Khronos Standards: Powering the Future of Embedded Vision

Neil Trevett, Khronos President
NVIDIA VP Developer Ecosystems
Khronos connects software to silicon

Open, royalty-free interoperability standards to harness the power of GPU, multiprocessor and XR hardware

3D graphics, augmented and virtual reality, parallel programming, inferencing and vision acceleration

Non-profit, member-driven standards organization, open to any company

Well-defined multi-company governance and IP Framework

Founded in 2000

>150 Members ~ 40% US, 30% Europe, 30% Asia

© 2021 The Khronos Group Inc.
Khronos Active Initiatives

3D Graphics
Desktop, Mobile and Web

3D Assets
Authoring and Delivery

Portable XR
Augmented and Virtual Reality

Parallel Computation
Vision, Inferencing, Machine Learning

Safety Critical APIs

© 2021 The Khronos Group Inc.
Khronos Compute Acceleration Standards

Higher-level Languages and APIs
Streamlined development and performance portability

Single source C++ programming with compute acceleration

Graph-based vision and inferencing acceleration

Intermediate Representation (IR) supporting parallel execution and graphics

Heterogeneous compute acceleration

Lower-level Languages and APIs
Direct Hardware Control

GPU rendering + compute acceleration

Increasing industry interest in parallel compute acceleration to combat the ‘End of Moore’s Law’

CPU
GPU
FPGA
DSP
AI/Tensor HW
Custom Hardware

SYCL and SPIR were originally OpenCL projects

© 2021 The Khronos Group Inc.
Programming and Runtime Framework for Application Acceleration
Offload compute-intensive kernels onto parallel heterogeneous processors
CPUs, GPUs, DSPs, FPGAs, Tensor Processors
OpenCL C or C++ kernel languages

Platform Layer API
Query, select and initialize compute devices

Runtime API
Build and execute kernels programs on multiple devices

Explicit Application Control
Which programs execute on what device
Where data is stored in memories in the system
When programs are run, and what operations are dependent on earlier operations

Complements GPU-only APIs
Simpler programming model
Relatively lightweight run-time
More language flexibility, e.g., pointers
Rigorously defined numeric precision
OpenCL Open-Source Project Momentum

# OpenCL-based GitHub Repos

Tripling in the last four years
OpenCL is Widely Deployed and Used

The industry’s most pervasive, cross-vendor, open standard for low-level heterogeneous parallel programming

Accelerated Implementations

## ML Compiler Steps

1. **Import Trained Network Description**
   - TensorFlow Graph, PyTorch, ONNX

2. **Graph-level optimizations**
   - e.g., node fusion, node lowering and memory tiling

3. ** Decompose to primitive instructions and emit programs for accelerated run-times**
   - TensorFlow Graph, PyTorch, ONNX

### Consistent Steps

<table>
<thead>
<tr>
<th>Import Formats</th>
<th>Front-end / IR</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe, Keras, MXNet, ONNX</td>
<td>NNVM / Relay IR</td>
<td>OpenCL, LLVM, CUDA, Metal</td>
</tr>
<tr>
<td>TensorFlow Graph, MXNet, PaddlePaddle, Keras, ONNX</td>
<td>nGraph / Stripe IR</td>
<td>OpenCL, LLVM, CUDA</td>
</tr>
<tr>
<td>PyTorch, ONNX</td>
<td>Glow Core / Glow IR</td>
<td>OpenCL, LLVM</td>
</tr>
<tr>
<td>TensorFlow Graph, PyTorch, ONNX</td>
<td>XLA HLO</td>
<td>OpenCL, LLVM, TPU IR, XLA IR TensorFlow Lite / NNAPI (inc. HW accel)</td>
</tr>
</tbody>
</table>

### Embedded NN Compilers

- CEVA Deep Neural Network (CDNN)
- Cadence Xtensa Neural Network Compiler (XNNC)

---

**Fast progress but still area of intense research**

If compiler optimizations are effective - hardware accelerator APIs can stay ‘simple’ and won’t need complex metacommands (e.g., combined primitive commands like DirectML)
OpenCL 3.0

Increased Ecosystem Flexibility
All functionality beyond OpenCL 1.2 queryable plus macros for optional OpenCL C language features
New extensions that become widely adopted will be integrated into new OpenCL core specifications

OpenCL C++ for OpenCL
Open-source C++ for OpenCL front end compiler
combines OpenCL C and C++17 replacing
OpenCL C++ language specification

Unified Specification
All versions of OpenCL in one specification for easier maintenance, evolution and accessibility
Source on Khronos GitHub for community feedback, functionality requests and bug fixes

Moving Applications to OpenCL 3.0
OpenCL 1.2 applications - no change
OpenCL 2.X applications - no code changes if all used functionality is present
Queries recommended for future portability

