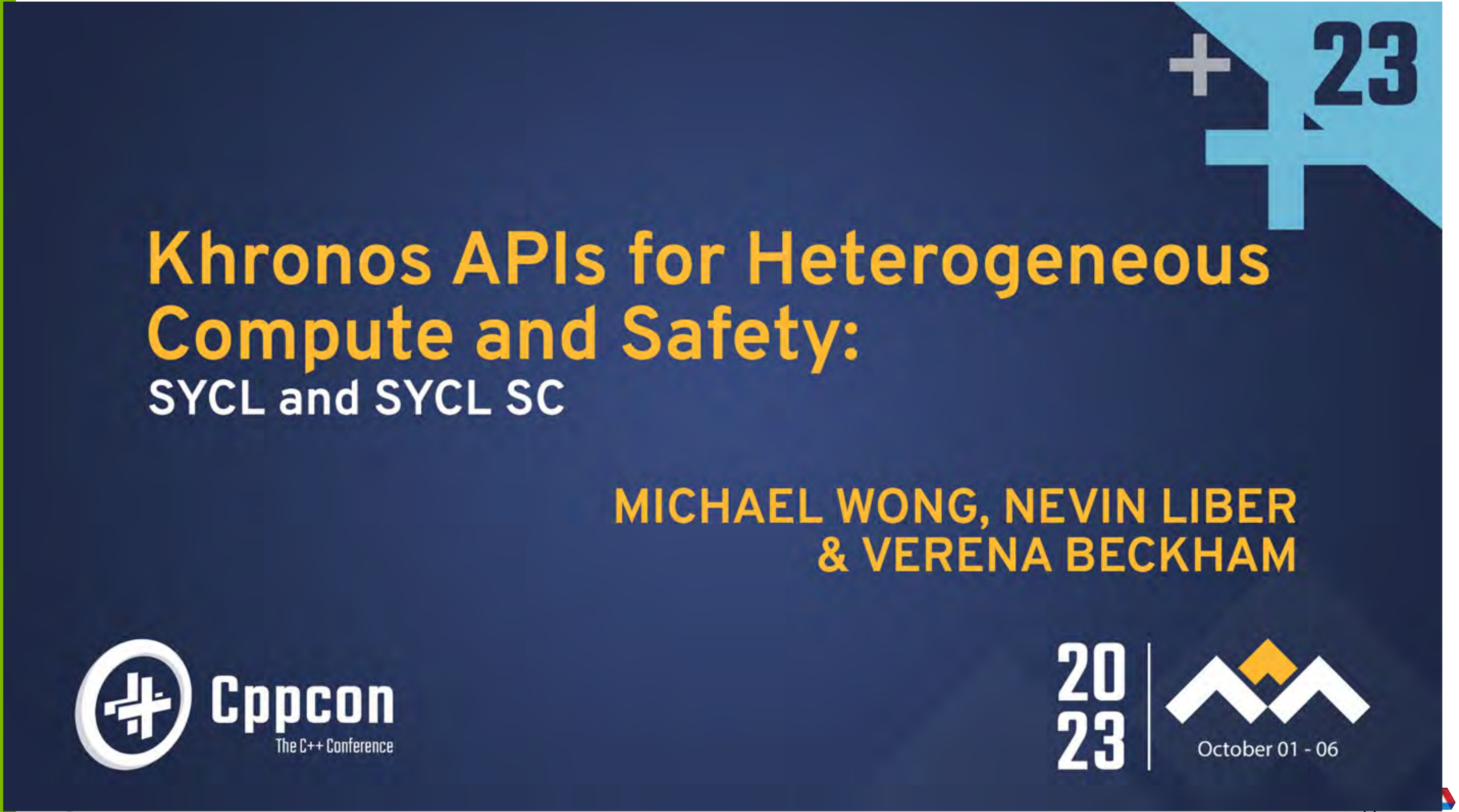


Khronos APIs for Heterogeneous Compute and Safety:

SYCL and SYCL SC

MICHAEL WONG, NEVIN LIBER
& VERENA BECKHAM

Cppcon
The C++ Conference
2023
October 01 - 06
+23

NATIONAL LABORATORY

SYCL IS MAINSTREAM

Open Standards and Open Source implementations, community driven

Open cross-company collaboration

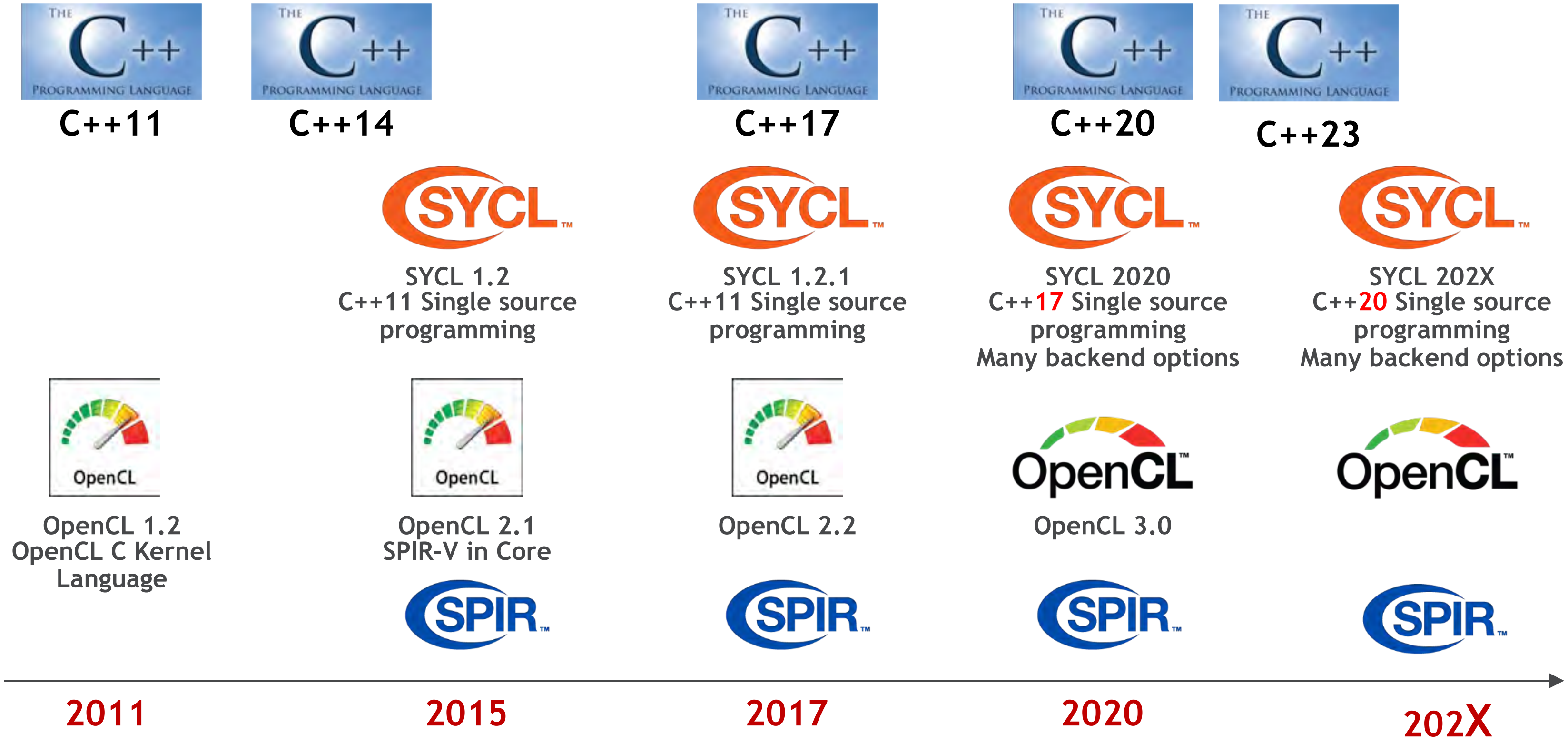
Co-design for all forms of extreme heterogeneity

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

PARALLEL INDUSTRY INITIATIVES



C++11

OpenCL 1.2
OpenCL C Kernel
Language

2011

C++14

SYCL 1.2
C++11 Single source
programming

OpenCL 2.1
SPIR-V in Core

2015

C++17

SYCL 1.2.1
C++11 Single source
programming

OpenCL 2.2

2017

C++20

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

OpenCL 3.0

2020

C++23

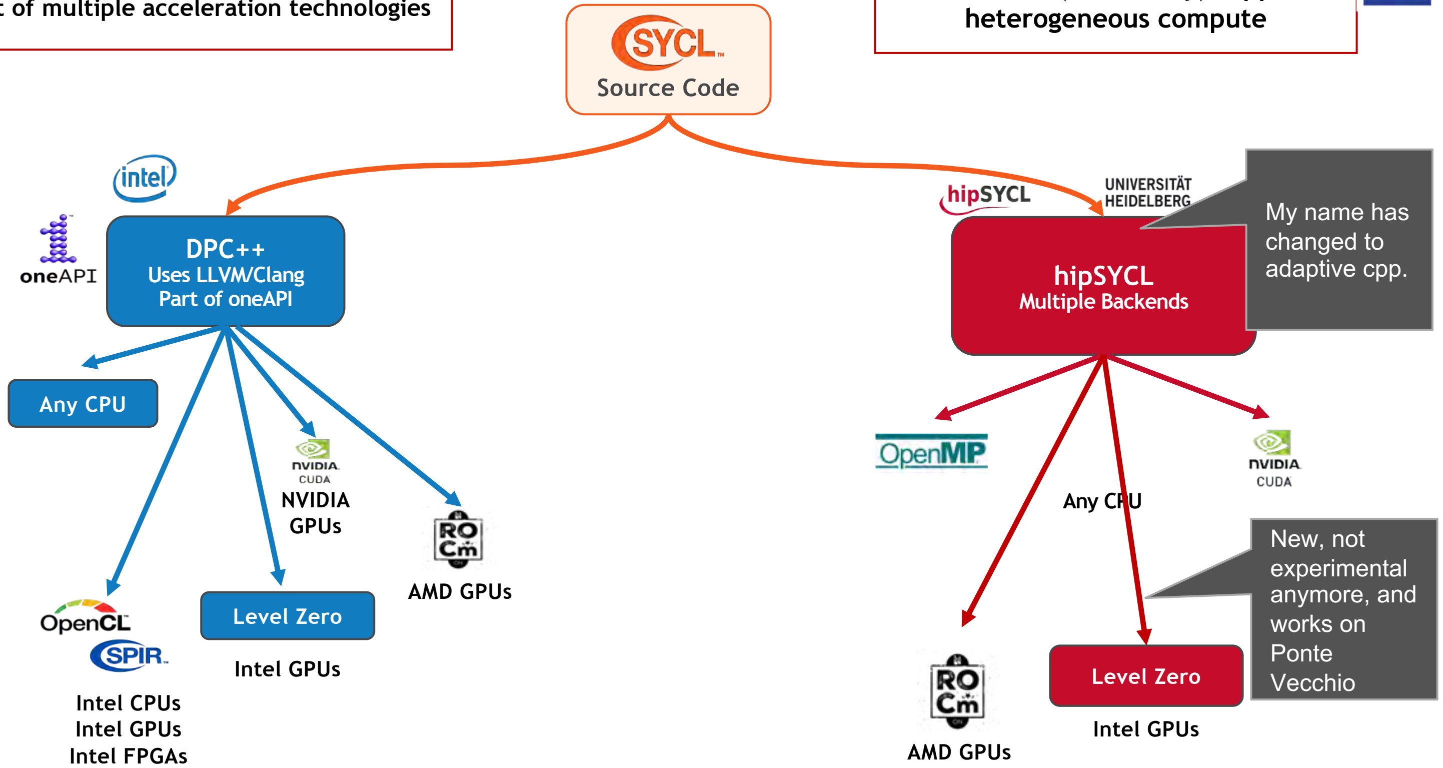
SYCL 202X
C++20 Single source
programming
Many backend options

