Neil Trevett
Vice President Mobile Ecosystem at NVIDIA
President of Khronos and Chair of the OpenCL Working Group
SIGGRAPH, August 2015
Khronos Connects Software to Silicon

Open Consortium creating ROYALTY-FREE, OPEN STANDARD APIs for hardware acceleration

Defining the roadmap for low-level silicon interfaces needed on every platform

Graphics, compute and vision processing

Rigorous specifications AND conformance tests for cross-vendor portability

Acceleration APIs BY the Industry FOR the Industry

Well over a BILLION people use Khronos APIs Every Day...
OpenCL - Portable Heterogeneous Computing

- Portable Heterogeneous programming of diverse compute resources
  - Targeting supercomputers -> embedded systems -> mobile devices
- One code tree can be executed on CPUs, GPUs, DSPs, FPGA and hardware
  - Dynamically interrogate system load and balance work across available processors
- OpenCL = Two APIs and Kernel language
  - C Platform Layer API to query, select and initialize compute devices
  - C Runtime API to build and execute kernels across multiple devices
OpenCL Desktop Usage

• Broad commercial uptake of OpenCL
  - Imaging, video, vision, simulation
  - Adobe, Apple, SONY, Corel, ArcSoft
  - Dassault, Houdini, Mathematica, MAYA...

• “OpenCL” on Sourceforge, Github, Google Code, Bitbucket finds over 2,000 projects
  - OpenCL implementations - Beignet, pocl
  - VLC, X264, FFmpeg, Handbrake
  - GIMP, ImageMagick, IrfanView
  - Hadoop, Memcached
  - WinZip, Crypto++ Etc. Etc.

• Desktop benchmarks use OpenCL
  - PCMark 8 - video chat and edit
  - Basemark CL, CompuBench Desktop

https://www.khronos.org/opencl/resources/opencl-applications-using-opencl
OpenCL 2.1 Provisional in Review

- New OpenCL C++ kernel language based on a subset of C++14
  - Significantly enhanced programmer productivity and code performance
  - OpenCL C still supported to preserve kernel code investment

- Support for the new Khronos SPIR-V intermediate language in core
  - SPIR-V now used by both OpenCL 2.1 and the new Vulkan graphics API

- Runs on any OpenCL 2.0-capable hardware
  - Only driver update required
OpenCL 2.0 Updates Released at SIGGRAPH

- Updates to OpenCL 2.0 specification
  - Bug fixes, clarifications

- OpenCL C++ Headers for OpenCL 2.0 also released today
  - Enables host side C++ code

- New OpenCL 1.2 implementations!
  - NVIDIA (Desktop)
  - Texas Instruments (Embedded)
  - Marvell (Mobile & Embedded)
OpenCL Implementations

Vendor timelines are first implementation of each spec generation.

- **OpenCL 1.0**
  - AMD 1.0 | May09
  - IBM 1.0 | Jan10
  - Intel 1.0 | May09
  - NVIDIA 1.0 | Feb11
  - ARM 1.0 | May09

- **OpenCL 1.1**
  - AMD 1.1 | Aug10
  - IBM 1.1 | Jun10
  - Intel 1.1 | Mar11
  - NVIDIA 1.1 | Mar11
  - ARM 1.1 | Aug12

- **OpenCL 1.2**
  - AMD 1.2 | May12
  - IBM 1.2 | Aug12
  - Intel 1.2 | May12
  - NVIDIA 1.2 | Oct12
  - ARM 1.2 | May12

- **OpenCL 2.0**
  - AMD 2.0 | Oct13
  - IBM 2.0 | Sep13
  - Intel 2.0 | Sep13
  - NVIDIA 2.0 | Sep13
  - ARM 2.0 | Oct13

- **OpenCL 2.1**
  - AMD 2.1 | Oct14
  - IBM 2.1 | Oct14
  - Intel 2.1 | Oct14
  - NVIDIA 2.1 | Oct14
  - ARM 2.1 | Oct14

- **Desktop**
  - 2.0 | Dec14
  - 1.2 | May15

- **Mobile**
  - 2.0 | Jul14
  - 1.2 | May15

- **Embedded**
  - 2.0 | Dec14
  - 1.2 | May15

- **FPGA**
  - 1.0 | Jul13
  - 1.2 | Aug15

© Copyright Khronos Group 2015 - Page 7
OpenCL on Mobile and Embedded

- Mobile SOCs now beginning to need more than just ‘GPU Compute’
  - Multi-core CPUs, GPUs, DSPs
- OpenCL can provide a single programming framework for all processors on a SOC
  - Even ISPs and specialized hardware blocks with Built-in Kernels for custom HW

