OpenCL Overview and Update

Neil Trevett | Khronos President
NVIDIA Vice President Developer Ecosystem
OpenCL Working Group Chair
ntrevett@nvidia.com | @neilt3d
Santa Clara, May 2019
Khronos Mission

Khronos members are industry leaders from around the world that join to safely cooperate - to advance their own businesses and the industry as a whole.

Khronos is an open, member-driven industry consortium developing royalty-free standards, and vibrant ecosystems, to harness the power of silicon acceleration for demanding graphics rendering and computationally intensive applications.
Active Khronos Standards

Industry involvement, engagement and feedback are the lifeblood of any successful standard
Levels of Khronos Engagement

- **OpenCL Forum and Slack Channel**
  - Open to all!
  - [https://community.khronos.org/c/opencl](https://community.khronos.org/c/opencl)
  - [www.khr.io/slack](http://www.khr.io/slack)

- **Contribute to open source specs, CTS and tools**
- **Advisory Panel**
- **Working Group**

  - Spec fixes and suggestions made under the Khronos IP Framework. Open source contributions under repo’s CLA - typically Apache 2.0
  - [https://github.com/KhronosGroup](https://github.com/KhronosGroup)

  - Advisors under the Khronos IP Framework can comment and contribute to requirements and draft specifications

  - Khronos members under IP Framework and NDA can participate and vote in working group meetings
OpenCL Heterogeneous Computing

A programming and runtime framework for heterogeneous compute resources
Low-level control over memory allocation and parallel task execution
Simpler and relatively lightweight compared to GPU APIs

Application

Fragmented GPU API Landscape

Growing number of optimized OpenCL libraries
Vision and Imaging
Machine Learning and Inferencing
Linear Algebra and Mathematics
Physics

Many mobile SOCs and Embedded Systems becoming increasingly heterogeneous
Autonomous vehicles
Vision and inference

Heterogeneous Compute Resources

CPU
GPU
FPGA
DSP
Custom Hardware
OpenCL - Low-level Parallel Programming

- Low-level programming of heterogeneous parallel compute resources
  - One code tree can be executed on CPUs, GPUs, DSPs and FPGA ...

- OpenCL C or C++ language to write kernel programs to execute on any compute device
  - Platform Layer API - to query, select and initialize compute devices
  - Runtime API - to build and execute kernels programs on multiple devices

- The programmer gets to control:
  - What programs execute on what device
  - Where data is stored in various speed and size memories in the system
  - When programs are run, and what operations are dependent on earlier operations
OpenCL Versions

OpenCL 1.2 is the de-facto baseline version for all non-FPGA processors

OpenCL C++ Kernel Language
- Static subset of C++14
- Templates and Lambdas
- SPIR-V 1.2 with C++14 support
  - e.g. constructors/destructors
- Pipes
  - Efficient device-scope communication between kernels
- Code Gen Optimizations:
  - Specialization constants at SPIR-V compilation time
  - Constructors and destructors of program scope global objects
  - User callbacks can be set at program release time

SPIR-V in Core
- Subgroups into core
- Subgroup query operations
- clCloneKernel
- Low-latency device timer queries

OpenCL 1.0: Dec08
- Specification
- 3-component vectors
- Additional image formats
- Multiple hosts and devices
- Buffer region operations
- Enhanced event-driven execution
- Additional OpenCL C built-ins
- Improved OpenGL data/event interop

OpenCL 1.1: Jun10
- Specification
- Device partitioning
- Separate compilation and linking
- Generic Address Space
- Enhanced image support
- Built-in kernels / custom devices
- Enhanced DX and OpenGL Interop

OpenCL 1.2: Nov11
- Specification
- OpenCL 1.2
- 18 months

OpenCL 2.0: Nov13
- Specification
- Enhanced event-driven execution
- Additional OpenCL C built-ins
- Improved OpenGL data/event interop

OpenCL 2.1: Nov15
- Specification
- 18 months

OpenCL 2.2: May17
- Specification
- 18 months

OpenCL 2.2 is the de-facto baseline version for all non-FPGA processors
OpenCL Conformant Implementations

Vendor timelines are first conformant submission for each spec generation

Dec08 OpenCL 1.0 Specification
Jun10 OpenCL 1.1 Specification
Nov11 OpenCL 1.2 Specification
Nov13 OpenCL 2.0 Specification
Nov15 OpenCL 2.1 Specification

Adoption Since Last EVS
Intel: 2.1 for latest processors on Windows and Linux
Intel: 1.0 on Arria 10 GX FPGA
NVIDIA: 1.2 for Turing GPUs on Windows and Linux
Qualcomm: 2.0 on Adreno GPUs on Android
OpenCL Apps and Libs

OpenCL is Pervasive!