202X

SYCL IMPLEMENTATIONS IN DEVELOPMENT

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



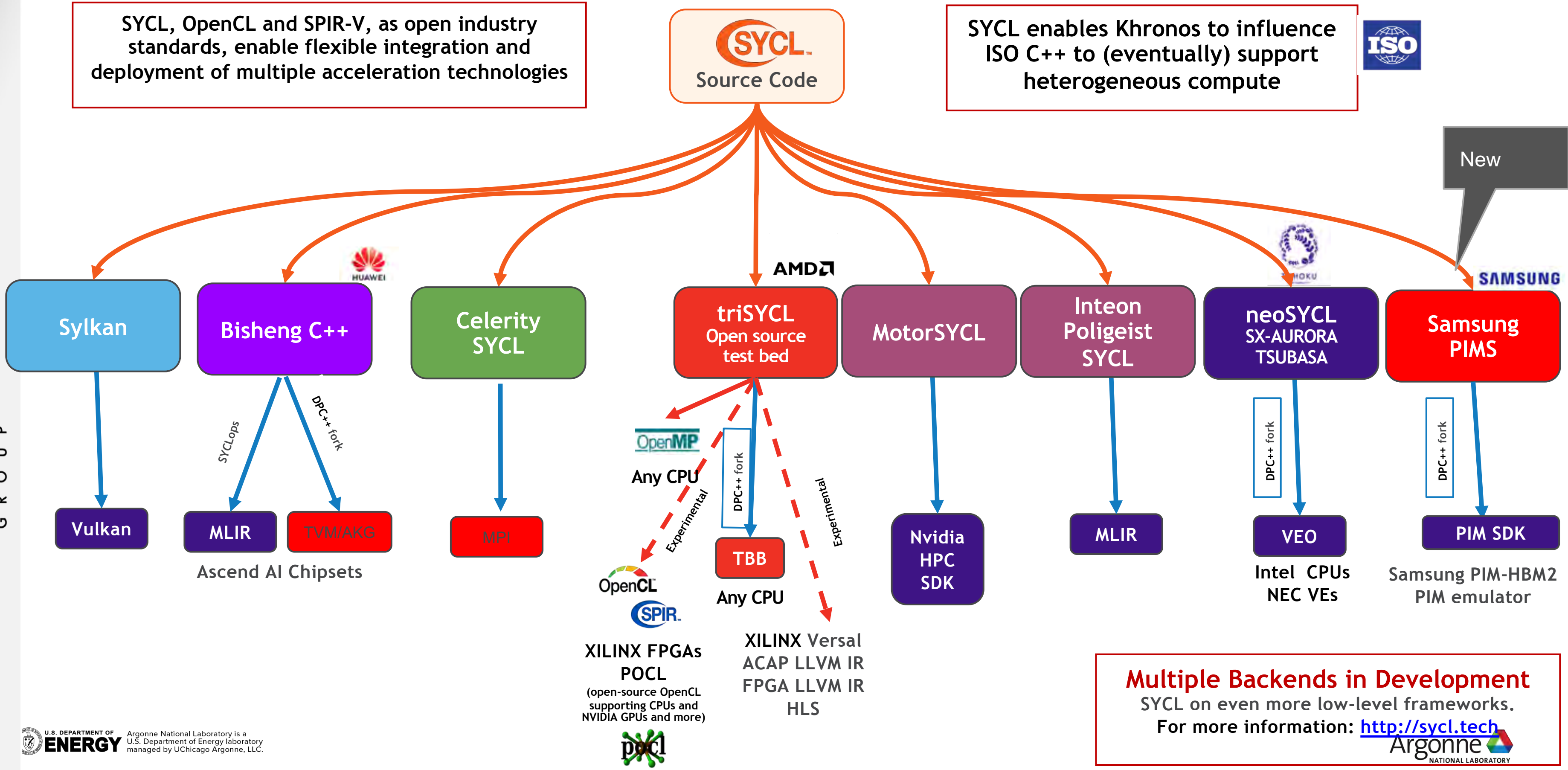
My name has changed to adaptive cpp.

New, not experimental anymore, and works on Ponte Vecchio

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



SYCL Source Code

New

Sylkan → Vulkan

Bisheng C++ → MLIR (SYCLops), TVM/AKG (DPC++ fork)

Ascend AI Chipsets

Celerity SYCL → MPI

triSYCL Open source test bed → OpenMP Any CPU; TBB Any CPU (DPC++ fork); OpenCL SPIR (Experimental); XILINX Versal (Experimental)

XILINX FPGAs POCL (open-source OpenCL supporting CPUs and NVIDIA GPUs and more)

XILINX Versal ACAP LLVM IR FPGA LLVM IR HLS

MotorSYCL → Nvidia HPC SDK

Inteon Poligeist SYCL → MLIR

neoSYCL SX-AURORA TSUBASA → VEO (DPC++ fork)

Intel CPUs NEC VEs

Samsung PIMS → PIM SDK (DPC++ fork)

Samsung PIM-HBM2 PIM emulator

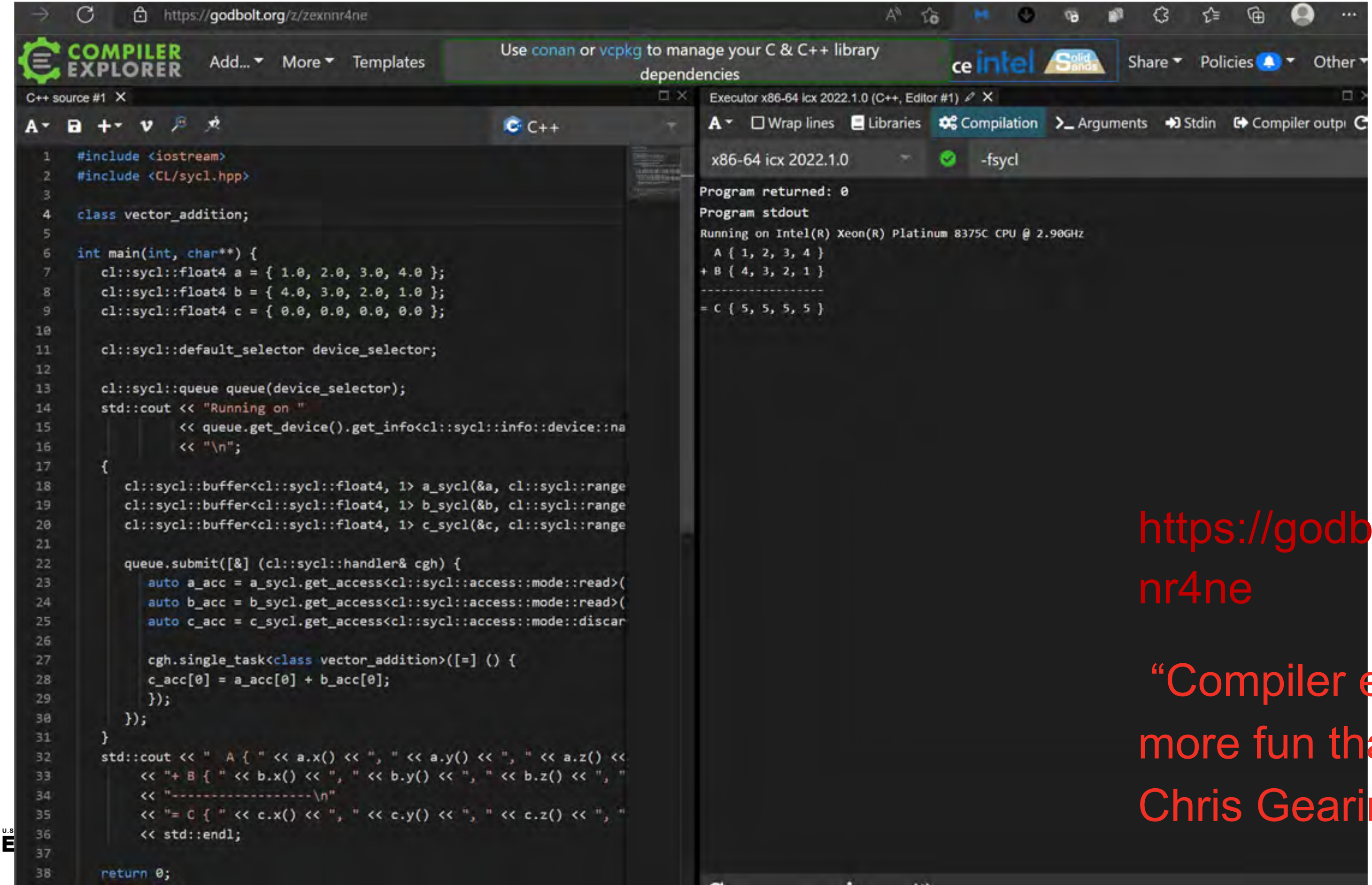
Multiple Backends in Development

SYCL on even more low-level frameworks.

For more information: http://sycl.tech

U.S. DEPARTMENT OF ENERGY Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

SYCL ON COMPILER EXPLORER (CPU EXECUTION)


https://godbolt.org/z/zexnnr4ne
COMPILER EXPLORER
Add...
More
Templates
Use conan or vcpkg to manage your C & C++ library dependencies
Share
Policies
Other
C++ source #1
C++
Executor x86-64 icx 2022.1.0 (C++, Editor #1)
Wrap lines
Libraries
Compilation
Arguments
Stdin
Compiler output

```cpp
#include <iostream>
#include <CL/sycl.hpp>

class vector_addition;

int main(int, char**) {
   cl::sycl::float4 a = { 1.0, 2.0, 3.0, 4.0 };
   cl::sycl::float4 b = { 4.0, 3.0, 2.0, 1.0 };
   cl::sycl::float4 c = { 0.0, 0.0, 0.0, 0.0 };

   cl::sycl::default_selector device_selector;

   cl::sycl::queue queue(device_selector);
   std::cout << "Running on "
             << queue.get_device().get_info<cl::sycl::info::device::na
             << "\n";
   {
      cl::sycl::buffer<cl::sycl::float4, 1> a_sycl(&a, cl::sycl::range
      cl::sycl::buffer<cl::sycl::float4, 1> b_sycl(&b, cl::sycl::range
      cl::sycl::buffer<cl::sycl::float4, 1> c_sycl(&c, cl::sycl::range

      queue.submit([&] (cl::sycl::handler& cgh) {
         auto a_acc = a_sycl.get_access<cl::sycl::access::mode::read>(
         auto b_acc = b_sycl.get_access<cl::sycl::access::mode::read>(
         auto c_acc = c_sycl.get_access<cl::sycl::access::mode::discar

         cgh.single_task<class vector_addition>([=] () {
         c_acc[0] = a_acc[0] + b_acc[0];
         });
      });
   }
   std::cout << "  A { " << a.x() << ", " << a.y() << ", " << a.z() <<
         << "+ B { " << b.x() << ", " << b.y() << ", " << b.z() << ", "
         << "------------------\n"
         << "= C { " << c.x() << ", " << c.y() << ", " << c.z() << ", "
         << std::endl;

   return 0;
```

x86-64 icx 2022.1.0 -fsycl

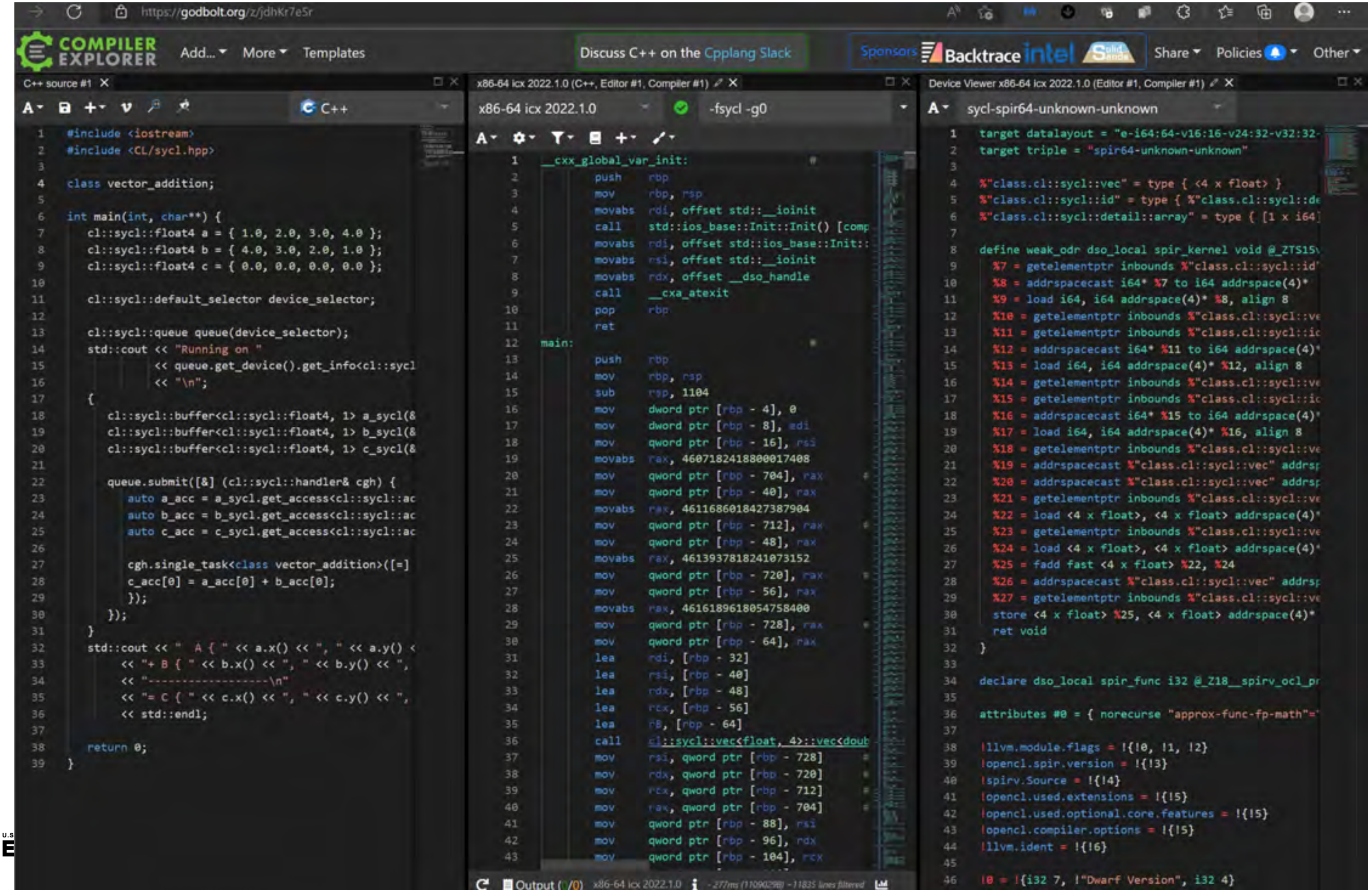
```
Program returned: 0
Program stdout
Running on Intel(R) Xeon(R) Platinum 8375C CPU @ 2.90GHz
  A { 1, 2, 3, 4 }
+ B { 4, 3, 2, 1 }
------------------
= C { 5, 5, 5, 5 }
```

