OpenCL Overview
Heterogeneous Parallel Computation

Neil Trevett
Khronos President and OpenCL Chair
VP Developer Ecosystems, NVIDIA
nptrevett@nvidia.com |@neilt3d
January 2021
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

Lower-level APIs
Direct Hardware Control

- GPU rendering + compute acceleration
- Intermediate Representation (IR) supporting parallel execution and graphics
- Heterogeneous compute acceleration

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

SYCL and SPIR were originally OpenCL Subgroups
OpenCL - Low-level Parallel Programming

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 is Widely Deployed and Used

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

Accelerated Implementations

OpenCL Open-Source Ecosystem Momentum

April 2020, 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.

October 2020, SARS-CoV-2 Simulations Go Exascale to Capture Spike Opening with over a million citizen scientists banding together through the Folding@home distributed computing project to create the first Exascale computer and simulate an unprecedented 0.1 seconds of the viral proteome.
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](https://cpp.opencl.net) 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 C:
- kernels,
- address spaces,
- special types,
...

Most of C++17:
- inheritance,
- templates,
- type deduction,
...
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
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
Roadmap: External Memory Sharing

- 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
Google Ports TensorFlow Lite to OpenCL

Even Faster Mobile GPU Inference with OpenCL

August 17, 2020

Posted by Jiyoun Lee and Roman Sarokin, Software Engineers

While the TensorFlow Lite (TFLite) GPU team continuously improves the existing OpenGL-based mobile GPU inference engine, we also keep investigating other technologies. One of those experiments turned out quite successful, and we are excited to announce the official launch of OpenCL-based mobile GPU inference engine for Android, which offers up to ~2x speedup over our existing OpenGL backend, on reasonably sized neural networks that have enough workloads for the GPU.

![OpenCL Logo](Image)

OpenCL providing ~2x inferencing speedup over OpenGL ES acceleration

TensorFlow Lite uses OpenGL ES as a backup if OpenCL not available ...

...but most mobile GPU vendors provide OpenCL drivers - even if not exposed directly to Android developers

OpenCL is increasingly used as acceleration target for higher-level framework and compilers

Improvements over the OpenGL Backend

Historically, OpenGL is an API designed for rendering vector graphics. Compute shaders were added with OpenGL ES 3.1, but its backward compatible API design decisions were limiting us.

![Graph 1](Image)

![Graph 2](Image)

![Graph 3](Image)

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

© The Khronos® Group Inc. 2021 - Page 9
## ML Compiler 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</td>
</tr>
<tr>
<td>TensorFlow Graph, PyTorch, ONNX</td>
<td>XLA HLO, MLIR</td>
<td>LLVM, TPU IR, XLA IR TensorFlowLite / NNAPI (inc. HW accel)</td>
</tr>
</tbody>
</table>

### Consistent Steps
1. Import Trained Network Description
2. Apply graph-level optimizations e.g., node fusion, node lowering and memory tiling
3. Decompose to primitive instructions and emit programs for accelerated run-times

### 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)
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 C kernels on Vulkan.
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]

Clspv

https://github.com/google/clspv

Clvk

https://github.com/kpet/clvk
Layered OpenCL over DirectX12

- GPU-accelerated OpenCL on any system with DX12
  - PC (x86 or Arm) and Cloud
- OpenCLOn12 - Microsoft and COLLABORA leveraging Clang/LLVM and MESA
  - OpenCL 1.2 over DX12 is in development
  - Also, OpenGLOn12 - OpenGL 3.3 over DX12
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/)

- OpenCL specification feedback on GitHub
  - [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 Khronos standard
  - 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