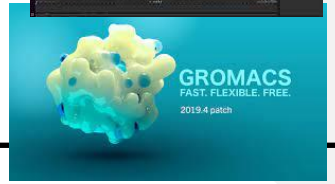
HUAWEI ComputeCpp™
hipSYCL

Research

Celerity
High-level C++ for Accelerator Clusters

ECP
EXASCALE COMPUTING PROJECT

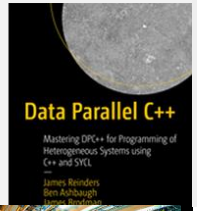
PYTORCH



ATLAS
EXPERIMENT

Flashlight

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

STREAM HPC

Argonne NATIONAL LABORATORY

arm

QUALCOMM

AMD

SYCL

universität innsbruck

intel

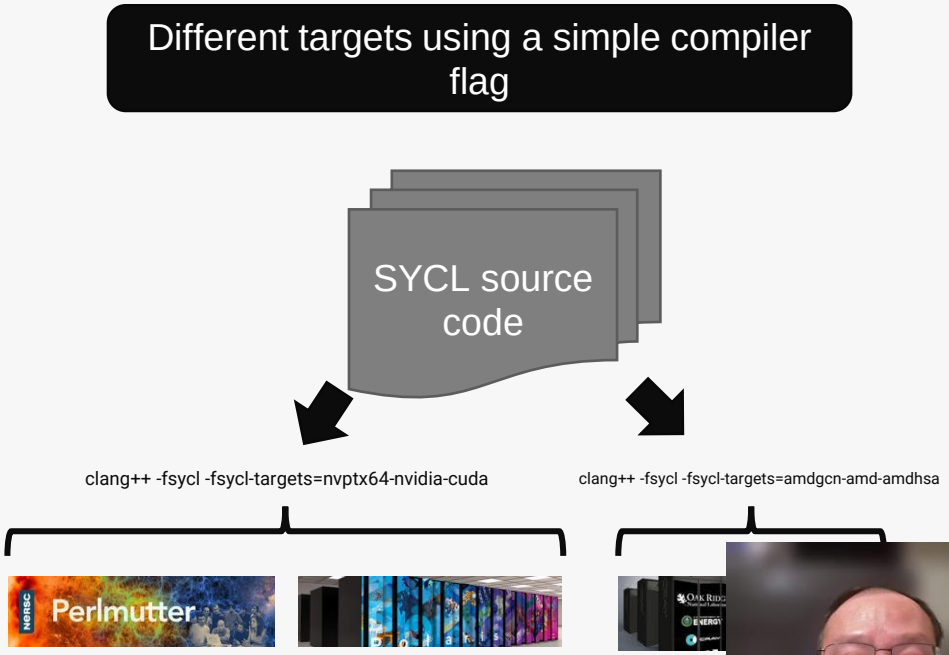
codeplay

University of BRISTOL

Working Group Members

Nvidia and AMD Support in DPC++

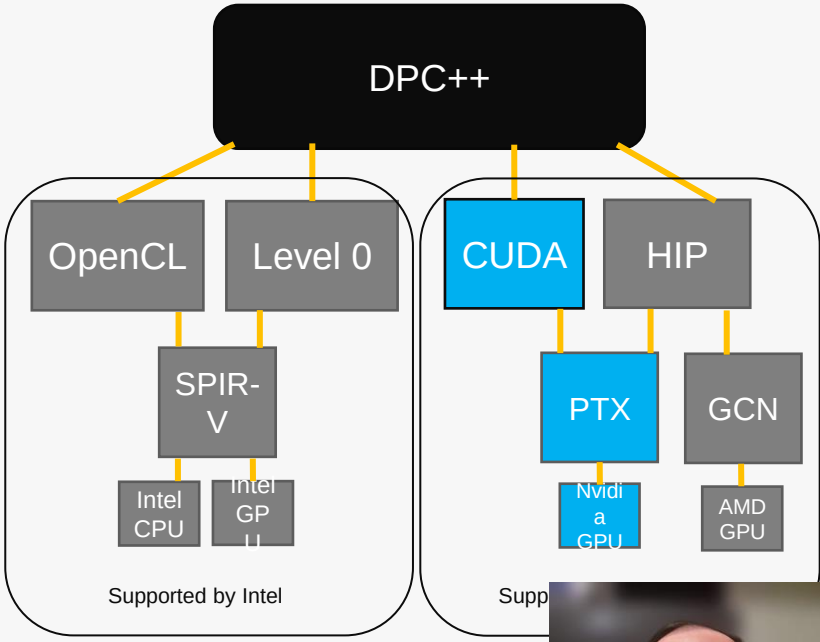
- Extending DPC++ to target Nvidia and AMD GPUs
- Supporting Perlmutter, Polaris and Frontier supercomputers
- Open source and available to everyone
- Codeplay commercially supports these implementations