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



C++ source #1 — C++

```
#include <iostream>
#include <CL/sycl.hpp>

class vector_addition;

int main(int, char**) {
   cl::sycl::float4 a = { 1.0, 2.0, 3.0, 4.0 };
   cl::sycl::float4 b = { 4.0, 3.0, 2.0, 1.0 };
   cl::sycl::float4 c = { 0.0, 0.0, 0.0, 0.0 };

   cl::sycl::default_selector device_selector;

   cl::sycl::queue queue(device_selector);
   std::cout << "Running on "
             << queue.get_device().get_info<cl::sycl
             << "\n";
   {
      cl::sycl::buffer<cl::sycl::float4, 1> a_sycl(&
      cl::sycl::buffer<cl::sycl::float4, 1> b_sycl(&
      cl::sycl::buffer<cl::sycl::float4, 1> c_sycl(&

      queue.submit([&] (cl::sycl::handler& cgh) {
         auto a_acc = a_sycl.get_access<cl::sycl::ac
         auto b_acc = b_sycl.get_access<cl::sycl::ac
         auto c_acc = c_sycl.get_access<cl::sycl::ac

         cgh.single_task<class vector_addition>([=]
         c_acc[0] = a_acc[0] + b_acc[0];
         });
      });
   }
   std::cout << "  A { " << a.x() << ", " << a.y() <
        << "+ B { " << b.x() << ", " << b.y() << ",
        << "------------------\n"
        << "= C { " << c.x() << ", " << c.y() << ",
        << std::endl;

   return 0;
}
```

x86-64 icx 2022.1.0 (C++, Editor #1, Compiler #1) — x86-64 icx 2022.1.0 — -fsycl -g0

```
__cxx_global_var_init:
        push    rbp
        mov     rbp, rsp
        movabs  rdi, offset std::__ioinit
        call    std::ios_base::Init::Init() [comp
        movabs  rdi, offset std::ios_base::Init::
        movabs  rsi, offset std::__ioinit
        movabs  rdx, offset __dso_handle
        call    __cxa_atexit
        pop     rbp
        ret
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 1104
        mov     dword ptr [rbp - 4], 0
        mov     dword ptr [rbp - 8], edi
        mov     qword ptr [rbp - 16], rsi
        movabs  rax, 4607182418800017408
        mov     qword ptr [rbp - 704], rax
        mov     qword ptr [rbp - 40], rax
        movabs  rax, 4611686018427387904
        mov     qword ptr [rbp - 712], rax
        mov     qword ptr [rbp - 48], rax
        movabs  rax, 4613937818241073152
        mov     qword ptr [rbp - 720], rax
        mov     qword ptr [rbp - 56], rax
        movabs  rax, 4616189618054758400
        mov     qword ptr [rbp - 728], rax
        mov     qword ptr [rbp - 64], rax
        lea     rdi, [rbp - 32]
        lea     rsi, [rbp - 40]
        lea     rdx, [rbp - 48]
        lea     rcx, [rbp - 56]
        lea     r8, [rbp - 64]
        call    cl::sycl::vec<float, 4>::vec<doub
        mov     rsi, qword ptr [rbp - 728]
        mov     rdx, qword ptr [rbp - 720]
        mov     rcx, qword ptr [rbp - 712]
        mov     rax, qword ptr [rbp - 704]
        mov     qword ptr [rbp - 88], rsi
        mov     qword ptr [rbp - 96], rdx
        mov     qword ptr [rbp - 104], rcx
```

Output (0/0) x86-64 icx 2022.1.0 - 277ms (11090298) ~11835 lines filtered

Device Viewer x86-64 icx 2022.1.0 (Editor #1, Compiler #1) — sycl-spir64-unknown-unknown

```
target datalayout = "e-i64:64-v16:16-v24:32-v32:32-
target triple = "spir64-unknown-unknown"

%"class.cl::sycl::vec" = type { <4 x float> }
%"class.cl::sycl::id" = type { %"class.cl::sycl::de
%"class.cl::sycl::detail::array" = type { [1 x i64]

define weak_odr dso_local spir_kernel void @_ZTS15v
  %7 = getelementptr inbounds %"class.cl::sycl::id"
  %8 = addrspacecast i64* %7 to i64 addrspace(4)*
  %9 = load i64, i64 addrspace(4)* %8, align 8
  %10 = getelementptr inbounds %"class.cl::sycl::ve
  %11 = getelementptr inbounds %"class.cl::sycl::ic
  %12 = addrspacecast i64* %11 to i64 addrspace(4)*
  %13 = load i64, i64 addrspace(4)* %12, align 8
  %14 = getelementptr inbounds %"class.cl::sycl::ve
  %15 = getelementptr inbounds %"class.cl::sycl::ic
  %16 = addrspacecast i64* %15 to i64 addrspace(4)*
  %17 = load i64, i64 addrspace(4)* %16, align 8
  %18 = getelementptr inbounds %"class.cl::sycl::ve
  %19 = addrspacecast %"class.cl::sycl::vec" addrsp
  %20 = addrspacecast %"class.cl::sycl::vec" addrsp
  %21 = getelementptr inbounds %"class.cl::sycl::ve
  %22 = load <4 x float>, <4 x float> addrspace(4)*
  %23 = getelementptr inbounds %"class.cl::sycl::ve
  %24 = load <4 x float>, <4 x float> addrspace(4)*
  %25 = fadd fast <4 x float> %22, %24
  %26 = addrspacecast %"class.cl::sycl::vec" addrsp
  %27 = getelementptr inbounds %"class.cl::sycl::ve
  store <4 x float> %25, <4 x float> addrspace(4)*
  ret void
}

declare dso_local spir_func i32 @_Z18__spirv_ocl_pr

attributes #0 = { norecurse "approx-func-fp-math"="

!llvm.module.flags = !{!0, !1, !2}
!opencl.spir.version = !{!3}
!spirv.Source = !{!4}
!opencl.used.extensions = !{!5}
!opencl.used.optional.core.features = !{!5}
!opencl.compiler.options = !{!5}
!llvm.ident = !{!6}

!0 = !{i32 7, !"Dwarf Version", i32 4}
```

SYCL 2020 CONFORMANCE TEST SUITE RELEASED

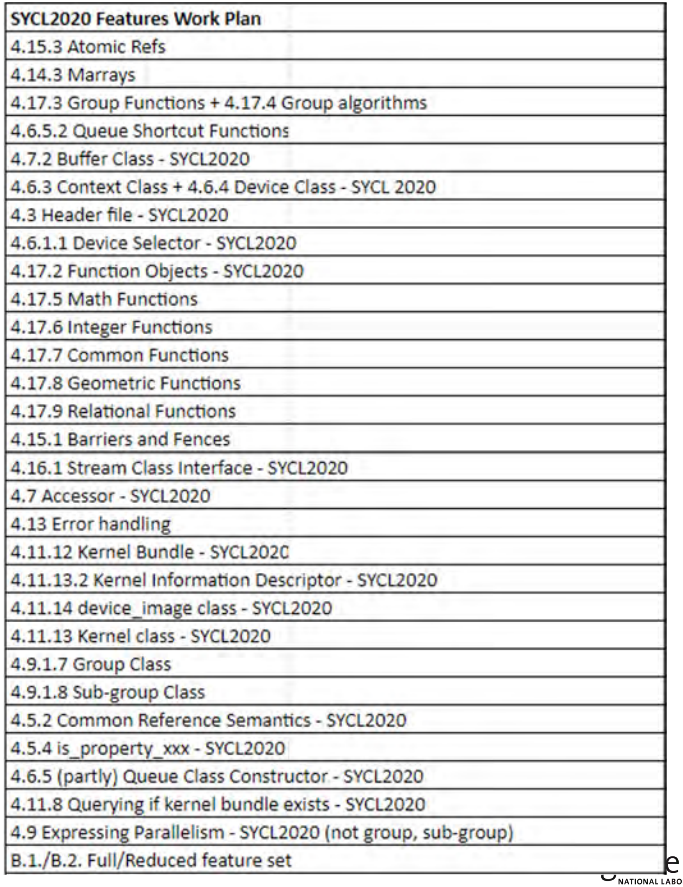
Expressiveness and simplicity for heterogeneous programming in modern C++

New Features

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

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

SYCL Conformance Test Suite



SYCL2020 Features Work Plan
4.15.3 Atomic Refs
4.14.3 Marrays
4.17.3 Group Functions + 4.17.4 Group algorithms
4.6.5.2 Queue Shortcut Functions
4.7.2 Buffer Class - SYCL2020
4.6.3 Context Class + 4.6.4 Device Class - SYCL 2020
4.3 Header file - SYCL2020
4.6.1.1 Device Selector - SYCL2020
4.17.2 Function Objects - SYCL2020
4.17.5 Math Functions
4.17.6 Integer Functions
4.17.7 Common Functions
4.17.8 Geometric Functions
4.17.9 Relational Functions
4.15.1 Barriers and Fences
4.16.1 Stream Class Interface - SYCL2020
4.7 Accessor - SYCL2020
4.13 Error handling
4.11.12 Kernel Bundle - SYCL2020
4.11.13.2 Kernel Information Descriptor - SYCL2020
4.11.14 device_image class - SYCL2020
4.11.13 Kernel class - SYCL2020
4.9.1.7 Group Class
4.9.1.8 Sub-group Class
4.5.2 Common Reference Semantics - SYCL2020
4.5.4 is_property_xxx - SYCL2020
4.6.5 (partly) Queue Class Constructor - SYCL2020
4.11.8 Querying if kernel bundle exists - SYCL2020
4.9 Expressing Parallelism - SYCL2020 (not group, sub-group)
B.1./B.2. Full/Reduced feature set

Significant SYCL adoption in Embedded, Desktop and HPC Markets

UNIFIED SHARED MEMORY

IMPLICIT VS EXPLICIT DATA MOVEMENT

Examples:

- SYCL, C++ AMP

Implementation:

- Data is moved to the device implicitly via cross host CPU / device data structures

Here we’re using C++ AMP as an example

```
array_view<float> ptr;
extent<2> e(64, 64);
parallel_for_each(e, [=](index<2> idx)
restrict(amp) {
  ptr[idx] *= 2.0f;
});
```

Examples:

- OpenCL, CUDA, OpenMP, SYCL 2020

Implementation:

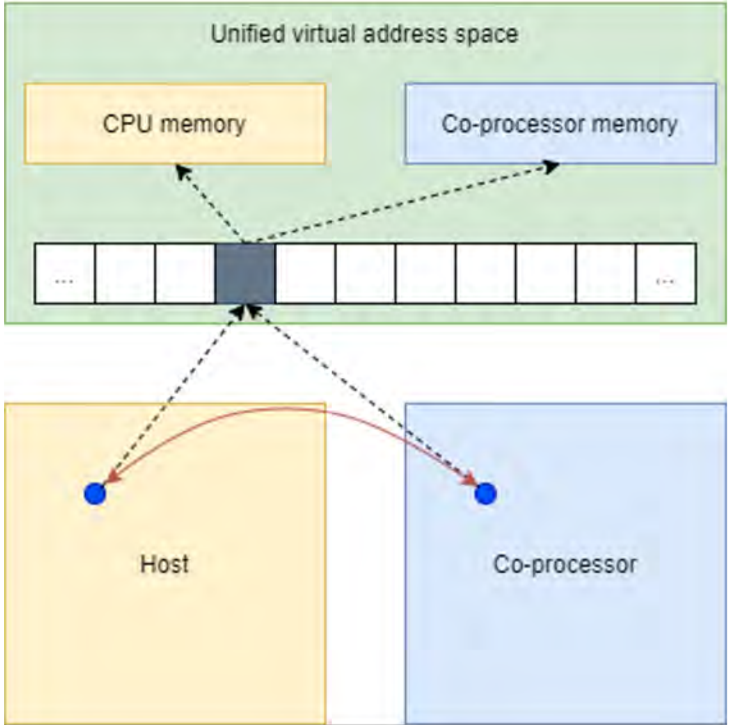
- Data is moved to the device via explicit copy APIs