Desktop Creative Apps

Parallel Computation Languages

Linear Algebra Libraries

Math and Physics Libraries

Vision and Imaging Libraries

OpenCL is Pervasive!

70% Increase in two years

# OpenCL-based GitHub Repos

OpenCL

OpenVINO

Intel

MACE

Huawei

cIDNN

Intel

TensorFlow

Arm Compute Library

Android NNAPI

Caffe

TI Deep Learning Library (TIDL)

AMD MIVisionX

Synopsys MetaWare EV

Glow

Machine Learning Inference Compilers

Machine Learning Libraries

syCL-BLAS

CLBlast

viennaCL

Android NNAPI

Arm Compute Library

Qualcomm Neural Processing SDK for AI

AMD MIVisionX

VeriSilicon

TensorFlow

Caffe

TI Deep Learning Library (TIDL)

OpenACC

Parallel Computation Languages

Linear Algebra Libraries

Math and Physics Libraries

Vision and Imaging Libraries

https://www.khronos.org/opencl/resources/opencl-applications-using-opencl

https://www.iwocl.org/resources/opencl-libraries-and-toolkits/
OpenCL Open Source Specs and Tests

**Khronos has open sourced OpenCL Specifications and Conformance Tests**

- Merge requests welcome from the community (subject to review by OpenCL working group)
- File bugs for specifications, headers etc.
- Mix your own documentation!

**Contributions and Distribution under Apache 2.0**

- Conformance Test Suite Source
- Spec and Ref Language Source
- Spec Build System and Scripts

- Contributions and Distribution under Apache 2.0
- Redistribution under CC-BY 4.0

- Khranos builds and Ratifies Canonical Specification under Khronos IP Framework. No changes or re-hosting allowed
- Spec and Ref Language Source and derivative materials. Re-mixable under CC-BY by the industry and community

**Khranos Adopters Program**

- Anyone can test any implementation at any time
- Conformant Implementations can use trademark and are covered by Khronos IP Framework

**Community built documentation and tools**

- Any bugs for specifications, headers etc. Mix your own documentation!
Underway - Unified OpenCL Specification

• Unified OpenCL API specification will describe the API for all versions of OpenCL
  - Rather than having a separate specification per version

• OpenCL SPIR-V environment, extension and SPIR-V specs are already unified
  - Working well - good developer feedback

• Easier for developers to navigate
  - And to consistently apply specification fixes and clarifications

• Working Group has started prototyping the spec work
  - Short introductory section describing the unified aspects
  - "missing before X.Y" and "deprecated by X.Y" language
  - Roughly as SPIR-V specification
  - https://github.com/KhronosGroup/OpenCL-Docs/issues/77

• DRAFT unified spec is already uploaded!
  - Feedback welcome!
OpenCL Evolution

OpenCL Extension Specs
- Vulkan / OpenCL Interop
- Scratch-Pad Memory Management
- Extended Subgroups
- SPIR-V 1.4 ingestion for compiler efficiency
- SPIR-V Extended debug info

Spec Maintenance Updates
- Regular updates for spec clarifications and bug fixes

Integration of Extensions plus New Core functionality
- Vulkan-like loader and layers
- ‘Flexible Profile’ for Deployment Flexibility

Target 2020 ‘OpenCL Next’

Repeat Cycle for next Core Specification

May 2017 OpenCL 2.2
Embedded Processors & OpenCL Conformance

- The embedded market is a new frontier needing advanced heterogenous compute
  - E.g. Vision and inferencing using a wide range of processor architectures
- BUT OpenCL is currently monolithic - and arguably desktop/HPC-centric
  - E.g. a processor without 32-bit IEEE floating point cannot realistically be conformant
  - Vendors and developers do not want software emulation of higher precisions
- Many functionality requirements change between different markets and processors

<table>
<thead>
<tr>
<th>Supported Precisions</th>
<th>DSP A</th>
<th>DSP B</th>
<th>DSP C</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit integer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>16-bit integer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>32-bit integer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>64-bit integer</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>16-bit float</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>32-bit float</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>64-bit float</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Possible to be OpenCL Compliant?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

OpenCL can drive significantly more adoption in the embedded processor market
Flexible Profile and Feature Sets

