SYCL State of the Union Keynote
SYCLcon 2022
Specification Release

Michael Wong
SYCL
Codeplay Distinguished Eng.
ISO/IEC JTC1/SC22/WG21
ISO C++ Director
michael@codeplay.com | wongmichael.com
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
SYCL user and developer Phenomenal Growth

Star history

600X growth over 7 years

Some open-source SYCL implementations/prototypes on GitHub

This work is licensed under a Creative Commons Attribution 4.0 International License
Github SYCL source files is over 30000

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

17.5k mentions of #include <CL/sycl.hpp> in Jan, 2022
21 K in April, 2022
## Market needs SYCL: easy to build SYCL on any device

<table>
<thead>
<tr>
<th>2016</th>
<th>2022</th>
</tr>
</thead>
<tbody>
<tr>
<td>- 21 projects</td>
<td>- 147 projects</td>
</tr>
<tr>
<td>- 2 implementations in work</td>
<td>- &gt;12 implementations in work</td>
</tr>
<tr>
<td>- 3-4 platforms</td>
<td>- 10+ platforms</td>
</tr>
<tr>
<td>- SYCL was a TSG of OpenCL</td>
<td>- SYCL is independent WG of Khronos</td>
</tr>
<tr>
<td>- 10 members attending TSG</td>
<td>- 20+ members attending WG</td>
</tr>
<tr>
<td>- &lt;10 GitHub stars</td>
<td>- &gt;600 GitHub stars</td>
</tr>
<tr>
<td>- &lt;10 StackOverflow questions</td>
<td>- &gt;600 StackOverflow questions</td>
</tr>
<tr>
<td>- No Safety Critical</td>
<td>- Safety Critical Exploratory Group</td>
</tr>
<tr>
<td>- No book, a few articles</td>
<td>- 1 book, many articles</td>
</tr>
<tr>
<td>- &lt; 100 GitHub include code</td>
<td>- ~30000 GitHub include code</td>
</tr>
<tr>
<td>- No Supercomputing presence</td>
<td>- SC BoF 5 years in a role</td>
</tr>
<tr>
<td>- No automotive</td>
<td>- Renesas R-Car</td>
</tr>
<tr>
<td>- No Clang/LLVM</td>
<td>- Clang/LLVM DPC++ active</td>
</tr>
<tr>
<td>- No common one-stop website</td>
<td>- SYCL.tech</td>
</tr>
<tr>
<td>- No common teaching material</td>
<td>- SYCL Academy</td>
</tr>
<tr>
<td>- No HPC</td>
<td>- ~7 HPC systems</td>
</tr>
</tbody>
</table>
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
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 gou...l free to suggest a topic in the comments.

<table>
<thead>
<tr>
<th>Topics</th>
<th>Interest Level</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI/ML in Programming Systems</td>
<td>21</td>
<td>45.7%</td>
</tr>
<tr>
<td>Challenges in Program Syntax</td>
<td>14</td>
<td>30.4%</td>
</tr>
<tr>
<td>Enabling Codeign with Progeny</td>
<td>11</td>
<td>23.9%</td>
</tr>
<tr>
<td>Performance, debugging, co-design, and verification</td>
<td>15</td>
<td>32.6%</td>
</tr>
<tr>
<td>Challenges in Language Level</td>
<td>12</td>
<td>26.1%</td>
</tr>
<tr>
<td>Infrastructure for DSL creation and systems</td>
<td>15</td>
<td>32.6%</td>
</tr>
<tr>
<td>Distributed Memory Models</td>
<td>12</td>
<td>26.1%</td>
</tr>
<tr>
<td>Programming novel von Neuman models</td>
<td>13</td>
<td>28.3%</td>
</tr>
<tr>
<td>Verification/Correctness: when first seen</td>
<td>17</td>
<td>37%</td>
</tr>
<tr>
<td>Challenges from Sparsity, 1rrgularity, and performance</td>
<td>12</td>
<td>26.1%</td>
</tr>
<tr>
<td>Performance portability on heterogeneous platforms</td>
<td>12</td>
<td>26.1%</td>
</tr>
<tr>
<td>Mapping and Scheduling on heterogeneous platforms</td>
<td>17</td>
<td>37%</td>
</tr>
<tr>
<td>Domains in need of a DSL?</td>
<td>19</td>
<td>41.3%</td>
</tr>
<tr>
<td>Task-based programming models</td>
<td>1</td>
<td>2.2%</td>
</tr>
<tr>
<td>Programming Systems Sustained for random</td>
<td>1</td>
<td>2.2%</td>
</tr>
<tr>
<td>Runtime adaptivity including automated machine programs</td>
<td>1</td>
<td>2.2%</td>
</tr>
</tbody>
</table>

