OpenCL 3.0

Neil Trevett
Khronos President
OpenCL Working Group Chair
NVIDIA VP Developer Ecosystems
ntrevett@nvidia.com | @neilt3d

April 2020
Agenda

- OpenCL Momentum
- OpenCL Evolution and OpenCL 3.0
- Extensions and Roadmap
- Layered OpenCL
- Get Involved!

https://www.khronos.org/registry/OpenCL/
Increasing industry interest in parallel compute acceleration to combat the ‘End of Moore’s Law’
OpenCL is Widely Deployed and Used

<table>
<thead>
<tr>
<th>Accelerated Implementations</th>
<th>The industry's most pervasive, cross-vendor, open standard for low-level heterogeneous parallel programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop Creative Apps</td>
<td>Intel OpenVINO, NVIDIA CUDA, AMD ML, ARM ML, Intel MKL, NVIDIA cuDNN, Intel MKL-DNN, AMD ML</td>
</tr>
<tr>
<td>Linear Algebra Libraries</td>
<td>Intel cuBLAS, NVIDIA cuBLAS, AMD BLAS, Intel MKL-DNN, NVIDIA cuDNN, AMD ML</td>
</tr>
<tr>
<td>Machine Learning Libraries</td>
<td>Intel OpenVINO, NVIDIA CUDA, AMD ML, ARM ML, Intel MKL-DNN, NVIDIA cuDNN, AMD ML</td>
</tr>
<tr>
<td>Machine Learning Frameworks</td>
<td>Intel OpenVINO, NVIDIA CUDA, AMD ML, ARM ML, Intel MKL-DNN, NVIDIA cuDNN, AMD ML</td>
</tr>
<tr>
<td>Molecular Modelling Libraries</td>
<td>Charmm, Gromacs, OpenMM, Fold@Home, P-R</td>
</tr>
</tbody>
</table>

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

OpenCL Open Source Ecosystem Momentum

# OpenCL-based GitHub Repos

Doubling in under three years

By April 13, Folding@Home hit a new record of 2.4 exaflops, faster than the top 500 traditional supercomputers combined, thanks to almost 1 million new members of the network. Folding@Home uses OpenCL to offload computations onto the GPUs contained in the networked home PCs.
OpenCL Standards Evolution

SYCL 1.2
Over OpenCL 1.2 using SPIR
2015

SYCL 2.2
Over OpenCL 2.X
2016

SYCL 1.2.1
Over OpenCL using SPIR-V
2017

SPIR
Using LLVM IR
2014

SPIR-V
Standalone IR
2015

OpenCL Layered
Over Vulkan using SPIR-V
2019

OpenCL C
Specification
2008

OpenCL 1.2
Creates Parallel Programming Baseline
2008-2011

OpenCL 2.X
Functionality in Monolithic Specification
2013-2017

OpenCL C++
Specification
2017

C++ for OpenCL using Clang/LLVM
2019

Vulkan 1.0
2016

OpenCL
Vulkan Convergence
2018

Vulkan 1.2
2020

Next Steps

SYCL 2020
Multiple backends

OpenCL Layered
Over Multiple Target APIs for Deployment Flexibility

Broad LLVM/Clang Language Cooperation

Increased Pervasive Core Functionality for App Portability

Increased Optionality for Embedded Flexibility

Design Influences

Kernel Languages

OpenCL APIs

SYCL

SPIR

Vulkan

OpenCL Layered Over Multiple Target APIs for Deployment Flexibility

Broad LLVM/Clang Language Cooperation

Increased Pervasive Core Functionality for App Portability

Increased Optionality for Embedded Flexibility

SYCL 2.2.1
Over OpenCL using SPIR-V
2017
OpenCL 3.0

Increased Deployment Flexibility
All functionality beyond OpenCL 1.2 is optional
Unified API specification slices OpenCL 2.X functionality into coherent, queryable, optionality
OpenCL C 3.0 language specification adds macros for optional language features

Subgroups with SPIR-V 1.3
New (optional) core functionality

Asynchronous DMA extension
Enabling a new class of Embedded Processors

OpenCL C++ not included
Ecosystem has transitioned to open source C++ for OpenCL

Easy for Developers to upgrade to OpenCL 3.0
NO code changes necessary if all used functionality is present
Applications encouraged to query used OpenCL 2.X functionality for future portability