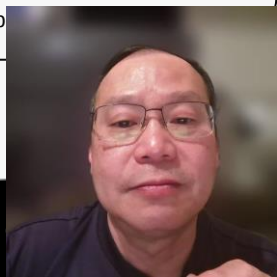


https://www.codeplay.com/oneapiforcuda
Resources for AMD coming soon

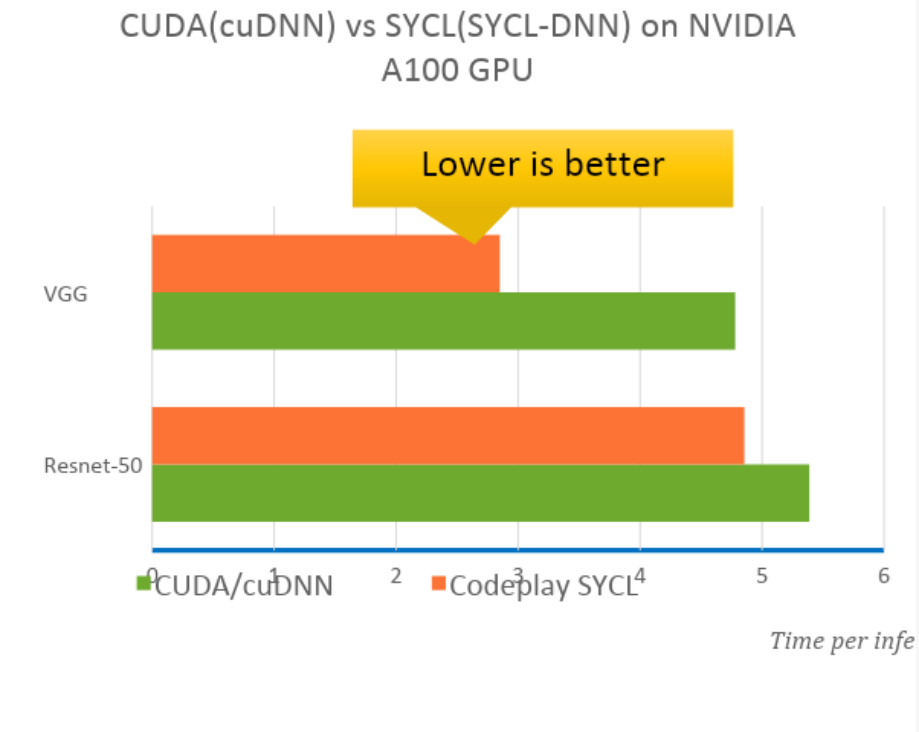
CUDA as a backend

- 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



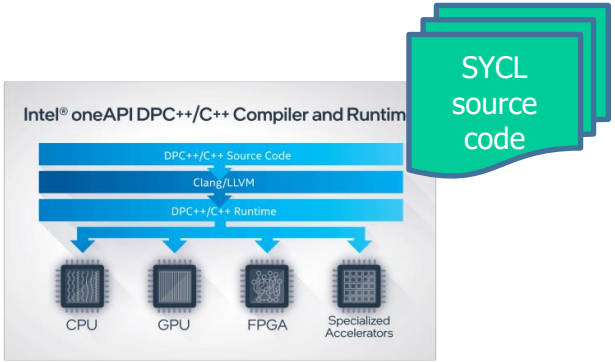


Performance comparison



- 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 increas performance

oneAPI and SYCL



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

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

kokkos
RAJA
Celerity
alpaka
BabelStream
LAMMPS
GROMACS
fast, flexible & free

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**

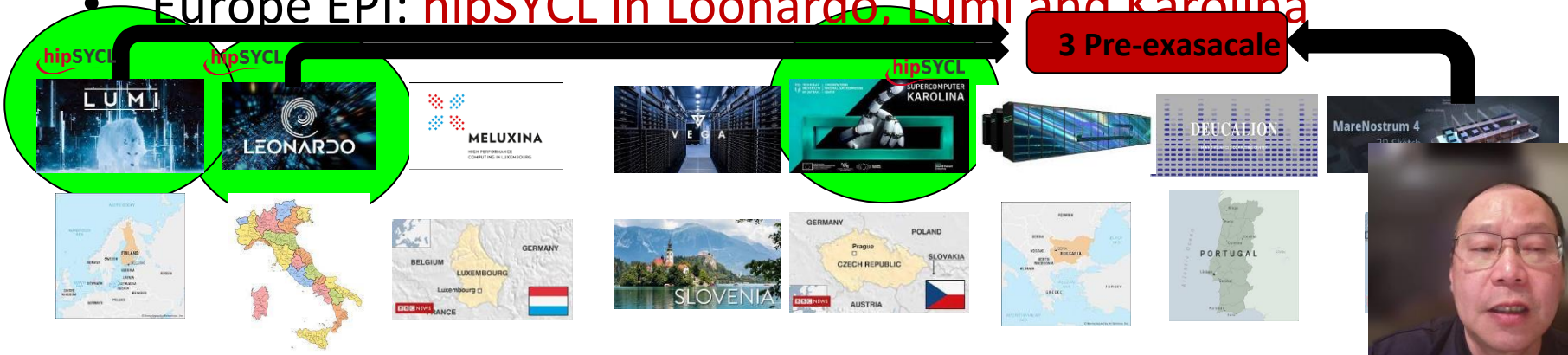