OpenCL is shipping today on multiple mobile and embedded processors and cores

Image Courtesy Qualcomm
Embedded Needs Driving OpenCL Roadmap

- OpenCL roadmap discussions focused on mobile and embedded markets
  - Very different needs and constraints to HPC/desktop

**Different Bottlenecks to HPC**
- Mobile GPU have less processing throughput than desktop
- GPU often busy with rendering hi-res screen
- Mobile memory bandwidth is often perf bottleneck

**Always-on vision / sensor processing key use case**
- Must run at very low power levels
- Thermal conditions often limit performance

**Vision Applications Must Not Drop Frames**
- Real-time / guaranteed QOS

**App performance portability is much more critical**
- More diverse architectures in mobile than HPC
- 1000s of different devices

**Many diverse hardware processing blocks**
- DSP, ISP, dedicated hardware

**Dynamic load-balancing**
- Instrumentation/control to route work to different compute resources depending on system loading

**Optimize Energy Efficiency**
- Reduce precision - even integer-only profiles?
- Autonomous frame processing - no host involvement?

**Extended Execution Model**
- Pre-emption, yielding

**Higher-level portability frameworks and engines**
- Built over powerful low-level APIs

**Expand support for Custom Devices**
- Extended pipes and graphs in OpenCL run-time?
OpenCL as Parallel Language Backend

JavaScript binding for initiation of OpenCL C kernels

Language for image processing and computational photography

MulticoreWare open source project on Bitbucket

Single Source C++ Programming for OpenCL

Java language extensions for parallelism

River Trail Language extensions to JavaScript

Compiler directives for Fortran, C and C++

PyOpenCL Python wrapper around OpenCL

Harlan High level language for GPU programming

Approaching 200 languages, frameworks and projects using OpenCL as a compiler target to access vendor optimized, heterogeneous compute runtimes
SPIR-V Transforms the Language Ecosystem

• First multi-API, intermediate language for parallel compute and graphics
  - Native representation for Vulkan shader and OpenCL kernel source languages

• Cross vendor intermediate representation
  - Language front-ends can easily access multiple hardware run-times
  - Acceleration hardware can leverage multiple language front-ends
  - Encourages tools for program analysis and optimization in SPIR form

Multiple Developer Advantages

• Same front-end compiler for multiple platforms
  • Reduces runtime kernel compilation time
  • Don’t have to ship shader/kernel source code
  • Drivers are simpler and more reliable
Driving the SPIR-V Open Source Ecosystem

SPIR-V Provisional spec (V31) updated today!

Khronos will open source these tools and translators

SPIR-V Tools
- SPIR-V Validator
- SPIR-V (Dis)Assembler

GLSL
- Third party kernel and shader Languages

OpenCL C

OpenCL C++

LLVM

LLVM to SPIR-V Bi-directional Translator

SPIR-V
- 32-bit Word Stream
- Extensible and easily parsed
- Retains data object and control flow information for effective code generation and translation

IHV Driver Runtimes

Other Intermediate Forms
# Speakers

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>Company</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neil Trevett</td>
<td>OpenCL Chair</td>
<td>NVIDIA</td>
<td>OpenCL 2.0 C++ Headers</td>
</tr>
<tr>
<td>Lee Howes</td>
<td>Senior Staff Engineer</td>
<td>Qualcomm</td>
<td>OpenCL 2.1 and OpenCL C++ SPIR-V Overview</td>
</tr>
<tr>
<td>Kenneth Benzie</td>
<td>Staff Software Engineer</td>
<td>Codeplay</td>
<td>SPIR-V Tools Ecosystem</td>
</tr>
<tr>
<td>Ralph Potter</td>
<td>Research Engineer</td>
<td>Codeplay</td>
<td>SYCL Overview</td>
</tr>
<tr>
<td>All</td>
<td></td>
<td></td>
<td>Wrap-up and Questions</td>
</tr>
</tbody>
</table>
OpenCL 2.0 Features

- Multiple bug fixes and cleanups
  - Improvements to versioning
- Switched to STL-compatible structures throughout
  - Memory allocation under developer control via typedefs
- Some changes break backward compatibility
  - Releasing as cl2.hpp
- Adds support for core 2.0 features:
  - Shared virtual memory
  - Pipes
  - Device queues
- Simplify trivial OpenCL programs by providing defaults
OpenCL 2.0 SVM

- SVM functionality provided by containers and allocators
  - Allow us of standard STL containers, or STL-compatible containers
  - Forward memory management to OpenCL runtime functionality
- `allocate_pointer`
  - Behaves like STL’s `allocate_shared`
- `allocate_svm`
  - Creates a parameterized allocator and uses that to allocate
- `coarse_svm_vector, fine_svm_vector, atomic_svm_vector`
  - Default to `std::vector` parameterized by an OpenCL allocator to create the matching form of SVM
- All functionality can be replaced for users who do not wish to use the STL
Header Versioning