Easy for Implementers to upgrade to OpenCL 3.0
Add queries for OpenCL 2.X functionality - missing or present
Update reported version and add minor entry points for improved app portability
OpenCL 3.0 Design Philosophy

**Increase deployment flexibility**
Conformant implementations can focus on functionality for their target markets
OpenCL 2.2 functionality sliced into coherent, queryable, optionality
Everything beyond OpenCL 1.2 is made optional
OpenCL C 3.0 language specification adds macros for optional language features

**Set the stage for new pervasively available core functionality**
New core specifications can carefully integrate new widely accepted functionality
Adoption not blocked by the monolithic OpenCL 2.X specification
OpenCL Roadmap

Unified API Specification
- All OpenCL versions documented in one place
- Tightly organized queries for all 2.X functionality
- OpenCL C 3.0 Language - macros for optional features
- Subgroups and SPIR-V 1.3
- New (optional) core functionality
- Asynchronous DMA extension
  - Enabling a new class of Embedded Processors

Extension Pipeline
- Extended Subgroups
- Device UUID Query
- Extended Debug Info
- External Memory Sharing
- Vulkan/OpenCL Interop
- Recordable Command buffers?
- Machine Learning Primitives?
- Indirect Dispatch?
- Device Topology?

Flexible Profile
- Finer-grain optional functionality for embedded processors
- ‘Layering’ Profile?
  - Defined queries and conformance for layered implementations?

OpenCL C 3.0 Language - macros for optional features

Asynchronous DMA extension
- Enabling a new class of Embedded Processors

C++ for OpenCL
- Open source C++ kernel language
  - front-end leveraging Clang and LLVM

Regular Maintenance Updates
- Clarifications, formatting, bug fixes

OpenCL 3.0
- April 2020

Extension Pipeline
- Extended Subgroups
- Device UUID Query
- Extended Debug Info
- External Memory Sharing
- Vulkan/OpenCL Interop
- Recordable Command buffers?
- Machine Learning Primitives?
- Indirect Dispatch?
- Device Topology?

Flexible Profile
- Finer-grain optional functionality for embedded processors
- ‘Layering’ Profile?
  - Defined queries and conformance for layered implementations?

OpenCL 3.0
- April 2020

New Pervasive Functionality in Core Specification
- Integrate proven, widely adopted extensions

OpenCL 3.0
- April 2020

Time
Unified OpenCL 3.0 API Specification

• Describes the API for all versions of OpenCL
  - Rather than having a separate specification per version
  - Matches SPIR-V environment, extension and SPIR-V specs

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

• Describes deprecation and version evolution rationale
  - Short introductory section describing the unified aspects
  - "missing before X.Y" and "deprecated by X.Y" language

• All specification sources in open source on GitHub
  - Accepting community bug reports and pull requests

https://www.khronos.org/registry/OpenCL/
Upgrading to OpenCL 3.0

For Developers

Upgrade from using OpenCL 1.2 to OpenCL 3.0
No code changes necessary
OpenCL 1.2 apps run unchanged on any OpenCL 3.0 device

Upgrade from using OpenCL 2.X to OpenCL 3.0
No code changes necessary if upgraded device drivers support all used functionality

Query for Deployment Portability
All applications encouraged to query used V2.X functionality
All OpenCL 2.x API features can be queried
OpenCL C 3.0 macros for optional language features

For Implementers

Upgrade OpenCL 1.2 driver to OpenCL 3.0
Easy upgrade with minimal effort
Update reported version and add queries to report OpenCL 2.X functionality as missing
Add some minor entry points for improved app portability
Free to add any desired OpenCL 2.X features

Upgrade OpenCL 2.X driver to OpenCL 3.0
Can continue to ship existing functionality with full backwards compatibility
Add queries for OpenCL 2.X functionality - missing or present
May choose to drop OpenCL 2.X features if not relevant to target markets to reduce costs and increase quality
C++ for OpenCL

- Open source offline compiler to SPIR-V or device binary
  - Replaces the OpenCL C++ kernel language
  - Language documentation available
    - https://github.com/KhronosGroup/Khronosdotorg/blob/master/api/opencl/assets/CXX_for_OpenCL.pdf

- Enables full OpenCL C and most C++17 capabilities
  - OpenCL C code is valid and fully compatible
  - Enables gradual transition to C++ for existing apps

- Uses Clang/LLVM
  - Generates SPIR-V 1.0 plus SPIR-V 1.2 where necessary
  - Experimental support added in Clang 9
  - Bug fixes and improvements in Clang 10