https://www.ligateproject.eu/

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 Loonardo, Lumi and Karolina



Easier, Short Path to Heterogeneous Programming

Coming Soon: An Open Source CUDA* to SYCL* Code Migration Tool – Project SYCLomatic, DPCT

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

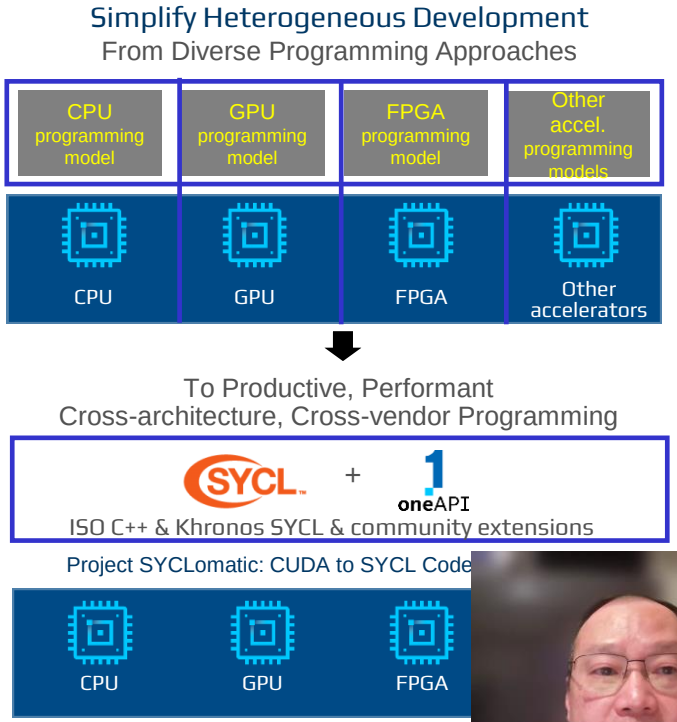
- Allows developers to expand the value of their investments across architectures
- Provides choice in hardware & freedom from proprietary, single-vendor lock-in

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 by end of May

  github.com/oneapi-src/SYCLomatic github.com/oneapi-src/SYCLomatic-test

  Use the tool, please provide feedback!