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

This work is licensed under a Creative Commons Attribution 4.0 International License
Specialization is more attractive now but Volume is needed
From Flops to TOPS in ML

SYCL can serve even more Extreme Heterogeneity

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)

Data Movement Is still King

BERT encoder

Output

Linear

Add & LayerNorm

Feedforward net ReLU

Multi-head attention

Positional embedding

Input embedding

Scaled dot-product attention softmax

Concetenate

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

ARTICLE

1. Introduction

Transformers (Narasimhan et al., 2017) are a class of deep neural network architectures for sequence transduction (e.g.,自然语言处理, NLP) with a wide applicability, e.g., BERT (Devlin et al., 2018) and GPT-3 (Radford et al., 2019). They have become a major expansion in machine learning, e.g., deep neural networks (DNNs) (Bengio et al., 2016) and language models (LMs) (Radford et al., 2019).

Our main research has a focus on deep neural networks and their applications in industry and academia. In particular, we are interested in understanding and improving the efficiency of deep neural networks, which is a critical aspect of their success. This includes understanding the computational requirements and optimizing the performance of these models.

2. Related Work

Many previous studies have investigated the impact of data movement on the performance of deep neural networks. For example, (Choromanska et al., 2015) showed that data movement can significantly impact the training time of neural networks. (Goyal et al., 2017) found that data movement can impact the accuracy and efficiency of deep neural networks. (Sutskever et al., 2014) demonstrated that data movement can have a significant impact on the training time and accuracy of deep neural networks.

3. Methodology

Data Movement (DM) is a strategy that focuses on minimizing the amount of data movement required during the training of deep neural networks. DM involves optimizing the model structure and training process to reduce the amount of data movement required.

4. Results

We evaluated the impact of Data Movement on the performance of BERT and GPT-3. Our results show that Data Movement can significantly improve the efficiency of deep neural networks, leading to faster training times and higher accuracy.

5. Conclusion

Data Movement is a critical factor in the performance of deep neural networks. By minimizing data movement, we can significantly improve the efficiency of deep neural networks, leading to faster training times and higher accuracy. Further research is needed to fully understand the impact of Data Movement on deep neural networks and to develop more effective strategies for optimizing data movement.

This work is licensed under a Creative Commons Attribution 4.0 International License
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...)
- Open Source and Extensible Compiler Frameworks (LLVM, MLIR)
- AI integrated applications
- Programming Abstractions, Frameworks, and Languages (SYCL, Kokkos, Raja)

Reimagining Codesign 2021 Position Papers (osti.gov)
Parallel Industry Initiatives

<table>
<thead>
<tr>
<th>Initiative</th>
<th>Year</th>
<th>Language Support</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++11</td>
<td>2011</td>
<td>C++11</td>
<td>SYCL 1.2 C++11 Single source programming</td>
</tr>
<tr>
<td>C++14</td>
<td>2015</td>
<td>C++11, C++20</td>
<td>SYCL 1.2.1 C++11 Single source programming, OpenCL 2.1, SPIR-V in Core</td>
</tr>
<tr>
<td>OpenCL 1.2</td>
<td></td>
<td></td>
<td>OpenCL 1.2 OpenCL C Kernel Language</td>
</tr>
<tr>
<td>OpenCL 2.1</td>
<td></td>
<td></td>
<td>OpenCL 2.1, SPIR-V in Core</td>
</tr>
<tr>
<td>SYCL 1.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C++17</td>
<td>2017</td>
<td>C++17</td>
<td>SYCL 2020 C++17 Single source programming, Many backend options</td>
</tr>
<tr>
<td>SPIR-V in Core</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenCL 2.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYCL 2020</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C++20</td>
<td>2020</td>
<td>C++20</td>
<td>SYCL 202X C++20 Single source programming, Many backend options</td>
</tr>
<tr>
<td>OpenCL 3.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYCL 202X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

DPC++
- Uses LLVM/Clang
- Part of oneAPI
- Any CPU
- Intel CPUs
- Intel GPUs
- Intel FPGAs
- NVIDIA GPUs
- OpenCL
- SPIR