- Check it out in Compiler Explorer
  - https://godbolt.org/z/NGZw9U

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

```clang```
clang -cl-std=clc++ test.cl
```
```cpp
__kernel void test(__global float* a, __global float* b)
{
    auto index = get_global_id(0);
    a[index] = add(b[index], b[index+1]);
}
```
Asynchronous DMA Extensions

OpenCL embraces a new class of Embedded Processors
Many DSP-like devices have Direct Memory Access hardware

Transfer data between global and local memories via DMA transactions
Transactions run asynchronously in parallel to device compute enabling wait for transactions to complete
Multiple transactions can be queued to run concurrently or in order via fences

OpenCL abstracts DMA capabilities via extended asynchronous workgroup copy built-ins
(New!) 2- and 3-dimensional async workgroup copy extensions support complex memory transfers
(New!) async workgroup fence built-in controls execution order of dependent transactions
New extensions complement the existing 1-dimensional async workgroup copy built-ins

Async Fence controls order of dependent transactions
All transactions prior to async_fence must complete before any new transaction starts, without a synchronous wait

Async 3D-3D Copy Transaction
Reshaping possible
$V_{global} = V_{local}$

The first of significant upcoming advances in OpenCL to enhance support for embedded processors
Roadmap: External Sharing and Interop

- Generic extension to import external memory and semaphores exported by other APIs
  - Explicitly hand-off memory ownership with OpenCL
  - Wait and signal imported external semaphores

- Layer with API-specific interop extensions
  - Vulkan interop first
  - DX12 and other APIs in the future

- Improved flexibility over previous interop APIs using implicit resources
  - As were used for DX9-11 and OpenGL
Roadmap: ‘Flexible Profile’

**Goals**

Conformant OpenCL implementations on diverse embedded processors and platforms
Enable vendors to ship conformant functionality precisely targeting their customers and markets
Implementers use OpenCL as flexible runtime framework that can be pervasively and cost-effectively deployed

<Diagram showing relationships between Apps and Libs, OpenCL Drivers, Accelerators, and OS>

**Design Philosophy**

Additional OpenCL features become optional for increased deployment flexibility
Optionality includes both API and language features e.g. floating-point precisions
Enhanced query mechanisms - precisely which features are supported by a device?
Enables minimal footprint OpenCL - ideal for Safety Certification

Vertically integrated apps, drivers and accelerators E.g. Embedded
Conformance for customer-focused functionality is implementer priority

App deployment across multiple vendors’ accelerators E.g. Desktop
Portability is developer priority
Shared common functionality

Vertically integrated apps, drivers and accelerators E.g. Embedded
Conformance for customer-focused functionality is implementer priority

<table>
<thead>
<tr>
<th>Apps and Libs</th>
<th>Apps and Libs</th>
<th>Apps and Libs</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenCL Drivers</td>
<td>OpenCL Drivers</td>
<td>OpenCL Drivers</td>
</tr>
<tr>
<td>Accelerators</td>
<td>Accelerators</td>
<td>Accelerators</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Apps and Libs</th>
<th>Apps and Libs</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenCL Drivers</td>
<td>OpenCL Drivers</td>
</tr>
<tr>
<td>Accelerator</td>
<td>Accelerator</td>
</tr>
</tbody>
</table>

OS

OS
## API Layering

<table>
<thead>
<tr>
<th>Layers Over</th>
<th>Vulkan</th>
<th>OpenGL</th>
<th>OpenCL</th>
<th>OpenGL ES</th>
<th>DX12</th>
<th>DX9-11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vulkan</td>
<td></td>
<td>Zink</td>
<td>clspv</td>
<td>GLOVE</td>
<td>vkd3d</td>
<td>DXVK</td>
</tr>
<tr>
<td></td>
<td>^</td>
<td></td>
<td>clvk</td>
<td>Angle</td>
<td></td>
<td>WineD3D</td>
</tr>
<tr>
<td>OpenGL</td>
<td>gfx-rs</td>
<td></td>
<td></td>
<td>Angle</td>
<td></td>
<td>WineD3D</td>
</tr>
<tr>
<td></td>
<td>Ashes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DX12</td>
<td>gfx-rs</td>
<td></td>
<td></td>
<td>Microsoft ‘GLOn12’</td>
<td></td>
<td>Microsoft D3D11On12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Microsoft ‘CLOn12’</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DX9-11</td>
<td>gfx-rs</td>
<td></td>
<td></td>
<td>Angle</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ashes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metal</td>
<td>MoltenVK</td>
<td>gfx-rs</td>
<td></td>
<td>MoltenGL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>clspv + SPIRV-Cross?</td>
<td>Angle</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**COLUMNS** Benefit ISVs by making an API available everywhere
**Application deployment flexibility by fighting platform fragmentation**
Making an API available across multiple platforms even if no native drivers available