• cl2.hpp as interface breaks backward compatibility in return for consistency

• Parameterizable with versioning macros:
  - #define CL_HPP_TARGET_OPENCL_VERSION 200
  - Intended to allow compatibility with range of underlying SDK versions

• Wide range of compatibility macros to reduce cost of moving or to avoid features:
  - STL disabling: CL_HPP_NO_STD_VECTOR, CL_HPP_NO_STD_ARRAY, CL_HPP_NO_STD_SHARED_PTR
  - Exceptions: CL_HPP_ENABLE_EXCEPTIONS
  - Switch to 1.2 kernel language for default compilation: CL_HPP_CL_1_2_DEFAULT_BUILD
OpenCL 2.1 API Enhancements

- **clCreateProgramWithIL**
  - SPIR-V support built-in to the runtime

- **Subgroup query operations**
  - Subgroups expose hardware threading in the core feature set

- **clCloneKernel** enables copying of kernel objects and state
  - Safe implementation of copy constructors in wrapper classes

- **Low-latency device timer queries**
  - Support alignment of profiling between device and host code

- **Priority and throttle hint extensions for queues**
  - Specify execution priority on a per-queue basis

- **Zero-size enqueue**
  - Zero-sized dispatches are valid from the host
OpenCL 2.1 API Create From IL

- Three defined forms of program creation:
  - From source
  - From binary
  - From IL

- IL is a well-defined input for a platform
  - Primarily SPIR-V

- Fully defined API support for the SPIR-V intermediate language
  - SPIR-V input generated offline, or through a runtime compiler library interface
  - IL is the primary input method for OpenCL kernels
OpenCL 2.1 API Subgroups

- Subgroup functionality moved into core

- Expose hardware threads/warps/wavefronts and their cross-lane operations

- Host queries for forward progress extension, and workgroup->subgroup mapping
  - Enables analysis of concurrency guarantees from the host for correct algorithm construction
OpenCL 2.1 API Clone Kernel

• Full copying of kernel objects is not possible in OpenCL 2.0
  - Either copy as a reference
  - Or recreate the kernel from scratch

• Recreation is imperfect
  - A caller has to maintain a lot of separate state and reconstruct the kernel object

• clCloneKernel copies a kernel object to produce an identical clone
  - Useful to pass to a second host thread, for example
  - Or when writing C++ wrappers for kernel objects
OpenCL 2.1 API Low-latency Timer

- Associate clocks between host and OpenCL devices
- Query host and device timer at a given point in time to relate device and host timers
- Profile when commands are run, even in the presence of high latency of dispatch
- Regularly query device timer maintaining relationship to host timer
Priority and Throttle Hints

- KHR extension for OpenCL 2.1

- Three priorities for each queue
  - Ensures that commands from a higher priority queue execute before those from a low priority queue where possible

- Three throttle levels for each queue
  - Provides a hint to the runtime to judge DVFS settings or other performance/power tradeoffs
OpenCL C++

- A static subset of C++14
  - Frees developers from low-level coding details without sacrificing performance

- Classes, templates, operator overloading etc..
  - Reusable device libraries and containers - fast and elegant sharable code
  - Templates enables meta-programming for highly adaptive software
  - Lambdas used to implement nested/dynamic parallelism

- A standard library optimized for data-parallel programming
  - Leverages C++11 standard library features
    - Atomics, meta-programming & type traits, math functions...
  - New library features
    - Work-item & Work-group functions, Dynamic parallelism, Image & Pipe functions...
Restrictions

- The dynamic_cast operator
- Type identification
- Recursive function calls
-Operators: new, delete
- goto
- Storage qualifiers: register, thread_local
- Function pointers
- Virtual functions and abstract classes
- Exception handling
- C++ standard library (with some small exceptions)
A Simple C++ Kernel

```cpp
#include <opencl_stdlib>
using namespace cl;

template<typename T>
void add_vectors(const T* srcA, const T* srcB, T* dst)
{
    size_t id = get_global_id(0);
    dst[id] = srcA[id] + srcB[id];
}

kernel void
add_vectors_float(const float* srcA, const float* srcB, float* dst)
{
    add_vectors<float>(srcA, srcB, dst);
}

kernel void
add_vectors_float4(const float4* srcA, const float4* srcB, float4* dst)
{
    add_vectors<float4>(srcA, srcB, dst);
}
```
OpenCL C++ Data Types

• Scalars
  - bool, char, short, int, float etc.
  - size_t, ptrdiff_t, intptr_t, uintptr_t, void

• Vectors
  - Vectors of 2, 3, 4, 8 or 16 boolean, integer or floating-point data types
  - Examples: int2, short4, float8, half3, bool4