- In OpenCL Next Flexible Profile features become optional for enhanced deployment flexibility
  - API and language features e.g. floating point precisions
- Feature Sets reduce danger of fragmentation
  - Defined to suit specific markets - e.g. desktop, embedded vision and inferencing
- Implementations are conformant if fully support feature set functionality
  - Supporting Feature Sets will help drive sales - encouraging consistent functionality per market
  - An implementation may support multiple Feature Sets
OpenCL Next Feature Set Discussions

- OpenCL Next Flexible Profile will leverage the new Unified OpenCL Specification
  - Feature Sets can easily select from any previous functionality

- What could be useful Feature Sets?
  - Feature sets for previous spec versions e.g. OpenCL 1.2?
  - ‘Desktop’ Feature Set to raise the universally available baseline above OpenCL 1.2?
  - Set of OpenCL functionality that runs efficiently over Vulkan?
  - Vertical market focused - e.g. inferencing, vision processing

- Some vertically integrated markets don’t care about cross-vendor app portability
  - But still value in industry standard programming framework - reduces costs for engineering, tooling, training etc.
  - Allow conformance for any combination of features - no Feature Sets
  - Enables minimal footprint OpenCL per system - ideal for Safety Certification

In some vertically integrated embedded markets, minimized driver footprint is key

In some markets, e.g. desktop, software portability across multiple vendors’ accelerators is key
OpenCL Tooling Ecosystem Subgroup

- Coordinating collaborative opportunities between SPIR-V and LLVM ecosystems
  - Encouraging joint development of new features and tooling integration
- Active open source projects - making SPIR-V a first-class LLVM citizen
  - Extending SPIR-V<->LLVM Translation for OpenCL - release 8.0 is out!
    - [https://github.com/KhronosGroup/SPIRV-LLVM-Translator](https://github.com/KhronosGroup/SPIRV-LLVM-Translator)
  - Libclc: implementation of the OpenCL C 1.1 library for use with Clang
    - [https://libclc.llvm.org/](https://libclc.llvm.org/)
  - Upstream SPIR-V backend translation from Clang/LLVM - in discussion
    - [https://clang.llvm.org/](https://clang.llvm.org/)
  - Front-end support for all OpenCL C language versions in Clang
- C++ for OpenCL in Clang
  - Experimental support for C++ in OpenCL

Khronos increasing efforts around developing, coordinating and releasing open source tooling

Examples and Tutorials

Make it approachable

Tools

Make it usable

Conformance Tests

Make it reliable

Specifications

Make it possible
C++ for OpenCL in Clang Project

- Front end with OpenCL C 2.0 and C++17 capabilities
  - Experimental support in Clang 9.0
  - Expect Alpha in September 2019

- Existing OpenCL C code is valid and fully compatible
  - Enables gradual transition to C++ for existing apps

- Offline compilation into SPIR-V or device binary
  - Generates SPIR-V 1.0 for most features
  - Uses SPIR-V 1.2 where necessary

- Works with any OpenCL 2.0 driver
  - Possible future driver updates may take advantage of enhanced language capabilities

- Check it out in Compiler Explorer
  - https://godbolt.org/z/nGvxAC

```
template<class T> T add( T x, T y )
{
    return x + y;
}

__kernel void test( __global float* a, __global float* b)
{
    auto index = get_global_id(0);
    a[index] = add(b[index], b[index+1]);
}
```

Good example of the power and flexibility of offline compilation using SPIR-V - how can we embrace and support this class of language project?
Current OpenCL Compilation Flow

Frontend and backend compilers ship with OpenCL driver
Adding new language capabilities → new driver version
Offline Compilation with SPIR-V

Offline compilation stage has no direct driver interaction

If offline compiler generates valid SPIR-V, front-end can add language capabilities without requiring a driver update
Examples: Variadic Macros, Atomic Functions w/ Address Spaces, Templates are possible with no SPIR-V changes
Community OSS-Driven Language Evolution

- Language capabilities can move faster with offline compilation
  - Benefits Khronos: easier to explore and iterate on new features
  - Benefits implementers: no driver updates required for new language features
  - Benefits developers: one consistent tool enables all implementations

- Proposal -> accelerate towards community-based language ecosystem
  - Host non-ratified offline language documentation at Khronos - agile updates
  - Add Compiler Capabilities extension to core OpenCL - using preprocessor #defines
  - Code can choose to interrogate extension for enhanced compiler capabilities

- N.B. NOT proposing to remove OpenCL C online compilation!
  - Key to many use-cases - and can also absorb new features over time
  - [https://github.com/KhronosGroup/OpenCL-Docs/issues/65](https://github.com/KhronosGroup/OpenCL-Docs/issues/65)
SPIR-V Ecosystem

