SYCL SC State of the Union

Lukas Sommer, Codeplay Software

On behalf of the SYCL SC working group

Agenda

- **SYCL SC Working Group**
- SYCL SC Motivation
- Past Year Overview
- Ecosystem & Engagement

SYCL SC Working Group Officers

Co-Chair & Spec Editor

Andriy Byzhynar

Co-Chair & Outreach Officer

Leonidas Kosmidis

SYCL SC Working Group Regular Members

Khronos Safety Critical Standards Evolution

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

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

OpenVX SC Extension – 2017
Graph-based vision and inferencing

SYCL 2020
C++-based heterogeneous parallel programming

March 2023
SYCL SC Working Group created to develop C++-based heterogeneous parallel compute programming framework for safety-critical systems

OpenVX 1.3 – 2019
SC Extension integrated into core OpenVX specification

Agenda

- SYCL SC Working Group
- **SYCL SC Motivation**
- Past Year Overview
- Ecosystem & Engagement

What is “Safety-Critical”?

- **A system is *safety-critical* if its failure could result in harm to people or property**
- **SC industries:** automotive, avionics, medical, rail, atomic
- **Often certified according to standards**
  - Automotive: ISO 26262
  - Avionics: DO-178C
  - Medical: IEC 62304
- **Standards define safety levels:** ASIL A-D / DAL A-E / Class A-C
- **Require *functional safety***
  - Absence of unreasonable risk caused by malfunction
    - Use cases must be defined
    - Risks must be analyzed and mitigated to a reasonable level
  - A property of the whole system, not just an individual component
  - Only somewhat related to programming language safety
  - Engineering processes & paper trails

What Hardware are SC Industries Using?

- **Automotive leads performance requirements**
  - **Tesla FSD 2:** 50 TOPS
  - **Intel A760A Automotive:** 229 TOPS
  - **Nvidia DRIVE Thor:** 1000 TOPS
- **Desktop-class GPUs**

Sources:
- https://www.autopilotreview.com/tesla-hardware-4-rolling-out-to-new-vehicles/
- https://download.intel.com/newsroom/2024/automotive/Intel-auto-dGPU-fact-sheet.pdf
- https://blogs.nvidia.com/blog/drive-thor/

TOPS = Operations / second x 10^{12}

Why C++ for Functional Safety?

- **The C++ language and C++ best practice are mature and under continuous improvement**
- **Programming language safety is only one of many concerns that need to be balanced**
- **Other concerns:**
  - **Development tools** are needed for productivity
  - **Analysis tools** are needed to support safety arguments
    - Static analyzers
    - Sanitizers
    - Performance analysis
  - **Optimized libraries** are needed for performance & portability
  - **Coding guidelines** are required by functional safety development processes
  - **Many experienced developers** are required to produce quality software at scale

The C++ ecosystem is very attractive to SC industries

What do SC Industries Need?

"Software-Defined Vehicle"

- Automotive trend
- Many small devices consolidated into few large devices
- Software increasingly important product differentiator
- Software increasingly important as value-add

Hardware Acceleration

- Greater maximum performance
- Better performance-per-power

???

Determinism

- The software behaves predictably
- The software does the same thing in the same way every time

Code Portability & Reuse

- As amount of code grows, economics of rewriting become less feasible
- Vendor lock-in a big risk

Safety Arguments

- Structured evidence to show that a system is sufficiently safe for the intended uses
- For software, coding guidelines (e.g., MISRA) are crucial to argumentation

Goals of SYCL SC

Software-Defined Vehicle
- Applications include ADAS/AD, infotainment, etc.
- Benefits from C++ tooling & ecosystem
- Many experienced C++ developers
- See also: AUTOSAR SYCL Demonstrator

Hardware Acceleration
- Exposes GPUs & other accelerators
- Focused on productivity

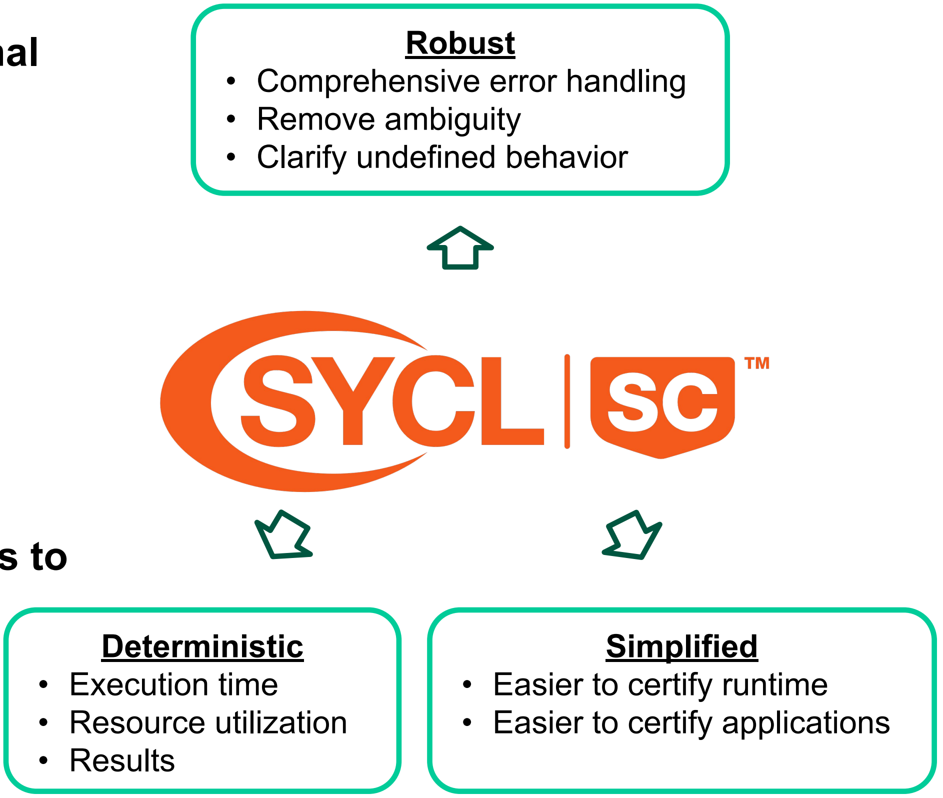
Determinism
- Identify sources of non-determinism
- API support to mitigate non-determinism