• Images
  - Now template
    - image2d<typename T, image_access::a>
  - T specifies the color type returned by image reads or input to image writes
  - image_access can be sample, read, write or read_write
  - Example: image2d<float, image_access::read>

• Sampler
OpenCL C++ Data Types

• Pipe
  - Host constructed:
    - pipe<typename T, pipe_access::a>
  - New program-scope pipes:
    - program_pipe<typename T>

• Types used for enqueuing kernels
  - queue, event and ndrange
OpenCL C++ Address Spaces

- OpenCL C specifies global, local, constant and private type qualifiers

- OpenCL C++ 2.1 does not use these address space qualifiers
  - Pointers also refer to allocations in the generic address space

- For local memory allocations, the following types must be used:
  - local_ptr<typename T>
  - local_array<typename T, size_t N>
  - local<T>

- For constant memory allocations, the following types must be used:
  - constant_ptr<typename T>
  - constant_array<typename T, size_t N>
  - constant<T>
OpenCL C++ Standard Library

• Work-Item Functions: local id queries etc.

• Math Functions
  - Single-, double- and half- precision floating-point

• Atomics: a subset of the C++11 atomics library

• Image Functions: image reads with a sampler, sampler-less reads and writes

• Work- & Sub-group Functions: barrier, broadcast, scan, reduction, all and any

• Pipe Functions: read, write, reserve, commit, num_packets

• Nested Parallelism: Functions for enqueuing kernels from within the OpenCL device

• C++11 type traits with additions
Nested/Dynamic Parallelism

- Kernels can independently launch work on the device
  - without host interaction

- Kernels can enqueue:
  - a kernel function
  - code represented as a kernel lambda function

- Execution order controlled using event dependencies

- A kernel lambda function is described as:
  - [ capture-list ] ( params ) kernel { body }
Nested Parallelism - Example

#include <opencl_stdlib>
using namespace cl;

extern void barA_kernel(...);
extern void barB_kernel(...);

kernel void
foo(queue q0, queue q1, ...)
{
...  
    clk_event_t evt0;
    // enqueue kernel to queue q0
    q0.enqueue_kernel(CLK_ENQUEUE_FLAGS_NO_WAIT, range_A,
                       0, NULL, &evt0,
                       [=] kernel { barA_kernel(...); });

    // enqueue kernel to queue q1
    q1.enqueue_kernel(CLK_ENQUEUE_FLAGS_NO_WAIT, range_B,
                       1, &evt0, NULL,
                       [=] kernel { barB_kernel(...); });
}
Compile-time Dataflow

- **program_pipe**
  - A pipe object declared in program scope

- **Compile-time dataflow**
  - Pipe connectivity between kernels known at compile time

- **Enables exploitation of device specific features**
  - Localized interconnects and memory structures (bounded reach and/or visibility)
  - Higher performance / lower power

- **Declaration**
  - `program_pipe<T>::program_pipe(size_t max_packets, const pipe_properties *properties=null)`
Compile-time Dataflow - An Example

```c
program_pipe<float2> myGlobalPipe1(200);

kernel void pipeline()
{
  event evt;
  q.enqueue_kernel(flags, ndrange, 0, null, &evt,
    [=] kernel {
      // produce
      pipe<int, pipe_access::write> p(myGlobalPipe1);
      p.write_pipe(...);
    });
  q.enqueue_kernel(flags, ndrange, 1, &evt, null,
    [=] kernel {
      // consume
      pipe<int, pipe_access::read> p(myGlobalPipe1);
      p.read_pipe(...);
    });
}
```
Enables compiler ecosystem for portable parallel programs

Goal:
1) Portable binary representation of shader programs and compute kernels for GPUs and parallel computers
2) Target for OpenCL C/C++, GLSL, and other compute kernel languages
What is SPIR-V?

- Intermediate language for input to Khronos graphics and compute APIs
  - Fully specified Khronos-defined standard
  - Can natively represent Khronos graphics and compute source languages
  - Allows for conversion to/from LLVM IR
    - Khronos is working on creating LLVM <-> SPIR-V conversion tools (seeking feedback - is this useful?)

- Core for OpenCL 2.1
  - Exposes machine model for OpenCL 1.2, 2.0, 2.1
  - Supports OpenCL 1.2, 2.0, 2.1 kernel languages

- Core for Vulkan
  - Exposes machine model for Vulkan
  - Supports GLSL shader language
Why use SPIR?