- **Third party kernel and shader languages**
  - GLSL
  - HLSL
  - glslang
  - DXC

- **SPIR-V (Dis)Assembler**
- **SPIRV-Cross**
- **SPIRV-opt | SPIRV-remap**

- **SPIR-V Validator**

- **Optimization Tools**
  - SPIR-V Magic
  - SPIR-V Version 99
  - Builder's Magic
  - OpMemoryModel
  - Logical
  - GLSL
  - 0
  - OpEntryPoint
  - Fragment shader
  - function
  - <id> = 4
  - OpTypeVoid
  - <id> = 2
  - OpTypeFunction
  - <id> = 3
  - Result Type <id> = 2
  - OpFunction
  - Result <id> = 4
  - 0
  - Function Type <id> = 3

- **Environment spec used for compilation needs to match target runtime**

- **IHV Driver Runtimes**
  - Vulkan
  - OpenGL

- **Khronos-hosted Open Source Projects**
  - 3rd Party Hosted OSS

- **SYCL for ISO C++ Front-end**
- **OpenCL C Front-end**
- **OpenCL C++ Front-end**

- **LLVM to SPIR-V Bi-directional Translators**

- **CLSpv**

- **LLVM**

- **C++ for OpenCL in clang Front-end**

- **SPIR-V**
  - Khronos-defined cross-API IR
  - Native graphics and parallel compute support
  - Easily parsed/extended 32-bit stream
  - Data object/control flow retained for effective code generation/translation

- **Khronos cooperating with clang/LLVM Community**

- **Website**
  - https://github.com/KhronosGroup/SPIRV-Tools
OpenCL Platform Deployment Flexibility

- **Clspv** - Google’s experimental compiler for OpenCL C to Vulkan SPIR-V
  - Open source - tracks top-of-tree LLVM and clang, not a fork
  - Originally tested on over 200K lines of Adobe OpenCL C production code
  - Sony is now working with Google to compile their production kernels
    - 355 kernels in 75 files - 40 files compiled successfully at first pass

- **Clvk** - experimental OpenCL to Vulkan API shim by Kevin Petit
  - Early days - but Halide’s OpenCL back-end successfully running over Vulkan

---

**OpenCL C Source**

**OpenCL Host Code**

Prototype open source project [https://github.com/google/clspv](https://github.com/google/clspv)

Prototype open source project [https://github.com/kpet/clvk](https://github.com/kpet/clvk)
Refining clspv with Diverse Workloads

Compile diverse OpenCL C kernel workloads

Efficient mapping to Vulkan SPIR-V?

Updates to compiler achieve efficient mapping?

Propose updates to Vulkan programming model

Kernel repositories for use in long-term perf/regression testing - including kernels from open source OpenCL libraries
https://github.com/KhronosGroup/OpenCL-Sample-Kernels

ISVs can compile and test commercially-sensitive kernels - and report issues on GitHub

Please let us know if you are interested to try running your kernels through clspv!

Vulkan is already expanding its compute model e.g. 16-bit storage, compact memory types and operations, Variable Pointers, Subgroups

Community assistance to add capabilities is welcome!

Increasing deployment options for OpenCL developers to any platform where Vulkan is a supported API e.g. Android
OpenCL Portability Initiative on Apple??

Is there community interest in cooperation over in a direct ‘OpenCL over Metal’ OSS project? Can Khronos host/help coordinate?

Would need runtime conversion of API calls. OpenCL is much closer to Metal than Vulkan - easier to shim.

SPIRV-Cross would need expanding to handle OpenCL dialect of SPIR-V.

macOS / iOS Run-time
Maps OpenCL to Metal API Calls
Get Involved!

- OpenCL is driving to new levels of usability and deployment flexibility
  - We want to know what you need from OpenCL!
- OpenCL Next and Feature Sets
  - Let us know what you think!
- Multiple ways to engage and help OpenCL evolve
  - Join Khronos for a voice and a vote in any of these standards
  - Or ask about an invite to the OpenCL Advisory Panel
  - Or consider getting involved in OpenCL OSS projects
    https://github.com/KhronosGroup
  - Or talk to us on Slack and the forums
    https://community.khronos.org/c/opencl
    www.khr.io/slack
- Neil Trevett
  - ntrevett@nvidia.com
  - @neilt3d
  - www.khronos.org

If you need OpenCL let your hardware vendors know!
Your voice counts!