ComputeCpp
- Multiple Backends
- Any CPU
- Intel CPUs
- Intel GPUs
- Intel FPGAs
- ARM Mali
- IMG PowerVR
- Renesas R-Car

hipSYCL
- Multiple Backends
- Any CPU
- AMD GPUs
- NVIDIA GPUs
- OpenCL
- SPIR

Level Zero
- Intel GPUs
- NVIDIA GPUs
- OpenCL
- SPIR

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

SYCL Experimental Development (2022/05/01)
Migration advice by Platform (2022/05/05): For a device kind, which SYCL compiler?

- 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
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

SYCL source code

clang++ -fsycl -fsycl-targets=nvptx64-nvidia-cuda
clang++ -fsycl -fsycl-targets=amdgcn-amd-amdhsa

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

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

3 Pre-exascale
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!

Simplify Heterogeneous Development
From Diverse Programming Approaches

CPU programming model | GPU programming model | FPGA programming model | Other accel. programming models

CPU | GPU | FPGA | Other accelerators

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

ISO C++ & Khronos SYCL & community extensions

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

*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

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

```cpp
void DiskGalaxySimulator::stepSim() {
    // Compute updated positions.
    constexpr int wq_size = 256;
    int nblocks = ((getNumParticles() - 1) / wq_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
    // dpt.
    auto start = std::chrono::steady_clock::now();
    for (size_t i = 0; i < params.maxIterationsPerFrame; i++) {
        for (particle_interaction<nblocks, wq_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();
}
```

SYCL

```cpp
void DiskGalaxySimulator::stepSim() {
    // Compute updated positions.
    const auto wq_size = 256;
    int nblocks = ((get_num_particles() - 1) / wq_size) + 1;

    auto start = std::chrono::steady_clock::now();
    for (size_t i = 0; i < params.max_iterations_per_frame; i++) {
        auto dpt = 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<sycl::range<sycl::range<nblocks, wq_size>>>() {
                particle_interaction(pos_d_ct0, pos_next_d_ct1, vel_d_ct2, params_ct3, item_ct1);
            });
        });
        std::swap(pos_d, pos_next_d);
        gpuErrchk(dpt.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();
}
```

Code changes are minimal
Performance Achieved

N-body demo running with CUDA on device: NVIDIA GeForce RTX 3060
Kernel time: 10.20 ms

N-body demo running with DPC++ on device: NVIDIA GeForce RTX 3060
Kernel time: 8.79 ms

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

Khronos SYCL Safety-Critical Exploratory Forum

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

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

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.

This work is licensed under a Creative Commons Attribution 4.0 International License
Khronos AUTOSAR Liaison What next?

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

<table>
<thead>
<tr>
<th>Document Title</th>
<th>Design guidelines for using parallel processing technologies on Adaptive Platform AUTOSAR AP R19-11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document Owner</td>
<td>AUTOSAR</td>
</tr>
<tr>
<td>Document Responsibility</td>
<td>AUTOSAR</td>
</tr>
<tr>
<td>Document Identification No</td>
<td>3584</td>
</tr>
<tr>
<td>Document Status</td>
<td>published</td>
</tr>
<tr>
<td>Part of AUTOSAR Standard</td>
<td>Adaptive Platform</td>
</tr>
<tr>
<td>Part of Standard Release</td>
<td>R19-11</td>
</tr>
</tbody>
</table>

### 3D Graphics
- Desktop, Mobile and Web
- Portable XR
  - Augmented and Virtual Reality

### 3D Assets
- Authoring and Delivery
- Portable XR
  - Augmented and Virtual Reality

### Parallel Computation
- Vision, Inferencing, Machine Learning
  - SYCL/OpenCL
  - Safety Critical APIs

![Diagram of AUTOSAR Working Groups and Project Leader Team](image)
Case Study: Automotive AI

- 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

“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 of 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
  - 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.khrionos.org/registry/SYCL/
  - Khronos SYCL Forums, Slack Channels, Stackoverflow, reddit, and SYCL.tech
    - 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://github.com/KhronosGroup/SYCL
t- Khronos GitHub
  - https://github.com/KhronosGroup/SYCL
  - https://github.com/KhronosGroup/SyclParallelSTL
  - https://github.com/codeplaysoftware/syclacademy
  - https://sycl.tech/