Without SPIR:
- Vendors shipping source
  - Risk IP leakage
- Limited Portability
  - No ISV control over Front end
  - Different Front end implementation per vendor
- Higher runtime compilation time
- Vendors shipping multiple binaries
  - Complexity
  - Miss optimizations in new compilers
  - Forward compatibility issues

With SPIR:
- Ship a single binary
- Improved Portability
  - ISV can create their own Front end tool chain
  - Can use common Front end across vendors
- Reduced runtime compilation time
- Shipped application can retarget new devices and new vendors

Opportunity to unleash innovation:
Domain Specific Languages, C++ Compilers, Halide, ....
SPIR-V kernel language support

- Full support for OpenCL “C” and “C++” kernel languages
  - Memory and execution models support OpenCL up to v2.1
  - Generic address space
  - Device side kernel enqueue
  - C11/C++11 atomics
  - Pipes

- Shading language support in development for Vulkan
  - Adds graphics-specific instructions and resources
  - Vertex Geometry, Tesselator, Fragment, and Compute shader

- Compiler chain split in two
  - Front end compiler emits SPIR-V portable binary IL
  - SPIR-V IL is compiled to machine-specific binary by driver

- OpenCL C++ front end NOT required in driver
  - Khronos working on a C++ kernel language off-line front end
SPIR-V Introduction

- A Binary Intermediate Language
  - A linear stream of words (32-bits)
SPIR-V Introduction

- A Binary Intermediate Language
  - A linear stream of words (32-bits)
- Functions inside a module contain a CFG (control-flow graph) of basic blocks
SPIR-V Introduction

- A Binary Intermediate Language
  - A linear stream of words (32-bits)
- Functions inside a module contain a CFG (control-flow graph) of basic blocks
- Load/Store instructions are used to access declared variables
SPIR-V Introduction

• A Binary Intermediate Language
  - A linear stream of words (32-bits)
• Functions inside a module contain a CFG (control-flow graph) of basic blocks
• Load/Store instructions are used to access declared variables
• Intermediate results are represented using single static-assignment (SSA)
SPIR-V Introduction

- A Binary Intermediate Language
  - A linear stream of words (32-bits)
- Functions inside a module contain a CFG (control-flow graph) of basic blocks
- Load/Store instructions are used to access declared variables
- Intermediate results are represented using single static-assignment (SSA)
- Data objects are represented logically, with hierarchical type information
  - e.g. No flattening of aggregates or assignment to physical registers

```c
struct {
    mat3x4;
    vec4[6];
    int;
};
```
SPIR-V Introduction

- A Binary Intermediate Language
  - A linear stream of words (32-bits)
- Functions inside a module contain a CFG (control-flow graph) of basic blocks
- Load/Store instructions are used to access declared variables
- Intermediate results are represented using single static-assignment (SSA)
- Data objects are represented logically, with hierarchical type information
  - e.g. No flattening of aggregates or assignment to physical registers

```
struct {
    mat3x4;
    vec4[6];
    int;
};
```
SPIR-V Introduction

• A Binary Intermediate Language
  - A linear stream of words (32-bits)

• Functions inside a module contain a CFG (control-flow graph) of basic blocks

• Load/Store instructions are used to access declared variables

• Intermediate results are represented using single static-assignment (SSA)

• Data objects are represented logically, with hierarchical type information
  - e.g. No flattening of aggregates or assignment to physical registers

• Selectable addressing model
  - Allow usage of pointers, or dictate a memory which is purely logical.
SPIR-V Introduction

• A Binary Intermediate Language
  - A linear stream of words (32-bits)

• Functions inside a module contain a CFG (control-flow graph) of basic blocks

• Load/Store instructions are used to access declared variables

• Intermediate results are represented using single static-assignment (SSA)

• Data objects are represented logically, with hierarchical type information
  - e.g. No flattening of aggregates or assignment to physical registers

• Selectable addressing model
  - Allow usage of pointers, or dictate a memory which is purely logical.

• Can be easily extended

```
1: ExtInstImport "GLSL.std.450"  // declare use
...
40: 8 Load 39                    // load a vec4
41: 8 ExtInst 1(GLSL.std.450) 28(sqrt) 40 // take its sqrt
```
SPIR-V Introduction

- A Binary Intermediate Language
  - A linear stream of words (32-bits)
- Functions inside a module contain a CFG (control-flow graph) of basic blocks
- Load/Store instructions are used to access declared variables
- Intermediate results are represented using single static-assignment (SSA)
- Data objects are represented logically, with hierarchical type information
  - e.g. No flattening of aggregates or assignment to physical registers
- Selectable addressing model
  - Allow usage of pointers, or dictate a memory which is purely logical.
- Can be easily extended
- Support debug information that can be safely stripped without changing the semantics of SPIR-V modules.
SPIR-V Introduction

- A Binary Intermediate Language
  - A linear stream of words (32-bits)
- Functions inside a module contain a CFG (control-flow graph) of basic blocks
- Load/Store instructions are used to access declared variables
- Intermediate results are represented using single static-assignment (SSA)
- Data objects are represented logically, with hierarchical type information
  - e.g. No flattening of aggregates or assignment to physical registers
- Selectable addressing model
  - Allow usage of pointers, or dictate a memory which is purely logical.
- Can be easily extended
- Support debug information that can be safely stripped without changing the semantics of SPIR-V modules.
- Khronos is working on creating LLVM<->SPIR-V and GLSL->SPIRV conversion tool
SPIR-V Provisional Revision 31 Available!