**ROWS** Benefit Platforms by adding APIs
Enable content without additional kernel level drivers

Enabled by growing robustness of open source compiler ecosystem
Layered OpenCL over Vulkan

- Clspv - Google’s open source OpenCL kernel to Vulkan SPIR-V compiler
  - Tracks top-of-tree LLVM and Clang, not a fork
- Clvk - prototype open source OpenCL to Vulkan run-time API translator
- Used for shipping production apps and engines on Android
  - Adobe Premiere Rush video editor - 200K lines of OpenCL C kernel code
  - Butterfly Network iQ Ultrasound on Android
  - Experimenting with Xiaomi MACE inferencing engine

[Diagram of Clspv and Clvk integration with OpenCL and Vulkan]
Layered OpenCL over DX12

- OpenCLOn12 - Microsoft and COLLABORA leveraging Clang/LLVM and MESA
  - OpenCL 1.2 over DX12 is in development
  - Also OpenGLOn12 - OpenGL 3.3 over DX12
Layered OpenCL over Metal??

Interest in OpenCL over Metal?
Khronos to host/coordinate open source project?

Need OpenCL Layered Profile?
Enables multiple layered subsets to be queryable and all present functionality to be tested for conformance

OpenCL over Vulkan
- OpenCL C or C++ for OpenCL Kernel Sources
- Clang+Clspv Compiler
- OpenCL Application Host Code
- Vulkan Runtime

OpenCL over Metal
- OpenCL C or C++ for OpenCL Kernel Sources
- Clang+Clspv Compiler
- Run-time conversion of API calls. Easier to shim OpenCL to Metal than Vulkan
- SPIRV-Cross Convert SPIR-V kernels to Metal Shaders
- Need SPIRV-Cross expansion to handle OpenCL dialect of SPIR-V
- Vulkan dialect SPIR-V from clspv IS ALREADY HANDLED

OR OpenCL C to Metal Source Translation?
Developers - Please Give Us Feedback!

- Is the set of optional features sliced too finely, or too coarsely?
  - Are they easy to understand?

- Which optional features do you expect to use in your application or library?
  - Usage data drives which optional features should be made mandatory in future

- What new features do you most need?
  - We will use extensions to prove new functionality before adding to core specification
  - What extensions would you like to see in the second half of 2020?

OpenCL Working Group has maximized information in
Khronos public GitHub to accelerate finalization

Provisional OpenCL 3.0
Specification sources released on GitHub
https://www.khronos.org/registry/OpenCL/

OpenCL 3.0 Conformance Tests WIP
sources released on GitHub
https://github.com/KhronosGroup/OpenCL-CTS

Spec feedback and
pull requests welcome on GitHub
https://github.com/KhronosGroup/OpenCL-Docs/issues

Tests feedback and
pull requests welcome on GitHub
https://github.com/KhronosGroup/OpenCL-CTS/issues

Vendor OpenCL 3.0
Implementations in flight

Urgency to Finalize and Ship
Finalized OpenCL 3.0 Specifications
Completed Conformance Tests
Multiple Shipping Conformant Implementations
Get Involved!

• OpenCL 3.0 increases deployment flexibility and sets the stage for raising the bar on pervasively available functionality
  - [https://www.khronos.org/registry/OpenCL/](https://www.khronos.org/registry/OpenCL/)

• Please provide specification feedback ASAP on GitHub for OpenCL 3.0 finalization!
  - [https://github.com/KhronosGroup/OpenCL-Docs/issues](https://github.com/KhronosGroup/OpenCL-Docs/issues)

• We want to know what you need next from OpenCL on the Khronos Forums!
  - [https://community.khronos.org/c/opencl](https://community.khronos.org/c/opencl)

• Engage with Khronos and help OpenCL evolve
  - Join as a Khronos member for a voice and a vote in any of these standards
  - Or request an invite to the OpenCL Advisory Panel
  - [https://www.khronos.org/members/](https://www.khronos.org/members/)

• Neil Trevett
  - ntrevett@nvidia.com
  - @neilt3d

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