Here we’re using CUDA as an example

```
float *h_a = { … }, d_a;
cudaMalloc((void **)&d_a, size);
cudaMemcpy(d_a, h_a, size,
  cudaMemcpyHostToDevice);
vec_add<<<64, 64>>>(a, b, c);
cudaMemcpy(d_a, h_a, size,
  cudaMemcpyDeviceToHost);
```

UNIFIED SHARED MEMORY

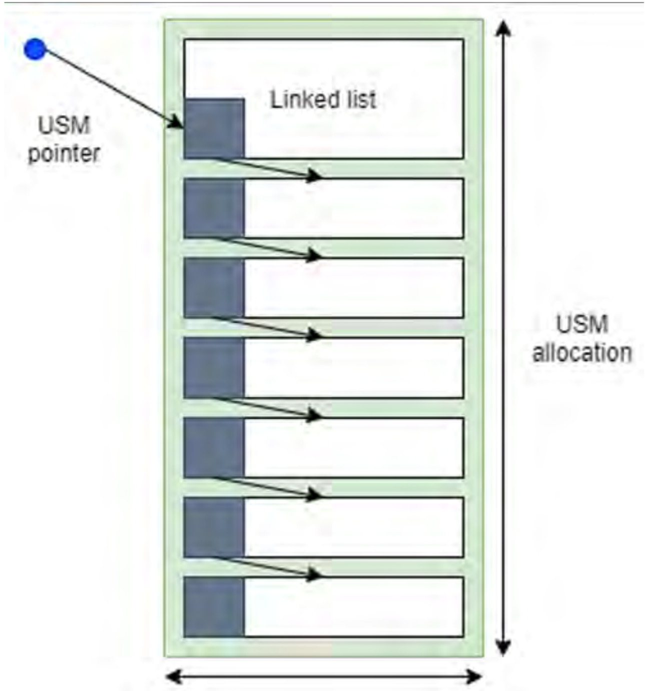
- Unified shared memory provides an alternative pointer-based data management model to the buffer/accessor mode
  - Unified virtual address space (consistent pointers)
  - Pointer-based structures
  - Explicit memory management
  - Shared memory allocations



USM memory allocations return pointers which are consistent between the host application and kernel functions on a device

Representing data between the host and device(s) does not require creating accessors

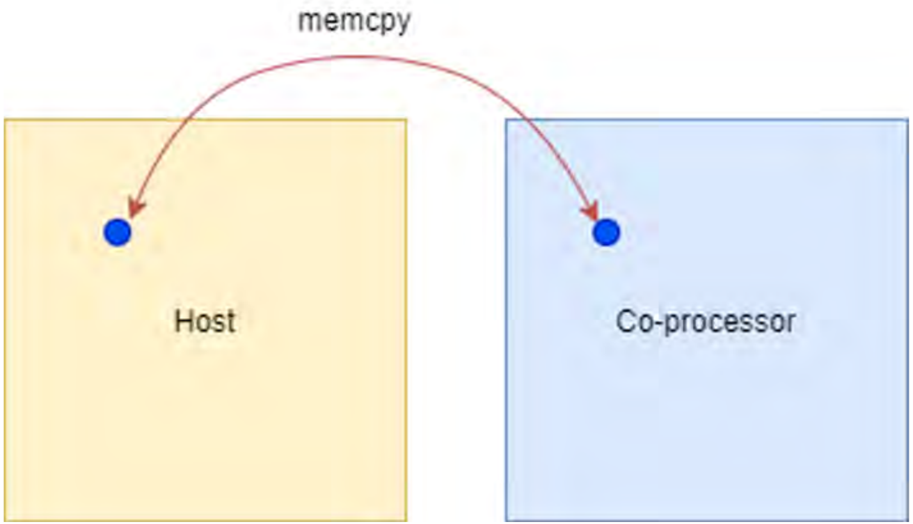
Pointer-based API more familiar to C/C++ developers



Data is moved between the host and device(s) in a span of bytes

Pointers within that region of memory can freely point to any other address in the region

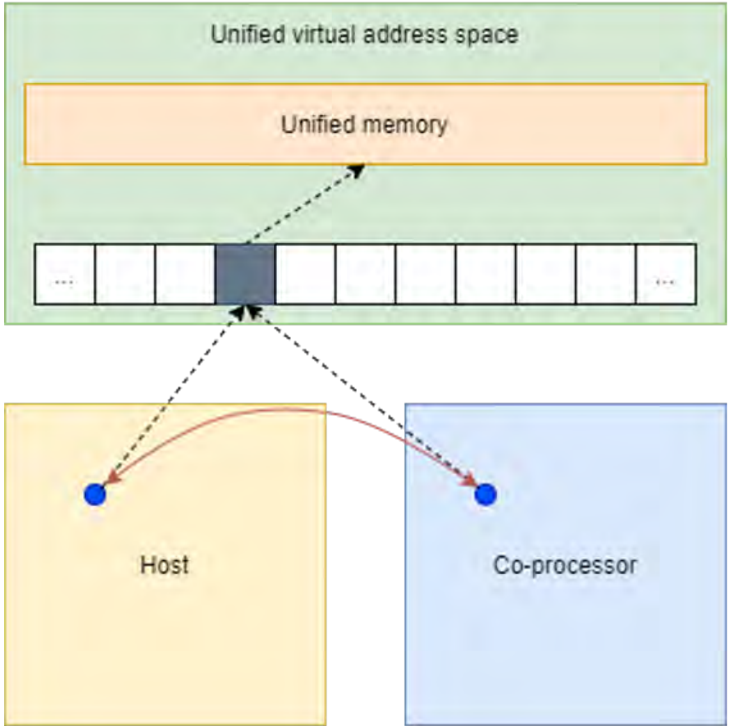
Easier to port existing C/C++ code to SYCL



Memory is allocated and data is moved using explicit routines

SYCL runtime does not perform any data dependency analysis

Dependencies are managed manually



Some platforms will support variants of USM where memory allocations share the same memory region between the host and device(s)

No explicit routines are required to move the data

	Explicit USM (minimum)	Restricted USM (optional)	Concurrent USM (optional)	System USM (optional)
Consistent pointers	✓	✓	✓	✓
Pointer-based structures	✓	✓	✓	✓
Explicit data movement	✓	✓	✓	✓
Shared access	✗	✓	✓	✓
Concurrent access	✗	✗	✓	✓
System allocations (malloc/new)	✗	✗	✗	✓

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

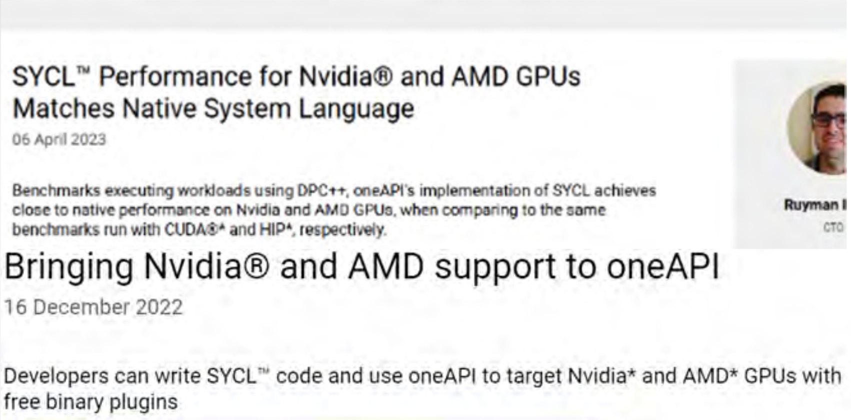
Starting from CUDA

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

SYCL NEWS, ECOSYSTEM, RESEARCH 2023/04/18



SYCL™ Performance for Nvidia® and AMD GPUs Matches Native System Language

06 April 2023

Benchmarks executing workloads using DPC++, 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 HIP*, respectively.

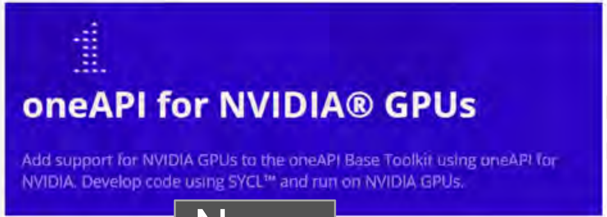
Ruyman I

CTO

Bringing Nvidia® and AMD support to oneAPI

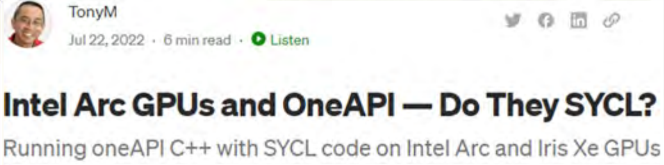
16 December 2022

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



oneAPI for NVIDIA® GPUs

Add support for NVIDIA GPUs to the oneAPI Base Toolkit using oneAPI for NVIDIA. Develop code using SYCL™ and run on NVIDIA GPUs.

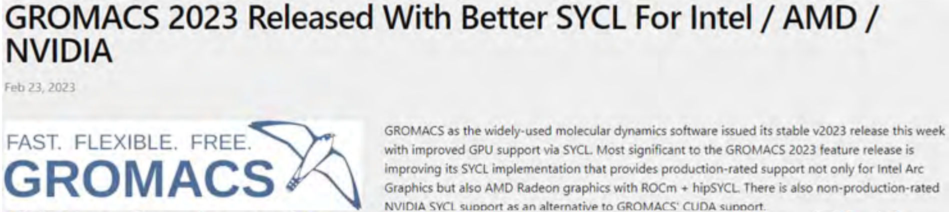


TonyM

Jul 22, 2022 · 6 min read · Listen

Intel Arc GPUs and OneAPI — Do They SYCL?

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



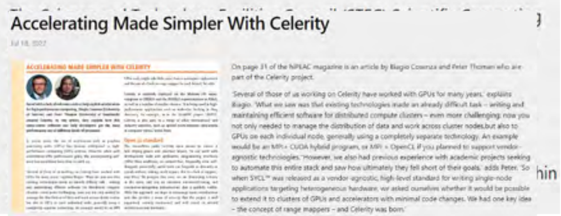
GROMACS 2023 Released With Better SYCL For Intel / AMD / NVIDIA

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 + hipSYCL. There is also non-production-rated NVIDIA SYCL support as an alternative to GROMACS' CUDA support.

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

Jan 5, 2023



Accelerating Made Simpler With Celerity

Jul 18, 2022

On page 31 of the HiPEAC magazine is an article by Biagio Cosenza and Peter Thoman who are part of the Celerity project.

'Several of those of us working on Celerity have worked with GPUs for many years,' explains Biagio. 'What we saw was that existing technologies made an already difficult task – writing and maintaining efficient software for distributed compute clusters – even more challenging: now you not only needed to manage the distribution of data and work across cluster nodes,but also to GPUs on each individual node, generally using a completely separate technology. An example would be an MPI + CUDA hybrid program, or MPI + OpenCL if you planned to support vendor-agnostic technologies.' However, we also had previous experience with academic projects seeking to automate this entire stack and saw how ultimately they fell short of their goals,' adds Peter. 'So when SYCL™ was released as a vendor-agnostic, high-level standard for writing single-node applications targeting heterogeneous hardware, we asked ourselves whether it would be possible to extend it to clusters of GPUs and accelerators with minimal code changes. We had one key idea – the concept of range mappers – and Celerity was born.'