- Samplers and Images unified between Vulkan and OpenCL!
  - Image and Sampler operations moved into SPIR-V core

- Execution model chapter now owned by each API that consumes SPIR-V

- OpenCL extended instructions merged into a single specification

- New storage classes, instructions, capabilities, and decorations

- Numerous bug fixes and clarifications
SPIR Reference Flow

Generation

Device program source → SPIR Generator → Standard Portable Intermediate

Consumption

Platform specific container → Standard Portable Intermediate → OpenCL Runtime → cl_spir
SPIR Today

**Generation**
- OpenCL C/C++
- Patched Clang
- Standard Portable Intermediate

**Consumption**
- OpenCL
- cl_spir

Device program source

Platform specific container

Standard Portable Intermediate

OpenCL Runtime
Call to Action

• Seeking feedback now on SPIR-V provisional
  - A Provisional specification, subject to change based on your feedback
  - Spec available at www.khronos.org/spir
  - Provide feedback at https://www.khronos.org/spir_feedback_forum

• Innovate on the Front end
  - New languages, abstractions
  - Target production quality Back ends

• Innovate on the Back end
  - New target platforms: Multi core, Vector, VLIW...
  - Reuse production quality frontends

• Innovate on Tooling
  - Program analysis, optimization
Overview

- Community Projects
- Common Problems
- How can Khronos help?
Overview

• Community Projects
• Common Problems
• How can Khronos help?
Context

- Provisional SPIR-V specification made public on March 2015 at GDC
- Numerous open source projects started within weeks
- New provisional version now available
  https://www.khronos.org/registry/spir-v/specs/1.0/SPIRV.html
Community Projects

- Bytecode Conversion
- Interpreter / Virtual Machine
- Assemblers / Disassemblers
- Machine Readable Specifications
Disclaimer

• All of the projects in this section are works in progress and created independently of Khronos by the open source community
spirthon - Python

- Python bytecode to SPIR-V conversion
- Write shaders or kernels in Python
- Python assembler/disassembler
- SPIR-V specification in json format
  https://github.com/cheery/spirthon
SpirvNet - C#

- Write shaders or kernels in C#
- .NET IL to SPIR-V conversion
- SPIR-V interpreter
- Debug shaders & kernels in C# (CPU)
  
  https://github.com/Philip-Trettner/SpirvNet
otherside - C++

- SPIR-V virtual machine
- Produces C like code from SPIR-V binaries
- Set through generated C code in debugger
- Basis for academic software rasterizer
  https://github.com/bonus2113/otherside
Shade - Xojo

- SPIR-V virtual machine
- Test and debug SPIR-V binaries
- Defines human readable format
- SPIR-V binary correctness
  [https://github.com/Zoclee/Shade](https://github.com/Zoclee/Shade)
SPARV - Rust

- Process SPIR-V from Rust
- (Dis)Assemble SPIR-V binaries

https://github.com/fkaa/sparv
Go SPIR-V Library

• SPIR-V library for Go
• (Dis)Assemble SPIR-V binaries
• In memory representation
  https://github.com/jteeuwen/spirv
Python SPIR-V Tools

- Encode and decode SPIR-V in Python
- High level human readable assembler syntax - similar to LLVM IR
  https://github.com/kristerw/spirv-tools
HSPRIV - Haskell EDSL

- SPIR-V like language embedded in Haskell
- Low level abstraction
- Significantly relaxed layout constraints

https://github.com/stevely/hspirv
Lisp Specification

- Lisp readable SPIR-V specification
- Basis for data driven Lisp tools
  https://github.com/cbaggers/spir-v
SPIR-V json specification

• Conversion of HTML SPIR-V specification to json format
• Basis for future data driven tools

https://github.com/Philip-Trettner/SpirvSpecToJson
Overview

• Community Projects
• Common Problems
• How can Khronos help?
Common Problems

• Data driven SPIR-V specification
• Verification of correct SPIR-V binaries
• Human readable format
Data Driven SPIR-V Specification

- A number of these open source projects have defined their own machine readable SPIR-V specification
- Divergent formats lead to confusion and fragmentation
Verification of SPIR-V binaries

