SYCL State of the Union Keynote
SYCLcon 2023
Build a thriving community

Michael Wong
SYCL WG Chair
Codeplay Distinguished Engineer
ISO C++ Directions Group
michael@codeplay.com | wongmichael.com/about
Agenda

Amazing Growth

Highlights of last 12 months

Ecosystem and future growth Directions
Programming Models Must Persist

Programming models should have high quality, portable implementations

Which programming languages have formal conformance test?

ECMAscript, HTTP, Kubernetes, Khronos languages
SYCL 2020 Conformance Test Suite released

Expressiveness and simplicity for heterogeneous programming in modern C++

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

https://github.com/KhronosGroup/SYCL-CTS

Significant SYCL adoption in Embedded, Desktop and HPC Markets
SYCL 2020 Adopters Program

Becoming an Adopter of a Khronos standard gives you access to the Khronos Conformance Testing Process:

- Download the source of the Khronos conformance tests to port and run on your implementation.
- Access the Adopters Mailing list; a priority channel for two-way interaction with Khronos Members who can offer assistance on running tests.
- Upload generated test results for Working Group review and approval to become officially conformant.
- Submit an unlimited number of products for that version of the standard (and earlier versions as indicated in the pricing table below).

<table>
<thead>
<tr>
<th>Develop Products</th>
<th>Adopter</th>
<th>Implementer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access to the public Khronos Specifications, documentation and support files</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Develop license-free, royalty-free products using Khronos Technologies</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conformance Testing</th>
<th>Adopter</th>
<th>Implementer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access to Adopter mailing list</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Formal Review Process</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Submit products to the conformance process (Must sign Adopter agreement and pay Adopter fee)</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Conformant Product can use API Trademark (Must pass conformance tests)</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Marketing</th>
<th>Adopter</th>
<th>Implementer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opportunity to mention products and be quoted in Khronos press releases, articles, and newsletters</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Company logo and description on Khronos web site</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>
SYCL Projects cumulative growth

Projects - SYCL.tech
SYCL user and developer Phenomenal Growth

A few open-source SYCL implementations/prototypes on GitHub
500x growth over 8 years

Stackoverflow questions annually on SYCL+oneapi

500X growth over 8 years
SYCL is mainstream

Open Standards and Open Source implementations, community driven

Open cross-company collaboration

Co-design for all forms of extreme heterogeneity

Open Source without a community is useless

Companies can play in the Khronos ecosystem w/o revealing IP

Focus on ease of portability support, capable of many backends, and demonstrated to support many platforms
Agenda

Amazing Growth

Highlights of last 12 months

Ecosystem and future growth Directions
SYCL 2020 V7 highlights (Editor’s corner)

Ronan Keryell, AMD
Usual 6-month maintenance release, many clarifications, no new feature.

- clarify buffer creation with nullptr;
- align more "concurrent" wording with ISO C++;
- precise that work-items provide weakly parallel forward progress guarantee;
- import forward progress definition from ISO C++ and clarify various aspects on atomicity and synchronization;
- C++17 replaced by just the C++ core language;
- fix description of max_work_item_sizes and clarify relationship to kernel dimensionality;
- clarify "group" meaning in algorithm descriptions;
- improve readability of group barrier description;
- mention kernel_handler in kernel function definition;
- relax requirement on backend traits being available;
- clarify the "reducer" member types and constants;
- clarify native_specialization_constant when empty;
- allow "empty" shared_ptr for buffer construction;
- add static constexpr `dimensions` member to all range/id-like types;
- clarify blocking behavior of `queue::submit`;
- clarifications to device copyable;
- clarify USM allocation of zero size coherent with std::malloc;
- clarify sycl::atomic_ref;
- clarify queue profiling behavior when unsupported;
- clarify the wording for the use of property::queue::in_order;
- reword guarantee about host-to-device fence synchronization;
- add single source single compiler pass (SSCP) to the glossary;
- add half to sycl::plus and sycl::multiplies and fix trait use;
- clarify any_device_has / all_devices_have;
- clarify that objects in global, local, or private address space can also be accessed via the generic address space;
- disallow ++ and -- for sycl::vec<bool>;
- no assignment for read-only accessors;
- clarifications to sub-group;
- clarify is_group and bool_constant alias relations;
- clarify out-of-bounds behavior for group_broadcast;

Plus many various changes…
SYCL Work in Progress
Open Standard for Single Source C++ Parallel Heterogeneous Programming

Processor-In-Memory extensions ([SYCL-Extension-Document/proposed at master · SAITpublic/SYCL-Extension-Document (github.com)])