C++ for OpenCL
Supported by Clang and uses the LLVM compiler infrastructure
OpenCL C code is valid and fully compatible
Supports most C++17 features
Generates SPIR-V kernels
OpenCL 3.0 Adoption

OpenCL 3.0 Adopters

- ARM
- Codeplay
- KALRAY
- Microsoft
- QNX
- Qualcomm
- VeriSilicon
- NVIDIA
- Imagination

Product Conformance Status
https://www.khronos.org/conformance/adopters/conformant-products/opencl

OpenCL 3.0 Adopters Already Shipping Conformant Implementations
**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 3D-3D Copy Transaction**

Reshaping possible

\[ V_{\text{global}} = V_{\text{local}} \]

---

**Async Fence controls order of dependent transactions**

Async 3D-3D Copy Transaction

Async Fence built-in controls execution order of dependent transactions:

- `async_copy_1`
- `async_copy_2`
- `async_fence`
- `async_copy_3`

All transactions prior to `async_fence` must complete before any new transaction starts, without a synchronous wait

---

The first of significant upcoming advances in OpenCL to enhance support for embedded processors

© 2021 The Khronos Group Inc.
Layered OpenCL Implementations

**clspv + clvk**
- clspv - Google’s open-source OpenCL kernel to Vulkan SPIR-V compiler
- Clvk - prototype open-source OpenCL to Vulkan run-time API translator
- Used by shipping apps and engines on Android e.g., Adobe Premiere Rush video editor - 200K lines of OpenCL C kernel code

**OpenCLon12**
- Microsoft and COLLABORA
- GPU-accelerated OpenCL on any DX12 PC and Cloud instance (x86 or Arm)
- Leverages Clang/LLVM AND MESA

OpenCL over DX12
https://github.com/microsoft/OpenCLon12

OpenCL over Vulkan
https://github.com/google/clspv
https://github.com/kpet/clvk

OpenCL over OpenGL
- OpenGL 3.3 over DX12 is already conformant
SPIR-V enables a rich ecosystem of languages and compilers to target low-level APIs such as Vulkan and OpenCL, including deployment flexibility: e.g., running OpenCL kernels on Vulkan.
SYCL Single Source C++ Parallel Programming

Complex ML frameworks can be directly compiled and accelerated

C++ templates and lambda functions separate host & accelerated device code

SYCL is ideal for accelerating larger C++-based engines and applications with performance portability

C++ Kernel Fusion can give better performance on complex apps and libs than hand-coding

Accelerated code passed into device OpenCL compilers
Expressiveness and simplicity for heterogeneous programming in modern C++
Closer alignment and integration with ISO C++ to simplify porting of standard C++ applications
Improved programmability, smaller code size, faster performance
Based on C++17, backwards compatible with SYCL 1.2.1
Backend acceleration API independent

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

Significant SYCL adoption in Embedded, Desktop and HPC Markets
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.

Multiple Backends in Development
SYCL beginning to be supported on multiple low-level APIs in addition to OpenCL e.g., ROCm and CUDA
For more information: http://sycl.tech
The Origin of OpenVX

**Engines and Applications**

**Vulkan**

3D Graphics API Driver

**GPU**

**Driver Model**

An open API standard enables multiple silicon vendors to ship drivers with their silicon

Silicon vendors can aggressively optimize drivers for their own silicon architecture

OpenVX is the industry's only API standard enabling portable access to vendor-optimized vision drivers

**Engines and Applications**

**OpenVX**

Vision API Driver

**GPU**

**DSP**

**HW**

**CPU**

**High-level Abstraction**

3D graphics is always accelerated by a GPU - so a low-level GPU-centric API still provides cross-vendor portability

Vision processing can be accelerated by a wide variety of hardware architectures

OpenVX needs a higher-level graph abstraction to enable optimized cross-vendor drivers

**Vision Processing Graph**
OpenVX Cross-Vendor Vision and Inferencing

High-level graph-based abstraction for portable, efficient vision processing
- Optimized OpenVX drivers created, optimized and shipped by processor vendors
- Implementable on almost any hardware or processor with performance portability
- Graph can contain vision processing and NN nodes for global optimization
- Run-time graph execution need very little host CPU interaction

OpenVX Graph

- Native Camera Control
- Vision Node
- CNN Nodes
- Vision Node
- Downstream Application Processing

NNEF Import converts a trained Neural Network into OpenVX Graph
- Layers are represented as OpenVX nodes

Open Source Convertors
- https://github.com/KhronosGroup/NNEF-Tools

Stable Specification

Vendors optimize and ship drivers for their platform
- Full list of conformant OpenVX implementations here:
  https://www.khronos.org/conformance/adopters/conformant-products/openvx

© 2021 The Khronos Group Inc.
OpenVX Efficiency through Graphs..