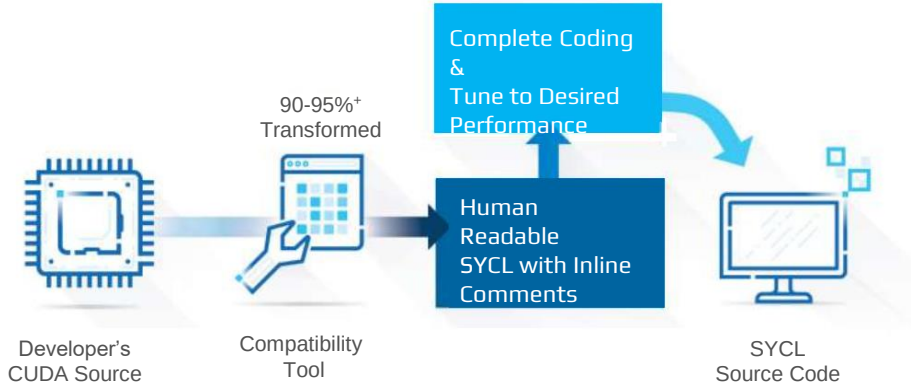
SYCLomatic Tool Usage Workflow
(newly open sourced by the end of May)

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



+ Intel estimates as of September 2021. Based on measurements on a set of 70 HPC benchmarks and samples, with examples like Rodinia, SHOC, PENNANT. Results may vary.

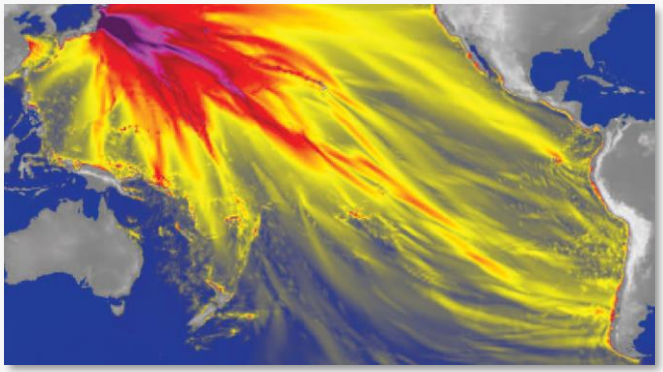
*Other names and brands may be claimed as the property of others.

Case Study: Zuse Institute Berlin

ZUSE INSTITUTE BERLIN

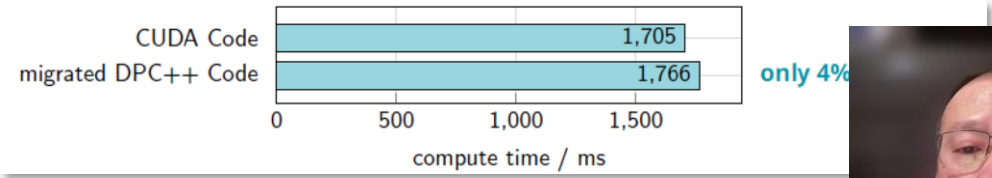
From CUDA to DPC++ and back to Nvidia GPUs... and FPGAs

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]

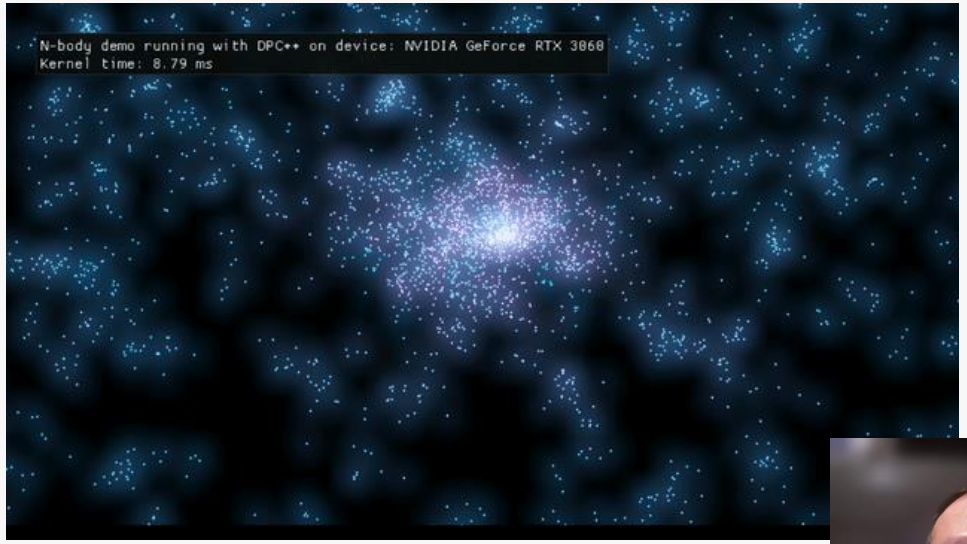
Even on NVIDIA GPU, SYCL code is only 4% slower (with Codeplay developed open-source compiler)