Integrate mdspan ([OpenSYCL/extensions.md at feature/pointer-future · OpenSYCL/OpenSYCL (github.com)])

Generalized dimensions ([triSYCL/generalized_dimension.cpp at master · triSYCL/triSYCL (github.com)])

oneAPI numba-dppy ([Heterogeneous Programming Using Data Parallel Extension for Numba*... (intel.com)])

User-driven kernel fusion ([SYCL][Doc] Add kernel fusion extension proposal by victor-eds · Pull Request #7098 · intel/llvm (github.com)])

Clarify address spacing rules ([Improved address space inference for SYCL programs - YouTube])

SYCL design philosophy ([SYCL Design Philosophy v0.1 by ProGTX · Pull Request #136 · codeplaysoftware/standards-proposals (github.com)])


Host task interop ([SYCL][DOC] Improved host task synchronization extension by npmiller · Pull Request #7076 · intel/llvm (github.com)], [https://github.com/OpenSYCL/OpenSYCL/blob/develop/doc/enqueue-custom-operation.md])

Memory model, forward Progress, and Context
Ecosystem: A few SYCL Projects

Khronos Group
Request for Proposals

SYCL 2020 CTS
June 2022

https://github.com/KhronosGroup/SYCL-CTS
SYCL on Compiler Explorer (CPU execution)

https://godbolt.org/z/zexnnr4ne

“Compiler explorer is more fun than work”, Chris Gearing, Mobileye
SYCL IR on Compiler Explorer  
https://godbolt.org/z/jdhKr7e5r
RISC-V

Technical Organization

https://riscv.org/

Alan Chia - Lego Color Bricks  CC BY-SA 2.0
SYCLOPS: SYCL RISC-V Datacenter Horizon Projects

https://www.syclops.org/

Advancing AI/data mining for extremely large and diverse data for Europe and beyond, by democratizing its acceleration through open standards and a healthy, competitive, and innovating ecosystem.
AERO: SYCL RISC-V Cloud Computing Horizon Project

https://aero-project.eu/

The Future of Cloud

AERO has the single mission of enabling the future heterogeneous EU cloud infrastructure.
Parallel Industry Initiatives

- **C++11**: 2011
  - OpenCL 1.2
  - OpenCL C Kernel Language
  - OpenCL 1.2

- **C++14**: 2014
  - OpenCL 2.1
  - SPIR-V in Core

- **C++17**: 2017
  - SYCL 1.2.1
  - C++11 Single source programming

- **C++20**: 2020
  - SYCL 2020
  - C++17 Single source programming
  - Many backend options

- **C++23**: 2023
  - SYCL 202X
  - C++20 Single source programming
  - Many backend options

- **SYCL 1.2**: 2012
  - C++11 Single source programming

- **SYCL 1.2.1**: 2015
  - C++11 Single source programming

- **SYCL 2020**: 2020
  - C++17 Single source programming
  - Many backend options

- **SYCL 202X**: 2023
  - C++20 Single source programming
  - Many backend options

- **OpenCL 3.0**: 2020

- **OpenCL 2.2**: 2017

- **OpenCL 3.0**: 2020

- **C++11**: 2011

- **C++14**: 2014

- **C++17**: 2017

- **C++20**: 2020

- **C++23**: 2023

- **C++11**: 2011

- **C++14**: 2014

- **C++17**: 2017

- **C++20**: 2020

- **C++23**: 2023

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

© The Khronos® Group Inc. 2023 - Page 19
SYCL Implementations in Development (2023/04/18)

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

DPC++
Uses LLVM/Clang
Part of oneAPI

ComputeCpp
Multiple Backends

hipSYCL
Multiple Backends

Any CPU
Intel CPUs
Intel GPUs
Intel FPGAs

AMD GPUs

NVIDIA GPUs

OpenCL
Intel CPUs
Intel GPUs
Intel FPGAs

AMD GPUs

NVIDIA GPUs

SPIR

Any CPU

OpenMP

Level Zero

Integrates with existing hardware

New, not experimental anymore, and works on Ponte Vecchio

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

© The Khronos® Group Inc. 2023 - Page 20
SYCL Experimental Development (2023/04/18)

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
Building Performance-Portable Software

Starting from scratch
SYCL is the best place to start: open, future-proof, no lock-in, easy to learn

Starting from C++
Easy to add SYCL to existing C++ software

Starting from CUDA
Easy to port from CUDA to SYCL: keep performance on GPUs

Starting from another language
SPIR-V standard enables not just SYCL, but also new languages
Agenda

Amazing Growth

Highlights of last 12 months

Ecosystem and future growth Directions
SYCL™ Performance for Nvidia® and AMD GPUs Matches Native System Language

06 April 2023

Benchmarks executing workloads using UPC++, oneAPI’s implementation of SYCL, achieves close to native performance on Nvidia and AMD GPUs, when comparing to the same benchmarks run with CUDA® and HP®, respectively.