**Graph Scheduling**

- Split graph execution across the whole system: CPU / GPU / dedicated HW
- Faster execution or lower power consumption

**Memory Management**

- Reuse pre-allocated memory for multiple intermediate data
- Less allocation overhead, more memory for other applications

**Kernel Fusion**

- Replace a sub-graph with a single faster node
- Better memory locality, less kernel launch overhead

**Data Tiling**

- Execute a sub-graph at tile granularity instead of image granularity
- Better use of data cache and local memory

---

**Performance comparable to hand-optimized, non-portable code**

- Real, complex applications on real-world hardware
- Much lower development effort and higher portability than hand-optimized code
OpenVX 1.3 and Extensibility

OpenVX 1.3 core specification defines market-targeted feature sets
- Baseline Graph Infrastructure (enables other Feature Sets)
- Default Vision Functions
- Enhanced Vision Functions
- Neural Network Inferencing (including tensor objects)
- NNEF Kernel import (including tensor objects)
- Binary Images
- Safety Critical (reduced features and graph import for easier safety certification)

OpenVX is Extensible
Fully accelerated custom nodes can be integrated into the OpenVX graph with OpenCL interop

OpenVX data objects

OpenCL Command Queue

Application

cl_mem buffers

Copy or export cl_mem buffers into OpenVX data objects
Map or copy OpenVX data objects into cl_mem buffers

Fully asynchronous host-device operations during data exchange

OpenVX/OpenCL Interop
Open Source OpenVX & Samples

Fully Conformant
Open Source OpenVX 1.3
for Raspberry Pi
Raspberry Pi 3 and 4 Model B with Raspbian OS
Memory access optimization via tiling/chaining
Highly optimized kernels on multimedia instruction set
Automatic parallelization for multicore CPUs and GPUs
Automatic merging of common kernel sequences

Open Source OpenVX Tutorial and Code Samples
https://github.com/rgiduthuri/openvx_tutorial
https://github.com/KhronosGroup/openvx-samples

"Raspberry Pi is excited to bring the Khronos OpenVX 1.3 API to our line of single-board computers. Many of the most exciting commercial and hobbyist applications of our products involve computer vision, and we hope that the availability of OpenVX will help lower barriers to entry for newcomers to the field."

Eben Upton
Chief Executive Raspberry Pi Trading

Check out the OpenVX 1.3 Session here at Embedded Vision Summit for more details!
APIs for Embedded Compute

Networks trained on high-end desktop and cloud systems

Applications link to compiled inferencing code or call vision/inferencing API

Diverse Embedded Hardware
- Multi-core CPUs, GPUs
- DSPs, FPGAs, Tensor Cores
* Vulkan only runs on GPUs

Open industry standards, enable flexible integration and deployment of multiple acceleration technologies

Neural Network Training

Compiled Code

Vision / Inferencing Engine

C++ Application Code

Hardware Acceleration APIs

Sensor Data

Compilation

Ingestion

© 2021 The Khronos Group Inc.
Need for Embedded Camera API Standards

Increasing Sensor Diversity
Including camera arrays and depth sensors such as Lidar

Multiple Sensors Per System
Synchronization and coordination become essential

Cost and time to integrate and utilize sensors in embedded systems is a major constraint on innovation and efficiency in the embedded vision market

Sophisticated Sensor Processing
Including inferencing. Sensor streams need to be efficiently generated and fed into acceleration APIs and processors

Proprietary Interfaces
Vendor-specific APIs to control cameras, sensors and close-to-sensor ISPs
Benefits of Embedded Camera API Standard

An effective open, cross-vendor open standard for camera, sensor and ISP control could provide multiple benefits

- Cross-vendor portability of camera/sensor code for easier system integration of new sensors
- Preservation of application code across multiple generations of cameras and sensors
- Sophisticated control over sensor stream generation increases effectiveness of downstream accelerated processing

Development of Camera and sensor APIs may also generate new requirements for downstream vision and inferencing acceleration APIs

© 2021 The Khronos Group Inc.
Embedded Camera API Exploratory Group

Any company is welcome to join

No cost or IP Licensing obligations

Project NDA to cover Exploratory Group Discussions

Online discussion forum and weekly Zoom calls, probably for a few months

Discuss industry requirements for open, royalty-free camera API(s)

No detailed design activity to protect participants IP

Explore if consensus can be built around an agreed Scope of Work document

Discuss what standardization activities can best execute actions in the Scope of Work

Scope of Work Document

Agreed SOW document released from NDA and made public

Proven Khronos Process to ensuring industry requirements are fully understood before starting standardization initiatives

Agreement with standardization bodies and/or open source projects on initiative(s) to execute the SOW under proven processes and IP Frameworks

Join and get involved!

https://www.khronos.org/embedded-camera/#getinvolved

© 2021 The Khronos Group Inc.
Khronos membership is open to any company
Influence the design and direction of key open standards that will drive your business
Accelerate time-to-market with early access to specification drafts
Provide industry thought leadership and gain insights into industry trends and directions
Benefit from Adopter discounts

www.khronos.org