New

STREAM HPC

Argonne NATIONAL LABORATORY

XILINX

New

QUALCOMM

University of BRISTOL

AMD

arm

SYCL™

Working Group Members

New

MOBILEYE

universität innsbruck

SAMSUNG

New

codeplay

intel

New

HUAWEI

TOHOKU UNIVERSITY

HIPPOCRATICA CIVITAS STUDIUM SALERNI

Argonne NATIONAL LABORATORY

UNIFIED ACCELERATION FOUNDATION (UXL)

MISSION

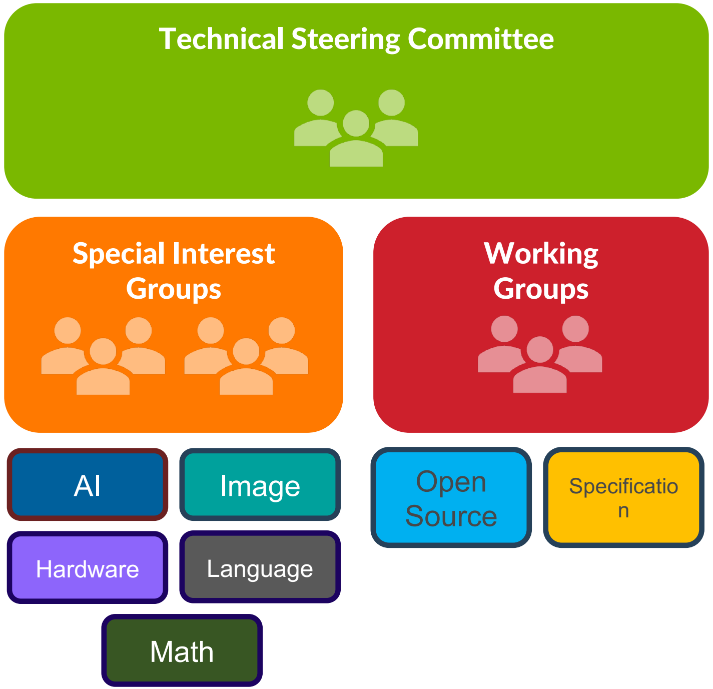
- Build a multi-architecture multi-vendor programming ecosystem for all accelerators
- Unify the heterogeneous compute ecosystem around open standards
- Focus Areas: AI, HPC, Edge AI, Edge Compute
- Open source collaboration

ROLE OF THE FOUNDATION

- Host a specification that defines standard interfaces for programming with common operations needed by developers
- Host code projects that can be used by software developers, and extended to support new processor targets
- Building awareness and a community around the foundation
- Driving the roadmap for the specification and projects
- Bringing a broad set of contributions to the projects

UNIFIED ACCELERATION FOUNDATION (UXL)

UXL Foundation Structure



Technical Steering Committee

Special Interest Groups

- AI
- Image
- Hardware
- Language
- Math

Working Groups

- Open Source
- Specification

UXL FOUNDATION BENEFITS

For Software Developers

- Single code base to maintain
- Save time to market
- Save money from developing across multiple architectures
- Develop with open standards for accelerator computing
- Standards and industry defined libraries

For Processor Developers

- Adopt an open standard with existing open-source implementations
- Enable an existing ecosystem of software and educational resources
- Leverage an existing tested and optimized toolchain

Free and based on open standards

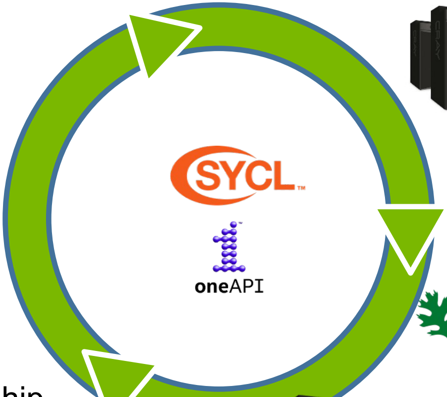
SYCL ENABLES SUPERCOMPUTERS

NERSC National Energy Research Scientific Computing Center

"this work supports the productivity of scientific application developers and users through performance portability of applications between Aurora and Perlmutter."

BERKELEY LAB

NVIDIA GPUs



Argonne NATIONAL LABORATORY

Intel GPUs

OAK RIDGE National Laboratory

AMD GPUs

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

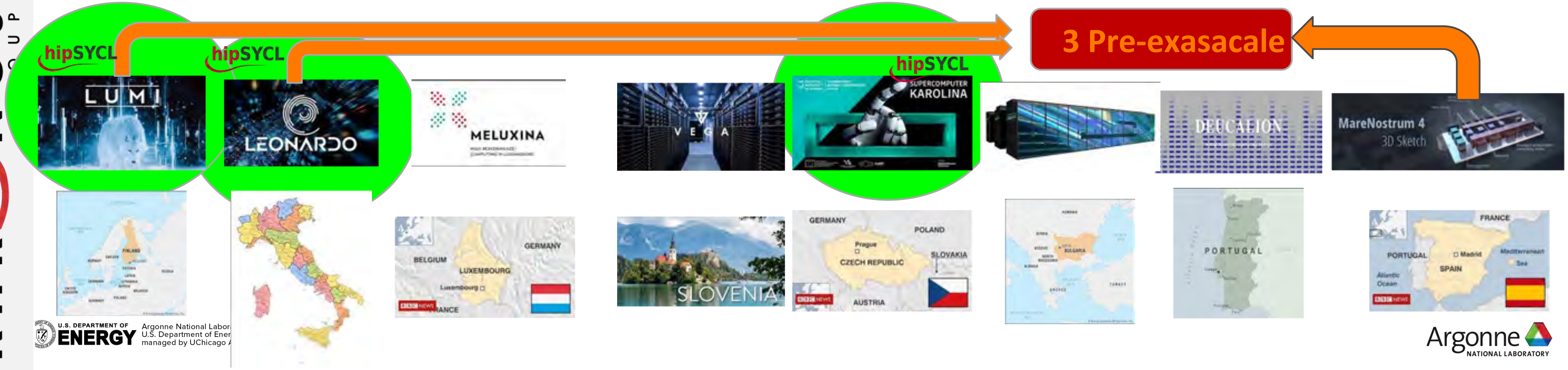
GROMACS
fast, flexible & free

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



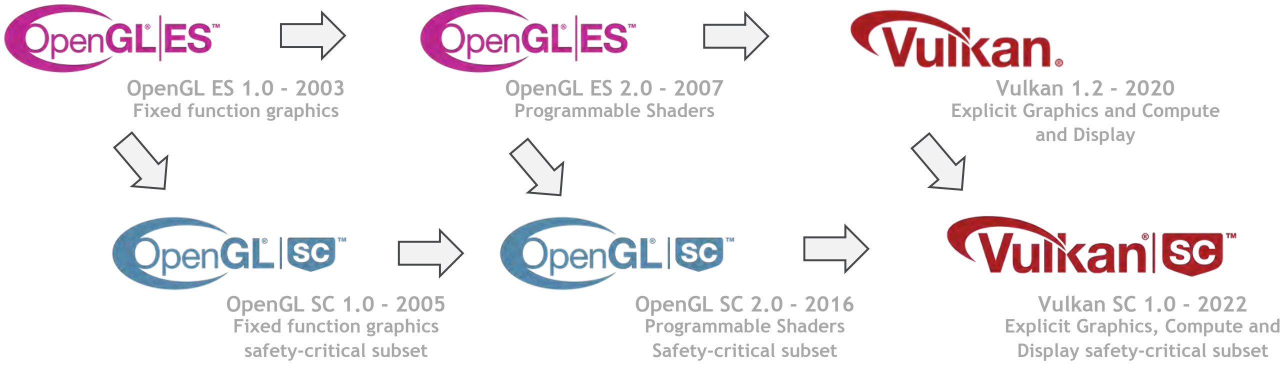
3 Pre-exasacale

U.S. DEPARTMENT OF ENERGY

Argonne National Labor
U.S. Department of Ener
managed by UChicago A

Argonne NATIONAL LABORATORY

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

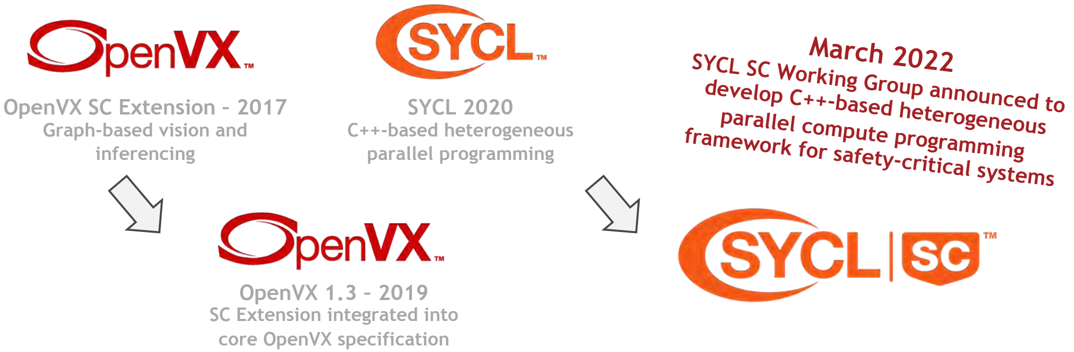
OpenGL SC 2.0 - 2016
Programmable Shaders Safety-critical subset

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



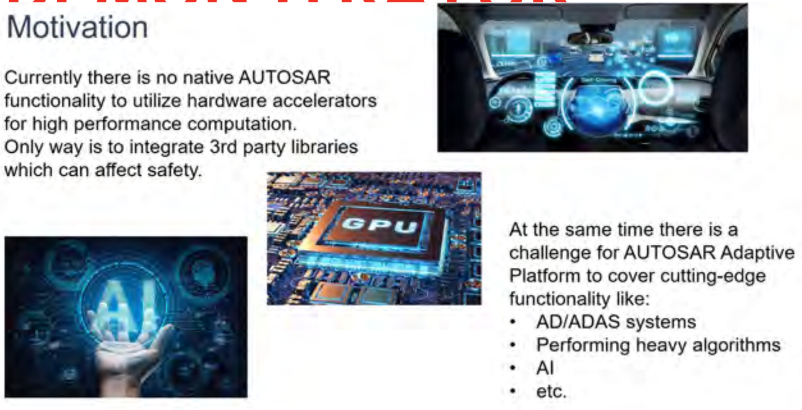
OpenVX SC Extension - 2017
Graph-based vision and inferencing

SYCL 2020
C++-based heterogeneous parallel programming

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

OpenVX 1.3 - 2019
SC Extension integrated into core OpenVX specification

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.

Thank you to AUTOSAR and Intellias

intellias
Intelligent Software Engineering

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.

ISO 26262

THURSDAY OCTOBER 5TH, 2023
CPPCON 2023
AURORA, COLORADO

PART II: HOW ARE C++ & SYCL ADDRESSING SAFETY AT THE COMMITTEE LEVEL?

NEVIN “:-)” LIBER
Computer Scientist
nliber@anl.gov

WHO AM I?

Nevin ":-)" Liber

- Argonne National Laboratory
  - Computer Scientist
    - Argonne Leadership Computing Facility
  - C++, SYCL, Kokkos
  - Aurora
  - WG21 - ISO C++ Committee
    - Vice Chair, Library Evolution Working Group Incubator (LEWGI / SG18)
    - Admin Chair
  - INCITS/C++ - US C++ Committee
    - Vice Chair
  - Khronos SYCL Committee Member

COMMITTEES

- Every person on each of these committees wants to make a better language
  - Even if no two of us can agree on what that is!

- It is Consensus by Committee
  - Not Design by Committee

- I don’t speak for the Committee
  - No one does
  - Polls

SAFETY & SECURITY

- What is *safety*?
  - Limiting the (accidental) damage to a *system* caused by bugs
  - Prefer *prevention* (compile-time) over *detection* (run-time)

- What is *security*?
  - Mitigating *deliberate* attacks against vulnerable parts of a *system*

- Absolute measures (“Is it *safe*?” “Is it *secure*?”) very hard to attain
- Relative measures (“Is it *safer*?” “Is it secure *against attack X*?”) easier to attain

SAFETY

- It is a *tradeoff*…
  - Performance
  - Correctness
    - *This is a bitter pill to swallow*

- Recovery
  - Getting the system back into a known state
  - Halting the system not always appropriate

UNDEFINED BEHAVIOR

- The only C++ knob for “Here Be Dragons”

- Code dependent on UB is *always* a bug
  - Sometimes possible to write & use tooling to detect this kind of bug

- Everything else is *<u>defined</u>* behavior
  - Code dependent on defined behavior may also be a bug
  - Hard to write & use (no-false-positive) tooling to detect this kind of bug

WRAPPING INTEGER MATH

- `unsigned` / `atomic` integer math wraps with respect to addition and subtraction
  - Hardware already does this
  - Not UB
  - Straightforward, reasonable definition
  - Except…

WRAPPING INTEGER MATH

- `unsigned` / `atomic` integer math wraps with respect to addition and subtraction
  - Hardware already does this
  - Not UB
  - Straightforward, reasonable definition
  - Except…

  - 90+% of the time that is a bug
    - Cannot write a no-false-positive sanitizer to detect it
      - Cannot tell the difference between accidental wrapping and deliberate wrapping

P0122R0 ARRAY_VIEW: BOUNDS-SAFE VIEWS FOR SEQUENCES OF OBJECTS

2015

- Precursor to `span` & `mdspan`
- “Any failure to meet `array_view`’s bounds-safety constraints will result in a call to `std::terminate()` (a “fail-fast” approach to safety). This is a critical aspect of `array_view`’s design, and allows users to rely on the guarantee that as long as a sequence is accessed via a correctly initialized `array_view`, then its bounds cannot be overrun.”
- Is this safe?