Bringing Nvidia® and AMD support to oneAPI

16 December 2022

Developers can write SYCL™ code and use oneAPI to target Nvidia® and AMD® GPUs with free binary plugins.

GROMACS 2023 Released With Better SYCL For Intel / AMD / Nvidia

Feb 23, 2023

GROMACS as the widely-used molecular dynamics software issued its stable v2023 release this week with improved GPU support via SYCL. Most significant to the GROMACS-2023 feature release is improving its SYCL implementation that provides production-rated support not only for Intel Arc Graphics but also AMD Radeon graphics with ROCm® and SYCL. There is also non-production-rated SYCL support for ARM Mali® or Mali-TPU/GPU and NVIDIA V100-SXM2.

STFC to Accelerate Exascale Software in Computational Fluid Dynamics and Code Coupling using SYCL

Jan 3, 2023

STFC funds to accelerate exascale software in Computational Fluid Dynamics and Code Coupling using SYCL.

Intel Arc GPUs and OneAPI — Do They SYCL?

Running oneAPI C++ with SYCL code on Intel Arc and Iris Xe GPUs

Accelerating Made Simpler With Celerity

Jul 5, 2022

OneAPI ecosystem expands to include a new accelerator from Celerity who are part of the Celerity project.
Special Interest Groups (SIGs) influence the specifications and implementations.

- Language: oneDPL, ISO C++
- AI: oneDNN, oneDAL
- Math: oneMKL
- Hardware: Level Zero, oneDNN, oneDAL, oneDPL, DPC++

Open Specification Working Groups: SYCL (Khronos)
Open Source Implementations: A general programming model, Deep Neural Network operations, Math operations, Hardware abstraction layer.
Contribute to the oneAPI Community Forum

• Join and lead SIGs and Working Groups
• Lead technical discussions
• Submit proposals for features and changes
• Vote on proposals

Drive the future of programming for heterogeneous architectures

https://oneapi.io/community
oneapi@codeplay.com
“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
More workloads need to be SYCL-ready

We have made great inroads with GROMACS, LAMMPS, NWChem workloads. But we need more!

The Hardware Accelerated Cosmology Code (HACC) framework uses N-body techniques to simulate the formation of structure in collisionless fluids under the influence of gravity in an expanding universe. It depends on external FFT library and is typically compute limited achieving 13.92 Petaflops, 69.2% of machine peak on Sequoia.

QMCPACK is a many-body ab initio quantum Monte Carlo code for computing the electronic structure of atoms, molecules, and solids. It is written primarily in C++, and its use of template metaprogramming is known to stress compilers. When run in production, the code is memory bandwidth sensitive, while still needing thread efficiency to realize good performance.

VPIC (Vector Particle-In-Cell) is a general purpose particle-in-cell simulation code for modeling kinetic plasmas. It employs a second-order, explicit, leapfrog algorithm to update charged particle positions and velocities in order to solve the relativistic kinetic equation for each species in the plasma, along with a full Maxwell description for the electric and magnetic fields evolved via a second-order finite-difference-time-domain (FDTD) solve.

Lagos solves the time-dependent Euler equation of compressible gas dynamics in a moving Lagrangian frame using unstructured high-order finite element spatial discretization and explicit high-order time-stepping. It is built on top of a general discretization library (MFEM) and supports two modes: full assembly, where performance is limited by the data.

Kripke is a structured deterministic (Sn) transport using RAJA. It contains wavefront algorithms that stress memory latency and/or bandwidth, and network latency.

Durham: SWIFT, Gadget (v3 and 4), arepo, bam, gizmo, ramses

Others: Grid, sphNG, BAM, Hydra, ATON, Phantom, Fargo3d, Pluto, cosmos++, borg-wl, prompi, GRChombo, swift, gadget, gizmo, rames, trove, milc, hirep,

- CASINO(*)
- CASTEP
- CESM2(*)
- Chemshell
- Code_Saturne
- CP2K
- CRYSTAL(*)
- FHI-aims
- GROMACS
- LAMMPS
- MITgcm
- Met Office Unified Model
- NAMD
- Nekter++
- NEMO
- NWChem
- ONETEP
- OpenFOAM
- ORCA(*)
- Quantum Espresso
- VASP
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
**SYCL Machine Learning**