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/topics/n-body-simulator

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();
}
```

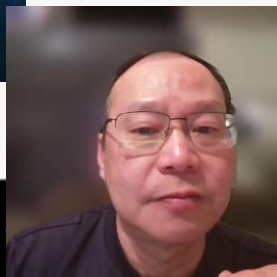
Code changes are minimal

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

www.codeplay.com/oneapiforcuda/



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

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

More information and signup instructions

https://www.khronos.org/syclsc

Any company is welcome to join

No cost or IP Licensing obligations

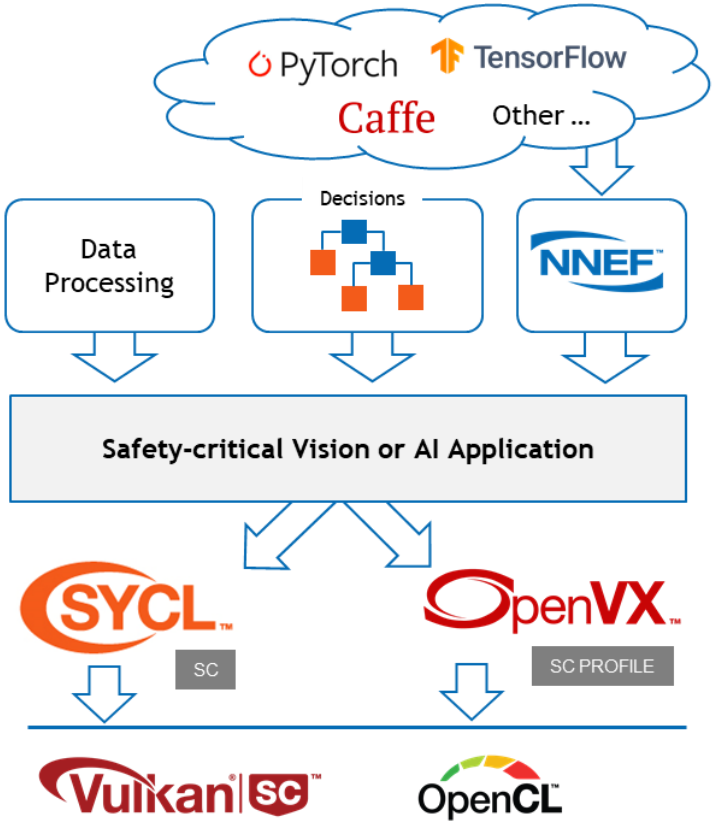
Project NDA to cover Exploratory Forum Discussions

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.

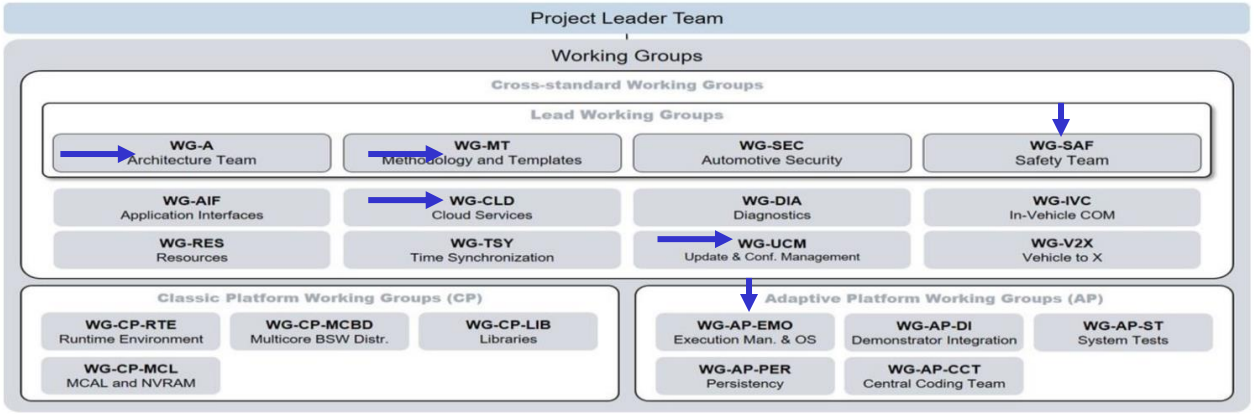
Once the model is trained it is exported and converted to NNEF before being passed to a safety-critical API for inferencing.

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 cou
used to accelerate higher-level APIs like SYCL SC & OpenVX

Khronos AUTOSAR Liaison What next?



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

AUTOSAR

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

Vulkan. ANARI

OpenGL|ES OpenGL

WebGL

3D Assets
Authoring and Delivery

glTF

COLLADA

3DCommerce

Portable XR
Augmented and Virtual Reality

OpenXR

Parallel Computation
Vision, Inferencing, Machine Learning

OpenGL|SC Vulkan|SC

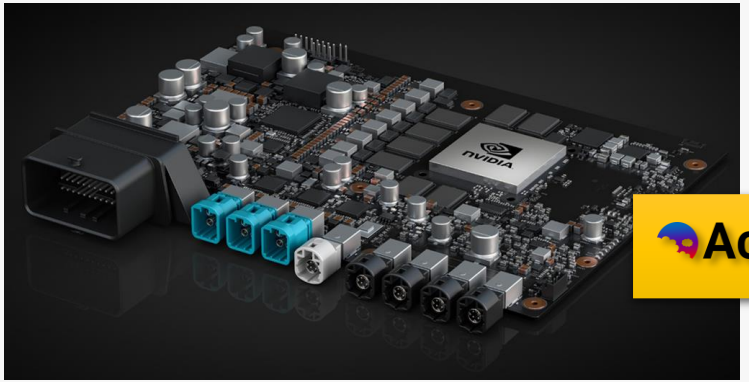
SYCL/OpenCL Safety Critical APIs