P0122R0 ARRAY_VIEW: BOUNDS-SAFE VIEWS FOR SEQUENCES OF OBJECTS

2015

- Precursor to `span` & `mdspan`
- "Any failure to meet `array_view`'s bounds-safety constraints will result in a call to `std::terminate()` (a "fail-fast" approach to safety). This is a critical aspect of `array_view`'s design, and allows users to rely on the guarantee that as long as a sequence is accessed via a *correctly initialized array_view*, then its bounds cannot be overrun."

- Is this safe?
  - Safer? Perhaps. Safe? No.
  - We can't check if either `pointer` or `[pointer, pointer + size)` is valid

P2687 DESIGN ALTERNATIVES FOR TYPE-AND-RESOURCE SAFE C++

2022

- It’s easy to break the type system
  - unions, dangling pointers, range errors, casting, etc.
- Safety violations
  - Logic errors, resource leaks, concurrency errors, memory corruption, type errors, overflows, unanticipated conversions, timing errors, termination errors
- Static guarantees based on the C++ Core Guidelines
  - Static analysis
  - Library components as alternatives to error-prone constructs (casts, naked pointers, etc.)
  - Rules to make static analysis more feasible

FUTURE OF C++

Kona 2022 Evening Session

- Lots of opinions
- Brainstorming on the fly ad hoc solutions
  - UB, ambiguous behavior, overflow, GC, concurrency, race conditions, signed/unsigned, etc.
  - Not always identifying the problem
- Not much in the way of principles to apply

- Formation of SG23 - Safety and Security

DIRECTION GROUP

P2759 DG OPINION ON SAFETY FOR ISO C++

Basic Tenets

- Do not radically break backwards compatibility
- Do not deliver safety at the cost of an inability to express the abstractions that are currently at the core of C++ strengths
- Do not leave us with a “safe” subset of C that eliminates C++’s productivity advantages
- Do not deliver a purely run-time model that imposes overheads that eliminate C++’s strengths in the area of performance
- Do not imply that there is exactly one form of “safety” that must be adopted by all
- Do not promise to deliver complete guaranteed type-and-resource safety for all uses
- Do offer paths to gradual and partial adoption
- Do not imply a freeze on further development of C++ in other directions
- Do not imply that equivalent-looking code written in different environments will have different semantics

P2759 DG OPINION ON SAFETY FOR ISO C++

Profiles

- Collection of restrictions, requirements, related semantics that define a safety property to be enforced
  - Specifically, not a subset of C++
    - E.g., a range-safety profile can’t ban unchecked subscripting, but instead provides runtime checked alternative
  - Do not change semantics of a valid program
    - With the exception of UB <==> well-defined behavior
- Open (hard) questions: how do different profiles in the same code base work together?

LIBRARY

MDSPAN

C++23

- mdspan is a *non-owning* multidimensional array view

```
template<class ElementType,
         class Extents,
         class LayoutPolicy = layout_right,
         class AccessorPolicy = default_accessor<ElementType>>
struct mdspan {
    template<class... OtherIndexTypes>
    explicit constexpr mdspan(data_handle_type p, OtherIndexTypes... exts);
    // ...
    template<class... OtherIndexTypes>
    constexpr reference operator[](OtherIndexTypes... indices) const;
}
```

MDSPAN

- Abstracts away tricky arithmetic
- More declarative
- `AccessorPolicy` could do bounds checking

- Still has to be initialized correctly

- P1684 `mdarray`: An Owning Multidimensional Array Analog of `mdspan`

P2821 SPAN.AT()

LEWG Electronic Polling

- Motivation: Safety, Consistency & Public Relations
  - “This new method is safe in the sense that it has defined behavior instead of undefined behavior. Further, the defined behavior is one that can be caught in the code by catching the exception.”

- Is this safe?

P2821 SPAN.AT()

LEWG Electronic Polling

- Motivation: Safety, Consistency & Public Relations
  - “This new method is safe in the sense that it has defined behavior instead of undefined behavior. Further, the defined behavior is one that can be caught in the code by catching the exception.”

- Is this safe?
  - Safer? Perhaps (if folks call `at()` instead of `[]`). Safe? No.
  - Still suffers from not knowing if `span` is correctly initialized
  - `at()` may or may not detect a bug, but if it does, `throwing` doesn’t (necessarily) fix it (put the program back into a known state)

PAPERS BY THIS AUTHOR

- P2730 Variable Scope (closed after P2740, P2742, P2750 discussions)
- P2724 Constant Dangling (closed after P2740, P2742, P2750 discussions)
- P2740 Simpler Implicit Dangling Resolution (4/6/9 Consensus Against)
- P2742 Indirect Dangling Identification (4/6/9 Consensus Against)
- P2750 C Dangling Reduction (4/6/9 Consensus Against)
- P2821 span.at()
- P2878 Reference Checking (R2 5/12/18 Consensus Against; now on R5)
- P2951 Shadowing is Good for Safety (not yet discussed)
- P2955 Safer Range Access (not yet discussed)

P2951 SHADOWING IS GOOD FOR SAFETY

SG23, EWG, LEWG

- Remove names
  - It would be beneficial if programmers could shadow a variable with void initialization instead of having to resort to a tag class
- Reinitialization
  - It would be beneficial if programmers could initialize shadowed variables with the variable that is being shadowed
- Same level shadowing
  - It would be beneficial if programmers could shadow variables without having to involve a child scope
- Conditional casting
  - All of the previous requests have either been hiding a variable altogether or replacing it with an unconditionally casting. It would be beneficial if programmers had a mechanism for conditional casting
- The checked range based for loop
  - This final request is very similar to the second request in example. Minimize the invalidation errors associated with range based for loop by limiting the usage of the instance being iterated over to const access only

P2955 SAFER RANGE ACCESS

Targeting SG23 & LEWGI

- `[]`, `front()`, `back()` not required to check for out of range errors
- They all return references which can lead to dangling and reference invalidation
  - Value semantics would be safer

- `[]`, `at()`, `front()`, `back()` get separate getter and setter functions that take and return values, possibly via `optional`

LANGUAGE

P2723 ZERO-INITIALIZE OBJECTS OF AUTOMATIC STORAGE DURATION

- Initialize all uninitialized “stack” (automatic) variables, including uninitialized member variables and padding bytes, with 0.
  - Now well-defined, not undefined, to read them
    - Developers may rely on this
      - May be intentionally using 0
      - May be a bug
  - Conforming compilers cannot diagnose anything
  - Minor performance hit
    - *Is it minor on GPUs?*

P2795 ERRONEOUS BEHAVIOR FOR UNINITIALIZED READS

- Reading an uninitialized variable is erroneous
  - Definitely a bug
  - Conforming compilers still have to accept it (which is not true about UB)
    - Stack variables are fully initialized (its value can’t change until assigned)
    - Can reject in non-conforming modes
  - Principles on applying it to current UB

- P2973 Erroneous behaviour for missing return from assignment
  - Add implicit `return *this;`

P2900 CONTRACTS WORKING PAPER

- Language support for checking preconditions, postconditions and assertions
  - More knobs for dealing with (library) undefined behavior
  - A principled approach

- Contract Checking Annotation (CCA)
  - Semantics when CCA predicates are evaluated
    - Ignore - unchecked
    - Enforce - terminate if violation detected
    - Observe - continue execution (still UB if violation detected)

SYCL

SYCL::MALLOC(0)

https://github.com/KhronosGroup/SYCL-Docs/pull/356

- C `malloc(0)` - either returns a null pointer value or returns an allocated non-dereferenceable pointer
  - If it returns a null pointer value, sufficient but not necessary to call `free()`
  - If it allocates, necessary to call `free()`

- Is it safer if we pick a specific behavior for `sycl::malloc(0)`?

- Ultimately we decided to mimic the C behavior

SYCL::COMPLEX

Proposed

- Modelled on `std::complex`
  - Initializes its members when default constructed
    - No UB here
  - Q: should we add a method for an uninitialized complex number?
    - Performance vs. Safety

TYPE PUNNING SYCL::COMPLEX TO STD::COMPLEX

```
template<typename T>
void bar(std::complex<T>& stc) { /* ... */ }

template<typename T>
void baz(sycl::complex<T> syc) {
    bar(reinterpret_cast<std::complex<T>&>(syc));
}
```

- “This works because both types have the same in-memory layout”

TYPE PUNNING SYCL::COMPLEX TO STD::COMPLEX

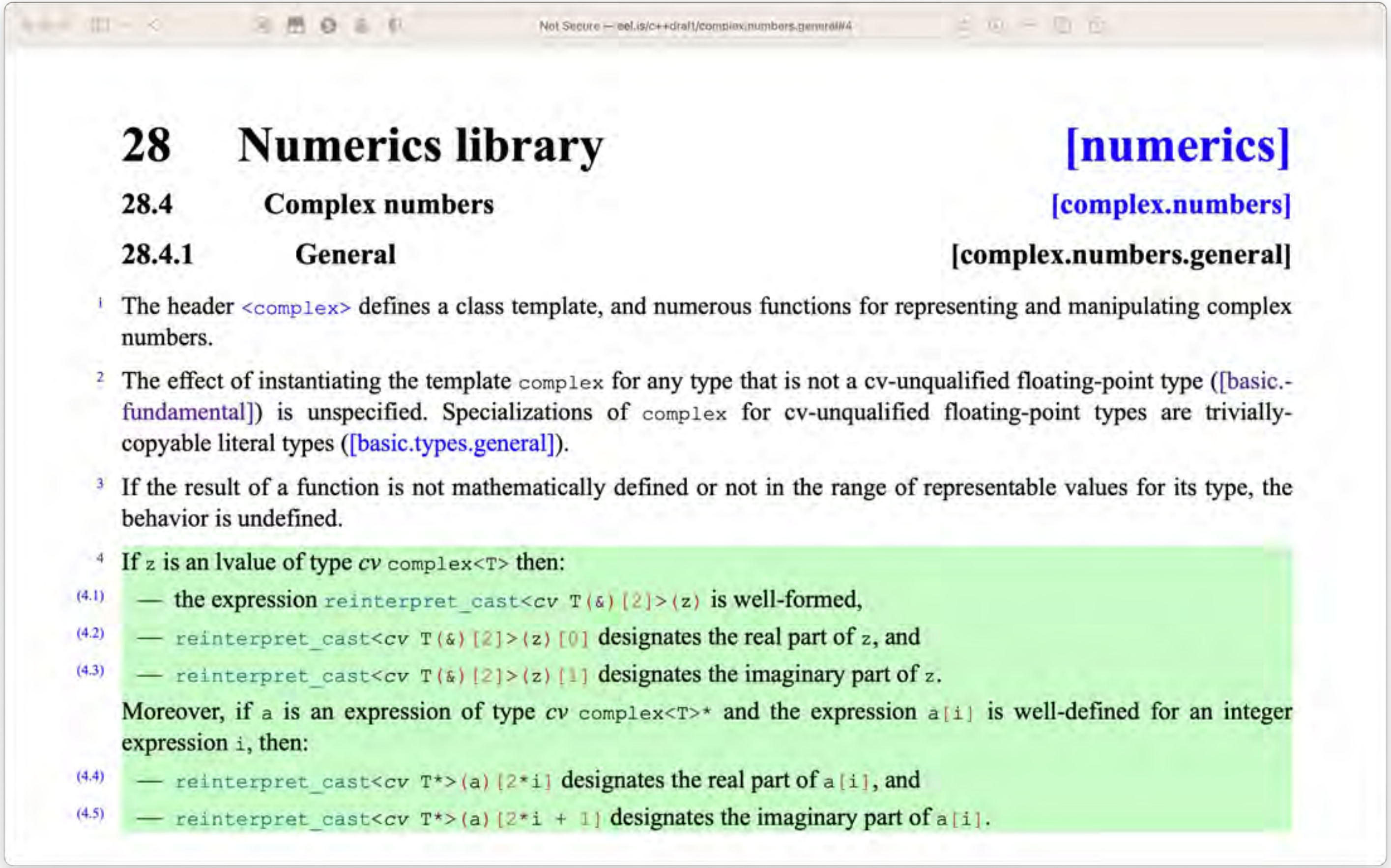
```
template<typename T>
void bar(std::complex<T>& stc) { /* ... */ }

template<typename T>
void baz(sycl::complex<T> syc) {
    ~~bar(reinterpret_cast<std::complex<T>&>(syc));~~
}
```

- ~~“This works because both types have the same in-memory layout”~~
  - ***Undefined Behavior!***

TYPE PUNNING & STRICT ALIASING

C++23 [basic.lval]p11

- Type punning via `reinterpret_cast` or a `union` is ***undefined behavior*** if a type is not *similar* to:
  - The dynamic type of the object
  - A type that is the signed or unsigned type corresponding to the object's dynamic type
  - A `char`, `unsigned char` or `std::byte` type
- Compiler assumes objects of dissimilar types are not aliased (for optimizations)
  - If non-empty, do not occupy the same memory
- Special dispensation for punning `std::complex`<T> and `T[2]` (`_Complex` harmony)
- What is the Strict Aliasing Rule and Why do we care? — *Shafik Yaghmour*
  - https://gist.github.com/shafik



28 Numerics library [numerics]

28.4 Complex numbers [complex.numbers]

28.4.1 General [complex.numbers.general]

1 The header `<complex>` defines a class template, and numerous functions for representing and manipulating complex numbers.