- Ensure that SPIR-V binaries provided to Vulkan and OpenCL are correct
- Khronos also need this to implement the Conformance Test Suite (CTS)
Human Readable Format

- Different use cases drive requirements
  - Low level useful for debugging developer tools
  - High level useful for handwriting and inspecting SPIR-V binaries
- Variety is the spice of life
Overview

- Community Projects
- Common Problems
- How can Khronos help?
Khronos SPIR-V Spec Formats

- Khronos is providing, today
- C/C++/C++11 SPIR-V headers
- json SPIR-V specification
- Python SPIR-V specification
- Lua SPIR-V specification

https://www.khronos.org/registry/spir-v/
Khronos SPIR-V Validator

- Khronos plans to provide a validator as part of CTS development
- Consume SPIR-V binaries
- Report errors according to specification rules
- Easily integrated into build configurations and tools
- Consistency across implementations
Khronos SPIR-V (Dis)Assembler

- Khronos plans to provide a bare bones (dis)assembler
- Written in C/C++
- Used to ensure coverage of all SPIR-V features in CTS
Conclusion

- We aim to solve common problems, so you don’t have to
- Khronos will provide useful developer tools and specification formats for SPIR-V
- Stimulate further community innovation
Thanks

- Thank you to all current and future contributors to the SPIR-V ecosystem, Khronos is excited to see further innovations!
Contact

- email: k.benzie@codeplay.com
- twitter: @KmBenzie
SYCL is not magic

SYCL is a practical, open, royalty-free standard to deliver high performance software on today’s highly-parallel systems
SYCL for OpenCL - Single Source C++

- Pronounced ‘sickle’
  - To go with ‘spear’ (SPIR)

- Royalty-free, cross-platform C++ programming layer
  - Builds on concepts portability & efficiency of OpenCL
  - Ease of use and flexibility of C++

- Single-source C++ development
  - C++ template functions can contain host & device code
    - e.g. parallel_sort<MyType> (myData);
  - Construct complex reusable algorithm templates using OpenCL for acceleration

- SYCL 1.2 Final spec released!
  - At IWOCL in May 2014

- Multiple implementations
  - Including open source triSYCL from AMD
  - https://github.com/amd/triSYCL
What Does This Mean for Developers?

• Enables a standard C++11 codebase targeting multiple OpenCL devices
  - As it’s C++, a host CPU device implementation can be provided in headers

• SYCL is cross-toolchain as well as cross-platform
  - No language extensions, standard C++ compilers can build SYCL source code

• Device compilers enable SYCL running on OpenCL devices
  - Can have multiple device compilers linking into final executable
  - Doesn’t affect original source build

• You could implement SYCL simply using C++ threads
  - All the synchronization, parallelism wins remain - but running on CPU
  - No external dependencies
What Does This Mean for Developers?

• Enables developers to move quickly into writing SYCL code
  - Provides methods for dealing with targets that do not have OpenCL(yet!)
  - Has other development benefits...

• A fallback CPU implementation is debuggable!
  - Using normal C++ debuggers
  - Profiling tools also work on CPU device

• Huge bonus for productivity and adoption
  - Cost of entry to use SYCL very low
Minimal Example

// Create a queue on the default device.
cl::sycl::queue& queue;

// Wrap data in a 1D buffer.
cl::sycl::buffer<float, 1>& buf(...);

// Begin submitting work to an OpenCL device.
queue.submit([&](cl::sycl::handler& cgh) {
   // Create an accessor that tracks the dependencies.
   auto acc = buf.get_access<access::write, access::global_buffer>(cgh);

   // The OpenCL kernel is represented by the lambda function here.
   cgh.parallel_for<class kernel>(range,
       [=](cl::sycl::item<2> item) {
           acc[item.get_global(0)] = item.get_global(0);
        });
});
```cpp
#include <CL/sycl.hpp>

int main () {
    // Device buffers
    buffer<float, 1> buf_a(array_a, range<1>(count));
    buffer<float, 1> buf_b(array_b, range<1>(count));
    buffer<float, 1> buf_c(array_c, range<1>(count));
    buffer<float, 1> buf_r(array_r, range<1>(count));

    queue myQueue;
    myQueue.submit([&](handler& cgh) {
        // Data accessors
        auto a = buf_a.get_access<access::read> ();
        auto b = buf_b.get_access<access::read> ();
        auto c = buf_c.get_access<access::read> ();
        auto r = buf_r.get_access<access::write> ();

        // Kernel
        cgh.parallel_for<class three_way_add>(count, [=](id<> i) {
            r[i] = a[i] + b[i] + c[i];
        });
    });
}
```
How did we come to our decisions?

What was our thinking?
How do we bring OpenCL features to C++?