Code Portability & Reuse
- Portable across vendors & backends
- Improved ability to use libraries

Safety Arguments
- MISRA compliance with published deviations
- API features to simplify arguments (e.g., “Bug X cannot happen because of …”)

What SYCL SC *is*

- **Derived from SYCL 2020**
- **Improved compatibility with functional safety**
  - On the code level (e.g., MISRA compatibility)
  - As a component of a safe system (e.g., support safety arguments)
- **Core philosophies:**
  - Robustness
  - Determinism
  - Simplification
- **Removals, modifications, & additions to SYCL 2020 in support of core philosophies**


Robust
• Comprehensive error handling
• Remove ambiguity
• Clarify undefined behavior
SYCL SC™
Deterministic
• Execution time
• Resource utilization
• Results
Simplified
• Easier to certify runtime
• Easier to certify applications

What SYCL SC is *Not*

SYCL SC will not

- Tell you how to implement a “safe” application
- Guarantee a safe application
- Tell you how to implement a “safe” SYCL SC runtime
- Guarantee a safe runtime
- Tell you how to apply any industry process or standard
- Be certified (as a standard)
- Make your hardware safe

- **SYCL SC will be compatible with you doing the above, but it cannot do it for you**
- **Functional safety is a system property; SYCL SC is only one component**
  - A safe system includes many other components, e.g., redundant hardware, EDC/ECC, hardware & software monitors, compliant development process, etc.

Agenda

- SYCL SC Working Group
- SYCL SC Motivation
- **<u>Past Year Overview</u>**
- Ecosystem & Engagement

Topics from the Past Year

⭐ = Headline feature

⊕ = More later

- **Error model** ⭐
  - How to handle errors without C++ exceptions
- **Object model & dynamic memory** ⭐
  - How to ensure determinism
- **C++ std containers …** ⊕
- **Device selection …** ⊕
- **Host-side thread safety**
- **Fallback queue**
- **Introspection**
  - E.g., kernel queries
- **Alignment with Base SYCL vs. changes to support certifiability**
  - Often a spectrum of possible solutions with various trade-offs
- **Specification management**
  - Git workflows

Example Topic: C++ std Containers

- **Base SYCL uses std::vector for both inputs and outputs**
- **Problematic for functional safety:**
  - May throw
  - Amortized complexity
    - SC cares about worst-case behavior
  - May allocate
    - Non-deterministic time
    - Non-deterministic space
    - Fragmentation in underlying allocator
- **C++ allocators & polymorphic allocators do not solve problems**
- **Require clearly specified rules on:**
  - Containers’ & contained objects’ lifetimes
  - Usage & ownership of underlying memory
- **C++26 std::inplace_vector has some beneficial features**
  - Doesn’t address all issues with std::vector
  - SYCL SC must use C++17 because MISRA guidelines available

Example Topic: Device Selection

- **Base SYCL uses *selectors***
- **Select devices based on a score**
- **Implementation-defined behavior in case of ties**
    - Under-specified for SC requirements
- **Default selector called by various Base SYCL functions**
- **Particularly problematic when a system has two or more identical GPUs**
    - Default selector is allowed to always return the same one
    - Default selector is allowed to return one at random
    - Potential source of non-determinism

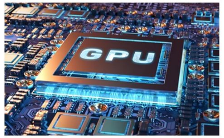
Agenda

- SYCL SC Working Group
- SYCL SC Motivation
- Past Year Overview
- **Ecosystem & Engagement**

Khronos AUTOSAR Liaison: SYCL Demonstrator

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.

The main aim: creation of generic API in AUTOSAR, which allows to utilize hardware acceleration for computation efficiency improvement. SYCL is the best candidate to be used under the hood. Moreover, SYCL SC will potentially add required safety compatibility.

Thank you to AUTOSAR and Intellias

AUTOSAR

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.

intellias
Intelligent Software Engineering

ISO 26262

Get Involved!

- www.khronos.org/syclsc
- sycl_sc_chair@lists.khronos.org
- **SYCL SC working group welcomes new members**
  - Open to Khronos members
    - Khronos membership is available to all companies
  - Individual contributors possible in special circumstances
- **Advisory panel for IP-free, non-Khronos discussions**
- **See also UXL Safety-Critical Special Interest Group (SC SIG)**
  - https://github.com/uxlfoundation/foundation#special-interest-groups-sigs
  - Khronos liaison organization
  - Open for IP-free community presentations & discussions

13th International Workshop on OpenCL and SYCL

IWOCL 2025

SYCL SC State of the Union

Lukas Sommer, Codeplay Software

On behalf of the SYCL SC working group