2 The effect of instantiating the template `complex` for any type that is not a cv-unqualified floating-point type ([basic.fundamental]) is unspecified. Specializations of `complex` for cv-unqualified floating-point types are trivially-copyable literal types ([basic.types.general]).

3 If the result of a function is not mathematically defined or not in the range of representable values for its type, the behavior is undefined.

4 If `z` is an lvalue of type *cv* `complex<T>` then:

(4.1) — the expression `reinterpret_cast<`*cv* `T(&)[2]>(z)` is well-formed,

(4.2) — `reinterpret_cast<`*cv* `T(&)[2]>(z)[0]` designates the real part of `z`, and

(4.3) — `reinterpret_cast<`*cv* `T(&)[2]>(z)[1]` designates the imaginary part of `z`.

Moreover, if `a` is an expression of type *cv* `complex<T>*` and the expression `a[i]` is well-defined for an integer expression `i`, then:

(4.4) — `reinterpret_cast<`*cv* `T*>(a)[2*i]` designates the real part of `a[i]`, and

(4.5) — `reinterpret_cast<`*cv* `T*>(a)[2*i + 1]` designates the imaginary part of `a[i]`.

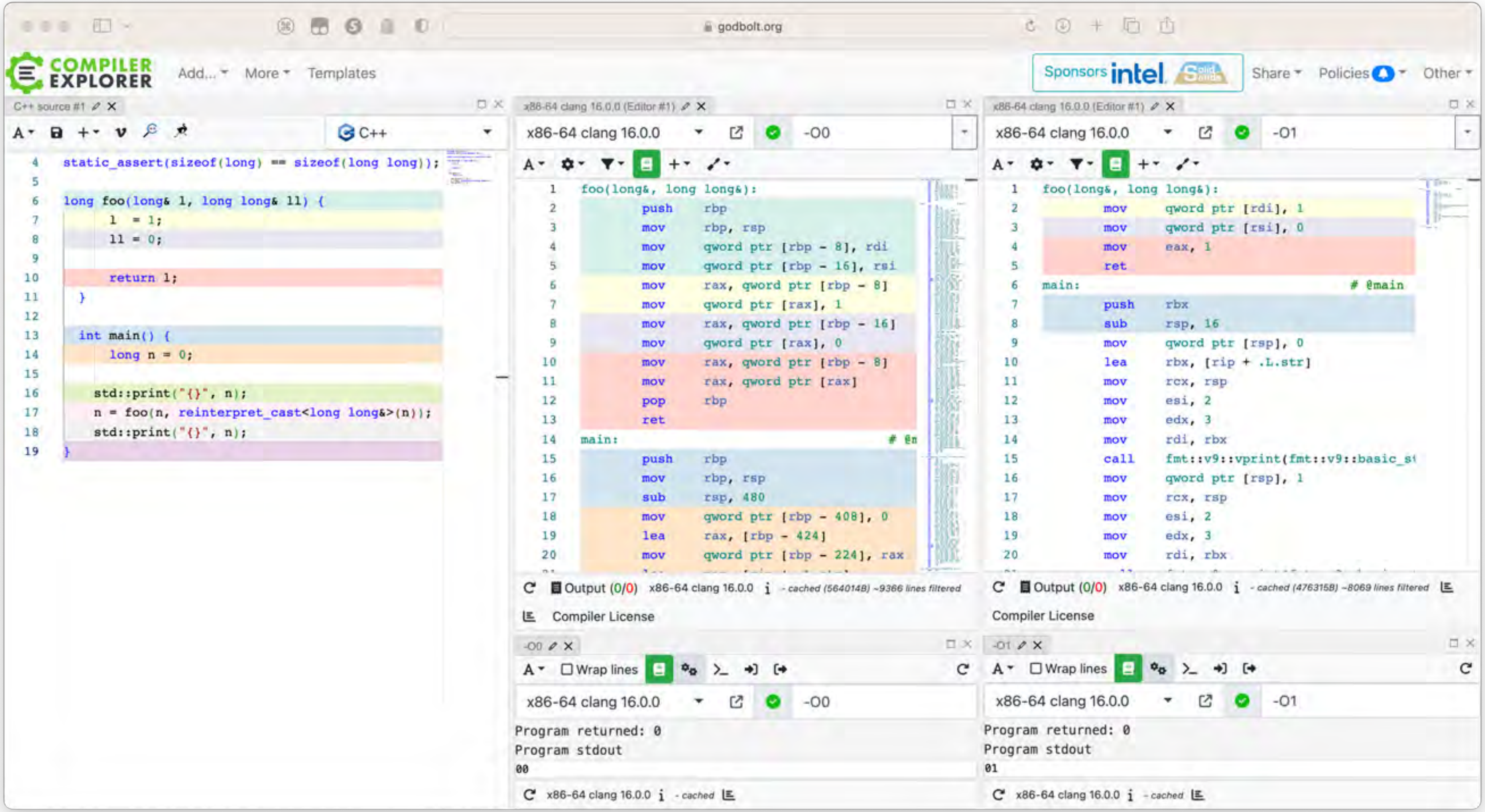
https://eel.is/c++draft/complex.numbers.general#4

TYPE PUNNING & STRICT ALIASING

reinterpret_cast between long and long long

```
static_assert(sizeof(long) == sizeof(long long));

long foo(long& l, long long& ll) {
      l  = 1;
      ll = 0;

      return l;
  }

  int main() {
      long n = 0;

    std::print("{}", n);
    n = foo(n, reinterpret_cast<long long&>(n));
    std::print("{}", n);
}
```

- What is the expected output?



C++ source #1:

```cpp
static_assert(sizeof(long) == sizeof(long long));

long foo(long& l, long long& ll) {
    l  = 1;
    ll = 0;

    return l;
  }

  int main() {
    long n = 0;

   std::print("{}", n);
   n = foo(n, reinterpret_cast<long long&>(n));
   std::print("{}", n);
}
```

x86-64 clang 16.0.0, -O0:

```
foo(long&, long long&):
        push    rbp
        mov     rbp, rsp
        mov     qword ptr [rbp - 8], rdi
        mov     qword ptr [rbp - 16], rsi
        mov     rax, qword ptr [rbp - 8]
        mov     qword ptr [rax], 1
        mov     rax, qword ptr [rbp - 16]
        mov     qword ptr [rax], 0
        mov     rax, qword ptr [rbp - 8]
        mov     rax, qword ptr [rax]
        pop     rbp
        ret
main:                                   # @m
        push    rbp
        mov     rbp, rsp
        sub     rsp, 480
        mov     qword ptr [rbp - 408], 0
        lea     rax, [rbp - 424]
        mov     qword ptr [rbp - 224], rax
```

Output (0/0) x86-64 clang 16.0.0 - cached (5640148) ~9366 lines filtered

x86-64 clang 16.0.0, -O1:

```
foo(long&, long long&):
        mov     qword ptr [rdi], 1
        mov     qword ptr [rsi], 0
        mov     eax, 1
        ret
main:                                   # @main
        push    rbx
        sub     rsp, 16
        mov     qword ptr [rsp], 0
        lea     rbx, [rip + .L.str]
        mov     rcx, rsp
        mov     esi, 2
        mov     edx, 3
        mov     rdi, rbx
        call    fmt::v9::vprint(fmt::v9::basic_s
        mov     qword ptr [rsp], 1
        mov     rcx, rsp
        mov     esi, 2
        mov     edx, 3
        mov     rdi, rbx
```

Output (0/0) x86-64 clang 16.0.0 - cached (4763158) ~8069 lines filtered

-O0 (x86-64 clang 16.0.0, -O0):

```
Program returned: 0
Program stdout
00
```

x86-64 clang 16.0.0 - cached

-O1 (x86-64 clang 16.0.0, -O1):

```
Program returned: 0
Program stdout
01
```

x86-64 clang 16.0.0 - cached

https://godbolt.org/z/Gnoq7hPf7

COMPLEX

- Recap
  - Type punning between `std::complex<T>` & `T[2]` (& `_Complex T`) is allowed because of special dispensation in the C++ standard

- As part of proposing this
  - GENE & GTENSOR
    - CUDA & ROCm
      - C `_Complex`
      - Required `reinterpret_cast` from `thrust::complex`
        - UB!

BUFFER LIFETIME

- `sycl::buffer` is a sometimes owning collection of data
- `sycl::accessor` is a non-owning view that defines the data depencies

- “The basic rule for the blocking behavior of a buffer destructor is that it blocks if there is some data to write back because a write accessor on it has been created, or if the buffer was constructed with attached host memory and is still in use.”
  - In the case when the buffer destructor blocks
- “When the buffer is destroyed, the destructor will block until all work in queues on the buffer have completed, then copy the contents of the buffer back to the host memory (if required) and then return.”

BUFFER LIFETIME

```
q.submit([&](handler &cgh) {
  buffer<int> b{range{N}}; // Buffer constructed w/o host data (destructor does NOT block)
  accessor a{b, cgh, read_write};
  cgh.parallel_for(N, [=](auto i){ a[i] = /*...*/; });
  /* buffer destroyed, command has dangling accessor */
});
```

- `submit` is an async action
  - Lambda copies accessor a into that action
    - Keeps the accessor alive until the end of the async operation
- After `parallel_for` returns, `buffer` goes out of scope and is destroyed

BUFFER LIFETIME

- What if we move buffer outside of submit scope?

```
buffer<int> b{range{N}};
q.submit([&](handler &cgh) {
  accessor a{b, cgh, read_write};
  cgh.parallel_for(N, [=](auto i){ a[i] = /*...*/; });
});
/* buffer destroyed, command has dangling accessor */
```

- Same problem

BUFFER LIFETIME

- What about non-owning buffer?

```
q.submit([&](handler &cgh) {
  int data[N];
  buffer b{data, range{N}};
  accessor a{b, cgh, read_write};
  cgh.parallel_for(N, [=](auto i){ a[i] = /*...*/; });
  /* buffer destroyed here - Destructor blocks until commands accessing buffer complete */
});
```

- Has a “command” been created at the point where the destructor runs?
- Is command created by “parallel_for”?
  - If so, code above has defined semantics
- Is command created when “submit” returns?
  - If so, behavior seems unclear. Nothing in spec says this is UB.

BUFFER LIFETIME

- What is the “safer” solution?
  - Accessors and buffers are distinct
    - Simpler semantics that are easier to reason about
    - But can lead to UB
  - Accessors can sometimes keep buffers alive
    - “Just works” in more circumstances
    - Harder for developers to reason about those circumstances

PART II SUMMARY

SUMMARY

- The committees are made up of C++ (and SYCL) experts, not (as a whole) safety experts
- Many of the solutions proposed are ad hoc
- We (desperately) need principles we can apply

- SYCL
  - Subset of customers that have specific and critical requirements

- Formation of SYCL SC (SYCL for Safety Critical Systems)
  - They have the SYCL expertise, safety expertise and safety principles to apply

Verena Beckham

VP of Safety Engineering
at
codeplay

Background:
LLVM compiler engineer,
backend

Chair of SYCL SC Working Group
in Khronos

WHAT IS “SAFETY-CRITICAL”?