• Key decisions:
  - We will not add any language extensions to C++
  - We will work with existing C++ compilers
  - We will provide the full OpenCL feature-set in C++
What features of OpenCL do we need?

• We want to enable all OpenCL features in C++ with SYCL
  - Images, work-groups, barriers, constant/global/local/private memory
  - Memory sharing: mapping and DMA
  - Platforms, contexts, events, queues
  - Support wide range of OpenCL devices: CPUs, GPUs, FPGAs, DSPs...

• We want to make it easy to write high-performance OpenCL code in C++
  - SYCL code in C++ must use memory and execute kernels efficiently
  - We must provide developers with all the optimization options they have in OpenCL

• We want to enable OpenCL C code to interoperate with C++ SYCL code
  - Sharing of contexts, memory objects etc.
Single-source vs C++ kernel language

• Single-source: a single-source file contains both host and device code
  - Type-checking between host and device
  - A single template instantiation can create all the code to kick off work, manage data and execute the kernel
    - e.g. sort<MyClass> (myData);
  - The approach taken by C++ 17 Parallel STL as well as SYCL

• C++ kernel language
  - Matches standard OpenCL C
  - Proposed for OpenCL v2.1
  - Being considered as an addition for SYCL v2.1
Why ‘name’ kernels?

- Enables implementers to have multiple, different compilers for host and different devices
  - With SYCL, software developers can choose to use the best compiler for CPU and the best compiler for each individual device they want to support
  - The resulting application will be highly optimized for CPU and OpenCL devices
  - Easy-to-integrate into existing build systems

- Only required for C++11 lambdas, not required for C++ functors
  - Required because lambdas don’t have a name to enable linking between different compilers

```cgh.parallel_for<class assign_elements>(myRange, [=](item<1> item)
{
    ptr[item.get_global(0)] = item.get_global(0);
});```
Buffers/images/accessors vs pointers

- OpenCL v1.2 supports a wide range of different devices and operating systems
- All shared data must be encapsulated in OpenCL memory objects: buffers and images
- To enable SYCL to achieve maximum performance of OpenCL, we follow OpenCL’s memory model approach
- But, we apply OpenCL’s memory model to C++ with buffers, images and accessors
- Separation of data storage and data access
- Enables efficient scheduling
const int n_items = 32;
range<1> r(n_items);

int array_a[n_items] = { 0 };
int array_b[n_items] = { 0 };

buffer<int, 1> buf_a(array_a, range<1>(r));
buffer<int, 1> buf_b(array_b, range<1>(r));

queue q;
q.submit([&](handler& cgh)
{
    auto acc_a = buf_a.get_access<read_write>(cgh);
    algorithm_a s(acc_a);
    cgh.parallel_for(n_items, s);
});

q.submit([&](handler& cgh)
{
    auto acc_b = buf_b.get_access<read_write>(cgh);
    algorithm_b s(acc_b);
    cgh.parallel_for(n_items, s);
});

q.submit([&](handler& cgh)
{
    auto acc_a = buf_a.get_access<read_write>(cgh);
    algorithm_c s(acc_a);
    cgh.parallel_for(n_items, s);
});
Hierarchical parallelism

• A whole new approach

• Enables high-performance, portable C++ template algorithms to work across CPUs, GPUs and other devices easily

• Is really just syntactical
Hierarchical parallelism

cgh.parallel_for_work_group< class k >(range,
  [=](group<2> groupID) {
    parallel_for_work_item(groupID, [=](item<2> itemID) {
      // ... Do something per work item...
    });
    // Potential per workgroup code and/or implicit barriers.
    // ...
    parallel_for_work_item(groupID, [=](item<2> itemID) {
      // ... Do something else per work item...
    });
  });
What can I do with SYCL?

Anything you can do with C++!

With the performance and portability of OpenCL
Building the SYCL for OpenCL ecosystem

To deliver on the full potential of high-performance heterogeneous systems
- We need the libraries
- We need integrated tools
- We need implementations
- We need training and examples

An open standard makes it much easier for people to work together
- SYCL is a group effort
- We have designed SYCL for maximum ease of integration
Questions

And maybe some volunteering of joining in to build the ecosystem?
Questions and Call to Action!

- These slides will be posted online later today
  - www.khronos.org

- Khronos seeking feedback on Vulkan, SPIR and OpenCL 2.1
  - Links provided on Khronos forums
  - https://www.khronos.org/opencl/opencl_feedback_forum
  - https://www.khronos.org/spir_v_feedback_forum
  - https://www.khronos.org/vulkan/vulkan_feedback_forum

- Any company or organization is welcome to join Khronos for a voice and a vote in any of these standards

- More Information
  - www.khronos.org
  - ntrevett@nvidia.com
  - @neilt3d