SAFE ADV

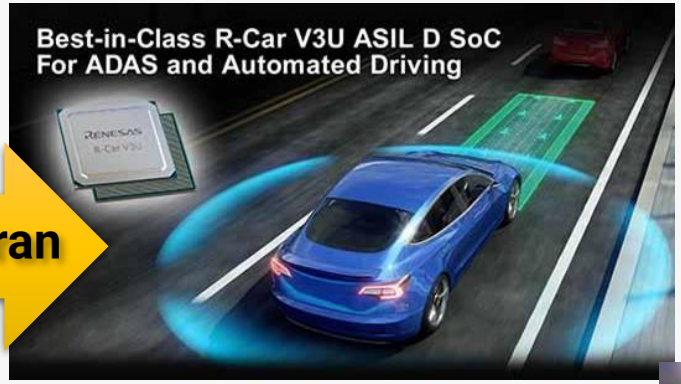
Case Study: Automotive AI

RENESAS R-Car Consortium

- Codeplay provides a full open-standards software stack (SYCL/OpenCL/SPIR-V) for the Renesas R-Car AI system-on-chip
- This enables easy transition from NVIDIA GPU platforms to R-Car platform



Acoran



"For Renesas, SYCL is a key enabler for automotive ADAS/AD software developers that allows them to easily use the highly-efficient, heterogeneous accelerators of the R-Car SoC Series through the open Khronos standard,"

Final words

- **Programming Models Must Persist**
- **SYCL 2020 Launched February 2021**
- **SYCL user and developer Phenomenal Growth**
- **Market needs SYCL: easy to build SYCL on any device**
- **SYCL is mainstream**
- **Market needs SYCL to Evolve (call to action)**
- **SYCL can serve even more Extreme Heterogeneity**
- **Data Movement is still King**
- **We need Codesign with extreme Heterogeneity**
- **Future SYCL: Emerging transformative technologies**
- **SYCL can be a part of a standard programming model for all HPC ,Embedded AI/ML, and Automotive**
- **SYCL is an open standard with multiple company contributions, lots o European/Asia projects**

Enabling Industry Engagement

- 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 Survey coming

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

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

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