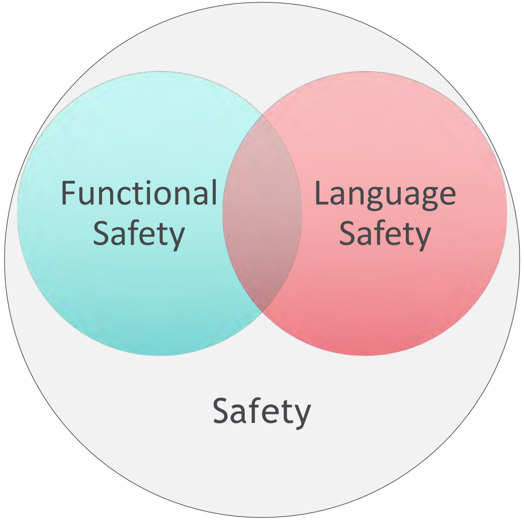
- A system is *Safety-Critical* if its failure could result in harm/death of people
- SC industries: automotive, avionics, medical, rail, atomic
- Often certified according to standards
  - Automotive: ISO 26262
  - Avionics: DO-178C
  - Medical: IEC 62304
- Standards define safety levels: ASIL A-B / DAL A-D / Class A-C
- Require *Functional Safety*
  - Absence of unreasonable risk caused by malfunction

=> Risk has been analyzed, mitigated to a reasonable level, proven

ACHIEVING FUNCTIONAL SAFETY

For software:

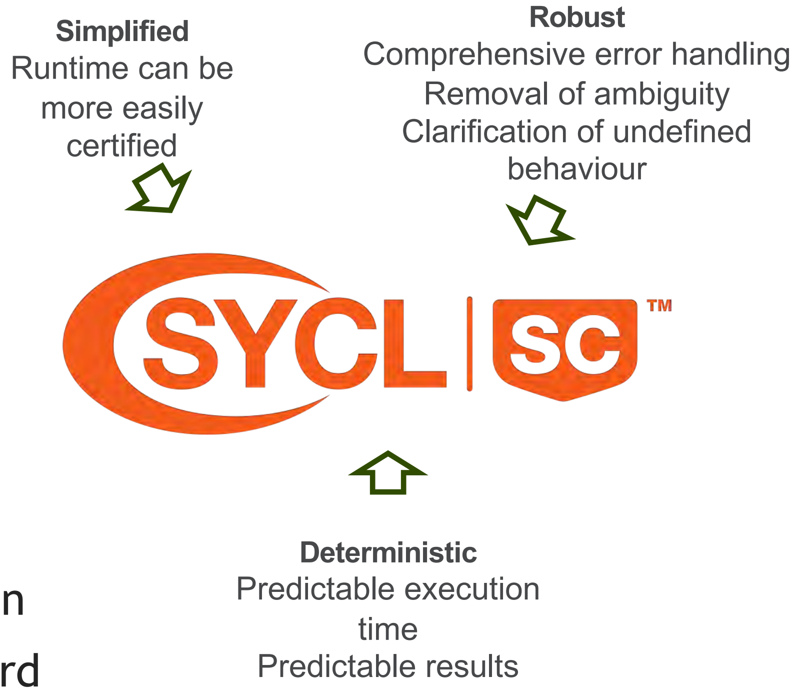
- Follow defined processes
  - Requirements management
  - Design of architecture & units
  - Formalised code review
  - Failure analysis
- Avoid bugs
  - Avoid error-prone features
  - Adhere to coding guidelines / best-practice
  - Rigorous testing & coverage checking
- Ensure errors are handled gracefully (bugs always remain!)
- Ensure deterministic timing
  - Work gets done on time, predictably
  - Errors are detected & mitigated within deadline



SYCL SC

New Khronos Working Group
Created March ‘23

- **Why?**
  - SC industries increasingly require *acceleration* of software, due to
    - Rising popularity of **AI** algorithms
    - Proliferation of **heterogeneous** computing
    - Increasing demand for **performance**
- **What?**
  - Based on SYCL 2020
  - Modifications to ease safety-certification
    - Of the implementation of the standard
    - Of the SYCL application



Simplified
Runtime can be more easily certified

Robust
Comprehensive error handling
Removal of ambiguity
Clarification of undefined behaviour

Deterministic
Predictable execution time
Predictable results

WHY C++ FOR SC?

- C++ is constantly adding features that help
- Safety features in the language are only one concern
- Other concerns:
  - To achieve productivity, you need **development tools**
  - To prove correctness and timing you need **analysis tools**
    - Static analysis
    - Performance analysis
  - To achieve performance, you need optimized **libraries**
  - To have confidence in your software you rely on **language maturity**
  - To be able to write robust code you require **guidelines**
  - To write good code you need **experienced developers**

→ The C++ ecosystem is very attractive to the SC industries

SOME CONCERNS OF SYCL SC

- C++ version requirement
  - Are certified tools available for the language?
  - Are guidelines available?
- Exception handling
  - Timing not predictable (as implemented in standard compilers)
  - SYCL error handling relies on them
- Dynamic memory allocation
  - Timing not predictable (in standard memory allocation)
    - Finding a chunk of memory may take time
    - Making a system call is unpredictable
  - Success not guaranteed
- buffer/accessor vs. USM
  - Example of Memory Safety vs. Control
    - Safety features are abstractions
    - Abstractions reduce control, timing not as predictable

Standard library relies on both!

SYCL relies on the standard library!

WHAT ABOUT DYNAMIC MEMORY?

“Cannot have dynamic memory allocation in the application.”

“Then you need all memory to be known statically! That’s pretty restrictive.”

An application is likely to have an initialization phase, in which dynamic memory allocation is fine.

“All object allocations need to be known ahead-of-time.”

“Wait, what about control flow?”

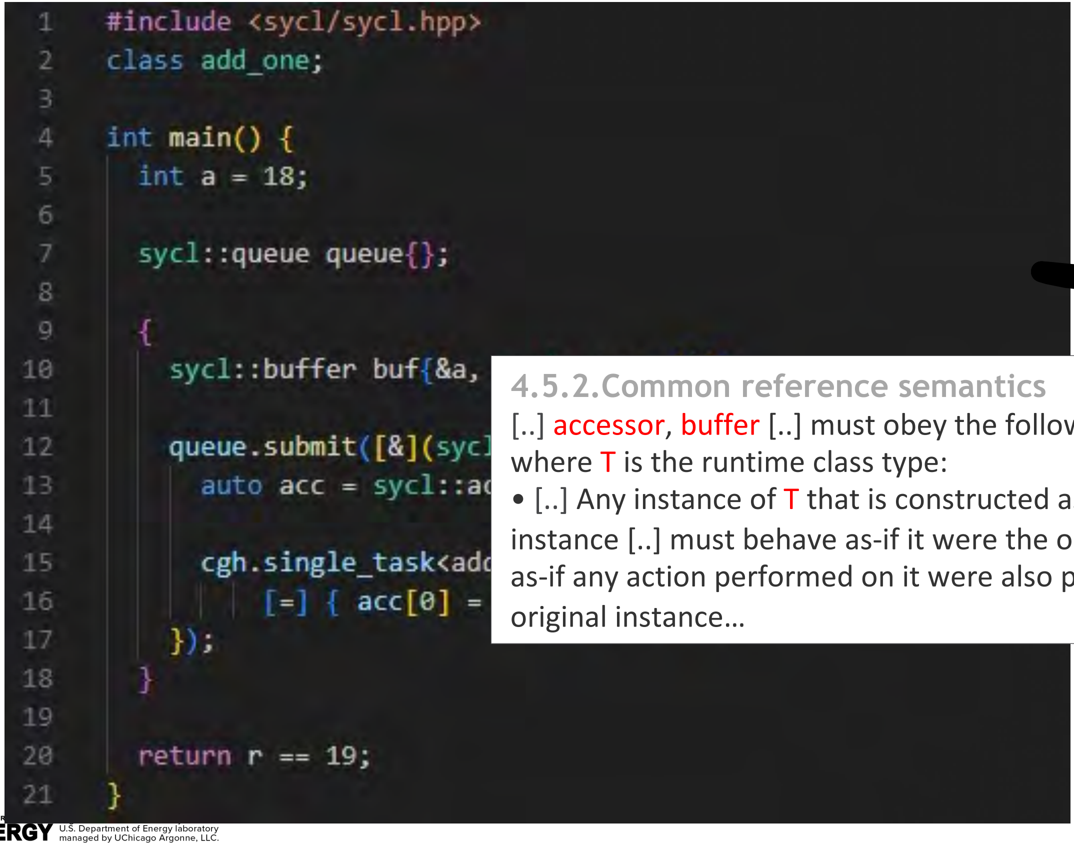
Only the upper bound of required memory needs to be known.

“All heap objects need to be allocated at initialization.”

Can allocate pools at initialization, then allocate objects from the pools.

WHERE IS THE DYNAMIC MEMORY??

“Just get rid of new/delete”



```
#include <sycl/sycl.hpp>
class add_one;

int main() {
  int a = 18;

  sycl::queue queue{};

  {
    sycl::buffer buf{&a,

    queue.submit([&](sycl
      auto acc = sycl::ac

      cgh.single_task<add
          [=] { acc[0] =
    });
  }

  return r == 19;
}
```

4.5.2.Common reference semantics

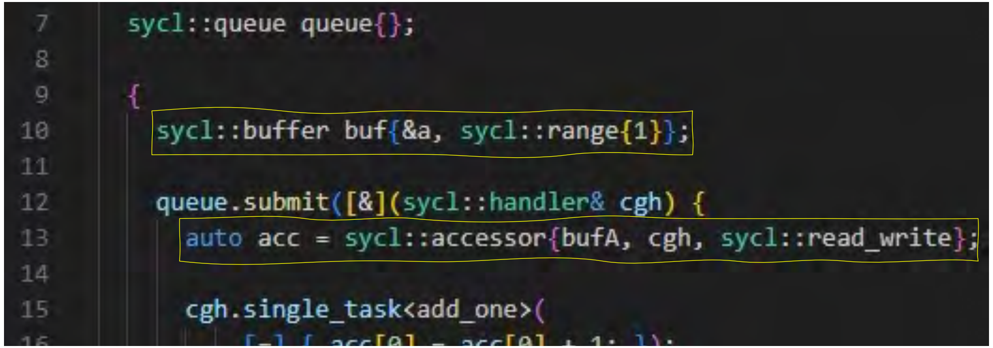
[..] accessor, buffer [..] must obey the following statements, where T is the runtime class type:

- [..] Any instance of T that is constructed as a copy of another instance [..] must behave as-if it were the original instance and as-if any action performed on it were also performed on the original instance…

[redacted]

Behavior like std::shared_ptr. Typically allocated on the heap by the SYCL runtime.

HERE IS THE DYNAMIC MEMORY



```
sycl::queue queue{};

{
  sycl::buffer buf{&a, sycl::range{1}};

  queue.submit([&](sycl::handler& cgh) {
    auto acc = sycl::accessor{bufA, cgh, sycl::read_write};

    cgh.single_task<add_one>(
        [=] { acc[0] = acc[0] + 1; });
```

There could be dynamic memory in the application AND in the runtime.

How to modify the API
to allow the implementation
to avoid dynamic memory?

GET INVOLVED!

Excited about getting your hands on this?

Are you piqued by the challenge?

SYCL SC™

Find me at CppCon!

Member of Khronos? Join the Working Group!

Not a member? Look out for Advisory Panels!

Visit www.khronos.org/syclsc
Contact sycl_sc-chair@lists.khronos.org
or verena@codeplay.com

FINAL WORDS

Programming Models Must Persist but also be high quality and portable with conformance tests

SYCL 2020 Launched February 2021

SYCL SC WG has started with many automotive, space agency interests

SYCL SC enables safety critical, functional safety based on ISO C++ and Khronos SYCL

SYCL user and developer Phenomenal Growth

Easy to build SYCL on any device

SYCL thriving community is our most important asset

Future SYCL: Emerging transformative technologies for HPC and Safety

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

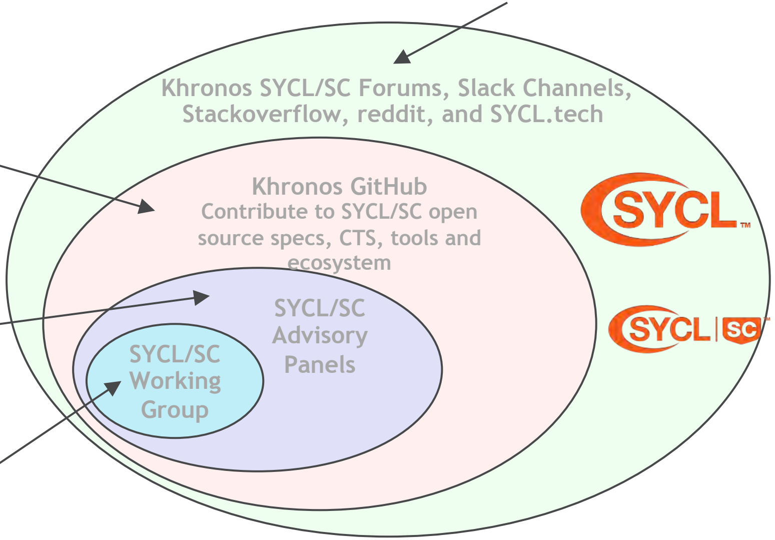
- SYCL/SC working group values industry feedback
  - https://community.khronos.org/c/sycl
  - https://sycl.tech
  - https://www.khronos.org/syclsc
- SYCL Academy
  - https://github.com/codeplaysoftware/syclacademy
- SYCL FAQ
  - https://www.khronos.org/blog/sycl-2020-what-do-you-need-to-know
- SYCL CTS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

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/

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/


Khronos SYCL/SC Forums, Slack Channels, Stackoverflow, reddit, and SYCL.tech
Khronos GitHub
Contribute to SYCL/SC open source specs, CTS, tools and ecosystem
SYCL/SC Advisory Panels
SYCL/SC Working Group
SYCL
SYCL SC

- Portions of this research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy (DOE) Office of Science and the National Nuclear Security Administration
- Portions of this research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357
- Portions of this research also used resources of the National Energy Research Scientific Computing Center (NERSC), a DOE Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract DE-AC02- 05CH11231