**RISC-V/RVV Kernel compilation flow FC**

- **Device Compiler**
  - SYCL Kernel
  - Scalar LLVM IR
  - SPIR/SPIM-V

- **CPU Compiler**
  - Codeplay Vectorizer ('vec2')
  - Vector LLVM IR
  - LLVM back-end

**Our Contribution: CCE SYCL Plugin**

- **sycl-blas**
  - An implementation of BLAS using the SYCL open standard for acceleration on OpenCL devices.
  - C++

- **sycl-dnn**
  - SYCL-DNN is a library implementing neural network algorithms written using SYCL.
  - C++

- **sycl-ml**
  - SYCL-ML is a C++ library implementing classical machine learning algorithms using SYCL.
  - C++

- **oneMKL**
  - oneMKL Interfaces is an open-source implementation of the oneMKL Data Parallel C++ (DPC++) Library.
  - C++

- **clsv**
  - Clsv is a prototype implementation of OpenCL 3.0 on top of Vulkan using clsv as the compiler.
  - C++

- **cvk**
  - Cvk is a prototype implementation of OpenCL 3.0 on top of Vulkan using clsv as the compiler.
  - C++

- **Eigen**
  - Collection of samples and utilities for using ComputeCpp, Codeplay’s SYCL implementation.
  - C++

- **visioncpp**
  - A machine vision library written in SYCL and C++ that shows performance-portable implementation of graph algorithms.
  - C++

- **TensorFlow™**
  - Collection of samples and utilities for using ComputeCpp, Codeplay’s SYCL implementation.
  - C++
Khronos Safety Critical Standards Evolution

Khronos has 20 years experience in standards for safety-critical markets

Leveraging proven mainstream standards with shipping implementations and developer tooling and familiarity

A choice of abstraction levels to suit different markets and developer needs

OpenGL ES 1.0 - 2003
Fixed function graphics

OpenGL ES 2.0 - 2007
Programmable Shaders

Vulkan 1.2 - 2020
Explicit Graphics and Compute and Display

OpenGL SC 1.0 - 2005
Fixed function graphics safety-critical subset

OpenGL SC 2.0 - 2016
Programmable Shaders Safety-critical subset

Vulkan SC 1.0 - 2022
Explicit Graphics, Compute and Display safety-critical subset

OpenVX SC Extension - 2017
Graph-based vision and inferencing

OpenVX 1.3 - 2019
SC Extension integrated into core OpenVX specification

OpenVX - 2019

SYCL 2020
C++-based heterogeneous parallel programming

SYCL SC Extension

March 2022
SYCL SC Working Group announced to develop C++-based heterogeneous parallel compute programming framework for safety-critical systems

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

Currently there is no native AUTOSAR functionality to utilize hardware accelerators for high performance computation. Only way is to integrate 3rd party libraries which can affect safety.

At the same time there is a challenge for AUTOSAR Adaptive Platform to cover cutting-edge functionality like:
• AD/ADAS systems
• Performing heavy algorithms
• AI
• etc.

Thank you to AUTOSAR and Intellias

The main goal of this concept is to enable parallel heterogeneous programming, using standardized C++ based API, for solving issue of high performance computing.

Important part of the concept is to consider ISO-26262 Standard without sacrificing of performance.
Final words

Programming Models Must Persist but also be high quality and portable with conformance tests
SYCL 2020 Launched February 2021
SYCL user and developer Phenomenal Growth
Easy to build SYCL on any device
SYCL is mainstream
Market needs SYCL to Evolve with more workloads
SYCL thriving community is our most important asset
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 of European/Asia projects
Enabling Industry Engagement (2023/04/18)

- 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 CTS
  - https://github.com/KhronosGroup/SYCL-CTS

- Advisory Panel Chaired by Tom Deakin of U of Bristol
  - Apr 25, 2023 4pm UK

- Regular meetings to give feedback on roadmap and draft specifications

• Open to all!
  - https://community.khronos.org
  - 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/

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/

Khrnos members
- https://www.khronos.org/members/
- https://www.khronos.org/registry/SYCL/

Khrnos SYCL Forums, Slack Channels, Stackoverflow, reddit, and SYCL.tech

Khrnos GitHub
Contribute to SYCL open source specs, CTS, tools and ecosystem

Khrnos SYCL Working Group
SYCL Advisory Panels
SYCL CTS

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

SYCL Practitioners Hackathon

Tutorial 1: Introduction to SYCL [1996]
Course Leaders: Christopher Edsall, University of Heidelberg
09:15 – 17:00 GMT
▶ show / hide abstract

Tutorial 2: SYCL Techniques and Best Practices
Tutorial Lead: Rod Burns, Codeplay Software
09:15 – 17:00 GMT
▶ show / hide abstract